ECE 657 ASSIGNMENT 3 REPORT

Problem 1: CNN for Image Classification

Using your preferred machine learning library, train a small convolutional network (CNN) to classify images from the CIFAR10 dataset. Using the API for loading the dataset will readily divide it into training and testing sets. Randomly sample 20% of the training set and use that as your new training set for the purposes of this problem. Use the test set for validation.

1. **Any pre-processing steps you made**
    a. ***Normalization:*** Scaled the pixel values of the images to a specific range, typically between 0 and 1. In the case of CIFAR dataset, which consists of RGB images, the pixel values range from 0 to 255. Normalized the data by dividing each pixel value by 255 ensures that all pixel values are between 0 and 1.
    b. ***One Hot Encoding:*** CIFAR dataset consists of categorical labels, which are the class labels representing the different categories of images. Each label is converted into a binary vector of the same length as the number of classes, with a 1 at the index corresponding to the class and 0 elsewhere.
    c. ***Data Shuffling:*** By Shuffling the data, the overall data distribution of each batch becomes generalized.
2. **Description of the output layer used and the loss function (use 1 setting for all 3 networks) and why you made these choices.**

    We used SoftMax activation function for the output layer and it's commonly used for the multiclass classification tasks. It takes the vector of real numbers as a inputs and outputs a probability distribution over multiple classes. The class with the highest probability is considered the predicted class.

    We used categorical cross entropy as a loss function which is well suited for multiclass classification. This loss function calculates the cross entropy between the predicted probabilities and the one hot encoded true label. By minimizing the cross-entropy loss during training, the model learns to improve its predictions and converge towards the correct class probabilities.

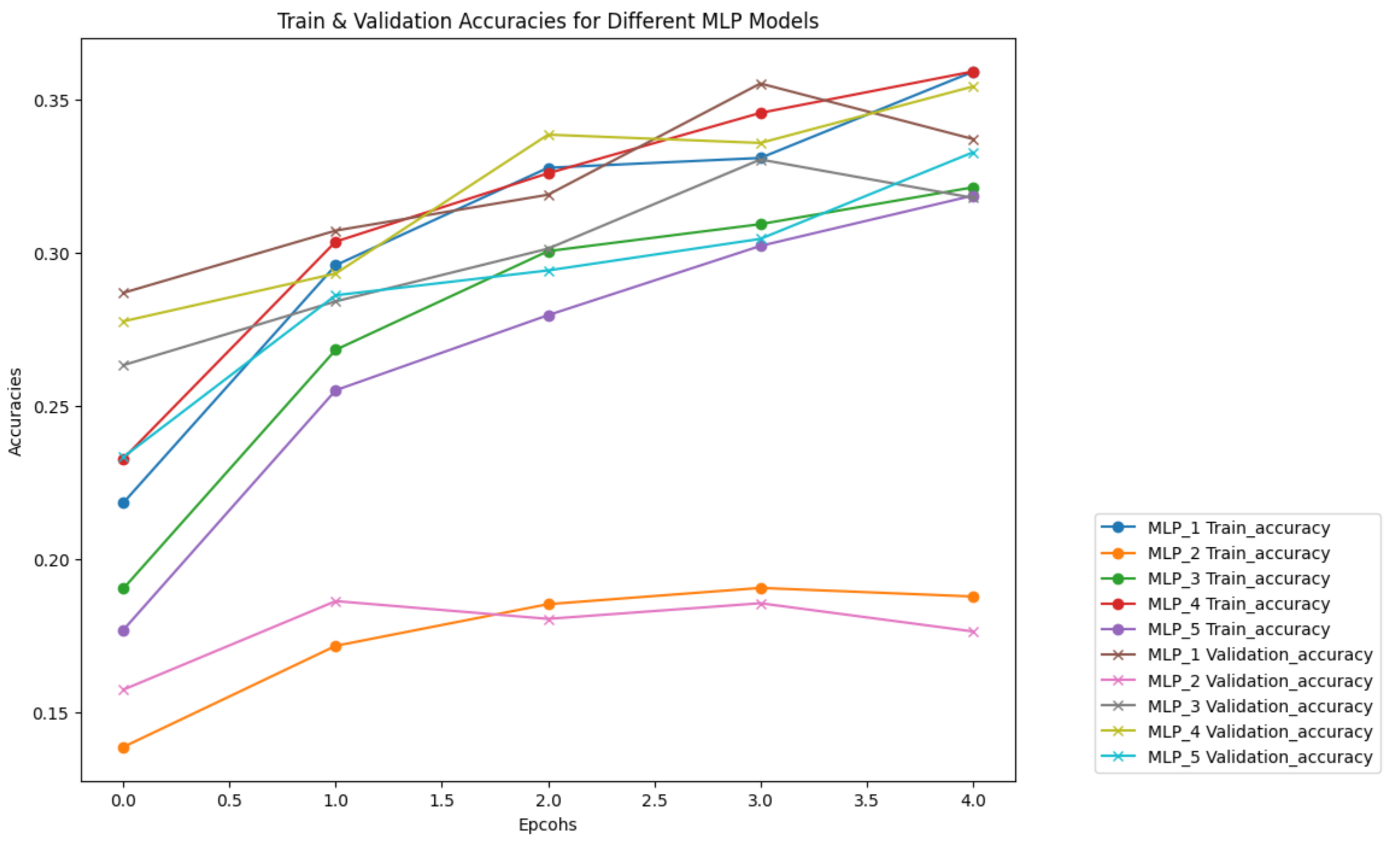
3. **Change the number of layers and the number of neurons per layer in the MLP, plot/tabulate the training and validation accuracies and comment on the results.**

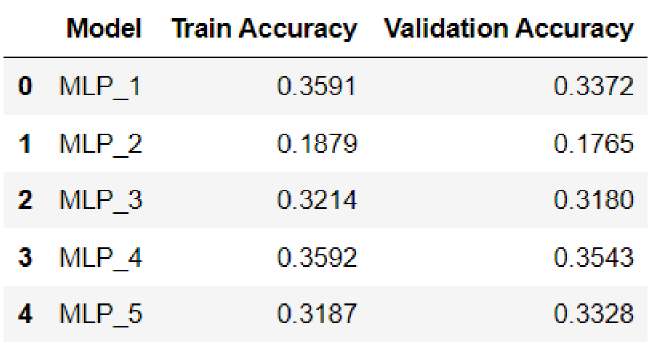
We have created a 4 MLP models by changing the number of layers and number of neurons per layer.

