İZMİR
KÂTİP ÇELEBİ
UNIVERSITY

2010

Izmir Katip Celebi University

Department of Electrical and Electronics Engineering

EEE420 - INTRODUCTION TO DEEP LEARNING

MARBLE CLASSIFICATION WITH DEEP LEARNING

Project Report

Mehmet Sefa Senbani 150403031

CONTENT

1) ABSTRACT

In this project, it is aimed to deep learning based marble classification. Using data which included marble images to train and test. This data has 516 different marble images. In this project this data has been trained with different deep learning algorithms in Matlab and Pyhton environment. 12 different models trained with Matlab and 6 different model trained in Pyhton which 5 are common. Models compared in each other and also with Pyhton and Matlab.

2) STAGES OF THE PROJECT

2.1) Dataset

In this project used dataset which has 516 different marble images with 5 different classes as beige-blue dark, beige- emp, beige-gri, trv-classic, trv-noce.

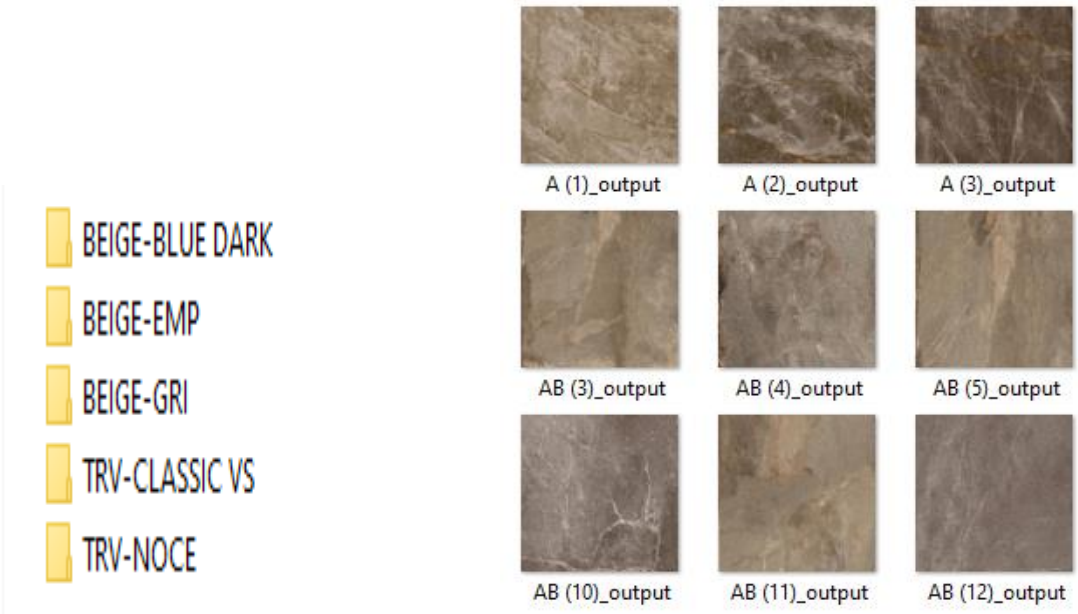

Figure.1 Sample marble images from dataset

2.2) Matlab

In this project firsly used Matlab Deep Learning Toolbox for understanding deep learing structure and faster results. 12 different models trained with the marble data which are Xception, GoggleNet, Resnet18, ResNet50, Resnet101, Alexnet, DarkNet19, DarkNet53, Vgg16, Vgg19, Inception, MobileNet.

In matlab firstly has been loaded marble images which shown below.

```
%LOAD DATA
digitDatasetPath = fullfile('D:\_MASAÜSTÜ\DATASET_MARBLE');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

Figure.2 A part of code (Matlab)

Then the data was splitted with 0.3 ratio to test and others train.

```
%Split the data into 70% training and 30% test data
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

Figure.3 A part of code (Matlab)

Thirdly, using Matlab Deep Network Designer Toolbox created the layers and weights.

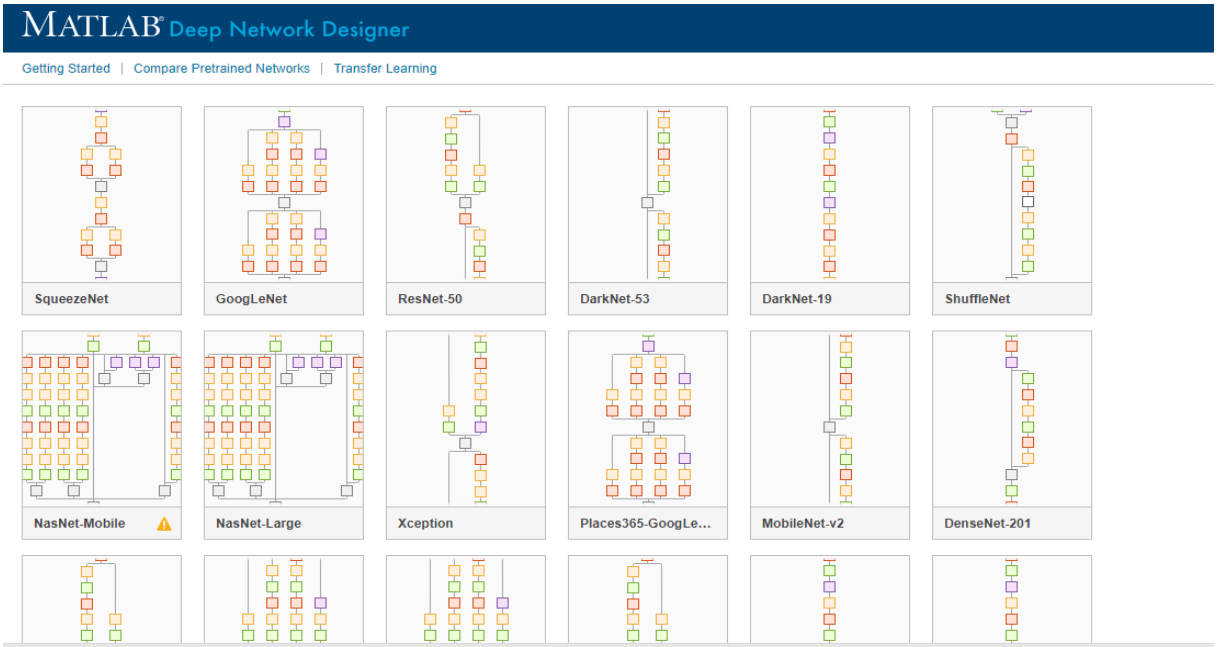

Figure.4 Interface of Deep Network Designer

```
tempLayers = [
    imageInputLayer([224 224 3],"Name","Input_gpu_0|data_0","Normalization","zscore")
    convolution2dLayer([3 3],24,"Name","node_1","Padding",[1 1 1 1],"Stride",[2 2])
    batchNormalizationLayer("Name","node_2")
    reluLayer("Name","node_3")
    maxPooling2dLayer([3 3],"Name","node_4","Padding",[1 1 1 1],"Stride",[2 2])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    groupedConvolution2dLayer([1 1],28,4,"Name","node_5")
    batchNormalizationLayer("Name","node_6")
    reluLayer("Name","node_7")
    helperNnetShufflenetLayerChannelShufflingLayer("shuffle_8to10",4,"in","out")
    groupedConvolution2dLayer([3 3],1,112,"Name","node_11","Padding",[1 1 1 1],"Stride",[2 2])
    batchNormalizationLayer("Name","node_12")
    groupedConvolution2dLayer([1 1],28,4,"Name","node_13")
    batchNormalizationLayer("Name","node_14")];
```

Figure.5 Sample code of created with Toolbox

After train ended the confussion matrix and random predict images created then trained model was saved.

```
%Confusion Matrix
plotconfusion(imdsValidation.Labels,YPred)


%Predict random images
idx = randperm(numel(imdsValidation.Files),16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsValidation,idx(i));
    imshow(I)
    label = YPred(idx(i));
    title(string(label) + ", " + num2str(100*max(scores(idx(i),:)),3) + "%");
end

%Save Model
DenseNet_Model_1 = net;
save DenseNet_Model_1
```

