TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE

BEYOND EDUCATION

Project Title

Assignment Documentation

BMCS2123
NATURAL LANGUAGE PROCESSING

202105 Session, Year 2021/22

Team Members : (a) Lim Chia Chung *(20WMR08877) Long Short-Term Memory Network (LSTM)*
(b) Lim Ming Jun *(20WMR08879) DistilBERT*
(c) Leong Yit Wee *(20WMR08876) Convolutional Neural Network (CNN)*

Programme : RDS3

Tutorial Class : Group 3

Lecturer : Dr. Tan Chi Wee

Table of Contents

Table of Contents 1

Introduction 2
- 1.1 Problem Background 2
- 1.2 Objectives/Aims 2
- 1.3 Motivation 2
- 1.4 Timeline/Milestone 3

Research Background 4
- 2.1 Background of the applications 4
- 2.2 Analysis of selected tool with any other relevant tools 4
- 2.3 Justify why the selected tool is suitable 5
  - 2.3.1 Long Short-Term Memory (LSTM) - Lim Chia Chung 5
  - 2.3.2 Distil Bidirectional Encoder Representations from Transformers (DistilBERT) - Lim Ming Jun 6
  - 2.3.3 Convolutional Neural Network (CNN) - Leong Yit Wee 7

Methodology 7
- 3.1 Description of dataset 7
- 3.2 Applications of the algorithm(s) 8
  - 3.2.1 Long Short-Term Memory (LSTM) - Lim Chia Chung 8
  - 3.2.2 Distil Bidirectional Encoder Representations from Transformers (DistilBERT) - Lim Ming Jun 9
  - 3.2.3 Convolutional Neural Network (CNN) - Leong Yit Wee 12
- 3.3 System flowchart 15
- 3.4 Proposed test plan/hypothesis 15
  - 3.4.1 Model Training Test Plan 15
  - 3.4.2 Single Movie Review Analysis Test Plan 16
  - 3.4.3 Multi Movie Reviews Analysis Test Plan 17

Result 18
- 4.1 Results 18
  - 4.1.1 Model Training 18
  - 4.1.2 Single Movie Review Analysis 22
  - 4.1.3 Multiple Movie Reviews Analysis 24
- 4.2 Discussion/Interpretation 25

Discussion and Conclusion 26
- 5.1 Achievements 26
- 5.2 Limitations and Future Works 26

References & Sources 28

Introduction

1.1 Problem Background

Nowadays, watching movies has become a frequent leisure activity for most people where most people will have the habit of viewing or listening to the movie reviews, comments or feedback from other people in the social media platforms (E.g. Facebook, Twitter, Youtube, Instagram) or comment sections from online movie databases (E.g. IMDb, TMDB, Netflix) before watching the movie. Because people want to know the quality of the movie and whether it is worthwhile or not. By doing this, people will then decide to watch or not to watch the movie they desire to watch. If they find that the movie they desire to watch has many negative comments or reviews, they may decide not to watch it. Conversely, they are more likely to watch the movie if many positive comments or reviews are found. However, some people may not understand some comments and reviews posted by other users whereas some of them will also be too lazy to read those long comments and reviews. Therefore, they cannot identify whether those comments and reviews are positive or negative, which may affect their decision to watch the movie.

1.2 Objectives/Aims

Movie review plays an important role in our life as it affects the audience's decision whether the movie is worth their time to watch. Hence, a robust sentiment analyzer will be helpful to help the audience to identify the sentiment of each movie review instead of waiting time from reading all the reviews. This assignment aims to build a sentiment analyzer application that can **receive input from users to help them to identify the sentiment of the movie review** that they don't understand as well as **carry out sentiment prediction on multiple reviews**. In this assignment, we will use IMDB movie reviews dataset to train 3 different models to build our sentiment analyzer application. Then, we will use our trained models to do sentiment prediction of our crawled movie reviews (well-labeled with positive & negative sentiment) from other websites to compare which models perform the best. We will compare these 3 models in terms of accuracy, precision, recall and F1-score to identify which gives better and more reasonable results.

1.3 Motivation

Sentiment analysis is a branch of text classification that analyzes subjective words or phrases associated with the positive, negative and neutral emotions. In addition, the words and phrases spread rapidly to social media as well as negative comments gain as quickly as possible. If the organization does not quickly and respectfully deal with dissatisfied customers, they may share their disappointment to their friends or even the public such as post on social media like Facebook or Twitter. Moreover, the organization can utilize sentiment analysis to analyze what the customer likes and dislikes about their services, brands and products which is critical to their business. Apart from tracking their own online records, the organization may also track their competitors' comments by using sentiment analysis to see how the business can be stacked up. Positive sentiment can assist in determining the key factor of the success of competitors. While negative sentiments can lead to opportunities for the organization.

1.4 Timeline/Milestone

Week	Progress
1	Browsing dataset and finalize the topic
2	Background studies
3	Selection of algorithms Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and Bidirectional Encoder Representations from Transformers (BERT)
4	Research for the suitable development tools.
5	Data Understanding
6	Data Preprocessing
7	Web Crawler Development
8 & 9	Model Building
10	Model Fine Tuning
11	Model Evaluation
12	Local Sentiment Analysis App
13	Documentation & Presentation

Research Background

2.1 Background of the applications

Sentiment analysis is the computerized preparation of analyzing content to determine the opinion communicated (positive, negative or neutral). A few prevalent sentiment analysis applications incorporate customer support management, social media monitoring and analysis of client feedback. In the foundation of assumption examination, advanced AI calculations apply dialect deconstruction procedures such as **tokenization**, **part-of-speech tagging**, **stemming** and **lemmatization** to decompose and understand content. As it were at that point can machine learning computer programs classify unstructured content by feeling and opinion. Automatic content examination can be performed on any content source, to sort study reactions and live chats, Twitter and Facebook posts, or to check emails and archives. All usually profitable data for companies, filled with bits of knowledge that can offer assistance to make data-driven choices.

2.2 Analysis of selected tool with any other relevant tools

Tools comparison	**Remark**	**Long Short-Term Memory (LSTM)**	**Distil Bidirectional Encoder Representations from Transformers (BERT)**	**Convolution Neural Networks (CNN)**
Type of license and open source license	State all types of license	Apache License 2.0	Apache License 2.0	Apache License 2.0
Year founded	When is this tool being introduced?	1977	2019	1980
Founding company	Owner	Juergen Schmidhuber	HuggingFace	Yann LeCun
License Pricing	Compare the prices if the license is used for development and business/commercialization	Free	Free	Free
Supported features	What features does it offer?	Capable of keeping or forgetting information and reducing the impact of short-term memory by carrying relevant information within the processing.	Text sequencing, text tokenizing, text padding, text encoding.	Text classification, image data processing and naturally and adaptively learn spatial pecking orders of highlights through a backpropagation calculation
Common applications	In what areas this tool is usually used?	Handwriting Generation, Machine Translation and Video to Text	Text summarization, sentiment classification, sentence prediction, conversational response generation	Image Tagging, Visual Search, Recommender Engine
Customer support	How the customer support is given, e.g. proprietary, online community, etc.	Proprietary, Online community	Proprietary, Online community	Proprietary, Online community
Limitations	The drawbacks of the software	Prone to overfitting and hard to apply Dropout feature to mitigate this question. LSTM	Requires a strong GPU and CPU to run the program if you apply any	A Convolutional neural network is altogether slower due to an operation such as

		has similarities to feed-forward neural networks as it will get affected by different random weight initialization.	type of BERT Model like DistilBERT, ALBERT, FlauBERT etc. This is because these models are highly pre-trained with millions of data. Users need to use a very long time to fine-tune them. Normal CPU is not enough.	maxpool. If the CNN has a few layers at that point the preparation takes a parcel of time on the off chance that the computer doesn't comprise of a great GPU. A ConvNet requires an extensive Dataset to handle and prepare the neural organization.

2.3 Justify why the selected tool is suitable

2.3.1 Long Short-Term Memory (LSTM) - Lim Chia Chung

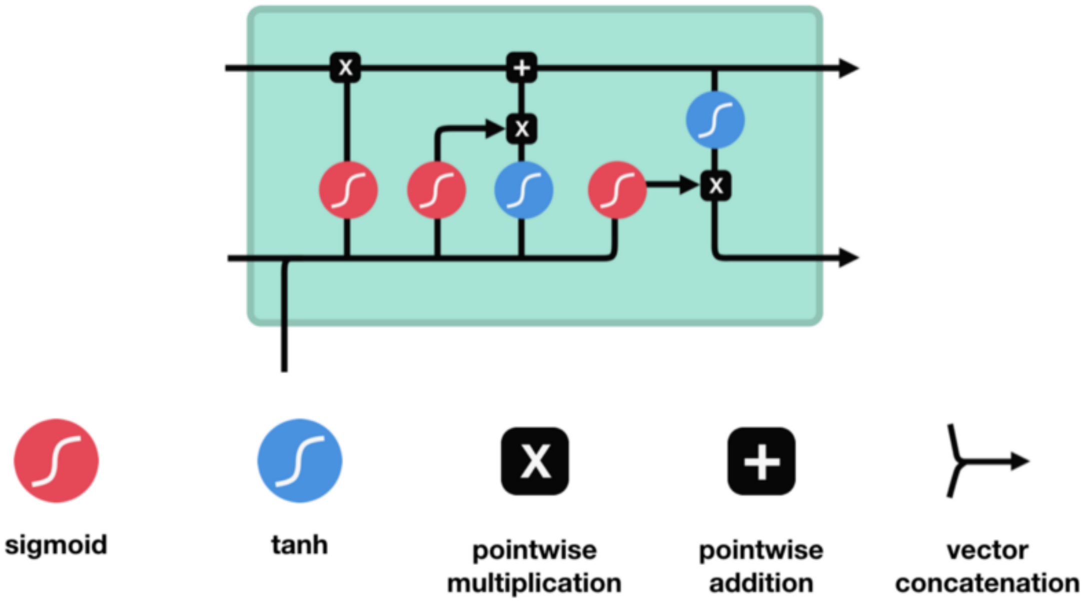

Figure 2.1 Architecture Diagram of Long Short-Term Memory

Long Short-Term Memory (LSTM) is considered a Recurrent Neural Network of deep learning. The other algorithms have been attempted multiple times before and now I would like to try something new and challenging. Nowadays, deep learning algorithms have been quite effective in performing sentiment analysis, which outperforms the conventional feature-based algorithm in terms of automatic feature selection, rich representations and overall performance. LSTM was born to resolve the drawbacks of **Vanishing Gradient** and **Exploding Gradient** of RNN where LSTM supports to either keep or forget the information during **Back Propagation**. A simple understanding of RNN is that it tends to the long sequences, indicating that it has a short-term memory. LSTM uses long-term memory to classify long sequence data, therefore it is appropriate to long-term dependencies very effectively and efficiently, especially for movie reviews because sentences of movie reviews are long enough and we have a 25,000 large enough dataset to train this model.

