EE569- Introduction to Digital Image Processing

Homework #5 Project

Sourabh J. Tirodkar
3589406164
Submission Date- 3rd May 2020

Problem 2: EE569 Competition- CIFAR 10 Classification

1. ABSTRACT AND MOTIVATION

CIFAR-10 dataset has been used to implement LeNet5 model architecture. Convolutional Neural Networks has wide variety of applications in Computer Vision, Image classification and recognition, etc. CNN are necessary to extract the features from the images which are key elements in those applications. The HW5 model gives the accuracy which can be improved and made the model better.

Many State of Art models have been published and I tried to implement some of those to create new model and achieve higher accuracy.
In most cases, modifying the parameters in the given architecture gives us the better result.

Also, Data Augmentation is also used as reading about it fetched me that it trains the model better. More details about using data augmentation is further discussed.

Model size is also one factor which is been considered while designing the same along with the model training time.

2. APPROACH

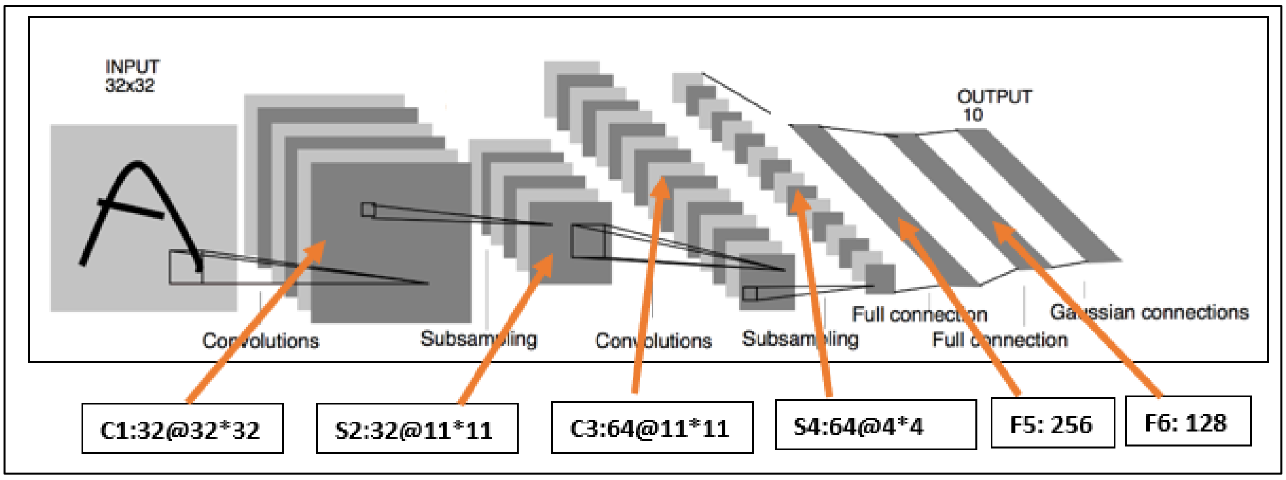


Fig. 1 Modified LeNet5 Architecture

The LeNet architecture is same as the one displayed in the previous section. The main functionality of architecture is to extract the key features from the data and using it to train the model. LeNet model consists of 7 layers. The layers are as follows:

1. C1- Convolution Layer
2. S2- Sub-sampling layer
3. C3- Convolution layer
4. S4- Sub-sampling layer
5. C5- Fully Connected layer
6. F6- Fully connected layer
7. Output- Softmax layer.

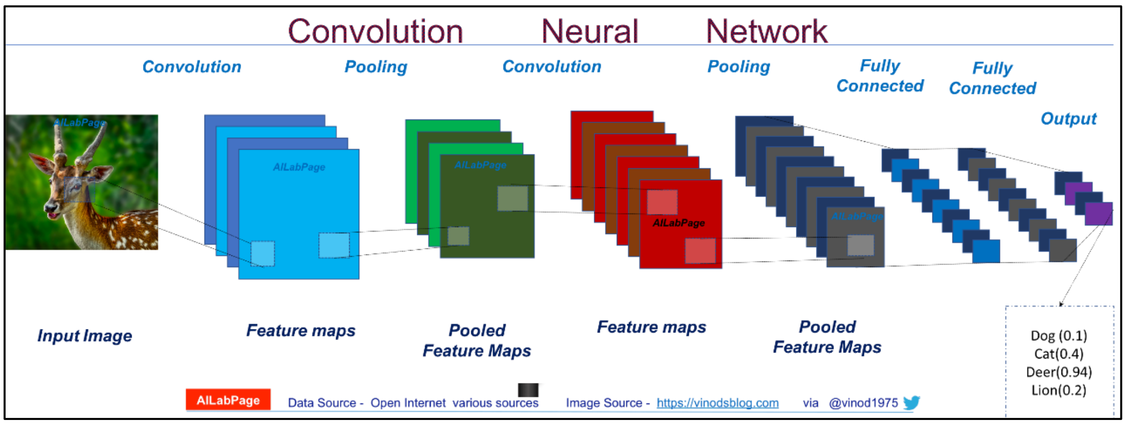
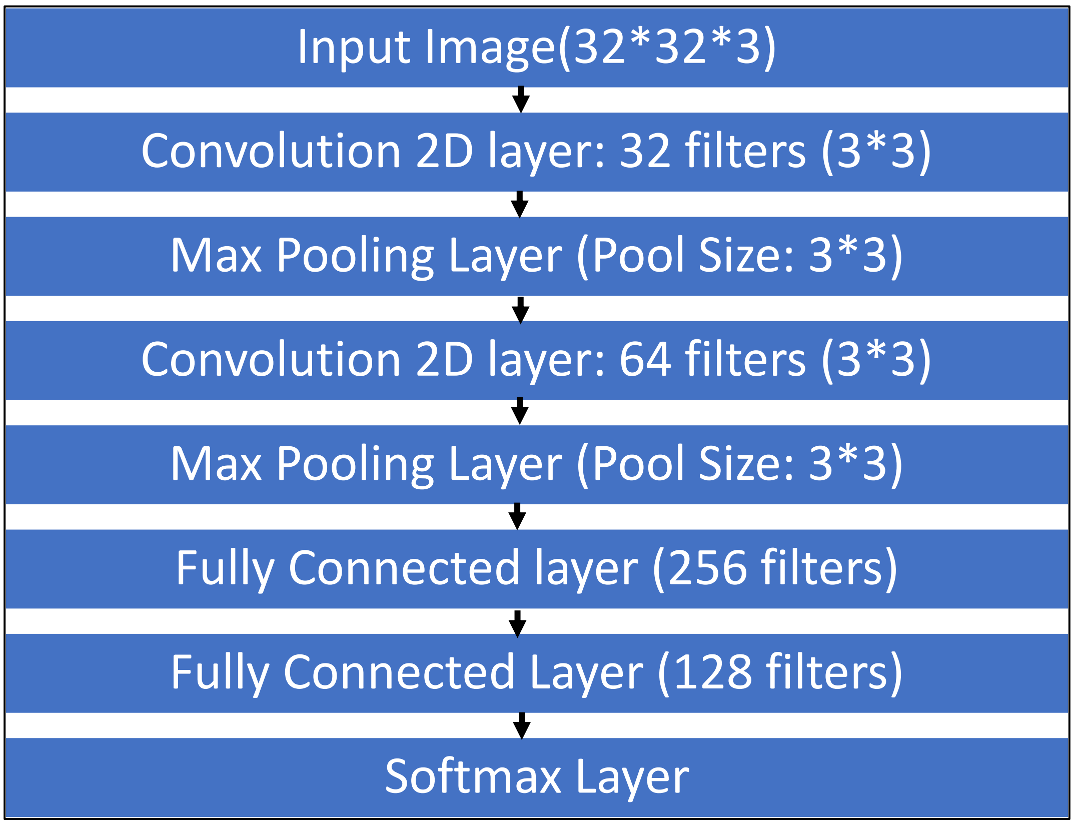


Fig. 2 LeNet5 Architecture

Flowchart


Input Image(32*32*3)
Convolution 2D layer: 32 filters (3*3)
Max Pooling Layer (Pool Size: 3*3)
Convolution 2D layer: 64 filters (3*3)
Max Pooling Layer (Pool Size: 3*3)
Fully Connected layer (256 filters)
Fully Connected Layer (128 filters)
Softmax Layer

Parameters: Algorithm run on CIFAR10 dataset

Output size calculation:

$$\text{Output width} = \frac{\text{width} - \text{filter size} + 2 \times \text{padding}}{\text{stride}} + 1$$

Input image size= 32*32, color image

C1: Kernel filter used in convolution: 3*3, 32 filters are used.
Padding used.
Activation- ReLU
Output image size= 3*3
Output- 32*32*32

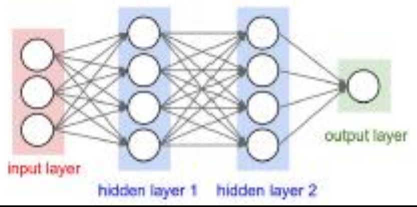
S2: Max Pooling
Padding used.
Pool size= 3*3
Output= 11*11*32

C3: Kernel filter used in convolution: 3*3, 64 filters are used.
Padding used.
Activation- ReLU
Output- 11*11*64

S4: Max Pooling
Padding used.
Pool size= 3*3
Output= 4*4*64

- Total learnable parameters = $n_{L-1} \times n_L$ + (n_L biases)

