ECE 657 Assignment3 Report

Problem 1

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.**

Normalized the pixel values to be between 0 and 1 in the dataset. Scaled data take less time for fitting the model for training that of unscaled data. Given a large dataset, scaling can be very useful.

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

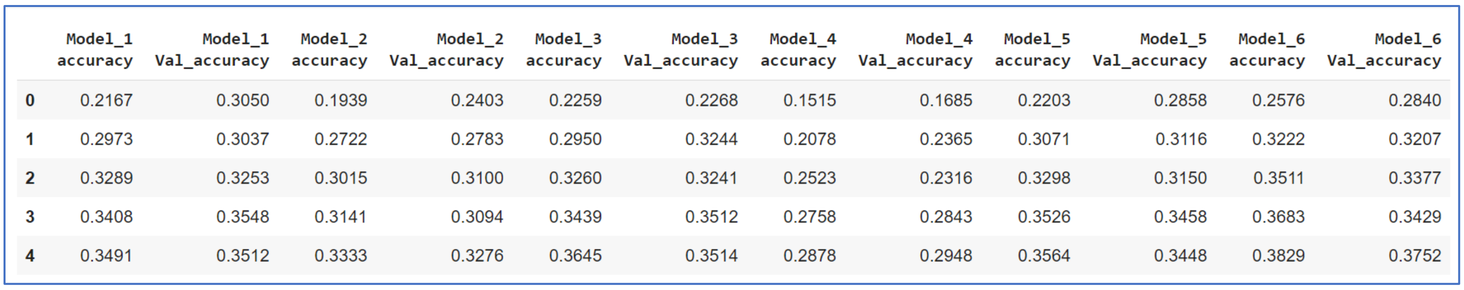
softmax is implemented at the output layer as an activation function. The softmax layer must have the same number of nodes as the output layer. softmax takes the concept of logistic regression and applies it to a multi-class environment. In a multi-class problem, it assigns decimal probability to each class. Since SoftMax is heavily used in the output layer of multiclass classification, it has been used in the output layer in this problem.

We utilize "categorical cross entropy" for the loss function, as it is used in multiclass classification. The categorical cross entropy is well suited to classification tasks. It quantifies the performance of a classification model with a probability value between 0 and 1.

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.**

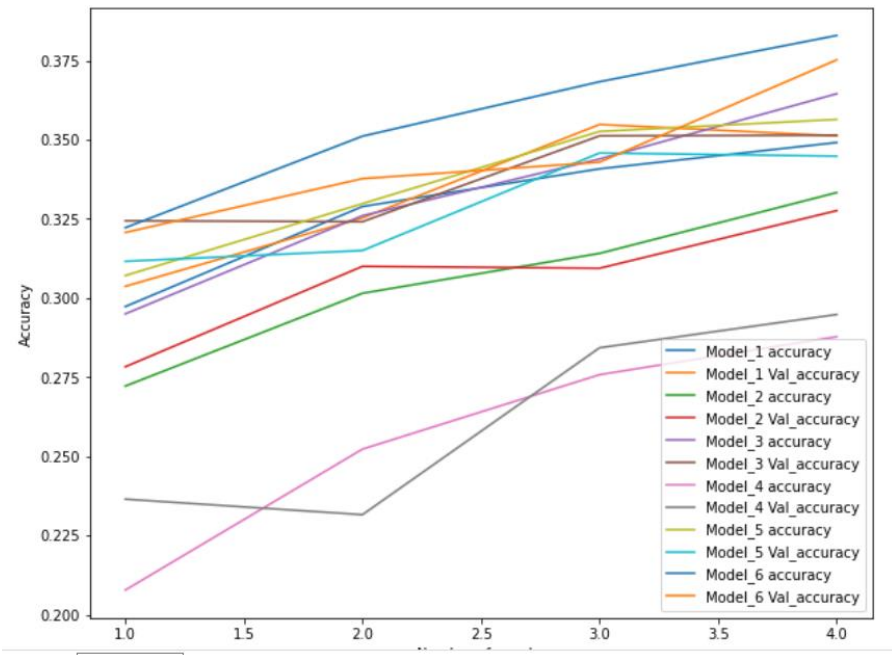
Here, I have created 6 MLP models by changing the number of layers and neurons and considered the given MLP as reference. Here the reference MLP layer has 2 hidden layers, and each hidden layer has 512 neurons.

