`Import` all imperative libraies for `building` the `model`

In [2]:
```
##### Import all necessity functions for Machine Learning #####
import sys
import math
import string
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as shc
import warnings
import zipfile
import cv2
import os
import random
from collections import Counter
from functools import reduce
from itertools import chain
from google.colab.patches import cv2_imshow
from keras.preprocessing import image
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.feature_selection import mutual_info_classif, mutual_info_regression, SelectKBest, chi2, VarianceT
from imblearn.under_sampling import RandomUnderSampler, NearMiss
from imblearn.over_sampling import RandomOverSampler, SMOTE, SMOTEN, SMOTENC, SVMSMOTE, KMeansSMOTE, Borderline
from imblearn.ensemble import EasyEnsembleClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor, NearestNeighbors
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier, SGDRegressor, Perceptron
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, ExtraTreeClassifier, ExtraTreeRegressor
from sklearn.ensemble import BaggingClassifier, BaggingRegressor, RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor, GradientBoostingClassifier, GradientBoostin
from sklearn.metrics import classification_report, mean_absolute_error, mean_squared_error, r2_score, accuracy_
from xgboost import XGBClassifier, XGBRegressor

##### Download keras #####
!pip install keras

##### Import all necessity functions for Neural Network #####
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Dense, Conv2D, LSTM, GRU, RNN, Flatten, AvgPool2D, MaxPool2D, GlobalAverage
from tensorflow.keras.activations import tanh, relu, sigmoid, softmax, swish
from tensorflow.keras.regularizers import L1, L2, L1L2
from tensorflow.keras.optimizers import SGD, Adagrad, Adadelta, RMSprop, Adam, Adamax, Nadam
from tensorflow.keras.initializers import HeNormal, HeUniform, GlorotNormal, GlorotUniform
from tensorflow.keras.losses import SparseCategoricalCrossentropy, CategoricalCrossentropy, hinge, MSE, MAE, Hu
import keras.utils as image

#### For Functional API ####
from tensorflow.keras.models import Model
from tensorflow.keras.layers import *
from tensorflow.keras.layers import concatenate, Input,Dense, Conv2D,\
 LSTM, GRU, RNN, Flatten, AvgPool2D, MaxPool2D, GlobalAveragePooling2D,\
  BatchNormalization, Dropout, LeakyReLU, ELU, PReLU

##### Plotting the confusion matrix #####
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import confusion_matrix

##### Remove all warnings #####
import warnings
warnings.filterwarnings("ignore")
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: keras in /usr/local/lib/python3.9/dist-packages (2.12.0)
```

`Access` the `Google Drive`

In [3]:
```
##### To Access the Google Drive #####
def google_drive(parameter = None):
  try:
    from google.colab import drive
    drive.mount('/content/drive',force_remount=True)
  except Exception as e:
```

```
    print(e.with_traceback)
  else:
    print('\nGoogle Drive access is done.\n'.title())

google_drive()
```

```
Mounted at /content/drive

Google Drive Access Is Done.
```

`Unzip` the folder of `Alzheimer` disease

In [4]:

```
##### To Unzip the folder #####
def unzip_file(parameter = None):
  try:
    link_folder = '/content/drive/MyDrive/CNN Dataset/Alzheimer_Disease_Update.zip'
    zip_ref = zipfile.ZipFile(link_folder, 'r')
    zip_ref.extractall()
    zip_ref.close()
  except Exception as e:
    print(e.with_traceback)
  else:
    print('Upzip is done succesfully'.title())

##### Call the Unzip function #####
unzip_file()
```

```
Upzip Is Done Succesfully
```

`Extract` the `train` folder with `Independent & Dependent` Features

In [5]:

```
##### Extract the Images from the Folder #####
try:
  _DIRECTORY   = '/content/Combined Dataset/train'
  _CATEGORIES  = ['Mild Impairment', 'Moderate Impairment', 'No Impairment', 'Very Mild Impairment']
  train_data   = []

  for each_category_ in _CATEGORIES:
    folder_path_ = os.path.join(_DIRECTORY, each_category_)
    """
    /content/Alzheimer_s _Dataset/MildDemented
    /content/Alzheimer_s _Dataset/ModerateDemented
    /content/Alzheimer_s _Dataset/NonDemented
    /content/Alzheimer_s _Dataset/VeryMildDemented
    """
    for all_images_ in os.listdir(folder_path_):
      each_image_directory = os.path.join(folder_path_, all_images_)
      """
      /content/Alzheimer_s _Dataset/MildDemented/mildDem697.jpg
      /content/Alzheimer_s _Dataset/ModerateDemented/32 (2).jpg
      /content/Alzheimer_s _Dataset/NonDemented/nonDem1118.jpg
      /content/Alzheimer_s _Dataset/VeryMildDemented/verymildDem892.jpg
      """
      ##### Read the images and converted into NumPy format using CV #####
      image_array_    = cv2.imread(each_image_directory)
      ##### Converted the images into a fixed size #####
      reshaped_arary_ = cv2.resize(image_array_, (128, 128))
      ##### find the categories from that image folder #####
      target_class_   = _CATEGORIES.index(each_category_)
      """
      0 1 2 3
      """
      ##### Apend the reshaped array(features of each images) with the traget class #####
      train_data.append([reshaped_arary_, target_class_])

    print('{} folder is completed - Feature Extracted with Target Class'.format(each_category_).capitalize(),'\

except Exception as e:
  print(e.with_traceback)
else:
  print('Completed.\n'.title())
```

```
Mild impairment folder is completed - feature extracted with target class

Moderate impairment folder is completed - feature extracted with target class

No impairment folder is completed - feature extracted with target class

Very mild impairment folder is completed - feature extracted with target class

Completed.
```

`Extract` the `validation` data with `Independent & Dependent` Features

In [6]:

```
##### Extract the Images from the Folder #####
```

```
try:
  _DIRECTORY   = '/content/Combined Dataset/test'
  _CATEGORIES  = ['Mild Impairment', 'Moderate Impairment', 'No Impairment', 'Very Mild Impairment']
  validation_data   = []

  for each_category_ in _CATEGORIES:
    folder_path_ = os.path.join(_DIRECTORY, each_category_)
    """
    /content/Alzheimer_s _Dataset/MildDemented
    /content/Alzheimer_s _Dataset/ModerateDemented
    /content/Alzheimer_s _Dataset/NonDemented
    /content/Alzheimer_s _Dataset/VeryMildDemented
    """
    for all_images_ in os.listdir(folder_path_):
      each_image_directory = os.path.join(folder_path_, all_images_)
      """
      /content/Alzheimer_s _Dataset/MildDemented/mildDem697.jpg
      /content/Alzheimer_s _Dataset/ModerateDemented/32 (2).jpg
      /content/Alzheimer_s _Dataset/NonDemented/nonDem1118.jpg
      /content/Alzheimer_s _Dataset/VeryMildDemented/verymildDem892.jpg
      """
      ##### Read the images and converted into NumPy format using CV #####
      image_array_    = cv2.imread(each_image_directory)
      ##### Converted the images into a fixed size #####
      reshaped_arary_ = cv2.resize(image_array_, (128, 128))
      ##### find the categories from that image folder #####
      target_class_   = _CATEGORIES.index(each_category_)
      """
      0 1 2 3
      """
      ##### Apend the reshaped array(features of each images) with the traget class #####
      validation_data.append([reshaped_arary_, target_class_])

    print('{} folder is completed - Feature Extracted with Target Class'.format(each_category_).capitalize(),'\

except Exception as e:
  print(e.with_traceback)
else:
  print('Completed.\n'.title())
```

```
Mild impairment folder is completed - feature extracted with target class

Moderate impairment folder is completed - feature extracted with target class

No impairment folder is completed - feature extracted with target class

Very mild impairment folder is completed - feature extracted with target class

Completed.
```

Check the `length` of `Train & Validation` dataset

In [7]:
```
class listEmptyException(Exception):
  def __init__(self, message):
    return message

def check_shape(train = None, validation = None):
  if (len(train) != 0 and len(validation) !=0):
    return len(train), len(validation)
  else:
    raise Exception('List is empty.'.capitalize())

try:
  train_shape, test_shape = check_shape(train = train_data, validation = validation_data)
except listEmptyException as e:
  print(e)
else:
  print('The length of train dataset is = {}'.format(train_shape))
  print('The length of validation dataset is = {}'.format(test_shape))
```

```
The length of train dataset is = 10240
The length of validation dataset is = 1279
```

`Extract` the `Independent & Dependent` Features with respect to `train & validation`

In [8]:
```
def extract_independent_dependent_features(data = None):
  #### Initialization X <- Independent and y <- Dependent ####
  X, y = [], []

  for (extracted_feature, target_class) in data:
    X.append(extracted_feature)
    y.append(target_class)

  return X, y

#### Call the 'extract_independent_dependent_features' ####
```

```
try:
  #### Shuffle the stored_data_ due to prevent the biasness ####
  random.shuffle(train_data)
  random.shuffle(validation_data)
  #### Call the function ####
  X_train, y_train = extract_independent_dependent_features(train_data)
  X_val, y_val     = extract_independent_dependent_features(validation_data)
except Exception as e:
  print(e.with_traceback)
else:
  print('Extraction is completed with independent and dependent'.title())
```

```
Extraction Is Completed With Independent And Dependent
```

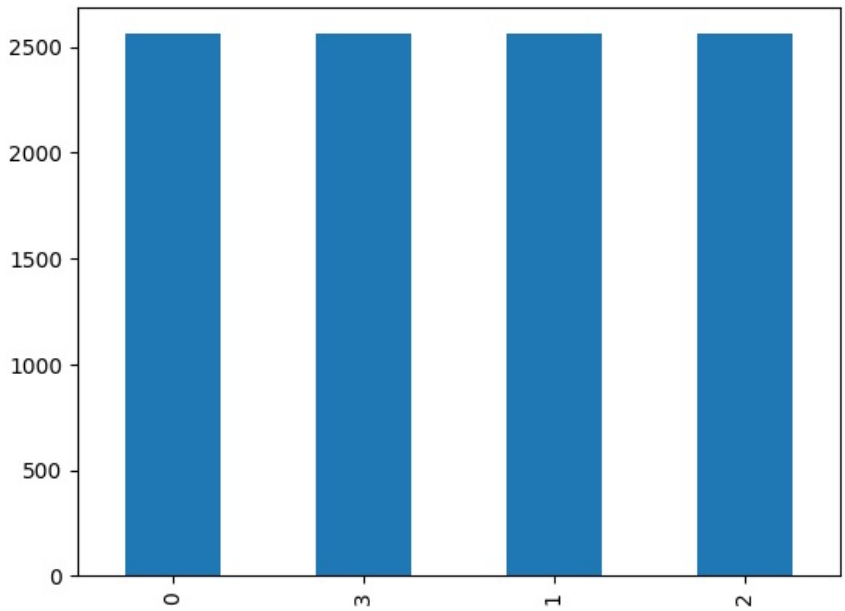
Convert the `Independent & Dependent` Features into `NumPy` Foramt

In [9]:
```
#### Convered the independent and dependent features into NumPy format ####
def converted_NumPy(independent = None, dependent = None):
  return np.array(independent), np.array(dependent)

#### Call 'converted_NumPy' function with X and y ####
try:
  X_train, y_train = converted_NumPy(X_train, y_train)
  X_val, y_val = converted_NumPy(X_val, y_val)
except Exception as e:
  print(e.with_traceback)
else:
  print('Converted into NumPy format successfully.'.title(),'\n')
  print('The shape of train dataset is = {}'.format(X_train.shape),'\n')
  print('The shape of validation dataset is = {}'.format(X_val.shape))
```

```
Converted Into Numpy Format Successfully.

The shape of train dataset is = (10240, 128, 128, 3)

The shape of validation dataset is = (1279, 128, 128, 3)
```

Plot the `countplot` of the `Target Class`

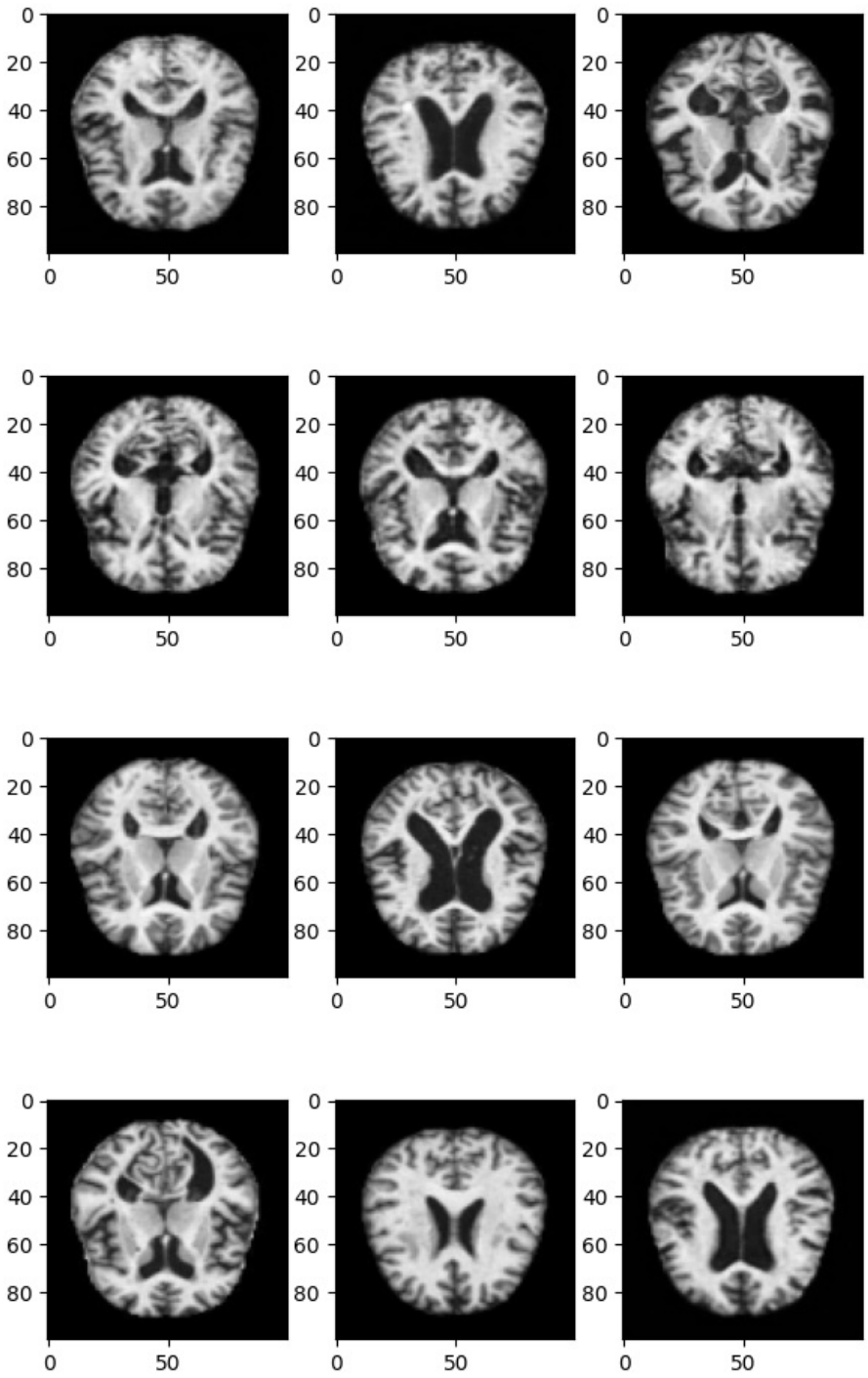
In [10]:
```
#### Define a function that returns the value counts ####
def countplot(target_class):
  if len(target_class) != 0:
    return pd.DataFrame(target_class)[0].value_counts()
  else:
    raise Exception('List is empty'.title())

#### Call this function from this current directory ####
try:
  dataframe = countplot(y_train)
except listEmptyException as e:
  print(e)
else:
  dataframe.plot(kind = 'bar')
  plt.show()
```



Plot the `Images`

In [11]:
```
#### Define a class that throws the user defined exception ####
def plotException(Exception):
  def __init__(self, message):
```

```
    return "plot exception".title()

#### Plot 3 images from each folders ####
_DIRECTORY   = '/content/Combined Dataset/train'
_CATEGORIES  = ['Mild Impairment', 'Moderate Impairment', 'No Impairment', 'Very Mild Impairment']
plot_dt_dic = {}

for each_category_ in _CATEGORIES:
  folder_path_ = os.path.join(_DIRECTORY, each_category_)
  count_, list_ = 0, []
  for all_images_ in os.listdir(folder_path_):
    if count_ < 3:
      each_image_directory = os.path.join(folder_path_, all_images_)
      #### In order to make the computation cost minimised the image size is = (100, 100) ####
      list_.append(cv2.resize(cv2.imread(each_image_directory), (100, 100)))
      count_ = count_ + 1
    else:
      break
  plot_dt_dic[each_category_] = list_

#### Plot the iamges ####
try:
  fig, ax = plt.subplots(4, 3, figsize = (7, 12))
  ax[0, 0].imshow(plot_dt_dic['Mild Impairment'][0])
  ax[0, 1].imshow(plot_dt_dic['Mild Impairment'][1])
  ax[0, 2].imshow(plot_dt_dic['Mild Impairment'][2])

  ax[1, 0].imshow(plot_dt_dic['Moderate Impairment'][0])
  ax[1, 1].imshow(plot_dt_dic['Moderate Impairment'][1])
  ax[1, 2].imshow(plot_dt_dic['Moderate Impairment'][2])

  ax[2, 0].imshow(plot_dt_dic['No Impairment'][0])
  ax[2, 1].imshow(plot_dt_dic['No Impairment'][1])
  ax[2, 2].imshow(plot_dt_dic['No Impairment'][2])

  ax[3, 0].imshow(plot_dt_dic['Very Mild Impairment'][0])
  ax[3, 1].imshow(plot_dt_dic['Very Mild Impairment'][1])
  ax[3, 2].imshow(plot_dt_dic['Very Mild Impairment'][2])
  plt.show()

except plotException as e:
  print(e)
```



Do the `Normalization` with respect to `Independent` Features

In [12]:
```
#### Normalized the independent features ####
def Normalized(independent):
  if len(independent) == 0:
     raise Exception('List is empty.'.title())
  else:
    return (independent/255)

#### Call the Normalized function ####
try:
  X_train_normalised = Normalized(X_train)
  X_validation_normalised = Normalized(X_val)
except listEmptyException as e:
  print("The exception is {}".format(e))
except Exception as e:
  print(e)
else:
  print('Normalization is done.\n'.title())
```

```
Normalization Is Done.
```

Split the dataset into `train & test` with respect to train dataset

In [13]:
```
#### Define the function that is responsible to train and test ####
def train_test_split(train_dataset = None, y_train = None):
```

```
  from sklearn.model_selection import train_test_split
  if len(train_dataset)  == 0:
    raise Exception('List is empty'.title(),'\n')
  else:
    X_train, X_test, y_train, y_test = train_test_split(train_dataset,\
                                y_train,\
                                test_size = 0.25,\
                                random_state = 42)
    return X_train, X_test, y_train, y_test


#### Call the function with this current directory ####
try:
  X_train, X_test, y_train, y_test = train_test_split(train_dataset = X_train_normalised, y_train = y_train)
except listEmptyException as e:
  print("The exception is {}".format(e))
except Exception as e:
  print("The exception is {}".format(e))
else:
  print('The shape of X_train is = {} '.format(X_train.shape),'\n'.capitalize())
  print('The shape of X_test  is = {} '.format(X_test.shape),'\n'.capitalize())
  print('The shape of y_train is = {} '.format(y_train.shape),'\n'.capitalize())
  print('The shape of y_test  is = {} '.format(y_test.shape),'\n'.capitalize())
```

```
The shape of X_train is = (7680, 128, 128, 3)

The shape of X_test  is = (2560, 128, 128, 3)

The shape of y_train is = (7680,)

The shape of y_test  is = (2560,)
```

Find `Class Weight` with respect to class

In [14]:
```
##### Call the class_weight #####
def class_weight(y_train = None):
  if len(y_train) == 0:
    raise Exception('List is empty'.title())
  else:
    # Calculate weights using sklearn
    from sklearn.utils import class_weight
    sklearn_weights =  class_weight.compute_class_weight(
          class_weight='balanced',
            classes=np.unique(y_train),
            y = y_train)

    print("Weights is :", sklearn_weights,'\n\n')

    # Transform array to dictionary
    sklearn_weights = dict(enumerate(sklearn_weights))
    print("Weights is :", sklearn_weights)

  return sklearn_weights

try:
  sklearn_weights = class_weight(y_train = y_train)
except listEmptyException as e:
  print('The exception is {}'.format(e))
```

```
Weights is : [1.00208768 1.00628931 0.98765432 1.0041841 ]


Weights is : {0: 1.0020876826722338, 1: 1.0062893081761006, 2: 0.9876543209876543, 3: 1.00418410041841}
```

Define `Vanilla CNN architecture` using `Functional` API for training the model

In [71]:
```
#### Define the input shape ####
input_shape = (128, 128, 3)
input_shape = Input(shape = input_shape)

#### Define the left hidden layer ####
left_hidden1 = Conv2D(filters = 64,\
                      kernel_size = (3, 3),\
                      strides = (2, 2),\
                      padding = 'same',\
                      activation = 'relu',\
                      kernel_initializer = 'he_normal')(input_shape)

#### Define the second left hidden layer ####
left_hidden2 = Conv2D(filters = 32,\
                      kernel_size = (3, 3),\
                      strides = (2, 2),\
                      padding = 'same',\
```

```
                            activation = 'relu',\
                            kernel_initializer = 'he_normal')(left_hidden1)

#### Use the Dropout rate with 0.4 ####
left_hidden2 = Dropout(rate = 0.4)(left_hidden2)

#### Do the Flatten operation ####
left_flatten = Flatten()(left_hidden2)

#### For the right Fuctional API ####
right_hidden1 = Conv2D(filters = 256,\
                            kernel_size = (3, 3),\
                            strides = (2, 2),\
                            padding = 'valid',\
                            activation = 'relu',\
                            kernel_initializer = 'he_normal')(input_shape)

####Define the right second hidden layer of Conv2d###
right_hidden2 = Conv2D(filters = 128,\
                            kernel_size = (3, 3),\
                            strides = (2, 2),\
                            padding = 'valid',\
                            activation = 'relu',\
                            kernel_initializer = 'he_normal')(right_hidden1)

#### Do the Flatten operation ####
right_flatten = Flatten()(right_hidden2)

#### Do the concatenation ####
combined = concatenate([left_flatten, right_flatten])

#### Use the Dropout rate = 0.3 ####
combined = Dropout(rate = 0.3)(combined)

#### Do the Fully conneted layer ####
first_hidden1 = Dense(units = 256,\
                      activation = 'relu',\
                      kernel_initializer = 'he_normal',\
                      kernel_regularizer = L2(l2 = 0.01))(combined)

#### Use the Dropout layer with ratio 0.5 ####
first_hidden1 = Dropout(rate = 0.5)(first_hidden1)

#### Use the Fully connected layer ####
second_hidden2 = Dense(units = 128,\
                       activation = 'relu',\
                       kernel_initializer = 'he_normal',\
                       kernel_regularizer = L2(l2 = 0.001))(first_hidden1)


second_hidden2 = concatenate([second_hidden2, combined])


#### Use the Dropout ratio 0.5 ####
second_hidden2 = Dropout(rate = 0.6)(second_hidden2)

#### Use the thrid hidden layer ####
third_hidden = Dense(units = 64,\
                       activation = 'relu',\
                       kernel_initializer = 'he_normal',\
                       kernel_regularizer = L2(l2 = 0.005))(second_hidden2)

#### Use the dropout ratio 0.5 ####
second_hidden2 = Dropout(rate = 0.5)(third_hidden)

#### This is responsible for the output layer ####
output_layer = Dense(units = 4,\
                     activation = 'softmax')(second_hidden2)

#### Connect the inputs and outputs in Model function for the Functional API ####
model = Model(inputs = input_shape,\
              outputs = output_layer)

#### This is responsible for the compile the model ####
model.compile(optimizer = Adam(learning_rate = 0.0005),\
              loss = SparseCategoricalCrossentropy(),\
              metrics = ['accuracy'])

model.summary()
```

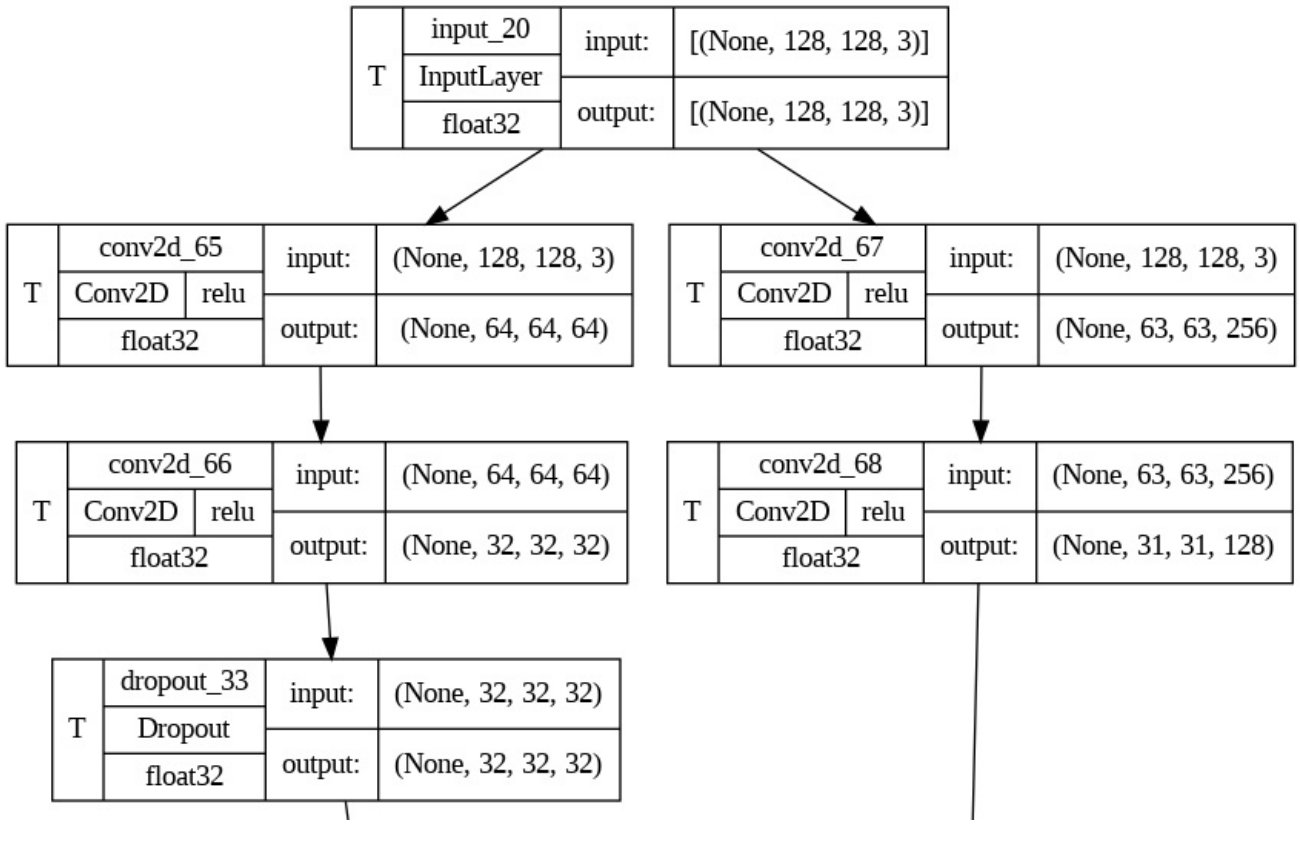
```
Model: "model_9"
```

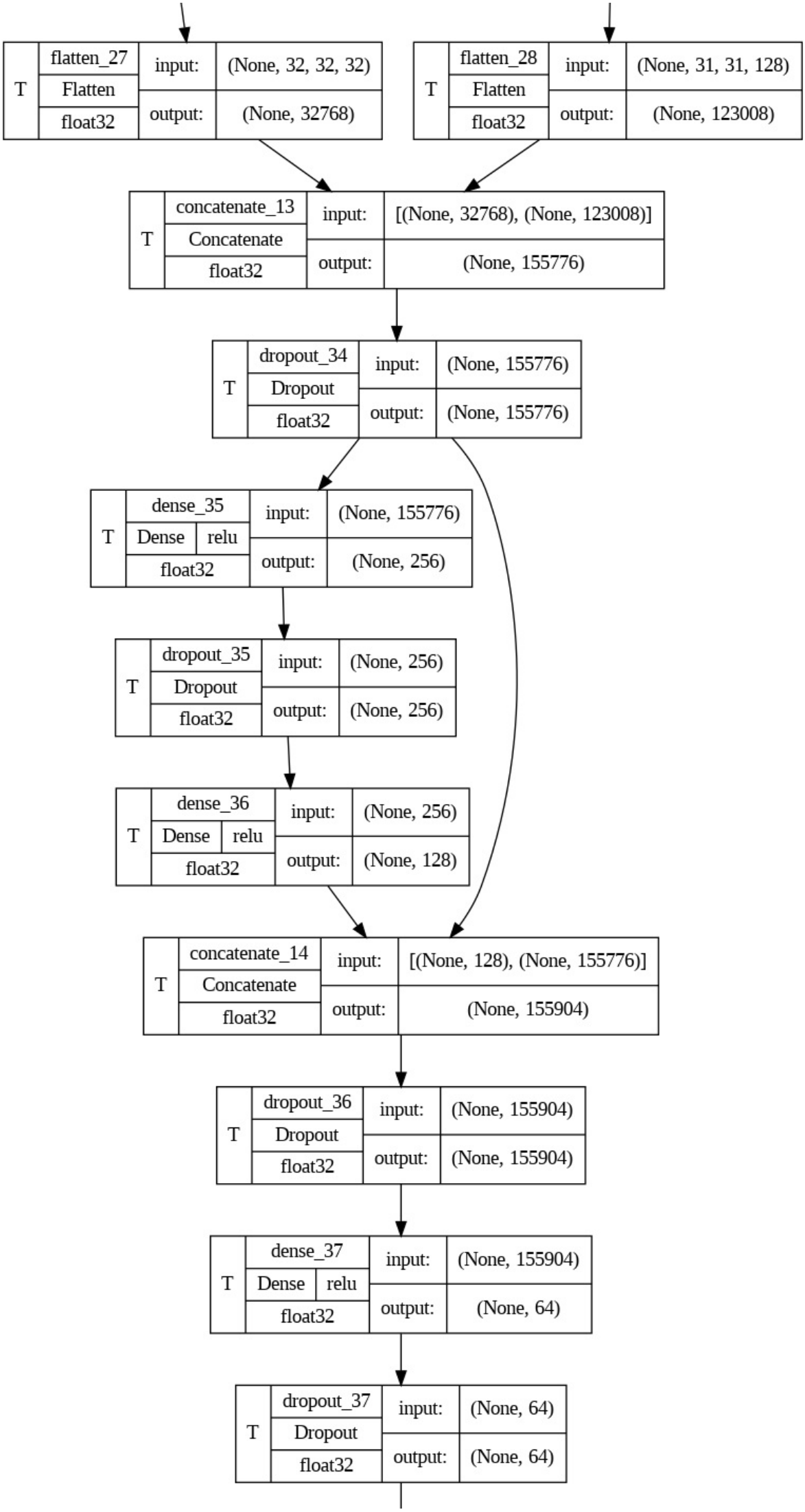
Layer (type)	Output Shape	Param #	Connected to
input_20 (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d_65 (Conv2D)	(None, 64, 64, 64)	1792	['input_20[0][0]']
conv2d_66 (Conv2D)	(None, 32, 32, 32)	18464	['conv2d_65[0][0]']
conv2d_67 (Conv2D)	(None, 63, 63, 256)	7168	['input_20[0][0]']
dropout_33 (Dropout)	(None, 32, 32, 32)	0	['conv2d_66[0][0]']
conv2d_68 (Conv2D)	(None, 31, 31, 128)	295040	['conv2d_67[0][0]']
flatten_27 (Flatten)	(None, 32768)	0	['dropout_33[0][0]']
flatten_28 (Flatten)	(None, 123008)	0	['conv2d_68[0][0]']
concatenate_13 (Concatenate)	(None, 155776)	0	['flatten_27[0][0]', 'flatten_28[0][0]']
dropout_34 (Dropout)	(None, 155776)	0	['concatenate_13[0][0]']
dense_35 (Dense)	(None, 256)	39878912	['dropout_34[0][0]']
dropout_35 (Dropout)	(None, 256)	0	['dense_35[0][0]']
dense_36 (Dense)	(None, 128)	32896	['dropout_35[0][0]']
concatenate_14 (Concatenate)	(None, 155904)	0	['dense_36[0][0]', 'dropout_34[0][0]']
dropout_36 (Dropout)	(None, 155904)	0	['concatenate_14[0][0]']
dense_37 (Dense)	(None, 64)	9977920	['dropout_36[0][0]']
dropout_37 (Dropout)	(None, 64)	0	['dense_37[0][0]']
dense_38 (Dense)	(None, 4)	260	['dropout_37[0][0]']

