MATH & PENCIL

JOSEPH MISITI
CO-OWNER
MATHEMATICIAN
PROGRAMMER
DATA SCIENTIST
NATURAL WINE ENTHUSIAST

DETECTING HUMAN EMOTIONS USING FACIAL RECOGNITION

I. Introduction

The ease at which humans can recognize a face, understand spoken words, read handwritten characters, identify our car keys in our pocket by feel, and decide whether an apple is ripe or not belies and astounding complex process that underlies these acts of pattern recognition [9].

Most pattern recognition problems are solved in three stages: data preprocessing, feature extraction, and classification. The first stage of the problem is data preprocessing. This step simplifies the subsequent operations without losing relevant data. For facial recognition, this process usually involves some sort of normalization so each incoming image has a common baseline. The second stage of the processing is feature extraction. The purpose of this process is to reduce the amount of data by measuring certain "features" or "properties". Arguably, this step is the most difficult to implement because choosing the correct features can be very difficult. There is no way to know which features will work best beforehand. Some features are redundant. Even if the difficultly or computation cost of finding features is of no concern, might we ever have too many features—is there some curse for working in very high dimensions?[9]

The features (the value of these features) are then passed to the final step of the process, the classifier, which evaluates the evidence presented and makes a final decision. There has been extensive research done in area of classifiers (machine learning algorithms, artificial intelligence, etc.). Different classifiers work well for different types of problems. For this purposes of this paper, I have decided to use Support Vector Machines. Other examples of classifiers include Boosting techniques, Random Forests, and Nearest Neighbor algorithms. An overview of this process can be seen in **Figure 1**.

Figure 1: Pattern recognition process. These three steps can be found in most standard pattern recognition applications.

Computational face recognition is an important research problem in computer vision, presenting many applications such as in human-computer interaction, security systems and surveillance. There are two main approaches to face recognition by computers, namely static and dynamic. While the former is related to recognizing people in still images, the ladder addresses the problem of detecting, tracking, extracting information, and recognizing moving people in video sequences [1]. For the purposes of this paper and the research, my methods will examine the former approach.

II. Approach

I developed an image classification system using MATLAB and a variety of open source software packages. The basic approach to my system can be seen in **Figure X**.

Figure 2: Overview of my image classification system. These steps are a more specific variation of the steps found in Figure 1.

The steps I used in developing and testing my system can be described in the following order:

1. Choose testing and training images. These images will be used to train and evaluate the performance of the support-vector machines. All images chosen initially will contain similar pose and illumination. Removing these factors will simplify the problem I am trying to solve.
2. Use face-cropping algorithms to crop the face region and removing redundant background. The cropped image should only

contain the face, from the tip of the chin to the top of the head.

3. Extract features from images by convolving each image with the Gabor filter. I will apply a variety of different Gabor wavelets at different orientations and frequencies.
4. Using the features extracted in III, construct the SVM kernels. The process for this construction can be found below.
5. Train the SVM algorithms using an image sequence that consists of positive and negative examples of features I am trying to identify. Initially, I propose only trying to detect frowns and smiles, so my training set will consist of positive and negative examples of both of these expressions.
6. Evaluate the performance of the feature extraction techniques using a test image sequence. Note, the test sequence *will not contain* any images that are included in the training sequence. This is an example of cheating from the computer vision perspective.
7. Analyze the results using receiver-operator curves (ROC)
8. If time permits, expand algorithms to identify other facial expressions.

III. Face Cropping

The first step in my classification process was to locate and crop the face region from an input image. The main reason I decided on this approach was due to the fact that since I am trying to classify facial expressions, the background of the image is not necessary and most likely adds noise to my classification process. If the input to my process is a frame containing only one face and no background objects, the feature extraction process becomes much easier and my classification will be more accurate.

My face-cropping algorithm was developed in MATLAB and utilizes a variety of algorithms provided in OpenCV 1.1 to find the bounding box of the face (see section IV). I chose to use cascading Haar filter banks to find the faces in my frame. The OpenCV algorithm (cvHaarDetectObjects) finds rectangular regions in a given image that are likely to contain objects the cascade has been trained to detect (in my cases faces). The algorithm scans the image several times at different scales. Each time it considers and overlapping regions in the image and applies the classifier to the regions [7]. I also have configured the algorithm to use Canny pruning to reduce the number of analyzed regions. Canny pruning uses the Canny edge detection algorithm to find regions in the image that contain too many edges for the object of interest. When these regions of the image are discovered they are thrown out and the complexity of the face detection problem is reduced significantly.

After determining the location of the face, the face has to be cropped to fit an N x M patch that contains only the face region. This cropping can be done fairly easily using an affine transformation. An affine transformation is a linear transformation that can map three indices of one matrix to three indices of another matrix. For the purposes of the cropping algorithms I am designing, the three points I will use for this transformation will be the pixel locations of the left eye, right eye, and the nose. These points can be mapped to three "canonical points" which I can place on the N x M patch. The locations of these points can be configured manually, or derived using the intraocular distance between a subject's eyes. This distance would have to be determined manually by averaging a number of different values from subjects in the face database. After the affine transformation is complete, the N x M patch will contain a cropped face.

The location of the two eyes and the nose can be determined using OpenCV's object detections algorithms. Although the algorithms are currently configured to find a face in an image, they can easily be re-trained to find the location of the two eyes and the nose. According to the OpenCV documentation, this re-training will require an extremely large number of images (>10K). Although most of the images in the training set will be negative examples of each of the three landmarks, I may have to add more positive examples than those which can be extracted from the AR face database. Also, using non-AR face database images is desirable for the training because it will add variation to the landmark detection algorithms and reducing the chances of overfitting.

contain the face, from the tip of the chin to the top of the head.

3. Extract features from images by convolving each image with the Gabor filter. I will apply a variety of different Gabor wavelets at different orientations and frequencies.
4. Using the features extracted in III, construct the SVM kernels. The process for this construction can be found below.
5. Train the SVM algorithms using an image sequence that consists of positive and negative examples of features I am trying to identify. Initially, I propose only trying to detect frowns and smiles, so my training set will consist of positive and negative examples of both of these expressions.
6. Evaluate the performance of the feature extraction techniques using a test image sequence. Note, the test sequence *will not contain* any images that are included in the training sequence. This is an example of cheating from the computer vision perspective.
7. Analyze the results using receiver-operator curves (ROC)
8. If time permits, expand algorithms to identify other facial expressions.

III. Face Cropping

The first step in my classification process was to locate and crop the face region from an input image. The main reason I decided on this approach was due to the fact that since I am trying to classify facial expressions, the background of the image is not necessary and most likely adds noise to my classification process. If the input to my process is a frame containing only one face and no background objects, the feature extraction process becomes much easier and my classification will be more accurate.

My face-cropping algorithm was developed in MATLAB and utilizes a variety of algorithms provided in OpenCV 1.1 to find the bounding box of the face (see section IV). I chose to use cascading Haar filter banks to find the faces in my frame. The OpenCV algorithm (cvHaarDetectObjects) finds rectangular regions in a given image that are likely to contain objects the cascade has been trained to detect (in my cases faces). The algorithm scans the image several times at different scales. Each time it considers and overlapping regions in the image and applies the classifier to the regions [7]. I also have configured the algorithm to use Canny pruning to reduce the number of analyzed regions. Canny pruning uses the Canny edge detection algorithm to find regions in the image that contain too many edges for the object of interest. When these regions of the image are discovered they are thrown out and the complexity of the face detection problem is reduced significantly.

After determining the location of the face, the face has to be cropped to fit an N x M patch that contains only the face region. This cropping can be done fairly easily using an affine transformation. An affine transformation is a linear transformation that can map three indices of one matrix to three indices of another matrix. For the purposes of the cropping algorithms I am designing, the three points I will use for this transformation will be the pixel locations of the left eye, right eye, and the nose. These points can be mapped to three "canonical points" which I can place on the N x M patch. The locations of these points can be configured manually, or derived using the intraocular distance between a subject's eyes. This distance would have to be determined manually by averaging a number of different values from subjects in the face database. After the affine transformation is complete, the N x M patch will contain a cropped face.

