EMOTION AI

AI Mini Project Report

ADITYA KHARE

SAP ID: 70021118067

ROLL No. : B258

B. Tech. CS 3rd Year 6th Semester

Table of content

I. Introduction

Artificial emotional intelligence or Emotion AI is a branch of AI that allows computers to understand human non-verbal cues such as body language and facial expressions.

II. Problem statement

The aim of this key project is to classify peoples emotion based on their face images .
In this case study I will assume that I AI /ML consultant I've been hired by a startup to build train and deploy a system that automatically monitors people emotions and expressions .
For this purpose, the team has collected more than 20,000 facial images with their associated facial expression labels and around 2000 images with their facial key point in annotations.

III. Proposed solution

To find a solution for the given a problem statement I have proposed model based on emotion AI model hey that takes an original image as an input and applies the algorithm to give the required output.
To achieve this output the proposed model is divided into 2 smaller prediction model. the first model is based on the facial key point detection and the second model is based on the facial expression that is in motion detection model both the model takes original image as input model one predict the facial key points on the image while the second model predicts the emotions class.
In the end, the 2 models are combined to get the required output.
In the first model that is facial key point detection I create am deep learning model based convolutional neural network and residual blocks to predict facial key points. The data set consists of x and y coordinates of 15 facial key points. Input images are 96 x 96 pixels. Images consists of only one color channel (Gray-Scale images).
Model construction is divided into several parts and tasks.
Parts are as follows:

I. Key facial point detection.
II. Facial expression detection.
III. Combining both facial expression and key points detection models.
IV. Deploying both trained models.

Tasks are as follows:

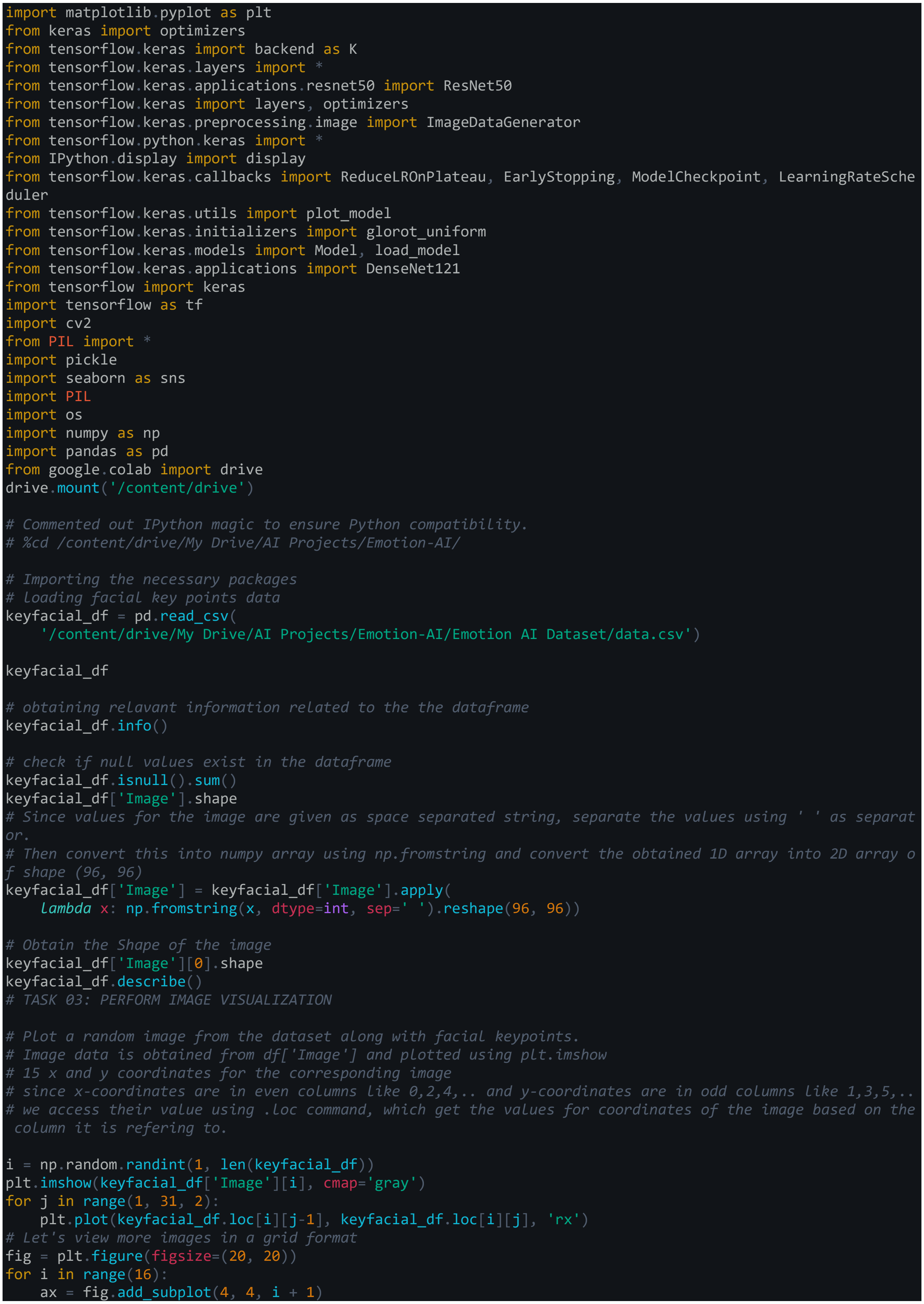
1. Understanding the problem statement in business case.
2. Importing libraries and data sets.
3. Perform image visualization.
4. Perform image of augmentation.
5. Perform data normalization and training data preparation.
6. Understand the theory and intuition behind neural networks.
7. Understand neural networks training process and gradient descent algorithm.
8. Understanding the theory and intuition behind convolutional neural networks and resnets.
9. Building deep residual neural networks key facial points detection model.
10. Compile and train key facial points detection deep learning model.
11. Assess trained key facial points detection model performance.
12. Import and explore dataset for facial expression detection.
13. Visualize images and plot labels.
14. Perform data preparation and image augmentation.

15. Build and train deep learning model for facial expression classification.
16. Understand how to assess classifier models (confusion matrix, accuracy, precision, and recall).
17. Assess the performance of trained facial expression classifier model.
18. Combine both models (1) Facial key detection and (2) Facial expression classification models.
19. Save the trained model for deployment.
20. Server the trained model using TensorFlow serving.
21. Make requests to model in TensorFlow serving.

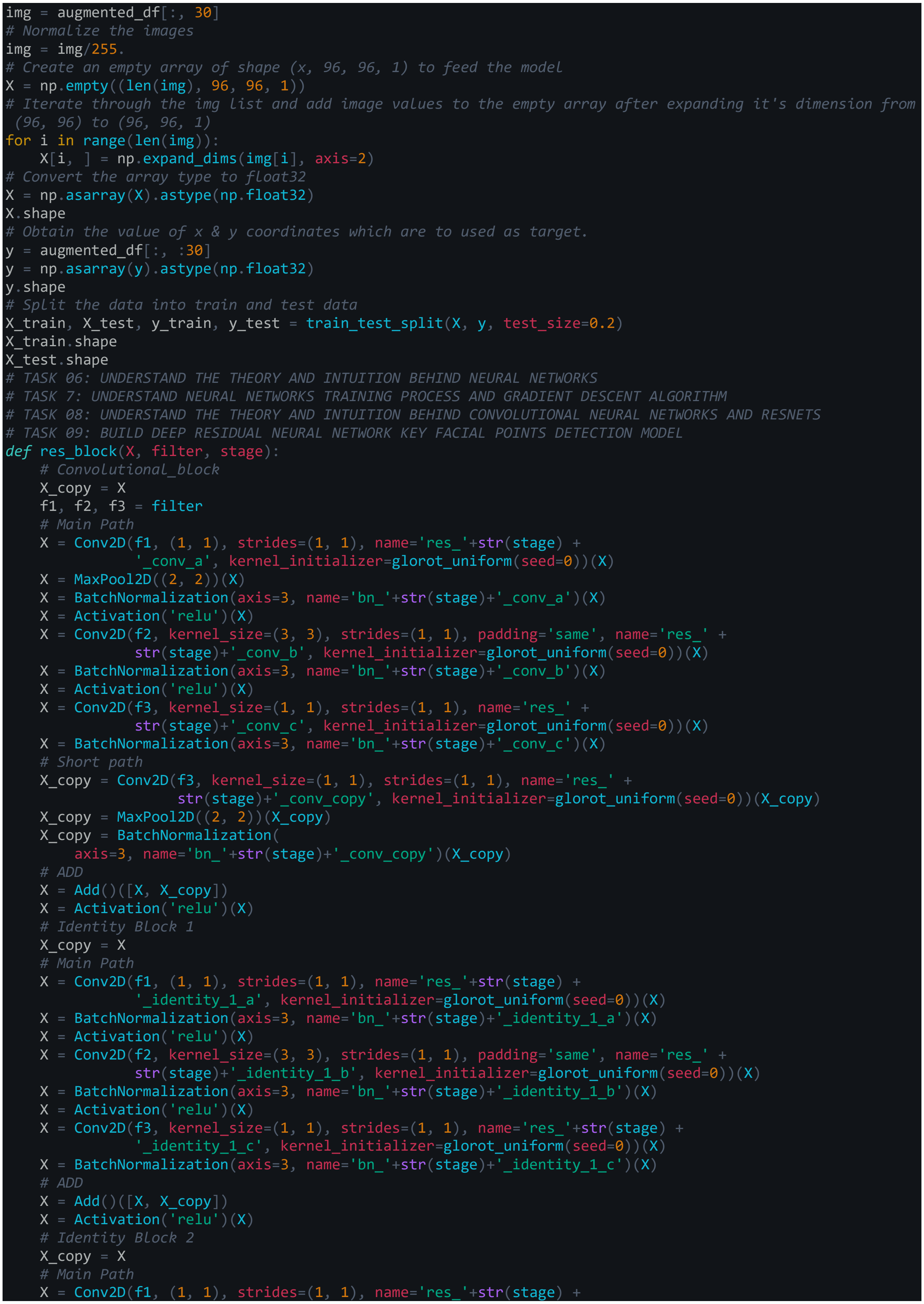
IV. Implementation

a. Code

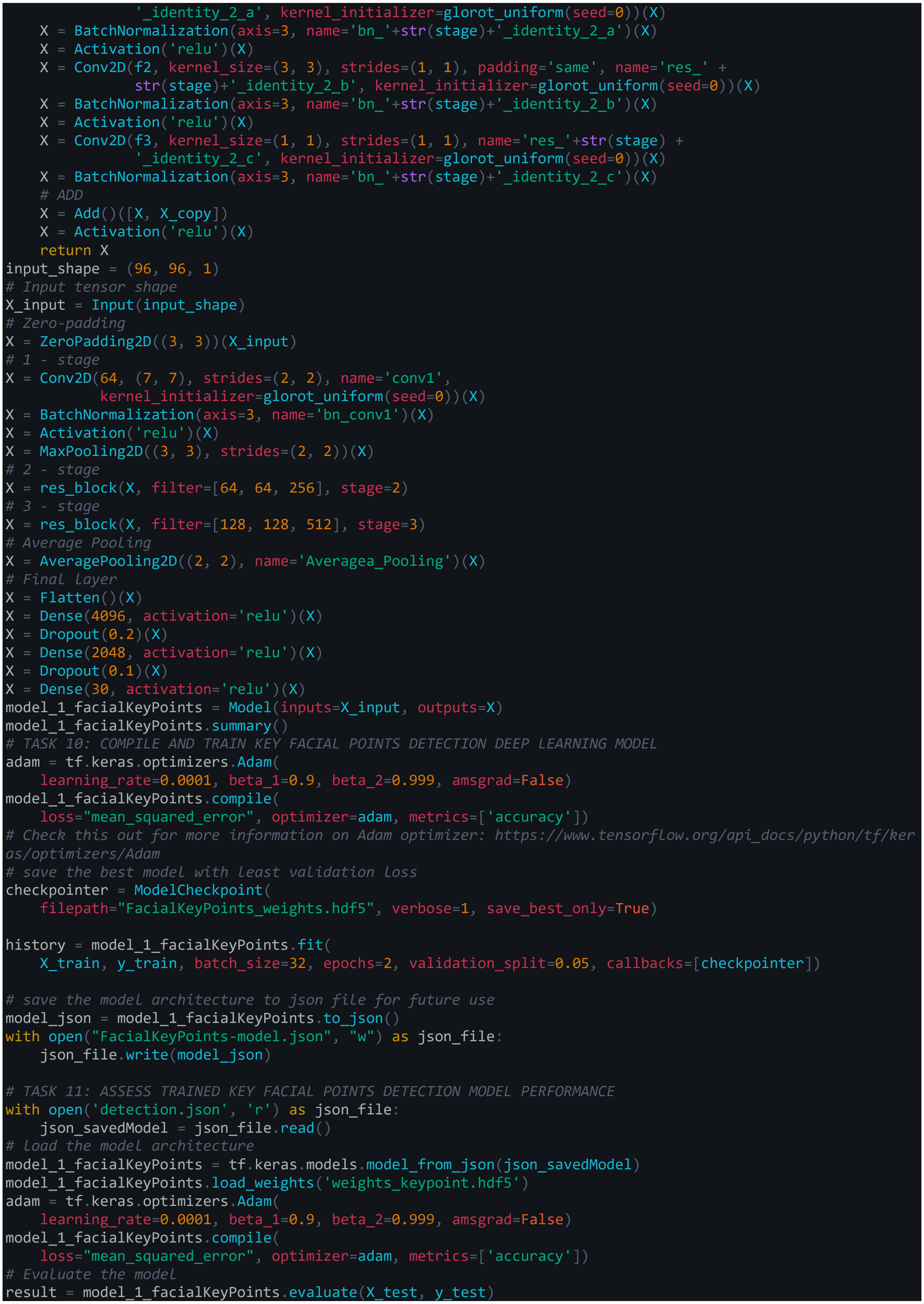
```
# -*- coding: utf-8 -*-
"""Emotion-AI.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1LkWmhXZNXMOr0YUzCve8U9G6vYAOw4TT

# TOPIC : EMOTION AI
# AIM :
The main aim of this project / case study is to generate a model that can be used to detect the emotion of
 a person using his/her facial points from a gray scale image.

# AUTHOR : ADITYA KHARE

This project/case study is divided into several smaller tasks which are further divied into various parts
which casn help to indentify the progress of the project more easily and will be convinent to track.

The several Tasks and their parts are as follows :
1. Understanding the problem statement and business care.
2.

# Task 01 : Understanding the problem statement and business case
# Project Overview :
1. The aim of this project is to  classify people's emotions based on their face images.
2. In this case study, I am going to assume that I am working as an AI/ML consultant.
3. I have been hired by a startup to build, train and deploy a system that automatically monitors people e
motions and expressions. For this a team shas collected more than 20,000 facial images, with their associa
ted facial expression labels and around 2,000 images with their facial key-point annotations.

# Part 01 : Key Facial Points Detection
1. In this we will create a deep learning model based on Convolutional Neural Network and Residual Blocks
to predict the facial key-points.
2. the dataset consists of x and y coordinates of facial key-points.
3. Input images are 96 x 96 pixels.
4. Images consists of only one color channel (gray-scale images).

# Task 02 : Importing Libraries and Datasets
"""

# Mounting the drive
import requests
import tensorflow.keras.backend as K
import json
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical
import copy
import random
from google.colab.patches import cv2_imshow
from sklearn.model_selection import train_test_split
```



