Convolution Neural Network on CIFAR-10

Topic: Different layer configurations for deeper network topologies and their effect on the model

We began with studying the effects of the Convolutional layers on the model, which specifies the number of feature maps to be created in each convolution.
The input dimension parameter is the No of feature detectors and the size of those feature detectors which usual are 3*3.

Model 1 –

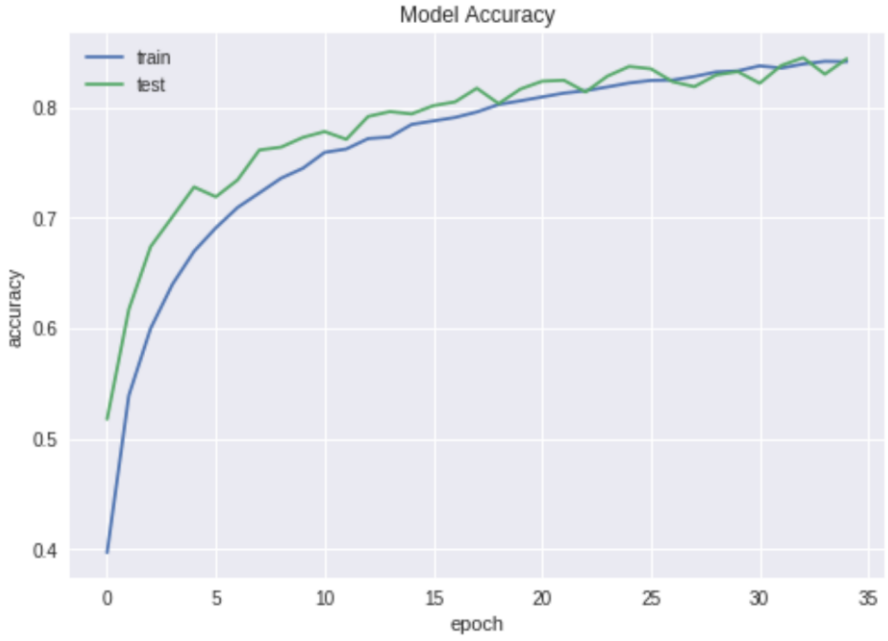
Network Configration:

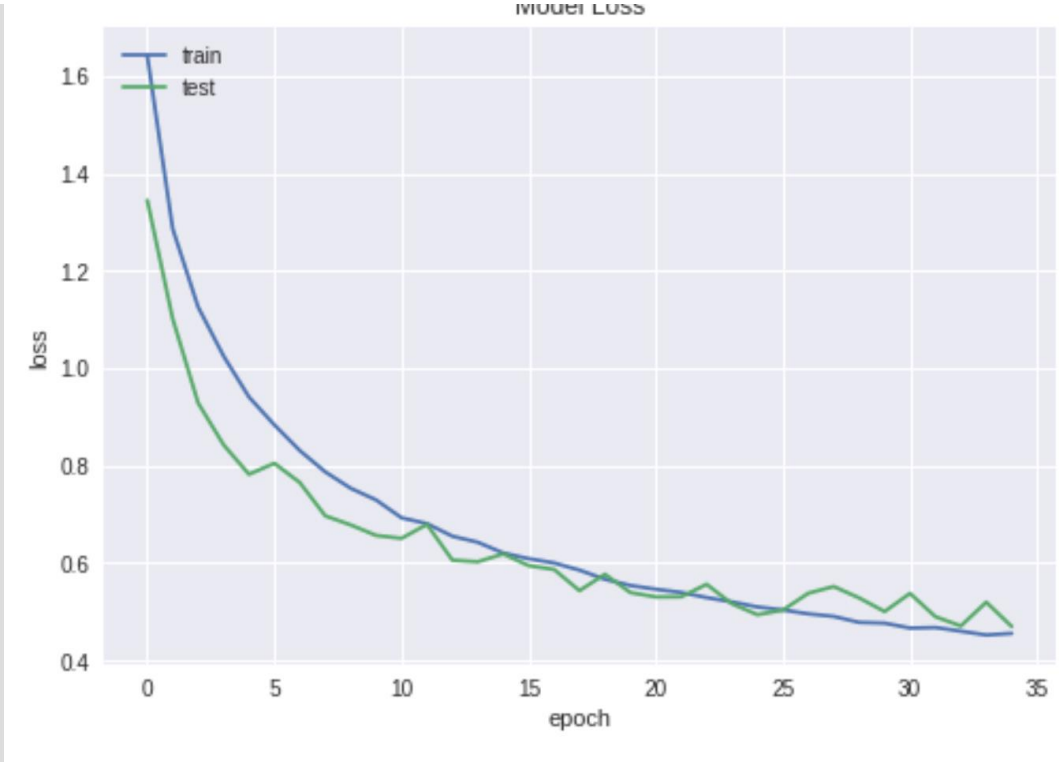
Batch Size – 64
Epochs - 35
Optimizer – Adamax()
Full Connection Layer – 512
Convolution Layers – 32, 32, 64, 64
Dropout – 0.2

