Using Deep Learning Methods for Sentiment Extraction From Image-Encoded Text

Harpreet Singh

Supervisors:

Associate Prof. Na Helian

Dr. Yi Sun

Prof. Roderick Adams

School of Physics, Engineering and Computer Science University of Hertfordshire

Submitted to the University of Hertfordshire in partial fulfilment of the requirement of the degree of PhD

November 2025

Acknowledgement

I offer my deepest gratitude to Nirankar, for this beautiful Universe and for placing such wonderful souls in my life to help me thrive.

I am profoundly grateful to my family, whose support has been my foundation. To my father, S. Gurcharan Singh, thank you for your unwavering support and for making my education possible. To my mother, Sdn. Swarndeep Kour, thank you for your constant and abiding belief in me.

My sincere thanks also go to my brother, Soldier Manpreet Singh, and my sister-in-law, Amanjot Kour, for their steadfast support. I am deeply grateful to my sister, Samreen Kour, and brother-in-law, Simranjeet Singh, for their invaluable help during several critical moments. To my wife, Paramjot Kour, I am profoundly grateful. Your motivation and encouragement during the most difficult times were an indispensable source of strength.

I extend my sincere gratitude to my supervision team for their exceptional guidance. To Dr. Na Helian, thank you for not only inspiring me to pursue this PhD but also for your constant support throughout this journey. I am indebted to Dr. Yi Sun for consistently challenging me to achieve greater rigor in my methods. I must also thank Prof. Roderic Adam, who was instrumental in helping me see the "bigger picture" and meticulously improving my writing skills.

My special thanks go to Dr. Emil for being wonderful friend; your invaluable discussions helped me unravel complex problems and find a clear path forward. I cherish our discussion on philosophy, theology and technology and our journey so far.

I also want to thank Phil for his invaluable assistance in structuring my experiments and ideas. Dr Jaspreet for being a phone call away when I had doubts or need second opinions.

My appreciation extends to all the members of the Bio-computation team. I am especially grateful to Dr. Reinoud for generously reviewing my manuscripts. I also thank Dr. Yaqoob and Dr. Tayarani-Najaran for the many insightful conversations and guidance. To Dr. Sam, Shavika, Dr. Ming, Dr. Rebecca, Dr. Maria, Dr. Pinaki, Dr. Deepak, Shamim, and Eleonora, thank you for the lively conversations and supportive atmosphere that provided a welcome respite.

A special note of thanks is reserved for Dr. Adrain Ghita from LMS and Kuldeep Singh from the SPECS. You have been exceptional mentors, helping me in every aspect of my journey. I am grateful not only for your professional guidance and personal generosity but also for the stimulating discussions and debates on wider worldviews, always conducted with mutual respect.

I am also thankful to my students, who inspired me to delve into concepts with greater depth. My gratitude extends to every person who has been part of my life's journey; I have learned invaluable lessons from all, whether they offered kindness and support or challenged me with critical perspectives. Both positive and difficult encounters have shaped me and contributed, in their own way, to the resilience required to complete this work.

Declaration

I hereby declare that this dissertation is the result of my own independent research, carried out in accordance with the regulations and Code of Practice for Research Degree Programmes of the University. The work contained herein is original, except where explicitly indicated by specific reference to the work of others, and it has not been submitted in whole or in part for any other academic award or qualification at this or any other university. Any views expressed within this dissertation are solely those of the author.

Harpreet Singh
Nov 2025

Abstract

The exponential growth of user-generated content on social media platforms has necessitated robust, automated methods for sentiment analysis to understand public opinion, identify potentially harmful content such as hate speech, and support the enforcement of community standards. While traditional text processing has evolved significantly, the potential for leveraging advanced computer vision techniques to analyse textual data remains underexplored. To the author's knowledge, this is the first work to treat two-dimensional text representations seriously as images and to evaluate this idea extensively across multiple architectures, datasets and experimental conditions. This allows deep Convolutional Neural Networks (DCNNs) to analyse the spatial structure of text representations. The research bridges Natural Language Processing (NLP) and Computer Vision by developing and evaluating hybrid architectures that combine sequential text modelling with networks previously trained to recognise visual patterns.

To address this aim, the thesis follows a staged experimental programme. It first establishes whether sequential text representations can be transformed into discriminative images and classified by purpose-built vision architectures. It then enriches these images with contextualised language representations, evaluates transfer learning from networks previously trained to recognise visual patterns, and finally combines visual, lexical and topic-based signals in an ensemble framework.

The first stage proposes BLSTM-ResNet, which replaces the shallow convolutional layers of the BLSTM-2DCNN baseline with deeper residual blocks. On Sentiment140, performance improved with network depth before saturating at four residual blocks. The resulting BLSTM-ResNet-12 achieved 78.80% accuracy on the 80,000-sample subset, outperforming the baseline and validating the Textual Image representation.

The second stage incorporates contextualised embeddings from a fine-tuned BERT model. BERT-BLSTM-ResNet outperformed standalone BERT on Sentiment140 subsets below 40,000 samples, whereas the performance gap diminished above 200,000 samples. Experiments on the IMDB dataset further showed that longer token sequences improved contextualised Textual Image classification. Across the evaluated configurations, direct BERT-ResNet integration retained the accuracy benefits of the hybrid approach while avoiding the additional sequential-processing stage, thereby providing a more computationally efficient architecture that is better suited to GPU-parallel training.

Furthermore, the thesis addresses sustainability in machine learning by exploring the transferability of networks previously trained to recognise visual patterns. The research confirms that fine-tuning publicly available image classifiers (e.g., ResNet50, EfficientNet, ViT) on synthetic Textual Images is a resource-efficient alternative to training custom vision models from scratch. The study identifies EfficientNet-B5 as achieving the highest individual accuracy (93.37% on the IMDB dataset) under a three-epoch, hardware-constrained setting. Nearest Neighbour interpolation was also identified as the preferred image-resizing method because of its computational efficiency without loss of accuracy.

Finally, the research demonstrates the complementarity of visual, lexical and topic signals. An ensemble strategy combining the Textual Image Classifier (BERT-ResNet) with a lexically enhanced BERT model (BERT-SWC) and a traditional topic-modelling (LSA, LDA) classifier achieved a best accuracy of 94.32% on the IMDB dataset using an XGBoost meta-learner. This thesis successfully establishes Textual Image Classification as a robust, high-performance, and sustainable methodology for sentiment analysis, providing a comprehensive framework for future multimodal text analysis research.

Abbreviations

1D: One Dimensional

2D: Two Dimensional

ANN: Artificial Neural Network

ASCII: American Standard Code for Information Interchange

BERT: Bidirectional Encoder Representations from Transformers

BGRU: Bidirectional Gated Recurrent Unit

BLSTM: Bidirectional Long Short-Term Memory

BERT-SWC: BERT model with Auxiliary SentiWordNet task

BN: Batch Normalisation

BOW: Bag of Words

CBOW: Continuous Bag of Words

CNN: Convolutional Neural Networks

CV: Cross Validation

DCNN: Deep Convolutional Neural Networks

DNN: Deep Neural Networks

GloVe: Global Vectors

GRU: Gated Recurrent Unit

IJCNN: International Joint Conference for Neural Network

LR: Linear Regression

LLM: Large Language Model

LSTM: Long Short-Term Memory

MLP: Multilayer Perceptron

NLP: Natural Language processing

NB: Naïve Bayes

ReLU: Rectified Linear Unit

ResNet: Residual Network

RNN: Recurrent Neural Networks

SA: Sentiment Analysis

SVM: Support Vector Machine

ViT: Vision Transformer

Contents

Abbreviations **5**

1 Introduction **20**

1.1 Introduction . . . 20
1.2 Sentiment Analysis . . . 21
1.3 Levels of Sentiment Analysis . . . 23
1.3.1 Document level . . . 23
1.3.2 Sentence level . . . 23
1.3.3 Phrase level . . . 24
1.3.4 Aspect level . . . 24
1.4 Machine Learning in Sentiment analysis . . . 24
1.4.1 Machine Learning and Sustainability . . . 25
1.5 Motivation for the research . . . 26
1.6 Aim of the research . . . 26
1.7 Research questions . . . 27
1.8 Contributions to knowledge . . . 27
1.9 Thesis organisation . . . 28

2 Background and Literature Review **29**

2.1 Vectorisation granularity . . . 30
2.2 Data Pre-Processing . . . 31
2.2.1 Data Cleaning and tokenisation . . . 31
2.2.2 Text normalisation . . . 31
2.2.3 Text vectorisation and feature extraction . . . 32
2.2.3.1 Bag of words (BOW): . . . 32
2.2.3.2 N-gram: . . . 32
2.2.3.3 Continuous Bag of words . . . 33
2.2.3.4 Word Embeddings . . . 33
2.2.4 GloVe . . . 33
2.2.5 Two-dimensional (2D) Convolution Neural Networks . . . 34
2.2.6 Forming images from text . . . 34
2.3 Contextualized Word representation . . . 36
2.3.1 Attention mechanism . . . 37
2.3.2 Transformer based text models . . . 38

2.3.3 Attention in Computer Vision . . . 38
2.4 Systematic literature review of Transformer adoption in documents level sentiment analysis. . . . 38
2.4.1 Lexical resources . . . 45
2.4.2 Why BERT? . . . 45
2.5 Sustainability . . . 45
2.6 Literature Review in Relation to the Research Questions . . . 46
2.6.1 RQ1 and Chapter 4: Deep 2D-CNNs for Textual Images . . . 46
2.6.2 RQ2 and Chapter 5: Contextualised Textual Images . . . 46
2.6.3 RQ3 and Chapter 6: Pre-trained Vision Backbones and Efficiency . . . 46
2.6.4 RQ4 and Chapter 6: Complementary Sentiment Models and Ensembling . . . 46
2.7 Conclusion . . . 47

3 Methodology **48**
3.1 Introduction . . . 48
3.2 Hardware and Software setup . . . 48
3.2.1 Hardware . . . 48
3.2.2 Software . . . 49
3.3 Datasets . . . 49
3.3.1 Sentiment140 . . . 49
3.3.2 IMDB Review . . . 49
3.4 Machine learning pipeline adopted in this work . . . 50
3.4.1 Pipeline 1 : Static Textual Images and a custom ResNet (RQ1) . . . 50
3.4.2 Pipeline 2 : Contextualised Textual Images with a BLSTM (RQ2) . . . 50
3.4.3 Pipeline 3 : Direct contextualised Textual Images without a BLSTM (RQ2) . . . 52
3.4.4 Pipeline 4 : pre-trained vision backbones (RQ3) . . . 52
3.4.5 Pipeline 5 : probability-level fusion of complementary representations (RQ4) . . . 52
3.5 Text preprocessing . . . 55
3.5.1 String cleaning . . . 55
3.5.1.1 URL . . . 55
3.5.1.2 Username . . . 56
3.5.1.3 Hashtag . . . 56
3.5.2 Tokenisation . . . 56
3.5.3 Normalisation . . . 56
3.5.3.1 Stop words removal . . . 57
3.5.3.2 Stemming . . . 57
3.5.3.3 Lemmatisation . . . 57
3.5.4 Lexical Resources . . . 58
3.5.4.1 Part-of-Speech (POS) Tagging . . . 58
3.5.4.2 SentiWordNet . . . 59
3.5.4.3 Word sentiment score (sw_score) . . . 60
3.6 Feature Extraction . . . 61
3.6.1 BOW textual encoding . . . 61

3.6.1.1 BOW Tokenisation . . . 61
3.6.1.2 Count Vectorisation . . . 61
3.6.1.3 TF-IDF vectorisation . . . 62
3.6.2 Latent Semantic Analysis (LSA) . . . 63
3.6.3 Latent Dirichlet Analysis (LDA) . . . 63
3.6.4 Traditional Machine learning for BOW classification . . . 64
3.6.4.1 Naïve Bayes (NB) . . . 64
3.6.4.2 Logistic Regression (LR) . . . 64
3.6.4.3 Support vector Machine (SVM) . . . 65
3.6.4.4 XGBoost (XGB) . . . 65
3.6.5 GloVe Embeddings . . . 65
3.6.5.1 Tokenisation (GloVe) . . . 65
3.7 Neural Network architecture and variants adopted in this work . . . 66
3.7.1 Feed forward Neural network (FFN) . . . 66
3.7.2 Activation function . . . 67
3.7.3 Optimisation . . . 68
3.7.4 Stochastic gradient descent (SGD): . . . 68
3.7.5 Adaptive Moment (Adam) estimation: . . . 69
3.7.6 Convolutional neural network (CNN) . . . 69
3.7.6.1 Residual network (ResNet) . . . 70
3.7.6.2 DenseNet . . . 71
3.7.6.3 EfficientNet . . . 71
3.7.6.4 Vision Transformer (ViT) . . . 71
3.7.6.5 Swin Transformer . . . 72
3.7.7 Recurrent Neural Network (RNN) . . . 73
3.7.7.1 LSTM: . . . 73
3.7.7.2 BLSTM . . . 75
3.8 Textual Images . . . 75
3.8.1 Image data . . . 75
3.8.2 Image classification . . . 76
3.8.3 Images from vectorised text and classification . . . 76
3.8.4 BLSTM as Textual Image generator . . . 76
3.8.5 BLSTM-2DCNN . . . 77
3.8.6 Our Proposed BLSTM-ResNet for Textual Image Classification . . . 77
3.8.6.1 BLSTM-ResNet model implementation . . . 77
3.8.7 The contextualised Textual Images generation using BERT . . . 80
3.8.7.1 Transformers . . . 80
3.8.7.2 Attention mechanism . . . 83
3.8.7.3 Scaled dot product attention in Transformers . . . 84
3.8.7.4 Multi-head Attention in Transformer . . . 84
3.8.7.5 Contextualised embeddings . . . 84

3.8.8 BERT . . . 85
3.8.8.1 WordPiece Tokenisation . . . 85
3.8.9 BERT as the Contextualised Textual Images generator . . . 87
3.8.10 Interpolation . . . 87
3.9 Neural network sustainable training strategies . . . 89
3.9.1 Early stopping . . . 89
3.9.2 Fine-tuning pre-trained models . . . 89
3.9.3 Ensembling . . . 89

4 Sentiment Analysis using BLSTM-ResNet on Textual Images 90
4.1 Introduction . . . 90
4.2 Our Proposed Textual Image classification method . . . 91
4.2.1 BLSTM-ResNet Variant . . . 91
4.3 Experiment Setup . . . 93
4.3.1 Data Preparation . . . 93
4.3.2 Learning Rate Strategies for BLSTM-ResNet . . . 93
4.3.3 Training Strategy . . . 93
4.4 Results . . . 94
4.4.1 Experiment 1: Impact of Network Depth on model performance . . . 94
4.4.2 Experiment 2: Impact of Dataset size on model Performance . . . 96
4.4.3 Experiment 3: Impact of Residual block construction on model Performance . . . 99
4.5 Contributions and Summary . . . 101

5 Using BERT to Generate Contextualised Textual Images for Sentiment Analysis 103
5.1 Introduction . . . 103
5.2 Experiment 1: With Sentiment140 dataset . . . 103
5.2.1 Training Setup . . . 105
5.2.2 Result and discussion . . . 105
5.3 Experiment 2: with IMDB dataset . . . 108
5.3.1 Experimental Setup: . . . 108
5.3.1.1 Models: . . . 109
5.3.2 Training Strategy . . . 110
5.3.3 Results: . . . 110
5.3.4 Statistical Analysis of BERT vs BERT extended Models Performance . . . 113
5.4 Summary . . . 116

6 Using a pre-trained image classifier for Textual Image 118
6.1 Introduction . . . 118
6.2 Fine-tuning a pre-trained image classifier on Textual Images . . . 119
6.3 Traditional and contemporary sentiment analysis techniques complementing each other . . . 119
6.4 Experimental Setup, Experiments and Results . . . 119
6.4.1 Experiment 1 . . . 120
6.4.2 Experiment 2 . . . 123

6.4.3 Experiment 3 . . . 124
6.4.4 Experiment 4 . . . 128
6.4.4.1 Comparison with published IMDB results . . . 132
6.5 Summary . . . 134

7 Conclusion **136**
7.1 Summary of the Research . . . 136
7.2 Synthesis of Findings Addressing Research Questions . . . 136
7.2.1 RQ1: Textual Images and Two-Dimensional Neural Pattern Recognition . . . 136
7.2.2 RQ2: Impact of Contextualised Embeddings . . . 137
7.2.3 RQ3: Pre-trained CNN and Transformer-Based Image Classifiers . . . 138
7.2.4 RQ4: Complementary Predictive Information and Ensembling . . . 138
7.3 Discussion in Relation to Previous Work . . . 139
7.3.1 From a precursor to a staged research programme . . . 139
7.3.2 Relationship to hybrid sequence-processing architectures . . . 139
7.3.3 Relationship to ensemble sentiment analysis . . . 139
7.3.4 Thesis-level contribution and its boundaries . . . 140
7.4 Limitations of the Experimental Work . . . 140
7.5 Prospects and Direction . . . 140
7.5.1 Multimodal sentiment analysis . . . 140
7.5.2 Representation design and long documents . . . 141
7.5.3 Architectures matched to Textual Image geometry . . . 141
7.5.4 Interpretability, robustness and responsible use . . . 141
7.6 Conclusion . . . 141

8 References **142**

9 Appendix A **151**
9.1 Cross-Validation . . . 151
9.1.1 Training Strategy . . . 153
9.1.2 Python development . . . 153
9.2 How RNN process information . . . 153

10 Appendix B **156**
10.1 Individual runs results for BLSTM-ResNet . . . 156
10.1.1 Average Accuracies (%) and Standard Deviations for BLSTM-ResNet variants at Various Dataset size for sentiment140 dataset. . . . 156
10.1.2 Weighted F1 Scores (%) and Standard Deviations for BLSTM-ResNet variants at Various Dataset size for sentiment140 dataset. . . . 156
10.1.3 Individual experiment run results for BLSTM-ResNet variant carried out in Section 4 on Sentiment140 dataset. . . . 157
10.1.4 Investigate the impact of the learning rate decay on BLSTM-ResNet-36. . . . 165

11 Appendix C **167**

11.1 Sentiment140 five fold cross validation runs on BERT and BERT-BLSTM-ReNet-18 . . . 167
11.1.1 5-fold cross validation result for BERT for Sentiment140 subsets . . . 167
11.1.2 5-fold cross validation result for BERT-ResNet18 for Sentiment140 subsets . . . 168
11.2 IMDB experiment result on various BLSTM and ResNet configurations extending the BERT output 168
11.2.1 BERT Model . . . 168
11.2.2 BERT-BLSTM . . . 168
11.2.3 BERT-BLSTM||ResNet50 . . . 168
11.2.4 BERT-BLSTM-RESNET 50 . . . 169
11.2.5 BERT-RESNET 50 . . . 169
11.2.6 Experiment result for 10 runs with average performance and standard deviations. . . . 169

12 Appendix D **171**

12.1 Sentiment140 performance on BERT, BERT-ResNet and BERT-VIT. . . . 171
12.1.1 BERT . . . 171
12.1.2 BERT -ResNet . . . 172
12.1.3 BERT-ViT . . . 172
12.1.4 Confusion Matrix & Agreement Analysis by Dataset Size (BERT vs BERT-ResNet vs BERT-ViT) . . . 172
12.2 IMDB Dataset (CNN vs Transformer) Models Results . . . 179
12.2.1 Kruskal-Wallis H Test with Dunn's Post-hoc Test on model performance . . . 180
12.3 Critical Value Calculation for difference in performance for different vision backbones (Dunn's Post-hoc Test) . . . 180
12.3.1 Significance test based on interpolation method . . . 181
12.3.1.1 One-Way ANOVA Test for Interpolation Methods . . . 182
12.3.1.2 Assumption Checks . . . 182
12.3.1.3 ANOVA Results . . . 182
12.3.2 Friedman test followed by Nemenyi post-hoc test . . . 183
12.3.3 Experiment Report . . . 184
12.4 Pre-trained Efficientnet accuracy on Contextualised Textual Images. . . . 184
12.4.1 Model: Efficientnet_b0 . . . 184
12.4.2 Model: Efficientnet_b1 . . . 185
12.4.3 Model: Efficientnet_b2 . . . 185
12.4.4 Model: Efficientnet_b3 . . . 185
12.4.5 Model: Efficientnet_b4 . . . 185
12.4.6 Model: Efficientnet_b5 . . . 186
12.4.7 Model: Efficientnet_b6 . . . 186
12.4.8 Model: Efficientnet_b7 . . . 186
12.4.9 Comparative plots of average performance with standard deviations of BERT-Efficient variants organised in various views. . . . 187
12.4.10 Comparative plots of best performance of BERT-Efficient variants organised in various views. 188
12.5 Choosing the Best LSA+LDA components pared with best machine learning model parameters. . . 189
12.5.1 Exponential increment of LSA and LDA components across different model best parameters 190
12.5.2 Logistic Regression . . . 192

12.5.3 Multi-Layer Perceptron (MLP) 193
12.5.4 Naïve Bayes 194
12.5.5 Support Vector Machine (SVM) 195
12.5.6 XGBoost 196
12.6 BERT-SWC performance vs Alpha. 196

List of Figures

1.1 Data categories (Structured, Semi-structured and Unstructured) 20

1.2 Classification of Text in object and subjective text 22

1.3 Levels of Sentiment Analysis [13] 23

2.1 Text classification pipeline 30

2.2 Depiction of a 7-word sentence with embedding size 3, processed by a BLSTM layer with 5 hidden units, followed by a 2x2 convolution, max-pooling operation and Flatten layer (source [27]). The five outputs in the first layer were chosen as an example and are not correlated with the five BLSTM hidden units. 35

2.3 Example of nonvolitional (left) and volitional (right) cues in an environment (source [89]). 37

3.1 Different pipelines used in this research to represent the text as an Image for text sentiment analysis 51

3.2 machine learning pipeline with Ensemble model 53

3.3 Feed Forward neural network with 2 hidden layer 66

3.4 Comparison of SRN-Simple Recurrent Unit and LSTM- Long Short-Term Memory unit. (source (Greff et al. 2016)). 74

3.5 LSTM internal operations 74

3.6 A simple Bidirectional LSTM cell as a hidden layer, where O and – represent the open and closed states of an individual LSTM cell, respectively. 75

3.7 A BLSTM-ResNet example, where the Input document has 5 timesteps (document length of 5) because there are 5 elements in the document with each word having a 7-dimensional word Embedding. It is processed using a BLSTM with 4 hidden units (not shown) where 'FXt' and 'BXt' are 1x4(hidden units) outputs of forward and backward LSTM respectively at a given timestep 't'. Four hidden units mean 1x4 output vector at the time 't' from both forward and backward LSTM which are added (+) elementwise to get a single 1x4 BLSTM (HXt) output. The 5(time steps) x4 (hidden units) output matrix obtained from BLSTM is reshaped to 5x4x1 to form a greyscale image equivalent matrix (1 denotes a single channel value for the image). The Textual Image is classified using ResNet-inspired 2D-CNN blocks followed by Average Pooling, Flatten and Output layers. 78

3.8 Pseudo code to implement BLSTM-ResNet architecture 78

3.9 Example datapoints and labels 79

3.10 Demo BLSTM-ResNet Model summary with one residual layer and one skip connection. Tensor shape at each layer is shown with some description of each layer. 81

3.11 Sample Textual Images generated after BLSTM in BLSTM- ResNet network 82

3.12 Transformer encoder(left) and decoder (right) architecture (Inspired by [91]). 82

3.13 Query, Keys and Value in artificial attention mechanism (Inspired by [89]). 83

3.14 Scaled dot product attention (Inspired by [91]) 84

3.15 multi-head attention with 'h' attention heads (Inspired by [91]) 85

3.16 BERT processes a sentence to output contextualized embeddings that are more representative of the true meaning (source [135]). 86

3.17 A simple text document with a length of 5 words is selected where [CLS] is a special token which signifies that the task is a classification task. This text document is tokenised and given as input to BERTBase. The contextualised embeddings are generated using BERTBase as a feature extractor and the output of the last encoder layer is used. Two different methods for creating contextualised Textual Images are depicted. The BERTBase output can be reshaped directly into a Type A greyscale contextualised Textual Image, or it can be processed by a BLSTM block and reshaped into a Type B contextualised Textual Image. 87

3.18 Comparing different interpolation methods 88

4.1 Sentiment Analysis pipeline showing the use of BLSTM to generate Textual Images and ResNet to extract sentiment from Textual Images. 90

4.2 Residual Block Type-1, including BN- Batch normalisation layers, where ReLU is the activation function and a convolution block where the kernel size of 3x3, and where f is the number of kernels, and the stride value, s, is 1 or 2. 91

4.3 Residual Block Type 2, BN- Batch normalization layers, where ReLU is activation function. The Convolution blocks have kernel size of 3x3 or 1x1, where f is the number of kernels, and the stride values, is 1 or 2. 92

4.4 Average model performance, with its standard deviation, for different numbers of Residual Blocks using the 80,000 dataset. 95

4.5 Average model performance with its standard deviation vs Number of Residual Blocks for the 200,000 dataset. 96

4.6 Average model performance with standard deviation vs Number of data points on various models. 97

4.7 Average model performance with standard deviation vs Number of data points on Residual Type-1 & Type-2. 100

4.8 Comparison of Experiment 2 results containing baseline plus type-1 residual block with type-2 residual block models. 101

5.1 A tokenised text document is shown with five tokens, where [CLS] is a special token signifying classification task. The contextualised embeddings are generated using $\text{BERT}_{\text{Base}}$ as a feature extractor and the output of the last encoder layer is used. $\text{BERT}_{\text{Base}}$ output can directly be reshaped into a contextualised Textual Image (Type A), or it can be given to the BLSTM block and then reshaped into a contextualised Textual Image (Type B). 104

5.2 Accuracy vs Dataset size for subset 20,000 to 1,600,000. Comparison between BERT-BLSTM-ResNet models with the Baseline model BLSTM-ResNet-12. 106

5.3 Accuracy vs Dataset size for subsets from 20,000 to 1,600,000. Comparison between BERT-BLSTM-ResNet models and the Baseline BLSTM-ResNet-12 with BERT 107

5.4 5-fold cross validation Accuracy vs Dataset size plot for the smaller subsets of Sentiment140 from 1250 - 160,000. Comparison between BERT-BLSTM-ResNet 18 model with BERT. 107

5.5 Log plot of Accuracy vs Dataset size over various datasets. Comparison between BERT-BLSTM-ResNet models with BERT. For clarity, only the BERT and ResNet18 variant are shown with a line to illustrate trends. 108

5.6 Token Density plot for IMDB train (left) and test set (right) 108

5.7 BERT and different BERT extended models including BLSTM and/or ResNet. Different configurations use different types of contextualised Textual Image. 109

5.8 Accuracy vs Token length for BERT and BERT Extended models 111

5.9 FLOPs vs Token length for BERT and extended Models 112

6.1 End to End pre-trained model for Textual Image classifier. 119

6.2 Pipelines of existing sentiment analysis. (a) employs transformer-based techniques adopted in contemporary research using lexical resources, while (b) employs Bag-of-Words techniques. 119

6.3 Average Accuracy with std vs Data size for BERT, BERT-ResNet and BERT-ViT 122

6.4 Best Accuracy vs Dataset size for BERT, BERT-ResNet and BERT-ViT . . . 122
6.5 Venn diagram showing the overlapping of the test set predictions for BERT, BERT-ResNet (as ResNet), BERT-VIT (as ViT) . . . 125
6.6 (a) Mean Accuracy with standard deviation on various interpolations for different visual backbone classifying Textual Images. (b) Different best performing Model across interpolation for classifying Textual Images. . . . 126
6.7 Average Accuracy Heatmap for the BERT-Efficient Models . . . 127
6.8 Exponentially Doubling Data Size vs Average Accuracy for different BERT-Efficient Models . . . 127
6.9 BERT-Efficient Model capacity vs Best Accuracy for Exponentially Doubling Data Sizes . . . 128
6.10 Average accuracy with standard deviation vs α for the Word sentiment Classifier . . . 129
6.11 LSA and LDA performance across various text preprocessing combinations using the SVM classifier for sentiment analysis. . . . 130
6.12 Prediction overlap analysis for various sentiment extraction methods (SVM represents combined LSA and LDA features and BERT-WordClassifier represents BERT-SWC) . . . 131

9.1 Example CV with 5 folds - bifurcating training set in to 5 equal sections . . . 151
9.2 Unrolled RNN like a Feedforward network . . . 154
9.3 Different configurations in which RNNs are used . . . 154
9.4 Internal operations inside a simple RNN . . . 154
9.5 I2o-Input to Output Linear layer inside RNN . . . 155
9.6 I2H-Input to Hidden state Linear layer inside RNN . . . 155
9.7 Multi-layer RNNs . . . 155

10.1 Different Learning Rate decay curves for 50 epochs with learning rate reducer . . . 166
10.2 Accuracy vs different Learning rate decay methods for 50 epochs training . . . 166

12.1 BERT-Efficient Model capacity vs Average Accuracy for Exponentially Doubling Data Sizes . . . 187
12.2 BERT-Efficient Model capacity vs Average Accuracy for Doubling Dataset Sizes . . . 187
12.3 Doubling Data Size vs Average Accuracy for different BERT-Efficient Model . . . 188
12.4 Exponentially Doubling Data Size vs Best Accuracy for different BERT-Efficient Model . . . 188
12.5 BERT-Efficient Model capacity vs Best Accuracy for Doubling Dataset Sizes . . . 189
12.6 Doubling Data Size vs Best Accuracy for different BERT-Efficient Model . . . 189
12.7 IMDB dataset top words . . . 189
12.8 IMDB dataset box plot . . . 190
12.9 LSA vs LDA components impact of MLP . . . 191
12.10 Various LSA + LDA components combined with different pre-processing combinations logistic Regression . . . 192
12.11 Various LSA + LDA components combined with different pre-processing combinations MLP . . . 193
12.12 Various LSA + LDA components combined with different pre-processing combinations Naïve Bayes 194
12.13 Various LSA + LDA components combined with different pre-processing combinations SVM . . . 195
12.14 Various LSA + LDA components combined with different pre-processing combinations XGBoost . 196

List of Tables

2.1 Systematic Literature review final model comparisons 40

3.1 Relationship between the experimental pipelines, controlled comparisons, experimental chapters and research questions. 54

3.2 Regular expression patterns, match and example. 56

3.3 Example words with same stem 57

3.4 Example words with same lemma 58

3.5 NLTK library POS tags examples with description 58

3.6 Example of different Synsets for word "watch" from WordNet and associated SentiWordNet scores 59

4.1 List of BLSTM-ResNet variants with the total number of residual blocks. The number following the BLSTM-ResNet model name indicates the total number of CNN layers. The Shallow/Deep type identifies the residual blocks used in each BLSTM-ResNet variant. 93

4.2 Average model performance, with its standard deviation, for different numbers of Residual Blocks using the 80,000 dataset. 94

4.3 Average Model performance with its Standard deviation for 200,000 dataset size 95

4.4 Average Accuracy with standard deviation for datasets with different sizes for models of various depths. 97

4.5 Average weighted F1 score with standard deviations for datasets with different sizes for models of various depths. 97

4.6 Friedman statistics on Experiment 2 results 98

4.7 Comparison of Residual Type-1 & Type-2 using Average Accuracy with standard deviation for 3 and 4 residual block models on different dataset sizes. 99

4.8 Comparison of Residual Type-1 & Type-2 using Average Weighted F1 scores with standard deviation for 3 and 4 residual block models on different dataset sizes. 99

4.9 Friedman statistics on Experiment 3 results. 100

5.1 $\text{BERT}_{\text{Base}}$-BLSTM model variants and acronyms. 105

5.2 Accuracy given as a percentage and the confusion matrix ([[TP, FN], [FP, TN]]) of various models on different Sentiment140 sizes. The best result for each dataset size is highlighted in bold, or blue, if the best is BERT on its own. 106

5.3 Model accuracy and standard deviation across different dataset sizes. 107

5.4 BERT and BBR Model results using Accuracy, shown as a percentage, for subsets from a size of 5k up to the entire dataset of Sentiment140. In this table, 'k' represents 1000. 107

5.5 Accuracy vs Token length for BERT and BERT Extended models. The best result for each token length is highlighted in bold. 110

5.6 Different Model Parameters as well as FLOPs for various token lengths. 112

5.7 BERT vs Extended model performance analysed using the Wilcoxon significance test 113

6.1 Comparing a pretrained ResNet and ViT as a viable substitute for the custom visual backbone for Textual Image classification using average accuracy, and number of Epochs needed for best performance, all averaged over 5 runs. Note: K stands for a thousand and M for a million, AVG for average, and STD for standard deviation. The best Average performance for each data set size is shown in bold. 120

6.2 Various pretrained vision backbone methods, showing the average accuracy with standard deviation across different interpolations for Textual Images along with the Average number of Epochs, with standard deviation, needed for reaching the best performance. In Each case the average is over 5 runs. Again, the best result is highlighted in bold. 123

6.3 Average Accuracy with standard deviation for BERT-EfficientNet Models (bold values represent best individual model performance at different data percentage and blue indicates the best performance across models for the that data percentage) 126

6.4 EfficientNet model number of parameters and Flops needed 127

6.5 Word sentiment Classifier average accuracy with standard deviation across various values of α . . . 128

6.6 Ensemble performance using various meta learners. 131

6.7 Various Ensembling results for Sentiment140 subsets. SOFT Voting with Equal Weights. WAVG is the weighted AVG. LR is logistic regression and SVM is Support vector machine with RBF kernel. 132

6.8 Comparison of the IMDB results in Chapter 6 with selected published results across architecture families. Values marked "converted" were reported as test error in the source and are shown as accuracy using $100 - \text{error}$. 132

6.9 Methodological comparison with closely related text-as-image vision-transfer work. These results use different data splits and visualisation procedures and must not be interpreted as a head-to-head numerical comparison. 134

10.1 Combined Average accuracy result with std for all the experiment in this Section 4. Some models have N/A for some datapoints as these models were not tested for these data points in any experiment. 156

10.2 Combined Average weighted F1 result with std for all the experiment in this Section 4. Some models have N/A for some datapoints as these models were not tested for these data points in any experiment. 156

10.3 Individual test set performance of BLSTM-2DCNN model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 157

10.4 Individual test set performance of BLSTM-ResNet-3 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 158

10.5 Individual test set performance of BLSTM-ResNet-6 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 159

10.6 Individual test set performance of BLSTM-ResNet-9 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 160

10.7 Individual test set performance of BLSTM-ResNet-9 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 161

10.8 Individual test set performance of BLSTM-ResNet-15 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 162

10.9 Individual test set performance of BLSTM-ResNet-18 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 162

10.10 Individual test set performance of BLSTM-ResNet-21 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 163

10.11 Individual test set performance of BLSTM-ResNet-27 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset. 163

10.12 Individual test set performance of BLSTM-ResNet-36 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset 164

10.13 Accuracy of various Learning rate decay methods for 50 epoch training 165

11.1 5-fold cross validation accuracy result and confusion metrics for Sentiment140 subsets for BERT model 167

11.2 5-fold cross validation accuracy result and confusion metrics for Sentiment140 subsets for BERT-BLSTM-ResNet18 model . . . 168
11.3 BERT Model parameters, Thop parameters, MACs and Flops . . . 168
11.4 BERT-BLSTM Model parameters, Thop parameters, MACs and Flops . . . 168
11.5 BERT-BLSTM||ResNet Model parameters, Thop parameters, MACs and Flops . . . 168
11.6 BERT-BLSTM-ResNet Model parameters, Thop parameters, MACs and Flops . . . 169
11.7 BERT-ResNet-50 Model parameters, Thop parameters, MACs and Flops . . . 169
11.8 Experiment result for 10 runs with average performance and standard deviations. . . . 169

12.1 Sentiment140 performance on BERT model. . . . 171
12.2 Sentiment140 performance on BERT-ResNet . . . 172
12.3 Sentiment140 performance on BERT-ViT . . . 172
12.4 Agreement Analysis by Dataset Size 625 (BERT vs BERT-ResNet vs BERT-ViT) . . . 173
12.5 Agreement Analysis by Dataset Size 1250 (BERT vs BERT-ResNet vs BERT-ViT) . . . 173
12.6 Agreement Analysis by Dataset Size 2500 (BERT vs BERT-ResNet vs BERT-ViT) . . . 174
12.7 Agreement Analysis by Dataset Size 5000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 174
12.8 Agreement Analysis by Dataset Size 10000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 175
12.9 Agreement Analysis by Dataset Size 20000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 175
12.10 Agreement Analysis by Dataset Size 40000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 176
12.11 Agreement Analysis by Dataset Size 80000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 176
12.12 Agreement Analysis by Dataset Size 160000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 177
12.13 Agreement Analysis by Dataset Size 320000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 177
12.14 Agreement Analysis by Dataset Size 640000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 178
12.15 Agreement Analysis by Dataset Size 1280000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 178
12.16 Agreement Analysis by Dataset Size 1600000 (BERT vs BERT-ResNet vs BERT-ViT) . . . 179
12.17 IMDB Dataset (CNN vs Transformer) Models accuracies across 5 runs along with the Epochs to best performing model. . . . 179
12.18 Model variants with different interpolation ranked based perfromace . . . 180
12.19 Critical Value Calculation for difference in performance for different vision backbones (Dunn's Post-hoc Test) . . . 180
12.20 Significance test based on interpolation method on Textual Image classification of pre-trained vision models. . . . 181
12.21 Significance is difference in average model rank for various interpolation methods. . . . 181
12.22 Friedman test followed by Nemenyi post-hoc Significance test based on interpolation and pretrained vision model variant pairs on Textual Image classification. . . . 183
12.23 Overall average Ranks of various interpolation method across various pre-trained vision classifier performance on contextual Textual Images. . . . 183
12.24 Overall average Ranks of various pre-trained vision classifier performance across various interpolation methods on contextual Textual Images. . . . 183
12.25 Pre-trained Vision Model parameters and their respective parameters and giga FLOPs required per contextualised Textual Images processing. . . . 184
12.26 Efficientnet_b0 accuracy on Contextualised Textual Images various runs in three epochs. . . . 184
12.27 Efficientnet_b1 accuracy on Contextualised Textual Images various runs in three epochs. . . . 185
12.28 Efficientnet_b2 accuracy on Contextualised Textual Images various runs in three epochs. . . . 185

12.29Efficientnet_b3 accuracy on Contextualised Textual Images various runs in three epochs. 185
12.30Efficientnet_b4 accuracy on Contextualised Textual Images various runs in three epochs. 185
12.31Efficientnet_b5 accuracy on Contextualised Textual Images various runs in three epochs. 186
12.32Efficientnet_b6 accuracy on Contextualised Textual Images various runs in three epochs. 186
12.33Efficientnet_b7 accuracy on Contextualised Textual Images various runs in three epochs. 186
12.34BERT-SWC model performance vs alpha. 197

Chapter 1

Introduction

1.1 Introduction

At its core, machine learning aims to create algorithms that learn from data to identify patterns, make predictions, and adapt to new information, thereby addressing complex problems across domains ranging from healthcare to finance, weather prediction to drug discovery. It is driving almost every major industry. Central to this process is the data itself, which serves as the foundation for building and optimising models.

The nature, quantity and quality of the data significantly influence the entire data modelling process. Data can be organised or unorganised as well as structured or unstructured. Structured data refers to information that is highly organised and easily searchable, typically stored in relational formats such as tables with rows and columns. Each column in such a database has set datatypes with or without constraints. The SQL databases are a classic example of this where every row is stored according to predefined rules and constraints.

For machine learning particularly, some degree of organisation and structure in the data is required for effective processing. Each row of data to be processed, representing a single instance or record, is transformed or vectorised into a numerical format suitable for computation. The vectors exist in mathematical spaces like Cartesian, Polar etc., thus enabling algorithms to analyse the relationships, distances and patterns in data.

The question arises, if data in the digital age can be stored in structured formats with defined constraints, why does so much of it remain unstructured or unorganized? The answer lies in the inherent diversity of data generation and collection. While storage systems may be capable of structure, the data itself encompasses a spectrum including structured, semi-structured, and unstructured formats, such as free text, high-resolution images, and audio. Therefore, it is required to process the data into a desired format appropriate for machine learning. Figure 1.1 shows the broad categories of data based on the structure.

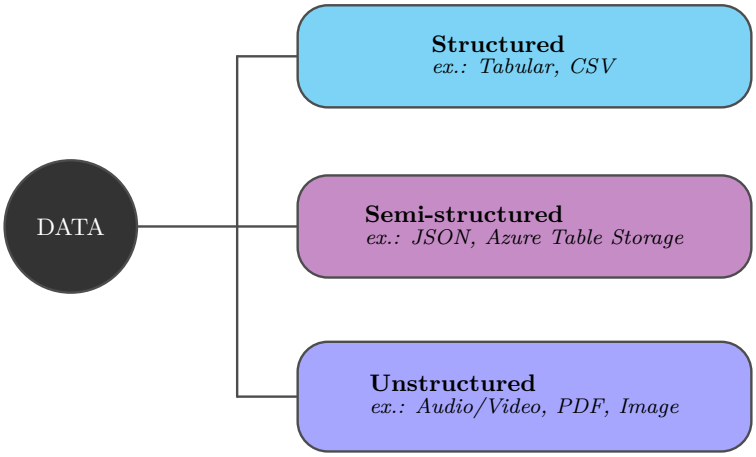


Figure 1.1: Data categories (Structured, Semi-structured and Unstructured)

Even when data is stored digitally, it often lacks the rigid structural constraints required for machine learning algorithms. Consider text data, which is stored with a basic organisation, such as having a string or character data type. However, text lacks certain properties associated with structured data, such as fixed constraints or upper limits, which are common in numerical or Boolean data types. Though digital data from one source may

have a fixed format, variability arises from different sources. To perform data analysis or data modelling, the data should have some fixed representation. The fixed representation for text data modelling is achieved primarily by establishing the maximum length of the input sequence.

This unstructured text data pervades digital ecosystems, appearing in diverse formats such as documents, social media posts, chat logs, and transcripts. While establishing a fixed sequence length provides a structural basis for modeling, it introduces significant challenges tied to the inherent complexity of language. Factors such as context, semantics, syntax, and variability across languages complicate efforts to constrain text into fixed forms without losing meaningful information. These issues highlight the intricate relationship between raw text and its mathematical representation, which constitutes the core focus of Natural Language Processing (NLP). A more detailed exploration of this topic, including the history and background of NLP, will be presented in Chapter 2

Text processing techniques have emerged as essential tools for transforming unstructured text into meaningful and analysable representations, enabling the application of machine learning and natural NLP methods.

One of the most prominent applications of text processing is text classification, a task that involves assigning predefined labels to text based on its content. Text classification underpins numerous NLP tasks, including spam detection, topic modelling, and sentiment analysis. Sentiment analysis has gained widespread popularity as a specialised form of text classification that focuses on determining the emotional tone or opinion expressed in a piece of text.

Text classification commonly treats a document's numerical representation as a sequence. However, arranging fixed-dimensional word representations by token position also produces a two-dimensional feature map. This thesis investigates whether analysing this structure as a Textual Image allows established two-dimensional pattern-recognition methods from computer vision to contribute to sentiment classification. The detailed motivation for this alternative perspective is developed in Section 1.5.

1.2 Sentiment Analysis

The Internet is flooded with social media platforms with an enormous user base. Large social-media platforms generate very high volumes of posts each day, creating large-scale streams of user-generated opinion data [1]. Online forums where the user can share their opinion on a topic are considered opinion-rich sources by [2]. One prominent example of such a platform is X, formerly known as Twitter [3]. It is a microblogging and social networking service, where users post and interact using short messages known as tweets. This text data is quite different from the traditional print media where a level of objectivity is presented.

The amount of data produced every day on social media is so huge that analysing it requires advanced Artificial Intelligence and Machine learning techniques. Online platforms made it possible to grasp the immediate reaction of the masses quickly on any topic facilitated through social media platforms. This form of user feedback was a tedious and time-consuming process in the pre-internet era. Different organizations are eager to know their users' opinions regarding their products and services, and political parties are curious to know the hot political issues, or their popularity among the voters the technique used for solving such problems is called sentiment analysis.

Sentiment analysis (SA) refers to the study of the emotion-based opinions of the subject over some observed topics. Sentiment and emotion might be used interchangeably in our daily language but the difference between emotion and sentiment lies in their definitions, scope, and how they are interpreted or analysed in contexts such as psychology, linguistics, and data analysis.

Humans display several emotions and emotions are complex psychological states that involve subjective experiences, physiological responses, and behavioural expressions. The term emotion designates a collection of responses triggered from parts of the brain to the body using neural and humoral pathways [2]. According to Paul Ekman, an American psychologist, emotions are universal traits that have evolved in humans. He found that people from diverse cultures, both Western and Eastern, could consistently agree on the emotional labels that match specific facial expressions for six basic emotions including anger, disgust, fear, happiness, sadness and surprise [4].

Sentiment analysis provides a general sense of attitude or opinion from a piece of text without delving into specific emotional states. Sentiment analysis is implemented in a wide range of areas like real-time election analysis [5, 6], analysis of movie reviews [7, 8], stock market prediction [9, 10, 11], depression diagnosis [12], and many more. The text under observation is categorised or classified broadly into three major groups based on the emotional tone expressed.

A text can be categorised as subjective or objective based on the nature of the information it holds. This classification helps filter text which reflects opinions, emotions, or personal judgments. Figure 1.2 shows the text categorisation based on the objective and subjective text along with further categorisation of subjective text.

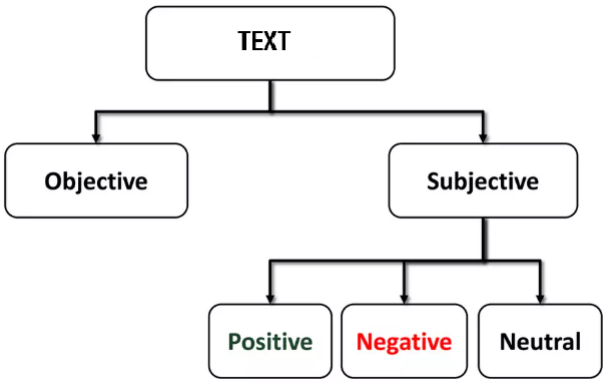


Figure 1.2: Classification of Text in object and subjective text

Objective text tends to be factual in nature or present information that is independent of personal feelings or opinions. These sentences do not convey any emotional tone or bias and can be found in scientific documents, technical manuals or descriptive data.

Example of objective text.

"The Earth revolves around the Sun and the time it takes to complete one revolution is equal to one Earth year."

Subjective text expresses personal opinions, emotions or attitudes reflecting writers' thoughts and feelings. Such text can be utilised for developing supervised classification models if labelled data is available. Based on the categories of labels present in the data set different types of sentiment analysis can be performed.

1. **Binary Sentiment Analysis:**

   This category classifies text as "Positive" and "Negative". Usually, in datasets, a Boolean value is used to represent the class label true for Positive and False for Negative, but it is also common to have values 0 and 1 to represent negative and positive values respectively.

   Example of a Negative Text:

   "Despite the product being advertised as durable, it broke within a week of use, which was incredibly frustrating and disappointing."

   Example of a Positive Text:

   "I absolutely loved how the event was organized, from the decorations to the seamless scheduling of activities."

2. **Ternary sentiment analysis:** This category adds one more category and classifies text as "Positive", "Negative" or "Neutral". This representation usually is -1 for negative, 0 for neutral and 1 for positive sentiment class.

   Example of Neutral Text:

   The event started on time and went as planned, nothing particularly memorable about the experience.

3. **Fine-grained sentiment analysis**: This Classifies text on a more granular scale to express the intensity of the sentiment such as rating it between 1-5 or 0-9 etc.

   Example of Fine-grained sentiment label:

   Slightly Positive or 6:

   "The product arrived on time and works as expected, but it doesn't stand out in any way compared to other options I've tried."

   Very Positive or 9:

   "The customer support team went above and beyond to address my concerns, responding promptly and ensuring everything was resolved seamlessly."

Negative or 2:

"The app frequently crashes, and many of the advertised features are either missing or don't function properly, leading to a frustrating experience overall."

1.3 Levels of Sentiment Analysis

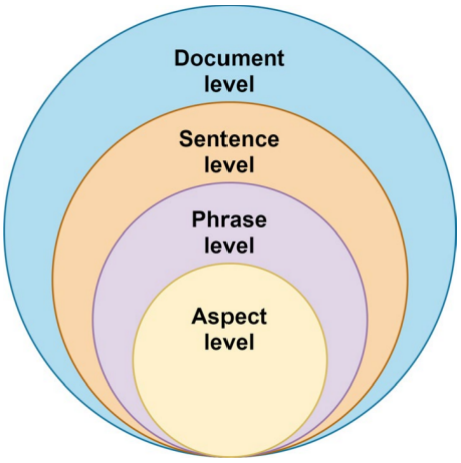


Figure 1.3: Levels of Sentiment Analysis [13]

Sentiment Analysis (SA) has been applied to text from a wide range of domains, with the approach tailored to the requirements of each task. SA can be more difficult than other text-analysis tasks. Text may contain abbreviations, jargon and implied meanings, and the meaning of a post can depend on its context. Social-media text is also influenced by social context [14]. Social context includes the user, content, relations and interactions, including links between a post, associated media and user reactions.

Research on SA is typically categorized by the granularity of the text being analysed. A common visual representation of these levels is shown in Figure 1.3, which depicts a nested hierarchy ranging from the broad Document level down to the Sentence, Phrase, and finally Aspect level [13]. While this model effectively illustrates the decreasing size of the textual length, it is important to note that the relationship between these levels is not strictly hierarchical. Specifically, the distinction between the first three levels is structural (based on text length), whereas the fourth, Aspect level, is semantic (based on the subject matter).

The proposed Textual Image approach addresses document-level sentiment analysis. Each tweet or review is represented as a single input document and assigned an overall sentiment label for binary classification. It does not attempt to produce separate sentence-, phrase- or aspect-level sentiment predictions.

1.3.1 Document level

When the SA is performed on an entire text document it is considered a document-level SA. In this setting, the entire document is under observation and is assigned an overall sentiment polarity. This resembles a text-classification task in which a text document is labelled as one class such as labelling a news article in different categories like politics, entertainment, fashion, or sports. Document-level SA is most appropriate when the document expresses a dominant overall opinion; it is less appropriate for long or complex documents containing multiple opinions or sentiment targets [13, 15]. Short social-media posts, including tweets, are often treated as document-level instances and are assigned one overall label in benchmark datasets. This is a modelling assumption for classification, rather than a claim that every tweet contains only one unambiguous sentiment [16]. For example, a review stating, "The film is engaging and well acted," would be assigned an overall positive label.

1.3.2 Sentence level

Sentence level SA considers each sentence during analysis, and a sentiment polarity is associated with each sentence or a group of sentences like an entire paragraph. This approach is suitable when a text document contains varied opinions, or a range of sentiments is associated with it. Sentence-level analysis is like subject-specific analysis where the document discusses a wide range of subjects. This level is a step further than the Document level

analysis, as the training data requires more sentiment labels and more training resources [17]. The sentence-level sentiment polarities can further aid in calculating the overall Document-level sentiment [13]. For example, in a review containing "The screen is clear." and "The battery drains quickly.", a sentence-level system can assign positive sentiment to the first sentence and negative sentiment to the second.

1.3.3 Phrase level

In phrase level SA, the focus is further narrowed to a phrase level identifying the collection of words that showcase the subjectivity. This analysis is useful when multiple lines of user review are provided on a product or service, and multiple aspects of the product are discussed all of which can have different opinions. A foundational advancement at this level introduced structural parse trees to evaluate sentiment composition across individual phrases within a sentence [17] In this level, sentiment regarding a particular product category can be identified based on the sentiment associated with it instead of all the overall text [13]. For example, in the sentence "The screen is clear but the battery drains quickly," the phrase "screen is clear" can be assigned positive sentiment and "battery drains quickly" negative sentiment.

1.3.4 Aspect level

Aspect-level SA represents the finest level of granularity in the standard hierarchy, yet it differs fundamentally from the preceding categories. As illustrated in Figure 1.3, some literature positions Aspect-level as a subset of Phrase-level analysis [7, 13]. However, this hierarchical placement is arguably an oversimplification. While Document, Sentence, and Phrase levels are defined by linguistic boundaries and length, Aspect-level analysis is defined by the target entity (the "aspect") toward which an opinion is directed [18].

For instance, a user might express a sentiment regarding a specific aspect (e.g., "battery life") using a single phrase, a whole sentence, or implicitly throughout a document. Therefore, while Figure 1.3 correctly places Aspect analysis at the deepest level of detail, conceptually, it does not merely "fit inside" a phrase. Instead, Aspect-level analysis cuts across these structural boundaries, requiring the system to identify both the target feature and its specific sentiment, regardless of the syntactic structure holding that information. This distinction makes Aspect-level analysis significantly more complex than simply analysing a smaller unit of text.

1.4 Machine Learning in Sentiment analysis

An early example of computer aid in content analysis is "*The General Inquirer System*" [19], developed at Harvard in the 1960s. This system was performing objective, quantitative and systemic analysis on the corpus of text from different domains like Psychology, Sociology, and Political sciences. The information from the text, which was in alphanumeric form, was fed to the system using the technology of the time. The quantitative analysis included counting positive and negative instances in the text.

With Machine learning models, many sentiment analysis models have surfaced which incorporate supervised learning to identify the sentiment of the text. Traditional machine learning frameworks like decision trees, Bayesian, and support vector-based models were devised to extract opinions from sentences [13]. These models utilised the Bag of Words technique, which involved converting the sentences into tokens and then forming a frequency-encoded vector from these tokens called word vectors [20]. BOW used token count to form the feature space. However, this method suffers from multiple limitations. First, it leads to high data sparsity, as the feature space grows linearly with the vocabulary size. Second, it disregards word order and syntactic structure, meaning it cannot distinguish between sentences containing the same words in different sequences. Finally, it fails to capture semantic relationships, as it treats synonymous words as completely unrelated features [20, 21].

Deep learning has gained popularity recently and emphasises a layered structure composed of different modules. An efficient method called Continuous Bag of Words was developed to create word vectors with smaller dimensions by learning from the surrounding context [22]. These vectors are referred to as word embeddings, and the layer used in deep learning to convert word tokens into such vectors is called the Embedding layer.

The convolutional layer is among the most popular deep-learning layers for extracting features from input tensors. Two dimensional (2D) Convolutional layers have become particularly successful in image-analysis tasks by learning local and hierarchical spatial patterns [23] and even surpassing human performance. On the ImageNet classification task, ResNet, a 2D convolutional neural network (CNN) varient, achieved a 3.57% top-5 error, which was below the reported 5.1% human error rate [23]. Similarly, a one-dimensional variant of the image-convolution layer is adopted to perform convolution on word vectors and extract crucial features for textual classification tasks such as

sentiment analysis by identifying salient local patterns from neighbouring tokens.

Recurrent Neural Networks (RNNs), which are popular for time series analysis, are also used in text analysis because they process word tokens sequentially while maintaining a hidden state that carries information from the previous time step (token). This enables them to model word order and temporal dependencies in tasks such as sentiment analysis. Common variants include Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks [24], which use gating mechanisms to retain information over longer sequences, and bidirectional RNNs, including Bidirectional LSTM (BLSTM), which analyse a sequence in both forward and backward temporal directions to capture word dependencies in both directions [25].

Hybrid architectures combining RNNs with convolutional feature extraction have also been explored for text analysis [26], because the two components provide complementary capabilities. In architectures using a 1D-CNN, the convolutional layer efficiently extracts salient local features, whereas the RNN models the sequential context and dependencies among these patterns. Their combination can therefore capture both local features and longer-range context within a text sequence.

A conventional 1D-CNN processes a text sequence represented by an embedding matrix of size $N \times D$, where N denotes the sequence length and D denotes the embedding dimension. A filter of size $k \times D$ is applied to windows of k consecutive tokens and moved along the sequence axis. Each operation therefore combines all D features of the tokens within a window to produce a single local feature. This is effective for detecting patterns similar to n-grams, but it restricts the convolution to one direction: the model learns how features vary across token positions without independently exploring patterns along the embedding-dimension axis. Moreover, the early aggregation of all D dimensions may obscure more localised interactions among subsets of embedding features.

This restriction motivates considering whether the embedding matrix can be analysed across both of its dimensions. Two-dimensional convolutional networks have achieved substantial success in image classification, with architectures such as ResNet [23] providing mature methods for learning spatial patterns from an Image. A document represented by stacked word embeddings can be viewed not only as a sequence but also as a two-dimensional feature map: one axis represents token position, while the other represents the dimensions of each word embedding. This representation is not a natural image, and its embedding values are not literal pixels. Instead, treating it as a Textual Image creates a testable hypothesis that two-dimensional neural operations may identify useful patterns across both axes for sentiment classification.

A 2D-CNN provides an alternative way to analyse the same $N \times D$ representation. Rather than using a filter that spans the complete embedding dimension, a smaller two-dimensional filter, such as 3×3, moves across both the token-position and embedding-dimension axes. Instead of processing the entire width of the embedding at once, a 2D-CNN can perform more granular processing, sliding not only across the sequence of words but also across the embedding dimensions. This operation allows the network to examine interactions within local regions of the feature map before combining them into higher-level representations.

An architecture that first uses a recurrent neural network to learn sequential dependencies and then applies a 2D-CNN to extract spatial features was explored by Zhou et al. [27]. This previous work provides a foundation for representing the numerical representation of text as an image-equivalent feature map. However, the problem has not been analysed in depth by systematically exploring different ways of forming Textual Images and then analysing them using a range of computer-vision approaches. Addressing this gap offers a mechanism for testing whether patterns across both axes contain useful information for sentiment analysis and whether Image-equivalent representations of text can benefit from more recent computer-vision architectures.

The motivation for testing this hypothesis and the associated architectural variations is developed in Section 1.5, while the supporting literature is reviewed in Chapter 2.

1.4.1 Machine Learning and Sustainability

The rapid advancements in machine learning, like any other transformative technology, have far-reaching implications for society. While these developments bring tremendous opportunities to solve complex problems and enhance decision-making processes, they also come with significant challenges, particularly in terms of resource consumption and environmental impact. Machine learning models, especially those involving deep learning architectures, often require substantial computational power and energy to train and deploy. This has led to increasing concerns about the sustainability of these technologies and the need for efficient, resource-conscious practices in model development.

To address these challenges, it is imperative to adopt sustainable model-building approaches that balance performance improvements with resource efficiency. Sustainable model development emphasizes optimising computational resources, reducing energy consumption, and minimizing the environmental footprint of machine learning processes. This requires exploring innovative techniques and methodologies, such as reducing model complexity, utilising precision optimisation strategies, and enhancing reusability through fine-tuning and ensembling methods.

1.5 Motivation for the research

Social-media platforms continuously generate large and rapidly changing streams of user opinions. The scale, speed and linguistic variability of this content make consistent manual analysis impractical, particularly when organisations need timely insight into public attitudes towards products, services and events. These conditions have necessitated robust and automated methods for Sentiment Analysis (SA) [1, 13]. Traditional text processing has evolved from simple Bag of Words (BOW) models to dense vector representations such as GloVe, while tremendous progress has also been made in image-processing methodologies. The motivation for this research therefore lies at the intersection of NLP and Computer Vision.

Previous work, particularly the BLSTM-2DCNN method proposed by Zhou et al. [27], demonstrated that a recurrent representation of text could be analysed using two-dimensional convolution (refer section 2.2.6 for further details). However, this work focused on a specific architecture rather than an extensive exploration of Textual Image representation. Based on the literature reviewed and to the best of the author's knowledge, this thesis is the first to investigate the image representation of text in an extensive and systematic manner. It extends beyond the existing BLSTM-2DCNN method by exploring different ways of forming Textual Images and analysing them using a range of computer-vision approaches, including deeper residual networks, pretrained image classifiers and vision Transformers.

The central motivation is to investigate whether text classification can be reframed, in part, as a two-dimensional pattern-recognition problem. When the word representations in a document are arranged by token position, they form a two-dimensional numerical structure. Treating this structure as a Textual Image provides an image-equivalent feature map on which neural operations developed for image analysis can be applied. This interpretation is an architectural hypothesis rather than an assumption that embedding matrices possess the same spatial properties as natural images or that their values are literal pixels. In particular, neighbouring embedding dimensions are not guaranteed to have an inherent spatial relationship. The first motivation is therefore to determine empirically whether deep 2D-CNN architectures such as ResNet, can identify useful relationships across token positions and embedding dimensions for sentiment analysis.

Furthermore, the landscape of NLP has shifted from static embeddings that assign the same vector representation to a word regardless of its position or context to contextualised representations that are influenced by both. The advent of Transformer-based models, such as BERT [28] (Bidirectional Encoder Representations from Transformers), allows for dynamic word representation based on sentence context. The second motivation is to integrate these contextual embeddings into the Textual Image generation pipeline and determine whether they provide an advantage over static embeddings. Combining BERT's contextual awareness with BLSTM-based sequential modelling and the spatial feature extraction of deep 2D-CNNs also makes it possible to examine how performance changes with image classifier depth, dataset size and text length.

As the field of Deep Learning evolves, so does the architecture of vision models. The introduction of Vision Transformers (ViT) challenges the dominance of CNNs in Computer Vision. At the same time, the pursuit of accuracy must be balanced against computational complexity and resource consumption. The third motivation is therefore to compare pretrained convolutional and Transformer-based image classifiers for Textual Image analysis. Reusing publicly available model weights and evaluating architectures of different complexities allows their classification performance and computational efficiency to be examined together.

Finally, Textual Image classifiers may learn predictive patterns that differ from those captured by established text-based sentiment models. The fourth motivation is to determine whether their predictions provide complementary information and whether combining Textual Image, contextual, lexical and topic-based representations through ensemble methods improves sentiment classification.

The models and architectures introduced in this section are discussed in detail in Chapter 2.

In summary, this research bridges the gap between the linguistic depth of NLP and the pattern-recognition capabilities of Computer Vision. The investigation progresses from establishing the Textual Image representation, to enriching it with contextual information, comparing pretrained vision architectures and computational requirements, and finally evaluating its complementarity with established sentiment models. These four motivations define the research aim and lead directly to the four research questions presented in the following sections.

1.6 Aim of the research

This research aims to systematically investigate Textual Image classification for sentiment analysis by examining image formation, contextual representation, deep and pretrained image classifiers, computational efficiency, and integration with complementary text-based models.

1.7 Research questions

RQ1 What is the impact of converting textual data into image-equivalent representations (Textual Images), to support two-dimensional neural pattern recognition, on analysing text for sentiment?

RQ2 What is the impact of incorporating contextual information into Textual Images on analysing text for sentiment across varying depth of image classifiers, dataset sizes and text lengths?

RQ3 How do the architecture and complexity of pre-trained Convolutional neural network and Transformer-based Image classifiers affect the performance and computational efficiency of sentiment classification using Textual Images?

RQ4 To what extent do Textual Image classifiers using pretrained Image classifier provide complementary predictive information to established text-based sentiment models, and how does combining these representations affect classification performance?

1.8 Contributions to knowledge

This research advances the field of sentiment analysis by establishing a comprehensive framework for Textual Image Classification. It bridges the gap between NLP and Computer Vision, demonstrating that text, when represented as Image (2D feature maps), can be effectively analysed using deep convolutional architectures. The specific contributions are:

This research provides a systematic empirical evaluation of deep 2D-CNN architectures applied to images generated from textual embeddings (Chapter 4)

- The study empirically establishes that deep spatial feature extraction, utilising Residual Networks (ResNet), is highly effective for sentiment analysis. The proposed BLSTM-ResNet architectures (Shallow and Deep variants) successfully replace traditional shallow 2D-CNN components, demonstrating that text possesses spatial-like dependencies when mapped to a 2D latent space.
- The research identifies the limits of depth scaling in Textual Image classification. Experimental results on the Sentiment140 dataset demonstrated that while increasing depth (via Type-1 and Type-2 residual blocks) generally improves performance, a saturation point is reached (specifically at four residual blocks for Type-1) whereas the Type-2 residual block has an edge at smaller dataset sizes

By integrating Transformer-based contextual embeddings with Convolutional Networks, this research contributes significant insights regarding sample efficiency and feature representation (Chapter 5).

- The research introduces and validates the generation of Contextualised Textual Images using fine-tuned BERT embeddings. This approach outperforms static embedding methods (GloVe), proving that incorporating attention-based context into the image generation stage significantly enhances the downstream classifier's ability to extract nuanced sentiment.
- The investigation establishes that hybrid visual-textual models (BERT-BLSTM-ResNet) exhibit superior sample efficiency compared to standalone Transformer models. The experimental findings demonstrate that the inductive bias—the architectural preference for learning certain patterns rather than others—of the BLSTM-ResNet architecture provides a regularization effect, allowing the hybrid model to outperform a standalone fine-tuned BERT model on smaller dataset subsets (e.g., <20,000 samples), thereby addressing performance degradation in resource-constrained environments.

Addressing the computational constraints of deep learning, this thesis contributes a methodology for reducing the training overhead of sentiment analysis models (Chapter 6).

- The research validates the transferability of pre-trained computer vision backbones—image-classification networks whose learned visual features are reused for a new task—(e.g., ResNet, DenseNet, EfficientNet, ViT) to synthetically generated Textual Images. This eliminates the need to train custom visual feature extractors from scratch, offering a low-computation alternative that maintains competitive accuracy.
- Performance-Compute Optimisation: The study identifies EfficientNet-B5 as the optimal architecture for the performance-to-compute ratio among the evaluated vision models. Conversely, it highlights the limitations of specific architectures, notably the failure of Swin Transformers to converge, suggesting a mismatch between patch-based hierarchical biases and the structure of Textual Images.

- The investigation empirically establishes that the choice of interpolation technique (Nearest, Bilinear, or Bicubic) does not statistically impact classification performance in Textual Image analysis. Consequently, Nearest Neighbour interpolation is identified as the preferred method solely due to its superior computational efficiency, minimizing preprocessing overhead without compromising accuracy.

The research demonstrates that Textual Image Classifiers learn decision boundaries that are distinct from, and complementary to, traditional SA approaches. (Chapter 6).

- Through comparative analysis (Venn diagrams), the research proves that visual interpretation of text captures semantic signals distinct from those captured by BERT-ResNet, topic modeling (LSA/LDA) and lexically enhanced BERT model (BERT-SWC).
- By introducing a meta-learning ensemble strategy (using XGBoost) that combines the Textual Image Classifier (BERT-ResNet) with a lexically enhanced BERT model (BERT-SWC) and topic modeling, the research achieved a performance improvement (+1.17% on the IMDB dataset). This underscores the necessity of using complex, non-linear learners to effectively decode the intricate relationships among these diverse predictive signals.

1.9 Thesis organisation

The remainder of this thesis is organised into the following sections, systematically addressing the research aim and specific Research Questions (RQ). Chapter 2 establishes the theoretical foundation, reviewing NLP techniques from standard word vectors to contextualised embeddings (BERT). It explores the application of Computer Vision in NLP, specifically forming images from text and using 2D-CNNs, explore the literature forming basis for this work.

Chapter 3 details the experimental designs, hardware specifications, dataset specifications and machine learning pipelines for various Textual Images classification methods and other techniques that can be ensembles with it. It defines the architectures for the baseline BLSTM-2DCNN and the proposed BLSTM-ResNet, setting the stage for addressing the research questions.

Chapter 4 addresses RQ1 and empirically evaluates the efficacy of deep 2D-CNN architectures for sentiment analysis. It introduces the BLSTM-ResNet model, investigating the impact of network depth and dataset size on performance to validate the use of deep spatial feature extraction for textual data.

Chapter 5 addresses RQ2 and integrates attention mechanisms into the textual image generation process. It evaluates the impact of using contextualised embeddings (BERT) versus static embeddings within hybrid architectures (BERT-BLSTM-ResNet), analysing performance across limited training data and varying sequence lengths.

Chapter 6 addresses RQ3 and explores resource-efficient Textual Image classification strategies. It compares Convolutional (ResNet, DenseNet, EfficientNet) against Transformer-based (ViT and SwinFormer) vision models (RQ3) and investigates the feasibility of replacing custom ResNet model (vision backbone for textual image classification pipeline) with pre-trained classifiers to reduce computational overhead. It concludes by evaluating an ensemble strategy to compliment the Textual Image classifier with existing traditional and contemporary approaches to sentiment analysis.

Chapter 7 summarises the key findings related to all research questions, reflects on the contributions to knowledge, discusses limitations, and proposes future directions for sustainable, multimodal sentiment analysis.

Chapter 2

Background and Literature Review

An extensive literature review has been conducted as part of this project. This chapter systematically builds the theoretical foundation for the thesis, mapping the progression from traditional text processing to contemporary deep learning techniques. It begins by addressing the fundamental challenge of Vectorisation granularity (Section 2.1), detailing how unstructured text is converted into numerical data. This leads into the critical steps of Data Pre-Processing (Section 2.2), which covers data cleaning, text normalisation, and essential feature extraction methods like word embeddings. The review then transitions to core deep learning architectures for textual classification, specifically Two-dimensional (2D) Convolution Neural Networks (Section 2.3), establishing the research background in text analysis and Image classification. This sets the stage for a discussion on the novel method of forming images from text (Section 2.4); this represents a distinct direction for text analysis that forms the basis of this work and explores potential architectures to advance the Textual Image classification landscape. The chapter also examines the shift to modern contextualized word representations (Section 2.5), such as the attention mechanism and Transformers, which can further enrich the Textual Image by adding semantic depth. Finally, a Systematic literature review (Section 2.6) covers the adoption of transformer models for document-level sentiment analysis to understand how existing literature improves sentiment analysis and how those advancements can be adopted in Textual Image classification to improve performance and Sustainability (Section 2.7) covers potential approaches to sustainable model building.

Natural Language Processing (NLP), a core subfield of Machine Learning (ML), concerns the analysis and processing of human speech and text. Early approaches involved statistical models [29, 30, 31] based on probabilities, statistics, graphs etc. More recently, deep learning approaches have dominated the area effectively tackling problems like speech to text translations [32, 33], speech recognition [34, 35], language modelling [36], Machine translation [37] etc. The next subsections will delve deeply into the process of text processing and analysis using Sentiment Analysis as an application for easy understanding.

Since the advent of the Transformer model, model sizes have been increasing year on year. The current state of the art model is no longer contributed to by Academia alone rather it mostly requires collaboration with industry, who fulfil the intense hardware requirements to build and train these models. Secondly, most big AI players had their proprietary software packages tailored to run efficiently on their hardware set. The good thing about this is that the majority of these software packages and trained models are opensource, but new researchers must decide early on which framework to adopt in the start of their project since switching between frameworks is not that simple and straightforward. For example, popular deep learning frameworks for doing Deep learning in python i.e., Keras [38], and TensorFlow [39] are developed and maintained by Google [40] whereas PyTorch [41] is developed and maintained by Meta [42].

Switching between the frameworks is hard as each ecosystem comes with own implementation of algorithms and underlying methods to perform mathematical and statistical functions. Being opensource means that the machine learning community contributes to these packages, and that they are updated often and if one framework releases a new method a similar one will be often available to others after a while. But these differences impact model performances and comparing them as implementations of one method in one framework does not guarantee the same performance in another framework [43, 44]. A direct comparison between a proposed model's performance and results reported in the published literature is insufficient. A proper benchmarking requires the replication of the published model in the local deep learning environment and assessment using a specific, common set of parameters [45]. This methodology is compromised when critical implementation details are omitted from the original research. In addition, new advancements occur in many elements of model building, including training strategies, model optimisation and regularisation. These changes make it difficult to evaluate published literature against a particular model. For example, benchmarking a newly proposed neural layer architecture against published literature that used different training strategies or optimisation settings makes an objective comparison difficult, because model

performance is sensitive to these aspects of model building [46].

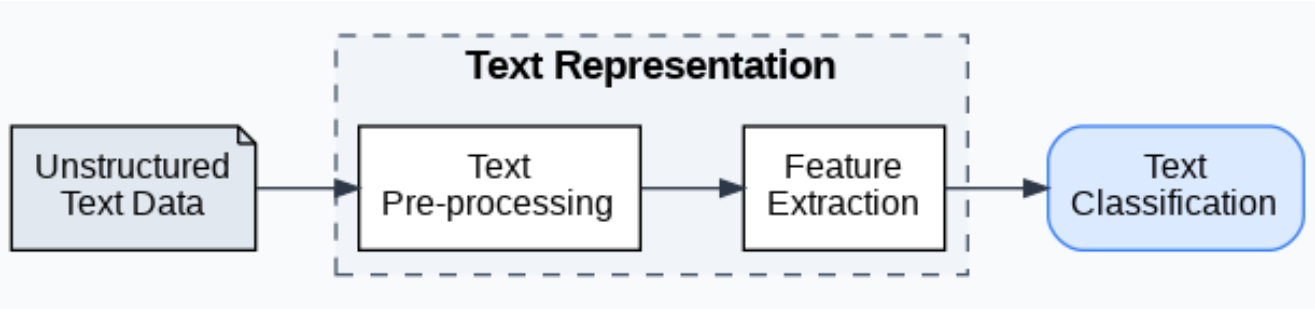


Figure 2.1: Text classification pipeline

Figure 2.1 illustrates the text classification pipeline, tracing the transformation of unstructured, low-quality text into definitive class labels through a comprehensive "Text Representation" stage. While the schematic delineates "Text Preprocessing" and "Features Extraction," it deliberately abstracts the modeling phase to reflect the architectural integration found in modern deep learning frameworks. Unlike traditional machine learning, which requires distinct steps for manual feature engineering and subsequent model training, deep neural networks perform feature learning and classification as an end-to-end process. In this context, the "Features Extraction" component represents the model's earlier layers learning latent representations, while the final fully connected layer generates the class probabilities; thus, the diagram logically groups these operations to show how the system flows directly from learned text representations to the final "Classification" output.

Throughout the subsequent sections, as each concept and its theoretical background are delineated, representative examples from the literature are integrated to cover the technical foundations. These examples served as the mechanism to comprehensively navigate the textual classification landscape, which in turn facilitated the proposal of the novel Textual Image classification method and the design of specific experiments to address the research questions outlined in Section 1.7. It is important to note that the investigation followed a cyclic process rather than a linear one; the formulation of research questions and the analysis of experimental results mutually informed one another throughout the thesis, ensuring that the approach to forming images from text was iteratively refined based on empirical evidence.

2.1 Vectorisation granularity

Text is an unorganised data which needs some transformation to represent in an organised manner. The organised representation of text is achieved by converting the text into a fixed length numeric representation, called vector representation, by encoding data like characters, words, phrases, sentences or entire documents.

Character level vector representation: Every character in a text document can individually be represented in a numerical form. The Unicode [29], associated with each character in any language, can be considered as a basic numeric representation of text on a character level. Even one-hot-encoding of each character in a text document is a simple example of character level encoding of text. The real benefit of this granular representation of text is to encode any text which has words that do not exist in a dictionary or has any misspelled word in the text [47]. It can be advantageous to a specific language [48] and aid in research involving social media posts [49]. These techniques have been used successfully [49]and also in detecting text in images [49, 50] as characters are more easily detected than words if the text is not in a straight line in an image used on Optical Character Recognition (OCR) tasks.

Word level vector representation: Word-level vector representation is crucial in Natural Language Processing (NLP) because words are often considered the fundamental unit of meaning and communication in languages [22]. Words function as the primary building blocks for expressing complex ideas, representing specific concepts, objects, actions, or qualities. Beyond their literal definitions (e.g., the word "dog" symbolizing a domesticated canine in English), written words also adhere to syntactic rules, contributing to the grammatical structure of a sentence through their part-of-speech function [51]. Furthermore, words often carry emotional or connotative meaning; for instance, "home" not only denotes a place of residence but also connotes feelings of safety, warmth, and family [52]. Techniques such as Continuous Bag-of-Words (CBOW) are seminal examples of generating a dense, low-dimensional vector representation of each word, known as a word embedding, which is designed to effectively capture both semantic and syntactic information [22].

Phrase level vector representation: A phrase in a language represents a group of words that function cohesively as a single unit to convey a specific meaning yet remains smaller than a complete sentence [25]. Phrases serve

various grammatical functions within a sentence, such as acting as a noun phrase (e.g., the subject) or an adjective phrase (e.g., a modifier) [52]. The concept of phrase-level vector representation aims to represent these multi-word semantics in a continuous vector space, like how individual words are represented [53]. Most methods for achieving this use existing word vector representations (word embeddings) as their foundation. Common techniques include using a simple weighted average of the constituent word embeddings to form the phrase vector, or, alternatively, concatenating the word embeddings to create a higher-dimensional representation that encodes the sequential nature of the phrase [25].

Sentence level vector representation: Sentence-level vector representation involves encoding an entire sentence into a fixed-length numeric vector designed to capture the sentence's full semantics, syntax, and context within a text document [53]. Like phrase-level methods, sentence representations frequently use existing word embeddings as their input to form the final vector [54]. An example of this method is the use of Recurrent Neural Networks (RNNs), in which the RNN processes the word embeddings sequentially. The final hidden state of the network, after processing the last word, often serves as the fixed-length vector representation for the entire sentence [55]. This final vector effectively summarizes the information flow and dependencies modelled across all words in the sentence.

Document level vector representation: This representation generates a unique numerical vector for an entire text document, effectively encoding the complete content and context [52]. Unlike character, word, phrase, or sentence representations, the document vector must capture the collective meaning across all these lower-level units. Crucially, methods for generating a document vector are flexible, as they can utilize any of the previously discussed representations from aggregated character statistics to sequences of word embeddings or entire sentence vectors as the building block for the final document vector [15]. A traditional, foundational approach is the Bag-of-Words (BOW) model, which generates a document vector based on the frequency of words in the document relative to a word corpus (the collection of all unique words across all documents) [52]. For instance, using one-hot encoding for word presence or frequency results in a vector where the dimension count equals the corpus size. This traditional BOW vector is characteristically high-dimensional and very sparse, contrasting sharply with modern continuous, dense vector representations like those generated by Paragraph Vector (Doc2Vec), which directly learns a fixed-length embedding to capture the document's semantics more efficiently [53].

In this research, document level sentiment analysis is employed as the primary evaluation task to validate the efficacy of the proposed Textual Image classification method; consequently, the adoption of document level vector representation is central to this work. Since the generated Textual Images are intended to visually represent the semantic content of the entire text document, the conversion process involves several critical intermediate stages to bridge the gap between raw text and visual data. Accordingly, the following section reviews the literature regarding the preprocessing of text documents, establishing the requisite computational steps that precede this analysis.

2.2 Data Pre-Processing

2.2.1 Data Cleaning and tokenisation

In the context of individual social media platforms, they have their platform specific protocols. Twitter [56], officially known as X, a micro blogging platform, at the beginning had a 140-character restriction, which means no tweet can have more than 140 characters, but after 2017 this has doubled to 280 characters[57]. Platforms provide their APIs (Application Programming Interface) to gather user comments, posts etc... which can come bundled with other information like timestamps of the post, comment time, location information etc... Most of the text data gathered from such internet forums contains many items which are not part of the main text or are not significant in Sentiment Analysis, these are referred to as the garbage in the data.

The garbage can include Usernames preceded by "@" symbol, Hashtags followed by the "#" symbol, URL hyperlinks to other web pages, and emoticons which are emotion icons representing facial expressions, repeated letters or any other irrelevant item [58, 59]. Either these items are completely removed or, in some cases, are partially removed. For instance, removing just the "@" or the "#" symbol instead of the whole word depends on the task at hand, like in a popular Airline dataset [60] where tweets are linked with respective airlines using the "@" symbol and keeping the name of the airline can be crucial in this case.

Tokenisation is the process of dividing the text into a token which can be a word, phrase or complete sentence and these tokens act as the inputs for further analysis steps [61, 62, 63].

2.2.2 Text normalisation

After removing the clutter from text, some pre-processing is generally performed known as text Normalisation. The popular processing techniques include segmentation [63], Stop word removal, Stemming, Lemmatization, Lower

case conversion, rare word or low frequency word removal[62, 63], etc. Some pre-processing techniques are not very popular but are worth mentioning like Word-sense disambiguation [62] and document indexing [64].

In stemming the tokens are reverted to their base words called Stems [61, 62, 64, 65], for example the word 'convert' is the base word for converting, conversions, converted etc...

Lemmatization is the process of trimming the inflectional endings, mostly with the use of vocabulary and morphological analysis of the words, and form Lemmas [65, 66, 67], for example "car", "cars", "car's", "cars' " all have same lemma "car". Lemmas and Stems prove very beneficial in the process of efficiently mapping different word instances in similar categories because they share the same lemma or stem.

Stop words removal is for removing common words like "I", "you", "me", "at", "the", "in", "of", etc to reduce the token size significantly [61, 62, 63, 64]. NLTK (Natural language toolkit) library is a provided package to detect all the stop words in the text corpus [68].

2.2.3 Text vectorisation and feature extraction

Text vectorization, or text representation, is the fundamental process of converting raw textual data into a numerical format that machine learning algorithms can interpret and process [52]. In this context, tokens (such as characters, words, phrases, etc) extracted from a text corpus are treated as features [15]. The presence or characteristic value (like frequency or weight) of each token in a specific document is mapped to a dimension in a vector. By aggregating these values, a fixed-length numerical vector is formed to represent the entire text document [69]. These text-based vectors can encode the semantic and syntactic properties of the language, enabling algorithms to perform tasks like classification and clustering. Techniques within the text pre-processing stage, such as stop word removal and stemming, play a critical role in reducing the overall number of tokens, thereby helping to create more compact and efficient vectors [52].

