Emotion-Based Music Recommender

A Project Report Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

M.Anil Kumar (2010030463)

K.Varun Krishna (2010030490)

D.Dedeepya (2010030526)

K.Sri Teja (2010030530)

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
K L DEEMED TO BE UNIVERSITY
AZIZNAGAR, MOINABAD , HYDERABAD-500 075

MARCH 2024

BONAFIDE CERTIFICATE

This is to certify that the project titled **Emotion-Based Music Recommender** is a bonafide record of the work done by

M. Anil Kumar (2010030463)
K. Varun Krishna (2010030490)
D. Dedeepya (2010030526)
K. Sri Teja (2010030530)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING** of the **K L DEEMED TO BE UNIVERSITY, AZIZNAGAR, MOINABAD, HYDERABAD-500 075**, during the year 2023-2024.

Dr. Lalitha Surya Kumari
Project Guide

Dr. Arpita Gupta
Head of the Department

Project Viva-voce held on ________________________

Internal Examiner

External Examiner

ABSTRACT

Facial emotion recognition is a challenging task within computer vision, garnering significant attention from researchers striving to improve both accuracy and processing efficiency. Numerous techniques have been proposed, ranging from standalone approaches to ensemble-based methods, all aimed at refining emotion classification. However, this particular study is dedicated to elevating accuracy and processing efficiency through the utilization of a single standalone neural network, adept at correctly categorizing human emotions based on facial expressions. Transfer learning serves as a cornerstone, with EfficientNetB3 serving as the chosen base model. Leveraging transfer learning enables the model to inherit knowledge gained from pre-training on a large dataset, subsequently fine-tuning its parameters to suit the specific task of facial emotion recognition. The FER2013 dataset serves as the primary data source, comprising 28,709 training images and 7,178 testing images. One notable challenge encountered during the project stems from the inherent class imbalance within the training data. To address this issue, oversampling techniques are employed, aiming to mitigate the disparities between different emotion classes. Such imbalances typically lead to subpar model performance, underscoring the importance of implementing strategies to rectify them.Upon thorough experimentation and refinement, the proposed model achieves notable results, boasting an accuracy of 93.30

ACKNOWLEDGEMENT

We would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible.

Dr. Lalitha Surya Kumari, our project guide, for helping us and guiding us in the course of this project .

Dr. Arpita Gupta, the Head of the Department, Department of Computer Science and Engineering.

Our internal reviewers, **Dr.Lalitha Surya Kumari** , **Mr.Aftab** , **Dr.Saidireddy M**, **Dr. Annalakshmi** for their insight and advice provided during the review sessions.

We would also like to thank our individual parents and friends for their constant support.

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

1.1 What is Artificial Intelligence(AI)?

Artificial intelligence (AI) refers to the simulation of human intelligence processes by computer systems. These processes include learning (the acquisition of information and rules for using it), reasoning (using rules to reach approximate or definite conclusions), and self-correction. AI is designed to perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation.

AI can be categorized into two types: narrow AI and general AI. Narrow AI, also known as weak AI, is designed to perform a narrow task, such as facial recognition or internet searches. General AI, also known as strong AI or AGI (Artificial General Intelligence), would have the ability to understand, learn, and apply knowledge across different domains, similar to human intelligence.

AI technologies include machine learning, where algorithms are trained on data to recognize patterns and make predictions, and deep learning, which involves artificial neural networks inspired by the structure of the human brain.

AI has applications in various fields, including healthcare, finance, transportation, entertainment, and more. Its development raises ethical, social, and economic considerations, such as job displacement, algorithmic bias, and privacy concerns. Thus, the responsible and ethical deployment of AI is essential for its beneficial integration into society.

1.2 Emotion-Based Music Recommender

An emotion-based music recommender is a system that suggests music to users based on their emotional state or preferences. Instead of solely relying on traditional factors like genre, artist, or popularity, these systems utilize techniques from affective computing and sentiment analysis to understand and respond to the user's emotions.

The recommender system typically begins by collecting data on users' emotions through various means, such as analyzing their past listening habits, tracking physiological responses like heart rate or facial expressions, or directly asking users to input their current mood. This emotional data is then used to generate personalized music recommendations that match or influence the user's emotional state.

For example, if a user indicates they are feeling sad, the recommender might suggest soothing or melancholic songs to match their mood. Conversely, if a user expresses happiness, the system might recommend upbeat or energetic tracks to enhance their positive feelings.

These systems often employ machine learning algorithms to continually refine their recommendations based on user feedback and evolving emotional states. Emotion-based music recommenders aim to enhance user satisfaction and engagement by providing music that resonates with their current emotional needs and preferences, thereby offering a more personalized and enriching listening experience.

1.3 Emotion Recognition using CNN

Here are some key components of an emotion-based music recommender system:

1. Data Collection: The system collects data from various sources, including the user's listening history, social media activity, and sensor data (such as heart rate or facial expressions).

2. Emotion Detection: The system uses machine learning algorithms to analyze the data and detect the user's emotional state. Some systems use facial recognition technology to analyze the user's facial expressions, while others use natural language processing (NLP) to analyze the user's social media activity.

3. Music Recommendation: Once the system has identified the user's emotional state, it recommends music that is likely to match that emotional state. This can involve analyzing the user's listening history and suggesting similar songs or artists, or it can involve recommending songs or playlists that are known to be associated with the user's emotional state.

4. Feedback Loop: As the user interacts with the system, the system learns more about the user's preferences and emotional states. This feedback loop allows the system to continually refine its recommendations and provide more personalized and accurate suggestions over time. There are several benefits of using an emotion-based music recommender system. Firstly, it can help users discover new music that they may not have otherwise found. Secondly, it can enhance the user's listening experience by providing music that matches their current emotional state. Finally, it can improve user engagement and satisfaction with music streaming platforms by providing a more personalized and intuitive experience.

1.4 Problem Statement

Despite the availability of music streaming services, users still struggle to find music that matches their current emotional state, leading to a less satisfying listening experience. Therefore, there is a need for an effective emotion-based music recommender system that can accurately detect and recommend music based on the user's emotional state.

1.5 Objectives

- To detect facial emotions accurately.
- To recognize different facial emotions such as happy, sad, angry, etc.
- To provide real-time feedback on detected facial emotions.
- To work under different lighting conditions and with different skin tones.

1.6 Scope of the Project

It includes research, data collection, system design, emotion recognition model development, recommendation algorithm implementation, user interface development, testing, deployment, maintenance, ethical considerations, and documentation. This involves gathering diverse music datasets, creating machine learning models for emotion recognition, designing intuitive user interfaces, and ensuring system reliability, security, and ethical integrity. Through thorough planning and execution across these areas, the project aims to deliver a personalized and engaging music recommendation experience tailored to users' emotional states and preferences while addressing ethical concerns and providing comprehensive documentation for knowledge sharing.

Chapter 2

Literature Review

2.1 Article 1

Title:-Deep Learning in Music Recommendation Systems.

Authors:-Markus Schedl.

published in:-29 August 2022.

Abstract:-This review article explains particularities of the music domain in RS research. It gives an overview of the state of the art that employs deep learning for music recommendation. The discussion is structured according to the dimensions of neural network type, input data, recommendation approach (content-based filtering, collaborative filtering, or both), and task (standard or sequential music recommendation). In addition, we discuss major challenges faced in MRS, in particular in the context of the current research on deep learning.

2.2 Article 2

Title:- An Emotional Recommender System for Music.

Authors:-Vincenzo Moscato; Antonio Picariello; Giancarlo Sperlí.

published in:-23 september 2023.

Abstract:-In this work, we describe a novel music recommendation technique based on the identification of personality traits, moods, and emotions of a single user, starting from solid psychological observations recognized by the analysis of user behavior within a social environment. In particular, users' personality and mood have been embedded within a content-based filtering approach to obtain more accurate and dynamic results.

2.3 Article 3

Title:- An integrated music recommendation system.

Authors:- Xuan Zhu; Yuan-Yuan Shi; Hyoung-Gook Kim; Ki-Wan Eom.

Published in:- August 2020.

Abstract:- In this paper, an integrated music recommendation system is proposed, which contains the functions of automatic music genre classification, automatic music emotion classification, and music similarity query. A novel tempo feature, named as log-scale modulation frequency coefficients, is presented in this paper

2.4 Article 4

Title:-Facial emotion recognition using convolutional neural networks.

Authors:-Ninad Mehendale

Published in:-18 February 2023.

Abstract:-In this paper, we propose a novel technique called facial emotion recognition using convolutional neural networks (FERC). The FERC is based on two-part convolutional neural network (CNN): The first-part removes the background from the picture, and the second part concentrates on the facial feature vector extraction.

2.5 Article 5

Title:-Facial Emotion Recognition Using Deep Convolutional Neural Network.

Authors:-E. Pranav; Suraj Kamal; C. Satheesh Chandran; M.H. Supriya.

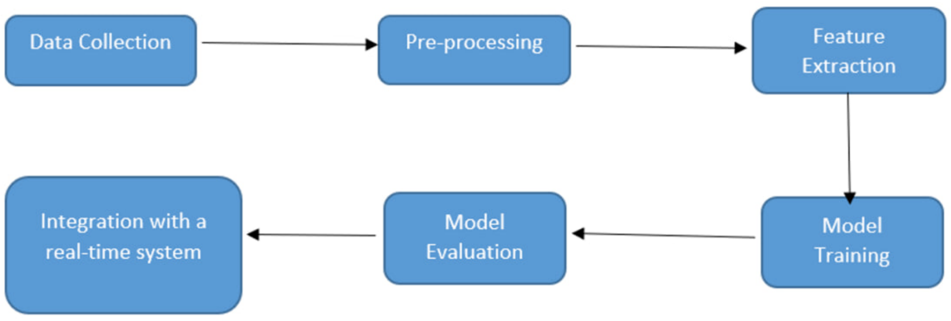
Published in:-04 February 2022.

Abstract:-An emotion recognition system can be built by utilizing the benefits of deep learning and different applications such as feedback analysis, face unlocking etc. can be implemented with good accuracy. The main focus of this work is to create a Deep Convolutional Neural Network (DCNN) model that classifies 5 different human facial emotions. The model is trained, tested and validated using the manually collected image dataset

Title	Author	Published-On	Pro's	Con's
Deep Learning in Music Recommendation Systems.	Markus Schedl.	29 August 2022	The CNN technique is used in this Prediction accuracy is high and having robust working when training example have error in them.	This techniques Require long training time
An Emotional Recommender System for Music.	Vincenzo Moscato, Antonio Picariello, Giancarlo Sperlı	23 september 2023.	It is easy to implement and gives quick result.	Privacy Concerns, Limited Generalization
An integrated music recommendation System.	Xuan Zhu, Yuan-Yuan Shi, Hyoun -Gook Kim, Ki-Wan Eom	August 2020	Enhanced Accessibility Enhanced Security Measures	It requires good online datasets to get access to it
Facial emotion recognition using convolutional neural networks.	Ninad Mehendale	18 February 2023	The CNN techniques have Good potential with ability to detect facial expression.	It Require more time
Facial Emotion Recognition using Deep convolutional neural networks.	E. Pranav, Suraj Kamal C. Satheesh Chandran, M.H. Supriya	04 February 2022	It detect signs of emotional distress or mental health conditions	Lack of Consent and Control, Ethical Implications

Chapter 3

Proposed System



Emotion Recognition using Convolutional Neural Networks (CNN) is a multifaceted and sophisticated project that involves a comprehensive exploration of various aspects, including dataset selection, data preprocessing, CNN architecture design, model training, evaluation, and deployment. Each component contributes significantly to the development of a robust and effective emotion recognition system capable of accurately identifying human emotions from facial expressions. By delving into the intricacies and nuances of each stage, this project aims to provide a comprehensive understanding of the challenges, methodologies, and considerations involved in building such a system.

The first critical step in the project is dataset selection. A well-curated dataset forms the foundation of the model's training and testing phases. In the context of emotion recognition, dataset such as FER2013 is utilized. This dataset contains a diverse range of facial images annotated with corresponding emotion labels, encompassing expressions of happiness, sadness, anger, fear, disgust, and surprise. The inclusion of such diverse expressions ensures that the model learns to recognize a broad spectrum of emotions, enhancing its generalization capabilities in real-world scenarios. Additionally, considerations such as dataset size, annotation quality, and demographic diversity play a crucial role in dataset selection, ensuring the dataset adequately represents the target population and mitigates biases.

Once the dataset is selected, the next step is data preprocessing. Preprocessing techniques are applied to enhance the quality, consistency, and diversity of the dataset, thereby improving the model's performance during training and evaluation. Common preprocessing steps include resizing images to a uniform size, typically ranging from 48x48 to 64x64 pixels, normalization of pixel values to a common scale (e.g., 0 to 1), and data augmentation to increase the variability of the training data. Data augmentation techniques such as rotation, translation, flipping, and zooming help in simulating real-world scenarios and reducing overfitting by exposing the model to a wider range of facial expressions. Additionally, techniques for handling class imbalance, if present, such as oversampling or data augmentation of minority classes, are employed to ensure fair representation of all emotion categories and mitigate biases during training.

Following data preprocessing, the next crucial phase in the project is the design of the CNN architecture tailored specifically for emotion recognition. CNNs are well-suited for image classification tasks due to their ability to automatically learn hierarchical features from raw pixel data. The architecture typically comprises multiple convolutional layers, interspersed with pooling layers to reduce spatial dimensions and extract essential features. Fully connected layers are then used to perform classification based on the learned features. The design of the CNN architecture is guided by considerations such as model complexity, computational efficiency, and the capacity to capture and represent facial features relevant to different emotions. Various architectures, including traditional CNNs, deep residual networks (ResNets), and lightweight architectures optimized for mobile devices, are explored and evaluated to identify the most suitable architecture for the task.

Once the CNN architecture is formulated, the model undergoes intensive training using the preprocessed dataset. The primary objective during training is to minimize a predefined loss function by iteratively adjusting the model's parameters through backpropagation. Common loss functions for multi-class classification tasks, such as categorical cross-entropy and softmax cross-entropy, guide the optimization process. The training process involves feeding batches of labeled images into the model, computing the loss, and updating the model's weights using optimization algorithms such as stochastic gradient descent (SGD), Adam, or RMSprop. Hyperparameter tuning, including adjustments to learning rates, batch sizes, dropout rates, and regularization techniques, is performed to optimize the model's performance and convergence properties.

