1. Introduction

Music is universal and easily accessible in our everyday lives, thanks to the tremendous growth of digital music technologies. People listen to music on various sources to improve their mood and enliven the environment. However, due to the enormous number of songs accessible, people are frequently perplexed when selecting a track.

People enjoy listening to different genres of music according to their mood. So, Naive recommendations do not work as intended. This necessitates the development of a context-sensitive music recommendation system. Context-aware recommendation systems also need accumulation of additional data, which can be difficult to come by but has shown to be worthwhile in some cases.

Context-aware music recommendation systems build on the principle that various personality traits connect with distinct item attributes (for example, acoustic qualities or musical genres) and that users in different emotional states and moods prefer various sorts of things.

Emotion-aware Recommendation systems have yielded encouraging results, demonstrating that using emotional states and reactions, recommendation algorithms may improve prediction accuracy and refine personalisation.

The emotions of users may be identified in a variety of ways. However, they all fall into one of two types. Face expressions, keystrokes, and mouse-click patterns are all examples of implicit ways of detecting emotion. Explicit methods, on the other hand, take direct input from the user. For the identification of emotion, we choose to utilise an approach based on facial expressions.

In the next sections, several Facial-Emotion Identification techniques, machine-learning models, and recommendation algorithms will be reviewed and contrasted. We further demonstrate that the emotion-identification method used has no effect on the recommendation algorithms.

1.1. Facial Emotion Identification

The task of recognising emotion from a user's face data is known as facial emotion identification. A Neural Network is typically used to accomplish the task. A Neural Network is a network of Interconnected perceptrons.

Working of an Artificial Neural Network:

A single neuron can either fire or not, so its activity level can be represented as a one or a zero. One neuron receives its information from a bunch of other neurons, but because the strength of these connections varies, each one might be assigned a different weight. Some connections are excitatory, so they have positive weights, while others are inhibitory, so they have negative weights.

The approach to find out whether a certain neuron fires is to multiply the activity of each input neuron by its weight, then add them all together. The neuron fires if their sum is larger than a value called the bias; if it is less than that, the neuron does not fire.

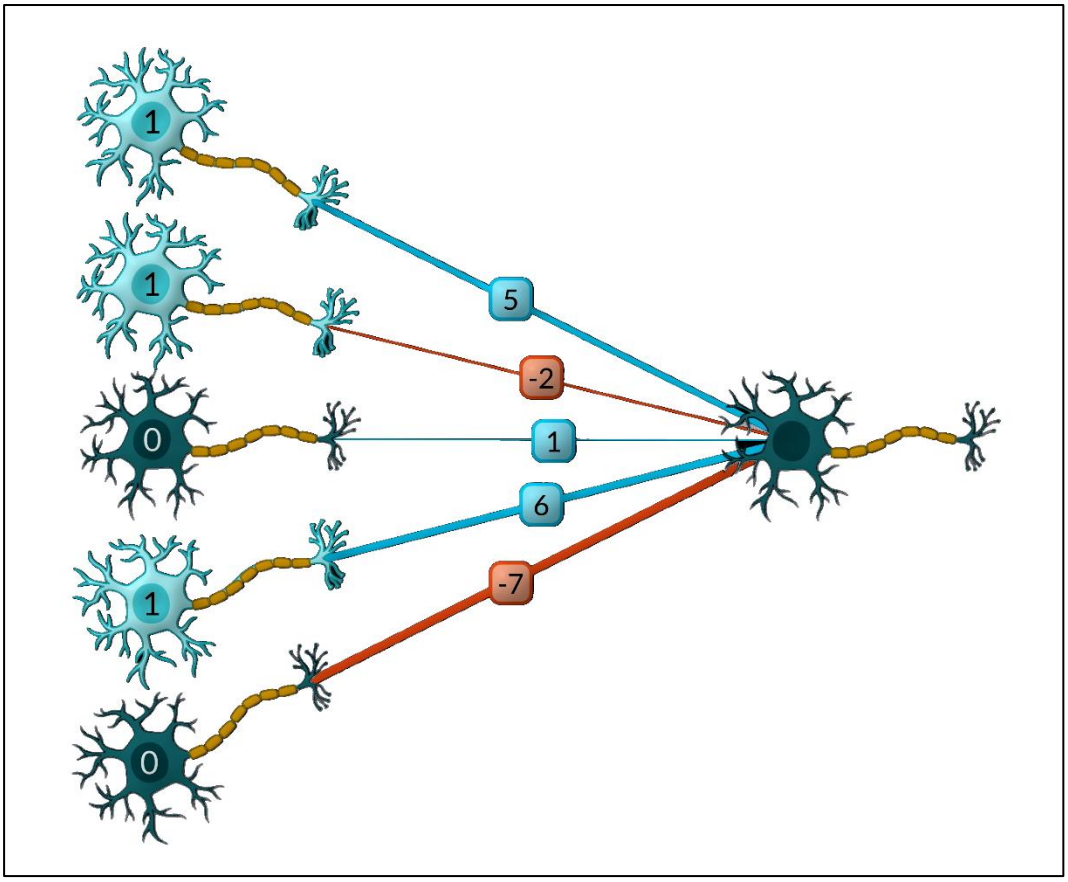

Figure 1.1.1: Basic model of how Neurons in Artificial Neural Networks work.

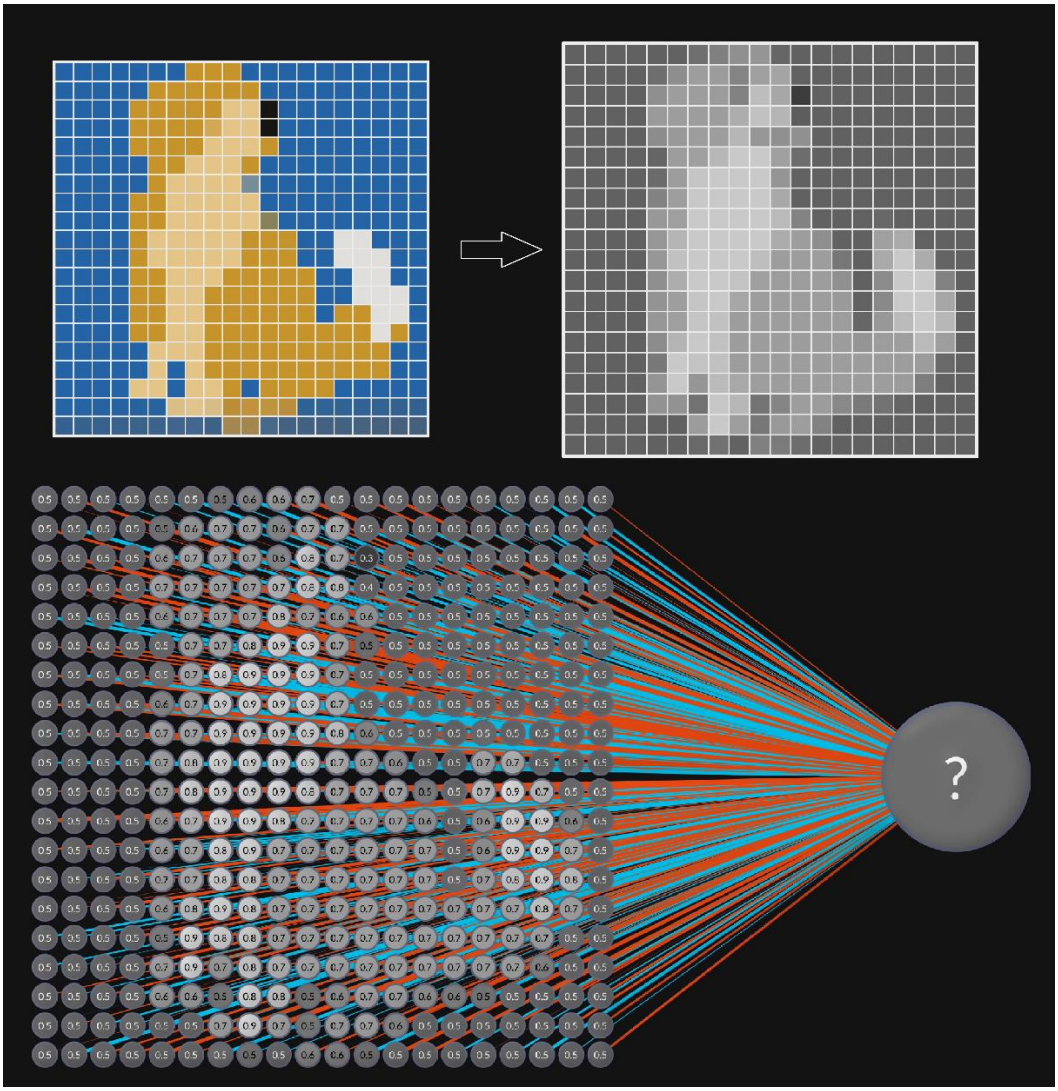

Figure 1.1.2: Processing of a digital image by a Neural network.

The Neural Network accepts an M x N matrix as input, which represents a picture with M x N pixels. Each pixel functions like a neuron, with the brightness of the pixel determining its activation. In most cases, the activation is normalised to a number between zero and one. Each of these neurons is linked to the next collection of neurons via their own adjustable weights, which is referred to as a layer.

Depending on the bias and the summation of the product of weights and activation, each of the neurons in the following layer will either fire or not. At the end of the process, the output layer will have a number of neurons, generally only one of which will fire.

The outcome of the Neural Network is represented by the output layer. Depending on what the Neural Network interprets the image as, just one neuron in the output layer will fire. The training algorithm will alter the weights associated with each neuron whenever an output neuron fires when it shouldn't. Similarly, when an output neuron does not fire when it should, the weights are modified.

The weights associated with the neurons in the hidden layer are continuously adjusted by the algorithm until the neural network properly recognises all of the training pictures. The process of adjusting the weights is called Back Propagation.

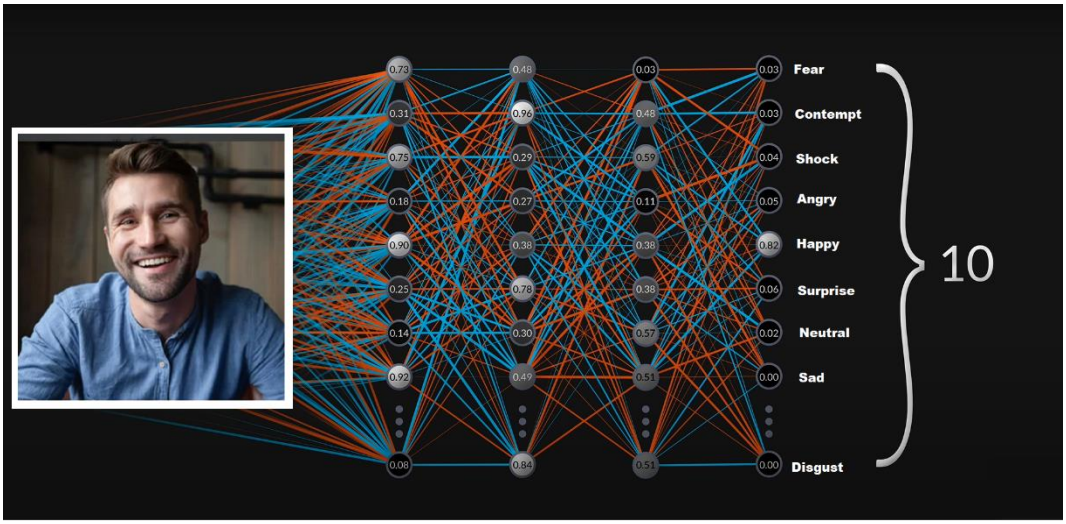

Figure 1.1.3: Emotion Identification by a Neural Network.

For more precise training, the Neural Network becomes denser. Processing a single image takes roughly 700,000,000 mathematical operations because of all the large Matrix Multiplications. This creates a huge need for faster processing equipment, particularly GPUs (graphics processing units) that are designed for rapid, simultaneous computations.

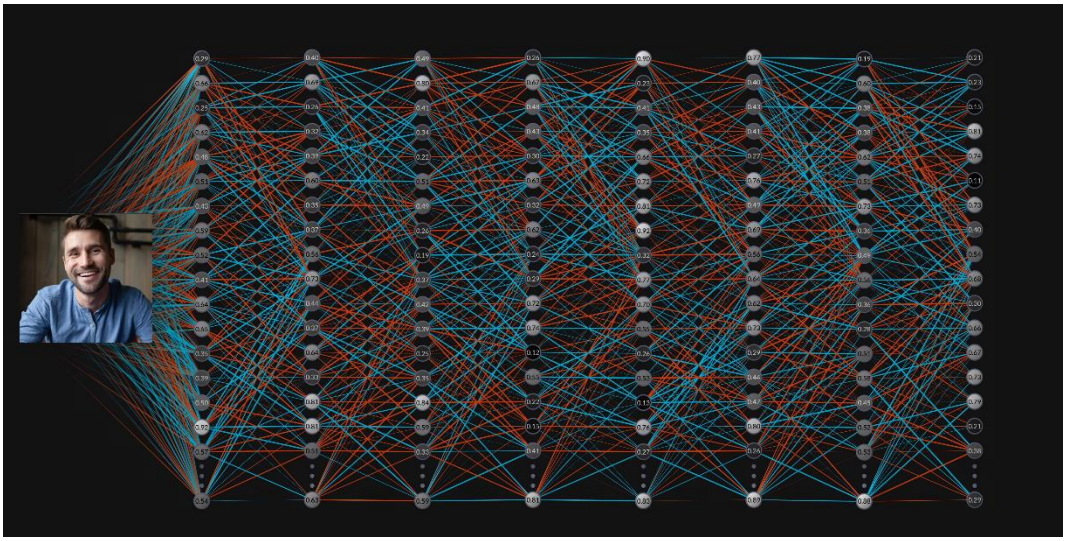

Figure 1.1.4: A Neural Network involving more hidden layers.