In this work, words are tokenisation is used as it is most popular tokenisation method in practice and availability of pretrained models' weights using vectors using word tokens.

2.2.3.1 Bag of words (BOW):

This is a classic method for text representation in early text analysis systems [20]. **BOW** is how it is mostly addressed and is the simplest and classic way of representing text tokens for text classification, information retrieval or other NLP tasks.

Count Vectorizer: The Count Vectorizer is one of the simplest and most foundational approaches to generating a BOW representation of text [52]. This method creates a vector for each document based on the raw frequency count of all tokens (typically words) present in the entire corpus (collection of documents) [69]. The resulting vector's dimension is equal to the size of the unique vocabulary in the corpus. Since these vectors can be thousands of dimensions long and consist of mostly zeros, indicating the word is not used, (a characteristic known as high-dimensionality and sparsity), a threshold is often applied to obtain a more manageable, fixed-length feature set [69]. For instance, using a vocabulary comprising only the 500 most frequent words that occur across the corpus effectively reduces the feature space, improving computational efficiency while retaining the most discriminative tokens.

TF and IDF: Term Frequency-Inverse Document Frequency (TF-IDF) is a weighting scheme used to measure the significance of a word token in a text document's BOW representation [69]. Unlike simple count methods, TF-IDF reflects how relevant a term is to a document within a large corpus. The Term Frequency (TF) component measures how often a word token appears within a specific document. This is often calculated as the raw count of the term within the document, or a normalized version where the term count is divided by the total number of tokens in that document [69]. The Inverse Document Frequency (IDF) component measures the general importance of a term across the entire corpus [70]. Terms that appear frequently across many documents (like common stop words) are penalized, while unique terms are weighted higher. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term. While TF and IDF can theoretically be used separately to form different feature vectors, they are typically multiplied together to form the final, combined TF-IDF value for a given term in a document [71]. This combined score forms the features of the document vector.

2.2.3.2 N-gram:

A shortcoming of BOW is that the information related to the structure of the sentence is lost. The order of the tokens in the text is not given any significance, so the n-gram model is an example of restoring some text order

information within a text. N in N-gram depicts the consecutive token occurrence which is used to form the text vectors. For instance, a bi-gram model will store every pair of the token as one compared to the use of single token in BOW and similarly, a trigram will store a group of 3 tokens occurring together and so on. A special case where N = 1 is equivalent to the original BOW model.

2.2.3.3 Continuous Bag of words

Continuous Bag of Word (**CBOW**) utilises a simple neural network model which employed a three-layer architecture including input layer, hidden layer and output layer to learn this representation. The model is provided with a few words before and after the word to be predicted depending on the parameter context window. The model adjusts its weights during training by comparing the predicted word with the actual target word and using backpropagation to minimize the error, thus improving the word representation. **Word2Vec** by Google uses CBOW to generate word embeddings [72].

2.2.3.4 Word Embeddings

Even though the N-gram model attempts to restore the spatial information to some extent, the vectors formed are still inefficient because of sparse representations and there being no word order information depicted in the final vectors formed. An automatic feature learning method, known as distributed vectors or word embeddings, maps each token in a vocabulary to an N-dimensional vector. This approach is based on the distributional hypothesis, which states that words appearing in similar contexts tend to share similar meanings. John Firth describes it as "you shall know a word by the company it keeps" [73].

Word embeddings are word representations in a vector space where semantically similar words are closer in the vector space, for example, 'apple' and 'banana' will be closed in the vector space then the word 'justice'. Consequently, these vectors inherit characteristics from a word's neighbouring words, effectively capturing the similarity between words. These vectors dimensions are not solely dependent on the topmost frequent tokens; thus, the same dimension word vectors can represent 10,000 vocabulary words or 40,000 depending on how dense a representation one requires.

Global Word Embeddings: When word Embeddings are created, it takes into consideration the context of the word in which the word appears but when the finalised vectors are generated, and used for application, they remain largely fixed. Some popular pre-trained word representations are available that one can use without needing to train every time. Such representations are obtained over a big corpus of data which, if required, can be fine-tuned to a task during training. Glove, Fast Text, Word2Vec, Wang2vec are examples of such pre-trained word representations [74].

Dynamic Word Embeddings: The above vectors are quite static and not very time-sensitive i.e., they cannot capture any word evolution depicting social trends of human history. Dynamic Word embeddings are word vectors developed using a dynamic statistical model and learn time aware word vector representation [75]. An easy way of understanding this is via an example. The name of the US President can change after every tenure, so word embeddings can progressively increase the association between the phrase 'US President' and 'Biden' and reduces its earlier association with the word 'Trump' around the time of the 2020 US presidential election.

The word embeddings come under the heading of continuous representation of text, where text is mapped from a corpus to the vectors directly independent of the task. What this means is that these vectors can be created for a large language corpus and then any task which involves that language can use these vectors without going through the trouble of generating these word embeddings. This is where the big organisations started influencing the NLP research as they had access to large corpora of data in many languages and can create the word embeddings in those languages. There are many methods for generating these continuous representation and huge research exist on this. We will focus of a few to help grasp the rational and concept behind them.

2.2.4 GloVe

GloVe (Global Vectors for Word Representation) was introduced in [51] and are obtained while training a corpus on aggregated global word-word co-occurrence statistics. CBOW effectively captures word semantics and relationships by leveraging local context windows. However, it lacks the ability to utilize global statistical information. To address this, GloVe was introduced by Stanford University NLP research team as a word embedding algorithm based on a global co-occurrence matrix. Each element in the matrix represents how often two words co-occur within a defined context window.

GloVe expands upon Word2Vec by creating a co-occurrence matrix from the corpus and then factorising it to derive word vectors. Like Word2Vec, GloVe also provides pre-trained embeddings in various dimensions (e.g., 100,

200, 300) trained on large datasets. This approach efficiently captures both local and global word relationships.

2.2.5 Two-dimensional (2D) Convolution Neural Networks

The Convolution Neural Network is the foundation of modern computer vision systems. Eyes for a machine are equivalent to cameras that generate three-dimensional vectors. These 3D vectors are analysed using 2D convolution windows called kernels, which slide over the image to perform convolution with the underlying pixel values. These convolution kernels can be compared to simple and complex cells in the human visual cortex described by David Hunter Hubel and Torsten Wiesel in 1959. In 1979, Kuniko Fukushima took inspiration from this human eye model and proposed Neocognitron [76], a multilayer artificial neural network, which consists of multiple types of S-cells and C-cell for different types of feature extraction from images.

One of the first uses of the modern architecture of a multilayer neural network is in LeNet [77] for handwritten character recognition which made auto sorting of mails possible based on the handwritten address on it. It utilised gradient-based learning with back propagation, which can synthesize complex decision surfaces, with minimal processing, for classifying high dimensional patterns. The technology was progressing but the conventional machine learning-based models, with extensive feature engineering, were performing better in computer vision tasks, like ImageNet than the ANN models. 2012 was the landmark period that made deep CNN the industry favourite in solving image classification tasks with the advent of AlexNet [78], which performed best in the ILSVRC-2012 competition. AlexNet has 7-layer deep network with one 5x5 convolution and two 3x3 convolutions along with few max-pooling layers and finally 2 fully connected dense layers. It incorporated distributed GPU training by utilising 2 GTX 580 GPUs with 3GB memory each. This is because the ImageNet data has 1.2 million training examples, which were too big for one GPU.

Why CNN took more than 20 years to trend? The answer is the use of a nonlinear activation function, ReLU, and a dropout layer as regularization, which enabled better gradient propagation in a deep neural network and prevent overfitting the training data by randomly removing neurons to force network to learn better perspective for new input data. After AlexNet, networks with even deeper layers started performing better on the ImageNet competition every year, for example, ZF in 2013, VGGNet [79] in 2014, GoogleNet [80] in 2014, ResNet [23] in 2015, GoogleNet-v4 in 2016 etc.

ResNet is short for Residual Network, and it was the first architecture that surpassed the human performance on the dataset. This Network is a deep 2D CNN architecture with solves the inherent problems of going deeper in a convolution neural network i.e., the Vanishing gradient problem. The architecture made it possible to train a very deep neural network without increasing the error propagation which earlier restricted the maximum depth of the network. The network has been adopted in Sentiment Analysis but again a 1D CNN layer replaced the 2D CNN layers as used in the original network [81]. The brilliant feature about ResNet is its Scaling up capabilities which are limited by the Image Resolution or Hardware available to run the network. Based on the resource availability different depths of the ResNet can be achieved, for example, from ResNet-18 to ResNet1202.

Since then, many architectures have adopted these concepts and produced different architectures. The improvements were made by experimenting with various arrangements in a residual block, like skip connections, network scaling etc. DenseNet[82], ResNet-Rs[83], EfficientNet[84] are a few examples of such architectures.

This review of 2D CNN evolution, particularly the architectural breakthrough of ResNet, establishes the theoretical justification for Research Question 1, which investigates whether successful image classifiers can be repurposed for textual analysis. By demonstrating that architectures like ResNet have solved the vanishing gradient problem and achieved superhuman performance in visual recognition, this section validates the choice of using Deep 2D CNNs as the primary engine for the proposed Textual Image classification experiments. Furthermore, the discussion regarding the trade-off between network depth (e.g., ResNet-18 vs. ResNet-1202) and computational resources directly supports Research Question 4 (I), providing the necessary background to evaluate how vision model complexity impacts both the performance and sustainability of the classification task.

2.2.6 Forming images from text

As delineated earlier in this chapter, deep neural architectures in computer vision also uniquely integrate feature extraction and classification within a single, end-to-end framework. To adapt this architectural advantage of computer vision to the domain of natural language processing, raw textual data must be transformed into a compatible visual representation that satisfies the input requirements of these image-centric models. This transformation is critical, as it enables the extraction of spatial features from the generated Textual Images, a process distinct from traditional sequential processing of text.

The basic unit of an image is a pixel, and the collection of pixels forms an image. More pixels mean better resolution.

An RGB or a colour image's pixel encodes 3 different values, one corresponding to each colour channel. Each pixel's channel numerical value combined represents the corresponding colour of the pixel while displaying it. The resolution of an image is $H \times W \times C$ where 'H' is height, 'W' is the width and 'C' is the number of channels. An image with only one colour channel is known as a grey scale image which is a monochromatic image. The value of each pixel in an image is between $0 - 255$ representing the intensity of colour.

The images are loaded in a neural network with a default shape (number of samples, image height, image width, number of channels) which is a four-dimensional tensor. The tensor shape of 1000 grey scale images with image resolution $48 \times 48 \times 1$ will be $(1000,\ 48,\ 48,\ 1)$ where every individual image is a 3D tensor. A 2D CNN requires a 3D tensor, like the shape of an image, to perform the 2D convolution operation.

Word-Embeddings, discussed in a previous section, is a high dimensional word representations but when represented in a matrix form, each word vector shape is $1 \times Embedding_size$ where $Embedding_size$ is the number of higher dimensions chosen to represent the words. Every word in the sentence has its word vector and collectively they form a 2D matrix shape $N \times Embedding_size$, where 'N' is the number of words in the text document. One can simply reshape this vector to $N \times Embedding_sizex1$ (number of elements remain same), then N can be considered the height of the image, embedding_size can become the width of the image, and 1 can represent a single channel, thus forming a grey scale image equivalent tensor.

Note that processing such a text image with a 2D CNN, will have inherent drawbacks because, unlike a real image, the pixel of such a text image processed using a limited 2D convolution window will not take into consideration the entire dimensional encoding of a word. That is why 1D CNN are popularly used in NLP, as in 1D CNN the window size is $Window_heightxEmbedding_Size$ where $Window_height$ determines the number of Word-Embeddings under observation in each convolution window. The convolution window for 1D CNN is 2D but unlike 2D CNN, the movement of the 1D CNN windows is restricted to one direction, Hence the name 1D CNN. Therefore, there is a need to create some clever method to encode text information into an image to then learn the feature dependencies using 2D CNNs.

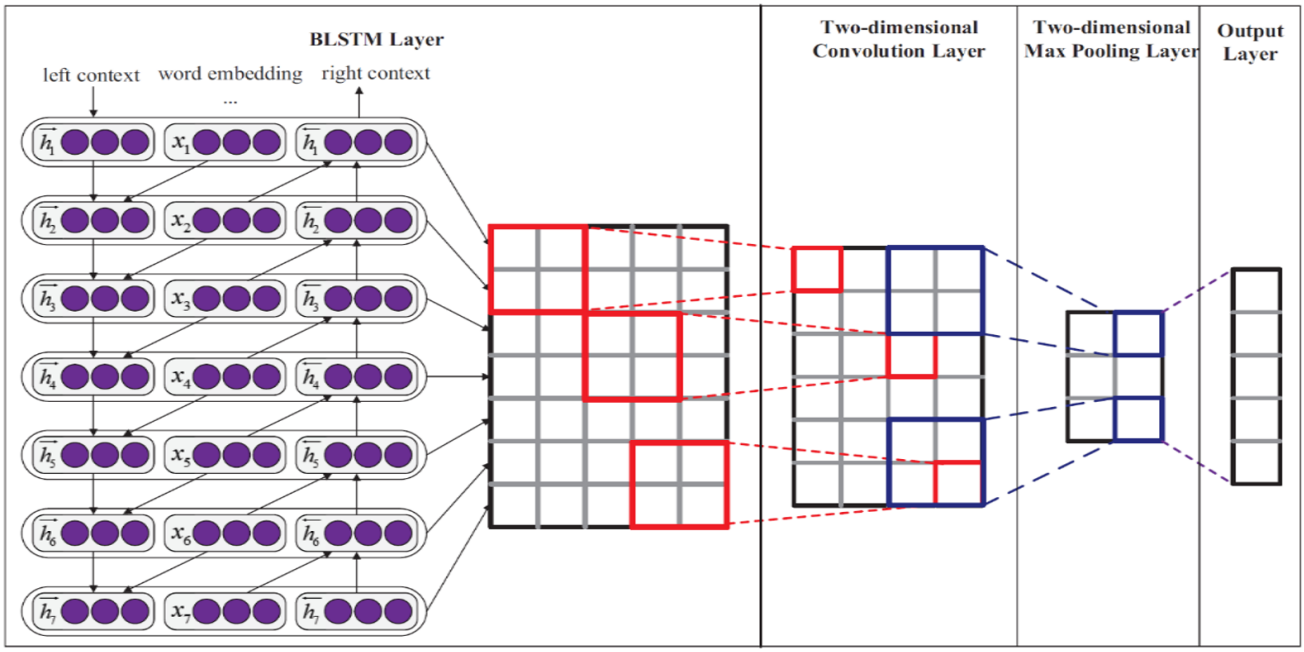


Figure 2.2: Depiction of a 7-word sentence with embedding size 3, processed by a BLSTM layer with 5 hidden units, followed by a 2x2 convolution, max-pooling operation and Flatten layer (source [27]). The five outputs in the first layer were chosen as an example and are not correlated with the five BLSTM hidden units.

The architecture in Figure 2.2 is a BLSTM-2DCNN [27] and is an excellent example of creating quality text images for a 2D CNN to extract features from. The name of the architecture is self-descriptive and represents different sequential layers in the network that is, it starts with BLSTM layer, which is followed by 2D convolution layer. The BLSTM in BLSTM-2DCNN captures word dependencies based on the words following and trailing the word under observation within the input text, which is in the form of word embeddings. It is very similar to RNN based architectures capturing temporal dependencies. As in most computer vision tasks, the convolution layers are followed by a pooling layer which helps accumulate features from the convolution maps, so a 2D max-pooling layer is used after the convolution layer for the same reason.

In Figure 2.2, text vector input 'x' represents a 7×3 matrix, where 7 is the length of the sentence and 3 is the

Embedding_size of each word vector, which is fed to a Bidirectional LSTM with 5 hidden units/cells. Note that the Embedding size would really be much larger than 3, but 3 is picked for practical illustration purposes. The outputs of both the LSTMs (forward and backward) have a shape of 1×5 at each timestep (timestep is equivalent to the number of words in a sentence), and these 1×5 tensors from both are added elementwise. Therefore, in the example after 7 timesteps a 7×5 output tensor is obtained. This 7×5 tensor is reshaped into $7 \times 5 \times 1$ to represent it as a grey scale image (this step is not depicted in the figure). A convolution operation with a 2×2 kernel is performed on this tensor, which, by sliding over the image with a step of 1, reduces the dimensions to $6 \times 4 \times 1$ and then a 2×2 max pooling operation further shrinks the output to $3 \times 2 \times 1$. Finally, the output layer contains 5 units providing 5 different probabilities concerning each class in the example classification task.

The above architecture gives the inspiration to create similar methods of Image creation, for example, the use of GRUs [24]. GRUs are effective over LSTMs in processing some time-dependent data. Just replacing LSTM cells with GRUs can create a new, different type of image which might aid in text classification or, indeed, other RNN inspired networks can also be used to replace LSTMs for the same purpose. Though, replacing LSTMs with another RNN architecture is a good idea they still will be all restricted by the inherent property of all RNNs, that is, not being able to remember very long-range dependencies (this is of particular importance when words in a document can be in hundreds, like in a News article etc). Other approaches can emphasise incorporating some feature engineering techniques like LSA along with word embeddings [72] [72, 21]. All together many different approaches for creating images should also be explored based on the current research in the NLP for text-based analysis.

Other, newer, architectures can prove beneficial in constructing new methods to form images using text-based vectors. In a recent paper [85], SA-SDCNN (sparse attention based separable dilated convolutional neural network) is proposed, which incorporates multichannel embedding for targeted sentiment analysis. Multichannel embeddings are formed by merging word2vec, Glove and Sentiment embedding into a deeper tensor which helped in improving the sentiment extraction. This Idea can be explored further by, say, in constructing an equivalent of a multichannel text image (like a colour image) instead of a grey scale text Image used in [27].

Most neural network models for text processing give equal importance to every word, but for sentiment analysis, few words hold valuable emotional information, so [85] proposed attention networks for semantic relation extraction, which make full use of word embeddings, part-of-speech tag embeddings and position embeddings. Taking the above inspiration while creating new images may encode better contextual information into the text-based images.

The hypothesis that integrating semantic and positional data could enhance the fidelity of Textual Images directly motivates the second research question. Specifically, this inquiry aims to empirically determine whether determining the impact of enriching Textual Images with contextualised textual information yields tangible improvements in Textual Image classification performance. Consequently, to establish the theoretical framework for this investigation, the following section reviews the evolution and mechanics of contextualised embeddings.

2.3 Contextualized Word representation

The concept of contextualised word representation originates from the integration of positional and semantic information into the embedding process to achieve a more information-rich representation. Unlike earlier text-representation approaches, this method produces dynamic embeddings where the same word is assigned a distinct vector representation, strictly contingent upon its specific placement within the text and the semantic influence of its neighbouring words in that instance. In practice, initial fixed word embeddings are processed by a deep neural architecture to generate these contextualised representations within the network's deeper layers analogous to the feature extraction phase in traditional machine learning before being utilised for downstream natural language processing tasks.

Several applications emerged in text analysis addressing sophisticated language tasks like machine translation, question answering task, language models which leverage deep learning models to achieve contextualised word representations. Pairing word-representation techniques with deep learning models trained for high-level language tasks enabled the deep neural model to learn hidden representation which enabled contextualisation in otherwise static word embeddings. This means that, whereas previously a vector representation for word token "Bank" had the same value regardless of the number of times and where it appears in the document but when these vectors were used in a deep learning model employed for a high-level language task, they started gaining contextual awareness. In short, the word token "Bank" will have different numerical representations throughout depending on the neighbouring tokens.

Early pioneering models in this area include Context2Vec, which used bidirectional LSTMs to capture the full sentential context of a word [86], and CoVe (Contextualized Word Vectors), which leveraged the internal states of a trained Neural Machine Translation encoder [87]. A major evolution in this field was the introduction of ELMo

(Embeddings from Language Models), which generated "deep" representations by learning a weighted combination of the internal hidden states from a bidirectional language model [88]. This approach, along with the rise of attention mechanisms, reinforced the idea that models must pay attention to neighbouring words to achieve true contextual awareness. In these modern architectures, the output representation of a word from the model's hidden layers is considered its contextualized word embedding.

2.3.1 Attention mechanism

To understand the contextualised embedding, we need to understand the use of Attention mechanisms in machine learning. Let's begin with a look at how attention works in a biological setting. The human sensory system is complex and relays information regarding internal and environmental conditions and stimuli. Based on this overabundance of stimuli to the sensory system, attention is used to concentrate the consciousness on concise stimuli and aid making decisions. For simple understanding, we can refer to the example of the primate visual system provided in [89]. The visual system receives information at 10^8 bits per second, which is significantly higher than the processing capability of the human brain. But not all visual stimuli are alike, and attention focusses or concentrates the consciousness on the object of interest, like prey for a predator. Attention dictates the experiences of a conscious being.

Attention in a biological setting, according to William James (father of American psychology) is based on non-volitional and volitional cues [90]. The nonvolitional cues depend on the saliency of the object in an environment and the attention is directed involuntarily. Whereas the volitional cue is based on cognitive control. Figure 2.3 depicts an environment with five items: a newspaper, a document, a red coffee cup, a notepad, and a book. Under nonvolitional cues the eye is focused on the red coffee cup as it has a distinct colour and attracts involuntary attention whereas to focus on a book, attention will put the spotlight on the book, and the eye muscles will move under cognitive control.



Figure 2.3: Example of nonvolitional (left) and volitional (right) cues in an environment (source [89]).

Like the working of attention cues in a biological setting, several frameworks in machine learning have been developed for designing an attention function. While several types of attention mechanisms are proposed, one of the most common and influential is formulated as a function that maps a Query and a set of Key-Value pairs to an output. In this formulation, according to [91], "An attention function can be described as mapping a query and a set of key-value pair to an output where query, key and value and output are all vectors". The keys are the nonvolitional input cues from the environment. The query (volitional cue) can interact with keys (nonvolitional cues) which sets the bias for the values (sensory inputs).

Before the Transformer model, attention was famously introduced to solve a bottleneck in RNN-based sequence-to-sequence (Seq2Seq) models [55]. The original Seq2Seq architecture followed an encoder-decoder design, where the entire input sequence (e.g., a French sentence) was compressed into a single, fixed-length "context vector" by the encoder block, which was then passed to the decoder to generate the output (e.g., the English translation). This fixed-length vector created a bottleneck, as it struggled to retain all the information from a long input sequence.

Two key attention mechanisms were developed to solve this:

Additive Attention: This mechanism, introduced by Bahdanau et al. [92], allows the decoder to look back at all hidden states of the input sequence at each output step. It uses a small feed-forward neural network (which "adds" the state vectors) to compute an alignment score, which determines how much "attention" to pay to each input word when generating the current output word.

Multiplicative Attention: Luong et al. [93] proposed a simpler and often more computationally efficient alternative. Instead of a feed-forward network, it uses dot-products (multiplication) between the decoder's current state and the encoder states to calculate the alignment scores.

2.3.2 Transformer based text models

The Transformer based models have been pre-trained on a huge corpus of data, so they have learned language representation through unsupervised training or self-supervised pre-trainings. These models can produce contextualised embeddings i.e., context-aware vector representation of the words in a document where the context not only depend on the neighbouring words but also on the later sentences due to efficient learning of long-term dependencies. These contextualised embeddings can then be used to form contextualised Textual Images.

The Transformer based models become a better alternative for complex RNN-based language models as these were highly parallelisable and can compute the whole sequence of text in a single time step. Although Transformers were introduced to deal with sequence-to-sequence modelling tasks, they can be used for text classification tasks like Sentiment Analysis as well.

A modified Transformer model containing only a decoder was introduced [94] which could generate Wikipedia articles and showcased that the decoder Transformer model can attend to a longer input sequence than an encoder-decoder model. Using this decoder-only Transformer model, [95] demonstrates the importance of generative pre-training of the language model. The pre-trained model can then be fine-tuned to competitively perform various NLP downstream tasks. This was the first Transformer based language model by OpenAI, and the model weights were publicly available under the name GPT (Generative Pre-trained Transformer). Later, the team at OpenAI introduced GPT-2 model with 1.5 billion parameters [96] which performed best in various language modelling tasks even in a zero-shot setting (which means the model did not undergo huge, supervised training on tasks-specific datasets).

On the other side, the Bidirectional Encoder Representation of Transformer (BERT) [28] uses only the encoder part of the Transformer. They tried to overcome the shortcomings of the existing transformer model which learned only unidirectionally representation as they have constrained self-attention and quote "Such restrictions are sub-optimal for sentence-level tasks and could be very harmful when applying fine-tuning-based approaches to token-level tasks". BERT has bidirectional self-attention and incorporates representation learning using two different self-supervised pre-training tasks which are the Masked Language Modelling (MLM) and Next Sentence Prediction (NSP).

In the MLM pre-training task, the masked tokens Ids are predicted from a sentence with randomly masked tokens (every word in the vocabulary is represented with a unique token id) whereas, in NSP, the model pre-trains on a task where it predicts the probability of a succeeding sentence in a sentence pair being from the same document. BERT has two model configurations $\text{BERT}_{\text{base}}$ (L=12, H=768, A=12, 110million parameter) and $\text{BERT}_{\text{large}}$ (L=24, H=1024, A=16, 340 million parameters) where L is the number of transformer blocks, H is hidden units, and A is the number of self-attention units. At the time of publishing, $\text{BERT}_{\text{large}}$ outperformed eleven NLP tasks including the Sentiment Analysis dataset. Many models have since been proposed that use the BERT architecture [97, 98, 99, 100] with different pretraining strategies, optimisation approaches, parameter Tuning, model distillation techniques and a lighter variant respectively.

The pre-trained BERT variants are available on Hugging face which are contributed to the platform by the company itself and other community contributors [101].

2.3.3 Attention in Computer Vision

Transformers, after revolutionising NLP, have successfully challenged the dominance of CNNs in computer vision. The Vision Transformer (ViT) was a pioneering model that applied a pure transformer architecture to sequences of image patches for classification [102]. This was enhanced by models like the Swin Transformer, which introduced a hierarchical structure and shifted window-based attention for greater computational efficiency and performance [103]. The DETR (DEtection TRansformer) model applied this architecture to object detection, successfully framing it as an end-to-end set prediction problem [104]. More recent architectures, like NaViT, have further increased flexibility by enabling transformers to process images at their native resolutions and aspect ratios, avoiding the traditional resizing step [105].

2.4 Systematic literature review of Transformer adoption in documents level sentiment analysis.

In this research, which extracts sentiment from a textual document using a Transformer to generate contextualised Textual Images, a systematic literature review was conducted on Transformer models used for sentiment analysis. In future, the Textual Image classification may be applied to more dedicated problems, so then it would be more appropriate to look at research focusing entirely on those areas. To keep the problem more general, this research

aims to look only at the document level transformer adoption.

A literature review is performed on the published literature on document level sentiment analysis using Transformer based models around March 2023. The literature review is limited to the document level sentiment analysis to remain within the scope of this work. The criteria for the literature review were to focus on the use of the pre-trained Transformer models used for sentiment analysis via fine-tuning. The reason for focusing on a pre-trained Transformers model is to make use of the intensive self-supervised trainings that most large language model are subjected to during the training phase. This self-supervised pretraining enables these large language models to generalise well for the downstream tasks like text classification. A prominent text classification application is sentiment analysis.

The aim of the literature review was to identify existing Transformer based literature which show potential in making contextualised word embedding more information rich in document level sentiment analysis. The documents found by the literature review could then inform possible extensions or additions to our initial research that might be worth pursuing.

Text analysis heavily relies on Word embeddings, and the richer the word embeddings the better the models perform. Richness of word embeddings is an active area of research. Several methods to incorporate additional information into existing techniques is used in the literature. To understand the use of transformer-based model leveraged to perform sentiment analysis a systematic literature review was conducted. To capture as many literature sources as possible, IEEE Xplore with a search string ((transformer*) AND (sentiment)) was searched. This search string fetched 330 papers in total.

Inclusion criteria:

- Not a comparative study or Literature review
- Not a low resource language
- Only text document sentiment classification

The papers fetched from the search strings then underwent inclusion and exclusion criteria in order to concentrate on the most pertinent papers. The inclusion and exclusion criteria are provided above and below, and these criteria focuses on text sentiment classification on high resource languages and do not include Aspect based classification or low resource languages. The reason for not choosing low resource languages is that majority of the methodology chosen in those papers are usually taken from techniques already used for high resource language or the methodology is specifically tailored to benefit the semantic and syntactic characteristics of the language. Some of the papers were regarding sentiment analysis techniques like aspect-based sentiment Analysis or image-based sentiment analysis. These were also excluded as our aim is to restrict our search specifically to document level sentiment analysis on text data. As the techniques developed here can easily be adopted to other forms of sentiment analysis.

The Inclusion criteria were first applied to the title of the papers, if the title does not reflect the content desired for our study, we dropped the paper from the list. If the title of the paper is ambiguous and it is difficult to take a decision at this stage, then the paper is kept for next step. A total of 85 papers failed to be included based on their title.

The second step is applying the inclusion criteria on the abstract of those papers which have not been dropped in the previous step. Like the previous step, if the abstract of the paper does not fulfil the inclusion criteria, then that paper will be dropped.

Exclusion criteria:

- Model-distillation
- Ensembles
- Optimisation
- No novelty
- Multi-modal networks
- No comparison with transformer model

After step one and two, 221 papers were dropped from the list of papers obtained based on the search string. Next, the exclusion criteria were applied to the rest of the papers. The exclusion criteria details are mentioned above, and they are focussing on excluding papers which have performed model distillation, optimisation or used ensembles.

Most of these models demonstrate no novelty in their methodology and implemented a published techniques for their application on a different applications domain like a model which belonged to the financial or hospitality sector etc... using minor modelling changes as per the dataset. We also encountered many multimodality studies for sentiment analysis where a text data is paired with data from other modes to perform cross modality analysis. In multimodality multiple different forms of inputs signals are used to perform sentiment classification, and the focus of the studies differ from our aim. These inputs can be text, audio, video etc... and these models were not addressing the aim of our literature review, so we also excluded these papers.

To apply the exclusion criteria, we must briefly scan the content of each paper to identify if the paper qualifies any of the exclusion criteria. While skimming through the list of papers, the exclusion criteria were updated as it became clearer that some categories of papers failed to correspond to the aim of the study. During the reading the papers stage, it was noticed that many of them have not included a direct comparison with the transformer model used in their proposed model, so we have also excluded these literatures as it is unclear if their methodology contributed any significant improvement compared to our baseline. In the end, only 10 papers were selected.

The selected papers were organised based on the themes and two different themes were identified. The first theme involved studies in which some lexical methods are used to incorporate more information to the transformer embeddings and the second theme is the use of additional neural layers, or architecture at the end of transformer models, which can add more information to embeddings via back propagation or the clever use of objective functions.

Table 2.1: Systematic Literature review final model comparisons

S.No. and title	**Methodology**	**SA Datasets and Evaluation**	**Conclusion**
1. Sentiment Analysis of Short Informal Text by Tunning Bert-BiLSTM Model [106]	Used pre-trained BERT embedding to get context-aware word embeddings and added a task-specific Bi-LSTM layer over BERT to achieve better sentiment from text. The Bi-LSTM helps to get the knowledge of local as well as global context of a sentence.	SemEval datasets. Evaluation: Recall, F1-Score, Accuracy.	Model outperforms all the previous state-of-the-art models, in analysing short texts and established that the contextual representations obtained from BERT Bi-LSTM layer over BERT, play a substantial role in deeply understanding the sentences.
2. BMT-Net: Broad Multitask Transformer Network for Sentiment Analysis [107]	BMT-Net can learn contextual representation due to its powerful capacity to search for the desirable features in deep and broad spaces. They merge all the task datasets and then shuffle it randomly for preprocessing data. The lexicon encoder first transforms the input data into a sequence of L1 embedding vectors of lexicon representations. Then, the transformer encoder captures the contextual information for each word via self-attention and then comes into being a series of contextual presentations in L2 layers. Simultaneously, the model will select the corresponding neural-network pathway to update the task-specific parameters for the downstream task.	Datasets: SemEval, SST-2. Evaluation: Marcro average F1, Accuracy.	BMT-Net can roundly learn the robust contextual representation due to its capacity of searching the suitable features in both depth and width feature spaces at the same time. Model analysis revealed that a more powerful contextual representation was learned, which contains both deep and broad information learned from tasks.

Continued on next page

S.No. and title	Methodology	SA Datasets and Evaluation	Conclusion
3. Sentiment Analysis of Online product reviews based on SENBERT-CNN [108]	The BERT structure is utilized to represent sentence semantics as text vectors effectively, followed by the extraction of deep sentence features using a CNN. The Bidirectional Encoder Representations from Transformers (BERT) model is fine-tuned on e-commerce reviews to enhance the performance of sentiment analysis tasks. Since BERT has demonstrated excellent results across various NLP tasks and employs a nearly uniform structure, its adaptation to the domain of online product reviews is considered to offer significant benefits for sentiment analysis. SenBERT-CNN is initialized with weights from BERT, pre-trained on general-domain corpora such as English Wikipedia and the Books Corpus. The pre-trained BERT is used to encode review sentences, enabling the model to learn semantic information within and across sentences. SenBERT-CNN is then fine-tuned on a collected dataset of mobile phone reviews, and a Softmax classifier is applied for sentiment analysis.	Binary Data is collected from "JD.com" for smartphones consisting of 8,000 training and 1,600 testing samples. Evaluation: Precision, Recall, F-beta, Accuracy.	The model first uses BERT to perform word vector coding to represent the semantic information of the text and then uses convolutional neural networks to further extract the text features in depth, thus achieve a deep understanding of the model's semantics of text information.
4. Linguistic Knowledge Application to Neuro-Symbolic Transformers in Sentiment Analysis [109]	A SentiLARE model using a neuro symbolic approach is proposed which uses lexical resource Princeton WordNet [110] on text along with their polarity extension for English SentiWordNet [111] and Polish plWordNet + plWordNet Emo. The lexical resources are used to identify part-of-speech using WordNet grouping cognitive similar concepts (synsets). The sentiment polarity for each word is then added as a feature during pre-processing step. Pre-trained HurtBERT model embeddings are selected, a model proposed for abusive language detection task where BERT and LSTM outputs were combined to make prediction. KEPLER technique to fine-tune the model on knowledge Embedding objective and task objective.	Datasets: IMDB, Multiemo/ polemo2.0, GoEmotions, Allegro Reviews. Evaluation: F1-macro.	The results highlighted the difficulties in designing an effective knowledge transfer method compatible with transformers, which function at a high level of abstraction. Most of the neuro-symbolic architectures tested were unsuccessful, as they failed to achieve improvements in F1-macro scores relative to the baseline. The only successful method for incorporating external knowledge involved embedding it directly into the model weights during the pre-training of the transformer blocks.

Continued on next page

S.No. and title	Methodology	SA Datasets and Evaluation	Conclusion
5. Sentiment Classification Based on Part-of-Speech and Self-Attention Mechanism [112]	The POS-Attention mechanism is designed to capture sentimental information. Part-of-speech (POS) tags are integrated into the network as vector representations, and an attention mechanism is applied to weight different parts-of-speech, allowing the model to learn distinct sentimental features. A new Part-of-Speech-based Transformer Attention Network (POS-TAN) is proposed, which combines part-of-speech information with the Self-Attention mechanism. The effectiveness of the model for sentiment classification tasks is validated through experimental results.	Datasets: TSB, Waimai, Weibo, NLPCC 2014, Yelp 2013, Amazon review. Evaluation: Accuracy	The proposed model leverages the Self-Attention mechanism to learn text feature representations while incorporating POS-Attention to capture sentimental information embedded in parts-of-speech. Additionally, Focal Loss is introduced to mitigate the effects of sample imbalance on classification performance. Extensive experiments are conducted on both Chinese and English datasets, and the following observations are made from the results: (1) sentimental information from parts-of-speech is effectively extracted by the model; (2) the combination of POS-Attention and Focal Loss enhances sentiment classification accuracy; (3) the visualization of POS-Attention weights reveals that different parts-of-speech exhibit varying degrees of sentimental intensity, enabling the network to focus more on words with higher weights.
6. Convolutional Transformer with Sentiment-aware Attention for Sentiment Analysis [113]	They propose a multi-window Convolutional Transformer (ConvTransformer) which takes the advantages of both CNN and Transformer for efficient text representation. ConvTransformer has a simplified multi-head multi-layer structure compared with Transformer. In each head, instead of self-attention, convolution operation over the text represented in different sub-spaces is performed to better capture local n-gram features. To effectively capture the n-grams with various lengths, they use multiple convolution window sizes in different convolutional heads, allowing different heads to focus on different n-gram features. They also discard the pooling operation to preserve sequential information in the output of ConvTransformer. Moreover, they propose a sentiment-aware attention mechanism to summarize the sequential output of ConvTransformer and obtain the global text representation by incorporating external knowledge of words' sentiment from SentiWordNet.	Datasets: IMDB, Yelp P. and Yelp F. datasets are constructed from Yelp Dataset 2015 Evaluation: Accuracy	It is more effective in capturing local n-gram features compared with CNN and Transformer and able to preserve the sequential information of texts along with incorporating external knowledge of words' sentiment intensity into the model with SentiWordNet. The proposed sentiment-aware attention mechanism takes each word's sentiment intensity and position information into consideration while obtaining the final text representation.

Continued on next page

S.No. and title	Methodology	SA Datasets and Evaluation	Conclusion
7. Long-Text Sentiment Analysis Based on Semantic Graph [114]	To address the practicality of extracting deep information from long-text corpora, a sentiment analysis model based on a semantic graph is proposed. Initially, two two-layer bidirectional LSTMs are utilized to extract word features and sentence features from the long text. These extracted features are then used to construct a semantic graph that effectively represents the original corpus while preserving all emotional information. Finally, the semantic graph is input into BERT, which leverages its strong feature extraction capabilities, for sentiment analysis.	Datasets: Bag of Words Meets Bags of Popcorn (BoWM-BoP) and online shopping 10 cat's datasets Evaluation: Accuracy	SG-BERT, when compared to existing models, is found to achieve the highest performance on corpora with lengths ranging from 500 to 1000 words across two different datasets. However, as the classification learning process is fully inherited from BERT, limitations are encountered in handling ultra-long text. When the semantic graph, which encapsulates all valid emotional information, exceeds its capacity, SG-BERT is unable to produce satisfactory results for ultra-long text.
8. Sentiment-Aware Transformer Using Joint Training [115]	A Transformer-based model for sentiment analysis is proposed. An auxiliary task is constructed based on the lexicon to predict the sentiment polarity of individual words within the target sentence. These two tasks, the sentence sentiment classification task and the word sentiment classification task are trained jointly to optimise the model. In essence, sentiment information is encoded into the model, creating a "sentiment-aware" Transformer model. Additionally, the part of speech of words in the target sentence is used to provide additional sentiment information assisting is extracting the overall sentiment polarity of a text document.	Datasets: SST-2, Movie Review, SemEval: IMDB, Evaluation: Accuracy	The auxiliary task introduced, utilizing a sentiment lexicon, enables the model to predict both word sentiment polarity and sentence sentiment simultaneously. The model is optimised using a hybrid loss function and a two-stage training strategy. Improved performance is demonstrated by the proposed model compared to ordinary Transformer-based sentiment analysis models on multiple public sentiment analysis datasets.

Continued on next page

S.No. and title	Methodology	SA Datasets and Evaluation	Conclusion
9. Sentiment Analysis of Movie Reviews based on Pretraining and Dual Branch Coding [116]	A Dual Branch Feature Enhanced Model (DBN-Ro) is proposed to better analyse the sentiment tendencies of text. The analytical and predictive capabilities are significantly improved through the enhancement of word embedding representation and feature extraction. RoBERTa is utilized instead of traditional word vector techniques to obtain vector representations of text. This approach allows the different semantics of words in varying contexts to be captured effectively thus enhancing the representational ability of text vectors. A dual-channel and dual-structure coding approach is applied to text processing, addressing the limitations of traditional single-channel and single-structure models to improve the model's feature extraction capability and enriches the extracted feature information. Temporal Convolutional Networks (TCN) are employed instead of CNN to effectively capture the temporal information in the text, enabling the extraction of both local and global text information. CNN layer is used for dimensionality reduction of the feature vectors, optimising the feature representation while maintaining critical information.	Datasets: Movie Review, IMDB. Evaluation: Accuracy	It has been observed that commonly used sentiment analysis methods rely on static word vector techniques, such as Word2Vec, for obtaining vector representations of text. Additionally, these methods often utilize single-channel structures based on RNN or CNN models to extract feature information from text vectors. To address these limitations, DBN-Ro is proposed in this paper, building upon existing research from two perspectives: improving the vector representation of text and enhancing feature extraction capabilities.
10. Towards Improved Deep Contextual Embedding for the identification of Irony and Sarcasm [117]	An end-to-end methodology is proposed to enhance the quality of social media text while effectively addressing previously defined language ambiguities. The focus of the research is placed on: (i) capturing words with opposite valence; (ii) representing the complex characteristics of word usage, including both semantic and syntactic aspects; (iii) identifying the sentiments associated with words; and (iv) detecting out-of-vocabulary (OOV) words. The model is designed with two primary layers: (a) a Representation layer and (b) a Bidirectional Long Short-Term Memory (BiLSTM) with an attention layer. The framework of the model is structured into four components: the Input (corpus), the Representation layer, the BiLSTM with Attention layer, and the Output layer.	Datasets: SemEval Evaluation: Accuracy	An end-to-end methodology, T-DICE, is proposed, utilizing the pre-trained ALBERT model in combination with other embeddings and an attention-based BiLSTM to address figurative language (e.g., irony and sarcasm) in social media text. The network is compared against all, to the best of their knowledge, published methodologies using three benchmark datasets. Language ambiguities, including polysemy, semantics, syntax, and word sentiment within the context of posts, are addressed by learning representations from six different embeddings. Furthermore, the attention-based BiLSTM enhances performance by focusing on key words in the text.

From exploring the final set of papers, refer Table 2.1, it is observed that the performance in sentiment polarity identification boost is achieved either by combining features extracted by the Transformer based model with previously existing feature extractors or by leveraging the lexical resource to incorporate more semantic information. More information is contributed to the contextualised word embeddings by complementing the transformer architecture with additional networks like Convolution based neural feature extraction [108, 113, 116] and Recurrent based neural feature extraction [106, 114, 116, 109]. Most of the models utilised pre-trained transformer-based models like BERT [106, 109, 114], RoBERTa [116] and HurtBERT [109] in their proposed methodology and some

have a modified Transformer architecture which required training the network from scratch [107]. The lexical information is also a popular method to enrich contextualised word embeddings by adding more domain knowledge using methods like lexicon embedding [107], part-of-speech-tagging [113, 112, 115, 117] and SentiWordNet polarity [109, 115].

2.4.1 Lexical resources

Part-of-speech (POS) is a text pre-processing step that involves annotating each word in a document with its corresponding grammatical category, such as noun, verb, adjective, adverb, or pronoun [52]. As a pre-processing task, it provides crucial syntactic context. For example, knowing a word's POS tag is essential for effective lemmatization, as it allows the algorithm to correctly reduce different forms of a word to its root. It also enables the filtering of specific word types, such as removing prepositions or conjunctions, to focus on content-bearing words [52]. POS tagging comes as a popular lexical resource that can be used to enhance the richness of the word embeddings [112, 118, 119, 107].

2.4.2 Why BERT?

The widespread adoption of the BERT model in the NLP community is attributable to several factors. The primary one is pre-training and fine-tuning. BERT models are pre-trained on massive unlabelled text corpora, allowing them to learn deep, bidirectional, linguistic representations. The original authors released these pre-trained model weights for various configurations (e.g., $\text{BERT}_{\text{Base}}$ and $\text{BERT}_{\text{Large}}$), which can then be efficiently fine-tuned for a particular task with minimal architectural changes to achieve state-of-the-art results on a wide array of downstream tasks, such as text classification and question answering [28]. Its ubiquity is evidenced by its sustained popularity on open-source repositories like Hugging Face, indicating deep adoption by the academic research community beyond just industry practitioners. Moreover, the research community actively contributes specialised, fine-tuned models for specific tasks and domains, conserving computational resources for other researchers.

This accessibility is critical, as it is vastly more computationally efficient to fine-tune an existing model than to undertake the prohibitive costs of pre-training from scratch. Training a large Transformer model from scratch requires substantial hardware investment, such as large-scale GPU or TPU clusters, and significant technical expertise resources that are often inaccessible to smaller academic research teams.

2.5 Sustainability

Sustainability is fundamentally defined as development that meets current needs without compromising the ability of future generations to meet theirs [120]. This concept encapsulates principles of interconnectedness, resilience, resource efficiency, and equity [120]. Interconnectedness emphasizes a holistic approach to environmental, social, and economic challenges. Resilience concerns developing systems that can adapt to shocks, while resource efficiency focuses on minimizing waste. Equity and justice ensure the benefits and burdens of development are distributed fairly across generations.

In deep learning, sustainability is associated with the environmental, social, and economic costs of model training and deployment. The goal is to ensure that advancements in AI do not come at the expense of long-term environmental health or ethical considerations [121]. While ML is heavily leveraged to promote sustainability in other industries, the field must also address its own footprint. I considered sustainability a core design principle and adopted specific considerations.

Environmental sustainability in ML is primarily concerned with the high computational requirements and associated energy consumption of training large models [121]. To address this research prioritised training efficiency by prioritising pretrained deep neural model instead of training for scratch and training-process optimisations. Specifically, mixed-precision model training can be adopted. This technique uses 16-bit floating-point (FP16) numbers for computation where possible, rather than the 32-bit (FP32) standard. This decision can provide three key sustainability benefits. Reduced Memory Usage by reducing the memory footprint by nearly 50%, allowing for larger batch sizes on the same hardware. This reduced memory access, combined with the use of Tensor Cores on modern GPUs (like the Tesla V100), accelerates training and inference speeds.

2.6 Literature Review in Relation to the Research Questions

The preceding sections reviewed the technical foundations of this thesis. This section synthesises the literature most directly related to RQ1–RQ4 and explains how it motivates the experimental programme in Chapters 4–6. The experimental findings are presented in the later chapters.

2.6.1 RQ1 and Chapter 4: Deep 2D-CNNs for Textual Images

Deep Convolutional Neural Networks (CNNs) have shown promise in sentiment classification, but their application in this domain has often been limited. Previous approaches primarily utilised one-dimensional (1D) convolutions, as seen in the work by [81, 122]. While effective, 1D convolutions extract features by sliding kernels in a single direction, which can restrict their ability to discern more complex textual patterns. In particular, Liu [122] identified performance-degradation challenges when extending 1D-CNNs with additional residual blocks for sentiment analysis.

Two-dimensional convolution has also been explored for text classification. The BLSTM-2DCNN model of Zhou et al. [27] demonstrated that BLSTM outputs can be treated as two-dimensional representations and classified with 2D convolution and pooling. Related work has also used 2D CNNs for sentiment analysis [123]; however, the convolutional depth in these approaches is shallow and fixed. This literature therefore motivates RQ1 and Chapter 4, which investigate whether deeper ResNet-based 2D-CNNs can replace the shallow visual component of the BLSTM-2DCNN pipeline for Textual Image sentiment classification.

2.6.2 RQ2 and Chapter 5: Contextualised Textual Images

Attention mechanisms improved sequence modelling in neural NLP [92], while the Transformer architecture established self-attention as a central mechanism for contextual language representation [91]. BERT subsequently demonstrated how Transformer encoders can be pre-trained and fine-tuned for downstream NLP tasks [28]. Its contextualised embeddings provide a potential means of generating more informative Textual Images.

Traditional recurrent networks, including BLSTM, process data sequentially and can struggle to capture long-term dependencies and global context. Transformer architectures, notably BERT, can provide richer contextual representations by modelling global semantic information and allowing parallel processing. The literature therefore motivates RQ2 and Chapter 5, which examine whether contextualised BERT embeddings improve Textual Image classification and whether the additional BLSTM stage remains beneficial across dataset sizes and text lengths.

2.6.3 RQ3 and Chapter 6: Pre-trained Vision Backbones and Efficiency

Training deep neural networks from scratch is computationally intensive because large architectures require extensive training data, time and hardware resources. Self-supervised pre-training provides an alternative by allowing models to learn generalisable representations from large amounts of unlabelled data using objectives such as masked language modelling [28] and contrastive instance discrimination [124, 125]. Fine-tuning such models for downstream classification can reduce both training time and data requirements relative to training from scratch [28], and may match or exceed fully supervised counterparts [124].

Although Textual Images differ from natural images, the availability of pre-trained visual backbones makes it important to test whether their learned representations can transfer effectively to this new input form.

This literature provides the motivation for RQ3 and Chapter 6. That chapter examines whether publicly available pre-trained image classifiers can replace the custom ResNet visual backbone used for Textual Image classification. It also compares CNN and Transformer-based vision architectures, interpolation methods, model scale, performance and computational efficiency.

2.6.4 RQ4 and Chapter 6: Complementary Sentiment Models and Ensembling

The literature reviewed in Sections 2.2.3.1, 2.3.2, 2.4 and 2.4.1 shows that sentiment can be represented using traditional count- and topic-based features, contextual Transformer representations and lexical knowledge. These approaches process different aspects of text and may therefore make different errors and provide complementary prediction signals. Rather than treating one representation as an unconditional replacement for another, this motivates examining whether their probabilities can be combined.

This provides the literature-based rationale for RQ4 and Experiment 4 in Chapter 6. The experiment evaluates

whether the Textual Image classifier provides complementary information when combined with a lexically enhanced BERT model and traditional LSA/LDA-based sentiment features through ensemble methods.

2.7 Conclusion

This chapter systematically reviewed the foundational literature for text classification, progressing from traditional vectorization to modern deep learning architectures. A central theme was the novel approach of forming images from text, a technique that bridges the gap between NLP and computer vision. The review highlighted seminal work, such as the BLSTM-2DCNN model, which demonstrates how sequential text data can be transformed into a feature-rich, image-like tensor. Subsequently, in the literature on 2D CNNs, particularly powerful computer vision architectures like ResNet , were examined for its potential to effectively classify these generated Textual Images.

The review then traced the evolution to contextualized word representations, focusing on Transformer-based models like BERT. This advancement provides the basis for creating contextualized Textual Images, capturing deep semantic relationships. The systematic literature review confirmed that enriching these contextual embeddings, often by combining Transformers with other neural layers or lexical resources, is a promising research direction. This comprehensive background establishes the theoretical groundwork for leveraging these hybrid, Textual Image classification methodologies for document-level analysis of sentiment analysis. The final section linked these literature strands to RQ1–RQ4 and to the experimental programme in Chapters 4–6, clarifying how the reviewed evidence motivates the subsequent investigations.

Chapter 3

Methodology

3.1 Introduction

Sentiment Analysis is a text classification problem where the text data is initially encoded into numerical vectors (Information encoding) and then mapped to predefined sentiment classes (information classification) using various supervised machine learning algorithms. This process enables models to automatically determine whether a given text expresses a positive, negative, or neutral sentiment by leveraging techniques such as Bag of Words (BOW), Word embeddings, or deep learning-based encodings for feature extraction, followed by classifiers like Naïve Bayes, Support Vector Machines (SVM), or deep neural networks for sentiment prediction (more details in next sections).

Textual Image classification integrates Natural Language Processing (NLP) techniques for information encoding with Computer Vision models for information classification, allowing textual content to be analysed in image form. This approach leverages methodologies from both NLP and Computer Vision, combining text representation techniques with image-based learning models to extract user sentiment. Consequently, the experimental methods in this work are derived from the intersection of these two fields.

This chapter outlines the methodology adopted to address the research questions outlined in Section 1.7. Detailed hardware and software requirements to replicate this work are provided. As with any text analysis research, this work has also undertaken various pre-processing and feature extraction steps. This chapter will attempt to demonstrate the utility of each step.

Within our experimentation, the convolution neural network (CNN) depth, the use of contextualised vectors, dataset size, lexical resources, model fine-tuning and model-ensemble impacts on Textual Image classification are investigated. We have explored various textual feature extraction methods using neural-network-based (BLSTM, BERT) methods to form Textual Images. We have explored two different computer vision classification approaches for classifying Textual Images i.e., CNN (ResNet, DenseNet, EfficientNet) and Transformer (ViT, SwinFormer) based classifiers.

This work investigates leveraging lexical resources like SentiWordNet to fine-tune a BERT model with an auxiliary task in combination with traditional text topic modeling techniques for textual classification which can complement Textual Image classification in an Ensembling setting.

We have also investigated the feasibility of leveraging pre-trained text and vision models for sustainable Textual Image modelling.

3.2 Hardware and Software setup

3.2.1 Hardware

All practical implementations and experiments were conducted on NVIDIA GPU compute nodes provided by the University High-Performance Computing (HPC) cluster [126]. GPU memory availability determined the feasible batch size, and jobs were configured to complete within the cluster's 96-hour time limit. Data Parallel and, where supported, Distributed Data Parallel were used for multi-GPU execution; some experiments were conducted on a single GPU.

As the HPC cluster was a shared resource, GPU availability varied between runs and the precise GPU model, number of GPUs and available memory were not recorded for every experiment. The cited cluster documentation

provides additional information about the infrastructure.

3.2.2 Software

The experiments were implemented in Python, beginning with Python 3.7, using CUDA 10, CUDA 11 or CUDA 12 according to the compatible cluster environment. The custom BLSTM and ResNet models were implemented with Keras and TensorFlow [38, 39], while PyTorch and Hugging Face Transformers [41, 127] were used for the pre-trained language and vision models. Supporting libraries included NumPy, pandas, NLTK [68], scikit-learn [128], OpenCV, Matplotlib and the Python regular-expression library [129].

Package versions were selected to remain compatible with the Python and CUDA versions available for each experiment. The Appendix (Section 9.1.2) and the accompanying code repository[1] provide the package list and the corresponding version requirements for reproducing the software environment.

Techniques such as early stopping and saving the best model weights based on validation accuracy were employed to optimise training. Additionally, trained model weights were reloaded to resume training in case of interruptions caused by any of the listed constraints.

To account for variability in model performance, all experiments were repeated at least five times, and average performance was considered for evaluation.

3.3 Datasets

3.3.1 Sentiment140

The main dataset used to test various neural architectures for text classification is Sentiment140 [16]. It is a Sentiment Analysis dataset with 1.6 million tweets. It was collected using Twitter API by the authors of [16]. The dataset has two labels "0" and "4". Positive tweets are represented by 4 and 0 represents a negative tweet. It is a balanced dataset with each category containing 0.8 million data points.

The max tweet length was 144 characters at the time this dataset was collected. 144 characters seems very small for the text document analysis, but the overall count of the data set is ideal to compare different model performance by varying the training samples. The label "4" is mapped to label "1" to make the analysis appropriate for binary classification. In our analysis we have created several balanced subset variants of the dataset from the original dataset with varying sizes. These variants contain 1250, 2500, 5000, 10,000, 20,000, 40,000, 60,000, 80,000, 160,000, 320,000, 640,000 and 1,280,000 samples respectively. These subsets are used in investigating the data set size impact of the Textual Image classification.

3.3.2 IMDB Review

This dataset is a popular benchmarking dataset and was published in 2011 by Stanford researchers, who wanted to capture sentiment information from text documents. They chose the IMDB (Internet Movie Database) website and gathered 50,000 reviews with up to 30 reviews per movie. This is a balanced dataset with two categories, positive and negative reviews [130]. IMDB rates a movie out of 10 and researchers labelled ratings 1 to 4 as negative and 7 to 10 as positive. This dataset does not contain any neutral samples.

This dataset consists of a total of 50,000 samples, split equally into training and testing sets with 25,000 samples each. Both sets are balanced, containing 12,500 samples per class. During our analysis we mapped "positive" review to "1" and "negative" review to "0". The average length of a text document is above 200 words which makes it ideal for comparing model performance based on the document length by creating different versions of data sets based on the document length size.

To systematically evaluate the impact of input length and training data scale on model performance, we derived multiple dataset variants from the original corpus. Specifically, we constrained the maximum document word length to fixed thresholds of 32, 64, 128, 254, 356, and 512 tokens, thereby generating versions that capture varying levels of contextual information. This design allows us to analyse how sequence length influences the representational capacity of the model. In addition, to study the effect of data availability on learning efficiency, we constructed training subsets by randomly sampling the original dataset at different proportions, namely 5%, 10%, and higher fractions. These subsets provide a controlled environment for examining model robustness under limited-resource scenarios and enable insights into data scaling properties.

[1] GitHub - harpreetsinghjk99-collab/Textual-Images: Code for Textual Images classifiers

Together, these dataset configurations form the experimental basis for assessing the trade-offs between input length, data size, and model performance in the context of Textual Image classification for sentiment analysis.

3.4 Machine learning pipeline adopted in this work

The primary focus of this work is exploring different Textual Image generation and Textual Image classification techniques. To help understand the machine learning pipeline when undertaking the Textual Image generation, classification and Ensembling with existing techniques, Figure 3.1 and Figure 3.2 are provided.

The five pipelines form a staged experimental framework rather than a set of independent alternatives. The investigation progresses from a static word representation, to a contextualised representation, to an assessment of whether the BLSTM remains necessary after contextual encoding, and then to transfer learning with pre-trained vision classifiers and the fusion of complementary sentiment representations. Pipeline 1 establishes the Textual Image approach using GloVe, BLSTM and a custom ResNet and is used to address RQ1. Pipelines 2 and 3 introduce BERT contextualisation and compare BLSTM-based and direct image formation, thereby addressing RQ2. Pipeline 4 replaces the custom image classifier with pre-trained CNN or Transformer-based vision backbones to address RQ3. Finally, Pipeline 5 combines the probability outputs of the Textual Image classifier with lexical and topic-based text models to evaluate predictive complementarity and address RQ4. Thus, the progression represented by the five pipelines is: static representation, contextual representation, BLSTM ablation, pre-trained vision transfer, and multimodel ensemble.

Figure 3.1 illustrates four distinct pipelines side by side, facilitating the comparison of techniques adopted at each stage of sentiment classification using Textual Images. The pipelines comprise several sequential stages: text data acquisition, tokenisation, text vectorisation and feature extraction, temporal feature extraction, Textual Image formation, choice of image classifier, and performance evaluation (accuracy).

Each pipeline changes a specific part of the preceding configuration so that the contribution of text representation, temporal processing and image classification can be examined. Their configurations, rationales and experimental roles are described below.

3.4.1 Pipeline 1 : Static Textual Images and a custom ResNet (RQ1)

Pipeline 1 provides the foundation for the subsequent pipelines. GloVe is used to represent each token with a static word embedding, after which a Bidirectional Long Short-Term Memory (BLSTM) network models the sequence in both directions. Unlike Pipelines 2–4, Figure 3.1 does not show a separate string-cleaning stage for Pipeline 1. This is because filtering is performed implicitly during GloVe vocabulary mapping: only tokens with a corresponding pre-trained GloVe vector are retained in the embedding matrix, while unmatched items such as URLs, hyperlinks and other tweet artefacts are omitted. Consequently, this filtering forms part of tokenisation and embedding lookup rather than a separate preprocessing stage. The BLSTM output is arranged as a single-channel Textual Image and classified using the purpose-built shallow or deep 2D ResNet variants proposed in this work. These models use custom residual-block configurations and their vision weights are initialised and trained from scratch on the sentiment dataset. The rationale is to test whether the latent sequential representation can be treated as an image-equivalent feature map and whether deeper residual spatial feature extraction improves sentiment classification. Chapter 4 therefore compares the proposed BLSTM–ResNet variants with the shallow BLSTM–2DCNN baseline and varies ResNet depth, Sentiment140 training-set size, and residual-block construction, including different block depths and bottleneck configurations. These experiments provide the principal evidence used to answer RQ1.

3.4.2 Pipeline 2 : Contextualised Textual Images with a BLSTM (RQ2)

Pipeline 2 introduces BERT as the text-vectorisation and feature- extraction component. Unlike the context-independent GloVe vectors in Pipeline 1, the fine-tuned BERT model generates a representation of each token that is conditioned on its surrounding text. The BLSTM is retained so that contextualisation can first be examined within the established BLSTM-based image-formation process. Its output forms the Type B contextualised Textual Image, which is classified using a standard single-channel ResNet.[2] The principal comparison with Pipeline 1 asks

[2]Here, "standard ResNet" denotes that the model-depth nomenclature and architectural parameters, including convolutional kernel sizes, numbers of filters, padding and pooling settings, follow the original ResNet architecture proposed by He et al. [23]. The input layer is adapted for a single-channel Textual Image and the output layer for the sentiment-classification task. Unlike the pre-trained vision backbones in Pipeline 4, its vision weights are initialised and trained from scratch; only the preceding BERT component begins with pre-trained language-model weights.

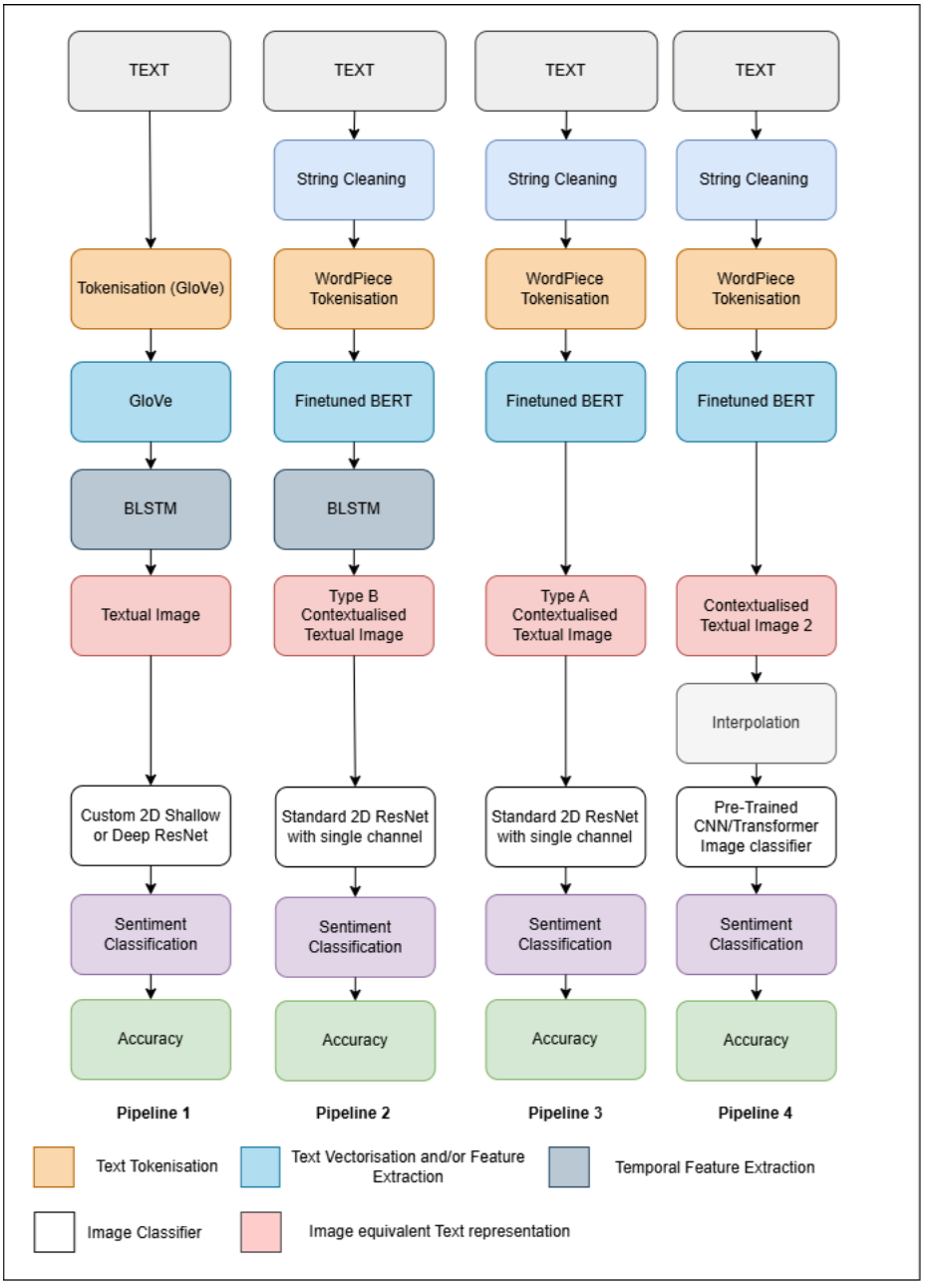


Figure 3.1: Different pipelines used in this research to represent the text as an Image for text sentiment analysis

whether replacing static embeddings with contextual embeddings produces more discriminative Textual Images. Experiment 1 in Chapter 5 examines this question across Sentiment140 dataset sizes and different ResNet depths, with standalone BERT also used as a text-classification baseline. Pipeline 2 therefore begins the investigation of RQ2.

3.4.3 Pipeline 3 : Direct contextualised Textual Images without a BLSTM (RQ2)

Pipeline 3 retains the fine-tuned BERT and the standard single-channel 2D ResNet trained from scratch in Pipeline 2, but removes the intermediate BLSTM. The final BERT hidden state is already a two-dimensional token-by-feature matrix, so it can be reshaped directly into a Type A contextualised Textual Image. This change tests whether the BLSTM contributes an additional useful sequential transformation after BERT has encoded bidirectional context, or whether it introduces unnecessary serial processing. The comparison between Pipelines 2 and 3 therefore acts as a BLSTM ablation. This is explored in Experiment 2 of Chapter 5, Section 5.3, using the BERT, BERT–BLSTM, BERT–ResNet and BERT–BLSTM–ResNet configurations illustrated in Figure 5.7. Their performance is evaluated across different IMDB token lengths, while FLOPs are examined to compare their computational requirements. Together, Pipelines 2 and 3 address the contextual-representation and architecture components of RQ2.

3.4.4 Pipeline 4 : pre-trained vision backbones (RQ3)

Pipeline 4 follows Pipeline 3 up to the formation of the Type A contextualised Textual Image but replaces the standard single-channel ResNet trained from scratch with a publicly available CNN or Transformer-based vision backbone whose weights were pre-trained on an image dataset and are subsequently fine-tuned on Textual Images. Interpolation is consequently added to resize the Textual Image to the fixed input dimensions required by the selected vision backbone. The rationale is to determine whether transfer learning from pre-trained image models can replace the purpose-built ResNet and offer a more resource-efficient route to Textual Image classification. The conceptual comparison between Pipelines 3 and 4 isolates the change from a standard ResNet trained from scratch to a pre-trained vision backbone. Experiments 1–3 in Chapter 6 evaluate this approach by comparing convolutional and Transformer-based vision architectures, dataset sizes, interpolation methods, model capacities, accuracy and computational requirements. These experiments provide the evidence used to address RQ3.

3.4.5 Pipeline 5 : probability-level fusion of complementary representations (RQ4)

Figure 3.2 illustrates pipeline 5, presenting an overview of an ensemble approach that integrates class probabilities derived from Textual Image classification techniques with existing approaches for sentiment analysis through a meta-learning framework. Specifically, the ensemble aggregates three probability estimates (P1, P2, and P3). Probability P1 is generated using pipeline 4, as previously outlined in Figure 3.1, involving Textual Image classification. Probability P2 originates from a BERT model fine-tuned on an auxiliary task leveraging lexical resources, thereby incorporating additional linguistic knowledge. Finally, Probability P3 is produced through traditional statistical and probabilistic text-modelling techniques employing conventional machine learning classifiers.

These individual probabilities are subsequently combined by a meta-learner, which integrates their complementary strengths to achieve a final sentiment classification. The rationale for Pipeline 5 is that the three branches represent sentiment differently: P1 captures spatial patterns in contextualised Textual Images, P2 adds lexical sentiment knowledge to contextual BERT representations, and P3 captures global distributional and topic information. The hypothesis is that these representational differences produce non-identical prediction errors that can be combined to improve classification. Supplying class probabilities rather than hard labels also preserves the confidence of each base classifier for use by the meta-learner. Experiment 4 in Chapter 6 examines the individual branches, their prediction overlap, and their combination through voting and learned meta-classification methods on the IMDB dataset and feasible Sentiment140 subsets. The comparison of the ensemble with its strongest individual classifier provides the evidence used to answer RQ4. This ensemble methodology demonstrates how Textual Image classification can be integrated with diverse text-based approaches to enhance sentiment classification performance.

The remainder of this chapter defines the methodological components required to implement the five pipelines presented in Figures 3.1 and 3.2. The discussion is organised by shared function rather than repeated separately for each pipeline. It first describes text preprocessing and feature extraction, followed by the neural architectures used for representation learning and classification. It then explains the formation of Textual Images and contextualised Textual Images, the interpolation required by pre-trained vision backbones, and the transfer-learning and meta-learning strategies used in Pipelines 4 and 5. This organisation establishes the common methodological foundation for the controlled comparisons undertaken in Chapters 4, 5 and 6.

Text Preprocessing

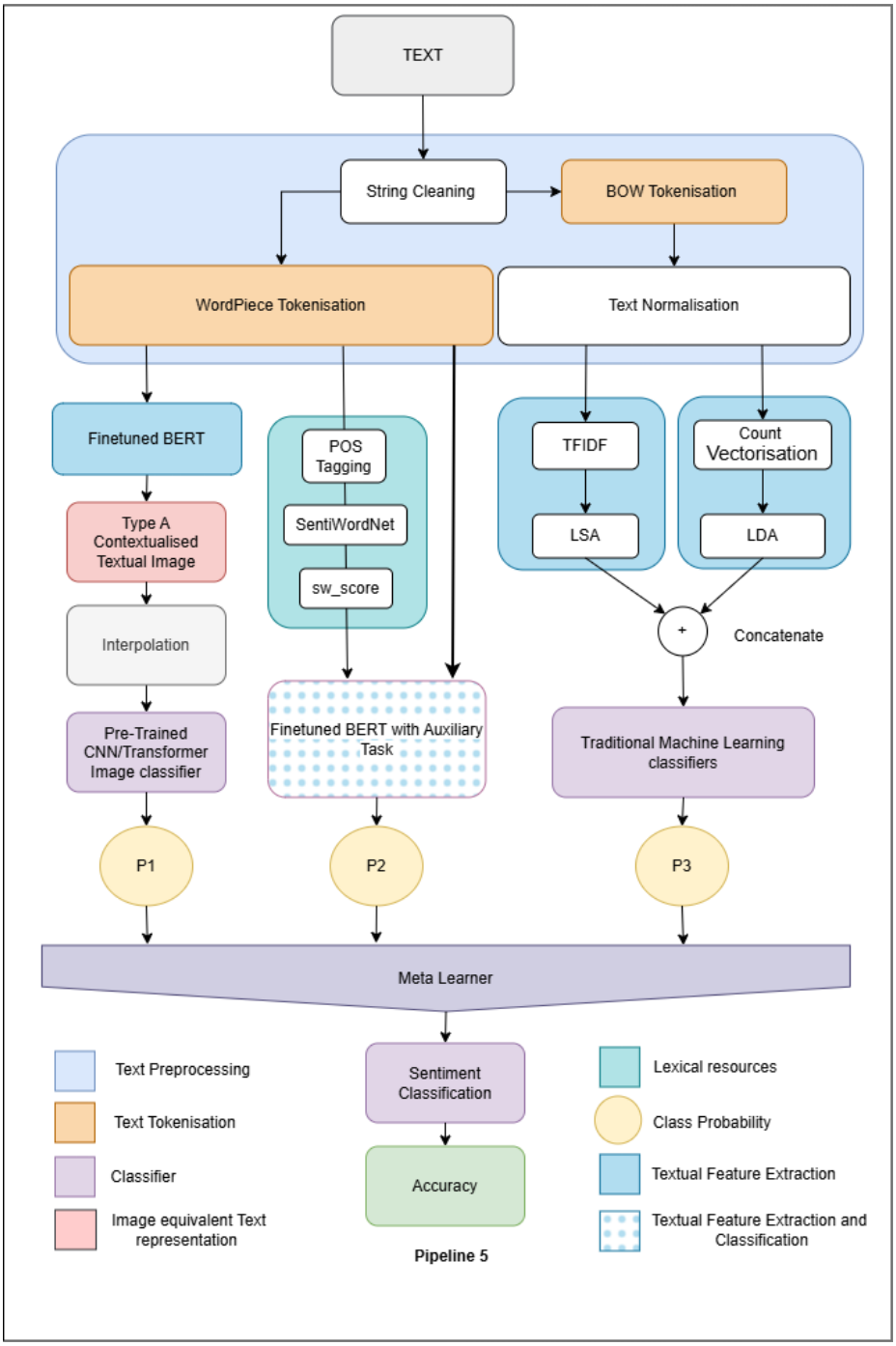


Figure 3.2: machine learning pipeline with Ensemble model

Table 3.1: Relationship between the experimental pipelines, controlled comparisons, experimental chapters and research questions.

Pipeline	**Main methodological change**	**Comparison and hypothesis**	**Experimental treatment**	**RQ**
1	Static GloVe embeddings are processed by a BLSTM to form a Textual Image classified by purpose-built ResNet variants trained from scratch.	Tests whether image-equivalent text representations support 2D spatial feature extraction and whether increased residual depth improves classification.	Chapter 4: ResNet depth, dataset size and residual-block construction.	RQ1
2	Fine-tuned BERT replaces GloVe while the BLSTM-based Type B image-formation stage and a standard single-channel ResNet trained from scratch are used.	Pipeline 1 versus Pipeline 2 examines whether contextual embeddings produce more discriminative Textual Images than static embeddings.	Chapter 5, Experiment 1: Sentiment140 dataset size and ResNet depth.	RQ2
3	The BLSTM is removed and the BERT output is reshaped directly into a Type A contextualised Textual Image classified by a standard single-channel ResNet trained from scratch.	Pipeline 2 versus Pipeline 3 tests the incremental value and computational cost of the BLSTM after BERT contextualisation.	Chapter 5, Experiment 2: architecture, IMDB token length, accuracy and FLOPs.	RQ2
4	The standard ResNet trained from scratch is replaced by an interpolated-input CNN or Transformer vision backbone with pre-trained weights.	Pipeline 3 versus Pipeline 4 examines transfer learning; Pipeline 4 variants compare backbone architecture, capacity and interpolation.	Chapter 6, Experiments 1–3: dataset size, interpolation, architecture and computational requirements.	RQ3
5	The probability outputs of the Textual Image, BERT–SWC and LSA+LDA branches are combined by a meta-learner.	Tests whether representational diversity creates complementary errors and whether probability-level fusion improves on the strongest individual classifier.	Chapter 6, Experiment 4: prediction overlap, voting and learned meta-classifiers.	RQ4

This section details the pre-process steps applied to the raw text input in some or all pipelines. It will address string cleaning methods, various tokenisation techniques, and text normalisation methods.

Feature Extraction

This section focuses on the methods used to transform text into vector representations. It will encompass traditional Text Vectorisation methods, including Count Vectorizer, Term Frequency-Inverse Document Frequency (TFIDF), Bag-of-Words (BOW), Glove (Global vectors) as well as topic modelling techniques like Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). Additionally, the integration of lexical resources (e.g., SentiWordNet) will be discussed.

Neural Network and variants adopted in this work

This section is dedicated to the application of neural networks in transforming and classifying data and start by understand the basic neural network architecture and the weight training with various optimiser. It covers most of the categories of neural model used in this work. This seem a detour, but it is important at this point in methodology to understand a neural model architecture and how the model learns before we can discuss how some of them are used to form Textual Images to perform sentiment analysis.

Textual Images

The section discusses the intuition for a Textual Image and how vectorised text can be used to generate Textual Images. Vectorised texts are discussed under feature extraction, but the vectorised text can be processed using deep neural model to generate representation in latent space which can then help in extracting opinions from the text. The latent representation of the deep neural model can also be considered as text features and used in feature extraction. In this work the latent representation will be used to create Images (Textual Image and Contextualised Textual Image). Following this, it will discuss various Textual Image classifiers, including the proposed models. This approach aligns with treating the early layers of deep learning models as feature extractors and the final layer performing Sentiment Classification as discussed in Vectorisation granularity Section 2.1.

Sustainable Neural Network development

This section addresses the advanced model training and model management techniques. It will cover the Meta Learner architecture utilized in Pipeline 5 for combining Class Probability outputs from multiple base classifiers (P1, P2, P3) to perform final Sentiment Classification. This discussion will be contextualized alongside other adopted sustainable training strategies for neural networks considered in this work.

3.5 Text preprocessing

To analyse a text document for sentiment analysis, it is crucial to preprocess the data like any other machine learning task to make them fit for analysis. In context of Text data, it is important to remove unwanted content from the text that is deemed unnecessary or garbage for sentiment extraction. To perform sentiment analysis on tweets and online reviews it is important to remove unrelated content form the text, prior to the analysis. Lexical analysis relying on word frequency utilise text transformation techniques to lower the overall word count in dataset word corpus which can result in compact word vectors.

3.5.1 String cleaning

The text documents can contain words and content which do not contribute to the overall sentiment associate with the document, and they can be removed using regular expressions. Regular expressions are sequence of characters that can be matched against the text. The matched characters, in our case, can be different unwanted artefacts of the social platforms the text is originated from. For cleaning social media text data, we have used different string cleaning steps.

3.5.1.1 URL

Uniform resource locators (URL) are address used to access the online resources on the Internet. The resources can be web pages, files, image or video. This web addresses tell the browser where to find this resource. The URLs are common in social media and review platforms as the users can link online resources with comments. The content of the URLs might contribute to the overall sentiment of the text but if explored further, the text associated with the link itself is an objective artifact and is removed during this analysis.

3.5.1.2 Username

The usernames are the unique identifiers or alias of the social media users which are similar to a name of a person in the real world. Online text can contain references to users which is usually preceded with the "@" symbol. These usernames are deemed unnecessary for document level text analysis and are removed during the preprocessing.

3.5.1.3 Hashtag

Hashtags are words or key phrases proceeded by the "#" symbol. These words or phrase are used to categorise the content and make them searchable on social media platforms. Sometime when hashtags are included in the text, it becomes a hyperlink (a clickable item linking to another webpage) grouping all the documents containing the same hashtag.

The Pattern column shown in the Table 3.2 represent strings that can be used to find and remove matching patterns in the text document. These strings can be combined using the "|" operator to match several patterns at once.

Table 3.2: Regular expression patterns, match and example.

Pattern	Match	Example
@\S+	Usernames(@mentions)	@harry449
https?://\S+	Full URLs (http & https)	https://example.com
#	Hashtags (#)	#Amazing

In this work, the sub method from Regular Expression Python package method [129] is used to clean the text documents using the matching strings "**https?://\S+|@\S+|#**" to find the matching stings in the test document and removing them.

Example

Cleaning string: "https?://\S+|@\S+|#"

Text input: @john_hinton Check out this amazing movie review https://t.co/review123 #spiderman Must watch

Cleaned Text: Check out this amazing movie review Must watch

3.5.2 Tokenisation

Tokenisation is a process of breaking a word document into individual words or terms, known as tokens. This is a fundamental step in NLP, as it enables analysis of raw text. Tokenisation is a pre-requisite in creating fixed vector representations of text, also known as textual encodings.

Example

Input Text 1: I watched the spider man movie, but it was horrible compared to previous movie.

Output Tokens 1: ['i', 'watched', 'the', 'spider', 'man', 'movie', ',', 'but', 'it', 'was', 'horrible', 'compared', 'to', 'previous', 'movie.']

As NLP has advanced, different tokenisation techniques have been developed to suit various tasks. The choice of tokenizer often depends on the type of feature extraction method adopted in the text processing pipeline.

In this work, I have applied different tokenizer functions tailored to different feature extraction and text analysis pipelines discussed in Section 3.4. Specific details are provided in the corresponding sections.

3.5.3 Normalisation

Text processing techniques relies on finding unique text tokens in the entire data set to create vectors which represent the input text document. This can result in very sparse high dimensional data vectors. Text normalisation is a technique of transforming text into a standard format for effective text analysis. Normalisation reduces variability in text and removes unwanted words.

Text normalisation can be leveraged to aid sentiment analysis by finding a word that can be a unique representation for different words and removing words that might not contribute to the overall sentiment of the word.

3.5.3.1 Stop words removal

Removing stop words from the text data can benefit BOW based sentiment analysis method. Stop words are common words that appear in the document like (is, am, or, of, the). These words might not contribute to the overall sentiment of the documents, so these are removed in analysis where the term frequency is important. Some NLP related Python packages provide a list of common words for different languages, and these words can be removed by matching the terms in the document with the stop word list.

Example

Input Text:

Check out this amazing movie review Mustwatch. I loved it, but it was too long and slow at times.

Input Tokens: ['check', 'out', 'this', 'amazing', 'movie', 'review', 'mustwatch', '.', 'i', 'loved', 'it', ',', 'but', 'it', 'was', 'too', 'long', 'and', 'slow', 'at', 'times', '.']

Stop word Removal:
['check', 'amazing', 'movie', 'review', ':', 'mustwatch', '.', 'loved', ',', 'long', 'slow', 'times', '.']

3.5.3.2 Stemming

Stemming is the process of reducing a word to their root form or stem. A Root word is considered a bare word with which a prefix or suffix can be attached to form a new word. In text analysis it is not important if the stem word is a valid root word, but several similar words should be mapped to same stem word. The list of stem words for various languages is provided in NLP related Python packages.

Table 3.3: Example words with same stem

Word 1	Word 2	Stemmed Word
running	runner	run
connectivity	connected	connect
happiness	happy	happi
arguing	argument	argu
caring	careful	care

The examples in Table 3.3 shows how different word forms are mapped to the same stem, which helps reduce the number of unique tokens. This reduction lowers memory usage, computation cost, and the running time of algorithms.

