ABSTRACT

It is widely acknowledged that music has a significant emotional impact on individuals. The mood of an individual may often be influenced by their facial expressions. The system seeks to create a prototype for a music recommendation system integrated with the facial expression of a user. The division of music into happy and sad moods is a universal emotion, and thereby they are considered the two primary emotional states that people usually experience.

The user's mood will be detected by this system, and the playlist will be automatically created based on the user's facial expression and the choice of music made in accordance with that mood. In essence, a user's face is snapped using a webcam or other camera, and the input is retrieved as an image. The suggested method provides a convolutional neural network-based method for music recommendation that takes into account a user's facial expressions.

This system extracts tracks using the Spotify Web Application Programming Interface, and these tracks are clustered into two genres: cheerful and sombre. The application will detect the face of the user using the Haar cascade classifier, and the emotions are classified by the Fernet mode. According to the mood, it will select the song from the genre that relates to the emotion. Finally, the system presents songs that are in high correlation with the mood of the user. Large playlists that reflect different moods might present difficulties that can be solved by this approach. The application attaches itself to the feelings of individuals, providing users with a personalized experience and recommending songs based on the user's emotions or feelings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

A sturdy and general form of communication, music may bring people together and generate a variety of emotions and sensations. It is possible for music to affect someone's emotions. Music has a deep connection to emotions because it may incite intense emotions and moods in listeners. It is a given fact that there is a strong correlation between music and emotion. The motivation for a mood-based music recommendation system using facial expressions lies in the potential benefits it could bring to users. By using facial expression recognition technology to recommend music based on the current emotional state of a user, this system could enhance the music listening experience for users in several ways such as leading a user to a deeper emotional connection with music, improved emotional management, and a better understanding of one's own emotions. Thus, a mood-based music recommendation system using facial expressions has the potential to enhance the music listening experience for users by providing personalized music recommendations that synchronize with their current emotional status.

1.2 Existing Systems and Solutions

Recommendation systems for music have been around for a while. A music recommendation system could be stated as a software program or algorithm that offers personalized suggestions or simply recommendations to users about music that they may like. These systems are designed to help users find new music that they may relish. There are several types of music recommendation systems, including collaborative filtering systems, content-based systems, and hybrid systems that combine both approaches to carry out a recommendation process. Existing systems consist of a classification of either only emotions or only genres. But there is no system that implements or integrates both features. Based on the current emotional state and behavior of a user, existing systems possess a lesser degree of accuracy in generating recommendations. Some existing systems tend to employ the use of human speech or sometimes even the use of additional hardware for the generation of an

automated playlist. This is not efficient, as the users' work of first searching and creating a specific playlist is considered. There also arises the problem of a cold start in the music recommendation system, as there isn't any previous data about the user. Another thing to notice is that it also alleviates the problem of data sparsity in some recommendation systems to some extent. Thus, the existing system comes in handy with some uncertainty. Overall, music recommendation systems are a powerful tool for music streaming platforms and other music-related enterprises to optimize user engagement and retention by providing personalized and relevant recommendations to their users.

1.3 Proposed System

Music has the unique potential to elevate one's mood. Facial expressions are a genuine way for us to convey those emotions. The face is the key for comprehending human emotions, and facial expressions serve a crucial role in detecting a person's mood. Music genre can be meticulously associated with facial emotions given the fact that a fast-paced, upbeat song may trigger feelings of happiness and excitement, while a slow, mournful ballad may elicit feelings of sadness or melancholy. Facial emotion recognition technology can be used to pinpoint the emotional state of an individual, and then recommendations can be rendered based on that emotional state. However, by using facial emotion recognition technology, a music recommendation system can attempt to establish more accurate and personalized genre recommendations based on the individual's current emotional state. Emotions have a big influence on how one listens to music. If a user acquires a recommendation based on their opinions, it will also boost their listing experience on that particular platform. Basically, the proposed system aims to develop a prototype for a music recommendation system incorporating facial emotion recognition technology. Even though there may be many different emotions and moods that can be conveyed through music, the fact that the allocation of music into happy and sad categories is universal. Happiness is generally tied to positive feelings such as joy, contentment, and excitement, while sadness is allied with negative feelings such as grief, loss, and disappointment. Fast and upbeat music with major chords and optimistic lyrics can provoke a happy or joyful mood, while slow and mournful music with minor chords and sad lyrics can stimulate a sad or melancholy mood. Hence, happiness and

sadness are considered the two main emotional states that people experience. The goal of this research is to use facial emotion detection techniques to generate inputs for a music recommendation system, thereby enhancing the accuracy of the subsequent music recommendation system. A mood-based music recommendation system can be beneficial in several ways, such that it can present a more personalized and enabling experience for users while also benefiting firms through boosted promotional activities in markets. It is expected that the project would be the development of an application from the input, which is basically a face that is extracted from a webcam, that would detect and classify emotions for the given input face image and finally recommend songs based on the person's emotion. As a corollary, proposing a data fusion-based emotion recognition method for a music recommendation system.

1.4 Product Development Timeline

Project duration December 2022 to April 2023.

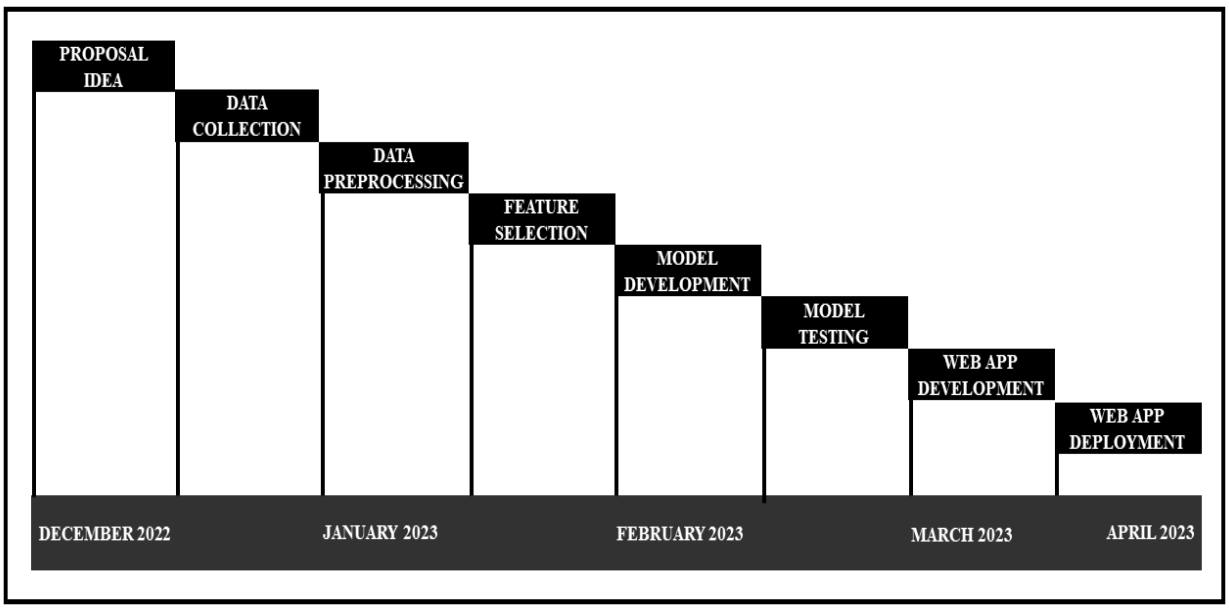


Figure: 1.1 Product Development Timeline

Chapter 2

LITERATURE REVIEW

2.1 Related Work 1: Traditional Music Recommender Algorithms

The music industry has spent years trying to solve the issue of providing personalized song recommendations. The two most common algorithms for making recommendations are content-based filtering and collaborative filtering. One approach proposes a music recommendation system using traditional algorithms such as the content-based recommendation algorithm and the collaborative filtering recommendation algorithm. The approach to a music recommendation system employs an analogous study between content-based and collaborative filtering algorithms. It extracts data from the Spotify web API and builds content-based algorithms by using a similarity inspection between music attributes and the cluster centroid of the K-means clustering algorithm. With the collaborative technique, the user rating for a particular song is taken into account, and user-user similarity is identified using modified cosine similarity. It uses the Movielens dataset to utilize the ratings of the users. Finally, the system recommends songs with the highest predicted ratings. It also tries to reduce the cold start problem by combining both algorithms. However, system takes much more time to generate the predictions [1].

When new items are introduced, collaborative filtering does not handle sparse data problems and suffers from issues like cold start problems when the user is new to the platform. Some systems intend to propose a hybrid technique for a music recommendation system. The hybrid model integrates both content and collaborative filtering techniques. When new items are introduced, collaborative filtering does not deal with sparse data problems and suffers from concerns like cold start problems when the user is fresh to the platform. The system tries to solve and overcome situations like cold-start problems. The approach makes use of user data that is crawled from web sources, and the data is split into training and testing in the ratio of 7:3. The approach shows an accuracy of about only 35% which indicates that the performance of the model is very poor [2].

Everyone listens to music in different ways, considering the track of their trends and offering the next song to them based on what they previously assessed. Another approach that studies are to conduct algorithms such as the popularity algorithm and the collaborative filtering algorithm for recommending songs. One thing to note is that the popularity algorithm seeks to recommend popular songs. The study uses the past historical data of the user to generate input for the algorithms. It evaluates the user's past ratings, then finds the item's similarities and recommends songs. The popularity model comes with a precision of 0.87 and a recall of 0.83. However many new songs the user might prefer, and if that song is not as popular as it is, this kind of system may not recommend that song which defaults for identifying user preferences [3].

Overall, traditional music recommendation systems rely on a combination of techniques and algorithms to provide personalized music recommendations to users, and the effectiveness of these systems depends on the quality and quantity of available data as well as the accuracy of the algorithms used.

2.2 Related Work 2: ML & DL based Music Recommender Algorithms

Machine learning is increasingly being used in music recommendation systems to personalize the listening experience for users. Some systems suggest the machine learning algorithms such as KNN for rendering a playlist. It incorporates content or collaborative methods with machine learning algorithms to enhance the systems. Based on collaborative filtering, KNN is considered one of the finest neighborhood algorithms, as the use of KNN seeks to group the nearest neighbours based on their similarities. Suggesting the KNN-Improved algorithm, which is built on the KNN algorithm and uses the baseline method's mean values in addition to increasing the rating's standard deviation. The model of the approach which is the KNN-Improved has an RMSE of 0.870. However, by replacing the original user rating with that of mean defaults to the quality of recommendations [4].

Another research investigates the combination of logistic regression and boosting techniques for the purpose of music song recommendation. The research uses a hybrid LX recommendation system that combines logistic regression and extreme gradient boosting which is the XGBoost. To assess the performance of the suggested

LX model, a series of experiments on a real-world music dataset had been implemented. The results reveal that the LX model's AUC, which is 0.8087. The approach reveals that the model overfits and as a result, the proposed model is not generalized [5].

Deep learning is a subset of machine learning that involves the use of neural networks and has also been increasingly used in music recommendation systems to improve the accuracy of recommendations and personalize the listening experience for users. Novel technology is considered for incorporation into the deep neural network as a classification model with traditional algorithms. The approach takes a hybrid content of input with content-based and collaborative filtering. It uses the user's historical data using the Gracenote API and measures the pairwise similarity between each song enjoyed by the user before recommending the song, and the precision score turns out to be 88%. However the quality of the recommendation is poor as it is built only using a small amount of dataset [6].

One popular deep-learning technique used in music recommendation systems is the use of convolutional neural networks for collaborative filtering. The research utilizes the deep-learning technique for music recommendation systems with collaborative filtering. CNN uses the music data crawled from music websites and outputs classified music. The collaborative filtering algorithm uses the data from CNN output and the data from the log file to recommend music to the user. The model uses a confidence score for evaluation, which is 90% only for certain genres and not all of them. The approach results well only for certain genres and not all of them [7].

One approach to the music recommendation system employs methods like audio embedding. Audio embedding tries to learn about musical representations using a Siamese network. Siamese networks are commonly used for tasks involving similarity or distance measurement between two inputs. The model's performance is based on tasks including genre classification and music recommendations, and the model has a precision score of 0.773 and an AUC score of 0.849. This is somewhat better than other proposed models and the use of Siamese networks yields better accuracy. However the generalization ability of the model needs improvement [8].