The contrasts, benefits, and limitations of the top three Neural Networks, **Convolutional Neural Networks**, **Deep Neural Networks**, and **Multi-task Cascaded Convolutional Neural Networks** are the topic of discussion of our project.

1.2. Music Recommendation

The ordinary individual makes a range of purposeful media consumption selections on any given day. As we manoeuvre these omnipresent options in our increasingly diversified environment, recommendation system algorithms that have become pervasive in our lives point us in the right direction.

The fundamental goal of recommendation algorithms is to assess user data so that personalised recommendations may be made.

1.2.1. Collaborative Filtering

To provide suggestions, collaborative filtering looks for similarities between users and products at the same time. Collaborative filtering is often used with very big data sets. The collaborative filtering strategy has the benefit of not relying on machine comprehensible material, thus it can properly recommend complicated items without requiring a "knowledge" of the item itself.

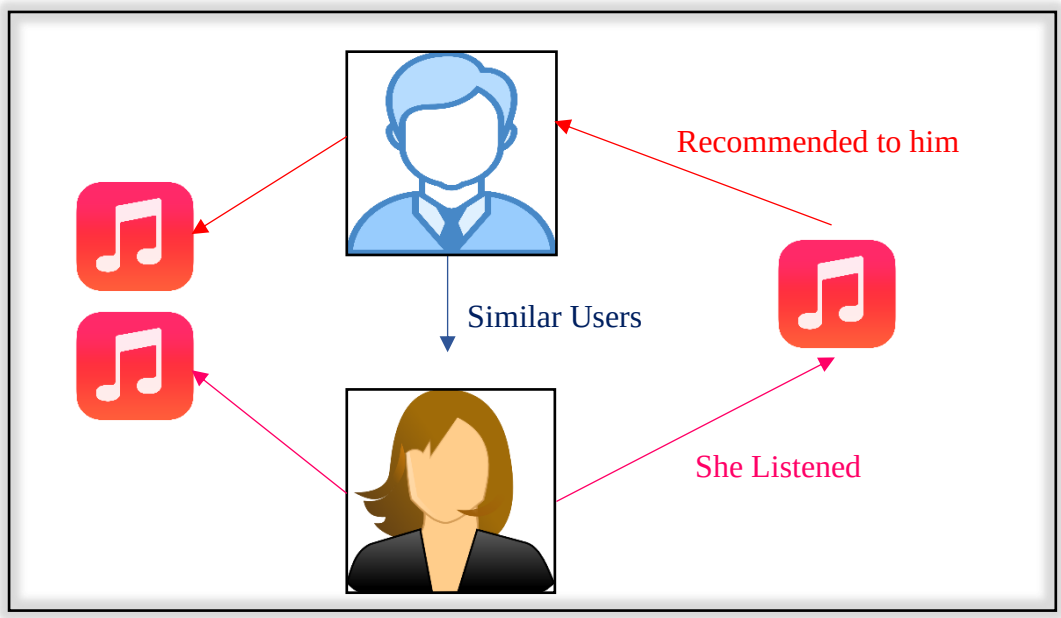

Figure 1.2.1: Collaborative Filtering

1.2.2. Content-based Filtering

Another prominent way for constructing recommender systems is content-based filtering. Based on the user's prior activities or explicit input, content-based filtering uses item attributes to recommend additional items that are similar to what they like.

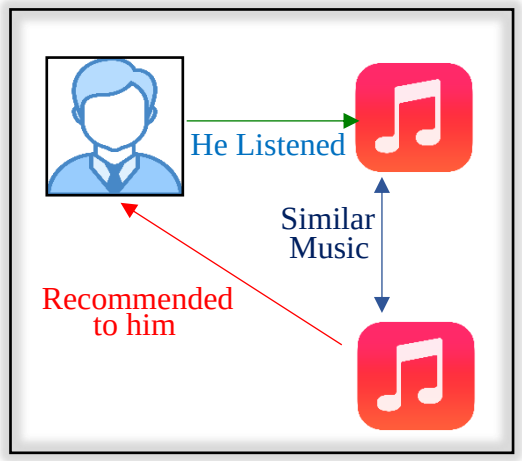

Figure 1.2.2: Content-based Filtering

2. Overview of Proposed System

The Proposed system consists of two phases:

2.1. Emotion Identification

2.2. Music Recommendation

2.1. Emotion Identification

Neural Networks are typically used for face classification and emotion identification. After extensive research and experimentation on different neural network models such as the CNN model, MTCNN model and DNN model, the following observations were made:

Parameter	CNN	Deep Face (DNN)	FER (MTCNN)
Accuracy	High	Low	High
Train Time	Low	High	High
Validation Time	Low	High	High
Advantages	Speed	Light Weight	Self-alignment of Face
Disadvantages	Large Training Data	Low Accuracy	High Train time
Common	Unable to detect rare facial expressions like Disgust		

Table 2.1.1: Compare & Contrast of Different Emotion Detection Models.

The MTCNN model has the highest accuracy compared to other models. Although the train time of MTCNN is high, it can be reduced using GPUs. Hence, we propose the MTCNN model for emotion identification.

In the proposed system, the MTCNN model takes the user's face as input and outputs the dominant emotion detected.

2.2. Music Recommendation

Content-based filtering is used to recommend music. Since there are no publicly available datasets on users' emotions while listening to songs, collaborative filtering cannot be used. Hence content-based filtering approach is proposed for music recommendation.

The proposed approach involves identifying the emotion of each song from the MuSe dataset. Each track in the dataset has valence, arousal, and dominance values, which represent the overall emotion of the song. K-means clustering is done after determining the initial centroids based on the VAD range of the dataset.

After clustering, we get 7 clusters, each representing one of the following seven emotions: happy, sad, angry, neutral, surprise, disgust, and fear. All the tracks with the same emotion get grouped into the corresponding cluster. Based on this, the songs in the dataset get annotated with their corresponding emotion.

Based on the emotion identified from the previous phase and using the tagged music dataset, top k tracks with the same emotion get recommended to the user. These get displayed as Spotify widgets, which can be played by clicking the widget.

3. System Design

The System consists of the following modules

1. Emotion Identification module
2. Music Tagging and Recommendation module
3. User Interface module

3.1. Emotion Identification Module

In this phase, we detect and identify the user's emotion. A Multi-task Cascaded Convolutional Network (MTCNN) model, trained on the FER Dataset, is used to identify the emotion. MTCNN has an added advantage of self-alignment of face compared to other models.

The MTCNN model takes the face as input and outputs the dominant emotion identified. The neural network is composed of numerous layers of weighted nodes. Each layer processes the input and feeds it on to the next layer. The node weights are adjusted during propagation. The node with the largest weight in the output layer correlates to the dominant emotion.

The model performs on par with the current state-of-the-art systems and has reported accuracy of 92.19%.

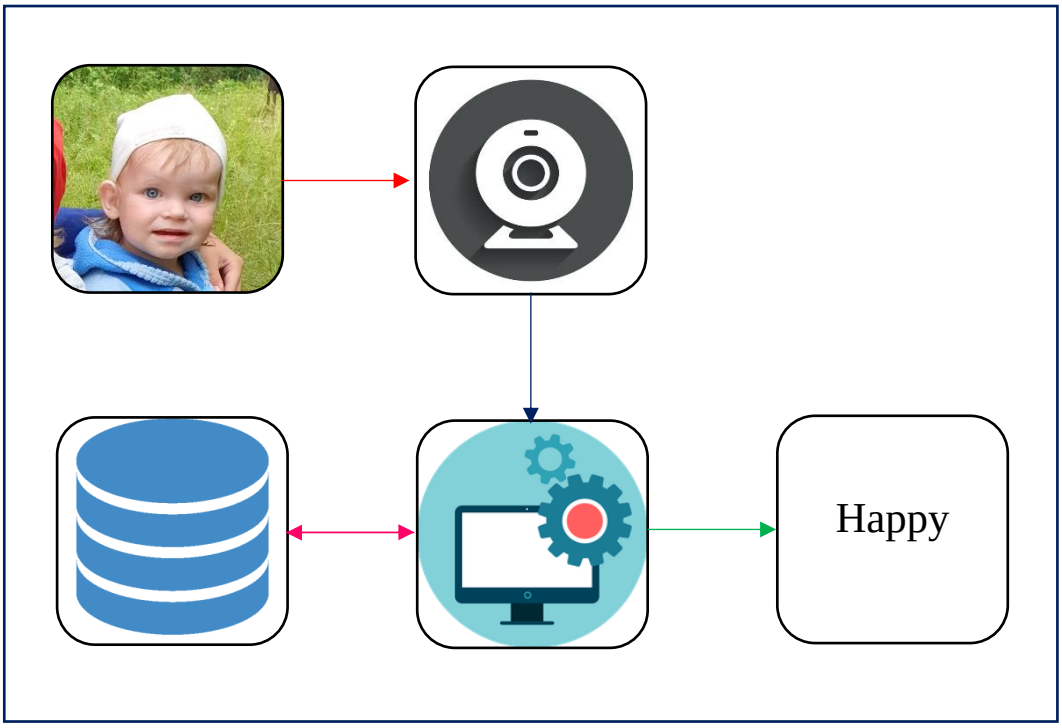

Figure 3.1.1: Emotion Identification Module.

3.2. Music Tagging and Recommendation Module

The next phase involves tagging music with its corresponding emotion. The MuSe dataset consists of over 90,000 tracks along with their valence, arousal, and dominance values. Valence represents the pleasantness dimension, Arousal represents the intensity dimension, and dominance represents the control dimension. However, the dataset doesn't consist of the emotion of the track. Hence each track needs to be tagged with its corresponding emotion.

Tagging involves clustering based on VAD values. VAD values are floating-point coordinates in 3-dimensional space, which represent the overall emotion of the song. VAD values are relative and can vary based on the range of the dataset. Based on the range of the MuSe dataset, VAD values for the seven emotions are determined.

K-Means clustering is used to group songs having same emotion. Initial centroids of the 7 clusters are obtained based on the VAD range. K-Means Clustering is then performed with the initial centroids. 7 Clusters are obtained at the end, each identifying an emotion.

Based on the emotion identified from module 1 and using the tagged music dataset, top k tracks with the same emotion get recommended to the user.

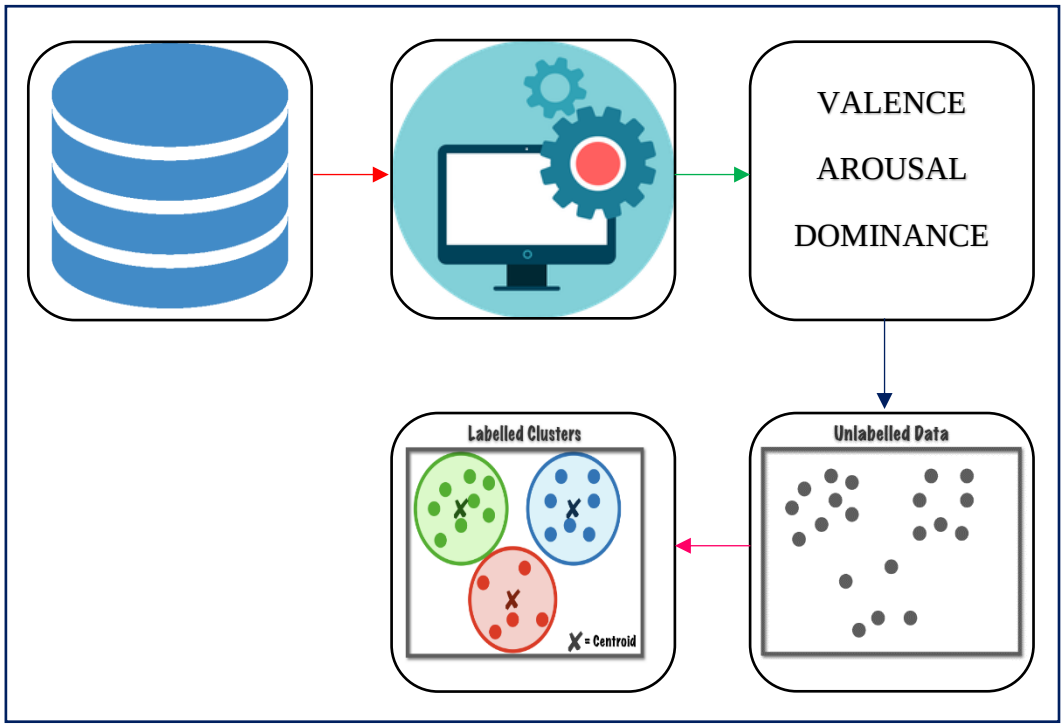

Figure 3.2.1: Music Recommendation Module.

3.3. User Interface Module

Through a web-based interface, the user interface module coordinates all interactions between users and our system. Users can capture their emotions using the camera module. Based on the user's emotion top 15 tracks are displayed as Spotify widgets, which get played on clicking the widget.