The location of the two eyes and the nose can be determined using OpenCV's object detections algorithms. Although the algorithms are currently configured to find a face in an image, they can easily be re-trained to find the location of the two eyes and the nose. According to the OpenCV documentation, this re-training will require an extremely large number of images (>10K). Although most of the images in the training set will be negative examples of each of the three landmarks, I may have to add more positive examples than those which can be extracted from the AR face database. Also, using non-AR face database images is desirable for the training because it will add variation to the landmark detection algorithms and reducing the chances of overfitting.

contain the face, from the tip of the chin to the top of the head.

3. Extract features from images by convolving each image with the Gabor filter. I will apply a variety of different Gabor wavelets at different orientations and frequencies.
4. Using the features extracted in III, construct the SVM kernels. The process for this construction can be found below.
5. Train the SVM algorithms using an image sequence that consists of positive and negative examples of features I am trying to identify. Initially, I propose only trying to detect frowns and smiles, so my training set will consist of positive and negative examples of both of these expressions.
6. Evaluate the performance of the feature extraction techniques using a test image sequence. Note, the test sequence *will not contain* any images that are included in the training sequence. This is an example of cheating from the computer vision perspective.
7. Analyze the results using receiver-operator curves (ROC)
8. If time permits, expand algorithms to identify other facial expressions.

III. Face Cropping

The first step in my classification process was to locate and crop the face region from an input image. The main reason I decided on this approach was due to the fact that since I am trying to classify facial expressions, the background of the image is not necessary and most likely adds noise to my classification process. If the input to my process is a frame containing only one face and no background objects, the feature extraction process becomes much easier and my classification will be more accurate.

My face-cropping algorithm was developed in MATLAB and utilizes a variety of algorithms provided in OpenCV 1.1 to find the bounding box of the face (see section IV). I chose to use cascading Haar filter banks to find the faces in my frame. The OpenCV algorithm (cvHaarDetectObjects) finds rectangular regions in a given image that are likely to contain objects the cascade has been trained to detect (in my cases faces). The algorithm scans the image several times at different scales. Each time it considers and overlapping regions in the image and applies the classifier to the regions [7]. I also have configured the algorithm to use Canny pruning to reduce the number of analyzed regions. Canny pruning uses the Canny edge detection algorithm to find regions in the image that contain too many edges for the object of interest. When these regions of the image are discovered they are thrown out and the complexity of the face detection problem is reduced significantly.

After determining the location of the face, the face has to be cropped to fit an N x M patch that contains only the face region. This cropping can be done fairly easily using an affine transformation. An affine transformation is a linear transformation that can map three indices of one matrix to three indices of another matrix. For the purposes of the cropping algorithms I am designing, the three points I will use for this transformation will be the pixel locations of the left eye, right eye, and the nose. These points can be mapped to three "canonical points" which I can place on the N x M patch. The locations of these points can be configured manually, or derived using the intraocular distance between a subject's eyes. This distance would have to be determined manually by averaging a number of different values from subjects in the face database. After the affine transformation is complete, the N x M patch will contain a cropped face.

The location of the two eyes and the nose can be determined using OpenCV's object detections algorithms. Although the algorithms are currently configured to find a face in an image, they can easily be re-trained to find the location of the two eyes and the nose. According to the OpenCV documentation, this re-training will require an extremely large number of images (>10K). Although most of the images in the training set will be negative examples of each of the three landmarks, I may have to add more positive examples than those which can be extracted from the AR face database. Also, using non-AR face database images is desirable for the training because it will add variation to the landmark detection algorithms and reducing the chances of overfitting.

contain the face, from the tip of the chin to the top of the head.

3. Extract features from images by convolving each image with the Gabor filter. I will apply a variety of different Gabor wavelets at different orientations and frequencies.
4. Using the features extracted in III, construct the SVM kernels. The process for this construction can be found below.
5. Train the SVM algorithms using an image sequence that consists of positive and negative examples of features I am trying to identify. Initially, I propose only trying to detect frowns and smiles, so my training set will consist of positive and negative examples of both of these expressions.
6. Evaluate the performance of the feature extraction techniques using a test image sequence. Note, the test sequence *will not contain* any images that are included in the training sequence. This is an example of cheating from the computer vision perspective.
7. Analyze the results using receiver-operator curves (ROC)
8. If time permits, expand algorithms to identify other facial expressions.

III. Face Cropping

The first step in my classification process was to locate and crop the face region from an input image. The main reason I decided on this approach was due to the fact that since I am trying to classify facial expressions, the background of the image is not necessary and most likely adds noise to my classification process. If the input to my process is a frame containing only one face and no background objects, the feature extraction process becomes much easier and my classification will be more accurate.

My face-cropping algorithm was developed in MATLAB and utilizes a variety of algorithms provided in OpenCV 1.1 to find the bounding box of the face (see section IV). I chose to use cascading Haar filter banks to find the faces in my frame. The OpenCV algorithm (cvHaarDetectObjects) finds rectangular regions in a given image that are likely to contain objects the cascade has been trained to detect (in my cases faces). The algorithm scans the image several times at different scales. Each time it considers and overlapping regions in the image and applies the classifier to the regions [7]. I also have configured the algorithm to use Canny pruning to reduce the number of analyzed regions. Canny pruning uses the Canny edge detection algorithm to find regions in the image that contain too many edges for the object of interest. When these regions of the image are discovered they are thrown out and the complexity of the face detection problem is reduced significantly.

After determining the location of the face, the face has to be cropped to fit an N x M patch that contains only the face region. This cropping can be done fairly easily using an affine transformation. An affine transformation is a linear transformation that can map three indices of one matrix to three indices of another matrix. For the purposes of the cropping algorithms I am designing, the three points I will use for this transformation will be the pixel locations of the left eye, right eye, and the nose. These points can be mapped to three "canonical points" which I can place on the N x M patch. The locations of these points can be configured manually, or derived using the intraocular distance between a subject's eyes. This distance would have to be determined manually by averaging a number of different values from subjects in the face database. After the affine transformation is complete, the N x M patch will contain a cropped face.

The location of the two eyes and the nose can be determined using OpenCV's object detections algorithms. Although the algorithms are currently configured to find a face in an image, they can easily be re-trained to find the location of the two eyes and the nose. According to the OpenCV documentation, this re-training will require an extremely large number of images (>10K). Although most of the images in the training set will be negative examples of each of the three landmarks, I may have to add more positive examples than those which can be extracted from the AR face database. Also, using non-AR face database images is desirable for the training because it will add variation to the landmark detection algorithms and reducing the chances of overfitting.

IV. The Viola-Jones Object Detection Algorithm

The object detection algorithms provided in OpenCV were developed using an architecture originally proposed by Paul Viola and Michael Jones. This architecture can be trained to detect a variety of different objects, but for the purposes of this paper I use a configuration that was trained to find human faces.

Paul Viola's and Michael Jones' revolutionary object detection algorithm contains three unique features: 1) the introduction to a new image representation known as the "integral image" 2) a learning algorithm, based on AdaBoost, which selects a small number of critical features and yields extremely efficient classifiers 3) a method for combining classifiers in a "cascade" which allows background regions of the image to be quickly discarded while spending more computation on object-like regions [17].

The V-J algorithm (Viola-Jones) classifies images based on the value of features. The implementation in the OpenCV library uses three simple features: the two-rectangular feature, the three-rectangular feature, and the four-rectangular feature. An example of each of these features can be seen in **Figure 3**.

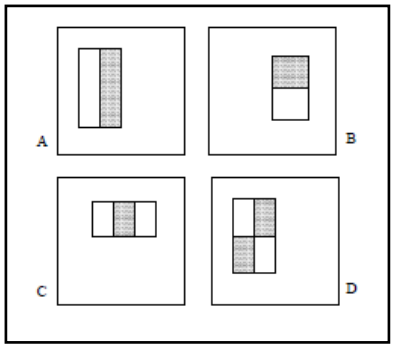

Figure 3: Features used by the VJ-Algorithm. These simple Haar filters combined with the integral image representation aid in the performance and speed of these algorithms.

The value of the two-rectangular feature is the difference between the sums of the pixels within the two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent. A three-rectangular feature computes the sum within the two outside rectangles subtracted from the sum of the center rectangle. Finally, a four-rectangular feature computes the difference between diagonal pairs of rectangles [17].