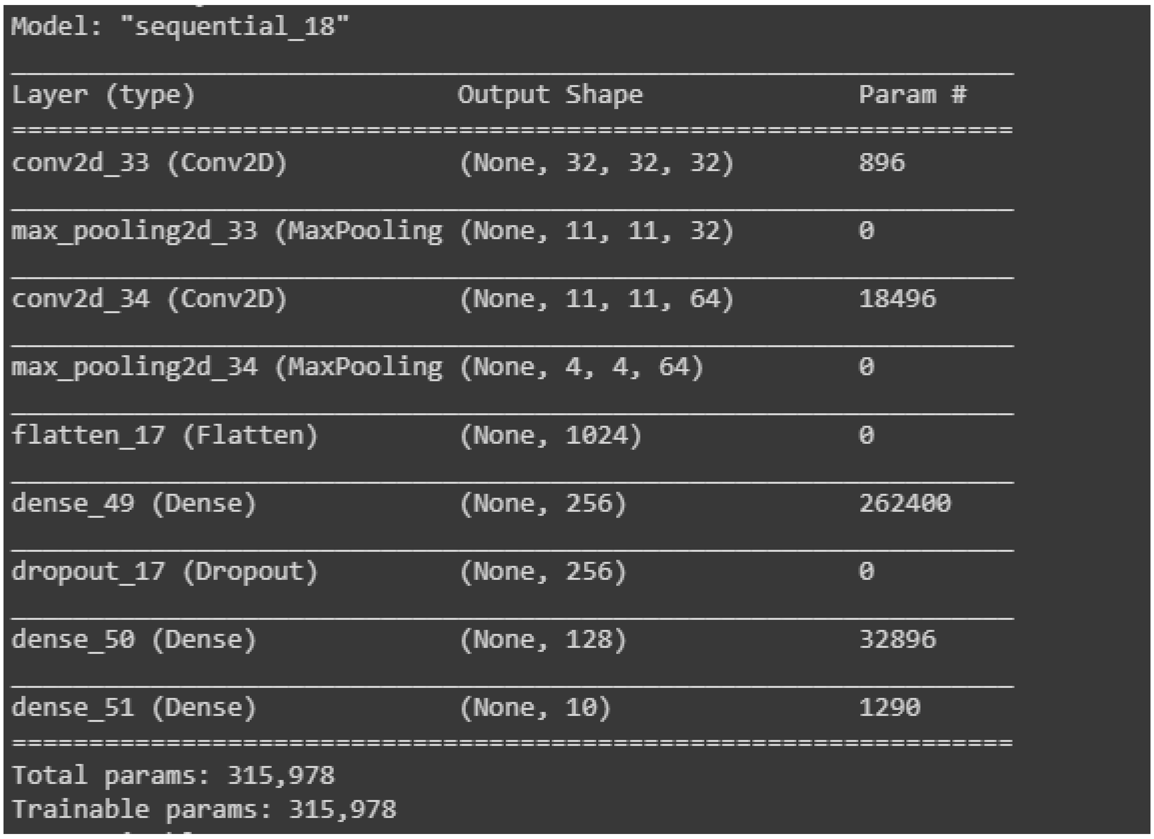
where n_{L-1} and n_L are the number of neurons in the (L-1)th and the Lth layer respectively



F5: 256 filters
Activation- ReLU

F6: 128 filters
Activation- ReLU

Output layer: Number of classes in the dataset= 10.
Activation- Softmax



```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 11, 11, 32)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_17 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_49 (Dense)             (None, 256)               262400
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_50 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_51 (Dense)             (None, 10)                1290
=================================================================
Total params: 315,978
Trainable params: 315,978
```

Fig. 3 Parameters in the architecture

<u>Reason choosing these parameters:</u>

Having done HW5 problem 1, I had a fair idea what each parameter does and the functionality of the same.

Firstly, I knew data augmentation was necessary which was not done by me in homework 5 thus increasing my accuracy here.

Adding, choosing the number of filters was trial and error as I knew more filters will make more trainable parameters which have potential to increase the accuracy but downfall of high model size. I have tried to keep the best number of filter and the kernel size.

Max Pooling was used in padding was set to avoid data loss around boundary and kernel of 3*3

Choosing the second convolutional layer, number of filters to be higher than 1st convolutional which is usually chosen, I did the same as well.

The second pooling layer parameter was chosen same as that of previous pooling layer.

The dense layer parameter was chosen in the power of 2 and decreasing thereby.

Dropout also helps in achieving higher performance which is why I have added the same. It is usually between 0 and 1, normally chosen as 0.1 To 0.4.

Finally, the dense layer using softmax with parameter 10 which is same as that of labels in the data.

Training Mechanism:

1. Data Augmentation: Convolutional Neural Network (CNN) is invariant to shift, scale, rotate, illumination, etc. This property is used in data augmentation. Data augmentation is used to randomly choose the images and make multiple copies of it using the original image by shifting, scaling, rotating, shearing, zooming, whitening, etc.

   In real word, we have very limited amount of data to access. So, whatever the data is, we need to utilize it fully. The given data can be made into a large amount of data by using data augmentation. Each image can have multiple copies with image looking different for the each one.

   The main reason to use data augmentation is to train the model better. Let say an inverted test image appeared in the model. This image should be correctly identified. To accommodate all the different test images, we make a model such that it correctly classifies all the varieties.

