Intelligent Attendance and Participation Monitoring

Neeraj Godbole, **Electrical and Computer Engineering, Carnegie Mellon University**
Kevan Dodhia, **Electrical and Computer Engineering, Carnegie Mellon University**
Omar Delen, **Electrical and Computer Engineering, Carnegie Mellon University**

Abstract— **We implemented a system to track attendance and participation in a class room setting, providing instructors with useful metrics both live during class and afterwards. We develop an easy script to gather student training data and then detect when they are present on a second-by-second basis during class. Participation is measured by detecting when and how long a hand are raised.**

Index Terms— **eigenfaces, PCA, LDA, facial recognition, facial detection, viola-jones, computer vision, machine learning**

I. Introduction

For our project, we are building an intelligent attendance and participation system. The system is designed for a classroom environment where professors my want to keep track of student attendance and participation without having to do this manually. The system is computer vision and machine learning based and spans the Software System and Signals & Systems ECE areas.

In order to track attendance and participation we set up a camera in a classroom and feed this footage into a laptop that will run all the software. To keep track of attendance, we detect when students enter and leave a class room. We use facial recognition to identify these students. In order to keep track of attendance, we detect when students raise their hands and how long they raise their hands for.

II. Design Specification

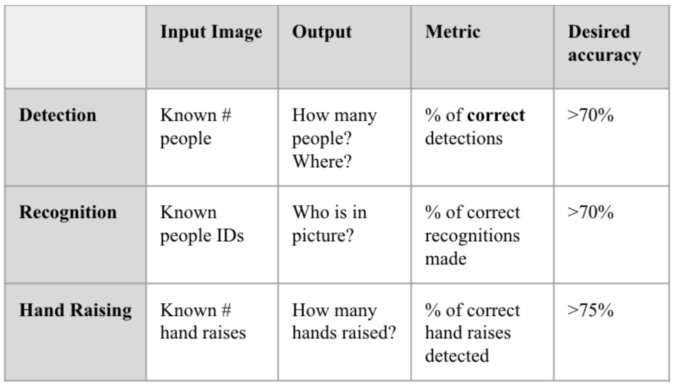
The application demands the following: a system that is able to track and record student attendance and participation in a classroom setting.

A camera-based system naturally arises where the camera records the class and provides video input to a laptop (or other PC) that performs all of the monitoring through an application/program.

The application running on the laptop can then address the problem by performing the following three functions:

- Detecting faces and performing facial recognition on identified faces, matching them against the database of members of the class.
- As part of recognition, also be able to detect strangers, i.e. faces detected that are not in the database
- Upon detecting faces, detect and identify raised hands. This is a means of tracking participation during a class.
- Consolidating this information showing attendance and displays participation data for each member of the class.

A table with our desired results for accuracy is below.



	Input Image	Output	Metric	Desired accuracy
Detection	Known # people	How many people? Where?	% of **correct** detections	>70%
Recognition	Known people IDs	Who is in picture?	% of correct recognitions made	>70%
Hand Raising	Known # hand raises	How many hands raised?	% of correct hand raises detected	>75%

Fig 1 – Table showing our design specification metrics

The metrics described in the table above were obtained considering that we ideally want a very high accuracy, but we also had to keep in mind that we were building our components from scratch and thus we dialed back the accuracy numbers so that the metrics were achievable.

III. Architecture/ Overall Design

In this section we provide an overview of the IPAM system along with the block diagram.

From the block diagram above, the flow of information is fairly clear.

Video capture is input to the Face Detection module that processes the video in real time computing bounding boxes for the faces in the camera's field of view.

The bounding box data is passed as input to the Facial Recognition module, which uses PCA and LDA (mostly) to perform facial recognition on the detected faces, associated each detected face with the appropriate matching student in the database, and marking absence of a face (i.e. student not present). This module also handles stranger detection, recognizing that a person is not part of the class if the detected face is not matched with any in the database.

The results of the facial detection/recognition are then passed on to Raised-hand detection module, which first seeks to learn the pigments of each of the students faces. This is done by utilizing the fact that our recognition and detection gives us the faces and identifications of students for free. After this, we extract pigments from these faces and scan the portion of the image around each face for similar pigments.

These results, along with the facial detection/recognition results will be sent to a basic frontend application that will display the updated attendance and participation records. The front-end/visualization part of the system provides the user with a live video capture from the webcam, overlaid with the face detection bounding boxes. Along with this live capture, there is a live plot showing the projection outputs of PCA + LDA along with the decision boundaries given to us by SVM.

A key thing to note here is that Detection, PCA, LDA and Hand Raise Detection are all built from scratch with the major algorithms being written without using libraries or APIs. We do use OpenCV to capture video data and read images and it is important to note that the SVM component is performed using scikit-learn.

