AMATH 582 Final Project: Facial Expression Recognition

Huasong Zhang
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
huasongz@uw.edu

Jun Song
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
juns113@uw.edu

Abstract

In this paper, we applied five classification methods to JAFFE [5]. These five methods are Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Naive Bayes and Neural Network. For pre-processing, we applied Principal Component Analysis (PCA) to reduce the dimension of the image data matrix so as to improve the computing efficiency and to solve the issues associated with the curse of dimensionality. At last, the accuracy of these five classification methods are compared.

1 Introduction and Overview

The human face plays a central role in social interaction, hence it is not surprising that automatic facial information processing is an important and highly active subfield of pattern recognition research [2]. The face displays a complex range of information about identity, age, gender, race well as emotional and attentional state. This paper focuses on the problem of recognizing facial expression of an individual face from single digital images with various classification models such as SVM, LDA, KNN, Naive Bayes and Neural Network. The paper also explores the uses of PCA to find the optimal dimension for best classification performance. The examples chosen to demonstrate our method are facial expressions. However, the technique may extend to other facial attributes.

The dataset we used is Japanese Female Facial Expression (JAFFE) Database [5], which contains 213 images of 7 facial expressions (6 basic facial expressions and 1 neutral) posed by 10 Japanese female models (Figure. 1). These 6 basic facial expressions are happy, sad, fear, surprise, disgust, and anger. We used this dataset because this dataset contains only Japanese women, which not only specifies culture, but also gender. Whether facial expressions are innate and universal versus learned and culture dependent is a subject of debate [1]. In this project, We only want to focus on the facial expressions attribute instead of other attributes such as race and gender. So this dataset brings simplicity to our experimental design. Also, all the photos are taken in the front and with the same light exposure and other natural conditions. Therefore, the classification result will not be affected by the quality of the photos and other surrounding factors.

2 Theoretical Background

2.1 Curse of Dimensionality

In machine learning problems there often involves tens of thousands of features for each training instance. This can be a problem as it makes our training extremely slow and prone to overfitting. This problem is commonly referred to as the curse of dimensionality. Hughes Phenomenon (Figure. 2) shows that as the number of features increases, the classifier's performance increases as well until we

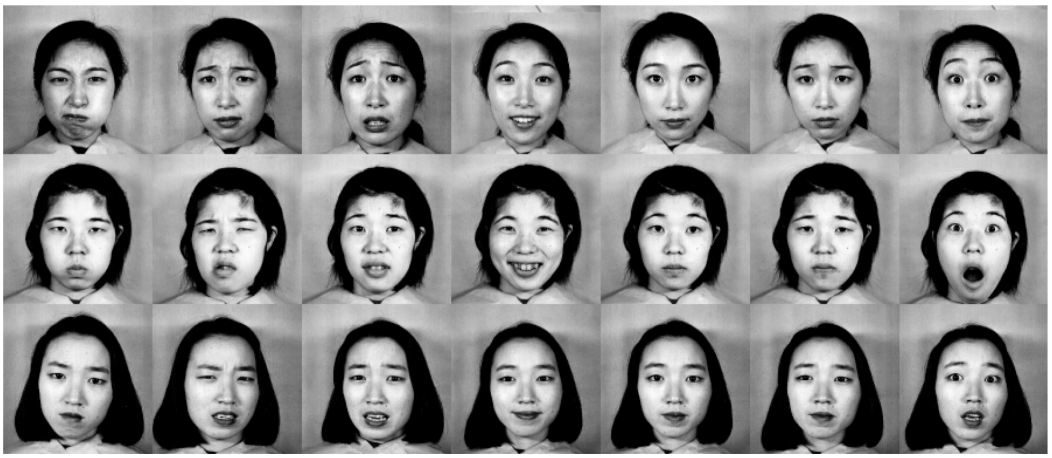

Figure 1: JAFFE samples

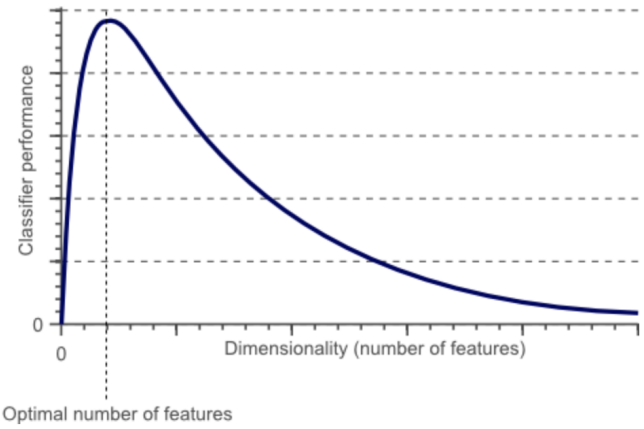

Figure 2: Hughes Phenomenon

reach the optimal number of features. Adding more features based on the same size as the training set will then degrade the classifier's performance.

