Classification of Human Facial Expressions with Principal Component Analysis and Neural Networks

by

Nancy Smith

A research paper submitted in partial fulfillment of the requirements for CISD760

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 CISD760

Classification of Human Facial Expressions
With Principal Component Analysis and Neural Networks

by
Nancy Smith

July 2009

Automatic detection of facial expressions can provide valuable input to computer systems that interact with humans. Classifying expressions is a high-dimensional pattern recognition problem, which was addressed by obtaining lower-dimensional feature vectors using principal component analysis of complete facial images. The PCA projections were then used as input into a feed forward neural network trained to classify the input into seven expressions. Networks were trained with the top 10 PCAs with a 40% success rate, and with 40 and 100 PCAs, both of which achieved an 80% success rate. This research demonstrates that PCA eigenvectors are able to represent the patterns unique to basic emotions across different faces.

Table of Contents

List of Figures

Figures

Chapter 1

Introduction

Statement of the problem

As computer systems become increasingly more sophisticated and involved in our lives, it becomes increasingly desirable to interact with them similar to the way we interact with humans. One of the most crucial features of human interaction is the ability to detect and adapt to each others emotional state. This allows a teacher to modify the level of detail presented to students who appear bored or confused. It allows security personnel to focus in on subjects who appear secretive or suspicious. It enables doctors to diagnose behavioral or neurological pathologies. Automatic detection of various facial affects could assist professionals such as these, minimizing the reliance on an individual's subjective perception of emotion. It would also be helpful for conducting research in behavioral science, anthropology, neurology, and psychiatry, where consistent and precise measurements of affect are ongoing problems. Automatic detection of boredom could be invaluable in roles where attention is crucial such as air traffic controllers or automobile drivers. Intelligent systems that can detect emotions can adapt and respond to these affective states and are likely to be perceived as more endearing and trustworthy. Researchers have even designed learning algorithms with weights representing emotions to aid in real-time reasoning (Khasman, 2008).

Classifying a facial expression is a problem in pattern recognition. Pattern recognition can be separated into the following components: sensing, segmentation, feature extraction, classification, and post-processing (Duda et al. 2001). To classify

facial expressions, sensing is done with a camera to obtain photographic images. Segmentation narrows the range of interest to a partition of the image, or a single individual face. Feature extraction constructs a representation of the characteristics of that partition that can be analyzed for classification. The features must be invariant to transformations such as size and rotation. Classification then uses that representation, or feature vector, to categorize the object.

There is a trade-off between feature extraction and classification. A perfect feature extractor would render the classification task trivial, whereas an omnipotent classifier would need no feature extractor (Duda et al. 2001). However, the dimensionality of the data in an image makes the omnipotent classifier far from attainable. High dimensionality suffers from several problems. The first is that identical features may appear unrelated in the raw data, as in images of an object that are being rotated or scaled. Another is that as the number of input features increases relative to the size of the training sample, the classifier loses its ability to generalize. In the most extreme case, it becomes a look-up table that matches only exact patterns it has seen before. Another problem has been called the 'curse of dimensionality' in that the number of example data points grows exponentially with the number of input features. Thus, the feature extraction step has a profound effect on the performance of the pattern recognition system of high-dimensional data (Bishop, 1995).

The feature extraction method used for the research described in this paper is principal component analysis (PCA). PCA is able to transform high dimensional data into a lower dimension, while preserving as much of the distinguishing information as possible.

PCA is a classical approach that copes with excessive dimensionality by combining features through linear transformations. Linear combinations are desirable because they are easy to compute and analytically tractable. PCA is an unsupervised approach to finding the most distinguishing features by projecting higher-dimensional data onto a lower-dimensional subspace in a way that optimizes the sum-squared error (Duda and Hart, 2001).

The feature vectors discovered through PCA are the input to a feed-forward back-propagation neural network (NN), which is trained to classify the vectors as either one of six emotions or a neutral expression. A neural network is able to provide general parameterized non-linear mappings between a set of input variables and a set of output variables. Neural networks have been shown to be quite effective in classification problems.

Significance of the study

Among the feature extraction mechanisms that have been used to classify facial emotions are Gabor wavelets (Donato, Bartlett, Hager, Ekman, & Sejnowski ,1999; Hu, De Silva, & Sengupta, 2002), x-y coordinates (Kurihara, Sugiyama, Matsumoto, Nishiuchi, & Masuda, 2009), Discrete Cosine Transform (Kharat & Dudul, 2008), and Haar features (Whitehill & Omlin, 2006). Each of these methods has disadvantages in terms of manual pre-processing and complexity. PCA has an advantage over other schemes in its speed and simplicity.

Turk and Pentland (1991) demonstrated the power of PCA for classification problems in their seminal paper on face recognition. PCA research has been further

developed and refined over the years in the recognition of human faces (Eleyan & Demirel (2005); Rizon et al, (2006); Sahoolizadeh, Heidari, & Dehghani (2008)) particularly with the type of classifier that is used with the PCA feature vectors.

PCA has been used with other image processing techniques to extract salient features that are particularly indicative of emotional state. A change in features in the areas of the eyebrow, eyelids, and mouth especially can convey information about what an individual is thinking or feeling. The Facial Action Coding System (FACS) is a detailed guide based on a human observation system that detects subtle changes in facial features (Ekman & Friesen, 1975). Viewing videotaped facial movement in slow motion video, trained observers can manually code in FACS all the possible facial expressions. The muscles that cause the smallest visible change in facial expression are called facial action units in FACS. Emotional expressions are not explicitly specified in FACS, even though Ekman and Friesen proposed that combinations of facial action units may represent emotional expressions. A total of 44 facial action units were defined in detail, 30 of which correspond to specific face muscles. Of these, 12 are related to the upper face, and 18 are for the lower face. Action units can occur alone or in combination with other action units. Although the number of facial action units is small, more than 7000 combinations have been observed.

FACS is purely descriptive and uses no emotion or other labels on the facial movements. The intent of FACS is only on identifying the facial action units responsible for producing the facial behavior. Emotional expressions are coded in separate systems that use FACS as the supporting research. The most widely used of these is EMFACS, also developed by Ekman and Friesen.

In order to take advantage of the research that has built on FACS, researchers have investigated numerous image processing techniques to extract individual facial action units. Principal component analysis has been used in conjunction with dense-flow tracking in the identification of facial action units (Lien, Kanade, Cohn, & Li, 2000). PCA has been used with LED feature tracking (Kapoor, Qi, & Picard, 2003) to extract facial action units. PCA has been used with locally linear embedding techniques to detect facial action units. These research efforts share a focus on "feature" PCAs, in which separate PCAs are obtained to represent different parts of the face, particularly the area around the mouth or the eyes.

The research conducted in this project used the full facial PCA projections rather than the feature projections that focus on the eyes and mouth. There is significant research to automatically segment and normalize a face. Therefore, the ability to identify an emotion based on the entire face could prove quite useful to researchers who only need to know the emotion and have no need to know the accompanying facial action units.

Barriers and issues

Emotions are most readily identified by the position of the mouth, eyes, and eyebrows. The set of PCA vectors for the entire face captures a lot of irrelevant data related to the face structure and hair which may dominate the patterns represented by the PCAs.

Elements, hypotheses, theories, or research questions

PCA is able to capture the differences in patterns. It is hypothesized that within

the PCA data set is not only information that can be used to distinguish individuals, but also information that can be used to distinguish expressions. By training a neural network to map the PCA eigenvectors to an emotion rather than to an individual, this pattern can be recognized by a neural network.

Delimitations of the study

The degree of difficulty of a classification problem depends on the variability of the features for objects in the same category relative to differences between features for objects in different categories. Feature variations may be due to complexity or to noise. Noise may be considered any property of the pattern which is not due to the underlying model but to randomness in the world or in the sensors. In order to minimize noise or absence of features due to problems such as occlusion, background, and points of view, the input images selected were normalized to be of similar format.

The images in this study were black and white, in tiff format, centered, of similar size and background, and had a resolution of 256x256. It is believed that having a similar format for all images decreases non-emotion related differences.

Definition of terms

Eigenvector is a vector which, when acted on by a linear transformation produces a scalar multiple of the original vector. The scalar is called an eigenvalue.

Facial Action Coding System (FACS) is a detailed technical guide that explains how to categorize facial behaviors based on the muscles that produce them.

Principal Component Analysis (PCA) is a mathematical procedure that transforms high dimensional data such that the greatest variance by any projection of the data is on

the first coordinate (the principal component), the second greatest variance on the second coordinate, and so on.

Receiver Operating Characteristic (ROC) curve is a plot of the true positive versus the false positive classification rate.

Summary

Recognition of facial expressions has the potential of enhancing a wide range of computer applications. Because of the high dimensionality of image data, feature extraction is extremely important. Principal component analysis can be used to obtain feature vectors representing differences in patterns, and neural networks can be trained to recognize patterns that are stored in the PCA vectors. The hypothesis of this research paper is that PCA can obtain feature vectors from images to adequately distinguish the basic emotions using a feed forward neural network.

Chapter 2

Review of the Literature

The theory and research literature specific to the topic

Techniques using PCA to extract eigenfaces have been used for recognition of faces and for classification of facial action units. Those used for face recognition obtain PCA eigenvectors of the entire face, while those used for action units focus on a small region of the face, particularly the mouth or eyes since these are important factors in identifying emotion. The research described in this paper extends these techniques by using PCA eigenvectors of the entire face to classify one of six basic emotions that were detailed by Ekman & Friesen (1978) in their seminal work on facial action units. These basic expressions are anger, joy, sadness, fear, disgust, and surprise. Following is a brief description of the relevant literature that inspired this project. The first three examples use PCA for face recognition, and the last three examples use PCA for facial action unit detection.

Eleyan & Demirel (2005) used PCA with a feed-forward NN for face recognition, comparing the results to PCA with a Euclidean distance classifier. The Olivetti Research Laboratory (ORL) database of 400 faces was used in the experiment. The researchers used 50 PCA eigenvectors as input, a hidden layer of 15 units, and an output layer of 40 individuals. They obtained an improvement over the Euclidean distance classifier of 1% to 7%, depending on the number of training images from each subject. The best performance obtained was 95%.

Rizon et al. (2006) also used PCA with a feed-forward NN. In this project, only 8 eigenvectors were used as input, with 5 hidden units, and 3 output units representing 8 classes. These researchers also showed that face recognition rates improved with each additional training example added, although specific success rates were not noted.

Sahoolizadeh et al. (2008) used PCA, Linear Discriminant Analysis (LDA), and neural networks for face recognition. These researchers used PCA to improve the capability of LDA. Since LDA is more effective than PCA with a large number of data samples, but cannot be used with high dimensional data, the researchers used PCA to reduce the dimensionality before applying LDA. The images were manually cut to remove background noise so that only the eyes, nose, and mouth were visible. This resulted in input images of 40x40 used for the PCA. They varied the number of PCA eigenvectors to 20, 40, 60, 80 and 100 to use in LDA. They varied the number of LDA features from 3 to 9 for each of the sets of eigenvectors. The NN configuration was 40 input units, 20 hidden units, and 10 output units. The best results were with 100 eigenvectors and 9 LDAs, with a recognition rate of 99.5%. Computation cost was significantly higher as more eigenvectors and LDAs were used.

