CPEN 291 Final Project

By Deepan Chakravarthy, Aryan Gandhi, Steve He, and Arya Phan

CPEN 291 Section 201

1.Project Overview

Music has been shown to play an important role in people's lives and have an impact on our emotions. The majority of people usually choose specific genres of music based on their current mood. For example, listening to upbeat music boosts up our energy; listening to sad, soothing songs help us feel calmer and relaxed. Our project dives into this concept and focuses on generating songs that can lift and boost up a user's mood.

This project is a web based application that predicts users' moods (e.g. sadness, anger, happiness, and neutral) based on their facial expressions using computer vision. Based on the prediction of our machine learning model, the app outputs a Spotify playlist that suits the result. Our web backend was built using the Django framework and Python. The machine learning model was built using Pytorch and a dataset comprising 500-600 images per emotion. The web frontend was designed using HTML, CSS, Javascript and Bootstrap.

2. Project Components

2.1 Overview and overall data flow diagram

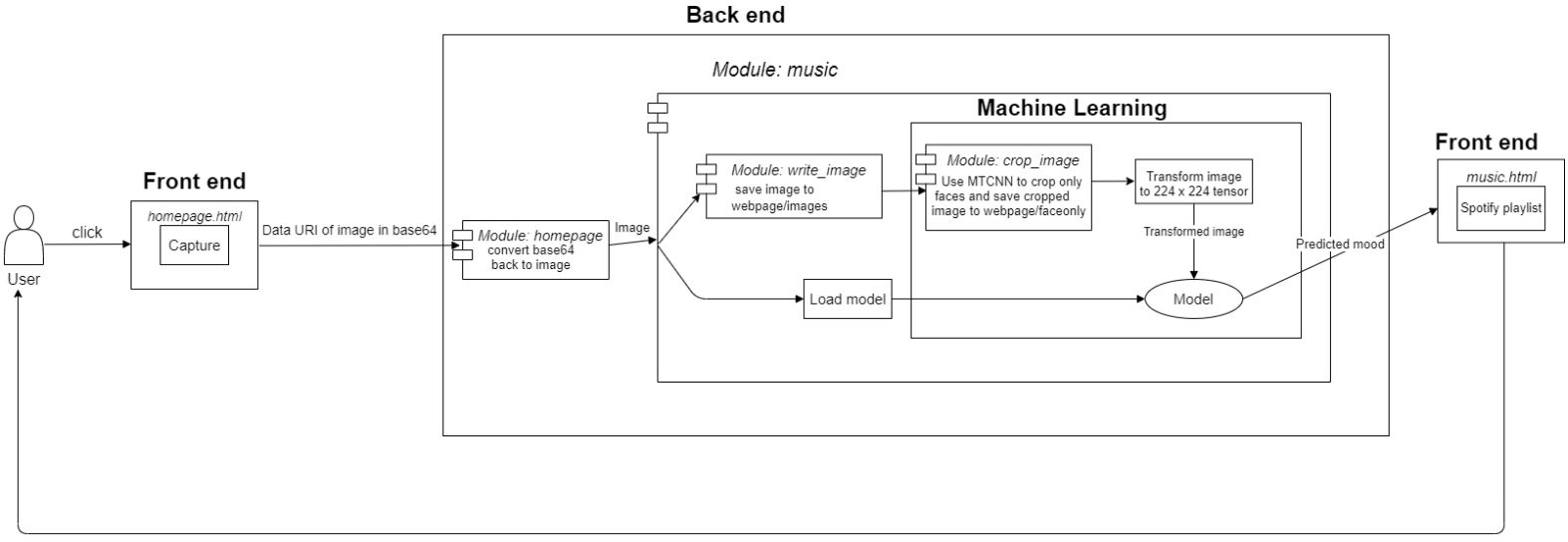


Figure 1: Overall data flow diagram

When a user submits their photo by clicking the "Capture" button on our homepage, the web frontend sends the input to our backend where different manipulation is performed on the image. It is then fed into our model to obtain the predicted mood and displays the

corresponding Spotify playlist. Please see the below sections for a detailed analysis of each component in the diagram.

