NLP Term Project: Ablation Study of Models for Sentiment Analysis

Jerome Wang[1]
Kristiyan Cholakov[1]
Teg Tiwana[1]
Ryan Goh[1]
Denzyl Peh[1]
Clement Tan[1]

[1]College of Computing and Data Science, Nanyang Technological University
{JWANG075, KRISTIYA001, TEGS0001, SGOH055, DENZ0003, CLEM0024}@e.ntu.edu.sg

Group G3

Name	Matric Num
Jerome Wang	U2120160F
Cholakov Kristiyan Kamenov	U2123543B
Goh Soong Wen Ryan	U2120980L
Denzyl David Peh	U2122190F
Clement Tan Kang Hao	U2122052K
Tiwana Teg Singh	U2122816B

Contributions: Everyone contributed equally :)

Introduction

Natural Language Processing (NLP) has become an essential field in understanding human language and enabling machines to interpret and respond to text-based data effectively. In this assignment, the Rotten Tomatoes movie review dataset was used to build a sentiment classification model.

To develop our classifier, various pre-processing techniques were applied to minimize the Out-of-Vocabulary (OOV) rate, enhancing the model's ability to interpret the dataset effectively. Furthermore, multiple models, including RNN, biLSTM, biGRU, CNN, and a vanilla transformer with self-attention and positional encoding, were implemented. Additionally, an ensemble model was utilized, combining the strengths of these individual models to further optimize accuracy as a performance metric and meet the assignment's goals.

1 Part 1: Preparing Word Embeddings

1.1 Raw Dataset Statistics

After tokenizing the training corpus using the NLTK tokenizer, the number of unique words (and hence vocabulary size) is **18,029**. When using Word2Vec embeddings, there are **3808 Out-of-Vocabulary words** (i.e. words that exist in the training corpus but not the vocabulary of the embeddings).

1.2 OoV Mitigation Strategies

As there seem to be a significant number of OoV words, we first switched the embeddings from Word2Vec to GloVe, which saw the number of OoV words drop from 3808 to 1865 words. We chose GloVe 300d as it was the largest dimensioned embedding that we could run on our Macintosh laptops without running into Out-of-Memory errors during model training.

In general, larger embedding dimensions are beneficial because they allow for richer semantic representation, capturing more nuanced relationships and contexts between words [4]. Specifically with GloVe, we can more feasibly use larger dimensions compared to Word2Vec due to its distinct training approach. GloVe is trained as a count-based model on global word co-occurrence statistics [3], resulting in embeddings that are computationally less expensive to generate and less prone to prolonged convergence issues compared to Word2Vec. In contrast, Word2Vec relies on a neural network architecture [2], which becomes increasingly computationally demanding as the embedding dimension grows. For Word2Vec, each additional dimension significantly extends the training time and may still not guarantee full convergence, especially with large datasets or complex semantic relationships. GloVe's efficiency in handling larger dimensions thus enables us to balance computational constraints with the benefits of higher-dimensional embeddings.

We further pre-processed the OoV words by firstly removing apostrophes (‘), then splitting the remaining OoVs on hyphens (-) and checking if the split parts exist in vocab (if they do not the parts are re-merged). This saw the OoV count drop to 668. Lemmatization then saw OoV drop to 659, and lastly stemming further dropped OoV count to 602. The details are summarized in table 1.

Pre-processing executed	Word2Vec OoV Words	GloVe OoV Words
None (Vanilla)	3808	1865
Pre-processing	2362	668
Lemmatization	2171	659
Stemming	1612	602

Table 1: OoV Comparison for Word2Vec and GloVe

2 Part 2: Model Training & Evaluation - RNN

2.1 Final RNN Configuration

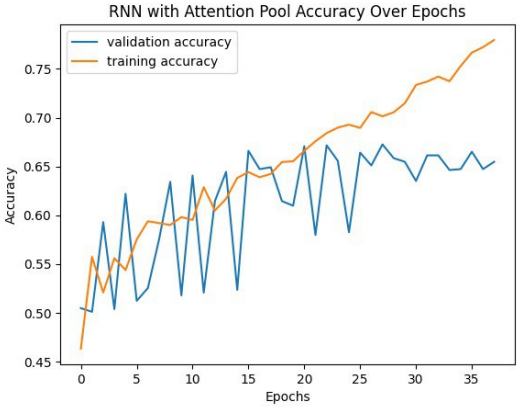
Selected Model: RNN with Attention Pooling

Model Configuration	Value
Number of Training Epochs	100
Optimizer	0.01
Batch Size	32
Early Stopping	Patience of 10 epochs, starting from epoch 20 (restoring best weights)
Loss Function	Binary Crossentropy
Metrics	Accuracy (As per assessment requirements)
Activation Functions	tanh (RNN Units), ReLU (non-linear layer), sigmoid (output layer)
L2 Regularization	True
Dropout	0.5

Table 2: Final RNN Configuration

2.2 Final RNN Performance

Selected Model: RNN with Attention Pooling



Test Metric	Value
Test Accuracy	**0.6585**
Validation Accuracy	0.6726

Figure 2.1: Training Performance of RNN with Attention Pooling

2.3 Derivation of Final Sequence Representation

To select the best method of representing the sequences, we explored Final State, Maximum, Average and Attention-based pooling.

2.3.1 Last State RNN (SentimentRNNLastState)

This model leverages only the last hidden state of the RNN to represent the entire sentence, focusing on the cumulative context up to the last word by condensing all preceding word information into a single vector. This method is particularly useful for tasks where the sentence's conclusion often reinforces the sentiment or main idea, effectively summarising the sequence. However, this may not work well for longer sentences, as the effect of earlier words may be severely diluted to a point where it becomes negligible, resulting in not as effective sentiment classification, as seen in table 3.

Test Metric	Value
Test Accuracy	**0.5966**
Validation Accuracy	0.5938

Table 3: Accuracy of RNN with Last State

2.3.2 Max Pool RNN (SentimentRNNMaxPool)

Maximum Pooling runs across all hidden states to retain the most activated features, likely representing sentiment-heavy words. This distils each feature to its strongest signal across the sentence and captures the peak response from each hidden dimension, performing feature-wise max selection. This approach can be formalized as follows:

$$c = \max(h_t)$$

where c is the maximum-valued hidden state that will be used for classification. This model might be effective for our movie reviews dataset, where sentiment can often hinge on a few keywords, such as "excellent" or "terrible," making max pooling a possible strategy for extracting a decisive sentiment representation. However, if the word contains contrasting sentiment words, it will simply select the strongest-sentiment word which may not representative sentiment for the sentence. For example, for the sentence "I had a fantastic meal but had limited enjoyment due to the company", intuitively, "fantastic" is a more sentiment-laden word than "limited", which may result in the model erroneously predicting a positive sentiment for that sentence. Thus, the results are still not ideal (Table 4).

