

[illegible]

Jeremy Midvidy, jam658

Northwestern University  
McCormick School of Engineering and Applied Science

## **Table of Contents**

---

|   |    |
|---|----|
| Course Overview .....                         | 2  |
| Project Introduction .....                    | 3  |
| Implementation .....                          | 5  |
| How the System Works .....                    |    |
| 5   |    |
| Building the Database of Verified Faces ..... |    |
| 5   |    |
| Testing Inputs .....                          | 8  |
| Implementation Specifics .....                | 10 |
| Feature Extraction .....                      | 10 |
| Training a Recognizer .....                   | 13 |
| Classification .....                          | 14 |
| Conclusion .....                              | 17 |
| Future Improvements .....                     | 17 |
| Final Thoughts .....                          | 18 |

## **Course Overview: What did I think of EECS 332?**

---

Before taking this course, I had almost no experience with Computer Vision or Digital Image Processing. At best, I knew that images could be represented as a matrix of pixels and that linear algebra was used in many applications of computer graphics. In short, I really liked the material in this course and the way assignments were structured. I think this course does a very good job at (1) introducing the topic, (2) *explaining* the usefulness of its application, and (3) discussing the optimal implementation at an algorithmic level. For those reasons, I really enjoyed learning about this kind of engineering and how computers can be used to manipulate and gain insights from images. I can say confidently that I understand my implementation to the machine problems, specifically: CCL, Histogram Quantization, Canny Edge Detection, Hough Transformations, and Skin Detection.

More importantly, I feel that this course has given me a good foundation of computer vision concepts, which will enable me to participate in more advanced courses in my future studies. I would recommend to anyone, before taking this course, that they learn Python. I found many elements of Python to be useful for implementing the machine problems. I think this was extremely important to my success in this course because I never experienced any syntactical headaches that many others may have experienced when using Matlab. Python has many useful features like a clear, readable syntax, easy iteration, and function definition (which is crucial for these assignments), as well as libraries like `numpy`, `matplotlib`, or `cv2`, which made these problem sets more seamless.

That being said, I would highly recommend this course to fellow students interested in learning about Computer Vision.

### **Project Introduction: Why build a Facial Recognition system?**

---

In today's society, Computer Vision has become one of the most essential elements of any sophisticated technology system. The military, for example, employs tracking and pattern recognition techniques in its attack aircrafts. Law enforcement agencies, moreover, have used computer vision algorithms to find a face in a crowded image (notably in the 2013 Boston Marathon Bombing). Autonomous driving - now a cliched silicon valley buzzword - is powered by impressive computer vision algorithms that can detect, extract, identify, and analyze neighboring vehicles and road obstructions in real time. Of all these areas, however, few are as widespread and as topical in computer vision as is the field of Facial Recognition. These facial recognition systems are found in smartphones, laptops, doorbells, security systems etc... and present an extremely important tool for consumers in how they interact daily with their products. Every minute, for instance, millions smartphone users access their phone with a facial recognition system. From all people's day-to-day interactions with such systems, it would be clear to anyone that facial recognition is a large part of the technology that is produced today.

Since it is so widespread and so important, I chose to implement a *simple* Facial Recognition system for my final project for EECS 332: Computer Vision, Fall 2018, at Northwestern University. This report will outline my implementation and approach to my system, as well as discuss the motivation behind my design decisions as well as some improvements that I can implement in the future (with more technical knowhow) that would make my system more robust.

### **Functional Overview: What will my system be able to do?**

---

The goal of my simple Facial Recognition system will be to emulate security system that grants access to “verified” individuals. For example, I can show my facial recognition system a picture of my face and say “this person should be allowed access.” Then, if I were to try and access the system, I would tell it “I am Jeremy and I would like to access the system.” Then, it will look at my face (via a camera, in this case, my computer’s built-in webcam) and be able to say either, (1) “YES, you ARE Jeremy. You can have access,” or (2) “NO, you are NOT Jeremy. You cannot have access.” Clearly, if I show the system my face, it should give me access. If I tell it I am Jeremy, and then I show the system my roommate, it should not grant access.

This is the simple functionality of the system. Clearly, it can simulate a facial recognition system that either only has one verified face, like an iPhone’s FaceID, or a system that has many verified faces, like all of the President’s national security team that would scan their face before entering the White House situation room. The point is to allow access - e.g. to *recognize* - only faces from the individuals that have been enrolled.

Clearly, this presents two distinct parts of my simple facial recognition system. The first will build the database of verified faces, and the second will be able to classify user input as to whether the inputted claim is really the verified person in the database. These two parts, when combined, emulate the a facial recognition system found in many useful technologies.

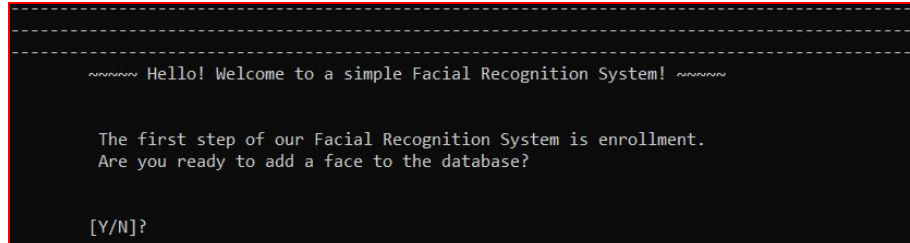
## **Implementation: How does my system work?**

---

As mentioned, my system has two distinct components. The first will build the database of verified faces and the second will be able to classify user input. I will now discuss each part in greater technical detail.

### **Part One: Building the Database of Verified Faces**

The simple system can be invoked from the command line with the command, `$ py FacesRun.py`, when in the working directory. When the program starts, it prompts users by saying “*The first step of our Facial Recognition System is enrollment.*”, like so:

A terminal window with a black background and white text. The text reads: "~~~~~ Hello! Welcome to a simple Facial Recognition System! ~~~~~", followed by "The first step of our Facial Recognition System is enrollment.", then "Are you ready to add a face to the database?", and finally "[Y/N]?".

```
~~~~~ Hello! Welcome to a simple Facial Recognition System! ~~~~~

The first step of our Facial Recognition System is enrollment.
Are you ready to add a face to the database?

[Y/N]?
```

Figure 1: Invoking the Program

Then, user's can begin the enrollment procedure. Enrollment is simple and user-friendly. When the user is ready, he or she can enter “Y”, and the program will enroll a face. Face enrollment has two steps: (1) enter the name of the face to be added to the database, then (2) the machine will then open the camera and capture images of the enrollee's face.

For example, if I hit “Y,” I will be given this prompt:

```

[Y/N]? Y

Great! Lets begin.

-----
Adding a face! Please enter the name of the person to be added.
Name:

```

Figure 2: Face Enrollment

If I enter my name, “Jeremy,” as the “Name” of the face to be added, the program will then instruct me to look at the camera as it captures pictures of my face that it will associated with the inputted name.

```

-----
Adding a face! Please enter the name of the person to be added.
Name: Jeremy
Launching Camera! Please look directly at the camera!

```

Figure 3: Camera Launch

Then, the camera will launch. Users will look at the camera until enough images are captured, at which point it, the camera will close. While users are looking at the camera, the program will print a blue rectangle around the face as the program tracks it through the frame.

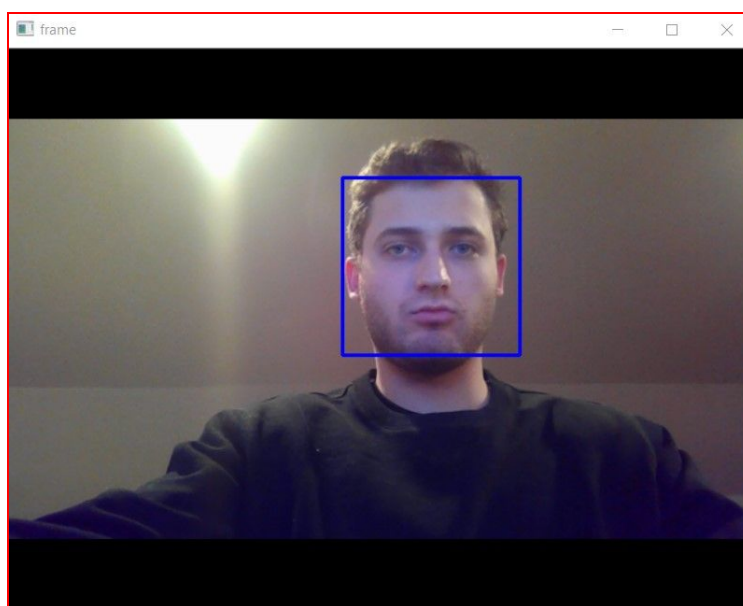


Figure 4: Face Tracking

The camera-frame will close when it has captured enough training images of face to be enrolled. Every time a face is added to the database, the program makes a new directory in the “/database/” directory in the project working directory. Each directory is given the filename of the inputted user. Later, as I will discuss, this filename will be the label of the face that the classification system will try to return.

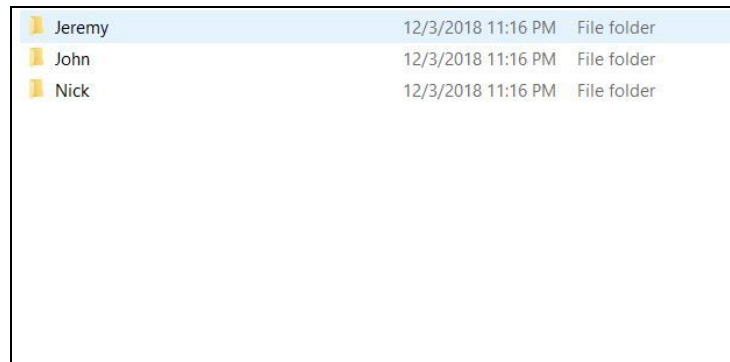
The program will then prompt the user, “*Would you like to add another?*” at which point the user can either respond “Y” or “N”. If the response is “Y”, the same process repeats as above - a name is entered and pictures of the new face to be enrolled are captured. If the response is “N,” the program continues to the next phase, classification.

Let’s say I enroll two more faces, my roommates Nick and John, and add their training images to the database.

```
Adding a face! Please enter the name of the person to be added.
Name: Jeremy
Launching Camera! Please look directly at the camera!
Jeremy added to database!
-----
Would you like to add another?
[Y/N]? Y
-----
Adding a face! Please enter the name of the person to be added.
Name: Nick
Launching Camera! Please look directly at the camera!
Nick added to database!
-----
Would you like to add another?
[Y/N]? Y
-----
Adding a face! Please enter the name of the person to be added.
Name: John
Launching Camera! Please look directly at the camera!
John added to database!
-----
Would you like to add another?
[Y/N]? N
```

Figure 5: Enrolling more Faces





|        |                    |             |
|--------|--------------------|-------------|
| Jeremy | 12/3/2018 11:16 PM | File folder |
| John   | 12/3/2018 11:16 PM | File folder |
| Nick   | 12/3/2018 11:16 PM | File folder |

Figure 6: Face Database in /database/

So that is how the face database gets constructed from user input. After the user has entered as many faces as he or she wishes to do, the program will use the database to *train* a *recognizer* which will be used in the next part to classify inputs. I will discuss training and classification in a later section in this report.

## Part 2: Testing Inputs

After enrollment, the user can then test the system to see if it correctly recognizes inputs. This part functions similarly to the first part. Users can claim to be a person in the enrolled database. Then, the program will launch a camera, capture frames of the claimant, and then perform a recognition on the trained data from enrollment to either classify the input as “YES, you ARE” the inputted user, or “NO, you are NOT” the inputted user. This can be said to simulate the functionality of granting or denying access in a facial recognition system designed for security, like FaceID or entry to a secure facility.

For example, if I tell the computer, “I am Jeremy,” it will then take pictures of my face, and using the data in the database under the label, “Jeremy,” it will see if I really am who I claim to be.

```
The Simple Facial Recognition System can now classify input!
Do you want to try and fool me?

[Y/N]? y

Ok then. Good luck!

-----
Testing a face! Who do you claim to be?
Name: Jeremy
```

Figure 7: User Testing the System

Like when building the database, the system will then capture photos of the face of the claimant. After capturing photos, it will perform the classification based on the data in the enrolled database, and then output a message signifying accept or reject. After the camera takes photos of me, the system accepts me because my face “matched” with the data in the database under the label “Jeremy”.

```
-----
Testing a face! Who do you claim to be?
Name: Jeremy
Launching Camera! Please look directly at the camera!
Yes, you ARE Jeremy !
-----
Would you like to test another?
[Y/N]?
```

Figure 8: Classification Output

Of course, users can test as many inputs as they would like on the simple facial recognition system. The process will repeat for each new claimant: enter a name and let the program capture images of the claimant's face. Then, the program will then perform classification on the inputted images, and an output message signifying whether the program agrees with the claim or not.

This demonstrates the functionality of my project, since it will allow any number of faces to be enrolled, and then use the enrolled data to classify inputs - just as an actual facial recognition system does. Of course, the integral part of this system is *how* the system trains the

recognizer and *how* the program classifies inputs. I will discuss this in greater detail in the next section.

### **Implementation: Feature Extraction, Training, and Classification**

---

After the user enrolls the database of faces, the system then trains a *recognizer* that can classify inputs. To train the recognizer, the system extracts features from the captured face. Likewise, when performing classification, the recognizer then uses the learned features for the specific label to gauge the similarity between the known label and the claimant's face. I will now discuss these two steps in greater detail.

#### **Part 1: Feature Extraction**

Each time a new face is added to the database during enrollment, the program opens the computer's camera and captures images of the face to be enrolled. When these images are captured, it can then extract useful information from the images of the enrolled person's face to use later for classification. This step is called *feature extraction*.

As demonstrated in *Figure 4*, the camera keeps track of the user's face as it captures. Actually, this is just a neat feature of feature extraction. The programs keeps the camera open

until it gets 100 *frames* of the user's face. At each frame in the capture, the program extracts the user's face from the image. To keep track of the user's face during image training, I just use `cv2`'s `draw rectangle` feature to draw the blue square on the camera screen. This simulates the tracking effect - in reality, it is extracting the face from frame-to-frame in real time, signaling the coordinates of the extraction, and then just drawing a rectangle. The frame-to-frame extraction is how the feature extraction is actually accomplished.

To understand this in greater detail, consider just a single frame captured when the program is capturing training data during enrollment. The process for extracting features from just a single frame is the same as all other frames. For example, look at the frame in Figure 9:

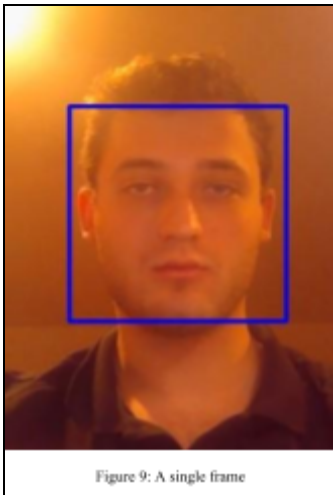


Figure 9: A single frame

When the program collects a training image to enroll a face into the database, it makes a function call to another Python file in the working directory, `FaceCapture.py`:

```
84: FaceCapture.main("database/" + newFaceName)
```

The input to the main function in `FaceCapture.py` is the newly created directory for the entered label that has been added to the database directory. Then, `FaceCapture.py` handles the image capturing and

feature extraction for the face to be enrolled. Consider the code of `FaceCapture.py`:

```

11 def main(out_path):
12     # play with these --> currently FRONTAL_FACE
13     face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')
14
15     # open image capture
16     cap = cv2.VideoCapture(0)
17     count = 0
18     while count < 100:
19         # capture frame by frame
20         ret, frame = cap.read()
21
22         # need to convert to gray
23         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
24
25         # get all faces from the given frame
26         faces = face_cascade.detectMultiScale(gray, scaleFactor=1.5, minNeighbors=5)
27
28         for (x,y,w,h) in faces:
29             #print(x,y,w,h)
30             # know that x,y,w,h is the ROI
31             roi_gray = gray[y:y+h, x:x+w]
32
33             # want to recognize frame and reigons of interest
34             # deep learned model to predict things
35             # Keras, TensorFlow, PyTorch, Scikit-learn
36             out_file = out_path + "/" + str(count) + ".png"
37             cv2.imwrite(out_file, roi_gray)
38
39             # draw a rectangle
40             color = (255, 0, 0) # BGR 0 -255
41             stroke = 2
42             end_cord_x = x + w
43             end_cord_y = y + h
44             cv2.rectangle(frame, (x,y), (end_cord_x, end_cord_y), color, stroke)
45             count += 1
46
47
48         # Display the resulting frame
49         cv2.imshow('frame', frame)
50
51         if (cv2.waitKey(10) and 0xFF == ord('q')):
52             break
53
54     # when everything is done, release the capture
55     cap.release()
56     cv2.destroyAllWindows()
57
58     return

```

Figure 10: Code for Image Capture

By inspecting the implementation, one can see that the extraction of facial features is rather intuitive. In line 16, the program initializes `cap` to be the `cv2.VideoCapture` object - this launches the camera. Then for each frame, the following logic is executed:

1. Extract the 2D numpy array representing the current image frame and store it in the variable `frame`. [line 20]
2. Convert the current `frame` from RGB to Grayscale using `cv2.cvtColor()` method, and store the greyscale 2D numpy array in the variable, `gray`. [line 23]
3. Then use a *haar cascade* to identify the regions of the face in the image represented by `gray`. I stored the detected *region of interest* in the variable, `faces`. [line 26]
  - a. A *haar cascade* is a highly precise, well trained, set of data points that can be used in computer vision to detect different object regions.
  - b. For example, there are different cascades for detecting eyes, smiles, cheeks, side-profiles and so on and so forth.
  - c. I initialize the *haar cascade* that I use on [line 13], and store it in the variable `face_cascade`.

4. The *haar cascade* will detect the face in given grayscale frame. This is called the *region of interest*, which denotes which section of the frame I am interested in analyzing.
5. Then, I write the *region of interest* 2D numpy array to an image file in the current enrollees database folder.

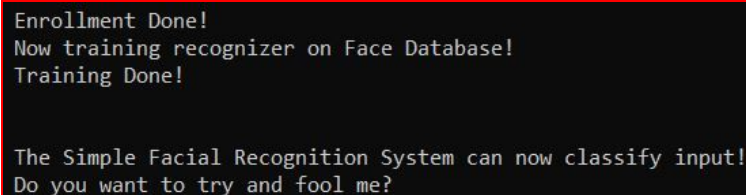
This process repeats for each frame: (1) identify the region of interest, (2) extract the region, (3) write the region to the database folder. Later, I will use these images to train the classifier. Currently, I only use the “front-facing” *haar cascade*, which requires participants to face the camera and not be wearing headwear accessories. This can limit the robustness of the system, since it needs front-facing photos to perform well. I will discuss this more at the end of the report.

After repeating this process until 100 *regions of interest* are gathered, the program closes the frame and is ready to enroll another face. If the user chooses to enroll another face, the process repeats. If not, the program then trains a recognizer which will later be used for classification.

## Part 2: Training a Recognizer

After enrollment, the program then constructs a recognizer from the database of verified faces. This is done before the program allows the user to make input claims, as illustrated in

*Figure 11.*



```
Enrollment Done!  
Now training recognizer on Face Database!  
Training Done!  
  
The Simple Facial Recognition System can now classify input!  
Do you want to try and fool me?
```

Figure 11: Program builds a recognizer

When the program is ready to construct the classifier, it makes a call to another Python file in the working directory, `FacesTrain.py`:

```
103: FacesTrain.main("database\\")
```

This file trains the recognizer. The full code is stored in the corresponding Python file found in the same directory as this report. I will, however, describe the logic that the trainer follows. When called, the trainer executes this logic:

1. Iterate through the database of enrolled images. For each folder, read each of the 100 images gathered when enrolling.
  - a. For each image, read it into a 2D numpy array.
  - b. Then append the 2D numpy array to the list, `x_train`, and the known label (folder name) to the list, `y_labels`.
2. After each folder, `x_train` and `y_labels` should each be 100 elements longer. `x_train` will have added 100 2D numpy arrays and `y_labels` will have added the same label 100 times (each image has the same label).
3. When `x_train` and `y_label` lists are formed from all 100 images in every folder in the `/database/` directory, create a `LBPHClassifier` from these two lists.

```
20: recognizer = cv2.face.LBPHFaceRecognizer_create()  
From x_train and y_labels per (1) and (2),  
56: recognizer.train(x_train, y_labels)
```
4. The LBPH classifier uses LBP (Local Binary Patterns) in the 2D numpy matrix to construct a recognizer. This recognizer can now take an input array and match it to the known LBP data.

In short, we have 100 images for each label. We use LBP to construct the local binary patterns of each training image (each image is the extracted *region of interest* from the enrolled face). The aggregate patterns for each image represent the learned recognizer for each label. Then, to classify inputs, it just does LBP on the input image and compares it to the learned data for each label. Here is a high level description of this process:

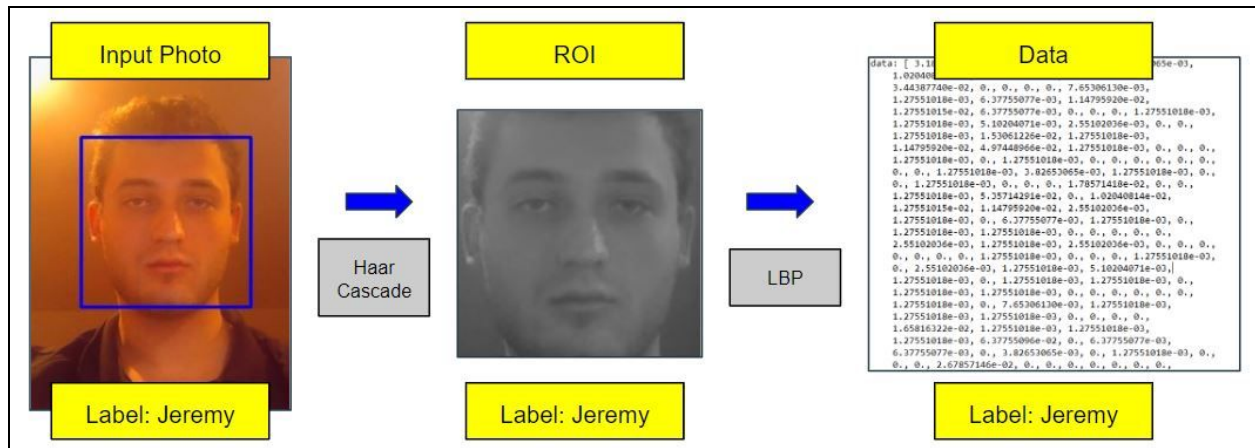


Figure 12: Input to LBP

The *region of interest* from each input frame is the training image for the recognizer.

Each label has 100 training images, so the recognizer has 100 LBP extractions per label. It aggregates the LBP data, and then knows what the “typical” LBP data is for each label. With this information, it can now easily perform classifications. All it has to do is read the input image, extract its region of interest, gather its LBP data, and then compare it to the LBP of each of the know labels, and return the label that has the most similar LBP data. I will explain this similarity metric, and classification, in more detail in the next section.

### Part 3: Classification

With the LBP data gathered, the recognizer is now ready to classify input images. As seen in *Figure 11*, the program tells the user, “*Training Done! The Simple Facial Recognition System can now classify input!*” Consider I enter “Jeremy” for the person I claim to be:

```
-----
Testing a face! Who do you claim to be?
Name: Jeremy
Launching Camera! Please look directly at the camera!
```

Figure 13: Program ready to classify

The program will then launch the camera, and like in enrollment, take 100 photos of my face. Then, the program will make a separate call to another Python file, `FacesTest.py`:

```
151: result = FacesTest.main(testing")
```



In short, the program calls the main function of `FacesTest.py`, and the main function iterates through the 100 captured images of the claimant stored in the `/testing/` directory. For each image, the follow line is executed:

```
24: predict_id, conf = recognizer.predict(curr_img)
```

The `recognizer.predict()` function takes in the current 2D image from the claimant (which, as we know, is a *region of interest* extracted by the program during capture). The recognizer then iterates through the LBP data it has trained on, and compares that to the LBP data it extracts from the inputted image. Then it returns the label which had the lowest possible difference in LBP information to the input image. The key here is that for each classification, the recognizer also returns a confidence interval. This confidence represents how *confident* the recognizer is that the current frame is actually the predicted label. As the confidence interval approaches 0, the recognizer becomes more confident in its classification. Likewise, as the confidence approaches infinity, the recognizer loses confidence in its prediction.

So, for all 100 images, the testing file keeps a list of confidence intervals. After getting the confidence interval for each of the 100 images, it finds the average. If the average is less than a learned threshold, then the classification is accepted, and the input is thus accepted. On the other hand, if the average is less than the learned threshold, the classification is rejected, and the input is thus rejected.

Of course, the natural question is now: how is the threshold learned? This seems like it would have a complicated answer, but the solution is rather simple. To learn the threshold, the recognizer finds the average confidence interval of each known label in the `/database/` of verified faces, and then averages those averages. By training against itself, the recognizer learns

how close the input images are to the LBP aggregate stored in the recognizer. Since this can take a long time (depending on the number of inputs in the database), I have just performed this experiment a few times and have hardcoded the threshold to be 50. This was about the average result of a few test cases, plus some wiggle room since the confidences are being gained from self testing.

So with the recognizer constructed, all the program has to do to classify is gather pictures of the claimant, get the confidence from the recognizer for each 100 images of the claimant, compute the average confidence, and then see if this average is less than 50. If it is, then the program accepts the claimant and output, “YES, you ARE Jeremy.” If the average confidence is greater than 50, then the program rejects the claimant and outputs, “NO, you are NOT Jeremy.” This can be thought simulating an iPhone granting access or rejecting someone attempting to unlock the phone with FaceID. In this, my simple facial recognition system can fully simulate a facial recognition system found on many other devices!

## **Conclusion: Future Improvements and Final Thoughts**

---

### Future Improvements

Since Facial Recognition systems are so important and found on so many technologies, researchers and academics are constantly making improvements to the field. My implementation, by just today's technologies, is extremely primitive. The first area for improvement in my system would be to improve the quality of images captured. My system just uses my laptop's built in webcam to take 100 RGB images of faces to be enrolled. Modern systems are much more advanced than this. Just consider an iPhoneX, which uses an infrared dot projector to capture facial features. This system involves extremely advanced hardware and software to implement. Of course, it can extract facial features in any profile, any lighting, with the user wearing any headwear, and so on and so forth. So, clearly, the more advanced my capture system, the more robust my facial recognition system can be.

Additionally, the processing and classification portion of my project can be improved. Modern systems use a variety of advanced measures, like deep learning or 3D facial image reconstruction. These methods are much more sophisticated than the LBP classifier that my system implements, so using one of the more advanced classification techniques would add a higher degree of precision to my classifier.

Clearly, there is a lot of improvement that I can make to my system by using more sophisticated and advanced approaches. But since my system can emulate a simple facial recognition system with good accuracy, I think it is a good starting point for anyone interested in building such a system.

### Final Thoughts

I learned a lot doing this project. First, I learned about general approaches to building a facial recognition system, like how many training images are needed for each enrolled face and how to build a recognizer using the captured training photos. Additionally, I learned the important elements of feature extraction with a *region of interest*, training with a *LBP recognizer*, and classification with *confidence intervals*. These three elements are the most important parts of any facial recognition system. Even though my implementation is quite barebones compared to commercial-grade technology, all facial recognition systems implement these three parts in some way or another. After completing my system, I feel that I can make it more robust by upgrading parts of my project with more advanced technology (e.g. using a deep-learning technique rather than LBP). I think the goal of the project was to get to this point, so it is good to know that I have built a tool which can be upgraded and approach the quality found in consumer technology.

This project, and this course as well, taught me a lot about the basic principles of computer vision and how to implement computer vision systems. In the machine problems and this project, I learned how tricky and how useful it can be to implement cv systems. Clearly, I am now extremely prepared to more learn high-level concepts, more advanced hardware systems, and the finer details that make high-grade computer vision systems so eminent in our society.