Utilizing Facial Recognition Software to Record Classroom Attendance

by

William E. Miller

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 5, 2018

Keywords: Facial Recognition, OpenCV, Attendance

Approved by

Dr. David Umphress, Chair, COLSA Corporation Cyber Security and Information Assurance Professor
Dr. James Cross, Professor of Computer Science and Software Engineering
Dr. Dean Hendrix, Associate Professor of Computer Science and Software Engineering

Abstract

Facial recognition software offers many opportunities for educators, especially in regards to recording classroom attendance. With so much time lost in the classroom each day, utilizing facial recognition software allows professors to reclaim that lost time, and, therefore, accomplish more in the classroom. In practice, this technology is relatively new. Within the past few decades advancements have been made to introduce facial recognition techniques to the general public. The majority of the research has been initiated by the intelligence community to track persons of interest. However, these practices have become common with various businesses and even churches. Corporations, such as Amazon and Microsoft, have taken this opportunity to provide customers with access to servers and APIs that allow individuals to incorporate facial recognition software in their own personal lives or for their businesses. There are also those who have elected to create their own facial recognition programs by using libraries, such as OpenCV. Combining the latter method with proper software process procedures has produced a program that is intended to aid professors in the classroom in regards to recording classroom attendance. Because of the litany of tools and libraries required to use facial recognition software, operating these programs within a separate preconfigured virtual environment is likely the best practice. Ubuntu has proven to be the best choice thus far. Using a process most similar to Scrum, this software was developed by utilizing multiple iterations in the form of weekly sprints. After detailing the creation of this software and validating that the solution works as expected, one should gain insight into the inner workings of facial recognition software. The result should fill the needs of school faculty and allow for future researchers to further optimize this technology to enhance its performance and efficiency.

Table of Contents

List of Figures

Chapter 1

Problem Description

1.1 Introduction

Facial recognition software has become ubiquitous in modern day life. From intelligence agencies to churches, many have found various uses for this relatively new technology. One particular faction could also greatly benefit from facial recognition software. This group would be college professors. Since class time is such a valuable commodity, spending the first few minutes of each class with the monotonous task of recording attendance can be wasteful. In fact, one can surmise that those who call roll exhaust four hours of class time each semester if five minutes are set aside during each class. Therefore, a new method should be used that avoids this tedious task. By using classroom cameras which are now installed in most classrooms, a professor only has to feed the lecture video into the software to determine who was in attendance.

The benefits of this method are not only to save time but also to verify the accuracy of attendance. The most common method currently used for college courses is a sign-in sheet. However, students can easily copy their friends' names with little accountability. Fingerprint scanners are an option, but this takes too much time as well. With facial recognition software, however, one can not only take attendance immediately, but a professor can also verify if the student was physically there. By using the student's face as a unique identifier, this technique eliminates fallibility in the process.

There are, of course, many factors to consider when dealing with this option. The chief issue is ensuring accuracy. The software must be able to return a correct response. A professor must have confidence in this system. Otherwise, the program will be essentially useless. Ensuring that the software is operating in an optimal manner is another goal of

this project. Any facial recognition technology (FRT) must be able to consistently detect, recognize, and report the faces that appear in the subject matter. With all of this in mind, one can then examine any prior work done in this field.

1.2 Thesis Breakdown

This thesis is broken down into five chapters. The first deals with the problem description. The second chapter will discuss previous work done in the field, as well as the foundation for this project. Chapter 3 will deal with the solution and the process used to develop the final result. The fourth chapter validates the solution. Finally, the final chapter will cover the conclusion and opportunities for future work.

Chapter 2

Previous Work

2.1 Potential Uses

There are many variables one must consider when developing and using FRT. For instance, one may wish to use this software to locate subjects other than faces. Depending on how the software is trained, one can theoretically detect anything they wish from a picture or video. These programs can be used to identify and extract text from specific frames. Security personnel might use this technology to identify weapons in a crowd. Border patrol could search for illegal contraband, such as drugs or even fruit. Need to find a car that was identified near a crime scene? An intelligence agency could input the appropriate criteria and scan the footage from city cameras to locate the suspect. The practical applications are nearly limitless, but the software must be trained to detect these items, so one must first decide what purpose the application will serve.

2.2 History of Facial Recognition

The genesis of FRT began in 1964 at a research facility in Palo Alto, California. Hired by an unnamed government intelligence agency to develop software with the ability to locate individuals in a crowd, Woody Bledsoe and his colleagues, Helen Chan and Charles Bisson, discovered the difficulties that still plague programmers today. Bledsoe stated,

> "This recognition problem is made difficult by the great variability in head rotation and tilt, lighting intensity and angle, facial expression, aging, etc. Some other attempts at facial recognition by machine have allowed for little or no variability in these quantities. Yet the method of correlation (or pattern matching)

> of unprocessed optical data, which is often used by some researchers, is certain to fail in cases where the variability is great. In particular, the correlation is very low between two pictures of the same person with two different head rotations" [1].

With the nearly infinite amount of unique faces and situations, discovering a means to cope with such issues became the primary goal. They soon learned that one could generate unique information of each image by breaking down the structure of a face into separate parts. Identifying the length of a nose, the width of a mouth, the height of the forehead, etc., allowed the researchers to quantify the different features of each face. Doing this allowed them to search for those measurements in an image even if the subject is not facing the camera. While this does not necessarily result in perfect accuracy, combining multiple images or frames can result in a high level of confidence.

However, Bledsoe and his colleagues did not want to be at the mercy of the subject's facial orientation. Therefore, they attempted to normalize these faces so that the comparison between the training data and the faces in their experiments would be similar. Thus, they had to account for all the directions a head could tilt and rotate. By generically mapping these movements and angles, the researchers were able to surmise a face's appearance from a forward-facing position. This greatly aided the software during practical use.

Peter Hart would continue the work of his predecessors while at Stanford. He strove to improve the accuracy of this budding new technology. Hart compiled a database that included over 2,000 unique faces. In his experiments, he was able to prove that his software could consistently recognize faces better than most humans [1].

During the 1970s, a paper entitled "Identification of Human Faces" was published by A. J. Goldstein, L. D. Harmon, and A. B. Lesk. Their research detailed the use of 22 specific features needed to identify faces in a crowd. From lip thickness to hair color, the researchers studied these markers to determine the best means of recognizing an individual. They determined that under certain conditions only six or seven features are needed, and

only 14 feature-descriptions are required for a population as great as 4,000,000 [2]. However, much of their model still required manual input and was impractical for most situations.

In the late 1980s, FRT began making significant advances. The Los Angeles County Sheriff's Department began compiling a database of mugshots in order to identify criminals with FRT [3]. The Eigenface technique was developed by L. Sirovich and M. Kirby in order to recognize the differences in faces and record those variations in a quantifiable manner. They proved that no more than a 100 values were needed to identify a face in typical conditions [4]. This research did much to progress facial recognition technology and created a strong basis for future research.

In 1991, Matthew Turk and Alex Pentland of MIT used the research done by Sirovich and Kirby to develop a computer system that could detect faces in an image almost immediately [5]. The ability to use this technology in almost real time was a great improvement upon previous attempts and also provided a means for the system to learn and recognize new faces automatically. Two years later, DARPA initiated the FacE REcognition Technology (FERET) program. By utilizing some of the sharpest minds from universities across the country, the FERET program first established the current status of FRT and then endeavored to create working software that could be used in real-life scenarios rather than controlled environments. Perhaps their greatest accomplishment, however, was the massive collection of training data they amassed. At the time, the size of their database was unparalleled with 14,126 facial images of 1199 individuals [6]. This database is still used to test new facial recognition algorithms. With this effort, FERET was able to push facial recognition software into the public market and encourage businesses to invest in this budding technology.

By the late 1990s, research in this field was being conducted around the world. From California and Maryland to Germany and the Netherlands, a growing interest was demonstrated by many due to the various applications facial recognition software possessed [7]. Airports and banks were beginning to utilize this technology, and other industries began to develop interest as well. In fact, by the turn of the century, this technology was introduced

to football as well. At Super Bowl XXXV, facial recognition software underwent an unprecedented test. Virtually everyone in attendance had their faces analyzed, collected, and compared to a criminal database. At the time, this was seen as an egregious intrusion of privacy [8]. Ultimately, the software did not perform well in the crowded stadium, but this was the first time that most people became aware of law enforcements' new abilities.

A Minnesota company named Identix developed algorithms in the early 2000s that enabled its software to use a three dimensional model of a face as training data. This allowed the computer to identify a great deal of detail in a target's facial structure. As mentioned earlier, in the past, programmers were mostly limited to dealing with only a two dimensional representation which made it extremely difficult for a computer to recognize people who were not facing the camera or in optimal lighting conditions. With this breakthrough, though, one could now collect 3D images of subjects' faces and then compare them to other 3D images that had been gathered, resulting in highly reliable results. If a 3D image wasn't available, however, Identix's algorithms could still allow for an accurate comparison with a 2D image [9]. While others had previously attempted to utilize three-dimensional figures, Identix's algorithms were by far some of the most efficient and accurate thus far.

Beginning in 2000 and continuing through 2006, the National Institute of Standards and Technology (NIST) conducted the Face Recognition Vendor Tests (FRVT). Similar to FERET, these tests were designed to measure the status of FRT throughout the public and private sectors. Weaknesses were pinpointed throughout each test so that improvements could be made in these areas. Example findings in 2002 include the following:

> "Given reasonable controlled indoor lighting, the current state of the art in face recognition is 90% verification at a 1% false accept rate.
>
> The use of morphable models, which maps a 2D image onto a 3D grid in an attempt to overcome lighting and pose variations, can significantly improve non-frontal face recognition.

> Watch list performance decreases as a function of gallery size performance using smaller watch lists is better than performance using larger watch lists.
>
> In face recognition applications, accommodations should be made for demographic information since characteristics such as age and sex can significantly affect performance" [10].

The 2006 tests concentrated on "high resolution still imagery (5 to 6 mega-pixels), 3D facial scans, multi-sample still facial imagery, and pre-processing algorithms that compensate for pose and illumination" [11]. This work helped push the boundaries of FRT. Those involved were able to establish a baseline and draw attention to the failures of the current state.

Alongside the 2006 FRVT tests, an additional effort was made by the US government to improve this technology. This initiative was dubbed the Face Recognition Grand Challenge. From 2004 to 2006, any interested parties were invited to participate in this contest. The primary goal was "to improve the performance of face recognition algorithms by an order of magnitude over the best results in Face Recognition Vendor Test (FRVT) 2002" [12]. Researchers were given "a six-experiment challenge problem along with a data corpus of 50,000 images. The data consists of 3D scans and high resolution still imagery taken under controlled and uncontrolled conditions" [12]. The experiments focused on a range of issues which included different levels of illumination and varying uses of 3D images. Once again, the goal was to determine the accuracy and efficiency of the proposed algorithms. In the end, the study concluded that the best algorithms were "10 times more accurate than the face recognition algorithms of 2002 and 100 times more accurate than those of 1995" [12]. Even identical twins could be uniquely recognized.

