Identity Recognition

Srihari (Hari) Seshadri [sseshadr]
Aneesh Devi [adevi]
Group #2
18-551 Spring 2011

Hari Seshadri

Aneesh Devi

Table of Contents

Section 1: Project Overview

So much about us, such as our names, contact information, educational background, work history, and even friends, is available on the internet; we've formed an "identity network" online. For our capstone design project, we chose to build an identity recognition application to index into this network. The design and implementation of our project required cross-disciplinary knowledge from signal processing, pattern recognition, and to an extent even machine learning. In a nutshell, the goal of our project was to retrieve data on an individual, representative of their identity, given just one image of their face.

1.1 Project Motivation

One question, which many people, companies, or agencies ending up asking, is "who are you?" Because it's often useful to know information about a person, many tools have been developed to facilitate gathering information on individuals. For instance, given a person's name, one can find contact information about that person in a phone book. To find about someone's educational background or field of interest, you might look them up in a college directory. If you're interested in their professional career, you might be able to learn about them on LinkedIn. Facebook acts as a similar data source for a target person's social life and connections. If you're willing to work a little bit more, you might be able to find even more information (such as what papers they've written, which articles their mentioned in, etc.) by using a web search engine. The point is that technology, and specifically computer internetworks, has made so much information about individuals available publicly. Unfortunately, it's very difficult to search through this data unless you come up with a good textual identifier of the target person (such as their full name). Sometimes, such as in crime-fighting, even retrieving the name of an individual can be challenging. Our goal was to tap into this vast resource and extract rich but pertinent data about a target given just an image of their face; we wanted to use an image of a person's face as the primary key to

index into online databases and extract data. To visualize the end result "goal" of our project, we designed a mock screenshot of the application:

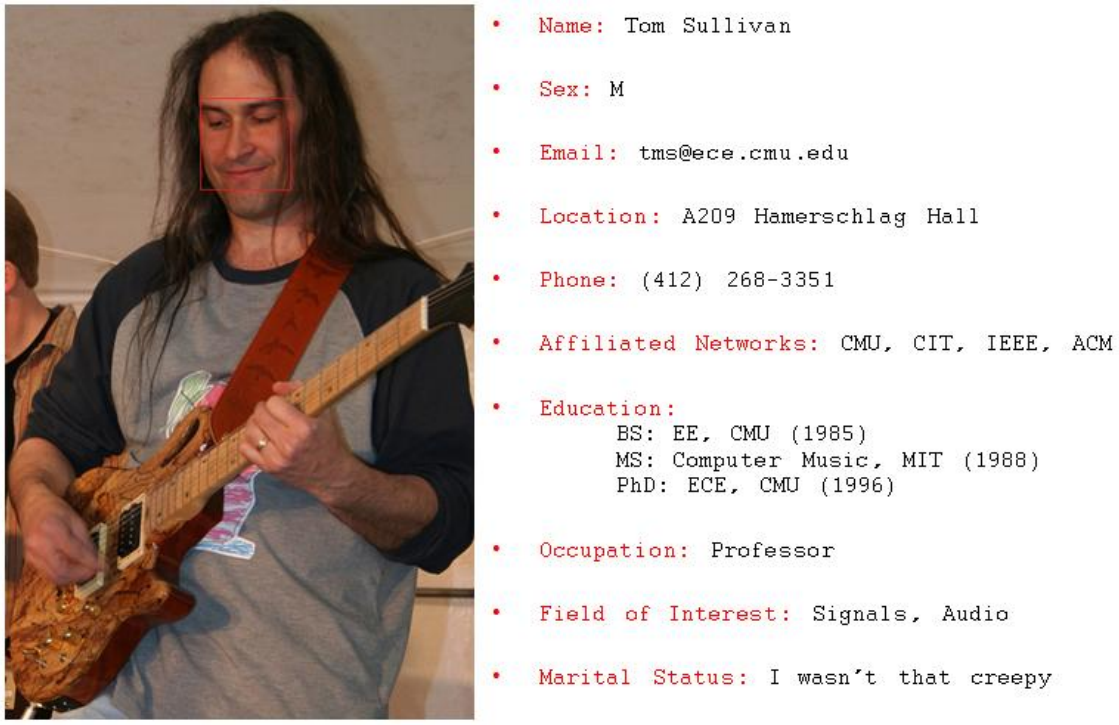


Figure 1: Mock Screenshot

1.2 Restricted Problem

An application which could effectively lookup information on arbitrary individuals from just their face would be incredibly useful. Surveillance cameras could link to this software to help find wanted individuals, and intelligence agencies would have tons of uses for it as well. We knew that trying to solve such a large problem in a semester was unrealistic, so we decided to limit the scope of the problem.

The first big limitation is that the target input image would be a full-face, frontal image planar to the camera lens; rotations in the frontal plane are tolerable, but skew in the other two dimensions are

not.

Figure 2: Planarity

The second limitation we made was that the only identifiable targets were the students and teaching staff for the course. This was much more practical than comparing the probe image to all of the targets in the world, and also allowed for us to have a cool demonstration at the end of class (discussed more in Section 7.2).

The third limitation was that the information we "looked up" on the target individuals would come only from the Andrew Directory (accessible on the Andrew UNIX machines via the `finger` utility). Unlike web queries, the information in the Andrew Directory is highly structured, easily accessible, and contains relatively low noise. In the real world, we would have to deal with problems such as incomplete data, categorization and parsing, and dynamic attributes (for instance a professor might have a "Field of Interest" but a professional wrestler may not). Making this third restriction helped limit the scope of our project to more signal processing than search and data mining. However, we made our application modular and separated the recognition from the data lookup, so all new features, such as a Facebook Lookup, can be added in the future (discussed more in Section 9.5.2).

1.3 Tasks

After outlining what we'd like the final product application to do, the next step was to come up with smaller tasks that partition the work. In our case, the work for the application could be divided into the following four sub-tasks: **face detection** (Section 2), **face normalization** (Section 3), **face recognition** (Section 5), and **entry lookup** (Section 6)**.** Face detection refers to finding where in the image the face lies. Face normalization is the process of extracting the same "features" in the face throughout various images so that the recognizer can compare apples to apples. Face recognition matches an untagged probe face to one of the target faces based on similarity in the features. Finally, the application goes through an "entry lookup" stage to retrieve information on the tagged probe.

1.4 Novelty

Groups in previous years have worked with facial detection and recognition. Such groups include:

1) Facebook Tagging – F09, Group 1
2) Identification of Surveillance Images – S02, Group 2
3) Restoration of Partially Faded Face Images – F05, Group 6
4) Transformation From Non-Frontal Facial Images to Front-View Pictures – F06, Group 11

Our project was somewhat similar to the Facebook Tagging project from Fall 2009, except the idea for us was more for recognition and bio-information retrieval than for simply tagging. The second project listed focused on *detection* almost entirely; the goal of their project was to identify which pictures from a surveillance source contain faces. The third project deals with faces too, but only to the extent for faded image restoration. Finally, the fourth project deals with a dimension we chose to ignore due to complexity, namely, non-planar faces. Moreover, our group came up with some genuinely new ideas. The most dramatic was the polygon-model for the face (as opposed to the traditional rectangle model),

discussed in Section 3.4. Another was the from-scratch implementation of Fisherfaces, as discussed in Fisherface paper[1]. Next was the concept of increasing the targets-per-class ratio with artificial illumination (Section 4.4). Finally, the thought of indexing into an identity network was a novel concept we formed as well.

Section 2: Face Detection

Face detection, or finding where the face is in an image, is vital for face recognition. To see why, let's assume that we didn't have a face detector, but rather ran the entire face recognition algorithm on the entire picture (as opposed to just the face). The recognition "learner" would not know any better than to assume that all of the pixels in the image are important. To simplify my adversarial argument, assume there are only two targets, John and Mary, which the recognition learner is trying to classify against. Furthermore, assume the target images are waist up pictures consisting mostly of the shirt, but also of the face, of the person in question. If John is always wearing a red shirt in the target sample pictures, and Mary is always wearing a white shirt, the recognition learner is likely to classify a red-shirted picture of Mary as John just because the shirt is such a large part of the picture. To avoid this problem, we need to feed the recognition learner only the pixels that matter; namely, the ones in the face.

2.1 What is Face Detection?

Before we can answer that, let's ask an even more fundamental question: what is a "face" to a computer? We might understand a face to be a semi-circular object with two eyes, a nose, and a mouth, but how is a computer to determine whether a given rectangular grid of pixels contains those elements in the right locations? For instance, if I took a picture of a human face and started blurring lines, shifting pixel blocks, and adjusting the colors, *when would it stop being a face?* There isn't one correct answer to that question, and in fact implementing a face detection program is non-trivial. In practice, the computer

[1] See Reference #1

is taught a "model" for a face (based on where it should find eyes, the nose, etc.) and only images which fit the model are considered to be faces. From this definition, we can define face detection as the process of finding pixel landmarks, in a probe image, corresponding to the model of a face. In order to save us time, the CyLab Research Group gave us a working detection binary which we used, as is, in our project.

2.2 Implementation, Functionality, and Restrictions

The face detector we were given was implemented using the popular Viola-Jones algorithm[2]; the exact implementation of Viola-Jones is outside the scope of this paper. The detector took as input an image containing a face, and produced a *facial landmark points* file with the interpolated pixel points[3] of the landmarks in the facial model. It could also annotate an image with facial landmark points file by drawing markers at the pixels near the landmarks. If the input image contained nothing which fit the facial model, the program would assert that no face was found and terminate. Finally, if multiple faces were present in the input image, the algorithm would tend to find the facial blob of highest pixel area.

Figure 3: Face Detection

The detector we were given, albeit extremely useful for our project, did come with quite a few restrictions. Firstly, the facial model was only able to, for the most part, recognize full-frontal, planar faces. This is part of the reason why we chose to restrict out entire identity recognition application input

[2] See Reference #2
[3] Although pixel points are integral, the facial landmark point files contain decimal pixel coordinates to indicate *precisely* where the algorithm found a given landmark (calculated by interpolating amongst nearby pixels).

to such images. Another restriction was in the speed of the detection; we found that the clock runtime to execute the given Windows binary (with a standard AMD64 laptop) on a 500 by 300 pixel input image was between 1 and 2 seconds. Although this wasn't so bad for our single probe image demonstration, this wouldn't be able to run real-time (in say, a video) at 30+ frames per second. Yet another restriction was the accuracy of the detector. Because "being a face" isn't an absolutely objective characteristic of images, it's tough to quantify success rate. In general, however, Viola-Jones face detectors are credited with having a low classification error rate[4]. Experimentally, we noticed that the face detector performed very poorly when the image had a planar rotation of more than 10 degrees. In that case, the face detector would usually either report that no face was found, or the landmarks found (specifically the jawline) would be off by a large margin. Even without rotations, the face detector often found incorrect landmarks. Below is a full-frontal, planar, facial image, with very little rotation, that's been annotated and landmarked by the given detector.