2.3.2 Distil Bidirectional Encoder Representations from Transformers (DistilBERT) - Lim Ming Jun

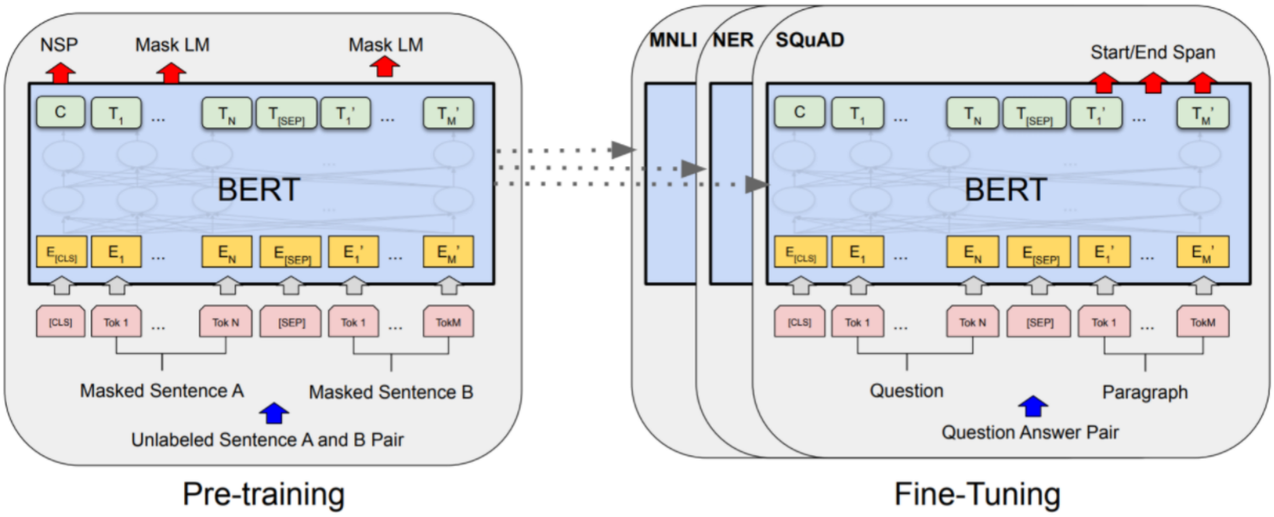

Figure 2.2 Architecture Diagram of DistilBERT

BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model **can be finetuned with just one extra output layer to build state-of-the-art models** for many various tasks, such as question answering and language inference, without requiring significant task-specific architecture modifications. Besides, one of the most significant issues in today's NLP industry is a **lack of sufficient training data**. Overall there is a massive amount of text data available, but if we want to create task-specific datasets, we need to divide that pile into very many different fields. As a result, we **end up with only a few thousand** or **a few hundred thousand human-labelled training examples**. So, researchers have used those massive piles of unannotated text on the web (this is known as pre-training) to build different techniques for training general-purpose language representation models to tackle the problem of lack of sufficient training data. These general-purpose pre-trained models can then be fine-tuned on smaller task-specific datasets, e.g., when working with problems like question answering and sentiment analysis. On the other hand, the model that we are using is the DistilBert model where it is a small, fast, cheap and light Transformer model trained by distilling Bert base. It has **40% fewer parameters** than the bert-base-uncased, runs **60% faster** while **preserving over 95% of BERT's performances**, which is as accurate as of the original BERT Model.

2.3.3 Convolutional Neural Network (CNN) - Leong Yit Wee

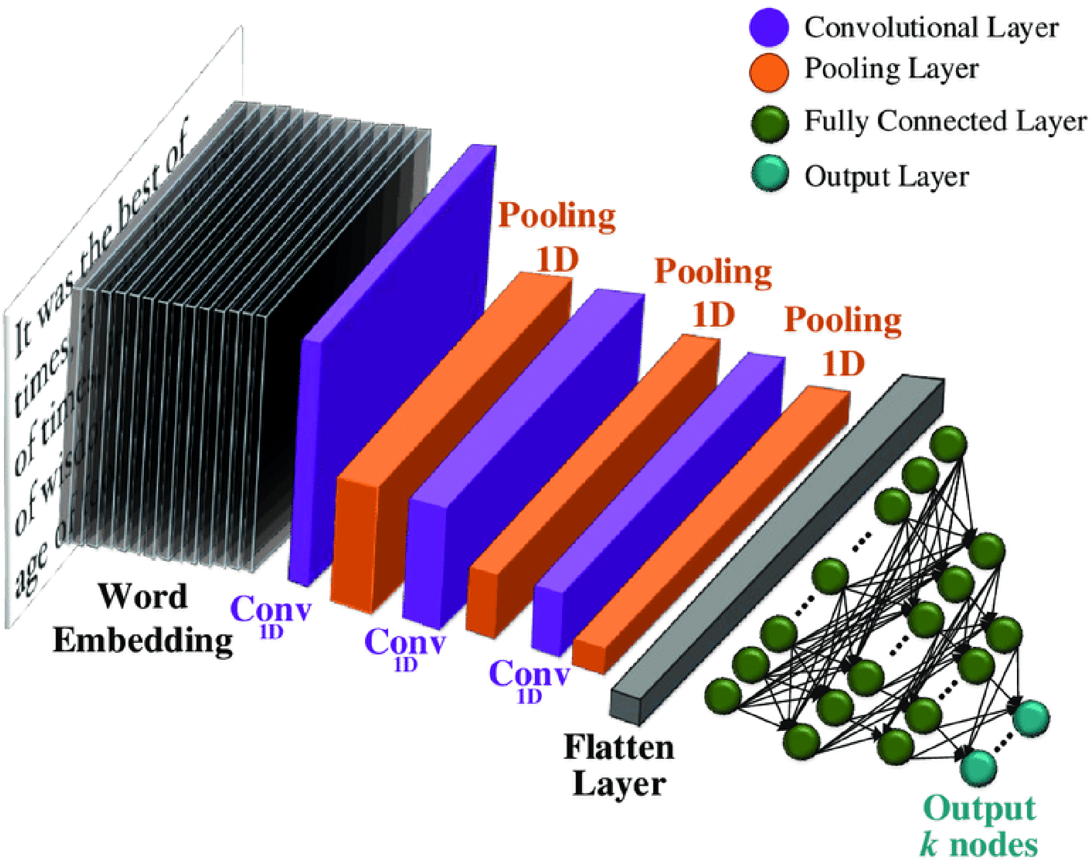

Figure 2.3 Architecture Diagram of Convolutional Neural Network

CNN (Convolutional Neural Network) may be a lesson of profound, feed-forward artificial neural networks (where associations between hubs don't frame a cycle) & utilize a variety of multilayer perceptrons planned to require minimal preprocessing. The convolutional layer is the centre building piece of a CNN. The layer's parameters consist of a set of learnable filters, which have a little responsive field but amplify through the total profundity of the input volume. The pooling layer utilized in between the convolutional layers decreases the dimensional complexity and still keeps the noteworthy data of the convolutions. In a CNN, the final layers are fully connected layers meaning each hub of one layer is connected to each hub of the other layer. The best qualities of CNNs are to **supply a proficient thick arrangement that performs the expectation productively**. Our data also matched the requirements of CNN because CNN needs large data for trains and we have 25,000.

Methodology

3.1 Description of dataset

<u>IMDB</u>

This dataset contains 50,000 rows of movie reviews with well-labelled positive and negative sentiment.

scrap_movie_reviews
This dataset contains 868 rows of movie reviews which we wrote a Python script to scrape it. There are 2 columns for this dataset which is Review and Sentiment. Then, we use this scraped dataset to test our sentiment analysis application.

3.2 Applications of the algorithm(s)

3.2.1 Long Short-Term Memory (LSTM) - Lim Chia Chung

First of all, the **embedding layer** is defined for transforming the corpus into word vectors before feeding into the Neural Networks. Secondly, Stacked Bidirectional LSTM is utilized in the model training. In order to be faster and more comprehensive at learning issues, it trains two LSTMs on the input sequence. The first and second **Bidirectional layers** are defined by the **64-units** dimensions of the output space as well as the rate **Dropout layers** are defined by **0.2**. In addition, the first hidden layer is defined by **64-units** dimensions of the output space with **Rectified Linear Unit** (ReLU) activation function as well as the rate of **Dropout layers** is set to **0.1** to avoid overfitting. Considering this is a binary classification of sentiment analysis and hence the Sigmoid activation function is appropriate to be set in the final hidden layer and with the **1 unit** dimension of output space.

