Final Project: Group 1

Steven Garrido, Zhaofang Qian, Xutong Qiu, Arya Saravanan

Table of Contents

Signature: Steven Garrido

Contributions of Steven Garrido to the final project included development and creation of the weak classifiers, and their related files, as well as creation and authorization of the final project report.

In addition, Steven Garrido adheres to the University of Maryland Honor Pledge, and vows that he has not given nor received any unauthorized assistance on this project.

Steven Garrido

Signature: Zhaofang Qian

Contributions of Zhaofang Qian to the final project included development and creation of the strong classifier class, training and classifying algorithm, as well as reading config actors and functions of calculating matrices.

In addition, Zhaofang Qian adheres to the University of Maryland Honor Pledge, and vows that he has not given nor received any unauthorized assistance on this project.

Zhaofang Qian

Signature: Xutong Qiu

Contributions of Xutong Qiu to the final project included preparing dataset, implementing the graph, classifier actors, multiple utility functions and actors, and feature detection algorithm. He also worked on project management, including git repository maintenance and local and cross compilation.

In addition, Xutong Qiu adheres to the University of Maryland Honor Pledge, and vows that he has not given nor received any unauthorized assistance on this project.

Xutong Qiu

Signature: Arya Saravanan

Contributions of Arya Saravanan to the final project included advancement on the training algorithm at the weak classifier level alongside the implementation and analysis of various metrics to analyze (both mathematically and graphically) the output of the data retained.

In addition, Arya Saravanan adheres to the University of Maryland Honor Pledge, and vows that he has not given nor received any unauthorized assistance on this project.

Executive Summary

The main goal of this project was to develop an Embedded Face Detection System that used the Viola-Jones algorithm to distinguish images that contained human faces from those that did not. The design of this project consisted of a cascading classifier structure to identify the presence of a face within an image and a training algorithm to strengthen this system and make it more accurate. In order to best tackle the requirements and limitations that said project demanded, it was split into several tasks and milestones which were each separated between the team members. While collaboration was commonplace in pursuit of its completion, the primary focus per individual on the team was the completion of their portion. Upon accomplishing that objective, each team member then worked to integrate their part into the overall construction of the EDFS. Upon completion, several metrics were implemented and a detailed analysis was done in order to document and understand the performance of this implementation and how it reflected upon the unique decisions that were made by the team versus that of which were made in the optimal implementation by Viola Jones. This analysis was then documented and the findings that were observed have been carefully documented within this report. Which drives the conclusion that we were successful in our initial objective to develop an Embedded Face Detection System.

Main Body

Goals and Design Overview

The main goal of this project was to successfully implement an embedded face detection system (EFDS) utilizing the Viola-Jones algorithm that was taught in class. As a group, we strived to use the Viola-Jones algorithm to create the EFDS, have it run on a Raspberry Pi, and ensure that it would be able to view an image and accurately detect whether or not the provided image contained a human face. The overall main design philosophy encompassed the two major parts of the project. The first part was the actual EFDS itself, which handled the facial recognition algorithm and was run on the Raspberry Pi. The second part to this project was the training module that was used to train the algorithm to better recognize facial features, and hence become more accurate at identifying which images were faces and which were not, adjusting the parameters the EFDS used so as to be as optimal as possible.

In a more in-depth analysis of our EFDS, the overall structure consisted of cascading classifiers that would feed into each other to create a sort of chain from one to the next. Figure 1 provides a visual example of this structure. As Figure 1 illustrates, each weak classifier is assigned to a certain facial feature (mouth, nose, eyes, etc.), and uses the Viola-Jones algorithm in order to determine if the specified feature is in fact present in the image. If the classifier determines that its desired feature is present in the image, the classifier passes the image to the next classifier to detect another feature, if the feature is not present, the classifier immediately aborts from the chain and it is presumed that the image does not contain a face. If every classifier in the chain asserts that their desired feature is present, then it is determined that the image does contain a face.

