Computer Vision Project Attendance System Based on Live Face Recognition

1st Lawysen
School Of Computer Science
Bina Nusantara University
Jakarta 11480, Indonesia
lawysen@binus.ac.id

2nd Mario
School Of Computer Science
Bina Nusantara University
Jakarta 11480, Indonesia
mario007@binus.ac.id

3rd Anggara, N.
School Of Computer Science
Bina Nusantara University
Jakarta 11480, Indonesia
nelsen.anggara@binus.ac.id

4th Cenggoro, W.
Computer Science Department,
School of Computer Science
Bina Nusantara University
Jl. Kebun Jeruk Raya No. 27, Jakarta
11530, Indonesia

***Abstract*— Technology continues to experience rapid development from time to time, therefore the various systems that exist in society are also developing. One area in particular that is significantly affected is computer vision. Nowadays computers can detect or recognize an existing object very quickly as demonstrated by various algorithms like face recognition technology. Face Recognition is a technology to detect and identify faces based on a person's facial features [1]. By utilizing face recognition, we created an automatic attendance system by recording the detected face's name and the current time in a file. This attendance system was coded using Python programming language within the Visual Studio Code application, along with various available Python libraries. In addition, we also used hardware, namely a webcam, to take live pictures of someone's face.**

Keywords*— *face recognition, machine learning, attendance system, computer science

I. INTRODUCTION

Face recognition technology has the potential to be used in a number of different ways that can benefit society. For example, it can be used to improve public safety by helping law enforcement agencies identify and track criminal suspects. It can also be used in the healthcare industry to quickly and accurately identify patients and ensure that they receive the correct treatment. In addition, facial recognition can be used in the financial industry to improve security and prevent fraud by verifying the identity of individuals conducting financial transactions. In workplaces, facial recognition is used to speed up employee attendance checks and ensure that outsiders cannot use office computers through webcams. Overall, facial recognition can help to improve efficiency, reduce costs, and increase security in a variety of different fields.

Oftentimes, a majority of people take the facial recognition process for granted. There are undoubtedly complex algorithms in place that make the system work. Therefore, as computer science students, we take it upon ourselves to code a simple and functional facial recognition system from scratch. This will allow us to truly understand the various processes involved in processing and identifying images of faces. Grasping such a basic yet important practical application of computer vision will enable us to broaden our technical understanding and potentially expand our skills to be applied for wider scenarios. As with many other computer vision projects, our work will be done using the Python programming language.

II. METHOD

A. Face Recognition Practice

Before creating a fully functional facial recognition system that can record detected faces and the current time, we first had to perform a little experiment to grasp how the face_recognition library works.

```
Import cv2
Import numpy as np
Import face_recognition
```

Fig. 1. Import Libraries

- "cv2" → common infrastructure for computer vision applications.
- "numpy" → several mathematical expressions and functions.
- "face_recognition" → specialised functions for recognizing and manipulating facial features.

```
imgElon =
face_recognition.load_image_file('LOKAS
I GAMBAR')
imgElon = cv2.cvtColor(imgElon,
cv2.COLOR_BGR2RGB)
imgTest =
face_recognition.load_image_file('LOKAS
I GAMBAR TEST')
imgTest = cv2.cvtColor(imgTest,
cv2.COLOR_BGR2RGB)
```

Fig. 2. Converting fetched images into RGB color format

- "face_recognition.load_image_file(...)" → to store images from the given file directories into the variables imgElon and imgTest.
- "cv2.cvtColor(..., cv2.COLOR_BGR2RGB)" → to convert images from BGR to RGB color space.

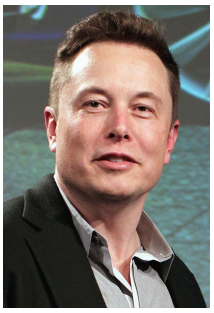

Fig. 3. “ingElon”

Fig. 4. “imgTest” before undergoing the facial recognition process

```
faceLoc =
face_recognition.face_locations(imgElon
)[0]
encodeElon =
face_recognition.face_encodings(imgElon
)[0]
cv2.rectangle(imgElon, (faceLoc[3],
faceLoc[0]), (faceLoc[1], faceLoc[2]),
(255, 0, 255), 2)

faceLocTest =
face_recognition.face_locations(imgTest
)[0]
encodeTest =
face_recognition.face_encodings(imgTest
)[0]
cv2.rectangle(imgTest, (faceLocTest[3],
faceLocTest[0]), (faceLocTest[1],
faceLocTest[2]), (255, 0, 255), 2)
```

