

Face recognition case study

Futun fahad alqudayri

Department of Computer Science

Majmaah *Computer Sciences and Information Technology* College

Majmaah ,Saudi arabia

Abstract: making cameras accessible to everyone opened the door to inventing new ways to increase our safety, entertainment, and comfort. In this paper, we will discuss using face recognition to solve various problems. We will talk about how face recognition works, and its algorithms, and we will make a real program that is able to recognize faces.

i. Introduction:

In this day and age, there are cameras everywhere, the streets, homes, and even people have cameras on them at all times.

In the streets and homes, one of the most popular applications of cameras is for surveillance systems. From public spaces like streets, airports, and shopping malls to private Areas such as homes and businesses, surveillance cameras provide non-stop monitoring and prevention against potential threats. These cameras capture real-time footage, allowing security professionals to detect suspicious activities, respond to emergencies quickly, and investigate incidents effectively.

The frequency of smartphones equipped with high-quality cameras has revolutionized how we document our lives, interact with others, and perceive the world around us. The prevalence of social media platforms further amplifies the significance of photos in our daily lives. Platforms like Instagram, Facebook, and Snapchat thrive on visual content, where photos serve as a means of communication, self-expression, and connection.

With the rise of artificial intelligence and its use in our daily lives, a huge amount of effort was made by AI experts to invent ways to improve the use of cameras for both security and personal applications. This Collaborative effort has led to significant innovations, including the development of facial recognition technology.

In this essay, I'll discuss the problems and potential improvements related to cameras and photos, along with how face recognition can be a solution.

ii. What we want to solve:

Problem 1: Even though security cameras can catch the faces of criminals in the security footage, it's almost impossible to catch criminals solely using their face pictures due to the huge amount of people in big cities. So if you don't already have the criminal in a suspects list that you have to go over manually, you will not be able to catch them.

Solution: a face recognition program that is trained on criminals' mugshots, thus if the criminal already has a criminal record the program will be able to recognize them.

Problem 2: Social media filters (such as TikTok) struggle to identify faces on the screen, we need a way for social media filters to find the faces on the screen to apply the face filters on it. It must recognize faces it has never seen before.

Solution: use a trained face recognition model capable of accurately detecting and recognizing new faces.

Problem 3: apple wants a way to recognize and authorize users to open the phone in a way faster and easier than a PIN code.

Solution: a face recognition application that can recognize the user using their previous photos.

All these problems have a common solution, a face recognition application is needed to solve them.

The Solution:

The solution to these problems can be split into two parts.

- 1- Theory: we will talk about what face recognition is, how face recognition works under the hood and the algorithms used with it.
- 2- Code: we will make a real application that can detect faces and recognize them.

iii. What is facial recognition?

Facial recognition is a technology that uses biometric measurements of an individual's facial features to identify or verify their identity using machine learning. It involves capturing and analyzing facial characteristics such as the distance between eyes, nose shape, jawline, and other unique facial attributes.

While machine learning and recognition systems have made considerable progress, their effectiveness is constrained by real-world conditions. For instance, accurately identifying facial images captured in uncontrolled settings, which may include variations in lighting, posture, facial expressions, disguises, or camera movements, remains a challenge.

Main Steps in Face Recognition Systems:

- 1- Input: The system takes an image or video as input.
- 2- Face Detection: It identifies and locates faces within the input image or video.
- 3- Feature Extraction: Unique features, such as facial characteristics, are extracted from the detected faces. It captures a feature vector known as the signature from the identified face. This signature should Accurately depict the

unique characteristics of the face and enable differentiation between individuals.