In this work, the PortStemmer method is used from NLTK Python package [131] to perform stemming on a given text document.

Example:

Text:
Check out this amazing movie review Mustwatch. I loved it, but it was too long and slow at times.

Tokens: ['check', 'out', 'this', 'amazing', 'movie', 'review', 'mustwatch', '.', 'i', 'loved', 'it', ',', 'but', 'it', 'was', 'too', 'long', 'and', 'slow', 'at', 'times', '.']

Stemmed Tokens:
['check', 'out', 'thi', 'amaz', 'movi', 'review', 'mustwatch', '.', 'i', 'love', 'it', ',', 'but', 'it', 'wa', 'too', 'long', 'and', 'slow', 'at', 'time', '.']

3.5.3.3 Lemmatisation

Lemmatisation is the process of grouping inflected forms of a word and replacing them with their base form, known as the lemma. Inflected forms are grammatically modified versions of a word that reflect categories such as tense, case, voice, aspect, person, number, or gender. For example, the verb *run* can appear in different inflected forms like *ran* or *running*, depending on the tense or aspect, a process known as verb conjugation.

During preprocessing, the text words are reduced to their lemmas. The process is like stemming where a dictionary

is used to link words to their lemmas.

Table 3.4: Example words with same lemma

Word 1	Word 2	Lemma
running	ran	run
better	best	good
caring	cared	care
argued	arguing	argue
drove	driven	drive

The example in Table 3.4 shows how different word inflections are mapped to the same lemma, helping reduce the number of unique tokens. This reduction lowers memory usage, computation cost, and the running time of algorithms.

In this work, the WordNetLemmatizer class from the NLTK Python package [131] is used to perform lemmatization on the input text documents.

Example:

Text: Text: The doctors were running to save the patients. The leaves of the trees were falling.

Tokens: ['the', 'doctors', 'were', 'running', 'to', 'save', 'the', 'patients', '.', 'the', 'leaves', 'of', 'the', 'trees', 'were', 'falling', '.']

Lemmatized Tokens: ['the', 'doctor', 'were', 'running', 'to', 'save', 'the', 'patient', '.', 'the', 'leaf', 'of', 'the', 'tree', 'were', 'falling', '.']

3.5.4 Lexical Resources

In NLP, lexical resources are databases or tools that provide structured information about words and their relationships, meanings, grammatical behaviour, and usage. Sentiment extraction can leverage these lexical resources to effectively extract the sentiment in the text.

3.5.4.1 Part-of-Speech (POS) Tagging

Part-of-Speech (POS) tagging is used to label a word in a text based on its grammatical category such as Noun, pronoun, verb, etc. The part of speech of certain word changes the subjectivity of the sentences so this can be useful in deciding the sentiment of the overall text. POS is done at a syntactic level as it identifies the role of a word in a sentence. In our analysis, we used the *pos_tag*() function from the NLTK Python package [131] to perform part-of-speech (POS) tagging.

Table 3.5: NLTK library POS tags examples with description

S No.	Sample word	Tag	Description
1	desk	NN	noun, singular
2	desks	NNS	noun plural
3	Harrison	NNP	proper noun, singular
4	Americans	NNPS	proper noun, plural
5	big	JJ	adjective
6	bigger	JJR	adjective, comparative
7	biggest	JJS	adjective, Superlative
8	take	VB	verb, base form
9	took	VBD	verb, past tense

Continued on next page

S No.	Sample word	Tag	Description
10	taking	VBG	verb, gerund/present participle
11	taken	VBN	verb, past participle
12	I, he, she, it	PRP	personal pronoun
13	my, his, hers	PRP$	possessive pronoun
14	silently	RB	adverb
15	better	RBR	adverb, comparative
16	best	RBS	adverb, superlative
17	but	CC	coordinating conjunction
18	the	DT	determiner
19	to	TO	to

The examples in Table 3.5 shows different tags that can be assigned to a word and their descriptions. The sample word represents an example that may receive the corresponding label during the POS tagging process. However, the actual tag assigned in practice may vary depending on the sentence context and the word's position or usage.

Steps involved in implementing POS tagging

INPUT text

TOKENISE the article into words

PERFORM POS tagging on each token using *nltk.pos_tag*()

Example:

Input Text: "I watched the spider man movie but it was horrible compared to previous movie"

Tokenised text: ['I', 'watched', 'the', 'spider', 'man', 'movie', 'but', 'it', 'was', 'horrible', 'compared', 'to', 'previous', 'movie']

Tagged tokens: [('I', 'PRP'), ('watched', 'VBD'), ('the', 'DT'), ('spider', 'NN'), ('man', 'NN'), ('movie', 'NN'), ('but', 'CC'), ('it', 'PRP'), ('was', 'VBD'), ('horrible', 'JJ'), ('compared', 'VBN'), ('to', 'TO'), ('previous', 'JJ'), ('movie', 'NN')]

3.5.4.2 SentiWordNet

To perform sentiment analysis, it would be beneficial to utilise tools which can identify a sentiment associated with each word in a language. SentiWordNet is one such tool. Sentiment score, used in [113][115], is generated with the help of WordNet and SentiWordNet. According to [115], "WordNet is a large lexical database which grouped words into synonym sets (synsets) to represent different meanings". SentiWordNet assigns each synset a positivity score, negativity score and objective score ranging from 0 to 1 to reflect its sentiment intensity". The sum of positive, negative and objective score equals 1 for each synsets.

Table 3.6: Example of different Synsets for word "watch" from WordNet and associated SentiWordNet scores

Synsets	Definition	Positive, Negative, objective scores
watch.n.01	a small portable timepiece	0.0, 0.0, 1.0
watch.n.02	a period of time (4 or 2 hours) during which some of a ship's crew are on duty	0.0, 0.0, 1.0
watch.n.03	a purposeful surveillance to guard or observe	0.0, 0.0, 1.0
watch.n.04	the period during which someone (especially a guard) is on duty	0.0, 0.0, 1.0
watch.v.01	look attentively	0.125, 0.0, 0.875

Continued on next page

Synsets	Definition	Positive, Negative, objective scores
watch.v.02	follow with the eyes or the mind	0.375, 0.0, 0.625
watch.v.03	see or watch	0.0, 0.0, 1.0
watch.v.04	observe with attention	0.0, 0.0, 1.0
watch.v.05	be vigilant, be on the lookout	0.375, 0.0, 0.625
watch.v.06	observe or determine by looking	0.0, 0.0, 1.0

The Table 3.6 shows an example of different synsets in the WordNet library for the word *watch* and the corresponding sentiment scores in the SentiWordNet library.

WordNet provides different synsets for most words in the English language which seems like POS tags but not at a granular level. POS works in context of a given sentence and assign tags at a syntactic level. WordNet focuses on semantic meaning by providing a different set of synonyms for each word, representing a different meaning. It uses "n", "v", "a", "r" to represent noun, verb, adjective and adverb respectively.

As the Tags used for POS are different then the tags provided by the WordNet/SentiWordNet, so the tags can be mapped using a simple dictionary key value pair.

nltk_to_sentiwordnet = {"NN": "n", "VB": "v", "JJ": "a", "RB": "r", "CC": "n", "DT": "n", "JJR": "a", "JJS": "a", "NNS": "n", "NNP": "n", "NNPS": "n", "PRP$": "n", "RBR": "r", "RBS": "r", "VBD": "v", "VBG": "v", "VBN": "v", "VBP": "v", }

3.5.4.3 Word sentiment score (sw_score)

In this study, we are dealing with only positive and negative sentiment class labels, so we have only considered the positive and negative score for each word. A single score for each word is generated by subtracting the negative score from the positive score of each synsets, called the word sentiment score. To keep the implementation simple, if a word has multiple synsets for associated parts of speech, an average of the negative scores is subtracted from the average of the positive scores for the word.

The following steps are unique to this work and showcase the process adopted for generating a sentiment score for each word:

Tokenise the Input text using the BERT tokenizer (refer 3.8.7 for mere details) **INITIALISE** sw_score list with a single element 0 for the [CLS] BERT token. POS tag each token (excluding the special BERT tokens [CLS] and [SEP] at the start and the end)

FOR each tokenized word:

> **RETRIEVE** the corresponding SentiWordNet tag from the *nltk_to_sentiwordnet* dictionary.
>
> **IF** no matching POS tag is found:
>
> > **APPEND** a sentiment score of 0 for the Token.
> > **CONTINUE** to the next token.
>
> **IF** a valid SentiWordNet POS tag exists:
>
> > **RETRIEVE** all possible synsets for the given token SentiWordNet tag
> > **IF** no synsets exist:
> >
> > > **APPEND** a sentiment score of 0 for the token.
> > > **CONTINUE** to the next token.
> >
> > **ELSE**:
> >
> > > Initialise a temporary variable for sum to 0
> > > **FOR** each retrieved synset:
> > >
> > > > **CALCULATE** the sentiment score (*positive_score* − *negative_score*).
> > > > **SUM** these scores into a temporary variable.
> > >
> > > **COMPUTE** the **average sentiment score** for the token divided by total number of retrieved synset.
> > > **APPEND** this averaged sentiment score to sw_score list.

ADD sentiment score of 0 at the end of the sentiment list "sw_score" for special token [SEP].

Example:

Input Text: "I watched the spider man movie but it was horrible compared to previous movie"

Tokenised Text: ['[CLS]', 'i', 'watched', 'the', 'spider', 'man', 'movie', 'but', 'it', 'was', 'horrible', 'compared', 'to', 'previous', 'movie', '[SEP]']

Output sw_score list: [0.0, 0.0, 0.0, 0.125, 0.1072, 0.0, 0.0, 0.0, 0.0, 0.01136, 0.0, 0.0, 0.0, 0.0, 0.0, 0.125, 0.00962, -0.625, -0.625, -0.125, 0.0, 0.0, 0.0, -0.1666, 0.0, 0.0, 0, 0, 0, 0]

3.6 Feature Extraction

3.6.1 BOW textual encoding

Bag of Words (BOW) is a simple, yet powerful technique used in Natural Language Processing (NLP) to convert text into numerical features for machine learning modelling. This generally converts a text document containing words/terms to a high-dimensional **document-term matrix** feature based on the words in dataset corpus.

Suppose we have D documents $(d_1\ , d_2\ , \ldots d_D)$, Vocabular V of terms $(t_1\ , t_2\ , \ldots t_V)$ where $d_{\mathrm{i}}\ \in \mathbb{R}^V$ with $V \gg D$ then document-term matrix is given by:

$$A\ =\ (a_{\mathrm{dt}})_{d=1\ldots D,\ t=1\ldots V} \in \mathbb{R}^{D\times V}$$

(3.6.1)

The element a_{dt} of document-term matrix represents the value of word/term t in document d and most of the time it is 0, meaning the word is absent in the document.

The Bag of word method loses the original arrangement of the words when the vectors are formed as each word/term in corpus is a feature representing a single text document.

3.6.1.1 BOW Tokenisation

In text modelling using the BOW technique, first text is tokenised by splitting text based on whitespace. For more accurate tokenisation, it is common to use punctuation, contractions, numbers, quotes etc along with white spaces for tokenisation. The *word_tokenize*() method from NLTK Python package [68] provides a method to perform tokenisation. Before tokenization, the text is lowercased to avoid duplication due to different cases.

Example:

Input Text 1 : It is very important to get customer feedback on the product to improve the product quality.
Output Tokens 1 : ['It','is','very','important','to','get','customer','feedback' 'on', 'the', 'product', 'to', 'improve', 'the', 'product', 'quality', '.']

Input Text 2: "I can't go. He's not ready, isn't he?"

Output Tokens 2 : ['I', 'ca', "n't", 'go', '.', 'He', "'s", 'not', 'ready', ',', 'is', "n't", 'he', '?']

The tokens are then used to create word vectors

3.6.1.2 Count Vectorisation

Count Vectorisation converts a text into a BOW representation where each element of document term matrix represents the frequency of a word or term within a document. In practice, the top frequency words are kept from the corpus, limited by the pre-defined vocabulary size.

A document-term matrix is generated from the input text documents. Where each word in vocabulary is a feature and the value depends on the frequency of times the word appears in the document.

In count vectorisation, the element of document-term matrix in (3.6.1) is:

$$a_{\mathrm{dt}} \in \{0,\ 1,\ 2,\ \ldots\}$$

(Count of term t in document d)

(3.6.2)

In this work, CountVectorizer method is used form scikit-learn python package [128] to perform count vectorisation on cleaned input text documents. First the method builds a vocabulary from the training set and converts each document into a **word document matrix, a vocabulary size of** 10000 is used. Once the vectorization process is complete, **test data** is transformed into the same feature space using the learned vocabulary.

Example:

Document 1: I am very happy to hear that you got the job. Happy job vibes high fives.

Document 2: world needs to learn that to stay happy we need to train our mind.

Document 3: Job of a person influence their social status.

Count Vectorizer Matrix for vocabulary of 5:

	Happy	Job	Of	That	To
Document 1	2	2	0	1	1
Document 2	1	0	0	1	3
Document 3	0	1	1	0	0

3.6.1.3 TF-IDF vectorisation

Term Frequency Inverse Document Frequency (TF-IDF) enhances count vectorisation by adjusting word importance based on how frequently a word appears across all documents. It has two metrics i.e., Term frequence TF and Inverse document frequency (IDF).

For every document d in corpus (D) *and* word t in vocabulary (V):

$$\mathrm{TF}\,(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$I\mathrm{DF}\,(t,D) = \log\left(\frac{\text{Total number of documents in the corpus}\,(D)}{\mathbf{1} + \text{Number of documents where the term } t \text{ appears}}\right)$$

(1 is added to IDF to avoid division by zero)

The element $a_{t,d}$ is given by:

$$TF - IDF\,(t,d,D) = \mathrm{TF}\,(t,d) \times \ \mathrm{IDF}\,(t,D)$$

(3.6.3)

Document 1: I am very happy to hear that you got the job.

Document 2: world needs to learn that being happy is a state of mind.

Document 3: Job of a person defines social status in society.

In this work, TfidfVectorizer is used to from Scikit-learn python package [128] on cleaned input text documents. First the method builds a vocabulary from the training set and converts each document into a **word document matrix**. Once the vectorization process is complete, **test data** is transformed into the same feature space using the learned vocabulary.

TF-IDF Matrix for vocabulary of 5:

	Happy	Job	Of	That	To
Document 1	0.5	0.5	0.0	0.5	0.5
Document 2	0.5	0.0	0.5	0.5	0.5

| Document 3 | 0.0 | 0.707 | 0.707 | 0.0 | 0.0 |

3.6.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a topic modelling technique in NLP used to discover latent (hidden) relationships between words and documents. LSA employs Singular Value Decomposition (SVD) for extracting abstract relationship and grouping words that occur in similar contexts called Topics. By using the low rank approximation, a compact representation of our data can be obtained, i.e., top k topics from our document-term matrix can effectively be extracted to achieve a dense vector representation of text data.

SVD, a matrix factorisation technique, decomposes document-term matrix A in (3.6.3) into three smaller matrices

$$A = \left(U\Sigma V^{\top}\right)$$

(3.6.4)

Where:

$U \in \mathrm{R}^{D\times D}$ is left singular matrix representing Term-topic matrix,

$\Sigma = diag\left(\sigma_1, \sigma_2, \ldots . \sigma_r\right) \in \mathrm{R}^{D\times V}$ where singular values $(\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0)$ show strength of each latent dimension.

$V \in \mathrm{R}^{V\times V}$ is right singular matrix representing Document-topic matrix and $V^{\top}$ is its transpose.

$r = rank\left(A\right) \leq \min\left\{D, V\right\} \in \mathrm{R}^{V\times V}$

The columns of U and V are orthonormal meaning every topic (latent dimension) is linearly independent.

The dimensionality reduction is done by low rank approximation of (3.6.4) i.e., choosing the top k topics where $k<r$

$$A_k = \left(U_k \Sigma_k V_k^{\top}\right)$$

(3.6.5)

Where $U_k \in \mathrm{R}^{D\times k}$, $\Sigma_k = diag\left(\sigma_1, \sigma_2, \ldots . \sigma_k\right) \in \mathrm{R}^{k\times k}$, $V_k \in \mathrm{R}^{V\times k}$

In this work, TruncatedSVD is used from Scikit-learn python package [128] to perform LSA. First, the top 'k' latent topics are extracted from the training set, capturing the most relevant semantic structures in the corpus. This process transforms the high-dimensional TF-IDF document -term matrix into a lower-dimensional topic space, where each document is represented in terms of learned topics rather than individual words. Once, the training matrix is decomposed, new test data is projected into the same topic space using the learned transformation, ensuring consistent semantic representation across datasets.

3.6.3 Latent Dirichlet Analysis (LDA)

Latent Dirichlet Allocation (LDA) is a probabilistic topic modelling technique in NLP that discovers hidden (latent) topics within a text corpus by treating documents as mixtures of topics and topics as distributions over words.

LDA represent a text corpus in three hierarchical level where each text document is a mixture of topics, each topic is a mixture of words, and each word is drawn from one topic. LDA models topics as probability distributions and assigns words to topics based on their likelihood of co-occurrence in the corpus.

LDA assumes that each document in the corpus is generated from a set of latent topics, where each topic is characterized by a distinct probability distribution over words. The algorithm determines how much each topic contributes to a document and assigns words to topics based on probability (Dirichlet priors). Dirichlet priors help maintain a balanced distribution of words across topics.

LDA introduce an additional latent dimension K in the document-term matrix with count values in (3.6.2) representing number of topics.

For hyper parameters $\alpha \in \mathbb{R}^{K}_{>0}\alpha$ and $\beta \in \mathbb{R}^{V}_{>0}\beta$:

$\forall k = \left\{1,\ 2,\ \ldots, K\right\}$ *, the topic word distribution is:*

$$\varphi_k \mid \beta \sim \text{Dirichlet}\,(\beta)\,,$$

where $\varphi_k \in \Delta^{V-1}$ (probability simplex over the vocabulary).

For each document $\forall d \in \{1, \ldots, D\}$, the topic distribution is:

$$\theta_d \mid \alpha \sim \text{Dirichlet}\,(\alpha)\,, \theta_d \in \Delta^{K-1}$$

For each word position $n \in \{1, \ldots, N_d\}$ *in that document*

Choose a topic:

$$\mathcal{Z}_{d,n} \mid \theta_d \sim Categorical\,(\theta_d)\,.$$

Choose a word:

$$w_{d,n} \mid \ddagger_{d,n},\ \{\varphi_k\} \sim Categorical\left(\varphi_{\mathcal{Z}_{d,n}}\right).$$

The joint probability distribution is given as:

$$p\,(W, \mathcal{Z}, \Theta, \Phi \mid \alpha, \beta) = \left[\prod_{k=1}^{K} p\,(\varphi_k \mid \beta)\right]\left[\prod_{d=1}^{D} p\,(\theta_d \mid \alpha) \prod_{n=1}^{N_d} p\left(\ddagger_{d,n} \mid \theta_d\right) p\left(w_{d,n} \mid \varphi_{\ddagger_{d,n}}\right)\right].$$

(3.6.6)

Here W is the set of all observed tokens; Θ and Φ are the latent parameter matrices whose columns are θ_d and φ_k, respectively.

The quantity of interest is the posterior:

LDA factorises the high-dimensional, sparse document–term data into two lower dimensional stochastic matrices Θ and Φ. The result is an interpretable, probabilistic topic model that scales to large corpora.

In this work, LatentDirichletAllocation method is used form Scikit-learn python package [128] to perform LDA. First, the model learns the top 'k' topics from the training set by iteratively adjusting topic-word and document-topic probabilities to maximize the likelihood of the given corpus. This process transforms the high-dimensional count vectorised document-term matrix into a probabilistic topic space, where each document is assigned a mixture of topics instead of being mapped to a fixed topic. Once the training process is complete, new test data is transformed into the same topic space by assigning topic distributions based on the learned model.

3.6.4 Traditional Machine learning for BOW classification

Classification of BOW textual encodings for sentiment extraction follows the same principles as any supervised classification task, where numerical representations of textual features are assigned to predefined sentiment categories. These encoded numerical values serve as input features for machine learning classifiers which then are used to extract the overall sentiment of the text. Different classifications algorithms are devised in this work to compare the performance of lexicon-based encodings. The GridSearchCV method from Scikit-learn python package [128] is adopted to identify the best model parameter for traditional models.

3.6.4.1 Naïve Bayes (NB)

Naïve Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between features. It works well for text classification tasks like sentiment analysis due to its efficiency in handling high-dimensional data. Despite its simplicity, Naïve Bayes often performs competitively in NLP tasks. In this work, the GaussianNB implementation of Scikit-learn python package [128] is used to perform NB modeling.

3.6.4.2 Logistic Regression (LR)

Logistic Regression is a linear model that predicts probabilities that text belonging to different classes using the sigmoid activation function. It is widely used for binary and multi-class classification problems and performs well

on text classification tasks when applied with BOW feature representations. In this work, the LogisticRegression implementation of Scikit-learn python package [128] is implementation is used to perform LR modelling.

3.6.4.3 Support vector Machine (SVM)

Support vector Machine is a powerful algorithm that finds the optimal hyperplane to separate text classes by maximizing the margin. It is particularly effective in high-dimensional spaces, making it robust for text classification tasks. In this work, the SVC implementation of Scikit-learn python package [128] is used to perform SVM modeling.

3.6.4.4 XGBoost (XGB)

Extreme Gradient Boosting (XGBoost) is an ensemble learning algorithm that uses decision trees to improve classification performance. It is particularly effective for capturing non-linear relationships in text features. XGBoost is widely used in real-world NLP applications. In this work, the GradientBoostingClassifier implementation of Scikit-learn python package [128] is used to perform XGBoost modelling.

3.6.5 GloVe Embeddings

Global Vectors for word Representation (GloVe) is a matrix factorisation approach for representing words in a text document into a numerical vector. GloVe constructs word embeddings using word co-occurrence statistics which results in similar words having higher cosine similarity then dissimilar words.

Example of cosine similarity for GloVe Embeddings for different English words:

Similarity between 'king' and 'queen': 0.7684

Similarity between 'man' and 'woman': 0.7892

Similarity between 'apple' and 'car': 0.2341

Similarity between 'happy' and 'joy': 0.8105

Similarity between 'dog' and 'cat': 0.7348

The example above shows that the word embedding have high cosine similarity if they have semantic relationships

The GloVe approach has different lengths of pre-trained word vectors available to use in different sizes which are 50D, 100D, 200D and 300D. The bigger the embedding size the better the word representation in higher dimension. We have chosen the 300-word vector in our experiment involving GloVe embedding to try to achieve the best performance.

Each word representation in the GloVe dictionary is called an embedding and the entire document represented with embedding is called a document embedding matrix.

3.6.5.1 Tokenisation (GloVe)

To generate a fixed length vector representation of a word using pre-trained Glove embeddings, the Tokenizer method of Keras python package [38] is used to tokenise the text document.

First, this method split the text into individual words based on white space (spaces, tabs, or line breaks). Then, it replaces each word in the dataset corpus with a unique integer identifier know as an index and stores the mapping in a dictionary for efficient lookup. As a result, the text document is represented as a sequence of integers, where each integer corresponds to a word.

Finally, these integer sequences are mapped to their respective pre-trained word vectors using an embedding matrix built from the GloVe vectors.

This two-step mapping process provides flexibility: the same integer-based sequence can be paired with GloVe embeddings of various dimensions (e.g., 50D, 100D, 200D, or 300D, where D represent dimension), or used to train custom word embeddings of any size from scratch. This separation of indexing and embedding allows modularity and reusability of text sequence in different NLP tasks.

Example with two text documents:

["I watched the spider man movie, but it was horrible compared to previous movie.",

"It is very important to get customer feedback on the product to improve the product quality."]

The word Index dictionary generated post Tokenisation:

{'the': 1, 'to': 2, 'movie': 3, 'it': 4, 'product': 5, 'i': 6, 'watched': 7, 'spider': 8, 'man': 9, 'but': 10, 'was': 11, 'horrible': 12, 'compared': 13, 'previous': 14, 'is': 15, 'very': 16, 'important': 17, 'get': 18, 'customer': 19, 'feedback': 20, 'on': 21, 'improve': 22, 'quality': 23}

Sequence of text documents post Tokenisation:

[[6, 7, 1, 8, 9, 3, 10, 4, 11, 12, 13, 2, 14, 3],

[4, 15, 16, 17, 2, 18, 19, 20, 21, 1, 5, 2, 22, 1, 5, 23]]

The original sequence of text can be recovered from the sequence of numbers by using the word index dictionary.

Notice this method automatically converts into lower case all the capital characters in a word. The sequence generated can then be used to map the vector from the trained GloVe dictionary for each unique word. For words which do not exist in the pretrained GloVe embedding dictionary, the word is simply dropped.

To convert a tweet in the dataset into a fixed-length vector, Glove word embeddings are used. The good thing about using Glove is that it only has embeddings for the actual English word and no vectors for anything else (for example, clutter in the tweets like URLs, hyperlinks, hashtags, etc), so if any unwanted artifact is left during text cleaning, it can be automatically removed.

3.7 Neural Network architecture and variants adopted in this work

3.7.1 Feed forward Neural network (FFN)

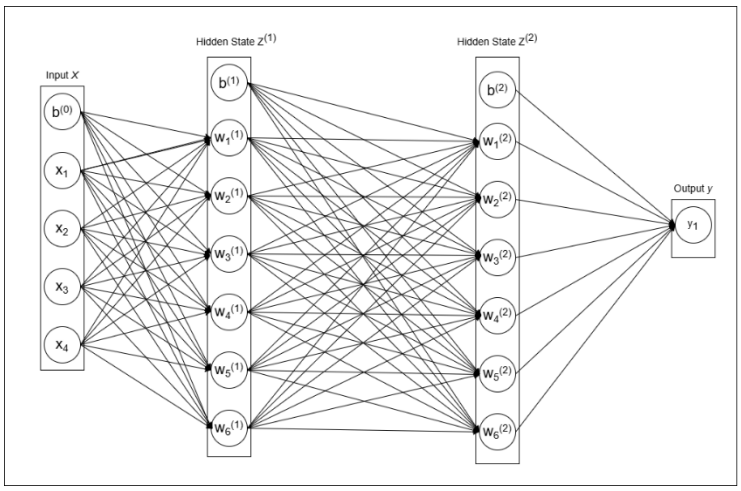


Figure 3.3: Feed Forward neural network with 2 hidden layer

A Feedforward Neural Network (FFN) is a layered neural architecture, as shown in Figure 3.3, in which information flows from the input layer through one or more hidden layers to the output layer without forming cycles.

Let $\mathbf{X} \in \mathbb{R}^{d_0}$ denote the input vector and $\widehat{y} \in \mathbb{R}^{d_{L+1}}$ the network output. The network aims to approximate a target function $y = \mathcal{F}^*(\mathbf{X})$ by learning a parameterised mapping:

$$\widehat{y} = \mathcal{F}(\mathbf{X}; \vartheta),$$

(3.7.1)

Where $\vartheta = \left\{W^{(l)}, b^{(l)}\right\}_{l=1}^{L+1}$ comprises of weights and biases across all layers.

Specifically:

" $\mathrm{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{d_l}$",

"d_0 denotes number of input features",

"d_l is number of units in the l-th layer".

"d_{L+1} denotes the number of output units (e.g., number of classes)"

The forward pass through the network is defined recursively as:

$a^{(0)} = \mathbf{X}$, $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$, $a^{(l)} = \sigma^{(l)}\left(z^{(l)}\right)$ for $l = 1, \ldots, L+1$,

(3.7.2)

Where $\sigma^{(l)}$ is the activation function of the l-th layer.

The choice of activation function of the at the output layer $\sigma^{(L+1)}$ depends on the task (Sigmoid for binary classification and SoftMax for multiclass classification).

To train the network, a loss function $\mathcal{J}(\vartheta)$ (e.g., Mean square error) is minimised using a backpropagation. The error at the output layer is computed as:

$\delta^{(L+1)} = \frac{\partial \mathcal{J}}{\partial z^{(L+1)}} = a^{(L+1)} - y$,

(3.7.3)

This error is propagated backwards through the layers:

$$\delta^{(l)} = \left(W^{(l+1)} \right)^{\top} \delta^{(l+1)} \circ \sigma^{(l)'}\left(z^{(l)}\right),$$

(3.7.4)

Where $\circ$ denotes element-wise multiplication. The parameter gradients are:

$$\frac{\partial \mathcal{J}}{\partial W^{(l)}} = \delta^{(l)} \left[a^{(l-1)} \right]^{\top}, \quad \frac{\partial \mathcal{J}}{\partial b^{(l)}} = \delta^{(l)} .$$

(3.7.5)

3.7.2 Activation function

Activation functions introduce non-linearity into a neural network, enabling it to learn complex and non-linear relationships between input and output. A differentiable activation function is essential for applying gradient-based learning methods such as backpropagation. Below are four commonly used activation functions and their derivatives, defined for a scalar input

h, or vector H in the case of softmax.

- Sigmoid:

  $\sigma(h) = \frac{1}{1 + e^{-h}}$, $\sigma'(h) = \sigma(h)(1 - \sigma(h))$

(3.7.6)

- Hyperbolic Tangent:

  $\sigma(h) = \frac{e^{h} - e^{-h}}{e^{h} + e^{-h}}$, $\sigma'(h) = 1 - \sigma(h)^2$

(3.7.7)

- Rectified linear unit (ReLU):

  $$\sigma(h) = \max(0, h), \ \sigma'(h) = \begin{cases} 1, & h > 0 \\ 0, & h \leq 0 \end{cases}$$

(3.7.8)

Note: ReLU is non-differentiability at $h = 0$*; in practice, the derivative at this point is arbitrarily defined as either 0 or 1.*

- SoftMax (for a vector $\mathbf{H} = [h_i, \dots h_n]^\top$):

$$\sigma_i(\mathbf{H}) = \frac{e^{h_i}}{\sum_{j=1}^{n} e^{h_j}}, \quad \frac{\partial \sigma_i}{\partial h_j} = \sigma_i(\mathbf{H})\left(\delta_{\text{ij}} - \sigma_j(\mathbf{H})\right),$$

(3.7.9)

here $\delta_{\text{ij}} = 1$ *if* $i = j$, *else* 0. SoftMax ensures $\sum_i \sigma_i(\mathbf{H}) = 1$.

3.7.3 Optimisation

The parameter space ϑ of a multilayer network is high-dimensional and the loss surface $\mathcal{J}(\vartheta)$ is non-convex, with many local minima and saddle points. As a result, analytical solutions for the global minimum are intractable.

To address this, gradient based optimisation algorithms are used to iteratively adjust the parameters in the direction of the negative gradient, reducing the loss function over time.

For a dataset $D = \left\{\left(\mathbf{X}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$, *and per sample loss* $\mathcal{L}\left(\mathcal{F}\left(\mathbf{X}^{(i)}; \vartheta\right), y^{(i)}\right)$, the empirical loss over the dataset (also called mean loss) is defined as:

$$\mathcal{J}(\vartheta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}\left(\mathcal{F}\left(\mathbf{X}^{(i)}; \vartheta\right), y^{(i)}\right)$$

(3.7.10)

In batch gradient descent, the parameters are updated using the full dataset at each iteration $\sqcup$, according to:

$$\vartheta_{\sqcup+1} = \vartheta_{\sqcup} - \eta \nabla_{\vartheta} \mathcal{J}(\vartheta_{\sqcup}),$$

(3.7.11)

where $\eta > 0$ is the learning rate, which control the step size of the update.

3.7.4 Stochastic gradient descent (SGD):

Stochastic gradient descent (SGD) is a variant of gradient descent that approximates the full gradient by computing it over a randomly selected mini batch of training data. This allows for faster updates and requires less memory.

A single pass over the entire dataset is referred to as an epoch. Mini-batch size is the number of examples used to estimate the gradient in one update. Iteration is one parameter update step, using one mini-batch. In each batch there are $\left\lceil \frac{N}{|\mathcal{B}|} \right\rceil$ iterations.

At iteration $\sqcup$, a mini batch $\mathcal{B}_{\sqcup} \subset D$ *of size* $|\mathcal{B}_{\sqcup}| = |\mathcal{B}|$ is sampled. The loss over this mini batch is:

$$\mathcal{J}_{\mathcal{B}_{\sqcup}}(\vartheta) = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{X},\ y) \in \mathcal{B}_{\sqcup}}^{N} \mathbf{l}\left(\mathcal{F}(\mathbf{X}; \vartheta), y\right),$$

(3.7.12)

And update

$$\vartheta_{\sqcup+1} = \vartheta_{\sqcup} - \eta \nabla_{\vartheta} \mathcal{J}_{\mathcal{B}_{\sqcup}}(\vartheta_{\sqcup}),$$

(3.7.13)

the weights and bias are updated as:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{1}{|\mathcal{B}_{\sqcup}|} \sum_{\mathbf{X} \in \mathcal{B}_{\sqcup}} \frac{\partial \mathcal{J}}{\partial W^{(l)}}, \quad b^{(l)} \leftarrow b^{(l)} - \eta \frac{1}{|\mathcal{B}_{\sqcup}|} \sum_{\mathbf{X} \in \mathcal{B}_{\sqcup}} \frac{\partial \mathcal{J}}{\partial b^{(l)}},$$

(3.7.14)

where η is the learning rate.

Since $\mathcal{B} \ll N$, *SGD significantly reduces the computational burden per iteration. Moreover, the stochastic noise introduced by sampling can help the optimiser escape shallow local minima or saddle points and often leads to better generalisation.*

3.7.5 Adaptive Moment (Adam) estimation:

Adaptive Moment Estimation (Adam) is an optimisation algorithm that extends stochastic gradient descent by computing adaptive learning rates for each parameter. While SGD uses the gradient of the current mini batch with a fixed learning rate, Adam incorporates momentum and gradient scaling by tracking both the first and second moments of the gradients. This often leads to faster convergence and reduces the need for manual hyperparameter tuning.

For parameter $\vartheta_{\sqcup}$at each iteration $\sqcup$, let mini-batch gradient be:

$$\mathrm{g}_{\sqcup} = \nabla_{\vartheta}\, \mathcal{J}_{\mathcal{B}_{\sqcup}} \left(\vartheta_{\sqcup}\right).$$

(3.7.15)

Adam maintains two exponentially decaying averages:

First moment (mean of gradient):

$$m_{\sqcup} = \beta_1 m_{\sqcup-1} + (1-\beta_1)\, g_{\sqcup},$$

(3.7.16)

Second moment (uncentered variance of gradient):

$$v_{\sqcup} = \beta_2 v_{\sqcup-1} + (1-\beta_2)\, g_{\sqcup}^2,$$

(3.7.17)

Where $g_{\sqcup}^2$ denotes the element-wise square of $g_{\sqcup}$. Typical default values are $\beta_1 = 0.9,\ \beta_2 = \ 0.999$.

Since both m_0 and v_0 initialise to zero, the estimates are biased toward zero at early iterations. Adam applies bias correction as follows:

$$\widehat{m}_{\sqcup} = \ \frac{m_{\sqcup}}{1\ -\ \beta_1^{\sqcup}}$$
$$\widehat{v}_{\sqcup} = \ \frac{v_{\sqcup}}{1\ -\ \beta_2^{\sqcup}}$$

(3.7.18)

Finally, the parameter is updated as:

$$\vartheta_{\sqcup+1} = \ \vartheta_{\sqcup}\ -\quad \eta \frac{\widehat{m}_{\sqcup}}{\sqrt{\widehat{v}_{\sqcup}}\ +\ \epsilon},$$

(3.7.19)

where η is base learning rate and ϵ is very small for example 10^{-8} for numerical stability to prevent division by zero.

By combining momentum $m_{\sqcup}$ and scaling $v_{\sqcup}$, Adam typically converges faster and requires less manual hyperparameter tuning than SGD.

3.7.6 Convolutional neural network (CNN)

Convolutions neural networks are deep learning networks that replace a Fully connected layer in FFN with a convolution and a pooling layer. Convolution neural networks have more convolution layers instead of fully

connected layers. The input layer of Convolution neural networks is a tensor $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$, *where* $C = 3$ for image RGB or else 1 for grey scale image.

A convolution layer performs a convolution operation. A convolutional operation is performed by sliding a kernel ***K,*** which is a learnable weight tensor of fixed special size, over the image.

Let input to a layer be an activation tensor:

$$a^{(l-1)} \in \mathbb{R}^{H\times W\times C},$$

(3.7.20)

where H, W are its height and width, and C the number of channels. A convolutional layer applies F learnable filters:

$$K^{(l,f)} \in \mathbb{R}^{k_h\times k_w\times C}, f = 1,\ldots,F,$$

(3.7.21)

each with its own bias $b^{(l,f)}\in \mathbb{R}$, stride $s^{(l)}$ and padding $p^{(l)}$. The resulting pre-activation feature map:

$Z^{(l,f)} \in \mathbb{R}^{H^{'}\times W^{'}}$

(3.7.22)

is given by discrete convolution:

$$Z^{(l,f)}[i,j] = \sum_{u=1}^{k_h}\sum_{v=1}^{k_w}\sum_{c=1}^{C} K(l,f)\ [u,v,c]\, a^{(l-1)}\left[s^{(l)}i+u-p^{(l)}-1, s^{(l)}j+v-p^{(l)}-1,\ c\right] + b^{(l,f)}$$

For $i = 1, \ldots, H^{'}$ and $j = 1,\ldots,W^{'}$, *where:*

$$H^{'} = \left[\frac{H+2p^{(l)}-k_h}{s^{(l)}}\right], W^{'} = \left[\frac{W+2p^{(l)}-k_w}{s^{(l)}}\right]$$

(3.7.23)

Finally, each feature map is passed through an activation:

$a^{(l,f)}[i,j] = \sigma^{(l)}(Z^{(l,f)}[i,j])$, $a^{(l,f)} = \left\{a^{(l,1)}, a^{(l,2)}, \ldots, a^{(l,F)}\right\}$

(3.7.24)

After activation maps are produced by the convolutional layer, a pooling layer applies a fixed, non-learnable operation (e.g. max or average) over small, non-overlapping windows to reduce spatial resolution while keeping the number of channels unchanged.

Let

Window size $k_p \times k_p$,

Stride S_p (*usually* $S_p = k_p$),

padding p_p

The pooled feature map $P^{(l,f)} \in \mathbb{R}^{H^{''}\times W^{''}}$ is

$$P^{(l,f)}[i,j] = \begin{cases} \max_{0\le u\le k_p}\ \max_{0\le v\le k_p}\ a^{(l,f)}[S_p i+u-p_p, S_p j+v-p_p], & (\max-\text{pool}), \\ \frac{1}{{k_p}^2}\sum_{u=0}^{k_p-1}\sum_{v=0}^{k_p-1} a^{(l,f)}[S_p i+u-p_p, S_p j+v-p_p], & (\text{avg}-\text{pool}). \end{cases}$$

(3.7.25)

3.7.6.1 Residual network (ResNet)

Very deep CNNs suffer from a vanishing/exploding gradients, making optimisation difficult. ResNet addresses this by adding identity skip-connections, allowing each block to learn a residual function, i.e. the difference between the desired mapping and the identity

A basic residual block for two 3 x 3 convolution layers (weights ${W_1}^{(l)}$, ${W_2}^{(l)}$ and biases ${b_1}^{(l)}$, ${b_2}^{(l)}$, the block output is:

$a^{(l+1)} = a^{(l)} + R^{(l)}\left(a^{(l)}; \vartheta^{(l)}\right)$,

$R^{(l)}\left(a^{(l)}; \vartheta^{(l)}\right) \;=\; \sigma\left({W_2}^{(l)}\sigma\left({W_1}^{(l)}a \;+\; {b_1}^{(l)}\right) +\; {b_2}^{(l)}\right)$,

(3.7.26)

Where $\sigma \;=\; ReLU$.

If the special dimension of channel count of the residual block input and the output changes (e.g. by stride 2), the skip path uses a 1x1 projection $E^{(l)}$ so that:

$$a^{(l+1)} = E^{(l)}a^{(l)} + R^{(l)}\left(a^{(l)}; \vartheta^{(l)}\right)$$

(3.7.27)

The identity term keeps gradients flowing even in very deep network (e.g., 152 layers), enabling easier optimisation and better performance.

3.7.6.2 DenseNet

The DenseNet further alleviates the gradient vanishing problem by connecting every layer to every subsequent layer with in same dense block. The DenseNet concatenates the feature maps instead of summing them like in ResNet. Like ResNet, DenseNet has a dense block containing multiple convolution layers.

For a DenseNet with a growth rate q :

$$a^{(l)} \;=\; \sigma\left(W^{(l)}\left[a^{(0)} \parallel a^{(1)} \parallel \ldots \parallel\; a^{(0)}\right] + b^{(l)}\right)$$

(3.7.28)

Where $\parallel$ denotes channel-wise concatenation.

In DenseNet, C_0 denotes the number of channels in the input tensor to the first layer of dense block. After k layers the base channel dimension C_0 grows to $C_0 + \; kq$. *This shows each layer in DenseNet add q* new feature maps. The directly connected layers encourage gradient flow and implicit deep supervision to the network.

3.7.6.3 EfficientNet

ResNet scales depth by stacking additional residual blocks, while DenseNet scales width by increasing channel dimensions along with depth. EfficientNet introduces compound scaling, jointly adjusting depth, width, and input resolution with a single coefficient $\phi \in \; \{0, 1, \; 2, \ldots\}$.

Starting from a baseline network B, a scaled variant Bϕ applies:

Depth scaling: α^{ϕ}, width scaling β^{ϕ}, and resolution γ^{ϕ}

And subject to the cost constraint $\alpha\beta^2\gamma^2 \approx \; 2$, $\alpha \geq 1$, $\beta \geq$, $\gamma \geq 1$ chosen empirically.

The resolution scaling means input height H and width W *of an image is scaled as $\gamma^{\phi}H$ and $\gamma^{\phi}W$ correspondingly.*

Each step $\phi \rightarrow \phi + 1$ roughly doubles the FLOPs (computation cost) while giving a predictable accuracy gain.

The compound scaling has the advantage that a single parameter replaces three independent search axes, reducing the network design time while delivering near optimal accuracy-FLOP trade-offs compared to ResNet or DenseNet.

3.7.6.4 Vision Transformer (ViT)

Like the evolution from ResNets to DenseNet and EfficientNet, the field of computer vision has seen a paradigm shift with the introduction of Transformer-based architectures, moving beyond the traditional convolutional centric models. The Vision Transformer (ViT) represent a pivotal advancement in this direction.

The Vision Transformer (ViT) adapts the Transformer architecture, originally designed for sequence-to-sequence tasks, for image classification. It moves away from the inductive biases of convolutions by treating an image as a sequence of patches.

An input image is first divided into N flattened patches $\{x_p^1, x_p^2, \ldots, x_p^N\}$. These are linearly projected into D-dimensional embeddings. A learnable class token embedding, x_{class}, is prepended to the sequence, and learnable positional embeddings, E_{pos}, are added to provide spatial information. This creates the input sequence for the Transformer encoder, $z^{(0)}$:

$$z^{(0)} = \left[x_{\text{class}}; x_p^1 E; x_p^2 E; \ldots; x_p^N E\right] + E_{\text{pos}}$$

(3.7.29)

Where E is the learnable embedding projection matrix.

The sequence $z^{(0)}$ is then passed through L identical encoder layers. Each layer consists of a Multi-Head Self-Attention (MSA) block and a Multi-Layer Perceptron (MLP) block, with Layer Normalization (LN) and residual connections. The operation for layer l is:

$$\widehat{z}^{(l)} = \text{MSA}\left(\text{LN}\left(z^{(l-1)}\right)\right) + z^{(l-1)}$$

(3.7.30)

$$z^{(l)} = \text{MLP}\left(\text{LN}\left(\widehat{z}^{(l)}\right)\right) + \widehat{z}^{(l)}$$

(3.7.31)

The MSA mechanism's global attention, which relates all patches to each other, allows ViT to model long-range dependencies. However, its computational complexity scales quadratically with the number of patches, making it less efficient for high-resolution images.

3.7.6.5 Swin Transformer

The Swin Transformer introduces a hierarchical architecture and a more efficient attention mechanism to make Transformers a viable general-purpose vision backbone. It re-introduces locality as an inductive bias by computing self-attention within local, non-overlapping windows.

To enable information flow across these windows, the Swin Transformer alternates between two types of attention in consecutive blocks: standard Window-based MSA (WMSA) and Shifted-Window MSA (SWMSA). The SW-MSA block displaces the window partitioning from the previous layer, effectively creating new connections between patches. This allows the model to learn features that span across window boundaries.

A pair of consecutive Swin Transformer blocks, forming the core of the architecture, operates on the input tokens $z^{(l-1)}$as follows:

$$\widehat{z}^{(l)} = \text{WMSA}\left(\text{LN}\left(z^{(l-1)}\right)\right) + z^{(l-1)}$$

(3.7.32)

$$z^{(l)} = \text{MLP}\left(\text{LN}\left(\widehat{z}^{(l)}\right)\right) + \widehat{z}^{(l)}$$

(3.7.33)

$$\widehat{z}^{(l+1)} = \text{SWMSA}\left(\text{LN}\left(z^{(l)}\right)\right) + z^{(l)}$$

(3.7.34)

$$z^{(l+1)} = \text{MLP}\left(\text{LN}\left(\widehat{z}^{(l+1)}\right)\right) + \widehat{z}^{(l+1)}$$

(3.7.35)

This combination of a hierarchical structure and the shifted-window attention mechanism achieve linear computational complexity with respect to the number of patches, enabling strong and efficient performance across a range of vision tasks.

3.7.7 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a feedback-based architecture designed for processing time series or sequential data $X = \left(x^{(1)}, x^{(2)}, \ldots, x^{(T)}\right)$, where the output at time step t depends on the prior elements in the sequence. The network maintains a hidden state $h^{(t)} \in R^{d_h}$, which serves as an internal memory of the sequence history.

The core of the RNN is its recursive definition, operating over T time steps. At each time step t $\in \{1, \ldots, T\}$, the hidden state $\mathbf{h}^{(t)}$ is updated using the current input $x^{(t)} \in R^{d_x}$ and the previous hidden state $h^{(t-1)}$:

$$h^{(t)} = \sigma_h \left(W_{\text{hh}} h^{(t-1)} + W_{xh} x^{(t)} + b_h \right)$$

(3.7.36)

The network's output $\widehat{y}^{(t)} \in R^{d_y}$ (e.g., the class probabilities) is then computed from the hidden state:

$$z^{(t)} = W_{hy} h^{(t)} + b_y$$

$$\widehat{y}^{(t)} = \sigma_y \left(z^{(t)} \right)$$

(3.7.37)

The initial hidden state $h^{(0)}$ is typically initialized to a zero vector.

The overall parameter set is $\vartheta = \{W_{\text{hh}}, W_{\text{xh}}, W_{\text{hy}}, b_h, b_y\}$, where:

- $W_{\text{hh}} \in R^{d_h \times d_h}$ and $W_{xh} \in R^{d_h \times d_x}$ are the weights for the state update.
- $W_{hy} \in R^{d_y \times d_h}$ are the weights for the output layer.
- σ_h is the hidden state activation function (e.g., $\tan h$).
- σ_y is the output activation function (e.g., SoftMax).

The RNN processes sequential data by maintaining a hidden state that captures information from previous time steps and influences future ones. This hidden state acts as memory, updated at each time step using the current input and the previous hidden state. When unrolled, an RNN resembles a feedforward neural network operating across time. The network is trained using Backpropagation Through Time (BPTT), which calculates gradients by unrolling the computation graph across all time steps.

Internally, RNNs use two operations on input: i2o, which computes output class probabilities using SoftMax, and i2h, which updates the hidden state (refer 9.2 for detailed understanding of how RNNs process the information).

3.7.7.1 LSTM:

Long Short-Term Memory (LSTM) networks are an extension of an RNN. The LSTM unit consists of four gated cells, namely the Input gate, the Output gate, the Memory gate and the Forget gate. The non-linearity is introduced with the use of non-linear activations, which act as gates. The activation includes three sigmoid and one Tanh shown in Figure 3.4 and Figure 3.5.

These gates control information propagation to the next cells. Each LSTM cell has three inputs $\mathbf{C_{t-1}}$, $\mathbf{h_{t-1}}$, $\mathbf{X_t}$ and two outputs $\mathbf{C_t}$ and $\mathbf{h_t}$. At every time step $\mathbf{t}$, $\mathbf{h_t}$ is the hidden state, $\mathbf{C_t}$ is the cell state or memory, $\mathbf{X_t}$ is the current input whereas $\mathbf{C_{t-1}}$ and $\mathbf{h_{t-1}}$ are the output of the cell at time step **t-1**. The operation is repeated for all the input sequences. This arrangement thoroughly processes input at every timestep by an LSTM unit before propagating the learned information for the next time step which is known as a forward pass.

Many variations of LSTM exist which are created by changing the original design of an LSTM cell, for example, a peephole LSTM has a connection that allows the knowledge of hidden cell state H at every time instant. The output of peephole type LSTMs at each timestep can help in obtaining an output 2D tensor of shape $T \times H$ where T is total timesteps.

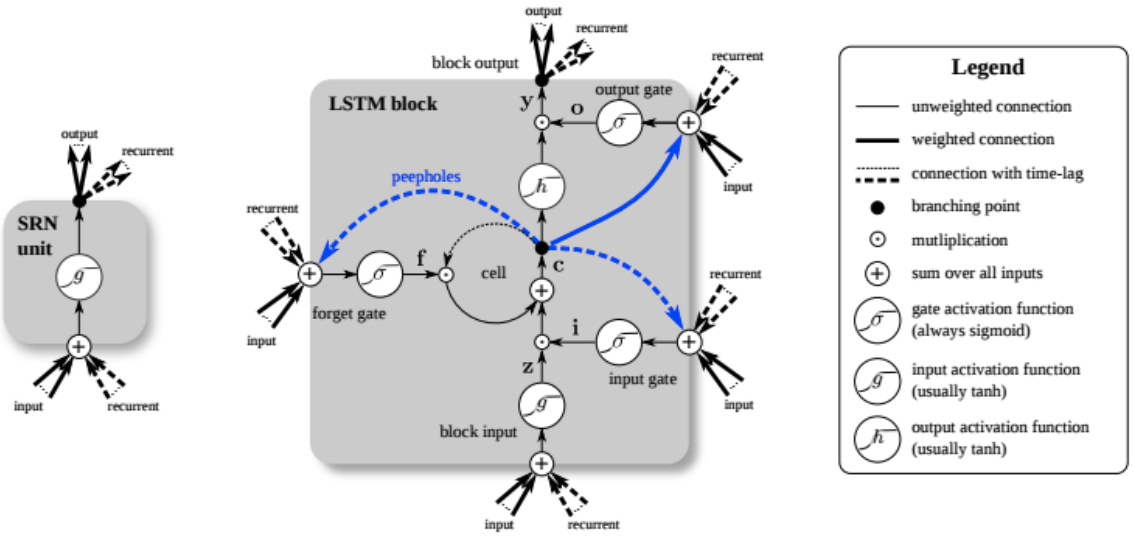


Figure 3.4: Comparison of SRN-Simple Recurrent Unit and LSTM- Long Short-Term Memory unit. (source (Greff et al. 2016)).

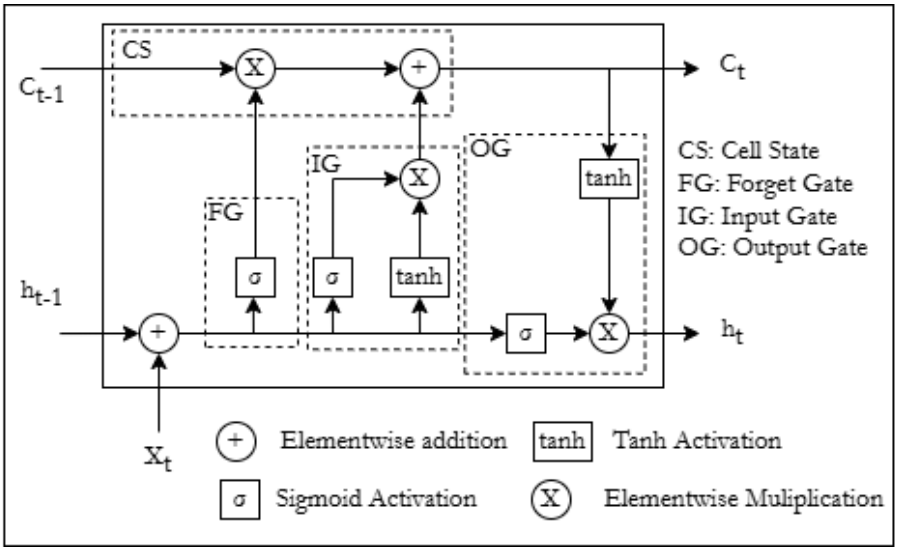


Figure 3.5: LSTM internal operations

3.7.7.2 BLSTM

A Bidirectional LSTM processes the information in the forward time sequence as well as the reverse time sequence and then couples the output to capture past and future dependency at each time step (refer Figure 3.6). This ability of BLSTM helps in capturing forward and backward context for each word in a sentence in case of text processing.

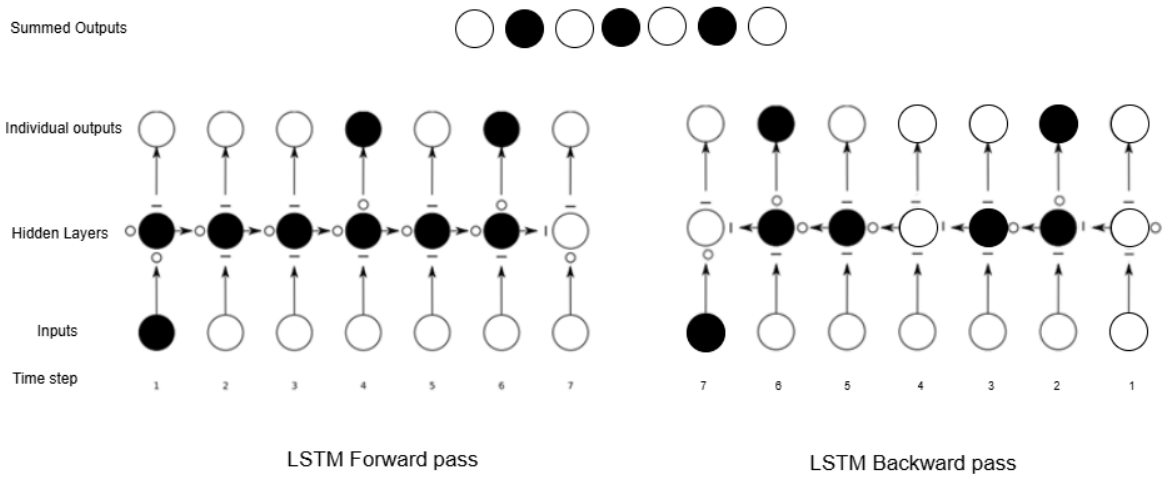


Figure 3.6: A simple Bidirectional LSTM cell as a hidden layer, where O and – represent the open and closed states of an individual LSTM cell, respectively.

The output from both forward and backward passes obtained at the end of the input sequence can be added/ multiplied or concatenated to get a 1D vector $1 \times H$ or $1 \times 2H$ respectively, where H is the number of hidden units in an LSTM cell.

The implementation of our LSTM layer is done using the LSTMCell layer implementation from Keras Python Package [38] and the Bidirectional layer implementation of Keras Python Package [38] is used for the Bidirectional processing.

3.8 Textual Images

3.8.1 Image data

Before discussing the Textual Image classification for extracting sentiment, it is important to understand the image classification process. For that we need to understand what an image is.

An image is a structured grid of pixels, where each pixel represents the intensity of colour at a specific location. In digital images, the colour intensity is typically quantified using values ranging from 0 *to* 255, where 0 represents the lowest intensity (black) and 255 represents the highest intensity (white or full colour intensity, depending on the colour channel).

A grayscale image is a type of image that contains only one-colour channel and is often referred to as a monochromatic image. Each pixel in a grayscale image holds a single intensity value, making it a 2D representation where only brightness variations are captured, without any colour information.

A colour images such as RGB images have three separates *channels* (*Red*, *Green*, *and Blue*) which combine to represent a full range of colours. These are stored in a 3D array structure with dimensions representing the height, width, and number of channels. For example, an RGB image would have a shape of 256x128x3 were 256 represent the height of the image, 128 represent width of the image and 3 represent colour channel red, blue and green.

3.8.2 Image classification

Image classification is a fundamental task in computer vision where the goal is to assign a label or category to an image based on its content. This task is widely used in applications such as medical imaging [132], object recognition [133], and automated surveillance [134]. Modern classification systems rely heavily on neural networks, particularly Convolutional Neural Networks (CNNs), which are capable of automatically extracting meaningful features from images and associating them with target classes.

The output of such models is often a probability distribution over all possible classes, and the class with the highest probability is selected as the model's prediction. For example, given an image of a cat, the model might return: {"cat": 0.87, "dog": 0.10, "car": 0.03}, and so classify the image as "cat."

3.8.3 Images from vectorised text and classification

An Image equivalent representation of Text is essential to process the text information using computer vision models effectively. The image based neural model has a unique input, so it is important to represent text in a similar image format.

Word embeddings are high-dimensional word representations as discussed in prior section. In a matrix form, each embedding shape is $1 \times Embedding_size$ where $Embedding_size$ is the number of dimensions chosen to represent the words. Every word in a text document has unique embedding and collectively they form a $2D$ matrix shape $N \times Embedding_size$, where 'N' is the total number of words in the text document.

One can simply reshape this vector to $N \times Embedding_size \times 1$ (number of elements remain same), then N can be considered the height of the image, $Embedding_size$ can become the width of the image, and 1 can represent a single channel, thus forming a greyscale image equivalent tensor.

The GloVe embeddings are used in this work have 300 dimensions. As a tweet has a character limit for a post and the embedding use word vectors so, an image of $Max_len \times Embedding_size(300) \times Channel$ (1) can be formed using the pre-trained Word embeddings. Max_len is the maximum no of words in a tweet in an entire dataset. Max_len for Sentiment140 dataset, in most of our experiments, varies between 30-38 and depends on the different data subset size.

Textual Image classification involves forming Textual Images through neural textual encoding and then classifying them using image classification models. The encoder transforms the input word embeddings into a two-dimensional matrix, in which the rows correspond to the words or tokens in the text and the columns correspond to embedding or learned feature dimensions. This representation can capture temporal, sequential and contextual dependencies. Depending on the architecture, this neural encoding can perform feature extraction, dimensionality reduction or dimensionality expansion.

The neural network is trained using the sentiment labels rather than a separate target image. For each labelled text, the encoder produces a Textual Image, the image classifier predicts a sentiment class, and a classification loss measures the difference between this prediction and the known label. During backpropagation, the loss gradient is propagated from the classifier through the Textual Image representation and into the trainable text encoder. An optimiser then updates the model parameters so that the intermediate Textual Images become more useful for discriminating sentiment classes. Thus, a Textual Image is a learned intermediate representation, rather than an image trained to resemble a natural image or a separately defined visual target.

Various neural models, including CNNs, RNNs and Transformer-based networks, can be used to encode textual information. In this work, several such methods are investigated for generating and classifying Textual Images.

3.8.4 BLSTM as Textual Image generator

Bidirectional Long short Memory (BLSTM) can be utilised to generate high quality Textual Images [27] where the text documents, represented by a collection of word embeddings, are non-linearly transformed into greyscale image equivalent using this network.

The hidden cell state sequence at each time step for a both forward and backward LSTM for each input word embedding is obtained by setting $return_sequence = True$ and a point wise addition is performed to obtain the final sequence. This provides a $2D$ image of Shape ($T \times H$). The tensor is converted into a 3D tensor ($T \times$ H $\times$ 1), to represent this text document representation as an 3D image ($Height \times$ Width $\times Channel$), where Channel $=$ 1 since it is a grey scale image. Which is then classified using a Computer Vision model.

3.8.5 BLSTM-2DCNN

The architecture in BLSTM-2DCNN [27] (refer Figure 2.2) is an example of creating text images (with a BLSTM) which a 2D CNN can be used to extract features. The name of the architecture is self-descriptive and represents two different sequential neural architectures, that is, it starts with the BLSTM, then is followed by the 2D CNN. The BLSTM in BLSTM-2DCNN captures word dependencies based on the words following and proceeding the word under observation within the text input, which is in the form of word embeddings (an in-depth description is in the next section). BLSTM, unlike RNN, captures temporal dependencies in two directions. The 2D CNN block contains a single 2DCNN layer followed by a 2D max-pooling layer, like the use of a CNN in computer vision tasks. The convolution layer followed by a pooling layer helps accumulate features from the convolution maps, so is used after the convolution layer for the same reason.

Referring to Figure 2.2 (BLSTM-2DCNN), text vector input x represents a 7×3 matrix, where 7 is the length of the sentence and 3 is the *Embedding_size* of each word vector, which is fed to a Bidirectional LSTM with 5 hidden units/cells. In the actual implementation, the embedding size will be much bigger than this. The outputs of both LSTM (forward and backwards) have a shape of 1×5 (5 being the number of hidden units) at each timestep (timestep is equivalent to the number of words in a sentence), and these 1×5 tensors from both are added elementwise. Therefore, in the example after 7 timesteps a 7×5 output tensor is obtained. This 7×5 tensor is reshaped into $7 \times 5 \times 1$ to represent it as a greyscale image (this step is not depicted in the figure). A convolution operation with a 2×2 kernel is performed on this tensor, which, by sliding over the image with a step of 1, reduces the dimensions to $6 \times 4 \times 1$ and then a 2×2 max pooling operation further shrinks the output to $3 \times 2 \times 1$. Finally, the output layer contains 5 units providing 5 different probabilities concerning each class in the example classification task. There is no relation between 5 class probabilities and the number of hidden units, it is just a convenient number to illustrate.

3.8.6 Our Proposed BLSTM-ResNet for Textual Image Classification

A BLSTM-ResNet was proposed for generating and classifying Textual Images sequentially in the same network. The BLSTM-ResNet is an extension of the BLSTM-2DCNN model [27] where the shallow 2D CNN was replaced with a deeper 2D CNN architecture inspired by ResNet [23]. The BLSTMs helped us to capture the temporal features from the text embeddings and to generate Textual Images which were then classified using the two-dimensional convolutional-based ResNet. Figure 3.7 contains a BLSTM-ResNet block diagram depicting Textual Image generation and its classification.

3.8.6.1 BLSTM-ResNet model implementation

To understand the working of network architecture, a basic example is implemented using the smallest BLSTM-ResNet 3 model with just one skip connection, below is the code block. The full details of all the layers and the residual blocks for variant of BLSTM-ResNet model adopted in this work is explained in Section 4.2.1

The BLSTM-ResNet function is a functional API implementation of a neural network in Keras framework [38], refer to Figure 3.8. The *resnet_layer* function is called to add 2D convolution layers to the network along with batch normalisation and activation based on the parameter values given. The dummy sentiment analysis dataset in Figure 3.9 is used to demonstrate each step in the pipeline of the sample BLSTM-ResNet shown in Figure 3.8. The dummy set is a binary classification task with five examples from each class. Label value 1 represents a positive sentiment and 0 corresponds to negative sentiment. The data is tokenised using the Keras Tokenizer method to obtain an indexed sequence matrix and paddings were added to obtain an equal length of nine (maximum sentence length in entire dataset) in input sequence.

Tokenizer output- Found 47 unique tokens.

Input Data after tokenisation- shape- (10, 9)

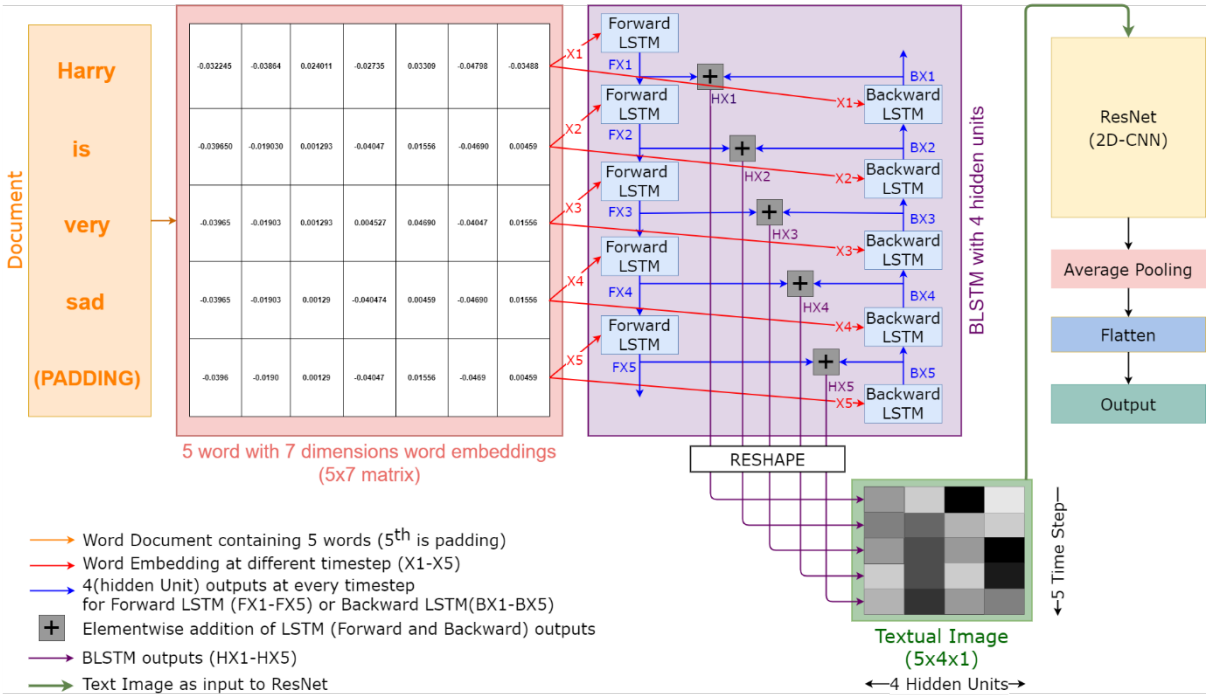


Figure 3.7: A BLSTM-ResNet example, where the Input document has 5 timesteps (document length of 5) because there are 5 elements in the document with each word having a 7-dimensional word Embedding. It is processed using a BLSTM with 4 hidden units (not shown) where 'FXt' and 'BXt' are 1x4(hidden units) outputs of forward and backward LSTM respectively at a given timestep 't'. Four hidden units mean 1x4 output vector at the time 't' from both forward and backward LSTM which are added (+) elementwise to get a single 1x4 BLSTM (HXt) output. The 5(time steps) x4 (hidden units) output matrix obtained from BLSTM is reshaped to 5x4x1 to form a greyscale image equivalent matrix (1 denotes a single channel value for the image). The Textual Image is classified using ResNet-inspired 2D-CNN blocks followed by Average Pooling, Flatten and Output layers.

```
FUNCTION resnet_layer(inputs, num_filters=16, kernel_size=3, strides=1, activation='relu', use_batch_norm=True)
    conv_op = CONV2D(num_filters, kernel_size, strides, padding='same', initializer='he_normal', regularizer=l2(1e-4))
    x = apply(conv_op, inputs)

    IF use_batch_norm THEN
        x = apply(BATCH_NORMALIZATION, x)
    END IF

    IF activation IS NOT NONE THEN
        x = apply(ACTIVATION(activation), x)
    END IF

    RETURN x

FUNCTION create_blstm_resnet(input_shape, num_classes, vocab_size, embedding_dim, maxlen)
    inputs = INPUT(shape=input_shape)

    x = apply(EMBEDDING(vocab_size, embedding_dim, input_length=maxlen, trainable=False), inputs)
    x = apply(DROPOUT(0.5), x)

    lstm_cell = LSTMCELL(units=embedding_dim, activation='softmax', dropout=0.2)
    lstm_bi = apply(BIDIRECTIONAL(RNN(lstm_cell, return_sequences=True, return_state=True, time_major=True), merge_mode='sum'), x)

    x = extract_index(*lstm_bi, 0)

    x_reshaped = apply(RESHAPE(maxlen, embedding_dim, 1), x)

    filters_block = 16
    strides_block = 1
    x_block1 = CALL resnet_layer(inputs=x_reshaped, num_filters=filters_block, strides=strides_block)
    x_block2 = CALL resnet_layer(inputs=x_block1, num_filters=64, strides=strides_block)
    x_block3 = CALL resnet_layer(inputs=x_block2, num_filters=64, strides=1, activation=None)

    x_final = apply(ADD_LAYERS, [x_reshaped, x_block3])
    x_final = apply(ACTIVATION('relu'), x_final)

    x = apply(AVERAGE_POOLING_2D(pool_size=2), x_final)
    x = apply(DROPOUT(0.5), x)
    x = apply(FLATTEN, x)
    outputs = apply(DENSE(num_classes, activation='softmax', kernel_initializer='he_normal'), x)

    model = MODEL(inputs=inputs, outputs=outputs)
    RETURN model
```

Figure 3.8: Pseudo code to implement BLSTM-ResNet architecture

```
data=["i am happy as today is my birthday",
      "i have a smile because I like him",
      "dan had good laughter when he fall",
      "sometime laughter is good for health",
      "the king order everyone to smile and laugh",
      "i am sad as today i lost my dog",
      "i feel low because they hate me",
      "hate in the world will increase",
      "dan hate king as he is always sad",
      "it is hate that made me sad"]
label = [1,1,1,1,1,0,0,0,0,0]
```

Figure 3.9: Example datapoints and labels

$$\begin{bmatrix}
0 & 1 & 6 & 18 & 4 & 7 & 2 & 8 & 19 \\
0 & 1 & 20 & 21 & 9 & 10 & 1 & 22 & 23 \\
0 & 0 & 11 & 24 & 12 & 13 & 25 & 14 & 26 \\
0 & 0 & 0 & 27 & 13 & 2 & 12 & 28 & 29 \\
0 & 15 & 16 & 30 & 31 & 32 & 9 & 33 & 34 \\
1 & 6 & 5 & 4 & 7 & 1 & 35 & 8 & 36 \\
0 & 0 & 1 & 37 & 38 & 10 & 39 & 3 & 17 \\
0 & 0 & 0 & 3 & 40 & 15 & 41 & 42 & 43 \\
0 & 11 & 3 & 16 & 4 & 14 & 2 & 44 & 5 \\
0 & 0 & 45 & 2 & 3 & 46 & 47 & 17 & 5
\end{bmatrix}$$

Each sample shape - (1, 9)

The tokenised dataset is split into 80%-20% training and testing sets. The train set contains 8 samples, and the test set contain 2 samples. The training set is input into the neural network. Figure 3.10 shows the model summary, showcasing various operations performed in the network on each data instance. These operations are described in detail with the first data sample as an example in the following sentences.

The input layer is just inputting the data into the network.

Input_layer shape- (1, 9)

Data[0] - [0.0, 1.0, 6.0,18.0, 4.0, 7.0, 2.0, 8.0, 19.0]

The embedding layer represents each token into a 10 vector long embedding which transforms the input sequence into a higher vector representation

Embedding_layer shape- (1, 9, 10)

The dropout layer drops a few neurons to prevent overfitting.

Dropout_layer shape- (1, 9, 10)

Bidirectional (RNN) has 2 LSTM cells, one for a forward pass and one for a backwards pass. Each cell has 10 hidden units, therefore 10 outputs at each timestep (using peephole LSTM cells) and then added elementwise. Moreover, an additional four outputs vectors are generated, two per LSTM pass, which include the final cell states and hidden states of each LSTM layer. These four values can be used to initialise RNN layers to further train the network.

Bidirectional_layer (*RNN*) *shape*- (1, 9, 10)

Reshape layer adds a single dimension to the existing tensor.

Reshape - (1, 9, 10, 1)

The output of the reshape layer can be considered as a greyscale equivalent and it is the required input shape for a 2D convolution layer. In the Convolution layer, filters are 16, padding is kept the same, and stride is 1.

Conv2d_layer shape- (1, 9, 10, 16)

This creates 16 different channels in the greyscale image using 16 different kernel values in each filter. The tensor dimensions before and after convolution remain the same because of the padding option. The batch normalization and activation function does not alter the shape of the tensor, and these operations are performed to avoid vanishing/exploding grading problems in a deeper neural network. The outputs after the first convolution, batch norm and activation are split into two. On the first branch, the two more similar operations are performed (Based on the BLSTM-ResNet type) and then the outputs from the two branches are added together. The average pooling layer creates a dense representation of each of the 16 images and reduces their dimensionality by half.

Ave*rage_Pooling* shape- (1, 4, 5, 16)

Flatten layer flattens all the output to form a very long fully connected layer

Flatten shape- (1, 320)

The dense layer is used with 2 outputs representing 2 different classes in the dataset.

Dense_layer shape- (10, 1)

During model training, the weights of each layer are updated after each training batch based on the calculated loss and a learning rate parameter. These final weights are then used to predict the class of unseen data.

To observe how the input is transformed through the network, the text embeddings are propagated through each layer using its learned parameters. In the BLSTM layer, the representation is computed recurrently at each timestep through gated updates of the forward and backward hidden and cell states, rather than through a single matrix multiplication. The resulting sequence of BLSTM hidden states provides the transformed vector representation of the input text.

The Figure 3.11 presents examples of various input texts along with the corresponding Textual Images produced after the BLSTM layer. The images look like random noise to the human eyes, but the neural network is capable of learning patterns from these word images and classify them.

The BLSTM-ResNet is a hybrid neural network that first uses a BLSTM architecture to generate a "text image," which is then classified using a 2D-CNN ResNet architecture. After each training batch, the network computes the loss, which is used to update the weights of all layers. These updated weights help produce more informative text images, which are then classified with improved ResNet classifier parameters.

The example above illustrates the core idea behind deeper variants of this network, where additional convolutional operations are introduced by increasing the number of residual blocks in the ResNet component.

3.8.7 The contextualised Textual Images generation using BERT

The GloVe Embeddings are great method to encode textual information and prove beneficial in Textual Image classification for sentiment analysis. However, the embedding for same words regardless of where it occurs in the sentence remains the same. BLSTM are a reasonable network to provide some context awareness in the word embeddings, but their inherent recurrent nature makes them slow to train as they cannot be parallelised like a CNN during training, and they still struggle with memorising long term word dependencies effectively. These drawbacks can be addressed with the use of Transformer based neural network which utilises attention mechanisms to encode word relationships and are parallelisable like CNN networks.

3.8.7.1 Transformers

The Transformer neural network was introduced for sequence-to-sequence text classification tasks like language translation tasks. It consists of encoder and decoder blocks. The language translation model before the introduction of Transformers was either using RNN or CNN-based architectures. Both have inherent limitations. The RNNs cannot be parallelly trained as output at every time step is dependent on the output of the previous time step. This makes RNNs very slow to train. The CNNs are computationally more expensive compared to RNNs when it comes to learning the long-term dependencies which require longer path length when mapping the position in a sequence-to-sequence task.

Transformers consist of multiple encoder-decoder blocks stacked together to process sequential data efficiently. Figure 3.12 presents the Transformer architecture, where input text embeddings are combined with positional encodings to help the model understand both absolute and relative token positions in a sequence.

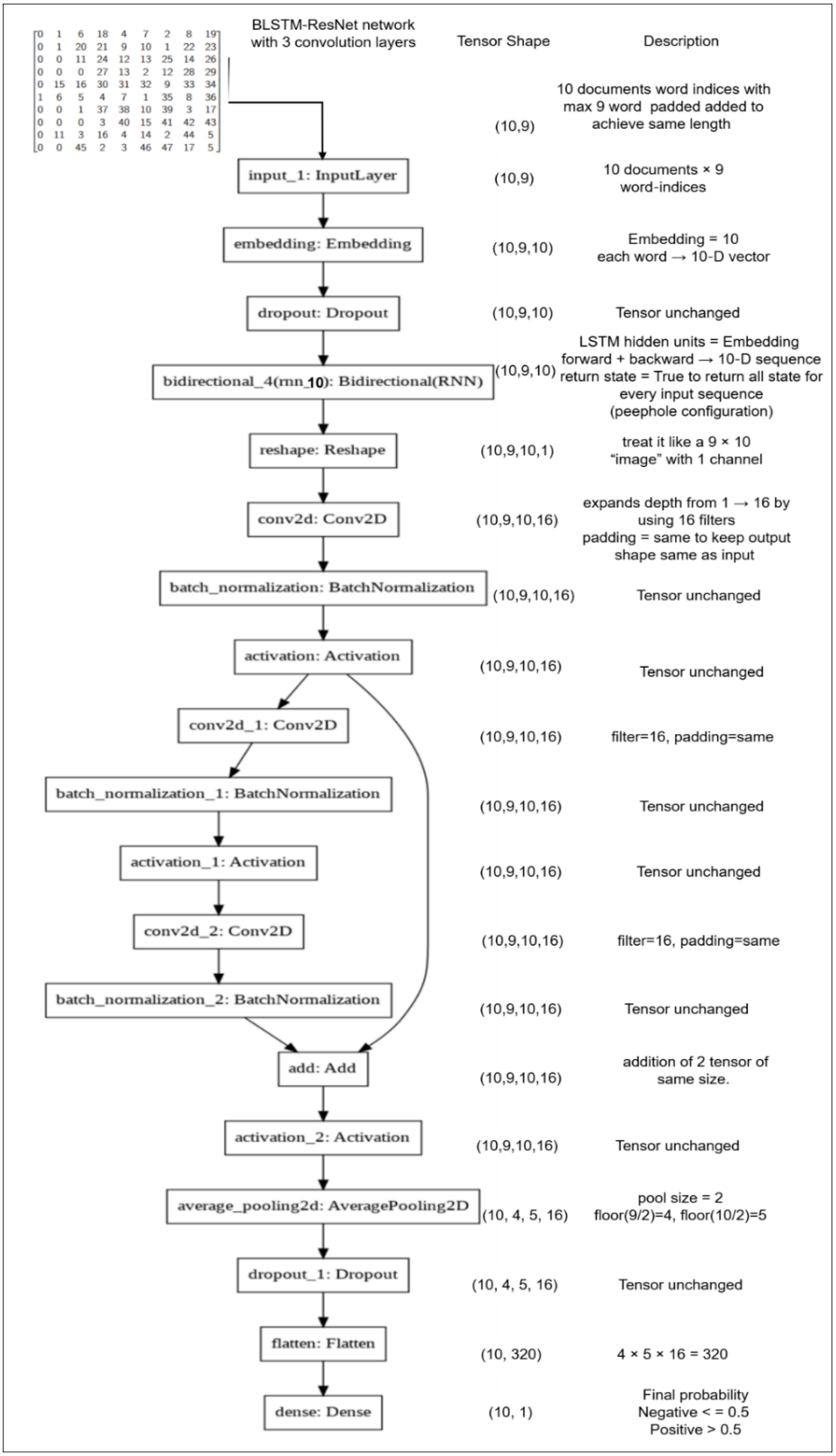


Figure 3.10: Demo BLSTM-ResNet Model summary with one residual layer and one skip connection. Tensor shape at each layer is shown with some description of each layer.



Figure 3.11: Sample Textual Images generated after BLSTM in BLSTM- ResNet network

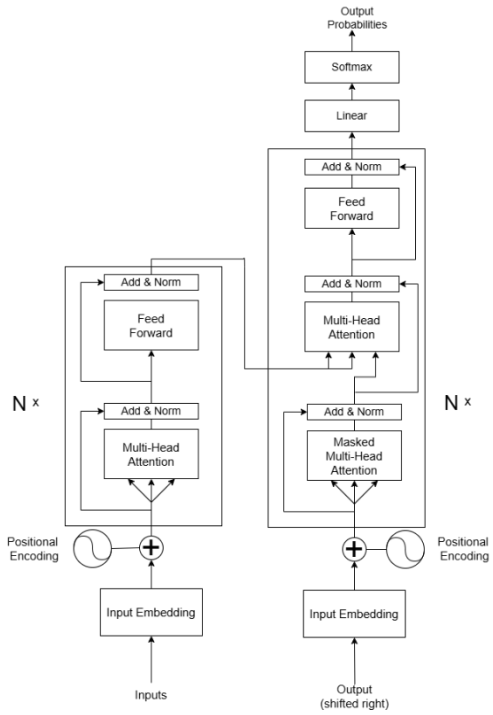


Figure 3.12: Transformer encoder(left) and decoder (right) architecture (Inspired by [91]).

Each encoder block contains multi-head self-attention and a feed-forward network, both followed by skip connections and layer normalization, ensuring stable training. Similarly, the decoder block has two multi-head attention layers and a feed-forward network. The first multi-head attention layer in the decoder attends to the output sequence (e.g., translated language tokens in machine translation), while the second multi-head attention attends to the encoder output, using its query and key vectors while taking the value vectors from the first decoder attention layer.

During training, the output sequence is shifted by one, enabling the model to predict the next word in the sequence. The Transformer model is parameterized by the number of encoder-decoder layers (L), attention heads (A), and hidden units (H) in the linear projection layer, allowing for scalable configurations suited to different NLP tasks.

Let understand the contextualised vectors with an example of sample texts.

Text 1: Please withdraw money from the bank.

Text 2: I prefer an evening walk by the riverbank.

The word embedding for the word "bank" will be the same in both sentences during the preprocessing stage. After performing the scaled dot product attention in multi-head attention, the value vector obtains incorporated information of the neighbouring words which form the contextualised embedding. Therefore, the word "bank" in the first sentence will have the context of a financial institution and in the second sentence will have the context of waterside for the same input word.

