COMP6200 Machine Learning

Homework 5

Name: Ibna Kowsar
Affiliation: MSCS
Submission Date: 02/28/2024

Table of contents

Introduction:

In this homework, we delve into the development of a real-time face recognition system leveraging the built-in camera of a computer. The system is specifically designed to recognize three distinct individuals, labeled as A, B, and C. The process involves several key steps, starting with the detection of ten faces for each person under varying lighting and background conditions, subsequently numbering them as faceA0 through faceA9 for person A, faceB0 through faceB9 for person B, and similarly for person C. These detected faces are then uniformly resized to a predetermined dimension, ensuring consistency across all samples.

Further, we compute the average face for each person by aggregating their respective resized faces and dividing the resultant sum by ten, thereby obtaining standard faces, denoted as stdFaceA, stdFaceB, and stdFaceC. For the face recognition task, we employ a straightforward image difference technique. A newly detected face is resized to match the standard faces' dimensions, followed by a comparison against the standard faces of A, B, and C through image subtraction. The matching error is quantified as the sum of the absolute pixel value differences, with the smallest error indicating the closest match.

This homework adopts an Object-Oriented Programming (OOP) approach to structure the code, enhancing its modularity, maintainability, and extendability. By encapsulating related functionalities within classes, such as image processing and face recognition, the system is designed to be both efficient and scalable.

Implementation:

For this task, I collect images of three people (10 for each) as displayed in Table 01. Here for the simplicity of the report writing (illustration), I changed the images to small shapes and the images show that each image is collected from different lighting conditions and different scenarios.

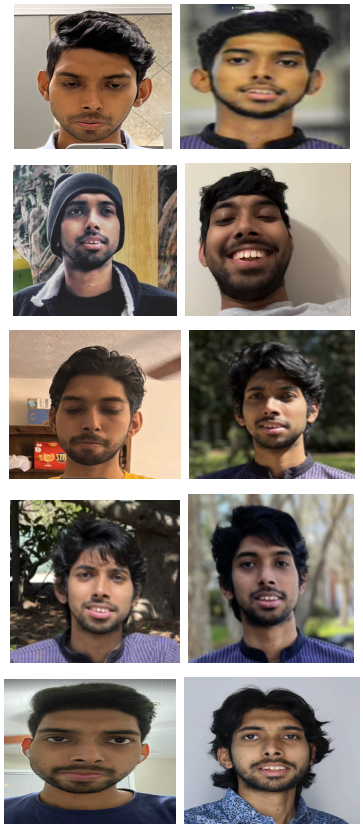
Person A	Person B	Person C
		

Table 1: Input images of three different people on different conditions

Once I have the input images ready I resized them to 100*100 pixels and I used the bilinear interpolation technique. I know that Lanczos interpolation algorithm gives better results but in real life, it is slower than bilinear, bicubic kind of interpolation techniques. Hence, for this task, I

used bilinear, a simple interpolation technique to resize my input facial images as shown in Figure 1.

```
def resize_faces(self):
    for person in self.person_names:
        for i in range(10):
            img_path = f"{self.base_dir}images/face{person}{i}.png"
            img = Image.open(img_path)
            img = img.resize((100, 100), Image.BILINEAR)
            img.save(f"{self.resized_dir}face{person}{i}.png")
```

Figure 01: Resizing input images for all three people

Once I have the resized images now I can calculate the mean of all the images of each individual and save the standard deviation image in a directory. For this task, I first add all the images for each person and then calculate the average by dividing by the total number of images (i.e., in this case total number of images is 10). To add all the images first I converted the images to float32 format to do pixel addition and then I used the cv2.add() function and saved it to sum_img global variable. Once I am done calculating the average image I convert the float32 version image to uint8 format and hence we get the stdfaceA, stdfaceB, and stdfaceC.

