**DAY 03**

**Dataset**

Examination dataset have been shared with three folders namely train, validation, validation2 and a Jupyter notebook. The shared dataset is a multi-class dataset. Each folder of Examination dataset has five more folders namely normal, cheat, phone, paperseeing, paperexchange.

Train set: train

Test set: validation

Validation set: validation2

*Google Drive*

<https://drive.google.com/drive/folders/1Zc0bVs66paccr6-yCYT8yI9lgqajFwBv?usp=sharing>

**Google Colab**

Anomaly Detection

## [1. Imports](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#im)

## [2. HyperParameters](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#hp)

## [3. Data Loading and Preprocessing](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#data)

## [4. DenseNet121 Feature Extractor](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#model)

## [5. Training](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#train)

## [6. Mutliclass AUC Curve](https://colab.research.google.com/drive/1re6r8c5DXnHsqSlWQQcBUVscZd-tFAae#auc)

IMPORTING LIBRARIES

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import os

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.preprocessing import LabelBinarizer

from sklearn.metrics import roc\_curve, auc, roc\_auc\_score

# from IPython.display import clear\_output

import warnings

warnings.filterwarnings('ignore')

HYPERPARAMETRERS AND DIRECTORIES

train\_dir = "D:/VIT/RESEARCH/dataset/ECDatatry1/train"

test\_dir = "D:/VIT/RESEARCH/dataset/ECDatatry1/validation"

SEED = 12

IMG\_HEIGHT = 64

IMG\_WIDTH = 64

BATCH\_SIZE = 64      #Was 46

EPOCHS = 1

LR =  0.001

NUM\_CLASSES = 14

CLASS\_LABELS = ['cheat','normal','peperexchange','paperseeing','phone']

DATA LOADING AND PRE-PROCESSING

preprocess\_fun = tf.keras.applications.resnet50.preprocess\_input

train\_datagen = ImageDataGenerator(

                                   horizontal\_flip=True,

                                   width\_shift\_range=0.1,

                                   height\_shift\_range=0.05,

                                   rescale = 1./255,

                                   validation\_split=0.2, # set validation split, Added

                                   preprocessing\_function=preprocess\_fun

                                  )

test\_datagen = ImageDataGenerator(rescale = 1./255,

                                  preprocessing\_function=preprocess\_fun

                                 )

train\_generator = train\_datagen.flow\_from\_directory(directory = train\_dir,

                                                    target\_size = (IMG\_HEIGHT ,IMG\_WIDTH),

                                                    batch\_size = BATCH\_SIZE,

                                                    shuffle  = True ,

                                                    color\_mode = "rgb",

                                                    class\_mode = "categorical",

                                                     subset='training',    #Added

                                                    seed = SEED

                                                   )

validation\_generator = train\_datagen.flow\_from\_directory(train\_dir, # same directory as training data

                                                    target\_size = (IMG\_HEIGHT ,IMG\_WIDTH),

                                                    batch\_size = BATCH\_SIZE,

                                                    shuffle  = True ,

                                                    color\_mode = "rgb",

                                                    class\_mode = "categorical",

                                                    subset='validation', # set as validation data,  #Added

                                                    seed = SEED

                                                   )

test\_generator = test\_datagen.flow\_from\_directory(directory = test\_dir,

                                                   target\_size = (IMG\_HEIGHT ,IMG\_WIDTH),

                                                    batch\_size = BATCH\_SIZE,

                                                    shuffle  = False ,

                                                    color\_mode = "rgb",

                                                    class\_mode = "categorical",

                                                    seed = SEED

                                                  )

Resnet50 Transfer Learning

from tensorflow.keras.callbacks import EarlyStopping

earlystop= tf.keras.callbacks.EarlyStopping(

           monitor="val\_loss",

           min\_delta=0.1,

           patience=3,

           verbose=0,

           mode="min",

           start\_from\_epoch =20,

from tensorflow.keras.callbacks import ModelCheckpoint

filepath="weights-improvementResnet50-{epoch:02d}-{val\_accuracy:.2f}.hdf5"

checkpoint = ModelCheckpoint(filepath, monitor='val\_accuracy', verbose=1, save\_best\_only=True, mode='max')

callbacks\_list = [checkpoint]

def feature\_extractor(inputs):

    feature\_extractor = tf.keras.applications.ResNet50(input\_shape=(IMG\_HEIGHT,IMG\_WIDTH, 3),