The Transformer encoder can be used to obtain the contextualised embeddings. The output from the last encoder block is mostly used as the final contextualised embedding, although the earlier block output can also be used as well.

3.8.7.2 Attention mechanism

To understand the contextualised embedding, we need to understand the use of Attention mechanisms in machine learning. "An attention function can be described as mapping a query and a set of key-value pairs to an output, where query, key and value and output are all vectors"[91].

Attention in a biological setting helps filter and prioritize sensory stimuli, allowing focus on relevant information, like how a predator fixates on prey. William James categorized attention into nonvolitional cues (automatic, based on saliency) and volitional cues (cognitive control) [89]. For example, a red coffee cup naturally attracts attention (nonvolitional), whereas focusing on a book requires conscious effort (volitional). In machine learning, attention mechanisms follow a similar principle, where an attention function maps a query to key-value pairs. The query component represents volitional cues, which interacts with keys (nonvolitional cues) to determine which values (sensory inputs) are most relevant, enabling efficient data processing.

Figure 3.13 shows a simplified attention mechanism. The keys are the nonvolitional input cues from the environment. The query (volitional cue) can interact with keys (nonvolitional cues) which sets the bias for the values (sensory inputs) to get the Output.

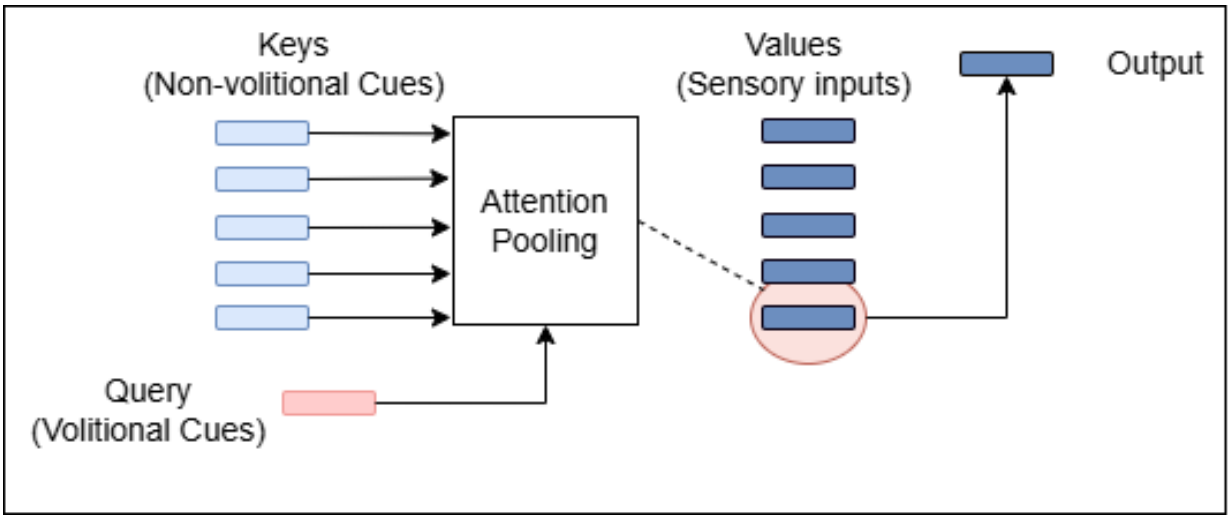


Figure 3.13: Query, Keys and Value in artificial attention mechanism (Inspired by [89]).

3.8.7.3 Scaled dot product attention in Transformers

Attention can be additive or multiplicative. The Transformer network finds multiplicative attention faster and memory efficient. Figure 3.14 depicts the scaled dot product function where the Q (query) and K (key) vectors are multiplied (dot product) and then scaled (divided by $\sqrt{d_k}$ where d_k is the dimension of the Q vector) then the SoftMax is performed and finally multiplied with V (value) vector.

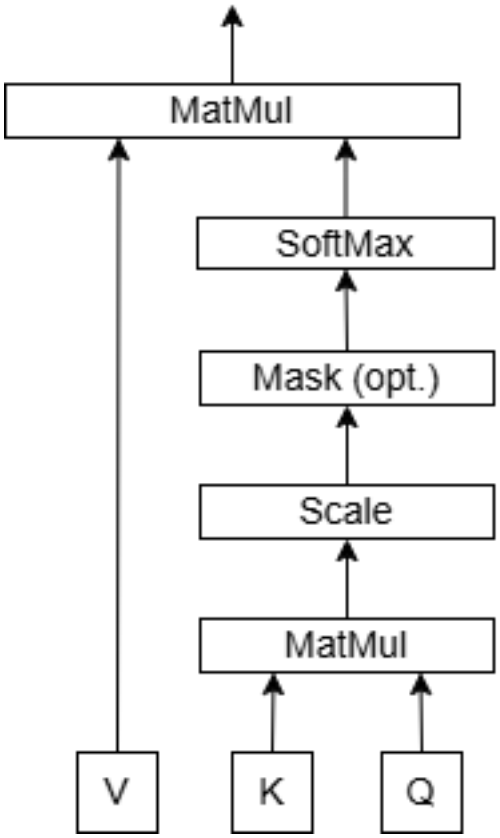


Figure 3.14: Scaled dot product attention (Inspired by [91])

So, the attention function can be written as $\text{Attention}\,(Q,\ K,\ V\ ) = \text{softmax}\left(\frac{\text{QK}^T}{\sqrt{d_k}}\right)V$ where T is the index of the input sequence.

3.8.7.4 Multi-head Attention in Transformer

The Transformer network (Vaswani et al. 2017a) performs several different attention functions h times (where h is the number of attention heads). Each attention output is considered as single head attention. All the h attention outputs were concatenated together to form the multi-head attention function. The Q, K and V are linearly projected to the individual attention heads with a parameterised fully connected layer, refer to Figure 3.15 .

3.8.7.5 Contextualised embeddings

In a Transformer model, every input word embedding has a dimension of 1×512. The output of the multi-head attention unit for every word (value vector) is 1×512. There is a total of 8 heads in the multi-head attention which means each head's output is 1×64 as the output of each head is concatenated. The input embedding token is linearly projected to a fully connected layer with 2048 parameters and has a $1x64$ output vector to form the query, key, and value vectors for multi-head attention. Figure 3.16 depicts the formation of contextualised embedding from the input word embedding in a multi-head attention model.

The value vector obtained after the multi-head attention is referred to as contextualised embedding. These vectors not only leverage the benefits of the word embedding mechanism but also incorporate the context based on the neighbouring word tokens. Every word in a sentence has a context and the context changes with the position and the surrounding words. Figure 3.16 highlights how the attention mechanisms are applied to input word embeddings to generate contextualised word embeddings. The same example of the word "Bank" is shown to have obtained a

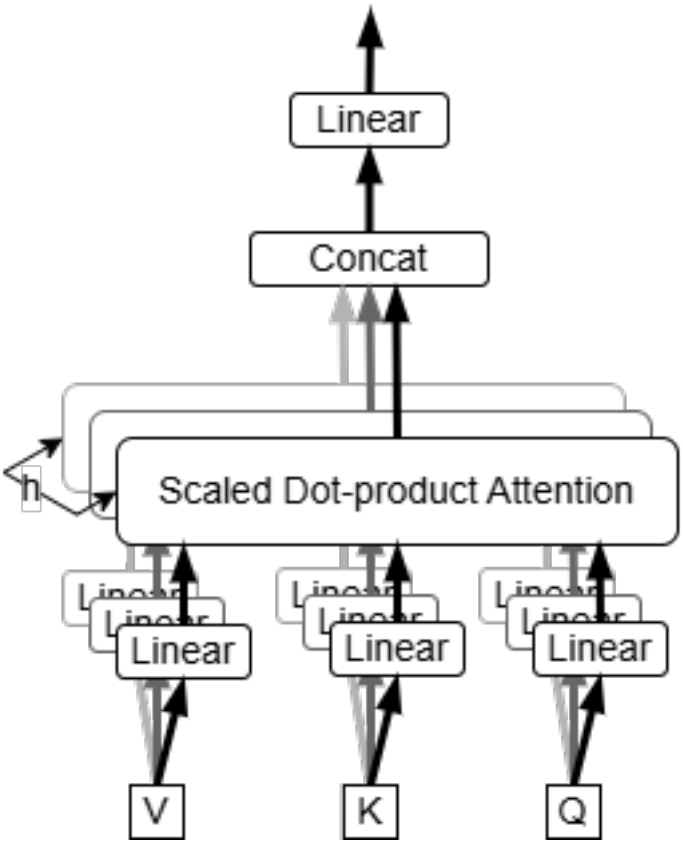


Figure 3.15: multi-head attention with 'h' attention heads (Inspired by [91])

contextualised word embeddings which is influenced by the presence of the word walk and river to understand an association of the word meaning of a riverside instead of a financial institute.

3.8.8 BERT

BERT stands for Bidirectional Encoder Representation from Transformers. BERT [28] uses only the encoder part of the Transformer. The BERT authors aimed to address the limitations of the original Transformer model, which learned only unidirectional representations. These unidirectional representations relied on constrained self-attention and, as stated in the BERT paper, "such restrictions are sub-optimal for sentence-level tasks and could be very harmful when applying fine-tuning-based approaches to token-level tasks". BERT has bidirectional self-attention and incorporates representation learning using two different self-supervised pre-training tasks which are Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

The Wordpiece tokenizer was introduced and popularised with BERT [28] which tokenises the sub-word (word piece) instead of a word as a unit. The Wordpiece tokenizer corpus is generated iteratively by training BERT on a massive Wikipedia database of over 100 languages. In total, the tokenizer contains 119547 unique tokens in hundreds of languages.

BERT at the time of publication introduced 2 variants $\text{BERT}_{\text{Large}}$ and $\text{BERT}_{\text{Base}}$ based on the number of encoder blocks and as the name suggests the former has more trainable parameters and performs better than the later. The $\text{BERT}_{\text{Base}}$ model contains 12 encoder blocks of the Transformer model.

In this work $\text{BERT}_{\text{Base}}$ is chosen for generating contextualised word embeddings due to the computation resource constraints. $\text{BERT}_{\text{Base}}$ can be thought as a feature extractor where the tokenised text document is input to the $\text{BERT}_{\text{Base}}$ model and corresponding word embeddings are generated.

3.8.8.1 WordPiece Tokenisation

The WordPiece Tokenizer is a sub word tokenization method used extensively in transformer-based models like BERT. The tokenizer splits the word into sub words called wordpiece. The method of splitting a word into wordpiece is effective in handling long, new, or rare words by capturing their subcomponents. The tokenizer can

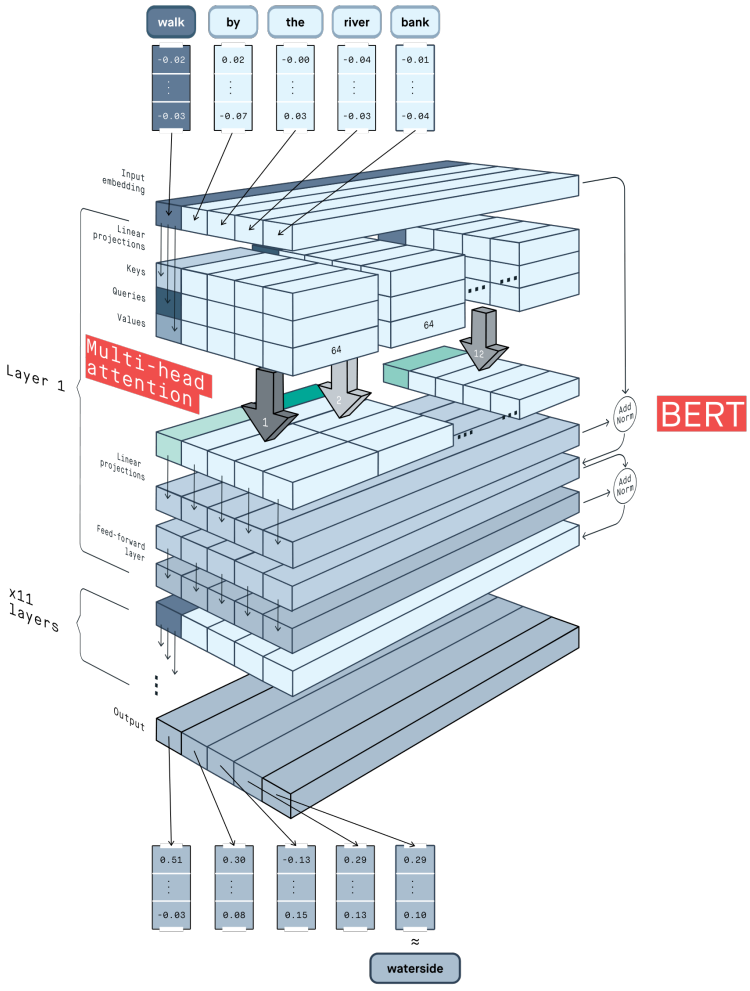


Figure 3.16: BERT processes a sentence to output contextualized embeddings that are more representative of the true meaning (source [135]).

also handle words with special characters. The BertTokenizer method from Transformer python package [102] is used to tokenise the text articles before generating the embeddings.

Examples of wordpiece tokenization:

Text 1: Tokenization splits words.

Tokens: ['token', '##ization', 'splits', 'words', '.']

Text 2 : Unbelievably complicated sentences exist!

Tokens: ['un', '##bel', '##ie', '##va', '##bly', 'complicated', 'sentences', 'exist', '!']

Text 3 : E-mail addresses like test@example.com.

Tokens: ['e', '-', 'mail', 'addresses', 'like', 'test', '@', 'example', '.', 'com', '.']

Text 4 : Covid19 pandemic hit in 2020-2021.

Tokens: ['co', '##vid', '##19', 'pan', '##de', '##mic', 'hit', 'in', '2020', '-', '2021', '.']

Thes token will be mapped to an index and mapped to pretrained Bert embeddings like the GloVe embeddings process.

3.8.9 BERT as the Contextualised Textual Images generator

The Contextualized Textual Images are also grayscale images resembling the Textual Images explored in a previous section. The difference is the use of contextualised embedding while creating the image. The contextualised embeddings are generated in this research work by using the BERT neural model which is based on a Transformer architecture.

The BERT is trained on sequence-to-sequence modelling tasks, but we can fine-tune the model weights for classification tasks to generate class probabilities as well. Once the contextualised embeddings are generated, they can either be reshaped directly to form a Type A contextualised Textual Image or be processed by a BLSTM block to form a Type B contextualised Textual Image. Images obtained from both methods are collectively called Contextualised Textual Images in this work. A Type A image has dimensions of (document length)x(embedding dimension)x1, whereas a Type B image has dimensions of (document length)x(BLSTM hidden units)x1. Figure 3.17 depicts a pipeline of a contextualised Textual Image generation with BERT and BERT plus BLSTM.

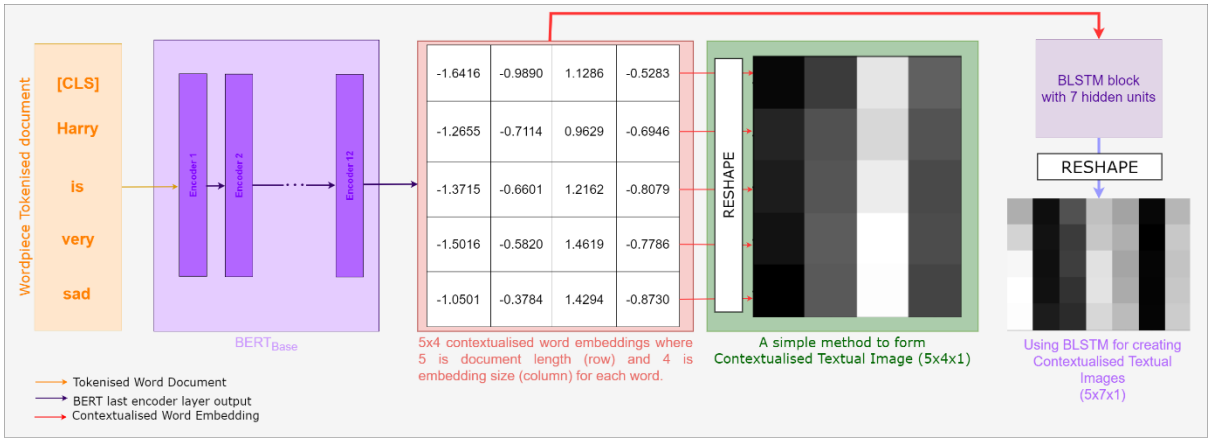


Figure 3.17: A simple text document with a length of 5 words is selected where [CLS] is a special token which signifies that the task is a classification task. This text document is tokenised and given as input to BERTBase. The contextualised embeddings are generated using BERTBase as a feature extractor and the output of the last encoder layer is used. Two different methods for creating contextualised Textual Images are depicted. The BERTBase output can be reshaped directly into a Type A greyscale contextualised Textual Image, or it can be processed by a BLSTM block and reshaped into a Type B contextualised Textual Image.

3.8.10 Interpolation

In many real-world applications, particularly during the preprocessing stage of machine learning projects involving image data, it is often necessary to resize images to standardise their dimensions prior to model training. This

resizing is typically carried out using interpolation techniques, which estimate and assign new pixel values when the spatial dimensions of an image are altered.

When the resized image contains more pixels than the original, the process is referred to as upscaling. Conversely, if the resized image contains fewer pixels, it is referred to as downscaling. Interpolation plays a crucial role in both cases, helping to preserve the visual quality and structural consistency of the image as much as possible.

For example, with interpolation we can resize a small (2x2) greyscale image to a (4x4) greyscale image by upsampling the pixels. Since images are grids of pixels, resizing involves adding or removing pixels while preserving the overall visual structure.

If two points are given $(x_0,\ f(x_0))$ and $(x_1,\ f(x_1))$ the interpolation aims to estimate a value $f(x)$ where $x_0 < x < x_1$

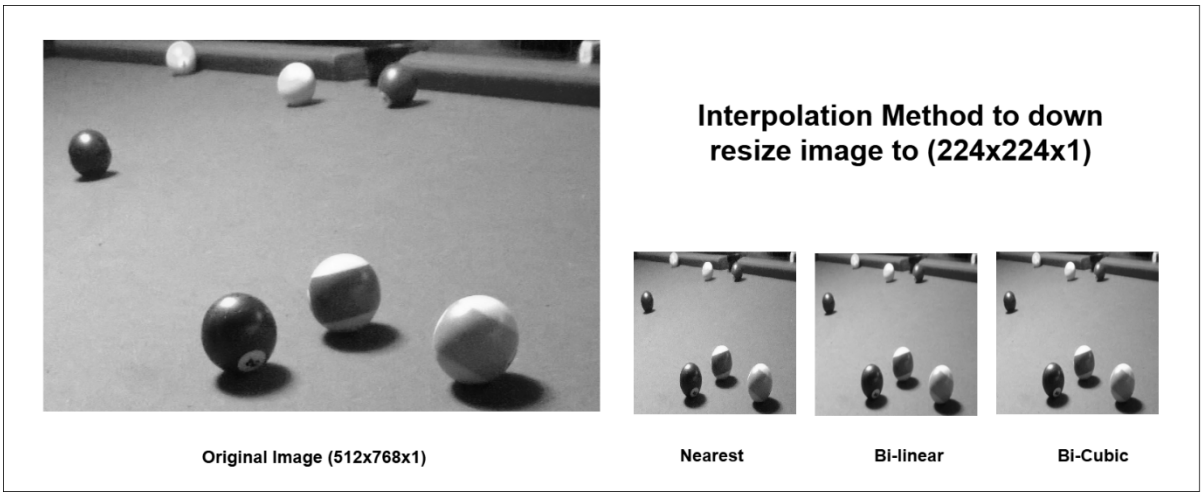


Figure 3.18: Comparing different interpolation methods

Nearest neighbour interpolation: This is the simplest interpolation, which estimates the new pixel location using the closest pixels in the original image. This method is fast and can produce jagged or blocky results.

For linear interpolation based closest neighbour

$$f(X) = f(X_0) + \left(\frac{(X - X_0)(f(X_1) - f(X_0))}{(X_1 - X_0)}\right)$$

(3.8,1)

The same principle is extended to 2D interpolation in Images, where pixels are estimated based on their neighbours.

Bilinear Interpolation: Considers the four nearest pixels in 2x2 neighbourhood. A weighted average is calculated based on the distance. This method produces smoother results than nearest neighbour but can still blur the edges slightly.

It is based on linear interpolation along both x and y axes:

$$f(x,y) = (1-a)(1-b) f_{00} + a(1-b) f_{10} + (1-a) b f_{10} + ab f_{11}$$

(3.8.2)

where a and b are fractional distance from the top-left pixel and f_{ij} are the values of the four surrounding pixels.

Bicubic Interpolation: Considers a 4x4 neighbourhood of pixels (16 pixels) and produces much smoother and visually pleasing results than previous methods but requires more computation than the Bilinear Interpolation. It is based on a polynomial along both axes. This leads to a smoother and more natural output.

$$F(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} w_i(x) \cdot \omega_j(y) \cdot f(\beth, j)$$

(3.8.3)

Where $w_i(x)$ and $w_j(y)$ are cubic weight functions.

Figure 3.18 shows all three interpolation method side by side.

3.9 Neural network sustainable training strategies

3.9.1 Early stopping

Early stopping is a regularization technique employed to mitigate overfitting. The method involves partitioning the training data to create a validation set (in this work, a 9:1 split), which is not used for gradient updates. The model's performance is monitored on this validation set, typically after each epoch. While the training loss generally continues to decrease, the validation performance (e.g., accuracy or loss) will often reach an optimal point and then begin to degrade. This inflection point signifies that the model has started to overfit the training data rather than learning generalizable patterns. Early stopping halts the training process at this point of optimal generalization.

In our implementation, the model checkpoint yielding the best validation accuracy is saved. A "patience" parameter is set, allowing training to continue for a specified number of epochs beyond the last improvement. If no new best validation accuracy is achieved within this patience window, the training is terminated, and the best-performing model is retained.

3.9.2 Fine-tuning pre-trained models

Fine-tuning is a transfer learning technique that leverages a model previously trained on a large, general dataset (e.g., large collection of images for vision-models or a large text corpus for language models). Instead of initializing a new model with random weights, fine-tuning uses the learned weights of the pre-trained model as a starting point. This provides a superior weight initialization, as the initial layers have already learned to recognize fundamental features (like edges and textures in images, or grammar in text). The training process then adjusts these weights for the specific downstream task, which is often on a smaller, more specialized dataset. This approach dramatically reduces the required training time and data, leading to more efficient model development and often superior performance. For example, a machine translation model trained on a vast text corpus can be effectively fine-tuned for more specific tasks like sentiment analysis. In this work, the pre-trained model used are BERT, ResNet50, DenseNet121, EfficientNetB0-B7, ViT, and SwinFormer.

3.9.3 Ensembling

Ensembling is a method that combines the predictions from multiple independently trained models to produce a single, final prediction. The underlying principle is that different models, even with identical architectures, will learn slightly different functions due to random initializations and stochastic elements in the training process. Their individual errors are often uncorrelated. By aggregating their prediction for instance, through majority voting for classification tasks or averaging for regression tasks, these random errors tend to cancel each other out. This process reduces the model's variance and leads to a more robust, accurate, and generalizable final prediction. While highly effective, the primary drawback of Ensembling is the significant increase in computational cost, as it requires training and maintaining multiple models.

Chapter 4

Sentiment Analysis using BLSTM-ResNet on Textual Images

4.1 Introduction

The primary focus of this chapter is to use deep 2D-CNN architectures to replace the shallow 2D-CNN in an existing BLSTM-2DCNN model [27]. This chapter investigates the proposed method for sentiment analysis (BLSTM-ResNet), striving to extract opinions from a text by first transforming the text into images using a BLSTM and leveraging powerful spatial feature extraction capabilities of a 2D ResNet. The text images are formed by transforming the word embeddings matrix into a greyscale image with the help of a BLSTM (Section 3.8.4).

This chapter investigates if deeper ResNet-based 2D CNNs, replacing the existing shallow 2D-CNN in the BLSTM-2DCNN model of [27], are more effective in identifying and extracting nuanced sentiment features from image-encoded text. The literature motivating the use of deep 2D-CNNs for Textual Image sentiment classification is reviewed in Section 2.6, particularly in the subsection addressing RQ1 and Chapter 4. The Literature identifies the limitations of 1D-CNN methods and the shallow, fixed-depth 2D-CNN approaches that motivate the present investigation.

Addressing these limitations, a BLSTM-ResNet model is proposed in this chapter. This model is a novel hybrid architecture combining BLSTM networks with deeper 2D-CNNs. The ResNet model is widely recognised in computer vision tasks where it increases network depth without facing performance degradation problems.

This chapter implements Pipeline 1, described in Section 3.4.1. Static GloVe embeddings are processed by a BLSTM to form a single-channel Textual Image, which is classified by a custom ResNet trained from scratch. The BLSTM–2DCNN is used as the shallow baseline within this pipeline. The shallow and deep BLSTM–ResNet architectures are Pipeline 1 variants: they retain the same representation pathway while varying residual-block depth and construction. Section 4.2.1 describes these variants in detail, including the Type-1 blocks used in the shallow models and the Type-2 blocks used in the deep models.

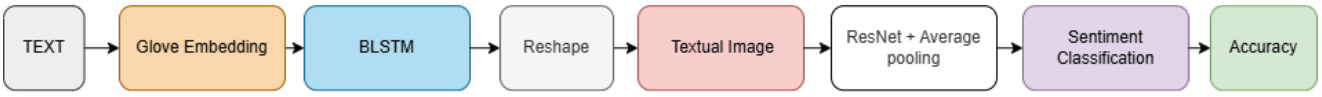


Figure 4.1: Sentiment Analysis pipeline showing the use of BLSTM to generate Textual Images and ResNet to extract sentiment from Textual Images.

Figure 4.1 represents the pipeline adopted in performing sentiment analysis using BLSTM-ResNet (extended version of Pipeline 1, described in Section 3.4.1). GloVe embeddings are used to generate the word embeddings from the clean text, and BLSTM is used to generate the images from the word embeddings for every input document (text). ResNet utilising 2D CNN is then used to extract sentiment from the Textual Images (refer Section 3.8.6.1).

Extensive experiments on the Sentiment140 dataset are conducted to systematically investigate factors influencing sentiment analysis performance using BLSTM-ResNet: the depth of the ResNet network, the size of the training dataset, and the specific configurations of the residual blocks. Our investigation is structured across three main experiments.

Experiment 1 delves into the impact of network depth, assessing how increasing the number of residual blocks in our BLSTM-ResNet models affects sentiment analysis performance. **Experiment 2** then focuses on the influence of dataset size, examining how the performance of our models varies when trained on different subsets of the

Sentiment140 dataset. **Experiment 3** explores the effectiveness of different residual block configurations, comparing shallow (Type-1) residual blocks with a deeper (Type-2) variant to see if further increases in convolutional layers within each block yield better results. By designing and testing several BLSTM-ResNet variants, categorised as shallow and deep BLSTM-ResNet models, across these three experimental setups, the effectiveness of the deep 2D-CNN model for sentiment analysis relative to the shallow BLSTM-2DCNN baseline [27] will be evaluated.

The following sections in this chapter will detail the different BLSTM-ResNet variants, Experimental setup, Experiments and results and offer insights into the effectiveness of the BLSTM-ResNet model for sentiment analysis. The insights and findings presented in this chapter have been published previously in [136] which included only the best model results and key trends. This chapter is a much-extended version of the published paper in that it provides the comprehensive results and model performance data that were omitted from the published paper due to length and formatting constraints.

4.2 Our Proposed Textual Image classification method

4.2.1 BLSTM-ResNet Variant

The proposed BLSTM-ResNet model consists of two major components. The first is the use of BLSTM for Textual Image generation from word Embeddings while capturing word temporal dependencies, and the second is the ResNet-inspired 2D-CNN residual blocks to explore feature space. These are followed by an Average Pooling layer, a Flatten layer and finally an Output layer.

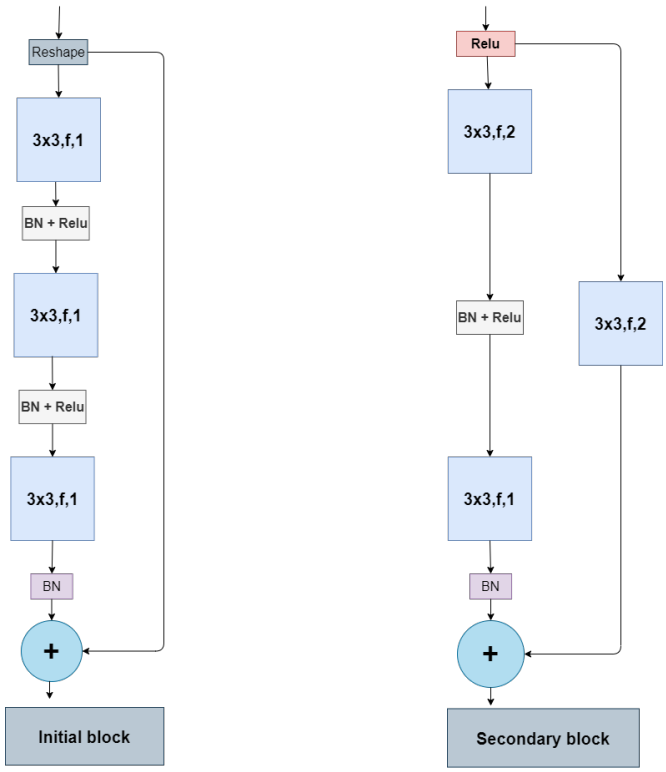


Figure 4.2: Residual Block Type-1, including BN- Batch normalisation layers, where ReLU is the activation function and a convolution block where the kernel size of 3x3, and where f is the number of kernels, and the stride value, s, is 1 or 2.

A neural network model is used with a direct connection between a preceding layer and a succeeding layer, which is called a residual/skip connection. Along with the layers in between, the two connected layers are called a residual block. Our network has two sets of different residual block constructions named Type-1 (Figure 4.2) and Type-2 (Figure 4.3) with each type having a unique Initial and Secondary block. Based on the type of residual block, either a shallow BLSTM-ResNet model (using Type-1) or a deep BLSTM-ResNet model (using Type-2) is created. In this thesis, the terms “shallow” and “deep” ResNet refer primarily to the internal construction of the residual blocks, rather than solely to the total number of residual blocks or convolutional layers in the network. Type-1 blocks define the shallow BLSTM-ResNet variants, whereas the more convolutionally elaborate Type-2 blocks define the deep variants; the number of blocks is varied separately within each family.

The two different models are used to investigate the impact of residual-block construction, that is, the depth of the network, on the performance of the network. A detailed implementation of the proposed BLSTM-ResNet network is explained in Section 3.8.6.1

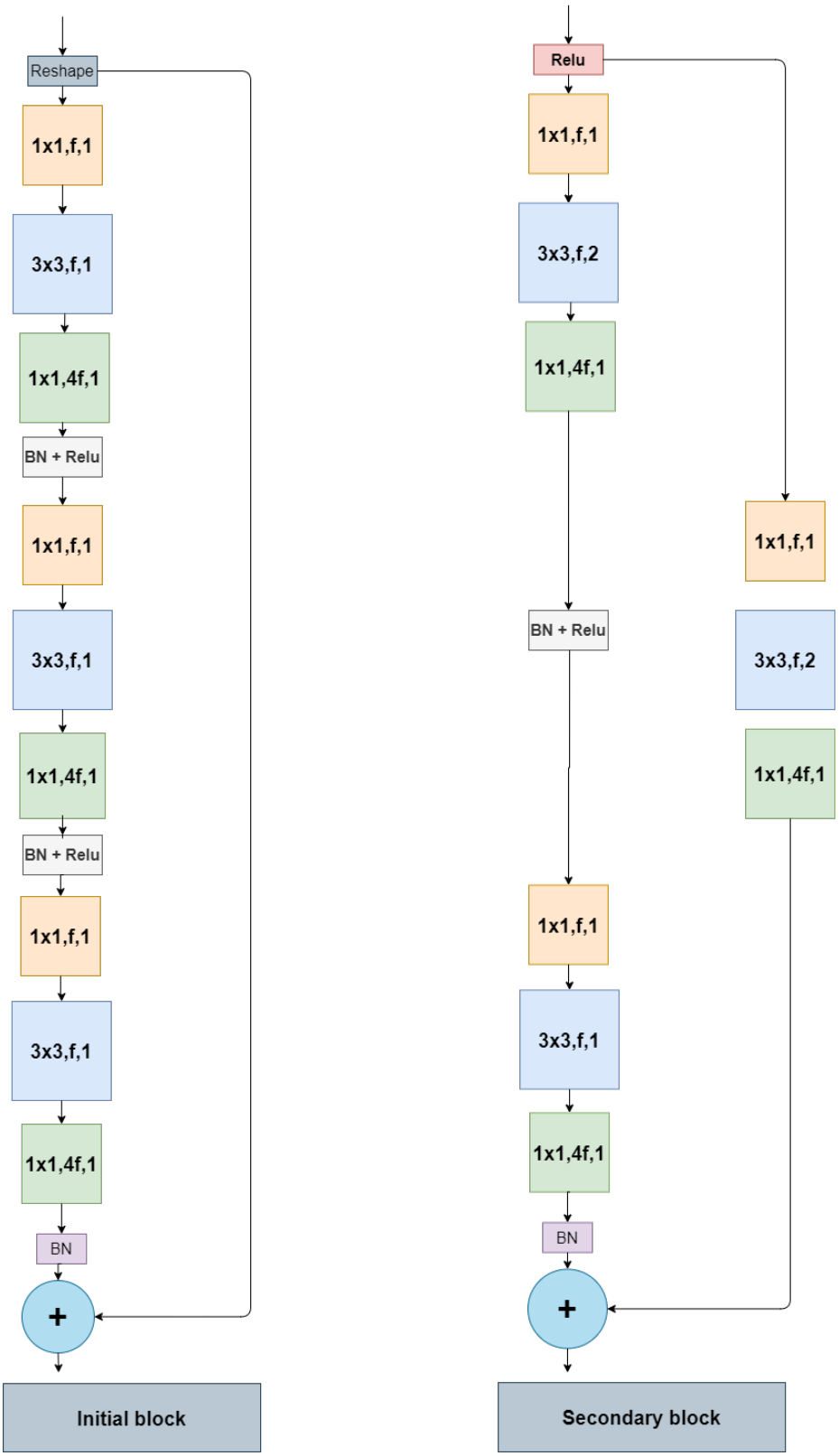


Figure 4.3: Residual Block Type 2, BN- Batch normalization layers, where ReLU is activation function. The Convolution blocks have kernel size of 3x3 or 1x1, where f is the number of kernels, and the stride values, is 1 or 2.

Based on the number and type of residual blocks used, ResNet models with various depths are formed. In Table 4.1, the number after BLSTM-ResNet represents the number of 2D-CNN layers in the network. This numbering style of the different depth of BLSTM-ResNet model is inspired by the original ResNet paper. BLSTM-ResNet can be categorised as either a Shallow BLSTM-ResNet or a Deep BLSTM-ResNet based on the type of residual blocks

used in the model.
Table 4.1: List of BLSTM-ResNet variants with the total number of residual blocks. The number following the BLSTM-ResNet model name indicates the total number of CNN layers. The Shallow/Deep type identifies the residual blocks used in each BLSTM-ResNet variant.

Model	**Residual Blocks**	**Total number of residual blocks**	**Type**
BLSTM-ResNet-3	Type-1: Initial only	one	Shallow
BLSTM-ResNet-6	Type-1: Initial + 1 Secondary	Two	Shallow
BLSTM-ResNet-9	Type-1: Initial + 2 Secondary	Three	Shallow
BLSTM-ResNet-12	Type-1: Initial + 3 Secondary	Four	Shallow
BLSTM-ResNet-15	Type-1: Initial + 4 Secondary	Five	Shallow
BLSTM-ResNet-18	Type-1: Initial + 5 Secondary	Six	Shallow
BLSTM-ResNet-21	Type-1: Initial + 6 Secondary	Seven	Shallow
BLSTM-ResNet-27	Type-2: Initial + 2 Secondary	Three	Deep
BLSTM-ResNet-36	Type-2: Initial + 3 Secondary	Four	Deep

4.3 Experiment Setup

The dataset used to test out our models is Sentiment140 [16]. The experiment in this chapter uses subsets of size 20,000, 40,000, 60,000, 80,000, 100,000 and 200,000 (refer Section 3.3 for details).

4.3.1 Data Preparation

To convert the words in the tweets in Sentiment140, GloVe word embeddings are used [51] which are pre-trained on a large English corpus. These pre-trained GloVe embeddings only have values for the actual English words and no vectors for anything else (for example, clutter in the tweets like URLs, hyperlinks, hashtags, and so on). So, all the non-English words and other clutter are omitted while tokenising the word documents. The Tokenizer method is used to tokenise the text and only the top 10,000 most frequent words are utilised to create the word vectors. The word embeddings used in the experiments have 300 dimensions and BLSTM hidden units are also kept to 300, as used in [27]. A tweet has a character count limit for each post, and the embedding uses word vectors. So, in the experiments below, an image formed after the BLSTM-layer has a resolution of $N \times 300\ (Hidden\ units) \times 1(Channel)$. N is the maximum number of words in a tweet within an entire dataset, which varies from 34 to 38 in the various subsets of data and to counteract this variation, a value for N of 35 is used across the experiments. The channel in our text image is one, which means it is a greyscale image.

4.3.2 Learning Rate Strategies for BLSTM-ResNet

The step decay technique is used in [23] to reduce the learning rate to 10% several times during the training. The same learning rate decay strategy is used in our experimentation. These are pre-scheduled learning rate decay values, and they are handled by the Learning Rate Scheduler method. This takes epochs and returns the corresponding pre-defined learning rate value for that epoch. The initial learning rate is 1e-3 and is scheduled to be reduced at 80,120,160,180 epoch numbers by a factor of 1e-1 while keeping the total training epochs to 200.

Along with the learning rate scheduler method, another method is used to reduce the learning rate when training hits a plateau. Every five epochs, when training is on a plateau, the learning rate reducer is called. The number 5 represents the patience value for this method, and the learning rate is decayed by a factor of $\sqrt{0.1}$ with 5e-7 as the lower learning rate bound.

4.3.3 Training Strategy

In this chapter, the original BLSTM-2DCNN of [27], is treated as a network with zero residual blocks and used as a baseline for comparison with our deeper models while investigating the impact of network depth and dataset size on performance. For all BLSTM-2DCNN training, the early stopping method is used to prevent overfitting. The best model weights are restored after training. The optimiser used in these experiments is Adam (adaptive momentum). Whereas the Adagrad optimiser (Adaptive gradient) is used in the paper BLSTM-2DCNN [27],. Both

these optimisers are used for adaptive learning rate depending on the history of gradient, but Adam solves the denominator decay problem of the Adagrad, thus helping the model to converge quickly.

In the case of BLSTM-ResNet, a Stochastic Gradient Descent (SGD) optimiser is used based on the original ResNet literature. During each training epoch, the validation loss is observed and for each lowest value, the corresponding model weights are stored. The best model weights are loaded back at the end of training as the final model weights.

While experimenting, various subsets from the Sentiment140 dataset are generated using the first n/2 rows of data from each class, where n is the size of the subset. This is done to keep consistency across the experiments instead of randomising the selection. For example, for the 20,000-size subset, the first 10,000 instances for both Negative and Positive classes are used. This subsampling method maintains consistency, but it can introduce bias in our performance depending on the data collection method used by the dataset authors .

Since all the subsets of Sentiment140 are balanced, accuracy and weighted F1 score are used as the performance metrics for comparison. During training, a 9:1 train-test stratified split is used for each subset, and 5-fold Cross-validation (CV) is performed on the training set to train 5 different models. Each CV model is then evaluated on the test set (Refer Appendix Section 9.1 for details). Employing a 5-fold cross-validation framework ensures reliability and generalizability of the findings. Accuracy and weighted F1 scores are used as test metrics to compare various model performances.

To compare the performance of various BLSTM-ResNet models and the baseline on Textual Images, the Friedman test was employed in Experiment 2 and Experiment 3. The Friedman test is a non-parametric statistical method designed to detect differences across multiple related groups. In the context of this study, these groups consist of the baseline model (BLSTM-2DCNN) and several proposed variants of the BLSTM-ResNet architecture, each evaluated across identical cross-validation folds. The use of the Friedman test aligns with established best practices for empirical comparison of machine learning models, as recommended by [137].

Deep learning experiments often violate parametric test assumptions due to data complexity and model variability. The Friedman test addresses this by ranking model performances on the test set for model trained in each cross-validation fold, thus accounting for paired, dependent observations. This approach enables a statistically robust comparison of multiple related models while controlling for fold-to-fold variability and reducing the risk of false positives.

4.4 Results

First, the shallow BLSTM-ResNet which utilises Type-1 residual blocks is evaluated. This model is used to investigate both the effect of depth and the effect of data size on prediction accuracy. Finally, the deep BLSTM-ResNet, formed using Type-2 residuals, is investigated to see if its extra depth can be of use. In total three different sets of experiments are conducted in this chapter, which are elaborated below.

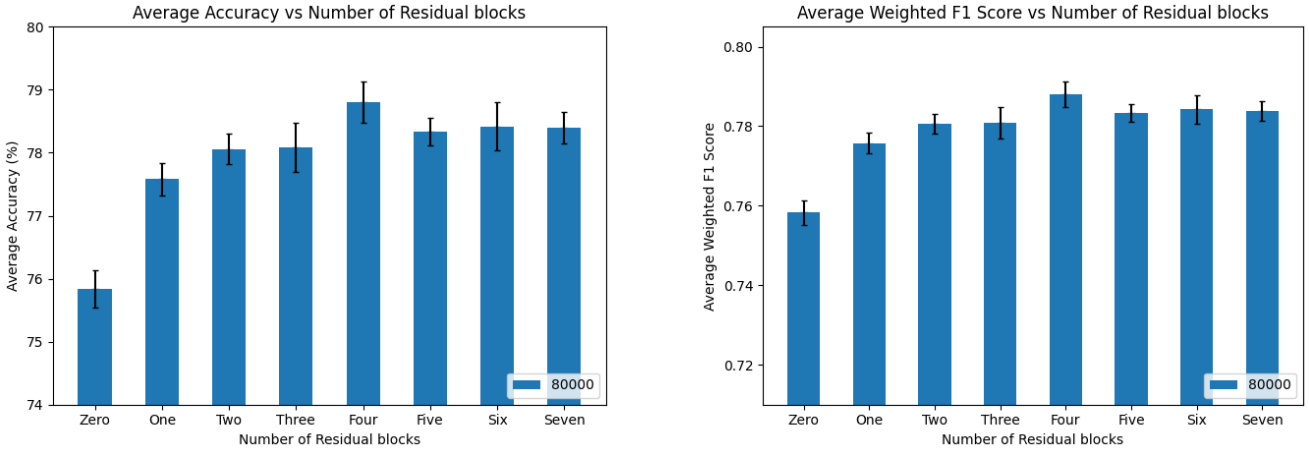
4.4.1 Experiment 1: Impact of Network Depth on model performance

The experiment aims to investigate the effect of varying depths within BLSTM-ResNet architecture variants all with Type-1 residuals(see Table 4.1 for the list), on model performance measured by two performance metrics, specifically accuracy and weighted F1 scores. Two distinct Sentiment140 subset sizes are used: initially 80,000 and subsequently 200,000 data points, and we follow all the experiment settings established in Section 4.3. Note that with the increase in depth of the network, the output resolution of the Textual Image reduces, which limits the maximum depth of a BLSTM-ResNet model. In this case, with Type-1 residuals, plus adopted data pre-processing, the maximum CNN depth achieved is 21 (refer Appendix Section 10.1).
Table 4.2: Average model performance, with its standard deviation, for different numbers of Residual Blocks using the 80,000 dataset.

Model	**Number of Residual Blocks**	**Average Accuracy (%)**	**Average Weighted F1**
BLSTM-2DCNN	Zero	75.84 ± 0.30	0.7583 ± 0.0031
BLSTM-ResNet-3	One	77.58 ± 0.26	0.7757 ± 0.0026
BLSTM-ResNet-6	Two	78.06 ± 0.24	0.7806 ± 0.0024
BLSTM-ResNet-9	Three	78.09 ± 0.39	0.7808 ± 0.0039
BLSTM-ResNet-12	Four	**78.80 ± 0.33**	**0.7880 ± 0.0033**

Continued on next page

Model	Number of Residual Blocks	Average Accuracy (%)	Average Weighted F1
BLSTM-ResNet-15	Five	78.33 ± 0.22	0.7833 ± 0.0022
BLSTM-ResNet-18	Six	78.42 ± 0.38	0.7842 ± 0.0037
BLSTM-ResNet-21	Seven	78.39 ± 0.25	0.7839 ± 0.0025



(a) (b)

Figure 4.4: Average model performance, with its standard deviation, for different numbers of Residual Blocks using the 80,000 dataset.

Initially, the experiment is conducted on 80,000 data points. The experimental results for accuracy and weighted F1-score are presented in columns 3 and 4 of Table 4.2, respectively. Additionally, the trends for these metrics are illustrated in Figure 4.4 (a) and (b), respectively. There was a discernible pattern of incremental improvement in model performance as the network depth increased from zero to four residual blocks. The baseline model, BLSTM-2DCNN, which is zero residual blocks, recorded an accuracy of approximately 75.84% ± 0.30. Progressively, as residual blocks were introduced, a steady rise in accuracy was observed, culminating in a peak performance of 78.80% ± 0.33 at the configuration with four residual blocks. Parallel trends were clearly reflected in the weighted F1 scores, which improved from an initial 0.7583 ± 0.0031 to a matching peak value of 0.7880 ± 0.0033. This consistent enhancement across both metrics indicates that the addition of residual blocks effectively captured more nuanced patterns within the Textual Image, translating into improved sentiment extraction. Notably, the standard deviations across these configurations remained relatively low, ranging from ±0.22% to ±0.39% for accuracy, and from ±0.0022 to ±0.0039 for the weighted F1-score.

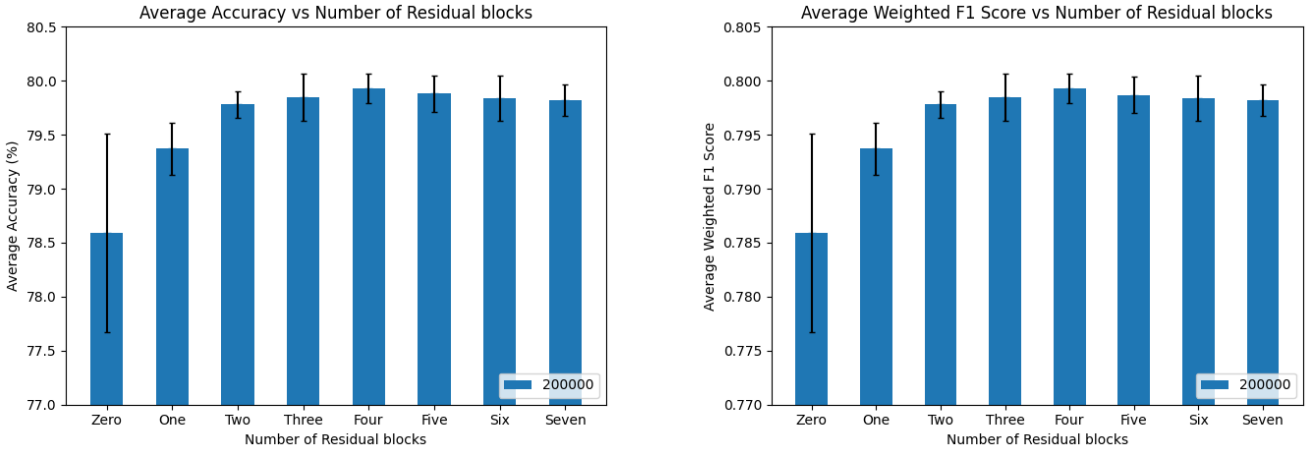
However, beyond the four residual-block configurations, a saturation effect became apparent. Further increases in network depth did not yield meaningful gains in performance metrics; instead, accuracy and weighted F1 scores either marginal declined or plateaued. For instance, after four residual blocks, accuracy slightly decreased to 78.33% ± 0.22 at five residual blocks and fluctuated minimally in subsequent deeper configurations. This critical juncture at four residual blocks signified an optimal trade-off between model depth and sentiment analysis performance for the given dataset size, marking a clear watershed point for additional explorations. The reduced resolution of the Textual Images with increasing residual blocks could become too small, for the deeper convolution layers, to extract anything meaningful to contribute further. Moreover, the padding added with every convolution operation (refer Section 3.8.6.1) can contribute to information dilution in deeper layers after a certain point and leads to the saturation.

Table 4.3: Average Model performance with its Standard deviation for 200,000 dataset size

Model	Number of Residual Blocks	Average Accuracy (%)	Average Weighted F1
BLSTM-2DCNN	Zero	78.59 ± 0.92	0.7859 ± 0.0092
BLSTM-ResNet-3	One	79.37 ± 0.24	0.7937 ± 0.0024
BLSTM-ResNet-6	Two	79.78 ± 0.12	0.7978 ± 0.0012
BLSTM-ResNet-9	Three	79.85 ± 0.22	0.7985 ± 0.0022
BLSTM-ResNet-12	Four	**79.93 ± 0.14**	**0.7993 ± 0.0014**
BLSTM-ResNet-15	Five	79.88 ± 0.17	0.7987 ± 0.0017

Continued on next page

Model	Number of Residual Blocks	Average Accuracy (%)	Average Weighted F1
BLSTM-ResNet-18	Six	79.84 ± 0.21	0.7984 ± 0.0021
BLSTM-ResNet-21	Seven	79.82 ± 0.15	0.7982 ± 0.0015



(a) (b)

Figure 4.5: Average model performance with its standard deviation vs Number of Residual Blocks for the 200,000 dataset.

To further validate and explore the observed saturation effect and to identify the optimal configuration, the experiment was extended using a larger subset of 200,000 data points. The resulting accuracy and weighted F1 metrics are presented in Table 4.3, with corresponding trends illustrated in Figure 4.5 (a) and (b), respectively. This additional investigation aimed to confirm whether the patterns observed at 80,000 data points would remain consistent with increased training data.

Interestingly, the larger dataset replicated and reinforced the initial findings. Starting from baseline metrics of 78.59 ± 0.92% accuracy and a weighted F1-score of 0.7859 ± 0.0092 (with zero residual blocks), performance improved steadily, reaching an optimal configuration at four residual blocks. At this point, the model achieved peak performance, with accuracy and weighted F1-score values of 79.93 ± 0.14% and 0.7993 ± 0.0014, respectively. As observed with the 80,000 dataset, additional increases in network depth beyond four residual blocks provided negligible performance gains. Notably, standard deviations observed with this larger dataset were lower, ranging from ±0.12% to ±0.24% for accuracy and from ±0.0012 to ±0.0024 for weighted F1-score, highlighting enhanced model performance stability across the training data folds resulting from increased data availability. While overall performance improved with the larger training set, the watershed effect seen clearly at 80,000 data points was less pronounced at this larger scale.

The experimental findings identified four residual blocks as the optimal network depth for BLSTM-ResNet models within the examined scenarios, providing the highest accuracy and weighted F1 scores without unnecessary model complexity by adding more residual blocks. Notably, increasing the dataset size from 80,000 to 200,000 data points improved performance across all model configurations. However, the extent of performance improvement varied between configurations, indicating that dataset size impacts model performance disproportionately. Additionally, the watershed effect observed clearly at 80,000 data points diminished with the larger dataset, suggesting that the benefit of additional depth becomes less pronounced as more training data is introduced. Furthermore, the standard deviations across model configurations decreased consistently with the larger dataset, ranging from ±0.22% to ±0.39% for accuracy and from ±0.0022 to ±0.0039 for the weighted F1-score, emphasizing enhanced stability and reliability due to increased data availability. These insights underscore the importance of carefully balancing dataset size and model depth. Further investigation into a range of dataset sizes beyond the current scope could provide additional clarity on optimising model architectures for efficiency and performance in similar balanced-data classification tasks.

4.4.2 Experiment 2: Impact of Dataset size on model Performance

The objective of Experiment 2 is to comprehensively evaluate the effect of dataset size on performance of various BLSTM-ResNet architectures compared to the baseline BLSTM-2DCNN model. Specifically, this experiment employs dataset subsets consisting of 20,000, 40,000, 60,000, 80,000, 100,000, and 200,000 data points. Based on the results of Experiment 1, the models used are from BLSTM-ResNet-3 (with 1 residual block) to BLSTM-ResNet-12

(with 4 residual blocks), since going deeper does not seem to add any significant improvement. For robust evaluation, a similar 5-fold cross-validation technique is adopted as again, providing insights into the average performance and variability through standard deviations for evaluation metrics. Specifically, accuracy results are presented in Table 4.4, while the corresponding weighted F1 scores are provided in Table 4.5 (refer Appendix Section 10.1.3 for individual run results).

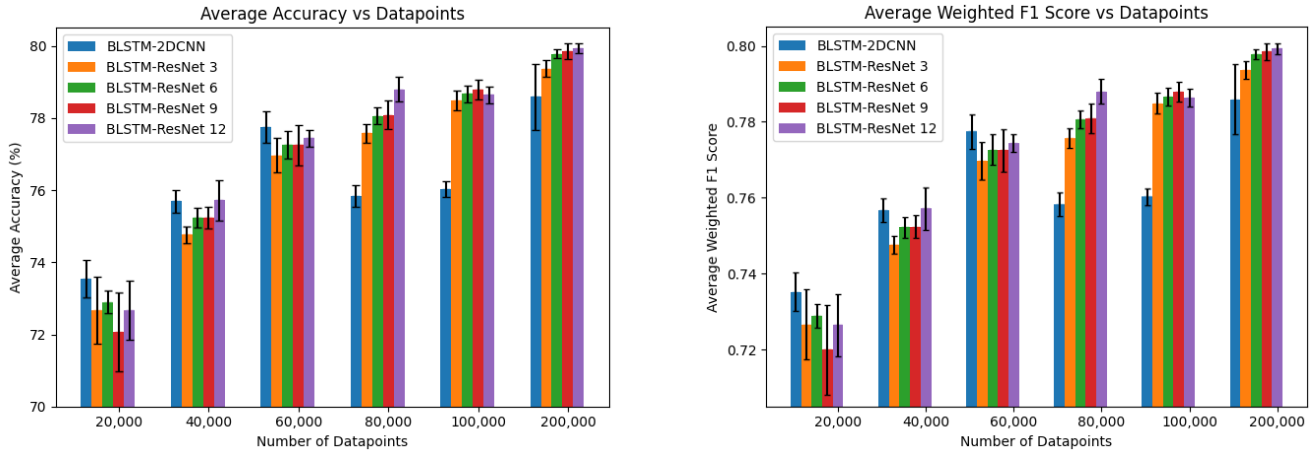
Table 4.4: Average Accuracy with standard deviation for datasets with different sizes for models of various depths.

Average Accuracy with standard deviation (%)						
Dataset size / Model	**20,000**	**40,000**	**60,000**	**80,000**	**100,000**	**200,000**
BLSTM-2DCNN	**73.54 ± 0.51**	75.69 ± 0.31	**77.75 ± 0.44**	75.84 ± 0.30	76.03 ± 0.22	78.59 ± 0.92
BLSTM-ResNet-3	72.67 ± 0.92	74.77 ± 0.23	76.97 ± 0.49	77.58 ± 0.26	78.49 ± 0.27	79.37 ± 0.24
BLSTM-ResNet-6	72.90 ± 0.31	75.23 ± 0.27	77.26 ± 0.39	78.06 ± 0.24	78.67 ± 0.24	79.78 ± 0.12
BLSTM-ResNet-9	72.07 ± 1.09	75.24 ± 0.31	77.25 ± 0.56	78.09 ± 0.39	**78.79 ± 0.27**	79.85 ± 0.22
BLSTM-ResNet-12	72.67 ± 0.82	**75.72 ± 0.56**	77.44 ± 0.23	**78.80 ± 0.33**	78.64 ± 0.24	**79.93 ± 0.14**

Table 4.5: Average weighted F1 score with standard deviations for datasets with different sizes for models of various depths.

Average Weighted F1 score with standard deviation						
Dataset size / Model	**20,000**	**40,000**	**60,000**	**80,000**	**100,000**	**200,000**
BLSTM-2DCNN	**0.7352 ± 0.0050**	0.7567 ± 0.0031	**0.7774 ± 0.0045**	0.7583 ± 0.0031	0.7603 ± 0.0022	0.7859 ± 0.0092
BLSTM-ResNet-3	0.7266 ± 0.0092	0.7476 ± 0.0023	0.7697 ± 0.0049	0.7757 ± 0.0026	0.7849 ± 0.0027	0.7937 ± 0.0024
BLSTM-ResNet-6	0.7289 ± 0.0031	0.7522 ± 0.0027	0.7726 ± 0.0040	0.7806 ± 0.0024	0.7867 ± 0.0023	0.7978 ± 0.0012
BLSTM-ResNet-9	0.7199 ± 0.0118	0.7524 ± 0.0031	0.7725 ± 0.0056	0.7808 ± 0.0039	**0.7879 ± 0.0027**	0.7985 ± 0.0022
BLSTM-ResNet-12	0.7265 ± 0.0082	**0.7571 ± 0.0056**	0.7744 ± 0.0023	**0.7880 ± 0.0033**	0.7864 ± 0.0024	**0.7993 ± 0.0014**

The average performance trends across the different dataset subsets are depicted as bar plots in Figure 4.6, with error bars representing the variability across the cross-validation folds. Specifically, Figure 4.6 (a) illustrates the accuracy metric, while Figure 4.6 (b) presents the corresponding weighted F1 scores.



(a) (b)

Figure 4.6: Average model performance with standard deviation vs Number of data points on various models.

The analysis begins with the smallest dataset size of 20,000 data points. Here, the BLSTM-2DCNN baseline model clearly outperforms all BLSTM-ResNet configurations in terms of both accuracy (73.54% ± 0.51) and weighted F1 score (0.7352 ± 0.0050). The BLSTM-ResNet variants demonstrate a general trend of decreased performance with increasing depth, with the lowest accuracy observed in BLSTM-ResNet-9 at 72.07% ± 1.09. Notably, standard deviation values on average indicate substantial variability, especially for the BLSTM-ResNet models showcasing

the impact of limited training data.

With dataset sizes increased to 40,000, performance across all models sees marked improvements. The baseline BLSTM-2DCNN continues to exhibit among the height's accuracy (75.69% ± 0.31) and weighted F1 score (0.7567% ± 0.0031). The BLSTM-ResNet models begin to close the performance gap, with BLSTM-ResNet-12 achieving the best accuracy of 75.72% ± 0.56 and weighted F1 of 0.7571 ± 0.0056. Standard deviations across models show a reduction like the findings of the previous experiment.

At 60,000 data points, the baseline model again achieves the highest accuracy (77.75% ± 0.44) and weighted F1 score (0.7774 ± 0.0045), but the BLSTM-ResNet models begin to exhibit comparable performance and follow a similar trend as with the previous dataset size. BLSTM-ResNet-3 achieves an accuracy of 76.97% ± 0.49, while deeper models (BLSTM-ResNet-6, 9, and 12) achieve similar or slightly better results as the model depth increases. It is observed that the standard deviations continue to shrink, indicating increased stability in the performance measures with increase training data.

A notable shift occurs at the dataset size of 80,000 and above, marking a critical threshold. Here, BLSTM-ResNet variants begin to consistently outperform the baseline BLSTM-2DCNN model. BLSTM-ResNet-12 yields the highest accuracy (78.80% ± 0.33) and weighted F1 score (00.7880% ± 0.0033), clearly surpassing the baseline (accuracy 75.84% ± 0.30, weighted F1 score 00.7583% ± 0.0031). This suggests that deeper architectures become more advantageous as dataset size increases, as deeper models can take advantage of larger amounts of data to extract appropriate patterns and so achieve optimal performance.

Continuing this trend, at a dataset size of 100,000 data points, all BLSTM-ResNet models consistently surpassed the baseline accuracy of 76.03%, achieving accuracies ranging from approximately 78.49% to 78.79%. The best-performing model was BLSTM-ResNet-9, with an accuracy of 78.79% ± 0.27% and a weighted F1-score of 0.7879 ± 0.0027. Notably, the watershed trend was observed at this scale, with BLSTM-ResNet-12 demonstrating a slight performance drop. Standard deviations remained relatively small, indicating stable and reliable model training at this dataset size.

At the largest dataset size, 200,000 data points, all models reach their highest performance. The BLSTM-ResNet-12 configuration attains the best accuracy (79.93% ± 0.14) and weighted F1 score (0.7993% ± 0.0014), marginally outperforming other BLSTM-ResNet variants. BLSTM-2DCNN, ranking the last, shows competitive results (accuracy 78.59% ± 0.92, weighted F1 78.59% ± 0.92) but with high variability compared to BLSTM-ResNet variants. Standard deviations across all BLSTM-ResNet-models at this dataset size are minimal, highlighting robust and reliable outcomes from extensive training data.

Throughout the analysis, accuracy (as ratio) and weighted F1 scores remain closely aligned, underscoring the balanced nature of the dataset. The decreasing standard deviations with increasing dataset size emphasize improved model stability and consistent generalization.

Table 4.6: Friedman statistics on Experiment 2 results

Dataset Size	**Friedman Stat**	**p-value**	**Best Model**	**Mean Ranks: BLSTM-2DCNN as B2DCNN BLSTM-ResNet-(3,6,9 & 12) as BR(3,6,9 & 12)**
20,000	4.64	0.3263	BLSTM-2DCNN	B2DCNN: 1.8, BR3: 3.4, BR6: 2.8, BR9: 3.8, BR12: 3.2
40,000	10.40	0.0342	BLSTM-ResNet-12	B2DCNN: 2.0, BR3: 4.6, BR6: 3.4, BR9: 3.2, BR12: 1.8
60,000	5.12	0.2752	BLSTM-2DCNN	B2DCNN: 2.0, BR3: 4.2, BR6: 2.8, BR9: 3.2, BR12: 2.8
80,000	16.69	0.0022	BLSTM-ResNet-12	B2DCNN: 5.0, BR3: 3.6, BR6: 2.8, BR9: 2.4, BR12: 1.2
100,000	11.19	0.0245	BLSTM-ResNet-9	B2DCNN: 5.0, BR3: 2.8, BR6: 2.8, BR9: 2.0, BR12: 2.4
200,000	16.64	0.0023	BLSTM-ResNet-12	B2DCNN: 5.0, BR3: 4.0, BR6: 2.4, BR9: 2.0, BR12: 1.6

Table 4.6 examines the performance, measured in terms of the Friedman statistic, of the BLSTM-2DCNN models along with range of BLSTM-ResNet variants (BR3, BR6, BR9, BR12) across increasing dataset sizes. For smaller datasets (e.g., 20,000 samples), the Friedman statistic (4.64, p = 0.3263) indicates no statistically significant difference (relative to significance level $\alpha = 0.05$) in model performance, with the baseline BLSTM-2DCNN achieving the best mean rank (1.8). As the dataset size increases to 40,000, the test yields a Friedman statistic of 10.40 (p = 0.0342), revealing a statistically significant difference among models, with BLSTM-ResNet-12 outperforming the baseline. A non-significant outcome is observed for 60,000 samples (p = 0.2752), while statistically significant results are detected for larger dataset sizes—particularly at 80,000 (Friedman stat 16.69, p = 0.0022), 100,000 (stat 11.19, p = 0.0245), and 200,000 (stat 16.64, p = 0.0023). In these cases, the deeper BLSTM-ResNet architectures (BR12, BR9) consistently outperform the baseline, as reflected in their lower mean ranks. These findings suggest that the advantages of BLSTM-ResNet architectures become increasingly pronounced with larger datasets, highlighting the interplay between model complexity and data availability.

This experiment provides clear evidence of the influence of dataset size on the performance of the BLSTM-ResNet models compared to the BLSTM-2DCNN baseline. The results demonstrate that increasing the training dataset size consistently improves model accuracy and weighted F1-score while simultaneously reducing performance variability, as reflected in the decreasing standard deviations. Small datasets (20,000–60,000) favour the simpler BLSTM-2DCNN model, as deeper BLSTM-ResNet configurations underperform and exhibit higher variability due to limited data. Larger datasets (80,000–200,000) lead to highly stable model training, reflected in minimal standard deviations for both accuracy and weighted F1 metrics. BLSTM-ResNet-12 achieves the overall best performance (79.93% accuracy and 0.7993 weighted F1 at 200,000 data points), but most BLSTM-ResNet variants converge to similar performance, reinforcing the diminishing returns of increasing depth beyond the optimal configuration.

In summary an increase in the number of residual blocks initially improves model performance before reaching a saturation point. A plausible reason for this saturation is that with each additional residual block, the Textual Image resolution is halved. At deeper layers, this reduction might result in feature maps that are too small to capture meaningful information, limiting further performance gains. To investigate this further, deeper BLSTM-ResNet models can be constructed by increasing the number of convolutional layers within each residual block. This approach leads to the development of Type-2 residual blocks, illustrated in Figure 4.3, and motivates the introduction of Experiment 3. These blocks achieve greater depth by incorporating 1×1 convolution operations both before and after the 3×3 convolution.

4.4.3 Experiment 3: Impact of Residual block construction on model Performance

This experiment aims to examine the impact of BLSTM-ResNet residual block depth on performance metrics, specifically accuracy and weighted F1 scores, across different dataset sizes in sentiment analysis tasks. In this experiment, two Shallow BLSTM-ResNet-models, BLSTM-ResNet-9 and BLSTM-ResNet-12, are used as the baseline models to compare with two Deep BLSTM-ResNet models, BLSTM-ResNet-27 and BLSTM-ResNet-36, to see if deep BLSTM-ResNet models would perform better.

The BLSTM-ResNet-9 model comprises three Type-1 residual blocks, while BLSTM-ResNet-12 includes four. In contrast, the BLSTM-ResNet-27 and BLSTM-ResNet-36 models utilize Type-2 residual blocks, with three and four blocks respectively. For a meaningful comparison, BLSTM-ResNet-9 is compared against BLSTM-ResNet-27, as both contain three residual blocks, and BLSTM-ResNet-12 is compared with BLSTM-ResNet-36, each having four blocks. The experimental setup follows previous configurations, employing datasets of varying sizes to assess how the different types of residual blocks perform across dataset scales. Table 4.7 presents average accuracies with standard deviations for each BLSTM-ResNet variant whereas Table 4.8 presents the corresponding weighted F1 score values (refer Appendix Section 10.1.3 for individual run results).

Table 4.7: Comparison of Residual Type-1 & Type-2 using Average Accuracy with standard deviation for 3 and 4 residual block models on different dataset sizes.

Model	20,000	40,000	60,000	80,000	100,000	200,000
BLSTM-ResNet-9	72.07 ± 1.09	75.24 ± 0.31	77.25 ± 0.56	78.09 ± 0.39	**78.79 ± 0.27**	79.85 ± 0.22
BLSTM-ResNet-27	72.52 ± 0.32	**76.13 ± 0.58**	77.70 ± 0.47	78.01 ± 0.62	78.73 ± 0.33	79.74 ± 0.24
BLSTM-ResNet-12	72.67 ± 0.82	75.72 ± 0.56	77.44 ± 0.23	**78.80 ± 0.33**	78.64 ± 0.24	**79.93 ± 0.14**
BLSTM-ResNet-36	**73.40 ± 0.58**	75.91 ± 0.91	**77.82 ± 0.21**	78.53 ± 0.22	78.60 ± 0.34	79.50 ± 0.27

Table 4.8: Comparison of Residual Type-1 & Type-2 using Average Weighted F1 scores with standard deviation for 3 and 4 residual block models on different dataset sizes.

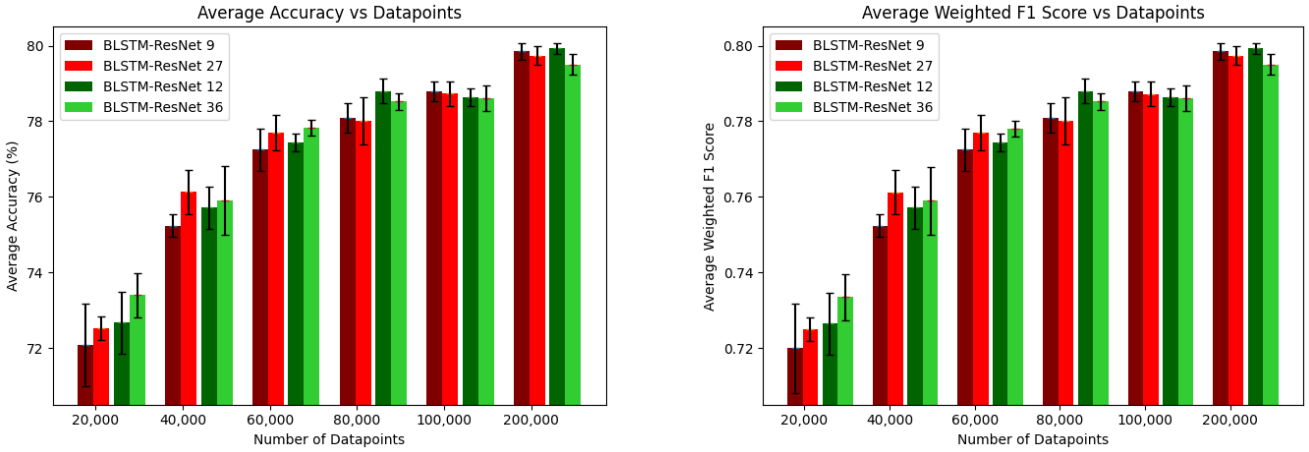
Model	20,000	40,000	60,000	80,000	100,000	200,000
BLSTM-ResNet-9	0.7199 ± 0.0118	0.7524 ± 0.0031	0.7725 ± 0.0056	0.7808 ± 0.0039	**0.7879 ± 0.0027**	0.7985 ± 0.0022
BLSTM-ResNet-27	0.7250 ± 0.0032	**0.7612 ± 0.0058**	0.7770 ± 0.0047	0.7801 ± 0.0062	0.7872 ± 0.0033	0.7974 ± 0.0024
BLSTM-ResNet-12	0.7265 ± 0.0082	0.7571 ± 0.0056	0.7744 ± 0.0023	**0.7880 ± 0.0033**	0.7864 ± 0.0024	**0.7993 ± 0.0014**
BLSTM-ResNet-36	**0.7335 ± 0.0061**	0.7590 ± 0.0090	**0.7781 ± 0.0021**	0.7853 ± 0.0022	0.7860 ± 0.0034	0.7950 ± 0.0027

The values of Table 4.7 and Table 4.8 are plotted in Figure 4.7 (a) and Figure 4.7 (b) respectively, to better visualise the trends. The most prominent feature observed from this figure is that Deep BLSTM-ResNet perform better than Shallow BLSTM-ResNet for datasets 20,000 to 60,000 but for dataset 80,000 to 200,000 data points, the trend is reversed. The BLSTM-ResNet-36 model, notably the deepest evaluated variant, showed superior performance

for smaller datasets (20,000 and 60,000 data points), achieving accuracies of 73.40% ± 0.58 and 77.82% ± 0.21, respectively and weighted F1 score of 0.7340 ± 0.58 and 0.7782 ± 0.21, respectively (refer Table 4.7). In contrast, BLSTM-ResNet-12 demonstrated slightly better performance on intermediate dataset sizes, particularly at 80,000 data points with an accuracy of 78.80% ± 0.33, indicating that Type-1 blocks are catching up at this data scale.

Interestingly, BLSTM-ResNet-9, the shallowest model, consistently lagged the deeper models at most dataset sizes, suggesting that increased depth, either by adding more residual blocks or adding more CNN layers per block, will result in a performance boot. The BLSTM-ResNet-27 model, despite its depth, did not consistently surpass the shallower ResNet-12, indicating the benefit of additional residual blocks irrespective of the residual type can yields positive returns.

Like in the previous experiment, BLSTM-ResNet-9 performed the best at 100,000 dataset size surpassing every other model for both the metrics. Also, the BLSTM-ResNet-27 performs the second best at this dataset. BLSTM-ResNet-12 and BLSTM-ResNet-36 though behind have very competitive performance. This reinforces the previous finding that the increases in training data will give diminished return on increasing the model complexity.



(a) (b)

Figure 4.7: Average model performance with standard deviation vs Number of data points on Residual Type-1 & Type-2.

The corresponding Average weighted F1 scores for this experiment are presented in Table 4.8, providing complementary insights into each model performance. Notably, weighted F1 scores closely mirror accuracy outcomes across all evaluated models and dataset sizes. For example, at 200,000 data points, BLSTM-ResNet-12 achieved a weighted F1 score of 0.7993% ± 0.14, closely matching its accuracy in ratio, indicative of balanced class predictions. This closer similarity in trends between accuracy and weighted F1 is due to the balanced Sentiment140 subsets classes. The minute difference in the metrics is due to a minute mismatch in the class distribution in the test sets introduced while creating the train-test split from the balance dataset. The mismatch is apparent in lower dataset sizes, but it diminishes as the dataset size increases. For instance, all the model performances have minute differences in both the metrics values at lowest dataset size, but the difference is negligible at intermediate sizes and there is no apparent difference in the highest data set size.

Comparing standard deviations, the Deep BLSTM-ResNet model generally demonstrated the lower variance at smaller datasets (20,000 to 60,000 data points) than the Shallow BLSTM-ResNet, reinforcing the stability of deeper architectures in this context. Conversely, the trend reverses generally for 80,000 dataset size and beyond.

Table 4.9: Friedman statistics on Experiment 3 results.

Dataset Size	**Friedman Stat**	**p-value**	**Best Model**	**Mean Ranks BLSTM-ResNet-(9,12,27 & 36) as BR(9,12,27 & 36)**
20,000	8.48	0.0323	BLSTM-ResNet-36	BR9: 3.7, BR12: 2.2, BR27: 2.6, BR36: 1.5
40,000	8.04	0.0452	BLSTM-ResNet-27	BR9: 3.8, BR12: 2.6, BR27: 1.8, BR36: 1.8
60,000	3.96	0.2658	BLSTM-ResNet-36	BR9: 3.0, BR12: 3.0, BR27: 2.4, BR36: 1.6
80,000	6.06	0.1087	BLSTM-ResNet-12	BR9: 3.2, BR12: 1.6, BR27: 3.0, BR36: 2.2
100,000	0.36	0.9484	BLSTM-ResNet-27	BR9: 2.6, BR12: 2.6, BR27: 2.2, BR36: 2.6
200,000	5.40	0.1447	BLSTM-ResNet-12	BR9: 2.2, BR12: 1.8, BR27: 2.4, BR36: 3.6

Table 4.9 focuses on a comparative evaluation among the Shallow BLSTM-ResNet variants themselves, distinguishing

between shallower and deeper configurations (BR9, BR12 vs. BR27, BR36). The Friedman test identifies statistically significant differences at dataset sizes of 20,000 (stat 8.48, p = 0.0323) and 40,000 (stat 8.04, p = 0.0452), where deeper architectures (e.g., BR36 and BR27) achieve the best mean ranks, indicating superior performance in smaller dataset sizes. However, as the dataset sizes grow, the observed differences are no longer statistically significant (e.g., p = 0.2658 for 60,000; p = 0.1087 for 80,000; p = 0.9484 for 100,000; p = 0.1447 for 200,000), and the distribution of mean ranks becomes more even. For example, at 100,000 the mean ranks of all the models are almost same. This result suggests that increasing the depth of the residual block with additional convolution layers results in significantly better feature extraction in the smaller dataset sizes. And there is a diminishing return on increasing architectural depth as the dataset size grows, possibly due to the convergence of all model performances with abundant data.

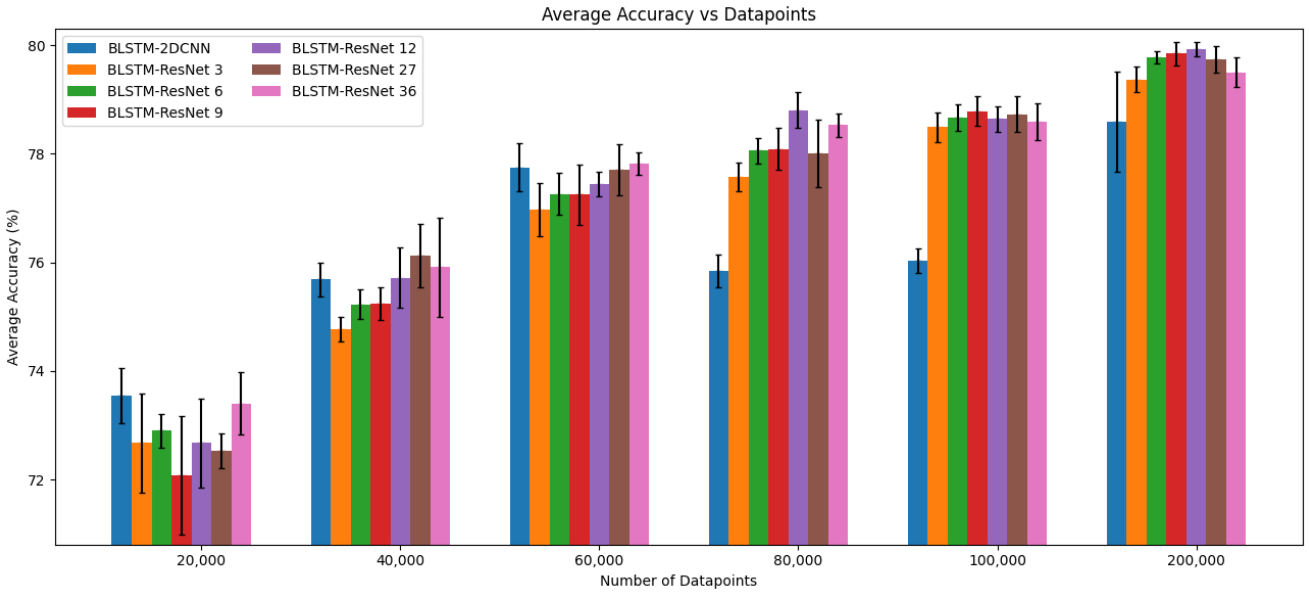


Figure 4.8: Comparison of Experiment 2 results containing baseline plus type-1 residual block with type-2 residual block models.

Finally, Figure 4.8 shows the performance of the original baseline model (BLSTM-2DCNN), the model that motivated the study, along with the Shallow and the Deep BLSTM-ResNet models. The figure has the values from Table 4.4 and Table 4.7. The original Baseline model performs well on the smallest 3 datasets. However, The Deep BLSTM-ResNet model outperforms this Baseline model in all dataset sizes except the smallest, dataset 20,000, where the accuracy of the BLSTM-ResNet-36 is only 0.19% behind the Baseline in average accuracy. In fact, for the 3 largest datasets the Deep BLSTM-ResNets outperforms the original Baseline model quite considerably (by 3.5%, 3.6% and 1.5% respectively). Even so, the shallow ResNet models still marginally outperform the Deep BLSTM-ResNet models for these 3 largest datasets.

In summary, Deep BLSTM-ResNet models generally offer improved performance compared to their shallower counterparts, especially noticeable with smaller datasets. However, their advantage becomes less pronounced as dataset size increases, where performance levels tend to be more competitive. This performance gain is largely due to the additional convolutional operations within each residual block, which extract richer features before reducing the spatial resolution. Furthermore, incorporating Type-2 residual blocks enhances the BLSTM-ResNet variants, enabling them to maintain a competitive edge over the baseline model even with the smaller datasets.

4.5 Contributions and Summary

To the best of our knowledge, no analysis has previously been conducted to investigate the depth impact of the 2D-CNN-based ResNet on sentiment analysis in any available literature. This work is the first example of using a variable depth 2D-CNN ResNet for sentiment analysis on images generated from social media text. The following contributions are collective findings of the three experiments conducted in this chapter.

This chapter introduced two different deep 2D-CNN based residual neural network models for sentiment analysis on text images. One is called shallow BLSTM-ResNet and the other deep BLSTM-ResNet. Both models replaced a shallow 2D-CNN construction of an existing model, BLSTM-2DCNN, with ResNet which was a deeper counterpart. The experimental results demonstrated that with an increase in the depth of BLSTM-ResNet, the network performance generally increases. Our best shallow BLSTM-ResNet models have achieved 2.96%, 2.76% and 1.34%

increases in average accuracy for Sentiment140 subset sizes of 80,000, 10,000 and 200,000 respectively, compared with the baseline BLSTM-2DCNN model.

In addition, an overall improvement is observed on accuracy with every additional residual block in our shallow BLSTM-ResNet model until accuracy saturates, and the same trend has been seen on the impact of dataset size on the performance. Our deep BLSTM-ResNet models show the same positive impact of network depth impact and dataset size on sentiment analysis. Further investigation on the shallow and deep BLSTM-ResNet models shows that the Deep BLSTM-ResNet model outperforms the shallow BLSTM-ResNet model in smaller dataset sizes and performs competitive with the baseline BLSTM-2DCNN network.

Additionally, the experiments revealed an exceptionally close alignment between accuracy and weighted F1 scores across all configurations for all dataset sizes. This strong correspondence indicates that both metrics provided equally reliable measures of model performance in the context of a balanced classification problem. The large sample sizes further contributed to the robustness and reliability of these metrics, as evidenced by the consistently minimal variance recorded across experiments.

