Emotion-Based Music Player Using CNN and Real-Time Facial Expression Recognition

Mini Project Report - Fundamentals of Machine Learning Lab (DSE 2241)

Department of Data Science & Computer Applications

B.Tech Data Science

4th Semester – Batch: A1

Submitted By

Piyush Verma	230968114
Advika Chaturvedi	230968046

Mentored By

Savitha G
Assistant Professor
DSCA, MIT

Chithra K
Assistant Professor
DSCA, MIT

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Date: 18^{th} April, 2025

CERTIFICATE

This is to certify that Piyush Verma (230968114) and Advika Chaturvedi (230968046) has successfully executed a mini project titled “Emotion-Based Music Player Using CNN and Real-Time Facial Expression Recognition” rightly bringing fore the competencies and skill sets they have gained during the course- Fundamentals of Machine Learning Lab (DSE 2242), thereby resulting in the culmination of this project.

Savitha G
Assistant Professor
DSCA, MIT

Chithra K
Assistant Professor-Senior
DSCA, MIT

ABSTRACT

Facial emotion recognition has emerged as a significant subfield of computer vision, with applications ranging from mental health monitoring to human-computer interaction. In this project, we present an emotion recognition system that leverages convolutional neural networks (CNNs) trained on the FER2013 dataset to classify facial expressions into seven categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The system further maps these emotional states to curated music playlists, thereby demonstrating an end-to-end pipeline that blends machine learning with multimedia response systems.

The methodology involves preprocessing the grayscale facial images into normalized 48x48 pixel arrays and feeding them into a deep CNN model composed of stacked Conv2D layers with batch normalization and dropout regularization. The model is trained using categorical cross-entropy loss and the Adam optimizer. To improve generalization, we used learning rate reduction based on validation accuracy. The model's architecture supports robust feature extraction and classification, validated via a confusion matrix and tested on unseen data from the test set.

The trained model achieved strong classification accuracy on the validation and test datasets. The emotion detection was integrated into a Streamlit application using OpenCV and Haar cascade classifiers for real-time webcam input. Based on the detected emotion, the system automatically plays a relevant audio track, showcasing a seamless blend of machine learning inference with multimedia interaction.

This project uses **TensorFlow**, **Keras**, **Streamlit**, **OpenCV**, and **Pygame**, and provides an accessible and intuitive demonstration of real-world emotion-based interaction systems.

Contents

1. Introduction **5**

2. Synopsis **6**

2.1 Proposed System **6**

2.2 Objectives **6**

3. Functional Requirements **7**

4. Implementation **9**

5.1 Machine Learning Model **9**

5.2 Hosting using Streamlit **13**

5. Result **18**

6. Conclusion and Future Work **20**

Chapter 1 - Introduction

Emotion recognition from facial expressions has emerged as a key research area within the domain of computer vision and machine learning. Human facial expressions convey rich and nuanced emotional information, making them a powerful source for understanding and interpreting mental states. In recent years, the increasing availability of annotated datasets and advancements in deep learning architectures have accelerated progress in this field, enabling real-time emotion detection with considerable accuracy. Applications of this technology span across industries, from healthcare and education to entertainment and human-computer interaction.

The process of emotion recognition involves detecting a face from an image or video frame, preprocessing the input data, extracting relevant features, and classifying the emotion into predefined categories. Traditional machine learning models often required handcrafted features and domain expertise to yield good performance. However, with the advent of deep learning—especially convolutional neural networks (CNNs)—it has become feasible to automatically learn and extract relevant patterns from raw image data. CNNs have become the preferred approach due to their robustness, scalability, and superior performance in image classification tasks.

This project leverages a deep CNN model trained on the FER2013 dataset, a well-known benchmark for facial emotion recognition. The model is trained to classify images into seven emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. After successful emotion classification, the system further enhances the user experience by mapping each detected emotion to a curated audio playlist and playing a song that corresponds to the user's emotional state. This integration of machine learning with real-time application logic highlights the relevance and practicality of deploying ML models in real-world scenarios.

