PICTURE PERFECT

18-551 FALL 2009

GROUP 4

Ajay Ghadiyaram (aghadiya@andrew.cmu.edu)
Minwoo Jeon (minwooj@andrew.cmu.edu)
Stelios Melachrinoudis (smelachr@andrew.cmu.edu)
Nihal Parekh (naparekh@andrew.cmu.edu)

Introduction

We have designed and implemented a facial feature recognition program, specifically addressing the eyes and mouths, which has a low probability of error detection. The system will be implemented on a Texas Instruments C67 Digital Signal Processing (DSP) Board. This report will explain in detail how we have attempted to implement this system.

Problem

Group photographs have always been marred with blinking eyes or mistimed smiles because of the auto timer mode. The mode just counts down without regard to whether the people posing are ready or not. The result ends up being that the photographs are deemed unacceptable to publish or showcase to others. With many pictures being taken with eyes closed and mouths that are in mid-smile, it is often frustrating and time consuming to take a satisfactory group photo.

Also it is extremely difficult to objectively classify a photo as good or bad based on features such as open/closed eyes and smiling/not smiling. For example it could be an issue of defining what a smile is. Different people have different ways of smiling where some could not be considered as a smile by some. Even something that may seem straightforward as determining if a subject’s eyes are open or closed can get tricky depending on the resolution of the camera and how that individual tends to smile. For example, Asians tend to have eyes that appear closed when they smile even though they are very much open. Training an algorithm to classify mouths as smiling and not smiling or eyes as open or closed separately may be extremely difficult in itself.

The algorithms and methods of analysis of our project can also further be applied in areas that go beyond simply discriminating between good and bad group photographs. Face detection and feature classification are used in human computer interaction studies where computers are used to detect human emotions. Over the years fairly successful methods have been developed that can also be used for purposes of analyzing photos as well.

Solution

The application we are attempting to address is related to image processing in that we are trying to analyze features of the face such as eyes and mouths. We define a photograph to be “good” if and only if everyone's eyes within the picture are open and everyone is smiling, otherwise it is a “bad” photograph. We propose an implementation that will do a better job of detecting eyes and mouths and integrating it into photography. We intend to detect faces by using the Viola-Jones algorithm. Unlike other previous projects, our project requires the detection of multiple faces in a photo.

Once the faces are detected they are cropped, scaled, and stored as grayscale face images to be processed for eye and mouth detection.

Afterwards, we intend to utilize a combination of linear discriminant analysis (LDA) and principal component analysis (PCA) to discriminate the two classes for eyes (open/closed) and for mouths (smiling/not smiling). The PCA is used to reduce the dimensionality of the stored grayscale image vectors that were obtained through the Viola-Jones face detection algorithm. The PCA results give a vector projection that best relates the class. So in our case, the PCA helps define the two different classes of faces in the picture: good or bad. LDA is then used to classify the different faces by creating a line that best separates the two different classes. Thus after training the PCA/LDA algorithm with the faces obtained from our database, we obtain data that classifies different faces depending on the position of the facial analysis in comparison to the line created from the LDA.

Novelty

Previous work in the 551 course has been done in regards to face detection and eye detections. In particular our group's project idea can be most related to that of Spring 2004 Group 10's "Sleepy Head, EYE Can See You" idea. The group used a database that they created with various images of eyes opened and closed taken at different angles to detect and analyze the eyes of a "driver" to prevent accidents caused by sleeping on the wheel. The images were cropped, scaled, and changed to grayscale after which they would use Sobel Edge detection in the preprocessing process. Then a distortion invariant filter was used to detect the eyes on the DSK. Finally the results would be fed back into the computer GUI where one could see the accuracy of detecting "sleepy drivers." In addition to the Spring 2004 Group 10's project, a couple other referenced projects were:

- Spring 2004 Group 10: Sleepy Head, EYE Can See You!
    - Distortion invariant filter to detect eyes implemented on DSP
- Spring 2004 Group 7: Iris Recognition for Biometric Analysis
    - Discrete Cosine Transform (DCT) for iris recognition implemented on EVM
- Spring 2003 Group 2: Face Verification for ATM Access
    - FFT-based face verification with real-time implementation on DSP

Our project uses the idea of determining whether a subject's eyes are open or closed, but goes about a different method of analysis. First of all our project not only needs to do analysis on the eyes, but also has to be able to analyze the mouth of the detected face to provide meaningful results. Also in comparison to only analyzing one face per image, our project is responsible for detecting multiple faces within a single photo and run tests on all faces to classify a photo as good or bad.

The database being used is also different from the previous projects in that we created our own database called MANS (Minwoo Ajay Nihal Stelios). The MANS database was used to train and test our face detection and good/bad photograph classifier. Overall the algorithms are different in doing analysis on the detected facial features. In comparison to using edge detection and distortion invariant filters, our project uses a combination of principal component analysis (PCA) and linear discriminant analysis (LDA) to produce a trained algorithm to discriminate between classes of good and bad photos. More detail on the algorithms and database are outlined in the rest of the paper.

Database

The OpenCV Viola-Jones algorithm is already tested and trained to detect faces from various images and thus does not need a separate training set for our purposes. We simply implemented the OpenCV code to detect faces in photographs that we would then classify using our classifier algorithm.

Training Set

The database that we used for training our classifier algorithm is a combination of frontal portrait face images from multiple databases: Bio Imaging, MIT-CBCL Face Database, Japanese Female Facial Expression (JAFFE) Database, and AT&T “The Database of Faces.” We named the new facial database the Minwoo Ajay Nihal Stelios (MANS) database. The reason for creating a new database using a combination of images from multiple databases was to ensure a wide variety of faces with eyes that are open and closed, mouths that are smiling and not, and have also have a good mix of gender and ethnicity.

The training set for the PCA/LDA algorithm consists of images from the MANS database, which contains a total of 60 facial images. The training set images are grayscale images that are cropped and scaled to just contain frontal portrait shots. The training set is chosen to be grayscale to reduce unnecessary memory and space used to store and analyze color images. Instead of having three values per pixel on the RGB scale, we can reduce dimensions by running an rgbtogrey function in Matlab. The images in the training set database consist of various ethnicities and images of eyes open/closed and mouths smiling/not smiling as described previously.

The training set and testing set are separated within the PCA/LDA algorithm code. By default a given database is separated 60/40 for training and testing. So if there are 100 images in a database, 60 images are used for training the algorithm to produce the vector projection used for classification and the remaining 40 images are used for testing the produced classifier. This is to ensure enough images are being trained and tested to reflect accurate results to be used for analysis.

Once the accuracies obtained from the classifier algorithm using PCA and LDA, a user is able to modify the ratio to train 100% of the images to test brand new images on the algorithm. By using all the images in the MANS database for training, we are able to have even more data points projected on the eigenspace plane and increase the accuracy of a relevant neighbor to be used to classify a given facial image.

Testing Set

The PCA/LDA algorithm testing set consists of the remaining 40% of the images in our MANS database as stated above.

The actual testing set for our picture classifier algorithm, which consists of the Viola-Jones face detection algorithm and PCA/LDA feature classifier algorithm, is tested using group photographs taken from the web. Various color photographs of multiple individuals are taken from Google, Picasa, Facebook, and other resources to build our testing set to test other lighting and environmental conditions. These images are all chosen at random ranging from family portraits to student organization group photos. The testing set consists of photos with subjects varying in ethnicity, gender, and age to ensure that tests encompass all possible cases.

Algorithms

The three main algorithms that will be used for our project are Viola-Jones, Principal component Analysis (PCA) and Linear Discriminant Analysis (LDA).

1. Viola-Jones [3]

Procedure to scan a sub-window capable of detecting faces across a given input image

- Scale invariant detector: Turns the original image into an integral image, i.e. if image is MxN, then $P_{xy} = \sum_{m=1}^{x}\sum_{n=1}^{y} \mathbf{p}_{mn}$, where p_{mn} is the value of the pixel (in grayscale) at column m, row n, and P_{xy} is the value of the pixel calculated for the integral image at column x, row y (all relative to upper left corner). If it is in color, then the value of the pixel will be determined by the norm of the vector formed.
- Adaboost Algorithm: A machine learning boosting algorithm capable of constructing a strong classifier through a weighted combination of weak classifiers
- Cascaded classifier (key to our project): Scan the detector many times through the same image—each time with a new size.
    - **Key point: "Instead of finding faces the algorithm should discard non-faces."** [3]

2. Principal Component Analysis (PCA) [4]

We perform the training for the classification on the PC in MATLAB using the function train_pca.m, which trains the original 'eigenface' method using Principle Component Analysis (PCA). PCA usually involves calculating the covariance matrix, which is extremely large if training on a lot of images, so we get around that by using Singular Value Decomposition (SVD) instead. 52 eigenfaces (eigenvectors) are used.

We then remove the mean of all the training images from each of the individual training images. We then perform SVD. This will come up with a square unitary matrix, U. Afterwards, we create a matrix by taking the top eigenvectors as eigenfaces (using the default setting of 100) on which we will apply the weights to the eigenvectors to ignore the first 10 eigenfaces.

3. Linear Discriminant Analysis (LDA) [6]

Linear Discriminant Analysis, otherwise known as the Fisherface algorithm does quite well as a classifier, although it is more computationally expensive. The trick to this method is to reduce the dimensionality of the images first using PCA and then perform classifications based upon the weights applied to the eigenvectors.

The first step is to calculate the number of classes, which in our case is 2 – good or bad (picture). We then obtain a vector from the unitary matrix U that we had gotten from SVD by disregarding the top 10 eigenvectors. We once again do SVD to obtain a new separation vector. Next, we calculate the mean for each class. This will allow us to calculate the between-class scatter matrix, Sb and the within-class scatter matrix, Sw. These scatter matrixes will be multiplied with the principal component weights and its transpose to project both the Sb and Sw into subspace. Then, we will find the generalized eigenvalues and eigenvectors of these projections.

4. Nearest Neighbor Classifier

The classifier algorithm performed on the detected faces projected on the eigen subspace uses a nearest neighbor approach. Each training set face is projected using the PCA and LDA algorithms as a point with x and y values. This means that there are ideally four distinct clusters of points on a plane to compare to. The face detected from Viola Jones is projected on using the same algorithms as the training set images.

