Indian Institute of Technology Delhi

ELL784 Introduction to Machine Learning: Assignment 2

Mohammed Jawahar Nausheen: 2019CS10371

Himanshu Nigam: 2021EET2828

Deepak: 2019MT10685

Problem 1 Part A

About dataset

The MNIST dataset contains black and white, hand-written (numerical) digits that are 28x28 pixels large. It contains 60,000 training and 10,000 test images.

Method Used

The training images are shuffled and 10,000 images from it are taken for validation and the model is trained using the remaining 50,000 images. We have normalised the dataset using torchvision.transorms.Normalize and then divided it into batch size of 64. Then we have run the feed forward neural network with two hidden layers using the PyTorch framework. We have used Stochastic Gradient Descent as the optimizer and ReLU as the activation function.

Loss Function

Since we had a multi-class classification problem, we have used nn.CrossEntopryLoss() as the Loss function.
Cross Entropy minimizes the distance between two probability distributions - predicted and actual. Our classifier predicts whether the given image is '0' or '1' or '2' or ... or '9' with a probability associated with each. Suppose the original image is of '1' and the model predicts 0.8, 0.1,0.05,0.04,... as the probabilities of the respective classes whereas the true probability looks like [0,1,0,0,...,0]. What we ideally want is that our predicted probabilities should be close to this original probability distribution. So cross entropy makes sure we are minimizing the difference between the two probability. This is the reason why we are using Cross Entropy as Loss Function.

Accuracy of the Model

Training Accuracy=100.0%
Testing Accuracy=98.06% dblfloatfix

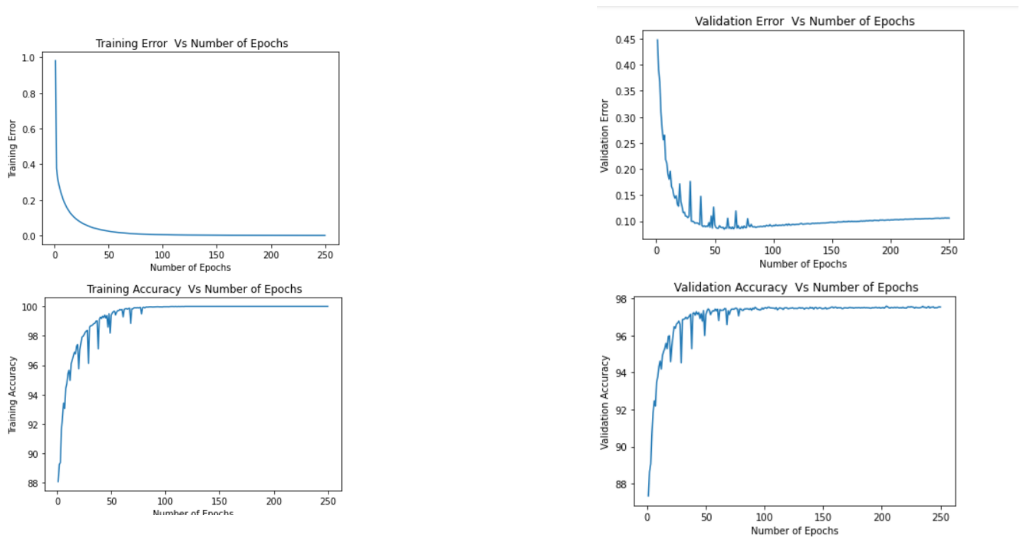


Figure 1: *Error and Accuracy vs No. of Epochs for Problem 1 part A*

Problem 1 part B

As the training accuracy in part A is 100% while the testing accuracy is 98.06%. Therefore our model has overfitted in part A. We have used the following regularization in order to overcome overfitting.

L2 Regularization

L2 regularization adds sum of squares of all weights in the model to cost function. It is able to learn complex data patterns and gives non-sparse solutions We have implemented L2 Regularization using the weight_decay parameter in optimizer. This adds regularization term to the loss function, with the effect of shrinking the parameter estimates, making the model simpler and less likely to overfit.

