Configuration Manual

MSc Research Project
MSc Data Analytics

Rajshri Pawar
Student ID: 22126571

School of Computing
National College of Ireland

Supervisor: Taimur Hafeez

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rajshri Pawar
Student ID:	22126571
Programme:	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Taimur Hafeez
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	868
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rajshri Pawar
22126571

1 Introduction

As the technology is rapidly changing and understanding Human emotions is a very difficult task and they changes in instance. A study to predict the emotions and to make them better by recommending songs and movies through the front-end.

1.1 General information

1.1.1 System information

Operating System: Windows 11 x64.
Version: 10.0.19044 Build 19044.
Processor: Intel(R) Core (TM) i7-9750H CPU @ 2.60GHz, 2592 MHz, 6 Core(s), 12 Logical Processor(s).
Installed Physical Memory (RAM): 16.0 GB.
Hard Disk: 500GB.

1.1.2 Platforms

Python(3.9): The python version used in the project is 3.9.
Google Collaboratory: An environment for developing the python code, which support GPU and TPU, gives faster execution.

1.2 Project Brief

DNN models were implemented to train the dataset for detecting the emotions through facial expressions.The Custom CNN, CNN and transfer learning models like VGG16 were used to get the desired accuracy for the system. For generating the dataset, OpenCV is used and half of the data is taken from kaggle.

1. Generated the images with OpenCV and downloaded the data from Kaggle which is an open site for accessing the Data without having any ethical issues.
Stored all the data in the specified emotions set, as we have 7 different emotions follows: Angry, Happy, Sad, Neutral, Disgust, Surprise, Fear.

2. Mediapipe is installed, which is an open-source framework used for various vision tasks. The built-in components allows to built a custom pipeline and intergrate the deep

learning models.

3. Integrating the google drive with the google collab for the data as the data is too large, 35000 images. It will ask for permission to access the drive for data.

4. Importing all the required libraries

```
[37] import cv2
import matplotlib.pyplot as plt
import pandas as pd
import mediapipe as mp
import numpy as np
import itertools
import tensorflow as tf
from tensorflow.keras import activations
from tensorflow.keras import losses
from tensorflow.keras.regularizers import l2, l1
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.optimizers import RMSprop, SGD, Adagrad, Adam
from tensorflow.keras.layers import Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization ,Activation
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras.layers import Input, MaxPooling2D, Flatten, Dense,InputLayer
from tensorflow.keras import regularizers
#/content/drive/MyDrive/Emotion_detection_data
```

Figure 1: Libraries

5. After getting the data performing the facial expressions landmark detection using the Face Mesh model. Facial Landmark defined like for example Left eye,right eye for the visualizations.

6. After landmarking, transferring the images to the specified path and the resizing it and again performing the landmarking with Gaussian blur for better image. After getting the output, visualizations are done using openCV.

```
mp_face_mesh = mp.solutions.face_mesh
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
LEFT_EYE = list(set(itertools.chain(*mp_face_mesh.FACEMESH_LEFT_EYE)))
RIGHT_EYE = list(set(itertools.chain(*mp_face_mesh.FACEMESH_RIGHT_EYE)))
LEFT_EYEBROW = list(set(itertools.chain(*mp_face_mesh.FACEMESH_LEFT_EYEBROW)))
RIGHT_EYEBROW = list(set(itertools.chain(*mp_face_mesh.FACEMESH_RIGHT_EYEBROW)))
LIPS = list(set(itertools.chain(*mp_face_mesh.FACEMESH_LIPS)))
CONTOURS = list(set(itertools.chain(*mp_face_mesh.FACEMESH_CONTOURS)))
OTHER = [1]
face_mesh = mp_face_mesh.FaceMesh(
                                    static_image_mode=True,
                                    max_num_faces=1,
                                    refine_landmarks=True,
                                    min_detection_confidence=0.5)
img=cv2.imread('/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection/test/fearful/im293.png')
#img=cv2.imread(df['image_path'][1])
#img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
if img is None: #added by rajshri
  print("Error: Could not load the image") #added by rajshri
else: #added by rajshri
  img = cv2.resize(img, (500, 500)) # Any size, just for visualization
  img = cv2.GaussianBlur(img, (3, 3), cv2.BORDER_DEFAULT)
  img_shape = img.shape[0]
  print("sahpe of the image it should be 500",img_shape)
  results = face_mesh.process(img)
  print("the results",results)
  annotated_image = img.copy()
  shape = [(lmk.x, lmk.y, lmk.z) for i, lmk in enumerate(results.multi_face_landmarks[0].landmark)]

  shape = np.array(shape)
  print(shape[1])
  print("======")
  print(len(shape))  #x,y,z #this after normalization
  shape = shape[LEFT_EYE + RIGHT_EYE + LEFT_EYEBROW + RIGHT_EYEBROW + LIPS + OTHER]
  print("the shape of the it ",shape.shape)
  for lmk in shape:
    cv2.circle(annotated_image, (int(lmk[0] * img_shape), int(lmk[1] * img_shape)), 2, (0, 0, 255))
  plt.imshow(annotated_image, interpolation='nearest')
```

Figure 2: Facial Landmark

```
sahpe of the image it should be 500 500
the results <class 'mediapipe.python.solution_base.SolutionOutputs'>
[ 0.52162093  0.67323411 -0.20950888]
======
478
the shape of the it  (93, 3)
```

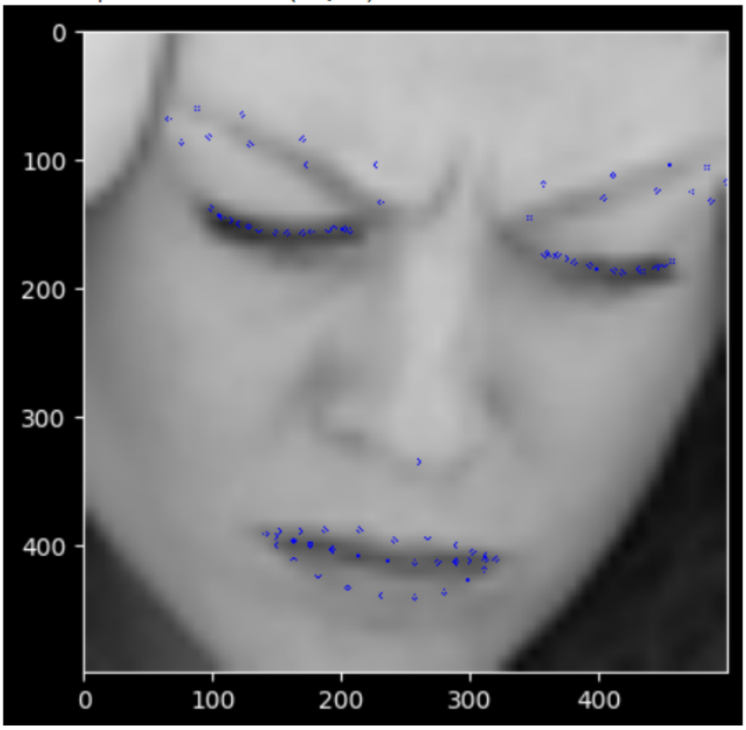

Figure 3: Facial Landmark

2 Preparation of Data set

1. Firstly setting the path for storing the specified data in the specified location. After that defining the 2D and 3D spaces.

2. Preparing a CSV file which will be containing the facial expression landmark Data. Storing all the images according to the specifically given directory for each emotion.

3. Loading the images again and then converting them into RGB formate to obtain the landmark predictions. After that calculating the euclidean distance in 2d and 3d from nose point.

4. Storing the dataFrame to a CSV file in the given mode as train or test set.

3 Model Creation

For the system, used models are Custom CNN, CNN and transfer learning model like VGG16.

1. Firstly, for creation of the model, using TensorFlow and Keras for the detection of the emotions.

2. After Importing the required libraries like: sequential, InputLayer, Conv2D MaxPooling2D, Flatten etc.

3. Set the parameters for image size and batch size and creating a generator for the

```python
import os
import cv2 # added by rajshri
import numpy as np # added by rajshri
import pandas as pd # added by rajshri

PTH = '/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection'
train_path = f'{PTH}/train'
test_path = f'{PTH}/test'

def euc2d(a, b): # in 2d dimension
    return np.sqrt( (a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1]) )

def euc3d(a, b):#in third dimension
    return np.sqrt( (a[0]-b[0])*(a[0]-b[0]) + (a[1]-b[1])*(a[1]-b[1]) + (a[2]-b[2])*(a[2]-b[2]) )

def prepare_csv(path, mode, face_mesh):

    emotions = os.listdir(path) #all directores of the train or the test the emotion exactly

    df = pd.DataFrame({
    #he choose the 92 becouse the range(5) => 0-4 so it will be acually a 5 number
    # *2  is to be x and y
    }, columns = [f"{i}" for i in range(92 * 2)] + ["y"])


    for i, emotion in enumerate(emotions): #also i we make it as labels
        images = os.listdir(f'{path}/{emotion}')
        for image in images:
            #pre process of the image
            img = cv2.imread(f"{path}/{emotion}/{image}")
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.GaussianBlur(img, (3, 3), cv2.BORDER_DEFAULT)

            results = face_mesh.process(img)

            if results.multi_face_landmarks:

                shape = [(lmk.x, lmk.y, lmk.z) for lmk in results.multi_face_landmarks[0].landmark]
                shape = np.array(shape) #all thing
                nose = shape[1] #the nose of the shape
                shape = shape[LEFT_EYE + RIGHT_EYE + LEFT_EYEBROW + RIGHT_EYEBROW + LIPS]
                #the interested indexes from the landmark

                distances2d = [round(euc2d(nose, x), 6) for x in shape]
                distances3d = [round(euc3d(nose, x), 6) for x in shape]

                df.loc[len(df)] = distances2d + distances3d + [i] #need more inveseitigation

    df.to_csv(f'{mode}.csv', index=False)


prepare_csv(train_path, 'train', face_mesh)
prepare_csv(test_path, 'test', face_mesh)
```

Figure 4: Data preparation

training data. After that the data generator for validation and training used to train the CNN model

```python
# Import necessary libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Set parameters
batch_size = 32
image_size = (48, 48)

# Create data generators
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
train_generator = datagen.flow_from_directory(
    '/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection/train/',
    target_size=image_size,
    batch_size=batch_size,
    color_mode="rgb",
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    '/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection/train/',
    target_size=image_size,
    batch_size=batch_size,
    color_mode="rgb",
    class_mode='categorical',
    subset='validation'
)
```

```
Found 20106 images belonging to 7 classes.
Found 8613 images belonging to 7 classes.
```

Figure 5: Data Generators

4. In the above image the results is showing that we have 7 classes of emotions containing the data.

Model 1 - CNN-1

```python
# Build a CNN model
cnn_model = Sequential()
cnn_model.add(InputLayer(input_shape=(image_size[0], image_size[1], 3)))

cnn_model.add(Conv2D(32, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(256, activation='relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(7, activation='softmax'))

cnn_model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.0001),
                  metrics=['accuracy'])

cnn_model.summary()


cnn_history = cnn_model.fit(train_generator, epochs=10,
                            steps_per_epoch=train_generator.n // batch_size,
                            validation_data=validation_generator,
                            validation_steps=validation_generator.n // batch_size)
```

Figure 6: Building CNN model

5. After training the CNN nodel using the TensorFlow and Keras for detection of 7 different emotion. we got the below results

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 46, 46, 32)        896

 max_pooling2d_18 (MaxPoolin  (None, 23, 23, 32)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 21, 21, 64)        18496

 max_pooling2d_19 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 conv2d_20 (Conv2D)          (None, 8, 8, 128)         73856

 max_pooling2d_20 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 2048)              0

 dense_23 (Dense)            (None, 512)               1049088

 dropout_14 (Dropout)        (None, 512)               0

 dense_24 (Dense)            (None, 256)               131328

 dropout_15 (Dropout)        (None, 256)               0

 dense_25 (Dense)            (None, 7)                 1799

=================================================================
Total params: 1,275,463
Trainable params: 1,275,463
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
628/628 [==============================] - 152s 240ms/step - loss: 1.8346 - accuracy: 0.2307 - val_loss: 1.7878 - val_accuracy: 0.2548
Epoch 2/10
628/628 [==============================] - 174s 277ms/step - loss: 1.7545 - accuracy: 0.2878 - val_loss: 1.6302 - val_accuracy: 0.3742
Epoch 3/10
628/628 [==============================] - 181s 289ms/step - loss: 1.6107 - accuracy: 0.3718 - val_loss: 1.5324 - val_accuracy: 0.4058
Epoch 4/10
628/628 [==============================] - 178s 283ms/step - loss: 1.5268 - accuracy: 0.4068 - val_loss: 1.4659 - val_accuracy: 0.4421
Epoch 5/10
628/628 [==============================] - 173s 275ms/step - loss: 1.4648 - accuracy: 0.4385 - val_loss: 1.4154 - val_accuracy: 0.4606
Epoch 6/10
628/628 [==============================] - 152s 243ms/step - loss: 1.4192 - accuracy: 0.4607 - val_loss: 1.3777 - val_accuracy: 0.4757
Epoch 7/10
628/628 [==============================] - 158s 252ms/step - loss: 1.3693 - accuracy: 0.4820 - val_loss: 1.3370 - val_accuracy: 0.4865
Epoch 8/10
628/628 [==============================] - 167s 267ms/step - loss: 1.3291 - accuracy: 0.4949 - val_loss: 1.3081 - val_accuracy: 0.4977
Epoch 9/10
628/628 [==============================] - 153s 243ms/step - loss: 1.2940 - accuracy: 0.5112 - val_loss: 1.3108 - val_accuracy: 0.4952
Epoch 10/10
628/628 [==============================] - 144s 229ms/step - loss: 1.2567 - accuracy: 0.5273 - val_loss: 1.2733 - val_accuracy: 0.5180
```

Figure 7: Output

6. The results showing the layers used in the CNN model.

Sequential : Linear stack of layers. Common way to create the models where the layer can be added on at a time.

Input layer: this layer will tell about the input shape for the model.

Conv2D: (Convolutional Layer) which performs the convolution operation on input images.

MaxPooling2D: It reduces the spatial dimensions, which helps in concentrating on the important features.

Flatten: It transfers the results from previous layer to one dimensional array, which is required for the densely connected layer.

Dense: this layer is fully connected layer. where each neuron is connected to every other neuron. It will mostly be used in the final layer.

Dropout: it helps to prevent from overfitting.

Output layer: The output will be having the 7 neurons.The softmax function is used for activation.

Model 2 - CNN-2

Bulding the CNN -2 model with more layers added.

```python
from tensorflow.keras.layers import BatchNormalization

# Build a CNN model with additional changes
cnn_model = Sequential()
cnn_model.add(InputLayer(input_shape=(image_size[0], image_size[1], 3)))

cnn_model.add(Conv2D(32, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))

cnn_model.add(Dense(256, activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(Dropout(0.5))

cnn_model.add(Dense(7, activation='softmax'))

cnn_model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.0001),
                  metrics=['accuracy'])

# Train the modified CNN model
cnn_history_2 = cnn_model.fit(train_generator, epochs=10,
                              steps_per_epoch=train_generator.n // batch_size,
                              validation_data=train_generator,
                              validation_steps=train_generator.n // batch_size)
```

Figure 8: CNN- 2 Model Build

The results of CNN-2

```
Epoch 1/10
628/628 [==============================] - 206s 322ms/step - loss: 2.6754 - accuracy: 0.2244 - val_loss: 1.7405 - val_accuracy: 0.3454
Epoch 2/10
628/628 [==============================] - 196s 312ms/step - loss: 2.2023 - accuracy: 0.3079 - val_loss: 1.5947 - val_accuracy: 0.4117
Epoch 3/10
628/628 [==============================] - 183s 292ms/step - loss: 1.9634 - accuracy: 0.3572 - val_loss: 1.5376 - val_accuracy: 0.4226
Epoch 4/10
628/628 [==============================] - 200s 319ms/step - loss: 1.8260 - accuracy: 0.3838 - val_loss: 1.4433 - val_accuracy: 0.4606
Epoch 5/10
628/628 [==============================] - 199s 316ms/step - loss: 1.6748 - accuracy: 0.4143 - val_loss: 1.4340 - val_accuracy: 0.4750
Epoch 6/10
628/628 [==============================] - 197s 314ms/step - loss: 1.5917 - accuracy: 0.4335 - val_loss: 1.3669 - val_accuracy: 0.4880
Epoch 7/10
628/628 [==============================] - 192s 305ms/step - loss: 1.4958 - accuracy: 0.4560 - val_loss: 1.3538 - val_accuracy: 0.4928
Epoch 8/10
628/628 [==============================] - 205s 327ms/step - loss: 1.4176 - accuracy: 0.4813 - val_loss: 1.4763 - val_accuracy: 0.4375
Epoch 9/10
628/628 [==============================] - 214s 341ms/step - loss: 1.3514 - accuracy: 0.4961 - val_loss: 1.3368 - val_accuracy: 0.5016
Epoch 10/10
628/628 [==============================] - 208s 331ms/step - loss: 1.2895 - accuracy: 0.5211 - val_loss: 1.2917 - val_accuracy: 0.5189
```

Figure 9: CNN-1 results

Model 3 - VGG16

1. Bulding the VGG16 model for the emotion detection.
2. Image size and batch size parameters is set.
3. Data generators are build for training and validation data and the the data is divided into training and validation subset.
4. VGG16 model will be loading with the data weights which are pre-trained on the ImageNet dataset.
5. The custom layers are added with the VGG16.then the model will be compiled.

```
[ ] # Import necessary libraries
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import InputLayer, Dense, Dropout, GlobalAveragePooling2D
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.applications import VGG19
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    import matplotlib.pyplot as plt


    # Set parameters
    batch_size = 32
    image_size = (48, 48)

    # Create data generators
    datagen = ImageDataGenerator(rescale=1./255, validation_split=0.3)
    train_generator = datagen.flow_from_directory(
        '/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection/train/',
        target_size=image_size,
        batch_size=batch_size,
        color_mode="rgb",
        class_mode='categorical',
        subset='training'
    )

    validation_generator = datagen.flow_from_directory(
        '/content/drive/MyDrive/Emotion_detection_data-20231205T222356Z-001/Emotion_detection_data/Emotion_detection/train/',
        target_size=image_size,
        batch_size=batch_size,
        color_mode="rgb",
        class_mode='categorical',
        subset='validation'
    )

    # Build a VGG19 transfer learning model
    base_model = VGG19(input_shape=(image_size[0], image_size[1], 3), include_top=False, weights='imagenet')
    base_model.trainable = False

    transfer_model = Sequential()
    transfer_model.add(base_model)
    transfer_model.add(GlobalAveragePooling2D())
    transfer_model.add(Dense(512, activation='relu'))
    transfer_model.add(Dropout(0.5))
    transfer_model.add(Dense(256, activation='relu'))
    transfer_model.add(Dropout(0.5))
    transfer_model.add(Dense(7, activation='softmax'))

    transfer_model.compile(loss='categorical_crossentropy',
                           optimizer=Adam(learning_rate=0.0001),
                           metrics=['accuracy'])


    transfer_model.summary()
```

Figure 10: VGG16 model

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense_20 (Dense)            (None, 512)               262656

 dropout_12 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 256)               131328

 dropout_13 (Dropout)        (None, 256)               0

 dense_22 (Dense)            (None, 7)                 1799

=================================================================
Total params: 20,420,167
Trainable params: 395,783
Non-trainable params: 20,024,384
_________________________________________________________________
```

Figure 11: VGG16 structure

Front end of the prototype

```python
# Function to preprocess the input image
def preprocess_image(img_path, target_size):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Rescale to [0,1]
    return img_array

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})
```

Figure 12: Code snippet for fron-end execution

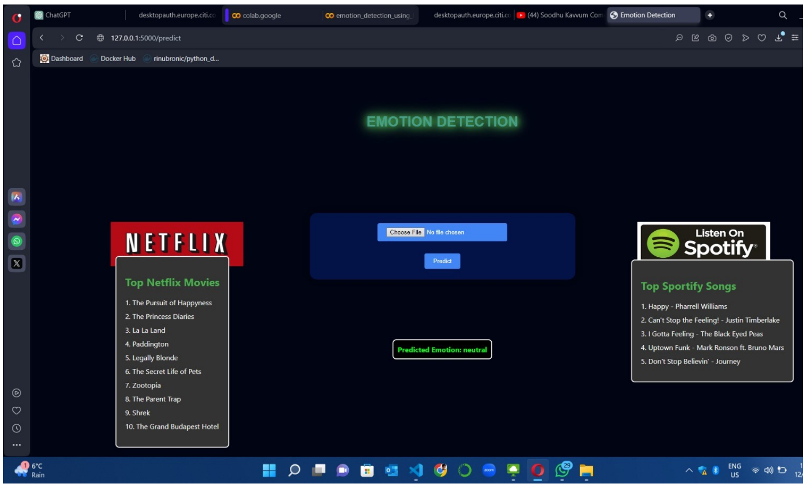

Figure 13: Front-end of the prototype

References