Problem-1

1. Any preprocessing steps you made -

- We scaled the Image pixel values between 0 and 1.
- Since data is limited to 10K(20%) images from the whole training set. We also applied image augmentation to randomly sampled images from the training set.
- From the 10K images sampled from the initial training data, we produced 2 more images for each one of them with the configured augmentation(using ImageDataGenerator). The idea is to make neural the CNN generalize better with limited data and decrease overfitting to the limited patterns presented by inducing a regularization effect.
- After all preprocessing the shapes of the data used are -
    - Train data shape - X_train - (30000, 32, 32, 3) , y_train - (30000, 1)
    - Test data shape - X_test - (5000, 32, 32, 3) , y_test - (5000, 1)
    - Validation data shape - X_val - (5000, 32, 32, 3) , y_val - (5000, 1)

2. Description of the output layer used and the loss function (use 1 setting for all 3 networks) and why you made these choices -

- The output layer for all the 3 networks has 10 nodes. Each node represents the probability of the input belonging to that class. We used softmax as the suitable activation function for this multiclass classification.
- Loss function used is the 'Sparse-categorical-crossentropy' - Since the output only contains the actual class the image belongs to. 'Sparse-categorical-crossentropy' used while training automatically takes argmax on the output from softmax and produces the network predicted class to be compared against the actual class in the data.

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 -

- Keeping nodes as 512 and varying the number of hidden layers from 1 to 5, all examples executed for **5 epochs** -
    - Shallow networks show higher accuracy initially but fail to generalize because of their simple architecture and limited capacity to learn.
    - Networks with too many layers do not show much improvement and learning is negligibly slow because of the possibility of vanishing gradients.
    - Networks in-between show a relatively stable behavior with gradual learning and generalization of the problem.

Number of hidden layers	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss
1	29.81	34.52	1.92	1.81
3	26.91	30.52	1.97	1.90
5	20.43	22.72	2.07	2.03
7	10.15	10.16	2.30	2.30

- Keeping the number of hidden layers constant(5) and varying the number of nodes for **5 epochs -**
    - With increasing nodes, we do not see improved performance or learning.

Number of nodes in every layer	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss
32	19.34	21.74	2.09	2.04
128	20.37	20.66	2.07	2.06
512	20.41	22.16	2.07	2.03
1024	19.55	21.28	2.09	2.02

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

Model	**Train Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Train Loss**	**Validation loss**
MLP	29.07	33	34.58	1.93	1.86
CNN1	81.87	50.28	51.64	0.54	1.72
CNN2	48.51	55.74	56.40	1.41	1.21

- **MLP -** The MLP model has a simple architecture with fully connected layers, it exhibits slow learning and after a certain point it will not be able to learn more and generalize well for the testing data. So we expect a low accuracy, which is similar to what was observed. The MLP model struggles to capture spatial patterns in the image data compared to the CNN models.

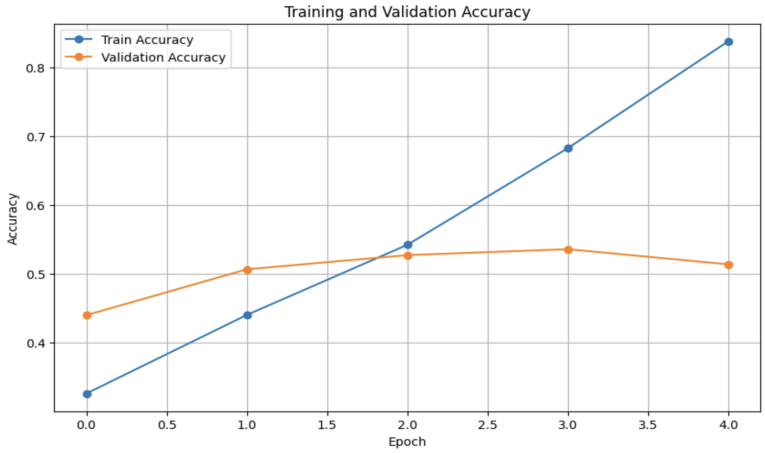
- **CNN1 -** CNN1 learns faster but overfits, but the architecture is complex it picks on minute details and tends to overfit.

- **CNN2 -** Adding dropouts has a regularizing effect which makes training faster but learning a bit slow. That is why after 5 epochs CNN2 has a lower accuracy compared to CNN1.

5. Plot the training and validation curves for the two CNNs and comment on the output. How does the training time compare for each of the CNNs? How does the different architectures influence these results? What do you expect the accuracies to be if the networks were trained for more epochs?

