Aim : To achieve a better model

Past Model : The best model was achieved by the following hyperparameters : batch_size = 32, epochs = 100 and by applying two dense layers.

Defining a New Model:

Through the presentation in the lecture we understood various other parameters we could tweak to achieve a better model with higher accuarcy.

Implementing Grid Search :

For this purpose, we decided to use the grid search to derive the best paramaters between epochs and batch size rather that manually tuning the parameters. We also tried implementing the same for dropouts.

```
In [*]: # create model
model = KerasClassifier(build_fn=create_model, verbose=1)

# initiate RMSprop optimizer
#opt = keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)


# define the grid search parameters
batch_size = [32, 70, 40]
epochs = [10, 30, 50]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)

print(grid_result)
print("yes")
```

```
Epoch 1/10
Epoch 1/10
  320/33333 [..............................] - ETA: 4:01 - loss: 3.6078 - acc: 0.1094Epoch 1/10
 3840/33333 [==>...........................] - ETA: 37s - loss: 2.3739 - acc: 0.1313Epoch 1/30
10944/33333 [========>.....................] - ETA: 18s - loss: 2.1499 - acc: 0.1949Epoch 1/30
14752/33333 [============>.................] - ETA: 14s - loss: 2.0455 - acc: 0.2369Epoch 1/30
18016/33333 [===============>..............] - ETA: 12s - loss: 1.9757 - acc: 0.2634Epoch 1/50
21312/33333 [==================>...........] - ETA: 9s - loss: 1.9075 - acc: 0.2900Epoch 1/50
33333/33333 [==============================] - 23s 703us/step - loss: 1.7412 - acc: 0.3557
30144/33333 [==========================>...] - ETA: 2s - loss: 1.7164 - acc: 0.3715Epoch 2/10
33333/33333 [==============================] - 23s 703us/step - loss: 1.6805 - acc: 0.3860
Epoch 2/10
33333/33333 [==============================] - 19s 567us/step - loss: 1.2154 - acc: 0.5627
Epoch 3/10
33333/33333 [==============================] - 19s 558us/step - loss: 1.2052 - acc: 0.5678
Epoch 3/10
33333/33333 [==============================] - 19s 568us/step - loss: 1.0087 - acc: 0.6410
Epoch 4/10
33333/33333 [==============================] - 19s 559us/step - loss: 1.0205 - acc: 0.6390
```

jupyter Untitled1 Last Checkpoint: 2 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python [conda env:tensorflow_p36]

Run Code

```
33333/33333 [==============================] - 19s 558us/step - loss: 1.2052 - acc: 0.5678
Epoch 3/10
33333/33333 [==============================] - 19s 568us/step - loss: 1.0087 - acc: 0.6410
Epoch 4/10
33333/33333 [==============================] - 19s 559us/step - loss: 1.0205 - acc: 0.6390
Epoch 4/10
33333/33333 [==============================] - 19s 568us/step - loss: 0.8912 - acc: 0.6836
Epoch 5/10
33333/33333 [==============================] - 19s 557us/step - loss: 0.8962 - acc: 0.6850
Epoch 5/10
33333/33333 [==============================] - 19s 568us/step - loss: 0.8090 - acc: 0.7157
Epoch 6/10
33333/33333 [==============================] - 19s 558us/step - loss: 0.8037 - acc: 0.7172
Epoch 6/10
33333/33333 [==============================] - 19s 569us/step - loss: 0.7389 - acc: 0.7424
Epoch 7/10
33333/33333 [==============================] - 19s 557us/step - loss: 0.7302 - acc: 0.7443
Epoch 7/10
33333/33333 [==============================] - 19s 559us/step - loss: 0.6707 - acc: 0.7647
Epoch 8/10
33333/33333 [==============================] - 19s 575us/step - loss: 0.6813 - acc: 0.7631
Epoch 8/10
33333/33333 [==============================] - 19s 557us/step - loss: 0.6142 - acc: 0.7847
Epoch 9/10
33333/33333 [==============================] - 19s 574us/step - loss: 0.6239 - acc: 0.7836
Epoch 9/10
33333/33333 [==============================] - 19s 558us/step - loss: 0.5643 - acc: 0.8039
Epoch 10/10
33333/33333 [==============================] - 19s 571us/step - loss: 0.5747 - acc: 0.8009
Epoch 10/10
33333/33333 [==============================] - 19s 559us/step - loss: 0.5185 - acc: 0.8224
33333/33333 [==============================] - 19s 571us/step - loss: 0.5270 - acc: 0.8174
16667/16667 [==============================] - 3s 182us/step
16667/16667 [==============================] - 3s 176us/step
33333/33333 [==============================] - 6s 173us/step
33333/33333 [==============================] - 5s 158us/step
Epoch 1/50
```

