CZ4045/SC4002 Natural Language Processing

Course Assignment

Group 30

Name	Matric No	Contribution
Leong Kit Ye	U2121111K	Part 1
Abdul Rahim Sahana	U2240147F	Part 2
Banerjee Mohor	U2222858E	Part 3
Lim Jun Rong, Ryan	U2220466E	Part 2
Augustine Jesuraj Senchia Gladine	U2222429C	Part 2
Jeyaseelan Harish Vasanth	U2122375C	Part 3

Introduction

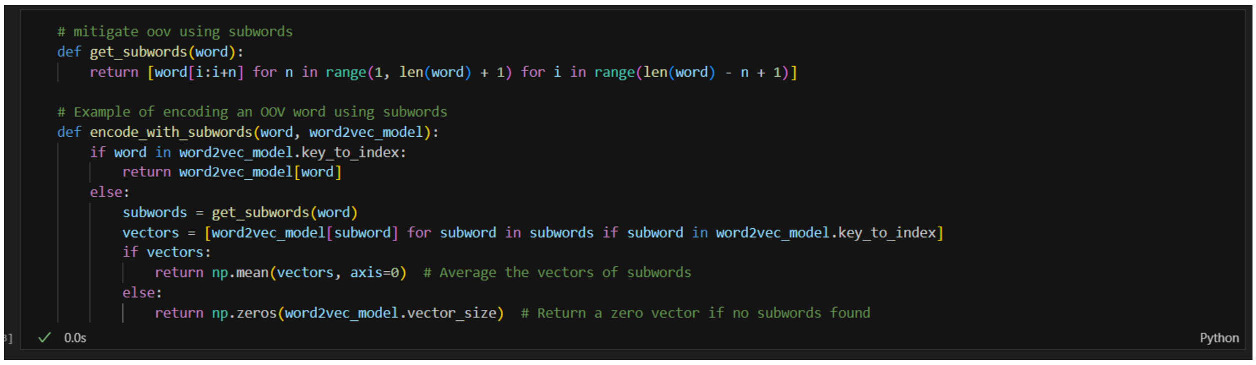
Natural Language Processing (NLP) is a branch of artificial intelligence focused on interactions between computers and human language, enabling machines to understand, interpret, and generate meaningful human language. One key application of NLP is sentiment analysis, which involves determining the emotional tone behind a series of words.

In this assignment, we use the Rotten Tomatoes dataset, consisting of movie reviews from critics and audiences, to analyse sentiments and classify reviews as positive or negative, providing insights into public perception of films.

Part 1: Dataset Preparation

To prepare the dataset, we used the NLTK word tokenizer to tokenize texts. The initial vocabulary size of the training set was 18,010. We removed punctuation and filtered out stopwords based on the NLTK corpus, as the Word2Vec model does not include stopwords. This reduced the vocabulary size to 17,879.

We used the Word2Vec model fasttext-wiki-news-subwords-300, trained on Google News. Initially, there were 1,960 out-of-vocabulary (OOV) words (3,600 without removing stopwords or punctuation). To address OOV words, we implemented subword embeddings, generating all possible subwords for each word and averaging valid subword embeddings. This reduced the number of OOV words to 1. The code snippet for this method is shown below.



```python
# mitigate oov using subwords
def get_subwords(word):
    return [word[i:i+n] for n in range(1, len(word) + 1) for i in range(len(word) - n + 1)]

# Example of encoding an OOV word using subwords
def encode_with_subwords(word, word2vec_model):
    if word in word2vec_model.key_to_index:
        return word2vec_model[word]
    else:
        subwords = get_subwords(word)
        vectors = [word2vec_model[subword] for subword in subwords if subword in word2vec_model.key_to_index]
        if vectors:
            return np.mean(vectors, axis=0)  # Average the vectors of subwords
        else:
            return np.zeros(word2vec_model.vector_size)  # Return a zero vector if no subwords found
```

Figure 1: Code snippet for generating subword embedding representation for OOV words

Part 2: Model Training & Evaluation - RNN

We trained and evaluated a simple Recurrent Neural Network (RNN) for sentiment classification. RNNs are effective for processing sequential data, making them suitable for sentiment analysis, where the word order and context in a sentence influence the sentiment. We also applied **average pooling** across the RNN's outputs at each time step to capture the sequence's overall feature representation. The Average Pooling method computes the mean of all hidden states across time steps, providing an overall semantic representation of the sentence.

2a) Finding the configuration for the best model

We used Random Search to find the optimal configuration for the model, tuning the number of layers, hidden size, learning rate, batch size, and dropout rate. Random Search sampled various hyperparameter combinations and evaluated model performance on the validation set.

Training was monitored by tracking validation accuracy after each epoch. Early stopping was applied to prevent overfitting, halting training when validation accuracy failed to improve for 5 consecutive epochs, leading to convergence at 15 epochs.