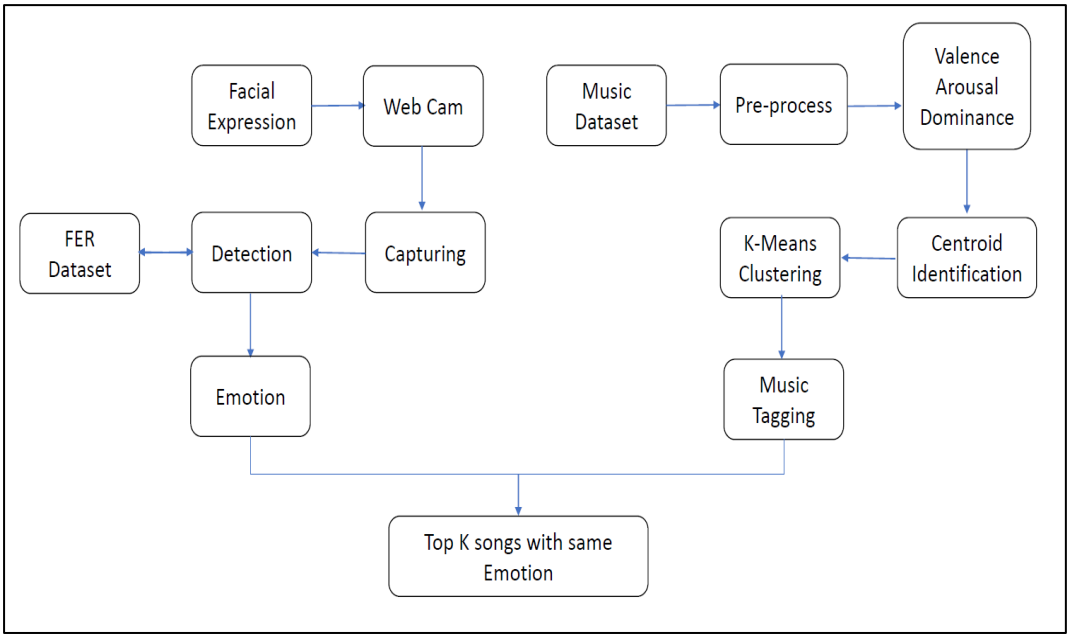

Figure 3.3.1 User-Interface Module.

4. Pseudo Algorithm/Implementation

4.1. Emotion Identification

This stage comprises training a neural network on a large number of visual pictures consisting of facial data to determine the emotion associated with each facial expression.

Rather than having to develop all of the algorithms from scratch, Python allows us to simply import the relevant modules and train models by giving the required inputs. As a result, we decided to use Python, a high-level programming language, to easily create and train Neural Networks.

4.1.1. Modules Used

- **Programming Language:** Python 3.10.4
- **Python Modules**
    1. FER
    2. DeepFace
    3. NumPy
    4. TensorFlow
    5. Sci-kit learn
    6. OpenCV
    7. Keras: Python deep learning API
    8. OS
    9. Time
    10. Sys Module
    11. Shutil
- **Operating System:** Ubuntu 20.04.4 LTS (Focal Fossa)
- **Source-code Editor:** Microsoft Visual Studio Code

4.1.2. CNN For Facial Emotion Identification

Convolutional Neural Networks use the hierarchical pattern in data to build extremely sophisticated patterns from smaller and simpler patterns imprinted in its filters. As a corollary, CNNs reside on the lower end of the interconnectedness and complexity spectrum.

Compiling a model, which is effectively a neural network, is the first step in implementing a Convolutional Neural Network. The following are the parameters that have to be defined when compiling a model.

- **Optimizers** : Define Weights and Learning Rate
- **Losses** : Choose what to minimize when training
- **Metrics** : Function used to Judge the performance of model
- **Train Set** : Set of Labelled Images used to train the Neural Network
- **Validation Set** : Set of Labelled Images used to validate the Neural Network

We chose the following settings to train our Neural Network.

Parameter	Value
Optimizer	`'adam'`
Losses	`'categorical_crossentroy'`
Metrics	`'accuracy'`
Train Set	FER 2013 Dataset
Validation Set	AffectNet Dataset

Table 4.1.2.1: Arguments passed to Keras Model.

Post Training, the model was tested against the AffectNet Dataset, consisting of around 49K labelled, coloured images of size 224 x 224. The script used for testing the accuracy of the Trained model follows.

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import (
    ImageDataGenerator,
)

from sklearn.metrics import (
    confusion_matrix,
    classification_report,
)
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, roc_auc_score

from IPython.display import clear_output
import warnings

warnings.filterwarnings("ignore")
import cv2
from os import listdir
import shutil

SEED = 12
IMG_HEIGHT = 64
IMG_WIDTH = 64
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 20
LR = 0.01
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA = 3
CLASS_LABELS = [
    "Anger",
    "Disgust",
    "Fear",
    "Happy",
    "Neutral",
    "Sad",
    "Surprise",
]

path = (
    "../Datasets/Facial Emotion Recognition/AffectNet/Annotated/images/"
)

model = tf.keras.models.load_model("best_model.h5")


def get_emotion(path: str) -> str:
    preprocess_fun = tf.keras.applications.densenet.preprocess_input
    test_datagen = ImageDataGenerator(
        rescale=1.0 / 255,
        validation_split=0.2,
        preprocessing_function=preprocess_fun,
    )
    test_generator = test_datagen.flow_from_directory(
        directory=path,
        target_size=(IMG_HEIGHT, IMG_WIDTH),
        batch_size=BATCH_SIZE,
        shuffle=False,
        color_mode="rgb",
        class_mode="categorical",
        seed=12,
    )
```

Figure 4.1.2.1: Script to find the Accuracy of Trained CNN Model (1).

```
    )
    results = model.predict(test_generator)
    freq = [0] * 7
    for result in results:
        freq[np.argmax(result)] += 1

    return CLASS_LABELS[np.argmax(freq)]


def main():
    labels = listdir(path)

    total_correct = 0
    total_present = 0

    for label in labels:
        if label == "Contempt":
            continue
        else:
            destination = "Stream/Stream/"

            label_correct = 0
            label_present = 0
            print(
                "Testing:",
                label,
                flush=True,
                file=open("LOG.txt", "a"),
            )
            for image in listdir(path + label):
                shutil.copy(
                    path + label + "/" + image,
                    destination + "cap.jpg",
                )
                got = get_emotion("Stream")
                expected = label
                if got.lower() == expected.lower():
                    label_correct += 1
                    total_correct += 1
                label_present += 1
                total_present += 1
            print(
                "Accuracy for",
                label,
                ":",
                label_correct / label_present,
                flush=True,
                file=open("LOG.txt", "a"),
            )
```

Figure 4.1.2.2: Script to find the Accuracy of Trained CNN Model (2).

```
            )
            print(
                "Label Present:",
                label_present,
                flush=True,
                file=open("LOG.txt", "a"),
            )
            print(
                "Label Correct:",
                label_correct,
                flush=True,
                file=open("LOG.txt", "a"),
            )
            print("", flush=True, file=open("LOG.txt", "a"))
    print(
        "Total Accuracy:",
        total_correct / total_present,
        flush=True,
        file=open("LOG.txt", "a"),
    )
    print(
        "Total Present:",
        total_present,
        flush=True,
        file=open("LOG.txt", "a"),
    )
    print(
        "Total Correct:",
        total_correct,
        flush=True,
        file=open("LOG.txt", "a"),
    )


main()

```

Figure 4.1.2.3: Script to find the Accuracy of Trained CNN Model (3).

4.1.3. Deep Neural Network

An artificial neural network (ANN) having numerous layers between the input and output layers is called a deep neural network (DNN). Deep Neural Networks don't loop back on themselves. As a result, they are a form of feedforward network, in which data flows from input to output.

Fortunately, a module for Facial Emotion Identification is included in Deep Face, a lightweight face recognition and facial attribute analysis framework. This module contains Deep Neural Network models that have been trained on several facial images.

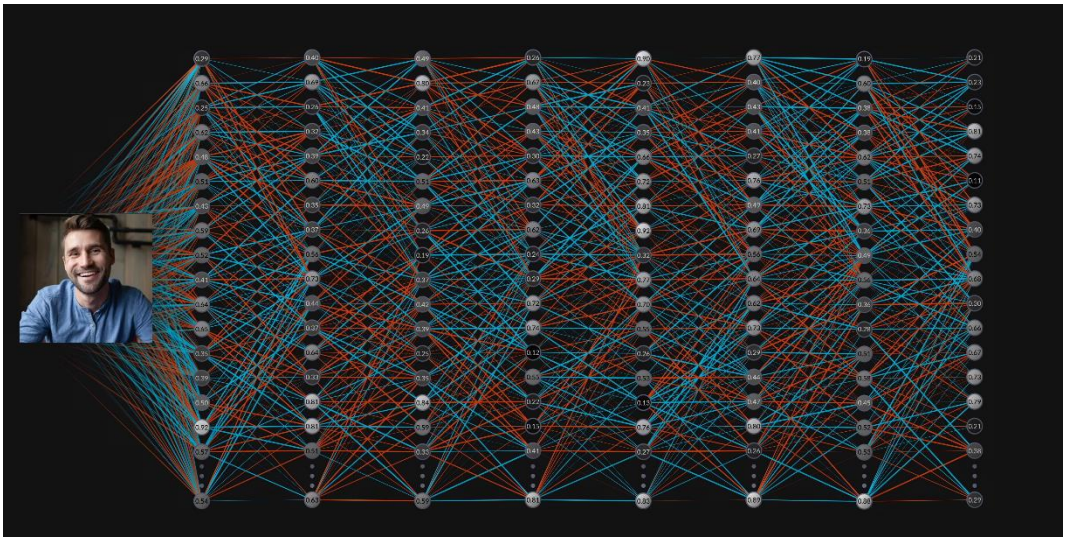

Figure 4.1.3.1: Deep Neural Network.

The Deep Face module makes detecting emotion through facial expression quite simple. The sample script that follows will probably be sufficient to identify the emotion.

```python
import cv2
from deepface import DeepFace

def get_emotion(img) -> str:
    try:
        result = DeepFace.analyze(img, actions = ["emotion"])
        result = result["dominant emotion"]
    except:
        return "No face detected"

if __name__ == '__main__':
    img_path = '/pictures/Test.jpg'
    img = cv2.imread(img_path)
    emotion = get_emotion(img)
    print("Identified Emotion: ", emotion)

```

Figure 4.1.3.2: Sample Snippet demonstrating the Deep Face Module.

Script used for Testing the accuracy of the DNN based Deep Face Model:

```python
import cv2
from deepface import DeepFace
import os

path = (
    "../Datasets/Facial Emotion Recognition/AffectNet/Annotated/images/"
)


def get_emotion(img) -> str:
    try:
        result = DeepFace.analyze(img, actions=["emotion"])
        return result["dominant_emotion"]
    except:
        return "No face detected"


def from_file(img_path: str) -> str:
    array = cv2.imread(img_path)
    emotion = get_emotion(array)
    return emotion


def validate():
    labels = os.listdir(path)
    total_correct = 0
    total_present = 0
    for label in labels:
        emotion_correct = 0
        emotion_total = 0
        if label == "Contempt":
            continue

        actual_emotion = label.lower()
        if actual_emotion == "anger":
            actual_emotion = "angry"

        print("Currently Testing " + str(label), flush=True)
        print(flush=True)

        for img in os.listdir(path + label):
            emotion = from_file(path + label + "/" + img)
            if emotion.lower() == actual_emotion:
                emotion_correct += 1
                total_correct += 1
            emotion_total += 1
            total_present += 1
```

Figure 4.1.3.3: Script used for Testing the accuracy of the Deep Face Model (1).

```
47              total_present += 1
48
49              if total_present % 1000 == 0:
50                  print(
51                      "Processed "
52                      + str(total_present)
53                      + " images so far",
54                      flush=True,
55                  )
56                  print(
57                      "Current Accuracy: "
58                      + str(total_correct / total_present),
59                      flush=True,
60                  )
61                  print(flush=True)
62
63          print(
64              "Accuracy for "
65              + label
66              + ": "
67              + str(emotion_correct / emotion_total),
68              flush=True,
69          )
70          print(
71              "Details: "
72              + str(emotion_correct)
73              + "/"
74              + str(emotion_total),
75              flush=True,
76          )
77          print(flush=True)
78
79      print(
80          "Total Accuracy: " + str(total_correct / total_present),
81          flush=True,
82      )
83      print("Total Correct: " + str(total_correct), flush=True)
84      print("Total Present: " + str(total_present), flush=True)
85      print(flush=True)
86
87
88  if __name__ == "__main__":
89      validate()
90
```

Figure 4.1.3.4: Script used for Testing the accuracy of the Deep Face Model (2).

4.1.4. Multi-Task Cascaded Neural Network

MTCNN stands for Multi-task Cascaded Convolutional Networks, which was created as a solution for both face identification and face alignment. The method includes three levels of convolutional networks that can detect faces and landmarks such as the eyes, nose, and mouth.

MTCNN has the benefit of automatically aligning poorly aligned faces. The precision may be enhanced even further this way. All of the slightly misaligned photos may now be fully aligned, and a rudimentary model can be used to identify face emotions.

A model had been trained and compiled for the Facial Emotion Recognition challenge (FER). The FER module, like the deep face module, is publicly available through Pip, Python's package installer. The sample below will probably be sufficient to identify the emotion through the FER's Emotion Detector.

```
1  from fer import FER
2  import cv2
3
4  img = cv2.imread("Sample.jpg")
5  detector = FER()
6  detector.detect_emotions(img)
```

Figure 4.1.4.1: Sample Snippet demonstrating MTCNN Model.

Script used for Testing the accuracy of the MTCNN based FER model follows. The emotion 'Contempt' is considered as a rare emotion; hence it is skipped when evaluation the Accuracy of the model.

```python
from fer import FER
import cv2
from sys import argv
from os import listdir

path = (
    "../Datasets/Facial Emotion Recognition/AffectNet/Annotated/images/"
)


def get_emotion(img_path: str) -> str:
    try:
        img = cv2.imread(img_path)
        detector = FER()
        emotions = detector.detect_emotions(img)
        # print(emotions)
        weights = emotions[0]["emotions"]
        weights = [(weights[emotion], emotion) for emotion in weights]
        weights.sort()
        return weights[-1][1]
    except:
        return "No face"


def main():
    labels = listdir(path)

    total_correct = 0
    total_present = 0

    for label in labels:
        if label == "Contempt":
            continue
        else:
            label_correct = 0
            label_present = 0
            print("Testing:", label, flush=True)
            for image in listdir(path + label):
                got = get_emotion(path + label + "/" + image)
                expected = label.lower()
                if expected == "anger":
                    expected = "angry"
                if got == expected:
                    label_correct += 1
                    total_correct += 1
                label_present += 1
                total_present += 1
            accuracy = label_correct / label_present
            print("Accuracy for", label, ":", accuracy, flush=True)
            print("Label Present:", label_present, flush=True)
            print("Label Correct:", label_correct, flush=True)
            print("", flush=True)
    print(
        "Total Accuracy:",
        total_correct / total_present,
        flush=True,
    )
    print("Total Present:", total_present, flush=True)
    print("Total Correct:", total_correct, flush=True)


main()
```

Figure 4.1.4.2: Script to find Accuracy of MTCNN Model.

4.2. Music Recommendation

Content-based filtering is used to recommend music. Since there are no publicly available datasets on users' emotions while listening to songs, collaborative filtering cannot be used. Hence content-based filtering approach is proposed for music recommendation.

K-Means clustering was done on the MuSe Dataset, as mentioned in the System Design portion of this publication. The Silhouette Score for Clustering was calculated using the snippets below. It is almost impossible to determine the accuracy since it requires real-time input from users' recommendations.