The project also includes the implementation of a live application using OpenCV for video capture and face detection, Keras for model training, and Streamlit for user interface deployment. This end-to-end system not only serves as a practical example of ML model development and deployment but also illustrates the interdisciplinary potential of machine learning when combined with multimedia systems. The need for such intelligent, emotion-aware applications is growing, and this project offers a foundational step toward building emotionally responsive systems.

Chapter 2 - Synopsis

2.1 Proposed System

The proposed system aims to classify human facial emotions in real-time and enhance user experience through personalized music recommendations. Using the FER2013 dataset, we train a convolutional neural network (CNN) model capable of recognizing seven core emotions from 48x48 grayscale facial images. The detected emotion is mapped to a pre-curated audio folder, and an emotion-specific song is played automatically.

The system integrates multiple machine learning and software components:

- A deep CNN for emotion classification.
- Image preprocessing pipeline for pixel normalization and reshaping.
- Real-time webcam-based face detection using OpenCV.
- A Streamlit interface for live feedback and visualization.
- Pygame for dynamic music playback based on the prediction.

This end-to-end system serves as a proof-of-concept for affective computing applications such as emotion-aware assistants, mood-based music players, and real-time feedback systems in healthcare or education.

2.2 Objectives

- To develop a CNN-based classifier for real-time emotion detection using the FER2013 dataset.
- To design and implement an image preprocessing pipeline for model-ready input transformation.
- To integrate face detection using OpenCV with the trained model for live video input processing.
- To map predicted emotional states to curated audio playlists and enable automatic music playback.

Chapter 3 - Functional Requirements

The application consists of several integrated modules, each responsible for a specific task such as capturing webcam input, detecting faces, processing images, predicting emotions using a trained CNN model, and playing music based on the detected emotion. The Streamlit interface supports real-time feedback and user interaction.

3.1 Webcam Integration & Frame Capture Module

Functionality: This module captures video input in real-time from the user's webcam.

Input	Access Webcam using OpenCV
PROCESSING	Start webcam and capture frame-by-frame video
OUTPUT	Live video feed available for further processing or error message

Table 3.1 Webcam Feed Input

3.2. Face Detection Module

Functionality: Detects faces in each frame using a pre-trained Haar cascade classifier.

Input	Single video frame from webcam
PROCESSING	Apply Haar cascade detection to identify facial regions
OUTPUT	Bounding box around detected face or "No face detected" message

Table 3.2 Face Detection

3.3 Preprocessing Module

Functionality: Convert the face image to grayscale, resize to 48x48, normalize pixel values, and reshape for model input.

Input	Detected face region (image)
PROCESSING	Grayscale conversion, normalization, resizing
OUTPUT	Preprocessed image tensor for CNN input

Table 3.3 Preprocessing Face Region

3.4 Emotion Prediction Module

Functionality: Predict the emotion using the trained CNN model and return the most probable label.

Input	Preprocessed face tensor (48x48x1)
PROCESSING	Pass input through CNN model to get emotion probabilities
OUTPUT	Predicted emotion label and confidence score

Table 3.4 Emotion Classification

3.5 Music Playback Module

Functionality: Map the predicted emotion to a music folder and play a random song corresponding to the detected emotion.

Input	Predicted emotion label
PROCESSING	Access music directory for emotion, select random file, play using mixer
OUTPUT	Song playback corresponding to emotion or fallback message if no file found

Table 3.5 Emotion-Based Music Selection

3.6 Streamlit User Interface Module

Functionality: Provide real-time visualization of webcam, current emotion, confidence score, and current song playing.