- Reference MLP (MLP model 1): 2 Hidden Layers and each layer has 512 neurons
- MLP model 2: 5 hidden layers with 512,256,128,64,32 neurons for each layer respectively
- MLP model 3: 3 hidden layers and each layer has 256 neurons
- MLP model 4: 2 hidden layers with 1024 and 512 neurons for each layer respectively
- MLP model 5: 3 hidden layers with 1024, 512 and 256 neurons for each layer respectively

Comparison:

Here is the comparison of the five MLP models with the reference model (MLP model 1) in terms of their train accuracy and validation accuracy:





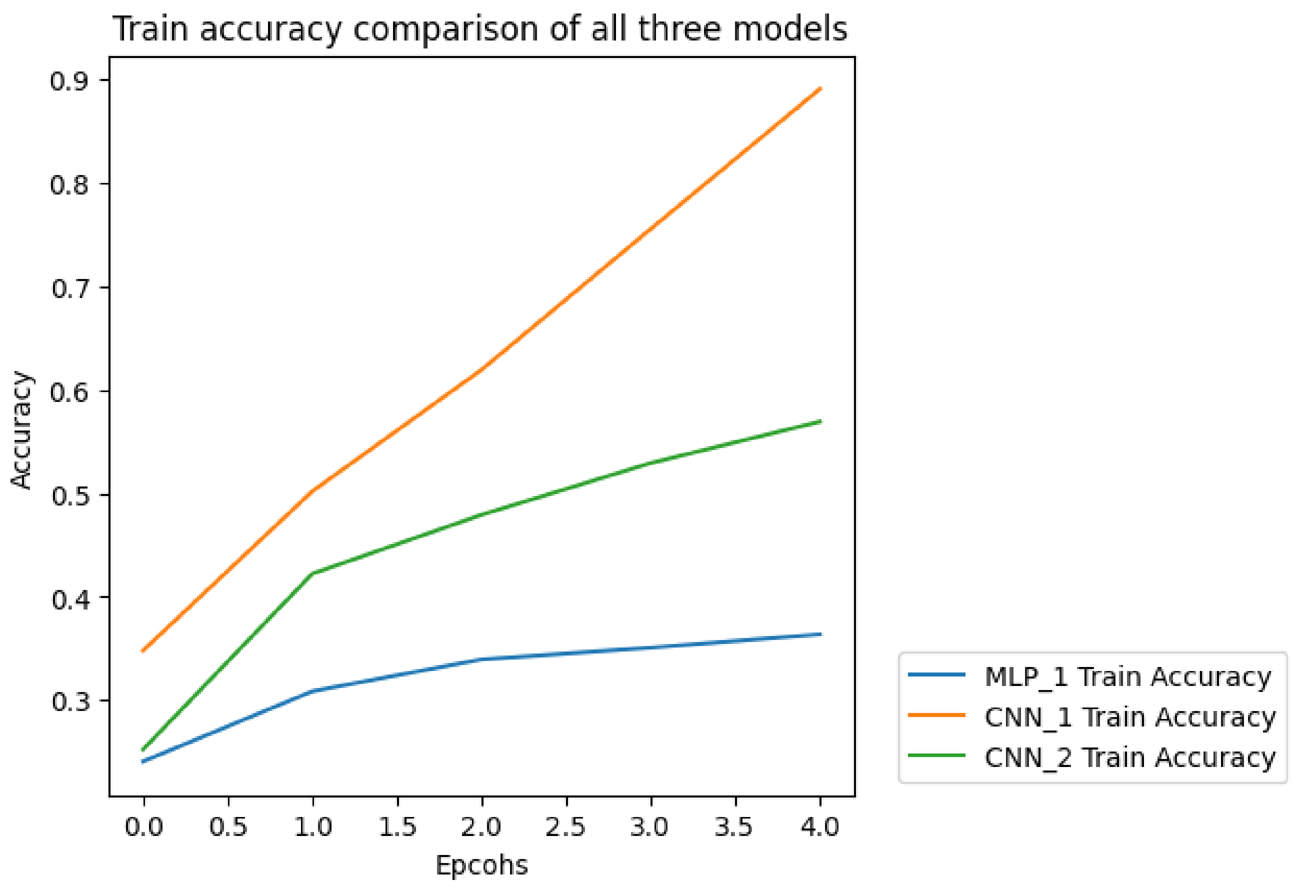
	Model	Train Accuracy	Validation Accuracy
0	MLP_1	0.3591	0.3372
1	MLP_2	0.1879	0.1765
2	MLP_3	0.3214	0.3180
3	MLP_4	0.3592	0.3543
4	MLP_5	0.3187	0.3328

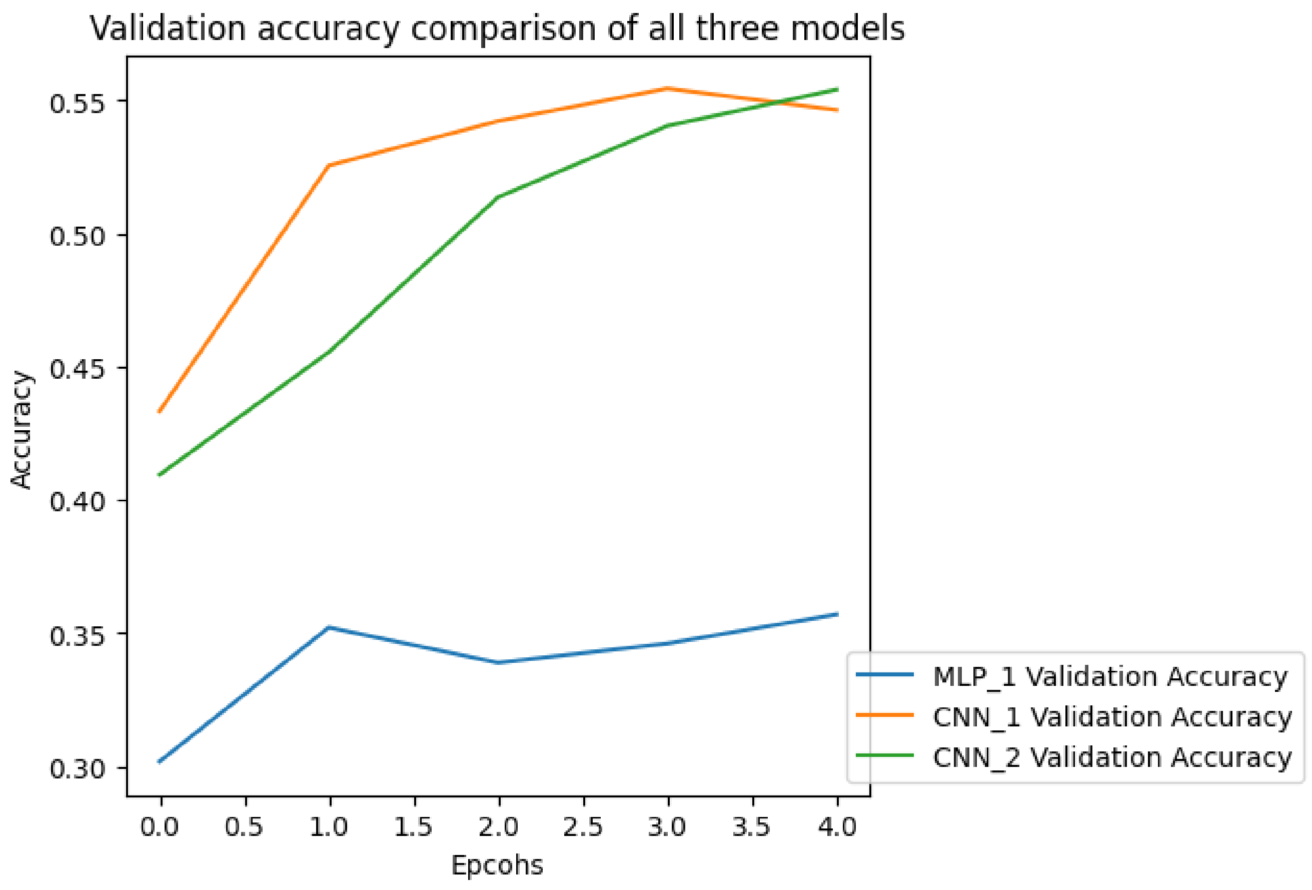
From the plots and table, we observed that,