```python
import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import numpy, pandas
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import UTILS

constants = UTILS.constants


def get_data_frame() -> pandas.DataFrame:
    print("Parsing Dataset...")
    path = constants["Music_Dataset"]
    dataframe = pandas.read_csv(path)
    dataframe = dataframe[dataframe["spotify_id"].notna()]
    return dataframe


def get_static_centroids() -> tuple:
    coordinates = constants["coordinates"]
    coordinates = [(key, value) for key, value in coordinates.items()]
    centroids = [list(item[1]) for item in coordinates]
    centroids = numpy.array(centroids)
    labels = [item[0] for item in coordinates]
    return (labels, centroids)


def split_dataframe(dataframe: pandas.DataFrame, train_size=0.80):
    train_percent = int(train_size * 100)
    test_percent = 100 - train_percent
    info = "Splitting Dataframe into Train({}\%) and Test({}\%) Set..."
    info = info.format(train_percent, test_percent)
    print(info)
    train_dataframe = dataframe.sample(frac=train_size, random_state=50)
    test_dataframe = dataframe.drop(train_dataframe.index)
    return (train_dataframe, test_dataframe)

```

Figure 4.2.1: Script to cluster using K-MEANS (1).

```
39
40
41  def KMeansAll(dataframe: pandas.DataFrame):
42      print("Performing K-Means Clustering without initial centroids...")
43      x = dataframe.iloc[:, 5:8].values
44      k_means_optimum = KMeans(
45          n_clusters=7,
46          n_init=1,
47          init="k-means++",
48          random_state=50,
49          tol=1e-8,
50      )
51      y = k_means_optimum.fit_predict(x)
52      # dataframe['cluster'] = y
53      score = silhouette_score(x, y)
54      print("Silhouette score: ", score)
55
56
57  def KMeans_given_Initial_Centroids(
58      dataframe: pandas.DataFrame, centroids: numpy.array
59  ):
60      print("Performing K-Means Clustering with initial centroids...")
61      x = dataframe.iloc[:, 5:8].values
62      k_means_optimum = KMeans(
63          n_clusters=7,
64          n_init=1,
65          init=centroids,
66          random_state=50,
67          tol=1e-8,
68      )
69      y = k_means_optimum.fit_predict(x)
70      # dataframe['cluster'] = y
71      score = silhouette_score(x, y)
72      print("Silhouette score: ", score)
73
74
75  def KMeans_divided_dataset(
76      whole: pandas.DataFrame,
77      train: pandas.DataFrame,
78      test: pandas.DataFrame,
79  ):
80      print("Performing K-Means Clustering on Train and Test Dataset...")
81      x = whole.iloc[:, 5:8].values
82      k_means_optimum = KMeans(
83          n_clusters=7,
84          n_init=1,
85          init="k-means++",
86          random_state=50,
87          tol=1e-8,
88      )
89      y = k_means_optimum.fit_predict(x)
90      whole["cluster"] = y
91      original_centroids = k_means_optimum.cluster_centers_
92      original_centroids = sorted(
93          [list(triple) for triple in original_centroids]
94      )
95
```

Figure 4.2.2: Script to cluster using K-MEANS (2).

```
95
96      train_x = train.iloc[:, 5:8].values
97      k_means_train = KMeans(
98          n_clusters=7,
99          init="k-means++",
100         random_state=50,
101         tol=1e-8,
102     )
103     y_train = k_means_train.fit_predict(train_x)
104     train["cluster"] = y_train
105     train_centroids = k_means_train.cluster_centers_
106     train_centroids = sorted(
107         [list(triple) for triple in train_centroids]
108     )
109
110     total_correct = 0
111     total_present = 0
112
113     for row in test.itertuples():
114         valence, arousal, dominance = map(
115             float,
116             [
117                 row.valence_tags,
118                 row.arousal_tags,
119                 row.dominance_tags,
120             ],
121         )
122         nearest1 = None
123         nearest_distance = 10**10
124         for i in range(len(original_centroids)):
125             x1, y1, z1 = original_centroids[i]
126             x2, y2, z2 = valence, arousal, dominance
127             distance_ = (x1 - x2) ** 2 + (y1 - y2) ** 2 + (z1 - z2) ** 2
128             if distance_ < nearest_distance:
129                 nearest_distance = distance_
130                 nearest1 = i
131
132         nearest2 = None
133         nearest_distance = 10**10
134         for i in range(len(train_centroids)):
135             x1, y1, z1 = train_centroids[i]
136             x2, y2, z2 = valence, arousal, dominance
137             distance_ = (x1 - x2) ** 2 + (y1 - y2) ** 2 + (z1 - z2) ** 2
138             if distance_ < nearest_distance:
139                 nearest_distance = distance_
140                 nearest2 = i
141         if nearest1 == nearest2:
142             total_correct += 1
143         total_present += 1
144
145     print("Total Rows in Test Set: ", total_present)
146     print("Total Correctly Classified: ", total_correct)
147     print("Accuracy: %.3f" % (100 * total_correct / total_present))
148
149
```

Figure 4.2.3: Script to cluster using K-MEANS (3).

```
148
149
150 if __name__ == "__main__":
151     dataframe = get_data_frame()
152
153     KMeansAll(dataframe.copy(deep=True))
154     print()
155
156     labels, centroids = get_static_centroids()
157     KMeans_given_Initial_Centroids(dataframe.copy(deep=True), centroids)
158     print()
159
160     train_df, test_df = split_dataframe(dataframe.copy(deep=True))
161     KMeans_divided_dataset(dataframe.copy(deep=True), train_df, test_df)
162
```

Figure 4.2.4: Script to cluster using K-MEANS (4).

4.3. The Backend

Python was used to create the application's backend. It primarily comprises of MuSe Dataset pre-processing algorithms, Emotion recognition tools, and K-Means Clustering utilities. The script also includes a list of Constants to assist reviewers.

```python
import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import numpy, pandas, collections, fer, random
from sklearn.cluster import KMeans

frame = '<iframe src="https://open.spotify.com/embed/track/{}" '
frame += (
    'width="260" height="380" frameborder="0" allowtransparency="true" '
)
frame += 'allow="encrypted-media"></iframe>'

constants = {
    "Music_Dataset": "Datasets/Music Recommendation/muse_v3.csv",
    "emotion_model": fer.FER(),
    "frame": frame,
    "coordinates": {
        "angry": (2.50, 5.93, 5.14),
        "happy": (8.21, 5.55, 7.00),
        "surprise": (7.21, 7.54, 7.25),
        "disgust": (1.69, 3.33, 4.46),
        "fear": (2.97, 5.16, 2.87),
        "sad": (2.40, 2.81, 3.84),
        "neutral": (4.12, 3.38, 4.43),
    },
    "colors": {
        "angry": "#ff3300",
        "happy": "#007399",
        "surprise": "#e13a37",
        "disgust": "#663300",
        "fear": "#666699",
        "sad": "#6b6b47",
        "neutral": "#862d2d",
        "left": "black",
    },
}


def detect_emotion(img: numpy.ndarray) -> str:
    try:
        emotion_detection_model = constants["emotion_model"]
        emotions = emotion_detection_model.detect_emotions(img)
        weights = emotions[0]["emotions"]
        weights = [(weights[emotion], emotion) for emotion in weights]
        weights.sort()
        return weights[-1][1]
    except:
        return None

```

Figure 4.3.1: Backend Script to Pre-process MuSe Dataset (1).

```
50
51
52  def distance(
53      x1: float,
54      y1: float,
55      z1: float,
56      x2: float,
57      y2: float,
58      z2: float,
59  ) -> float:
60      return (x1 - x2) ** 2 + (y1 - y2) ** 2 + (z1 - z2) ** 2
61
62
63  def map_emotion(
64      valence: float, arousal: float, dominance: float
65  ) -> tuple:
66      min_distance = 10**10
67      closest_emotion = None
68      for emotion in constants["coordinates"]:
69          x2, y2, z2 = constants["coordinates"][emotion]
70          distance_to_emotion = distance(
71              valence, arousal, dominance, x2, y2, z2
72          )
73          if distance_to_emotion < min_distance:
74              min_distance = distance_to_emotion
75              closest_emotion = emotion
76      return (closest_emotion, min_distance)
77
78
79  def get_centroids() -> tuple:
80      coordinates = constants["coordinates"]
81      coordinates = [(key, value) for key, value in coordinates.items()]
82      centroids = [list(item[1]) for item in coordinates]
83      centroids = numpy.array(centroids)
84      labels = [item[0] for item in coordinates]
85      return (labels, centroids)
86
87
88  def pre_process_static() -> dict:
89      dataframe = pandas.read_csv(constants["Music_Dataset"])
90      dataframe = dataframe[
91          [
92              "track",
93              "artist",
94              "genre",
95              "spotify_id",
96              "valence_tags",
97              "arousal_tags",
98              "dominance_tags",
99          ]
100     ]
101
102     tracks = collections.defaultdict(list)
103
104     for index, row in dataframe.iterrows():
105         valence, arousal, dominance = map(
106             float,
107             [
108                 row["valence_tags"],
109                 row["arousal_tags"],
110                 row["dominance_tags"],
111             ],
112         )
```

Figure 4.3.2: Backend Script to Pre-process MuSe Dataset (2).

```
        )
        spotify_id = row["spotify_id"]
        if pandas.notna(spotify_id):
            emotion, distance = map_emotion(valence, arousal, dominance)
            tracks[emotion].append((spotify_id, distance))

    for track in tracks:
        tracks[track].sort(key=lambda x: x[-1])
        tracks[track] = [item[0] for item in tracks[track]]

    return tracks


def pre_process_cluster() -> dict:
    data = pandas.read_csv(constants["Music_Dataset"])
    data = data[data["spotify_id"].notna()]
    labels, centroids = get_centroids()
    x = data.iloc[:, 5:8].values
    k_means_optimum = KMeans(
        n_clusters=7,
        n_init=1,
        init=centroids,
        random_state=50,
        tol=1e-8,
    )
    y = k_means_optimum.fit_predict(x)
    data["cluster"] = y
    tracks = collections.defaultdict(list)
    for index, row in data.iterrows():
        spotify_id = row["spotify_id"]
        cluster = row["cluster"]
        tracks[labels[cluster]].append(spotify_id)

    return tracks


# Uncomment the following line for
# static pre-processing
# Static pre-processing uses defined coordinates

# tracks = pre_process_static()

# The following relies on KMeans
# Clustering which gives better results

tracks = pre_process_cluster()


def get_top_k(emotion: str, k=15, sample_size=100) -> list:
    top_k = random.sample(tracks[emotion][:sample_size], k)
    return top_k

```

Figure 4.3.3: Backend Script to Pre-process MuSe Dataset (3).

4.4. The Frontend

Because the focus was on analysing the methodologies for constructing a Context-aware Recommendation System, little work was put into developing the Web Application's front-end. The Front-end of the application was built using Streamlit, a free Python module. Streamlit makes it very simple to code the front-end using Python's core capabilities.

```python
import streamlit, UTILS, PIL, numpy
import streamlit.components.v1 as components


def populate_tracks(tracks: list) -> None:
    tag = "<h4 style='text-align: center;'>Top Tracks</h4>"
    streamlit.markdown(tag, unsafe_allow_html=True)
    with streamlit.container():
        col1, col2, col3 = streamlit.columns([3, 3, 3])
        for i, track in enumerate(tracks):
            if i % 3 == 0:
                with col1:
                    components.html(frame.format(track), height=400)
            elif i % 3 == 1:
                with col2:
                    components.html(frame.format(track), height=400)
            else:
                with col3:
                    components.html(frame.format(track), height=400)


def populate_emotion(emotion: str) -> None:
    left_color = UTILS.constants["colors"]["left"]
    left_span = '<span style = "color: {}">Identified Emotion: </span>'
    left_span = left_span.format(left_color)

    emotion_color = UTILS.constants["colors"][emotion]
    right_span = '<span style = "color: {}">{}</span>'
    right_span = right_span.format(emotion_color, emotion.capitalize())

    tag = "<h2 style='text-align: center;'>{}</h2>"
    tag = tag.format(left_span + right_span)
    streamlit.markdown(tag, unsafe_allow_html=True)


```

Figure 4.4.1: Script to create User Interface using Streamlit (1).