Throughout the training process, various techniques are employed to monitor and enhance the model's performance. Techniques such as learning rate schedules, early stopping, and model checkpoints help prevent overfitting and ensure convergence to the optimal solution. Moreover, techniques for visualizing model activations, such as gradient-weighted class activation mapping (Grad-CAM), provide insights into the regions of the input image that contribute most to the model's predictions, aiding in model interpretation and debugging.

Once the model is trained, it undergoes thorough evaluation using a separate test dataset to assess its performance metrics. Key evaluation metrics, including accuracy, precision, recall, and F1-score, provide quantitative insights into the model's ability to correctly classify emotions across different facial expressions. Comparative analysis against baseline methods and state-of-the-art approaches offers valuable benchmarks for gauging the model's effectiveness and generalization capabilities. Additionally, qualitative examination of misclassified images enables the identification of potential shortcomings and areas for improvement, guiding subsequent iterations and refinements.

Upon achieving satisfactory performance, the trained CNN model is primed for deployment in real-world applications. Integration into software solutions or embedding into edge devices enables the model to perform real-time emotion recognition tasks in diverse environments. In software applications, live video streams or pre-recorded videos can be analyzed, providing instantaneous feedback on individuals' emotional states. Edge devices, such as smartphones, tablets, or cameras, leverage on-device inference capabilities, ensuring privacy preservation and low-latency emotion recognition without reliance on cloud-based services.

Throughout the project's lifecycle, ethical considerations pertaining to privacy, fairness, and transparency remain paramount. Safeguarding individuals' privacy rights entails adopting measures to protect sensitive information and ensure consent-based data usage. Mitigating biases in the dataset and model architecture fosters fair and equitable representation across diverse demographics, ensuring that the model's predictions are unbiased and inclusive. Moreover, promoting transparency in the model's decision-making process through explainability techniques fosters trust and accountability, empowering users to understand and scrutinize the model's predictions.

Chapter 4
Implementation

4.1 Tools and Technologies used

4.1.1 Data Analysis and Visualization

Data analysis involves exploring and examining datasets to understand their structure, patterns, and relationships. It often includes tasks such as data cleaning, data manipulation, and statistical analysis. Python provides powerful libraries such as Pandas, NumPy, and SciPy that facilitate these tasks. Pandas, in particular, are widely used for data manipulation, transformation, and analysis, offering flexible data structures and data analysis tools. Visualization, on the other hand, involves representing data visually using graphs, charts, and plots. Python provides several libraries for data visualization, the most popular being Matplotlib and Seaborn. Matplotlib offers a wide range of customizable plots, while Seaborn provides higher-level functions for creating aesthetically pleasing statistical visualizations. Additionally, libraries like Plotly and Bokeh enable interactive and web-based visualizations. We have plotted Bar charts and Pie charts for visualizing and better understanding of the dataset we are going to work on.

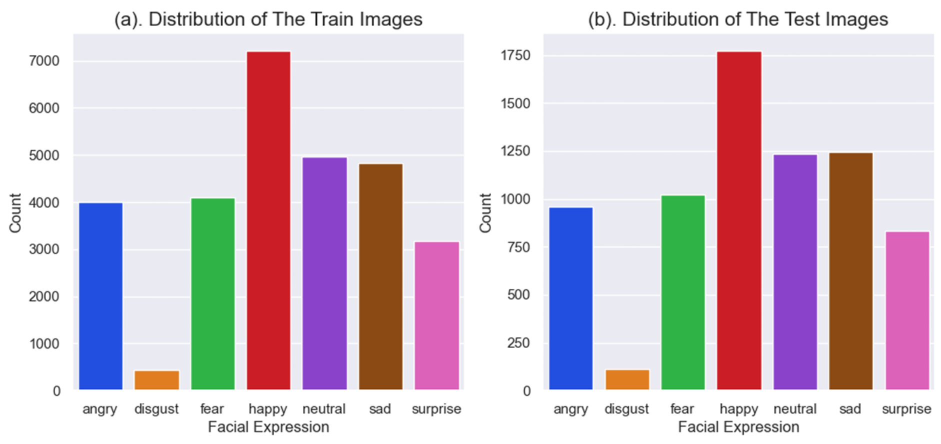


Figure 4.1: Bar charts for Data Visualization

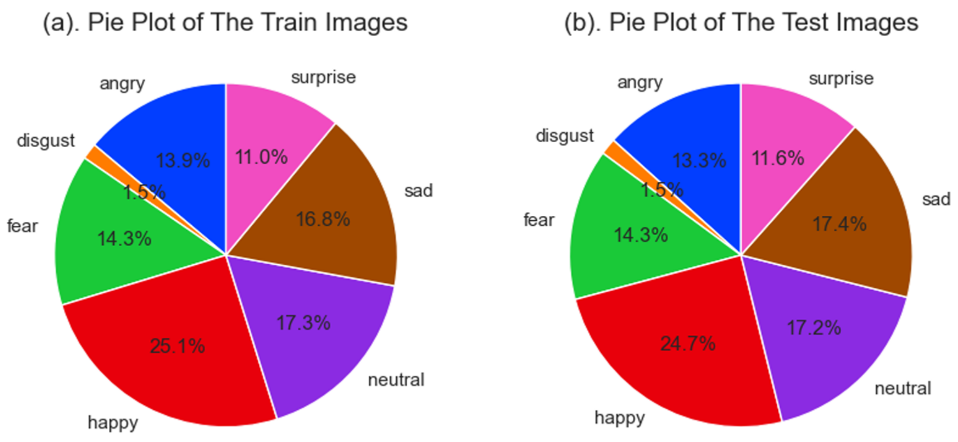


Figure 4.2: Pie charts for Data Visualization

4.1.2 Data Pre-processing

Data pre-processing is a crucial step in the data analysis pipeline. It involves preparing and transforming raw data into a format that is suitable for analysis and modeling. Data pre-processing helps in improving the quality and reliability of the data, removing inconsistencies, handling missing values, and ensuring that the data is in a consistent and usable form. Oversampling the minority class helps to provide more training examples for the model to learn from and can improve the model's ability to accurately classify the minority class. However, it is essential to be cautious with oversampling, as blindly applying it can lead to overfitting or artificially inflating the importance of the minority class. Careful evaluation and consideration of different oversampling techniques are necessary to ensure that the model benefits from the increased representation of the minority class without introducing biases or degrading the performance of the majority class. This project oversamples the minority classes. The goal is to increase the representation of the minority class in the training dataset by creating additional synthetic samples. Overall, our code performs oversampling by duplicating or creating synthetic samples to increase the representation of the minority class in the training dataset.

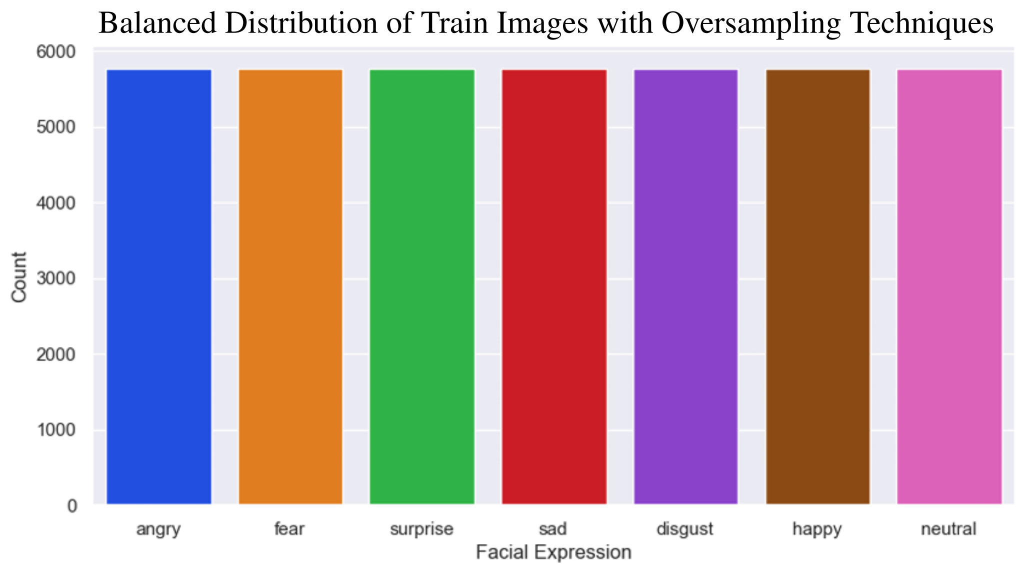


Figure 4.3: Bar chart of Train Data after Oversampling

4.1.3 Model Development

We have applied transfer learning in this project, that is, we have taken a pre-trained model as the base model and upgraded the same. This project makes use of EfficientNetB3. EfficientNetB3 is a specific variant of the EfficientNet architecture. It refers to the third model in the EfficientNet series, which is known for its balance between model size and performance. EfficientNetB3 is designed by scaling the base EfficientNet architecture using the compound scaling method. It has more parameters and is deeper compared to EfficientNetB0 and EfficientNetB1, but it is still more efficient than larger models like EfficientNetB4 or EfficientNetB7. The scaling of EfficientNetB3 involves increasing the depth, width, and resolution of the network. The depth is increased by adding more layers, while the width is increased by widening the channels in each layer. The resolution is increased by using larger input image sizes. EfficientNetB3 has been pre-trained on a large-scale dataset such as ImageNet, allowing it to learn general features from a diverse range of images. It can be fine-tuned or used as a feature extractor for various computer vision tasks, including image classification, object detection, and image segmentation. Compared to earlier versions of EfficientNet, EfficientNetB3 typically offers improved performance on image classification benchmarks while maintaining a relatively efficient model size and computational cost.

The model description is as below:

• We provide EfficientNetB3 as our base model.

• After the base model, we have added the BatchNormalizatoin layer which normalizes the activations of the previous layer. It helps in stabilizing and accelerating the training process.

• After this, we add a Dense layer.

• Now we apply Dropout. This layer randomly sets a fraction of the input units to 0 at each update during training to prevent overfitting.

• Finally, we add one more Dense layer which will be the model's Output layer.

4.1.4 Model Evaluation

We measure the Accuracy and the Loss of our model. Accuracy is a metric that measures the proportion of correctly classified samples out of the total number of samples. It provides an overall assessment of how well the model is performing in terms of correct predictions. Accuracy is often used in classification tasks where the goal is to assign the correct label or class to each input. Loss, also referred to as the loss function or objective function, quantifies the difference between the predicted outputs of the model and the actual ground truth labels. It represents the error or discrepancy between the predicted values and the true values.

4.1.5 Saving the model

We save this model as an HDF5 (.h5) file. These files are often used to save and load trained models. When training a deep learning model, the weights, architecture, and other necessary parameters of the model can be saved in a .h5 file format. This allows you to save the model's state and use it later for prediction, fine-tuning, or sharing with others.

4.1.6 WebApp

The application we described consists of multiple modules that work together to create a web-based system for real-time face detection, track recommendation, and streaming video. Here's a combined explanation of each module's purpose:

i.Spotipy:

• Spotipy is a module used to establish a connection to Spotify and retrieve tracks using the Spotipy wrapper.

• It provides functionality to authenticate with Spotify, access user playlists, search for tracks, and retrieve track information.

ii.haarcascade:

• The haarcascade module is used for face detection.

• It includes pre-trained classifiers (haarcascade XML files) that can detect faces in images or video frames.

• These classifiers utilize Haar-like features and machine learning techniques to identify facial features.

iii.camera.py:

• The camera.py module is responsible for video streaming, frame capturing, prediction, and track recommendation.

• It utilizes the webcam or camera input to capture video frames.

• It performs real-time face detection using the haarcascade module.

• For each detected face, it makes predictions and recommends tracks based on the detected emotion or facial expression.

iv. main.py:

• main.py is the main Flask application file.

• It defines routes and handles HTTP requests from the web page.

• It interacts with the camera.py module to initiate video streaming, capture frames, and receive predictions and track recommendations.

• It sends the processed data to the web page for display and interaction.

v. index.html:

• index.html is an HTML file located in the 'templates' directory.

• It serves as the web page for the application.

• It provides the user interface and displays the video stream, detected faces, predicted emotions, and recommended tracks.

• It includes basic HTML and CSS code for structuring and styling the web page.

vi. utils.py:

• utils.py is a utility module used for video streaming from the web camera.

• It employs threads to enable real-time video capture and processing.

• It assists in achieving smooth and responsive streaming by utilizing concurrent execution.

4.2 Flow of the System

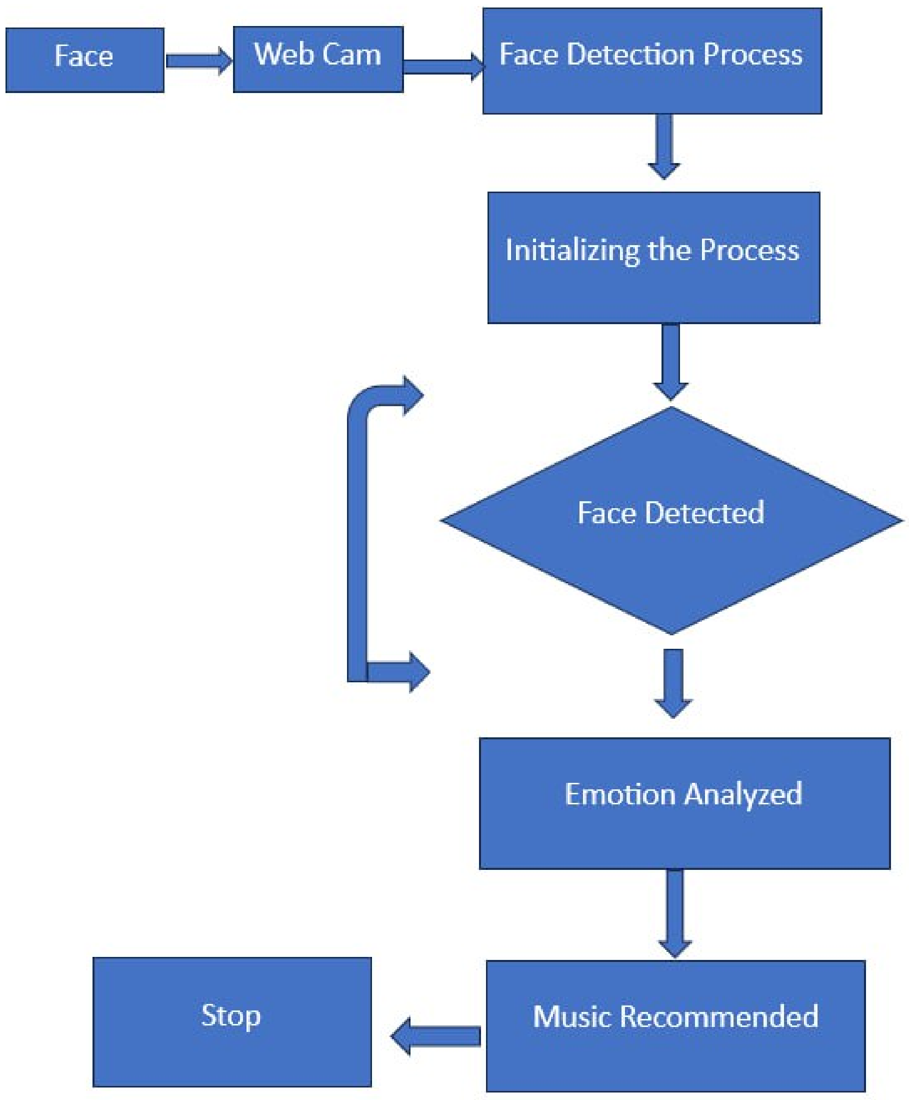


Figure 4.4: Overall architecture of proposed model

Gather a dataset of images containing facial expressions representing various emotions (e.g., happiness, sadness, anger). Preprocess the images to ensure consistency in size, color, and orientation. Common preprocessing steps include resizing, normalization, and grayscale conversion.