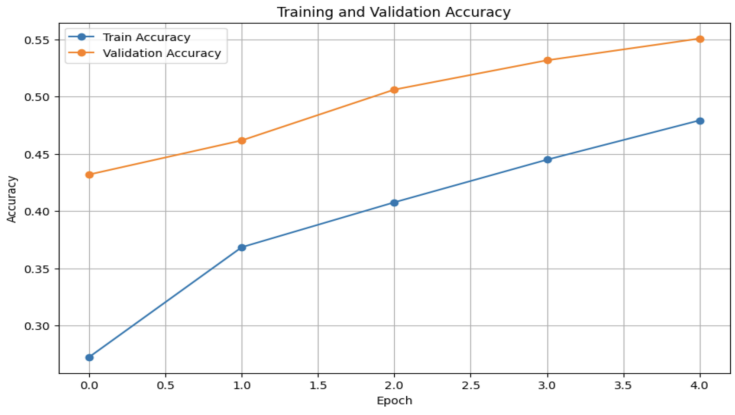
CNN1 -

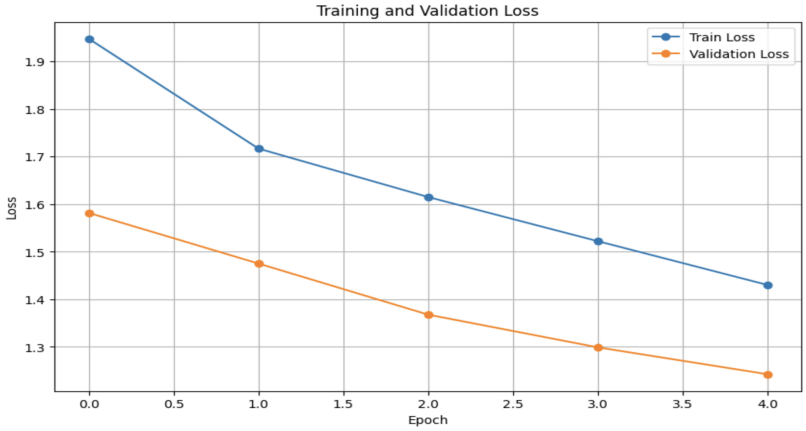
- We can see from the below graph for 5 epochs that our model overfits.
- Although we did preprocessing by augmenting input images, still model is not able to generalize well for the validation data.
- If the model is trained with more epochs there will not be much learning because the model will overfit on the training data and will not learn much to perform better when exposed to unseen data.



CNN2 -

- We saw improved learning due to the introduction of dropouts which have a regularization effect on our model.
- Also, we can see from the graph that model does not overfit(at least in the initial epochs)





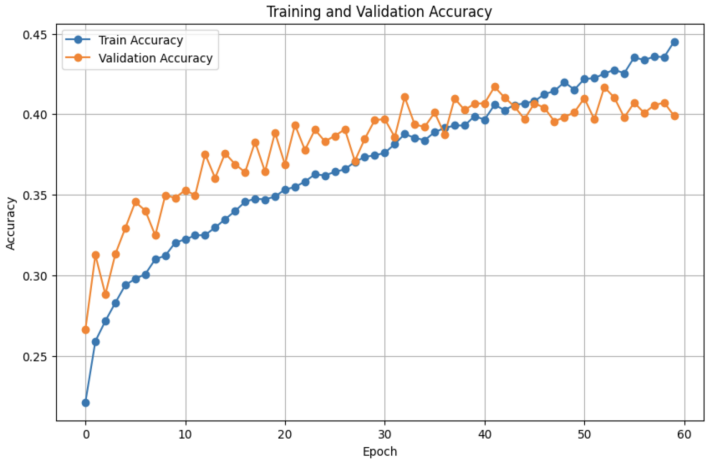
Execution time for 5 epochs -

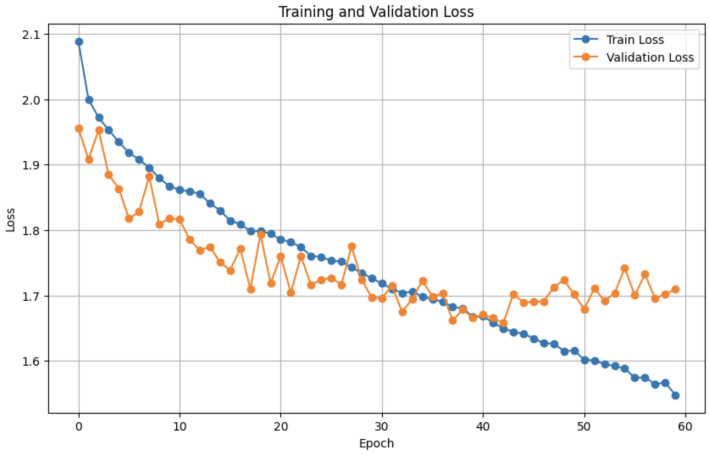
- **CNN1 ~** 83 seconds
- **CNN2 ~** 22 seconds

The reason CNN2 has a smaller execution time is that we have introduced dropouts between the 2 fully connected layers so which has a **thinning effect** on our network so the forward and backward passes have comparatively lesser nodes(that are randomly dropped out for an iteration) to update and our model trains in a short time compared to the architecture without any dropouts.