Lien, Kanade, Cohn, & Li 2000 (1999) focused on the dynamic aspect of facial movements by classifying sequences of input vectors using optical flow to track motion. They compared facial feature point tracking, dense flow tracking, and high gradient component analysis to extract facial features. PCA was used to process the dense flows rather than the raw images. Facial feature tracking manually marked the normal position of the contours of the eyebrows and then tracked automatically through the remaining frames. A horizontal and vertical displacement vector was calculated between the

normalized frame and the current frame. This method tracks large displacement well, but to detect finer movement, dense flow is used to track each pixel in a 110x240 area around the eyes and eyebrows. PCA was used to create 10 eigenflows from the vertical direction and 10 eigenflows from the horizontal direction. Finally, high gradient components were extracted using edge detectors to detect furrows. The gradient intensity was compared with the initial frame to distinguish between permanent and temporary wrinkles or furrows. Sequences of the extracted features were classified with a Hidden Markov Model (HMM) to identify 14 action units from the upper face as defined by the Facial Action Coding System. The best results obtained were with dense flow tracking with PCA at 93%.

Kapoor et al. (2003) developed a fully automated system that didn't require manual pre-processing of the images for recognizing facial action units. They automated the normalization of the images by using infrared LEDs to detect pupils and align the images. Areas around the eyes and eyebrows were cropped, and the facial features in each frame are normalized with respect to pose and zoom. The top 40 PCA eigenvectors were obtained and used as input to a support vector machine (SVM) to identify 5 facial action units. A separate SVM was trained for each action unit. They achieved a 69% success rate for individual action unit recognition using a dataset with movements and occlusions. The dataset was created by selecting frames from a video of children playing games rather than using a database containing images that are normalized and using the extremes of emotional expression.

Reilley et al. (2007) used Kernel PCA, which differs from normal PCA in that the data is projected into a higher dimensional feature space prior to performing eigenvector

decomposition using the kernel trick. They also used the local linear embedding (LLE) algorithm for comparison. The outputs from both the KPCA and LLE models are fed into a SVM for classifying 4 action unit combinations for the mouth region. The LLE model consistently outperformed the KPCA model. The KPCA model was unable to differentiate two of the combinations from each other.

The research in this paper uses PCA to extract eigenvectors of the entire face, similar to the techniques in Eleyan & Demirel (2005), and Rizon et al. (2006) in their research on face recognition. The eigenvector projections of the entire face are used as direct input to the neural network. Two of the research papers discussed in this section used PCA to assist another methodology. Sahoolizadeh et al (2008) used PCA to improve LDA performance. Lien et al (1999) used PCA to analyze dense flows. Two other research papers used PCA eigenvectors from a small region of the face to detect facial action units. Kapoor et al. (2003) used PCA on a subregion segmented around the eyes using infrared LEDs. Reilley et al. (2007) used a variation called KPCA on a subregion around the mouth, using the kernel trick similar to support vector machines.

Summary of what is known and not known

PCA has been used successfully for facial recognition. When combined with other techniques, it has also been successful in recognizing small subsets of facial action units. PCA eigenvectors of the entire face may contain a lot of irrelevant information with regard to emotion. Items such as face structure, hair, and background will create noise in the patterns captured in the PCA eigenvectors. Whether neural networks can be trained to recognize the basic emotions in the presence of this noise may not be fully established.

The contribution this study will make to the field

The contribution this research provides is to demonstrate the effectiveness of neural networks to recognize subpatterns that exist in the dominant patterns within a set of PCA eigenvectors. Since full facial PCA eigenvectors have been used to recognize individuals, it is known that faces with different expressions exhibit a great deal of similarity when compared to other faces. From this, we know that the principal components may represent features unique to an individual across a wide range of expressions. The research described by this paper will show that the same principal components that have been used to identify individuals can also be used to identify an emotion.

Chapter 3

Methodology

Research methods employed

The images used in this study were from the Japanese Female Facial Expression database (Lyons, Akamatsu, Kamachi, & Gyoba 1998). This database is comprised of ten individuals exhibiting six basic emotions and one neutral expression. Several images of each expression exist for a total of 213 images. The images are black and white, centered, and are comprised of 256x256 pixels.

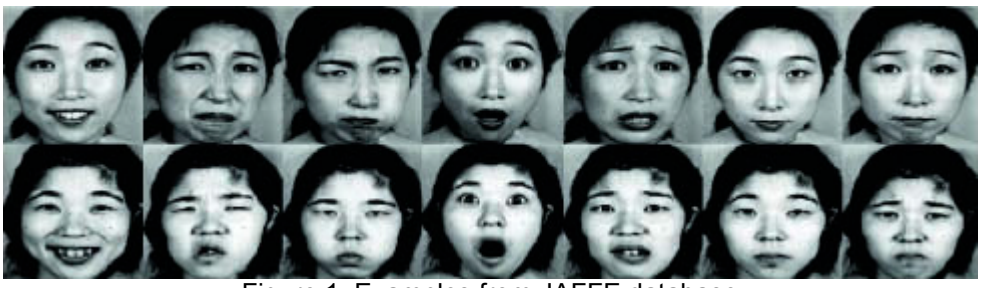


Figure 1. Examples from JAFFE database

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 vectors of 65,536 dimensions 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.

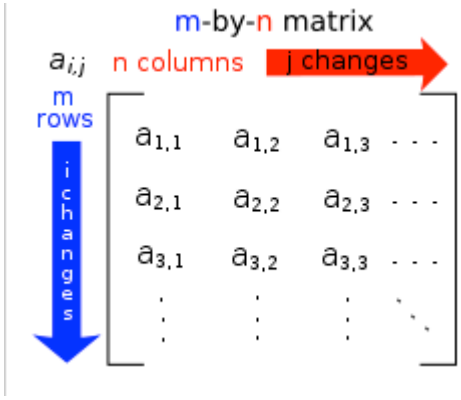
Consider an image of 256x256 pixels. It can be viewed as a point in 65,536-dimensional space. An ensemble of images maps to a collection of points in this huge space. Faces have many similarities and the distribution will not be random. The eigenvectors describe this subspace of facial images. For a 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

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

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

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

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

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 to be employed

Matlab was used as the environment for PCA manipulation of the images, as well as for the neural network classification. Commands can be entered individually in Matlab, or can be combined in script files. Scripts were written to automate the process so that many variations on the basic experiment could be conducted. The scripts are included in Appendix A.

File images 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. Twenty images were reserved for testing. The top N principal components were projected onto the feature space and used as input to the neural network classifier. N was chosen to be 10, 40, and 100.

The NN had seven outputs, one of each different expression (angry, disgust, fear, happy, sad, surprise, neutral).

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 six times, using different initial weights each time. The best results of the six 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 15-20 hidden neurons (for the 40 and 100 PCA NN, respectively) and early stopping resulted in the best generalization. The NN used scaled conjugate gradient training algorithm, with MSE as the performance function.

Formats for presenting results

The results are presented in the form of progress screens, confusion matrices, and receiver operating characteristics (ROC) for the best run of each of the three experiments. The first experiment used 10 PCA eigenvectors, the second used 40 eigenvectors, and the third used 100 eigenvectors.

The progress screen provides feedback while the NN is training. It provides information on the number of epochs, performance, gradient, and validation checks for early stopping.

The confusion matrix shows the number of correct and incorrect responses, with the main diagonal containing the correct entries.

The ROC plot is generated from the confusion matrix and is used to compare classifiers.

Receiver Operating Characteristic

The ROC curve is a plot of the true positive rate versus the false positive rate as the threshold varied. The most frequently used performance measure in ROC analysis is the area under the ROC curve, or AUC. The AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. A perfect fit would cluster the points in the upper-left corner. Although Matlab doesn't provide the actual ROC, a visual inspection of the ROC can provide insight into the characteristics of a classifier.

To further clarify the ROC, consider a classifier that maps instances to one element of the set {positive, negative}. Given a classifier and an instance, there are four possible outcomes:

- If the instance is positive and it is classified as positive, it is a true positive.
- If the instance is positive and it is classified as negative, it is a false negative.
- If the instance is negative and it is classified as negative, it is a true negative.
- If the instance is negative and it is classified as positive, it is a false positive.

Metrics are commonly taken from an associated confusion matrix. Given a classifier and set of instances, a 2x2 confusion matrix is made using the counts of the correct and incorrect classifications of the instances. The diagonal represents the correct decisions, while the non-diagonal entries represent the errors between the classes. The true positive rate is then:

$$\text{tp rate} \approx \text{true postives/ total positives}$$

and the false positive rate is:

$$\text{fp rate} \approx \text{false positives/total negatives}$$

Points in ROC space are better the closer they are to the upper left corner. Classifiers on the left of the ROC graph may be thought of as conservative, in that they only make positive classifications with strong evidence. They make few false positives, but may miss many true positives as well. On the other hand, classifiers on the upper right side of an ROC graph may be thought of as liberal in that they make positive classifications on weak evidence so they classify nearly all the positives correctly but have a high false positive rate. Any classifiers below the y=x line would be worse than random guessing (Fawcett, 2003).

The area under the ROC curve (AUC) has the statistical property that it is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. It is effective as a scalar value to compare the performance of two classifiers. The AUC is equal to the Wilcoxon statistic, which is used to test the hypothesis that the distribution of some variable (x) from one population (p) is equal to that of a second population (n).

$$H0 : x_p = x_n.$$

If this null hypothesis is rejected, we can calculate the probabilities $x_p>x_n$, $x_p<x_n$, or $x_p \neq x_n$. For a classifier, we want $p(x_p>x_n)$ to be as close to unity as possible. The AUC effectively measures $p(x_p>x_n)$ (Bradley, 1997).

In this project, the ROC is more complex as multiple classes are involved. The confusion matrix is nxn (with n=7). Instead of representing a trade-off between true positives and false positives, the ROC reflects n benefits versus n^2-n errors (diagonal versus non-diagonal entries in the confusion matrix). It has been shown that ROC analysis extends to multiple classes as a performance measure (Fawcett, 2003).

Resources Used

This project was conducted on a Dell Latitude D500 laptop computer running Windows XP. This computer runs at 1.3GHZ and has 500MB of DDR.

The programming environment was the student version of Matlab, which is a high level language that is widely used in research and engineering. It is especially powerful when analyzing and manipulating vectors and matrices. This software is commercially available from MathWorks.

The images used in this study were from the Japanese Female Facial Expression database (Lyons, Akamatsu, Kamachi, & Gyoba 1998).

Summary

The raw images used as input for this project are comprised of ten individuals exhibiting seven different facial expressions. Principal component analysis was used to extract the top 10, 20, and 40 eigenvectors from these images. The best eigenvectors are determined by the value of the eigenvalues since the largest eigenvalue reflects the greatest variance in the image. The eigenvector projections are used as input to a multilayer neural network that is trained with a variation of gradient descent called scaled conjugate gradient. The minimum number of layers and hidden neurons was discovered through experimentation, and early stopping was used to avoid overfitting.

Matlab was used to conduct the experiments and scripts were written to automate the process. The scripts are included in Appendix A. Results are provided in the form of

progress screens, confusion matrices, and ROC graphs for the best network discovered for each of the sets of 10, 20, and 40 PCA eigenvectors used as feature vectors.

Chapter 4

Results

Findings

Numerous experiments were conducted for each of the three different sized sets of eigenvectors. The best network discovered for each of these three sets is presented below.