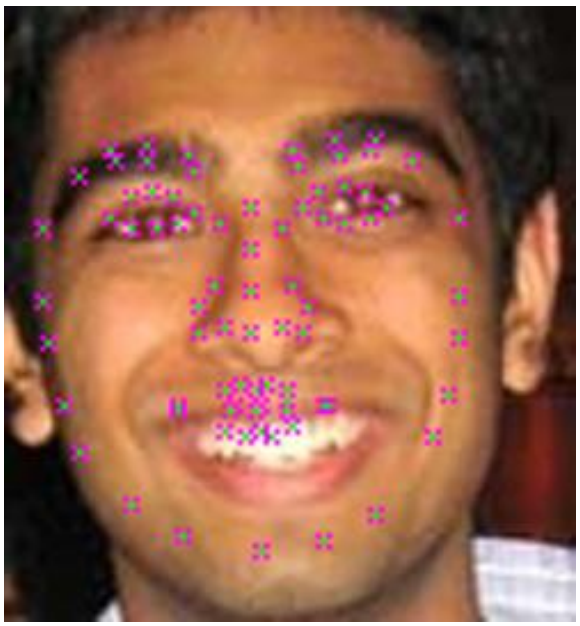


Figure 4: Incorrect Landmarks

[4] Based on the comparison amongst detectors on page 21 of the Viola-Jones paper

Upon close observation, we see that the detector incorrectly found the jaw-line as just stretch marks due to the smile. The mouth wasn't found correctly, and even the locations of the right eyebrow and eyes are slightly off. With that said, we rarely observed a total false positive or false negative with the face detector; generally the errors were the placement of the landmarks.

2.2.1 Available Landmarks

There were 79 landmarks which were annotated in the facial landmark points file. There exact indices and locations are shown in the figure below.

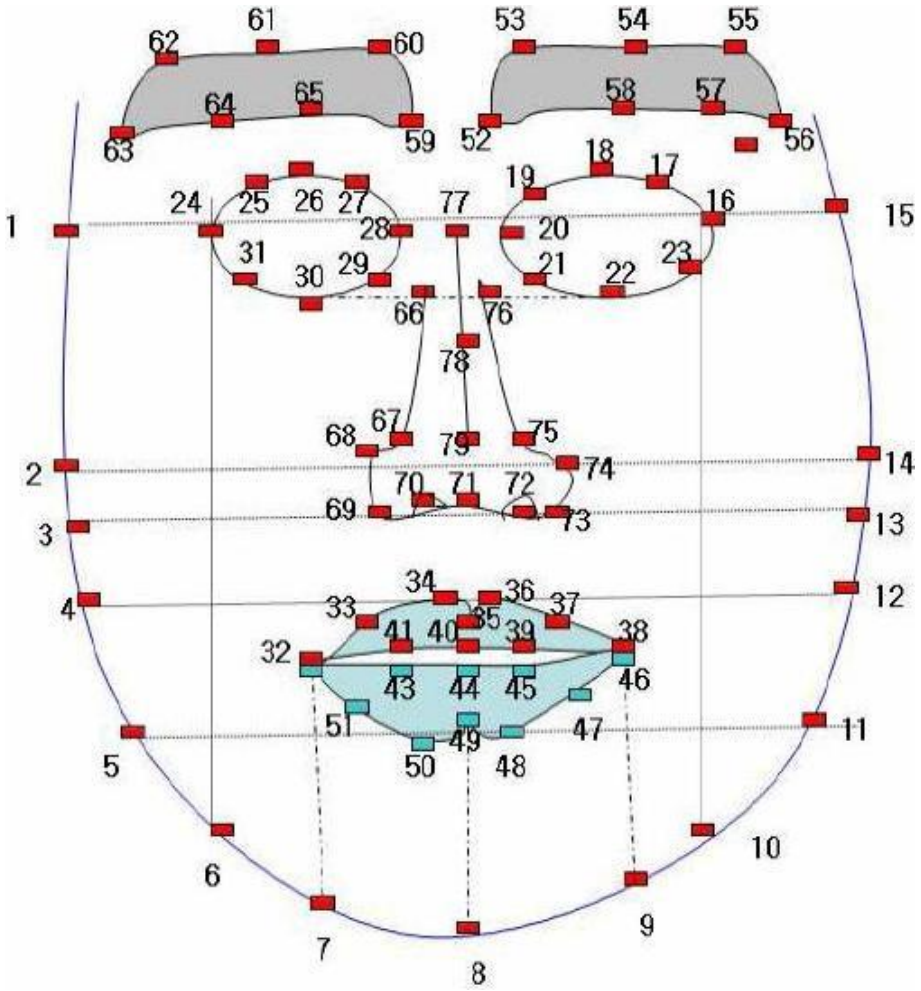


Figure 5: Facial Landmark Points

2.3 Interfacing

Our program, written in Visual C++, needed to interface with the given Windows binary face detection program. We did this by spawning a process to run the binary. Specifically, given an input image, we would launch the face detector binary in a new process, which would then write the interpolated pixel coordinates for each of the 79 landmarks out to a file. By reading this file, our program was able to interface with the face detector.

Section 3: Face Normalization

The distinction between face normalization and face recognition is actually an artificial conceptual barrier used for modularity. In truth, the normalization process goes hand in hand with the classifier; having a great normalizer makes the job of the recognition engine simple, and vice versa.

3.1 What is Face Normalization?

The concept of normalization is not unique to faces, or even images. Usually normalization is meant to offset some sort of affect which, if left untouched, would largely skew the results. For instance, when comparing the bytes corresponding to two audio files, one might normalize them so that the average energy in the audio files is the same (to account for one file being louder than the other). Likewise, we came up with our own criterion for normalization (within the domain of facial images):

1) After normalization, two different pictures of the same person should look similar. This applies even if the individual has a different pose, facial expression, or is in different lighting.
2) After normalization, pictures of different people should not look similar.
3) Any facial picture, after normalization, will have exactly the same number of dimensions as any other post-normalized facial picture.

The first criterion is the most obvious for normalization; the normalizer should account for variations in the face (such as pose, tilt, and mouth expression) so that all pictures of "John" end up being similar. The second allows the face recognition engine to discriminate between targets; if everyone looked similar discrimination would be tough. Thus the requirement that different individuals should not look similar. The third criterion is a specific limitation because of how we plan to *use* these normalized faces. The algorithms which we implemented later on (PCA[5] and Fisherface recognition[6]) required that the training and targets all have the same dimensionality. In machine learning or pattern recognition, each training or target instance (in our case facial image) is characterized by a set of <feature: value> pairs. Each feature is called a dimension. In an advanced recognition system, features for a face might include soft biometrics such as hair color and gender (see Section 9.3). In our case, a "dimension," or feature, was simply a pixel in the normalized image. Thus, the third criterion simply states that the normalized images must have the same number of pixels.

3.2 Naïve Normalization

At first, normalization doesn't seem like such a daunting task since we're given a face detector. One obvious normalization scheme would run detection on the facial image, and crop the image down to the smallest rectangle which contains the face. The y-coordinate of the top-most landmark and the x-coordinate of the left-most landmark would together form the top-left corner, and the bottom-right corner could be calculated similarly. Unfortunately, this algorithm entirely neglects variations due to planar rotations in the facial image.

3.2.1 Planar Rotations

[5] Principal Component Analysis is discussed in Section 5.3
[6] Fisherface recognition is discussed in Section 5.5

A refined normalization procedure would take into account pose by performing tilt correction prior to cropping the image. One way to tell the "slant" of a face is by looking at the slope between the pupils in the face; a positive slope means that the image is rotated counter-clockwise, and a negative slope means the image is rotated clockwise. The pixel coordinates of the pupils were estimated as just the average of the "eye landmark points" (in Figure 5, points 24-31 [L] and 16-23 [R]). The next figure shows an intermediary stage when finding the slope of between the pupils prior to performing the rotation.

Figure 6: Tilt Correction

Unfortunately, the face detector itself doesn't always find the eyes perfectly (especially when the input image is rotated), so often times the slope correction isn't enough to make up for the true rotation. Also, if the image is rotated enough (more than 15 degrees roughly), the face detector will fail to even *find any face* in the image.

Assuming we are given the exact slope between the pupils, how can we account for this and rotate the image? First, define m as the slope of the line and θ as the counter-clockwise angle between the horizontal line and the line connecting the pupils. By some elementary trigonometry, we see that

$$\theta = \tan^{-1} m \tag{3.1}$$

Now, let $(x, y) \rightarrow (x', y')$ define a mapping from the tilted image to the tilt-corrected image. Specifically the tilt-corrected image will be the collection of (x', y') pixels which result from applying the mapping over the pixels (x, y) in the original image. Note that any pixel (x', y') which does not get mapped to can be colored black. It turns out that the mapping itself is linear (with weights proportional to trigonometric functions of θ). Specifically, the mapping is

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.2}$$

By applying this map to the image shown before, we get the tilt-corrected version, shown below:

Figure 7: Linear Mapping Rotation

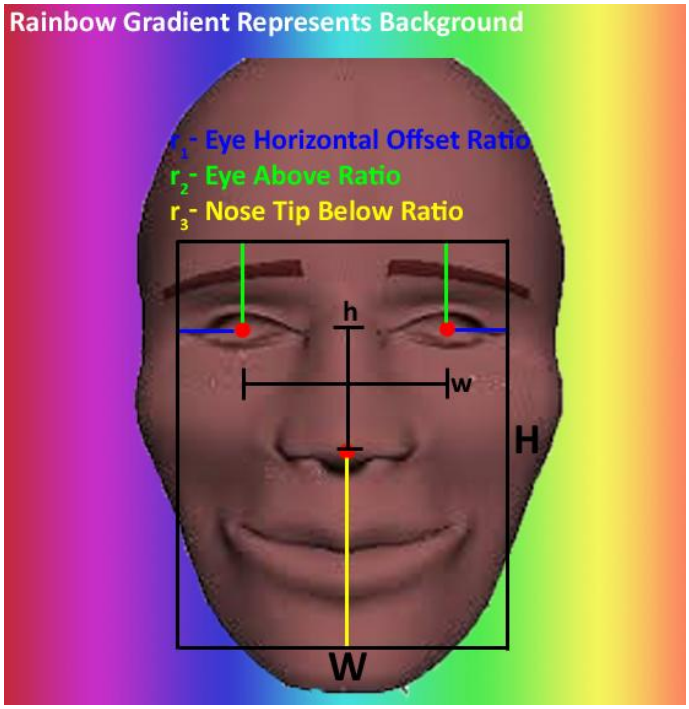
So, our new normalizer would perform the rotation and then find the smallest rectangle which contains the entire face. Note that because we don't perform any cropping before the rotation, we wouldn't have blacked out portions of the face as we see in the figure above.