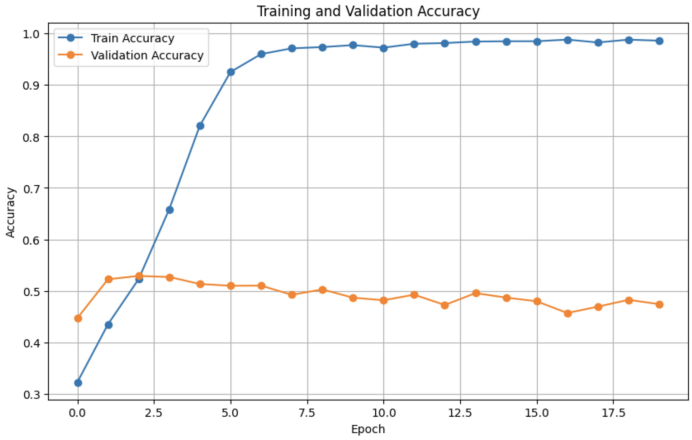
Expectation if networks are trained for more epochs -

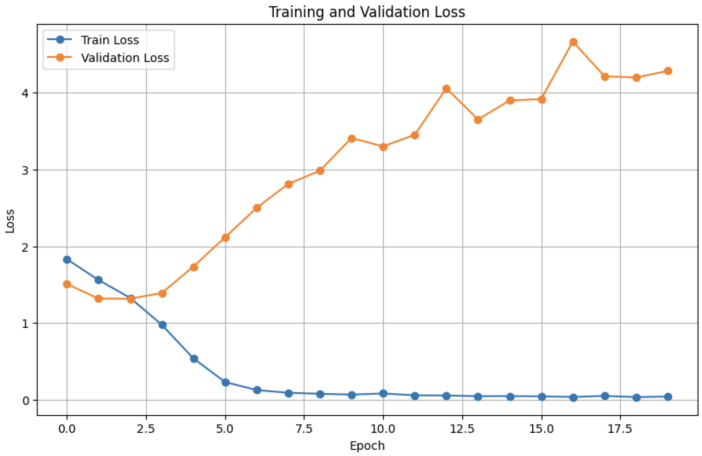
MLP - After training MLP for **60 epochs** we see that model still has low accuracy on validation data and shows no improvement after the **40th epoch** although training accuracy seems to be increasing. We can conclude the MLP with its architecture will not be able to learn the problem efficiently.