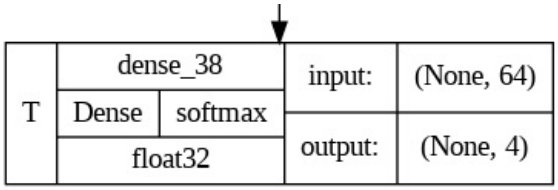
```
Total params: 50,212,452
Trainable params: 50,212,452
Non-trainable params: 0
```

In [72]:
```
plot_model(model = model,\
           show_shapes = True,\
           show_dtype = True,\
           show_layer_names = True,\
           show_layer_activations = True,\
           show_trainable = True)
```

Out[72]:




T
flatten_27
Flatten
float32
input:
output:
(None, 32, 32, 32)
(None, 32768)
T
flatten_28
Flatten
float32
input:
output:
(None, 31, 31, 128)
(None, 123008)
T
concatenate_13
Concatenate
float32
input:
output:
[(None, 32768), (None, 123008)]
(None, 155776)
T
dropout_34
Dropout
float32
input:
output:
(None, 155776)
(None, 155776)
T
dense_35
Dense
relu
float32
input:
output:
(None, 155776)
(None, 256)
T
dropout_35
Dropout
float32
input:
output:
(None, 256)
(None, 256)
T
dense_36
Dense
relu
float32
input:
output:
(None, 256)
(None, 128)
T
concatenate_14
Concatenate
float32
input:
output:
[(None, 128), (None, 155776)]
(None, 155904)
T
dropout_36
Dropout
float32
input:
output:
(None, 155904)
(None, 155904)
T
dense_37
Dense
relu
float32
input:
output:
(None, 155904)
(None, 64)
T
dropout_37
Dropout
float32
input:
output:
(None, 64)
(None, 64)



In [73]:
```
history = model.fit(x = X_train,\
                    y = y_train,\
                    epochs = 100,\
                    batch_size = 128,\
                    validation_data = (X_test, y_test),\
                    verbose = 1)
```

```
Epoch 1/100
60/60 [==============================] - 21s 219ms/step - loss: 2.5063 - accuracy: 0.3638 - val_loss: 1.3635 -
val_accuracy: 0.5930
Epoch 2/100
60/60 [==============================] - 12s 203ms/step - loss: 1.3261 - accuracy: 0.5652 - val_loss: 1.0224 -
val_accuracy: 0.7078
Epoch 3/100
60/60 [==============================] - 12s 205ms/step - loss: 1.1256 - accuracy: 0.6281 - val_loss: 0.8776 -
val_accuracy: 0.8305
Epoch 4/100
60/60 [==============================] - 12s 204ms/step - loss: 1.0207 - accuracy: 0.6612 - val_loss: 0.8019 -
val_accuracy: 0.8371
Epoch 5/100
60/60 [==============================] - 12s 201ms/step - loss: 0.9539 - accuracy: 0.6682 - val_loss: 0.7417 -
val_accuracy: 0.8578
Epoch 6/100
60/60 [==============================] - 12s 199ms/step - loss: 0.9098 - accuracy: 0.6842 - val_loss: 0.6502 -
val_accuracy: 0.8633
Epoch 7/100
60/60 [==============================] - 12s 200ms/step - loss: 0.8581 - accuracy: 0.6820 - val_loss: 0.6233 -
val_accuracy: 0.8805
Epoch 8/100
60/60 [==============================] - 12s 200ms/step - loss: 0.8404 - accuracy: 0.6885 - val_loss: 0.6271 -
val_accuracy: 0.8848
Epoch 9/100
60/60 [==============================] - 12s 201ms/step - loss: 0.8108 - accuracy: 0.6958 - val_loss: 0.5644 -
val_accuracy: 0.8820
Epoch 10/100
60/60 [==============================] - 12s 202ms/step - loss: 0.7878 - accuracy: 0.6980 - val_loss: 0.5181 -
val_accuracy: 0.8965
Epoch 11/100
60/60 [==============================] - 12s 202ms/step - loss: 0.7806 - accuracy: 0.7044 - val_loss: 0.5322 -
val_accuracy: 0.8875
Epoch 12/100
60/60 [==============================] - 12s 201ms/step - loss: 0.7632 - accuracy: 0.7118 - val_loss: 0.4667 -
val_accuracy: 0.9023
Epoch 13/100
60/60 [==============================] - 12s 201ms/step - loss: 0.7671 - accuracy: 0.7125 - val_loss: 0.5148 -
val_accuracy: 0.9023
Epoch 14/100
60/60 [==============================] - 12s 201ms/step - loss: 0.7370 - accuracy: 0.7276 - val_loss: 0.4777 -
val_accuracy: 0.9148
Epoch 15/100
60/60 [==============================] - 12s 200ms/step - loss: 0.7652 - accuracy: 0.7210 - val_loss: 0.4884 -
val_accuracy: 0.9105
Epoch 16/100
60/60 [==============================] - 12s 201ms/step - loss: 0.7393 - accuracy: 0.7288 - val_loss: 0.4540 -
val_accuracy: 0.9250
Epoch 17/100
60/60 [==============================] - 12s 200ms/step - loss: 0.7481 - accuracy: 0.7260 - val_loss: 0.4744 -
val_accuracy: 0.9199
Epoch 18/100
60/60 [==============================] - 13s 209ms/step - loss: 0.7297 - accuracy: 0.7431 - val_loss: 0.4346 -
val_accuracy: 0.9301
Epoch 19/100
60/60 [==============================] - 12s 201ms/step - loss: 0.7118 - accuracy: 0.7578 - val_loss: 0.4515 -
val_accuracy: 0.9332
Epoch 20/100
60/60 [==============================] - 12s 202ms/step - loss: 0.6900 - accuracy: 0.7728 - val_loss: 0.4426 -
val_accuracy: 0.8969
Epoch 21/100
60/60 [==============================] - 12s 202ms/step - loss: 0.6778 - accuracy: 0.7780 - val_loss: 0.4645 -
val_accuracy: 0.9074
Epoch 22/100
60/60 [==============================] - 12s 202ms/step - loss: 0.6943 - accuracy: 0.7844 - val_loss: 0.4483 -
val_accuracy: 0.9402
Epoch 23/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6911 - accuracy: 0.7823 - val_loss: 0.4135 -
val_accuracy: 0.9426
Epoch 24/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6810 - accuracy: 0.7906 - val_loss: 0.4320 -
val_accuracy: 0.9414
Epoch 25/100
```

```
60/60 [==============================] - 12s 201ms/step - loss: 0.6652 - accuracy: 0.7953 - val_loss: 0.4278 -
val_accuracy: 0.9422
Epoch 26/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6715 - accuracy: 0.7971 - val_loss: 0.4148 -
val_accuracy: 0.9441
Epoch 27/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6549 - accuracy: 0.8040 - val_loss: 0.4243 -
val_accuracy: 0.9461
Epoch 28/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6787 - accuracy: 0.7996 - val_loss: 0.4106 -
val_accuracy: 0.9547
Epoch 29/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6617 - accuracy: 0.8034 - val_loss: 0.4349 -
val_accuracy: 0.9375
Epoch 30/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6596 - accuracy: 0.8070 - val_loss: 0.4342 -
val_accuracy: 0.9445
Epoch 31/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6557 - accuracy: 0.8156 - val_loss: 0.4032 -
val_accuracy: 0.9488
Epoch 32/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6528 - accuracy: 0.8122 - val_loss: 0.4112 -
val_accuracy: 0.9426
Epoch 33/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6457 - accuracy: 0.8203 - val_loss: 0.4023 -
val_accuracy: 0.9430
Epoch 34/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6210 - accuracy: 0.8283 - val_loss: 0.3927 -
val_accuracy: 0.9422
Epoch 35/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6175 - accuracy: 0.8251 - val_loss: 0.4071 -
val_accuracy: 0.9238
Epoch 36/100
60/60 [==============================] - 12s 201ms/step - loss: 0.6277 - accuracy: 0.8276 - val_loss: 0.4218 -
val_accuracy: 0.9441
Epoch 37/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6291 - accuracy: 0.8264 - val_loss: 0.3801 -
val_accuracy: 0.9516
Epoch 38/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6107 - accuracy: 0.8348 - val_loss: 0.3637 -
val_accuracy: 0.9543
Epoch 39/100
60/60 [==============================] - 12s 201ms/step - loss: 0.5994 - accuracy: 0.8327 - val_loss: 0.3842 -
val_accuracy: 0.9516
Epoch 40/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6182 - accuracy: 0.8345 - val_loss: 0.3984 -
val_accuracy: 0.9508
Epoch 41/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6150 - accuracy: 0.8417 - val_loss: 0.3763 -
val_accuracy: 0.9566
Epoch 42/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6153 - accuracy: 0.8438 - val_loss: 0.4398 -
val_accuracy: 0.9305
Epoch 43/100
60/60 [==============================] - 12s 200ms/step - loss: 0.6011 - accuracy: 0.8475 - val_loss: 0.3948 -
val_accuracy: 0.9605
Epoch 44/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5957 - accuracy: 0.8561 - val_loss: 0.4093 -
val_accuracy: 0.9422
Epoch 45/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5878 - accuracy: 0.8501 - val_loss: 0.3731 -
val_accuracy: 0.9637
Epoch 46/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5739 - accuracy: 0.8596 - val_loss: 0.3980 -
val_accuracy: 0.9570
Epoch 47/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5852 - accuracy: 0.8685 - val_loss: 0.4301 -
val_accuracy: 0.9434
Epoch 48/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5758 - accuracy: 0.8727 - val_loss: 0.4207 -
val_accuracy: 0.9523
Epoch 49/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5683 - accuracy: 0.8767 - val_loss: 0.4053 -
val_accuracy: 0.9617
Epoch 50/100
60/60 [==============================] - 12s 200ms/step - loss: 0.5715 - accuracy: 0.8727 - val_loss: 0.3978 -
val_accuracy: 0.9563
Epoch 51/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5651 - accuracy: 0.8772 - val_loss: 0.3927 -
val_accuracy: 0.9664
Epoch 52/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5462 - accuracy: 0.8831 - val_loss: 0.3847 -
val_accuracy: 0.9699
Epoch 53/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5525 - accuracy: 0.8762 - val_loss: 0.3841 -
val_accuracy: 0.9703
Epoch 54/100
60/60 [==============================] - 12s 207ms/step - loss: 0.5662 - accuracy: 0.8818 - val_loss: 0.4269 -
val_accuracy: 0.9516
```

```
Epoch 55/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5584 - accuracy: 0.8905 - val_loss: 0.3846 -
val_accuracy: 0.9723
Epoch 56/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5522 - accuracy: 0.8831 - val_loss: 0.3965 -
val_accuracy: 0.9656
Epoch 57/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5432 - accuracy: 0.8837 - val_loss: 0.3848 -
val_accuracy: 0.9691
Epoch 58/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5224 - accuracy: 0.8973 - val_loss: 0.3818 -
val_accuracy: 0.9723
Epoch 59/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5278 - accuracy: 0.8908 - val_loss: 0.3777 -
val_accuracy: 0.9730
Epoch 60/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5277 - accuracy: 0.8936 - val_loss: 0.3738 -
val_accuracy: 0.9738
Epoch 61/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5323 - accuracy: 0.8913 - val_loss: 0.3752 -
val_accuracy: 0.9723
Epoch 62/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5351 - accuracy: 0.8900 - val_loss: 0.3741 -
val_accuracy: 0.9738
Epoch 63/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5243 - accuracy: 0.8943 - val_loss: 0.3587 -
val_accuracy: 0.9762
Epoch 64/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5221 - accuracy: 0.8922 - val_loss: 0.3804 -
val_accuracy: 0.9734
Epoch 65/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5277 - accuracy: 0.8923 - val_loss: 0.3890 -
val_accuracy: 0.9664
Epoch 66/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5274 - accuracy: 0.8944 - val_loss: 0.3875 -
val_accuracy: 0.9719
Epoch 67/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5251 - accuracy: 0.8928 - val_loss: 0.3803 -
val_accuracy: 0.9691
Epoch 68/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5124 - accuracy: 0.8932 - val_loss: 0.3698 -
val_accuracy: 0.9711
Epoch 69/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5160 - accuracy: 0.9013 - val_loss: 0.3865 -
val_accuracy: 0.9688
Epoch 70/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5049 - accuracy: 0.8995 - val_loss: 0.3528 -
val_accuracy: 0.9707
Epoch 71/100
60/60 [==============================] - 12s 199ms/step - loss: 0.4984 - accuracy: 0.9014 - val_loss: 0.3586 -
val_accuracy: 0.9715
Epoch 72/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5005 - accuracy: 0.9004 - val_loss: 0.3579 -
val_accuracy: 0.9770
Epoch 73/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5068 - accuracy: 0.9000 - val_loss: 0.3512 -
val_accuracy: 0.9750
Epoch 74/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5048 - accuracy: 0.9031 - val_loss: 0.3654 -
val_accuracy: 0.9742
Epoch 75/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5113 - accuracy: 0.9051 - val_loss: 0.3837 -
val_accuracy: 0.9680
Epoch 76/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5140 - accuracy: 0.9009 - val_loss: 0.3804 -
val_accuracy: 0.9734
Epoch 77/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5144 - accuracy: 0.9030 - val_loss: 0.3753 -
val_accuracy: 0.9762
Epoch 78/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5026 - accuracy: 0.9077 - val_loss: 0.3718 -
val_accuracy: 0.9754
Epoch 79/100
60/60 [==============================] - 12s 199ms/step - loss: 0.5048 - accuracy: 0.9047 - val_loss: 0.3669 -
val_accuracy: 0.9754
Epoch 80/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5042 - accuracy: 0.9044 - val_loss: 0.3641 -
val_accuracy: 0.9777
Epoch 81/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5069 - accuracy: 0.9051 - val_loss: 0.3708 -
val_accuracy: 0.9766
Epoch 82/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5007 - accuracy: 0.9073 - val_loss: 0.3610 -
val_accuracy: 0.9758
Epoch 83/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5081 - accuracy: 0.9033 - val_loss: 0.3628 -
val_accuracy: 0.9758
Epoch 84/100
60/60 [==============================] - 12s 198ms/step - loss: 0.5168 - accuracy: 0.9009 - val_loss: 0.3661 -
```

```
val_accuracy: 0.9789
Epoch 85/100
60/60 [==============================] - 12s 197ms/step - loss: 0.5085 - accuracy: 0.9066 - val_loss: 0.3739 -
val_accuracy: 0.9734
Epoch 86/100
60/60 [==============================] - 12s 197ms/step - loss: 0.5111 - accuracy: 0.9042 - val_loss: 0.3686 -
val_accuracy: 0.9762
Epoch 87/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4991 - accuracy: 0.9065 - val_loss: 0.3642 -
val_accuracy: 0.9785
Epoch 88/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4914 - accuracy: 0.9117 - val_loss: 0.3625 -
val_accuracy: 0.9789
Epoch 89/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4901 - accuracy: 0.9105 - val_loss: 0.3690 -
val_accuracy: 0.9715
Epoch 90/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4820 - accuracy: 0.9128 - val_loss: 0.3500 -
val_accuracy: 0.9758
Epoch 91/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4796 - accuracy: 0.9129 - val_loss: 0.3651 -
val_accuracy: 0.9688
Epoch 92/100
60/60 [==============================] - 12s 196ms/step - loss: 0.4847 - accuracy: 0.9083 - val_loss: 0.3463 -
val_accuracy: 0.9797
Epoch 93/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4844 - accuracy: 0.9100 - val_loss: 0.3559 -
val_accuracy: 0.9758
Epoch 94/100
60/60 [==============================] - 12s 197ms/step - loss: 0.4834 - accuracy: 0.9096 - val_loss: 0.3486 -
val_accuracy: 0.9777
Epoch 95/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4840 - accuracy: 0.9083 - val_loss: 0.3603 -
val_accuracy: 0.9750
Epoch 96/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4884 - accuracy: 0.9094 - val_loss: 0.3530 -
val_accuracy: 0.9758
Epoch 97/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4973 - accuracy: 0.9057 - val_loss: 0.3641 -
val_accuracy: 0.9746
Epoch 98/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4984 - accuracy: 0.9076 - val_loss: 0.3690 -
val_accuracy: 0.9742
Epoch 99/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4953 - accuracy: 0.9098 - val_loss: 0.3649 -
val_accuracy: 0.9785
Epoch 100/100
60/60 [==============================] - 12s 198ms/step - loss: 0.4837 - accuracy: 0.9147 - val_loss: 0.3605 -
val_accuracy: 0.9742
```

In [74]:
```
print('The training performace of this model is given below.\n\n'.title())

predicted_ = model.predict(X_train)
predicted_ = np.argmax(predicted_, axis = 1)

print('The accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_train),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_train, average = 'm
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_train, average = 'macr
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_train, average = 'macro'),

print('The testing performace of this model is given below.\n\n'.title())

predicted_ = model.predict(X_test)
predicted_ = np.argmax(predicted_, axis = 1)

print('\nThe accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_test),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_test, average = 'ma
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_test, average = 'macro
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_test, average = 'macro'),'
```

```
The Training Performace Of This Model Is Given Below.


240/240 [==============================] - 3s 10ms/step
The accuracy of this Neural Network is = 0.9998697916666667
The precision of this Neural Network is  = 0.9998713991769548
The reacll of this Neural Network is     = 0.9998693152117093
The f1_score of this Neural Network is   = 0.9998703235689268
The Testing Performace Of This Model Is Given Below.


80/80 [==============================] - 1s 10ms/step

The accuracy of this Neural Network is = 0.97421875
The precision of this Neural Network is  = 0.9739566995001777
The reacll of this Neural Network is     = 0.9739950805178585
The f1_score of this Neural Network is   = 0.9739750180379956
```

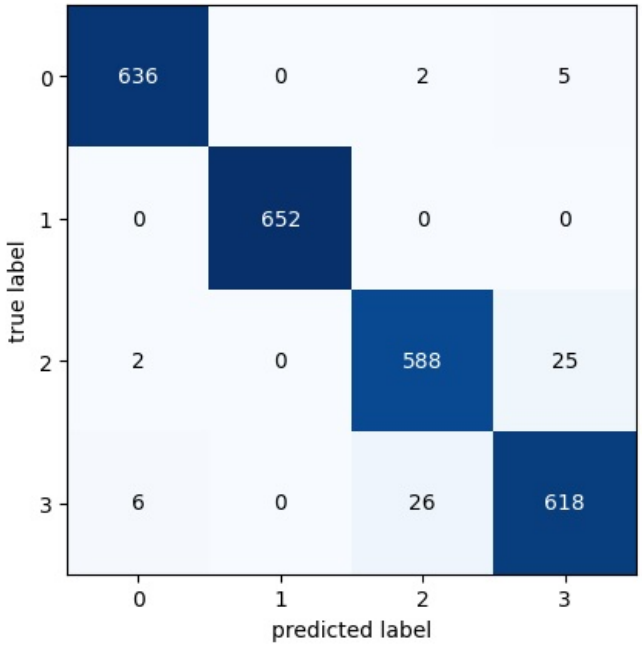
In [75]:
```
print('The classification report of this testing model is given below.\n'.capitalize())
```

```
print(classification_report(predicted_, y_test))
```

```
The classification report of this testing model is given below.

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       643
           1       1.00      1.00      1.00       652
           2       0.95      0.96      0.96       615
           3       0.95      0.95      0.95       650

    accuracy                           0.97      2560
   macro avg       0.97      0.97      0.97      2560
weighted avg       0.97      0.97      0.97      2560
```

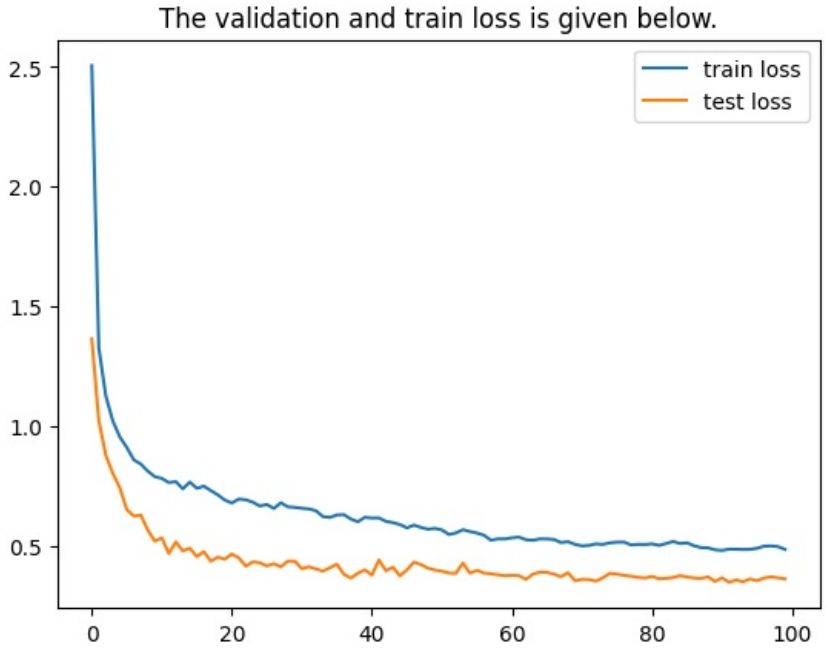
Plot the `Confusion` Matrix

In [76]:
```
#### Plot the confusion matrix ####
confusion_mat = confusion_matrix(predicted_, y_test)

fig, ax = plot_confusion_matrix(conf_mat = confusion_mat)

plt.show()
```

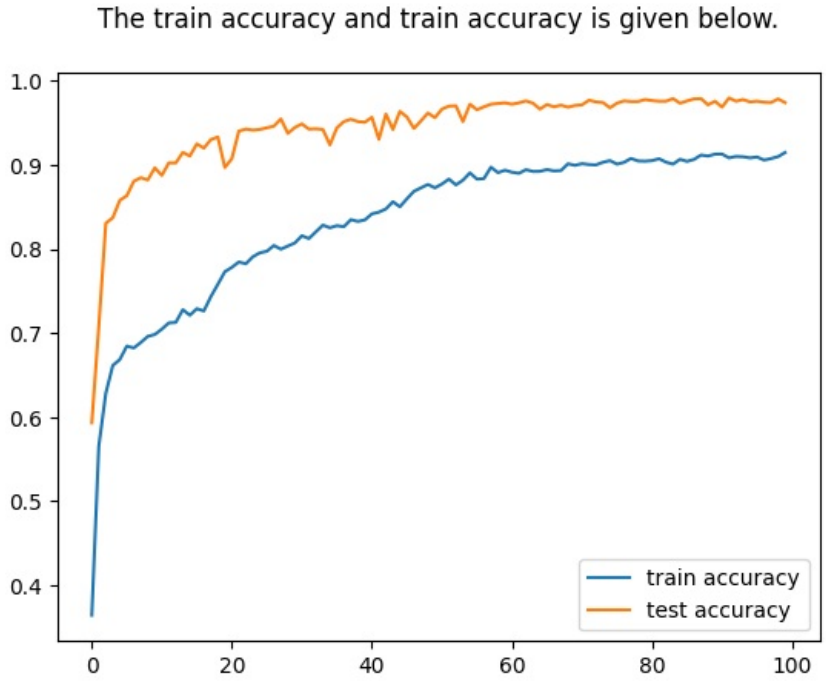


Plot `train and test` loss and `train and test` accuracy

In [77]:
```
##### Plot the validation loss and train loss #####
plt.title('The validation and train loss is given below.')
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'test loss')
plt.legend()
plt.show()

print('*'*120,'\n')

plt.title('The train accuracy and train accuracy is given below.\n')
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.plot(history.history['val_accuracy'], label = 'test accuracy')
plt.legend()
plt.show()
```



****



Check the performance for `Validation` dataset

In [78]:
```
##### Show the performance of this model #####
predicted = model.predict(X_validation_normalised)
predicted = np.argmax(predicted, axis = 1)

print('\nThe accuracy of this Neural Network is  = {} '.format(accuracy_score(predicted,\
                                                                 y_val),'\n'))
print('The precision of this Neural Network is = {} '.format(precision_score(predicted, y_val,\
                                                              average = 'macro'),'\n'))
print('The reacll of this Neural Network is    = {} '.format(recall_score(predicted, y_val,\
                                                           average = 'macro'),'\n'))
print('The f1_score of this Neural Network is  = {} '.format(f1_score(predicted, y_val,\
                                                         average = 'macro')))
```

```
40/40 [==============================] - 1s 17ms/step

The accuracy of this Neural Network is  = 0.9632525410476935
The precision of this Neural Network is = 0.972090420490822
The reacll of this Neural Network is    = 0.9698747867012347
The f1_score of this Neural Network is  = 0.9709176970355273
```

Save the model for the `web app` application

In [79]:
```
try:
  model.save('model.h5')
except Exception as e:
```

```
  print('The exception is {}'.format(e).title())
else:
  print('The Model has been saved successfully'.title())
```

```
The Model Has Been Saved Successfully
```

Do the `training` with respect to `Vanilla` architecture - plain architecture

In [ ]:
```
#### Define a function that is responsible for training ####
def train_model(train_data = None, optimizer = None):
  if len(train_data) == 0:
    raise listEmptyException('List is empty'.title())
  else:
    #### Create a sequential model ####
    model = Sequential()

    #### Create first Convolutional Layer with 32 kernels ####
    model.add(Conv2D(filters = 64, kernel_size = (3, 3),\
                     strides = (1, 1),\
                     padding = 'valid',\
                     activation = 'relu',\
                     kernel_initializer = 'he_normal',\
                     input_shape = train_data.shape[1:]))

    #### Use the MaxPooling Layer with Strides = 2, shape = (2, 2) ####
    model.add(MaxPool2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))

    #### Create second Convolutional Layer with 32 kernels ####
    model.add(Conv2D(filters = 32,\
                     kernel_size = (3, 3),\
                     strides = (1, 1),\
                     padding = 'valid',\
                     activation = 'relu' ,\
                     kernel_initializer = 'he_normal'))

    #### Use the MaxPooling Layer with Strides = 2, shape = (2, 2) ####
    model.add(MaxPool2D(pool_size = (2, 2),\
                        strides = (2, 2),\
                        padding = 'valid'))


    #### Create third and last Convolutional Layer with 16 filters ####
    model.add(Conv2D(filters = 16,\
                     kernel_size = (3, 3),\
                     strides = (1, 1),\
                     padding = 'valid',\
                     activation = 'relu',\
                     kernel_initializer = 'he_normal'))

    #### Use the MaxPooling Layer with shape (2, 2) and Strides = (2, 2) ####
    model.add(MaxPool2D(pool_size = (2, 2),\
                        strides = (2, 2),\
                        padding = 'valid'))

    #### Flatten the Convolutional Layer ####
    model.add(Flatten())

    #### Create first hidden layer with 256 neurons with L2 regularization ####
    model.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

    #### Use the Dropout Layer with the p value = 0.5 ####
    model.add(Dropout(0.6))

    #### Create second hidden layer with 128 neurons ####
    model.add(Dense(units = 64, activation = 'relu', kernel_initializer = 'he_normal', kernel_regularizer = L2(

    #### Use the Dropout Layer with the p value = 0.5 ####
    model.add(Dropout(0.6))

    #### Create an output layer with softmax ####
    model.add(Dense(units = 4, activation = 'softmax'))

    #### Compile the model ####
    model.compile(optimizer = optimizer, loss = SparseCategoricalCrossentropy(), metrics = ['accuracy'])

  return model
```

Plot the summary of this model

In [ ]:
```
try:
  model = train_model(train_data = X_train_normalised, optimizer = 'Adam')
except listEmptyException as e:
  print("The exception of this model is {}".format(e))
except Exception as e:
  print(e.with_traceback)
else:
```

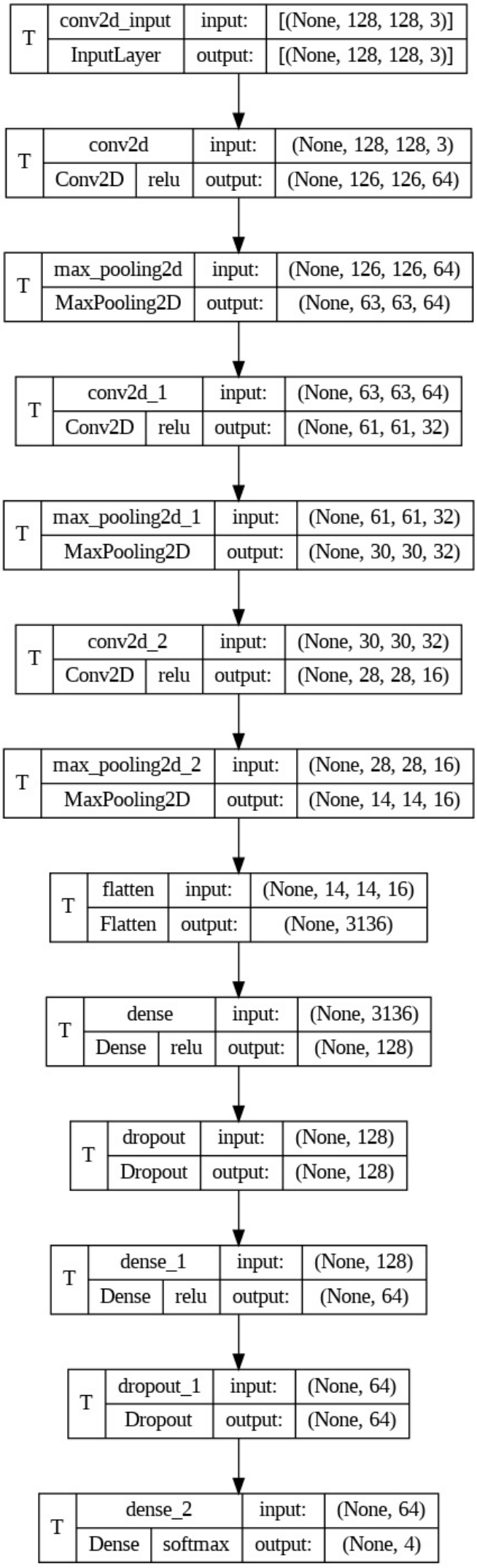
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 63, 63, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 16)       0
 2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 4)                 260

=================================================================
Total params: 434,932
Trainable params: 434,932
Non-trainable params: 0
_________________________________________________________________
```

Plot the `model` architecture

In [ ]:
```
plot_model(model = model,\
           show_shapes = True,\
           show_layer_names = True,\
           show_layer_activations = True,\
           show_trainable = True)
```

Out[ ]:


T
conv2d_input
InputLayer
input:
output:
[(None, 128, 128, 3)]
[(None, 128, 128, 3)]
T
conv2d
Conv2D
relu
input:
output:
(None, 128, 128, 3)
(None, 126, 126, 64)
T
max_pooling2d
MaxPooling2D
input:
output:
(None, 126, 126, 64)
(None, 63, 63, 64)
T
conv2d_1
Conv2D
relu
input:
output:
(None, 63, 63, 64)
(None, 61, 61, 32)
T
max_pooling2d_1
MaxPooling2D
input:
output:
(None, 61, 61, 32)
(None, 30, 30, 32)
T
conv2d_2
Conv2D
relu
input:
output:
(None, 30, 30, 32)
(None, 28, 28, 16)
T
max_pooling2d_2
MaxPooling2D
input:
output:
(None, 28, 28, 16)
(None, 14, 14, 16)
T
flatten
Flatten
input:
output:
(None, 14, 14, 16)
(None, 3136)
T
dense
Dense
relu
input:
output:
(None, 3136)
(None, 128)
T
dropout
Dropout
input:
output:
(None, 128)
(None, 128)
T
dense_1
Dense
relu
input:
output:
(None, 128)
(None, 64)
T
dropout_1
Dropout
input:
output:
(None, 64)
(None, 64)
T
dense_2
Dense
softmax
input:
output:
(None, 64)
(None, 4)

Do the `train` with respect to train and test

In [ ]:
```
def do_train(x = None, y = None, batch_size = None, epochs = None):
  #### Fit the model and check the performance with batch size = 64 ####
  history_ = model.fit(x = x,\
                       y = y,\
                       batch_size = batch_size,\
                       epochs = epochs,\
                       validation_data = (X_test, y_test),\
                       verbose = 1,\
                       class_weight = sklearn_weights)
  return history_
try:
  history = do_train(x = X_train, y = y_train, batch_size = 128, epochs = 100)
except Exception as e:
  print('The exception is {}'.format(e))
```