                                               include\_top=False,

                                               weights="imagenet")(inputs)

    return feature\_extractor

def classifier(inputs):

    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)

    x = tf.keras.layers.Dense(256, activation="relu")(x)

    x = tf.keras.layers.Dropout(0.3)(x)

    x = tf.keras.layers.Dense(1024, activation="relu")(x)

    x = tf.keras.layers.Dropout(0.5)(x)

    x = tf.keras.layers.Dense(512, activation="relu")(x)

    x = tf.keras.layers.Dropout(0.4) (x)

    x = tf.keras.layers.Dense(5, activation="softmax", name="classification")(x)

    return x

def final\_model(inputs):

    densenet\_feature\_extractor = feature\_extractor(inputs)

    classification\_output = classifier(densenet\_feature\_extractor)

    return classification\_output

def define\_compile\_model():

    inputs = tf.keras.layers.Input(shape=(IMG\_HEIGHT ,IMG\_WIDTH,3))

    classification\_output = final\_model(inputs)

    model = tf.keras.Model(inputs=inputs, outputs = classification\_output)

    model.compile(optimizer=tf.keras.optimizers.SGD(LR),

                loss='categorical\_crossentropy',

                metrics = ['accuracy'])

    return model

model = define\_compile\_model()

# clear\_output()

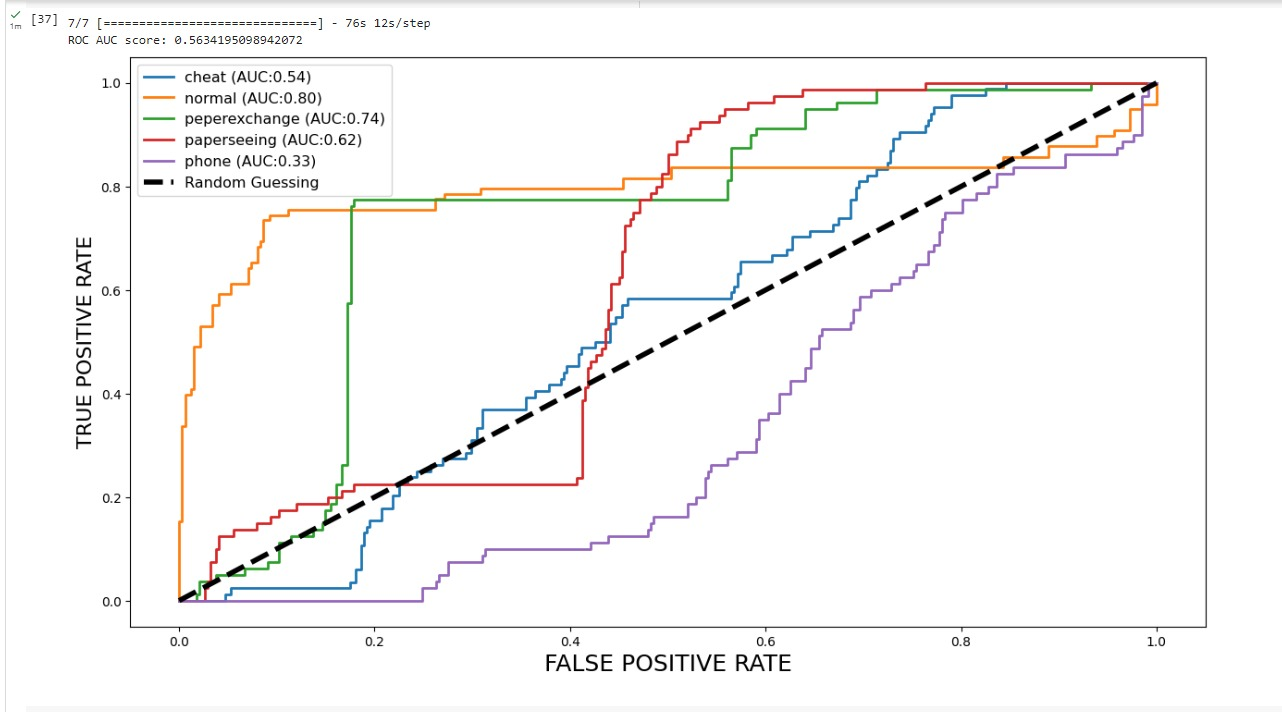
model.summary()

history = model.fit(x = train\_generator,validation\_data=validation\_generator,epochs = 100,callbacks=callbacks\_list)