The distance between the testing image and all the training images are found using the Euclidean distance formula. The Euclidean distance formula shown below computes the distance between two points by taking the square root of the sum of all the squared differences in x-values and y-values.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \cdots + (p_n-q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i-q_i)^2}.$$

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2}. \qquad (1)$$

Euclidean distance formulas [10]

Once the Euclidean distances have been calculated between the testing value and the training values, the smallest value is searched. The smallest value represents the shortest distance between two points, the testing image and the training image that most resembles it. This method of analyzing and classifying an object using Euclidean distance “neighbors” is referred to as the k-nearest neighbor algorithm. K refers to the number of neighbors to take into consideration when classifying an unknown. This can be easily explained using the diagram below.

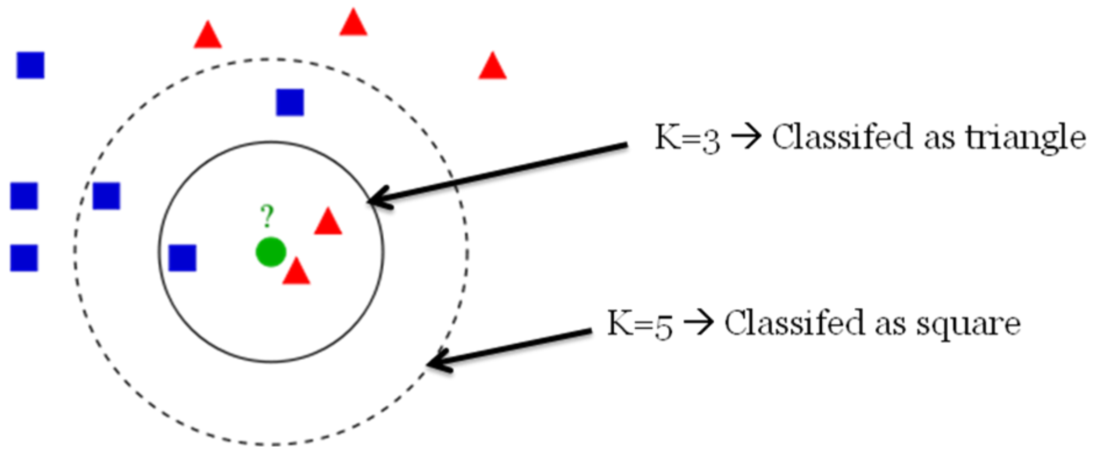


Figure 1. K-nearest Neighbor Algorithm Diagram w/ k=3 and k=5 [9]

The above diagram shows the green circle as the unknown image to be classified. In our case this would be the testing image that is obtained by running the Viola-Jones algorithm on our photograph. In the case where k=3, the object is classified using its three nearest neighbors. It shows that the three nearest neighbors consist of two red triangles and one blue square. Thus when k=3 for the above diagram, the object would be classified as a red triangle. When k=5, however, the five nearest neighbors are three blue squares and two red triangles so the object is classified as a square. So for this project, the blue squares can be seen as one class (eyes open and mouth smiling) and the red triangles can be seen as another class (eyes closed and mouth not smiling). The classifier algorithm would then look at the nearest neighbors like described above to classify the unknown face.

In our case, we stuck with the traditional k=1, where we classify the testing face image using the single nearest neighbor training face image’s class. So if the unknown face’s nearest neighbor on the plane is a data point that refers to a training image with class 1 (eyes open and mouth smiling) then the unknown face is classified as class 1.

Implementation

Step 1: Initial Training Set Development

The first step we will take is to form a training set from the MANS face database. This database consists of a collection of face images from the web. This training set will allow us to implement the Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) algorithms. These algorithms will be needed to distinguish the two classes of eyes (open/closed) and mouths (smiling/not smiling).

The pictures we obtained had to be classified to three different classes, described in the table below.

Eyes Open + Mouth Smiling (Good)	1
Eyes Closed + Mouth Smiling (Bad)	2
Eyes Open + Mouth Not Smiling (Bad)	3
Eyes Closed + Mouth Not Smiling (Bad)	4

Table 1. Database classifications (Good and Bad)

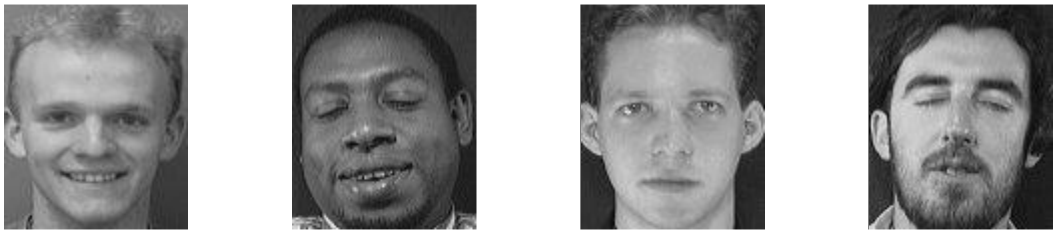


Figure 2. Examples of Classified Faces (Left: 1, Middle-left: 2, Middle-right: 3, Right: 4)

With images classified into separate classes using the method described above, the PCA/LDA algorithms are trained and tested to analyze whether the faces of the photo are “good.” This process can be somewhat time consuming and subjective because each photo has to be looked at and the eyes and mouths have to be classified by people. The issue of ambiguity when determining whether a subject is smiling or not and even if the eyes are open or not can be resolved by the fact that the PCA/LDA algorithm is trained using a large set of different facial images. Using a large database allows for averaging of different faces and features and eliminates some of the issues that could arise from training using a smaller database.

Step 2: PCA/LDA Algorithm Training and Testing

The main idea of PCA is to reduce the dimensionality of a data set consisting of a large number of interrelated variables while keeping as much variation present in the data set as possible.[4] PCA will enable us to get by computing the dxd covariance matrix by taking advantage of the fact that the covariance matrix is symmetric, and therefore has orthogonal eigenvectors. By normalizing the eigenvectors to unit norm, we have an orthogonal basis to represent the data. Therefore each facial image will be constructed as a linear combination of basis vectors multiplied by p_i scalars (weight coefficients). Since this is an orthogonal basis, we can easily find the coefficients p. These are just the projections of the signal onto each basis.

$$p_1 = x^T v_1$$
$$p_2 = x^T v_2$$
$$p_n = x^T v_n \qquad (2)$$

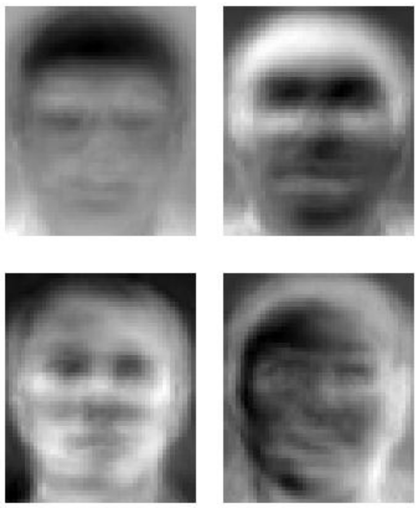


Figure 3. Examples of Eigenfaces [6]

To complete our training set, we will also be incorporating LDA to separate the classes (i.e. eyes open/closed and mouths smiling/not smiling). Our goal is to create Fisherfaces from the images by maximizing our objective function J(w). Fisherfaces will allow us to have a better discriminant for classification and will require fewer images from each class than just using eigenfaces to distinguish open/closed eyes and smiling/not smiling mouths.

Step 3: Face Detection using Viola-Jones Algorithm

We will perform pattern recognition analysis by implementing the Viola Jones algorithm on the PC to detect faces. Once we have the faces, we will use the Eigenfaces we obtained from performing PCA, and project them onto the PCA/LDA classifier lines that were trained using the algorithm. Then we can see where the data lies in comparison to the lines and determine which class the detected face is in. Depending on where the data lies, we will be able to classify if an eye is open or not and also if someone is smiling or not. The image we analyze is considered to be a 'good' photo if and only if both the eyes are open and the mouth is smiling. The Viola-Jones algorithm is explained in more detail in the next few pages.

The Viola Jones Algorithm

The Viola-Jones algorithm is an intricate algorithm that is primarily used in detecting faces. Since our project uses facial recognition as a first step, we will use the algorithm to detect all the faces in a group photo. The basic principle of the Viola-Jones algorithm is to scan a sub-window capable of detecting faces across a given input image, rescale the face detector, and run the detector many times through the image – each time with a different size. There are three contributions that Viola and Jones make in their paper. They introduce an image representation called the "Integral Image," which allows features used by the detector to be computed quickly, a learning algorithm based on Adaboost that selects a small number of critical visual features and yields extremely efficient classifiers, and a method of cascading classifiers to allow background regions of the image to be quickly discarded; “instead of finding faces the algorithm should discard non-faces.” [3]

Calculating the Integral Image

The first step of the Viola-Jones Face Detection algorithm is to turn the input image into an integral image. By computing the integral image, it enables us to compute Haar-like features, regardless of scale or location, in constant time. A simple rectangular Haar-like feature can be defined as the difference of the sum of pixels of areas inside the specified rectangle, which can be at any position and scale within the original image. To calculate the integral image, we calculate the sum of all pixels above and to the left of the concerned pixel (including the pixel itself) and substitute each value as the new pixel. This is shown in Figure 4.

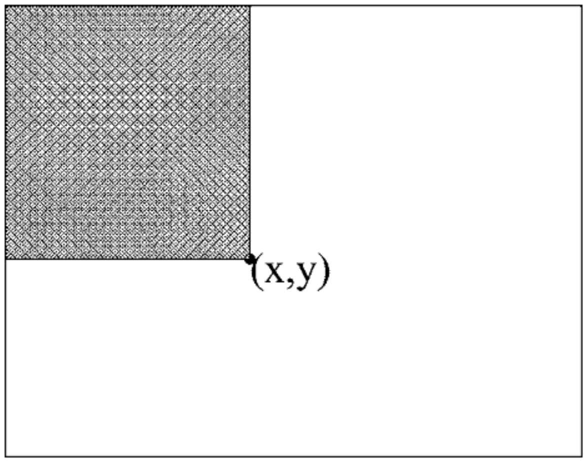


Figure 4. The value of the integral image at point (x,y) is the sum of all the pixels above and to the left of the given pixel[7].

The equation to calculate the integral image is:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y'), \qquad (1)$$

where ii(x,y) represents the pixel value of the point (x,y) of the integral image and i(x,y) represents the pixel value of the point (x,y) in the original image [7].