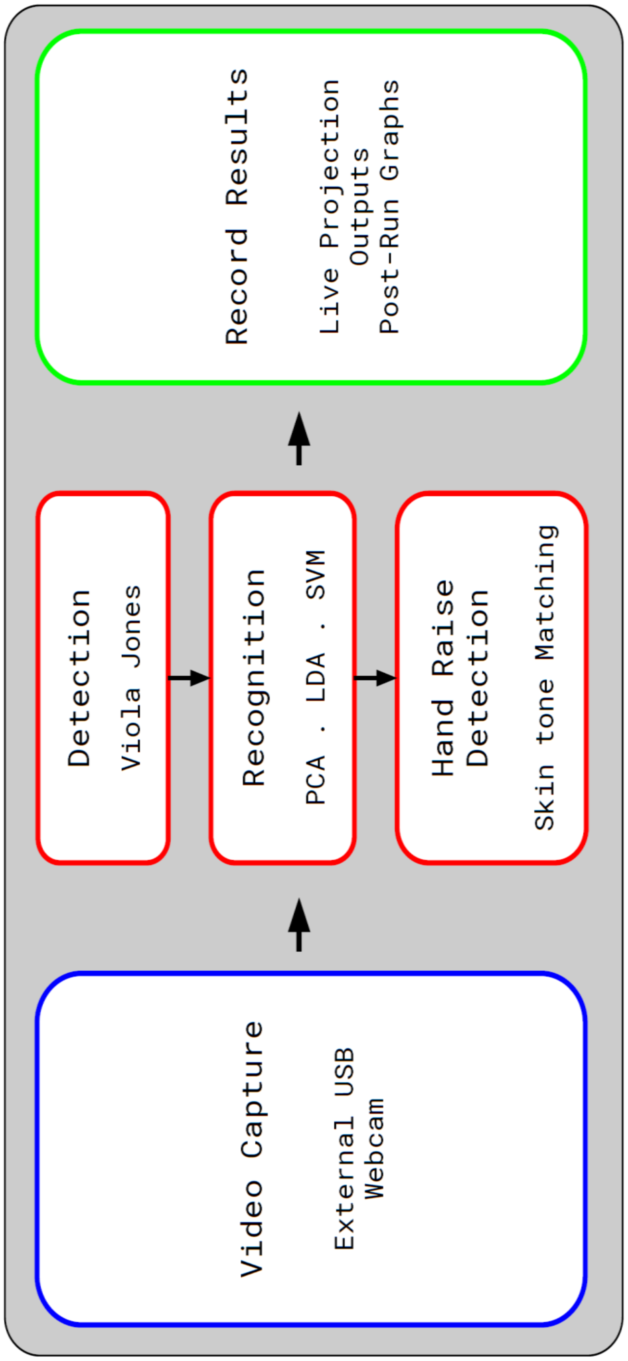


Fig. 2 – Overall system block diagram

IV. System Description and Implementation Detail

In this section we describe our system in depth, covering the final implementation detail and algorithms used in each of the modules in our complete solution, as well as discussing the experimentation and alternatives considered that allowed us to arrive at our proposed implementation plan for each module.

A. *Face Detection Module*

We implemented the well-known Viola-Jones algorithm in order to detect faces. The algorithm can be summarized into the following components:

1. Haar Features construction, which are used to match regularities in human faces.
2. Integral image, which allows the features used by the detector be computed very quickly.
3. AdaBoost training, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers.
4. Cascading Classifiers, which results in a classifier that consist of several simpler classifiers that are applied to a region of interest until at some stage the candidate is rejected, or all the stages are passed.

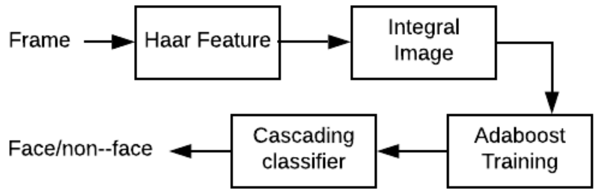


Fig. 3 – Block diagram outlining the Viola-Jones Facial Detection Algorithm

The algorithm for AdaBoost training, which selects the T best features from 160,000 potential features is as follows:

1. Initialize weights for m negative and l positive examples:
$$w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$$
2. For $t = 1, .., T$:
   a. Normalize the weights.
   b. For each feature, train to find optimal threshold and polarity.
   c. Choose a feature with the lowest error, $\in_t$.
   d. Update weights for all positive and negative examples: $w_{t+1,i} = w_{t,i}\left(\frac{\in_t}{1-\in_t}\right)$ if example was classified correctly, else $w_{t+1,i} = w_{t,i}$.

We implemented and trained our AdaBoost classifier on AWS, using 8000 face and 10,000 non-faces images. With just 10 features we got test accuracies of ~81%. With 75 features, we get 95% accuracy.