Results using 10 PCA Eigenvectorss

The following results were obtained using 10 principal components as input, 15 hidden neurons, and 7 output neurons. This network was retrained many times to obtain these results, as the typical network showed only 10%-20% accuracy. This suggests that the top 10 PCA eigenvectors do not adequately represent facial expressions. It is possible that this network was "trained" to recognize this specific set of test data and would not continue to maintain a 40% success rate on additional unseen examples.

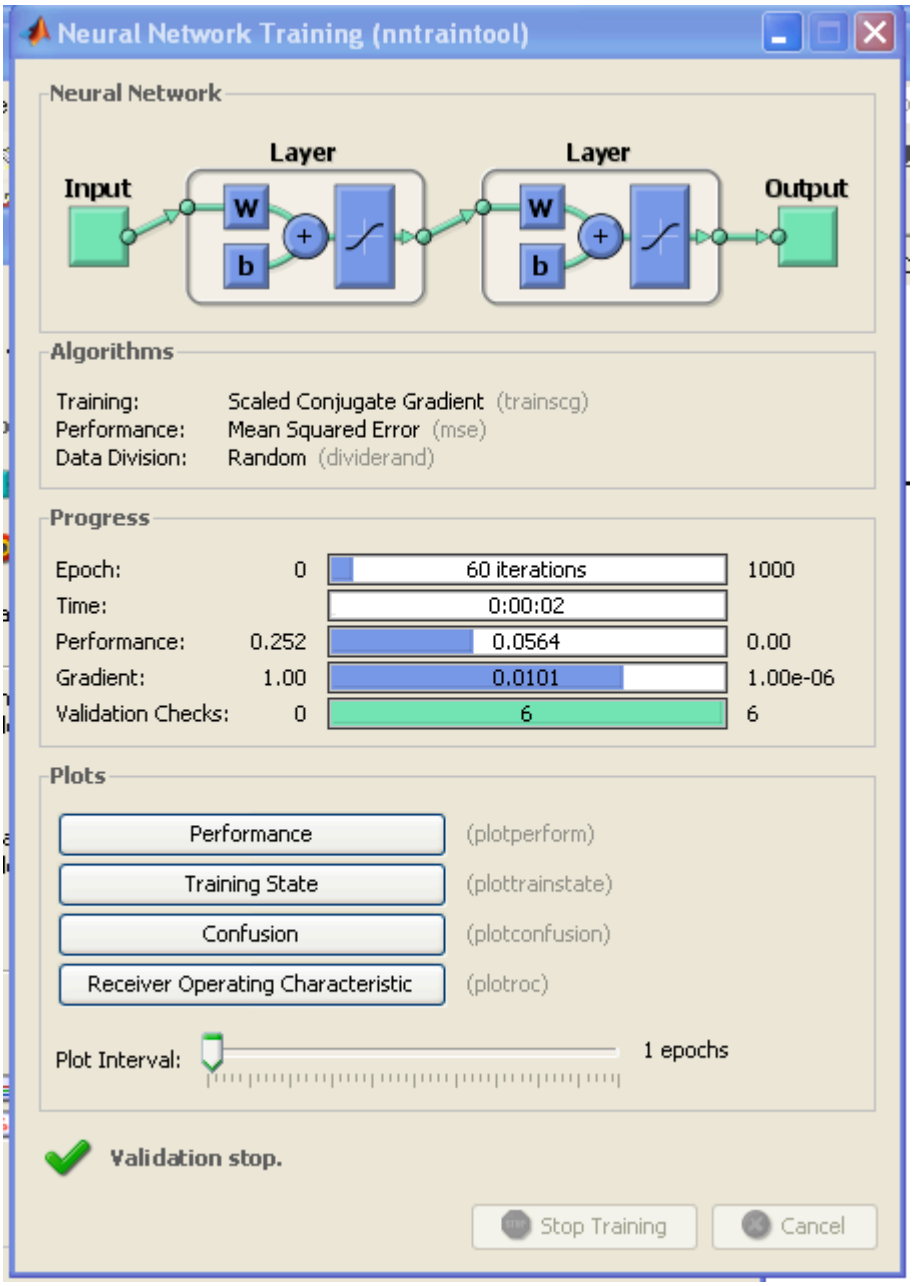


Figure 2: NN Progress Screen using 10 PCA Eigenvectors

For this NN, the progress shows that the training was stopped early on six validation checks. The gradient was .0101 and the MSE was .0564.

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 a 40% accuracy rate for the 20 test cases.

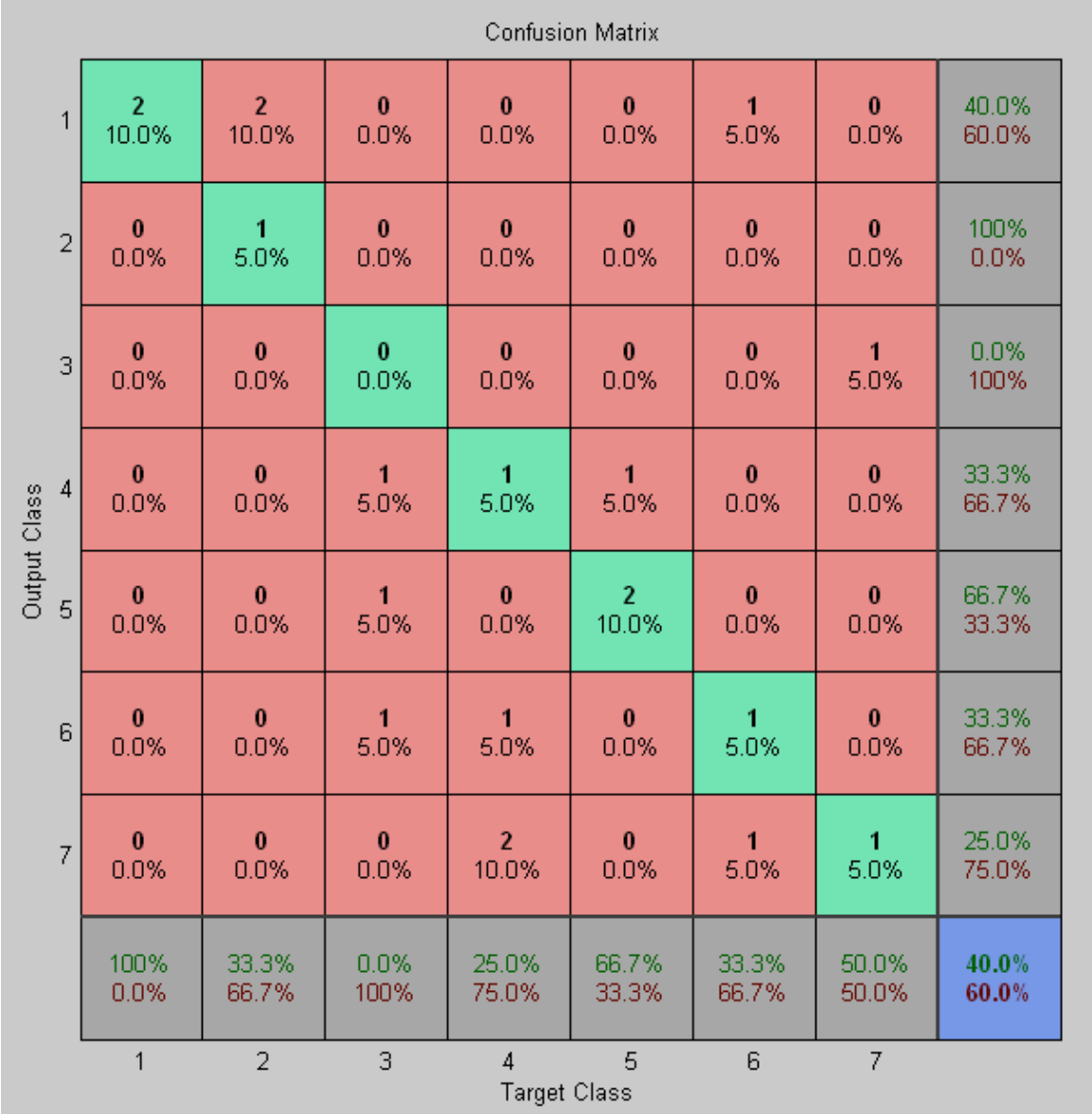


Figure 3: Confusion Matrix using 10 PCA Eigenvectors

The following two figures show the ROC obtained during the training phase, and the testing phase, respectively.

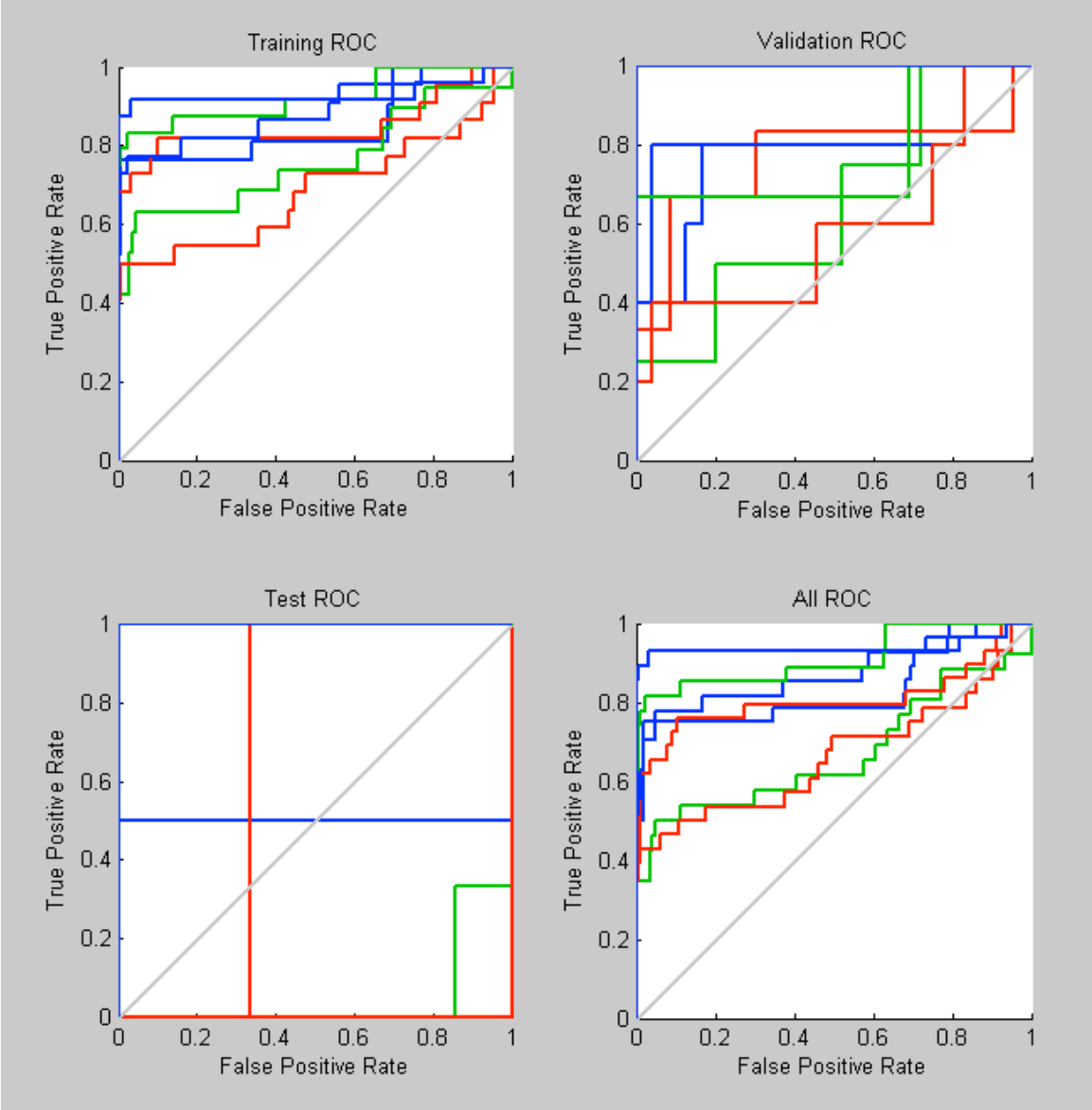


Figure 4: ROC curve for training, validation, and testing set for 10 PCA Eigenvectors