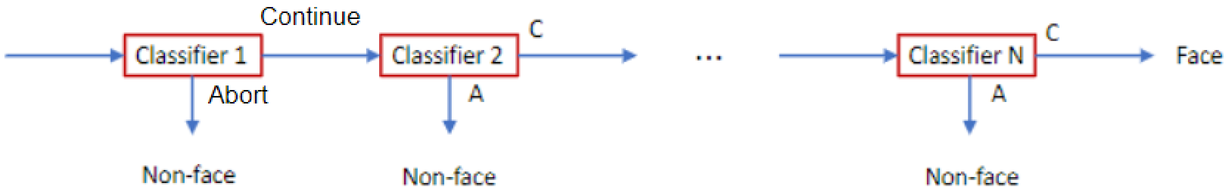


Figure 1: A visual representation of the cascading classifier structure used in the EFDS. (Source: Prof. Shuvra S. Bhattacharyya, "ENEE 408M: Project Specification")

Looking closer at the individual classifiers, each one utilized a simple five-mode system internally, illustrated in Figure 2, in order to carry out their required functionality. Each classifier was designed with two input ports and two output ports ("Continue" and "Abort"). As previously mentioned, each classifier also had five modes; "Configure", "Read", "Classify", "True", and "False". Each classifier would begin in the Configure mode, in which it would load all the parameters onto the actor state. Next it would proceed to the Read mode, in which the classifier would consume the image token in order to read it, as long as the token wasn't null (in which case it'd go directly to the false mode) the classifier would then change to the Classify mode. In the Classify mode, the classifier would use the actual Viola-Jones algorithm on the image referenced by the token, and determine if the specified facial feature was present. From here, if the algorithm determined that the facial feature was present, the classifier would go into the True mode, which would simply produce the token on the Continue output to either be sent to the next classifier, confirming the presence of a face if it were the final classifier. Otherwise, if the algorithm failed to detect the feature, the classifier would proceed to the False mode, in which the token would be produced on the Abort output, and from there the process would immediately stop and the program would determine that a face was not present.

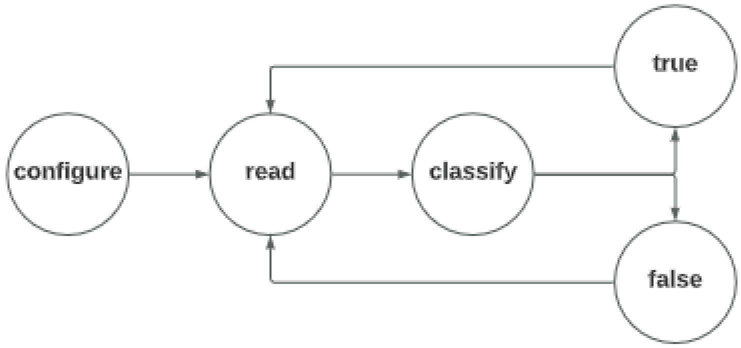


Figure 2: An illustration of the internal modes within a classifier and the possible transitions each mode can make. (Source: Prof. Shuvra S. Bhattacharyya, "ENEE 408M: Project Specification")

Zooming out, each chain of cascading weak classifiers all together form a strong classifier, which is a product of all the weak classifiers searching for their own individual features. The overall goal of the strong classifiers is to determine, based on the results of the weak classifiers, if there is a face present in an image. Each strong classifier, internally, has a set, and unique, amount of weak classifiers chained together. The strong classifiers are also chained together to make cascading strong classifiers, each of which feed into the next whether or not the result of their cascading weak classifiers yielded a face or not. If all of them agree that a face was detected, then the final result is the confirmation of a face present in the image.

Going further beyond the EFDS itself was the training program. This was an essential component within the project as it allowed the EFDS to detect more accurately, and more precisely, what images contained a human face and which did not. The training program allowed the EFDS to change its parameters within the classifiers in order to get a better understanding on which configuration of parameters provided the most accurate results, and thus were the best combination of parameters to use in the EFDS.