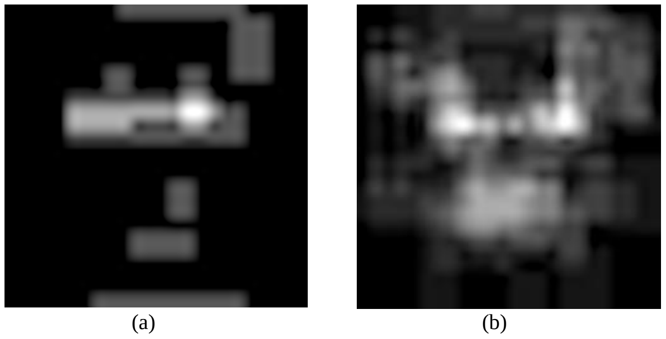


Fig. 4 – Reconstruction with (a) 10 features and (b) 75 features

In order to detect all faces in a frame, the algorithm would need to use 75 features in every 24x24 sub-window in the target frame. However, this makes detection very slow (~10s per frame). In order to increase speeds, we used a cascaded classifier with layers containing an increasing number of features in each layer. If in any layer the classifier does not think the sub-window contains a face, the cascade stops. Only sub-windows with a high probability of being a face will make it through all the layers. In their paper, Viola and Jones introduce an algorithm that selects the optimal number of features in each layer in order to minimize the false positive rate per layer. However, for the sake of simplicity we manually chose the design on each layer. We chose layers of size 1, 5, 10, 25, 50 and 75.

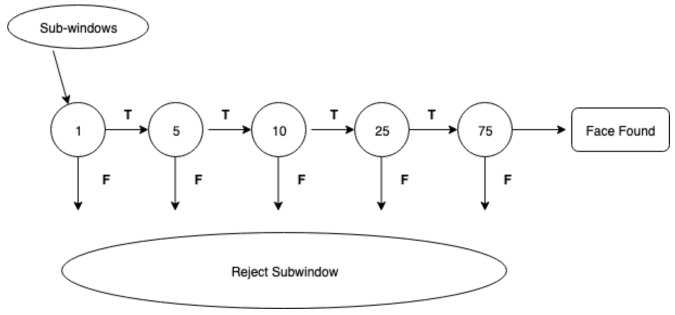


Fig. 5 – Diagrammatic depiction of detection cascade used.

After training a cascaded classifier, we maintained accuracies of 95% and managed to achieve a 10x speedup for detection for a single frame. With cascading we managed to reduce the detection time to 1s per frame.

Upon running this algorithm, we get bounding boxes around all faces in a frame. However, the algorithm will always find multiple bounding boxes surrounding every face in the frame. While all the bounding boxes may be valid, it is important to merge all the overlapping bounding boxes so that the detector doesn't report back that multiple faces have been found when there is only one. We used Non-maximum suppression in order to merge the images.

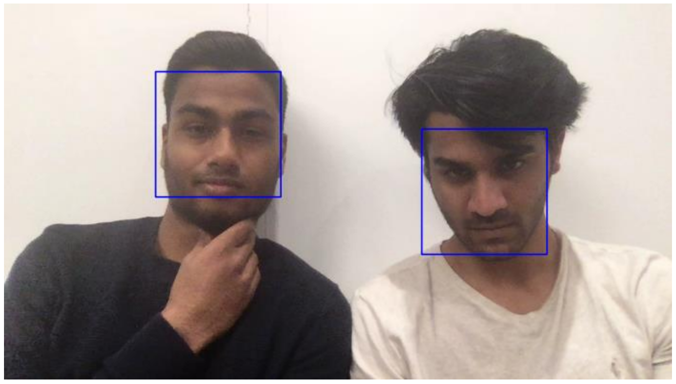


Fig. 6 – Example of bounding boxes outputted from the Viola-Jones Facial Detection Algorithm

B. *Recognition*

Our method of recognition involves using PCA, LDA and SVM.

We begin with using PCA and eigenfaces for recognition. PCA allows you to take a higher dimensional space and project it into a smaller space. This is useful because images are $m \cdot n$ dimensionality (where *m* denotes width and *n* denotes height) and we cannot compare and cluster images for recognition with such high dimensionality. Thus, PCA develops a set of basis vectors, eigenvectors of a covariance matrix to be precise, which span the vector space of the face image. PCA is followed by LDA in order to better separate our classes so that classification produces better results, and also to reduce the dimensionality of our image. Finally, we apply SVM to the projected training samples to obtain our decision boundaries. Below we detail the entire recognition pipeline.

The first stage of the recognition pipeline, PCA, is implemented as follows:

1. Collect all N training images, each with dimensions (*k* x *k)* into one matrix **X** with dimensions (k^2 x *N*). Each column of **X** is an image vector, with the image reshaped into a 1-D column vector.
2. Compute the mean image vector **μ**, and subtract **μ** from each column of **X**.
3. Now we find the covariance of the mean subtracted **X**, denoting this as **Σ .**
4. We then optimize:
$$\max \boldsymbol{\omega}^T \boldsymbol{\Sigma} \boldsymbol{\omega} \;\; s.t \; |\boldsymbol{\omega}| = 1$$
5. This simplifies to an eigenvalue problem and we obtain a matrix of eigenvectors **V,** sorted by decreasing eigenvalue. These values are obtained using numpy's *linalg* module. We then discard the columns corresponding to eigenvalues we don't want.
6. We compute $\mathbf{P} = \mathbf{V}^{\mathbf{T}}(\mathbf{X} - \mathbf{q})$ which is our matrix of basis coefficients.
7. It is this matrix of basis coefficients **P** that is the input to LDA.
8. The PCA function outputs **P,** the mean vector **μ** and the eigenvectors **V**.

