Assignment 2 - Group: A2 – Group 111

Group members:
1. **540212639**
2. **530813967**

Abstract

In this project assignment, our group select the dataset for sentiment analysis in this assignment, which is derived from Twitter data. The procedure begins with data preparation, when extraneous characters are removed to enhance the dataset's quality. For embedding, we utilize Word2Vec, which translates text into numerical representations that correspond to the dataset's word context. To analyze the sentiment, we employ three distinct deep learning models: Convolutional Neural Network (CNN), Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (BiLSTM). Based on the preprocessed and merged Twitter data, grid search was used to modify all of the models in order to improve their sentiment categorization performance. With the use of this technique, we were able to methodically investigate several configurations and determine which ones worked best for each model. According to our findings, the BiLSTM model outperformed the others overall, particularly in terms of accuracy and precision (80.78% accuracy). On the other hand, hypertuning resulted in slight performance drops in CNN and BiLSTM, while no appreciable changes were observed in the GRU model following tuning. This implies that hypertuning must be done carefully since it can result in overfitting. To further enhance performance, future research may investigate more sophisticated models like BERT.

Introduction

This project will give an understanding of how to implement the deep learning model into some tasks in a given dataset, such as classification, regression and clustering. Our group choose classification problem, that makes our group choose the dataset that will be determining sentiment analysis from over 300,000 tweets where we will classify if this text belongs to neutral, negative or positive sentences. The dataset consists of six attributes which are, sentiment (binary label), tweet ID, date, query, username, and the tweet's text. Analysing sentiment analysis, especially if we are talking about tweet in social media has numerous real-world applications, including brand reputation management, customer feedback analysis, and public opinion tracking. It allows businesses, political campaigns, and public health officials to gain valuable insights into public sentiment in real time[1, 2] . For example, companies can know customer opinions on their products, and public health authorities can monitor the public's response to crisis communications, such as during the COVID-19 pandemic[3].

To have better understanding for this project, we will look for some previous techniques that already used in classification of sentiment analysis. The techniques are from the simple one, which uses semantic approaches like lexicon-based approaches[4], moving on the more complex model using some machine learning algorithm such as Support Vector Machine (SVM) or Naïve Bayes[5], and lately with the advanced deep learning namely Recurrent Neural Network, LSTM and the most popular model right now Transformers that have a more superior performance by capturing the intricate context of language[6].

In this project, our group will try several deep-learning algorithms to make the classification from the dataset. The **outline methods** of our group are shown below:

- First, the dataset will be analysed by knowing what information the dataset has. Identifying the characteristics of the dataset, such as the number of samples and the difficulties of the dataset.
- Second, we will do some pre-processing to the dataset, which includes selecting the important features, removing unnecessary characters or words and doing tokenization.
- Third, we will split the dataset and train the data into three distinct deep-learning models that our group chose. The models are Convolutional Neural Network (CNN), Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).
- After looking at the performance of the based model, we will improve the model by tuning the parameters such as batch_size, epochs, layer of each model, etc, to get the best value for each model using grid search.
- Finally, we will evaluate and compare each model that gives the best performance in classifying the sentiment analysis from the tweet dataset.

Based on the experiment, we found some **key findings** in our modelling. The three models, LSTM, GRU, and CNN, performed comparably and produced good outcomes. Bidirectional LSTM (Bi-LSTM), which collects context from both directions in the text and enables a more sophisticated interpretation of the tweet content, is a crucial component of the LSTM model's minor performance advantage over the other models. Despite their impressive performance, all models showed symptoms of overfitting after a few training epochs. This was because they became overly fixated on the training set, which hindered their ability to learn from fresh data. Our solution to this was to use a tuner search to do hyperparameter tuning. This helped to improve the model's parameters and lessen overfitting, which in turn improved the classification results' accuracy and resilience.

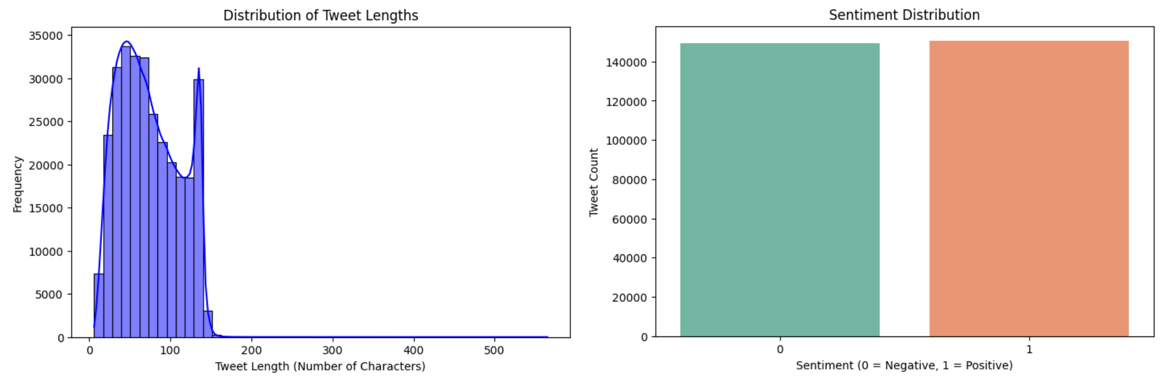
Data

A. Data Description

The dataset that we applied was from Kaggle about sentiment analysis that had already sampled 300,000 tweets, each labeled for sentiment classification negative or positive[7]. Every tweet's information in the collection comes with six attributes: *Binary label* denoting either positive (1) or negative (0) attitude, *ID*: Every tweet's individual identification number, *Date*: The timestamp the tweet was posted under. *Question*: Details about the query (for most records put "NO_QUERY"). Username: The name of the person tweeting this *Text*: The tweet's actual content, which forms the primary basis for sentiment assessment. *Text* feature is the most crucial in determining the sentiment as it contains the raw tweet.