2.2 Frontend components

Our web frontend was built using HTML, CSS, and Javascript. In addition, we used Bootstrap to support our web design. The web app consists of 3 webpages: homepage.html, music.html, and about.html which are rendered in the Django framework (views.py) corresponding to the specified url patterns.

The homepage is our main web page that shows up when the web app starts up. We keep the design short and simple as it needs to deliver to visitors what the web app offers within a few seconds. This is where users can upload their photos taken by webcams so that the website can send the input to our web backend. Javascript was used to display webcams on the page and send a base64 version of the image to our web backend through the button “Capture”.

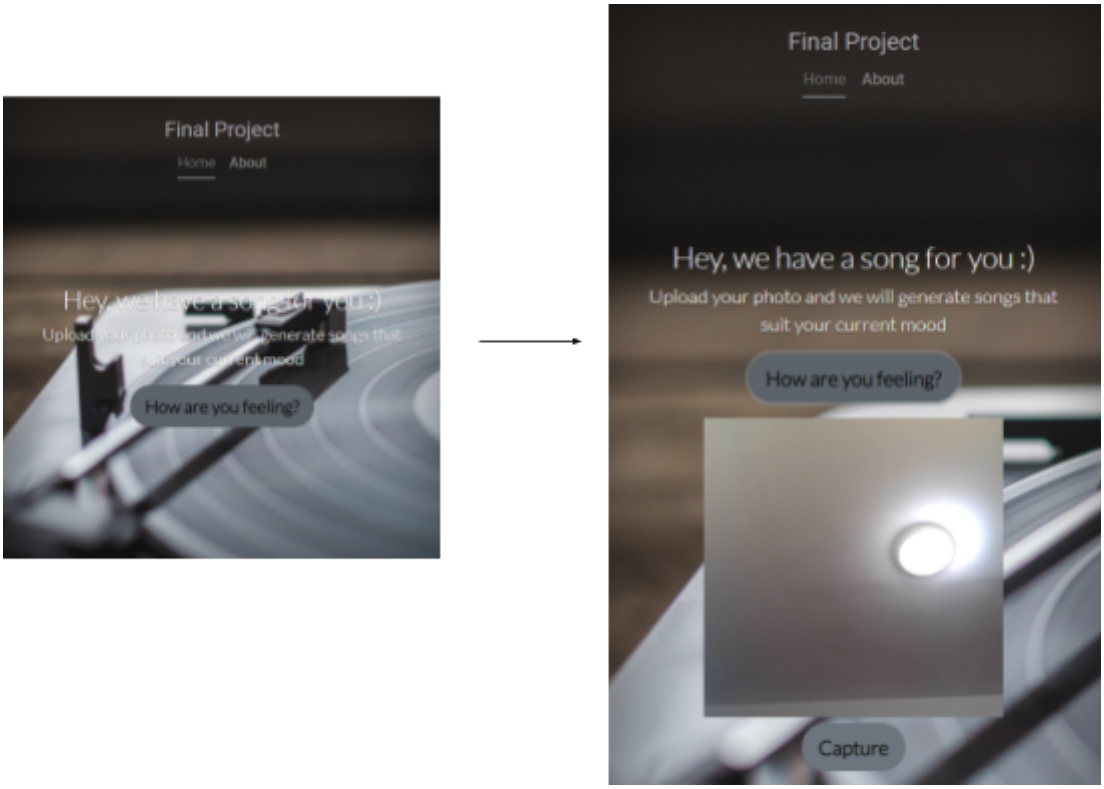


Figure 2: “Capture” button used to send image data to the backend

music.html is the output webpage where we show users our recommended playlist that suits their current mood. After our backend obtains the predicted result from the machine learning model, a Spotify playlist’s URL corresponding to the result is sent to music.html. Here, the output web page displays a message and a Spotify player based on the predicted mood. The Spotify player displays the first 100 songs and users will need to log in Spotify to see our full playlist. In addition, users have an option to go back to the homepage to change their mood.