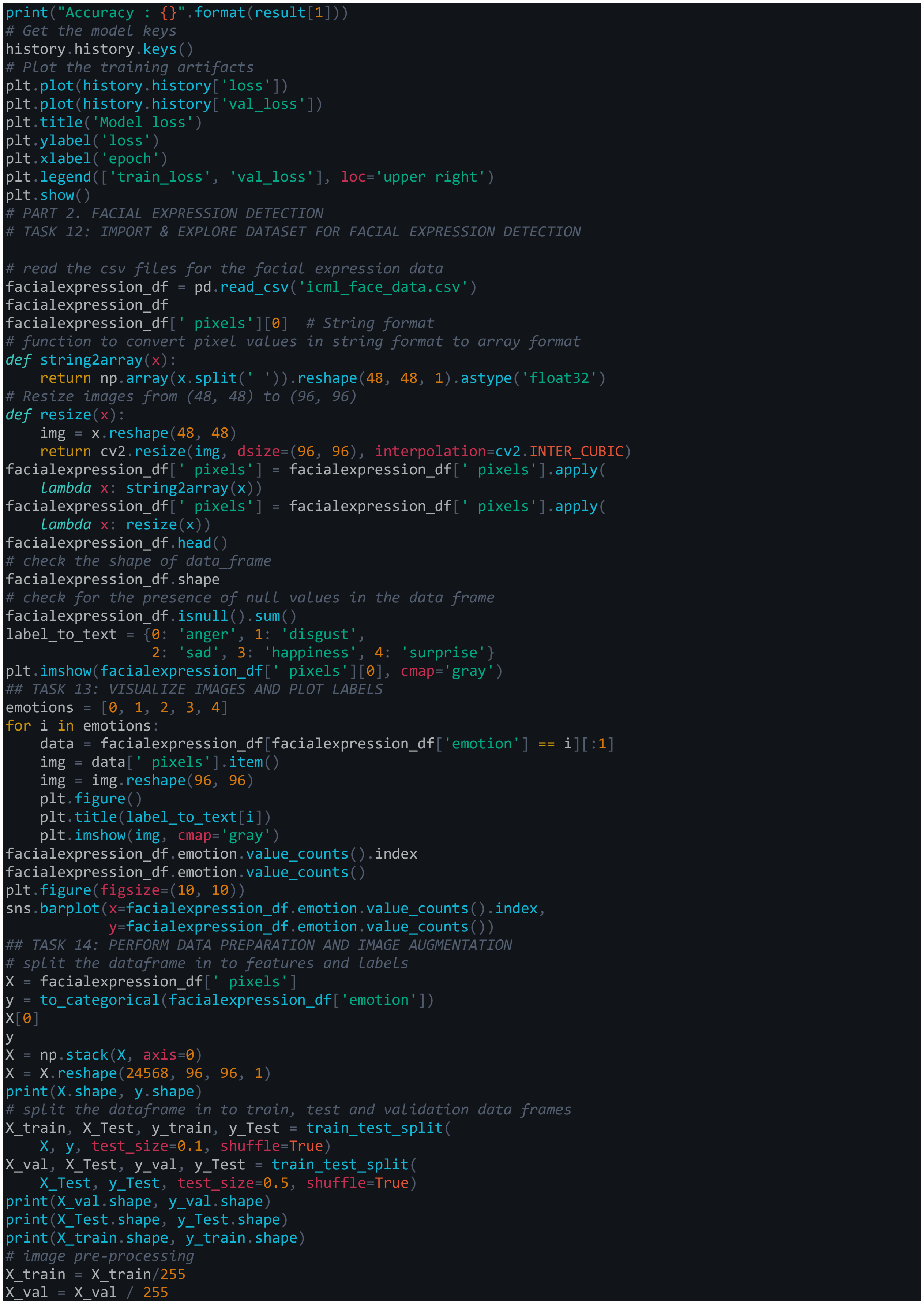
```
import matplotlib.pyplot as plt
from keras import optimizers
from tensorflow.keras import backend as K
from tensorflow.keras.layers import *
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import layers, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import *
from IPython.display import display
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateSche
duler
from tensorflow.keras.utils import plot_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.applications import DenseNet121
from tensorflow import keras
import tensorflow as tf
import cv2
from PIL import *
import pickle
import seaborn as sns
import PIL
import os
import numpy as np
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/My Drive/AI Projects/Emotion-AI/

# Importing the necessary packages
# loading facial key points data
keyfacial_df = pd.read_csv(
    '/content/drive/My Drive/AI Projects/Emotion-AI/Emotion AI Dataset/data.csv')

keyfacial_df

# obtaining relavant information related to the the dataframe
keyfacial_df.info()

# check if null values exist in the dataframe
keyfacial_df.isnull().sum()
keyfacial_df['Image'].shape
# Since values for the image are given as space separated string, separate the values using ' ' as separat
or.
# Then convert this into numpy array using np.fromstring and convert the obtained 1D array into 2D array o
f shape (96, 96)
keyfacial_df['Image'] = keyfacial_df['Image'].apply(
    lambda x: np.fromstring(x, dtype=int, sep=' ').reshape(96, 96))

# Obtain the Shape of the image
keyfacial_df['Image'][0].shape
keyfacial_df.describe()
# TASK 03: PERFORM IMAGE VISUALIZATION

# Plot a random image from the dataset along with facial keypoints.
# Image data is obtained from df['Image'] and plotted using plt.imshow
# 15 x and y coordinates for the corresponding image
# since x-coordinates are in even columns like 0,2,4,.. and y-coordinates are in odd columns like 1,3,5,..
# we access their value using .loc command, which get the values for coordinates of the image based on the
 column it is refering to.

i = np.random.randint(1, len(keyfacial_df))
plt.imshow(keyfacial_df['Image'][i], cmap='gray')
for j in range(1, 31, 2):
    plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))
for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1)
    image = plt.imshow(keyfacial_df['Image'][i], cmap='gray')
    for j in range(1, 31, 2):
        plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(64):
    k = random.randint(1, len(keyfacial_df))
    ax = fig.add_subplot(8, 8, i + 1)
    image = plt.imshow(keyfacial_df['Image'][k], cmap='gray')
    for j in range(1, 31, 2):
        plt.plot(keyfacial_df.loc[k][j-1], keyfacial_df.loc[k][j], 'rx')

## TASK 04: PERFORM IMAGE AUGMENTATION
# Create a new copy of the dataframe
keyfacial_df_copy = copy.copy(keyfacial_df)

# Obtain the columns in the dataframe
columns = keyfacial_df_copy.columns[:-1]
columns

# Horizontal Flip - flip the images along y axis
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(
    lambda x: np.flip(x, axis=1))
# since we are flipping horizontally, y coordinate values would be the same
# Only x coordiante values would change, all we must do is to subtract our initial x-
coordinate values from width of the image(96)
for i in range(len(columns)):
    if i % 2 == 0:
        keyfacial_df_copy[columns[i]] = keyfacial_df_copy[columns[i]].apply(
            lambda x: 96. - float(x))
# Show the Original image
plt.imshow(keyfacial_df['Image'][0], cmap='gray')
for j in range(1, 31, 2):
    plt.plot(keyfacial_df.loc[0][j-1], keyfacial_df.loc[0][j], 'rx')
# Show the Horizontally flipped image
plt.imshow(keyfacial_df_copy['Image'][0], cmap='gray')
for j in range(1, 31, 2):
    plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
# Concatenate the original dataframe with the augmented dataframe
augmented_df = np.concatenate((keyfacial_df, keyfacial_df_copy))
augmented_df.shape
# Randomingly increasing the brightness of the images
# We multiply pixel values by random values between 1.5 and 2 to increase the brightness of the image
# we clip the value between 0 and 255
keyfacial_df_copy = copy.copy(keyfacial_df)
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(
    lambda x: np.clip(random.uniform(1.5, 2) * x, 0.0, 255.0))
augmented_df = np.concatenate((augmented_df, keyfacial_df_copy))
augmented_df.shape

# Show Image with increased brightness
plt.imshow(keyfacial_df_copy['Image'][0], cmap='gray')
for j in range(1, 31, 2):
    plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')

keyfacial_df_copy = copy.copy(keyfacial_df)
keyfacial_df_copy['Image'] = keyfacial_df_copy['Image'].apply(
    lambda x: np.flip(x, axis=0))
for i in range(len(columns)):
    if i % 2 == 1:
        keyfacial_df_copy[columns[i]] = keyfacial_df_copy[columns[i]].apply(
            lambda x: 96. - float(x))

plt.imshow(keyfacial_df_copy['Image'][0], cmap='gray')
for j in range(1, 31, 2):
    plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')

## TASK 05: PERFORM DATA NORMALIZATION AND TRAINING DATA PREPARATION
# Obtain the value of images which is present in the 31st column (since index start from 0, we refer to 31
st column by 30)
```