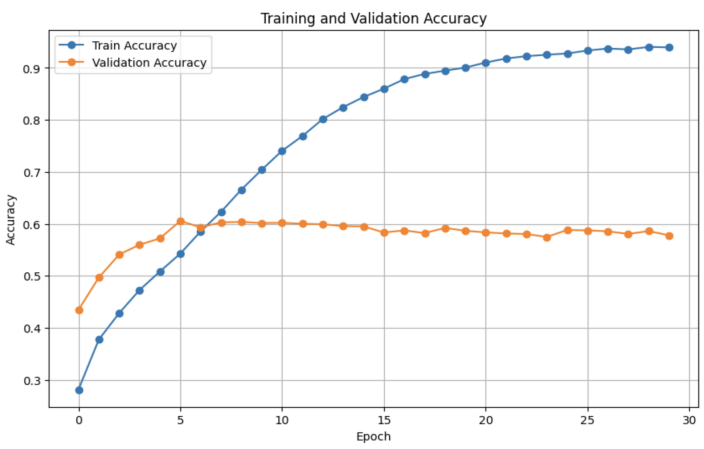


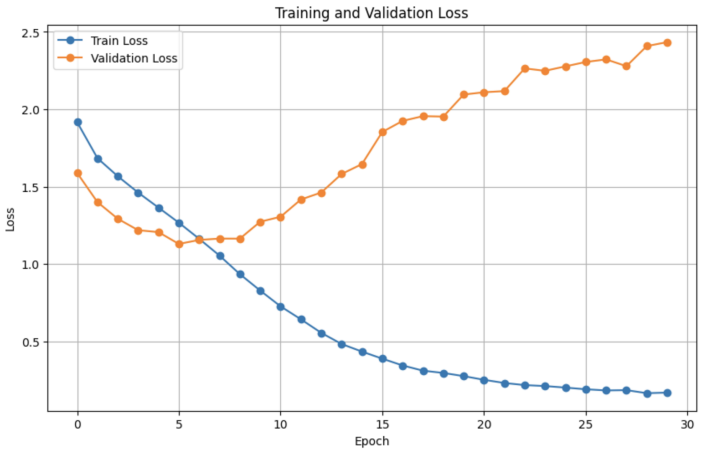
CNN1 - On running CNN1 for **20 epochs**, we can see that our **model overfits** and does show any improvement in learning after approximately the 3rd epoch.





CNN2 - On running CNN2 for **60 epochs**, due to the introduction of dropout in CNN2 we see some improvement **against overfitting** but after increasing epochs, we notice the same problem and the model overfits and shows no improvement in learning the problem.





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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 64)        1792

 conv2d_5 (Conv2D)           (None, 32, 32, 64)        36928

 batch_normalization_4 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 128)       73856

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       147584

 batch_normalization_5 (Batc  (None, 16, 16, 128)      512
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 dropout_3 (Dropout)         (None, 8, 8, 128)         0

 flatten_1 (Flatten)         (None, 8192)              0

 dense_3 (Dense)             (None, 64)                524352

 batch_normalization_6 (Batc  (None, 64)               256
 hNormalization)

 dropout_4 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 64)                4160

 batch_normalization_7 (Batc  (None, 64)               256
 hNormalization)