- MLP_1 (Given): It performs the best among all the models in terms of both train accuracy (0.35) and validation accuracy (0.33)
- MLP_2: This model is deeper and more complex with five hidden layers and decreasing neurons. However, it achieves lower accuracy on both train (0.1879) and validation (0.1765) sets, indicating that the complexity may cause overfitting and difficulty in optimization.
- MLP_3: A simpler model compared to MLP_2, it has three hidden layers with 256 neurons in each layer. It performs better than MLP_2 but still has lower accuracy than the reference model, with train accuracy of 0.3214 and validation accuracy of 0.3180.
- MLP_4: This model has larger hidden layers compared to MLP_3, with 1024 neurons in the first hidden layer. It achieves slightly better results than MLP_1 in terms of both train accuracy (0.3592) and validation accuracy (0.3543).
- MLP_5: An extension of MLP_4 with an extra hidden layer containing 256 neurons. The additional complexity did not significantly improve the performance, resulting in a train accuracy of 0.3187 and validation accuracy of 0.3328.

The reference model (MLP_1) with 2 hidden layers and 512 neurons in each layer performs the best among the models, demonstrating that a moderate number of layers and neurons can achieve better generalization in this scenario of limited data. MLP_2, being deeper and more complex, suffers from overfitting and optimization challenges. MLP_4 shows promising results with larger hidden layers, while the additional complexity in MLP_5 offers minimal improvement.

4. **Train and test accuracy for all three networks and comment (what happens and why) on the performance of the MLP vs CNNs.**



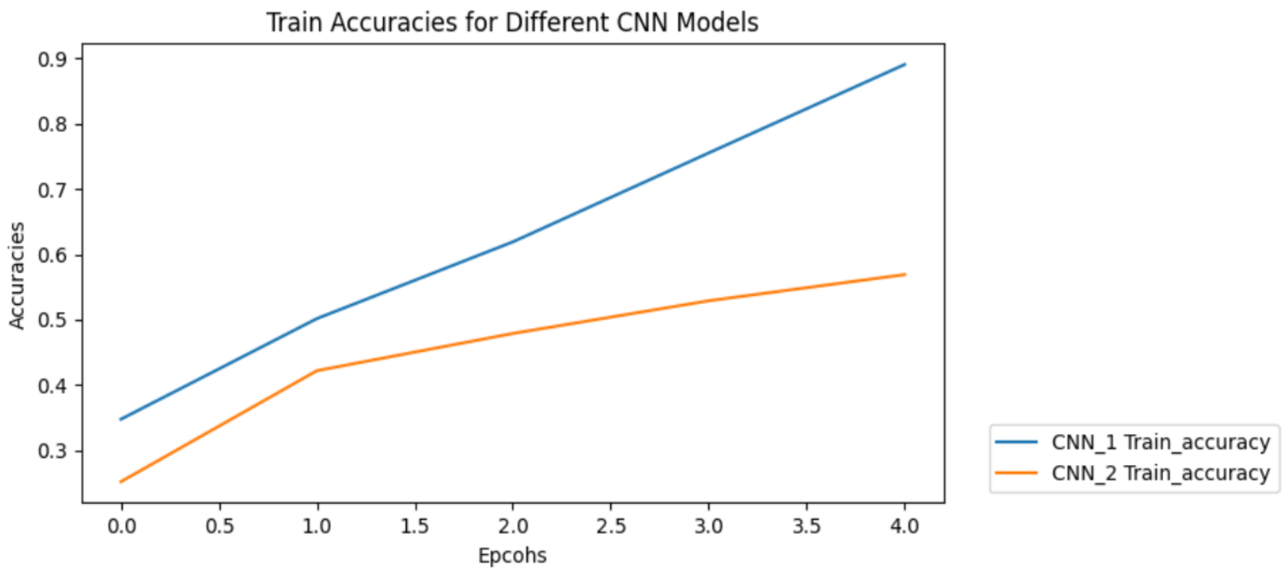


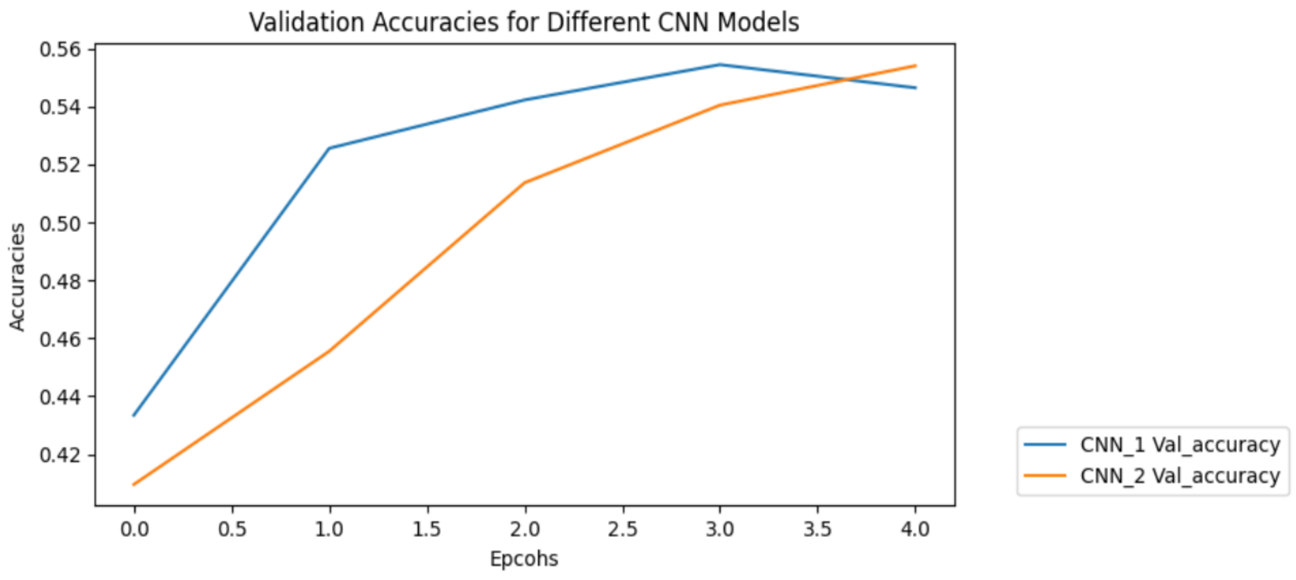
CNN 1 achieved the highest training accuracy among the models, while the MLP performed the worst in terms of training accuracy, lagging behind both CNNs. CNN 1 demonstrated a rapid and significant improvement in accuracy over the epochs, whereas CNN 2 showed a more moderate rate of improvement. On the other hand, the MLP had the slowest progress in training accuracy.