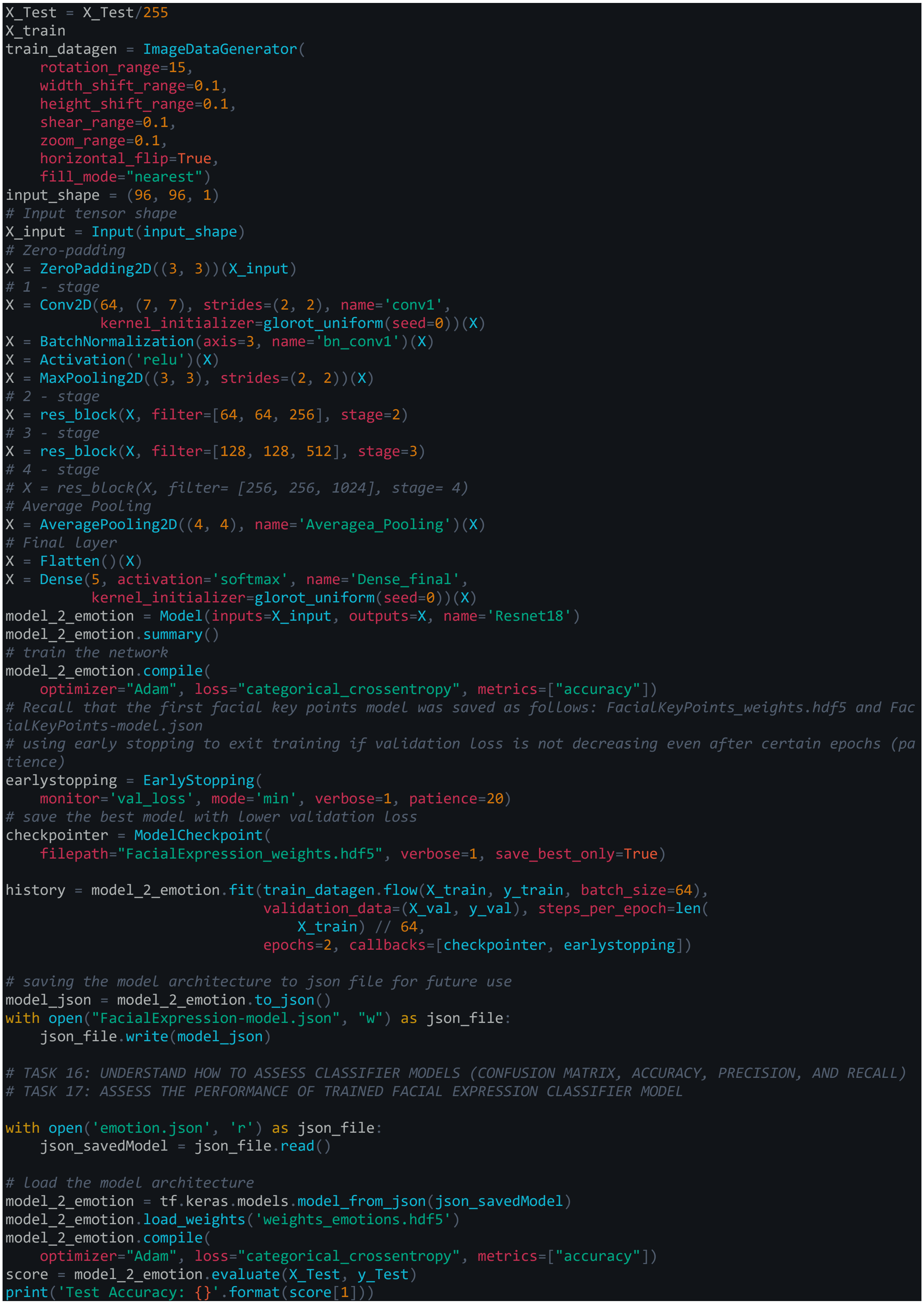
```
img = augmented_df[:, 30]
# Normalize the images
img = img/255.
# Create an empty array of shape (x, 96, 96, 1) to feed the model
X = np.empty((len(img), 96, 96, 1))
# Iterate through the img list and add image values to the empty array after expanding it's dimension from
 (96, 96) to (96, 96, 1)
for i in range(len(img)):
    X[i, ] = np.expand_dims(img[i], axis=2)
# Convert the array type to float32
X = np.asarray(X).astype(np.float32)
X.shape
# Obtain the value of x & y coordinates which are to used as target.
y = augmented_df[:, :30]
y = np.asarray(y).astype(np.float32)
y.shape
# Split the data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train.shape
X_test.shape
# TASK 06: UNDERSTAND THE THEORY AND INTUITION BEHIND NEURAL NETWORKS
# TASK 7: UNDERSTAND NEURAL NETWORKS TRAINING PROCESS AND GRADIENT DESCENT ALGORITHM
# TASK 08: UNDERSTAND THE THEORY AND INTUITION BEHIND CONVOLUTIONAL NEURAL NETWORKS AND RESNETS
# TASK 09: BUILD DEEP RESIDUAL NEURAL NETWORK KEY FACIAL POINTS DETECTION MODEL
def res_block(X, filter, stage):
    # Convolutional_block
    X_copy = X
    f1, f2, f3 = filter
    # Main Path
    X = Conv2D(f1, (1, 1), strides=(1, 1), name='res_'+str(stage) +
               '_conv_a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = MaxPool2D((2, 2))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_conv_a')(X)
    X = Activation('relu')(X)
    X = Conv2D(f2, kernel_size=(3, 3), strides=(1, 1), padding='same', name='res_' +
               str(stage)+'_conv_b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_conv_b')(X)
    X = Activation('relu')(X)
    X = Conv2D(f3, kernel_size=(1, 1), strides=(1, 1), name='res_' +
               str(stage)+'_conv_c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_conv_c')(X)
    # Short path
    X_copy = Conv2D(f3, kernel_size=(1, 1), strides=(1, 1), name='res_' +
                    str(stage)+'_conv_copy', kernel_initializer=glorot_uniform(seed=0))(X_copy)
    X_copy = MaxPool2D((2, 2))(X_copy)
    X_copy = BatchNormalization(
        axis=3, name='bn_'+str(stage)+'_conv_copy')(X_copy)
    # ADD
    X = Add()([X, X_copy])
    X = Activation('relu')(X)
    # Identity Block 1
    X_copy = X
    # Main Path
    X = Conv2D(f1, (1, 1), strides=(1, 1), name='res_'+str(stage) +
               '_identity_1_a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_1_a')(X)
    X = Activation('relu')(X)
    X = Conv2D(f2, kernel_size=(3, 3), strides=(1, 1), padding='same', name='res_' +
               str(stage)+'_identity_1_b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_1_b')(X)
    X = Activation('relu')(X)
    X = Conv2D(f3, kernel_size=(1, 1), strides=(1, 1), name='res_'+str(stage) +
               '_identity_1_c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_1_c')(X)
    # ADD
    X = Add()([X, X_copy])
    X = Activation('relu')(X)
    # Identity Block 2
    X_copy = X
    # Main Path
    X = Conv2D(f1, (1, 1), strides=(1, 1), name='res_'+str(stage) +
```



```
               '_identity_2_a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_2_a')(X)
    X = Activation('relu')(X)
    X = Conv2D(f2, kernel_size=(3, 3), strides=(1, 1), padding='same', name='res_' +
               str(stage)+'_identity_2_b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_2_b')(X)
    X = Activation('relu')(X)
    X = Conv2D(f3, kernel_size=(1, 1), strides=(1, 1), name='res_'+str(stage) +
               '_identity_2_c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_'+str(stage)+'_identity_2_c')(X)
    # ADD
    X = Add()([X, X_copy])
    X = Activation('relu')(X)
    return X
input_shape = (96, 96, 1)
# Input tensor shape
X_input = Input(input_shape)
# Zero-padding
X = ZeroPadding2D((3, 3))(X_input)
# 1 - stage
X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
           kernel_initializer=glorot_uniform(seed=0))(X)
X = BatchNormalization(axis=3, name='bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((3, 3), strides=(2, 2))(X)
# 2 - stage
X = res_block(X, filter=[64, 64, 256], stage=2)
# 3 - stage
X = res_block(X, filter=[128, 128, 512], stage=3)
# Average Pooling
X = AveragePooling2D((2, 2), name='Averagea_Pooling')(X)
# Final layer
X = Flatten()(X)
X = Dense(4096, activation='relu')(X)
X = Dropout(0.2)(X)
X = Dense(2048, activation='relu')(X)
X = Dropout(0.1)(X)
X = Dense(30, activation='relu')(X)
model_1_facialKeyPoints = Model(inputs=X_input, outputs=X)
model_1_facialKeyPoints.summary()
# TASK 10: COMPILE AND TRAIN KEY FACIAL POINTS DETECTION DEEP LEARNING MODEL
adam = tf.keras.optimizers.Adam(
    learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_1_facialKeyPoints.compile(
    loss="mean_squared_error", optimizer=adam, metrics=['accuracy'])
# Check this out for more information on Adam optimizer: https://www.tensorflow.org/api_docs/python/tf/ker
as/optimizers/Adam
# save the best model with least validation loss
checkpointer = ModelCheckpoint(
    filepath="FacialKeyPoints_weights.hdf5", verbose=1, save_best_only=True)

history = model_1_facialKeyPoints.fit(
    X_train, y_train, batch_size=32, epochs=2, validation_split=0.05, callbacks=[checkpointer])

# save the model architecture to json file for future use
model_json = model_1_facialKeyPoints.to_json()
with open("FacialKeyPoints-model.json", "w") as json_file:
    json_file.write(model_json)

# TASK 11: ASSESS TRAINED KEY FACIAL POINTS DETECTION MODEL PERFORMANCE
with open('detection.json', 'r') as json_file:
    json_savedModel = json_file.read()
# load the model architecture
model_1_facialKeyPoints = tf.keras.models.model_from_json(json_savedModel)
model_1_facialKeyPoints.load_weights('weights_keypoint.hdf5')
adam = tf.keras.optimizers.Adam(
    learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_1_facialKeyPoints.compile(
    loss="mean_squared_error", optimizer=adam, metrics=['accuracy'])
# Evaluate the model
result = model_1_facialKeyPoints.evaluate(X_test, y_test)
```

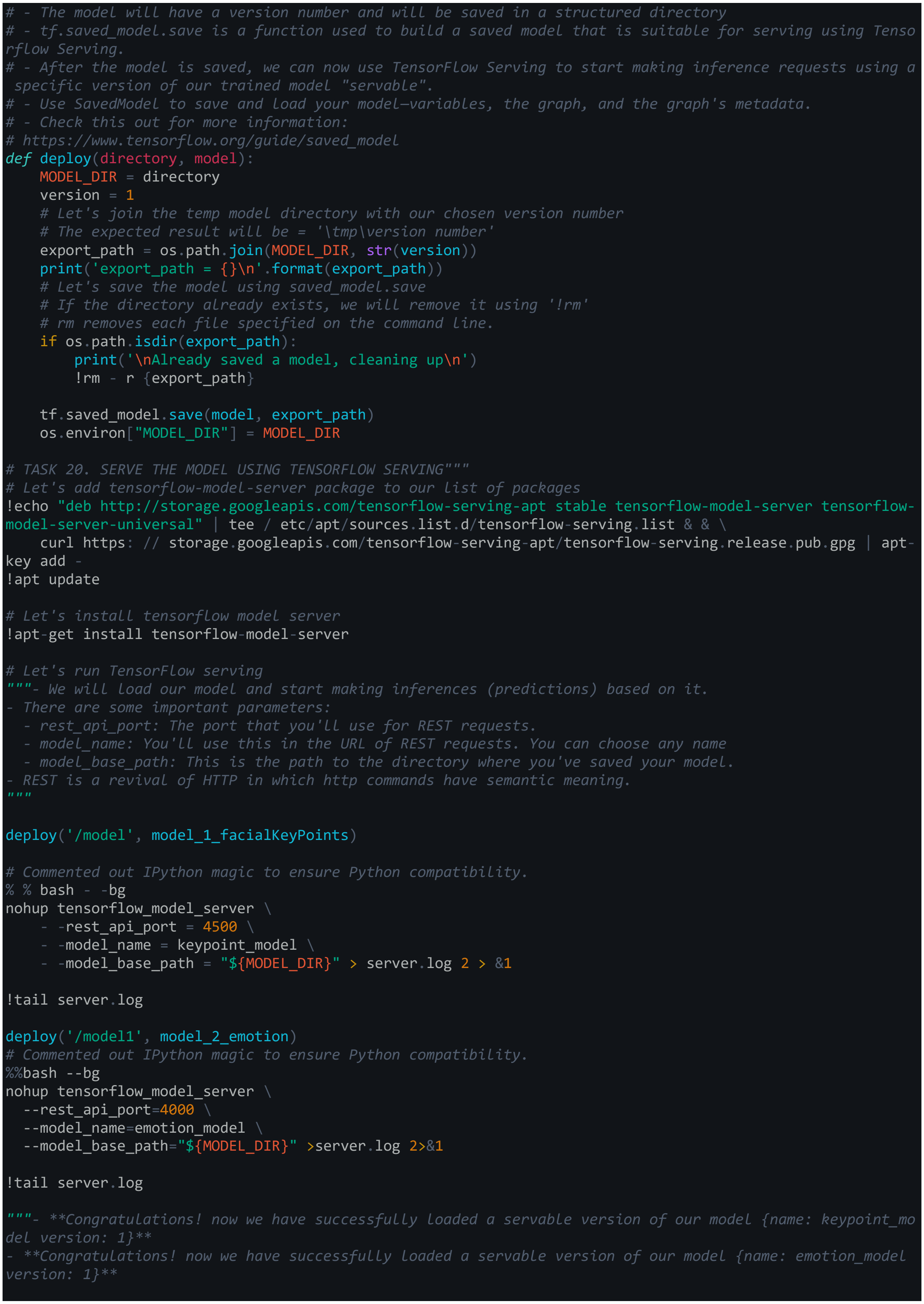


```
print("Accuracy : {}".format(result[1]))
# Get the model keys
history.history.keys()
# Plot the training artifacts
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss', 'val_loss'], loc='upper right')
plt.show()
# PART 2. FACIAL EXPRESSION DETECTION
# TASK 12: IMPORT & EXPLORE DATASET FOR FACIAL EXPRESSION DETECTION

# read the csv files for the facial expression data
facialexpression_df = pd.read_csv('icml_face_data.csv')
facialexpression_df
facialexpression_df[' pixels'][0]  # String format
# function to convert pixel values in string format to array format
def string2array(x):
    return np.array(x.split(' ')).reshape(48, 48, 1).astype('float32')
# Resize images from (48, 48) to (96, 96)
def resize(x):
    img = x.reshape(48, 48)
    return cv2.resize(img, dsize=(96, 96), interpolation=cv2.INTER_CUBIC)
facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(
    lambda x: string2array(x))
facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(
    lambda x: resize(x))
facialexpression_df.head()
# check the shape of data_frame
facialexpression_df.shape
# check for the presence of null values in the data frame
facialexpression_df.isnull().sum()
label_to_text = {0: 'anger', 1: 'disgust',
                 2: 'sad', 3: 'happiness', 4: 'surprise'}
plt.imshow(facialexpression_df[' pixels'][0], cmap='gray')
## TASK 13: VISUALIZE IMAGES AND PLOT LABELS
emotions = [0, 1, 2, 3, 4]
for i in emotions:
    data = facialexpression_df[facialexpression_df['emotion'] == i][:1]
    img = data[' pixels'].item()
    img = img.reshape(96, 96)
    plt.figure()
    plt.title(label_to_text[i])
    plt.imshow(img, cmap='gray')
facialexpression_df.emotion.value_counts().index
facialexpression_df.emotion.value_counts()
plt.figure(figsize=(10, 10))
sns.barplot(x=facialexpression_df.emotion.value_counts().index,
            y=facialexpression_df.emotion.value_counts())
## TASK 14: PERFORM DATA PREPARATION AND IMAGE AUGMENTATION
# split the dataframe in to features and labels
X = facialexpression_df[' pixels']
y = to_categorical(facialexpression_df['emotion'])
X[0]
y
X = np.stack(X, axis=0)
X = X.reshape(24568, 96, 96, 1)
print(X.shape, y.shape)
# split the dataframe in to train, test and validation data frames
X_train, X_Test, y_train, y_Test = train_test_split(
    X, y, test_size=0.1, shuffle=True)
X_val, X_Test, y_val, y_Test = train_test_split(
    X_Test, y_Test, test_size=0.5, shuffle=True)
print(X_val.shape, y_val.shape)
print(X_Test.shape, y_Test.shape)
print(X_train.shape, y_train.shape)
# image pre-processing
X_train = X_train/255
X_val = X_val / 255
```