```
Epoch 1/100
60/60 [==============================] - 14s 92ms/step - loss: 2.3402 - accuracy: 0.3393 - val_loss: 1.8532 - v
al_accuracy: 0.6242
Epoch 2/100
60/60 [==============================] - 4s 72ms/step - loss: 1.5192 - accuracy: 0.5572 - val_loss: 1.1152 - va
l_accuracy: 0.6969
Epoch 3/100
60/60 [==============================] - 4s 71ms/step - loss: 1.1091 - accuracy: 0.6370 - val_loss: 0.8540 - va
l_accuracy: 0.7242
Epoch 4/100
60/60 [==============================] - 4s 71ms/step - loss: 0.8815 - accuracy: 0.6708 - val_loss: 0.6952 - va
l_accuracy: 0.7660
Epoch 5/100
60/60 [==============================] - 4s 72ms/step - loss: 0.7693 - accuracy: 0.6960 - val_loss: 0.6103 - va
l_accuracy: 0.7801
Epoch 6/100
60/60 [==============================] - 4s 71ms/step - loss: 0.6996 - accuracy: 0.7117 - val_loss: 0.5510 - va
l_accuracy: 0.8016
Epoch 7/100
60/60 [==============================] - 4s 72ms/step - loss: 0.6499 - accuracy: 0.7245 - val_loss: 0.5154 - va
l_accuracy: 0.8008
Epoch 8/100
60/60 [==============================] - 4s 72ms/step - loss: 0.5977 - accuracy: 0.7401 - val_loss: 0.4583 - va
l_accuracy: 0.8285
Epoch 9/100
60/60 [==============================] - 4s 72ms/step - loss: 0.5697 - accuracy: 0.7547 - val_loss: 0.4701 - va
l_accuracy: 0.8309
Epoch 10/100
60/60 [==============================] - 4s 71ms/step - loss: 0.5373 - accuracy: 0.7643 - val_loss: 0.4185 - va
l_accuracy: 0.8391
Epoch 11/100
60/60 [==============================] - 4s 72ms/step - loss: 0.5275 - accuracy: 0.7681 - val_loss: 0.4169 - va
l_accuracy: 0.8430
Epoch 12/100
60/60 [==============================] - 4s 71ms/step - loss: 0.5207 - accuracy: 0.7654 - val_loss: 0.3885 - va
l_accuracy: 0.8414
Epoch 13/100
60/60 [==============================] - 4s 74ms/step - loss: 0.5077 - accuracy: 0.7753 - val_loss: 0.3923 - va
l_accuracy: 0.8516
Epoch 14/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4848 - accuracy: 0.7898 - val_loss: 0.3612 - va
l_accuracy: 0.8645
Epoch 15/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4872 - accuracy: 0.7859 - val_loss: 0.3684 - va
l_accuracy: 0.8484
Epoch 16/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4774 - accuracy: 0.7888 - val_loss: 0.3801 - va
l_accuracy: 0.8617
Epoch 17/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4647 - accuracy: 0.7962 - val_loss: 0.3631 - va
l_accuracy: 0.8578
Epoch 18/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4623 - accuracy: 0.7977 - val_loss: 0.3413 - va
l_accuracy: 0.8742
Epoch 19/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4683 - accuracy: 0.7898 - val_loss: 0.3477 - va
l_accuracy: 0.8562
Epoch 20/100
60/60 [==============================] - 4s 70ms/step - loss: 0.4366 - accuracy: 0.8133 - val_loss: 0.3199 - va
l_accuracy: 0.8859
Epoch 21/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4240 - accuracy: 0.8217 - val_loss: 0.3197 - va
l_accuracy: 0.8836
Epoch 22/100
60/60 [==============================] - 4s 71ms/step - loss: 0.4209 - accuracy: 0.8189 - val_loss: 0.2889 - va
l_accuracy: 0.8867
Epoch 23/100
60/60 [==============================] - 4s 72ms/step - loss: 0.4331 - accuracy: 0.8120 - val_loss: 0.2976 - va
l_accuracy: 0.8813
Epoch 24/100
60/60 [==============================] - 4s 73ms/step - loss: 0.4223 - accuracy: 0.8243 - val_loss: 0.2876 - va
```

```
l_accuracy: 0.8914
Epoch 25/100
60/60 [==============================] - 4s 72ms/step - loss: 0.4069 - accuracy: 0.8289 - val_loss: 0.2888 - va
l_accuracy: 0.8863
Epoch 26/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3988 - accuracy: 0.8319 - val_loss: 0.2826 - va
l_accuracy: 0.8965
Epoch 27/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3909 - accuracy: 0.8432 - val_loss: 0.2766 - va
l_accuracy: 0.8879
Epoch 28/100
60/60 [==============================] - 4s 72ms/step - loss: 0.4088 - accuracy: 0.8337 - val_loss: 0.2950 - va
l_accuracy: 0.8977
Epoch 29/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3903 - accuracy: 0.8435 - val_loss: 0.2588 - va
l_accuracy: 0.9102
Epoch 30/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3766 - accuracy: 0.8452 - val_loss: 0.2648 - va
l_accuracy: 0.9074
Epoch 31/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3948 - accuracy: 0.8344 - val_loss: 0.2621 - va
l_accuracy: 0.9082
Epoch 32/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3863 - accuracy: 0.8418 - val_loss: 0.2554 - va
l_accuracy: 0.9137
Epoch 33/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3648 - accuracy: 0.8569 - val_loss: 0.2498 - va
l_accuracy: 0.9176
Epoch 34/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3512 - accuracy: 0.8578 - val_loss: 0.2432 - va
l_accuracy: 0.9187
Epoch 35/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3634 - accuracy: 0.8589 - val_loss: 0.2433 - va
l_accuracy: 0.9180
Epoch 36/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3533 - accuracy: 0.8613 - val_loss: 0.2407 - va
l_accuracy: 0.9207
Epoch 37/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3639 - accuracy: 0.8548 - val_loss: 0.2455 - va
l_accuracy: 0.9156
Epoch 38/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3501 - accuracy: 0.8615 - val_loss: 0.2203 - va
l_accuracy: 0.9258
Epoch 39/100
60/60 [==============================] - 4s 70ms/step - loss: 0.3532 - accuracy: 0.8625 - val_loss: 0.2311 - va
l_accuracy: 0.9203
Epoch 40/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3487 - accuracy: 0.8643 - val_loss: 0.2333 - va
l_accuracy: 0.9195
Epoch 41/100
60/60 [==============================] - 4s 70ms/step - loss: 0.3376 - accuracy: 0.8711 - val_loss: 0.2245 - va
l_accuracy: 0.9195
Epoch 42/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3478 - accuracy: 0.8672 - val_loss: 0.2136 - va
l_accuracy: 0.9309
Epoch 43/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3187 - accuracy: 0.8793 - val_loss: 0.2477 - va
l_accuracy: 0.9129
Epoch 44/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3283 - accuracy: 0.8751 - val_loss: 0.2129 - va
l_accuracy: 0.9270
Epoch 45/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3303 - accuracy: 0.8717 - val_loss: 0.2288 - va
l_accuracy: 0.9234
Epoch 46/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3297 - accuracy: 0.8741 - val_loss: 0.2227 - va
l_accuracy: 0.9312
Epoch 47/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3270 - accuracy: 0.8745 - val_loss: 0.2260 - va
l_accuracy: 0.9293
Epoch 48/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3236 - accuracy: 0.8746 - val_loss: 0.2178 - va
l_accuracy: 0.9324
Epoch 49/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3078 - accuracy: 0.8835 - val_loss: 0.2188 - va
l_accuracy: 0.9320
Epoch 50/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3318 - accuracy: 0.8746 - val_loss: 0.2266 - va
l_accuracy: 0.9180
Epoch 51/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3213 - accuracy: 0.8780 - val_loss: 0.1919 - va
l_accuracy: 0.9352
Epoch 52/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3211 - accuracy: 0.8773 - val_loss: 0.2070 - va
l_accuracy: 0.9309
Epoch 53/100
60/60 [==============================] - 4s 71ms/step - loss: 0.3163 - accuracy: 0.8820 - val_loss: 0.2024 - va
l_accuracy: 0.9383
Epoch 54/100
```

```
60/60 [==============================] - 4s 71ms/step - loss: 0.3021 - accuracy: 0.8904 - val_loss: 0.1896 - va
l_accuracy: 0.9379
Epoch 55/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2967 - accuracy: 0.8885 - val_loss: 0.1835 - va
l_accuracy: 0.9438
Epoch 56/100
60/60 [==============================] - 4s 70ms/step - loss: 0.3137 - accuracy: 0.8852 - val_loss: 0.2037 - va
l_accuracy: 0.9344
Epoch 57/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3130 - accuracy: 0.8833 - val_loss: 0.1815 - va
l_accuracy: 0.9410
Epoch 58/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2944 - accuracy: 0.8901 - val_loss: 0.1816 - va
l_accuracy: 0.9410
Epoch 59/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2995 - accuracy: 0.8882 - val_loss: 0.1860 - va
l_accuracy: 0.9441
Epoch 60/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2862 - accuracy: 0.8940 - val_loss: 0.1833 - va
l_accuracy: 0.9477
Epoch 61/100
60/60 [==============================] - 4s 72ms/step - loss: 0.3024 - accuracy: 0.8883 - val_loss: 0.1880 - va
l_accuracy: 0.9422
Epoch 62/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2944 - accuracy: 0.8904 - val_loss: 0.1903 - va
l_accuracy: 0.9445
Epoch 63/100
60/60 [==============================] - 4s 75ms/step - loss: 0.3028 - accuracy: 0.8885 - val_loss: 0.2074 - va
l_accuracy: 0.9250
Epoch 64/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2913 - accuracy: 0.8919 - val_loss: 0.1834 - va
l_accuracy: 0.9504
Epoch 65/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2832 - accuracy: 0.8941 - val_loss: 0.1881 - va
l_accuracy: 0.9438
Epoch 66/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2888 - accuracy: 0.8958 - val_loss: 0.1757 - va
l_accuracy: 0.9473
Epoch 67/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2891 - accuracy: 0.8943 - val_loss: 0.1812 - va
l_accuracy: 0.9484
Epoch 68/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2786 - accuracy: 0.8975 - val_loss: 0.1848 - va
l_accuracy: 0.9469
Epoch 69/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2813 - accuracy: 0.8970 - val_loss: 0.1867 - va
l_accuracy: 0.9445
Epoch 70/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2874 - accuracy: 0.8919 - val_loss: 0.1841 - va
l_accuracy: 0.9480
Epoch 71/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2804 - accuracy: 0.8952 - val_loss: 0.1809 - va
l_accuracy: 0.9465
Epoch 72/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2832 - accuracy: 0.8957 - val_loss: 0.1760 - va
l_accuracy: 0.9508
Epoch 73/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2829 - accuracy: 0.8928 - val_loss: 0.1937 - va
l_accuracy: 0.9492
Epoch 74/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2767 - accuracy: 0.8990 - val_loss: 0.2194 - va
l_accuracy: 0.9445
Epoch 75/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2893 - accuracy: 0.8935 - val_loss: 0.1919 - va
l_accuracy: 0.9410
Epoch 76/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2766 - accuracy: 0.9017 - val_loss: 0.1808 - va
l_accuracy: 0.9480
Epoch 77/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2747 - accuracy: 0.8999 - val_loss: 0.1886 - va
l_accuracy: 0.9469
Epoch 78/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2883 - accuracy: 0.8948 - val_loss: 0.2038 - va
l_accuracy: 0.9367
Epoch 79/100
60/60 [==============================] - 4s 70ms/step - loss: 0.2787 - accuracy: 0.8987 - val_loss: 0.1856 - va
l_accuracy: 0.9434
Epoch 80/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2780 - accuracy: 0.8966 - val_loss: 0.1751 - va
l_accuracy: 0.9473
Epoch 81/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2579 - accuracy: 0.9073 - val_loss: 0.1743 - va
l_accuracy: 0.9523
Epoch 82/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2775 - accuracy: 0.8951 - val_loss: 0.1732 - va
l_accuracy: 0.9535
Epoch 83/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2808 - accuracy: 0.8951 - val_loss: 0.1812 - va
l_accuracy: 0.9512
```

```
Epoch 84/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2700 - accuracy: 0.8988 - val_loss: 0.1914 - va
l_accuracy: 0.9484
Epoch 85/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2697 - accuracy: 0.9020 - val_loss: 0.1700 - va
l_accuracy: 0.9516
Epoch 86/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2603 - accuracy: 0.9039 - val_loss: 0.1974 - va
l_accuracy: 0.9469
Epoch 87/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2712 - accuracy: 0.8980 - val_loss: 0.1760 - va
l_accuracy: 0.9539
Epoch 88/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2714 - accuracy: 0.9048 - val_loss: 0.1760 - va
l_accuracy: 0.9531
Epoch 89/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2670 - accuracy: 0.9005 - val_loss: 0.1961 - va
l_accuracy: 0.9488
Epoch 90/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2687 - accuracy: 0.9030 - val_loss: 0.1836 - va
l_accuracy: 0.9508
Epoch 91/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2610 - accuracy: 0.9052 - val_loss: 0.1801 - va
l_accuracy: 0.9535
Epoch 92/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2557 - accuracy: 0.9043 - val_loss: 0.1918 - va
l_accuracy: 0.9480
Epoch 93/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2546 - accuracy: 0.9091 - val_loss: 0.1908 - va
l_accuracy: 0.9570
Epoch 94/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2665 - accuracy: 0.9025 - val_loss: 0.1920 - va
l_accuracy: 0.9547
Epoch 95/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2618 - accuracy: 0.9036 - val_loss: 0.1934 - va
l_accuracy: 0.9563
Epoch 96/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2584 - accuracy: 0.9046 - val_loss: 0.1775 - va
l_accuracy: 0.9531
Epoch 97/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2643 - accuracy: 0.9049 - val_loss: 0.2274 - va
l_accuracy: 0.9457
Epoch 98/100
60/60 [==============================] - 4s 72ms/step - loss: 0.2509 - accuracy: 0.9100 - val_loss: 0.1670 - va
l_accuracy: 0.9570
Epoch 99/100
60/60 [==============================] - 4s 71ms/step - loss: 0.2481 - accuracy: 0.9096 - val_loss: 0.1692 - va
l_accuracy: 0.9531
Epoch 100/100
60/60 [==============================] - 4s 70ms/step - loss: 0.2596 - accuracy: 0.9072 - val_loss: 0.1710 - va
l_accuracy: 0.9535
```

Show the performance of this model

In [ ]:

```
print('The training performace of this model is given below.\n\n'.title())

predicted_ = model.predict(X_train)
predicted_ = np.argmax(predicted_, axis = 1)

print('The accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_train),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_train, average = 'm
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_train, average = 'macr
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_train, average = 'macro'),

print('The testing performace of this model is given below.\n\n'.title())

predicted_ = model.predict(X_test)
predicted_ = np.argmax(predicted_, axis = 1)

print('\nThe accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_test),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_test, average = 'ma
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_test, average = 'macro
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_test, average = 'macro'),'
```

```
The Training Performace Of This Model Is Given Below.


240/240 [==============================] - 2s 6ms/step
The accuracy of this Neural Network is = 0.99375
The precision of this Neural Network is  = 0.9937613323337635
The reacll of this Neural Network is     = 0.9938361030297886
The f1_score of this Neural Network is   = 0.9937478357696101
The Testing Performace Of This Model Is Given Below.


80/80 [==============================] - 0s 6ms/step

The accuracy of this Neural Network is = 0.953515625
The precision of this Neural Network is  = 0.9537187162318618
The reacll of this Neural Network is     = 0.9555001586519056
The f1_score of this Neural Network is   = 0.9542903689395154
```

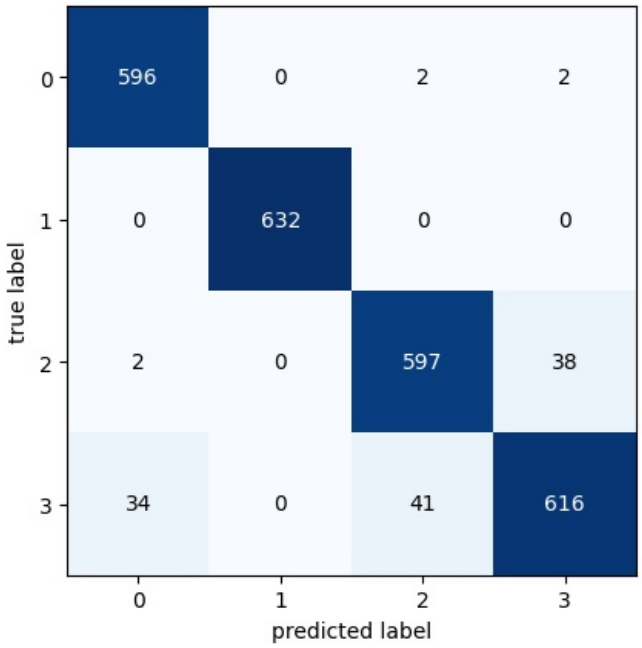
Show the `Classification` report to this model

In [ ]:
```
print('The classification report of this testing model is given below.\n'.capitalize())
print(classification_report(predicted_, y_test))
```

```
The classification report of this testing model is given below.

              precision    recall  f1-score   support

           0       0.94      0.99      0.97       600
           1       1.00      1.00      1.00       632
           2       0.93      0.94      0.94       637
           3       0.94      0.89      0.91       691

    accuracy                           0.95      2560
   macro avg       0.95      0.96      0.95      2560
weighted avg       0.95      0.95      0.95      2560
```

Plot the `Confusion` Matrix

In [ ]:
```
#### Plot the confusion matrix ####
confusion_mat = confusion_matrix(predicted_, y_test)

fig, ax = plot_confusion_matrix(conf_mat = confusion_mat)

plt.show()
```

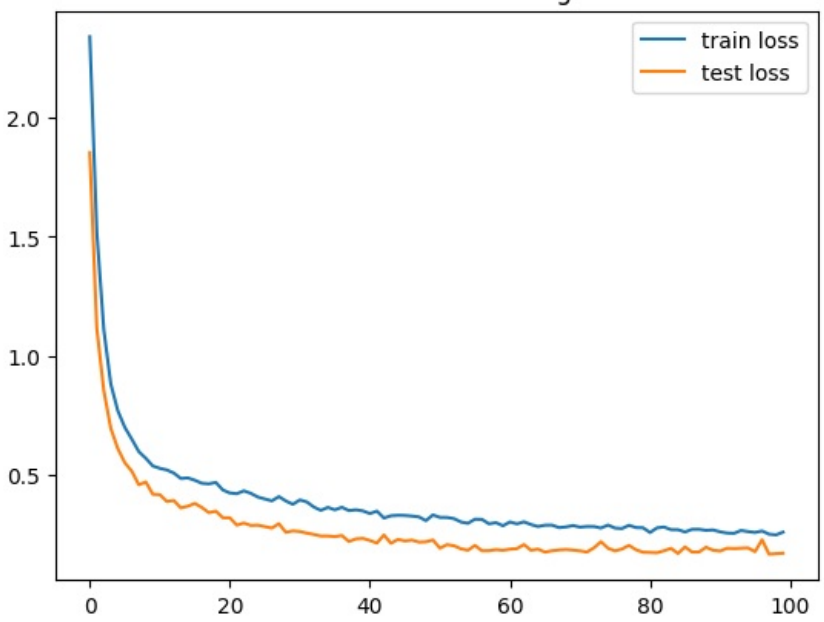


Plot `train and test` loss and `train and test` accuracy

In [ ]:
```
##### Plot the validation loss and train loss #####
plt.title('The validation and train loss is given below.')
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'test loss')
plt.legend()
plt.show()

print('*'*120,'\n')

plt.title('The train accuracy and train accuracy is given below.\n')
```

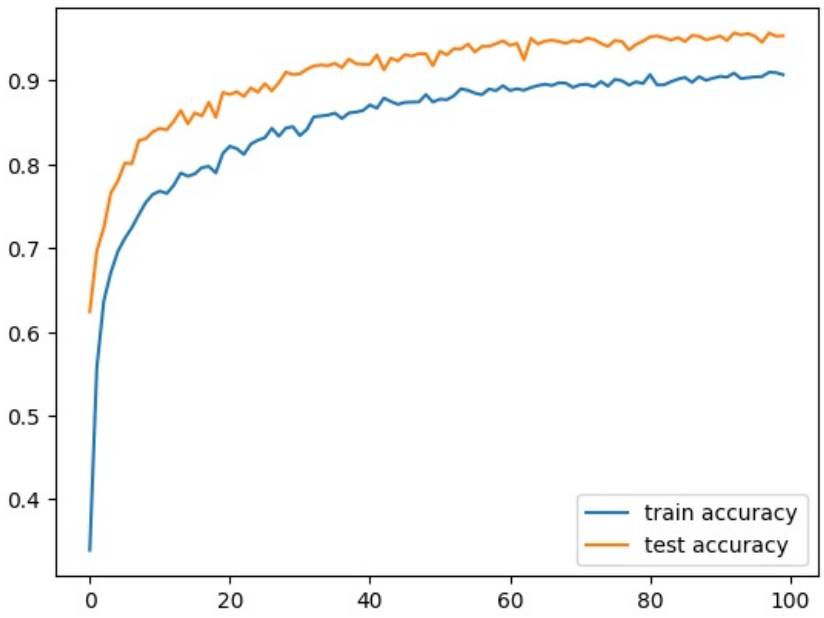
```
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.plot(history.history['val_accuracy'], label = 'test accuracy')
plt.legend()
plt.show()
```

The validation and train loss is given below.



****

The train accuracy and train accuracy is given below.



Save the `MODEL` for further work

In [ ]:
```
try:
  model.save('model.h5')
except Exception as e:
  print(e.with_traceback())
else:
  print('Model successfully saved'.title())
```

```
Model Successfully Saved
```

`Load` the model and show the performance of `Evaluation` data

In [ ]:
```
from tensorflow import keras
model = keras.models.load_model('/content/model.h5')
```

Plot the model summary

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 63, 63, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 16)       0
 2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 4)                 260

=================================================================
Total params: 434,932
Trainable params: 434,932
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
##### Show the performance of this model #####
predicted = model.predict(X_validation_normalised)
predicted = np.argmax(predicted, axis = 1)

print('\nThe accuracy of this Neural Network is  = {} '.format(accuracy_score(predicted,\
                                                                 y_val),'\n'))
print('The precision of this Neural Network is = {} '.format(precision_score(predicted, y_val,\
                                                               average = 'macro'),'\n'))
print('The reacll of this Neural Network is    = {} '.format(recall_score(predicted, y_val,\
                                                            average = 'macro'),'\n'))
print('The f1_score of this Neural Network is  = {} '.format(f1_score(predicted, y_val,\
                                                          average = 'macro')))
```

```
40/40 [==============================] - 1s 10ms/step

The accuracy of this Neural Network is  = 0.90852228303362
The precision of this Neural Network is = 0.9155766784716679
The reacll of this Neural Network is    = 0.9397671329753067
The f1_score of this Neural Network is  = 0.9260239502434559
```

Use `KFold` - 3 Cross validation to prevent the overfitting problem and check the model performance

