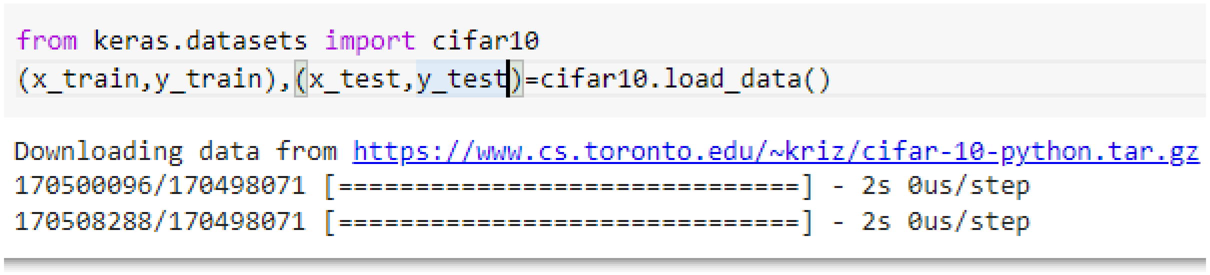
Deep Learning Project Report

Mian Muaz Razaq
2021327118

Architecture Used:

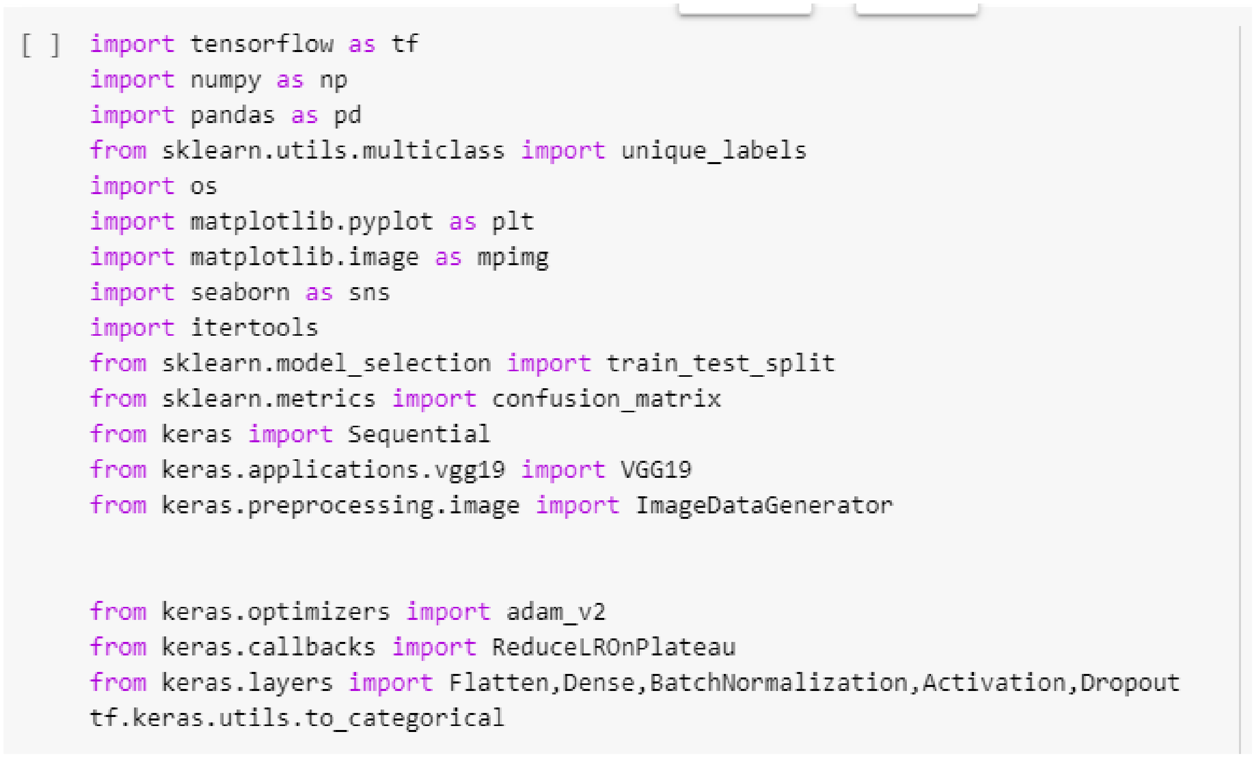
VGG19 (**FINE-TUNED**)

1. First I imported cifar10 dataset from Keras datasets.



```
from keras.datasets import cifar10
(x_train,y_train),(x_test,y_test)=cifar10.load_data()

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

2. Then, I imported all the necessary Packages which will be needed in completion of this project.

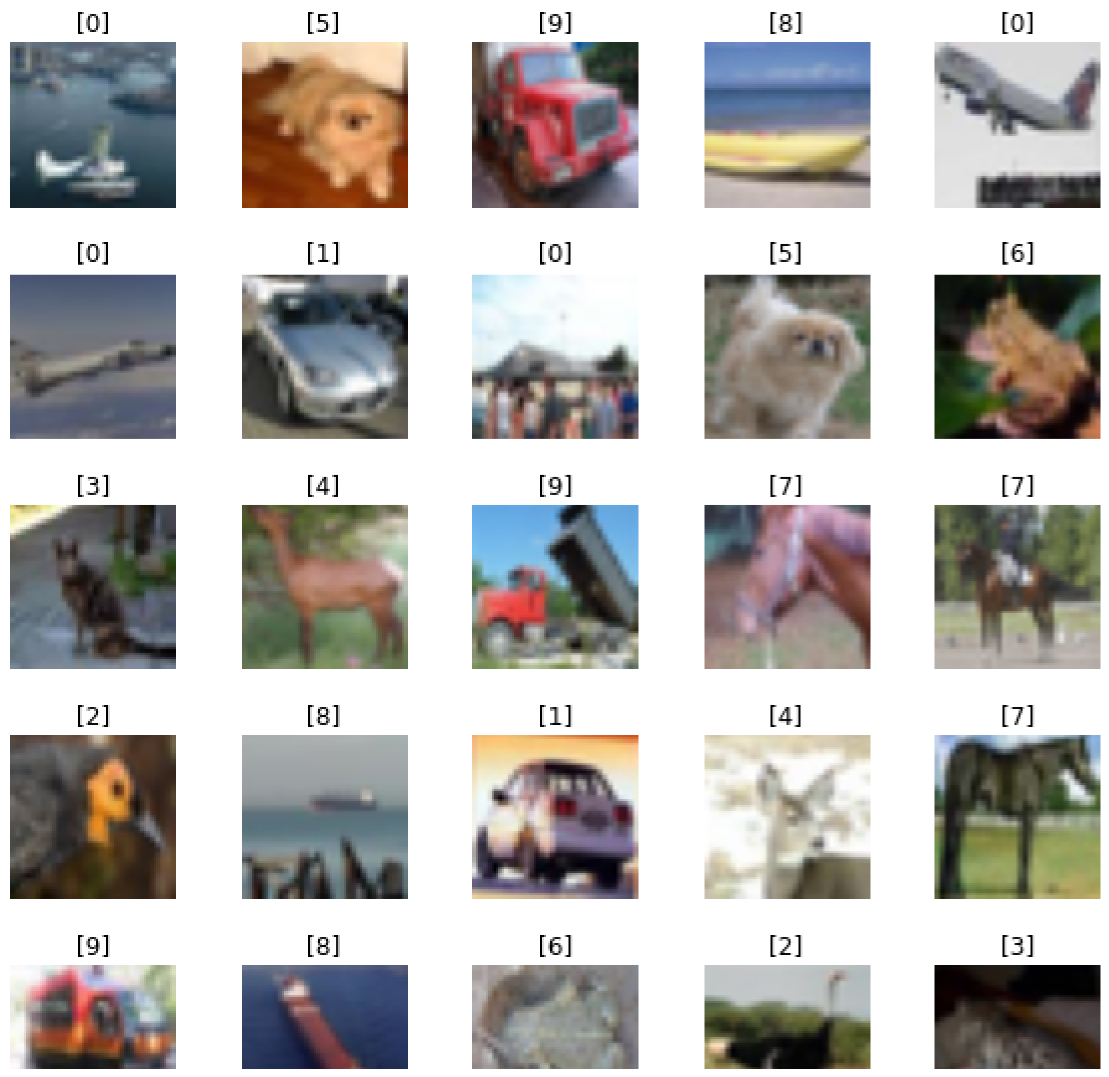