Figure.6 Final part of code

2.2.1) Xception

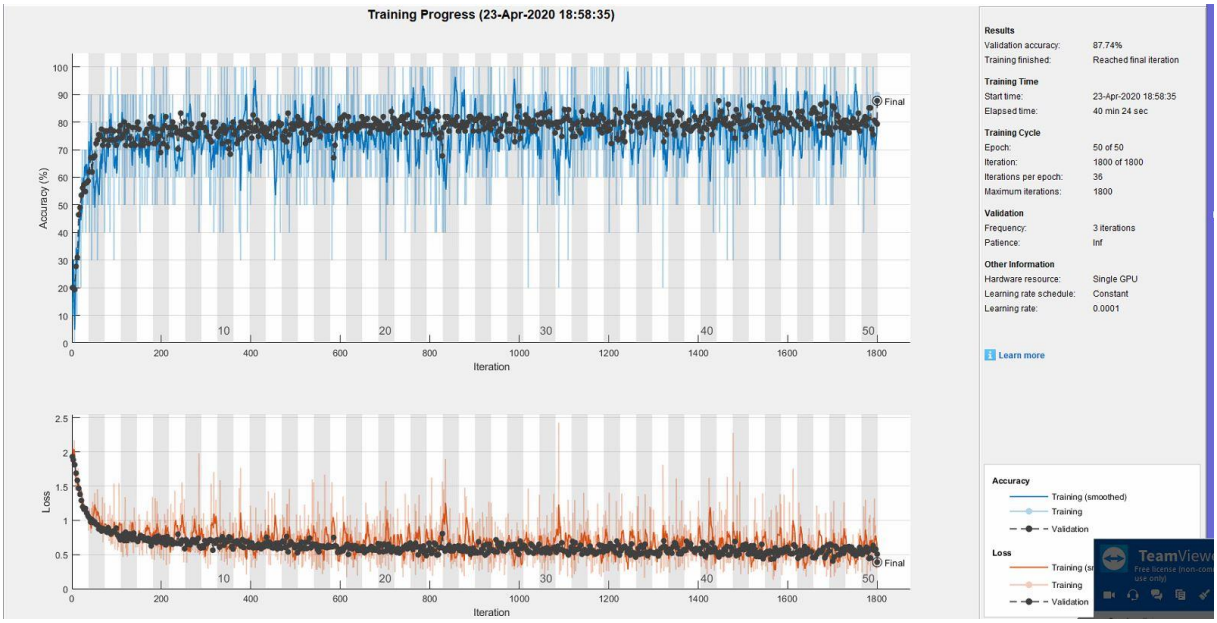

Figure.7 Xception Training Result

Figure.8 Xception Confussion Matrix

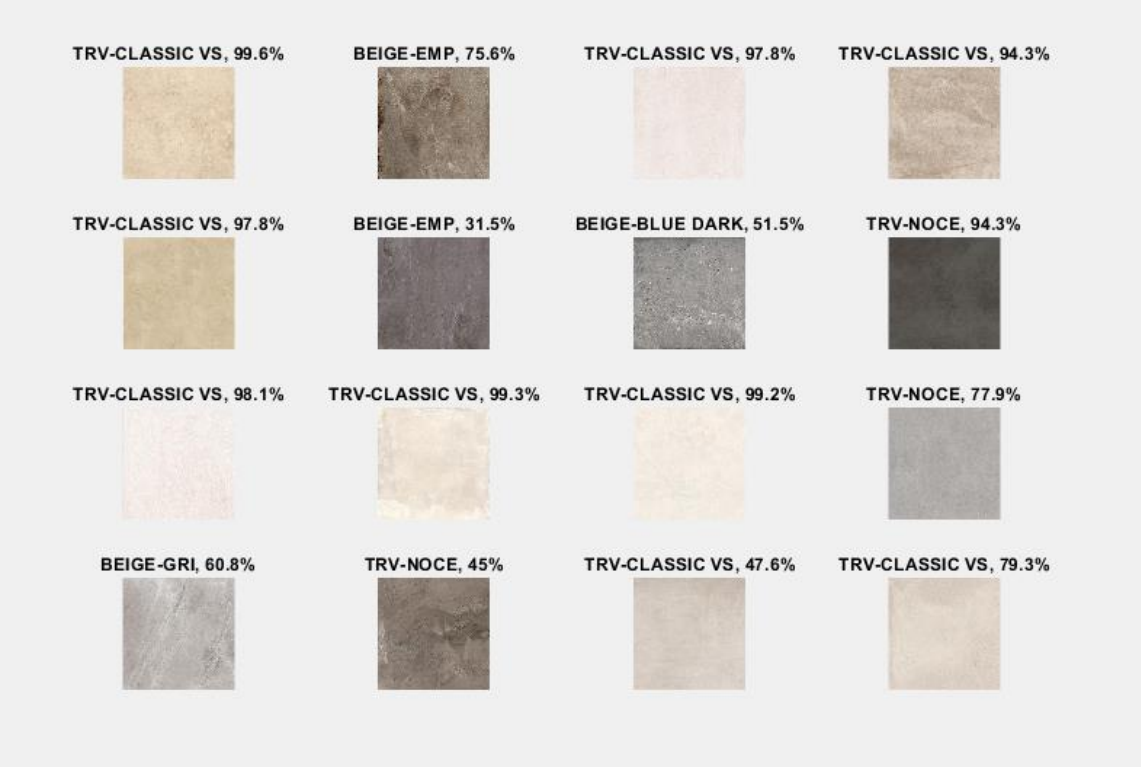

Figure.9 Xception Prediction of Random Images

2.2.2) VGG16

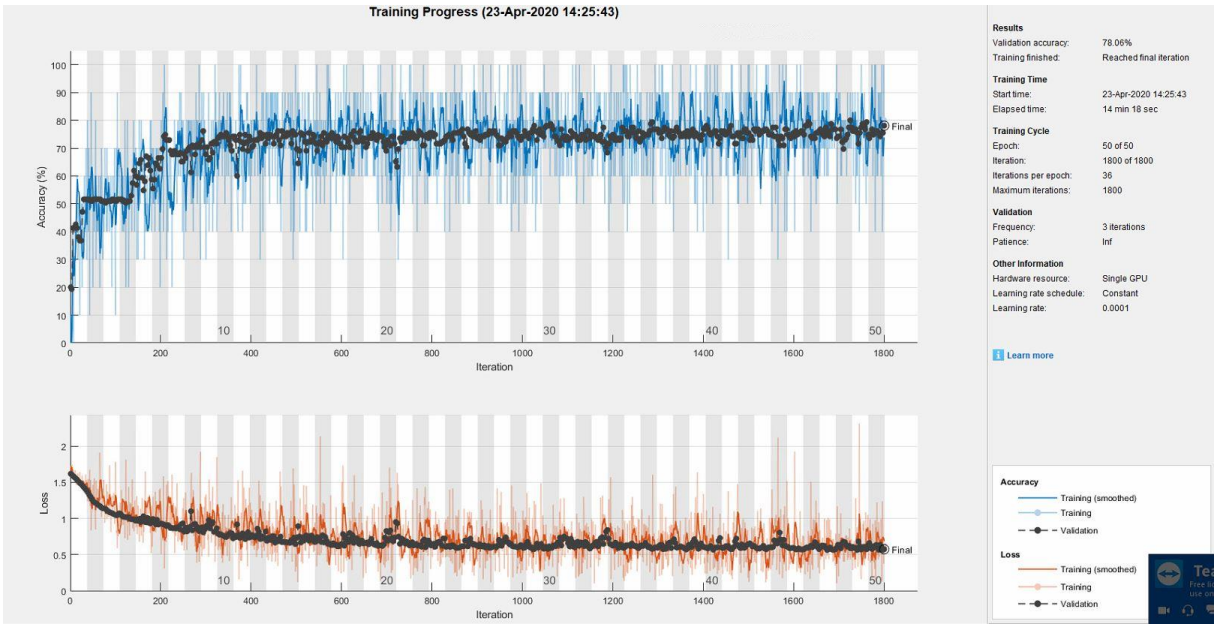

Figure.10 Vgg16 Training Result

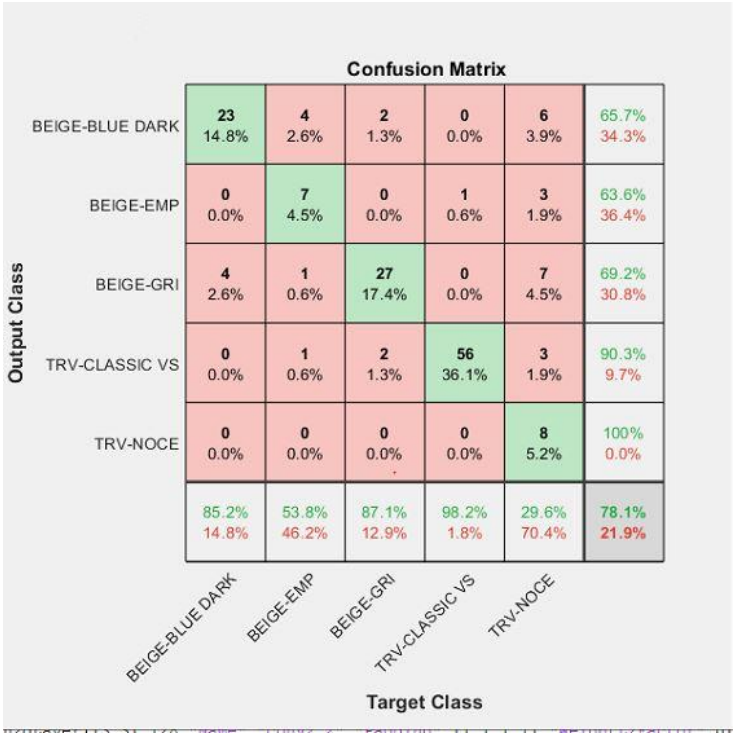

Figure.11 Vgg16 Confussion Matrix

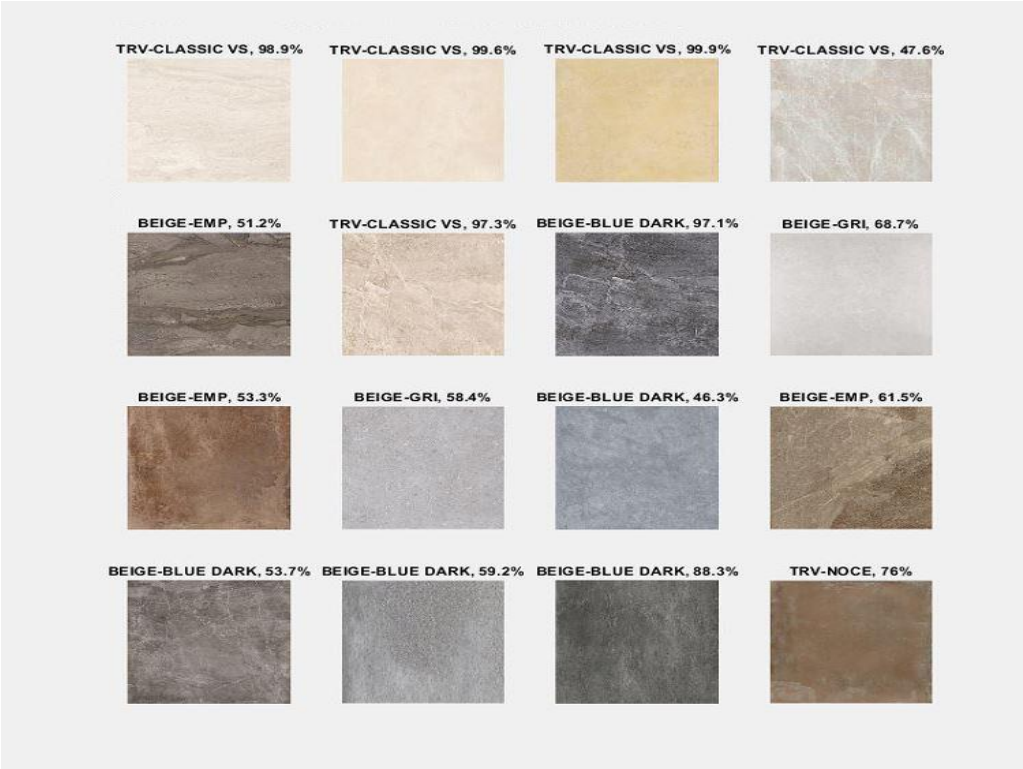

Figure.12 Vgg16 Prediction of Random Images

2.2.3) VGG19

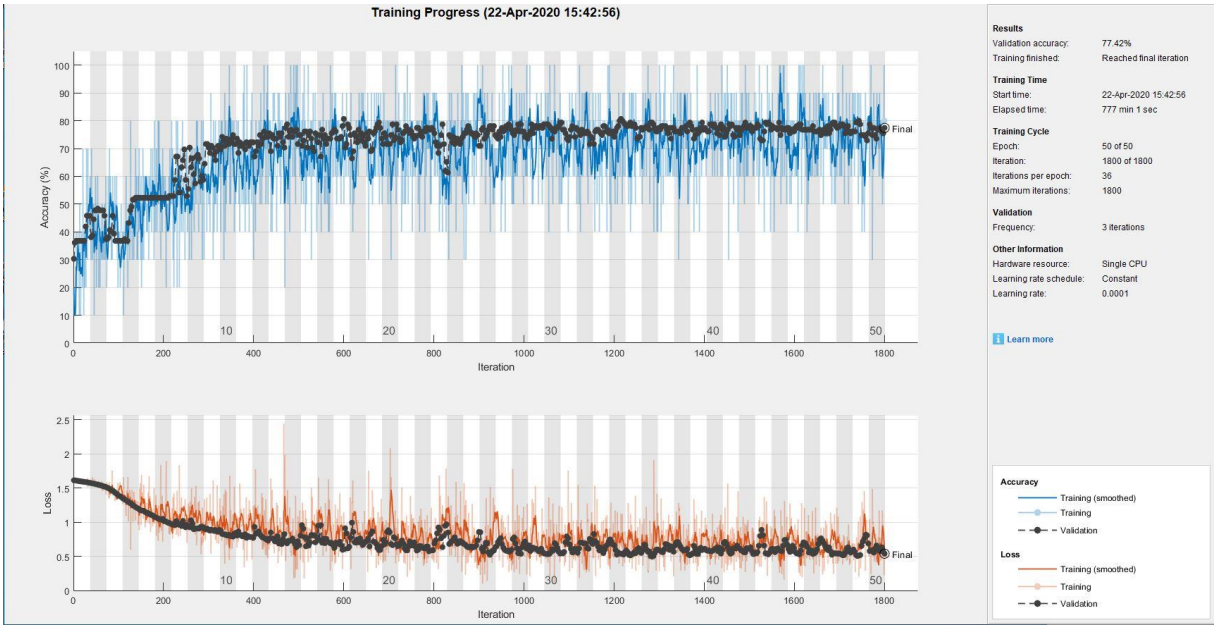

Figure.13 Vgg19 Training Result

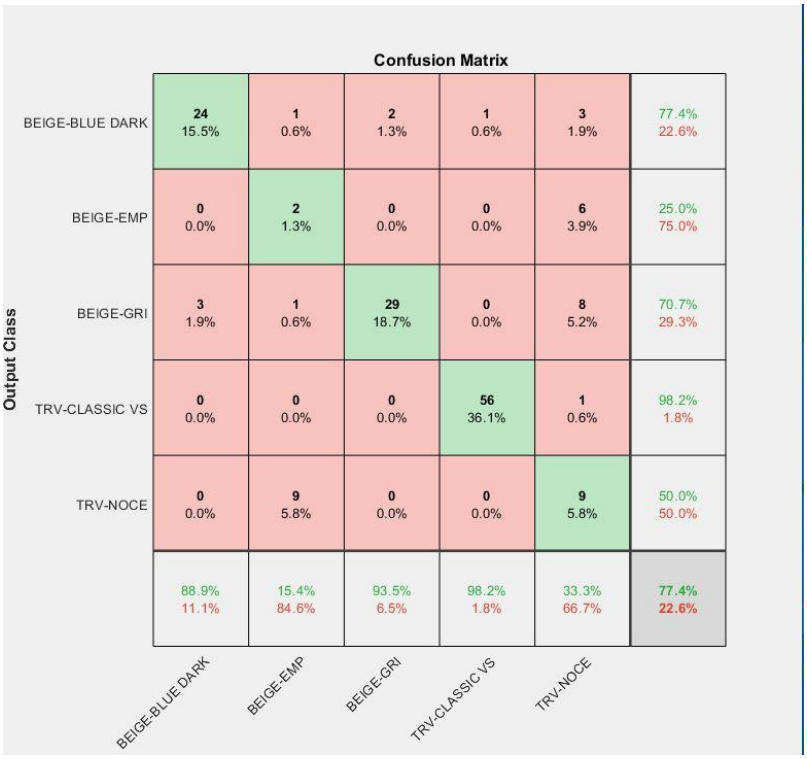

Figure.14 Vgg19 Confussion Matrix

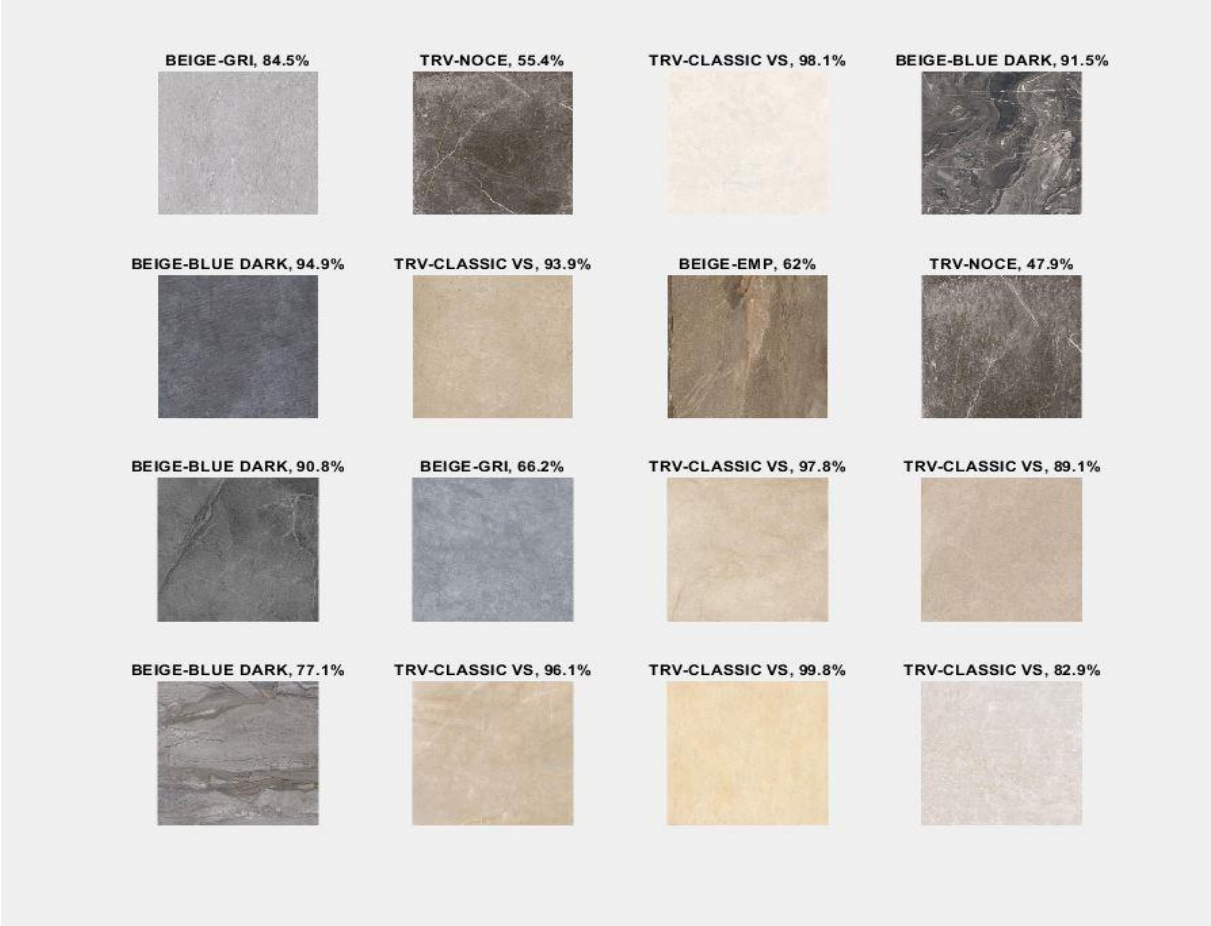

Figure.15 Vgg19 Prediction of Random Images

2.2.4) ResNet18

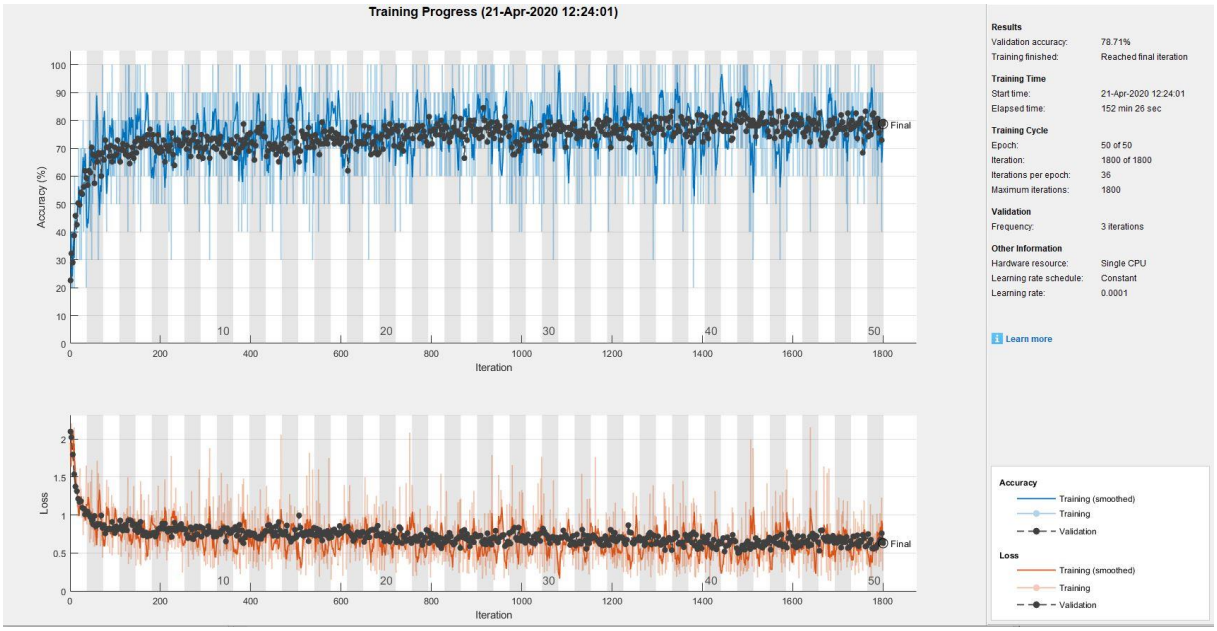

Figure.16 Resnet18 Training Result

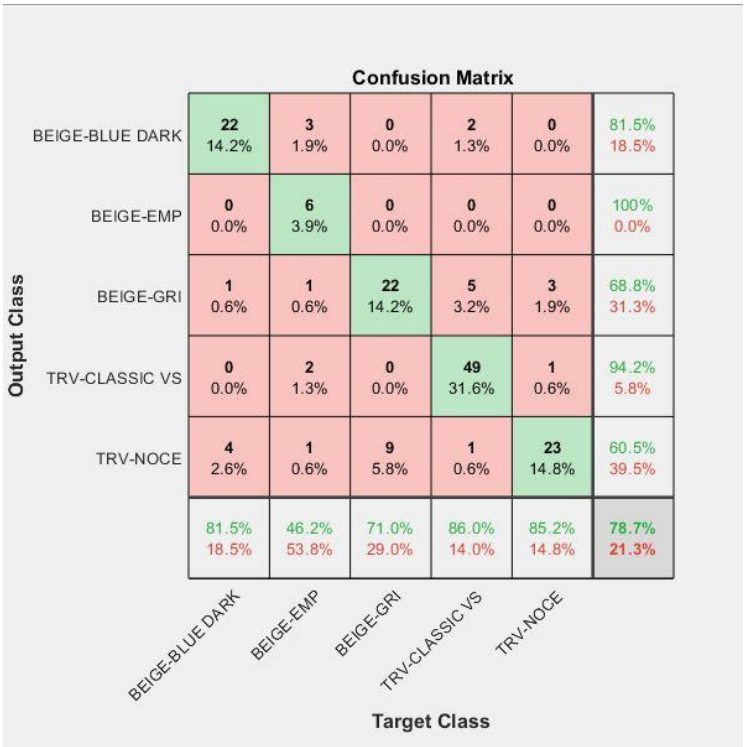

Figure.17 ResNet18 Confussion Matrix

Figure.18 Resnet18 Prediction of Random Images

2.2.5) Resnet50

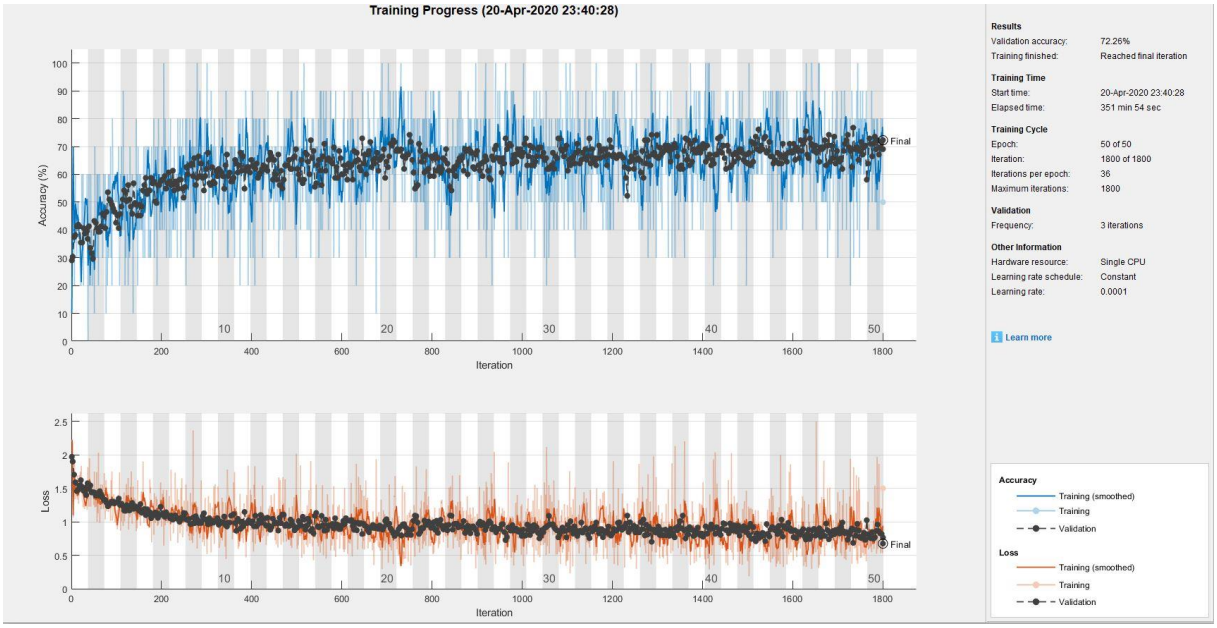

Figure.19 Resnet50 Training Result

Figure.20 Resnet50 Confussion Matrix

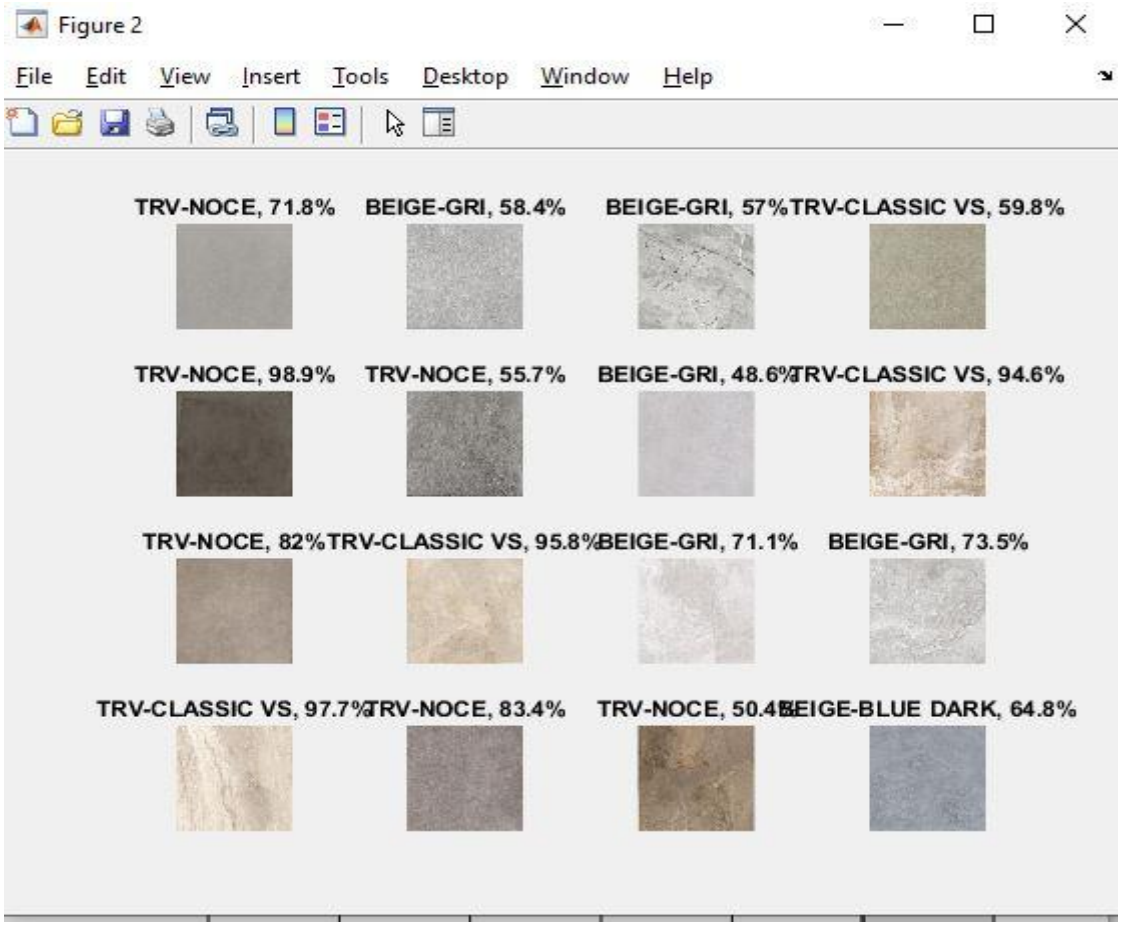

Figure.21 Resnet50 Prediction of Random Images

2.2.6) Resnet101

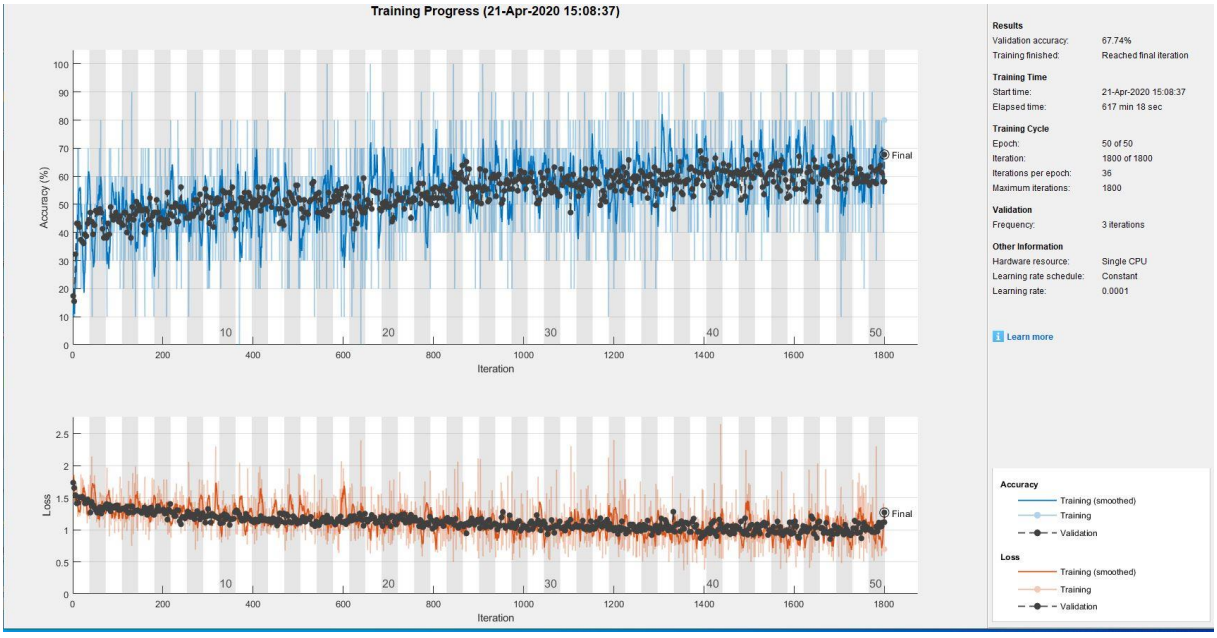

Figure.22 Resnet101 Training Result

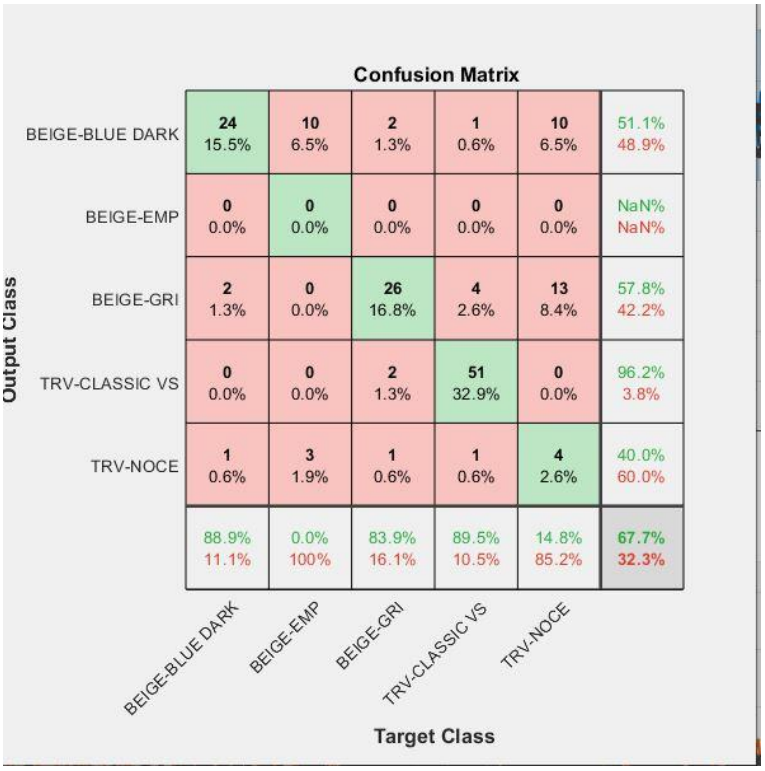

Figure.23 Resnet101 Confussion Matrix

Figure.24 Resnet101 Prediction of Random Images

2.2.7) MobileNet

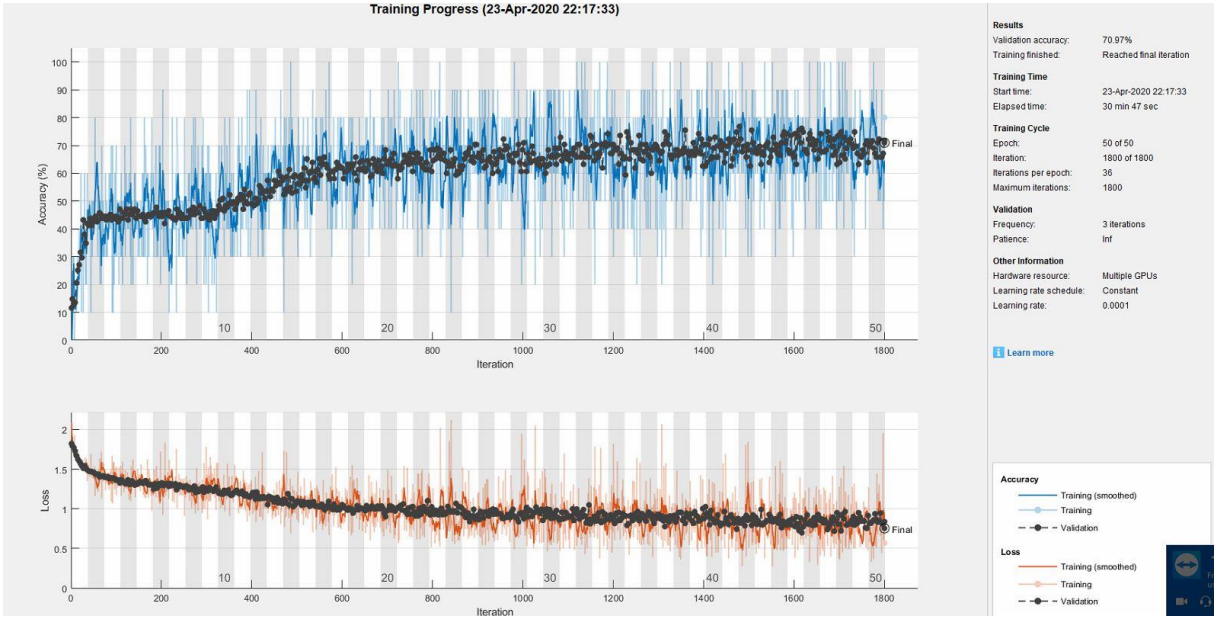

Figure.25 MobileNet Training Result

Figure.26 MobileNet Confussion Matrix

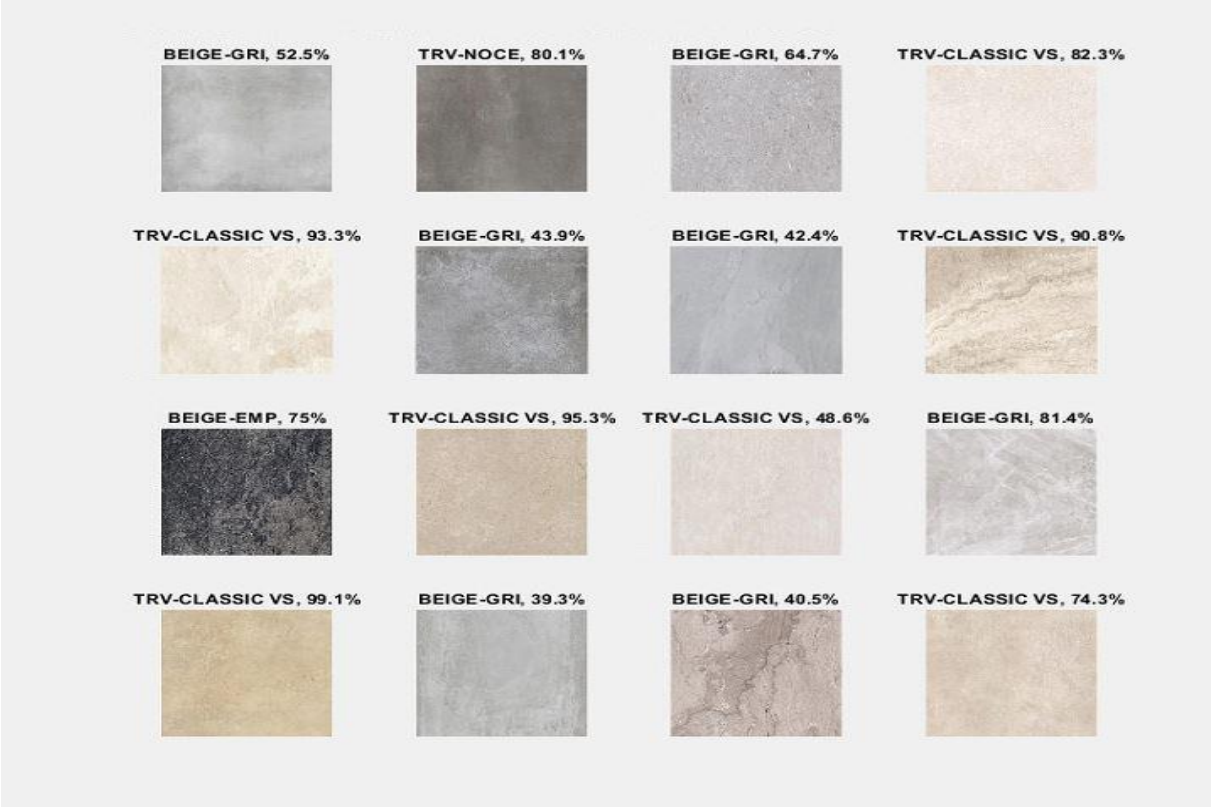

Figure.27 MobileNet Prediction of Random Images

2.2.8) Inception

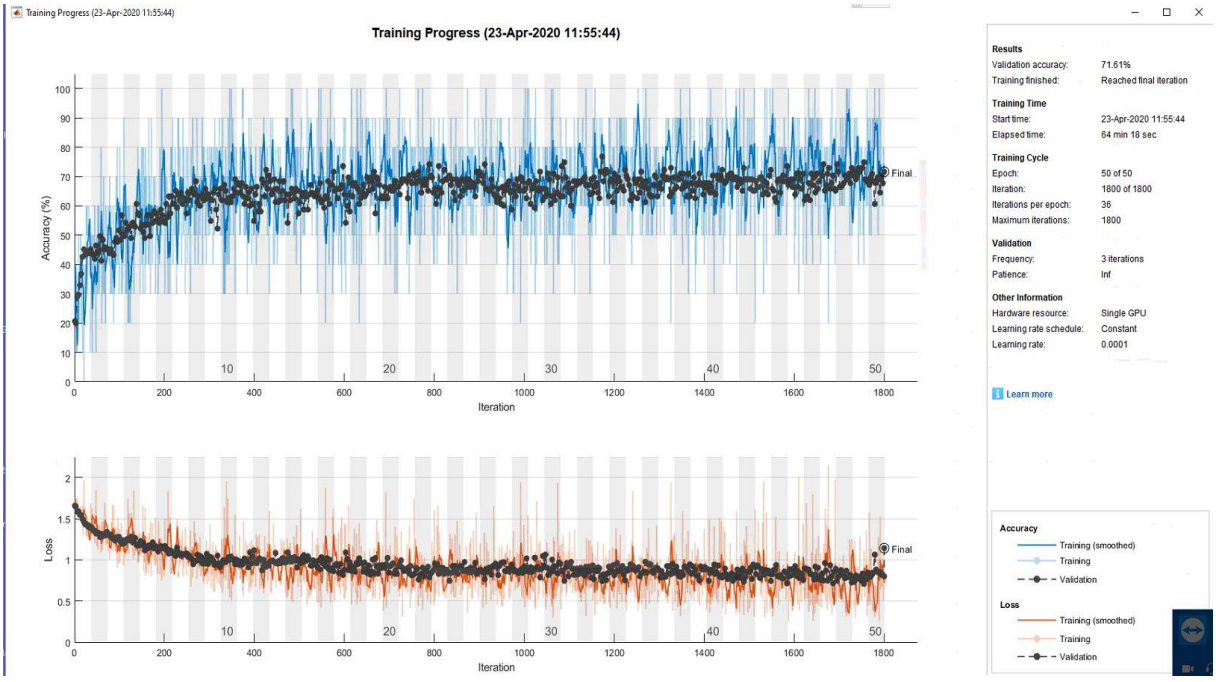

Figure.28 Inception Training Result

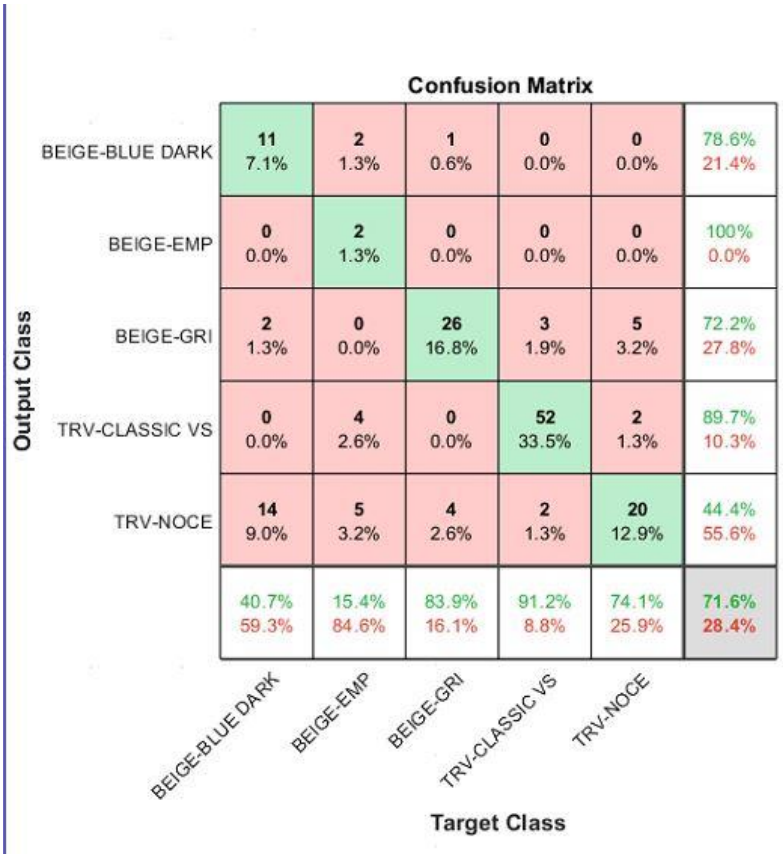

Figure.29 Inception Confussion Matrix

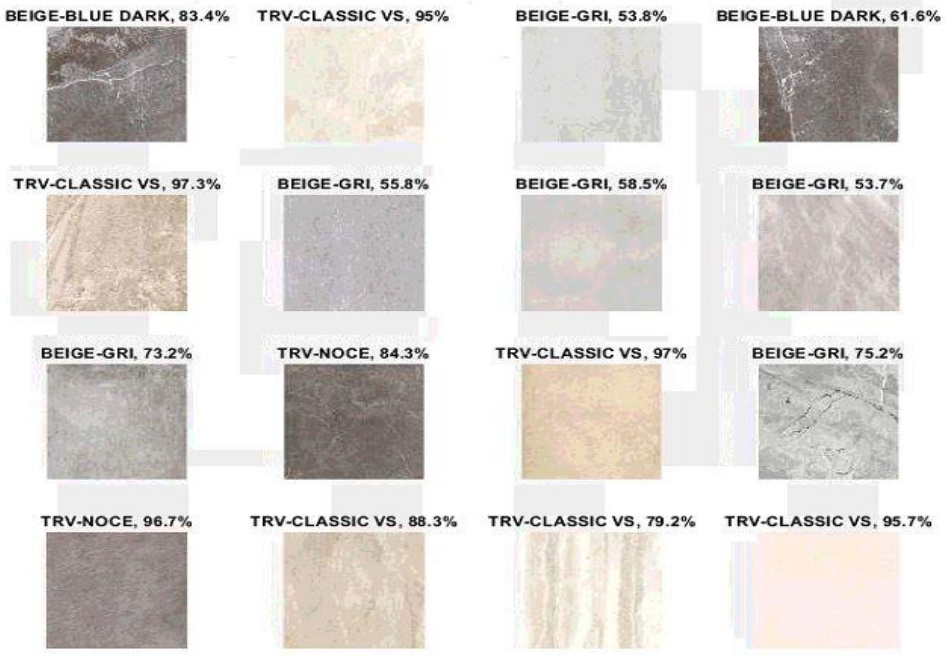

Figure.30 Inception Prediction of Random Images