Fig. 5. Determining the face’s location in imgElon and imgTest

- “face_recognition.face_locations(...)[0]” → to detect the location of the face in a rectangle. Returns the 4 coordinates of the rectangle’s corners.
  - The face_recognition.face_locations function make use of the HOG (Histogram of Oriented Gradients) Based Model. HOG is one among many computer vision algorithms to detect and locate objects within a digital image. This algorithm starts by dividing an image into smaller parts called cells. Then, the gradient histograms from each cell is calculated before being combined to create a feature vector. This feature vector will be used for object detection in images [2, 3].
- “face_recognition.face_encodings(...)[0]” → returns a numpy array, which stores the numerical representation of a face’s unique characteristics.
- “cv2.rectangle(..., (..., …), (..., …), (255, 0, 255), 2)” → visualizes the rectangle using the coordinates of the rectangle corners obtained, giving it a 2-pixel thick purple border.

```
results =
face_recognition.compare_faces([encodeE
lon], encodeTest)
faceDis =
face_recognition.face_distance([encodeE
lon], encodeTest)
```

Fig. 6. Checking face similarity between imgElon and imgTest

- “face_recognition.compare_faces([...], …)” → returns true/false depending on whether or not each face encoding in the list matches the face encoding to compare.
- “face_recognition.face_distance([...], …)” → returns a list of euclidean distances between each face encoding in the list and the face encoding to compare. Smaller values mean more similar face encodings.
  - From the several distance measurements that can be used in computer vision, the face_recognition.face_distance function makes use of the Euclidean distance. The Euclidean distance between any two points on a 2-dimensional plane is the straight line between them. The Euclidean distances will be used to measure the similarity of 2 images. The formula to obtain a euclidean distance between pixel coordinate (x1, y1) from the first image and pixel coordinate (x2, y2) from the second image is: d = √((x2 - x1)^2 + (y2 - y1)^2). This results of the calculation will return a measure of the difference between the two images in terms of color, intensity and shape. The smaller the Euclidean Distance, the more similar the two images are [4, 5].

```
cv2.putText(imgTest, f'{results}
{round(faceDis[0], 2)}', (50,50),
cv2.FONT_HERSEY_COMPLEX,1,(0,0,255), 2)
cv2.imshow('Elon Mush', imgElon)
cv2.imshow('Elon Test', imgTest)
cv2.waitKey(0)
```

Fig. 7. Displaying the result of facial recognition, showing that the face of imgElon can be found in imgTest.

- “cv2.putText(..., f'...', (50,50), cv2.FONT_HERSEY_COMPLEX,1,(0,0,255), 2)” → Add a 2-pixel thick, normal-sized, red text (of the font “heresy complex”) on an image. The text is located 50 pixels from the image’s left edge and 50 pixels from the image’s top edge. What’s written in the text will be whatever is inside “ f’...’ “.

- "cv2.imshow('...', …)" → Display the resulting image on a seperate window.
- "cv2.waitKey(0)" → Any displayed window will be shown indefinitely until a keyboard key is pressed.

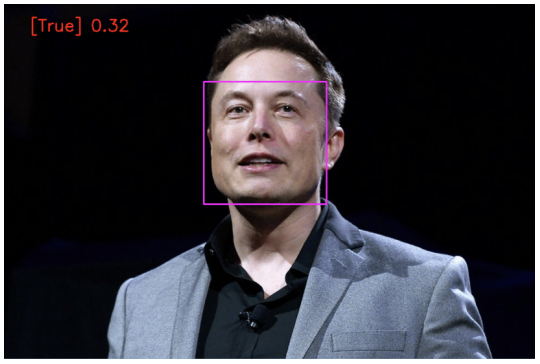

Fig. 8. "imgTest" after undergoing the facial recognition process

B. Actual Face Recognition Project

Once we got a glimpse as to what the face_recognition library is capable of, we proceeded with developing a simple program that uses the webcam to detect and record faces in real-time. To reduce unnecessary text, all functions that have been previously explained in the "Practice" section will not be explained for a second time in this section.

```
Import cv2
Import numpy as np
Import face_recognition
Import os
From datetime import datetime
```

Fig.9 . import Libraries

- "os" → management of files and file directories.
- "datetime" → manipulate date and time object data.

```
path = 'ImageAttendance'
images = []
classNames = []
myList = os.listdir(path)
```