```
[ ]  import tensorflow as tf
     import numpy as np
     import pandas as pd
     from sklearn.utils.multiclass import unique_labels
     import os
     import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     import seaborn as sns
     import itertools
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix
     from keras import Sequential
     from keras.applications.vgg19 import VGG19
     from keras.preprocessing.image import ImageDataGenerator


     from keras.optimizers import adam_v2
     from keras.callbacks import ReduceLROnPlateau
     from keras.layers import Flatten,Dense,BatchNormalization,Activation,Dropout
     tf.keras.utils.to_categorical
```

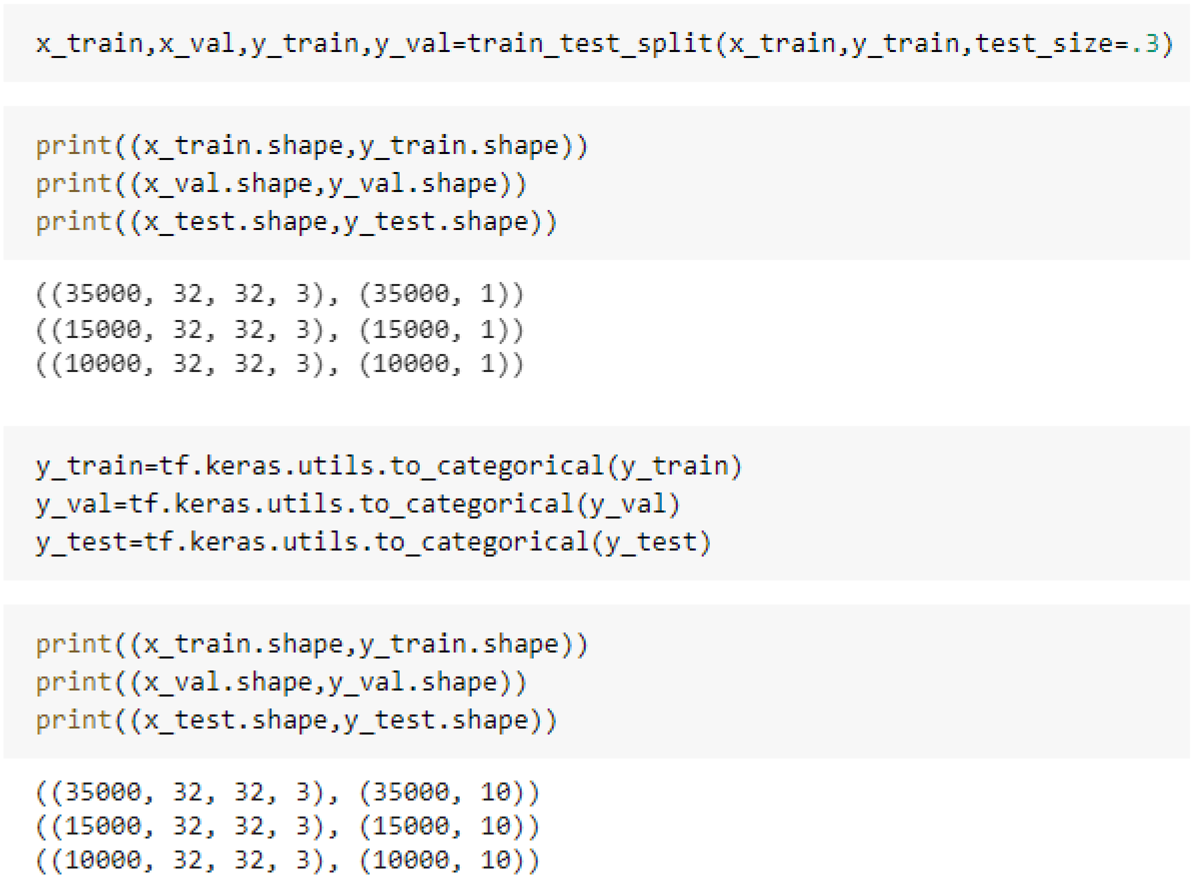
3. Then for checking purpose I displayed the pictures with their respected labels i.e. to which class they belong

```python
W_grid=5
L_grid=5
fig,axes = plt.subplots(L_grid,W_grid,figsize=(10,10))
axes=axes.ravel()
n_training=len(x_train)
for i in np.arange(0,L_grid * W_grid):
    index=np.random.randint(0,n_training)
    axes[i].imshow(x_train[index])
    axes[i].set_title(y_train[index])
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/text.py:1165: FutureWar
  if s != self._text:
```



4. Then I split the dataset into training, validation and testing data. 35000 images were assigned to training, 15000 for validation and 10000 for testing. After that converted y_train, y_val and y_test to one-hot vector.