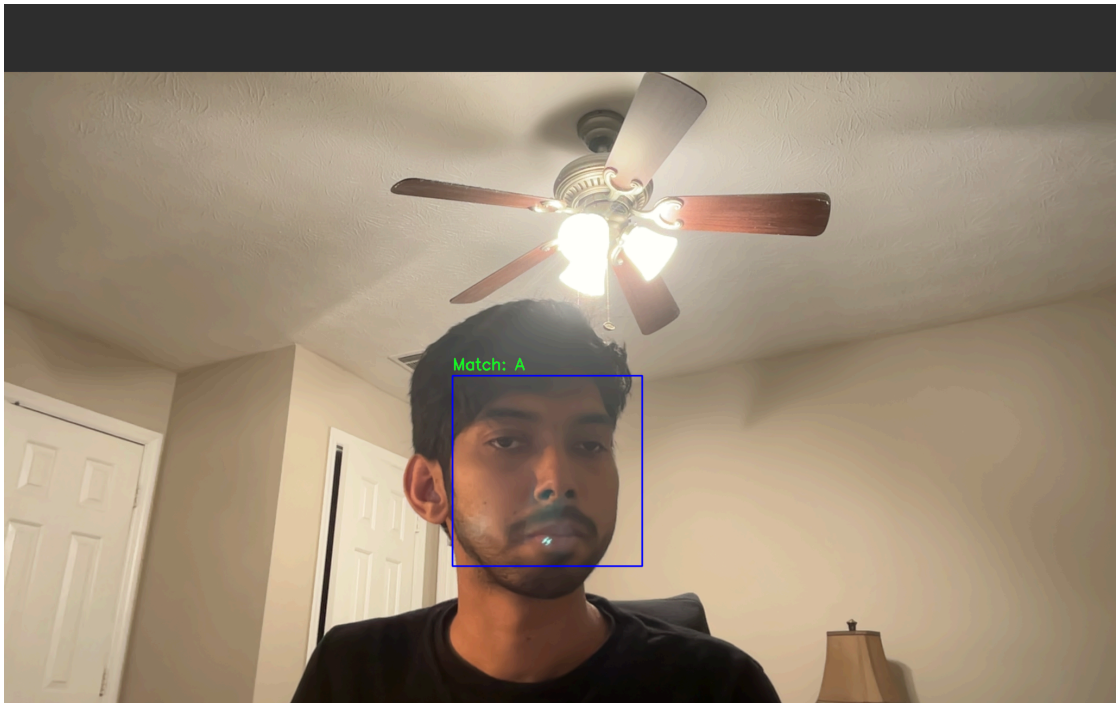
```
def find_average_face(self, num_images=10):
    for person in self.person_names:
        sum_img = None
        valid_images = 0
        for i in range(num_images):
            img_path = f"{self.resized_dir}face{person}{i}.png"
            img = cv2.imread(img_path)
            if img is None:
                print(f"Warning: Image at '{img_path}' could not be loaded.")
                continue
            img = img.astype(np.float32)
            if sum_img is None:
                sum_img = img
            else:
                sum_img = cv2.add(sum_img, img)
            valid_images += 1

        if valid_images > 0:
            avg_img = (sum_img / valid_images).astype(np.uint8)
            cv2.imwrite(f"{self.std_dir}stdFace{person}.png", avg_img)
            print(f"Average face for {person} saved.")
        else:
            print(f"No valid images processed for {person}.")
```

Figure 02: Adding all the images and calculating the average face for each person

Person A	Person B	Person C
		

Figure 03: Output images for each person after calculating the average face

Now that we have the average facial images saved we can use this information to detect the face of these people using our real-time face detection algorithm. Figure 04 shows that the real-time face matches with persons A, B, and C accordingly which is correct. However, I ran this experiment multiple times and it seems I did not get accurate results always and it is dependent on the facial direction, movement, and even light as shown in Figure 05. I also show that if a new person D is selected for face recognition using my algorithm, it gives wrong results and recognizes as a random person from Person A, B, or C.