There are still two major problems with this type of normalization. First, nothing enforces that each normalized picture has the same dimensionality (number of pixels); even if the camera

is the same, images taken closer to the camera will have more facial pixels than a face far away. Fortunately, the fix to that is to just resize[7] the image to a preset resolution. The second is that the corner landmarks produced by a Viola-Jones detect *are far less accurate* than the landmarks of a landmark detected within the object. Thus, "the smallest rectangle containing the whole face" is not a very good crop.

3.3 Landmark-fixing Normalization

Landmark-fixing normalization starts the same way as naïve normalization: before any further steps are taken, the image is corrected for tilt. Then, instead of finding the smallest rectangle which contains the face, our landmark-fixing normalizer finds a much more robust set of landmarks and *crops the facial image based on those landmarks*. Specifically, our landmark-based normalizer uses the **left pupil**, **right pupil**, and the **tip of the nose** as the robust landmarks to decide where the "face" lies.



Meaning of the ratios r_1, r_2, r_3

1) $r_1 \in [0, 0.5)$
The x-coordinate of the left pupil will be normalized to $r_1 W$, and the x-coordinate of the right-pupil will be fixed to $W(1 - r_1)$

2) $r_2 \in [0, 1)$
The y-coordinate of the pupils will be $r_2 H$

3) $r_3 \in [0, 1 - r_2)$
The y-coordinate of the nose tip will be $H(1 - r_3)$

$$W = \frac{w}{1-2r_1} \qquad H = \frac{h}{1-r_2-r_3}$$

Note: The origin is defined as the top-left corner of the black rectangle. Also recall that each input face determines w and h.

Figure 8: Landmark-fixing Normalization

[7]There are many ways to interpolate pixels during a resize. We used the OpenCV1.1 default resize implementation.

Note that the coordinates of the three red dots (left pupil, right pupil, and nose-tip) can be inferred from the given landmarks. r_1 is the parameter which determines how far sideways from the eyes, with respect to the distance between the eyes, to consider the facial region. r_2 is the parameter which determines how far above the eyes, with respect to the distance between the eyes and the nose, to consider the face. Finally, r_3 determines how far below the tip of the nose, with respect to the distance between the nose tip and the eyes, to include in the face. Recall that since this is done *after* the tilt correction, we are guaranteed that the eyes have the same y-coordinate.

In effect, setting these three parameters fixes the locations of the two pupils. For instance, if I resize all of my faces to 200 by 200 pixel image, then I can guarantee that (for a given r_1, r_2, r_3), the two eyes and will always be at the exact same pixel coordinates. I can even guarantee that the y-coordinate of the nose tip will be the same (but since the nose isn't horizontally aligned I can't make such a claim about its x-coordinate). It is this guarantee that makes landmark-fixing normalizers inherently better than the naïve normalizers.

3.4 Etching

One idea we used in this project which wasn't based on any papers or other projects was the concept of etching. Traditionally, face recognition systems have modeled the 2D face as a rectangular grid of pixels. Not only is this a reasonable representation of how the face actually looks, but it also is very convenient as almost all image interchange formats store image dimensions based on a width and a height. The motivation behind etching was that a face, in two dimensions, isn't really a rectangle, but rather more complex geometric figure with some smooth curves. Because it would be too difficult to model smooth curves, we decided to make an approximation and model the 2D face as a polygon. Specifically, the points 1-15, 56, 55, 54, 53, 60, 61, 62 (in that order) formed the circumference of our 22-

sided face mask polygon. The motivation behind this was so that pixels that lie in the rectangle of your face, but not actually in your face (such as under your jawline) shouldn't be fed into the face recognizer.

A true polygon model for a face would consider *only the pixels inside the polygon.* Although this would be an ideal mask, it would make the third criterion for normalization almost unattainable; there isn't an easy way to resize arbitrary 22-sided polygons to have the same number of pixels. Thus, the closest we were able to get was by simply etching out the pixels of the rectangle which are outside of the polygon. In effect, we are not allowing the recognition learner to learn any information outside of the polygon. It's important to note that the *rectangular region* from which points are etched is determined by the landmark-fixing normalization parameters r_1, r_2, r_3, and that the actual points which are etched out or left in are determined by the polygon mask. Note that because the polygon mask could be different for every face, there is no guarantee that if a given coordinate is etched out in one normalized facial image that it will be etched out in another.

Figure 9: Polygon Mask with Etching (parameters $r_1 = 0.25, r_2 = 0.2, r_3 = 0.5$)

Although we didn't realize it right away, it turns out that etching a pixel to the black color can have harmful side effects in the recognition stage. As stated at the end of the last paragraph, there is no

guarantee that a given pixel coordinate is etched out for a set of parameters, because every face polygon is different. Also, because the RGB components of black are very different from the components of usual skin tones, the recognition engine will discriminate against someone *heavily* for having a different etched circumference (as opposed to what's actually inside the etched circumference). To prevent the recognizer from discriminating too heavily based on the etched out pixel values, we had another option to color the etched out pixels to *the average skin color of the pixels in the face*. This is to prevent a large difference in RGB values from biasing the recognition learner on the basis of the etch circumference.

Figure 10: Polygon mask with Etching Variant (parameters $r_1 = 0.25, r_2 = 0.2, r_3 = 0.5$)

Because we had already implemented code which could determine whether a pixel was inside or outside a polygon (specified by its circumference pixels), we thought about dealing with another normalization issue: facial expression. If we were able to etch out the pixels inside the mouth of the facial image, this would normalize open mouth pixels, closed mouth pixels, and teeth pixels! Since we're given the pixel coordinates of the circumference of the mouth in the facial landmark points file, we were able to implement this feature as well. Shown below is an image with the pixels inside the mouth polygon, or outside the face polygon, etched out to the average skin color.

Figure 11: Etching out the mouth to eliminate variation in facial expression (parameters $r_1 = 0.25, r_2 = 0.2, r_3 = 0.5$)

Notice that all of these "improvements" will not necessarily help the recognizer; these are merely options we are incorporating into the normalizer so that we can choose which set of parameters give us the best results. Although I'm helping achieve the first normalization criterion (making all images of the same person look similar), I might be hurting the second normalization as I might be etching out discriminating features between people. For instance say one target has a distinctive mark on the upper left lip which could potentially be used to recognize him. This etching feature would throw away that feature before the recognizer could even make use of it, potentially hurting the performance of the overall program.

3.4.1 Point-in-Polygon Algorithms

Some of the more interesting algorithms in this project were the point-in-polygon algorithms. If I drew the three vertices of a triangle, and then plotted a fourth point, a human could easily tell me whether that point lies in the polygon defined by the first three points by visual estimation. Now instead if I gave you n coordinates (of the form (x,y)) which defined some arbitrary (not necessarily convex) n-gon and then gave you a probe coordinate, it would be slightly harder to determine whether the probe point lies inside or outside the polygon; a human

might draw out the polygon, plot the point, and use visual estimation from there. Since computers don't have the skill of visual estimation, in order for us to implement the "etching" feature, we needed to code decision procedure to tell us whether a point is inside or outside a polygon.

The first such algorithm uses an angle summation trick to see where a point lies relative to a polygon. Specifically, the angles in between the lines connecting the probe image and the vertices of the polygon sum to 2π if and only if the point lies within the polygon.

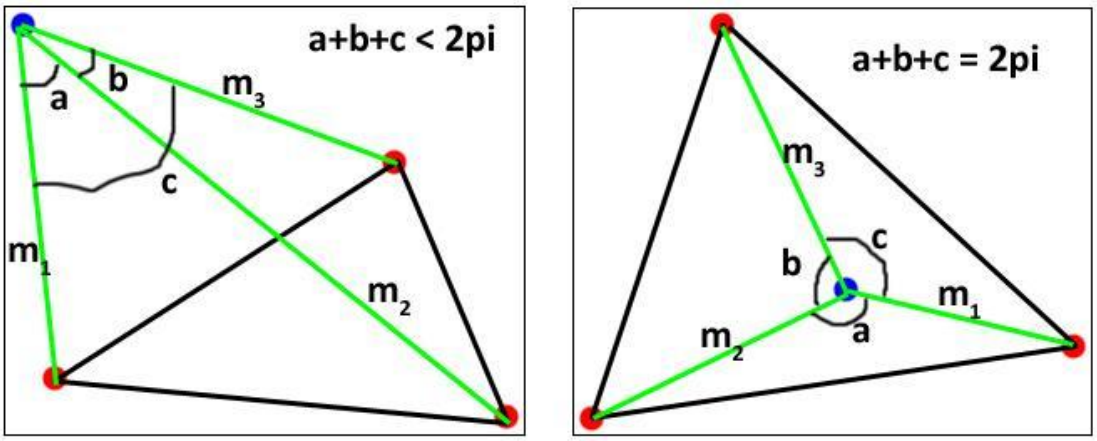


Figure 12: Point-in-Polygon Algorithm #1 decides that the point is outside (LEFT) or inside (RIGHT)

Note that the counter-clockwise angle, $\theta \in (-\pi, \pi]$, between one line (of slope m_1) and another (of slope m_2) is given by

$$\theta = \tan^{-1}\left(\frac{m_2 - m_1}{1 + m_1 m_2}\right) \tag{3.3}$$

Thus, by calculating the slopes between the probe coordinate and the vertices, the angle summation algorithm can *usually* determine whether or not the probe lies within the polygon defined by the vertices. Unfortunately, this algorithm doesn't work whenever the vertices describe a concave polygon. Recall that a concave polygon is one in which at least one its angles exceeds π radians in measure. Specifically, the algorithm will treat a concave polygon as the

equivalent convex polygon of lesser vertices formed by extending back all obtuse angles until they are down to π radians.