1. MLP Model_1 has 1 hidden layer each hidden layer has 512 neurons. Trainable params: 1,841,162
2. MLP Model_2 has 3 hidden layer each hidden layer has 512 neurons. Trainable params: 2,103,818
3. MLP Model_3 has 2 hidden layer each hidden layer has 1024 neurons. Trainable params: 4,206,602
4. MLP Model_5 has 3 hidden layer each hidden layer has 1024 neurons. Trainable params: 5,256,202
5. MLP Model_3 has 2 hidden layer each hidden layer has 256 neurons. Trainable params: 855,050
6. MLP Model_6 has 1 hidden layer each hidden layer has 256 neurons. Trainable params: 789,258



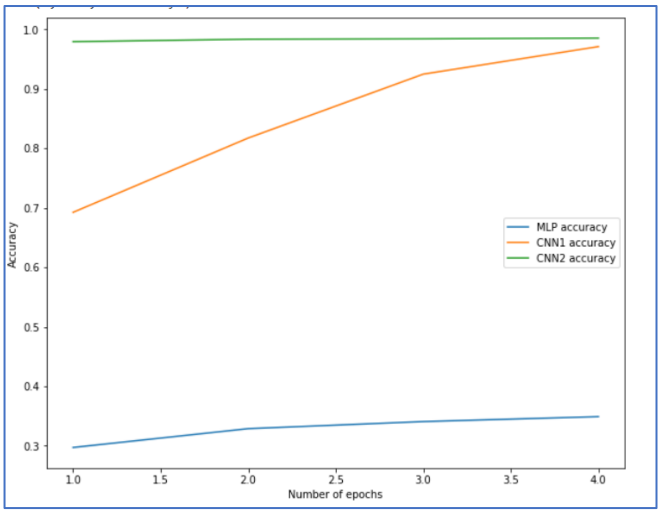
	Model_1 accuracy	Model_1 Val_accuracy	Model_2 accuracy	Model_2 Val_accuracy	Model_3 accuracy	Model_3 Val_accuracy	Model_4 accuracy	Model_4 Val_accuracy	Model_5 accuracy	Model_5 Val_accuracy	Model_6 accuracy	Model_6 Val_accuracy
0	0.2167	0.3050	0.1939	0.2403	0.2259	0.2268	0.1515	0.1685	0.2203	0.2858	0.2576	0.2840
1	0.2973	0.3037	0.2722	0.2783	0.2950	0.3244	0.2078	0.2365	0.3071	0.3116	0.3222	0.3207
2	0.3289	0.3253	0.3015	0.3100	0.3260	0.3241	0.2523	0.2316	0.3298	0.3150	0.3511	0.3377
3	0.3408	0.3548	0.3141	0.3094	0.3439	0.3512	0.2758	0.2843	0.3526	0.3458	0.3683	0.3429
4	0.3491	0.3512	0.3333	0.3276	0.3645	0.3514	0.2878	0.2948	0.3564	0.3448	0.3829	0.3752

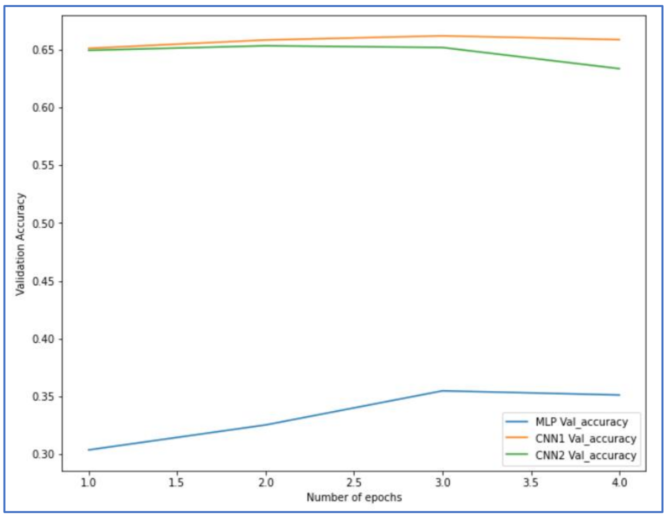
```
Text(0, 0.5, 'Accuracy')
```



From the accuracy plot, we can see that MLP Model_6 has the highest Training accuracy of 38.29%, and the model has lowest number of hidden layers and neurons. MLP Model_5 has the lowest Training accuracy of 29.48%, but it has the highest number of hidden layers and neurons. The above thing is also true for validation accuracy. From the graph, we can see that lightweight models in terms of both number of layers and neurons give good training and validation accuracy.