Implementation 1: do_int_image1

In practice, equation (1) is not very useful for code implementation. When running a C implementation (see loadImage.c, do_int_image1a and do_int_image1b) for calculating the integral image of a 512 x 512 photo on the lab computers, which are Dell Precision T3400 machines equipped with a 2.4 GHz Quad Core processor and 3.25 GB of RAM, it would take a long 88 seconds to execute this part of the algorithm alone. Pipelining and loop unrolling 3 operations in a single loop would only make this a little bit faster (69.67 seconds), but the algorithm would still take a long time to execute. Granted that this is on the PC, the DSK would probably not perform any better, so the implementation of equation (1) is therefore too slow for use in situations where quick face detection (no more than a few seconds) is required.

Recursive Viola-Jones

Because equation (1) is slow in computing the integral image, Viola and Jones suggested a recursive form to obtain the integral image, as follows [7]:

$$s(x, y) = s(x, y-1) + i(x, y) \tag{2}$$

$$ii(x, y) = s(x - 1, y) + s(x, y) \tag{3}$$

The use of both equations together would not be recommended on the DSK for grayscale images >= 683 x 683. If we assume that each point on the integral image has a value of 128 (the average value of each pixel of most images), then the lower right corner would have a value of $683^2 * 128 = 59,710,592$. At 683 x 683, each pixel in integral image *ii* and recursive step *s* must be of type `int`. Since type `int` is 4 bytes, then we have a total of 683 x 683 * (4 bytes) = 1,865,956 for each of *ii* and *s.* In total, we require 3,731,912 bytes of memory to allocate *ii* and *s* combined. Adding the original image, which is 512^2 * (sizeof(unsigned char) = 1 byte) = 466,489 bytes to this, we obtain 4,198,401 bytes. Since the DSK only allows for 4 MB of external memory, which corresponds to 4,194,304 bytes, we exceed the DSK's limits for external memory. When considering this in real-time, the size of the image is usually very large based on the resolution of pictures taken by a digital camera, which can be as huge as 12.1 megapixels (MP) in a camera such as the Canon Powershot[1], or as low as 3 MP for the iPhone, for example[2]. To ensure accuracy in Viola-Jones, the images must be captured with as much resolution as possible to maintain quality.

Implementation 2: do_int_image2

Since the implementation described by Viola-Jones will take up more than 4 MB at 683 x 683, the recursion formula can be tweaked so that it will work for images up to 915 x 915. The derivation for the tweak is demonstrated in Figure 5. Note that Figure 5 is NOT necessarily drawn to scale.

[1] Amazon.com. "Canon Powershot SD780IS 12.1 MP Digital Camera." http://www.amazon.com/Canon-PowerShot-SD780IS12-1-Stabilized-Black/dp/B001SER47Y/ref=sr_1_1?ie=UTF8&s=electronics&qid=1260163062&sr=8-1
[2] Apple.com. "iPhone – Technical Specifications." http://www.apple.com/iphone/specs.html

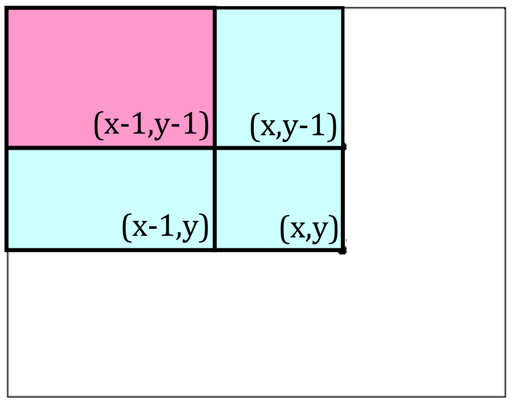


Figure 5. For the implementation, we can calculate *ii*(x,y) in terms of *ii*(x-1,y), *ii*(x-1, y-1), and *ii*(x-1, y) as follows. If we take the sum of i(x,y) + *ii*(x-1,y) + *ii*(x, y-1), we end up double counting *ii*(x-1, y-1). Therefore, ***ii*(x, y)** can be expressed simply as **i(x,y) + *ii*(x, y-1) + *ii*(x-1, y) – *ii*(x-1, y-1)**. The region in red is double counted and the part in blue is added only once [7].

The formula that we use is

$$ii(x, y) = i(x, y) + ii(x, y-1) + ii(x-1, y) - ii(x-1, y-1), \qquad (4)$$

instead of the one described by Viola-Jones, under the assumption that ii(-1,-1), ii(-1,0), and ii(0,-1) = 0. By doing this, we save memory because storing s(x, y) would require an additional 512^2 * (4 bytes) = 1,882,384 bytes. Thus, compared to the 4,198,401 bytes required by using the formula described by Viola and Jones, we only require 2,311,920 bytes, a savings of 45%. The method, do_int_image_2, executes in only 0.1363051 seconds, or 136.3 ms. With this in mind, the original image cannot be larger than 915 x 915 since 915^2 * (4 bytes for each pixel in int_image) + 915^2 * (1 byte for unsigned char) = 4,186,125 bytes, which is just less than the maximum of 4,194,304 bytes. 915 x 915 corresponds to an image size of only .83 MP, which is less than the number of MP used for digital photography.

Implementation 3: do_int_image3

Taking it a step further, we wanted to improve the memory usage for the integral image step so that it would work for images at the normal camera resolution. One such idea was to make a lossy compression algorithm where instead of storing the integral image *ii*, each value of *ii*(x,y) is divided by the number of pixels that needed to be added to obtain the integral image to form the average integral image, *aii*. If the coordinate of each pixel of the integral image is (x,y), then each pixel would be divided by (x + 1)*(y + 1), with (0,0) corresponding to the upper left pixel of the integral image. If we let X be the width of the image and Y be the height of the image, I be the set of i(x,y) $\in$ image *i*, II be the set of ii(x,y) $\in$ integral image *ii*, and AII be the set of aii(x,y) $\in$ average integral image *aii*, then for $\forall i(x,y) \in$ I, $\forall ii(x,y) \in$ II, $\forall aii(x,y) \in$ AII, and set Z being the set of positive integers,

i(x,y) $\in$ [0, 255] $\cap$ Z,

ii(x,y) $\in$ [0, 255*X*Y] $\cap$ Z, and

aii(x,y) $\in$ [0, 255] $\cap$ R

can be implied. We know that I $\in$ [0, 255] $\cap$ Z because we know that in grayscale, the values of pixels in images must take on 256 integer values, ranging from 0 to 255. In extreme cases, the image I can have all 0s (black) or all 255s (white) for pixels to yield a minimum of 0 for and a maximum of 255 *X*Y for $\forall ii$(x,y) $\in$ II. More often than not, the value of each pixel in set II will fall between the minimum and maximum values. The final statement, aii(x,y) $\in$ [0, 255] $\cap$ R is true because pixel values in an image will vary and thus the average of the pixel values over (x+1)*(y+1) need not be an integer, and can take on real values.

With this in mind, we know that currently *i*(x,y) takes on type `unsigned char`, *ii*(x, y) takes on type `int`, and *aii*(x, y) takes on type `float`. Because this would not save us any memory (total size is X*Y*(1+4+4) = 9*X*Y bytes), we have to perform lossy compression. Instead of creating the set II, we could simply calculate the value of each *ii*(x,y) and divide it by (x+1)(y+1) to avoid using *ii*(x,y), which would leave us with only values for *i*(x,y) and *ai*(x,y). To get back *ii*(x,y), it would only require us to find the product of (x+1)(y+1)**aii*(x,y) to obtain *ii*(x,y). This would leave us with 5*X*Y bytes, which is the same number of bytes as that required to perform do_int_image2, which still does not save us any memory.

Because *aii*(x, y) falls within the range [0, 255], we perform the compression here by truncating each *aii*(x, y) so that all values are integers. Thus, *aii*(x, y) can now take on `unsigned char` and we now only use up X*Y*(1+1) = 2*X*Y bytes, which saves us 60% over the memory usage of do_comp_2, and 88% over the formula suggested by Viola and Jones. While this may be better in terms of memory usage in implementation, it is very inaccurate as a compression algorithm. This is because once an average has a decimal part, truncation always rounds it down. Thus, error from this will propagate in the recursive step. To try to avoid this, we introduced a new variable, r(x,y) $\in$ R, which stores the integer value formed by the first two decimal places of *ii*(x, y) / ((x+1) * (y+1)). By doing this, *r*(x,y) takes on `unsigned char` also, so we only use up 3*X*Y bytes, which saves us 40% over the memory usage of do_comp_2, and 67% over the implementation suggested by Viola and Jones. Thus, we establish that $\underline{i(x,y) \in [0, 255] \cap Z, aii(x,y) \in [0, 255] \cap Z, \text{ and } r(x,y) \in [0, 99] \cap Z}$.

Even with this implementation, there was still a lot of error propagation when using it on a 512 x 512 image. Against the correct *ii*(x, y) obtained from do_int_image2, the upper right corner and lower left corner integral image pixel values are both inaccurate by only .6%. Even with that being said, the value of the integral image of the pixel at (128,128) was inaccurate by 12.78%, the value of the pixel at (256,256) was inaccurate by 36%, and the value of the integral image of the pixel at (512, 512) was inaccurate by 71%, which is not acceptable as a final result. To try to deal with truncation, a few conditions were added to enable rounding up when *ii*(x,y), but this showed only minimal improvement (no more than 5%). To improve this in the future, an implementation with truncation up to four decimal places could be instituted to save 20% over the memory usage of do_int_image2.

Because any image could be put into the Viola-Jones face detector, and we cannot put in an image larger than what the DSK can handle, we did not want to concern ourselves with this restriction and thus, we implemented the Viola-Jones algorithm in MATLAB instead where there is more flexibility as to what one can do with the images.

Implementation 4: do_int_image4

An attempt to optimize do_int_image2 was done with loop unrolling. The time it took to do this was 0.13556076 seconds or 135.56 ms, which is not a noticeable difference over the time taken in do_int_image2. The reason that it was not a noticeable difference was because even when the loop unrolling took place, there would not be a sufficient supply of registers that could be reused in order for loop unrolling to happen, since the formula for calculating the integral image itself has five terms, corresponding to five different registers.