The rectangular features described above can be computed very rapidly using a new, intermediate representation known as the integral image. An example of the integral can be seen in **Figure 4**.

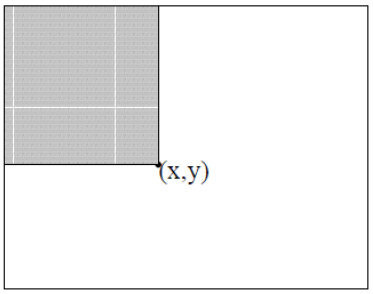

Figure 4: The definition of the Integral Image. This new representation revolutionized object detection frameworks.

The integral image at location (x, y) contains the sum of the pixels above and to the left of the (x, y) location, inclusive [17]:

$$IM(x,y) = \sum_{x<x' y<x'} ii(x',y')$$

Where IM is the integral image and ii is the original image. Using the following pair of recurrences:

$$s(x,y) = s(x,y-1) + ii(x,y)$$
$$ii(x,y) = ii(x-1,y) + s(x,y)$$

(Where s(x, y) is the cumulative row sum, s(x, -1) = 0, and ii (-1, y) = 0) the integral image can be computed in one pass over the image.

Using the integral image any rectangular sum can be computed in *four array references*. Since two of the rectangular features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangular feature, and nine in the case of the four-rectangular feature [17]. This reduced memory access significantly increases the performance of this system when detecting objects.

Although the features can be computed efficiently and with minimal computation, the feature space

using a scanning window is extremely large. Viola and Jones indicated that there are 45,396 rectangular features associated with each sub-window. Scanning an image beginning with a sub window of 24 x 24 pixels and increasing it based on the dimensions of the image will obviously create an extremely large and unmanageable amount of features. The main challenge was to find a small number of features that can be combined to create an effective classifier.

The V-J algorithm uses a variant of the AdaBoost algorithm used both to select the features and train the classifiers. In its original form, the AdaBoost machine-learning algorithm is used to boost the classification performance of a simple learning algorithm. It accomplishes this by combining a collection of weak classification functions to form a strong classifier [17]. In machine learning language, this is known as a "weak learner".

The conventional AdaBoost algorithm can be easily be interpreted as a greedy feature-selection process. Because AdaBoost is combining a large set of classification functions using a weighted majority, it is seeking to associate a large weight with the good classification functions and a smaller weight with the poor functions. AdaBoost is an effective procedure for searching out a small number of good "features" which nevertheless have a significant variety. Viola and Jones modified the traditional AdaBoost algorithms, making the weak classifier constrained so that each weak classifier returned can depend on only a single feature. In support of this goal, the algorithm is designed to select the single rectangular feature that best separates the positive and negative examples (faces) [17]. As a result of this modification, each stage of the boosting process, which selects a new weak classifier, can be viewed as a feature selection process.

The third unique aspect of the V-J algorithm is the cascade of classifiers used which increases the speed of the detector by focusing on regions of interest (sub-sections containing a face) while discarding all other sections of the image. The basic idea is that smaller and therefore more efficient, boosted classifiers can be constructed which are able to reject many of the negative sub-windows before the more complex classifiers are used to identify the face [17].

The algorithms implemented use a cascade of classifiers that were constructed using AdaBoost. The simple two-rectangular feature extract (from above) is used to extract features from each of the sub-windows. The thresholds of the classifiers are also adjusted to achieve minimum false-positive rates (i.e. high detection rates). Although the detection performance of this classifier is far from acceptable as an object detection system, it significantly reduces the number of sub-windows that need further processing. Similar classifiers to the one described above are trained using sub-windows that are not rejected by the previous classifier. As a result, each additional classifier faces a more difficult task than the one before it. Examples that make it through the first state are "harder" than typical examples. The more difficult examples faced by deeper classifiers therefore have high false-positive rates.

The overall form of the detection process is that of a degenerate decision tree. A positive result from the first classifier triggers an evaluation of a second classifier which has also been adjusted to achieve high detection rates. A positive result from the second classifier triggers a third classifier, and so on. A negative outcome at any point along the cascade of classifiers leads to an immediate rejection of the sub-window.

V. The Time-Frequency Problem

In all signal and image processing analysis, there is always a classical choice between spatial and frequency domain representations. For image processing, spatial representation consists of a two dimensional array of pixels. This is the typical format used for acquisition and display, but it is also common for storage and processing. You can think of spatial representation as a natural representation, and is important for applying common techniques such as texture analysis, shape analysis, and edge detection.

Although there are many useful applications in the spatial domain, there are many tasks that we can perform in the Fourier domain (spatial frequency) in a more natural way, such as filtering, correlations, etc [10]. Although the Fourier domain provides an accurate global description of the frequency content of an image, it does not tell us much regarding the

temporal locations of where those frequencies occurred.

Fourier analysis is one of the major accomplishments of physics and mathematics. It is indispensible to signal theory and signal processing for several reasons. First is the universal concept of frequency on which the analysis is rooted. In many applications, analysis in the frequency domain can be much more useful than analysis in the time domain. Secondly, the mathematical structure of the Fourier transform itself, as it is naturally suited to common transform methods (linear filtering) by its ability to render them in a particularly simple form [11]. Finally, from a pragmatic point of view, the collection of these advantages has led to a large number of programs, algorithms, processors, and machines use for frequency analysis, all of which contribute to its good reputation of practical use.

Although Fourier analysis is extremely important from a mathematical point of view, it possesses several restrictions concerning its physical interpretation and its range of applicability. Looking at the definition of the Fourier Transform:

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-i2\pi ft} dt$$

We can see that the computation of one frequency value $X(f)$ requires the knowledge of the complete history of the signal ranging from negative infinity to positive infinity. Conversely, looking at the definition of the inverse Fourier Transform:

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{+i2\pi ft} df$$

We can see that any instance of $x(t)$ at one instant of time can be regarded as an infinite superposition of complex exponentials, or everlasting and completely non-local waves [11]. Even if this mathematical point of view may reveal the true properties of a signal in certain cases ("quasi-monochromatic" situations, steady state, etc), it can also distort the physical reality. For example, with transient signals which vanish outside a certain time interval. Although the values which are null are reflected by the Fourier analysis, this only occurs in an artificial manner: the null values are a result of an infinite superposition of virtual waves that interfere such that they annihilate each other and sum to zero. Hence, the situation on the domain, where the signal vanishes, can be described as a "dynamic" zero (there exists waves whose resulting contribution is zero by interference). This contradicts any proper understanding of the real physical situation as a "static" zero (the signal does not exist) [11].

In quantum mechanics, the Heisenberg uncertainty principle states that certain pairs of physical properties, such as position and momentum, cannot both be known to arbitrary precision [12]. In other words, the more precisely a value is known, the less precisely the other value is known. To prove this fact, we start with signal that has bounded support in time (with duration T) and frequency (with bandwidth B). Any nonzero signal with these properties would satisfy the relation:

$$x(t) = \int_{-B/2}^{+B/2} X(f) e^{+i2\pi ft} df = 0 \quad |t| > T/2$$

If $x(t)$ is bounded in duration, then the same is true of its nth derivative. Also, using the fact that the signal of interest is bounded outside the bandwidth B, we can derive the following conclusion:

$$\frac{d^{(n)}x(t)}{dt^{(n)}} = \int_{-B/2}^{+B/2} (i2\pi f)^n X(f) e^{+i2\pi ft} df = 0 \quad |t| > T/2 \quad \forall\ n \geq 0$$

Now, the value of this signal at a point s, where s is outside the time domain bounds proposed above can be described by the following equation:

$$x(s) = \int_{-B/2}^{+B/2} X(f) e^{+i2\pi fs} df = 0 \quad |t| > T/2 \quad |s| < T/2$$

$$x(s) = \int_{-B/2}^{+B/2} X(f) e^{+i2\pi f(s-t)} e^{+i2\pi ft} df = 0 \quad |t| > T/2 \quad |s| < T/2$$

The power series expansion of the first complex exponential from above is:

$$e^{+i2\pi f(s-t)} = \sum_{n=0}^{\infty} \frac{[i2\pi(s-t)]^n}{n!} f^n$$

By substituting in this power series in the equations above, we arrive at the following equation:

$$x(s)=\sum_{n=0}^{\infty}\frac{[(s-t)]^n}{n!}\int_{-B/2}^{+B/2}(i2\pi f)^n X(f)e^{-i2\pi ft}df=0 \quad |t|>T/2$$

Since this hold for all $|s|<T/2$, it contradicts our original assumption of $|t|>T/2$. Even if we relax the constraints of the finite supports defined above, it is well known that the essential support of a signal cannot be arbitrarily small both in time and frequency. This type of constraint is imposed by the Fourier duality (which exists between the time and the frequency representations of signals) [11]. This is the well-known time-frequency problem.