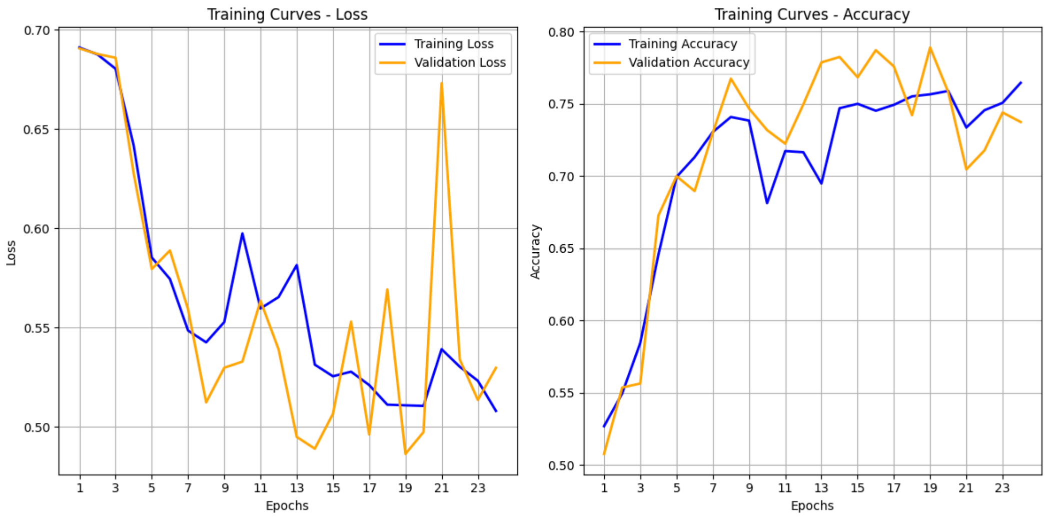


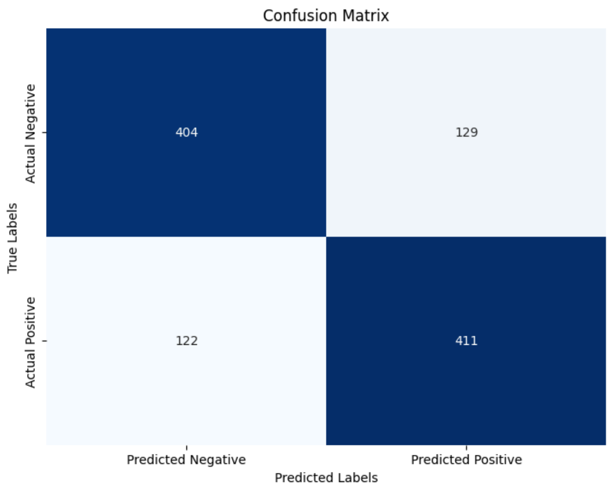
Figure 2: Training and validation loss and accuracy histories for the RNN model with the highest validation accuracy

Number of hidden layers	**Hidden layer size**	**Learning rate**	**Batch size**	**Dropout rate**
2	256	0.001	128	0.9

Table 1: Final configuration of hyperparameters based on best RNN model

2b) Reporting the accuracy scores

Given the small training dataset, the model likely didn't learn enough features for accurate predictions and showed signs of overfitting, as the best model required a high dropout rate.

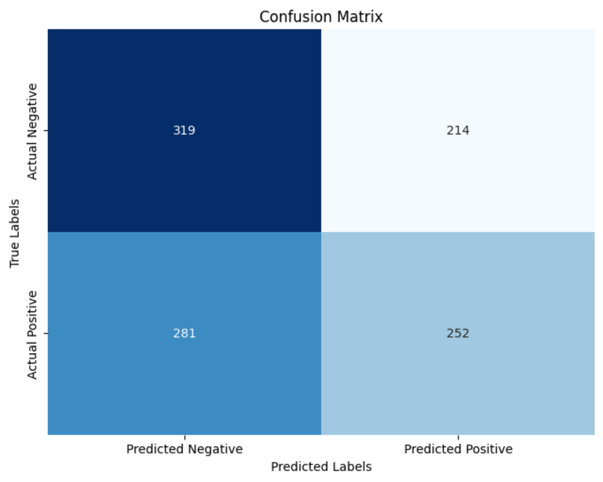


Test Accuracy: 0.76
Precision: 0.76
F1 Score: 0.77

Figure 3: Confusion matrix and for RNN - Average Pooling model with highest validation accuracy

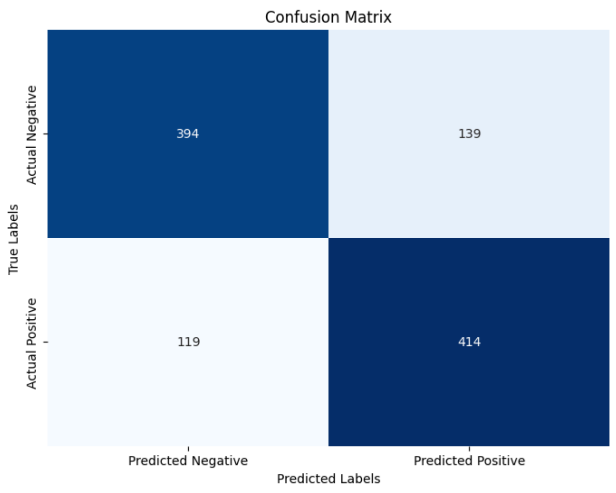
2c) Other methods of deriving final sentence representation