Test Metric	Value
Test Accuracy	**0.5000**
Validation Accuracy	0.6679

Table 4: Accuracy of RNN with Max Pooling

2.3.3 Average Pool RNN (SentimentRNNAvgPool)

This model averages hidden states, thereby treating each word's contribution equally in representing sentence sentiment. AvgPoolRNN generates a holistic sentence representation by capturing overall sentiment trends across words. Unlike max pooling, which relies solely on the strongest activated features, average pooling captures overall tone by accounting for every word's contribution. For example, this review, "Starts with a bang, but fizzles like a wet stick of dynamite at the end," average pooling provides a balanced representation, recognising the mix of positive and negative sentiment. This approach can be formalized as follows:

$$c = \text{mean}(h_t)$$

where c is the mean of all hidden states that will be used for classification. This model, however, wrongly assumes a uniform distribution of importance, thereby allocating equal weight to each state. Consequently, it may overfocus on less "important" states (corresponding to less "important" words) and underfocus on more "important" words, leading to an underrepresentation of particularly salient words or phrases. To illustrate, in the sentence above, the ending negative sentiment, "wet stick of dynamite" may not be adequately emphasized as the model failed to take into account its relation to the contrastive conjunction "but". Similarly, the results are still not ideal (Table 5).

Test Metric	Value
Test Accuracy	**0.6201**
Validation Accuracy	0.6407

Table 5: Accuracy of RNN with Avg Pooling

2.3.4 Attention Pool RNN (SentimentRNNAttentionPool)

As attention pooling yielded the best test accuracy (Figure A), we chose this method as the base for future enhancements. This model integrated the Bahdanau attention mechanism (also known as

additive attention)[1], which allowed the model to dynamically weigh the importance of each word in a sentence, resulting in a weighted representation that better models the nuances of human language.

Firstly, the weight vectors for each hidden state are derived by applying a dense transformation with hyperbolic tangent activation (which helps capture complex, non-linear relationships) on each of these states. This transformation creates a "feedforward network" that projects each hidden state into a common space, calculating attention scores without being too computationally expensive. Such a process results in the weight vector u_t for the t-th hidden state, formally defined as:

$$u_t = \tanh(W \cdot h_t + h_{\text{final}})$$

where W is the weight matrix, h_t is the hidden state at time step t, and h_{final} is the final hidden state of the sequence.

Next, we transform the weight vector to a scalar by applying another dense layer with a single output unit (no activation), effectively projecting the weight vector into a single value for each hidden state, which will serve as our "alignment score." As this is a Seq2Class task with no decoder, the alignment score in this context represents how much each hidden state h_t contributes to the final hidden state.

$$e_t = v \cdot u_t$$

where v is a learnable parameter vector. Following attention score computation, softmax is applied to generate a probability distribution for each e_t across all timestamps, where α_t is the attention weight for each timestep:

$$\alpha_t = \frac{\exp(e_t)}{\sum_k \exp(e_k)}$$

Each of these attention weights α_t represents the degree of focus required for each hidden state h_t, as computed from the degree of correlation between h_t and the final hidden state as derived from u_t.

Finally, we use the attention weights to scale each hidden state and then sum them, resulting in the weighted sum c that serves as the context vector representing the entire sentence:

$$c = \sum_t \alpha_t \cdot h_t$$

Although Bahdanau attention was originally developed for a Seq2Seq task, where alignment scores correlate each source word to its corresponding target word, it also functions well for our Seq2Class task. Here, it is able to correlate the importance of each word with respect to the overall classification decision, as the alignment scores correlate the importance of each hidden state to the final hidden state.

3 Part 3: Enhancement

3.1 Trainable Embeddings

Instead of freezing the embeddings, we could also train it. This gives us **0.6670** for the test accuracy (Table 6), which is almost the same as when we set it to non-trainable (**0.6585**). However, it took 22 epochs, which is 11 more than when embeddings were frozen (11 epochs), to reach peak performance (before overfitting started). This suggests that the unfreezing of embeddings not only has no substantial positive impact on the performance on the model, but also resulted in longer convergence time.

Test Metric	**Value**
Test Accuracy	**0.6670**
Validation Accuracy	0.6811

Table 6: Accuracy of RNN with Attention Pooling and Trainable Embeddings

3.2 OoV Mitigation

The accuracy of the model had improved significantly, jumping from **0.6585** to **0.7627**. This indicates that the number of OoV has a significant impact on model performance, which makes sense as the less OoV words there are, the more context the model has for training and inference.

Test Metric	Value
Test Accuracy	**0.7627**
Validation Accuracy	0.7730

Table 7: Accuracy of RNN with Attention Pooling, Trainable Embeddings and Mitigated OoV

3.3 BiLSTM

The BiLSTM (Bidirectional Long Short-Term Memory) layer offers notable advantages over traditional RNNs and BiGRUs (Bidirectional Gated Recurrent Units) in handling long-term dependencies in sequential data. Unlike RNNs, which process information in one direction, BiLSTMs operate in both forward and backward directions, capturing contextual information from both the past and future within a sentence. For instance, in "The movie was not as good as I hoped," BiLSTMs can understand that the word "hoped" at the end influences the sentiment of the entire sentence, something a single-directional RNN might miss. It is thus no surprise that BiLSTMs performs better, as can be seen in Table 8.

Test Metric	Value
Test Accuracy	**0.7636**
Validation Accuracy	0.7824

Table 8: Accuracy of BiLSTM with Attention Pooling, Trainable Embeddings and Mitigated OoV

3.4 BiGRU

Compared to BiGRUs, BiLSTMs have additional gates (input, output, and forget) that provide finer control over which information to retain, discard, or output, making them better suited for long sequences with complex dependencies. GRUs, with a simpler update gate, are more computationally efficient but may be less effective over lengthy sequences. For tasks like sentiment analysis, where context may span the entire text, BiLSTMs are advantageous for retaining nuanced information, as can be seen in Table 9.

Test Metric	Value
Test Accuracy	**0.7824**
Validation Accuracy	0.7927

Table 9: Accuracy of BiGRU with Attention Pooling, Trainable Embeddings and Mitigated OoV

3.5 CNN

Though Convolutional Neural Networks (CNN) are not commonly used for NLP tasks, it is interesting to see how it performs for Sentiment Classification. Surprisingly, it performs better than the RNN with Attention pool (0.7627), with a test accuracy of **0.7777**.

Test Metric	Value
Test Accuracy	**0.7777**
Validation Accuracy	0.7720

Table 10: Accuracy of CNN with Attention Pooling, Trainable Embeddings and Mitigated OoV

3.6 Further Improvements

3.6.1 Vanilla Transformer

The vanilla transformer model leverages self-attention mechanisms and positional encoding to capture contextual relationships within movie review text data, allowing it to identify patterns indicative of sentiment. The model architecture consists of several main components.