Training Accuracy=96.87%
Testing Accuracy=96.54%

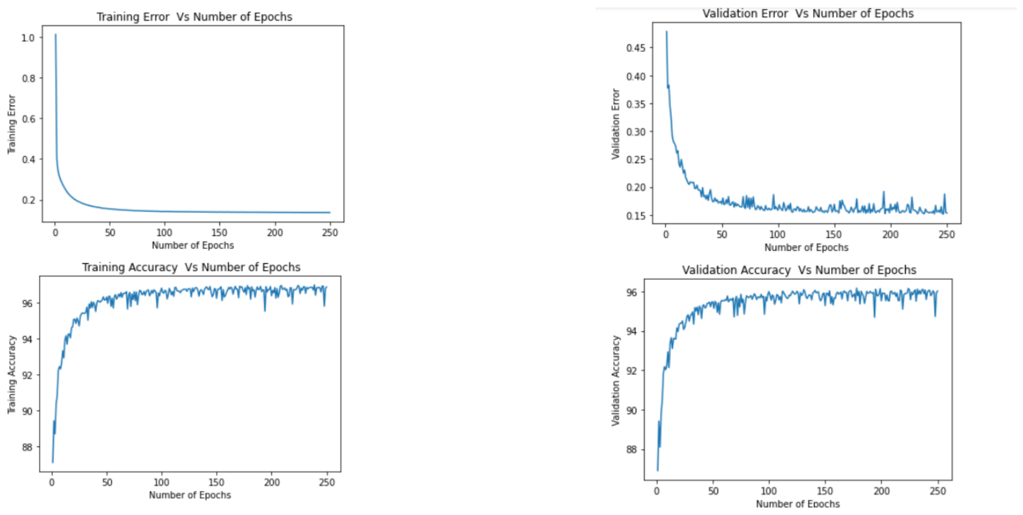


Figure 2: *Error and Accuracy vs No. of Epochs for Problem 1 part B (L2 Regularisation)*

Dropout

Dropout refers to dropping out units in a neural network. By dropping a unit out, it means to remove it temporarily from the network. The choice of which units to drop is random. Each unit is retained with a fixed probability p independent of other units.
This procedure effectively generates slightly different models with different neuron topologies at each iteration, thus giving neurons in the model, less chance to coordinate in the memorisation process that happens during overfitting. Thus making it better at generalization and cope with overfitting issue.
We implemented dropout using torch.nn.Dropout(p) where it randomly zeroes some of the elements of the input tensor with probability p(taken as 0.5 for problem 1). Output shape will remain same as of input while implementing dropout

Training Accuracy=99.99%
Testing Accuracy=98.38%

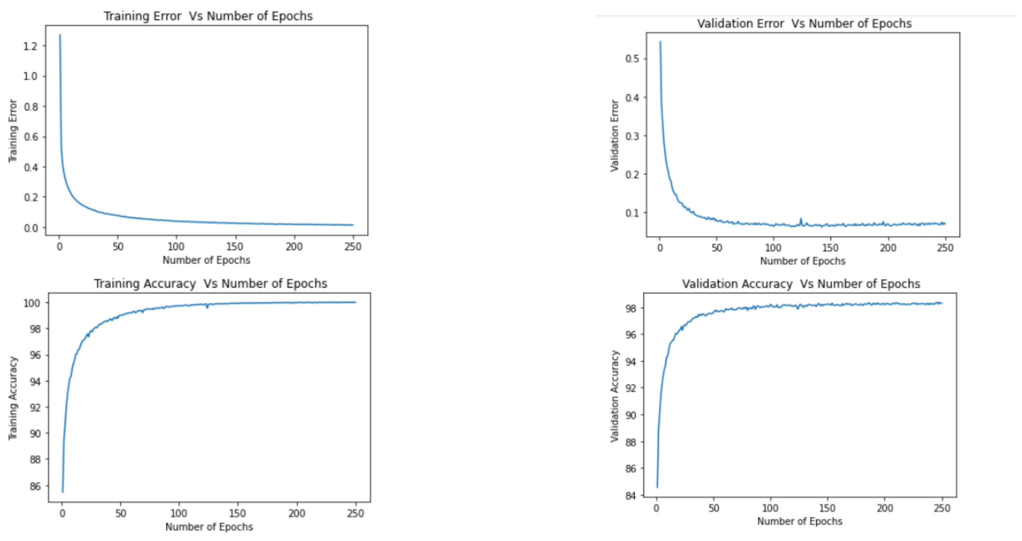


Figure 3: *Error and Accuracy vs No. of Epochs for Problem 1 part B (Dropout)*

Early Stopping

Early Stopping implies to stop training of the model early before it reaches overfitting stage. On training the model, initially, both training accuracy and validation accuracy increases. But as number of epochs increases, training accuracy increases while validation accuracy becomes constant or it start decreasing. This is the stage when our model starts to overfit. So, we stop the further epochs in the model for training
Number of epochs for early stopping=30
Training Accuracy=99.379%
Testing Accuracy=98.06%

