1 CIFAR10

1.1 Preprocessing

Firstly, downloading and loading *CIFAR10* dataset from *Keras* library and using the *train_test_split* function to randomly sample 20 percent of the training set as the new training set in order to save computation.
Secondly, divide the RGB images by 255 to normalize the input vectors.
Thirdly, using a 0.5 ratio to split the testing set into validation and testing.

1.2 Output Layer and Loss Function

The activation function used in the output layer is **softmax** which inputs an N-dimensional vector of real numbers and outputs a probability distribution. And totally 10 neurons are used in the output layer, one represents each category in this dataset. The loss function used in this question is **categorical_crossentropy** as there are 10 classes in this dataset and the labels are in *one_hot* representation.

1.3 MLP

Fig.1 shows the training and validation accuracy for the MLP model with two fully connected layers with 512 units and a sigmoid activation function and an output layer.
The training accuracy after 5 epochs is 35.77% and the validation accuracy is 35.16%. Both of them show an upward trend observed from fig.1 meaning that the result is under-fitting as the training epoch and number of training samples are small. And the testing accuracy provided by the MLP model is 36.52%.

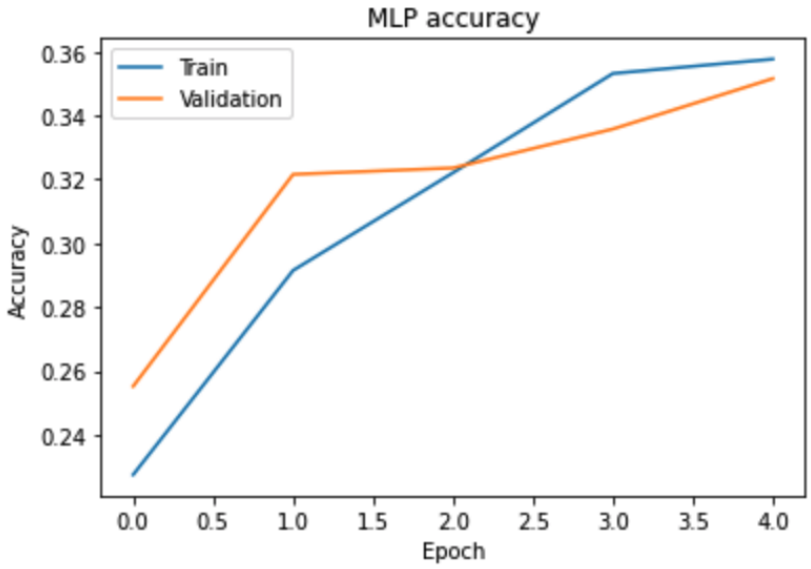

Figure 1: Training and validation accuracy for MLP model

Fig.2 and fig.3 show the training and validation accuracy for the MLP models after changing the number of layers and neurons per layer, where the MLP-v1 model has three fully connected layers with 512, 512, 256 units respectively and MLP-v2 model has only one fully connected layer with only 10 neurons.

The training and validation accuracy for the three fully connected layer MLP model after 5 epochs are 33.26% and 34.44%, and the testing accuracy for this model is 34.06%. The training and validation accuracy for the one fully connected layer MLP model after 5 epochs are 31.60% and 29.98%, and the testing accuracy for this model is 31.26%.

Table 1 summarizes the accuracy of the three models mentioned above. The accuracies for all the three MLP models are very low due to the small number of training epochs and the small number of training samples. And sometimes the validation accuracy and testing accuracy are higher than training accuracy, this is may because the number of