```
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=.3)
```

```
print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

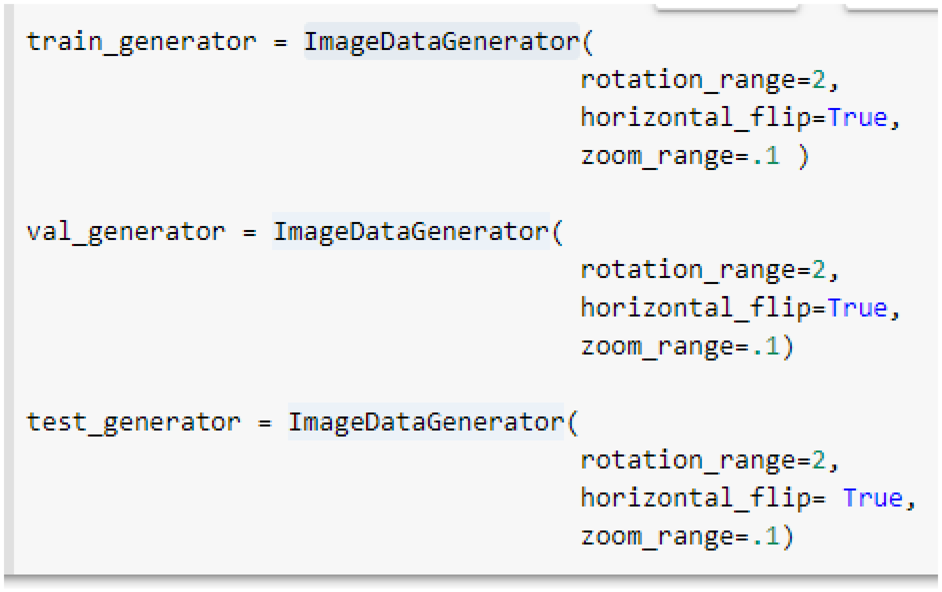
```
((35000, 32, 32, 3), (35000, 1))
((15000, 32, 32, 3), (15000, 1))
((10000, 32, 32, 3), (10000, 1))
```

```
y_train=tf.keras.utils.to_categorical(y_train)
y_val=tf.keras.utils.to_categorical(y_val)
y_test=tf.keras.utils.to_categorical(y_test)
```

```
print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

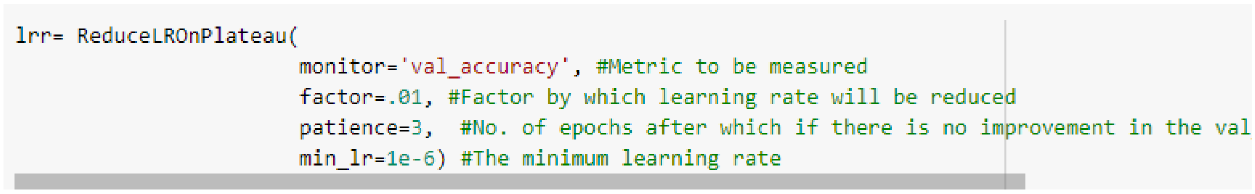
```
((35000, 32, 32, 3), (35000, 10))
((15000, 32, 32, 3), (15000, 10))
((10000, 32, 32, 3), (10000, 10))
```

5. Then I did Data Augmentation so that the model trains well by flipping, rotating and zooming the pictures.



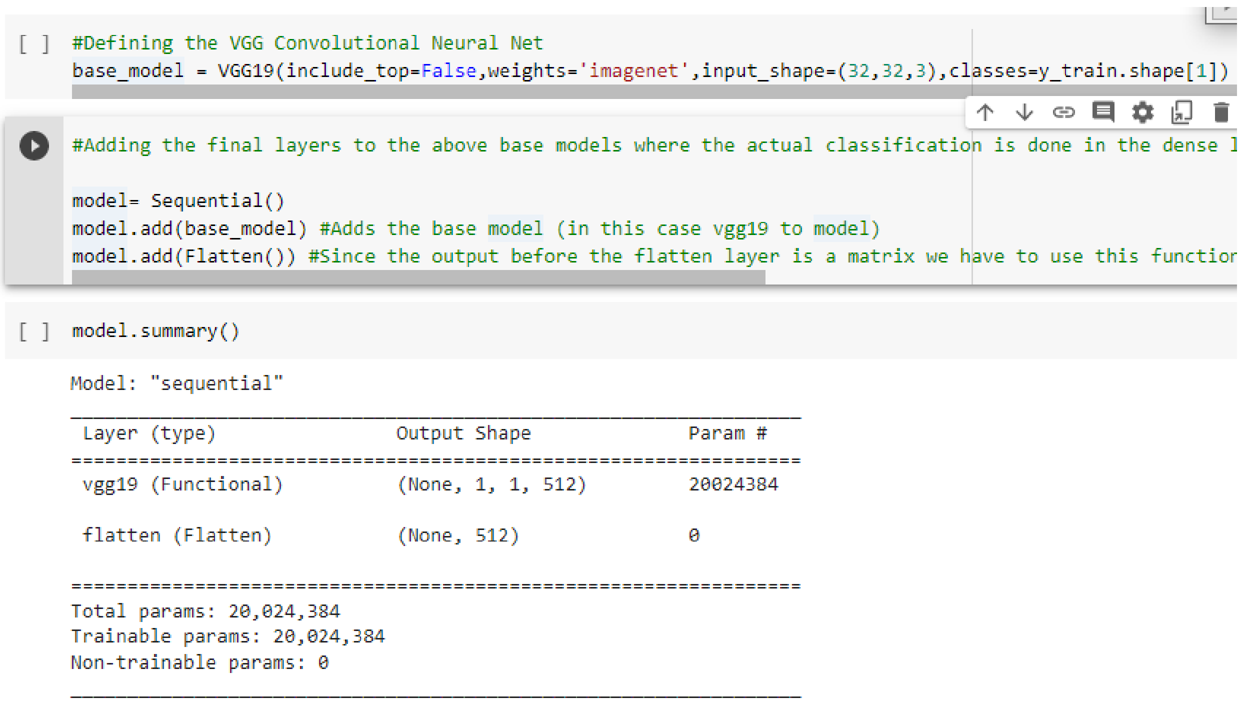
```
train_generator = ImageDataGenerator(
                                    rotation_range=2,
                                    horizontal_flip=True,
                                    zoom_range=.1 )

val_generator = ImageDataGenerator(
                                    rotation_range=2,
                                    horizontal_flip=True,
                                    zoom_range=.1)

test_generator = ImageDataGenerator(
                                    rotation_range=2,
                                    horizontal_flip= True,
                                    zoom_range=.1)
```

6. I used Learning rate reduction so that the learning rate reduces after it feels that the model is saturated and is not improving.



