K-Means Clustering For Detection of Human Faces in Databases

by

Nancy Smith
smitnanc@nova.edu
N00869580

A research paper submitted in partial fulfillment of the requirements for CISD794

Graduate School of Computer and Information Sciences
Nova Southeastern University

2009

An Abstract of a Research Paper Submitted to Nova Southeastern University
in Partial Fulfillment of CISD794

K-Means Clustering for the Detection of
Human Faces in Databases

by
Nancy Smith

December 5, 2009

The ability to segment faces out of images in databases is an important initial step in many important applications, especially those requiring face recognition. The k-means algorithm has been shown to be effective in segmenting face candidates, which are then classified as faces, typically using the existence of facial features, such as eyes and mouths.

The research in this paper also used k-means to segment eye and mouth candidates within the face candidates. The principal component eigenvectors of these images were used to train three neural networks: one to classify faces, one to classify eyes, and one to classify mouths. The classifications achieved an 88.9%, 85.4%, and 97.8% success rate, respectively. However, many of the images did not generate any valid candidates for classification. The k-means algorithm was able to segment valid face candidates in 97% of the cases, but 20% of the images failed to generate any valid eye or mouth candidates.

Table of Contents

List of Figures

Figures

Chapter 1

Introduction

Statement of the problem

An important step in pattern recognition systems is segmentation of an image to detect the existence of an interesting pattern, and the isolation of the cluster containing the pattern. The process of clustering is composed of pattern representation, which may include feature extraction and/or selection, defining a pattern proximity measure, clustering, and possibly data abstraction and assessment (Jain, Murty, & Flynn, 1999). These steps are clearly domain dependent. The domain of this project is clustering high dimensional image data. In particular, the project focuses on segmenting a photographic image to detect a human face.

There is a tremendous amount of ongoing research in this area since there are many important applications that could benefit from facial analysis. Once a face has been segmented out of an image, it can be classified as a specific person, gender, emotion, behavior, etc. The security implications, marketing opportunities, and ability to enhance human-computer interaction cannot be underestimated. Police mug shot albums could be scanned for individuals; family albums could be scanned for a particular individual; security clearance for entry into buildings could be validated by scanning a database of authorized personnel; and search engines could locate unlabeled images.

Objectives

This research proposes a clustering methodology to increase the accuracy rate of face detection by using principal component analysis (PCA) on features extracted from the face candidate clusters. K-means is performed on the original image to segment out potential faces based on the lab color space. K-means is then performed using the grayscale intensity space to segment potential eyes and mouths from the face segments.

The primary hypothesis is that since eyes and mouths show up as dark clusters, or holes, inside face clusters, K-means will be able to effectively segment these facial features for classification.

The secondary hypothesis is that since there is less irrelevant background information in the eye and mouth segments than in the full face images, PCA will be able to encode the differences in these patterns more effectively.

Chapter 2

Review of the Literature

Existing Methodologies for Face Detection

The primary technique for clustering an image to detect faces is based on skin color. The color space and similarity measure are major variables in the research. Similarity measures include simple distance rules, Bayesian classifiers, self-organizing maps, and Gaussian joint probability density functions (Vezhnevets, Sazonov, & Andreeva, 2003). Color spaces include RGB, CMY, YCbCR, HIS (HSV), YIQ, YUV.

One of the more common algorithms used for clustering faces in recent research is based on the research by Viola & Jones (2004), and involves representing the image as Harr-like features, which can be computed at any scale or location in constant time. This algorithm also uses AdaBoost to select a small number of features, and an optimization method to focus subsequent processing on promising regions.

Grangero, Jesus, & Correia (2009) use the Viola-Jones algorithm with a skin filter based on the RGB color space to accept or reject the presence of a face in a sub-window indicated by the face detection algorithm. They also propose an algorithm for pose estimation using different AdaBoost classifiers (Support Vector Machines (SVM's)) trained with frontal or profile only images.

Ruan & Yin (2009) also use the Viola-Jones algorithm with a skin filter, but in the YCbCr color space which is less sensitive to lighting variations than RGB. They use an SVM in the clustering process to rule out non-face regions, but use the eyes and mouth to

verify the face candidates. The color difference is used to detect the eyes and mouth, and the geometric relationship is used to verify the face region. Since their dataset contained faces of varying sizes, their method also incorporated resizing and merging clusters.

Ying-hui, Xiao-juan, Chun-xia, & Ojong-fang (2009) perform preprocessing to remove highlighting and shadows caused by lighting. The image is segmented using a gray-level threshold to get the face candidates. The candidates are validated using holes, area, center of mass, aspect ratio, and template matching.

Lekshmi, Kumar, & Vidyadharan (2008) used the k-means clustering algorithm to detect candidate faces. Histograms were used to select the initial cluster centers. The existence of facial features with appropriate distances relative to one another was used to validate the cluster as a face.

Kobayashi & Zhao (2007) also used the k-means clustering algorithm. The images were preprocessed to compensate for lighting variances. After clustering, linear discriminate analysis (LDA) was applied, and a neural network validated whether the candidate was a face. This method was compared to using principal component analysis in place of LDA, and also to using neural networks alone. The researchers also varied the value of k in the k-means algorithm and concluded the error rate was significantly reduced with an increased number of clusters, settling on a value of 30 in their published results.

Significance of research

The research described in this paper is based on the observation that Kobayashi et al. (2007) applied LDA and PCA validation techniques on the entire face cluster, while

the other researchers typically use facial features in their validation. It is known that the eyes and mouth are the most distinguishing features indicating the presence of a face. By reducing the initial dimension, from full face to eyes and mouth, much of the irrelevant data is removed from the validation process. PCA can reduce the dimensionality even further, capturing the most salient aspects of facial features.

PCA is used instead of LDA, since PCA is more effective than LDA with a limited number of samples (Vezhnevets, et al. 2003). The number of samples required to train a classifier increases exponentially with the dimensionality of the sample. Bishop (1995) refers to this as the 'curse of dimensionality.' Even after PCA has been applied, the dimensionality of the input feature vector is still fairly high. This is supported by Kobayashi et al. (2007), with PCA yielding lower error rates than LDA.

Barriers and Open Problems

Most face detection algorithms perform well on normalized databases, but their performance deteriorates significantly with occlusions, variations in lighting, poses, and facial expressions (Grangeiro, et al. 2009).

Chapter 3

Methodology and Algorithms

Methodology

The research described in this paper used k-means clustering to obtain face candidates. Clustering was then performed on the face candidates to obtain eye and mouth candidates. PCA was performed on the face, eye, and mouth candidates to form a feature vector that was used as input to a neural network (NN). There are three NNs trained for binary classification: face or not-face, eye or not-eye, mouth or not-mouth. The results were compared to determine the advantage of using facial features over the entire face for classification.

The remainder of this section discusses k-means, principal component analysis, and neural networks as they relate to this project. The following section contains more specific information on the procedures used in this experiment.

K-means

Image segmentation exhaustively partitions an image into multiple regions. It is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. Pixels are comprised of attributes such as location, color, intensity, and texture. Pixels are the objects that are clustered using the k-means algorithm.

K-means clustering is a method for finding clusters and cluster centers in a set of unlabeled data. The k-means algorithm partitions a set of n objects into k clusters so that

the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured with respect to the mean of the objects in the cluster.

Similarity Measure

Similarity is expressed in terms of a distance function. Two closely related vectors have a small distance and a large similarity. It is important to keep the similarity measure simple since the algorithm repeatedly calculates the similarity of each pixel to the mean of each cluster. Image segmentation uses a weighted distance measure using pixel coordinates, color and/or intensity. Consider that we have measurements x_{ij} for i=1,2,..N, on variables j=1,2,…,p (the pixel attributes). We define a dissimilarity $d_j(x_{ij}, x_{i'j})$ between values of the jth attribute, and then define

$$D(x_{i,} x_{i'}) = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j})$$

as the dissimilarity between objects i and i'. The most common choice, and the one used in our image segmentation, is the squared distance:

$$d_j(x_{ij},x_{i'j})=(x_{ij}-x_{i'j})^2.$$

To combine the p attribute dissimilarities into a single overall measure of dissimilarity $D(x_i, xi.)$ between two objects $(x_i, x_{i'})$, a weighted average is used:

$$D(x_i, x_{i'}) = \sum_{j=1}^{p} w_j * d_j(x_{ij}, x_{i'j}); \quad \sum_{j=1}^{p} w_j = 1$$

where w_j is a weight assigned to the jth attribute regulating the relative influence of the variable in determining the overall dissimilarity between objects.