These government sponsored challenges have done a great deal in advancing the state of the art. In fact, NIST has determined that from 1993 to 2010 the error rate of automatic face-recognition systems has decreased by a factor of 272 [12].

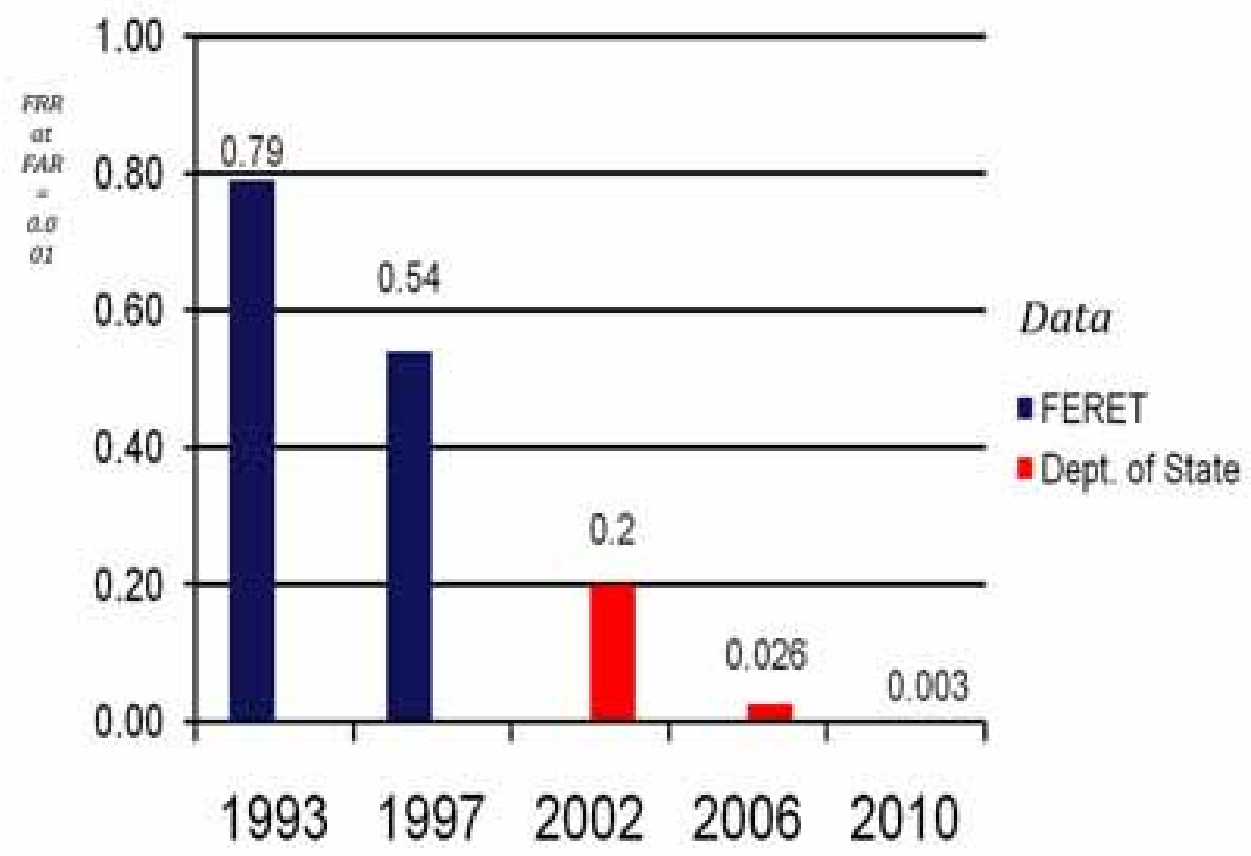


Figure 2.1: "Progress is quantified from the 1993 evaluations to MBE 2010. Improvement is reported at five key milestones. For each milestone, the false rejection rate (FRR) at a false acceptance rate (FAR) of 0.001 (1 in 1,000) is given for a representative state-of-the-art algorithm. For each milestone, the year and evaluation are provided. Beginning with the FRVT 2002, the evaluations switched to a benchmark dataset provided by the US Department of State (DoS), which is comparable to the FERET dataset" [13].

Once again, law enforcement continued to push the boundaries of facial recognition. In 2011, Pinellas County Sherriffs Office in Florida assembled a forensic database that pulled images from Florida's Department of Highway Safety and Motor Vehicles. By equipping officers with body cams, taking pictures of suspects, and cross-referencing those images with the database, officers were able to identify and arrest criminals in a much quicker fashion [14]. Also in 2011, facial recognition software was used to verify the death of Osama Bin Laden [14]. Law enforcement organizations throughout California, as well as the FBI, DEA, DOJ, and U.S. Marshalls, began implementing facial recognition in the field as well. These agencies began developing networks to share the information they gathered, and by 2014, the FBI's Interstate Photo System provided "law enforcement agencies across the country the ability to process up to 52 million images associated with criminal identities" [14]. This desire to pool and disseminate information within government agencies has resulted in a massive collection of data that continues to grow with each passing year.

The advent of social media allowed for a testing ground that would have been unimaginable to researchers in previous decades. What was seen as a controversial technology in 2001 at Super Bowl XXXV had now become commonplace for most by the 2010s. Facebook began implementing facial recognition algorithms in 2010 and while this news was somewhat controversial at the time, the flourishing new company continued to grow. As a result, "more than 350 million photos are uploaded and tagged using face recognition each day" [14]. With the introduction of Facebook's DeepFace system in 2015, the accuracy rating improved to 97.25% [15].

In Russia, Artem Kukharenko and Alexander Kabakov have developed an application that allows users to compare pictures to the "200 million profile images on a Russian social media platform" [16]. Dubbed FindFace, this software can be used to identify strangers in a crowd with a 70% confidence level. Regarding the application of this technology, creator Kabakov stated, "If you see someone you like, you can photograph them, find their identity, and then send them a friend request...It also looks for similar people. So you could just

upload a photo of a movie star you like, or your ex, and then find 10 girls who look similar to her and send them messages" [16].

With the advent of the iPhone X in 2017, the ability to use one's face as a unique identifier had many rushing to the stores. In China, users consistently use facial recognition as a means to conduct financial transactions, not just online but in retail stores as well. Train passengers in Shanghai will soon not even have to buy tickets but simply walk through the station [17]. CCTV cameras are being added every day, and by 2020, China will likely be using over 500 million of these cameras [17]. Thus, the infrastructure necessary for facial recognition software continues to expand.

FaceFirst has also recently introduced a new model that will provide FRT to virtually any business that desires it. This Watchlist as a Service (WaaS) will notify the user of anyone that appears to be a possible threat. This product reinforces the idea that facial recognition has improved a great deal over recent years and is now available to almost everyone [14].

Though this is only a partial history of the progression of facial recognition software, it is evident that this new technology will continue to grow in its efficiency and accuracy. Each day FRT becomes more ubiquitous and the applications grow more varied. Due to its exponential growth, it will be difficult to predict its degree of pervasiveness in the coming decades.

2.3 Related Work

The failures of common attendance recording practices have been evident for years. Much research has been done in this area, and many solutions have been given. However, very few ideas have used FRT.

In 2009, Mohd Ikhsan Moksin and Norizan Mohd Yasin published a paper that recommended scanning the barcodes found on student identification cards as a means to record attendance [18]. Similarly, researchers in Malaysia encouraged the use of student ID cards coupled with Radio Frequency Identification (RFID) technology in 2012 [19]. By scanning

each student's ID card as he or she walks through the door, a system is able to verify the presence of the student and record the information to a database. However, these methods can take too much time if the class size is large and can be subject to fraud if a student scans his fellow classmates' cards.

In 2015, researchers at Ibaraki University in Japan created a system using a low energy Bluetooth beacon and the students' personal Android devices. During class, a "magic number" is sent to each student's smartphone. The phone then replies with the magic number and student ID, and the small broadcast area forces students to be in the classroom [20]. The primary issue with this proposal is that each student must have carry a compatible device.

Fingerprint and iris scanners have been suggested as well. Once again, though, these methods take some time. Also, additional equipment is required for these suggestions. Lastly, these ideas can be seen as invasive since many people tend to guard their personal information, such as fingerprint and iris patterns.

Therefore, many researchers have instead turned to facial recognition for attendance systems in the last few years. A paper published by Refik Samet and Muhammed Tanriverdi in 2017 encourages this very option. Together, they developed "a filtering system based on Euclidean distances calculated by three face recognition techniques, namely Eigenfaces, Fisherfaces and Local Binary Pattern, has been developed for face recognition" [21]. By using separate mobile applications for teachers, students, and parents, every party involved is able to manage and monitor the attendance process. The data is sent to a cloud-based server, and RESTful web services were used for communication.

Each application presents different permissions to the user. The teacher application allows for full administrative access. Typically, the teacher will take a picture of the classroom during the lecture. The picture will then be sent to the cloud server where facial detection and recognition will occur. The results are then recorded in a database and also sent to the teacher immediately through the app. The teacher can also create, edit, and remove

student profiles. The uploaded photo in each student profile is used for recognition purposes. Additionally, attendance results can be requested in an Excel spreadsheet. The student application also allows the student to edit his or her profile. The student can also view his or personal attendance records. If the student is recorded as missing, then he or she will be sent a notification. If this is deemed an error, the student can message the teacher about the issue. The family application gives the same permissions as the student application but allows for multiple children to be listed under one parent [21].

As with all facial recognition software, facial detection is needed first before the program can continue. These researchers chose the Viola-Jones face detection method with Adaboost training to identify faces in an image. "In the most basic sense, the desired objects are firstly found and introduced according to a certain algorithm. Afterwards, they are scanned to find matches with similar shapes" [21]. Deciding to utilize an appearance-based approach over a feature-based approach allowed the researchers to reduce the time and computational power needed to recognize faces. The three most popular appearance-based techniques are Eigenfaces, Fisherfaces and LBP. While all three algorithms are used, LBP is given preference if none of the algorithms return matching results.

In the end, this project worked adequately. The average accuracy rating ranged from 69% to 84% depending on the number of recognition processes ran and the amount of pictures per student profile [21]. Distance from the camera was the greatest challenge as the success level greatly diminished with the students sitting in the back of the class. Also, when the picture is being taken, there is a possibility that the student could be blocked from view. It is recommended that further experiments be done with higher quality cameras and more powerful mobile devices.

In the paper "Student Attendance System in Classroom Using Face Recognition Technique" which was authored by Samuel Lukas, Aditya Rama Mitra, Ririn Ikana Desanti, and Dion Krisnadi, the researchers attempted to enhance facial recognition "by combining Discrete Wavelet Transforms (DWT) and Discrete Cosine Transform (DCT) to extract the

features of students face" [22]. DWT allows for efficient image processing by decomposing pictures into its scaling function and wavelet coefficients. The Discrete Cosine Transforms (DCT) allows for domain scaling, de-correlation, energy compaction, separability, and symmetry [22]. Thus, DCT can efficiently manage data without causing distortion. However, all images must be the same size since this method does not allow scaling. This effort yielded an 82% success rate, though recognition process and feature extraction could be improved [22].