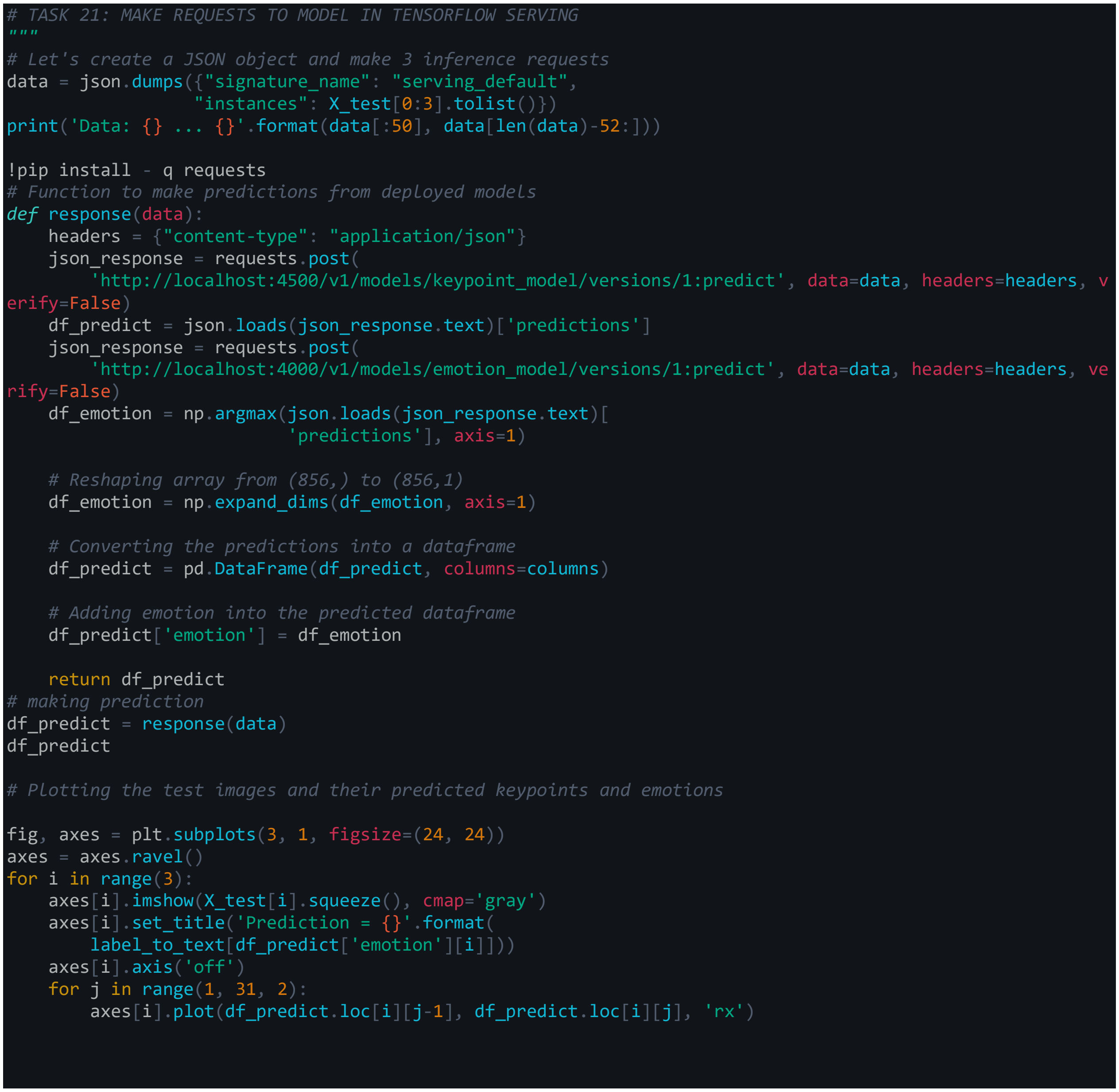


```python
X_Test = X_Test/255
X_train
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest")
input_shape = (96, 96, 1)
# Input tensor shape
X_input = Input(input_shape)
# Zero-padding
X = ZeroPadding2D((3, 3))(X_input)
# 1 - stage
X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
           kernel_initializer=glorot_uniform(seed=0))(X)
X = BatchNormalization(axis=3, name='bn_conv1')(X)
X = Activation('relu')(X)
X = MaxPooling2D((3, 3), strides=(2, 2))(X)
# 2 - stage
X = res_block(X, filter=[64, 64, 256], stage=2)
# 3 - stage
X = res_block(X, filter=[128, 128, 512], stage=3)
# 4 - stage
# X = res_block(X, filter= [256, 256, 1024], stage= 4)
# Average Pooling
X = AveragePooling2D((4, 4), name='Averagea_Pooling')(X)
# Final layer
X = Flatten()(X)
X = Dense(5, activation='softmax', name='Dense_final',
          kernel_initializer=glorot_uniform(seed=0))(X)
model_2_emotion = Model(inputs=X_input, outputs=X, name='Resnet18')
model_2_emotion.summary()
# train the network
model_2_emotion.compile(
    optimizer="Adam", loss="categorical_crossentropy", metrics=["accuracy"])
# Recall that the first facial key points model was saved as follows: FacialKeyPoints_weights.hdf5 and FacialKeyPoints-model.json
# using early stopping to exit training if validation loss is not decreasing even after certain epochs (patience)
earlystopping = EarlyStopping(
    monitor='val_loss', mode='min', verbose=1, patience=20)
# save the best model with lower validation loss
checkpointer = ModelCheckpoint(
    filepath="FacialExpression_weights.hdf5", verbose=1, save_best_only=True)

history = model_2_emotion.fit(train_datagen.flow(X_train, y_train, batch_size=64),
                              validation_data=(X_val, y_val), steps_per_epoch=len(
                                  X_train) // 64,
                              epochs=2, callbacks=[checkpointer, earlystopping])

# saving the model architecture to json file for future use
model_json = model_2_emotion.to_json()
with open("FacialExpression-model.json", "w") as json_file:
    json_file.write(model_json)

# TASK 16: UNDERSTAND HOW TO ASSESS CLASSIFIER MODELS (CONFUSION MATRIX, ACCURACY, PRECISION, AND RECALL)
# TASK 17: ASSESS THE PERFORMANCE OF TRAINED FACIAL EXPRESSION CLASSIFIER MODEL

with open('emotion.json', 'r') as json_file:
    json_savedModel = json_file.read()

# load the model architecture
model_2_emotion = tf.keras.models.model_from_json(json_savedModel)
model_2_emotion.load_weights('weights_emotions.hdf5')
model_2_emotion.compile(
    optimizer="Adam", loss="categorical_crossentropy", metrics=["accuracy"])
score = model_2_emotion.evaluate(X_Test, y_Test)
print('Test Accuracy: {}'.format(score[1]))
```

```
history.history.keys()
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()

# predicted_classes = model.predict_classes(X_test)
predicted_classes = np.argmax(model_2_emotion.predict(X_Test), axis=-1)
y_true = np.argmax(y_Test, axis=-1)
y_true.shape

cm = confusion_matrix(y_true, predicted_classes)
plt.figure(figsize=(10, 10))
sns.heatmap(cm, annot=True, cbar=False)
L = 10
W = 10
fig, axes = plt.subplots(L, W, figsize=(24, 24))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i].reshape(96, 96), cmap='gray')
    axes[i].set_title('Prediction = {}\n True = {}'.format(
        label_to_text[predicted_classes[i]], label_to_text[y_true[i]]))
    axes[i].axis('off')
plt.subplots_adjust(wspace=1)
print(classification_report(y_true, predicted_classes))

# PART 3. COMBINE BOTH FACIAL EXPRESSION AND KEY POINTS DETECTION MODELS
# TASK 18: COMBINE BOTH MODELS (1) FACIAL KEY POINTS DETECTION AND (2) FACIAL EXPRESSION MODELS
def predict(X_test):
    # Making prediction from the keypoint model
    df_predict = model_1_facialKeyPoints.predict(X_test)
    # Making prediction from the emotion model
    df_emotion = np.argmax(model_2_emotion.predict(X_test), axis=-1)
    # Reshaping array from (856,) to (856,1)
    df_emotion = np.expand_dims(df_emotion, axis=1)
    # Converting the predictions into a dataframe
    df_predict = pd.DataFrame(df_predict, columns=columns)
    # Adding emotion into the predicted dataframe
    df_predict['emotion'] = df_emotion
    return df_predict
df_predict = predict(X_test)
df_predict.head()

# Plotting the test images and their predicted keypoints and emotions
fig, axes = plt.subplots(4, 4, figsize=(24, 24))
axes = axes.ravel()

for i in range(16):

    axes[i].imshow(X_test[i].squeeze(), cmap='gray')
    axes[i].set_title('Prediction = {}'.format(
        label_to_text[df_predict['emotion'][i]]))
    axes[i].axis('off')
    for j in range(1, 31, 2):
        axes[i].plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')

# PART 4. DEPLOY BOTH TRAINED MODELS
# TASK 19. SAVE THE TRAINED MODEL FOR DEPLOYMENT
# - We now need to save our trained model and it must be saved in a SavedModel format.
```