 dropout_5 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 790,602
Trainable params: 789,962
Non-trainable params: 640
_________________________________________________________________
```

Improved architecture -

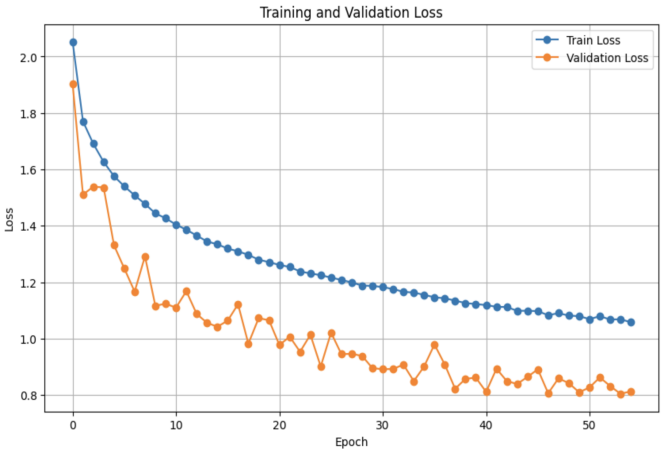
Adding more convolutional layers and increasing filter size for deeper layers - We see that CNN2 overfits when we increase the number of epochs. Adding convolutional layers can help the network learn more patterns and generalize better. Deeper layers work on learning more abstract features by combining features learned in initial layers, so increasing the filter count helps learn and generalize better on unseen data.

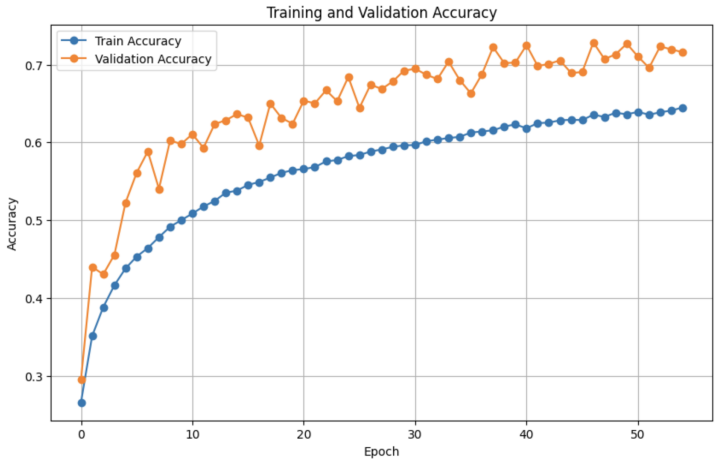
Increasing dropouts - Further increasing dropouts values introduces a higher regularization effect on the network against increasing the number of layers and nodes and also increases model efficiency.

Batch normalization - Also introduces a regularization effect on the network by improving performance and smoothening the learning process.

Increasing size of the training set - We notice that the training data used in is limited. On increasing training data to 50%(25K images) from 20% (10K images) we see an increase in accuracy and faster convergence on the same architecture.

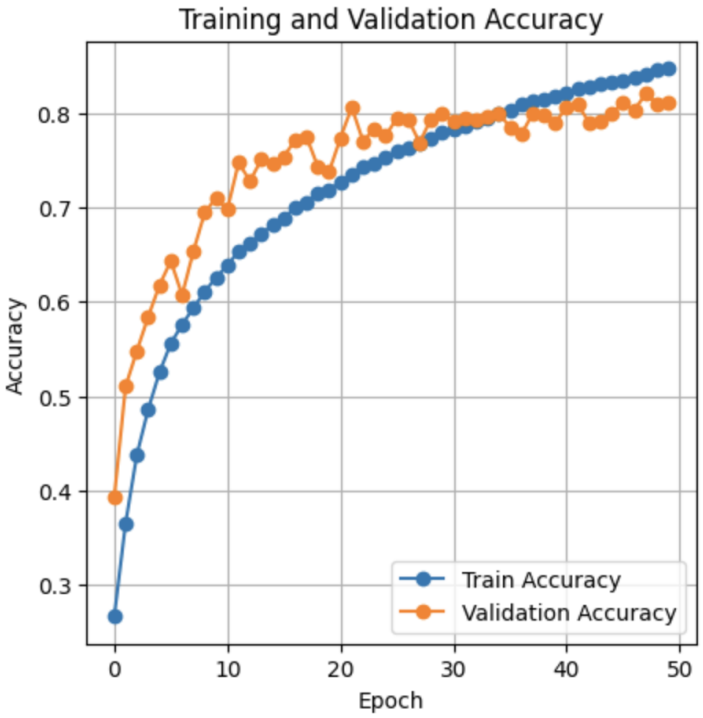
Running improved architecture using 20% of training data (10K randomly sampled images)- Test accuracy - 72.28%





Running improved architecture using 50% of training data(25K randomly sampled images) - Using the improved architecture we see an accuracy of ~ 80% on validation data, 10% increase compared to the same architecture when provided with 20%(10K images) of the training dataset. Also a 10% increase in test accuracy.

Test accuracy - 80.62%



PROBLEM 2

Explanation of how the dataset was created:

The dataset was created by using the latest 3 days' data as the features and the next day's opening price as the target. Each row in the dataset corresponds to one sample instance. The features for each sample consist of the opening prices for the three previous days, while the target is the opening price of the following day.

Any preprocessing steps followed:

Before training the neural network, some preprocessing steps were performed on the dataset:

1. Feature Scaling: Min-max scaling was applied to scale all the features and the target variable within the range of [0, 1]. This step is essential to ensure that all the values have a similar scale, which helps in faster convergence during training.

2. Reshaping for LSTM: LSTM networks require input in a 3D format [samples, time steps, features]. The features were reshaped to have a single time step for each sample.

All design steps to find the best network and final design:

The design steps include:
1. Initializing a sequential model.
2. Adding an LSTM layer with 50 units and a ReLU activation function. The input shape is set to (1, n), where n is the number of features.
3. Adding a Dense layer with 1 unit to output the predicted value.