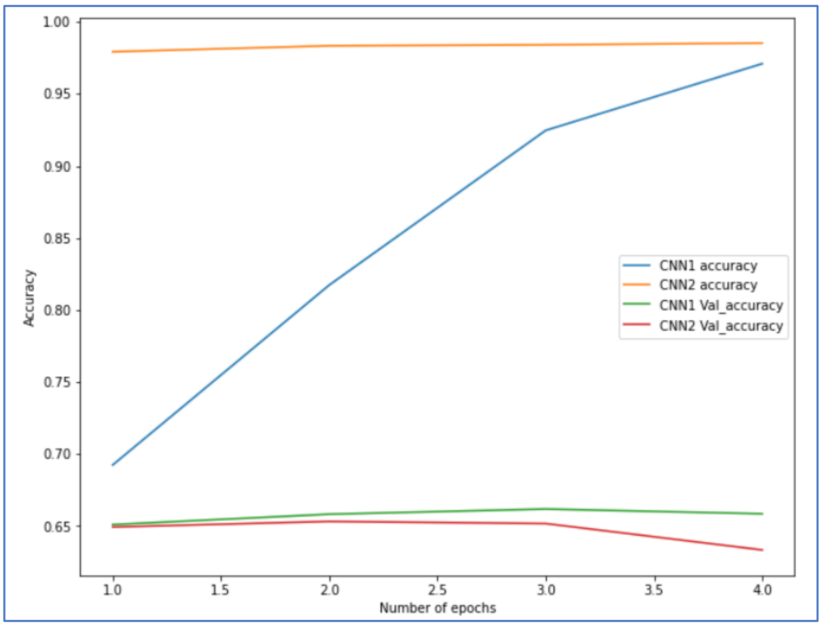
4. **Train and test accuracy for all three networks and comment (what happens and why) on the performance of the MLP vs CNNs.**





In the graph, "accuracy" means Training Accuracy and "val_accuracy" means Validation Accuracy. Again "loss" means training loss and "val_loss" means validation loss. In case of training accuracy, CNN1 has highest accuracy and MLP has worst accuracy. But the rate of improvement of accuracy is more in case of CNN1 while CNN2 has moderate rate of improvement but MLP has very slow improvement over the number of epochs. In case of validation accuracy, CNN2 has the highest validation accuracy while MLP has the worst accuracy after 5 epochs.

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?**



The terms "accuracy" and "val accuracy" in the graph refer to training and validation accuracy, respectively. "loss" refers to training loss, and "val loss" refers to validation loss.

In terms of training accuracy, CNN1 outperforms CNN2 after 5 epochs, while the validation accuracy of CNN1 and CNN2 is nearly identical after 5 epochs. CNN has a total of 25,997,130 trainable parameters, whereas CNN2 has 1,486,666. As a result, CNN1 has about 17 times more trainable parameters than CNN2. CNN1 takes about 8 seconds to complete 1 epoch, but CNN2 takes 3 seconds, which is to be anticipated given that CNN1 has more learnable parameters than CNN2. Even if the networks were trained for a larger number of epochs, CNN1 would outperform CNN2 in terms of accuracy because CNN1 had more learnable parameters.

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

* For CNN, Keras tuner can be used to find out the best hyper parameter for the network. In a keras tuner, we give the hyperparameter combinations that should be tested. A certain number of hyperparameter combinations is generated using iteration loops using a keras library function. Each model can be checked for accuracy using a validation set. The model giving the highest accuracy will be selected.

* A large dataset is crucial for improving the performance of the deep learning model. However, we can improve the performance of the model by augmenting the data we already have. Data augmentation techniques include affine transformations, perspective transformations, contrast changes, gaussian noise, dropout of regions, hue/saturation changes, cropping/padding, blurring etc.

* The number of hidden layer and neurons for MLP can be changed in various way to find out model giving the best accuracy.

Convolutional Neural Network (CNN): Convolutional Neural Network (CNN) is a deep neural network class which is used to analyse visual images.