Regarding validation accuracy, CNN 2 eventually obtained the highest accuracy, but before that, CNN 1 showed considerable improvements. The MLP consistently had the lowest validation accuracy throughout the 5 epochs.

MLP is a basic neural network not suitable for tasks like image recognition, slower improvement in accuracy, Whereas CNN is specifically designed for grid-like data (e.g., images), excels at capturing spatial patterns and features, faster and more significant improvement in accuracy for image-related tasks.

5. **Plot the training and validation curves for the two CNNs and comment on the output.**





CNN 1 lacks Max Pooling and Dropout layers. In the case of training performance, it achieved the highest training accuracy due to its ability to learn complex patterns with the multiple convolutional layers but in the case of validation set, it showed considerable improvement in accuracy but it doesn't perform well like CNN2. It means it struggled to generalize well to unseen data, potentially due to overfitting.

CNN2 has Max Pooling and Dropout layers for regularization. CNN2 showed a moderate improvement in training accuracy due to regularization effect of Max Pooling and Dropout layers. Eventually, CNN2 obtained the highest validation accuracy, indicating better generalization to unseen data. The Max Pooling and Dropout layers contributed to reducing overfitting.

a. **How does the training time compare for each of the CNNs? How does the different architectures influence these results?**

CNN1 has a much larger number of trainable parameters (25,997,130), which means it requires more computations and memory to train the model during each epoch so it took 250 ms average. On the other hand, CNN2 has substantially fewer trainable parameters (1,486,666), making it quicker to update the weights and biases during training so it took 35 ms average.

The reduction in the number of parameters in CNN2 is a result of its simpler architecture, which includes fewer convolutional filters and smaller Dense layers. This simpler architecture leads to faster training times, which can be advantageous when working with large datasets.

b. **What do you expect the accuracies to be if the networks were trained for more epochs?**

When the number of epochs was increased from 5 to 10 for CNN model 1, the training accuracy reached 100%, indicating potential overfitting. However, the validation accuracy only reached 56%, suggesting that the model did not generalize well to unseen data.

In contrast, when the number of epochs was increased from 5 to 10 for CNN model 2, the training accuracy reached 91%, indicating good performance on the training data. Additionally, the validation accuracy reached 60.80%, suggesting that the model performed relatively well on unseen data.

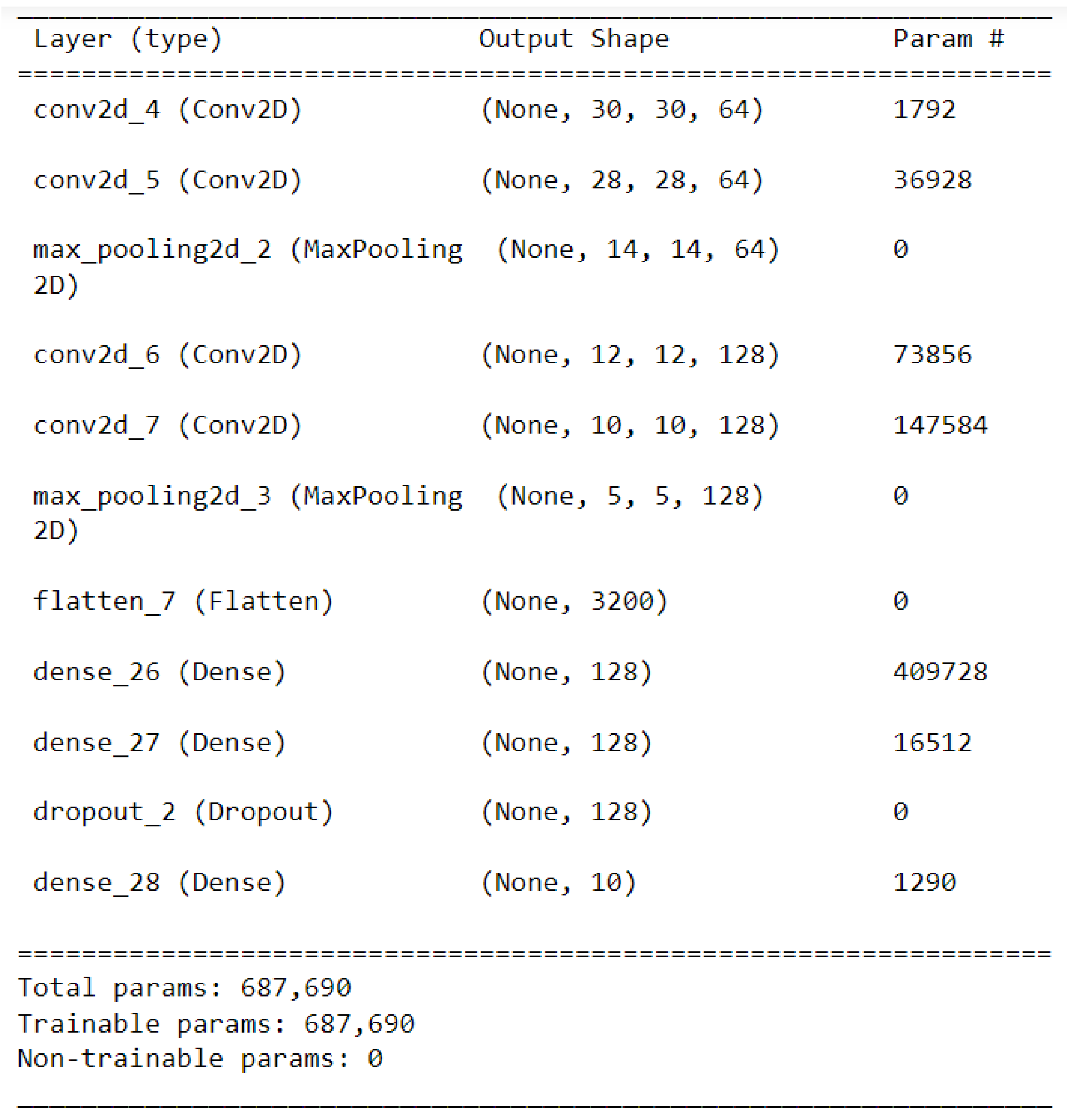
6. **Recommendations to improve the network. What changes would you make to the architecture and why?**

Improvements in the model_cnn3:

a. ***Increased Depth:*** model_cnn3 has more layers, with additional Conv2D and MaxPooling2D layers compared to the previous models. The addition of these layers allows the model to learn more complex and abstract representations of the input data.