Using the Integral Image

In order to calculate specific rectangle features that would be used later in Adaboost, where weak classifiers would be combined to form a stronger classifier, the integral image is used to make calculations easier. The features that are primarily used are rectangle features, which are shown in Figure 6 [7]:

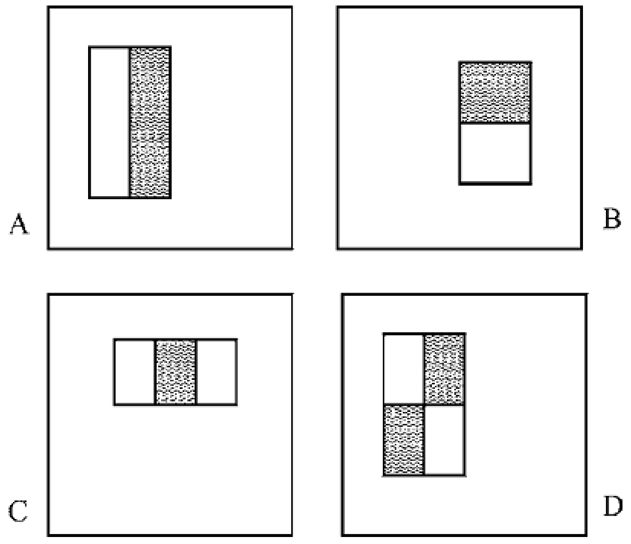


Figure 6. Example rectangle features shown relative to the given detection window. Each feature is calculated by taking the sum of the grey pixels and subtracting it by the sum of the white pixels.

In order to calculate these features, the results from the integral image must be used so that the calculations of the sums of pixels of the white and grey rectangles can be done in constant time. Illustrated in Figure 7 is how the rectangle sums can be derived in constant time.

The formula for calculating the sum of the pixels (SoP) in rectangle D is

$$\text{SoP in D} = ii(4) + ii(1) - ii(2) - ii(3), \qquad (5)$$

and the proof for this is shown in Figure 7. Given the base resolution of the detector is 24 x 24, Viola and Jones found 160,000 rectangle features. In the OpenCV implementation of Viola-Jones provided by Intel, only about 2,000 filters are actually used with different types and sizes and the subwindow, which is of a size of 24 x 24, is moved over the image in steps with 50% overlap. When the subwindow is resized, for each pass, it increases by a factor of 10% in each pass [8].

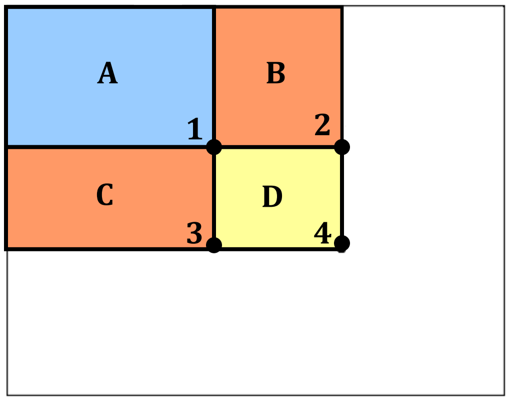


Figure 7. Using only four array references, the sum of the pixels in rectangle D can be computed as $ii(4) + ii(1) - ii(2) - ii(3)$, provided that $ii(1) = A$, $ii(2) = A + B$, $ii(3) = A + C$, and $ii(4) = A + B + C + D$. When adding them altogether as $ii(1) + ii(2) + ii(3) + ii(4)$, blue indicates a rectangle that is triple counted, orange indicates a rectangle that is double counted, and yellow indicates a rectangle that is only single counted [7].

Thus, the amount of possible features vastly outnumbers the 576 pixels contained in the detector at base resolution and they can be found with a high degree of computational efficiency, which is suitable for our purposes.

Adaboost

Although we did not do training for Viola-Jones and a training database exists for frontal faces through OpenCV, Adaboost is explained for completeness. Adaboost is a machine learning algorithm posed by Freund and Schapire in their 1996 paper, in which it is capable of constructing a strong classifier through a weighed combination of weak classifiers [7].

A weak classifier is defined as:

$$h(x, f, p\ \theta) = \begin{cases} 1 & \text{if } pf(x) > p\theta \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

where *x* is a 24*24 pixel sub-window, *f* is the applied feature, *p is* the polarity and θ is the threshold that decides whether *x* should be classified as a positive (a face) or a negative (a non-face). Since only a small amount of the possible 160,000 feature values are expected to be potential weak classifiers the AdaBoost algorithm is modified to select only the best features [3]. The main advantage of AdaBoost over other feature selection mechanisms is that a classifier can be learned in only O(MNK) time, where M is the number of weak classifiers, K is the total number of features, and N is the number of examples analyzed [7].

Cascading Filter

The basic principle of the Viola-Jones face detection algorithm is to scan the 24 x 24 detector many times through the same image – each time with a new size. Even if an image should contain one or more faces it is obvious that an excessively large amount of the evaluated sub-windows would still be non-faces [3].

We therefore reason in that “Instead of finding faces, the algorithm should discard non-faces.” [3] With the cascaded classifier, we determine at each stage whether a given sub-window is definitely not a face or possibly a face, denoted as a maybe. Once a sub-window is classified as a non-face it is discarded and a face classified as maybe moves on to the next stage. As a given sub-window passes through more stages of the cascade, the chance that the sub-window contains a face increases. To construct the cascade, AdaBoost is used to train the classifiers. Starting with a two-feature strong classifier, the face filter is obtained by adjusting the strong classifier threshold from AdaBoost to minimize false negatives. By lowering the threshold, detection rates and false positive rates increase, which explains the situations where false positives exist [7]. Each stage of the cascade is essentially another strong classifier obtained from AdaBoost with a different threshold at each step to maximize the accuracy of the algorithm. Figure 8 shows the cascading filter in action:

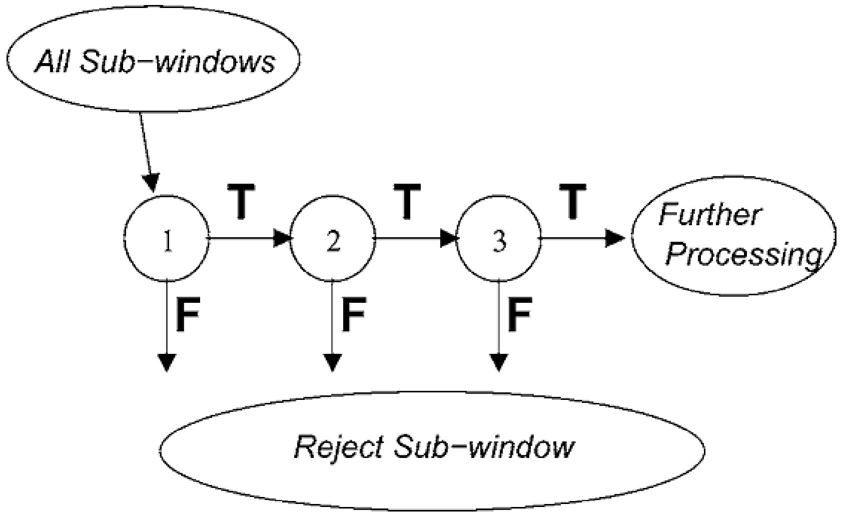


Figure 8. This diagram shows how the classifier cascade is meant to work. The first stage eliminates 50% of the non-faces and subsequent classifiers use more scaled directional filters and omits more non-faces.

Additions to Viola-Jones

With these three steps alone, our implementation of the Viola-Jones algorithm in MATLAB detects all the faces correctly, but produces several false positives (type I errors), i.e. other regions of the image that are clearly not faces are classified as faces. The problem with having false positives at this step is that when it comes time to determine whether each region of interest (ROI), which could be a face or one of the other false positives from Viola-Jones, falls into the four classes we defined earlier in this report, it is possible that a "non-face" will be projected onto the separation vector from LDA, which will result in an output of "bad photo" from our complete photo detection system. This may be fine if the photo is bad since there is a person who isn't smiling, but it isn't if the photo is actually good, thus we have a false negative result from our system. To minimize the likelihood of false positives from Viola-Jones, we incorporated a fourth step and used the following inequalities on RGB for skin color discrimination on the center point of the face [5].

$$(|R - G| > 15) \;\&\&\; (R > G) \;\&\&\; (R > B) \qquad (7)$$

By incorporating the skin color discrimination line to the MATLAB code, there were 8 false positives from Viola-Jones that it was able to eliminate (see "Analysis" section). Originally Viola-Jones detected 65 faces, 55 of which were correctly identified, which corresponds to a detection accuracy of 84.62%. When applying the skin color discrimination line to the MATLAB code, Viola-Jones detected 57 faces, 55 of which were correctly identified, which corresponds to a detection accuracy of 96.49%--an 11.87% improvement. There can still be some false positives and negatives, which can be explained by the fact that Viola-Jones starts at the base scale of 24 x 24 and so some false positives are registered into Viola-Jones when only a single "maybe" is detected and thought of as a face and there are no other faces around it. Another side effect from Viola-Jones is the failure to detect a face due to excessive tilt or phase shift. Viola-Jones can only detect faces that are tilted up to about ±15 degrees in plane and about ± 45 degrees out of phase toward a profile view [7]. The detector becomes unreliable with more rotation than this.

Step 4: Face Data Projection using PCA and LDA on DSK

Once we have detected, cropped and resized the individual faces in the image using Viola-Jones, we classify each of the faces as “good” or “bad” by simply projecting the faces onto the PCA/LDA classifier and finding the k-nearest neighbor using Euclidean distances. The projection process involves first subtracting the average face from the face to be classified. We then multiply the matrix p0 by the normalized face (face with average subtracted) to obtain the testWeights. Once this is done, we multiply the testWeights by v_manx, v_many and v_manz to obtain the x, y and z co-ordinates respectively [1]. This is the projection process. The coordinates obtained are then used to find the nearest neighbor and thereby classify the current face.