Accuracy:
Using this model our Testing accuracy was 84.41% and closer to the training accuracy of 84.17% .This was a good model considering that the Network is 4 convolution layers and 2 Max pooling layers, along with just a epoch of 35.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.adamax()

# Let's train the model using RMSprop
```

```
10000/10000 [==============================] - 4s 362us/step
Test loss: 0.46965204153060913
Test accuracy: 0.8441
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Model 2 –

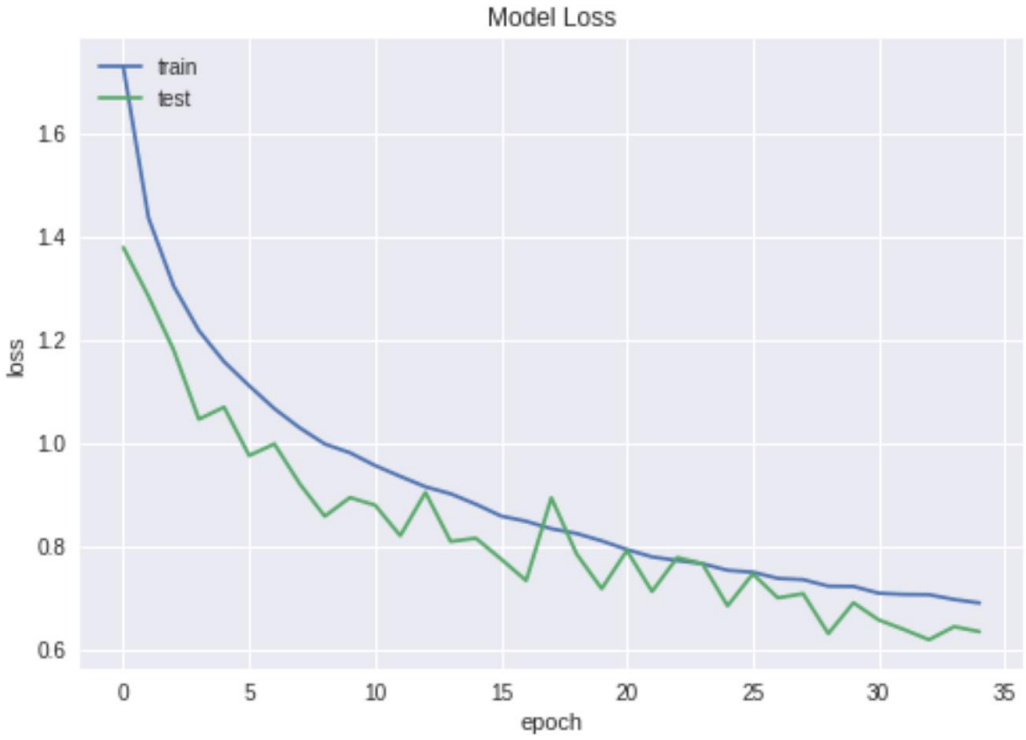
Now we try to create a more dense network in order to get a better accuracy and lesser loss;

Network Configration:

Batch Size – 64
Epochs - 35
Optimizer – Adamax()
Full Connection Layer – 512, 1024
Convolution Layers – 32, 32, 64, 64
Dropout – 0.2

Accuracy:
Using this model our Testing accuracy was 78.46% . This is lesser than the previous modelThis was a good model considering that the Network is 4 convolution layers and 2 Max pooling layers, along with just a epoch of 35. But even after trying to increase the epochs to 65, the accuracy was 79.8%. The training accuracy was also lower than the testing.

```
781/781 [==============================] - 66s 85ms/step - loss: 0.6903 - acc: 0.
Saved trained model at /content/saved_models/keras_cifar10_trained_model.h5
10000/10000 [==============================] - 3s 250us/step
Test loss: 0.634798113822937
Test accuracy: 0.7846
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



```
Saved trained model at /content/saved_models/keras_cifar10_trained_model
10000/10000 [==============================] - 1s 146us/step
Test loss: 0.6030456302642823
Test accuracy: 0.798
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

Model 3–

Now we try to create a more dense network in order to get a better accuracy and lesser loss with increasing the convolution layers to 6 -