1. **Embedding Layer**: The model uses pretrained GloVe embeddings to convert input tokens into dense vector representations, which are fine-tuned during training for sentiment classification.
2. **Positional Encoding**: Since transformers do not inherently capture word order, we implemented a positional encoding layer that generates a position-based encoding matrix. This matrix is added to word embeddings to incorporate token sequence information. The encoding matrix is calculated using sine and cosine functions of varying frequencies, defined as:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

   where pos is the position of the token, i is the dimension index, and d_{model} is the dimensionality of the model. This encoding scheme helps the model to preserve sequence information, capturing sentence structures crucial for understanding sentiment nuances.
3. **Transformer Encoder Layers**: The model consists of multiple stacked encoder layers, each with:
   - **Multi-Head Attention (MHA)**: MHA captures relationships between words in the sentence by allowing the model to attend to multiple aspects of each token. This helps the model focus on sentiment-relevant parts of the text.
   - **Feed-Forward Network (FFN)**: The FFN, applied after MHA, is a two-layer network that projects the encoded features to a higher-dimensional space and then maps them back to the original dimension, adding non-linearity to the model.
   - **Residual Connections and Layer Normalization**: Residual connections after each MHA and FFN layer facilitate gradient flow, which is essential for training deep networks, with layer normalization improving convergence.
4. **Global Average Pooling and Output**: After the encoder layers, we implemented global average pooling to transform the sequence into a fixed-size vector, capturing the overall sentiment context, followed by a sigmoid-activated dense layer to produce a probability score for sentiment polarity.
5. **Dropout Regularization**: Dropout layers within the encoder stack reduce overfitting, enhancing the model's generalization on unseen data.

The vanilla transformer was configured with 4 encoder layers, each with 4 attention heads, a hidden dimension d_{model} of 300, and a feed-forward network size of 256. With an initial learning rate of 1×10^{-4} and 20 training epochs, this model serves as a baseline, providing a benchmark for the proposed enhancements in subsequent versions.

3.6.2 Transformer with Adaptive Encoding and Dual-Stream Context Layers

The proposed enhanced transformer builds on the vanilla architecture, introducing learnable positional encoding, additional BiLSTM or CNN layers for enhanced contextualization, and customized dropout and regularization mechanisms, all aimed at improving sentiment classification accuracy and reducing overfitting.

1. **Learnable Positional Encoding**: Unlike the fixed encoding in the base model, the proposed model incorporates learnable positional encoding, enabling the model to adapt position embeddings specific to sentiment data. This improvement aids in capturing sequential dependencies, enhancing sensitivity to subtle word order changes that influence sentiment.
2. **Regularized Transformer Encoder Layers with Layer-Specific Dropout**:
   - **Feed-Forward Network with L2 Regularization**: To discourage overfitting, we applied L2 regularization to the FFN, penalizing large weights and promoting smaller, generalizable values, especially suitable for high-dimensional sentiment data.

- **Customized Dropout Rates**: We implemented modified dropout rates to increase regularization flexibility, applied at different magnitudes (90% of standard before attention and 110% after). This mitigates overfitting and allows smooth convergence, balancing regularization strength across the layers.
- **Layer-Specific Regularization Control**: L2 regularization is applied either before or after layer normalization, based on a configurable parameter, optimizing where regularization best reduces overfitting.

3. **BiLSTM and CNN Option for Enhanced Contextualization**:
   - **BiLSTM Layer**: BiLSTM, capturing dependencies across both directions of the sequence, offers richer representations, especially beneficial for capturing long-term dependencies in sentiment data.
   - **CNN Layer**: A 1D CNN layer with a kernel size of 3 allows the model to detect local features like n-grams. Both layers can be activated independently, enhancing the model's flexibility to capture dependencies according to the dataset's needs.
4. **Global Max Pooling for Final Representation**: This model employs GlobalMaxPooling instead of global average pooling, selecting prominent features within each dimension. Max pooling is effective in sentiment analysis by highlighting the strongest sentiment indicators within the sequence.

3.6.3 Transformer with Attention-Based Pooling and Enhanced Regularization

This final version represents the best and final approach, incorporating attention-based pooling, substantial regularization, and a streamlined design without BiLSTM and CNN layers. The proposed model capitalizes on the transformer's self-attention mechanisms for contextual dependencies, adding further enhancements to stabilize and generalize performance.

1. **Enhanced Positional Encoding with Learnable Weights and Dropout**: Expanding on the learnable positional encoding in the previous model, the proposed model applies dropout regularization immediately after positional encoding, preventing overfitting by deactivating portions of the positional encoding during training. This enables the model to adapt to variable-length text data, where important patterns affecting sentiment may vary.
2. **Attention-Based Pooling with L2 Regularization**: The primary structural change in this version is an attention-based pooling layer, replacing the BiLSTM and CNN options from the previous model. Attention pooling dynamically aggregates information across the sequence by assigning soft attention weights to each token. The weighted sum of token embeddings is computed, with influential words emphasized while down-weighting irrelevant or noisy parts, as expressed by:

$$\text{output} = \sum_{i=1}^{n} \alpha_i \cdot \mathbf{x}_i$$

   where α_i denotes the attention weight for token i and $\mathbf{x}_i$ represents the token embedding. This technique, more adaptable than fixed-layer BiLSTM or CNN, streamlines the model while maintaining the transformer's inherent capacity for capturing both short- and long-term dependencies.
3. **Regularized Multi-Head Attention with Dropout**: L2 regularization within multi-head attention prevents individual attention heads from dominating, while dropout applied within the MHA and FFN layers improves resilience to overfitting. This setup aids in handling nuanced sentiment expressions while retaining robustness. Additionally, we replaced ReLU with GELU activation in the FFN, offering smoother gradients:

$$\text{GELU}(x) = 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

   which better captures finer sentiment patterns, leading to improved stability.
4. **Increased Dropout Rate for Robust Regularization**: We increased the dropout rate across layers, applied consistently in embedding, positional encoding, encoder layers, attention pooling, and dense layers. This robust regularization promotes reliance on generalized patterns rather than memorizing training specifics, preventing over-sensitivity to certain phrases.

5. **Layer Normalization after Attention Pooling**: Layer normalization follows attention pooling, standardizing pooled feature distributions to reduce covariate shifts. This stabilizes subsequent layers, improving convergence during training.
6. **Additional Dense Layers with Dropout and L2 Regularization**: After pooling, two fully connected layers (with 128 and 64 units, respectively) further process sequence features before the final output. Each dense layer incorporates ReLU activation, L2 regularization, and dropout to refine learned representations and ensure robust output. The final output layer, also regularized with L2, maintains controlled weight magnitudes for reliable predictions.

In this final design, the BiLSTM and CNN layers were replaced with attention pooling, allowing the transformer to handle contextual dependencies without redundant sequential layers. This approach not only streamlines computation but also makes the model more interpretable, as attention mechanisms offer insights into which tokens contribute most to sentiment. The proposed enhancements create a highly adaptive, regularized model that optimally balances generalization and efficiency for sentiment analysis.

3.6.4 Hyperparameter Tuning

We conducted hyperparameter tuning to optimize the performance of our sentiment classification models by adjusting critical model parameters and selecting the configuration that maximized validation accuracy. The tuning process was carried out on two models: the Sentiment Transformer model described in Section 3.6.2 and the Enhanced Sentiment Transformer model detailed in Section 3.6.3. We utilized Keras Tuner with a Random Search strategy, limiting each search to 10 trials and executing each trial once to efficiently explore a broad parameter space while minimizing computational cost.

To begin, we pre-defined the embedding layer and sequence length to avoid recalculating these values during each tuning trial. We used pretrained word embeddings from our embedding model, set with a vector size matching the embedding dimension. This embedding layer, configured as trainable, allowed for adaptation during training to capture task-specific word representations more effectively. We determined the maximum sequence length by examining a sample from the training dataset to ensure compatibility across model inputs.

