A PROJECT REPORT

on

"SONG SUGGESTION BASED ON FACIAL EXPRESSIONS"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE & ENGINEERING

BY

Advaita Krishna	**1905155**

UNDER THE GUIDANCE OF
Prof. Lalit Kumar Vashishtha

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024
July 2022

KIIT Deemed to be University

School of Computer Engineering

Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

“SONG SUGGESTION BASED ON FACIAL EXPRESSIONS“

submitted by

Advaita Krishna	1905155

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during the year 2022-2023, under our guidance.

Date: 06/07/2022

Prof. Lalit Kumar Vashishtha
Project Mentor

Acknowledgements

We are profoundly grateful to Prof Lalit Kumar Vashishtha**,** Dean, Training & Placement for his expert guidance and continuous encouragement throughout to see that this project meets its target since its commencement to its completion.

I also thank my friends for the immense support and help during this project.

Advaita Krishna

ABSTRACT

A user's emotion or mood can be detected by his/her facial expressions. These expressions can be derived from uploading the images in the system. A lot of research is being conducted in the field of Computer Vision and Machine Learning (ML), where machines are trained to identify various human emotions or moods. Machine Learning provides various techniques through which human emotions can be detected. One such technique is to use the Keras, which generates a small size trained model and makes ML easier.

Music is a great connector. It unites us across markets, ages, backgrounds, languages, preferences, political leanings and income levels. Music players and other streaming apps have a high demand as these apps can be used anytime, anywhere and can be combined with daily activities, travelling , sports, etc. With the rapid development of mobile networks and digital multimedia technologies, digital music has become the mainstream consumer content sought by many young people. People often use music as a means of mood regulation, specifically to change a bad mood, increase energy level or reduce tension. Also, listening to the right kind of music at the right time may improve mental health. Thus, human emotions have a strong relationship with music.

In our proposed system, a mood-based music player is created which performs real time mood detection and plays songs as per detected mood. An important benefit of incorporating mood detection is customer satisfaction. The objective of this system is to analyze the user's image, predict the expression of the user and suggest songs suitable to the detected mood.

Keyword: Face Recognition, Image Processing, Computer Vision, Emotion Detection, Music, Mood detection

Contents

List of Figures

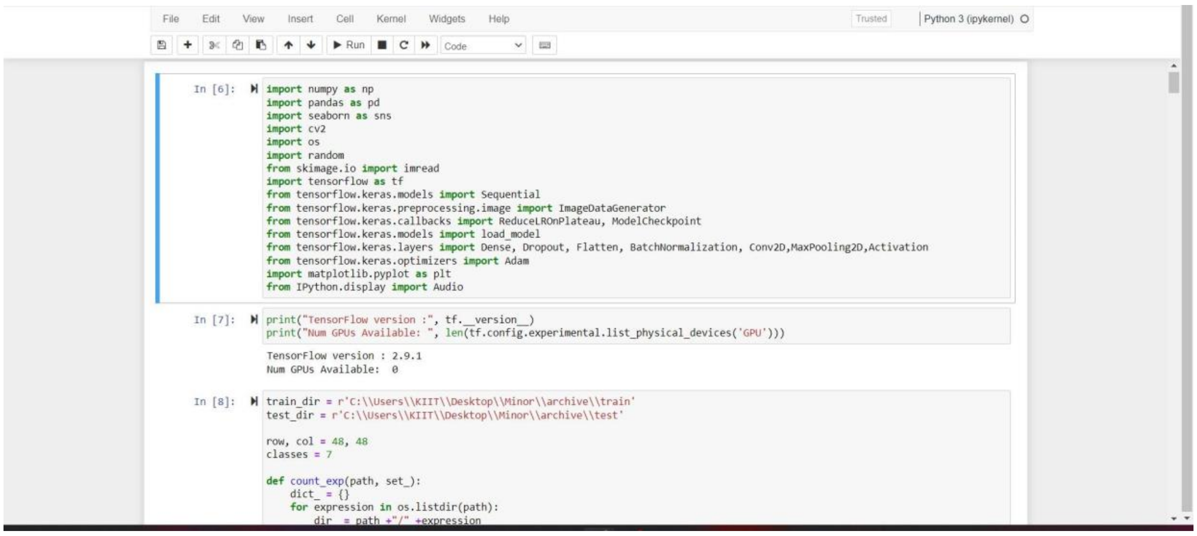
Chapter 1

Introduction

Human emotions can be broadly classified as: fear, disgust, anger, surprise, sad, happy and neutral. A large number of other emotions such as cheerful (which is a variation of happy) and contempt (which is a variation of disgust) can be categorized under this umbrella of emotions. These emotions are very subtle. Facial muscle contortions are very minimal, and detecting these differences can be very challenging as even a small difference results in different expressions. Also, expressions of different or even the same people might vary for the same emotion, as emotions are hugely context dependent. While the focus can be on only those areas of the face which display a maximum of emotions like around the mouth and eyes, how these gestures are extracted and categorized is still an important question. Neural networks and machine learning have been used for these tasks and have obtained good results. Machine learning algorithms have proven to be very useful in pattern recognition and classification, and hence can be used for mood detection as well.

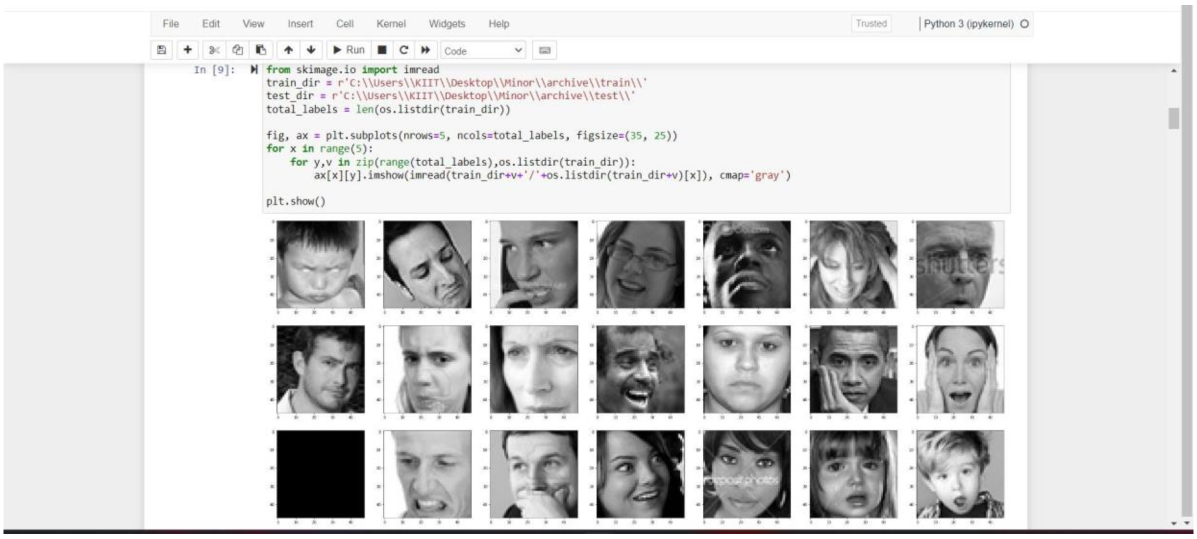
With the development of digital music technology, the development of a personalized music recommendation system which recommends music for users is essential. It is a big challenge to provide recommendations from the large data available on the internet. E-commerce giants like Amazon, EBay provide personalized recommendations to users based on their taste and history while companies like Spotify, Pandora use Machine Learning and Deep Learning techniques for providing appropriate recommendations. There has been some work done on personalized music recommendation to recommend songs based on the user's preference. There exist two major approaches for the personalized music recommendation. One is the content based filtering approach which analyzes the content of music that users liked in the past and recommends the music with relevant content. The main drawback of this approach is that the model can only make recommendations based on existing interests of the user. In other words, the model has limited ability to expand on the users' existing interests. The other approach is the collaborative filtering approach which recommends music that a peer group of similar preference likes. Both recommendation approaches are based on the user's preferences observed from the listening behavior. The major drawback of this approach is the popularity bias problem: popular (i.e., frequently rated) items get a lot of exposure while less popular ones are under-represented in the recommendations. Generally, a hybrid approach is implemented in which both content and collaborative techniques are combined to extract maximum accuracy and to overcome drawbacks of both types.

In this work, the aim is to create a music recommendation system/music player which will detect the user's face, identify the current mood and then recommend a song based on the detected mood.



```
In [6]:  import numpy as np
         import pandas as pd
         import seaborn as sns
         import cv2
         import os
         import random
         from skimage.io import imread
         import tensorflow as tf
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.preprocessing.image import ImageDataGenerator
         from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
         from tensorflow.keras.models import load_model
         from tensorflow.keras.layers import Dense, Dropout, Flatten, BatchNormalization, Conv2D,MaxPooling2D,Activation
         from tensorflow.keras.optimizers import Adam
         import matplotlib.pyplot as plt
         from IPython.display import Audio

In [7]:  print("TensorFlow version :", tf.__version__)
         print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))

         TensorFlow version : 2.9.1
         Num GPUs Available:  0

In [8]:  train_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\train'
         test_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\test'

         row, col = 48, 48
         classes = 7

         def count_exp(path, set_):
             dict_ = {}
             for expression in os.listdir(path):
                 dir_ = path +"/" +expression
```

Figure 1.1: Importing Libraries



```
In [9]:  from skimage.io import imread
         train_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\train\\'
         test_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\test\\'
         total_labels = len(os.listdir(train_dir))

         fig, ax = plt.subplots(nrows=5, ncols=total_labels, figsize=(35, 25))
         for x in range(5):
             for y,v in zip(range(total_labels),os.listdir(train_dir)):
                 ax[x][y].imshow(imread(train_dir+v+'/'+os.listdir(train_dir+v)[x]), cmap='gray')

         plt.show()
```

Figure 1.2: Importing images

Chapter 2

Basic Concepts

1. **Deep Learning based Facial Expression Recognition using Keras:** Using this algorithm, up to six distinct facial emotions can be detected in real time. It runs on top of a Convolutional Neural Network (CNN) that is built with the help of Keras whose backend is TensorFlow in Python. The facial emotions that can be detected and classified by this system are Happy, Sad, Anger, Surprise, Disgusting, and Neutral. OpenCV is used for image processing tasks where a face is identified from a live webcam feed which is then processed and fed into the trained neural network for emotion detection. Deep learning based facial expression recognition techniques bring down to a greater extent, the dependency on face-physics based models and other pre-processing techniques by enabling lengthwise learning to occur in the pipeline directly from the input images.