Step 5: Nearest Neighbor Classifier on DSK

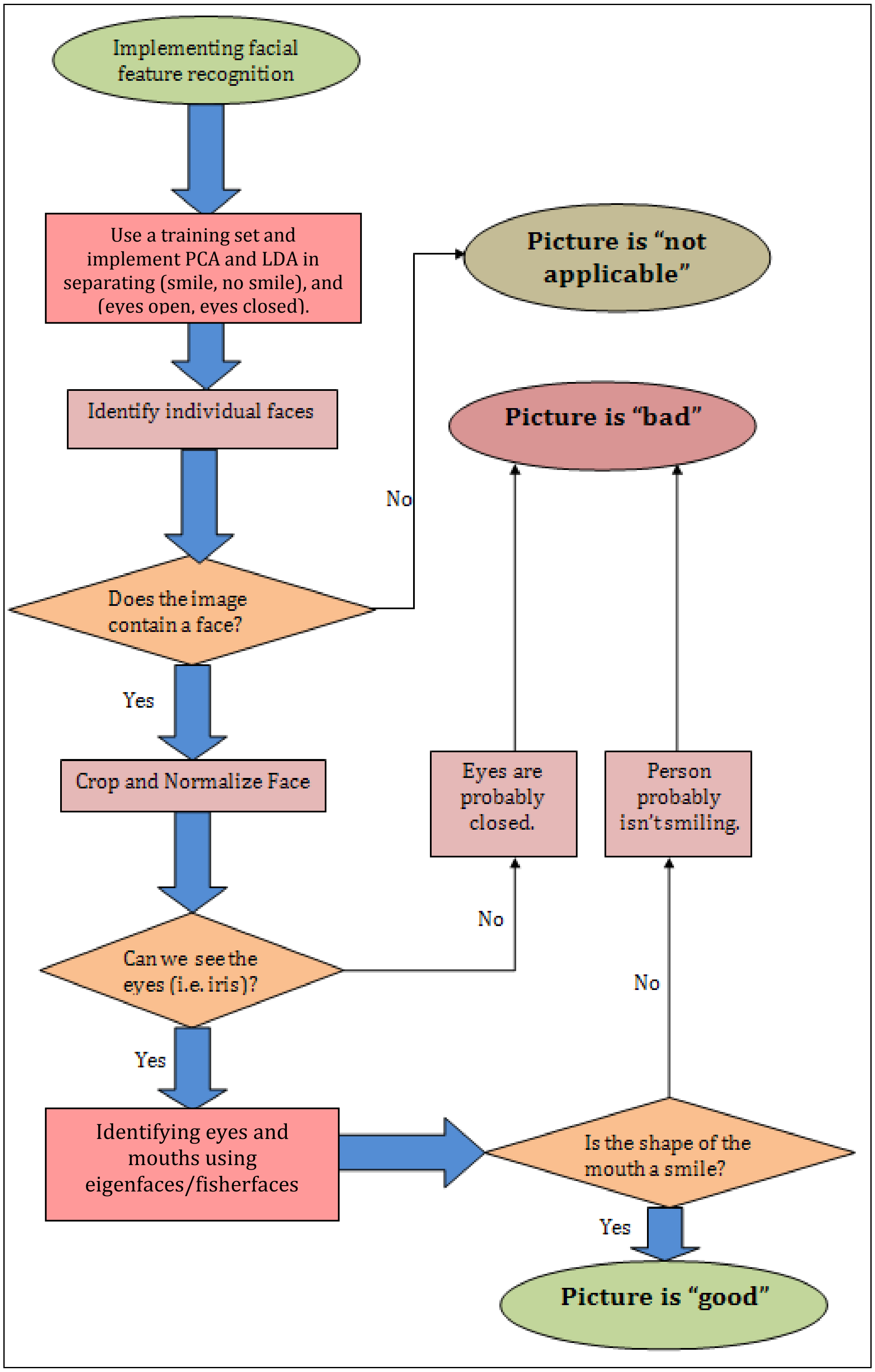
The facial image detected using the Viola-Jones algorithm would be projected onto the eigenspace using the LDA algorithm. Then a distance array with a size of the total number of training set images is filled with Euclidean distances measured between each training image value and the testing image value. Starting with the distance of the first training image being the best or shortest distance, each Euclidean distance in the array is compared with the current best. If the distance being compared is shorter than the current best, it is replaced with the current value and comparisons continue till the end of the array. Finally the best value will define the shortest Euclidean distance between two points and thus reflect that the chosen training facial image is the nearest neighbor. The value of the nearest neighbor is then used to obtain the class that the testing image takes on.

Step 6: Result (Good or Bad) shown in GUI

Finally the results of the analysis, good or bad, are transferred back from the DSK to PC using TCP. If any face detected using the Viola-Jones algorithm on the DSK was classified as a “bad” face, then the GUI would show the inputted image with a sad face to show that the photo is a “bad” photo. If all faces are classified as “good” faces with open eyes and a smiling mouth, the GUI produces a smiley face with the image.

Figure 9. GUI Result for Good and Bad Photo (Left = Good, Right = Bad)

The above steps that we see in the process of analyzing our photographs are visually detailed by the flow chart shown on the next page in Figure 10. The flowchart of the interaction and task distribution between the PC and the DSK are shown on the page afterwards in Figure 11.


Implementing facial feature recognition
Use a training set and implement PCA and LDA in separating (smile, no smile), and (eyes open, eyes closed).
Identify individual faces
Does the image contain a face?
No
Picture is "not applicable"
Yes
Crop and Normalize Face
Can we see the eyes (i.e. iris)?
No
Eyes are probably closed.
Picture is "bad"
Yes
Identifying eyes and mouths using eigenfaces/fisherfaces
Is the shape of the mouth a smile?
No
Person probably isn't smiling.
Yes
Picture is "good"
Figure 10. Flowchart of Implementation Process

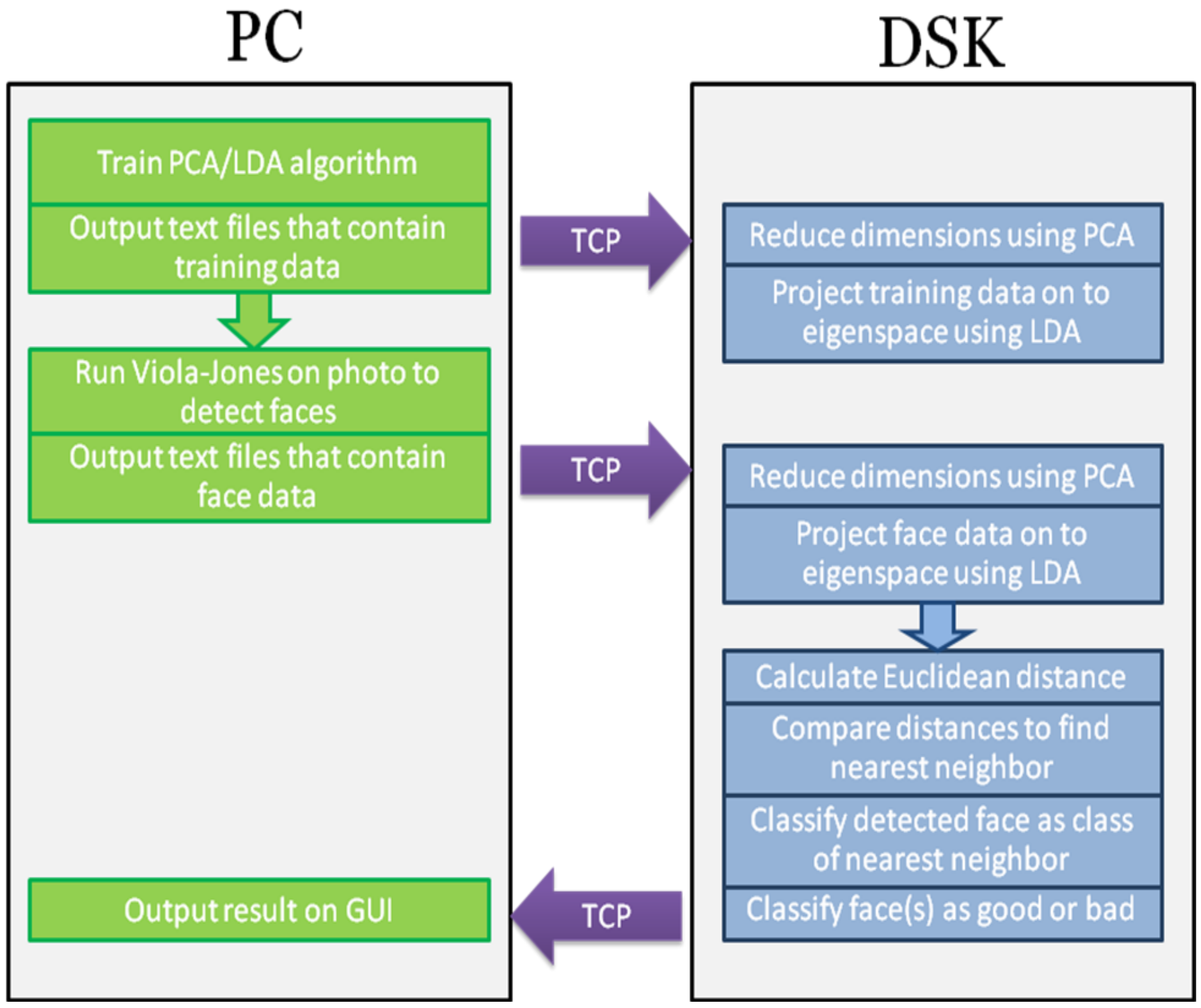

PC
DSK
Train PCA/LDA algorithm
Output text files that contain training data
Run Viola-Jones on photo to detect faces
Output text files that contain face data
Output result on GUI
TCP
TCP
TCP
Reduce dimensions using PCA
Project training data on to eigenspace using LDA
Reduce dimensions using PCA
Project face data on to eigenspace using LDA
Calculate Euclidean distance
Compare distances to find nearest neighbor
Classify detected face as class of nearest neighbor
Classify face(s) as good or bad

Figure 11. PC and DSK Flowchart

Data, Code, and Pixel Sizes

Data Size

The data sizes for our files were all fairly small, which made it easier to put everything on the DSK. The text files that are outputted by the LDA classifier training and placed into the DSK include: resizedFace.txt, v.txt, p0.txt, trainWeights.txt, trainIds.txt, and fbgAvgFace.txt. The sizes of each of the text files containing data needed for PCA dimension reduction, LDA projection, and nearest neighbor classification are listed in the table below. Of these the largest file contains the p0 values because it reflects the principal components of all the training set images that are to be projected on to the eigenspace using the LDA algorithm on the DSK.

Data File	Size
resizedFace.txt	5kB
v.txt	1kB
p0.txt	112kB
trainWeights.txt	4kB
trainIds.txt	1kB
fbgAvgFace.txt	2kB

Table 2. Table of Data Sizes of DSK Input Text Files

Code Size

The sizes of our PC and DSK side code are fairly small. Our PC side code for Picture Perfect (ppPC.c) is 4kB while the DSK side code (ppDSK.c) is 6kB. The PC side code reads in the text files containing the input values needed for the PCA/LDA algorithm and classifier on the DSK and transfers the information to the DSK. It also accepts the transferred output from the DSK and outputs the Boolean to a text value to be displayed to the user through the GUI. The DSK side code performs PCA, LDA, and the nearest neighbor classifier algorithm using the various data transferred from the PC. Then it uses the classified faces to determine whether the photograph is good or bad and transfers the output back to the PC.

