Multilayer Perceptron (MLP) on CIFAR-10

A multilayer perceptron is the basic kind of a deep neural network an here we have altered various parameters of the CIFAR-10 dataset to come up with our best model by providing different observations and provided further suggestions on how the accuracy of the model can be increased.

Observation 1 –

Best Optimizer –

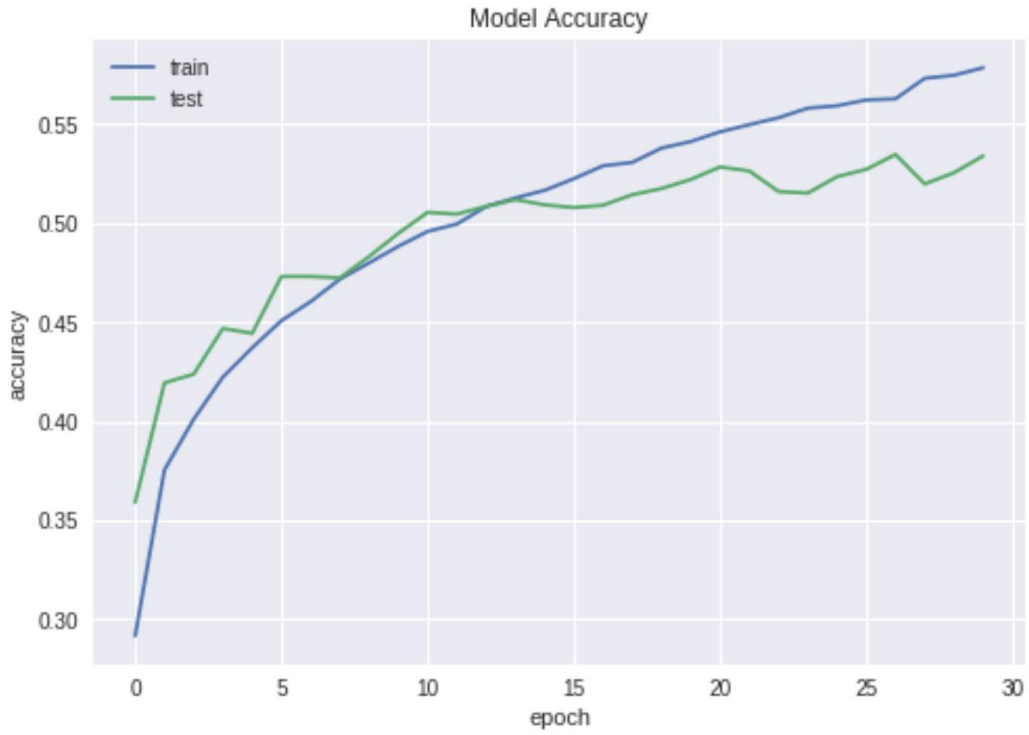
I tried RMSprop , Adamax , Adam, SGD, Adadelta for the models with base config -