Criterion Function

The within cluster point scatter can be written as:

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} \| x_i - x_{i'} \|^2$$

$$= \sum_{k=1}^{K} N_k \sum_{C(i)=k} \| x_i - \bar{x}_k \|^2$$

where $\bar{x}_k = (\bar{x}_{1k},...,\bar{x}_{pk})$ is the mean vector associated with the kth cluster, and $N_k = \sum_{i=1}^{n} I(C(i) = k)$. Thus, the criterion is minimized by assigning the n objects to the k clusters in such a way that within each cluster the average dissimilarity of the objects from the cluster mean, as defined by the points in that cluster, is minimized. This makes clusters as compact and as separate as possible.

K-means algorithm

The k-means algorithm is comprised of the following steps:

1. Choose k cluster centers, either randomly or based on heuristics.
2. For a given cluster assignment C, minimize the total cluster variance

$$\min_{C,\{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} \| x_i - m_k \|^2$$

with respect to {m1,…mk} yielding the means of the currently assigned clusters

$$x_S = \arg\min_m \sum_{i \in S} \| x_i - m \|^2$$

3. Given a current set of means {m1,…,mk}, the total cluster variance is minimized by assigning each object to the closest current cluster mean.

$$C(i) = \arg\min_{1 \le k \le K} \| x_i - m_k \|^2$$

4. Repeat steps 2 and 3 until the criterion function converges. (No pixels change clusters).

K-means Issues

The k-means algorithm suffers from the problem of local optima. The quality of the solution depends on the initial cluster centers. Thus, it is important to either choose good initial centers or to rerun the algorithm several times and keep the best solution. Quality functions that measure intracluster similarity and intercluster dissimilarity are used to determine the best solution.

Another issue with k-means is that choosing the number of clusters is difficult. Milligan & Cooper (1985) compared over 30 rules for estimating the optimal number of clusters, and concluded that there is no best solution. The best is data dependent. Some are good only in specific examples.

The complexity of k-means is O(nkt) where n = number of objects, k= number of clusters, and t = number of iterations. For image segmentation, n is quite large and processing time is sensitive to the number of clusters. Because most of the convergence takes place in the early iterations, the condition in step 4 of the algorithm noted above is often replaced with a weaker condition, e.g., repeat until only 1% of the pixels change clusters.

Principal Component Analysis

Feature extraction plays an essential role in pre-processing the images before input into a classifier. There are several reasons that it is undesirable to use very large vectors comprised of each set of pixel attributes for classification, primary among these is overfitting or losing the ability to generalize.

Principal Component Analysis was used to reduce the dimensionality of the data while maintaining the maximum information about the patterns in the data. The idea

behind PCA is to express the large one-dimension vector of pixels constructed from the two-dimension facial images into compact principal components of the feature space. This is called the Eigenspace projection. It is calculated by identifying the eigenvectors of the covariance matrix derived from a set of facial images.

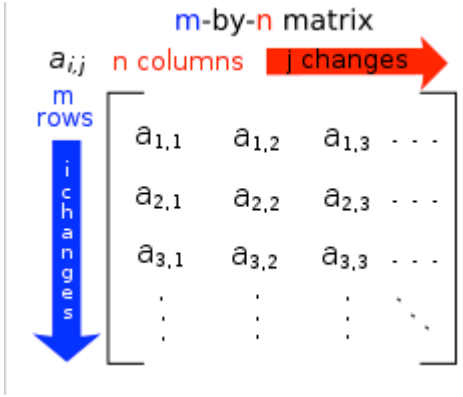
Consider an image of 256x256 pixels. It can be viewed as a point in 65,536-dimensional space. An ensemble of images maps to a collection of points in this huge space. Faces have many similarities and the distribution will not be random. The eigenvectors describe this subspace of facial images. For a more formal definition, let A be a square matrix. A non-zero vector C is called an eigenvector of A if and only if there exists a number (real or complex) λ such that $AC = \lambda C$. The value λ is called an Eigenvalue.

Principal Component Analysis steps

The following steps were used to extract eigenvectors of the principal components:

1. Convert the images into a matrix of vectors.

   Let $X = (a_1\ a_2\ \ldots\ a_{k*k})$ represent a single image of k^2 pixels.
   Let $I = [\ X_1..X_m\ ]^T$ represent a matrix of m images of form X, where m=number of images, n=number of pixels per image (k^2).



2. Subtract the mean image from each image vector.

   Let ψ represent the mean image.

$$\psi = \frac{1}{m} \sum_{i=1}^{m} X_i$$

   where m is the number of image vectors.

   Let w_j represent the mean centered image

$$w_j = X_i - \psi$$

3. Calculate the covariance matrix.

$$C = WW^T$$

   where W is a matrix composed of column vectors w_j placed side by side.

3. Calculate the eigenvectors and eigenvalues of the covariance matrix

   Our goal is to find a set of e_i's which have the largest possible projection onto each of the w_i's. We want to find a set of m orthonormal vectors e_i for which the quantity

$$\lambda_i = \frac{1}{M} \sum_{n=1}^{M} (e_i^T w_n)^2$$

   is maximized with the orthonormality constraint

$$e_i^T e_k = \delta_{ik}$$

   It has been shown that the e_i's and λ_i's are given by the eigenvectors and eigenvalues of the covariance matrix C. Since the size of C is very large, it is not practical to solve for the eigenvectors of C directly. A common theorem in linear algebra states that the vectors e_i and scalars λ_i can be obtained by solving for the eigenvectors and eigenvalues of the matrix W^TW.

   Let d_i and u_i be the eigenvectors and eigenvalues of W^TW

$$W^TWd_i = u_id_i$$

   By multiplying both sides by W

$$WW^T(Wd_i) = u_i(Wd_i)$$

Which means the first M-1 eigenvectors e_i and eigenvalues λ_i of WW^T are given by Wd_i and u_i.

4. Sort according to the eigenvalues

   The eigenvector with the largest eigenvalue reflects the greatest variance in the image. They decrease exponentially, so that approximately 90% of the total variance is contained in the top 5% to 10% of the eigenvectors. These are the principal components.

Neural Network Classification

Neural networks can be trained to perform complex non-linear functions, such as pattern recognition. This research employs a feedforward neural network trained with back propagation. Input vectors (eigenvectors) and the corresponding target vectors (emotions) are used to train the network.

The problem of learning in a neural network can be framed as minimization of an error function E (Bishop, 1995). The error is a function of the weights and biases in a network, which can be grouped together into a single W-dimensional weight vector $w_1..w_W$. The training algorithm used in this project is a variation of gradient descent called scaled conjugate gradient.

Gradient Descent

The gradient descent algorithm searches along the direction of steepest descent, and the weights are updated using

$$\Delta w^r = -\eta \nabla E^n \big|_{w^r} \qquad (1)$$

where η is the learning rate, and provided it is sufficiently small, the value of E will decrease each step leading to a minima where the vector $\nabla E=0$. There are problems with

convergence with the simple gradient descent algorithm, however. It is difficult to find a suitable value for η. The error surface may contain areas where most points do not point towards the minimum, resulting in a very inefficient procedure. The basic algorithm can be enhanced by adding a momentum term μ to smooth out the oscillations, and by updating the learning rate (Bishop, 1995).

Conjugate Gradient Descent

Another issue with gradient descent is choosing a suitable search direction. Suppose we have minimized along a line given by the local gradient vector. Choosing successive search directions can lead to oscillations while making little progress toward the minimum. For this problem, conjugate gradients are employed. Suppose a line search has been performed along the direction d^r starting from point w^r to give an error minimum along the search path at the point w^{r+1}. The direction d^{r+1} is said to be conjugate to the direction d^r if the component of the gradient parallel to the direction d^r, which has been made zero, remains zero as we move along the direction d^{r+1}. It can be shown that the minimum of a general quadratic error function can be found in at most W steps using conjugate gradients (Bishop, 1995).