Overall, machine learning and deep learning has the potential to significantly improve the accuracy and personalization of music recommendation systems, leading to a better listening experience for users.

2.3 Related Work 3: Emotion based Music Recommender Algorithms

Emotion is essential for recommendation algorithms, as music is often associated with emotions and can have a significant impact on the listener's mood and well-being. In some cases, the user's social network information can be useful for recommendation algorithms, as it can provide additional context and insights into the user's preferences, interests, and activity on social media platforms. One way to extract the emotions of a user is by analyzing their social media platforms. It uses a neural network with a labelled dataset to extract various kinds of emotions, and sentiment values from the lyrics of a song are also extracted to perform the music recommendation. After evaluating the model, the overall precision score is 0.903, and the F1 score is 0.901. However, using a user's social sites may cause privacy and security concerns [9].

Another application proposes an interface for the user to pick a color to recommend the song. Initially, the approach tends to develop an application for music recommendation by the user itself. It proposes an interface for the user to pick a color to recommend the song. The approach creates a music library dealing with the association of color with emotion and music with emotion. The approach utilizes the HSV color model for the music library data with hue, saturation, and value, where hue represents musical instruments, saturation refers to tempo, and value is key or pitch. The second stage includes two types of graphical user interfaces (GUI) for selecting the color. And the final stage concludes by collecting the data from conducting an experiment. In the trials, it is found that the overall accuracy rate turns out to be 51.11%, with hue at 50%, saturation at 36.67%, and value at 66.67%. But one thing to notice in the proposed approach is that the system does not define the backend process of an application [10].

Overall, incorporating emotion into recommendation algorithms can help to provide a more personalized and engaging music experience for users and can enhance the overall effectiveness of music recommendation systems.

Chapter 3

DATA COLLECTION

3.1 Description of the Data

The effectiveness of a deep learning model is greatly influenced by the quantity and quality of the data. A well-curated dataset is indispensable for the development of a facial emotion-based music recommendation system, as the system hinges on understanding the emotions expressed by the facial expressions of an individual in real-time for training the model. A facial emotion-based music recommendation system uses the input data, which is basically streaming video frames extracted with the help of a webcam. The input data is an image, which consists of the face of the user. Basically, to train the neural network model for detecting and classifying the image, given the face as input, and detecting the emotion from that facial expression, the proposed approach utilizes the FER-2013 dataset.

The FER dataset is the most well-known and trendy dataset for facial emotion recognition. It packs a total of 35,887 images of faces, and the visuals are annotated with class labels such as anger, disgust, fear, happiness, sadness, surprise, or neutral. The dataset basically comprises two main collections, which are the training and testing directories. The training directory encompasses 28,709 examples of images, whereas the testing directory entails 3,589 instances of images. These directories were further subdivided into subfolders, which relate to different categories of facial emotions and contain approximately 35,000 image files kept in the jpg format. These image files are basically face images that have a dimension of about 48x48 pixels and are gray-scale in nature.

Usually, the recommender system necessitates a huge amount of data. Since the recommender system suggests a playlist for the finished output, there is a demand to create a dataset for recommending the songs. This can be solved with the help of the Spotify developer website, as the Spotify Web Application Programming Interface provides access to a wealth of information about songs accessible on the Spotify platform. There are a lot of features that can be extracted, and the following is a breakdown of the data that can be obtained or extracted through the Spotify Web API,

such as track information and metadata, artist and album information, audio features and analysis, playlist information, user information and user listening history, and external URLs.

Table: 3.1 Feature Description of Song Data

Features	**Description**
Track ID	Unique identifier of each track assigned on Spotify.
Track Title and Track Name	Refers to the name of a particular track on Spotify.
Artist Name	The name of the artist who made the track.
Album Name	The name of the album on that the track is in that album.
Artist Genre	Refers to the genre or various genres that associated with the artist.
Popularity	A metric that reflects the popularity of a song on Spotify.
Length	The duration of the song in milliseconds.
Danceability	A metric that measures the suitability of a song for dancing.
Loudness	A metric that measures the overall loudness of a song in decibels.
Speechiness	A metric that measures the number of spoken words in a song.
Acousticness	A measure that measures the presence of acoustic elements or instrumentation in a song.
Liveness	A metric that measures the presence of a live audience or performance in a song.

Instrumentalness	A metric that measures the presence of instrumental content in a song.
Energy	A metric that measures the perceived energy of a song.
Valence	A metric that measures the emotional positivity or negativity of a song.
Mode	A metric that indicates the modality (major or minor) of a song.
Key	A metric that indicates the set of notes and chords in a song.
Tempo	A metric that measures the tempo (beats per minute) of a song.
Time Signature	A metric that indicates the time signature of a song.

Thus, the song database is created in such a way that the data is directly extracted from the Spotify API using developer credentials, with some help from inputting some playlist ids, and these songs are stored in a database for later use.

3.2 Source and Methods of Collecting Data

The FER stands for Facial Emotion Recognition and this dataset is a publicly available dataset that is typically used for training and testing facial emotion recognition models. It was released in the year 2013 and has since become a well-known benchmark dataset for facial expression recognition studies. It was developed and designed by researchers from the Computer Vision Centre (CVC) at the Autonomous University of Barcelona. The dataset was developed in a way by collecting images from the web and then manually annotating each image with the appropriate emotion. The images were then transformed to eliminate any

background noise and normalize the illumination and contrast. This FER dataset is real-time data collected from Google and contributed to Kaggle. The image files consist of faces that have been automatically registered and every image occupies about the volume of space. This dataset also comes with the fact that it may precisely reflect real-world distinctions in facial expressions. Additionally, the dataset may be indicative of all ethnicities and ages, which may represent the generalizability of the dataset. Below implements a sample code for importing the data into the working environment.

```
train = "data/fer_data/full_data/train"
test = "data/fer_data/full_data/test"
```

Figure: 3.1 Code Snippet for Fer Data Source

The recommender system suggests the recommended songs. These song files are extracted, and the songs are stored in CSV files as metadata. The metadata is created in such a way that the data is right away extracted from the Spotify API using developer credentials. Below is a snippet of code for extracting the data from Spotify Web API.

```
def get_data(user, playlist_id):

    track_ids = get_track_ids(user, playlist_id)

    track_list = []

    for i in range(len(track_ids)):
        track_data = get_track_features(track_ids[i])
        track_list.append(track_data)

        df = pd.DataFrame(track_list, columns = ['track_id','track_name',
                                                 'album_name','artist_name','genres','popularity',
                                                 'danceability','loudness','speechiness',
                                                 'acousticness','liveness','instrumentalness',
                                                 'energy','valence','mode',
                                                 'key','tempo','time_signature','length'
                                                ])

    return df
```

Figure: 3.2 Code Snippet for Song Data Source

The data accessible through the Spotify Web API is normally up-to-date. The basic and general steps to extract data from the Spotify Web API for the proposed system begin with registering for a Spotify developer account and creating a new Spotify app, which will give access to the Spotify Web API and the necessary API keys to form requests. Extract the desired or required data, and in this case, the desired playlist IDs are given to extract songs and storing it in a CSV file.

Chapter 4

PREPROCESSING AND FEATURE SELECTION

4.1 Overview of Preprocessing Methods

The dataset gathered might not be consistent with the desired format. Preprocessing is a prerequisite step as it is used to elevate the quality of the data for model building. Some possible measures for preprocessing the data pertain to data cleaning, data transformation, and data reduction. The data acquired for study or model-building tasks is likely to contain errors, missing values, or irrelevant information. Data cleaning assists in comprehending and tidying such data by taking off duplicate entries, fixing errors, imputing missing values, and removing irrelevant information. Preprocessing also entails transforming data to make it appropriate as an input for model training. This involves strategies such as feature extraction, normalization, and scaling. Overall, preprocessing is crucial to ensure that the data is in an ideal format for model-building or analysis. It helps to optimize the quality of the data and aids in boosting the accuracy and efficiency of the analysis or model.

4.2 Overview of Feature Selection Methods

The selection of features from a provided set of data is a necessary stage in the successful creation of a model. Feature selection is a procedure that allows for diminishing the size of the dataset while conserving important and required information. This drastically reduces the dimensions of the data and makes it a compatible size for a model to learn. After carefully noticing the data, the indispensable features are selected in such a way that the features should not lead the model in a biased direction. Selecting the most significant features from a dataset may aid the model in preventing bias and hinder the model from overfitting. Thus, feature selection lets one create a model that is more generalized. Overall, the key for developing a productive mood-based music recommendation system is to cautiously select the most needed and required data, extract relevant features, and select the most informative features.

4.3 Preprocessing and Feature Selection Steps

The preprocessing and feature selection of the data for the mood-based music recommendation system comes in three different modules, such as FER data preprocessing, song data preprocessing, and user data preprocessing.

FER Module: FER data preprocessing and feature selection methods involve FER data creation and converting the data to a suitable format for model building.

Table: 4.1 Fer Data Preprocessing Steps Description

FER Data Preprocessing	**Description**
Data Reduction	Reducing to interested emotions
Image Conversion	Converting image to raw pixel values
Data Upsampling	Upsampling the minority class
Data Saving	Saving the data as CSV file
Data Reshaping	Reshaping data to one dimension
Data Preparation	Extracting input and target variables
Data Splits	Splitting the data for training and testing
Data Normalization	Scaling down the values of data

- Data Reduction: Given the fact that happy and sad are two moods that are universal in nature, the proposed approach is only interested in those two classes. The image files from those two classes are extracted and saved in a new directory. And from a keen observation of the FER dataset, it is found that the majority of classes belong to Happy and Sad.

```
interested_emotions = ['happy', 'sad']

def new_data(directory, folders, name):
    for folder in folders:
        folder_path = directory + "/" + folder
        source_dir = directory + "/" + folder
        destination_dir = "data/fer_data/new_data/" + name + "/" + folder
        shutil.copytree(source_dir, destination_dir, dirs_exist_ok=True)
        print("Data created at", destination_dir)

new_data(train, interested_emotions, 'new_train')
new_data(test, interested_emotions, 'new_test')
```

Figure: 4.1 Code Snippet for Fer Data Reduction

- Image Conversion: As the image data is a little hard to compute, converting images to raw pixel values with the use of the OpenCV Python library and storing the data in a data frame with pixel data in the pixels column and relevant emotions in the emotions column.

```
def data(directory):
    labels = os.listdir(directory)
    emotion_data = []
    pixel_data = []
    for emotion_label in labels:
        path = os.path.join(directory, emotion_label)
        class_label = labels.index(emotion_label)
        for image in os.listdir(path):
            image_array = cv2.imread(os.path.join(path,image), cv2.IMREAD_GRAYSCALE)
            image_sized = cv2.resize(image_array,(48, 48))
            image_pixel = ' '.join(map(str, image_sized.flatten().tolist()))
            emotion_data.append(class_label)
            pixel_data.append(image_pixel)
    data = {'emotion': emotion_data, 'pixels': pixel_data}
    df = pd.DataFrame(data)
    return df
```

Figure: 4.2 Code Snippet for Fer Data Image Conversion

- Data Upsampling: From a close look at the data, it seems that the data is imbalanced, with emotion class 0 containing 8989 samples and emotion class 1 containing 607 samples. So up-sampling the data with the majority class using the resample method in the sklearn.utils library to balance the samples.

```
df_maj = df[(df.emotion == 0)]
df_min = df[(df.emotion == 1)]
df_min_upsampled = resample(df_min, replace=True, n_samples=len(df_maj), random_state=42)
df_new = pd.concat([df_min_upsampled, df_maj])
df_new.index = pd.RangeIndex(start=0, stop=len(df_new.index), step=1)
```

Figure: 4.3 Code Snippet for Fer Data Upsampling

- Data Saving: Saving the up-sampled data into a CSV file for further processing of the data for the model training.

```
df_new.to_csv('data/fer_data/meta_data/fer.csv', index=False)
```

Figure: 4.4 Code Snippet for Fer Data Saving

- Data Reshaping: Reshaping the image pixel data into a one-dimensional array using the NumPy library. So every image is stored as a one-dimensional array with the shape of (1, 48, 48, 1) and having the image properties such as a height and width of 48 units and a depth of 1, indicating that the image is grayscale.

```
img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48, 1).astype('float32'))
img_array = np.stack(img_array, axis=0)
```

Figure: 4.5 Code Snippet for Fer Data Reshaping

- Data preparation: Extracting X as input variables and y as target variables for the model. The input variables are the image pixel data as an array, and the target variables are the image class labels.

```
img_array.shape
img_labels = df.emotion.values
img_labels.shape
```

Figure: 4.6 Code Snippet for Fer Data Preparation

- Data Splits: The dataset is split into the train, test, and validation data sets, with 81.0% of the train set, 10.0% of the test set, and 9.0% of the validation set.

```
X_train, X_test, y_train, y_test = train_test_split(img_array,
                                                    img_labels,
                                                    test_size=0.1,
                                                    shuffle=True,
                                                    random_state=42)

X_train, X_val, y_train, y_val = train_test_split(X_train,
                                                  y_train,
                                                  test_size=0.1,
                                                  random_state=42)
```

Figure: 4.7 Code Snippet for Fer Data Splits

- Data Normalization: Finally, scale down all of the input pixel data by using the normalization technique of dividing the input pixels by 255 to make the values range between 0 and 1.

```
X_train = X_train / 255.

X_val = X_val / 255.

X_test = X_test / 255.
```

Figure: 4.8 Code Snippet for Fer Data Normalization

Song Module: Song data preprocessing and feature selection methods involve song data extraction, song genre identification, and storing the songs in a database.