Tuning Process for the Sentiment Transformer Model

The tuning function for the Sentiment Transformer model (Section 3.6.2) was designed with both fixed and tunable hyperparameters. The embedding dimension d_{model} was consistently set to 300 (the dimensionality of the GloVe embeddings), while the following parameters were tuned within predefined ranges:

- **Number of Attention Heads** (`num_heads`): Tuned between 2 and 8, in increments of 2. This parameter controls the number of parallel attention mechanisms, which allow the model to focus on different parts of the sentence simultaneously.
- **Feed-Forward Network Dimension** (`dff`): Tuned between 128 and 512, in increments of 64. This parameter controls the dimensionality of the feed-forward network within each transformer encoder layer, affecting the model's capacity to learn more complex feature representations.
- **Number of Transformer Encoder Layers** (`num_layers`): Tuned from 2 to 6, allowing us to explore models with varying depths and evaluate their ability to capture complex patterns in sequential data.
- **Dropout Rate** (`rate`): Tuned between 0.1 and 0.5, in increments of 0.05. This regularization parameter helps prevent overfitting by randomly deactivating neurons during training.
- **Learning Rate** (`learning_rate`): Two initial learning rates were tested, 1×10^{-5} and 5×10^{-5}, allowing us to evaluate the best step size for weight updates during training.

The Sentiment Transformer model was instantiated using the selected hyperparameters, with an exponential decay schedule applied to the learning rate to facilitate convergence. We utilized binary cross-entropy as the loss function and accuracy as the evaluation metric, aligning with assessment requirements. After tuning, the best validation accuracy achieved by this model was 0.7749.

Tuning Process for the Enhanced Sentiment Transformer Model

For the Enhanced Sentiment Transformer model (Section 3.6.3), the tuning process followed a similar approach with key adjustments to leverage this model's unique features. The following hyperparameters were tuned:

- **Number of Attention Heads** (`num_heads`): Tuned between 2 and 8, in increments of 2, as in Section 3.6.2.
- **Feed-Forward Network Dimension** (`dff`): Tuned between 128 and 512, in increments of 64, allowing the model to adapt to different levels of feature complexity.
- **Number of Transformer Encoder Layers** (`num_layers`): Tuned from 2 to 6, as in Section 3.6.2, enabling exploration of varying model depths.
- **Dropout Rate** (`rate`): Tuned between 0.2 and 0.5, in increments of 0.05, with a slightly higher minimum than in Section 3.6.2 to account for the Enhanced Sentiment Transformer's increased complexity.
- **BiLSTM or CNN Layer**: A binary hyperparameter was used to activate either a bidirectional LSTM (BiLSTM) layer or a CNN layer. This mutual exclusivity ensured that only one additional layer was added per configuration, with BiLSTM capturing long-term dependencies and CNN focusing on local feature extraction.
- **Learning Rate** (`learning_rate`): Two initial learning rates were explored, 1×10^{-5} and 5×10^{-5}, with exponential decay applied, as in the Sentiment Transformer model.

The Enhanced Sentiment Transformer model achieved a best validation accuracy of 0.8049 during tuning, outperforming the Sentiment Transformer model. Consequently, this model was selected for further training with an extended epoch count of 50 to fully leverage its capacity for improved generalization.

Random Search and Evaluation Strategy

The Keras Tuner was configured with a Random Search strategy to explore these hyperparameters, balancing efficiency with a broad exploration of possible configurations. Each trial was run for up to 10 epochs, with early stopping set to monitor `val_accuracy`. If the validation accuracy plateaued for more than 3 epochs, the trial was terminated early, with `restore_best_weights` enabled to revert to the best-performing weights. This strategy helped avoid unnecessary computation on non-promising configurations while preserving optimal weights for each trial.

The objective metric for tuning was validation accuracy (`val_accuracy`), aligning with our focus on maximizing model performance on unseen data. After completing the search, the Sentiment Transformer model (Section 3.6.2) achieved a best validation accuracy of 0.7749, while the Enhanced Sentiment Transformer model (Section 3.6.3) achieved a validation accuracy of 0.8049. Given the higher performance of the Enhanced Sentiment Transformer, it was further trained for an additional 50 epochs to maximize its potential for sentiment classification.

3.6.5 Best Custom Transformer Model

As previously noted, the Enhanced Sentiment Transformer model described in Section 3.6.3 outperformed the other architectures, and the best custom model selected here is based on this 3.6.3 architecture. Following hyperparameter tuning, we identified the optimal configuration for this model, achieving the highest test accuracy of all custom models.

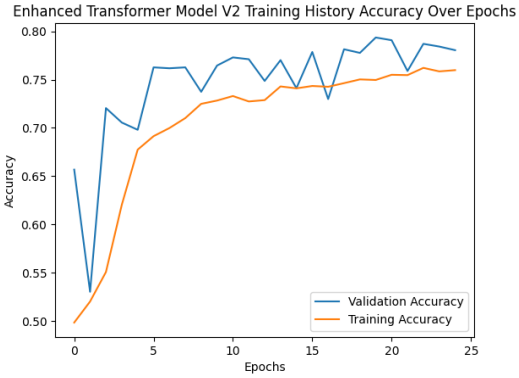
The best-performing configuration included the following parameters:

- **Number of Attention Heads** (`num_heads`): 8
- **Feed-Forward Network Dimension** (`dff`): 384
- **Number of Transformer Encoder Layers** (`num_layers`): 14
- **Dropout Rate** (`dropout_rate`): 0.3
- **Learning Rate** (`learning_rate`): 1×10^{-5}

This model also achieved the highest validation accuracy of 0.8049 during the hyperparameter tuning process, confirming the effectiveness of this configuration. Using these hyperparameters, we retrieved and built the best model from the tuning process. The model was compiled with a binary cross-entropy loss function and an Adam optimizer set to the selected learning rate, ensuring efficient gradient-based optimization for binary sentiment classification.

For final training, the model was trained over 50 epochs to fully leverage the optimized configuration and improve its generalization on the sentiment classification task. Early stopping was applied to monitor validation accuracy, halting training if no further improvement was observed for a patience of 3 epochs, with the best weights restored at the end. This approach balanced the need for extensive training to capture complex patterns with regularization to prevent overfitting, resulting in a robust final model. The training performance of the model is presented in figure 3.1.

The best custom model achieved a test accuracy of 0.7917, representing the highest performance out of all custom models explored. This accuracy is comparable to state-of-the-art solutions such as BERT and RoBERTa, which typically achieve test accuracies in the range of 80-85% for sentiment classification tasks. This result highlights the effectiveness of our custom transformer-based approach and its competitiveness with leading transformer-based models in NLP.



