Music Recommendation Using Emotion Detection from Facial Expression in Portable Devices

Shehu Adamu[1]
Department Computer Science
Federal Polytechnic Bali
Taraba, Nigeria
shehuadamushehu@gmail.com

Oreofe J. OLURIN[2]
Department of Computer Science
University of Bradford
Bradford, United Kingdom
ojolurin@bradford.ac.uk

Daniel Adeboye[3]
Data & AI Team
Lights On Heights
Lagos, Nigeria
daniela@lightsonheights.com

Garba Aliyu[4]
Department of Computer Science
Ahmadu Bello University
Zaria, Nigeria
algarba@abu.edu.ng

Auwal Nataala[5]
Department of Computer Science
Federal Polytechnic Bali
Taraba, Nigeria
auwalkude@gmail.com

Augustine N. Egere[6]
Department Computer science
Federal Polytechnic Bali
Taraba, Nigeria
austinendudi@yahoo.com

Hashidu Baba[7]
Department Computer Science
Federal Polytechnic Bali
Taraba, Nigeria
hashidubaba88@gmail.com

Bashir Timizi Yusuf[8]
Department of Computer Science
Federal Polytechnic Bali
Taraba, Nigeria
bashirtirmizi@gmail.com

***Abstract*— This research presents a novel approach to music recommendation by utilizing artificial intelligence for real-time emotion detection from facial expressions, integrated into a portable device. The study acknowledges the profound connection between music and human emotions, highlighting the significance of employing facial expression analysis to enhance music recommendation systems. Leveraging the Yolo-V3 architecture as an object detection layer, the model is trained on a comprehensive facial expression database (RafDB datasets) to accurately detect three primary emotional classes: happiness, sadness, and neutrality. The resultant model achieves a commendable loss of 1.69 after a rigorous training regimen spanning 100 epochs. The research culminates in the development of a music recommendation API that processes facial expression images through a POST request, providing emotion class, confidence scores, and bounding box details. This valuable output directs the selection of music tailored to the detected emotion, seamlessly interfacing with Spotify's Playlist API to retrieve pertinent songs and metadata. The API's integration into a portable ESP32 device, complemented by RGB LED color displays corresponding to detected emotions, demonstrates a practical implementation of the model. The training of the emotion detection model utilized the PyTorch library, while the API was constructed using the Python FastAPI web framework. By presenting an effective marriage of emotion detection and music recommendation, this research illuminates a promising trajectory for the fusion of artificial intelligence, emotional awareness, and music, envisioning a more personalized and emotionally engaging user experience.**

Keywords*— *Music recommendation; artificial intelligence, object detection, Yolo-V3, ESP32, API, emotion-based model, music, IoT Device.

I. INTRODUCTION

Advancements in computational speed and media content have led to a surge in music demand, making it challenging for users to find music that matches their emotions. Artificial intelligence techniques, especially deep learning, are being used to recommend music based on user preferences. Various methods, including acoustic music, metadata, social tagging, and similarity networks, have gained popularity for music recommendation. Humming and collaborative filtering are also employed for personalized services. This paper proposes a novel music recommendation approach using facial expression analysis to detect user emotions in real-time. Emotions detected from facial images are used to select appropriate songs from Spotify playlists representing happiness, sadness, and neutrality. The goal is to create an automated way to provide users with music that resonates with their emotional state by capturing facial expressions using portable devices. The research aims to answer questions related to identifying emotions from facial expressions and the role of emotions in music recommendation for real-time use

Some of the work that has gained popularity ranges from recommendations using acoustic music to metadata and online text related to the artist. Some use content-based methods, social tagging, and similarity networks. Currently, several applications use humming from the users to query the right music, and some online platforms, such as All Music Guide, use similar artist recommendations. This platform uses collaborative filtering to recommend music to customers, utilizing the ability of machine learning to learn from data. This improvement helps customers derive custom services that best suit their needs. [1] uses unsupervised learning to predict the outcome of videos to be able to improve the searching and filtering of video content. In [2] collaborative filtering is also used to streamline music, which users will most certainly be interested in.

However, most of this work focuses on recommendations from the user's query which also leaves a lot of room for improvement. Therefore, we proposed a new means of music recommendation which is based on detecting users' emotions from their facial expressions and then using the results to select the right song from a music API.

Since music has some strong similarities with our emotional state and could be seen as an expression of one's emotional state. We can use the emotion detected from an image of their facial expression to recommend music, that a user will be more interested in listening to. This could be used in portable devices in real-time. The music to be recommended are highly selected songs from three different playlists on Spotify. Each playlist represents each class of emotion: (Happiness, Sadness, and Neutrality).

This will help to provide automated means of providing consumers with the music they are most interested in by moving a portable device to their faces to get a picture of their facial expressions.

The paper is organized into six sections:

1. Introduction: Introduces music demand challenges and AI for music recommendation.

2. Literature Review: Explores existing methods for music recommendation, highlighting the limitations and emphasizing the potential enhancements achievable through emotion-driven recommendations.

3. Methodology: Details the Yolo-V3 architecture, dataset training, and the integration of emotion detection into the music recommendation process. Describes the Spotify playlist selection mechanism based on detected emotions.

4. Results: Presents model performance metrics and sample music recommendations.

5. Discussion: Analyze the results, compare the approach with existing methods, address limitations, and discuss potential advancements in the proposed system.

6. Conclusion and Future Work: Summarize the contributions and significance of the research, offering insights into future research directions to further optimize emotion-driven music recommendation.

II. LITERATURE REVIEW

Research into emotion detection has gained significant recognition in recent times, although it is a relatively new area in computer vision and human-computer interaction. With the advancement in object detection algorithms with deep learning. Algorithms can be used to localize objects and detect the class of emotion from the image of facial expression. This paper utilizes the capabilities of deep learning-based algorithms to build an emotion detector that is capable of detecting emotions of three different classes. It also built a REST API that can serve as an intermediary for any client application to recommend songs based on images of facial expressions. This work proposes an emotion recognition system through images of faces and uses it to recommend music to consumers. Some of the work that has been done related to this field is as follows:

A. Datasets

Many databases have been created to build an emotion recognition model. [8] provides a well-annotated database of facial emotions in the wild for public use. It contains more than 100,000 facial images obtained from the internet by querying words related to emotion in different languages.

[14] developed **RafDB (Real-world Affective Faces Database);** a mixture of compound expression and single expression of 300,000 images, to imitate real-world scenarios.

B. Object Detection

This provides a way to accurately classify images into various classes and also precisely estimate the concepts and location of objects contained in each image [16]. It consists of a different subset of tasks such as face detection, and pedestrian detection [11], and can be applied to areas such as autonomous driving. In this project, we aim to apply it to detecting emotion from facial expressions. The architecture of object detection models can mainly be categorized into the following two categories respectively one follows the process of combining region proposals with convolutional neural networks, and the other frames object detection as a regression and classification task.