Cao Weifeng and Li Bin authored a paper detailing their work in facial recognition software. In "Attendance System Applied in Classroom Based on Face Image," a main focus was dealing with skin color [23]. Given the options of RGB, YCbCr, HSV, YIQ, etc., Weifing and Bin decided to use the YCbCr color space to more easily record an image's various colors within a small distinct range. Therefore, within their skin-color model, they converted an image's RGB values into a YCbCr space. They then defined a specific threshold which identified skin versus objects of differing colors. Another point of interest in their research was morphological processing. This method removes the noise from the image by creating a binary image from the picture that previously processed using the skin-color model. That which is identified as skin will be shown as white while everything else will be black. The face is then distinguished from any other white patches by the size of the target. The potential face must remain within the set size threshold to be considered a face. The face is then denoted by a red rectangle. The Principle Component Analysis (PCA) algorithm is then utilized to extract facial features. The software was tested on students in a laboratory and amphitheater [23]. The result was a 100% accuracy rating, but the sample size was small and the tests were few.

From the preceding work, one can take note of the successes and failures to enhance one's own facial recognition software. The features and structure of Refik Samet and Muhammed Tanriverdi's create a solid foundation. The paper by Samuel Lukas, Aditya Rama Mitra, Ririn Ikana Desanti, and Dion Krisnadi provides a better understanding of the challenges

one may face and how to best improve image processing. Finally, Cao Weifeng and Li Bin's paper demonstrates multiple methods of facial detection and how to best combine these approaches.

Chapter 3

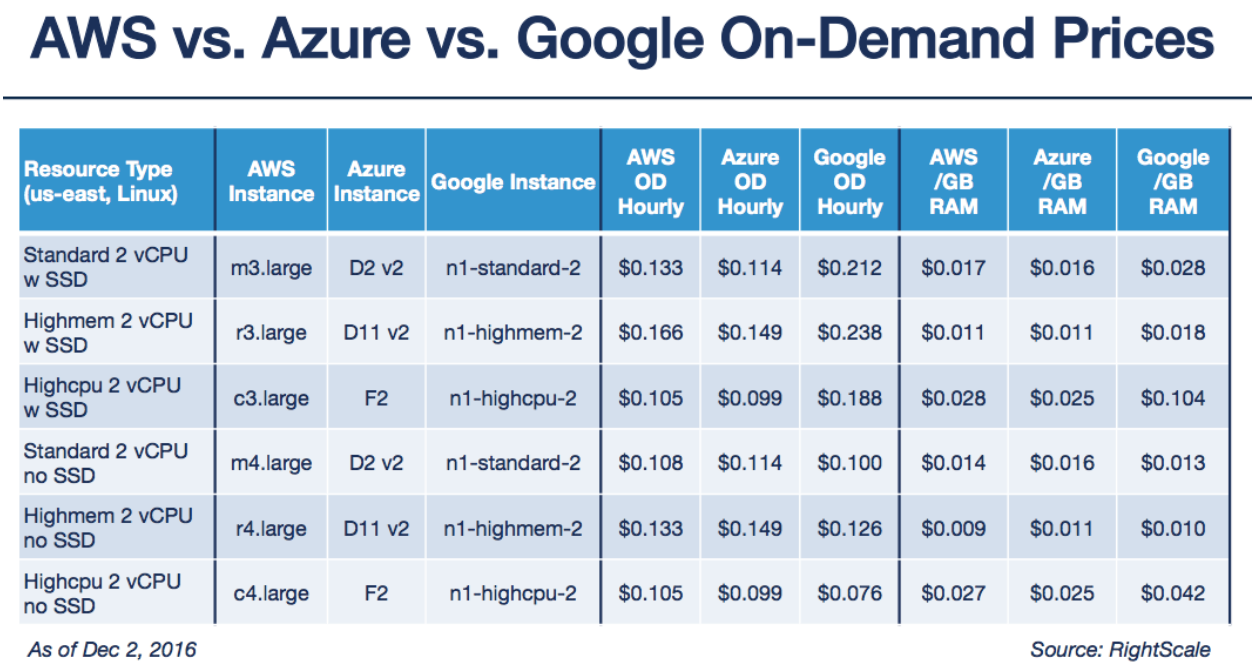
Solution

3.1 Library Options

When developing FRT, there are several options currently available in regards to libraries. Depending on one's goals, objectives, and budget, one may choose a proprietary route or an open source path. Some of the most popular avenues lie with Amazon Rekognition, Microsoft Azure, and OpenCV.

Amazon Rekognition provides an easy to use application programming interface (API) that provides a programmer many tools and functions to build the desired product. Since Rekognition is compatible with other Amazon Web Services (AWS), one can use Amazon S3 to store a large quantity of information. AWS Lambda is also available to grant the user the ability to run event-driven applications with little trouble. This option also allows for one to utilize Amazon Kinesis Video Streams. This presents a means to analyze live video, such as classroom lectures. The Rekognition API allows for facial recognition, as well as object, activity, or even text detection. The ability to perceive unsafe content and track suspicious individuals is also available. This API can allow software to distinguish emotions or verify identities. Since this service is constantly being trained by millions of new images, the accuracy and efficiency of Amazon Rekognition continues to improve. It is also highly scalable. However, this service does cost money and the proprietary software prevents one from editing the underlying code [24].

Microsoft Azure is a cloud computing service which provides a plethora of tools, libraries, languages, and frameworks. Topping out at over 600 unique services, this option creates a nearly endless supply of opportunities for designing and implementing applications. The Azure Face API possesses nearly all the capabilities of Amazon Rekognition and is compatible



AWS vs. Azure vs. Google On-Demand Prices

Resource Type (us-east, Linux)	AWS Instance	Azure Instance	Google Instance	AWS OD Hourly	Azure OD Hourly	Google OD Hourly	AWS /GB RAM	Azure /GB RAM	Google /GB RAM
Standard 2 vCPU w SSD	m3.large	D2 v2	n1-standard-2	$0.133	$0.114	$0.212	$0.017	$0.016	$0.028
Highmem 2 vCPU w SSD	r3.large	D11 v2	n1-highmem-2	$0.166	$0.149	$0.238	$0.011	$0.011	$0.018
Highcpu 2 vCPU w SSD	c3.large	F2	n1-highcpu-2	$0.105	$0.099	$0.188	$0.028	$0.025	$0.104
Standard 2 vCPU no SSD	m4.large	D2 v2	n1-standard-2	$0.108	$0.114	$0.100	$0.014	$0.016	$0.013
Highmem 2 vCPU no SSD	r4.large	D11 v2	n1-highmem-2	$0.133	$0.149	$0.126	$0.009	$0.011	$0.010
Highcpu 2 vCPU no SSD	c4.large	F2	n1-highcpu-2	$0.105	$0.099	$0.076	$0.027	$0.025	$0.042

As of Dec 2, 2016 *Source: RightScale*

Figure 3.1: On-Demand Compute Pricing: AWS vs Azure vs Google Cloud

with all of Azure's database management, storage, and software services. Obviously like AWS, though, Microsoft Azure is neither free nor open source [25].

The final option discussed is the open source library Open Source Computer Vision (OpenCV). Launched in 1999 by Intel, this project has been consistently active for the last two decades. The objectives of this enterprise are stated as follows:

- "Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free with a license that did not require code to be open or free itself" [27].

Originally written in C and updated to a primary C++ interface, OpenCV also allows for Python, Java, and MATLAB bindings as well. From facial recognition and image segmentation to motion tracking and augmented reality, OpenCV provides a range of tools to develop such applications. This library also runs on every major operating system. Since it is open source, the transparency of the code and the freedom from monthly payments is of great benefit. However, since OpenCV does not automatically come with the host of features that a cloud computing service like Azure or AWS does, one must be careful to install every language, integrated development environment (IDE), library, etc., needed to accomplish the proposed task.

Though Azure and AWS offer a myriad of benefits to the user, it was decided that the open source option was optimal for this project since it is free, and the scope of the assignment is relatively small. To escape the compatibility issues that could be caused if the program is used on an unprepared system, it was decided that this project would run in a Linux environment via a virtual machine using VMWare, the distro being Ubuntu Linux Desktop 16.04 LTS 64-bit. Python was selected as the primary programming language with the version number being 3.5. The dlib library (version 19.9) is installed to aid in image processing and machine learning among other things. The machine learning library TensorFlow (1.5) is included as well. Keras 2 is included to simplify the interface. Adam Geitgey's face_recognition Python API is utilized also. PyCharm Community Edition is the IDE of choice, and finally, OpenCV 3.2 is used as well. All of this is included in a pre-configured virtual machine to be transferred as a .tar available to all interested parties.

3.2 Cascade Classifiers

Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" described a method that could be used to detect specific objects in an image [28]. The ViolaJones object detection framework has since become a popular

option for face detection. Using Haar-like features which utilizes the intensity of surrounding pixels, this strategy once trained can be used to identify features, such as eyes, noses, cheeks, etc. However, this process will take a relatively long time, so the concept of Cascade of Classifiers was introduced. By categorizing features into general and specific stages, a program can first run a general group of feature searches on a given area. If this technique does not detect a potential face initially, then there is no reason to continue the process, and the program will continue on to the next area. This is a much quicker approach, instead of closely analyzing each individual pixel. Therefore, Haar-cascade detection is the method of choice for this project.

3.3 Face Recognition Algorithms

Once the language, libraries, and operating system have been decided, the algorithms must then be considered. Three specific algorithms are automatically included with the OpenCV library. They are Eigenfaces, Fisherface, and Local Binary Patterns Histograms (LBPH). Each one possesses its own strengths and weaknesses, so it is important to first analyze these algorithms and understand how they work.