- 4- Classification: The extracted features are classified or labeled based on predefined criteria.
- 5- Verification: The system verifies the identity of the individual based on the classified features, comparing them to stored data or templates.[1]

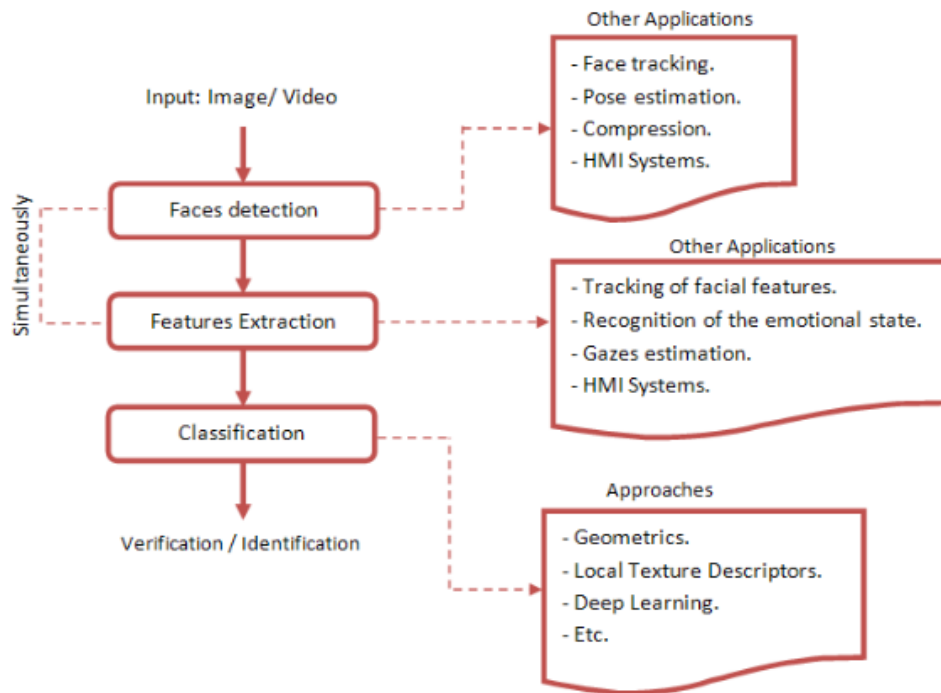


Figure 1 The standard design of an automated face-recognition system.

iv. Face recognition algorithms:

Many models can be used. in this paper, we will focus on the HOG and CNN algorithms, because they are the algorithms that will be used in the practical/ code section of the paper.

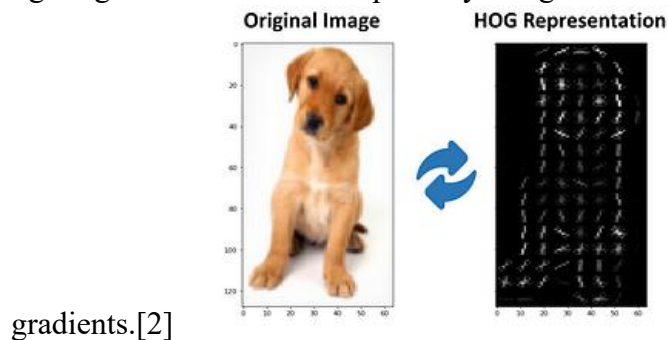
HOG:

HOG stands for Histogram of Oriented Gradients. It is a feature descriptor used in image processing and computer vision for object detection and recognition tasks. The HOG feature descriptor calculates gradients (changes in pixel intensities) in localized regions of an image and then represents these gradients as histograms of orientation. This technique captures the shape and texture information of objects in an image, making it particularly effective for detecting objects with varying shapes and structures.

HOG steps:

- 1- Image preprocessing: standardize image size
- 2- Calculate Gradient & Orientation
- 3- Build a histogram: plot these gradient and orientation values on a histogram.

- 4- Normalization: Localized gradients in an image can be influenced by overall lighting conditions. We can partially mitigate this by normalizing the



CNN:

CNN stands for Convolutional Neural Network. It's a type of artificial neural network commonly used in deep learning and computer vision tasks. CNNs are particularly effective in tasks like image recognition, object detection, and classification. CNNs have significantly advanced the field of image recognition due to their ability to automatically learn hierarchical representations of features directly from raw image data.