Because of the issues associated with the curse of dimensionality, it is necessary to reduce the number of features/dimensions considerably to help increase our model's performance and enables us to arrive at an optimal solution for our machine learning model.

2.2 PCA

PCA addresses two objectives:

- Generally, it is desirable to reduce the number of features used to represent the data.
- Generally, it is desirable for the set of features to describe a large amount of "information."

Principal components analysis (PCA) can find low dimensional approximations to the data by projecting the data onto linear subspaces. PCA attempts to create features or dimensions sequentially based upon the amount of information they capture. We can do Singular Value Decomposition (SVD) on the large data matrix and get U, Σ, and V matrices. If the data matrix X to apply svd is a $n \times p$ matrix where n is the number of pixels in each image, and p is number of images. Then U of size $n \times n$ is eigenface basis. Σ has diagonals of singular values in descending order, it will have p

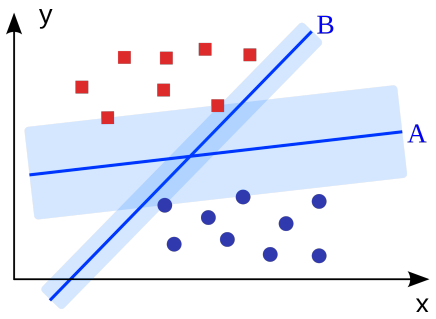

Figure 3: SVM

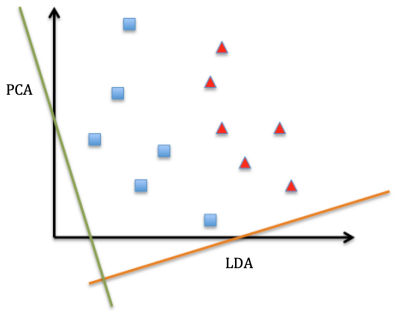

Figure 4: LDA and PCA

singular values in total. V of size $p \times p$ is a coefficient matrix. By plotting the Σ matrix, we can find out the rank r of the data matrix. When low-dimensional structure is present in the data (the singular values of Σ decrease sharply to zero with only a limited number of dominant modes), rank r will be much smaller than original dimension n. Therefore, we can use the rank to build the reconstructed matrix. This is an important pre-processing step to achieve dimension reduction and to solve issues associated with the curse of dimensionality.

2.3 SVM

Support Vector Machine is a classification method where we are trying to find the best hyperplane to separate different classes. The word "best" means the distance between the hyperplane and the classes it separates can be maximized. As we can see in Figure. 3 that line A and B both separate two groups of data. However, the distance between the closest data from two groups and B line is much closer comparing to that for line A. Therefore, line A is the one we want since the distance is maximized. A is the best decision boundary. The algorithm intends to find this perfect hyperplane, so that when we apply the test set on the trained model, the model can classify the test data based on which side of the hyperplane it belongs to.

2.4 LDA

Linear Discriminant Analysis is another classification method which intends to project all the data on a new axis where the separation of clusters can be maximized. During the process of doing LDA, the dimensionality of the matrix gets reduced at the same time. However, the difference between PCA and LDA is that PCA tries to reduce the dimension while still captures the largest variance of all the data; and LDA tries to best separate the data. Figure. 4 best represents the difference between PCA and LDA. By applying LDA algorithm, we will get the trained model and when we have new test data, the data will be separated and classified based on the train model which are those hyperplanes.

2.5 KNN

KNN (K Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data (a vector) from other. The KNN's steps are [3]:

1. Receive an unclassified data;
2. Measure the distance (Euclidian, Manhattan, Minkowski or Weighted) from the new data to all others data that is already classified;
3. Gets the K(K is a parameter that you difine) smaller distances;
4. Check the list of classes had the shortest distance and count the amount of each class that appears;
5. Takes as correct class the class that appeared the most times;
6. Classifies the new data with the class that you took in step 5;