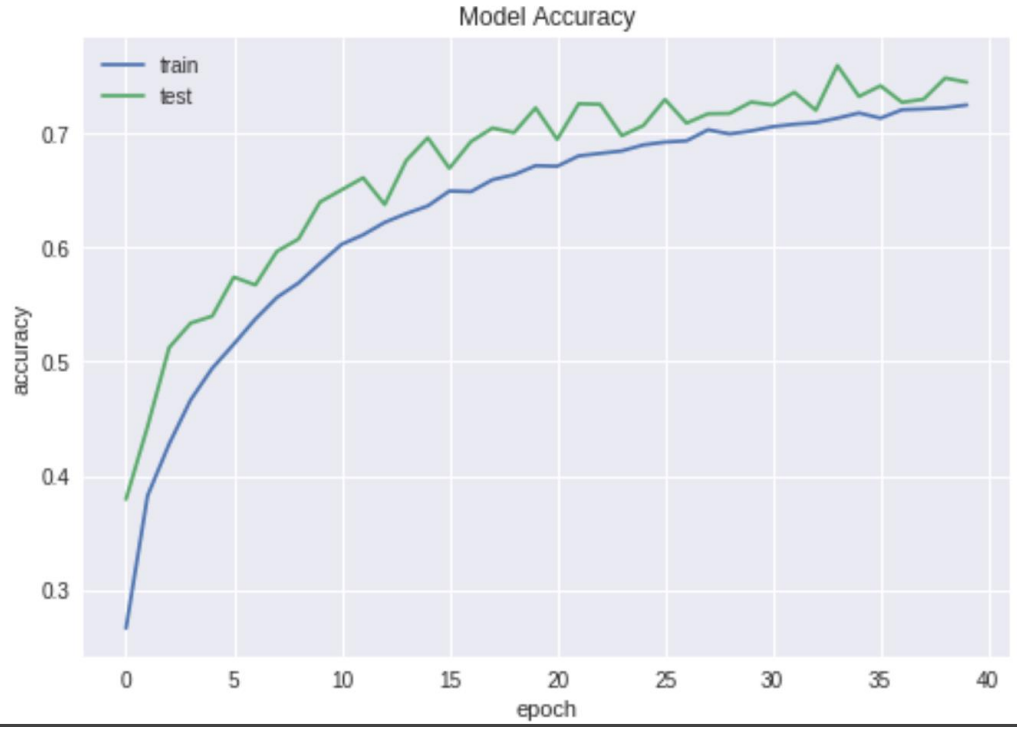
Network Configration:

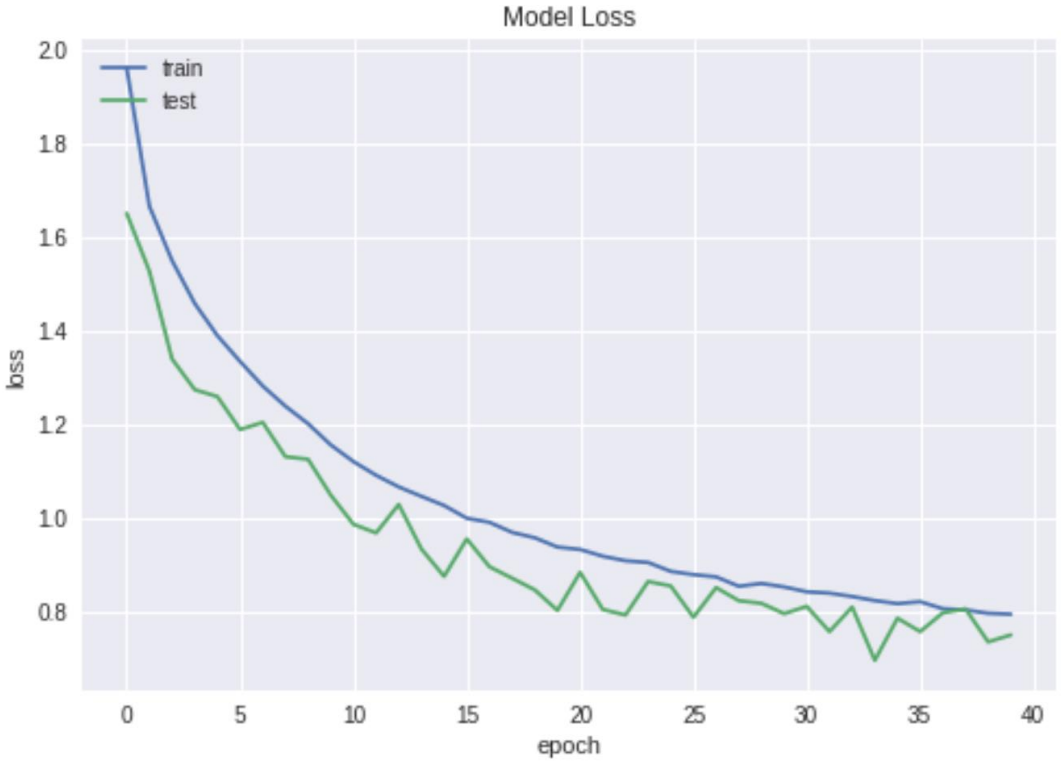
Batch Size – 128
Epochs - 40
Optimizer – Adamax()
Full Connection Layer – 256
Convolution Layers – 32, 32, 64, 64 , 80, 80
Dropout – 0.2
Data Augmentation – true (Rotation range - 45)