Here's a simplified explanation of how CNNs work:

1. input layer: receives raw image data as its input. Images are usually represented as matrices containing pixel values. The dimensions of the input layer match the size of the input images.
2. Convolutional Layers: These layers apply convolution operations to the input data. It utilizes filters that are applied to input images to identify features and patterns. This process helps extract features like edges, and textures from the input images.
3. Pooling Layers: Pooling layers downsample the feature maps obtained from convolutional layers. This helps reduce the spatial dimensions of the data while retaining important information. Common pooling operations include max pooling and average pooling.
4. Fully Connected Layers: After the convolutional and pooling layers, fully connected layers are added to the network. These layers connect every neuron in one layer to every neuron in the next layer, enabling the network to learn high-level representations and make predictions based on the extracted features.
5. Output Layer: contains neurons corresponding to the number of unique classes in the classification task. This layer generates classification probabilities or predictions for each class, indicating the likelihood of the input image being associated with a specific class.[3]

v. Code:

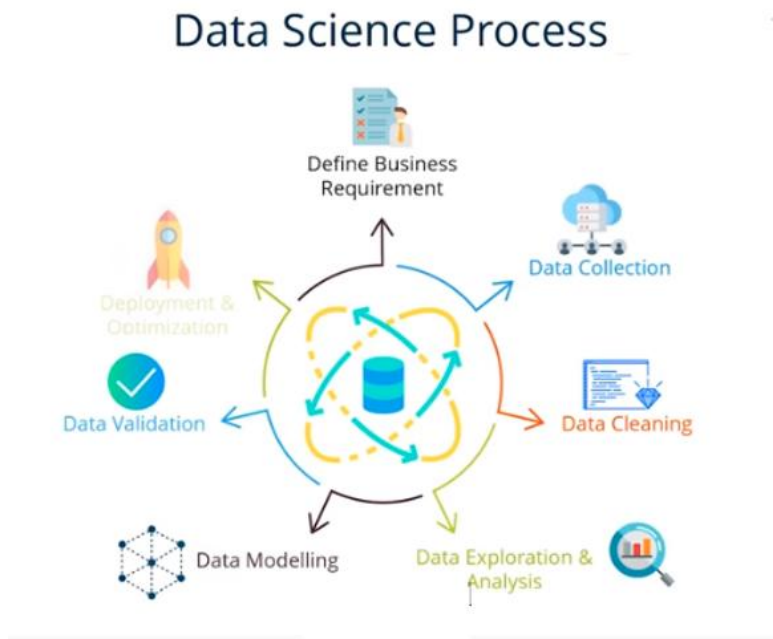
we will use a couple of libraries to help us:

1-pickle: a library is used for serialization and deserialization of Python objects. Serialization refers to converting Python objects into a format that can be easily stored or transmitted, such as a byte stream. Deserialization is the process of reconstructing Python objects from this serialized format.

2-PIL: PIL (Python Imaging Library) module, now known as Pillow, is a popular library in Python used for working with images. It provides functionalities for opening, manipulating, and saving various image file formats.

3-face_recognition: a library that is a Python module used for face detection, recognition, and manipulation tasks. It provides convenient functions for working with facial recognition algorithms and features.

the Data Science process:



1- Define the business requirement:

- An application that can detect faces
- An application that can recognize the faces of both famous and normal people
- An application that can work with new data easily

2- Data collection:

you'll need to find a dataset for training and validating your data. We will use two sources

- An already made and cleaned data set that contains pictures of several celebrities with their name.

<https://www.kaggle.com/datasets/adg1822/7-celebrity-images/code>

3- Data cleaning

When it comes to pictures. Data cleaning will mean hand-picking pictures where the face is fully within the frame, visible, and has a good lighting condition. also choosing pictures where the facial expression doesn't majorly affect the way the face looks.