At this point in my research the transformer-based networks suddenly became available and were used to improve results in many areas of research. So, the next logical step was to investigate the use of these networks, with their enhanced attention mechanisms, for sentiment analysis. Hence, In the next, chapter it will be shown that attention mechanisms can be incorporated in the models to better capture sentiment by encoding richer textual information into these Textual Images. This research was particularly motivated by the success of [138] and [123] in their respective text classification classifiers. In the work in the next chapter the primarily aim will be to leverage contextualized word embeddings generated by a pre-trained BERT [28] model to create more informative Textual Images, which is a promising direction of development and is supported by recent research.

Chapter 5

Using BERT to Generate Contextualised Textual Images for Sentiment Analysis

5.1 Introduction

This chapter addresses RQ2 by investigating whether contextualised BERT embeddings improve Textual Image classification for sentiment analysis, and whether the BLSTM stage remains beneficial after contextual information has been incorporated.

The literature motivating the use of attention mechanisms, BERT contextualised embeddings and BLSTM–Transformer combinations for Textual Image classification is reviewed in Section 2.6, particularly in the subsection addressing RQ2 and Chapter 5.

This chapter applies Pipelines 2 and 3 from Section 3.4.2 and Section 3.4.3, respectively. Pipeline 2 is implemented through the BERT–BLSTM–ResNet architecture, whereas Pipeline 3 is implemented through the direct BERT–ResNet architecture. The formation of Type A and Type B contextualised Textual Images is explained in Figure 5.1; the full descriptions of the comparison models are provided in Section 5.3.1.1.

Two complementary experiments are conducted. Experiment 1 uses Sentiment140 subsets to examine the effect of dataset size when fine-tuned BERT embeddings are processed through a BLSTM to form contextualised Textual Images. Experiment 2 uses the longer IMDB reviews to examine the effect of maximum input token length and to compare Pipeline 2, Pipeline 3 and the related comparison variants. Together, these experiments assess the performance and computational trade-offs of retaining or removing the BLSTM stage. The findings from Experiment 1 were published in [139].

5.2 Experiment 1: With Sentiment140 dataset

It has already been found that producing Textual Images improved performance [27] of sentiment analysis. This experiment investigates if there was any further improvement possible when adding contextualise embeddings before the formation of the images from text documents. The contextualised embeddings were supplied by BERT [28], a pre-trained and exceptionally well-performing model. So, to investigate if an attention mechanism would also improve the Textual Image classification task the fine-tuned BERT is used to generate contextualised word embeddings.

Contextualized Textual Images are greyscale images like the Textual Images explored in [136]. Contextualised embeddings are generated in this chapter by using the $\text{BERT}_{\text{Base}}$ neural model. The $\text{BERT}_{\text{Base}}$ model contains 12 encoder blocks of the Transformer model and can process a maximum of 512-word tokens (Refer Section 3.8.9 for details). This model is used as a feature extractor where the tokenised text document is input to the model to produce the corresponding word embeddings. The Transformer encoder model uses the attention mechanism to generate the context in the embeddings, and usually multiple attention layers are stacked, and in this case, 12 stacked layers are used. The output from the last encoder block is used as the final contextualised embedding for each word token.

Once the contextualised embeddings are generated using $\text{BERT}_{\text{Base}}$, they can either be reshaped in the form of an image (Type A) or can be transformed into images using a BLSTM block (Type B), see Figure 5.1. Images obtained from both these methods are called Contextualised Textual Images. The image dimension, in

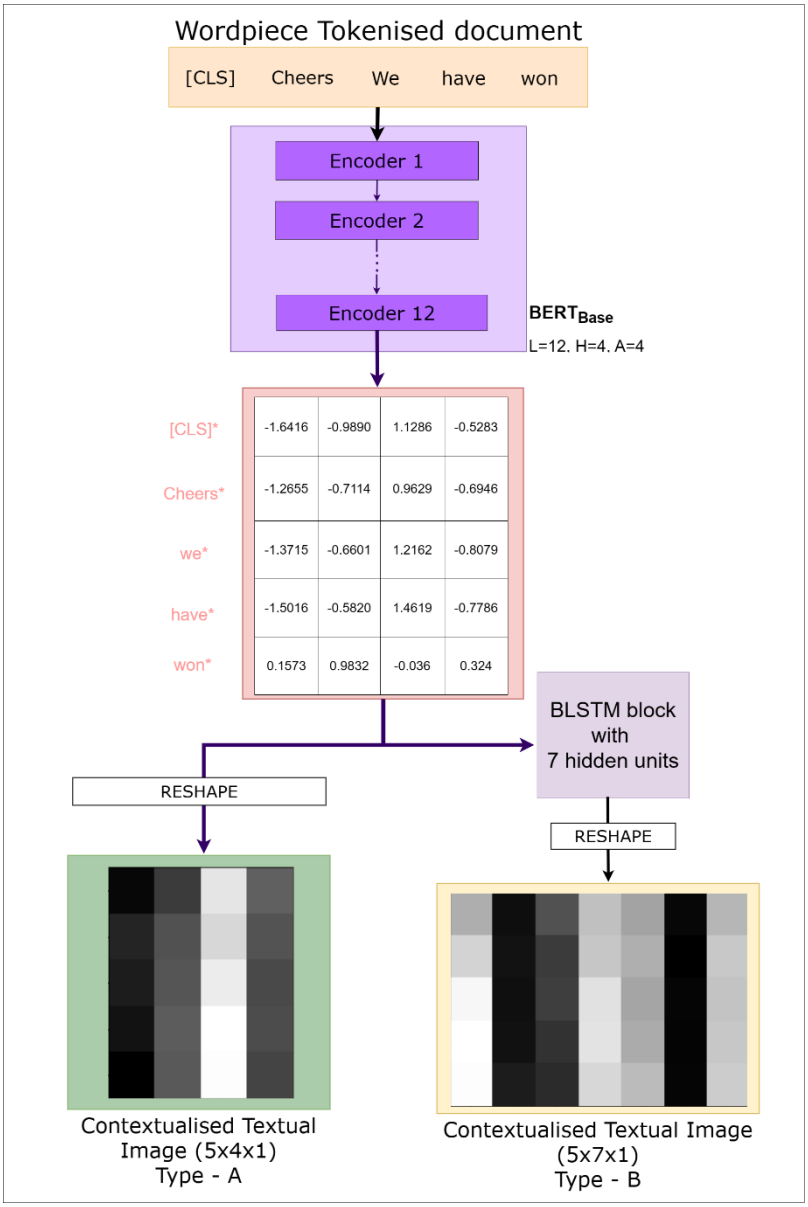


Figure 5.1: A tokenised text document is shown with five tokens, where [CLS] is a special token signifying classification task. The contextualised embeddings are generated using $\text{BERT}_{\text{Base}}$ as a feature extractor and the output of the last encoder layer is used. $\text{BERT}_{\text{Base}}$ output can directly be reshaped into a contextualised Textual Image (Type A), or it can be given to the BLSTM block and then reshaped into a contextualised Textual Image (Type B).

a Type A image, is (*document length*) × ($\text{BERT}_{\text{Base}}$ *hidden units*) × 1. In comparison, Type B has dimensions of (document *l*ength) × (BLSTM hidden *u*nits) × 1. The 'X 1' in these formulae indicates that these are just grey-scale images requiring a dimension of 1 (single channel image). These formed contextualised Textual Images can then be classified using a standard 2D ResNet model. This first experiment only uses Type B images to stay consistent with the Experiments conducted in Section 4.3. The only difference is the addition of $\text{BERT}_{\text{Base}}$ based embeddings instead of GloVe. Combining $\text{BERT}_{\text{Base}}$ contextualised embeddings as input to a BLSTM to form contextualised Textual Images and with the inclusion of different ResNet variants, allows various models to be created (refer Table 5.1).

Table 5.1: $\text{BERT}_{\text{Base}}$-BLSTM model variants and acronyms.

Model	**Acronym**
$\text{BERT}_{\text{Base}}$-BLSTM-ResNet18	BBR18
$\text{BERT}_{\text{Base}}$-BLSTM-ResNet34	BBR34
$\text{BERT}_{\text{Base}}$-BLSTM-ResNet50	BBR50

The suffix $\text{BERT}_{\text{Base}}$ indicates that the contextualised embeddings are used, and the tag BLSTM in models indicates using the BLSTM block for image generation. The ResNet followed by a number represents a ResNet model with the number of convolution layers in the ResNet block (more details in [23, 136]).

The Sentiment140 dataset is used to construct the contextualised word embeddings generated by $\text{BERT}_{\text{Base}}$. The generated contextualised Textual Images produced are compared with the Textual Images generated using GloVe word embeddings in the BLSTM-ResNet model. As the dataset consists of 1.6 million tweets (refer Section 3.3.1 for details), the dataset size impact on the contextualised Textual Images can be explored using several ResNet depths.

5.2.1 Training Setup

For simplicity $\text{BERT}_{\text{Base}}$ will be referred to as BERT from now on in this chapter. A pre-trained BERT model is fine-tuned on our training set, and the output of the last transformer block is used to generate the actual contextualised embeddings. This model is then used as the feature extraction step in our BERT-BLSTM-ResNet models.

Various subsets of Sentiment140 are used in this experiment (1,250, 2,500, 5,000, 10,000, 20,000, 40,000, 80,000, 160,000, 200,000, 400,000, 800,000 and 1,600,000) using stratified sampling from the entire Sentiment140 dataset, using a seed value of 42, to analyse model performances for a different range of data sizes. A Training/Test set split ratio of 9:1 is then used for each subset. For example, for a 20,000 subset, 18,000 samples will be used for training and 2000 samples for the testing set.

For detailed BERT and BERT-BLSTM-ResNet comparison on small dataset sizes, a 5-fold cross-validation on small data subsets is performed. Using stratified sampling, each subset is divided into five folds, ensuring that each fold maintains a representative class distribution and using the random seed of 42 for reproducibility. In each iteration of the cross-validation, four folds are combined for training, and the remaining fold is used for testing. For example, a 5,000-sample subset is split into a 4,000-sample training set and a 1,000-sample test set for each run.

The maximum token length of the Sentiment140 after data cleaning and word piece tokenisation was just below 60; therefore 60 is used as the maximum document length and padding is applied to individual text documents with shorter lengths. The number of hidden units chosen for the BLSTM model is 300 based on [27][136]. The BLSTM-ResNet12 model is used as the baseline model as it uses non-contextualised GloVe embeddings and is among the best-performing model in [136]. Also, for standardisation, the Standard ResNet model configurations proposed in [23] are used instead of custom ResNet blocks explored in [136].

The optimiser is Adam, the Batch size is 32 (this represents the maximum size that can be used given the GPU availability), and the maximum number of epochs is kept to 20. A validation set of 10% is created from the training set to evaluate the model validation performance across different training epochs. The remaining training samples are used to fine-tune the weights of the pre-trained BERT and the proposed models. The early stopping training method is adopted during training, and the best model weights are saved based on their accuracy on the validation set. The saved model weights are then used to record the performance of the model on the test set. During Experimentation, Accuracy is used for model performance comparison.

5.2.2 Result and discussion

Table 5.2: Accuracy given as a percentage and the confusion matrix ([[TP, FN], [FP, TN]]) of various models on different Sentiment140 sizes. The best result for each dataset size is highlighted in bold, or blue, if the best is BERT on its own.

Model \ Dataset size		20,000	200,000	400,000	800,000	1,600,000
BLSTM ResNet12	Accuracy	72.85%	78.82%	79.89%	80.68%	81.74%
	[[TP, FN], [FP, TN]]	[[726, 274], [269, 731]]	[[7901, 2099], [2137, 7863]]	[[15963, 4037], [4007, 15993]]	[[32301, 7699], [7757, 32243]]	[[65335, 14665], [14551, 65449]]
BERT-BLSTM-ResNet18	Accuracy	**83.40%**	**83.50%**	84.61%	85.54%	84.50%
	[[TP, FN], [FP, TN]]	[[836, 164], [168, 832]]	[[8330, 1670], [1630, 8370]]	[[16939, 3061], [3095, 16905]]	[[34194, 5806], [5762, 34238]]	[[67661, 12339], [12461, 67539]]
BERT-BLSTM-ResNet34	Accuracy	82.20%	83.10%	**84.65%**	85.68%	84.66%
	[[TP, FN], [FP, TN]]	[[819, 181], [175, 825]]	[[8329, 1671], [1709, 8291]]	[[16947, 3053], [3087, 16913]]	[[34238, 5762], [5694, 34306]]	[[67659, 12341], [12203, 67797]]
BERT-BLSTM-ResNet50	Accuracy	82.55%	83.00%	84.58%	**85.72%**	**84.73%**
	[[TP, FN], [FP, TN]]	[[823, 177], [172, 828]]	[[8283, 1717], [1683, 8317]]	[[16898, 3102], [3066, 16934]]	[[34322, 5678], [5746, 34254]]	[[67849, 12151], [12281, 67719]]
BERT	Accuracy	81.00%	83.24%	85.60%	86.10%	85.07%
	[[TP, FN], [FP, TN]]	[[808, 192], [188, 812]]	[[8338, 1662], [1690, 8310]]	[[17107, 2893], [2867, 17133]]	[[34465, 5535], [5585, 34415]]	[[68015, 11985], [11903, 68097]]

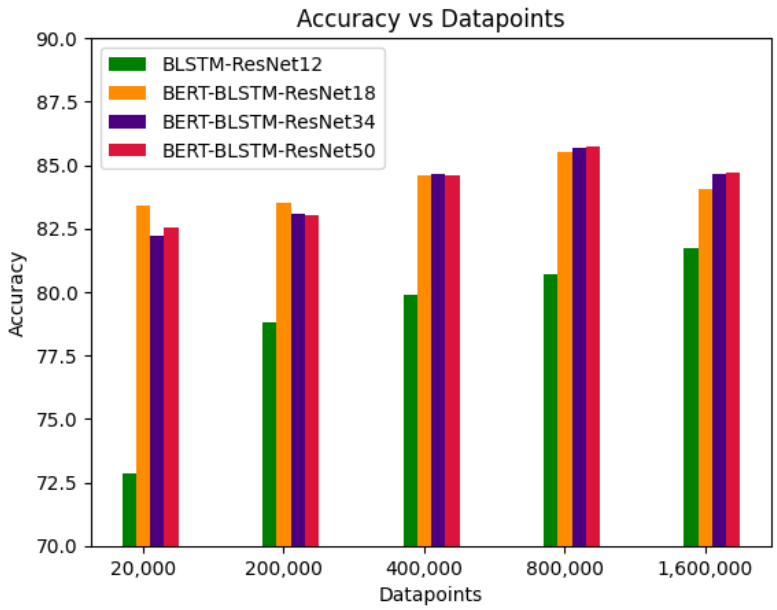


Figure 5.2: Accuracy vs Dataset size for subset 20,000 to 1,600,000. Comparison between BERT-BLSTM-ResNet models with the Baseline model BLSTM-ResNet-12.

Subsets of size 20,000, 200,000, 400,000, 800,000, and 1,600,000 of the Sentiment140 dataset are selected. The BERT-BLSTM-ResNet models and BLSTM-ResNet12 are trained on these subsets, and the corresponding accuracy is shown in the first four rows of Table 5.2. To help visualise the results of the BERT-BLSTM-ResNet variants and the BLSTM-ResNet12 performance Figure 5.2 is plotted. Figure 5.2 clearly shows that using a fine-tuned BERT contextualised embedding with the BLSTM-ResNet model significantly improved the performance compared to the Baseline (best previous) model. This demonstrates the significance of using context-aware embeddings in Textual Image classification.

Since 20,000 is a small dataset size, the ResNet18 variant performed better than other BERT-BLSTM-ResNet variants. As the size of the dataset gets larger, the additional depth of the models can be made use of so that first, the ResNet34 variant is best, and then the ResNet50 variant takes over. These findings are in line with the findings of [136].

Since introducing BERT to BLSTM-ResNet produced a significant improvement, it will be appropriate to know the performance of fine-turned BERT by itself, which curiously, many publications involving fine-tuned BERT do not include. Interestingly, as the blue results in the last row of Table 5.2 and Figure 5.3 show, BERT on its own either

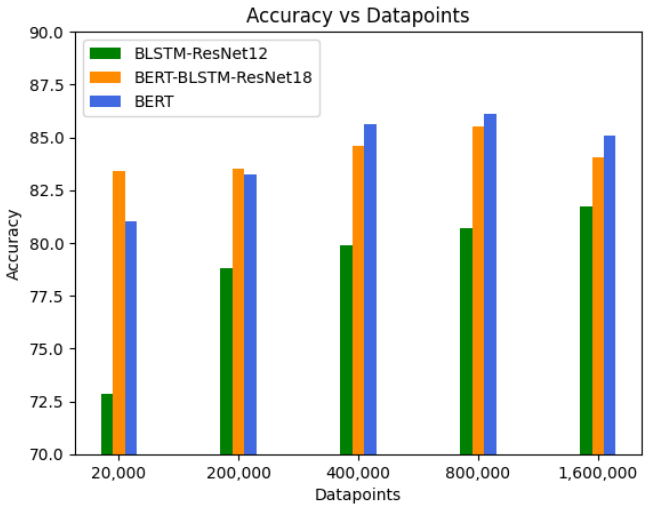


Figure 5.3: Accuracy vs Dataset size for subsets from 20,000 to 1,600,000. Comparison between BERT-BLSTM-ResNet models and the Baseline BLSTM-ResNet-12 with BERT

outperforms all the other models or gives close results. Significantly, it seems that the dataset size is the deciding factor. BERT alone is better for datasets of a size of about 200,000 and larger but is less good for smaller datasets. Additionally, based on the confusion matrices in Table 5.2, the percentage difference in class predictions for each model decreases as the dataset size increases. For the entire table, this difference in true positive and negative ranges from 0.05125% (BERT at 1,600,000) to 0.3% (BERT-BLSTM-ResNet34 at 20,000) of the test set size.

Table 5.3: Model accuracy and standard deviation across different dataset sizes.

Model \ Dataset		1,250	2,500	5,000	10,000	20,000	40,000	80,000	160,000
BERT	Accuracy	71.98%	80.56%	78.05%	80.22%	81.03%	82.21%	**83.53%**	**84.21%**
	Standard deviation	2.02%	1.46%	0.42%	0.58%	0.53%	0.22%	0.17%	0.09%
BBR18	Accuracy	**82.80%**	**87.40%**	**89.70%**	**85.53%**	**86.37%**	**83.78%**	82.15%	82.98%
	Standard deviation	4.60%	3.64%	5.77%	1.48%	3.39%	6.01%	0.99%	1.16%

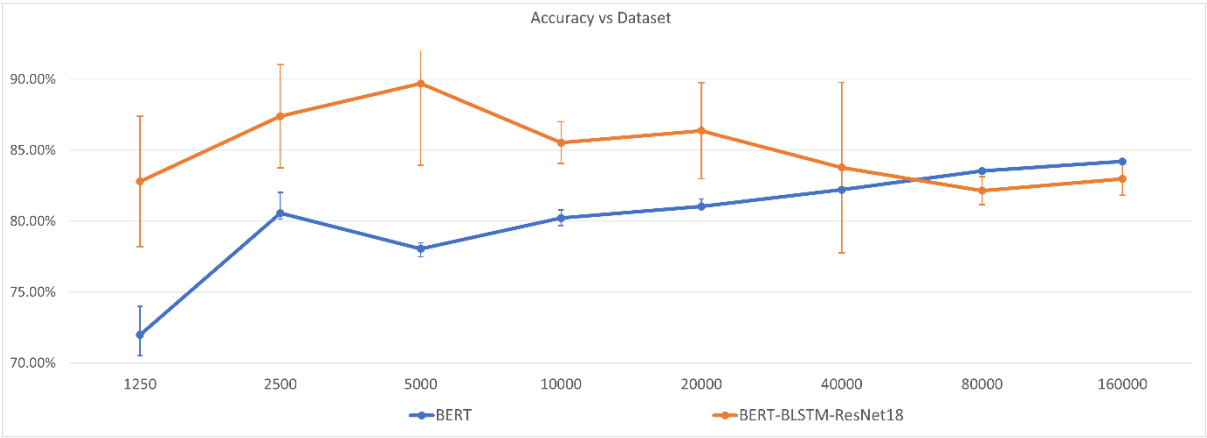


Figure 5.4: 5-fold cross validation Accuracy vs Dataset size plot for the smaller subsets of Sentiment140 from 1250 - 160,000. Comparison between BERT-BLSTM-ResNet 18 model with BERT.

To further investigate this result and justify the above claim, a more computationally intensive task of using 5-fold cross-validation on the results of several smaller subsets is performed. This is shown in Figure 5.4 and Table 5.3. These give a value for the margin of error and shows a point around 60,000 where the two models are very similar and is a possible crossover point. Sentiment140 smaller subset sizes 1250, 2500, 5,000, 10,000, 20,000, 40,000, 80,000 and 160,000 were used (refer Appendix Section 11.1 for individual fold performance). This is an important result as it establishes that BERT requires more data to optimise the Sentiment140 data fully and needs a certain minimum dataset size to release its full potential. Cross-validation is performed only on BERT and BBR18 because of the lack of available computation resources.

To visualise all the model performances from a smaller to a bigger dataset size, a single BERT is trained on various subsets. Then, the model weights were used to train all BERT-BLSTM-ResNet variants, see Table 5.4. The values are then plotted in logarithmic scale, see Figure 5.5. This plot captures the entire trend of the model performances. BERT underperforms initially and then overtakes as the dataset size increases. BBR18 performs best till 40,000 subsets and is then mostly superseded by other models. BERT seems to be performing well on its own, but for a

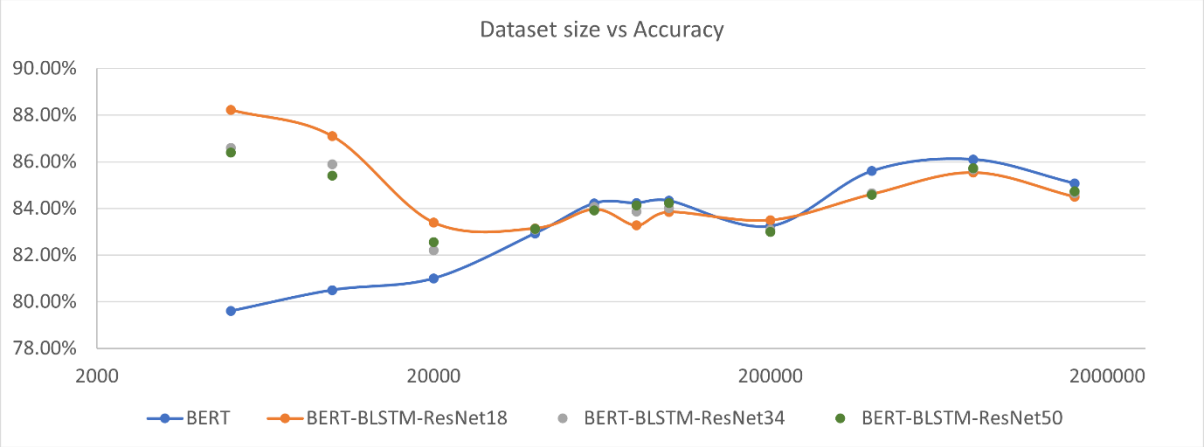


Figure 5.5: Log plot of Accuracy vs Dataset size over various datasets. Comparison between BERT-BLSTM-ResNet models with BERT. For clarity, only the BERT and ResNet18 variant are shown with a line to illustrate trends.

5.3 Experiment 2: with IMDB dataset

This study investigates the performance of BERT and BERT extended models using BLSTM and/or ResNet blocks across varying token lengths (32, 64, 128, 256, 384, 512). For each token length, a comparison is made between a baseline BERT model and four other model architectures: BERT-BLSTM, BERT-BLSTM-ResNet50, BERT-BLSTM||ResNet50, and BERT-ResNet50. The Sentiment140 dataset has a very short document length (maximum 38 words) and $\text{BERT}_{\text{Base}}$ can process 512 tokens in each instance. Not every word equals one token, some words can be converted into multiple tokens based on the tokenisation used. For BERT model, the WordPiece tokenizer is used which breaks up words into several tokens (refer Section 3.8.8.1 for details). So, to investigate the token length impact on the model performance data with a longer document length is needed. The IMDB review dataset is ideal for this purpose. Figure 5.6 show that both the training and test sets have documents with long token length. Hence, different train-test datasets pairs of various token lengths can be created.

5.3.1 Experimental Setup:

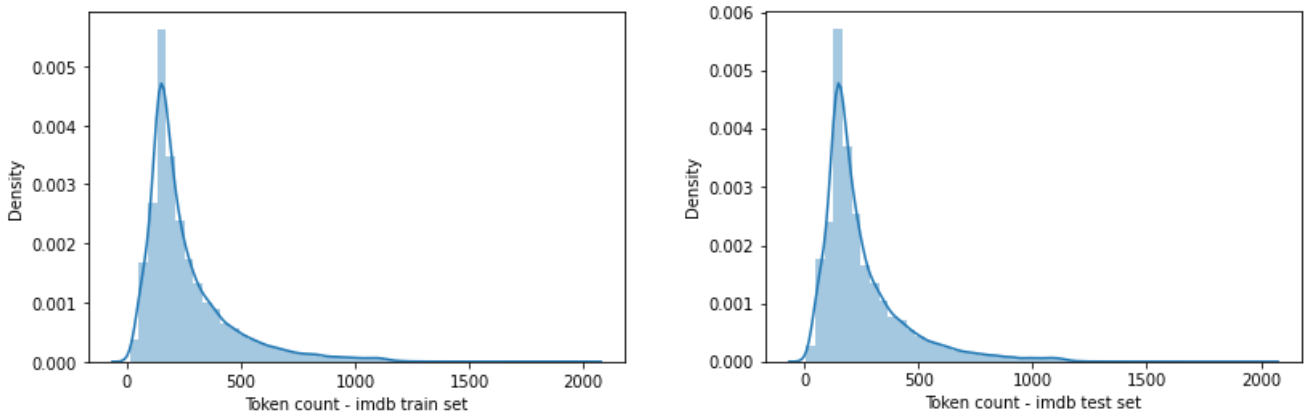


Figure 5.6: Token Density plot for IMDB train (left) and test set (right)

Figure 5.6 The pre-trained BERT component in all models used in this comparison is fine-tuned first on the new data and then used as a feature extractor. This dependency implies that the performance measurements for the compared models are not independent. For each model comparison at a given token length the models are trained 10 times to get an average accuracy and the standard deviation. Multiple configurations integrating BLSTM and ResNet networks were investigated to assess the influence of each architecture on sentiment classification outcomes, depending on the nature of the contextualised Textual Image representations employed. Refer Figure 5.7 for all the models used in this experiment and the following subsection provide some explanation for each model.

When considering computational cost Floating Point Operations (FLOPs) are used to compare BERT and BERT extended model computation requirement as this is a hardware agnostic method to compare model computation requirements.

5.3.1.1 Models:

1. BERT (Baseline): The BERT model is used as a baseline for comparison. It is a straightforward fine-tuning of the pre-trained BERT architecture for the sentiment classification task. The output of the BERT encoder is passed through a classification head without any additional sequence modeling using a recurrent network or any convolutional layers. This model serves as a reference point to evaluate the performance improvements introduced by integrating additional architectures such as BLSTM and ResNet.

2. BERT-BLSTM: In this configuration, the output of the BERT encoder is treated as a contextualized sequence and passed directly into a Bidirectional Long Short-Term Memory (BLSTM) network. The BLSTM layer is used to further model sequential dependencies in the textual data. Unlike the image-based configurations, this model remains in the textual feature space and does not utilize any convolutional neural network. It helps in understanding the benefit of sequence modeling after BERT, without image transformation.

3. BERT-BLSTM-ResNet: This model builds upon the BERT-BLSTM combination by converting the BLSTM outputs into 2D matrices that are then interpreted as Type B contextualised Textual Images as used in Section 5.2. These image representations are then fed into a ResNet model for classification. The BLSTM again processes the sequential output of BERT. This configuration is designed to leverage both temporal sequence modeling and spatial pattern recognition by fusing BLSTM and ResNet architectures.

4. BERT-BLSTM || ResNet (Type A): In this variant, the output from the BERT model is simultaneously processed by two parallel paths: One path passes the BERT embeddings through a BLSTM network like the BERT-BLSTM model. The other path converts the BERT output directly into a Type A contextualised Textual Image without any BLSTM processing. This image is then used as input to the ResNet model which is like the BERT-ResNet model. The final classification combines features from both the BLSTM and ResNet outputs by flattening both their outputs and concatenating them before passing the combined feature through the classification head. This configuration allows for capturing both sequential and spatial features independently, assessing how their parallel combination affects sentiment classification performance.

5. BERT-ResNet: In this model, the output of the BERT encoder is transformed into a Type A contextualised Textual Image, which is then processed directly by a ResNet-50 convolutional neural network for sentiment classification. This architecture excludes any intermediate BLSTM layer, allowing the model to focus exclusively on the spatial features captured from BERT's contextual embeddings. The goal of BERT-ResNet configuration is to assess whether BERT alone is sufficient for generating meaningful spatial representations of text that can be effectively interpreted by a CNN. By avoiding the sequential processing overhead of BLSTM networks, this model can benefit from parallel computation on GPUs, which results in faster training and reduced model parameters. This setup is particularly valuable for determining if the sequential dependencies typically modelled by BLSTMs can instead be captured through the spatial encoding of BERT representations, enabling a more scalable and computationally efficient pipeline for sentiment analysis.

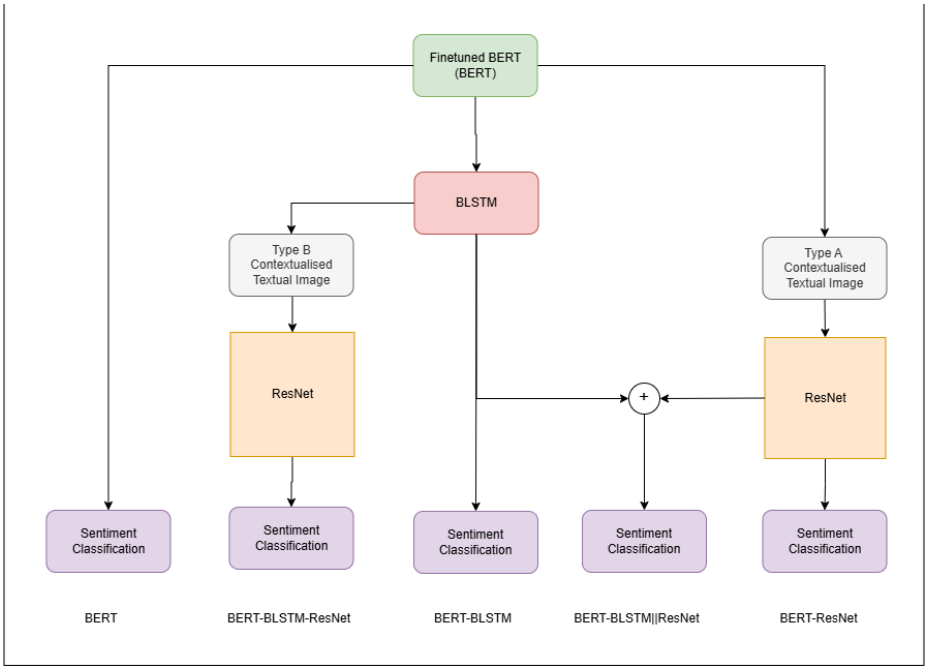


Figure 5.7: BERT and different BERT extended models including BLSTM and/or ResNet. Different configurations use different types of contextualised Textual Image.

5.3.2 Training Strategy

A pre-trained $\text{BERT}_{\text{Base}}$ model (referred to as BERT in below discussion) is fine-tuned on the IMDB dataset to generate contextualised embeddings, which are then used as input features across all subsequent architectures. To investigate the impact of input token length on sentiment classification performance, the dataset is divided into subsets with token lengths of 32, 64, 128, 256, 356, and 512. The token lengths primarily follow a doubling pattern; however, a token length of 356 is additionally included to better observe any subtle changes in models' performance between 256 and 512 tokens, allowing for a more detailed analysis.

To evaluate the effect of token length on performance, both the training and testing sets are processed using a consistent token length for each experiment. For example, in the 32-token configuration, only the first 32 tokens of each input sample are used in both the training and testing sets during training and evaluation.

Ten separate runs of each model are trained as the deep neural networks are non-deterministic during training and each time a model is trained the performance varies. Instead of average cross validation accuracies, average test accuracy across ten different trained models is recorded due to the availability of a separate test set for the IMDB dataset. The average accuracies and the standard deviation are then used for comparing the model performances across different model configurations as well as exploring their performance across datasets of different token lengths. All the models require the BERT model as the feature extractor so first the BERT model is fine-tuned and then all the Models use the same BERT features to generate contextualised textual embeddings.

ResNet50 is used for this experiment as it uses a bottle neck approach in its residual block construction and is a popular model in computer vision for benchmarking. The ResNet model is initialised with random weights, all the models involving a BLSTM network use 300 hidden units and the same training strategies, and setup is used, as utilised in Section 5.2.1.

5.3.3 Results:

The Table 5.5 and Figure 5.8 present the average model accuracy and standard deviation across 10 runs for various BERT-based architectures at different token lengths (32, 64, 128, 256, 384, 512) (Refer Appendix Section 11.2 or individual run performance). A consistent trend observed across all models is that accuracy generally improves as the token length increases. This is expected, as longer sequences typically provide more contextual information, allowing the models to better gauge the associated sentiments. The gains in accuracy are particularly prominent when moving from shorter lengths (32, 64) to medium lengths (128, 256) and then plateau as the token length approaches 384 and 512.

Table 5.5: Accuracy vs Token length for BERT and BERT Extended models. The best result for each token length is highlighted in bold.

Model \ Token size	32	64	128	256	384	512
BERT	76.28 ± 1.29	82.32 ± 0.99	88.60 ± 0.83	91.56 ± 0.34	92.55 ± 0.21	93.05 ± 0.17
BERT-BLSTM	77.95 ± 3.01	**84.44 ± 2.15**	89.26 ± 1.52	91.96 ± 0.58	93.03 ± 0.60	93.39 ± 0.45
BERT-BLSTM-ResNet50	78.20 ± 2.77	83.49 ± 1.93	**89.87 ± 1.84**	**92.19 ± 1.17**	**93.15 ± 0.53**	**93.61 ± 0.59**
BERT-BLSTM‖ResNet50	**78.36 ± 2.86**	83.72 ± 2.09	89.17 ± 1.10	91.86 ± 0.69	92.85 ± 0.40	93.54 ± 0.31
BERT-ResNet50	77.25 ± 2.21	82.86 ± 1.35	88.97 ± 1.02	91.86 ± 0.59	93.03 ± 0.28	93.42 ± 0.37

For token lengths of 384 and 512, the accuracy of all models converges to a similarly high level, and the standard deviations become small. This suggests that at longer contexts, the models reach saturation in their ability to extract further useful information from the additional tokens increases in context beyond a certain point.

BERT shows a steady increase in accuracy with token length, going from 76.28% to 93.05% average accuracy as the model processed incrementally longer token lengths. The standard deviation consistently drops from 1.29% to 0.17% with increasing token length, indicating consistent performance across runs as the token length increases. For instance, at 512 tokens, it achieves 93.05±0.17, showing small deviation in the results.

BERT-BLSTM consistently outperforms BERT, especially at shorter and medium token lengths. For example, at 32 and 64 tokens, BERT-BLSTM has an accuracy of 77.95±3.01 compared to BERT's 76.28±1.29. This suggests that the BLSTM layer effectively captures local sequential dependencies that enhance performance, particularly when the initial context is limited. The model average accuracy is best among all models for the sequence length 64 highlighting the capabilities of having BLSTM for sentiment extraction for shorter sequence lengths. However, its standard deviation is often higher than BERT, particularly at shorter lengths (e.g., ±3.01 at 32 tokens), indicating

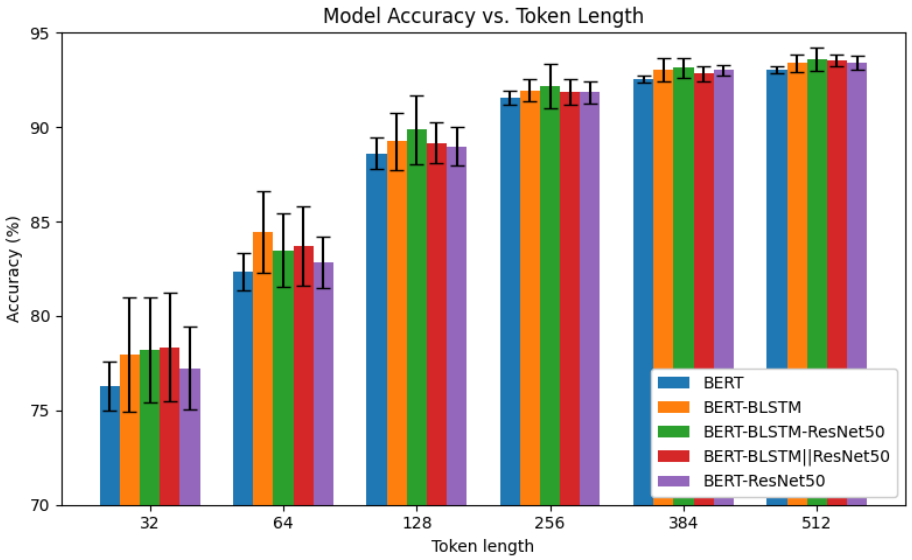


Figure 5.8: Accuracy vs Token length for BERT and BERT Extended models

more variability in its performance across different runs. This variability might be due to the increased complexity of the BLSTM layer and its training dynamics.

BERT-BLSTM-ResNet50 generally performs at par with or slightly better than BERT-BLSTM. The inclusion of ResNet50 after BLSTM appears to provide marginal gains at most lengths and the model outperforms most other model performances from 128 token length onwards, demonstrating its robustness. However, the standard deviation is among the highest at medium to large token lengths compared to others like 1.17 at 256 and 0.59 at 512.

BERT-BLSTM||ResNet50, where BLSTM and ResNet50 operate in parallel, also consistently outperforms BERT for all token lengths. It often shows competitive or slightly better accuracy than BERT-BLSTM-ResNet50, particularly at 32 (78.36±2.86) and 64 tokens (83.72±2.09), achieving the highest accuracy at 32 tokens. The parallel integration seems to leverage the strengths of both components effectively. Its standard deviations are mostly lower than the BERT-BLSTM-ResNet50 model showcasing more consistency in its performance.

Interestingly, **BERT-ResNet50**, which integrates ResNet50 directly with BERT, also shows consistent improvements over BERT, particularly at medium token lengths. It performs the second best at 384 (93.03 ± 0.28) sharing the place with BERT-BLSTM with a much lower standard deviation. In general, the model has the lowest standard deviation after BERT, suggesting greater stability in its performance when trained. Its performance is mostly close to the BLSTM-enhanced models, especially at longer token lengths, reinforcing the idea that ResNet50 is effective at extracting useful information from reshaped BERT features directly.

The results clearly indicate that augmenting BERT with either BLSTM, ResNet50, or their combinations consistently leads to accuracy improvements, especially for shorter token lengths where the baseline BERT might lack sufficient context. While the added complexity is beneficial at shorter token lengths, the performance differences among all models become negligible at longer token lengths (384 and 512). This implies that for tasks where longer contexts are naturally present, or can be provided, the simpler BERT might suffice, or the computational overhead of the additional layers might not be justified by the marginal accuracy gains.

BERT-BLSTM-ResNet50 and BERT-BLSTM||ResNet50 generally show a good accuracy gain over BERT, especially at shorter token lengths. BERT-ResNet50 also demonstrates a strong and stable performance. The choice between these might depend on computational resources and the specific characteristics of the data at hand. The sequential nature of the BLSTM does mean longer training times and incapability to parallelise the training across processors as do the Transformer and CNN networks.

Model Parameters and FLOPs The Table 5.6 contains the computational cost, measured in Floating Point Operations (FLOPs), for different BERT-based models across varying token lengths (32, 64, 128, 256, 384, 512). The table also provides the number of parameters for each model. Figure 5.9 presents a nice visual representation of computation requirements measured in FLOPs of the various models against the token length.

Table 5.6: Different Model Parameters as well as FLOPs for various token lengths.

Model	Parameters	32	64	128	256	384	512
BERT	109.48 $\times 10^6$	5.442 $\times 10^9$	10.883 $\times 10^9$	21.764 $\times 10^9$	43.527 $\times 10^9$	65.290 $\times 10^9$	87.053 $\times 10^9$
BERT-BLSTM	112.06 – 112.20 $\times 10^6$	5.607 $\times 10^9$	11.212 $\times 10^9$	22.423 $\times 10^9$	44.844 $\times 10^9$	67.266 $\times 10^9$	89.688 $\times 10^9$
BERT-BLSTM-ResNet50	135.55 $\times 10^6$	7.188 $\times 10^9$	14.376 $\times 10^9$	28.750 $\times 10^9$	57.499 $\times 10^9$	86.247 $\times 10^9$	114.996 $\times 10^9$
BERT-BLSTM||ResNet50	135.56 – 135.71 $\times 10^6$	9.577 $\times 10^9$	19.153 $\times 10^9$	38.304 $\times 10^9$	76.607 $\times 10^9$	114.910 $\times 10^9$	153.213 $\times 10^9$
BERT-ResNet50	132.99 $\times 10^6$	9.412 $\times 10^9$	18.823 $\times 10^9$	37.645 $\times 10^9$	75.289 $\times 10^9$	112.934 $\times 10^9$	150.578 $\times 10^9$

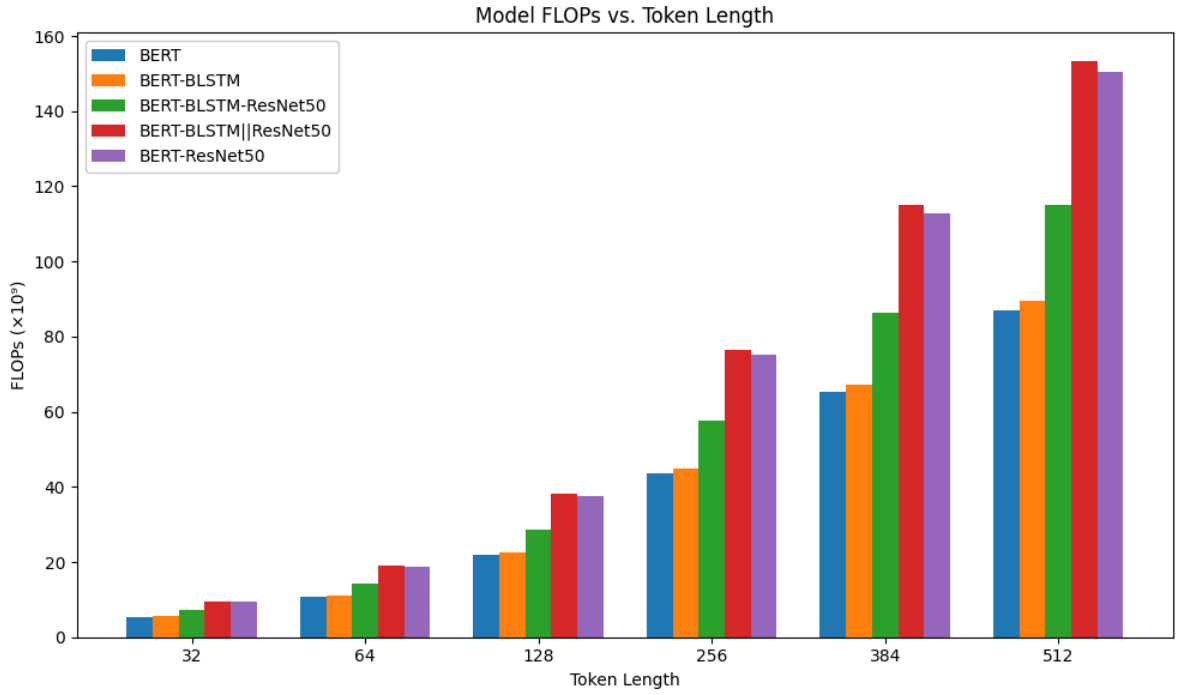


Figure 5.9: FLOPs vs Token length for BERT and extended Models

The baseline BERT model contains 109.48 million parameters. Adding a BLSTM layer (BERT-BLSTM) results in a minor increase to approximately 112.06M–112.20 million parameters. Integrating the ResNet50 component causes a more substantial jump in model size, as seen in the BERT-ResNet50 model, which has 132.99 million parameters. Models incorporating both the BLSTM and ResNet50 components are the largest; the BERT-BLSTM-ResNet50 has 135.55 million parameters, while its parallel counterpart, BERT-BLSTM||ResNet50, has a nearly identical count at 135.56–135.71 million, indicating that the parallel integration method results in a comparable model size.

For all models, the computational cost measured in FLOPs exhibits a clear upward trend with token length increase which is an expected behaviour for sequence processing models like text models, as computational cost scales proportionally with the input sequence size. Doubling the token length roughly doubles the number of FLOPs.

The BERT baseline model has the smallest cost in terms of FLOPs over all token lengths. While the BERT-BLSTM model consistently requires slightly more FLOPs than the base BERT model across all token lengths. This is due to the additional computations involved in the BLSTM layer, which processes the contextualized embeddings from BERT. The increase is relatively minor compared to the ResNet-inclusive models.

The models incorporating ResNet50 (BERT-BLSTM-ResNet50, BERT-BLSTM||ResNet50, and BERT-ResNet50) are more computationally expensive than BERT and BERT-BLSTM. This is evident from their much higher FLOP counts at every token length. The BERT-BLSTM||ResNet50 model consistently shows a higher FLOPs than BERT-BLSTM-ResNet50, with increasing difference at longer token lengths. This is because the model combines the computational burden of both BLSTM and ResNet50 in parallel. The ResNet component, designed for image processing, has many convolutional and pooling operations that contribute heavily to the FLOP count.

While still significantly higher than BERT, BERT-ResNet50 generally has slightly lower FLOPs than BERT-

BLSTM||ResNet50. This highlights that the BLSTM layer, while small in terms of parameters, does add to the overall computational cost.

The fact that BERT-ResNet50 has more model parameters than BERT-BLSTM-ResNet50, even without any BLSTM block, is due to the difference in the size of the input image to the ResNet network. The BLSTM hidden units dictate the resultant image size (which is 300) for the Type B Textual Image, whereas the Type A Textual Image is dependent only on output of BERT which has 512 features at each sequence. Therefore, the models using Type A Textual Images (BERT-ResNet50 and BERT-BLSTM||ResNet50) have higher model parameters and have high slope of model computational requirements than the BERT-BLSTM-ResNet50 with increasing token lengths.

Recurrent Neural Networks (RNNs), like BLSTM, process sequences token by token, maintaining a hidden state that is updated sequentially. This inherent sequential dependency makes them difficult to parallelize effectively across the sequence dimension during training. While computations within a single time step can be parallelized, processing the next token often requires the output from the previous token. This sequential bottleneck can significantly slow down training, especially for very long sequences, as the computations must largely proceed one after another.

Convolutional Neural Networks (CNNs), like ResNet, in contrast, are inherently parallelizable. Convolutional operations, which are the core of CNNs, involve applying filters across local regions of the input. These operations can be performed simultaneously across different parts of the Textual Image, independent of each other. This allows for massive parallelization on modern hardware (GPUs, TPUs), where many operations can be computed concurrently.

While models with BLSTM might have fewer total FLOPs, the inability to compute them in parallel means that its training time could be longer than a CNN with a higher FLOP count but high parallelizability [140]. BERT-ResNet50 models have a higher total FLOP count than BERT-BLSTM-ResNet50, but, with the ability to parallelize operations, result in reduced training time on GPUs and TPUs. This can translate to faster training times per epoch compared to architectures heavily reliant on sequential processing, even for complex models. This is a critical advantage in deep learning research and deployment, where computational efficiency directly impacts iteration speed.

5.3.4 Statistical Analysis of BERT vs BERT extended Models Performance

The standard deviations are important. While the BERT enhanced models achieve higher average accuracies over baseline (BERT), some show very high variability across runs. This suggests a need to perform statistical testing to understand the significance of their performance's gains.

This statistical analysis of the performance differences between various deep learning models, incorporating Textual Images, specifically compares different architectures against a baseline BERT model. The analysis utilizes Wilcoxon Signed-Rank Test, a non-parametric statistical approach, to assess the significance of observed performance differences.

Table 5.7: BERT vs Extended model performance analysed using the Wilcoxon significance test

Token Length	Wilcoxon Statistic	p-value	Effect Size (r)	Power
BERT vs BERT-BLSTM				
32	5.0	0.0391	-0.7	0.468
64	1.0	0.0039	-0.8	0.557
128	9.0	0.0645	-0.8	0.557
256	7.0	0.0371	-0.6	0.381
384	6.0	0.0254	-0.8	0.557
512	8.5	0.0527	-0.4	0.226
BERT vs BERT-BLSTM-ResNet50				
32	4.0	0.0137	-0.6	0.381
64	7.5	0.0410	-0.4	0.226
128	7.0	0.0371	-0.6	0.381
256	7.0	0.0332	-0.8	0.557
384	0.0	0.0020	-1.0	0.723
512	1.0	0.0039	-0.8	0.557
BERT vs BERT-BLSTM||ResNet50				

Continued on next page

Token Length	Wilcoxon Statistic	p-value	Effect Size (r)	Power
32	6.0	0.0273	-0.4	0.226
64	1.0	0.0039	-0.8	0.557
128	4.0	0.0137	-0.8	0.557
256	10.0	0.0840	-0.6	0.381
384	0.0	0.0020	-1.0	0.723
512	0.0	0.0020	-1.0	0.723
BERT vs BERT-ResNet50				
32	7.5	0.0410	-0.6	0.381
64	6.0	0.0254	-0.8	0.557
128	4.0	0.0137	-0.6	0.381
256	5.0	0.0391	-0.7	0.468
384	1.0	0.0039	-0.8	0.557
512	5.0	0.0195	-0.6	0.381

Given the experimental design, the choice of the Wilcoxon Signed-Rank Test is appropriate due to the inherent dependency between the comparison groups. Since BERT is fine-tuned and then used as a feature extractor for the other models, the measurements are paired or related. Each experiment run for a specific token length generates a performance score for BERT and a corresponding score for the other models. This "before-and-after" nature of the data violates the independence assumption of parametric tests like the independent samples t-test. The Wilcoxon Signed-Rank Test is specifically designed for such paired or dependent samples.

Deep learning model performance metrics may not adhere to the assumptions of parametric tests, such as normality of differences in small-sample or non-independent and non-identically distributed settings. The Wilcoxon Signed-Rank Test does not assume a specific distribution of the differences between paired observations. With only 10 experiments for each comparison, the sample size is relatively small. In such scenarios, relying on the central limit theorem for normality (which would justify a paired t-test) can be risky.

The Wilcoxon Signed-Rank Test is good alternative for smaller sample sizes where distributional assumptions are difficult to verify. A significance level (α) of 0.05 is used for interpreting p-values. Effect size (r) is interpreted based on Cohen's guidelines: $|r| = 0.1$ - 0.3 (small effect), $|r| = 0.3$ - 0.5 (moderate effect), and $|r| \geq 0.5$ (large effect). A negative effect size indicates that BERT performed worse than the extended model.

This analysis investigates the performance differences between several deep learning models based on BERT, specifically comparing BERT against BERT-BLSTM, BERT-BLSTM-ResNet50, BERT-BLSTM||ResNet50, and BERT-ResNet50 across various token lengths.

The Table 5.7 contains the statistical values for the comparison. The Wilcoxon signed-rank test determines statistical significance, with p-values less than 0.05 indicating a significant difference. The effect size (r) measures the magnitude and direction of the observed differences, and power indicates the probability of detecting a true effect if one exists.

General Null Hypothesis: For each comparison, the null hypothesis states that there is no significant difference in performance between BERT and the Other model at a given token length.

1. BERT vs. BERT-BLSTM: This comparison evaluates whether augmenting BERT with a Bidirectional Long Short-Term Memory (BLSTM) layer significantly impacts performance.

At token lengths 32, 64, a highly significant difference was observed with (p=0.0391, r=-0.7), (p=0.0039, r=-0.8) respectively. A significant difference was also observed at 256 with (p=0.0371, r=-0.6) where BERT-BLSTM showed better performance and at 384, where BERT-BLSTM clearly outperformed BERT with a significant difference of (p=0.0254, r=-0.8).

Token length 128, did not register a statistically significant result with (p=0.0645), although the effect size (r=-0.8) was large, indicating a potential trend where BERT-BLSTM still performs better. Whereas 512 token length showed no statistically significant difference with (p=0.0527), but with moderate effect size (r=-0.4), suggesting a weaker advantage for BERT-BLSTM at this length.

BERT-BLSTM generally outperformed BERT across most token lengths with strong negative effect sizes indicating BERT's inferior performance. The strongest effects were observed at token lengths 64 and 384.

2. BERT vs. BERT-BLSTM-ResNet50: This comparison investigates the impact of combining BLSTM with a ResNet50 architecture on top of BERT.

A significant difference was observed at token length 32 (p=0.0137, r=-0.6) and 128 (p=0.0371, r=-0.6) where

BERT lagged in average performance compared to BERT-BLSTM-ResNet50 (76.28, 88.60 vs 78.20, 89.87 for token length 32, 64 respectively). Whereas borderline significance was observed at token length 64 (p=0.0410, r=-0.4) with a moderate effect size.

BERT-BLSTM-ResNet50 consistently outperformed BERT, especially at longer token lengths (256, 384, 512), with very strong negative effect sizes (up to r=-1.0). This very strong negative effect size suggests that the BERT-BLSTM-ResNet50 model consistently and substantially outperformed BERT at these lengths and that the BLSTM-ResNet50 architecture effectively enhances BERT's performance.

In this architecture, ResNet50 processes features that have already been transformed by the BLSTM layer. The consistent and often very strong negative effect sizes (e.g., r=-1.0 at 384 and r=-0.8 at 512) suggest that ResNet50 effectively extracts additional hierarchical or spatial patterns from the sequence-processed features provided by BLSTM. This indicates a powerful synergy, where BLSTM captures temporal dependencies, and ResNet50 then leverages these enriched features, presented as a "Textual Image," to further enhance performance.

3. BERT vs. BERT-BLSTM||ResNet50: This comparison examines an alternative integration of BLSTM and ResNet50 in parallel with BERT.

Here, ResNet50 operates on features that are processed in parallel to the BLSTM branch. The performance of this model, with two instances of r=-1.0, highlights ResNet50's strong ability to extract valuable information from the BERT features when presented in a Textual Image format. The parallel processing suggests that both the sequential information (from BLSTM) and Textual Image representations (fed to ResNet50) contribute independently and significantly to the overall improved performance. This implies that even without sequential transformation by BLSTM, ResNet50 can still discern crucial patterns from the reshaped BERT features.

At token length 32, a moderately significant difference was observed (p=0.0273, r=-0.4). BERT-BLSTM||ResNet50 showed better performance. whereas, for token length 256, no statistically significant difference was detected (p=0.0840), though the effect size (r=-0.6) was moderate.

However, this model consistently and substantially outperformed BERT, at all other token lengths (64, 128, 384, and 512), with some of the strongest effect sizes observed (r=-1.0). The parallel integration appears highly effective in extracting more sentiment from the BERT features.

4. BERT vs. BERT-ResNet50: This comparison investigates the impact of directly adding a ResNet50 architecture to BERT, without the BLSTM component.

A significant difference was observed at token length 32 (p=0.0410, r=-0.6), 64 (p=0.0254, r=-0.8), 128 (p=0.0137, r=-0.6), 256 (p=0.0391, r=-0.7) and 512 (p=0.0195, r=-0.6) where BERT-ResNet50 showed better performance. At token length 384, a highly significant difference was observed (p=0.0039, r=-0.8), suggesting a strong improvement.

In this simpler configuration, ResNet50 directly processes the features from BERT, reshaped into a Textual Image. The consistent negative effect sizes (r=-0.6 to -0.8) across all token lengths demonstrate that ResNet50 alone, without the intermediate BLSTM processing, can significantly improve performance over baseline BERT. This suggests that BERT's output, when structured as an image, contains rich spatial or convolutional patterns that ResNet50 is adept at recognizing, leading to better feature extraction.

In summary, the consistent superior performance of models incorporating ResNet50, regardless of whether it processes only BERT features or BLSTM-transformed features, underscores the effectiveness of representing textual information as an image. ResNet50's hierarchical feature learning capabilities appear to be highly advantageous for extracting more nuanced and robust representations from these reshaped textual inputs, thereby leading to improved performance compared to a standalone BERT model. This indicates that the added complexity of these architectures is indeed beneficial, successfully leading to the extraction of additional useful information beyond what BERT already captures effectively, by leveraging the distinct strengths of convolutional processing on structured feature maps.

BERT-ResNet50 consistently outperformed BERT across all token lengths, with strong effect sizes, particularly at 64, 256, and 384. This suggests that the integration of ResNet50 alone provides a consistent and significant performance boost over baseline BERT.

The overarching finding across BERT vs all the extended model comparisons is that extending the fine-tuned BERT model with additional architectural components (BLSTM and ResNet50 combinations) consistently leads to an improvement in performance. In almost all cases where a statistically significant difference was found, the effect size was negative and often large, indicating that the extended model performance is superior to the baseline BERT for the task at hand. This suggests that the added complexity of these architectures is indeed beneficial, particularly through their capacity to process the Textual Image formed by reshaping the features of the preceding architectures.

The performance of ResNet50, given the different Contextualise Textual Images it receives, is capable of efficiently

extracting more sentiment information and is well suited to replace the simple classification output layer.

In this analysis, the consistently large negative effect sizes (often $|r| \geq 0.6$) are crucial. They indicate that the observed performance differences are not merely statistically significant (due to small p-values) but also practically meaningful. For instance, an r=-0.8 or r=-1.0 suggests a substantial and consistent improvement in performance when the additional architectural components are introduced. An effect size of r=-1.0 is particularly noteworthy, implying a perfect separation or consistent superiority of the comparator model. This is a strong indication of the practical significance of the observed improvements.

Power refers to the probability of correctly rejecting a false null hypothesis. In this study, the power values range from 0.226 to 0.723, with many values below the conventional target of 0.8. For example, a power of 0.226 means there's only a 22.6% chance of detecting a true effect if it exists.

The consistently low power values (especially when the p-value is not significant, e.g., token length 128 for BERT vs BERT-BLSTM with Power = 0.557, or token length 512 for BERT vs BERT-BLSTM with Power = 0.226) are a significant limitation. Low power implies an increased risk of Type II errors (failing to detect a true effect).

This limitation can be directly tied to the computational expense of deep learning experiments. Training and evaluating deep learning models, especially large language models like BERT and those incorporating complex architectures like ResNet, are inherently computationally expensive. Factors contributing to this high cost include:

A) Deep learning models have millions, or even billions of parameters, requiring significant memory and processing power for training. Training on large datasets, even for fine-tuning, demands extensive computational resources. While training over many epochs with specific batch sizes further increases the computational load. High-performance computing resources (GPUs or TPUs) are essential, and their availability can be prohibitive for many runs. While a limitation, it is an understandable compromise given the current landscape of deep learning research.

B) Training and evaluating complex deep learning models, especially across multiple configurations and token lengths, requires substantial computational resources (e.g., GPUs, TPUs, significant training time). Increasing the sample size (e.g., more training runs, more distinct datasets) to achieve higher power would exponentially increase these computational demands, making such studies prohibitively expensive or time-consuming for many research groups.

Therefore, while increasing the number of experiments beyond 10 per comparison would undoubtedly enhance the statistical power and reduce the likelihood of Type II errors, it would lead to an exponential increase in computational time and resources, making such extensive experimentation difficult in this research. This trade-off between statistical rigor (high power) and practical feasibility (computational cost) is a common challenge in deep learning research. Future work with access to greater computational resources would ideally aim for more experimental runs to achieve higher statistical power and validate these findings more robustly.

5.4 Summary

In summary, this chapter first explored a novel method of generating contextualised Textual Image for extracting sentiment. The BLSTM-ResNet model of Section 4 was extended to incorporate a fine-tuned BERT model to extract a contextualised embedding, which was then used to create Contextualised Textual Images and a standard ResNet network is then used to classify them. The first experimental results with the Sentiment140 dataset demonstrated that the BERT-BLSTM-ResNet network outperformed the BLSTM-ResNet. It is also noted that the BERT model on its own performed well; however, on smaller datasets, it can benefit from the use of the full BLSTM-ResNet model. The BLSTM-ResNet reaps benefit from BERT for smaller dataset, in short when the training data is limited.

In addition, the chapter investigated the token length impact on the BERT-BLSTM-ResNet architectures using the IMDB dataset along with various combinatory BLSTM and ResNet configurations extending the BERT generate contextualised word embeddings. The BERT-BLSTM-ResNet outperformed BERT on most token lengths, so did BERT-BLSTM||ResNet and BERT-ResNet, with highest average accuracy compared to other models. The BERT-ResNet model also emerged as good alternative with potential for faster training by circumventing the BLSTM block and directly utilising the contextualised Textual Images generated using BERT but has more model parameters than BERT-BLTM-ResNet model due to the large input image size generated from the BERT output.

While adding complexity to BERT generally increases both parameter count and FLOPs, the integration of ResNet, despite its higher FLOPs, offers significant parallelization benefits that can lead to faster training compared to purely sequential processing, a crucial factor in the iterative development and deployment of deep learning models.

Further, the pre-trained computer vision models, as exemplified by BERT, can be leveraged as well, instead of training the image classifier (as needs to be done by ResNet) from scratch. Additional methods to further enrich the contextualised word embeddings can be explored along with methods to reduce the limitations of the BERT-ResNet model.

Chapter 6

Using a pre-trained image classifier for Textual Image

6.1 Introduction

The literature motivating the use of pre-trained vision backbones, fine-tuning and computationally efficient Textual Image classification is reviewed in Section 2.6, particularly in the subsection addressing RQ3 and Chapter 6. It establishes the transfer-learning rationale for testing whether pre-trained image classifiers can be applied effectively to Textual Images.

Relationship to the Chapter 3 pipelines. This chapter applies Pipeline 4, described in Section 3.4.4, in Experiments 1–3. It retains the direct Type A contextualised Textual Image treatment of Pipeline 3, adds interpolation where required, and replaces the custom ResNet with a pre-trained CNN or Transformer vision backbone. Experiment 4 applies Pipeline 5 (Section 3.4.5), which combines the probabilities of the Textual Image classifier, BERT-SWC and LSA/LDA branches through a meta-learner. The P1, P2 and P3 notation used in the ensemble denotes these probability branches within Pipeline 5, rather than three additional pipelines.

In the previous chapter, the BERT-ResNet framework was introduced, wherein textual embeddings produced by BERT are transformed into images and classified using a custom ResNet model trained from scratch. While this approach proved effective, it inherits the computational cost of training a bespoke visual backbone. Addressing this limitation in this chapter, Experiment 1 investigates whether pretrained image classifiers can serve as a substitute for the custom ResNet component, thereby reducing training overhead while preserving or improving performance.

Building on this, **Experiment 2** extends the investigation beyond ResNet to include a diverse set of pretrained image classifiers, encompassing both convolutional architectures (e.g., DenseNet, EfficientNet) and transformer-based vision models (e.g., Vision Transformer, Swin Transformer). These models, however, introduce a critical constraint: most are pretrained on natural image datasets such as ImageNet, where the input resolution is substantially lower than that of the Textual Images produced by BERT. As a result, the generated Textual Images generated by the BERT model must be scaled down using interpolation to the required size, a process that risks the loss of fine-grained textual information. To address this, Experiment 2 also examines the impact of different interpolation strategies on classification performance.

Experiment 3 further explores the impact of pretrained visual backbone model scaling on Textual Image classification performance by systematically evaluating pretrained image classifiers of varying sizes under different interpolation methods in a fixed three-epoch training setup. This investigation aims to determine whether different vision models, despite their varying computational demands, provide measurable advantages in Textual Image classification for sentiment analysis in a low computation resource setting.

Finally, an additional observation is that Textual Image classifiers produce prediction patterns distinct from those generated by conventional text-based sentiment analysis methods. This divergence suggests potential complementarity between the two approaches. To examine this hypothesis and motivated by insights from the literature review presented in Section 2.4, we investigated various ensemble strategies that integrate Textual Image classifiers with established text-based sentiment analysis techniques. **Experiment 4** evaluates these ensemble methods with the objective of enhancing overall model performance in sentiment extraction.

The following sections in this chapter will detail the use of pre-trained image classifier for Textual Image classification; Traditional sentiment analysis techniques complementing a Textual Image classifier; Experimental setup;

Experiments and results; and offer insights into the effectiveness of the pre-trained image Classifier instead of custom ResNet model for sentiment analysis.

6.2 Fine-tuning a pre-trained image classifier on Textual Images



Figure 6.1: End to End pre-trained model for Textual Image classifier.

Figure 6.1 illustrates the architecture for the Textual Image Classifier, which utilizes a pre-trained image classification model for sentiment extraction. This pipeline functions similarly to the BERT-ResNet model introduced in Section 5, with the key difference being the replacement of the custom ResNet component with a publicly available pre-trained image classifier whose weights are leveraged. Like in Section 5.2.1, the BERT model is first fine-tuned on the dataset before it is used to generate the contextualized Textual Images. The remainder of the classification pipeline remains identical. A necessary preprocessing step involves image interpolation to resize the grayscale Textual Images produced by reshaping the BERT output to match the specific, fixed input size required by the pre-trained image model (Section 3.8.10). This pipeline will be utilized in all the experiments in this chapter.

6.3 Traditional and contemporary sentiment analysis techniques complementing each other

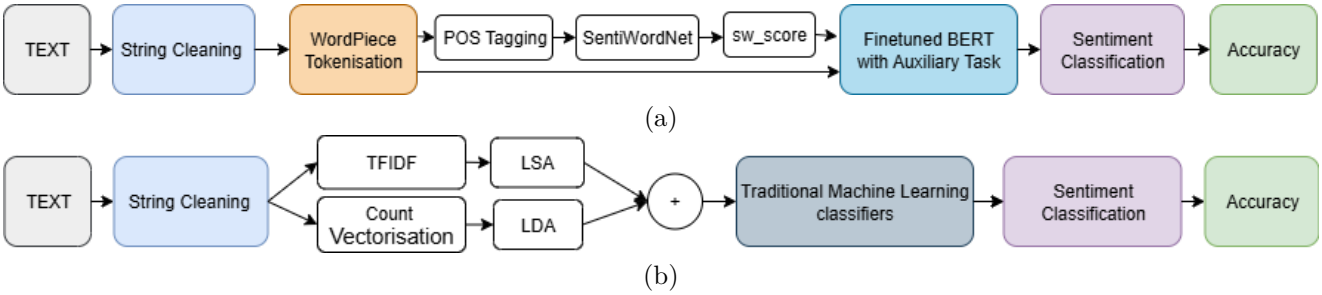


Figure 6.2: Pipelines of existing sentiment analysis. (a) employs transformer-based techniques adopted in contemporary research using lexical resources, while (b) employs Bag-of-Words techniques.

Figure 6.2 presents two distinct pipelines for conventional sentiment analysis methods: (a) a contemporary approach employing Transformer-based architectures (refer to Section 2.4), and (b) a traditional approach utilizing Bag-of-Words (BOW) techniques for sentiment extraction. The predictive outcomes (probabilities) generated by these established text-based methods will be used as a comparative model against the performance of the Textual Image Classification approach that employs pre-trained image classifiers. The text-preprocessing steps are the same as those used by the Textual Image classifier. The pipelines differ in their word embeddings and modelling, and pipeline (b) includes an additional auxiliary task. Further details are given in the description of Experiment 4. Furthermore, the individual classification probabilities from these diverse approaches will be integrated using meta-learners through an ensemble methodology. This demonstrates the potential for combining the complementary strengths of Textual Image classification with existing text-based models to achieve enhanced overall performance in complex sentiment analysis tasks.

6.4 Experimental Setup, Experiments and Results

For all the experiments conducted, the standard fine-tuned $\text{BERT}_{\text{base}}$ (referred to as BERT in this chapter) model serves as the performance baseline. Specifically, within this chapter, any reference to the BERT-ResNet model, unless explicitly stated otherwise, denotes an architecture using a publicly available pre-trained ResNet50 backbone that has been fine-tuned on contextualized Textual Images.

Experiment 1 evaluates whether publicly available, pre-trained image classifiers can replace the custom visual backbone used in the BERT-ResNet model explored in previous chapter. It compares BERT-ResNet (convolutional-

based) and BERT-ViT (Transformer-based) models, fine-tuned on Textual Images generated from a single BERT encoder, across a range of Sentiment140 dataset sizes, with fine-tuned BERT serving as the text-only baseline.

The training parameters and hyperparameters follow the setup described in Section 5.3.1. The additional processing begins after BERT generates greyscale contextualised Textual Images of size $512 \times 768 \times 1$. These images are resized using Bicubic interpolation and converted to three-channel images of size $224 \times 224 \times 3$, which form the input to the pre-trained ResNet and ViT models. These models were selected as representative CNN and Transformer vision architectures with publicly available ImageNet pre-trained weights. Early stopping used a patience of five epochs for ResNet and ten epochs for ViT; the model with the best validation performance was then evaluated on the test set. The maximum number of training epochs was reduced as dataset size increased, reflecting the available hardware and runtime limits (Section 3.2.1).

For Experiment 2, the experimental setup is the same as for Experiment 1, with the only difference being that a single BERT model is used. The BERT model is fine-tuned once on the sentiment analysis dataset and then used as a Textual Image generator to fine-tune all the pre-trained image classifiers on the contextualised Textual Images. Three interpolation techniques (nearest, Bilinear and Bicubic) were compared to produce the required input shape. All the models chosen in this experiment are used because all of them are pre-trained on same Image Dataset and the trained model weights are available.

For Experiment 3, the experimental setup is the same as for Experiment 2, but nearest-neighbour interpolation is used based on the findings of Experiment 2. Also, only EfficientNet models are chosen because they have all the different sizes of pre-trained models available on ImageNet, so it will be appropriate to compare the model size impact of pre-trained image classifiers trained on the same image dataset. The models were fine-tuned for a fixed three epochs to mimic the hardware bottleneck.

Experiment 4 explores and identifies the optimal feature components for both LSA and LDA. These are combined into a single set, which is subsequently used to train the five different machine learning algorithms (logistic regression (LR), support vector machine (SVM), multi-layer perceptron, Naïve Bayes (NB) and XGBoost). A parameter grid search is performed to choose the best model parameters, and the results identify the best-performing model in prior tests (refer to Appendix Section 12.5 for details). Separately, for the word sentiment classifier, a pre-trained BERT model is employed. This model is fine-tuned on our text data using a multi-task approach that includes an auxiliary task of predicting word-level sentiments. These word-level sentiment labels were generated using SentiWordNet, a process that results in two distinct losses (one for the sentiment analysis task and one for the auxiliary regression task of predicting the word sentiment score). Both losses are combined using a weighting parameter, alpha (α) and the optimal alpha value, determined experimentally, is used for the final model (refer Appendix Section 12.6 for details).

6.4.1 Experiment 1

The results are reported in Table 6.1 and Figure 6.4. They compare the performance and convergence behaviour of the evaluated models across the different dataset sizes.

Table 6.1: Comparing a pretrained ResNet and ViT as a viable substitute for the custom visual backbone for Textual Image classification using average accuracy, and number of Epochs needed for best performance, all averaged over 5 runs. Note: K stands for a thousand and M for a million, AVG for average, and STD for standard deviation. The best Average performance for each data set size is shown in bold.

Dataset Size	**BERT**		**BERT-ResNet**		**BERT-ViT**		**Max Epochs**
	Accuracy (AVG ± STD)	**Epochs** (AVG ± STD)	**Accuracy** (AVG ± STD)	**Epochs** (AVG ± STD)	**Accuracy** (AVG ± STD)	**Epochs** (AVG ± STD)	
625	80.00 ± 4.97	4.6 ± 0.89	80.00 ± 3.98	10.4 ± 6.84	**84.47 ± 1.73**	14.0 ± 2.12	50
1250	74.08 ± 1.45	5.6 ± 2.70	**74.40 ± 2.47**	11.4 ± 7.40	72.96 ± 1.31	12.2 ± 1.79	50
2.5K	77.20 ± 0.80	5.2 ± 1.92	76.64 ± 0.73	6.4 ± 4.93	**78.32 ± 1.04**	15.2 ± 2.68	40
5K	**80.68 ± 0.48**	1.8 ± 1.79	80.48 ± 0.52	6.4 ± 5.32	79.72 ± 0.69	11.2 ± 1.48	40
10K	80.96 ± 0.86	9.6 ± 4.28	81.80 ± 0.69	4.8 ± 3.11	**81.86 ± 0.55**	13.4 ± 2.30	30
20K	80.40 ± 0.88	5.4 ± 2.97	**81.68 ± 0.28**	4.8 ± 3.35	81.29 ± 0.47	11.0 ± 3.74	30

Continued on next page

Dataset Size	BERT		BERT-ResNet		BERT-ViT		Max Epochs
	Accuracy (AVG ± STD)	**Epochs** (AVG ± STD)	**Accuracy** (AVG ± STD)	**Epochs** (AVG ± STD)	**Accuracy** (AVG ± STD)	**Epochs** (AVG ± STD)	
40K	82.99 ± 0.40	1.4 ± 0.55	83.28 ± 0.32	5.6 ± 3.78	**83.41 ± 0.22**	11.6 ± 1.82	25
80K	83.00 ± 0.15	1.0 ± 0.00	**83.36 ± 0.12**	6.2 ± 2.28	83.22 ± 0.13	12.4 ± 8.62	25
160K	**84.13 ± 0.15**	2.0 ± 0.00	84.10 ± 0.05	4.0 ± 1.22	84.09 ± 0.06	10.6 ± 2.07	20
320K	83.98 ± 0.12	2.2 ± 0.45	**84.43 ± 0.08**	4.4 ± 1.52	84.29 ± 0.05	7.2 ± 2.68	20
640K	84.15 ± 0.10	2.8 ± 0.45	**84.52 ± 0.07**	2.4 ± 0.89	83.99 ± 0.60	11.0 ± 3.67	15
1.28M	85.33 ± 0.08	2.0 ± 0.00	**85.56 ± 0.04**	2.2 ± 0.45	82.03 ± 5.84	7.8 ± 2.86	15
1.6M	85.39 ± 0.07	1.6 ± 0.55	**85.68 ± 0.05**	1.6 ± 0.55	83.13 ± 3.78	6.8 ± 2.05	10

The analysis of performance across data scales reveals distinct behavioural patterns across the data range. While the smallest dataset sizes (625 and 1,250) are excluded from the main trend analysis due to disproportionately high run-to-run standard deviations caused by the small test split, the rest can be analysed as follows. The trend for dataset sizes at the low end, ranging from 2,500 to 20,000, shows a clear movement in performance across all models. BERT improved from its lowest average accuracy of 77.20%±0.80% (at 2.5K) to a peak of 80.96%±0.86% (at 10K), before slightly receding to 80.40% at 20K. BERT-ResNet showed a steadier ascent, moving from its starting point of 76.64%±0.73% (at 2.5K) to consistently outperform the baseline at 10000 and 20000, achieving a high of 81.80%±0.69% at 1000 and settling at 81.68% at 20000. BERT-ViT was the most volatile in this range, starting strong at 78.32%±1.04% (at 2.5K). It reached its peak for this segment at 81.86%±0.55% (at 10K) but then showed a noticeable drop in performance at 20k with 81.29%±0.47%, indicating early instability.

For the intermediate data size range (40K to 160K), BERT-ResNet consistently achieved a modest but reliable accuracy advantage of 0.22 to 0.36 percentage points over the BERT baseline at the 40k and 80k sizes (e.g., 83.36% vs. 83.00% at 80k). Furthermore, it demonstrates a significantly faster convergence than BERT-ViT, peaking at a mean of 4.0 to 6.2 epochs. Conversely, BERT-ViT, despite achieving the highest raw accuracy at 40k (83.41%), consistently required noticeably more epochs (e.g., 11.6 to 12.4 mean epochs at 40k and 80k), suggesting a higher degree of optimisation sensitivity and greater computational expense to reach the best validation accuracy during training compared to the ResNet variant.

At the largest data range (320K to 16M), where performance reaches an asymptote, BERT-ResNet maintains a small but reliable advantage over the BERT baseline, reaching its highest overall average accuracy of 85.68%±0.05% at 1.6M samples, slightly edging out BERT's peak of 85.39%±0.07%. Conversely, BERT-ViT's performance occasionally degrades at these largest sizes, suggesting that the CNN-based ResNet backbones are better for Textual Image classification.

The BERT-ResNet model consistently outperforms the performance the other models at the High data ranges and BERT-ViT performance degrades on average, this can be due to low total epochs set for training at high data ranges due to the hardware restraints, limiting ViT to achieve the best performance on the Validation set during training.

This overall strong and stable performance of the CNN-based BERT-ResNet model, coupled with its low standard deviations, validates the core hypothesis: the deep, pre-trained convolutional features transfer effectively to the task of classifying synthetically generated Textual Images, providing an accurate, drop-in replacement for a custom CNN with a substantially lower optimisation burden. However, this accuracy gain introduces an operational trade-off related to convergence dynamics. While BERT typically reaches its optimum in 1-3 epochs at large scales, BERT-ResNet generally requires 2-6 epochs, and BERT-ViT is the most demanding, requiring 7-12 epochs. Ultimately, the slight edge in accuracy and stability provided by BERT-ResNet makes it the most favourable architecture for the task.

Figure 6.4 illustrates the best accuracy achieved by BERT, BERT-ResNet, and BERT-ViT as the dataset size increases. A dominant trend is the strong, monotonic increase in best accuracy for all models across the entire range, confirming that data availability is the primary factor driving peak performance, moving from a low of ≈77.8% at 2,500 samples to a high of ≈85.6% at 1,600,000 samples. While the BERT-ViT model starts with the highest accuracy at the smallest size (2,500 samples, ≈79.7%), BERT-ResNet quickly emerges as the overall best performer from 5,000 samples onwards, consistently matching or slightly surpassing the other two models across the mid-to-large range. At the largest dataset size (1,600,000), all models converge, but BERT-ResNet still achieves the marginal highest peak accuracy (≈85.6%), confirming that the deep, pre-trained convolutional features offer the most robust and consistent performance advantage for Textual Image classification across nearly all data scales.

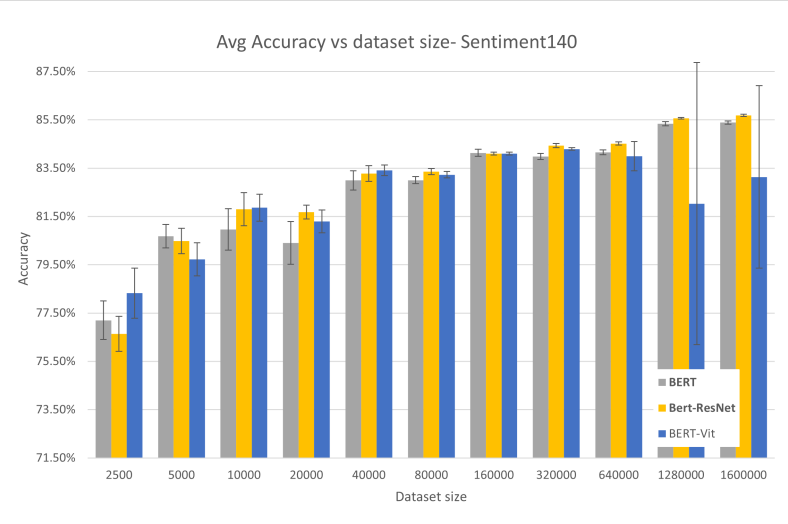


Figure 6.3: Average Accuracy with std vs Data size for BERT, BERT-ResNet and BERT-ViT

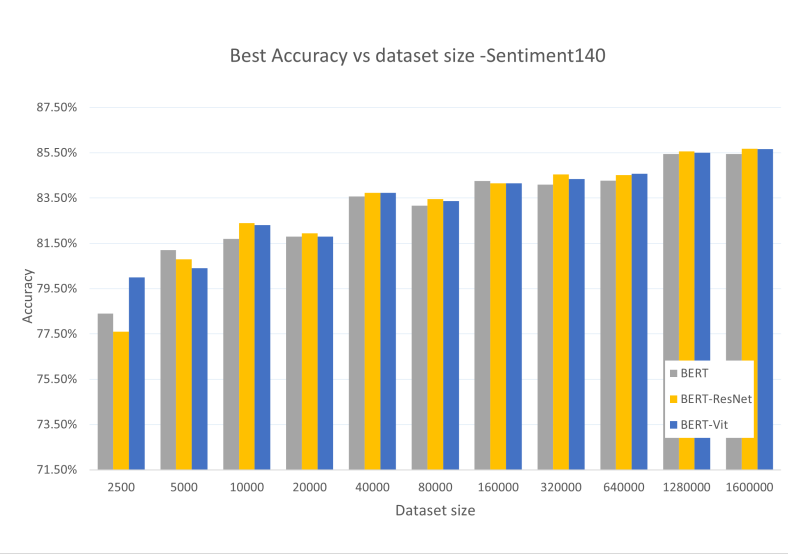


Figure 6.4: Best Accuracy vs Dataset size for BERT, BERT-ResNet and BERT-ViT

The close model performance observed in the top performing models in Figure 6.4 inspired exploration of the heterogeneity in model predictions. Figure 6.5 presents Venn diagrams illustrating the overlap in correctly predicted test instances for all three models across the data size ranges. The colors in the Venn diagram are used to visually map the regions where different models agree or disagree on their predictions, helping to quantify their complementary strengths. The Lavender (Central Area) represents cases where all three models have made identical predictions. The three arcs (Pink, Green, and Blue) identify regions of unique contribution, where only one specific model (BERT, BERT-ResNet, or BERT-ViT, respectively) made a particular correct prediction that the others missed. Finally, the three regions Orange (Top-Central Arc), Purple (Bottom-Left Arc), and Blue (Right-Central Arc) represent areas of pairwise agreement, showing the cases where only two of the three models have the same predictions.