Region Proposals-Based Frameworks: [16] The region proposal-based methods mainly include R-CNNB [5], Fast R-CNN [4], Faster R-CNN [13], and Mask R-CNN [5], some of which are correlated with each other. R-CNN is implemented in three distinctive steps It adopts a selective search to generate 2000 region proposals for different images, followed by a CNN-based deep feature extraction. In this stage, each region proposal is wrapped or cropped into a fixed resolution, and a CNN [6] architecture is utilized to extract features for final representation. Lastly a classification and localization stage

Fast R-CNN builds on R-CNN to efficiently classify object proposals using deep convolutional networks [4].

Regression and Classification Based Framework. The regression and classification-based methods mainly include MultiBox [3], AttentionNet [15], G-CNN [9], YOLO [12], and SSD [7]. In [12] object detection is approached as a regression problem to spatially separated bounding boxes and associated class probabilities It uses a single neural network to directly predict bounding box and class probability from a single image. This unified architecture is fast.

C. ESP32 IN Object Detection

Object detection in real-time has a variety of uses in military, surveillance, autonomous vehicles, and many more. Much work has been done to deploy object detection models for real-time applications.

ESP32 is an ideal device for deploying object detection models because of its ease of use. Some of the work done utilizing ESP32 for object detection includes ESP32 CAM Object detection and identification with OpenCV [9]. It uses a pre-trained YOLOV3 model to detect objects captured by the EPS32 camera.

III. METHODoLOGY

This section highlights the chosen approach and explains the reason for the approach and how they are implemented to build the recommendation system. The music recommendation system using emotion detection from facial expressions comprises three stages of implementation:

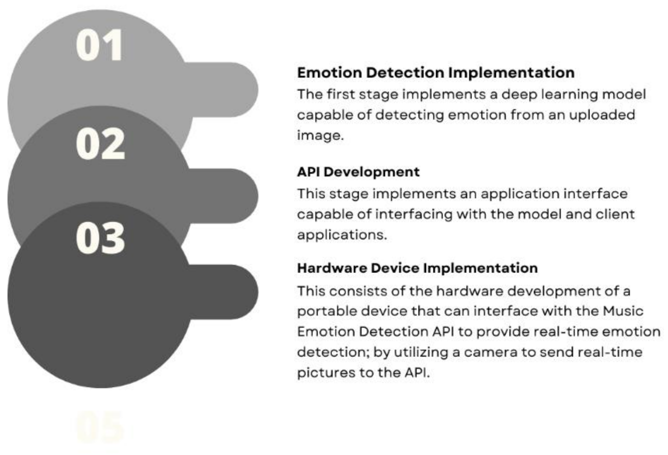

01
02
03
05
Emotion Detection Implementation
The first stage implements a deep learning model capable of detecting emotion from an uploaded image.
API Development
This stage implements an application interface capable of interfacing with the model and client applications.
Hardware Device Implementation
This consists of the hardware development of a portable device that can interface with the Music Emotion Detection API to provide real-time emotion detection; by utilizing a camera to send real-time pictures to the API.

Figure 1: *Implementation Stages*

A. Emotion Detection Implementation

The emotion-detector is a trained deep learning model that is capable of detecting three classes of emotions: happy, sad, and neutral from facial expressions in images. Training a deep learning model requires a large volume of datasets. The dataset selected for this task is the RafDB dataset which contains two sets of images:

1. Images with a single emotion
2. Images with compound emotion

The single emotion images were selected for training the emotion detector because We want to be able to detect a single emotion of a user to be able to recommend songs that best match their emotional state. The selected datasets are properly labeled with their corresponding bounding box, these bounding boxes are simply four coordinates locations of the faces in each image represented by the values of the upper left corner of the bounding box and lower right corner of the bounding box

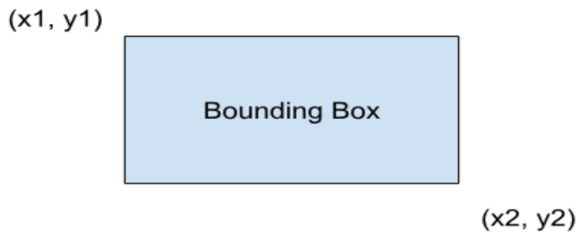


Figure 2: *Schematic illustration of how a bounding box is represented*

The values of the bounding box are the labels for each image, which are in the Pascal format. To use them for training a YOLO architecture they were converted to the center axis, the width and height of the bounding box as described earlier. The PyTorch library is used to implement and train the *YOLO-V3 Architecture* otherwise known as the *Darknet53 Architecture.* The model was trained on the Google Colab Pro-Plus plan to take advantage of its faster GPU and background tasks capabilities.

B. API Development

This stage implements an application interface capable of interfacing with the model and client applications. The API is developed using the Python FastAPI Web framework. FastAPI is the modern, fast (high-performance) Web framework for building APIs with Python. Because Python is already used in this project, it was a no-brainer to choose Python for consistency and the Python FastAPI Web framework for high performance and reliability. The API integrates into Spotify's playlist API to provide music based on emotions detected. Spotify is one of the largest music streaming service providers, with over 433 million monthly active users, including 188 million paying subscribers, as of June 2022. Spotify provides a robust open API for developers to build applications on. The API is deployed as a containerized application using Docker to

Cloud Giant, and AWS (Amazon Web Services), to ensure optimal reliability and scalability.

C. Hardware Device Implementation

ESP32 is an ultra-low power consumption MCU (microcontroller unit) with integrated Wi-Fi and Bluetooth connectivity for mobile devices, wearable electronics, and IoT applications. For this project, we microcontroller board based on the ESP32 Chip that integrates a camera that helps speed up prototyping. The ESP32 is attached to a camera with some of its pins dedicated to housing a camera. The camera module has no USB port for programming like the ESP32 or the ESP8266 microcontroller board hence the need for FTDI232 USB to TTL Converter for programming. The FTDI232 USB to TTL Converter helps to establish Serial communication between the ESP32 CAM and the PC for programming via the ESP32 Cam's TX and RX pin. The ESP32 Chip on the ESP32 cam has Bluetooth and Wi-Fi capabilities, so the ESP32 Cam connects to the internet via Wi-Fi, once it is connected to the internet it can now access the API Endpoint. Once pictures are uploaded to the endpoint the picture is analyzed by the API and the result is sent back to the ESP, The ESP, in turn, uses the Emotion component of the response to decide on the Color of the LED to turn ON. The following are the Components Used:

i. Ai-Thinker ESP32 Cam
ii. FTDI232 USB to TTL Converter
iii. USB to Micro-B cable
iv. RGB LED
v. Female-Female Jumper Cable