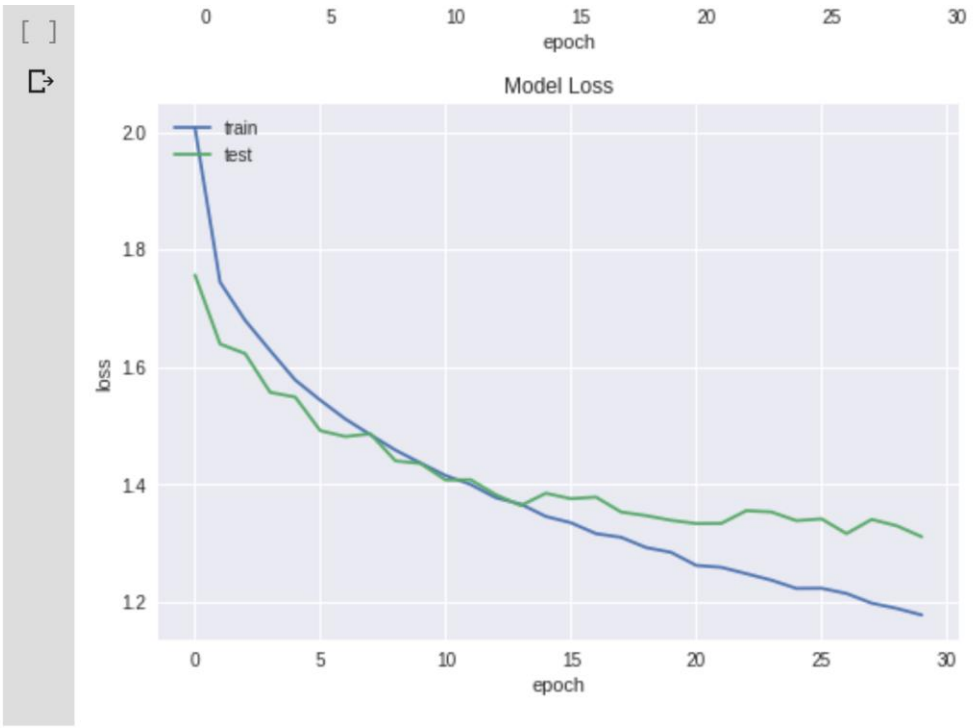
Batch Size – 128
Epochs - 30
Optimizer – Adamax() - best
Full Connection Layer – 512, 512
Dropout – 0.2

Accuracy:
From these models, I found out that the best optimizer for this dataset is Adamax and Adam.

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)


#building the model for Multi Layer Perceptron
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(3072,)
model.add(Dropout(0.1))
model.add(Dense(512, activation='relu', input_shape=(3072,)
model.add(Dropout(0.1))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=adamax(),
              metrics=['accuracy'])
```

```
Test Loss:  1.3108868904113768
Test Accuracy 0.534
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





So, the best Optimizer is Adamax as per the above experiments .