To know the **characteristics** of the dataset, we will do some exploration data analysis (EDA) of the dataset. We tried to create several plot to find the insight and patterns of the dataset, based on our exploration, we found some key characteristics on the dataset after EDA, which are:

- We found a variation in the tweet length. The tweets range start from very short sentences, maybe a single word to a really long text (Figure 1a). The variability in the length of tweets and content can affect how well the model performs, especially for the model that is really sensitive to long sequence length. We might do some data pre-processing, like truncating the long tweet, and we will explain the details in data pre-processing techniques.



(a) (b)

Figure 1a. EDA for exploring tweet lengths **1b**. EDA for exploring the distribution

- Second characteristics that we found was on the class distribution. The dataset's roughly balanced sentiment class distribution, that have value of the negative sentiment and positive sentiment around approximately 15,000 (Figure 1b). It helps to prevent the bias that is frequently present in imbalanced datasets. Because of this balance, the models are better able to generalize and are not skewed toward any one sentiment type[8].
- The dataset comes with many noises, such as irrelevant tokens (e.g., URLs, user mentions, hashtags) and misspellings, which could degrade the model's performance if not addressed. During the data cleaning, we will remove and filter this noise to make the model perform better.
- We also found that the tweet contains some informal language, *slang,* emoticons, and abbreviations that will also be a problem if we do some cleaning for this issue.

Based on the characteristics, we found some **challenges or difficulties** that might come up in the modelling process. We can see that the problem that we will face is the text noises and the informal language from the tweet. These text noises don't contribute to the overall sentiment of the tweet but are frequently present in social media text. Additionally, the informal language used in tweets characterized by slang, abbreviations, and non-standard grammar can make it difficult for models to capture meaning and sentiment accurately. Data pre-processing needs to be done to address this issues and make sure the data for training the model is clean.

B. Data Pre-Processing

Data pre-processing is needed to address the challenges that consists in the dataset. **The first** thing that our group did was select the most important features, which are the sentiment label and the text. **Next,** we will do the text-cleaning. **The justification of text-cleaning** is to normalize the language and eliminate unnecessary noise. This is crucial for social media datasets, as tweets frequently include extraneous symbols, links, and user-specific references. This procedure, which is in line with recommended techniques for text preprocessing for sentiment analysis, decreases the dimensionality of the text data and enhances model focus on terms that convey sentiment[9, 10]. This clean text is defined into a function that will do some pre-processing such as:

- **Lower-casing**: all the text on the tweet will be converted into lowercase to ensure uniformity because sentiment analysis is case-sensitive[11]. For example, we convert he word "NICe" into "nice".
- **Removing-URL:** Text on Twitter usually has some irrelevant information, like URL, that doesn't affect the sentiment analysis. Removing them ensures that the model focuses on meaningful words[12].
- **Removing mentions:** Mentions are irrelevant for sentiment analysis because they relate to individual users. Eliminating them stops the model from concentrating on the actual content of the tweet and stops it from learning user-specific patterns[13].
- **Removing Punctuation and Special Characters:** If punctuation and special letters are left in, they can confuse the model and detract from the text's meaning. The model is able to better understand the actual words by concentrating solely on letters and whitespace[13].

Next, the text will be converted into a vector that can be inputted in machine learning, which is called **tokenization**. Each word in a tweet is treated as a token, and this tokenized data is used to represent the text numerically[14]. After that, we will apply **Word2Vec** as an embedding technique. In order to help the model comprehend how words are used in comparable settings, Word2Vec records the semantic links between words. Word2Vec, for instance, allows the model to better understand emotion even when different words are employed because the vector representations of the words "happy" and "joyful" are similar[15]. **The justification** why We used this approach because Word2Vec's semantic similarity feature enables words with related meanings to express the same emotion. As a result, generalization and prediction accuracy are increased, and the algorithm is able to comprehend the underlying meaning of the words used in tweets[16]. Some techniques, such as stemming and lemmatization, are not used by our group because we think their removal could make the text less rich and make it more difficult for the model to accurately identify sentiment.

Methodology

In this part, we will explain the details of deep-learning algorithms that will be used to be our machine-learning model for classifying sentiment analysis. Those models are **Convolutional Neural Network (CNN), Gated Recurrent Unit (GRU)** and **Long-Short Term Memory**.

A. Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is one of the popular deep learning models to do classification tasks. In the context of sentiment analysis, CNNs are capable of recognizing critical n-grams (e.g., phrases or word sequences) that are essential for sentiment detection. The convolutional layers slide filters across the input text, identifying patterns such as sentiment-bearing word combinations. The feature map is reduced to the most significant signals by the GlobalMaxPooling layer, which is subsequently passed to a completely connected layer for final classification[17, 18]. In addition, CNN also fit our model well since it can capture local relationships with fewer parameters than recurrent networks due to their efficient processing of sequential input. Because of their parallelizable processes, they are also more efficient to train, which makes them a good option for predicting the classification of large datasets like sentiment analysis on Twitter[19]. Those are the **strengths** that CNN had, making it a solid reason to choose CNN for our project.

Still, CNN is not a perfect model and comes with some **disadvantages,** For example, CNNs have a tendency to focus more on local patterns and struggle to recognize long-range relationships in text. This increases the difficulty of global context understanding for CNNs operating without additional layers, including Recurrent Neural Networks (RNNs) or attention mechanisms. CNN's performance may be significantly affected by hyperparameters such as the quantity and size of filters, which, if not set properly, could add complexity and complicate the system[20 ,21].