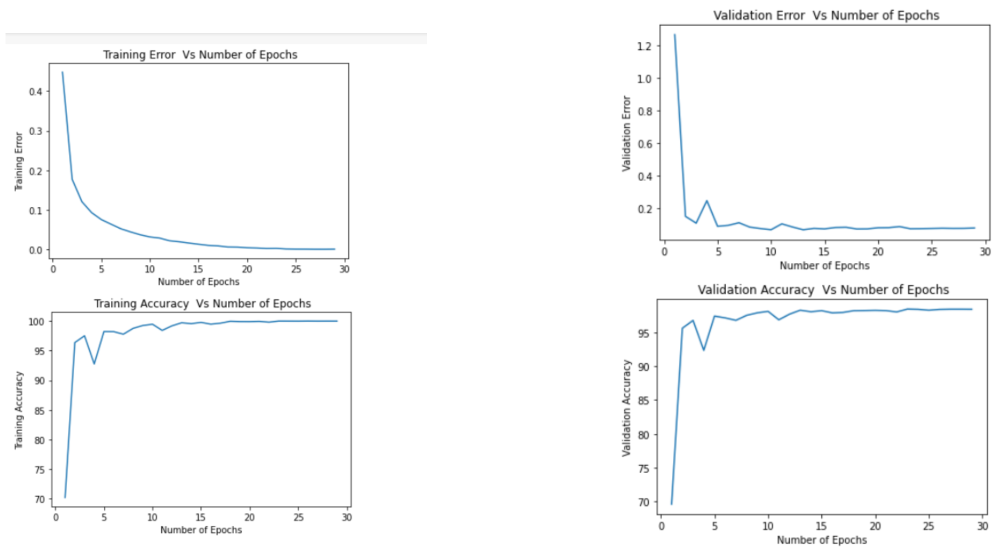


Figure 4: *Error and Accuracy vs No. of Epochs for Problem 1 part B (Early Stopping)*

Problem 2

About dataset

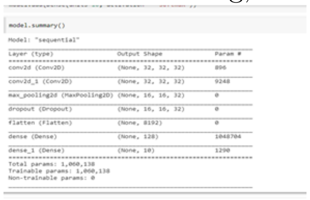
a. CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. There are 50,000 training images and 10,000 Test images. The dataset is divided into 5 training batch and 1 test batch with 10,000 images each.
b. There are 10 classes in dataset. The classes are mutually exclusive and have no overlap between them.
c. The following parameters were used to analyse the data:
i. Number of convolutional (Conv) layers
ii. Fully connected (FC) layers
iii. No of filters in different layers
iv. Max-pooling
v. Number of epochs
vi. Stride
vii. Activations

Pre-processing the data

The dataset was normalised by dividing each pixel value by 255. This is because pixel values ranges from 0-255 for each of the channel i.e RGB and they will be regularised to range 0 to 1.

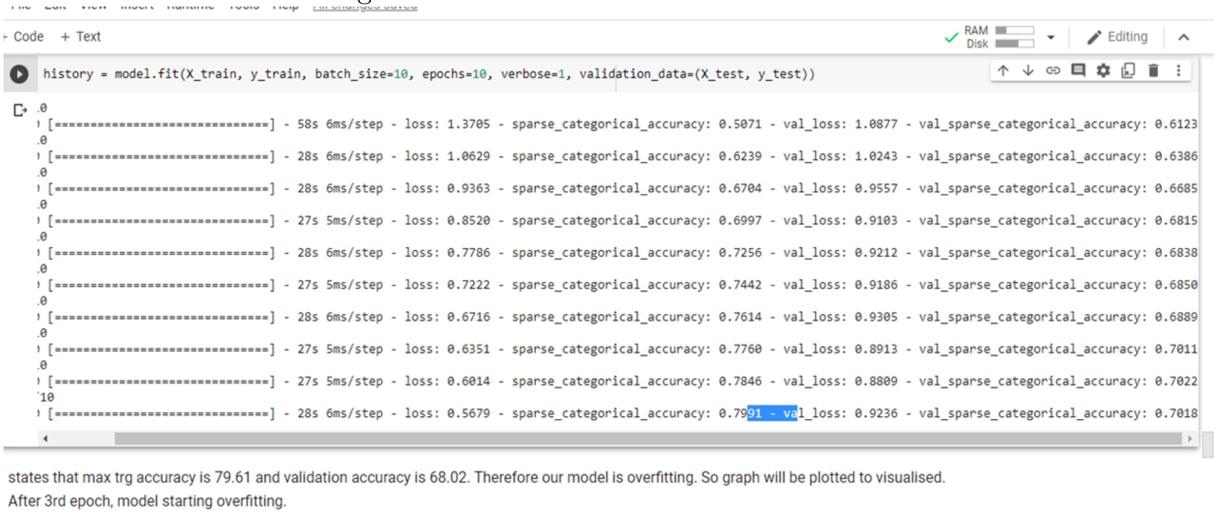
Building the model

For initial building, following parameters were used:



```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1048704
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,060,138
Trainable params: 1,060,138
Non-trainable params: 0
_________________________________________________________________
```

Adam optimizer was used for 10 epochs, and maximum training accuracy achieved was around 79.9 percent however, validation accuracy achieved is around 70.18 percent. This clearly states that the model is over-fitting.