Convolutional Layer: Computers images are represented as a matrix of pixels (NxNxd) — (Height X Width X Depth). Images can use three channels (RGB) or only one (Grayscale). In our dataset, the images are in RGB.

CNN Filters: CNN uses a collection of learnable filters. The filters detect the existence of a specific features in the original image. It is usually expressed as a matrix (MxMx3) for RGB image. When compared to the input file, it has a smaller dimension but the same depth. To generate the activation map, the filter is convolved across the input image. (Both width and height). The Conv2D function adds a c convolutional layer. For example, in the first convolution layer we create 64 filters of size 3x3. As we progress deeper into the network, the number of filters increases.

Activation Function: Activation function is placed at the end or in between a neural network. It decides whether a neuron will fire or not. There are various types of activation function.
Ex: Sigmoid, ReLU, SoftMax

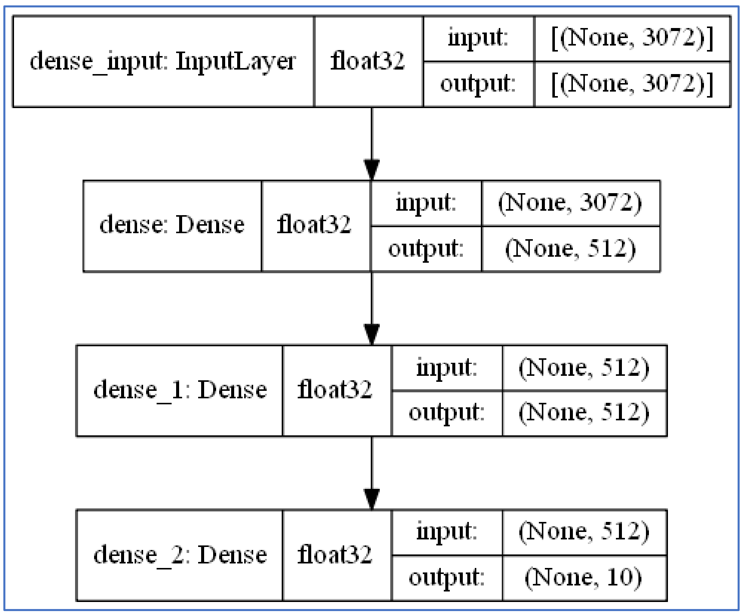
Pooling Layer: Its function is to reduce the spatial size of the representation to gradually reduces the number of parameters and computation in the network. Pooling layer acts independently on each feature map. Pooling layer controls overfitting by progressively reducing the spatial size of the network. Pooling is mostly used to down sample the feature map while retaining the relevant information.

Max-pooling: Max-pooling takes out the maximum from a pool. Filters are silded through the input image and at every stride, the maximum value is selected. MaxPool2D creates a max pooling layer, the only argument is the window size. We use a 2x2 window. In convolutional layer, the depth changes but it remains unaltered in max-pooling.