Scaled Conjugate Gradient

A basic problem with line search is that every line minimization involves several error function evaluations, each of which is computationally expensive. The procedure also involves a parameter whose value determines the termination criteria for each line search. The performance is sensitive to this value. The scaled conjugate gradient algorithm avoids the expense of line minimization by evaluating Hd_j where H is the Hessian matrix comprised of the second derivatives of the error

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}$$

However, it it necessary to ensure that H is positive definite so that the denominator doesn't become negative and thus increase the error. This is done by adding a multiple of the unit matrix

$$H + \lambda I$$

Where I is the unit matrix and $\lambda \geq 0$ is a scaling coefficient. The formula for the step length is then given by

$$\alpha_j = -\frac{d_j^T g_j}{d_j^T H_j d_j + \lambda_j \| d_j \|^2}$$

where d_j is the direction at step j, and g_j is the gradient vector at the jth step orthogonal to all previous conjugate directions. The suffix j on λ_j reflects that the optimum value for this parameter can vary on each iteration. Techniques like this are well known in standard optimization where they are called model trust regions. The model is only trusted in a small region around the current search point. The size of the trust region is controlled by λ_j so that for large λ_j the trust region is small. In regions where the quadratic approximation is good, the value of λj should be reduced, while if the quadratic approximation is poor, λ_j should be increased. This is achieved by considering the following comparison parameter

$$\Delta_j = \frac{2\{E(w_j) - E(w_j + \alpha_j d_j)\}}{\alpha_j d_j^T g_j}$$

The value of λ_j is then adjusted with

$$\text{If } \Delta_j > 0.75 \text{ then } \lambda_{j+1} = \frac{\lambda_j}{2}$$

$$\text{If } \Delta_j < 0.25 \text{ then } \lambda_{j+1} = 4\lambda_j$$

$$\text{Else } \lambda_{j+1} = \lambda_j$$

If $\Delta_j < 0$, the step would actually increase the error so the weights are not updated, but instead the value of λ_j is increased and Δ_j is re-evaluated. Eventually an error decrease will occur since once λ_j becomes large enough, the algorithm will be taking a small step in the direction of the negative gradient. The two stages of increasing λj if required and adjusting λ_j are applied in succession after each weight update (Bishop, 1995).

Hidden Layers

There is no theoretical reason to ever use more than two hidden layers, and for the majority of practical problems, there is no reason to use more than one hidden layer. The problems with multiple hidden layers include longer training times, the gradient is more unstable, and the number of false minima increases dramatically (Masters, 1993).

Long training times, overfitting and loss of generalization can be caused by too many hidden neurons. The network may learn insignificant aspects of the training set that are irrelevant to the general population. Too few neurons and the network is not able to learn the pattern at all. The number of required neurons is dependent on the complexity of the function to be learned, and was discovered through experimentation.

Early Stopping

Also to avoid overfitting, a technique called early stopping was used. In early stopping, the data is divided into three subsets. The first is the training set and is used for computing the gradient and updating the weights. The second subset is the validation set. The error on the validation set is monitored during the training process. It normally decreases during the initial phase of training, along with the training set error. However,

when the network begins to overfit the data, the error on the validation set typically begins to rise. In this research project, when the validation error increased for six iterations, the training was stopped and the weights at the minimum of the validation error were returned. The testing set is not used during training. It is used to check the generalization ability of the network.

Specific Procedures of Methodology

The steps required for face detection are as follows:

1. Select a database that contains faces.
2. Segment out potential faces with the K-means algorithm.
3. Create an array of flags indicating face or not-face.
4. Create a matrix of principal components of the potential faces.
5. Train the neural network classifier with the PCA matrix and flag array.

The steps required for eye and mouth detection are identical with some variation in the segmentation step, which will be described in the following sections.

Development Environment

Matlab was used for image processing, K-Means clustering, PCA analysis, and neural network classification. Matlab is a high level language that is widely used in research and engineering. It is especially powerful when analyzing and manipulating vectors and matrices. The scripts used in this research are included in Appendix A.

Database Selection

The images used in this study were from the University of Massachusetts Database "Labeled Faces in the Wild." (Huang, et al. 2007). These images are of

varying sizes and facial expressions, and may contain occlusions, rotations, multiple faces, eyeglasses, facial hair, background structures, and various lighting conditions.

Figure 1: Example Images from Faces in the Wild Database

Kmeans Clustering of Face Candidates

The face candidate database was created from 100 images in the Faces in the Wild database. Each image was read into a 3-dimensional array. A decorrelation stretch was performed to enhance the color separation. The color space was converted to Lab. The matrix values were converted to double, and the matrix was reshaped to combine the rows and columns into a single column. For example, a image with the dimensions of 162x198x3 becomes 32,076x2. The rows correspond to points, and the columns correspond to variables for input into the kmeans algorithm.

Kmeans was performed for k=3, using a weighted distance measure for pixel coordinates and color attributes. Clustering was repeated three times, each with a new random set of initial centroids. The K-means algorithm returns a vector containing the cluster indices of each point, and the centroid locations.

The cluster indices vector is reshaped into original dimensions of rows and columns (162x198 in our example). The centroid locations are sorted and the highest value identifies the segment containing face candidates.

At this point, we have a 2-dimensional gray scale matrix. A disk-shaped morphological structuring element with a radius of 3 was used to eliminate small clusters.

Edges were obtained using the Sobel method, which finds edges using the Sobel approximation to the derivative of the intensity values of the image. It returns a matrix containing 1's at those points were the gradient of the image is maximum.

A two-dimensional convolution was performed on the edge matrix with a smoothing image to reduce the number of connected components.

The edge matrix typically contains 20-30 connected components at this point, with many tiny components inside the larger components. To eliminate these tiny components, we calculate the size of each component and discard those below a value of 100 pixels. We are typically left with 4-8 connected components which are the outlines of our potential faces.

For each of the remaining connected components, a bounding box is calculated and displayed with the original image. The user is prompted to identify whether the image is a face, and the response is saved. The image is cropped with the coordinates of the bounding box, resized to 100x80, and saved to the face candidate database.