Realistic Constraints

(Programming interfaces constraints)The first major realistic constraint that we faced was the specific application programming interfaces, namely the enable function and invoke function. In regards to the invoke function, in our original design, we expected the scheduler to control the behavior of the classifiers at runtime, since the classifiers need to behave differently based on their position and classification result. For example, when the classification result is true, a classifier enters TRUE mode, but only the last classifier needs to write the result into the output file. Since all the classifiers are instances of the same class, this cannot be done by modifying the classifier class. Therefore, we expected the scheduler to control the behavior of the classifiers at runtime. However, the invoke function in our project overwrites the Welter's default invoke function, which doesn't accept any parameter. This means the scheduler cannot control the classifiers' behavior when calling their invoke function.

The solution we came up with was to connect each classifier actor with a file sink actor that was invoked only when the classifier was the last in the cascade and the result is true. This is done by using the if else statement where the classification result is the condition. When a classification result was false, a classifier entered FALSE mode where it wrote false to the output file and aborted, regardless of its position.

(Platform constraints) The nature of the project requires the team to code and test in the welter environment. However, the welter environment only exists within the enee408m directory, this constrained the project as it made it so that the project could only be accessed and worked on while connected to the University of Maryland wifi network or the University of Maryland vpn, not only did this make the project less accessible by constraining where the team had to be physically located so that they could to work on the project, but it also made the project heavily dependent on the teams access to a stable internet connection, since the directory could not be accessed offline.

Additionally, the use of git imposed constraints on the team's progress as well, since the team had not had much experience familiarizing with git prior to the project, the team would often find themselves overwriting each other's progress while trying to add to the project with their updated work.

(Testing constraints) The testing constraints come from the computation speed of the ENEE408M system. Since the project has to run and be trained under it, the speed of it determines how fast the team can train its classifiers. For example, choosing 4 best features from 1103 features for a strong classifier to use takes around 1 hours. When the team debug or change the parameter of the training scheme, the train process has to be rerun.

(Programming language and communication constraints) Unlike Python, which is a script language, the syntax of C++ is more complicated. Moreover, in terms of data type, Python can fit different data types into the same array, which makes it easier for communication between classes. This feature in Python can make the graph run more efficiently. For example, in C++, the graph has to fire twice in order to pass two different types of array pointers to the next actor. If implemented in Python, it only needs to fire once, making the classification to run faster.

(Language implementation constraints) Various implementations of Viola Jones were explored in subsets of C++, Java, Python, as well as MATLAB in order to understand the benefits versus drawbacks of employing each language. It was quickly determined that the selected utilization of C++ for this implementation was a choice that reflected in customizability rather than simplicity. MATLAB specifically, for example, had several functions which could simply perform many of the tasks that this Viola Jones implementation desired in a much more streamlined and simplistic manner than the implementation that is done in C++. OpenCV as a library partially accomplishes this, but not to the same extent as MATLAB's functional capabilities (also recognizing the fact that a restriction was in place against the use of OpenCV or its associated libraries).

Engineering Standards

When developing this project, the team remained constantly aware of how the design choices made could drastically affect the development of the embedded face detection software. For this reason, every choice was made with great consideration for how it would impact the outcome of the project, and the design philosophy going forward. Each decision made was a conscious effort made to preserve a high engineering standard and to use every tool as effectively as possible by utilizing their greatest strengths.

One standard incorporated into the project was the use of the C++ and bash programming languages. We took advantage of their features to make our project progress efficiently. For example, in terms of dataset, our team decided to precompute all the integral images and divided them into different sets. This process requires image to text conversion for more than 6000+ images and involves a lot of shell commands. Therefore, Bash scripts become an efficient tool to accomplish the goal. The team implemented six bash scripts and two C++ programs that computed the integral image and utilized the embedded enee408mimg2txt macro. Bash scripts are also used in testing. With choosing a different folder, the team could run training or testing functions under different directories with different datasets easily. Bash scripts let the ream using Linux commands such as “ls | wc -l” to return the total number of files in a folder. This helps calculate the time needed to run the classify function.