Training Data Preparation:

Split the dataset into training, validation, and testing sets. Label each image with the corresponding emotion category.

CNN Model Architecture Design:

Design a CNN architecture suitable for image classification tasks. This typically involves stacking convolutional layers, pooling layers, and fully connected layers.Experiment with different architectures, activation functions, and regularization techniques to optimize performance.

Model Training:

Train the CNN model using the labeled training data. Utilize techniques such as stochastic gradient descent (SGD), backpropagation, and dropout to update the model parameters and minimize the loss function.

Validation and Hyperparameter Tuning:

Validate the trained model using the validation set to assess its performance and generalization ability. Fine-tune hyperparameters such as learning rate, batch size, and layer configurations to improve performance.

Model Evaluation:

Evaluate the trained model using the independent testing set to measure its accuracy, precision, recall, and F1-score. Analyze the confusion matrix to identify common misclassifications and areas for improvement.

Inference and Deployment:

Deploy the trained model for real-time or batch inference on new, unseen images.Integrate the model into an application or system where it can recognize emotions from live video streams or static images.

Monitoring and Maintenance:

Monitor the deployed system for performance degradation or drift over time.Retrain the model periodically with new data to adapt to changes in the input distribution and improve accuracy.

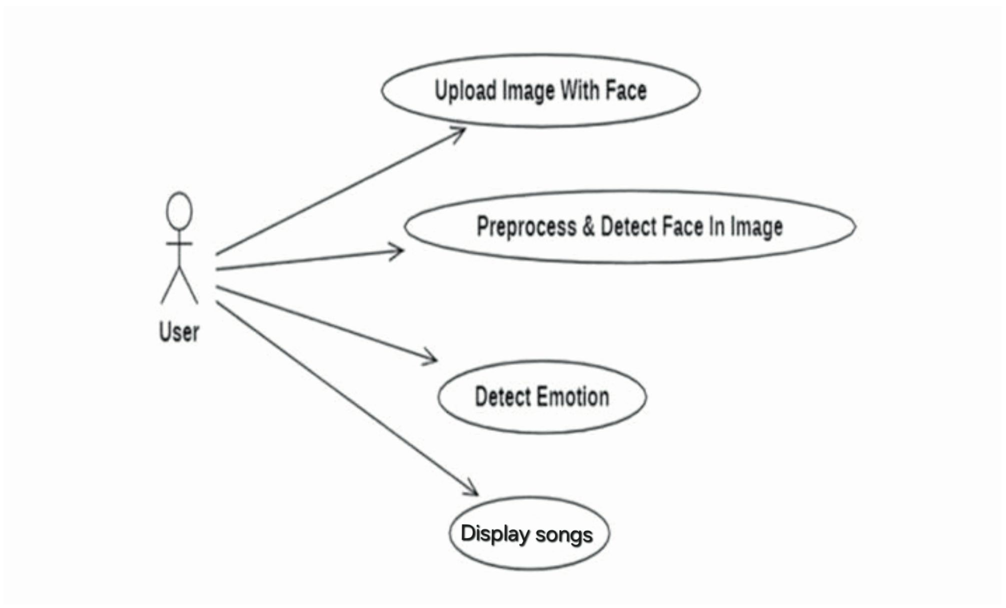


Figure 4.5: Use Case Diagram

Chapter 5

Results and Analysis

5.1 Results

In this project, we used transfer learning by taking EfficientNetB3 as the base model. After training the model we calculate the Accuracy and Loss of our model.

Accuracy	**0.9330**
Loss	**0.3046**

Figure 5.1: Accuracy and Loss values proposed model

We have plotted the Accuracy and Loss graphs for our model:

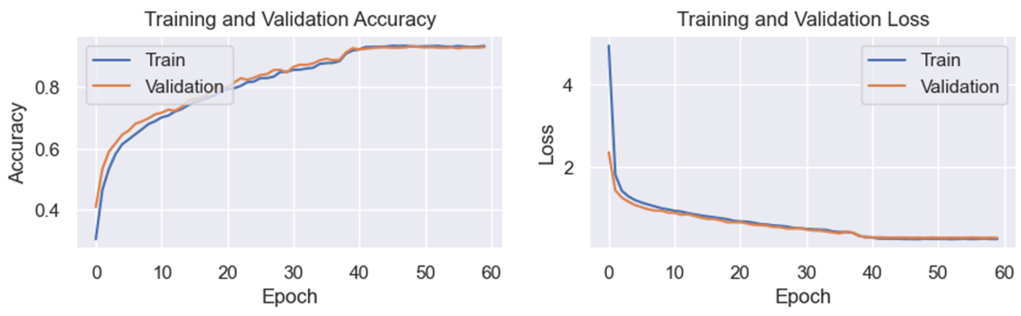


Figure 5.2: Accuracy and Loss Plots

We have also plotted the Confusion matrix:

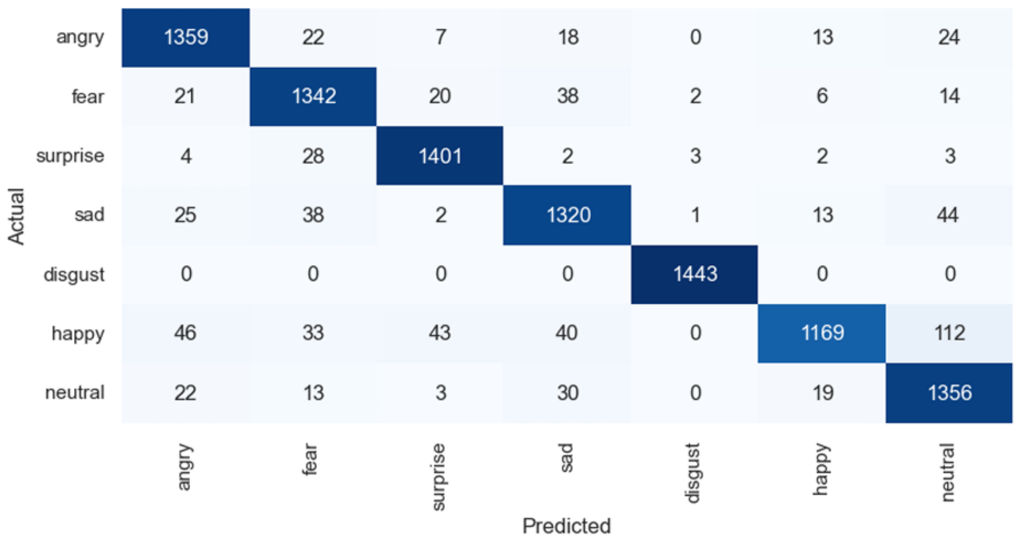


Figure 5.3: Confusion Matrix on Validation Data

5.2 Comparison with existing systems

Emotion recognition, a vital aspect of human-computer interaction and affective computing, has witnessed significant advancements with the advent of Convolutional Neural Networks (CNNs). CNNs have revolutionized the field by offering superior accuracy, scalability, and adaptability compared to traditional methods. In this comparison, we'll delve into the strengths of CNN-based emotion recognition systems, highlighting their advantages over existing approaches.

1. Feature Learning: Traditional methods often rely on handcrafted features extracted from images, such as histograms of oriented gradients (HOG) or local binary patterns (LBP). While these techniques can capture basic facial attributes, they may fail to encapsulate complex spatial relationships and subtle variations in facial expressions.

CNNs, however, employ a data-driven approach to learn hierarchical features directly from raw pixel values. Through convolutional and pooling layers, CNNs automatically extract relevant features at different spatial scales, enabling them to discern intricate patterns in facial images. This end-to-end feature learning process enhances the model's ability to discriminate between different emotions, leading to improved recognition accuracy.

2. Scalability: Handcrafted feature-based methods often require manual intervention for feature engineering and selection. This process becomes cumbersome and impractical when dealing with large and diverse datasets containing a multitude of facial expressions.

CNN-based approaches offer scalability and adaptability to varying data distributions and complexities. As CNNs learn features directly from data, they can efficiently handle large-scale datasets without the need for manual feature engineering. This scalability makes CNNs well-suited for real-world applications where the variability in facial expressions may be extensive.

3. Performance: CNNs have demonstrated superior performance compared to traditional methods in numerous benchmark datasets and real-world scenarios. Their ability to capture both local and global spatial dependencies in facial images enables them to learn discriminative features that distinguish between subtle nuances in facial expressions.

Moreover, CNNs excel in recognizing complex emotions and handling variability in facial expressions across different individuals and demographic groups. This robustness and accuracy make CNN-based emotion recognition systems highly effective in practical applications such as human-computer interaction, affective computing, and social robotics.

4. Adaptability: CNNs offer flexibility and adaptability to changing data distributions and emerging trends in facial expressions. Transfer learning techniques allow pre-trained CNN models to be repurposed for emotion recognition tasks with minimal data requirements, thereby reducing the need for extensive training on new datasets.

Furthermore, CNNs can be fine-tuned or retrained with additional data to improve performance or generalize to new domains. This adaptability enables CNN-based systems to continuously evolve and enhance their capabilities over time, ensuring their relevance and effectiveness in dynamic environments.

5. Real-time Processing: CNN-based emotion recognition systems can achieve real-time performance on modern hardware platforms, making them suitable for applications requiring low-latency inference, such as video conferencing, augmented reality, and interactive gaming.

The inherent parallelism and computational efficiency of CNNs enable rapid processing of facial images, allowing for seamless integration into interactive systems and applications. This real-time processing capability enhances user experience and engagement by enabling responsive and context-aware interactions.

5.3 Limitations and future scope

5.3.1 Limitations

Data Bias and Generalization:

CNNs may struggle to generalize across diverse demographics, cultural backgrounds, and facial expressions not adequately represented in training data. Biases in the training dataset can lead to disparities in performance across different population groups.

Interpretability:

CNNs are often regarded as black-box models, making it challenging to interpret the learned features and understand the decision-making process. This lack of interpretability can hinder trust and transparency in the system, particularly in sensitive applications such as healthcare or criminal justice.

Data Efficiency:

CNNs typically require large amounts of labeled data for training, which may be costly and time-consuming to acquire, particularly for datasets with fine-grained emotion labels or diverse facial expressions. Data scarcity can limit the model's ability to generalize to unseen scenarios.

Robustness to Noise and Variability:

CNNs may exhibit sensitivity to noise, occlusions, lighting conditions, and facial variations, leading to performance degradation in real-world environments where such factors are prevalent. Ensuring robustness to these factors is crucial for deploying CNN-based emotion recognition systems in practical applications.

Real-time Processing Constraints:

Despite their computational efficiency, CNNs may still face challenges in achieving real-time performance on resource-constrained devices or in applications requiring low-latency inference. Optimizing model architectures and inference algorithms is essential for meeting real-time processing requirements.

5.3.2 Future scope

Future research should focus on mitigating biases in emotion recognition datasets and models to ensure fairness and equity across diverse demographic groups. Techniques such as data augmentation, adversarial training, and bias mitigation algorithms can help reduce biases and improve model generalization.

Interpretable Models:

Developing interpretable CNN architectures and explainable AI techniques can enhance transparency and trust in emotion recognition systems. Methods such as attention mechanisms, saliency maps, and model-agnostic explanations can provide insights into the model's decision-making process and facilitate human-computer collaboration.

Semi-supervised and Self-supervised Learning:

Exploring semi-supervised and self-supervised learning approaches can alleviate the data efficiency constraints of CNNs by leveraging unlabeled or weakly labeled data. Techniques such as contrastive learning, generative adversarial networks (GANs), and self-supervised pre-training can enhance model performance with limited labeled data.

Robustness and Adversarial Defense:

Enhancing the robustness of CNNs to noise, occlusions, and adversarial attacks is crucial for deploying emotion recognition systems in real-world scenarios. Research into robust optimization techniques, adversarial training, and defense mechanisms can improve model resilience and reliability.

Efficient Inference and Edge Computing:

Optimizing CNN architectures and inference algorithms for efficient computation and memory usage can enable real-time processing on edge devices with limited resources. Techniques such as model pruning, quantization, and hardware acceleration can reduce computational complexity while maintaining accuracy.

Chapter 6

Conclusion and Recommendations

6.1 Summary of the Project

The emotion-based music recommender system utilizing Convolutional Neural Networks (CNNs) integrates advanced deep learning techniques to enhance the user's music listening experience by aligning it with their emotional state. This system first collects data on user emotions, either through direct input, physiological signals, or facial expressions. The CNN model is then trained on a dataset of images representing various emotional states, learning to recognize patterns and features indicative of different emotions. When a user indicates their current emotional state, the CNN processes this information and generates music recommendations tailored to match or influence the user's mood. These recommendations are based on the learned associations between emotional cues and music preferences, allowing the system to suggest tracks that evoke similar emotions or provide a desired emotional response. The user interface provides seamless interaction, allowing users to explore recommended playlists or adjust preferences based on their feedback. While the system offers personalized recommendations aligned with users' emotions, it also respects privacy and ethical considerations by safeguarding user data and ensuring transparent operation. The future scope of this system includes refining the CNN model for better emotion recognition accuracy, integrating additional data sources for enhanced personalization, and expanding the platform's capabilities to support multi-modal inputs and real-time adaptation to users' changing emotional states. With further advancements in deep learning and affective computing, emotion-based music recommender systems using CNNs have the potential to revolutionize the way people discover and engage with music, fostering deeper emotional connections and enriching their listening experiences.