```
22  def populate_emotion(emotion: str) -> None:
23      left_color = UTILS.constants["colors"]["left"]
24      left_span = '<span style = "color: {}">Identified Emotion: </span>'
25      left_span = left_span.format(left_color)
26
27      emotion_color = UTILS.constants["colors"][emotion]
28      right_span = '<span style = "color: {}">{}</span>'
29      right_span = right_span.format(emotion_color, emotion.capitalize())
30
31      tag = "<h2 style='text-align: center;'>{}</h2>"
32      tag = tag.format(left_span + right_span)
33      streamlit.markdown(tag, unsafe_allow_html=True)
34
35
36  if __name__ == "__main__":
37      tracks = UTILS.tracks
38      frame = UTILS.constants["frame"]
39      picture = streamlit.camera_input("")
40      if picture is not None:
41          picture = PIL.Image.open(picture)
42          pixels = numpy.array(picture)
43          emotion = UTILS.detect_emotion(pixels)
44          if emotion:
45              tracks = UTILS.get_top_k(emotion)
46              populate_emotion(emotion)
47              populate_tracks(tracks)
48          else:
49              tag = (
50                  '<h3 style="text-align: center;">No face detected</h3>'
51              )
52              streamlit.markdown(tag, unsafe_allow_html=True)
53
```

Figure 4.4.2: Script to create User Interface using Streamlit (2).

5. Results

5.1. Accurate Predictions

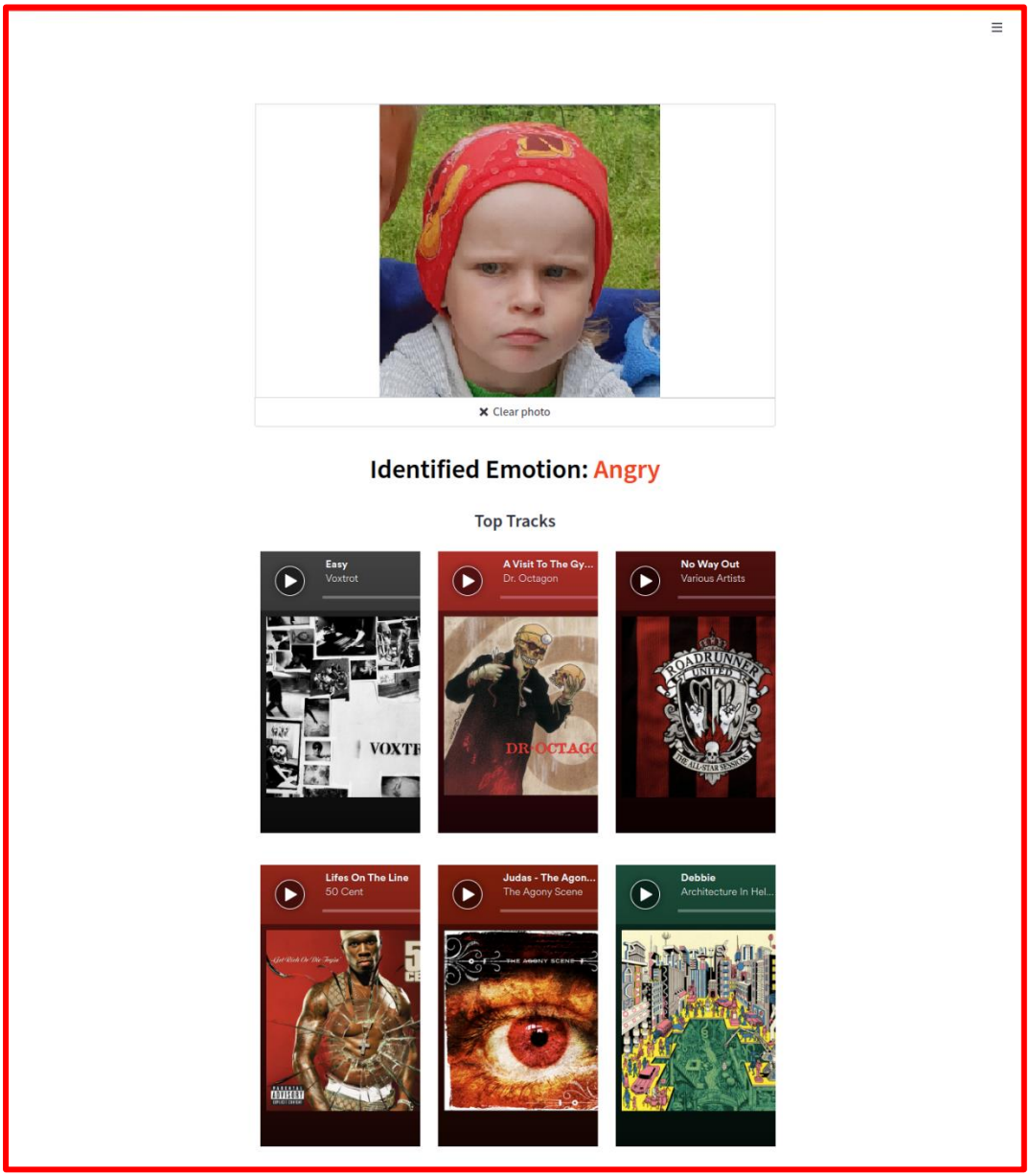

Figure 5.1.1: Application correctly Identifying Angry Emotion.

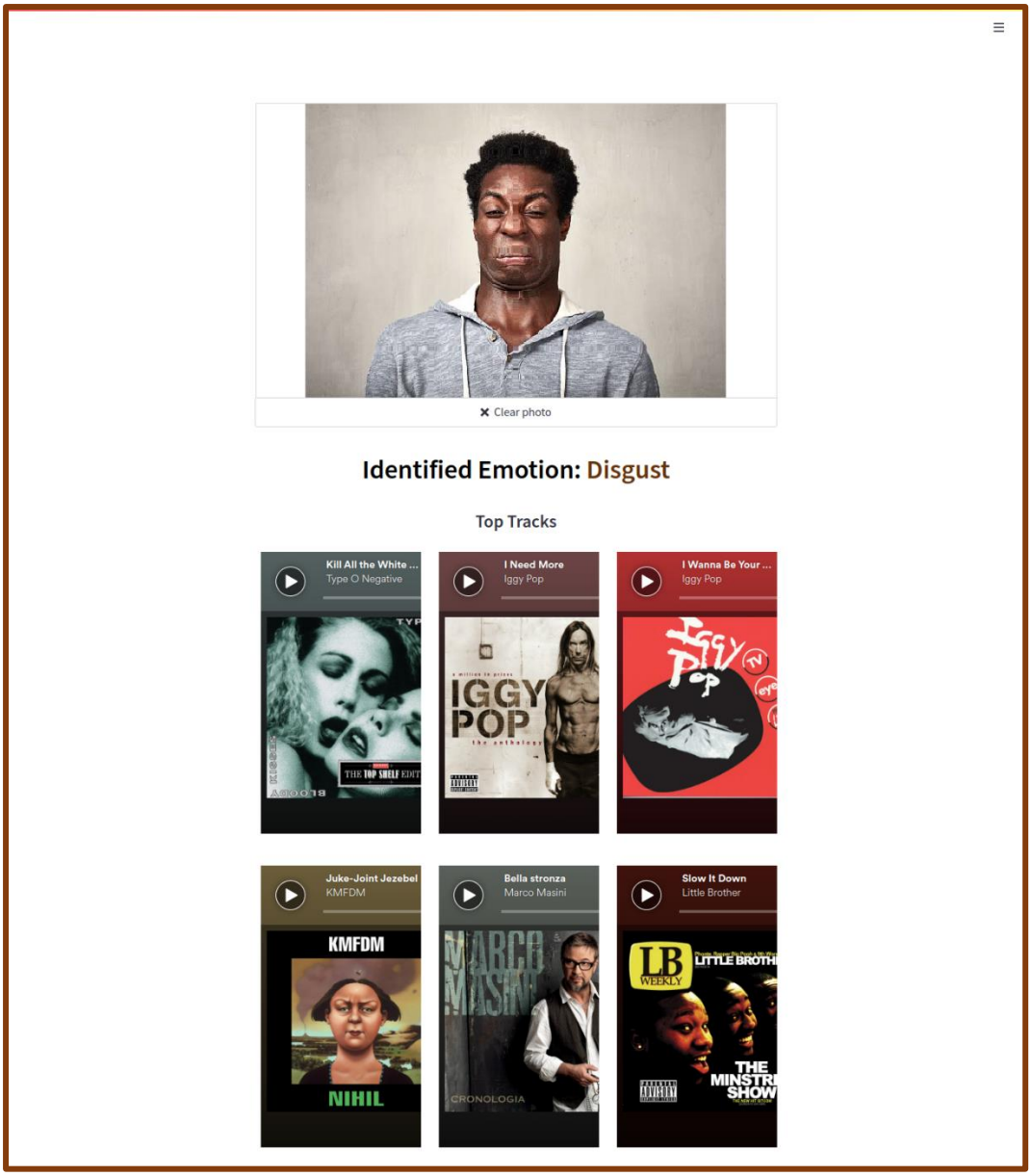

Figure 5.1.2: Application correctly Identifying Disgust Emotion.

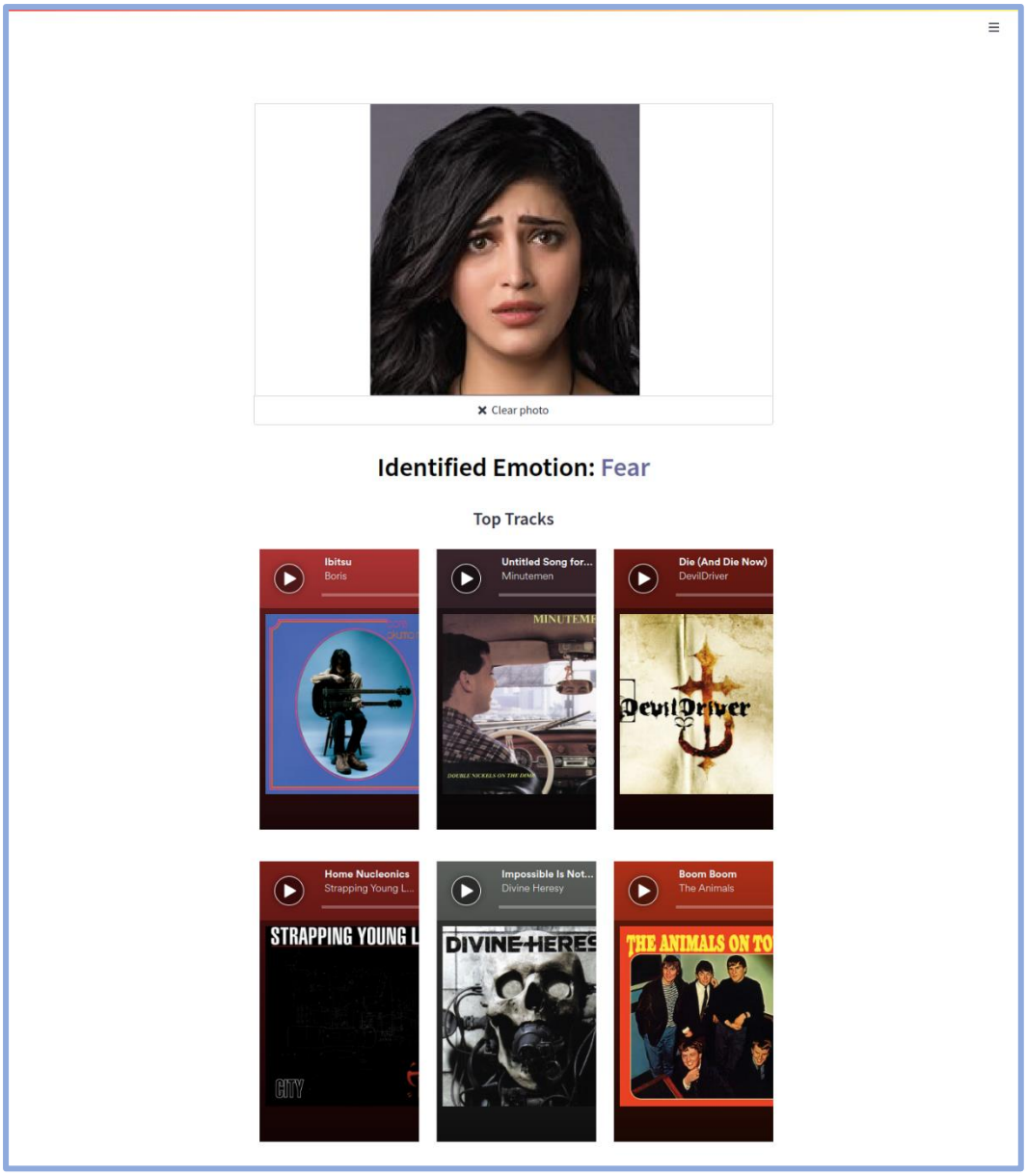

Figure 5.1.3: Application correctly Identifying Fear Emotion.

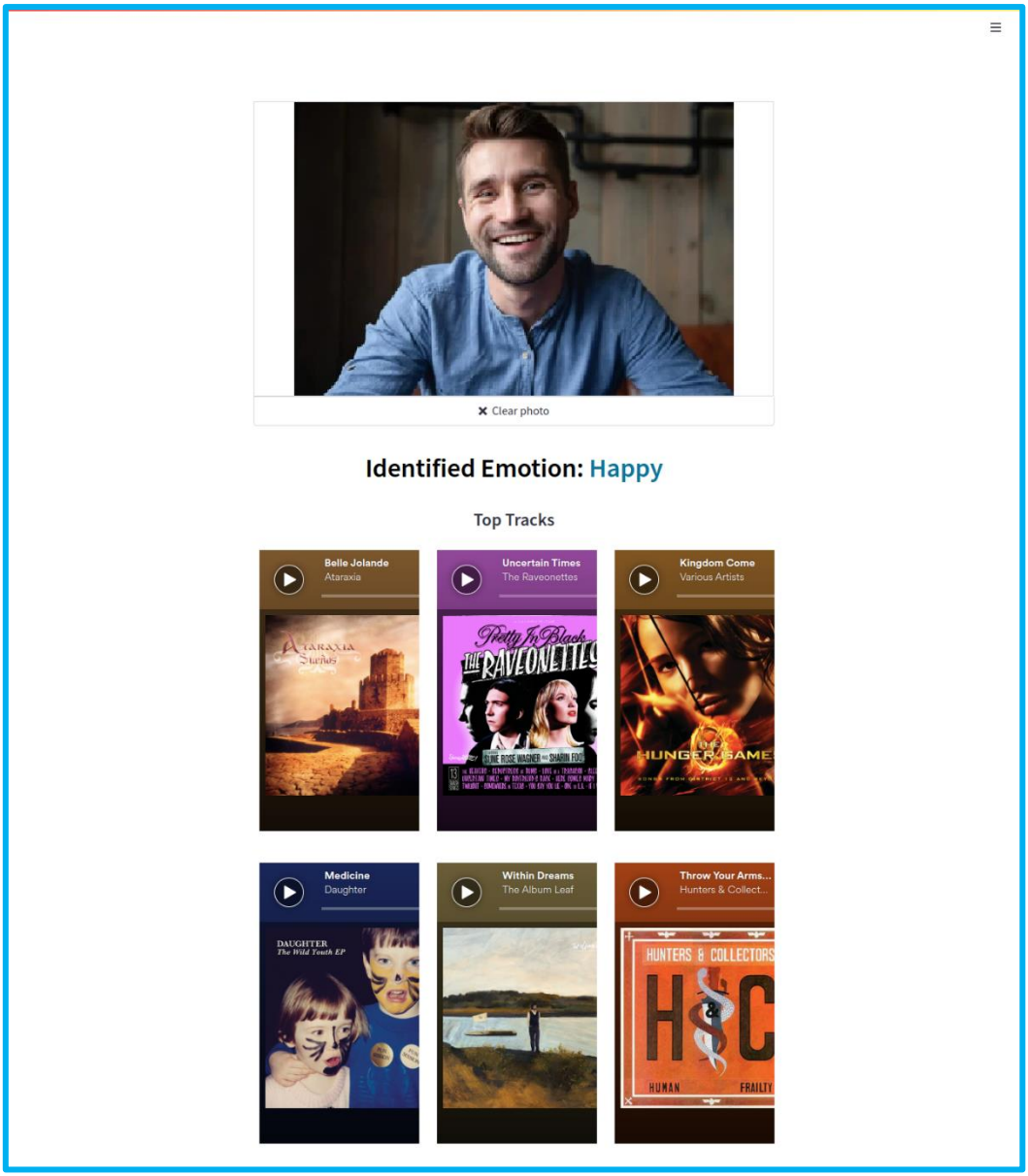

Figure 5.1.4: Application correctly Identifying Happy Emotion.

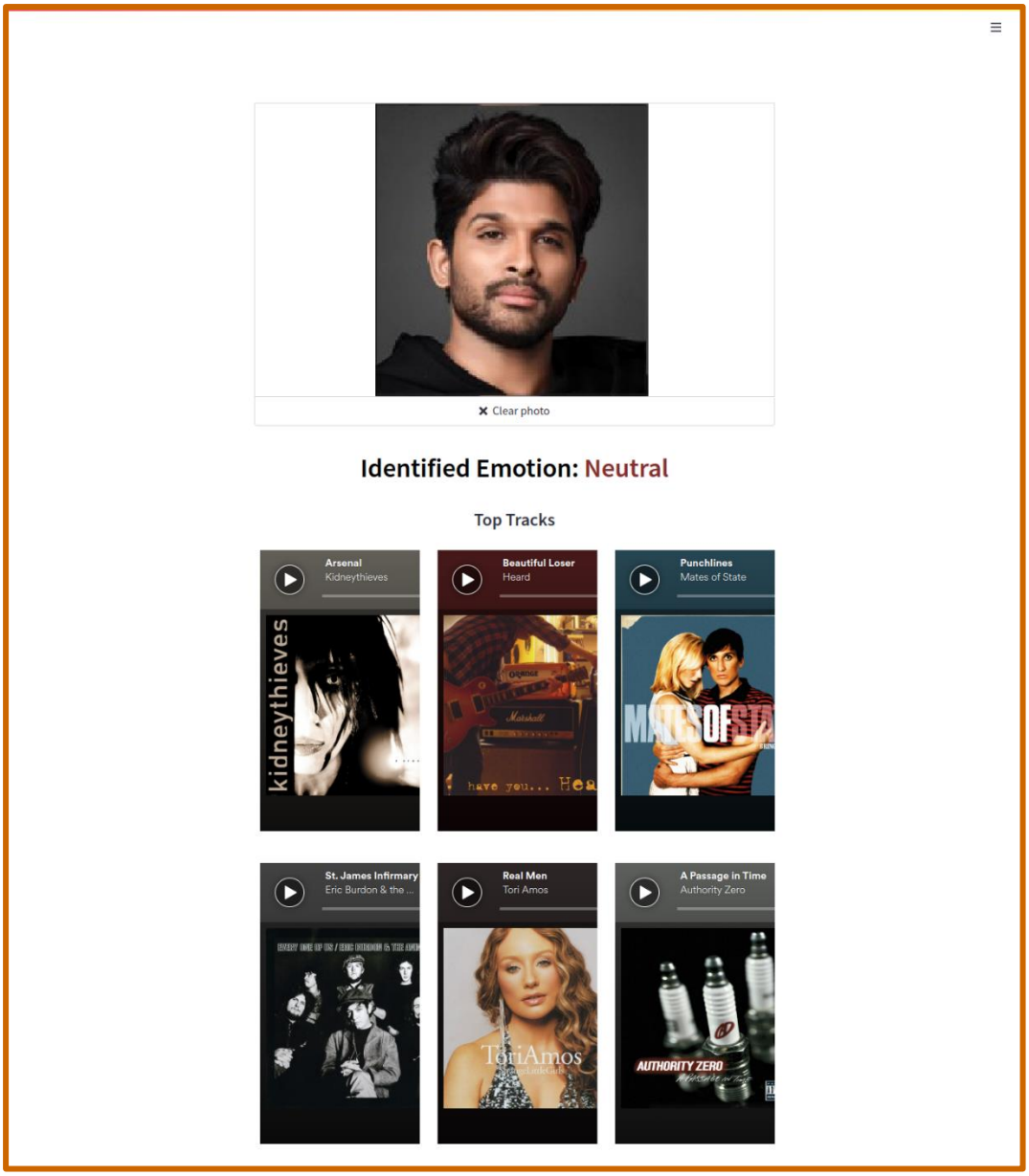

Figure 5.1.5: Application correctly Identifying Neutral Emotion.

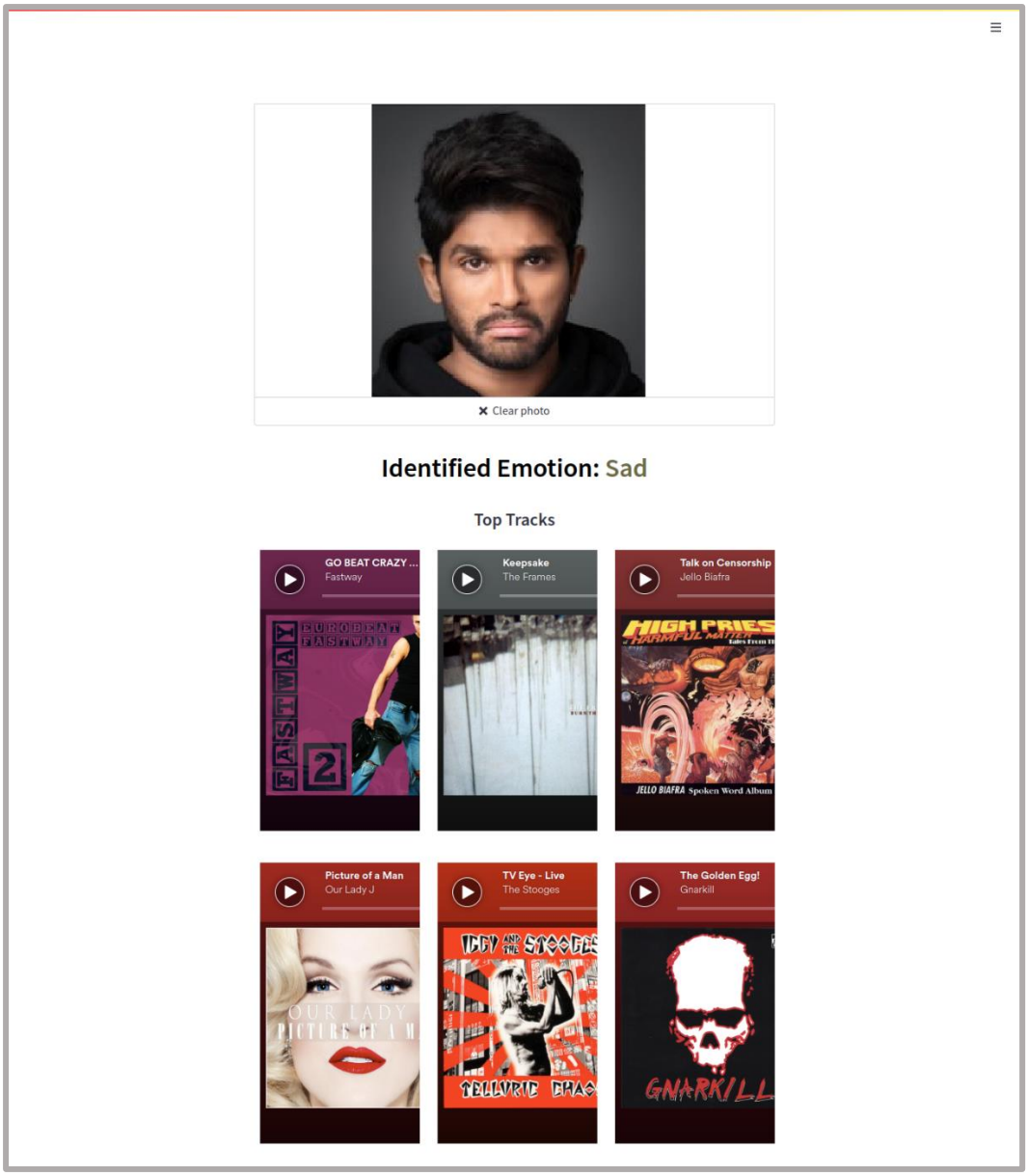

Figure 5.1.6: Application correctly Identifying Sad Emotion.

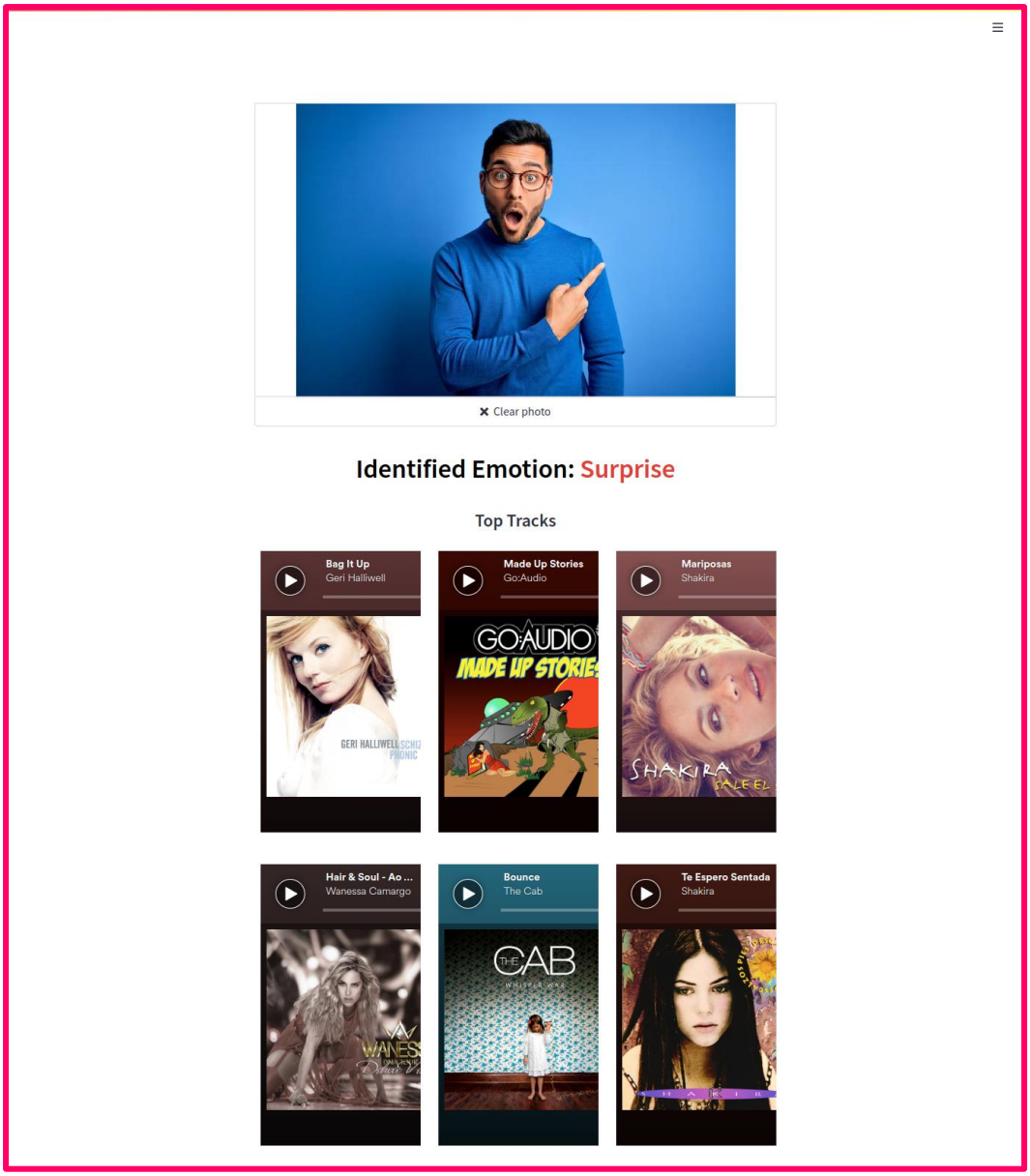

Figure 5.1.7: Application correctly Identifying Surprise Emotion.

5.2. Inaccurate Predictions

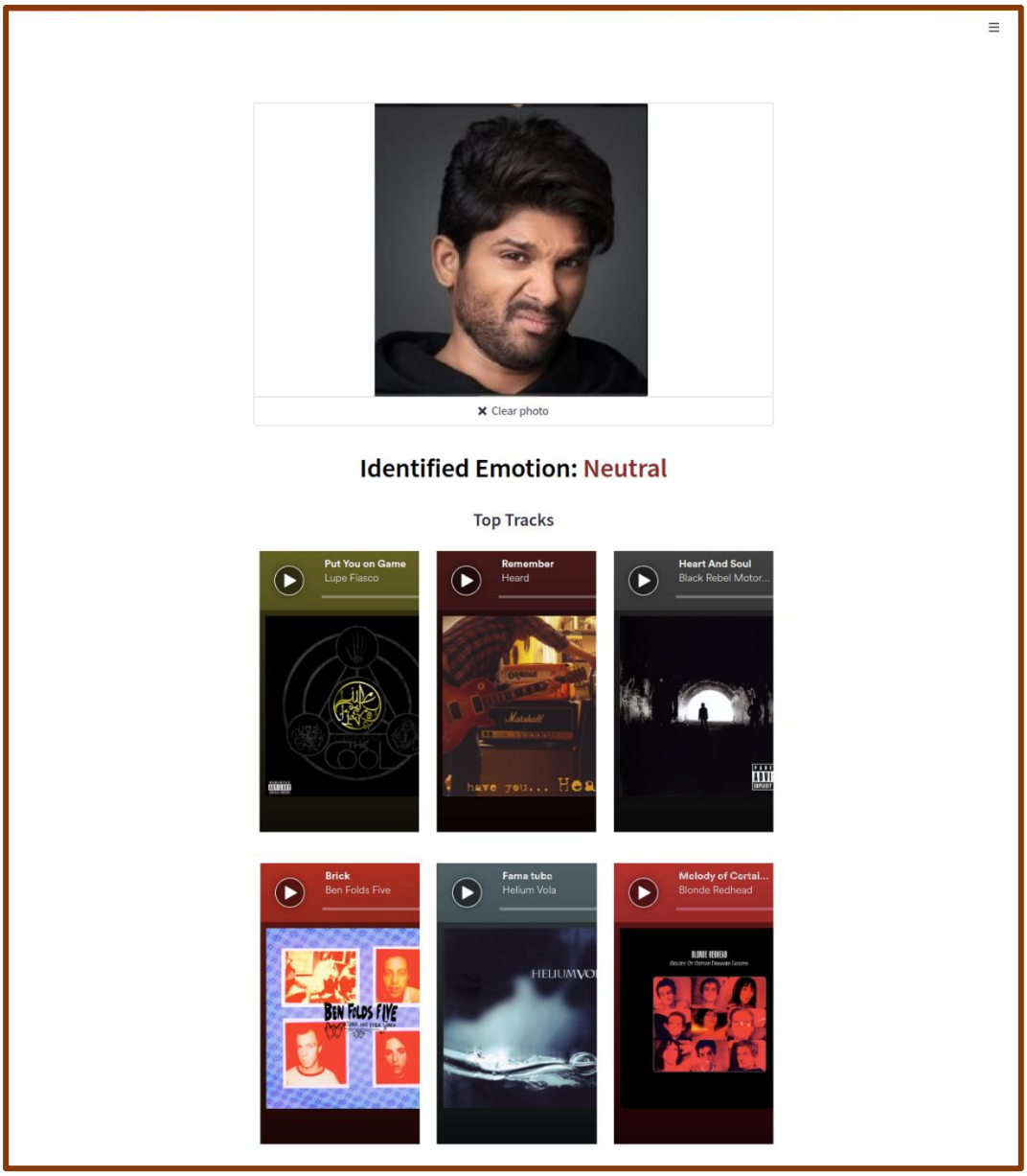

Figure 5.2.1: Application wrongly classifying Disgust as Neutral.

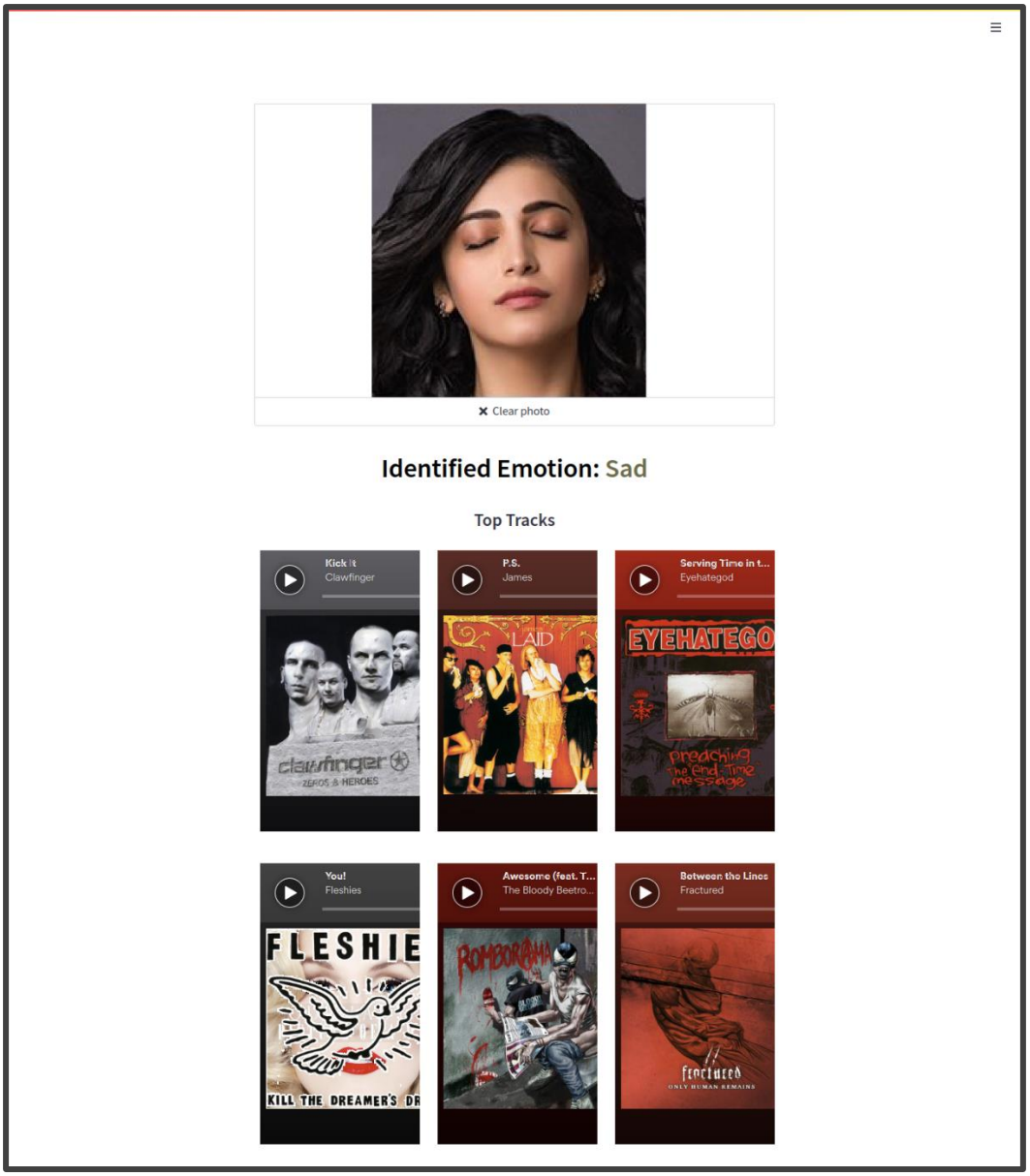

Figure 5.2.2: Application wrongly classifying Neutral as Sad.

5.3. No-Face-Detected

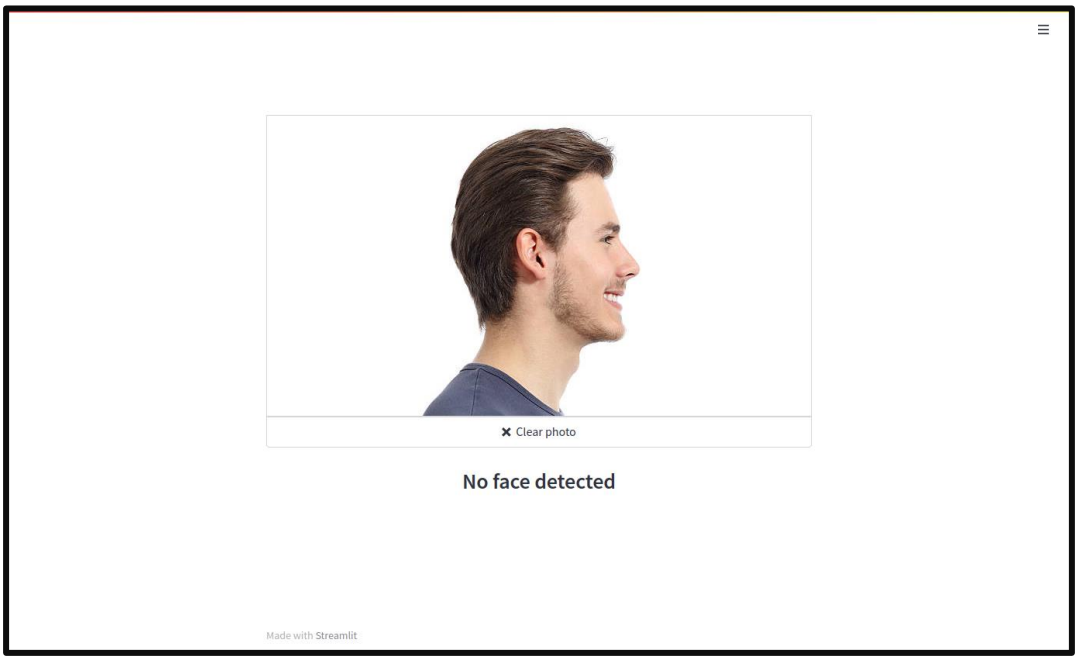

Figure 5.3.1: Application failing to detect Face.

6. Testing

6.1. Accuracy of CNN Model