The central region of complete consensus among all three models remains exceptionally robust across the entire spectrum of dataset sizes, with the number of jointly agreed-upon samples ranging significantly from 238 to 143,414. Critically, the unique, non-overlapping regions—where only one model correctly classifies a sample are non-zero for all models across every tested data subset. The number of these unique contributions demonstrates distinct growth patterns: it remains in a relatively smaller range for the initial dataset sizes (2.5K to 20K), then transitions to a linear increase across the intermediate data ranges (40K to 160K), and finally exhibits a nonlinear acceleration in the large data regions (320K to 1.6M) as the dataset size grows. This effect is pronounced at the 1.6 million sample size, where BERT has the largest unique contribution with 14,309 samples, while the visual models contribute 1,188 samples uniquely by BERT-ViT and 1,089 by BERT-ResNet.

At the maximum dataset size of 1.6 million samples (test samples 160,000), there are 14,309 samples uniquely classified by BERT, demonstrating instances where the text-only model succeeds where the visual models fail. In contrast, the visual classification models show highly specific, smaller unique contributions: 1,188 samples are uniquely classified by BERT-ViT, and 1,089 samples are uniquely classified by BERT-ResNet.

The presence of these unique correct predictions strongly indicates that each pre-trained image classifier trained on Textual Images closely mimics the predictions of the BERT model. And show meaningful deviation in higher data range so at higher data ranges it ought to be beneficial to combine the individual results of BERT and the Textual Image classifiers.

6.4.2 Experiment 2

This experiment will rigorously evaluate a diverse set of contemporary pre-trained vision models as effective replacements for a custom visual backbone in Textual Image classification, while concurrently analysing the impact of image interpolation methods (Nearest, Bilinear, and Bicubic) necessary for resizing the high-resolution BERT-generated Textual Images. The investigation includes traditional CNNs (ResNet50, DenseNet121, EfficientNet_b4,5) and modern Vision Transformers (ViT, Swin Transformer). The IMDB data set is chosen for this comparison as it has a separate test set and will be excellent to compare the different model's performance on longer texts.

The average model performance with standard deviation across five runs for each interpolation method is presented in Table 6.2 and Figure 6.6 (a) shows the trends. Figure 6.6 (b) shows the best performing interpolation for each model. The individual run information is provided in Appendix Section 12.2 for details.

Table 6.2: Various pretrained vision backbone methods, showing the average accuracy with standard deviation across different interpolations for Textual Images along with the Average number of Epochs, with standard deviation, needed for reaching the best performance. In Each case the average is over 5 runs. Again, the best result is highlighted in bold.

Model \ **Interpolation**		**Bicubic**	**Bilinear**	**Nearest**
BERT-ResNet50	Accuracy	93.16 ± 0.07	93.08 ± 0.08	93.11 ± 0.07
	Epochs	8.80 ± 6.46	6.60 ± 3.78	6.80 ± 2.77
BERT-SwinFormer	Accuracy	50.00 ± 0.00	50.00 ± 0.00	50.00 ± 0.00
	Epochs	1 ± 0.00	1 ± 0.00	1 ± 0.00
BERT-ViT	Accuracy	93.10 ± 0.10	93.15 ± 0.09	93.18 ± 0.08
	Epochs	11.00 ± 4.85	12.60 ± 9.69	6.80 ± 3.27
BERT-DenseNet-121	Accuracy	93.18 ± 0.17	93.12 ± 0.13	93.07 ± 0.10

Continued on next page

Model \ Interpolation		Bicubic	Bilinear	Nearest
	Epochs	7.40 ± 5.37	10.00 ± 8.40	6.00 ± 4.42
BERT-EfficientNet-B4	Accuracy	93.17 ± 0.07	93.02 ± 0.10	93.14 ± 0.07
	Epochs	2.40 ± 1.67	7.80 ± 11.39	6.00 ± 2.83
BERT-EfficientNet-B5	Accuracy	**93.22 ± 0.11**	**93.22 ± 0.09**	**93.23 ± 0.10**
	Epochs	4.20 ± 2.59	3.40 ± 1.67	3.60 ± 3.71

All the pre-trained visual models, apart from BERT-SwinFormer performed higher than 93.05% accuracy of the BERT baseline. The EfficientNet-B5 model emerged as the strongest, achieving the overall best mean accuracy of 93.23%±0.10% using Nearest neighbour interpolation. The slight but consistent advantage of the Nearest method (as seen in Figure 6.6, (a)) implies that preserving the discrete, sharp boundaries between the contextualized embedding vectors might be important, indicating that the raw, non-smoothed pixel-wise information might be more valuable than smoothed spatial representations (the smoothing effect inherent in Bilinear and Bicubic). Given the tiny accuracy spread, the practical implication is to adopt Nearest interpolation for its speed, thereby minimising the computational overhead of the interpolation step.

While all CNN models and BERT-ViT achieved highly competitive performance, around 93.1% to 93.2% average accuracy, the BERT-Efficientnet_B5 model secured the highest individual peak performance of 93.37% (Figure 6.6, (b)), marginally surpassing BERT-DenseNet-121 (93.33%). This result suggests that the efficient scaling and specialised architecture of EfficientNet are exceptionally well-suited for processing these synthetically generated Textual Images. Conversely, the BERT-SwinFormer model consistently and completely failed to converge, retaining a 50.00% accuracy across all interpolation methods (Table 6.2). This failure points to a fundamental incompatibility between the Swin architecture's patch-based hierarchical feature extraction and the non-natural, spatial patterns of the Textual Images, indicating a potential mismatch in inductive biases that requires significant specialized engineering to resolve before drawing any conclusions about the architecture itself.

In terms of training efficiency, the models show notable differences in convergence (Table 6.2). While the BERT baseline converges quickly, the visual backbones require more epochs for stabilisation, particularly BERT-ViT, which generally requires the most epochs (e.g., 10.60 to 12.60 mean epochs). The BERT-ResNet and BERT-EfficientNet variants demonstrate faster convergence than ViT, with BERT-Efficientnet_B5 requiring the fewest epochs among the visual backbones (mean epochs between 3.40 and 4.20).

Given the proximity of the average model performances, statistical testing was essential to determine if these small differences were significant. So, the Kruskal-Walli's H test, a non-parametric equivalent of the one-way ANOVA, was employed. This test is particularly suitable as it does not assume a normal data distribution, which is appropriate for the small sample sizes in this experiment. For all tests, the null hypothesis (H_0) proposed that no difference exists between the group means, against an alternative hypothesis (H_1) that at least one group differs. A significance level of ($p < 0.05$) was established as the threshold for rejecting the null hypothesis.

Applying the Kruskal-Wallis test to compare the five visual backbones (ResNet50, ViT, DenseNet121, EfficientNetB4, and EfficientNetB5) revealed no statistically significant evidence to suggest the superiority of any single model for Textual Image classification. Similarly, a comparison of the three interpolation techniques using the same test showed no significant performance difference, indicating that the choice between Nearest, Bilinear, and Bicubic methods did not have a discernible impact on model accuracy (refer Appendix Section 12.2.1 for details).

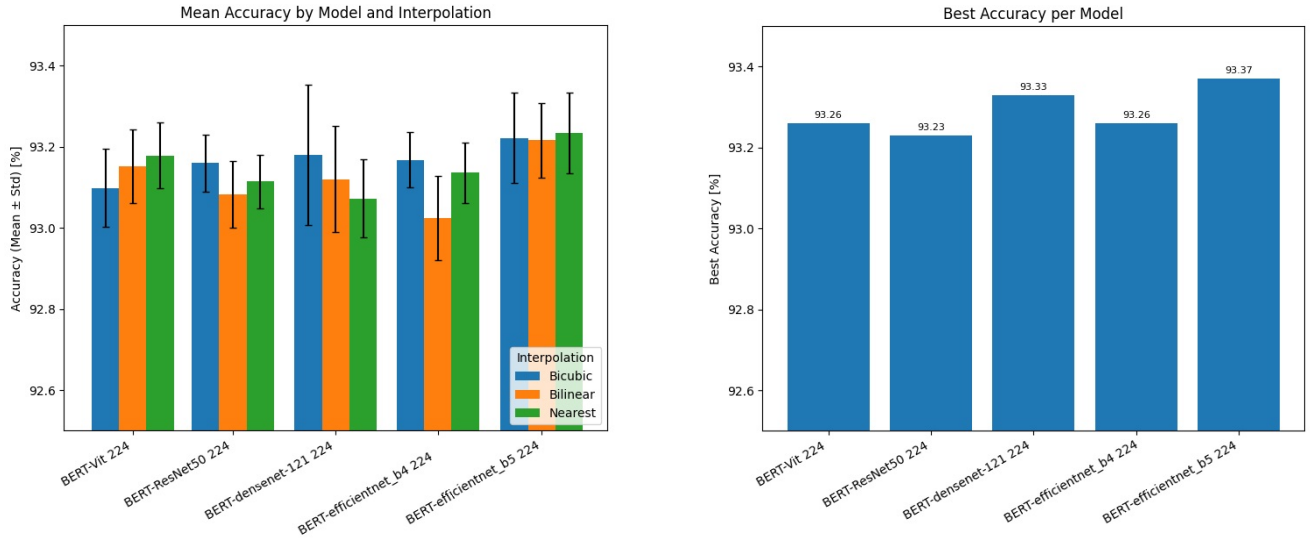
To further validate the finding for the interpolation methods, a one-way ANOVA was also conducted, leveraging the larger aggregated sample size (n=25 per group). The prerequisites for this parametric test, namely, normality (confirmed by the Shapiro-Wilk test) and homogeneity of variances (confirmed by Levene's test), were satisfied. The ANOVA results were consistent with the Kruskal-Wallis findings, confirming the absence of a statistically significant difference among the interpolation techniques (refer Appendix Section 12.3.1.2 for details). All the statistical approaches robustly support the conclusion that the observed variations in the group performance are not significant and could be attributed to random chance.

6.4.3 Experiment 3

Experiment 3 examines the critical trade-off between model size and data availability. Using the insights from Experiment 2 (specifically, adopting Nearest interpolation), this experiment focuses exclusively on the EfficientNet architecture, evaluating its full range of scaled models (Efficientnet_b0 to b7) across various dataset sizes (from 5% to 100% of the IMDB training data with the test set being the original IMDB test set) . This systematic



Figure 6.5: Venn diagram showing the overlapping of the test set predictions for BERT, BERT-ResNet (as ResNet), BERT-VIT (as ViT)



(a) (b)

Figure 6.6: (a) Mean Accuracy with standard deviation on various interpolations for different visual backbone classifying Textual Images. (b) Different best performing Model across interpolation for classifying Textual Images.

scaling study aims to explore the impact of the model complexity (from the smaller b0 to the larger b7) on model performance in a fixed Epoch setting

The fine-tuned BERT model use is the same as in Experiment 2. Experiment 3 has two main aims: first to understand how the scaling the learning parameter of the pre-trained image classification model impacts on the performance of the Textual Image classifiers; second to understand the size of training data impact on the performance of Textual Image classifiers using pre-trained image classifiers. The number of epochs was restricted to 3 (model convergence is not a concern here). The very low Epoch number is used particularly to test the model in a low computation setting as availability of hardware is always a bottleneck in deep learning.

Table 6.3: Average Accuracy with standard deviation for BERT-EfficientNet Models (bold values represent best individual model performance at different data percentage and blue indicates the best performance across models for the that data percentage)

Data / Model	**0.05%**	**0.1%**	**0.2%**	**0.4%**	**0.6%**	**0.8%**	**1.0%**
Efficientnet_b0	91.57% ± 0.62%	92.58% ± 0.21%	92.85% ± 0.24%	93.02% ± 0.13%	**93.21% ± 0.18%**	93.14% ± 0.11%	93.18% ± 0.11%
Efficientnet_b1	91.39% ± 0.68%	92.63% ± 0.26%	92.75% ± 0.20%	92.72% ± 0.27%	93.24% ± 0.27%	**93.38% ± 0.07%**	93.03% ± 0.25%
Efficientnet_b2	92.08% ± 0.15%	92.55% ± 0.09%	92.76% ± 0.38%	92.77% ± 0.16%	93.17% ± 0.28%	**93.28% ± 0.11%**	93.09% ± 0.11%
Efficientnet_b3	91.54% ± 0.52%	92.70% ± 0.18%	92.86% ± 0.14%	92.78% ± 0.26%	93.22% ± 0.15%	**93.32% ± 0.17%**	93.06% ± 0.15%
Efficientnet_b4	91.78% ± 0.36%	92.68% ± 0.10%	92.71% ± 0.29%	92.95% ± 0.13%	93.13% ± 0.13%	**93.19% ± 0.17%**	93.06% ± 0.13%
Efficientnet_b5	92.00% ± 0.59%	92.35% ± 0.27%	92.81% ± 0.20%	92.95% ± 0.11%	93.02% ± 0.15%	**93.40% ± 0.08%**	93.12% ± 0.04%
Efficientnet_b6	92.10% ± 0.56%	92.59% ± 0.15%	92.95% ± 0.12%	93.07% ± 0.13%	93.30% ± 0.04%	**93.34% ± 0.12%**	93.14% ± 0.08%
Efficientnet_b7	91.71% ± 0.32%	92.70% ± 0.83%	92.75% ± 0.21%	93.02% ± 0.04%	93.20% ± 0.17%	93.40% ± 0.16%	93.12% ± 0.14%

Table 6.3 provides the average accuracy across five runs with standard deviation for each EfficientNet variant for increasing dataset percentages. The data of average accuracy is presented as a heat map in Figure 6.7. Table 6.4 presents the giga FLOPS an Efficient model takes in processing a 224x224x3 image and the corresponding model number of parameter in millions. FLOPs provide a hardware-agnostic metric to assess computational cost.

The comprehensive analysis across varying data percentages and model capacities reveals a complex interplay between model complexity, data availability, and efficiency in a low-epoch fine-tuning setting. Performance for

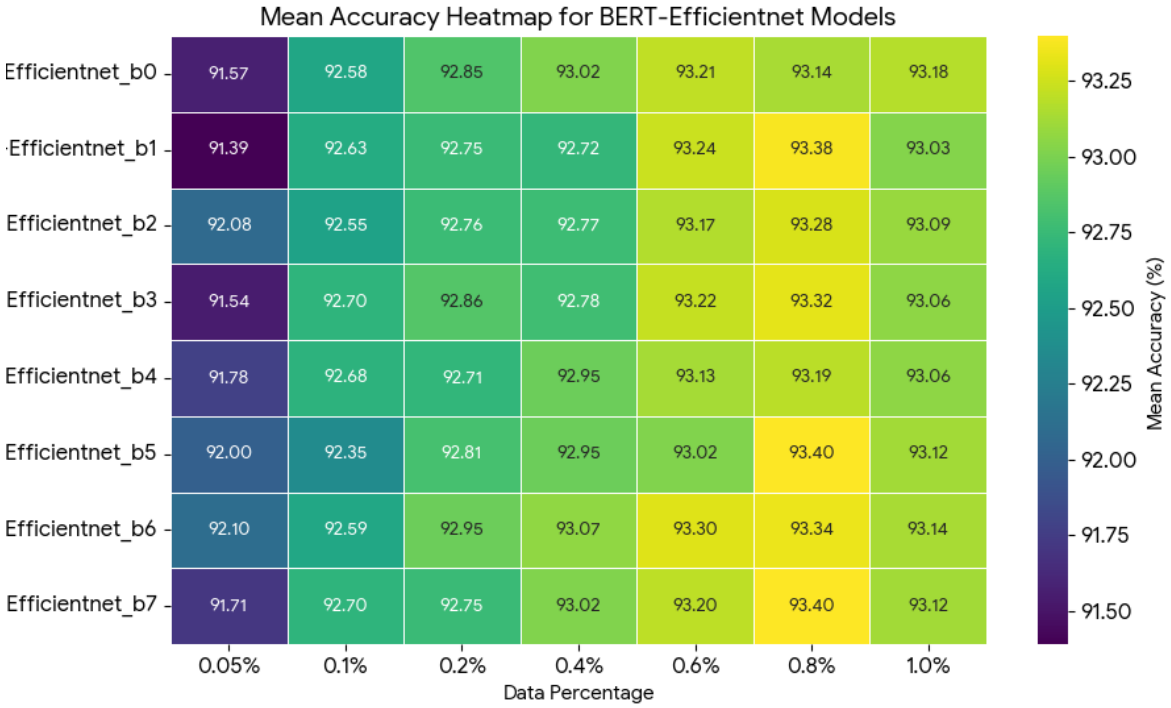
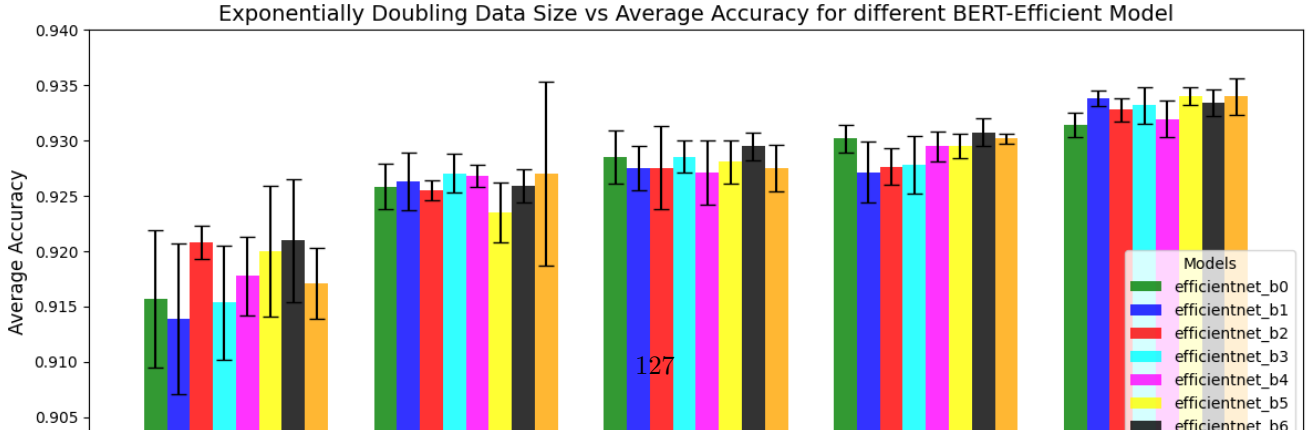


Figure 6.7: Average Accuracy Heatmap for the BERT-Efficient Models

Table 6.4: EfficientNet model number of parameters and Flops needed

Model	**Parameters (Millions)**	**FLOPs (Giga) to process 224x224x3 image**
Efficientnet_b0	4.01	0.41
Efficientnet_b1	6.51	0.6
Efficientnet_b2	7.7	0.69
Efficientnet_b3	10.7	1.01
Efficientnet_b4	17.55	1.56
Efficientnet_b5	28.34	2.44
Efficientnet_b6	40.74	3.47
Efficientnet_b7	63.79	5.32

Crucially, the results demonstrate diminishing returns from model scaling as the data percentage increases at the highest dataset sizes. At the highest dataset sizes (80% and 100%), all models, including the computationally lighter Efficientnet_b0 and Efficientnet_b1, converge to a similar peak accuracy (clustering around 93.1%-93.4%). This convergence highlights a fundamental efficiency finding given the constraints of a low-epoch fine-tuning regimen, the smaller, resource-efficient models (Efficientnet_b0-b1) provide the best performance to compute trade off, aligning with the goal of minimizing computational demands while still achieving near-peak classification performance.



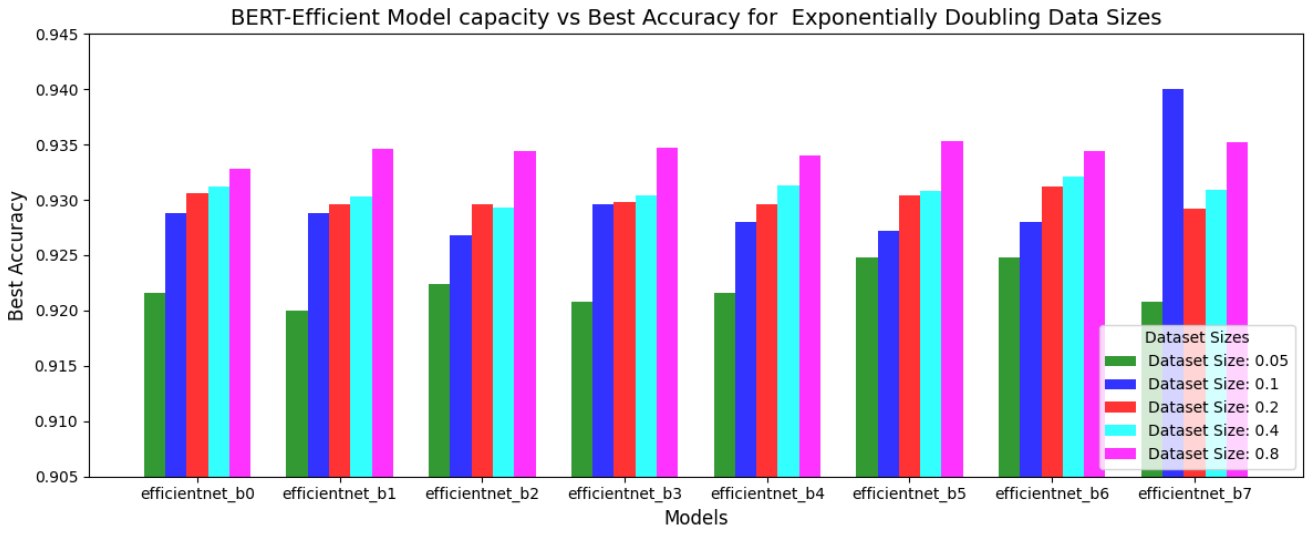


Figure 6.9: BERT-Efficient Model capacity vs Best Accuracy for Exponentially Doubling Data Sizes

6.4.4 Experiment 4

The preceding experiments demonstrated that Textual Image classifiers yield predictions closely comparable to baseline BERT models (as shown by the Venn diagrams in **Figure 6.5**) especially at low data sizes. So, to improve the Textual Image classification performance further, additional sentiment information can be added to our existing pipeline to achieve it. One method is ensemble where multiple predictions can be combined using a meta learner. The insights from the literature Section 2), discussed various traditional and contemporary approaches to sentiment extraction. Experiment 4 investigates if developing ensemble strategies to combine the probabilistic outputs of sentiment extraction methods along with Textual Image classification will yield any enhancement in overall performance. The ensemble is conducted using the combined probability outputs of three distinct models, refer to Figure 3.2.

1.P1: The Textual Image Classifier BERT-ResNet (Figure 6.1)

2.P2: Fine-tuned BERT model integrated with SentiWordNet via an auxiliary task (BERT-SWC)(Figure 6.2 (a))

3.P3: SVM trained on LSA+LDA features using BOW (Figure 6.2 (b))

The first stage involved optimising the auxiliary task model (P2) by tuning the α weighting factor, which controls the balance between the main sentiment classification loss and the word sentiment loss derived from SentiWordNet.

BERT-SWC Results To explore this in-detail, the training and test set of the IMDB data set is combined and then a 2% subset is chosen as the test set. This was done to maximise the training samples, used to determine the impact of the alpha parameter on performance, as the deep leaning models are data hungry, as is evident in findings of Experiment 1 in this chapter. The result of this is present in Table 6.5 and Figure 6.10 shows the trend.

Table 6.5: Word sentiment Classifier average accuracy with standard deviation across various values of α

Alpha (α)	**Average Accuracy**	**Std**
0.0	49.88	0.178885438
0.1	92.20	1.252996409
0.2	93.90	0.608276253
0.3	93.22	0.486826458
0.4	93.78	0.204939015
0.5	93.08	0.356370594
0.6	93.18	0.148323970
0.7	93.72	0.973139250

Continued on next page

Alpha (α)	Average Accuracy	Std
0.8	93.30	0.578791845
0.9	93.54	0.433589668
1.0	93.28	0.563027530

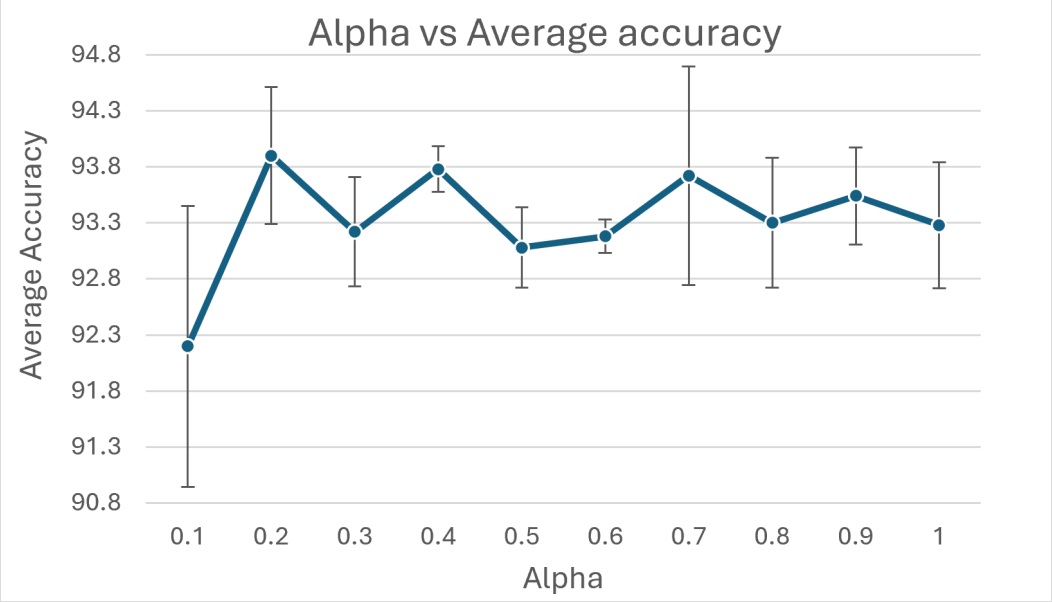


Figure 6.10: Average accuracy with standard deviation vs α for the Word sentiment Classifier

An α value of 0.4 provided the best combination of high average accuracy (93.78%) and low standard deviation (0.204%). It is important to note that setting α=0.0 resulted in performance equivalent to random chance (49.88%), confirming that the SentiWordNet loss term alone is not enough to be effective in extracting sentiment knowledge since the class labels are also required. Also, α=1.0 shows a performance drop showing that the auxiliary task does add additional information to aid in model predictions.

LSA + LDA method best parameters: To investigate the best set of parameters for a Topic modelling method to aid Textual Image classification, we investigated various aspect of Topic modelling using the IMDB dataset (refer Appendix Section 12.5 for details

1. The optimal number of components for LSA and LDA and their best combination.
2. The best classifier for LSA and LDA component for sentiment extraction.
3. Best order of text-preprocessing methods to improve sentiment extraction using LSA and LDA components.

Increasing the LSA + LDA feature count resulted in a performance gain with diminishing returns as the number of features grows for most classifiers we used. Text cleaning like removing hashtags, usernames and URLs should be the first step followed by Stemming and then Lemmatisation was found to be the best order of text-preprocessing to prepare the text data for Topic modeling to extract sentiment. An SVM performed the best on the combined LSA + LDA features, and the performance increased linearly with increasing feature count with diminishing returns (refer Figure 6.11). Notably the SVM performance on LSA only features are very comparable to the combined features in higher components count but at lower components counts the combined components for LSA and LDA had a slight advantage over LSA only components.

Testing Ensemble on the IMDB dataset: To investigate the suitability of Ensembling Textual Image classifications with other investigated sentiment analysis methods, we performed the prediction analysis. We trained all the methods on the IMDB training set, and the test set was used to evaluate each method, and their probabilities were recorded. Like Experiment 1, a Venn diagram is plotted showing the overlap and disagreement in prediction of various methods (refer Figure 6.12). From the Venn diagram, it is quite clear that the predictions of all the methods are not identical, so that their predictions can potentially be combined to improve the overall performance for sentiment analysis.

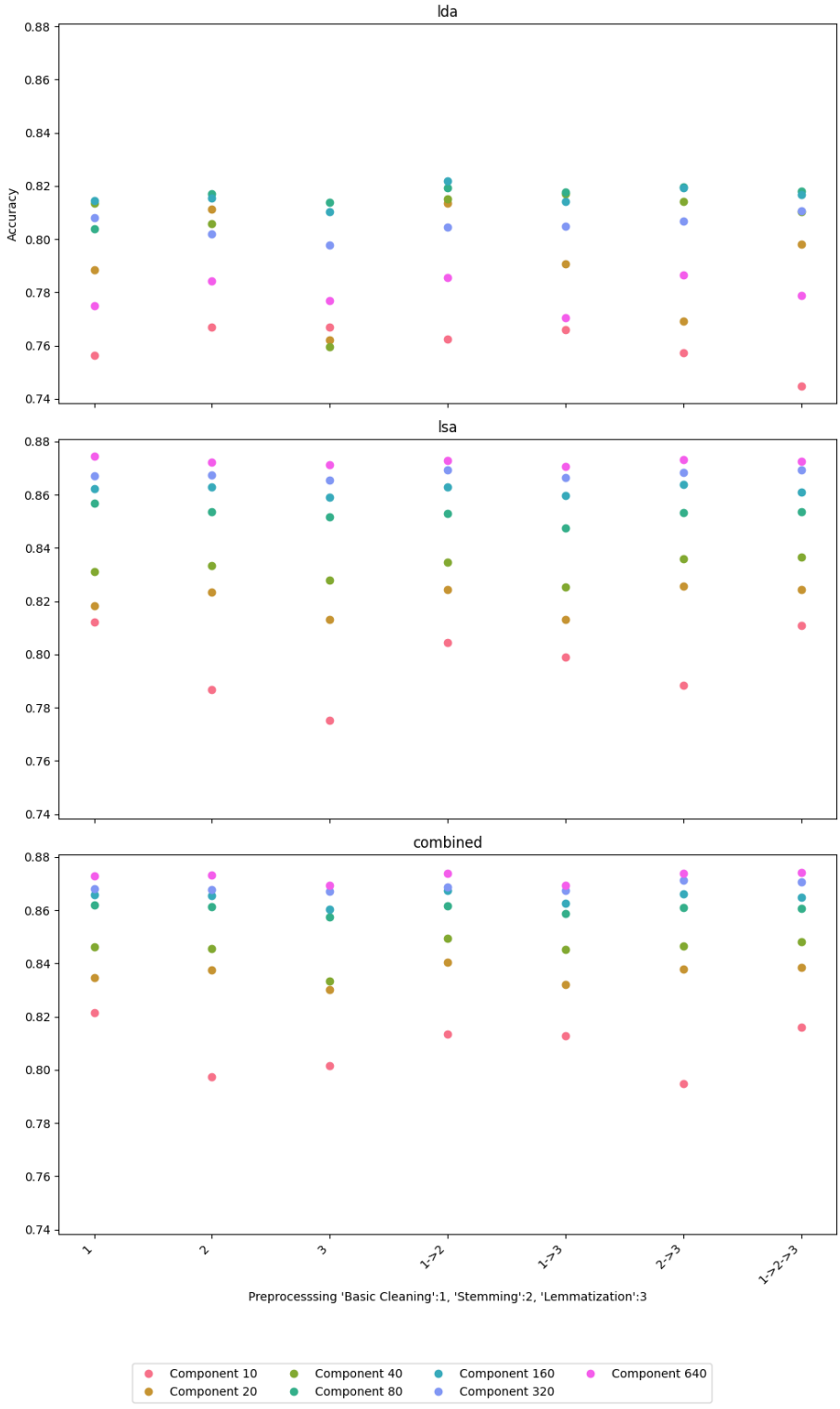


Figure 6.11: LSA and LDA performance across various text preprocessing combinations using the SVM classifier for sentiment analysis.

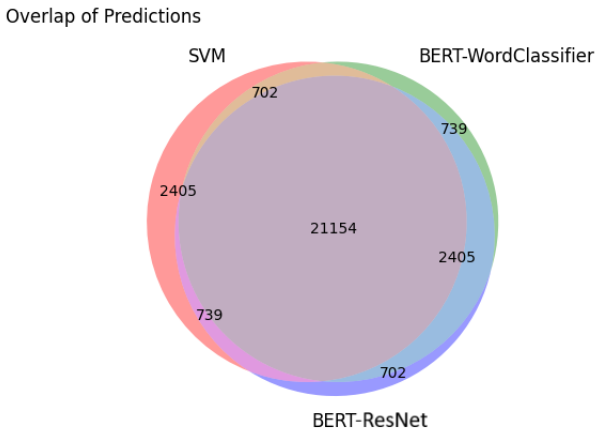


Figure 6.12: Prediction overlap analysis for various sentiment extraction methods (SVM represents combined LSA and LDA features and BERT-WordClassifier represents BERT-SWC)

The Best individual model accuracy is for BERT-ResNet at 93.15%, then BERT-SWC at 92.83% and last the SVM classifier using combined LSA and LDA components at 87.33%

Table 6.6: Ensemble performance using various meta learners.

Ensemble Method	**Accuracy**
SOFT Voting with Equal Weights (AVG)	93.34%
Weighted Average (WAVG)	93.33%
Logistic Regression	93.26%
SVM with RBF	93.16%
Bayesian Optimisation Ensemble	93.27%
XGBoost Ensemble	**94.32%**

To combine the model performance various meta classifiers are used (refer Figure 6.12). All the ensemble learner managed to combine the individual model combination to improve overall sentiment performance, but XGBoost had the best performance at 94.32% which is more than a 1% improvement in performance over the previous individual best, BERT-ResNet.

Ensemble performance on Sentiment 140: The success of Ensemble on the IMDB dataset led us to evaluate the method on Sentiment140. While the BERT-SWC is effective and showed a variation in prediction from the BERT-ResNet predictions, its fine-tuning process, using a BERT model (From Experiment 1) along with auxiliary task, is significantly more time-consuming compared to BERT-ResNet, presenting a substantial hardware availability constraint. This limitation restricted the BERT-SWC's performance evaluation to smaller data ranges ($\leq$20k) on the Sentiment140 dataset so here we could only evaluate the smaller data ranges. In addition, the Training epochs for BERT-SWC were also reduced with increase in data size (dropping from 25 to 10) due to hardware constraints. We have selected the best performing model in Experiment 1 in this test for BERT-ResNet. And, as the maximum number of tokens in the sentiment dataset after tokenisation is around 60, so 100 components for the LSA and LDA combination was selected. Table 6.7: Various Ensembling results for Sentiment140 subsets. Table 6.7 contains the individual model performance and various ensemble learners' performance.

Table 6.7: Various Ensembling results for Sentiment140 subsets. SOFT Voting with Equal Weights. WAVG is the weighted AVG. LR is logistic regression and SVM is Support vector machine with RBF kernel.

Size	Individual model accuracy			Ensembling method accuracy				
	BERT-Word Classifier	**BERT-ResNet**	**LSA-LDA (SVM)**	**AVG**	**WAVG**	**LR**	**SVM**	**XGBoost**
625	**87.30%**	77.78%	63.49%	84.13%	79.37%	80.95%	77.78%	85.13%
1,250	77.60%	76.00%	66.40%	80.00%	78.40%	80.80%	80.80%	**81.00%**
2.5K	75.20%	78.40%	66.00%	78.40%	78.40%	75.60%	76.40%	**79.20%**
5K	80.60%	81.00%	70.00%	82.20%	81.20%	81.60%	81.20%	**82.80%**
10K	**84.70%**	81.90%	70.40%	84.30%	**84.70%**	84.20%	84.00%	83.90%
20K	80.75%	82.40%	69.80%	82.15%	82.60%	82.60%	**82.80%**	**82.80%**

The BERT based model accuracy on this dataset varied widely, peaking at 87.30% for the 625-sample size but showing considerable instability as data size increased, dipping to 75.20% at 2,500 samples before rising again as this model used an already fine-tuned BERT model which impacted the final performance. The impact of reduced epochs for 20K could be the reason for a significant drop in its performance.

The LSA-LDA (SVM) model consistently has the weakest standalone component, with its accuracy remaining the lowest across all dataset sizes, peaking at only 70.40% at 10K samples. Conversely, the advanced BERT based models establish themselves as the dominant base learners, often setting the bar for the highest individual performance, although their lead varies; for example, BERT-SWC achieves 87.30% at 625 samples, while BERT-ResNet leads at 2.5K with 78.40%. This variation underscores the complementary nature of their prediction signals, which is necessary for effective Ensembling.

The true strength of the ensemble methodology is evident in the fact that the meta-learners consistently surpass the average performance of the base models. However, with 625 samples all the learners failed to improve performance, but low sample size can be the reason for these poor performances. Still XGBoost did provide the best accuracy of 85.13% among the ensemble learners. In fact, the XGBoost classifier consistently proved to be the most effective overall meta-learner across nearly all the data ranges. At 5K samples, XGBoost achieved 82.80%, which is not only the highest score in that row but also surpasses the best base model performance (BERT-ResNets 81.00%) by 1.8 percentage points. This confirmed that complex, non-linear ensemble techniques are vital for extracting the full benefit from the highly diverse prediction patterns of the three base models on tweets.

Overall, the ensemble methods successfully integrated the complementary strengths of the constituent models against the BERT-ResNet Textual Image classifier, resulting in a performance gain over the strongest single-model baseline. The XGBoost classifier acts as the best overall meta-learner across different datasets and sizes.

6.4.4.1 Comparison with published IMDB results

The preceding experiments compare the proposed Textual Image classifiers with the accompanying BERT and ensemble baselines. To provide external context, Table 6.8 contrasts the principal Experiment 2 results with selected published IMDB results from traditional sparse-feature classifiers, recurrent neural networks, pre-trained language models, and transformer-based models. The figures should be interpreted cautiously: the cited studies differ in model size, pre-training data, additional task-specific pre-training, preprocessing, and hyperparameter selection. They therefore provide benchmark context rather than a direct head-to-head comparison.
Table 6.8: Comparison of the IMDB results in Chapter 6 with selected published results across architecture families. Values marked "converted" were reported as test error in the source and are shown as accuracy using 100 − error.

Architecture family	**Study and model**	**Accuracy**	**Protocol and comparability note**
Bag-of-words	Linear SVM with delta-idf bag-of-words features [130]	88.89%	Standard IMDB train/test split; linear SVM over sparse lexical features.

Continued on next page

Architecture family	**Study and model**	**Accuracy**	**Protocol and comparability note**
Bag-of-ngrams	NBSVM with unigram and bigram features [141]	91.22%	Hybrid Naive Bayes–SVM classifier; a strong non-neural benchmark on the standard IMDB split using bi-gram.
RNN with pre-trained word2vec embeddings	Vanilla LSTM with word2vec initialisation [142]	89.10%	Single-layer LSTM with 128 hidden units; word2vec embeddings are fine-tuned. The study used a held-out validation subset, leaving 21,143 training reviews and the 25,000-review test set.
RNN token embeddings learned from scratch during pre-training tasks	ULMFiT (LSTM variant) [143]	95.40% (converted)	Reported as 4.60% error. Uses language-model pre-training and a different fine-tuning pipeline.
Transformer	BERT-base [144]	94.60% (converted)	Reported as 5.40% error. A conventional BERT fine-tuning reference, but with different experimental choices.
Transformer	BERT-base with in-task pre-training [144]	95.63% (converted)	Reported as 4.37% error. Includes an additional target-corpus adaptation stage and is not directly matched to the Chapter 6 BERT baseline.
Transformer	BERT-large with in-task pre-training [144]	95.79% (converted)	Reported as 4.21% error. Uses a larger 24-layer BERT model and additional target-corpus adaptation.
Transformer	XLNet-large [98]	96.80% (converted)	Reported as 3.20% error. Uses a large pretrained language model with a different pre-training objective than BERT model.
This thesis: transformer baseline	Fine-tuned BERT (Chapter 6 Experiment 2)	93.05%	Chapter 6 BERT baseline on the IMDB test set.
This thesis: Textual Image classifier	BERT–EfficientNet-B5, nearest-neighbour interpolation (Chapter 6 Experiment 2)	93.23% ± 0.10%	Mean over five runs on the IMDB test set. BERT embeddings are converted into contextual Textual Images and classified by a fine-tuned ImageNet-pretrained EfficientNet-B5.

The progression across architecture families is informative. Sparse bag-of-words and bag-of-ngram classifiers provide strong lexical baselines (88.23–91.22%), while the word2vec-initialised vanilla LSTM reports 89.10%. Language-model pre-training substantially changes this picture: ULMFiT, an LSTM variant approach, reports 95.40%, and the transformer-based systems report still higher results under their respective configurations. The BERT–EfficientNet-B5 Textual Image classifier preserves performance close to the directly fine-tuned BERT baseline while replacing the custom visual backbone with a pre-trained image classifier. Importantly, this work uses the smaller $\text{BERT}_{\text{base}}$ configuration, rather than $\text{BERT}_{\text{large}}$. $\text{BERT}_{\text{base}}$ has 12 transformer encoder layers and approximately 110 million parameters, whereas $\text{BERT}_{\text{large}}$ has 24 layers and approximately 340 million parameters [28]. Consequently, the $\text{BERT}_{\text{large}}$ and $\text{XLNet}_{\text{large}}$ results in Table 6.8 are not capacity-matched comparators; their stronger reported accuracy is expected in part because they use substantially larger language models. The BERT-large results also include additional in-task pre-training, which further strengthens their configuration. Thus, the contribution of the present experiment is not a claim of state-of-the-art IMDB performance. Instead, it demonstrates that contextual Textual Images can support competitive sentiment classification with ImageNet-pretrained visual backbones while retaining performance close to a $\text{BERT}_{\text{base}}$ baseline.

Table 6.9: Methodological comparison with closely related text-as-image vision-transfer work. These results use different data splits and visualisation procedures and must not be interpreted as a head-to-head numerical comparison.

Study	**Text-to-image and vision-model protocol**	**Evaluation and result**
This thesis: BERT–EfficientNet-B5 (Chapter 6 Experiment 2)	Fine-tuned BERT embeddings are reshaped and resized into contextual Textual Images; the ImageNet-pretrained EfficientNet-B5 backbone is fine-tuned.	Original IMDB test set; 93.23% ± 0.10% mean accuracy over five runs.
Benarab [145]	BERT embeddings are transformed into grayscale images using t-SNE; ImageNet-pretrained visual feature extractors are frozen.	40,000/10,000 train/validation split; approximately 85–87% validation accuracy across the reported backbones (AlexNet, ResNet, ResNext, ShuffleNet, VGG16).

A closely related approach was reported by Benarab [145], who transformed BERT embeddings into grayscale images and evaluated ImageNet-pretrained visual models. That study reported approximately 85–87% validation accuracy, but used frozen feature extractors and a 40,000/10,000 train/validation split rather than the predefined IMDB test set. Consequently, it is a useful methodological comparison, but its results are not numerically head-to-head with those in Table 6.9.

6.5 Summary

The central conclusion of this chapter is that replacing the computationally intensive custom visual backbone of the BERT-ResNet framework in the previous chapter with pre-trained image classifiers offers a low computation alternative for Textual Image classification.

Experiment 1 established the feasibility of this approach, showing that architectures adopting pre-trained visual backbones, like BERT-ResNet and BERT-ViT, perform comparably to baseline BERT while reducing the training overhead (faster trainings). Additionally, at low dataset ranges the BERT-ResNet mimics the underlying BERT model predictions. This is therefore a good way to provide text as an image equivalent representation without potentially harming the text model performance, which is beneficial to multimodality research. At higher data ranges (640k-1.6M), the increase in data available results in only a small divergence in predictions with results like increases of 0.4% - 0.2% on average BERT-ResNet performance compared to baseline.

Experiment 2 broadened the scope to compare Convolutional (ResNet, DenseNet, EfficientNet) and Transformer-based (ViT, Swin Transformer) pre-trained vision models. This critical analysis also evaluated the impact of image interpolation techniques necessary for resizing the Textual Images to fit the pre-trained inputs of these new models. The findings favoured the CNN architecture, particularly Efficientnet_b5, which achieved the highest accuracy (93.37%) with lowest average training epochs, though proximity in performance for all the models is observed with no significant difference in average performance. Crucially, the Nearest neighbor method is the preferred interpolation method, due to its lower computation overhead, as the model performance is not significantly impacted by interpolation. So, nearest neighbour interpolation preserves the crucial sentiment information competitively when compared to other interpolation methods, while requiring the least computation resources. A notable result was the failure of the Swin Transformer to converge, underscoring the specialized nature of these synthetically generated images compared to natural images.

Experiment 3 investigated the effect of model scaling and data availability using the family of EfficientNet models (b0 to b7) in a fixed 3 epoch setting to mimic low computational resource availability. The results demonstrated a clear trend: while accuracy increased with larger training datasets, the performance benefits of moving from smaller, highly efficient models (b0) to larger, computationally expensive models (b7) diminished significantly as data size increased. This finding is central to adopting sustainable ML practices, as it validates that computationally lighter architectures, such as (B3-B6), offer a sweet spot for a performance-to-compute trade-off for this task, particularly when abundant training data is available. Surprisingly, the (b0) model performance is competitive in several data availabilities.

Experiment 4 investigated Ensembling BERT-ResNet output (P1) with a lexically enhanced BERT-SWC model (P2) and a traditional topic-modeling classifier (P3), the ensemble achieved a maximal accuracy of 94.32% using an XGBoost meta-learner on the IMDB dataset. This represents a 1.17% improvement over the best individual BERT-ResNet performance. The success of the XGBoost meta-learner over simple Soft Voting (93.34%) underscores

the necessity of using complex, non-linear learners to decode the intricate relationships among diverse predictions.

In essence, fine-tuning efficient pre-trained vision models on contextual Textual Images have been established as a strong methodology that is both accurate and aligned with sustainable machine learning practices.

Chapter 7

Conclusion

7.1 Summary of the Research

This research bridges Natural Language Processing (NLP) and Computer Vision through the Textual Image Classification paradigm for Sentiment Analysis (SA). While Optical Character Recognition (OCR) traditionally translates visual data into text, this thesis explores the complementary direction: transforming text representations into structured visual forms so that the spatial feature-extraction strengths of computer vision models can be used for sentiment classification.

The primary aim was to explore methodologies for classifying text by converting it into greyscale images. This was achieved by utilising NLP text embeddings approaches to form Textual Images and using image classifiers to extract sentiment information. Various Textual Images classification methods were explored, evaluating the trade-offs between computational cost and model performance.

To achieve this, the thesis developed and systematically evaluated hybrid architectures combining RNN (BLSTM) and Transformers (BERT) with deep computer vision backbones (ResNet, EfficientNet, DenseNet, and Vision Transformers). The investigation was guided by four research questions, progressing from the validation of deep 2D CNN extraction as Textual Image classifier to the integration of attention mechanisms to generate contextualised Textual Images. Finally, to the incorporation of sustainable model building approaches like using pre-trained BERT language model weights, dropping BLSTM to fully parallelise the training for hardware optimisation, replacing visual backbone for Textual Images classifier with pre-trained vision models and using Ensembling methods to combine Textual Image classifier probabilities with other complementary contemporary as well as traditional approaches for sentiment analysis.

7.2 Synthesis of Findings Addressing Research Questions

This section synthesises the evidence from Chapters 4–6 to answer the four Research Questions (RQs) stated in Section 1.7. Each subsection gives a direct answer, identifies the supporting experiments, interprets the result within its experimental conditions and places it in relation to relevant work.

7.2.1 RQ1: Textual Images and Two-Dimensional Neural Pattern Recognition

RQ1: What is the impact of converting textual data into image-equivalent representations (Textual Images), to support two-dimensional neural pattern recognition, on analysing text for sentiment?

The experiments in Chapter 4 demonstrate that text, when represented as Image (2D feature maps), can be effectively analysed using deep convolutional architectures. Textual Image classification integrates Natural Language Processing techniques for information encoding with Computer Vision models for information classification, allowing textual content to be analysed in image form.

Textual Image classification involves forming Textual Images through neural textual encoding and then classifying them using image classification models. The encoder transforms the input word embeddings into a two-dimensional matrix, in which the rows correspond to the words or tokens in the text and the columns correspond to embedding or learned feature dimensions. This representation can capture temporal, sequential and contextual dependencies. In the first pipeline, the BLSTM output is arranged as a single-channel Textual Image and classified using the

purpose-built shallow or deep 2D ResNet variants proposed in this work.

This chapter introduced two different deep 2D-CNN based residual neural network models for sentiment analysis on Textual Images. One is called shallow BLSTM–ResNet and the other deep BLSTM–ResNet. Both models replaced a shallow 2D-CNN construction of the existing BLSTM–2DCNN model with ResNet, which was a deeper counterpart. The experimental results demonstrated that, with an increase in the depth of BLSTM–ResNet, the network performance generally increases. On the 80,000-sample Sentiment140 subset, BLSTM–ResNet–12 achieved 78.80% ± 0.33, compared with 75.84% ± 0.30 for BLSTM–2DCNN.

In addition, an overall improvement is observed in accuracy with every additional residual block in the shallow BLSTM–ResNet model until accuracy saturates. The deep BLSTM–ResNet models show the same positive impact of network depth and dataset size on sentiment analysis. Further investigation of the shallow and deep BLSTM–ResNet models shows that the deep BLSTM–ResNet model outperforms the shallow BLSTM–ResNet model at smaller dataset sizes and performs competitively with the BLSTM–2DCNN network.

The BLSTM–ResNet is an extension of the BLSTM–2DCNN model proposed by [27], in which the shallow 2D-CNN was replaced with a deeper 2D-CNN architecture inspired by ResNet. The BLSTM captures temporal features from the text embeddings and generates Textual Images, which are then classified using the two-dimensional convolutional-based ResNet. The thesis therefore acknowledges the BLSTM–2DCNN approach as a precursor and does not claim that the combination of a BLSTM and a residual network is itself new. Its contribution is the investigation of the impact of residual networks for sentiment analysis on Textual Images, including residual depth and block construction.

The ResNet–BLSTM end-to-end speech-recognition architecture reported by [146] provides a useful contrast with the BLSTM–ResNet adopted in this thesis. In the speech-recognition model, the ResNet is placed first to extract and compress local acoustic patterns from time–frequency spectrograms, including robust features from noisy speech signals. The BLSTM then processes these acoustic features as a temporal sequence before they are passed to a Connectionist Temporal Classification decoder for variable-length speech recognition. In contrast, this thesis uses a BLSTM first to capture temporal features from text embeddings and generate Textual Images; the subsequent ResNet extracts two-dimensional features for document-level sentiment classification. Thus, a reverse ordering of ResNet and BLSTM already exists for speech recognition, but the blocks serve different purposes and are applied to a different input representation and task.

7.2.2 RQ2: Impact of Contextualised Embeddings

RQ2: What is the impact of incorporating contextual information into Textual Images on analysing text for sentiment across varying depth of image classifiers, dataset sizes and text lengths?

Chapter 5 first explored a novel method of generating contextualised Textual Images for extracting sentiment. The BLSTM–ResNet model was extended to incorporate a fine-tuned BERT model to extract contextualised embeddings, which were then used to create Contextualised Textual Images and classified using a standard ResNet network. The first experimental results with the Sentiment140 dataset demonstrated that the BERT–BLSTM–ResNet network outperformed the BLSTM–ResNet. On the 20,000-sample Sentiment140 comparison, BLSTM–ResNet–12 achieved 72.85%, whereas BERT–BLSTM–ResNet18 achieved 83.40%.

The BERT model on its own also performed well; however, on smaller datasets it can benefit from the use of the full BLSTM–ResNet model. The results on Sentiment140 show that BERT–BLSTM–ResNet18 outperformed fine-tuned BERT if the training data size is lower. A key finding was the crossover point of 20000 regarding dataset size.

In addition, the chapter investigated token-length impact on the BERT–BLSTM–ResNet architectures using the IMDB dataset, together with various BLSTM and ResNet configurations extending the BERT-generated contextualised word embeddings. The BERT–BLSTM–ResNet, BERT–BLSTM ||ResNet and BERT–ResNet models outperformed BERT at most token lengths. Accuracy generally improves as token length increases, with the gains plateauing as token length approaches 384 and 512. The BERT–ResNet model also emerged as a good alternative with the potential for faster training by circumventing the BLSTM block and directly utilising the contextualised Textual Images generated using BERT, but it has more model parameters than the BERT–BLSTM–ResNet model because of the large input image size generated from the BERT output.

While adding complexity to BERT generally increases both parameter count and FLOPs, the integration of ResNet, despite its higher FLOPs, offers significant parallelisation benefits that can lead to faster training compared to purely sequential processing.

7.2.3 RQ3: Pre-trained CNN and Transformer-Based Image Classifiers

RQ3: How do the architecture and complexity of pre-trained Convolutional neural network and Transformer-based Image classifiers affect the performance and computational efficiency of sentiment classification using Textual Images?

The findings of Chapter 6 answer RQ3 by showing that image-classification networks previously trained to recognise visual patterns can be fine-tuned to classify Textual Images. Based on the findings in Chapter 5, BERT can itself generate Contextualised Textual Images, but direct BERT–ResNet produces larger Textual Images. These networks can benefit when the custom ResNet image classifier is replaced with pre-trained vision models. ResNet and ViT were selected as representatives of the CNN and Transformer vision-model families, respectively.

Experiment 1 and Experiment 2 established the efficacy of pre-trained vision classifiers as a sustainable alternative to training a custom ResNet from scratch. The comparative analysis suggests that, for the synthetic Textual Images used in this work, CNN-based architectures are strong alternatives to the tested Vision Transformer models in both performance and convergence behaviour. The pre-trained CNN image classifiers, particularly ResNet50 and EfficientNet, achieved competitive performance, while EfficientNet-B5 achieved the highest individual accuracy of 93.37% and the highest five-run mean accuracy of 93.23% $\pm$ 0.10 on IMDB using nearest-neighbour interpolation. The ViT variant achieved similarly competitive accuracy, but the convergence epochs varied with the model and interpolation method. Thus, the experiments favour EfficientNet-B5 in the tested setting, although they do not establish a universal performance ranking between CNNs and Vision Transformers.

A critical negative result was the failure of the Swin Transformer to converge, with accuracy remaining at 50.00% in the tested configuration. This highlights that not all computer-vision architectures are directly transferable to Textual Image classification without modification. It may reflect a mismatch between the shifted-window attention mechanism, developed for natural images, and the discrete structure of semantic vectors in Textual Images; further experiments are required before treating this as a fundamental incompatibility. Experiment 2 also showed that nearest-neighbour interpolation is preferred because it has lower computation overhead, while model performance was not significantly affected by the interpolation method.

Experiment 3 validated the scaling effect of EfficientNet-B0–B7 in a constrained training setup of three epochs. While larger models can perform better, the performance gain diminishes as model complexity and training-data size increase. Computationally lighter models, particularly EfficientNet-B3 to B6, offer a useful performance-to-compute trade-off for this task, while EfficientNet-B0 remained competitive at several data availabilities. This aligns with the sustainable machine-learning objective of selecting a model on the basis of both predictive performance and computational cost.

7.2.4 RQ4: Complementary Predictive Information and Ensembling

RQ4: To what extent do Textual Image classifiers using pretrained Image classifier provide complementary predictive information to established text-based sentiment models, and how does combining these representations affect classification performance?

Experiment 4 in Chapter 6 investigated ensembling the BERT–ResNet output (P1) with a lexically enhanced BERT–SWC model (P2) and a traditional topic-modelling classifier (P3). The ensemble achieved a maximal accuracy of 94.32% using an XGBoost meta-learner on the IMDB dataset. This represents a 1.17% improvement over the best individual BERT–ResNet performance. The success of the XGBoost meta-learner over simple soft voting (93.34%) underscores the necessity of using complex, non-linear learners to decode the intricate relationships among diverse predictions.

Overall, the ensemble methods successfully integrated the complementary strengths of the constituent models against the BERT–ResNet Textual Image classifier, resulting in a performance gain over the strongest single-model baseline. The XGBoost classifier acts as the best overall meta-learner across different datasets and sizes.

Ensembling for sentiment analysis is not new. For example, Tan et al. [147] proposed an ensemble hybrid deep-learning model that integrates RoBERTa embeddings with LSTM, BiLSTM and GRU networks. The ensemble combines individual model predictions through averaging and majority voting to use the complementary structural strengths of the models and reduce individual classification errors.

In contrast, the present work combines probabilities from different types of sentiment-analysis approaches and different text-processing representations: the BERT–ResNet Textual Image classifier, the lexically enhanced BERT–SWC model, and the traditional LSA–LDA/SVM topic-modelling classifier. Rather than combining recurrent variants operating on the same RoBERTa representation, the ensemble tests whether visual, lexical and topic-based probabilities provide complementary information. The XGBoost meta-learner is then used to learn the relationship among these diverse predictions. This is an architectural comparison rather than a direct ranking of accuracy,

because preprocessing, data splits, label balance and evaluation settings may differ.

7.3 Discussion in Relation to Previous Work

7.3.1 From a precursor to a staged research programme

The BLSTM–2DCNN architecture proposed by [27] is an example of creating Textual Images with a BLSTM that can be used by a 2D-CNN to extract features. The BLSTM–ResNet proposed in this thesis is an extension of the BLSTM–2DCNN model, in which the shallow 2D-CNN was replaced with a deeper 2D-CNN architecture inspired by ResNet. The BLSTM captures temporal features from the text embeddings and generates Textual Images, which are then classified using a two-dimensional convolutional-based ResNet.

The primary focus of this work is exploring different Textual Image generation and Textual Image classification techniques. The five pipelines form a staged experimental framework rather than a set of independent alternatives. The investigation progresses from a static word representation, to a contextualised representation, to an assessment of whether the BLSTM remains necessary after contextual encoding, and then to transfer learning with pre-trained vision classifiers and the fusion of complementary sentiment representations. Pipeline 1 establishes the Textual Image approach using GloVe, BLSTM and a custom ResNet. Pipelines 2 and 3 introduce BERT contextualisation and compare BLSTM-based and direct image formation. Pipeline 4 replaces the custom image classifier with pre-trained CNN or Transformer-based vision classifiers. Finally, Pipeline 5 combines the probability outputs of the Textual Image classifier with lexical and topic-based text models.

This work therefore extends the BLSTM–2DCNN approach through the investigation of residual depth and block construction for sentiment analysis on Textual Images, followed by contextualised Textual Image generation, pre-trained image classifiers and ensembling. The experimental results demonstrate that text, when represented as Image (2D feature maps), can be effectively analysed using deep convolutional architectures.

7.3.2 Relationship to hybrid sequence-processing architectures

The ResNet–BLSTM end-to-end speech-recognition architecture reported by [146] shows that a reverse ordering of ResNet and BLSTM has been used in another sequential domain. In that model, the ResNet is placed first to extract and compress local acoustic patterns from time–frequency spectrograms. The BLSTM then processes these acoustic features as a temporal sequence before they are passed to a Connectionist Temporal Classification decoder for speech recognition.

In contrast, this thesis uses the BLSTM first to capture temporal features from text embeddings and to generate Textual Images; the subsequent ResNet extracts two-dimensional features for sentiment classification. The present work does not claim that the combination of ResNet and BLSTM is itself novel. Its contribution is the use of this combination for Textual Image sentiment analysis and the investigation of contextualised embeddings, ResNet depth, dataset size, token length, pre-trained image classifiers and Ensembling methods.

7.3.3 Relationship to ensemble sentiment analysis

Ensembling for sentiment analysis is not new. For example, [147] proposed an ensemble hybrid deep-learning model that integrates RoBERTa embeddings with LSTM, BiLSTM and GRU networks. The ensemble combines individual model predictions through averaging and majority voting to use the complementary structural strengths of the models and reduce individual classification errors.

This work investigates Ensembling BERT–ResNet output (P1) with a lexically enhanced BERT–SWC model (P2) and a traditional topic-modelling classifier (P3). The three branches represent sentiment differently: P1 captures spatial patterns in contextualised Textual Images, P2 adds lexical sentiment knowledge to contextual BERT representations, and P3 captures global distributional and topic information. These individual probabilities are subsequently combined by a meta-learner, which integrates their complementary strengths to achieve a final sentiment classification. The use of an XGBoost meta-learner demonstrates the necessity of using complex, non-linear learners to decode the intricate relationships among diverse predictions.

The present work therefore combines probabilities from different text-processing approaches rather than recurrent variants operating on the same RoBERTa representation. This is an architectural comparison, rather than a direct ranking of accuracy, because preprocessing, data splits, label balance and evaluation settings may differ.

7.3.4 Thesis-level contribution and its boundaries

This research advances the field of sentiment analysis by establishing a comprehensive framework for Textual Image Classification. It bridges the gap between Natural Language Processing and Computer Vision, demonstrating that text, when represented as Image (2D feature maps), can be effectively analysed using deep convolutional architectures. The research introduces and validates Contextualised Textual Images using fine-tuned BERT embeddings, validates the transferability of pre-trained computer-vision models to the generated Textual Images, and demonstrates that Textual Image classifiers can be combined with lexical and topic-modelling approaches in an Ensembling setting.

7.4 Limitations of the Experimental Work

While the research successfully meets its objectives and demonstrates the potential of Textual Image Classification, several limitations should be acknowledged to contextualise the findings.

First, the datasets define the task scope. Sentiment140 and IMDB provide English binary document-level sentiment data, with short social-media posts and longer movie reviews respectively. They do not establish performance for neutral or fine-grained sentiment, aspect-level sentiment, other languages, dialects, domains, sarcasm or temporal change. The RQ conclusions should therefore be read as evidence for the document-level binary settings evaluated here.

Second, the compute environment constrained the experimental design. The available High-Performance Computing resources required smaller batch sizes and early stopping, particularly for the 1.6-million-sample Sentiment140 experiments. These conditions are part of the reported performance–compute trade-offs; they limit conclusions about the performance attainable with substantially longer training or different hardware configurations.

Third, the number of repetitions was limited. The five to ten runs used in the statistical analyses led to low statistical power for some comparisons, including parts of the Chapter 5 analysis. Large observed effects are informative, but small differences between models and non-significant comparisons require caution until they are replicated with more runs and fully recorded hardware conditions.

Fourth, several representation and architecture choices were fixed to maintain a manageable scope. The BLSTM hidden size was fixed at 300, which determines the width of the generated Textual Image. The selected BERT representation, residual-block designs and conventional 3×3 convolutional kernels also restrict conclusions about optimal image resolution, feature-axis layout, kernel geometry and model capacity.

Fifth, the transfer-learning experiments use ImageNet-pre-trained visual models that require fixed input sizes and interpolation. BERT inputs were also evaluated only up to 512 tokens. These choices limit conclusions about very long documents and about information loss or distortion caused by resizing a Textual Image for a pre-trained classifier. ImageNet pre-training provides a useful initialisation, but does not demonstrate a semantic equivalence between natural images and Textual Images.

Finally, the evaluation concentrates on accuracy and weighted F1 in balanced experimental settings. It does not establish calibration, robustness, fairness or interpretability. In particular, the ensemble improvement supports complementary predictions, but does not show which token–feature structures or linguistic phenomena are responsible for the gain.

7.5 Prospects and Direction

The domain of Textual Image Classification is still nascent. The findings of this thesis provide several directions for further work on Textual Image generation, Textual Image classification and Ensembling with existing sentiment-analysis techniques.

7.5.1 Multimodal sentiment analysis

Multimodality is a promising direction for Textual Image Classification. In multimodality, multiple different input signals, such as text, audio, video and images, are used to perform sentiment classification. The BERT–ResNet approach provides text as an image-equivalent representation without potentially harming text-model performance, which is beneficial to multimodality research.

Future work can investigate the suitability of combining Textual Image classification with original images, video

frames, audio and other signals available in social-media data. The probabilities generated by the different classifiers can be combined using a meta-learner, as in the Ensembling experiments in this thesis. This would investigate whether Textual Image classification can complement sentiment information extracted from other input modes.

7.5.2 Representation design and long documents

Further investigation can vary the BLSTM hidden units, contextual representation representations and Textual Image dimensions. The BLSTM hidden units were fixed at 300 in the present work and determine the width of the Type B Textual Image. Controlled experiments with different hidden-unit sizes can investigate the impact of Textual Image size and resolution on image-classifier performance.

The IMDB experiments also show the importance of document length. Further work can investigate Textual Image classification on longer text documents and compare different methods of forming Textual Images from such documents. This can include splitting a document into smaller sequences, forming Textual Images from sentence-level representations, and comparing different document lengths.

The *inverse OCR* analogy remains a useful direction for future research. Further work can investigate whether image-classification methods used for segmentation and localisation can be used with Textual Images for topic modelling and aspect-level sentiment analysis.

7.5.3 Architectures matched to Textual Image geometry

The standard 3×3 kernels used in the ResNet models are adopted from image classification. Future work can investigate non-square kernels and different kernel sizes for Textual Images, as the height and width represent different textual information. The Vision Transformer and Swin Transformer models can also be investigated further with different input sizes and model settings. The failure of the Swin Transformer to converge in the current experiment highlights the need for further investigation of Transformer-based image classifiers for Textual Images.

7.5.4 Interpretability, robustness and responsible use

Further work can investigate which parts of a Textual Image contribute to a sentiment prediction. Prediction analysis, word masking and error analysis can be used to explore the textual information extracted by the image classifier. The Textual Image classification methods can also be tested on text with spelling variations, negation, different languages and different application domains.

7.6 Conclusion

This thesis has successfully explored the viability of Textual Image Classification as a robust, high-performance, and sustainable alternative for Sentiment Analysis. By bridging the gap between the linguistic depth of NLP and the spatial feature extraction capabilities of Computer Vision, this research demonstrated that text, when transformed into a visual latent representation, can be analysed with precision using deep convolutional backbones.

The work highlights that while pure Transformer models (BERT) are powerful, hybrid architectures that visualise text offer superior sample efficiency and distinct predictive capabilities that are complementary in ensemble settings for datasets with over a million samples. By validating the use of pre-trained visual classifiers and establishing the efficacy of contextualised Textual Images, this research provides a foundation for a new class of text analysis algorithms that recast the text problem into a vision problem.

Chapter 8

References

[1] DataReportal, *Digital 2024: Global overview report*, 2024. [Online]. Available: https://datareportal.com/reports/digital-2024-global-overview-report

[2] A. R. Damasio, "Emotion in the perspective of an integrated nervous system," *Brain Research Reviews*, vol. 26, pp. 83–86, 2-3 May 1998, ISSN: 0165-0173. DOI: 10.1016/S0165-0173(97)00064-7

[3] I. Mehta, *Twitter has officially changed its logo to x*, 2023. [Online]. Available: https://techcrunch.com/2023/07/24/twitter-has-officially-changed-its-logo-to-x/

[4] M. N. Shiota, "Ekman's theory of basic emotions," *The SAGE Encyclopedia of Theory in Psychology*, pp. 249–250, Mar. 2016. DOI: 10.4135/9781483346274

[5] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A system for real-time twitter sentiment analysis of 2012 u.s. presidential election cycle," Tech. Rep., 2012, pp. 8–14. [Online]. Available: http://t.co/qEns1Pmi

[6] J. Ramteke, S. Shah, D. Godhia, and A. Shaikh, "Election result prediction using twitter sentiment analysis," *Proceedings of the International Conference on Inventive Computation Technologies, ICICT 2016*, vol. 1, Jul. 2016. DOI: 10.1109/INVENTIVE.2016.7823280

[7] V. K. Singh, R. Piryani, A. Uddin, and P. Waila, "Sentiment analysis of movie reviews: A new feature-based heuristic for aspect-level sentiment classification," in *Proceedings - 2013 IEEE International Multi Conference on Automation, Computing, Control, Communication and Compressed Sensing, iMac4s 2013*, 2013, pp. 712–717, ISBN: 9781467350891. DOI: 10.1109/iMac4s.2013.6526500

[8] K. Dashtipour, M. Gogate, A. Adeel, H. Larijani, and A. Hussain, "Sentiment analysis of persian movie reviews using deep learning," *Entropy 2021*, vol. 23, 5 2021, ISSN: 1099-4300. DOI: 10.3390/e23050596 [Online]. Available: https://doi.org/10.3390/e23050596

[9] S. Bharathi and A. Geetha, "Sentiment analysis for effective stock market prediction," *International Journal of Intelligent Engineering and Systems*, vol. 10, pp. 146–154, 3 2017, ISSN: 21853118. DOI: 10.22266/ijies2017.0630.16

[10] N. Jing, Z. Wu, and H. Wang, "A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction," *Expert Systems with Applications*, vol. 178, p. 115 019, Sep. 2021, ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2021.115019

[11] A. E. de Oliveira Carosia, G. P. Coelho, and A. E. A. da Silva, "Investment strategies applied to the brazilian stock market: A methodology based on sentiment analysis with deep learning," *Expert Systems with Applications*, vol. 184, p. 115 470, Dec. 2021, ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2021.115470

[12] M. Mohan, A. K. Abhinav, A. Ashok, A. V. Akhil, and P. R. Achinth, "Depression detection using facial expression and sentiment analysis," *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pp. 1–6, Aug. 2021. DOI: 10.1109/ASIANCON51346.2021.9544819 [Online]. Available: https://ieeexplore.ieee.org/document/9544819/

[13] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A survey on sentiment analysis methods, applications, and challenges," *Artificial Intelligence Review 2022 55:7*, vol. 55, pp. 5731–5780, 7 Feb. 2022, ISSN: 1573-7462. DOI: 10.1007/S10462-022-10144-1 [Online]. Available: https://link.springer.com/article/10.1007/s10462-022-10144-1

[14] J. F. Sánchez-Rada and C. A. Iglesias, "Social context in sentiment analysis: Formal definition, overview of current trends and framework for comparison," *Information Fusion*, vol. 52, pp. 344–356, Dec. 2019, ISSN: 1566-2535. DOI: 10.1016/J.INFFUS.2019.05.003

[15] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," pp. 79–86, 2002. DOI: 10.3115/1118693.1118704 [Online]. Available: https://dl.acm.org/doi/pdf/10.3115/1118693.1118704

[16] A. Go, R. Bhayani, and L. Huang, *Twitter sentiment classification using distant supervision*, 2009. [Online]. Available: http://tinyurl.com/cvvg9a

[17] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2013, pp. 1631–1642. DOI: 10.18653/v1/D13-1170

[18] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2004, pp. 168–177. DOI: 10.1145/1014052.1014073

[19] P. J. Stone and E. B. Hunt, "A computer approach to content analysis: Studies using the general inquirer system," in *AFIPS Conference Proceedings - 1963 Spring Joint Computer Conference, AFIPS 1963*, ACM Press, 1963, pp. 241–256. DOI: 10.1145/1461551.1461583

[20] U. Naseem, I. Razzak, S. K. Khan, and M. Prasad, "A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models," *Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, 5 Jun. 2021, ISSN: 23754702. DOI: 10.1145/3434237 [Online]. Available: https://dl.acm.org/doi/10.1145/3434237

[21] H. Liang, X. Sun, Y. Sun, and Y. Gao, *Text feature extraction based on deep learning: A review*, Dec. 2017. DOI: 10.1186/s13638-017-0993-1 [Online]. Available: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-017-0993-1

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, Jan. 2013. [Online]. Available: https://arxiv.org/pdf/1301.3781

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 770–778, Dec. 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.90

[24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Gated feedback recurrent neural networks*, Jun. 2015. [Online]. Available: http://proceedings.mlr.press/v37/chung15.html

[25] Z. Huang, B. Research, W. Xu, and K. Y. Baidu, "Bidirectional lstm-crf models for sequence tagging," Aug. 2015. [Online]. Available: https://arxiv.org/pdf/1508.01991

[26] C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau, *A c-lstm neural network for text classification*, 2015. arXiv: 1511.08630 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1511.08630

[27] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, "Text classification improved by integrating bidirectional lstm with two-dimensional max pooling," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, Association for Computational Linguistics, Nov. 2016, pp. 3485–3495, ISBN: 9784879747020. arXiv: 1611.06639. [Online]. Available: https://aclanthology.org/C16-1329/

[28] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of the 2019 Conference of the North*, pp. 4171–4186, 2019. DOI: 10.18653/V1/N19-1423 [Online]. Available: https://aclanthology.org/N19-1423/

[29] D. Meretakis, D. Fragoudis, H. Lu, S. Likothanassis, and ². L. Gr, "Scalable association-based text classification," Tech. Rep., 2000.

[30] J. Suontausta and J. Häkkinen, "Decision tree based text-to-phoneme mapping for speech recognition," 2000. [Online]. Available: http://www.iscaaspeech.org/archive

[31] R. Basili, A. Moschitti, and M. T. Pazienza, "Nlp-driven ir: Evaluating performances over a text classification task," Tech. Rep.

[32] Y. Jia, M. Johnson, W. Macherey, R. J. Weiss, Y. Cao, C. C. Chiu, N. Ari, S. Laurenzo, and Y. Wu, "Leveraging weakly supervised data to improve end-to-end speech-to-text translation," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2019-May, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 7180–7184, ISBN: 9781479981311. DOI: 10.1109/ICASSP.2019.8683343

[33] P. Bahar, T. Bieschke, and H. Ney, "A comparative study on end-to-end speech to text translation," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Dec. 2019, pp. 792–799, ISBN: 9781728103068. DOI: 10.1109/ASRU46091.2019.9003774

[34] Z. Zhang, J. Pohjalainen, A. E.-d. Mousa, W. Jin, J. Geiger, A. E.-D. Mousa, B. Schuller, W. Jin, J. Pohjalainen, and A. E.-D. Mousa, "Deep learning for environmentally robust speech recognition: An overview of recent developments," *ACM Transactions on Intelligent Systems and Technology*, vol. 9, p. 49, 5 2018. DOI: 10.1145/3178115 [Online]. Available: https://doi.org/10.1145/3178115

[35] K. J. Han, S. Hahm, B. H. Kim, J. Kim, and I. Lane, "Deep learning-based telephony speech recognition in the wild," in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2017-Augus, International Speech Communication Association, 2017, pp. 1323–1327. DOI: 10.21437/Interspeech.2017-1695

[36] P. Klosowski, "Deep learning for natural language processing and language modelling," *Signal Processing - Algorithms, Architectures, Arrangements, and Applications Conference Proceedings, SPA*, vol. 2018-Septe, pp. 223–228, Dec. 2018, ISSN: 23260319. DOI: 10.23919/SPA.2018.8563389

[37] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, and T.-Y. Liu, "Incorporating bert into neural machine translation," Feb. 2020. [Online]. Available: http://arxiv.org/abs/2002.06823

[38] *Keras: Deep learning for humans*. [Online]. Available: https://keras.io/

[39] *Tensorflow*. [Online]. Available: https://www.tensorflow.org/

[40] *Google - wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Google

[41] *Pytorch*. [Online]. Available: https://pytorch.org/

[42] *Meta - wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Meta

[43] M. ; Oproescu, O. P. ; Stan, C. E. Gordan, N. Doulamis, M. W. Wo´zniak, O.-C. Novac, M. C. Chirodea, C. M. Novac, N. Bizon, M. Oproescu, O. P. Stan, and C. E. Gordan, "Analysis of the application efficiency of tensorflow and pytorch in convolutional neural network," *Sensors 2022, Vol. 22, Page 8872*, vol. 22, p. 8872, 22 Nov. 2022, ISSN: 1424-8220. DOI: 10.3390/S22228872 [Online]. Available: https://www.mdpi.com/1424-8220/22/22/8872/htm%20https://www.mdpi.com/1424-8220/22/22/8872

[44] M. Ashraf, Y. K. Salal, S. M. Abdullaev, M. Zaman, and M. A. Bhut, "Introduction of feature selection and leading-edge technologies viz. tensorflow, pytorch, and keras: An empirical study to improve prediction accuracy of cardiovascular disease," pp. 19–31, 2022, ISSN: 2194-5365. DOI: 10.1007/978-981-16-3071-2_2 [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-16-3071-2_2

[45] L. Liu, Y. Wu, W. Wei, W. Cao, S. Sahin, and Q. Zhang, "Benchmarking deep learning frameworks: Design considerations, metrics and beyond," *Proceedings - International Conference on Distributed Computing Systems*, vol. 2018-July, pp. 1258–1269, Jul. 2018. DOI: 10.1109/ICDCS.2018.00125 [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8416388

[46] R. Ahmad, I. Alsmadi, and M. Al-Ramahi, "Optimization of deep learning models: Benchmark and analysis," *Advances in Computational Intelligence 2023 3:2*, vol. 3, pp. 1–15, 2 Mar. 2023, ISSN: 2730-7808. DOI: 10.1007/S43674-023-00055-1 [Online]. Available: https://link.springer.com/article/10.1007/s43674-023-00055-1

[47] X. Zheng, J. Feng, Y. Chen, H. Peng, and W. Zhang, "Learning context-specific word/character embeddings," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, pp. 3393–3399, 1 Feb. 2017, ISSN: 2374-3468. DOI: 10.1609/AAAI.V31I1.10985 [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10985

[48] Y. Li, W. Li, F. Sun, and S. Li, "Component-enhanced chinese character embeddings," *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pp. 829–834, Aug. 2015. DOI: 10.18653/v1/d15-1098 [Online]. Available: https://arxiv.org/pdf/1508.06669

[49] J. Du, S. Michalska, S. Subramani, H. Wang, and Y. Zhang, "Neural attention with character embeddings for hay fever detection from twitter," *Health Information Science and Systems 2019 7:1*, vol. 7, pp. 1–7, 1 Oct. 2019, ISSN: 2047-2501. DOI: 10.1007/S13755-019-0084-2 [Online]. Available: https://link.springer.com/article/10.1007/s13755-019-0084-2

[50] J. Li, C. Zhang, Y. Sun, J. Han, and E. Ding, "Detecting text in the wild with deep character embedding network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11364 LNCS, pp. 501–517, 2019, ISSN: 16113349. DOI: 10.1007/978-3-030-20870-7_31/TABLES/5 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-20870-7_31

[51] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[52] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*, 3rd. Aug. 2025. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[53] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, pp. 2931–2939, May 2014. [Online]. Available: https://arxiv.org/pdf/1405.4053

[54] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," *EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, pp. 670–680, 2017. DOI: 10.18653/V1/D17-1070 [Online]. Available: https://aclanthology.org/D17-1070/

[55] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, vol. 4, pp. 3104–3112, January Sep. 2014, ISSN: 10495258. [Online]. Available: https://arxiv.org/pdf/1409.3215

[56] *Twitter - wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Twitter

[57] *Tweet (social media) - wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Tweet_(social_media)

[58] T. Singh and M. Kumari, "Role of text pre-processing in twitter sentiment analysis," in *Procedia Computer Science*, vol. 89, Elsevier B.V., Jan. 2016, pp. 549–554. DOI: 10.1016/j.procs.2016.06.095

[59] J. Gnana, R. #1, A. Aloysius, and S. Banumathi, "An enhanced data cleaning algorithm for twitter data," Tech. Rep., 2018. [Online]. Available: www.ijsrcsams.com

[60] *Twitter us airline sentiment | kaggle*. [Online]. Available: https://www.kaggle.com/crowdflower/twitter-airline-sentiment

[61] V. Gurusamy, S. Kannan, and A. Professor, *Preprocessing techniques for text mining multilingual natural language processing view project agent based distributed framework for big data view project preprocessing techniques for text mining*, 2014. [Online]. Available: https://www.researchgate.net/publication/273127322

[62] A. Cogato, F. Meggio, M. De, A. Migliorati, and F. Marinello, "Sustainability extreme weather events in agriculture: A systematic review," *Sustainability 2019, Vol. 11, Page 2547*, vol. 11, p. 2547, 9 May 2019. DOI: 10.3390/su11092547 [Online]. Available: www.mdpi.com/journal/sustainability

[63] Y. Zhang, Y. Wang, and W. Yang, "Research on data cleaning in text clustering," in *Proceedings - 2010 International Forum on Information Technology and Applications, IFITA 2010*, vol. 1, 2010, pp. 305–307, ISBN: 9780769541150. DOI: 10.1109/IFITA.2010.73

[64] V. Srividhya and R. Anitha, "Evaluating preprocessing techniques in text categorization," *International Journal of Computer Science and Application Issue*, 2010, ISSN: 0974-0767.

[65] T. Korenius, J. Laurikkala, K. Järvelin, and M. Juhola, "Stemming and lemmatization in the clustering of finnish text documents," in *International Conference on Information and Knowledge Management, Proceedings*, Association for Computing Machinery (ACM), 2004, pp. 625–633. DOI: 10.1145/1031171.1031285 [Online]. Available: http://www.lingsoft.fi/cgi-bin/fintwol/

[66] V. Balakrishnan and E. Lloyd-Yemoh, "Stemming and lemmatization: A comparison of retrieval performances," in *Proceedings of SCEI Seoul Conferences*, 2014, pp. 174–179.

[67] L. Nada and M. D. P. Joel, "A rule based approach to word lemmatization," in *Proceedings C of the 7th International Multi-Conference Information Society IS*, 1996.