Alternative Designs and Design Choices

The total number of features we used is 161,000 different features. With such a mass number of features, each iteration of training 2 features for a strong classifier takes around 1 to 2 hours. Thus, we decided to implement two strong classifiers with 4 weak classifiers in the first one and 15 weak classifiers in the second. We believe that a better result could be obtained as long as spending more time on the training process. However, because of the constraints of connecting to the VPN network, many training processes end up with a loss of signal. We would discuss the Pareto points from the data we luckily gathered.

When using only the first strong classifier, it takes 9.45 seconds to classify 1443 pictures. The average latency is 6.55 ms per picture. The Truth Positive rate is 99.28% for a total of 555 inputs and the False Positive rate is 39.30% for a total of 888 inputs. This gives us a precision as 71.64% and recall as 62.06%. The $F_{measure}$ is calculated as 0.665.

When using the combination of two strong classifiers, it takes 10.83 seconds to classify 1443 pictures. This is expected because a more complex system needs more time to classify the same inputs. The average latency is 7.51ms per picture. The Truth Positive rate is 93.68% for a 554 inputs that are classified as Face by the first strong classifier. The False Positive rate is 62.56% for a 358 inputs that are classified as Face by the first strong classifier. With a precision as 59.96% and recall as 71.44%, the $F_{measure}$ is 0.652. The total Truth Positive rate would be 93%, and the False Positive rate is 24.58%.

Besides our final strong classifier configuration, there is another strong classifier with Truth Positive rate as 73.35% and False Positive rate as 10.16%. The average latency is 4.46ms. This is a Pareto Points compared with the first strong classifier. However, we didn't choose this as the first one because of the low Truth Positive rate.