As mentioned previously, in 1987, Sirovich and Kirby sought a means of encoding a face image into as few values as possible to minimize overhead [4]. Using principal component analysis (PCA), "a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables" [29], allowed them to accomplish their goal. By breaking down a series of face images into different values based on the subject's features, these resulting Eigenpictures could be used as templates to compare with new photos. These "building blocks" could then be used to describe a new face based on shared characteristics of the base images. This concept works off the principle that most faces share similarities. By combining those similar traits from multiple templates, the original face can be restored. From this idea, M. Turk and A. Pentland and introduced the Eigenface method in 1991

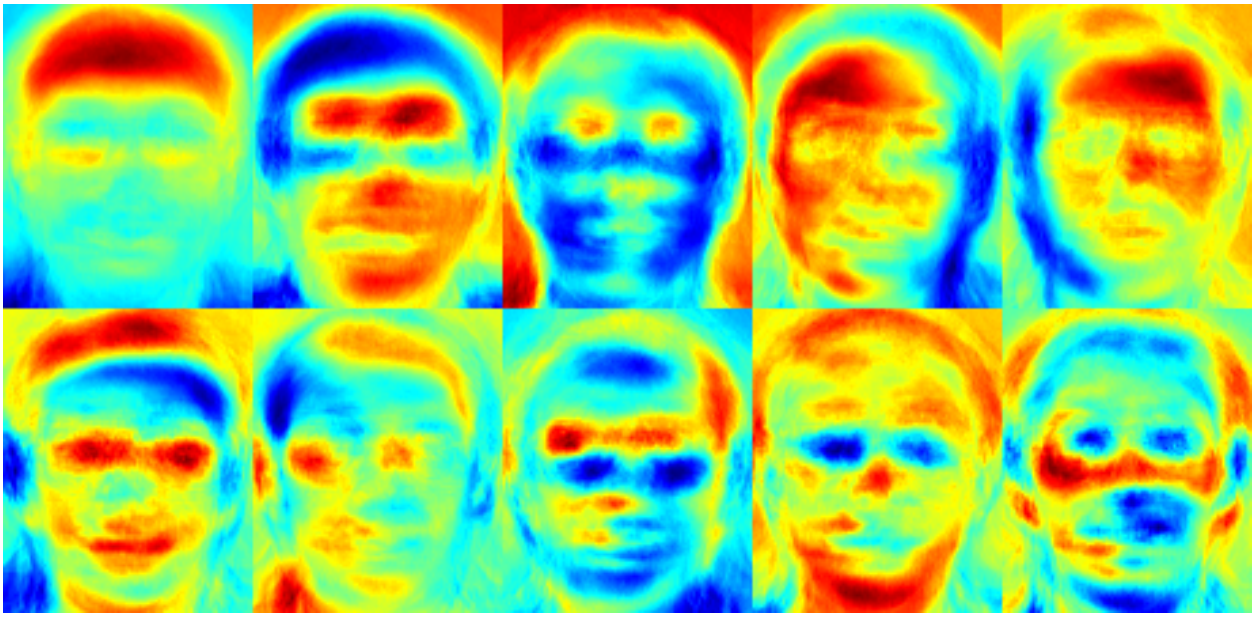


Figure 3.2: Sample Eigenfaces provided by OpenCV documentation that demonstrates both facial features and illumination are encoded [30]

specifically for facial recognition [5]. Their goal was to make the basic concept even more efficient, so they created a means "to extract the eigenvectors based on matrices sized by the number of images rather than the number of pixels" [5]. However, while this method performs well when reducing overhead, it suffers when the training set is limited, and the image size variance is great.

While the Eigenfaces strategy deals with examining the similarities between faces, the Fisherfaces method concentrates on the differences. This scheme makes it easier to differentiate between people. Using the work done by Turk and Pentland as a foundation, Belhumeur, Hespanha and Kriegman further developed their ideas and introduced the Fisherfaces method in their paper "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection" (1997) [31]. Researcher Naotoshi Seo describes the procedure by stating, "The Fisherface method is an enhancement of the Eigenface method that it uses (British statistician Ronald) Fishers Linear Discriminant Analysis (FLDA or LDA) for the dimensionality reduction. The LDA maximizes the ratio of between-class scatter to that of within-class scatter, therefore, it works better than PCA for purpose of discrimination. The Fisherface is

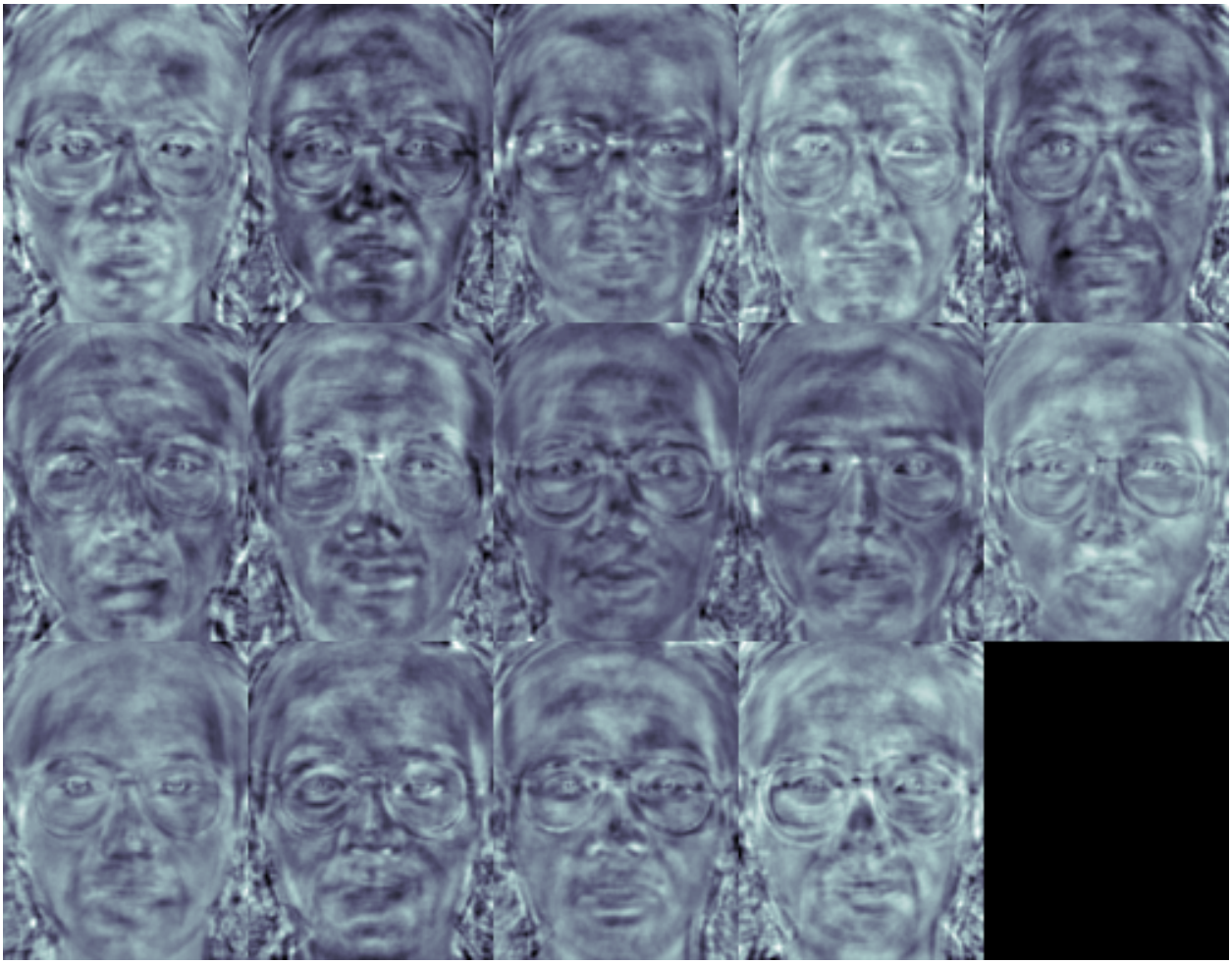


Figure 3.3: Sample Fisherfaces provided by OpenCV documentation that demonstrates distinctions between persons but does not account for lighting conditions [30]

especially useful when facial images have large variations in illumination and facial expression" [32]. While this technique could be seen as an improvement by some, the Fisherfaces method still has some weaknesses. The algorithm can be wasteful and costly while also needing a large data set like its predecessor.

The remaining approach is the Local Binary Patterns Histograms (LBPH) method. This algorithm was introduced in 1994 by T. Ojala, M. Pietikinen, and D. Harwood [33]. While the prior algorithms use entire datasets to develop a mathematical representation of an image, the LBPH technique examines each new image, develops a description, and compares the image with the every other image separately. This strategy divides a picture

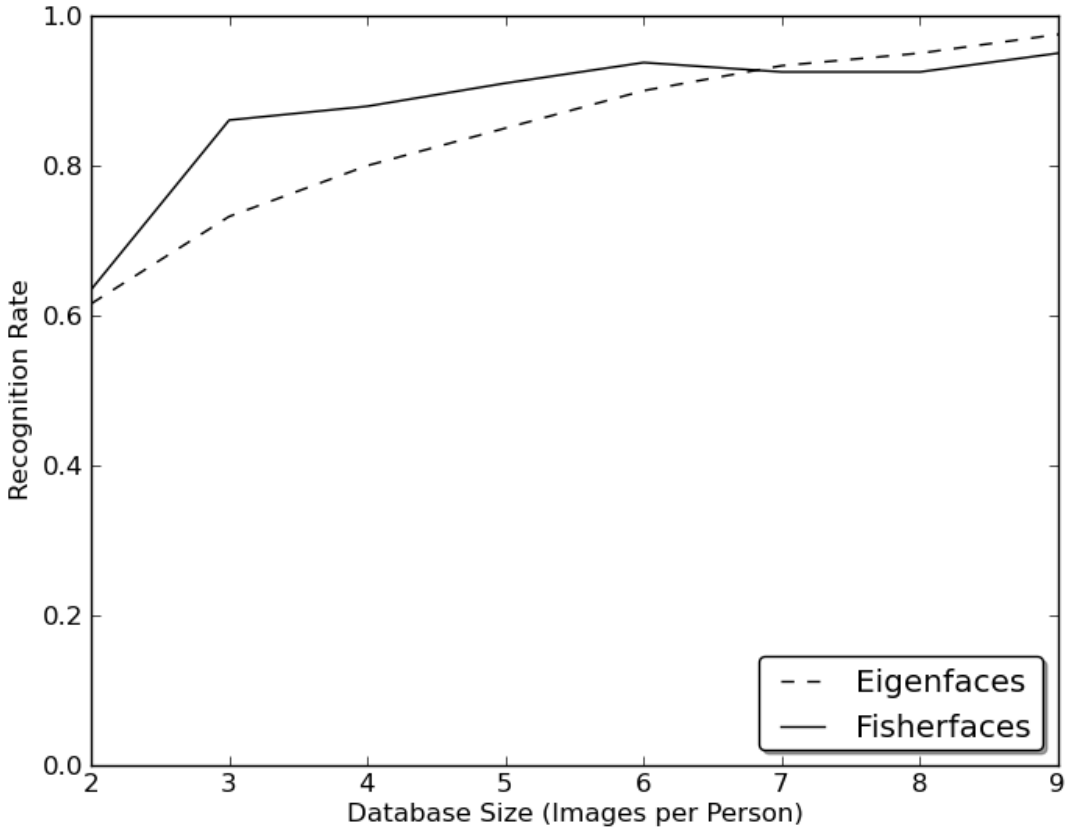


Figure 3.4: Provided by OpenCV documenation, this figure is a comparison of Eigenfaces and Fisherfaces in regards to face recognition. [30]