```
lrr= ReduceLROnPlateau(
                       monitor='val_accuracy', #Metric to be measured
                       factor=.01, #Factor by which learning rate will be reduced
                       patience=3,  #No. of epochs after which if there is no improvement in the val
                       min_lr=1e-6) #The minimum learning rate
```

7. Base Model used is VGG19, and the model is sequential

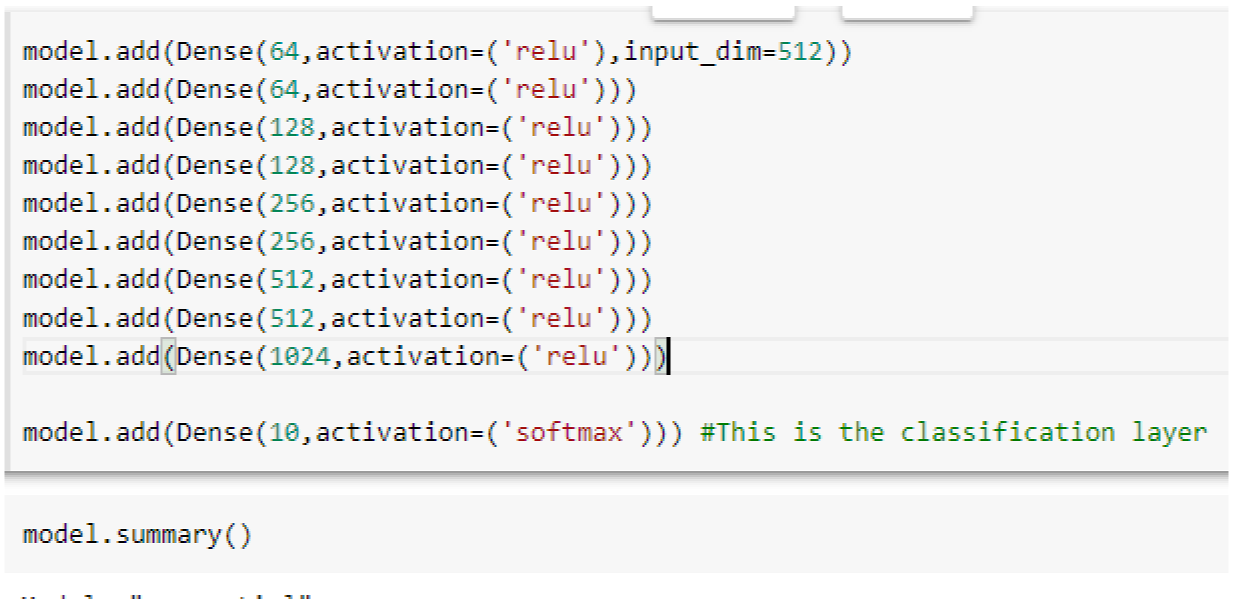


```
[ ]  #Defining the VGG Convolutional Neural Net
     base_model = VGG19(include_top=False,weights='imagenet',input_shape=(32,32,3),classes=y_train.shape[1])
```

```
     #Adding the final layers to the above base models where the actual classification is done in the dense l

     model= Sequential()
     model.add(base_model) #Adds the base model (in this case vgg19 to model)
     model.add(Flatten()) #Since the output before the flatten layer is a matrix we have to use this functior
```

```
[ ]  model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      vgg19 (Functional)          (None, 1, 1, 512)         20024384

      flatten (Flatten)           (None, 512)               0

     =================================================================
     Total params: 20,024,384
     Trainable params: 20,024,384
     Non-trainable params: 0
     _________________________________________________________________
```

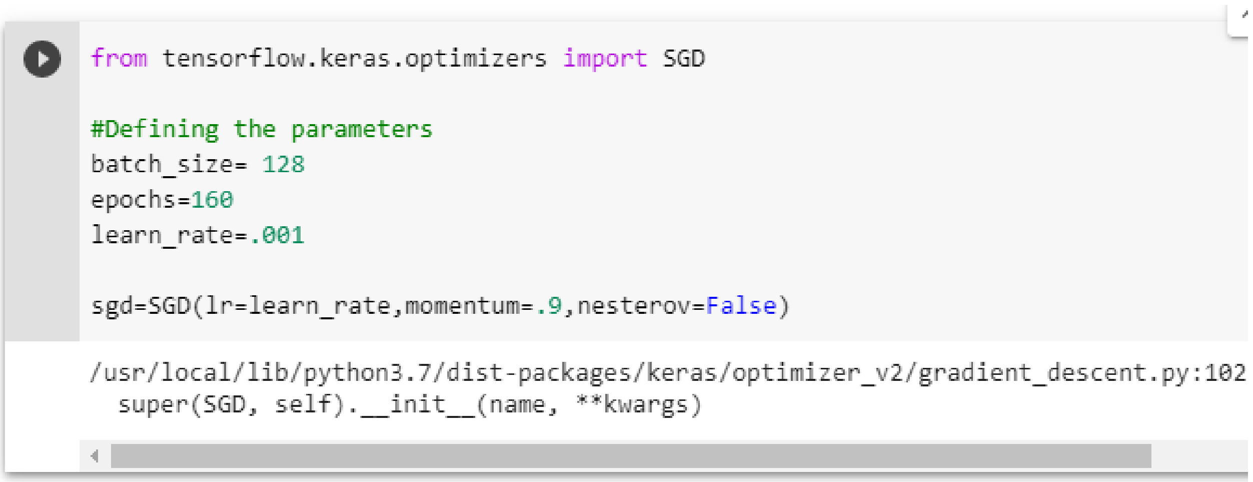
8. I added Dense layers to the model to increase efficiency, the output layer has 10 neurons as we have only 10 classes, the activation function used is RELU and at the output layer is SOFTMAX.