In [*]:

```python
# create model
model = KerasClassifier(build_fn=create_model, verbose=1)

# define the grid search parameters
batch_size = [32,64,128]
epochs = [20,30,40]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)

print(grid_result)
print("yes")
```

```
Epoch 1/20
  320/33333 [..............................] - ETA: 3:26 - loss: 2.5929 - acc: 0.1094Epoch 1/20
 4128/33333 [==>...........................] - ETA: 28s - loss: 2.3346 - acc: 0.1160Epoch 1/20
 8320/33333 [======>.......................] - ETA: 18s - loss: 2.1945 - acc: 0.1826Epoch 1/30
13120/33333 [==========>...................] - ETA: 12s - loss: 2.0507 - acc: 0.2387Epoch 1/30
18240/33333 [===============>..............] - ETA: 8s - loss: 1.9319 - acc: 0.2867Epoch 1/30
23072/33333 [===================>..........] - ETA: 5s - loss: 1.8410 - acc: 0.3214Epoch 1/40
29024/33333 [=========================>....] - ETA: 2s - loss: 1.7595 - acc: 0.3519Epoch 1/40
33333/33333 [==============================] - 16s 492us/step - loss: 1.7137 - acc: 0.3685
Epoch 2/20
33333/33333 [==============================] - 11s 322us/step - loss: 1.2406 - acc: 0.5520
Epoch 3/20
33333/33333 [==============================] - 11s 322us/step - loss: 1.0493 - acc: 0.6251
Epoch 4/20
33333/33333 [==============================] - 11s 323us/step - loss: 0.9299 - acc: 0.6708
Epoch 5/20
33333/33333 [==============================] - 11s 321us/step - loss: 0.8464 - acc: 0.7034
Epoch 6/20
33333/33333 [==============================] - 11s 321us/step - loss: 0.7731 - acc: 0.7291
Epoch 7/20
```

```
33333/33333 [==============================] - 19s 555us/step - loss: 0.4673 - acc: 0.8388
Epoch 13/20
33333/33333 [==============================] - 19s 564us/step - loss: 0.4236 - acc: 0.8545
Epoch 14/20
33333/33333 [==============================] - 17s 516us/step - loss: 0.3837 - acc: 0.8690
Epoch 15/20
33333/33333 [==============================] - 11s 321us/step - loss: 0.3450 - acc: 0.8846
Epoch 16/20
33333/33333 [==============================] - 11s 328us/step - loss: 0.3104 - acc: 0.8972
Epoch 17/20
33333/33333 [==============================] - 13s 388us/step - loss: 0.2796 - acc: 0.9071
Epoch 18/20
33333/33333 [==============================] - 19s 558us/step - loss: 0.2459 - acc: 0.9221
Epoch 19/20
33333/33333 [==============================] - 18s 554us/step - loss: 0.2184 - acc: 0.9315
Epoch 20/20
33333/33333 [==============================] - 19s 559us/step - loss: 0.1904 - acc: 0.9433
16667/16667 [==============================] - 3s 199us/step
33333/33333 [==============================] - 6s 193us/step
Epoch 1/40
```

Model 1

Characteristics:

```
batch_size = 64
num_classes = 10
epochs = 50
data_augmentation = True
num_predictions = 20
```

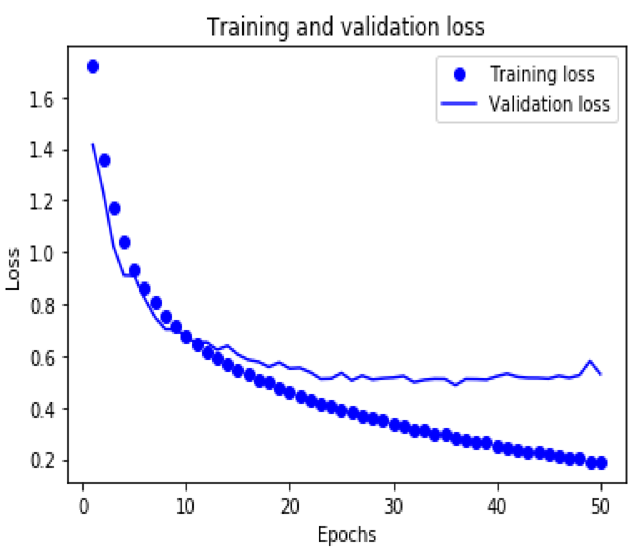
We implemented the above model using 6 layers and sgd as an optimizer.

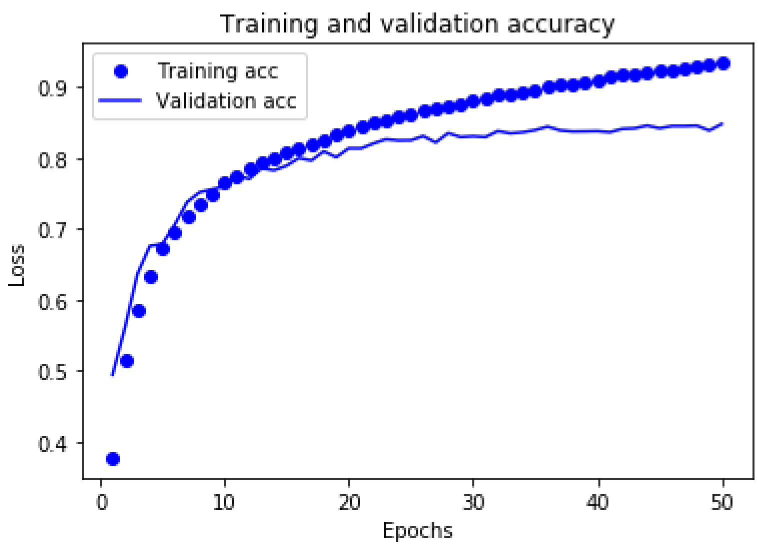
The reason behind sgd was that it does away with the redudancy of performing one updated at a time making the trainning of the model much faster.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 16, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 16, 16)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 16, 16)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 8, 8)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 128, 8, 8)         73856
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 128, 8, 8)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 128, 4, 4)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250
=================================================================
Total params: 2,395,562
Trainable params: 2,395,498
Non-trainable params: 64
```

```
Epoch 35/50
1563/1563 [==============================] - 28s 18ms/step - loss: 0.2946 - acc: 0.8965 - val_loss: 0.5101 - val_acc: 0.8393
Epoch 36/50
1563/1563 [==============================] - 28s 18ms/step - loss: 0.2852 - acc: 0.8998 - val_loss: 0.4856 - val_acc: 0.8442
Epoch 37/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2700 - acc: 0.9039 - val_loss: 0.5099 - val_acc: 0.8389
Epoch 38/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2694 - acc: 0.9044 - val_loss: 0.5093 - val_acc: 0.8373
Epoch 39/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2624 - acc: 0.9075 - val_loss: 0.5073 - val_acc: 0.8375
Epoch 40/50
1563/1563 [==============================] - 29s 19ms/step - loss: 0.2526 - acc: 0.9097 - val_loss: 0.5203 - val_acc: 0.8378
Epoch 41/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2432 - acc: 0.9135 - val_loss: 0.5306 - val_acc: 0.8363
Epoch 42/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2387 - acc: 0.9166 - val_loss: 0.5182 - val_acc: 0.8410
Epoch 43/50
1563/1563 [==============================] - 28s 18ms/step - loss: 0.2307 - acc: 0.9177 - val_loss: 0.5142 - val_acc: 0.8418
Epoch 44/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2269 - acc: 0.9202 - val_loss: 0.5138 - val_acc: 0.8457
Epoch 45/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2181 - acc: 0.9224 - val_loss: 0.5112 - val_acc: 0.8418
Epoch 46/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2152 - acc: 0.9234 - val_loss: 0.5223 - val_acc: 0.8450
Epoch 47/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.2034 - acc: 0.9274 - val_loss: 0.5141 - val_acc: 0.8450
Epoch 48/50
1563/1563 [==============================] - 28s 18ms/step - loss: 0.2023 - acc: 0.9289 - val_loss: 0.5249 - val_acc: 0.8455
Epoch 49/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.1919 - acc: 0.9326 - val_loss: 0.5796 - val_acc: 0.8387
Epoch 50/50
1563/1563 [==============================] - 29s 18ms/step - loss: 0.1892 - acc: 0.9339 - val_loss: 0.5294 - val_acc: 0.8479
```

Through the graph we realized that the model was overfitting due to the layers and the drop out function.





Model 2:

Characteristics:

```
batch_size = 32
num_classes = 10
epochs = 30
data_augmentation = True
```

We decreased the batch , epochs and removed the batch normalization layer while addding a dropout.

```
def base_model():

    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=x_train.shape[1:]))
    model.add(Dropout(0.2))

    model.add(Conv2D(32,(3,3),padding='same', activation='relu'))
    #model.add(BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True,
                                #beta_initializer='zeros', gamma_initializer='ones', moving_mean_initializer='zeros',
                                #moving_variance_initializer='ones', beta_regularizer=None,
                                #gamma_regularizer=None, beta_constraint=None, gamma_constraint=None))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Conv2D(32,(3,3),padding='same',activation='relu'))
    model.add(Dropout(0.2))

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

    model.add(Conv2D(128,(3,3),padding='same',activation='relu'))
   # model.add(Dropout(0.2))

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

    model.add(Flatten())
    #model.add(Dropout(0.2))
    model.add(Dense(1024,activation='relu'))
    #model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))


    opt = optimizers.SGD(lr=0.0008, decay=0.000, momentum=0.9, nesterov=True)
    # Train model

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 32, 16, 16)        0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 16, 16)        9248      
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 16, 16)        0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 16, 16)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 8, 8)          0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 128, 8, 8)         73856     
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 128, 8, 8)         147584    
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 128, 4, 4)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              2098176   
_________________________________________________________________
dense_4 (Dense)              (None, 10)                10250     
=================================================================
Total params: 2,367,754
Trainable params: 2,367,754
Non-trainable params: 0
_________________________________________________________________
```

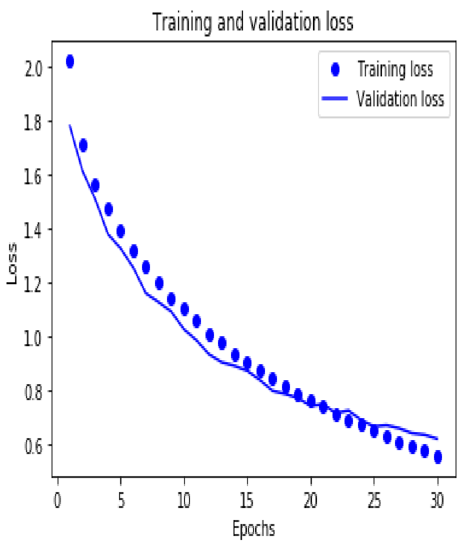
```
Epoch 13/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.9784 - acc: 0.6557 - val_loss: 0.9030 - val_acc: 0.6806
Epoch 14/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.9365 - acc: 0.6699 - val_loss: 0.8902 - val_acc: 0.6882
Epoch 15/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.9049 - acc: 0.6805 - val_loss: 0.8718 - val_acc: 0.6887
Epoch 16/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8760 - acc: 0.6927 - val_loss: 0.8376 - val_acc: 0.7051
Epoch 17/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8455 - acc: 0.7027 - val_loss: 0.7975 - val_acc: 0.7190
Epoch 18/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.8164 - acc: 0.7130 - val_loss: 0.7859 - val_acc: 0.7185
Epoch 19/30
1563/1563 [==============================] - 24s 16ms/step - loss: 0.7863 - acc: 0.7226 - val_loss: 0.7709 - val_acc: 0.7268
Epoch 20/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7619 - acc: 0.7323 - val_loss: 0.7415 - val_acc: 0.7407
Epoch 21/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7377 - acc: 0.7412 - val_loss: 0.7475 - val_acc: 0.7386
Epoch 22/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7136 - acc: 0.7475 - val_loss: 0.7144 - val_acc: 0.7481
Epoch 23/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6909 - acc: 0.7569 - val_loss: 0.7243 - val_acc: 0.7533
Epoch 24/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6743 - acc: 0.7633 - val_loss: 0.6889 - val_acc: 0.7579
Epoch 25/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6500 - acc: 0.7711 - val_loss: 0.6652 - val_acc: 0.7681
Epoch 26/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6259 - acc: 0.7825 - val_loss: 0.6698 - val_acc: 0.7700
Epoch 27/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.6093 - acc: 0.7864 - val_loss: 0.6585 - val_acc: 0.7747
Epoch 28/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5956 - acc: 0.7944 - val_loss: 0.6402 - val_acc: 0.7790
Epoch 29/30
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5780 - acc: 0.7971 - val_loss: 0.6343 - val_acc: 0.7855
Epoch 30/30
1563/1563 [==============================] - 24s 16ms/step - loss: 0.5557 - acc: 0.8067 - val_loss: 0.6195 - val_acc: 0.7892
```

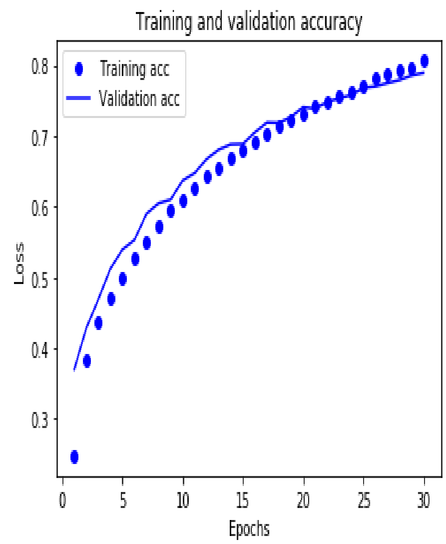
```
Saved trained model at /home/ubuntu/mynotebooks/saved_models/keras_cifar10_trained_model_revised3.h5
10000/10000 [==============================] - 2s 164us/step
Test loss: 0.6194539049625397
Test accuracy: 0.7892
```

Accuracy:

Through the previous model we seen that after a certain epoch the overfitting was increasing. So we added a dropout and decreased the epochs and batch size.

With this we achieved lesser accuracy than the previous model but a better trainning and validation loss ratio and also trainning accuracy and validation accuracy.





<u>Model 3 :</u>

Here first we started with the base code that was given. After running 100 epochs we found out the accuracy to be around 74.82.

```
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(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
Epoch 93/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7593 - acc: 0.7504 - val_loss: 0.6223 - val_acc: 0.7952
Epoch 94/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7591 - acc: 0.7488 - val_loss: 0.6657 - val_acc: 0.7789
Epoch 95/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7576 - acc: 0.7504 - val_loss: 0.6197 - val_acc: 0.7950
Epoch 96/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7637 - acc: 0.7486 - val_loss: 0.6637 - val_acc: 0.7861
Epoch 97/100
1563/1562 [==============================] - 40s 26ms/step - loss: 0.7604 - acc: 0.7480 - val_loss: 0.6502 - val_acc: 0.7847
Epoch 98/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7617 - acc: 0.7491 - val_loss: 0.6298 - val_acc: 0.7898
Epoch 99/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7649 - acc: 0.7485 - val_loss: 0.7144 - val_acc: 0.7638
Epoch 100/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.7690 - acc: 0.7482 - val_loss: 0.6448 - val_acc: 0.7847
Saved trained model at D:\NEU\CC & DNN\saved_models\keras_cifar10_trained_model.h5
10000/10000 [==============================] - 3s 272us/step
Test loss: 0.6448269945144653
Test accuracy: 0.7847
```

Model 4:

Now in order to increase this accuracy, various parameters needed to be changed and tried on to attain a better accuracy.

First, we started with changing the optimizer from rmsprop to adam. Because as discussed in he class by other students, adam is better at increasing the accuracy rather than rmsprop. Also here I have increased the learning rate from 0.001 to 0.002.

```
model.add(Activation('relu'))
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(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1, decay=0.0)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

```
Epoch 14/20
1563/1562 [==============================] - 39s 25ms/step - loss: 1.5820 - acc: 0.4192 - val_loss: 1.4567 - val_acc: 0.4743
Epoch 15/20
1563/1562 [==============================] - 40s 25ms/step - loss: 1.5556 - acc: 0.4289 - val_loss: 1.4305 - val_acc: 0.4862
Epoch 16/20
1563/1562 [==============================] - 39s 25ms/step - loss: 1.5407 - acc: 0.4362 - val_loss: 1.4156 - val_acc: 0.4901
Epoch 17/20
1563/1562 [==============================] - 39s 25ms/step - loss: 1.5201 - acc: 0.4451 - val_loss: 1.3784 - val_acc: 0.5046
Epoch 18/20
1563/1562 [==============================] - 37s 24ms/step - loss: 1.5048 - acc: 0.4497 - val_loss: 1.3686 - val_acc: 0.5080
Epoch 19/20
1563/1562 [==============================] - 43s 27ms/step - loss: 1.4898 - acc: 0.4585 - val_loss: 1.3609 - val_acc: 0.5127
Epoch 20/20
1563/1562 [==============================] - 40s 26ms/step - loss: 1.4730 - acc: 0.4656 - val_loss: 1.3305 - val_acc: 0.5224
Saved trained model at D:\NEU\CC & DNN\saved_models\keras_cifar10_trained_model.h5
10000/10000 [==============================] - 4s 355us/step
Test loss: 1.3305378057479857
Test accuracy: 0.5224

---------------------------------------------------------------------------
```

Model 5:

```
model.add(Flatten())
model.add(Dense(1024))
leakyrelu = keras.layers.advanced_activations.LeakyReLU(alpha=0.3)
model.add(leakyrelu)
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1, decay=0.0)
```

```
Epoch 13/20
1563/1562 [==============================] - 87s 56ms/step - loss: 1.5204 - acc: 0.4481 - val_loss: 1.4122 - val_acc: 0.4915
Epoch 14/20
1563/1562 [==============================] - 77s 49ms/step - loss: 1.5005 - acc: 0.4538 - val_loss: 1.3829 - val_acc: 0.5001
Epoch 15/20
1563/1562 [==============================] - 36s 23ms/step - loss: 1.4812 - acc: 0.4616 - val_loss: 1.3623 - val_acc: 0.5151
Epoch 16/20
1563/1562 [==============================] - 36s 23ms/step - loss: 1.4613 - acc: 0.4703 - val_loss: 1.3308 - val_acc: 0.5228
Epoch 17/20
1563/1562 [==============================] - 39s 25ms/step - loss: 1.4389 - acc: 0.4754 - val_loss: 1.3193 - val_acc: 0.5292
Epoch 18/20
1563/1562 [==============================] - 38s 24ms/step - loss: 1.4224 - acc: 0.4863 - val_loss: 1.3043 - val_acc: 0.5348
Epoch 19/20
1563/1562 [==============================] - 36s 23ms/step - loss: 1.4076 - acc: 0.4900 - val_loss: 1.2737 - val_acc: 0.5475
Epoch 20/20
1563/1562 [==============================] - 36s 23ms/step - loss: 1.3882 - acc: 0.4995 - val_loss: 1.2558 - val_acc: 0.5540
Saved trained model at D:\NEU\CC & DNN\saved_models\keras_cifar10_trained_model.h5
10000/10000 [==============================] - 2s 211us/step
Test loss: 1.255837732887268
Test accuracy: 0.554
```

Now we tried changing the dense layer to 1024 in an attempt to try get a better accuracy and making the end layer more dense. I alslo replaced the ReLU function with Leaky ReLU again because it was discussed that it is better than ReLU function as it also has a small negative edge which works better with negative values.

Model 6 :

Here I have increased the 2D convolution layers, and I now all the activation functions are Leaky ReLU. We made the convolution layers double and as result accuracy increased to 83.26 % with any sign of overfitting. The validation accuracy is 82.11%.

```python
leakyrelu = keras.layers.advanced_activations.LeakyReLU(alpha=0.3)
model.add(leakyrelu)
model.add(Conv2D(conv2d, (3, 3)))
model.add(Conv2D(conv2d, (3, 3)))
model.add(leakyrelu)
model.add(leakyrelu)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(conv2d1, (3, 3), padding='same'))
model.add(leakyrelu)
model.add(Conv2D(conv2d1, (3, 3)))
model.add(Conv2D(conv2d1, (3, 3)))
model.add(leakyrelu)
model.add(leakyrelu)
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.1))

model.add(Flatten())
model.add(Dense(dense))
model.add(leakyrelu)
model.add(Dropout(dropout))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
```

```
Epoch 42/50
1563/1562 [==============================] - 55s 35ms/step - loss: 0.5136 - acc: 0.8243 - val_loss: 0.5306 - val_acc: 0.8214
Epoch 43/50
1563/1562 [==============================] - 78s 50ms/step - loss: 0.5039 - acc: 0.8270 - val_loss: 0.5537 - val_acc: 0.8138
Epoch 44/50
1563/1562 [==============================] - 70s 45ms/step - loss: 0.5070 - acc: 0.8272 - val_loss: 0.5041 - val_acc: 0.8287
Epoch 45/50
1563/1562 [==============================] - 53s 34ms/step - loss: 0.5017 - acc: 0.8293 - val_loss: 0.5207 - val_acc: 0.8237
Epoch 46/50
1563/1562 [==============================] - 54s 35ms/step - loss: 0.4975 - acc: 0.8303 - val_loss: 0.5099 - val_acc: 0.8235
Epoch 47/50
1563/1562 [==============================] - 55s 35ms/step - loss: 0.4938 - acc: 0.8316 - val_loss: 0.5166 - val_acc: 0.8256
Epoch 48/50
1563/1562 [==============================] - 55s 35ms/step - loss: 0.4877 - acc: 0.8333 - val_loss: 0.5083 - val_acc: 0.8293
Epoch 49/50
1563/1562 [==============================] - 72s 46ms/step - loss: 0.4885 - acc: 0.8322 - val_loss: 0.5120 - val_acc: 0.8283
Epoch 50/50
1563/1562 [==============================] - 90s 57ms/step - loss: 0.4862 - acc: 0.8326 - val_loss: 0.5238 - val_acc: 0.8211
```

Model 7:

Here we have increased the number of epochs of 100 and I have kept everything the same as observation 4. This is our best accuracy found from all our observations. 85.54%

```
batch_size = 32
num_classes = 10
epochs = 100
num_predictions = 20
conv2d=32
conv2d1=64
verbose=1
input_shape= 784
data_augmentation = True
dense=512
dropout=0.2
```

```
model_name = 'keras_cifar10_trained_model.h5'

# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
# 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)

model = Sequential()
model.add(Conv2D(conv2d, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))

leakyrelu = keras.layers.advanced_activations.LeakyReLU(alpha=0.3)
model.add(leakyrelu)
model.add(Conv2D(conv2d, (3, 3)))
model.add(leakyrelu)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(conv2d1, (3, 3), padding='same'))
model.add(leakyrelu)
model.add(Conv2D(conv2d1, (3, 3)))
model.add(leakyrelu)
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.1))

model.add(Flatten())
model.add(Dense(dense))
model.add(leakyrelu)
model.add(Dropout(dropout))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
```

```
1563/1562 [==============================] - 43s 28ms/step - loss: 0.4305 - acc: 0.8517 - val_loss: 0.5047 - val_acc: 0.8337
Epoch 94/100
1563/1562 [==============================] - 42s 27ms/step - loss: 0.4229 - acc: 0.8542 - val_loss: 0.4632 - val_acc: 0.8435
Epoch 95/100
1563/1562 [==============================] - 42s 27ms/step - loss: 0.4246 - acc: 0.8544 - val_loss: 0.4705 - val_acc: 0.8436
Epoch 96/100
1563/1562 [==============================] - 41s 26ms/step - loss: 0.4209 - acc: 0.8525 - val_loss: 0.4684 - val_acc: 0.8431
Epoch 97/100
1563/1562 [==============================] - 40s 25ms/step - loss: 0.4226 - acc: 0.8549 - val_loss: 0.4727 - val_acc: 0.8438
Epoch 98/100
1563/1562 [==============================] - 39s 25ms/step - loss: 0.4203 - acc: 0.8550 - val_loss: 0.4816 - val_acc: 0.8417
Epoch 99/100
1563/1562 [==============================] - 41s 26ms/step - loss: 0.4196 - acc: 0.8558 - val_loss: 0.4849 - val_acc: 0.8402
Epoch 100/100
1563/1562 [==============================] - 41s 26ms/step - loss: 0.4175 - acc: 0.8554 - val_loss: 0.4607 - val_acc: 0.8471

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-6af073ed1e9f> in <module>()
```

Conclusion:

We achieved the best acuraccy in Model 7 with 85.54% by tweaking the hyperparameters and layers. From the previous task and the new task we derived the best model with leaky relu.