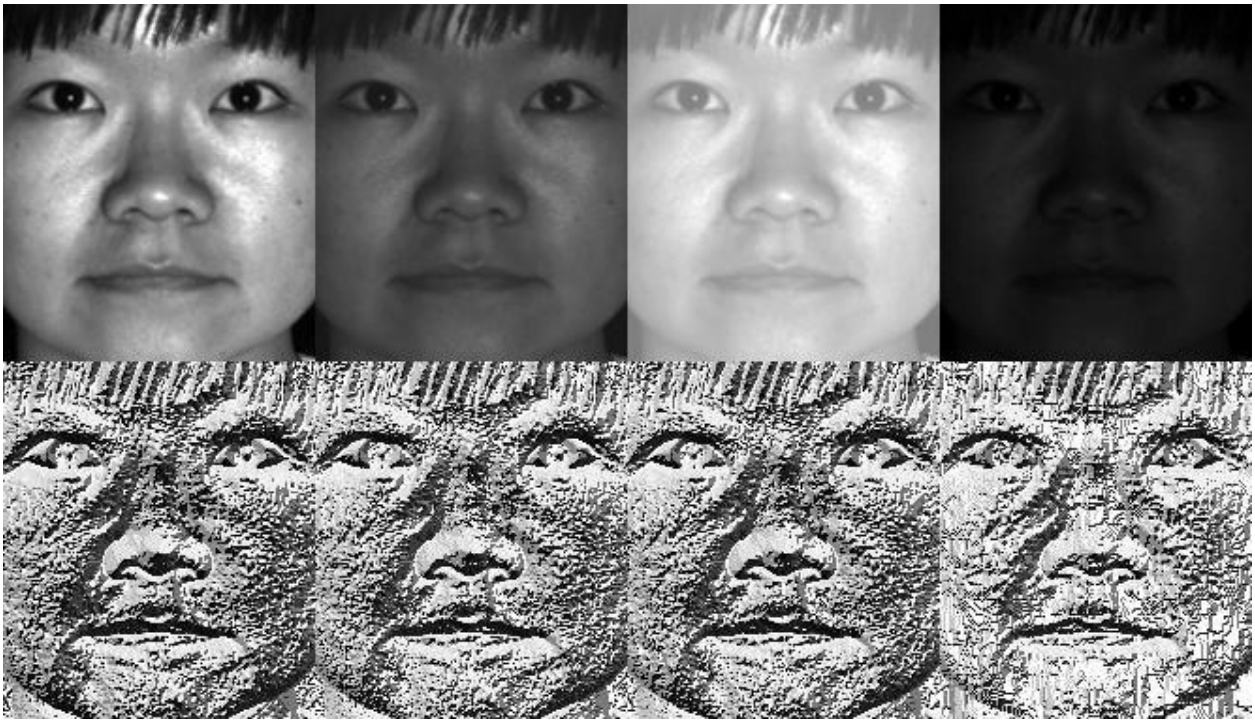


Figure 3.5: Provided by OpenCV documenation, this figure demonstrates the results of the LBPH algorithm in different lighting conditions. [30]

into cells and defines each pixel binarily based on the intensity of the center pixel in the cell. These patterns characterize each image and allows for simple recognition if the patterns of two images share similarities within a given threshold. Once again, though, no algorithm is perfect and a dataset of at least 10 photos per person is recommended to increase confidence in recognition.

There are many variables to consider when developing FRT. Lighting, for instance, can cause a great deal of deviation, even among pictures of the same person. The primary issue for this project is the lack of photos given per student. By using each student's student ID picture, the project is limited to one sample image per person. Studies have been done regarding this "one sample per person" problem. In "Face Recognition from a Single Image per Person: A Survey," Xiaoyang Tan, Songcan Chen, Zhi-Hua Zhou, and Fuyan Zhang suggest that extra information can be derived using other biometrics, such as voice patterns or fingerprints [34]. However, if this is not possible, they conclude that no algorithm currently

exists that can solve the "one sample per person" problem completely. Therefore, the goal should be expanding the current dataset if possible, and, perhaps, using these algorithms in tandem.

3.4 Design

3.4.1 Functional Requirements

- Provided pictures should be coupled with students' names to create specific training sets for each image.
- Detect each face in lecture video.
- Recognize faces based on training set.
- Confidence levels should be assigned to each recognition.
- If the confidence level is over 90%, face image should be cropped and saved to that student's training set.
- Results should be written to a .csv file.

3.4.2 Non-functional Requirements

- Reliability: The software should consistently return the correct recognition (90% success rate).
- Speed: The results should be returned within a reasonable amount of time (10 minutes).
- Scalability: New students should be able to be added with little difficulty.

3.4.3 Initial Use Cases

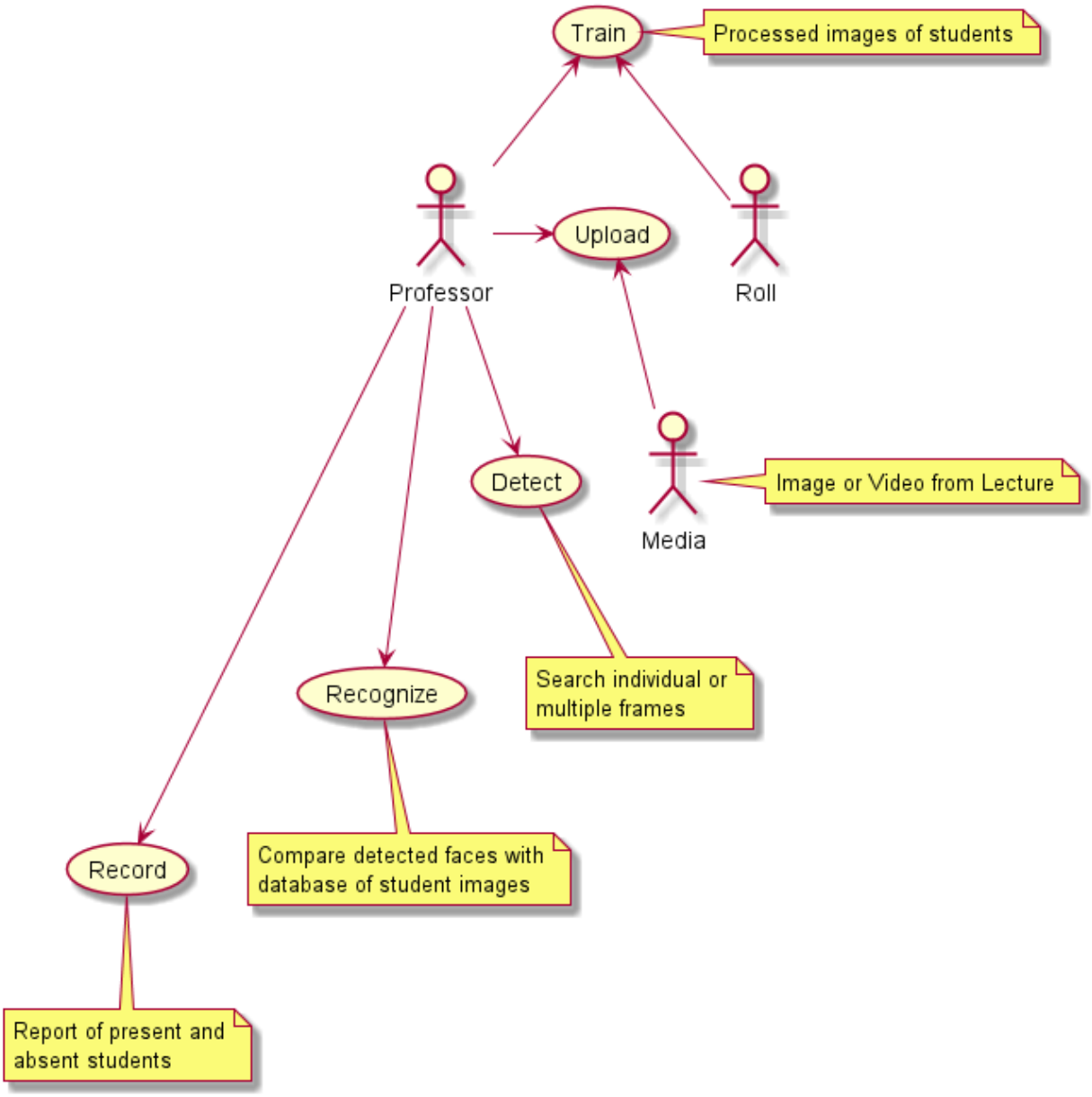


Figure 3.6: Use Case Diagram

Use Cases	Description
Upload Media	The user is able to upload images or video for roll or comparison.
Train Program	The program processes images from roll and stores values.
Face Detection	The user requests the program to detect face from media.
Face Recognition	The program matches detected faces with roll.
Report Results	The comparison results are presented to user

Figure 3.7: Use Case Intro

Upload Media	
Use Cases	Description
Case No.	1
Name	Upload Media
Actor	Professor, Media
Description	The user requests to upload media to program
Pre-condition	The program interface must be open
Post-condition	Uploaded media is stored by program
Flow of Events	
Steps	Description
1	User clicks upload button
2	File explorer pops up and allows user to browse files
3	Selected file(s) are uploaded and stored by program
Entry Conditions	
User must have interface open	
Exit Conditions	
User selects Cancel	

Figure 3.8: Use Case 1

Train Program	
Use Cases	Description
Case No.	2
Name	Train Program
Actor	Professor, Roll
Description	The program processes roll images
Pre-condition	Roll images must be uploaded
Post-condition	Roll images broken down into values
Flow of Events	
Steps	Description
1	User uploads roll images
2	Program systematically examines each image
3	Values of each image are stored in .txt file
Entry Conditions	
Roll images must be uploaded	
Exit Conditions	
User selects Cancel	

Figure 3.9: Use Case 2

Face Detection	
Use Cases	Description
Case No.	3
Name	Face Detection
Actor	Professor
Description	The program detects faces from lecture image or video
Pre-condition	Roll images and Lecture image/video must be uploaded
Post-condition	All faces in the lecture image/video will be identified and highlighted
Flow of Events	
Steps	Description
1	User selects face detection button
2	Program searches image/video for faces
3	Faces are processed and stored as values in .txt file
Entry Conditions	
Roll images and Lecture image/video must be uploaded	
Exit Conditions	
User selects Cancel	

Figure 3.10: Use Case 3

Face Recognition	
Use Cases	Description
Case No.	4
Name	Face Recognition
Actor	Professor
Description	The program compares detected faces with roll images
Pre-condition	Face Detection has been run on current image
Post-condition	Successful comparisons will be stored in .txt file
Flow of Events	
Steps	Description
1	User selects face recognition button
2	Values of roll and lecture image/video will be compared
3	Successful comparisons will be stored in .txt file
Entry Conditions	
Face detection must be run on current image	
Exit Conditions	
User selects Cancel	

Figure 3.11: Use Case 4

Record Results	
Use Cases	Description
Case No.	5
Name	Report Results
Actor	Professor
Description	Successful comparisons are displayed to user
Pre-condition	Face Recognition has been run on current image
Post-condition	Successful comparisons will be displayed in easily printable format
Flow of Events	
Steps	Description
1	User selects results button
2	Results are displayed to user via interface
3	Results can be saved locally
Entry Conditions	
Face recognition must be run on current image	
Exit Conditions	
User selects Cancel	

Figure 3.12: Use Case 5

3.4.4 Software Methodology

The intended primary actor of this software is the professor. The professor will first upload a collection of photos that feature each student. A .csv file with the corresponding image file name should be included as well. The program will create a group of folders, one for each student. The professor will then provide a lecture video or image. When the program is ran, facial detection will commence. The software will then attempt to match the detected faces with the images stored in the training sets. The recognition portion will note the confidence levels and include the results in a .csv file. If the confidence level is above 90%, the student will be considered present, and the student's image will be cropped and stored in that student's folder for future use, thus providing training for each execution.