Table: 4.2 Song Data Preprocessing Steps Description

Song Data Preprocessing	**Description**
Data Extraction	Extracting data using web API
Drop Duplicates	Dropping the duplicate data
Data Normalization	Scaling down the numeric features
Genre Clustering	Identifying the genre of a track
Data Storing	Storing the data into database

- Data Extraction: The primary step is to extract track or song data using the Spotify Web API. It is done by using Spotify developer credentials and with the help of some playlist links. Thus extracting features such as track_id, track_name, album_name, artist_name, genres, popularity, danceability, loudness, speechiness, acousticness, liveness, instrumentalness, energy, valence, mode, key, tempo, time signature, and length.

```
client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

df1 = get_data('spotify', '37i9dQZF1DWZKuerrwoAGz')
df2 = get_data('spotify', '4WloBZWLuV80F07SCPxs09')
df3 = get_data('spotify', '3Kz5KBE3Ksupz9odBGwze6')
frames = [df1, df2, df3]
df = pd.concat(frames)
df.index = pd.RangeIndex(start=0, stop=len(df.index), step=1)
```

Figure: 4.9 Code Snippet for Song Data Extraction

- Dropping duplicates: The duplicate values of the same tracks are then removed from the dataset using drop duplicates method after it has been analyzed.

```
df = df.drop_duplicates(subset=['track_id'])

df = df.drop_duplicates(subset=['track_name'])

print("Are all samples unique: ", len(pd.unique(df.track_name))==len(df))
```

Figure: 4.10 Code Snippet for Song Data Duplicates Dropping

- Data Normalization: All the numeric features are normalized using the min-max scaler.

```
minmax = MinMaxScaler()

scaled_features = minmax.fit_transform(features)
```

Figure: 4.11 Code Snippet for Song Data Normalization

- Genre Clustering: Selecting the most required feature for genre identification of each track after a thorough analysis of the data. Then with the help of the K-means clustering algorithm, the dataset has been clustered into two main clusters, and the genre for each track has been identified using the cluster as either cheerful or sombre.

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
df['cluster'] = kmeans.predict(X)

df.loc[df['cluster'] == 0, 'genres'] = 'cheerful'
df.loc[df['cluster'] == 1, 'genres'] = 'sombre'
```

Figure: 4.12 Code Snippet for Song Data Genre Clustering

- Data Storing: Only the required features are selected to store the data in a database for generating recommendations. Features selected such as track_id, track_name, artist_name, genres, and popularity. Lastly, the required features are stored in the database using sqlite3, and a table named songs is created to store the data.

```
conn = sqlite3.connect('songs.sqlite')
table_name = 'songs'
query = f'Create table if not Exists {table_name} (track_id TEXT, track_name TEXT, artist_name TEXT, genres TEXT, popularity REAL)'
conn.execute(query)
song_data.to_sql(table_name, conn, if_exists='replace', index=False)
```

Figure: 4.13 Code Snippet for Song Data Storing in Database

User Module: User data preprocessing and feature selection methods involve data collection, data extraction, and preprocessing of the extracted data for model prediction.

Table: 4.3 User Data Preprocessing Steps Description

User Data Preprocessing	**Description**
Data Collection	Collecting data from user
Data Conversion	Converting the image to grayscale
Feature Selection	Selecting the region of interest
Data Reshaping	Reshaping the data for the model

- Data Collection: Collecting data on the facial expressions of a user while they listen to music. This could involve using a webcam or a camera to capture the face of a user.

```
cap = cv2.VideoCapture(0)
display_handle = display(None, display_id=True)

while True:
    print('.'*5)
    print('Camera is open!')
    sus, frame = cap.read()
```

Figure: 4.14 Code Snippet for User Data Collection

- Data Conversion: Converting the captured frame to grayscale.

```
print('.'*5)
print('Converting frame to grayscale.')
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Figure: 4.15 Code Snippet for User Data Conversion

- Feature selection: Selecting the region of interest from the captured and detected face of the user.

```
print('.'*5)
print('Selecting the region of interest.')
for (x, y, w, h) in faces:
    ret = cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), 2)
    sus, frame_ret = cv2.imencode('.jpeg', ret)
    display_handle.update(Image(data=frame_ret))
```

Figure: 4.16 Code Snippet for User Data Feature Selection

- Data Reshaping: Reshaping the selected image with a dimension compatible with the model to give input to the model to predict the emotion class.

```
print('.'*5)
print('Reshaping the image.')
roi = np.reshape((cv2.resize(roi, (48,48))) / 255.0, (1,48,48,1))
```

Figure: 4.17 Code Snippet for User Data Reshaping

Overall, the key to developing an effective mood-based music recommendation system is to carefully preprocess the data, extract relevant features, select the most informative features, and use robust techniques for preprocessing and feature selection.

Chapter 5

MODEL DEVELOPMENT

5.1 Model Architecture

The main ideology of the proposed system revolves around the concepts of facial emotion recognition and music recommendation. The primary objective of the architecture aims to identify the most suitable mood expressed by a user and recommend songs of a relevant genre based on the emotion that the deep learning model classifies. The proposed setup of the system employs real-time facial emotion recognition and music recommendation. The architecture basically consists of three different modules that involve the FER module, the song module, and the user module. The FER module comprises a facial emotion recognition task. The song module includes song extraction, genre identification, and storing the songs in a database. The user module is an integration of both the FER and song modules, which recommend songs of relevant genres based on the emotions classified by the model. The whole systematic design of the proposed architecture by the mood-based music recommendation system is given below.

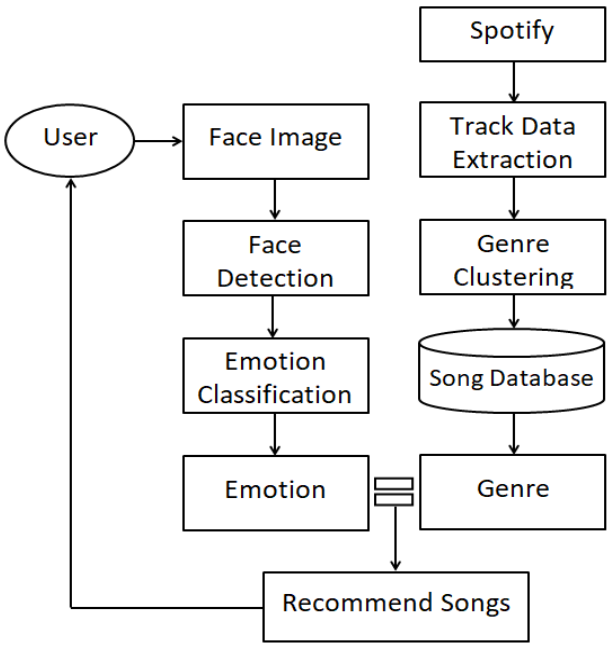


Figure: 5.1 Proposed Model Workflow

In this system, live video is streamed, and a webcam is adopted to extract a face as an input image. The facial expressions are detected using the Haar Cascade Classifier, which yields Haar Cascade features. The Haar Cascade Classifier is one of the oldest and most highly regarded object detection algorithms used to identify faces in images or real-time videos. Then, the region of interest, which is basically the face image from the cascade classifier, is passed into the facial emotion model for emotion classification. The facial emotion model extracts the features from the given input image using convolutional layers and finally classifies the output as class labels using the dense layers at the end of the model's architecture. The output labels from the facial emotion model are binary, and the class label would be either happy or sad. The output generated from the model is compared with the genre of that particular song, which is stored in the database. If the facial emotion model rendered the class happy, the songs in the genre cheerful are recommended, and if it is sad, the songs in the genre somber are recommended. Finally, the top seven most popular songs within the genre are recommended. Consequently, for a particular individual's mood, the algorithm offers a playlist with the seven most popular songs.

5.2 Algorithms Applied

The proposed system suggests a facial emotion classification task. This model draws on an input image of a user's face and classifies the output as either happy or sad. For the binary classification tasks of facial emotions, the popular benchmark models such as Vision Transformer and Resnet50 are used, and later a custom model called Fernet is created and implemented. The details of the algorithms used are depicted.

ViT Model: The Vision Transformer (ViT) is a deep learning model that adheres to the Transformer architecture, originally put forward for tasks such as image classification. The ViT model operates by breaking down the input image into a sequence of fixed-size patches, which are then flattened into a 1D sequence and passed through a series of Transformer encoder layers. The result of the final encoder layer is then fed into a multi-layer perceptron (MLP) classifier to predict the class of the input image. The key innovation of the ViT model is the use of self-attention mechanisms in the transformer encoder layers to capture global context

information from the image. The vision transformer architecture is given in the following image below.

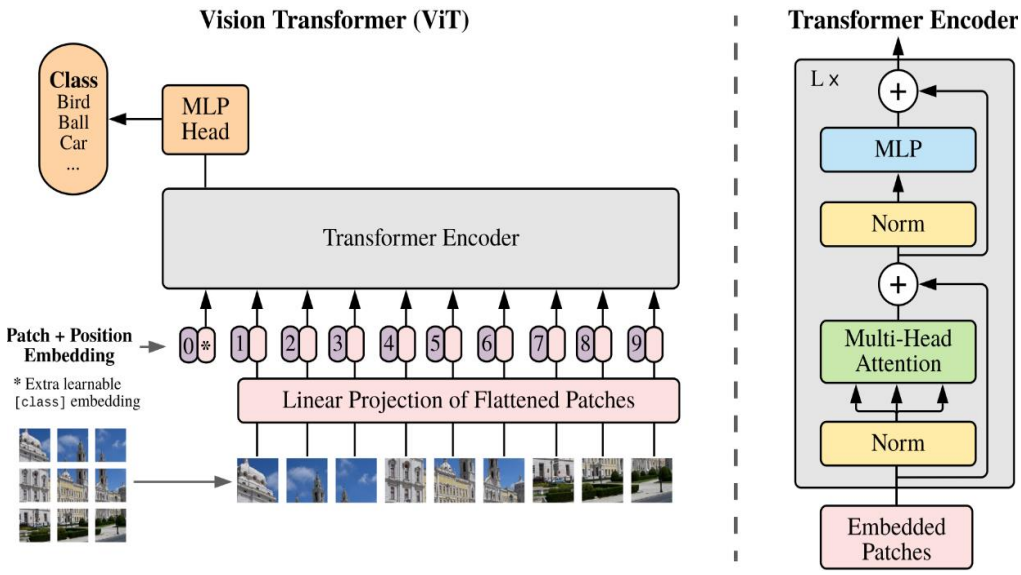


Figure: 5.2 Vision Transformer Architecture

ResNet-50 Model: It is a convolutional neural network architecture that is a part of the ResNet clan of neural networks. This model is designed to address the problem of vanishing gradients that can occur in very deep neural networks. ResNet-50 consists of 50 layers, including a convolutional layer, followed by several blocks of residual layers, which contain multiple convolutional layers and shortcut connections that bypass some of the convolutional layers. The shortcut connections are used to propagate the gradient more efficiently through the network, thereby allowing for much deeper neural networks to be trained. ResNet-50 is a pre-trained model that uses the concept of transfer learning. It has been trained on large-scale image recognition tasks such as those in the ImageNet dataset and has achieved state-of-the-art performance on these tasks. The resnet50 architecture is given in the following image below.