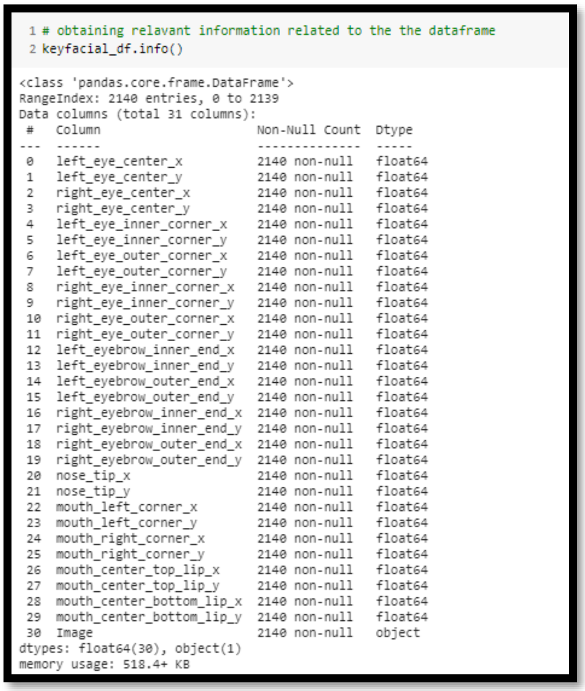
```
# - The model will have a version number and will be saved in a structured directory
# - tf.saved_model.save is a function used to build a saved model that is suitable for serving using Tenso
rflow Serving.
# - After the model is saved, we can now use TensorFlow Serving to start making inference requests using a
 specific version of our trained model "servable".
# - Use SavedModel to save and load your model—variables, the graph, and the graph's metadata.
# - Check this out for more information:
# https://www.tensorflow.org/guide/saved_model
def deploy(directory, model):
    MODEL_DIR = directory
    version = 1
    # Let's join the temp model directory with our chosen version number
    # The expected result will be = '\tmp\version number'
    export_path = os.path.join(MODEL_DIR, str(version))
    print('export_path = {}\n'.format(export_path))
    # Let's save the model using saved_model.save
    # If the directory already exists, we will remove it using '!rm'
    # rm removes each file specified on the command line.
    if os.path.isdir(export_path):
        print('\nAlready saved a model, cleaning up\n')
        !rm - r {export_path}

    tf.saved_model.save(model, export_path)
    os.environ["MODEL_DIR"] = MODEL_DIR

# TASK 20. SERVE THE MODEL USING TENSORFLOW SERVING"""
# Let's add tensorflow-model-server package to our list of packages
!echo "deb http://storage.googleapis.com/tensorflow-serving-apt stable tensorflow-model-server tensorflow-
model-server-universal" | tee / etc/apt/sources.list.d/tensorflow-serving.list & & \
    curl https: // storage.googleapis.com/tensorflow-serving-apt/tensorflow-serving.release.pub.gpg | apt-
key add -
!apt update

# Let's install tensorflow model server
!apt-get install tensorflow-model-server

# Let's run TensorFlow serving
"""- We will load our model and start making inferences (predictions) based on it.
- There are some important parameters:
  - rest_api_port: The port that you'll use for REST requests.
  - model_name: You'll use this in the URL of REST requests. You can choose any name
  - model_base_path: This is the path to the directory where you've saved your model.
- REST is a revival of HTTP in which http commands have semantic meaning.
"""

deploy('/model', model_1_facialKeyPoints)

# Commented out IPython magic to ensure Python compatibility.
% % bash - -bg
nohup tensorflow_model_server \
    - -rest_api_port = 4500 \
    - -model_name = keypoint_model \
    - -model_base_path = "${MODEL_DIR}" > server.log 2 > &1

!tail server.log

deploy('/model1', model_2_emotion)
# Commented out IPython magic to ensure Python compatibility.
%%bash --bg
nohup tensorflow_model_server \
  --rest_api_port=4000 \
  --model_name=emotion_model \
  --model_base_path="${MODEL_DIR}" >server.log 2>&1

!tail server.log

"""- **Congratulations! now we have successfully loaded a servable version of our model {name: keypoint_mo
del version: 1}**
- **Congratulations! now we have successfully loaded a servable version of our model {name: emotion_model
version: 1}**
```



```
# TASK 21: MAKE REQUESTS TO MODEL IN TENSORFLOW SERVING
"""
# Let's create a JSON object and make 3 inference requests
data = json.dumps({"signature_name": "serving_default",
                  "instances": X_test[0:3].tolist()})
print('Data: {} ... {}'.format(data[:50], data[len(data)-52:]))

!pip install - q requests
# Function to make predictions from deployed models
def response(data):
    headers = {"content-type": "application/json"}
    json_response = requests.post(
        'http://localhost:4500/v1/models/keypoint_model/versions/1:predict', data=data, headers=headers, v
erify=False)
    df_predict = json.loads(json_response.text)['predictions']
    json_response = requests.post(
        'http://localhost:4000/v1/models/emotion_model/versions/1:predict', data=data, headers=headers, ve
rify=False)
    df_emotion = np.argmax(json.loads(json_response.text)[
                           'predictions'], axis=1)

    # Reshaping array from (856,) to (856,1)
    df_emotion = np.expand_dims(df_emotion, axis=1)

    # Converting the predictions into a dataframe
    df_predict = pd.DataFrame(df_predict, columns=columns)

    # Adding emotion into the predicted dataframe
    df_predict['emotion'] = df_emotion

    return df_predict
# making prediction
df_predict = response(data)
df_predict

# Plotting the test images and their predicted keypoints and emotions

fig, axes = plt.subplots(3, 1, figsize=(24, 24))
axes = axes.ravel()
for i in range(3):
    axes[i].imshow(X_test[i].squeeze(), cmap='gray')
    axes[i].set_title('Prediction = {}'.format(
        label_to_text[df_predict['emotion'][i]]))
    axes[i].axis('off')
    for j in range(1, 31, 2):
        axes[i].plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```

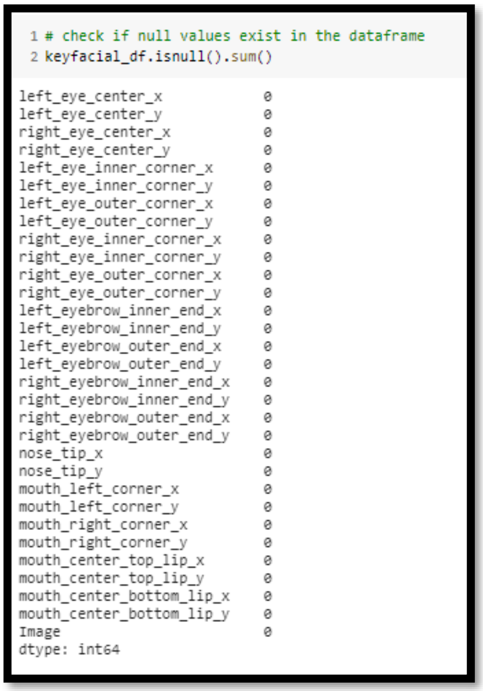
b. Output

After implementing this case study in goggle Colaboratory there are few specific outputs that I have included here. The complete implementation file is attached with this file in pdf as well as in .ipynb file format including all the dataset files and server logfile.



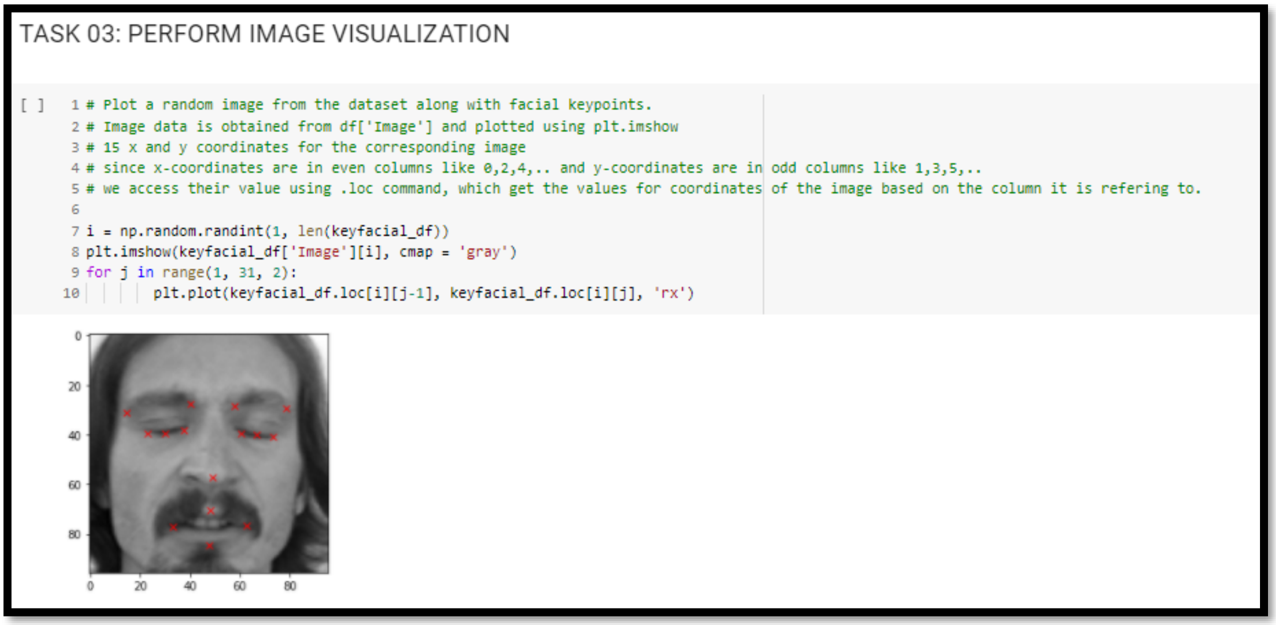
```
# obtaining relavant information related to the the dataframe
keyfacial_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2140 entries, 0 to 2139
Data columns (total 31 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   left_eye_center_x          2140 non-null   float64
 1   left_eye_center_y          2140 non-null   float64
 2   right_eye_center_x         2140 non-null   float64
 3   right_eye_center_y         2140 non-null   float64
 4   left_eye_inner_corner_x    2140 non-null   float64
 5   left_eye_inner_corner_y    2140 non-null   float64
 6   left_eye_outer_corner_x    2140 non-null   float64
 7   left_eye_outer_corner_y    2140 non-null   float64
 8   right_eye_inner_corner_x   2140 non-null   float64
 9   right_eye_inner_corner_y   2140 non-null   float64
 10  right_eye_outer_corner_x   2140 non-null   float64
 11  right_eye_outer_corner_y   2140 non-null   float64
 12  left_eyebrow_inner_end_x   2140 non-null   float64
 13  left_eyebrow_inner_end_y   2140 non-null   float64
 14  left_eyebrow_outer_end_x   2140 non-null   float64
 15  left_eyebrow_outer_end_y   2140 non-null   float64
 16  right_eyebrow_inner_end_x  2140 non-null   float64
 17  right_eyebrow_inner_end_y  2140 non-null   float64
 18  right_eyebrow_outer_end_x  2140 non-null   float64
 19  right_eyebrow_outer_end_y  2140 non-null   float64
 20  nose_tip_x                 2140 non-null   float64
 21  nose_tip_y                 2140 non-null   float64
 22  mouth_left_corner_x        2140 non-null   float64
 23  mouth_left_corner_y        2140 non-null   float64
 24  mouth_right_corner_x       2140 non-null   float64
 25  mouth_right_corner_y       2140 non-null   float64
 26  mouth_center_top_lip_x     2140 non-null   float64
 27  mouth_center_top_lip_y     2140 non-null   float64
 28  mouth_center_bottom_lip_x  2140 non-null   float64
 29  mouth_center_bottom_lip_y  2140 non-null   float64
 30  Image                      2140 non-null   object 
dtypes: float64(30), object(1)
memory usage: 518.4+ KB
```



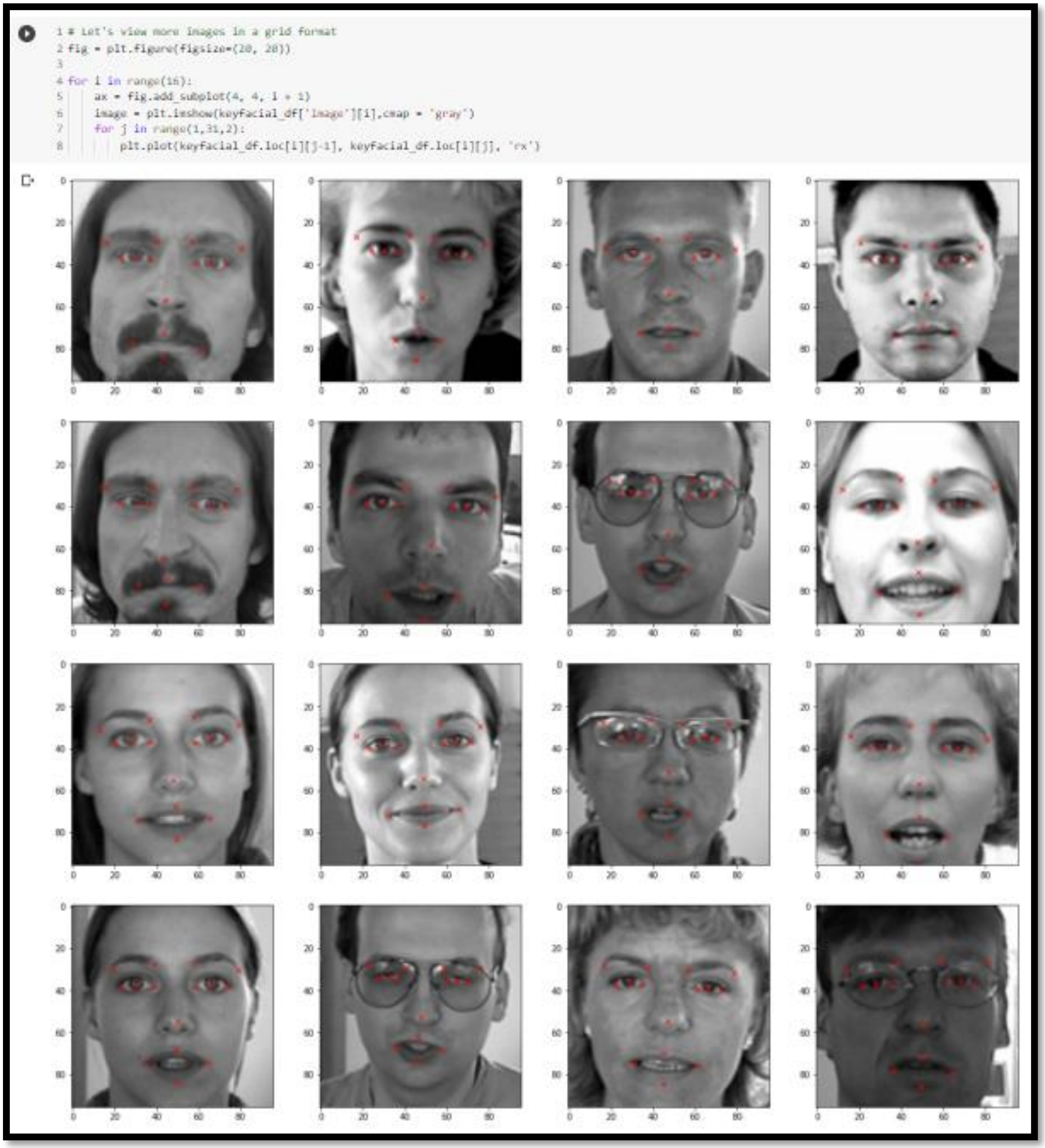
```
# check if null values exist in the dataframe
keyfacial_df.isnull().sum()
```

```
left_eye_center_x            0
left_eye_center_y            0
right_eye_center_x           0
right_eye_center_y           0
left_eye_inner_corner_x      0
left_eye_inner_corner_y      0
left_eye_outer_corner_x      0
left_eye_outer_corner_y      0
right_eye_inner_corner_x     0
right_eye_inner_corner_y     0
right_eye_outer_corner_x     0
right_eye_outer_corner_y     0
left_eyebrow_inner_end_x     0
left_eyebrow_inner_end_y     0
left_eyebrow_outer_end_x     0
left_eyebrow_outer_end_y     0
right_eyebrow_inner_end_x    0
right_eyebrow_inner_end_y    0
right_eyebrow_outer_end_x    0
right_eyebrow_outer_end_y    0
nose_tip_x                   0
nose_tip_y                   0
mouth_left_corner_x          0
mouth_left_corner_y          0
mouth_right_corner_x         0
mouth_right_corner_y         0
mouth_center_top_lip_x       0
mouth_center_top_lip_y       0
mouth_center_bottom_lip_x    0
mouth_center_bottom_lip_y    0
Image                        0
dtype: int64
```

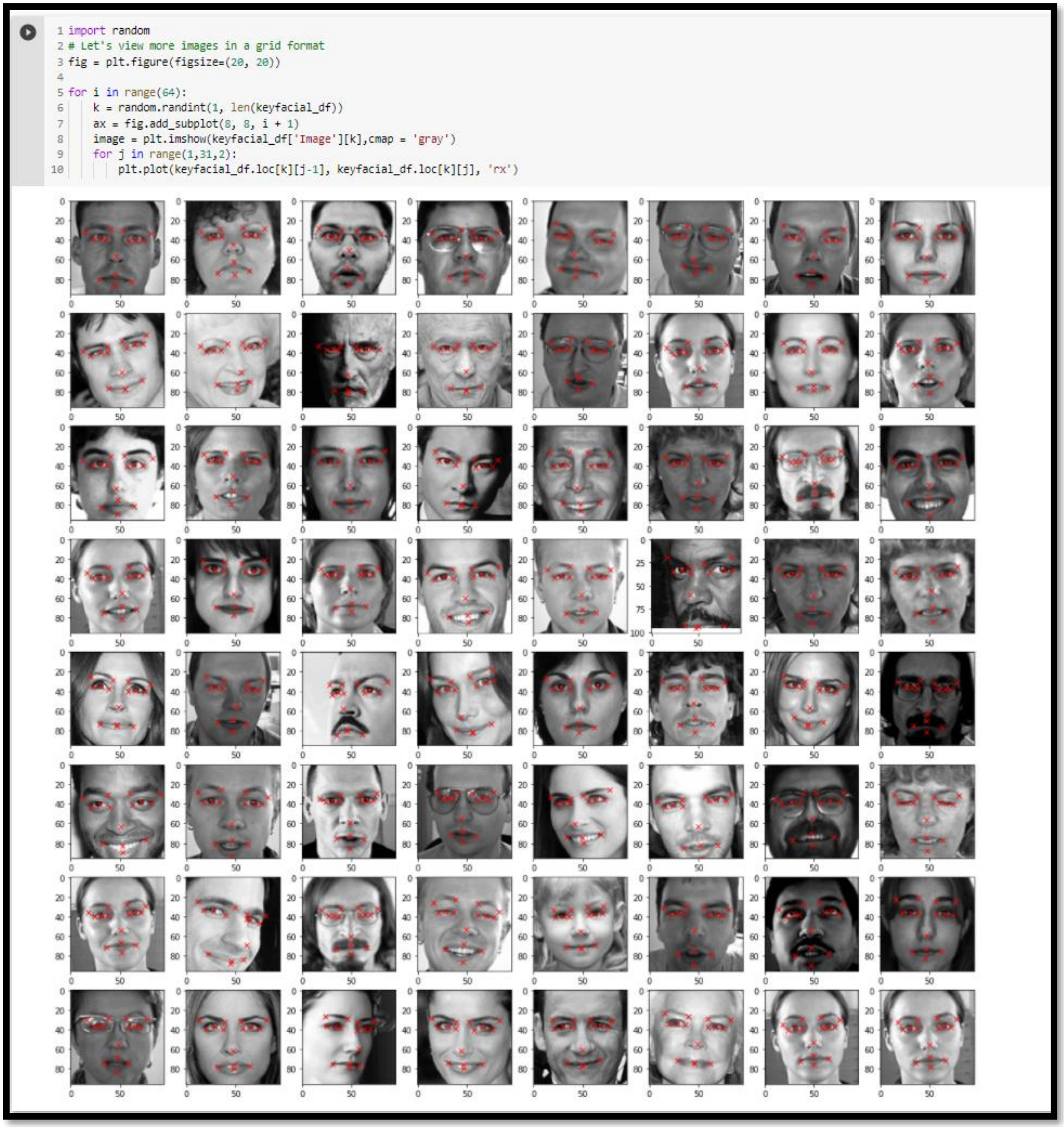


TASK 03: PERFORM IMAGE VISUALIZATION

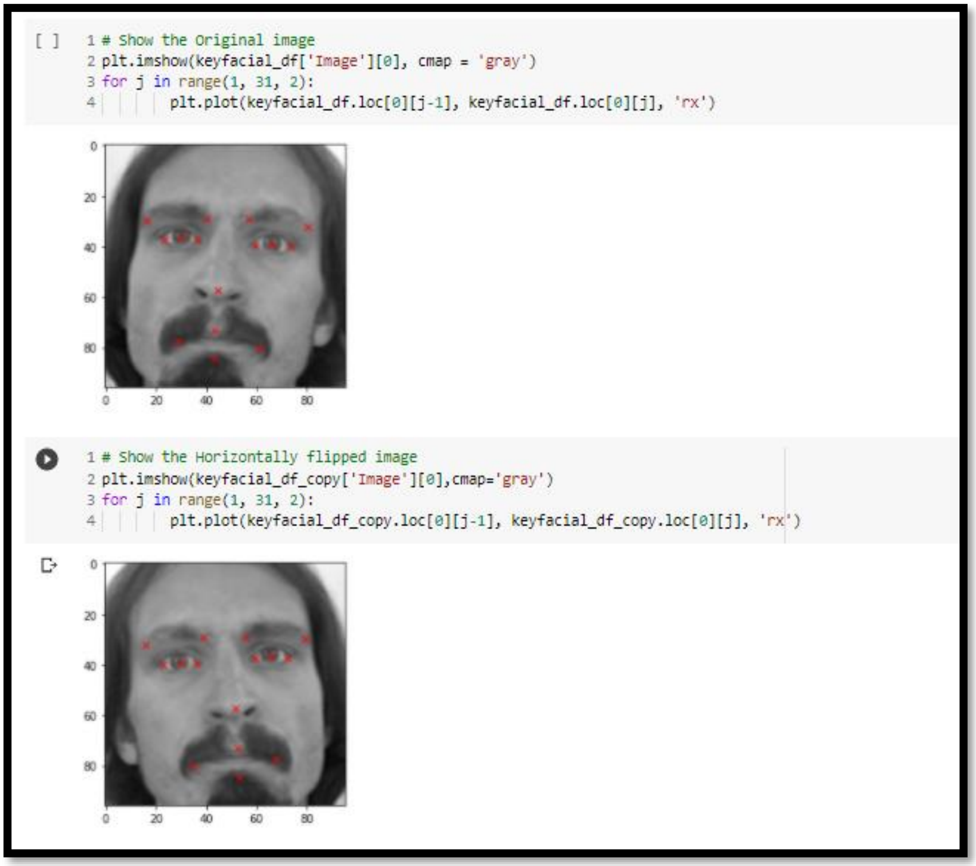
```
[ ]
# Plot a random image from the dataset along with facial keypoints.
# Image data is obtained from df['Image'] and plotted using plt.imshow
# 15 x and y coordinates for the corresponding image
# since x-coordinates are in even columns like 0,2,4,.. and y-coordinates are in odd columns like 1,3,5,..
# we access their value using .loc command, which get the values for coordinates of the image based on the column it is refering to.