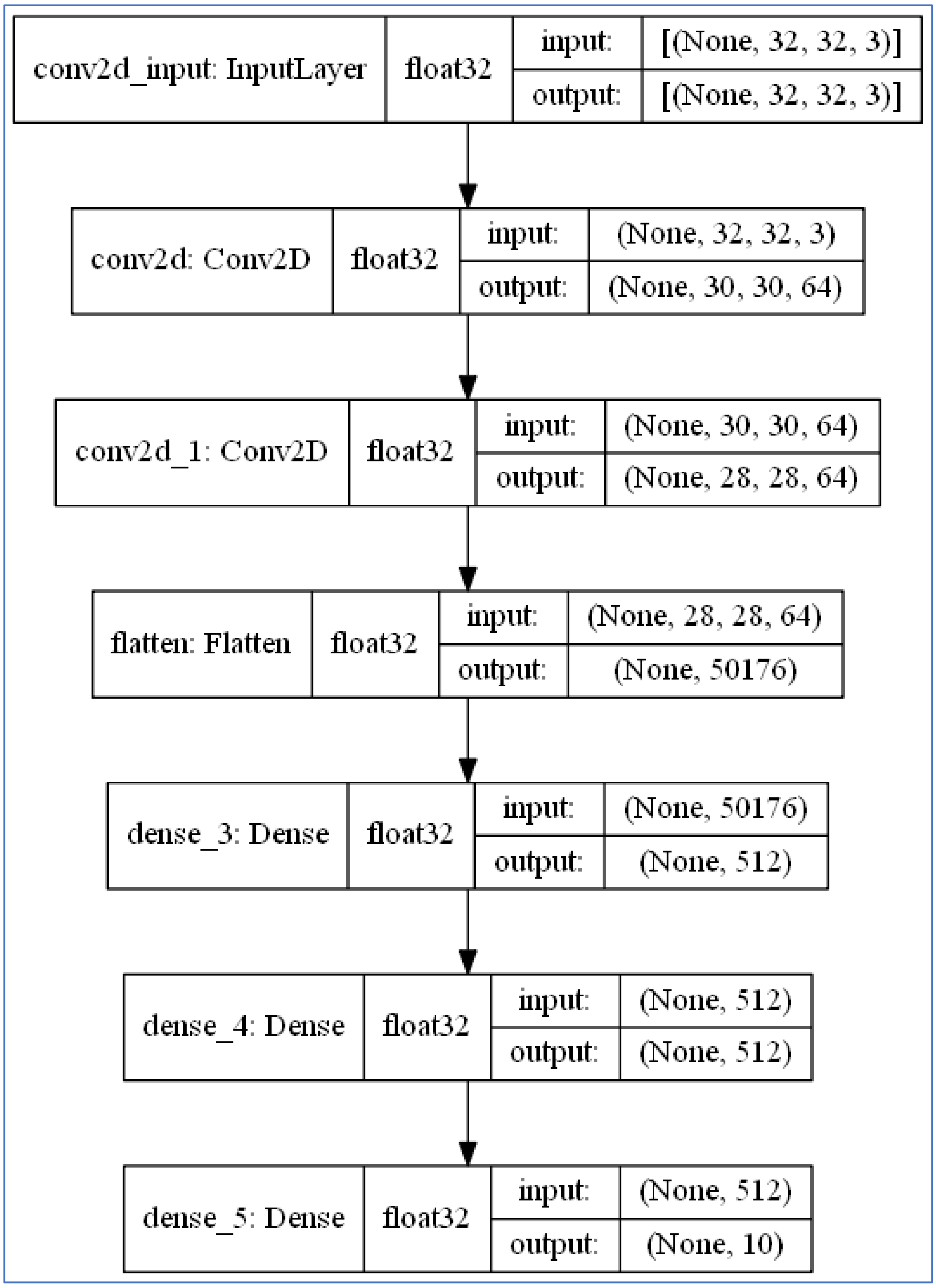
Flatten: There is a 'Flatten' layer between the convolutional layer and the fully linked layer. It transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier. Flattening is arranging the 3D volume of numbers into a 1D vector.

Dropout: Dropout is used in deep neural networks for regularization. Dropout is a technique used to prevent overfitting. During training, a neuron is briefly "dropped" with probability p at each iteration. We used a dropout rate of 0.2. It prevents the network from being dependent on a small number of neurons and forces each neuron to be able to operate independently. Dropout can be applied on input or hidden nodes but not on the output nodes.

CNN 2



CNN 1


conv2d_input: InputLayer
float32
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d: Conv2D
float32
input: (None, 32, 32, 3)
output: (None, 30, 30, 64)
conv2d_1: Conv2D
float32
input: (None, 30, 30, 64)
output: (None, 28, 28, 64)
flatten: Flatten
float32
input: (None, 28, 28, 64)
output: (None, 50176)
dense_3: Dense
float32
input: (None, 50176)
output: (None, 512)
dense_4: Dense
float32
input: (None, 512)
output: (None, 512)
dense_5: Dense
float32
input: (None, 512)
output: (None, 10)

Problem 2

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.**

We need to generate a new dataset using the features described above to estimate the next day's open using the previous three days i.e., open, high, low prices, and volume. I begin by loading the original dataset, which includes the date, close/last, Volume, Open, High, and Low. I get the values, using for loops to store the next day's open using the previous three days as the targets, and save the other four values of the most recent specified days (here it is 3) as the features, where each sample has 12 features, and save them all as a new dataset.