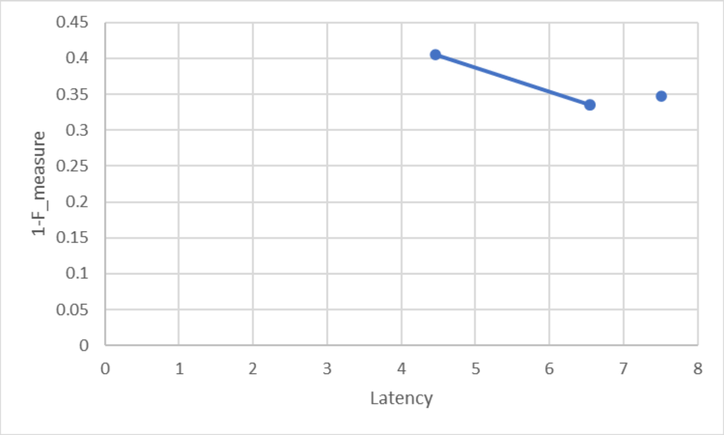


Figure 3: Pareto Plot of three strong classifiers

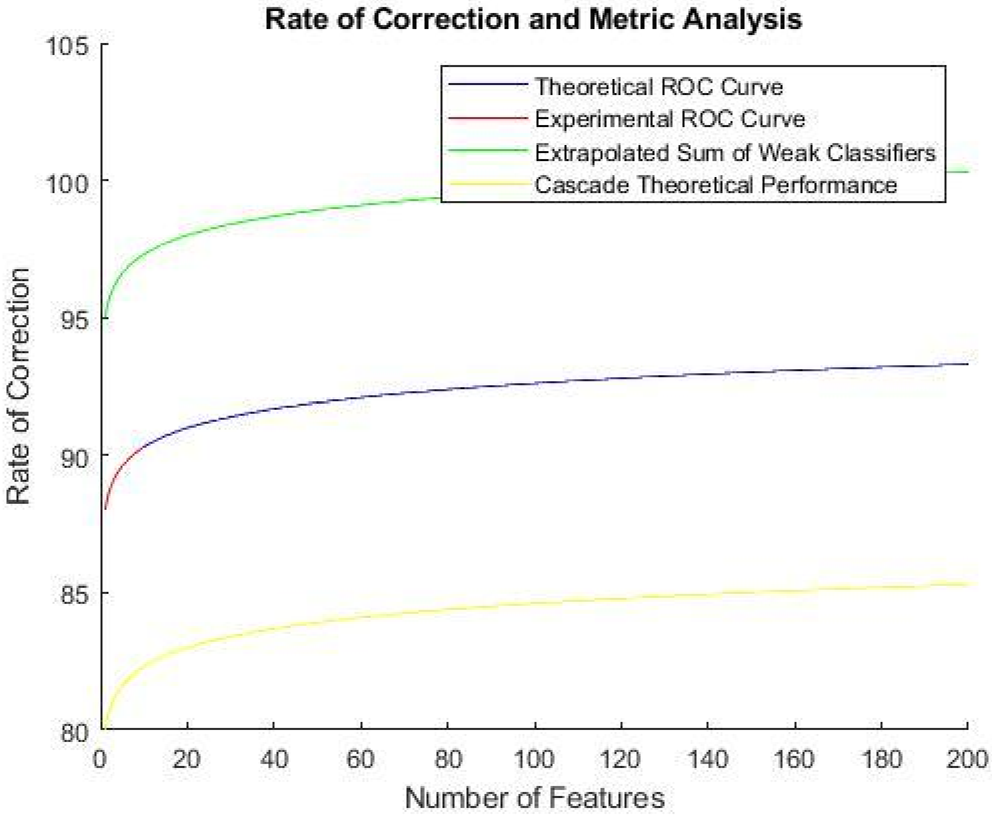


Figure 4: The full theoretical and experimental curve with respect to metric analysis

A major portion of the project which was highlighted is the use of different metrics in order to derive conclusions with respect to our design choices. These metrics became core in analyzing performance. Namely, there are two major metrics that were utilized on top of correction rate. The first was the extrapolated sum of weak classifiers. This meant that taking any feature set and if an image is determined by at least one to be a face, then a positive is outputted.The second was the cascaded performance where any the average true positives and false negatives were averaged to determine the overall successful nature of the program. While we intended to perform a very high-level, in depth analysis for up to 200 features per. Certain aspects of them quickly became limited by the nature of our EDFS and its limited processing power in computing strong classifier cascades composed of a large number of features (said number was indeterminate). Figure 3 above demonstrates this. The blue line exhibits what the ideal curve would be in the instance that we were able to test our program with a cascade size of 200 features. However, it is purely a regression fit of the data that was acquired. The data which we acquired with high confidence is displayed by the red line. From there, the expected performance based on the sum of the individual performance of features is demonstrated by the green line, which is clearly inflated due to the inaccurate nature of such a methodology followed by the unreliable nature of it. Finally, the yellow line demonstrates the performance of a cascade of those features that is theoretically computed. Interestingly enough, the actual performance of the cascade seemed to be the difference between the ideal feature-sum performance and that of the feature-cascade performance. That of which was a very interesting discovery.

Technical Analysis for Systems and Subsystems

The bottom structure of our project is the utility functions, which provides image read and feature detection functionalities. Above that lies the weak classifier class, which stores one feature and uses the feature detection function in the utility to classify an image. Going up by another level is the strong classifier class, which contains (how many) weak classifiers. The strong classifier classifies an image based on the results from its weak classifiers. Above the strong classifier is the classifier actor, which has one strong classifier instance stored and has two input edges to read in the image, config data, and two output edges to pass the image to the next classifier actor and classification result to a file sink actor. When the actor enters the config mode, it reads two tokens, which are loaded with the configuration data from its read_config edge. The top level is the graph that connects the two classifier actors and other actors together.

It can be seen that the team broke the project up into as many independent components as possible to allow members to work in parallel. The team progressed quickly when implementing code but then encountered coordination problems in the testing phase. For example, the team members working on the strong classifier class may not be able to test their code until both the weak classifier class and feature training are completed. In order to solve the issue, the team decided to let each member prioritize implementing the core functions in their classes. In terms of feature training, the team handcrafted a few features to help progress strong classifier testing.