i = np.random.randint(1, len(keyfacial_df))
plt.imshow(keyfacial_df['Image'][i], cmap = 'gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
```



```
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1)
    image = plt.imshow(keyfacial_df['image'][i],cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(keyfacial_df.loc[i][j-1], keyfacial_df.loc[i][j], 'rx')
```

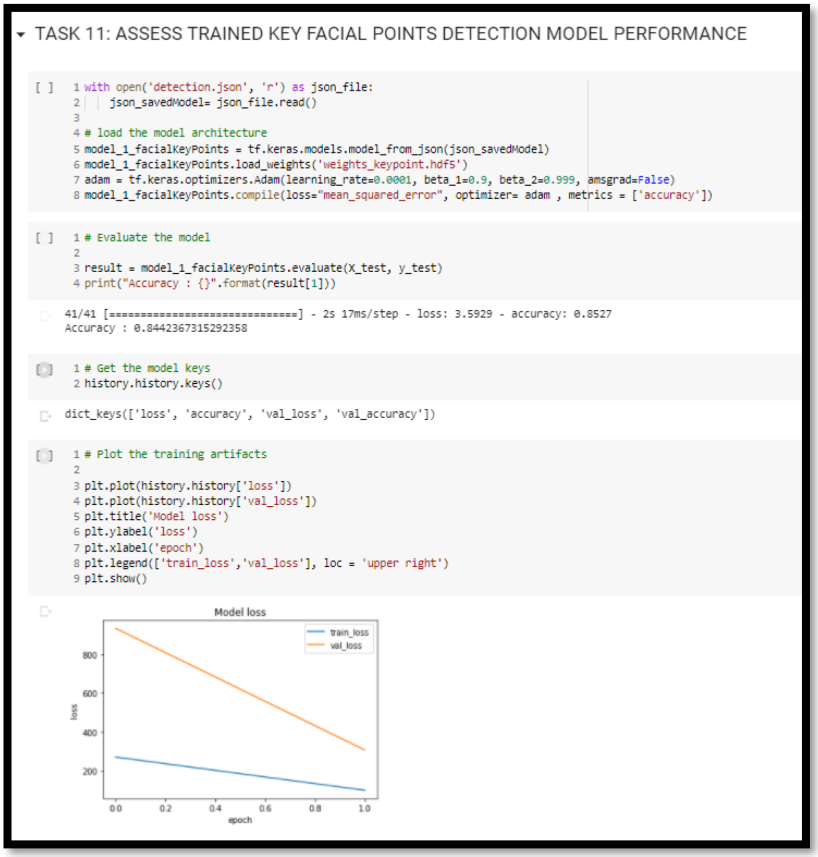


```
import random
# Let's view more images in a grid format
fig = plt.figure(figsize=(20, 20))

for i in range(64):
    k = random.randint(1, len(keyfacial_df))
    ax = fig.add_subplot(8, 8, i + 1)
    image = plt.imshow(keyfacial_df['Image'][k],cmap = 'gray')
    for j in range(1,31,2):
        plt.plot(keyfacial_df.loc[k][j-1], keyfacial_df.loc[k][j], 'rx')
```



```
# Show the Original image
plt.imshow(keyfacial_df['Image'][0], cmap = 'gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df.loc[0][j-1], keyfacial_df.loc[0][j], 'rx')
```

```
# Show the Horizontally flipped image
plt.imshow(keyfacial_df_copy['Image'][0],cmap='gray')
for j in range(1, 31, 2):
        plt.plot(keyfacial_df_copy.loc[0][j-1], keyfacial_df_copy.loc[0][j], 'rx')
```



TASK 11: ASSESS TRAINED KEY FACIAL POINTS DETECTION MODEL PERFORMANCE

```
with open('detection.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_1_facialKeyPoints = tf.keras.models.model_from_json(json_savedModel)
model_1_facialKeyPoints.load_weights('weights_keypoint.hdf5')
adam = tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_1_facialKeyPoints.compile(loss="mean_squared_error", optimizer= adam , metrics = ['accuracy'])
```

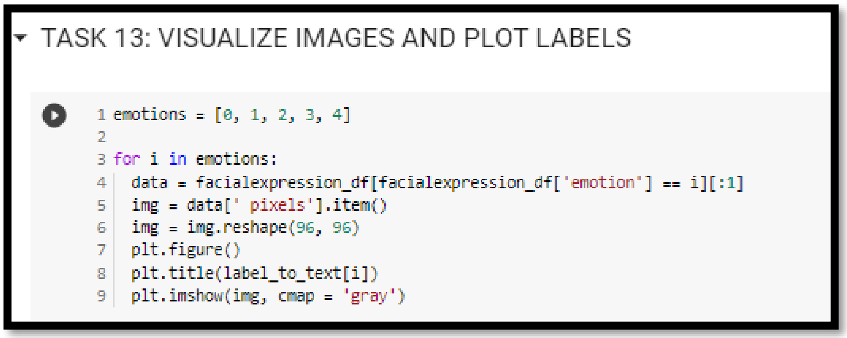
```
# Evaluate the model

result = model_1_facialKeyPoints.evaluate(X_test, y_test)
print("Accuracy : {}".format(result[1]))
```

```
41/41 [==============================] - 2s 17ms/step - loss: 3.5929 - accuracy: 0.8527
Accuracy : 0.8442367315292358
```

```
# Get the model keys
history.history.keys()
```

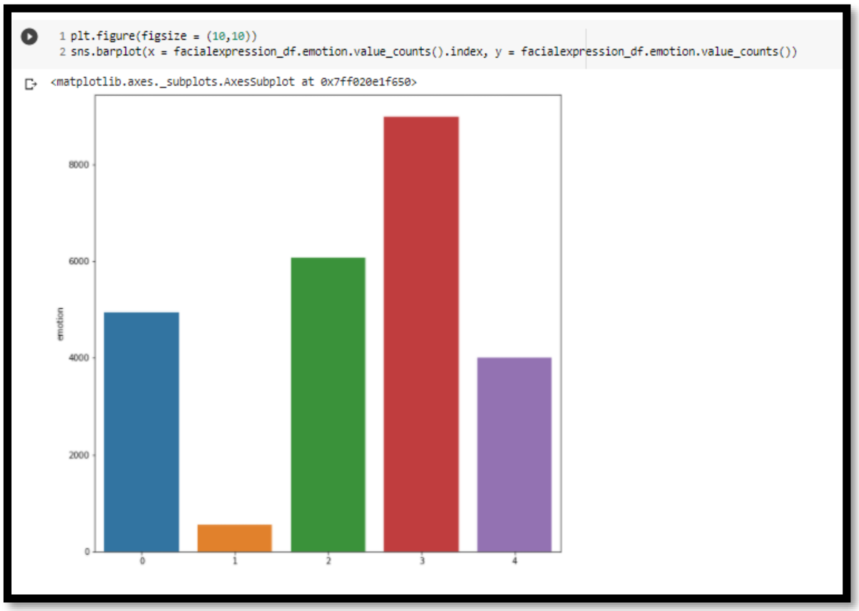
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
# Plot the training artifacts

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train_loss','val_loss'], loc = 'upper right')
plt.show()
```



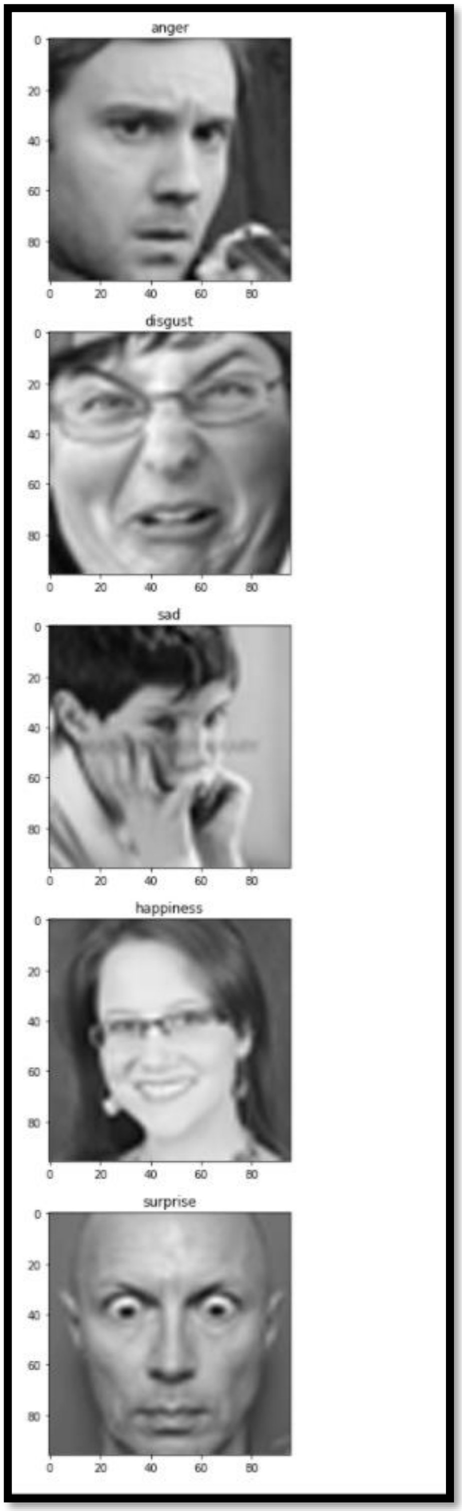
TASK 13: VISUALIZE IMAGES AND PLOT LABELS