Observation 2 –

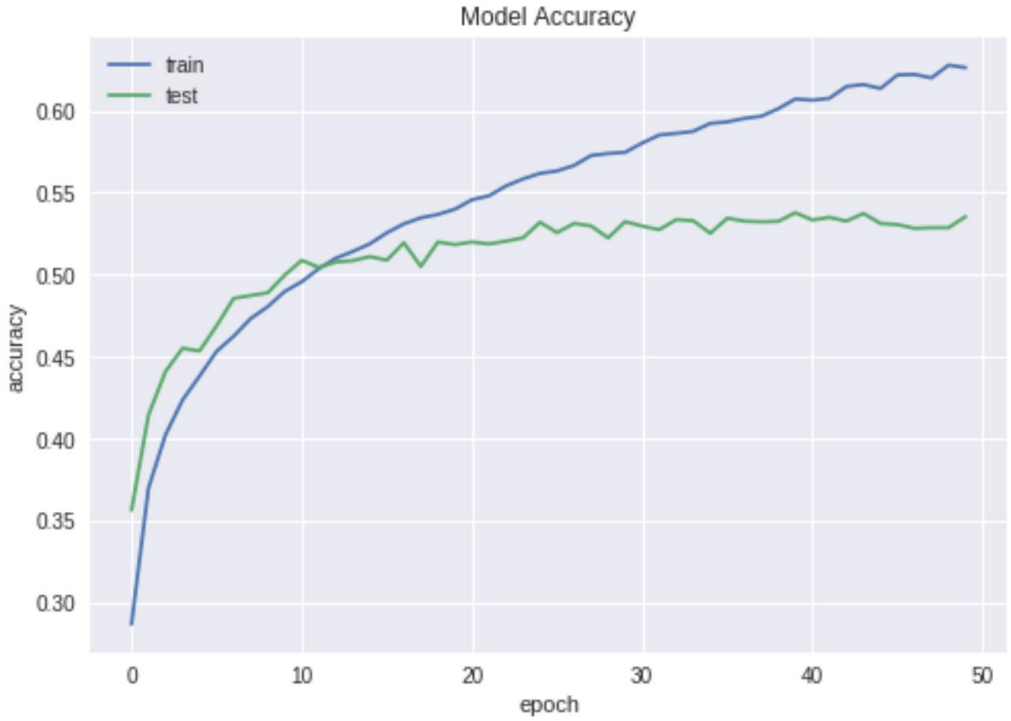
Best Number of epochs –

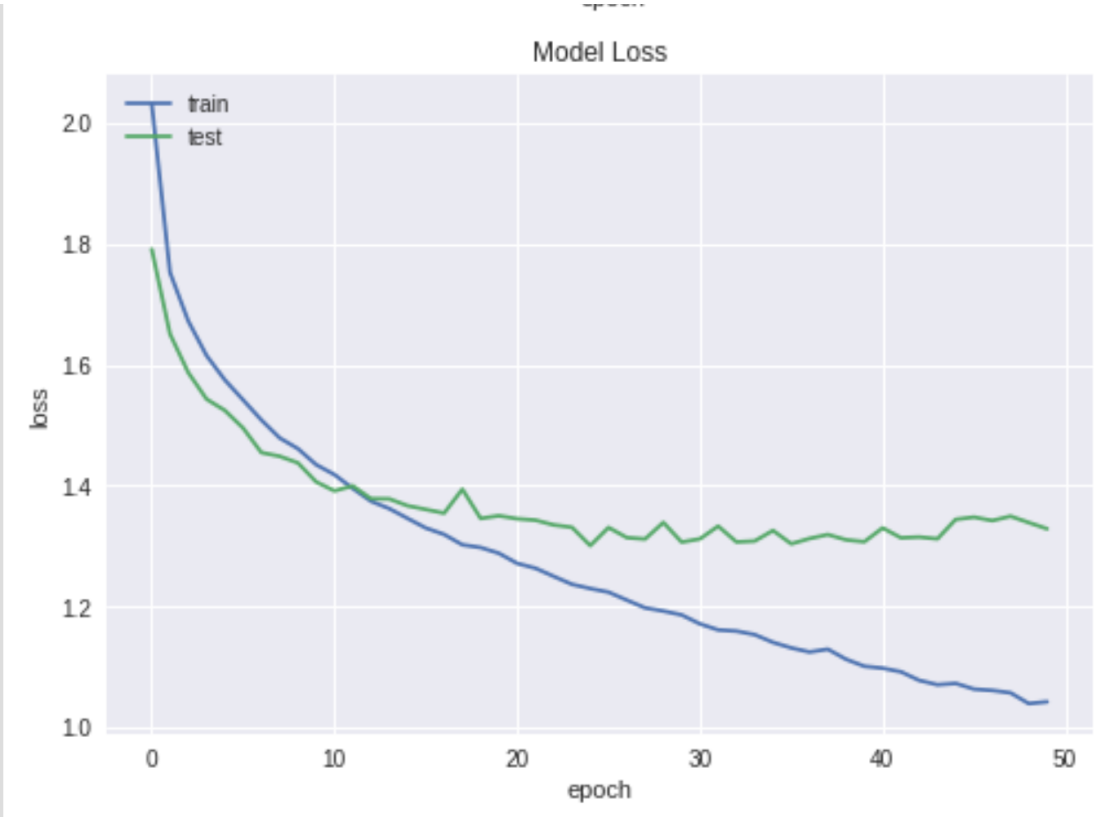
I tried 10, 25, 50, 60, 100 for the models with base config and the best Optimizer found in the last step -

Batch Size – 128
Epochs – 50 - best
Optimizer – Adamax() -
Full Connection Layer – 512, 512
Dropout – 0.2