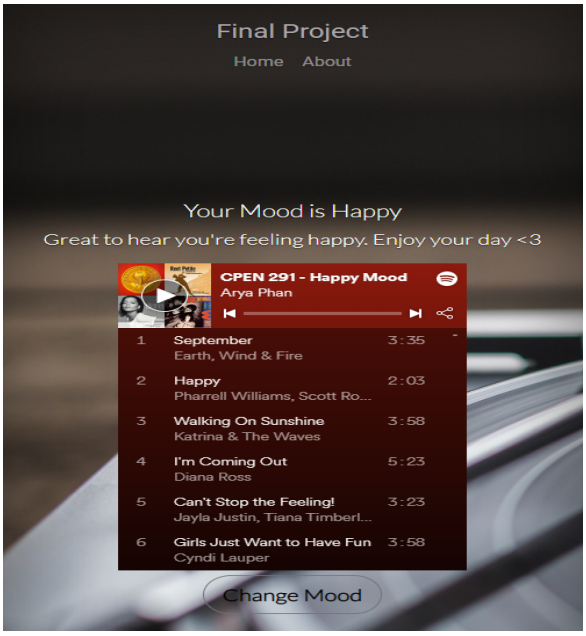


Figure 3: Output webpage

The About page displays the information about our project. Specifically, users will find a summary of the purpose of our project, how the web app works, and what software tools we used to implement it.

2.3 Backend web page components

2.3.1 Webcam Capturing

Since our model was trained on images, our web app requires users' pictures taken by their webcams as inputs. Since we are using HTML for the frontend, the use of javascript for the webcam was a viable solution as it simply links with the HTML code through using the tag <script>. Below is the code used to set up the webcam.

```
//access webcam
async function init() {
    try {
        const stream = await navigator.mediaDevices.getUserMedia(constraints);
        handleSuccess(stream);
    } catch(e) {
        errorMsgElement.innerHTML = `navigator.getUserMedia.error:${e.toString()}`;
    }
}
```

Figure 4: Code for webcam setup

We use try and catch blocks in order to capture any errors. The main line that captures webcam is the line stream = await navigator.mediaDevices.getUserMedia(constraints).

2.3.2 POST and Django Request

```
var context = canvas.getContext('2d');
snap.addEventListener("click", function(){
    context.drawImage(video, 0, 0, 224, 224);
    var myImage = canvas.toDataURL("image/png");
    document.getElementById('imageData').value = myImage;
});
```

Figure 5: Code for converting image to data URI in base64

As in Figure 1, when the “Capture” button is pressed, the canvas (webcam display) takes a snapshot of the image from the webcam and resizes it to a 224x224 image. The image is then converted to a data URI in base64 (see Figure 5) and sent as a POST request from the blank form on our html file as shown below.

```
<canvas id="canvas" width="224" height="224" hidden></canvas>
<form id="imageForm" name="imageForm" method="post" enctype="multipart/form-data" >
    {% csrf_token %}
```

Figure 6: Code for sending data as a POST request to our backend

```
def homepage(request):

    data_uri = request.POST.get('imageData')

    if data_uri != None:
        __, encoded = data_uri.split(",", 1)

        data = base64.b64decode(encoded)
        im_arr = np.frombuffer(data, dtype=np.uint8)
        img = cv2.imdecode(im_arr, flags=cv2.IMREAD_COLOR)
        return music(request, img)

    return render(request, 'webpage/homepage.html')
```

Figure 7: homepage function for converting base64 back into an image

The POST request is captured by the homepage python function.The function decodes the Data URI in order to convert the base64 version back into an image before sending it to the music function where we will perform different transformations to the input.