[68] *Nltk :: Natural language toolkit*. [Online]. Available: https://www.nltk.org/

[69] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008, ISBN: 0521865719. [Online]. Available: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[70] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 60, pp. 493–502, 5 Oct. 2004, ISSN: 00220418. DOI: 10.1108/00220410410560573

[71] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management: an International Journal*, vol. 24, pp. 513–523, 5 Aug. 1988, ISSN: 03064573. DOI: 10.1016/0306-4573(88)90021-0 [Online]. Available: https://dl.acm.org/doi/10.1016/0306-4573%5C%2888%5C%2990021-0

[72] R. Ju, P. Zhou, C. H. Li, and L. Liu, "An efficient method for document categorization based on word2vec and latent semantic analysis," *EEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pp. 2276–2283, 2015. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=%5C&arnumber=7363382

[73] FIRTH and J. R., "A synopsis of linguistic theory, 1930-1955," *Studies in Linguistic Analysis*, 1957. [Online]. Available: https://ci.nii.ac.jp/naid/10020680394

[74] N. Hartmann, E. Fonseca, C. Shulby, M. Treviso, J. Rodrigues, and S. Aluisio, "Portuguese word embeddings: Evaluating on word analogies and natural language tasks," Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.06025

[75] Z. Yao, Y. Sun, W. Ding, N. Rao, and H. Xiong, "Dynamic word embeddings for evolving semantic discovery," in *WSDM 2018 - Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, vol. 2018-Febua, Association for Computing Machinery, Inc, Feb. 2018, pp. 673–681, ISBN: 9781450355810. DOI: 10.1145/3159652.3159703 [Online]. Available: http://arxiv.org/abs/1703.00607%20http://dx.doi.org/10.1145/3159652.3159703

[76] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 4 Apr. 1980, ISSN: 03401200. DOI: 10.1007/BF00344251 [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/7370364/

[77] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2323, 11 1998, ISSN: 00189219. DOI: 10.1109/5.726791

[78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, 6 Jun. 2017, ISSN: 15577317. DOI: 10.1145/3065386 [Online]. Available: http://code.google.com/p/cuda-convnet/.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Sep. 2014. [Online]. Available: https://arxiv.org/abs/1409.1556v6

[80] Z. Zhong, L. Jin, and Z. Xie, "High performance offline handwritten chinese character recognition using googlenet and directional feature maps," *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 2015-Novem, pp. 846–850, Nov. 2015. DOI: 10.1109/ICDAR.2015.7333881

[81] M. Lapata, P. Blunsom, and A. Koller, "Very deep convolutional networks for text classification," *the Association for Computational Linguistics*, vol. 1, pp. 3–07, 2017. [Online]. Available: http://eprints.whiterose.ac.uk/154414/

[82] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 2261–2269, Aug. 2016. DOI: 10.48550/arxiv.1608.06993 [Online]. Available: https://arxiv.org/abs/1608.06993v5

[83] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T. Y. Lin, J. Shlens, and B. Zoph, "Revisiting resnets: Improved training and scaling strategies," *Advances in Neural Information Processing Systems*, vol. 27, pp. 22 614–22 627, Mar. 2021, ISSN: 10495258. DOI: 10.48550/arxiv.2103.07579 [Online]. Available: https://arxiv.org/abs/2103.07579v1

[84] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, May 2019, pp. 6105–6114. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

[85] C. Gan, L. Wang, Z. Zhang, and Z. Wang, "Sparse attention based separable dilated convolutional neural network for targeted sentiment analysis," *Knowledge-Based Systems*, vol. 188, p. 104 827, Jan. 2020, ISSN: 09507051. DOI: 10.1016/j.knosys.2019.06.035

[86] O. Melamud, J. Goldberger, and I. Dagan, "Context2vec: Learning generic context embedding with bidirectional lstm," *CoNLL 2016 - 20th SIGNLL Conference on Computational Natural Language Learning, Proceedings*, pp. 51–61, 2016. DOI: 10.18653/V1/K16-1006 [Online]. Available: https://aclanthology.org/K16-1006/

[87] B. McCann, J. Bradbury, C. Xiong, and R. Socher, "Learned in translation: Contextualized word vectors," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6295–6306, Aug. 2017, ISSN: 10495258. [Online]. Available: https://arxiv.org/pdf/1708.00107

[88] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 2227–2237, 2018. DOI: 10.18653/V1/N18-1202 [Online]. Available: https://aclanthology.org/N18-1202/

[89] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *Journal of the American College of Radiology*, vol. 17, pp. 637–638, 5 Jun. 2021, ISSN: 1558349X. DOI: 10.48550/arxiv.2106.11342 [Online]. Available: https://arxiv.org/abs/2106.11342v3

[90] W. James, *The Principles of Psychology | Project Gutenberg*. Henry Holt and Company, 1890. [Online]. Available: https://www.gutenberg.org/files/57628/57628-h/57628-h.htm

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 5999–6009, Jun. 2017, ISSN: 10495258. [Online]. Available: https://arxiv.org/abs/1706.03762v5

[92] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Sep. 2014. [Online]. Available: https://arxiv.org/pdf/1409.0473

[93] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, Aug. 2015. DOI: 10.18653/v1/d15-1166 [Online]. Available: https://arxiv.org/pdf/1508.04025

[94] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, Ł. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Jan. 2018. DOI: 10.48550/arxiv.1801.10198 [Online]. Available: https://arxiv.org/abs/1801.10198v1

[95] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[96] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

[97] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, and P. G. Allen, "Roberta: A robustly optimized bert pretraining approach," Jul. 2019. DOI: 10.48550/arxiv.1907.11692 [Online]. Available: https://arxiv.org/abs/1907.11692v1

[98] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in Neural Information Processing Systems*, vol. 32, Jun. 2019, ISSN: 10495258. DOI: 10.48550/arxiv.1906.08237 [Online]. Available: https://arxiv.org/abs/1906.08237v2

[99] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," Oct. 2019. DOI: 10.48550/arxiv.1910.01108 [Online]. Available: https://arxiv.org/abs/1910.01108v4

[100] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, and G. Research, "Albert: A lite bert for self-supervised learning of language representations," Sep. 2019. DOI: 10.48550/arxiv.1909.11942 [Online]. Available: https://arxiv.org/abs/1909.11942v6

[101] *Hugging face – the ai community building the future.* [Online]. Available: https://huggingface.co/

[102] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021.

[103] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9992–10 002, Mar. 2021, ISSN: 15505499. DOI: 10.1109/ICCV48922.2021.00986 [Online]. Available: https://arxiv.org/pdf/2103.14030

[104] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12346 LNCS, pp. 213–229, May 2020, ISSN: 16113349. DOI: 10.1007/978-3-030-58452-8_13 [Online]. Available: https://arxiv.org/pdf/2005.12872

[105] M. Dehghani, B. Mustafa, J. Djolonga, J. Heek, M. Minderer, M. Caron, A. Steiner, J. Puigcerver, R. Geirhos, I. Alabdulmohsin, A. Oliver, P. Padlewski, A. A. Gritsenko, M. Lucic, and N. Houlsby, "Patch n' pack: Navit, a vision transformer for any aspect ratio and resolution," *Advances in Neural Information Processing Systems*, vol. 36, Jul. 2023, ISSN: 10495258. [Online]. Available: https://arxiv.org/pdf/2307.06304

[106] S. Agrawal, S. Dutta, and B. K. Patra, "Sentiment analysis of short informal text by tuning bert - bi-lstm model," *EUROCON 2021 - 19th IEEE International Conference on Smart Technologies, Proceedings*, pp. 98–102, Jul. 2021. DOI: 10.1109/EUROCON52738.2021.9535535

[107] T. Zhang, X. Gong, and C. L. Chen, "Bmt-net: Broad multitask transformer network for sentiment analysis," *IEEE Transactions on Cybernetics*, vol. 52, pp. 6232–6243, 7 Jul. 2022, ISSN: 21682275. DOI: 10.1109/TCYB.2021.3050508

[108] F. Wu, Z. Shi, Z. Dong, C. Pang, and B. Zhang, "Sentiment analysis of online product reviews based on senbert-cnn," *Proceedings - International Conference on Machine Learning and Cybernetics*, vol. 2020-December, pp. 229–234, Dec. 2020, ISSN: 21601348. DOI: 10.1109/ICMLC51923.2020.9469551

[109] J. Baran and J. Kocon, "Linguistic knowledge application to neuro-symbolic transformers in sentiment analysis," *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 395–402, Nov. 2022. DOI: 10.1109/ICDMW58026.2022.00059 [Online]. Available: https://ieeexplore.ieee.org/document/10031186/

[110] *Wordnet*. [Online]. Available: https://wordnet.princeton.edu/

[111] S. Baccianella, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," [Online]. Available: http://wordnetcode.princeton.

[112] K. Cheng, Y. Yue, and Z. Song, "Sentiment classification based on part-of-speech and self-attention mechanism," *IEEE Access*, vol. 8, pp. 16 387–16 396, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2967103

[113] P. Li, P. Zhong, J. Zhang, and K. Mao, "Convolutional transformer with sentiment-aware attention for sentiment analysis," *Proceedings of the International Joint Conference on Neural Networks*, Jul. 2020. DOI: 10.1109/IJCNN48605.2020.9206796

[114] L. Zhang, Y. Lei, and Z. Wang, "Long-text sentiment analysis based on semantic graph," *2020 IEEE International Conference on Embedded Software and Systems, ICESS 2020*, Dec. 2020. DOI: 10.1109/ICESS49830.2020.9301570

[115] H. Huang, Y. Jin, and R. Rao, "Sentiment-aware transformer using joint training," *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, vol. 2020-November, pp. 1154–1160, Nov. 2020, ISSN: 10823409. DOI: 10.1109/ICTAI50040.2020.00175

[116] F. Wang, G. Liu, Z. Wang, and X. Wu, "Sentiment analysis of movie reviews based on pretraining and dual branch coding," *Proceedings of the 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021*, vol. 2, pp. 1051–1055, 2021. DOI: 10.1109/IDAACS53288.2021.9661049

[117] U. Naseem, I. Razzak, P. Eklund, and K. Musial, "Towards improved deep contextual embedding for the identification of irony and sarcasm," *Proceedings of the International Joint Conference on Neural Networks*, Jul. 2020. DOI: 10.1109/IJCNN48605.2020.9207237

[118] C. D. Manning, "Part-of-speech tagging from 97% to 100%: Is it time for some linguistics?" *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6608 LNCS, pp. 171–189, PART 1 2011, ISSN: 1611-3349. DOI: 10.1007/978-3-642-19400-9_14 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-19400-9_14

[119] A. Chiche and B. Yitagesu, "Part of speech tagging: A systematic review of deep learning and machine learning approaches," *Journal of Big Data 2022 9:1*, vol. 9, pp. 1–25, 1 Jan. 2022, ISSN: 2196-1115. DOI: 10.1186/S40537-022-00561-Y [Online]. Available: https://link.springer.com/articles/10.1186/s40537-022-00561-y%20https://link.springer.com/article/10.1186/s40537-022-00561-y

[120] U. Secretary-General, W. C. on Environment, and Development, *Report of the world commission on environment and development :: Note /: By the secretary-general.* Aug. 1987. [Online]. Available: https://digitallibrary.un.org/record/139811

[121] E. Strubell, A. Ganesh, and A. Mccallum, "Energy and policy considerations for deep learning in nlp," *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019. DOI: 10.18653/V1/P19-1355 [Online]. Available: https://aclanthology.org/P19-1355/

[122] H. H. X. B. J. Liu, "Attention-based resnet for chinese text sentiment classification," *Advances in Computer Science Research*, vol. 80, pp. 495–499, 2018. [Online]. Available: https://scholar.googleusercontent.com/scholar?q=cache:Ni0y2GhAX0gJ:scholar.google.com/+attention+based+resnet+for+chineese+text+sentiment+classification%5C&hl=en%5C&as_sdt=0,5

[123] D. Jiang and J. He, "Text semantic classification of long discourses based on neural networks with improved focal loss," *Computational Intelligence and Neuroscience*, vol. 2021, 2021, ISSN: 16875273. DOI: 10.1155/2021/8845362

[124] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 9726–9735, Nov. 2019, ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00975 [Online]. Available: https://arxiv.org/pdf/1911.05722

[125] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *37th International Conference on Machine Learning, ICML 2020*, vol. PartF168147-3, pp. 1575–1585, Feb. 2020. [Online]. Available: https://arxiv.org/pdf/2002.05709

[126] *Clusterwiki*. [Online]. Available: https://uhhpc.herts.ac.uk/wiki/index.php?title=Main_Page

[127] *Bert — transformers 4.7.0 documentation*. [Online]. Available: https://huggingface.co/transformers/v4.8.2/model_doc/bert.html?highlight=berttokenizer%5C#berttokenizer

[128] *Scikit-learn · pypi*. [Online]. Available: https://pypi.org/project/scikit-learn/

[129] *Re — regular expression operations — python 3.14.0 documentation*. [Online]. Available: https://docs.python.org/3/library/re.html

[130] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, Jun. 2011, pp. 142–150. [Online]. Available: https://aclanthology.org/P11-1015.pdf

[131] *Nltk · pypi*. [Online]. Available: https://pypi.org/project/nltk/

[132] A. Aggarwal, S. Sharma, K. Singh, H. Singh, and S. Kumar, "A new approach for effective retrieval and indexing of medical images," *Biomedical Signal Processing and Control*, vol. 50, pp. 10–34, Apr. 2019, ISSN: 1746-8094. DOI: 10.1016/J.BSPC.2019.01.009

[133] W. Zhiqiang and L. Jun, "A review of object detection based on convolutional neural network," *Chinese Control Conference, CCC*, pp. 11 104–11 109, Sep. 2017, ISSN: 21612927. DOI: 10.23919/CHICC.2017.8029130

[134] A. A. Sodemann, M. P. Ross, and B. J. Borghetti, "A review of anomaly detection in automated surveillance," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, pp. 1257–1272, 6 2012, ISSN: 10946977. DOI: 10.1109/TSMCC.2012.2215319

[135] T. AI, *Getting meaning from text: Self-attention step-by-step video | towards ai*, Sep. 2020. [Online]. Available: https://towardsai.net/p/nlp/getting-meaning-from-text-self-attention-step-by-step-video

[136] H. Singh, N. Helian, R. Adams, and Y. Sun, "Sentiment analysis using blstm-resnet on textual images," *International Joint Conference on Neural network (IJCNN)*, pp. 1–8, Sep. 2022. DOI: 10.1109/IJCNN55064.2022.9892883

[137] DemšarJanez, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, Dec. 2006. DOI: 10.5555/1248547.1248548 [Online]. Available: https://dl.acm.org/doi/pdf/10.5555/1248547.1248548

[138] Z. Zhang, T. Zhou, Y. Zhang, and Y. Pang, "Attention-based deep residual learning network for entity relation extraction in chinese emrs," *BMC Medical Informatics and Decision Making*, vol. 19, pp. 171–177, 2 Apr. 2019, ISSN: 14726947. DOI: 10.1186/S12911-019-0769-0/TABLES/6 [Online]. Available: https://0-bmcmedinformdecismak-biomedcentral-com.brum.beds.ac.uk/articles/10.1186/s12911-019-0769-0

[139] H. Singh, N. Helian, R. Adams, and Y. Sun, "Using bert to generate contextualised textual images for sentiment analysis," pp. 234–243, 2025, ISSN: 1611-3349. DOI: 10.1007/978-3-031-84353-2_20 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-84353-2_20

[140] C. Junliang, "Cnn or rnn: Review and experimental comparison on image classification," *2022 IEEE 8th International Conference on Computer and Communications, ICCC 2022*, pp. 1939–1944, 2022. DOI: 10.1109/ICCC56324.2022.10065984

[141] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Association for Computational Linguistics, 2012, pp. 90–94. [Online]. Available: https://aclanthology.org/P12-2018/

[142] T.-J. Fu and W.-Y. Ma, "Speed reading: Learning to read forbackward via shuttle," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2018, pp. 4439–4448. DOI: 10.18653/v1/D18-1474 [Online]. Available: https://aclanthology.org/D18-1474/

[143] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031 [Online]. Available: https://aclanthology.org/P18-1031/

[144] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" In *Chinese Computational Linguistics*, Springer, 2019, pp. 194–206. DOI: 10.1007/978-3-030-32381-3_16 [Online]. Available: https://arxiv.org/abs/1905.05583

[145] C. E. Benarab, "Classifying textual data with pre-trained vision models through transfer learning and data transformations," *ArXiv*, vol. abs/2106.12479, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:235606043

[146] C. Zou, J. Luo, and C. Huang, "End to end speech recognition based on resnet-blstm," *Computer Life (CPL)*, 2020.

[147] K. L. Tan, C. P. Lee, K. M. Lim, and K. S. M. Anbananthen, "Sentiment analysis with ensemble hybrid deep learning model," *IEEE Access*, vol. 10, pp. 103 694–103 704, 2022. DOI: 10.1109/ACCESS.2022.3210182

[148] D. Salomon, "The ascii code," *Data Compression*, pp. 301–303, 1998. DOI: 10.1007/978-1-4757-2939-9_6 [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4757-2939-9_6

[149] M. Cai, "Sentiment analysis of tweets using deep neural architectures," in *32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018, pp. 1–8. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15786247.pdf

[150] *Tfidfvectorizer — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[151] *Truncatedsvd — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

[152] *Latentdirichletallocation — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html

[153] *Gridsearchcv — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[154] *Gaussiannb — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[155] *Logisticregression — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[156] *Svc — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[157] *Gradientboostingclassifier — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[158] *Mlpclassifier — scikit-learn 1.7.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[159] *Text preprocessing*. [Online]. Available: https://keras.io/api/layers/preprocessing_layers/text/

[160] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, B. Ch, and J. ¨. U. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, Jan. 2016. [Online]. Available: https://www.academia.edu/34357424/LSTM_A_Search_Space_Odyssey

[161] *Lstm cell layer*. [Online]. Available: https://keras.io/api/layers/recurrent_layers/lstm_cell/

[162] *Bidirectional layer*. [Online]. Available: https://keras.io/api/layers/recurrent_layers/bidirectional/

[163] S. Gullipalli, S. Grandhi, S. Nimmaka, A. Proffessor, and B. T. Student, "Classification of epistles using distant supervision," *International Research Journal of Engineering and Technology*, vol. 4, pp. 723–729, 4 Apr. 2017, ISSN: 2395-0056. [Online]. Available: http://tinyurl.com/cvvg9a

[164] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181

[165] E. Hernandez and J. Andreas, "The low-dimensional linear geometry of contextualized word representations," in *Proceedings of the 25th Conference on Computational Natural Language Learning*, Association for Computational Linguistics, 2021, pp. 82–93. DOI: 10.18653/v1/2021.conll-1.7

Chapter 9

Appendix A

9.1 Cross-Validation

It will be a methodological mistake to learn the parameter of a function and then test it on the same data. Such a system will give a perfect score but will not predict properly on unseen data i.e., it will not generalize well. It is known as overfitting. To avoid these situations and to make the model generalise well, resampling techniques like Cross-Validation is used.

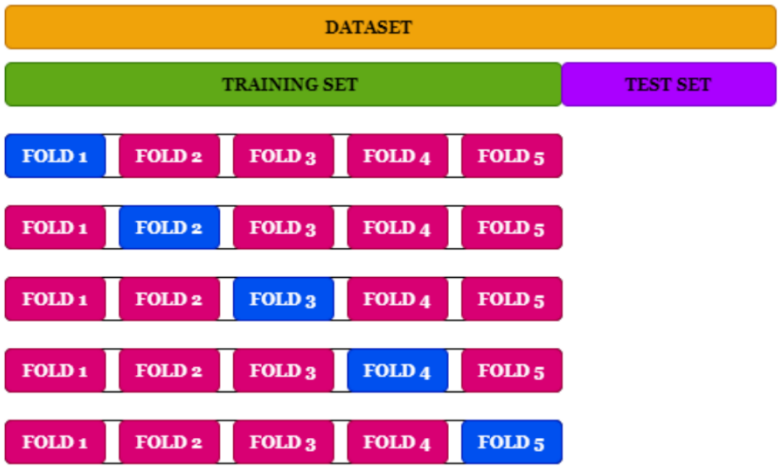


Figure 9.1: Example CV with 5 folds - bifurcating training set in to 5 equal sections

Usually, data is split into training, validation, and test sets but this method is only good for a dataset that is evenly distributed, meaning that an equal proportion of classes are present in all three sets and classes in each set is evenly representing all scenarios. But for a dataset with a varied count of classes splitting the dataset can be challenging. The concept of validation set was brought so that the model does not optimise its parameter every time it is used on the test set for performance. This way, the information about the test set gets leaked.

A prominent cross-validation technique is K-fold CV which divides the training set into K section where K is a non-zero integer value. Every subsection of the training set is called a fold and each fold appears as the part of the training set K-1 times. This method makes a model learn all the underlying data distribution as all the data get a chance to appear while the training. Figure 9.1 demonstrates a complete 5- fold CV process.

Methodology

- First, the dataset is divided into the training set and the test set. A popular package Scikit-learn is used for the same. The function used is train_test_split.

Example-

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.1, random_state=42)

Where X_train and X_test is training set and test set respectively, similarly y_train and y_test are corresponding labels. The parameter of the function X represents dataset, y is for labels, test_size gives the ratio of the test set

concerning the whole dataset and random_state value is used for dividing the dataset in a fixed order. Generally, the random_state value is used for reproducing the same split every time you run the program for the constituency.

- After splitting the dataset. The k-fold CV is performed on the training set. The package used is "KFold" from sklearn.model_selection

Example –

KF = KFold(n_splits=3, random_state=None, shuffle =false)

for train, validate in KF.split(X_train):

X_training, X_validation = X_train[train], X_train[validate]

y_training, y_validation = y_train[train], y_train[validate]

KF instance is initialised with 3 n_splits which means 3-fold CV. A random_state can also be provided for reproducing the same in future. KF.split generate 2 lists of indexes for 3 different folds. These are stored in variable train and validate. For every train and validate index pair, the training and validation set is chosen.

Output of various index pairs with KF.split(X) where X_train has 15 data samples.

KFold(n_splits=3)

fold – 1, train- [5, 6, 7, 8, 9, 10, 11, 12, 13, 14], validate [0,1,2,3,4]

fold – 2, train- [0, 1, 2, 3, 4, 10, 11, 12, 13, 14], validate [5,6,7,8,9]

fold – 3, train- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], validate [10,11,12,13,14]

KFold(n_splits=3, shuffle=True) - Every time we run this a new shuffled a different value pair is generated

fold – 1, train- [1, 2, 3, 5, 6, 7, 8, 9, 10, 14], validate [0,4,11,12,13]

fold – 2, train- [0, 3, 4, 5, 7, 9, 10, 11, 12, 13], validate [1,2,6,8,14]

fold – 3, train- [0, 1, 2, 4, 6, 8, 11, 12, 13, 14], validate [3,5,7,9,10]

KFold(n_splits=3, random_state = 42, shuffle=True) will generated a different index pair with a different seed value. But every seed will always reproduce same result. (to remove order in the dataset shuffle is used)

fold – 1, train- [1, 2, 3, 4, 6, 7, 8, 10, 12, 14], validate [0,5,9,11,13]

fold – 2, train- [0, 3, 5, 6, 7, 9, 10, 11, 12, 13], validate [1,2,4,8,14]

fold – 3, train- [0, 1, 2, 4, 5, 8, 9, 11, 13, 14], validate [3,6,7,10,12]

After training the model on the training set, the model is evaluated with the corresponding validation set. This is done for every fold and the performance parameter of each training is recorded. The final score is averaged over the performance of all individual pieces of training.

For comparing different architectures or to produce the best model, each model architecture will perform the cross-validation and compare their average performance score. The model architecture with best will finally be trained with the complete training set and the model is ready to be tested on the test set or to predict on unseen data.

Below is an algorithm, demonstrating the procedure that we follow to find the best architecture.

Step 1. Randomly divide the whole dataset into a Training and a Testing set, if these sets are not separately provided.

Step 2. Select one model from all the model architectures (in our case various depths 3,6,9,12 etc).

Step 3. Apply a k-fold cv on the training set.

Step 4. Use the nth sub-dataset as validation dataset and the rest of k-1 sub-datasets as

CV-training set.

Step 5. Train the model on the CV-training set samples.

Step 6. Evaluate the model using the CV-validation set and generate the confusion metrics and save the best

model weights.

Step 7. Repeat Steps 3 to 6 for all the values of K.

Step 8. calculate the average of evaluation metrics for the K loops for the current model.

Step 9. Repeat Step2 to 8 for all the architectures.

Step 10. Find the best model by comparing the CV performance of various models.

Step 11. Use the best model to train again on the original train set.

Step 12. Use the trained model from the previous step to evaluate the Test set.

Step 13. Repeat from steps 1 to 12 for desired times to get the average results from step 12.

9.1.1 Training Strategy

While training, the validation accuracy is tracked with the ModeCheckpoint method in Keras which is set to monitor the validation accuracy and saves the model weights whenever the validation accuracy is increased from the previous best, thus replacing the best old weights. These best model weights are loaded, at the end of every training, into the model before the model is evaluated on the test set (validation sets in our case are generated using 5-fold cv).

BLSTM-2DCNN is treated as a network with zero residual blocks and as a baseline to compare the other models. Along with the Model Checkpoint method, an additional function named EarlyStopping is utilised for all BLSTM-2DCNN training. This method monitors the validation loss for 25 epochs after every lowest validation loss value is obtained. 25 is also a patience value. After the patience epochs, if the validation loss is still not improved, then the model will stop training automatically and the best model weights are restored. The optimiser used in these experiments is Adam (adaptive momentum). Whereas, Adagrad optimiser (Adaptive gradient) is used in the original BLSTM-2DCNN paper[27]. Both these optimisers are used for adaptive learning rate depending on the history of gradient, but Adam solves the denominator decay problem of the Adagrad thus helping the model to converge quickly.

9.1.2 Python development

Python packages used

1. Keras
2. Pytorch
3. NumPy
4. NLTK
5. Sklearn
6. Matplotlib
7. Transformer
8. Regular expression
9. CV2
10. pandas

9.2 How RNN process information

The sequence of data is fed to an RNN one by one. A hidden state is generated at every sequence which acts like a memory and combines with the next input in the sequence. This is how an RNN influences the future sequence based on the knowledge gained from previous sequences. The use of a hidden state from previous data with the new data is like feedback and this step is repeated till the end of the sequence. If we unfold a basic RNN, it looks like an ordinary feedforward neural network.

The Figure 9.2 illustrates a Recurrent Neural Network (RNN) in its unrolled form, showing how information flows through time. x (blue ovals) represents the input at time t, h_t (red rectangles) is the hidden state carrying information from previous steps, and $\overrightarrow{y}$ (green ovals) is the predicted output. The recurrent connections allow h_t

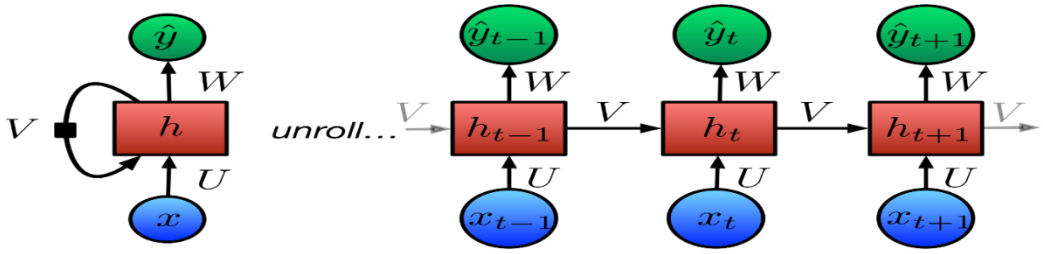


Figure 9.2: Unrolled RNN like a Feedforward network

to depend on both x_t and h_{t-1}, enabling sequential data processing, making RNNs suitable for tasks like speech recognition, NLP, and time-series forecasting.

Figure 9.3 represents the use of RNNs in a variety of classification problems when used in different configurations. Pink blocks are input, blue blocks are output, and green blocks are RNN units. In this research, RNNs have been used in a configuration that is equivalent to the video classification example shown.

In short, for every input in a sequence, a corresponding output will be obtained. This is because, the sequence of text in a document will be fed to RNN to obtain an image, where every row of image pixel represents a word, and all the rows combined represent a complete text document. Here, the text document is a collection of words, for example, the collection of words in a single tweet.

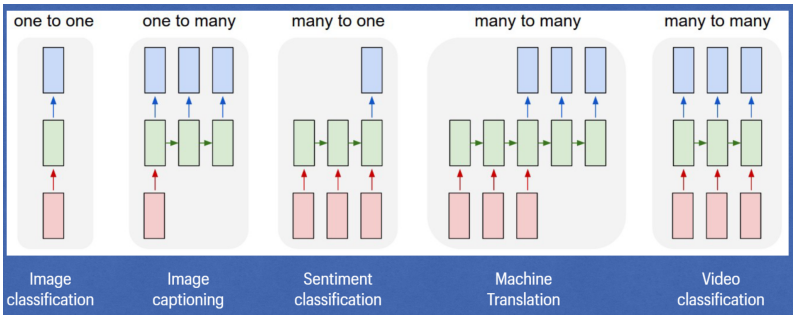


Figure 9.3: Different configurations in which RNNs are used

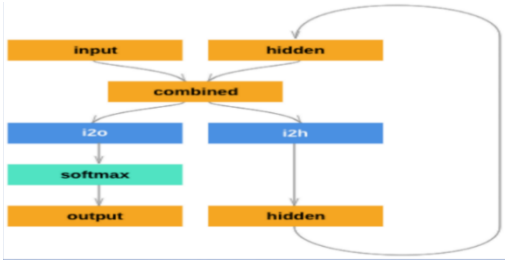


Figure 9.4: Internal operations inside a simple RNN

The internal operation at every time step is quite straightforward. Both Input signals and the hidden states are combined (concatenated) and passed into conventional linear layers (i2o and i2h in

Figure 9.4). The output of the i2o layer is subjected to SoftMax operation to obtain the probabilities of each output class whereas the output of the i2h layer replaces the internal hidden state of the RNN unit.

Figure 9.5 is a demonstration of operation i2o with inputs and outputs. First, the input signal is concatenated with the hidden unit and then passed to a linear layer with nonlinear activation. The output is equal to the different classes. The SoftMax function takes the outputs corresponding to different classes and the class with maximum probability becomes the output of the RNN.

Figure 9.6 represents the i2h operation which takes the same combined input as i2o, and the output tensor is equal to the length of hidden units. This output tensor becomes the new value for the Hidden unit and replaces the previous value and will be combined with the next input sequence at the next time step.

Like any other neural network, Multiple RNNs can be stacked on top of each other to form a multilayer RNN network. As shown in Figure 9.7, At each time step, all the RNNs will produce output and each preceding RNN layer takes the previous RNN output as input. All the RNN layer will have their own set of hidden states.

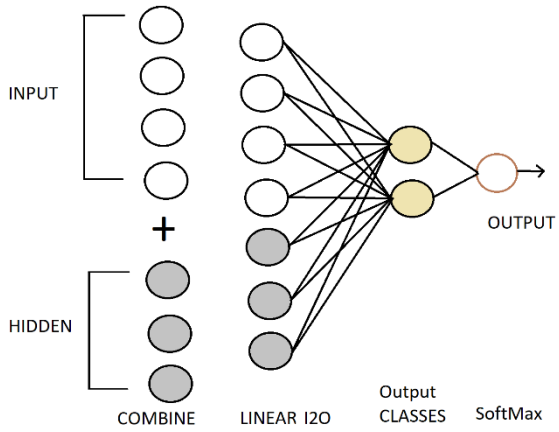


Figure 9.5: I2o-Input to Output Linear layer inside RNN

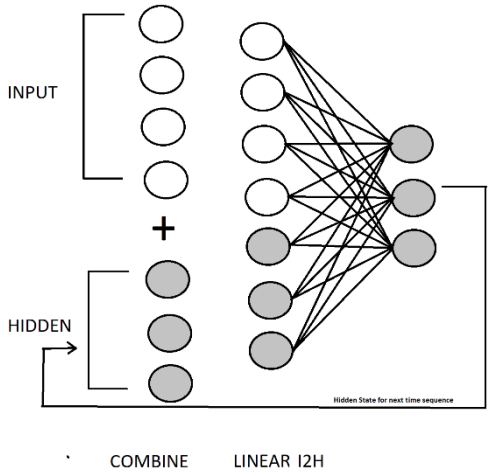


Figure 9.6: I2H-Input to Hidden state Linear layer inside RNN

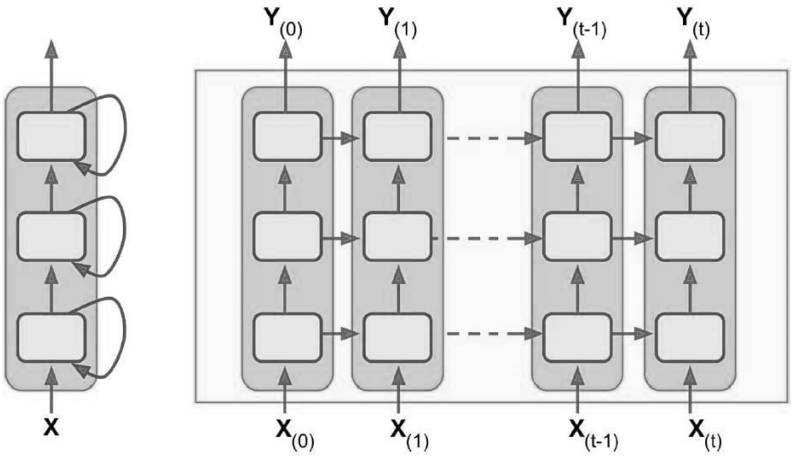


Figure 9.7: Multi-layer RNNs

Chapter 10

Appendix B

10.1 Individual runs results for BLSTM-ResNet

10.1.1 Average Accuracies (%) and Standard Deviations for BLSTM-ResNet variants at Various Dataset size for sentiment140 dataset.

Table 10.1: Combined Average accuracy result with std for all the experiment in this Section 4. Some models have N/A for some datapoints as these models were not tested for these data points in any experiment.

Model	20,000	40,000	60,000	80,000	100,000	200,000
BLSTM-2DCNN	73.54 ± 0.51	75.69 ± 0.31	77.75 ± 0.44	75.84 ± 0.30	76.03 ± 0.22	78.59 ± 0.92
BLSTM-ResNet-3	72.67 ± 0.92	74.77 ± 0.23	76.97 ± 0.49	77.58 ± 0.26	78.49 ± 0.27	79.37 ± 0.24
BLSTM-ResNet-6	72.90 ± 0.31	75.23 ± 0.27	77.26 ± 0.39	78.06 ± 0.24	78.67 ± 0.24	79.78 ± 0.12
BLSTM-ResNet-9	72.07 ± 1.09	75.24 ± 0.31	77.25 ± 0.56	78.09 ± 0.39	78.79 ± 0.27	79.85 ± 0.22
BLSTM-ResNet-12	72.67 ± 0.82	75.72 ± 0.56	77.44 ± 0.23	78.80 ± 0.33	78.64 ± 0.24	79.93 ± 0.14
BLSTM-ResNet-15	N/A	N/A	N/A	78.33 ± 0.22	N/A	79.88 ± 0.17
BLSTM-ResNet-18	N/A	N/A	N/A	78.42 ± 0.38	N/A	79.84 ± 0.21
BLSTM-ResNet-21	N/A	N/A	N/A	78.39 ± 0.25	N/A	79.82 ± 0.15
BLSTM-ResNet-27	72.52 ± 0.32	76.13 ± 0.58	77.70 ± 0.47	78.01 ± 0.62	78.73 ± 0.33	79.74 ± 0.24
BLSTM-ResNet-36	73.40 ± 0.58	75.91 ± 0.91	77.82 ± 0.21	78.53 ± 0.22	78.60 ± 0.34	79.50 ± 0.27

10.1.2 Weighted F1 Scores (%) and Standard Deviations for BLSTM-ResNet variants at Various Dataset size for sentiment140 dataset.

Table 10.2: Combined Average weighted F1 result with std for all the experiment in this Section 4. Some models have N/A for some datapoints as these models were not tested for these data points in any experiment.

Model	20,000	40,000	60,000	80,000	100,000	200,000
BLSTM-2DCNN	73.52 ± 0.50	75.67 ± 0.31	77.74 ± 0.45	75.83 ± 0.31	76.03 ± 0.22	78.59 ± 0.92
BLSTM-ResNet-3	72.66 ± 0.92	74.76 ± 0.23	76.97 ± 0.49	77.57 ± 0.26	78.49 ± 0.27	79.37 ± 0.24
BLSTM-ResNet-6	72.89 ± 0.31	75.22 ± 0.27	77.26 ± 0.40	78.06 ± 0.24	78.67 ± 0.23	79.78 ± 0.12
BLSTM-ResNet-9	71.99 ± 1.18	75.24 ± 0.31	77.25 ± 0.56	78.08 ± 0.39	78.79 ± 0.27	79.85 ± 0.22
BLSTM-ResNet-12	72.65 ± 0.82	75.71 ± 0.56	77.44 ± 0.23	78.80 ± 0.33	78.64 ± 0.24	79.93 ± 0.14

Continued on next page

Model	20,000	40,000	60,000	80,000	100,000	200,000
BLSTM-ResNet-15	-	-	-	78.33 ± 0.22	-	79.87 ± 0.17
BLSTM-ResNet-18	-	-	-	78.42 ± 0.37	-	79.84 ± 0.21
BLSTM-ResNet-21	-	-	-	78.39 ± 0.25	-	79.82 ± 0.15
BLSTM-ResNet-27	72.50 ± 0.32	76.12 ± 0.58	77.70 ± 0.47	78.01 ± 0.62	78.72 ± 0.33	79.74 ± 0.24
BLSTM-ResNet-36	73.35 ± 0.61	75.90 ± 0.90	77.81 ± 0.21	78.53 ± 0.22	78.60 ± 0.34	79.50 ± 0.27

10.1.3 Individual experiment run results for BLSTM-ResNet variant carried out in Section 4 on Sentiment140 dataset.

BLSTM-2DCN

Table 10.3: Individual test set performance of BLSTM-2DCNN model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-2DCNN				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(798, 225), (288, 689)	0.7435	0.7430
2	20000	(778, 245), (285, 692)	0.7350	0.7347
3	20000	(720, 303), (221, 756)	0.7380	0.7379
4	20000	(774, 249), (288, 689)	0.7315	0.7312
5	20000	(733, 290), (252, 725)	0.7290	0.7290
1	40000	(1571, 434), (529, 1466)	0.7592	0.7591
2	40000	(1449, 556), (406, 1589)	0.7595	0.7593
3	40000	(1549, 456), (540, 1455)	0.7510	0.7509
4	40000	(1562, 443), (528, 1467)	0.7572	0.7572
5	40000	(1434, 571), (399, 1596)	0.7575	0.7572
1	60000	(2193, 801), (586, 2420)	0.7688	0.7686
2	60000	(2467, 527), (803, 2203)	0.7783	0.7780
3	60000	(2307, 687), (633, 2373)	0.7800	0.7800
4	60000	(2282, 712), (609, 2397)	0.7798	0.7798
5	60000	(2302, 692), (624, 2382)	0.7807	0.7806
1	80000	(3114, 861), (1049, 2976)	0.7612	0.7612
2	80000	(3115, 860), (1076, 2949)	0.7580	0.7579
3	80000	(3122, 853), (1052, 2973)	0.7619	0.7618
4	80000	(3219, 756), (1183, 2842)	0.7576	0.7573
5	80000	(2873, 1102), (871, 3154)	0.7534	0.7532
1	100000	(3934, 1127), (1288, 3651)	0.7585	0.7584
2	100000	(3826, 1235), (1174, 3765)	0.7591	0.7591
3	100000	(3881, 1180), (1202, 3737)	0.7618	0.7618
4	100000	(3719, 1342), (1076, 3863)	0.7582	0.7582

Continued on next page

BLSTM-2DCNN				
5	100000	(3939, 1122), (1239, 3700)	0.7639	0.7638
1	200000	(7662, 2299), (2348, 7691)	0.7677	0.7676
2	200000	(8002, 1959), (2202, 7837)	0.7920	0.7919
3	200000	(8036, 1925), (2251, 7788)	0.7912	0.7912
4	200000	(8058, 1903), (2321, 7718)	0.7888	0.7888
5	200000	(7876, 2085), (2117, 7922)	0.7899	0.7899

BLSTM-ResNet-3

Table 10.4: Individual test set performance of BLSTM-ResNet-3 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-3				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(741, 282), (277, 700)	0.7205	0.7204
2	20000	(756, 267), (272, 705)	0.7305	0.7304
3	20000	(769, 254), (262, 715)	0.7420	0.7419
4	20000	(754, 269), (280, 697)	0.7255	0.7254
5	20000	(739, 284), (286, 691)	0.7150	0.7149
1	40000	(1514, 491), (507, 1488)	0.7505	0.7505
2	40000	(1540, 465), (554, 1441)	0.7452	0.7451
3	40000	(1480, 525), (496, 1499)	0.7448	0.7447
4	40000	(1524, 481), (523, 1472)	0.7490	0.7490
5	40000	(1539, 466), (539, 1456)	0.7488	0.7487
1	60000	(2319, 675), (714, 2292)	0.7685	0.7685
2	60000	(2278, 716), (681, 2325)	0.7672	0.7672
3	60000	(2330, 664), (664, 2342)	0.7787	0.7787
4	60000	(2327, 667), (713, 2293)	0.7700	0.7700
5	60000	(2240, 754), (660, 2346)	0.7643	0.7643
1	80000	(3109, 866), (963, 3062)	0.7714	0.7714
2	80000	(3158, 817), (946, 3079)	0.7796	0.7796
3	80000	(3184, 791), (1001, 3024)	0.7760	0.7759
4	80000	(3102, 873), (917, 3108)	0.7762	0.7762
5	80000	(3143, 832), (963, 3062)	0.7756	0.7756
1	100000	(3924, 1137), (1014, 3925)	0.7849	0.7849
2	100000	(4028, 1033), (1101, 3838)	0.7866	0.7865
3	100000	(4008, 1053), (1093, 3846)	0.7854	0.7854
4	100000	(4008, 1053), (1069, 3870)	0.7878	0.7878
5	100000	(3926, 1135), (1065, 3874)	0.7800	0.7800

Continued on next page

BLSTM-ResNet-3				
1	200000	(7902, 2059), (2022, 8017)	0.7960	0.7959
2	200000	(7966, 1995), (2127, 7912)	0.7939	0.7939
3	200000	(7973, 1988), (2079, 7960)	0.7966	0.7966
4	200000	(7956, 2005), (2162, 7877)	0.7916	0.7916
5	200000	(7897, 2064), (2125, 7914)	0.7905	0.7905

BLSTM-ResNet-6

Table 10.5: Individual test set performance of BLSTM-ResNet-6 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-6				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(742, 281), (269, 708)	0.7250	0.7250
2	20000	(756, 267), (277, 700)	0.7280	0.7279
3	20000	(766, 257), (280, 697)	0.7315	0.7313
4	20000	(747, 276), (257, 720)	0.7335	0.7335
5	20000	(742, 281), (265, 712)	0.7270	0.7270
1	40000	(1562, 443), (540, 1455)	0.7542	0.7541
2	40000	(1534, 471), (504, 1491)	0.7562	0.7562
3	40000	(1543, 462), (539, 1456)	0.7498	0.7497
4	40000	(1556, 449), (542, 1453)	0.7522	0.7521
5	40000	(1547, 458), (546, 1449)	0.7490	0.7489
1	60000	(2277, 717), (635, 2371)	0.7747	0.7746
2	60000	(2333, 661), (668, 2338)	0.7785	0.7785
3	60000	(2278, 716), (677, 2329)	0.7678	0.7678
4	60000	(2308, 686), (674, 2332)	0.7733	0.7733
5	60000	(2256, 738), (650, 2356)	0.7687	0.7686
1	80000	(3080, 895), (862, 3163)	0.7804	0.7804
2	80000	(3094, 881), (908, 3117)	0.7764	0.7764
3	80000	(3125, 850), (906, 3119)	0.7805	0.7805
4	80000	(3174, 801), (931, 3094)	0.7835	0.7835
5	80000	(3156, 819), (923, 3102)	0.7823	0.7822
1	100000	(4000, 1061), (1028, 3911)	0.7911	0.7911
2	100000	(3964, 1097), (1045, 3894)	0.7858	0.7858
3	100000	(3929, 1132), (1027, 3912)	0.7841	0.7841
4	100000	(3992, 1069), (1062, 3877)	0.7869	0.7869
5	100000	(3987, 1074), (1069, 3870)	0.7857	0.7857
1	200000	(8139, 1822), (2247, 7792)	0.7965	0.7965

Continued on next page

BLSTM-ResNet-6				
2	200000	(8051, 1910), (2126, 7913)	0.7982	0.7982
3	200000	(7954, 2007), (2049, 7990)	0.7972	0.7972
4	200000	(8033, 1928), (2076, 7963)	0.7998	0.7998
5	200000	(7954, 2007), (2051, 7988)	0.7971	0.7971

BLSTM-ResNet-9

Table 10.6: Individual test set performance of BLSTM-ResNet-9 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-9				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(830, 193), (400, 577)	0.7035	0.7007
2	20000	(769, 254), (300, 677)	0.7230	0.7226
3	20000	(748, 275), (298, 679)	0.7135	0.7133
4	20000	(758, 265), (273, 704)	0.7310	0.7309
5	20000	(694, 329), (206, 771)	0.7325	0.7322
1	40000	(1532, 473), (520, 1475)	0.7518	0.7517
2	40000	(1558, 447), (526, 1469)	0.7568	0.7567
3	40000	(1537, 468), (514, 1481)	0.7545	0.7545
4	40000	(1527, 478), (531, 1464)	0.7478	0.7477
5	40000	(1540, 465), (530, 1465)	0.7512	0.7512
1	60000	(2310, 684), (702, 2304)	0.7690	0.7690
2	60000	(2233, 761), (637, 2369)	0.7670	0.7669
3	60000	(2309, 685), (681, 2325)	0.7723	0.7723
4	60000	(2272, 722), (652, 2354)	0.7710	0.7710
5	60000	(2289, 705), (596, 2410)	0.7832	0.7831
1	80000	(3116, 859), (904, 3121)	0.7796	0.7796
2	80000	(3107, 868), (915, 3110)	0.7771	0.7771
3	80000	(3255, 720), (1007, 3018)	0.7841	0.7840
4	80000	(3117, 858), (928, 3097)	0.7768	0.7767
5	80000	(3215, 760), (946, 3079)	0.7867	0.7867
1	100000	(3987, 1074), (1004, 3935)	0.7922	0.7922
2	100000	(3940, 1121), (980, 3959)	0.7899	0.7899
3	100000	(3922, 1139), (998, 3941)	0.7863	0.7863
4	100000	(3946, 1115), (1030, 3909)	0.7855	0.7855
5	100000	(3991, 1070), (1075, 3864)	0.7855	0.7855
1	200000	(8111, 1850), (2126, 7913)	0.8012	0.8012
2	200000	(7873, 2088), (2022, 8017)	0.7945	0.7945

Continued on next page

BLSTM-ResNet-9				
3	200000	(8058, 1903), (2110, 7929)	0.7994	0.7993
4	200000	(7989, 1972), (2045, 7994)	0.7992	0.7991
5	200000	(8017, 1944), (2087, 7952)	0.7984	0.7984

BLSTM-ResNet-12

Table 10.7: Individual test set performance of BLSTM-ResNet-9 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-12				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(759, 264), (279, 698)	0.7285	0.7283
2	20000	(760, 263), (294, 683)	0.7215	0.7212
3	20000	(747, 276), (275, 702)	0.7245	0.7244
4	20000	(773, 250), (267, 710)	0.7415	0.7413
5	20000	(722, 301), (264, 713)	0.7175	0.7175
1	40000	(1497, 508), (450, 1545)	0.7605	0.7605
2	40000	(1581, 424), (521, 1474)	0.7638	0.7636
3	40000	(1456, 549), (430, 1565)	0.7552	0.7551
4	40000	(1521, 484), (481, 1514)	0.7588	0.7587
5	40000	(1550, 455), (555, 1440)	0.7475	0.7474
1	60000	(2334, 660), (677, 2329)	0.7772	0.7772
2	60000	(2296, 698), (649, 2357)	0.7755	0.7755
3	60000	(2381, 613), (735, 2271)	0.7753	0.7753
4	60000	(2333, 661), (717, 2289)	0.7703	0.7703
5	60000	(2310, 684), (674, 2332)	0.7737	0.7737
1	80000	(3136, 839), (886, 3139)	0.7844	0.7844
2	80000	(3156, 819), (872, 3153)	0.7886	0.7886
3	80000	(3175, 800), (910, 3115)	0.7863	0.7862
4	80000	(3213, 762), (886, 3139)	0.7940	0.7940
5	80000	(3164, 811), (895, 3130)	0.7867	0.7867
1	100000	(3915, 1146), (1005, 3934)	0.7849	0.7849
2	100000	(3977, 1084), (1038, 3901)	0.7878	0.7878
3	100000	(3874, 1187), (989, 3950)	0.7824	0.7824
4	100000	(3942, 1119), (1001, 3938)	0.7880	0.7880
5	100000	(3951, 1110), (1000, 3939)	0.7890	0.7890
1	200000	(8201, 1760), (2275, 7764)	0.7983	0.7982
2	200000	(8087, 1874), (2126, 7913)	0.8000	0.8000
3	200000	(8024, 1937), (2072, 7967)	0.7995	0.7995

Continued on next page

BLSTM-ResNet-12				
4	200000	(8079, 1882), (2169, 7870)	0.7974	0.7974
5	200000	(8063, 1898), (2072, 7967)	0.8015	0.8015

BLSTM-ResNet-15

Table 10.8: Individual test set performance of BLSTM-ResNet-15 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-15				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	80000	(3164, 811), (902, 3123)	0.7859	0.7859
2	80000	(3128, 847), (896, 3129)	0.7821	0.7821
3	80000	(3192, 783), (961, 3064)	0.7820	0.7820
4	80000	(3162, 813), (898, 3127)	0.7861	0.7861
5	80000	(3133, 842), (913, 3112)	0.7806	0.7806
1	200000	(8119, 1842), (2213, 7826)	0.7973	0.7972
2	200000	(8031, 1930), (2107, 7932)	0.7982	0.7981
3	200000	(8094, 1867), (2185, 7854)	0.7974	0.7974
4	200000	(8061, 1900), (2063, 7976)	0.8018	0.8018
5	200000	(8135, 1826), (2193, 7846)	0.7991	0.7990

BLSTM-ResNet-18

Table 10.9: Individual test set performance of BLSTM-ResNet-18 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-18				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	80000	(3143, 832), (951, 3074)	0.7771	0.7771
2	80000	(3013, 962), (763, 3262)	0.7844	0.7842
3	80000	(3213, 762), (935, 3090)	0.7879	0.7878
4	80000	(3216, 759), (958, 3067)	0.7854	0.7853
5	80000	(3170, 805), (904, 3121)	0.7864	0.7864
1	200000	(8175, 1786), (2289, 7750)	0.7963	0.7962
2	200000	(8111, 1850), (2168, 7871)	0.7991	0.7991
3	200000	(8191, 1770), (2190, 7849)	0.8020	0.8020
4	200000	(8091, 1870), (2200, 7839)	0.7965	0.7965
5	200000	(8292, 1669), (2369, 7670)	0.7981	0.7980

BLSTM-ResNet-21

Table 10.10: Individual test set performance of BLSTM-ResNet-21 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-21				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	80000	(3177, 798), (943, 3082)	0.7824	0.7824
2	80000	(3161, 814), (913, 3112)	0.7841	0.7841
3	80000	(3111, 864), (834, 3191)	0.7877	0.7877
4	80000	(3248, 727), (993, 3032)	0.7850	0.7849
5	80000	(3248, 727), (1030, 2995)	0.7804	0.7802
1	200000	(8125, 1836), (2228, 7811)	0.7968	0.7968
2	200000	(8090, 1871), (2183, 7856)	0.7973	0.7973
3	200000	(8116, 1845), (2133, 7906)	0.8011	0.8011
4	200000	(8072, 1889), (2143, 7896)	0.7984	0.7984
5	200000	(8159, 1802), (2246, 7793)	0.7976	0.7975

BLSTM-ResNet-27

Table 10.11: Individual test set performance of BLSTM-ResNet-27 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset.

BLSTM-ResNet-27				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(724, 299), (239, 738)	0.7310	0.7310
2	20000	(771, 252), (297, 680)	0.7255	0.7251
3	20000	(740, 283), (270, 707)	0.7235	0.7235
4	20000	(759, 264), (293, 684)	0.7215	0.7213
5	20000	(768, 255), (296, 681)	0.7245	0.7242
1	40000	(1509, 496), (486, 1509)	0.7545	0.7545
2	40000	(1603, 402), (552, 1443)	0.7615	0.7612
3	40000	(1550, 455), (473, 1522)	0.7680	0.7680
4	40000	(1564, 441), (489, 1506)	0.7675	0.7675
5	40000	(1532, 473), (507, 1488)	0.7550	0.7550
1	60000	(2418, 576), (763, 2243)	0.7768	0.7767
2	60000	(2342, 652), (653, 2353)	0.7825	0.7825
3	60000	(2259, 735), (655, 2351)	0.7683	0.7683
4	60000	(2337, 657), (673, 2333)	0.7783	0.7783
5	60000	(2327, 667), (658, 2348)	0.7792	0.7792
1	80000	(3133, 842), (873, 3152)	0.7856	0.7856
2	80000	(3002, 973), (878, 3147)	0.7686	0.7686

Continued on next page

BLSTM-ResNet-27				
3	80000	(3101, 874), (843, 3182)	0.7854	0.7854
4	80000	(3112, 863), (904, 3121)	0.7791	0.7791
5	80000	(3111, 864), (881, 3144)	0.7819	0.7819
1	100000	(3822, 1239), (933, 4006)	0.7828	0.7827
2	100000	(3923, 1138), (962, 3977)	0.7900	0.7900
3	100000	(3848, 1213), (949, 3990)	0.7838	0.7838
4	100000	(4014, 1047), (1045, 3894)	0.7908	0.7908
5	100000	(3969, 1092), (1019, 3920)	0.7889	0.7889
1	200000	(7983, 1978), (2052, 7987)	0.7985	0.7985
2	200000	(7973, 1988), (2029, 8010)	0.7992	0.7991
3	200000	(8078, 1883), (2258, 7781)	0.7930	0.7929
4	200000	(8108, 1853), (2155, 7884)	0.7996	0.7996
5	200000	(7969, 1992), (2070, 7969)	0.7969	0.7969

BLSTM-ResNet-36

Table 10.12: Individual test set performance of BLSTM-ResNet-36 model trained on different Cross Validation dataset for various subset sizes of Sentiment140 dataset

BLSTM-ResNet-36				
Exp	**Dataset Size**	**Confusion Matrix**	**Accuracy**	**Weighted F1-Score**
1	20000	(688, 335), (180, 797)	0.7425	0.7420
2	20000	(722, 301), (230, 747)	0.7345	0.7345
3	20000	(763, 260), (277, 700)	0.7315	0.7313
4	20000	(778, 245), (282, 695)	0.7365	0.7362
5	20000	(634, 389), (161, 816)	0.7250	0.7235
1	40000	(1530, 475), (468, 1527)	0.7642	0.7642
2	40000	(1626, 379), (526, 1469)	0.7738	0.7735
3	40000	(1532, 473), (500, 1495)	0.7568	0.7567
4	40000	(1508, 497), (510, 1485)	0.7482	0.7482
5	40000	(1503, 502), (488, 1507)	0.7525	0.7525
1	60000	(2437, 557), (761, 2245)	0.7803	0.7802
2	60000	(2338, 656), (672, 2334)	0.7787	0.7787
3	60000	(2297, 697), (634, 2372)	0.7782	0.7781
4	60000	(2320, 674), (649, 2357)	0.7795	0.7795
5	60000	(2359, 635), (720, 2286)	0.7742	0.7741
1	80000	(3134, 841), (907, 3118)	0.7815	0.7815
2	80000	(3142, 833), (883, 3142)	0.7855	0.7855
3	80000	(3148, 827), (865, 3160)	0.7885	0.7885

Continued on next page

BLSTM-ResNet-36				
4	80000	(3174, 801), (919, 3106)	0.7850	0.7850
5	80000	(3192, 783), (929, 3096)	0.7860	0.7860
1	100000	(3998, 1063), (1123, 3816)	0.7814	0.7813
2	100000	(4003, 1058), (1031, 3908)	0.7911	0.7911
3	100000	(3865, 1196), (927, 4012)	0.7877	0.7877
4	100000	(4043, 1018), (1118, 3821)	0.7864	0.7863
5	100000	(3959, 1102), (1064, 3875)	0.7834	0.7834
1	200000	(7936, 2025), (2091, 7948)	0.7942	0.7942
2	200000	(8097, 1864), (2158, 7881)	0.7989	0.7989
3	200000	(8055, 1906), (2152, 7887)	0.7971	0.7971
4	200000	(7960, 2001), (2168, 7871)	0.7915	0.7915
5	200000	(7921, 2040), (2093, 7946)	0.7933	0.7933

10.1.4 Investigate the impact of the learning rate decay on BLSTM-ResNet-36.

Based on the experiments in Section 4, BLSTM-ResNet-performed better in extracting sentiment. The training strategy used SGD optimised and adopted predefined a step wise decay of learning rate as followed in original ResNet paper. Every model is trained for 200 epochs which results in longer training times. The training time can be reduced if the epochs can be lowered but it can impact the model performance. So, several learning rate decay approaches can be is used to reduce the learning continuously instead of step wise reduction. The BLSTM-RESNET-36 is used as the test model and four different learning rate decay strategies. First are a stepwise decay and the rest of them are continuous learning rate decay following polynomial function of order 1, 3 and 5. For simplicity, the model is trained on 200000 data points of the Sentiment140 dataset for a total of 50 epochs in each case. The dataset size is chosen

The Figure 10.1 demonstrates a training example of various learning rate decay along with the learning rate reducer method. The spikes represent the training instances where the learning rate reducer is called. Table 10.13 contains average cross validation accuracy for different learning rate trained for 50 epochs.

Table 10.13: Accuracy of various Learning rate decay methods for 50 epoch training

Learning Rate decay method	Average Accuracy	Standard deviation
Step wise decay	78.84%	0.207%
Polynomial decay of order 1	78.38%	0.705%
Polynomial decay of order 3	78.74%	0.305%
Polynomial decay of order 5	79.42%	0.179%

Polynomial order one learning rate decay performed the worst followed by polynomial order 3. The stepwise decay method is second best, and the 5^{th}-degree polynomial performed the best. Not only the accuracy is increasing with the increase in the polynomial decay order, but the standard deviation is also reducing. It is worth noting that in just 50 epochs of training the order 5 polynomial decay reached average accuracy of 79.42% whereas in 200 epochs training with stepwise yielded an average accuracy of 79.68% (refer to BLSTM-RESNET-36 value for 200000 in Section 4.4.3). The training time is significantly reduced from 95 hours to under 24 hours of distributed training across three Tesla V100 GPUs for 5-fold cross-validation. This is massive drop-in training time. The BLSTM-ResNet-training time can be reduced significantly while maintaining the performance if the learning rate is reduced in a higher degree polynomial when trained with SGD optimiser.

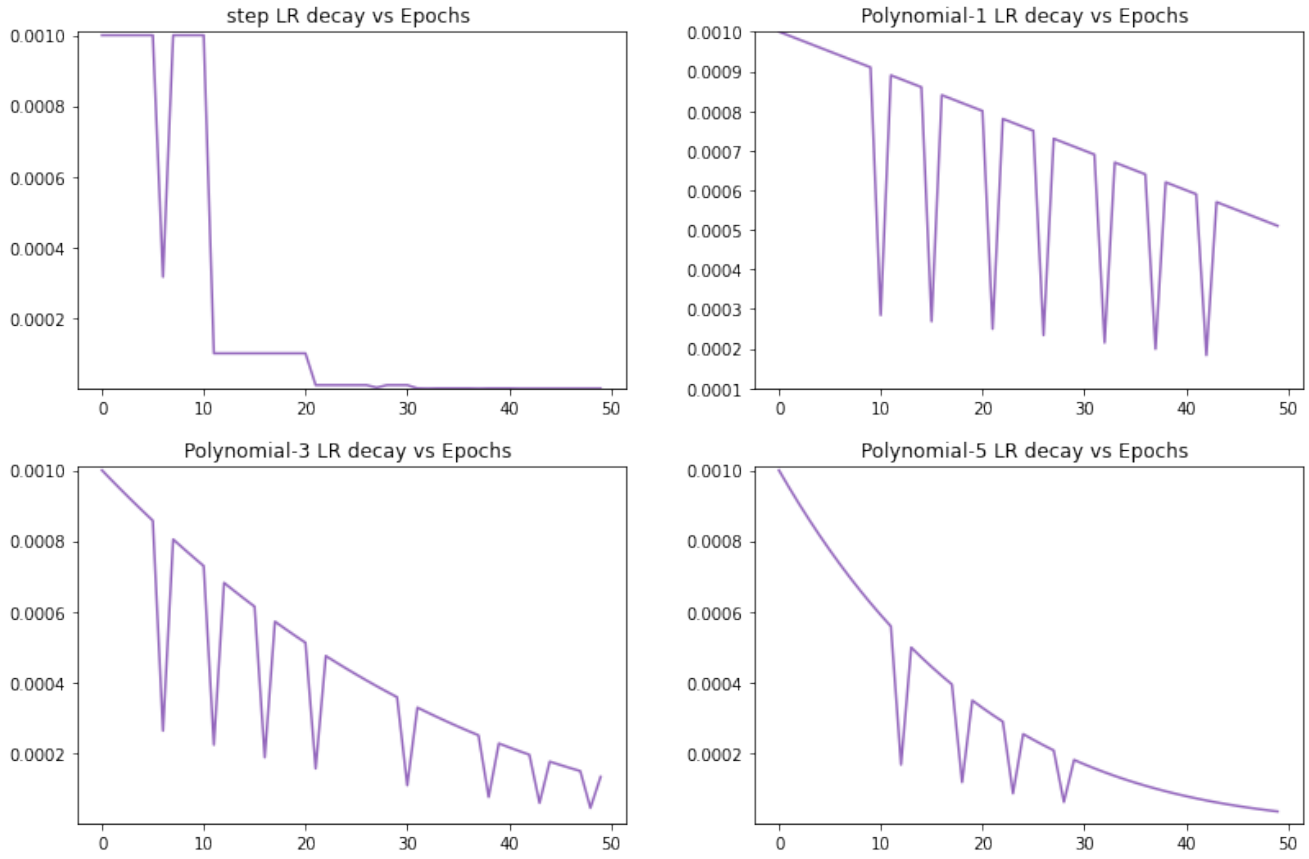


Figure 10.1: Different Learning Rate decay curves for 50 epochs with learning rate reducer

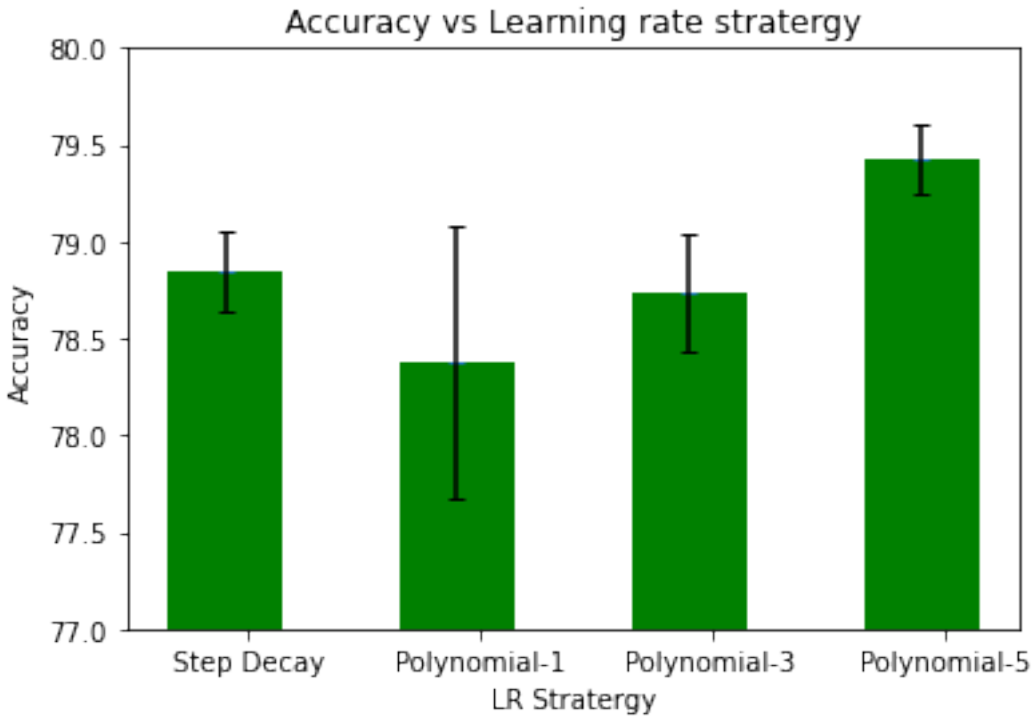


Figure 10.2: Accuracy vs different Learning rate decay methods for 50 epochs training

Chapter 11

Appendix C

The torchinfo.summary reports all learnable parameters (including BERT's embedding matrices, LayerNorm weights, etc.), whereas thop.profile only counts parameters in the operators it knows how to "profile" (mostly Convs, *Linears*, *MatMuls*, etc.). In particular, the large BERT embedding tables (token embeddings, position embeddings, segment embeddings) and many LayerNorm weights in the Transformer layers are not included in thop's "params_count."

Comparing the two parameter-counts for $seq_len = 32$, torchinfo.summary give Total params: 112,059,841 which adds up every trainable weight in the BERT+BLSTM model (BERT's token embeddings: 30 k vocabulary × 768 dim ≈ 23,440,896 parameters + BERT's position embeddings: 512 positions × 768 dim ≈ 393,216 + BERT's segment embeddings: 2 segments × 768 dim = 1,536 + All of BERT's Transformer-layer weights (12 layers × $attention + FFN$ matrices) + Pooler weights (Linear 768×768 = 590,592) + BLSTM weights (~2.57 M) + final Linear (9,601) + LayerNorm biases). Summing those gives ≈ 112 million total trainable parameters. Whereas thop.profile provide Total parameters: 88,224,193. Thop's *params_count* only includes parameters that are inside the layers it explicitly "sees" and counts for its FLOP calculation—primarily like all the Linear/Conv/LSTM weight matrices that participate in multiply–accumulate operations. It also tends to skip parameters inside LayerNorm (since LayerNorm merely does elementwise scaling/shift, not a big "multiply–add" operator).

If you subtract out the two largest chunks that thop omits, you recover the missing ~24 million parameters: Token embeddings (≈ 23.44 M) + Position embeddings (≈ 0.39 M) + segment embeddings (0.0015 M) + LayerNorm scalars (≈ 0.024 M) Those together are about ≈ 23.44 + 0.39 + 0.0015 + 0.024 ≈ 23.85 million. That gap precisely matches BERT's embedding matrices and other "non-MulAdd" parameters that thop does not count.

11.1 Sentiment140 five fold cross validation runs on BERT and BERT-BLSTM-ReNet-18

11.1.1 5-fold cross validation result for BERT for Sentiment140 subsets

Table 11.1: 5-fold cross validation accuracy result and confusion metrics for Sentiment140 subsets for BERT model

Subset size	**1250**	**2500**	**5000**	**10000**	**20000**	**40000**	**80000**	**160000**
CV Test size	**250**	**500**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
1	72.4% [[90, 35], [34, 91]]	79.4% [[197, 53], [50, 200]]	78.3% [[389, 111], [106, 394]]	80.65% [[803, 197], [190, 810]]	81.8% [[1620, 380], [348, 1652]]	81.9375% [[3265, 735], [710, 3290]]	83.54375% [[6737, 1263], [1370, 6630]]	84.25% [[13521, 2479], [2561, 13439]]
2	70.00% [[86, 39], [36, 89]]	82.2% [[208, 42], [47, 203]]	78.7% [[396, 104], [109, 391]]	80.6% [[801, 199], [189, 811]]	81.175% [[1635, 365], [388, 1612]]	82.2375% [[3295, 705], [716, 3284]]	83.825% [[6722, 1278], [1310, 6690]]	84.15% [[13332, 2668], [2404, 13596]]
3	71.2% [[88, 37], [35, 90]]	79.6% [[197, 53], [49, 201]]	77.8% [[385, 115], [107, 393]]	80.4% [[811, 189], [203, 797]]	80.4% [[1617, 383], [401, 1599]]	82.5875% [[3287, 713], [680, 3320]]	83.35% [[6722, 1278], [1386, 6614]]	84.14061% [[13385, 2615], [2460, 13540]]
4	75.6% [[95, 30], [31, 94]]	82.4% [[207, 43], [45, 205]]	77.5% [[384, 116], [109, 391]]	80.35% [[805, 195], [198, 802]]	81.3% [[1638, 362], [386, 1614]]	82.0375% [[3245, 755], [682, 3318]]	83.5625% [[6634, 1366], [1264, 6736]]	84.38126% [[13472, 2528], [2470, 13530]]
5	70.4% [[89, 36], [38, 87]]	79.2% [[200, 50], [54, 196]]	77.9% [[389, 111], [110, 390]]	79.1% [[799, 201], [217, 783]]	80.375% [[1602, 398], [387, 1613]]	82.25% [[3317, 683], [737, 3263]]	83.375% [[6602, 1398], [1262, 6738]]	84.15% [[13404, 2596], [2476, 13524]]

11.1.2 5-fold cross validation result for BERT-ResNet18 for Sentiment140 subsets

Table 11.2: 5-fold cross validation accuracy result and confusion metrics for Sentiment140 subsets for BERT-BLSTM-ResNet18 model

Subset	1250	2500	5000	10000	20000	40000	80000	160000
CV size	250	500	1000	2000	4000	8000	16000	32000
1	85.2% [[108, 17], [20, 105]]	86.2% [[217, 33], [36, 214]]	94.4% [[473, 27], [29, 471]]	87.9% [[877, 123], [119, 881]]	90.85% [[1827, 173], [193, 1807]]	75.45% [[2997, 1003], [961, 3039]]	82.7625% [[6582, 1418], [1340, 6660]]	84.13125% [[13389, 2611], [2467, 13533]]
2	77.2% [[97, 28], [29, 96]]	88.6% [[224, 26], [31, 219]]	95.7% [[476, 24], [19, 481]]	86.5% [[872, 128], [142, 858]]	85.55% [[1727, 273], [305, 1695]]	84.9375% [[3430, 570], [635, 3365]]	83.0375% [[6696, 1304], [1410, 6590]]	82.590625% [[13272, 2728], [2843, 13157]]
3	81.6% [[103, 22], [24, 101]]	88.2% [[221, 29], [30, 220]]	83.2% [[418, 82], [86, 414]]	84.9% [[850, 150], [152, 848]]	82.975% [[1666, 334], [347, 1653]]	90.95% [[3614, 386], [338, 3662]]	80.41875% [[6490, 1510], [1623, 6377]]	81.5125% [[12976, 3024], [2892, 13108]]
4	90.4% [[112, 13], [11, 114]]	92.6% [[231, 19], [18, 232]]	82.2% [[409, 91], [87, 413]]	84.4% [[853, 147], [165, 835]]	89.8% [[1801, 199], [209, 1791]]	78.375% [[3112, 888], [842, 3158]]	82.86875% [[6584, 1416], [1325, 6675]]	84.53125% [[13398, 2602], [2348, 13652]]
5	79.6% [[101, 24], [27, 98]]	81.4% [[204, 46], [47, 203]]	92.9% [[463, 37], [34, 466]]	83.9% [[836, 164], [158, 842]]	82.725% [[1652, 348], [343, 1657]]	89.2% [[3589, 411], [453, 3547]]	81.68125% [[6469, 1531], [1400, 6600]]	82.140625% [[13067, 2933], [2782, 13218]]

11.2 IMDB experiment result on various BLSTM and ResNet configurations extending the BERT output

11.2.1 BERT Model

Table 11.3: BERT Model parameters, Thop parameters, MACs and Flops

Sequence Length	Total Trainable Params	THOP-Counted Params (MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
32	109.48 M	85.65 M	2.721	5.442
64	109.48 M	85.65 M	5.441	10.883
128	109.48 M	85.65 M	10.882	21.764
256	109.48 M	85.65 M	21.764	43.527
384	109.48 M	85.65 M	32.645	65.290
512	109.48 M	85.65 M	43.526	87.053

11.2.2 BERT-BLSTM

Table 11.4: BERT-BLSTM Model parameters, Thop parameters, MACs and Flops

Sequence Length	Total Trainable Params	THOP-Counted Params(MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
32	112.06 M	88.22 M	2.803	5.607
64	112.07 M	88.23 M	5.606	11.212
128	112.09 M	88.25 M	11.211	22.423
256	112.13 M	88.29 M	22.422	44.844
384	112.17 M	88.33 M	33.633	67.266
512	112.20 M	88.37 M	44.844	89.688

11.2.3 BERT-BLSTM||ResNet50

Table 11.5: BERT-BLSTM||ResNet Model parameters, Thop parameters, MACs and Flops

Sequence Length	Total Trainable Params	THOP-Counted Params (MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
32	135.56 M	111.73 M	4.788	9.577
64	135.57 M	111.74 M	9.576	19.153

Sequence Length	Total Trainable Params	THOP-Counted Params (MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
128	135.59 M	111.76 M	19.152	38.304
256	135.63 M	111.80 M	38.303	76.607
384	135.67 M	111.83 M	57.455	114.910
512	135.71 M	111.87 M	76.606	153.213

11.2.4 BERT-BLSTM-RESNET 50

Table 11.6: BERT-BLSTM-ResNet Model parameters, Thop parameters, MACs and Flops

Sequence Length	Total Trainable Params	THOP-Counted Params (MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
32	135.55 M	111.72 M	3.594	7.188
64	135.55 M	111.72 M	7.188	14.376
128	135.55 M	111.72 M	14.375	28.750
256	135.55 M	111.72 M	28.749	57.499
384	135.55 M	111.72 M	43.124	86.247
512	135.55 M	111.72 M	57.498	114.996

11.2.5 BERT-RESNET 50

Table 11.7: BERT-ResNet-50 Model parameters, Thop parameters, MACs and Flops

Sequence Length	Total Trainable Params	THOP-Counted Params (MAC-based)	MACs ($\times 10^9$)	FLOPs ($\times 10^9$)
32	132.99 M	109.15 M	4.706	9.412
64	132.99 M	109.15 M	9.412	18.823
128	132.99 M	109.15 M	18.823	37.645
256	132.99 M	109.15 M	37.645	75.289
384	132.99 M	109.15 M	56.467	112.934
512	132.99 M	109.15 M	75.289	150.578

11.2.6 Experiment result for 10 runs with average performance and standard deviations.

Table 11.8: Experiment result for 10 runs with average performance and standard deviations.

Sentence Length	Accuracies (exp 1-10) in percentage	Average Accuracy	Std. Deviation
BERT			
32	77.11, 78.88, 75.52, 75.70, 75.88, 77.81, 75.14, 74.19, 76.59, 75.97	**76.28**	**1.29**
64	81.84, 81.90, 81.57, 84.15, 82.54, 82.82, 82.52, 83.74, 80.94, 81.19	**82.32**	**0.99**
128	88.44, 90.03, 89.32, 88.47, 87.79, 87.51, 89.81, 88.60, 87.59, 88.48	**88.6**	**0.83**
256	91.65, 91.76, 90.88, 91.32, 91.78, 91.39, 92.16, 91.57, 91.25, 91.83	**91.56**	**0.34**
384	92.44, 92.32, 92.95, 92.58, 92.41, 92.80, 92.65, 92.58, 92.53, 92.22	**92.55**	**0.21**
512	92.80, 92.88, 93.15, 93.16, 93.26, 93.08, 92.89, 92.81, 93.16, 93.27	**93.05**	**0.17**
BERT-BLSTM			
32	77.12, 85.07, 77.91, 78.08, 73.63, 77.81, 75.67, 81.24, 76.91, 76.08	**77.95**	**3.01**
64	81.23, 82.94, 83.38, 88.43, 83.82, 87.21, 83.79, 86.37, 84.83, 82.41	**84.44**	**2.15**
128	88.54, 90.10, 90.41, 89.37, 87.88, 89.52, 91.00, 89.01, 91.03, 85.76	**89.26**	**1.52**
256	92.32, 92.27, 90.62, 91.21, 92.79, 92.50, 92.36, 91.58, 91.90, 91.85	**91.96**	**0.58**
384	92.50, 93.50, 92.50, 94.07, 92.48, 92.87, 93.25, 93.99, 92.67, 92.48	**93.03**	**0.6**
512	92.73, 93.10, 92.83, 93.60, 94.10, 93.96, 93.72, 93.41, 92.94, 93.51	**93.39**	**0.45**

Sentence Length	Accuracies (exp 1-10) in percentage	Average Accuracy	Std. Deviation
BERT-BLSTM-ResNet50			
32	83.20, 81.11, 75.31, 76.01, 75.27, 78.83, 81.94, 75.97, 77.31, 77.08	**78.2**	**2.77**
64	82.63, 81.95, 81.12, 87.52, 85.15, 82.77, 84.55, 84.46, 80.90, 83.83	**83.49**	**1.93**
128	88.54, 93.01, 91.22, 89.11, 88.97, 92.81, 90.69, 88.47, 87.69, 88.19	**89.87**	**1.84**
256	93.25, 93.20, 90.99, 90.33, 92.71, 91.45, 94.42, 92.26, 91.36, 91.95	**92.19**	**1.17**
384	92.73, 92.33, 93.74, 93.27, 93.53, 92.82, 92.70, 93.15, 94.20, 93.02	**93.15**	**0.53**
512	93.21, 93.46, 94.01, 93.15, 94.32, 93.17, 93.21, 93.09, 94.96, 93.47	**93.61**	**0.59**
BERT||BLSTM|| ResNet50			
32	77.10, 82.74, 76.32, 75.67, 76.65, 82.98, 76.76, 82.34, 76.24, 76.79	**78.36**	**2.86**
64	81.94, 82.47, 82.66, 87.71, 84.05, 86.89, 82.73, 85.07, 80.85, 82.81	**83.72**	**2.09**
128	88.59, 89.75, 90.13, 90.06, 87.81, 87.67, 91.14, 89.11, 87.91, 89.54	**89.17**	**1.1**
256	92.19, 91.86, 90.98, 91.28, 92.84, 91.55, 92.98, 91.59, 92.43, 90.92	**91.86**	**0.69**
384	92.56, 92.53, 93.04, 93.56, 92.51, 92.90, 93.58, 92.63, 92.62, 92.55	**92.85**	**0.4**
512	93.34, 93.41, 93.24, 93.76, 94.11, 93.82, 93.17, 93.15, 93.74, 93.67	**93.54**	**0.31**
BERT-ResNet50			
32	77.00, 82.11, 75.73, 75.75, 75.93, 77.95, 76.08, 74.35, 77.50, 80.14	**77.25**	**2.21**
64	82.03, 83.02, 82.25, 84.32, 82.68, 85.79, 82.68, 83.57, 81.01, 81.26	**82.86**	**1.35**
128	88.39, 90.23, 89.73, 89.21, 88.02, 87.61, 90.31, 89.92, 87.41, 88.85	**88.97**	**1.02**
256	91.92, 92.36, 90.88, 91.38, 92.37, 91.48, 92.24, 92.96, 91.33, 91.67	**91.86**	**0.59**
384	92.75, 92.88, 93.51, 93.46, 92.76, 92.77, 92.79, 93.04, 93.19, 93.16	**93.03**	**0.28**
512	92.75, 93.35, 93.10, 93.64, 93.94, 93.22, 93.29, 93.90, 93.20, 93.77	**93.42**	**0.37**

Chapter 12

Appendix D

12.1 Sentiment140 performance on BERT, BERT-ResNet and BERT-VIT.

12.1.1 BERT

Table 12.1: Sentiment140 performance on BERT model.