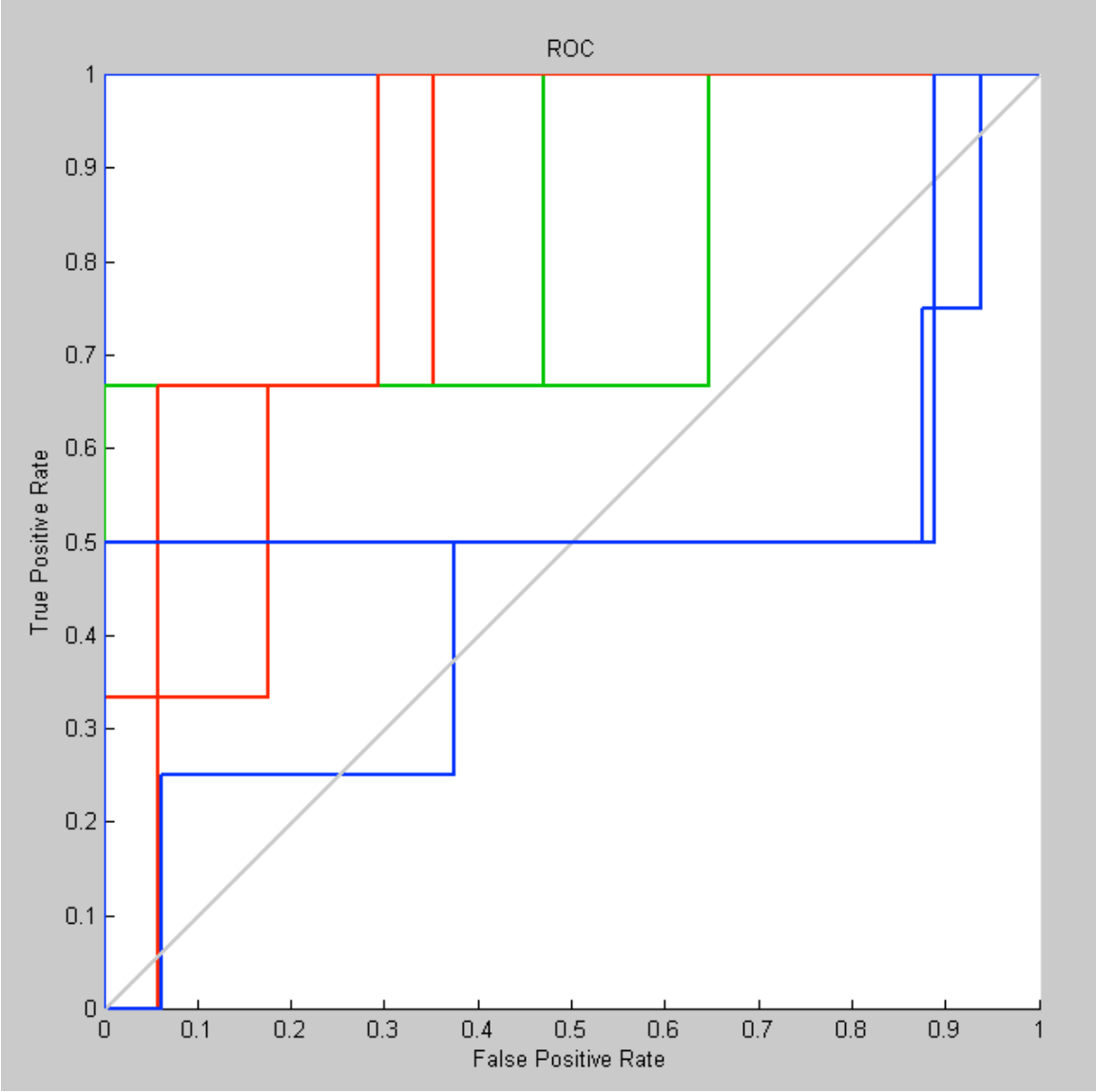


Figure 5: ROC curve for separate testing set with 10 PCA Eigenvectors

Results using 40 PCA Eigenvectors

The following results were obtained using 40 principal components as input, 20 hidden neurons, and 7 output neurons.

These results were typical over several retrainings of the network using these parameters, although the specific misclassifications were slightly different each time.

Sadness was the most often misclassified emotion, followed by neutral.

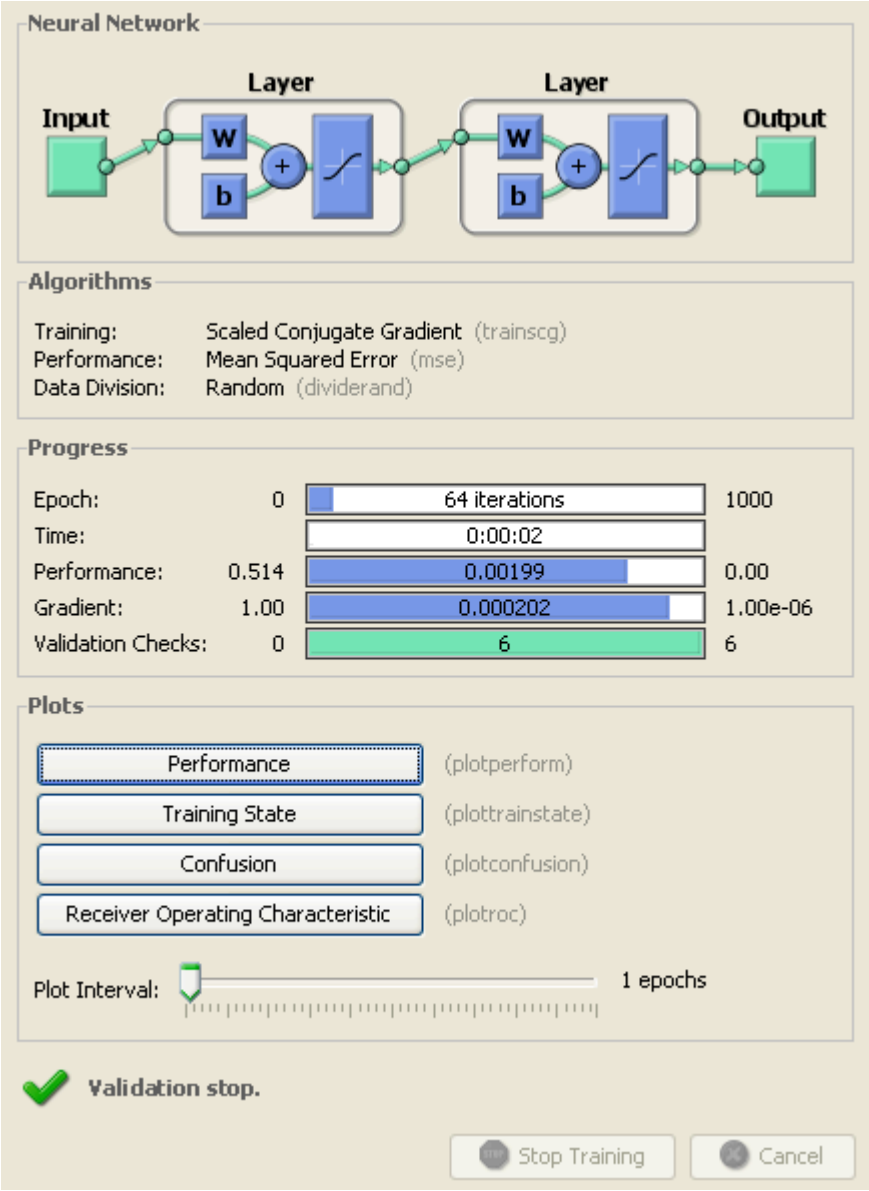


Figure 6: NN Progress Screen using 40 PCA Eigenvectors

The progress screen shows that the training was stopped early on six validation checks. The gradient was .000202 and the MSE was .00199.

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 80% accuracy rate for the 20 test cases.

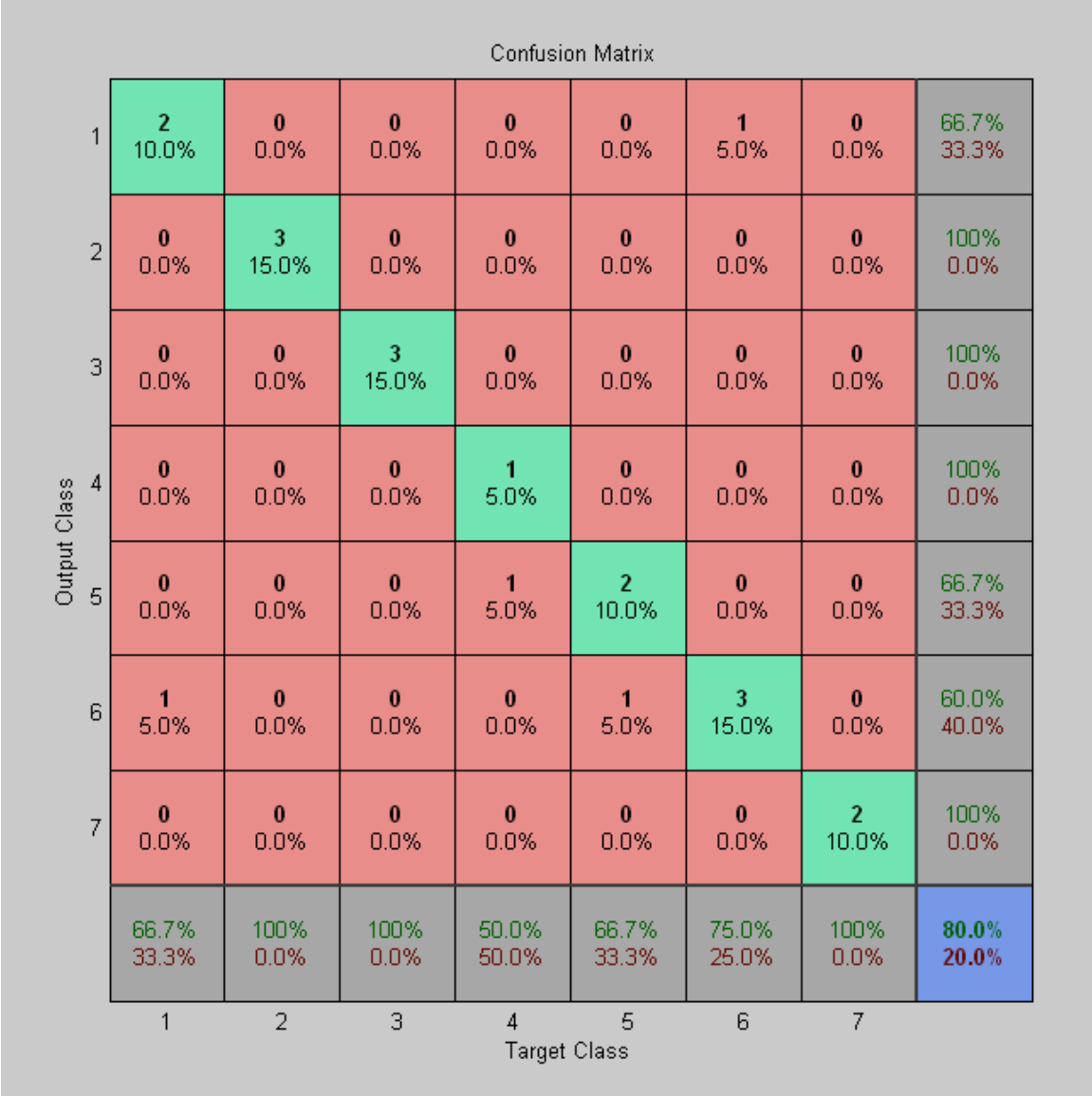


Figure 7: Confusion Matrix using 40 PCA Eigenvectors

The following two figures show the ROC obtained during training, and during testing, respectively. The clustering of the data near the upper left corner indicates a good, but not perfect, fit of the data to the expected values.