3.4.5 Software Architecture

For the first use, a script named "createFolders.py" has been provided to organize the training images based on each student's name. Using a .csv file that gives the relative path name in one column and the student name in the other, this program will either gather all the images into one folder or create a unique folder for each student based on the needs of the user. Creating folders for each student allows for a location to save future images for that person.

The primary program "faceRecog.py" can then be ran to accomplish the goal of face recognition. The Python file will first import all the necessary packages, such as cv2, os, numpy, and the face recognition api developed by Adam Geitgey [35]. The "faceDetect.py" script will also be imported as well. faceRecog.py will then parse through each image file in the designated path and assign the paths and corresponding names to separate arrays. The software will then loop through each file, load the image into the program, encode the picture with a 128 value description, and assign those figures to a two-dimensional array. This array's dimensions will be the number of students by the 128 values. An if-else statement is included that allows for this array to be stored in an .npy file, so these calculations do not need to be made again. This .npy file only needs to be loaded into the program when the user uses this software again which saves valuable time.

FaceRecog.py will then call faceDetect.py. Within faceDetect.py, four different cascade classifiers may be used to detect those in the sample video. Those cascade classifier are the following: haarcascadeFrontalfaceAlt.xml, haarcascadeFrontalfaceAlt2.xml, haarcascadeFrontalfaceDefault.xml, and haarcascadeProfileFace.xml. These files can be found within the Opencv repository on Github [36]. These were found to be the most reliable for the given scenarios. Only one is necessary, but all four can be used simultaneously to increase accuracy. The face_recognition api also offers a face location method as well.

FaceDetect.py will open the lecture video and begin sending each frame through the loop. If necessary, the program can be altered to only process every ith frame to improve

speed. The cascade classifier will then be ran against the given image to detect the location of a face in the picture. A four digit array will be returned describing the border around which the face was found. The face location will be saved in an array, and then facial detection will resume typically until the number of captured frames is equal to the number of total frames in the video. Similar to the student roll, the face location array can be saved to another .npy file if the user wishes to run another instance using the same video. The face locations array is then returned and passed to the faceRecog.py file.

Within the faceRecog.py file, the lecture video is then opened and encoded in the same manner as the original student images. The unknown face's 128 value descriptor is then compared to each face encoding from the student roll once the program ensures that it is within the proper parameters. If the program locates a face that is very similar to a currently known face, then a value of "True" will be returned for that element within the array. The tolerance can altered as well to allow for stricter or more lenient conditions. If the face hasn't been recognized previously, then the name will be included in a list of students who are considered "present." This will continue until every face location has been examined.

A window will also be opened that reveals which faces are being detected and recognized. Additionally, each new recognized face will be cropped, and saved to the student's personal folder. This will create more data to allow for easier recognition of a student during the next iteration. This data can then be fed into the aforementioned facial recognition algorithms, such as Eigenfaces, Fisherfaces, and LBPH. By using training programs, such as Anirban Kar's face recognition software from codacus.com, one can continue to improve the performance of the attendance recording process [38]. Finally, the list of recognized faces will be stored for the user in a text file with easy readability.

3.4.6 Development Process

Using a process most similar to Scrum, each week was given an objective to pursue. On weeks where an objective was unable to be completed, the goal was reevaluated and adjusted

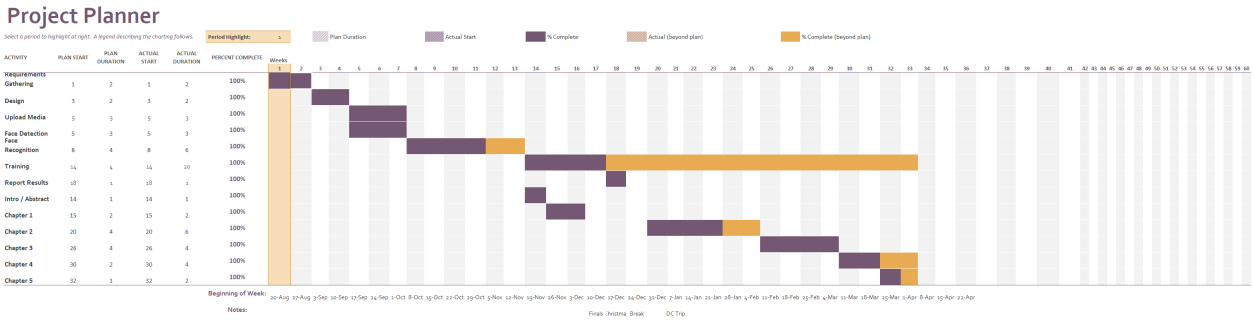


Figure 3.13: Gantt Chart showcasing progression of project.

at the beginning of the next week. The following roughly describes the process undergone to accomplish this project.

3.4.7 Summary

After determining the necessary libraries, algorithms, and design, the coding began. Many challenges, such as the "one sample per person" problem were presented. However, each problem was met with a solution. The following chapter showcases the results of this project.

Chapter 4

Solution Validation

4.1 Face Detection

Detecting faces in a crowd can be a difficult task, but since the software is intended for classroom use, it can be tailored for the appropriate audience. By setting reasonable tolerances, most faces can be detected immediately as in the figures below. However, if a camera pans over the classroom, the detection rate increases tremendously.

Using haarcascadeFrontalfaceAlt.xml allows solid detection results in a group environment. Typically, 85% of faces are detected. The greatest difficulty is dealing with those students on the front row. Due to the angle of the camera and the tendency of students to look down at their notes, it can be difficult to obtain a detection if both eyes are not visible. However, when coupled with additional cascade classifiers, the detection rate can increase to 95% depending on the lecture video and the position of each student's face. However, more false positives will likely also be introduced.

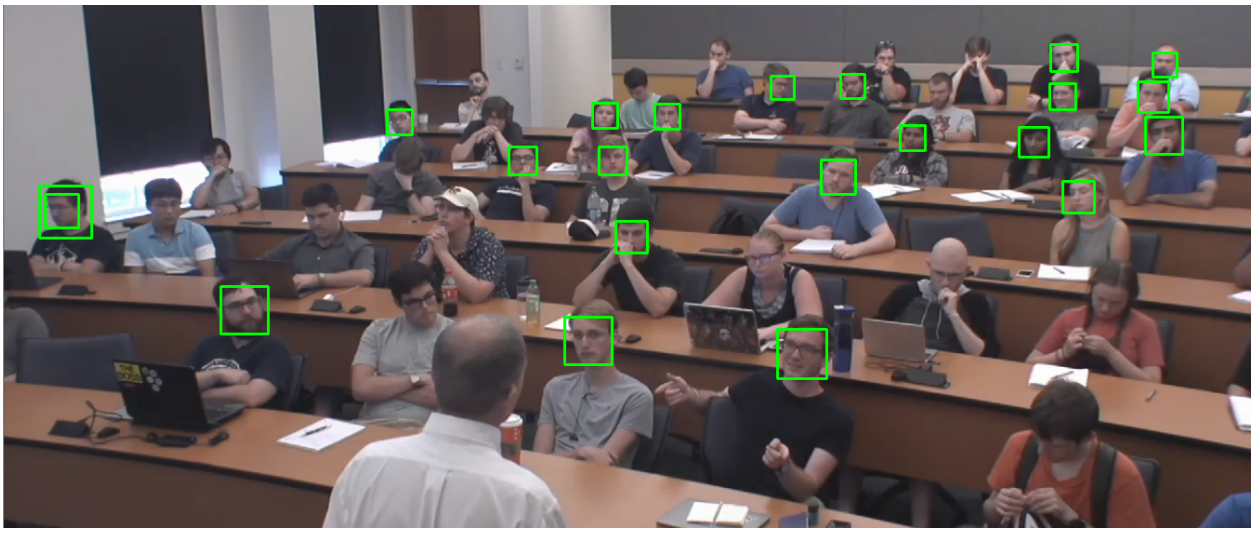


Figure 4.1: Face Detection (Left)

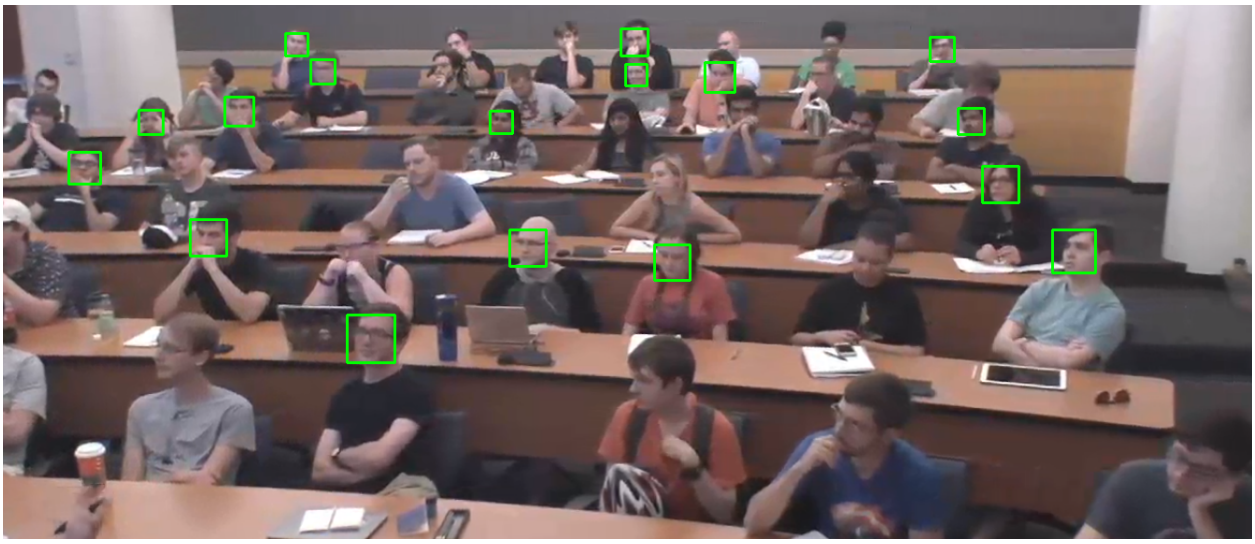


Figure 4.2: Face Detection (Right)

4.2 Face Recognition

Though this software can correctly identify nearly half of those present on average, these results prevent this software from being initially reliable. The tolerance level can be lowered to enhance accuracy and prevent false positives, but if the initial sample image is too dissimilar from the current student's appearance due to age, facial hair, glasses, etc., the software will have difficulty recognizing the student. Additionally, determining a confidence level is also not possible since a sizable amount of data is needed to compare against a sample image. The best process for using FRT is likely to have each student submit a collection of recent photos at the beginning of the semester. These images can then be used to train an algorithm, such as LBPH. Thus, the best use for this software is likely to collect new data upon recognition of students' faces assuming that the initial photo is somewhat similar.