2. **Hybrid approach of Music Recommendation:** There are several drawbacks to relying solely on collaborative filtering to recommend music. The biggest problem is the "Cold Start." Music tracks are only tagged as often as listeners are discovering or listening to them. In other words, there are little or no available 'tags' to describe new music or music that has not been discovered yet. Additionally, listeners are more willing to supply tags for songs they enjoy most than for songs they mildly enjoy or do not enjoy at all.

3. **Viola–Jones object detection framework:** The Viola-Jones algorithm is a widely used mechanism for object detection. The main property of this algorithm is that training is slow, but detection is fast. This algorithm uses Haar basis feature filters, so it does not use multiplications. The efficiency of the Viola-Jones algorithm can be significantly increased by first generating the integral image.

Chapter 3

Problem Statement / Requirement Specifications

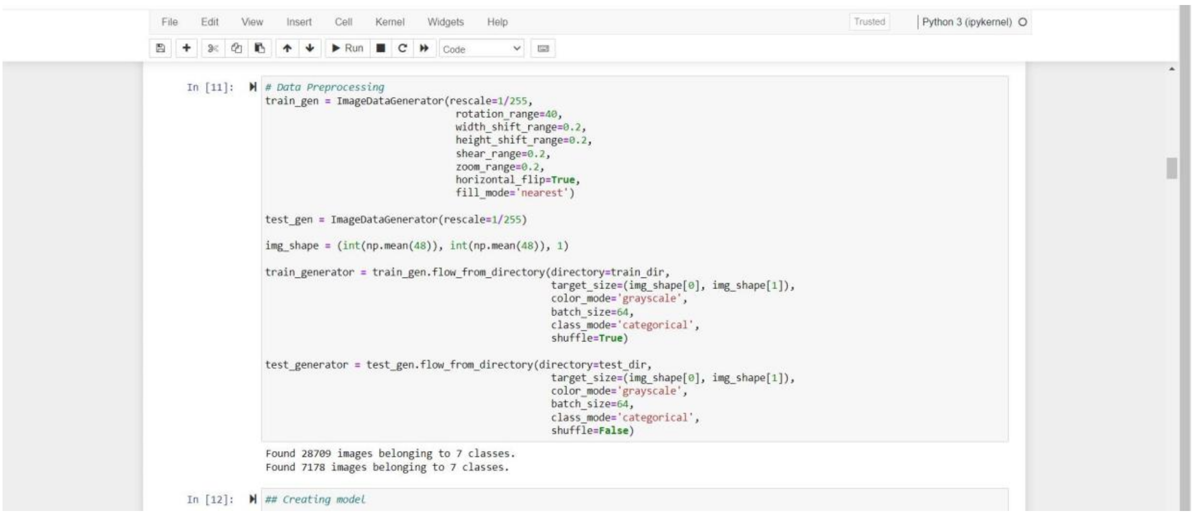
The main goal is suggesting the best set of options to the user. For a specific user we had their song history frequency list liked songs. From all this information we had to predict what songs the user might like then the question comes: how can we use all this information to achieve our goal. As it not a straightforward task to find the relevance between various songs it might be possible that one song which looks similar to other may be completely different and users may dislike that song or may be that song is not of users taste there are lots of user around the world and lots of songs so making a relevance between songs and users is a tedious task.

OBJECTIVES-

1. To implement a Mood Based Music Recommendation System using Face Detection.
2. To develop a personalized system, where the user's current emotion is analyzed using Machine Learning Algorithm.
3. Calculating the overall score, the user input is fed into a model which classifies whether the user's response was 'Happy', 'Neutral','Disgusting', 'Angry','Surprise' or 'Sad'.
4. To play the imported song.

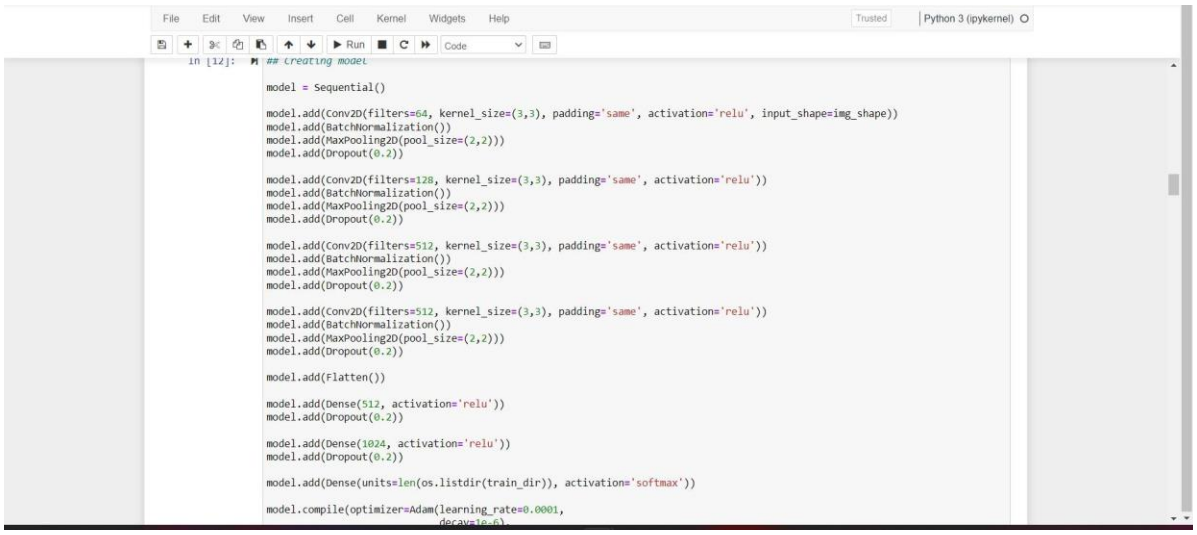
3.1 Project Planning

In a particular system ,OpenCV,Pycharm IDE(Jupyter), Anaconda and Python 3.6 softwares were used to test the functionality and Viola-Jones and haar cascade algorithms were used for face detection. Similarly, Kaggle dataset and VGG (Visual Geometry Group) were used with CNN (Convolution Neural Network) model which was designed with an accuracy of 75%, for face recognition and classification that validated the performance measures. However, the results proved that the network architecture designed had better advancements than existing algorithms. The popularity and ease of access for the original Kaggle dataset this is seen as a very valuable addition to the already existing corpora. It was also stated that for a fully automatic system to be robust for all expressions in a myriad of realistic scenarios, more data is required. For this to occur very large reliably coded datasets across a wide array of visual variabilities are required (at least 5 to 10k examples for each action) which would require a collaborative research effort from various institutions.It was observed in a cross-database experiment that raw features worked best with Logistic Regression for testing RaFD (Radboud Faces Database) database and Mobile images dataset. The accuracy achieved was 66% and 36% respectively for both using CK+ dataset as a training set.



```
In [11]:  # Data Preprocessing
          train_gen = ImageDataGenerator(rescale=1/255,
                                         rotation_range=40,
                                         width_shift_range=0.2,
                                         height_shift_range=0.2,
                                         shear_range=0.2,
                                         zoom_range=0.2,
                                         horizontal_flip=True,
                                         fill_mode='nearest')

          test_gen = ImageDataGenerator(rescale=1/255)

          img_shape = (int(np.mean(48)), int(np.mean(48)), 1)

          train_generator = train_gen.flow_from_directory(directory=train_dir,
                                                          target_size=(img_shape[0], img_shape[1]),
                                                          color_mode='grayscale',
                                                          batch_size=64,
                                                          class_mode='categorical',
                                                          shuffle=True)

          test_generator = test_gen.flow_from_directory(directory=test_dir,
                                                          target_size=(img_shape[0], img_shape[1]),
                                                          color_mode='grayscale',
                                                          batch_size=64,
                                                          class_mode='categorical',
                                                          shuffle=False)

          Found 28709 images belonging to 7 classes.
          Found 7178 images belonging to 7 classes.

In [12]:  ## Creating model
```

Figure 3.1.1: Data preprocessing of the dataset



```
In [12]:  ## Creating model

          model = Sequential()

          model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu', input_shape=img_shape))
          model.add(BatchNormalization())
          model.add(MaxPooling2D(pool_size=(2,2)))
          model.add(Dropout(0.2))

          model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
          model.add(BatchNormalization())
          model.add(MaxPooling2D(pool_size=(2,2)))
          model.add(Dropout(0.2))

          model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', activation='relu'))
          model.add(BatchNormalization())
          model.add(MaxPooling2D(pool_size=(2,2)))
          model.add(Dropout(0.2))

          model.add(Conv2D(filters=512, kernel_size=(3,3), padding='same', activation='relu'))
          model.add(BatchNormalization())
          model.add(MaxPooling2D(pool_size=(2,2)))
          model.add(Dropout(0.2))

          model.add(Flatten())

          model.add(Dense(512, activation='relu'))
          model.add(Dropout(0.2))

          model.add(Dense(1024, activation='relu'))
          model.add(Dropout(0.2))

          model.add(Dense(units=len(os.listdir(train_dir)), activation='softmax'))

          model.compile(optimizer=Adam(learning_rate=0.0001,
                                       decay=1e-6),
```

Figure 3.1.2: Creating model

3.2 Project Analysis

Before the project, we planned out our idea, and in the process, decided on the songs that were going to be used, the mood expressions that were going to be detected, and the frameworks that we were going to incorporate.We were working on training the ML model and making the routes on the backend.