To have a better understanding of the principles and architectures of the CNN, the figures of the block diagram below (Figure 2) show how our CNN model works. We implement 1D CNN for sentiment classification:

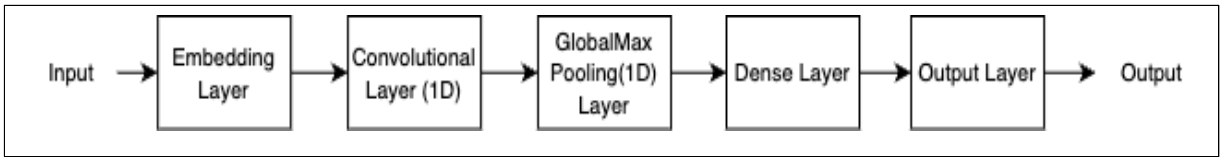


Figure 2. CNN 1D architecture diagram for our model.

- **Embedding Layer**: This layer will map the words to a dense vector that will capture their semantic relationships, which will help the model understand the context of the word[14]. As we stated in the data pre-processing section, we will use pre-trained Word2Vec embeddings to capture the model effectively.
- **Convolutional Layer:** This layer will apply filters to capture local text pattern, which in this case, will detect the n-grams of the word that will be useful for sentiment analysis[18]. The model uses A filter size of 32, which balances learning capacity and model simplicity, while a kernel size of 3 is chosen to detect important trigrams in the text.
- **GlobalMaxPooling Layer:** layer pooling minimize dimensionality by focusing on the most important attributes, which will aid the model in identifying important trends[18]. By ensuring that only the most pertinent sentiment signals are forwarded to the following layer, GlobalMaxPooling lowers the possibility of overfitting.
- **Dense Layer:** Non-linearity is introduced by dense layers, which combine features to provide more difficult decisions[21]. The model in this project uses 10 units to prevent overfitting while maintaining enough capacity to combine learned features efficiently.
- **Output Layer:** This layer has a function to map the output. The model uses a sigmoid function that is suitable for binary classification. The sigmoid function will turn the value into 0 (negative) or 1 (positive)[22].

We will also do some **hyperparameter tuning** using grid search to improve the performance of CNN by adjusting the value of **filters**, **kernel size**, and **dense units (**more details in Jupyter Notebook). Filters control the depth of feature extraction, kernel size adjusts the receptive field for feature detection, and dense units affect the model's ability to learn non-linear feature combinations. These adjustments are aimed at improving model performance[23].

B. Gated Recurrent Unit (GRU)

Recurrent neural networks (RNNs) of the Gated Recurrent Unit (GRU) type are intended to effectively capture sequential dependencies in data, such as in text classification. GRUs are especially good at managing long-range dependencies since they help mitigate the vanishing gradient issue that regular RNNs are known for. With fewer gates (reset and update gates) than Long Short-Term Memory (LSTM) units, GRUs are computationally lighter while maintaining the capacity to recognize significant sequence patterns[24]. The reason GRU is selected is because of some **strengths** and **advantages** that it offers. First, GRU is efficient in the handling of sequential data and also computationally lighter since they use fewer gates, making them faster to train[24]. GRUs are also useful for sentiment analysis because they can detect long-range dependencies in text, particularly when the context of a sentence as a whole is significant[25]. Because GRU have fewer parameters, GRUs reduce the risk of overfitting, particularly in smaller datasets[26]. However, there are still some **weaknesses** in this model. GRU might not perform as well as LSTMs in tasks requiring complex memory retention over very long sequences, but the training time is still long enough compared to CNN[21, 26].

The architecture that our group created uses **2 (two) GRU units** to create the sentiment analysis classification. (Figure 3). In sentiment analysis, the use of two GRU layers facilitates the capture of both short- and long-term dependencies. While the second layer enhances the model's capacity to handle intricate structures like sarcasm or negation, the first layer finds local patterns. By iteratively improving text representations, stacking GRUs improves generalization and increases its efficacy for subtle sentiment classification. Multiple recurrent layers have been shown to enhance performance in text tasks [27, 28].

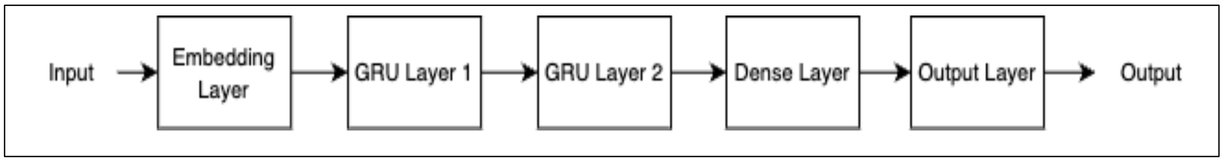


Figure 3. GRU 2 Layer architecture diagram for our model

- **Embedding Layer**: Similar to the CNN model, we will turns the text into dense vector using Word2Vec to capture the semantic similarities[14].
- **GRU Layer 1:** This is the first layer that will capture sequential dependencies across the text. A GRU layer use 32 units and the "return_sequences=True" parameter is used to stack GRU layers, as it ensures that the full sequence of outputs is passed to the next GRU layer[22].
- **GRU Layer 2:** With 16 (sixteen) units, the second GRU layer processes the first GRU's enhanced data. "Return_sequences=False" tells this layer to only output the last hidden state, condensing the sequence into a single vector for additional categorization[22].
- **Dense Layer:** Dense layer will apply the non-linearity function to the process that will combined features from previous layers.
- **Output Layer:** Then into the final layer same with CNN, sigmoid activation to output the probability of sentiment (positive or negative).

We will grid search for parameter **GRU units 1, 2**, and **dense units (**more details in Jupyter Notebook). Sequential pattern learning is affected by the first and second GRU units' memory and feature extraction capacities across layers. The fully connected layer's dense units determine the model's capacity to blend learned features into output. By tuning these parameters, the model can handle difficult sequential tasks better[30].