Pixel Size

The pixel sizes of the photograph to be analyzed using our Picture Perfect program can be any image size as long as has a total pixel size of less than 1000. For the image to be displayed properly on the GUI, however, the image needs to have a size of 640X480 pixels. Also the normalized face images are cropped and resized to 97X110 pixels to be displayed on the GUI.

Optimizations

Parallelization

By allowing for pipelining, the program is able to run much more efficiently on the DSK. This makes use of multitasking as multiple computations occur in parallel instead of waiting for one to complete before starting the next one. In the function projectFace(), where a series of matrix multiplication computations are called to project the facial data onto the eigenspace, parallelization takes place. Three iterations of matrix multiplication of the values v_manx and v_many are performed in parallel. In addition the sqroot computation also runs three in parallel. Another function where parallelization takes place is classifyNearest(). Finding the shortest distance by comparing the current best distance with the rest of the Euclidean distances in the array runs four iterations in parallel. Parallelization of Viola-Jones functions is explained for each do_int_image in the above section, “The Viola-Jones Algorithm.”

Running Times

Viola-Jones functions

Function	**Time**
do_int_image1a	88000 ms = 88 s
do_int_image1b	69670 ms = 69.67 s
do_int_image2	136.3 ms
do_int_image4	135.56 ms

Classification functions

Function	**O2 (cycles)**	**O3 (cycles)**
projectFace() PCA, LDA projection	225,055 = .99ms	225,677 = .99ms
classifyNearest() nearest neighbor classification	6,087,120 = 26.78ms	5,932,553 = 26.10ms
euclidean() calculate euclidean distance	6,045,621 = 26.60ms	6,023,589 = 26.50ms
do_comp() analyze photo for good/bad faces	6,290,537 = 27.68ms	6,160,155 = 27.10ms

Tables 3-4. Running times.

Final Lab Demo

The lab demo will be done using a GUI where we will analyze multiple group photographs. We will be using TCP to perform both PC → DSK and DSK → PC transfers. The image files will be fed into the DSK where the Viola-Jones algorithm will detect the faces and process them to be projected onto the PCA/LDA vector projection to classify each face detected in the image. The results of the analysis done on each of the face are then compared. If any face was detected as a “bad” face whether the eyes were closed or the mouth was not smiling, the DSK would return the Boolean value for a bad photo to the PC. On the other hand, if every face in the image was found to be a nice smiling face, the PC receives the Boolean value for a good photo. This information is then displayed to the user using the GUI with a smiley face for good photos and a sad face for bad photos. The sequence of demo outputs are shown in the following pages.

LDA Classification Analysis:

Image	**#Faces Detected**	**Correctly Classified**	**Incorrectly Classified**	**Classification Accuracy %**	**Comments**
Single					These single portrait images for the most part did not have any issues with being classified correctly
asiangirl.jpg	1	1	0	100.00%	
lena.jpg	1	1	0	100.00%	
eyesclosedgirl.jpg	1	1	0	100.00%	
Group Pics					
prom.jpg	2	2	0	100.00%	Viola Jones did not recognize two of the faces in picture
Group 5a.jpg	13	6	7	85.71%	
Group 6a.jpg	6	4	2	66.67%	
Group 8a.jpg	14	8	6	57.14%	One false positive picked up by VJ.
Group Nihal.jpg	6	3	3	50.00%	Tilted/Side profile faces ignored by VJ detection
fiveguys.jpg	6	2	4	33.33%	One false positive picked up by VJ.
fivemoreguys.jpg	5	0	5	0.00%	
				68.81%	

Table 5. LDA Classification Analysis.

Our results were based on face images with the size of 18 x 15 pixels with varying number of faces in each group image. More test results follow on the next page.

Problems/Errors:

Figure 12: Group 8a.jpg

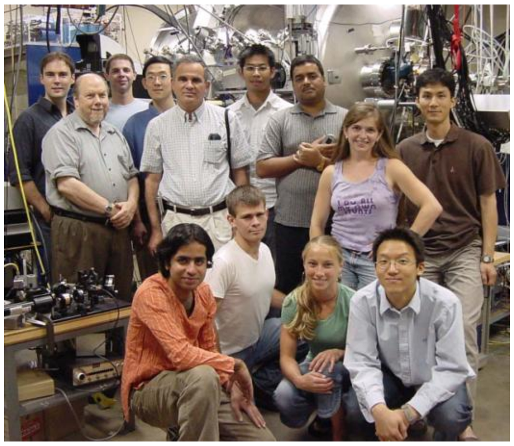
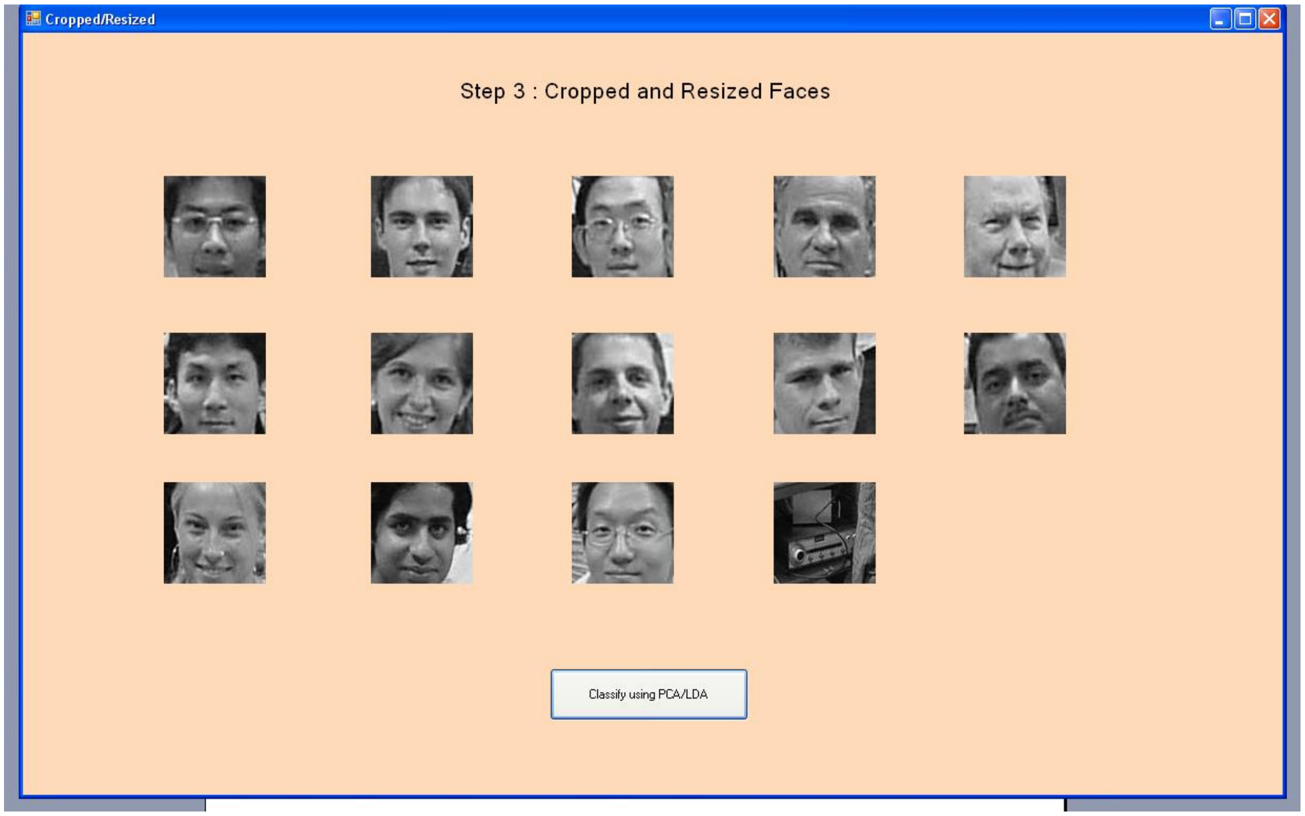


Table 6: Group 8a Faces

Face	Classified Class	Actual Class
1	3	3
2	3	1
3	3	3
4	3	3
5	3	2
6	3	1
7	1	1
8	3	1
9	3	3
10	3	3
11	3	1
12	3	3
13	3	3
14	3	4

Figure 13: Group 8a faces cropped/resized



Note: The faces in the GUI correspond to the spreadsheet in sequential order left to right, and then top to bottom. (i.e. 1st row – 1 to 5; 2nd row – 6 to 10; 3rd row – 11 to 14)

Figure 14: fivemoreguys.jpg

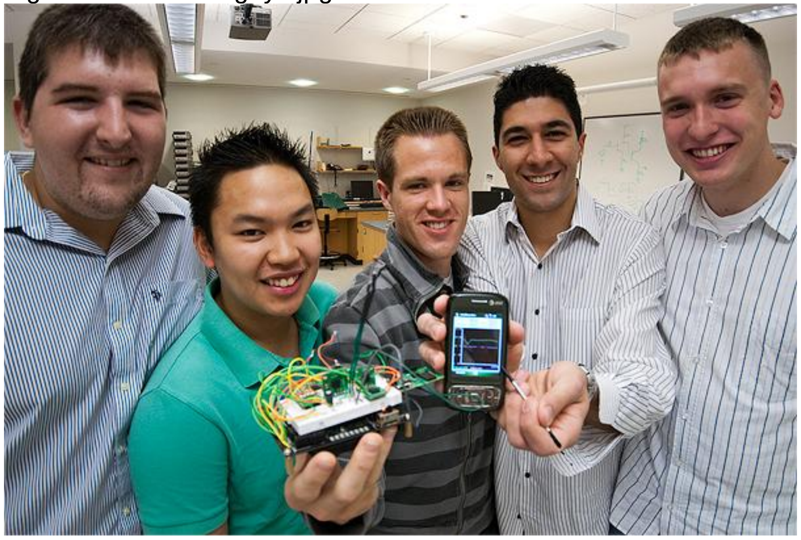


Figure 15: fivemoreguys cropped/resized faces

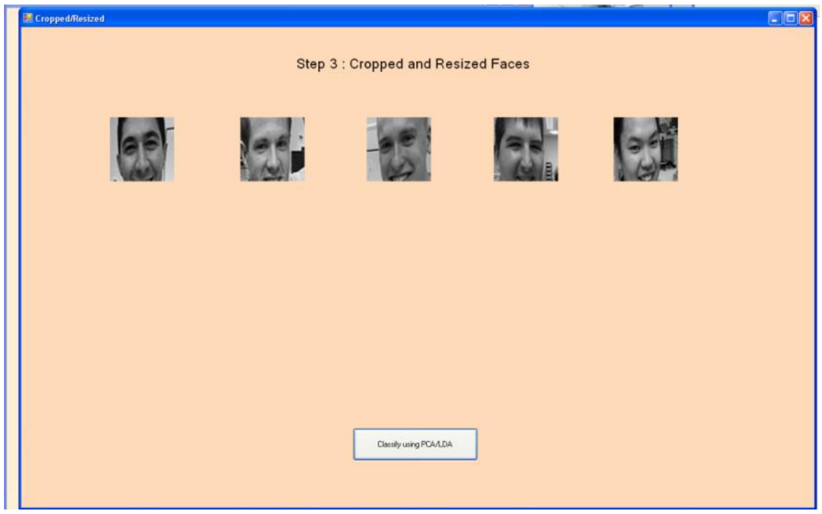


Table 7: fivemoreguys Faces.