2.3.3 Passing to Model

```
def write_image(image):
    cv2.imwrite("webpage\\images\\image.png", image)
    source_dir = 'webpage/images'
    dest_dir = 'webpage/faceonly'
    uncropped_files_list=crop_image(source_dir, dest_dir,1)

def music(request, img):
    write_image(img)
    model=torch.load('projectmod',map_location=torch.device('cpu'))
    model.eval()
    xform = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor()])
    img=Image.open('webpage\\faceonly\\image.png')
    data = xform(img)
    output = model(data[None, ...])
```

Figure 8: Functions that manipulates data and feeds the image to the machine learning model

The three functions, music, write_image, and crop_image, are used to manipulate our image data. When the image is passed to the music function, it is first written to the directory webpage/images by the write_image function using cv2. Next, the crop_image function accesses the saved input and crop the picture using MTCNN so that it contains only the user's face (more detail about MTCNN is discussed in section 2.4.3 below). The purpose of cropping the image is to improve the accuracy of the model when faces are not in the center of the image. The music function then transforms the cropped image into a 224x224 tensor and passes it to the loaded model to determine the mood of the user.

2.3.4 Output Mood

```
    preds=torch.max(output.detach(), 1)[1].item()
    if(preds==0):
        #return angry list
        return render(request,'webpage/music.html', {'mood': 'Angry', 'playlist': 'playlist/5XBvBzMBRcdCrKUAs5VoR4'})
    if(preds==1):
        #return happy list
        return render(request,'webpage/music.html', {'mood': 'Happy', 'playlist': 'playlist/14FMPAe3yffuI47MUX3Alg'})
    if(preds==2):
        #return neutral list
        return render(request,'webpage/music.html', {'mood': 'Neutral', 'playlist': 'playlist/31n4J7BzWnoOPLuTCyZYRv'})
    if(preds==3):
        #return sad list
        return render(request,'webpage/music.html', {'mood': 'Sad', 'playlist': 'playlist/42zINuckj3dU1CHcGBSrwk'})
```

Figure 9: Code for the output mood

Using the output from the model and torch.max allows us to get the prediction from the model. By using if statements, we are able to determine the mood and pass the model output and the

corresponding playlist packaged as a context to the output web page where the recommended songs are displayed to the user.

.

2.4 Backend Machine Learning and training features

2.4.1 Dataset

Our dataset contains images for 4 different emotions: happy, sad, angry, and neutral. These images were scraped from Bing using two Bing image downloader tools:

- https://github.com/ostrolucky/Bulk-Bing-Image-downloader
    - This scraping tool does not download duplicates and allows us to apply filters on the searches. It ensures all images are unique and eliminates irrelevant photos. For example, we used a filter that gives us results of human faces only. This helped us eliminate photos that are not appropriate for the project, such as emojis, cartoon images, etc. The main limitation is that it only downloads a small number of images at a time.
- https://github.com/gurugaurav/bing_image_downloader
    - This tool allows us to set the number of images to be downloaded. As such, we can use this for downloading a large number of images at a time.

We executed the tools' commands with different Bing query strings to obtain more unique images. The combined results of two tools helped us achieve 500-600 images for each mood after removing rejected photos. The rejects were eliminated due to various reasons and were hand selected based on our judgement. The images are saved in 4 folders and each folder name is served as a label for our machine learning model.

2.4.2 Training diagram and brief description

The overall training process was transfer learning on a pretrained Resnet18 model. The below diagram shows the flow of data. The sections below will explain each component for training.

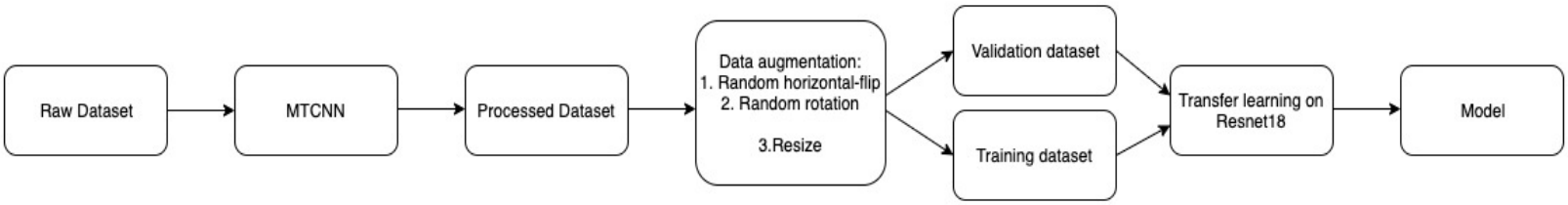


Figure 10: Training data flow diagram

2.4.3 MTCNN

This model takes an image as its only parameter and outputs the bounding box coordinates for the face's location and the facial features' (eyes, nose, mouth) locations. The Raw dataset contains images of faces but with other obstructing features such as backgrounds and other body parts. Hence, all images were passed through the MTCNN model where the bounding boxes were received and all images were cropped according to the bounding boxes. Below is an example of this process.