Accuracy:
From these models, I found out that the best number of epoch for this dataset with avoiding the overfitting is 50.

```
[ ]  Epoch 50/50
     50000/50000 [==============================] - 22s 440us/step - loss:
     Test Loss:  1.3284688913345337
     Test Accuracy 0.5355
     dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Observation 3–

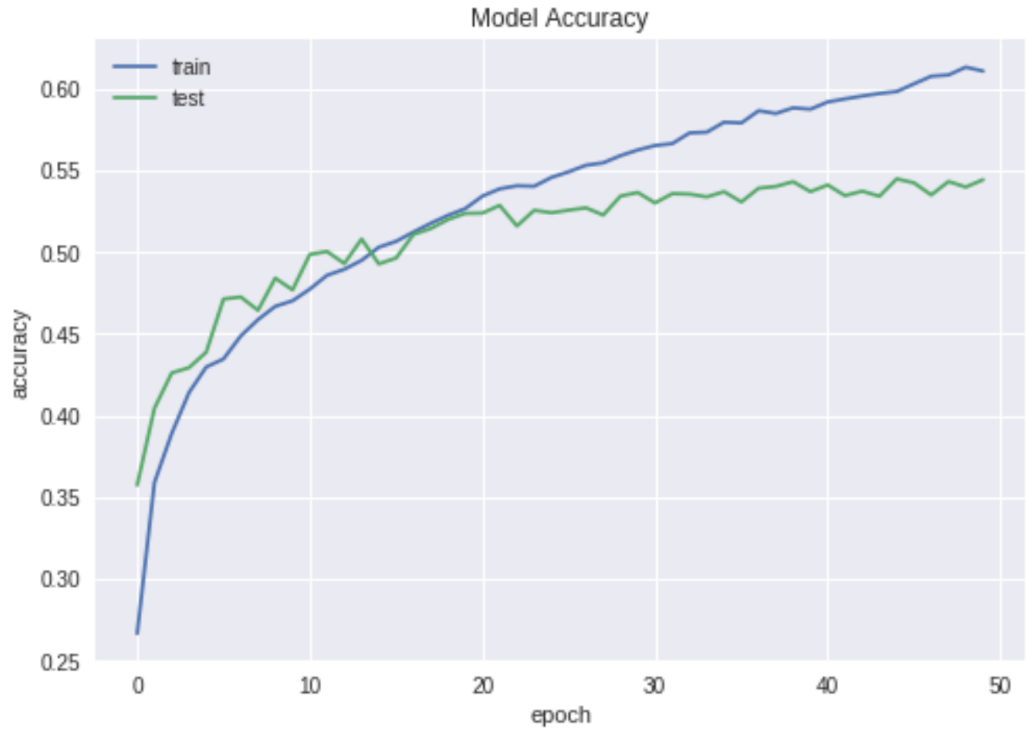
Best Number of Batch Size –

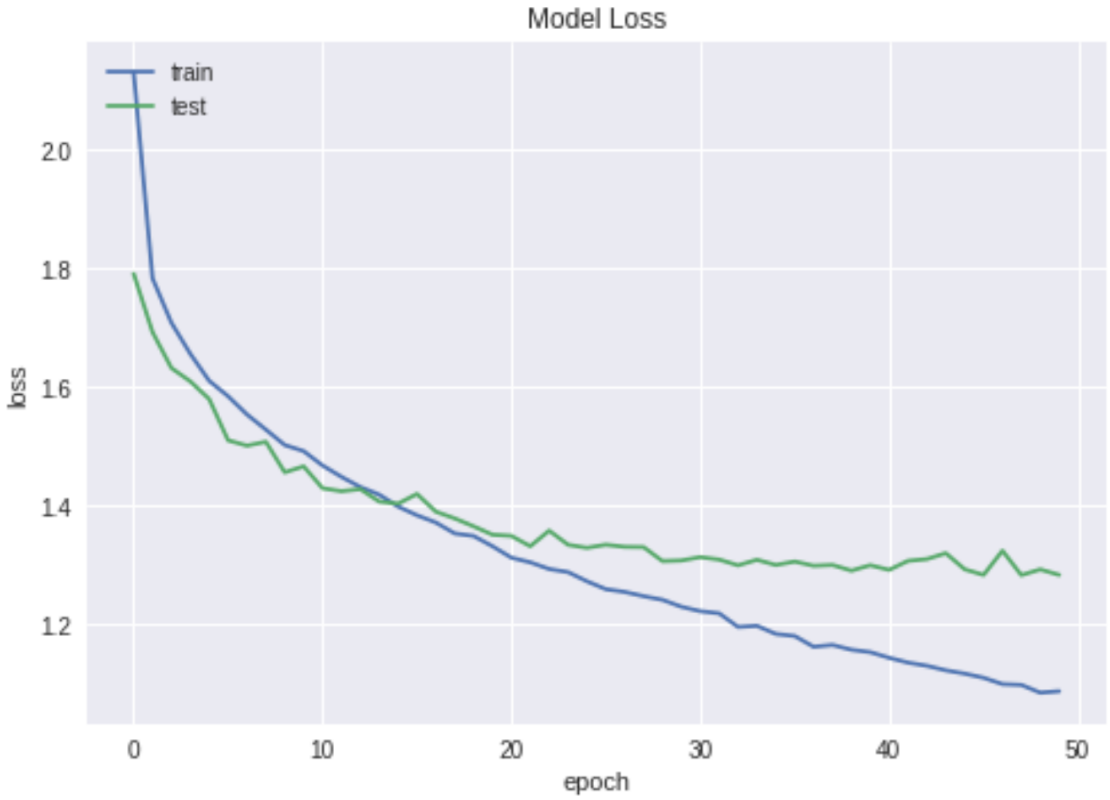
I tried 16, 32, 64, 128, 256, 512 for the models with base config and the best Optimizer found in the last step -

Batch Size – 256 – best
Epochs – 50
Optimizer – Adamax() -
Full Connection Layer – 512, 512
Dropout – 0.2

Accuracy:
From these models, I found out that the best Batch_size for 50 epochs and Adamax is 256.