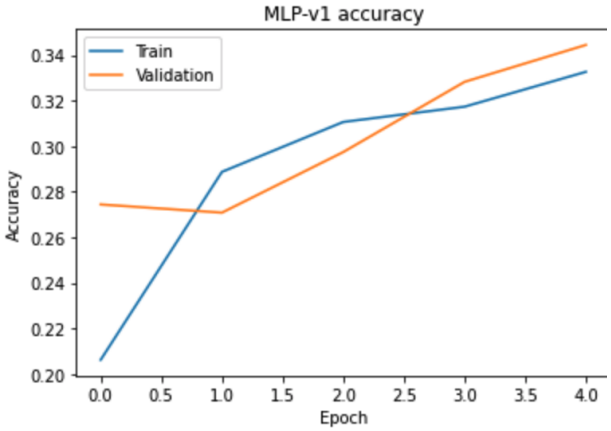

Figure 2: Training and validation accuracy for version1 MLP model

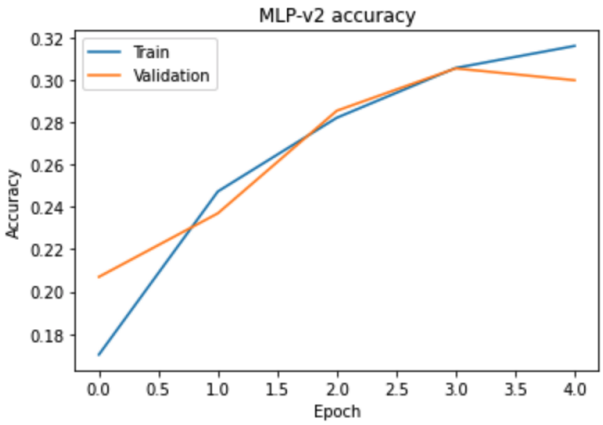

Figure 3: Training and validation accuracy for version2 MLP model

training samples (10000) is insufficient compared with the validation set (5000) and testing set (5000), and its samples' distribution affected the results.

The reason that the three-fully-connected-layer MLP-v1 model's accuracy is lower than the original two-fully-connected-layer model is that the architecture of the network is more complicated and with the same number of training epochs, its result is more under-fitting than the simple model. And the reason that the one-fully-connected-layer MLP-v2 model's accuracy is lower is that the small number of neurons (only 10) cannot perform the classification task very well.

Netword	Training Accuracy	Validation Accuracy	Testing Accuracy
MLP	35.77%	35.16%	36.53%
MLP-v1	33.26%	34.44%	34.06%
MLP-v2	31.06%	29.98%	31.26%

Table 1: Accuracy for three MLP models

1.4 Train and Test Accuracy for MLP and CNNs

Table 2 shows the train and test accuracy for all the three networks. It can be observed that both of the two CNN models perform better than the MLP model based on the testing accuracy. This result is as expected, as now Convolutional Neural Network is mainly used in computer vision.

Netword	Training Accuracy	Validation Accuracy	Testing Accuracy
MLP	35.77%	35.16%	36.53%
CNN1	94.25%	54.38%	55.40%
CNN2	61.78%	57.24%	57.68%

Table 2: Accuracy for MLP and CNNs models

In general, the number of weight parameters in the MLP model would be very large as the hidden layer is fully-connected which would cause inefficiency and requires higher computational and memory resources. Besides, the MLP model inputs flatten vectors ignoring the spatial information in images. Both of these characteristics leading to lower accuracy.

The panning of the filters in the Convolutional Neural Network allows it to share weight parameters and find specific patterns from every corner of the input image. Usually, different layers have different tasks to find different patterns from simple to complicate. The use of filters works well for object detection since it allows the network to find patterns in more than one place in the input image.

The training accuracy (94.25%) for the first CNN model is much higher than the second one. But its validation accuracy and testing accuracy are only 54.38 % and 55.4%. The high variance means this model is overfitting as it doesn't contain any pooling layer and dropout layer to avoid overfitting.

The second CNN model holds the best performance among the three models, which provides a 61.78% training accuracy and 57.68% testing accuracy.

1.5 Training and Validation curves for CNNs

Fig.4 and fig.5 show the training and validation curves for the two CNN models.