6.2 Recommendations for future work

several avenues for exploration and enhancement present themselves, each promising to further advance the capabilities and effectiveness of the emotion recognition system. These recommendations encompass both technical and research-oriented aspects, aiming to push the boundaries of the current state-of-the-art in emotion recognition technology.

Dataset Expansion and Diversity: Consider expanding the dataset used for training the CNN model to encompass a broader range of demographics, cultural backgrounds, and facial expressions. This can help improve the model's generalization capabilities and ensure equitable representation across diverse populations. Additionally, collecting data in real-world settings and under different environmental conditions can further enhance the robustness and reliability of the model.

Fine-tuning Model Architectures: Explore alternative CNN architectures and model configurations tailored specifically for emotion recognition tasks. Investigate the efficacy of ensemble methods, hybrid architectures, and attention mechanisms to improve model performance and robustness. Additionally, consider incorporating multimodal information, such as audio and text data, to develop more comprehensive emotion recognition systems capable of capturing subtle cues across different modalities.

Addressing Class Imbalance: Develop advanced techniques for mitigating class imbalance within the dataset, particularly for emotion categories with limited samples. Investigate strategies such as data augmentation, synthetic data generation, and advanced sampling methods to balance the distribution of samples across different emotion classes. This can help alleviate biases and improve the model's ability to recognize underrepresented emotions.

Enhancing Model Interpretability: Explore techniques for enhancing the interpretability and explainability of the CNN model's predictions. Investigate methods such as attention maps, saliency maps, and gradient-based attribution techniques to identify regions of the input image that contribute most to the model's predictions. This can provide valuable insights into the decision-making process of the model and facilitate trust and transparency in its operation.

Cross-Dataset Evaluation and Generalization: Conduct cross-dataset evaluation experiments to assess the generalization capabilities of the trained model across different datasets and scenarios. Evaluate the model's performance on external datasets with varying demographics, image quality, and cultural contexts to gauge its robustness and adaptability in real-world applications. Additionally, investigate domain adaptation techniques to improve the model's performance when deployed in new environments or domains.

Real-Time and Edge Deployment: Investigate methods for optimizing the trained CNN model for real-time inference and deployment on edge devices with limited computational resources. Explore techniques such as model quantization, pruning, and compression to reduce model size and complexity while maintaining performance. Additionally, consider deploying the model on specialized hardware accelerators, such as GPUs or TPUs, to achieve low-latency inference in resource-constrained environments.

Ethical and Societal Implications: Continue to prioritize ethical considerations and societal implications throughout the development and deployment of the emotion recognition system. Address concerns related to privacy, consent, bias, and fairness, ensuring that the system operates responsibly and equitably across diverse populations. Engage with stakeholders to solicit feedback on the ethical dimensions of the technology.

Bibliography

[1] Talegaonkar, Isha and Joshi, Kalyani and Valunj, Shreya and Kohok, Rucha and Kulkarni, Anagha, Real Time Facial Expression Recognition using Deep Learning (May 18, 2019). Proceedings of International Conference on Communication and Information Processing (ICCIP) 2019, http://dx.doi.org/10.2139/ssrn.3421486.

[2] L. Zahara, P. Musa, E. Prasetyo Wibowo, I. Karim and S. Bahri Musa, ”The Facial Emotion Recognition (FER-2013) Dataset for Prediction System of Micro-Expressions Face Using the Convolutional Neural Network (CNN) Algorithm based Raspberry Pi,” 2020 Fifth International Conference on Informatics and Computing (ICIC), Gorontalo, Indonesia, 2020, pp. 1-9, doi: 10.1109/ICIC50835.2020.9288560.

[3] Kusuma Negara, I Gede Putra Jonathan, Jonathan Lim, Andreas. (2020). Emotion Recognition on FER-2013 Face Images Using Fine-Tuned VGG-16. Advances in Science, Technology and Engineering Systems Journal. 5. 315-322. 10.25046/aj050638.

[4] Yousif Khaireddin and Zhoufa Chen. (2021). Facial Emotion Recognition: State of the Art Performance on FER2013. https://doi.org/10.48550/arXiv.2105.03588

[5] Abdellaoui, Benyoussef Moumen, Aniss Idrissi, Younes Remaida, Ahmed. (2021). Training the Fer2013 Dataset with Keras Tuner. 409-412. 10.5220/0010735600003101.

[6] Rahmeh Abou Zafra, Lana Ahmad Abdullah, Rouaa Alaraj, Rasha Albezreh, Tarek Barhoum, Khloud Al Jallad. (2022). An experimental study in Real-time Facial Emotion Recognition on new 3RL dataset. https://doi.org/10.33140/JCTCSR

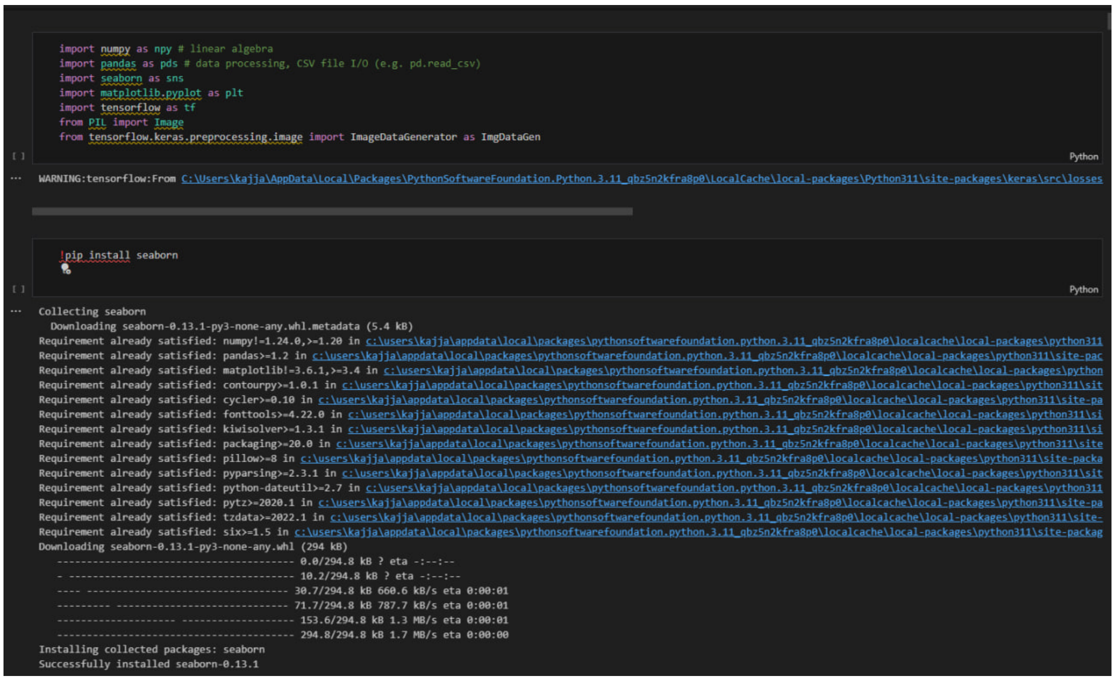
[7] by Mengyu Rao, Ruyi Bao, and Liangshun Dong. (2022). Face Emotion Recognization Using Dataset Augmentation Based on Neural Network. https://doi.org/10.1145/3561518.3561519

[8] Ozioma Collins Oguine, Kanyifeechukwu Jane Oguine, Hashim Ibrahim Bisallah, Daniel Ofuani. (2022). Hybrid Facial Expression Recognition (FER2013) Model for Real-Time Emotion Classification and Prediction. https://doi.org/10.48550/arXiv.2206.09509

Appendices

Appendix A

Source code

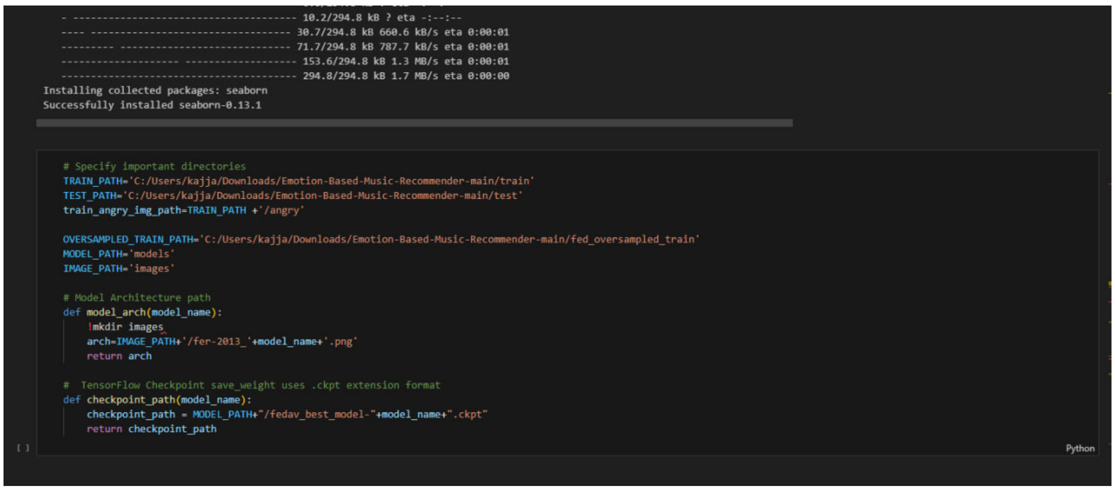


```
import numpy as npy # linear algebra
import pandas as pds # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator as ImgDataGen
```

```
WARNING:tensorflow:From C:\Users\kajja\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\keras\src\losses
```

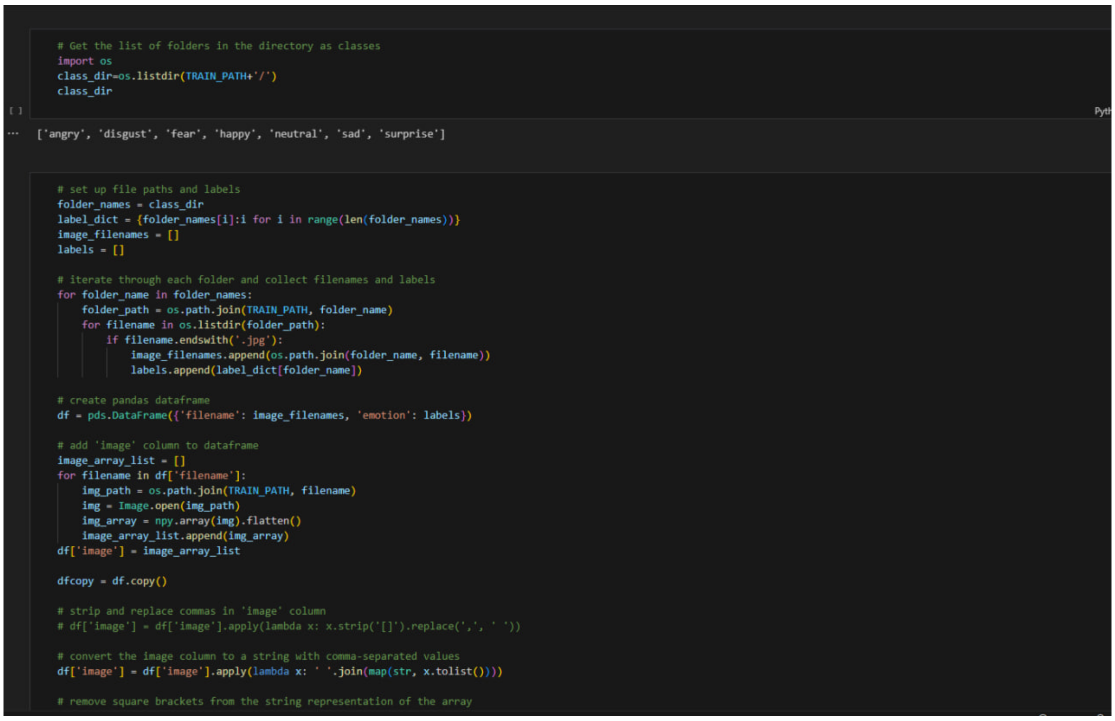
```
!pip install seaborn
```

```
Collecting seaborn
  Downloading seaborn-0.13.1-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311
Requirement already satisfied: pandas>=1.2 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-pac
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python
Requirement already satisfied: contourpy>=1.0.1 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\sit
Requirement already satisfied: cycler>=0.10 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-pa
Requirement already satisfied: fonttools>=4.22.0 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\si
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\si
Requirement already satisfied: packaging>=20.0 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site
Requirement already satisfied: pillow>=8 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packa
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\sit
Requirement already satisfied: python-dateutil>=2.7 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311
Requirement already satisfied: pytz>=2020.1 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-pa
Requirement already satisfied: tzdata>=2022.1 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-
Requirement already satisfied: six>=1.5 in c:\users\kajja\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packag
Downloading seaborn-0.13.1-py3-none-any.whl (294 kB)
   ---------------------------------------- 0.0/294.8 kB ? eta -:--:--
   - -------------------------------------- 10.2/294.8 kB ? eta -:--:--
   ---- ----------------------------------- 30.7/294.8 kB 660.6 kB/s eta 0:00:01
   --------- ------------------------------ 71.7/294.8 kB 787.7 kB/s eta 0:00:01
   ------------------- -------------------- 153.6/294.8 kB 1.3 MB/s eta 0:00:01
   ---------------------------------------- 294.8/294.8 kB 1.7 MB/s eta 0:00:00
Installing collected packages: seaborn
Successfully installed seaborn-0.13.1
```



```
   - -------------------------------------- 10.2/294.8 kB ? eta -:--:--
   ---- ----------------------------------- 30.7/294.8 kB 660.6 kB/s eta 0:00:01
   --------- ------------------------------ 71.7/294.8 kB 787.7 kB/s eta 0:00:01
   ------------------- -------------------- 153.6/294.8 kB 1.3 MB/s eta 0:00:01
   ---------------------------------------- 294.8/294.8 kB 1.7 MB/s eta 0:00:00
Installing collected packages: seaborn
Successfully installed seaborn-0.13.1
```