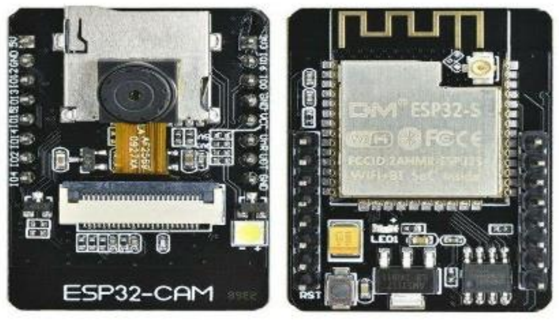


Figure 3: *ESP32-CAM: front and rear views*

Figure 4*: FTDI232 USB to TTL Converter*

Figure 5: *RGB LED*

D. System Model

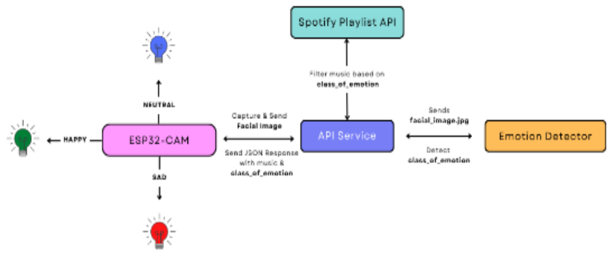


Figure 6: *System Model*

IV. Result and Discussion

To detect emotion from facial expressions we trained an object detector using the deep-learning model for classifying and locating objects in an image. To build the emotion detector we used Darknet53 Architecture which was implemented and trained on the RafDB datasets.

A. Datasets

The datasets consist of 3000 training images and 2000 testing images with three different classes namely happy, sad, and neutral. Each image has its corresponding bounding boxes for the location of faces in the image.

B. Data-Preprocessing

YOLO-V3 Bounding Boxes The original bounding boxes consist of location and give the location of the faces in terms of the upper left axis and lower right axis. We converted the bounding box into the YOLO format which is the center axis and the width and height of the bounding box

Let x1 y1 x2 and y2 be the coordinates of the original bounding boxes then the YOLO format be xc, yc, w, and h which is calculated as follows:

$$c_x = \frac{x_2 + x_1}{2}$$

$$c_y = \frac{y_2 + y_1}{2}$$

$$w = x_2 - x_1$$

$$h = y_2 - y_1$$

Eq. (a): Yolo Bounding box axis.

Anchor Boxes: The anchor box used is the default anchor box from the YOLO-V3 paper which consists of nine anchor boxes that are grouped into three for the three different scales for making predictions. *Given as:*

$$(10 \times 13),\ (16 \times 30),\ (33 \times 23)$$
$$(30 \times 61),\ (62 \times 45),\ (59 \times 119)$$
$$(116 \times 90),\ (156 \times 198),\ (373 \times 326)$$

Eq. (b): Anchor boxes dimension

Normalization: The bounding boxes for each image are normalized by dividing the center x coordinate and the width of the bounding box by the actual width of the image and also the center y coordinate and the height of the bounding boxes are divided by the actual height of the image, given as: For each image i having a width of wi and height of hi it is normalized by:

$$c_x = \frac{c_x}{w_i}$$
$$c_y = \frac{c_y}{h_i}$$
$$w = \frac{w}{w_i}$$
$$h = \frac{h}{h}$$

Eq. (c).: Normalization of bounding box axis

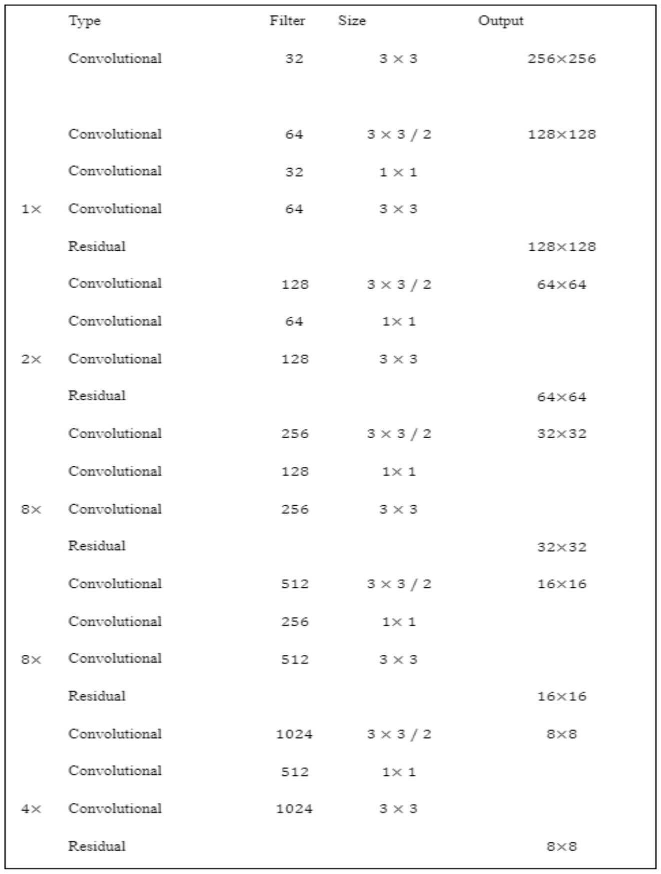
Image Augmentation*:* The only image augmentation technique used is horizontal flip to represent a different aspect of the image to the model and during training.

C. Training

To train the model, the Darknet53 architecture is used to extract the features from the image and then the prediction is made from a single convolutional layer at three different scales. The following steps were taken to train the model.

Feature Extraction: The feature extractor is implemented according to the *YOLO-V3* Architecture; the input image size is resized to 416 to get prediction at the three different scales which are shown in the table below. The feature extractor consists of successive 3×3 and 1×1 convolutional layers with skip connections and in total it has 53 convolutional layers.

Table 1 *Darknet-5*



	Type	Filter	Size	Output
	Convolutional	32	3 × 3	256×256
	Convolutional	64	3 × 3 / 2	128×128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128×128
	Convolutional	128	3 × 3 / 2	64×64
	Convolutional	64	1× 1	
2×	Convolutional	128	3 × 3	
	Residual			64×64
	Convolutional	256	3 × 3 / 2	32×32
	Convolutional	128	1× 1	
8×	Convolutional	256	3 × 3	
	Residual			32×32
	Convolutional	512	3 × 3 / 2	16×16
	Convolutional	256	1× 1	
8×	Convolutional	512	3 × 3	
	Residual			16×16
	Convolutional	1024	3 × 3 / 2	8×8
	Convolutional	512	1× 1	
4×	Convolutional	1024	3 × 3	
	Residual			8×8

D. Prediction

Predictions are made at three different scales, from a single convolutional layer with an output channel of [3 × (4 + 1 + 3)], so that the predicted output is of the shape [3 × (4 + 1 + 3)] × N × N where: N is the width and height of the image, with 3 boxes for the four bounding boxes offset with one confidence score and three classes of emotion. To get the predicted output which is the bounding box for each image, the resulting output from the single convolutional layer is further processed accordingly.