4- Data exploration and analysis:

First, we need to divide our folders according to this folder structure:

```
face_recognizer/  
├── output/  
├── training/  
│   ├── famous person name/  
│   │   ├── img1.jpg  
│   │   └── img2.png  
│   └── validation/  
│       ├── famous person name .jpg  
│       └── famous person2 name .jpg  
├── detector.py  
├── requirements.txt  
└── unknown.jpg
```

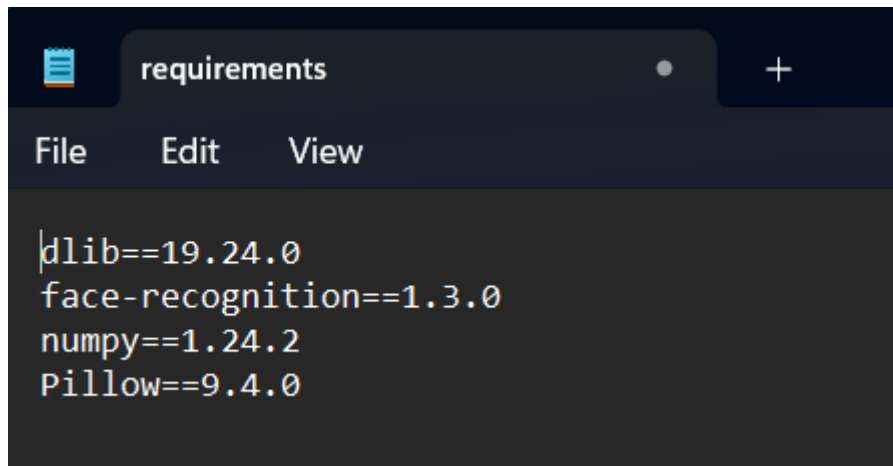
The training folder contains folders named with the pictured person's name, each folder contains the images we will use to train our model.

The validation folder contains the images we will use for testing our model.

Detector.py contains our program.

Unlown.jpg contains a picture that we will use to test face detection.

requirements.txt contains the modules we will use in our program and their version.

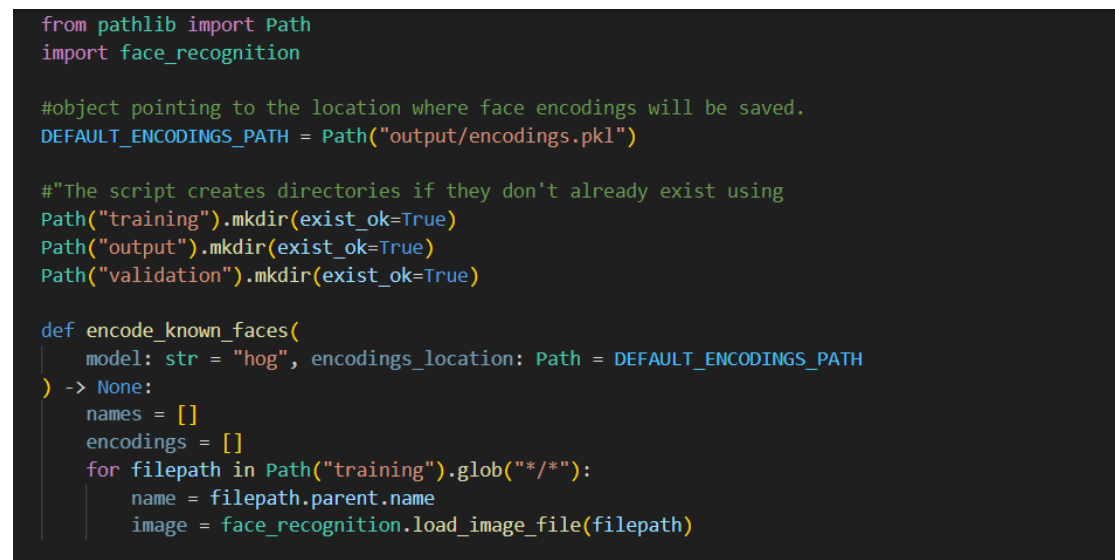


```
requirements

File Edit View

dlib==19.24.0
face-recognition==1.3.0
numpy==1.24.2
Pillow==9.4.0
```