C. Long-Short Term Memory (LSTM)

The next model that we will implement is LSTM. LSTM is another form of RNN, which also designed to handle sequential data while addressing the vanishing gradient problem that occurs in traditional RNNs. The input, forget, and output gates of the LSTM gating mechanism enable the network to selectively store or forget information as needed, which makes it useful for learning long-range dependencies in sequences, like text. LSTMs perform well in jobs like sentiment analysis where context over longer sequences is essential because of their capacity to retain and reject information[31]. LSTM has a few varieties model that can be built, but in this project, we will implement the **Bidirectional-LSTM (Bi-LSTM).** The reason Bi-LSTM is chosen is because the architecture is fit for sentiment analysis. By processing the input sequence in both forward and backward directions, Bi-LSTM expands upon ordinary LSTM. Accordingly, a more thorough comprehension of sentence structure and meaning is made possible by Bi-LSTM's ability to record word relationships in both past and future contexts. In sentiment analysis, sentiment is frequently determined by words and phrases dispersed across the sentence (contextual understanding). Bi-LSTM, however, can offer a more comprehensive understanding of the relationships between the sentence's many components[32]. Another **advantage** of Bi-LSTM is that When it comes to capturing crucial elements of a phrase for tasks like sentiment analysis, BI-LSTM performs exceptionally well at managing lengthy dependencies and typically outperforms other models in text tasks by maintaining relevant information across sequences[33]. Bi-LSTM also has some weaknesses. One of the **disadvantages** that Bi-LSTM has is computational complexity. Because bi-LSTMs process the sequence in both ways, double the memory and processing time required, they demand more computer resources than unidirectional LSTMs or GRUs. Bi-LSTM also has a large number of parameters that leads them to easily overfit for smaller datasets[32, 33].

The figure below (Figure 4) will explain the architecture of Bi-LSTM model that our group choose to classify the sentiment analysis.

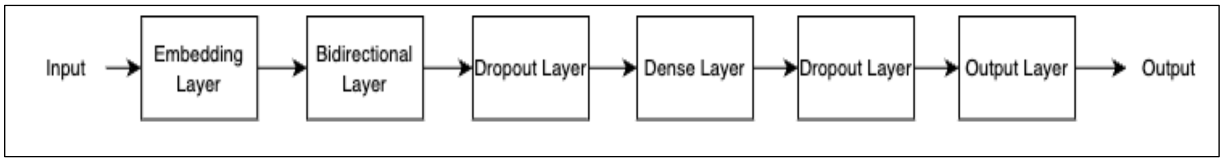


Figure 4. Bi-LSTM Layer architecture diagram for our model

- **Embedding Layer**: Same with other models, the first step of the process is to convert the text into dense vector in the embbeding layer[14].
- **Bidirectional Layer:** This is the layer where the text will be processed both forward and backward to capturing the dependencies in both directions. In our model we used 128 units of Bidirectional layer to provide a more complete understanding of the sentence structure [32].
- **Dropout Layer:** Dropout layer is used to prevent overfitting from the model. We use 2 (two) dropout layers before we map the input into the output layer[35].
- **Dense Layer:** A dense layer with 128 units adds non-linearity to the model and enables it to learn complex interactions between features extracted by the Bi-LSTM.
- **Output Layer:** The output layer uses a sigmoid activation function to produce a probability score between 0 and 1, suitable for binary sentiment classification.

The **hyperparameter tuning** for the Bi-LSTM are **LSTM units**, **dense units**, and **dropout rate** (more details in Jupyter Notebook)**.** LSTM units determine the model's sequential data memory capacity, affecting its long-term retention. Dense units regulate output layer non-linear representation learning, and the dropout rate randomly drops neurons during training to reduce overfitting and improve model generalization. Tuning these hyperparameters improves the model's sequential performance[36].

Experimental Results

Our group succesfullty train the model and get the best parameter for each model. When training the models (CNN, Bi-LSTM and RNN-GRU), we apply early stopping by comparing the training and validation loss, if the validation loss is higher than the training loss then the iterations (epoch) stop. The early stop is used to avoid the overfitting from the models. A sample graph (Figure 5) of the Bi-LSTM training loss and accuracy are shown below (another model in Jupyter Notebook).

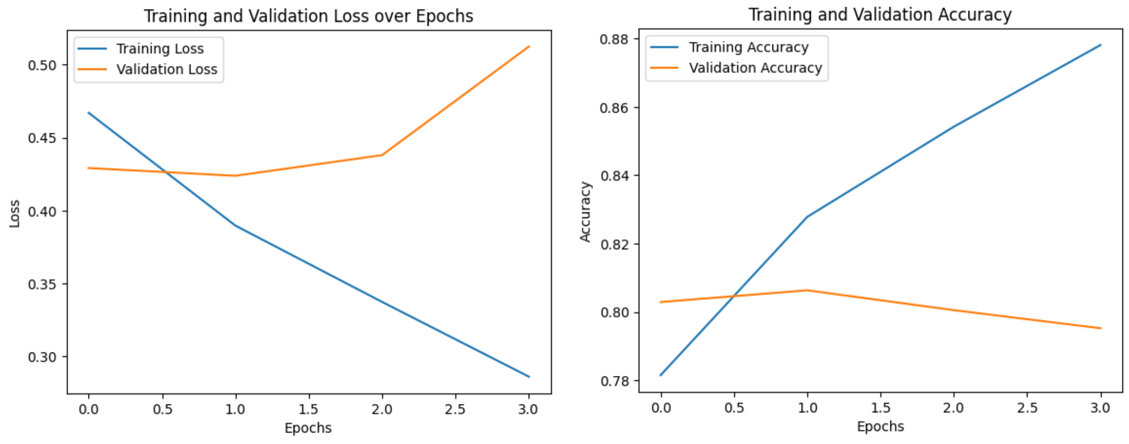


Figure 5. Training and Validation results for loss and accuracy for Bi-LSTM Model

