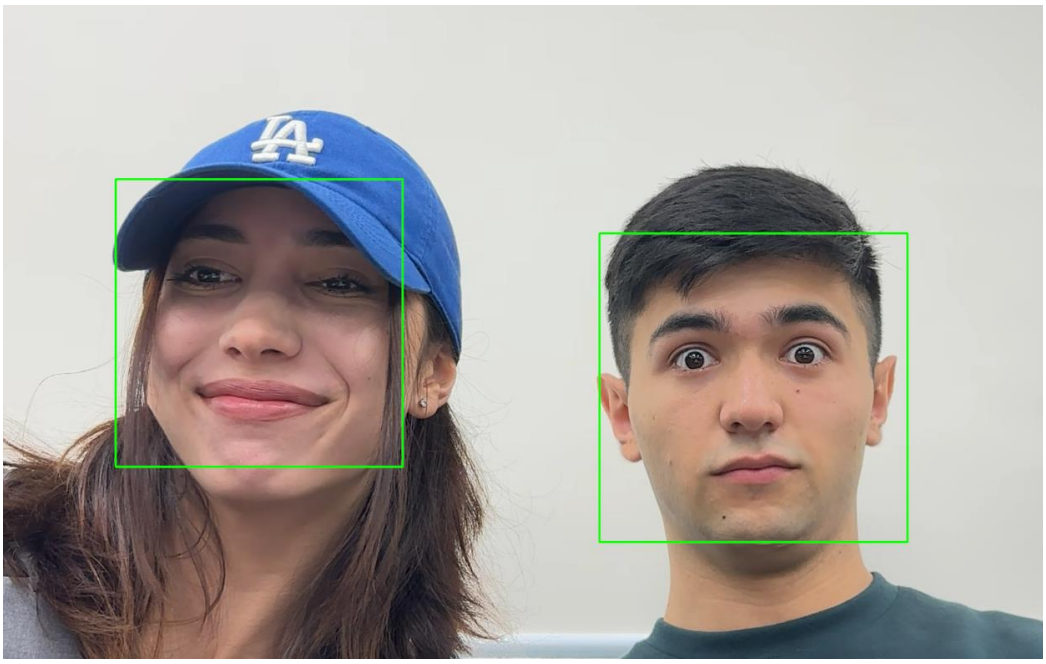


FACE DETECTION USING VJ ALGORITHM

REPORT FILE

CONTRIBUTORS

- Nartozhieva Nigara
  12200276
  Akhrorov Temurbek
  12204574

Course

DIGITAL IMAGE PROCESSING by KAKANI VIJAY

Collecting datasets:

1. Positive

First, we found a positive dataset for face recognition from the Kaggle website (https://www.kaggle.com/datasets/atulanandjha/lfwpeople/).

Then, to extract all the images from multiple nested files and gather them into one folder we used Python with the Pillow library:

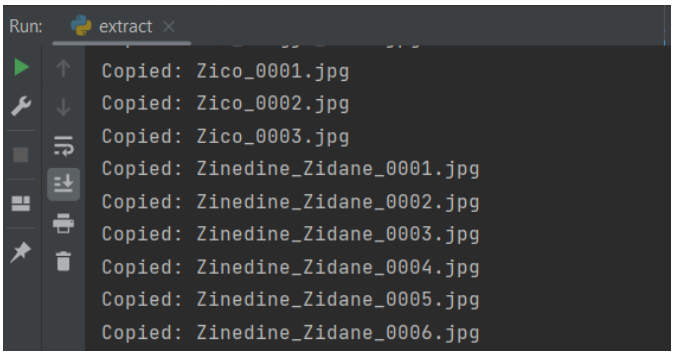
1. Install Pillow using pip:

pip install Pillow

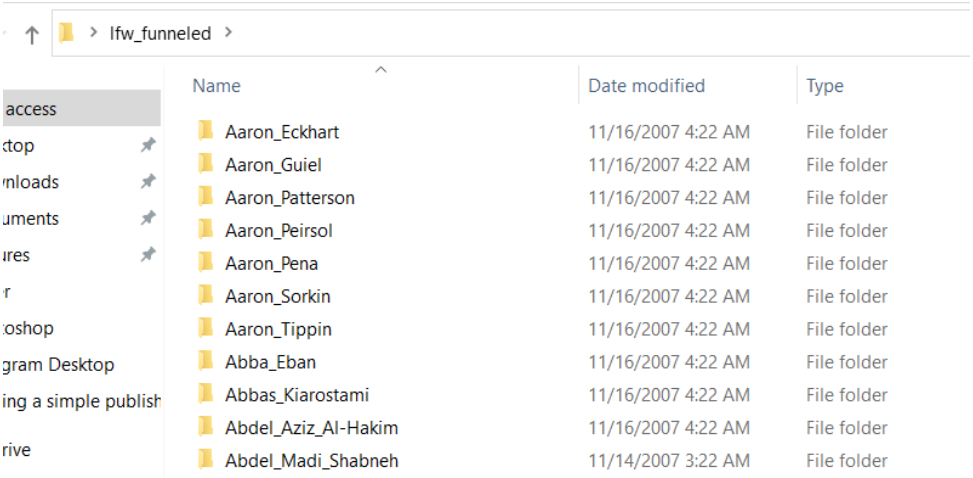
2. The code to extract:

```
import os
from PIL import Image
import shutil

# Set the source directory containing nested files and folders
source_directory = '../../Desktop/lfw_funneled'

# Set the destination directory to collect all images
destination_directory = '../../Desktop/p'

# Ensure the destination directory exists
os.makedirs(destination_directory, exist_ok=True)

# Function to extract and copy images
def extract_images(src_dir, dest_dir):
    for root, dirs, files in os.walk(src_dir):
        for file in files:
            # Check if the file is an image (you can add more image formats
as needed)
            if file.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp',
'.tif', '.tiff')):
                img_path = os.path.join(root, file)
                with Image.open(img_path) as img:
                    img.save(os.path.join(dest_dir, file))
                    print(f'Copied: {file}')

# Call the function to extract and copy images
extract_images(source_directory, destination_directory)
```

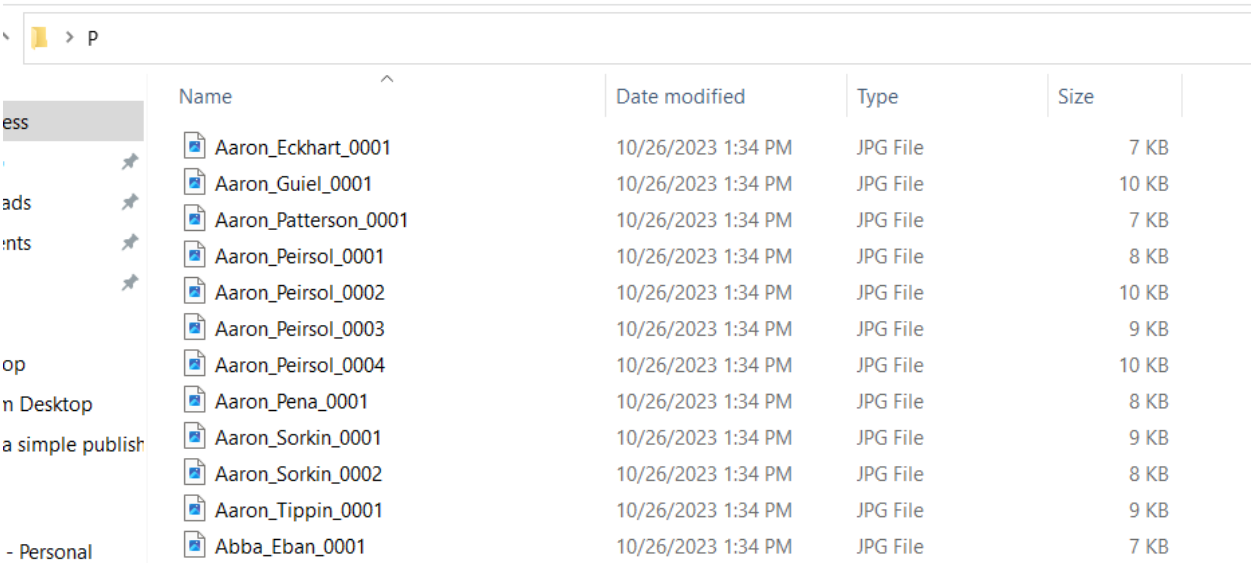
3. Output:



BEFORE:



AFTER:



Obviously, 13233 images are quite a lot to process, so we decided to reduce the number of images to 1500. Thus, I used python code to randomly choose 11333 images and delete them:

```
import os
import random

# Set the directory where the images are located
directory = '../../Desktop/p'

# Get a list of all image files in the directory
image_files = [file for file in os.listdir(directory) if
file.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tif',
'.tiff'))]

# Specify the number of images to delete
images_to_delete = 11333

# Ensure the number of images to delete is not greater than the total number
of images
images_to_delete = min(images_to_delete, len(image_files))

# Randomly select images to delete
images_to_delete_indices = random.sample(range(len(image_files)),
images_to_delete)

# Delete the selected images
for index in sorted(images_to_delete_indices, reverse=True):
    file_to_delete = os.path.join(directory, image_files[index])
    os.remove(file_to_delete)
    print(f'Deleted: {image_files[index]}')

print(f'Total {images_to_delete} images deleted.')
```

Negative:

Link for negative dataset for face recognition: (https://www.kaggle.com/datasets/puneet6060/intel-image-classification) with 24335 images of buildings, landscapes, streets, animals, indoor objects, geometric shapes and we included our own negative samples by taking images of our classroom which is more than enough for 1:5 ratio of positive and negative image samples.

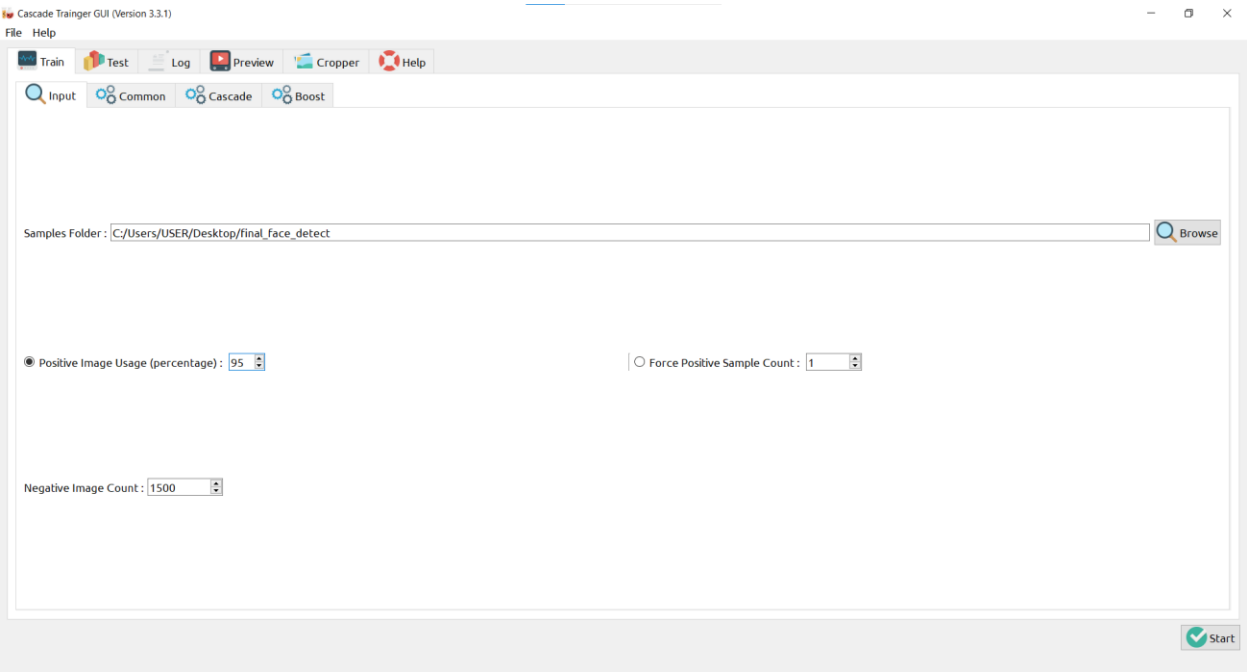
Our own images for dataset:

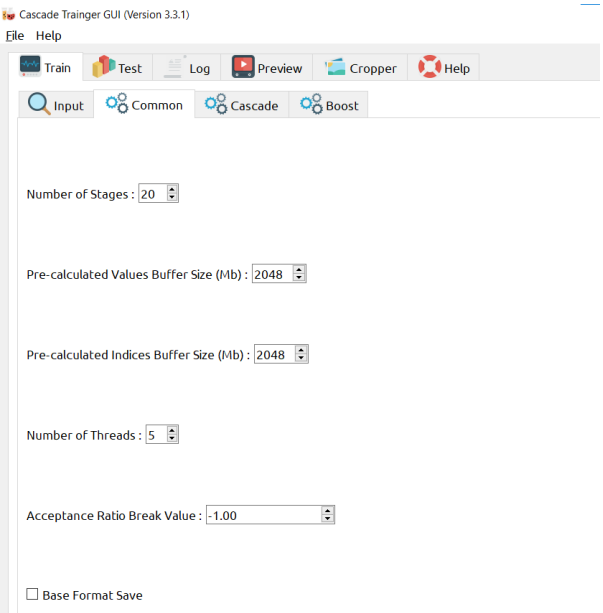
Initially, we had our own 200 positive image samples with 1280x720 pixels, and to fasten the process of training we reduced the size of the images to 250x250 pixels using the following code:

```
from PIL import Image
import os

# Set the directory containing the images
input_directory = r'C:\Users\USER\Desktop\n2'

# Set the output directory where resized images will be saved
output_directory = r'C:\Users\USER\Desktop\new_1\n'

# Set the desired width and height
new_width = 250
new_height = 250

# Create the output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

# Iterate through all image files in the input directory
for filename in os.listdir(input_directory):
    if filename.endswith(('.jpg', '.jpeg', '.png', '.bmp', '.gif')):
        # Open the image
        with Image.open(os.path.join(input_directory, filename)) as img:
            # Resize the image
            img = img.resize((new_width, new_height), Image.ANTIALIAS)
            # Save the resized image to the output directory
            img.save(os.path.join(output_directory, filename))

print("Images resized and saved to", output_directory)
```

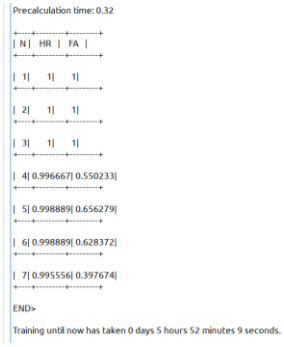
Training:

After collecting all datasets, we started training the datasets for face detection using VIOLA JONES Algorithm by using Cascade-Trainer-GUI:





The training process took almost 6 hours with 20 stages:



Running the xml file:

```
import cv2

# Load the trained cascade classifier for face detection
face_cascade = cv2.CascadeClassifier('repository of the xml file')

# Open a connection to the camera (0 is typically the built-in webcam, but you can
change it to the appropriate camera index)
cap = cv2.VideoCapture(0)

while True:
    # Read a frame from the camera
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to grayscale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.01, minNeighbors=5,
minSize=(100, 100))

    # Draw rectangles around detected faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the frame with detected faces
    cv2.imshow('Face Detection', frame)

    # Break the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

This is code for using the xml file in real-time VOD mode.

⚫ Problem 1: Find the minimum detectable size of faces

⚫ Problem 2: Find the maximum detectable size of faces

⚫ Problem 3: Find the average detection time

▪ Per image

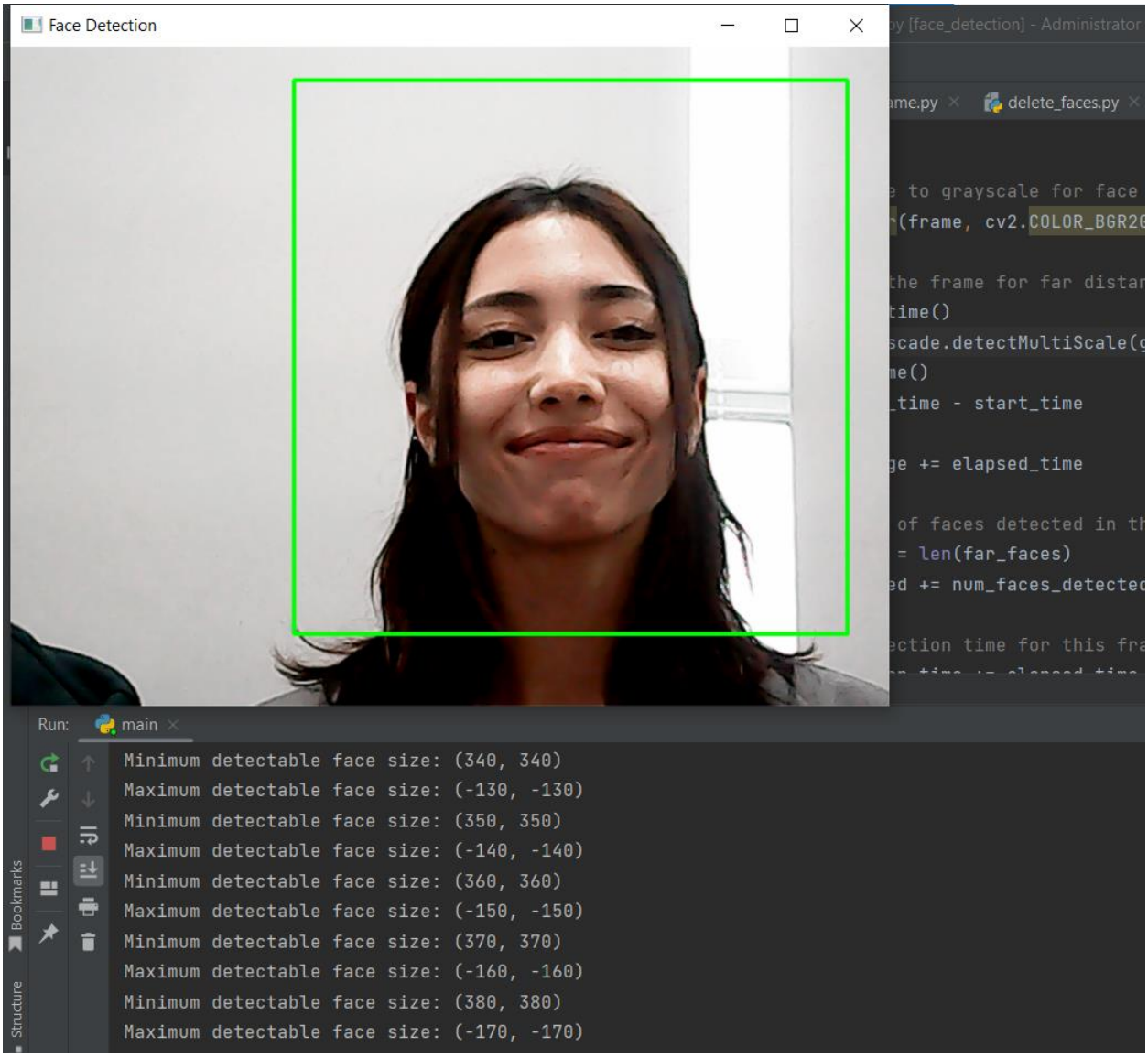
▪ Per face

Code for these problems:

```
import cv2
import time

# Load the trained cascade classifiers for close and far distance face
detection
far_cascade =
cv2.CascadeClassifier(r'C:\Users\USER\Desktop\final_face_detect\classifier\ca
scade_final.xml')