4. Compiling the model using the Adam optimizer and mean squared error loss function.

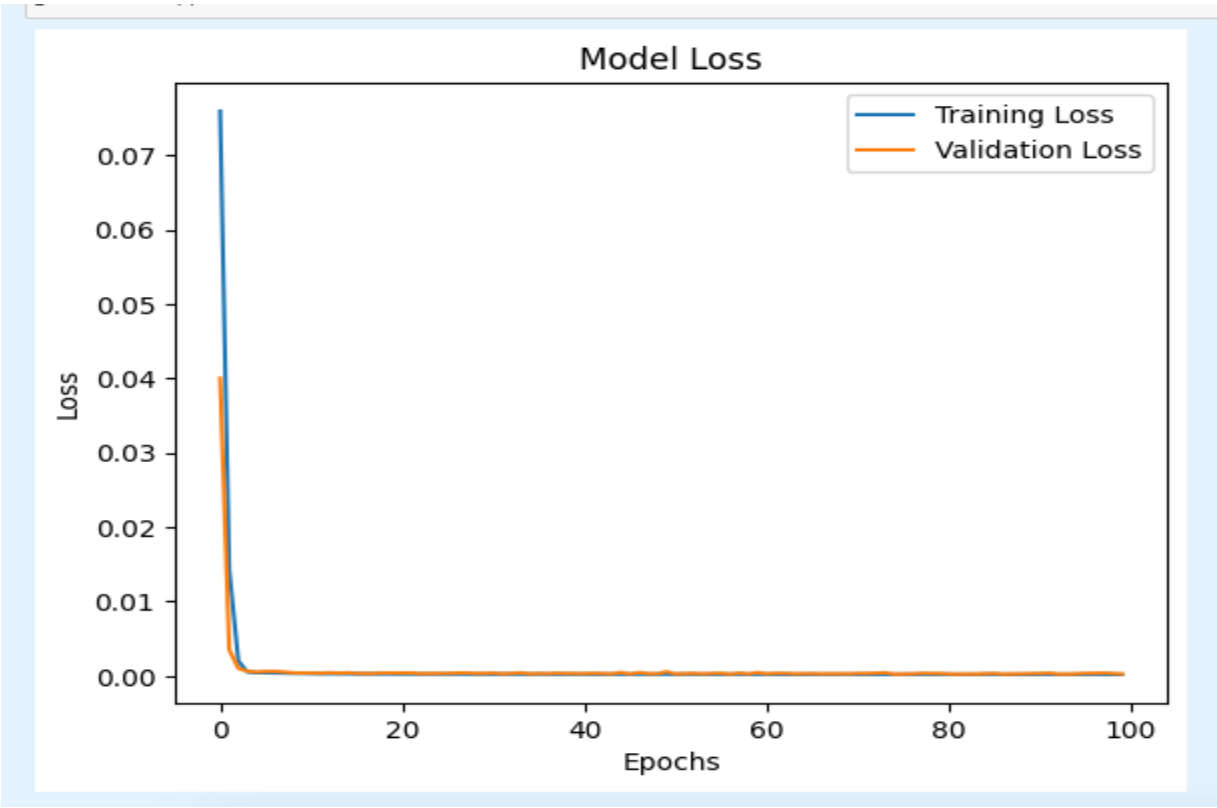
Architecture of the final network, number of epochs, batch size, loss function, training algorithm, etc.:

The final network architecture is as follows:
- LSTM layer with 50 units and ReLU activation.
- Dense layer with 1 unit.

Number of epochs: 100
Batch size: 32
Loss function: Mean squared error (MSE)
Optimizer: Adam

Output of the training loop with comments on your output:

The model is trained on the training data for 100 epochs using a batch size of 32. During training, the ModelCheckpoint callback is used to save the best model based on validation loss. The final training loss is printed at the end of the training loop.



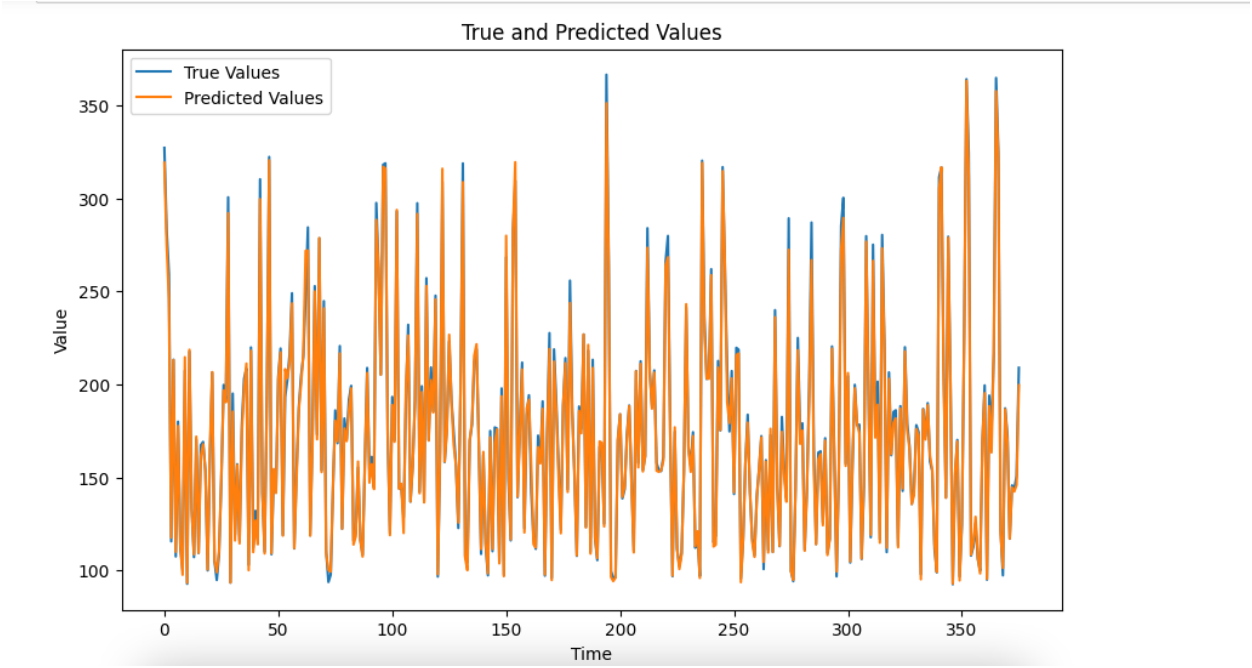
From the provided figure, it is evident that as the number of epochs increases, the loss gradually decreases. Initially, the decrease is rapid, and it almost converges at a training loss of approximately 25 and a validation loss of around 20.

The final training loss is impressively low at 0.0001937259512487799, which is likely achievable due to the use of the Adam optimizer. Unlike classical stochastic gradient descent, the Adam optimizer dynamically updates the learning rate during the training phase. This adaptive learning rate helps the model efficiently navigate the loss landscape and converge to a better solution.

In summary, the observed decrease in loss with increasing epochs, especially the low final training loss, can be attributed to the benefits of utilizing the Adam optimizer, which facilitates effective learning rate adjustments during the training process.

Output from testing, including the final plot and your comment on it:

The trained model is loaded, and predictions are made on the test data. The test loss is calculated and printed. A plot is generated to compare the true opening prices from the test data with the predicted opening prices from the model. The plot shows how well the model's predictions align with the true values.



Upon loading the saved model and making predictions, it is observed that the Mean Squared Error (MSE) test loss is measured at 0.00024488221970386803. Overall, the model seems to perform well, as most of the predicted values closely match the true values from the testing dataset.

However, it is worth noting that there are a few predictions that do not align closely with the actual testing values. These discrepancies might be attributed to various factors, such as noisy or challenging data points, model complexity, or potential overfitting of the training data.

While the majority of the predictions being accurate is a positive sign, the presence of some less accurate predictions indicates that there may still be room for improvement in the model's generalization capability.

Further analysis and fine-tuning of the model, or potentially exploring different architectures, hyperparameters, or regularization techniques, could help address the deviations between predictions and testing values and enhance the overall performance of the model.