As you can see on the figure 5, the models tend to overfitting over several epochs. That is the reason our group using **early stopping** method to maintain the best value or parameter without having the overfitting to much. Multiple runs were conducted for all three models using the base model and hyperparameter tuning using grid search to find the best performance (accuracy as the main metric) against the training and the validation data set. The models then are tested against the test dataset with the results are shown below (Figure 6)

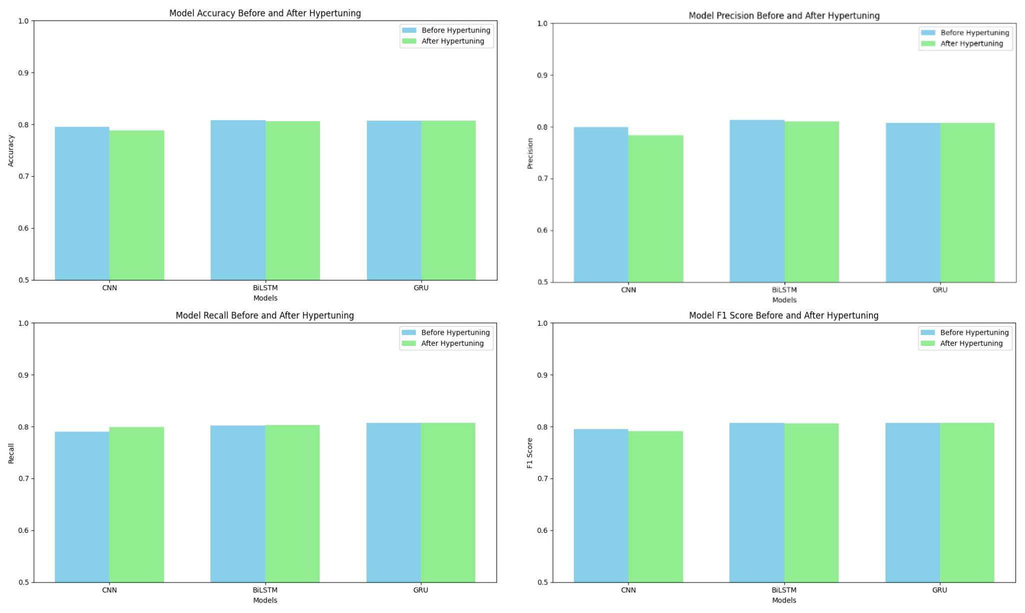


Figure 6. Composition analysis of performance matrix for each models.

Results shows that the base model of CNN, Bi-LSTM and RNN-GRU performed slightly better than the hypertuned models in terms of accuracy and precision. That is also the case when using F1 Score as the metric, the hypertuned models performed slightly worse than the base models. However, the hypertuned models performed slightly better than the base models in terms of recall score. This could be related to overfitting during the hyperparameter tuning phase. Our models has better training and validation accuracy after hypertuned but not the same case when the model use in test case. When models are hypertuned, especially in smaller datasets or with complicated architectures, they may become too specialized to the training data, leading to poor generalization performance on unseen data. As a result, metrics like accuracy and precision could drop, as the models become less capable of correctly predicting a greater range of data. Improved recall, on the other hand, indicates that the hypertuned models may have been better at recognizing more true positives, maybe due to better detection of minority classes or edge cases [36]. In classification tasks, where changing parameters like dropout, learning rate, and layer units can result in changes in model behavior, this trade-off between precision and recall is typical[17].

BiLSTM's capacity to more successfully capture long-term dependencies in sequential data explains its better performance than CNN and GRU in this work. Particularly helpful in sentiment analysis, where knowledge of the sentiment usually depends on context spread over the sentence, the bidirectional structure lets the model learn from both past and future situations. Moreover, as BiLSTM allows the model retain pertinent information for a longer duration, which can enhance both recall and generalization, its memory capacity is more suited for sequential activities like natural language processing[32]. Below is the summary of the comparison (Table 1) between base models versus hypertuned models:

Table 1. Summary of comparison performance metrics from each model

	Base CNN	Hypertuned CNN	Base Bi-LSTM	Hypertuned Bi-LSTM	Base RNN-GRU	Hypertuned RNN-GRU
Accuracy	0.7949	0.7879	0.8077	0.8061	0.8069	0.8069
Precision	0.7993	0.7833	0.8129	0.8095	0.8071	0.8071
Recall	0.7902	0.7991	0.8017	0.8030	0.8069	0.8069
F1-Score	0.7947	0.7911	0.8073	0.8063	0.8068	0.8068

The table's findings show how hypertuning affects the models' performance over several criteria. While the hypertuned model exhibited a nominal gain in recall (0.7991 vs. 0.7902), the base version of the **CNN model** somewhat outperformed the hypertuned version in both accuracy (0.7949 vs. 0.7879) and precision (0.7993 vs. 0.7833). Reflecting that although hypertuning somewhat improved recall, it did not produce overall performance increase, the F1-Score stayed somewhat close between the two (0.7947 vs. 0.7911).

For the **BiLSTM model**, the base version also outperformed the hypertuned version in terms of accuracy (0.8077 vs. 0.8061), precision (0.8120 vs. 0.8095), and F1-Score (0.8073 vs. 0.8063). But on the recall performance, hypertuned BiLSTM did perform better (0.8030 vs. 0.8017). This validates the earlier theory that hypertuning can cause overfitting to the training and validation data, hence enhancing recall by catching more true positives, but somewhat compromising accuracy and precision.

The **RNN-GRU model** demonstrated no siginficane difference between the base and hypertuned versions across all measures, as all configurations provided similar results. This can be explained by the hypertuning process converging to the same parameters as the original model, delivering no gains. Furthermore as was already mentioned, the performance drop following hypertuning for the other models could result from tuning based on training and validation sets, which might have somewhat different distributions than the test set, so overfitting and so reducing the generalizing capacity of the model.