```
Testing: Neutral
Accuracy for Neutral : 0.326
Label Present: 500
Label Correct: 163

Testing: Sad
Accuracy for Sad : 0.596
Label Present: 500
Label Correct: 298

Testing: Happy
Accuracy for Happy : 0.920
Label Present: 500
Label Correct: 460

Testing: Anger
Accuracy for Anger : 0.350
Label Present: 500
Label Correct: 175

Testing: Disgust
Accuracy for Disgust : 0.000
Label Present: 500
Label Correct: 0

Testing: Surprise
Accuracy for Surprise : 0.116
Label Present: 500
Label Correct: 58

Testing: Fear
Accuracy for Fear : 0.088
Label Present: 500
Label Correct: 44

Total Accuracy: 0.342
Total Present: 3500
Total Correct: 1198

```

Figure 6.1.1: Accuracy of CNN Model against AffectNet Test.

6.2. Accuracy for Deep Face Model

6.2.1. Dataset: AffectNet Test

```
Currently Testing Neutral

Processed 1000 images so far
Current Accuracy: 0.348

Processed 2000 images so far
Current Accuracy: 0.355

Processed 3000 images so far
Current Accuracy: 0.362

Processed 4000 images so far
Current Accuracy: 0.354

Processed 5000 images so far
Current Accuracy: 0.351

Processed 6000 images so far
Current Accuracy: 0.349

Processed 7000 images so far
Current Accuracy: 0.345

Processed 8000 images so far
Current Accuracy: 0.345

Processed 9000 images so far
Current Accuracy: 0.344

Processed 10000 images so far
Current Accuracy: 0.344

Processed 11000 images so far
Current Accuracy: 0.344

Processed 12000 images so far
Current Accuracy: 0.344

Processed 13000 images so far
Current Accuracy: 0.345

Accuracy for Neutral: 0.345
Details: 4527/13093

Currently Testing Sad

Processed 14000 images so far
Current Accuracy: 0.338

Processed 15000 images so far
Current Accuracy: 0.330

Processed 16000 images so far
Current Accuracy: 0.323

Processed 17000 images so far
Current Accuracy: 0.317

Accuracy for Sad: 0.223
Details: 1014/4529
```

Figure 6.2.1: Accuracy of Deep Face Model against AffectNet Test.

6.2.2. Dataset: AffectNet Train

```
61
62  Currently Testing Happy
63
64  Processed 18000 images so far
65  Current Accuracy: 0.318
66
67  Processed 19000 images so far
68  Current Accuracy: 0.330
69
70  Processed 20000 images so far
71  Current Accuracy: 0.338
72
73  Processed 21000 images so far
74  Current Accuracy: 0.347
75
76  Processed 22000 images so far
77  Current Accuracy: 0.355
78
79  Processed 23000 images so far
80  Current Accuracy: 0.363
81
82  Processed 24000 images so far
83  Current Accuracy: 0.369
84
85  Processed 25000 images so far
86  Current Accuracy: 0.376
87
88  Processed 26000 images so far
89  Current Accuracy: 0.381
90
91  Processed 27000 images so far
92  Current Accuracy: 0.386
93
94  Processed 28000 images so far
95  Current Accuracy: 0.390
96
97  Processed 29000 images so far
98  Current Accuracy: 0.395
99
100 Processed 30000 images so far
101 Current Accuracy: 0.400
102
103 Processed 31000 images so far
104 Current Accuracy: 0.404
105
106 Processed 32000 images so far
107 Current Accuracy: 0.407
108
109 Processed 33000 images so far
110 Current Accuracy: 0.411
111
112 Processed 34000 images so far
113 Current Accuracy: 0.414
114
115 Processed 35000 images so far
116 Current Accuracy: 0.417
117
118 Processed 36000 images so far
119 Current Accuracy: 0.421
120
121 Processed 37000 images so far
122 Current Accuracy: 0.424
```

Figure 6.2.2: Accuracy of Deep Face Model against AffectNet Train (1).