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

If more days were used for features, the model would have access to additional historical data for each sample instance. This may lead to a more comprehensive representation of the data and potentially improve the model's ability to capture underlying patterns and trends.

However, using more days for features may also increase the complexity of the model and make it more susceptible to overfitting, especially if the dataset is limited. It would be important to strike a balance between the number of days used as features and the overall dataset size.

To try using more days for features, we would need to modify the "days" variable in the initial dataset creation code. For example, if we set "days = 5," the model would use the latest 5 days as features and the opening price of the 6th day as the target. We would need to experiment with different values of "days" and evaluate the model's performance on a validation set to find the optimal number of days to use for features.

Problem 3 – Sentiment Analysis

Data Preprocessing:

The dataset was downloaded as 'aclImdb' directory which consists of train and test directories separately. Using glob library, each of the text files were read and its data was stored into a list of all training and test reviews. The labels were stored in a separate list with 1's for positive class and 0 for negative class. After this, we performed a cleanup of the texts to remove all punctuations, html tags, extra whitespace and then converted the whole text to lowercase. Then the whole textual data was converted into words using regular expressions tokenizer from NLTK and stopwords were removed. We have applied lemmatization to convert words into their base forms using WordNetLemmatizer from NLTK library. Following that, we fit a text tokenizer from the Keras library onto the training set which first tokenizes the text and assigns a unique integer to each word in the vocabulary and then converts the text into a sequence of integers based on the learned vocabulary. The average length of all reviews at this point was 121 words and the maximum length was 1429. Since we need the input length to be the same for all reviews, the shorter sequences were padded with zeros at the end and longer sequences were trimmed to a maximum of 1000. After executing all the above mentioned steps, we split the set into 80% training set and 20% validation set.

Network Architecture:

The following Figure 1 shows the network design chosen, which is a sequential CNN model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1000, 16)          2131520

 dropout (Dropout)           (None, 1000, 16)          0

 conv1d (Conv1D)             (None, 999, 16)           528

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 32)                544

 dropout_2 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 2,132,625
Trainable params: 2,132,625
Non-trainable params: 0
```