Conclusion

In conclusion, this assignment demonstrated the effects of hypertuning on several deep learning models—CNN, BiLSTM, and RNN-GRU—applied to sentiment analysis using Twitter data. The experiment findings show that in terms of accuracy, precision and F1-Score the **base Bi-LSTM performed the best** out of all the experiments with 80.77%, 81.29% and 80.73% respectively. While the base and hypertuned RNN-GRU performed best in terms of recall with a recall score of 80.68%. Notably, BiLSTM had the best overall performance, benefiting from its capacity to capture long-term dependencies in sequential data. However, hypertuning did not offer significant improvements, particularly for RNN-GRU, where the adjusted parameters matched the default ones, resulting in similar performance.

The potential overfitting caused by hypertuning on the training and validation sets, whose distributions varied somewhat from the test set, was a significant drawback of our work. This emphasizes how important it is to have consistent data distributions over all datasets in order to do hypertuning efficiently. To enhance generalization, future research could look into more sophisticated hyperparameter tuning techniques like random search or Bayesian optimization. Further improving model performance may involve increasing the amount of the dataset and experimenting with different embedding strategies, such transformers.

The most crucial lesson we took away from this assigment was the need for careful balance when hypertuning models. Even while fine-tuning might improve model performance, it's important to weigh the trade-offs between different measures and the possibility of overfitting, particularly when working with tiny variations in the distributions of training and test data. This experience made clear how crucial it is to thoroughly assess and validate models before developing them.

References

1. Hemmatian F, Sohrabi MK. A survey on classification techniques for opinion mining and sentiment analysis. Artif Intell Rev. 2019;52(3):1495–545.
2. Medhat W, Hassan A, Korashy H. Sentiment analysis algorithms and applications: A survey. Ain Shams Eng J. 2014;5(4):1093–113.
3. Liu Q, Zheng Z, Zheng J, Chen Q, Liu G, Chen S, et al. Health communication through news media during the early stage of the COVID-19 outbreak in China: digital topic modeling approach. J Med Internet Res. 2020;22(4):e19118.
4. Sadia A, Khan F, Bashir F. An overview of lexicon-based approach for sentiment analysis. In: 2018 3rd International Electrical Engineering Conference (IEEC 2018). 2018. p. 1–6.
5. Rahat AM, Kahir A, Masum AKM. Comparison of Naive Bayes and SVM Algorithm based on sentiment analysis using review dataset. In: 2019 8th International Conference System Modeling and Advancement in Research Trends (SMART). IEEE; 2019. p. 266–70.
6. Zyout I, Zyout M. Sentiment analysis of student feedback using attention-based RNN and transformer embedding. Int J Artif Intell. 2024;13(2):2173–84.
7. Sentiment140 dataset with 1.6 million tweets [Internet]. [cited 2024 Sep 27]. Available from: https://www.kaggle.com/datasets/kazanova/sentiment140
8. Japkowicz N, Stephen S. The class imbalance problem: A systematic study. Intell Data Anal. 2002;6(5):429–49.
9. Saif H, He Y, Alani H. Semantic sentiment analysis of twitter. In: The Semantic Web–ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I 11. Springer; 2012. p. 508–24.
10. Cao J, Sheng Q, Qi P, Zhong L, Wang Y, Zhang X. False news detection on social media. ArXiv Prepr ArXiv190810818. 2019;
11. Sarker A, Islam MR, Srizon AY. A comprehensive pre-processing approach for high-performance classification of Twitter data with several machine learning algorithms. In: 2020 IEEE Region 10 Symposium (TENSYMP). IEEE; 2020. p. 630–3.
12. Uysal AK, Gunal S. The impact of preprocessing on text classification. Inf Process Manag. 2014;50(1):104–12.
13. Gimpel K, Schneider N, O'connor B, Das D, Mills DP, Eisenstein J, et al. Part-of-speech tagging for twitter: Annotation, features, and experiments. In: Proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies. 2011. p. 42–7.
14. Mikolov T. Efficient estimation of word representations in vector space. ArXiv Prepr ArXiv13013781. 2013;
15. Ma L, Zhang Y. Using Word2Vec to process big text data. In: 2015 IEEE International Conference on Big Data (Big Data). IEEE; 2015. p. 2895–7.
16. Al-Saqqa S, Awajan A. The use of word2vec model in sentiment analysis: A survey. In: Proceedings of the 2019 international conference on artificial intelligence, robotics and control. 2019. p. 39–43.
17. Zhang Y, Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. ArXiv Prepr ArXiv151003820. 2015;
18. Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences. ArXiv Prepr ArXiv14042188. 2014;
19. Gehring J, Auli M, Grangier D, Dauphin YN. A convolutional encoder model for neural machine translation. ArXiv Prepr ArXiv161102344. 2016;