IoU Intersection over Union also known as the Jaccard index is one of the most popular evaluation matrices for object detection tasks and related tasks such as segmentation. IoU plays a very important role in object detection since object detection localizes the object and then classifies the object. IoU is used in both training and inference time to accurately predict the location of the bounding box.
Localization in object detection means drawing a 2D bounding box around the object in an image. In this case around the faces in the image.
The Iou is calculated as shown below

$$IoU = \frac{A \cap B}{A \cup B} = \frac{|I|}{|U|}$$

Eq. (d).: IoU

where A and B are the prediction and ground truth bounding boxes.

One Important property of IoU is the scale invariance: which enables it to take into consideration the width and height of the two bounding boxes.

*a) **Non-maximum Suppression***

This method is used in object detection to select a bounding box from many overlapping bounding boxes. The criteria for selection are usually to discard bounding boxes that are below a given probability bound while the bounding box with the highest probability of containing the output is selected. With remaining entities, repeatedly pick the entity with the highest probability, output that as the prediction, and describe the remaining box where $IoU \geq 0.5\, with$ the box output in the previous step.

*b) **Loss Functions***

The YOLO architecture predicts multiple bounding boxes for each grid cell. To compute, only one must be responsible for predicting the object. so, therefore, select the one with the highest Intersection over union with the ground truth bounding box. This makes each bounding box specialize in predicting certain features of the object. and each prediction gets better at predicting certain sizes and aspect ratios.

It uses the sum of the squared error between the predictions according to the ground truth to calculate the loss according to YOLO. The loss function comprises:

*c) **Classification Loss***

For every detected image, the classification loss is calculated for each cell as the squared error of the class conditional probabilities for each class that is given.

*d) **Localization loss***

The localization loss is responsible for measuring the difference between the actual bounding box (ground truth) and the predicted bounding box for boxes detecting the object

*e) **Confidence Loss***

For every detected object in the box, the confidence loss which measures the abjectness score is given.

E. Object Detection Code Implementation

The images in ***Figure 7:*** *Code snippet for loading the necessary libraries, parsing the directory of the necessary files, and values for hyper-parameters,* Appendix A shows the codes for performing object detection. The trained weights are loaded into the model and then used in performing detection.

a. The darknet architecture is initialized
b. The weights are then loaded into the architecture
c. The image file path is parsed into the image folder class and then, performs all the necessary preprocessing on the image
d. The output from the model is passed into the function which performs non-maximum suppression and returns the predicted bounding box and the class of the image, and the image.
e. The matplotlib library is used to draw the predicted bounding box on the image and save it to a folder.

The function detect which takes no argument performs all of the data-preprocessing and then detects the emotion from the image as shown in the image Appendix A, Figure 8.

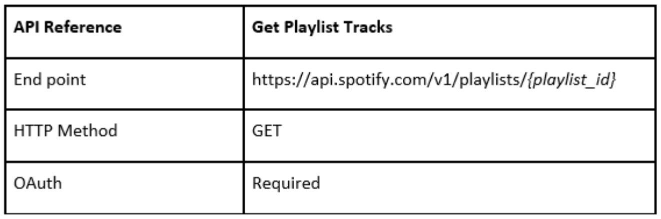
Figure 9: *Code snippet of using the result from the detect function in* Appendix A. The output of the detect function is the class of emotion and the confidence score, while the image containing the predicted bounding box is saved in a separate folder.

F. Music Recommendation Using Spotify API

The research selected three different playlists on Spotify; each playlist contains a collection of songs that have been carefully selected and fall into one of the three different categories; happy, sad, and neutral.

Whenever a class of emotion is predicted from the emotion detector, this is used to automatically retrieve the playlist ID of the predicted class of emotion which is then used as a query argument to request to the Spotify API. Below is the API reference table for the playlist API endpoint

Table 2: *Spotify Playlist API reference table*



API Reference	**Get Playlist Tracks**
End point	https://api.spotify.com/v1/playlists/*{playlist_id}*
HTTP Method	GET
OAuth	Required

The Spotify API response provides metadata for each track which includes, Title, List of Artists, Song URL, and Preview URLs. A track is selected at random and its metadata is returned along with the class of emotion and a link to the bounding box image as a response to the music recommendation API. The API is packaged into a docker application to ensure consistency across operating systems and pushed to Docker HUB (a docker image repository). The image is deployed on AWS ECS (Elastic Container Service) which is a fully managed container orchestration service that makes it easy to deploy, manage, and scale containerized applications. An API URL endpoint is exposed at *http://ed.wevied.co*. Figures 10 and 11 in Appendix A, are sample codes of APA requests and responses.

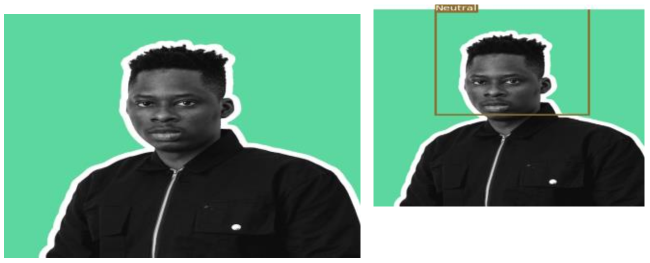


Figure 12: *Image (a) Original image uploaded and (b) Predicted image from the Emotion detected*

The ***bounding_box_image*** URL contains the predicted bounding box and the predicted class. It maintains the same aspect ratio as the original image. The figure below shows the predicted image and the original image. It can be seen that the detector was able to localize the position of the face and correctly classify the class of emotion.

G. Implementation of ESP32 Device for Inference

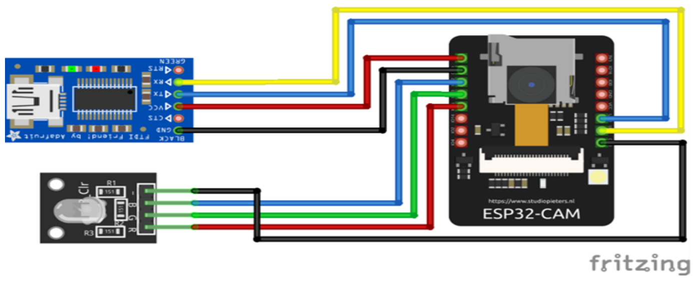


Figure 13: *Circuit diagram of the ESP32 Implementation*

The following Steps were taken to implement the circuit:

a. Connected the GND pin of the FTDI USB Converter to the GND of the ESP32 Cam.---
b. Connected the VCC pin of the FTDI USB Converter to 5V of the ESP32 Cam.
c. Connected the TX pin of the FTDI USB Converter to the VOR of the ESP32 Cam.
d. Connected the Rx pin of the FTDI USB Converter to the VOT of the ESP32 Cam.
e. Connected the GND (-) of the RGB module to the GND of the ESP32 CAM.
f. Connected B pin of the RGB module to IO12 of the ESP32 CAM.
g. Connected the G pin of the RGB module to IO13 of the ESP32 CAM.
h. Connected the R pin of the RGB module to IO15 of the ESP32 CAM.