It will first try to locate the person's face utilizing the Haar Cascade machine learning algorithm, then classify it using a trained CNN. CNN was trained on an open-source

Kaggle dataset provided by a competition named "Challenges in Representation Learning: Facial Expression Recognition Challenge." CNN is able to classify emotions as: happy, sad, angry, disgust,surprise,neutral. Depending on the detected mood, the music player will then recommend an appropriate song to play.

3.3 System Design

3.3.1 Design Constraints

A. Hardware Requirements-

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The hardware requirements required for this project are:

• Minimum 4 Gigabyte (GB) RAM (used for processing)

• 100 MB Memory space (approximate value)

B. Software Requirements-

Software Requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed. The software requirements that are required for this project are:

• Python 3.6

• OpenCV 3.1

• PyCharm IDE (Jupyter)

3.3.2 System Architecture OR Block Diagram

A. System Architecture

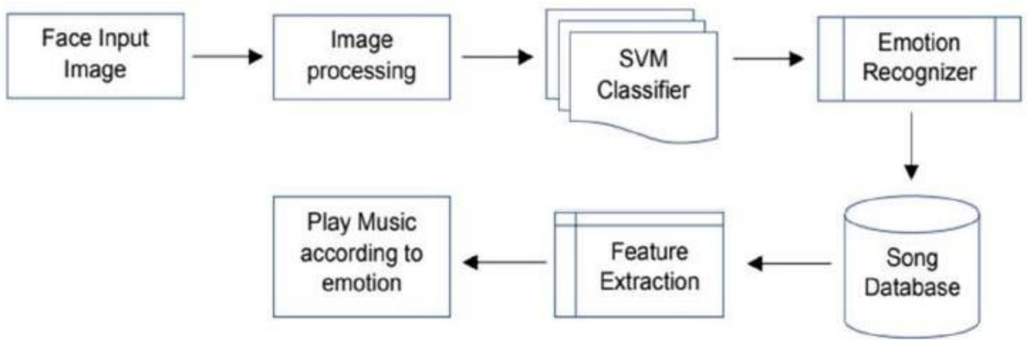


Figure 3.3.2.1: System Architecture

The proposed framework is first prepared to distinguish a face from a static picture. When the information picture is perceived, the picture is handled. The picture is exposed to SVM classifiers for subtleties to perceive the feeling displayed by the face. The subtleties recuperated from the image are utilized by the feeling classifier to discover feeling. The song database and feature extraction module function simultaneously. The songs are disintegrated into several music pieces and the mood of the song is recognized. The songs are stored based on the mood detected. Once the emotion recognizer reports the mood, the songs pertaining to the mood are played by the music player.

B. Data flow Diagram

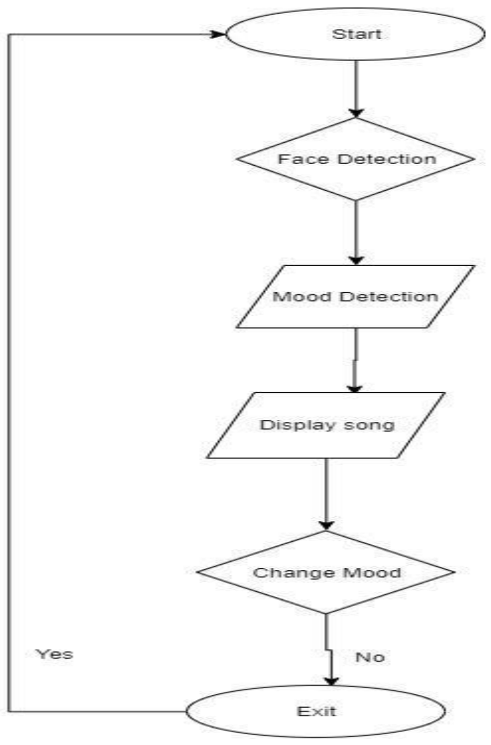


Figure 3.3.2.2: Data Flow Diagram of the system

Chapter 4
Implementation

4.1 Methodology OR Proposal

A. Convolutional Neural Networks- Convolutional Neural Networks (CNN) are feed-forward networks that were developed for image classification problems, and it uses feature learning where it takes 2-D input representing an image's pixels and color channels. This process can also be applied to 1-D sequences of data inputs. The pre-processing needed in a CNN is much lower as compared to other classification algorithms. It can learn the filters/characteristics with enough training given to them. In a CNN model, each input image is passed through a series of convolution layers with filters, also known as Kernals, pooling layers, fully connected layers (FC), and an activation function such as Softmax function is used to classify an object and bring the values within the range [0, 1]. The most important role of a CNN is to take in image inputs and reduce them in size without losing their important features which help with a good prediction of the model.

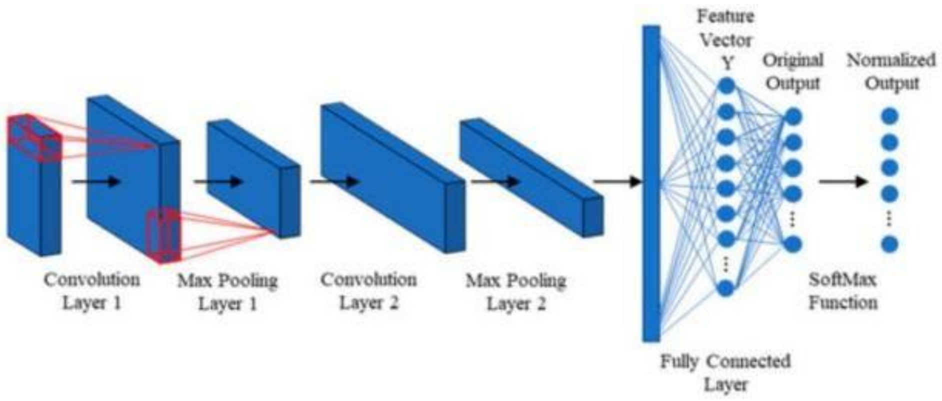


Figure 4.1.1: 1 D Convolutional Neural Network Architecture

B. OpenCV- Computer vision is a part of Artificial Intelligence (AI), that enables computers to gain meaningful insights from the data provided to it (the data can be in the form of images, videos, or any other visual input) and make relevant decisions based on the knowledge it acquired from the given data. OpenCV stands for Open-Source Computer Vision (Library). It is the most common and popularly used, well-documented Computer Vision library. OpenCV is an open-source library that incorporates numerous computer vision algorithms. OpenCV increases computational efficiency and assists with real-time applications. Face Feature Extraction Pictures are spoken to as weighted eigen vectors that are consolidated and known as "Eigenfaces''. One of the focal points taken by Eigenfaces is the comparability between the pixels among pictures by methods for their covariance network. Following are the means required to perceive the outward appearances utilizing this Eigenfaces approach: Eigen Faces: Not all the parts of the face are important for emotion recognition. This key fact is considered to be important and useful. Face recognition techniques focus on recognizing eyes, nose, cheek and forehead and how the change with respect to each other. Overall, the areas with maximum changes, mathematically, areas with high variations are targeted. When multiple faces are considered, they are compared by detecting these parts of the faces because these parts are the most useful and important parts of a face. They tend to catch the maximum change among faces, specifically, the change that helps to differentiate one face from the other. This is how Eigen Faces face recognizer works.

Let $X=\{x_1,x_2,\ldots,x_n\}x_i \in R^d$

Here X be a random vector with observations.

1. Calculate the mean μ:

$$\mu=\frac{n}{1}\sum_{i=1}^{n} x_i$$

2. Calculate the covariance matrix S:

$$S=\frac{n}{1}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T$$

3. Compute the eigenvectors v_i and eigenvalues λ_i of S:

$$Sv_i=\lambda_i v_i,\ i=1,2,\ldots,n$$

4. The eigenvectors are arranged by their egeinvalue in descending order:

$$y=W^T(x-\mu)$$

5. Calculate eigenfaces.

Figure 4.1.2: Eigen Faces face recognizer working

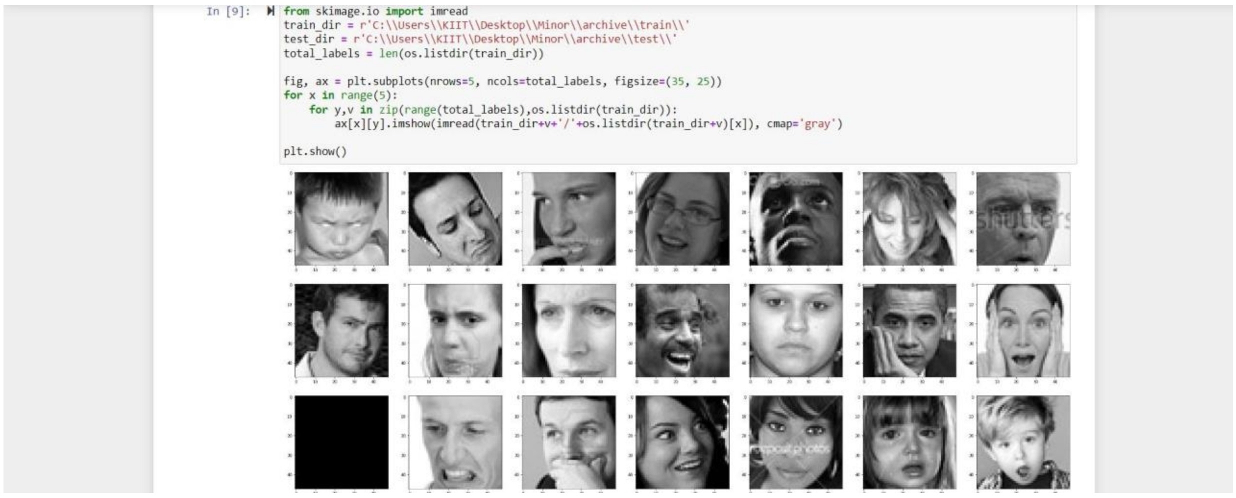
C. Tensor Flow

Using TensorFlow to build face recognition and detection models might require effort, but it is worth it in the end. As mentioned, TensorFlow is the most used Deep Learning framework and it has pre-trained models that easily help with image classification.