In [ ]:
```
KFold_ = KFold(n_splits = 3, random_state = 42, shuffle = True)

accuracy, precision, recall, f1, count = [], [], [], [], 1

for train_index, test_index in KFold_.split(X_train_normalised):

  print('# of Cross Validation is {} is running'.title().format(count),'\n\n')
  X_train, X_test = X_train_normalised[train_index], X_train_normalised[test_index]
  y_train_, y_test_ = y_train[train_index], y_train[test_index]

  model = Sequential()

  #### Create first Convolutional Layer with 32 kernels ####
  model.add(Conv2D(filters = 64, kernel_size = (3, 3),\
                   strides = (1, 1),\
                   padding = 'valid',\
                   activation = 'relu',\
                   kernel_initializer = 'he_normal',\
                   input_shape = X_train.shape[1:]))

  #### Use the MaxPooling Layer with Strides = 2, shape = (2, 2) ####
  model.add(MaxPool2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))

  #### Create second Convolutional Layer with 32 kernels ####
  model.add(Conv2D(filters = 32,\
                   kernel_size = (3, 3),\
                   strides = (1, 1),\
                   padding = 'valid',\
                   activation = 'relu' ,\
```

```
                    kernel_initializer = 'he_normal'))

  #### Use the MaxPooling Layer with Strides = 2, shape = (2, 2) ####
  model.add(MaxPool2D(pool_size = (2, 2),\
                      strides = (2, 2),\
                      padding = 'valid'))


  #### Create third and last Convolutional Layer with 16 filters ####
  model.add(Conv2D(filters = 16,\
                   kernel_size = (3, 3),\
                   strides = (1, 1),\
                   padding = 'valid',\
                   activation = 'relu',\
                   kernel_initializer = 'he_normal'))

  #### Use the MaxPooling Layer with shape (2, 2) and Strides = (2, 2) ####
  model.add(MaxPool2D(pool_size = (2, 2),\
                      strides = (2, 2),\
                      padding = 'valid'))

  #### Flatten the Convolutional Layer ####
  model.add(Flatten())

  #### Create first hidden layer with 256 neurons with L2 regularization ####
  model.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

  #### Use the Dropout Layer with the p value = 0.5 ####
  model.add(Dropout(0.6))

  #### Create second hidden layer with 128 neurons ####
  model.add(Dense(units = 64, activation = 'relu', kernel_initializer = 'he_normal', kernel_regularizer = L2()))

  #### Use the Dropout Layer with the p value = 0.5 ####
  model.add(Dropout(0.6))

  #### Create an output layer with softmax ####
  model.add(Dense(units = 4, activation = 'softmax'))

  #### Compile the model ####
  model.compile(optimizer = 'Adam', loss = SparseCategoricalCrossentropy(), metrics = ['accuracy'])

  history = model.fit(x = X_train, y = y_train_, epochs = 100, batch_size = 128, validation_data = (X_test, y_t

  predicted_ = model.predict(X_test)
  predicted_ = np.argmax(predicted_, axis = 1)

  print('The accuracy of this Neural Network is  = {} '.format(accuracy_score(predicted_, y_test_),'\n'))
  print('The precision of this Neural Network is = {} '.format(precision_score(predicted_, y_test_, average = '
  print('The reacll of this Neural Network is    = {} '.format(recall_score(predicted_, y_test_, average = 'mac
  print('The f1_score of this Neural Network is  = {} '.format(f1_score(predicted_, y_test_, average = 'macro')

  accuracy.append(accuracy_score(predicted_, y_test_))
  precision.append(precision_score(predicted_, y_test_, average = 'macro'))
  recall.append(recall_score(predicted_, y_test_, average = 'macro'))
  f1.append(f1_score(predicted_, y_test_, average = 'macro'))

  count = count + 1
```

```
# Of Cross Validation Is 1 Is Running


Epoch 1/100
54/54 [==============================] - 18s 115ms/step - loss: 2.3353 - accuracy: 0.2901 - val_loss: 1.9198 -
val_accuracy: 0.6388
Epoch 2/100
54/54 [==============================] - 4s 73ms/step - loss: 1.4910 - accuracy: 0.5617 - val_loss: 0.9888 - va
l_accuracy: 0.6980
Epoch 3/100
54/54 [==============================] - 4s 74ms/step - loss: 0.9847 - accuracy: 0.6824 - val_loss: 0.7505 - va
l_accuracy: 0.7493
Epoch 4/100
54/54 [==============================] - 4s 74ms/step - loss: 0.7924 - accuracy: 0.7266 - val_loss: 0.6288 - va
l_accuracy: 0.7709
Epoch 5/100
54/54 [==============================] - 4s 73ms/step - loss: 0.6586 - accuracy: 0.7551 - val_loss: 0.5017 - va
l_accuracy: 0.8240
Epoch 6/100
54/54 [==============================] - 4s 73ms/step - loss: 0.5631 - accuracy: 0.7820 - val_loss: 0.4350 - va
l_accuracy: 0.8266
Epoch 7/100
54/54 [==============================] - 4s 72ms/step - loss: 0.5212 - accuracy: 0.7958 - val_loss: 0.4191 - va
l_accuracy: 0.8310
Epoch 8/100
54/54 [==============================] - 4s 73ms/step - loss: 0.4676 - accuracy: 0.8154 - val_loss: 0.3878 - va
l_accuracy: 0.8465
Epoch 9/100
54/54 [==============================] - 4s 78ms/step - loss: 0.4190 - accuracy: 0.8309 - val_loss: 0.3582 - va
l_accuracy: 0.8576
```

```
Epoch 10/100
54/54 [==============================] - 4s 72ms/step - loss: 0.3976 - accuracy: 0.8441 - val_loss: 0.3437 - va
l_accuracy: 0.8670
Epoch 11/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3574 - accuracy: 0.8572 - val_loss: 0.3069 - va
l_accuracy: 0.8805
Epoch 12/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3351 - accuracy: 0.8715 - val_loss: 0.2908 - va
l_accuracy: 0.8902
Epoch 13/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3217 - accuracy: 0.8721 - val_loss: 0.2728 - va
l_accuracy: 0.8957
Epoch 14/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2866 - accuracy: 0.8907 - val_loss: 0.2691 - va
l_accuracy: 0.8925
Epoch 15/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2745 - accuracy: 0.8947 - val_loss: 0.2689 - va
l_accuracy: 0.9001
Epoch 16/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2649 - accuracy: 0.9013 - val_loss: 0.2319 - va
l_accuracy: 0.9189
Epoch 17/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2584 - accuracy: 0.9045 - val_loss: 0.2334 - va
l_accuracy: 0.9151
Epoch 18/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2489 - accuracy: 0.9048 - val_loss: 0.2058 - va
l_accuracy: 0.9288
Epoch 19/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2161 - accuracy: 0.9224 - val_loss: 0.2104 - va
l_accuracy: 0.9279
Epoch 20/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2223 - accuracy: 0.9159 - val_loss: 0.1910 - va
l_accuracy: 0.9329
Epoch 21/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2022 - accuracy: 0.9219 - val_loss: 0.1881 - va
l_accuracy: 0.9376
Epoch 22/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1946 - accuracy: 0.9284 - val_loss: 0.1850 - va
l_accuracy: 0.9335
Epoch 23/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1740 - accuracy: 0.9414 - val_loss: 0.1713 - va
l_accuracy: 0.9420
Epoch 24/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1710 - accuracy: 0.9395 - val_loss: 0.1879 - va
l_accuracy: 0.9408
Epoch 25/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1695 - accuracy: 0.9408 - val_loss: 0.1556 - va
l_accuracy: 0.9484
Epoch 26/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1516 - accuracy: 0.9486 - val_loss: 0.1775 - va
l_accuracy: 0.9402
Epoch 27/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1476 - accuracy: 0.9511 - val_loss: 0.1455 - va
l_accuracy: 0.9520
Epoch 28/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1312 - accuracy: 0.9533 - val_loss: 0.1563 - va
l_accuracy: 0.9449
Epoch 29/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1404 - accuracy: 0.9505 - val_loss: 0.1526 - va
l_accuracy: 0.9508
Epoch 30/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1363 - accuracy: 0.9521 - val_loss: 0.1430 - va
l_accuracy: 0.9540
Epoch 31/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1105 - accuracy: 0.9645 - val_loss: 0.1750 - va
l_accuracy: 0.9435
Epoch 32/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1221 - accuracy: 0.9563 - val_loss: 0.1492 - va
l_accuracy: 0.9523
Epoch 33/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1332 - accuracy: 0.9558 - val_loss: 0.1374 - va
l_accuracy: 0.9537
Epoch 34/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1146 - accuracy: 0.9618 - val_loss: 0.1365 - va
l_accuracy: 0.9584
Epoch 35/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1228 - accuracy: 0.9572 - val_loss: 0.1369 - va
l_accuracy: 0.9537
Epoch 36/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1058 - accuracy: 0.9688 - val_loss: 0.1342 - va
l_accuracy: 0.9610
Epoch 37/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1024 - accuracy: 0.9681 - val_loss: 0.1327 - va
l_accuracy: 0.9634
Epoch 38/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0990 - accuracy: 0.9678 - val_loss: 0.1416 - va
l_accuracy: 0.9590
Epoch 39/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1157 - accuracy: 0.9607 - val_loss: 0.1491 - va
```

```
l_accuracy: 0.9543
Epoch 40/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1082 - accuracy: 0.9653 - val_loss: 0.1678 - va
l_accuracy: 0.9455
Epoch 41/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0983 - accuracy: 0.9653 - val_loss: 0.1266 - va
l_accuracy: 0.9649
Epoch 42/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0970 - accuracy: 0.9719 - val_loss: 0.1414 - va
l_accuracy: 0.9596
Epoch 43/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0916 - accuracy: 0.9700 - val_loss: 0.1357 - va
l_accuracy: 0.9640
Epoch 44/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0928 - accuracy: 0.9688 - val_loss: 0.1285 - va
l_accuracy: 0.9646
Epoch 45/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0905 - accuracy: 0.9720 - val_loss: 0.1362 - va
l_accuracy: 0.9578
Epoch 46/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0879 - accuracy: 0.9713 - val_loss: 0.1251 - va
l_accuracy: 0.9637
Epoch 47/100
54/54 [==============================] - 4s 75ms/step - loss: 0.0852 - accuracy: 0.9719 - val_loss: 0.1288 - va
l_accuracy: 0.9640
Epoch 48/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0862 - accuracy: 0.9742 - val_loss: 0.1473 - va
l_accuracy: 0.9613
Epoch 49/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0864 - accuracy: 0.9717 - val_loss: 0.1318 - va
l_accuracy: 0.9643
Epoch 50/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0856 - accuracy: 0.9736 - val_loss: 0.1285 - va
l_accuracy: 0.9657
Epoch 51/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0854 - accuracy: 0.9700 - val_loss: 0.1257 - va
l_accuracy: 0.9690
Epoch 52/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0820 - accuracy: 0.9747 - val_loss: 0.1717 - va
l_accuracy: 0.9555
Epoch 53/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0816 - accuracy: 0.9766 - val_loss: 0.1325 - va
l_accuracy: 0.9605
Epoch 54/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0783 - accuracy: 0.9760 - val_loss: 0.1157 - va
l_accuracy: 0.9684
Epoch 55/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0755 - accuracy: 0.9755 - val_loss: 0.1235 - va
l_accuracy: 0.9692
Epoch 56/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0673 - accuracy: 0.9777 - val_loss: 0.1265 - va
l_accuracy: 0.9681
Epoch 57/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0796 - accuracy: 0.9757 - val_loss: 0.1404 - va
l_accuracy: 0.9616
Epoch 58/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0800 - accuracy: 0.9754 - val_loss: 0.1197 - va
l_accuracy: 0.9698
Epoch 59/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0730 - accuracy: 0.9767 - val_loss: 0.1506 - va
l_accuracy: 0.9607
Epoch 60/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0750 - accuracy: 0.9761 - val_loss: 0.1084 - va
l_accuracy: 0.9707
Epoch 61/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0797 - accuracy: 0.9722 - val_loss: 0.1230 - va
l_accuracy: 0.9654
Epoch 62/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0758 - accuracy: 0.9776 - val_loss: 0.1305 - va
l_accuracy: 0.9657
Epoch 63/100
54/54 [==============================] - 4s 75ms/step - loss: 0.0647 - accuracy: 0.9802 - val_loss: 0.1297 - va
l_accuracy: 0.9657
Epoch 64/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0706 - accuracy: 0.9785 - val_loss: 0.1363 - va
l_accuracy: 0.9607
Epoch 65/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0895 - accuracy: 0.9694 - val_loss: 0.1394 - va
l_accuracy: 0.9649
Epoch 66/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0694 - accuracy: 0.9783 - val_loss: 0.1128 - va
l_accuracy: 0.9698
Epoch 67/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0621 - accuracy: 0.9823 - val_loss: 0.1270 - va
l_accuracy: 0.9646
Epoch 68/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0622 - accuracy: 0.9824 - val_loss: 0.1334 - va
l_accuracy: 0.9663
Epoch 69/100
```

```
54/54 [==============================] - 4s 74ms/step - loss: 0.0573 - accuracy: 0.9836 - val_loss: 0.1189 - va
l_accuracy: 0.9695
Epoch 70/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0686 - accuracy: 0.9805 - val_loss: 0.1219 - va
l_accuracy: 0.9678
Epoch 71/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0610 - accuracy: 0.9801 - val_loss: 0.1184 - va
l_accuracy: 0.9698
Epoch 72/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0636 - accuracy: 0.9814 - val_loss: 0.1397 - va
l_accuracy: 0.9628
Epoch 73/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0632 - accuracy: 0.9807 - val_loss: 0.1269 - va
l_accuracy: 0.9687
Epoch 74/100
54/54 [==============================] - 4s 75ms/step - loss: 0.0650 - accuracy: 0.9812 - val_loss: 0.1193 - va
l_accuracy: 0.9687
Epoch 75/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0701 - accuracy: 0.9777 - val_loss: 0.1286 - va
l_accuracy: 0.9725
Epoch 76/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0563 - accuracy: 0.9815 - val_loss: 0.1030 - va
l_accuracy: 0.9716
Epoch 77/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0546 - accuracy: 0.9848 - val_loss: 0.1294 - va
l_accuracy: 0.9713
Epoch 78/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0598 - accuracy: 0.9821 - val_loss: 0.1207 - va
l_accuracy: 0.9713
Epoch 79/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0537 - accuracy: 0.9836 - val_loss: 0.1319 - va
l_accuracy: 0.9722
Epoch 80/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0585 - accuracy: 0.9808 - val_loss: 0.1471 - va
l_accuracy: 0.9643
Epoch 81/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0667 - accuracy: 0.9812 - val_loss: 0.1359 - va
l_accuracy: 0.9681
Epoch 82/100
54/54 [==============================] - 4s 75ms/step - loss: 0.0650 - accuracy: 0.9811 - val_loss: 0.1264 - va
l_accuracy: 0.9701
Epoch 83/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0616 - accuracy: 0.9820 - val_loss: 0.1294 - va
l_accuracy: 0.9707
Epoch 84/100
54/54 [==============================] - 4s 72ms/step - loss: 0.0666 - accuracy: 0.9805 - val_loss: 0.1419 - va
l_accuracy: 0.9643
Epoch 85/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0545 - accuracy: 0.9862 - val_loss: 0.1184 - va
l_accuracy: 0.9739
Epoch 86/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0503 - accuracy: 0.9848 - val_loss: 0.1557 - va
l_accuracy: 0.9669
Epoch 87/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0557 - accuracy: 0.9854 - val_loss: 0.1391 - va
l_accuracy: 0.9722
Epoch 88/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0470 - accuracy: 0.9865 - val_loss: 0.1281 - va
l_accuracy: 0.9704
Epoch 89/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0476 - accuracy: 0.9865 - val_loss: 0.1218 - va
l_accuracy: 0.9733
Epoch 90/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0568 - accuracy: 0.9834 - val_loss: 0.1422 - va
l_accuracy: 0.9669
Epoch 91/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0554 - accuracy: 0.9826 - val_loss: 0.1380 - va
l_accuracy: 0.9713
Epoch 92/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0585 - accuracy: 0.9836 - val_loss: 0.1314 - va
l_accuracy: 0.9725
Epoch 93/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0498 - accuracy: 0.9867 - val_loss: 0.1487 - va
l_accuracy: 0.9695
Epoch 94/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0509 - accuracy: 0.9867 - val_loss: 0.1257 - va
l_accuracy: 0.9725
Epoch 95/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0508 - accuracy: 0.9864 - val_loss: 0.1199 - va
l_accuracy: 0.9736
Epoch 96/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0514 - accuracy: 0.9862 - val_loss: 0.1073 - va
l_accuracy: 0.9725
Epoch 97/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0501 - accuracy: 0.9854 - val_loss: 0.1298 - va
l_accuracy: 0.9710
Epoch 98/100
54/54 [==============================] - 4s 74ms/step - loss: 0.0462 - accuracy: 0.9868 - val_loss: 0.1105 - va
l_accuracy: 0.9751
```

```
Epoch 99/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0463 - accuracy: 0.9855 - val_loss: 0.1461 - va
l_accuracy: 0.9701
Epoch 100/100
54/54 [==============================] - 4s 73ms/step - loss: 0.0552 - accuracy: 0.9858 - val_loss: 0.1477 - va
l_accuracy: 0.9687
107/107 [==============================] - 1s 6ms/step
The accuracy of this Neural Network is  = 0.9686584651435266
The precision of this Neural Network is = 0.9685304061089961
The reacll of this Neural Network is    = 0.9686908948781064
The f1_score of this Neural Network is  = 0.9685101743962414
# Of Cross Validation Is 2 Is Running


Epoch 1/100
54/54 [==============================] - 8s 117ms/step - loss: 2.3776 - accuracy: 0.3310 - val_loss: 1.7942 - v
al_accuracy: 0.6326
Epoch 2/100
54/54 [==============================] - 4s 74ms/step - loss: 1.5414 - accuracy: 0.5551 - val_loss: 1.1324 - va
l_accuracy: 0.6944
Epoch 3/100
54/54 [==============================] - 4s 73ms/step - loss: 1.1134 - accuracy: 0.6407 - val_loss: 0.8861 - va
l_accuracy: 0.7114
Epoch 4/100
54/54 [==============================] - 4s 73ms/step - loss: 0.8861 - accuracy: 0.6786 - val_loss: 0.6853 - va
l_accuracy: 0.7568
Epoch 5/100
54/54 [==============================] - 4s 74ms/step - loss: 0.7463 - accuracy: 0.7213 - val_loss: 0.5846 - va
l_accuracy: 0.7917
Epoch 6/100
54/54 [==============================] - 4s 73ms/step - loss: 0.6517 - accuracy: 0.7451 - val_loss: 0.5484 - va
l_accuracy: 0.7978
Epoch 7/100
54/54 [==============================] - 4s 73ms/step - loss: 0.5936 - accuracy: 0.7615 - val_loss: 0.4631 - va
l_accuracy: 0.8066
Epoch 8/100
54/54 [==============================] - 4s 75ms/step - loss: 0.5325 - accuracy: 0.7832 - val_loss: 0.4392 - va
l_accuracy: 0.8175
Epoch 9/100
54/54 [==============================] - 4s 73ms/step - loss: 0.5013 - accuracy: 0.7880 - val_loss: 0.4263 - va
l_accuracy: 0.8210
Epoch 10/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4628 - accuracy: 0.8031 - val_loss: 0.3825 - va
l_accuracy: 0.8383
Epoch 11/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4528 - accuracy: 0.8072 - val_loss: 0.3708 - va
l_accuracy: 0.8482
Epoch 12/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4167 - accuracy: 0.8258 - val_loss: 0.3418 - va
l_accuracy: 0.8591
Epoch 13/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4007 - accuracy: 0.8257 - val_loss: 0.3261 - va
l_accuracy: 0.8690
Epoch 14/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3760 - accuracy: 0.8354 - val_loss: 0.3099 - va
l_accuracy: 0.8693
Epoch 15/100
54/54 [==============================] - 4s 75ms/step - loss: 0.3812 - accuracy: 0.8370 - val_loss: 0.3368 - va
l_accuracy: 0.8564
Epoch 16/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3647 - accuracy: 0.8485 - val_loss: 0.2938 - va
l_accuracy: 0.8799
Epoch 17/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3617 - accuracy: 0.8453 - val_loss: 0.2982 - va
l_accuracy: 0.8769
Epoch 18/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3332 - accuracy: 0.8594 - val_loss: 0.2679 - va
l_accuracy: 0.8916
Epoch 19/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3288 - accuracy: 0.8598 - val_loss: 0.2583 - va
l_accuracy: 0.8925
Epoch 20/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3024 - accuracy: 0.8680 - val_loss: 0.2599 - va
l_accuracy: 0.8936
Epoch 21/100
54/54 [==============================] - 4s 75ms/step - loss: 0.3023 - accuracy: 0.8740 - val_loss: 0.2609 - va
l_accuracy: 0.8907
Epoch 22/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3019 - accuracy: 0.8698 - val_loss: 0.2431 - va
l_accuracy: 0.8998
Epoch 23/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2781 - accuracy: 0.8792 - val_loss: 0.2289 - va
l_accuracy: 0.9139
Epoch 24/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2751 - accuracy: 0.8857 - val_loss: 0.2302 - va
l_accuracy: 0.9098
Epoch 25/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2752 - accuracy: 0.8830 - val_loss: 0.2315 - va
l_accuracy: 0.9168
```

```
Epoch 26/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2674 - accuracy: 0.8871 - val_loss: 0.2444 - va
l_accuracy: 0.9077
Epoch 27/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2497 - accuracy: 0.8967 - val_loss: 0.2089 - va
l_accuracy: 0.9209
Epoch 28/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2589 - accuracy: 0.8900 - val_loss: 0.2059 - va
l_accuracy: 0.9235
Epoch 29/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2379 - accuracy: 0.9071 - val_loss: 0.2684 - va
l_accuracy: 0.8975
Epoch 30/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2396 - accuracy: 0.9000 - val_loss: 0.1927 - va
l_accuracy: 0.9303
Epoch 31/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2198 - accuracy: 0.9112 - val_loss: 0.1981 - va
l_accuracy: 0.9294
Epoch 32/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2144 - accuracy: 0.9134 - val_loss: 0.1908 - va
l_accuracy: 0.9329
Epoch 33/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2150 - accuracy: 0.9134 - val_loss: 0.1938 - va
l_accuracy: 0.9268
Epoch 34/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2163 - accuracy: 0.9096 - val_loss: 0.1807 - va
l_accuracy: 0.9341
Epoch 35/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2115 - accuracy: 0.9095 - val_loss: 0.1977 - va
l_accuracy: 0.9265
Epoch 36/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2162 - accuracy: 0.9140 - val_loss: 0.1931 - va
l_accuracy: 0.9367
Epoch 37/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1999 - accuracy: 0.9161 - val_loss: 0.1852 - va
l_accuracy: 0.9382
Epoch 38/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1909 - accuracy: 0.9221 - val_loss: 0.1783 - va
l_accuracy: 0.9373
Epoch 39/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1843 - accuracy: 0.9275 - val_loss: 0.1808 - va
l_accuracy: 0.9396
Epoch 40/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1907 - accuracy: 0.9288 - val_loss: 0.1649 - va
l_accuracy: 0.9437
Epoch 41/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1815 - accuracy: 0.9224 - val_loss: 0.1854 - va
l_accuracy: 0.9382
Epoch 42/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1742 - accuracy: 0.9295 - val_loss: 0.1853 - va
l_accuracy: 0.9352
Epoch 43/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1851 - accuracy: 0.9238 - val_loss: 0.1733 - va
l_accuracy: 0.9405
Epoch 44/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1808 - accuracy: 0.9287 - val_loss: 0.1743 - va
l_accuracy: 0.9373
Epoch 45/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1716 - accuracy: 0.9344 - val_loss: 0.1890 - va
l_accuracy: 0.9423
Epoch 46/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1797 - accuracy: 0.9278 - val_loss: 0.1634 - va
l_accuracy: 0.9452
Epoch 47/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1763 - accuracy: 0.9303 - val_loss: 0.1574 - va
l_accuracy: 0.9484
Epoch 48/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1822 - accuracy: 0.9272 - val_loss: 0.1869 - va
l_accuracy: 0.9212
Epoch 49/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1612 - accuracy: 0.9332 - val_loss: 0.1715 - va
l_accuracy: 0.9423
Epoch 50/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1522 - accuracy: 0.9395 - val_loss: 0.2023 - va
l_accuracy: 0.9402
Epoch 51/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1939 - accuracy: 0.9234 - val_loss: 0.1692 - va
l_accuracy: 0.9396
Epoch 52/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1751 - accuracy: 0.9336 - val_loss: 0.1982 - va
l_accuracy: 0.9352
Epoch 53/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1653 - accuracy: 0.9364 - val_loss: 0.1854 - va
l_accuracy: 0.9411
Epoch 54/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1662 - accuracy: 0.9328 - val_loss: 0.1860 - va
l_accuracy: 0.9452
Epoch 55/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1485 - accuracy: 0.9388 - val_loss: 0.1651 - va
```

```
l_accuracy: 0.9417
Epoch 56/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1502 - accuracy: 0.9427 - val_loss: 0.1724 - va
l_accuracy: 0.9440
Epoch 57/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1392 - accuracy: 0.9474 - val_loss: 0.1662 - va
l_accuracy: 0.9493
Epoch 58/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1392 - accuracy: 0.9443 - val_loss: 0.1773 - va
l_accuracy: 0.9440
Epoch 59/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1527 - accuracy: 0.9401 - val_loss: 0.1706 - va
l_accuracy: 0.9473
Epoch 60/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1542 - accuracy: 0.9376 - val_loss: 0.1977 - va
l_accuracy: 0.9385
Epoch 61/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1531 - accuracy: 0.9411 - val_loss: 0.1900 - va
l_accuracy: 0.9440
Epoch 62/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1469 - accuracy: 0.9408 - val_loss: 0.1692 - va
l_accuracy: 0.9522
Epoch 63/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1537 - accuracy: 0.9420 - val_loss: 0.1752 - va
l_accuracy: 0.9493
Epoch 64/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1294 - accuracy: 0.9514 - val_loss: 0.1588 - va
l_accuracy: 0.9558
Epoch 65/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1329 - accuracy: 0.9464 - val_loss: 0.1723 - va
l_accuracy: 0.9563
Epoch 66/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1795 - accuracy: 0.9307 - val_loss: 0.1730 - va
l_accuracy: 0.9429
Epoch 67/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1467 - accuracy: 0.9471 - val_loss: 0.1761 - va
l_accuracy: 0.9455
Epoch 68/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1319 - accuracy: 0.9505 - val_loss: 0.1987 - va
l_accuracy: 0.9402
Epoch 69/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1387 - accuracy: 0.9481 - val_loss: 0.1891 - va
l_accuracy: 0.9467
Epoch 70/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1363 - accuracy: 0.9452 - val_loss: 0.1635 - va
l_accuracy: 0.9593
Epoch 71/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1293 - accuracy: 0.9536 - val_loss: 0.1750 - va
l_accuracy: 0.9473
Epoch 72/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1242 - accuracy: 0.9533 - val_loss: 0.1985 - va
l_accuracy: 0.9517
Epoch 73/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1456 - accuracy: 0.9455 - val_loss: 0.1887 - va
l_accuracy: 0.9476
Epoch 74/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1323 - accuracy: 0.9505 - val_loss: 0.1600 - va
l_accuracy: 0.9555
Epoch 75/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1185 - accuracy: 0.9518 - val_loss: 0.1767 - va
l_accuracy: 0.9525
Epoch 76/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1170 - accuracy: 0.9550 - val_loss: 0.1891 - va
l_accuracy: 0.9519
Epoch 77/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1241 - accuracy: 0.9487 - val_loss: 0.2368 - va
l_accuracy: 0.9440
Epoch 78/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1211 - accuracy: 0.9506 - val_loss: 0.1548 - va
l_accuracy: 0.9508
Epoch 79/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1176 - accuracy: 0.9522 - val_loss: 0.1881 - va
l_accuracy: 0.9522
Epoch 80/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1180 - accuracy: 0.9550 - val_loss: 0.1768 - va
l_accuracy: 0.9558
Epoch 81/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1252 - accuracy: 0.9534 - val_loss: 0.1891 - va
l_accuracy: 0.9546
Epoch 82/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1240 - accuracy: 0.9514 - val_loss: 0.1936 - va
l_accuracy: 0.9546
Epoch 83/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1091 - accuracy: 0.9609 - val_loss: 0.1811 - va
l_accuracy: 0.9461
Epoch 84/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1200 - accuracy: 0.9522 - val_loss: 0.1788 - va
l_accuracy: 0.9549
Epoch 85/100
```

```
54/54 [==============================] - 4s 74ms/step - loss: 0.1188 - accuracy: 0.9539 - val_loss: 0.2083 - va
l_accuracy: 0.9452
Epoch 86/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1302 - accuracy: 0.9483 - val_loss: 0.1576 - va
l_accuracy: 0.9528
Epoch 87/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1115 - accuracy: 0.9565 - val_loss: 0.1681 - va
l_accuracy: 0.9531
Epoch 88/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1146 - accuracy: 0.9588 - val_loss: 0.1879 - va
l_accuracy: 0.9558
Epoch 89/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1087 - accuracy: 0.9618 - val_loss: 0.1686 - va
l_accuracy: 0.9607
Epoch 90/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1056 - accuracy: 0.9628 - val_loss: 0.1829 - va
l_accuracy: 0.9522
Epoch 91/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1198 - accuracy: 0.9580 - val_loss: 0.1910 - va
l_accuracy: 0.9549
Epoch 92/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1087 - accuracy: 0.9599 - val_loss: 0.1705 - va
l_accuracy: 0.9596
Epoch 93/100
54/54 [==============================] - 4s 75ms/step - loss: 0.0947 - accuracy: 0.9673 - val_loss: 0.1811 - va
l_accuracy: 0.9540
Epoch 94/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1017 - accuracy: 0.9624 - val_loss: 0.1614 - va
l_accuracy: 0.9549
Epoch 95/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1003 - accuracy: 0.9624 - val_loss: 0.2238 - va
l_accuracy: 0.9517
Epoch 96/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1048 - accuracy: 0.9593 - val_loss: 0.1791 - va
l_accuracy: 0.9522
Epoch 97/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1054 - accuracy: 0.9631 - val_loss: 0.2010 - va
l_accuracy: 0.9511
Epoch 98/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1157 - accuracy: 0.9587 - val_loss: 0.2256 - va
l_accuracy: 0.9432
Epoch 99/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1134 - accuracy: 0.9584 - val_loss: 0.1849 - va
l_accuracy: 0.9558
Epoch 100/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1161 - accuracy: 0.9565 - val_loss: 0.2226 - va
l_accuracy: 0.9549
107/107 [==============================] - 1s 7ms/step
The accuracy of this Neural Network is  = 0.9548784060943452
The precision of this Neural Network is = 0.9551523791738649
The reacll of this Neural Network is    = 0.9554489639939934
The f1_score of this Neural Network is  = 0.9550529567758452
# Of Cross Validation Is 3 Is Running


Epoch 1/100
54/54 [==============================] - 7s 102ms/step - loss: 2.4255 - accuracy: 0.2795 - val_loss: 2.0759 - v
al_accuracy: 0.5702
Epoch 2/100
54/54 [==============================] - 4s 75ms/step - loss: 1.7643 - accuracy: 0.4501 - val_loss: 1.3064 - va
l_accuracy: 0.6575
Epoch 3/100
54/54 [==============================] - 4s 73ms/step - loss: 1.2068 - accuracy: 0.6132 - val_loss: 0.9008 - va
l_accuracy: 0.6663
Epoch 4/100
54/54 [==============================] - 4s 74ms/step - loss: 0.9346 - accuracy: 0.6715 - val_loss: 0.7467 - va
l_accuracy: 0.7334
Epoch 5/100
54/54 [==============================] - 4s 75ms/step - loss: 0.8138 - accuracy: 0.6933 - val_loss: 0.6428 - va
l_accuracy: 0.7803
Epoch 6/100
54/54 [==============================] - 4s 74ms/step - loss: 0.7085 - accuracy: 0.7292 - val_loss: 0.5863 - va
l_accuracy: 0.7844
Epoch 7/100
54/54 [==============================] - 4s 74ms/step - loss: 0.6471 - accuracy: 0.7407 - val_loss: 0.5478 - va
l_accuracy: 0.8011
Epoch 8/100
54/54 [==============================] - 4s 75ms/step - loss: 0.6153 - accuracy: 0.7466 - val_loss: 0.4955 - va
l_accuracy: 0.8066
Epoch 9/100
54/54 [==============================] - 4s 75ms/step - loss: 0.5756 - accuracy: 0.7708 - val_loss: 0.4822 - va
l_accuracy: 0.8057
Epoch 10/100
54/54 [==============================] - 4s 74ms/step - loss: 0.5528 - accuracy: 0.7759 - val_loss: 0.4382 - va
l_accuracy: 0.8280
Epoch 11/100
54/54 [==============================] - 4s 76ms/step - loss: 0.5544 - accuracy: 0.7662 - val_loss: 0.4723 - va
l_accuracy: 0.8192
Epoch 12/100
```

```
54/54 [==============================] - 4s 74ms/step - loss: 0.5107 - accuracy: 0.7936 - val_loss: 0.3996 - va
l_accuracy: 0.8400
Epoch 13/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4829 - accuracy: 0.8043 - val_loss: 0.3808 - va
l_accuracy: 0.8535
Epoch 14/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4703 - accuracy: 0.8071 - val_loss: 0.3612 - va
l_accuracy: 0.8517
Epoch 15/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4615 - accuracy: 0.8140 - val_loss: 0.3454 - va
l_accuracy: 0.8567
Epoch 16/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4667 - accuracy: 0.8214 - val_loss: 0.3815 - va
l_accuracy: 0.8476
Epoch 17/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4348 - accuracy: 0.8332 - val_loss: 0.3280 - va
l_accuracy: 0.8690
Epoch 18/100
54/54 [==============================] - 4s 74ms/step - loss: 0.4150 - accuracy: 0.8439 - val_loss: 0.3351 - va
l_accuracy: 0.8640
Epoch 19/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3806 - accuracy: 0.8600 - val_loss: 0.2782 - va
l_accuracy: 0.8907
Epoch 20/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3903 - accuracy: 0.8524 - val_loss: 0.3158 - va
l_accuracy: 0.8731
Epoch 21/100
54/54 [==============================] - 4s 75ms/step - loss: 0.3776 - accuracy: 0.8619 - val_loss: 0.2701 - va
l_accuracy: 0.9030
Epoch 22/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3723 - accuracy: 0.8623 - val_loss: 0.2663 - va
l_accuracy: 0.9033
Epoch 23/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3616 - accuracy: 0.8661 - val_loss: 0.2636 - va
l_accuracy: 0.9042
Epoch 24/100
54/54 [==============================] - 4s 75ms/step - loss: 0.3429 - accuracy: 0.8778 - val_loss: 0.2531 - va
l_accuracy: 0.9130
Epoch 25/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3288 - accuracy: 0.8828 - val_loss: 0.2277 - va
l_accuracy: 0.9188
Epoch 26/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3229 - accuracy: 0.8885 - val_loss: 0.2525 - va
l_accuracy: 0.9124
Epoch 27/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2990 - accuracy: 0.9019 - val_loss: 0.2402 - va
l_accuracy: 0.9144
Epoch 28/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3255 - accuracy: 0.8852 - val_loss: 0.2283 - va
l_accuracy: 0.9191
Epoch 29/100
54/54 [==============================] - 4s 73ms/step - loss: 0.3142 - accuracy: 0.8937 - val_loss: 0.2160 - va
l_accuracy: 0.9297
Epoch 30/100
54/54 [==============================] - 4s 74ms/step - loss: 0.3069 - accuracy: 0.8992 - val_loss: 0.1983 - va
l_accuracy: 0.9352
Epoch 31/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2880 - accuracy: 0.9017 - val_loss: 0.2090 - va
l_accuracy: 0.9309
Epoch 32/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2850 - accuracy: 0.9065 - val_loss: 0.1963 - va
l_accuracy: 0.9341
Epoch 33/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2831 - accuracy: 0.9073 - val_loss: 0.1884 - va
l_accuracy: 0.9364
Epoch 34/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2749 - accuracy: 0.9089 - val_loss: 0.1810 - va
l_accuracy: 0.9417
Epoch 35/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2609 - accuracy: 0.9146 - val_loss: 0.1810 - va
l_accuracy: 0.9429
Epoch 36/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2637 - accuracy: 0.9134 - val_loss: 0.2015 - va
l_accuracy: 0.9241
Epoch 37/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2683 - accuracy: 0.9086 - val_loss: 0.2822 - va
l_accuracy: 0.9086
Epoch 38/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2658 - accuracy: 0.9136 - val_loss: 0.1981 - va
l_accuracy: 0.9420
Epoch 39/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2707 - accuracy: 0.9076 - val_loss: 0.1965 - va
l_accuracy: 0.9402
Epoch 40/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2561 - accuracy: 0.9175 - val_loss: 0.1694 - va
l_accuracy: 0.9505
Epoch 41/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2380 - accuracy: 0.9256 - val_loss: 0.1589 - va
l_accuracy: 0.9484
```

```
Epoch 42/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2618 - accuracy: 0.9169 - val_loss: 0.1889 - va
l_accuracy: 0.9435
Epoch 43/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2551 - accuracy: 0.9156 - val_loss: 0.1748 - va
l_accuracy: 0.9522
Epoch 44/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2341 - accuracy: 0.9243 - val_loss: 0.1687 - va
l_accuracy: 0.9458
Epoch 45/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2375 - accuracy: 0.9234 - val_loss: 0.1976 - va
l_accuracy: 0.9396
Epoch 46/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2430 - accuracy: 0.9208 - val_loss: 0.1704 - va
l_accuracy: 0.9490
Epoch 47/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2373 - accuracy: 0.9247 - val_loss: 0.1777 - va
l_accuracy: 0.9408
Epoch 48/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2328 - accuracy: 0.9266 - val_loss: 0.1805 - va
l_accuracy: 0.9426
Epoch 49/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2353 - accuracy: 0.9235 - val_loss: 0.1851 - va
l_accuracy: 0.9420
Epoch 50/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2256 - accuracy: 0.9297 - val_loss: 0.1710 - va
l_accuracy: 0.9449
Epoch 51/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2079 - accuracy: 0.9326 - val_loss: 0.1646 - va
l_accuracy: 0.9464
Epoch 52/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2293 - accuracy: 0.9262 - val_loss: 0.1699 - va
l_accuracy: 0.9531
Epoch 53/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2028 - accuracy: 0.9367 - val_loss: 0.1590 - va
l_accuracy: 0.9499
Epoch 54/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2289 - accuracy: 0.9271 - val_loss: 0.1577 - va
l_accuracy: 0.9499
Epoch 55/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2168 - accuracy: 0.9298 - val_loss: 0.1728 - va
l_accuracy: 0.9355
Epoch 56/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2180 - accuracy: 0.9313 - val_loss: 0.1551 - va
l_accuracy: 0.9540
Epoch 57/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2123 - accuracy: 0.9335 - val_loss: 0.1855 - va
l_accuracy: 0.9546
Epoch 58/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2178 - accuracy: 0.9338 - val_loss: 0.1608 - va
l_accuracy: 0.9587
Epoch 59/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2097 - accuracy: 0.9325 - val_loss: 0.1762 - va
l_accuracy: 0.9505
Epoch 60/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1990 - accuracy: 0.9370 - val_loss: 0.1775 - va
l_accuracy: 0.9481
Epoch 61/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2084 - accuracy: 0.9335 - val_loss: 0.1654 - va
l_accuracy: 0.9552
Epoch 62/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2098 - accuracy: 0.9351 - val_loss: 0.1818 - va
l_accuracy: 0.9426
Epoch 63/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2030 - accuracy: 0.9354 - val_loss: 0.1509 - va
l_accuracy: 0.9543
Epoch 64/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2030 - accuracy: 0.9379 - val_loss: 0.1688 - va
l_accuracy: 0.9522
Epoch 65/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1955 - accuracy: 0.9402 - val_loss: 0.1614 - va
l_accuracy: 0.9537
Epoch 66/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2171 - accuracy: 0.9323 - val_loss: 0.1483 - va
l_accuracy: 0.9604
Epoch 67/100
54/54 [==============================] - 4s 73ms/step - loss: 0.2098 - accuracy: 0.9335 - val_loss: 0.1725 - va
l_accuracy: 0.9543
Epoch 68/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2279 - accuracy: 0.9235 - val_loss: 0.1817 - va
l_accuracy: 0.9391
Epoch 69/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2107 - accuracy: 0.9316 - val_loss: 0.1547 - va
l_accuracy: 0.9563
Epoch 70/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2151 - accuracy: 0.9314 - val_loss: 0.1485 - va
l_accuracy: 0.9610
Epoch 71/100
54/54 [==============================] - 4s 75ms/step - loss: 0.2080 - accuracy: 0.9345 - val_loss: 0.1469 - va
```

```
l_accuracy: 0.9537
Epoch 72/100
54/54 [==============================] - 4s 74ms/step - loss: 0.2013 - accuracy: 0.9354 - val_loss: 0.1408 - va
l_accuracy: 0.9596
Epoch 73/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1953 - accuracy: 0.9380 - val_loss: 0.1567 - va
l_accuracy: 0.9566
Epoch 74/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1961 - accuracy: 0.9370 - val_loss: 0.1655 - va
l_accuracy: 0.9496
Epoch 75/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1908 - accuracy: 0.9405 - val_loss: 0.1926 - va
l_accuracy: 0.9499
Epoch 76/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1927 - accuracy: 0.9394 - val_loss: 0.1731 - va
l_accuracy: 0.9473
Epoch 77/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1950 - accuracy: 0.9405 - val_loss: 0.1458 - va
l_accuracy: 0.9607
Epoch 78/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1977 - accuracy: 0.9361 - val_loss: 0.1409 - va
l_accuracy: 0.9604
Epoch 79/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1726 - accuracy: 0.9474 - val_loss: 0.1521 - va
l_accuracy: 0.9561
Epoch 80/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1768 - accuracy: 0.9446 - val_loss: 0.1380 - va
l_accuracy: 0.9587
Epoch 81/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1897 - accuracy: 0.9424 - val_loss: 0.1615 - va
l_accuracy: 0.9558
Epoch 82/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1934 - accuracy: 0.9388 - val_loss: 0.1798 - va
l_accuracy: 0.9546
Epoch 83/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1960 - accuracy: 0.9399 - val_loss: 0.1636 - va
l_accuracy: 0.9584
Epoch 84/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1912 - accuracy: 0.9388 - val_loss: 0.1609 - va
l_accuracy: 0.9478
Epoch 85/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1887 - accuracy: 0.9413 - val_loss: 0.1392 - va
l_accuracy: 0.9584
Epoch 86/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1940 - accuracy: 0.9395 - val_loss: 0.1392 - va
l_accuracy: 0.9613
Epoch 87/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1806 - accuracy: 0.9451 - val_loss: 0.1541 - va
l_accuracy: 0.9610
Epoch 88/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1729 - accuracy: 0.9487 - val_loss: 0.1506 - va
l_accuracy: 0.9569
Epoch 89/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1925 - accuracy: 0.9383 - val_loss: 0.1418 - va
l_accuracy: 0.9625
Epoch 90/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1749 - accuracy: 0.9451 - val_loss: 0.1551 - va
l_accuracy: 0.9631
Epoch 91/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1798 - accuracy: 0.9436 - val_loss: 0.1738 - va
l_accuracy: 0.9590
Epoch 92/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1898 - accuracy: 0.9395 - val_loss: 0.1509 - va
l_accuracy: 0.9578
Epoch 93/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1877 - accuracy: 0.9410 - val_loss: 0.1782 - va
l_accuracy: 0.9566
Epoch 94/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1836 - accuracy: 0.9421 - val_loss: 0.2063 - va
l_accuracy: 0.9537
Epoch 95/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1858 - accuracy: 0.9454 - val_loss: 0.1725 - va
l_accuracy: 0.9572
Epoch 96/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1677 - accuracy: 0.9492 - val_loss: 0.1375 - va
l_accuracy: 0.9631
Epoch 97/100
54/54 [==============================] - 4s 74ms/step - loss: 0.1822 - accuracy: 0.9411 - val_loss: 0.1448 - va
l_accuracy: 0.9616
Epoch 98/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1850 - accuracy: 0.9417 - val_loss: 0.2002 - va
l_accuracy: 0.9561
Epoch 99/100
54/54 [==============================] - 4s 73ms/step - loss: 0.1923 - accuracy: 0.9379 - val_loss: 0.1367 - va
l_accuracy: 0.9637
Epoch 100/100
54/54 [==============================] - 4s 75ms/step - loss: 0.1790 - accuracy: 0.9445 - val_loss: 0.1629 - va
l_accuracy: 0.9607
107/107 [==============================] - 1s 5ms/step
```

```
The accuracy of this Neural Network is  = 0.9607383533548198
The precision of this Neural Network is = 0.9607073410987146
The reacll of this Neural Network is    = 0.961019951540071
The f1_score of this Neural Network is  = 0.9608109227079336
```