20. Yin W, Kann K, Yu M, Schütze H. Comparative study of CNN and RNN for natural language processing. ArXiv Prepr ArXiv170201923. 2017;
21. Bai S, Kolter JZ, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. ArXiv Prepr ArXiv180301271. 2018;
22. Bengio Y, Goodfellow I, Courville A. Deep learning. Vol. 1. MIT press Cambridge, MA, USA; 2017.
23. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. ArXiv Prepr ArXiv14091556. 2014;
24. Cho K. On the Properties of Neural Machine Translation: Encoder-decoder Approaches. ArXiv Prepr ArXiv14091259. 2014;
25. Yin X, Liu C, Fang X. Sentiment analysis based on BiGRU information enhancement. In: Journal of Physics: Conference Series. IOP Publishing; 2021. p. 032054.
26. Shewalkar A, Nyavanandi D, Ludwig SA. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. J Artif Intell Soft Comput Res. 2019;9(4):235–45.
27. Ravanelli M, Bengio Y. Speaker recognition from raw waveform with sincnet. In: 2018 IEEE spoken language technology workshop (SLT). IEEE; 2018. p. 1021–8.
28. Wang Y, Huang M, Zhu X, Zhao L. Attention-based LSTM for aspect-level sentiment classification. In: Proceedings of the 2016 conference on empirical methods in natural language processing. 2016. p. 606–15.
29. Tang D, Qin B, Liu T. Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of the 2015 conference on empirical methods in natural language processing. 2015. p. 1422–32.
30. Cho K. On the Properties of Neural Machine Translation: Encoder-decoder Approaches. ArXiv Prepr ArXiv14091259. 2014;
31. Hochreiter S. Long Short-term Memory. Neural Comput MIT-Press. 1997;
32. Schuster M, Paliwal KK. Bidirectional recurrent neural networks. IEEE Trans Signal Process. 1997;45(11):2673–81.
33. Liu P, Qiu X, Huang X. Recurrent neural network for text classification with multi-task learning. ArXiv Prepr ArXiv160505101. 2016;
34. Shi T, Keneshloo Y, Ramakrishnan N, Reddy CK. Neural abstractive text summarization with sequence-to-sequence models. ACM Trans Data Sci. 2021;2(1):1–37.
35. Kamyab M, Liu G, Adjeisah M. Attention-based CNN and Bi-LSTM model based on TF-IDF and glove word embedding for sentiment analysis. Appl Sci. 2021;11(23):11255.
36. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929–58.

Appendix

A. Software Configuration (How to run the code and the results of Hypertuning)

- To run this code, there are some configuration that needs to be installed before we can run the code.

- The first thing, as usual for the device that can run our code, is to make sure that your device already has Python 3.7 or higher. The Python can be directly downloaded from the website, or you can use a third-party app like Anaconda to install the python. Then on your terminal, you should check your Python version by running the code “python –version”

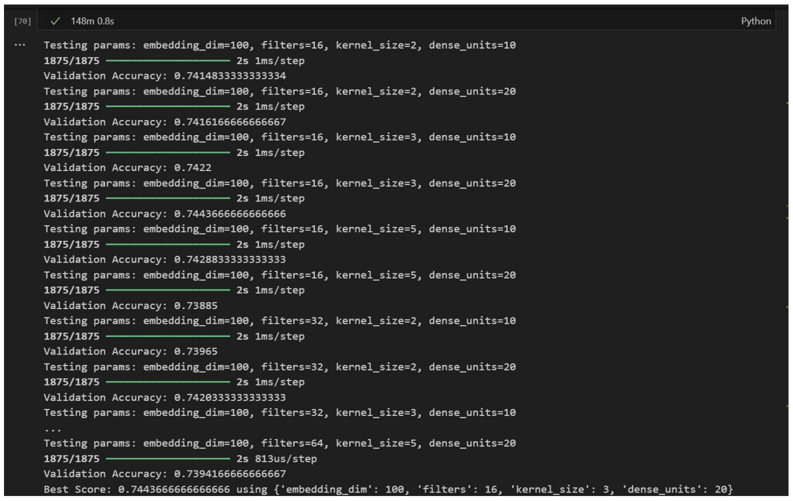
- Next, you can create a new virtual environment (venv) but it is optional. Same with the instructions on the assignment 1, we can create the virtual environment using code “python -m venv myenv” Then to activate if you are on windows “myenv\Scripts\activate”. If you are on mac “source myenv/bin/activate” You need to ensure that you have all the libraries that you need for our Jupyter Notebook. We will use mostly libraries from tensorflow because we are building deep learning models. The details of the libraries are shown below:
  1. **numpy**==1.24.3 (used for numerical operations)
  2. **pandas**==2.0.3 (used for data manipulation)
  3. **matplotlib**==3.7.2 (used for plotting the data)
  4. **seaborn**==0.12.2 (used for advance visualisation)
  5. **wordcloud ==**1.9.3(used to plot wordcloud)
  6. **scikit-learn**==1.3.0 (used for data preprocessing and machine learning models)
  7. **tensorflow ==** 2.12 or higher (used for deep learning models)
  8. **gensim ==** 4.1.3 (used for word2Vec embedding)

- The installation also can be done using third party app or using terminal using the command “pip install pandas matplotlib wordcloud seaborn numpy scikit-learn tensorflow gensim”.

- Then you can check if your installation is correcltly done by running this command “python -c "import pandas, matplotlib, worcloud, seaborn, numpy, sklearn, tensorflow, gensim; print('All libraries installed correctly')". If there are no errors occur, then your packages is ready.

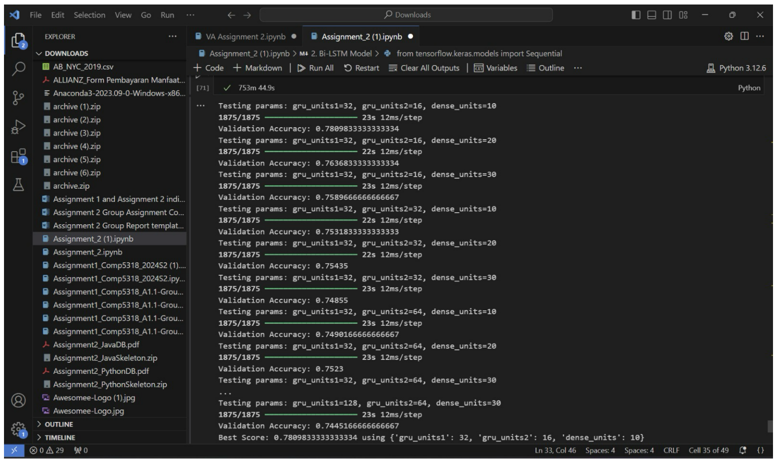
- For you can run this code, make sure that the Jupyter Notebook is in the same folder as the dataset. The dataset name that we used is “training.300000.processed.noemoticon.csv”

- **IMPORTANT NOTE:** In our Jupyter notebook, the hypertuning code for CNN, BiLSTM, and GRU is commented out, which means it won’t run when you try to execute the code unless you uncomment the lines. This is to prevent the notebook from running for too long because the hyperparameter tuning that we did for each model can take **up to 6 hours**. For the details, the **CNN hypertuned** is done in **148 minutes, GRU 750 minutes,** and **BiLSTM 242 minutes.** We will provide the results of the hypertuning for each model in the figure below .