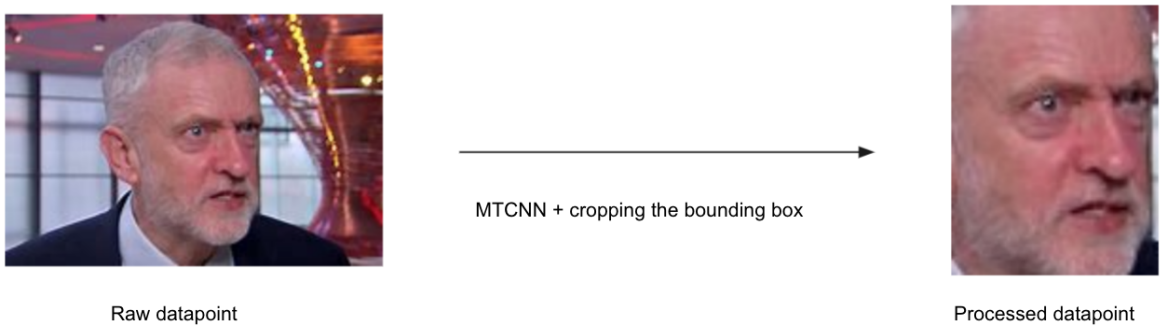


Figure 11: MTCNN and cropping

2.4.4 Processed dataset

The outputs from the previous component are separately saved and used for the training process. This dataset is of higher quality than the raw dataset as only facial features will be passed into the neural networks and not background objects.

2.4.5 Data augmentation

The dataset is then passed into a data loader with pytorch transforms to do data augmentation. Specifically, random horizontal flip and random rotation of +15 and -15 degrees were applied. We had around 500-600 data points per emotion for a total of around 2085 data points. With data augmentation over multiple epochs, there will be more combinations of data being passed to train the model. Hence, the model trains on more data. This led to a 2-3% increase of validation accuracy.

2.4.6 Validation-training data split

The dataset was split where 80% of data belongs to the training dataset and the remaining is used for testing. This was done to check for accuracy for a new and unknown image and also compare to other literature accuracies.

2.4.7 Transfer learning on Resnet18

Resnet18 is a pretrained model that is able to classify all labels under the imagenet database. Resnet18 is already able to classify a large number of animals to a reasonable accuracy. Hence it was predicted that it should also be able to identify human facial features. As such, Resnet18 was chosen for our project.

The below diagram shows the final FC layer's contents after modifying it for our 4 labels.

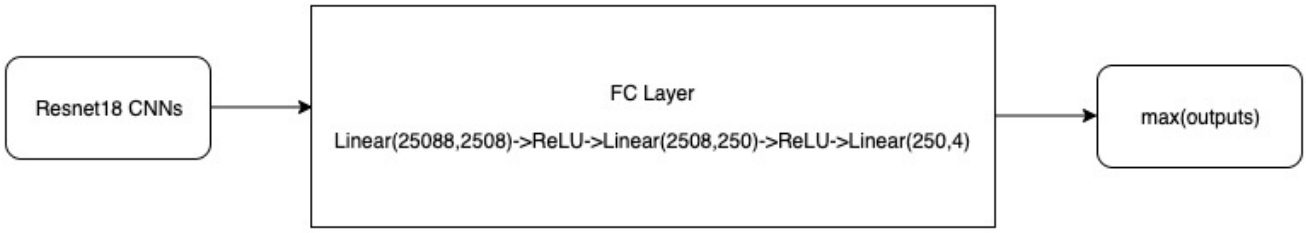


Figure 12: Model,contents of FC layer

Moreover, the average pooling layer was removed to allow the 25088 features to be passed on. This allowed for more features to be filtered throughout the 3 linear layers instead of collapsing them, hence improving accuracy.