The images are classified using CNN. In most cases, to generate a model means the classification of the images only needs to provide a similar image which is the positive image. The image is then trained and retrained through a process known as anchoring or Transfer Learning.

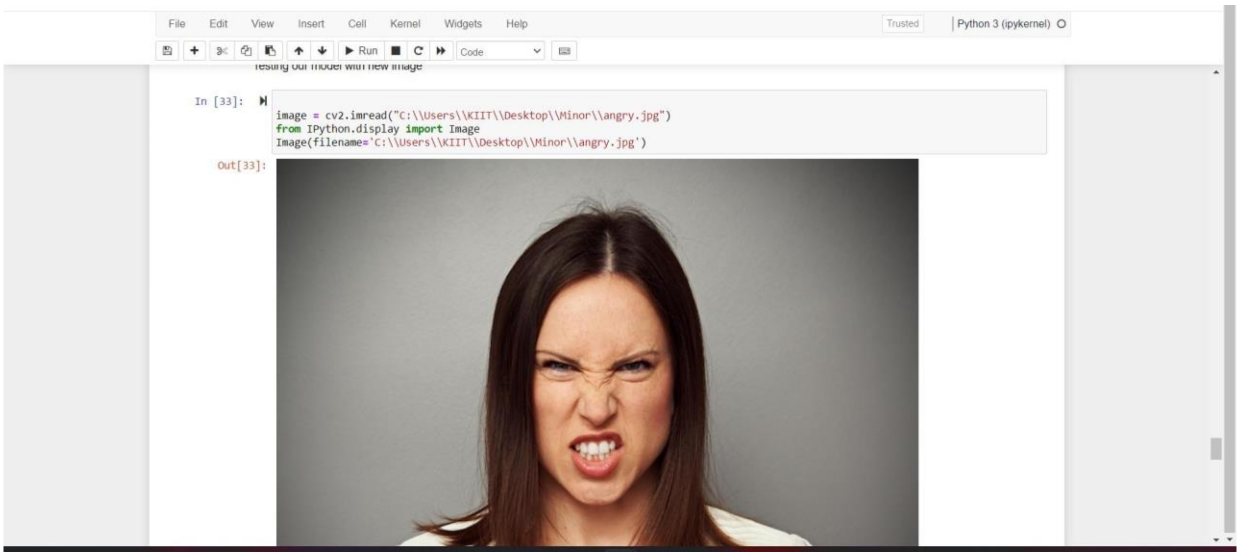
Years back, finding that model for training and retraining was difficult. Now, TensorFlow has simplified the process. Thanks to its huge open-source community, no one has to go through the task of generating a model once another developer from the other end of the world had done it for all to use.

4.2 Testing OR Verification Plan



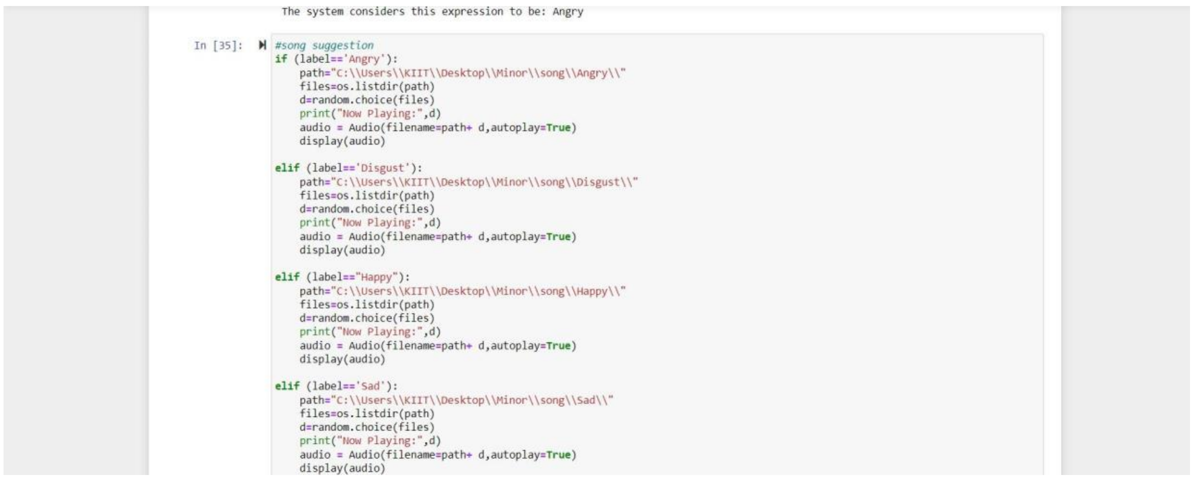
```
In [9]:  from skimage.io import imread
         train_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\train\\'
         test_dir = r'C:\\Users\\KIIT\\Desktop\\Minor\\archive\\test\\'
         total_labels = len(os.listdir(train_dir))

         fig, ax = plt.subplots(nrows=5, ncols=total_labels, figsize=(35, 25))
         for x in range(5):
             for y,v in zip(range(total_labels),os.listdir(train_dir)):
                 ax[x][y].imshow(imread(train_dir+v+'/'+os.listdir(train_dir+v)[x]), cmap='gray')

         plt.show()
```

Figure 4.2.1: Testing imported images



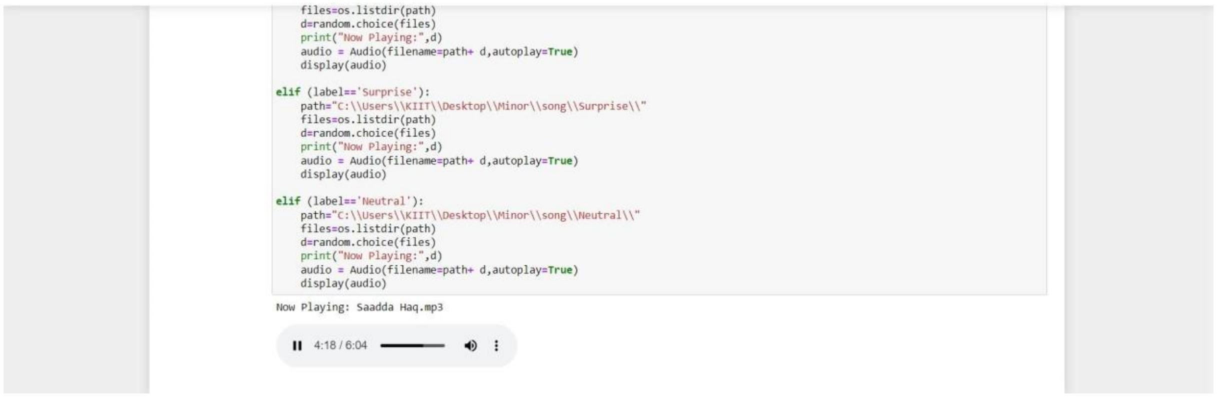
```
In [33]:  image = cv2.imread("C:\\Users\\KIIT\\Desktop\\Minor\\angry.jpg")
          from IPython.display import Image
          Image(filename='C:\\Users\\KIIT\\Desktop\\Minor\\angry.jpg')

Out[33]:
```

Figure 4.2.2:Testing the model with image



```
The system considers this expression to be: Angry

In [35]:  #song suggestion
          if (label=='Angry'):
              path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Angry\\"
              files=os.listdir(path)
              d=random.choice(files)
              print("Now Playing:",d)
              audio = Audio(filename=path+ d,autoplay=True)
              display(audio)

          elif (label=='Disgust'):
              path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Disgust\\"
              files=os.listdir(path)
              d=random.choice(files)
              print("Now Playing:",d)
              audio = Audio(filename=path+ d,autoplay=True)
              display(audio)

          elif (label=="Happy"):
              path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Happy\\"
              files=os.listdir(path)
              d=random.choice(files)
              print("Now Playing:",d)
              audio = Audio(filename=path+ d,autoplay=True)
              display(audio)

          elif (label=='Sad'):
              path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Sad\\"
              files=os.listdir(path)
              d=random.choice(files)
              print("Now Playing:",d)
              audio = Audio(filename=path+ d,autoplay=True)
              display(audio)
```