Apart from **Average Pooling** shown above, we also compared two sentence representation methods: **Last Time Step Representation** and **ConcatPooling**. In the Last Time Step Representation approach, the RNN processes the sequence, and only the last time step's hidden state is used as the sentence representation. This final hidden state is assumed to encapsulate the context of the entire sentence, which is then classified via a fully connected layer. For ConcatPooling, it combines multiple aspects of the RNN's output by concatenating the first hidden state, the last hidden state, and the max-pooled representation across all time steps. This concatenated vector captures information from the beginning, end, and the most salient features of the sequence, which is then passed through a fully connected layer for classification.



Accuracy: 0.54
Precision: 0.54
F1 Score: 0.47

Figure 4: Confusion matrix for the RNN - Last Time Step Representation on Test Set



Accuracy: 0.76
Precision: 0.75
F1 Score: 0.76

‘

Figure 5: Confusion matrix and performance statistics for the RNN - ConcatPooling model on Test set

Overall, the **Average Pooling** method was the best performing method for deriving the final sentence representation, followed by ConcatPooling and Last Time Step Representation. This is likely because, by averaging over all time steps, it captures more global features of the sentence, making it more robust to variations in word order or sentence structure. Averaging over all time steps can reduce the effect of outliers or noisy signals that might distort the final representation.

Part 3: Model Enhancements

3a) RNN with trainable embeddings

Pretrained embeddings, like Word2Vec, capture basic language patterns and word relationships, but may not fully grasp domain-specific nuances, such as sentiment in movie reviews. Fine-tuning these embeddings ensures they are optimised for sentiment classification. For each word, the model retrieves its embedding or, for out-of-vocabulary (OOV) words, averages embeddings of known subwords. This approach helps the model handle unknown words. By using trainable embeddings, we fine-tune the word representations during training, adapting them specifically for the sentiment classification task.

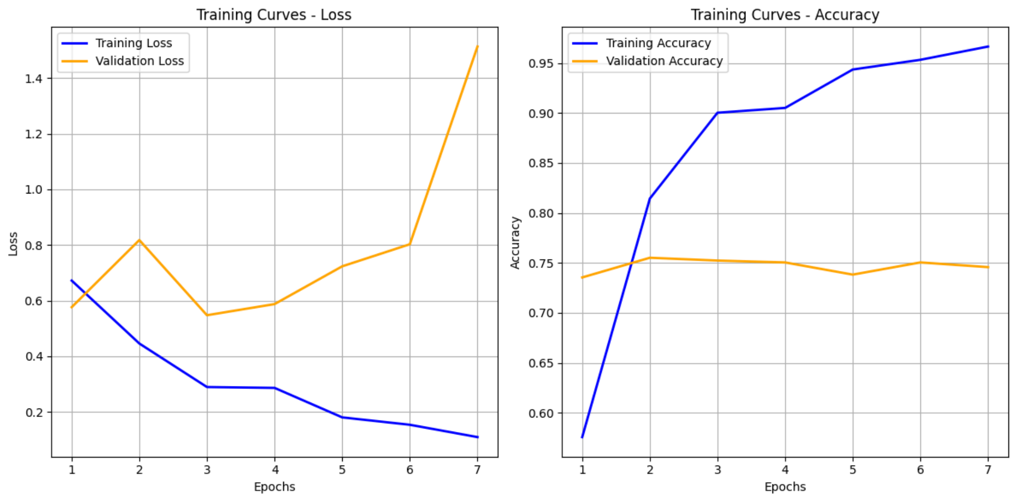
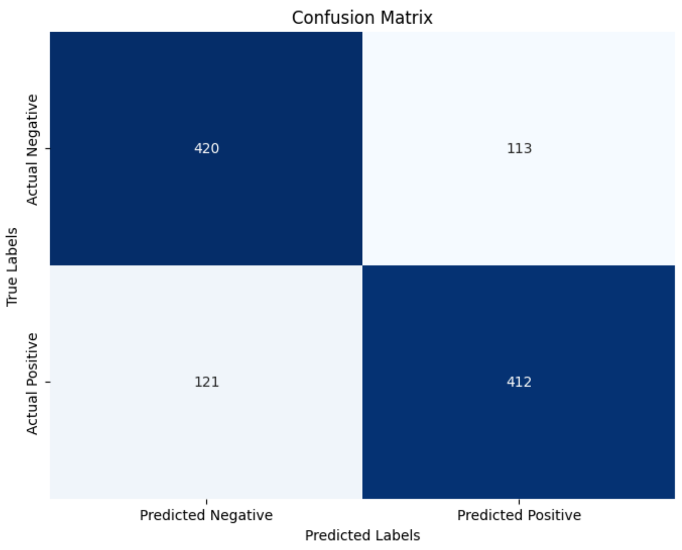


Figure 6: Training and validation loss and accuracy histories for the RNN with trainable embeddings



Test Accuracy: 0.78
Precision: 0.78
F1 Score: 0.78

Figure 7: Confusion Matrix and performance statistics for RNN Trainable Embeddings performance on test set

After this, we experimented with our subword embedding solution to mitigate the influence of OOV vocabulary.