# Open a connection to the camera (0 is typically the built-in webcam, but
you can change it to the appropriate camera index)
cap = cv2.VideoCapture(0)

min_face_size = (10, 10)  # Starting minimum size
max_face_size = (200, 200)  # Starting maximum size

total_time_per_image = 0
total_face_detection_time = 0
total_faces_detected = 0
num_frames = 0

while True:
    num_frames += 1

    # Read a frame from the camera
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to grayscale for face detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame for far distance
    start_time = time.time()
    far_faces = far_cascade.detectMultiScale(gray, scaleFactor=1.01,
minNeighbors=3, minSize=(120, 120))
    end_time = time.time()
    elapsed_time = end_time - start_time
```

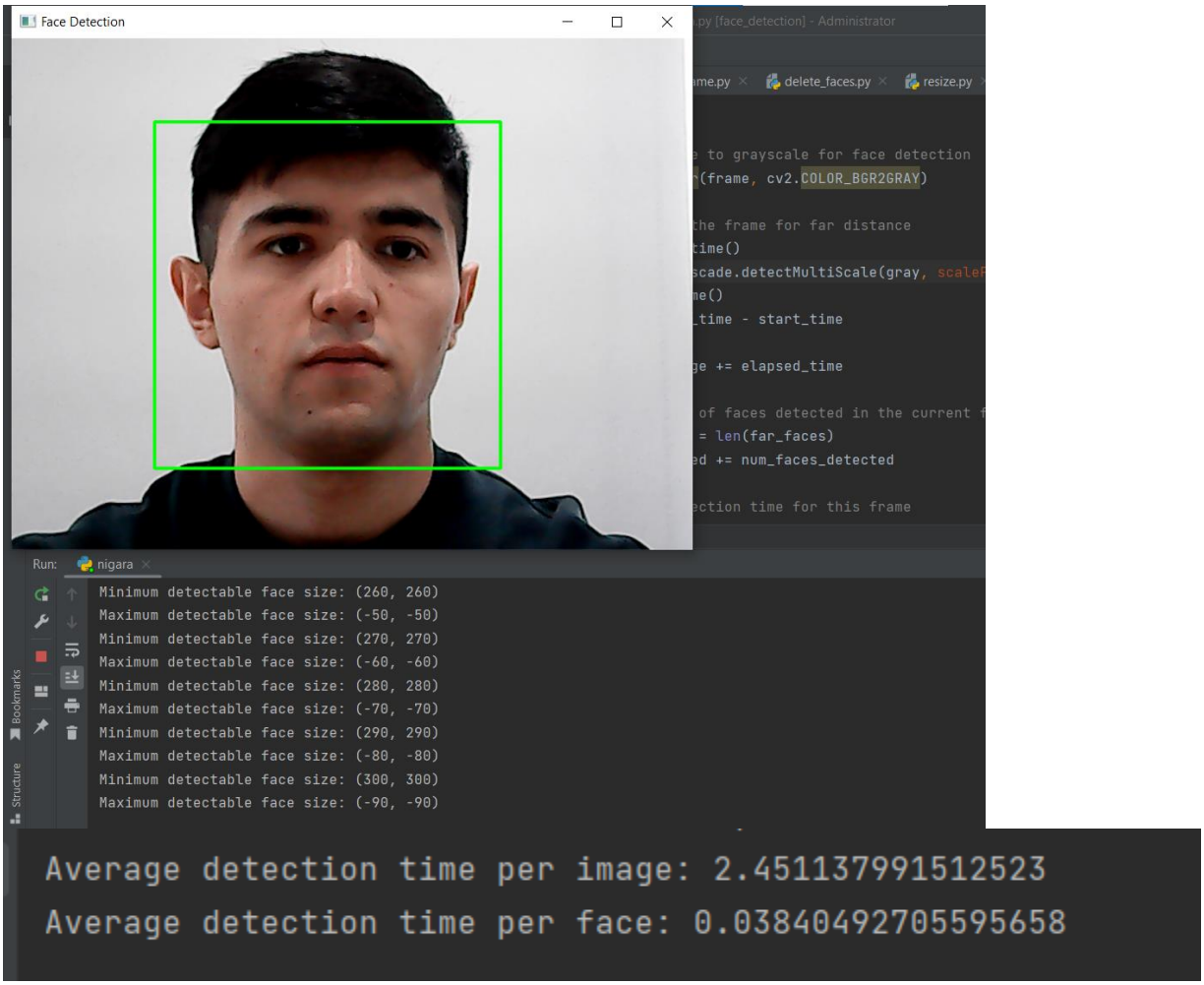
```
    total_time_per_image += elapsed_time

    # Count the number of faces detected in the current frame
    num_faces_detected = len(far_faces)
    total_faces_detected += num_faces_detected