Face	Classified Class	Actual Class
1	2	1
2	2	1
3	2	1
4	2	4
5	2	1

Note: The faces in the GUI correspond to the spreadsheet in sequential order left to right, and then top to bottom

The classifying algorithm places each face in this group photo in class 2, but this should not be the case. All of the faces except Face 4 should have been in class 1 because their eyes are actually open, but since they are not clearly wide open, but rather squinting; it is understandable to see why they'd be placed in class 1. Face 4 was classified as 2, but should have been placed in class 4, because although the person's teeth are clearly showing, it does not seem as though the person is actually smiling. Both of these problems are similar to the ones in the previous image, and would be resolved if more training images were added to each of the various classes depicting different ranges of smiles and eyelid openings/closings.

Shown on the next page is the GUI implementation of our photo classification system.

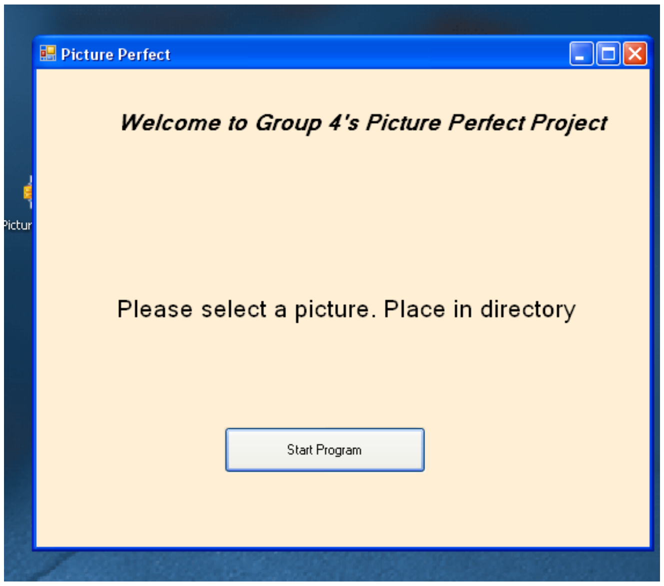

Picture Perfect
Welcome to Group 4's Picture Perfect Project
Please select a picture. Place in directory
Start Program

Figure 16. Welcome Page

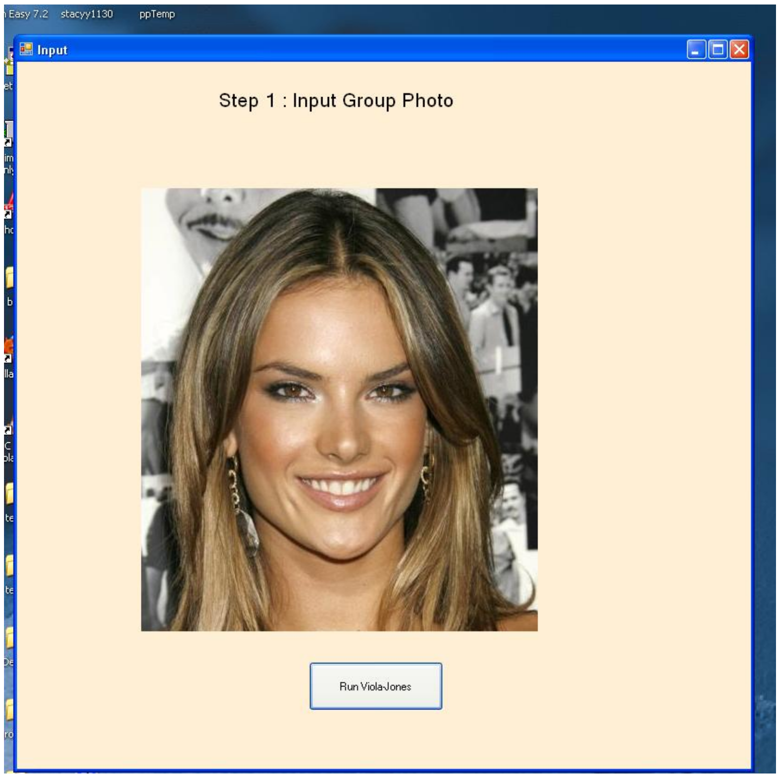


Figure 17. Input Photo

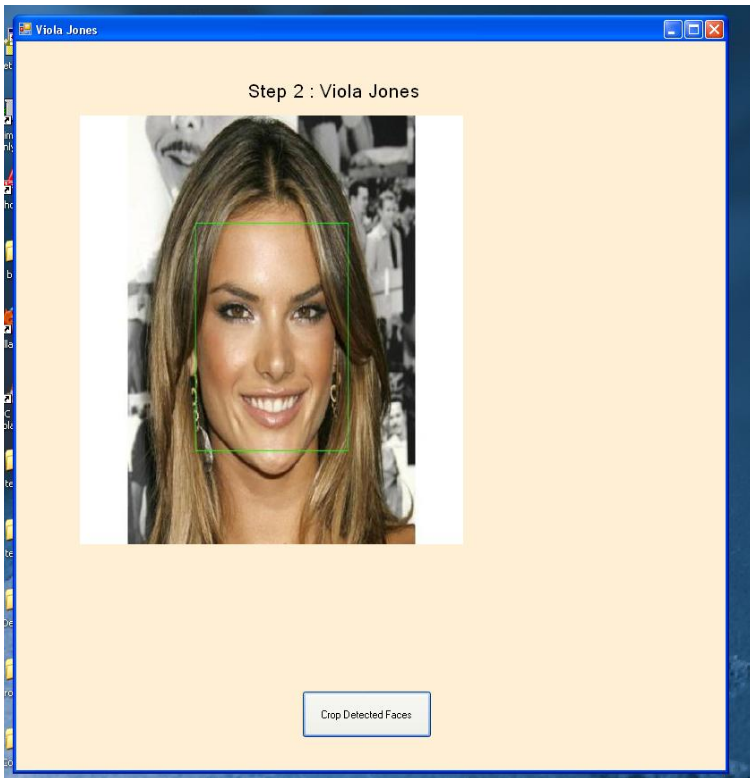


Figure 18. Viola-Jones Detected Face(s)

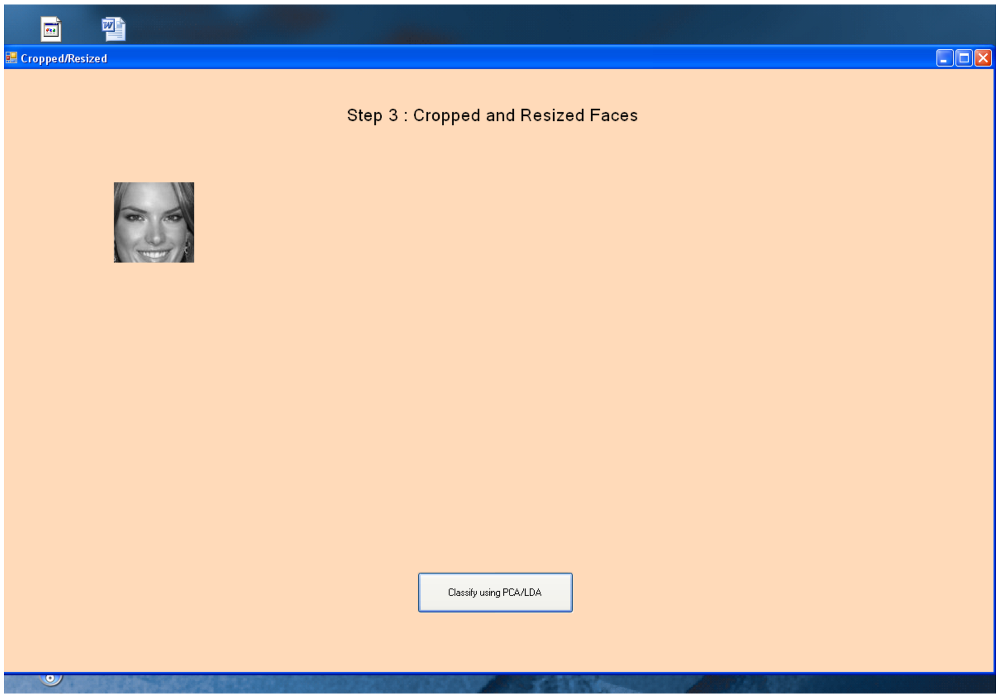


Figure 19. Cropped and Resized Detected Face(s)