2.4.8 Final model and results

Using transfer learning above, two models were produced. First, a model without MTCNN and next a model with MTCNN. Below are the accuracies acheived.

Model Version	Training Accuracy	Validation loss	Validation Accuracy
Model without MTCNN	88.81%	0.0815	88.31%
Model with MTCNN	99.88%	0.1080	86.09%

From the above versions, the Model with MTCNN was chosen. Although there was a 2% reduction in validation accuracy, the almost 10% increase in training accuracy means the model trained well on the faces-only dataset. Compared to previous work in facial sentiment analysis, our accuracy is acceptable. For instance, in the paper ‘Automatic Emotion Recognition for the Calibration of Autonomous Driving Functions’ by Violante, they used the popular FER2013 dataset of faces and achieved an accuracy of around 78.9%. This might be due to FER2013 being an older dataset and ours being newly collected and passed through the MTCNN.

3. Contributions of team members

Deepan Chakravarthy

I was responsible for the machine learning components of the project. I first created the processed dataset by using the MTCNN model together with our original dataset. Then, I used these data points to train a model that would predict the four labels. I experimented on different parameters of the ML component to achieve the maximum accuracy possible. For instance, modifying learning rate, batch size, etc. Moreover, I tested with different pytorch transforms to identify those that increase validation accuracy. Finally, I also modified the FC layer of the Resnet18 and tried various additions such as using dropouts to minimize overfitting.

Aryan Gandhi

Throughout the project, I was responsible for working on the frontend and backend development. I worked on setting up the webcam and allowing it to send POST request from a javascript form in order to send a base64 version of the image to the django request where I converted the number back to an image using numPy and OpenCV and thus allowing the model to receive it. I also worked on linking the html web pages to the django code in order to receive the requests and render the correct pages. Additionally, I worked on testing the code and ensured no major bugs were missed and all edge cases were taken into account. As well, I did code cleanup, worked on fixing small design things on the frontend and writing about the section for the website.