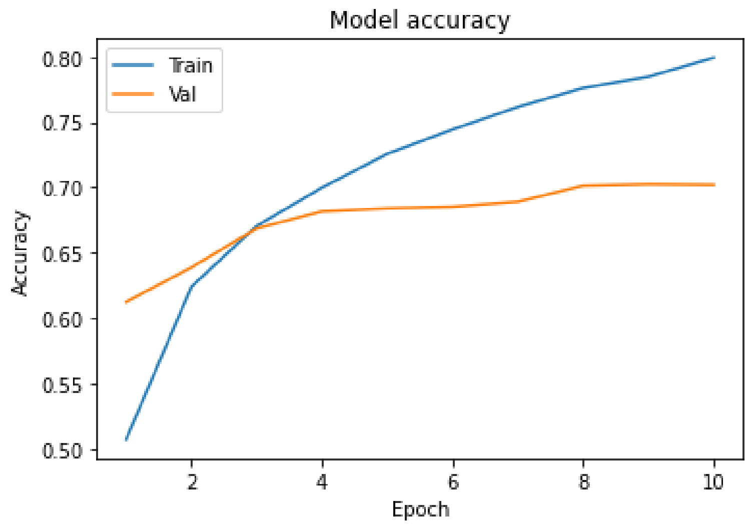
```
history = model.fit(X_train, y_train, batch_size=10, epochs=10, verbose=1, validation_data=(X_test, y_test))
```

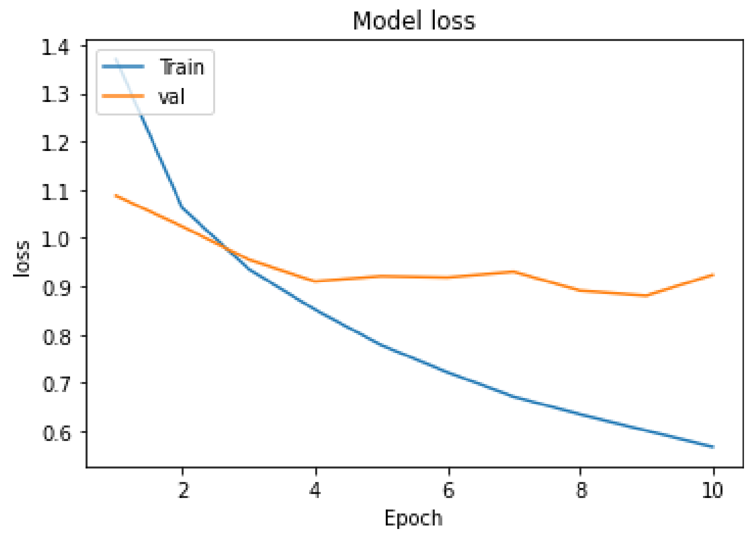
```
.0
[==============================] - 58s 6ms/step - loss: 1.3705 - sparse_categorical_accuracy: 0.5071 - val_loss: 1.0877 - val_sparse_categorical_accuracy: 0.6123
.0
[==============================] - 28s 6ms/step - loss: 1.0629 - sparse_categorical_accuracy: 0.6239 - val_loss: 1.0243 - val_sparse_categorical_accuracy: 0.6386
.0
[==============================] - 28s 6ms/step - loss: 0.9363 - sparse_categorical_accuracy: 0.6704 - val_loss: 0.9557 - val_sparse_categorical_accuracy: 0.6685
.0
[==============================] - 27s 5ms/step - loss: 0.8520 - sparse_categorical_accuracy: 0.6997 - val_loss: 0.9103 - val_sparse_categorical_accuracy: 0.6815
.0
[==============================] - 28s 6ms/step - loss: 0.7786 - sparse_categorical_accuracy: 0.7256 - val_loss: 0.9212 - val_sparse_categorical_accuracy: 0.6838
.0
[==============================] - 27s 5ms/step - loss: 0.7222 - sparse_categorical_accuracy: 0.7442 - val_loss: 0.9186 - val_sparse_categorical_accuracy: 0.6850
.0
[==============================] - 28s 6ms/step - loss: 0.6716 - sparse_categorical_accuracy: 0.7614 - val_loss: 0.9305 - val_sparse_categorical_accuracy: 0.6889
.0
[==============================] - 27s 5ms/step - loss: 0.6351 - sparse_categorical_accuracy: 0.7760 - val_loss: 0.8913 - val_sparse_categorical_accuracy: 0.7011
.0
[==============================] - 27s 5ms/step - loss: 0.6014 - sparse_categorical_accuracy: 0.7846 - val_loss: 0.8809 - val_sparse_categorical_accuracy: 0.7022
'10
[==============================] - 28s 6ms/step - loss: 0.5679 - sparse_categorical_accuracy: 0.7991 - val_loss: 0.9236 - val_sparse_categorical_accuracy: 0.7018
```

states that max trg accuracy is 79.61 and validation accuracy is 68.02. Therefore our model is overfitting. So graph will be plotted to visualised.
After 3rd epoch, model starting overfitting.