Accuracy:
Using this model our Testing accuracy was 74.54%. This was a decent model because the accuracy and loss were going in tandem with the training. But it is a slower one so it will need more computation power to get better accuracy.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.2))

model.add(Conv2D(80, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(80, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
10000/10000 [==============================] - 43s 389us/step
Test loss: 0.7500793851852418
Test accuracy: 0.7454
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Model 4–

Now we try to create a more dense network in order to get a better accuracy and lesser loss with increasing the convolution layers to 6 with more number of feature detectors in the middle layer-

Also, Adamax gives the best accuracy than RMSprop so we have used that instead as the optimizer.

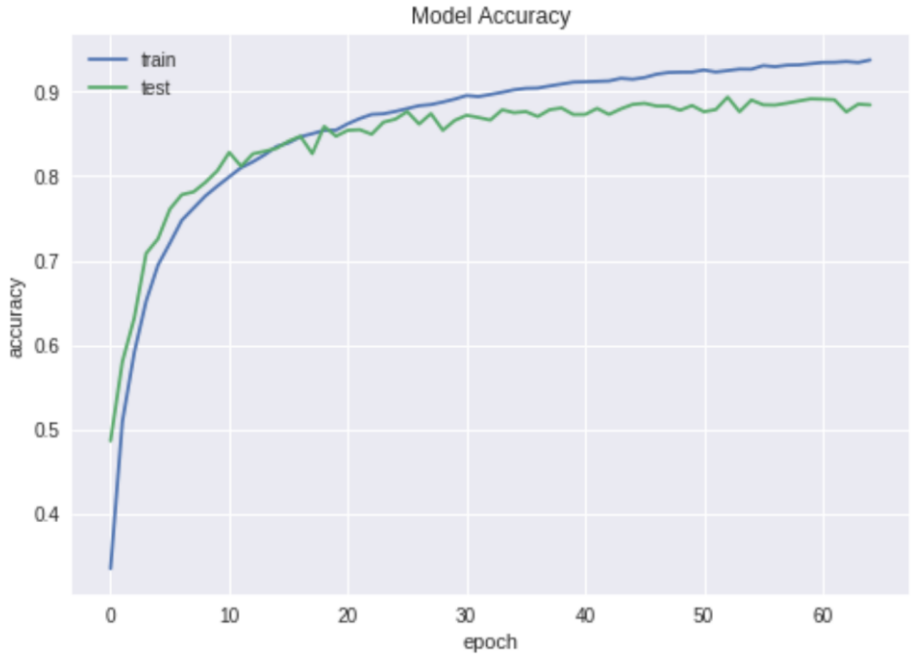
Network Configration:

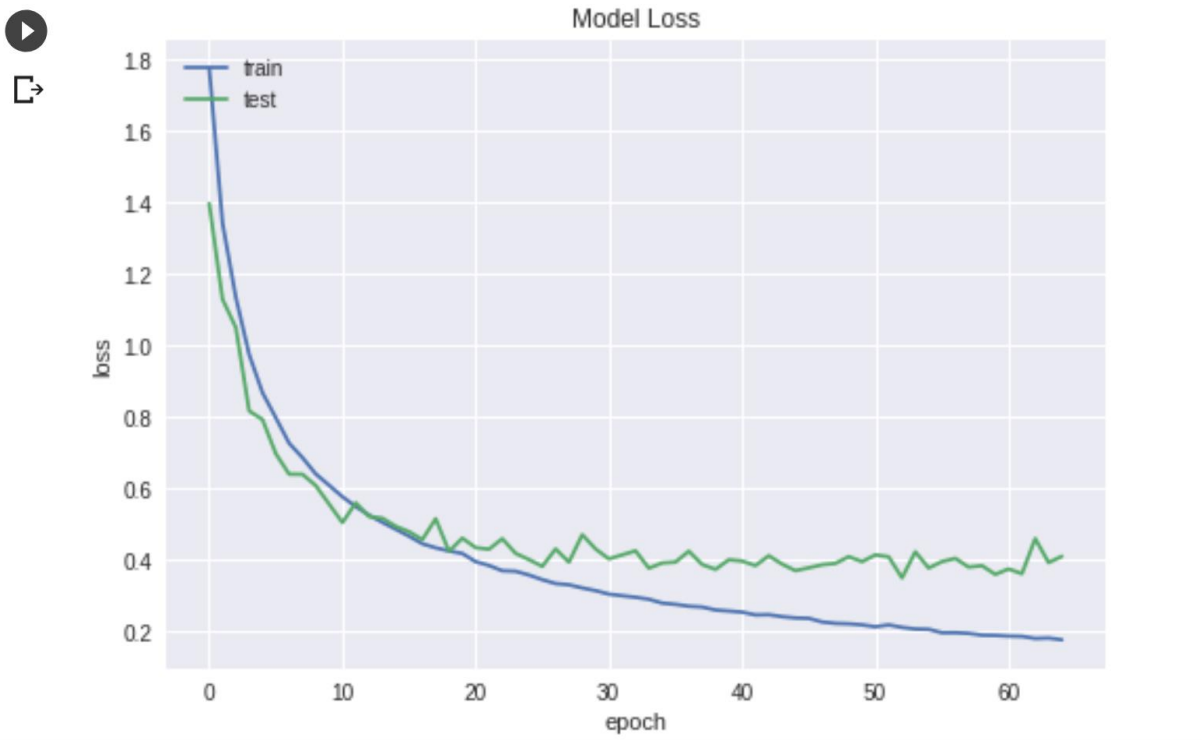
Batch Size – 128
Epochs - 65
Optimizer – Adamax()
Full Connection Layer – 512
Convolution Layers – 128,128, 256,256, 128
Dropout – 0.25
Data Augmentation – true (Rotation range - 45)