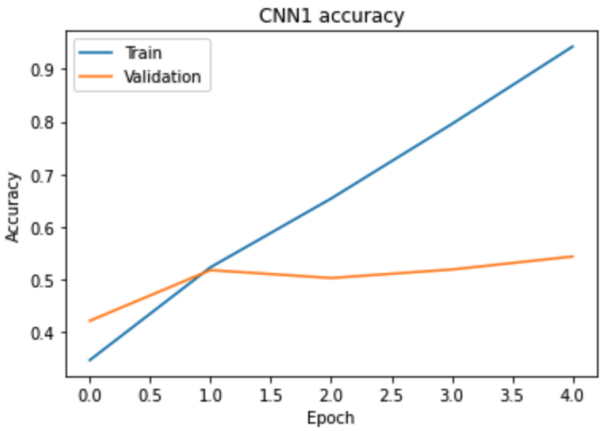

Figure 4: Training and validation accuracy for CNN1 model

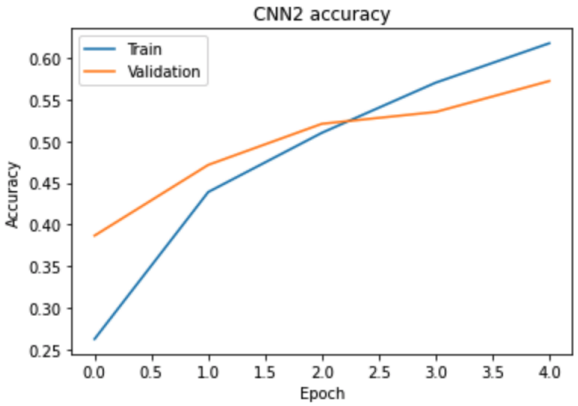

Figure 5: Training and validation accuracy for CNN2 model

It takes around 300 seconds to finish one epoch in the first CNN model, whereas it only takes about 50 seconds to complete one epoch in the second CNN model. This is mainly because the added Maxpooling layers in the second CNN model which reduce the dimensions of the data, thus reduce the amount of the weight parameters and lower the computation.

As already mentioned in the subsection 1.4, the first CNN model is overfitting after 5 epochs, if the networks were trained for more epochs, the results of it would not be better. But as the added pooling layers and dropout layers in the second CNN model help it avoid overfitting, the loss is still decreasing after 5 epochs. Thus, training more epochs (within a rational range)would let the model has higher accuracy (both training and testing). But if it was trained too long, it will also become overfitting.

1.6 Improve Recommendations

For the MLP model, training longer and bigger would help it reduce the avoidable bias as it is still underfitting after 5 epochs. But its characteristics decided that its accuracy would not be very high. For higher accuracy, a better neuron network architecture is required (for example, CNN).

For the first CNN model, several methods can be used to avoid overfitting, thus improve the network. For example, using more training data. Or adding pooling layers. Or implementing regularization, which including add L2 regularization to the Conv2D layer or Dense layer, or add dropout layer. Besides, early stopping can also help to avoid overfitting, which means training fewer epochs.

For the second CNN model, training slightly more epochs would help it perform better. Additionally, using more training samples, tuning the hyperparameters, changing optimizer, and loss function, all might help improve the network.

2 Problem 2

2.1 Dataset

Firstly, use *read_csv* function from *pandas* to load the original dataset, then use the *.iloc* method to select features and labels for the dataset, where the data from the columns **Open**, **High**, **Low**, and **Volume** are used as features and the data from the column "Open" is labels. As the data from each column varies a lot, especially the **Volume**, it needs to be normalized to help the model converge quicker to find the local or global

minimum. *MinMaxScaler* imported from *sklearn* is used to transform each feature individually into the same range. After normalization, two empty lists, features and labels, are created to store values, and a for-loop is conducted to generate samples using a 3 window size. Each sample contains three rows (12 values in total) as features and uses the previous row's **Open** value as its label. Finally, convert the list features and list labels into NumPy array to feed into the LSTM network.

In the testing set, almost the same preprocessing steps are done on the *test_data_RNN.csv* file, except the MinMaxScaler is applied to the training set first to fit the training data, then the scaler is applied on the testing data.

2.2 Model

In the final model, the first layer is an LSTM layer with 100 units. The second layer is a Dense layer with 20 units using *Relu* as activation function to improve performance, and the output layer is a normal Dense layer with one unit using *Linear* activation as this is a regression task, not a classification task, just needs to output the prediction values.

Mean_absolute_error is used to calculate the loss in each epoch, and the optimizer used in this model is *Adam*. Totally 50 epochs are run to obtain the final model.

In the design procedure, only 1 LSTM layer with 32 units was used at first, following the idea that starting with a simpler model. As the model cannot achieve low loss after running many epochs, a Dense layer using *Relu* was added to improve the model's performance and the number of units was increased to 100 in the LSTM layer. At the same time, to avoid overfitting, a Dropout layer with a dropout ratio of 0.1 was added between the two layers mentioned above. But by observing the final loss in both train-

ing and testing, this model actually didn't occur overfitting with the training data. So, the Dropout later was removed in the end, and the final loss dropped to 3 from 6.

2.3 Results

The training loss after 50 epochs is around 3.2, which is 63 at the beginning (first epoch). Actually the loss already reaches around 3.5 at the 20 epoch and dropping extremely slowly after that. To achieve higher accuracy, the architecture of the model need to be tuned instead of running more epochs.
The loss of testing data is 3.9 using the stored model. This value is expected, meaning that this model hasn't overfitted the training set. And fig.6 shows the plot of the true and predicted value, in which the predicted values follow the general trend but differences do exist in each sample.