2.2.9) GoggleNet

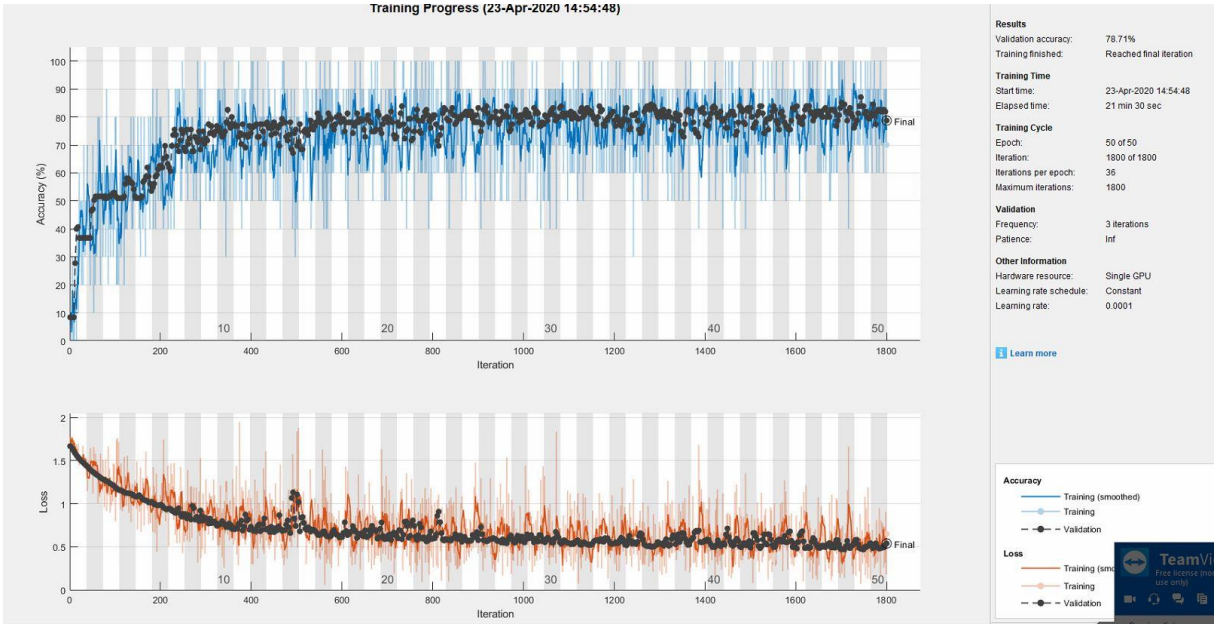

Figure.31 GoggleNet Training Result

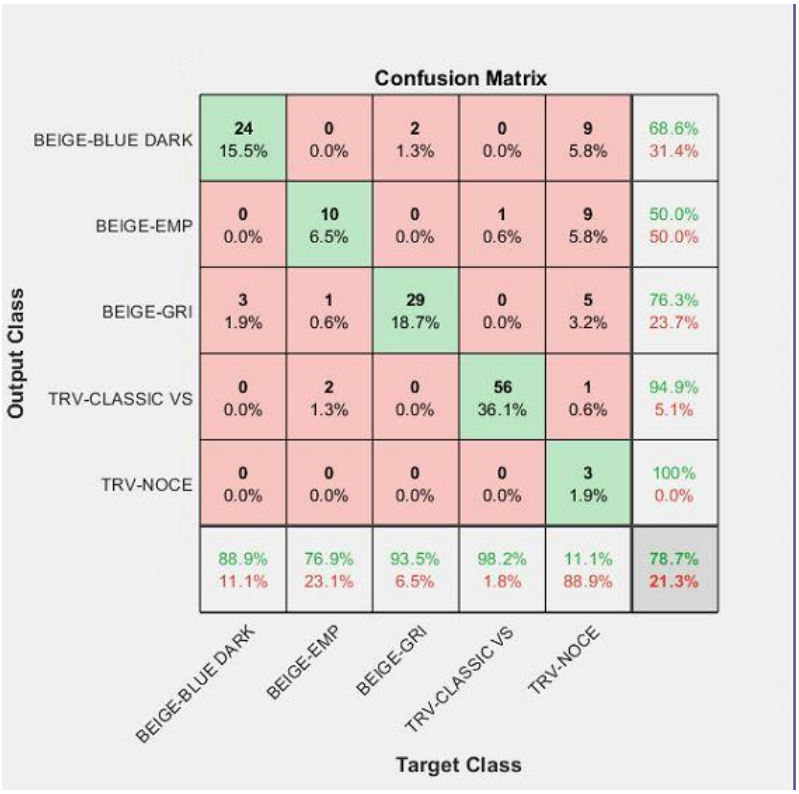

Figure.32 GoogleNet Confussion Matrix

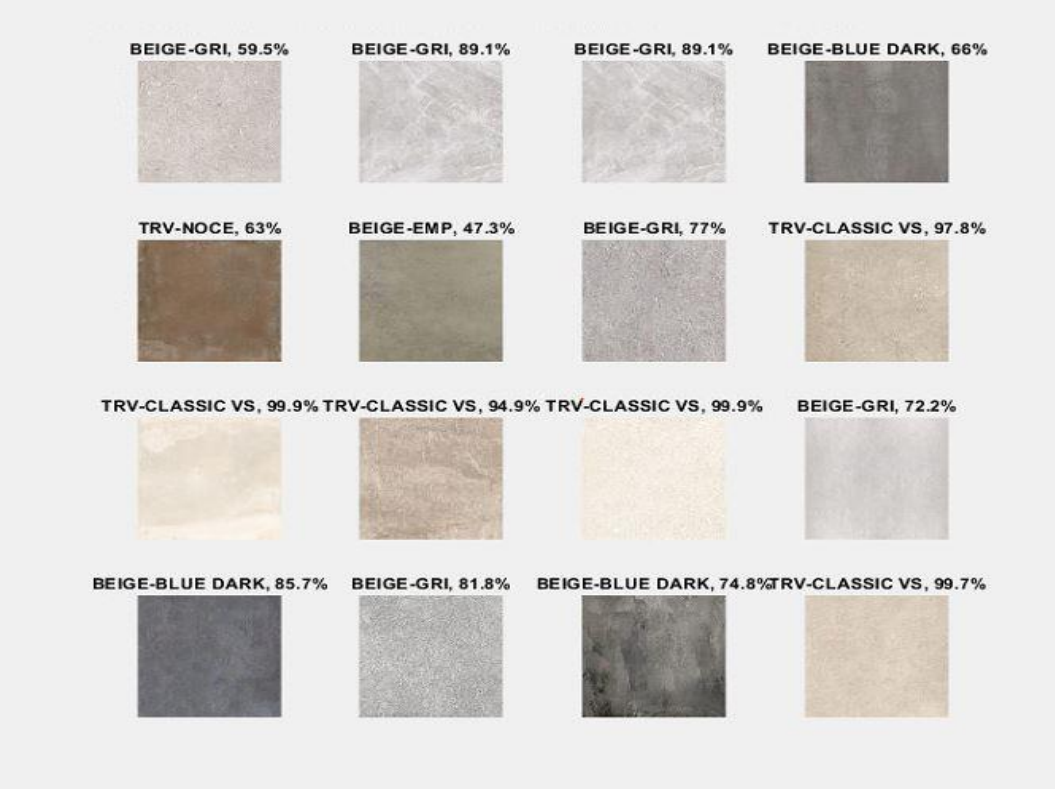

Figure.33 GoogleNet Prediction of Random Images

2.2.10) DarkNet19

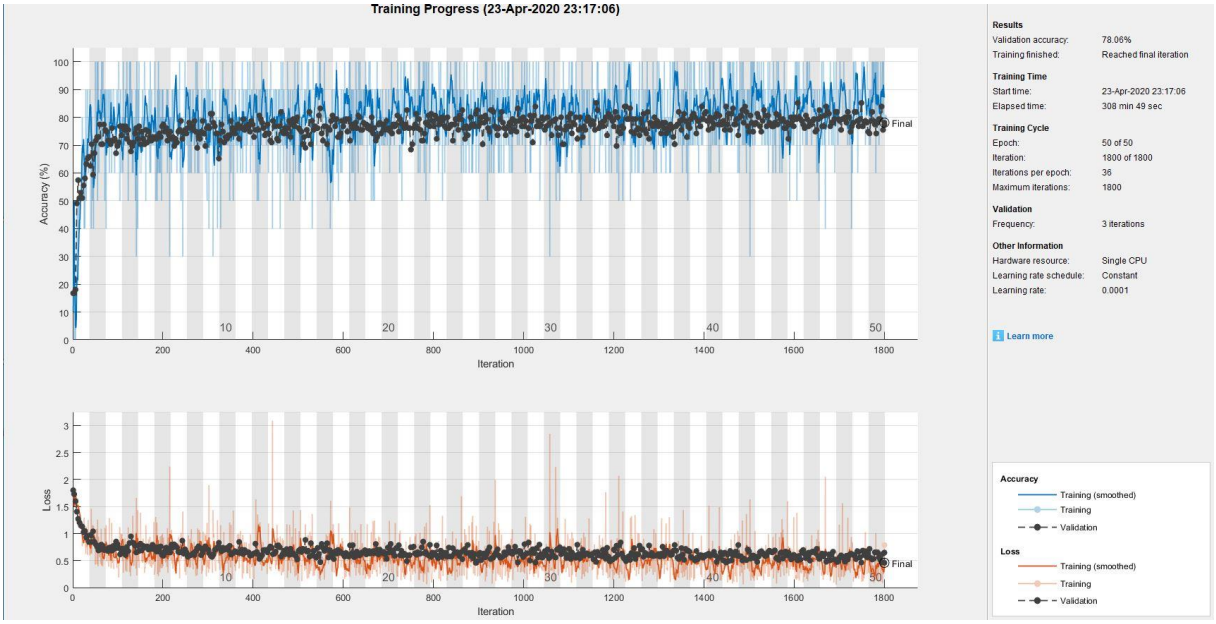

Figure.34 DarkNet19 Training Result

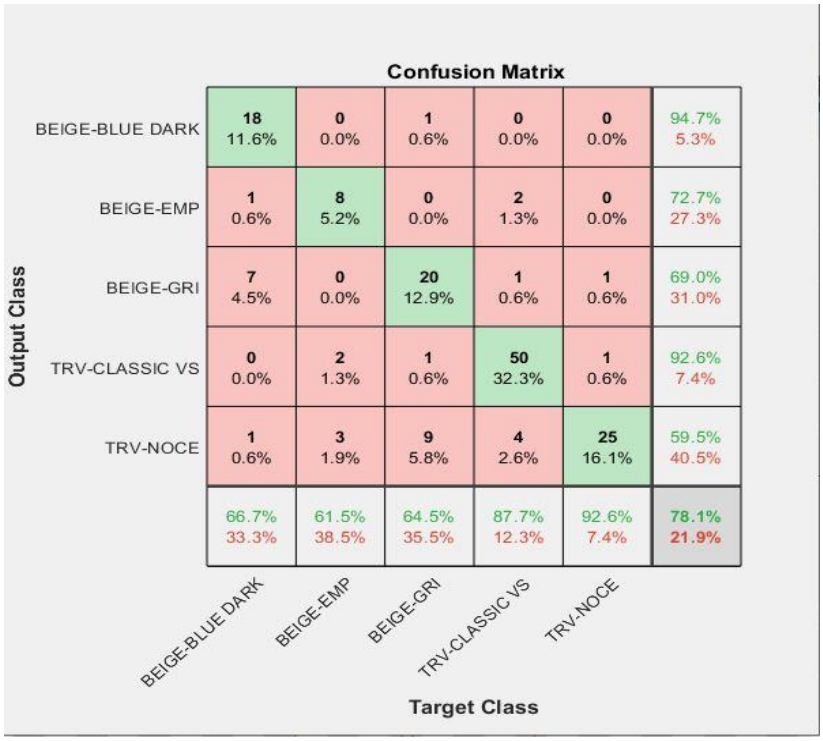

Figure.35 DarkNet19 Confussion Matrix

Figure.36 DarkNet19 Prediction of Random Images

2.2.11) DarkNet53

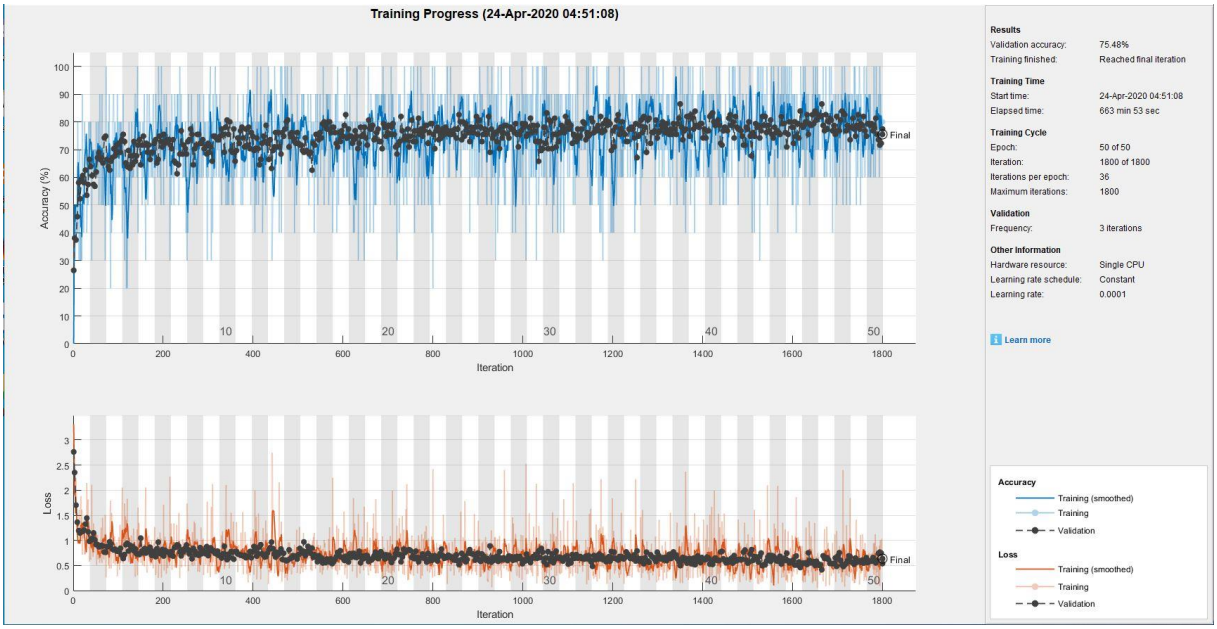

Figure.37 DarkNet53 Training Result

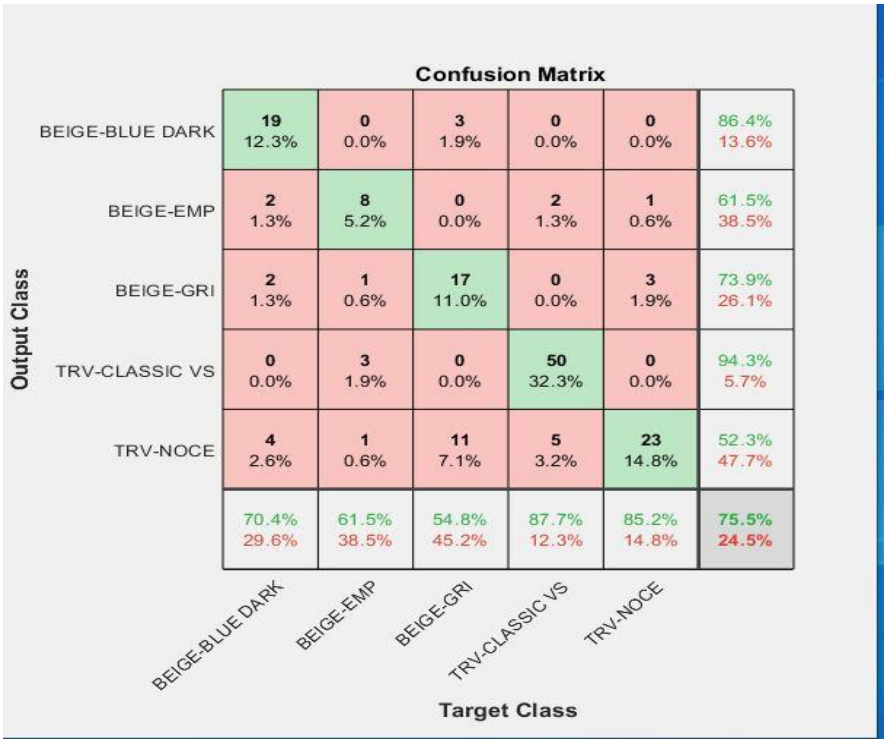

Figure.38 DarkNet53 Confussion Matrix

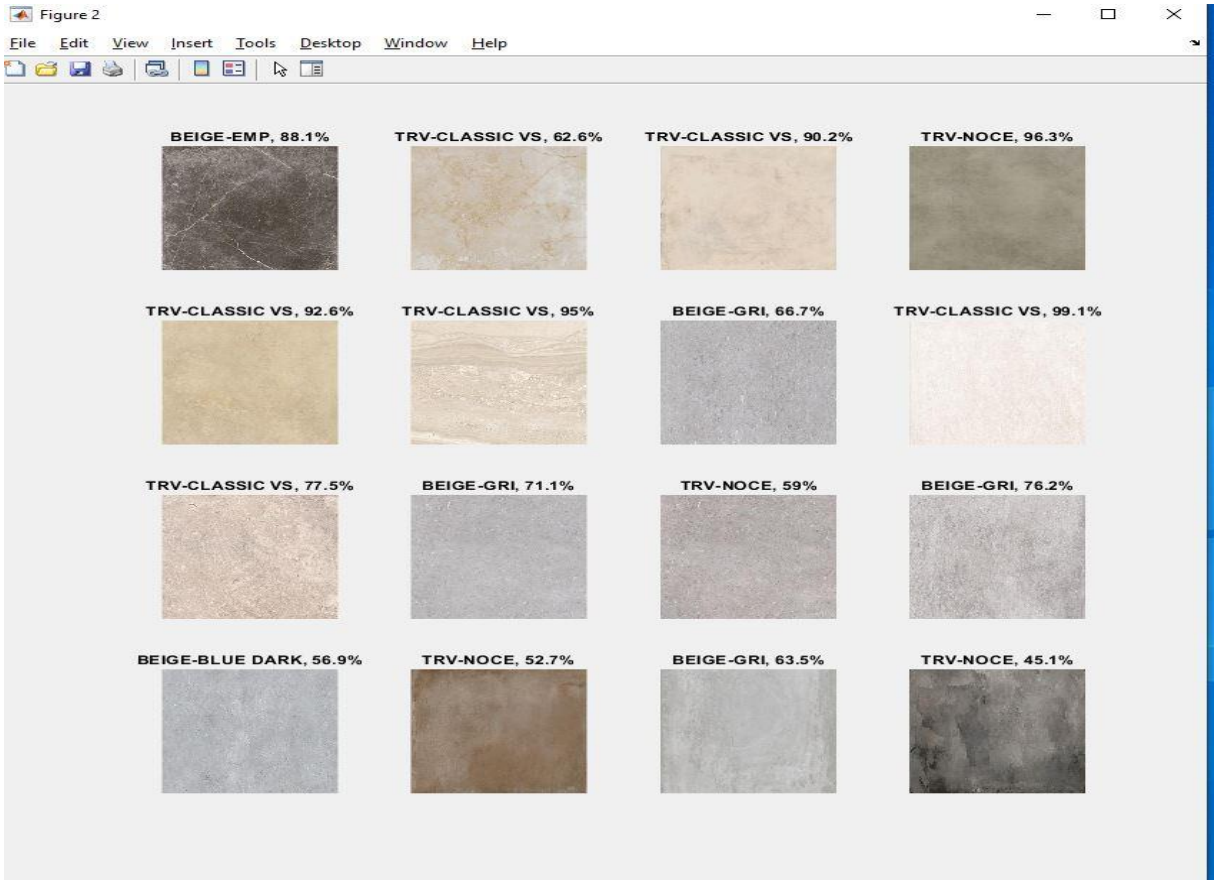

Figure.39 DarkNet53 Prediction of Random Images

2.2.12) AlexNet

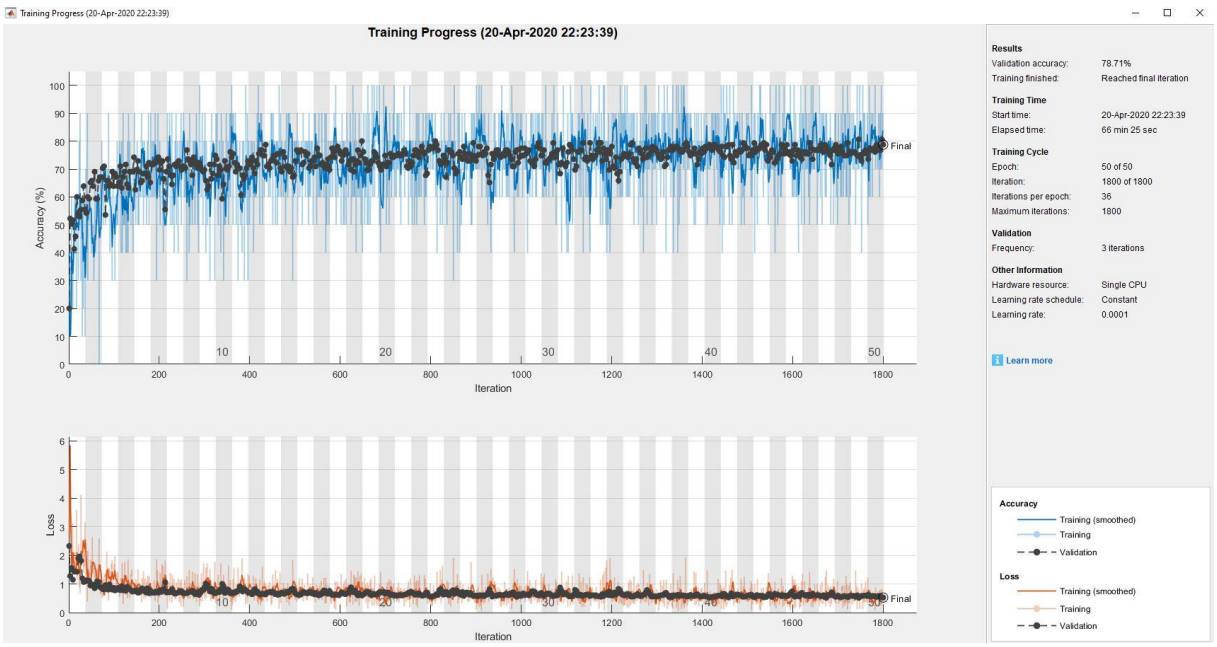

Figure.40 AlexNet Training Result

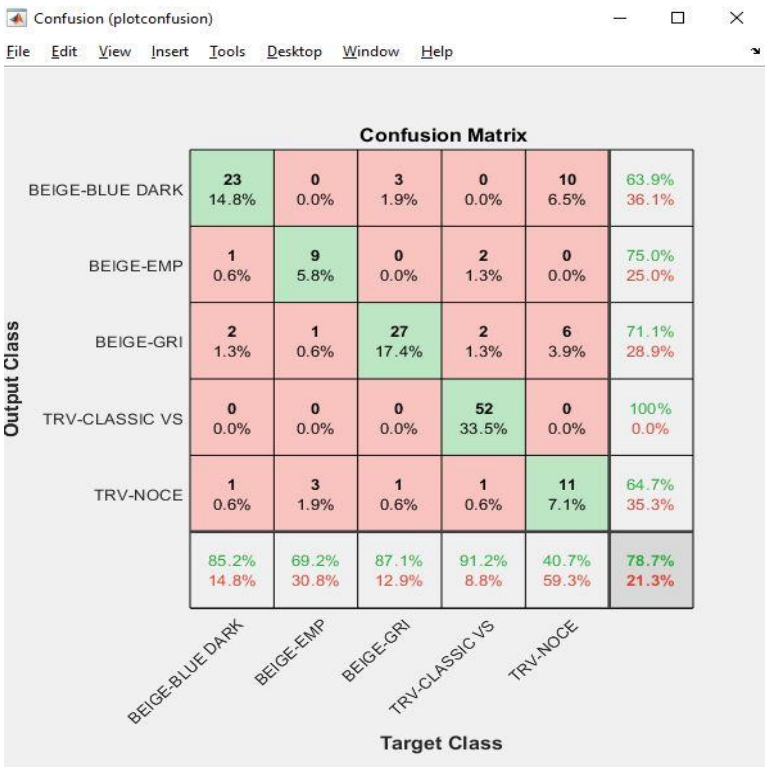

Figure.41 Alexnet Confussion Matrix

Figure.42 AlexNet Prediction of Random Images

2.2.13) Matlab Models Result

Model Name	Accuracy Result
Xception	87,74
GoogleNet	78,78
Resnet18	78,71
AlexNet	78,06
DarkNet19	78,06
Vgg16	78,05
Vgg19	77,42
Darkne53	75,48
Resnet50	72,26
Inception	71,61
MobileNet	70,97
Resnet101	67,74

Figure.43 Matlab Result Table

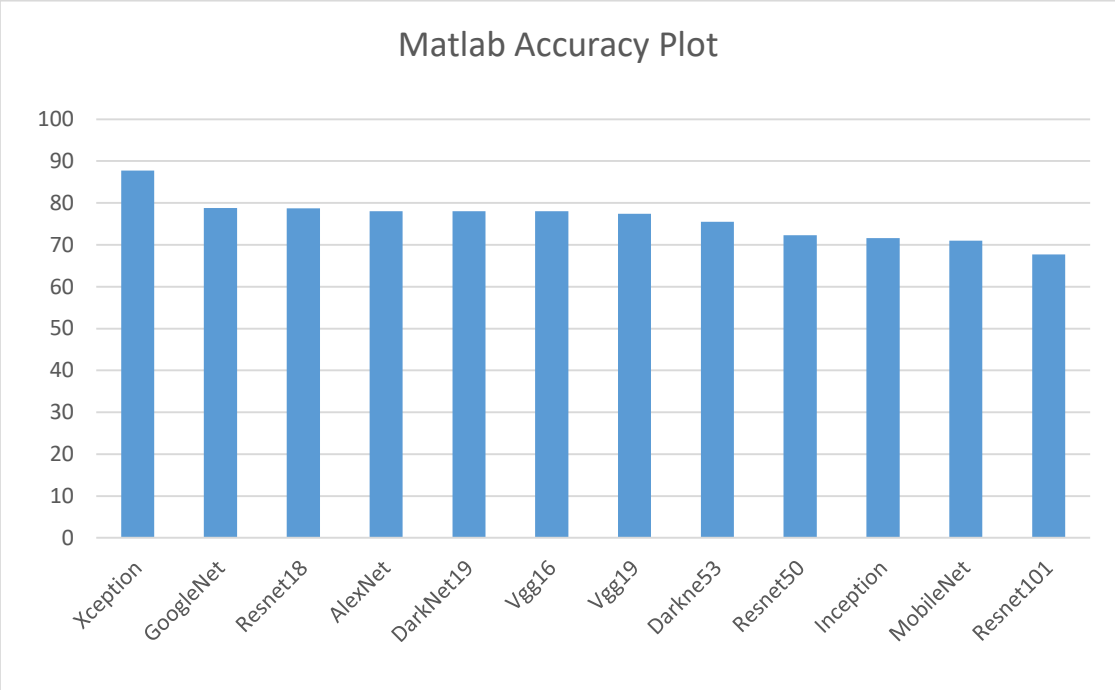

Figure.44 Matlab Result Plot

2.2.14) Matlab Model Result with Subclasses

After all 12 model trained in matlab environment with 5 classes marble data with 5 models trained with same images but with 2 subclass as Trv and Beige. And these t class diveded 2 and 3 into the diffrerent class respectively.

Model Name	5 Class Training Accuracy (%)	SubClass Training Accuracy (%)
Xception	87.74	87.74
GoogleNet	78.78	79.35
Vgg16	78.05	76.77
Vgg19	77.42	74.14
AlexNet	78.06	71.64

Figure.45 Matlab Accuracy Comparison Plot

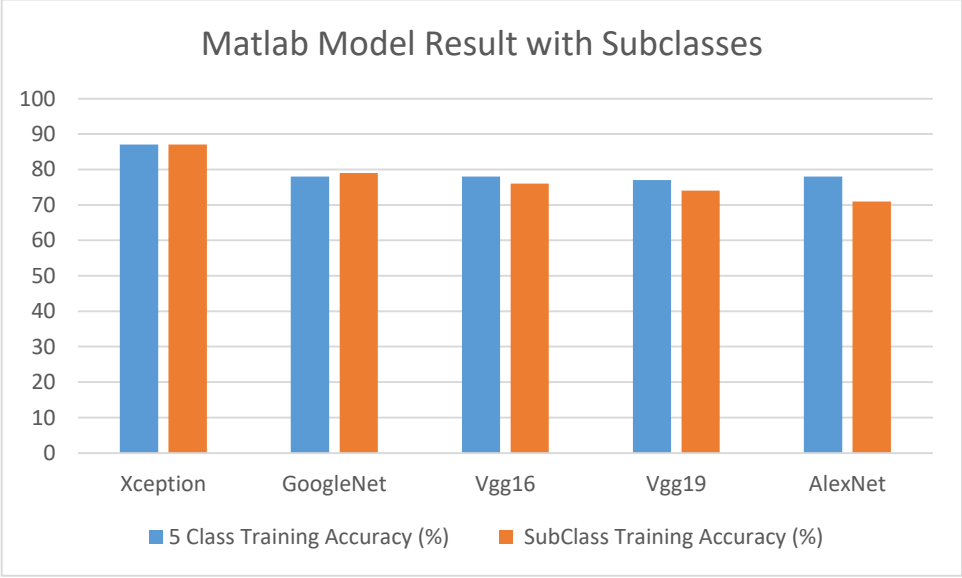

Figure.46 Matlab Accuracy Comparison Graph