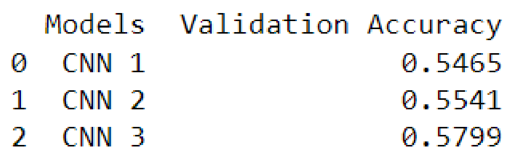
b. ***Large Number of filters:*** In model_cnn3, there are 64 filters in the first two Conv2D layers and 128 filters in the subsequent two Conv2D layers. Increasing the number of filters can help the model to capture a wider range of features

c. ***Use of Dropouts:*** The inclusion of a Dropout layer (with a dropout rate of 0.2) after the Dense layers introduces regularization to prevent overfitting.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 30, 30, 64)        1792

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 12, 12, 128)       73856

 conv2d_7 (Conv2D)           (None, 10, 10, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten_7 (Flatten)         (None, 3200)              0

 dense_26 (Dense)            (None, 128)               409728

 dense_27 (Dense)            (None, 128)               16512

 dropout_2 (Dropout)         (None, 128)               0

 dense_28 (Dense)            (None, 10)                1290

=================================================================
Total params: 687,690
Trainable params: 687,690
Non-trainable params: 0
_________________________________________________________________
```

Performance comparison:



```
  Models  Validation Accuracy
0  CNN 1               0.5465
1  CNN 2               0.5541
2  CNN 3               0.5799
```

- model_cnn3 achieved a training accuracy of 60% and a validation accuracy of 58%.
- Comparing to model_cnn1 and model_cnn2, model_cnn3 shows an improvement in generalization. Although the training accuracy might be lower than model_cnn1, it performs better on the validation set, indicating that it is less likely to be overfitting the data.

Problem 2: Recurrent Neural Networks for Regression

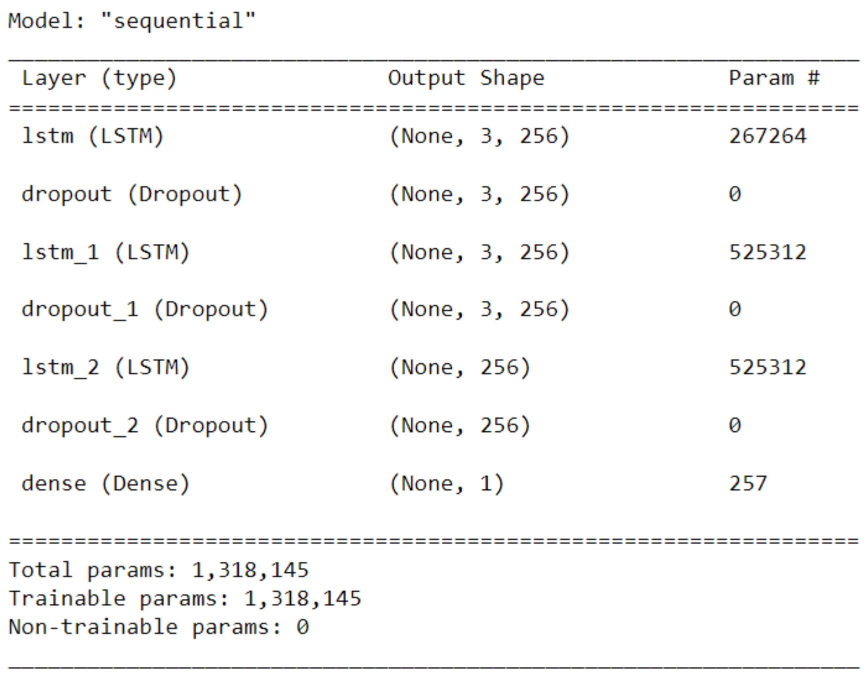
You are provided with a dataset for stock price prediction for 5 years with one sample per day (q2_dataset.py). Create a Recurrent Neural Network using the machine learning platform of your choice to predict the next day opening price using the past 3 days Open, High, and Low prices and volume. Therefore, each sample will have (4*3 =) 12 features.

1. **Explanation of how you created your dataset.**
   - Generated the new dataset for estimating the next day's open price begins by loading the original dataset, which contains various stock-related columns such as date, close/last, volume, open, high, and low prices.
   - To create the dataset for training the RNN model, we defined a feature list consisting of 'Open', 'High', 'Low', and 'Volume'.
   - With the feature list in place, we initiated two empty lists, 'X' and 'y', to store the features and target, respectively, for each sample in the new dataset.
   - Using for loops, we iterated through the original dataset, excluding the last three days since there wouldn't be sufficient data to create a complete sequence of three days as features along with a target.
   - For each iteration, we stored the features by considering the previous three days' data, i.e., 'Open', 'High', 'Low', and 'Volume'. The next day's opening price is considered as the target value.
2. **Any pre-processing steps you followed**
   1. ***a. Normalization:*** We used Min-Max scaling to normalize the features. This process scales the features to a range of [0, 1], ensuring that all features have the same scale.
   2. ***b. Random Shuffling:*** Before splitting the data into training and testing sets, we randomly shuffled the data to prevent any inherent order or pattern in the dataset from influencing the model's performance.
   3. c. ***Train – Test Split:*** The shuffled data was then split into training and testing sets using a 70%-30% split ratio.
3. **All design steps you went through in finding the best network in your report and how you chose your final design**
   - Initially, we started with a basic LSTM model with two layers, each having 50 units. The model's performance was evaluated on the testing set and there was some overfitting observed.
   - To enhance the model's capacity and reduce overfitting, we tried changing the number of LSTM units from 32 to 256 in each layer.

- Additionally, we used return_sequences=True in the first two LSTM layers to enable the model to capture long-term dependencies.
- To prevent overfitting, we introduced Dropout layers with a dropout rate of 0.2 after each LSTM layer.
- We optimized the model loss by varying the LSTM units from 32 to 256 and dropout rates from 0.1 to 0.5.
- After training and evaluating the updated LSTM model, we found that it performed significantly better than the initial model.
- The training and validation losses were well balanced, indicating that the model generalized well to unseen data.
- In our model, we used Mean Squared Error (MSE) as the loss function and Mean Absolute Error (MAE) as the evaluation metric.
- MSE measures the squared average distance between the predicted and actual data, while MAE calculates the absolute average distance.
- Since the features are normalized but the targets are not, MAE is more suitable for assessing prediction accuracy, as it considers the differences in scale between the features and targets.

4. **Architecture of your final network, number of epochs, batch size (if needed), loss function, training algorithm, etc.**

   a. ***Architecture of final network***



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 3, 256)            267264    
                                                                 
 dropout (Dropout)           (None, 3, 256)            0         
                                                                 
 lstm_1 (LSTM)               (None, 3, 256)            525312    
                                                                 
 dropout_1 (Dropout)         (None, 3, 256)            0         
                                                                 
 lstm_2 (LSTM)               (None, 256)               525312    
                                                                 
 dropout_2 (Dropout)         (None, 256)               0         
                                                                 
 dense (Dense)               (None, 1)                 257       
                                                                 
=================================================================
Total params: 1,318,145
Trainable params: 1,318,145
Non-trainable params: 0
_________________________________________________________________
```

Finally, we went with the three LSTM layers with 256 neurons with return sequences for the first two LSTM layers and dropout of 0.2 was added after every LSTM layer.