In [ ]:
```
print('Using KFold - 3, the accuracy is  = {}'.format(np.array(accuracy).mean(),'\n'))
print('Using KFold - 3, the precision is = {}'.format(np.array(precision).mean(),'\n'))
print('Using KFold - 3, the recall is    = {}'.format(np.array(recall).mean(),'\n'))
print('Using KFold - 3, the f1 score is  = {}'.format(np.array(f1).mean(),'\n'))
```

```
Using KFold - 3, the accuracy is  = 0.9614250748642306
Using KFold - 3, the precision is = 0.9614633754605251
Using KFold - 3, the recall is    = 0.961719936804057
Using KFold - 3, the f1 score is  = 0.9614580179600067
```

Use `Transfer Learning` Technique to check the performance with respect to Dataset

- VGG16
- ResNet
- InceptionNet
- MobileNet

`VGG16` - Model used to evaluate the model

In [ ]:
```
from tensorflow.keras.applications import VGG16, ResNet50, InceptionV3, MobileNet
```

In [ ]:
```
def VGG16_model_function(include_top = None, weights = None, input_shape = None, classes = None):
  if len(input_shape) == 0:
    raise listEmptyException('Input shape is empty'.title())
  else:
    print('Welcome to VGG16')
    VGG16_model = VGG16(include_top = include_top,\
                        weights = weights,\
                        input_shape = input_shape,\
                        classes = classes)

  return VGG16_model

try:
  VGG16_model = VGG16_model_function(include_top = False,\
                      weights = 'imagenet',\
                      input_shape = X_train.shape[1:],\
                      classes = len(np.unique(y_train)))
except listEmptyException as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  VGG16_model.summary()
```

```
Welcome to VGG16
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 3s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Make the `trainable` parameter false

In [ ]:

```
try:
  VGG16_model.trainable = False
except Exception as e:
  print('The exception is {}'.format(e))
else:
  VGG16_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Do the `Fine-tuning` with respect to VGG16 model

In [ ]:

```python
class VGG16Exception(Exception):
  def __init__(self, message):
    return message.title()

def VGG16_fine_tuning(VGG16_model = None, activate = None):
  if activate == 'YES':
    #### Create a sequential model ####
    model_VGG16 = Sequential()

    #### Add the VGG16 model to this sequential model ####
    model_VGG16.add(VGG16_model)

    #### Do the Flatten operation ####
    model_VGG16.add(Flatten())

    #### Add the user defined - fully connected layer with respect to problem description ####

    model_VGG16.add(Dense(units = 512, activation = 'relu', kernel_initializer = 'he_normal', kernel_regularize

    #### Use the Dropout layer with the ratio = 0.5 ####

    model_VGG16.add(Dropout(rate = 0.1))

    #### Add another connected layer with neurons 128 ####

    model_VGG16.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

    #### Use the Dropout layer with the ratio = 0.6 ####

    model_VGG16.add(Dropout(rate = 0.6))

    #### Add the output layer ####
    model_VGG16.add(Dense(units = 4, activation = 'softmax'))

    #### Compile the model and check the performace ####
    model_VGG16.compile(optimizer = Adam(learning_rate = 0.0001), loss = SparseCategoricalCrossentropy(), metri

  else:
```

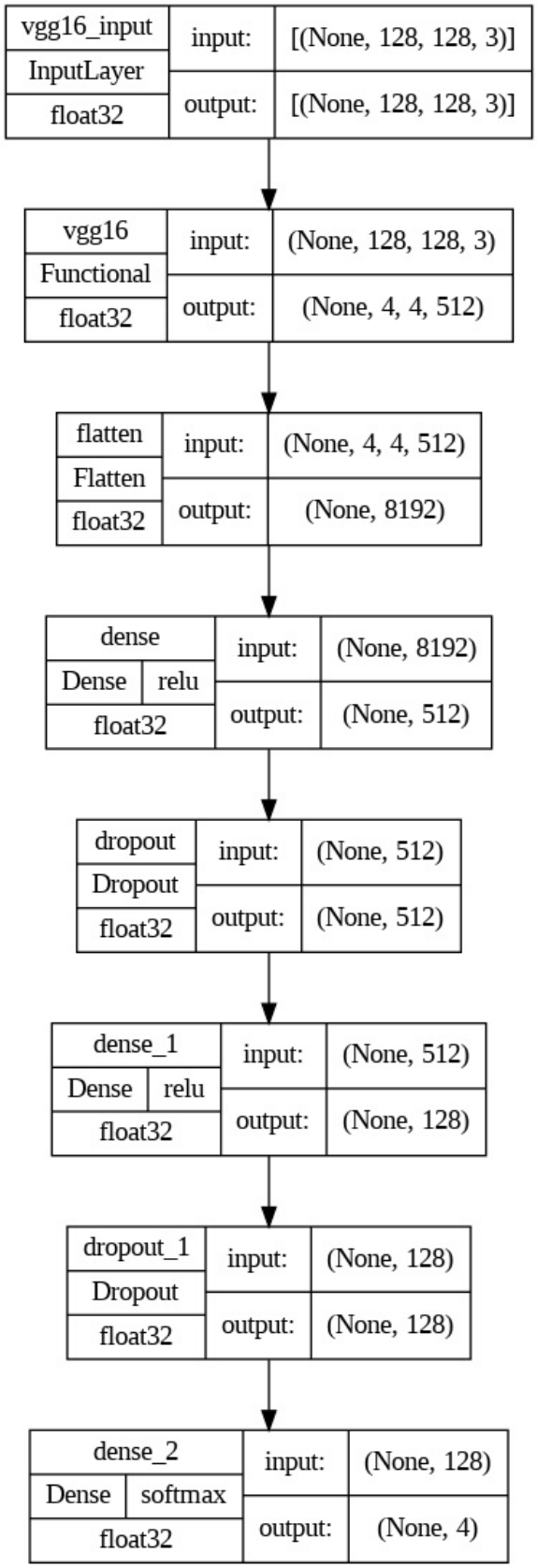
```
    raise Exception('VGG16 cannot be accessible')

  return model_VGG16

try:
  VGG16_model = VGG16_fine_tuning(VGG16_model = VGG16_model, activate = 'YES')
except VGG16Exception as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  VGG16_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 18,975,684
Trainable params: 4,260,996
Non-trainable params: 14,714,688
_________________________________________________________________
```

`plot` the VGG16 model as a graph

In [ ]:
```
plot_model(model = VGG16_model,\
           show_shapes = True,\
           show_dtype = True,\
           show_layer_names = True,\
           show_layer_activations = True)
```

Out[ ]:



In [ ]:

```
def training(x = None, y = None, epochs = None, batch_size = None):
  if (len(x) == 0 or len(y) == 0):
    raise listEmptyException('List is empty in VGG16'.title())
  else:
    history = VGG16_model.fit(x = x,\
                              y = y,\
                              epochs = epochs,\
                              batch_size = batch_size,\
                              verbose = 1,\
                              validation_data = (X_test, y_test))
  return history, VGG16_model

try:
  history, VGG16_model = training(x = X_train,\
                              y = y_train,\
                              epochs = 50,\
                              batch_size = 128)
except listEmptyException as e:
  print('The exception is {}'.format(e))
```

```
except Exception as e:
  print('The exception is {}'.format(e))
else:
  print('Completed !')
```

```
Epoch 1/50
60/60 [==============================] - 27s 216ms/step - loss: 9.3601 - accuracy: 0.3932 - val_loss: 7.1920 -
val_accuracy: 0.6062
Epoch 2/50
60/60 [==============================] - 12s 200ms/step - loss: 5.8911 - accuracy: 0.5759 - val_loss: 4.6235 -
val_accuracy: 0.6824
Epoch 3/50
60/60 [==============================] - 12s 206ms/step - loss: 3.9518 - accuracy: 0.6525 - val_loss: 3.2363 -
val_accuracy: 0.6984
Epoch 4/50
60/60 [==============================] - 13s 212ms/step - loss: 2.8531 - accuracy: 0.6863 - val_loss: 2.4074 -
val_accuracy: 0.7324
Epoch 5/50
60/60 [==============================] - 13s 217ms/step - loss: 2.2084 - accuracy: 0.7132 - val_loss: 1.8974 -
val_accuracy: 0.7563
Epoch 6/50
60/60 [==============================] - 13s 222ms/step - loss: 1.8381 - accuracy: 0.7215 - val_loss: 1.6177 -
val_accuracy: 0.7785
Epoch 7/50
60/60 [==============================] - 14s 232ms/step - loss: 1.5825 - accuracy: 0.7423 - val_loss: 1.4361 -
val_accuracy: 0.7746
Epoch 8/50
60/60 [==============================] - 14s 239ms/step - loss: 1.4171 - accuracy: 0.7576 - val_loss: 1.2892 -
val_accuracy: 0.7875
Epoch 9/50
60/60 [==============================] - 14s 229ms/step - loss: 1.3072 - accuracy: 0.7635 - val_loss: 1.2120 -
val_accuracy: 0.7910
Epoch 10/50
60/60 [==============================] - 13s 224ms/step - loss: 1.2054 - accuracy: 0.7790 - val_loss: 1.1222 -
val_accuracy: 0.8023
Epoch 11/50
60/60 [==============================] - 13s 223ms/step - loss: 1.1389 - accuracy: 0.7849 - val_loss: 1.0530 -
val_accuracy: 0.8051
Epoch 12/50
60/60 [==============================] - 13s 224ms/step - loss: 1.0754 - accuracy: 0.7922 - val_loss: 1.0041 -
val_accuracy: 0.8191
Epoch 13/50
60/60 [==============================] - 14s 227ms/step - loss: 1.0304 - accuracy: 0.7949 - val_loss: 0.9789 -
val_accuracy: 0.8031
Epoch 14/50
60/60 [==============================] - 14s 229ms/step - loss: 0.9910 - accuracy: 0.8027 - val_loss: 0.9190 -
val_accuracy: 0.8270
Epoch 15/50
60/60 [==============================] - 14s 228ms/step - loss: 0.9504 - accuracy: 0.8059 - val_loss: 0.9087 -
val_accuracy: 0.8211
Epoch 16/50
60/60 [==============================] - 13s 226ms/step - loss: 0.9109 - accuracy: 0.8143 - val_loss: 0.8785 -
val_accuracy: 0.8219
Epoch 17/50
60/60 [==============================] - 13s 225ms/step - loss: 0.8822 - accuracy: 0.8169 - val_loss: 0.8649 -
val_accuracy: 0.8047
Epoch 18/50
60/60 [==============================] - 13s 225ms/step - loss: 0.8646 - accuracy: 0.8182 - val_loss: 0.8526 -
val_accuracy: 0.8199
Epoch 19/50
60/60 [==============================] - 14s 226ms/step - loss: 0.8328 - accuracy: 0.8208 - val_loss: 0.7917 -
val_accuracy: 0.8285
Epoch 20/50
60/60 [==============================] - 14s 228ms/step - loss: 0.7976 - accuracy: 0.8337 - val_loss: 0.7528 -
val_accuracy: 0.8492
Epoch 21/50
60/60 [==============================] - 15s 257ms/step - loss: 0.7843 - accuracy: 0.8305 - val_loss: 0.7528 -
val_accuracy: 0.8328
Epoch 22/50
60/60 [==============================] - 14s 226ms/step - loss: 0.7601 - accuracy: 0.8380 - val_loss: 0.7375 -
val_accuracy: 0.8359
Epoch 23/50
60/60 [==============================] - 14s 227ms/step - loss: 0.7327 - accuracy: 0.8366 - val_loss: 0.7355 -
val_accuracy: 0.8273
Epoch 24/50
60/60 [==============================] - 14s 227ms/step - loss: 0.7253 - accuracy: 0.8389 - val_loss: 0.7134 -
val_accuracy: 0.8395
Epoch 25/50
60/60 [==============================] - 14s 227ms/step - loss: 0.6985 - accuracy: 0.8509 - val_loss: 0.6715 -
val_accuracy: 0.8621
Epoch 26/50
60/60 [==============================] - 14s 228ms/step - loss: 0.6843 - accuracy: 0.8480 - val_loss: 0.6572 -
val_accuracy: 0.8629
Epoch 27/50
60/60 [==============================] - 14s 228ms/step - loss: 0.6731 - accuracy: 0.8514 - val_loss: 0.6841 -
val_accuracy: 0.8320
Epoch 28/50
60/60 [==============================] - 14s 227ms/step - loss: 0.6552 - accuracy: 0.8557 - val_loss: 0.6370 -
val_accuracy: 0.8582
```

```
Epoch 29/50
60/60 [==============================] - 13s 226ms/step - loss: 0.6575 - accuracy: 0.8529 - val_loss: 0.6180 -
val_accuracy: 0.8660
Epoch 30/50
60/60 [==============================] - 13s 226ms/step - loss: 0.6327 - accuracy: 0.8587 - val_loss: 0.6108 -
val_accuracy: 0.8648
Epoch 31/50
60/60 [==============================] - 14s 226ms/step - loss: 0.6107 - accuracy: 0.8695 - val_loss: 0.6074 -
val_accuracy: 0.8676
Epoch 32/50
60/60 [==============================] - 14s 226ms/step - loss: 0.6096 - accuracy: 0.8681 - val_loss: 0.6238 -
val_accuracy: 0.8457
Epoch 33/50
60/60 [==============================] - 14s 227ms/step - loss: 0.6059 - accuracy: 0.8641 - val_loss: 0.5893 -
val_accuracy: 0.8691
Epoch 34/50
60/60 [==============================] - 14s 227ms/step - loss: 0.5973 - accuracy: 0.8624 - val_loss: 0.6477 -
val_accuracy: 0.8328
Epoch 35/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5797 - accuracy: 0.8678 - val_loss: 0.5914 -
val_accuracy: 0.8617
Epoch 36/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5681 - accuracy: 0.8755 - val_loss: 0.5577 -
val_accuracy: 0.8719
Epoch 37/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5633 - accuracy: 0.8719 - val_loss: 0.5729 -
val_accuracy: 0.8656
Epoch 38/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5387 - accuracy: 0.8865 - val_loss: 0.5734 -
val_accuracy: 0.8570
Epoch 39/50
60/60 [==============================] - 14s 227ms/step - loss: 0.5510 - accuracy: 0.8754 - val_loss: 0.5276 -
val_accuracy: 0.8848
Epoch 40/50
60/60 [==============================] - 15s 256ms/step - loss: 0.5367 - accuracy: 0.8823 - val_loss: 0.5315 -
val_accuracy: 0.8832
Epoch 41/50
60/60 [==============================] - 13s 225ms/step - loss: 0.5230 - accuracy: 0.8897 - val_loss: 0.5454 -
val_accuracy: 0.8703
Epoch 42/50
60/60 [==============================] - 14s 227ms/step - loss: 0.5162 - accuracy: 0.8870 - val_loss: 0.5126 -
val_accuracy: 0.8836
Epoch 43/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5034 - accuracy: 0.8922 - val_loss: 0.5817 -
val_accuracy: 0.8434
Epoch 44/50
60/60 [==============================] - 14s 228ms/step - loss: 0.5027 - accuracy: 0.8924 - val_loss: 0.5343 -
val_accuracy: 0.8723
Epoch 45/50
60/60 [==============================] - 14s 227ms/step - loss: 0.4878 - accuracy: 0.8956 - val_loss: 0.5215 -
val_accuracy: 0.8770
Epoch 46/50
60/60 [==============================] - 14s 226ms/step - loss: 0.4740 - accuracy: 0.9021 - val_loss: 0.4831 -
val_accuracy: 0.8938
Epoch 47/50
60/60 [==============================] - 14s 226ms/step - loss: 0.4704 - accuracy: 0.9023 - val_loss: 0.4887 -
val_accuracy: 0.8891
Epoch 48/50
60/60 [==============================] - 14s 227ms/step - loss: 0.4788 - accuracy: 0.8943 - val_loss: 0.4796 -
val_accuracy: 0.8898
Epoch 49/50
60/60 [==============================] - 13s 226ms/step - loss: 0.4695 - accuracy: 0.8973 - val_loss: 0.4918 -
val_accuracy: 0.8844
Epoch 50/50
60/60 [==============================] - 14s 226ms/step - loss: 0.4579 - accuracy: 0.9062 - val_loss: 0.4654 -
val_accuracy: 0.8934
Completed !
```

Check the performance `VGG16`

In [ ]:

```
print('The training performace of this model is given below.\n\n'.title())

predicted_ = VGG16_model.predict(X_train)
predicted_ = np.argmax(predicted_, axis = 1)

print('The accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_train),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_train, average = 'm
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_train, average = 'macr
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_train, average = 'macro'),

print('The testing performace of this model is given below.\n\n'.title())

predicted_ = VGG16_model.predict(X_test)
predicted_ = np.argmax(predicted_, axis = 1)

print('\nThe accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_test),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_test, average = 'ma
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_test, average = 'macro
```

```
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_test, average = 'macro'),'
```

```
The Training Performace Of This Model Is Given Below.


240/240 [==============================] - 12s 41ms/step
The accuracy of this Neural Network is = 0.937109375
The precision of this Neural Network is  = 0.9373796156005102
The reacll of this Neural Network is     = 0.9399236864430675
The f1_score of this Neural Network is   = 0.9372358400554148
The Testing Performace Of This Model Is Given Below.


80/80 [==============================] - 3s 41ms/step

The accuracy of this Neural Network is = 0.893359375
The precision of this Neural Network is  = 0.8923187406112252
The reacll of this Neural Network is     = 0.897130371656491
The f1_score of this Neural Network is   = 0.8926269283180108
```

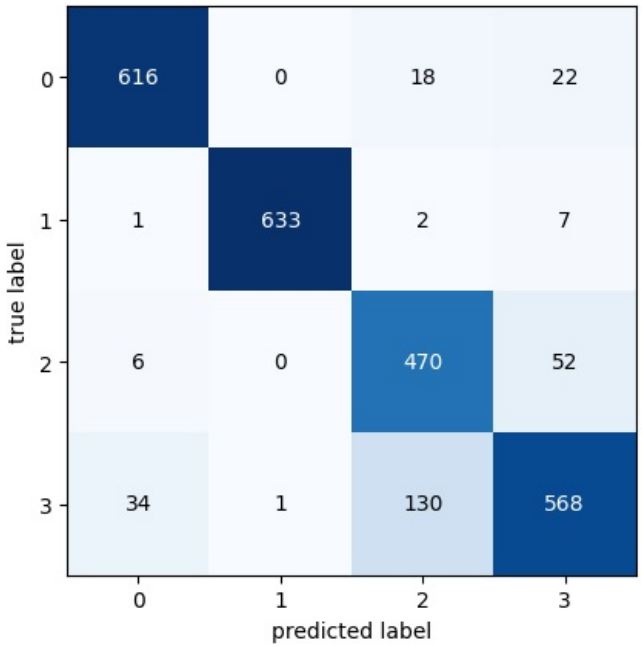
Show the `Classification` report to this model

In [ ]:
```
print('The classification report of this testing model is given below.\n'.capitalize())
print(classification_report(predicted_, y_test))
```

```
The classification report of this testing model is given below.

              precision    recall  f1-score   support

           0       0.94      0.94      0.94       656
           1       1.00      0.98      0.99       643
           2       0.76      0.89      0.82       528
           3       0.88      0.77      0.82       733

    accuracy                           0.89      2560
   macro avg       0.89      0.90      0.89      2560
weighted avg       0.90      0.89      0.89      2560
```

Plot the `Confusion` Matrix

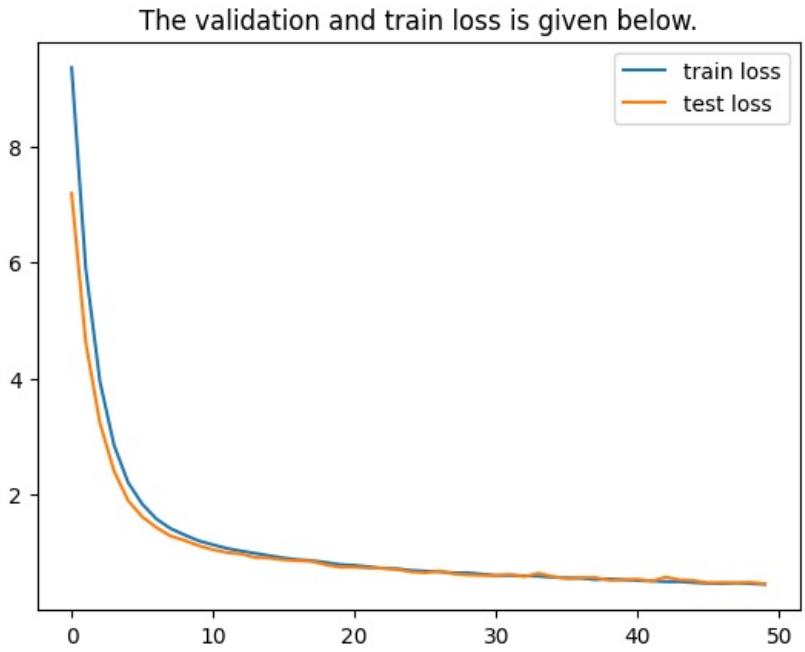
In [ ]:
```
#### Plot the confusion matrix ####
confusion_mat = confusion_matrix(predicted_, y_test)

fig, ax = plot_confusion_matrix(conf_mat = confusion_mat)

plt.show()
```



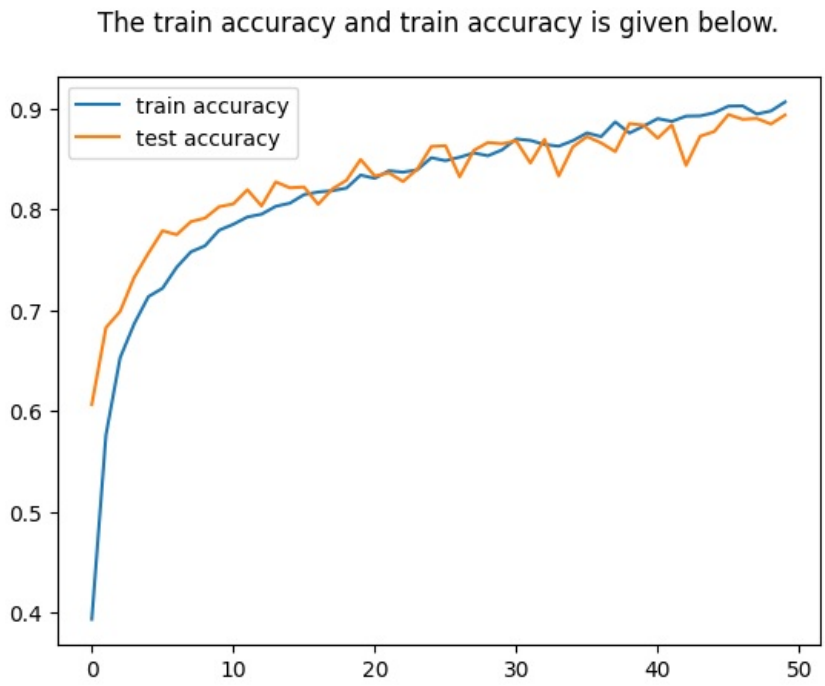
Plot `train and test` loss and `train and test` accuracy

In [ ]:
```
##### Plot the validation loss and train loss #####
plt.title('The validation and train loss is given below.')
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'test loss')
plt.legend()
plt.show()

print('*'*120,'\n')
```

```
plt.title('The train accuracy and train accuracy is given below.\n')
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.plot(history.history['val_accuracy'], label = 'test accuracy')
plt.legend()
plt.show()
```



****



Use `ResNet` architecture with respect to this dataset

In [ ]:
```
def ResNet_model_function(include_top = None, weights = None, input_shape = None, classes = None):
  if len(input_shape) == 0:
    raise listEmptyException('Input shape is empty'.title())
  else:
    print('Welcome to ResNet')
    ResNet_model = ResNet50(include_top = include_top,\
                        weights = weights,\
                        input_shape = input_shape,\
                        classes = classes)

  return VGG16_model

try:
  ResNet_model = ResNet_model_function(include_top = False,\
                      weights = 'imagenet',\
                      input_shape = X_train.shape[1:],\
                      classes = len(np.unique(y_train)))
```

```
except listEmptyException as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  ResNet_model.summary()
```

```
Welcome to ResNet
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_d
im_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 4s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 18,975,684
Trainable params: 4,260,996
Non-trainable params: 14,714,688
_________________________________________________________________
```

Make the `trainable` parameter false

In [ ]:
```
try:
  ResNet_model.trainable = False
except Exception as e:
  print('The exception is {}'.format(e))
else:
  ResNet_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 18,975,684
Trainable params: 0
Non-trainable params: 18,975,684
_________________________________________________________________
```

Do the `Fine-tuning` with respect to ResNet50 model

In [ ]:
```
class ResNetException(Exception):
  def __init__(self, message):
    return message.title()

def ResNet_fine_tuning(ResNet_model = None, activate = None):
  if activate == 'YES':
    #### Create a sequential model ####
    model_ResNet = Sequential()

    #### Add the ResNet model to this sequential model ####
    model_ResNet.add(ResNet_model)

    #### Do the Flatten operation ####
    model_ResNet.add(Flatten())

    #### Add the user defined - fully connected layer with respect to problem description ####
```

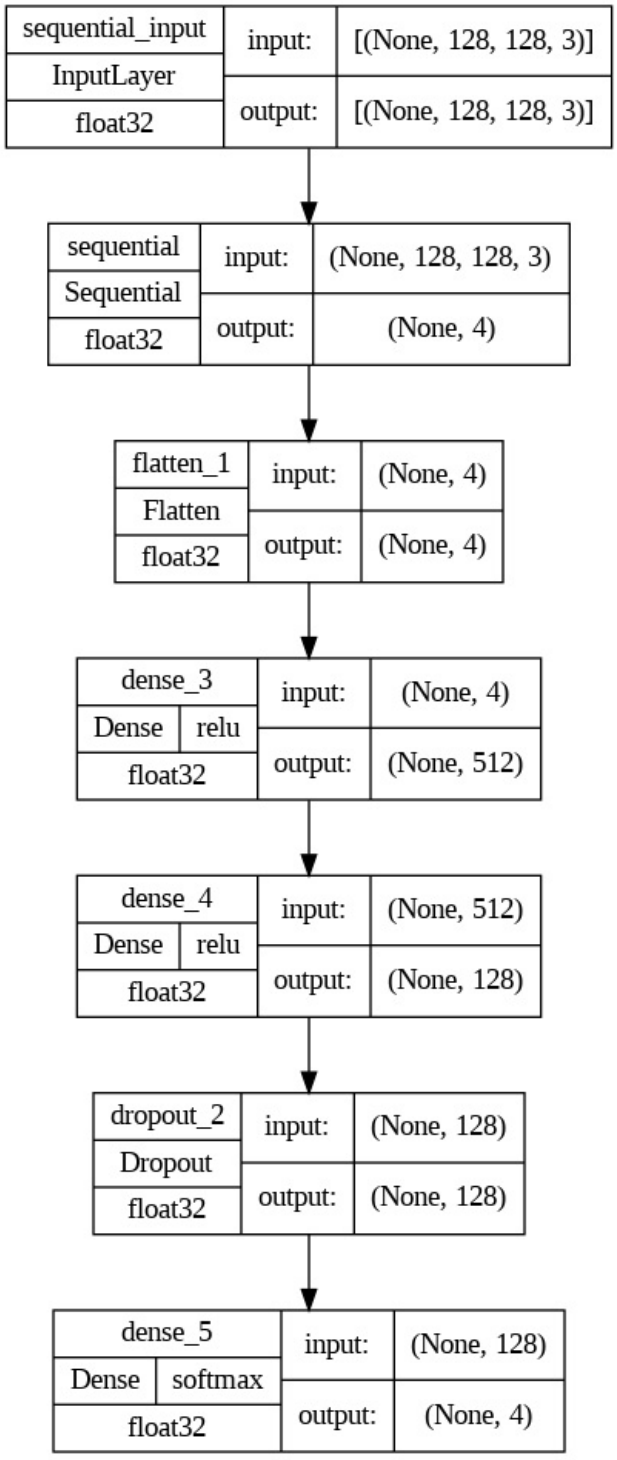
```
    model_ResNet.add(Dense(units = 512, activation = 'relu', kernel_initializer = 'he_normal', kernel_regulariz

    #### Add another connected layer with neurons 128 ####

    model_ResNet.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

    #### Use the Dropout layer with the ratio = 0.6 ####

    model_ResNet.add(Dropout(rate = 0.4))

    #### Add the output layer ####
    model_ResNet.add(Dense(units = 4, activation = 'softmax'))

    #### Compile the model and check the performace ####
    model_ResNet.compile(optimizer = Adam(learning_rate = 0.0001), loss = SparseCategoricalCrossentropy(), metr

  else:
    raise Exception('ResNet cannot be accessible')

  return model_ResNet

try:
  model_ResNet = ResNet_fine_tuning(ResNet_model = ResNet_model, activate = 'YES')
except ResNetException as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  model_ResNet.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 4)                 18975684

 flatten_1 (Flatten)         (None, 4)                 0

 dense_3 (Dense)             (None, 512)               2560

 dense_4 (Dense)             (None, 128)               65664

 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 4)                 516

=================================================================
Total params: 19,044,424
Trainable params: 68,740
Non-trainable params: 18,975,684
_________________________________________________________________
```

`plot` the VGG16 model as a graph

In [ ]:
```
plot_model(model = model_ResNet,\
           show_shapes = True,\
           show_dtype = True,\
           show_layer_names = True,\
           show_layer_activations = True)
```