Steve He

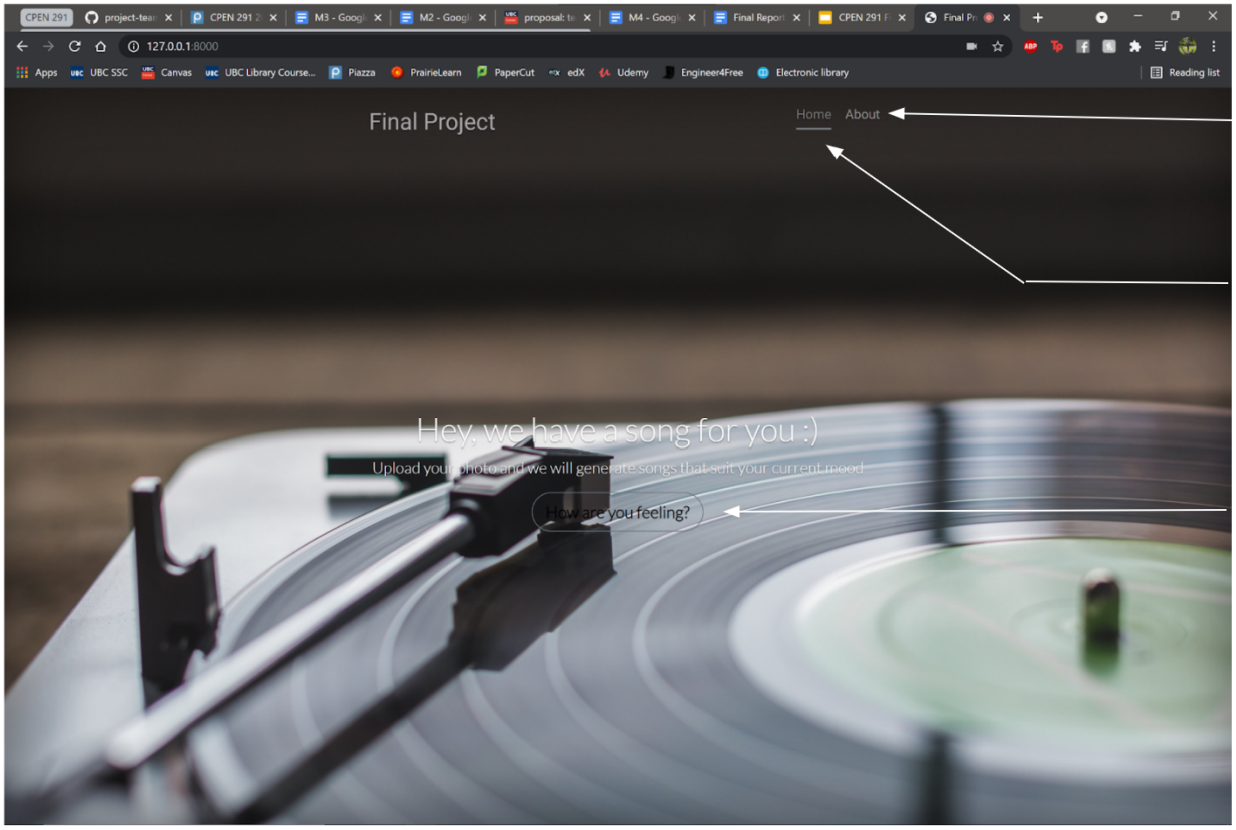
I was responsible for saving and integrating the final model and MTCNN model into our django webpage using pytorch features so that it takes the image captured by a webcam as input and first crops the image using the MTCNN model and passes the image with face only to our trained model. Then our model gives predictions about the user's current mood. Based on the output from our model, different playlists will be presented on our output web page. I was also responsible for finding any bugs and issues in our website structure and resolving them.

Arya Phan

I was responsible for building the dataset, designing and building our web frontend using HTML, CSS, Javascript and Bootstrap, and linking our web frontend to the backend. I also wrote the content displayed on our web pages. Additionally, I created Spotify playlists corresponding to each mood and sent the corresponding playlist URL from the backend to the output page based on the predicted result. Furthermore, I tested our web app using different images to ensure the model is working properly. Lastly, I cleaned up our code and prepared slides and a walkthrough demo for our project showcase video.

4. Web Demo

Please see the images below for a walkthrough example of our project..