Initially, you should import images from the training/ directory and proceed to train your model using these images. To begin, open your preferred text editor, and create a file named detector.py that contains this code.



```
from pathlib import Path
import face_recognition

#object pointing to the location where face encodings will be saved.
DEFAULT_ENCODINGS_PATH = Path("output/encodings.pkl")

#"The script creates directories if they don't already exist using
Path("training").mkdir(exist_ok=True)
Path("output").mkdir(exist_ok=True)
Path("validation").mkdir(exist_ok=True)

def encode_known_faces(
    model: str = "hog", encodings_location: Path = DEFAULT_ENCODINGS_PATH
) -> None:
    names = []
    encodings = []
    for filepath in Path("training").glob("*/*"):
        name = filepath.parent.name
        image = face_recognition.load_image_file(filepath)
```

You will create the function `encode_known_faces()`. Within this function, a loop is employed to iterate over each directory within the "training/" directory. It extracts the label from each directory and assigns it to `name`, after which it utilizes the `load_image_file()` function from the `face_recognition` library to load each image and assign it to `image`.

The `encode_known_faces()` will need two parameters,

- 1- `model` which will be the model used to locate faces. We have two choices:
 - HOG: the default
 - CNN
- 2- `encodings_location` which contains the path for storing the encodings.

we will add more code inside the `encode_known_faces` function.

```

import pickle
def encode_known_faces(
    model: str = "hog", encodings_location: Path = DEFAULT_ENCODINGS_PATH
) -> None:
    names = []
    encodings = []

    for filepath in Path("training").glob("*/*"):
        name = filepath.parent.name
        image = face_recognition.load_image_file(filepath)

        #new code starts here
        face_locations = face_recognition.face_locations(image, model=model)
        face_encodings = face_recognition.face_encodings(image, face_locations)

        #goes add all faces and encodings to different lists
        for encoding in face_encodings:
            names.append(name)
            encodings.append(encoding)

    #save the encoding to disk
    name_encodings = {"names": names, "encodings": encodings}
    with encodings_location.open(mode="wb") as f:
        pickle.dump(name_encodings, f)

encode_known_faces()

```

`face_location`, as the name suggests, contains the location of a face in an image and frames it using a square.

The `face_encoding` generates encodings for the identified faces in an image. an encoding is a numerical representation of facial features, crucial for matching similar faces based on their features.

Then we will use `Pickle` to save our encoding to the computer.

Lastly, we will call the function to start the process of loading and finding the faces in the training folder. Then save a file called `encodings.pkl` in the output folder.

5- Data modeling

Unlabeled faces:

The previous code is trained to locate and recognize the faces that appear on the training folder. What about unlabeled images?

We will be using a function that is called `recognize_faces`

First, the function will prepare our unlabeled image.


```

def recognize_faces(
    image_location: str,
    model: str = "hog",
    encodings_location: Path = DEFAULT_ENCODINGS_PATH,
) -> None:

    #get the encodings from the previous code
    with encodings_location.open(mode="rb") as f:
        loaded_encodings = pickle.load(f)

    # load the unlabeled image
    input_image = face_recognition.load_image_file(image_location)

    #locate the face and encode its features
    input_face_locations = face_recognition.face_locations(
        input_image, model=model
    )
    input_face_encodings = face_recognition.face_encodings(
        input_image, input_face_locations
    )

```

locating the face and encoding it will help me compare it to the faces in the training dataset.

```

def recognize_faces(
    image_location: str,
    model: str = "hog",
    encodings_location: Path = DEFAULT_ENCODINGS_PATH,
) -> None:
    with encodings_location.open(mode="rb") as f:
        loaded_encodings = pickle.load(f)
    input_image = face_recognition.load_image_file(image_location)
    input_face_locations = face_recognition.face_locations(
        input_image, model=model)
    input_face_encodings = face_recognition.face_encodings(
        input_image, input_face_locations)

    #new code
    for bounding_box, unknown_encoding in zip(
        input_face_locations, input_face_encodings
    ):
        name = _recognize_face(unknown_encoding, loaded_encodings)
        if not name:
            name = "Unknown"
        print(name, bounding_box)

```

in this code, we will iterate over every face that is detected in the unlabeled picture, and every picture on the training dataset.

Then using the `_recognize_face` (a function that we haven't defined yet) it will compare the unlabeled picture features to the features of other pictures in the training dataset, it will return the most likely match. If no match is found it will name the picture "unknown".

Then we will print the name of the person in the picture with the bounding box that outlines their face.

```

from collections import Counter
def _recognize_face(unknown_encoding, loaded_encodings):
    boolean_matches = face_recognition.compare_faces(
        loaded_encodings["encodings"], unknown_encoding
    )
    votes = Counter(
        name
        for match, name in zip(boolean_matches, loaded_encodings["names"])
        if match
    )
    if votes:
        return votes.most_common(1)[0][0]

recognize_faces("unknown.jpg")

```

`_recognize_face()` will compare the unlabeled picture features to the features of other pictures in the training dataset. It will return the most likely match or none.

Previously we encoded every face from the training dataset. Now we will iterate over every encoding and using the `compare_faces` function we will see if the images are similar or not. The function will return true or false for every comparison.

Using the counter function we will count how much each name got “true”. We will store the result in the votes list. The name with the highest votes will be the label of the unknown image.

Now if I run the code on this image.



This is the output:

```

jerry_seinfeld (480, 972, 1035, 418)
Unknown (726, 2513, 1281, 1959)

```

However, this format of output is not easily understandable. We will modify our code to be able to present this output on the picture directly.

```

from PIL import Image, ImageDraw
def recognize_faces(
    image_location: str,
    model: str = "hog",
    encodings_location: Path = DEFAULT_ENCODINGS_PATH,
) -> None:
    """
    Given an unknown image, get the locations and encodings of any faces and
    compares them against the known encodings to find potential matches.
    """
    with encodings_location.open(mode="rb") as f:
        loaded_encodings = pickle.load(f)

    input_image = face_recognition.load_image_file(image_location)

    input_face_locations = face_recognition.face_locations(
        input_image, model=model
    )
    input_face_encodings = face_recognition.face_encodings(
        input_image, input_face_locations
    )
    #new code
    pillow_image = Image.fromarray(input_image)
    draw = ImageDraw.Draw(pillow_image)

    for bounding_box, unknown_encoding in zip(
        input_face_locations, input_face_encodings
    ):
        name = _recognize_face(unknown_encoding, loaded_encodings)
        if not name:
            name = "Unknown"
        _display_face(draw, bounding_box, name)

    del draw
    pillow_image.show()

```

We will use the PIL library.

`Pillow_image` makes an image object of the input image.

`draw` is an `ImageDraw` object that will draw the bounding box.

Instead of printing the output, we will call the `_display_face()` function (we will define it in the next step)

To save space in our computer, we need to delete the draw variable after using it.

Then using the `show()` method we will show the resulting picture.

Now it is time to make the `_display_face` function.

```

BOUNDING_BOX_COLOR = "blue" # Define the color for the bounding box
TEXT_COLOR = "white" # Define the color for the text inside the bounding box

def _display_face(draw, bounding_box, name):
    # Unpack the bounding box coordinates into variables
    top, right, bottom, left = bounding_box

    # Draw a rectangle for the bounding box using BOUNDING_BOX_COLOR
    draw.rectangle(((left, top), (right, bottom)), outline=BOUNDING_BOX_COLOR)

    # Calculate the bounding box for the text
    text_left, text_top, text_right, text_bottom = draw.textbbox((left, bottom), name)

    # Draw a filled rectangle for the text background using BOUNDING_BOX_COLOR
    draw.rectangle(
        ((text_left, text_top), (text_right, text_bottom)),
        fill="blue",
        outline="blue",
    )

    # Draw the text inside the filled rectangle using TEXT_COLOR
    draw.text(
        (text_left, text_top),
        name,
        fill="white",
    )

```

As the comments explain we will make a bounding box that outlines the face and the name of the person on the picture, picking the color of both face outline and name outline and text at the same time.