Out[ ]:



In [ ]:

```
def training(x = None, y = None, epochs = None, batch_size = None):
  if (len(x) == 0 or len(y) == 0):
    raise listEmptyException('List is empty in ResNet50'.title())
  else:
    history = model_ResNet.fit(x = x,\
                               y = y,\
                               epochs = epochs,\
                               batch_size = batch_size,\
                               verbose = 1,\
                               validation_data = (X_test, y_test))
  return history, model_ResNet

try:
  history, model_ResNet = training(x = X_train,\
                               y = y_train,\
                               epochs = 20,\
                               batch_size = 64)
except listEmptyException as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  print('Completed !')
```

```
Epoch 1/20
120/120 [==============================] - 20s 122ms/step - loss: 11.1219 - accuracy: 0.8004 - val_loss: 10.720
9 - val_accuracy: 0.8988
Epoch 2/20
120/120 [==============================] - 14s 119ms/step - loss: 10.5452 - accuracy: 0.9185 - val_loss: 10.426
0 - val_accuracy: 0.9000
Epoch 3/20
120/120 [==============================] - 14s 113ms/step - loss: 10.2561 - accuracy: 0.9187 - val_loss: 10.151
6 - val_accuracy: 0.9012
Epoch 4/20
120/120 [==============================] - 13s 109ms/step - loss: 9.9766 - accuracy: 0.9210 - val_loss: 9.8869
- val_accuracy: 0.9020
Epoch 5/20
120/120 [==============================] - 13s 108ms/step - loss: 9.7138 - accuracy: 0.9206 - val_loss: 9.6304
- val_accuracy: 0.9012
Epoch 6/20
120/120 [==============================] - 13s 109ms/step - loss: 9.4540 - accuracy: 0.9199 - val_loss: 9.3768
- val_accuracy: 0.9004
Epoch 7/20
120/120 [==============================] - 13s 109ms/step - loss: 9.2106 - accuracy: 0.9210 - val_loss: 9.1299
- val_accuracy: 0.9016
Epoch 8/20
120/120 [==============================] - 13s 111ms/step - loss: 8.9706 - accuracy: 0.9232 - val_loss: 8.8926
- val_accuracy: 0.9027
Epoch 9/20
120/120 [==============================] - 13s 111ms/step - loss: 8.7394 - accuracy: 0.9220 - val_loss: 8.6601
- val_accuracy: 0.9023
Epoch 10/20
120/120 [==============================] - 13s 109ms/step - loss: 8.4961 - accuracy: 0.9240 - val_loss: 8.4344
- val_accuracy: 0.9004
Epoch 11/20
120/120 [==============================] - 13s 109ms/step - loss: 8.2787 - accuracy: 0.9203 - val_loss: 8.2134
- val_accuracy: 0.9000
Epoch 12/20
120/120 [==============================] - 13s 109ms/step - loss: 8.0559 - accuracy: 0.9237 - val_loss: 7.9965
- val_accuracy: 0.9016
Epoch 13/20
120/120 [==============================] - 13s 110ms/step - loss: 7.8483 - accuracy: 0.9233 - val_loss: 7.7866
- val_accuracy: 0.9020
Epoch 14/20
120/120 [==============================] - 13s 110ms/step - loss: 7.6516 - accuracy: 0.9161 - val_loss: 7.5815
- val_accuracy: 0.9016
Epoch 15/20
120/120 [==============================] - 13s 110ms/step - loss: 7.4351 - accuracy: 0.9221 - val_loss: 7.3829
- val_accuracy: 0.9008
Epoch 16/20
120/120 [==============================] - 13s 111ms/step - loss: 7.2357 - accuracy: 0.9232 - val_loss: 7.1873
- val_accuracy: 0.9008
Epoch 17/20
120/120 [==============================] - 13s 110ms/step - loss: 7.0536 - accuracy: 0.9212 - val_loss: 6.9960
- val_accuracy: 0.9008
Epoch 18/20
120/120 [==============================] - 13s 110ms/step - loss: 6.8579 - accuracy: 0.9242 - val_loss: 6.8119
- val_accuracy: 0.9020
Epoch 19/20
120/120 [==============================] - 13s 110ms/step - loss: 6.6743 - accuracy: 0.9249 - val_loss: 6.6298
- val_accuracy: 0.9012
Epoch 20/20
120/120 [==============================] - 13s 110ms/step - loss: 6.4916 - accuracy: 0.9250 - val_loss: 6.4557
- val_accuracy: 0.9004
Completed !
```

Check the performance `ResNet50`

In [ ]:

```
print('The training performace of ResNet50 model is given below.\n\n'.title())

predicted_ = model_ResNet.predict(X_train)
predicted_ = np.argmax(predicted_, axis = 1)

print('The accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_train),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_train, average = 'm
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_train, average = 'macr
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_train, average = 'macro'),

print('The testing performace of ResNet50 model is given below.\n\n'.title())

predicted_ = model_ResNet.predict(X_test)
predicted_ = np.argmax(predicted_, axis = 1)

print('\nThe accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_test),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_test, average = 'ma
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_test, average = 'macro
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_test, average = 'macro'),'
```

```
The Training Performace Of Resnet50 Model Is Given Below.


240/240 [==============================] - 10s 42ms/step
The accuracy of this Neural Network is = 0.9432291666666667
The precision of this Neural Network is  = 0.9432605586874345
The reacll of this Neural Network is     = 0.943348370803551
The f1_score of this Neural Network is   = 0.9432866951378033
The Testing Performace Of Resnet50 Model Is Given Below.


80/80 [==============================] - 3s 41ms/step

The accuracy of this Neural Network is = 0.900390625
The precision of this Neural Network is  = 0.9001622785792429
The reacll of this Neural Network is     = 0.8999959349409035
The f1_score of this Neural Network is   = 0.9000749459792728
```

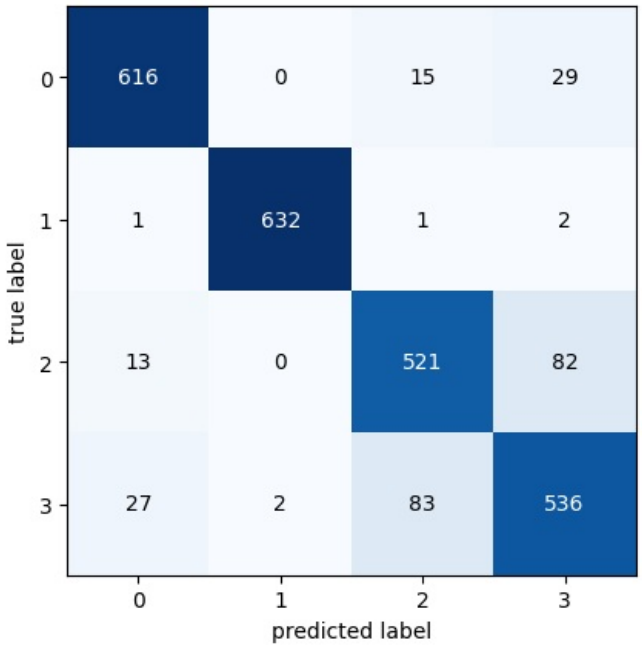
Show the `Classification` report to this model

In [ ]:
```
print('The classification report of this testing model is given below.\n'.capitalize())
print(classification_report(predicted_, y_test))
```

```
The classification report of this testing model is given below.

              precision    recall  f1-score   support

           0       0.94      0.93      0.94       660
           1       1.00      0.99      1.00       636
           2       0.84      0.85      0.84       616
           3       0.83      0.83      0.83       648

    accuracy                           0.90      2560
   macro avg       0.90      0.90      0.90      2560
weighted avg       0.90      0.90      0.90      2560
```

Plot the `Confusion` Matrix

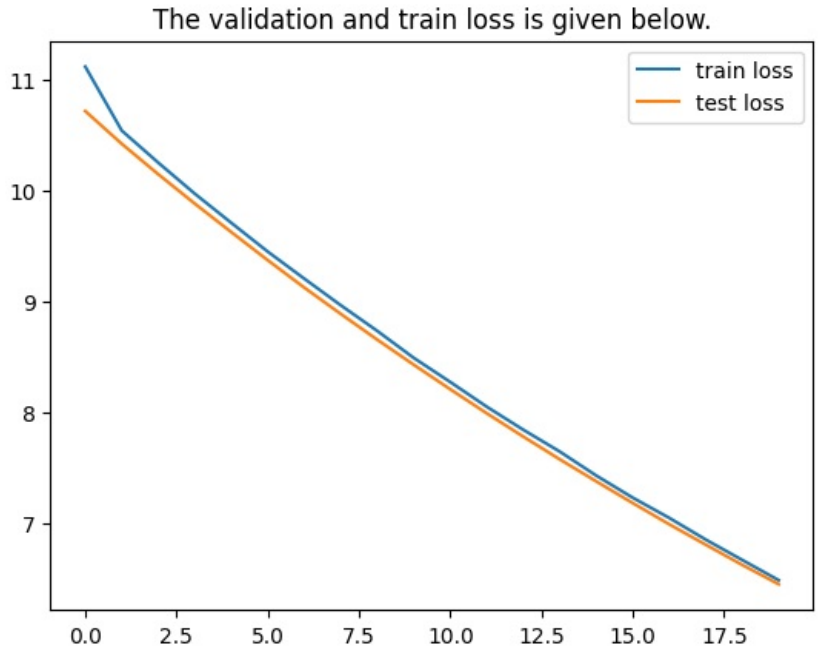
In [ ]:
```
#### Plot the confusion matrix ####
confusion_mat = confusion_matrix(predicted_, y_test)

fig, ax = plot_confusion_matrix(conf_mat = confusion_mat)

plt.show()
```



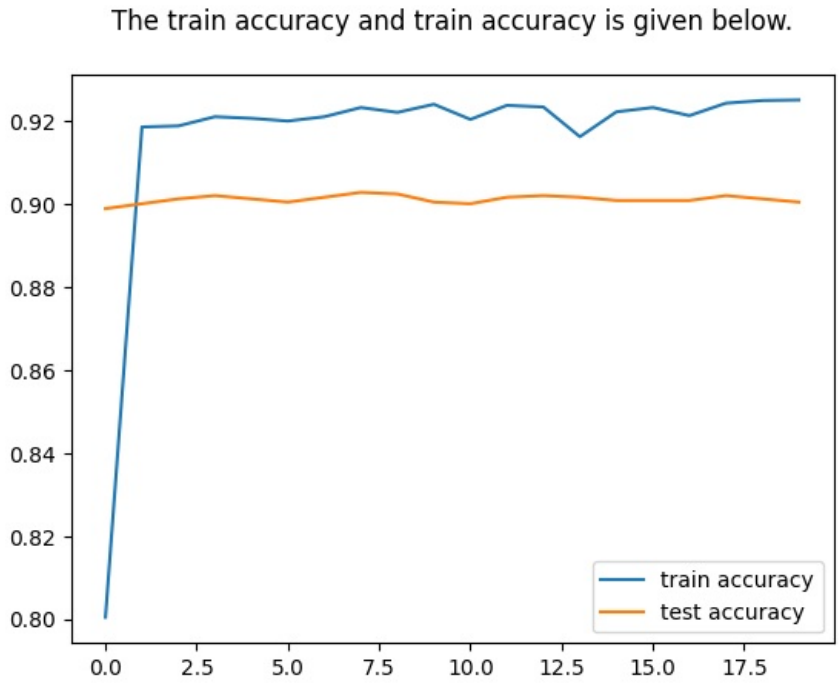
Plot `train and test` loss and `train and test` accuracy

In [ ]:
```
##### Plot the validation loss and train loss #####
plt.title('The validation and train loss is given below.')
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'test loss')
plt.legend()
plt.show()

print('*'*120,'\n')

plt.title('The train accuracy and train accuracy is given below.\n')
```

```
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.plot(history.history['val_accuracy'], label = 'test accuracy')
plt.legend()
plt.show()
```



****



Use `InceptionNet` architecture with respect to this dataset

In [ ]:
```
def Inception_model_function(include_top = None, weights = None, input_shape = None, classes = None):
  if len(input_shape) == 0:
    raise listEmptyException('Input shape is empty'.title())
  else:
    print('Welcome to ResNet')
    Inception_model = InceptionV3(include_top = include_top,\
                        weights = weights,\
                        input_shape = input_shape,\
                        classes = classes)

  return Inception_model

try:
  Inception_model = Inception_model_function(include_top = False,\
                      weights = 'imagenet',\
                      input_shape = X_train.shape[1:],\
                      classes = len(np.unique(y_train)))
except listEmptyException as e:
  print('The exception is {}'.format(e))
```

```
except Exception as e:
  print('The exception is {}'.format(e))
else:
  Inception_model.summary()
```

```
Welcome to ResNet
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 4s 0us/step
Model: "inception_v3"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 128, 128, 3)]	0	[]
conv2d (Conv2D)	(None, 63, 63, 32)	864	['input_3[0][0]']
batch_normalization (BatchNorm alization)	(None, 63, 63, 32)	96	['conv2d[0][0]']
activation (Activation)	(None, 63, 63, 32)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 61, 61, 32)	9216	['activation[0][0]']
batch_normalization_1 (BatchNo rmalization)	(None, 61, 61, 32)	96	['conv2d_1[0][0]']
activation_1 (Activation)	(None, 61, 61, 32)	0	['batch_normalization_1[0][0]']
conv2d_2 (Conv2D)	(None, 61, 61, 64)	18432	['activation_1[0][0]']
batch_normalization_2 (BatchNo rmalization)	(None, 61, 61, 64)	192	['conv2d_2[0][0]']
activation_2 (Activation)	(None, 61, 61, 64)	0	['batch_normalization_2[0][0]']
max_pooling2d (MaxPooling2D)	(None, 30, 30, 64)	0	['activation_2[0][0]']
conv2d_3 (Conv2D)	(None, 30, 30, 80)	5120	['max_pooling2d[0][0]']
batch_normalization_3 (BatchNo rmalization)	(None, 30, 30, 80)	240	['conv2d_3[0][0]']
activation_3 (Activation)	(None, 30, 30, 80)	0	['batch_normalization_3[0][0]']
conv2d_4 (Conv2D)	(None, 28, 28, 192)	138240	['activation_3[0][0]']
batch_normalization_4 (BatchNo rmalization)	(None, 28, 28, 192)	576	['conv2d_4[0][0]']
activation_4 (Activation)	(None, 28, 28, 192)	0	['batch_normalization_4[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 192)	0	['activation_4[0][0]']
conv2d_8 (Conv2D)	(None, 13, 13, 64)	12288	['max_pooling2d_1[0][0]']
batch_normalization_8 (BatchNo rmalization)	(None, 13, 13, 64)	192	['conv2d_8[0][0]']
activation_8 (Activation)	(None, 13, 13, 64)	0	['batch_normalization_8[0][0]']
conv2d_6 (Conv2D)	(None, 13, 13, 48)	9216	['max_pooling2d_1[0][0]']
conv2d_9 (Conv2D)	(None, 13, 13, 96)	55296	['activation_8[0][0]']
batch_normalization_6 (BatchNo rmalization)	(None, 13, 13, 48)	144	['conv2d_6[0][0]']
batch_normalization_9 (BatchNo rmalization)	(None, 13, 13, 96)	288	['conv2d_9[0][0]']
activation_6 (Activation)	(None, 13, 13, 48)	0	['batch_normalization_6[0][0]']
activation_9 (Activation)	(None, 13, 13, 96)	0	['batch_normalization_9[0][0]']
average_pooling2d (AveragePool ing2D)	(None, 13, 13, 192)	0	['max_pooling2d_1[0][0]']
conv2d_5 (Conv2D)	(None, 13, 13, 64)	12288	['max_pooling2d_1[0][0]']
conv2d_7 (Conv2D)	(None, 13, 13, 64)	76800	['activation_6[0][0]']
conv2d_10 (Conv2D)	(None, 13, 13, 96)	82944	['activation_9[0][0]']
conv2d_11 (Conv2D)	(None, 13, 13, 32)	6144	['average_pooling2d[0][0]']
batch_normalization_5 (BatchNo rmalization)	(None, 13, 13, 64)	192	['conv2d_5[0][0]']

```
batch_normalization_7 (BatchNo  (None, 13, 13, 64)  192         ['conv2d_7[0][0]']
rmalization)

batch_normalization_10 (BatchN  (None, 13, 13, 96)  288         ['conv2d_10[0][0]']
ormalization)

batch_normalization_11 (BatchN  (None, 13, 13, 32)  96          ['conv2d_11[0][0]']
ormalization)

activation_5 (Activation)      (None, 13, 13, 64)   0           ['batch_normalization_5[0][0]']

activation_7 (Activation)      (None, 13, 13, 64)   0           ['batch_normalization_7[0][0]']

activation_10 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_10[0][0]']

activation_11 (Activation)     (None, 13, 13, 32)   0           ['batch_normalization_11[0][0]']

mixed0 (Concatenate)           (None, 13, 13, 256)  0           ['activation_5[0][0]',
                                                                 'activation_7[0][0]',
                                                                 'activation_10[0][0]',
                                                                 'activation_11[0][0]']

conv2d_15 (Conv2D)             (None, 13, 13, 64)   16384       ['mixed0[0][0]']

batch_normalization_15 (BatchN  (None, 13, 13, 64)  192         ['conv2d_15[0][0]']
ormalization)

activation_15 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_15[0][0]']

conv2d_13 (Conv2D)             (None, 13, 13, 48)   12288       ['mixed0[0][0]']

conv2d_16 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_15[0][0]']

batch_normalization_13 (BatchN  (None, 13, 13, 48)  144         ['conv2d_13[0][0]']
ormalization)

batch_normalization_16 (BatchN  (None, 13, 13, 96)  288         ['conv2d_16[0][0]']
ormalization)

activation_13 (Activation)     (None, 13, 13, 48)   0           ['batch_normalization_13[0][0]']

activation_16 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_16[0][0]']

average_pooling2d_1 (AveragePo  (None, 13, 13, 256)  0         ['mixed0[0][0]']
oling2D)

conv2d_12 (Conv2D)             (None, 13, 13, 64)   16384       ['mixed0[0][0]']

conv2d_14 (Conv2D)             (None, 13, 13, 64)   76800       ['activation_13[0][0]']

conv2d_17 (Conv2D)             (None, 13, 13, 96)   82944       ['activation_16[0][0]']

conv2d_18 (Conv2D)             (None, 13, 13, 64)   16384       ['average_pooling2d_1[0][0]']

batch_normalization_12 (BatchN  (None, 13, 13, 64)  192         ['conv2d_12[0][0]']
ormalization)

batch_normalization_14 (BatchN  (None, 13, 13, 64)  192         ['conv2d_14[0][0]']
ormalization)

batch_normalization_17 (BatchN  (None, 13, 13, 96)  288         ['conv2d_17[0][0]']
ormalization)

batch_normalization_18 (BatchN  (None, 13, 13, 64)  192         ['conv2d_18[0][0]']
ormalization)

activation_12 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_12[0][0]']

activation_14 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_14[0][0]']

activation_17 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_17[0][0]']

activation_18 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_18[0][0]']

mixed1 (Concatenate)           (None, 13, 13, 288)  0           ['activation_12[0][0]',
                                                                 'activation_14[0][0]',
                                                                 'activation_17[0][0]',
                                                                 'activation_18[0][0]']

conv2d_22 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed1[0][0]']

batch_normalization_22 (BatchN  (None, 13, 13, 64)  192         ['conv2d_22[0][0]']
ormalization)

activation_22 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_22[0][0]']

conv2d_20 (Conv2D)             (None, 13, 13, 48)   13824       ['mixed1[0][0]']
```

```
conv2d_23 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_22[0][0]']

batch_normalization_20 (BatchN  (None, 13, 13, 48)  144         ['conv2d_20[0][0]']
ormalization)

batch_normalization_23 (BatchN  (None, 13, 13, 96)  288         ['conv2d_23[0][0]']
ormalization)

activation_20 (Activation)     (None, 13, 13, 48)   0           ['batch_normalization_20[0][0]']

activation_23 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_23[0][0]']

average_pooling2d_2 (AveragePo  (None, 13, 13, 288)  0         ['mixed1[0][0]']
oling2D)

conv2d_19 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed1[0][0]']

conv2d_21 (Conv2D)             (None, 13, 13, 64)   76800       ['activation_20[0][0]']

conv2d_24 (Conv2D)             (None, 13, 13, 96)   82944       ['activation_23[0][0]']

conv2d_25 (Conv2D)             (None, 13, 13, 64)   18432       ['average_pooling2d_2[0][0]']

batch_normalization_19 (BatchN  (None, 13, 13, 64)  192         ['conv2d_19[0][0]']
ormalization)

batch_normalization_21 (BatchN  (None, 13, 13, 64)  192         ['conv2d_21[0][0]']
ormalization)

batch_normalization_24 (BatchN  (None, 13, 13, 96)  288         ['conv2d_24[0][0]']
ormalization)

batch_normalization_25 (BatchN  (None, 13, 13, 64)  192         ['conv2d_25[0][0]']
ormalization)

activation_19 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_19[0][0]']

activation_21 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_21[0][0]']

activation_24 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_24[0][0]']

activation_25 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_25[0][0]']

mixed2 (Concatenate)           (None, 13, 13, 288)  0           ['activation_19[0][0]',
                                                                 'activation_21[0][0]',
                                                                 'activation_24[0][0]',
                                                                 'activation_25[0][0]']

conv2d_27 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed2[0][0]']

batch_normalization_27 (BatchN  (None, 13, 13, 64)  192         ['conv2d_27[0][0]']
ormalization)

activation_27 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_27[0][0]']

conv2d_28 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_27[0][0]']

batch_normalization_28 (BatchN  (None, 13, 13, 96)  288         ['conv2d_28[0][0]']
ormalization)

activation_28 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_28[0][0]']

conv2d_26 (Conv2D)             (None, 6, 6, 384)    995328      ['mixed2[0][0]']

conv2d_29 (Conv2D)             (None, 6, 6, 96)     82944       ['activation_28[0][0]']

batch_normalization_26 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_26[0][0]']
ormalization)

batch_normalization_29 (BatchN  (None, 6, 6, 96)    288         ['conv2d_29[0][0]']
ormalization)

activation_26 (Activation)     (None, 6, 6, 384)    0           ['batch_normalization_26[0][0]']

activation_29 (Activation)     (None, 6, 6, 96)     0           ['batch_normalization_29[0][0]']

max_pooling2d_2 (MaxPooling2D)  (None, 6, 6, 288)   0           ['mixed2[0][0]']

mixed3 (Concatenate)           (None, 6, 6, 768)    0           ['activation_26[0][0]',
                                                                 'activation_29[0][0]',
                                                                 'max_pooling2d_2[0][0]']

conv2d_34 (Conv2D)             (None, 6, 6, 128)    98304       ['mixed3[0][0]']

batch_normalization_34 (BatchN  (None, 6, 6, 128)   384         ['conv2d_34[0][0]']
ormalization)

activation_34 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_34[0][0]']
```

```
conv2d_35 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_34[0][0]']

batch_normalization_35 (BatchN  (None, 6, 6, 128)   384         ['conv2d_35[0][0]']
ormalization)

activation_35 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_35[0][0]']

conv2d_31 (Conv2D)             (None, 6, 6, 128)    98304       ['mixed3[0][0]']

conv2d_36 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_35[0][0]']

batch_normalization_31 (BatchN  (None, 6, 6, 128)   384         ['conv2d_31[0][0]']
ormalization)

batch_normalization_36 (BatchN  (None, 6, 6, 128)   384         ['conv2d_36[0][0]']
ormalization)

activation_31 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_31[0][0]']

activation_36 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_36[0][0]']

conv2d_32 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_31[0][0]']

conv2d_37 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_36[0][0]']

batch_normalization_32 (BatchN  (None, 6, 6, 128)   384         ['conv2d_32[0][0]']
ormalization)

batch_normalization_37 (BatchN  (None, 6, 6, 128)   384         ['conv2d_37[0][0]']
ormalization)

activation_32 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_32[0][0]']

activation_37 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_37[0][0]']

average_pooling2d_3 (AveragePo  (None, 6, 6, 768)   0           ['mixed3[0][0]']
oling2D)

conv2d_30 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed3[0][0]']

conv2d_33 (Conv2D)             (None, 6, 6, 192)    172032      ['activation_32[0][0]']

conv2d_38 (Conv2D)             (None, 6, 6, 192)    172032      ['activation_37[0][0]']

conv2d_39 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_3[0][0]']

batch_normalization_30 (BatchN  (None, 6, 6, 192)   576         ['conv2d_30[0][0]']
ormalization)

batch_normalization_33 (BatchN  (None, 6, 6, 192)   576         ['conv2d_33[0][0]']
ormalization)

batch_normalization_38 (BatchN  (None, 6, 6, 192)   576         ['conv2d_38[0][0]']
ormalization)

batch_normalization_39 (BatchN  (None, 6, 6, 192)   576         ['conv2d_39[0][0]']
ormalization)

activation_30 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_30[0][0]']

activation_33 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_33[0][0]']

activation_38 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_38[0][0]']

activation_39 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_39[0][0]']

mixed4 (Concatenate)           (None, 6, 6, 768)    0           ['activation_30[0][0]',
                                                                 'activation_33[0][0]',
                                                                 'activation_38[0][0]',
                                                                 'activation_39[0][0]']

conv2d_44 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed4[0][0]']

batch_normalization_44 (BatchN  (None, 6, 6, 160)   480         ['conv2d_44[0][0]']
ormalization)

activation_44 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_44[0][0]']

conv2d_45 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_44[0][0]']

batch_normalization_45 (BatchN  (None, 6, 6, 160)   480         ['conv2d_45[0][0]']
ormalization)

activation_45 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_45[0][0]']

conv2d_41 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed4[0][0]']

conv2d_46 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_45[0][0]']
```

```
batch_normalization_41 (BatchN  (None, 6, 6, 160)   480         ['conv2d_41[0][0]']
ormalization)

batch_normalization_46 (BatchN  (None, 6, 6, 160)   480         ['conv2d_46[0][0]']
ormalization)

activation_41 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_41[0][0]']

activation_46 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_46[0][0]']

conv2d_42 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_41[0][0]']

conv2d_47 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_46[0][0]']

batch_normalization_42 (BatchN  (None, 6, 6, 160)   480         ['conv2d_42[0][0]']
ormalization)

batch_normalization_47 (BatchN  (None, 6, 6, 160)   480         ['conv2d_47[0][0]']
ormalization)

activation_42 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_42[0][0]']

activation_47 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_47[0][0]']

average_pooling2d_4 (AveragePo  (None, 6, 6, 768)   0           ['mixed4[0][0]']
oling2D)

conv2d_40 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed4[0][0]']

conv2d_43 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_42[0][0]']

conv2d_48 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_47[0][0]']

conv2d_49 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_4[0][0]']

batch_normalization_40 (BatchN  (None, 6, 6, 192)   576         ['conv2d_40[0][0]']
ormalization)

batch_normalization_43 (BatchN  (None, 6, 6, 192)   576         ['conv2d_43[0][0]']
ormalization)

batch_normalization_48 (BatchN  (None, 6, 6, 192)   576         ['conv2d_48[0][0]']
ormalization)

batch_normalization_49 (BatchN  (None, 6, 6, 192)   576         ['conv2d_49[0][0]']
ormalization)

activation_40 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_40[0][0]']

activation_43 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_43[0][0]']

activation_48 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_48[0][0]']

activation_49 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_49[0][0]']

mixed5 (Concatenate)           (None, 6, 6, 768)    0           ['activation_40[0][0]',
                                                                 'activation_43[0][0]',
                                                                 'activation_48[0][0]',
                                                                 'activation_49[0][0]']

conv2d_54 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed5[0][0]']

batch_normalization_54 (BatchN  (None, 6, 6, 160)   480         ['conv2d_54[0][0]']
ormalization)

activation_54 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_54[0][0]']

conv2d_55 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_54[0][0]']

batch_normalization_55 (BatchN  (None, 6, 6, 160)   480         ['conv2d_55[0][0]']
ormalization)

activation_55 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_55[0][0]']

conv2d_51 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed5[0][0]']

conv2d_56 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_55[0][0]']

batch_normalization_51 (BatchN  (None, 6, 6, 160)   480         ['conv2d_51[0][0]']
ormalization)

batch_normalization_56 (BatchN  (None, 6, 6, 160)   480         ['conv2d_56[0][0]']
ormalization)

activation_51 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_51[0][0]']

activation_56 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_56[0][0]']

conv2d_52 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_51[0][0]']
```

```
conv2d_57 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_56[0][0]']

batch_normalization_52 (BatchN  (None, 6, 6, 160)   480         ['conv2d_52[0][0]']
ormalization)

batch_normalization_57 (BatchN  (None, 6, 6, 160)   480         ['conv2d_57[0][0]']
ormalization)

activation_52 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_52[0][0]']

activation_57 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_57[0][0]']

average_pooling2d_5 (AveragePo  (None, 6, 6, 768)   0           ['mixed5[0][0]']
oling2D)

conv2d_50 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed5[0][0]']

conv2d_53 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_52[0][0]']

conv2d_58 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_57[0][0]']

conv2d_59 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_5[0][0]']

batch_normalization_50 (BatchN  (None, 6, 6, 192)   576         ['conv2d_50[0][0]']
ormalization)

batch_normalization_53 (BatchN  (None, 6, 6, 192)   576         ['conv2d_53[0][0]']
ormalization)

batch_normalization_58 (BatchN  (None, 6, 6, 192)   576         ['conv2d_58[0][0]']
ormalization)

batch_normalization_59 (BatchN  (None, 6, 6, 192)   576         ['conv2d_59[0][0]']
ormalization)

activation_50 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_50[0][0]']

activation_53 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_53[0][0]']

activation_58 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_58[0][0]']

activation_59 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_59[0][0]']

mixed6 (Concatenate)           (None, 6, 6, 768)    0           ['activation_50[0][0]',
                                                                 'activation_53[0][0]',
                                                                 'activation_58[0][0]',
                                                                 'activation_59[0][0]']

conv2d_64 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

batch_normalization_64 (BatchN  (None, 6, 6, 192)   576         ['conv2d_64[0][0]']
ormalization)

activation_64 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_64[0][0]']

conv2d_65 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_64[0][0]']

batch_normalization_65 (BatchN  (None, 6, 6, 192)   576         ['conv2d_65[0][0]']
ormalization)

activation_65 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_65[0][0]']

conv2d_61 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

conv2d_66 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_65[0][0]']

batch_normalization_61 (BatchN  (None, 6, 6, 192)   576         ['conv2d_61[0][0]']
ormalization)

batch_normalization_66 (BatchN  (None, 6, 6, 192)   576         ['conv2d_66[0][0]']
ormalization)

activation_61 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_61[0][0]']

activation_66 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_66[0][0]']

conv2d_62 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_61[0][0]']

conv2d_67 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_66[0][0]']

batch_normalization_62 (BatchN  (None, 6, 6, 192)   576         ['conv2d_62[0][0]']
ormalization)

batch_normalization_67 (BatchN  (None, 6, 6, 192)   576         ['conv2d_67[0][0]']
ormalization)

activation_62 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_62[0][0]']
```

```
activation_67 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_67[0][0]']

average_pooling2d_6 (AveragePo  (None, 6, 6, 768)   0           ['mixed6[0][0]']
oling2D)

conv2d_60 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

conv2d_63 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_62[0][0]']

conv2d_68 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_67[0][0]']

conv2d_69 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_6[0][0]']

batch_normalization_60 (BatchN  (None, 6, 6, 192)   576         ['conv2d_60[0][0]']
ormalization)

batch_normalization_63 (BatchN  (None, 6, 6, 192)   576         ['conv2d_63[0][0]']
ormalization)

batch_normalization_68 (BatchN  (None, 6, 6, 192)   576         ['conv2d_68[0][0]']
ormalization)

batch_normalization_69 (BatchN  (None, 6, 6, 192)   576         ['conv2d_69[0][0]']
ormalization)

activation_60 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_60[0][0]']

activation_63 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_63[0][0]']

activation_68 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_68[0][0]']

activation_69 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_69[0][0]']

mixed7 (Concatenate)           (None, 6, 6, 768)    0           ['activation_60[0][0]',
                                                                 'activation_63[0][0]',
                                                                 'activation_68[0][0]',
                                                                 'activation_69[0][0]']

conv2d_72 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed7[0][0]']

batch_normalization_72 (BatchN  (None, 6, 6, 192)   576         ['conv2d_72[0][0]']
ormalization)

activation_72 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_72[0][0]']

conv2d_73 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_72[0][0]']

batch_normalization_73 (BatchN  (None, 6, 6, 192)   576         ['conv2d_73[0][0]']
ormalization)

activation_73 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_73[0][0]']

conv2d_70 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed7[0][0]']

conv2d_74 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_73[0][0]']

batch_normalization_70 (BatchN  (None, 6, 6, 192)   576         ['conv2d_70[0][0]']
ormalization)

batch_normalization_74 (BatchN  (None, 6, 6, 192)   576         ['conv2d_74[0][0]']
ormalization)

activation_70 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_70[0][0]']

activation_74 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_74[0][0]']

conv2d_71 (Conv2D)             (None, 2, 2, 320)    552960      ['activation_70[0][0]']

conv2d_75 (Conv2D)             (None, 2, 2, 192)    331776      ['activation_74[0][0]']

batch_normalization_71 (BatchN  (None, 2, 2, 320)   960         ['conv2d_71[0][0]']
ormalization)

batch_normalization_75 (BatchN  (None, 2, 2, 192)   576         ['conv2d_75[0][0]']
ormalization)

activation_71 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_71[0][0]']

activation_75 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_75[0][0]']

max_pooling2d_3 (MaxPooling2D)  (None, 2, 2, 768)   0           ['mixed7[0][0]']

mixed8 (Concatenate)           (None, 2, 2, 1280)   0           ['activation_71[0][0]',
                                                                 'activation_75[0][0]',
                                                                 'max_pooling2d_3[0][0]']

conv2d_80 (Conv2D)             (None, 2, 2, 448)    573440      ['mixed8[0][0]']

batch_normalization_80 (BatchN  (None, 2, 2, 448)   1344        ['conv2d_80[0][0]']
```

```
ormalization)

activation_80 (Activation)     (None, 2, 2, 448)    0           ['batch_normalization_80[0][0]']

conv2d_77 (Conv2D)             (None, 2, 2, 384)    491520      ['mixed8[0][0]']

conv2d_81 (Conv2D)             (None, 2, 2, 384)    1548288     ['activation_80[0][0]']

batch_normalization_77 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_77[0][0]']
ormalization)

batch_normalization_81 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_81[0][0]']
ormalization)

activation_77 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_77[0][0]']

activation_81 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_81[0][0]']

conv2d_78 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_77[0][0]']

conv2d_79 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_77[0][0]']

conv2d_82 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_81[0][0]']

conv2d_83 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_81[0][0]']

average_pooling2d_7 (AveragePo  (None, 2, 2, 1280)  0           ['mixed8[0][0]']
oling2D)

conv2d_76 (Conv2D)             (None, 2, 2, 320)    409600      ['mixed8[0][0]']

batch_normalization_78 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_78[0][0]']
ormalization)

batch_normalization_79 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_79[0][0]']
ormalization)

batch_normalization_82 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_82[0][0]']
ormalization)

batch_normalization_83 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_83[0][0]']
ormalization)

conv2d_84 (Conv2D)             (None, 2, 2, 192)    245760      ['average_pooling2d_7[0][0]']

batch_normalization_76 (BatchN  (None, 2, 2, 320)   960         ['conv2d_76[0][0]']
ormalization)

activation_78 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_78[0][0]']

activation_79 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_79[0][0]']

activation_82 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_82[0][0]']

activation_83 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_83[0][0]']

batch_normalization_84 (BatchN  (None, 2, 2, 192)   576         ['conv2d_84[0][0]']
ormalization)

activation_76 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_76[0][0]']

mixed9_0 (Concatenate)         (None, 2, 2, 768)    0           ['activation_78[0][0]',
                                                                 'activation_79[0][0]']

concatenate (Concatenate)      (None, 2, 2, 768)    0           ['activation_82[0][0]',
                                                                 'activation_83[0][0]']

activation_84 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_84[0][0]']

mixed9 (Concatenate)           (None, 2, 2, 2048)   0           ['activation_76[0][0]',
                                                                 'mixed9_0[0][0]',
                                                                 'concatenate[0][0]',
                                                                 'activation_84[0][0]']

conv2d_89 (Conv2D)             (None, 2, 2, 448)    917504      ['mixed9[0][0]']

batch_normalization_89 (BatchN  (None, 2, 2, 448)   1344        ['conv2d_89[0][0]']
ormalization)

activation_89 (Activation)     (None, 2, 2, 448)    0           ['batch_normalization_89[0][0]']

conv2d_86 (Conv2D)             (None, 2, 2, 384)    786432      ['mixed9[0][0]']

conv2d_90 (Conv2D)             (None, 2, 2, 384)    1548288     ['activation_89[0][0]']

batch_normalization_86 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_86[0][0]']
ormalization)

batch_normalization_90 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_90[0][0]']
```

```
ormalization)

activation_86 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_86[0][0]']

activation_90 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_90[0][0]']

conv2d_87 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_86[0][0]']

conv2d_88 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_86[0][0]']

conv2d_91 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_90[0][0]']

conv2d_92 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_90[0][0]']

average_pooling2d_8 (AveragePo  (None, 2, 2, 2048)  0           ['mixed9[0][0]']
oling2D)

conv2d_85 (Conv2D)             (None, 2, 2, 320)    655360      ['mixed9[0][0]']

batch_normalization_87 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_87[0][0]']
ormalization)

batch_normalization_88 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_88[0][0]']
ormalization)

batch_normalization_91 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_91[0][0]']
ormalization)

batch_normalization_92 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_92[0][0]']
ormalization)

conv2d_93 (Conv2D)             (None, 2, 2, 192)    393216      ['average_pooling2d_8[0][0]']

batch_normalization_85 (BatchN  (None, 2, 2, 320)   960         ['conv2d_85[0][0]']
ormalization)

activation_87 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_87[0][0]']

activation_88 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_88[0][0]']

activation_91 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_91[0][0]']

activation_92 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_92[0][0]']

batch_normalization_93 (BatchN  (None, 2, 2, 192)   576         ['conv2d_93[0][0]']
ormalization)

activation_85 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_85[0][0]']

mixed9_1 (Concatenate)         (None, 2, 2, 768)    0           ['activation_87[0][0]',
                                                                 'activation_88[0][0]']

concatenate_1 (Concatenate)    (None, 2, 2, 768)    0           ['activation_91[0][0]',
                                                                 'activation_92[0][0]']

activation_93 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_93[0][0]']

mixed10 (Concatenate)          (None, 2, 2, 2048)   0           ['activation_85[0][0]',
                                                                 'mixed9_1[0][0]',
                                                                 'concatenate_1[0][0]',
                                                                 'activation_93[0][0]']

==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

Make the `trainable` parameter false

In [ ]:
```
try:
  Inception_model.trainable = False
except Exception as e:
  print('The exception is {}'.format(e))
else:
  Inception_model.summary()
```

```
Model: "inception_v3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 128, 128, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 63, 63, 32)   864         ['input_3[0][0]']

 batch_normalization (BatchNorm  (None, 63, 63, 32)  96          ['conv2d[0][0]']
 alization)
```

```
activation (Activation)        (None, 63, 63, 32)   0           ['batch_normalization[0][0]']

conv2d_1 (Conv2D)              (None, 61, 61, 32)   9216        ['activation[0][0]']

batch_normalization_1 (BatchNo  (None, 61, 61, 32)  96          ['conv2d_1[0][0]']
rmalization)

activation_1 (Activation)      (None, 61, 61, 32)   0           ['batch_normalization_1[0][0]']

conv2d_2 (Conv2D)              (None, 61, 61, 64)   18432       ['activation_1[0][0]']

batch_normalization_2 (BatchNo  (None, 61, 61, 64)  192         ['conv2d_2[0][0]']
rmalization)

activation_2 (Activation)      (None, 61, 61, 64)   0           ['batch_normalization_2[0][0]']

max_pooling2d (MaxPooling2D)   (None, 30, 30, 64)   0           ['activation_2[0][0]']

conv2d_3 (Conv2D)              (None, 30, 30, 80)   5120        ['max_pooling2d[0][0]']

batch_normalization_3 (BatchNo  (None, 30, 30, 80)  240         ['conv2d_3[0][0]']
rmalization)

activation_3 (Activation)      (None, 30, 30, 80)   0           ['batch_normalization_3[0][0]']

conv2d_4 (Conv2D)              (None, 28, 28, 192)  138240      ['activation_3[0][0]']

batch_normalization_4 (BatchNo  (None, 28, 28, 192)  576        ['conv2d_4[0][0]']
rmalization)

activation_4 (Activation)      (None, 28, 28, 192)  0           ['batch_normalization_4[0][0]']

max_pooling2d_1 (MaxPooling2D)  (None, 13, 13, 192)  0         ['activation_4[0][0]']

conv2d_8 (Conv2D)              (None, 13, 13, 64)   12288       ['max_pooling2d_1[0][0]']

batch_normalization_8 (BatchNo  (None, 13, 13, 64)  192         ['conv2d_8[0][0]']
rmalization)

activation_8 (Activation)      (None, 13, 13, 64)   0           ['batch_normalization_8[0][0]']

conv2d_6 (Conv2D)              (None, 13, 13, 48)   9216        ['max_pooling2d_1[0][0]']

conv2d_9 (Conv2D)              (None, 13, 13, 96)   55296       ['activation_8[0][0]']

batch_normalization_6 (BatchNo  (None, 13, 13, 48)  144         ['conv2d_6[0][0]']
rmalization)

batch_normalization_9 (BatchNo  (None, 13, 13, 96)  288         ['conv2d_9[0][0]']
rmalization)

activation_6 (Activation)      (None, 13, 13, 48)   0           ['batch_normalization_6[0][0]']

activation_9 (Activation)      (None, 13, 13, 96)   0           ['batch_normalization_9[0][0]']

average_pooling2d (AveragePool  (None, 13, 13, 192)  0         ['max_pooling2d_1[0][0]']
ing2D)

conv2d_5 (Conv2D)              (None, 13, 13, 64)   12288       ['max_pooling2d_1[0][0]']

conv2d_7 (Conv2D)              (None, 13, 13, 64)   76800       ['activation_6[0][0]']

conv2d_10 (Conv2D)             (None, 13, 13, 96)   82944       ['activation_9[0][0]']

conv2d_11 (Conv2D)             (None, 13, 13, 32)   6144        ['average_pooling2d[0][0]']

batch_normalization_5 (BatchNo  (None, 13, 13, 64)  192         ['conv2d_5[0][0]']
rmalization)

batch_normalization_7 (BatchNo  (None, 13, 13, 64)  192         ['conv2d_7[0][0]']
rmalization)

batch_normalization_10 (BatchN  (None, 13, 13, 96)  288         ['conv2d_10[0][0]']
ormalization)

batch_normalization_11 (BatchN  (None, 13, 13, 32)  96          ['conv2d_11[0][0]']
ormalization)

activation_5 (Activation)      (None, 13, 13, 64)   0           ['batch_normalization_5[0][0]']

activation_7 (Activation)      (None, 13, 13, 64)   0           ['batch_normalization_7[0][0]']

activation_10 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_10[0][0]']

activation_11 (Activation)     (None, 13, 13, 32)   0           ['batch_normalization_11[0][0]']

mixed0 (Concatenate)           (None, 13, 13, 256)  0           ['activation_5[0][0]',
                                                                 'activation_7[0][0]',
```

```
                                                                 'activation_10[0][0]',
                                                                 'activation_11[0][0]']

conv2d_15 (Conv2D)             (None, 13, 13, 64)   16384       ['mixed0[0][0]']

batch_normalization_15 (BatchN  (None, 13, 13, 64)  192         ['conv2d_15[0][0]']
ormalization)

activation_15 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_15[0][0]']

conv2d_13 (Conv2D)             (None, 13, 13, 48)   12288       ['mixed0[0][0]']

conv2d_16 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_15[0][0]']

batch_normalization_13 (BatchN  (None, 13, 13, 48)  144         ['conv2d_13[0][0]']
ormalization)

batch_normalization_16 (BatchN  (None, 13, 13, 96)  288         ['conv2d_16[0][0]']
ormalization)

activation_13 (Activation)     (None, 13, 13, 48)   0           ['batch_normalization_13[0][0]']

activation_16 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_16[0][0]']

average_pooling2d_1 (AveragePo  (None, 13, 13, 256)  0         ['mixed0[0][0]']
oling2D)

conv2d_12 (Conv2D)             (None, 13, 13, 64)   16384       ['mixed0[0][0]']

conv2d_14 (Conv2D)             (None, 13, 13, 64)   76800       ['activation_13[0][0]']

conv2d_17 (Conv2D)             (None, 13, 13, 96)   82944       ['activation_16[0][0]']

conv2d_18 (Conv2D)             (None, 13, 13, 64)   16384       ['average_pooling2d_1[0][0]']

batch_normalization_12 (BatchN  (None, 13, 13, 64)  192         ['conv2d_12[0][0]']
ormalization)

batch_normalization_14 (BatchN  (None, 13, 13, 64)  192         ['conv2d_14[0][0]']
ormalization)

batch_normalization_17 (BatchN  (None, 13, 13, 96)  288         ['conv2d_17[0][0]']
ormalization)

batch_normalization_18 (BatchN  (None, 13, 13, 64)  192         ['conv2d_18[0][0]']
ormalization)

activation_12 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_12[0][0]']

activation_14 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_14[0][0]']

activation_17 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_17[0][0]']

activation_18 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_18[0][0]']

mixed1 (Concatenate)           (None, 13, 13, 288)  0           ['activation_12[0][0]',
                                                                 'activation_14[0][0]',
                                                                 'activation_17[0][0]',
                                                                 'activation_18[0][0]']

conv2d_22 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed1[0][0]']

batch_normalization_22 (BatchN  (None, 13, 13, 64)  192         ['conv2d_22[0][0]']
ormalization)

activation_22 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_22[0][0]']

conv2d_20 (Conv2D)             (None, 13, 13, 48)   13824       ['mixed1[0][0]']

conv2d_23 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_22[0][0]']

batch_normalization_20 (BatchN  (None, 13, 13, 48)  144         ['conv2d_20[0][0]']
ormalization)

batch_normalization_23 (BatchN  (None, 13, 13, 96)  288         ['conv2d_23[0][0]']
ormalization)

activation_20 (Activation)     (None, 13, 13, 48)   0           ['batch_normalization_20[0][0]']

activation_23 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_23[0][0]']

average_pooling2d_2 (AveragePo  (None, 13, 13, 288)  0         ['mixed1[0][0]']
oling2D)

conv2d_19 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed1[0][0]']

conv2d_21 (Conv2D)             (None, 13, 13, 64)   76800       ['activation_20[0][0]']

conv2d_24 (Conv2D)             (None, 13, 13, 96)   82944       ['activation_23[0][0]']
```

```
conv2d_25 (Conv2D)             (None, 13, 13, 64)   18432       ['average_pooling2d_2[0][0]']

batch_normalization_19 (BatchN  (None, 13, 13, 64)  192         ['conv2d_19[0][0]']
ormalization)

batch_normalization_21 (BatchN  (None, 13, 13, 64)  192         ['conv2d_21[0][0]']
ormalization)

batch_normalization_24 (BatchN  (None, 13, 13, 96)  288         ['conv2d_24[0][0]']
ormalization)

batch_normalization_25 (BatchN  (None, 13, 13, 64)  192         ['conv2d_25[0][0]']
ormalization)

activation_19 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_19[0][0]']

activation_21 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_21[0][0]']

activation_24 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_24[0][0]']

activation_25 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_25[0][0]']

mixed2 (Concatenate)           (None, 13, 13, 288)  0           ['activation_19[0][0]',
                                                                 'activation_21[0][0]',
                                                                 'activation_24[0][0]',
                                                                 'activation_25[0][0]']

conv2d_27 (Conv2D)             (None, 13, 13, 64)   18432       ['mixed2[0][0]']

batch_normalization_27 (BatchN  (None, 13, 13, 64)  192         ['conv2d_27[0][0]']
ormalization)

activation_27 (Activation)     (None, 13, 13, 64)   0           ['batch_normalization_27[0][0]']

conv2d_28 (Conv2D)             (None, 13, 13, 96)   55296       ['activation_27[0][0]']

batch_normalization_28 (BatchN  (None, 13, 13, 96)  288         ['conv2d_28[0][0]']
ormalization)

activation_28 (Activation)     (None, 13, 13, 96)   0           ['batch_normalization_28[0][0]']

conv2d_26 (Conv2D)             (None, 6, 6, 384)    995328      ['mixed2[0][0]']

conv2d_29 (Conv2D)             (None, 6, 6, 96)     82944       ['activation_28[0][0]']

batch_normalization_26 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_26[0][0]']
ormalization)

batch_normalization_29 (BatchN  (None, 6, 6, 96)    288         ['conv2d_29[0][0]']
ormalization)

activation_26 (Activation)     (None, 6, 6, 384)    0           ['batch_normalization_26[0][0]']

activation_29 (Activation)     (None, 6, 6, 96)     0           ['batch_normalization_29[0][0]']

max_pooling2d_2 (MaxPooling2D)  (None, 6, 6, 288)   0           ['mixed2[0][0]']

mixed3 (Concatenate)           (None, 6, 6, 768)    0           ['activation_26[0][0]',
                                                                 'activation_29[0][0]',
                                                                 'max_pooling2d_2[0][0]']

conv2d_34 (Conv2D)             (None, 6, 6, 128)    98304       ['mixed3[0][0]']

batch_normalization_34 (BatchN  (None, 6, 6, 128)   384         ['conv2d_34[0][0]']
ormalization)

activation_34 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_34[0][0]']

conv2d_35 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_34[0][0]']

batch_normalization_35 (BatchN  (None, 6, 6, 128)   384         ['conv2d_35[0][0]']
ormalization)

activation_35 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_35[0][0]']

conv2d_31 (Conv2D)             (None, 6, 6, 128)    98304       ['mixed3[0][0]']

conv2d_36 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_35[0][0]']

batch_normalization_31 (BatchN  (None, 6, 6, 128)   384         ['conv2d_31[0][0]']
ormalization)

batch_normalization_36 (BatchN  (None, 6, 6, 128)   384         ['conv2d_36[0][0]']
ormalization)

activation_31 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_31[0][0]']

activation_36 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_36[0][0]']
```

```
conv2d_32 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_31[0][0]']

conv2d_37 (Conv2D)             (None, 6, 6, 128)    114688      ['activation_36[0][0]']

batch_normalization_32 (BatchN  (None, 6, 6, 128)   384         ['conv2d_32[0][0]']
ormalization)

batch_normalization_37 (BatchN  (None, 6, 6, 128)   384         ['conv2d_37[0][0]']
ormalization)

activation_32 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_32[0][0]']

activation_37 (Activation)     (None, 6, 6, 128)    0           ['batch_normalization_37[0][0]']

average_pooling2d_3 (AveragePo  (None, 6, 6, 768)   0           ['mixed3[0][0]']
oling2D)

conv2d_30 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed3[0][0]']

conv2d_33 (Conv2D)             (None, 6, 6, 192)    172032      ['activation_32[0][0]']

conv2d_38 (Conv2D)             (None, 6, 6, 192)    172032      ['activation_37[0][0]']

conv2d_39 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_3[0][0]']

batch_normalization_30 (BatchN  (None, 6, 6, 192)   576         ['conv2d_30[0][0]']
ormalization)

batch_normalization_33 (BatchN  (None, 6, 6, 192)   576         ['conv2d_33[0][0]']
ormalization)

batch_normalization_38 (BatchN  (None, 6, 6, 192)   576         ['conv2d_38[0][0]']
ormalization)

batch_normalization_39 (BatchN  (None, 6, 6, 192)   576         ['conv2d_39[0][0]']
ormalization)

activation_30 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_30[0][0]']

activation_33 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_33[0][0]']

activation_38 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_38[0][0]']

activation_39 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_39[0][0]']

mixed4 (Concatenate)           (None, 6, 6, 768)    0           ['activation_30[0][0]',
                                                                 'activation_33[0][0]',
                                                                 'activation_38[0][0]',
                                                                 'activation_39[0][0]']

conv2d_44 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed4[0][0]']

batch_normalization_44 (BatchN  (None, 6, 6, 160)   480         ['conv2d_44[0][0]']
ormalization)

activation_44 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_44[0][0]']

conv2d_45 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_44[0][0]']

batch_normalization_45 (BatchN  (None, 6, 6, 160)   480         ['conv2d_45[0][0]']
ormalization)

activation_45 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_45[0][0]']

conv2d_41 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed4[0][0]']

conv2d_46 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_45[0][0]']

batch_normalization_41 (BatchN  (None, 6, 6, 160)   480         ['conv2d_41[0][0]']
ormalization)

batch_normalization_46 (BatchN  (None, 6, 6, 160)   480         ['conv2d_46[0][0]']
ormalization)

activation_41 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_41[0][0]']

activation_46 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_46[0][0]']

conv2d_42 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_41[0][0]']

conv2d_47 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_46[0][0]']

batch_normalization_42 (BatchN  (None, 6, 6, 160)   480         ['conv2d_42[0][0]']
ormalization)

batch_normalization_47 (BatchN  (None, 6, 6, 160)   480         ['conv2d_47[0][0]']
ormalization)
```

```
activation_42 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_42[0][0]']

activation_47 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_47[0][0]']

average_pooling2d_4 (AveragePo  (None, 6, 6, 768)   0           ['mixed4[0][0]']
oling2D)

conv2d_40 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed4[0][0]']

conv2d_43 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_42[0][0]']

conv2d_48 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_47[0][0]']

conv2d_49 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_4[0][0]']

batch_normalization_40 (BatchN  (None, 6, 6, 192)   576         ['conv2d_40[0][0]']
ormalization)

batch_normalization_43 (BatchN  (None, 6, 6, 192)   576         ['conv2d_43[0][0]']
ormalization)

batch_normalization_48 (BatchN  (None, 6, 6, 192)   576         ['conv2d_48[0][0]']
ormalization)

batch_normalization_49 (BatchN  (None, 6, 6, 192)   576         ['conv2d_49[0][0]']
ormalization)

activation_40 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_40[0][0]']

activation_43 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_43[0][0]']

activation_48 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_48[0][0]']

activation_49 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_49[0][0]']

mixed5 (Concatenate)           (None, 6, 6, 768)    0           ['activation_40[0][0]',
                                                                 'activation_43[0][0]',
                                                                 'activation_48[0][0]',
                                                                 'activation_49[0][0]']

conv2d_54 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed5[0][0]']

batch_normalization_54 (BatchN  (None, 6, 6, 160)   480         ['conv2d_54[0][0]']
ormalization)

activation_54 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_54[0][0]']

conv2d_55 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_54[0][0]']

batch_normalization_55 (BatchN  (None, 6, 6, 160)   480         ['conv2d_55[0][0]']
ormalization)

activation_55 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_55[0][0]']

conv2d_51 (Conv2D)             (None, 6, 6, 160)    122880      ['mixed5[0][0]']

conv2d_56 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_55[0][0]']

batch_normalization_51 (BatchN  (None, 6, 6, 160)   480         ['conv2d_51[0][0]']
ormalization)

batch_normalization_56 (BatchN  (None, 6, 6, 160)   480         ['conv2d_56[0][0]']
ormalization)

activation_51 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_51[0][0]']

activation_56 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_56[0][0]']

conv2d_52 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_51[0][0]']

conv2d_57 (Conv2D)             (None, 6, 6, 160)    179200      ['activation_56[0][0]']

batch_normalization_52 (BatchN  (None, 6, 6, 160)   480         ['conv2d_52[0][0]']
ormalization)

batch_normalization_57 (BatchN  (None, 6, 6, 160)   480         ['conv2d_57[0][0]']
ormalization)

activation_52 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_52[0][0]']

activation_57 (Activation)     (None, 6, 6, 160)    0           ['batch_normalization_57[0][0]']

average_pooling2d_5 (AveragePo  (None, 6, 6, 768)   0           ['mixed5[0][0]']
oling2D)

conv2d_50 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed5[0][0]']

conv2d_53 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_52[0][0]']
```

```
conv2d_58 (Conv2D)             (None, 6, 6, 192)    215040      ['activation_57[0][0]']

conv2d_59 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_5[0][0]']

batch_normalization_50 (BatchN  (None, 6, 6, 192)   576         ['conv2d_50[0][0]']
ormalization)

batch_normalization_53 (BatchN  (None, 6, 6, 192)   576         ['conv2d_53[0][0]']
ormalization)

batch_normalization_58 (BatchN  (None, 6, 6, 192)   576         ['conv2d_58[0][0]']
ormalization)

batch_normalization_59 (BatchN  (None, 6, 6, 192)   576         ['conv2d_59[0][0]']
ormalization)

activation_50 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_50[0][0]']

activation_53 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_53[0][0]']

activation_58 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_58[0][0]']

activation_59 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_59[0][0]']

mixed6 (Concatenate)           (None, 6, 6, 768)    0           ['activation_50[0][0]',
                                                                 'activation_53[0][0]',
                                                                 'activation_58[0][0]',
                                                                 'activation_59[0][0]']

conv2d_64 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

batch_normalization_64 (BatchN  (None, 6, 6, 192)   576         ['conv2d_64[0][0]']
ormalization)

activation_64 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_64[0][0]']

conv2d_65 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_64[0][0]']

batch_normalization_65 (BatchN  (None, 6, 6, 192)   576         ['conv2d_65[0][0]']
ormalization)

activation_65 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_65[0][0]']

conv2d_61 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

conv2d_66 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_65[0][0]']

batch_normalization_61 (BatchN  (None, 6, 6, 192)   576         ['conv2d_61[0][0]']
ormalization)

batch_normalization_66 (BatchN  (None, 6, 6, 192)   576         ['conv2d_66[0][0]']
ormalization)

activation_61 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_61[0][0]']

activation_66 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_66[0][0]']

conv2d_62 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_61[0][0]']

conv2d_67 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_66[0][0]']

batch_normalization_62 (BatchN  (None, 6, 6, 192)   576         ['conv2d_62[0][0]']
ormalization)

batch_normalization_67 (BatchN  (None, 6, 6, 192)   576         ['conv2d_67[0][0]']
ormalization)

activation_62 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_62[0][0]']

activation_67 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_67[0][0]']

average_pooling2d_6 (AveragePo  (None, 6, 6, 768)   0           ['mixed6[0][0]']
oling2D)

conv2d_60 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed6[0][0]']

conv2d_63 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_62[0][0]']

conv2d_68 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_67[0][0]']

conv2d_69 (Conv2D)             (None, 6, 6, 192)    147456      ['average_pooling2d_6[0][0]']

batch_normalization_60 (BatchN  (None, 6, 6, 192)   576         ['conv2d_60[0][0]']
ormalization)

batch_normalization_63 (BatchN  (None, 6, 6, 192)   576         ['conv2d_63[0][0]']
ormalization)

batch_normalization_68 (BatchN  (None, 6, 6, 192)   576         ['conv2d_68[0][0]']
```

```
ormalization)

batch_normalization_69 (BatchN  (None, 6, 6, 192)   576         ['conv2d_69[0][0]']
ormalization)

activation_60 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_60[0][0]']

activation_63 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_63[0][0]']

activation_68 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_68[0][0]']

activation_69 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_69[0][0]']

mixed7 (Concatenate)           (None, 6, 6, 768)    0           ['activation_60[0][0]',
                                                                 'activation_63[0][0]',
                                                                 'activation_68[0][0]',
                                                                 'activation_69[0][0]']

conv2d_72 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed7[0][0]']

batch_normalization_72 (BatchN  (None, 6, 6, 192)   576         ['conv2d_72[0][0]']
ormalization)

activation_72 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_72[0][0]']

conv2d_73 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_72[0][0]']

batch_normalization_73 (BatchN  (None, 6, 6, 192)   576         ['conv2d_73[0][0]']
ormalization)

activation_73 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_73[0][0]']

conv2d_70 (Conv2D)             (None, 6, 6, 192)    147456      ['mixed7[0][0]']

conv2d_74 (Conv2D)             (None, 6, 6, 192)    258048      ['activation_73[0][0]']

batch_normalization_70 (BatchN  (None, 6, 6, 192)   576         ['conv2d_70[0][0]']
ormalization)

batch_normalization_74 (BatchN  (None, 6, 6, 192)   576         ['conv2d_74[0][0]']
ormalization)

activation_70 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_70[0][0]']

activation_74 (Activation)     (None, 6, 6, 192)    0           ['batch_normalization_74[0][0]']

conv2d_71 (Conv2D)             (None, 2, 2, 320)    552960      ['activation_70[0][0]']

conv2d_75 (Conv2D)             (None, 2, 2, 192)    331776      ['activation_74[0][0]']

batch_normalization_71 (BatchN  (None, 2, 2, 320)   960         ['conv2d_71[0][0]']
ormalization)

batch_normalization_75 (BatchN  (None, 2, 2, 192)   576         ['conv2d_75[0][0]']
ormalization)

activation_71 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_71[0][0]']

activation_75 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_75[0][0]']

max_pooling2d_3 (MaxPooling2D)  (None, 2, 2, 768)   0           ['mixed7[0][0]']

mixed8 (Concatenate)           (None, 2, 2, 1280)   0           ['activation_71[0][0]',
                                                                 'activation_75[0][0]',
                                                                 'max_pooling2d_3[0][0]']

conv2d_80 (Conv2D)             (None, 2, 2, 448)    573440      ['mixed8[0][0]']

batch_normalization_80 (BatchN  (None, 2, 2, 448)   1344        ['conv2d_80[0][0]']
ormalization)

activation_80 (Activation)     (None, 2, 2, 448)    0           ['batch_normalization_80[0][0]']

conv2d_77 (Conv2D)             (None, 2, 2, 384)    491520      ['mixed8[0][0]']

conv2d_81 (Conv2D)             (None, 2, 2, 384)    1548288     ['activation_80[0][0]']

batch_normalization_77 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_77[0][0]']
ormalization)

batch_normalization_81 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_81[0][0]']
ormalization)

activation_77 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_77[0][0]']

activation_81 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_81[0][0]']

conv2d_78 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_77[0][0]']
```

```
conv2d_79 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_77[0][0]']

conv2d_82 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_81[0][0]']

conv2d_83 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_81[0][0]']

average_pooling2d_7 (AveragePo  (None, 2, 2, 1280)  0           ['mixed8[0][0]']
oling2D)

conv2d_76 (Conv2D)             (None, 2, 2, 320)    409600      ['mixed8[0][0]']

batch_normalization_78 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_78[0][0]']
ormalization)

batch_normalization_79 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_79[0][0]']
ormalization)

batch_normalization_82 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_82[0][0]']
ormalization)

batch_normalization_83 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_83[0][0]']
ormalization)

conv2d_84 (Conv2D)             (None, 2, 2, 192)    245760      ['average_pooling2d_7[0][0]']

batch_normalization_76 (BatchN  (None, 2, 2, 320)   960         ['conv2d_76[0][0]']
ormalization)

activation_78 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_78[0][0]']

activation_79 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_79[0][0]']

activation_82 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_82[0][0]']

activation_83 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_83[0][0]']

batch_normalization_84 (BatchN  (None, 2, 2, 192)   576         ['conv2d_84[0][0]']
ormalization)

activation_76 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_76[0][0]']

mixed9_0 (Concatenate)         (None, 2, 2, 768)    0           ['activation_78[0][0]',
                                                                 'activation_79[0][0]']

concatenate (Concatenate)      (None, 2, 2, 768)    0           ['activation_82[0][0]',
                                                                 'activation_83[0][0]']

activation_84 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_84[0][0]']

mixed9 (Concatenate)           (None, 2, 2, 2048)   0           ['activation_76[0][0]',
                                                                 'mixed9_0[0][0]',
                                                                 'concatenate[0][0]',
                                                                 'activation_84[0][0]']

conv2d_89 (Conv2D)             (None, 2, 2, 448)    917504      ['mixed9[0][0]']

batch_normalization_89 (BatchN  (None, 2, 2, 448)   1344        ['conv2d_89[0][0]']
ormalization)

activation_89 (Activation)     (None, 2, 2, 448)    0           ['batch_normalization_89[0][0]']

conv2d_86 (Conv2D)             (None, 2, 2, 384)    786432      ['mixed9[0][0]']

conv2d_90 (Conv2D)             (None, 2, 2, 384)    1548288     ['activation_89[0][0]']

batch_normalization_86 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_86[0][0]']
ormalization)

batch_normalization_90 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_90[0][0]']
ormalization)

activation_86 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_86[0][0]']

activation_90 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_90[0][0]']

conv2d_87 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_86[0][0]']

conv2d_88 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_86[0][0]']

conv2d_91 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_90[0][0]']

conv2d_92 (Conv2D)             (None, 2, 2, 384)    442368      ['activation_90[0][0]']

average_pooling2d_8 (AveragePo  (None, 2, 2, 2048)  0           ['mixed9[0][0]']
oling2D)

conv2d_85 (Conv2D)             (None, 2, 2, 320)    655360      ['mixed9[0][0]']

batch_normalization_87 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_87[0][0]']
```

```
ormalization)

batch_normalization_88 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_88[0][0]']
ormalization)

batch_normalization_91 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_91[0][0]']
ormalization)

batch_normalization_92 (BatchN  (None, 2, 2, 384)   1152        ['conv2d_92[0][0]']
ormalization)

conv2d_93 (Conv2D)             (None, 2, 2, 192)    393216      ['average_pooling2d_8[0][0]']

batch_normalization_85 (BatchN  (None, 2, 2, 320)   960         ['conv2d_85[0][0]']
ormalization)

activation_87 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_87[0][0]']

activation_88 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_88[0][0]']

activation_91 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_91[0][0]']

activation_92 (Activation)     (None, 2, 2, 384)    0           ['batch_normalization_92[0][0]']

batch_normalization_93 (BatchN  (None, 2, 2, 192)   576         ['conv2d_93[0][0]']
ormalization)

activation_85 (Activation)     (None, 2, 2, 320)    0           ['batch_normalization_85[0][0]']

mixed9_1 (Concatenate)         (None, 2, 2, 768)    0           ['activation_87[0][0]',
                                                                 'activation_88[0][0]']

concatenate_1 (Concatenate)    (None, 2, 2, 768)    0           ['activation_91[0][0]',
                                                                 'activation_92[0][0]']

activation_93 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_93[0][0]']

mixed10 (Concatenate)          (None, 2, 2, 2048)   0           ['activation_85[0][0]',
                                                                 'mixed9_1[0][0]',
                                                                 'concatenate_1[0][0]',
                                                                 'activation_93[0][0]']

==================================================================================================
Total params: 21,802,784
Trainable params: 0
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

Do the `Fine-tuning` with respect to Inception model

In [ ]:

```python
class InceptionNetException(Exception):
  def __init__(self, message):
    return message.title()

def Inception_fine_tuning(Inception_model = None, activate = None):
  if activate == 'YES':
    #### Create a sequential model ####
    model_InceptionNet = Sequential()

    #### Add the ResNet model to this sequential model ####
    model_InceptionNet.add(Inception_model)

    #### Do the Flatten operation ####
    model_InceptionNet.add(Flatten())

    #### Add the user defined - fully connected layer with respect to problem description ####

    model_InceptionNet.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

    #### Use the Dropout layer with the ratio = 0.5 ####

    model_InceptionNet.add(Dropout(rate = 0.3))

    #### Add another connected layer with neurons 128 ####

    model_InceptionNet.add(Dense(units = 64, activation = 'relu', kernel_initializer = 'he_normal'))

    #### Use the Dropout layer with the ratio = 0.6 ####

    model_InceptionNet.add(Dropout(rate = 0.4))

    #### Add the output layer ####
    model_InceptionNet.add(Dense(units = 4, activation = 'softmax'))

    #### Compile the model and check the performace ####
    model_InceptionNet.compile(optimizer = Adam(learning_rate = 0.0001), loss = SparseCategoricalCrossentropy()

  else:
```

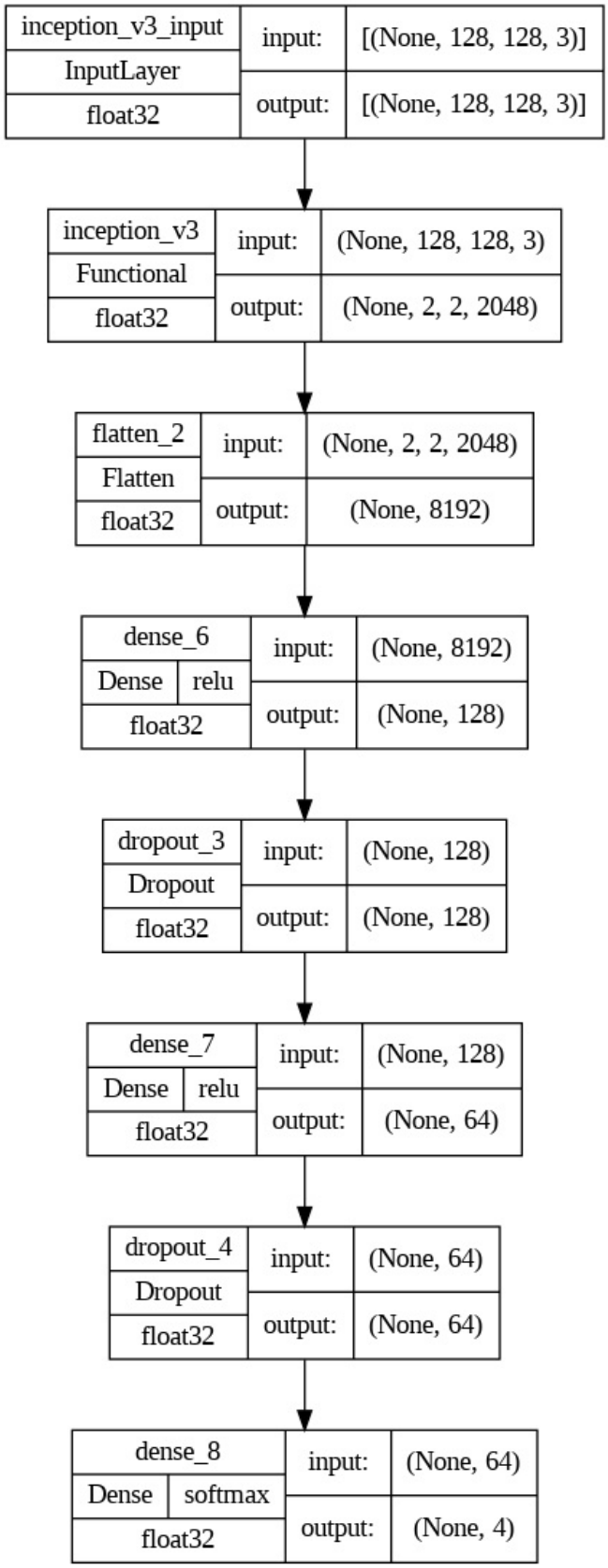
```
    raise Exception('InceptionNet cannot be accessible')

  return model_InceptionNet

try:
  model_InceptionNet = Inception_fine_tuning(Inception_model = Inception_model, activate = 'YES')
except InceptionNetException as e:
  print('The exception is {}'.format(e))
except Exception as e:
  print('The exception is {}'.format(e))
else:
  model_InceptionNet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2, 2, 2048)        21802784

 flatten_2 (Flatten)         (None, 8192)              0

 dense_6 (Dense)             (None, 128)               1048704

 dropout_3 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 64)                8256

 dropout_4 (Dropout)         (None, 64)                0

 dense_8 (Dense)             (None, 4)                 260

=================================================================
Total params: 22,860,004
Trainable params: 1,057,220
Non-trainable params: 21,802,784
_________________________________________________________________
```

`plot` the InceptionNet model as a graph

In [ ]:
```
plot_model(model = model_InceptionNet,\
           show_shapes = True,\
           show_dtype = True,\
           show_layer_names = True,\
           show_layer_activations = True)
```

Out[ ]:



In [ ]:

```
def training(x = None, y = None, epochs = None, batch_size = None):
  if (len(x) == 0 or len(y) == 0):
    raise listEmptyException('List is empty in model InceptionNet'.title())
  else:
    history = model_InceptionNet.fit(x = x,\
                                y = y,\
                                epochs = epochs,\
                                batch_size = batch_size,\
                                verbose = 1,\
                                validation_data = (X_test, y_test))
  return history, model_InceptionNet

try:
  history, model_InceptionNet = training(x = X_train,\
                                y = y_train,\
                                epochs = 20,\
                                batch_size = 128)
except listEmptyException as e:
  print('The exception is {}'.format(e))
```

```
except Exception as e:
  print('The exception is {}'.format(e))
else:
  print('Completed !')
```

```
Epoch 1/20
60/60 [==============================] - 15s 141ms/step - loss: 1.4177 - accuracy: 0.3844 - val_loss: 1.0277 -
val_accuracy: 0.5664
Epoch 2/20
60/60 [==============================] - 6s 99ms/step - loss: 1.0593 - accuracy: 0.5130 - val_loss: 0.8440 - va
l_accuracy: 0.6469
Epoch 3/20
60/60 [==============================] - 6s 101ms/step - loss: 0.9613 - accuracy: 0.5660 - val_loss: 0.7635 - v
al_accuracy: 0.6742
Epoch 4/20
60/60 [==============================] - 6s 101ms/step - loss: 0.8707 - accuracy: 0.6094 - val_loss: 0.7225 - v
al_accuracy: 0.6867
Epoch 5/20
60/60 [==============================] - 6s 102ms/step - loss: 0.8088 - accuracy: 0.6374 - val_loss: 0.6698 - v
al_accuracy: 0.7105
Epoch 6/20
60/60 [==============================] - 6s 102ms/step - loss: 0.7758 - accuracy: 0.6520 - val_loss: 0.6383 - v
al_accuracy: 0.7266
Epoch 7/20
60/60 [==============================] - 6s 101ms/step - loss: 0.7342 - accuracy: 0.6759 - val_loss: 0.6085 - v
al_accuracy: 0.7203
Epoch 8/20
60/60 [==============================] - 6s 100ms/step - loss: 0.7062 - accuracy: 0.6807 - val_loss: 0.5960 - v
al_accuracy: 0.7234
Epoch 9/20
60/60 [==============================] - 6s 99ms/step - loss: 0.6760 - accuracy: 0.7090 - val_loss: 0.5743 - va
l_accuracy: 0.7508
Epoch 10/20
60/60 [==============================] - 6s 98ms/step - loss: 0.6484 - accuracy: 0.7135 - val_loss: 0.5504 - va
l_accuracy: 0.7461
Epoch 11/20
60/60 [==============================] - 6s 99ms/step - loss: 0.6354 - accuracy: 0.7201 - val_loss: 0.5575 - va
l_accuracy: 0.7484
Epoch 12/20
60/60 [==============================] - 6s 98ms/step - loss: 0.6068 - accuracy: 0.7327 - val_loss: 0.5348 - va
l_accuracy: 0.7621
Epoch 13/20
60/60 [==============================] - 6s 99ms/step - loss: 0.6021 - accuracy: 0.7391 - val_loss: 0.5362 - va
l_accuracy: 0.7492
Epoch 14/20
60/60 [==============================] - 6s 99ms/step - loss: 0.5880 - accuracy: 0.7443 - val_loss: 0.5209 - va
l_accuracy: 0.7652
Epoch 15/20
60/60 [==============================] - 6s 98ms/step - loss: 0.5657 - accuracy: 0.7556 - val_loss: 0.5144 - va
l_accuracy: 0.7777
Epoch 16/20
60/60 [==============================] - 6s 99ms/step - loss: 0.5614 - accuracy: 0.7542 - val_loss: 0.5126 - va
l_accuracy: 0.7668
Epoch 17/20
60/60 [==============================] - 6s 100ms/step - loss: 0.5489 - accuracy: 0.7579 - val_loss: 0.4790 - v
al_accuracy: 0.7781
Epoch 18/20
60/60 [==============================] - 6s 100ms/step - loss: 0.5240 - accuracy: 0.7757 - val_loss: 0.4781 - v
al_accuracy: 0.7812
Epoch 19/20
60/60 [==============================] - 6s 100ms/step - loss: 0.5195 - accuracy: 0.7723 - val_loss: 0.4865 - v
al_accuracy: 0.7781
Epoch 20/20
60/60 [==============================] - 6s 100ms/step - loss: 0.5097 - accuracy: 0.7732 - val_loss: 0.4893 - v
al_accuracy: 0.7738
Completed !
```

Check the performance `InceptionNet`

In [ ]:
```
print('The training performace of InceptionNet model is given below.\n\n'.title())

predicted_ = model_InceptionNet.predict(X_train)
predicted_ = np.argmax(predicted_, axis = 1)

print('The accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_train),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_train, average = 'm
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_train, average = 'macr
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_train, average = 'macro'),

print('The testing performace of ResNet50 model is given below.\n\n'.title())

predicted_ = model_InceptionNet.predict(X_test)
predicted_ = np.argmax(predicted_, axis = 1)

print('\nThe accuracy of this Neural Network is = {} '.format(accuracy_score(predicted_, y_test),'\n'))
print('The precision of this Neural Network is  = {} '.format(precision_score(predicted_, y_test, average = 'ma
print('The reacll of this Neural Network is     = {} '.format(recall_score(predicted_, y_test, average = 'macro
print('The f1_score of this Neural Network is   = {} '.format(f1_score(predicted_, y_test, average = 'macro'),'
```

```
The Training Performace Of Inceptionnet Model Is Given Below.


240/240 [==============================] - 8s 23ms/step
The accuracy of this Neural Network is = 0.817578125
The precision of this Neural Network is  = 0.8169858479244576
The reacll of this Neural Network is     = 0.8257546500319926
The f1_score of this Neural Network is   = 0.8143686041115529
The Testing Performace Of Resnet50 Model Is Given Below.


80/80 [==============================] - 2s 22ms/step

The accuracy of this Neural Network is = 0.773828125
The precision of this Neural Network is  = 0.7754969034462695
The reacll of this Neural Network is     = 0.7784407726427209
The f1_score of this Neural Network is   = 0.7704376794826641
```

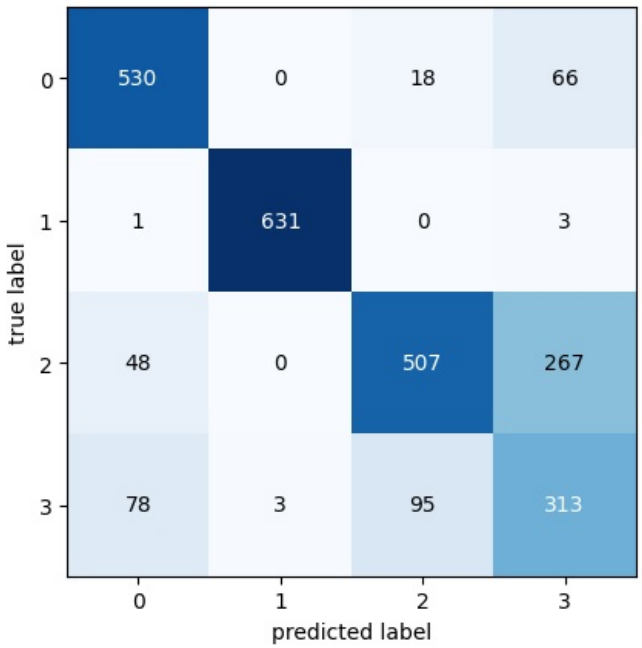
Show the `Classification` report to this model

In [ ]:
```
print('The classification report of this testing model is given below.\n'.capitalize())
print(classification_report(predicted_, y_test))
```

```
The classification report of this testing model is given below.

              precision    recall  f1-score   support

           0       0.81      0.86      0.83       614
           1       1.00      0.99      0.99       635
           2       0.82      0.62      0.70       822
           3       0.48      0.64      0.55       489

    accuracy                           0.77      2560
   macro avg       0.78      0.78      0.77      2560
weighted avg       0.80      0.77      0.78      2560
```

Plot the `Confusion` Matrix

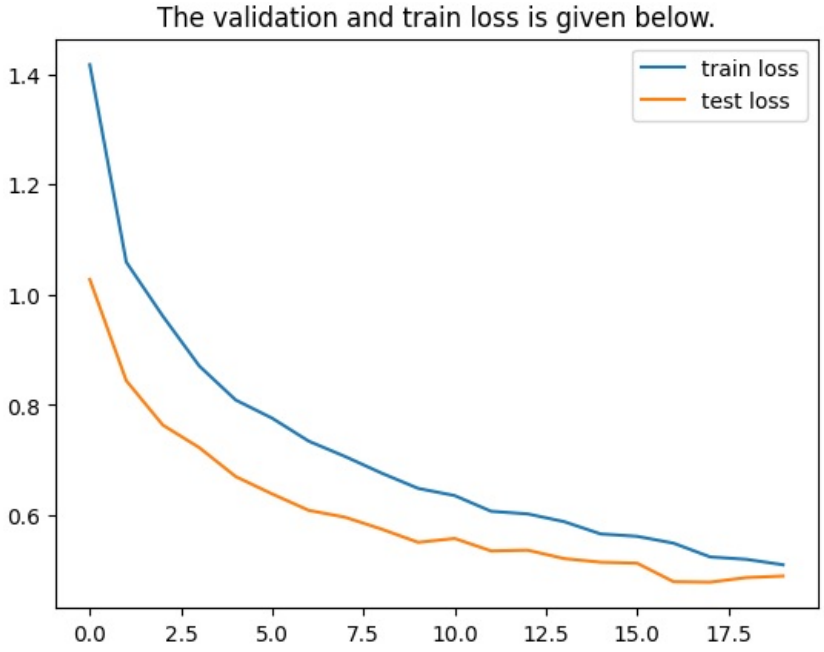
In [ ]:
```
#### Plot the confusion matrix ####
confusion_mat = confusion_matrix(predicted_, y_test)

fig, ax = plot_confusion_matrix(conf_mat = confusion_mat)

plt.show()
```



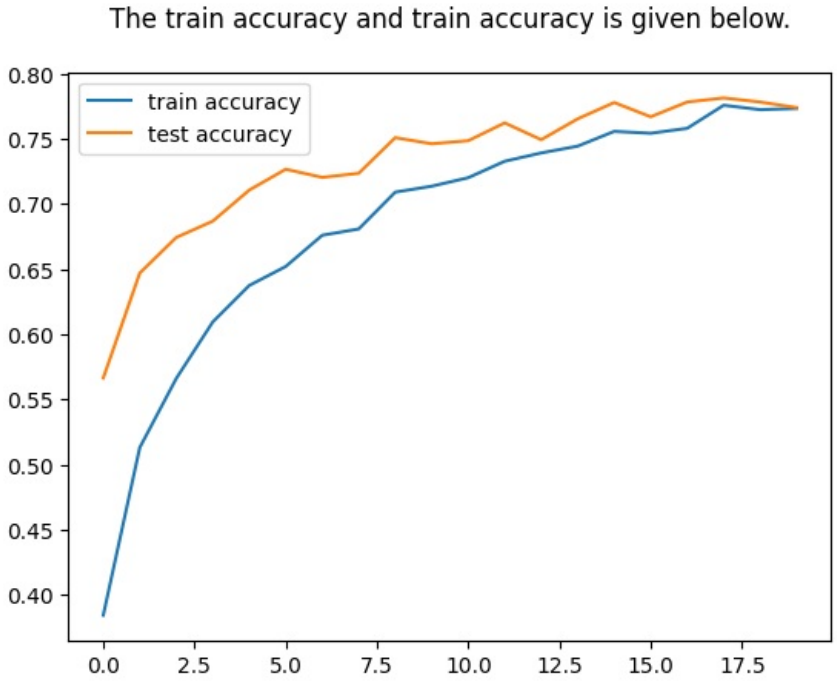
Plot `train and test` loss and `train and test` accuracy

In [ ]:
```
##### Plot the validation loss and train loss #####
plt.title('The validation and train loss is given below.')
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'test loss')
plt.legend()
plt.show()

print('*'*120,'\n')
```

```
plt.title('The train accuracy and train accuracy is given below.\n')
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.plot(history.history['val_accuracy'], label = 'test accuracy')
plt.legend()
plt.show()
```



****



Do the `Cross Validation` with `K` - 3 for the `ResNet50` - beacuse amomg all transfer learning it provided good performance with respect to dataset

- I use K = 3, due to memory issue. If I increase then I am facing the memory problem

In [ ]:
```
KFold_ = KFold(n_splits = 3, random_state = 42, shuffle = True)

accuracy, precision, recall, f1, count = [], [], [], [], 1

for train_index, test_index in KFold_.split(X_train_normalised):

  print('# of Cross Validation is {} is running'.title().format(count),'\n\n')
  X_train, X_test = X_train_normalised[train_index], X_train_normalised[test_index]
  y_train_, y_test_ = y_train[train_index], y_train[test_index]

  #### Create a sequential model ####
  model_VGG16 = Sequential()

  #### Add the ResNet model to this sequential model ####
  model_ResNet.add(ResNet_model)
```

```
  #### Do the Flatten operation ####
  model_ResNet.add(Flatten())

  #### Add the user defined - fully connected layer with respect to problem description ####

  model_ResNet.add(Dense(units = 512, activation = 'relu', kernel_initializer = 'he_normal', kernel_regularizer

  #### Add another connected layer with neurons 128 ####

  model_ResNet.add(Dense(units = 128, activation = 'relu', kernel_initializer = 'he_normal'))

  #### Use the Dropout layer with the ratio = 0.6 ####

  model_ResNet.add(Dropout(rate = 0.4))

  #### Add the output layer ####
  model_ResNet.add(Dense(units = 4, activation = 'softmax'))

  #### Compile the model and check the performace ####
  model_ResNet.compile(optimizer = Adam(learning_rate = 0.0001), loss = SparseCategoricalCrossentropy(), metric


  history = model_ResNet.fit(x = X_train, y = y_train_, epochs = 20, batch_size = 64, validation_data = (X_test

  predicted_ = model_ResNet.predict(X_test)
  predicted_ = np.argmax(predicted_, axis = 1)

  print('The accuracy of this Neural Network is  = {} '.format(accuracy_score(predicted_, y_test_),'\n'))
  print('The precision of this Neural Network is = {} '.format(precision_score(predicted_, y_test_, average = '
  print('The reacll of this Neural Network is    = {} '.format(recall_score(predicted_, y_test_, average = 'mac
  print('The f1_score of this Neural Network is  = {} '.format(f1_score(predicted_, y_test_, average = 'macro')

  accuracy.append(accuracy_score(predicted_, y_test_))
  precision.append(precision_score(predicted_, y_test_, average = 'macro'))
  recall.append(recall_score(predicted_, y_test_, average = 'macro'))
  f1.append(f1_score(predicted_, y_test_, average = 'macro'))

  count = count + 1
```

```
# Of Cross Validation Is 1 Is Running


Epoch 1/20
107/107 [==============================] - 31s 167ms/step - loss: 7.8754 - accuracy: 0.5259 - val_loss: 5.0352
- val_accuracy: 0.6807
Epoch 2/20
107/107 [==============================] - 13s 122ms/step - loss: 3.7564 - accuracy: 0.6840 - val_loss: 2.7418
- val_accuracy: 0.7299
Epoch 3/20
107/107 [==============================] - 13s 124ms/step - loss: 2.2932 - accuracy: 0.7240 - val_loss: 1.9030
- val_accuracy: 0.7147
Epoch 4/20
107/107 [==============================] - 13s 125ms/step - loss: 1.7022 - accuracy: 0.7430 - val_loss: 1.4931
- val_accuracy: 0.7730
Epoch 5/20
107/107 [==============================] - 13s 122ms/step - loss: 1.4242 - accuracy: 0.7684 - val_loss: 1.3121
- val_accuracy: 0.7800
Epoch 6/20
107/107 [==============================] - 13s 121ms/step - loss: 1.2660 - accuracy: 0.7800 - val_loss: 1.2060
- val_accuracy: 0.7789
Epoch 7/20
107/107 [==============================] - 13s 121ms/step - loss: 1.1645 - accuracy: 0.7826 - val_loss: 1.0993
- val_accuracy: 0.8064
Epoch 8/20
107/107 [==============================] - 13s 122ms/step - loss: 1.0791 - accuracy: 0.7936 - val_loss: 1.0447
- val_accuracy: 0.7961
Epoch 9/20
107/107 [==============================] - 13s 122ms/step - loss: 1.0136 - accuracy: 0.8050 - val_loss: 0.9727
- val_accuracy: 0.8087
Epoch 10/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9522 - accuracy: 0.8109 - val_loss: 0.9342
- val_accuracy: 0.8090
Epoch 11/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9111 - accuracy: 0.8198 - val_loss: 0.8979
- val_accuracy: 0.8052
Epoch 12/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8687 - accuracy: 0.8241 - val_loss: 0.8432
- val_accuracy: 0.8316
Epoch 13/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8321 - accuracy: 0.8301 - val_loss: 0.8333
- val_accuracy: 0.8122
Epoch 14/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7968 - accuracy: 0.8355 - val_loss: 0.7866
- val_accuracy: 0.8307
Epoch 15/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7534 - accuracy: 0.8443 - val_loss: 0.7512
- val_accuracy: 0.8392
Epoch 16/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7452 - accuracy: 0.8361 - val_loss: 0.7495
```

```
- val_accuracy: 0.8257
Epoch 17/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7242 - accuracy: 0.8440 - val_loss: 0.7130
- val_accuracy: 0.8380
Epoch 18/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7141 - accuracy: 0.8394 - val_loss: 0.7643
- val_accuracy: 0.8029
Epoch 19/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6769 - accuracy: 0.8501 - val_loss: 0.6731
- val_accuracy: 0.8430
Epoch 20/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6568 - accuracy: 0.8509 - val_loss: 0.7029
- val_accuracy: 0.8231
107/107 [==============================] - 6s 40ms/step
The accuracy of this Neural Network is  = 0.8230814294083187
The precision of this Neural Network is = 0.8254231498076718
The reacll of this Neural Network is    = 0.8499167276636506
The f1_score of this Neural Network is  = 0.8189446657132637
# Of Cross Validation Is 2 Is Running


Epoch 1/20
107/107 [==============================] - 20s 171ms/step - loss: 7.8670 - accuracy: 0.5276 - val_loss: 5.0594
- val_accuracy: 0.6959
Epoch 2/20
107/107 [==============================] - 13s 123ms/step - loss: 3.7834 - accuracy: 0.6874 - val_loss: 2.7573
- val_accuracy: 0.7375
Epoch 3/20
107/107 [==============================] - 13s 122ms/step - loss: 2.3020 - accuracy: 0.7227 - val_loss: 1.8937
- val_accuracy: 0.7507
Epoch 4/20
107/107 [==============================] - 13s 122ms/step - loss: 1.7131 - accuracy: 0.7529 - val_loss: 1.5145
- val_accuracy: 0.7741
Epoch 5/20
107/107 [==============================] - 13s 122ms/step - loss: 1.4378 - accuracy: 0.7668 - val_loss: 1.3103
- val_accuracy: 0.7902
Epoch 6/20
107/107 [==============================] - 13s 122ms/step - loss: 1.2672 - accuracy: 0.7837 - val_loss: 1.1965
- val_accuracy: 0.7873
Epoch 7/20
107/107 [==============================] - 13s 122ms/step - loss: 1.1651 - accuracy: 0.7885 - val_loss: 1.1004
- val_accuracy: 0.8046
Epoch 8/20
107/107 [==============================] - 13s 122ms/step - loss: 1.0800 - accuracy: 0.8036 - val_loss: 1.0295
- val_accuracy: 0.8192
Epoch 9/20
107/107 [==============================] - 13s 122ms/step - loss: 1.0101 - accuracy: 0.8102 - val_loss: 0.9709
- val_accuracy: 0.8148
Epoch 10/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9454 - accuracy: 0.8214 - val_loss: 0.9213
- val_accuracy: 0.8131
Epoch 11/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8992 - accuracy: 0.8229 - val_loss: 0.8884
- val_accuracy: 0.8192
Epoch 12/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8610 - accuracy: 0.8223 - val_loss: 0.8990
- val_accuracy: 0.7931
Epoch 13/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8382 - accuracy: 0.8251 - val_loss: 0.8077
- val_accuracy: 0.8268
Epoch 14/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7933 - accuracy: 0.8339 - val_loss: 0.7850
- val_accuracy: 0.8309
Epoch 15/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7642 - accuracy: 0.8351 - val_loss: 0.7439
- val_accuracy: 0.8403
Epoch 16/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7533 - accuracy: 0.8377 - val_loss: 0.7252
- val_accuracy: 0.8432
Epoch 17/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7238 - accuracy: 0.8436 - val_loss: 0.7742
- val_accuracy: 0.8057
Epoch 18/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6898 - accuracy: 0.8484 - val_loss: 0.6968
- val_accuracy: 0.8453
Epoch 19/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6887 - accuracy: 0.8436 - val_loss: 0.6933
- val_accuracy: 0.8371
Epoch 20/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6633 - accuracy: 0.8507 - val_loss: 0.6744
- val_accuracy: 0.8345
107/107 [==============================] - 4s 41ms/step
The accuracy of this Neural Network is  = 0.8344564898915909
The precision of this Neural Network is = 0.8363466309799272
The reacll of this Neural Network is    = 0.8431410743059349
The f1_score of this Neural Network is  = 0.834147438010938
# Of Cross Validation Is 3 Is Running
```

```
Epoch 1/20
107/107 [==============================] - 16s 136ms/step - loss: 7.8417 - accuracy: 0.5276 - val_loss: 5.0169
- val_accuracy: 0.6631
Epoch 2/20
107/107 [==============================] - 13s 126ms/step - loss: 3.7313 - accuracy: 0.6764 - val_loss: 2.7401
- val_accuracy: 0.7111
Epoch 3/20
107/107 [==============================] - 13s 125ms/step - loss: 2.2550 - accuracy: 0.7205 - val_loss: 1.8235
- val_accuracy: 0.7594
Epoch 4/20
107/107 [==============================] - 13s 122ms/step - loss: 1.6729 - accuracy: 0.7422 - val_loss: 1.4749
- val_accuracy: 0.7577
Epoch 5/20
107/107 [==============================] - 13s 121ms/step - loss: 1.4102 - accuracy: 0.7596 - val_loss: 1.2654
- val_accuracy: 0.8013
Epoch 6/20
107/107 [==============================] - 13s 121ms/step - loss: 1.2492 - accuracy: 0.7749 - val_loss: 1.1901
- val_accuracy: 0.7767
Epoch 7/20
107/107 [==============================] - 13s 122ms/step - loss: 1.1535 - accuracy: 0.7825 - val_loss: 1.0740
- val_accuracy: 0.8101
Epoch 8/20
107/107 [==============================] - 13s 122ms/step - loss: 1.0664 - accuracy: 0.7919 - val_loss: 0.9890
- val_accuracy: 0.8222
Epoch 9/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9960 - accuracy: 0.8043 - val_loss: 0.9277
- val_accuracy: 0.8315
Epoch 10/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9369 - accuracy: 0.8157 - val_loss: 0.8927
- val_accuracy: 0.8263
Epoch 11/20
107/107 [==============================] - 13s 122ms/step - loss: 0.9037 - accuracy: 0.8096 - val_loss: 0.8702
- val_accuracy: 0.8110
Epoch 12/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8707 - accuracy: 0.8068 - val_loss: 0.8266
- val_accuracy: 0.8339
Epoch 13/20
107/107 [==============================] - 13s 122ms/step - loss: 0.8336 - accuracy: 0.8176 - val_loss: 0.7737
- val_accuracy: 0.8418
Epoch 14/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7879 - accuracy: 0.8304 - val_loss: 0.7417
- val_accuracy: 0.8506
Epoch 15/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7747 - accuracy: 0.8274 - val_loss: 0.7174
- val_accuracy: 0.8523
Epoch 16/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7397 - accuracy: 0.8357 - val_loss: 0.7557
- val_accuracy: 0.8157
Epoch 17/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7104 - accuracy: 0.8414 - val_loss: 0.6874
- val_accuracy: 0.8550
Epoch 18/20
107/107 [==============================] - 13s 122ms/step - loss: 0.7066 - accuracy: 0.8378 - val_loss: 0.6728
- val_accuracy: 0.8517
Epoch 19/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6735 - accuracy: 0.8481 - val_loss: 0.6817
- val_accuracy: 0.8347
Epoch 20/20
107/107 [==============================] - 13s 122ms/step - loss: 0.6587 - accuracy: 0.8459 - val_loss: 0.6357
- val_accuracy: 0.8515
107/107 [==============================] - 5s 42ms/step
The accuracy of this Neural Network is  = 0.8514503369469675
The precision of this Neural Network is = 0.8481405458929461
The reacll of this Neural Network is    = 0.8582048121054733
The f1_score of this Neural Network is  = 0.8473113307393185
```

In [ ]:

```
print('Using KFold - 3, the accuracy is  = {}'.format(np.array(accuracy).mean(),'\n'))
print('Using KFold - 3, the precision is = {}'.format(np.array(precision).mean(),'\n'))
print('Using KFold - 3, the recall is    = {}'.format(np.array(recall).mean(),'\n'))
print('Using KFold - 3, the f1 score is  = {}'.format(np.array(f1).mean(),'\n'))
```

```
Using KFold - 3, the accuracy is  = 0.836329418748959
Using KFold - 3, the precision is = 0.8366367755601818
Using KFold - 3, the recall is    = 0.8504208713583529
Using KFold - 3, the f1 score is  = 0.8334678114878401
```