2.3) Pyhton

After all 12 models trained in Matlab part result was noted. And some models which gives over the overall accucacy levels coded on Anaconda Jupyter Notebook which are Vgg16, Vgg19, Xception, Inception, MobileNet and Resnet50. In this pyhton part as same Matlab as started with imported libraries and loaded data.

```
data_dir = r'D:\_MASAÜSTÜ\2020 _BAHAR\Deep learning\DATA_299x299'

image_generator = ImageDataGenerator(rescale=1/255, validation_split=0.2)


train_data_dir = image_generator.flow_from_directory(batch_size=32,
                                                     directory=data_dir,
                                                     shuffle=True,
                                                     target_size=(224, 224),
                                                     subset="training",
                                                     class_mode='categorical')

validation_dir = image_generator.flow_from_directory(batch_size=32,
                                                     directory=data_dir,
                                                     shuffle=True,
                                                     target_size=(224, 224),
                                                     subset="validation",
                                                     class_mode='categorical')
```

Figure.47 A part of code from pyhton (loaded data)

After loaded 516 marble images to the system these are split with 0.2 ratio. So 414 marble images used for train and other 102 marble images used for test.

```
model = tf.keras.Sequential(
    [
        tf.keras.layers.Conv2D(32,(3,3),padding='same',kernel_initializer='he_normal',input_shape=(224,
        tf.keras.layers.Activation('elu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(32,(3,3),padding='same',kernel_initializer='he_normal',input_shape=(224,