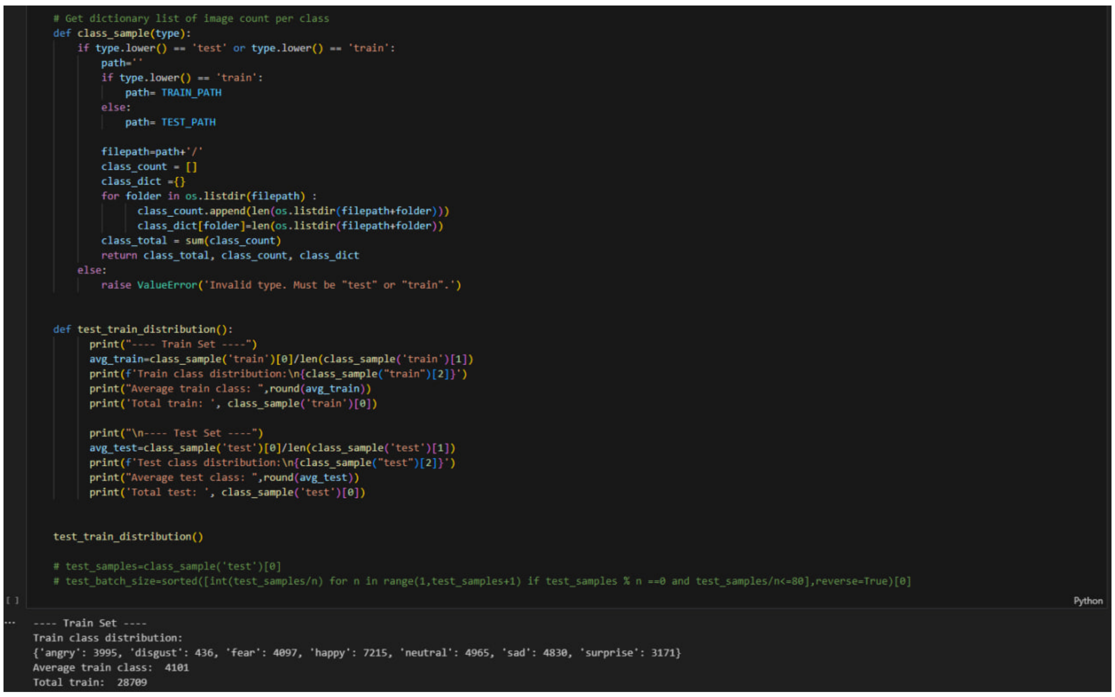
```
# Specify important directories
TRAIN_PATH='C:/Users/kajja/Downloads/Emotion-Based-Music-Recommender-main/train'
TEST_PATH='C:/Users/kajja/Downloads/Emotion-Based-Music-Recommender-main/test'
train_angry_img_path=TRAIN_PATH +'/angry'

OVERSAMPLED_TRAIN_PATH='C:/Users/kajja/Downloads/Emotion-Based-Music-Recommender-main/fed_oversampled_train'
MODEL_PATH='models'
IMAGE_PATH='images'

# Model Architecture path
def model_arch(model_name):
    !mkdir images
    arch=IMAGE_PATH+'/fer-2013_'+model_name+'.png'
    return arch

#  TensorFlow Checkpoint save_weight uses .ckpt extension format
def checkpoint_path(model_name):
    checkpoint_path = MODEL_PATH+"/fedav_best_model-"+model_name+".ckpt"
    return checkpoint_path
```



```
# Get the list of folders in the directory as classes
import os
class_dir=os.listdir(TRAIN_PATH+'/')
class_dir
```

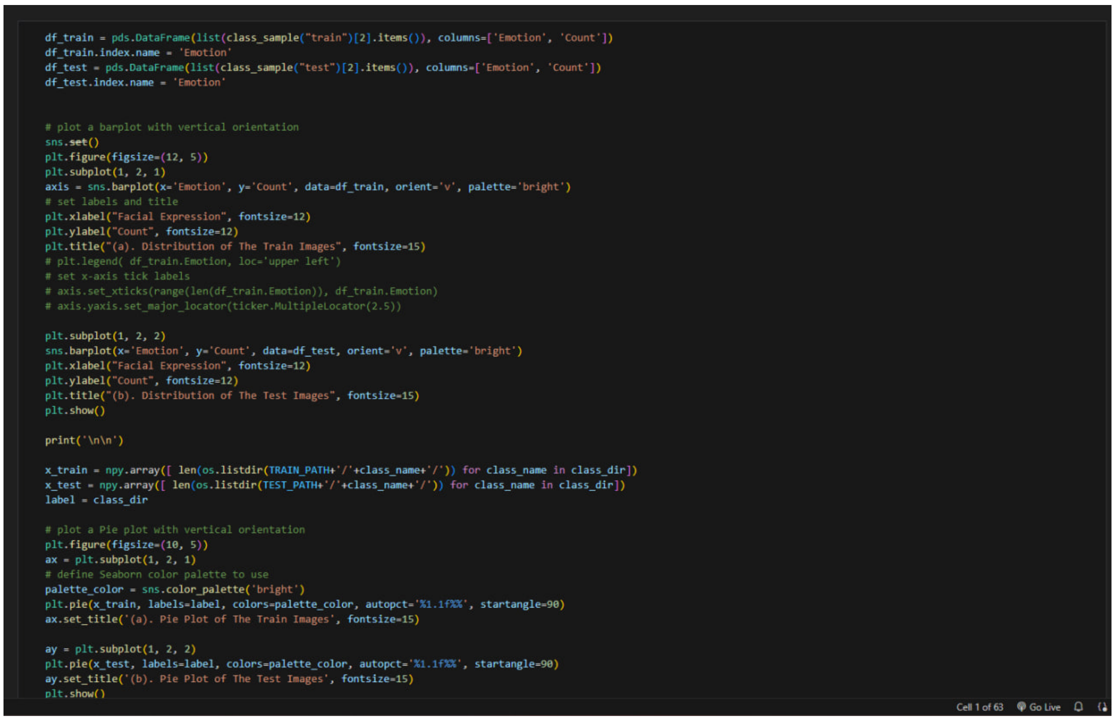
```
['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
```

```
# set up file paths and labels
folder_names = class_dir
label_dict = {folder_names[i]:i for i in range(len(folder_names))}
image_filenames = []
labels = []

# iterate through each folder and collect filenames and labels
for folder_name in folder_names:
    folder_path = os.path.join(TRAIN_PATH, folder_name)
    for filename in os.listdir(folder_path):
        if filename.endswith('.jpg'):
            image_filenames.append(os.path.join(folder_name, filename))
            labels.append(label_dict[folder_name])

# create pandas dataframe
df = pds.DataFrame({'filename': image_filenames, 'emotion': labels})

# add 'image' column to dataframe
image_array_list = []
for filename in df['filename']:
    img_path = os.path.join(TRAIN_PATH, filename)
    img = Image.open(img_path)
    img_array = npy.array(img).flatten()
    image_array_list.append(img_array)
df['image'] = image_array_list

dfcopy = df.copy()

# strip and replace commas in 'image' column
# df['image'] = df['image'].apply(lambda x: x.strip('[]').replace(',', ' '))

# convert the image column to a string with comma-separated values
df['image'] = df['image'].apply(lambda x: ' '.join(map(str, x.tolist())))

# remove square brackets from the string representation of the array
```

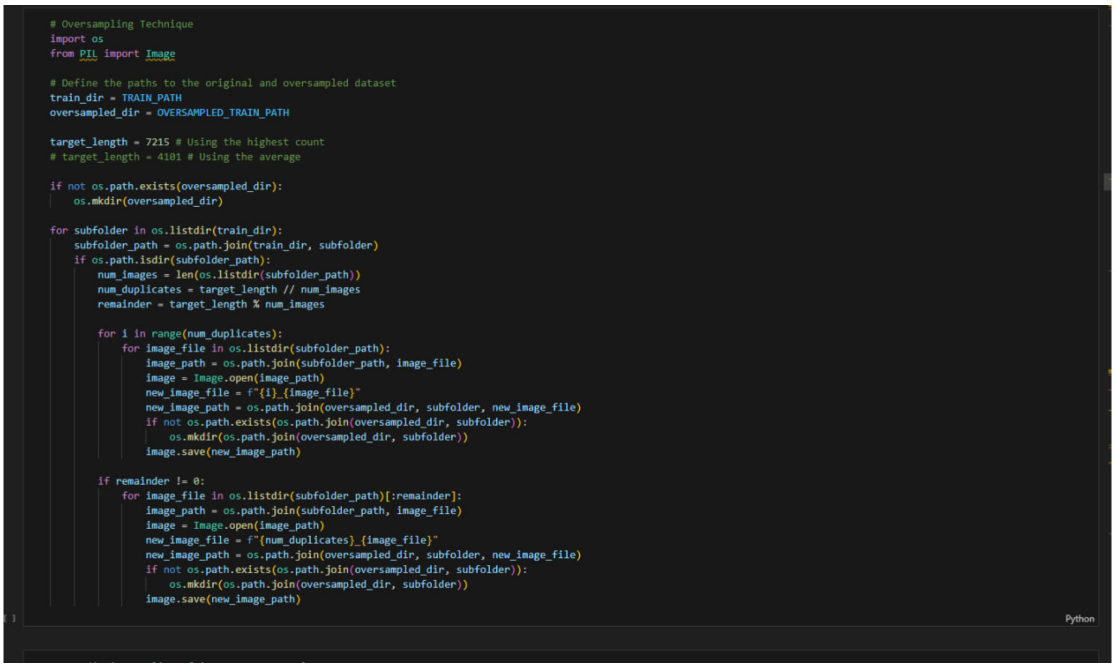


```
# Get dictionary list of image count per class
def class_sample(type):
    if type.lower() == 'test' or type.lower() == 'train':
        path=''
        if type.lower() == 'train':
            path= TRAIN_PATH
        else:
            path= TEST_PATH

        filepath=path+'/'
        class_count = []
        class_dict ={}
        for folder in os.listdir(filepath) :
                class_count.append(len(os.listdir(filepath+folder)))
                class_dict[folder]=len(os.listdir(filepath+folder))
        class_total = sum(class_count)
        return class_total, class_count, class_dict
    else:
        raise ValueError('Invalid type. Must be "test" or "train".')


def test_train_distribution():
      print("---- Train Set ----")
      avg_train=class_sample('train')[0]/len(class_sample('train')[1])
      print(f'Train class distribution:\n{class_sample("train")[2]}')
      print("Average train class: ",round(avg_train))
      print('Total train: ', class_sample('train')[0])

      print("\n---- Test Set ----")
      avg_test=class_sample('test')[0]/len(class_sample('test')[1])
      print(f'Test class distribution:\n{class_sample("test")[2]}')
      print("Average test class: ",round(avg_test))
      print('Total test: ', class_sample('test')[0])


test_train_distribution()

# test_samples=class_sample('test')[0]
# test_batch_size=sorted([int(test_samples/n) for n in range(1,test_samples+1) if test_samples % n ==0 and test_samples/n<=80],reverse=True)[0]
```

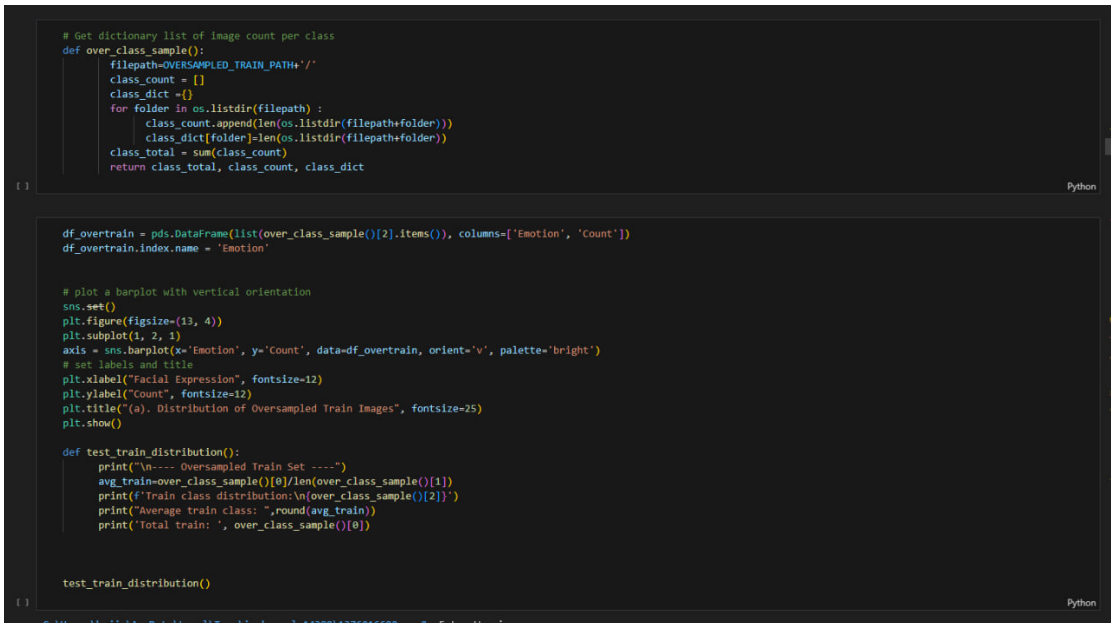
```
---- Train Set ----
Train class distribution:
{'angry': 3995, 'disgust': 436, 'fear': 4097, 'happy': 7215, 'neutral': 4965, 'sad': 4830, 'surprise': 3171}
Average train class:  4101
Total train:  28709
```



```python
df_train = pds.DataFrame(list(class_sample("train")[2].items()), columns=['Emotion', 'Count'])
df_train.index.name = 'Emotion'
df_test = pds.DataFrame(list(class_sample("test")[2].items()), columns=['Emotion', 'Count'])
df_test.index.name = 'Emotion'


# plot a barplot with vertical orientation
sns.set()
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
axis = sns.barplot(x='Emotion', y='Count', data=df_train, orient='v', palette='bright')
# set labels and title
plt.xlabel("Facial Expression", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.title("(a). Distribution of The Train Images", fontsize=15)
# plt.legend( df_train.Emotion, loc='upper left')
# set x-axis tick labels
# axis.set_xticks(range(len(df_train.Emotion)), df_train.Emotion)
# axis.yaxis.set_major_locator(ticker.MultipleLocator(2.5))

plt.subplot(1, 2, 2)
sns.barplot(x='Emotion', y='Count', data=df_test, orient='v', palette='bright')
plt.xlabel("Facial Expression", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.title("(b). Distribution of The Test Images", fontsize=15)
plt.show()

print('\n\n')

x_train = npy.array([ len(os.listdir(TRAIN_PATH+'/'+class_name+'/')) for class_name in class_dir])
x_test = npy.array([ len(os.listdir(TEST_PATH+'/'+class_name+'/')) for class_name in class_dir])
label = class_dir

# plot a Pie plot with vertical orientation
plt.figure(figsize=(10, 5))
ax = plt.subplot(1, 2, 1)
# define Seaborn color palette to use
palette_color = sns.color_palette('bright')
plt.pie(x_train, labels=label, colors=palette_color, autopct='%1.1f%%', startangle=90)
ax.set_title('(a). Pie Plot of The Train Images', fontsize=15)

ay = plt.subplot(1, 2, 2)
plt.pie(x_test, labels=label, colors=palette_color, autopct='%1.1f%%', startangle=90)
ay.set_title('(b). Pie Plot of The Test Images', fontsize=15)
plt.show()
```