Now that the circuit is complete, however, the IO0 of the ESP32 Cam must be connected to GND during program upload to put ESP32 CAM in flashing mode, once program upload is complete I disconnected IO0 from GND.

The following steps were taken to the program:

a. Connected IO0 of the ESP32 cam to GND to put the ESP32 cam in flashing mode.
b. Connected the Circuit to my laptop with the USB-B mini port on the FTDI programmer.
c. Utilized PlatformIO plugin on VSCode to upload the code to the ESP32 cam.

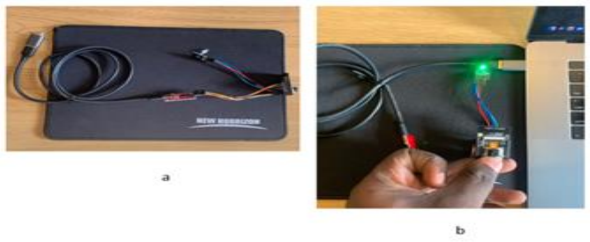


Figure 13: *Complete setup image of the ESP32 Implementation*

H. Inference

The following section covers inference done by using a real-time image captured from the ESP32 to detect emotions and then used in recommending songs. Three separate inferences are carried out for the different categories of emotion; happy, sad, and neutral. The three volunteers are asked to make facial expressions that depict happy, sad, and neutral emotions.

Happy Facial Expression: The image shown below contains the original image and the predicted image containing the predicted bounding box and the class of emotion. It is seen below that the original image is resized while keeping the aspect ratio; this is done in the preprocessing phase.

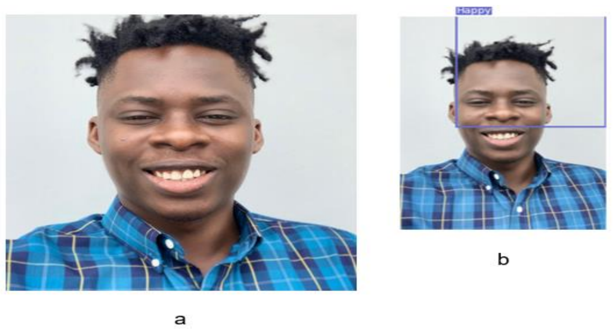


Figure 14: *Image (a) predicted image from the Emotion detected and (b) original image uploaded*

The predicted class which is happy as shown in (b) above is then used to select the playlist ID containing happy songs on Spotify. The JSON response from the API is shown in Figures 15, 17, and 19Appendix A which contains the recommended title of the music, a list of artists, the music URL, the free music preview URL, the confidence score, and the bounding box image containing the prediction.

This inference is reached on both **Neutral Facial Expressions** and **Sad Facial Expressions** as seen below:

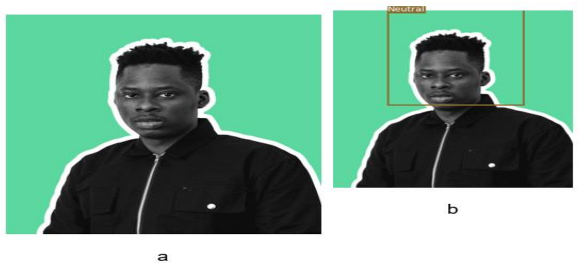


Figure 16: *Image (a) predicted image from the Emotion detected and (b) original image uploaded*

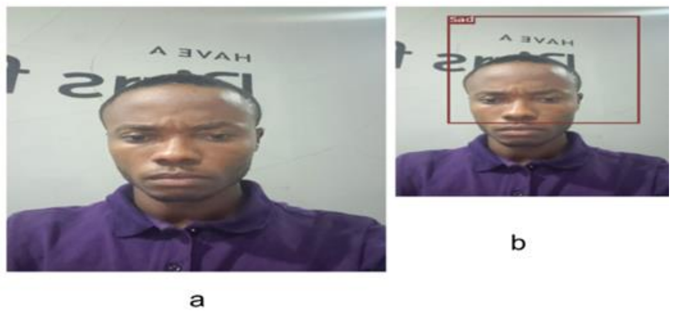


Figure 18: *Image (a) predicted image from the Emotion detected and (b) original image uploaded*

I. Version Control System

GitHub was used as a version control system for the research. ***GitHub*** is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. **GitHub** was used as the code repository and all source codes were pushed to a public repository. Here is the link to the **emotion-detection** GitHub repository - https://github.com/oreofeolurin/emotion-detection

V. Evaluation

a) Model Evaluation

The model was able to achieve a mean average precision value of 0.63 and an inference time of 22ms. This is a good tradeoff for both time and accuracy.

Table 3: Average Precision and Mean Average Precision of the three classes.

Classes	**Average Precision (AP)**
Happiness **(0)**	0.71
Sadness **(1)**	0.58
Neutral **(2)**	0.61
Mean Average Precision	**0.63**

b) End-to-end Music Recommendation App Evaluation

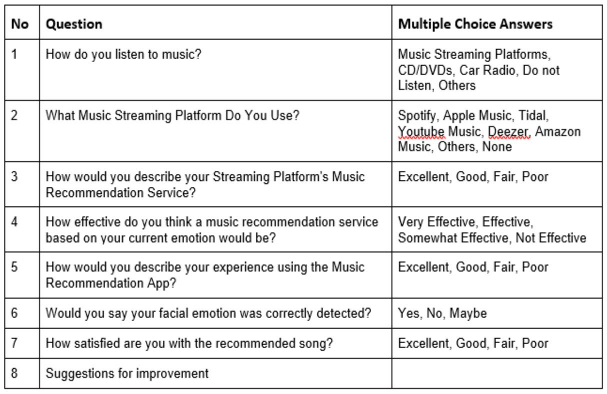
To evaluate the entire project and its relevance we selected 10 different volunteers who used the implemented ESP32 device for music recommendation. After the volunteers used the device, they were sent a Google form to provide feedback based on their overall experience.

To properly evaluate how relevant and important the recommendation system is we consider the following factors that the system has to meet to be considered relevant.

1. The model must be capable of detecting user's emotions from their facial expression.
2. The recommended songs must meet the user's present emotional needs
3. The music recommendation system must be easy to use and improve music searching.

To fully evaluate the following factors above, we created a survey that contains the following questions which each of the volunteers answered after using the device to get music recommendations.