        tf.keras.layers.Activation('elu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Conv2D(64, (3, 3), activation='elu', padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(64, (3, 3), activation='elu', padding='same',kernel_initializer='he_norm
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Conv2D(128, (3, 3), activation='elu', padding='same', kernel_initializer='he_n
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(128, (3, 3), activation='elu', padding='same',  kernel_initializer='he_
        tf.keras.layers.BatchNormalization(),
```

Figure.48 Model part of code with ‘elu’ activation function

```
earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                           min_delta=0,
                           patience=3,
                           verbose=1,
                           restore_best_weights=True
                           )
```

Figure.49 Early Stop Code

Early stopping is a technique used to terminate the training before overfitting occurs.

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=3,
                              verbose=1,
                              min_delta=0.0001)
```

Figure.50 Reduce Learning Rate Code

Reduce learning rate when a metric has stopped improving. Then coded for the Show train and test result as a accuracy.

```
import matplotlib.pyplot as plt
print(history.history.keys())
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Figure.51 Accuracy Plot

2.3.1) Inception (50 epochs, 32 batch size)

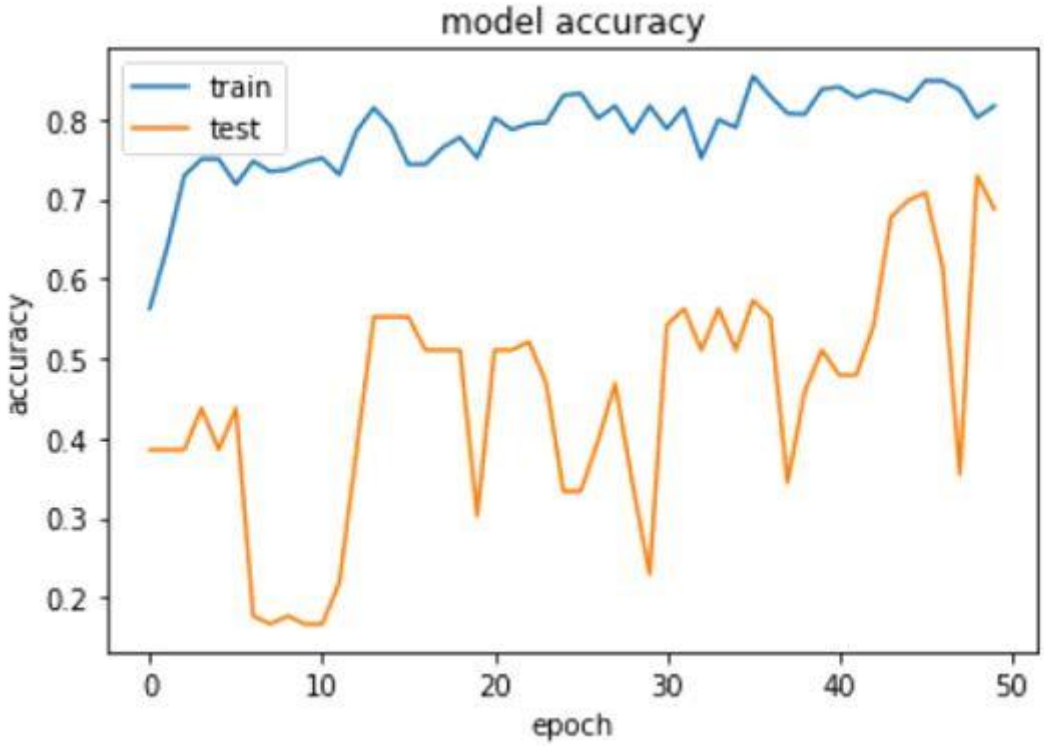

Figure.52 Inception Accuracy Plot

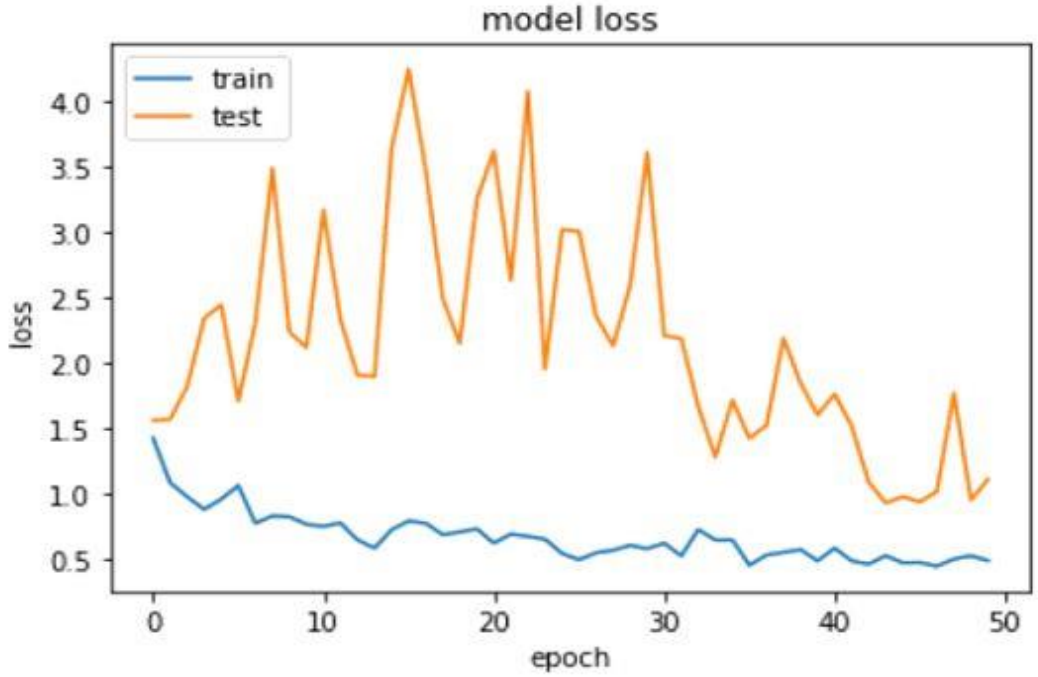

Figure.53 Inception Accuracy Loss Plot

2.3.2) Resnet50 (50 epochs, 32 batch size)

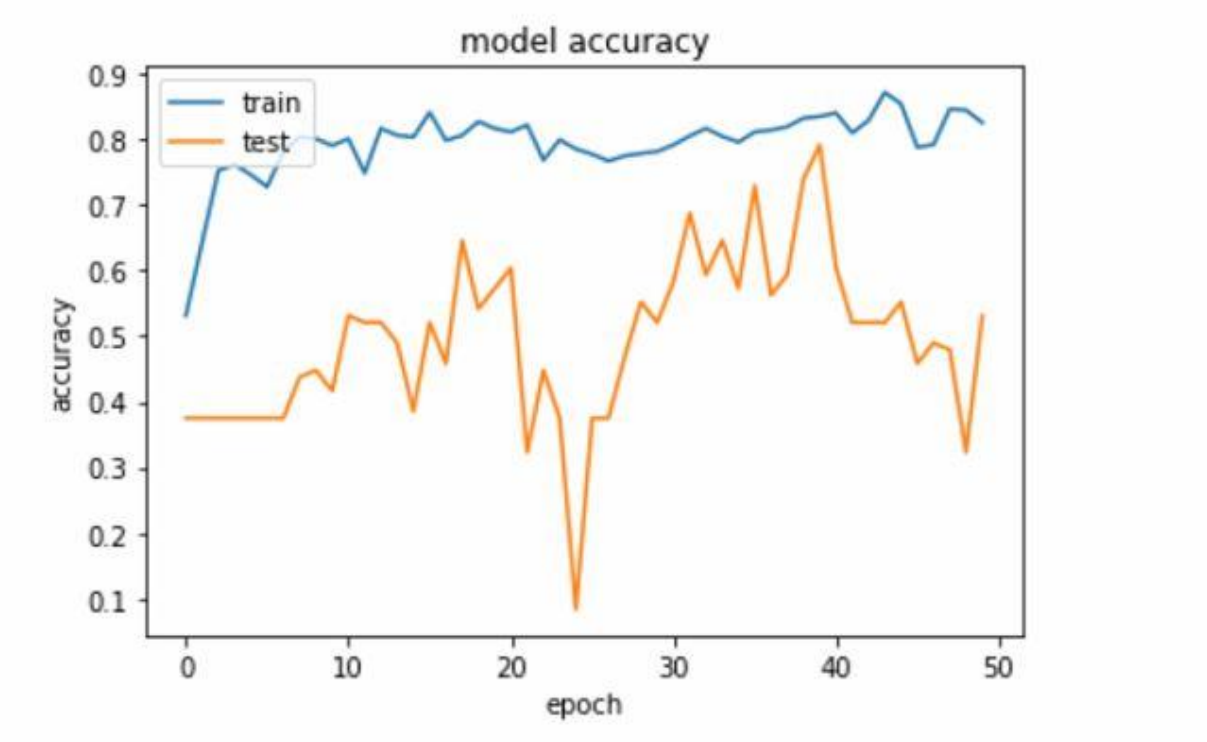

Figure.54 Resnet50 Accuracy Plot

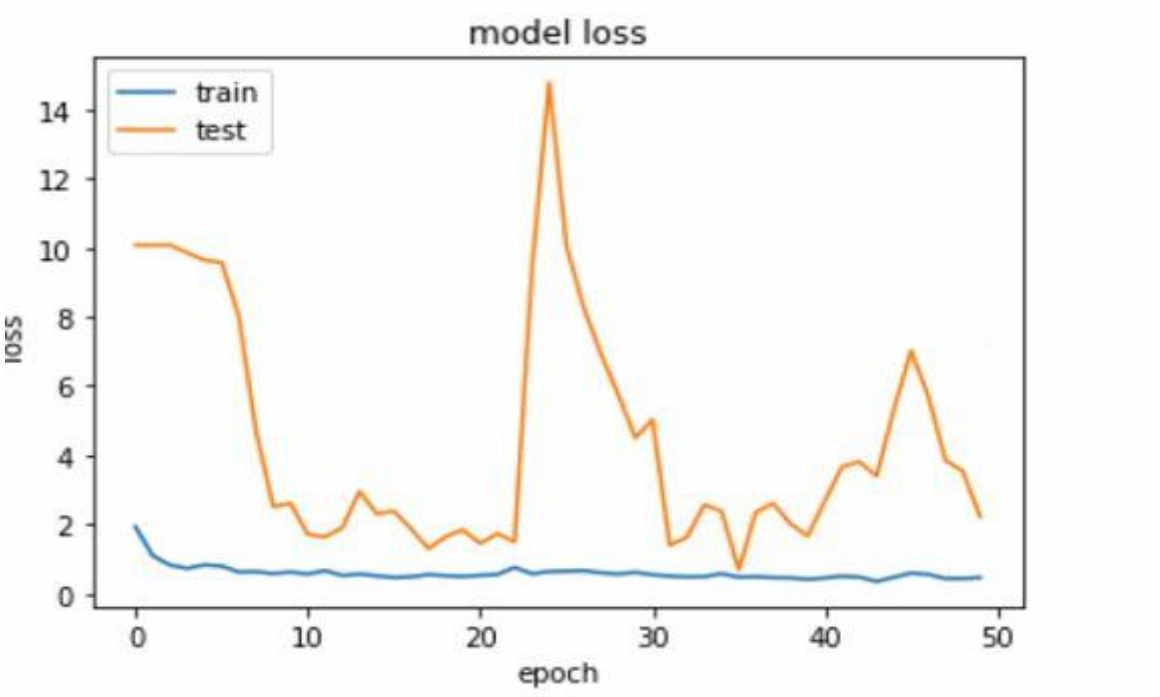

Figure.55 Resnet50 Accuracy Loss Plot

2.3.3) Resnet50 (100 epochs, 32 batch size)