Input	Output from other modules (emotion, confidence, song)
PROCESSING	Render live video and statistics in a web interface
OUTPUT	Updated interface showing real-time predictions and actions

Table 3.6 User Interface and Feedback

Chapter 4 - Implementation

5.1 Machine Learning Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools

np.random.seed(2)

import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
import keras.optimizers as opt
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
SeparableConv2D, Dropout, Flatten, Dense, BatchNormalization
from keras.layers import Input

data = pd.read_csv("/kaggle/input/d/deadskull7/fer2013/fer2013.csv")
data.head()

groups = [g for _, g in data.groupby('Usage')]
train = groups[2]
val = groups[1]
test = groups[0]

train = train.drop(labels=['Usage'], axis=1)
val = val.drop(labels=['Usage'], axis=1)
test = test.drop(labels=['Usage'], axis=1)

Y_train = train["emotion"]
Y_val = val["emotion"]
Y_test = test["emotion"]

X_train = train["pixels"]
X_val = val["pixels"]
```

```
X_test = test["pixels"]

def preprocess(X):
     X = np.array([np.fromstring(image, np.uint8, sep=' ') for image in
X])
    X = X/255.0
    X = X.reshape(-1, 48, 48, 1)
    return X

X_train = preprocess(X_train)
X_val = preprocess(X_val)
X_test = preprocess(X_test)

plt.imshow(X_train[0][:,:,0])

plt.figure(figsize=(30, 7))

plt.subplot(1, 3, 1)
ax = sns.countplot(x=Y_train)
ax.set(ylabel="count", xlabel="emotion")
plt.title("Counts per emotion in training set")

plt.subplot(1, 3, 2)
ax = sns.countplot(x=Y_val)
ax.set(ylabel="count", xlabel="emotion")
plt.title("Counts per emotion in validation set")

plt.subplot(1, 3, 3)
ax = sns.countplot(x=Y_test)
ax.set(ylabel="count", xlabel="emotion")
plt.title("Counts per emotion in testing set")

plt.show()

Y_train = to_categorical(Y_train, num_classes=7)
Y_val = to_categorical(Y_val, num_classes=7)
Y_test = to_categorical(Y_test, num_classes=7)

print("Y_train shape:", Y_train.shape)
print("Y_val shape:", Y_val.shape)
print("Y_test shape:", Y_test.shape)

Y_train = train["emotion"]
```

```
Y_val = val["emotion"]
Y_test = test["emotion"]

Y_train = Y_train.to_numpy().flatten()
Y_val = Y_val.to_numpy().flatten()
Y_test = Y_test.to_numpy().flatten()

from tensorflow.keras.utils import to_categorical
Y_train = to_categorical(Y_train, num_classes=7)
Y_val = to_categorical(Y_val, num_classes=7)
Y_test = to_categorical(Y_test, num_classes=7)

print("Y_train shape:", Y_train.shape)
print("Y_val shape:", Y_val.shape)
print("Y_test shape:", Y_test.shape)

model = Sequential()
model.add(Input(shape=(48, 48, 1)))
model.add(Conv2D(32, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (5, 5), padding="same", activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.5))
model.add(Conv2D(64, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (5, 5), padding="same", activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.5))
model.add(Conv2D(128, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (5, 5), padding="same", activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(7, activation='softmax'))

optimizer = opt.Adam(learning_rate = 0.001)
lr_anneal = ReduceLROnPlateau(monitor = 'val_accuracy', patience=3,
factor=0.2, min_lr=1e-6)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
history = model.fit(X_train, Y_train, validation_data=[X_val, Y_val],
epochs=15, batch_size = 100, callbacks=[lr_anneal])

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)


    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
         for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

Y_pred = model.predict(X_val)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(Y_val,axis = 1)
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
plot_confusion_matrix(confusion_mtx, classes = range(10))

score, acc = model.evaluate(X_test, Y_test, batch_size=100)
print('Test score:', score)
print("Test accuracy:", acc)

score, acc = model.evaluate(X_test, Y_test, batch_size=100)
print('Test score:', score)
```