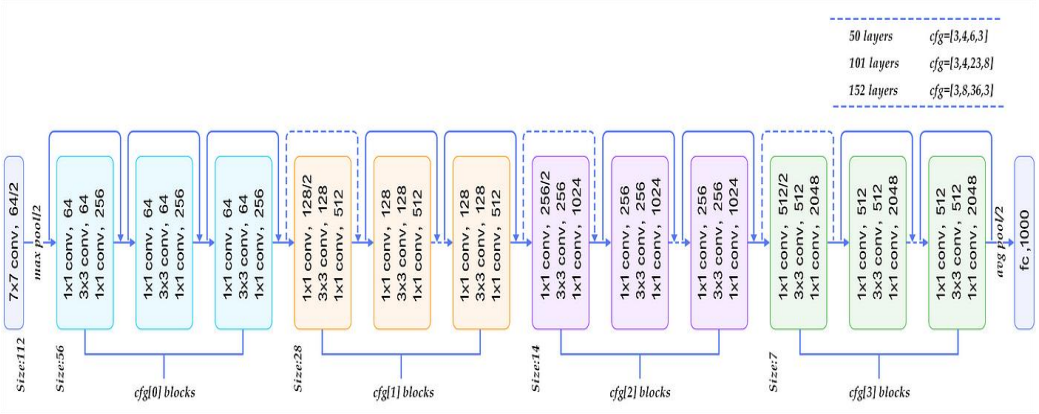


Figure: 5.3 ResNet50 Architecture

Fernet Model: This model depicts an architecture consisting of a series of four convolution blocks, a flattened layer, two fully connected dense layers, and an output dense layer for classification at the end. Each convolution block consists of two convolution layers with the same filters and kernel size and a batch normalization layer in between them, and the block ends with a series of a batch normalization layer, a max-pooling layer, and a dropout layer. Every convolution layer has the same padding around the inputs and utilizes the Relu activation function. Every max-pooling layer has a kernel size of about 2 x 2 dimensions. After the convolution blocks, the flattened layer is used to transform the tensors into single-dimension vectors to feed into the dense layers. Then for the classification task, the architecture utilizes three dense layers, and the last layer utilizes a sigmoid activation function for the prediction of binary output labels. The fernet architecture is given in the following image below.

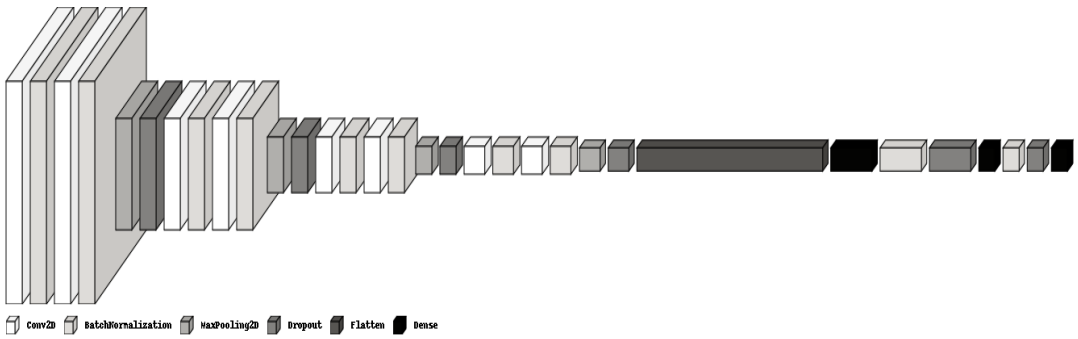


Figure: 5.4 Fernet Architecture.

5.3 Training Overview

Training corresponds to the process of optimizing the parameters of a model by making use of a dataset. The goal of training is to tweak the model parameters so that the model can accurately predict the right output for new, unseen inputs. The training process encompasses iteratively updating the model parameters using an optimization algorithm until the model converges on a set of optimal parameters that minimize the difference between the predicted output and the true output. As the proposed approach revolves around concepts such as facial emotion recognition, there is a need to train a model for such tasks as classifying emotions. The facial emotion recognition model utilizes the FER dataset created after the preprocessing techniques.

As the dataset is very large to train the model, and it is noticeable that the data is full of inputs such as images, which literally take more computational time, the approach utilizes the GPU provided by Google Colab for training deep learning models. Using a powerfully allocated GPU, Nvidia's Tesla T4, for the acceleration of the training of deep-learning models, provided with a driver version of 525.85.12 and a CUDA version of 12.0, and using a memory of 373/15360 MiB. The model learns using the trainable and non-trainable parameters. During training, the model is fed a batch of input data, and the output of the model is compared to the true output using a loss function. The loss function computes the difference between the predicted output and the true output, and the optimization algorithm adjusts the model parameters to minimize this difference. This process is repeated for multiple epochs, where each epoch involves training on the entire training dataset once. Once the training process is complete, it is saved using callbacks, which oversee the training of the model and save the best possible version using model checkpoints.

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Figure: 5.5 Code Snippet for training models using GPU.

Chapter 6

EXPERIMENTAL DESIGN AND EVALUATION

6.1 Experimental Design

A well-generalized and robust model is possible by conducting many experiments to make sure that it doesn't overfit or underfit. The experimental setup of the system uses a real-time facial emotion recognition model. Different experiments showcase the different models used for the emotion classification task. Each model requires its own preprocessing of the data, but almost all the models use the same dataset.

Experiment-1: Training ViT Model

Primarily cloning the required dependencies, such as the hugging face transformers dataset repository from GitHub, and importing the necessary libraries. The dataset is loaded into the working environment and the data is split, with 81% of the data for training, 10% of the data for testing, and 9% of the data for validation. Preparing FER data for the vision transformer in the hugging face dataset format and downsampling the data for RAM issues. Preprocessing the dataset with images in their original size to a dataset with pixel values computed by the vit feature extractor. Finally, the vision transformer model is built, and the model is trained for six epochs with 16 as the batch size. The model is evaluated, and the results are noted. Below image is a sample code snippet for ViT model training.

```
model = ViTForImageClassification()

trainer = Trainer(
    model = model,
    args = args,
    train_dataset = preprocessed_train_ds,
    eval_dataset = preprocessed_val_ds,
    compute_metrics = compute_metrics,
)

trainer.train()
```

Figure: 6.1 Code Snippet for training ViT model.

Experiment-2: Training ResNet50 Model

Initially importing the necessary libraries and loading the dataset into the working environment. Preprocessing the dataset involves formatting the data, such as reshaping the data and converting the images from grayscale to RGB format for the model. The data is split, with 81% of the data for training, 10% of the data for testing, and 9% of the data for validation. The ResNet50 model is built sequentially in such a way that the top layers are pre-trained upon imagenet, and the last layer consists of a dense layer with a sigmoid activation function for binary classification of emotion labels. The model has a total of 23,589,761 parameters, with trainable parameters of 23,536,641 and non-trainable parameters of 53,120. The model utilizes the Adam optimizer for convergence. The model is trained for ten epochs, the model is evaluated, and the results are noted. Below image is a sample code snippet for ResNet50 model training.

```
resnet50 = Sequential()

resnet50.add(ResNet50(input_shape=(48,48,3), include_top=False, weights='imagenet', pooling='max'))

resnet50.add(Dense(1, activation='sigmoid'))

resnet50.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

resnet50_history = resnet50.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10)
```

Figure: 6.2 Code Snippet for training ResNet50 model.

Experiment-3: Training Fernet Model

This experiment depicts the building and training of the baseline model, or simply the "fernet" model. It utilizes the basic CNN, which stands for convolutional neural network. CNN has a wide range of applications, including image classification and facial recognition tasks. They are widely used due to their ability to learn complex patterns and features from data. In a CNN, the input data is passed through several layers of convolutional and pooling operations. These layers are designed to extract features from the input data, which are then fed into a fully connected neural network for classification. The model is fed with the FER dataset, and necessary preprocessing steps are made. The data is split, with 81% of the data for training, 10% of the data for testing, and 9% of the data for validation. The

Fernet model is built using eight convolutional layers with the addition of some batch normalization layers, max-pooling layers, and dropout layers in between the convolutional layers, as depicted in the model's architecture, and a total of 2,440,929 parameters, including 2,437,729 trainable parameters and 3,200 non-trainable parameters, are used. The training process of the model is done with 64 as the batch size for a total of 30 epochs. Once the training process is complete, it is saved using callbacks, which oversee the training of the model and save the best possible version using model checkpoints. The trained model is evaluated, and the results are noted. Below image is a sample code snippet for Fernet model training.

```
checkpoint = ModelCheckpoint(filepath='fernet.h5',
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             moniter='val_accuracy')

early_stopping = EarlyStopping(monitor='val_accuracy',
                           min_delta=0.00005,
                           patience=11,
                           verbose=1,
                           restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.5,
                              patience=7,
                              verbose=1,
                              min_lr=1e-7)

callbacks = [checkpoint, early_stopping, reduce_lr]

fernet = model()
batch_size = 32
epochs = 50
steps_per_epoch = len(X_train) / batch_size
validation_steps = len(X_val) / batch_size
history = fernet.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     steps_per_epoch=steps_per_epoch,
                     epochs=epochs,
                     callbacks=callbacks,
                     validation_steps=validation_steps)
```

Figure: 6.3 Code Snippet for training Fernet model.

6.2 Experimental Results

Even for humans, capturing emotions has always been challenging. Performing the same task with better accuracy with the help of a deep learning model has always been the part where new innovations take place. Since the user is recommended a playlist of songs based on their facial emotions, which ultimately represent that individual's mood, the only way to measure how well the recommendations are made is to measure the performance of a model, which is the base one for making predictions about emotions and generating songs based on the emotion classified.

In a classification task, the performance of a model is often evaluated using a variety of metrics, including accuracy, precision, and recall. These metrics use parameters such as True Positives (TP), True Negatives (TN), False Positives (FP) and False negatives (FN) to measure how well the model is able to correctly identify different classes of inputs. TP represents the number of correctly predicted positive samples, TN represents the number of correctly predicted negative samples, FP represents the number of incorrectly predicted positive samples, and FN represents the number of incorrectly predicted negative samples. In summary, accuracy measures the overall correctness of the model's predictions, precision measures the proportion of positive predictions that were actually correct, and recall measures the proportion of actual positives that were correctly identified by the model. All three metrics are important for evaluating the performance of a classification model.

Experiment-1: Evaluating ViT Model

The performance of the vision transformer model for facial emotion classification tasks depends on factors such as the size and quality of the dataset used. Upon training the model, it is observed that the model performs quite well. It is observed that the loss of the model during training is 0.11 and 0.18 during testing. The performance of the vision transformer model is assessed using metrics such as accuracy, precision, and recall. The training accuracy is up to 95%, while the testing accuracy is 92%. The model scores a precision of about 0.92, and the recall tends to be 0.92. However, while the model performed fairly, comparatively, the accuracy rate is lower.

Experiment-2: Evaluating ResNet50 Model

Resnet50 is a pre-trained model, and the basis of its performance is the learnable parameters of the architecture. It is observed that for the classification task of identifying emotions based on facial expression, this model is not generalized as the interpretation of the result plots. The model has a train loss of 0.19, while the validation loss is 0.25 and the test loss is 0.24. The model yields about 92.99% during training, the validation accuracy seems to be 89.45%, and the testing accuracy is 90%. It is seen that the area under the ROC curve of the mode is 0.89. The model has a precision score of about 0.90, and the recall turns out to be 0.91. Even though the model performed fairly, the accuracy rate is lower comparatively. As a pre-trained model on ImageNet, the performance of this model is quite low.