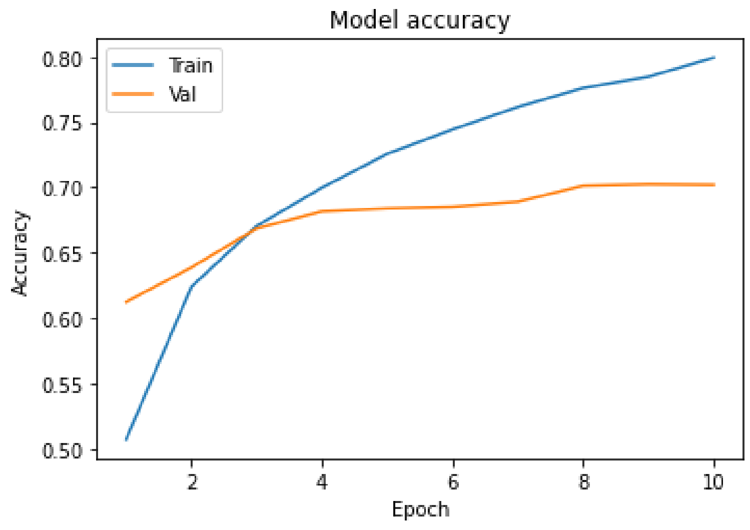
Plotting learning curve

Graph was plotted against loss and accuracy of training and validation set:

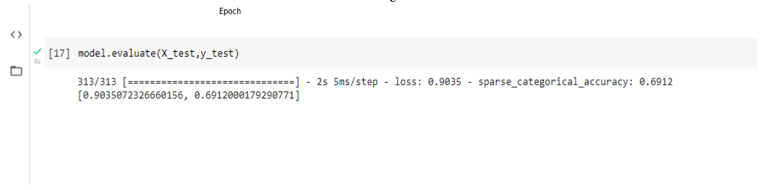




The graph clearly shows that accuracy to training set kept increasing till 3 epoch thereafter become constant.
The accuracy of test data is shown below.



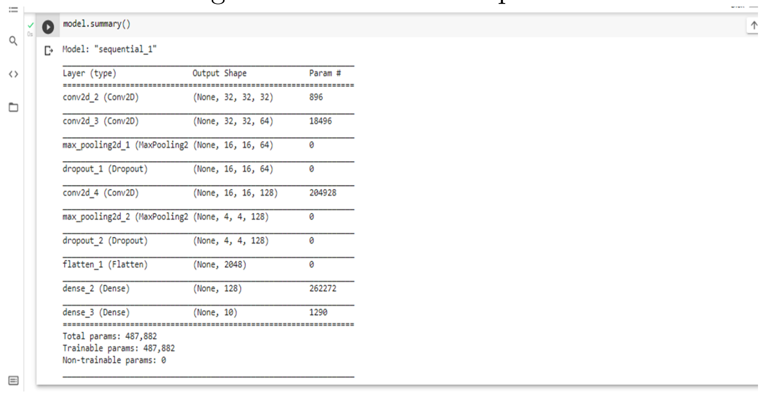
The graph clearly shows that accuracy to training set kept increasing till 3 epoch thereafter become constant. The accuracy of test data is shown below.



```
[17] model.evaluate(X_test,y_test)

313/313 [==============================] - 2s 5ms/step - loss: 0.9035 - sparse_categorical_accuracy: 0.6912
[0.9035072326660156, 0.6912000179290771]
```

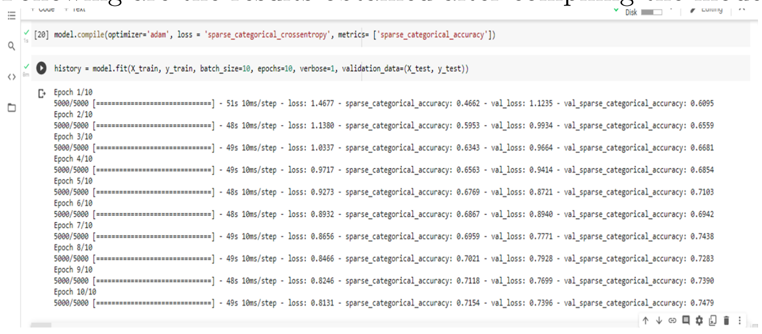
Refining the model

Various combinations of above mention parameters were applied to further improve the performance of training data. The details of parameters are :



```
model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 487,882
Trainable params: 487,882
Non-trainable params: 0
_________________________________________________________________
```

Following are the results obtained after compiling the model:

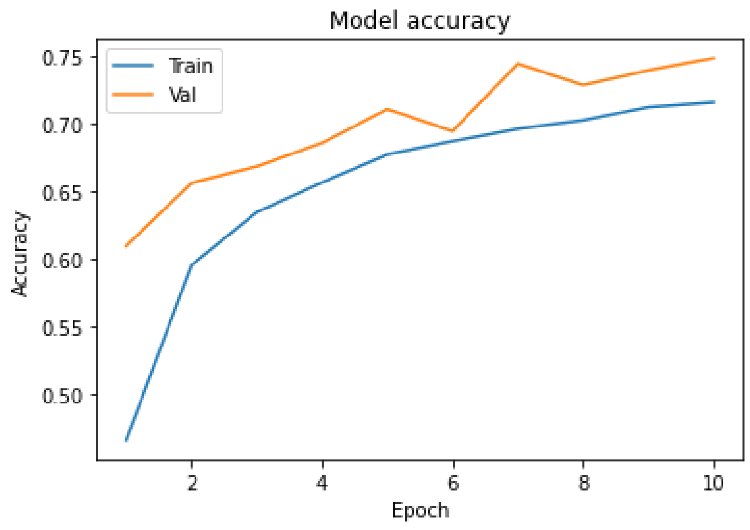


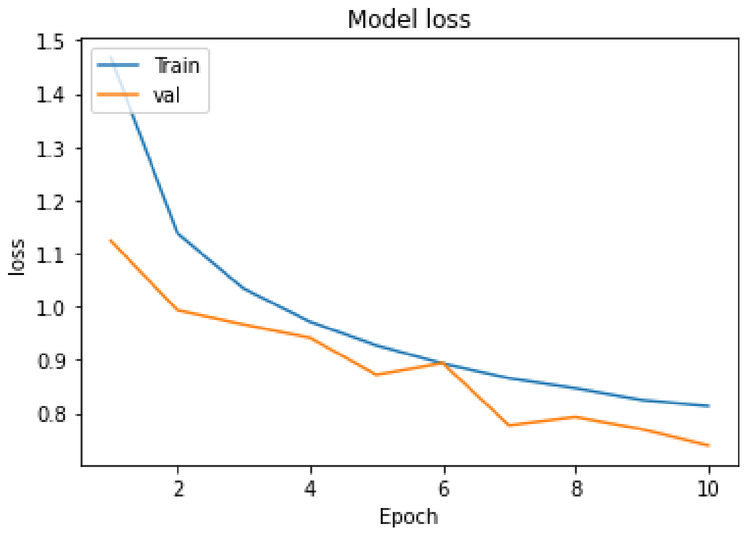
```
[20] model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy', metrics= ['sparse_categorical_accuracy'])

history = model.fit(X_train, y_train, batch_size=10, epochs=10, verbose=1, validation_data=(X_test, y_test))

Epoch 1/10
5000/5000 [==============================] - 51s 10ms/step - loss: 1.4677 - sparse_categorical_accuracy: 0.4662 - val_loss: 1.1235 - val_sparse_categorical_accuracy: 0.6095
Epoch 2/10
5000/5000 [==============================] - 48s 10ms/step - loss: 1.1380 - sparse_categorical_accuracy: 0.5953 - val_loss: 0.9934 - val_sparse_categorical_accuracy: 0.6559
Epoch 3/10
5000/5000 [==============================] - 49s 10ms/step - loss: 1.0337 - sparse_categorical_accuracy: 0.6343 - val_loss: 0.9664 - val_sparse_categorical_accuracy: 0.6681
Epoch 4/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.9717 - sparse_categorical_accuracy: 0.6563 - val_loss: 0.9414 - val_sparse_categorical_accuracy: 0.6854
Epoch 5/10
5000/5000 [==============================] - 48s 10ms/step - loss: 0.9273 - sparse_categorical_accuracy: 0.6769 - val_loss: 0.8721 - val_sparse_categorical_accuracy: 0.7103
Epoch 6/10
5000/5000 [==============================] - 48s 10ms/step - loss: 0.8932 - sparse_categorical_accuracy: 0.6867 - val_loss: 0.8940 - val_sparse_categorical_accuracy: 0.6942
Epoch 7/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.8656 - sparse_categorical_accuracy: 0.6959 - val_loss: 0.7771 - val_sparse_categorical_accuracy: 0.7438
Epoch 8/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.8466 - sparse_categorical_accuracy: 0.7021 - val_loss: 0.7928 - val_sparse_categorical_accuracy: 0.7283
Epoch 9/10
5000/5000 [==============================] - 48s 10ms/step - loss: 0.8246 - sparse_categorical_accuracy: 0.7118 - val_loss: 0.7699 - val_sparse_categorical_accuracy: 0.7390
Epoch 10/10
5000/5000 [==============================] - 49s 10ms/step - loss: 0.8131 - sparse_categorical_accuracy: 0.7154 - val_loss: 0.7396 - val_sparse_categorical_accuracy: 0.7479
```

The result clearly indicates, accuracy of training model is decreased to 71.54percent however, Validation accuracy is increased to 74.7 percent. This indicated that our model is no longer over-fitting.