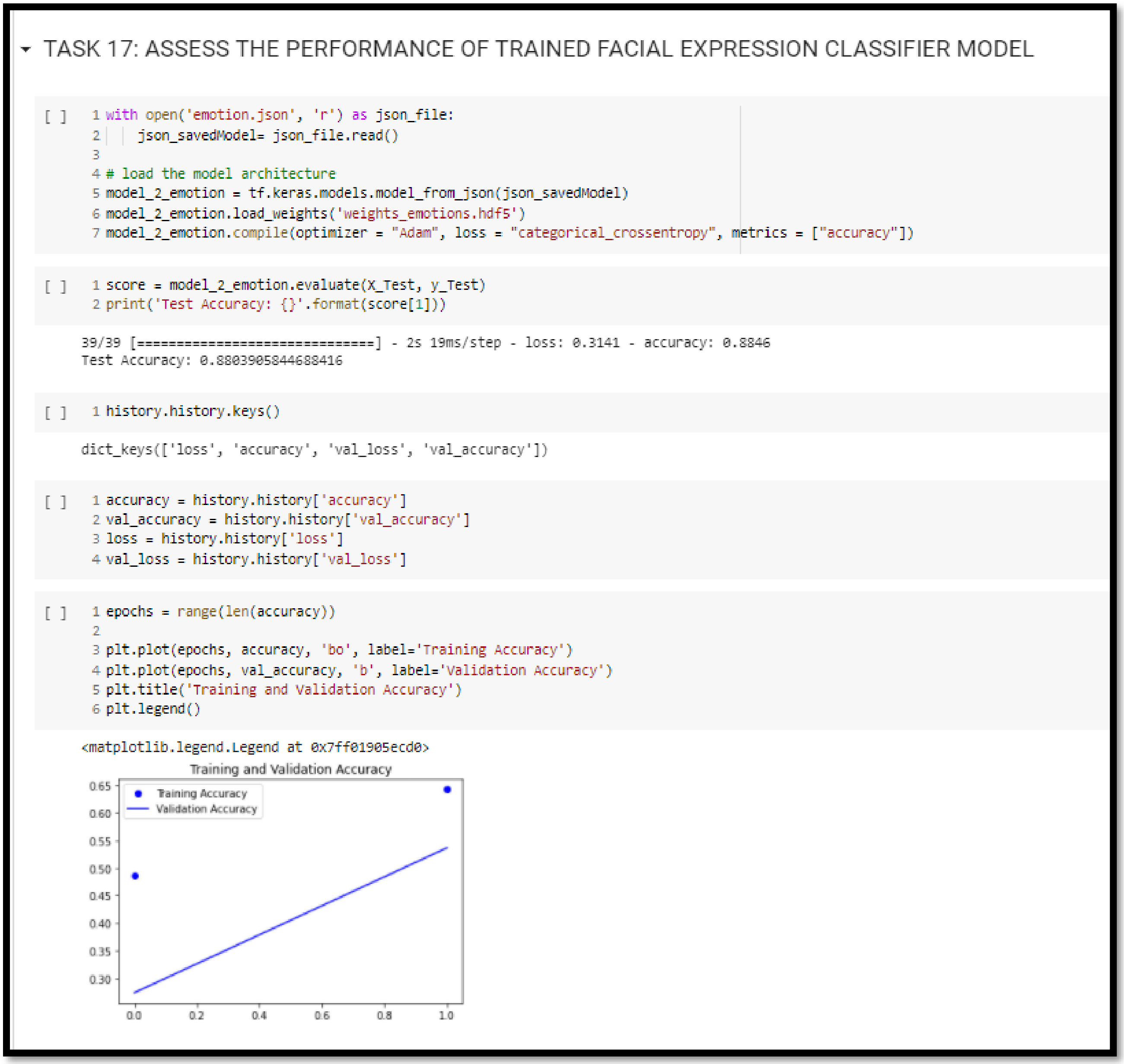
```
emotions = [0, 1, 2, 3, 4]

for i in emotions:
  data = facialexpression_df[facialexpression_df['emotion'] == i][:1]
  img = data[' pixels'].item()
  img = img.reshape(96, 96)
  plt.figure()
  plt.title(label_to_text[i])
  plt.imshow(img, cmap = 'gray')
```



```
plt.figure(figsize = (10,10))
sns.barplot(x = facialexpression_df.emotion.value_counts().index, y = facialexpression_df.emotion.value_counts())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff020e1f650>
```





TASK 17: ASSESS THE PERFORMANCE OF TRAINED FACIAL EXPRESSION CLASSIFIER MODEL

```
with open('emotion.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_2_emotion = tf.keras.models.model_from_json(json_savedModel)
model_2_emotion.load_weights('weights_emotions.hdf5')
model_2_emotion.compile(optimizer = "Adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
score = model_2_emotion.evaluate(X_Test, y_Test)
print('Test Accuracy: {}'.format(score[1]))
```

```
39/39 [==============================] - 2s 19ms/step - loss: 0.3141 - accuracy: 0.8846
Test Accuracy: 0.8803905844688416
```

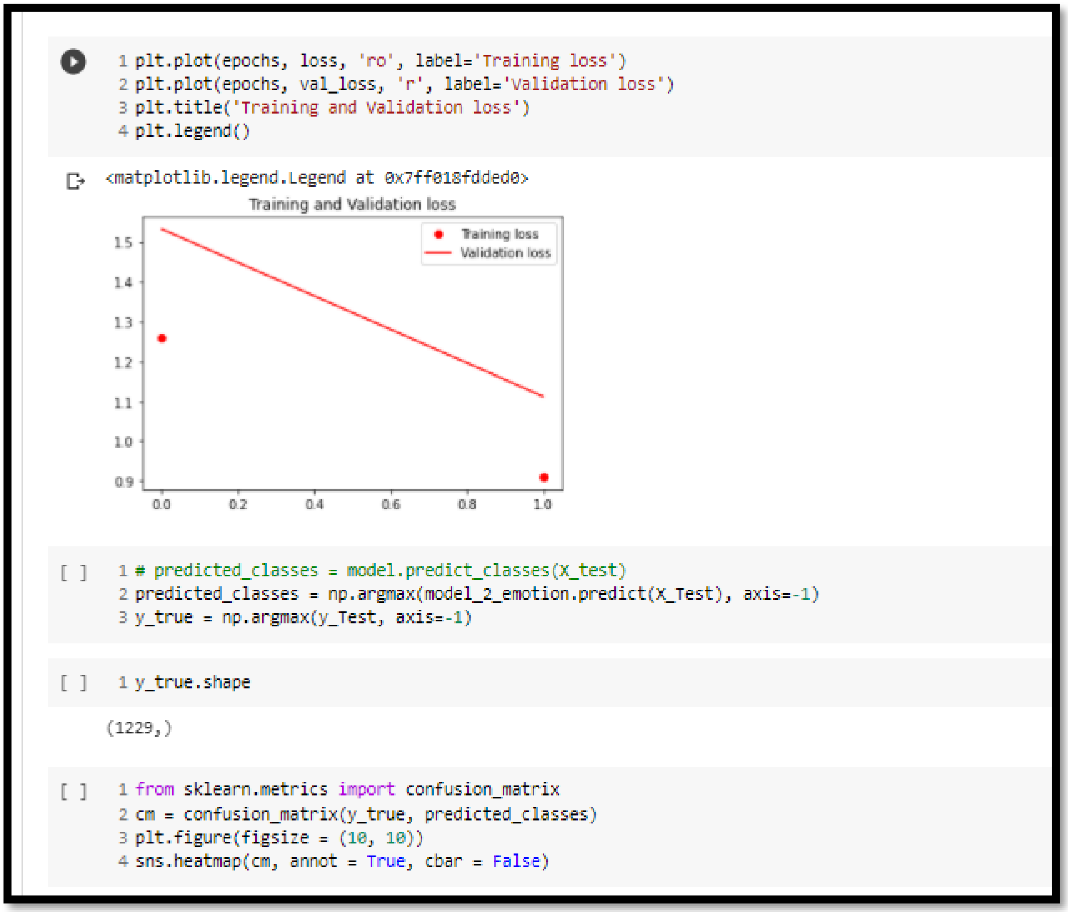
```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ff01905ecd0>
```



```
plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
```

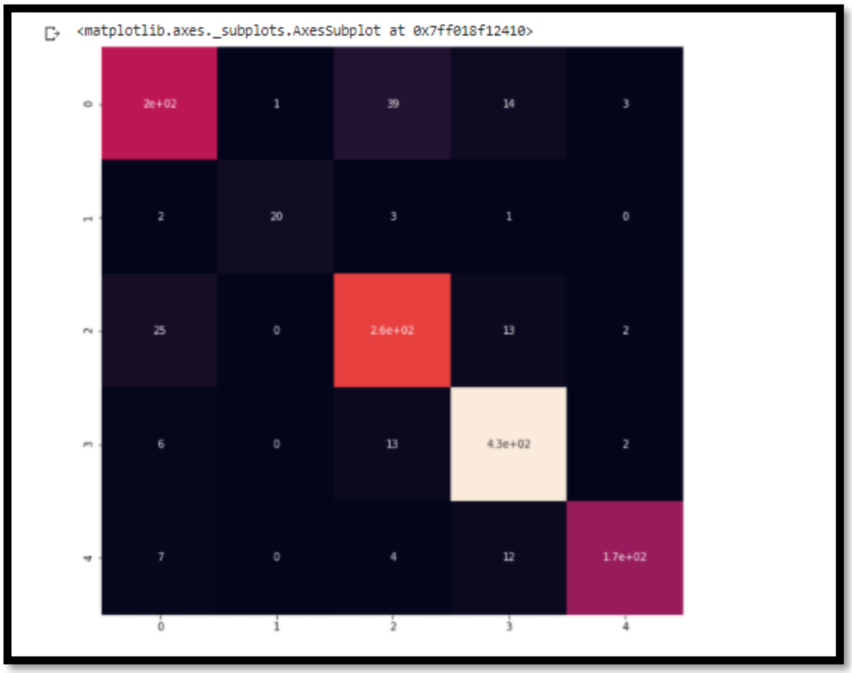
```
<matplotlib.legend.Legend at 0x7ff018fdded0>
```

```
[ ] # predicted_classes = model.predict_classes(X_test)
    predicted_classes = np.argmax(model_2_emotion.predict(X_Test), axis=-1)
    y_true = np.argmax(y_Test, axis=-1)
```

```
[ ] y_true.shape
```

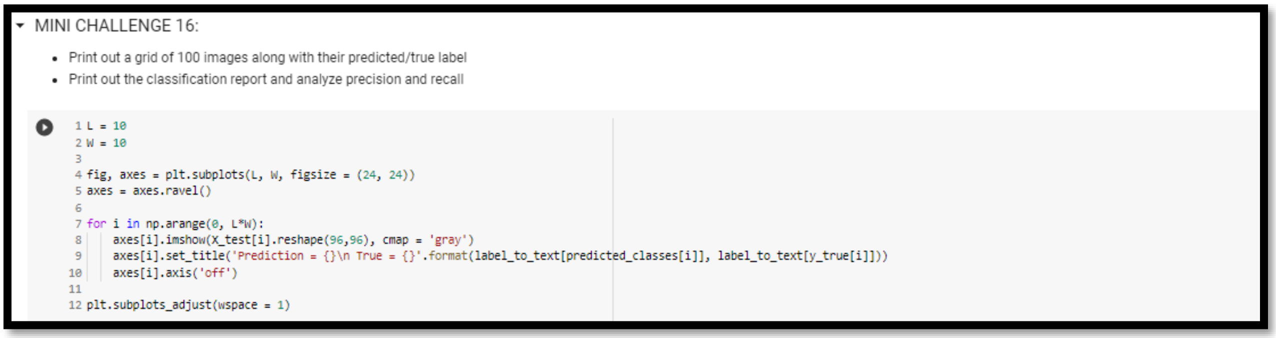
```
(1229,)
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_true, predicted_classes)
    plt.figure(figsize = (10, 10))
    sns.heatmap(cm, annot = True, cbar = False)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff018f12410>
```

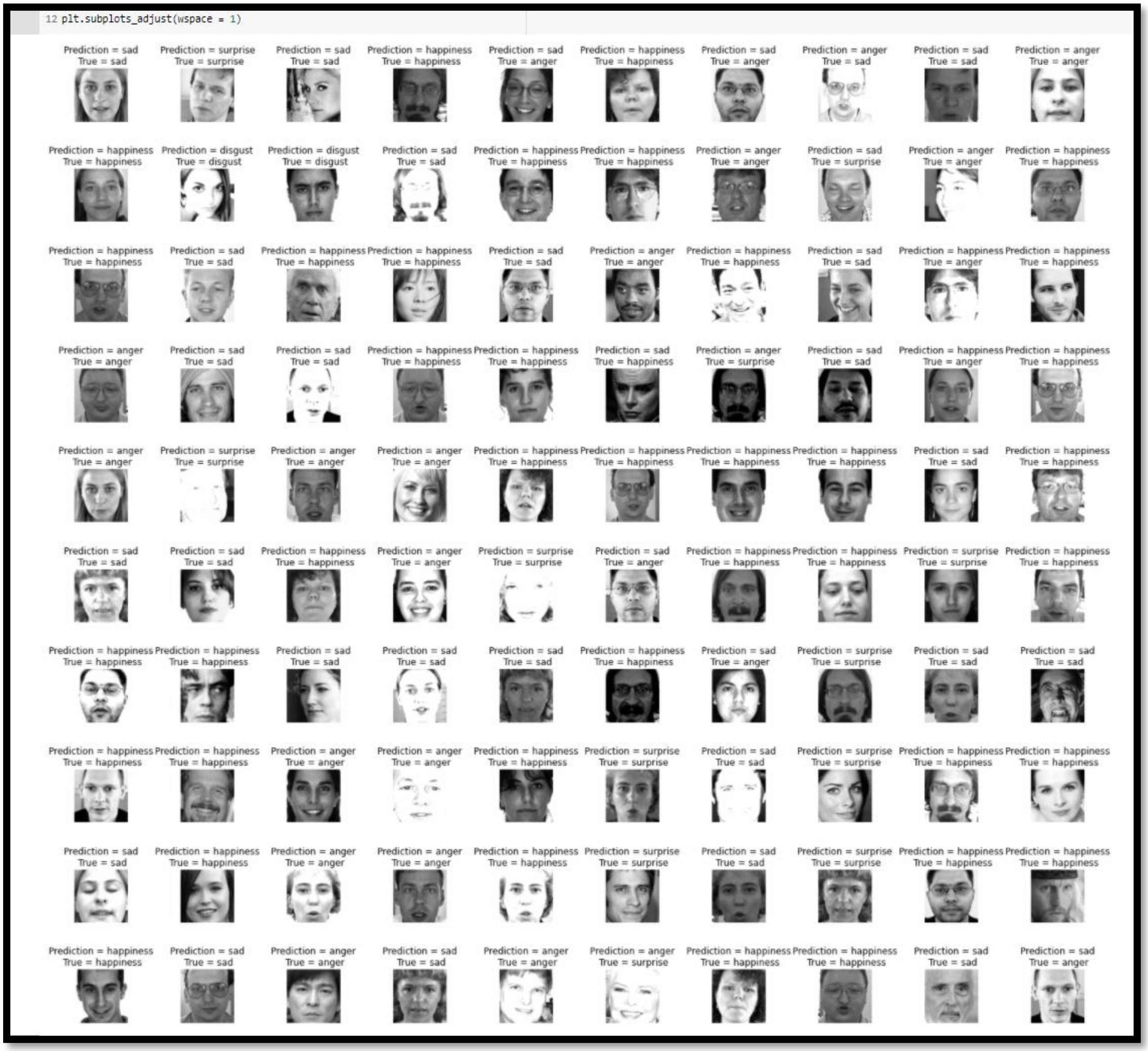
	0	1	2	3	4
0	2e+02	1	39	14	3
1	2	20	3	1	0
2	25	0	2.6e+02	13	2
3	6	0	13	4.3e+02	2
4	7	0	4	12	1.7e+02