```
[ ] Test Loss:  1.283493635559082
    Test Accuracy 0.5443
    dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Observation 4 –

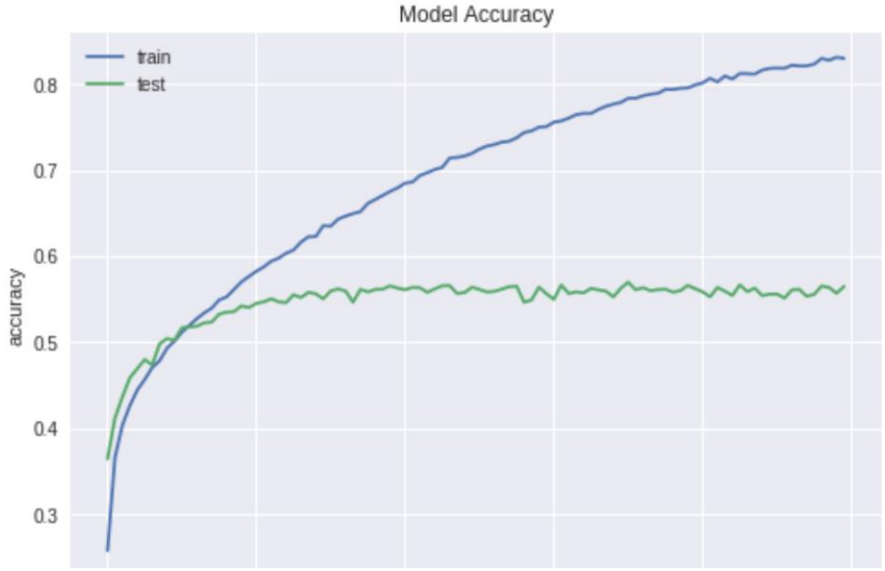
Best Network Configuration –

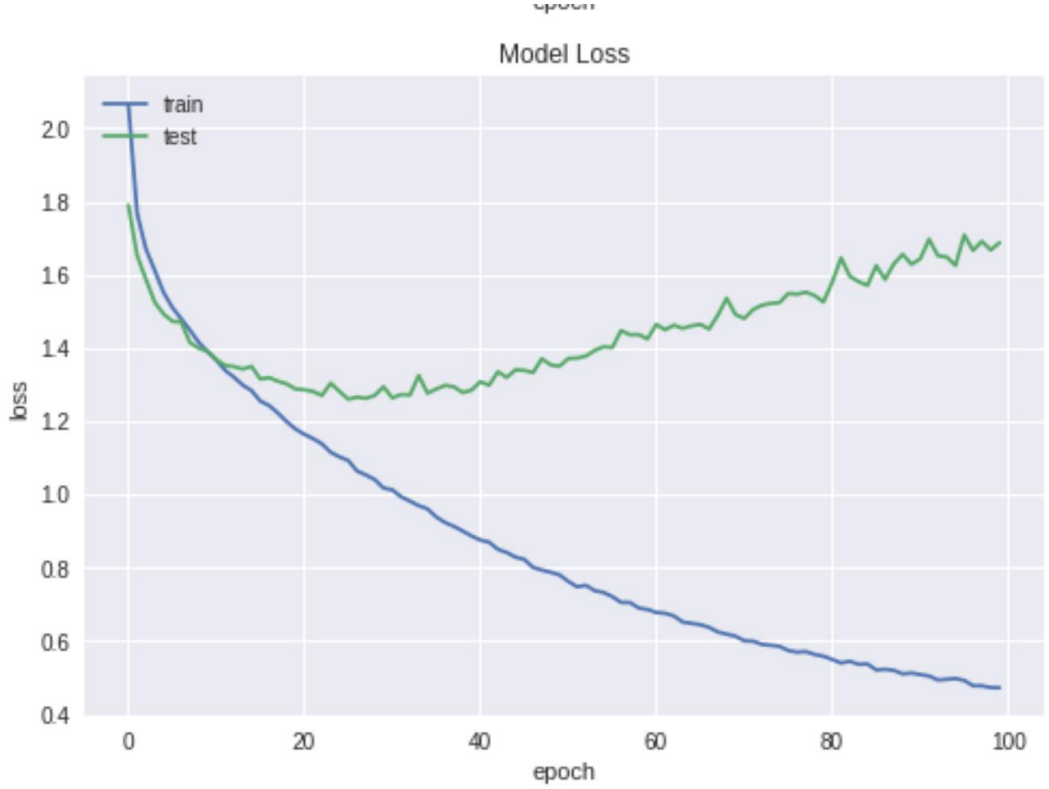
Batch Size – 256
Epochs – 50
Optimizer – Adamax() -
Full Connection Layer – 512, 512, 128
Dropout – 0.2

Accuracy:
From these models, I tried a lot of layer configurations from 2 to 4 layers with neurons ranging from 128 to 1024 and found out that the best accuracy is for – 512, 512, 128