2. **Any preprocessing steps you followed.**

- I divided the features and targets from the loaded dataset.
- Applied MinMaxScaler to the features both on the training and testing sets. This step's goal is to scale each feature to a specific range.
- After min-max normalisation, the minimum values for each feature are changed to 0, the highest value is transformed to 1, and all other values are converted to a decimal between 0 and 1. The targets are kept as-is, without any normalisation.

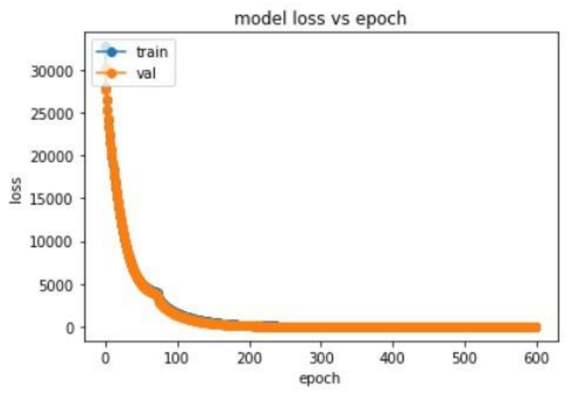
3. **All design steps you went through in finding the best network in your report and how you chose your final design.**

For the network architecture, I first attempted simple RNN layers, and then for this prediction purpose, I chose a Long Short-Term Memory network, which is a sophisticated time-series model that can handle both long-term and short-term data. The LSTM is a type of RNN that is supposed to avoid the problem of long-term dependency. I also tried to alter the unit numbers of each layer from small to large to get better results on the testing set. I use mean square error as a loss function and mean absolute error as an assessment metric to assess performance. The squared average distance between the real data and the anticipated data is measured by MSE. The absolute average distance between the real data and the anticipated data is measured by MAE. MAE is more accurate in determining whether the predicted value is close to the true value because the features are normalised, but the targets are not.

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

My final network consists of two LSTM layers, the last of which is a fully connected layer. Because I find that with more units, it takes fewer epochs to converge with good prediction performance, the first LSTM layer has 50 units, and the second LSTM layer has 150 units. The number of epochs was set to 600 because I believe the network has converged at this stage. Mean square error (MSE) is the loss function I use for training. The batch size is set to 32 by default, and the training method is defined as the "adam" optimizer, which is a common form of gradient descent that automatically tunes itself and produces good results.

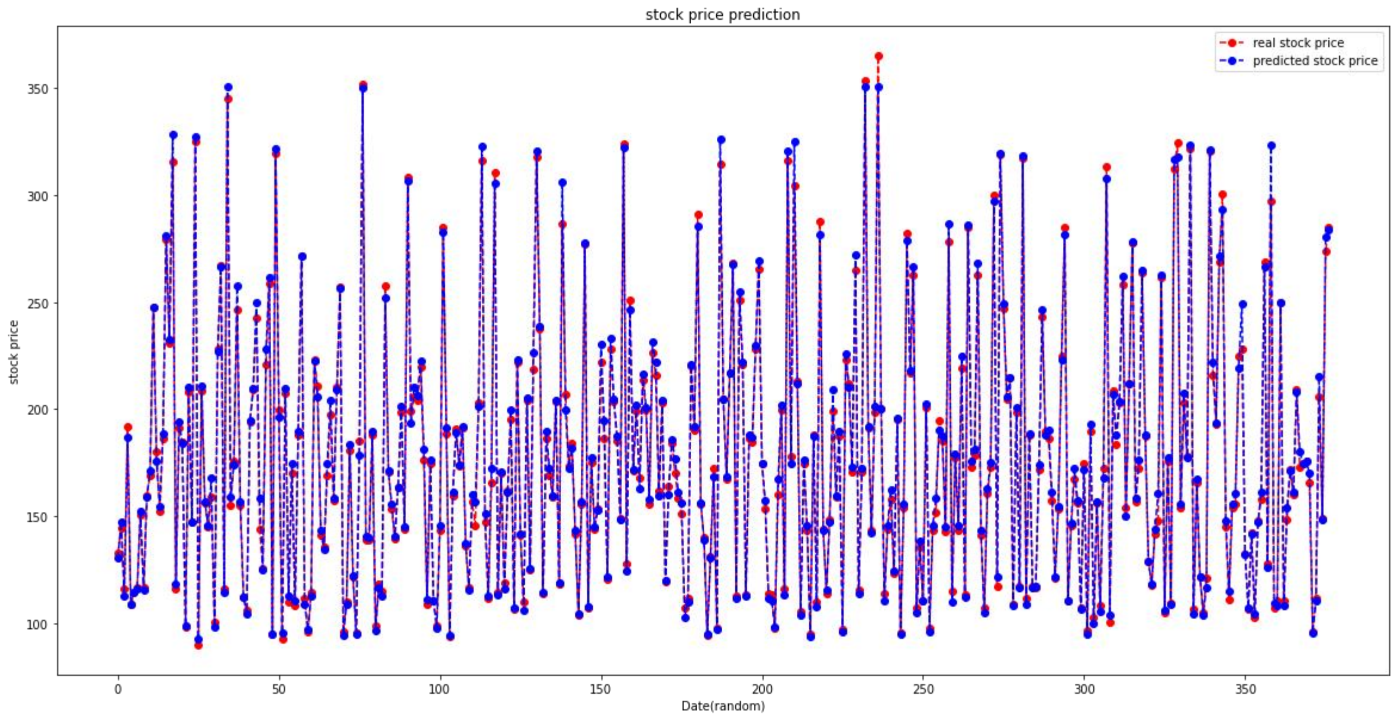
5. Output of the training loop with comments on your output