This behavior of duality of the Fourier transform is defined in its simplest form in the *Heisenberg-Gabor uncertainty principle*. It was named after the uncertainty principles developed by Heisenberg in the 1920's in the context of quantum mechanics, and after Gabor, who performed similar studies after World War II in the area of communications theory and time-frequency analysis [11].

To establish this inequality, consider a signal $x(t)$ with finite energy:

$$E_x=\int_{-\infty}^{+\infty}|x(t)|^2\,dt<\infty$$

Also, assuming that the signal and its Fourier transform having vanishing centers of gravity (for simplicity):

$$\int_{-\infty}^{+\infty}t|x(t)|^2\,dt=0 \quad \int_{-\infty}^{+\infty}f|X(f)|^2\,df=0$$

We will use the following moments of inertia as measures of the time-frequency supports of the signal:

$$\Delta t^2=\frac{1}{E_x}\int_{-\infty}^{+\infty}t^2|x(t)|^2\,dt \quad and \quad \Delta f^2=\frac{1}{E_x}\int_{-\infty}^{+\infty}f^2|X(f)|^2\,df$$

And finally, we define the following auxiliary quantity:

$$I\equiv\int_{-\infty}^{+\infty}tx^*(t)\frac{dx}{dt}(t)dt$$

By using Parseval's identity and the Cauchy-Schwarz inequality, we can quickly derive the following:

$$[R\{I\}]^2\le|I|^2\le\left(\int_{-\infty}^{+\infty}t^2|x(t)|^2\,dt\right)\left(\int_{-\infty}^{+\infty}\left|\frac{dx}{dt}(t)\right|^2\,dt\right)=4\pi^2E_x^2\Delta t^2\Delta f^2$$

Using integration by parts [13], I can calculate the following value for integral I:

$$\int u\frac{dv}{dx}dx=uv-\int v\frac{du}{dx}dx$$

$$I=\left[\tfrac{1}{2}|x(t)|^2\right]_{-\infty}^{+\infty}-E_x-\int_{-\infty}^{+\infty}tx(t)\frac{dx^*}{dt}(t)dt=-E_x-I^*$$

$$\therefore$$

$$\mathrm{Re}\{I\}=-\frac{E_x}{2}$$

Using the results derived above we can construct the Heisenberg-Gabor uncertainty principle as follows:

$$[R\{I\}]^2\le 4\pi^2E_x^2\Delta t^2\Delta f^2$$

$$\frac{E_x^2}{4}\le 4\pi^2E_x^2\Delta t^2\Delta f^2$$

$$\frac{1}{16\pi^2}\le\Delta t^2\Delta f^2$$

$$\frac{1}{4\pi}\le\Delta t\ \Delta f$$

In the simplest terms, this inequality specifies that given a signal with duration of Δt and a bandwidth of Δf, the "duration-bandwidth" product is bounded from below. The Heisenberg-Gabor Uncertainty Principle sets the lower limit for joint time-frequency uncertainty. Viewed from a geometrical point of view, this means that that the two vectors in the inner product are collinear. In other words, the two signals $tx(t)$ and $(dx/dt)(t)$ must be proportional [11]. For elements of the vector space of real-valued signals, this implies the differential equation:

$$\frac{dx}{x}ktdt \quad k\in R$$

The solution to this equation is of the general form:

$$x(t) = Ce^{-\alpha t} \qquad (C,\alpha) \in R \; x \; R$$

Hence the Gaussian functions provide maximum joint localization and therefore achieve the lower bound of the joint uncertainty derived above. The conclusion played an important role Daugman's (etc all) decision to use the Gabor wavelet to model the receptive fields of the visual cortex.

VI. The Gabor Wavelet

Vision is considered to be a computation problem whose real-time solution would require parallel computers with performance in the order of 10^12 operations per second [2]. Past research has created a number of biologically motivated computationally intensive methods for image pattern recognition. The methods are biologically motivated in that scientists have tried to mimic and used mechanisms employed by biological vision systems, more specifically the visual systems of primates [3]. Since natural selection has provided adequate solutions to vision problems in man's natural environment, it would be reasonable to assume that such an approach would lead to an effective solution to the computation problem of facial recognition.

Extensive neurophysiological research in the past fifty years has led to the accumulation of considerable knowledge of the visual systems in primates and cats. Using this neurophysiological research, teams have set up computational models of the visual neurons that attempt to mimic their functional behavior [3]. Besides trying to understand and model the biological vision problems, other researchers have tried to use these computer models and simulations for building artificial vision systems to solve technical problems.

The primary visual cortex is the best studied visual area of the brain. It is the simplest, earliest cortical visual area. It is highly specialized for processing information about static and moving objects and is excellent in pattern recognition [5]. The research concerning simple cells in the primary visual cortex led to the widespread use of Gabor wavelets for facial recognition applications. The receptive fields of these cells were found to consist of a number of oriented altering parallel excitatory and inhibitory zones (see **Figure 1**). Although many scientists have attempted to mathematically describe the linear spatial summation properties zones, J.G Daugman proposed to model their spatial summation properties using two-dimensional Gabor functions that minimize the product of the variances in the space and spatial domains [4].

The Gabor functions were first discovered and proposed by Dennis Gabor as a tool for signal noise detection. Since 1980, several theorists concerned with spatial vision have recognized the suitability of Gabor's elementary signals as models of simple cell receptive field profiles. Typically two to five interleaved regions of excitatory and inhibitory influences weighted by a tapered envelop constitute the receptive-field profile of a simple cell, and Gabor signals with suitable chosen parameters invariably give a good fit to such spatial profiles [4].

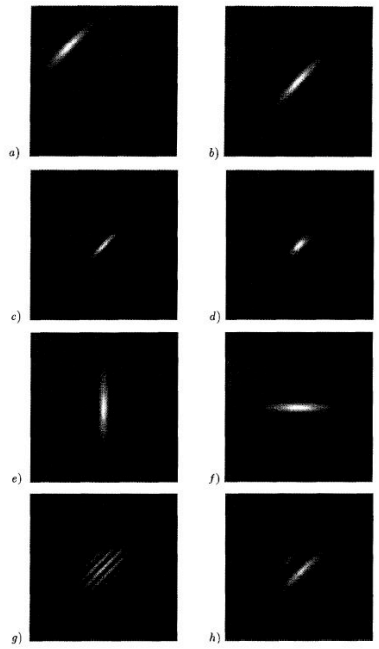

Figure 5: Gabor-like filters found in the human's visual cortex. These filters are the inspiration behind Gabor filters in image processing applications.

The Gabor filter is a linear filter used in image processing for feature detection. Its impulse response is defined by a harmonic function multiplied by a Gaussian function [6]. The two-dimensional Gabor wavelets is defined as follows:

$$g(x) = \frac{\|k_i\|}{\sigma} e^{\left[\frac{\|k_i\|^2 \|x_i\|^2}{2\sigma^2}\right]} \left(e^{jk_i x_i} - e^{\frac{\sigma^2}{2}} \right)$$

Where

$$k_i = \begin{pmatrix} k_{ix} \\ k_{iy} \end{pmatrix} = \begin{pmatrix} k_v \cos\theta_u \\ k_v \sin\theta_u \end{pmatrix}$$

In the equations above, k_v is the scale while θ_u is the orientation.

While optimal filter bank characteristics have been extensively studied for in speech recognition literature, little work has been done to systemically explore which frequencies and orientation bands are optimal for face recognition applications [15]. Bartlett and Fasel have done research in this area and discovered that the optimal characteristics for Gabor filter banks use eight orientation and five spatial frequencies (4:16 pixels per cycle at ½ octave steps) [14, 15]. For this project, I decided to initially use the frequencies and orientations provided in their paper. I plan on modifying these parameters if needed to achieve maximum performance. The frequency-domain representations of the forty different frequency/orientation combinations I have chosen initially can be seen below in **Figure 6**.

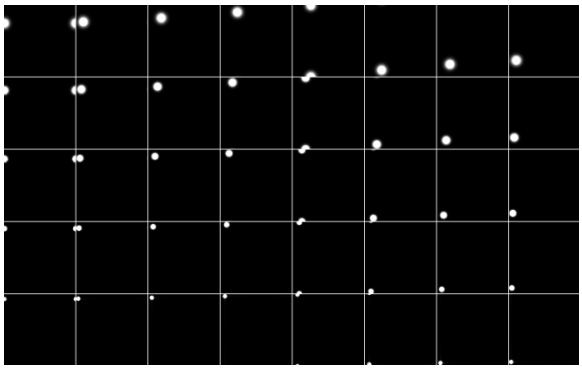

Figure 6: Frequency response of forty Gabor filters

The corresponding space-domain Gabor filter representations of the forty different frequency/orientations chosen can be seen below in **Figure 7**.

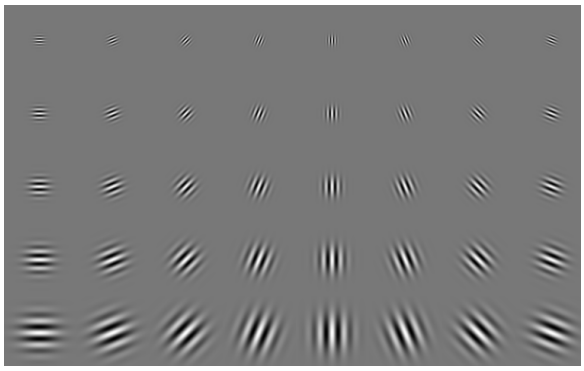

Figure 7: Gabor filters in the spatial domain

VII. Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression. A classification task usually involves testing and training data that consists of some data instances. Each instance in the training set contains one "target value" (class labels) and several "attributes" (features) [?]. The goal of an SVM is to produce a model that predicts target value of data instances in the testing set which are given only by attributes.

Mathematically, the SVM model represents each of the examples (training and testing) as points in some space. The points are mapped so that each example is divided by a clear gap (the hyperplane) that is as wide as possible. Each new example is than mapped into the same space that was created during the training sessions and predicted based on where it falls in relation to this hyperplane. An example of the support vector machine can be found below in **Figure 8**.

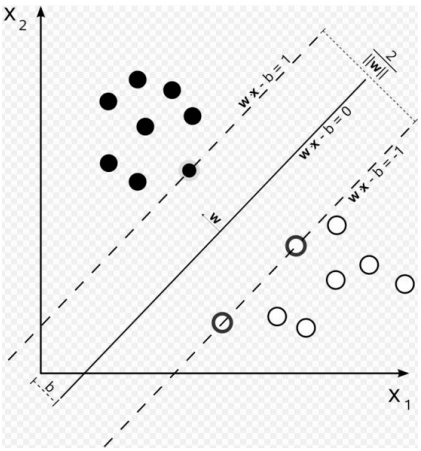

Figure 8: The hyperplane of the support vector machine (SVM). The algorithm seeks to maximize the distance between the data.

Given training data D, a set of n points of the form:

$$D = \{(x_i, c_i) | x_i \in \mathrm{R}^{\mathrm{p}}, c_i \in \{+1, -1\}\}_{i=1}^{n}$$

In the above equation, x_i is our training example and c_i is our training label, indicating the class to which x_i belongs. A hyperplane can be written as the set of points x satisfying:

$$w * x - b = 0$$

Where w is the normal vector (perpendicular to the hyperplane) and * denotes the dot product. The parameter $\frac{b}{\|w\|}$ specifies the offset of the hyperplane from the origin along the normal vector w. The goal of the model is to choose w and b to maximize the distance between the parallel hyperplanes while still separating the data. The hyperplanes in an SVM model can be described by the following equations:

$$w * x - b = 1 \quad and \quad w * x - b = -1$$

We also have to impose the constraint that that all of the data is outside the margins:

$$w * x - b \geq 1 \quad \forall \; x_i^1$$

And

$$w * x - b \geq 1 \quad \forall \; x_i^2$$

Where x_i^1 denotes data of class 1 and x_i^2 denotes data of class 2. This can be rewritten as the following:

$$c_i (w * x_i - b) \geq 1 \quad \forall \quad i = 1, ..., n$$

Finally, we form the following optimization problem:

$$\min_{w,b} \|w\| \quad subject\ to \quad c_i (w * x_i - b) \geq 1 \quad i = 1, ..., n$$

This optimization problem is difficult to solve because of the calculation of the norm of $\|w\|$ involves calculating a square root. By rearranging the formulation, we can alter the equation without changing the final solution and avoid this issue. The new formula becomes a quadratic programming problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad subject\ \ to \quad c_i (w * x_i - b) \geq 1 \quad i = 1, ..., n$$

Finally, using non-negative Lagrange multipliers α_i we can express the problem in a Lagrange formulation:

$$\min_{w,b,\alpha} \{\frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [c_i (w * x_i - b) - 1]\}$$

Expressing the problem in this form allows us to easily solve the problem using quadratic programming techniques. Since the training data only appears in the dot products between vectors, it also allows the problem to be generalized to non-linear cases. The solution can be expressed in terms of linear combinations of training vectors [?] as:

$$w = \sum_{i=1}^{n} \alpha_i c_i x_i$$

A critical step in SVM classification is choosing a suitable kernel. The kernel measures similarity between data points and must be positive definite. If the training data we are using is linearly separable,

the SVM can easily calculate two hyperplanes in a way that there are no points between them and try to maximize their distance. Although this is still an interesting case, in rarely happens in practice. When trying to build a model using training data that is not linearly separable, we need to extend the SVM to a non-linear case. This, in turn, means we have to transform or map the data to some other Euclidean space. This mapping is achieved using a kernel function.

A kernel is a mapping function that maps the inner products of the training data as follows:

$$x_i^T x_j \rightarrow \phi(x_i)^T \phi(x_j)$$

Since the training algorithms defined above only depend on training data in the dot products, we can use kernel function K such that:

$$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$$

Writing the classification rule in its unconstrained form reveals that the maximum margin of the hyperplane and therefore the classification task is only a function of the support vectors (the training data that lies on the margins). We can rewrite the Lagrange formulation as follows:

$$L(w,b,\alpha) = \frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i \left[1 - c_i (w * x_i + b)\right]$$

Now, since we seek to maximize the margins, take the first derivative and set it to zero:

$$\frac{dL}{dw} = 0$$

$$w - \sum_{i=1}^{n} a_i c_i x_i = 0 \quad note \quad \frac{dL}{dw}\left(\frac{1}{2} w^T w\right) = w$$

$$w = \sum_{i=1}^{n} a_i c_i x_i$$

Now substituting in the value of w into the original Lagrange formulation:

$$\frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i \left[1 - c_i (w * x_i + b)\right]$$

$$\frac{1}{2} w^T w + \sum_{i=1}^{n} \left[\alpha_i + \alpha_i c_i w x_i - \alpha_i c_i b\right]$$

$$\frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i c_i w x_i - \sum_{i=1}^{n} \alpha_i c_i b$$

$$\frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j c_i c_j x_i x_j^T + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_j \alpha_i \alpha_j c_i c_j x_i x_j^T - \sum_{i=1}^{n} \alpha_i c_i b$$

$$\sum_i \alpha_i - \sum_{i=1}^{n} \alpha_i c_i b - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j c_i c_j x_i x_j^T$$

Thus for $\sum_i \alpha_i c_i = 0$ and $\alpha_i \geq 0$ we arrive at:

$$\theta(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j c_i c_j x_i^T x_j$$

$$\theta(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j c_i c_j k(x_i, x_j)$$

The kernel here is defined as $k(x_i, x_j) = x_i^T x_j$.

Using the derivations from above, the decision function can be rewritten as:

$$f(x) = w * x - b = \sum_i \alpha_i c_i x_i^T x_j - b$$

And the update function can be rewritten as:

$$c_i \left(\sum_i \alpha_i c_i x_i^T x_j - b \right) \leq 0$$

Duality is an important feature of SVMs for two reasons: (1) some linear programming applications can take advantage of this property and solve these problems faster than primal (2) the duality property allows us to use the "kernel trick" and process high dimensional feature spaces at *no additional cost*.

VIII. Cross Validation

Cross-Validation is a technique for accessing how the results of a statistical analysis will generalize to an independent dataset [16]. It is generally applied in settings where the end goal of the process is a form of predication, where one is seeking to estimate how accurately a predictive model will perform in practice (in my case, on the testing data). Because of this

characteristic, it is an ideal tool to complement support vector machines.

Suppose we have a model with one or more known parameters, and a data set to which the model can be fit (training data set). The fitting process optimizes the model parameters to make the model fit the training data as well as possible. If we take an independent sample of validation data from the sample population as the training data, it will generally turn out that the model does not fit the validation data (testing data) as well as it fits the training data. This is called *overfitting*, and is particularly likely to happen when the size of the training data set is small, or when the number of parameters in the model is large. Cross validation is a way to predict the fit of the model to a hypothetical validation set when an explicit validation set is not available [16].

Cross-validation involves partitioning the training data into subsets, training the SVMs on one of the subsets and testing on the other subset. As long as the subsets are partitioned appropriately (the union between the testing and training subjects is null), this process will provide an accurate prediction of how the algorithms will perform blind testing set. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results (area under the roc curves) are averaged over the rounds.

The algorithms I developed for this project use K-fold cross validation. In K-fold cross validation, the original sample is randomly partitioned into K subsamples. Of the K subsamples, a single subsample is retained as the validation data for testing the model, while the remaining K-1 subsamples are used as the training data. The cross validation process is than repeated K times (the folds), with each of the K samples used exactly once as the validation set [16]. The K results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both the training and validation, and each observation is used for validation exactly once [16].

IX. AR Face Database

I developed and tested my face detection system using the AR Face Database [18]. The AR Face Database was created by Alexis Martinez and Robert Benavante in the Computer Vision Center (CVC) at U.A.B. It contains over 4,000 colored images corresponding to 126 people's faces (70 men and 56 women). Image feature frontal views with different facial expressions, illumination conditions, and occlusions (sunglasses and scarves). The pictures were taken under strictly controlled conditions. No restriction on wear (cloths, glasses, etc.), make-up, hair style, etc. were imposed on the participants. Each person participated in two different sessions, separated by two weeks. The same pictures were taken in both sessions [18].

The size of the images was originally 768 x 576. For the purposes of this project and my software, I converted each of the images to grayscale, and initially manually cropped out the face in a 96 X 96 frame. An example of the images after this processing can be seen below in **Figure 9**.

Figure 9: Example of faces found in AR face database.

X. Results

I began training classifiers for the following three facial expressions: smiles, neutral, and screams. I began with a total of 758 training images and initially

partitioned the datasets based on the subject IDs. The distribution of the data sets can be seen in **Figure 10**.

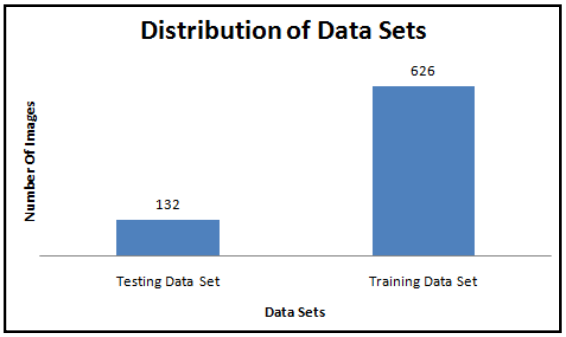

Figure 10: Distribution of training and testing sets

The distribution of the datasets based on sex can be seen in **Figure 11**.

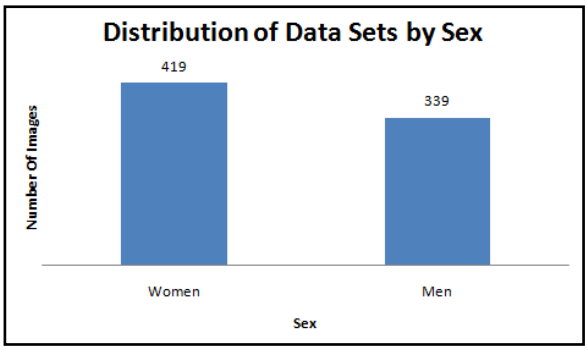

Figure 11: The distribution of the training set by sex

Effects of Gabor Orientation on Classifier Performance

I began by using a cropped image size of 72x72 pixels. I used three-fold cross-validation and calculated the average under the roc curves for each of the three folds. I generated my features and kernels using six different Gabor variances across three different numbers of orientations (4, 8, 16). The results from this analysis can be seen in the following three figures:

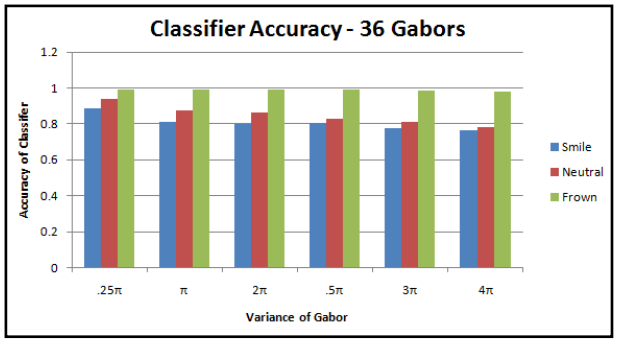

Figure 12: Performance of the algorithms using 36 different Gabor filters. Experiments were performed using an image size of 72 x 72.

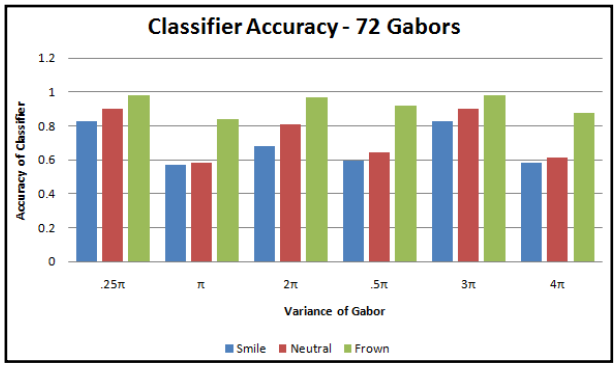

Figure 13: Performance of the algorithms using 72 different Gabor filters. Experiments were performed using an image size of 72 x 72.

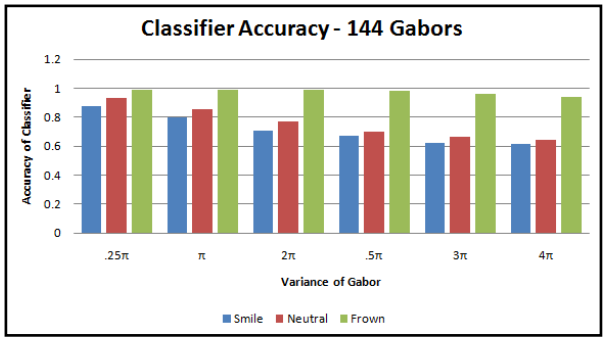

Figure 14: Performance of the algorithms using 144 different Gabor filters. Experiments were performed using an image size of 72 x 72.

My results indicate that my algorithm is able to detect the three emotions with an extremely high degree of

accuracy. All though all of the classifiers performed well (by machine learning standards), the scream classifier was able to detect the screaming facial expression over 99% of the time in each configuration I tested. Although this result was initially surprising, the features extracted from a screaming facial expression are very unique because of the shape of the mouth and the exposure of the teeth. I suspect this may have to do with the high degree of accuracy I am seeing in this classifier. My results also indicate that the optimal variance for the Gabor filters in the 72x72 feature space appears to be .25π. This is useful discovery because in the future I plan on hard-coding this value in the feature extraction algorithm and then searching for other optimal parameters. One surprising results from this analysis was that using four Gabor orientations (rotations of π/4) performed considerably better than eight and sixteen orientations. I suspect this is because the quadratic problem the support vector machine is attempting to solve is becoming significantly harder when you increase the size of the feature space (by adding orientations). This hypothesis can further be supported by monitoring the amount of time it takes the SVMs to solve the higher-order problems.

Next, I increased the dimensions of the image to 96x96 and performed similar analysis, varying the number of orientations and the variance of the Gabor filters. I use the new Gabor filters to extract features from my images, and perform three-fold cross-validation to determine the accuracy of the new settings. The results from these simulations can be seen in the following three figures:

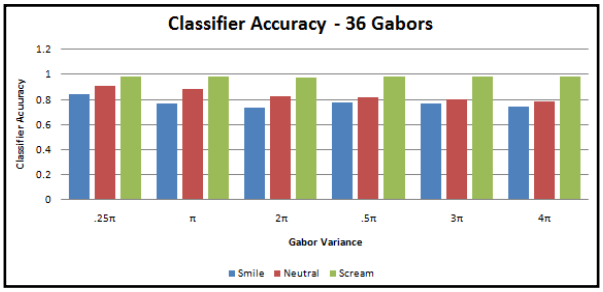

Figure 15: Performance of algorithms using 36 Gabor filters. Experiments were performed using an image size of 96 x 96.

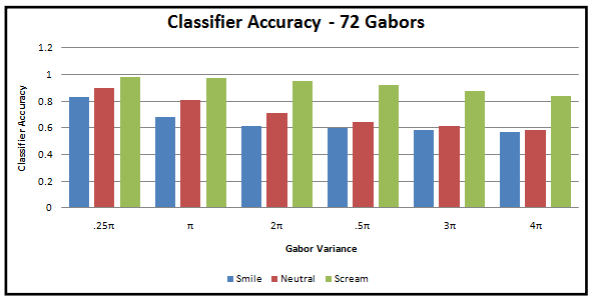

Figure 16: Performance of algorithms using 72 Gabor filters. Experiments were performed using an image size of 96 x 96.

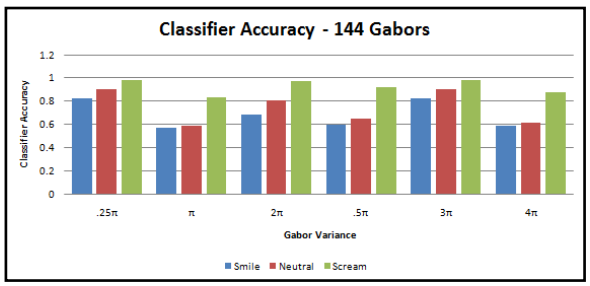

Figure 17: Performance of algorithms using 144 Gabor filters. Experiments were performed using an image size of 96 x 96.

My results for the 96x96 dimension images were almost identical to the results for the 72x72 dimension images: the classifiers performed best with a smaller number of orientations and with the Gabor variance set to .25π. Since the accuracy did not increase when I increased the size of the images, it would make sense for this system to use 72x72 size images rather than 96 x 96. It

Effects of Gabor Frequency on Classifier Performance

I investigated the effects of increasing and decreasing the number of frequencies the Gabor filters use and then monitoring the effects on the performance of the classifiers. Increasing the number of frequencies causes and increase in the size of the feature space. At some point, the feature space will become too complicated for the SVM to solve efficiently and the performance of the classifiers will begin to decrease.

The results of doubling the number of frequencies (to sixteen) can be seen below in **Figure 18**.

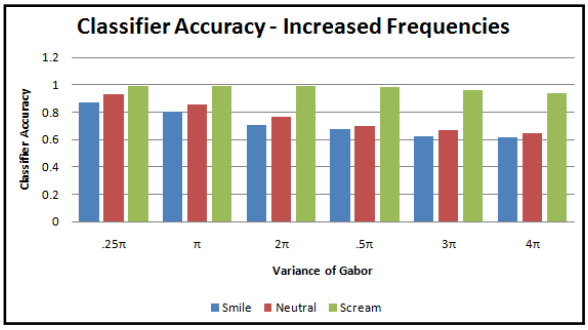

Figure 18: Performance of algorithms after doubling the number of frequencies used by the Gabor filters in the feature extraction process.

My results indicated the adding more frequencies to the process did not have a significant increase on the performance of the classifiers, for any of the expressions tested. Although I found this result somewhat surprising, it is extremely useful nonetheless.

Effects of Size of Training Set on Classifier Performance

I investigated the effects of the size of the training set on the performance of the three classifiers. Each training set contained roughly an equal number of positive examples of each of the three facial expressions (smile, scream, neutral). As the training sets got larger (greater than 800), I had to start using additional facial expressions from the AR face database. These samples would correspond to additional negative examples when training each of the three classifiers. For a performance metric, I used three-fold cross-validation across each of the six Gabor variances used previously. I averaged the area under the ROC curve for each of these folds. The results from cross-validations using different trainings can be found below in **Figure X**.

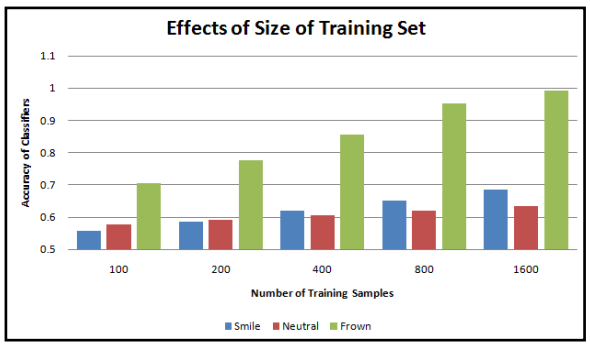

Figure 19: Effects of the size of the training set on the performance of the classifiers across three facial expressions.

My results indicate that as I increased the number of training samples, the performance of my classifiers increased. Although I have only shown the performance across five different training sessions, as I increased the size of the training set beyond 1600 samples, the performance of my algorithms reached a plateau. These results were anticipated, as increasing the number of examples will provide the SVM with more data tat will aid the algorithm when trying to solve the quadric problem of classification. Although adding more data to the training set will aid the SVMs, eventually we reached a point where additional data did not aid in solving this quadratic problem. In economics, this is known as the Law of Diminishing Marginal Returns. This point corresponds to the plateau in performance I saw after increasing the size of the training set beyond 1600 samples.

Effects of Size of Feature Space on Classifier Performance

I investigated the effects of applying a mask to the feature space prior to classification. The idea behind this method is the following: since I am attempting to classify three facial expressions (scream, neutral, and smiles), does the information above the nose (eyes, forehead, etc.) add unnecessary complexity to the problem I am trying to solve. It seems reasonable to think that the SVM only needs to see the lips to classify a smile, rather than the whole face. The same can be said about neutral faces and screaming. I added a function to my algorithms that will throw out the top of the head (anything above the nose) after

the feature extraction process. This results a reduction of the feature space and kernel space significantly, and quadratic problem the SVM is attempting to solve should become much simpler. The results from the feature mask can be found in the following figures.

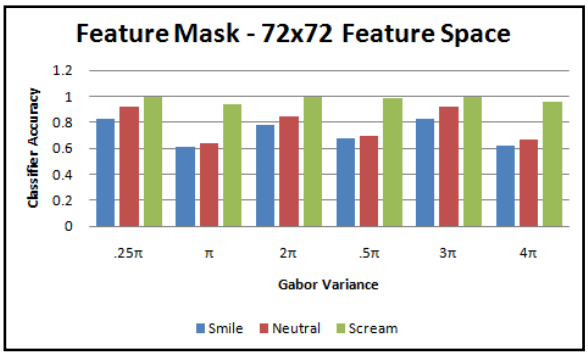

Figure 20: Performance of algorithms using feature-reduction algorithms. The image size used in this experiment was 72x72.

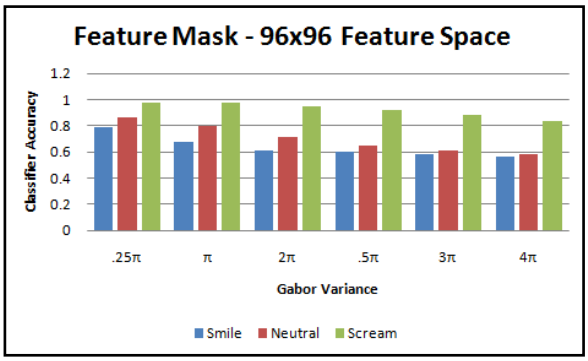

Figure 21: Performance of algorithms using feature-reduction algorithms. The image size used in this experiment was 96x 96.