2.4 Using More Features

Using the same model, only change the window size to 10 when generating data samples, that is to say, using ten days of data as one sample's features. The running time would increase as more features are used, but at the same time, the loss might drop under the same epoch.

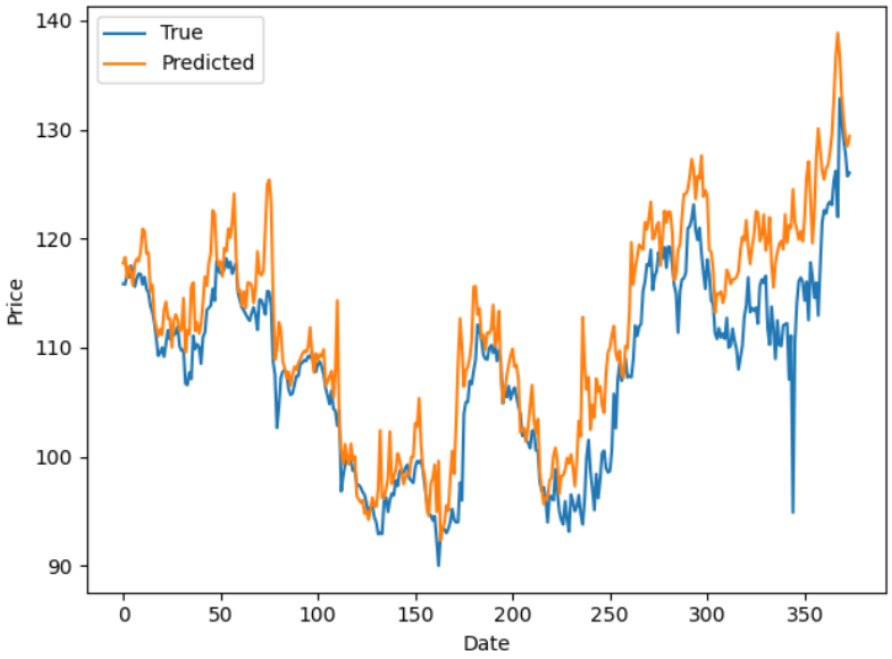

Figure 6: True and predicted values

3 Problem 3

3.1 Preprocessing

In the *train_NLP.py* file, firstly load the original dataset from *./data/aclImdb* folder and create two empty lists to store positive samples and negative samples as they come from different files. Then use the *gen_set* function to generate each sample from */train/pos* and */train/neg* respectively and put them into the corresponding positive list or negative list (using *.append()*) method. Combine the positive list and negative list to form the original training set. After having the original training set, the training labels are created by *np.ones* and *np.zeros*, in which positive samples obtained labels 1 and negative samples obtained labels 0. 20000 was chosen as the vocabulary size parameter, meaning that only the first 20000 words with the highest frequency would be retained. And 100 is the max length size parameter, that is to say, every sample would have the

same length, 100 words. Then, conducting text vectorization, using *texts_to_sequences* method from *Tokenizer* to transfer the text into numbers. And as some sentences are shorter than 100 words themselves or some sentences contain less than 100 words after filter the low-frequency words, the padding step is conducted to the obtained sequences. In the *test_NLP.py* file, the same preprocessing steps are implemented to the testing set. In the text vectorization procedure, as testing samples need to be fit in the same tokenizer as the training samples, the tokenizer is generated again in *test_NLP.py* file.

3.2 Model

The first layer is an *Embedding* layer using 32 as the word vector dimension parameter. *Embedding* layer provides the same effect as a fully connected layer who has conducted one-hot encoding to the input data, thus improving training efficiency. As the output of the *Embedding* layer is a 2D vector, in which each word in the input sequence of words has one embedding. A *Flatten* layer is added after the *Embedding* layer to flatten the 2D matrix into a 1D vector to feed into the next *Dense* layer. After the *Embedding* and *Flatten* layers, a *Dense* layer with 256 units using *Relu* activation function is added. And finally, the output layer with 1 units using *Sigmoid* function is added to the NLP model to complete the binary classification task.

Totally 4 epochs are run on training set, with a 512 batch size using *adam* optimizor and *binary_crossentropy* loss function. And the model is saved as 20862738_NLP_model.h5.

3.3 Accuracy

The final training accuracy reaches 98.7% after 4 epochs which seems kind of overfitting, but the testing accuracy is around 84% which is not very bad.