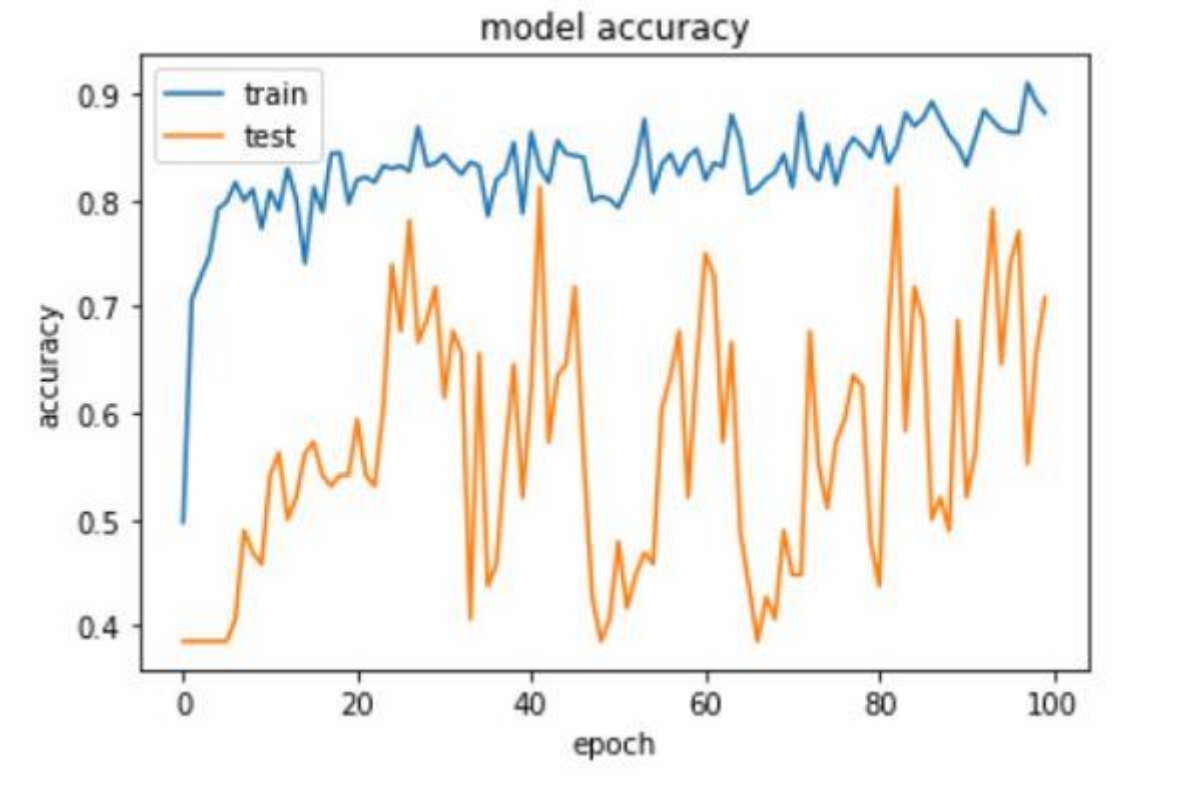

Figure.56 Resnet50_2 Accuracy Plot

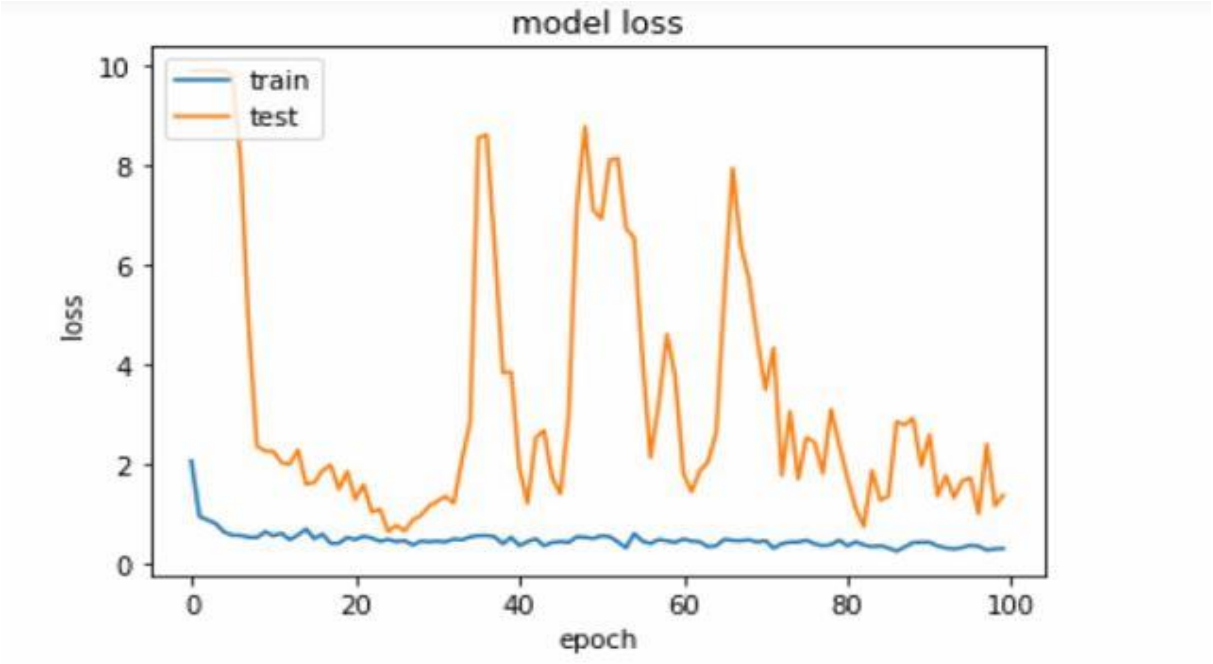

Figure.57 Resnet50_2 Accuracy Loss Plot

2.3.4) Vgg16 (50 epochs, 32 batch size, elu activation function)

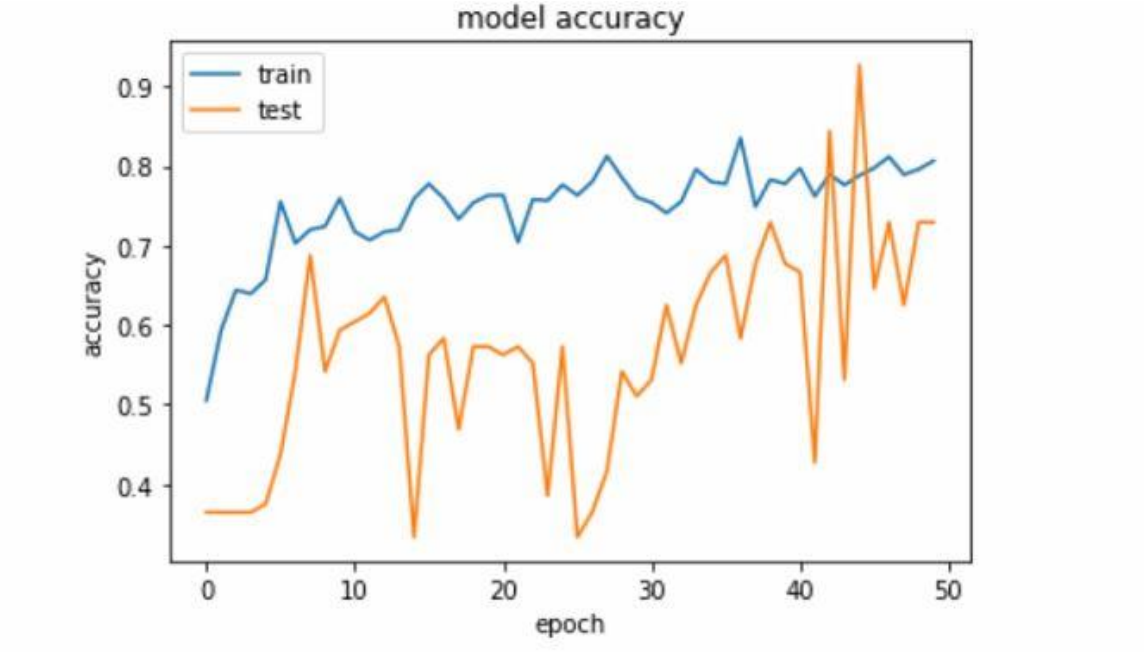

Figure.58 Vgg16_1 Accuracy Plot

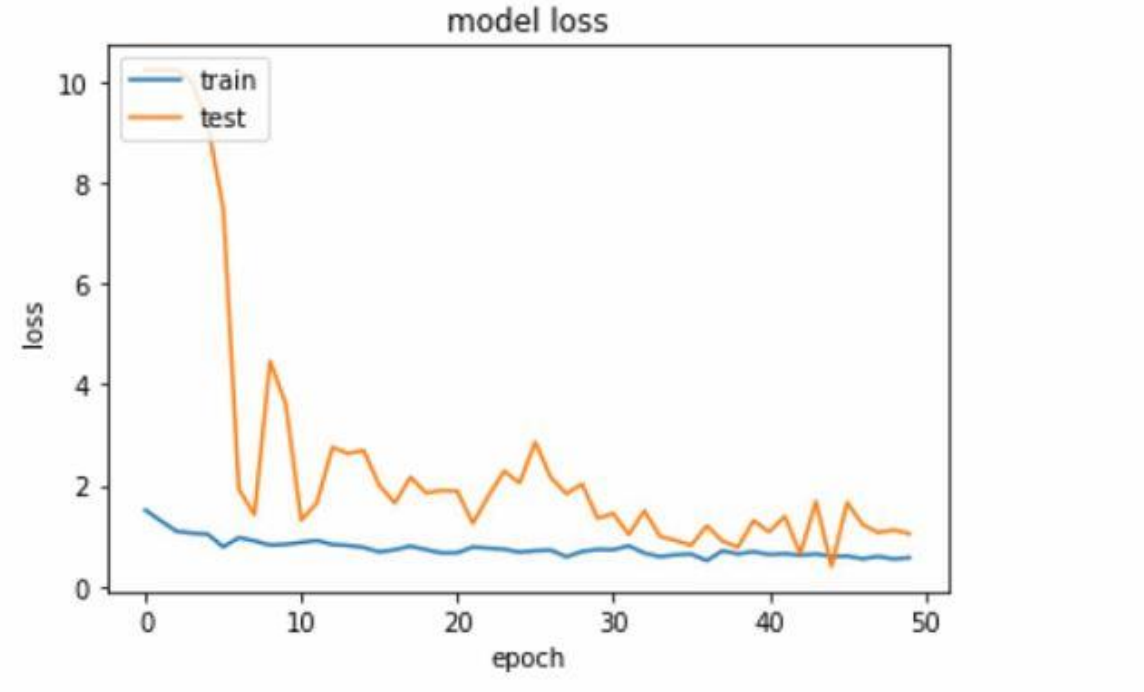

Figure.59 Vgg16_1 Accuracy Loss Plot

2.3.5) Vgg16_2 (50 epochs, 32 batch size, relu activation function)

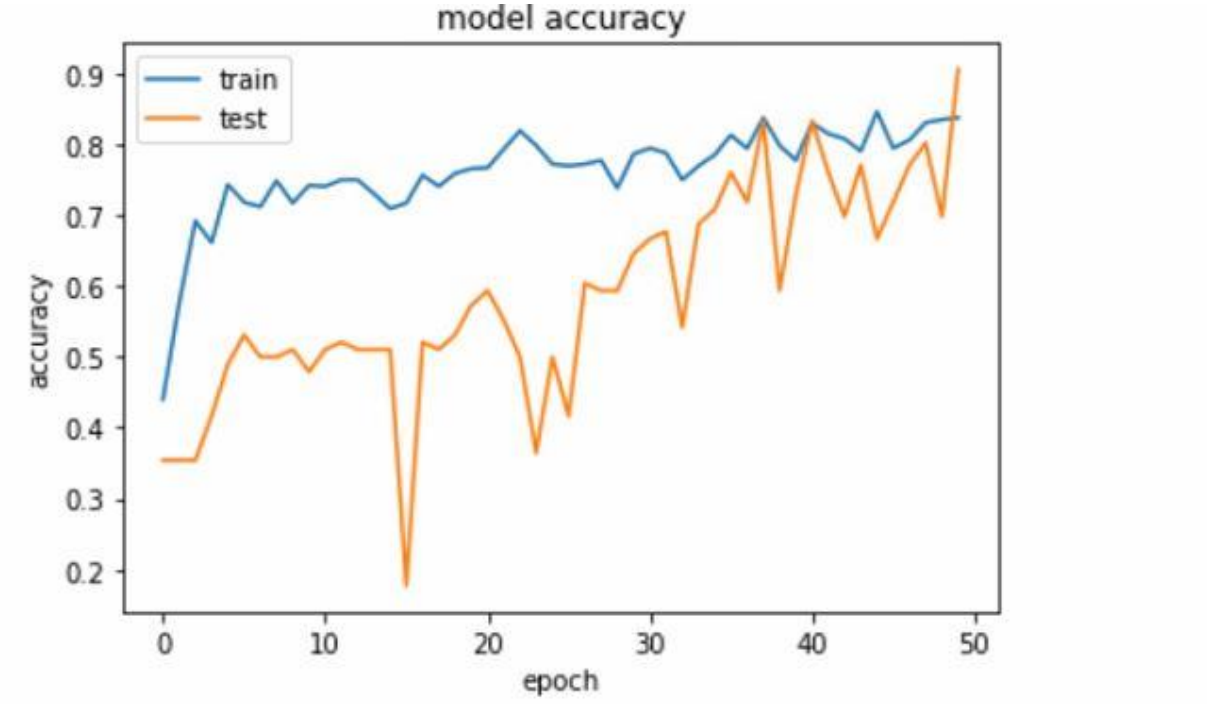

Figure.60 Vgg16_2 Accuracy Plot

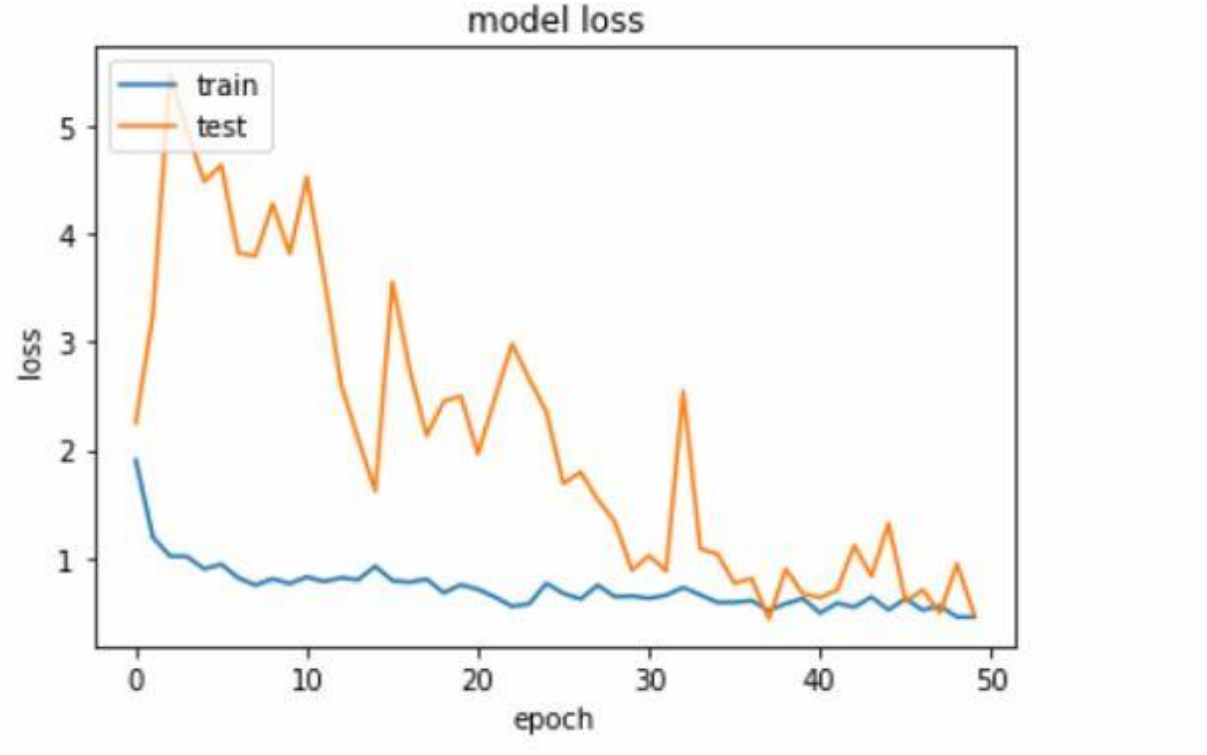

Figure.61 Vgg16_2 Accuracy Loss Plot

2.3.6) Vgg16_3 (100 epochs, 32 batch size, relu activation function)