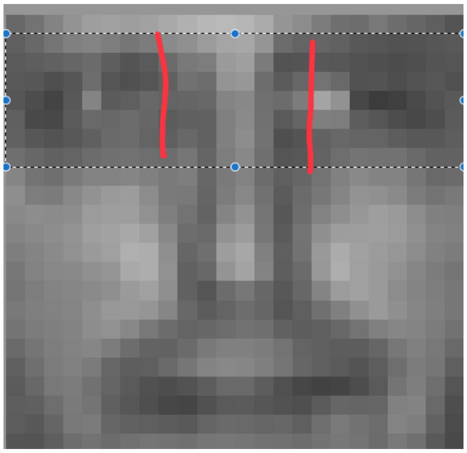


Figure 5: A handcrafted feature example of eyes

Design Validation for Systems and Subsystems

The main three matrices we used are latency, False Positive (FP) rate, and Truth Positive (TP) rate. The latency is calculated by recording the start time and end time of running the classify function. The total number of images, N_{pos} , is printed on STDOUT. The total execution time, T_{pos}, is returned by Bash scripts. Then, we use $\frac{Tpos}{Npos}$ to calculate the time needed to execute one image. Because all these tests are run on the server, the latency of execution also depends on the network speed and is not stable and accurate if we record the start and end time of classifying each image. By calculating the average latency would give us a more stable estimate.

False positive rate is calculated by using the total number of images that are not faces but classified as faces divided by the total number of inputs. $R_{FP} = \frac{\rho_{FP}(Z)}{\sigma_n(Z)}$ is the equation we used to calculate the False Positive rate, and the Truth Positive rate shares a similar equation, $R_{TP} = \frac{\rho_{TP}(Z)}{\sigma_p(Z)}$. We care about the Truth Positive rate because it shows how accurate our classify algorithm is. The ideal TP rate should approach 99%. False Positive rate would give the rate our classify algorithm failed to classify the image. Besides having a large TP rate, we want our FP rate to be as low as possible. An ideal FP rates that could be used commercially is around $10^{-5} \sim 10^{-6}$.

Test Plan

As we are working in parallel, the Test suite is important and helpful to guarantee our code could function properly. For the Integral Image, we used some simple 3 x 3 pixel images to calculate the sum of the pixel value and they could be converted to .txt format. For weak classifiers, it could be tested by feeding in validation images to verify that the classification would reach an acceptable threshold. We heavily rely on our test suite when implementing the strong classifier. We assume some sample weak classifier output when the strong classifier needs the output of the weak classifier and we also print out variables and exit promptly to save calculation time and verify one step in the strong classifier.

After the process of implementation, we designed test suites with simple drivers to make sure different parts of the EFDS's graph could function. With Bash script, we run our code on validation sets and record the Truth Positive rate, False Positive rate, and latency.

We were also planning to properly shift our train/validation set and train/test our strong classifier. Dividing the train and validation set into 5 sections and setting each section as a validation set differently and finding the best strong classifier that would give the most accurate result while having an average execution time. However, due to time constraints, we may not be able to implement a functional test suite to achieve this goal.

Project Planning and Management

The project was broken up into multiple components so that each member could be assigned an aspect of the project of which they could be responsible for completing. The project was broken into two major components; the embedded face detection system, and the training and testing of the system. Due to the nature of this project, in the early stages of the project, all focus was put on developing the embedded face detection system, and very little thought had yet gone towards the training system, as there was nothing to train at the time. As the embedded face detection system came closer to completion, individual tasks became more focused on the training algorithm and the project report.

From the start, each member of the project was assigned to work on a specific feature of the algorithm, though they were also responsible for helping each other should a member become stuck. As the project progressed, and aspects were completed, members moved on to the next task that they were made responsible for, once it was possible to begin working on their designated aspect of the project.