On the other hand, **Adam** optimizer is utilized as it will compute an appropriate learning rate for Back Propagation whereas the **learning rate** is set to **0.001**. As for the loss function, Binary Cross-entropy is used to compute the loss between **y_true** and **y_pred**. Since the sentiment analysis is only focused on accuracy, thus the accuracy metric is chosen for model evaluation.

Besides, **5 epochs** and **64 batch size** are selected during the model fitting. Several fine-tunings have been performed, with the conclusion that the higher the number of epochs, the higher the likelihood of overfitting.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1430, 64)          13067328
_________________________________________________________________
bidirectional (Bidirectional (None, 1430, 128)         66048
_________________________________________________________________
dropout (Dropout)            (None, 1430, 128)         0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               98816
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 13,240,513
Trainable params: 13,240,513
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.1 Model Summary

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/keras/backend.py:4994: UserWarning:

"`binary_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus

352/352 [==============================] - 248s 672ms/step - loss: 0.4576 - accuracy: 0.8074 - val_loss: 0.3133 - val_accuracy: 0.8748
Epoch 2/5
352/352 [==============================] - 235s 667ms/step - loss: 0.1821 - accuracy: 0.9432 - val_loss: 0.4607 - val_accuracy: 0.8152
Epoch 3/5
352/352 [==============================] - 233s 662ms/step - loss: 0.0806 - accuracy: 0.9793 - val_loss: 0.8608 - val_accuracy: 0.7776
Epoch 4/5
352/352 [==============================] - 232s 660ms/step - loss: 0.0592 - accuracy: 0.9866 - val_loss: 0.8676 - val_accuracy: 0.7764
Epoch 5/5
352/352 [==============================] - 233s 662ms/step - loss: 0.0479 - accuracy: 0.9883 - val_loss: 0.7445 - val_accuracy: 0.7988
```

Figure 3.2 Model Training

3.2.2 Distil Bidirectional Encoder Representations from Transformers (DistilBERT) - Lim Ming Jun

During the DistilBERT modelling phase, I have used a Python package called **ktrain** to help me to fine-tune this model for my sentiment classification task. Ktrain is a **lightweight wrapper for the deep learning library TensorFlow Keras** (and some other libraries) to **help build, train, and deploy neural networks** and **other machine learning models**. Inspired by ML framework extensions like fastai and ludwig, it is designed to make deep learning and AI more accessible and easier to apply for both newcomers and experienced practitioners, especially for newcomers or beginners that are not familiar with TensorFlow and Deep Learning.

On top of that, ktrain **provides support for applying many pre-trained deep learning architectures** in the domain of Natural Language Processing and **BERT** or **DistilBert is one of them**. So, in this assignment, I will be using the implementation of pre-trained DistilBERT provided by ktrain and fine-tune it to classify whether the reviews are positive or negative.

We first need to feed the ktrain with training dataset, testing dataset, text column

(column that contains users reviews), label column (column that we want to predict its result), max length (maximum words for each document) and preprocess_mode (chosen pre-trained deep learning architecture, we select **'DistilBert'** in this assignments) to help us to conduct preprocessing of the text data we provide. Text **tokenizing** and **encoding** are the two main preprocessing techniques that will be gone through by BERT models, in this case, ktrain will help us to do all these things, we do not need to do it manually by ourselves.

```
(train, val, preproc) = text.texts_from_df(train_df=df_train, text_column='cleaned_bert_review', label_columns='sentiment',
                        val_df = df_test,
                        maxlen = 512,
                        preprocess_mode = 'distilbert')
```

Figure 3.3

After that, it's time for us to create a model. We just need to feed the ktrain with **train**, **val** and **preproc** returned by the above figure to create the classifier model.

```
model = text.text_classifier(name = 'distilbert', train_data = train, preproc = preproc)
```

```
Is Multi-Label? False
maxlen is 512
done.
```

Figure 3.4

Then, we will feed the ktrain with the **model** that we have just created as well as the **train**, **val** that we got from *Figure 3.3* and **batch size** equal to 6 to create the learner. This learner can help us to find the best or most optimum learning rate to fine-tune our classifier.

```
learner = ktrain.get_learner(model = model, train_data = train, val_data = val, batch_size = 6)
learner2 = ktrain.get_learner(model = model, train_data = train, val_data = val, batch_size = 6)
learner3 = ktrain.get_learner(model = model, train_data = train, val_data = val, batch_size = 6)
```

Figure 3.5

We then can use the **learner** that we have just created to invoke the **lr_find** function by specifying maximum epochs that you want to find out the best or most optimum learning rate. Next, we must invoke the **lr_plot** function to visually inspect the loss plot to help identify the maximal learning rate associated with falling loss.

```
# Find out best learning rate BUT it may take days or many days to find out although your GPU or CPU is strong enough.
learner.lr_find(max_epochs = 49)
```

```
simulating training for different learning rates... this may take a few moments...
Epoch 1/49
4166/4166 [==============================] - 887s 210ms/step - loss: 0.6603 - accuracy: 0.6824
Epoch 2/49
4166/4166 [==============================] - 878s 211ms/step - loss: 0.4073 - accuracy: 0.8652
Epoch 3/49
```

Figure 3.6

```
learner.lr_plot(suggest=True, return_fig=True)
```

```
Two possible suggestions for LR from plot:
        Min numerical gradient: 6.53E-07
        Min loss divided by 10: 1.95E-06
```

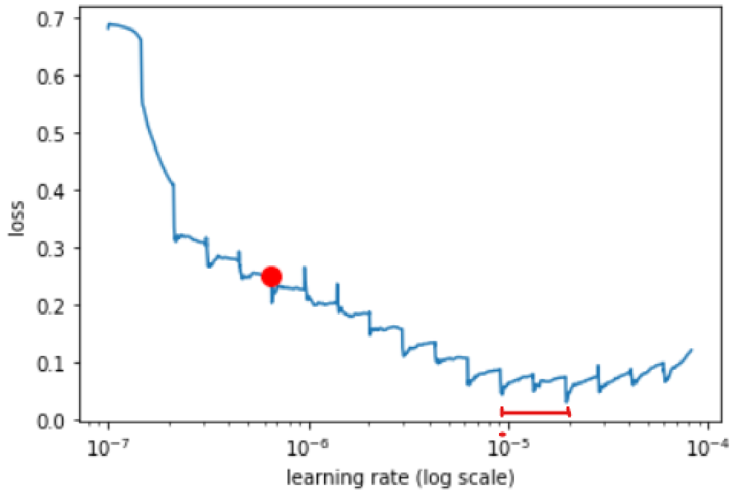

Figure 3.7

Later, we can see from the graph that at around 1e-5 (underline with red) has the minimum loss. So now I will use the learning rates **2e-5**, **1.28e-5** and **1.95e-5** (which lie under the red line) to train or fine-tune our 3 classifiers. In this case, we have 3 classifiers which are **learner**, **learner2** and **learner3** that we have created in *Figure 3.5*.

```
[ ] learner.fit_onecycle(lr=2e-5, epochs=2)
```

```
begin training using onecycle policy with max lr of 2e-05...
Epoch 1/2
4167/4167 [==============================] - 1114s 265ms/step - loss: 0.2997 - accuracy: 0.8754 - val_loss: 0.2259 - val_accuracy: 0.9104
Epoch 2/2
4167/4167 [==============================] - 1107s 265ms/step - loss: 0.1608 - accuracy: 0.9408 - val_loss: 0.1852 - val_accuracy: 0.9316
<tensorflow.python.keras.callbacks.History at 0x28f44a93b20>
```

```
learner2.autofit(lr = 1.28e-05)
```

```
early_stopping automatically enabled at patience=5
reduce_on_plateau automatically enabled at patience=2


begin training using triangular learning rate policy with max lr of 1.28e-05...
Epoch 1/1024
4167/4167 [==============================] - 1110s 266ms/step - loss: 0.0924 - accuracy: 0.9701 - val_loss: 0.2162 - val_accuracy: 0.9270
Epoch 2/1024
4167/4167 [==============================] - 1108s 265ms/step - loss: 0.0545 - accuracy: 0.9831 - val_loss: 0.2573 - val_accuracy: 0.9256
Epoch 3/1024
4167/4167 [==============================] - 1111s 266ms/step - loss: 0.0379 - accuracy: 0.9886 - val_loss: 0.2773 - val_accuracy: 0.9255

Epoch 00003: Reducing Max LR on Plateau: new max lr will be 6.4e-06 (if not early_stopping).
Epoch 4/1024
4167/4167 [==============================] - 1110s 266ms/step - loss: 0.0195 - accuracy: 0.9952 - val_loss: 0.3280 - val_accuracy: 0.9240
Epoch 5/1024
```

```
learner3.autofit(lr = 1.95e-5)
```

```
early_stopping automatically enabled at patience=5
reduce_on_plateau automatically enabled at patience=2


begin training using triangular learning rate policy with max lr of 1.95e-05...
Epoch 1/1024
4167/4167 [==============================] - 1107s 265ms/step - loss: 0.2883 - accuracy: 0.8772 - val_loss: 0.1906 - val_accuracy: 0.9246
Epoch 2/1024
4167/4167 [==============================] - 1104s 264ms/step - loss: 0.1650 - accuracy: 0.9380 - val_loss: 0.1914 - val_accuracy: 0.9292
Epoch 3/1024
4167/4167 [==============================] - 1105s 264ms/step - loss: 0.1017 - accuracy: 0.9655 - val_loss: 0.2164 - val_accuracy: 0.9289

Epoch 00003: Reducing Max LR on Plateau: new max lr will be 9.75e-06 (if not early_stopping).
Epoch 4/1024
```

Figure 3.8

After going through all the above steps, we can just use all the learners that we have created and trained above to invoke the **validate** function to compare their results. Choose the one that performs the best. I personally will choose the one that produces the highest accuracy and best confusion matrix result (Highest number of True Negative and True Positive). In this case, I will choose **learner**.