We next perform LDA with the following process:

1. LDA takes inputs **P** (final matrix of basis coefficients for each training image), and an **imgToClass** list which allows us to find which class each image (or column of **P**) belongs to.
2. Using the two inputs, compute the between-class scatter matrix $\mathbf{S_B}$ and the within-class scatter matrix $\mathbf{S_W}$.
3. We then optimize the following equation:
$$\max \frac{\boldsymbol{\omega}^T \boldsymbol{S_B} \boldsymbol{\omega}}{\boldsymbol{\omega}^T \boldsymbol{S_W} \boldsymbol{\omega}} \;\; s.t \; |\omega| = 1$$
This simplifies to an eigenvalue problem, and we obtain a matrix of eigenvalues **W**, which is used to project our input onto a vector space of a smaller dimensionality than the input and one in which the classes are more scattered.

Finally, we must now classify the faces in our training set and this is done using SVM. This is the one component of the project that was not implemented from scratch. We instead used *scikit-learn*. This is because the decision to use an SVM was made very late in the project's development. SVM aims to find decision boundaries for the classes such that the margins from the decision boundaries to the training samples nearest to the decision boundary, are maximized.

Therefore, when given a new test image, the recognition pipeline/procedure is as follows:

1. Reshape the image into a 1-D column vector and subtract **μ** (mean vector output from PCA) from it. We denote this using **J**.
2. We now want to project this image onto the vector space we obtained through PCA + LDA. To do this we first obtain basis coefficients for the image, $\mathbf{P_{im}} = \mathbf{V^T J}$.
3. Now we project the image by computing $\mathbf{W^T P_{im}}$**.**
4. We now use the decision boundaries obtained by SVM to classify this projected sample.

Below is a plot display our PCA eigenvalues for all of our eigenvectors. Note, it is a log plot on the y-axis. The first three eigenvalues were rather large, and we suspected they were representing meaningless brightness variations in the images.

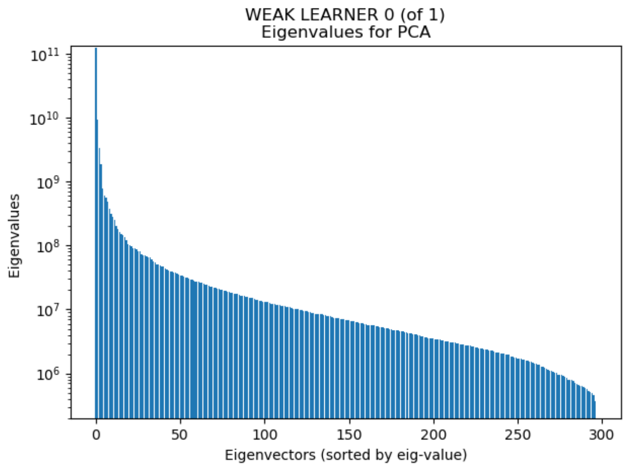


Fig. 7 – Eigenvalues obtained through PCA for training set

Initially, we did not achieve good performance with our PCA and LDA implementation. Our data showed poor separation of classes, and we had to rewrite our PCA code along with fine tuning parameters, such as how many eigenvectors to keep, how many training images to have, what lighting conditions we should take training images in etc.

After some experimentation, we observed that the following changes were necessary to improve PCA and LDA performance:

- Rather than drop the first three PCA eigenvectors we instead drop the first two.
- Instead of gathering only around 40 images per class, gather more than a hundred per class. Our demo training set consisted of 100 images per class.
- After discarding the first two PCA eigenvectors, use as many as possible. We keep 90 eigenvectors in our implementation.

After these changes we observed that the PCA + LDA pipeline did an excellent job of projecting the data into a vector space where the classes are nicely separated, i.e. the between class scatter was increased.

During our design stage, we had decided on using K-means clustering to perform final classification. Our K-means clustering was not performing as we liked, and our accuracies were hovering in the 50 – 60 percent range. It would often confused predictions for clusters that appeared elongated (when intra-class variance was not reduced enough). This is why we switched to an SVM for classification of points in this eigenspace. We experimented with both linear and non-linear SVM and we settled upon a linear SVM as our data-set was well-suited to being separated by straight lines. The figure below shows the outputs of one of our later training sets once projected using PCA and LDA, along with the decision boundaries found. As can be seen, there is excellent separation of the classes and the decision boundaries can be roughly deduced by eye.