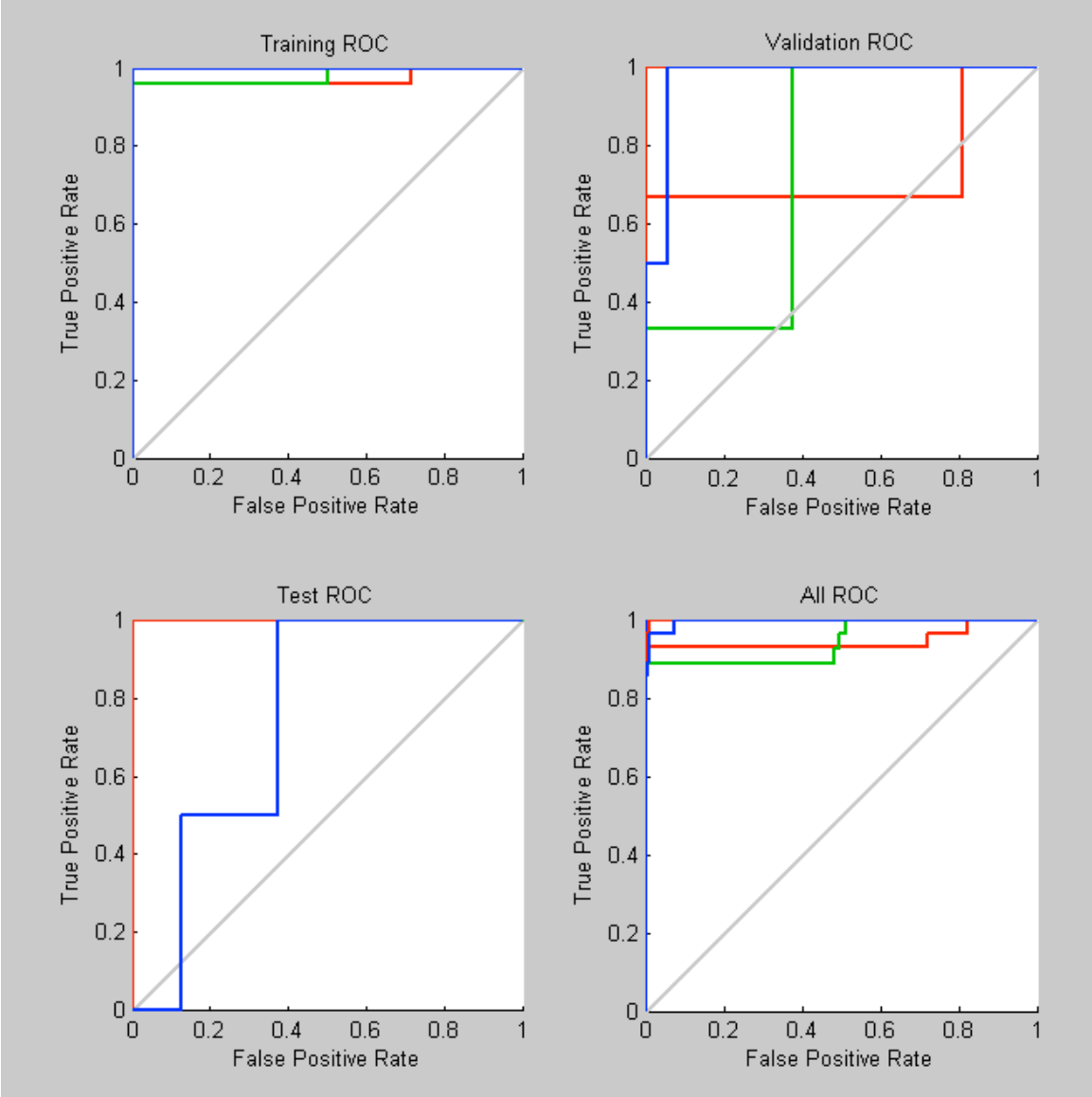


Figure 8: ROC curve for training, validation, and testing set for 40 PCA Eigenvectors

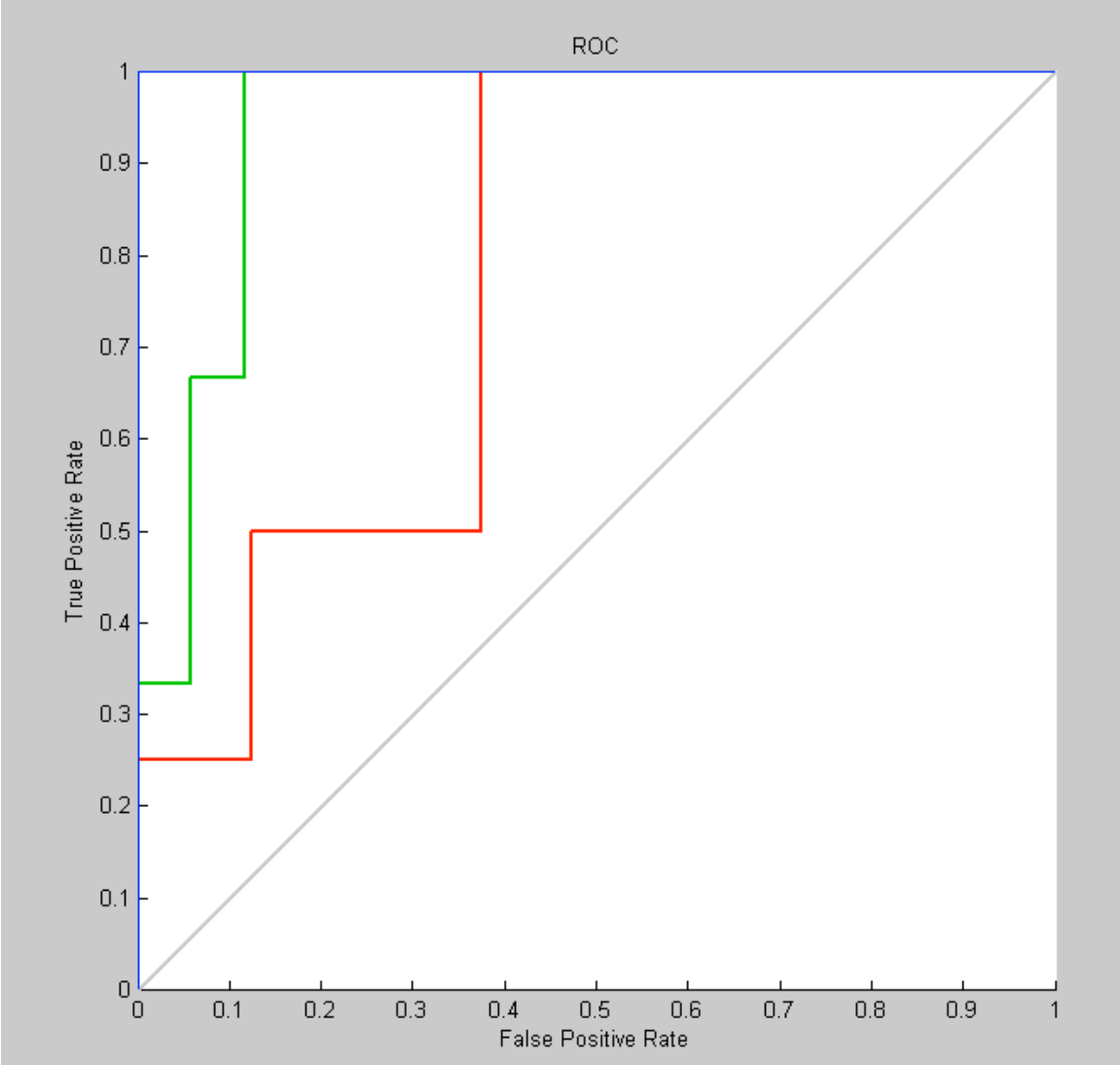


Figure 9: ROC curve for separate testing set

Results using 100 PCA Eigenvectors

An 80% success rate was also obtained using 100 principal components as input, 15 hidden neurons, and 7 output neurons. As with 40 PCA eigenvectors, these results were typical across many retrainings of the network.

It is interesting to note that a 75% success rate was obtained using only 10 hidden neurons, and a 70% success rate was obtained using a mere 5 hidden neurons. The 40 PCA network did not exhibit similar high success rates with a reduced number of hidden

neurons.