```python
print("Test accuracy:", acc)

model.save('cnn_model003.h5')
```

5.2 Hosting using Streamlit

```python
import streamlit as st
import cv2
import numpy as np
import time
import random
import os
from pygame import mixer
import tensorflow as tf  # Assuming your model is in TensorFlow

# Load your pre-trained model (update the path)
model = tf.keras.models.load_model('cnn_model002.h5')

# Emotions list (update according to your model)
EMOTIONS = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise',
'Neutral']

# Music folder path (organize your mp3 files in subfolders named after
emotions)
MUSIC_PATH = "MLProjSongs"

# Check model output shape
num_classes = model.output_shape[-1]  # Get the number of output
classes from the model
print(f"Model output classes: {num_classes}")  # Debug print

# Emotions list (update this based on your model's actual classes)
if num_classes != len(EMOTIONS):
    st.warning(f"Model expects {num_classes} classes, but EMOTIONS list
has {len(EMOTIONS)}. Please update EMOTIONS.")
    # Example: If your model has 7 classes, update EMOTIONS like this:
        # EMOTIONS = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad',
'Surprise', 'Neutral']

# Initialize mixer
mixer.init()

def load_random_music(emotion):
```

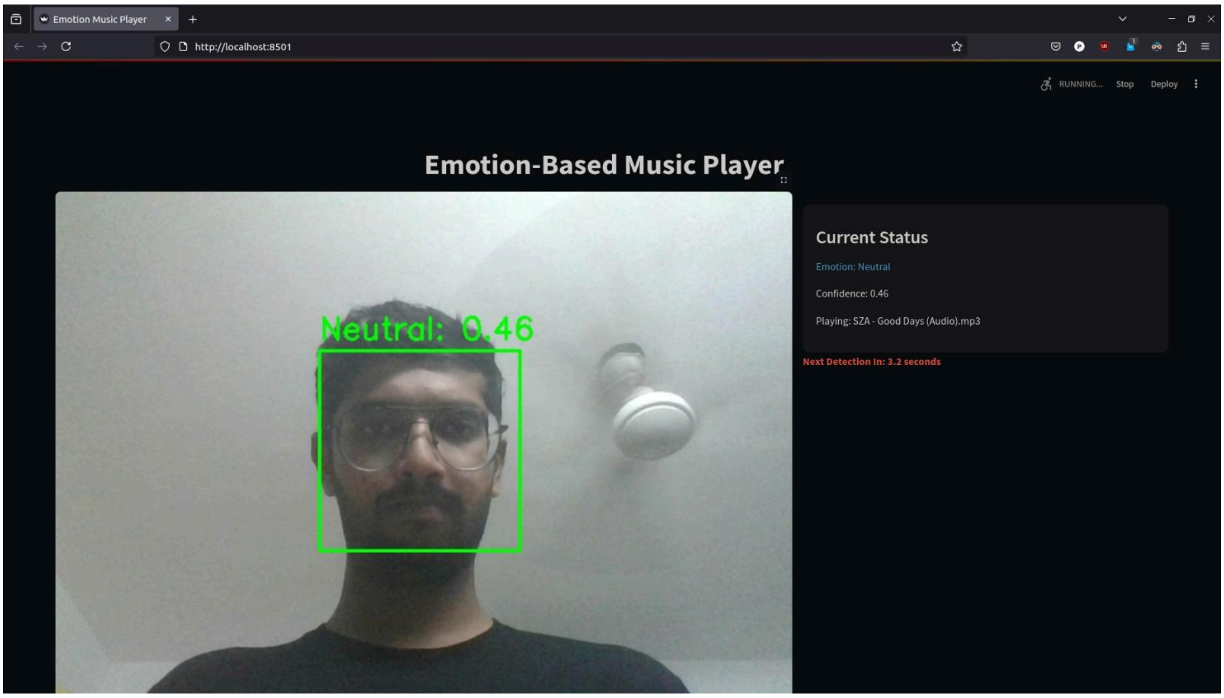
```
    emotion_folder = os.path.join(MUSIC_PATH, emotion)
     music_files = [f for f in os.listdir(emotion_folder) if f.endswith('.mp3')]
    if music_files:
        return os.path.join(emotion_folder, random.choice(music_files))
    return None