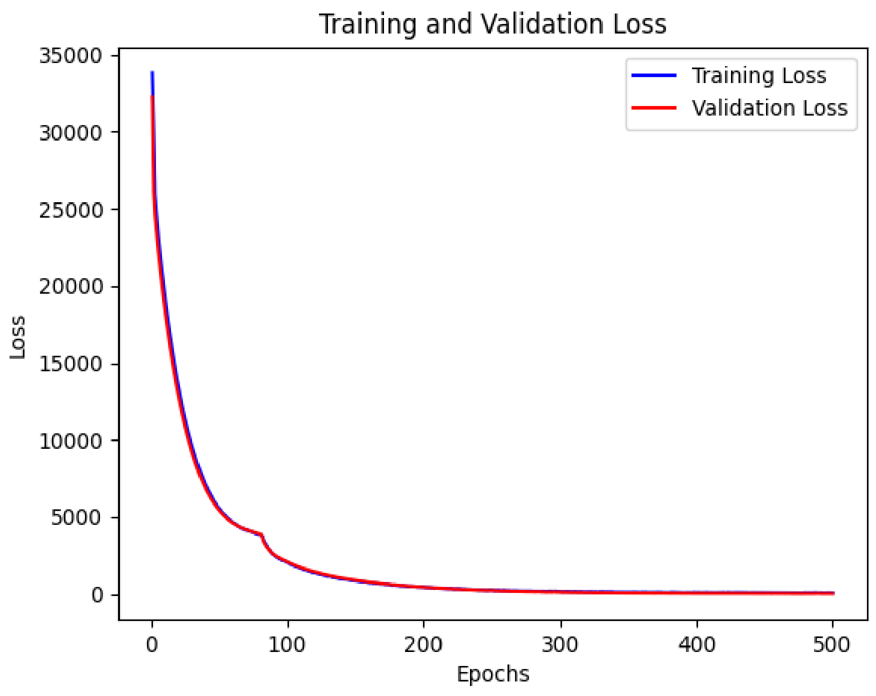
b. Epochs, batch size

- We used 500 epochs. In this case it might allow model to converge to a more optimal solution, potentially resulting in the better accuracy for both training and validation accuracy.
- Larger batch size can provide more stable updates to the model's parameters.

c. Optimizer and Loss function (training algorithm)

- The purpose of the Adam optimizer is to efficiently update the model's parameters during training in order to minimize the loss function and improve the model's performance.
- In our model, we used Mean Squared Error (MSE) as the loss function and Mean Absolute Error (MAE) as the evaluation metric.
- MSE measures the squared average distance between the predicted and actual data, while MAE calculates the absolute average distance.

5. Output of the training loop with comments on your output

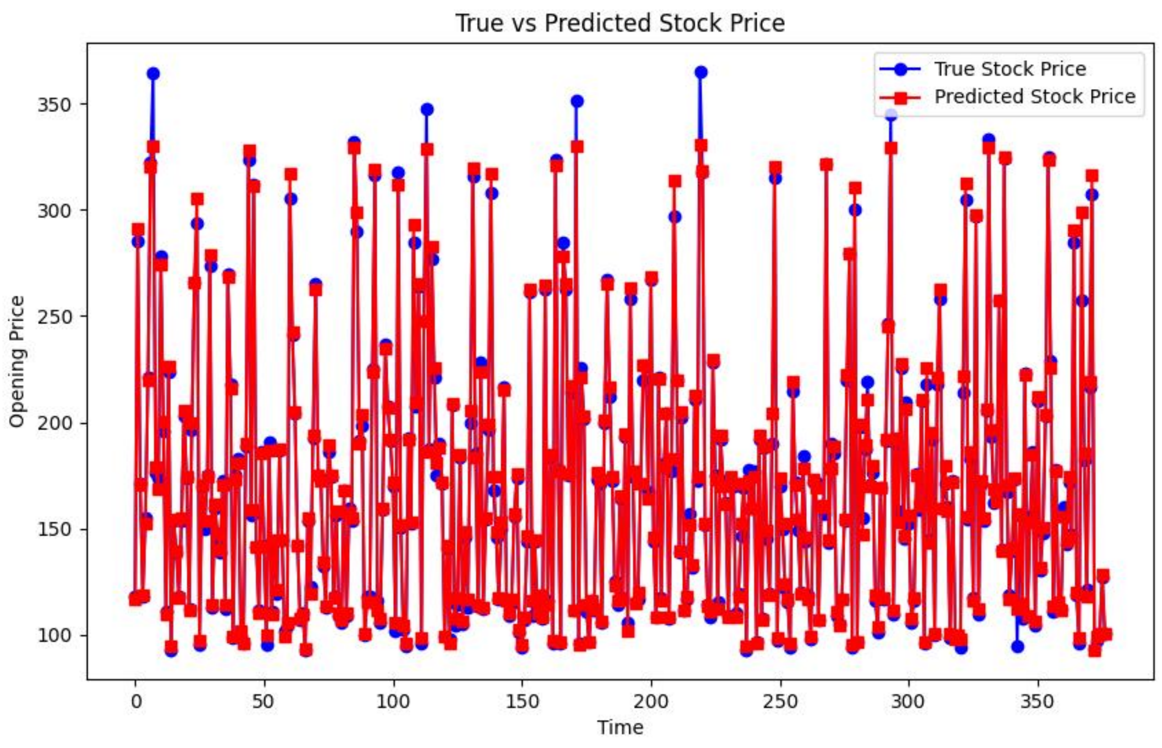


Comments

- As the training epochs increase, the loss steadily decreases. In the initial epochs, the loss drops rapidly, indicating that the model quickly learns from the data. After a certain number of epochs, the rate of reduction in loss slows down, and the model starts to converge. At this point, the loss reaches around 5000.

- However, the training process continues, and the loss gradually decreases further, eventually converging to a training loss of approximately 50 to 60 and a validation loss of about 15 to 30.
- The training and validation mean absolute error are around 5.7 and 3.2 approximately. This might be due to Adam optimizer which often leads to faster convergence and improved optimization performance especially for complex and high-dimensional datasets.

6. **Output from testing, including the final plot and your comment on it**



Comments:

After completing the training process, we loaded the saved model to make predictions on the testing set. The figure above illustrates the results of these predictions. The Mean Squared Error (MSE) loss on the testing dataset is approximately 24.6, while the Mean Absolute Error (MAE) is 2.7. The figure shows that the majority of the forecasts closely resemble the actual target values, indicating good predictive performance. However, there are a few predictions that deviate from the testing targets, suggesting some instances where the model's accuracy is not as close to the true values.

7. **What would happen if you used more days for features (feel free to actually try it – but do not upload the datasets).**

   When we used more days for features, it provided with the model with additional historical information about the stock prices and volumes. By incorporating more days as features, the model gained a deeper understanding of the historical trends and patterns in the stock prices and volume. As a result, the model made some accurate predictions, especially for the recurring patterns in the stock market data.