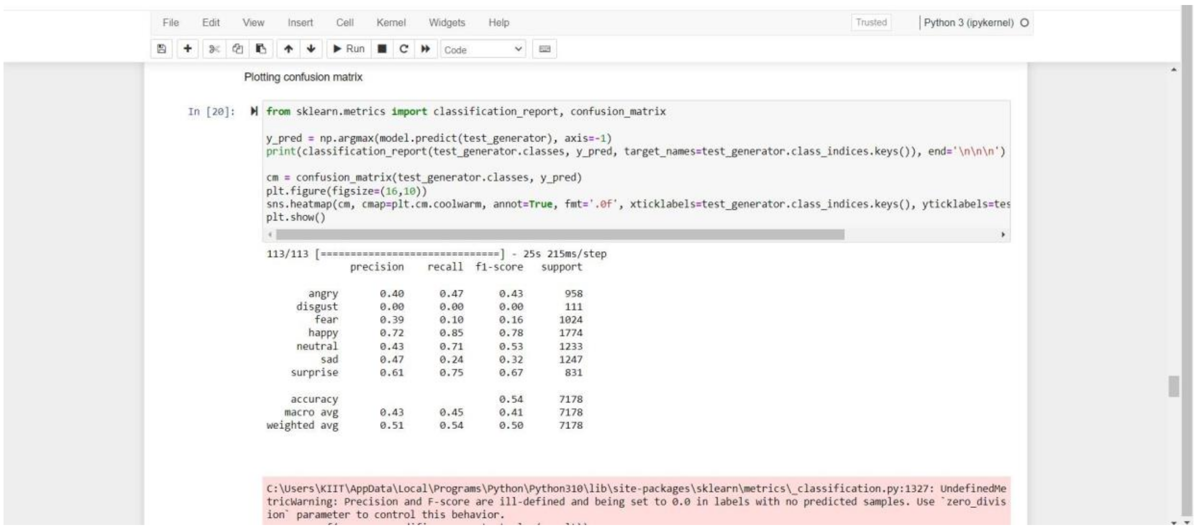
Figure 4.2.3: Conditions for song suggestion(1)



```
        files=os.listdir(path)
        d=random.choice(files)
        print("Now Playing:",d)
        audio = Audio(filename=path+ d,autoplay=True)
        display(audio)

    elif (label=='Surprise'):
        path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Surprise\\"
        files=os.listdir(path)
        d=random.choice(files)
        print("Now Playing:",d)
        audio = Audio(filename=path+ d,autoplay=True)
        display(audio)

    elif (label=='Neutral'):
        path="C:\\Users\\KIIT\\Desktop\\Minor\\song\\Neutral\\"
        files=os.listdir(path)
        d=random.choice(files)
        print("Now Playing:",d)
        audio = Audio(filename=path+ d,autoplay=True)
        display(audio)
```

```
Now Playing: Saadda Haq.mp3
```

Figure 4.2.4: Conditions for song suggestion(2)



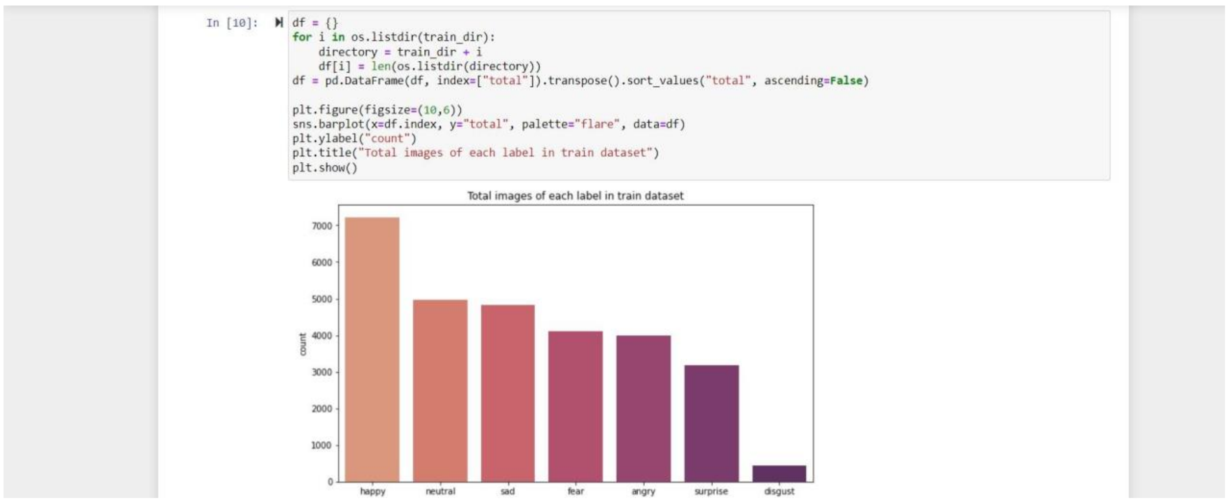
Plotting confusion matrix

```
In [20]:
from sklearn.metrics import classification_report, confusion_matrix

y_pred = np.argmax(model.predict(test_generator), axis=-1)
print(classification_report(test_generator.classes, y_pred, target_names=test_generator.class_indices.keys()), end='\n\n\n')

cm = confusion_matrix(test_generator.classes, y_pred)
plt.figure(figsize=(16,10))
sns.heatmap(cm, cmap=plt.cm.coolwarm, annot=True, fmt='.0f', xticklabels=test_generator.class_indices.keys(), yticklabels=tes
plt.show()
```

```
113/113 [==============================] - 25s 215ms/step
              precision    recall  f1-score   support

       angry       0.40      0.47      0.43       958
     disgust       0.00      0.00      0.00       111
        fear       0.39      0.10      0.16      1024
       happy       0.72      0.85      0.78      1774
     neutral       0.43      0.71      0.53      1233
         sad       0.47      0.24      0.32      1247
    surprise       0.61      0.75      0.67       831

    accuracy                           0.54      7178
   macro avg       0.43      0.45      0.41      7178
weighted avg       0.51      0.54      0.50      7178


C:\Users\KIIT\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1327: UndefinedMe
tricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
```

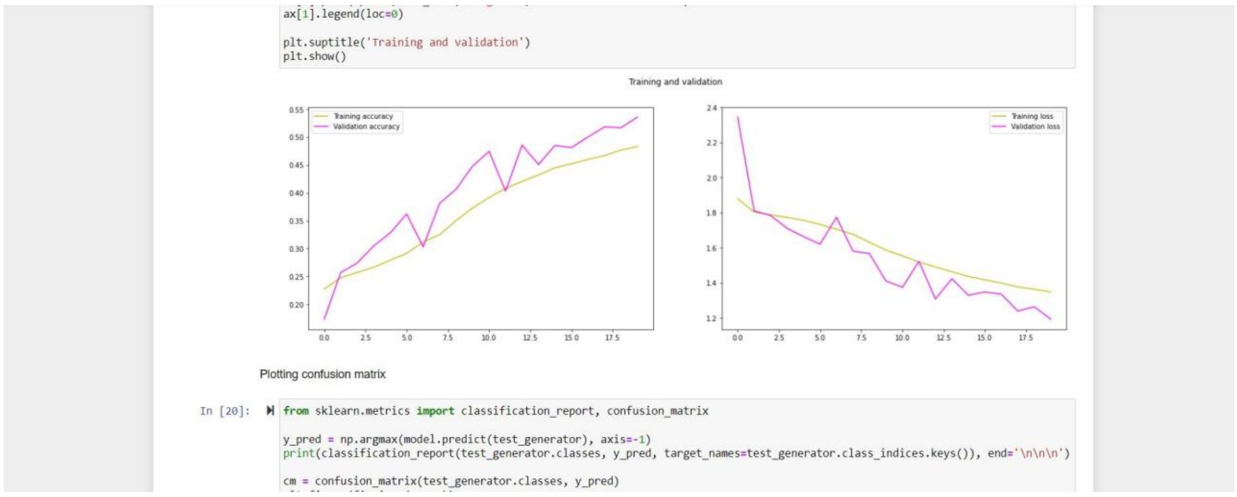
Figure 4.2.5: Output of confusion matrix

4.3 Result Analysis



```
In [10]:
df = {}
for i in os.listdir(train_dir):
    directory = train_dir + i
    df[i] = len(os.listdir(directory))
df = pd.DataFrame(df, index=["total"]).transpose().sort_values("total", ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x=df.index, y="total", palette="flare", data=df)
plt.ylabel("count")
plt.title("Total images of each label in train dataset")
plt.show()
```

Figure 4.3.1: Total number of images of each label in trained dataset



```
ax[1].legend(loc=0)

plt.suptitle('Training and validation')
plt.show()
```

Plotting confusion matrix

```
In [20]:  from sklearn.metrics import classification_report, confusion_matrix

          y_pred = np.argmax(model.predict(test_generator), axis=-1)
          print(classification_report(test_generator.classes, y_pred, target_names=test_generator.class_indices.keys()), end='\n\n\n')

          cm = confusion_matrix(test_generator.classes, y_pred)
```