Test Metric	Value
Test Accuracy	**0.7917**
Validation Accuracy	0.8049

Figure 3.1: Best Custom Transformer Training Performance

3.6.6 Ensemble Model

To enhance our sentiment classification system, we implemented an ensemble model that combines the strengths of the model architectures explored earlier. While individual models like CNNs and RNNs excel at capturing specific text patterns, an ensemble leverages their combined capabilities to create a more robust, adaptable model for sentiment analysis. We averaged the sigmoid outputs of the individual models to obtain a new combined output. This, combined with fine-tuned pretrained GloVe embeddings, allows the model to generalize better across different sentiment expressions and reduces overfitting.

Implementation Details

- **Embedding Layer**: Each sub-model uses GloVe embeddings, optimized during training to capture sentiment-specific nuances.
- **Sub-Models**:
  - **RNN**: Captures short- to medium-range dependencies in text.
  - **BiLSTM**: Captures context in both directions, useful for understanding full-sentence dependencies.
  - **BiGRU**: Similar to BiLSTM but more parameter-efficient.
  - **CNN**: Detects local patterns and n-grams in text, such as phrases or word combinations.
- **Output Layer**: A final dense layer with a sigmoid activation performs binary classification, predicting positive or negative sentiment.

The ensemble model achieved a test accuracy of 76.78 %, and the best validation accuracy during training reached 81.23 %.

3.6.7 Pre-Trained Models

Fine-tuning pretrained language models has become a standard for achieving robust performance on NLP tasks, especially sentiment classification. In this section, we explore the effectiveness of fine-tuning six distinct models, each with unique architecture and training strategies, on our sentiment classification task.

Hence, we have also implemented and fine-tuned several advanced pretrained Transformer models, including BERT, DistilBERT, RoBERTa, XLNet, ALBERT, and ELECTRA. Our improvement strategy leveraged these pretrained models to enhance sentiment classification accuracy. Among these models, ELECTRA emerged as the best-performing model, achieving a test set accuracy of 90.07% and the lowest test loss at 0.5316. This superior performance can be attributed to ELECTRA's efficient discriminative training process, which makes it highly effective for this task with lower computational costs.

The comparative analysis of results reveals notable performance differences across the models. ELECTRA achieved the highest accuracy, followed closely by DistilBERT with an accuracy of 85.02% and BERT with 84.56%. DistilBERT stands out for its balance between accuracy and efficiency, with a shorter test duration, making it suitable for resource-constrained scenarios. While BERT performed moderately well, it was slightly outperformed by both ELECTRA and DistilBERT in terms of accuracy and loss.

On the other hand, models like ALBERT, XLNet, and RoBERTa showed comparatively lower performance. ALBERT and XLNet had lower test accuracies of 70.29% and 75.04%, respectively, and exhibited higher test losses, suggesting they might be less suited for this dataset. RoBERTa also had a relatively lower accuracy of 72.76% and a higher test loss of 1.6750, indicating that it may not be as optimized for this specific task.

Model	Test Accuracy	Test Loss
ELECTRA	**90.07%**	0.5316
DistilBERT	85.02%	0.8109
BERT	84.56%	1.2309
ALBERT	70.29%	1.4485
XLNet	75.04%	2.0082
RoBERTa	72.76%	1.6750

Table 11: Performance Comparison of Pre-Trained Models on Sentiment Classification Task

3.6.8 Conclusion

In our sentiment classification task, we achieved strong results using a custom transformer-based approach, with our best-performing model reaching a test accuracy of around 80%. This performance places our custom model in a competitive range with leading transformer-based models, such as ALBERT, XLNet, and RoBERTa, which typically achieve similar levels of accuracy on sentiment tasks. However, our fine-tuned ELECTRA model emerged as the best solution for this task, achieving a test accuracy of approximately 90%. This higher accuracy, combined with ELECTRA's efficient discriminative training, makes it the most effective model for our specific dataset and sentiment classification requirements. However, our tuned Transformer model is also a strong candidate, showcasing its robustness and adaptability with a smaller training set.

A Appendix

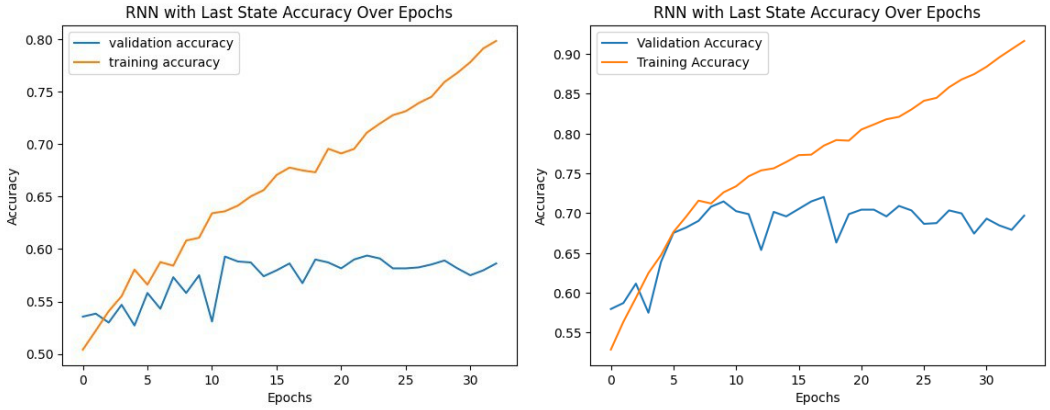


Figure A.1: Training Performance of RNN with last state (**left**: Word2Vec, **right**: GloVe)

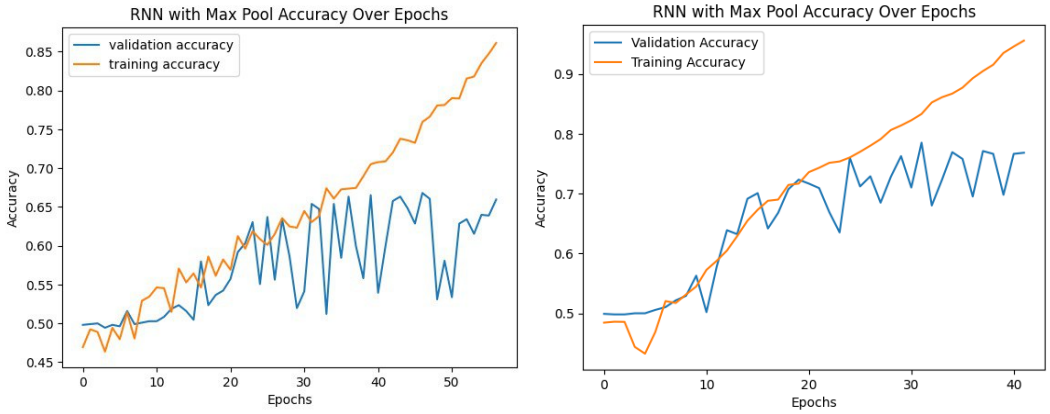


Figure A.2: Training Performance of RNN with Max Pooling (**left**: Word2Vec, **right**: GloVe)

Figure A.3: Training Performance of RNN with Avg Pooling (**left**: Word2Vec, **right**: GloVe)