Accuracy:

Using this model our Testing accuracy was 88.45%. The training accuracy was 93.78%. This is the second best model we could achieve, and the graphs below shows that increasing the middle convolution layers in a Medium complexity dataset of images like cifar will increase the accuracy, as the middle layer detects the low level features but a little better then the first layer.

```
Test loss: 0.41086669574975965
Test accuracy: 0.8845
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





Model 4–

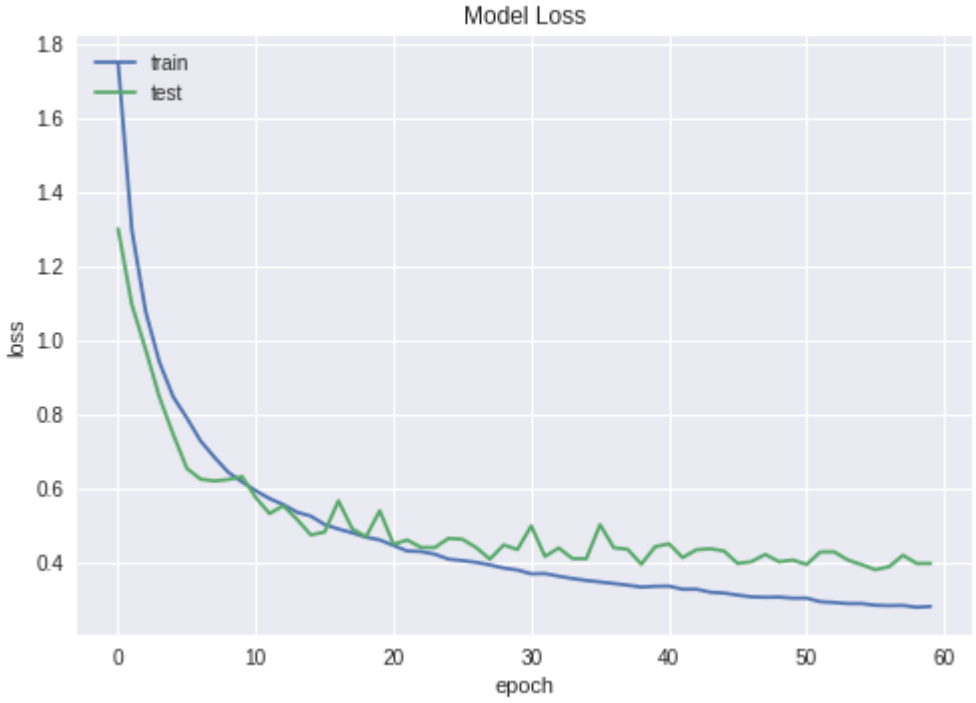
Going on the same track we changed few layers.
Network Configration:

Batch Size – 128
Epochs - 65
Optimizer – Adamax()
Full Connection Layer – 512
Convolution Layers – 128,128, 128,128,128,128
Dropout – 0.25
Data Augmentation – true (Rotation range - 45)

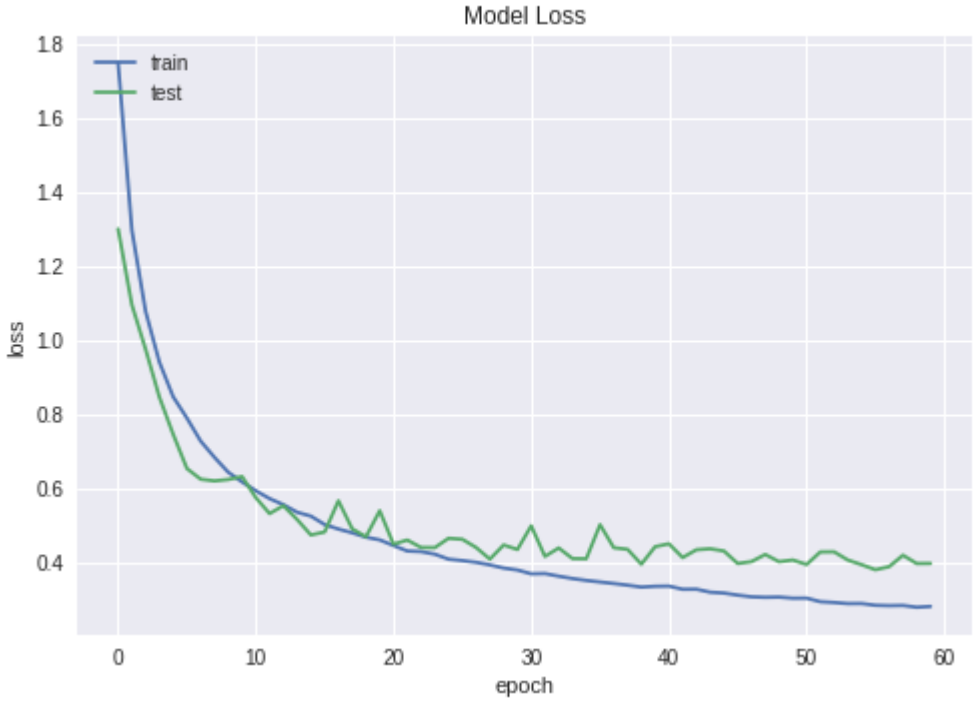
Accuracy:
Using this model our Testing accuracy was 87.78%. This is the third best model we could achieve, and the graphs below shows that middle layer indeed affect the accuracy in this. So we could again increase that and try for our best model.

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.adamax()
```

```
Test loss: 0.39712506093978883
Test accuracy: 0.8778
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