Another area of improvement is the speed of the program. For a video containing nearly 7000 frames or roughly four minutes of video, the detection and recognition process can take over thirty minutes to complete. To speed up the process, one can choose to skip a designated amount of frames. However, this will affect the overall accuracy of the software. The face locations and face encodings can also be saved for future use if the user needs to run the software again on the same video.

Implementing the face recognition feature can be tricky. Once again, since only one image is initially provided for each student, recognizing a student in the beginning can be difficult if the student fails to directly look towards the camera. This creates scenarios that result in drastic inconsistencies and varying results. Since most of the current work only deals with searching for specific faces in limited space, there is very little information regarding the use of FRT in this scope. This project is intended to be used on a large scale across multiple classes. Thus, accuracy tends to diminish as convenience grows. This further proves the concept of "no free lunch." If one wishes to maintain complete precision and accuracy, using training programs, such as the one mentioned previously, is the best option. However, 20 to 30 samples must be collected for each individual, and the software must be trained

Figure 4.3: Face Recognition

manually for each student in the class. This is nearly impossible for large classes, so the best option is the continued improvement of this facial recognition feature.

4.3 Final Results

Many variables have to be considered when conducting facial detection and recognition. The use of facial detection technology continues to advance at a great rate. However, with only one sample per person, facial recognition can be extremely difficult and demands many tests before it can be used effectively.

Chapter 5

Conclusions and Future Efforts

The role of facial recognition technology (FRT) will continue to grow in society as advancements are made. Through the many applications provided by FRT, the general public, as well as government entities, will continue to benefit from these new opportunities. Through this paper, it is clear that this technology will become exponentially more sophisticated within the coming decade. The widespread use of FRT by individuals further supports this point.

However, there is still a lot of progress that must be made before the general public can fully experience the same benefits that large companies and intelligence agencies use. Due to deficiencies in the Opencv library and accompanying APIs, the ability to match a great number of known faces to an equally large crowd can still be a challenge, especially when one considers the "one sample per person" problem. This does not prevent one from using FRT, but it does require more manual input from the user. To better facilitate the use of this software in an academic environment, functions and methods that provide more accurate comparisons should be developed. Other methods of optimization need to also be considered in regards to how an image is encoded and how each frame is handled since the majority of time lost is during these events. With that being said, this an exciting time for facial recognition research, and many improvements will likely be made within the coming years.

Bibliography

[1] Norman, J. (2018). Woodrow Bledsoe Originates of Automated Facial Recognition (1964 1966) : HistoryofInformation.com. [online] Historyofinformation.com. Available at: http://www.historyofinformation.com/expanded.php?id=2495 [Accessed 9 Jan. 2018].

[2] J. Goldstein, L. D. Harmon, and A. B. Lesk, Identification of human faces, Proceeding IEEE, May 1971 vol. 59, no. 5, pp. 748-760 [Accessed 9 Jan. 2018].

[3] Manikpuri, M. (2018). Biometric Security Systems for Beginner. 1st ed. Educreation Publishing, pp.6-7.

[4] Sirovich, L. and Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. Journal of the Optical Society of America A, 4(3), p.519.

[5] Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. Journal of Cognitive Neuroscience, 3(1), pp.71-86.

[6] NIST. (2011). Face Recognition Technology (FERET). [online] Available at: https://www.nist.gov/programs-projects/face-recognition-technology-feret [Accessed 14 Jan. 2018].

[7] "Mugspot Can Find A Face In The Crowd – Face-Recognition Software Prepares To Go To Work In The Streets". ScienceDaily. 12 November 1997. [Accessed 14 Jan. 2018].

[8] Rogers, K. (2016). That Time the Super Bowl Secretly Used Facial Recognition Software on Fans. Motherboard. [Accessed 20 Jan. 2018].

[9] Griffin, P. (2005). Understanding The Face Image Format Standards. [ebook] pp.25-26. Available at: https://www.nist.gov/sites/default/files/documents/2016/12/12/griffin-face-std-m1.pdf [Accessed 20 Jan. 2018].

[10] P. J. Phillips, P. Grother, R. J. Micheals, D. M. Blackburn, E.Tabassi, and J. M. Bone, "Face Recognition Vendor Test 2002 Overview and Summary," March 2003 ¡http://www.frvt.org¿.

[11] Phillips, J. (2010). Face Recognition Vendor Test (FRVT) 2006. [online] NIST. Available at: https://www.nist.gov/itl/iad/image-group/face-recognition-vendor-test-frvt-2006 [Accessed 22 Jan. 2018].

[12] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer and W. Worek, "Preliminary Face Recognition Grand Challenge Results," 7th International Conference on Automatic Face and Gesture Recognition (FGR06), Southampton, 2006, pp. 15-24.

[13] Phillips, P. (2011). Improving Face Recognition Technology. Computer, 44(3), pp.84-86. [Accessed 2 Feb. 2018].

[14] West, J. (2017). History of Face Recognition - Facial recognition software. [online] FaceFirst Face Recognition Software. Available at: https://www.facefirst.com/blog/brief-history-of-face-recognition-software/ [Accessed 4 Feb. 2018].

[15] Chowdhry, A. (2014). Facebook's DeepFace Software Can Match Faces With 97.25% Accuracy. [online] Forbes.com. [Accessed 4 Feb. 2018].

[16] Walker, S. (2016). Face recognition app taking Russia by storm may bring end to public anonymity. [online] the Guardian. Available at: https://www.theguardian.com/technology/2016/may/17/findface-face-recognition-app-end-public-anonymity-vkontakte [Accessed 6 Feb. 2018].

[17] McElroy, G. (2017). 10 Fascinating Facts About Facial Recognition Technology - Listverse. [online] Listverse. Available at: https://listverse.com/2017/12/18/10-fascinating-facts-about-facial-recognition-technology/ [Accessed 5 Feb. 2018].

[18] M. I. Moksin and N. M. Yasin, "The Implementation of Wireless Student Attendance System in an Examination Procedure," 2009 International Association of Computer Science and Information Technology - Spring Conference, Singapore, 2009, pp. 174-177.

[19] Kassim, Murizah and Mazlan, Hasbullah Zaini and Norliza and Salleh, Muhammad Khidhir. (2012). Web-based student attendance system using RFID technology. 213-218.

[20] Noguchi, Shota & Niibori, Michitoshi & Zhou, Erjing & Kamada, Masaru. (2015). Student Attendance Management System with Bluetooth Low Energy Beacon and Android Devices.

[21] R. Samet and M. Tanriverdi, "Face Recognition-Based Mobile Automatic Classroom Attendance Management System," 2017 International Conference on Cyberworlds (CW), Chester, 2017, pp. 253-256.

[22] S. Lukas, A. R. Mitra, R. I. Desanti and D. Krisnadi, "Student attendance system in classroom using face recognition technique," 2016 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2016, pp. 1032-1035.

[23] Weifong, C. and Bin, L. (2015). Attendance System Applied in Classroom Based on Face Image. Nanjing, China: Nanjing Institute of Technology.

[24] Amazon Web Services, Inc. (2018). Amazon Rekognition Video and Image - AWS. [online] Available at: https://aws.amazon.com/rekognition/ [Accessed 20 Feb. 2018].

[25] SteveMSFT, Roth, A. and Hu, X. (2018). Face API Service overview - Microsoft Cognitive Services. [online] Docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/azure/cognitive-services/face/overview [Accessed 20 Feb. 2018].

[26] Weins, K. (2016). AWS vs Azure vs Google Cloud Pricing: Compute Instances. [online] Rightscale.com. Available at: https://www.rightscale.com/blog/cloud-cost-analysis/aws-vs-azure-vs-google-cloud-pricing-compute-instances [Accessed 20 Feb. 2018].

[27] Bradski, Gary; Kaehler, Adrian (2008). Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc. p. 6.

[28] Jones, M. and Viola, P. (2001). Rapid object detection using a boosted cascade of simple features. [online] Citeseer.ist.psu.edu. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.6807 [Accessed 24 Feb. 2018].

[29] Jain, P. and Shandliya, V. (2013). A survey paper on comparative study between Principal Component Analysis (PCA) and Exploratory Factor Analysis (EFA). [online] Researchpublications.org. Available at: http://researchpublications.org/IJCSA/NCAICN-13/240.pdf [Accessed 2 Mar. 2018].

[30] Docs.opencv.org. (2018). Face Recognition with OpenCV OpenCV 2.4.13.6 documentation. [Accessed 7 Mar. 2018].

[31] Belhumeur, P., Hespanha, J. and Kriegman, D. (1997). Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. ACM Digital Library.

[32] Seo, N. (2006). Project: Eigenfaces and Fisherfaces -. [online] Note.sonots.com. Available at: http://note.sonots.com/SciSoftware/FaceRecognition.html [Accessed 9 Mar. 2018].

[33] Ojala, T., Pietikinen, M. and Harwood, D. (1994). A comparative study of texture measures with classification based on featured distributions. Science Direct.

[34] Tan, X., Chen, S., Zhou, Z. and Zhang, F. (2006). Face recognition from a single image per person: A survey. ACM Digital Library.

[35] GitHub. (2018). ageitgey/face_recognition. [online] Available at: https://github.com/ageitgey/face_recognition [Accessed 29 Mar. 2018].

[36] GitHub. (2018). opencv/opencv. [online] Available at: https://github.com/opencv/opencv/tree/master/data/haarcascades [Accessed 29 Mar. 2018].

[37] Anon, (n.d.). Computer vision: OpenCV realtime face detection in Python — Amori in corso. [online] Available at: https://www.lucaamore.com/?p=638

[38] GitHub. (2018). thecodacus/Face-Recognition. [online] Available at: https://github.com/thecodacus/Face-Recognition [Accessed 1 Apr. 2018].