def process_frame(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (48, 48))
    normalized = resized / 255.0
    input_data = np.expand_dims(normalized, axis=(0, -1))

    prediction = model.predict(input_data)
    emotion_idx = np.argmax(prediction)
    confidence = prediction[0][emotion_idx]

    if emotion_idx >= len(EMOTIONS):
        return "Unknown", confidence
    return EMOTIONS[emotion_idx], confidence

def main():
    st.set_page_config(page_title="Emotion Music Player", layout="wide")

    st.markdown("""
    <style>
    .main {
        background-color: #f0f2f6;
        padding: 20px;
    }
    .title {
        color: #2c3e50;
        font-size: 2.5em;
        text-align: center;
        margin-bottom: 20px;
    }
    .video-container {
        border-radius: 10px;
        overflow: hidden;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    }
    .stats-box {
```

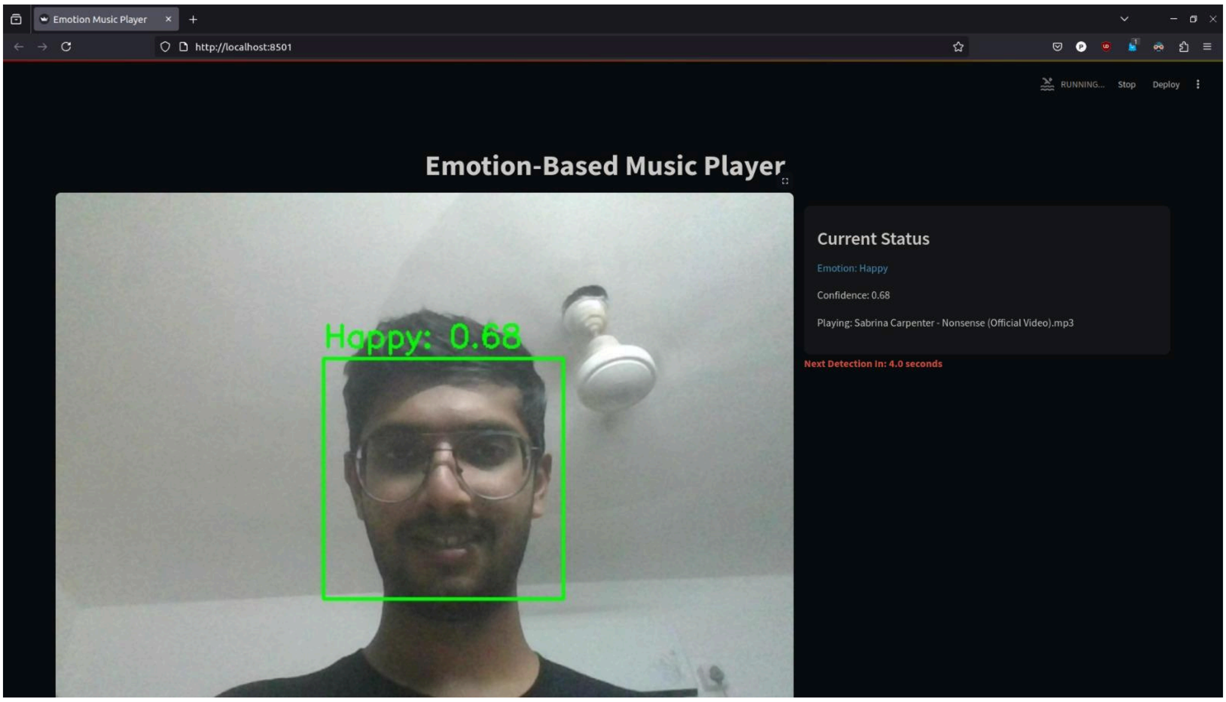
```
            background-color: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            margin-top: 20px;
        }
        .emotion-text {
            font-size: 1.5em;
            color: #2980b9;
        }
        .timer-text {
            font-size: 1.2em;
            color: #e74c3c;
            font-weight: bold;
        }
        </style>
        """, unsafe_allow_html=True)