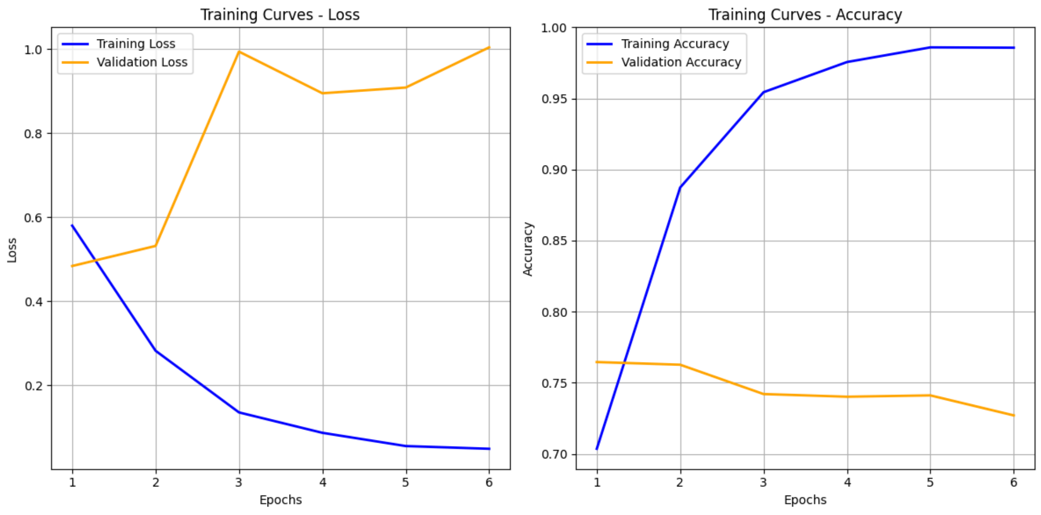
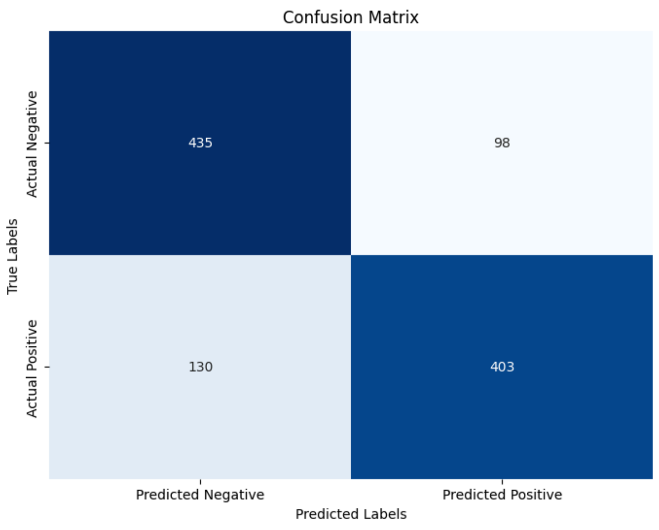


Figure 8: Training and validation loss and accuracy histories for the RNN with OOV Mitigation



Test Accuracy: 0.79
Precision: 0.80
F1 Score: 0.78

Figure 9: Confusion Matrix for RNN with OOV Mitigation performance on test set

3b) BiLSTM

A BiLSTM Model is able to understand information between words by processing sequences in both forward and backward directions. This allows it to use more context than a simple RNN or a simple LSTM model would. For sentiment classification, a BiLSTM can perform better than a regular RNN because it considers the whole sentence context, making it easier to detect sentiment cues for movie reviews, especially when they depend on surrounding words.