```
# First learner
learner.validate(class_names = preproc.get_classes())

              precision    recall  f1-score   support

         neg       0.93      0.94      0.93     12500
         pos       0.94      0.93      0.93     12500

    accuracy                           0.93     25000
   macro avg       0.93      0.93      0.93     25000
weighted avg       0.93      0.93      0.93     25000

array([[11718,   782],
       [  927, 11573]], dtype=int64)
```

```
[ ] # Second learner
learner2.validate(class_names = preproc.get_classes())

              precision    recall  f1-score   support

         neg       0.93      0.92      0.92     12500
         pos       0.92      0.93      0.92     12500

    accuracy                           0.92     25000
   macro avg       0.92      0.92      0.92     25000
weighted avg       0.92      0.92      0.92     25000

array([[11521,   979],
       [  905, 11595]], dtype=int64)
```

```
[ ] # Third learner
```

Figure 3.9

That's come to the end of our fine-tuning or training the classifier. Now we just need to get the **predictor** as shown in the figure below by calling the **get_predictor** method with the **chosen learner** in *Figure 3.8* passes in as the argument. Then, use this predictor to analyze whether a text sentence is positive or negative (Refer to *Figure 3.11*).

Choose the best learner from **Section 3.4**

```
predictor = ktrain.get_predictor(learner.model, preproc)
```

Figure 3.10

```
data = ['This movie was really bad. Acting was also bad', 'I will not watch again',
        'The story of this movie quite interesting. I will see it again', 'Another great movie. Everyone must watch']
```

```
[ ] predictor.predict(data)

['neg', 'neg', 'pos', 'pos']
```

Figure 3.11

3.2.3 Convolutional Neural Network (CNN) - Leong Yit Wee

A preservationist CNN arrangement is utilized with **32 filters** (parallel areas for handling words) and a **kernel estimate of 8** with a corrected straight ('**relu**') enactment work taken after by a **pooling layer** that decreases the yield of the convolutional layer by half. **Max Pooling** may be a pooling operation that calculates the most extreme esteem for patches of a highlight outline and employs it to make a pooled include outline. Another, the 2D output from the CNN portion of the demonstration is **flattened** to one long 2D vector to represent the 'features' extricated by the CNN. The back-end of the demonstration could be a standard Multilayer Perceptron layer to decipher the CNN highlights. The yield layer uses a **Sigmoid** enactment work to output esteem between 0 and 1 for the negative and positive sentiment within the review. At last grid search was also performed by using kerasclassifier and the result is the same with previous 85% accuracy.

```
# define model
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=max_sequence_len))
model.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1430, 100)         20417700
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1423, 32)          25632
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 711, 32)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 22752)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                227530
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 11
=================================================================
Total params: 20,670,873
Trainable params: 20,670,873
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3.12

```
# create model
model = KerasClassifier(build_fn=create_model, verbose=0)

# define the grid search parameters
batch_size = [32,64]
epochs = [1,5]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.769442 using {'batch_size': 32, 'epochs': 5}
0.726764 (0.080990) with: {'batch_size': 32, 'epochs': 1}
0.769442 (0.041215) with: {'batch_size': 32, 'epochs': 5}
0.602531 (0.213556) with: {'batch_size': 64, 'epochs': 1}
0.623559 (0.089774) with: {'batch_size': 64, 'epochs': 5}
```

Figure 3.13

3.3 System flowchart

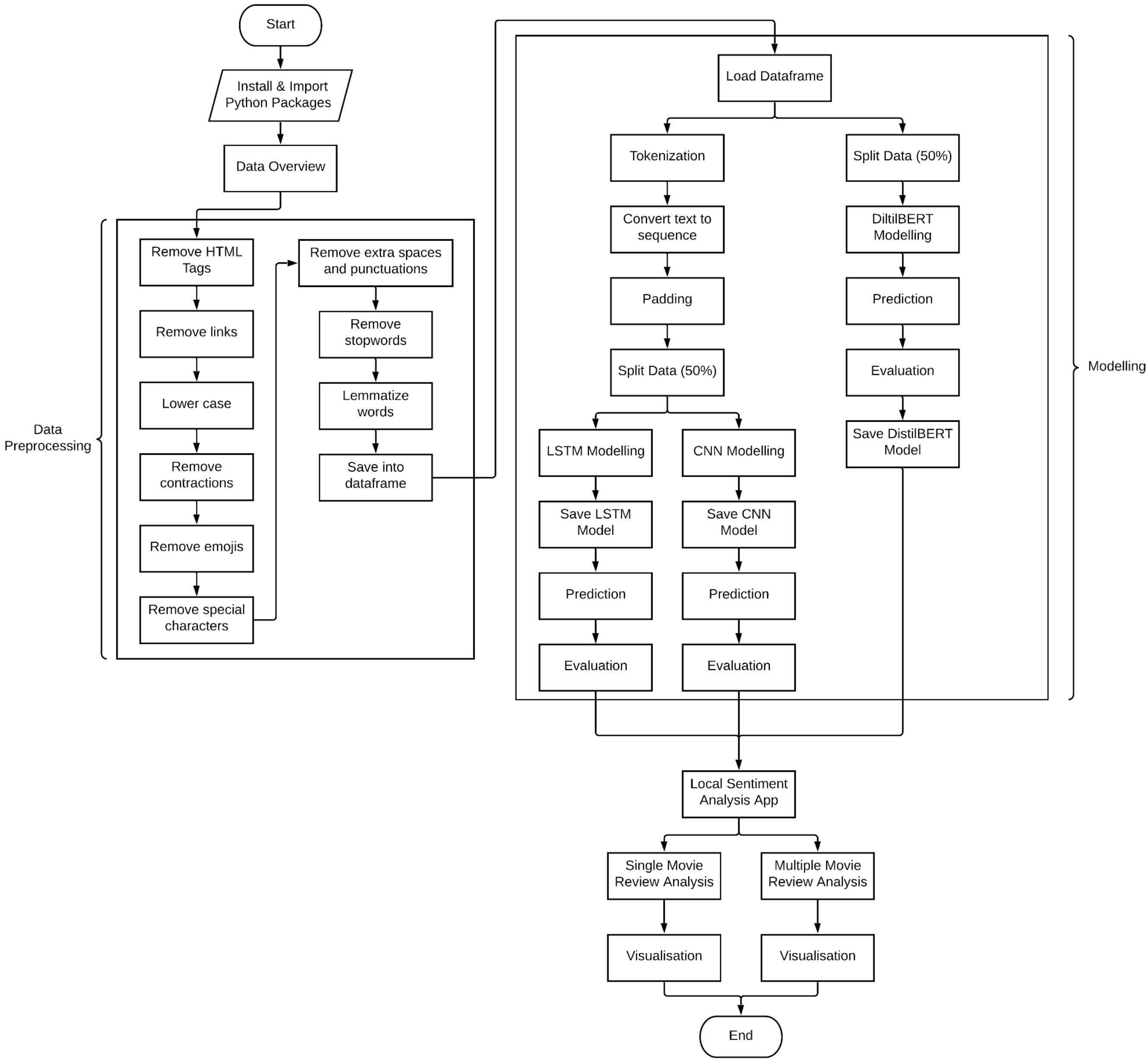

Figure 3.14: System Flowchart

3.4 Proposed test plan/hypothesis

3.4.1 Model Training Test Plan

Below is the test plan for the performance of different models after training using 50% of the preprocessed IMDb dataset. Another 50% are used for testing. Results are as below:

Dataset	User Requirement	Model	Expected Result	Actual Result	Pass / Fail
cleaned_IMDB.csv	Evaluate the performance of all models in terms of accuracy, recall, precision, F1-Score & Confusion Matrix	DistilBERT	Accuracy > 0.5 Recall > 0.5 Precision > 0.5 F1-score > 0.5	Accuracy = 0.93 Recall = 0.94 Precision = 0.93 F1-score = 0.93	Pass
		LSTM		Accuracy = 0.8305 Recall = 0.842 Precision = 0.8231 F1-score = 0.8324	Pass
		CNN		Accuracy = 0.8460 Recall = 0.8110 Precision = 0.8720 F1-score = 0.8404	Pass

NOTE: Please refer to **section 4.1.1** for the confusion matrix results.

3.4.2 Single Movie Review Analysis Test Plan

Below is the test plan for the performance of different models when receiving some reviews from users. Note, these models h are trained using the IMDb dataset.