```
model.add(Dense(64,activation=('relu'),input_dim=512))
model.add(Dense(64,activation=('relu')))
model.add(Dense(128,activation=('relu')))
model.add(Dense(128,activation=('relu')))
model.add(Dense(256,activation=('relu')))
model.add(Dense(256,activation=('relu')))
model.add(Dense(512,activation=('relu')))
model.add(Dense(512,activation=('relu')))
model.add(Dense(1024,activation=('relu')))

model.add(Dense(10,activation=('softmax'))) #This is the classification layer
```

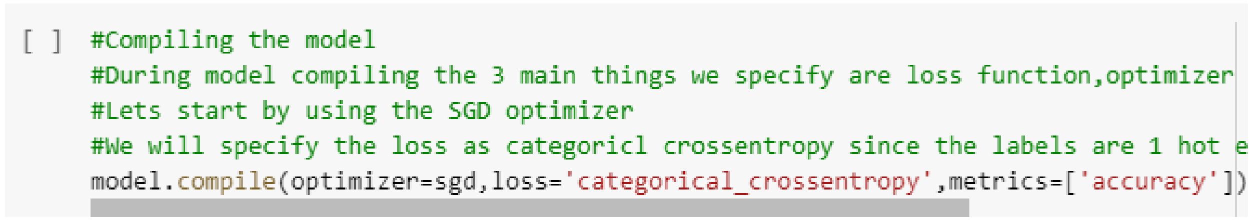
```
model.summary()
```

9. I define the parameters and Optimizer that is Stochastic Gradient Descent (I also used ADAM but the accuracy for that was low, so I then choose SGD), as the dataset is of multi-classes so I used categorical cross entropy rather than the binary cross entropy.



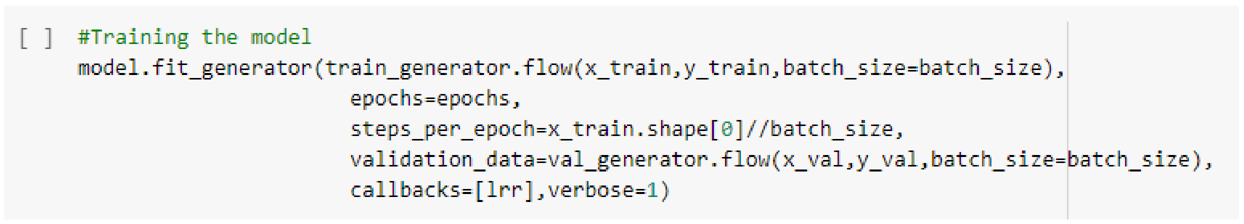
```
from tensorflow.keras.optimizers import SGD

#Defining the parameters
batch_size= 128
epochs=160
learn_rate=.001

sgd=SGD(lr=learn_rate,momentum=.9,nesterov=False)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/gradient_descent.py:102
  super(SGD, self).__init__(name, **kwargs)
```



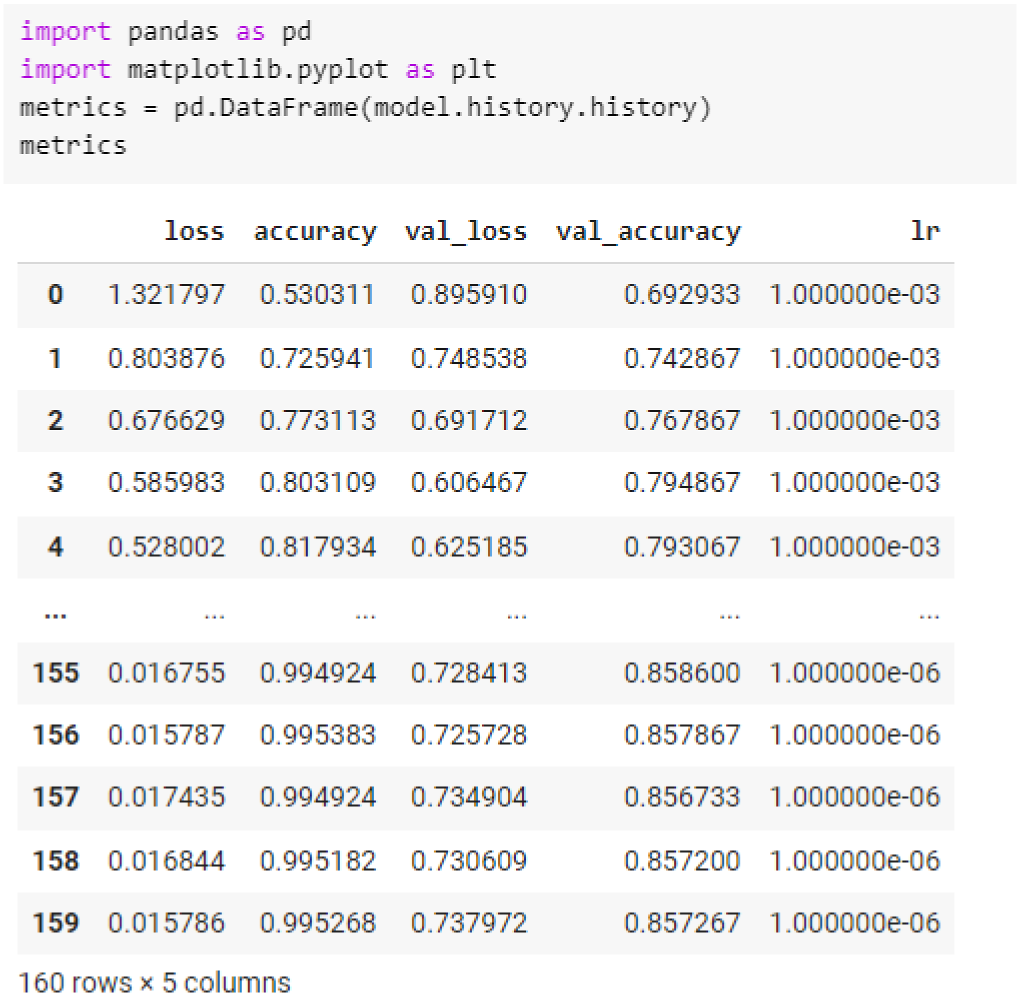
```
[ ] #Compiling the model
    #During model compiling the 3 main things we specify are loss function,optimizer
    #Lets start by using the SGD optimizer
    #We will specify the loss as categoricl crossentropy since the labels are 1 hot e
    model.compile(optimizer=sgd,loss='categorical_crossentropy',metrics=['accuracy'])
```

10. After defining the hyper parameters and Optimizer I trained my model.