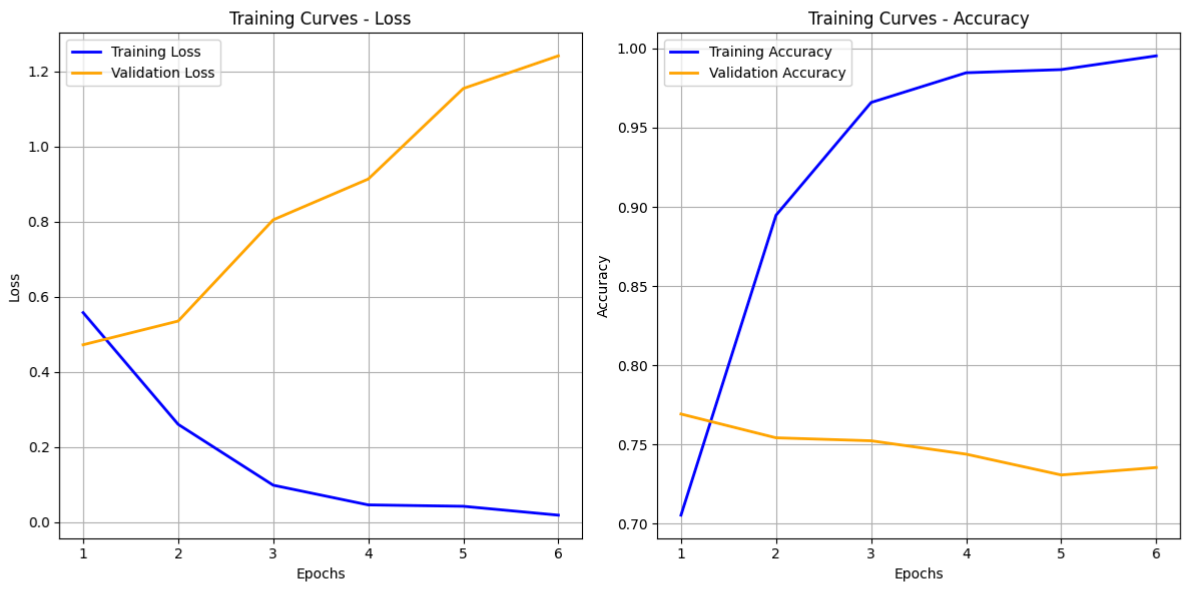
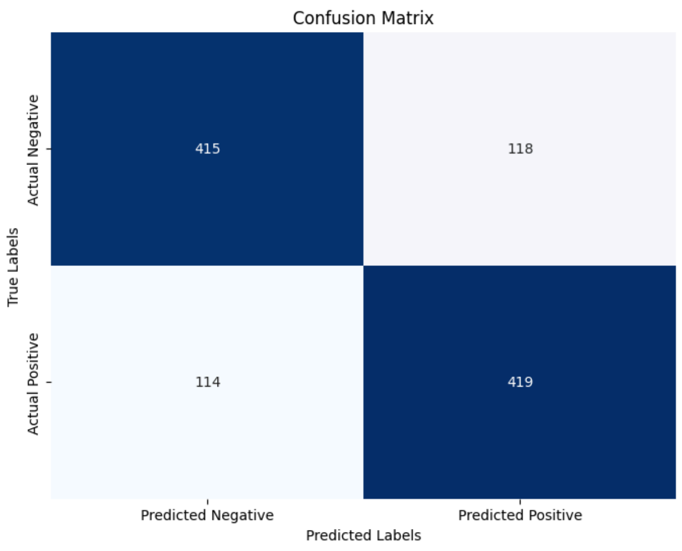


Figure 10: Training and validation loss and accuracy histories for BiLSTM



Test Accuracy: 0.78
Precision: 0.78
F1 Score: 0.78

Figure 11: Confusion matrix for BiLSTM performance on test set

3c) BiGRU

Similar to BiLSTM , The BiGRU model architecture leverages a bidirectional GRU (Gated Recurrent Unit) layer to capture both forward and backward dependencies in the input text, while being simpler and faster due to having fewer parameters. We experiment with BiGRUs to see if they are better suited for this dataset.

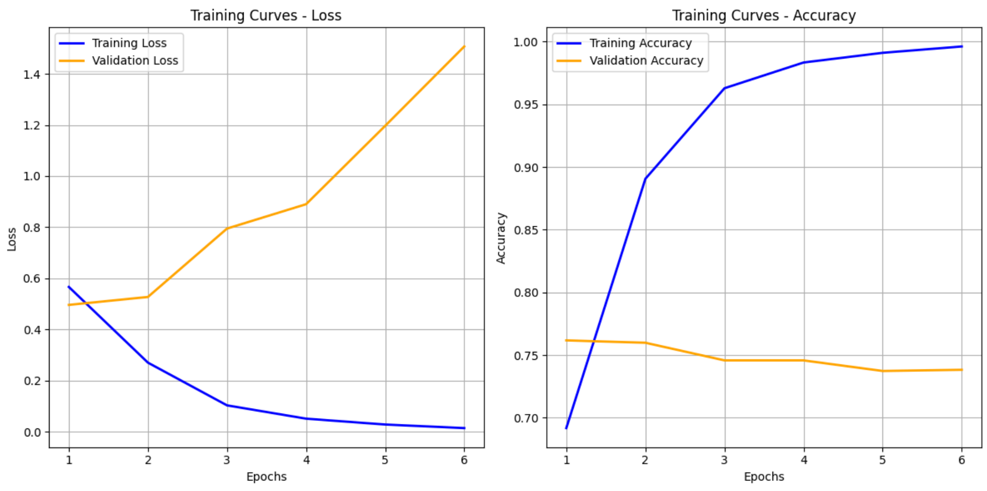
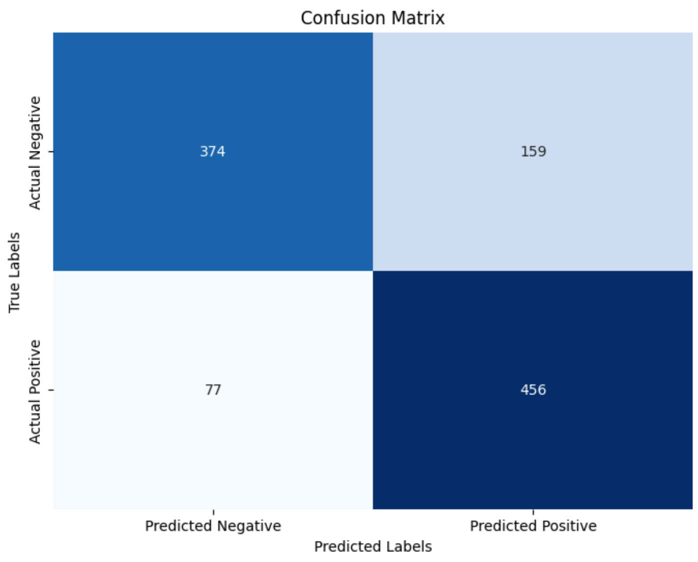