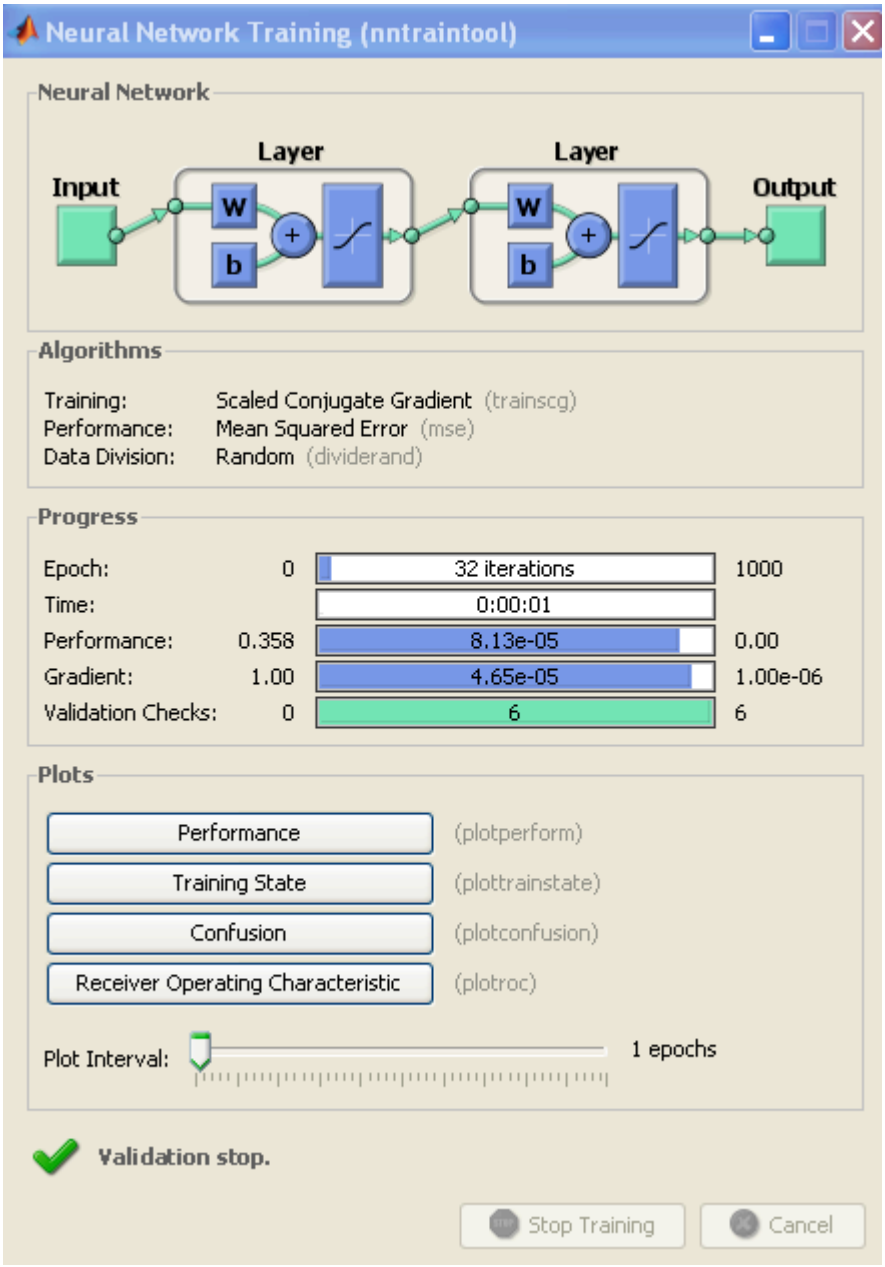


Figure 10: Progress window for 100 PCA Eigenvectors

The progress screen shows that the training was stopped early on six validation checks. The gradient was 4.65e-05 and the MSE was 8.13e-05, which is significantly better than the values achieved using 40 PCA eigenvectors.

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 80% accuracy rate for the 20 test cases.

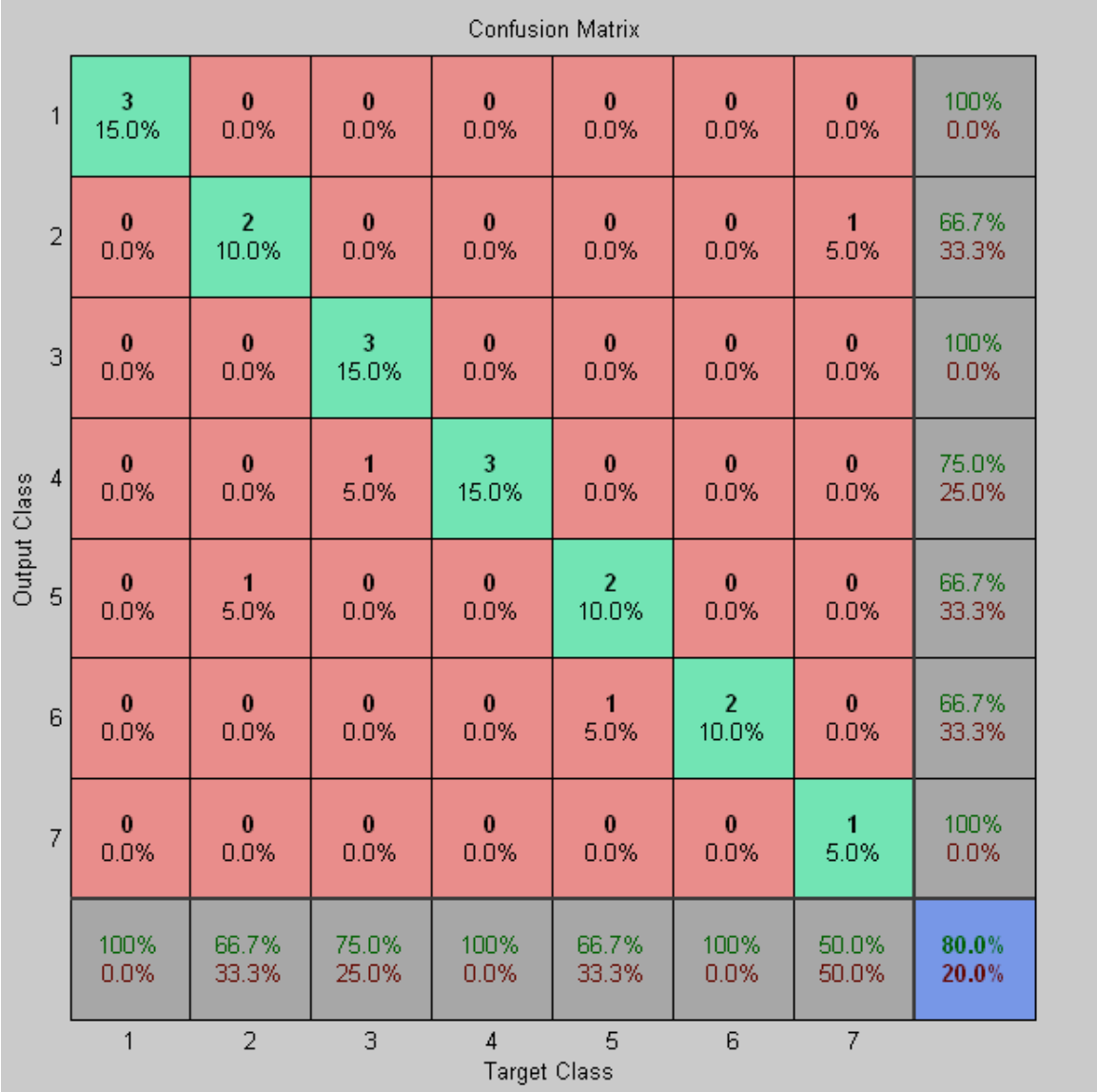


Figure 11: Confusion Matrix using 100 PCA Eigenvectors

The following figures show the ROC obtained during training and testing, respectively. The data clusters toward the upper left corner, indicating a better fit of the data to the expected values than that obtained with 40 PCA eigenvectors, even though the

number of correct classifications was the same.

ROCs measure the classifier's ability to get good relative scores. This is because the ROC is measuring the ability of the classifier to rank positive instances relative to the negative instances. From this perspective, the 100-PCA network performs significantly better than the 40-PCA network. Since it has been shown that ROC is preferable to overall accuracy as a performance measurement, this suggests that the 100 PCA NN would be a significantly better classifier on future unseen examples (Bradley, 1997).

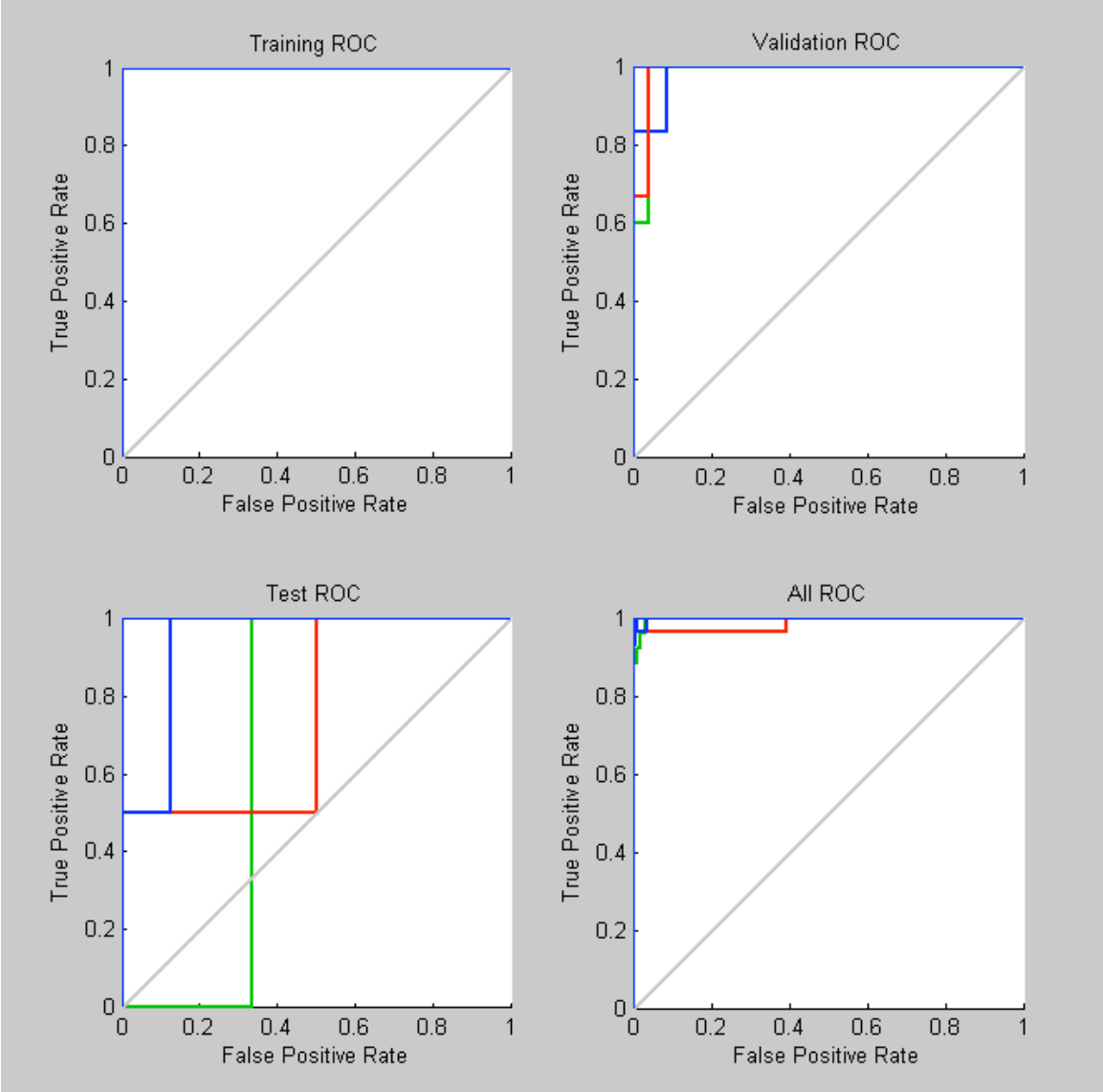


Figure 12: ROC curve for training, validation, and testing set for 100 PCA Eigenvectors