```
[ ] #Training the model
    model.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),
                        callbacks=[lrr],verbose=1)
```

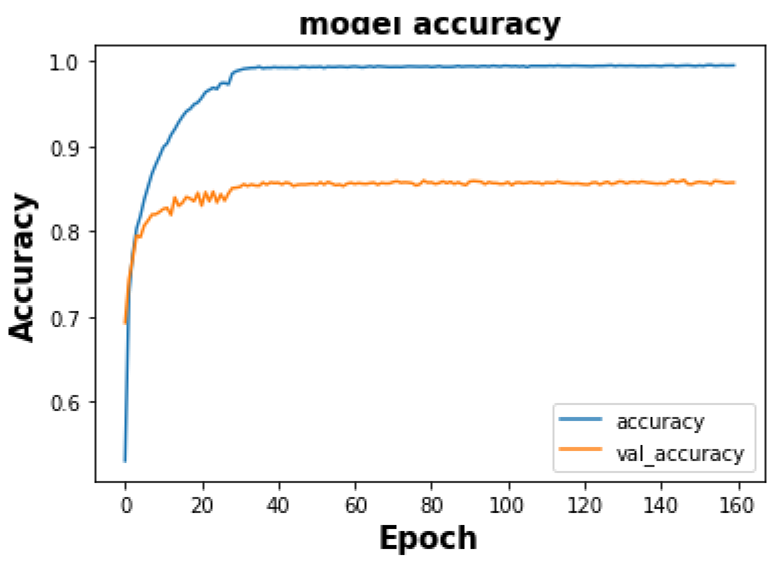
11. I got **VALIDATION** accuracy of 85.73 for my model

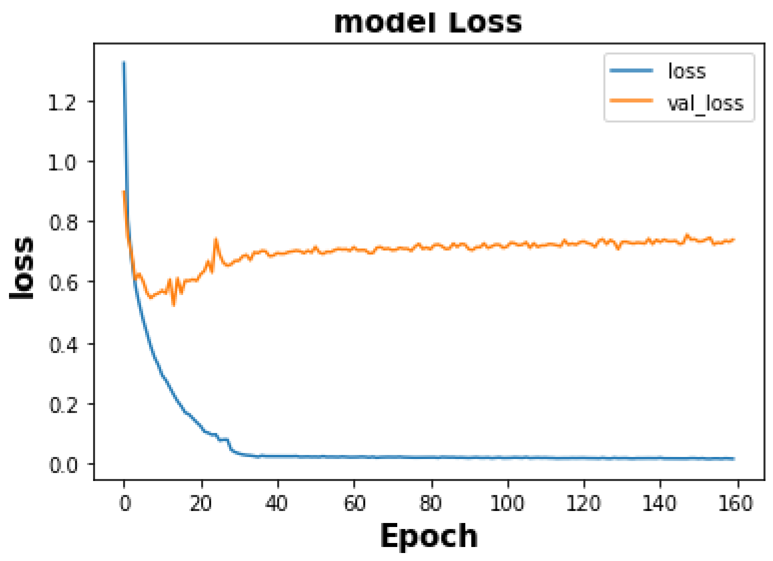