Experiment-3: Evaluating Fernet Model

The Fernet model is evaluated, and the results are noted. It is observed that for the classification task of identifying emotions based on facial expression, this Fernet model is well generalized as the interpretation of the result plots. The model has a train loss of 0.04, while the validation loss is 0.17. The model yields about 98.70% during training, and the validation accuracy seems to be 94.38%. The result plots such as model accuracy and model loss by Fernet is given in the below image.

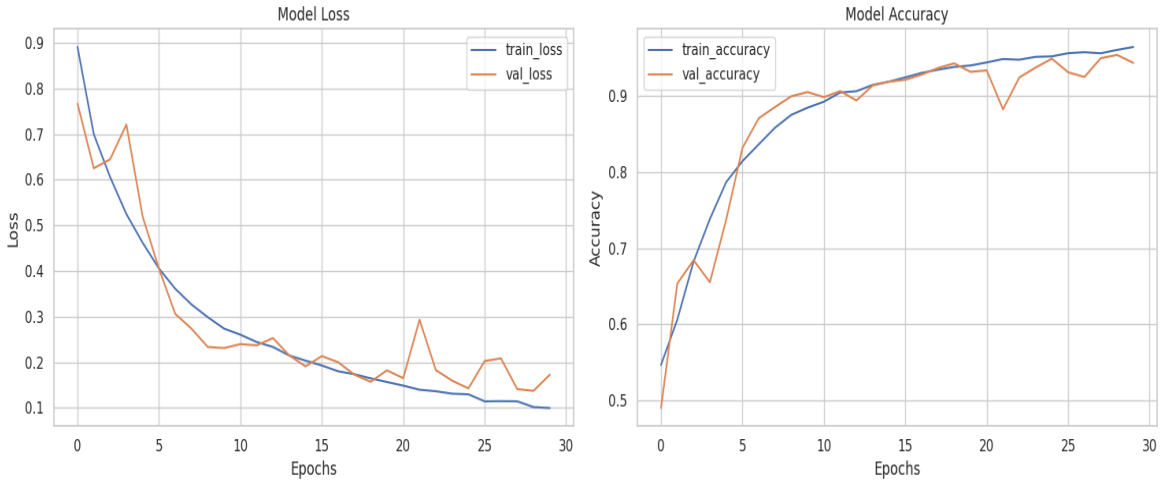


Figure: 6.4 Fernet Result Plots.

The Confusion Matrix by Fernet model is given in the below image and it depicts the true positives, true negatives, false positives and false negatives.

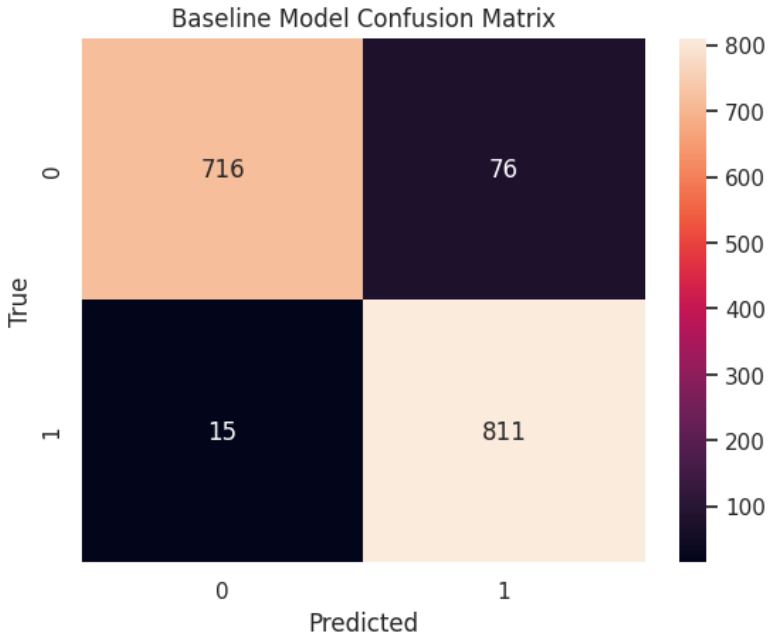


Figure: 6.5 Fernet Confusion Matrix.

It is observed that the area under the receiver operating characteristics curves is well provided by the Fernet model. It is seen that the area under the ROC curve of the mode is 0.94. The ROC curve of the model attains an area of about 0.94, and the curve is given in the figure below.

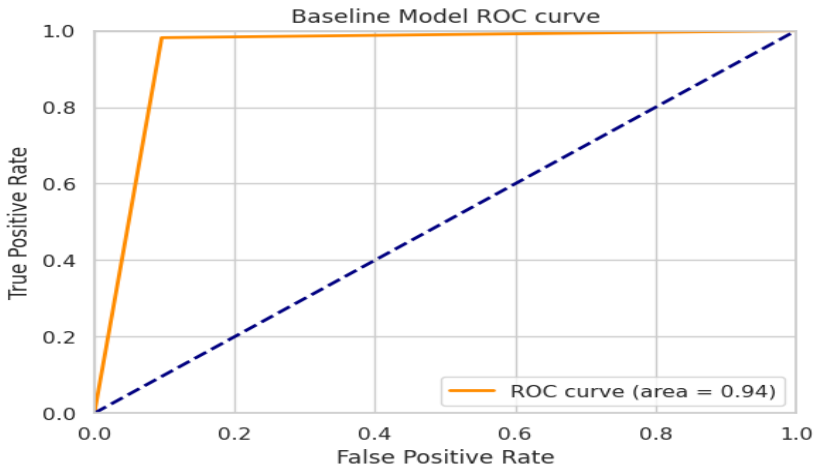


Figure: 6.6 Fernet ROC Curve.

The overall accuracy of the model is given as 94%. The overall results of the Fernet model are depicted in the table given below.

Table: 6.1 Fernet Results.

Metrics	Value
Accuracy	0.94
Precision	0.90
Recall	0.98
F1-Score	0.94

As the experiment for the facial emotion classification task is determined by training different models, the comaparative analysis is depicted in the following given table below. It is observed that the custom model Fernet of the proposed system outperforms both of the other benchmark models.

Table: 6.2 Comparative Analysis of Model Results

Metrics	Accuracy	Precision	Recall	F1-Score
ViT	0.92	0.92	0.92	0.92
ResNet50	0.90	0.90	0.91	0.90
Fernet	0.94	0.90	0.98	0.94

Chapter 7

MODEL OPTIMIZATION

7.1 Overview of Model Tuning and Best Parameters Selection

Model tuning is the process of adjusting the hyperparameters of a model to improve its performance. They determine the behaviour of the model during training and can significantly impact its performance. The process of model tuning involves selecting a range of values for each hyperparameter, training the model with different combinations of hyperparameters, and selecting the combination that yields the best performance. The process of model tuning and selecting the best parameters for a convolutional neural network (CNN) model is similar to that of any other machine learning model. However, there are some specific hyperparameters that are unique to CNN models that need to be tuned to achieve optimal performance. Some of the key hyperparameters that need to be tuned when building a CNN model are the initializer techniques, filters, layers, early stopping, learning rate, etc. Once the best hyperparameters are identified, the model can be trained using these parameters and evaluated on a separate test set to assess its generalization ability.

7.2 Model Tuning Process and Experiments

The baseline model, which is the Fernet model, is evaluated, and the results observed are quite good. But fine-tuning the model with hyperparameters to generate the best possible model. The model uses the He initializer technique for initializing weights. Other than that, Fernet utilizes early stopping in order not to overfit the model, changes the learning rate as hyperparameters using reduceonplateau with a factor of 0.5, and patience of 7. The model then decreases the batch size to 32 and trains it for 50 epochs. After the model is trained with the given parameters the results are noted.

The Fernet model is fine-tuned with the hyper-parameters and and the results are noted. The model has a train loss of 0.01, while the validation loss is 0.13. The model yields about 99.90% during training, and the validation accuracy seems to be 96.54%. These training results are a good indication that the model performs well and good. The Confusion Matrix by Fernet after the fine-tuning process with the hyper-parameters is given in the below image.

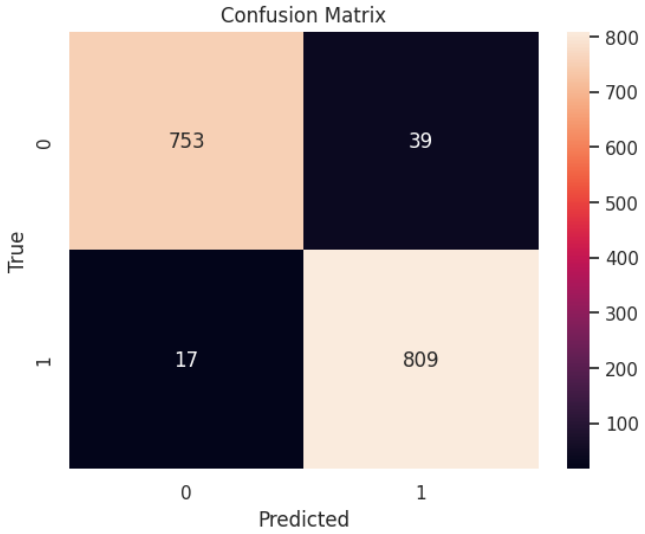


Figure: 7.1 Fine-tuned Fernet Confusion Matrix.

The model surprisingly performs well and the accuracy turns out to be 96% and the precision scores to 0.94 and the recall turns out to be 0.98. The results after the fine-tuning process are depicted in the table given below.

Table: 7.1 Fine-tuned Fernet Results.

Metrics	Value
Accuracy	0.96
Precision	0.94
Recall	0.98
F1-Score	0.96

Chapter 8

USER INTERFACE DESIGN AND EVALUATION

8.1 Designing Graphical User Interface

The mood-music application incorporates a Graphical User Interface (GUI) design for rendering a smooth user interface. The application is designed to offer users a seamless and intuitive encounter, letting them conveniently move around through the various features and settings available in the application. Besides, the GUI layout also highlights the overall aesthetic entice of the application, ensuring it is more visually fascinating and engaging for users.

The application comes with three modules, listed below as follows:

- Video Streaming Module
- Emotion Prediction Module
- Song Recommendation Module.

The video streaming module is where the video of a user is streamed in the interface. Because the app makes use of video streaming, the streaming video from the web camera is rendered back for displaying the video in the interface. It consists of start and stop buttons for better control of the video streaming process, allowing users to easily start and stop the video feed as needed. The emotion prediction module is where the facial emotions of the user are predicted using the Fernet model run in the backend of the application. The user is provided with a button to predict their emotions, trigger the emotion prediction process and receive real-time feedback on their emotional state. It results in an emotion class label and an emoji related to that emotion class. The song recommendation module is where the songs are displayed and comes in handy with a button. It results in displaying some seven songs recommended by the application. Then the user may touch that link to listen to the songs listed which takes them to the Spotify platform. The whole layout of the GUI design of the mood music application is depicted in the given diagram below.