Figure 4.3.2: Training and validation of dataset

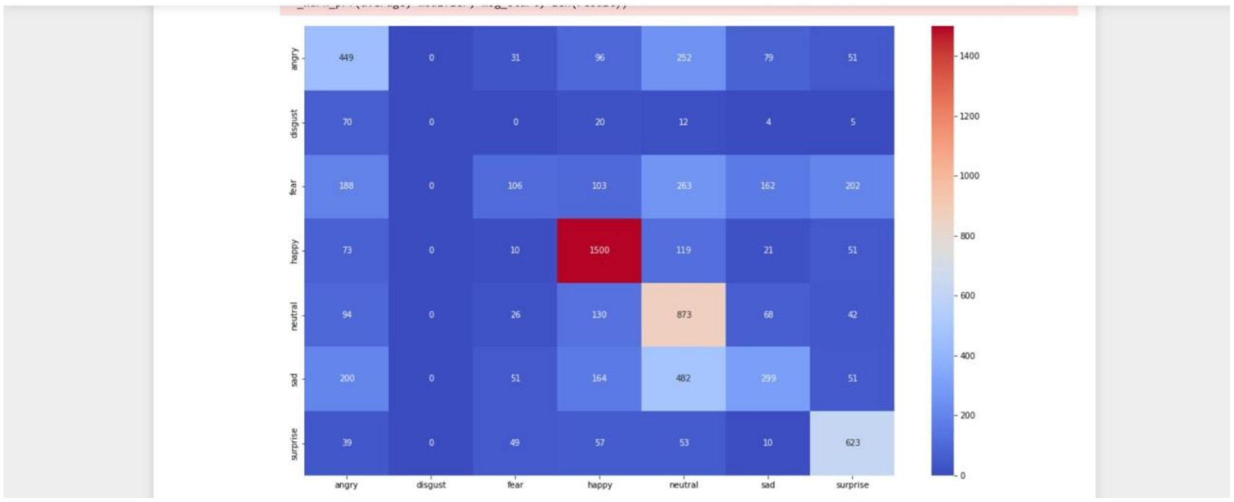
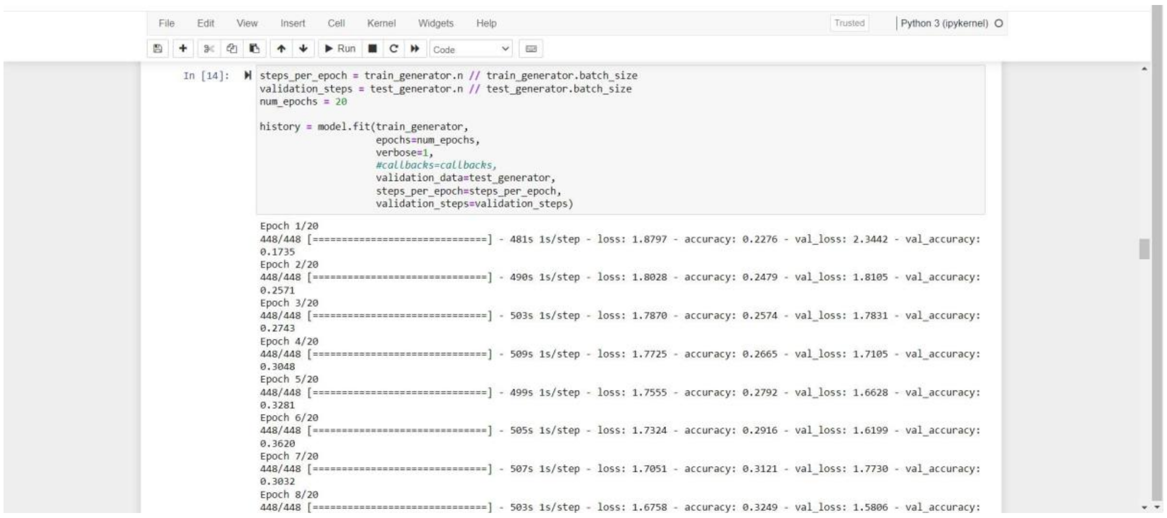


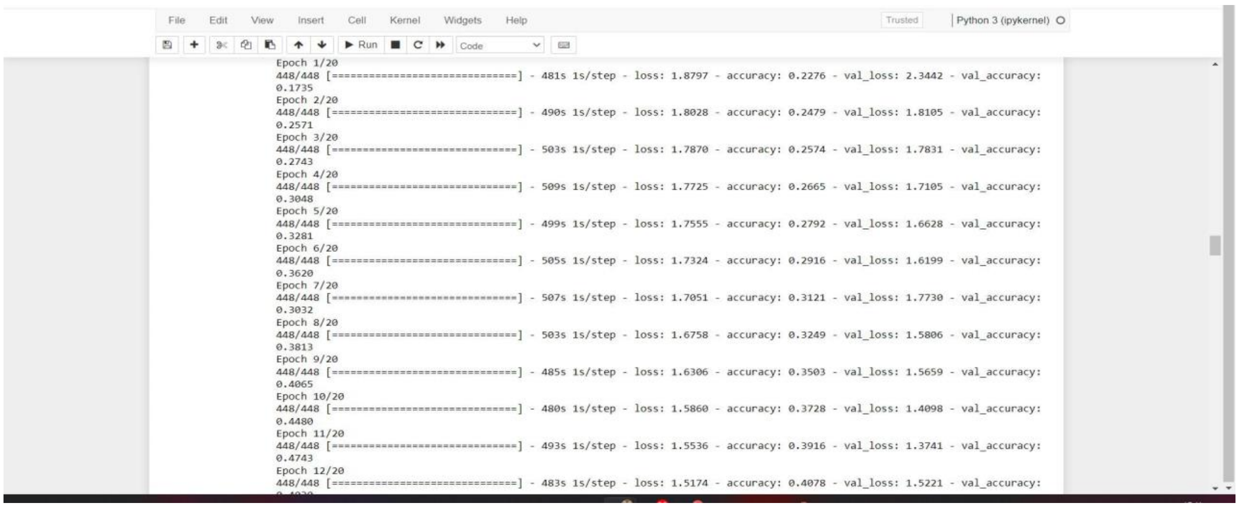
Figure 4.3.3: Confusion matrix of the dataset

4.4 Quality Assurance



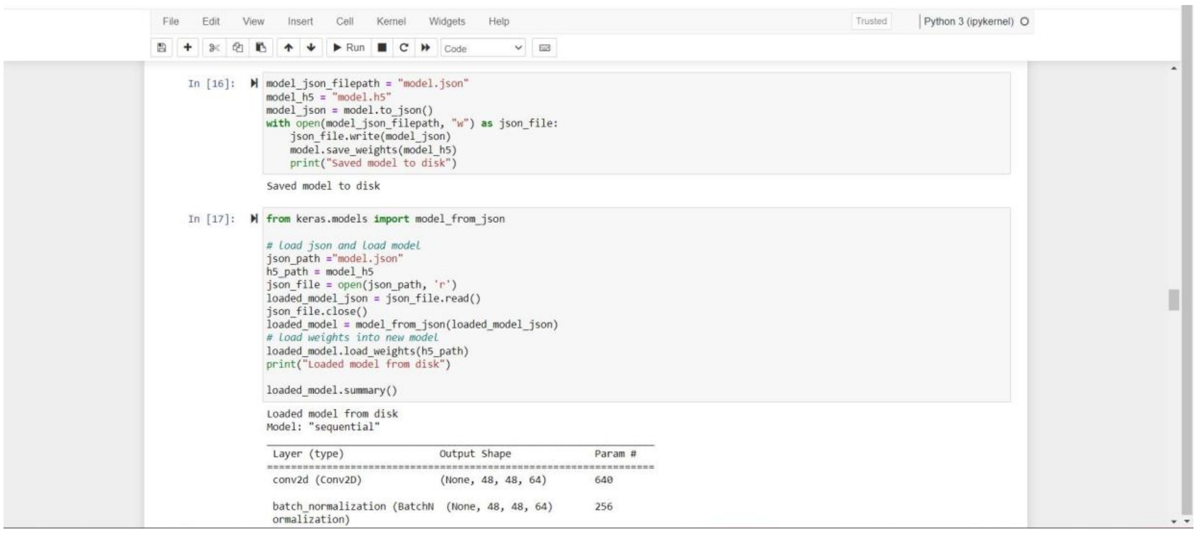
```
In [14]:  steps_per_epoch = train_generator.n // train_generator.batch_size
          validation_steps = test_generator.n // test_generator.batch_size
          num_epochs = 20

          history = model.fit(train_generator,
                              epochs=num_epochs,
                              verbose=1,
                              #callbacks=callbacks,
                              validation_data=test_generator,
                              steps_per_epoch=steps_per_epoch,
                              validation_steps=validation_steps)

          Epoch 1/20
          448/448 [==============================] - 481s 1s/step - loss: 1.8797 - accuracy: 0.2276 - val_loss: 2.3442 - val_accuracy:
          0.1735
          Epoch 2/20
          448/448 [==============================] - 490s 1s/step - loss: 1.8028 - accuracy: 0.2479 - val_loss: 1.8105 - val_accuracy:
          0.2571
          Epoch 3/20
          448/448 [==============================] - 503s 1s/step - loss: 1.7870 - accuracy: 0.2574 - val_loss: 1.7831 - val_accuracy:
          0.2743
          Epoch 4/20
          448/448 [==============================] - 509s 1s/step - loss: 1.7725 - accuracy: 0.2665 - val_loss: 1.7105 - val_accuracy:
          0.3048
          Epoch 5/20
          448/448 [==============================] - 499s 1s/step - loss: 1.7555 - accuracy: 0.2792 - val_loss: 1.6628 - val_accuracy:
          0.3281
          Epoch 6/20
          448/448 [==============================] - 505s 1s/step - loss: 1.7324 - accuracy: 0.2916 - val_loss: 1.6199 - val_accuracy:
          0.3620
          Epoch 7/20
          448/448 [==============================] - 507s 1s/step - loss: 1.7051 - accuracy: 0.3121 - val_loss: 1.7730 - val_accuracy:
          0.3032
          Epoch 8/20
          448/448 [==============================] - 503s 1s/step - loss: 1.6758 - accuracy: 0.3249 - val_loss: 1.5806 - val_accuracy:
```

Figure 4.4.1: Output of Quality Assurance(1)