```python
import pandas as pd
import matplotlib.pyplot as plt
metrics = pd.DataFrame(model.history.history)
metrics
```

	loss	accuracy	val_loss	val_accuracy	lr
0	1.321797	0.530311	0.895910	0.692933	1.000000e-03
1	0.803876	0.725941	0.748538	0.742867	1.000000e-03
2	0.676629	0.773113	0.691712	0.767867	1.000000e-03
3	0.585983	0.803109	0.606467	0.794867	1.000000e-03
4	0.528002	0.817934	0.625185	0.793067	1.000000e-03
...	...	...	...	...	...
155	0.016755	0.994924	0.728413	0.858600	1.000000e-06
156	0.015787	0.995383	0.725728	0.857867	1.000000e-06
157	0.017435	0.994924	0.734904	0.856733	1.000000e-06
158	0.016844	0.995182	0.730609	0.857200	1.000000e-06
159	0.015786	0.995268	0.737972	0.857267	1.000000e-06

160 rows × 5 columns

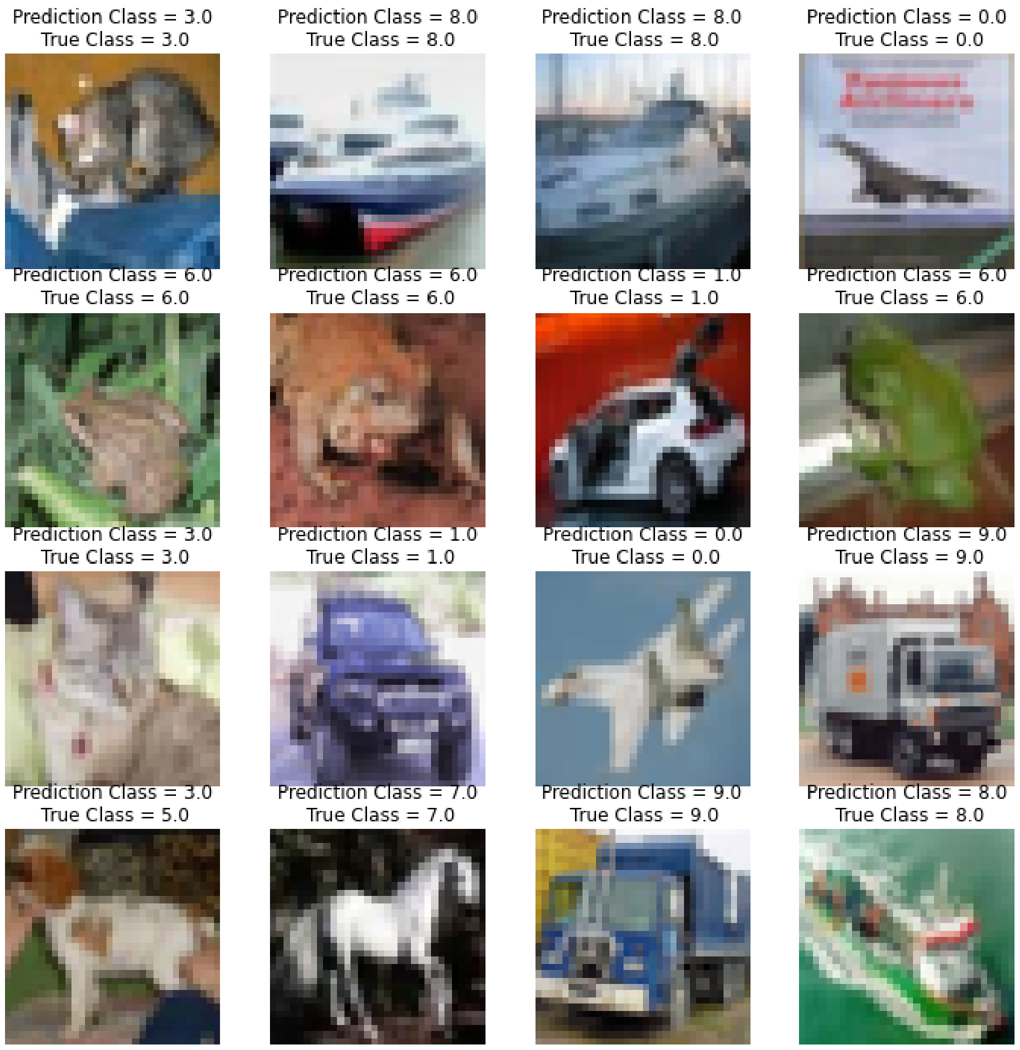




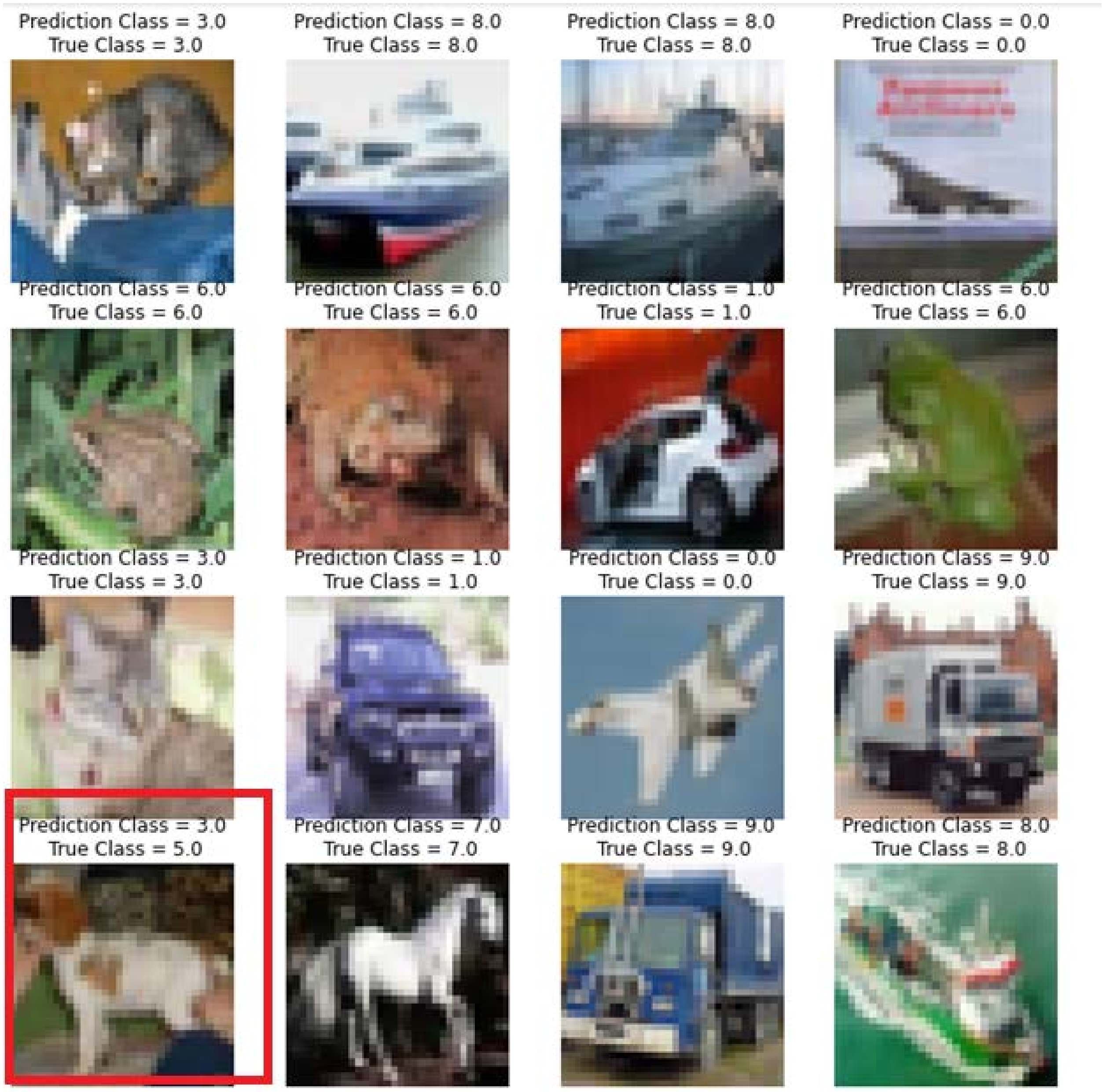
11. Next I also used predictions to check my model and many of predictions were right apart from some

```
[ ]  #Making prediction
     y_predict = np.argmax(model.predict(x_test), axis=-1)
     y_true=np.argmax(y_test,axis=1)
```

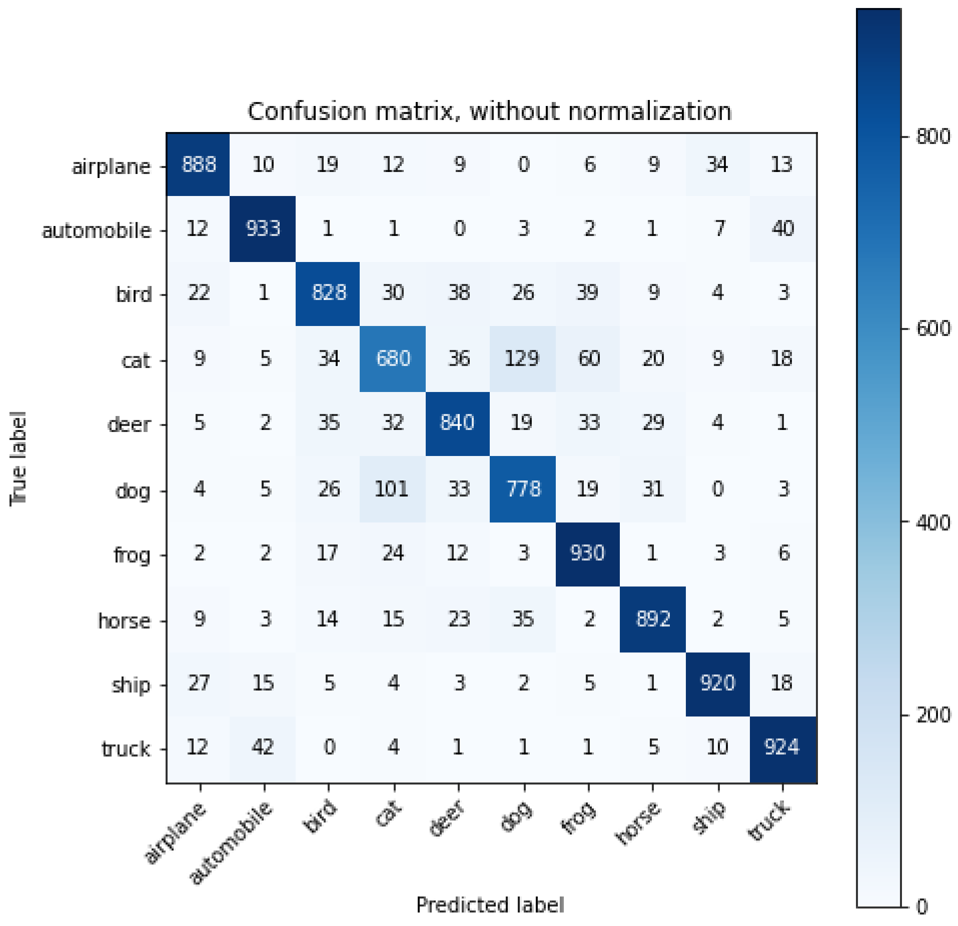
```
[ ]  L = 4
     W = 4
     fig, axes = plt.subplots(L, W, figsize = (12,12))
     axes = axes.ravel()