Movie Review Test	User Requirement	Model	Expected Result	Actual Result	Pass / Fail
Negative test: 'Disturbingly Cringe. It's all bad jokes and cringes. An hour and a half of my life wasted'	Identify the sentiment of this review	DistilBERT	Predict it as negative (Score > 0.5)	Negative (Score = 0.97136)	Pass
		LSTM	Predict it as negative (Score > 0.5)	Negative (Score = 0.99919)	Pass
		CNN	Predict it as negative (Score > 0.5)	Negative (Score = 0.97665)	Pass
Negative test: 'I didn't really like the plot. It felt like it was trying to do way too many things at once. Overall the acting was good but the plot needed a rewrite or more editing.'	Identify the sentiment of this review	DistilBERT	Predict it as negative (Score > 0.5)	Negative (Score = 0.95105)	Pass
		LSTM	Predict it as negative (Score > 0.5)	Negative (Score = 0.99459)	Pass
		CNN	Predict it as negative (Score > 0.5)	Negative (Score = 0.97665)	Pass
Positive test: 'Fun, vibrant and endlessly	Identify the sentiment of this review	DistilBERT	Predict it as positive (Score > 0.5)	Positive (Score = 0.97162)	Pass

entertaining. There wasn't a moment of this film that I didn't enjoy.'		LSTM	Predict it as positive (Score > 0.5)	Positive (Score = 0.94405)	Pass
		CNN	Predict it as positive (Score > 0.5)	Positive (Score = 0.74551)	Pass
<u>Positive test</u>: 'I love it. A very good movie with handsome Ryan Reynolds AKA Deadpool. Everywhere is some good easter egg here and I really enjoy it.'	Identify the sentiment of this review	DistilBERT	Predict it as positive (Score > 0.5)	Positive (Score = 0.99688)	Pass
		LSTM	Predict it as positive (Score > 0.5)	Positive (Score = 0.99284)	Pass
		CNN	Predict it as positive (Score > 0.5)	Positive (Score = 1.0)	Pass

3.4.3 Multi Movie Reviews Analysis Test Plan

Below is the test plan for the performance of different models when identifying each review's sentiment from our crawled dataset which has been well-labelled with negative and positive. Note, these models are trained using the IMDb dataset.

Dataset	User Requirement	Model	Expected Result	Actual Result	Pass / Fail
metacritic	Evaluate the performance of all models in terms of accuracy, recall, precision & F1-Score	DistilBERT	Accuracy > 0.5 Recall > 0.5 Precision > 0.5 F1-score > 0.5	Accuracy = 0.8535 Recall = 0.7741 Precision = 0.8892 F1-score = 0.8277	Pass
		LSTM		Accuracy = 0.489 Recall = 0.4693 Precision = 0.5362 F1-score = 0.5006	Fail
		CNN		Accuracy = 0.4394 Recall = 0.4788 Precision = 0.3108 F1-score = 0.3769	Fail

Result

4.1 Results

4.1.1 Model Training

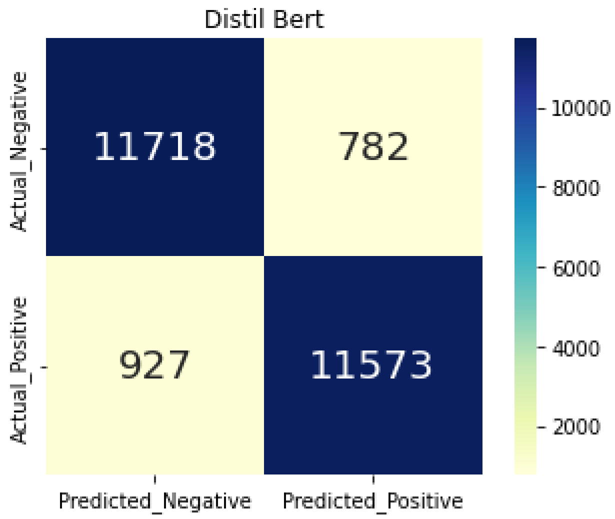

Figure 4.1: DistilBERT Confusion Matrix

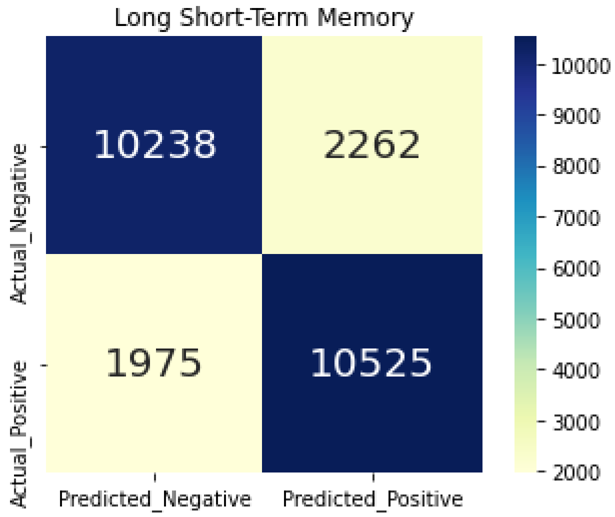

Figure 4.2: Long Short-Term Memory Confusion Matrix

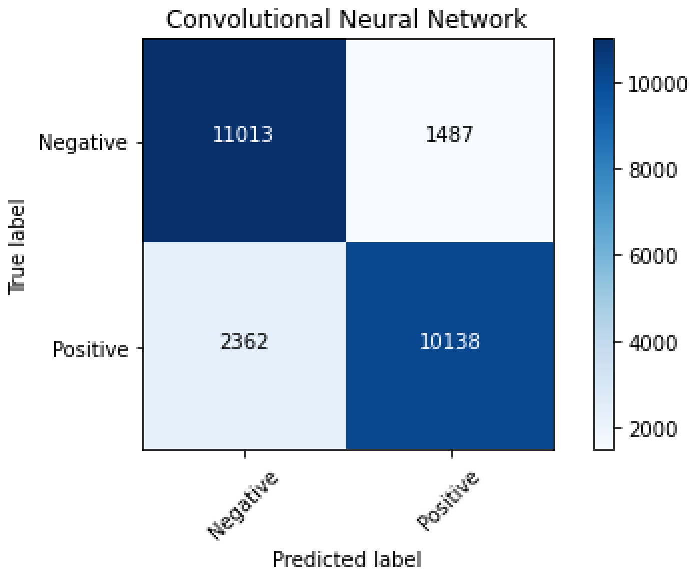

Figure 4.3: Convolutional Neural Network Confusion Matrix

	precision	recall	f1-score	support
neg	0.93	0.94	0.93	12500
pos	0.94	0.93	0.93	12500
accuracy			0.93	25000
macro avg	0.93	0.93	0.93	25000
weighted avg	0.93	0.93	0.93	25000

Figure 4.4: DistilBERT Classification Report

	precision	recall	f1-score	support
0	0.84	0.82	0.83	12500
1	0.82	0.84	0.83	12500
accuracy			0.83	25000
macro avg	0.83	0.83	0.83	25000
weighted avg	0.83	0.83	0.83	25000

Figure 4.5: Long Short-Term Memory Classification Report

	precision	recall	f1-score	support
0	0.82	0.88	0.85	12500
1	0.87	0.81	0.84	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Figure 4.6: Convolutional Neural Network Classification Report