```
Test loss: 0.39712506093978883
Test accuracy: 0.8778
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



Model 5 –

Going on the same track we changed few layers.
Network Configration:

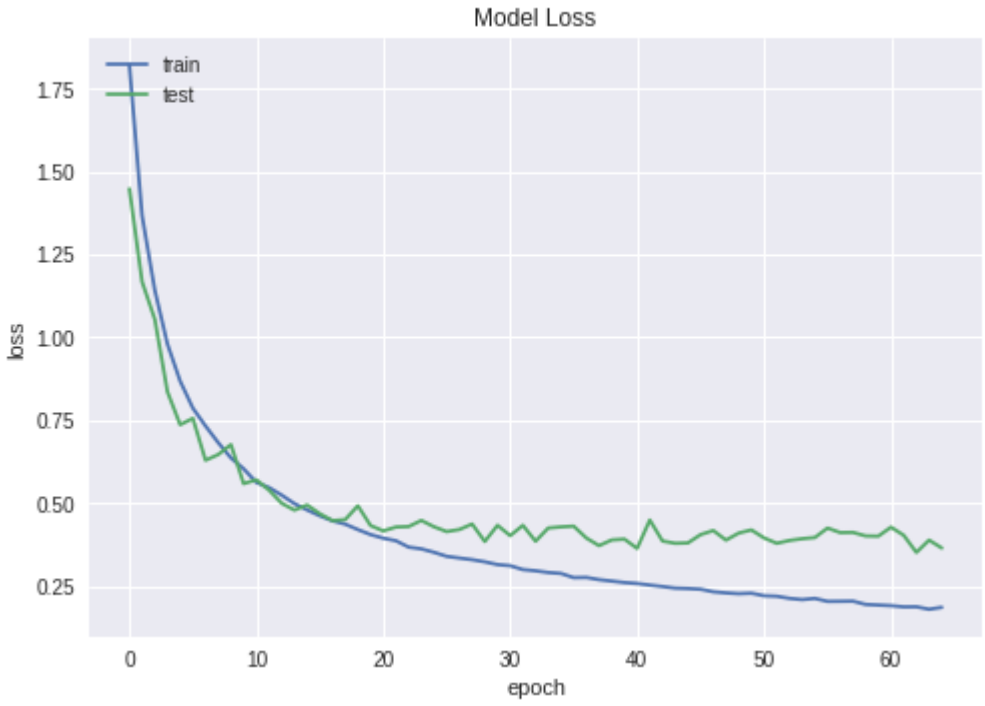
Batch Size – 128
Epochs - 65

Optimizer – **Adamax()**
Full Connection Layer – **512, 1024**
Convolution Layers – **128,128, 256,256,128,128**
Dropout – **0.25**
Data Augmentation – **true (Rotation range - 45)**