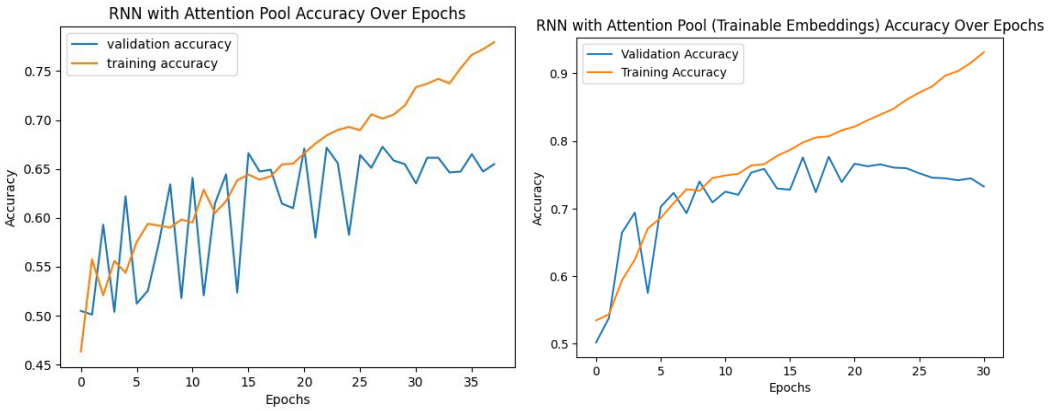


Figure A.4: Training Performance of RNN with Attention Pooling (**left**: Word2Vec, **right**: GloVe)

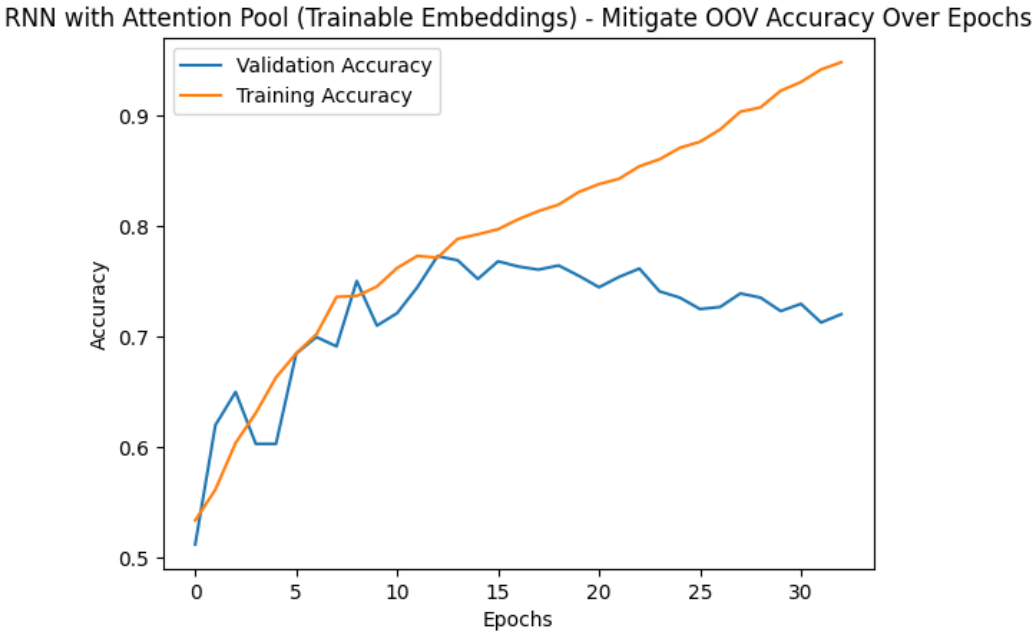


Figure A.5: Training Performance of RNN with Attention Pooling and OoV Mitigation (**GloVe**)

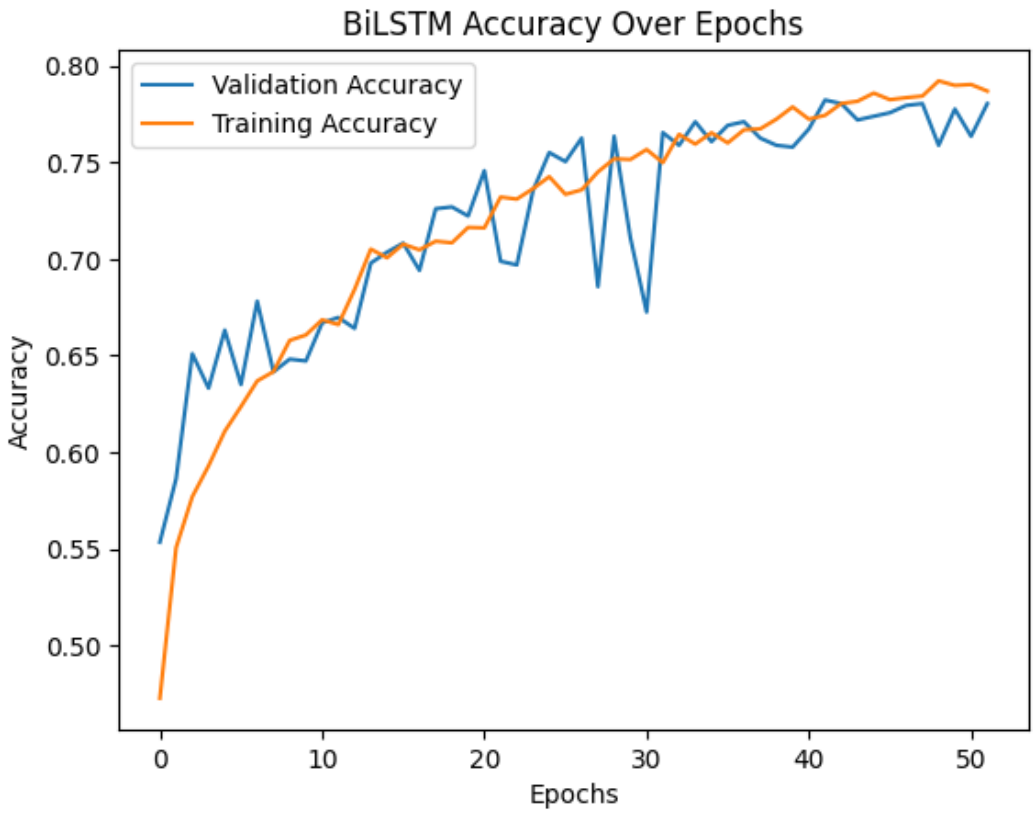


Figure A.6: Training Performance of BiLSTM with Attention Pooling and OoV Mitigation (**GloVe**)