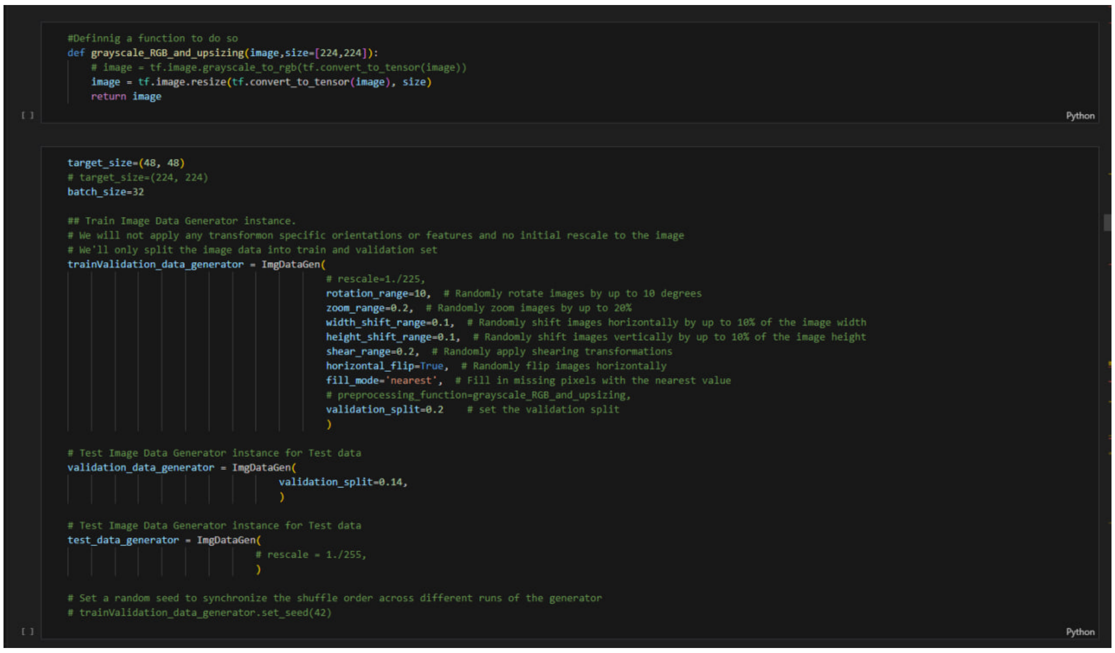


```python
# Oversampling Technique
import os
from PIL import Image

# Define the paths to the original and oversampled dataset
train_dir = TRAIN_PATH
oversampled_dir = OVERSAMPLED_TRAIN_PATH

target_length = 7215 # Using the highest count
# target_length = 4101 # Using the average

if not os.path.exists(oversampled_dir):
    os.mkdir(oversampled_dir)

for subfolder in os.listdir(train_dir):
    subfolder_path = os.path.join(train_dir, subfolder)
    if os.path.isdir(subfolder_path):
        num_images = len(os.listdir(subfolder_path))
        num_duplicates = target_length // num_images
        remainder = target_length % num_images

        for i in range(num_duplicates):
            for image_file in os.listdir(subfolder_path):
                image_path = os.path.join(subfolder_path, image_file)
                image = Image.open(image_path)
                new_image_file = f"{i}_{image_file}"
                new_image_path = os.path.join(oversampled_dir, subfolder, new_image_file)
                if not os.path.exists(os.path.join(oversampled_dir, subfolder)):
                    os.mkdir(os.path.join(oversampled_dir, subfolder))
                image.save(new_image_path)

        if remainder != 0:
            for image_file in os.listdir(subfolder_path)[:remainder]:
                image_path = os.path.join(subfolder_path, image_file)
                image = Image.open(image_path)
                new_image_file = f"{num_duplicates}_{image_file}"
                new_image_path = os.path.join(oversampled_dir, subfolder, new_image_file)
                if not os.path.exists(os.path.join(oversampled_dir, subfolder)):
                    os.mkdir(os.path.join(oversampled_dir, subfolder))
                image.save(new_image_path)
```



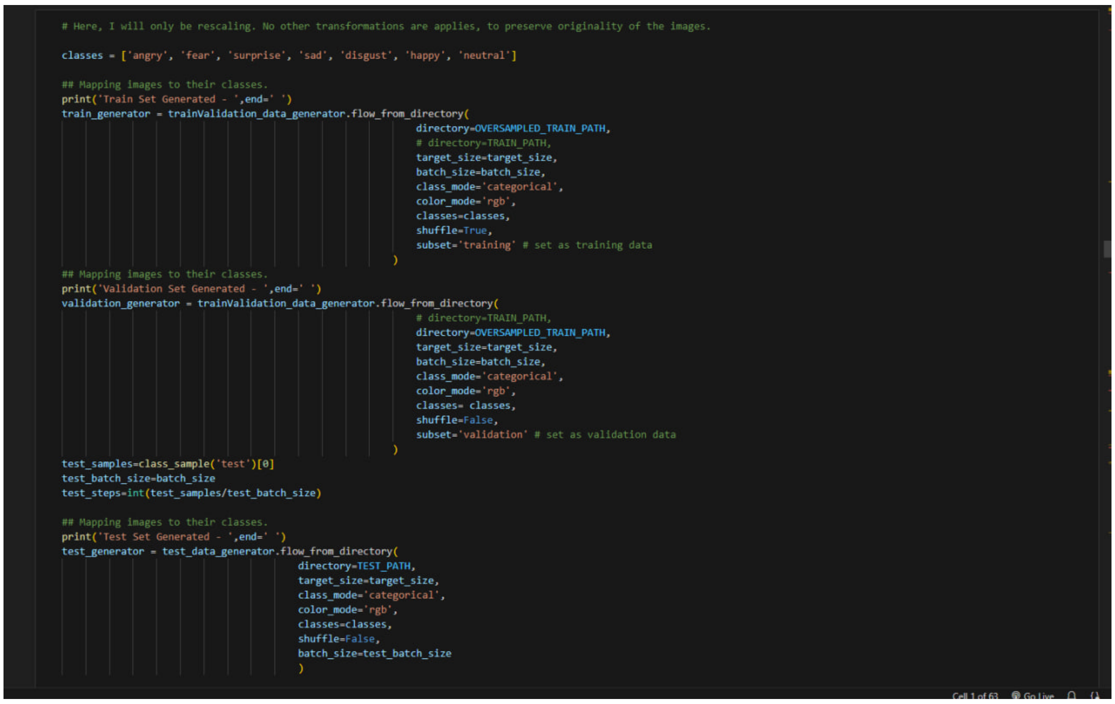
```python
# Get dictionary list of image count per class
def over_class_sample():
        filepath=OVERSAMPLED_TRAIN_PATH+'/'
        class_count = []
        class_dict ={}
        for folder in os.listdir(filepath) :
            class_count.append(len(os.listdir(filepath+folder)))
            class_dict[folder]=len(os.listdir(filepath+folder))
        class_total = sum(class_count)
        return class_total, class_count, class_dict
```

```python
df_overtrain = pds.DataFrame(list(over_class_sample()[2].items()), columns=['Emotion', 'Count'])
df_overtrain.index.name = 'Emotion'


# plot a barplot with vertical orientation
sns.set()
plt.figure(figsize=(13, 4))
plt.subplot(1, 2, 1)
axis = sns.barplot(x='Emotion', y='Count', data=df_overtrain, orient='v', palette='bright')
# set labels and title
plt.xlabel("Facial Expression", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.title("(a). Distribution of Oversampled Train Images", fontsize=25)
plt.show()

def test_train_distribution():
      print("\n---- Oversampled Train Set ----")
      avg_train=over_class_sample()[0]/len(over_class_sample()[1])
      print(f'Train class distribution:\n{over_class_sample()[2]}')
      print("Average train class: ",round(avg_train))
      print('Total train: ', over_class_sample()[0])


test_train_distribution()
```

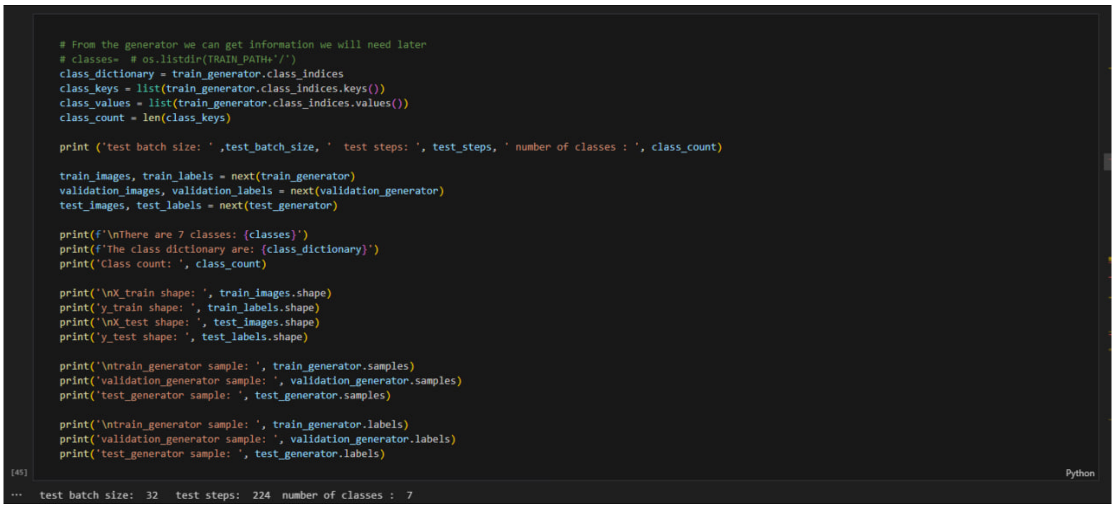


```python
#Definnig a function to do so
def grayscale_RGB_and_upsizing(image,size=[224,224]):
    # image = tf.image.grayscale_to_rgb(tf.convert_to_tensor(image))
    image = tf.image.resize(tf.convert_to_tensor(image), size)
    return image
```

```python
target_size=(48, 48)
# target_size=(224, 224)
batch_size=32

## Train Image Data Generator instance.
# We will not apply any transformon specific orientations or features and no initial rescale to the image
# We'll only split the image data into train and validation set
trainValidation_data_generator = ImgDataGen(
                                            # rescale=1./225,
                                            rotation_range=10,  # Randomly rotate images by up to 10 degrees
                                            zoom_range=0.2,  # Randomly zoom images by up to 20%
                                            width_shift_range=0.1,  # Randomly shift images horizontally by up to 10% of the image width
                                            height_shift_range=0.1,  # Randomly shift images vertically by up to 10% of the image height
                                            shear_range=0.2,  # Randomly apply shearing transformations
                                            horizontal_flip=True,  # Randomly flip images horizontally
                                            fill_mode='nearest',  # Fill in missing pixels with the nearest value
                                            # preprocessing_function=grayscale_RGB_and_upsizing,
                                            validation_split=0.2    # set the validation split
                                            )

# Test Image Data Generator instance for Test data
validation_data_generator = ImgDataGen(
                                    validation_split=0.14,
                                    )

# Test Image Data Generator instance for Test data
test_data_generator = ImgDataGen(
                                # rescale = 1./255,
                                )

# Set a random seed to synchronize the shuffle order across different runs of the generator
# trainValidation_data_generator.set_seed(42)
```

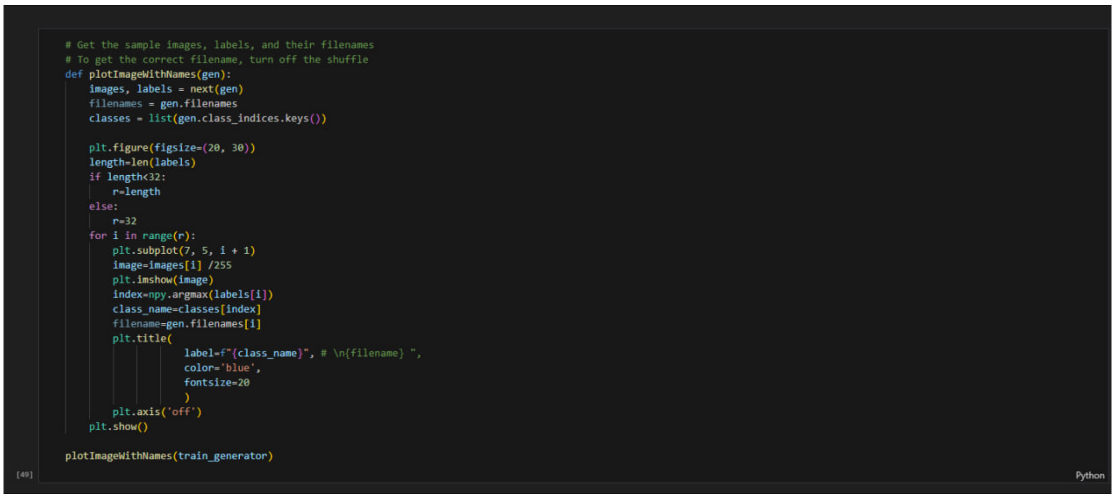


```python
# Here, I will only be rescaling. No other transformations are applies, to preserve originality of the images.

classes = ['angry', 'fear', 'surprise', 'sad', 'disgust', 'happy', 'neutral']

## Mapping images to their classes.
print('Train Set Generated - ',end=' ')
train_generator = trainValidation_data_generator.flow_from_directory(
                                                            directory=OVERSAMPLED_TRAIN_PATH,
                                                            # directory=TRAIN_PATH,
                                                            target_size=target_size,
                                                            batch_size=batch_size,
                                                            class_mode='categorical',
                                                            color_mode='rgb',
                                                            classes=classes,
                                                            shuffle=True,
                                                            subset='training' # set as training data
                                                        )
## Mapping images to their classes.
print('Validation Set Generated - ',end=' ')
validation_generator = trainValidation_data_generator.flow_from_directory(
                                                            # directory=TRAIN_PATH,
                                                            directory=OVERSAMPLED_TRAIN_PATH,
                                                            target_size=target_size,
                                                            batch_size=batch_size,
                                                            class_mode='categorical',
                                                            color_mode='rgb',
                                                            classes= classes,
                                                            shuffle=False,
                                                            subset='validation' # set as validation data
                                                        )
test_samples=class_sample('test')[0]
test_batch_size=batch_size
test_steps=int(test_samples/test_batch_size)

## Mapping images to their classes.
print('Test Set Generated - ',end=' ')
test_generator = test_data_generator.flow_from_directory(
                                        directory=TEST_PATH,
                                        target_size=target_size,
                                        class_mode='categorical',
                                        color_mode='rgb',
                                        classes=classes,
                                        shuffle=False,
                                        batch_size=test_batch_size
                                        )
```