Steven Garrido was designated to be responsible for developing the functionality for the weak classifiers to use. He was made responsible for all files in regards to the functionality of the weak classifiers, all of which he wrote in C++. After completing the weak classifier functionality and the related files, Steven was then made responsible for creating and writing a majority of the project report, which he was able to complete alongside his teammates as they handled their own responsibilities, and reported their findings to him.

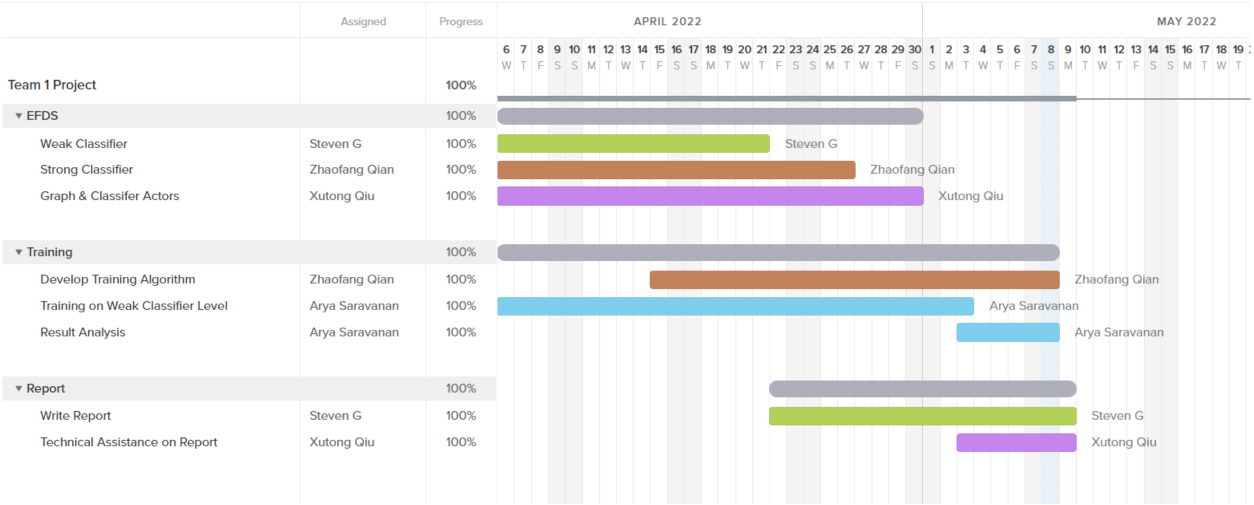
Zhaofang Qian was put in charge of writing the strong classifier class and its related files, all of which were written in C++. He was also made responsible for reading config actors and functions related to calculating matrices. After completing the tasks, Zhaofang was responsible for writing the training and classifying algorithms that were used in the classifiers to identify faces in the images, which were once again all written in C++.

Xutong Qiu's responsibilities consisted of preparing the dataset, implementing the graph, implementing the classifier actors, implementing the utility functions and actors, and implementing the feature detection algorithm. Xutong was able to complete these tasks effectively and efficiently, all in C++ and Bash scripts. On Top of these responsibilities, Xutong also acted as the project manager as he became the main project organizer and successfully managed each person. He also maintained the git repository and handled the local, and cross compilation.

Arya Saravanan was tasked with writing and equipping the training functionality at the weak classifier level and developing various metrics/methods of analysis that would be used in order to measure the project's success. Both the weak classifier training alongside the metrics that were measured were obtained through a C++ implementation. From there, a MATLAB analysis was done in pursuit of graphical representations and mathematical synthesis of the data that was obtained upon running the program through multiple iterations.

Every team member was also held responsible for assisting any team member that was stuck and required assistance. It was for the betterment of the team, and project as a whole, that the team carried out this work ethic so that each member, and the whole team, could thrive.

Although a decisive and in-depth plan was made by the group early on in the project's lifespan, which can be seen in the team's Gantt chart in figure 5, it was unanimously agreed that the overall application of the plan would be much less rigid, and would be very fluid based on the needs of the entire group. On multiple occasions members would come to the side of another member that required assistance.



	Assigned	Progress
Team 1 Project		100%
▾ EFDS		100%
Weak Classifier	Steven G	100%
Strong Classifier	Zhaofang Qian	100%
Graph & Classifer Actors	Xutong Qiu	100%
▾ Training		100%
Develop Training Algorithm	Zhaofang Qian	100%
Training on Weak Classifier Level	Arya Saravanan	100%
Result Analysis	Arya Saravanan	100%
▾ Report		100%
Write Report	Steven G	100%
Technical Assistance on Report	Xutong Qiu	100%

Figure 6: A Gantt chart created by the group to show progress on the project, and individual member tasks.

Though the team did not follow the Gantt chart strictly, and used it more as a guide rather than a set of instructions, the team still felt that the fluid work ethic philosophy that was adopted improved the overall efficiency of the team, and quality of the final product. The group only vaguely followed the planned schedule, which is also what they planned to do from the beginning, as they held the belief that sticking strictly to a plan that was created too far in advance of the end goal would lead to complications.

Conclusions

In conclusion, the project was designed to be broken down into as many parts and features as possible to allow the team to work independently of each other, so that they may work in parallel in order to meet the desired deadlines. The entire project was strategically divided into two separate parts (which were then broken down further): the embedded face detection system, and the training algorithm. The feature detection was broken down into its many components (such as weak classifiers, strong classifiers, and the Viola-Jones algorithm) allowing each team member to contribute. After the EFDS was completed, the team moved on to working on the training algorithm and the report. The training algorithm was worked on by Zhaofang and Arya, while Xutong made final adjustments on the actors and graphs and Steven began work on the report.

Though the team found that working in this manner held many benefits, it also brought its own challenges. The greatest challenge to overcome was coordination. Due to team members having the freedom to work independently, the work each team member produced would often not completely line up with the work of another team member whose product they needed to work in conjunction with. This would often lead to team members working on certain aspects for longer than anticipated as additional time had to be spent correcting the inconsistencies. Another challenge that came up was team members being unable to progress in their work, because they required another team member’s work to be completed first. Due to this project’s many connected parts, many aspects of it are dependent on each other, meaning that certain parts of the project could not be started until another aspect was completed, leading to time being wasted as no progress could be made. Despite this challenge, the team still found a way to be effective during the few instances where this occurred by actively helping the team member whose work they depended on in order to continue with their own work.

During the project implementation, we found some similarities between neural networks and the Viola-Jones algorithm. For example, neural networks have multiple layers which consist of multiple nodes with different weights. This resembles the Viola-Jones algorithm, where each strong classifier is similar to a layer and each weak classifier is like a node. Each strong classifier also applies different weights to the weak classifiers stored. But unlike layers in neural networks,whose output depends on the previous layer, strong classifiers in the Viola-Jones algorithm work relatively independently with each other. They don't take any value from and don't know the result of the previous classifier.

One of the major lessons learned was that though working in parallel has the benefits of allowing teammates to work independently and at their own pace, it also brings the downsides of less cooperation and bottlenecks in work efficiency due to internal-project dependencies.

Another lesson learned is that though it may be easier to comprehend working from the ground up, oftentimes it is more important to work on key features of the project first, such as the training algorithm, in order to ensure that as much time as needed is dedicated to important features so that they can be implemented fully and correctly. This lesson was made obvious to the team as they approached the final deadline as many of the key features they had produced were done later in the project's lifespan, meaning that they were not as completed and functional as the team had originally envisioned they would be. Overall this led to a lack of quality in features that were more integral to the project, rather than having flaws that had a lesser impact on the project (had lower-priority components of the project been worked on after key features).

References

1. *Prof. Shuvra S. Bhattacharyya, "ENEE 408M: Project Specification"*
2. *Paul Viola "Rapid Object Detection using a Boosted Cascade of Simple Features"*