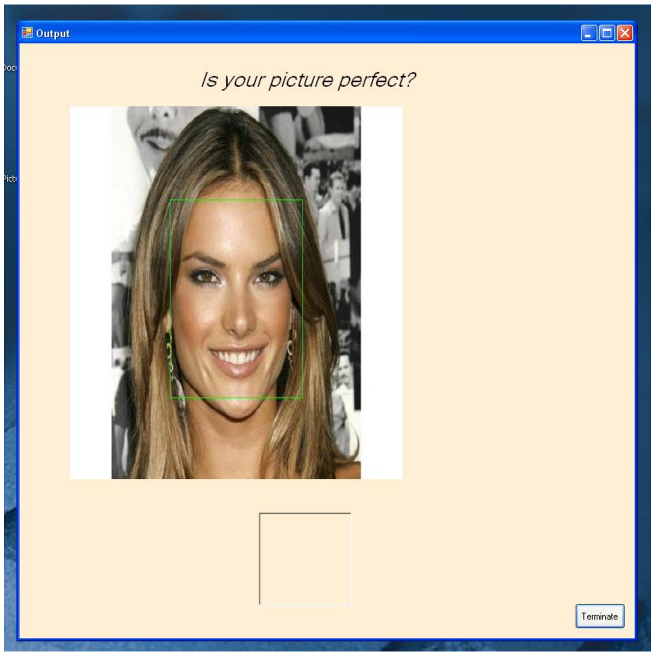


Figure 20. Face Classified Using Nearest Neighbor Algorithm

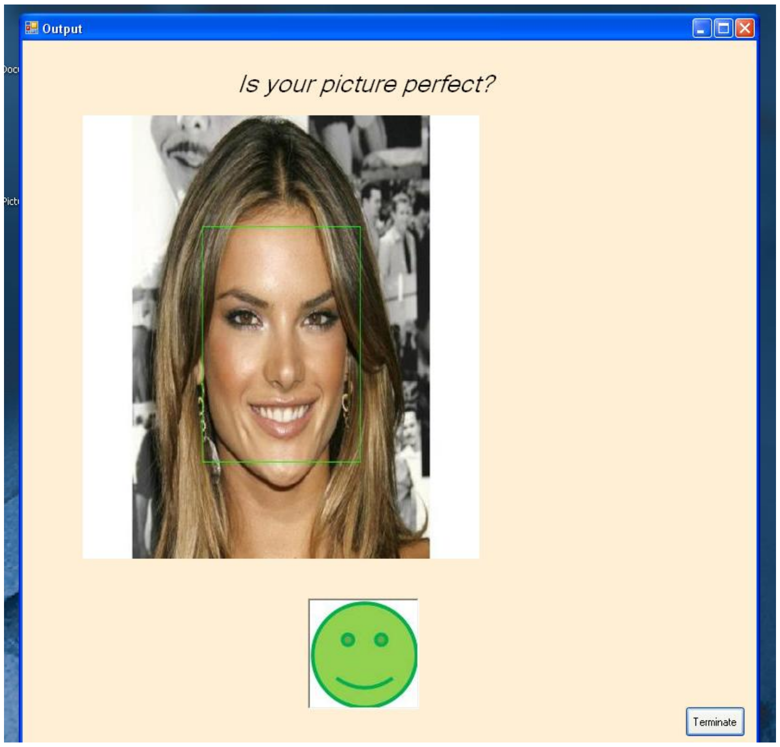


Figure 21. Final Output (Smiley Face = Good, Frowning Face = Bad)

Analysis of Results

Running our Picture Perfect program on multiple testing images and analyzing the results allowed us to see where we could make improvements to enhance the accuracy of our classifier algorithm.

Viola-Jones Detection

Viola-Jones does a very good job in identifying the faces in each image, especially the photos with only one person in them; however, Viola-Jones either does not catch all of the faces, particularly in prom.jpg, or triggers false positive faces, as in fiveguys.jpg. The reason that Viola-Jones sometimes does not detect all the faces is because it can fail to detect a face due to excessive tilt or phase shift. Viola-Jones can only detect faces that are tilted up to about ±15 degrees in plane and about ± 45 degrees out of phase toward a profile view [7]. The detector becomes unreliable with more rotation than this. Viola-Jones detects all of the faces in Group Nihal.jpg, even when one of the faces falls into this category. False positives from Viola-Jones can occur either because of low thresholds set in the cascading classifiers as explained earlier, or because the initial size for detecting faces is 24 x 24, which also happens to be the minimum subwindow for which Viola-Jones starts detecting faces. In this case, some false positives are registered into Viola-Jones when only a single “maybe” is detected and thought of as a face and there are no other faces around it.

Classifying Algorithm

The classifying algorithm does a solid job of clearly recognizing those faces that are in class 1 within this group photo. However, it has a bit of an issue of over classifying images into class 3. For the most part, this is due to the fact that our training set only contains 15 images of each class, and thus it is difficult to take into account the varying degrees of eyelid openings and the varying degrees of a smile. For instance, Faces 2, 6, 8, and 11 should have been classified under class 1, but instead were done so under class 3 because our training set tends to lean heavily towards smiles being of those with teeth showing. Also, Face 5 was classified as 3, but considering the person’s eyes are more closed than open, it should have been classified as class 2. This issue can be explained by the fact that our training set does not contain enough discriminating training images of the various degrees of eyelid closure. Lastly, Face 4 was misclassified because it technically is not a face, but was able to bypass the Viola-Jones and Skin Tone Detection in MATLAB.

Illumination Variations

Illumination variations refer to the effects that different lighting may have on a given face image. Depending on how a face is lighted, it can look very different even to the point where two images of the same person appear to be separate identities. To handle illumination variations, the first 10 eigenfaces were discarded. The reason for this is because the first few eigenfaces tend to carry a lot of illumination variations that make it difficult to distinguish key features.

Removing the mean face calculated from the training set images normalizes the detected face to be tested and classified. Subtracting the mean face from the testing image removes illumination variations and other characteristics that do not help in classification. [10] This process thus leaves the most distinguishing features of the face to make comparisons to test images more effective in successfully determining the correct class of the face.

We previously discarded the top 10 eigenfaces to deal with illumination variations, but we realized that the training set images we were using in our MANS database didn't carry much illumination variation. Thus by discarding the top eigenfaces that contain the most energy, we were jeopardizing the effectiveness and correctness of our algorithm. So we tried keeping the first 10 eigenfaces and found that instead of decreasing the accuracy of our PCA/LDA classifier algorithm, the accuracy stayed the same. In effect, in our case discarding the top eigenfaces neither helped nor harmed the final results of our algorithm.

Future Considerations

Analyzing the results that we obtained from our Picture Perfect project showed that there is still much improvement to be made to have a truly perfect photograph classifier based on facial feature analysis.

One thing that should be considered to increase the effectiveness and accuracy of this project would be to create a database with a very even distribution of ethnicities, gender, and classes for training the algorithm. The method we use with a nearest neighbor approach to classifying the detected faces puts the most importance on the distribution of the training set facial images that are projected on the eigenspace to be compared with the face to be classified. Each race has distinct facial features that define that particular ethnicity. These features include the eyes and the mouth. So if there aren't enough images of a particular ethnicity in the training set, the classification accuracy for that particular face may not be as high. For our project, we lacked training images for faces of African ethnicity and thus found difficulty in classifying them when detected using Viola-Jones and projected using PCA and LDA. Other races such as Asian, Hispanic, and Indian should also be considered when making sure the database used to train the classifier algorithm has equal distributions and fair representations for all different types of people. Gender and age are also things to consider when making the database as well represented of all possible test images. The database used to train the algorithm could be seen as the single most important aspect of classification and thus can use improvement for future projects considering the concept of classifying faces using the nearest neighbor approach.

Another thing to consider in the future in regards to this project is to increase k when using the nearest neighbor approach to classify faces. For our project we simply used k=1 to maintain the simplicity and efficiency of the algorithm. The problem with classifying a facial image exclusively using its single nearest neighbor on the eigenspace projection may not be an accurate representation of whether the face has open eyes and a smiling mouth. Increasing k would be more accurate because it would take into account more neighbors of a testing facial image. For example, if the Viola-Jones algorithm detected a bearded face. Using a single nearest neighbor approach may simply look for a face in the training set that most resemble the given image, which would have a high probability of being a bearded face. The two faces may have totally different characteristics in terms of the eyes being open or closed or the mouth smiling or not. Thus by taking another step and looking at additional neighbors could increase the effectiveness of the classifier algorithm and increase the accuracy.

Schedule

This project will take two months to complete because we need to be able to implement our code in Matlab and in C on the DSK, as well as to create a GUI.

Date	Task	Member
September 21	Algorithm Decisions for Project	Everyone
September 28	Research on PCA Research on LDA Research on Viola-Jones Research on Nearest Neighbor Classifier	Nihal Ajay Stelios Minwoo
October 5	Database Comparisons	Minwoo
October 12	OpenCV Viola-Jones Algorithm	Stelios
October 19	MANS Database Creation Euclidean Distance Code in C	Ajay Minwoo
October 26	Viola-Jones Algorithm Matlab Testing Nearest Neighbor Classifier Code in C	Stelios Minwoo
November 2	PCA Algorithm in C LDA Algorithm in C Database Testing of Classifier Algorithm	Nihal Ajay Minwoo
November 9	Project Oral Update	Everyone
November 16	DSK Transfers	Everyone
November 23	GUI Design Optimization on DSK	Ajay Nihal
November 30	Final Presentation + Demo	Everyone
December 7	Final Written Report	Everyone

Table 8. Approximate Schedule with Responsibilities

References

[1] Becker, B.C., Ortiz, E.G., "Evaluation of Face Recognition Techniques for Application to Facebook," in Proceedings of the 8th IEEE International Automatic Face and Gesture Recognition Conference, 2008 (Matlab)

[2] Brooks, Alan in collaboration with Li Gao. ECE 432 Computer Vision with Professor Ying Wu. “Face Recognition: Eigenface and Fisherface Performance Across Pose Report,” submitted on June 9, 2004. <http://dailyburrito.com/projects/facerecog/FaceRecReport.html>. (Matlab)

[3] Jensen, Ole Helvig. “Implementing the Viola-Jones Face Detection Algorithm.” Kongens Lyngby, 2008.

[4] Jolliffe, I.T. *Principal Component Analysis,* 2nd Ed. New York: Springer Verlag New York Inc, 2002.

[5] Nusirwan Anwar bin Abdul Rahman, Kit Chong Wei, and Johm See. *Faculty of Information Technology, Multimedia University*. “RGB-H-CbCr Skin Colour Model for Human Face Detection”

[6] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection," in IEEE TPAMI. vol. 19, 1997, pp. 711-720

[7] Viola, P. and Jones, M. “Robust Real-Time Face Detection”, in *International Journal of Computer Vision*, 57(2). Kluwer Academic Publishers, 2004.

[8] Hewitt, Robin. *Seeing With OpenCV*. Servo, Volume 2, 2007. Found at http://dasl.mem.drexel.edu/~noahKuntz/openCvPart02.pdf

[9] "k-nearest neighbor algorithm - Wikipedia, the free encyclopedia." Wikipedia, the free encyclopedia. N.p., n.d. Web. 5 Dec. 2009. <http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm>.

[10] "Face Recognition using Eigenfaces and Distance Classifiers: A Tutorial « Onionesque Reality." Onionesque Reality. N.p., n.d. Web. 6 Dec. 2009. <http://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/>.

[11] OpenCV Viola-Jones Algorithm (Matlab)