Table 4: *Survey Questions and Answers*



No	Question	Multiple Choice Answers
1	How do you listen to music?	Music Streaming Platforms, CD/DVDs, Car Radio, Do not Listen, Others
2	What Music Streaming Platform Do You Use?	Spotify, Apple Music, Tidal, Youtube Music, Deezer, Amazon Music, Others, None
3	How would you describe your Streaming Platform's Music Recommendation Service?	Excellent, Good, Fair, Poor
4	How effective do you think a music recommendation service based on your current emotion would be?	Very Effective, Effective, Somewhat Effective, Not Effective
5	How would you describe your experience using the Music Recommendation App?	Excellent, Good, Fair, Poor
6	Would you say your facial emotion was correctly detected?	Yes, No, Maybe
7	How satisfied are you with the recommended song?	Excellent, Good, Fair, Poor
8	Suggestions for improvement	

Apple Music and Spotify are most used by the respondents, each with 40% usage. About 90% of the respondents believed their facial emotion was correctly detected. 60% of the respondents were very satisfied with the recommended song, while 10% answered **"Fair"**.

Some of the respondents provided suggestions for the device, here are some of them:

1. The music suggestions seem generalized. *The fact that I frowned didn't mean I was heartbroken, it could be work stress.*
2. Music should be more diverse with other genres.
3. Recommending five different music will be cool
4. Having the API work in wearables would be nice.
5. Different genres of music. I'm a Christian and I want gospel songs of different moods.
6. The device was cumbersome to use and should be designed to be more user-friendly.

VI. CONCULSION AND RECOMMENDATION

In conclusion, this research successfully introduces an innovative methodology employing image recognition integrated into portable devices for music recommendation based on emotion detection from facial expressions. The deep learning-based emotion recognition model exhibited remarkable performance, achieving a minimal loss of 1.69 after extensive training across a hundred epochs. The augmentation techniques implemented further enhanced the model's robustness and ability to generalize effectively to unseen datasets. The seamless integration of this model into portable devices underscores the practical applicability and relevance of this research, providing an enhanced approach to music recommendation, and ultimately enriching user experiences. By addressing various technological domains such as software development, deep learning, and the Internet of Things, this study contributes significantly to the intersection of these fields and opens up exciting avenues for future advancements in emotion-based music recommendation systems. As technology continues to evolve, this research can serve as a foundational step towards incorporating emotion-aware algorithms into a wide range of applications, enhancing personalized user interactions, and paving the way for a more emotionally intelligent digital landscape.

VII. RECOMMENDATION AND FUTURE WORK

The entire work is well-planned and implemented using techniques that follow software and hardware best practices. The following highlights some possible improvements:

a. Optimizing Prediction Accuracy

Training for more epochs will increase the model's performance in locating the faces. This was not done due to the challenge faced during training as training models in the cloud are financially costly and time-consuming.

b. Optimizing support for Blurry Images

The model does not seem to perform well on blurry images; this could be rectified by training on blurrier images.

c. Enhancing User Experience

Creating an on-device user interface and the ability to play the music right from the device would help improve the user experience.

d. Deep Learning in Portable Devices

One of the common setbacks of developing real-time applications is latency. Our current architecture leverages the Internet's HTTP protocol to provide access to the model in the form of a REST API; this introduces a latency of around 90-120ms in between calls. This could be easily eliminated if the model was deployed directly on the ESP32.

References

[1] Baradel, F. (2021). *Structured deep learning for video analysis*. https://hal.archives-ouvertes.fr/tel-03117344v2/document

[2] Carterette, B. (n.d.). Music recommendation at Spotify.

[3] Erhan, D., Szegedy, C., Toshev, A., & Anguelov, D.(2013). *Scalable Object Detection using Deep Neural Networks*. https://arxiv.org/abs/1312.2249

[4] Girshick, R. (2015). *Fast R-CNN*. https://arxiv.org/abs/1504.08083

[5] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). *Rich feature hierarchies for accurate object detection and semantic segmentation*. http://arxiv.org/abs/1311.2524

[6] Krizhevsky, A., Sutskever, l., & Hinton, G. E. (2017). *ImageNet Classification with Deep Convolutional Neural Networks*. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[7] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2015). *SSD: Single Shot MultiBox Detector*. https://doi.org/10.1007/978-3-319-46448-0_2

[8] Mollahosseini, A., Hasan, B., & H. Mahoor, M. (2017). *AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild*. https://arxiv.org/abs/1708.03985

[9] Ninad, M. (2022). *Object Detection using ESP 32 CAM*. SSRN. 10.2139/ssrn.4152378.

[10] Najibi, M., Rastegari, M., & Davis, L. S. (2015). *G-CNN: an Iterative Grid Based Object Detector*. https://arxiv.org/abs/1512.07729

[11] Perona, P., Wojek, C., Schiele, B., & Dollár, P. (2012). *Pedestrian detection: an evaluation of the state of the art*. PubMed. https://pubmed.ncbi.nlm.nih.gov/21808091/

[12] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. https://arxiv.org/abs/1506.02640

[13] Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. http://arxiv.org/abs/1506.01497

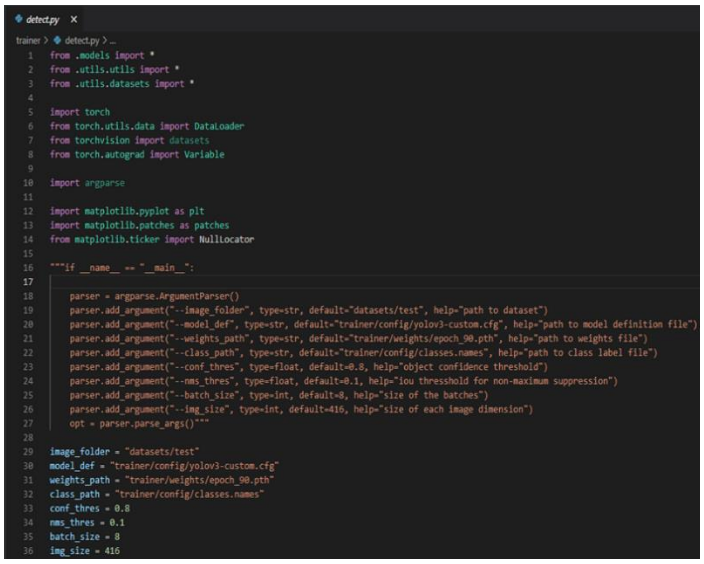
[14] Shan Li, Weihong Deng, and JunPing Du, Li, S., Deng, W., & Du, J. P. D. (1995). *Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild*. http://whdeng.cn/RAF/li_RAFDB_2017_CVPR.pdf

[15] Yoo, D., Park, S., Lee, J.-Y., Paek, A. S., & Kweon, I. S. (2015). *AttentionNet: Aggregating Weak Directions for Accurate Object Detection*. https://arxiv.org/abs/1506.07704

[16] Zhao, Z.-Q., Zheng, P., Xu, S., & Wu, X. (2018). *Object Detection with Deep Learning: A Review*. http://arxiv.org/abs/1807.05511