    # Sum the face detection time for this frame
    total_face_detection_time += elapsed_time

    # Draw rectangles around detected far-distance faces
    for (x, y, w, h) in far_faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Display the frame with detected faces
    cv2.imshow('Face Detection', frame)

    # Break the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Update minimum and maximum face sizes
    if num_frames == 30:
        print("Min face size:", min_face_size)
        print("Max face size:", max_face_size)
        min_face_size = (min_face_size[0] + 10, min_face_size[1] + 10)
        max_face_size = (max_face_size[0] - 10, max_face_size[1] - 10)
        num_frames = 0

# Calculate averages
average_time_per_image = total_time_per_image / num_frames
average_time_per_face = total_face_detection_time / total_faces_detected

print("Average detection time per image:", average_time_per_image)
print("Average detection time per face:", average_time_per_face)

# Release the video capture object and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Output of the first version:




```
Minimum detectable face size: (490, 490)
Maximum detectable face size: (-280, -280)
Average detection time per image: 3.261000871658325
Average detection time per face: 0.131315471341946
```

Output of the second version:



Moreover, we had two different versions of xml files, this one above is the first, and the second one included 10000+ negative image samples and 1500+ positive images. Each has its own advantages and drawbacks.

So, first version detects human faces perfectly in far distances and when person approaches a camera closer, detection fails.

Turning to second version, it detects faces very precisely in any distance, however, sometimes it detects other objects.

Demonstration and explanations will be in the Demo video.

Challenges:

1. Challenges in training:

Failure in image files names. In training process, we had issues with file naming and the GUI just would not proceed with training. To solve this problem we used python code that would rename all the pictures in the file starting from 1.

```
import os

# Set the directory containing your image files
directory = r'C:\Users\USER\Desktop\n2'

# List all files in the directory
files = os.listdir(directory)

# Sort the files to ensure they are processed in order
files.sort()

# Counter for the new file names
counter = 1

# Loop through the files and rename them
for filename in files:
    # Get the file extension
    file_extension = os.path.splitext(filename)[1]

    # New name for the file
    new_name = f"{counter}{file_extension}"

    # Build the full paths for the old and new names
    old_path = os.path.join(directory, filename)
    new_path = os.path.join(directory, new_name)

    # Rename the file
    os.rename(old_path, new_path)

    # Increment the counter
    counter += 1

print("All files have been renamed.")
```

2. Faces in negative image samples:

We had a large number of negative image samples with small faces, and going through each image is almost impossible. To solve the case, we used this code that would use haarcascade_frontalface_default.xml and detect faces in the file and delete those images.

```
import os
import cv2

# Load a pre-trained face detection classifier
```

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


# Function to check if an image contains a face
def has_face(image_path, min_face_size=100):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(min_face_size, min_face_size))

    # If any faces are found, consider the image as having a face
    return len(faces) > 0


# Define the directory where your images are located
image_directory = r'C:\Users\USER\Desktop\new_1000&5000\n'

# Minimum face size threshold (you can adjust this)
min_face_size_threshold = 100  # Adjust as needed

# Loop through all the images in the directory
for filename in os.listdir(image_directory):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        image_path = os.path.join(image_directory, filename)
        if has_face(image_path, min_face_size_threshold):
            print(f"Deleting {filename} because it contains a face.")
            os.remove(image_path)
        else:
            print(f"{filename} does not contain a face.")

print("Image processing completed.")
```

3. OpenCV Error: Bad Argument (Insufficient number of positive images)

We played with settings and decreased the percentage of the Positive Image Usage from 100 to 90. Thus, the problem was solved.

Conclusion:

As part of this project, our team used the Viola-Jones algorithm and the Cascade Trainer GUI to create a custom XML file for face identification. We had to overcome a number of obstacles, such as building our own datasets of positive and negative images, organizing and preprocessing sizable datasets, and identifying and eliminating false positives in the negative samples. The procedure required considerable problem-solving, hundreds of picture renamings, and image resizing. Our endeavor took place over the course of five or six days, and teamwork was essential to reaching our objective. We learnt a great deal about the complexities of object detection during the project, as well as how to properly use machine learning algorithms for real-world applications.