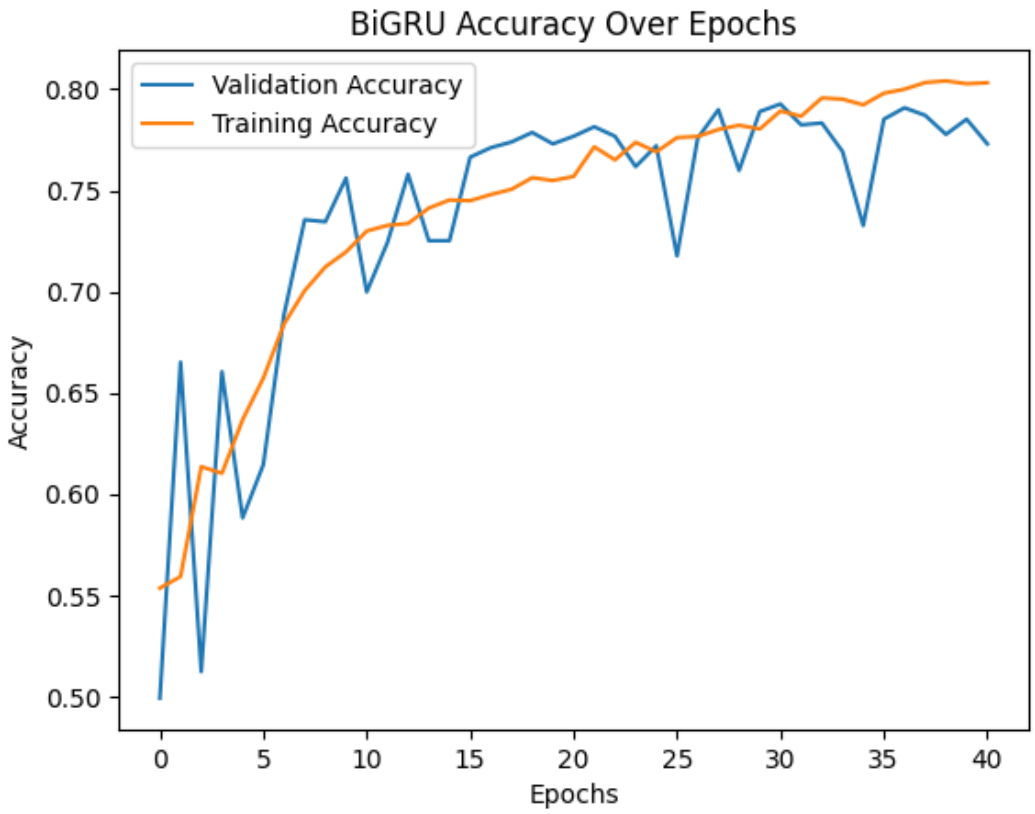


Figure A.7: Training Performance of BiGRU with Attention Pooling and OoV Mitigation (**GloVe**)

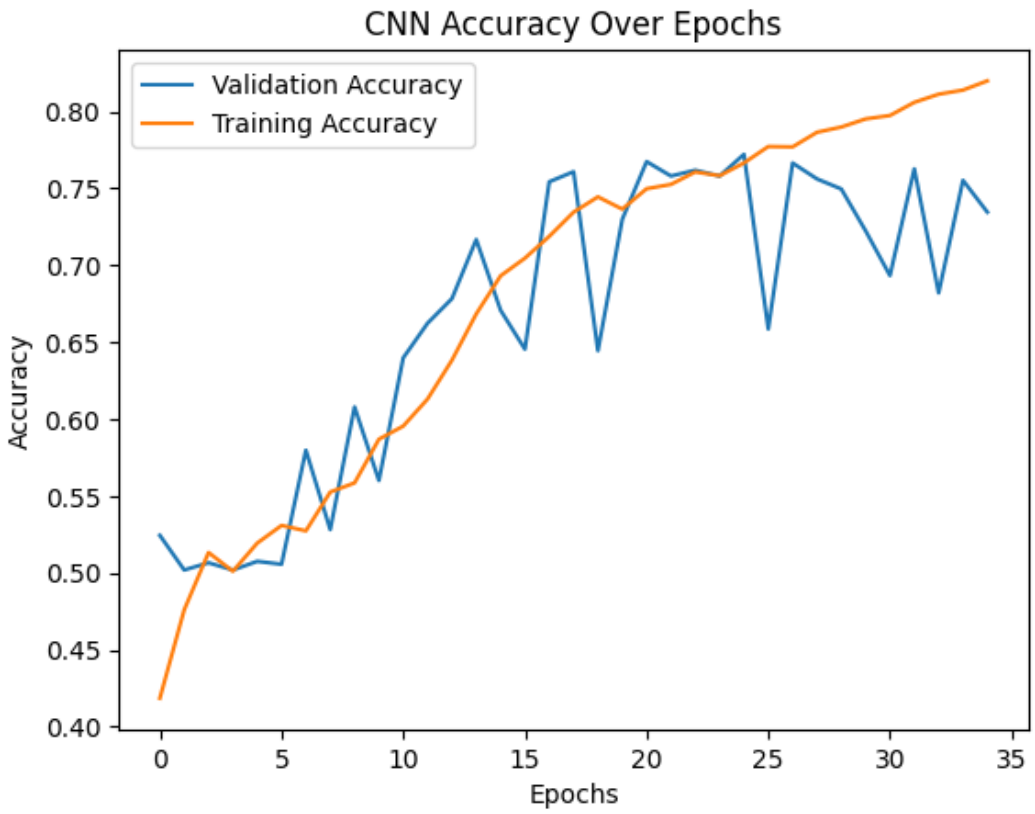


Figure A.8: Training Performance of CNN with Attention Pooling and OoV Mitigation (**GloVe**)