Figure 12: Training and validation loss and accuracy histories for BiGRU



Test Accuracy : 0.78
Precision: 0.74
F1 Score: 0.79

Figure 13: Confusion matrix for BiGRU performance on test set

3d) CNN

The CNN model captures local patterns in text, which are crucial for sentiment analysis. It uses multiple filter sizes (e.g., 3, 4, and 5) to capture different n-gram features, with smaller filters focusing on short phrases and larger ones detecting longer expressions. Each filter creates a feature map highlighting specific patterns across the sentence. After convolution, max pooling is applied to retain the most significant features. The pooled outputs from all filters are concatenated, forming a comprehensive representation of the sentence.

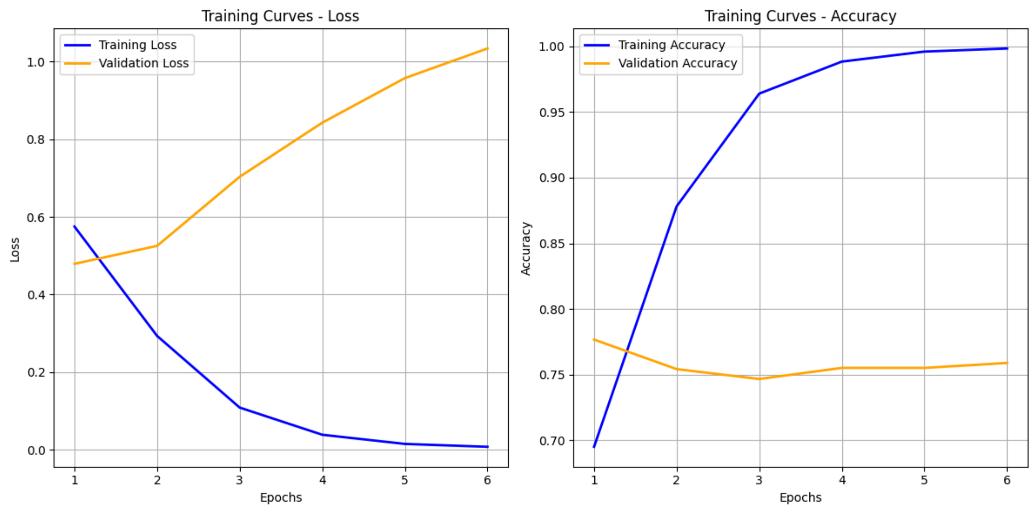
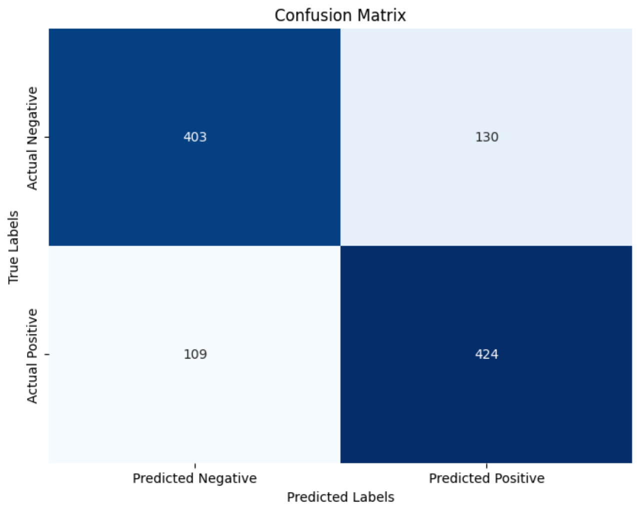