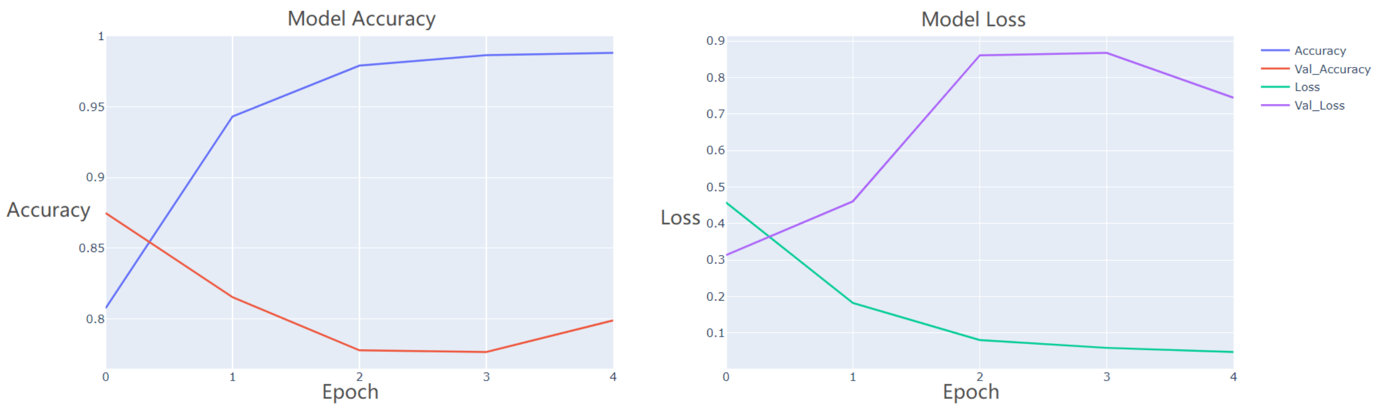

Figure 4.7: Long Short-Term Memory Model Accuracy and Model Loss

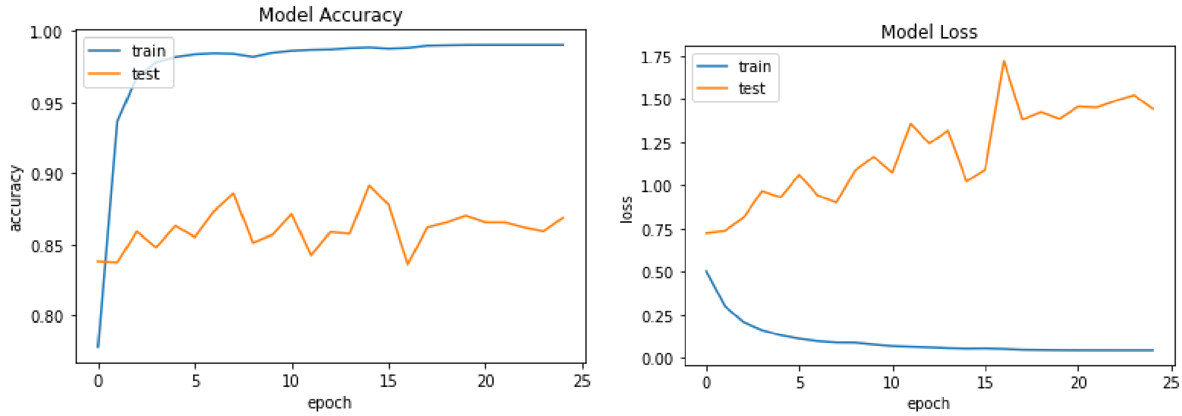

Figure 4.8: Convolutional Neural Network Model Accuracy and Model Loss

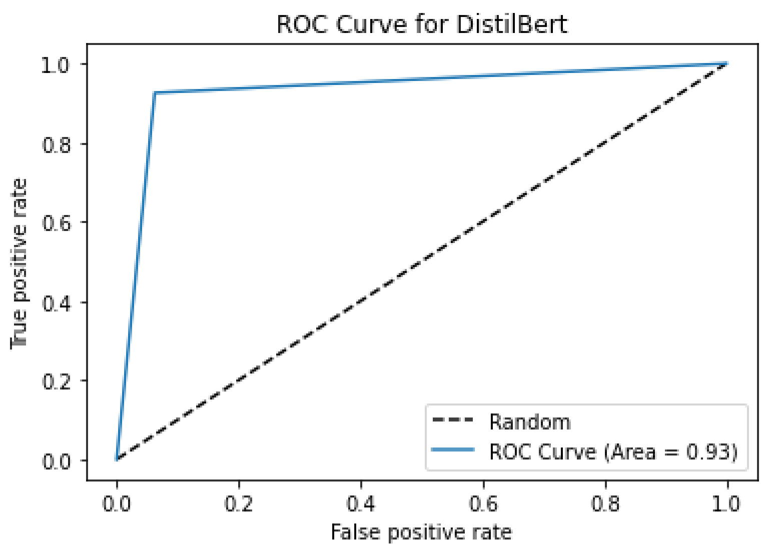

Figure 4.9: DistilBERT Classification Report ROC Curve

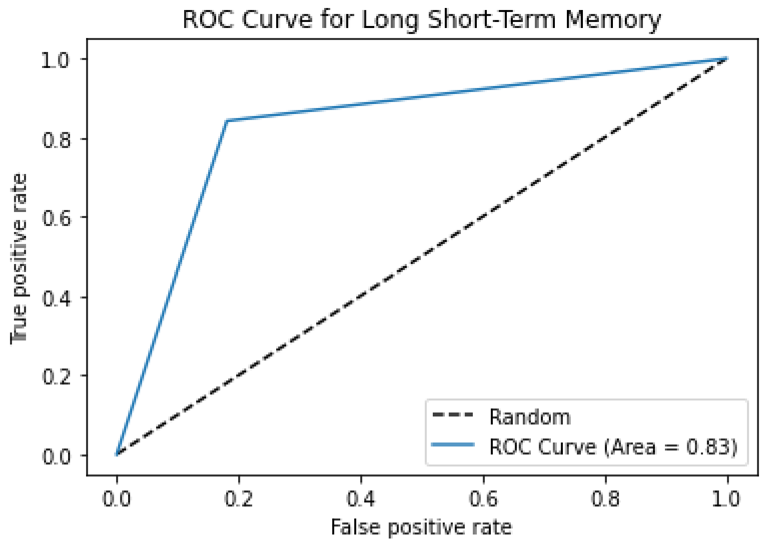

Figure 4.10: Long Short-Term Memory ROC Curve

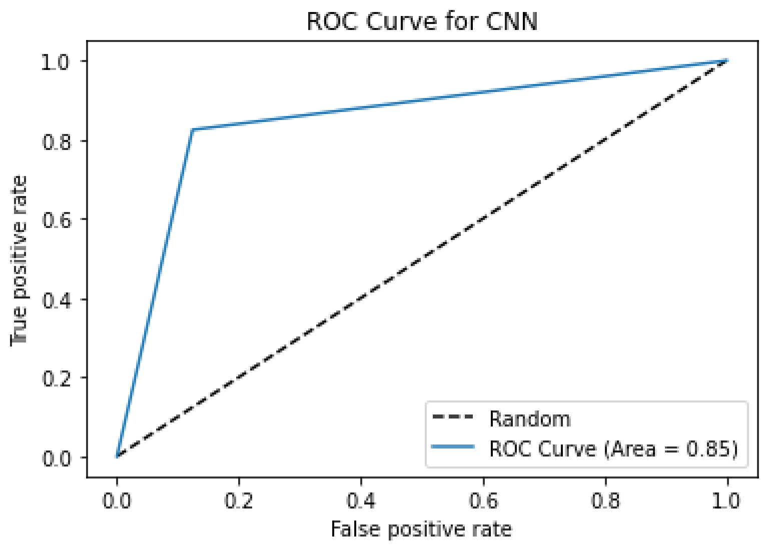

Figure 4.11: Convolutional Neural Network ROC Curve

4.1.2 Single Movie Review Analysis

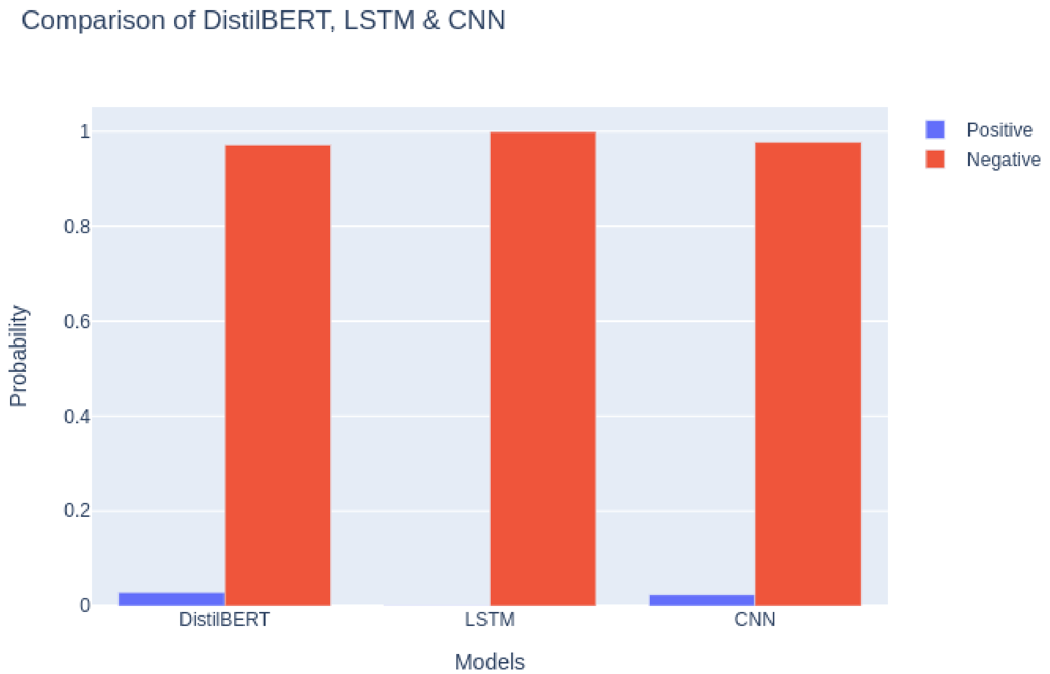

Figure 4.12: Comparison of DistilBERT, LSTM & CNN of a negative review (Single Movie Review)

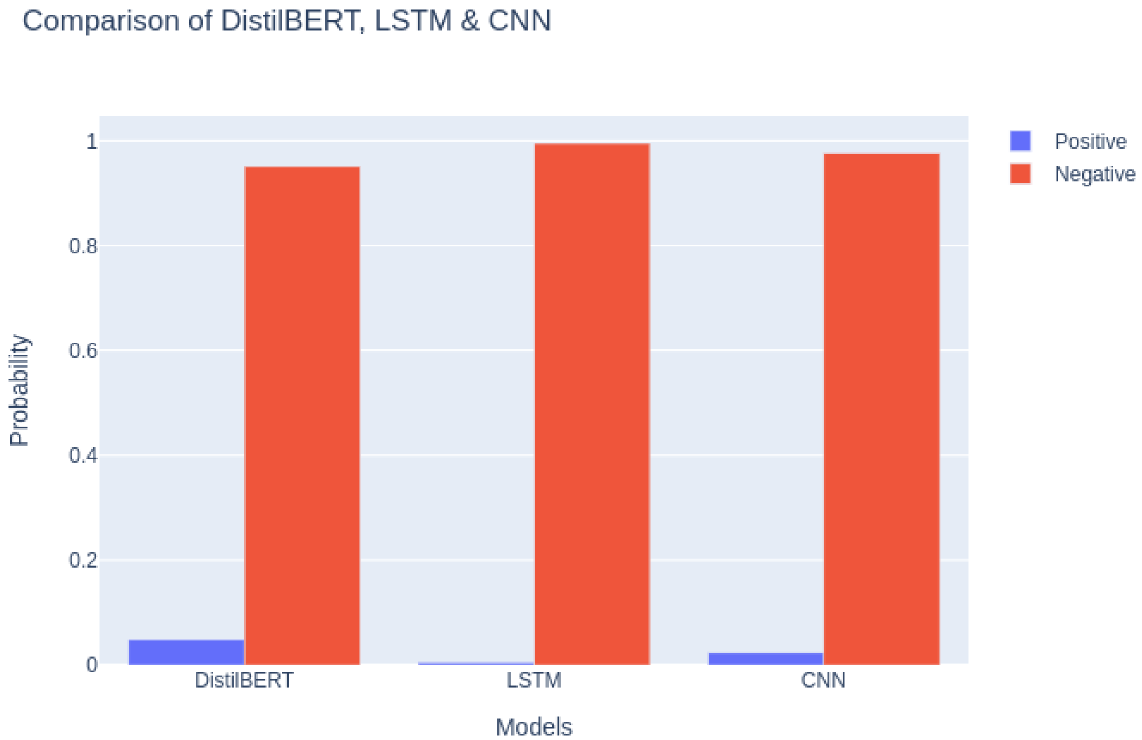

Figure 4.13: Comparison of DistilBERT, LSTM & CNN of a negative review (Single Movie Review)