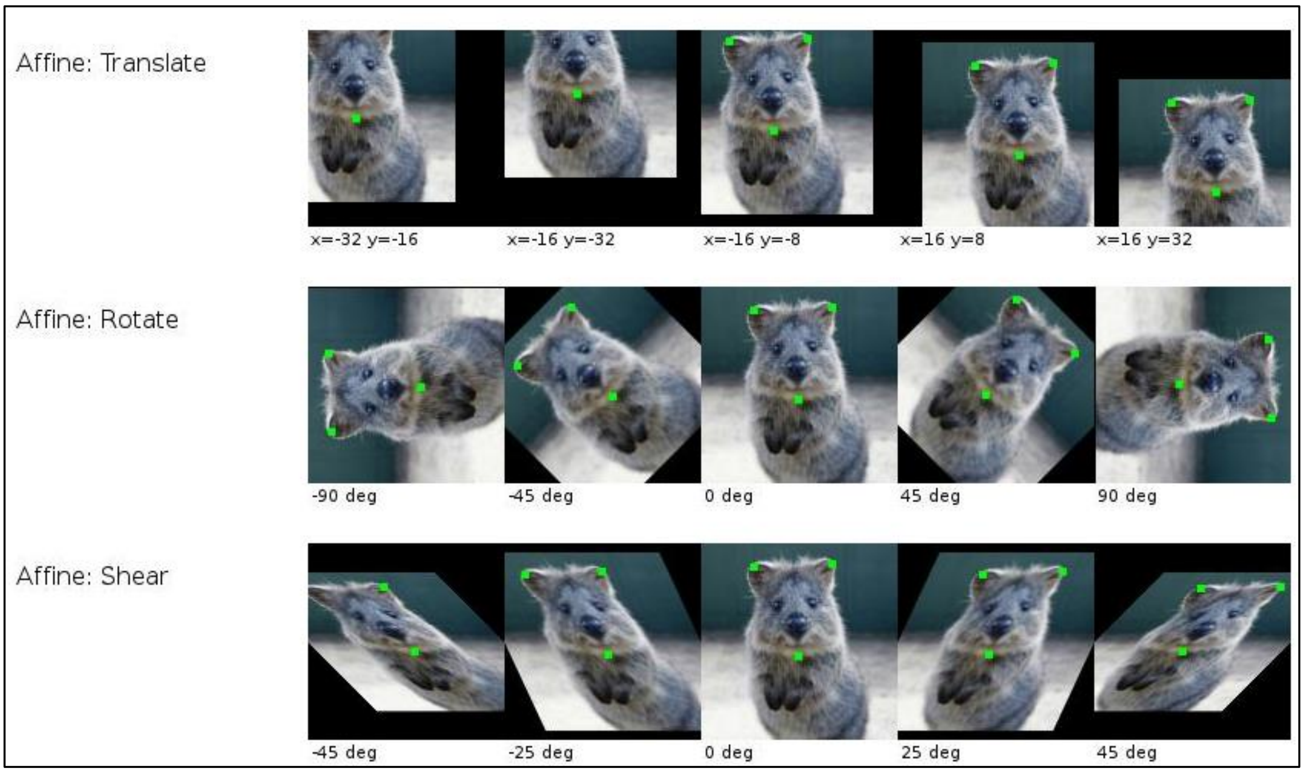


Fig. 4 Data Augmentation usage

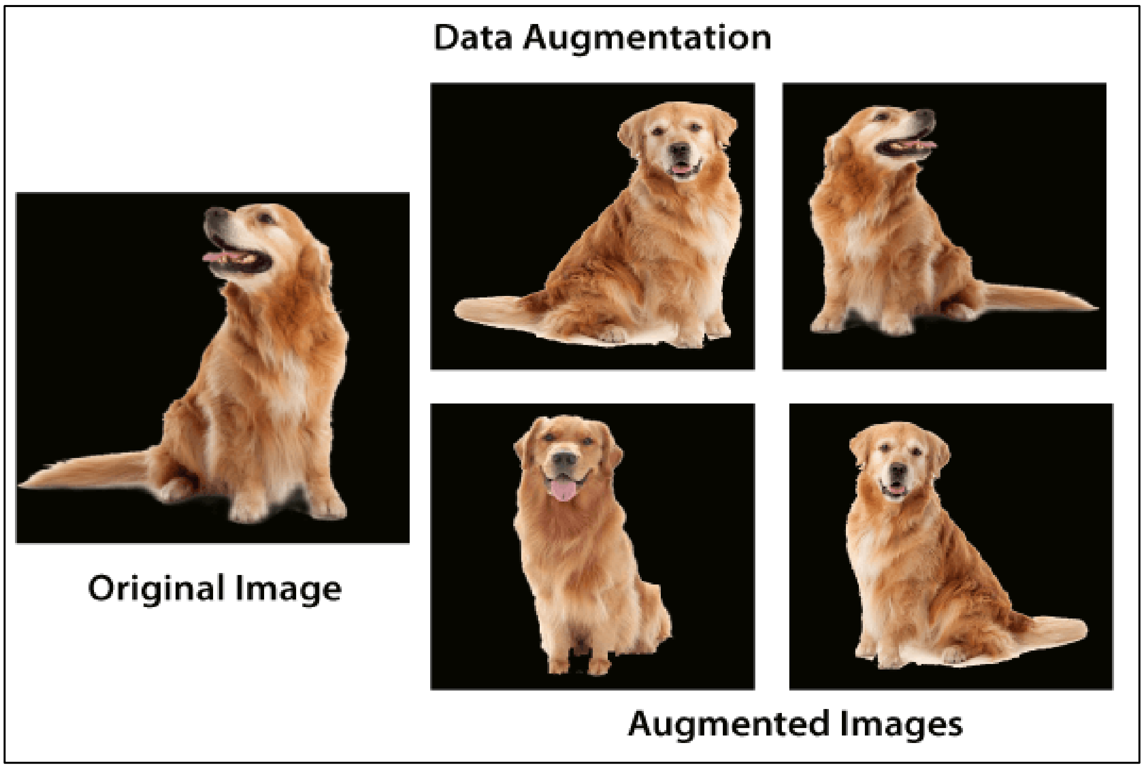


Fig. 5 Data Augmentation usage

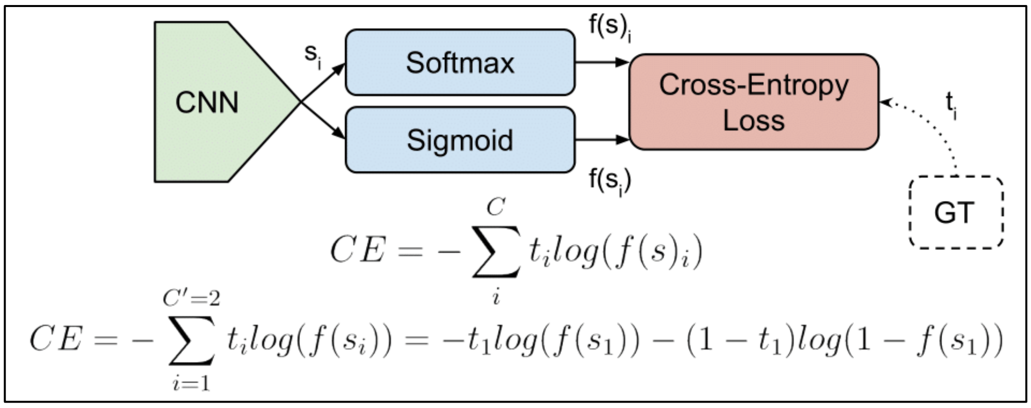
Parameters used in Data Augmentation:

a. Width Shift range= 0.1
b. Height Shift range= 0.1
c. Fill Mode= Nearest
d. Horizontal Flip= True
Many more parameters can be altered. I found these parameters to be best and got more performance accuracy using those.

2. <u>Loss function:</u> Cross Entropy Loss which is also called log loss is used as it measures the performance when the output is a probability value between 0 and 1. As we are using softmax function in the last layer which gives the output in 0 and 1, I have used cross entropy as the loss function.
Moreover, Cross entropy is a good loss function for the classification problem as it works on the principle minimizing the distance between two probability function, which are the predicted function and actual function.

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

Map inputs to probabilistic predictions
closer to ground-truth probabilities
Number of output =number of classes



$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Fig. 6 Cross Entropy

3. Optimizer: I have used 'Adam' optimizer which is supposed to give better results in most cases. SGD is also one which is used, but my accuracy was better in using Adam.
   The parameters in Adam are invariant to the rescaling of the gradient unlike SGD optimizer.

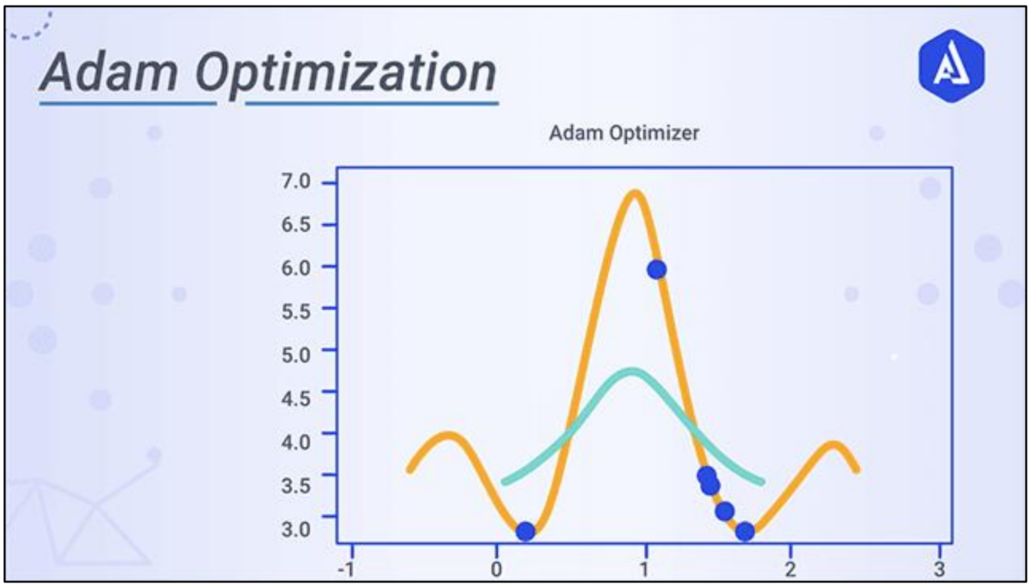


Fig. 7 Adam Optimizer working

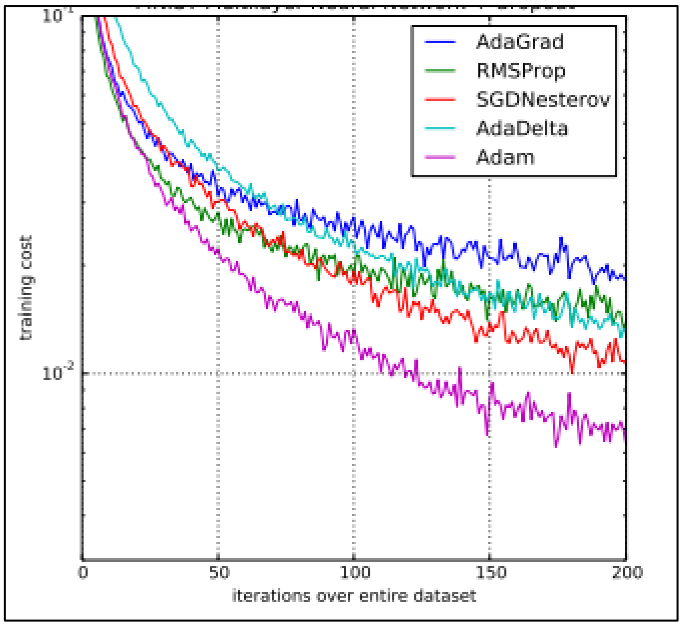


Fig. 8 Graphs of various optimizer vs cost

4. Batch Size:

   Batch Size is the number of training samples in one pass. Batch Size becomes important as the number specified are trained in one epoch. If it is not mentioned, all the training data would be trained in all the epochs. This will create a huge load on the memory. Generally, the smaller the batch size is better which is chosen in the powers of 2 majorly 32,64,128,256. The networks trained better using the mini batch sizes as the update of the parameters takes after every batch size. I have chosen batch size of 128 here as it gives more accuracy then the batch size of generally use size of 32.

5. Epochs:

   It is defined as the one forward and one backward pass for all the training data. It is a hyperparameter meaning the number of times the learning algorithm would work over the entire training data.

Algorithm:

1. Loading training and testing images from CIFAR10 dataset.
2. Training and testing labels converted to categorical type.
3. Pre-Processing the training data to get 0 mean and unit variance. This is necessary step as the data present maybe out of range.
4. Using sequential model and adding layers onto it.
5. Model includes 7 layers which are as follows: convolution, max pooling, convolution, max pooling, fully connected, fully connected, softmax layer. (Parameters given as per discussed above.)
6. Model compilation using categorical loss function.
7. Choosing best optimizer for the model. (Here- Adam)
8. Data Augmentation is used to make the model train better.
9. Training data passed through the model and fitting it into model.
10. Passing testing data through the model over specifying number of epochs.