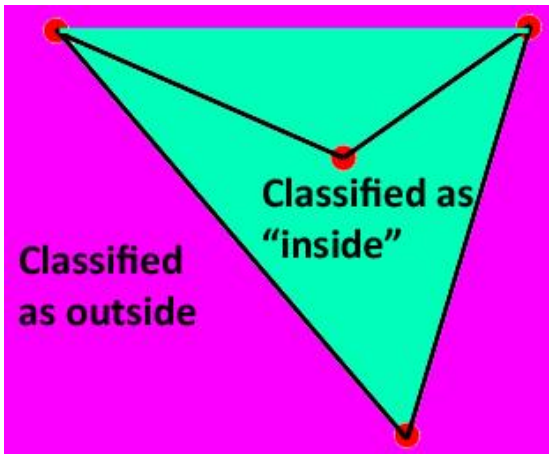


Figure 13: Point-in-polygon Algorithm #1 fails for concave polygons

Since the polygon mask we used for a face is potentially concave (between the eyebrows), as is the mask for the mouth, the above algorithm didn't cut it. The actual point-in-polygon algorithm we used works by counting the number of intersections between the edges of polygon and an arbitrary ray shot out from the probe coordinate.

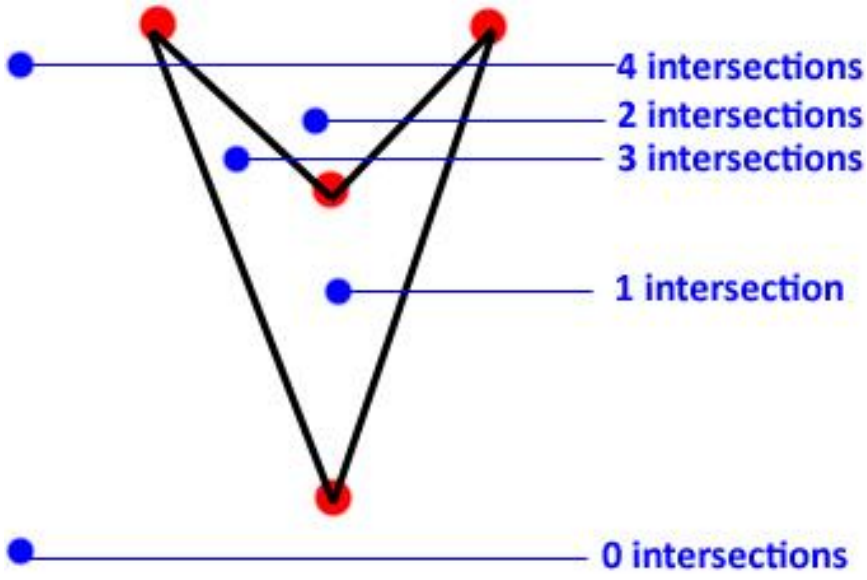


Figure 14: Point-in-polygon Algorithm #2 counting intersections

If the number of intersections with the ray (shown to be horizontal, but can be of any orientation) is even, then the point is outside. Otherwise, the point lies inside the polygon. Note that this holds regardless of the concavity of the polygon. The only logic which we had to implement was determining whether a horizontal ray would intersect with a given edge. Pseudocode for that decision is shown below:

```
/*    The edge in question has end-points (x1,y1), (x2,y2).
      Assume WLOG that y1<y2

      The point from which the horizontal ray is shot out (rightward)
      is at (x,y)

*/

if (not (y1 < y < y2))
      then no intersection

else

      /* Calculate the x-coordinate of the intersection of the
      horizontal line going through (x,y) by similar triangles */

      x_intersect = x2 - ((x2-x1)*(y2-y)/(y2-y1))



      /* If the ray shoots out rightward, then the intersection is
      only valid if it lies at an x coordinate larger than the point
      itself */

      if (x_intersect > x)
            then intersection
      else
            no intersection
```

Note: The code above doesn't deal with vertical edges or horizontal rays that intersect exactly with a vertex. Those conditions must be special cased for the algorithm to work properly.

3.5 Normalization Parameter Selection

Based on the discussion of face normalization until now, there are six parameters which will determine how the normalized face will turn out given a facial image:

Parameter Name	Description	Values it can take	Notes
Normalized Image Size	Height and Width	<Integer x Integer>	Recall that by the third criterion for face normalization, all normalized images must have the same dimensionality.
r_1	Horizontal eye offset ratio	[0, 0.5)	See Figure 8
r_2	Above eye ratio	[0, 1)	See Figure 8
r_3	Below nose tip ratio	[0, $1 - r_2$)	See Figure 8
ETCH	Whether the facial polygon mask should be used	{0, 1, 2}	0- Do not use any polygon mask 1- Mask and etch to black 2- Mask and etch to average color
KEEP_MOUTH	Whether the mouth should be etched out or not	{0, 1}	0- Keep the mouth 1- Etch out the mouth and fill in whatever color specified by ETCH

Figure 15: Normalization Parameters

Note: In the interest of time, we never adjusted the Normalized Image Size parameter, but fixed it permanently to be 80 by 60 pixels (width by height).

3.5.1 Manual Hill Climbing

Although it's nice to have parameters to play with, how can we determine what set of parameters are optimal for face normalization? In mathematical analysis, one might come up with a numeric rubric function of the six parameters and solve for relative maxima with respect to the parameters. In machine learning, people might solve parameter selection problems by writing a gradient ascent variant; such a program would try a set of parameters, score how well those settings performed, and make incremental changes to the parameters until the score stops improving. Regardless, it's a common theme that we needed to be able to **score** a set of parameters; otherwise we can't even decide if one set of parameters is better or worse than another!

Once we designed a method to assign a grade to a set of parameters, we can manually search through space of parameter combinations (making changes which we think would improve the score based on past results) until we find a parameter set that's good enough. This process is much like gradient ascent (or hill climbing), except that we would do it manually instead of writing a program to do the searching. But, before we can do this, we must first come up with a way to score a parameter set.

3.5.2 Testing Database

The only way to score a normalization parameter set is to see how it performs. Unfortunately, judging how well normalized faces matched the three criterion set forth is very subjective. Thus, we tested the performance of normalized images by seeing how well our *recognition learner* would perform on them. This is preferable for two reasons: first, grading a recognition engine is much more objective and quantitative, and second because this is a true score because this is indeed what the normalization process will be used for. Unfortunately, this meant that we would need to code up a recognition learner before we could proceed with testing and scoring. The learner which we implemented to help us decide this was the Single-Target PCA Nearest Neighbor classifier[8], as it was the easiest to get off the ground.

Even with a recognition engine coded, we still needed a *test set* on which to run the normalization and recognition. To come up with this set, we (manually) crawled the Facebook profiles of nine of our friends and found a total of 92 pictures of them in various poses. We tagged these pictures, normalized them with the given parameter set, and then scored that set

[8] PCA discussed in Section 5.3, Nearest Neighbor discussed in Section 5.6.1, and the results in 5.7.1

based on how well the PCA-NN classifier performed[9]. More specifically we used four metrics to gauge how well a given performed of parameters did on the testing set:

1) Average correct NN-confidence[10]
2) Accuracy with one guess
3) Accuracy with top two guesses
4) Accuracy with top three guesses

Before looking at the numbers, consider the actual problem at hand: we are trying to classify probe images amongst 9 different targets. Just by random guessing, one would expect the one-guess accuracy to be about 11.11% (10/92), two-guess to be roughly 22.22% (20/92), and three-guess to be around 33.33% (31/92).

3.5.3 Data

All in all, we tested 98 parameter sets and scored them based on the four metrics above. Because we had four metrics by which to grade a parameter set, we didn't come up with one universal "best set of parameters," but rather got a feel as to which parameter combinations tend to work well with each other. Recall that out of the six parameters, namely image size, r_1(abbreviated HORIZ), r_2 (abbreviated ABOVE), r_3 abbreviated (NOSE), KEEP_MOUTH (abbreviated M), and ETCH (abbreviated E), we fixed the image size leaving 5 variable parameters.

Here is some data which we obtained by using testing the Single-Target PCA + NN classifier (with various parameter combinations) on the testing set of Facebook pictures:

[9] How exactly normalization and the PCA classifier work together is discussed in Section 5.3
[10] NN-confidence is discussed in 5.6.1

M	E	ABOVE	HORIZ	NOSE	CONF	TOP1	TOP2	TOP3
0	2	0.10	0.20	0.45	16.0422%	37/92	47/92	60/92
1	2	0.10	0.20	0.45	15.822206%	36/92	48/92	55/92
1	0	0.10	0.20	0.45	15.821445%	36/92	48/92	55/92
1	1	0.10	0.20	0.45	15.816855%	36/92	48/92	55/92
1	0	0.15	0.20	0.45	15.720995%	38/92	49/92	56/92
0	2	0.15	0.20	0.45	15.689998%	37/92	49/92	65/92

Figure 16: Scoring Metrics on Single-Target PCA + NN for Top 6 (Sorted based on Average Correct NN-Confidence)

M	E	ABOVE	HORIZ	NOSE	CONF	TOP1	TOP2	TOP3
1	1	0.10	0.25	0.50	14.094015%	42/92	50/92	55/92
1	1	0.15	0.20	0.45	15.438646%	39/92	51/92	56/92
0	1	0.10	0.20	0.40	14.237755%	39/92	47/92	55/92
1	0	0.15	0.20	0.45	15.720995%	38/92	49/92	56/92
1	2	0.15	0.20	0.45	15.642862%	38/92	49/92	56/92
0	2	0.10	0.20	0.50	15.002675%	38/92	50/92	61/92

Figure 17: Scoring Metrics on Single-Target PCA + NN for Top 6 (Sorted based on one-guess accuracy)

To see a how all 98 parameter sets performed, see Appendix A[11].

3.5.4 Results

This data we collected in this step actually served two purposes. First, the intended purpose, which was to give us indicators as to which parameter sets performed well, and which didn't. Second, this gave us a benchmark performance level for Single-Target PCA and Nearest

[11] Appendix A has an additional column titled "SKIP." This is discussed in Section 4.3.

Neighbor. In a way, his helped score both individual parameter sets as well as the entire PCA classification algorithm.

Before getting into which parameter sets we decided were the most-fitting, consider the big-picture results which we had already achieved: amongst nine targets, we were able to recognize the correct target (with one guess) **42.7%** of the time! Furthermore, we had another parameter set whose three-guess accuracy was **70.6%** (67/92 correctly classified test targets).

Anyway, now that we had done all of this testing, the goal was to select the top parameter sets and forget about the rest. By sorting the results based on the four metrics, we concluded that the following 5 parameter sets worked pretty well[12] and would be the only sets used for any further testing:

M	E	ABOVE	HORIZ	NOSE	CONF	TOP1	TOP2	TOP3
0	2	0.10	0.20	0.45	16.0422%	37/92	47/92	60/92
1	2	0.10	0.20	0.45	15.822206%	36/92	48/92	55/92
0	2	0.15	0.20	0.45	15.689998%	37/92	49/92	65/92
0	1	0.10	0.20	0.50	13.791333%	35/92	53/92	62/92
1	1	0.10	0.25	0.50	14.094015%	42/92	50/92	55/92
1	1	0.15	0.20	0.45	15.438646%	39/92	51/92	56/92

Figure 18: PCA-proven highest performing parameter sets chosen for normalization

The above six parameter sets are the only parameter sets used in any further testing, because they experimentally proved to perform the best.