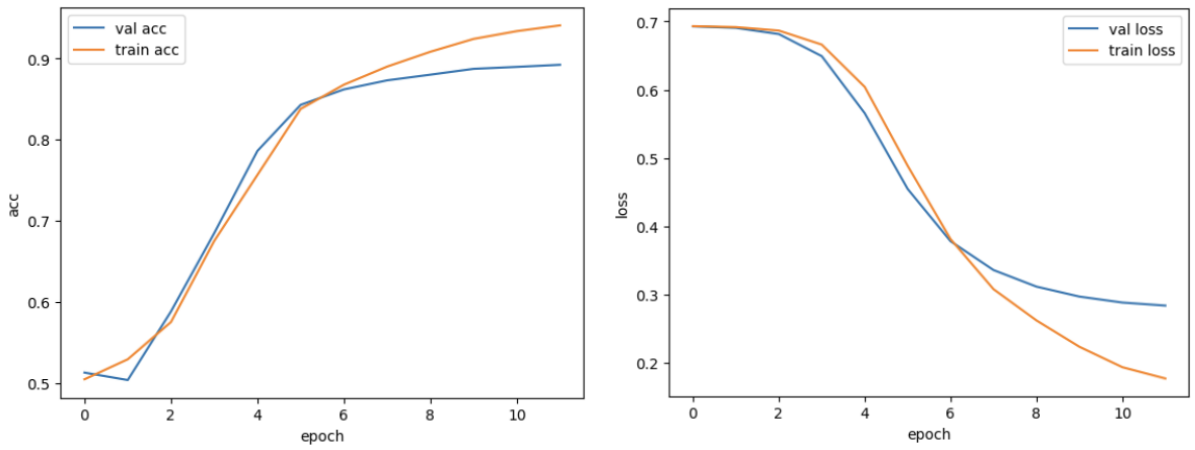
Figure 1: CNN structure

First layer is an Embedding layer which is used to convert integer sequences into vectors of fixed size. After that, there is a one-dimensional convolutional layer with 16 filters, kernel size of 2, no padding and ReLU activation function. Average pooling layer of one dimension has been added that reduces spatial dimensions of data. Dense adds a fully connected layer with 32 units and tanh activation function. The final layer has 1 unit and sigmoid activation function since this is a binary classification problem. Dropout layers have been added in between operations to regularize the model and avoid overfitting.

Training and Testing Accuracies:

```
Epoch 1/12
40/40 [==============================] - 5s 115ms/step - loss: 0.6930 - acc: 0.5044 - val_loss: 0.6927 - val_acc: 0.5126
Epoch 2/12
40/40 [==============================] - 4s 109ms/step - loss: 0.6917 - acc: 0.5290 - val_loss: 0.6905 - val_acc: 0.5034
Epoch 3/12
40/40 [==============================] - 5s 116ms/step - loss: 0.6865 - acc: 0.5749 - val_loss: 0.6814 - val_acc: 0.5880
Epoch 4/12
40/40 [==============================] - 5s 115ms/step - loss: 0.6660 - acc: 0.6749 - val_loss: 0.6491 - val_acc: 0.6846
Epoch 5/12
40/40 [==============================] - 5s 114ms/step - loss: 0.6041 - acc: 0.7564 - val_loss: 0.5655 - val_acc: 0.7860
Epoch 6/12
40/40 [==============================] - 5s 113ms/step - loss: 0.4886 - acc: 0.8378 - val_loss: 0.4547 - val_acc: 0.8428
Epoch 7/12
40/40 [==============================] - 5s 116ms/step - loss: 0.3812 - acc: 0.8676 - val_loss: 0.3781 - val_acc: 0.8618
Epoch 8/12
40/40 [==============================] - 5s 116ms/step - loss: 0.3079 - acc: 0.8899 - val_loss: 0.3359 - val_acc: 0.8730
Epoch 9/12
40/40 [==============================] - 5s 113ms/step - loss: 0.2622 - acc: 0.9082 - val_loss: 0.3117 - val_acc: 0.8800
Epoch 10/12
40/40 [==============================] - 5s 113ms/step - loss: 0.2236 - acc: 0.9241 - val_loss: 0.2971 - val_acc: 0.8872
Epoch 11/12
40/40 [==============================] - 5s 114ms/step - loss: 0.1937 - acc: 0.9337 - val_loss: 0.2884 - val_acc: 0.8896
Epoch 12/12
40/40 [==============================] - 5s 113ms/step - loss: 0.1772 - acc: 0.9408 - val_loss: 0.2839 - val_acc: 0.8922
```

Figure 2: Results per epoch



(a) (b)

Figure 3: (a) Training vs Validation set accuracy comparison, (b) Training vs validation set loss comparison

After training the model for 25 epochs, it was noticed that the model was overfitting beyond 10-12 epochs and thus, the final model has been trained over 12 epochs with batch size of 512. As per the figures above, the training loss is 0.1772, validation loss is 0.2839, training accuracy reaches 94.08% and validation set accuracy reaches 89.22%. The test set accuracy is 88.136%.