11. Calculating test loss, training and testing accuracy.
12. Plotting train and test accuracy graphs.

3. RESULTS

Best Choice of Parameter

Epochs= 50
Batch Size= 128
Dropout=0.2
Optimizer= Adam

Model Parameter:	C1- 32 filters, kernel 3*3
	S1- Max Pooling, 3*3
	C2- 64 filters, kernel 3*3
	S2- Max Pooling, 3*3
	F1: 256 filter
	F2: 128 filters
	F3: 10 filters softmax

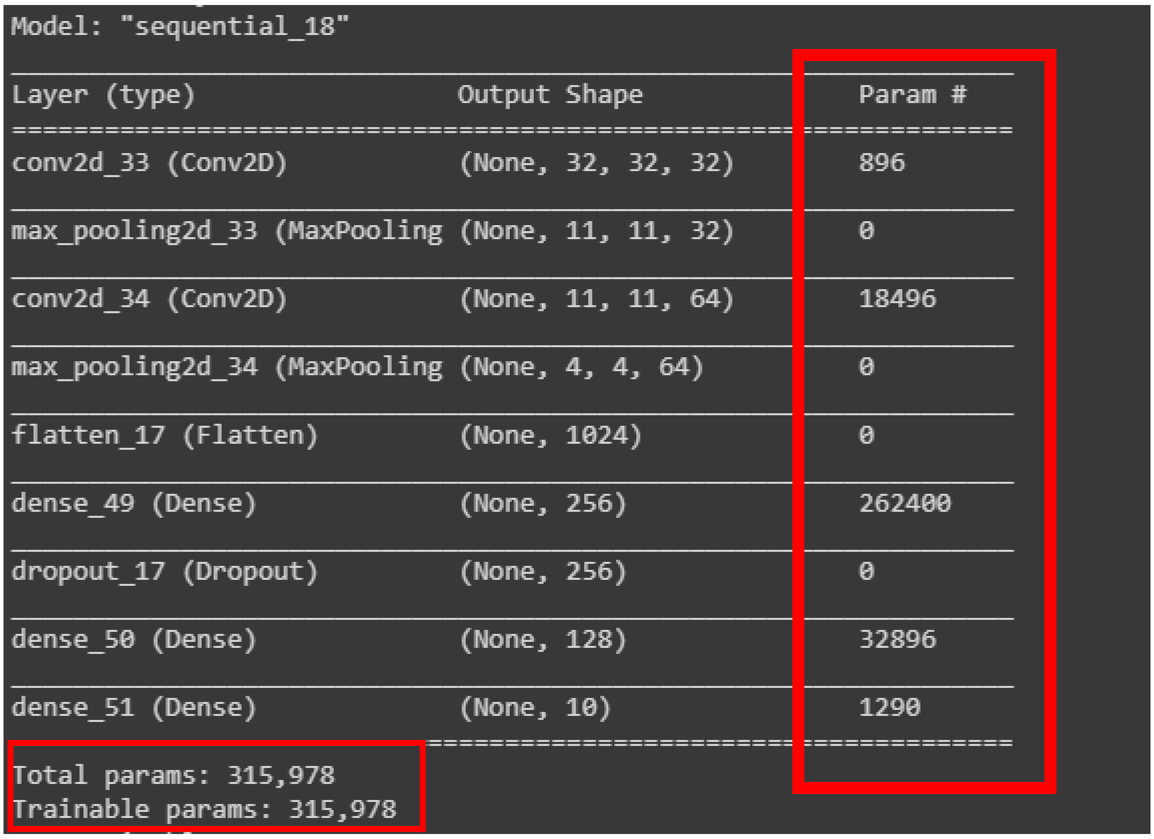
Training Accuracy= 0.8239
Training Loss= 0.4987

Testing Accuracy= 0.8041
Testing Loss= 0.6002

Model Size:

The model size should be small as possible. While trying various parameters, I realized the model size can go pretty high very quickly. To monitor it, the parameters while training the model become super important. I have selected those which I feel the total learnable parameters are enough. As seen from the below figure, it does give the total parameters after each layer in the model and also the total model size.

Total model size: 315,978



```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 11, 11, 32)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_17 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_49 (Dense)             (None, 256)               262400
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_50 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_51 (Dense)             (None, 10)                1290
=================================================================
Total params: 315,978
Trainable params: 315,978
```

Fig. 9 Highlighted parameter size

Model Time:

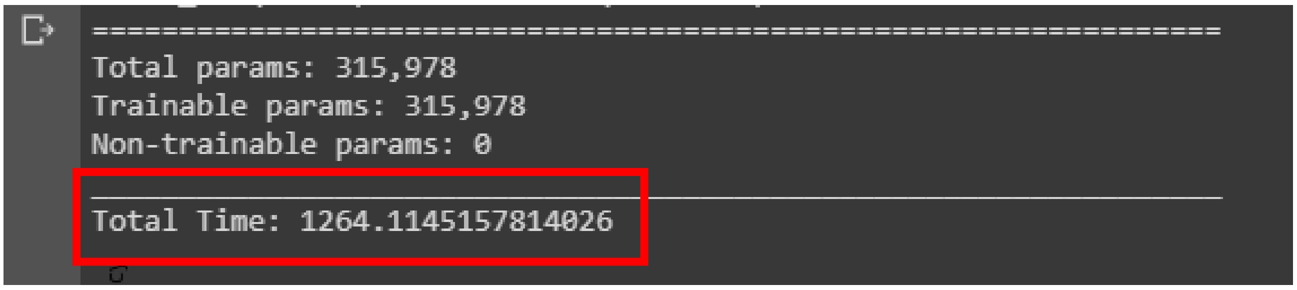
The model time can be divided into 2 forms: training time and the inference time. The training time is the time, which is used for the training data, while the inference time is the time to infer the predictions on test data using the pretrained model. Ideally, the training time should also be smaller, and a lot of factors depend on it like the CPU, GPU or the memory one system has.

As per my model run on GPU, I get each epoch runtime as 25 seconds for each epoch. I am running for 50 epochs which means 50*25= 1250 seconds= 20.83 mins.

So, my model training time = 20.83 mins

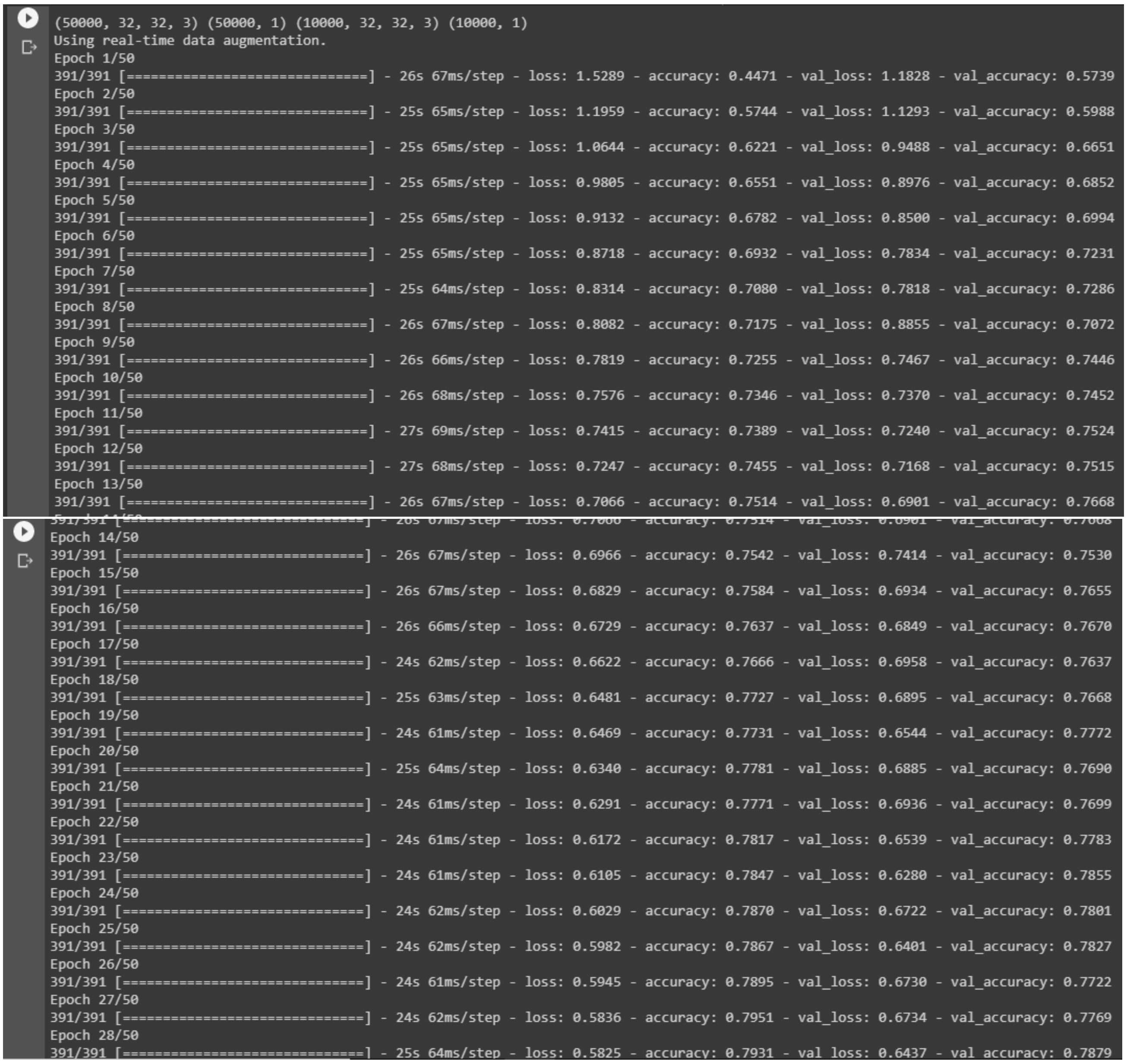
About inference time, the testing data here is of 10K images which hardly takes around 1-2 seconds to predict once model is trained.