    st.markdown('<h1 class="title">Emotion-Based Music Player</h1>',
unsafe_allow_html=True)

    col1, col2 = st.columns([2, 1])

    with col1:
        video_placeholder = st.empty()

    with col2:
        stats_placeholder = st.empty()
        timer_placeholder = st.empty()  # New placeholder for the timer

    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        st.error("Error: Could not open webcam.")
        return

    last_detection_time = time.time() - 5  # Start immediately
    current_emotion = None
    current_music = None
    confidence = 0.0
    DETECTION_INTERVAL = 5  # Set to 5 seconds

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

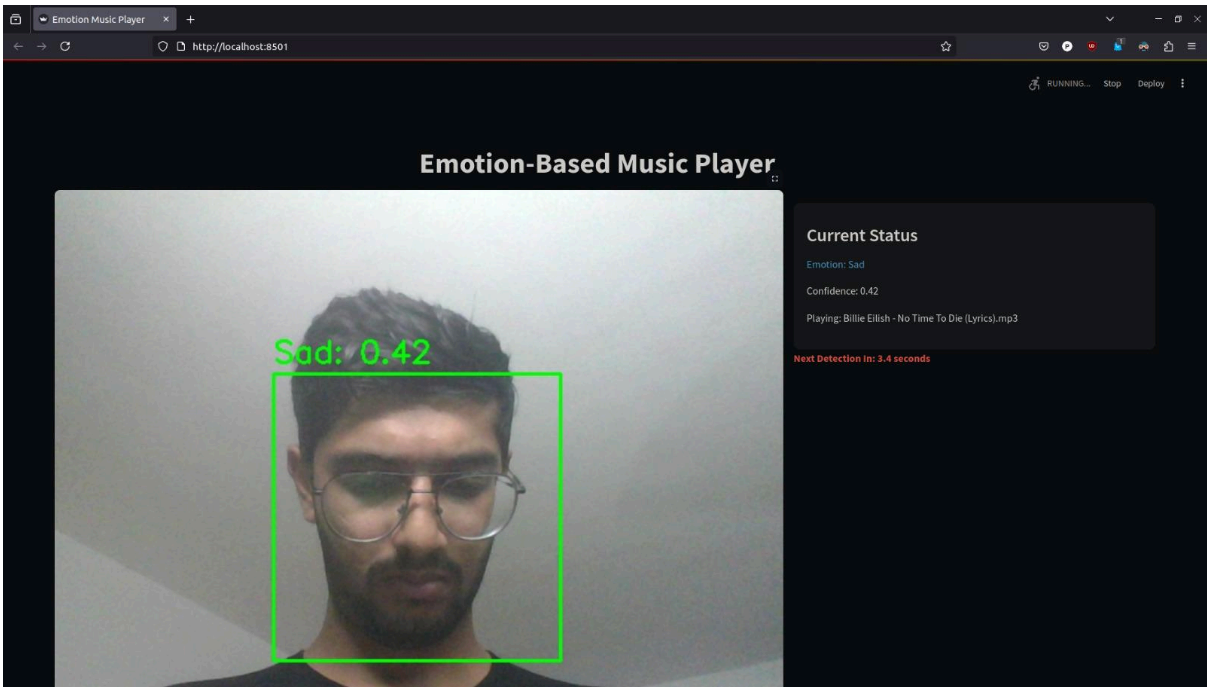
```
    if face_cascade.empty():
        st.error("Error: Could not load face cascade classifier.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            st.error("Error: Could not read frame from webcam.")
            break

          faces = face_cascade.detectMultiScale(frame, scaleFactor=1.3,
minNeighbors=5)
        current_time = time.time()

        # Calculate remaining time for the next detection
        elapsed_time = current_time - last_detection_time
        remaining_time = max(0, DETECTION_INTERVAL - elapsed_time)

        # Emotion detection every 5 seconds
        if elapsed_time >= DETECTION_INTERVAL and len(faces) > 0:
            for (x, y, w, h) in faces:
                face_roi = frame[y:y+h, x:x+w]
                emotion, conf = process_frame(face_roi)
                confidence = conf

                if emotion != current_emotion:
                    if mixer.music.get_busy():
                        mixer.music.stop()

                    music_file = load_random_music(emotion)
                    if music_file:
                        mixer.music.load(music_file)
                        mixer.music.play()
                        current_music = os.path.basename(music_file)
                    current_emotion = emotion

                break
            last_detection_time = current_time

        # Draw bounding boxes and text
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            if current_emotion:
```

```
                text = f"{current_emotion}: {confidence:.2f}"
                                  cv2.putText(frame,  text,  (x,  y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    video_placeholder.image(frame_rgb,  channels="RGB",
use_container_width=True)

        # Update stats
        stats_html = f"""
        <div class="stats-box">
            <h3>Current Status</h3>
                  <p  class="emotion-text">Emotion:  {current_emotion  or
'Detecting...'}</p>
            <p>Confidence: {confidence:.2f}</p>
            <p>Playing: {current_music or 'None'}</p>
        </div>
        """
        stats_placeholder.markdown(stats_html, unsafe_allow_html=True)

        # Update timer
        timer_html = f"""
          <p class="timer-text">Next Detection In: {remaining_time:.1f}
seconds</p>
        """
        timer_placeholder.markdown(timer_html, unsafe_allow_html=True)

    cap.release()

if __name__ == "__main__":
    main()
```