Figure 14: Training and validation loss and accuracy histories for CNN

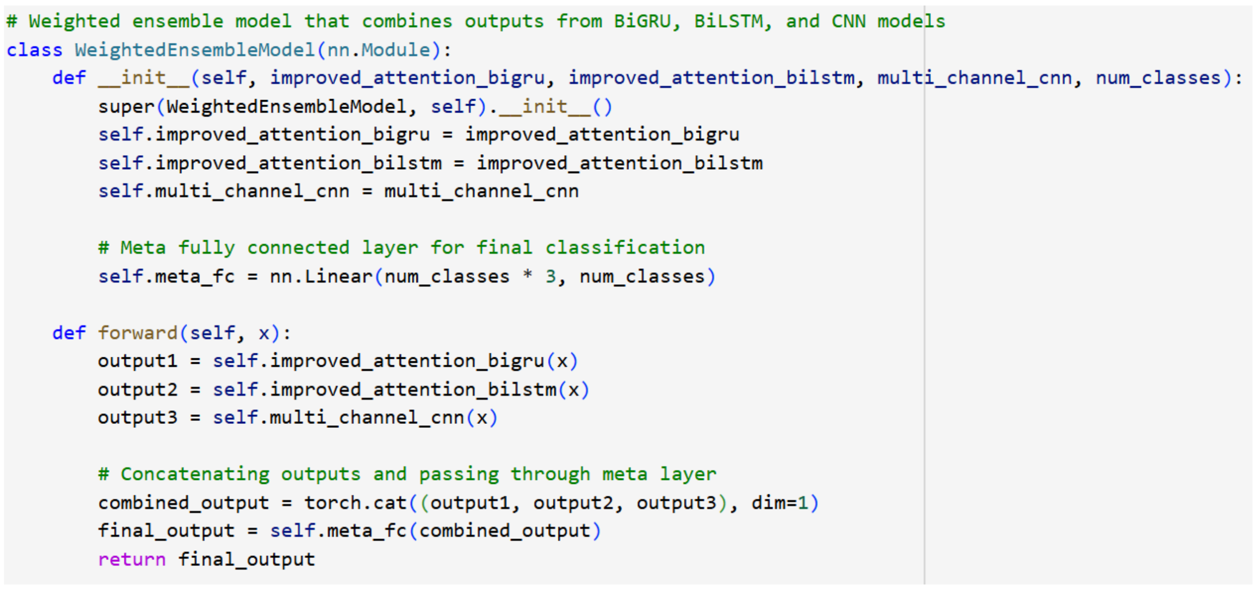


Test Accuracy: 0.78
Precision: 0.77
F1 Score: 0.78

Figure 16: Confusion matrix for CNNperformance on test set

3e) Further Improvement Strategy : Ensemble Model Architecture.

The final improvement involved developing a weighted ensemble model that combines BiGRU, BiLSTM, and CNN sub-models to capture sequential, contextual, and local patterns in text. Ensemble models enhance accuracy, reduce bias, and minimise overfitting by integrating multiple models' predictions. These three models, which performed well in previous experiments, were enhanced with a multi-head attention mechanism for BiGRU and BiLSTM to focus on different parts of the input sequence, capturing distant dependencies. The CNN model was modified into a Multichannel CNN with varying kernel sizes (e.g., 3, 4, 5) to capture different n-gram features. After max pooling, the most significant features are retained for each channel. The outputs from these sub-models are concatenated and passed through a fully connected "meta" layer, which dynamically balances each sub-model’s contributions, allowing the ensemble to effectively combine all patterns for robust sentiment prediction.



```python
# Weighted ensemble model that combines outputs from BiGRU, BiLSTM, and CNN models
class WeightedEnsembleModel(nn.Module):
    def __init__(self, improved_attention_bigru, improved_attention_bilstm, multi_channel_cnn, num_classes):
        super(WeightedEnsembleModel, self).__init__()
        self.improved_attention_bigru = improved_attention_bigru
        self.improved_attention_bilstm = improved_attention_bilstm
        self.multi_channel_cnn = multi_channel_cnn

        # Meta fully connected layer for final classification
        self.meta_fc = nn.Linear(num_classes * 3, num_classes)

    def forward(self, x):
        output1 = self.improved_attention_bigru(x)
        output2 = self.improved_attention_bilstm(x)
        output3 = self.multi_channel_cnn(x)

        # Concatenating outputs and passing through meta layer
        combined_output = torch.cat((output1, output2, output3), dim=1)
        final_output = self.meta_fc(combined_output)
        return final_output
```