The total time which includes the model training time, inference time, plotting, saving the model, etc. has been reported which is 1264.1145 = 21.06 mins. (Fig. 10)

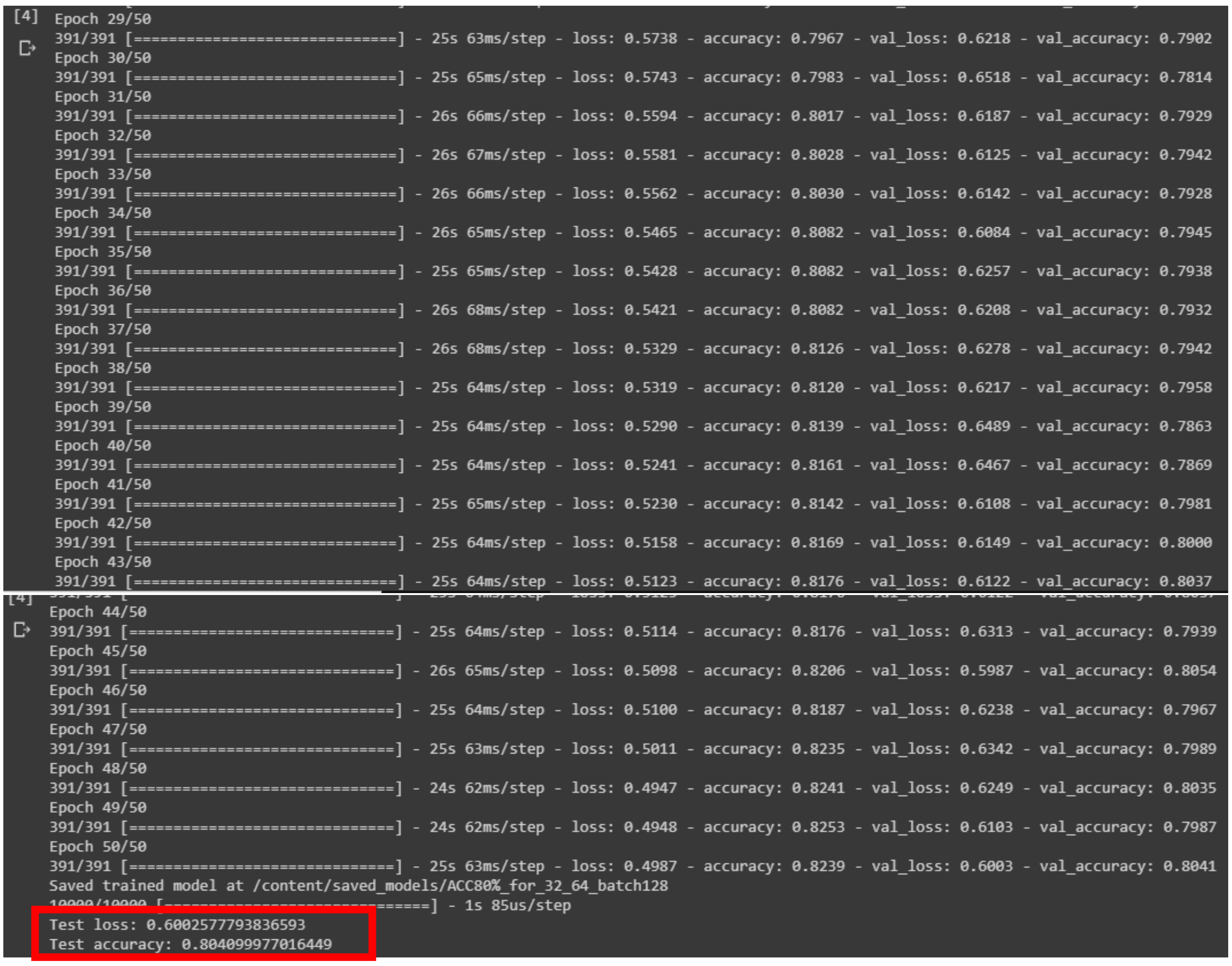


```
=================================================================
Total params: 315,978
Trainable params: 315,978
Non-trainable params: 0
_________________________________________________________________
Total Time: 1264.1145157814026
```

Fig. 10 Total time computed



```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
Using real-time data augmentation.
Epoch 1/50
391/391 [==============================] - 26s 67ms/step - loss: 1.5289 - accuracy: 0.4471 - val_loss: 1.1828 - val_accuracy: 0.5739
Epoch 2/50
391/391 [==============================] - 25s 65ms/step - loss: 1.1959 - accuracy: 0.5744 - val_loss: 1.1293 - val_accuracy: 0.5988
Epoch 3/50
391/391 [==============================] - 25s 65ms/step - loss: 1.0644 - accuracy: 0.6221 - val_loss: 0.9488 - val_accuracy: 0.6651
Epoch 4/50
391/391 [==============================] - 25s 65ms/step - loss: 0.9805 - accuracy: 0.6551 - val_loss: 0.8976 - val_accuracy: 0.6852
Epoch 5/50
391/391 [==============================] - 25s 65ms/step - loss: 0.9132 - accuracy: 0.6782 - val_loss: 0.8500 - val_accuracy: 0.6994
Epoch 6/50
391/391 [==============================] - 25s 65ms/step - loss: 0.8718 - accuracy: 0.6932 - val_loss: 0.7834 - val_accuracy: 0.7231
Epoch 7/50
391/391 [==============================] - 25s 64ms/step - loss: 0.8314 - accuracy: 0.7080 - val_loss: 0.7818 - val_accuracy: 0.7286
Epoch 8/50
391/391 [==============================] - 26s 67ms/step - loss: 0.8082 - accuracy: 0.7175 - val_loss: 0.8855 - val_accuracy: 0.7072
Epoch 9/50
391/391 [==============================] - 26s 66ms/step - loss: 0.7819 - accuracy: 0.7255 - val_loss: 0.7467 - val_accuracy: 0.7446
Epoch 10/50
391/391 [==============================] - 26s 68ms/step - loss: 0.7576 - accuracy: 0.7346 - val_loss: 0.7370 - val_accuracy: 0.7452
Epoch 11/50
391/391 [==============================] - 27s 69ms/step - loss: 0.7415 - accuracy: 0.7389 - val_loss: 0.7240 - val_accuracy: 0.7524
Epoch 12/50
391/391 [==============================] - 27s 68ms/step - loss: 0.7247 - accuracy: 0.7455 - val_loss: 0.7168 - val_accuracy: 0.7515
Epoch 13/50
391/391 [==============================] - 26s 67ms/step - loss: 0.7066 - accuracy: 0.7514 - val_loss: 0.6901 - val_accuracy: 0.7668
Epoch 14/50
391/391 [==============================] - 26s 67ms/step - loss: 0.6966 - accuracy: 0.7542 - val_loss: 0.7414 - val_accuracy: 0.7530
Epoch 15/50
391/391 [==============================] - 26s 67ms/step - loss: 0.6829 - accuracy: 0.7584 - val_loss: 0.6934 - val_accuracy: 0.7655
Epoch 16/50
391/391 [==============================] - 26s 66ms/step - loss: 0.6729 - accuracy: 0.7637 - val_loss: 0.6849 - val_accuracy: 0.7670
Epoch 17/50
391/391 [==============================] - 24s 62ms/step - loss: 0.6622 - accuracy: 0.7666 - val_loss: 0.6958 - val_accuracy: 0.7637
Epoch 18/50
391/391 [==============================] - 25s 63ms/step - loss: 0.6481 - accuracy: 0.7727 - val_loss: 0.6895 - val_accuracy: 0.7668
Epoch 19/50
391/391 [==============================] - 24s 61ms/step - loss: 0.6469 - accuracy: 0.7731 - val_loss: 0.6544 - val_accuracy: 0.7772
Epoch 20/50
391/391 [==============================] - 25s 64ms/step - loss: 0.6340 - accuracy: 0.7781 - val_loss: 0.6885 - val_accuracy: 0.7690
Epoch 21/50
391/391 [==============================] - 24s 61ms/step - loss: 0.6291 - accuracy: 0.7771 - val_loss: 0.6936 - val_accuracy: 0.7699
Epoch 22/50
391/391 [==============================] - 24s 61ms/step - loss: 0.6172 - accuracy: 0.7817 - val_loss: 0.6539 - val_accuracy: 0.7783
Epoch 23/50
391/391 [==============================] - 24s 61ms/step - loss: 0.6105 - accuracy: 0.7847 - val_loss: 0.6280 - val_accuracy: 0.7855
Epoch 24/50
391/391 [==============================] - 24s 62ms/step - loss: 0.6029 - accuracy: 0.7870 - val_loss: 0.6722 - val_accuracy: 0.7801
Epoch 25/50
391/391 [==============================] - 24s 62ms/step - loss: 0.5982 - accuracy: 0.7867 - val_loss: 0.6401 - val_accuracy: 0.7827
Epoch 26/50
391/391 [==============================] - 24s 61ms/step - loss: 0.5945 - accuracy: 0.7895 - val_loss: 0.6730 - val_accuracy: 0.7722
Epoch 27/50
391/391 [==============================] - 24s 62ms/step - loss: 0.5836 - accuracy: 0.7951 - val_loss: 0.6734 - val_accuracy: 0.7769
Epoch 28/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5825 - accuracy: 0.7931 - val_loss: 0.6437 - val_accuracy: 0.7879
```



```
[4] Epoch 29/50
391/391 [==============================] - 25s 63ms/step - loss: 0.5738 - accuracy: 0.7967 - val_loss: 0.6218 - val_accuracy: 0.7902
Epoch 30/50
391/391 [==============================] - 25s 65ms/step - loss: 0.5743 - accuracy: 0.7983 - val_loss: 0.6518 - val_accuracy: 0.7814
Epoch 31/50
391/391 [==============================] - 26s 66ms/step - loss: 0.5594 - accuracy: 0.8017 - val_loss: 0.6187 - val_accuracy: 0.7929
Epoch 32/50
391/391 [==============================] - 26s 67ms/step - loss: 0.5581 - accuracy: 0.8028 - val_loss: 0.6125 - val_accuracy: 0.7942
Epoch 33/50
391/391 [==============================] - 26s 66ms/step - loss: 0.5562 - accuracy: 0.8030 - val_loss: 0.6142 - val_accuracy: 0.7928
Epoch 34/50
391/391 [==============================] - 26s 65ms/step - loss: 0.5465 - accuracy: 0.8082 - val_loss: 0.6084 - val_accuracy: 0.7945
Epoch 35/50
391/391 [==============================] - 25s 65ms/step - loss: 0.5428 - accuracy: 0.8082 - val_loss: 0.6257 - val_accuracy: 0.7938
Epoch 36/50
391/391 [==============================] - 26s 68ms/step - loss: 0.5421 - accuracy: 0.8082 - val_loss: 0.6208 - val_accuracy: 0.7932
Epoch 37/50
391/391 [==============================] - 26s 68ms/step - loss: 0.5329 - accuracy: 0.8126 - val_loss: 0.6278 - val_accuracy: 0.7942
Epoch 38/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5319 - accuracy: 0.8120 - val_loss: 0.6217 - val_accuracy: 0.7958
Epoch 39/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5290 - accuracy: 0.8139 - val_loss: 0.6489 - val_accuracy: 0.7863
Epoch 40/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5241 - accuracy: 0.8161 - val_loss: 0.6467 - val_accuracy: 0.7869
Epoch 41/50
391/391 [==============================] - 25s 65ms/step - loss: 0.5230 - accuracy: 0.8142 - val_loss: 0.6108 - val_accuracy: 0.7981
Epoch 42/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5158 - accuracy: 0.8169 - val_loss: 0.6149 - val_accuracy: 0.8000
Epoch 43/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5123 - accuracy: 0.8176 - val_loss: 0.6122 - val_accuracy: 0.8037
Epoch 44/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5114 - accuracy: 0.8176 - val_loss: 0.6313 - val_accuracy: 0.7939
Epoch 45/50
391/391 [==============================] - 26s 65ms/step - loss: 0.5098 - accuracy: 0.8206 - val_loss: 0.5987 - val_accuracy: 0.8054
Epoch 46/50
391/391 [==============================] - 25s 64ms/step - loss: 0.5100 - accuracy: 0.8187 - val_loss: 0.6238 - val_accuracy: 0.7967
Epoch 47/50
391/391 [==============================] - 25s 63ms/step - loss: 0.5011 - accuracy: 0.8235 - val_loss: 0.6342 - val_accuracy: 0.7989
Epoch 48/50
391/391 [==============================] - 24s 62ms/step - loss: 0.4947 - accuracy: 0.8241 - val_loss: 0.6249 - val_accuracy: 0.8035
Epoch 49/50
391/391 [==============================] - 24s 62ms/step - loss: 0.4948 - accuracy: 0.8253 - val_loss: 0.6103 - val_accuracy: 0.7987
Epoch 50/50
391/391 [==============================] - 25s 63ms/step - loss: 0.4987 - accuracy: 0.8239 - val_loss: 0.6003 - val_accuracy: 0.8041
Saved trained model at /content/saved_models/ACC80%_for_32_64_batch128
10000/10000 [==============================] - 1s 85us/step
Test loss: 0.6002577793836593
Test accuracy: 0.804099977016449
```