Chapter 5 - Result






Emotion Music Player
http://localhost:8501
RUNNING...
Stop
Deploy
Emotion-Based Music Player
Sad: 0.42
Current Status
Emotion: Sad
Confidence: 0.42
Playing: Billie Eilish - No Time To Die (Lyrics).mp3
Next Detection in: 3.4 seconds

Chapter 6 - Conclusion and Future Work

6.1 Conclusion

The project successfully demonstrates the application of convolutional neural networks for facial emotion recognition using the FER2013 dataset. By leveraging a well-structured CNN architecture with appropriate regularization techniques and hyperparameter tuning, we were able to achieve a high degree of accuracy on the test data. The integration with OpenCV and Streamlit allowed for real-time detection and feedback, while the dynamic music playback provided an engaging extension of the model's utility. This project reflects key machine learning principles such as supervised learning, data preprocessing, model evaluation, and deployment.

6.2 Scope for Future Work

While the current system demonstrates satisfactory real-time performance, several avenues can be explored for further improvement:

- **Model Enhancement**: Implement deeper architectures like ResNet or EfficientNet to improve prediction accuracy and generalization.
- **Data Augmentation**: Incorporate more sophisticated augmentation techniques to reduce overfitting.
- **Multi-modal Emotion Detection**: Integrate speech-based or physiological signals to augment facial emotion recognition.
- **Real-world Deployment**: Extend the application to mobile or edge devices using tools like TensorFlow Lite.
- **Academic Research**: Conduct a comparative study of CNN-based models vs transformer-based models like Vision Transformers (ViT), and analyze their trade-offs in the context of emotion recognition.
- **Conference Publication**: The system, with further enhancements and benchmarking, can be formalized into a research paper for submission to conferences focusing on human-computer interaction, computer vision, or applied machine learning (e.g., ICMI, ICCV Workshops).