My initial results indicate that pruning the feature space caused the performance of the classifiers to degrade in both 72x72 and 96x96 dimensions. I believe this is due to the fact that the classifiers are using areas above the nose (the areas of the feature space I removed) to separate the hyperplanes. More testing can be done to find areas of the feature space that are redundant for classification purposes.

Effects of Number of Folds on Classifier Accuracy

I investigated the effects on the number of folds chosen for the cross-validation process on the performance of the classifiers. For the analysis, I used 72x72 size images and 72 Gabor filters (8 orientations and 9 frequencies) for the feature extract process. My algorithms randomly selected subjects from the entire image database for each of the N folds. During the training process, my algorithms would make sure that each cross-validation (training and testing) contained both positive and negative examples of each of the three facial expressions. This test ensures the cross-validation are valid, as the algorithm was provide valid data to construct the classification models. The performance metric chosen was average area under the ROC curves across the six Gabor variances used in previous analysis. The results for the fold analysis can be found below in **Figure X**.

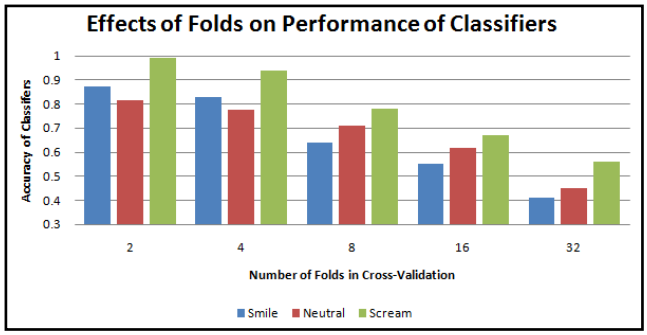

Figure 22: Performance of the classifiers after varying the number of folds used in the cross-validation process.

My results indicate the changing the number of folds had a significant effect on the performance of the classifiers. Specifically, using a smaller number of folds in the cross-validations yielded much better results than the large fold values. A somewhat surprising result was that as the number of fold increased to size 32, the classifiers used to detect smiles and neutral expressions degraded to below 50%. This means that a human guessing the results would yield a better performance (theoretically 50%) than the performance of these classifiers. Digging a little deeper in the analysis, I discovered that these results actually made sense. As the number of folds increases, the size of the training and testing sets used the in the cross-validations decreases. The terrible results exhibited when 32 folds were used were most likely a result of an extremely small number of

positive examples of the smile and neutral faces in the testing sets of the cross-validations. A small number of positive examples mean that one missed classification can have significant effects on the area under the ROC curves for these classifiers. A smaller number of folds yield a larger testing set in the cross-validation process, and that is why I believe I am seeing the corresponding increase in performance with the lower numbers.

XI. Conclusions and Further Work

For this project, I constructed (from scratch) a highly accurate facial expression recognition system using Gabor wavelets for feature extraction and support vector machines for the classification process. Although I chose to only analyze my performance across three different facial expressions (neutral faces, smiles, and screams), the framework I developed could be easily extended to support and classify any number of different facial expressions, provided that you had a database large enough to effectively train the support vector machines used in my system.

I analyzed the performance of my algorithms across a larger number of different parameters and provided insight into the results I obtained. Overall, the algorithms were extremely successful at classifying each of the three facial expressions I targeted. Specifically, after finding the optimal configurations for the software, *I was able to obtain classification accuracies of over 85% for both smiles and neutral faces, and over 99% for faces containing screams!*

Although I made a considerable amount of progress developing the algorithms needed for a face classification system, I did not end up having time to complete my face-cropping algorithm. The algorithms used in my training data were generated using a manual face-cropping tool I developed. Although I completed most of the code for the face cropping algorithms, I ran out of time training my landmark detectors: I decided to use the locations of the left and right eye, as well as the top of the nose as my three landmarks I would use for the cropping. To detect these three locations, I was going to use the OpenCV object detection frame (Haar cascades). Since OpenCV did not provide filters to detect these locations, I was going to have to train the classifiers to detect the three locations before my cropping algorithms could be complete and tested.

Although this strategy would no doubt be successful, training the OpenCV required much more work than I anticipated. Not only did the trainings sets for these algorithms have to be larger than ten thousand images, training each of the three landmark classifiers took up to three days on my laptop. The performance could only be analyzed every 1.5 weeks, and after three weeks of training, I have still not achieved the performance need to effectively locate and crop the faces in images from the AR face database. Even though I did not have time to complete my face cropping algorithms, I provided a description of how the algorithms would work in section III.

Also, although my results were generated using a training and tested set which contained manually cropped faces, it is highly likely that after developing an effective face cropping scheme, the performance of my fully-automated system would be very similar to the results published in this paper. Detect faces in static environments (such as the environment of the AR face data) is considered to be a mostly solved problem in computer science. Located and cropping the faces effectively and with high accuracy is the last piece software needed to complete a fully-automated facial expression classification system.

Finally, also worth noting is that although I achieve extremely accurate results in the tests performed for this paper, these results would certainly degrade if my system was tested in a real-life environment (aka security camera, etc). Face cropping and expression classification using posed subjects such as those found in the AR face database is much easier to classification and detection using real camera feeds. The complexity of the problem comes from all of the noise added to the process (other people, other objects, etc.). If this system were to be adapted to different type of conditions, the training set would have to expand significantly and more work would have to go into making the algorithms more real-time. Face tracking might be a nice addition in more real-life applications.

XII. Software & Databases

OpenCV 1.1: OpenCV is a computer vision library originally developed by Intel. It focuses mainly on real-time image processing. I use this open-source software for my face-detection and face-cropping schemes. The OpenCV software and documentation can be found here: http://sourceforge.net/projects/opencvlibrary/

LIBSVM: LIBSVM is an integrated software module for support vector classification. It supports multi-class classification. I used the open-source library for the machine-learning aspects of the project. The code and documentation for these libraries can be found at: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Flickr API: Flickr is an image and video hosting website, web-service suite, and online community. With over 4 billion downloadable images, this site provides all of the data I need to train and test my image classification algorithms. The Flickr API provides a accessible interface for me to download images in an automated way. The documentation and software for accessing the API can be found here: http://www.flickr.com/services/api/

MATLAB Image Processing Toolbox: I use the functionality provided in MATLAB's image processing toolbox to perform all of my image loading, filtering, and manipulation.

XIII. References

[1] Feris, R and Cesar, R. *Locating and Tracking Facial Landmarks Using Gabor Wavelet Networks*.

[2] Kepenekci, B. *Facial Recognition Using Gabor Wavelet Transform*

[3] Petkov, N. *Biologically motivated computationally intensive approaches to image pattern recognition.*

[4] Daugman, J. *Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters.*

[5] "Visual Cortex". *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. Web. http://en.wikipedia.org/wiki/Visual_cortex

[6] "Gabor Filter." *The Free Encyclopedia*. Wikimedia Foundation, Inc. Web. http://en.wikipedia.org/wiki/Gabor_filter

[7] "Experimental Functionality Reference". http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_Experimental.htm#decl_cvHaarDetectObjects

[8] "Support Vector Machine." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Support_vector_machine

[9] Dula, R. Hart, P. Stork, D. "Pattern Classification"

[10] Navarro, R. Tabernero, A. Cristobal, G. "Image Representation Using Gabor Wavelets and Its Applications"

[11] Flandrin, P. "Time-Frequency/Time-Scale Analysis"

[12] "Uncertainty Principle". *The Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Uncertainty_principle

[13] "Integration by Parts". *The Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Integration_by_parts

[14] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[15] Fasel, I. Bartlett, M. Movellan, J. A Comparison of Gabor Filter Methods for Automatic Detection of Facial Landmarks

[16] "Cross Validation (statistics)". *The Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Cross-validation_(statistics)

[17] Viola, P., Jones, M. "Robust Real-Time Object Detection"

[18] A.M. Martinez and R. Benavente. *The AR Face Database*. CVC Technical Report #24