BERT	Test Accuracy Results (S140 dataset)							Epochs to best training model						
Dataset Size	Run1	Run2	Run3	Run4	Run5	AVG	Std	E1	E2	E3	E4	E5	AVG	Std
625	74.60	77.78	87.30	82.54	77.78	80.00	4.97	5	6	4	4	4	4.6	0.89
1250	73.60	74.40	74.40	72.00	76.00	74.08	1.45	3	6	10	5	4	5.6	2.70
2500	77.60	78.40	76.80	76.80	76.40	77.20	0.80	2	6	7	5	6	5.2	1.92
5000	80.80	81.20	81.00	80.00	80.40	80.68	0.48	1	5	1	1	1	1.8	1.79
10000	81.40	81.70	79.50	81.00	81.20	80.96	0.86	12	7	7	16	6	9.6	4.28
20000	79.60	79.70	80.55	80.35	81.80	80.40	0.88	8	8	6	4	1	5.4	2.97
40000	82.60	82.75	83.57	83.23	82.80	82.99	0.40	1	1	2	1	2	1.4	0.55
80000	83.11	82.97	82.96	82.79	83.17	83.00	0.15	1	1	1	1	1	1.0	0.00
160000	84.20	83.96	83.98	84.26	84.24	84.13	0.15	2	2	2	2	2	2.0	0.00
320000	84.08	84.02	83.87	84.09	83.82	83.98	0.12	3	2	2	2	2	2.2	0.45
640000	84.27	84.10	84.02	84.20	84.18	84.15	0.10	3	3	3	2	3	2.8	0.45
1280000	85.44	85.38	85.22	85.30	85.33	85.33	0.08	2	2	2	2	2	2.0	0.00
1600000	85.45	85.44	85.29	85.40	85.36	85.39	0.07	2	2	2	1	1	1.6	0.55

12.1.2 BERT -ResNet

Table 12.2: Sentiment140 performance on BERT-ResNet

BERT-ResNet50	Test Accuracy Results (S140 dataset)							Epochs to best training model						
Dataset Size	Run1	Run2	Run3	Run4	Run5	AVG	Std	E1	E2	E3	E4	E5	AVG	Std
625	80.95	73.02	82.54	82.54	80.95	80.00	3.98	9	3	17	5	18	10.4	6.84
1250	76.00	76.80	74.40	70.40	74.40	74.40	2.47	13	23	3	9	9	11.4	7.40
2500	76.80	76.80	77.60	76.40	75.60	76.64	0.73	13	5	2	10	2	6.4	4.93
5000	79.60	80.80	80.40	80.80	80.80	80.48	0.52	15	2	2	7	6	6.4	5.32
10000	81.00	82.40	81.10	82.20	82.30	81.80	0.69	7	8	1	2	6	4.8	3.11
20000	81.35	81.85	81.95	81.85	81.40	81.68	0.28	10	3	5	1	5	4.8	3.35
40000	83.30	82.90	83.73	83.05	83.40	83.28	0.32	4	10	4	1	9	5.6	3.78
80000	83.46	83.41	83.43	83.15	83.34	83.36	0.12	9	8	4	6	4	6.2	2.28
160000	84.06	84.15	84.11	84.15	84.03	84.10	0.05	3	4	4	3	6	4.0	1.22
320000	84.40	84.38	84.55	84.34	84.48	84.43	0.08	6	4	3	6	3	4.4	1.52
640000	84.60	84.51	84.41	84.55	84.52	84.52	0.07	3	2	1	3	3	2.4	0.89
1280000	85.61	85.59	85.52	85.53	85.57	85.56	0.04	2	2	2	2	3	2.2	0.45
1600000	85.73	85.70	85.66	85.69	85.61	85.68	0.05	2	2	1	1	2	1.6	0.55

12.1.3 BERT-ViT

Table 12.3: Sentiment140 performance on BERT-ViT

BERT-ViT	Test Accuracy Results (S140 dataset)							Epochs to best training model						
Dataset Size	Run1	Run2	Run3	Run4	Run5	AVG	Std	E1	E2	E3	E4	E5	AVG	Std
625	84.24	84.13	87.30	84.13	82.54	84.47	1.73	14	15	12	12	17	14.0	2.12
1250	71.20	73.60	72.00	73.60	74.40	72.96	1.31	11	13	11	15	11	12.2	1.79
2500	78.40	77.20	78.00	78.00	80.00	78.32	1.04	14	17	13	13	19	15.2	2.68
5000	80.40	79.60	78.60	80.00	80.00	79.72	0.69	12	11	13	9	11	11.2	1.48
10000	81.00	82.20	82.20	82.30	81.60	81.86	0.55	16	13	13	10	15	13.4	2.3
20000	80.95	81.75	81.80	80.75	81.20	81.29	0.47	8	17	8	10	12	11.0	3.74
40000	83.53	83.73	83.27	83.27	83.23	83.41	0.22	10	11	14	13	10	11.6	1.82
80000	83.26	83.21	83.37	83.27	83.01	83.22	0.13	16	7	26	7	6	12.4	8.62
160000	84.09	84.09	84.00	84.15	84.14	84.09	0.06	10	9	11	14	9	10.6	2.07
320000	84.34	84.24	84.23	84.33	84.32	84.29	0.05	6	6	6	6	12	7.2	2.68
640000	84.58	84.56	84.04	83.54	83.24	83.99	0.60	9	9	17	12	8	11.0	3.67
1280000	85.51	85.40	71.83	82.40	85.00	82.03	5.84	8	4	6	10	11	7.8	2.86
1600000	85.67	85.60	76.57	83.60	84.23	83.13	3.78	9	6	5	5	9	6.8	2.05

12.1.4 Confusion Matrix & Agreement Analysis by Dataset Size (BERT vs BERT-ResNet vs BERT-ViT)

Dataset size: 625

Table 12.4: Agreement Analysis by Dataset Size 625 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)

Model	TN	FP	FN	TP
BERT	33	4	4	22
BERT-ResNet	25	12	4	22
BERT-ViT	32	5	4	22

B) Pairwise differences and overlaps

Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	32	[[19, 2], [1, 10]]	[[11, 7], [1, 13]]	31	12, 14, 2, 3
BERT vs BERT-ViT	3	[[1, 0], [1, 1]]	[[0, 1], [1, 1]]	60	21, 32, 4, 3
BERT-ResNet vs BERT-ViT	33	[[12, 8], [1, 12]]	[[19, 3], [1, 10]]	30	10, 13, 4, 3

C) Agreement breakdown across all three models

Category	Count
All models agree	29
BERT & ResNet agree (not ViT)	2
BERT & ViT agree (not ResNet)	31
ResNet & ViT agree (not BERT)	1
BERT only	1
ResNet only	31
ViT only	2

Dataset size: 1250

Table 12.5: Agreement Analysis by Dataset Size 1250 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)

Model	TN	FP	FN	TP
BERT	39	16	14	56
BERT-ResNet	37	18	14	56
BERT-ViT	40	15	14	56

B) Pairwise differences and overlaps

Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	6	[[3, 1], [1, 1]]	[[1, 3], [1, 1]]	119	55, 36, 15, 13
BERT vs BERT-ViT	1	[[0, 1], [0, 0]]	[[1]]	124	56, 39, 15, 14
BERT-ResNet vs BERT-ViT	5	[[0, 3], [1, 1]]	[[3, 0], [1, 1]]	120	55, 37, 15, 13

C) Agreement breakdown across all three models

Category	Count
All models agree	119
BERT & ResNet agree (not ViT)	0
BERT & ViT agree (not ResNet)	5
ResNet & ViT agree (not BERT)	1
BERT only	1
ResNet only	5
ViT only	0

Dataset size: 2500

Table 12.6: Agreement Analysis by Dataset Size 2500 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)

Model	TN	FP	FN	TP
BERT	97	34	20	99
BERT-ResNet	98	33	23	96
BERT-ViT	98	33	17	102

B) Pairwise differences and overlaps

Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	8	[[1, 2], [1, 4]]	[[2, 1], [4, 1]]	242	95, 96, 32, 19
BERT vs BERT-ViT	6	[[1, 2], [3, 0]]	[[2, 1], [0, 3]]	244	99, 96, 32, 17
BERT-ResNet vs BERT-ViT	10	[[2, 2], [6, 0]]	[[2, 2], [0, 6]]	240	96, 96, 31, 17

C) Agreement breakdown across all three models

Category	Count
All models agree	238
BERT & ResNet agree (not ViT)	4
BERT & ViT agree (not ResNet)	6
ResNet & ViT agree (not BERT)	2
BERT only	2
ResNet only	6
ViT only	4

Dataset size: 5000

Table 12.7: Agreement Analysis by Dataset Size 5000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)

Model	TN	FP	FN	TP
BERT	223	44	50	183
BERT-ResNet	214	53	43	190
BERT-ViT	212	55	43	190

B) Pairwise differences and overlaps

Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	18	[[10, 1], [7, 0]]	[[1, 10], [0, 7]]	482	183, 213, 43, 43
BERT vs BERT-ViT	18	[[11, 0], [7, 0]]	[[0, 11], [0, 7]]	482	183, 212, 44, 43
BERT-ResNet vs BERT-ViT	20	[[7, 5], [4, 4]]	[[5, 7], [4, 4]]	480	186, 207, 48, 39

C) Agreement breakdown across all three models

Category	Count
All models agree	472
BERT & ResNet agree (not ViT)	10
BERT & ViT agree (not ResNet)	10
ResNet & ViT agree (not BERT)	8
BERT only	8
ResNet only	10
ViT only	10

Dataset size: 10000

Table 12.8: Agreement Analysis by Dataset Size 10000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	388	81	102	429
BERT-ResNet	397	72	104	427
BERT-ViT	390	79	98	433

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	21	[[0, 9], [5, 7]]	[[9, 0], [7, 5]]	979	422, 388, 72, 97
BERT vs BERT-ViT	24	[[5, 7], [8, 4]]	[[7, 5], [4, 8]]	976	425, 383, 74, 94
BERT-ResNet vs BERT-ViT	27	[[9, 2], [11, 5]]	[[2, 9], [5, 11]]	973	422, 388, 70, 93

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	964
BERT & ResNet agree (not ViT)	15
BERT & ViT agree (not ResNet)	12
ResNet & ViT agree (not BERT)	9
BERT only	9
ResNet only	12
ViT only	15

Dataset size: 20000

Table 12.9: Agreement Analysis by Dataset Size 20000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	860	126	238	776
BERT-ResNet	855	131	230	784
BERT-ViT	836	150	214	800

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	59	[[14, 9], [22, 14]]	[[9, 14], [14, 22]]	1941	762, 846, 117, 216
BERT vs BERT-ViT	84	[[30, 6], [36, 12]]	[[6, 30], [12, 36]]	1916	764, 830, 120, 202
BERT-ResNet vs BERT-ViT	79	[[26, 7], [31, 15]]	[[7, 26], [15, 31]]	1921	769, 829, 124, 199

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	1889
BERT & ResNet agree (not ViT)	52
BERT & ViT agree (not ResNet)	27
ResNet & ViT agree (not BERT)	32
BERT only	32
ResNet only	27
ViT only	52

Dataset size: 40000

Table 12.10: Agreement Analysis by Dataset Size 40000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	1697	296	361	1646
BERT-ResNet	1694	299	352	1655
BERT-ViT	1666	327	324	1683

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	98	[[20, 17], [35, 26]]	[[17, 20], [26, 35]]	3902	1620, 1677, 279, 326
BERT vs BERT-ViT	126	[[44, 13], [53, 16]]	[[13, 44], [16, 53]]	3874	1630, 1653, 283, 308
BERT-ResNet vs BERT-ViT	110	[[41, 13], [42, 14]]	[[13, 41], [14, 42]]	3890	1641, 1653, 286, 310

C) Agreement breakdown across all three models	
Category	Count
All models agree	3833
BERT & ResNet agree (not ViT)	69
BERT & ViT agree (not ResNet)	41
ResNet & ViT agree (not BERT)	57
BERT only	57
ResNet only	41
ViT only	69

Dataset size: 80000

Table 12.11: Agreement Analysis by Dataset Size 80000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	3530	515	831	3124
BERT-ResNet	3447	598	725	3230
BERT-ViT	3519	526	804	3151

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	375	[[122, 39], [160, 54]]	[[39, 122], [54, 160]]	7625	3070, 3408, 476, 671
BERT vs BERT-ViT	264	[[66, 55], [85, 58]]	[[55, 66], [58, 85]]	7736	3066, 3464, 460, 746
BERT-ResNet vs BERT-ViT	351	[[43, 115], [57, 136]]	[[115, 43], [136, 57]]	7649	3094, 3404, 483, 668

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	7505
BERT & ResNet agree (not ViT)	120
BERT & ViT agree (not ResNet)	231
ResNet & ViT agree (not BERT)	144
BERT only	144
ResNet only	231
ViT only	120

Dataset size: 160000

Table 12.12: Agreement Analysis by Dataset Size 160000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	6927	1008	1510	6555
BERT-ResNet	6811	1124	1412	6653
BERT-ViT	6925	1010	1526	6539

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	528	[[184, 68], [187, 89]]	[[68, 184], [89, 187]]	15472	6466, 6743, 940, 1323
BERT vs BERT-ViT	528	[[121,119],[136,152]]	[[119,121],[152,136]]	15472	6403, 6806, 889, 1374
BERT-ResNet vs BERT-ViT	448	[[49, 163], [61, 175]]	[[163, 49], [175, 61]]	15552	6478, 6762, 961, 1351

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	15248
BERT & ResNet agree (not ViT)	224
BERT & ViT agree (not ResNet)	224
ResNet & ViT agree (not BERT)	304
BERT only	304
ResNet only	224
ViT only	224

Dataset size: 320000

Table 12.13: Agreement Analysis by Dataset Size 320000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	14108	1861	3231	12800
BERT-ResNet	13803	2166	2779	13252
BERT-ViT	13476	2493	2517	13514

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	945	[[352, 47], [499, 47]]	[[47, 352], [47, 499]]	31055	12753, 13756, 1814, 2732
BERT vs BERT-ViT	1488	[[665, 33], [752, 38]]	[[33, 665], [38, 752]]	30512	12762, 13443, 1828, 2479
BERT-ResNet vs BERT-ViT	977	[[410, 83], [373, 111]]	[[83, 410], [111, 373]]	31023	13141, 13393, 2083, 2406

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	30295
BERT & ResNet agree (not ViT)	760
BERT & ViT agree (not ResNet)	217
ResNet & ViT agree (not BERT)	728
BERT only	728
ResNet only	217
ViT only	760

Dataset size: 640000

Table 12.14: Agreement Analysis by Dataset Size 640000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	28839	3206	6861	25094
BERT-ResNet	27507	4538	5319	26636
BERT-ViT	27486	4559	5310	26645

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	2928	[[1343, 11], [1558, 16]]	[[11, 1343], [16, 1558]]	61072	25078, 27496, 3195, 5303
BERT vs BERT-ViT	2996	[[1366, 13], [1584, 33]]	[[13, 1366], [33, 1584]]	61004	25061, 27473, 3193, 5277
BERT-ResNet vs BERT-ViT	1196	[[306, 285], [307, 298]]	[[285, 306], [298, 307]]	62804	26338, 27201, 4253, 5012

C) Agreement breakdown across all three models

Category	**Count**
All models agree	60440
BERT & ResNet agree (not ViT)	632
BERT & ViT agree (not ResNet)	564
ResNet & ViT agree (not BERT)	2364
BERT only	2364
ResNet only	564
ViT only	632

Dataset size: 1280000

Table 12.15: Agreement Analysis by Dataset Size 1280000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	57176	6711	12005	52108
BERT-ResNet	55658	8229	10192	53921
BERT-ViT	55712	8175	10374	53739

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	7847	[[2506, 988], [3083, 1270]]	[[988, 2506], [1270,3083]]	120153	50838, 54670, 5723,8922
BERT vs BERT-ViT	7955	[[2504,1040],[3021,1390]]	[[1040,2504],[1390,3021]]	120045	50718, 54672, 5671,8984
BERT-ResNet vs BERT-ViT	2736	[[623, 677], [627, 809]]	[[677, 623], [809, 627]]	125264	53112, 55035, 7552,9565

C) Agreement breakdown across all three models

Category	**Count**
All models agree	118731
BERT & ResNet agree (not ViT)	1422
BERT & ViT agree (not ResNet)	1314
ResNet & ViT agree (not BERT)	6533
BERT only	6533
ResNet only	1314
ViT only	1422

Dataset size: 1600000

Table 12.16: Agreement Analysis by Dataset Size 1600000 (BERT vs BERT-ResNet vs BERT-ViT)

A) Per-model confusion matrices (TN, FP, FN, TP)				
Model	TN	FP	FN	TP
BERT	69881	10199	14236	65684
BERT-ResNet	71162	8918	13915	66005
BERT-ViT	71281	8799	14133	65787

B) Pairwise differences and overlaps					
Pair	Rows different	Diff matrix A	Diff matrix B	Rows same	Overlap (TP,TN,FP,FN)
BERT vs BERT-ResNet	15398	[[2844, 4125], [4375, 4054]]	[[4125, 2844], [4054, 4375]]	144602	61630, 67037, 6074, 9861
BERT vs BERT-ViT	15497	[[2805, 4205], [4295, 4192]]	[[4205, 2805], [4192, 4295]]	144503	61492, 67076, 5994, 9941
BERT-ResNet vs BERT-ViT	2277	[[442, 561], [528, 746]]	[[561, 442], [746, 528]]	157723	65259, 70720, 8357, 13387

C) Agreement breakdown across all three models	
Category	**Count**
All models agree	143414
BERT & ResNet agree (not ViT)	1188
BERT & ViT agree (not ResNet)	1089
ResNet & ViT agree (not BERT)	14309
BERT only	14309
ResNet only	1089
ViT only	1188

12.2 IMDB Dataset (CNN vs Transformer) Models Results

Table 12.17: IMDB Dataset (CNN vs Transformer) Models accuracies across 5 runs along with the Epochs to best performing model.

Interpolation	**BERT-ResNet50**	**BERT-SwinFormer**	**BERT-ViT**	**BERT-DenseNet121**	**BERT-EfficientnetB4**	**BERT-EfficientnetB5**
	Run | Acc | Epoch	Run | Acc | Epoch	Run | Acc | Epoch	Run | Acc | Epoch	Run | Acc | Epoch	Run | Acc | Epoch
Nearest	1|93.10|4 2|93.16|6 3|93.12|11 4|93.18|8 5|93.01|5	1|50.0|1 2|50.0|1 3|50.0|1 4|50.0|1 5|50.0|1	1|93.04|4 2|93.23|10 3|93.23|3 4|93.22|7 5|93.17|10	1|92.99|4 2|93.15|12 3|93.19|4 4|92.97|9 5|93.06|1	1|93.12|4 2|93.02|10 3|93.22|4 4|93.17|4 5|93.15|8	1|93.12|1 2|93.19|9 3|93.37|6 4|93.19|1 5|93.30|1
Bilinear	1|93.02|4 2|92.99|5 3|93.11|7 4|93.20|4 5|93.09|13	1|50.0|1 2|50.0|1 3|50.0|1 4|50.0|1 5|50.0|1	1|93.26|2 2|93.06|2 3|93.11|20 4|93.09|20 5|93.24|19	1|93.18|4 2|93.08|23 3|93.27|5 4|92.92|14 5|93.15|4	1|92.91|3 2|92.94|2 3|93.11|1 4|93.01|28 5|93.15|5	1|93.22|3 2|93.27|1 3|93.24|3 4|93.06|5 5|93.29|5
Bicubic	1|93.23|1 2|93.23|16 3|93.16|12 4|93.08|12 5|93.10|3	1|50.0|1 2|50.0|1 3|50.0|1 4|50.0|1 5|50.0|1	1|93.14|13 2|93.00|3 3|92.99|14 4|93.16|10 5|93.20|15	1|93.25|3 2|93.33|5 3|93.20|3 4|92.88|15 5|93.24|11	1|93.13|3 2|93.22|2 3|93.10|5 4|93.26|1 5|93.13|1	1|93.16|6 2|93.29|1 3|93.34|5 4|93.06|2 5|93.26|7

12.2.1 Kruskal-Wallis H Test with Dunn's Post-hoc Test on model performance

Table 12.18: Model variants with different interpolation ranked based perfromace

Algorithm = (Model/interpolation)	All 5 Accuracies in percentage corresponding to each run	Global Ranks	Sum of Ranks for algorithm	Avg Global Rank (/5)
BERT-EfficientnetB5/Bicubic	93.16, 93.29, 93.34, 93.06, 93.26	45.5, 71, 74, 27.5, 68.5	286.5	57.30
BERT-EfficientnetB5/Bilinear	93.22, 93.27, 93.24, 93.06, 93.29	63, 70, 66.5, 27.5, 71	298	59.60
BERT-EfficientnetB5/Nearest	93.12, 93.19, 93.37, 93.19, 93.30	35, 54.5, 75, 54.5, 72.5	291.5	58.30
BERT-ResNet50/Bicubic	93.23, 93.23, 93.16, 93.08, 93.10	65, 65, 45.5, 30.5, 33.5	239.5	47.90
BERT-ResNet50/Bilinear	93.02, 92.99, 93.11, 93.20, 93.09	20.5, 15, 35, 60.5, 32	163	32.60
BERT-ResNet50/Nearest	93.10, 93.16, 93.12, 93.18, 93.01	33.5, 45.5, 35, 51.5, 18	183.5	36.70
BERT-DenseNet121/Bicubic	93.25, 93.33, 93.20, 92.88, 93.24	67, 73, 60.5, 1, 66.5	268	53.60
BERT-DenseNet121/Bilinear	93.18, 93.08, 93.27, 92.92, 93.15	51.5, 30.5, 70, 3, 43	198	39.60
BERT-DenseNet121/Nearest	92.99, 93.15, 93.19, 92.97, 93.06	15, 43, 54.5, 7, 27.5	147	29.40
BERT-ViT/Bicubic	93.14, 93.00, 92.99, 93.16, 93.20	41.5, 17, 15, 45.5, 60.5	179.5	35.90
BERT-ViT/Nearest	93.04, 93.23, 93.23, 93.22, 93.17	23, 65, 65, 63, 49.5	265.5	53.10
BERT-ViT/Bilinear	93.26, 93.06, 93.11, 93.09, 93.24	68.5, 27.5, 35, 32, 66.5	229.5	45.90
BERT-EfficientnetB4/Bicubic	93.13, 93.22, 93.10, 93.26, 93.13	39.5, 63, 33.5, 68.5, 39.5	244	48.80
BERT-EfficientnetB4/Bilinear	92.91, 92.94, 93.11, 93.01, 93.15	2, 4, 35, 19, 43	103	20.60
BERT-EfficientnetB4/Nearest	93.12, 93.02, 93.22, 93.17, 93.15	35, 20.5, 63, 49.5, 43	211	42.20

12.3 Critical Value Calculation for difference in performance for different vision backbones (Dunn's Post-hoc Test)

Table 12.19: Critical Value Calculation for difference in performance for different vision backbones (Dunn's Post-hoc Test)

Model	***(Avg Global Ranks) across interpolations***
BERT-EfficientnetB5	*(58.30+57.30+59.60)/3=58.4*
BERT-ViT	*(53.10+45.90+35.90)/3=44.97*
BERT-DenseNet121	*(29.40+39.60+53.60)/3=40.87*
BERT-ResNet50	*(36.70+32.60+47.90)/3=39.07*
BERT-EfficientnetB4	*(42.20+20.60+48.80)/3=37.2*

The Kruskal-Walli's test will determine if any group differs. Assuming it is significant, we proceed to Dunn's test for pairwise comparisons.

For Dunn's test with k = 15 groups and , the $\alpha = 0.05$, the critical value z_{crit} (adjusted using the Bonferroni correction for multiple comparisons) is approximately **3.45**.

The calculation of the critical -value for Dunn's test involves finding the standard z -score that corresponds to an adjusted significance level (α_{adjusted}) dictated by the **Bonferroni correction**.

The Bonferroni correction is used to control the Family-Wise Error Rate when performing multiple pairwise comparisons. For k algorithms, the total unique pairwise comparison m is given by

$$m = \frac{k(k-1)}{2} = \frac{15(15-1)}{2} = 105$$

Therefore the $\alpha_{\text{adjusted}} = \frac{\alpha}{m} = \frac{0.05}{105} \approx 0.000476$ and for 2-tailed test, it will be 0.000238.

The z-score corresponding to a cumulative area of $1 - 0.000230 = 0.999762 \approx 3.45$

The test statistic Q for comparing two groups, i and j, is:

$$Q = \frac{|\overline{R_i} - \overline{R_j}|}{\sqrt{\left(\frac{N_T(N_T+1)}{12}\right)\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

Where:

- $N_T = 75$ *(total sample size)*
- $n_i = n_j = 5$ *(sample size for each algorithm)*
- $\overline{R_i}$*and* $\overline{R_j}$ *are the average global ranks.*

Given , the denominator simplifies to a constant **Critical Difference (CD)** factor:

$CD\ Factor = \sqrt{\left(\frac{75(76)}{12}\right)\left(\frac{1}{5} + \frac{1}{5}\right)} \approx 13.784$

Therefore, the pairwise comparison is:

To be significant, we need: $\frac{|\overline{R_i} - \overline{R_j}|}{13.784} > 3.45$, which means

$$CD = 3.45 \times 13.784 \approx 47.56$$

a difference in the average global ranks greater than **47.56** indicates a statistically significant difference in performance.

The highest ranked algorithm overall is BERT-EfficientnetB5/Bilinear (Avg Rank 59.60) and worst ranked algorithm is EfficientnetB4/Bilinear (Avg Rank 20.60) and their difference is $39.00 < 47.56$.

To declare significance, the difference in average global ranks must be greater than 47.56 which is not observed in any of the algorithm pairs so no statistically significant difference in accuracy exist between empirical best model and any other viable alternative.

12.3.1 Significance test based on interpolation method

To compare the statistical differences in the groups using different interpolation methods. The below table provides the Average Global rank

Table 12.20: Significance test based on interpolation method on Textual Image classification of pre-trained vision models.

Interpolation Method	Accuracies ($n = 25$)	Global Ranks Sum (Ri)	Avg Global Rank ($\overline{Ri}$)
Bicubic	25 scores	1536	1536.0/25=61.44
Bilinear	25 scores	998.5	998.5/25=39.94
Nearest	25 scores	1240.5	1240.5/25=49.62

For Dunn's test with $n = 25$*,* $k = 3$ *groups and , the* $\alpha = 0.05$*, the critical value* z_{crit} *(adjusted using the Bonferroni correction for multiple comparisons) is approximately* **2.39,** $m = 3$, Standard error ≈ 6.164 and the Critical distance ≈ 14.73 .

Table 12.21: Significance is difference in average model rank for various interpolation methods.

Comparison	Difference ($R_i - R_j$)	Statistically Significant?
Bicubic vs. Nearest	61.44 - 49.62 = 11.82	No (11.82 < 14.73)
Bicubic vs. Bilinear	61.44 - 39.94 = 21.50	Yes (21.50 > 14.73)
Nearest vs. Bilinear	49.62 - 39.94 = 9.68	No (9.68 < 14.73)

12.3.1.1 One-Way ANOVA Test for Interpolation Methods

A one-way Analysis of Variance (ANOVA) was conducted to determine if there were any statistically significant differences between the mean accuracies of the three interpolation methods: Nearest, Bilinear, and Bicubic. Each interpolation method had 25 samples, consisting of the accuracies from five different models, each run five times.

12.3.1.2 Assumption Checks

Before performing the ANOVA, two key assumptions were checked:

1. Normality: The Shapiro-Wilk test was used to determine if the accuracy data for each interpolation method was normally distributed.

- Nearest Interpolation: p-value = 0.6060
- Bilinear Interpolation: p-value = 0.3069
- Bicubic Interpolation: p-value = 0.4636

Since all p-values are greater than 0.05, the data for all three groups are considered to be normally distributed, and the normality assumption is met.

2. Homogeneity of Variances: Levene's test was conducted to check if the variances of the accuracies were equal across the three interpolation groups.

- Levene's test statistic: 0.5857
- p-value: 0.5593

The p-value from Levene's test is greater than 0.05, indicating that the variances are not significantly different across the groups. Thus, the homogeneity of variances assumption is met.

12.3.1.3 ANOVA Results

With both assumptions satisfied, the one-way ANOVA was performed.

- F-statistic: 1.2213
- p-value: 0.3009

The F-statistic is the ratio of the variance between the groups to the variance within the groups. A larger F-statistic suggests a greater difference between the groups.

Conclusion and Significance

The p-value (0.3009) is greater than the chosen significance level of . Therefore, we fail to reject the null hypothesis.

This result indicates that there is no statistically significant difference in the mean accuracies among the Nearest, Bilinear, and Bicubic interpolation techniques. This aligns with the conclusion from the previous Kruskal-Wallis test, providing further evidence that the choice of interpolation method, in this case, does not significantly impact model performance.

12.3.2 Friedman test followed by Nemenyi post-hoc test

Table 12.22: Friedman test followed by Nemenyi post-hoc Significance test based on interpolation and pretrained vision model variant pairs on Textual Image classification.

Model/Interpolation	Run 1	Run 2	Run 3	Run 4	Run 5	Sum of Ranks	Avg Rank
	Accuracy(rank)	Accuracy(rank)	Accuracy(rank)	Accuracy(rank)	Accuracy(rank)		
BERT-EfficientnetB5/ Nearest	93.12% (5)	93.19% (3)	93.37% (1)	93.19% (2)	93.30% (1)	12	2.4
BERT-EfficientnetB5/ Bilinear	93.22% (2)	93.27% (2)	93.24% (3)	93.06% (10)	93.29% (1)	18	3.6
BERT-EfficientnetB5/ Bicubic	93.16% (3)	93.29% (1)	93.34% (2)	93.06% (10)	93.26% (2)	18	3.6
BERT-DenseNet121/ Nearest	92.99% (13)	93.15% (7)	93.19% (4)	92.97% (13)	93.06% (10)	47	9.4
BERT-DenseNet121/ Bilinear	93.18% (3)	93.08% (9)	93.27% (2)	92.92% (14)	93.15% (5)	33	6.6
BERT-DenseNet121/ Bicubic	93.25 % (1)	93.33% (1)	93.20% (4)	92.88% (15)	93.24% (3)	24	4.8
BERT-ViT/Nearest	93.04% (9)	93.23% (3)	93.23% (3)	93.22% (1)	93.17% (4)	20	4
BERT-ViT/Bilinear	93.26% (1)	93.06% (11)	93.11% (8)	93.09% (9)	93.24% (3)	32	6.4
BERT-ViT/Bicubic	93.14% (3)	93.00% (15)	92.99% (15)	93.16% (5)	93.20% (4)	42	8.4
BERT-EfficientnetB4/ Nearest	93.12% (5)	93.02% (13)	93.22% (4)	93.17% (4)	93.15% (5)	31	6.2
BERT-EfficientnetB4/ Bilinear	92.91% (15)	92.94% (15)	93.11% (8)	93.01% (12)	93.15% (5)	55	11
BERT-EfficientnetB4/ Bicubic	93.13% (4)	93.22% (4)	93.10% (9)	93.26% (0)	93.13% (6)	23	4.6
BERT-ResNet50/Nearest	93.10% (8)	93.16% (6)	93.12% (7)	93.18% (3)	93.01% (14)	38	7.6
BERT-ResNet50/Bilinear	93.02% (12)	92.99% (14)	93.11% (8)	93.20% (2)	93.09% (9)	45	9
BERT-ResNet50/Bicubic	93.23% (2)	93.23% (3)	93.16% (5)	93.08% (8)	93.10% (8)	26	5.2

Table 12.23: Overall average Ranks of various interpolation method across various pre-trained vision classifier performance on contextual Textual Images.

Interpolation	Avg Rank (Overall)
Nearest	(2.40+9.40+4.00+6.20+7.60)/5=5.92
Bilinear	(3.60+6.60+6.40+11.00+9.00)/5=7.32
Bicubic	(3.60+4.80+8.40+4.60+5.20)/5=5.32

Table 12.24: Overall average Ranks of various pre-trained vision classifier performance across various interpolation methods on contextual Textual Images.

Model	Avg Rank (Overall)
BERT-EfficientnetB5	(2.40+3.60+3.60)/3=3.20
BERT-ViT	(4.00+6.40+8.40)/3=6.27
BERT-ResNet50	(7.60+9.00+5.20)/3=7.27
BERT-DenseNet121	(9.40+6.60+4.80)/3=6.93
BERT-EfficientnetB4	(6.20+11.00+4.60)/3=7.27

12.3.3 Experiment Report

Table 12.25: Pre-trained Vision Model parameters and their respective parameters and giga FLOPs required per contextualised Textual Images processing.

Model	FLOPs (Giga)	Parameters (Millions)
ResNet18	1.82	11.18
ResNet34	3.68	21.29
ResNet50	4.13	23.51
ResNet101	7.86	42.5
ResNet152	11.6	58.15
Efficientnet-B0	0.41	4.01
Efficientnet-B1	0.6	6.51
Efficientnet-B2	0.69	7.7
Efficientnet-B3	1.01	10.7
Efficientnet-B4	1.56	17.55
Efficientnet-B5	2.44	28.34
Efficientnet-B6	3.47	40.74
Efficientnet-B7	5.32	63.79
DenseNet-121	2.9	6.95
DenseNet-161	7.85	26.47
DenseNet-169	3.44	12.49
DenseNet-201	4.39	18.09
ViT-Base	12.02	85.8

12.4 Pre-trained Efficientnet accuracy on Contextualised Textual Images.

12.4.1 Model: Efficientnet_b0

Table 12.26: Efficientnet_b0 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b0	0.05	0.9056	0.9144	0.9216	0.9176	0.9192	91.57%	0.62%
Efficientnet_b0	0.1	0.9236	0.9244	0.9268	0.9256	0.9288	92.58%	0.21%
Efficientnet_b0	0.2	0.9306	0.929	0.93	0.9244	0.9286	92.85%	0.24%
Efficientnet_b0	0.4	0.9304	0.9302	0.9311	0.9281	0.9312	93.02%	0.13%
Efficientnet_b0	0.6	0.932	0.9331	0.933	0.9335	0.9291	93.21%	0.18%
Efficientnet_b0	0.8	0.9316	0.9306	0.9301	0.9328	0.9321	93.14%	0.11%
Efficientnet_b0	1.0	0.9326	0.9316	0.93	0.9324	0.9326	93.18%	0.11%

12.4.2 Model: Efficientnet_b1

Table 12.27: Efficientnet_b1 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b1	0.05	0.9168	0.9136	0.92	0.9168	0.9024	91.39%	0.68%
Efficientnet_b1	0.1	0.9228	0.9244	0.9276	0.928	0.9288	92.63%	0.26%
Efficientnet_b1	0.2	0.9274	0.9242	0.9282	0.9282	0.9296	92.75%	0.20%
Efficientnet_b1	0.4	0.9299	0.9254	0.9258	0.9244	0.9303	92.72%	0.27%
Efficientnet_b1	0.6	0.9325	0.9345	0.9278	0.9328	0.9342	93.24%	0.27%
Efficientnet_b1	0.8	0.9346	0.9345	0.9336	0.9335	0.9329	93.38%	0.07%
Efficientnet_b1	1.0	0.9264	0.9295	0.9328	0.9316	0.9313	93.03%	0.25%

12.4.3 Model: Efficientnet_b2

Table 12.28: Efficientnet_b2 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b2	0.05	0.9192	0.9216	0.9192	0.9216	0.9224	92.08%	0.15%
Efficientnet_b2	0.1	0.926	0.9252	0.9252	0.9268	0.9244	92.55%	0.09%
Efficientnet_b2	0.2	0.9294	0.9292	0.9296	0.9208	0.9288	92.76%	0.38%
Efficientnet_b2	0.4	0.9293	0.9264	0.9279	0.9291	0.9256	92.77%	0.16%
Efficientnet_b2	0.6	0.9327	0.9331	0.9327	0.9267	0.9332	93.17%	0.28%
Efficientnet_b2	0.8	0.9323	0.9323	0.9333	0.9316	0.9344	93.28%	0.11%
Efficientnet_b2	1.0	0.9296	0.9314	0.9321	0.9314	0.93	93.09%	0.11%

12.4.4 Model: Efficientnet_b3

Table 12.29: Efficientnet_b3 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b3	0.05	0.9072	0.916	0.9144	0.9208	0.9184	91.54%	0.52%
Efficientnet_b3	0.1	0.9296	0.9276	0.9252	0.9272	0.9256	92.70%	0.18%
Efficientnet_b3	0.2	0.9298	0.9266	0.9276	0.929	0.9298	92.86%	0.14%
Efficientnet_b3	0.4	0.9264	0.9285	0.9296	0.9304	0.924	92.78%	0.26%
Efficientnet_b3	0.6	0.9339	0.933	0.9311	0.9301	0.9327	93.22%	0.15%
Efficientnet_b3	0.8	0.9344	0.9324	0.9347	0.9338	0.9306	93.32%	0.17%
Efficientnet_b3	1.0	0.9293	0.9298	0.9297	0.9328	0.9316	93.06%	0.15%

12.4.5 Model: Efficientnet_b4

Table 12.30: Efficientnet_b4 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b4	0.05	0.92	0.9192	0.9144	0.9216	0.9136	91.78%	0.36%
Efficientnet_b4	0.1	0.926	0.9256	0.9272	0.9272	0.928	92.68%	0.10%
Efficientnet_b4	0.2	0.9254	0.9284	0.9296	0.9228	0.9294	92.71%	0.29%
Efficientnet_b4	0.4	0.9313	0.9283	0.9287	0.9286	0.9305	92.95%	0.13%
Efficientnet_b4	0.6	0.9325	0.9297	0.9313	0.9327	0.9303	93.13%	0.13%
Efficientnet_b4	0.8	0.9296	0.9326	0.934	0.9324	0.9311	93.19%	0.17%
Efficientnet_b4	1.0	0.9301	0.9311	0.9304	0.9325	0.9291	93.06%	0.13%

12.4.6 Model: Efficientnet_b5

Table 12.31: Efficientnet_b5 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b5	0.05	0.9104	0.9232	0.9184	0.9248	0.9232	92.00%	0.59%
Efficientnet_b5	0.1	0.9272	0.9196	0.924	0.9232	0.9234	92.35%	0.27%
Efficientnet_b5	0.2	0.9304	0.928	0.929	0.925	0.928	92.81%	0.20%
Efficientnet_b5	0.4	0.9288	0.9287	0.9307	0.9308	0.9286	92.95%	0.11%
Efficientnet_b5	0.6	0.9319	0.9317	0.9291	0.9285	0.93	93.02%	0.15%
Efficientnet_b5	0.8	0.9332	0.9353	0.9339	0.9342	0.9336	93.40%	0.08%
Efficientnet_b5	1.0	0.931	0.9306	0.9312	0.9316	0.9317	93.12%	0.04%

12.4.7 Model: Efficientnet_b6

Table 12.32: Efficientnet_b6 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b6	0.05	0.9248	0.9216	0.9232	0.9112	0.924	92.10%	0.56%
Efficientnet_b6	0.1	0.9264	0.9252	0.928	0.924	0.926	92.59%	0.15%
Efficientnet_b6	0.2	0.9296	0.9278	0.929	0.9312	0.9298	92.95%	0.12%
Efficientnet_b6	0.4	0.9321	0.9313	0.9308	0.9287	0.9308	93.07%	0.13%
Efficientnet_b6	0.6	0.933	0.9328	0.9325	0.9336	0.933	93.30%	0.04%
Efficientnet_b6	0.8	0.934	0.9334	0.9344	0.9314	0.934	93.34%	0.12%
Efficientnet_b6	1.0	0.931	0.9312	0.9305	0.9321	0.9324	93.14%	0.08%

12.4.8 Model: Efficientnet_b7

Table 12.33: Efficientnet_b7 accuracy on Contextualised Textual Images various runs in three epochs.

Model	Dataset Size	Exp1	Exp2	Exp3	Exp4	Exp5	Avg	std
Efficientnet_b7	0.05	0.9176	0.9192	0.9152	0.9208	0.9128	91.71%	0.32%
Efficientnet_b7	0.1	0.924	0.9284	0.9172	0.9256	0.94	92.70%	0.83%
Efficientnet_b7	0.2	0.9238	0.928	0.9284	0.9292	0.928	92.75%	0.21%
Efficientnet_b7	0.4	0.9298	0.9309	0.93	0.9301	0.9301	93.02%	0.04%
Efficientnet_b7	0.6	0.9322	0.9328	0.9337	0.9292	0.932	93.20%	0.17%
Efficientnet_b7	0.8	0.9352	0.9312	0.9345	0.935	0.934	93.40%	0.16%
Efficientnet_b7	1.0	0.9305	0.9317	0.9296	0.9332	0.9312	93.12%	0.14%

12.4.9 Comparative plots of average performance with standard deviations of BERT-Efficient variants organised in various views.

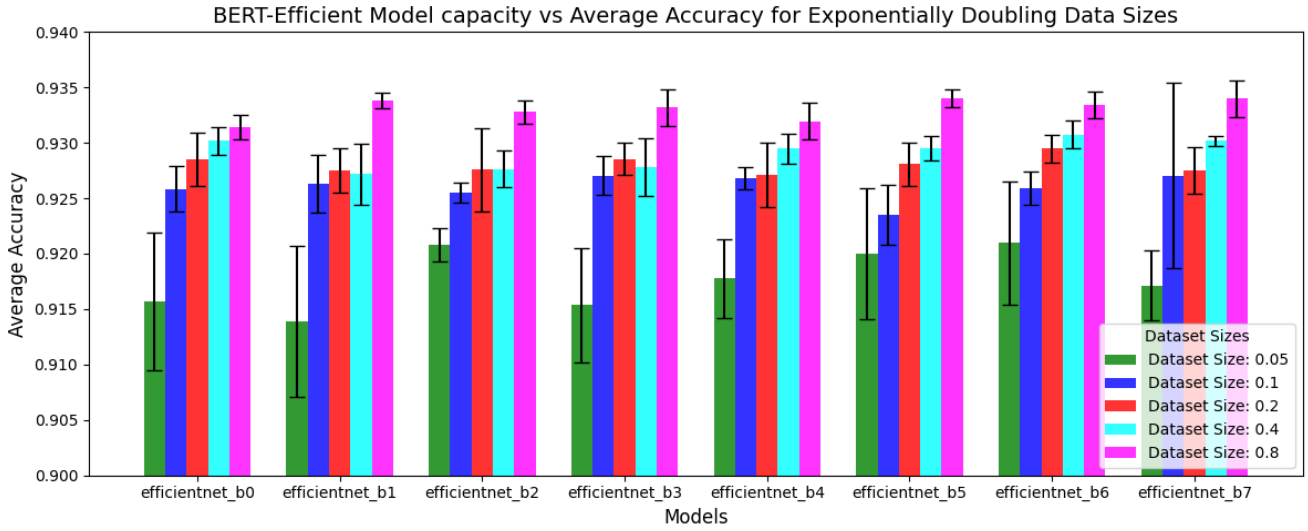


Figure 12.1: BERT-Efficient Model capacity vs Average Accuracy for Exponentially Doubling Data Sizes

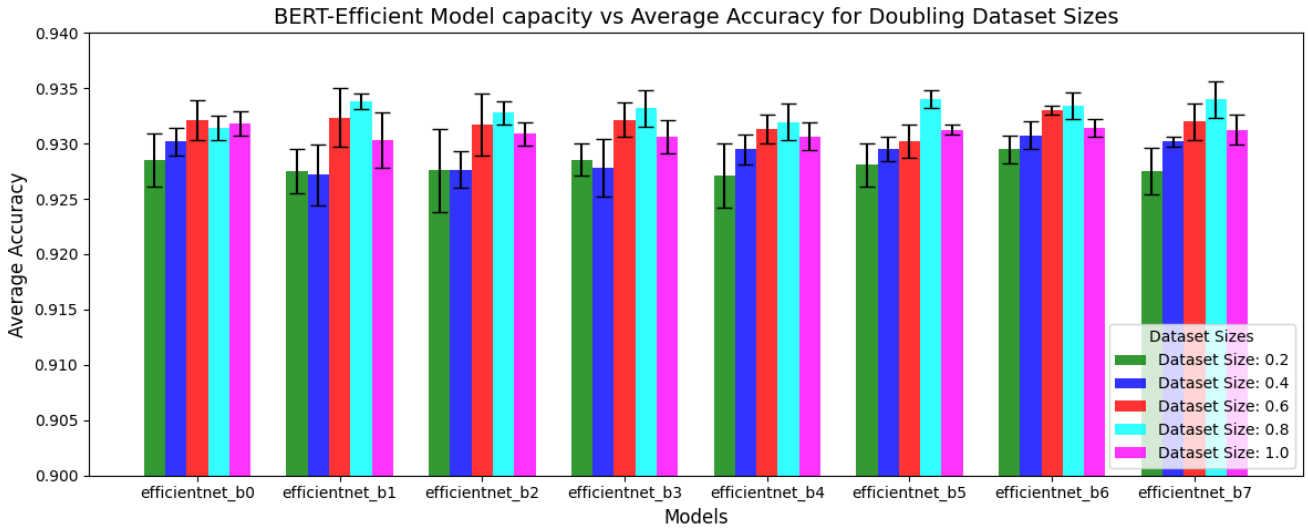


Figure 12.2: BERT-Efficient Model capacity vs Average Accuracy for Doubling Dataset Sizes

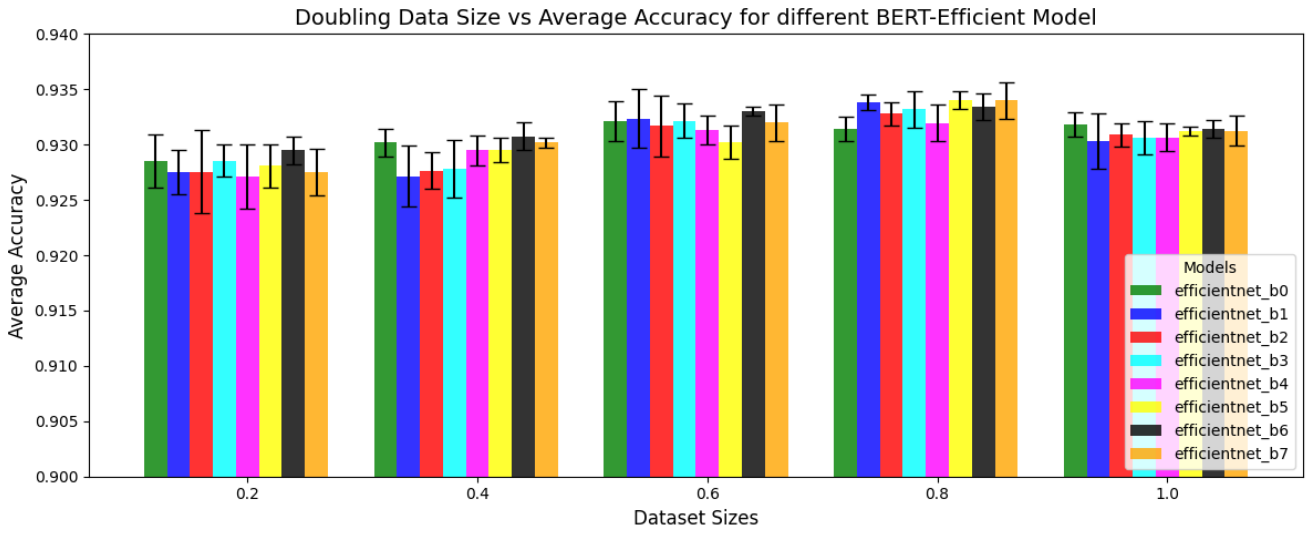


Figure 12.3: Doubling Data Size vs Average Accuracy for different BERT-Efficient Model

12.4.10 Comparative plots of best performance of BERT-Efficient variants organised in various views.

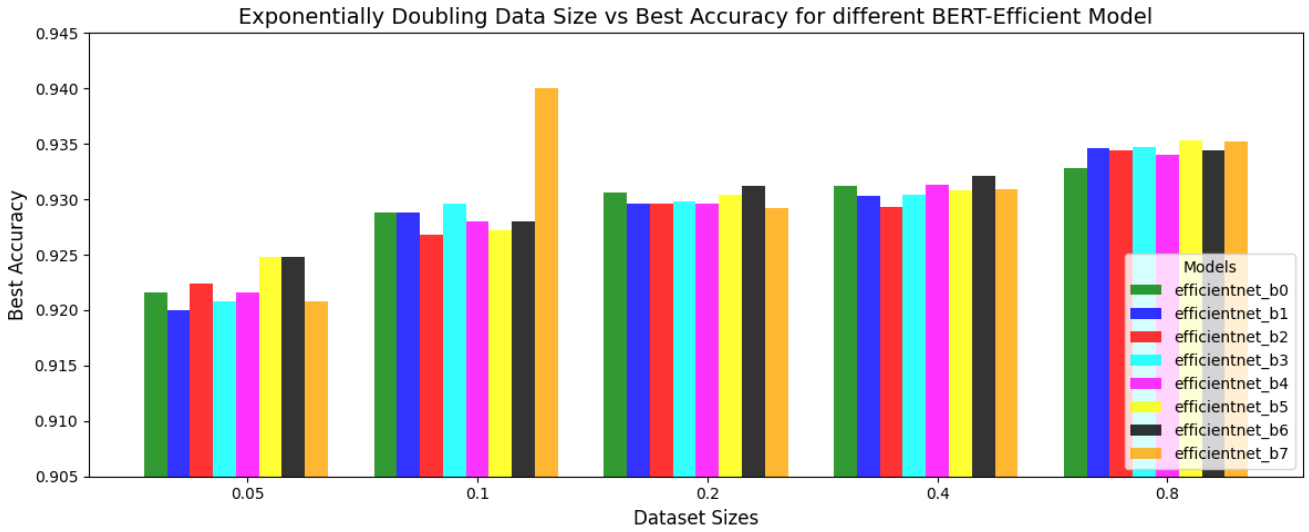


Figure 12.4: Exponentially Doubling Data Size vs Best Accuracy for different BERT-Efficient Model

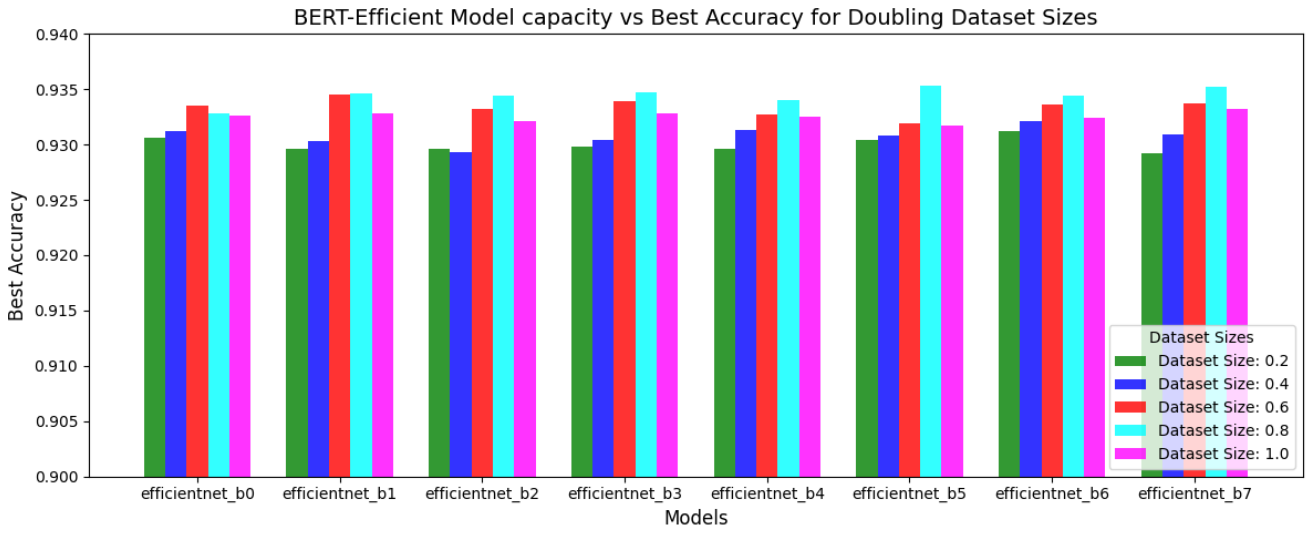


Figure 12.5: BERT-Efficient Model capacity vs Best Accuracy for Doubling Dataset Sizes

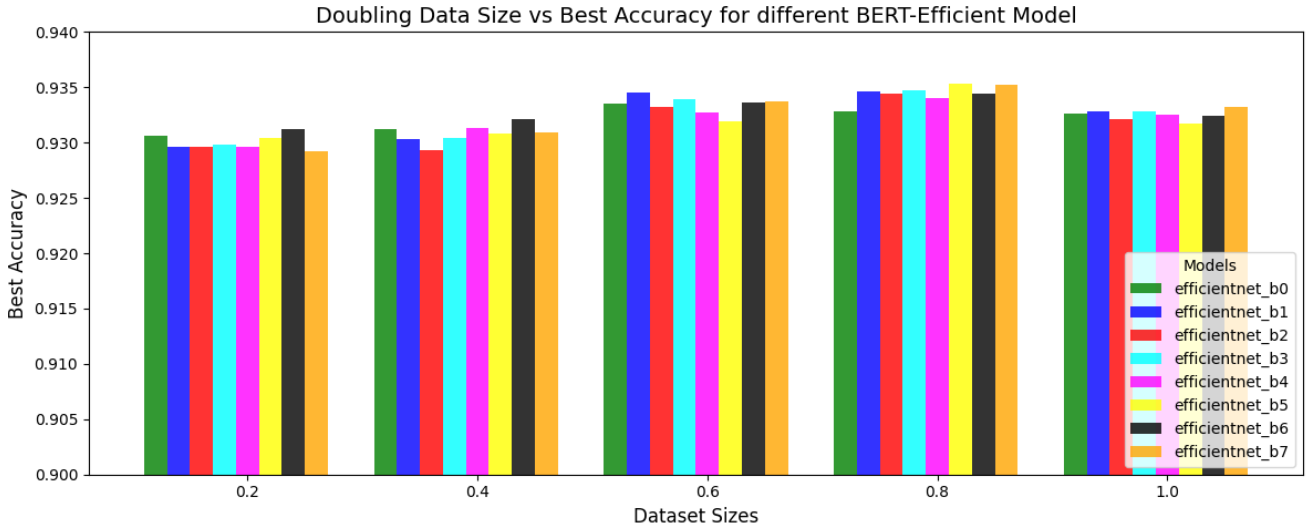


Figure 12.6: Doubling Data Size vs Best Accuracy for different BERT-Efficient Model

12.5 Choosing the Best LSA+LDA components pared with best machine learning model parameters.



Figure 12.7: IMDB dataset top words

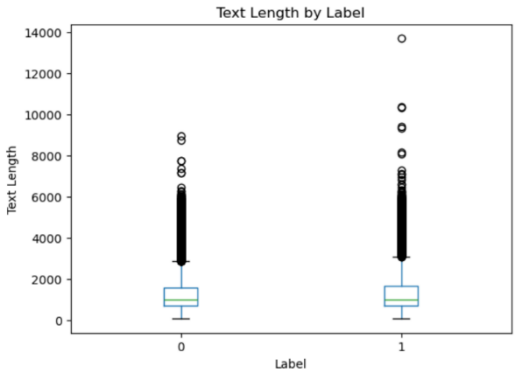


Figure 12.8: IMDB dataset box plot

In this experiment first we combined the IMDB train and test set and then choose 90-10 train-test split so maximise the training data. Then model evaluation is performed to find the best machine learning models on default setting just to understand the landscape of LSA + LDA component impact on performance.

1.Initialize parameter grid for the model and generate a LSA, LDA combined feature dataset with up to 450 components to generate features from word vectors with increment of 50 with initial 50 components. Order the features based on significance. which will be used to create several subsets with different no of LSA and LDA features from each dataset.

2. perform following tasks

a. Load train and test datasets for a current LSA and LDA values.

b. Separate features and labels from the datasets.

c. Initialize Model and perform GridSearchCV:

- Use 5-fold cross-validation and specified hyperparameters.

- Fit the model on the training data.

d. Predict the test data using the best model.

e. Compute and log accuracy and classification report.

3. Plot accuracy trends for each LDA value across LSA components.

12.5.1 Exponential increment of LSA and LDA components across different model best parameters

The Figure 12.9 demonstrate the MLP performance on various combinations of LSA + LDA on the performance of the model. A similar trend is observed for most machine learning model where the increase in either LSA or LDA component resulted in model performance. It is observed that some LSA + LDA combinations are performing better than others in some instances, but these differences fizzle out we compare the results overall. If we fix one either of the component and observe the trend, then we see the huge variability in final performance when the total components increase. This is true for both LSA and LDA. For Example, for 50 LDA components if we observe the performance variability across various components of LSA then it is higher than the variability observed at 200 LDA components which in turn is higher than 400 components. Same is observed if we fix LSA components and observe the variability across the LDA components. In short, the higher the number of components will result in increase in the model performance.

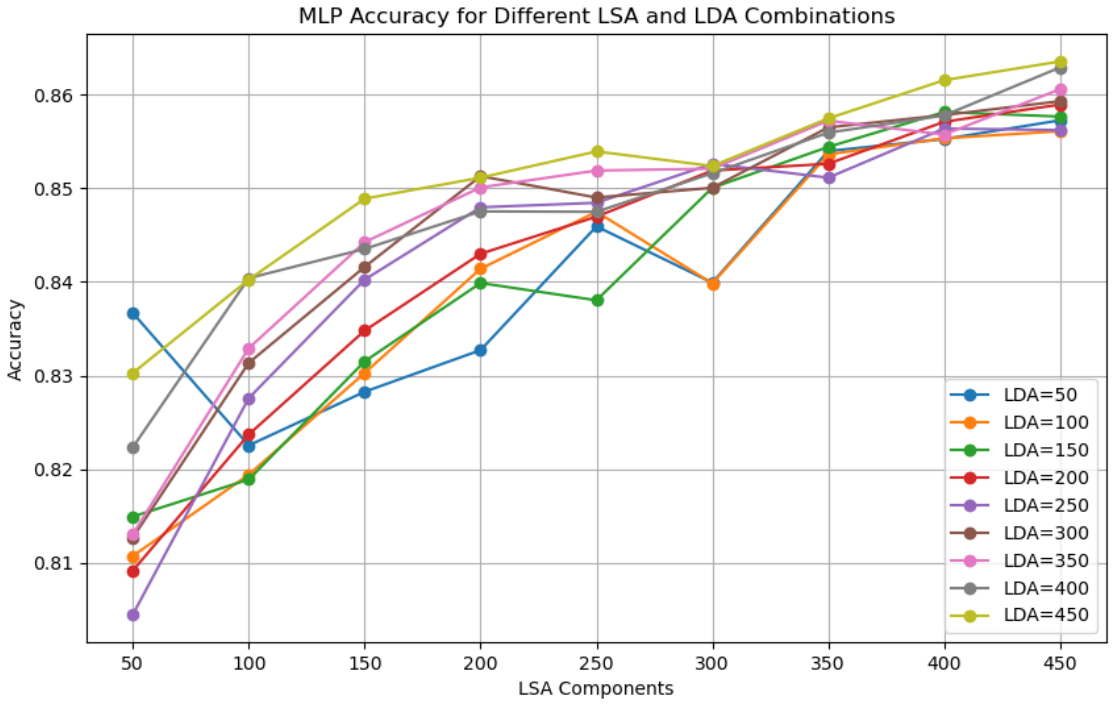


Figure 12.9: LSA vs LDA components impact of MLP

After this, the impact of various pre-processing and their combinations is observed across different machine learning models. For this we fixed the total combination of LSA and LDA reduce the number of total runs. Also, I performed grid search for every preprocessing combination tested to then use the best performing model parameter as the representative in the plot below. So, looking at each plot, it is observable that if we LDA leads to more performance gain with its increase in component than LSA in most machine learning models, if we let the model parameter optimise on the components. For this we used the original Training and Test set, so the peak performance obsessed is lower than the previous experiment as the training samples reduced and the testing sample increased so these set of experiments are more robust.

The LSA and LDA already use English stop words so no need to implement them separately they are available inherently. The Br removal is made part of the Basic cleaning to simplify the cleaning process. The above plot shows the repeating patterns in some of the results, so combinations were removed from the analysis. We have kept only processing which yielded some good results in the final predictions. Basic cleaning, removing Br, stemming and lemmatisation.

12.5.2 Logistic Regression

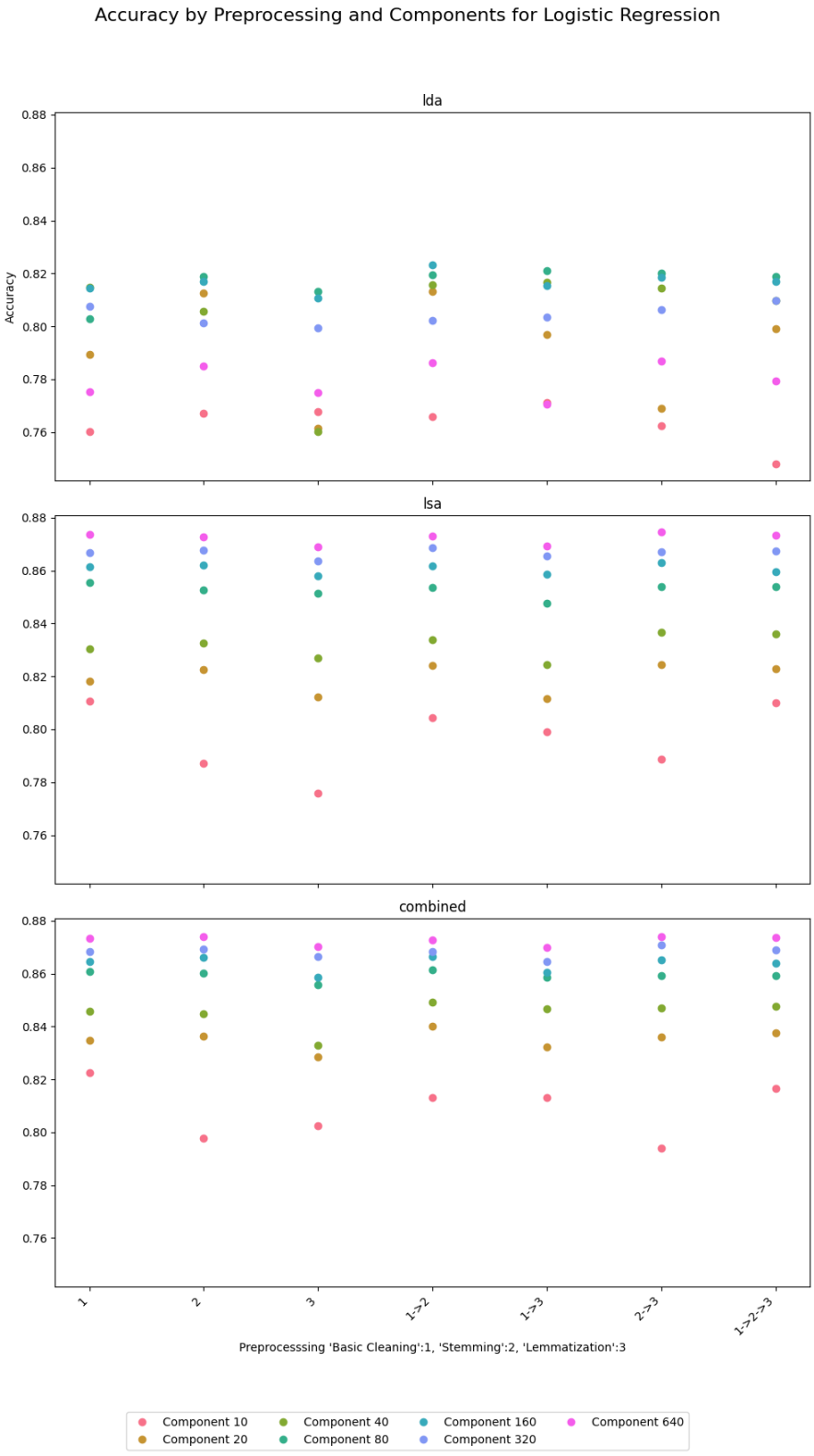


Figure 12.10: Various LSA + LDA components combined with different pre-processing combinations logistic Regression

12.5.3 Multi-Layer Perceptron (MLP)

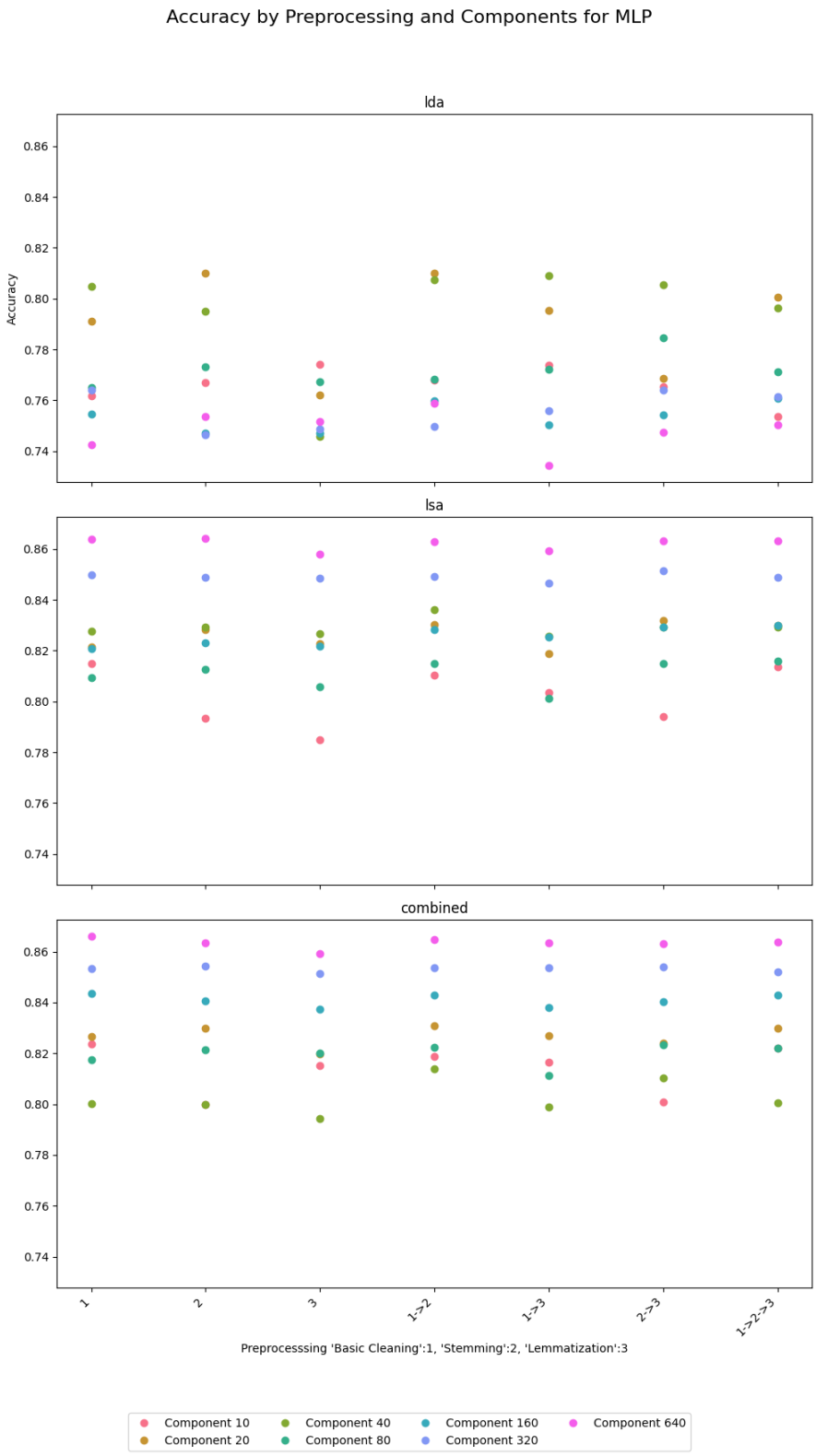


Figure 12.11: Various LSA + LDA components combined with different pre-processing combinations MLP

12.5.4 Naïve Bayes

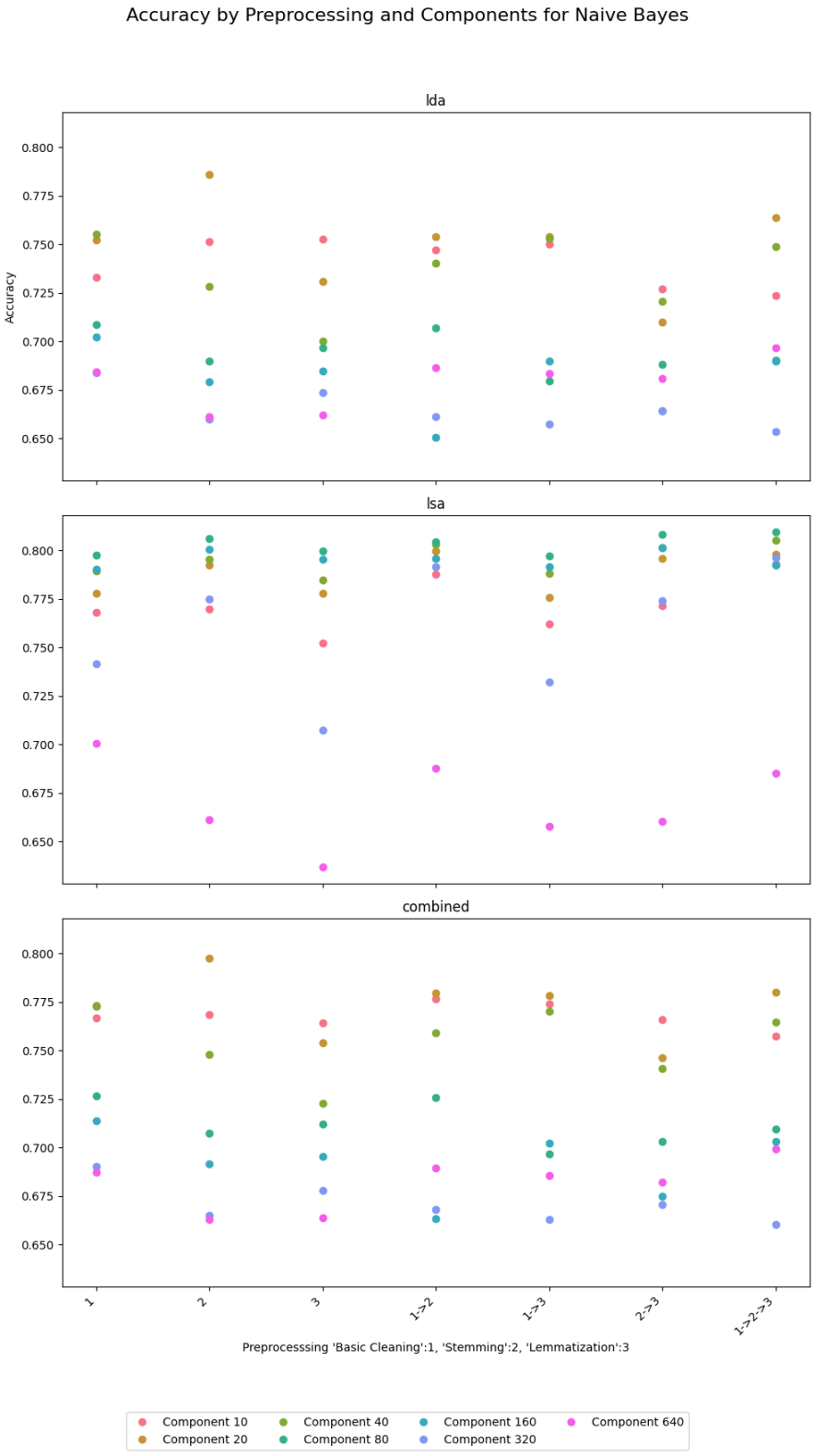


Figure 12.12: Various LSA + LDA components combined with different pre-processing combinations Naïve Bayes

12.5.5 Support Vector Machine (SVM)

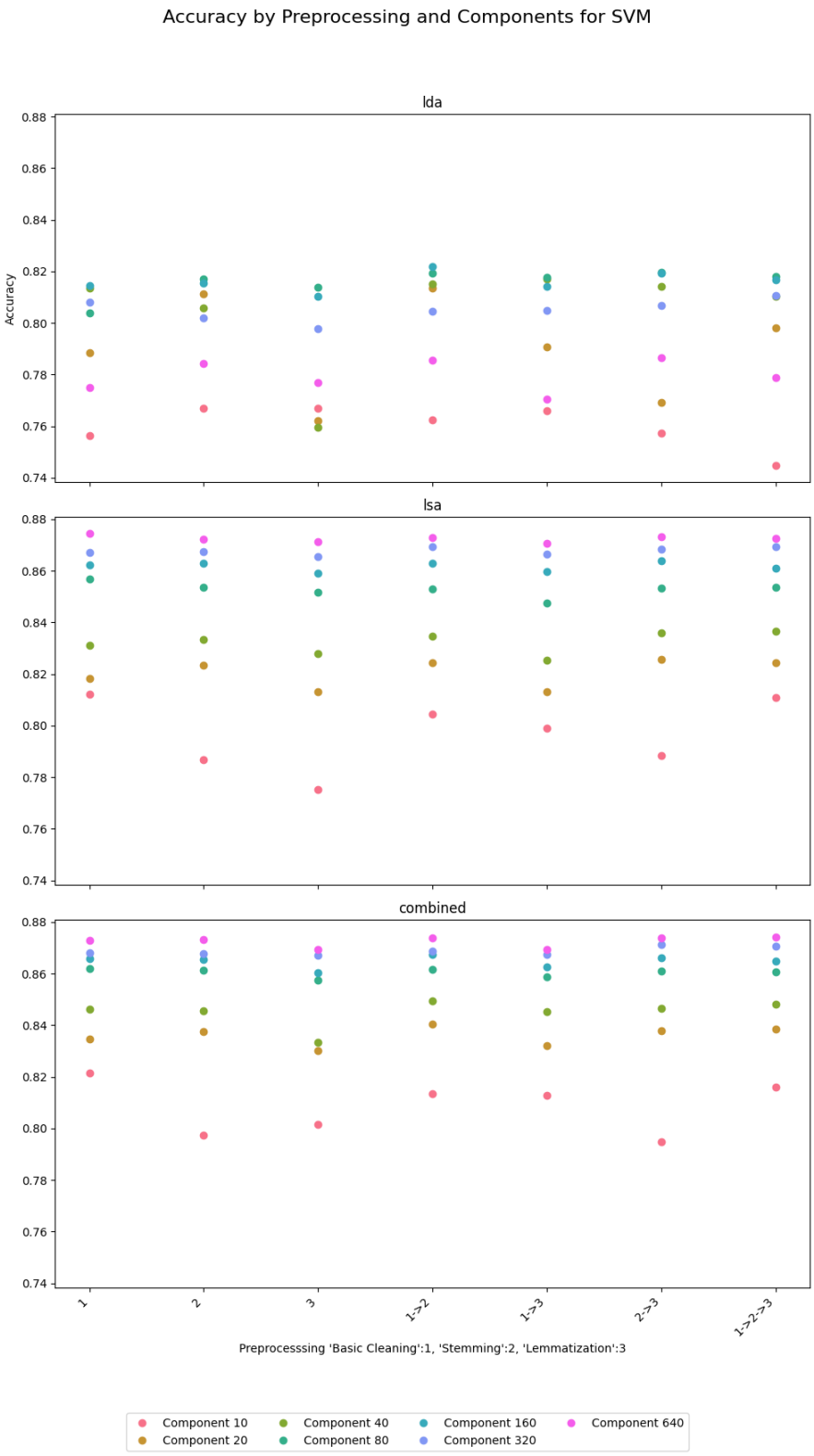


Figure 12.13: Various LSA + LDA components combined with different pre-processing combinations SVM

12.5.6 XGBoost

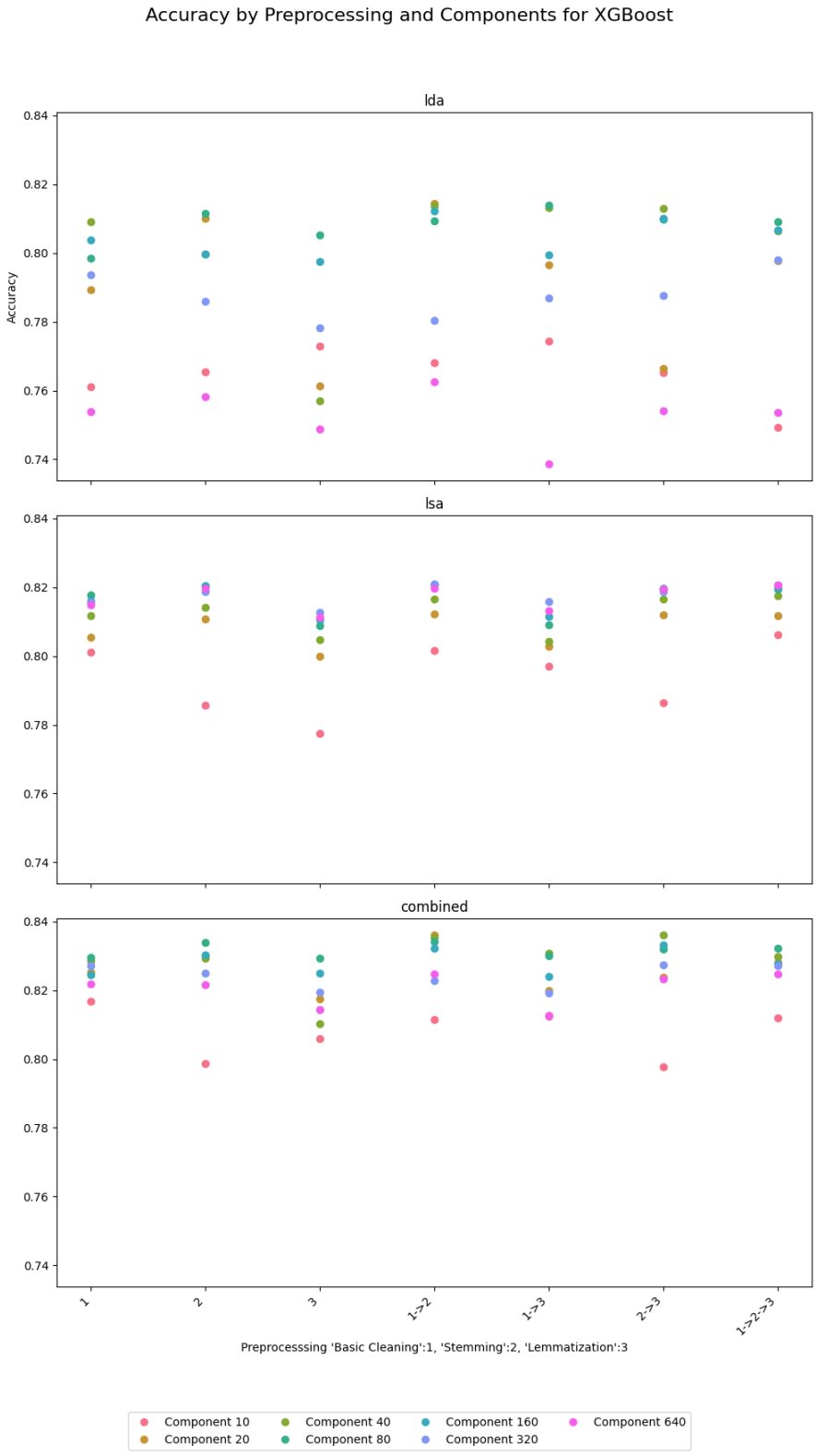


Figure 12.14: Various LSA + LDA components combined with different pre-processing combinations XGBoost

12.6 BERT-SWC performance vs Alpha.

Table 12.34: BERT-SWC model performance vs alpha.

Alpha		Run1	Run2	Run3	Run4	Run5
0.0	Accura y	49.80	49.80	49.80	49.80	50.20
	Confusion metric	[[0 502] [0 498]]	[[0 502] [0 498]]	[[0 502] [0 498]]	[[0 502] [0 498]]	[[502 0] [498 0]]
0.1	Accura y	93.30	92.60	92.10	92.90	90.10
	Confusion metric	[[461 41] [26 472]]	[[470 32] [42 456]]	[[439 63] [16 482]]	[[468 34] [37 461]]	[[453 49] [50 448]]
0.2	Accura y	93.00	94.00	94.00	94.70	93.80
	Confusion metric	[[476 26] [44 454]]	[[481 21] [39 459]]	[[468 34] [26 472]]	[[478 24] [29 469]]	[[478 24] [38 460]]
0.3	Accura y	92.90	93.70	92.90	93.80	92.80
	Confusion metric	[[454 48] [23 475]]	[[475 27] [36 462]]	[[459 43] [28 470]]	[[466 36] [26 472]]	[[463 39] [33 465]]
0.4	Accura y	93.60	94.00	93.70	93.60	94.00
	Confusion metric	[[471 31] [33 465]]	[[478 24] [36 462]]	[[470 32] [31 467]]	[[471 31] [33 465]]	[[468 34] [26 472]]
0.5	Accura y	93.20	92.80	93.60	93.10	92.70
	Confusion metric	[[476 26] [42 456]]	[[457 45] [27 471]]	[[465 37] [27 471]]	[[463 39] [30 468]]	[[463 39] [30 468]]
0.6	Accura y	93.40	93.20	93.00	93.10	93.20
	Confusion metric	[[467 35] [31 467]]	[[473 29] [39 459]]	[[467 35] [35 463]]	[[452 50] [19 479]]	[[471 31] [37 461]]
0.7	Accura y	93.90	95.10	92.60	94.00	93.00
	Confusion metric	[[477 25] [36 462]]	[[469 33] [16 482]]	[[461 41] [33 465]]	[[474 28] [32 466]]	[[474 28] [42 456]]
0.8	Accura y	93.80	93.50	92.90	92.50	93.80
	Confusion metric	[[466 36] [26 472]]	[[462 40] [25 473]]	[[453 49] [22 476]]	[[449 53] [22 476]]	[[479 23] [39 459]]
0.9	Accura y	93.00	93.50	94.20	93.40	93.60
	Confusion metric	[[457 45] [25 473]]	[[465 37] [28 470]]	[[477 25] [33 465]]	[[469 33] [33 465]]	[[471 31] [33 465]]
1.0	Accuracy	93.60	92.40	93.30	93.20	93.90
	Confusion metric	[[481 21] [43 455]]	[[471 31] [27 471]]	[[462 40] [27 471]]	[[467 35] [33 465]]	[[473 29] [32 466]]