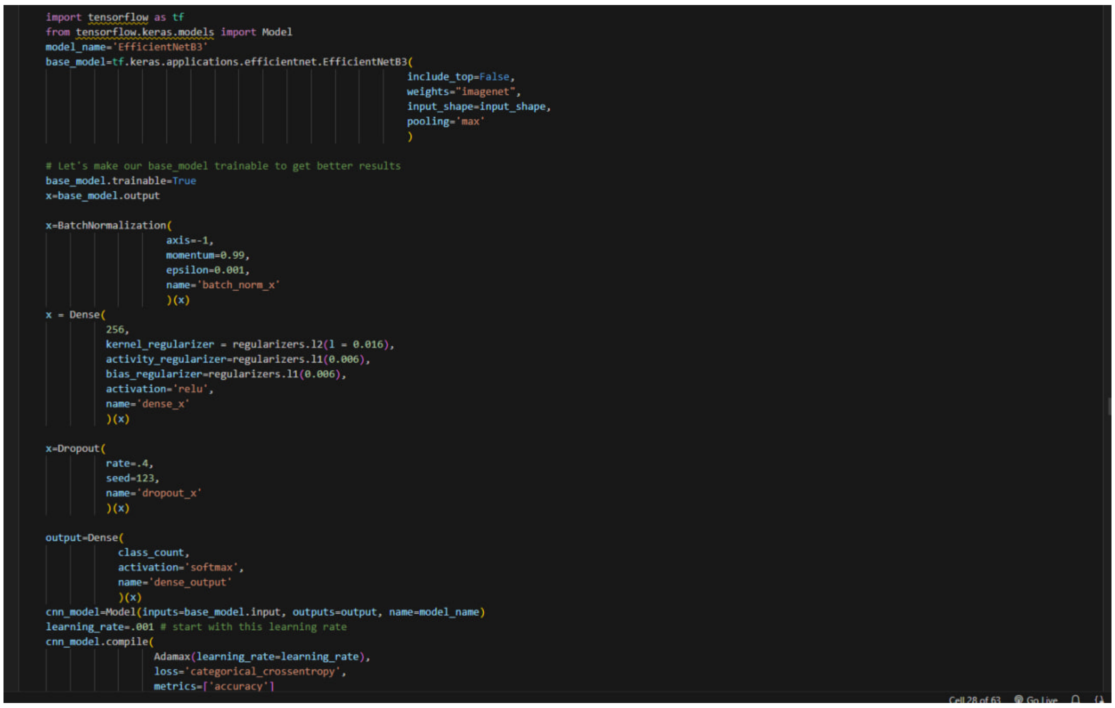


```python
# From the generator we can get information we will need later
# classes=  # os.listdir(TRAIN_PATH+'/')
class_dictionary = train_generator.class_indices
class_keys = list(train_generator.class_indices.keys())
class_values = list(train_generator.class_indices.values())
class_count = len(class_keys)

print ('test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)

train_images, train_labels = next(train_generator)
validation_images, validation_labels = next(validation_generator)
test_images, test_labels = next(test_generator)

print(f'\nThere are 7 classes: {classes}')
print(f'The class dictionary are: {class_dictionary}')
print('Class count: ', class_count)

print('\nX_train shape: ', train_images.shape)
print('y_train shape: ', train_labels.shape)
print('\nX_test shape: ', test_images.shape)
print('y_test shape: ', test_labels.shape)

print('\ntrain_generator sample: ', train_generator.samples)
print('validation_generator sample: ', validation_generator.samples)
print('test_generator sample: ', test_generator.samples)

print('\ntrain_generator sample: ', train_generator.labels)
print('validation_generator sample: ', validation_generator.labels)
print('test_generator sample: ', test_generator.labels)
```

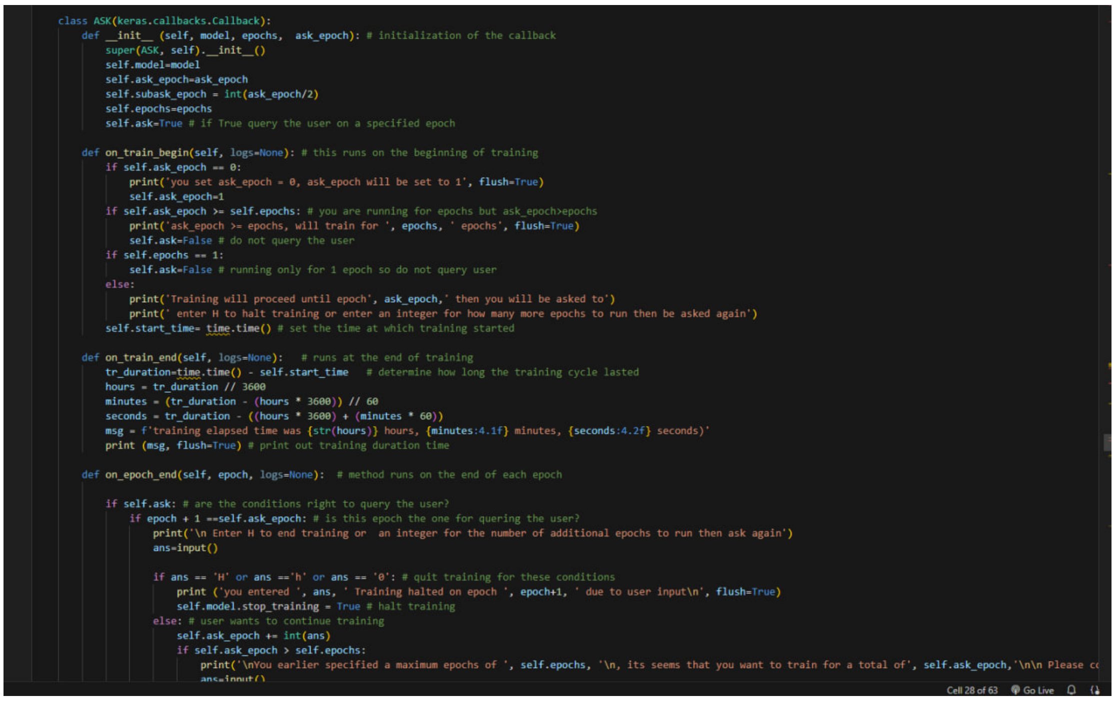
```
test batch size:  32   test steps:  224  number of classes :  7
```



```python
# Get the sample images, labels, and their filenames
# To get the correct filename, turn off the shuffle
def plotImageWithNames(gen):
    images, labels = next(gen)
    filenames = gen.filenames
    classes = list(gen.class_indices.keys())

    plt.figure(figsize=(20, 30))
    length=len(labels)
    if length<32:
        r=length
    else:
        r=32
    for i in range(r):
        plt.subplot(7, 5, i + 1)
        image=images[i] /255
        plt.imshow(image)
        index=npy.argmax(labels[i])
        class_name=classes[index]
        filename=gen.filenames[i]
        plt.title(
                    label=f"{class_name}", # \n{filename} ",
                    color='blue',
                    fontsize=20
                    )
        plt.axis('off')
    plt.show()

plotImageWithNames(train_generator)
```



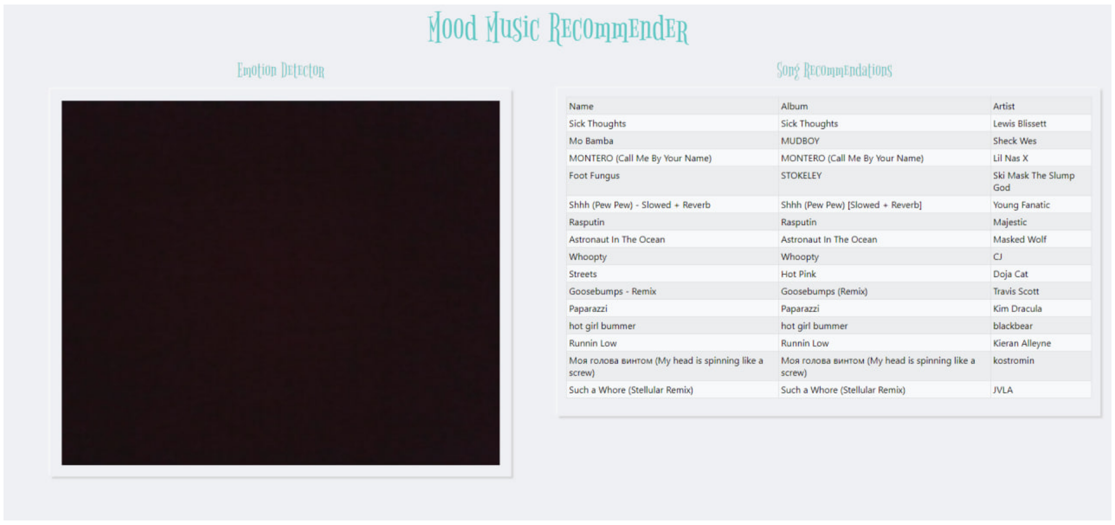
```
import tensorflow as tf
from tensorflow.keras.models import Model
model_name='EfficientNetB3'
base_model=tf.keras.applications.efficientnet.EfficientNetB3(
                                                            include_top=False,
                                                            weights="imagenet",
                                                            input_shape=input_shape,
                                                            pooling='max'
                                                            )

# Let's make our base_model trainable to get better results
base_model.trainable=True
x=base_model.output

x=BatchNormalization(
                    axis=-1,
                    momentum=0.99,
                    epsilon=0.001,
                    name='batch_norm_x'
                    )(x)
x = Dense(
          256,
          kernel_regularizer = regularizers.l2(l = 0.016),
          activity_regularizer=regularizers.l1(0.006),
          bias_regularizer=regularizers.l1(0.006),
          activation='relu',
          name='dense_x'
          )(x)

x=Dropout(
          rate=.4,
          seed=123,
          name='dropout_x'
          )(x)

output=Dense(
            class_count,
            activation='softmax',
            name='dense_output'
            )(x)
cnn_model=Model(inputs=base_model.input, outputs=output, name=model_name)
learning_rate=.001 # start with this learning rate
cnn_model.compile(
                  Adamax(learning_rate=learning_rate),
                  loss='categorical_crossentropy',
                  metrics=['accuracy']
```



```
class ASK(keras.callbacks.Callback):
    def __init__ (self, model, epochs,  ask_epoch): # initialization of the callback
        super(ASK, self).__init__()
        self.model=model
        self.ask_epoch=ask_epoch
        self.subask_epoch = int(ask_epoch/2)
        self.epochs=epochs
        self.ask=True # if True query the user on a specified epoch

    def on_train_begin(self, logs=None): # this runs on the beginning of training
        if self.ask_epoch == 0:
            print('you set ask_epoch = 0, ask_epoch will be set to 1', flush=True)
            self.ask_epoch=1
        if self.ask_epoch >= self.epochs: # you are running for epochs but ask_epoch>epochs
            print('ask_epoch >= epochs, will train for ', epochs, ' epochs', flush=True)
            self.ask=False # do not query the user
        if self.epochs == 1:
            self.ask=False # running only for 1 epoch so do not query user
        else:
            print('Training will proceed until epoch', ask_epoch,' then you will be asked to')
            print(' enter H to halt training or enter an integer for how many more epochs to run then be asked again')
        self.start_time= time.time() # set the time at which training started

    def on_train_end(self, logs=None):   # runs at the end of training
        tr_duration=time.time() - self.start_time   # determine how long the training cycle lasted
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} minutes, {seconds:4.2f} seconds)'
        print (msg, flush=True) # print out training duration time

    def on_epoch_end(self, epoch, logs=None):  # method runs on the end of each epoch

        if self.ask: # are the conditions right to query the user?
            if epoch + 1 ==self.ask_epoch: # is this epoch the one for quering the user?
                print('\n Enter H to end training or  an integer for the number of additional epochs to run then ask again')
                ans=input()

                if ans == 'H' or ans =='h' or ans == '0': # quit training for these conditions
                    print ('you entered ', ans, ' Training halted on epoch ', epoch+1, ' due to user input\n', flush=True)
                    self.model.stop_training = True # halt training
                else: # user wants to continue training
                    self.ask_epoch += int(ans)
                    if self.ask_epoch > self.epochs:
                        print('\nYou earlier specified a maximum epochs of ', self.epochs, '\n, its seems that you want to train for a total of', self.ask_epoch,'\n\n Please cc
                        ans=input()
```