Comments:

The output of the training loop is shown in the figure above; as the number of training epochs increases, the loss reduces progressively. Initially, it declines extremely quickly, but after a few epochs, it converges to around 5000 and then continues to fall again, eventually converging to training loss of around 25, validation loss of about 20. The training MAE is around 3.2, whereas the val MAE is approximately 3.5. This might be due to the influence of Adam optimizer. Adam updates the learning rate, as opposed to traditional stochastic gradient descent, which maintains a single learning rate that does not vary throughout training.

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



Comments:

After training, I loaded the stored model to make the prediction. Figure above depicts the results of the tests. It predicts the results of the testing set. On the testing dataset, the MSE loss is about 22, while the MAE is 3.15. The figure shows that most forecasts are like the real value, while a few predictions are not near to the testing targets.

7. What would happen if you used more days for features?

I update the variable "days" in the train python script to use additional days as features to train the network. The model's performance increased to some amount after a few more days for features. After a few more days, the model's performance improved slightly in terms of features. When compared to a simple RNN, the LSTM is better at long-term memory. If there are more features, the network structure must be updated; however, the network structure is unchanged here, therefore there is more work to be done, not simply adding features, to obtain better performance.

Problem 3

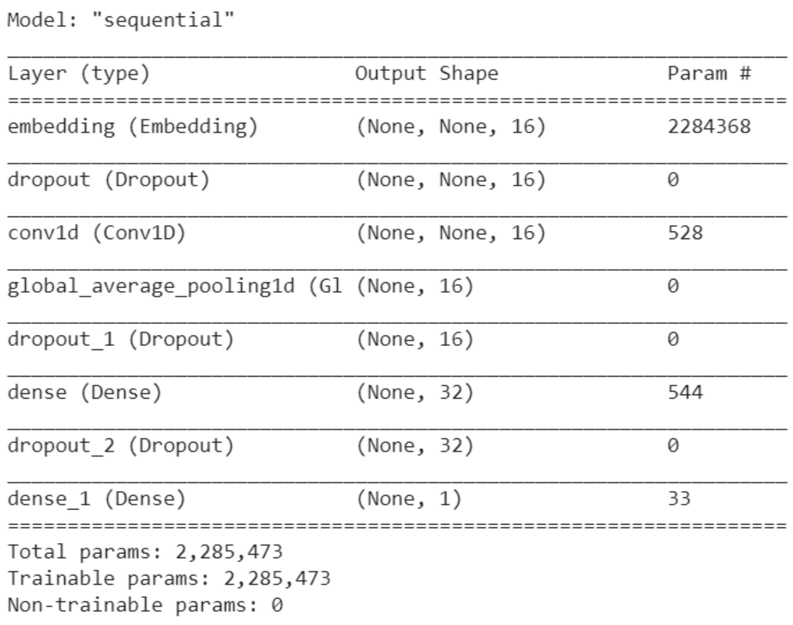
The IMDB large movie review dataset has many positive and negative reviews on movies. Download the dataset here (http://ai.stanford.edu/~amaas/data/sentiment/). Check the README file as it provides a description of the dataset. Use the provided training and testing data for your network. You would need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