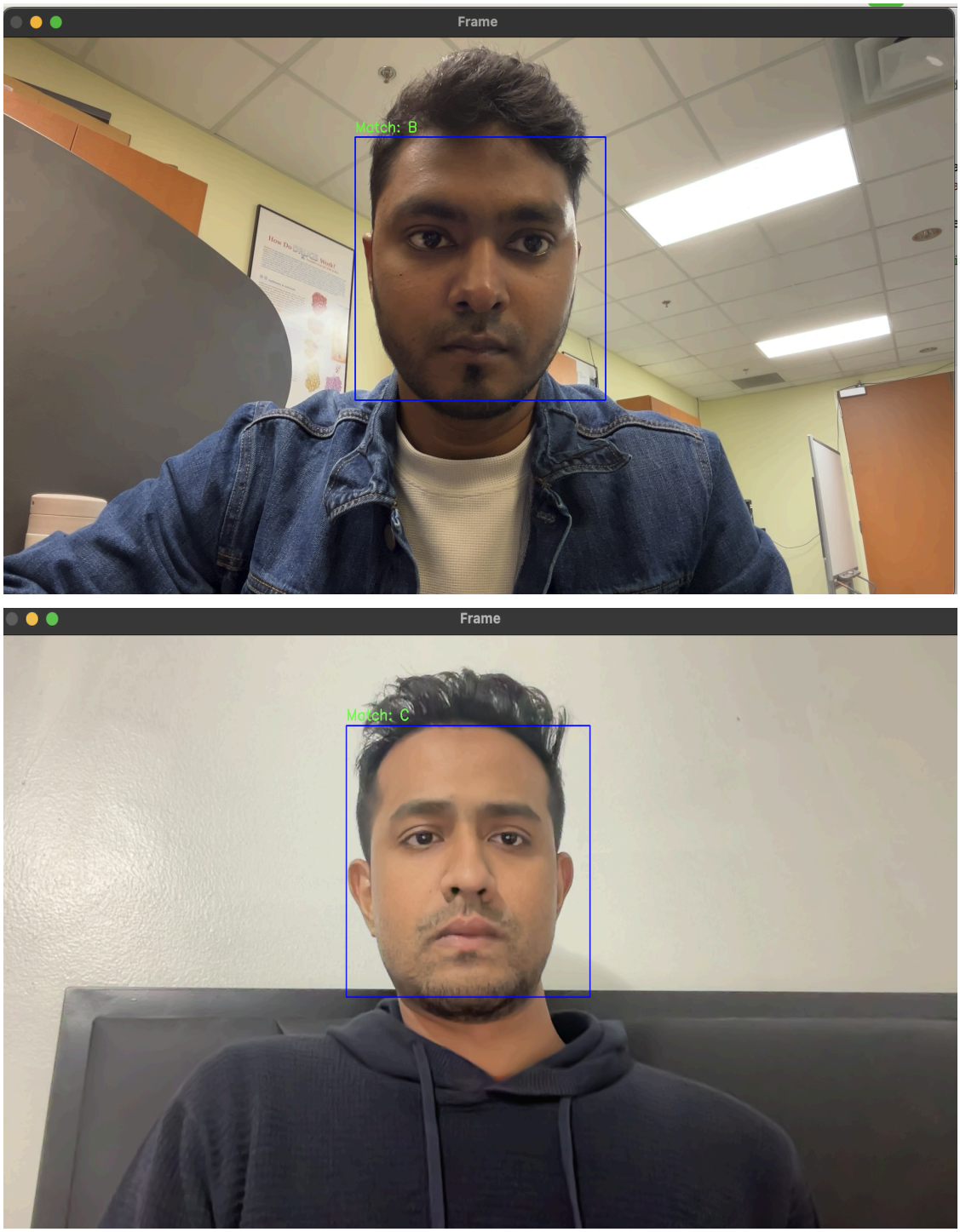


Figure 04: Face detection in real-time using the saved average facial images

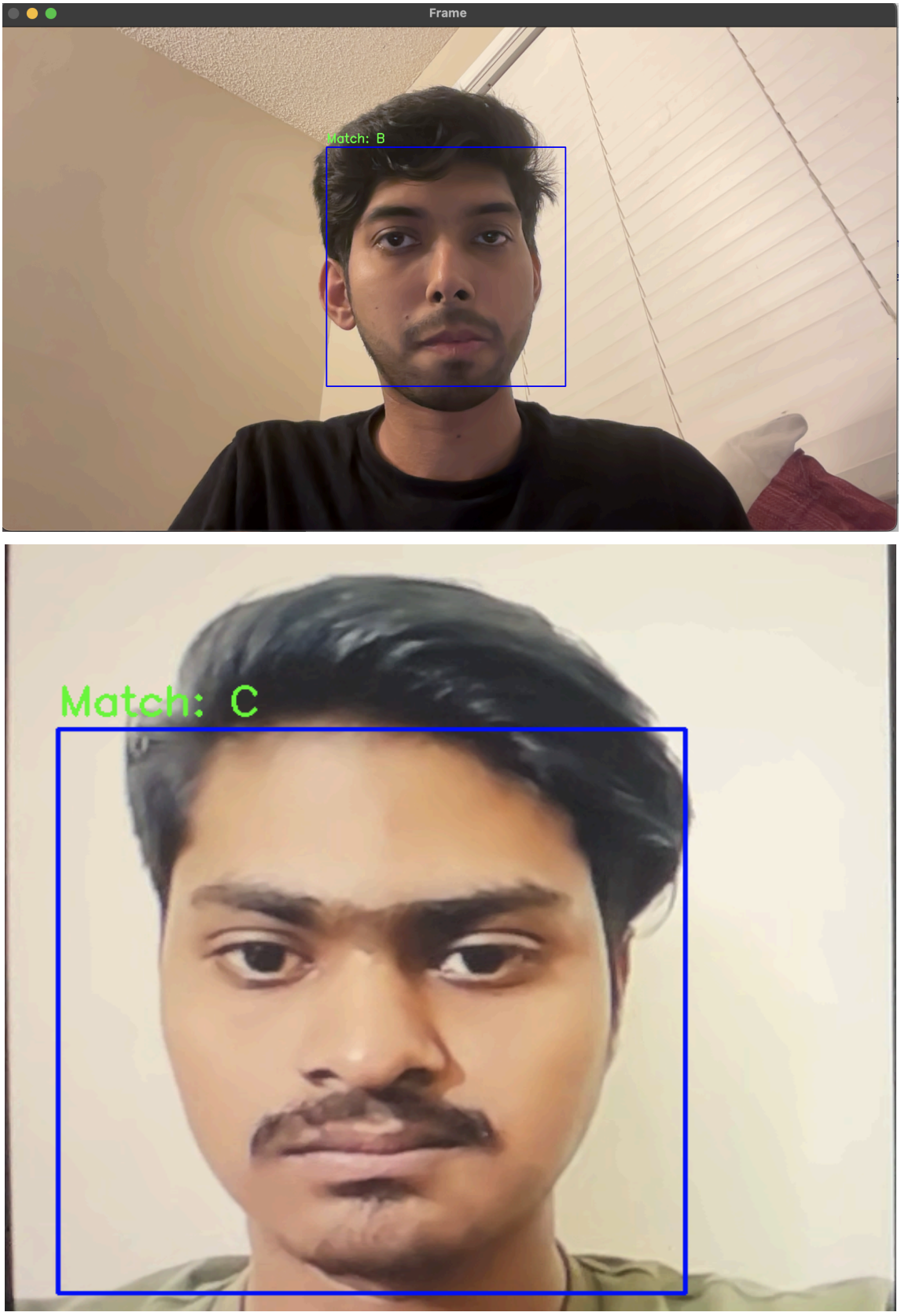


Figure 04: Face detected wrong in real-time

To detect the exact match of the image I first convert the image to grayscale and then resize it to 100*100 so that we can fairly compare it with our saved stdfaceA, stdfaceB and stdfaceC. Therefore, once I get a frame I compare it with all the stdfaceX images by calculating the absolute difference between the std_face image and real-time captured image. I also kept track of the error for each average face and finally whichever has the lowest error is the matched person. Therefore as Figure 04 shows the real-time frame matches with Person A, B and C.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = self.face_cascade.detectMultiScale(gray, 1.1, 4)

for (x, y, w, h) in faces:
    face = gray[y:y+h, x:x+w]
    face_resized = cv2.resize(face, self.face_size)
    min_error = np.inf
    match = None
    for person, std_face in self.std_faces.items():
        error = np.sum(cv2.absdiff(face_resized, std_face))
        if error < min_error:
            min_error = error
            match = person