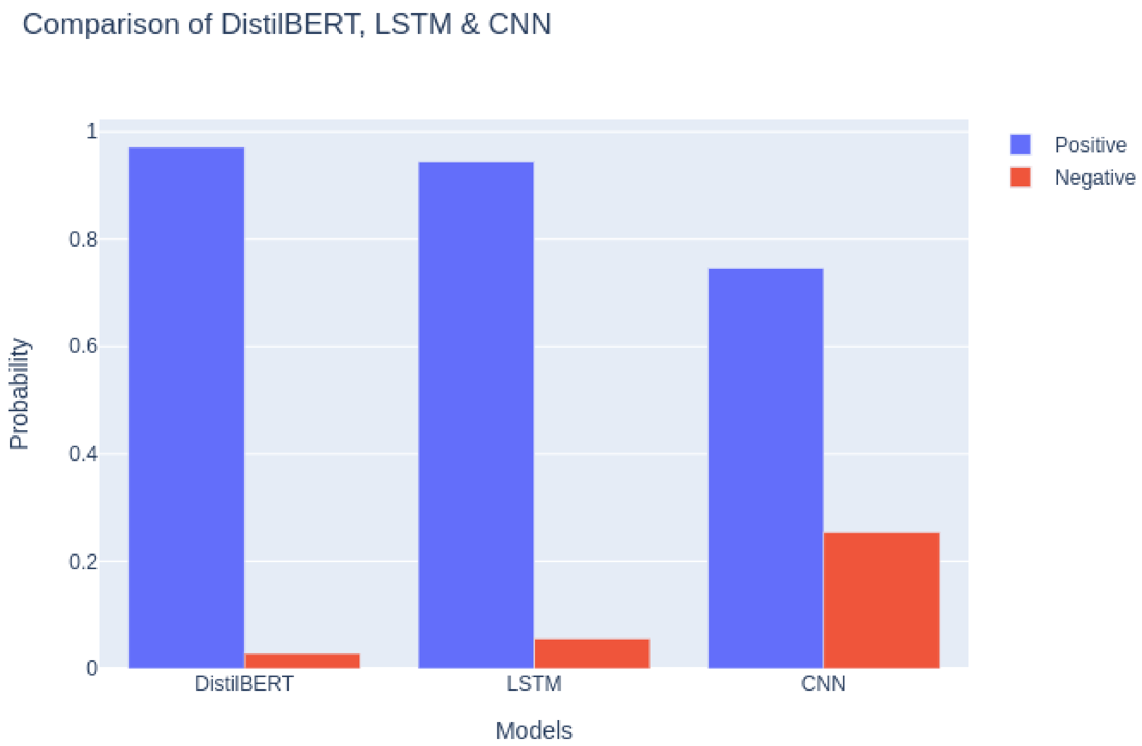

Figure 4.14: Comparison of DistilBERT, LSTM & CNN of a positive review (Single Movie Review)

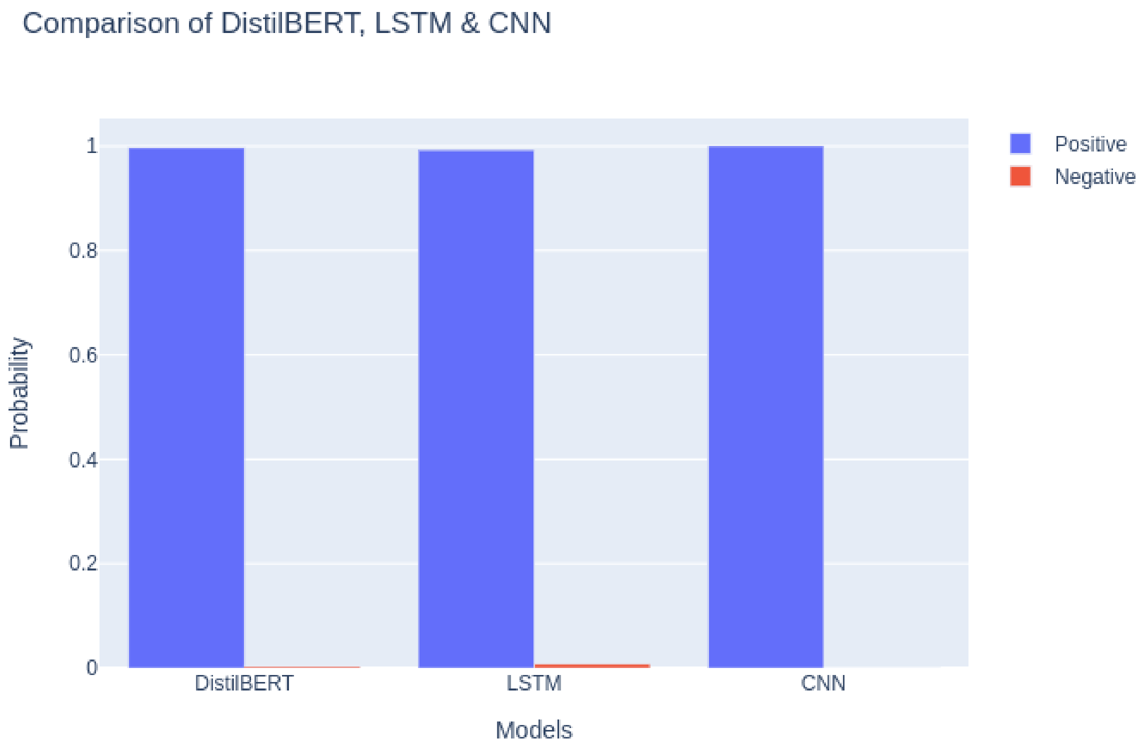

Figure 4.15: Comparison of DistilBERT, LSTM & CNN of a positive review (Single Movie Review)

4.1.3 Multiple Movie Reviews Analysis

Model	Accuracy_score	Recall_score	Precision_score	F1_score
DistilBERT	0.8535	0.7741	0.8892	0.8277
LSTM	0.489	0.4693	0.5362	0.5006
CNN	0.4394	0.4788	0.3108	0.3769

Figure 4.16: Model Performances Report (Multiple Movie Reviews)