Preprocessing Steps

- Loading the aclImdb_v1.tar.gz from the data directory using get_file.
- Texts of movie reviews are stored as .txt files in /pos and /neg directories, I read the texts from the path and return to the raw texts and their labels.
- Cleaning the data and removing punctuation and special characters that are not useful for training.
- Vectorizing the text into a list of integers.
- Using keras text tokenizer I turned the cleaned text into a sequence of integers where each integer is the index of a token in the dictionary.
- The original reviews are of varying lengths, but the network requires inputs of the same length. As a result, I utilized a pad sequence to ensure that all the inputs had the same length. Sequences lower than the max length we specified will be padded with zeros, while sequences larger than that length will be cut.

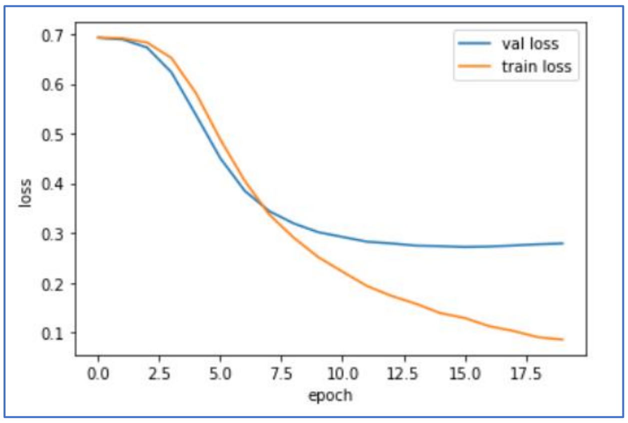
Building the Network

Below image shows the neural network structure. The network is basically a sequential CNN model because CNN's ability to extract local features from text is beneficial for sentiment analysis. Firstly, there is an embedding layer that transforms each integer into the ith line of the embedding weights matrix, followed by a convolution layer with 16 filters and a kernel size of 2 with a relu activation function, and a global average pooling layer, followed by two fully connected layers, one with 32 nodes and a relu activation function. The final layer has one output node and uses a sigmoid activation function. To minimize overfitting, I added Dropout layers after the operation layers.

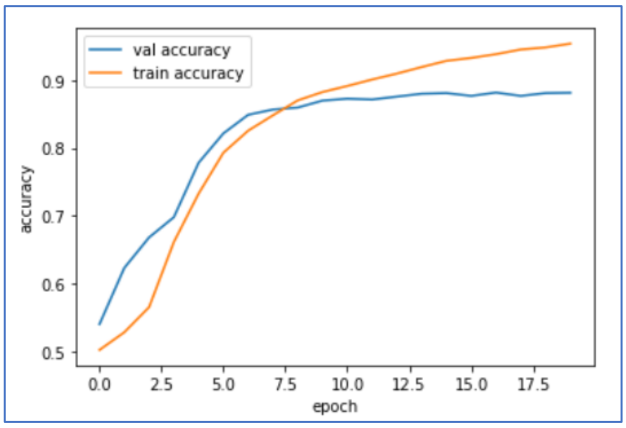


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 16)          2284368
_________________________________________________________________
dropout (Dropout)            (None, None, 16)          0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 16)          528
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0
_________________________________________________________________
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: 2,285,473
Trainable params: 2,285,473
Non-trainable params: 0
```

Training and Testing Accuracy



i. train loss and val loss



ii. train and val accuracy

```
Epoch 20/20
35/35 [==============================] - 10s 276ms/step - loss: 0.1418 - acc: 0.9539 - val_loss: 0.2973 - val_acc: 0.8815
```

As is shown in above plot, after training for 20 epochs, the training loss reaches to 0.14 and the validation reaches to 0.29. And the training accuracy is 95.3%, and the validation accuracy is 88.1%. When evaluating on the test dataset, the test accuracy is 85.9%.

Observations

We observe from training and testing accuracies that the training accuracy goes to increase over 94%, however, both the validation accuracy and testing accuracy seems not going to increase over 90% as this model has some limits. To try to find a better approach, we may consider about training with more samples, processing the data to be better to train, focusing on adjusting the network and hyper-parameters, or maybe trying combine CNN together with RNN to see if it will get better results.