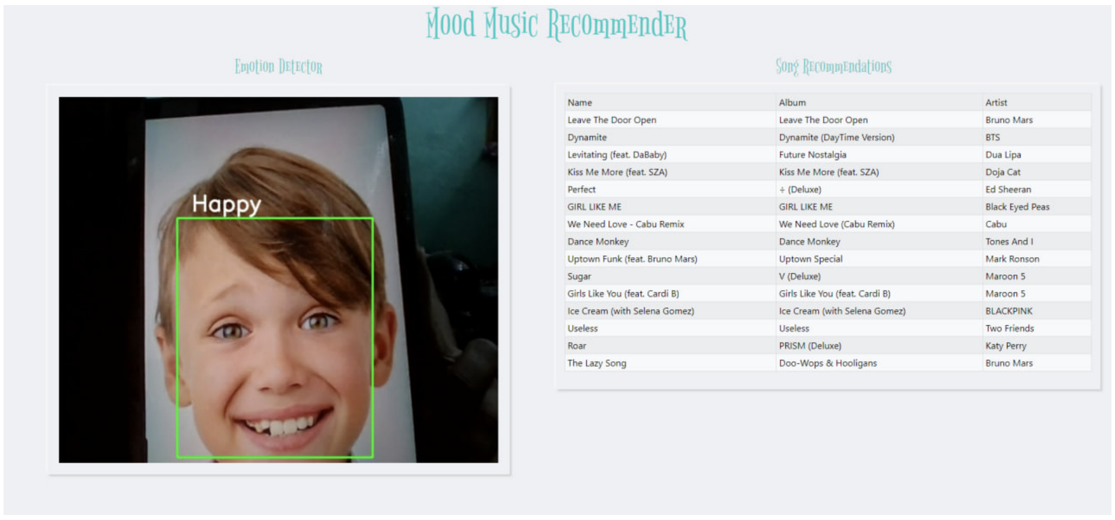
Appendix B

Screen shots



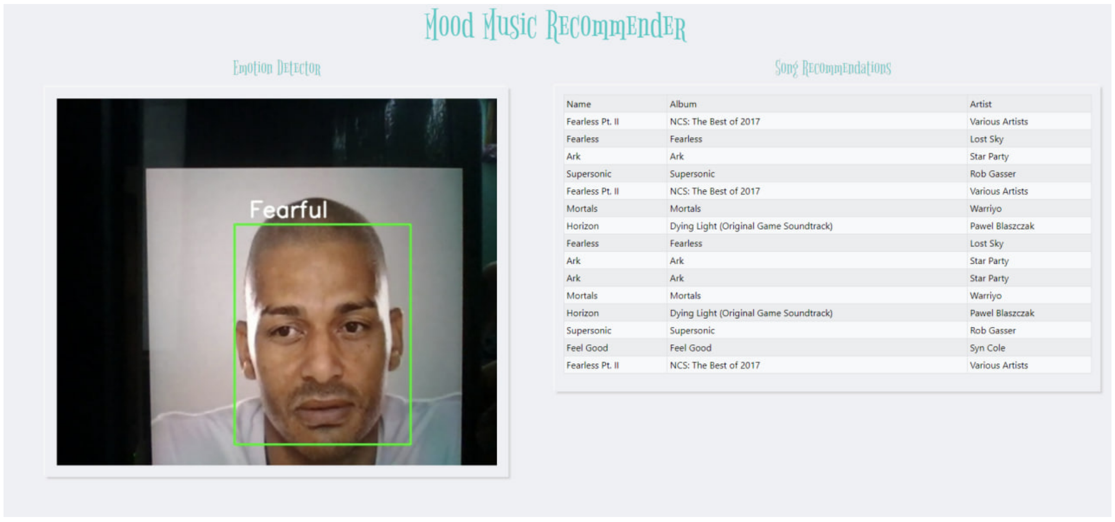
Name	Album	Artist
Sick Thoughts	Sick Thoughts	Lewis Blissett
Mo Bamba	MUDBOY	Sheck Wes
MONTERO (Call Me By Your Name)	MONTERO (Call Me By Your Name)	Lil Nas X
Foot Fungus	STOKELEY	Ski Mask The Slump God
Shhh (Pew Pew) - Slowed + Reverb	Shhh (Pew Pew) [Slowed + Reverb]	Young Fanatic
Rasputin	Rasputin	Majestic
Astronaut In The Ocean	Astronaut In The Ocean	Masked Wolf
Whoopty	Whoopty	CJ
Streets	Hot Pink	Doja Cat
Goosebumps - Remix	Goosebumps (Remix)	Travis Scott
Paparazzi	Paparazzi	Kim Dracula
hot girl bummer	hot girl bummer	blackbear
Runnin Low	Runnin Low	Kieran Alleyne
Моя голова винтом (My head is spinning like a screw)	Моя голова винтом (My head is spinning like a screw)	kostromin
Such a Whore (Stellular Remix)	Such a Whore (Stellular Remix)	JVLA

Figure B.1: Application



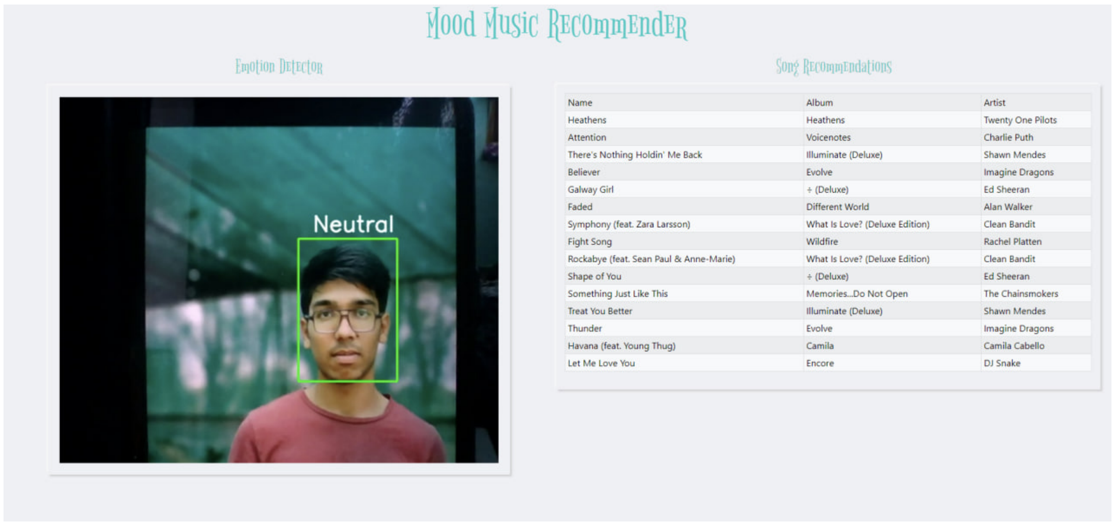
Name	Album	Artist
Leave The Door Open	Leave The Door Open	Bruno Mars
Dynamite	Dynamite (DayTime Version)	BTS
Levitating (feat. DaBaby)	Future Nostalgia	Dua Lipa
Kiss Me More (feat. SZA)	Kiss Me More (feat. SZA)	Doja Cat
Perfect	÷ (Deluxe)	Ed Sheeran
GIRL LIKE ME	GIRL LIKE ME	Black Eyed Peas
We Need Love - Cabu Remix	We Need Love (Cabu Remix)	Cabu
Dance Monkey	Dance Monkey	Tones And I
Uptown Funk (feat. Bruno Mars)	Uptown Special	Mark Ronson
Sugar	V (Deluxe)	Maroon 5
Girls Like You (feat. Cardi B)	Girls Like You (feat. Cardi B)	Maroon 5
Ice Cream (with Selena Gomez)	Ice Cream (with Selena Gomez)	BLACKPINK
Useless	Useless	Two Friends
Roar	PRISM (Deluxe)	Katy Perry
The Lazy Song	Doo-Wops & Hooligans	Bruno Mars

Figure B.2: Happy Expression



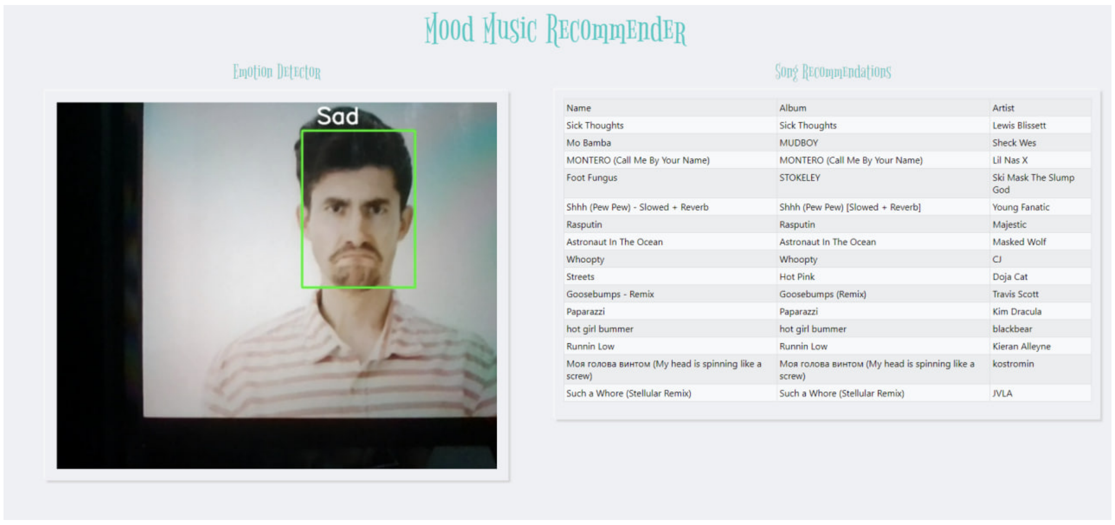
Name	Album	Artist
Fearless Pt. II	NCS: The Best of 2017	Various Artists
Fearless	Fearless	Lost Sky
Ark	Ark	Star Party
Supersonic	Supersonic	Rob Gasser
Fearless Pt. II	NCS: The Best of 2017	Various Artists
Mortals	Mortals	Warriyo
Horizon	Dying Light (Original Game Soundtrack)	Pawel Blaszczak
Fearless	Fearless	Lost Sky
Ark	Ark	Star Party
Ark	Ark	Star Party
Mortals	Mortals	Warriyo
Horizon	Dying Light (Original Game Soundtrack)	Pawel Blaszczak
Supersonic	Supersonic	Rob Gasser
Feel Good	Feel Good	Syn Cole
Fearless Pt. II	NCS: The Best of 2017	Various Artists

Figure B.3: Fearful Expression



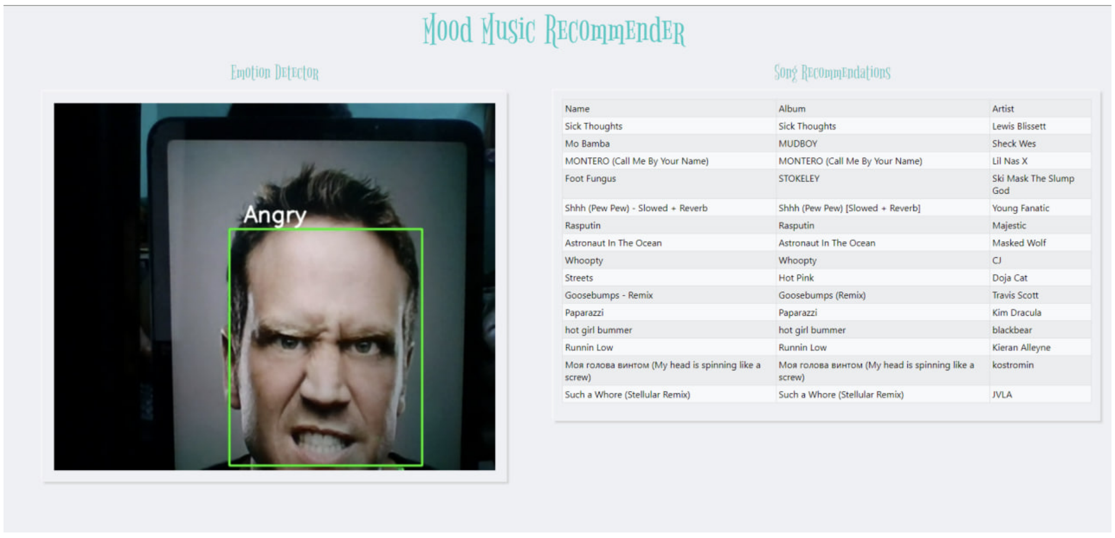
Name	Album	Artist
Heathens	Heathens	Twenty One Pilots
Attention	Voicenotes	Charlie Puth
There's Nothing Holdin' Me Back	Illuminate (Deluxe)	Shawn Mendes
Believer	Evolve	Imagine Dragons
Galway Girl	÷ (Deluxe)	Ed Sheeran
Faded	Different World	Alan Walker
Symphony (feat. Zara Larsson)	What Is Love? (Deluxe Edition)	Clean Bandit
Fight Song	Wildfire	Rachel Platten
Rockabye (feat. Sean Paul & Anne-Marie)	What Is Love? (Deluxe Edition)	Clean Bandit
Shape of You	÷ (Deluxe)	Ed Sheeran
Something Just Like This	Memories...Do Not Open	The Chainsmokers
Treat You Better	Illuminate (Deluxe)	Shawn Mendes
Thunder	Evolve	Imagine Dragons
Havana (feat. Young Thug)	Camila	Camila Cabello
Let Me Love You	Encore	DJ Snake

Figure B.4: Neutral Expression



Name	Album	Artist
Sick Thoughts	Sick Thoughts	Lewis Blissett
Mo Bamba	MUDBOY	Sheck Wes
MONTERO (Call Me By Your Name)	MONTERO (Call Me By Your Name)	Lil Nas X
Foot Fungus	STOKELEY	Ski Mask The Slump God
Shhh (Pew Pew) - Slowed + Reverb	Shhh (Pew Pew) [Slowed + Reverb]	Young Fanatic
Rasputin	Rasputin	Majestic
Astronaut In The Ocean	Astronaut In The Ocean	Masked Wolf
Whoopty	Whoopty	CJ
Streets	Hot Pink	Doja Cat
Goosebumps - Remix	Goosebumps (Remix)	Travis Scott
Paparazzi	Paparazzi	Kim Dracula
hot girl bummer	hot girl bummer	blackbear
Runnin Low	Runnin Low	Kieran Alleyne
Моя голова винтом (My head is spinning like a screw)	Моя голова винтом (My head is spinning like a screw)	kostromin
Such a Whore (Stellular Remix)	Such a Whore (Stellular Remix)	JVLA

Figure B.5: Sad Expression



Name	Album	Artist
Sick Thoughts	Sick Thoughts	Lewis Blissett
Mo Bamba	MUDBOY	Sheck Wes
MONTERO (Call Me By Your Name)	MONTERO (Call Me By Your Name)	Lil Nas X
Foot Fungus	STOKELEY	Ski Mask The Slump God
Shhh (Pew Pew) - Slowed + Reverb	Shhh (Pew Pew) [Slowed + Reverb]	Young Fanatic
Rasputin	Rasputin	Majestic
Astronaut In The Ocean	Astronaut In The Ocean	Masked Wolf
Whoopty	Whoopty	CJ
Streets	Hot Pink	Doja Cat
Goosebumps - Remix	Goosebumps (Remix)	Travis Scott
Paparazzi	Paparazzi	Kim Dracula
hot girl bummer	hot girl bummer	blackbear
Runnin Low	Runnin Low	Kieran Alleyne
Моя голова винтом (My head is spinning like a screw)	Моя голова винтом (My head is spinning like a screw)	kostromin
Such a Whore (Stellular Remix)	Such a Whore (Stellular Remix)	JVLA

Figure B.6: Angry Expression