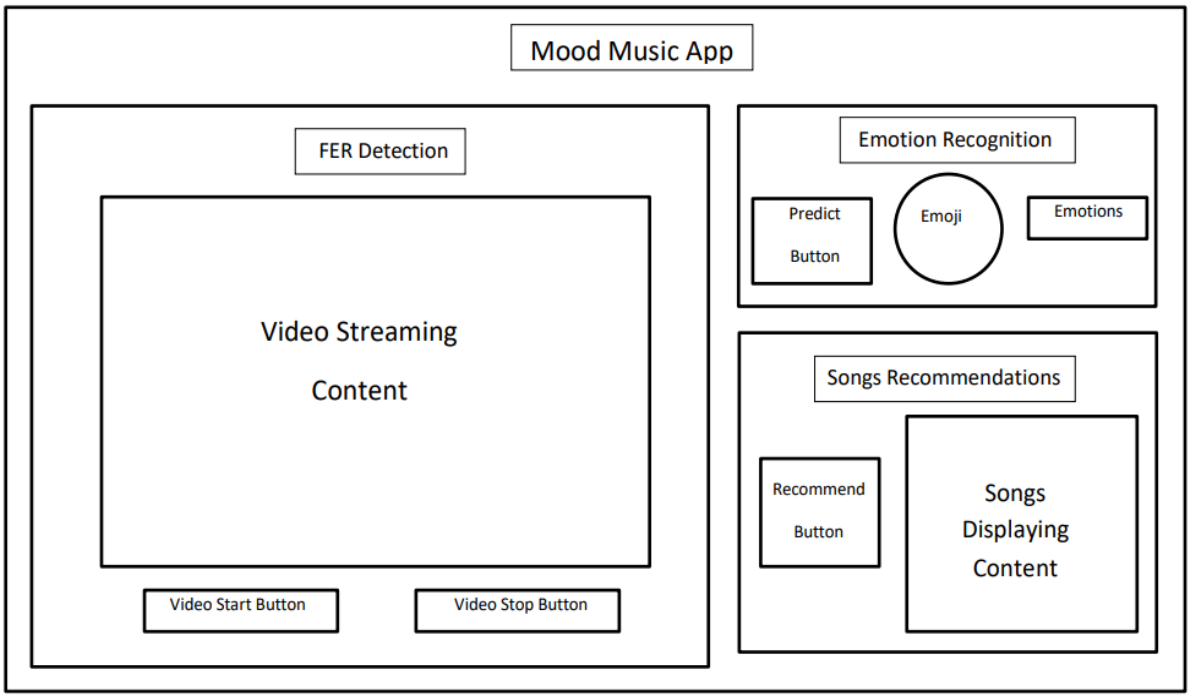


Figure: 8.1 App Design Layout.

8.2 Testing Graphical User Interface

The application is tested for its functions and uses of the interface. All the modules are tested that are present in that application. The testing comes with various types of testing for evaluating the performance of the designed GUI. Various tests were conducted, such as Accessibility tests, Functionality tests, Performance tests, Compatibility tests and Usability tests.

The app has a layout in which all three modules are present in one single window display for better accessibility. The elements, such as buttons and video streaming content, are functioning properly as expected. Despite the glitches in the application due to internet issues, the application performs well for its users. The app seems to be compatible with the Chrome browser and is designed only for use in landscape mode. Thus, the above factors provide better usability for its users. Overall, testing the GUI is essential to ensure that it is functional, easy to use, and meets the needs of the user. Hence the above testing ensures that the GUI is effective and efficient in a variety of situations and scenarios.

Chapter 9

PRODUCT DELIVERY AND DEPLOYMENT

9.1 User Manuals

The proposed approach tries to implement a web application that is developed using the Flask framework and deployed using the Render Cloud Platform. The app instance is initialized, and the coding for the running of the application is implemented in the app.py file and necessary files needed for running the app.py file such as index.html, style.css, required images and saved Fernet model, cascade model, etc. are also created. The app could be tested to run on a local server using the command "python app.py". The structure of the directory is given in the below image.

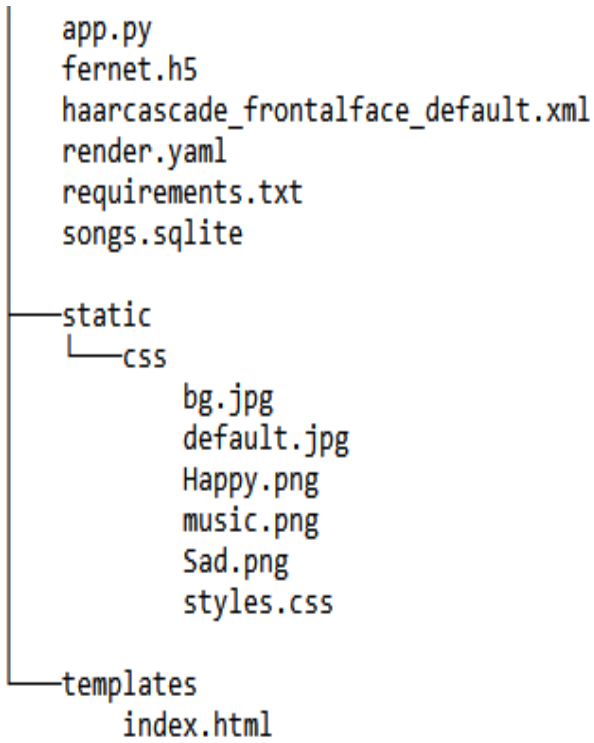


Figure: 9.1 App Directory Structure

9.2 Delivery Schedule

The mood-based music recommendation system using the facial emotion model proposes a web application called mood-music-app. This application consists of many files such app.py, static and templates folder, etc. And the schedule for developing and delivering this app is shown in the table below.

Table: 9.1 Product Delivery Schedule

Schedule	Deliverables
First Week	Project Proposal
Second Week	Designing
Fourth Week	Changing the Design
Sixth Week	Developing the app
Seventh Week	Run in local server
Eighth Week	Deployment
Tenth Week	Product delivery

9.3 Deployment Process

The application is deployed in Render which is more like Heroku. Render is a cloud-based platform that provides a range of services for building, deploying, and scaling web applications and services. Render supports a variety of languages and frameworks, including Python and Flask. Initially, an account is created in Render. The required files needed for the application are pushed to the GitHub main branch. Then the GitHub repo is connected in Render. Then a new web service is created, the required build and start commands are given and the app is deployed using Render.

Chapter 10

CONCLUSION

10.1 Summary

Music can be closely related to the mood of an individual. But popularly, music recommendation systems have been used with any kind of filtering technique, like content-based filtering, collaborative filtering, and hybrid-based filtering. However, these methods and techniques come with a drawback in that the mood of the user is not detected, as the listing preferences are based on the mood of an individual. Music genre can be closely related to the facial emotions of an individual, as certain genres of music are more likely to evoke specific emotional responses in listeners. So the goal of this research is to use facial emotion detection techniques to generate inputs for a music recommendation system, thereby enhancing the accuracy of the ensuing music recommendation system. Overall, a facial emotion-based music recommendation system can provide a more personalized and engaging experience for users while also benefiting businesses through enhanced marketing opportunities along with the fact that it depreciates the cold start issues of many recommendation systems.

10.2 Limitations and Future Work

Nevertheless, the system defaults in certain aspects, particularly when dealing with protecting user privacy or security issues, while the system utilizes the face of an individual because extracting the user's face is a violation of their privacy. But as future work, this situation could be addressed with the use of Blockchain technology to enhance the security of the system as well as protect the privacy of the user.

REFERENCES

[1] Parmar Darshna, "Music Recommendation Based on Content and Collaborative Approach & Reducing Cold Start Problem", Proceedings of the Second International Conference on Inventive Systems and Control (ICISC 2018), IEEE Xplore, 2018.

[2] Wang Wenzhen, "Personalized Music Recommendation Algorithm based on Hybrid Collaborative Filtering Technology", International Conference on Smart Grid and Electrical Automation (ICSGEA), 2019.

[3] Mukkamala. S.N.V. Jitendra, Y. Radhika, "An Automated Music Recommendation System Based on Listener Preferences", Recent Trends in Intensive Computing M. Rajesh et al. (Eds.) © 2021 The authors and IOS Press, 2021.

[4] Gang Li, Jingjing Zhang, "Music personalized recommendation system based on improved KNN algorithm", IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2018.

[5] Haoye Tian, Haini Cai, Junhao Wen, Shun Li, Yingqiao Li, "A Music Recommendation System Based on logistic regression and extreme Gradient Boosting", IJCNN 2019, International Joint Conference on Neural Networks. Budapest, Hungary, 14-19 July, 2019.

[6] Ferdos Fessahaye, Luis Perez, Tiffany Zhan, Raymond Zhang, Calais Fossier, Robyn Markarian, Carter Chiu, Justin Zhan, Laxmi Gewali, and Paul Oh, "T-RECSYS: A Novel Music Recommendation System Using Deep Learning", University of Nevada Las Vegas, Las Vegas, United States, 2019.

[7] Shun-Hao Changa, Ashu Abdula, Jenhui Chen, and Hua-Yuan Liao, "A Personalized Music Recommendation System Using Convolutional Neural Networks Approach", Proceedings of IEEE International Conference on Applied System Innovation, 2018.

[8] Ke Chen, Beici Liang, Xiaoshuan Ma, Minwei Gu, "Learning Audio Embeddings with user listening data for content based Music Recommendation", IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021.

[9] M.Ravidu Shehan Perera and Sulochana Rupasinghe, "Music Recommendation System based on Emotions in User's Social Media behaviour", TechRxiv, 2021.

[10] Kajornsak Kittimathaveenan, Chanathip Pongskul, Salisa Mahatanarat, "Music Recommendation Based on Color", Middlesex University, 2020.

APPENDIX-A

DATA SET

FER DATA SAMPLE:

Happy Images:

Sad Images:

SONG DATA SAMPLE:

track_id	track_name	artist_name	genres	popularity
0VjIjW4GlUZAMYd2vXMi3b	Blinding Lights	The Weeknd	cheerful	1
6UelLqGlWMcVH1E5c4H7lY	Watermelon Sugar	Harry Styles	cheerful	0.977011
7ef4DlsgrMEH11cDZd32M6	One Kiss (with Dua Lipa)	Calvin Harris	sombre	0.965517
3w3y8KPTfNeOKPiqUTakBh	Locked out of Heaven	Bruno Mars	cheerful	0.954023
7qiZfU4dY1lWllzX7mPBI3	Shape of You	Ed Sheeran	sombre	0.954023
0ct6r3EGTcMLPtrXHDvVjc	The Nights	Avicii	cheerful	0.942529
1mea3bSkSGXuIRvnydlB5b	Viva La Vida	Coldplay	sombre	0.942529
1zi7xx7UVEFkmKfv06H8x0	One Dance	Drake	cheerful	0.942529