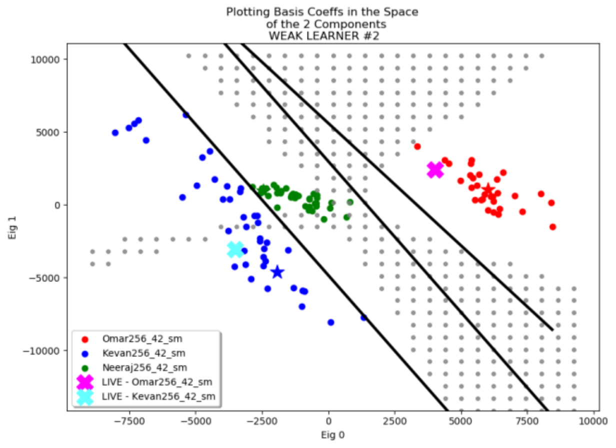


Fig 8 – Chart depicting the eigenspace with class data points plotted, as well as faces gathered live being plotted.

We also utilize a boosting technique. We make use of several classifiers/learners, each classifier training in a somewhat different manner. The logic for this is that if each classifier is better than random guessing and if each has insight that others do not have, then combining their results boosts overall performance. Each classifier not only looked at a randomized subset of a training image but the size of that subset was also somewhat randomized. Furthermore, each classifier utilized a different dimension (red, blue, green, or gray) for the images. We experimented with using Hue as well but that did not help. After each classifier runs, they all vote on the final prediction.

To ameliorate the results even further, we introduced a Random Sample Consensus, RANSAC, algorithm. For example, we spin up 100 classifiers, measure their test accuracy individually, then take the top 10% classifiers and use them to vote in the above-mentioned boosting algorithm. This was helpful because taking certain subsets of training data led to better results (removed certain bad data) and classifiers that were randomly assigned these subsets did very well.

Stranger Detection

We devised two methods for detecting strangers:

1. Use an image's E2 K-means distance. This distance is its distance to the nearest centroid. If this distance is larger than some threshold, we consider the image as that of a stranger. We decide the threshold value by taking it as the E2 K-means distance of the training data (e.g. 75% of training data images had E2 distance < 500 so the threshold is 500)
2. Similar threshold to before except we use the cosine distance as a metric.

The E2 distance method seemed promising, but we ran into some difficulties. Below is a chart summarizing the source of inaccuracies we detected in our training and testing. The False Positive part of the stacked bar represent errors where our k-means correctly predicted the identity but because of our thresholding we designated that image as a stranger.

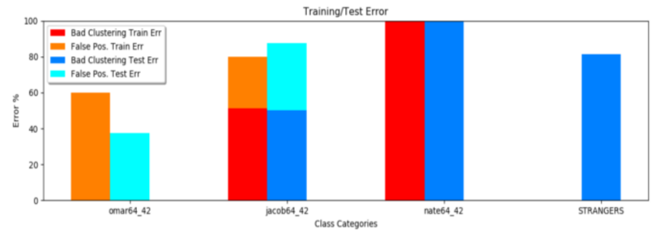


Fig 9 – Chart depicting source of inaccuracies, with dark blue as poor clustering test error, light blue as false positive test error and red and orange depicting the same for train error respectively. Each set of bars represents a different person

After experimentation, we narrowed down the cause of this poor E2 performance to be that we were not collecting enough training data and our thresholding was too stringent. We gathered more than 300 images total of training data (thus getting more of a representative view of the E2 distances of the training set) and then took threshold of nearly 100%.

We also utilized cosine distance as a metric for detecting strangers. Understanding whether to use E2 or cosine was given lots of analysis. Below we plot their histograms (top row) and percentile distributions (bottom row). We seek to understand which gives a better delineation between stranger and non-stranger data points. We deemed that cosine gave the best delineation. Notice the top right plot shows cosine distances having the training distances being concentrated in a large spike. This means most training images had very low cosine distances. The top left plot shows E2 having more overlap and less distinction.

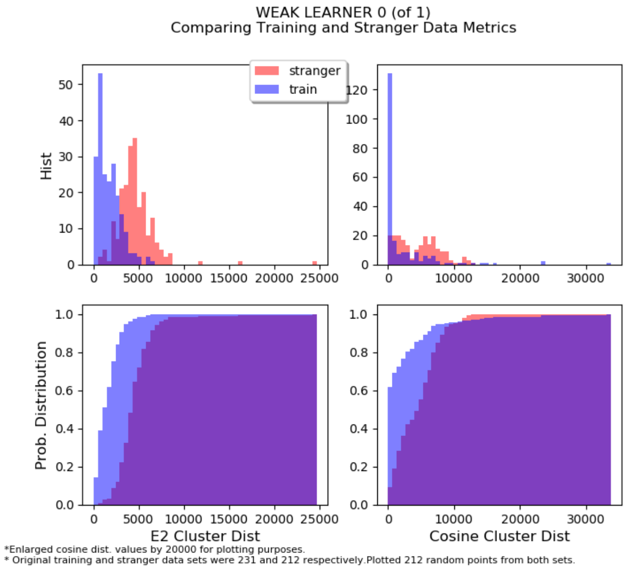


Fig 10 – Plotting E2 and cosine distances as histograms (top row) and percentile distributions (bottom row).