Appendices

.1 createFolder.py

```
# This program takes a group of student images and
# places them in a separate folder for each student,
# creating a unique training set to reference
#
# The path should exactly match the path given in a
# class roll file. Similarly, the class roll file should
# formatted in a csv file with the first column including the
# relative image path name and the second column should contain
# the student's corresponding name.
#
# If the file is not in .jpg format, please alter line 51 to include
# appropriate format.

import glob, os, csv, shutil
#from shutil import copyfile

IDs = {}
fileNames = {}
folders = []

# Parse through csv file for student names
```

```
with open('C:\Users\WillM\Desktop\Facial Recognition\Roll.csv','r+') as csvfile:
    timeReader = csv.reader(csvfile, delimiter=',')
    for row in timeReader:
        print(row)
        csvFileName = row[0]
        if "/" in row[0]:
            folders.append(csvFileName.split("/")[0])
        fileNames[csvFileName] = row[1]

inputPath= "/home/facerec/Desktop/Facial Recognition/Input"
outputPath = "/home/facerec/Desktop/Facial Recognition/OutputAll"

# Assign names and copy files to output folder
folderSet = set(folders)
for folder in folderSet:
    print("Searching folder name: " + folder)
    folderToSearch = os.path.join(inputPath, folder)
    if os.path.isdir(folderToSearch):
        for oldName in os.listdir(folderToSearch):
            # Ignore files in path which aren't in the csv file
            oldNameCSV = folder + "/" + oldName
            print("Old Name: " + oldNameCSV)
            if oldNameCSV in fileNames:
                print("New Name: " + fileNames[oldNameCSV])
                print()
                try:
                    newFileName = fileNames[oldNameCSV]
```

```
                    if "/" in newFileName:
                        newFileName = newFileName.split("/")[1]
                    shutil.copyfile(os.path.join(folderToSearch, oldName), os.path.join(
                except:
                    print('File ' + oldName + ' could not be renamed to ' + IDs[oldName]

# Create folder for each student
for file_path in glob.glob(os.path.join(outputPath, '*.*')):
    new_dir = file_path.rsplit('.', 1)[0]
    try:
        os.mkdir(os.path.join(outputPath, new_dir))
    except:
        print(file_path)
        print("Already exists!")
        print()
        # Target directory already exists.
        pass
    shutil.move(file_path, os.path.join(new_dir, os.path.basename(file_path)))
```

.2 faceDetect.py

```
import cv2
import numpy as np


class FaceDetect:
```

```
def faceDetect(self):
    # List of recommended classifiers
    faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
    faceCascade2 = cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
    faceCascade3 = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    faceCascade4 = cv2.CascadeClassifier('haarcascade_profileface.xml')

    face_locations = []
    i = 0
    # Video opened
    cap = cv2.VideoCapture("sampleBetaShort.mp4")
    while not cap.isOpened():
        cap = cv2.VideoCapture("sampleBetaShort.mp4")
        cv2.waitKey(1000)
        print("Wait for the header")

    # Face Detection Loop
    # Multiple Classifiers can be used if necessary
    pos_frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
    if (True == True):
        while True:
            flag, frame = cap.read()
            # if (flag and (i%10 == 0 or i == 0)):
            if flag:
                # The frame is ready and already captured
                # cv2.imshow('video', frame)
                pos_frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
```

```
print(str(pos_frame) + " frames")

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# gray = frame

faces = face_recognition.face_locations(gray)
face_locations.append(faces)

# faces = faceCascade.detectMultiScale(
#     gray,
#     scaleFactor=1.1,
#     minNeighbors=5,
#     minSize=(30, 30),
#     flags=cv2.CASCADE_SCALE_IMAGE
# )

# face_locations.append(faces)

# if not faces:
#     print("Face #: ", k)
#     k=k+1

# print("1", faces)
# i=i+1


# faces2 = faceCascade2.detectMultiScale(
```

```
#     gray,
#     scaleFactor=1.1,
#     minNeighbors=5,
#     minSize=(30, 30),
#     flags=cv2.CASCADE_SCALE_IMAGE
# )
#
# face_locations.append(faces2)
# print("2", faces2)
#
# faces3 = faceCascade3.detectMultiScale(
#     gray,
#     scaleFactor=1.1,
#     minNeighbors=5,
#     minSize=(30, 30),
#     flags=cv2.CASCADE_SCALE_IMAGE
# )
#
# face_locations.append(faces3)
# print("3", faces3)
#
#
# faces4 = faceCascade4.detectMultiScale(
#     gray,
#     scaleFactor=1.1,
#     minNeighbors=5,
#     minSize=(30, 30),
```

```
        #     flags=cv2.CASCADE_SCALE_IMAGE
        # )
        #
        # face_locations.append(faces4)
        # print("4", faces4)

    if not flag:
        # The next frame is not ready, so we try to read it again
        cap.set(cv2.CAP_PROP_POS_FRAMES, pos_frame - 1)
        print("frame is not ready")
        # It is better to wait for a while for the next frame to be ready
        cv2.waitKey(1000)

    # Termination Conditions
    # Save Face Locations if necessary
    if cv2.waitKey(10) == 27:
        cap.release()
        # cv2.destroyAllWindows()
        print("Final Face Locations: ", face_locations)
        return face_locations
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT)
        # If the number of captured frames is equal to the total number of f
        # we stop
        cap.release()
        # cv2.destroyAllWindows()
        # iprint("Final Face Locations: ", face_locations)
        open(r'C:\Users\WillM\Desktop\Facial Recognition\savedImages\locatio
```

```
                    np.save(r'C:\Users\WillM\Desktop\Facial Recognition\savedImages\loca
                    return face_locations
                # i += 1
        # Load Face Locations if needed
        else:
            face_locations = np.load(r'C:\Users\WillM\Desktop\Facial Recognition\savedIm
            print("Face Locations: ", face_locations)
            return face_locations
```

.3 faceRecog.py

```
import cv2
import os
import numpy as np
import face_recognition as face_recognition
from faceDetect import FaceDetect

# Text file containing those in attendance
attendance = open(r'C:\Users\WillM\Desktop\Facial Recognition\savedImages\present.txt',

# Recover image paths
roll = []
rollNames = []
for path, subdirs, files in os.walk(r'C:\Users\WillM\Desktop\Facial Recognition\OutputAl
    for name in files:
        roll.append(os.path.join(path, name))
        rollNames.append(name.replace(".jpg", ""))
```

```
samplePictureArray = []
samplePictureNameArray = []
rollLength = len(roll)
samplePictureEncodingArray = np.empty((0, 128))

# Encode Picture from Student Roll
if (True == True):
    for i in range(0, rollLength):
        print(rollLength, i)
        try:
            samplePicture = face_recognition.load_image_file(roll[i])

            samplePictureEncoding = face_recognition.face_encodings(samplePicture)[0]

            samplePictureEncodingArray = np.append(samplePictureEncodingArray, [samplePi
            print(samplePictureEncodingArray.shape)

            samplePictureNameArray.append(rollNames[i])
            print(samplePictureNameArray)

        except IndexError as s:

            print("Corrupt Image: ", rollNames[i], i)
        except OSError as e:
            print("Could not recognize image: ", rollNames[i], i)
            pass
```

```
    np.save(r'C:\Users\WillM\Desktop\Facial Recognition\savedImages\save.npy', samplePic
    np.save(r'C:\Users\WillM\Desktop\Facial Recognition\savedImages\name.npy', samplePic

else:
    samplePictureEncodingArray = np.load(r'C:\Users\WillM\Desktop\Facial Recognition\sav
    samplePictureNameArray = np.load(r'C:\Users\WillM\Desktop\Facial Recognition\savedIm
    print(samplePictureEncodingArray.shape)

# Initialize some variables
face_encodings = []
face_names = []
process_this_frame = True
fd = FaceDetect()
face_locations = fd.faceDetect()
i = 0

# Video File
video_capture = cv2.VideoCapture('sampleBetaShort.mp4')
length = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))
frameLength = video_capture.get(cv2.CAP_PROP_FRAME_COUNT)
frameLength = int(frameLength)

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()
```

```
# Resize frame of video to 1/4 size for faster face recognition processing
# small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
small_frame = frame
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Only process every other frame of video to save time
if process_this_frame:
    # Find all the faces and face encodings in the current frame of video
    face_encodings = []
    print(face_locations[i])

    if len(face_locations[i]) == 2:
        print("Length of Face Locations: ", len(face_locations[i]))
        face1 = np.array_split(face_locations[i], 2)
        print("Splitting Face Locations")
        face_encodings = face_recognition.face_encodings(small_frame, face1[1], 1)

    else:
        face_encodings = face_recognition.face_encodings(small_frame, face_locations

    print('face location: ', face_locations[i])
    print('face encoding length: ', len(face_encodings))
    if not face_encodings:
        face_encodings = [0] * 128
        print(i, "Creating an array of zeros", face_encodings)
        print("Length of FaceEncoding: ", len(face_encodings))
```

```
print('face encoding2: ', face_encodings)
if np.count_nonzero([face_encodings]) < 100:
    print(i, "Ignore array of zeros: ", np.count_nonzero([face_encodings]))
    print("Length of FaceEncoding: ", len(face_encodings))

else:
    print("Shape of SampleArray: ", samplePictureEncodingArray.shape)
    print("Length of FaceEncoding: ", len(face_encodings))
    matches = face_recognition.compare_faces(samplePictureEncodingArray, face_en
    print(i, "Matches: ", matches)

    # If there is a new match, write new name to array
    if True in matches:
        match_index = matches.index(True)
        print("Match Index: ", match_index)
        name = samplePictureNameArray[match_index]
        if samplePictureNameArray[match_index] not in face_names:
            face_names.append(name)
            attendance.write(name)
            attendance.write('\n')
            print("Match Found")

            # If there is a new match, write new face to student's output folder
            for (top, right, bottom, left) in face_locations[i]:
                cropped = frame[top:bottom, left:right]
                cv2.imwrite(r'C:\Users\WillM\Desktop\Facial Recognition\Output'
                print("Added new picture to student's folder. ")
```

```
    print(i, "names", face_names)

if i == frameLength-1:
    break

process_this_frame = not process_this_frame
i = i + 1
# if(i%10 == 0):
#     process_this_frame = True
# else:
#     process_this_frame = False

for (top, right, bottom, left), name in zip(face_locations[i], face_names[-1:]):

    # Draw a box around the face
    cv2.rectangle(small_frame, (left, top), (right, bottom), (0, 0, 255), 2)

    # Draw a label with a name below the face
    cv2.rectangle(small_frame, (left, bottom - 35), (right, bottom), (0, 0, 255), 2)
    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(small_frame, name, (left + 6, bottom - 6), font, .5, (255, 255, 255)

# Display the resulting image
cv2.imshow('Video', small_frame)

# Hit 'q' on the keyboard to quit!
```

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

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
attendance.close()
print("Ending Search: ")
print("Students in Attendance: ", face_names)
```

Acknowledgments

I would first like to thank my thesis advisor Dr. David Umphress of the CSSE Department at Auburn University. He has been an encouragement and a guide to me throughout my time at Auburn. I have been provided many different opportunities while at Auburn, largely due to his direction and guidance. He has been both patient and kind to me while he endured my barrage of questions and concerns over the last few semesters. For that, I am extremely grateful.

I would also like to thank Dr. James Cross and Dr. Dean Hendrix from the CSSE Department at Auburn University for agreeing to be part of my committee. I appreciate them sacrificing their time and aiding me throughout my studies at Auburn. Without their participation, I would be unable to accomplish my goals.

I would also like to acknowledge Austin Wofford, Justin Belcher, and Anthony Crumley for their patience and helpfulness. Even though I bombarded them with questions constantly, they continued to support me throughout my studies.

Finally, I am profoundly grateful to my parents, brothers, grandparents, and other family members for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

William Miller

wem0019@auburn.edu

Education

- Pursuing M.S. in Computer Science, Auburn University, 2018
- B.S. Computer Information Systems, Shorter University, 2016.

Employment

- City of Rome, IT Intern, 2016
- Summer Classics, IT Technician, 2016.

Publications

- Developing *Utilizing Facial Recognition Software to Record Classroom Attendance*, Auburn University, 2018
- *Americas Mythology*, Shorter University, 2016

Awards

- CyberCorps Scholarship for Service Recipient, 2017
- Honors Academy, Sigma Beta Delta, Shorter University, 2016
- Robert H. Ledbetter Sophomore Achievement Award, 2015
- Gulf South Conference Student-Athlete Honor Roll, 2014 & 2015
- Alfred Shorter Scholarship Recipient, 2013