APPENDIX-B

SOURCE CODE

FER MODEL CODE:

```
# # FER Model Creation

# ## Imports

import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_theme(style="whitegrid")
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Dropout, Dense, Flatten
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import *

# ## Data

df = pd.read_csv('/content/drive/MyDrive/fer.csv')

# ## EDA

df.head()
df.shape
df.info()
df.emotion.value_counts()

# ## Pre-processing

img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48, 1).astype('float32'))
img_array = np.stack(img_array, axis=0)
img_array.shape
img_labels = df.emotion.values
img_labels.shape

# ## Splitting the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(img_array, img_labels, test_size=0.1, shuffle=True, random_state=42)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1,
        random_state=42)
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("\n")
print("X_val shape: {}".format(X_val.shape))
print("y val shape: {}".format(y_val.shape))
print("\n")
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
print(np.round(((X_train.shape[0]) / (img_array.shape[0]))*100), "% of Train set")
print(np.round(((X_val.shape[0]) / (img_array.shape[0]))*100), "% of Validation set")
print(np.round(((X_test.shape[0]) / (img_array.shape[0]))*100), "% of Test set")

# ## Image Properties

img_width = X_train.shape[1]
img_height = X_train.shape[2]
img_depth = X_train.shape[3]
print("The following are the image properties :")
print("   Width =", img_width)
print("   Height =", img_height)
print("   Depth =", img_depth)

# ## Normalization

X_train = X_train / 255.
X_val = X_val / 255.
X_test = X_test / 255.

# ## Model

def model():

   net = Sequential(name='DCNN')

   net.add(
      Conv2D(
         filters=32,
         kernel_size=(5,5),
         input_shape=(img_width, img_height, img_depth),
         activation='relu',
         padding='same',
         kernel_initializer='he_normal',
         name='conv2d_1'))

   net.add(BatchNormalization(name='batchnorm_1'))

   net.add(
      Conv2D(
```

```
        filters=32,
        kernel_size=(5,5),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_2'))

net.add(BatchNormalization(name='batchnorm_2'))
net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_1'))
net.add(Dropout(0.4, name='dropout_1'))

net.add(
    Conv2D(
        filters=64,
        kernel_size=(3,3),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_3'))

net.add(BatchNormalization(name='batchnorm_3'))

net.add(
    Conv2D(
        filters=64,
        kernel_size=(3,3),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_4'))

net.add(BatchNormalization(name='batchnorm_4'))
net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_2'))
net.add(Dropout(0.4, name='dropout_2'))

net.add(
    Conv2D(
        filters=128,
        kernel_size=(3,3),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_5'))

net.add(BatchNormalization(name='batchnorm_5'))

net.add(
    Conv2D(
        filters=128,
        kernel_size=(3,3),
```

```
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_6'))

net.add(BatchNormalization(name='batchnorm_6'))
net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_3'))
net.add(Dropout(0.5, name='dropout_3'))

net.add(
    Conv2D(
        filters=256,
        kernel_size=(3,3),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_7'))

net.add(BatchNormalization(name='batchnorm_7'))

net.add(
    Conv2D(
        filters=256,
        kernel_size=(3,3),
        activation='relu',
        padding='same',
        kernel_initializer='he_normal',
        name='conv2d_8'))

net.add(BatchNormalization(name='batchnorm_8'))
net.add(MaxPooling2D(pool_size=(2,2), name='maxpool2d_4'))
net.add(Dropout(0.5, name='dropout_4'))

net.add(Flatten(name='flatten'))

net.add(
    Dense(
        512,
        activation='relu',
        kernel_initializer='he_normal',
        name='dense_1'))

net.add(BatchNormalization(name='batchnorm_9'))
net.add(Dropout(0.6, name='dropout_5'))

net.add(
    Dense(
        128,
        activation='relu',
        kernel_initializer='he_normal',
```

```
            name='dense_2'))

    net.add(BatchNormalization(name='batchnorm_10'))
    net.add(Dropout(0.6, name='dropout_6'))


    net.add(
        Dense(
            1,
            activation='sigmoid',
            name='out_layer'))

    net.compile(
        loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    net.summary()

    return net

# ## Callbacks

checkpoint = ModelCheckpoint(filepath='fernet.h5',
                             save_best_only=True,
                             verbose=1,
                             mode='min',
                             moniter='val_accuracy')
early_stopping = EarlyStopping(monitor='val_accuracy',
                           min_delta=0.00005,
                           patience=11,
                           verbose=1,
                           restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.5,
                              patience=7,
                              verbose=1,
                              min_lr=1e-7)
callbacks = [checkpoint, early_stopping, reduce_lr]

# ## Model Summary

fernet = model()
get_ipython().system('pip install visualkeras')
import visualkeras
from tensorflow.keras import layers
from collections import defaultdict
color_map = defaultdict(dict)
color_map[layers.Conv2D]['fill'] = '#ffffff'
color_map[layers.BatchNormalization]['fill'] = '#dedcd9'
```

```
color_map[layers.MaxPooling2D]['fill'] = '#b0afac'
color_map[layers.Dropout]['fill'] = '#82817f'
color_map[layers.Dense]['fill'] = '#030302'
color_map[layers.Flatten]['fill'] = '#575552'
visualkeras.layered_view(fernet, legend=True, color_map=color_map, draw_volume=True, draw_funnel=False)

# ## Model Training

batch_size = 32
epochs = 50
steps_per_epoch = len(X_train) / batch_size
validation_steps = len(X_val) / batch_size
history = fernet.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     steps_per_epoch=steps_per_epoch,
                     epochs=epochs,
                     callbacks=callbacks,
                     validation_steps=validation_steps)

# ## Result Plots

def plot_curves(history):

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    accuracy = history.history["accuracy"]
    val_accuracy = history.history["val_accuracy"]

    epochs = range(len(history.history["loss"]))

    plt.figure(figsize=(15,5))

    #plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label = "train_loss")
    plt.plot(epochs, val_loss, label = "val_loss")
    plt.title("Model Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    #plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label = "train_accuracy")
    plt.plot(epochs, val_accuracy, label = "val_accuracy")
    plt.title("Model Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
```

```
    plt.legend()
    plt.tight_layout()
plot_curves(history)
df_accu = pd.DataFrame({'train': history.history['accuracy'], 'val': history.history['val_accuracy']})
df_loss = pd.DataFrame({'train': history.history['loss'], 'val': history.history['val_loss']})
fig = plt.figure(0, (14, 4))
ax = plt.subplot(1, 2, 1)
sns.violinplot(x="variable", y="value", data=pd.melt(df_accu), showfliers=False)
plt.title('Accuracy')
plt.tight_layout()
ax = plt.subplot(1, 2, 2)
sns.violinplot(x="variable", y="value", data=pd.melt(df_loss), showfliers=False)
plt.title('Loss')
plt.tight_layout()
plt.show()

# ### Model Evaluation

train_loss, train_accuracy = fernet.evaluate(X_train, y_train)
print("Train loss = {:.2f}".format(train_loss))
print("Train accuracy = {:.2f}%".format(train_accuracy*100))
val_loss, val_accuracy = fernet.evaluate(X_val, y_val)
print("Validation loss = {:.2f}".format(val_loss))
print("Validation accuracy = {:.2f}%".format(val_accuracy*100))
val_pred = fernet.predict(X_val)
y_val_pred = np.where(val_pred > 0.5, 1, 0)
print('Confusion Matrix')
cm = tf.math.confusion_matrix(y_val, y_val_pred, num_classes=2, dtype=tf.dtypes.int32)
sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
fpr, tpr, _ = roc_curve(y_val, y_val_pred)
roc_auc = auc(fpr, tpr)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```

```
# ### New data for Predictions

pred = fernet.predict(X_test)
y_pred = np.where(pred > 0.5, 1, 0)
print('Classification Report:')
print(classification_report(y_test, y_pred))
test_loss, test_accuracy = fernet.evaluate(X_test, y_test)
print("Test loss = {:.2f}".format(test_loss))
print("Test accuracy = {:.2f}%".format(test_accuracy*100))
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
print(f"Accuracy: {acc:.2f} \nPrecision: {prec:.2f} \nRecall: {rec:.2f}")
Emotion_Classes = {0 : 'happy', 1 : 'sad'}
np.random.seed(2)
random_happy_imgs = np.random.choice(np.where(y_test[:]==0)[0], size=9)
random_sad_imgs = np.random.choice(np.where(y_test[:]==1)[0], size=9)
fig = plt.figure(1, (20, 5))
for i, (happyidx, sadidx) in enumerate(zip(random_happy_imgs, random_sad_imgs)):

    ax = plt.subplot(2, 9, i+1)
    hap_sample_img = X_test[happyidx,:,:,0]
    hap_img_resized = hap_sample_img.reshape(1,48,48,1)
    hap_pred = fernet.predict(hap_img_resized)
    y_pred_hap = np.where(hap_pred > 0.5, 1, 0)

    if (y_pred_hap[0]) == (y_test[happyidx]):
       color_hap = "green"
    else:
       color_hap = "red"

    hap_img = hap_img_resized.reshape(48,48)
    ax.imshow(hap_img, cmap='gray')
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title(f"true:happy, predict:{Emotion_Classes[y_pred_hap[0][0]]}", color=color_hap)

    ax = plt.subplot(2, 9, i+10)
    sad_sample_img = X_test[sadidx,:,:,0]
    sad_img_resized = sad_sample_img.reshape(1,48,48,1)
    sad_pred = fernet.predict(sad_img_resized)
    y_pred_sad = np.where(sad_pred > 0.5, 1, 0)

    if (y_pred_sad[0]) == (y_test[sadidx]):
       color_sad = "green"
    else:
       color_sad = "red"

    sad_img = sad_img_resized.reshape(48,48)
    ax.imshow(sad_img, cmap='gray')
```

```
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"true:sad, predict:{Emotion_Classes[y_pred_sad[0][0]]}", color=color_sad)

        plt.tight_layout()

# `Baseline model`
```

SONG DATA EXTRACTION CODE:

```
# # Data Extraction using Spotify Web API

# ## Imports

import pandas as pd
import spotipy
import spotipy.oauth2 as oauth2
from spotipy.oauth2 import SpotifyOAuth
from spotipy.oauth2 import SpotifyClientCredentials


# ## Credentials

with open("spotify_credentials.txt") as f:
    credentials = f.readlines()
    cid = credentials[0][:-1]
    secret = credentials[1]

client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

# ## Functions to extract data

def get_track_ids(user, playlist_id):

    track_ids = []

    playlist = sp.user_playlist(user, playlist_id)

    for item in playlist['tracks']['items']:
        track = item['track']
        track_ids.append(track['id'])

    return track_ids

def get_track_features(track_ids):

    meta = sp.track(track_ids)
    features = sp.audio_features(track_ids)
```

```
    track_id = meta['id']
    track_name = meta['name']
    album_name = meta['album']['name']
    artist_name = meta['album']['artists'][0]['name']
    popularity = meta['popularity']

    danceability = features[0]['danceability']
    loudness = features[0]['loudness']
    speechiness = features[0]['speechiness']
    acousticness = features[0]['acousticness']
    liveness = features[0]['liveness']
    instrumentalness = features[0]['instrumentalness']
    energy = features[0]['energy']
    valence = features[0]['valence']
    key = features[0]['key']
    mode = features[0]['mode']
    tempo = features[0]['tempo']
    time_signature = features[0]['time_signature']
    length = features[0]['duration_ms']

    art_result = sp.search(artist_name)
    art_track = art_result['tracks']['items'][0]
    artist = sp.artist(art_track["artists"][0]["external_urls"]["spotify"])
    genres = artist["genres"]

    track_data = [track_id,track_name,
                  album_name,artist_name,genres,popularity,
                  danceability,loudness,speechiness,
                  acousticness,liveness,instrumentalness,
                  energy,valence,mode,
                  key,tempo,time_signature,length
                 ]

    return track_data

def get_data(user, playlist_id):

    track_ids = get_track_ids(user, playlist_id)

    track_list = []

    for i in range(len(track_ids)):
        track_data = get_track_features(track_ids[i])
        track_list.append(track_data)

        df = pd.DataFrame(track_list, columns = ['track_id','track_name',
                                        'album_name','artist_name','genres','popularity',
                                        'danceability','loudness','speechiness',
                                        'acousticness','liveness','instrumentalness',
```

```
                                    'energy','valence','mode',
                                    'key','tempo','time_signature','length'
                                   ])

    return df

df1 = get_data('spotify', '37i9dQZF1DWZKuerrwoAGz')
df2 = get_data('spotify', '4WloBZWLuV80F07SCPxs09')
df3 = get_data('spotify', '3Kz5KBE3Ksupz9odBGwze6')

# ## Saving the data to a file

frames = [df1, df2, df3]
df = pd.concat(frames)
df.index = pd.RangeIndex(start=0, stop=len(df.index), step=1)
df.to_csv('data/spotify_data/spotify.csv', sep=',', index=False)

# `Created spotify tracks data`
```

SONG GENRE IDENTIFICATON CODE:

```
# # Spotify Tracks Genre Classification

# ## Imports

import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
sns.set_theme(style="whitegrid")
import plotly.express as px
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE


# ## Data

df = pd.read_csv('data/spotify_data/spotify.csv')

# ## EDA

df.head(3)
```

```
df.tail(3)
print("The dataset consists of :")
print("   The total rows =", df.shape[0])
print("   The total columns =", df.shape[1])
print("   The totsl size =", df.size)
df.columns
df.info()

# ## Checking for missing values

df.isnull().sum()

# ## Dropping duplicate values

df = df.drop_duplicates(subset=['track_id'])
df = df.drop_duplicates(subset=['track_name'])
print("Are all samples unique: ", len(pd.unique(df.track_name))==len(df))
print("After drooping the duplicate samples the dataset consists of :")
print("   The total rows =", df.shape[0])
print("   The total columns =", df.shape[1])
print("   The totsl size =", df.size)

# ## Normalization

features = df.select_dtypes(np.number)
features = features.values
minmax = MinMaxScaler()
scaled_features = minmax.fit_transform(features)
df[['popularity','danceability','loudness','speechiness','acousticness',
   'liveness','instrumentalness','energy','valence',
   'mode','key','tempo','time_signature','length']] = scaled_features


# ## Feature selection

x                                                                                =
['danceability','speechiness','acousticness','liveness','instrumentalness','mode','tempo','time_signat
ure']
X = df.filter(x)

# ## Identifying genre by clustering

Sum_of_squared_distances = []
K = range(1,10)

for k in K:
   km = KMeans(n_clusters=k)
   km = km.fit(X)
   Sum_of_squared_distances.append(km.inertia_)
```

```
for n_clusters in range(2,10):