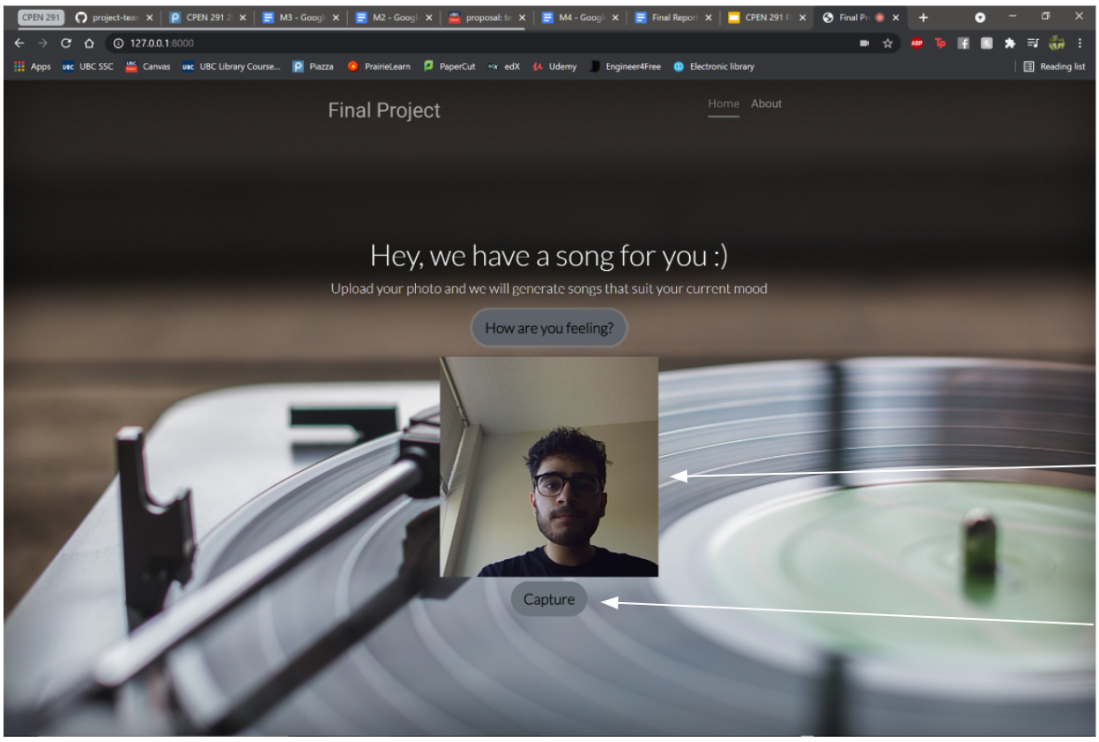
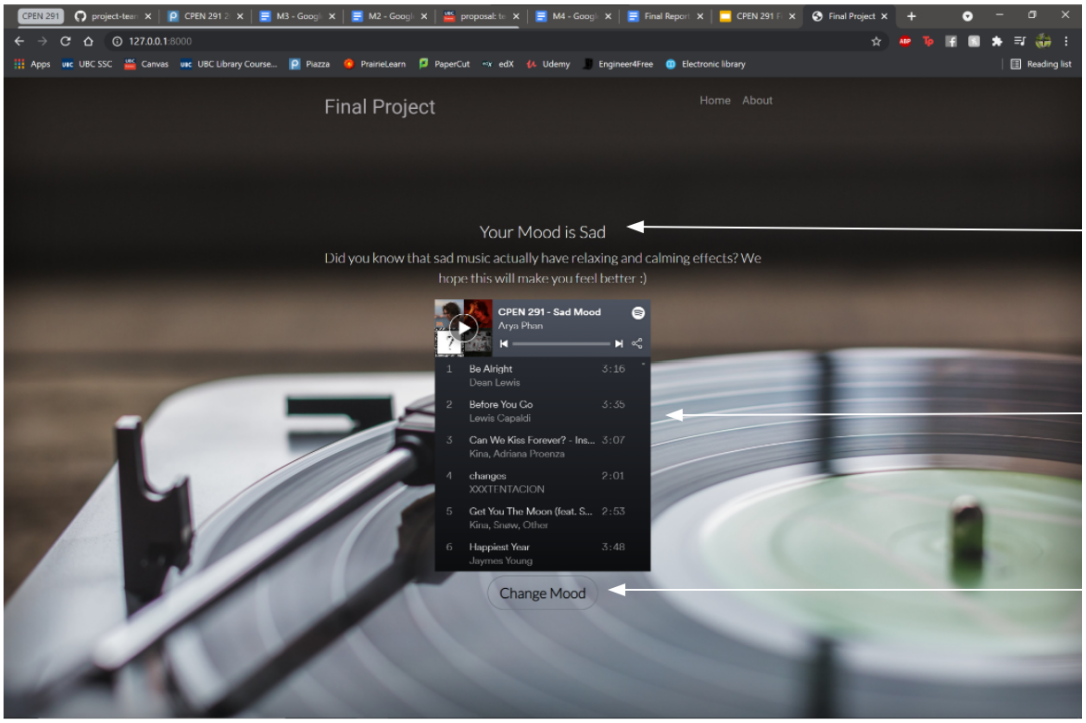


Image displaying what the webcam is capturing

Button to capture image



You mood and message below

Spotify Playlist for the specific mood

Button to go back to homepage and start process

Video Link for Discussion and Demo of project:

https://youtu.be/dTJ-ppo-kN0