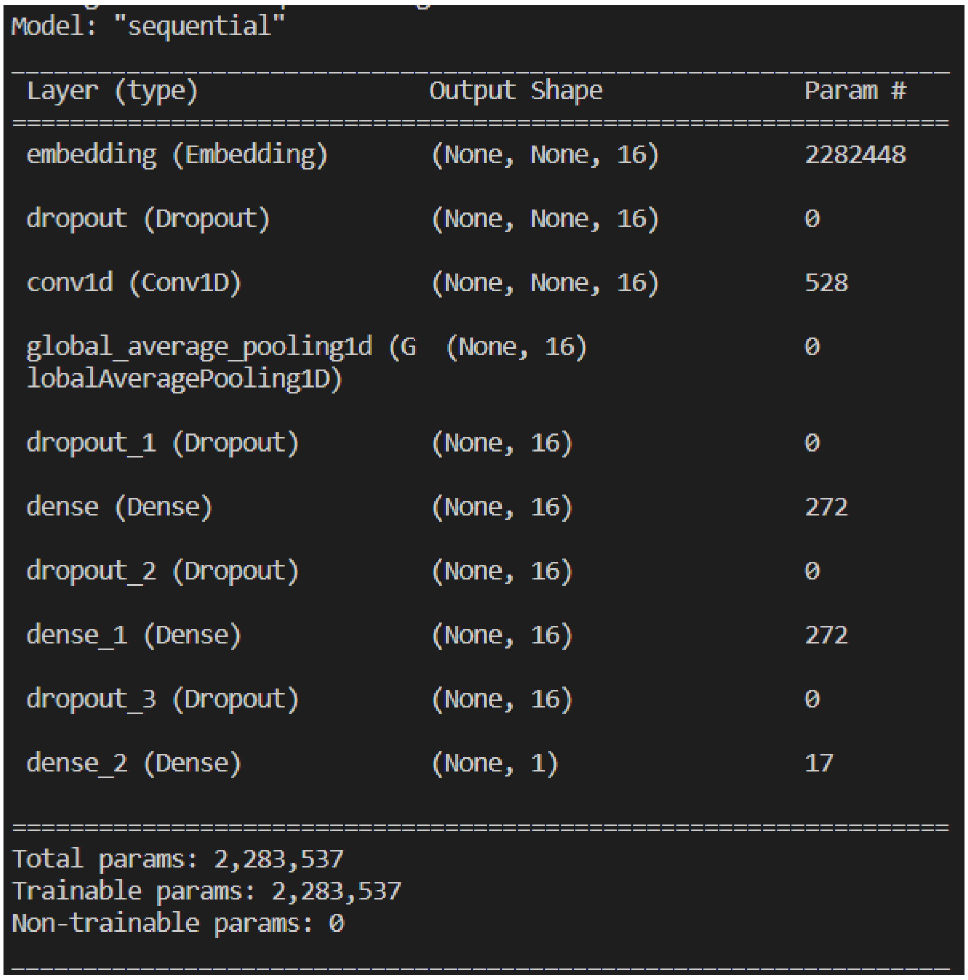
Problem 3: Sentiment Analysis

The IMDB large movie review dataset has many positive and negative reviews on movies. Download the dataset here (http://ai.stanford.edu/~amaas/data/sentiment/). Check the README file as it provides a description of the dataset. Use the provided training and testing data for your network. You would need to go through data pre-processing to prepare the data for NLP. Then, create a network to classify the review as positive or negative.

1. **Pre-processing Steps**
   - ***a. Loading data / Removing HTML Tags / Removing Special characters / Case Normalization:*** The function (load_data) loads text data and corresponding labels from files stored in subdirectories for each class ('neg' for negative reviews and 'pos' for positive reviews).
     It iterates over each label's subdirectory, reads the text files, and pre-processes the text by removing HTML tags and non-alphanumeric characters, converting the text to lowercase.
   - ***b. Tokenization*****:** The Tokenizer class is used to tokenize the text data into individual words and build a vocabulary based on the training data. It assigns a unique integer index to each word in the vocabulary. The text data from both the training and test sets are converted into sequences of integers using the vocabulary generated by the Tokenizer.
   - ***c. Padding:*** The purpose of padding is to ensure that all input sequences have the same length, which is a requirement for feeding data into many NLP models. Using this technique, we added extra tokens (usually with value 0) to the sequences to make them all the same length.
   - d. ***Label Encoder:*** Label Encoder class from sklearn.preprocessing is used to convert string labels (categories) into integer labels. This process is necessary for many machine learning algorithms, including neural networks, as they often require numerical values as inputs for target labels.