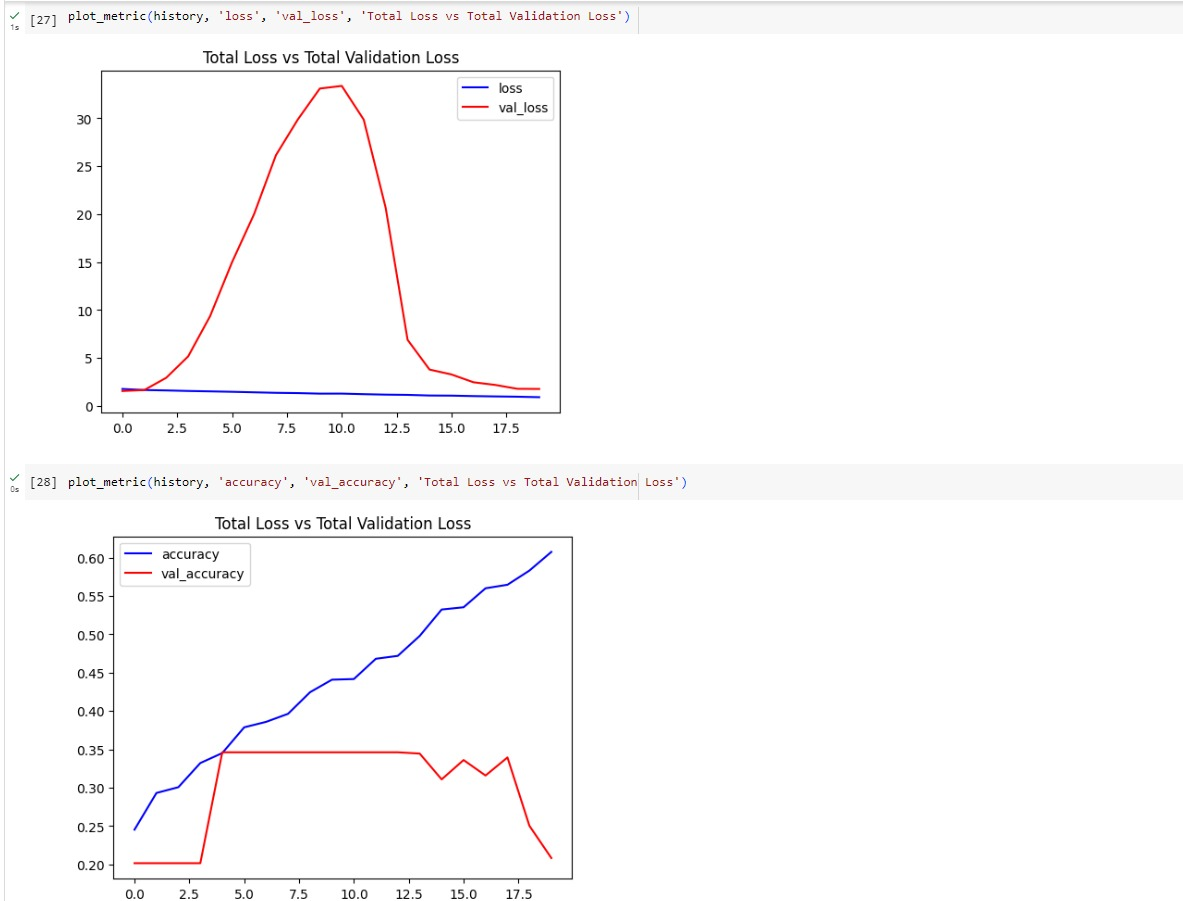
#history = model.fit(x = train\_generator,validation\_split=0.2,epochs = EPOCHS)

y\_test = test\_generator.classes

print (y\_test)

Multiclass AUC Curve

Loss and Accuracy



Confusion Matrix