Accuracy:
This is the best accuracy we have which is indeed very good knowing that most research papers also has a accuracy of only 90-92% for Cifar10 , we also increased the Core layer more dense with this and added another layer of 1024 after 512 which resulted in a much better loss and accuracy. Eeven the training accuracy was **93.57%.** The Testing accuracy was **89.03%** .

```
model = Sequential()
model.add(Conv2D(128, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.24))


model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
#model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.24))
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.24))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.adamax()
```

```
Test loss: 0.3645875212132931
Test accuracy: 0.8903
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



Conclusion :

The lower layers of the network are able to learn a dense compact representation of the input dataset that retains most of the relevant information in the input. This is what is referred to as 'representation learning' and is equivalent to hand engineering features from the input dataset before feeding the features to a traditional machine learning model. The upper layers are focused on the actual learning

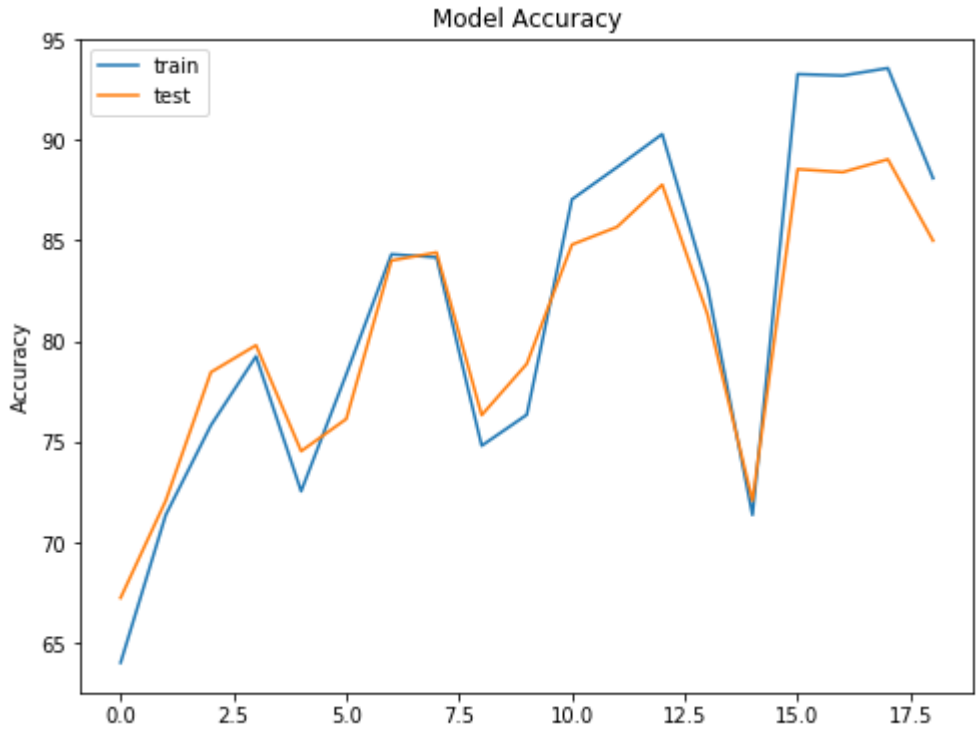
task. In our example above, edges are really lower level features, face & tyres etc are higher level aggregates/signals learned from a combination of the lower level features.

The dataset of Cifa10 is not that large so we cant use high/very low Dropout as dropping out data at random will cause underfitting/overfitting.
The purpose of having the flattening function is to get the desired shape. Hence, we give it before the denser layer.

Final model consists of the core layers with values: dense layer as 512, 1024, epochs 65, activation = 'relu'.
This model has 6 layers with CNN1 = 128, CNN2 = 128, CNN3 =256, CNN4 = 256, CNN5 = 128, CNN6 = 128. The accuracy has been the highest of all models with 89.03% and the loss is very minimal with 0.185

Visualization

1. **All models Test Accuracy vs Train Accuracy:**



From this plot, we can tell that all the model's training and testing accuracy were almost in sync. This can say the models were neither over fitting or under fitting.

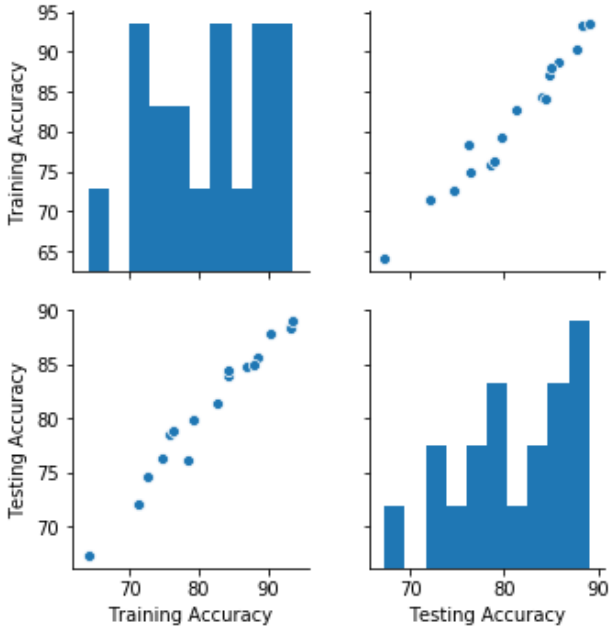
2. All models Test Loss vs Train Loss:

From this plot, we can tell that all the model's training and testing losses were minimal. This shows all the models are good.

3. Pairplot for Testing Accuracy vs Training Accuracy:

```
In [11]: sns.pairplot(data=df[['Training Accuracy','Testing Accuracy']])
```

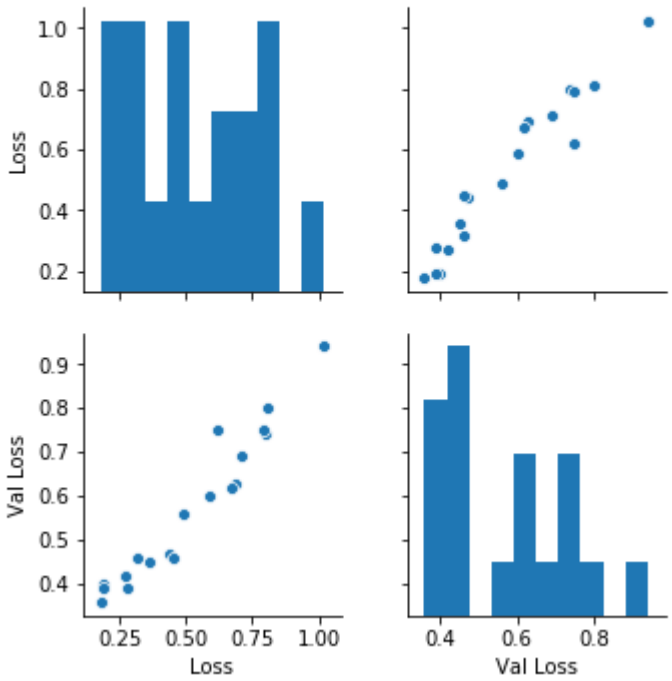
```
Out[11]: <seaborn.axisgrid.PairGrid at 0x208c60c10f0>
```



4. Pairplot for Test Loss vs Train Loss:

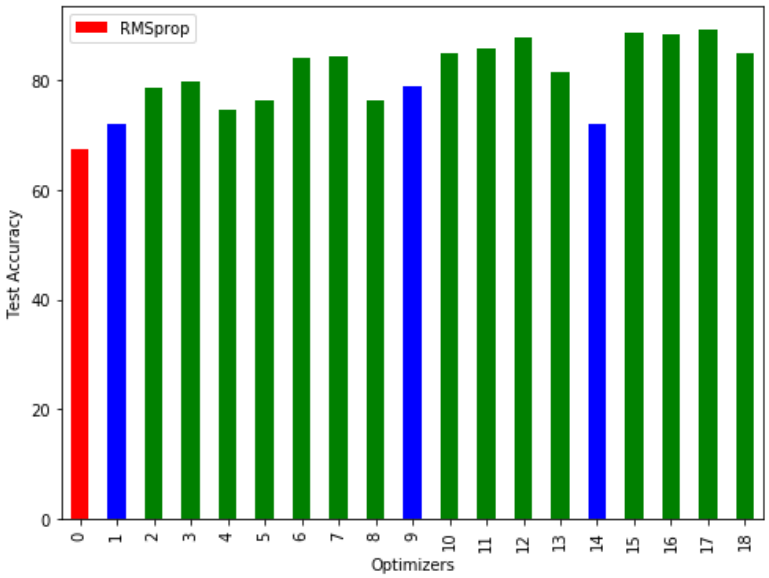
```
sns.pairplot(data=df[['Loss','Val Loss']])
```

```
<seaborn.axisgrid.PairGrid at 0x208c6401a20>
```



5. Barchart for Optimizers vs Testing Accuracy:

In [13]:
```
colors = {'RMSprop': 'r', 'Adam': 'b', 'Adamax': 'g'}
df['Testing Accuracy'].plot(kind='bar', color=[colors[i] for i in df['Optimizer']])
plt.ylabel('Test Accuracy')
plt.xlabel('Optimizers')
plt.legend(labels=('RMSprop','Adam','Adamax'), loc='upper left',prop={'size': 10})
plt.rcParams['figure.figsize'] = (8, 6)
plt.show()
```



This BarChart helps us choose the optimizer for the models. Among which **Adamax** was the best.