Plotting learning curve

Again the graph was plotted against loss and accuracy of training and validation set and following plot was observed:

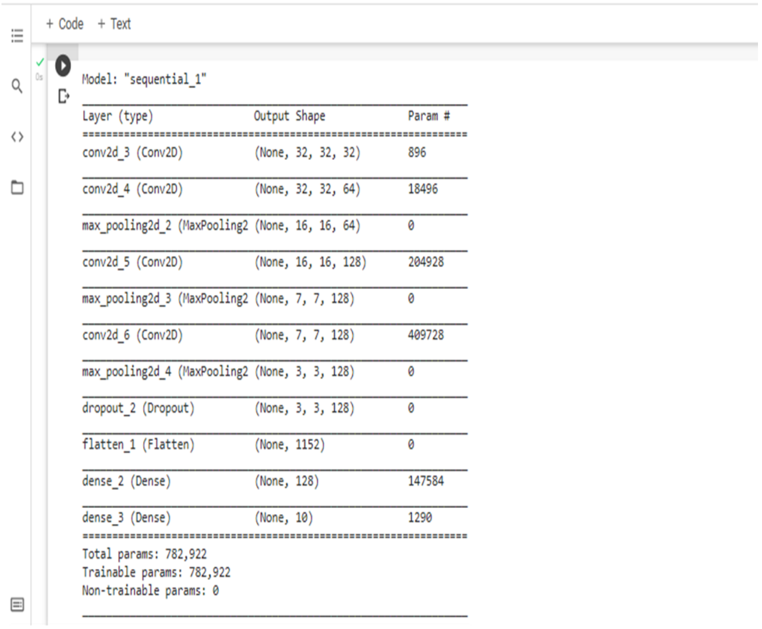




After analysing the graph , it is evident that problem of over-fitting is resolved. The validation accuracy has also increased . Some more refinements were carried out to improve the performance of the model subsequently.

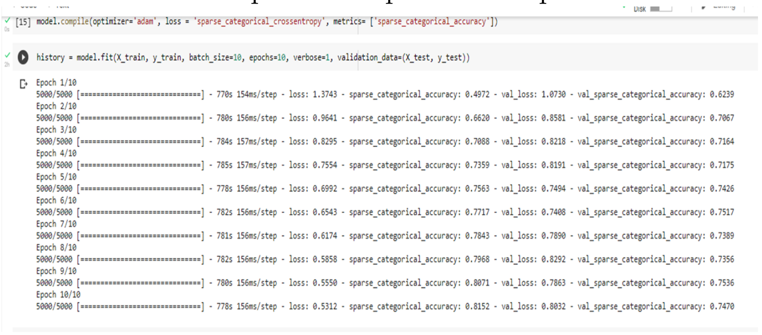
Final model

The final model consists of following attributes:



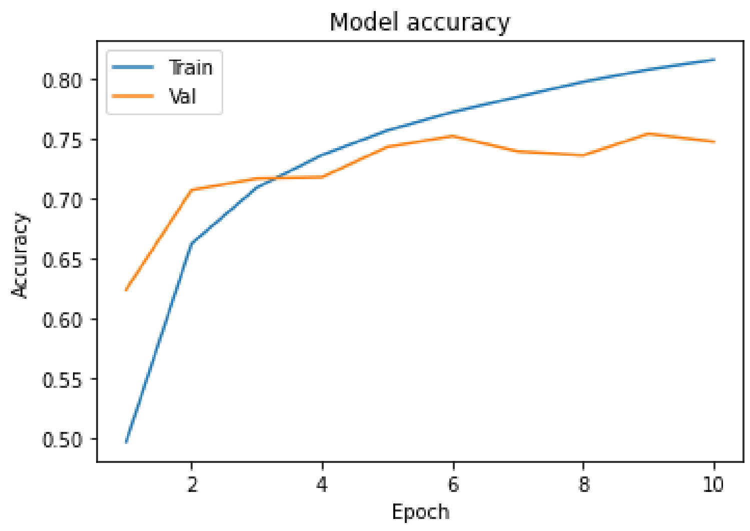
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       204928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 128)         409728
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               147584
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 782,922
Trainable params: 782,922
Non-trainable params: 0
_________________________________________________________________
```

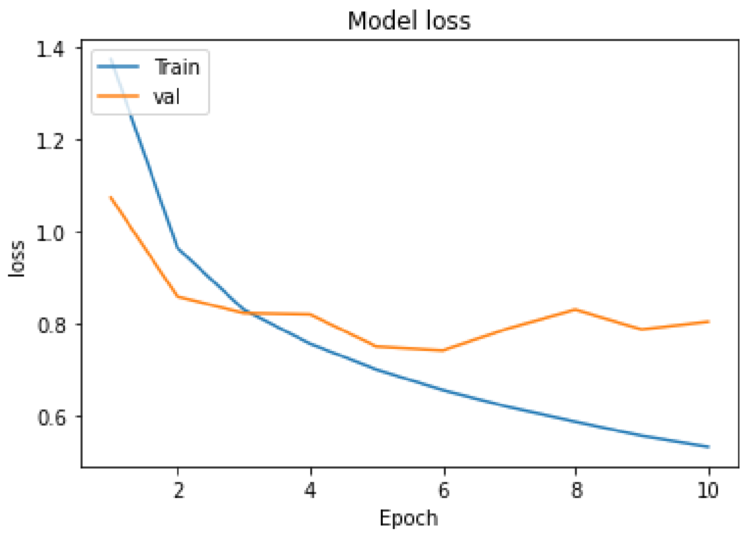
It was observed that further increasing the Conv2D layer would not increase the accuracy and thus, overall 4 layers with 32 , 32, 128 and 128 filters were used.
The overall training efficiency achieved was 81.2 percent whereas maximum validation accuracy achieved was 75.36 percent. Epoch was kept 10.