Fig.10. Initializing some variables and storing images

- "os.listdir(...)" → get a list of all the files and directories in a given directory.

```
for cls in myList:
    curImg =
cv2.imread(f'{path}/{cls}')
    images.append(curImg)

classNames.append(os.path.splitext(cls[
0]))
```

Fig.11. Inserting all images and classes into myList

- "cv2.imread" → read an image from a file and store it in a multi-dimensional Numpy array.
- "os.path.splitext(cls[0])" → split a file path into a tuple containing the root and extension of the file.

```
def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img,
cv2.COLOR_BGR2RGB)
        encode =
face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList
```

Fig.12. A function to calculate the encodings from all images

```
def markAttendance(name):
    with open('Attendance.csv', 'r+')
as f:
        myDataList = f.readLines()
        nameList = []
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
        if name not in nameList:
            now = datetime.now()
            dtString =
now.strftime('%H:%M:%S')
            f.writelines(f'\n{name},
{dtString}')
```

Fig.13. A function to record the image face's name and time into a csv/excel database

- "entry = line.split(','), "nameList.append(entry[0])" → To only take the person's name from the file name.
- "now = datetime.now()", "dtString = now.strftime('%H:%M:%S')", "f.writelines(f'\n{name}, {dtString}')" → write the face's name and recordred time into the csv/excel file.

```
encodeListKnown =
findEncodings(images)
```

Fig.14. By calling a function, store all image encodings into a variable

```
cap = cv2.VideoCapture(0)
```

Fig.15. Capture video (a continuous stream of image frames) from the camera

- "cv2.VideoCapture(0)" → capture video from the index 0 camera, which is usually the default camera like a laptop's built-in webcam.

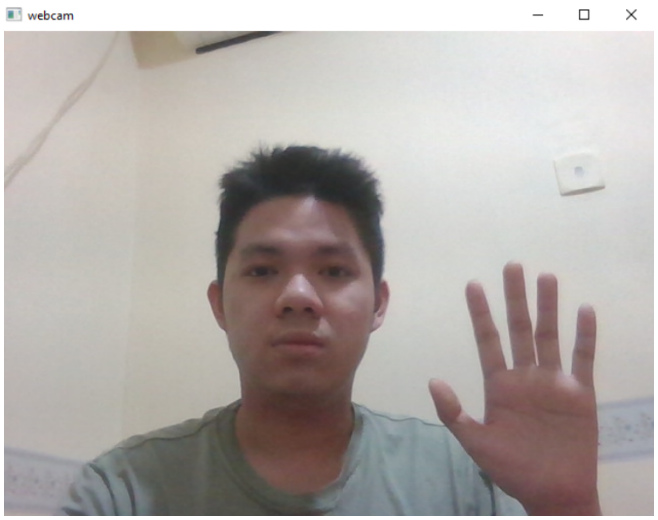

Fig.16. An example result of video camera capture

```
while True:
    success, img = cap.read()
    imgS = cv2.resize(img,(0,0), None,
0.25, 0.25)
    imgS = cv2.cvtColor(img,
cv2.COLOR_BGR2RGB)
    faceCurFrame =
face_recognition.face_locations(imgS)
    encodesCurFrame =
face_recognition.face_encodings(imgS,
faceCurFrame)
    for encodeFace, faceLoc in
zip(encodesCurFrame, faceCurFrame):
        matches =
face_recognition.compare_faces(encodeLi
stKnown, encodeFace)
        faceDis =
face_recognition.face_distance(encodeLi
stKnown, encodeFace)
        matchIndex = np.argmin(faceDis)
        if matches[matchIndex]:
            name =
classNames[matchIndex].upper()
            # print(name)
            y1,x2,y2,x1 = faceLoc
            y1,x2,y2,x1 =
y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img, (x1,y1),
(x2,y2), (0,255,0), 2)
            cv2.rectangle(img,
(x1,y2-35), (x2,y2), (0,255,0),
cv2.FILLED)
              cv2.putText(img, name,
(x1+6, y2-6), cv2.FONT_HERSHEY_COMPLEX,
1, (255,255,255), 2)
            markAttendance(name)
    cv2.imshow('webcam', img)
    cv2.waitKey(1)
```

Fig.17. While the video capture is still active, repeat the following in real-time: Detect face's location and identity, display the face's name and location (rectangle), record the face's name and recorded date into the csv/excel database

- "cv2.resize(...,(0,0), None, 0.25, 0.25)" → Speeding up the image processing by reducing the file size down to one-quarter of its original size.
- "np.argmin(...)" → Returns the index of the smallest value in an array
- ".upper()" → turns all letter characters into uppercase.
- "cv2.waitKey(1)" → Any displayed window is shown only for 1 millisecond. With an endless loop, it will show the output image in the fastest possible frames per second.