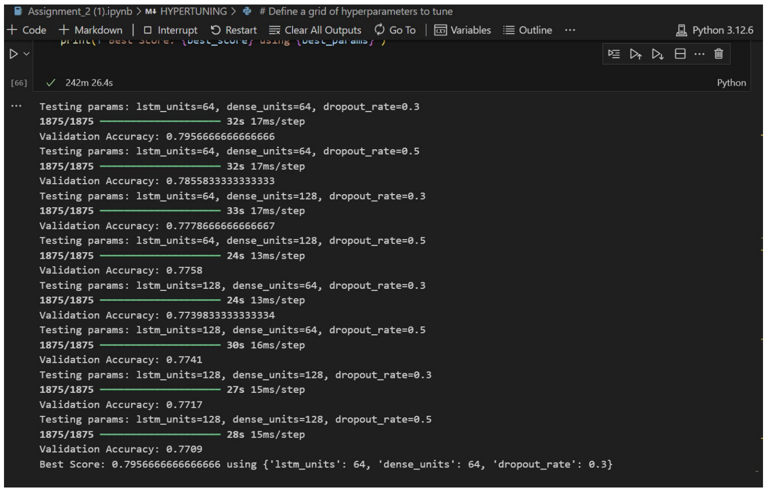
```
Testing params: embedding_dim=100, filters=16, kernel_size=2, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7414833333333334
Testing params: embedding_dim=100, filters=16, kernel_size=2, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7416166666666667
Testing params: embedding_dim=100, filters=16, kernel_size=3, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7422
Testing params: embedding_dim=100, filters=16, kernel_size=3, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7443666666666666
Testing params: embedding_dim=100, filters=16, kernel_size=5, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7428833333333333
Testing params: embedding_dim=100, filters=16, kernel_size=5, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.73885
Testing params: embedding_dim=100, filters=32, kernel_size=2, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.73965
Testing params: embedding_dim=100, filters=32, kernel_size=2, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Validation Accuracy: 0.7420333333333333
Testing params: embedding_dim=100, filters=32, kernel_size=3, dense_units=10
...
Testing params: embedding_dim=100, filters=64, kernel_size=5, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 813us/step
Validation Accuracy: 0.7394166666666667
Best Score: 0.7443666666666666 using {'embedding_dim': 100, 'filters': 16, 'kernel_size': 3, 'dense_units': 20}
```

Figure 7. CNN Hypertuned Results



```
Testing params: gru_units1=32, gru_units2=16, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.7809833333333334
Testing params: gru_units1=32, gru_units2=16, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step
Validation Accuracy: 0.7636833333333334
Testing params: gru_units1=32, gru_units2=16, dense_units=30
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.7589666666666667
Testing params: gru_units1=32, gru_units2=32, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step
Validation Accuracy: 0.7531833333333333
Testing params: gru_units1=32, gru_units2=32, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step
Validation Accuracy: 0.75435
Testing params: gru_units1=32, gru_units2=32, dense_units=30
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.74855
Testing params: gru_units1=32, gru_units2=64, dense_units=10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.7490166666666667
Testing params: gru_units1=32, gru_units2=64, dense_units=20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.7523
Testing params: gru_units1=32, gru_units2=64, dense_units=30
...
Testing params: gru_units1=128, gru_units2=64, dense_units=30
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step
Validation Accuracy: 0.7445166666666667
Best Score: 0.7809833333333334 using {'gru_units1': 32, 'gru_units2': 16, 'dense_units': 10}
```

Figure 8. GRU Hypertuned Results



```
Testing params: lstm_units=64, dense_units=64, dropout_rate=0.3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step
Validation Accuracy: 0.7956666666666666
Testing params: lstm_units=64, dense_units=64, dropout_rate=0.5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step
Validation Accuracy: 0.7855833333333333
Testing params: lstm_units=64, dense_units=128, dropout_rate=0.3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 17ms/step
Validation Accuracy: 0.7778666666666667
Testing params: lstm_units=64, dense_units=128, dropout_rate=0.5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 24s 13ms/step
Validation Accuracy: 0.7758
Testing params: lstm_units=128, dense_units=64, dropout_rate=0.3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 24s 13ms/step
Validation Accuracy: 0.7739833333333334
Testing params: lstm_units=128, dense_units=64, dropout_rate=0.5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step
Validation Accuracy: 0.7741
Testing params: lstm_units=128, dense_units=128, dropout_rate=0.3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 27s 15ms/step
Validation Accuracy: 0.7717
Testing params: lstm_units=128, dense_units=128, dropout_rate=0.5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 28s 15ms/step
Validation Accuracy: 0.7709
Best Score: 0.7956666666666666 using {'lstm_units': 64, 'dense_units': 64, 'dropout_rate': 0.3}
```

Figure 9. Bi-LSTM Hypertuned Results

B. Hardware Configuration

The hardware configuration that we used is still the same with **the Assignment 1** for creating the model and hypertuning, which are:

- **1st Device: Lenovo Legion 14i**

Operating System	: Windows 11
Processor	: Ryzen 7 4800
GPU	: RTX 4060
RAM	: 32 GB
STORAGE	: 1 TB

- **2nd Device: Macbook Air M2**

Operating System	: Sonoma 14.5
Processor	: Apple M2
GPU	: M2 8-Core GPU
RAM	: 8 GB
STORAGE	: 256 GB