Section 4: Illumination

[12] These parameter sets were chosen subjectively by also looking at the data presented in Figures 16 and 17, as well as sorting the results by two-guess and three-guess accuracy.

4.1 The Problem

One problem which image processing tools across all disciplines try to fight is that of illumination variance. We mentioned that normalized facial images of the same person should look similar, even in different settings (such as under different lighting). Unfortunately, this turned out to be difficult to achieve; we needed to find another way to combat the illumination problem. Specifically, the problem is that the pixels in two facial images of *different people* but in the *same lighting* are often more similar than the pixels in two facial images of *the same person* but in *different lighting*. In a sense, the recognition engine would be learning what kind of lighting the picture was taken in, not what the person's face looks like. The underlying reason for this is that in high ambient light, the pixels in a picture tend to approach white (RGB value of <255,255,255>) whereas in little ambient light the pixels tend to approach black (RGB value of <0,0,0>). Because these numbers are so far apart, pixel based learners have trouble finding real facial features under the mask of highly varying illumination.

Figure 19: Effect of varying illumination on the face

Another similar issue is that of color balance. Different cameras use different ratios of Red, Green, and Blue when producing images. Fortunately, we were able to deal with this easily by grayscaling the images before working with them. Dealing with illumination, however, wasn't so easy.

4.2 MATLAB Illumination Package

Before we dove into tackling the illumination problem, we wanted to first see whether or not variance in illumination was even affecting our results. In other words, if we were somehow to eliminate this problem, would our results improve significantly? It was brought to our attention that MATLAB had a built in Illumination package called INFace[13][14] which had the ability to normalize all images for illumination. If we could run this tool in the pre-processing before running our recognition software, then we'd be able to tell how our recognizer would do in a perfect world with no illumination. If the results were much better, then we'd know that illumination was indeed an issue we need to deal with. When we did this, the results came back inconclusive. We found lower (on average) recognition rates.

M	E	ABOVE	HORIZ	NOSE	CONF	TOP1	TOP2	TOP3
0	2	0.10	0.20	0.45	14.61889%	33/92	46/92	52/92
1	2	0.10	0.20	0.45	15.65333%	34/92	47/92	55/92
0	2	0.15	0.20	0.45	15.28442%	31/92	45/92	58/92
0	1	0.10	0.20	0.50	13.973133%	34/92	51/92	58/92
1	1	0.10	0.25	0.50	13.289593%	35/92	45/92	55/92
1	1	0.15	0.20	0.45	14.95732%	35/92	48/92	58/92

Figure 20: Results when pre-processed with MATLAB's INFace Illumination correction package (contrast with Figure 18)

However, when we looked at what MATLAB had done, it seemed as though the illumination correction didn't adjust the pictures very much. Because we were unsure as to how to interpret these results, we continued our search to solve the illumination problem.

[13] INFace was created by Vitomur Struc from the University of Ljubljana in 2009.
[14] The tool uses a single-scale-retinex algorithm with wavelet-based normalization tested on 128 by 128 pixel images.

4.3 Quick Fixes

After reading a few papers about illumination, we found that people had suggested two main "quick fixes" to this problem. We tried these quick-fixes, and found that the first helped greatly, but the second hurt out performance significantly.

4.3.1 Skipping Eigenvectors

The first quick-fix was specific to dealing with illumination in PCA-reduced images. Because our classifier was a PCA classifier, this fix was applicable. Specifically, the paper[15] suggested skipping the first three eigenvectors (principle components) when calculating the nearest neighbor in the recognition stage. The motivation behind this is that because PCA places the most prominent components with the highest eigenvalues, and since illumination is indeed very prominent amongst facial pictures[16], by eliminating the first few principle components (dubbed "illumination components") we are left working with only the true face data. We played around the number of eigenvectors to skip (as shown in the SKIP column of Appendix A), and found indeed that skipping three worked the best for us as well. **All the results shown up until now were calculated including this fix.**

4.3.2 Pixel Logarithms

The second fix, which was brought to our attention at the CyLab, was to take a logarithm of the pixels as a pre-processing step before working with them. The motivation behind this was that because of the decreasing slope of the logarithm function, true lower-valued pixels would be preserved whereas higher-valued (illuminated) wouldn't appear as different (in the logarithm

[15] See Reference #1

[16] Illumination will only be a major component if PCA is trained on a training set which has illumination variance. More about how we trained our PCA in Section 5.3.2

domain). When applying this transform, we achieved terrible results, with a one-guess recognition rate of no higher than 26/92 (as compared to the 42/92 originally) for any parameter set.

4.4 Artificial Illumination

4.4.1 Motivation behind artificial variants

One of the new ideas which we came up with during this project was the idea to construct artificially illuminated variants of facial images. The motivation behind this was two-fold. First, having multiple images of the same person could help our recognition learner[17]. Second, if we were able to realistically construct how the target face might look in different lighting, we would be able to solve the illumination problem!

4.4.2 Looking ahead: Linear Dependence

It was mentioned that having multiple samples of one target could potentially help the recognition engine. In our project, the Fisherface recognizer (built on LDA) was able to benefit from having many sample images of one target, but only **if the samples were linearly independent from one another**[18]. One naïve way to construct artificial variants would be by multiplying all of the pixels in the image by a parameter α. If $\alpha > 1$, the image would become brighter, and if $\alpha < 1$ the image becomes darker. Clearly, these variants are not linearly independent; in fact no two variants are linearly independent from each other. Another naïve idea would be to add constants $\{0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$ to all the pixels to generate n variants. However, no three of those variants will be linearly independent, as shown below.

[17] In fact, LDA (the second recognition learner we used) can **only** work if there is more than one sample per target image. More about LDA in Section 5.4

[18] See Section 5.4.1

Recall that three vectors $\boldsymbol{x}, \boldsymbol{y}$, and $\boldsymbol{z}$ are linearly independent if and only if there exist no constants a, b such that $a\boldsymbol{x} + b\boldsymbol{y} = z$. If each image has d pixels (or dimensions), and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ correspond to the first three variants, then $a = \left(1 - \frac{\alpha_2}{\alpha_1}\right), b = \left(\frac{\alpha_2}{\alpha_1}\right)$ shows that the images are not linearly independent.

$$\left(1 - \frac{\alpha_2}{\alpha_1}\right)\begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_d \end{bmatrix} + \left(\frac{\alpha_2}{\alpha_1}\right)\begin{bmatrix} p_1 + \alpha_1 \\ p_2 + \alpha_1 \\ \dots \\ p_d + \alpha_1 \end{bmatrix} = \begin{bmatrix} p_1 + \alpha_2 \\ p_2 + \alpha_2 \\ \dots \\ p_d + \alpha_2 \end{bmatrix} \tag{4.1}$$

Thus, we had to come up with a clever way to generate artificially illuminated variants whose pixel values are linearly independent from one another.

4.4.3 Gradient based Illumination

Our initial idea was to have a variable location light-source which would affect how the pixels in the facial image were illuminated. This would be quite accurate, because generally light can be modeled as coming from a point source. Unfortunately, we would need to be able to render the face in three dimensions and build a ray tracer, both of which were too difficult to implement this semester. Instead, we generated gradients **which varied spatially throughout the image** which would affect how a pixel "lit up." One such gradient had higher values in the left than the right, which would mean that there is a light source coming from the left of the picture. For illustrative purposes, a simple "left-high" gradient matrix is detailed below:

$$\begin{bmatrix} 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \\ 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \\ 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \\ 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \\ 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \\ 1.5 & 1.4 & 1.3 & \dots & 1.05 & 1 \end{bmatrix}$$

By performing a *saturated element-wise multiplication* between the above gradient matrix and the input facial image, we generated an illuminated version of the facial image which

would be linearly independent with other illuminate versions generated by different gradients. The saturated element-wise multiplication calculates a cell-by-cell product for each matrix element and rounds the value to the nearest integer in the range $[0, 255]$ (RGB value range). Formally,

$$a \otimes b = \begin{cases} 255 : ab \geq 254.5 \\ \lfloor ab + 0.5 \rfloor : otherwise \end{cases} \quad (4.2)$$

By choosing different orientations for the gradient matrices and by varying the magnitude of the values in the gradient matrix (corresponding to "how strong is the light source"), were able to come up with **40 artificially generated, linearly independent,** samples per target image.

Figure 21: Five artificially generated illumination variants. The first simulates a light source from the top, the second from the left, the third from the bottom, the fourth front-on, and the fifth from far away. The original image was the one pictured in Figure 4.

One caveat about this gradient-based artificial illumination is that the input facial image must contain very little other than the face itself. Because the gradient will adjust pixel values *throughout the image*, if the actual face only occupies a small portion of the image then there will be little differences in the variants. To combat this, we first ran face detection on a new target, cropped the image to include facial region, and then created the illumination variants.

Note that this illumination pre-processing step took place before the tilt correction and polygon cropping stage; the 40 target facial images for each individual were generated and then

each illuminated variant was normalized. However, because the facial landmarks lie in the same pixel locations in each image (regardless of illumination), the face detector need only run once per individual. Then, these normalized variants were passed to our two multi-target recognition learners (namely PCA+SVM and Fisherfaces). The results of this are discussed in Sections 5.7.2 and 5.7.3.

Section 5: Face Recognition

5.1 What is Face Recognition?

Recognition, in our case, is the classification problem of determining which target sample most closely matches the probe facial image. Throughout this section, we'll discuss three different algorithms and optimizations for making this decision. These algorithms will perform the same three-step procedure:

1) Reduce the dimensionality of each target image, and tag the new dimensions with an identifier for who that image is a picture of.
2) Reduce the dimensionality of the probe image.
3) Pass the set of tagged points in the reduced dimensionality space, as well as the reduced probe image, into a classifier.

The first algorithm reduced the dimensionality using Single-Target PCA, and then applied the Nearest Neighbor classifier. The second algorithm reduced the dimensionality using Multi-Target PCA, and then applied an SVM classifier. The third algorithm reduced the dimensionality using the Fisherface algorithm, and then applied a Nearest Neighbor classifier. The results of these three different recognition algorithms are discussed in Section 5.7.

5.2 Dimensionality Reduction

The concept of dimensionality reduction is crucial to many subjects in engineering fields because the complexity of a problem can increase exponentially when dealing with more dimensions. This phenomenon, given the nickname "the curse of dimensionality," forces higher-order systems to be reduced before any analysis can be performed. In other words, it's often useful to compress points in higher dimension space down to a lower dimension space, even at the cost of some information. This idea is known as dimensionality reduction.