Appendices

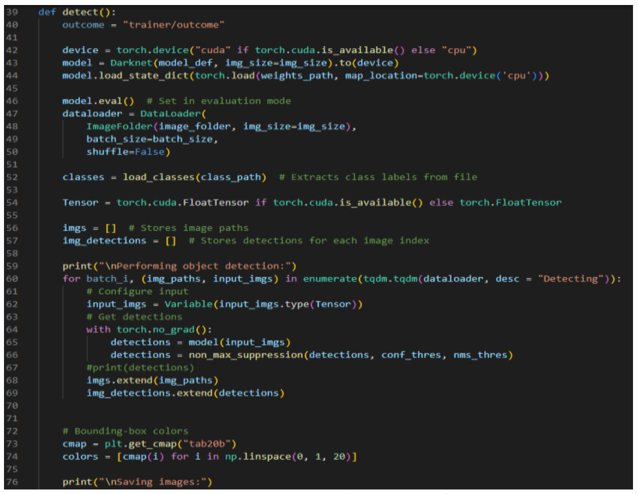
Appendix A – Code Implementation



```python
from .models import *
from .utils.utils import *
from .utils.datasets import *

import torch
from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import argparse

import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.ticker import NullLocator

"""if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument("--image_folder", type=str, default="datasets/test", help="path to dataset")
    parser.add_argument("--model_def", type=str, default="trainer/config/yolov3-custom.cfg", help="path to model definition file")
    parser.add_argument("--weights_path", type=str, default="trainer/weights/epoch_90.pth", help="path to weights file")
    parser.add_argument("--class_path", type=str, default="trainer/config/classes.names", help="path to class label file")
    parser.add_argument("--conf_thres", type=float, default=0.8, help="object confidence threshold")
    parser.add_argument("--nms_thres", type=float, default=0.1, help="iou thresshold for non-maximum suppression")
    parser.add_argument("--batch_size", type=int, default=8, help="size of the batches")
    parser.add_argument("--img_size", type=int, default=416, help="size of each image dimension")
    opt = parser.parse_args()"""

image_folder = "datasets/test"
model_def = "trainer/config/yolov3-custom.cfg"
weights_path = "trainer/weights/epoch_90.pth"
class_path = "trainer/config/classes.names"
conf_thres = 0.8
nms_thres = 0.1
batch_size = 8
img_size = 416
```

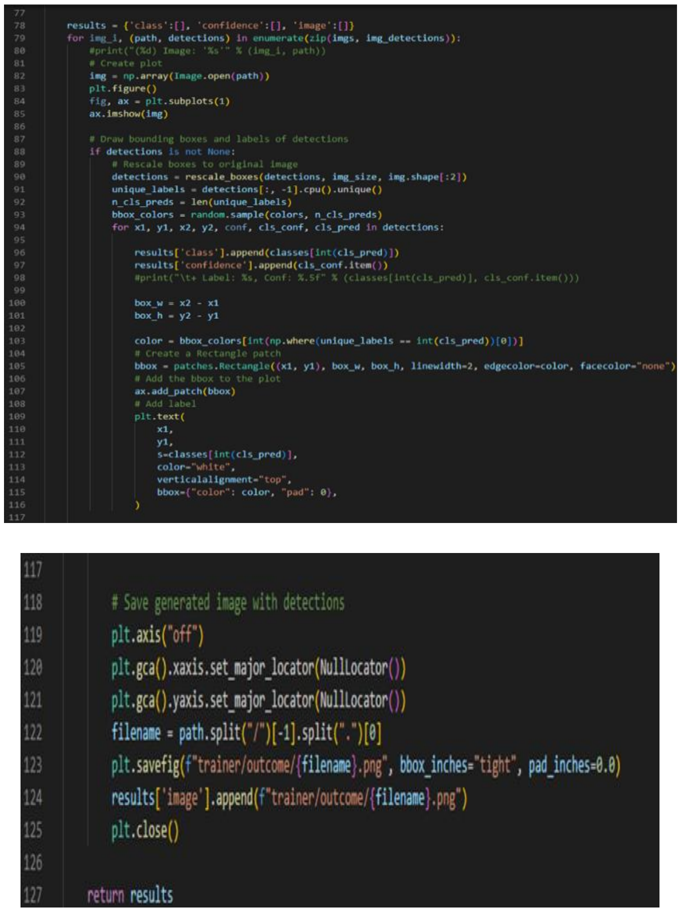
Figure 7: *Code snippet for loading the necessary libraries, parsing the directory of the necessary files, and values for hyper-parameters.*

The function detects which takes no argument performs all the data-preprocessing and then detects the emotion from the image as shown in the image below:



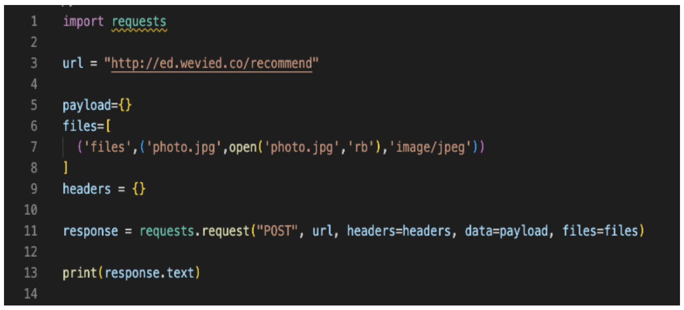
```python
def detect():
    outcome = "trainer/outcome"

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = Darknet(model_def, img_size=img_size).to(device)
    model.load_state_dict(torch.load(weights_path, map_location=torch.device('cpu')))

    model.eval()  # Set in evaluation mode
    dataloader = DataLoader(
        ImageFolder(image_folder, img_size=img_size),
        batch_size=batch_size,
        shuffle=False)

    classes = load_classes(class_path)  # Extracts class labels from file

    Tensor = torch.cuda.FloatTensor if torch.cuda.is_available() else torch.FloatTensor

    imgs = []  # Stores image paths
    img_detections = []  # Stores detections for each image index

    print("\nPerforming object detection:")
    for batch_i, (img_paths, input_imgs) in enumerate(tqdm.tqdm(dataloader, desc = "Detecting")):
        # Configure input
        input_imgs = Variable(input_imgs.type(Tensor))
        # Get detections
        with torch.no_grad():
            detections = model(input_imgs)
            detections = non_max_suppression(detections, conf_thres, nms_thres)
        #print(detections)
        imgs.extend(img_paths)
        img_detections.extend(detections)


    # Bounding-box colors
    cmap = plt.get_cmap("tab20b")
    colors = [cmap(i) for i in np.linspace(0, 1, 20)]

    print("\nSaving images:")
```