Figure 17: Code to define weighted ensemble model

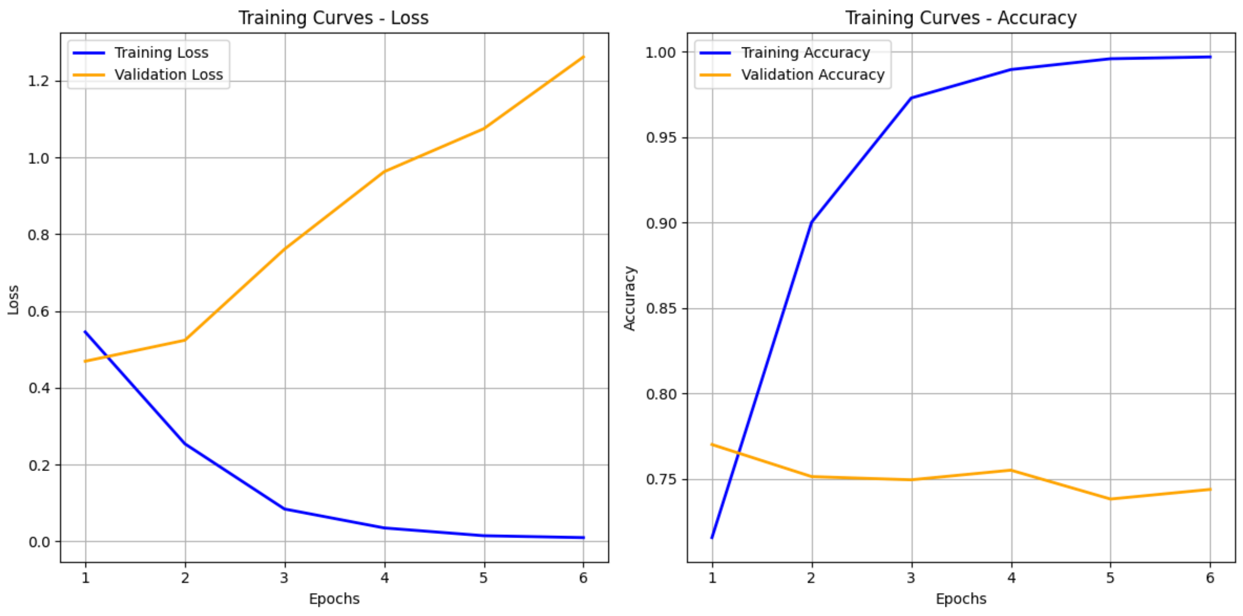
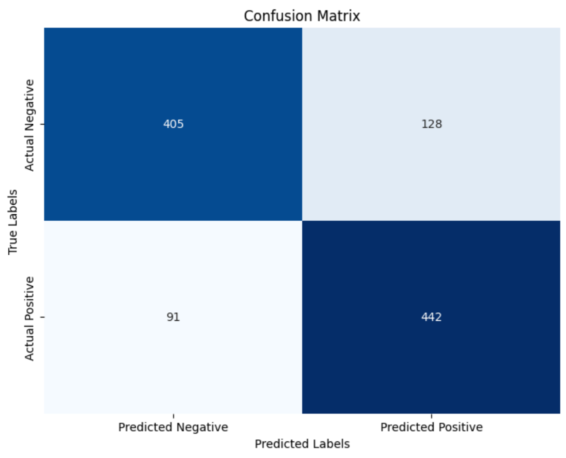


Figure 18: Training and validation loss and accuracy histories for weighted ensemble model



Test Accuracy: 0.79
Precision: 0.78
F1 Score: 0.80

Figure 19: Confusion matrix for performance of weighted ensemble model on test set

3f) Analysis

We observed that the RNN models had a comparable performance to BiLSTM and BiGRU models. This is despite the standard limitations of standard RNNs in handling sentiment classification tasks, where RNNs suffer from vanishing gradients during backpropagation, especially when processing longer sequences, resulting in gradients becoming too small to update the weights effectively, so the RNN struggles to learn long-term dependencies. Unlike RNNs, BiLSTMs and BiGRUs use specialised gating mechanisms to manage information from both past and current inputs. This bidirectional structure enables them to preserve contextual information effectively, leading to more accurate sentiment predictions. Even so, BiLSTMs and BiGRUs performed similarly to the RNNs with improvements. We believe this is due to the implementation of concat pooling, which allows RNNs to learn semantic relationships by combining both the hidden states from each timestep as well as the max and average pooling of those states. By providing the RNN with this comprehensive view, it can better recognize patterns and dependencies that might otherwise be missed in a traditional pooling approach.

When analysing our results with BiLSTMs and BiGRUs, we observed that these models were highly prone to overfitting, even after implementing dropout and normalisation layers to mitigate this issue. Despite our efforts, the model's performance plateaued early in training (after 1-3 epochs) , likely due to the relatively limited number of training samples (8,530), which may have restricted the model's ability to generalise effectively to new data. This suggests that additional regularisation techniques such as data augmentation or a larger dataset could improve model robustness and prevent overfitting.

Interestingly, CNNs, which are traditionally popular in image processing, have proven effective in the sentiment analysis task, even outperforming the RNNs. CNNs are designed to capture localised patterns in data through convolutional filters, which makes them effective for recognizing n-grams (e.g., short phrases) in text. Furthermore, they are more parameter-efficient because they focus on local features instead of processing sequentially. This allows CNNs to process data in parallel rather than depending on previous steps, making them computationally lighter and faster.

The final ensemble model achieved a test accuracy of 79.45%, indicating strong performance on the test dataset. The model demonstrates a balanced performance with high accuracy, precision, and F1 score, making it reliable for general sentiment classification. Both classes (positive and negative) are well-represented, indicating a robust model for the dataset's distribution.

From these experiments, it is clear that the choice of model architecture greatly impacts performance in sentiment analysis. Standard RNNs were inadequate, while more complex models like BiLSTM, BiGRU, and CNN were more successful at handling the nuances of sentiment classification than simpler RNNs. The ensemble model, in particular, provided the best balance and reliability across metrics, making it an ideal model for this sentiment analysis task on the given dataset.

However, it's also important to consider the computational cost, the cost of training the ensemble model is 3x the cost of training a regular BiLSTM , BiGRU or CNN model. Is the 1 ~ 2 % improvement in accuracy worth the additional training required for the ensemble model? It depends on the specific application and available resources.