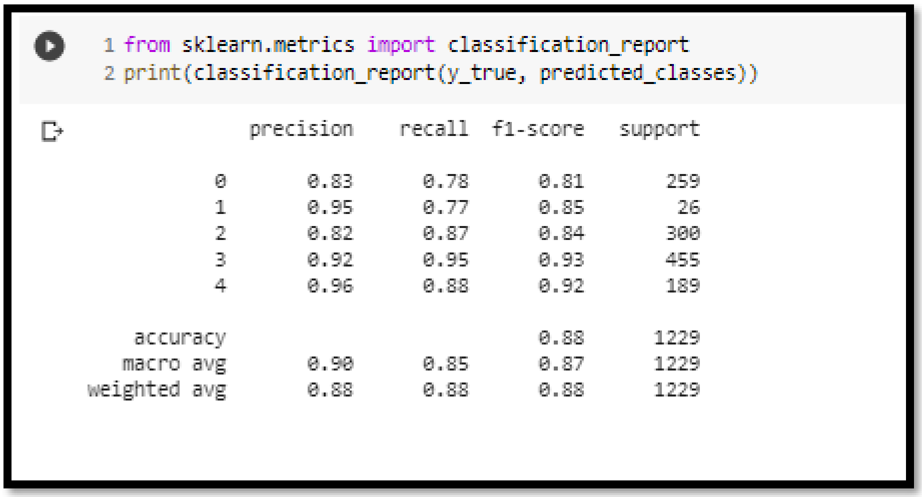
MINI CHALLENGE 16:

- Print out a grid of 100 images along with their predicted/true label
- Print out the classification report and analyze precision and recall

```
L = 10
W = 10

fig, axes = plt.subplots(L, W, figsize = (24, 24))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i].reshape(96,96), cmap = 'gray')
    axes[i].set_title('Prediction = {}\n True = {}'.format(label_to_text[predicted_classes[i]], label_to_text[y_true[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace = 1)
```

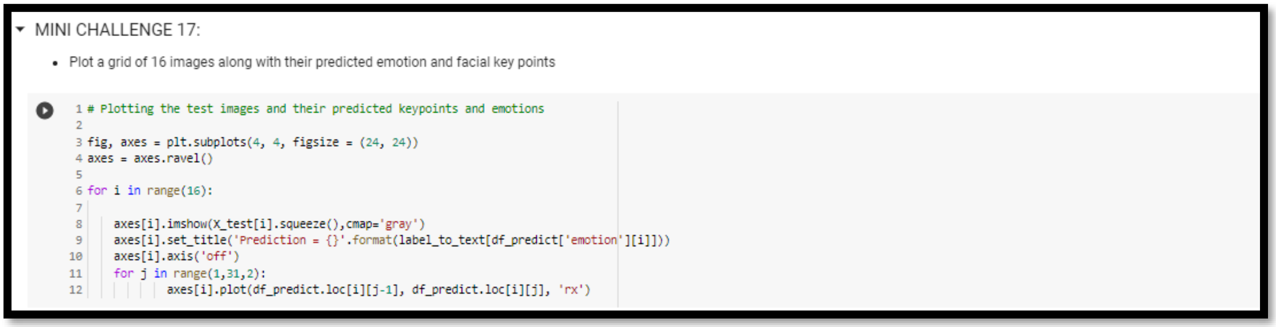


```
plt.subplots_adjust(wspace = 1)
```



```python
from sklearn.metrics import classification_report
print(classification_report(y_true, predicted_classes))
```

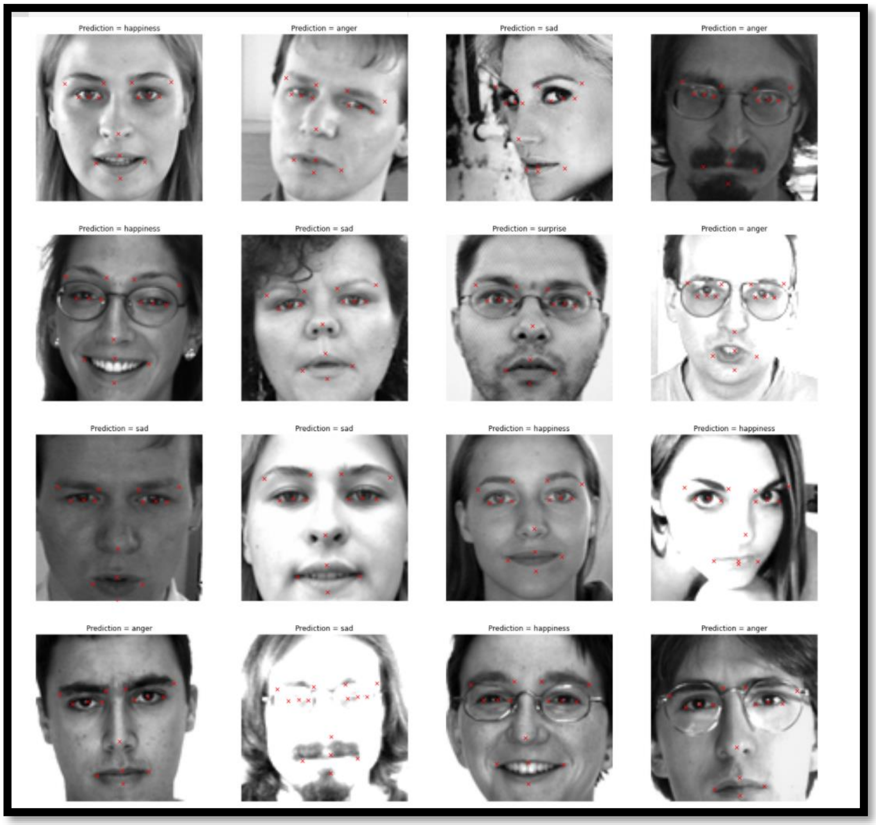
```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81       259
           1       0.95      0.77      0.85        26
           2       0.82      0.87      0.84       300
           3       0.92      0.95      0.93       455
           4       0.96      0.88      0.92       189

    accuracy                           0.88      1229
   macro avg       0.90      0.85      0.87      1229
weighted avg       0.88      0.88      0.88      1229
```



MINI CHALLENGE 17:

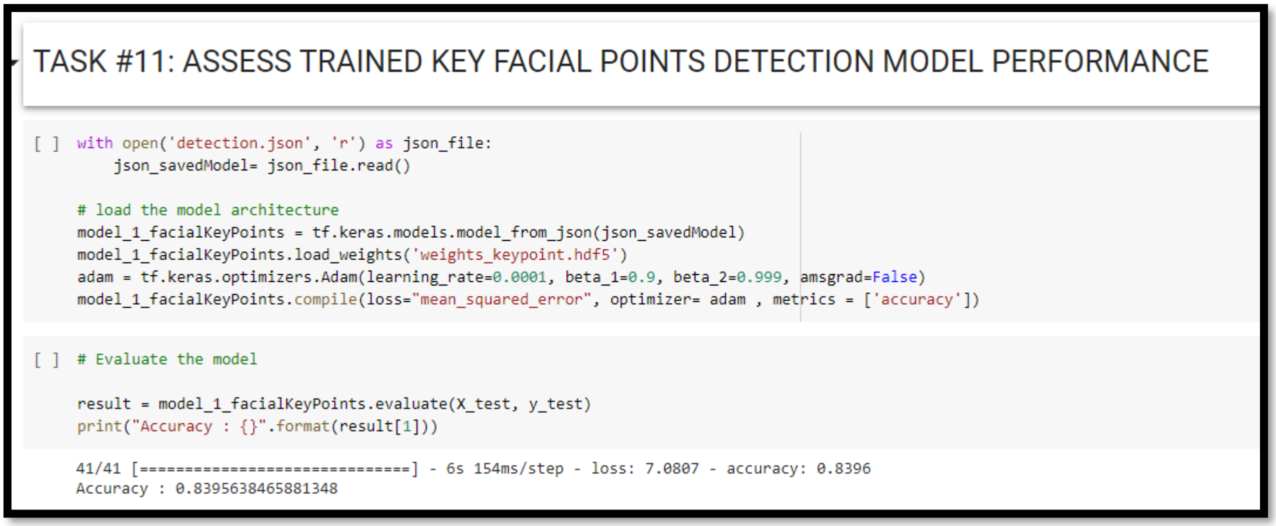
- Plot a grid of 16 images along with their predicted emotion and facial key points

```python
# Plotting the test images and their predicted keypoints and emotions

fig, axes = plt.subplots(4, 4, figsize = (24, 24))
axes = axes.ravel()

for i in range(16):

    axes[i].imshow(X_test[i].squeeze(),cmap='gray')
    axes[i].set_title('Prediction = {}'.format(label_to_text[df_predict['emotion'][i]]))
    axes[i].axis('off')
    for j in range(1,31,2):
        axes[i].plot(df_predict.loc[i][j-1], df_predict.loc[i][j], 'rx')
```



V. Result

The model after combining the two smaller models gave the following result in the assessing step of the model.

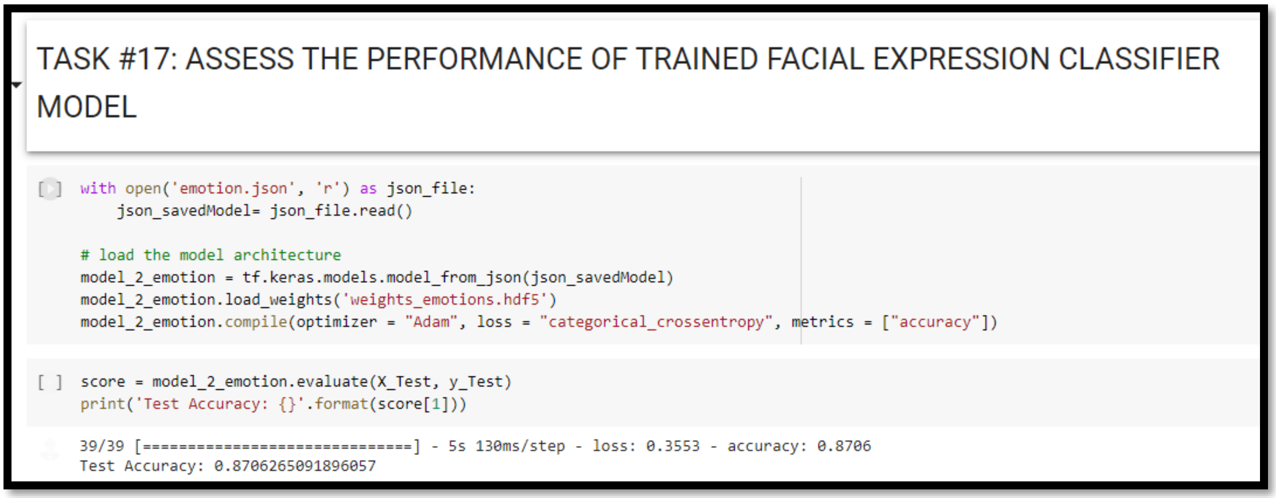


TASK #11: ASSESS TRAINED KEY FACIAL POINTS DETECTION MODEL PERFORMANCE

```
with open('detection.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_1_facialKeyPoints = tf.keras.models.model_from_json(json_savedModel)
model_1_facialKeyPoints.load_weights('weights_keypoint.hdf5')
adam = tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_1_facialKeyPoints.compile(loss="mean_squared_error", optimizer= adam , metrics = ['accuracy'])
```

```
# Evaluate the model

result = model_1_facialKeyPoints.evaluate(X_test, y_test)
print("Accuracy : {}".format(result[1]))
```

```
41/41 [==============================] - 6s 154ms/step - loss: 7.0807 - accuracy: 0.8396
Accuracy : 0.8395638465881348
```

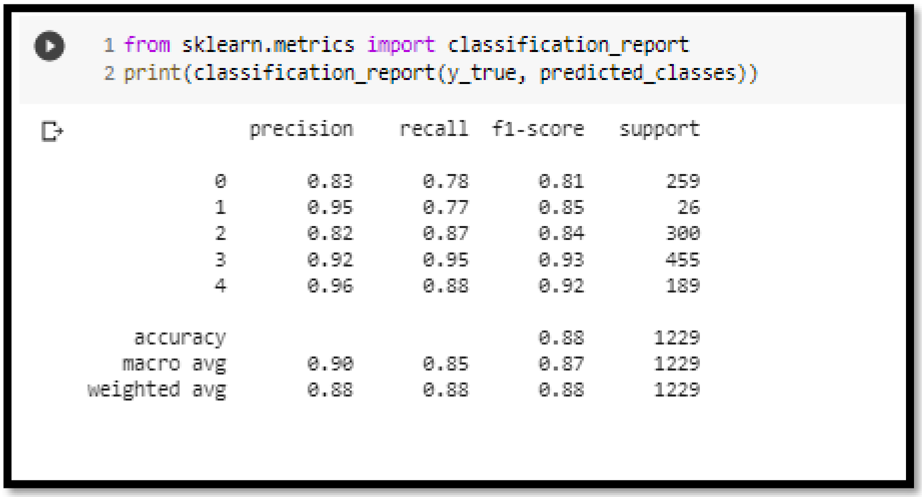


TASK #17: ASSESS THE PERFORMANCE OF TRAINED FACIAL EXPRESSION CLASSIFIER MODEL

```
with open('emotion.json', 'r') as json_file:
    json_savedModel= json_file.read()

# load the model architecture
model_2_emotion = tf.keras.models.model_from_json(json_savedModel)
model_2_emotion.load_weights('weights_emotions.hdf5')
model_2_emotion.compile(optimizer = "Adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
score = model_2_emotion.evaluate(X_Test, y_Test)
print('Test Accuracy: {}'.format(score[1]))
```

```
39/39 [==============================] - 5s 130ms/step - loss: 0.3553 - accuracy: 0.8706
Test Accuracy: 0.8706265091896057
```



```
from sklearn.metrics import classification_report
print(classification_report(y_true, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81       259
           1       0.95      0.77      0.85        26
           2       0.82      0.87      0.84       300
           3       0.92      0.95      0.93       455
           4       0.96      0.88      0.92       189

    accuracy                           0.88      1229
   macro avg       0.90      0.85      0.87      1229
weighted avg       0.88      0.88      0.88      1229
```

VI. Conclusion

Successfully implemented and completed this project/case study. As required in the problem statement got the same output. The model which I built by combining the 2 different models i.e., model 01 – Facial key points detection model and model 02 – Facial expression classification model, just performed as required in order to get the required output for the problem statement given to me.