Figure 8: *Code Snippet of the detect function.*



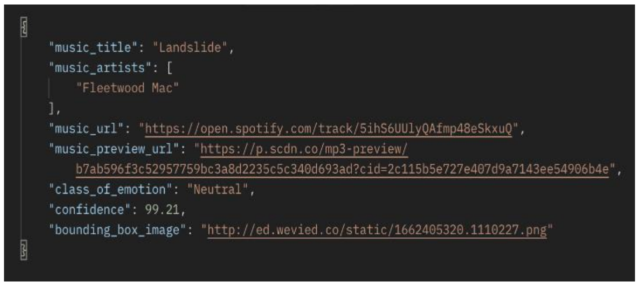
```python

    results = {'class':[], 'confidence':[], 'image':[]}
    for img_i, (path, detections) in enumerate(zip(imgs, img_detections)):
        #print("(%d) Image: '%s'" % (img_i, path))
        # Create plot
        img = np.array(Image.open(path))
        plt.figure()
        fig, ax = plt.subplots(1)
        ax.imshow(img)

        # Draw bounding boxes and labels of detections
        if detections is not None:
            # Rescale boxes to original image
            detections = rescale_boxes(detections, img_size, img.shape[:2])
            unique_labels = detections[:, -1].cpu().unique()
            n_cls_preds = len(unique_labels)
            bbox_colors = random.sample(colors, n_cls_preds)
            for x1, y1, x2, y2, conf, cls_conf, cls_pred in detections:

                results['class'].append(classes[int(cls_pred)])
                results['confidence'].append(cls_conf.item())
                #print("\t+ Label: %s, Conf: %.5f" % (classes[int(cls_pred)], cls_conf.item()))

                box_w = x2 - x1
                box_h = y2 - y1

                color = bbox_colors[int(np.where(unique_labels == int(cls_pred))[0])]
                # Create a Rectangle patch
                bbox = patches.Rectangle((x1, y1), box_w, box_h, linewidth=2, edgecolor=color, facecolor="none")
                # Add the bbox to the plot
                ax.add_patch(bbox)
                # Add label
                plt.text(
                    x1,
                    y1,
                    s=classes[int(cls_pred)],
                    color="white",
                    verticalalignment="top",
                    bbox={"color": color, "pad": 0},
                )


        # Save generated image with detections
        plt.axis("off")
        plt.gca().xaxis.set_major_locator(NullLocator())
        plt.gca().yaxis.set_major_locator(NullLocator())
        filename = path.split("/")[-1].split(".")[0]
        plt.savefig(f"trainer/outcome/{filename}.png", bbox_inches="tight", pad_inches=0.0)
        results['image'].append(f"trainer/outcome/{filename}.png")
        plt.close()

    return results
```

Figure 9: *Code snippet of using the result from the detect function.*



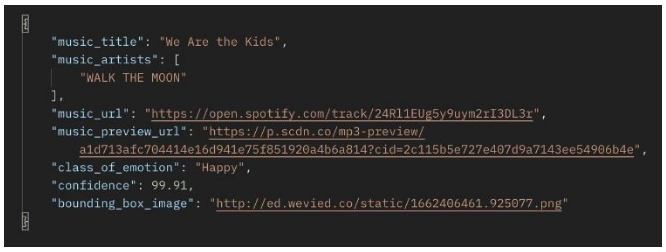
```python
import requests

url = "http://ed.wevied.co/recommend"

payload={}
files=[
  ('files',('photo.jpg',open('photo.jpg','rb'),'image/jpeg'))
]
headers = {}

response = requests.request("POST", url, headers=headers, data=payload, files=files)

print(response.text)

```

Figure 10: *Sample API Request*



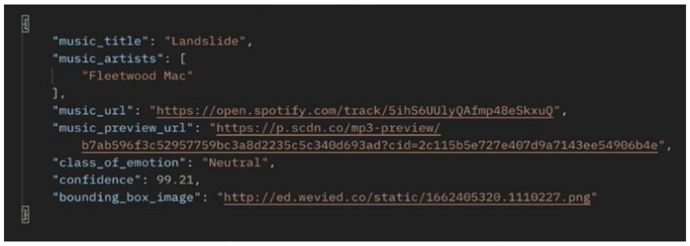
```json
{
    "music_title": "Landslide",
    "music_artists": [
        "Fleetwood Mac"
    ],
    "music_url": "https://open.spotify.com/track/5ihS6UUlyQAfmp48eSkxuQ",
    "music_preview_url": "https://p.scdn.co/mp3-preview/
        b7ab596f3c52957759bc3a8d2235c5c340d693ad?cid=2c115b5e727e407d9a7143ee54906b4e",
    "class_of_emotion": "Neutral",
    "confidence": 99.21,
    "bounding_box_image": "http://ed.wevied.co/static/1662405320.1110227.png"
}
```

Figure 11: *Sample API Response*



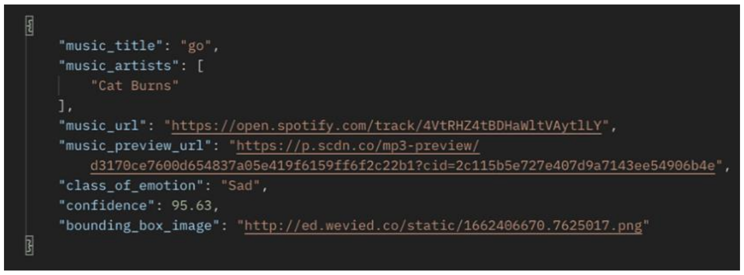
```json
{
    "music_title": "We Are the Kids",
    "music_artists": [
        "WALK THE MOON"
    ],
    "music_url": "https://open.spotify.com/track/24Rl1EUg5y9uym2rI3DL3r",
    "music_preview_url": "https://p.scdn.co/mp3-preview/
        a1d713afc704414e16d941e75f851920a4b6a814?cid=2c115b5e727e407d9a7143ee54906b4e",
    "class_of_emotion": "Happy",
    "confidence": 99.91,
    "bounding_box_image": "http://ed.wevied.co/static/1662406461.925077.png"
}
```

Figure 15: *The response from the recommendation API after sending an image file containing a happy facial expression*



```json
{
    "music_title": "Landslide",
    "music_artists": [
        "Fleetwood Mac"
    ],
    "music_url": "https://open.spotify.com/track/5ihS6UUlyQAfmp48eSkxuQ",
    "music_preview_url": "https://p.scdn.co/mp3-preview/
        b7ab596f3c52957759bc3a8d2235c5c340d693ad?cid=2c115b5e727e407d9a7143ee54906b4e",
    "class_of_emotion": "Neutral",
    "confidence": 99.21,
    "bounding_box_image": "http://ed.wevied.co/static/1662405320.1110227.png"
}
```

Figure 17: The response from the recommendation API after sending an image file containing a neutral facial expression



```json
{
    "music_title": "go",
    "music_artists": [
        "Cat Burns"
    ],
    "music_url": "https://open.spotify.com/track/4VtRHZ4tBDHaWltVAytlLY",
    "music_preview_url": "https://p.scdn.co/mp3-preview/
        d3170ce7600d654837a05e419f6159ff6f2c22b1?cid=2c115b5e727e407d9a7143ee54906b4e",
    "class_of_emotion": "Sad",
    "confidence": 95.63,
    "bounding_box_image": "http://ed.wevied.co/static/1662406670.7625017.png"
}
```

Figure 19: *The response from the recommendation API after sending an image file containing a sad facial expression*