With a accuracy of 56.47

```
Epoch 100/100
50000/50000 [==============================] - 16s 312us/step - loss: 0.4722 - acc
Test Loss:  1.6869818399429322
Test Accuracy 0.5647
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Observation 5 –

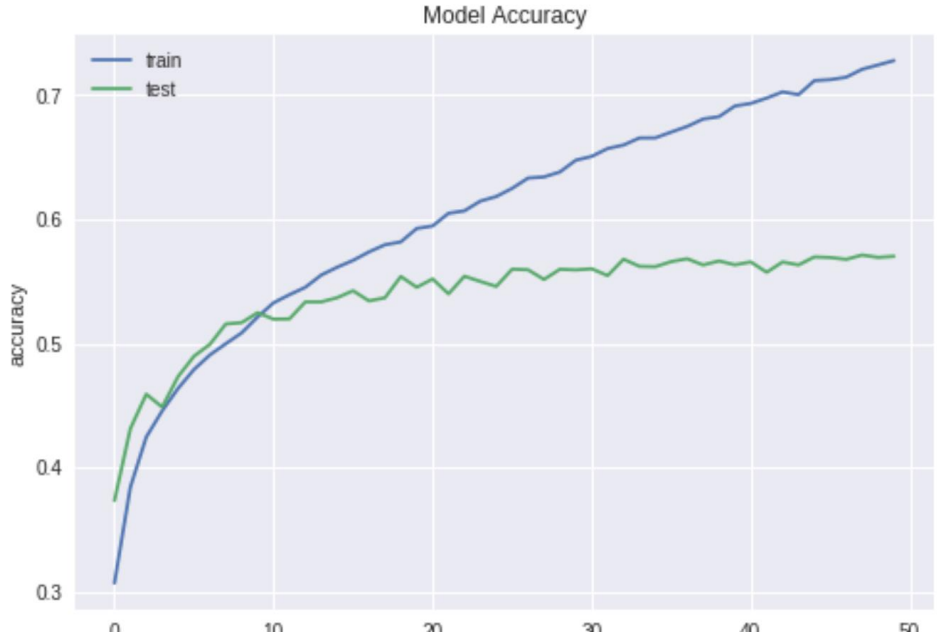
Best Activation Function–

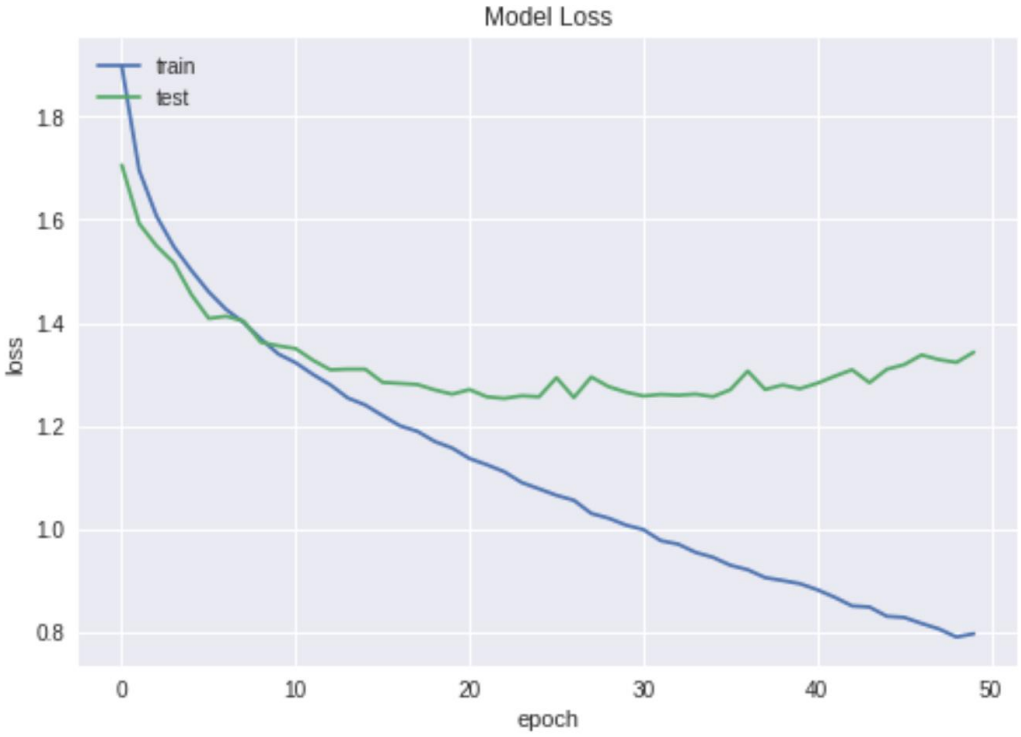
Batch Size – 256
Epochs – 50
Optimizer – Adamax() -
Full Connection Layer – 512, 512, 128
Dropout – 0.2
Activation – elu , Swich

Accuracy:
From these models, I tried a lot of Activation Functions including Relu , LeakyReLU , Elu , Swich which is proposed by Google recently and Tanh ,
The elu and Swich gave the best accuracy.
With a accuracy of 57.46 and a training accuracy of 67.68

```
#building the model for Multi Layer Perceptron
model = Sequential()
model.add(Dense(512, activation = swish ,kernel_initializer = 'uniform', input_shape=(3072,)))
model.add(Dropout(0.18))
model.add(Dense(512,activation = swish, kernel_initializer = 'uniform', input_shape=(3072,)))
model.add(Dropout(0.25))
model.add(Dense(128,activation = swish, kernel_initializer = 'uniform', input_shape=(3072,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax', kernel_initializer = 'uniform'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=adamax(),
              metrics=['accuracy'])
```

```
Epoch 50/50
50000/50000 [==============================] - 2s 43us/step - loss: 0.7496
Test Loss:  1.3884738193511963
Test Accuracy 0.5702
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Conclusion :

I have tried and tested different parameters and found out which works the best individually and which gives the best accuracy. I got a final testing accuracy of 57.02 % which is very good for a model with low epochs.

We could run this with Cloud and better infra to get a high accuracy and use Batch Normalalization techniques to get even lower loss.