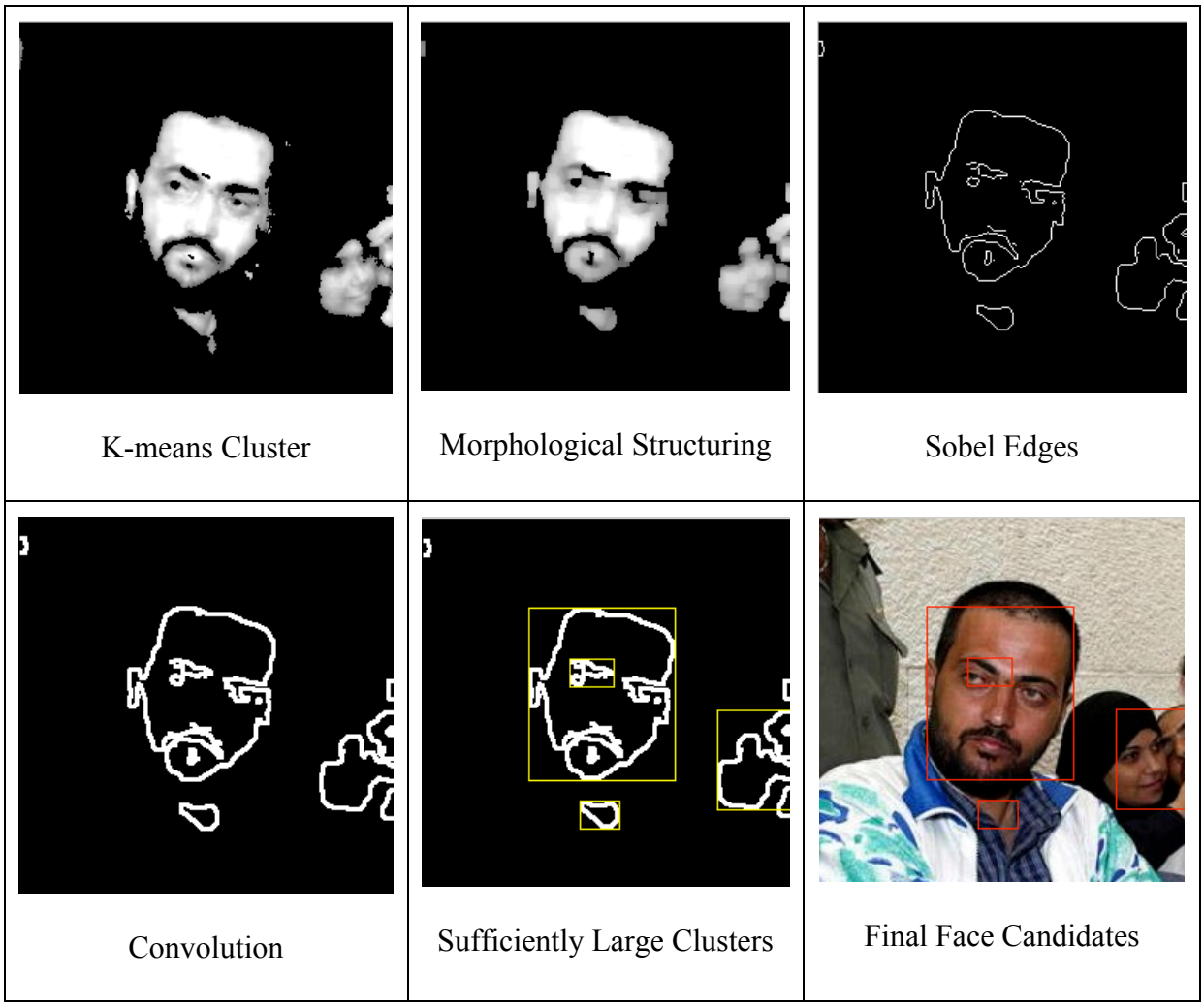


Figure 2: Example of Steps in Face Segmentation

Kmeans Clustering of Eye and Mouth Candidates

The eye and mouth candidate databases were created from 97 images of the faces that had been segmented from the original image. Many pre- and post-processing techniques, as well as several different color spaces were tried, and the best results were obtained with the following method.

An image was read into a 3-dimensional array, and converted to a 2-dimensional grayscale image. The matrix values were converted to double, and the matrix was reshaped to combine the rows and columns into a single column. For example, an image

with the dimensions of 100x80 becomes 8000x1. The rows correspond to points, and the columns correspond to variables for input into the k-means algorithm. A histogram of the intensity values was used to initialize the centroids for the k-means algorithm.

Edges were obtained using the Sobel method, and the edges were convoluted with a smoothing image to reduce the number of connected components. .A lower and upper bound was placed on the size of acceptable components. Bounding boxes were obtained for each remaining disjoint set of edges.

Each bounding box was displayed against the original image, and the user was prompted to identify the component as eye, mouth, or neither. The responses were saved in the isEye or isMouth array. Images were resized to 50x50 for eyes, and to 25x100 for mouths and saved to their respective databases. The images identified as 'neither' were randomly selected to be non-eye or non-mouth examples, and were resized and saved accordingly.

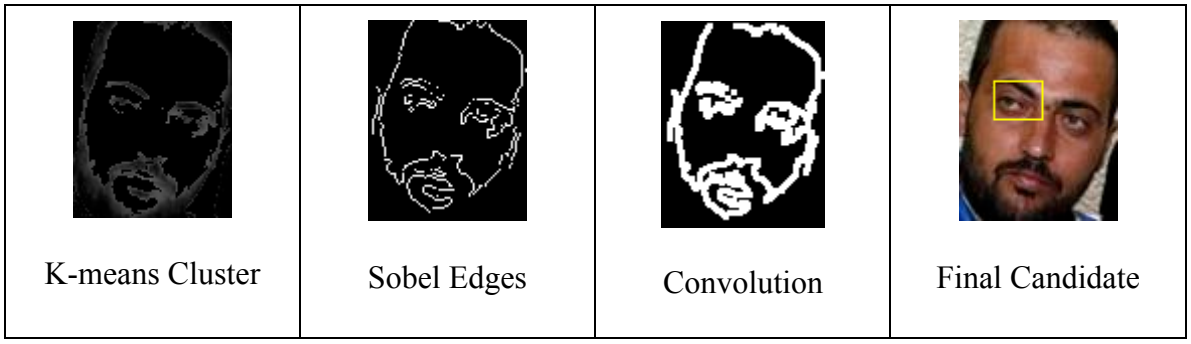


Figure 3: Example of Steps in Facial Feature Segmentation

Feature Extraction – Principal Component Analysis

File images from the newly created database were loaded into Matlab into a matrix of images, with each row encoding one image.

Eigenvectors were obtained from the matrix of images and sorted according to

eigenvalue. The top 40 principal components were projected onto the feature space and used as input to the neural network classifier for the face candidates. The top 20 principal components were used for the eye and mouth candidates, as the images were smaller and contained less irrelevant detail.

Training the Neural Network

The neural network for face classification had 40 inputs, 15 hidden neurons, and 1 output. This network was trained with 659 examples, 97 of which were positive.

The NN for eye classification had 20 inputs, 10 hidden neurons, and 1 output. This network was trained with 322 examples, 97 of which were positive.

The NN for mouth classification had 20 inputs, 10 hidden neurons, and 1 output. This network was trained with 301 examples, 28 of which were positive.

The goal underlying the network design was to discover the simplest network architecture possible so that overfitting could be avoided, and generalization could be maximized. Bishop (1995) refers to this as finding the balance between bias and variance. The procedure was to start with a network that was too small to learn the problem, and continue adding hidden neurons (and if necessary, hidden layers) until the error function was acceptable, and there was insignificant improvement from the previous trial.

The initial network had a single hidden layer with five neurons, and was trained ten times, using different initial weights each time. The best results of the ten training sessions were recorded. A particular training run is sensitive to the initial weights, and it is necessary to repeat the training to discover the best network.

It was determined through numerous trial runs that a NN with 15 hidden neurons

and early stopping resulted in the best generalization for the face NN, while both the eye and mouth NNs performed best with 10 hidden neurons. Each of the NN's used the scaled conjugate gradient training algorithm, with MSE as the performance function.

Chapter 4

Research Results

Format of Results

The results are presented in the form of confusion matrices and receiver operating characteristics (ROC) for the best run of each of the three experiments. The first experiment is for face detection, the second is for eye detection, and the third is for mouth detection.

Results for full face detection

From 100 images containing faces, 659 face candidates were segmented using the kmeans algorithm. 97 of these candidates were identified as faces by the user. Thus, if the NN were to classify the images perfectly, we would at best achieve a 97% success rate in face detection.

The following results were obtained from the NN using 40 principal components as input, 15 hidden neurons, and 1 output neuron.

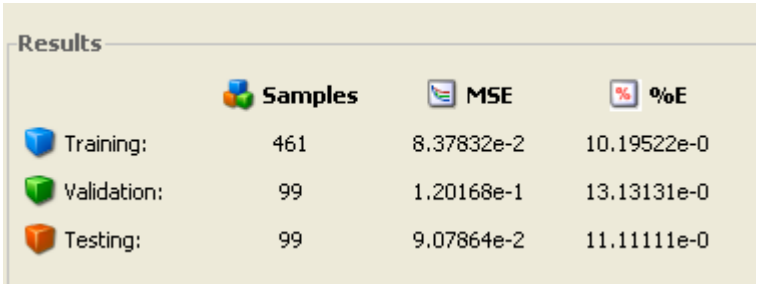


Figure 4: NN mean square error using 40 PCAs for entire face

The following figure shows the ROC for the test cases. The clustering of the data near the upper left corner indicates a good fit to the expected values.

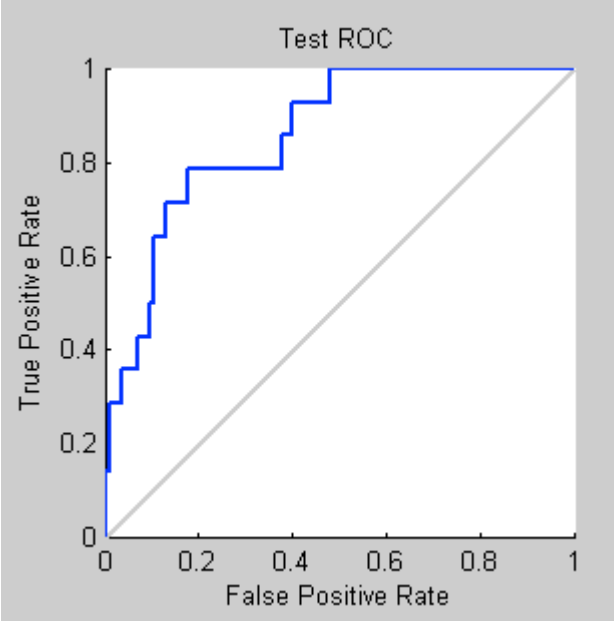


Figure 5: ROC using 40 PCAs for entire face

The confusion matrix shows the number of correct responses in the green squares, and the incorrect responses in the red squares. The overall accuracy is reflected in the bottom right square and shows an 88.9% accuracy rate for the 99 test cases.

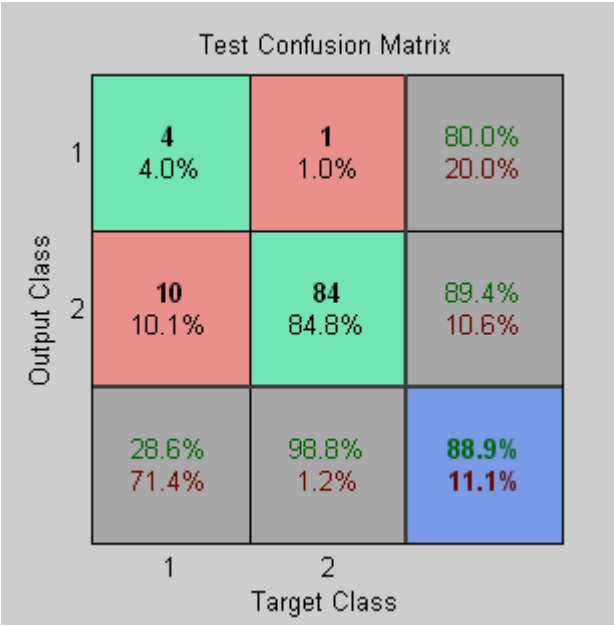


Figure 6: Confusion Matrix using 40 PCAs for entire face

Results for eye detection

From 97 segmented images containing faces, 623 eye and mouth candidates were segmented using the k-means algorithm. 87 of these candidates were identified as eyes by the user. 245 were treated as non-eye examples, for a total of 322 examples to be used with the eye classifier.

Slightly better results were obtained using 20 PCAs rather than the 40 that was used on the entire face.

The following NN results were obtained using 20 principal components as input, 10 hidden neurons, and 1 output neuron. 70% of the examples were used for training, 15% were used for validation, and 15% were used for testing.

Results

	Samples	MSE	%E
Training:	226	6.46716e-2	7.52212e-0
Validation:	48	1.01800e-1	12.50000e-0
Testing:	48	1.07645e-1	14.58333e-0

Figure 7: NN mean square error using 20 PCAs for Eye

The following figure shows the ROC for the test cases. The clustering of the data near the upper left corner indicates a good fit of the data to the expected values.

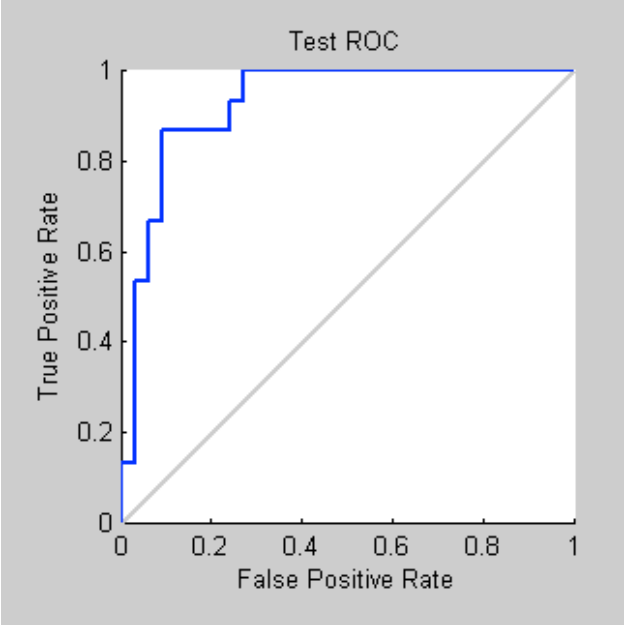


Figure 8: ROC using 20 PCAs for Eye

The confusion matrix shows the number of correct responses in the green squares, and the incorrect responses in the red squares. The overall accuracy is reflected in the bottom right square and shows an 85.4% accuracy rate for the 48 test cases.

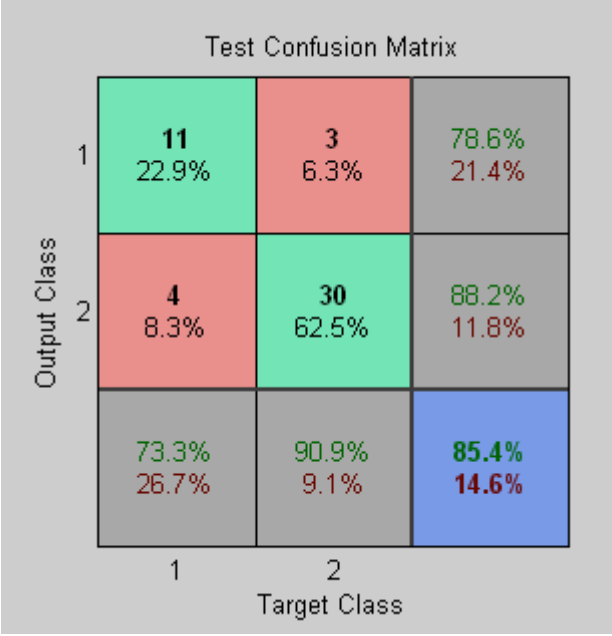


Figure 9: Confusion Matrix using 20 PCAs for eye

Results for mouth detection

From 97 images containing faces, 623 eye and mouth candidates were segmented using the kmeans algorithm. 28 of these candidates were identified as mouths by the user. 274 were treated as non-mouth examples, for a total of 301 examples.

Slightly better results were obtained using 20 PCAs rather than the 40 that was used on the entire face.

The following NN results were obtained using 20 principal components as input, 10 hidden neurons, and 1 output neuron. 70% of the examples were used for training, 15% were used for validation, and 15% were used for testing.

Results

	Samples	MSE	%E
Training:	211	9.30963e-2	11.37440e-0
Validation:	45	6.19817e-2	6.66666e-0
Testing:	45	2.73868e-2	2.22222e-0

Figure 10: NN mean square error using 20 PCAs for mouth

The following figure shows the ROC for the mouth test cases.

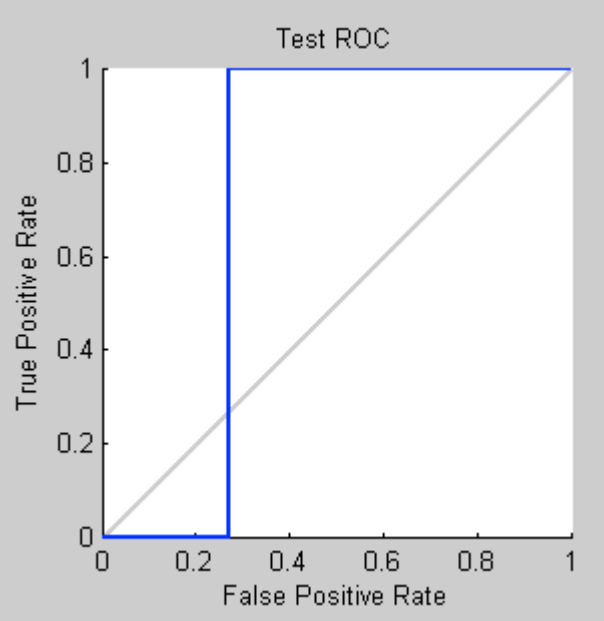


Figure 11: ROC using 20 PCAs for mouth

The confusion matrix shows the number of correct responses in the green squares, and the incorrect responses in the red squares. The overall accuracy is reflected in the bottom right square and shows an 97.8% accuracy rate for the 45 test cases.

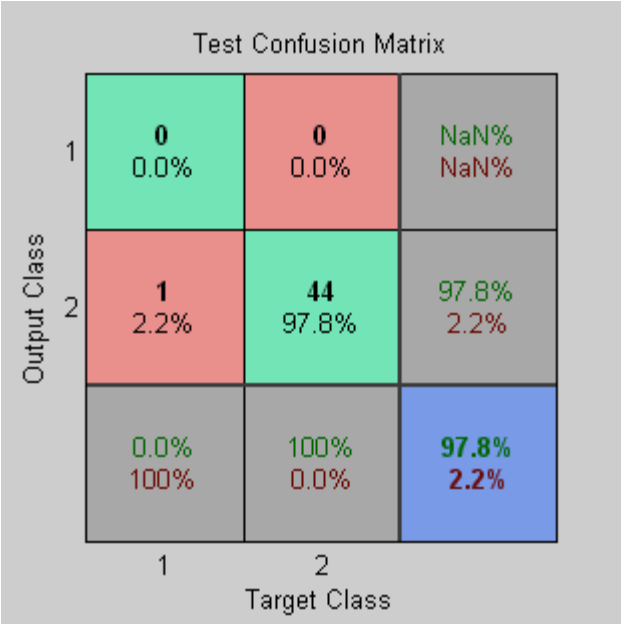


Figure 12: Confusion Matrix using 20 PCAs for Mouth

Summary of Results

Out of a starting set of 100 images, 659 face candidates were segmented using K-means. 97 of the 659 segmented images were actual faces. The top 40 eigenvectors of these 659 images were used as input into a NN with 15 hidden neurons. The NN correctly classified the faces 88.9% of the time.

Out of 97 segmented faces, 623 eye and mouth candidates were segmented using K-means. K-means failed to find any actual eyes or mouths in 19 of the 97 faces.

Out of 322 eye candidates, 87 were actual eyes. The top 20 eigenvectors of these 322 images were used as input into a NN with 10 hidden neurons. The NN correctly classified the eyes 85.4% of the time.

Out of 301 mouth candidates, 28 were actual mouths. The top 20 eigenvectors of these 301 images were used as input into a NN with 10 hidden neurons. The NN correctly classified the mouths 97.8% of the time.

Conclusions and Research Directions

Face candidates were correctly classified in 88.9% of the cases. Eye and Mouth candidates were identified in 85.4% and 97.8% of the cases, respectively. However, while the k-means algorithm was able to segment actual faces quite successfully, it was less effective at segmenting eyes and mouths. Often the eyes blend into the shadows of the face or the segments touch the hair so that the eye becomes part of a larger cluster. On darker complexions, the eyes could not be detected at all. The mouth similarly blended into large clusters, especially when facial hair was present. Often, only small portions of the mouth were segmented due to variations in intensity caused by glimpses

of teeth. Of the 97 faces that were segmented from the original images, 19 had no positive mouth or eye candidates segmented from them. Thus, even if the classifier were perfect, we could only achieve an 80% success rate in face detection using k-means to identify potential eyes and mouths.

It should be noted that face detection research is most often performed on normalized databases, with faces oriented facing toward the camera, with similar lighting and resolution. The 88.9% success rate for face detection using PCA on the entire face is quite good considering the noise and variations in the Faces in the Wild database.

It is possible that the use of template matching may increase the effectiveness of k-means to isolate the small features of the face. The geometry of the eyes and mouth fall within a well-defined range that could be used to heuristically select a portion of a larger cluster as a potential eye or mouth candidate. An interesting method would be to classify each cluster in the post k-means processing phase, so that if an eye is positively identified, that information could be used to select remaining potential candidates from large clusters that would otherwise be discarded.

It would be interesting to continue the research using a video database. Feature tracking could be used over several video frames to capture a sequence that could be classified as a particular gesture. This could be on the level of faces such as turning the head, or on the level of features such as lifting an eyebrow. It would also be interesting to use a more 3-dimensional model, such as optical flow techniques, to extract the facial features.

Appendix A

Program Scripts

```
% ------------------------------------------------------
% kmeansFace.m
% A script file that takes the images identified in the filelist,
% uses k-means to get face candidates, prompts the user to identify
% if the image is a face, and creates a matrix of the top 40 pca
% eigenvectors of those images.
%
% Output: trainin
%         targetin
% ------------------------------------------------------

% set up the input data
inputImagePath = 'C:\nova\AI\images\FacesInWild\training\';
outputImagePath = 'C:\nova\AI\images\FacesInWild\training\facek\';
filelist = 'C:\nova\AI\images\FacesInWild\training\facelist.dat';

% use k-means to segment and save face candidate images
[ allBB ] = getFaceCandidates( filelist, inputImagePath,
outputImagePath );

% get all the segmented images into a matrix
inTrainingImgPath = [inputImagePath 'facek\'];
TrainImgFileList = [inTrainingImgPath 'facelist.dat'];
[Imgs,w,h]=load_images(TrainImgFileList, inTrainingImgPath);

% get numPca eigenvectors per image
numPca=40;
[Vecs,Vals,Psi]=pc_evectors(Imgs, numPca);

% get the Pca Projections
getPcas;                    % output ProjectionInv

% rename for obviousness
isFace = allBB(:,6);        % target vector
trainin = ProjectionInv;    % use as input to neural network training
targetin = isFace;          % use as input to neural network target
%-------------------------------------------------------
```

```
function [ allBB ] = getFaceCandidates( filelist, inputImagePath,
outputImagePath )
% usage: allBBs=getFaceCandidates( filelist, inFilePath, outFilePath );
%    filelist is the name of the file with the list of image file names
%    inFilePath is name where image files are located
%    outFilePath is name where face candidates will be stored
%
%    allBBs has the following format for all face candidates:
%   InImgNum  BB        BB        BB       BB        isFace?
%    1.0000  103.5000  119.5000   63.0000  108.0000    1.0000
%    1.0000  146.5000   71.5000   26.0000   30.0000         0
%    2.0000    1.5000   23.5000   49.0000   81.0000         0
%    2.0000   45.5000   59.5000  131.0000  194.0000    1.0000
%-------------------------------------------------------------

% inputImagePath = 'C:\nova\AI\images\FacesInWild\training\';
% outputImagePath = 'C:\nova\AI\images\FacesInWild\training\facek\';
% filelist = 'C:\nova\AI\images\FacesInWild\training\facelist.dat';
numimgs = linecount(filelist);

fid = fopen(filelist,'r');
 if fid < 0 | numimgs < 1
   error(['Cannot get list of images from file "' filelist, '"']);
 end;

allBB=[];          % Bounding Boxes, image number, and isFace flag
singleBB=[];       % Bounding Boxes for single image
imgNumBB=[];       % singleBB with imgNum column
isFaceBB=[];       % singleBB with imgNum and isFace columns

for i=1:numimgs                          % for each image in input file
    % get input image
    imgname = fgetl(fid);                % get name of image to read
    if ~isstr(imgname)                   % EOF is not a string
      break;                             % Exit from loop on EOF
    end;
    imgnameIn = [inputImagePath, imgname ];
    Img = imread(imgnameIn);           % read img into full color matrix

    fprintf('Processing image %s\n', imgname);
    [singleBB] = getFaceBoundingBoxesKmeans(Img);%use kmeans to get BBs
    imshow(Img);

    %add columns for image num and isFace to BB array
    dimBB=size(singleBB);                % (n 4) for n images
    xnumBBs=dimBB(1);
    imgNumColumn=zeros(xnumBBs,1);       % make new empty column
    imgNumBB=[imgNumColumn singleBB];    % prepend column to BB array
    isFaceColumn=zeros(xnumBBs,1);       % make new empty column

    % get user classification into new column, and save face candidates
    for j=1:xnumBBs                      % for each face candidate in BB
       imgNumBB(j)= i;                   % populate image number
       rect = singleBB(j,:);             % get BB for this candidate
```

```
        faceCandidate=imcrop(Img, rect); % crop candidate from orig img

        displayRect=rectangle('EdgeColor','r');%display rectangle on img
        set(displayRect, 'Visible', 'on');
        set(displayRect, 'Position', rect);

        userInput = input('Face? y/n : ', 's'); % prompt user for input
        set(displayRect,'Visible','off');

        if userInput=='y'                % set isFace flag to true
           isFaceColumn(j)=1;
        end

        %get unique output filename
        imgnameOut = [outputImagePath, imgname ];
        imgnameDim=size(imgnameOut);
        imgnameSize=imgnameDim(2);
        imgnameExt=imgnameOut(imgnameSize-3:imgnameSize);
        numPos=imgnameSize-3;
        numToAppend=num2str(j);
        if j<10
          imgnameOut(numPos)='0';
          imgnameOut(numPos+1)=numToAppend;
        else
          imgnameOut(numPos:numPos+1)=numToAppend;
        end
        imgnameOut(numPos+2:numPos+5)=imgnameExt;

        % resize image so they are all uniform
        faceCandidate=imresize( faceCandidate, [ 100 80 ]);

        %save cropped and resized image to new file
        imwrite(faceCandidate, imgnameOut, 'jpg');
    end
    isFaceBB=[imgNumBB isFaceColumn ];
    allBB=[allBB; isFaceBB];           % add BBs for this image to allBB
 end

fclose(fid);

dimBB=size(allBB);
numCandidates=dimBB(1);
fprintf(1,'Completed processing %d images with %d
candidates.\n',numimgs, numCandidates);
%-------------------------------------------------------
```

```
function [ BBarray ] = getFaceBoundingBoxesKmeans( inputImage )
% usage: BBarray = getFaceBoundingBoxes( Image )
% given an input image, this function returns an array containing
% the bounding boxes of all the face candidates in the image.
% The size of BBarray varies with number of face candidates.
%
% BBarray example for 4 face candidates:
%   76.5000   62.5000  124.0000  120.0000
%  136.5000  214.5000   65.0000   78.0000
%  203.5000    1.5000   42.0000   15.0000
%  326.5000    9.5000   44.0000   39.0000
%-------------------------------------------------------

decorrCIR=decorrstretch(inputImage,'Tol',0.01); %increase color separ.
he=decorrCIR;
cform = makecform('srgb2lab');           %convert to lab color space
lab_he = applycform(he,cform);

ab = double(lab_he(:,:,2:3));            % prepare input for k-means
nrows = size(ab,1);
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,2);

nColors = 3;                             % k=3 clusters
[cluster_idx cluster_center] =
kmeans(ab,nColors,'distance','sqEuclidean','Replicates',3);

pixel_labels = reshape(cluster_idx,nrows,ncols);
segmented_images = cell(1,3);
rgb_label = repmat(pixel_labels,[1 1 3]);
for k = 1:nColors
color = inputImage;
color(rgb_label ~= k) = 0;
segmented_images{k} = color;
end

%determine which cluster has the faces.. red cluster
mean_cluster_value = mean(cluster_center,2);
[tmp, idx] = sort(mean_cluster_value);
L = lab_he(:,:,1);
red_cluster_num = idx(3);

%get outline of clusters
ima = segmented_images{red_cluster_num};
im1 = ima(:,:,1);

%eliminate small clusters
sedisk = strel('disk',3);
im1 = imopen(im1, sedisk);

%get edges
BW = edge(im1,'sobel');     %finding edges
[imx,imy]=size(BW);
```

```
msk=[0 0 0 0 0;
     0 1 1 1 0;
     0 1 1 1 0;
     0 1 1 1 0;
     0 0 0 0 0;];
B=conv2(double(BW),double(msk)); %Smoothing  image to reduce the number
                                 %of connected components
L = bwlabel(B,8);                % Calculating connected components
mx=max(max(L));                  % number of connected components

BBarray = zeros(mx,4);           %init array for bounding boxes
numCandidates = 0;

for j=1:mx                       %for each face candidate

    %the following steps are done to eliminate remaining tiny clusters
    %that occur inside bigger clusters.
    [r,c] = find(L==j);
    rc = [r c];
    [sx sy]=size(rc);
    n1=zeros(imx,imy);
    for i=1:sx
        x1=rc(i,1);
        y1=rc(i,2);
        n1(x1,y1)=255;
    end                      % Storing the extracted image in an array

    n=find(n1);
    [n,m]=size(n);

     if n>100                %  make sure cluster is large enough
        numCandidates = numCandidates + 1;
        data = regionprops( L, 'basic');
        BBarray(j,:)= data(j).BoundingBox; %Bounding box of the jth img
    end

end

SumRows = sum(BBarray,2);
NonEmptyRows = find( SumRows > 0 );
BBarray = BBarray( NonEmptyRows, : );
```

```
 %-------------------------------------------------------

% kmeansEyes.m
% A script file that takes the images identified in the filelist,
% uses k-means to get eye and mouth candidates, prompts the user
% to identify if the image is an eye or mouth, and creates a
% matrix of the top 20 pca eigenvectors of those images.
%
% Output: trainin
%         targetin
% -------------------------------------------------------

% set up the input data
inputImagePath = 'C:\nova\AI\images\FacesInWild\training\facek';
filelist = 'C:\nova\AI\images\FacesInWild\training\facek\facelist.dat';

% use k-means to segment and save eye and mouth candidate images
[ EyeBB MouthBB ] = getEyeCandidates( filelist, inputImagePath );

% get all the segmented images into a matrix
inTrainingImgPath = [inputImagePath 'eye\'];
TrainImgFileList = [inTrainingImgPath 'eyelist.dat'];
[Imgs,w,h]=load_images(TrainImgFileList, inTrainingImgPath);

% get numPca eigenvectors per image
numPca=40;
[Vecs,Vals,Psi]=pc_evectors(Imgs, numPca);

% get the Pca Projections
getPcas;                    % output ProjectionInv

% rename for obviousness
isEye = EyeBB(:,6);         % target vector
trainin = ProjectionInv;    % use as input to neural network training
targetin = isEye;           % use as input to neural network target
%-------------------------------------------------------------
```

```
 function [ EyeBB MouthBB ] = getEyeCandidates( filelist,
inputImagePath )
% usage: [ EyeBB MouthBB ] = getEyeCandidates( filelist, inputImagePath
)
%    filelist is the name of the file with the list of image file names
%    inFilePath is name where image files are located.
%    Assumes eye and mouth subdirectories exist under inputImagePath
%
%    EyeBB/MouthBB has the following format for all eye candidates:
%   InImgNum    BB        BB        BB        BB     isEye/isMouth
%    1.0000  103.5000  119.5000   63.0000  108.0000    1.0000
%    1.0000  146.5000   71.5000   26.0000   30.0000         0
%    2.0000    1.5000   23.5000   49.0000   81.0000         0
%    2.0000   45.5000   59.5000  131.0000  194.0000    1.0000
%-------------------------------------------------------------

% inputImagePath = 'C:\nova\AI\images\FacesInWild\training\facek\';
% filelist =
'C:\nova\AI\images\FacesInWild\training\facek\facelist.dat';

outputEye = [inputImagePath 'eye\'];
outputMouth = [inputImagePath 'mouth\'];

numimgs = linecount(filelist);

fid = fopen(filelist,'r');
 if fid < 0 | numimgs < 1
   error(['Cannot get list of images from file "' filelist, '"']);
 end;

singleBB=[];    % Bounding Boxes for single image
EyeBB=[];       % Bounding Boxes for eye/non-eye imgs saved in eye dir.
MouthBB=[];     % Bounding Boxes for mouth/non-mouth imgs in mouth dir.

for i=1:numimgs                          % for each image in input file
    imgname = fgetl(fid);                % get name of image to read
    if ~isstr(imgname)                   % EOF is not a string
      break;                             % Exit from loop on EOF
    end;
    imgnameIn = [inputImagePath, imgname ];
    Img = imread(imgnameIn);             % read img into matrix

    fprintf('Processing image %s\n', imgname);
    imshow(Img);
    [singleBB] = getEyeMouthBBKmeans(Img);

    dimBB=size(singleBB);                % get num of BBs for this image
    xnumBBs=dimBB(1);
    imshow(Img);

    for j=1:xnumBBs
       rect = singleBB(j,:);             % get jth bounding box

       displayRect=rectangle('EdgeColor','y');
```

```
set(displayRect, 'Visible', 'on');
set(displayRect, 'Position', rect);  % display jth bounding box


userInput = input('Eye or Mouth? e/m/n : ', 's');
Candidate = imcrop(Img, rect); % crop candidate from orig img
dim=size(Candidate);
if (dim(1)==0) | (dim(2)==0) | (dim(3)==0)
    fprintf(1,'Terminating on %d %d %s \n', i, j, imgname);
    break;
end

if userInput=='e'              % set isEye flag to true
    Candidate=imresize(Candidate, [ 50 50 ]);
    imgname2 = '__e';
    outputImagePath = outputEye;
    bb = [i rect 1];
    EyeBB=[EyeBB; bb];
  elseif userInput=='m'              % set isMouth flag to true
    isMouthColumn(j)=1;
    Candidate=imresize(Candidate, [ 20 100 ]);
    imgname2 = '__m';
    outputImagePath = outputMouth;
    bb = [i rect 1];
    MouthBB=[MouthBB; bb];
  else
    if mod(j,2)==0
      isEyeColumn(j)=0;
      Candidate=imresize(Candidate, [ 50 50 ]);
      imgname2 = '_ne';
      outputImagePath = outputEye;
      bb = [i rect 0];
      EyeBB=[EyeBB; bb];
    else
      isMouthColumn(j)=0;
      Candidate=imresize(Candidate, [ 25 100 ]);
      imgname2 = '_nm';
      outputImagePath = outputMouth;
      bb = [i rect 0];
      MouthBB=[MouthBB; bb];
    end
  end

  set(displayRect, 'Visible', 'off');

  %get unique output filename
  imgnameOut = [outputImagePath, imgname ];
  imgnameDim=size(imgnameOut);
  imgnameSize=imgnameDim(2);
  imgnameExt=imgnameOut(imgnameSize-3:imgnameSize);
  numPos=imgnameSize-3;
  numToAppend=num2str(j);
  if j<10
    imgnameOut(numPos)='0';
    imgnameOut(numPos+1)=numToAppend;
  else
```

```
            imgnameOut(numPos:numPos+1)=numToAppend;
          end
          imgnameOut(numPos+2:numPos+4)=imgname2;
          imgnameOut(numPos+5:numPos+8)=imgnameExt;

          %save cropped and resized image to new file
          imwrite(Candidate, imgnameOut, 'jpg');
          fprintf(1,'Writing %d %d %s \n', i, j, imgnameOut);

    end
 end

fclose(fid);


 %-------------------------------------------------------

function [ BBarray ] = getEyeMouthBBKmeans( inputImage )
% usage: BBarray = getFaceBoundingBoxes( Image )
% given an input image, this function returns an array containing
% the bounding boxes of all the face candidates in the image.
% The size of BBarray varies with number of face candidates.
%
% BBarray example for 4 face candidates:
%   76.5000   62.5000  124.0000  120.0000
%  136.5000  214.5000   65.0000   78.0000
%  203.5000    1.5000   42.0000   15.0000
%  326.5000    9.5000   44.0000   39.0000
%-------------------------------------------------------

ImgGray=rgb2gray(inputImage);       % convert image to gray scale

[mu,mask]=kmeansBW( ImgGray, 3);    % perform k-means

pixel_labels = mask;
nColors=3;

segmented_images = cell(1,3);
rgb_label = repmat(pixel_labels,[1 1 3]);
for k = 1:nColors
color = inputImage;
color(rgb_label ~= k) = 0;
segmented_images{k} = color;
end

im1=segmented_images{1};
im1=im1(:,:,1);

BW = edge(im1,'sobel');         % get edges
[imx,imy]=size(BW);

% convolution
msk=[0 0 0 0 0;
     0 1 1 1 0;
```

```
     0 1 1 1 0;
     0 1 1 1 0;
     0 0 0 0 0;];
B=conv2(double(BW),double(msk)); %Smoothing  image to reduce the number
of connected components
L = bwlabel(B,8);                % Calculating connected components
mx=max(max(L));                  % number of connected components

BBarray = zeros(mx,4);       %init return array for bounding boxes of
face candidates
numCandidates = 0;

for j=1:mx                   %for each face candidate, get bounding box

    %the following steps are done to eliminate remaining tiny clusters
that
    %occur inside bigger clusters.
    [r,c] = find(L==j);
    rc = [r c];
    [sx sy]=size(rc);
    n1=zeros(imx,imy);
    for i=1:sx
        x1=rc(i,1);
        y1=rc(i,2);
        n1(x1,y1)=255;
    end                      % Storing the extracted image in an array

    n=find(n1);
    [n,m]=size(n);           % if n is less than 100x1, it should be
eliminated as too small
                             % goes from 13 to 7 in SteveBill, gets rid
of dots inside face

    % make sure candidate is not too small or too large
    if (n>20) & (n<500)
        numCandidates = numCandidates + 1;
        data = regionprops( L, 'basic');
        BBarray(j,:)= data(j).BoundingBox; %Bounding box of the jth
image
    end

end

% eliminate empty rows for images too small to consider
SumRows = sum(BBarray,2);
NonEmptyRows = find( SumRows > 0 );
BBarray = BBarray( NonEmptyRows, : )

userInput = input('confirmation getEyeMouthBBKmeans ', 's');

fprintf(1,'Returning dim %d BB for %d face candidates.\n',mx,
numCandidates);

%-------------------------------------------------------------
```

```
% kmeansMouth.m
% A script file that takes the images identified in the filelist,
% uses k-means to get eye and mouth candidates, prompts the user
% to identify if the image is an eye or mouth, and creates a
% matrix of the top 20 pca eigenvectors of those images.
%
% Input:  inputImagePath
%         filelist
%
% Output: trainin
%         targetin
% ------------------------------------------------------

% set up the input data
inputImagePath = 'C:\nova\AI\images\FacesInWild\training\facek';
filelist='C:\nova\AI\images\FacesInWild\training\facek\facelist.dat';

% This step was already done when we processed eyes.  It only has to
% be executed once since it gets all the eye and mouth candidates.
% use k-means to segment and save eye and mouth candidate images
% [ EyeBB MouthBB ] = getEyeCandidates( filelist, inputImagePath );

% get all the segmented images into a matrix
inTrainingImgPath = [inputImagePath 'mouth\'];
TrainImgFileList = [inTrainingImgPath 'mouthlist.dat'];
[Imgs,w,h]=load_images(TrainImgFileList, inTrainingImgPath);

% get numPca eigenvectors per image
numPca=20;
[Vecs,Vals,Psi]=pc_evectors(Imgs, numPca);

% get the Pca Projections
getPcas;                    % output ProjectionInv

% rename for obviousness
isMouth = MouthBB(:,6);     % target vector
trainin = ProjectionInv;    % use as input to neural network training
targetin = isMouth;         % use as input to neural network target
```

References

Bishop, Christopher M. (1995) Neural Networks for Pattern Recognition. New York: Oxford University Press.

Duda, R.O., Hart, P.E., Stork D.G. (2001) Pattern classification (2nd ed). New York: John Wiley & Sons, Inc.

Grangeiro, F., Jesus, R., Correia, N. (2009). "Face recognition and gender classification in personal memories," icassp, pp.1945-1948, 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, 2009

Han, J., Kamber, M., Pei, J. (2005). Data Mining: Concepts and Techniques. Morgan Kaufmann; 2nd edition (November 3, 1005).

Huang, G., Ramesh, M., Berg, T., and Learned-Miller, E. (2007). Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. University of Massachusetts, Amherst, Technical Report 07-49, October, 2007. http://vis-www.cs.umass.edu/lfw/

Jain, A.K., Murty, M.N., and Flynn, P.J (1999). Data Clustering: A Review. In ACM Computing Surveys, 31(3):265-323, 1999.

Kobayashi, H. and Zhao, Q. (2007). Face Detection Based on LDA and NN. In Proceedings of the 2007 Japan-China Joint Workshop on Frontier of Computer Science and Technology (November 01 - 03, 2007).

Lekshmi, P., Kumar, V., and Vidyadharan, D. (2008). Face Detection and Localization of Facial Features in Still and Video Images. In Proceedings of the 2008 First international Conference on Emerging Trends in Engineering and Technology - Volume 00 (July 16 - 18, 2008).

Lyons, M., Akamatsu, S., Kamachi, and M., Gyoba, J (1998). Coding Facial Expressions with Gabor Wavelets. Proceedings, Third IEEE International Conference on Automatic Face and Gesture Recognition, April 14-16 1998, Nara Japan, IEEE Computer Society, 200-205.

Milligan, G., and Cooper, M. (1985). An Examination of Procedures for Determining the Number of Clusters in a Dataset, Psychometrika, 50, 159-179.

Ruan, J., Yin, J. (2009) "Face Detection Based on Facial Features and Linear Support Vector Machines," iccsn, pp.371-375, 2009 International Conference on Communication Software and Networks, 2009.

Vezhnevets, V., Sazonov, V., and Andreeva, A. (2003). A survey on pixel-based skin color detection techniques., Graphicon2003, 13th International Conference on the Computer Graphics and Vision, 2003,Moscow, Russia, September.

Viola, P. and Jones, M. J. (2004). Robust Real-Time Face Detection. Int. J. Comput. Vision 57, 2 (May. 2004), 137-154.

Ying-hui, W., Xiao-juan, N., Chun-xia, Y., Qiong-fang, W. (2009). "A Novel Method for Face Detection across Illumination Changes," gcis, vol. 2, pp.374-378, 2009 WRI Global Congress on Intelligent Systems, 2009