This is the result:



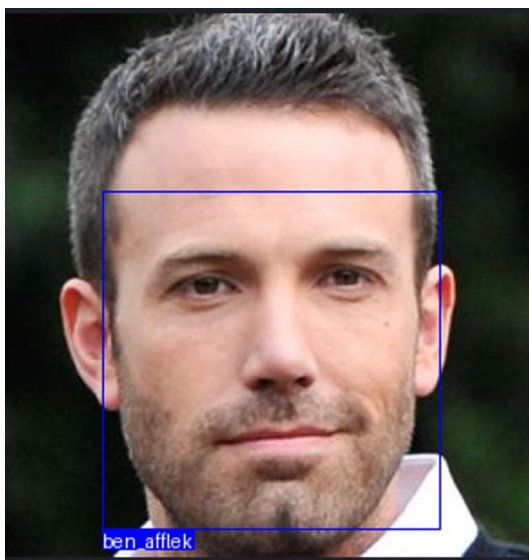
6- Data validation

```
def validate(model: str = "hog"):
    # Iterate over all files in the "validation" directory and its subdirectories.
    for filepath in Path("validation").rglob("*"):
        # Check if the current filepath corresponds to a regular file.
        if filepath.is_file():
            # Call the recognize_faces function with the absolute path of the file and the specified model.
            recognize_faces(
                image_location=str(filepath.absolute()), # Absolute path of the file as image_location argument.
                model=model # Specified model argument.
            )

# Call the validate function, initiating the validation process.
validate()
```

The validation function will open the validation file. Then it will use the `recognize_faces` function on each picture. Which will open all the pictures in the validation folder with the predicted name on it.

I will show one picture from the validation folder depicting a celebrity with their predicted name.



7-deployment:

The simplest way to make your code public is using github.

GitHub is a platform that provides version control and collaboration tools for software development projects. It also has the ability to host your code in repositories. You can choose to make your repository public or private.

The steps you need to deploy your code:

- 1- Make a GitHub account
- 2- Make a new repository
- 3- Choose the public option in the repository configuration

- 4- Add your coding files by dragging them from the file manager to the repository
- 5- In your readme file you have to give credit to any code you used from outside sources. In our code, we will credit Kyle Stratis.

Now you can share your code with anyone using a GitHub link to your repository, people can also find your repository from the search bar in GitHub.[4]

vi. Conclusion

In conclusion, in this paper we addressed three key problems that we face in our modern world. Not being able to identify criminal faces in security surveillance, needing face detectors on social media filters, and Apple needing to be able to recognize the face of the phone owner were all solved using a face recognition program that operates using the HOG or CNN algorithms.

We covered the theatrical part of the program, explaining how the face recognition process works, and how the HOG and CNN algorithms work. We also covered the coding part of the application while incorporating the data science process into it.

References:

- [1] Adjabi, I., Ouahabi, A., Benzaoui, A., & Taleb-Ahmed, A. (2020). Past, present, and future of face recognition: A review. *Electronics*, 9(8), 1188.
- [2] AI, S. (2022). Histogram of oriented gradients (hog)- simplest intuition. Retrieved from <https://medium.com/@skillcate/histogram-of-oriented-gradients-hog-simplest-intuition-2392995f8010>
- [3] Sharma, D. (2024). Image classification using CNN: Step-wise tutorial. Retrieved from <https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/>
- [4] Stratis, K. (2023). Build your own face recognition tool with python. Retrieved from <https://realpython.com/face-recognition-with-python/#step-6-add-command-line-argument>