Fig. 11 Each Epoch time and accuracy

<u>Graph:</u>

The model accuracy vs no of epochs and model loss vs no of epochs graph for the modified model is as shown below. It can be inferred that the model is neither underfitting nor overfitting.

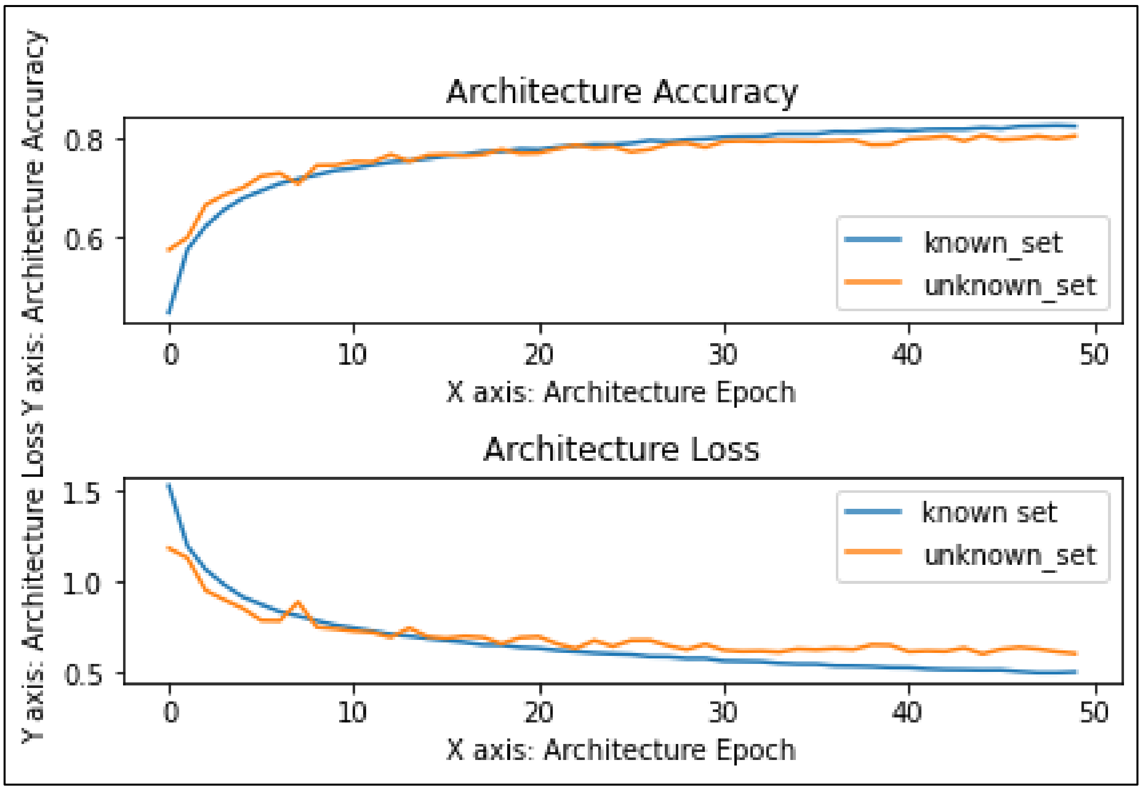
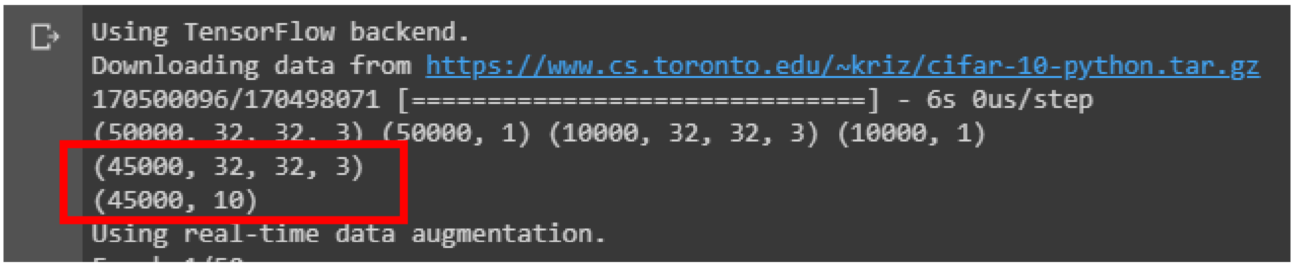


Fig. 12 Accuracy vs epoch and Loss vs Epoch graph

Dropping random train samples:

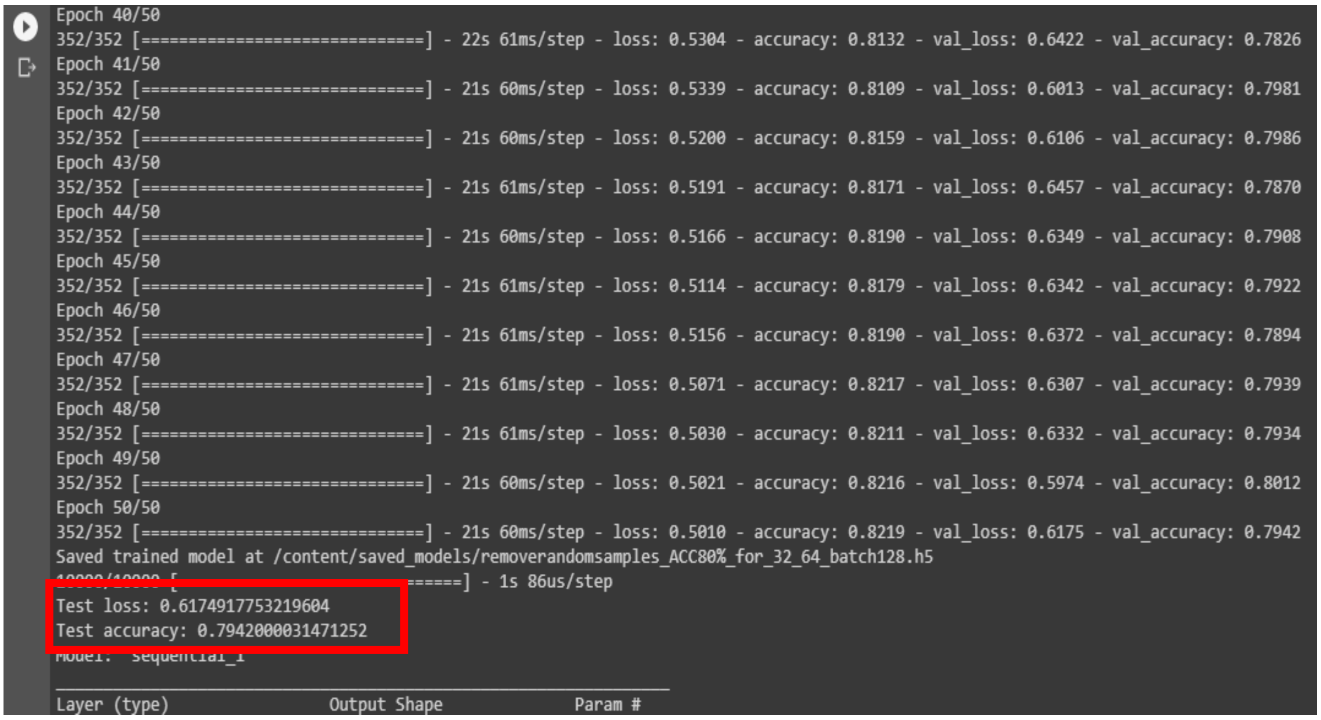
Case 1: 45K training images (5K images dropped)



```
Using TensorFlow backend.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 6s 0us/step
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
(45000, 32, 32, 3)
(45000, 10)
Using real-time data augmentation.
```

Fig. 13 Size of new train data after dropping 5K images

Training Accuracy: 82.19
Testing Accuracy: 79.42



```
Epoch 40/50
352/352 [==============================] - 22s 61ms/step - loss: 0.5304 - accuracy: 0.8132 - val_loss: 0.6422 - val_accuracy: 0.7826
Epoch 41/50
352/352 [==============================] - 21s 60ms/step - loss: 0.5339 - accuracy: 0.8109 - val_loss: 0.6013 - val_accuracy: 0.7981
Epoch 42/50
352/352 [==============================] - 21s 60ms/step - loss: 0.5200 - accuracy: 0.8159 - val_loss: 0.6106 - val_accuracy: 0.7986
Epoch 43/50
352/352 [==============================] - 21s 61ms/step - loss: 0.5191 - accuracy: 0.8171 - val_loss: 0.6457 - val_accuracy: 0.7870
Epoch 44/50
352/352 [==============================] - 21s 60ms/step - loss: 0.5166 - accuracy: 0.8190 - val_loss: 0.6349 - val_accuracy: 0.7908
Epoch 45/50
352/352 [==============================] - 21s 61ms/step - loss: 0.5114 - accuracy: 0.8179 - val_loss: 0.6342 - val_accuracy: 0.7922
Epoch 46/50
352/352 [==============================] - 21s 61ms/step - loss: 0.5156 - accuracy: 0.8190 - val_loss: 0.6372 - val_accuracy: 0.7894
Epoch 47/50
352/352 [==============================] - 21s 61ms/step - loss: 0.5071 - accuracy: 0.8217 - val_loss: 0.6307 - val_accuracy: 0.7939
Epoch 48/50
352/352 [==============================] - 21s 61ms/step - loss: 0.5030 - accuracy: 0.8211 - val_loss: 0.6332 - val_accuracy: 0.7934
Epoch 49/50
352/352 [==============================] - 21s 60ms/step - loss: 0.5021 - accuracy: 0.8216 - val_loss: 0.5974 - val_accuracy: 0.8012
Epoch 50/50
352/352 [==============================] - 21s 60ms/step - loss: 0.5010 - accuracy: 0.8219 - val_loss: 0.6175 - val_accuracy: 0.7942
Saved trained model at /content/saved_models/removerandomsamples_ACC80%_for_32_64_batch128.h5
10000/10000 [                          ======] - 1s 86us/step
Test loss: 0.6174917753219604
Test accuracy: 0.7942000031471252
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

Fig. 14 Accuracy after dropping 5K images

Graph:

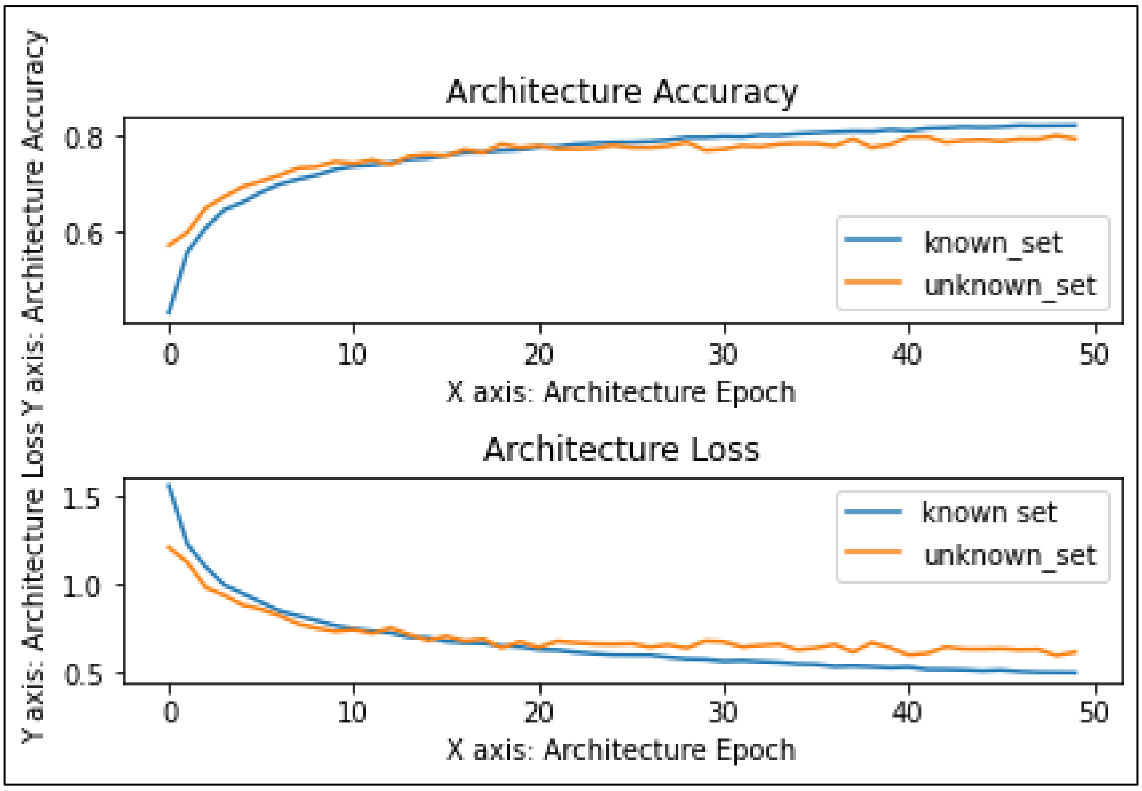
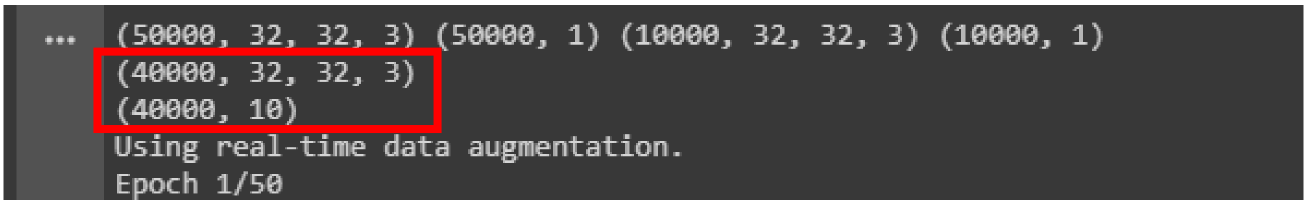


Fig. 15 Accuracy vs epoch and Loss vs Epoch graph after dropping 5K images

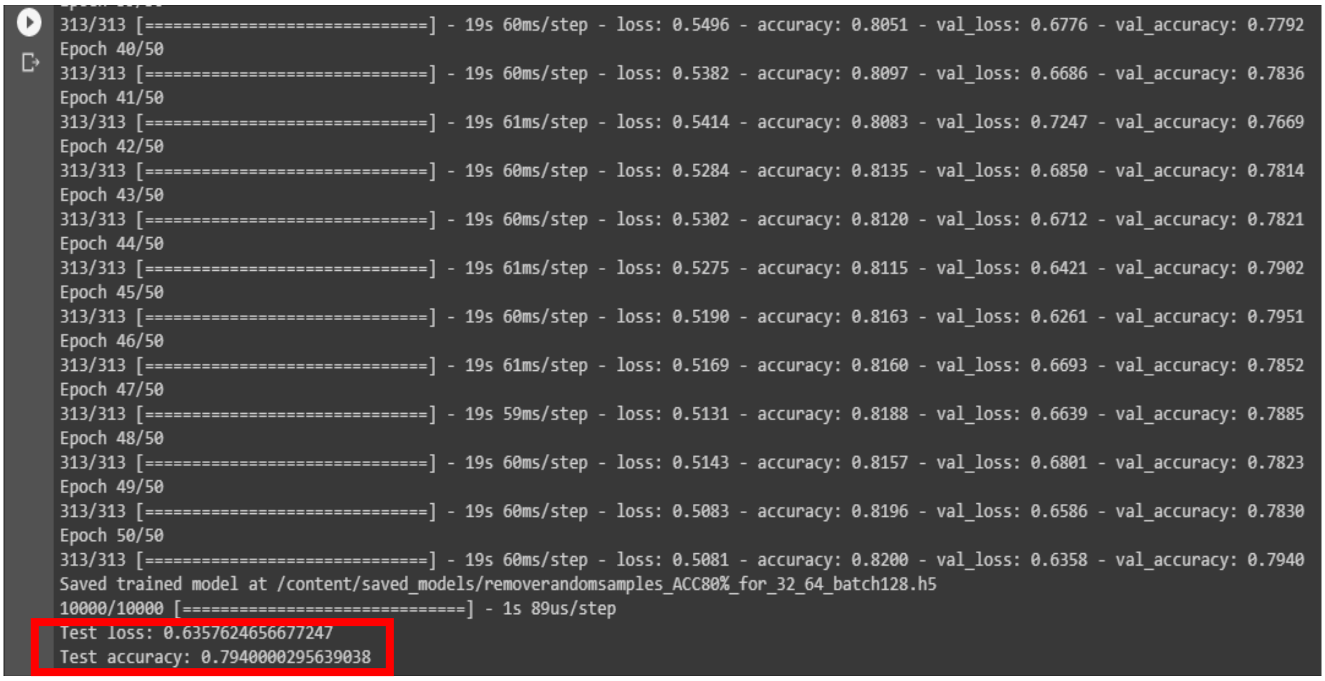
Case 2: 40K training images (10K images dropped)