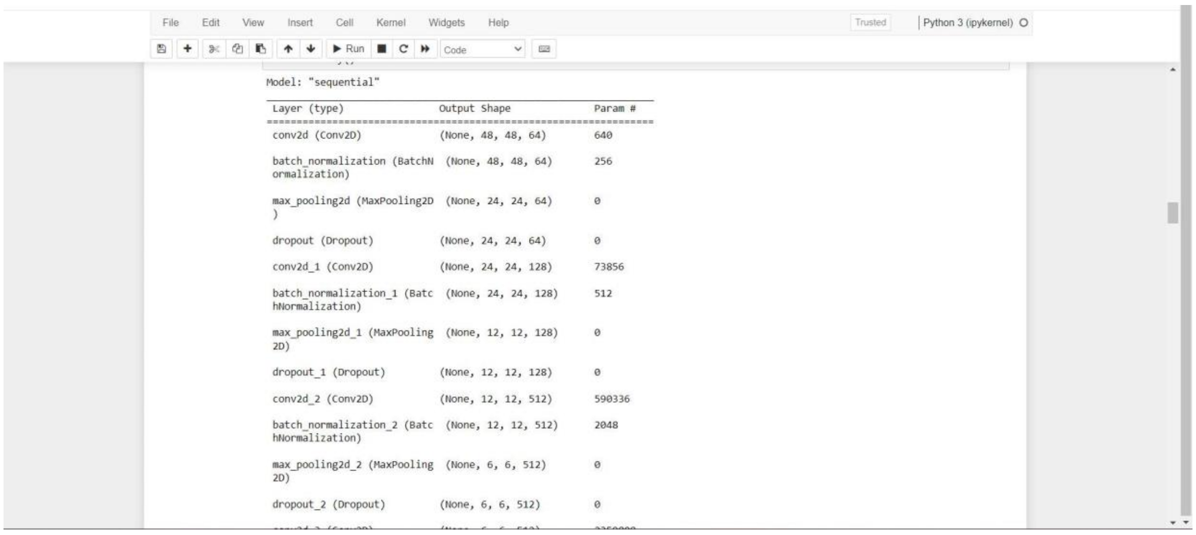
```
Epoch 1/20
448/448 [==============================] - 481s 1s/step - loss: 1.8797 - accuracy: 0.2276 - val_loss: 2.3442 - val_accuracy:
0.1735
Epoch 2/20
448/448 [==============================] - 490s 1s/step - loss: 1.8028 - accuracy: 0.2479 - val_loss: 1.8105 - val_accuracy:
0.2571
Epoch 3/20
448/448 [==============================] - 503s 1s/step - loss: 1.7870 - accuracy: 0.2574 - val_loss: 1.7831 - val_accuracy:
0.2743
Epoch 4/20
448/448 [==============================] - 509s 1s/step - loss: 1.7725 - accuracy: 0.2665 - val_loss: 1.7105 - val_accuracy:
0.3048
Epoch 5/20
448/448 [==============================] - 499s 1s/step - loss: 1.7555 - accuracy: 0.2792 - val_loss: 1.6628 - val_accuracy:
0.3281
Epoch 6/20
448/448 [==============================] - 505s 1s/step - loss: 1.7324 - accuracy: 0.2916 - val_loss: 1.6199 - val_accuracy:
0.3620
Epoch 7/20
448/448 [==============================] - 507s 1s/step - loss: 1.7051 - accuracy: 0.3121 - val_loss: 1.7730 - val_accuracy:
0.3032
Epoch 8/20
448/448 [==============================] - 503s 1s/step - loss: 1.6758 - accuracy: 0.3249 - val_loss: 1.5806 - val_accuracy:
0.3813
Epoch 9/20
448/448 [==============================] - 485s 1s/step - loss: 1.6306 - accuracy: 0.3503 - val_loss: 1.5659 - val_accuracy:
0.4065
Epoch 10/20
448/448 [==============================] - 480s 1s/step - loss: 1.5860 - accuracy: 0.3728 - val_loss: 1.4098 - val_accuracy:
0.4480
Epoch 11/20
448/448 [==============================] - 493s 1s/step - loss: 1.5536 - accuracy: 0.3916 - val_loss: 1.3741 - val_accuracy:
0.4743
Epoch 12/20
448/448 [==============================] - 483s 1s/step - loss: 1.5174 - accuracy: 0.4078 - val_loss: 1.5221 - val_accuracy:
```

Figure 4.4.2: Output of Quality Assurance(2)



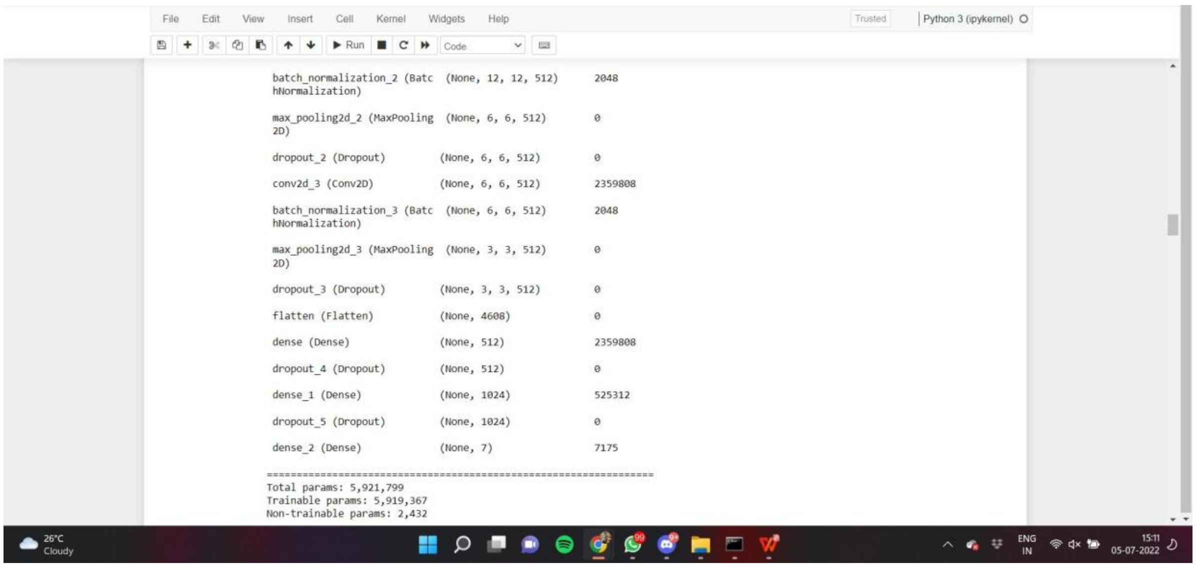
```
In [16]:  model_json_filepath = "model.json"
          model_h5 = "model.h5"
          model_json = model.to_json()
          with open(model_json_filepath, "w") as json_file:
              json_file.write(model_json)
              model.save_weights(model_h5)
              print("Saved model to disk")

          Saved model to disk

In [17]:  from keras.models import model_from_json

          # load json and load model
          json_path ="model.json"
          h5_path = model_h5
          json_file = open(json_path, 'r')
          loaded_model_json = json_file.read()
          json_file.close()
          loaded_model = model_from_json(loaded_model_json)
          # load weights into new model
          loaded_model.load_weights(h5_path)
          print("Loaded model from disk")

          loaded_model.summary()

          Loaded model from disk
          Model: "sequential"
          _________________________________________________________________
           Layer (type)                Output Shape              Param #
          =================================================================
           conv2d (Conv2D)             (None, 48, 48, 64)        640

           batch_normalization (BatchN  (None, 48, 48, 64)       256
           ormalization)
```

Figure 4.4.3: Loading json and loading model



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        640

 batch_normalization (BatchN  (None, 48, 48, 64)       256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 24, 24, 64)       0
 )

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 128)       73856

 batch_normalization_1 (Batc  (None, 24, 24, 128)      512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 12, 12, 128)       0

 conv2d_2 (Conv2D)           (None, 12, 12, 512)       590336

 batch_normalization_2 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 6, 6, 512)         0
```

Figure 4.4.4: Output of the loaded model(1)



```
 batch_normalization_2 (Batc  (None, 12, 12, 512)      2048
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 6, 6, 512)         0

 conv2d_3 (Conv2D)           (None, 6, 6, 512)         2359808

 batch_normalization_3 (Batc  (None, 6, 6, 512)        2048
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 512)        0
 2D)

 dropout_3 (Dropout)         (None, 3, 3, 512)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 512)               2359808