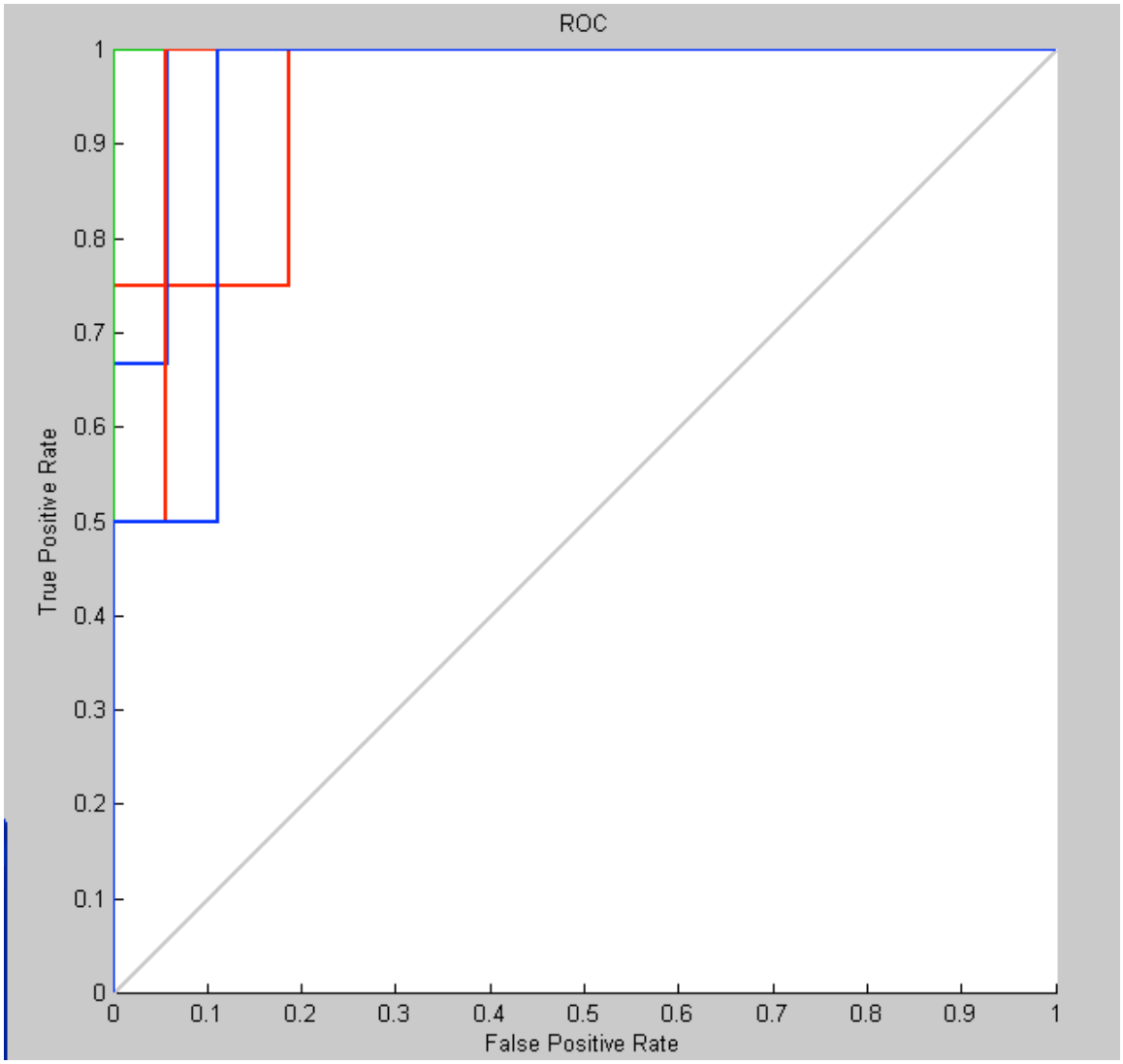


Figure 13: ROC curve for separate testing set for 100 PCA Eigenvectors

Summary of results

The best network configuration for the 10 PCA eigenvectors had a hidden layer with 15 neurons. It had a gradient was .0101 and an MSE of .0564. This network achieved a 40% success rate.

The best network configuration for the 20 PCA eigenvectors had a hidden layer with 20 neurons. It had a gradient of .000202 and an MSE of .00199. The ROC clustered well to the upper left of the diagonal. This network achieved an 80% success rate.

The best network configuration for the 100 PCA eigenvectors had a hidden layer with 15 neurons. It had a gradient of 4.65e-05 and an MSE of 8.13e-05. The ROC clustered close to the upper left corner of the graph. This network also achieved an 80% success rate.

Chapter 5

Conclusion, Implications, Recommendations, and Summary

Conclusions

The experiment using 10 PCA eigenvectors showed very poor performance. Typical accuracy was only in the 10-20% range. The best network obtained a 40% success rate, but since its results were atypical, it is suspected that the network was inadvertently trained to recognize this specific set of test data. It seems likely that 10 eigenvectors were not enough to capture the salient differences in the patterns needed for classification

The network using 40 PCA eigenvectors as input achieved an 80% success rate regularly, with a gradient of .000202 and an MSE of .00199. The ROC clustered well to the upper left of the diagonal, indicating a good fit of the data to the expected values.

The network using 100 PCA eigenvectors also reached an 80% success rate, and needed fewer neurons than the 40 PCA network. The gradient was 4.65e-05 and the MSE was 8.13e-05 which is significantly better than the 40 PCA network. Perhaps most significant, the ROC of the 100 PCA network clustered much closer to the upper left corner than that of the 40 PCA network. This indicates that even though performance was identical for both the 40 and 100 PCA networks, the 100 PCA network can be expected to classify additional unseen examples more accurately than the 40 PCA network.

These results support the hypothesis that PCA can obtain feature vectors from images to distinguish between many of the basic emotions using a feed forward neural network.

Implications

This research contributes to the understanding of the effectiveness of PCA and neural networks to being able to recognize different patterns in the same data based on how the network is trained. PCA has been shown many times to be effective at recognizing faces from a set of eigenvectors. This research shows that the same eigenvectors can be used to recognize emotions.

Recommendations

In order to achieve performance rates higher than 80%, more variations on the above experiments could be performed. Some of these include cropping the face so that only the eyes, nose, and mouth are visible; discarding the topmost N eigenvectors on the assumption they may contain more dominant pattern differences irrelevant to emotions; or continuing to increase the number of eigenvectors.

It is also possible that different types of neural networks might work better with this problem. Support Vector Machines in particular would be an interesting comparison to the standard multilayer network used in this research.

Summary

The project implemented a facial emotion recognition system, using PCA pre-processing and a feed-forward neural network for classification into seven categories. The PCA eigenvector projections were used as input to the NN for both training and

testing. Experiments were conducted using 10, 40, and 100 PCA eigenvectors. A dismal 40% success rate was the best obtained with the networks using 10 PCA eigenvectors, while an 80% success rate was common with networks using either 40 or 100 PCA eigenvectors. However, the 100 PCA eigenvector network had a more favorable MSE value and significantly better ROC plot, suggesting it would be superior in classifying additional unseen examples.

This research shows that a NN can be trained to recognize basic facial expressions using PCA projections of the entire face as feature vectors with an 80% accuracy rate.