     for i in np.arange(0, L * W):
         axes[i].imshow(x_test[i])
         axes[i].set_title(f"Prediction Class = {y_predict[i]:0.1f}\n True Class = {y_true[i]:0.1f}")
         axes[i].axis('off')
```



Few wrong prediction in TEST data (RED Rectangle):



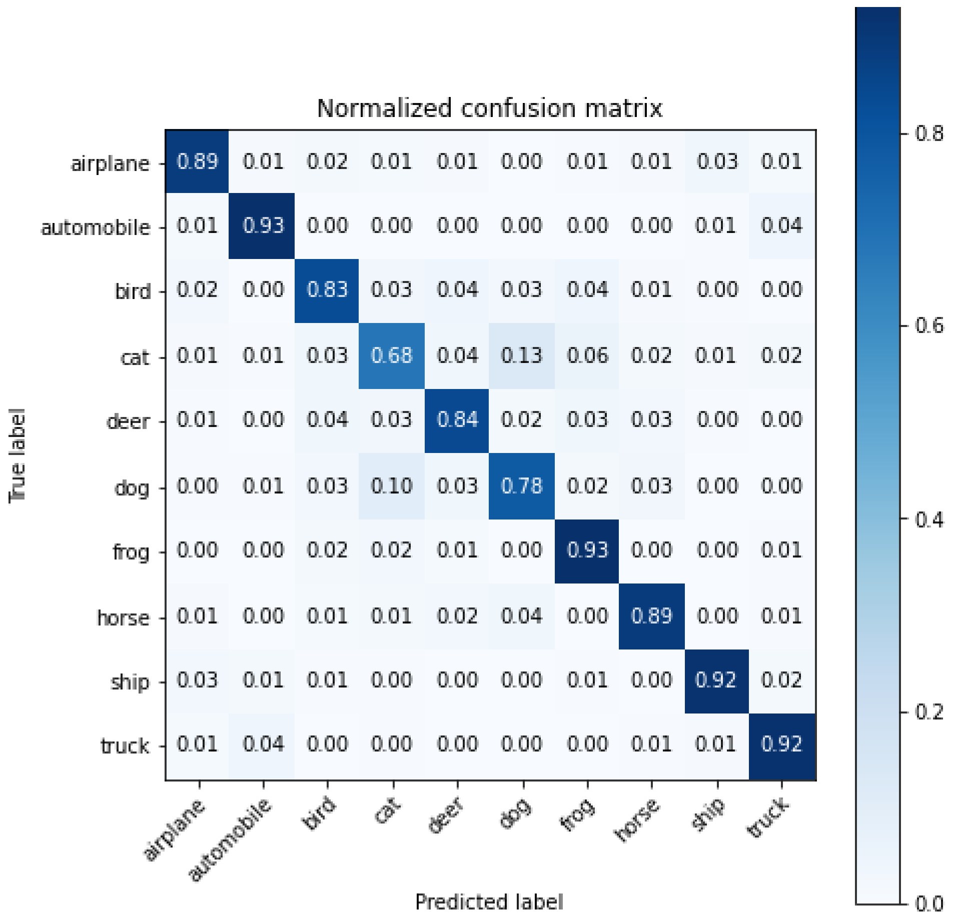
The confusion Matrix of my results are as follow:



True label \ Predicted label	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	888	10	19	12	9	0	6	9	34	13
automobile	12	933	1	1	0	3	2	1	7	40
bird	22	1	828	30	38	26	39	9	4	3
cat	9	5	34	680	36	129	60	20	9	18
deer	5	2	35	32	840	19	33	29	4	1
dog	4	5	26	101	33	778	19	31	0	3
frog	2	2	17	24	12	3	930	1	3	6
horse	9	3	14	15	23	35	2	892	2	5
ship	27	15	5	4	3	2	5	1	920	18
truck	12	42	0	4	1	1	1	5	10	924

The diagonal values are the TRUE predictions while all others are wrong predictions of my Trained Model.

The Normalized Confusion Matrix is as Follow:



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.89	0.01	0.02	0.01	0.01	0.00	0.01	0.01	0.03	0.01
automobile	0.01	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.04
bird	0.02	0.00	0.83	0.03	0.04	0.03	0.04	0.01	0.00	0.00
cat	0.01	0.01	0.03	0.68	0.04	0.13	0.06	0.02	0.01	0.02
deer	0.01	0.00	0.04	0.03	0.84	0.02	0.03	0.03	0.00	0.00
dog	0.00	0.01	0.03	0.10	0.03	0.78	0.02	0.03	0.00	0.00
frog	0.00	0.00	0.02	0.02	0.01	0.00	0.93	0.00	0.00	0.01
horse	0.01	0.00	0.01	0.01	0.02	0.04	0.00	0.89	0.00	0.01
ship	0.03	0.01	0.01	0.00	0.00	0.00	0.01	0.00	0.92	0.02
truck	0.01	0.04	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.92

COMMENTS:

From the normalized Confusion Matrix we can observe how much Percentage of Each Class Predictions are Accurate and Right (Shown on the Diagonal), Here it should be noted that our trained Model did the least accurate predictions on **DOG** and **CAT** class, as it is confused due to its similar traits (That is one of the main reason why our model validation accuracy is lower i.e. 85.73), It can be further improved.