```
(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)
(40000, 32, 32, 3)
(40000, 10)
Using real-time data augmentation.
Epoch 1/50
```

Fig. 16 Size of new train data after dropping 10K images

Training Accuracy: 82.00
Testing Accuracy: 79.40



```
313/313 [==============================] - 19s 60ms/step - loss: 0.5496 - accuracy: 0.8051 - val_loss: 0.6776 - val_accuracy: 0.7792
Epoch 40/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5382 - accuracy: 0.8097 - val_loss: 0.6686 - val_accuracy: 0.7836
Epoch 41/50
313/313 [==============================] - 19s 61ms/step - loss: 0.5414 - accuracy: 0.8083 - val_loss: 0.7247 - val_accuracy: 0.7669
Epoch 42/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5284 - accuracy: 0.8135 - val_loss: 0.6850 - val_accuracy: 0.7814
Epoch 43/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5302 - accuracy: 0.8120 - val_loss: 0.6712 - val_accuracy: 0.7821
Epoch 44/50
313/313 [==============================] - 19s 61ms/step - loss: 0.5275 - accuracy: 0.8115 - val_loss: 0.6421 - val_accuracy: 0.7902
Epoch 45/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5190 - accuracy: 0.8163 - val_loss: 0.6261 - val_accuracy: 0.7951
Epoch 46/50
313/313 [==============================] - 19s 61ms/step - loss: 0.5169 - accuracy: 0.8160 - val_loss: 0.6693 - val_accuracy: 0.7852
Epoch 47/50
313/313 [==============================] - 19s 59ms/step - loss: 0.5131 - accuracy: 0.8188 - val_loss: 0.6639 - val_accuracy: 0.7885
Epoch 48/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5143 - accuracy: 0.8157 - val_loss: 0.6801 - val_accuracy: 0.7823
Epoch 49/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5083 - accuracy: 0.8196 - val_loss: 0.6586 - val_accuracy: 0.7830
Epoch 50/50
313/313 [==============================] - 19s 60ms/step - loss: 0.5081 - accuracy: 0.8200 - val_loss: 0.6358 - val_accuracy: 0.7940
Saved trained model at /content/saved_models/removerandomsamples_ACC80%_for_32_64_batch128.h5
10000/10000 [==============================] - 1s 89us/step
Test loss: 0.6357624656677247
Test accuracy: 0.7940000295639038
```

Fig. 17 Accuracy after dropping 10K images

Graph:

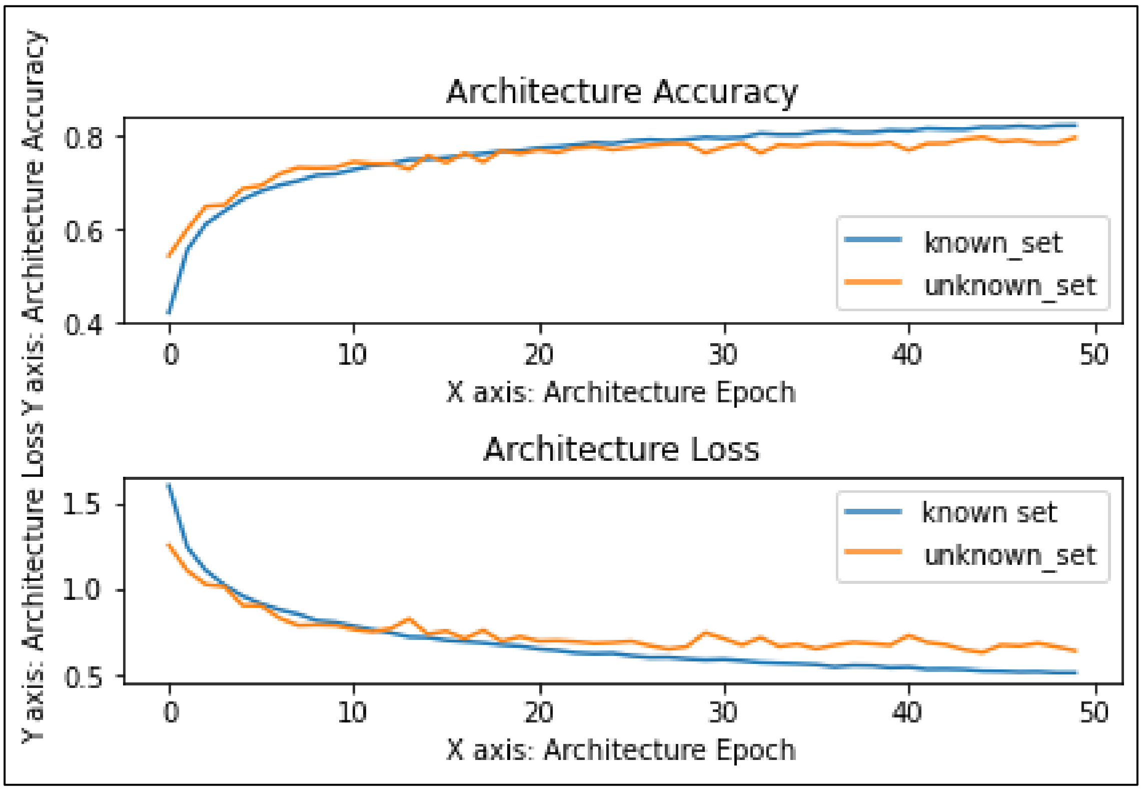


Fig. 18 Accuracy vs epoch and Loss vs Epoch graph
after dropping 10K images

4. DISCUSSION

a. Performance Improvement from the result of homework 5 problem 1B:
The parameters and hyperparameters in the homework 5 problem 1B were given which weren't giving the high accuracy as of this one.

Parameters	**Problem 1B (HW5)**	**Current Model Updated**
Batch Size	16	128
Pre-Processing	Yes	Yes
Data Augmentation	No	Yes
Optimizer	SGD	Adam
1st Conv layer	6 filters, 5*5 kernel	32 filters, 3*3 kernel
2nd Conv layer	16 filters, 5*5 kernel	64 filters, 3*3 kernel
1st Max Pooling	Pool size: 2*2	Pool size: 3*3
2nd Max Pooling	Pool size: 2*2	Pool size: 3*3
Padding	Valid (No Pad)	Same (Pad)
1st Dense layer	128	256
2nd Dense layer	84	128
Dropout	0.3	0.2
Last Dense layer	10	10
Epochs	20	50

Table 1. Comparison of models

Problem 1B Result:
Training Accuracy: 77.13
Testing Accuracy: 65.16

Current Model:
Training Accuracy: 82.39
Testing Accuracy: 80.41

As already discussed, the current model has more trainable parameters which results in higher accuracy then the previous one. This is done by increasing the filter size in both convolution layers.
Data Augmentation was added to the current model which helps in better performance as well.
The dense layer filters size has been increased to accommodate more features, same with the padding around the layers.
Batch Size has been increased to train more data in one pass.
All these collectively effect into the better training and testing accuracy for the CIFAR10 dataset.

b. Degradation when randomly drop train data:
We are training all 50K images in the training data while making the model. But as we drop some random images from the training data, we see the degradation of the training and testing accuracy.
This is because we don't have much data present in the network. It is always better to train the model with as much data one can provide. The model needs to be trained perfectly which is not possible if we give less training data as the input.
I have used 45K and 40K images as my trial for randomly dropping images which means I have dropped 5K and 10K respectively and trained the model and noted the results.
It can be seen in the result section that the training and testing accuracy in both cases have been decreased. The decrease has been 1-2% from the original model. If more samples are reduced the accuracy would drop down more. (Fig. 14 and 17) The graphs for Accuracy vs Epoch and Loss vs Epoch for both cases have also been plotted in which over entire epochs the accuracy can be visualized. (Fig. 15 and 18)

REFERENCES:

1. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

2. http://keras.io/