Eventually we deemed the best approach was to use both. They both gathered useful insights that the other did not have. If either threshold was exceeded, the image was considered a stranger.

C. *Participation – Hand Raise Detection*

We had many different ideas for the design We eventually settled upon a simple, yet effective algorithm. First, we seek to learn the pigments of each of the students faces. This is done by utilizing the fact that our recognition and detection gives us the faces and identifications of students for free. After this, we extract pigments from these faces and scan the portion of the image around each face for similar pigments.

More specifically we learn the pigments of the face by taking a circle centered at the face bounding box and gathering all pixels in that region. It's important we do not get any hair or background pixels. These pixels are each in 3-D (for RGB colors) and we perform k-means in 3-dimensions with some small number of clusters (about 6 clusters). These clusters represent the main colors on the persons faces.

Fig. 11 – The 5 dominant clusters following k-mean. Each cluster centroid is displayed by displaying its RGB color above.

Of these colors we select a few of them (about 2) to use in masking the region of the image around the persons face. These two colors would be the dominant centroids. We take the portion of the image around the detected face and for each pixel in that region we decide whether that pixel's color is "close enough" to one of our two dominant colors we got from the face. Closeness is decided using Euclidean distance. More specifically, for all the training pixels from the face that were assigned to dominant color number *i*, we find at what E2 distance does 60% of those training pixels lie below. This distance is used now as a threshold for pixels lying outside the face (i.e. above the face where a raised hand would be). If the pixel E2 distance to the dominant color is less than the threshold distance, then that pixel is kept. Thus, this produced a mask (below).

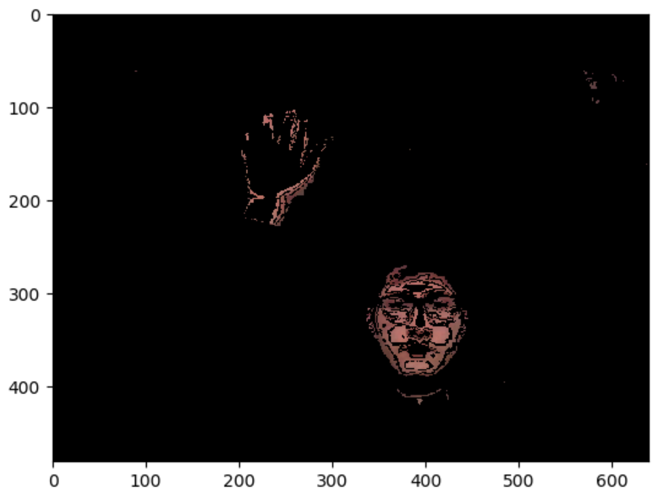


Fig. 12 – Mask produced from the hand raise detection algorithm

Finally, we perform another k-means. This time we perform it on the masked image, where the data points are only the kept pixel row and column locations. We use two centroids. Thus, if a hand was raised, the first centroid would be on the face and the second would be on the hand (see above image). If no hand was raised, k-means would seek to put both centroids in the face, in which case we can easily detect that.

This simple algorithm performed remarkably well.

D. *Front-End UI/ Visualization*

Our front-end UI consists of three different live plots. The first of these is the live video capture that is developed using OpenCV. The video capture shows the user the facial bounding boxes given by Face Detection and also shows the classification of the face above the bounding box. An example of this is found in Figure 13.

Next, we have a live plot which shows the user where the detected faces in the current video frame are projected to in the PCA + LDA computed vector space. This plot shows the training images, colored by class, along with the decision boundaries. An example of this is found in Figure 8.

Finally, we have a plot for each student which shows when the camera detects them as present and has dots plotted on the line when it detects a hand raise. It also has a row for stranger where it plots when strangers were detected in the video frame. An example of this is found in Figure 14.

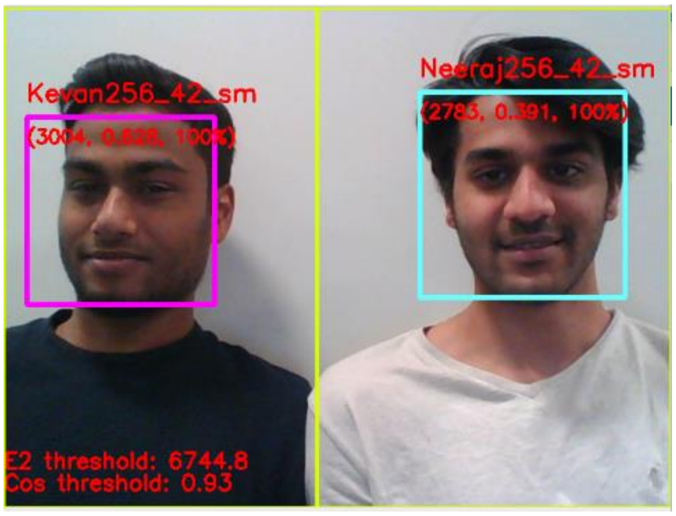


Fig. 13 – Live Video Capture

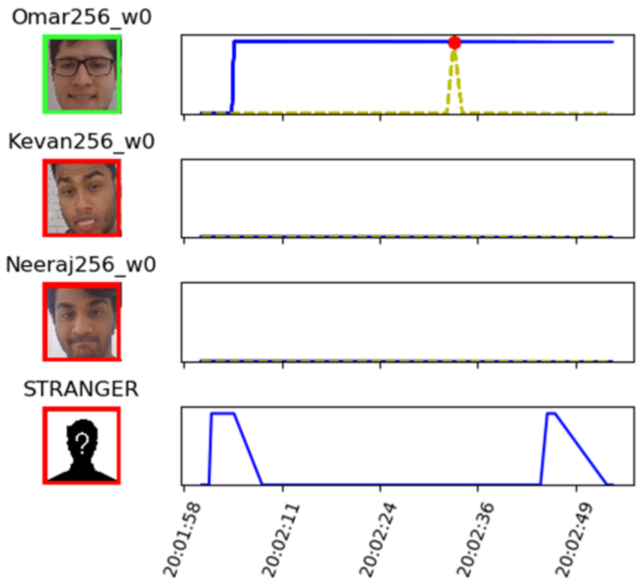


Fig 14 – Live graph showing who has been detected in the video during the video capture session and at what point they raised hands.

We developed a script to very easily gather a student's training images. The student must simply look into a camera for 100 seconds, changing his or her facial expressions and head tilt while the script does all of the work of gathering clean data.

In summary, the various components of the implementation and how they are developed, i.e. library handled or developed from scratch are below:

Detection
Viola/Jones algorithms from scratch, using OpenCV and numpy for image I/O, numpy for matrix operations.

Recognition
We developed all algorithms (PCA, LDA) from scratch with the exception of basic math functions like eigenvalue computation, matrix multiplication, done using numpy. For eye detection, we will use OpenCV and Dlib. For the SVM component, we use scikit-learn SVM.

Participation
We develop this from scratch, with the help of scikit K-means.

Collection of Results
The live plots are done using OpenCV and Matplotlib.

V. Final Results

The testing for our project is two-fold, automated and manual. Since all the separate algorithms can be developed and tested individually, we use automated testing for each of them as they are developed. This involves having set testing images that are fed to each of the modules with known truth values that we compare the results to. Manual testing refers to testing done with the live webcam.

Automated testing for detection resulted in an accuracy of *95.2%* in detecting faces

In the automated testing on our test image set for recognition we obtained *100%* accuracy on our final test set, where we collected training data in the same lighting conditions as test conditions. Our stranger detection accuracy was *92%* on this test set as well. These results clear our original metrics table (Figure 1).

Our live testing for facial detection produced very good results (as seen in Fig. 5). Lighting conditions played a role in detection rates, with poorly lit rooms resulting in the worst detection. We adjust the threshold value to handle the different lighting conditions.

Live testing for recognition had similar performance to detection. When training data was obtained from lighting conditions similar to testing conditions, we obtain very high accuracy in recognition. It was difficult to obtain exact numbers on live testing since testing with the still image test set was more straightforward to collect results.

Hand Raise Detection also produced excellent results, once again with the caveat that results were sensitive to lighting and background changes. In a room with a plain white or gray background, the Hand Raise Detection detects correctly identifies hand raises between 90 – 100% of the time. This includes false positives where we accidentally classify segments in the background as a hand raise.

VI. Project Management

A. *Member Responsibilities and Scheduling*

The division of labor for the project was as follows:

Task	Team Member
Facial Recognition algorithm	Neeraj, Omar
Facial Detection	Kevan
Raised hand recognition algorithm	Omar
Integration of Facial Recognition with an Attendance system	Everyone
Testing and fine tuning the parameters for optimizing performance	Everyone

Fig. 15 – Table showing division of labor

Our original schedule can be seen in Figure 16. We had to change our original schedule a decent amount over the course of the semester due to our performance for recognition and detection being much lower than desired. We did build the main file which integrates the various components when we were supposed to in the schedule but completed most of it earlier than expected and didn't spend much more time on refining it since we wanted to work on our recognition and detection accuracy first. Hand detection was completed at the same time as the main file, as the skin tone matching was initially done as an experiment but produced very good results, and so we decided to use it as our final detection method. In summary, our project schedule ended up as follows:

- **(Feb 11 – March 11)** Working in parallel on prototyping and implementing Viola Jones, PCA and LDA.
- **(March 12 – March 17)** Main file for integrating detection and recognition completed along with Hand Raise Detection.
- **(March 18 – April 19)** Continued work on Viola Jones, to improve accuracy, involving rewriting the base implementation, retraining with AWS.
  Also continued work on recognition. Involved rewrite of PCA and LDA, changing from K-means to SVM.
- **(April 20 – May 6)** Final testing and fine-tuning parameters in all areas of the project for the demo and collecting results. Also involved collected many more sets of images for testing use.

B. *Tools Used, Bill of Materials*

The overall budget for the project is the cost of two camera stands, two web-cameras, and an extension cord. This was $128 for the camera and $26 for the cables.

Otherwise all of the other requirements are software requirements which can all be satisfied using free and open source software. Combined with AWS credit we are still under the $600 budget for the project.

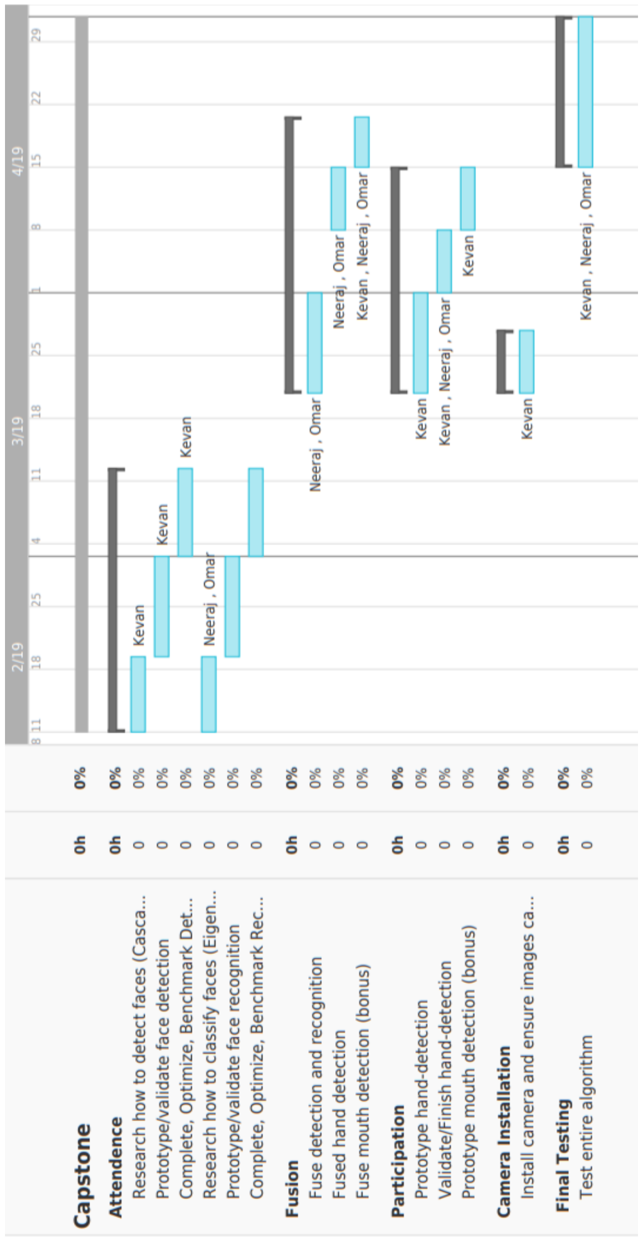


Fig. 16 – Initial schedule of tasks to complete

C. *AWS Credits*

We used AWS to train our viola-jones classifier. Training 75 features with 8000 positive and 10000 negative images took 5 days on a 16 core GPU optimized instance. Due to this, we used $450 worth of AWS credit to train multiple classifiers (with some trial and error runs). We made various changes to the training algorithm over the semester, which is why we had to train the classifier multiple times and used ~480 hours of AWS compute.

We would like to thank Amazon for providing us with these AWS credits to help us complete our project.

VII. Summary

In conclusion our final implementation performed quite well. We met the specifications that we set out when it came to testing of still images and performed very well when it came to live testing when conditions matched those in our training database images. Given that we provide a means of easily collecting database images, it is reasonable that images can be collected in classroom conditions itself.

Our projects limitations point to some very obvious areas for future work that would not require large amounts of effort. Lighting conditions impact our results quite a bit some obvious further work would be to include some preprocessing to make our implementation robust to lighting changes. We also could try and run the code on a GPU to improve our live demo framerate as that had a strongly negative impact on our ability to do live testing for the project.

Overall, we are very proud of the results and work that went into this project as we built most of the components from the ground up, only using external libraries to assist in computing things such as equations. This allows us to claim that we truly understand the major components of our project in great depth and that is certainly pleasing.

Lessons Learned

We learned a lot from doing this project and there is definitely insightful advice that can be passed on to future students. First and foremost, we believe that more consideration could have gone into our schedule at the start of the semester. This would include splitting up larger tasks on the schedule into much smaller tasks and spending more time thinking about what would be challenging and consume larger amounts of time. We also believe that rewriting our code was very helpful when we had errors and wish we had done so sooner as the rewrite of our code resulted in our results improving significantly. Finally, when it comes to computer vision or machine learning related projects, better training data can sometimes make the difference when it comes to accuracy.

VIII. References

Here we provide references for sources that were helpful during the project, even if they were not specifically used to write the report.

[1] Rosebrook, Adrian. “Face Alignment with OpenCV and Python.” *PyImageSearch*, 3 Aug. 2017, www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/.

[2] Wagner, Phillip. “Face Recognition with Python.” *Bytefish.de*, 18 July 2012, www.bytefish.de/.