```
[15] model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy', metrics= ['sparse_categorical_accuracy'])

history = model.fit(X_train, y_train, batch_size=10, epochs=10, verbose=1, validation_data=(X_test, y_test))

Epoch 1/10
5000/5000 [==============================] - 770s 154ms/step - loss: 1.3743 - sparse_categorical_accuracy: 0.4972 - val_loss: 1.0730 - val_sparse_categorical_accuracy: 0.6239
Epoch 2/10
5000/5000 [==============================] - 780s 156ms/step - loss: 0.9641 - sparse_categorical_accuracy: 0.6620 - val_loss: 0.8581 - val_sparse_categorical_accuracy: 0.7067
Epoch 3/10
5000/5000 [==============================] - 784s 157ms/step - loss: 0.8295 - sparse_categorical_accuracy: 0.7088 - val_loss: 0.8218 - val_sparse_categorical_accuracy: 0.7164
Epoch 4/10
5000/5000 [==============================] - 785s 157ms/step - loss: 0.7554 - sparse_categorical_accuracy: 0.7359 - val_loss: 0.8191 - val_sparse_categorical_accuracy: 0.7175
Epoch 5/10
5000/5000 [==============================] - 778s 156ms/step - loss: 0.6992 - sparse_categorical_accuracy: 0.7563 - val_loss: 0.7494 - val_sparse_categorical_accuracy: 0.7426
Epoch 6/10
5000/5000 [==============================] - 782s 156ms/step - loss: 0.6543 - sparse_categorical_accuracy: 0.7717 - val_loss: 0.7408 - val_sparse_categorical_accuracy: 0.7517
Epoch 7/10
5000/5000 [==============================] - 781s 156ms/step - loss: 0.6174 - sparse_categorical_accuracy: 0.7843 - val_loss: 0.7890 - val_sparse_categorical_accuracy: 0.7389
Epoch 8/10
5000/5000 [==============================] - 782s 156ms/step - loss: 0.5858 - sparse_categorical_accuracy: 0.7968 - val_loss: 0.8292 - val_sparse_categorical_accuracy: 0.7356
Epoch 9/10
5000/5000 [==============================] - 780s 156ms/step - loss: 0.5550 - sparse_categorical_accuracy: 0.8071 - val_loss: 0.7863 - val_sparse_categorical_accuracy: 0.7536
Epoch 10/10
5000/5000 [==============================] - 778s 156ms/step - loss: 0.5312 - sparse_categorical_accuracy: 0.8152 - val_loss: 0.8032 - val_sparse_categorical_accuracy: 0.7470
```

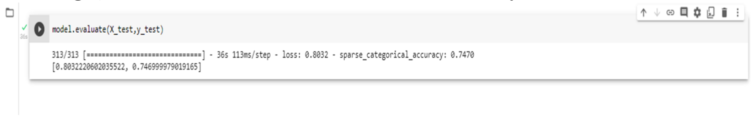
Table also shows that there is over fitting which can further be visualized using following graph.





Results

Given the settings , max accuracy achieved by training set was 81.5 percent and for validation set, 75.36 percent.However final accuracy for validation set in this model came out to be 74.70 percent. Accuracy can further be improved using other parameters, however, using current settings, this was the maximum accuracy achieved.



```
model.evaluate(X_test,y_test)

313/313 [==============================] - 36s 113ms/step - loss: 0.8032 - sparse_categorical_accuracy: 0.7470
[0.8032220602035522, 0.746999979019165]
```

Activations used

Out of all the activations tries, Relu gave the best result in our model. Relu or Rectified Linear Unit was chosen since it does not activate all the neurons at the same time. This implies that neurons will be deactivated when output of linear transformation is less than 0. For the negative input values, the result is zero that means the neuron does no gets activated and hence Relu is far more efficient computationally when compared to sigmoid and tanH function because Relu just needs to pick up max(0,x) and not perform expensive exponential operations as in sigmoid
.
In pratical, networks with Relu show better convergence performance then sigmoid.