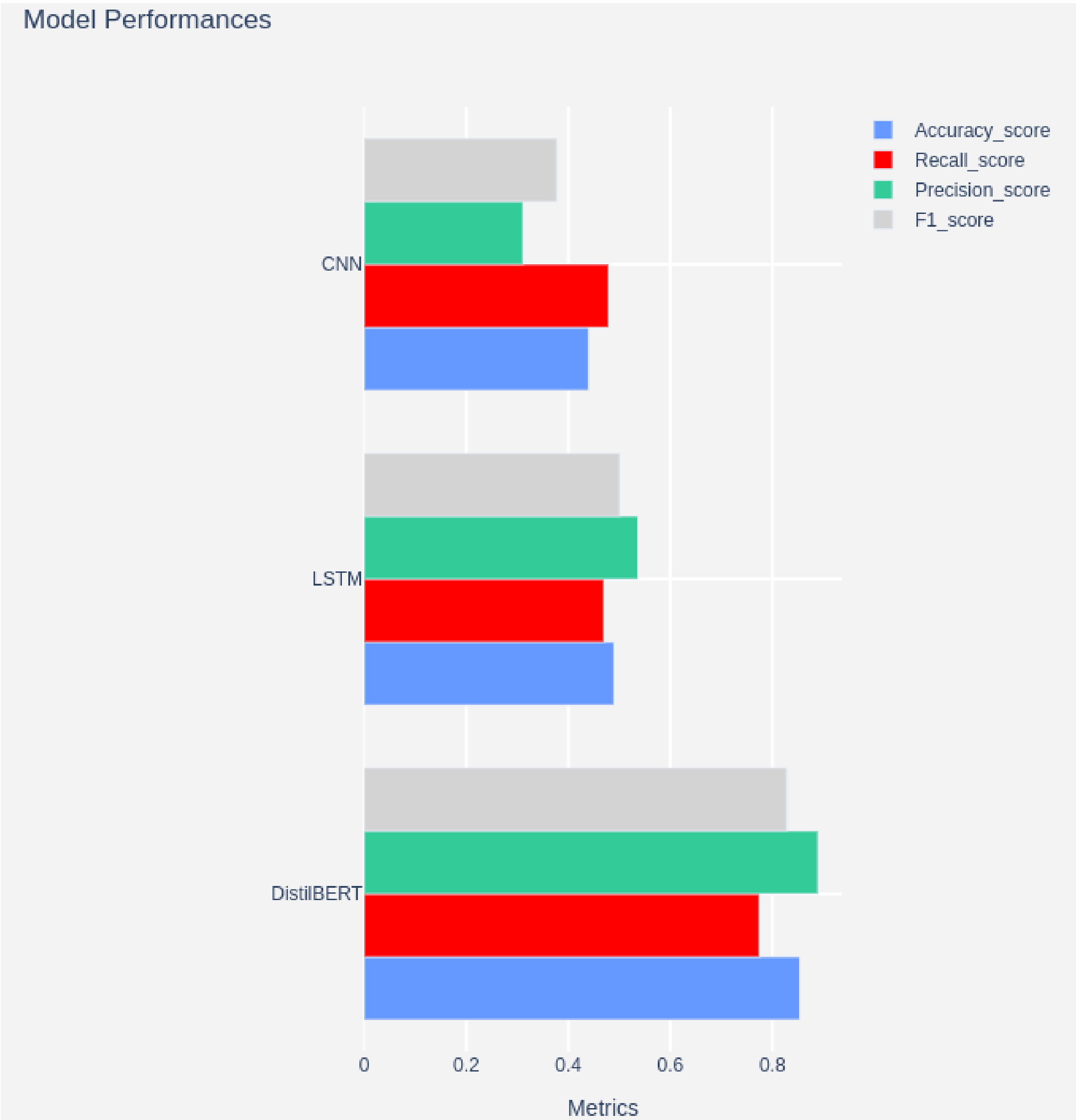

Figure 4.17: Model Performances Report (Multiple Movie Reviews)

4.2 Discussion/Interpretation

Based on the above results, each of the models can do **pretty well performing** sentiment analysis on the IMDb test dataset after going through the training with the IMDb train dataset. Each model can achieve **accuracy of more than 80%** where DistilBERT recorded the highest accuracy 93%, followed by CNN 85% and then LSTM 83%, indicating that DistilBERT model outperforms the other 2 models. There is a clear-cut difference in performance between DistilBERT and other 2 models. When it comes to **multiple movie review analysis** which these models need to do sentiment prediction on the scraped dataset that they have never trained before, the clear-cut difference in performance between these 3 models is further widened where **DistilBERT recorded the highest accuracy 85%**, followed by LSTM with only accuracy of 49% which slightly better than CNN with 44% of accuracy. Even though these 3 models have been well trained with the IMDb dataset, the performance of the **LSTM and CNN model dropped dramatically** when they were tasked to **predict the sentiment on the unseen dataset**. However, DistilBERT remains stable where its accuracy score maintains at 85% and same goes to other metrics scores such as F1-score, recall score still remains at 80% above, just that its precision slightly drops to 77% but overall still performs far better than other 2 models.

This is because the DistilBERT model is **highly pre-trained** with millions of unannotated texts on the web where BERT pre-trained using the plain text corpus of Wikipedia. By doing this so, the DistilBERT model can then be **fine-tuned on smaller task-specific datasets**, e.g., when working with problems like question answering and sentiment analysis. Back to our case, the LSTM and CNN models are **not a pre-trained** type of model. As a result, when we use the LSTM and CNN models to predict the sentiment on the text that have never trained before, their **accuracy will drop significantly** as these 2 models have trained insufficient text data, whereas the DistilBERT model can still perform very well after fine-tuning it with the IMDB dataset because it has been highly pre-trained with millions of text data. So, from this assignment, we can observe that there is a big difference between training on a smaller task-specific dataset from scratch and fine-tuning on a smaller task-specific dataset using a highly pre-trained model.

Apart from that, LSTM had the lowest accuracy during the model training of the IMDB dataset. Even though LSTM resolved the drawbacks of RNN and might have outperformed other models. Several fine tunings have been gone through including the reducing of epoch from 50 to 5, still have **failed to avoid overfitting** on LSTM. Therefore, LSTM does not perform well for the testing dataset. Moreover, LSTM has a major shortcoming here is that it contains both long-term and short-term memory, which makes it **require more memory to train the model**, resulting in longer training period.

Discussion and Conclusion

5.1 Achievements

Throughout this assignment, we have successfully built an application or program called **Local Sentiment Analyzer App** that can help to identify the sentiment of a movie review. The application is able to receive the movie review input by users and identify the sentiment of the input sentence. The system will not only show the identified sentiment of the sentence but also will show the confidence score of getting the identified sentiment. Meanwhile, we also have successfully trained a strong enough deep learning model called **DistilBERT model** that can help to do multiple movie reviews sentiment analysis to predict each movie review sentiment provided by users. This is such a quite big achievement for us as the whole process of building this application and model was not easy at all as we were exposed to some advanced deep learning models that we have not learned before such as DistilBERT, LSTM and CNN. Furthermore, we were just exposed to some fundamental machine learning models like KNN, decision tree etc and basic deep learning models like ANN only in our previous study.

Therefore, we have to explore everything ourselves in this assignment when we deal with all the advanced deep learning models like DistilBERT, LSTM, RNN and CNN. We have gone through many tutorials, articles and videos online in order to understand the functions and features provided by those deep learning models and learned how to apply it in the assignment so that we can build the models that we want to build and try our best to make all those models perform better. Some most unexpected gain we got from this assignment is we found out some very useful and yet powerful python library, which is DistilBERT for NLP modelling, cleantext and contractions for text preprocessing. In a nutshell, this assignment has achieved the requirement that we wanted in the beginning where it **achieved our goal of identifying the sentiment of each movie review** and **trained a strong enough deep learning model** that can **carry out sentiment prediction on multiple reviews**.

5.2 Limitations and Future Works

Even if our application or program can identify the correct sentiment of each text sentence input by users for most of the time. However, it is **not convenient enough** for the user to use it. This is because the **program or application is deployed locally** in the local machine (PC). If other people want to try to use our application, they need to set up the same workspace as us to run the application. For example, we are using Python version 3.8.0, Windows 10 operating system, NVIDIA CUDA GPU support and so on. They need to have the same workspace requirements with us in order to test or use our application. This is very inefficient and inconvenient. Another solution is using cloud ready workspace like Google Colab where they do not need to set up all those things that I have mentioned just now if we build everything of our application or program using Google Colab, they just need to log in to the Google Colab notebook to test and use our application. However, this is also **inefficient and inconvenient** as they also need to run every cell of code 1 after 1 and it takes time. The best solution and **improvement that can be carried out is to deploy our application online**. For example, we probably can use python web application frameworks like Streamlit, Flask, Gradio and others to build a user-friendly interface application and deploy it online. By doing this, we can just send a link that connects our application to other people to test and use our sentiment analyzer application. This would be very efficient, convenient and comfortable for other people to use. They can just access our application whenever they want as long as their devices are connected to the internet and the user-interface of the application will also make the user feel easier to use. Additionally, a good online deployment of web applications will also make the app run faster than locally without relying on the hardwares and softwares like GPU and Python Interpreter.

References & Sources

1. TensorFlow 2019, TensorFlow, TensorFlow, viewed 18 September 2021, <https://www.tensorflow.org/>.

2. Team, K n.d., Keras documentation: Keras API reference, keras.io, viewed 18 September 2021, <https://keras.io/api/>.

3. Plotly Python Graphing Library n.d., plotly.com, viewed 19 September 2021, <https://plotly.com/python/>.

4. Maiya, AS 2020, amaiya/ktrain, GitHub, viewed 18 September 2021, <https://github.com/amaiya/ktrain>.

5. Team, TA & Team, TA n.d., Text Classification with RNN – Towards AI — The Best of Tech, Science, and Engineering, viewed 18 September 2021, <https://towardsai.net/p/deep-learning/text-classification-with-rnn>.

6. Donges, N 2019, *Recurrent neural networks 101: Understanding the basics of RNNs and LSTM*, Built In, viewed 18 September 2021, <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>.

7. Amidi, A & Amidi, S 2019, *CS 230 - Recurrent Neural Networks Cheatsheet*, Stanford.edu, viewed 18 September 2021, <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>.

8. Katte, A 2018, Meet Juergen Schmidhuber — The Father Of LSTM, An Outsider In The World Of Deep Learning, Analytics India Magazine, viewed 18 September 2021, <https://analyticsindiamag.com/meet-juergen-schmidhuber-the-father-of-lstm-an-outsider-in-the-world-of-deep-learning/>.

9. Nguyen, M 2019, Illustrated Guide to LSTM's and GRU's: A step by step explanation, Medium, viewed 18 September 2021, <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>.

10. Eckhardt, K 2018, *Choosing the right Hyperparameters for a simple LSTM using Keras*, Medium, viewed 18 September 2021, <https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-usng-keras-f8e9ed76f046>.

11. Agrawal, S 2020, *Sentiment Analysis using LSTM Step-by-Step*, Medium, viewed 18 September 2021,

<https://towardsdatascience.com/sentiment-analysis-using-lstm-step-by-step-50d074f09948>.

12. Python, R n.d., *Practical Text Classification With Python and Keras – Real Python*, realpython.com, viewed 18 September 2021, <https://realpython.com/python-keras-text-classification/>.

13. Maheshwari, A 2018, Report on Text Classification using CNN, RNN & HAN, Medium, viewed 18 September 2021, <https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f>.

14. choubey 2020, *Text classification using CNN*, Medium, viewed 18 September 2021, <https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9>.

15. Alammar, J 2018, The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), Github.io, viewed 20 September 2021, <https://jalammar.github.io/illustrated-bert/>.

16. Khalid, S 2019, BERT Explained: A Complete Guide with Theory and Tutorial, Towards Machine Learning, viewed 20 September 2021, <https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/>.