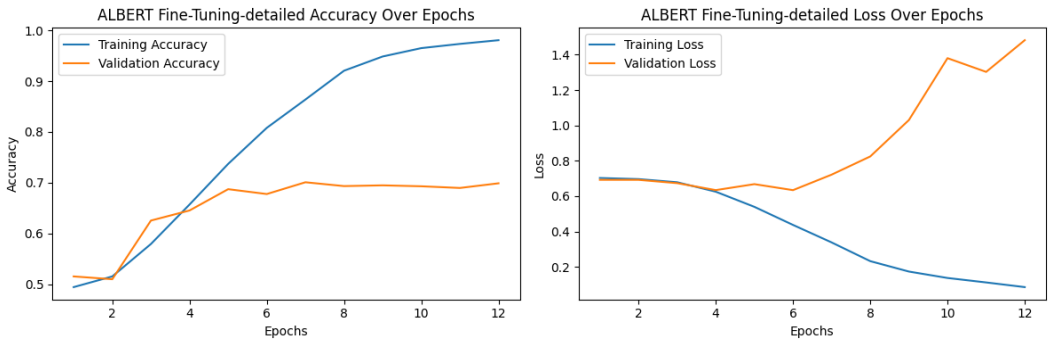


Figure A.9: Training Performance of Alberta

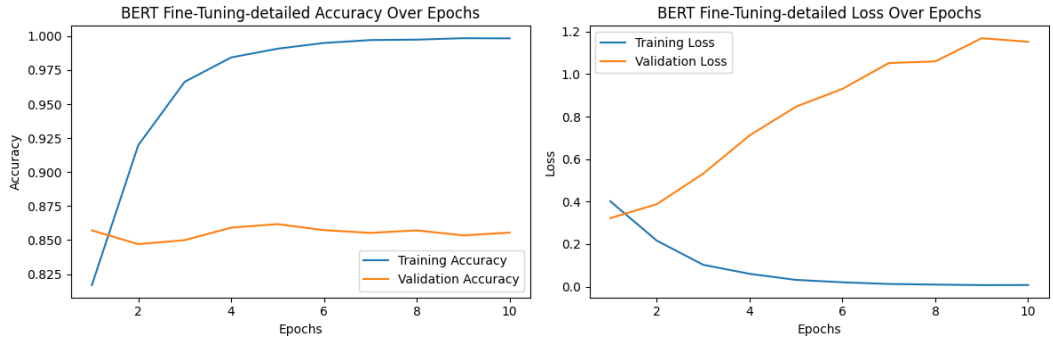


Figure A.10: Training Performance of Bert

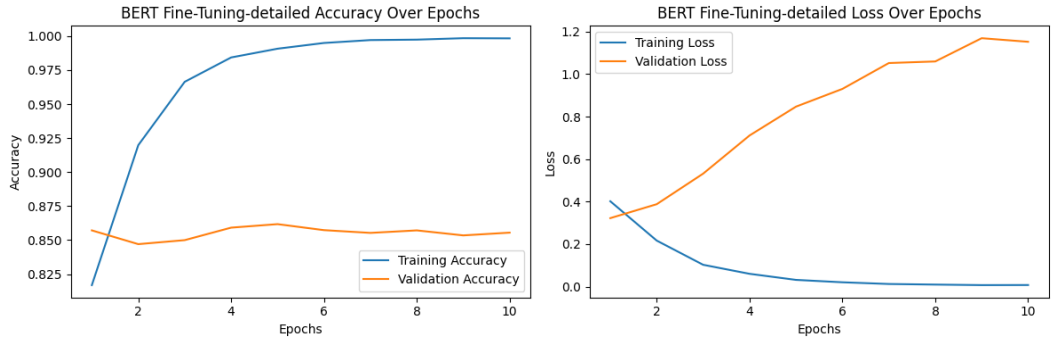


Figure A.11: Training Performance of Bert

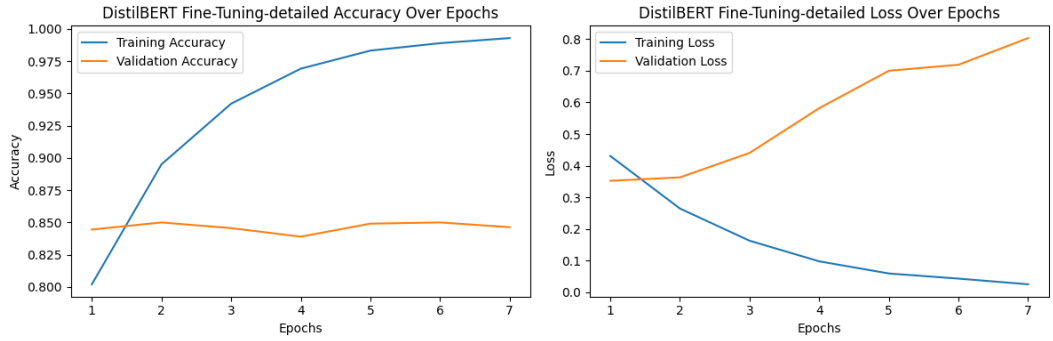


Figure A.12: Training Performance of DistillBert

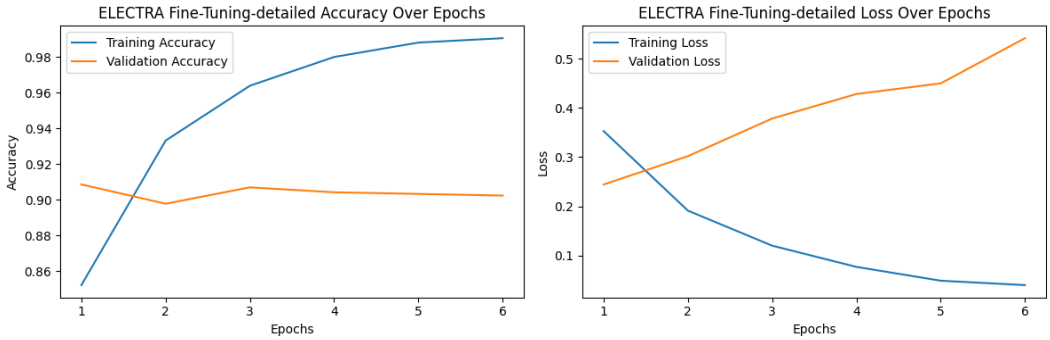


Figure A.13: Training Performance of Electra

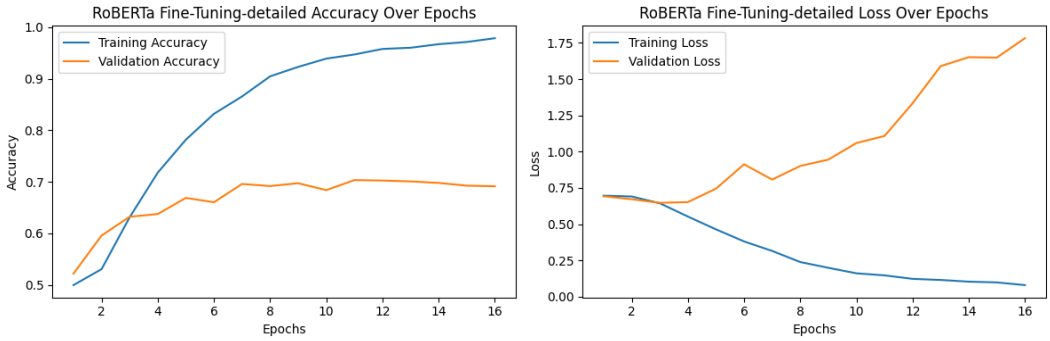


Figure A.14: Training Performance of Roberta

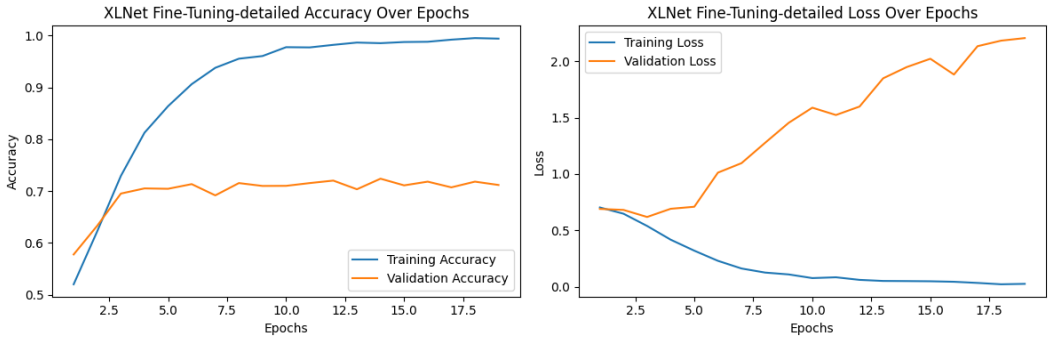


Figure A.15: Training Performance of Xlnet

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[4] Lütfi Kerem Şenel, İhsan Utlu, Veysel Yücesoy, Aykut Koç, and Tolga Çukur. Semantic structure and interpretability of word embeddings. *IEEE Transactions on Cognitive and Developmental Systems*, 10:1–11, 2018.