III. RESULTS AND ANALYSIS

The result generated from all of the Python codes above is a functional attendance system, where people must show their faces in front to the camera in order to get them detected by the program. After the system has successfully detected the face of the user, it will record its name as well as when the face was detected into the .csv file. The system will not record the presence of a person's face if they are absent or if they have been previously recorded (hence preventing the case where the same face is recorded every second).

Fig. 18. A training image used to help the program identify "nelsen"

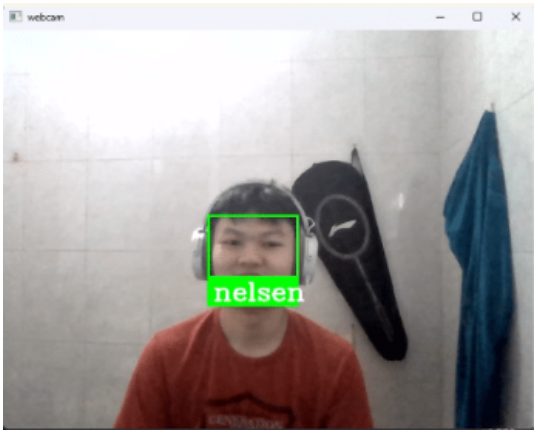

Fig. 19. “nelsen” identified and tracked in the webcam

	A	B
1	Name,Time	
2		
3	nelsen, 14:08:21	
4		
5		

Fig. 20. Recording the name and detected time for “nelsen”

One notable disadvantage of the system we created is that if the face detected by the system is not recognized (in other words, unregistered), then the system may look for an existing face that most resembles this new face, resulting in a possible misidentification. Additionally, whenever we would like to register new faces for the system to recognize, this process must be done manually by giving it static images of the person into the training folders. It is definitely more time-efficient if the face registration process can be done using the same live, automatic webcam face recognition.

IV. CONCLUSION

As a brief conclusion, we can begin to comprehend how powerful the face_recognition library truly is. There are definitely a ton of behind-the-scenes code that allow the python library to work as intended. By using minimal lines of python code, we were able to create a working attendance system based on real-time facial recognition with acceptable levels of accuracy. Understandably, we acknowledge that not many business will implement attendance systems based on facial-recognition as they tend to be more costly to install and maintain compared to other alternatives like fingerprint-scanning or id card detection.

Perhaps in the future, we can use the knowledge gained from this project to develop more challenging facial recognition projects in the future which may be beneficial for the society. Nothing is permanent except change. There will undoubtedly be some adventurous coders who can come up with even more powerful python libraries related to image processing and facial identification. We can already see such impacts today with several mind-blowing ai tools. For instance, some applications can change photographs into anime characters. The vast potential that computer vision technologies can bring us in the future can be exhilarating.

REFERENCES

[1] Li, L., Mu, X., Li, S., & Peng, H. (2020). A review of Face Recognition Technology. *IEEE Access*, *8*, 139110–139120. https://doi.org/10.1109/access.2020.3011028

[2] Bhattarai, B. et al. (2023) “Histogram of oriented gradients meet Deep learning: A novel multi-task deep network for 2D surgical image semantic segmentation,” Medical Image Analysis, p. 102747. Available at: https://doi.org/10.1016/j.media.2023.102747.

[3] Chandrakala, M. and Durga Devi, P. (2021) “Two-stage classifier for face recognition using hog features,” Materials Today: Proceedings, 47, pp. 5771–5775. Available at: https://doi.org/10.1016/j.matpr.2021.04.114.

[4] Hualong, Y., Leihong, Z. and Dawei, Z. (2021) “Non-imaging target recognition algorithm based on projection matrix and image euclidean distance by computational ghost imaging,” Optics & Laser Technology, 137, p. 106779. Available at: https://doi.org/10.1016/j.optlastec.2020.106779.

[5] Saleem, S. et al. (2021) “Face recognition using facial features,” Materials Today: Proceedings [Preprint]. Available at: https://doi.org/10.1016/j.matpr.2021.07.402.