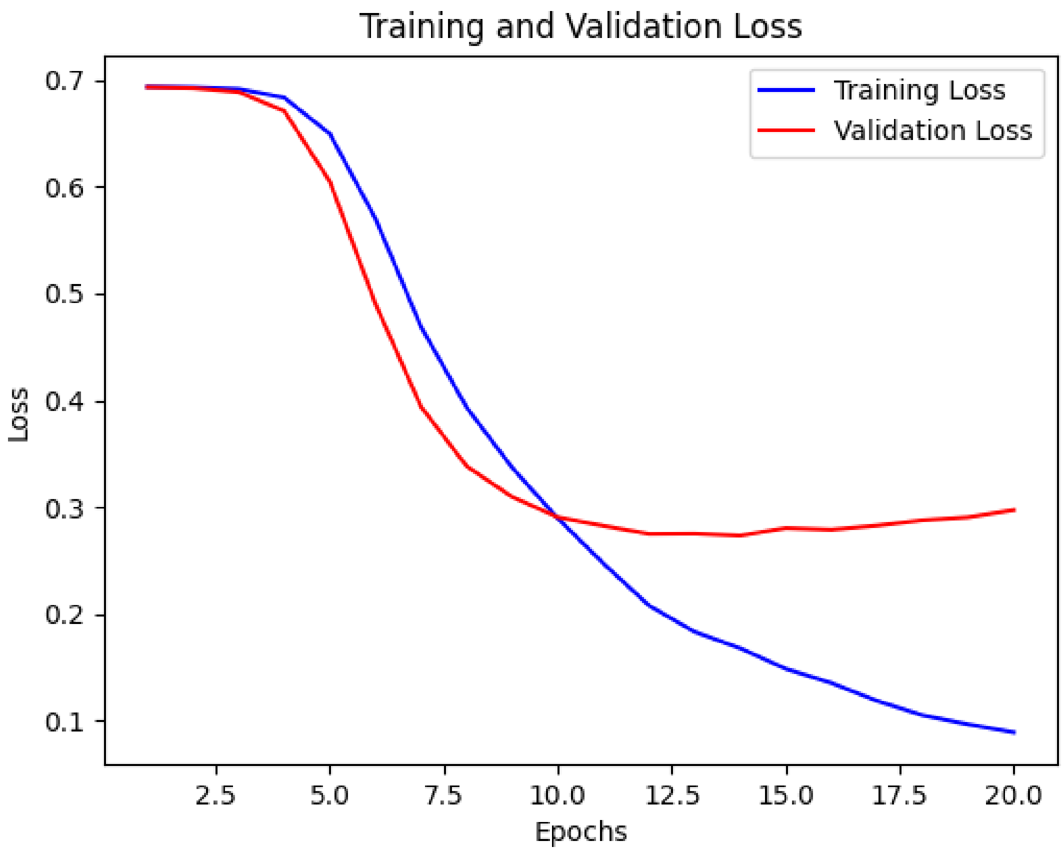
2. Building the Network

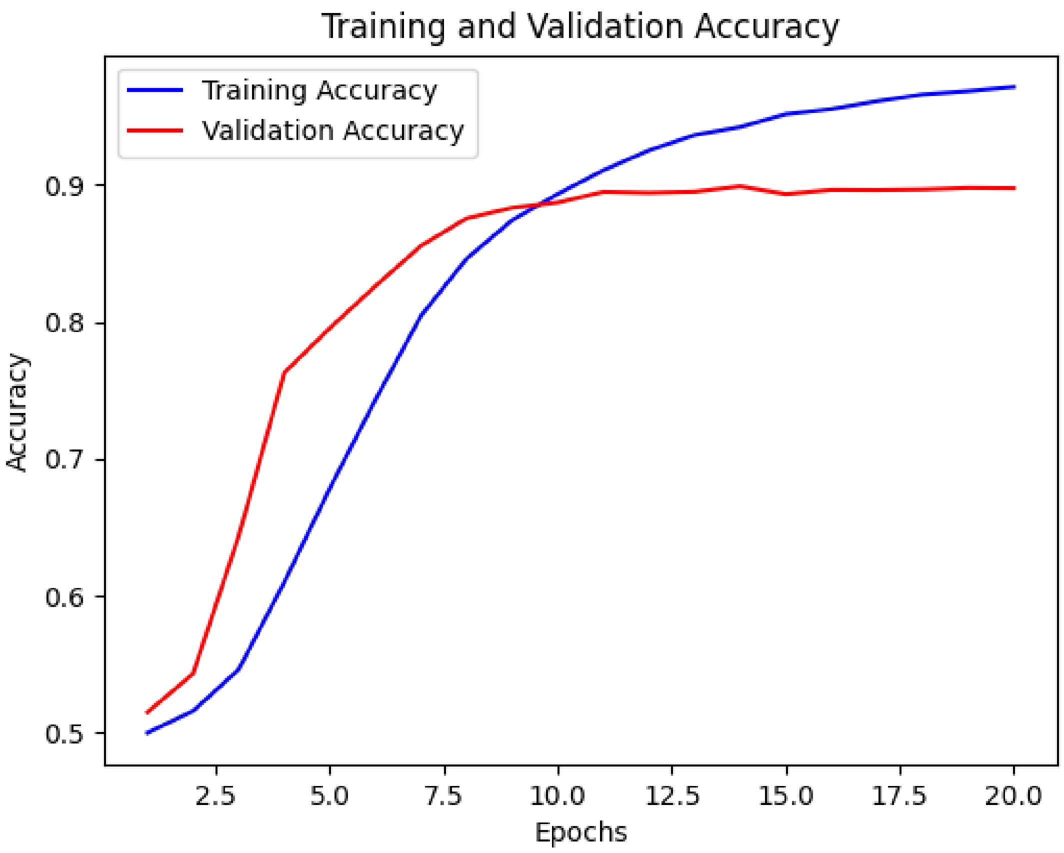
- The model starts with an Embedding layer, which converts integer-encoded vocabulary indices into dense vector representations. Each word in the vocabulary is represented by a 16-dimensional dense vector.
- To prevent overfitting and improve generalization, Dropout layers are introduced at various stages in the network.
- The Conv1D layer applies 16 filters of size 2, followed by the ReLU activation function, introducing non-linearity to the model
- The GlobalAveragePooling1D layer is added to reduce the spatial dimensions of the data and convert the 2D output from the Conv1D layer into a 1D vector.
- After pooling, two additional Dense layers with 16 units each are introduced. These fully connected layers help the model capture more high-level patterns and relationships in the data.
- Finally, a Dropout layer is applied before the last Dense layer, which has 1 unit with a sigmoid activation function. The sigmoid activation function transforms the output into a probability value between 0 and 1

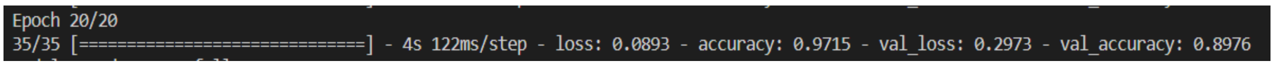


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 16)          2282448

 dropout (Dropout)           (None, None, 16)          0

 conv1d (Conv1D)             (None, None, 16)          528

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 16)                272

 dropout_2 (Dropout)         (None, 16)                0

 dense_1 (Dense)             (None, 16)                272

 dropout_3 (Dropout)         (None, 16)                0

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 2,283,537
Trainable params: 2,283,537
Non-trainable params: 0
_________________________________________________________________
```

3. Training and Validation Accuracy



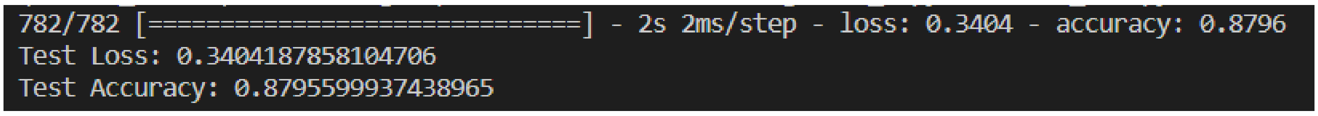




```
Epoch 20/20
35/35 [==============================] - 4s 122ms/step - loss: 0.0893 - accuracy: 0.9715 - val_loss: 0.2973 - val_accuracy: 0.8976
```

- The accuracy graph reveals a notable trend where the training accuracy consistently improves throughout all epochs, ultimately achieving a high accuracy of 97%.
- On the other hand, the validation accuracy shows a rising trend up to the 10th epoch, after which it stabilizes and remains constant.
- It concludes with a validation accuracy of approximately 89% after 20 epochs, suggesting that the model's performance plateaus after a certain point.
- As for the loss graph, it exhibits a pattern that mirrors the accuracy graph but in an inverse manner. The training loss steadily decreases with each epoch, indicating that the model becomes more adept at minimizing its error on the training data.
- Similarly, the validation loss decreases until the 10th epoch but then levels off, reflecting the model's ability to generalize well on unseen data up to a certain extent.

4. **Test Accuracy**



```
782/782 [==============================] - 2s 2ms/step - loss: 0.3404 - accuracy: 0.8796
Test Loss: 0.3404187858104706
Test Accuracy: 0.8795599937438965
```

We got a test accuracy of 87.95%. It indicates that the model has learned valuable insights from the data and it is capable of making reasonably accurate predictions.