Assume we had an algorithm which could reduce the dimensionality of a given set of data (and subsequently reconstruct the data) without losing too much information. It's obvious how such a tool can be used for something such as file compression. What's more interesting is how we can use a reduction tool for classification.

If we open up the black box of a dimensionality reduction tool, we see that it has to **intelligently select** which pieces of information it keeps, and which it doesn't. Consider a reduction tool which reduces points in $\mathbb{R}^n$ space to $\mathbb{R}^{\frac{n}{2}}$ space by simply selecting the first half of the coordinates, and reconstructs the original points by appending $\frac{n}{2}$ zeros to the reduced point). Although this will indeed reduce dimensionality, if the latter half of the original coordinates were all non-zero, then we've lost a lot of data. A good reduction tool would need to reformat the data and keep only **the most critical features** from the data set. Thus, we can use a dimensionality reduction tool as a feature selector. This is how it helps in our classification problem.

Consider a probe facial image and a set of target facial images which the probe will be matched against. Before reduction, the images may contain thousands of pixels (dimensions) of data. After reduction, the images may be reduced to far fewer dimensions **representative of features** (i.e. {"no-glasses", "dark skin tone", "no facial hair"}). Now, working in this reduced dimension space where each dimension means much more than a single pixel, we can pass the tagged targets and probe to a classifier

and hope to achieve better results. This is the inspiration behind face recognition with dimensionality reduction.

We used two dimensionality reducers: one built on PCA and the other on LDA. Although there are dimensionality reducers which take into account soft biometrics (such as the "hair color" and "glasses"), the ones we used found features which are harder to describe in English. Instead PCA finds the features corresponding to the strongest numerical principal components, and LDA finds the features which maximize the differences between the targets.

5.3 PCA

Principal component analysis (PCA) is one of the most common dimensionality reduction algorithms with a wide variety of uses. We don't discuss the details of the actual algorithm itself here (PCA is a very well-known algorithm, one good explanation of it can be found in Reference #3), but instead discuss our implementation and training set. PCA works by finding the top eigenvectors of the covariance matrix of the training data (after mean subtraction) corresponding to the highest eigenvalues. If you use more eigenvectors, you achieve a higher reconstruction ratio at the expense of lower compression. **Our program had a hard-coded reconstruction ratio of 0.95, yielding between 27 and 34 eigenvectors [reduced from 4800 pixel image]**. This reconstruction ratio was suggested by members at CyLab. We implemented the entire PCA routine (mean/eigenvector calculation, reconstruction, and reduction) in VC++ using OpenCV's `cvSVD` routine for eigenvector calculations.

5.3.1 Training Database

The database which we used for training the PCA eigenvectors was given to us by the CyLab. Specifically, we were given 250 facial images (of arbitrary human faces) from the FRGC database which we trained on. Recall that because PCA would work to reduce normalized

images, we normalized these images (using the different settings) before running the PCA algorithm.

5.4 LDA

Linear Discriminant Analysis (LDA) is another well known algorithm for dimensionality reduction. The implementation of LDA is slightly more involved than PCA, and is described in the Fisherface paper (Reference #1). A key distinction between LDA and PCA is that the projections which LDA makes **vary based on the targets to classify against**. For PCA, the eigenvectors and means were calculated solely based on the training database. For LDA, the Wilson vectors are generated based on the database of targets. Practically, this means that anytime I added or deleted a target in the application, I would need to recalculate the Wilson vectors. We implemented, from scratch, a multi-class LDA reduction algorithm. The most computationally difficult part of this algorithm was an eigenvector calculation. There was no OpenCV routine which was capable of finding the eigenvectors of an asymmetric matrix, and because the Wilson vectors of LDA are the eigenvectors of such a matrix, we had to spawn MATLAB (with a script) to perform this one calculation. This was the only place in the application that ran in MATLAB.

5.5 Fisherfaces

Unfortunately, LDA requires the dimensionality of the input to be bounded by the difference between the number of targets and the samples per target. Because an image would typically have the dimensionality in the thousands (and we were running on less than 100 targets), we weren't able to use LDA as is. Instead, we performed PCA reduction on the faces first and then fed those results to our linear discriminator. This process of combining PCA and LDA on facial images is known as Fisherface recognition.

5.6 Classifiers

5.6.1 Nearest Neighbor Classifier

The nearest neighbor classifier takes a set of target points in $\mathbb{R}^n$ and another probe point in $\mathbb{R}^n$ and simply finds which of the target points has the minimum Euclidean distance to the probe point. The confidence of any given match was given as the ratio of "inverse distance." For example, if the probe image was 1 unit away from A, 4 units away from B, and 5 units away from C, then the confidences are:

$$A' = \frac{1+4+5}{1} = 10$$

$$B' = \frac{1+4+5}{4} = 2.5$$

$$C' = \frac{1+4+5}{5} = 2$$

$$Conf(A) = \frac{A'}{A'+B'+C'}, Conf(B) = \frac{B'}{A'+B'+C'}, Conf(C) = \frac{C'}{A'+B'+C'}$$

5.6.2 SVM

We were also given, by CyLab, a linear kernel SVM classifier which we used in conjunction with the artificially generated illumination variants. However, the SVM classifier wasn't able to give confidences with its selection, and only gave one selection (the top pick).

5.7 Final Results

5.7.1 Single Target PCA + NN

The results of this are attached in Appendix A. The highlight was a maximum one-guess recognition rate of 42.7% and three-guess recognition rate of 70.6%

5.7.2 Multiple Target PCA + SVM

M	E	ABOVE	HORIZ	NOSE	CORRECTNESS	PERCEN
0	2	0.10	0.20	0.45	34/92	37.0%
1	2	0.10	0.20	0.45	36/92	39.13%
0	2	0.15	0.20	0.45	33/92	35.9%
0	1	0.10	0.20	0.50	35/92	38.0%
1	1	0.10	0.25	0.50	41/92	44.6%
1	1	0.15	0.20	0.45	39/92	42.4%

We found that the multiple targets (artificially illuminated faces) with an SVM classifier had roughly the same performance as the single targets with a nearest neighbor classifier. The highlight was a 44.6% recognition rate.

5.7.3 Fisherfaces

M	E	ABOVE	HORIZ	NOSE	CONF	TOP1	TOP2	TOP3
0	2	0.10	0.20	0.45	14.53832%	33/92	41/92	55/92
1	2	0.10	0.20	0.45	14.19486%	32/92	43/92	54/92
0	2	0.15	0.20	0.45	13.99008%	35/92	45/92	56/92
0	1	0.10	0.20	0.50	13.30584%	37/92	50/92	60/92
1	1	0.10	0.25	0.50	12.30449%	35/92	44/92	55/92
1	1	0.15	0.20	0.45	13.62686%	35/92	44/92	55/92

The highlight was that the maximum one-guess recognition rate was 40.2% by the third fourth parameter set. This same parameter set achieved a three-guess recognition rate of 65.2%, which was also the highest performance we saw from the Fisherfaces.

Section 6: Entry Lookup

The final piece of the puzzle was retrieving information on a probe image once I knew who it was. This was implemented by running a background Perl script on an Andrew machine which would listen for requests. The protocol we defined had the script listen for an Andrew ID and spit back any information it could find on that individual. The information the script picked up was anything in the Andrew Directory (programmatically accessible via the `finger` utility). The Perl script was linked with the VC++ application through TCP/IP Socket communication. This simple 50-line script could be beefed up to scour the web, Facebook, and other information sources and provide a much more detailed description of the person. However, as the focus of the project was face recognition, we chose not to devote more time to the Entry Lookup stage.

Section 7: Computation and Application

7.1 DSK vs. PC

Our group, with permission from the instructors, did not use the DSK for any computation during this project. However, we did diagram which parts of the code could be run on the DSK and which would need to run on the PC. Note that the diagram below is specific for the Single-Target PCA + NN recognizer. Any computation that didn't require OpenCV or eigenvector calculations could be thrown onto the DSK. In practice, it would make sense to do the training (PCA mean and eigenvectors) pre-runtime and simply download those values onto the DSK during runtime.

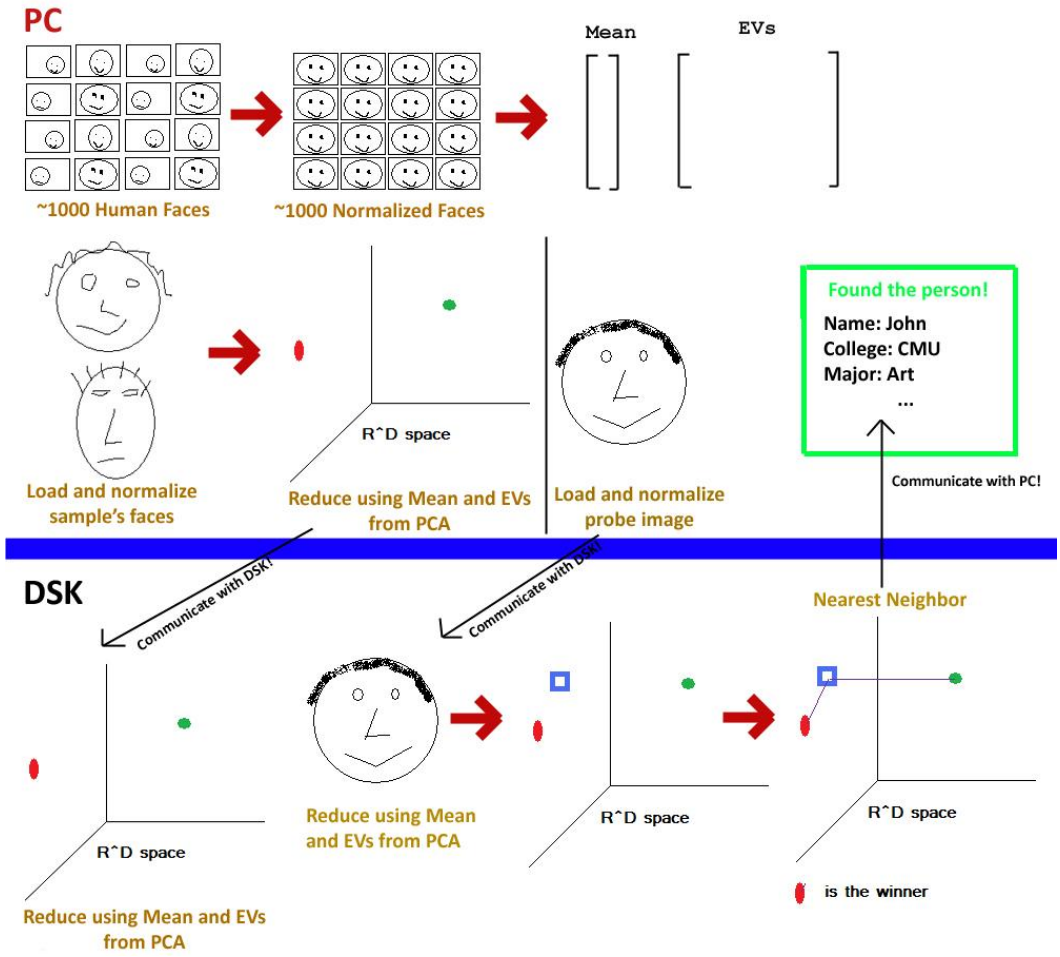


Figure 22: DSK/PC split up

7.2 GUI and Demo

The entire application we developed was written in VC++ (using VS2008). No MATLAB was used except for one eigenvector calculation[19]. During our demonstration, we had our program take a target picture of the students and faculty members, and then recognize them based on a different photograph. Although we didn't take down exact statistics during the demonstration, we exhibited **far better accuracy rates** during the actual demonstration than on the testing data. This can most probably be explained because pictures on Facebook have a tremendous amount of pose and illumination variation, whereas during the demo all pictures were taken indoors and under similar circumstances. Note that all PCA training was done prior to demonstration. Some screenshots of the GUI follow:

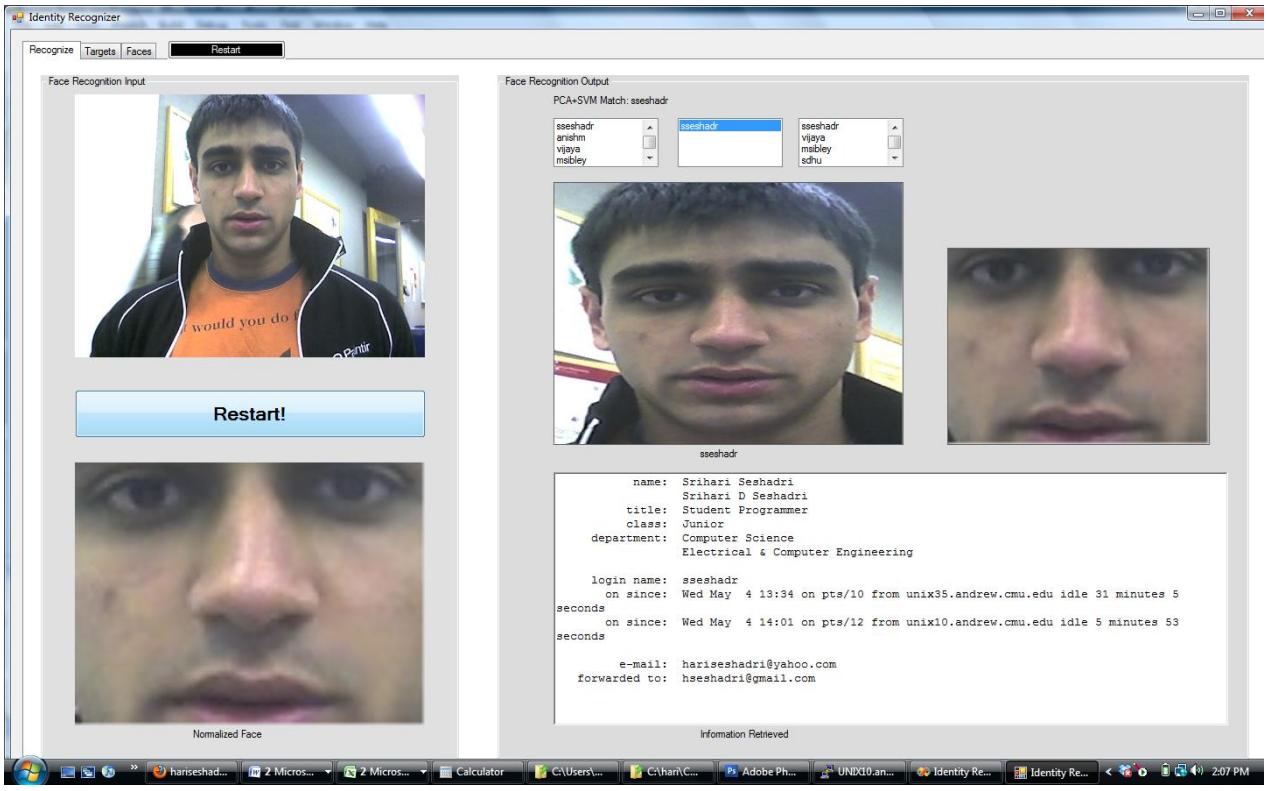


Figure 23: GUI Main Screen

[19] We found no way to find the eigenvectors of an asymmetric matrix in OpenCV (See Section 5.4). For that, we had to spawn a process to run "`matlab -r …`". No MATLAB was used anywhere else in the application.

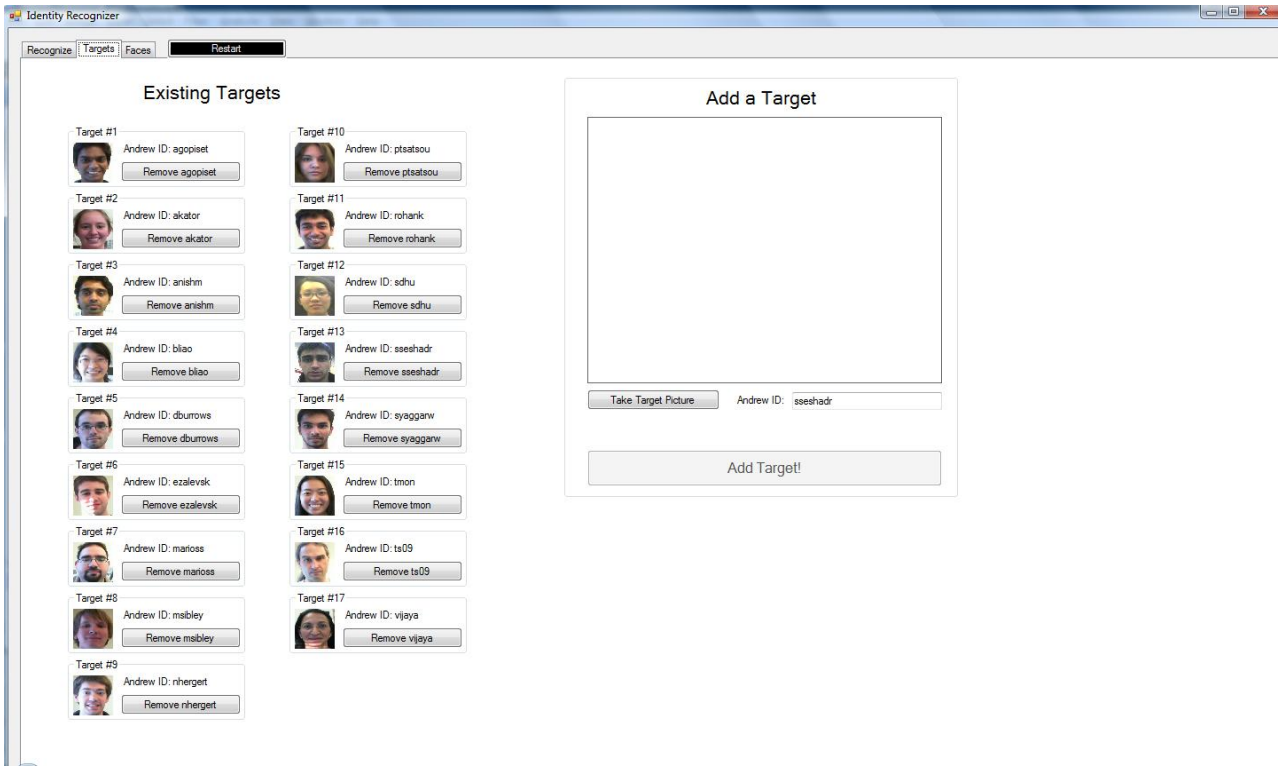


Figure 24: GUI Targets Screen

Section 8: Schedule

We had set forth a schedule at the start of our project detailing when we would reach the various milestones of the project. Overall we kept to the schedule pretty well, finishing things earlier rather than later. One major deviation concerned face detection. Originally, we had planned on leaving out face detection and assuming that the input image was a pre-cropped face. However, since we received a face detection binary, we decided to incorporate the detection into our program. Another deviation was expediting the implementation of classifiers (such as minimum distance). Because we couldn't actually select the normalization parameters without a functional classifier, we had to get the nearest neighbor classifier working much before the time we anticipated.

Date	Objective
02/18	Have chosen a technology to implement the GUI
02/25	Ability to talk to the Webcam, Entry Lookup Utility coded
03/18	PCA Working. Ability to load a picture taken from the webcam, display in GUI, and send it to the DSK
03/25	LDA Working. Can listen for feedback from the DSK matching the picture to a name (or andrew ID)
04/08	Minimum Distance Detector working. Interface with Data Searching Utility. Can listen for and parse feedback in the form of <attribute, value> pairs from search utility and render them in GUI
If we have time	Allow user to crop the face of the input image to select the face

Figure 25: Original Schedule

The true schedule was something like the itinerary below (note that no work was actually done with the DSK):

Date	Objective
02/18	Chose VC++ (VS2008) to write the GUI.
02/25	Entry Lookup Working. All Webcam driver issues were resolved.
03/18	PCA Working, Minimum Distance Detector working. Normalization parameter selection was in process
03/25	LDA Working (but buggy). Began research on Fisherfaces
04/08	Interfaced properly with Entry Lookup Utility. Fixed bug with LDA and began testing with SVMLight.
Rest of the time	UI Design and debugging

Figure 26: Revised Schedule

Section 9: Future Work

9.1 Multiple Face Recognition

Would it be possible to process images with multiple faces? The first limitation is that the face detector only finds one face (at most) per image. However, we can build a multiple face detector given the single face Viola-Jones detector by repeatedly running the latter and etching out the discovered face until no more faces are found. This would could be used to perhaps identify groups of people and tag images on Facebook.

9.2 MACE and other learners

Although we touched on quite a few recognition algorithms, there are many that remain unmentioned. One set of learners are correlation-filter based classifiers, such as MACE. Another set are Bayesian learners. Implementing additional classifiers and comparing their performance, in the context of face recognition, to ones discussed before would be enlightening.