```
123
124  Processed 38000 images so far
125  Current Accuracy: 0.427
126
127  Processed 39000 images so far
128  Current Accuracy: 0.430
129
130  Processed 40000 images so far
131  Current Accuracy: 0.433
132
133  Accuracy for Happy: 0.525
134  Details: 12265/23325
135
136  Currently Testing Anger
137
138  Processed 41000 images so far
139  Current Accuracy: 0.434
140
141  Processed 42000 images so far
142  Current Accuracy: 0.428
143
144  Processed 43000 images so far
145  Current Accuracy: 0.422
146
147  Processed 44000 images so far
148  Current Accuracy: 0.416
149
150  Processed 45000 images so far
151  Current Accuracy: 0.411
152
153  Accuracy for Anger: 0.172
154  Details: 745/4314
155
156  Currently Testing Disgust
157
158  Accuracy for Disgust: 0.054
159  Details: 36/657
160
161  Currently Testing Surprise
162
163  Processed 46000 images so far
164  Current Accuracy: 0.404
165
166  Processed 47000 images so far
167  Current Accuracy: 0.397
168
169  Processed 48000 images so far
170  Current Accuracy: 0.391
171
172  Accuracy for Surprise: 0.091
173  Details: 216/2357
174
175  Currently Testing Fear
176
177  Processed 49000 images so far
178  Current Accuracy: 0.386
179
180  Accuracy for Fear: 0.204
181  Details: 225/1101
182
183  Total Accuracy: 0.385
184  Total Correct: 19028
185  Total Present: 49376
186
187
```

Figure 6.2.3: Accuracy of Deep Face Model against AffectNet Train (2).

```
123
124  Processed 38000 images so far
125  Current Accuracy: 0.427
126
127  Processed 39000 images so far
128  Current Accuracy: 0.430
129
130  Processed 40000 images so far
131  Current Accuracy: 0.433
132
133  Accuracy for Happy: 0.525
134  Details: 12265/23325
135
136  Currently Testing Anger
137
138  Processed 41000 images so far
139  Current Accuracy: 0.434
140
141  Processed 42000 images so far
142  Current Accuracy: 0.428
143
144  Processed 43000 images so far
145  Current Accuracy: 0.422
146
147  Processed 44000 images so far
148  Current Accuracy: 0.416
149
150  Processed 45000 images so far
151  Current Accuracy: 0.411
152
153  Accuracy for Anger: 0.172
154  Details: 745/4314
155
156  Currently Testing Disgust
157
158  Accuracy for Disgust: 0.054
159  Details: 36/657
160
161  Currently Testing Surprise
162
163  Processed 46000 images so far
164  Current Accuracy: 0.404
165
166  Processed 47000 images so far
167  Current Accuracy: 0.397
168
169  Processed 48000 images so far
170  Current Accuracy: 0.391
171
172  Accuracy for Surprise: 0.091
173  Details: 216/2357
174
175  Currently Testing Fear
176
177  Processed 49000 images so far
178  Current Accuracy: 0.386
179
180  Accuracy for Fear: 0.204
181  Details: 225/1101
182
183  Total Accuracy: 0.385
184  Total Correct: 19028
185  Total Present: 49376
186
187
```

Figure 6.2.4: Accuracy of Deep Face Model against AffectNet Train (3).

6.3. Accuracy of MTCNN based FER Model

```
Testing: Neutral
Accuracy for Neutral : 0.574
Label Present: 500
Label Correct: 287

Testing: Sad
Accuracy for Sad : 0.374
Label Present: 500
Label Correct: 187

Testing: Happy
Accuracy for Happy : 0.690
Label Present: 500
Label Correct: 345

Testing: Anger
Accuracy for Anger : 0.252
Label Present: 500
Label Correct: 126

Testing: Disgust
Accuracy for Disgust : 0.102
Label Present: 500
Label Correct: 51

Testing: Surprise
Accuracy for Surprise : 0.184
Label Present: 500
Label Correct: 92

Testing: Fear
Accuracy for Fear : 0.336
Label Present: 500
Label Correct: 168

Total Accuracy: 0.358
Total Present: 3500
Total Correct: 1256
```

Figure 6.3.1: Accuracy of MTCNN based FER model.

6.4. Accuracy for K-Means Clustering

```
Parsing Dataset...
Performing K-Means Clustering without initial centroids...
Silhouette score:  0.309

Performing K-Means Clustering with initial centroids...
Silhouette score:  0.318

Splitting Dataframe into Train(80%) and Test(20%) Set...
Performing K-Means Clustering on Train and Test Dataset...
Total Rows in Test Set:  12326
Total Correctly Classified:  9583
Accuracy: 77.746

```

Figure 6.4.1: Accuracy for K-MEANS Clustering.

6.5. Compare & Contrast Emotion detection models

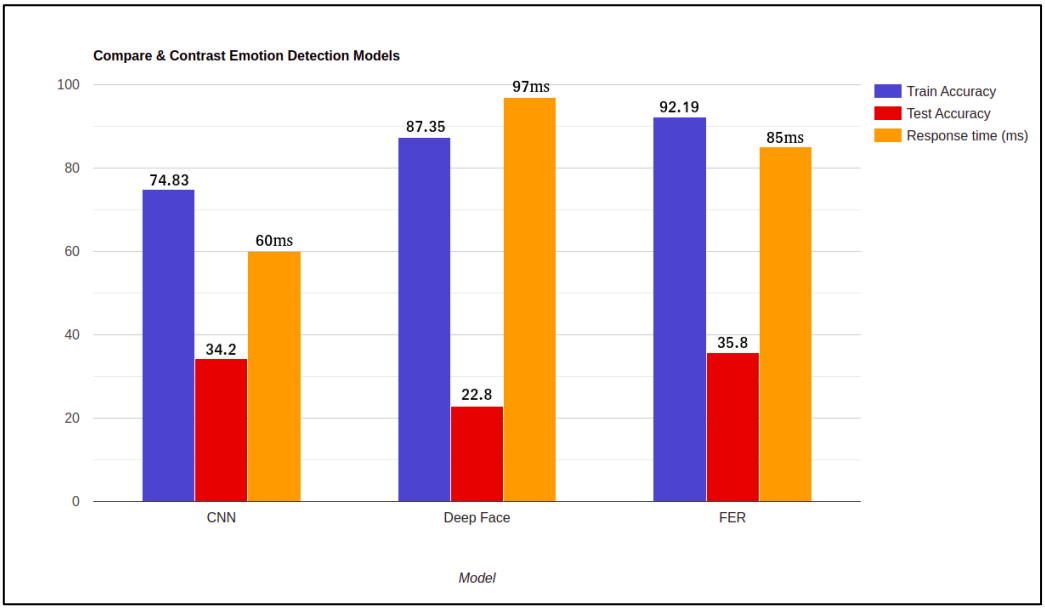

Figure 6.5.1: Compare & Contrast of Emotion Detection Models.

6.6. Emotion-wise Accuracies of all Models

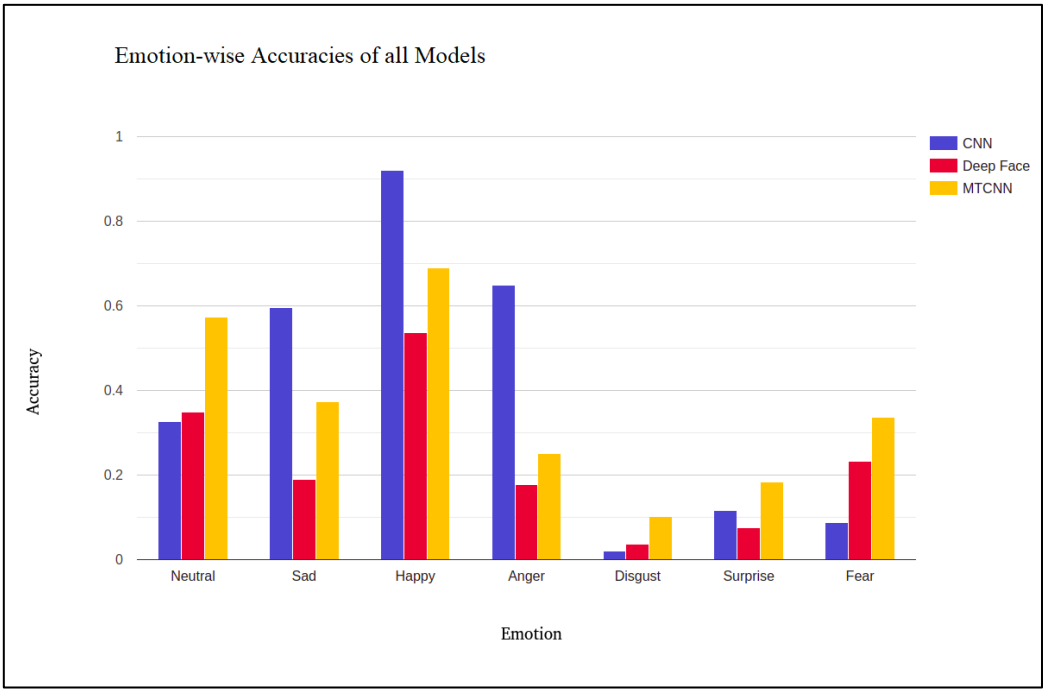

Figure 6.6.1: Emotion-wise Accuracies of all Models.

7. Conclusion and Future Work

General music recommendation systems are quite ineffective because they don't consider contextual information. The aim of this work was to build a comprehensive context-aware music recommendation system, which recommends music based on the user's emotion. The overarching goal of this novel system is to blend user emotion with music recommendation, to improve accuracy and enhance the listening experience.

After extensive research and experimentation, we have proposed an integrated system that uses the MTCNN model for emotion detection and content-based filtering for music recommendation. The results suggest that the performance of the system was on par with current state-of-the-art systems.

We acknowledge some shortcomings in our work that opens up possibilities for future work. We would like to explore the following for future work:

1. Incorporate additional contextual information such as mouse click patterns, keystrokes, and user heart rate.

2. Introduce a rating system where users can rate the recommended songs. This helps in improving the accuracy of further recommendations.

3. Improve the accuracy of the MTCNN model by training against extremely large datasets using GPUs.

References

[1]. Sunitha M, Adilakshmi T (2018) Music recommendation system with user-based and item-based collaborative filtering technique. In Networking Communication and Data Knowledge Engineering: Springer.

[2]. H.-C. Kwon and M. Kim. Lyrics-based emotion classification using feature selection by partial syntactic analysis. 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence (ICTAI 2011), 2011.

[3]. J. A. Speck, E. M. Schmidt, B. G. Morton, and Y. E. Kim. A comparative study of collaborative vs. traditional musical mood annotation. In A. Klapuri and C. Leider, editors, ISMIR, University of Miami, 2011.

[4]. K. Bischoff, C. Firan, R. Paiu, W. Nejdl, C. Laurier and M. Sordo. Music Mood and Theme Classification a Hybrid Approach. In Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009.

[5]. Hyung, Z. Park, J.S. Lee, K. Utilizing context-relevant keywords extracted from a large collection of user-generated documents for music discovery. Inf. Process. Manag. 2017.

[6]. M. V. Zaanen and P. Kanters. Automatic mood classification using tf*idf based on lyrics. In Proceedings of the 11th International Society for Music Information Retrieval Conference, Utrecht, The Netherlands, August 9-13, 2010.

[7]. R. Malheiro, R. Panda, P. Gomes, and R. Paiva. Bi-modal music emotion recognition: Novel lyrical features and dataset. In 9th International Workshop on Music and Machine Learning MML2016 in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECML/PKDD 2016.

[8]. Ji-Oh Yoo Han-Saem Park and Sung-Bae Cho. L. Wang. A context-aware MRS using fuzzy Bayesian networks with utility theory. In FSKD 2006.

[9]. Giz H. He, J. Jin, Y. Xiong, B. Chen, W. Sun, and L. Zhao. Language Feature Mining for Music Emotion Classification via Supervised Learning from Lyrics. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[10]. Y. Hu, X. Chen, and D. Yang. Lyric-based Song Emotion Detection with Affective Lexicon and Fuzzy Clustering Method. In ISMIR, 2009.

[11]. C. Laurier, J. Grivolla, and P. Herrera. "Multimodal Music Mood Classification Using Audio and Lyrics. 2008 Seventh International Conference on Machine Learning and Applications, 2008.

[12]. F. H. Rachman, R. Sarno, and C. Fatichah, "Music Emotion Classification based on LyricsAudio using Corpus based Emotion," Int. Journal of Electrical and Computer Engineering (IJECE), 2018.

[13]. R. Malheiro, R. Panda, P. Gomes, and R. Paiva. Music emotion recognition from lyrics: A comparative study. In 6th International Workshop on Machine Learning and Music, 2013.

[14]. Shlok Gilda, Husain Zafar, Chintan Soni, and Kshitija Waghurdekar. Smart music player integrating facial emotion recognition and music mood recommendation. In 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE, 2017.

[15]. I. Bakker, T. van der Voordt, P. Vink, and J. de Boon. Pleasure, arousal, dominance: Mahrabian and russell revisited. Current Psychology, 2014.

[16]. Emotion based Music Recommendation System (https://github.com/Udhay-Brahmi/Emotion-based-music-recommendation-system).

[17]. Retinaface (https://github.com/serengil/retinaface).

[18]. FER (https://github.com/justinshenk/fer).

[19]. Realtime Emotion Detection (https://github.com/neha01/Realtime-Emotion-Detection).

[20]. Emotion Detection (https://github.com/dhruvpandey662/Emotion-detection).

[21]. Keras Optimizers (https://keras.io/api/optimizers/).

[22]. Keras Metrics (https://keras.io/api/metrics/).

[23]. Keras Losses (https://keras.io/api/losses/).

[24]. Future Computers will be Radically Different (https://www.youtube.com/watch?v=GVsUOuSjvcg).