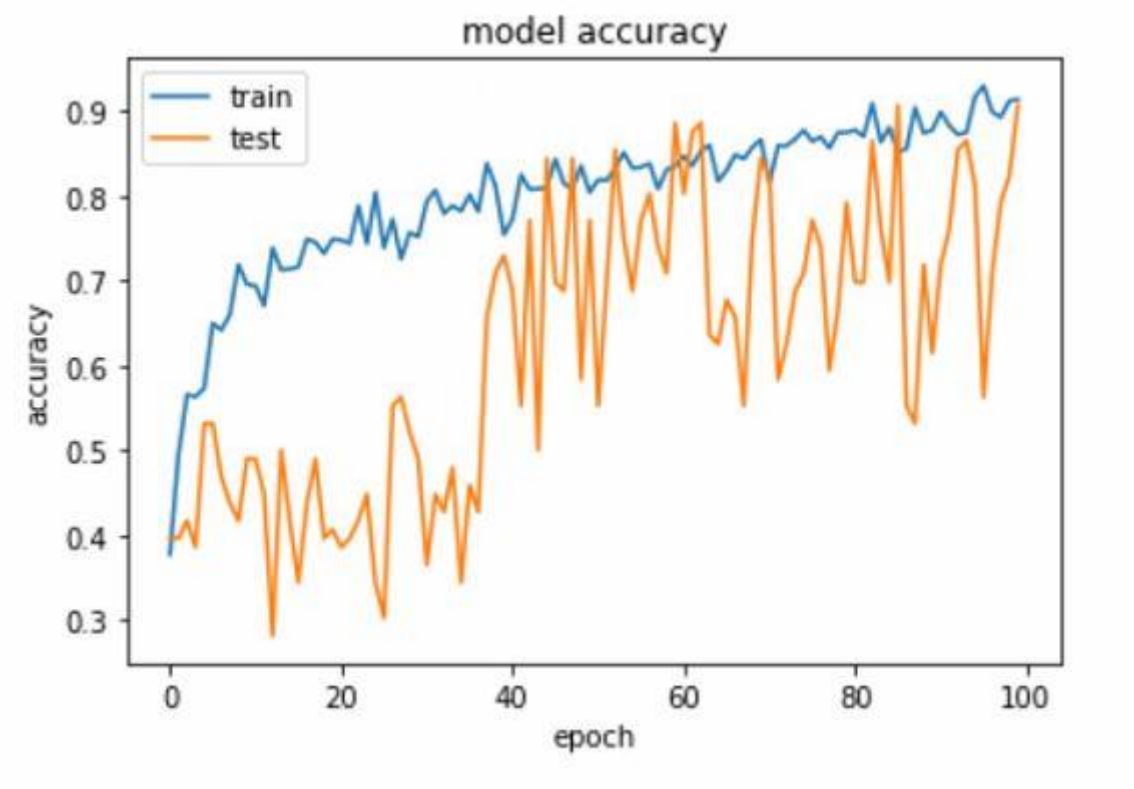

Figure.62 Vgg16_3 Accuracy Plot

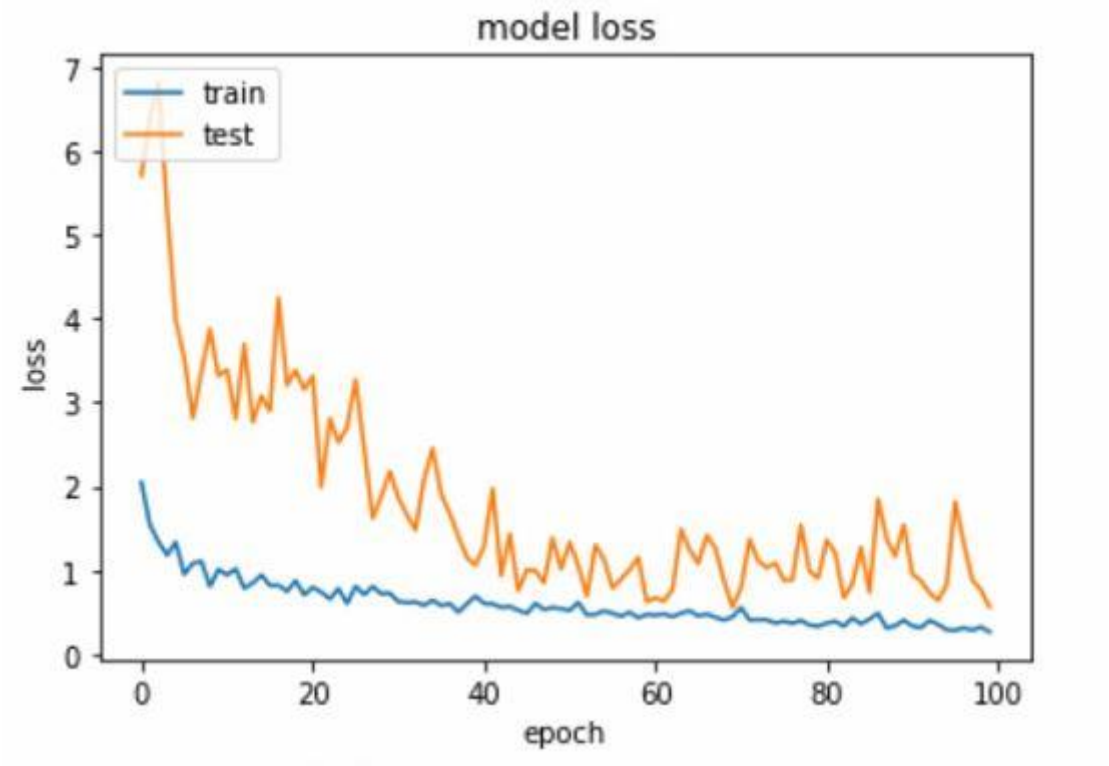

Figure.63 Vgg16_3 Accuracy Loss Plot

2.3.7) Vgg19

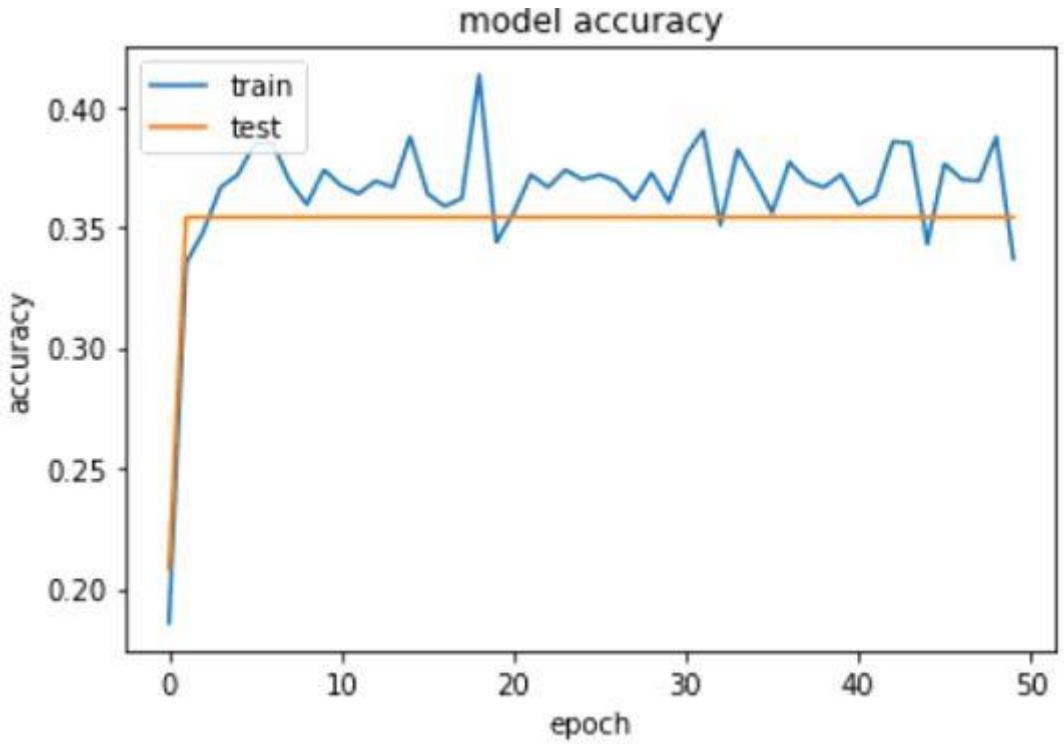

Figure.64 Vgg19 Accuracy Plot

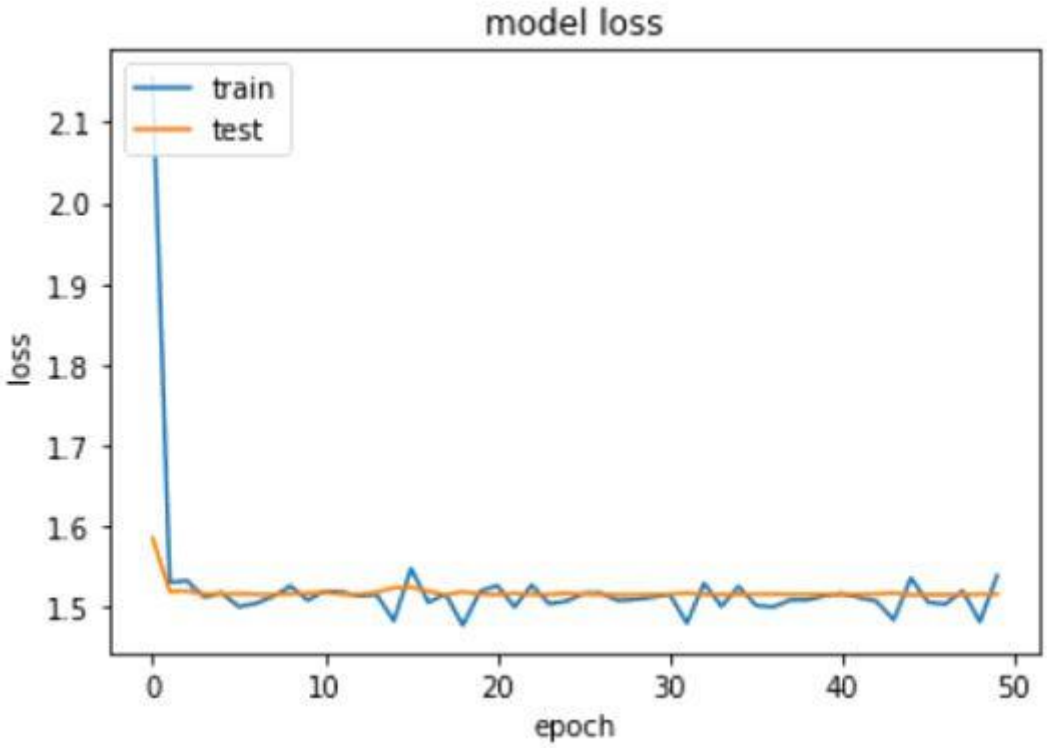

Figure.65 Vgg19 Accuracy Loss Plot

2.3.8) Xception (50 Epochs)

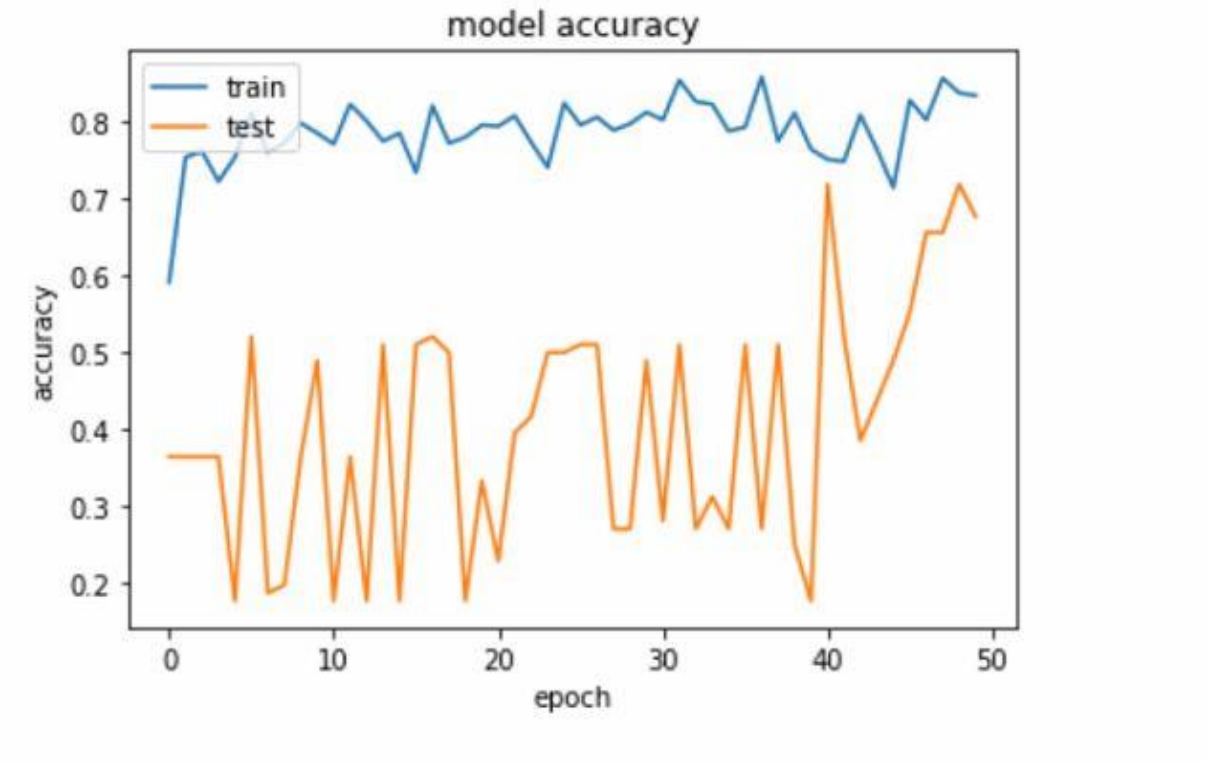

Figure.66 Xception_1 Accuracy Plot

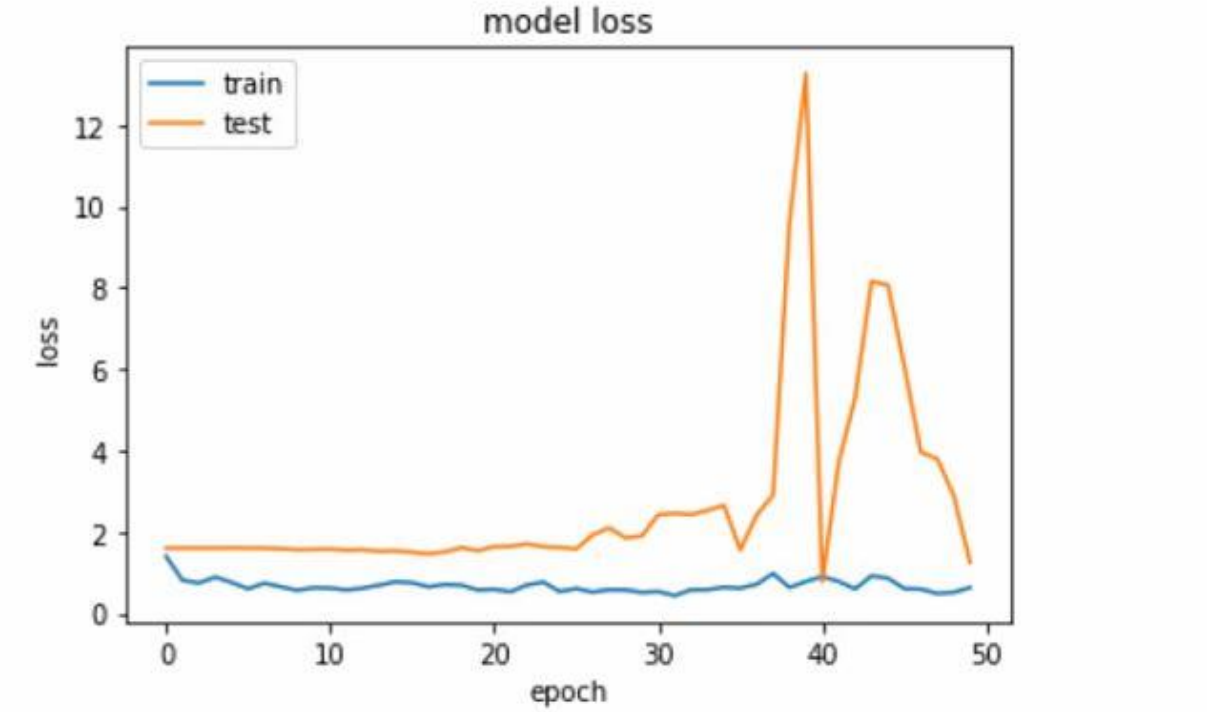

Figure.67 Xception_1 Accuracy Loss Plot

2.3.9) Xception_2 (50 Epochs, EarlyStop Disabled, 64 batch size)

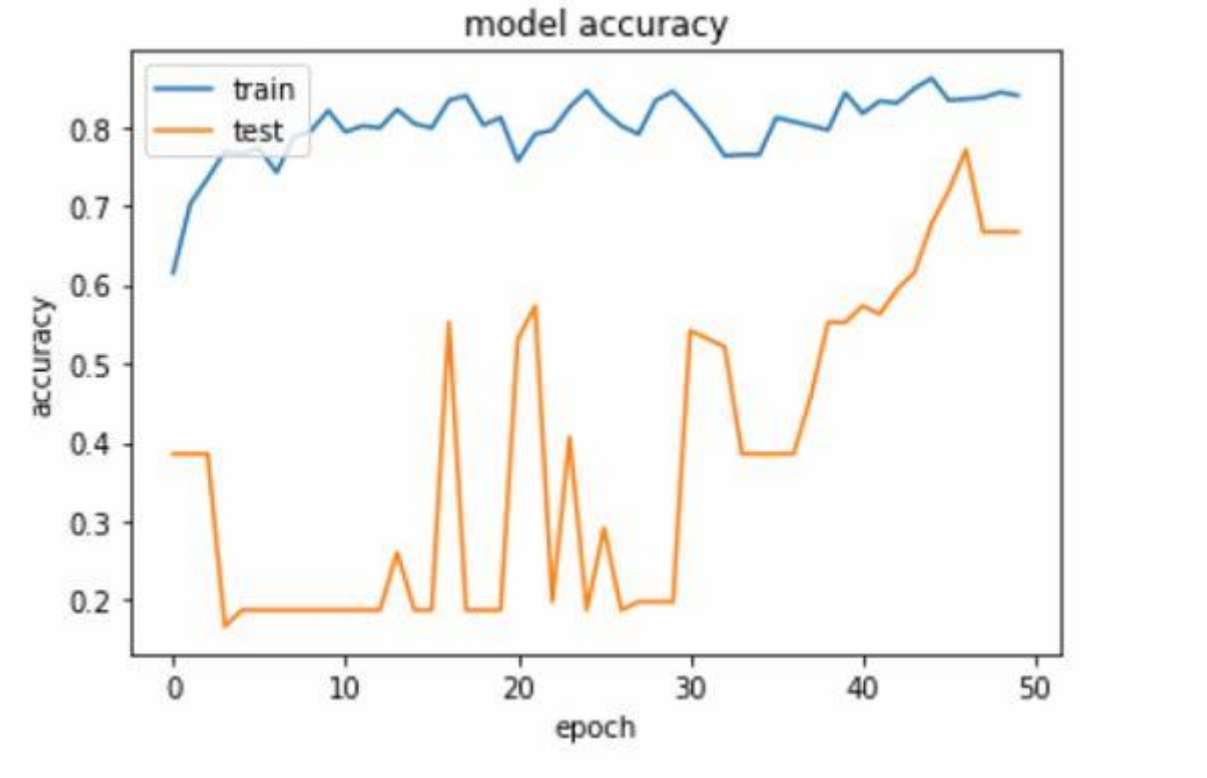

Figure.68 Xception_2 Accuracy Plot

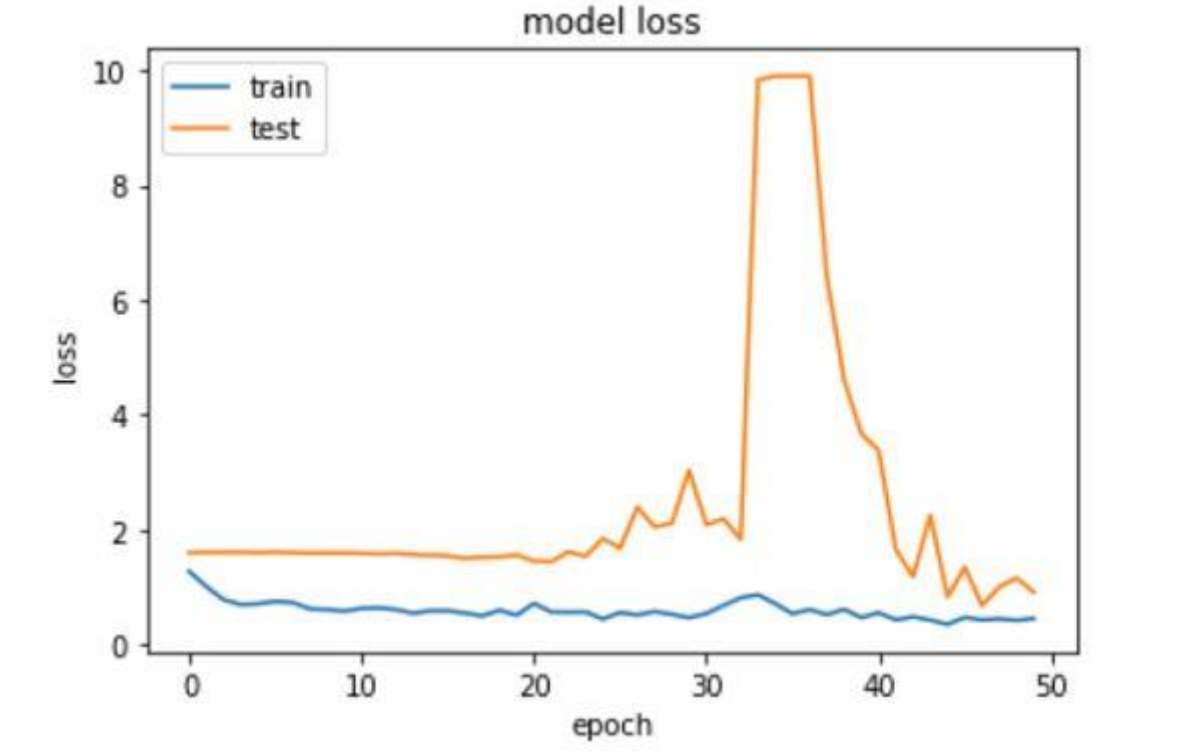

Figure.69 Xception_2 Accuracy Loss Plot

2.3.10) Xception_3 (EarlyStopped Disabled, 50 Epochs, 32 Batch Size)