9.3 Soft Biometric Tagging

Soft biometrics are real, human features which can be extracted from an image. For instance, the eyebrow color and facial hair formations could be extracted from a facial image. Gender, to an extent, can be learned as well. By tagging each target with a set of soft biometric data, one might be able to increase accuracy. A very basic soft biometric learner might have a rule such as "never classify an individual with black eyebrows as an individual with blonde eyebrows, regardless of how similar the rest of the face is."

9.4 Illumination Filtering

Although our group spent a great deal of time fighting illumination variance, one approach which we did not take was processing illumination in the frequency domain. Because illumination tends to manifest itself in the lower frequencies (relative to the frequency content of the rest of the face), it would make sense to high-pass-filter all the images before feeding them into the recognition learner.

9.5 Facebook

9.5.1 Target Crawling

If we were able to improve the accuracy of the application, it would be an interesting tool which could be used to identify people in a crowd (that might potentially be your friends). In another application, the program would use the Facebook profile pictures of anyone who you have mutual friends with to identify an unknown person as a friend of a friend.

9.5.2 Entry Lookup Enhancement

One big restriction to the functionality of our project is that it can only lookup information for users in the Andrew Directory. Although it worked well for the demo, there are many ways that one could improve the Entry Lookup stage (Section 6). One highly structured, programmatically accessible, data source which we didn't tap into is the information on Facebook (accessible through the Facebook API). By linking up with this API, we could not only bring the user's graduating class and major, but also a list of their friends, hometown, and relationship status!

Section 11: References

We'd like to acknowledge the help given to us by the CyLab group, for their face detection Windows binary. Also, we'd like to acknowledge the creators of SVM light for their help as well.

1) P. Belhumeur, J. Hespanha, D. Kriegman- Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection [paper]
2) P. Viola, M. Jones: Robust Real-time Object Detection [paper]
3) Marios' PCA and LDA Lecture slides

Appendix A: Performance of Single-Target PCA + NN with variable parameters

TOPx has the value 0<=y<=1 where y is the fraction of targets (out of 92) correctly classified in the top x guesses

H	W	M	E	ABOVE	EYE	NOSE	SKIP	CONFIDENCE	TOP1	TOP2	TOP3
60	80	1	1	10	25	50	3	14.094015	0.456522	0.543478	0.597826
60	80	1	1	15	20	45	3	15.438646	0.423913	0.554348	0.608696
60	80	0	1	10	20	40	3	14.237755	0.423913	0.51087	0.597826
60	80	1	0	15	20	45	3	15.720995	0.413043	0.532609	0.608696
60	80	1	2	15	20	45	3	15.642862	0.413043	0.532609	0.608696
60	80	0	2	10	20	50	3	15.002675	0.413043	0.543478	0.663043
60	80	1	2	10	25	50	3	14.820762	0.413043	0.51087	0.608696
60	80	0	2	15	25	40	3	14.807936	0.413043	0.532609	0.652174
60	80	0	2	20	20	40	3	14.60041	0.413043	0.51087	0.608696
60	80	0	1	10	25	40	4	14.489232	0.413043	0.51087	0.608696
60	80	0	1	10	25	40	3	14.098663	0.413043	0.48913	0.586957
60	80	1	1	15	20	50	3	13.618634	0.413043	0.543478	0.597826
60	80	0	1	10	25	45	4	13.362928	0.413043	0.521739	0.554348
60	80	0	1	10	20	40	4	13.332061	0.413043	0.48913	0.586957
60	80	0	2	10	20	45	3	16.044223	0.402174	0.51087	0.652174
60	80	0	2	15	20	45	3	15.689998	0.402174	0.532609	0.706522
60	80	0	2	15	20	40	3	15.460188	0.402174	0.543478	0.630435
60	80	1	2	15	25	40	3	15.026397	0.402174	0.5	0.663043
60	80	1	1	10	25	45	3	14.376659	0.402174	0.48913	0.597826
60	80	1	2	15	20	50	3	13.874504	0.402174	0.532609	0.608696
60	80	0	2	15	20	50	3	13.858735	0.402174	0.532609	0.630435
60	80	0	1	15	20	40	3	13.855999	0.402174	0.48913	0.597826
60	80	1	2	10	20	45	3	15.822206	0.391304	0.521739	0.597826
60	80	1	0	10	20	45	3	15.821445	0.391304	0.521739	0.597826
60	80	1	1	10	20	45	3	15.816855	0.391304	0.521739	0.597826
60	80	0	2	10	20	40	3	15.270896	0.391304	0.5	0.619565
60	80	1	2	10	25	45	3	15.1637	0.391304	0.5	0.597826
60	80	1	1	10	20	50	3	15.066255	0.391304	0.576087	0.641304
60	80	1	2	10	20	50	3	15.017933	0.391304	0.554348	0.619565
60	80	1	0	15	25	40	3	14.827169	0.391304	0.5	0.641304
60	80	1	2	15	25	45	3	14.819914	0.391304	0.521739	0.586957

60	80	1	1	15	25	40	3	14.768444	0.391304	0.543478	0.619565
60	80	0	2	25	20	40	3	14.699147	0.391304	0.51087	0.619565
60	80	0	1	10	20	45	3	14.353811	0.391304	0.521739	0.597826
60	80	1	1	15	25	50	3	13.382205	0.391304	0.51087	0.619565
60	80	1	2	15	20	40	3	15.454376	0.380435	0.565217	0.652174
60	80	1	0	15	20	40	3	15.442279	0.380435	0.565217	0.652174
60	80	0	2	10	25	40	3	14.972031	0.380435	0.543478	0.663043
60	80	0	2	10	25	50	3	14.945505	0.380435	0.554348	0.641304
60	80	1	0	10	20	50	3	14.896721	0.380435	0.554348	0.619565
60	80	0	2	20	25	40	3	14.439614	0.380435	0.51087	0.608696
60	80	1	1	15	25	45	3	14.311139	0.380435	0.48913	0.597826
60	80	1	0	20	20	40	3	14.011296	0.380435	0.543478	0.576087
60	80	0	1	10	20	50	3	13.79133	0.380435	0.576087	0.673913
60	80	0	1	10	25	50	4	13.039233	0.380435	0.51087	0.597826
60	80	1	0	25	20	40	3	15.395019	0.369565	0.565217	0.641304
60	80	1	1	15	20	40	3	15.366336	0.369565	0.554348	0.652174
60	80	1	0	10	25	40	3	14.48655	0.369565	0.565217	0.641304
60	80	0	1	15	20	40	4	14.185993	0.369565	0.456522	0.565217
60	80	1	1	20	25	40	3	13.177512	0.369565	0.5	0.586957
60	80	1	1	20	20	40	3	13.160648	0.369565	0.48913	0.532609
60	80	0	2	25	20	45	3	14.01726	0.362637	0.527473	0.626374
60	80	1	0	20	20	45	3	15.13748	0.358696	0.48913	0.576087
60	80	1	2	10	20	40	3	14.774276	0.358696	0.543478	0.619565
60	80	1	0	10	20	40	3	14.771306	0.358696	0.543478	0.619565
60	80	1	1	10	20	40	3	14.763526	0.358696	0.543478	0.619565
60	80	0	1	10	20	50	4	13.615929	0.358696	0.576087	0.630435
60	80	0	1	10	20	45	4	13.603153	0.358696	0.456522	0.565217
60	80	0	1	15	20	45	3	13.521976	0.358696	0.5	0.576087
60	80	0	1	15	20	45	4	13.353715	0.358696	0.48913	0.576087
60	80	0	1	10	25	50	3	13.352289	0.358696	0.5	0.597826
60	80	1	1	25	20	45	3	13.434397	0.351648	0.527473	0.648352
60	80	1	1	10	25	40	3	14.455024	0.347826	0.554348	0.652174
60	80	1	2	10	25	40	3	14.442399	0.347826	0.554348	0.652174
60	80	0	2	20	20	45	3	14.252526	0.347826	0.48913	0.619565
60	80	1	0	15	20	50	3	13.650084	0.347826	0.521739	0.608696

60	80	1	0	15	25	45	3	13.544127	0.347826	0.48913	0.565217
60	80	0	1	10	25	45	3	13.219519	0.347826	0.478261	0.554348
60	80	1	1	20	25	45	3	11.577401	0.347826	0.445652	0.521739
60	80	0	2	20	20	50	3	13.796469	0.340659	0.516484	0.637363
60	80	1	0	25	20	45	3	13.722169	0.340659	0.571429	0.637363
60	80	0	2	15	25	45	3	14.721966	0.336957	0.51087	0.663043
60	80	1	0	10	25	45	3	14.180597	0.336957	0.51087	0.576087
60	80	0	2	25	25	40	3	14.033678	0.336957	0.478261	0.619565
60	80	0	2	15	25	50	3	14.469914	0.326087	0.521739	0.641304
60	80	1	0	25	25	40	3	13.751245	0.326087	0.434783	0.543478
60	80	1	1	25	20	40	3	13.552581	0.326087	0.456522	0.543478
60	80	1	0	10	25	50	3	13.175638	0.326087	0.456522	0.576087
60	80	1	1	20	20	50	3	13.422553	0.318681	0.483516	0.626374
60	80	0	2	10	25	45	3	14.863969	0.315217	0.467391	0.630435
60	80	0	2	20	25	45	3	13.806532	0.315217	0.456522	0.597826
60	80	1	1	20	20	45	3	12.325194	0.315217	0.5	0.576087
60	80	0	2	25	25	45	3	14.539412	0.307692	0.505495	0.604396
60	80	0	2	20	25	50	3	13.9494	0.307692	0.461538	0.604396
60	80	1	1	20	25	50	3	12.698827	0.307692	0.450549	0.56044
60	80	1	0	20	25	40	3	13.465286	0.304348	0.51087	0.630435
60	80	1	0	20	25	45	3	12.557279	0.304348	0.423913	0.543478
60	80	1	0	15	25	50	3	12.106322	0.282609	0.456522	0.5
60	80	1	0	20	20	50	3	13.026812	0.252747	0.43956	0.582418
60	80	1	0	25	25	45	3	12.604094	0.252747	0.395604	0.516484
60	80	1	1	25	25	40	3	12.413975	0.25	0.434783	0.51087
60	80	1	1	25	20	50	3	12.869922	0.24359	0.282051	0.461538
60	80	1	0	20	25	50	3	11.932245	0.21978	0.340659	0.428571
60	80	0	2	25	20	50	3	13.195505	0.217949	0.282051	0.461538
60	80	0	2	25	25	50	3	11.712359	0.205128	0.307692	0.346154
60	80	1	0	25	20	50	3	10.731205	0.141026	0.205128	0.282051
60	80	1	0	25	25	50	3	10.062284	0.128205	0.192308	0.269231