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 1024)              525312

 dropout_5 (Dropout)         (None, 1024)              0

 dense_2 (Dense)             (None, 7)                 7175

=================================================================
Total params: 5,921,799
Trainable params: 5,919,367
Non-trainable params: 2,432
```

Figure 4.4.5: Output of the loaded model(2)

Chapter 5

Standards Adopted

5.1 Design Standards

Below are the processes that we follow in our program designing:

Data Visualization

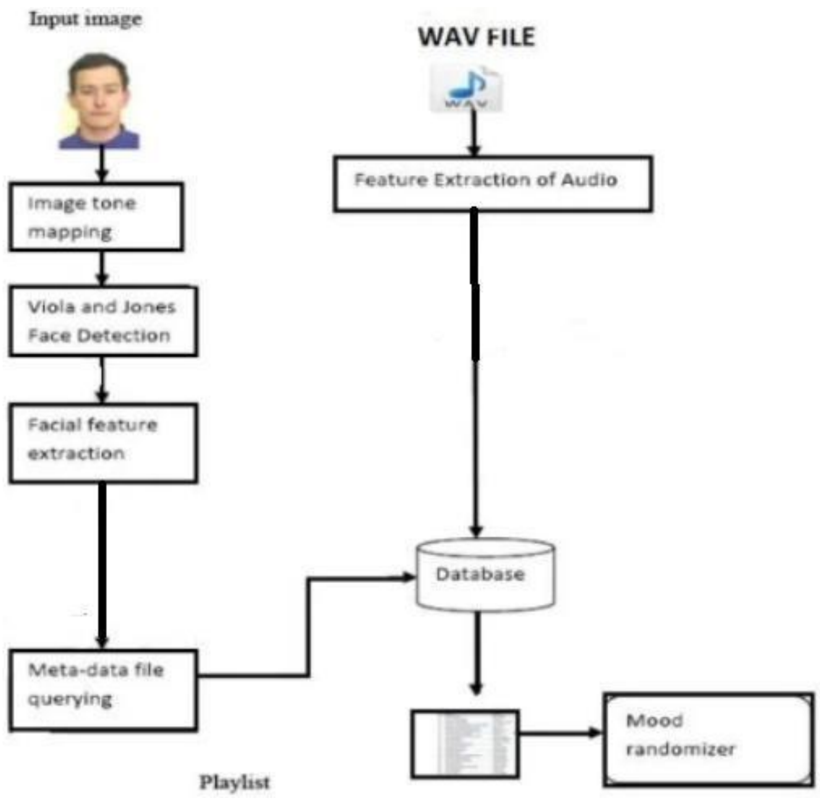


Figure 5.1.1: Data Visualization

5.2 Coding Standards

Coding standards are collections of coding rules, guidelines, and best practices. Few of the coding standards are:

1. Writing as few lines as possible.
2. Using appropriate naming conventions.
3. Leaving comments and prioritizing documentation.
4. Segmenting blocks of code in the same section into paragraphs.
5. Use indentation to mark the beginning and end of control structures.Clearly specify the code between them.
6. Don't use lengthy functions. Ideally, a single function should carry out a single task.
7. Using the DRY principle.Automating repetitive tasks when necessary.

8. Avoiding Deep Nesting.Too many nesting levels make code harder to read and follow.

9. Avoiding long lines of code.It is easier for humans to read blocks of lines horizontally short and vertically long.

10. Turning daily backups into an instinct.Multiple events can trigger data loss-system crash,dead battery,software glitch,hardware damage,etc.To prevent this,save code daily and after every modification,no matter how minuscule it may be.Back up the workflow on TFS,SVN or any other version control mechanism.

11. Trying to formalize Exception Handling

'Exception' refers to problems, issues, or uncommon events that occur when code is run and disrupts the normal flow of execution. This either pauses or terminates program execution, which is a scenario that must be avoided.

5.3 Testing Standards

1. ISO/IEC/IEEE 29119-1

This software testing standard focuses on definitions and concepts of all other standards in the 29119 series of quality standards. It helps user understand the vocabulary on which the other standards in the series are built as well as provides relevant examples to show the way in which each concept works in practice.

A complete knowledge bank, the 29119-1 can also be considered to be the foundation of IEEE software testing standards. Some of the topics that are included in this series are introduction to software testing, testing processes in SDLC, risk-based testing, common test practices, defect management, etc.

2. ISO/IEC/IEEE 29119-2

This standard has been designed with an aim to develop a generic process model that can be used for conducting testing in any SDLC.

As per this standard, the testing process will work on a three-layer process that would include organizational test specifications such as organizational test policy and test strategy, test management and dynamic testing.

Laying a special emphasis on alleviation of risks, this standard allows the process of testing to focus on product's key features and attributes under test.

3. ISO/IEC/IEEE 29119-3

With a special focus on documentation, this standard provides standardized templates that are designed in a way that covers the entire SDLC. However, these templates can also be customized as per each organization's unique requirements.

Designed in alignment with the process defined in the ISO/IEC/IEEE 29119-2, this software testing standard is likely to surpass the IEEE 829 standard.

4. ISO/IEC/IEEE 29119-4

With a focus on software test design techniques developed for organizations and SDLC models, the techniques present in this standard can be an effective way to develop test cases that can be used to present evidence that each requirement mentioned under test have been successfully met or there are certain defects that need to be rectified.

Designed on the basis of BS-7925-2 component testing standard, this standard covers a number of dynamic software testing techniques such as equivalence partitioning, classification tree method and boundary value analysis.

5. ISO/IEC/IEEE 29119-5

This standard supports those techniques and approaches that support keyword driven testing, a technique that involves describing test cases on the basis of predefined set of keywords.

These keywords are based on a set of actions that are required to be performed using specific steps in a test case. Writing keywords in a natural language helps one gain an easy understanding of these test cases.

Implementing these standards may enable companies to deliver better products in the market, a lot of these are encouraged to follow authentic and reliable software techniques and approaches as a whole.

`

Chapter 6

Conclusion & Future Scope

6.1 Conclusion

It was found that the accuracy was more than 75% for most of the cases,and is able to detect seven moods accurately: anger, disgust, happy, sad, surprise and neutral as seen in the confusion matrix.A simple system is proposed here for the music recommendation using face emotion recognition. It suggests music by extracting different facial emotions of a person: anger, disgust, happy, sad, surprise and neutral. Even though human emotions are complex and subtle, it is possible for a machine learning model to be trained to accurately detect a set of emotions which can be differentiated from each other with certain facial expressions. The expression on a person's face can be used to detect their mood, and once a certain mood has been detected, music suitable for the person's detected mood can be suggested.

6.2 Future Scope

For accurate detection of fear and disgust moods, additional parameters such as heart rate or body temperature must also be considered rather than solely depending on facial expressions. In addition to that, finding suitable music to be played on detection anger or disgust mood is also a challenge. As a result, it can be considered as a future scope for our project. Our trained model is an overfit model, which can sometimes lead to fluctuations in accurate detection. For example, the "disgust" mood is mostly classified as an "angry" mood since the facial features (eyebrows, cheeks) are similar for both.

References

1) A. Abdul, J. Chen, H.-Y. Liao, and S.-H. Chang, “An Emotion-Aware Personalized Music Recommendation System Using a Convolutional Neural Networks Approach,” Applied Sciences.

2) D Priya, Face Detection, Recognition and Emotion Detection in 8 lines of code!, towards data science, April 3 Accessed on: July 12, 2020 [Online], Available at: https://towardsdatascience.com/facedetection-recognition-and-emotion-detection-in-8-lines-of-codeb2ce32d4d5de

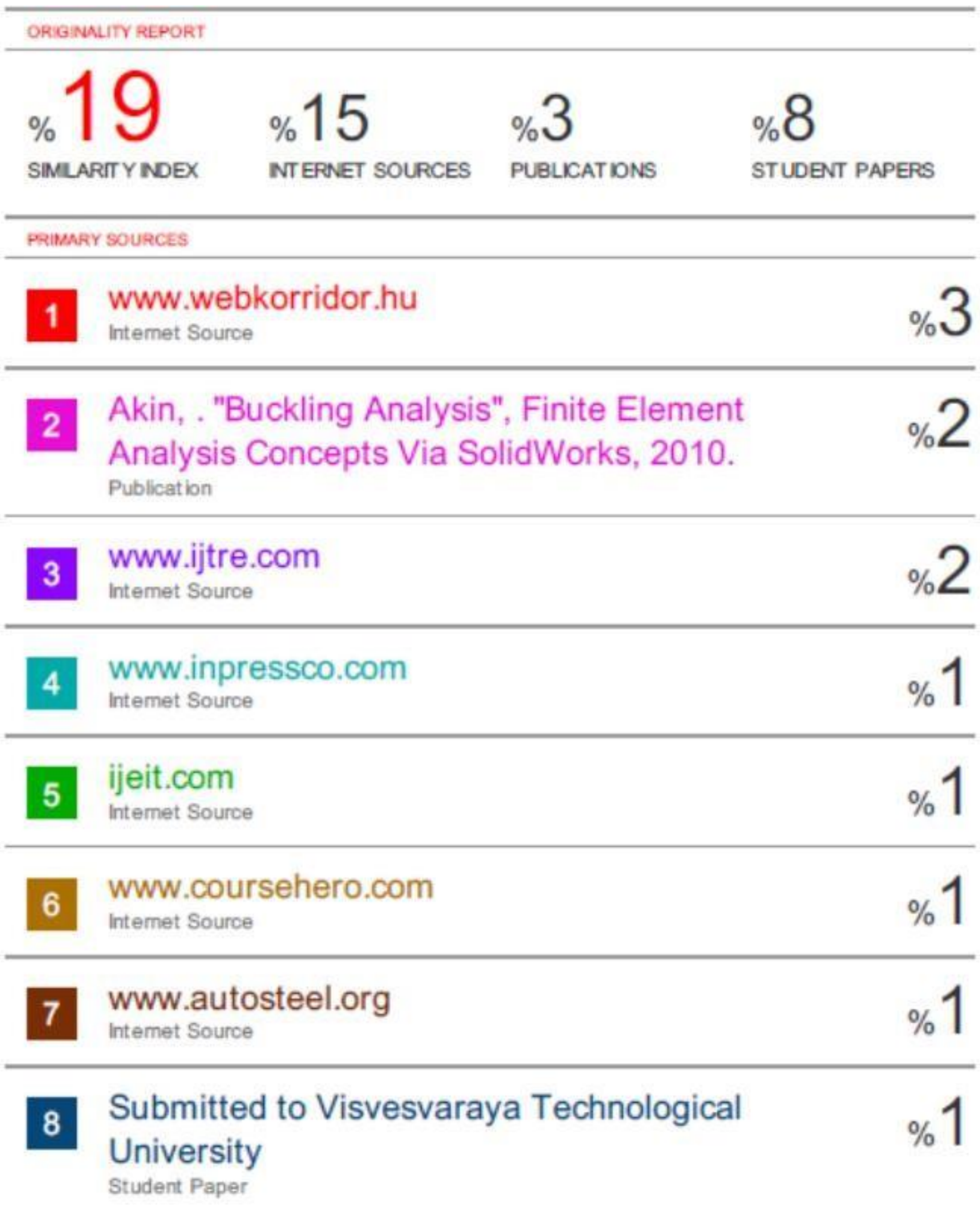
3) Music Recommendation System: “Sound Tree”, Dcengo Unchained: Sıla KAYA, BSc.; Duygu KABAKCI, BSc.; Işınsu KATIRCIOĞLU, BSc. and Koray KOCAKAYA BSc. Assistant : Dilek Önal Supervisors: Prof. Dr. İsmail Hakkı Toroslu, Prof. Dr. Veysi İşler Sponsor Company: ARGEDOR

4) Tim Spittle, lucyd, Git Hub, , April 16, 2020. Accessed on: [Online], Available at: https://github.com/timspit/lucyd

5) Puri, Raghav & Gupta, Archit & Sikri, Manas & Tiwari, Mohit & Pathak, Nitish & Goel, Shivendra. (2020). Emotion Detection using Image Processing in Python. [11] P

TURNITIN PLAGIARISM REPORT

(The plagiarism is below 25%)



ORIGINALITY REPORT

%19	%15	%3	%8
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.webkorridor.hu Internet Source	%3
2	Akin, . "Buckling Analysis", Finite Element Analysis Concepts Via SolidWorks, 2010. Publication	%2
3	www.ijtre.com Internet Source	%2
4	www.inpressco.com Internet Source	%1
5	ijeit.com Internet Source	%1
6	www.coursehero.com Internet Source	%1
7	www.autosteel.org Internet Source	%1
8	Submitted to Visvesvaraya Technological University Student Paper	%1