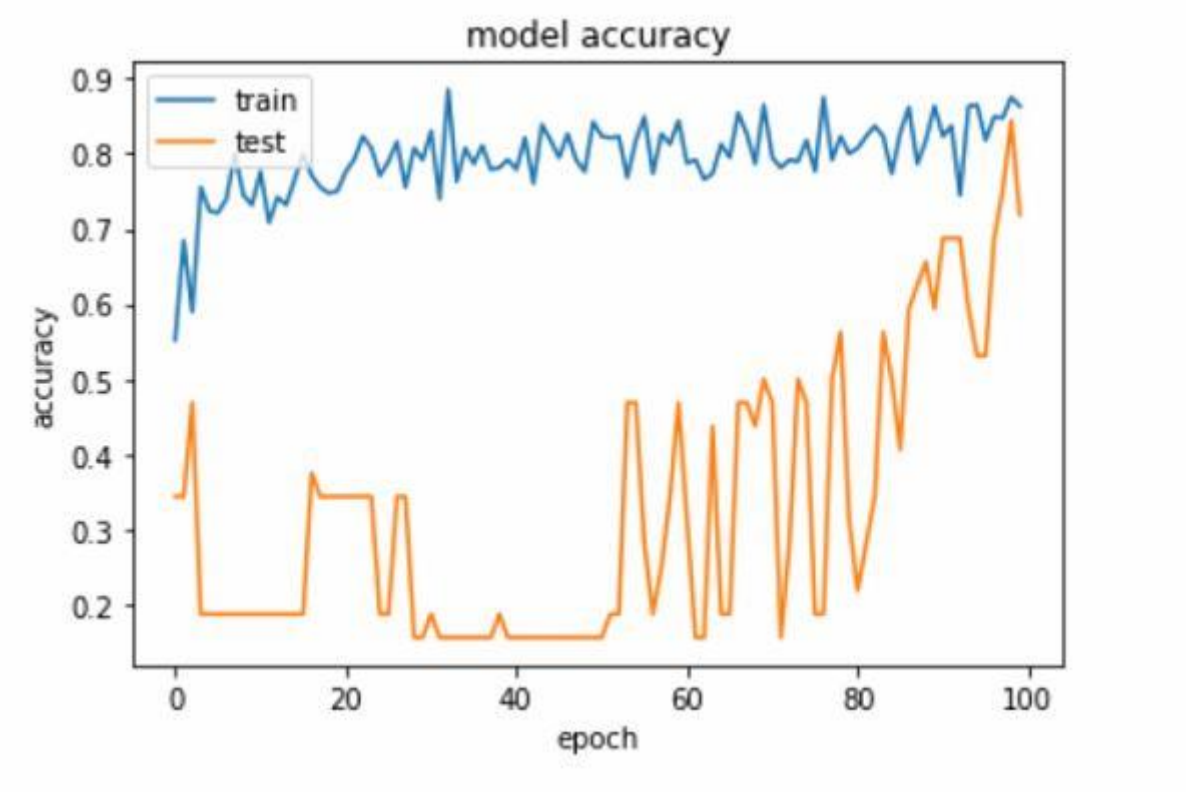

Figure.70 Xception_3 Accuracy Plot

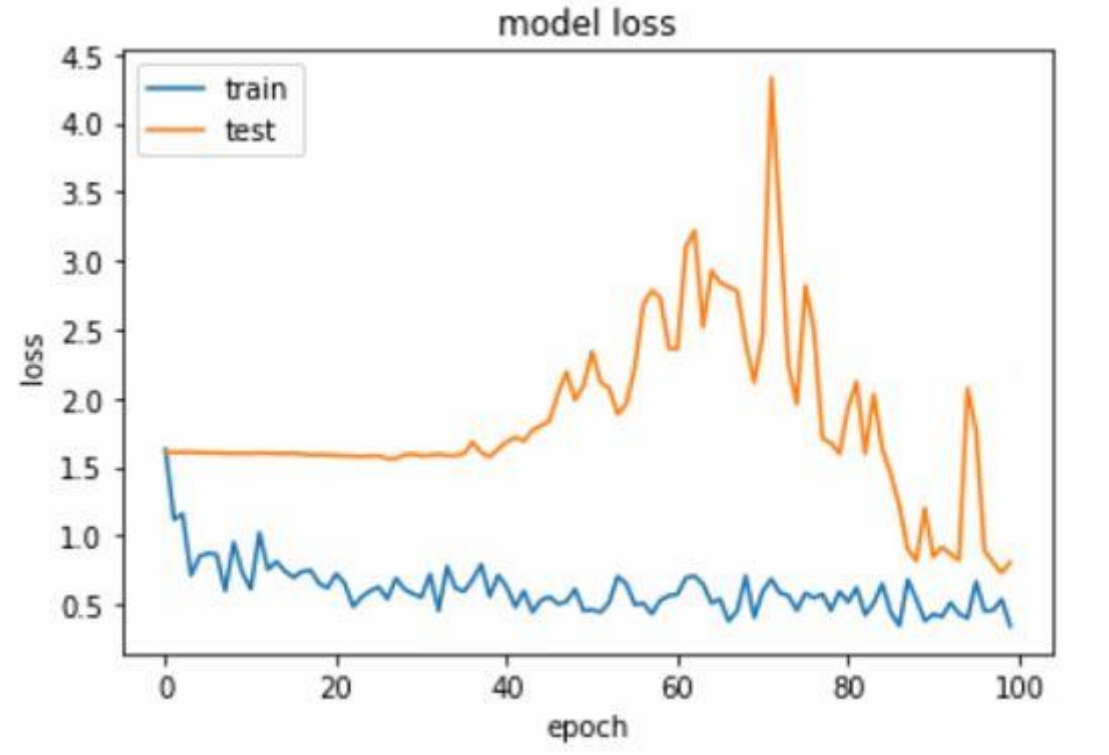

Figure.71 Xception_3 Accuracy Loss Plot

2.3.11) Xception_4 (100 Epoch, EarlyStop Disabled)

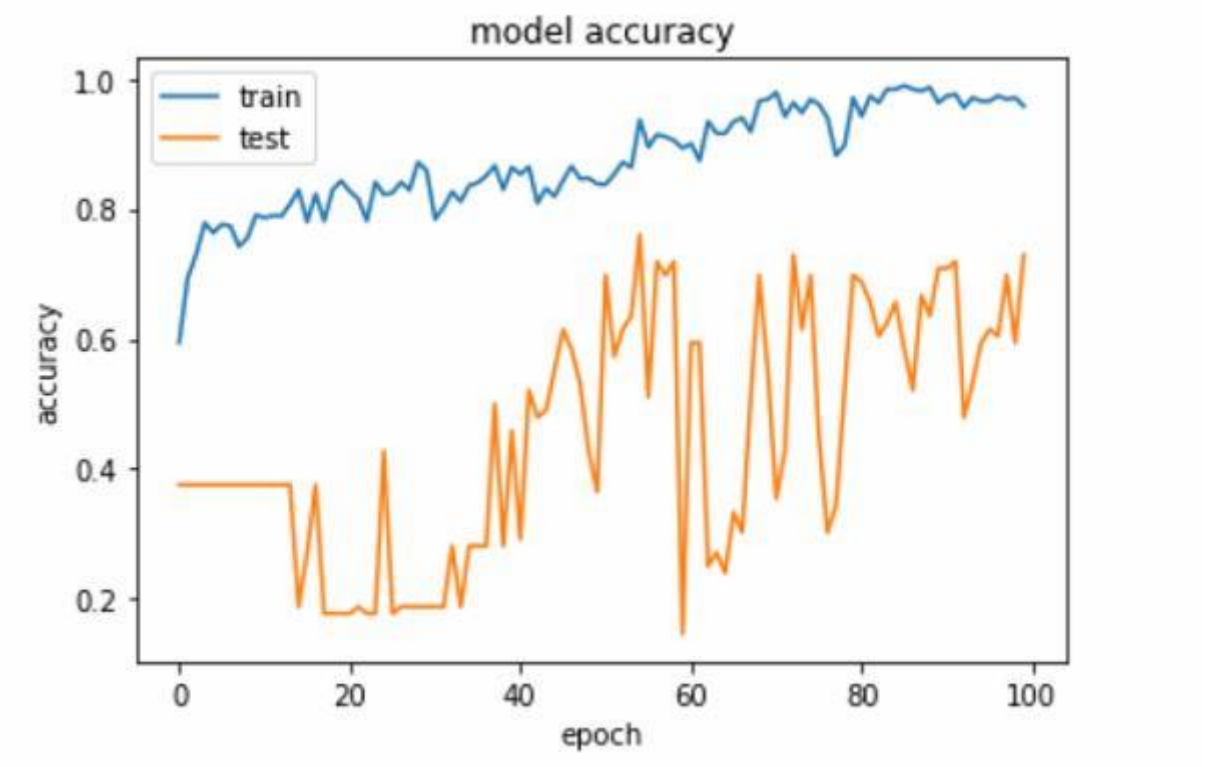

Figure.72 Xception_4 Accuracy Plot

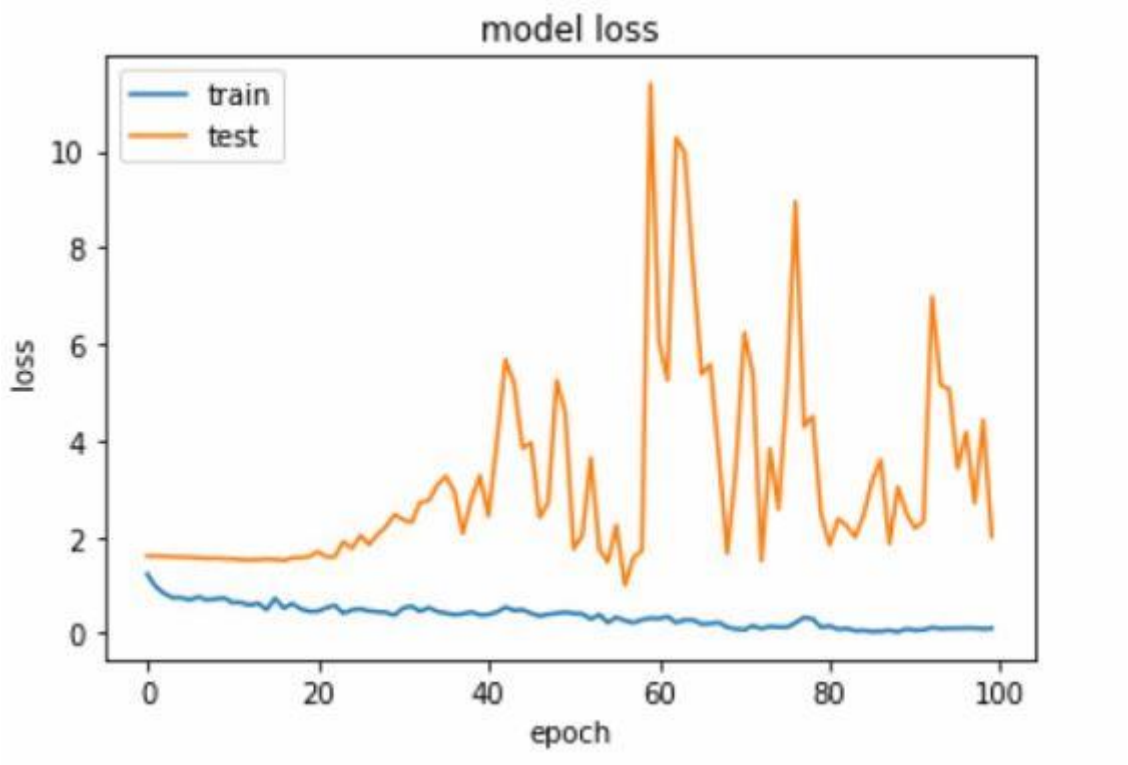

Figure.73 Xception_4 Accuracy Loss Plot

2.3.12) Pyhton Model Result

Model Name	Accuracy Plot (%)
Vgg16-2	92
Vgg16-3	91
Xception-4	77
Xception-3	75
Vgg16-1	75
Resnet50-2	71
Xception-2	70
Inception	69
Xception-1	69
Resnet50	53
MobileNet	21

Figure.74 Pyhton Models Result Table

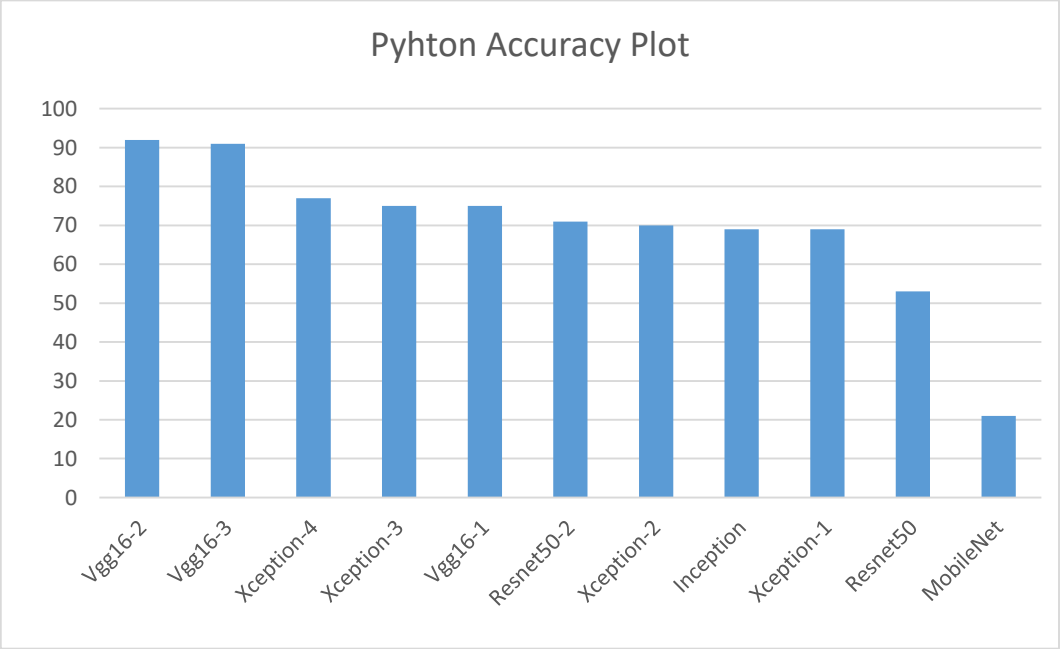

Figure.75 Pyhton Models Result Plot

3) CONCLUSION

According to the results obtained in the project 12 different model in Matlab and 5 different model trained in Pyhton. In a result table shown in below most of the model result almost same in Pyhton and Matlab.

Model Name	Matlab (%)	Pyhton (%)
Xception	87	77
Vgg16	78	92
Vgg19	77	35
Resnet50	72	71
Inception	71	69

Figure.76 Result Accuracy Table

After anaylized table Xception model best fit for marble data in matlab and Vgg16 model best fit for marble data in pyhton. But somehow Vgg19 never reached to over of 0.35 acccuracy ratio.

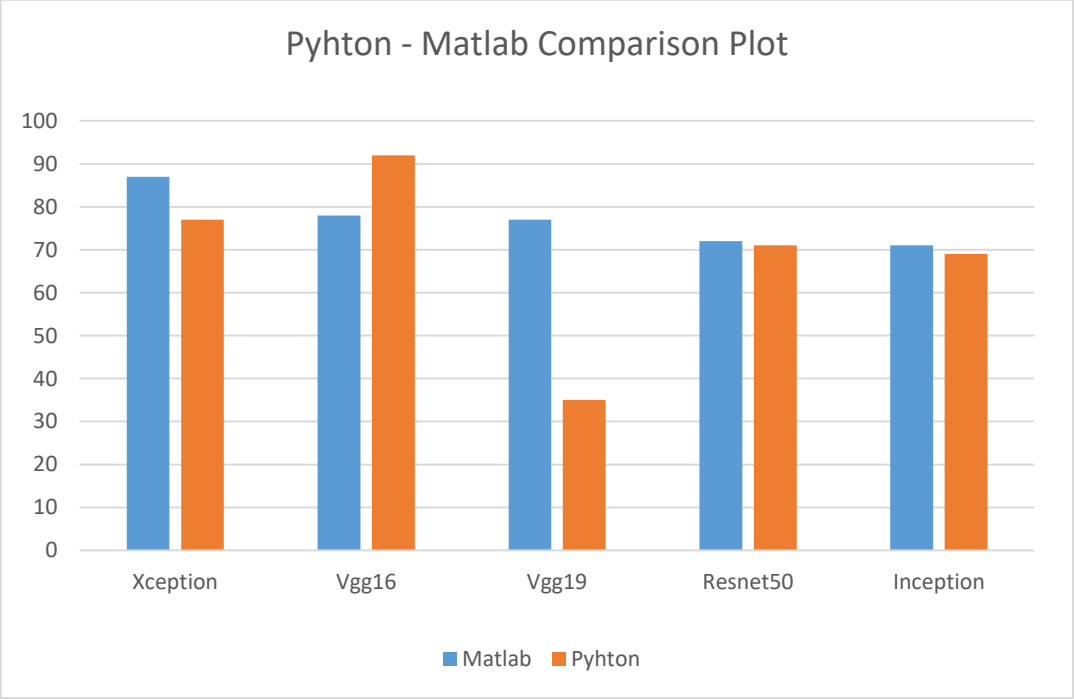

Figure.77 Result Accuracy Plot