    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict(X)

    centers = clusterer.cluster_centers_

    score = silhouette_score(X, preds, metric='euclidean')

    print("For n_clusters = {}, silhouette score = {}".format(n_clusters, score))

plt.plot(K, Sum_of_squared_distances, 'gx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()

kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
df['cluster'] = kmeans.predict(X)

# ## Visualizing the Clusters

pca = PCA(n_components=2)
pca_song_embedding = pca.fit_transform(X)

pca_projection = pd.DataFrame(columns=['x', 'y'], data=pca_song_embedding)
pca_projection['popularity'] = df['popularity']
pca_projection['song'] = df['track_name']
pca_projection['cluster'] = df['cluster']

fig = px.scatter(pca_projection,
                 x='x', y='y',
                 color='cluster',
                 hover_data=['x', 'y', 'song'],
                 size="popularity",
                 size_max=30,
                 template="plotly_dark",
                 title="Genre clusters")
fig.show()

# ## Genre Classification of songs

df[df['cluster'] == 0].track_name
df.loc[df['cluster'] == 0, 'genres'] = 'cheerful'
# **Genre of cluster 0 be __`CHEERFUL`__**
df[df['cluster'] == 1].track_name
df.loc[df['cluster'] == 1, 'genres'] = 'sombre'
# **Genre of cluster 1 be __`SOMBRE`__**
df.to_csv('data/spotify_data/spotify_genre.csv', index=False)
```

SONG DATA STORE IN DATABASE CODE:

```
# # Song data to database

# ## Imports

import sqlite3
import pandas as pd

# ## Data

full_data = pd.read_csv('data/spotify_data/spotify_genre.csv')

# ## Feature Selection

song_data = full_data[['track_id', 'track_name', 'artist_name', 'genres', 'popularity']]

# ## Storing the data to database

conn = sqlite3.connect('songs.sqlite')
table_name = 'songs'
query = f'Create table if not Exists {table_name} (track_id TEXT, track_name TEXT, artist_name TEXT, genres TEXT, popularity REAL)'
conn.execute(query)
song_data.to_sql(table_name, conn, if_exists='replace', index=False)
data = pd.read_sql("select * from songs", conn)
conn.commit()
conn.close()
data.to_csv('data/spotify_data/songs.csv', sep=',', index=False)

# `Stored the data in the database`
```

UTILITIES CODE:

```
import cv2
import time
import sqlite3
import numpy as np
import pandas as pd
import ipywidgets as widgets
from keras.models import load_model
from IPython.display import display, Markdown, Image

stopButton = widgets.ToggleButton()

cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

fernet = load_model('fernet.h5', compile=False)
```

```
label_map = ['Happy', 'Sad']

def connect_db():
    conn = sqlite3.connect('songs.sqlite')
    df = pd.read_sql("select * from songs", conn)
    conn.commit()
    conn.close()
    return df

def recommend_songs(emotion, data):
    if (emotion=='Happy'):
        Play = data[data['genres'] =='cheerful']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[['track_id','track_name','artist_name']][:7].reset_index(drop=True)
        return Play
    if (emotion=='Sad'):
        Play = data[data['genres'] =='sombre']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[['track_id','track_name','artist_name']][:7].reset_index(drop=True)
        return Play

def play_song(data):
    for names, ids in zip(data.track_name, data.track_id):
        display(Markdown('[{}](https://open.spotify.com/track/{})'.format(names, ids)))
```

SYSTEM CODE:

```
# # Facial Emotion based Music Recommendation System

# ## Imports

import threading
from utilities import *

# ## System

def system(stopButton):

    cap = cv2.VideoCapture(0)
    display_handle = display(None, display_id=True)

    while True:
        print('Camera is open!')
        sus, frame = cap.read()

        print('Converting frame to grayscale.')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        sus, frame_dis = cv2.imencode('.jpeg', frame)
        display_handle.update(Image(data=frame_dis.tobytes()))
```

```
        print('Detecting the face.')
        faces = cascade.detectMultiScale(gray,1.3,2)

        print('Selecting the region of interest.')
        for (x, y, w, h) in faces:
            ret = cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), 2)
            sus, frame_ret = cv2.imencode('.jpeg', ret)
            display_handle.update(Image(data=frame_ret))

        print('Coping the image.')
        for x,y,w,h in faces:
            roi = gray[y:y+h, x:x+w]
            roi = cv2.imwrite("face.jpg", roi)
            time.sleep(0.1)
        print('Reshaping the image.')
        roi = np.reshape((cv2.resize(roi, (48,48))) / 255.0, (1,48,48,1))

        print('Emotion Recognition using fernet.')
        model_pred = fernet.predict(roi, verbose=0)

        print('Emotion Classification using fernet.')
        label = np.where(model_pred > 0.5, 1, 0)

        emotion = label_map[label[0][0]]
        print('Emotion classified is', emotion, '!!!')

        print('Fetching the song data from database.')
        song_data = connect_db()

        print('Recommending songs !!!')
        song_data = recommend_songs(emotion, song_data)
        display(song_data)

        print('Play the playlist !!!')
        play_song(song_data)

        if stopButton.value==False:
            stopButton.value==True
            cap.release()

        time.sleep(10*10*10*10)

thread = threading.Thread(target=system, args=(stopButton,))
thread.start()

# `Created the Facial Emotion based Music Recommendation System`
```

FLASK APP CODE:

```
# # Flask app
import cv2
import sqlite3
import numpy as np
import pandas as pd
from keras.models import load_model
from flask import Flask, Response, render_template, request, jsonify, url_for

app = Flask(__name__)

fernet = load_model('fernet.h5', compile=False)
global roi
global emotion
global emotion_img_path

def process_video():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
        faces = face_cascade.detectMultiScale(gray, 1.3, 2)
        for x,y,w,h in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 255), 2)
            face_roi = gray[y:y+h, x:x+w]
            face_roi = np.reshape((cv2.resize(face_roi, (48,48))) / 255.0, (1,48,48,1))
            global roi
            roi = face_roi

        ret, jpeg = cv2.imencode('.jpg', frame)
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')
    cap.release()

def connect_db():
    conn = sqlite3.connect('songs.sqlite')
    df = pd.read_sql("select * from songs", conn)
    conn.commit()
    conn.close()
    return df

def recommend_songs(emotion, data):
    if (emotion=='Happy'):
        Play = data[data['genres'] =='cheerful']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[['track_id','track_name','artist_name']][:7].reset_index(drop=True)
        return Play
    if (emotion=='Sad'):
```

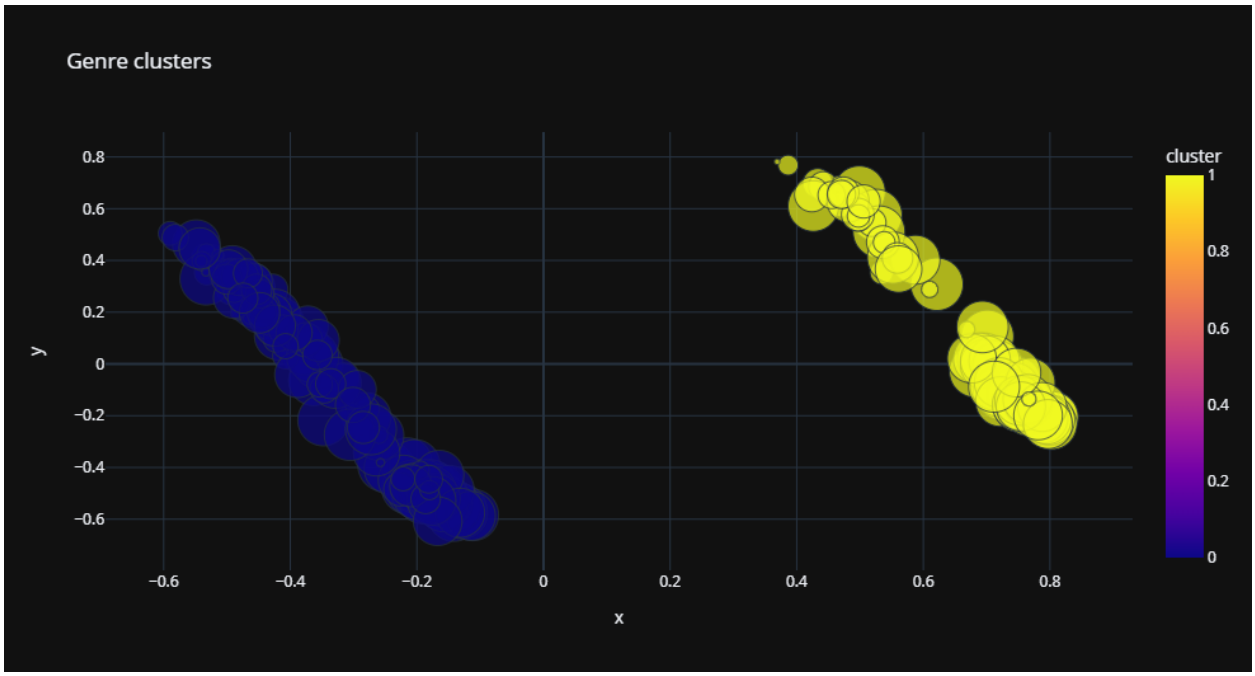
```
        Play = data[data['genres'] =='sombre']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[['track_id','track_name','artist_name']][:7].reset_index(drop=True)
        return Play

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

@app.route('/start_streaming', methods=['GET','POST'])
def start_streaming():
    return Response(process_video(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/stop_streaming', methods=['GET','POST'])
def stop_streaming():
    return jsonify({'success': True})

@app.route('/emotion_prediction_function', methods=['POST', 'GET'])
def emotion_prediction_function():
    global roi
    model = load_model('fernet.h5', compile=False)
    label_map = ['Happy', 'Sad']
    img_path = ['css/Happy.png' , 'css/Sad.png']
    model_pred = fernet.predict(roi, verbose=0)
    label = np.where(model_pred > 0.5, 1, 0)
    global emotion
    emotion = label_map[label[0][0]]
    global emotion_img_path
    emotion_img_path = img_path[label[0][0]]
    return render_template('index.html',emotion_prediction_label=emotion,
image_path=emotion_img_path)

@app.route('/static/<path:path>', methods=['GET'])
def send_static(path):
    return send_from_directory('static', path)

@app.route('/recommend_tracks_function', methods=['POST', 'GET'])
def recommend_tracks_function():
    song_data = connect_db()
    global emotion
    data = recommend_songs(emotion, song_data)
    song_dict = {}
    for i in range(7):
        names = data.iloc[i,1] + ' - '+ data.iloc[i,2]
        url = "https://open.spotify.com/track/" + data.iloc[i,0]
        song_dict[names] = url
    return render_template("index.html", emotion_prediction_label=emotion,
image_path=emotion_img_path, results=song_dict)

if __name__ == '__main__': app.run(debug=True,  use_reloader=False)
```

APPENDIX-C

OUTPUT SCREEN SHOTS

FERNET RESULS:

GENRE CLUSTERING FOR SONGS:



SYSTEM EXPERIMENTAL RESULT:

```
Camera is open!
.....
Converting frame to grayscale.
.....
Displaying Video.
.....
Detecting the face.
.....
Selecting the region of interest.
.....
Coping the image.
.....
Reshaping the image.
.....
Emotion Recognition using fernet.
.....
Emotion Classification using fernet.
.....
Emotion classified is Happy !!!
.....
Fetching the song data from database.
```

```
Recommending songs !!!
```

	track_id	track_name	artist_name
0	0VjIjW4GlUZAMYd2vXMi3b	Blinding Lights	The Weeknd
1	6UelLqGlWMcVH1E5c4H7lY	Watermelon Sugar	Harry Styles
2	3w3y8KPTfNeOKPiqUTakBh	Locked out of Heaven	Bruno Mars
3	4RVwu0g32PAqgUiJoXsdF8	Happier Than Ever	Billie Eilish
4	1HNkqx9Ahdgi1lxy2xkKkL	Photograph	Ed Sheeran

```
.....
Play the playlist !!!
```

Blinding Lights

Watermelon Sugar

Locked out of Heaven

Happier Than Ever

Photograph

APPLICATION DEPLOYMENT SCREENSHOT: