* 1

#Importing library

import keras

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

import numpy as np

np.random.seed(1000)

#Instantiation

AlexNet = Sequential()

#1st Convolutional Layer

AlexNet.add(Conv2D(filters=96, input\_shape=(32,32,3), kernel\_size=(11,11), strides=(4,4), padding='same'))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

AlexNet.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer

AlexNet.add(Conv2D(filters=256, kernel\_size=(5, 5), strides=(1,1), padding='same'))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

AlexNet.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer

AlexNet.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='same'))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

#4th Convolutional Layer

AlexNet.add(Conv2D(filters=384, kernel\_size=(3,3), strides=(1,1), padding='same'))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

#5th Convolutional Layer

AlexNet.add(Conv2D(filters=256, kernel\_size=(3,3), strides=(1,1), padding='same'))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

AlexNet.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer

AlexNet.add(Flatten())

# 1st Fully Connected Layer

AlexNet.add(Dense(4096, input\_shape=(32,32,3,)))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

# Add Dropout to prevent overfitting

AlexNet.add(Dropout(0.4))

#2nd Fully Connected Layer

AlexNet.add(Dense(4096))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

#Add Dropout

AlexNet.add(Dropout(0.4))

#3rd Fully Connected Layer

AlexNet.add(Dense(1000))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('relu'))

#Add Dropout

AlexNet.add(Dropout(0.4))

#Output Layer

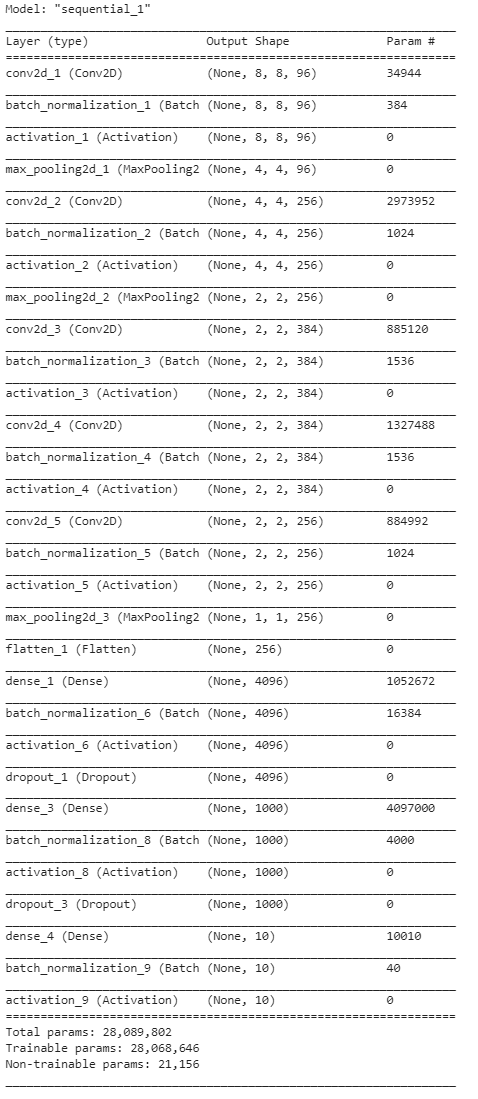
AlexNet.add(Dense(10))

AlexNet.add(BatchNormalization())

AlexNet.add(Activation('softmax'))

#Model Summary

AlexNet.summary()



# Compiling the model

AlexNet.compile(loss = keras.losses.categorical\_crossentropy, optimizer= 'adam', metrics=['accuracy'])

#Keras library for CIFAR dataset

from keras.datasets import cifar10

(x\_train, y\_train),(x\_test, y\_test)=cifar10.load\_data()

#Train-validation-test split

from sklearn.model\_selection import train\_test\_split

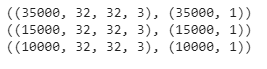
x\_train,x\_val,y\_train,y\_val=train\_test\_split(x\_train,y\_train,test\_size=.3)

#Dimension of the CIFAR10 dataset

print((x\_train.shape,y\_train.shape))

print((x\_val.shape,y\_val.shape))

print((x\_test.shape,y\_test.shape))



#Onehot Encoding the labels.

from sklearn.utils.multiclass import unique\_labels

from keras.utils import to\_categorical

#Since we have 10 classes we should expect the shape[1] of y\_train,y\_val and y\_test to change from 1 to 10

y\_train=to\_categorical(y\_train)

y\_val=to\_categorical(y\_val)

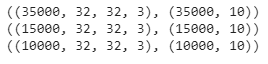
y\_test=to\_categorical(y\_test)

#Verifying the dimension after one hot encoding

print((x\_train.shape,y\_train.shape))

print((x\_val.shape,y\_val.shape))

print((x\_test.shape,y\_test.shape))



#Image Data Augmentation

from keras.preprocessing.image import ImageDataGenerator

train\_generator = ImageDataGenerator(rotation\_range=2, horizontal\_flip=True,zoom\_range=.1 )

val\_generator = ImageDataGenerator(rotation\_range=2, horizontal\_flip=True,zoom\_range=.1)

test\_generator = ImageDataGenerator(rotation\_range=2, horizontal\_flip= True,zoom\_range=.1)

#Fitting the augmentation defined above to the data

train\_generator.fit(x\_train)

val\_generator.fit(x\_val)

test\_generator.fit(x\_test)

#Defining the parameters

batch\_size= 100

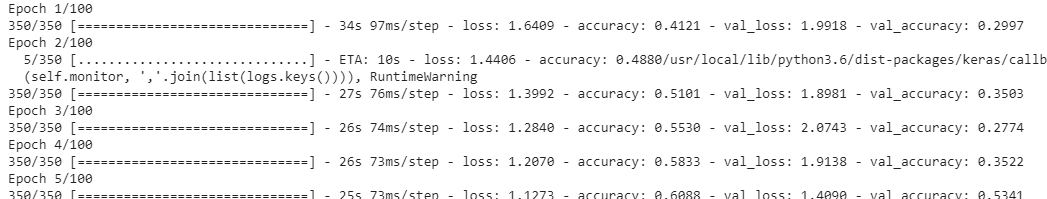
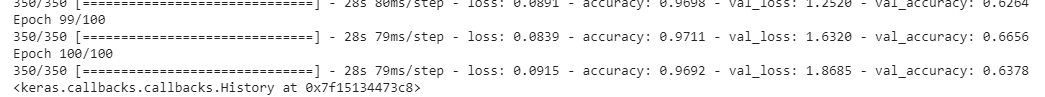
epochs=100

learn\_rate=.001

Now, we will train our defined AlexNet model.

#Training the model

AlexNet.fit\_generator(train\_generator.flow(x\_train, y\_train, batch\_size=batch\_size), epochs = epochs, steps\_per\_epoch = x\_train.shape[0]//batch\_size, validation\_data = val\_generator.flow(x\_val, y\_val, batch\_size=batch\_size), validation\_steps = 250, callbacks = [lrr], verbose=1)

#After successful training, we will visualize its performance.

import matplotlib.pyplot as plt

#Plotting the training and validation loss

f,ax=plt.subplots(2,1) #Creates 2 subplots under 1 column

#Assigning the first subplot to graph training loss and validation loss

ax[0].plot(AlexNet.history.history['loss'],color='b',label='Training Loss')

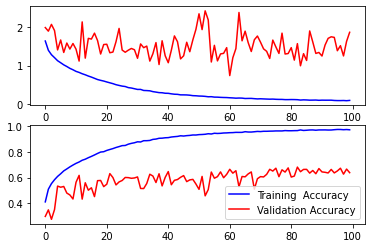
ax[0].plot(AlexNet.history.history['val\_loss'],color='r',label='Validation Loss')

#Plotting the training accuracy and validation accuracy

ax[1].plot(AlexNet.history.history['accuracy'],color='b',label='Training  Accuracy')

ax[1].plot(AlexNet.history.history['val\_accuracy'],color='r',label='Validation Accuracy')

plt.legend()



We will see the classification performance using a non-normalized and a normalized confusion matrices. For this purpose, first, we will define a function through which the confusion matrices will be plotted.

#Defining function for confusion matrix plot

def plot\_confusion\_matrix(y\_true, y\_pred, classes,

                         normalize=False,

                         title=None,

                         cmap=plt.cm.Blues):

   if not title:

       if normalize:

           title = 'Normalized confusion matrix'

       else:

           title = 'Confusion matrix, without normalization'

   # Compute confusion matrix

   cm = confusion\_matrix(y\_true, y\_pred)

   if normalize:

       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

       print("Normalized confusion matrix")

   else:

       print('Confusion matrix, without normalization')

#Print Confusion matrix

   fig, ax = plt.subplots(figsize=(7,7))

   im = ax.imshow(cm, interpolation='nearest', cmap=cmap)

   ax.figure.colorbar(im, ax=ax)

   # We want to show all ticks...

   ax.set(xticks=np.arange(cm.shape[1]),

          yticks=np.arange(cm.shape[0]),

          xticklabels=classes, yticklabels=classes,

          title=title,

          ylabel='True label',

          xlabel='Predicted label')

   # Rotate the tick labels and set their alignment.

   plt.setp(ax.get\_xticklabels(), rotation=45, ha="right",

            rotation\_mode="anchor")

   # Loop over data dimensions and create text annotations.

   fmt = '.2f' if normalize else 'd'

   thresh = cm.max() / 2.

   for i in range(cm.shape[0]):

       for j in range(cm.shape[1]):

           ax.text(j, i, format(cm[i, j], fmt),

                   ha="center", va="center",

                   color="white" if cm[i, j] > thresh else "black")

   fig.tight\_layout()

   return ax

np.set\_printoptions(precision=2)

In the next step, we will predict the class labels for the test images using the trained AlexNet model.

#Making prediction

y\_pred=AlexNet.predict\_classes(x\_test)

y\_true=np.argmax(y\_test,axis=1)

#Plotting the confusion matrix

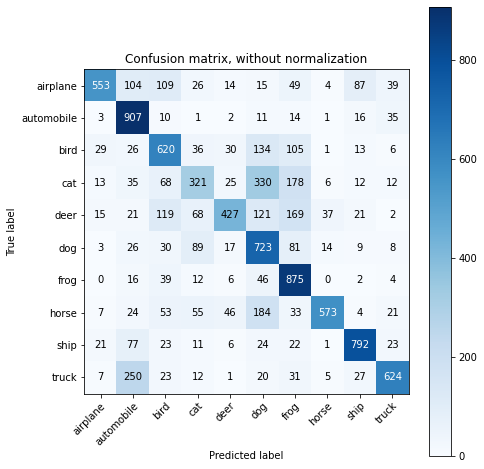
from sklearn.metrics import confusion\_matrix

confusion\_mtx=confusion\_matrix(y\_true,y\_pred)

class\_names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

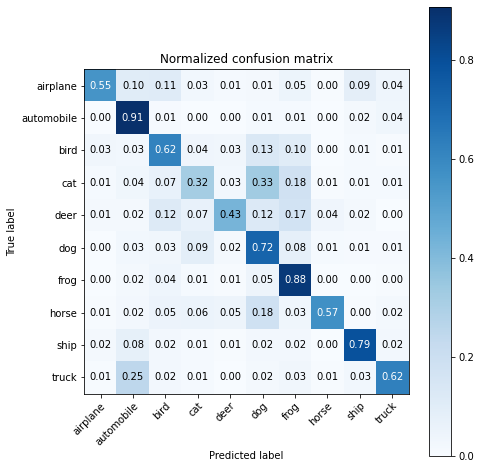
# Plotting non-normalized confusion matrix

plot\_confusion\_matrix(y\_true, y\_pred, classes = class\_names,title = 'Confusion matrix, without normalization')



# Plotting normalized confusion matrix

plot\_confusion\_matrix(y\_true, y\_pred, classes=class\_names, normalize=True, title='Normalized confusion matrix')



The average accuracy score in classifying the unseen test data will be obtained now.

#Classification accuracy

from sklearn.metrics import accuracy\_score

acc\_score = accuracy\_score(y\_true, y\_pred)

print('Accuracy Score = ', acc\_score)

alexnet accuracy score