    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.putText(frame, f'Match: {match}', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)
```

Figure 05: Sample code to show the calculation of absolute differences between images and matching the face

Conclusions:

In conclusion, this homework provided hands-on experience in building a simple yet effective real-time face recognition system. By focusing on the foundational aspects of image processing and recognition—such as face detection under varying conditions, image resizing, averaging, and difference calculation—we laid the groundwork for understanding more complex recognition algorithms. Adopting an Object-Oriented Programming (OOP) framework enhanced the project's structure, making it more manageable and adaptable for future enhancements. This exercise fortified our skills in applying basic computer vision techniques and highlighted the significance of programming paradigms in developing robust and efficient applications.

Appendix A: Source code

```
 from PIL import Image
import numpy as np
import cv2
import os
#Image processing for facial image recognition
class ImageProcessor:
       def   __init__(self,   person_names,   base_dir='./',   resized_dir='./resized/',
std_dir='./std_img/'):
       self.person_names = person_names
       self.base_dir = base_dir
       self.resized_dir = resized_dir
       self.std_dir = std_dir
       if not os.path.exists(self.resized_dir):
           os.makedirs(self.resized_dir)
       if not os.path.exists(self.std_dir):
           os.makedirs(self.std_dir)

   def resize_faces(self):
       for person in self.person_names:
           for i in range(10):
               img_path = f"{self.base_dir}images/face{person}{i}.png"
               img = Image.open(img_path)
               img = img.resize((100, 100), Image.BILINEAR)
               img.save(f"{self.resized_dir}face{person}{i}.png")

   def find_average_face(self, num_images=10):
       for person in self.person_names:
           sum_img = None
           valid_images = 0
           for i in range(num_images):
               img_path = f"{self.resized_dir}face{person}{i}.png"
               img = cv2.imread(img_path)
               if img is None:
                   print(f"Warning: Image at '{img_path}' could not be loaded.")
                   continue
               img = img.astype(np.float32)
               if sum_img is None:
                   sum_img = img
               else:
                   sum_img = cv2.add(sum_img, img)
```

```
                valid_images += 1

            if valid_images > 0:
                avg_img = (sum_img / valid_images).astype(np.uint8)
                cv2.imwrite(f"{self.std_dir}stdFace{person}.png", avg_img)
                print(f"Average face for {person} saved.")
            else:
                print(f"No valid images processed for {person}.")



#Face recognition from average image
class FaceRecognizer:
    def __init__(self, std_faces, face_size=(100, 100)):
        self.std_faces = std_faces
        self.face_size = face_size
                self.face_cascade   =   cv2.CascadeClassifier(cv2.data.haarcascades   +
'haarcascade_frontalface_default.xml')

    def load_standard_faces(self, std_dir):
        for person in self.std_faces.keys():
            std_face = cv2.imread(f'{std_dir}stdFace{person}.png', cv2.IMREAD_COLOR)
            if std_face is not None:
                self.std_faces[person] = cv2.cvtColor(std_face, cv2.COLOR_BGR2GRAY)
            else:
                print(f"Error loading standard face for person {person}")

    def recognize_face_real_time(self):
        cap = cv2.VideoCapture(0)

        while True:
            ret, frame = cap.read()
            if not ret:
                print("Failed to grab frame")
                break

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = self.face_cascade.detectMultiScale(gray, 1.1, 4)

            for (x, y, w, h) in faces:
                face = gray[y:y+h, x:x+w]
                face_resized = cv2.resize(face, self.face_size)
```

```
                min_error = np.inf
                match = None
                for person, std_face in self.std_faces.items():
                    error = np.sum(cv2.absdiff(face_resized, std_face))
                    if error < min_error:
                        min_error = error
                        match = person

                cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
                                cv2.putText(frame,  f'Match:  {match}',  (x,  y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)

            cv2.imshow('Frame', frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        cap.release()
        cv2.destroyAllWindows()

#Diver Code
person_names = ['A', 'B', 'C']
std_faces = {name: None for name in person_names}

# Initialize the ImageProcessor and process images
image_processor = ImageProcessor(person_names)
image_processor.resize_faces()
image_processor.find_average_face()

# Initialize the FaceRecognizer, load standard faces, and start real-time recognition
face_recognizer = FaceRecognizer(std_faces)
face_recognizer.load_standard_faces(image_processor.std_dir)
face_recognizer.recognize_face_real_time()
```