Appendix A

Program Scripts

```
% ------------------------------------------------------
% runPcaScripts1 executes all the pca scripts after the filelist has
% been created up to getting the tests.  getTestIndices should be rerun
% until a good distribution is obtained.  Then runPcaScripts2 will
% execute the remaining scripts up to the neural network execution.
%
% inputs must exist in the matlab workspace, except filelist.dat which
% must exist on the harddrive. Outputs are in the workspace.
%
% input: list[ numImg, namelength ]
%        numPca
%        numTests
%        numCat
%        filelist.dat
% output: testIndices
%         testDistribution
%
% ------------------------------------------------------

% get list of emotions for each image
getEmotion;

% get all images in filelist.dat into a matrix
 [Imgs,w,h]=load_images('filelist.dat');

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

% get the Pca Projections
getPcas;

% add emotion vector before randomizing
ProjectionInv = [ProjectionInv Emotion];

% randomize rows
[PcaData, x]=randomize(ProjectionInv);

% subtract emotion vector after randomize
clear Emotion;
Emotion = PcaData(:,numPca+1);
PcaData(:,numPca+1)=[];

% randomly gets 20 indexes from dataset and
% shows how many of each category it contains.
% rerun this until distribution is acceptable.
getTestIndices;
```

```
% ------------------------------------------------------
% runPcaScripts2 executes the remaining scripts after obtaining
%    the desired test distribution up to the neural network execution.
%
% input PcaData[ numImg x numPca]
%       Emotion[ numImg x 1 ]
%       testIndices[ numTests ]
%       numImg, numTests, numCat
% outputs train_in[ numTrain, numPca ]
%       train_out[ numTrain, 1]
%       test_in[ numTests, numPca ]
%       test_out[ numTests, 1]
% ------------------------------------------------------

% separate training and test sets
getInputDist;

% turn values into categories
getEnumVector;




% ------------------------------------------------------
% getEmotion() - A script file that outputs a vector of the emotions
%                for each image in list
% input:   list(numImgs,15)
% outputs: Emotion[numImg]
%          Emotion_enum[numImg, 7]
%          numA, numD, numF, numH, numN, numS, numSu
% ------------------------------------------------------

numImg=length(list);
numA=0;
numD=0;
numF=0;
numH=0;
numN=0;
numS=0;
numSu=0;
Emotion=zeros(numImg,1);
Emotion_enum = zeros(numImg, 7);

for i=1:numImg
    if list(i,4)=='A'
        numA=numA+1;
        Emotion(i) = 1;
        Emotion_enum(i,1)=1;
    elseif list(i,4)=='D'
        numD=numD+1;
        Emotion(i) = 2;
        Emotion_enum(i,2)=1;
    elseif list(i,4)=='F'
```

```
        numF=numF+1;
        Emotion(i) = 3;
        Emotion_enum(i,3)=1;
    elseif list(i,4)=='H'
        numH=numH+1;
        Emotion(i) = 4;
        Emotion_enum(i,4)=1;
    elseif list(i,4)=='N'
        numN=numN+1;
        Emotion(i) = 5;
        Emotion_enum(i,5)=1;
    elseif list(i,4)=='S' && list(i,5)=='A'
        numS=numS+1;
        Emotion(i) = 6;
        Emotion_enum(i,6)=1;
    elseif list(i,4)=='S' && list(i,5)=='U'
        numSu=numSu+1;
        Emotion(i) = 7;
        Emotion_enum(i,7)=1;
    end;
end;
```

```
% ------------------------------------------------------------
function [Images,w,h] = load_images(filelist,downscale_f)
%LOAD_IMAGES Load a set of images listed in a file.
%
%            [IMGS,W,H] = LOAD_IMAGES(FILELIST) Treat each line of
%            the file named FILELIST as the filename of a PGM image,
%            and load each image as one column of the return array
%            IMGS.
%
%            LOAD_IMAGES(FILELIST,DOWNSCALE_F) Do the same thing,
%            but downscale each image's width and height by a factor
%            of DOWNSCALE_F.  Useful if the images are too big to be
%            loaded into memory all at once.
%
% This file adapted from http://www.cs.ait.ac.th/~mdailey/matlab/
% Matthew Dailey 2000
% ------------------------------------------------------------

  % Check argument consistency
  if nargin < 1 | nargin > 2
    error('usage: load_images(filelist[,downscale_f]');
  end;
  if nargin == 1
    downscale_f = 1.0;
  end;
  Images = []; old_w = 0; old_h = 0; w=0; h=0;

  % Open input file
  numimgs = linecount(filelist);
  fid = fopen(filelist,'r');
  if fid < 0 | numimgs < 1
```

```
    error(['Cannot get list of images from file "' filelist, '"']);
  end;

  % Get the images
  for i = 1:numimgs
    imgname = fgetl(fid);
    if ~isstr(imgname)            % EOF is not a string
      break;                      % Exit from loop on EOF
    end;
    fprintf(1,'loading PGM file %s\n',imgname);
    Img = imread(imgname);
    if i==1                       % If this is first image…
      old_w = size(Img,2);        % like sizes of the downscaled images
      old_h = size(Img,1);
      if downscale_f <= 1.0
    w = old_w; h = old_h;
      else
    w = round(old_w/downscale_f); h = round(old_h/downscale_f);
      end;
      Images = zeros(w*h,numimgs);   %preallocate size of return matrix
    end;
    if downscale_f > 1.0
      Img = im_resize(Img,w,h); % downscale using bicubic spline interp
    end;
    Images(1:w*h,i) = reshape(Img',w*h,1);   % Make a column vector
  end;
  fclose(fid);                    % Close the filelist when done

  fprintf(1,'Read %d images.\n',numimgs);

  % The function returns the output arguments Images, w, and h here.

End



% --------------------------------------------------------
function [Vectors,Values,Psi] = pc_evectors(A,numvecs)
%PC_EVECTORS Get the top numvecs eigenvectors of the covariance matrix
%          of A, using Turk and Pentland's trick for numrows >> numcols
%          Returns the eigenvectors as the colums of Vectors and a
%          vector of ALL the eigenvectors in Values.
%
% This file was adapted from http://www.cs.ait.ac.th/~mdailey/matlab/
% Matthew Dailey 2000
% --------------------------------------------------------

  % Check arguments
  if nargin ~= 2
    error('usage: pc_evectors(A,numvecs)');
  end;

  nexamp = size(A,2);
```

```
  % Now compute the eigenvectors of the covariance matrix, using
  % a little trick from Turk and Pentland 1991

  % Compute the "average" vector
  % mean(A) gives you a row vector containing the mean of each column
  % of A

  fprintf(1,'Computing average vector and vector differences from
avg...\n');
  Psi = mean(A')';

  % Compute difference with average for each vector

  for i = 1:nexamp
    A(:,i) = A(:,i) - Psi;
  end;

  % Get the patternwise (nexamp x nexamp) covariance matrix

  fprintf(1,'Calculating L=A''A\n');
  L = A'*A;

  % Get the eigenvectors (columns of Vectors) and eigenvalues (diag of
  % Values)

  fprintf(1,'Calculating eigenvectors of L...\n');
  [Vectors,Values] = eig(L);

  % Sort the vectors/values according to size of eigenvalue

  fprintf(1,'Sorting evectors/values...\n');
  [Vectors,Values] = sortem(Vectors,Values);

  % Convert the eigenvectors of A'*A into eigenvectors of A*A'

  fprintf(1,'Computing eigenvectors of the real covariance
matrix..\n');
  Vectors = A*Vectors;

  % Get the eigenvalues out of the diagonal matrix and
  % normalize them so the evalues are specifically for cov(A'), not
  % A*A'.

  Values = diag(Values);
  Values = Values / (nexamp-1);

  % Normalize Vectors to unit length, kill vectors corr. to tiny
  % evalues

  num_good = 0;
  for i = 1:nexamp
    Vectors(:,i) = Vectors(:,i)/norm(Vectors(:,i));
    if Values(i) < 0.00001
      % Set the vector to the 0 vector; set the value to 0.
```

```
      Values(i) = 0;
      Vectors(:,i) = zeros(size(Vectors,1),1);
    else
      num_good = num_good + 1;
    end;
  end;
  if (numvecs > num_good)
    fprintf(1,'Warning: numvecs is %d; only %d
exist.\n',numvecs,num_good);
    numvecs = num_good;
  end;
  Vectors = Vectors(:,1:numvecs);

end



% ------------------------------------------------------
% function[ ProjectionInv ] = getPcas(Imgs, numPca)
% GETPCAS(numPca) - A script file that gets the top numPcas PCAs from
% the image data
% inputs:   Imgs[numPixels x numImg]
%           numPca
% outputs:  Projection[numPca x numImg]
%           ProjectionInv[numImg x numPca]
% ------------------------------------------------------
x=size(Imgs);
numImg=x(2);

if numPca == 0;
    numPca = 40;
    fprintf('numPca not initialized: setting numPca to 40 ');
end;

for i=1:numImg
    tempImg = Imgs(:,i);
    Projection(:,i) = Vecs(:,1:numPca)'*(tempImg - Psi);
end

ProjectionInv = Projection';



% ------------------------------------------------------
function [ B,I ] = randomize( A,rowcol )
% randomize row orders or column orders of A matrix
% rowcol: if 0 or omitted, row order
%         if 1, column order
% ------------------------------------------------------
rand('state',sum(100*clock))
if nargin==1,
    rowcol=0;
end

if rowcol==0,
    [m,n]=size(A);
```

```
    p=rand(m,1);
    [p1,I]=sort(p);
    B=A(I,:);
elseif rowcol == 1,
    Ap=A';
    [m,n]=size(Ap);
    p=rand(m,1);
    [p1,I]=sort(p);
    B=Ap(I,:)';
end
```

```
% ---------------------------------------------------------
% GETTESTINDICES - A script file that randomly generates indices
% to be separated out for the test set of data.
% input: Emotion(numImg), numTests, numCat
% output: testIndices(numTests), testDistribution(numCat)
% ---------------------------------------------------------
testDistribution = zeros(numCat,1);

% populate random testIndices
testIndices = rand(1,numTests);         % get rand 0<num<1
testIndices=round(numImg*testIndices);  % minIndex<num<maxIndex

% get the counts for each category
% correct for zeros since array is 1-based
 for i=1:numTests
     if testIndices(i)==0
         testIndices(i)= 1;
     end;
     tempInx=testIndices(i);
     tempEm=Emotion(tempInx);
     testDistribution(tempEm) = testDistribution(tempEm)+1;
 end

 % display the counts
 testDistribution
```

```
% ---------------------------------------------------------
% GETINPUTDIST - A script file that separates the testing and
% training data from the complete pca data set.
% input: PcaData[numImg x numPca]
%        Emotion[numImg x 1]
%        testIndices[numTests]
%        numImg, numTests
% outputs: train_in[ numTrain, numPca ]
%        train_out[ numTrain, 1]
%        test_in[ numTests, numPca ]
%        test_out[ numTests, 1]
% ---------------------------------------------------------
```

```
test=1;
train=1;
isTest = 0;
clear test_in;
clear test_out;
clear train_in;
clear train_out;

numImg = length(PcaData);

for i=1:numImg
    % determine if this is a test case
    for j=1:numTests
        if i==testIndices(j)
            isTest = 1;
            tempx=Emotion(i);
            j=numTests;
        end
    end

    if isTest==1          % save test case
        test_in(test,:) = PcaData(i,:);
        test_out(test,:)=Emotion(i,:);
        test = test+1;
        isTest = 0;
    else              % save training case
        train_in(train,:) = PcaData(i,:);
        train_out(train,:)= Emotion(i,:);
        train = train+1;
    end;
end;

train_in = train_in';
train_out = train_out';
test_in = test_in';
test_out = test_out';



% ------------------------------------------------------
% getEnumVector - A script file that converts values to categories
%  input: numImg, numCat, numTests
%         train_out[ numTrain, 1]
%         test_out[ numTests, 1]
%  output train_out_enum[ numCat, numTrain ]
%         test_out_enum[ numCat, numTests ]
% ------------------------------------------------------

numTrain = numImg-numTests;

train_out_enum = zeros(numCat, numTrain);
test_out_enum = zeros(numCat, numTests);

for i=1:numTrain
    x=train_out(i);
```

```
    train_out_enum(x,i)=1;
end;

for i=1:numTests
    x=test_out(i);
    test_out_enum(x,i)=1;
end;
```

Reference List

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

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30(7), 1145-1159.

Donato, G., Bartlett, M. S., Hager, J. C., Ekman, P., and Sejnowski, T. J. (1999). Classifying Facial Actions. *IEEE Trans. Pattern Anal. Mach. Intell*. 21:10, Oct. 1999, 974-989.

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

Ekman, P., Friesen, W.V. (1978). Facial Action Coding System Investigator's Guide, Consulting Psychologist Press, Palo Alto, CA, 1978.

Eleyan, A., and Demirel, H. (2005) Face Recognition System based on PCA and Feedforward Neural Networks, *Proc IWANN 2005, Lecture Notes in Computer Science,* v3512, 935-942.

Fawcett, T. (2003). ROC graphs: notes and practical considerations for data mining researchers. *Technical Report HPL-2003–4*, Palo Alto, CA. HP Laboratories.

Hu, T., De Silva, L.C., and Sengupta, K. (2002). A hybrid approach of NN and HMM for facial emotion classification. *Pattern Recognition Letters* 23, 11 (Sep. 2002), 1303-1310.

Kapoor, A., Qi, Y., and Picard, R. W. (2003). Fully Automatic Upper Facial Action Recognition. *In Proceedings of the IEEE international Workshop on Analysis and Modeling of Faces and Gestures*. 10-17, 2003.

Kharat, G. & Dudul, S. (2008). Neural Network Classifier for Human Emotion Recognition from Facial Expressions Using Discrete Cosine Transform. *In Proceedings of the 2008 First international Conference on Emerging Trends in Engineering and Technology*. 653-658.

Khashman, A. (2008) A Modified Backpropagation Learning Algorithm with added emotional coefficients. *IEEE Transactions on Neural Networks*. 19, 11 (Nov, 2008), 1896-1909.

Kurihara, K., Sugiyama, D., Matsumoto, S., Nishiuchi, N., and Masuda, K. 2009. Facial emotion and gesture reproduction method for substitute robot of remote person.

Comput. Ind. Eng. 56, 2 (Mar. 209), 631-647.

Lien, J.J., Kanade, T., Cohn, J., and Li, C. (1999) Detection, Tracking, and Classification of Action Units in Facial Expression. *Journal of Robotics and Autonomous Systems*, July, 1999.

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*, 200-205.

Masters, T. (1993). Practical Neural Network Recipes in C++. Academic Press Professional, Inc.

Reilly, J., Ghent, J., and McDonald, J. (2007). Non-Linear Approaches for the Classification of Facial Expressions at Varying Degrees of Intensity. *In Proceedings of the international Machine Vision and Image Processing Conference.* 125-132.

Rizon, M., Yaacob, S., Desa H., Karthigayan M., Hashim F., Saad P., Shakaff, A., Saad, A., Mamat M. (2006). Face Recognition using Eigenfaces and Neural Networks. *American Journal of Applied Sciences* 2 (6): 1872-1875, 2006.

Sahoolizadeh, A., Heidari, B., Dehghani, C. (2008). A New Face Recognition Method using PCA, LDA and Neural Network. *International Journal of Computer Science and Engineering*. 2:4, 2008.

Turk, M., & Pentland, A. (1991). Eigenfaces for Recognition, *Journal of Cognitive Neuroscience*, Vol. 3, pp. 71-86, 1991.

Whitehill, J. & Omlin, C. (2006). Haar Features for FACS AU Recognition. *In Proceedings of the 7th international Conference on Automatic Face and Gesture Recognition*, 97-101.