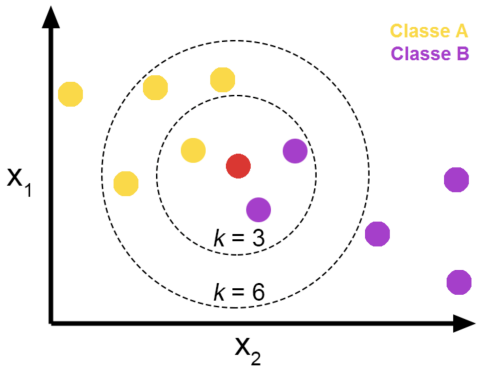

Figure 5: KNN Example

In Figure. 5, when $K = 3$, among the 3 neighbor points, there are 2 from purple class and 1 from yellow class. So we will classify the test data point as purple. However, if we choose $K = 6$, among the 6 neighbor points, there are 4 from yellow class and 2 from purple class. So we will classify it as yellow.

2.6 Naive Bayes

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction .

2.7 Neural Network

Artificial neural networks (ANN) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

The components of neural networks are [7]:

- Neurons: A neuron with label j receiving an input $p_j(t)$ from predecessor neurons consists of the following components: an activation $a_j(t)$, the neuron's state; a threshold θ_j; an activation function f that computes the new activation $a_j(t+1)$ from $a_j(t)$, θ_j and $p_j(t)$; an output function f_{out} that computes the output of activation.
- Connections, weights and biases: The network consists of connections, each connection transfers the output of a neuron i to the input of a neuron j. Each connection is assigned a weight w_{ij}.
- Propagation function: The propagation function computes the input $p_j(t)$ to the neuron j from the outputs $o_i(t)$ of predecessor neurons and typically has the form:

$$p_j(t) = \sum_i o_i(t) w_{ij}$$

  When a bias value is added with the function, the above form changes to the following:

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{oj}$$

- Learning rule: The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output.

This learning process typically amounts to modifying the weights and thresholds of the variables within the network.

3 Algorithm Implementation and Development

3.1 Build original data matrix

Each face image is reshaped into a column vector then appended to data matrix X. The data matrix X is of size $n \times p$ where n is the number of pixels of each image and p is the number of images.

3.2 Separate training and test set

In order to get the unbiased classification rate, we first separated the entire data matrix into training set and testing set by using randperm to permute numbers. We took out the first 70% of columns as our training set X_{train} and the last 30% percent as our test set X_{test}. We applied the same permutation to the labels as well.

3.3 Pre-processing: PCA with SVD for dimension reduction

We applied SVD to data matrix: $X = U\Sigma V^*$. U of size $n \times n$ is eigenface basis. Σ has diagonals of singular values in descending order, it will have p singular values in total. V of size $p \times p$ is a coefficient matrix. In order to find out the effects of reconstructed matrix, we tried several thresholds: 0.5, 0.7, 0.9, 0.95 and 0.99. It means we reconstruct the data matrix X with 50%, 70%, 90%, 95% or 99% energy preserved.

The energy percentage of the first r principal components is: $(\sum_1^r \sigma_i)/(\sum_1^p \sigma_j)$ where σ_i is the singular value of the i-th principal component. To build a reconstructed matrix with rank r, we used $X_{\text{reconst}} = U_r' \cdot X$ where Ur is the first r columns of the eigenbasis U. X_{reconst} will be of size $r \times p$, thus we reduced the dimension of an face image from n to r.

3.4 Classify with SVM, LDA, KNN, Naive Bayes, Neural Network

Then we trained MATLAB built in SVM, LDA, Naive Bayes and KNN classifier models with reconstructed training set ($U_r' \cdot X_{train}$) and classify the reconstructed test set ($U_r' \cdot X_{test}$).

For Neural Network, we used imported framework Simple Neural Network [6]. And when computing classification accuracy, we averaged the results over 10 experiments. The classification accuracy is defined as $\frac{1}{n}\sum_{i=1}^{n} I(\tilde{y_i} == y_i)$, where y_i is the true label, $\tilde{y_i}$ is the predicted label, $I(e) = 1$ if e is true and $I(e) = 0$ otherwise.

4 Computational Results

4.1 Hughes Phenomenon on SVM, LDA, KNN, Naive Bayes

For dimension reduction, we chose five thresholds: 0.5, 0.7, 0.9, 0.95. 0.99. It means we reconstruct the image data matrix with 50%, 70%, 90%, 95% or 99% energy preserved respectively. As shown in Figure 6 and Figure 7 that for the same image, the ones constructed by 50 percent do not have clear boundaries of the faces. While, the reconstructed images by using 90 percent of the energy are pretty clear; they are even as clear as the original images. Therefore, the higher threshold we use, the more clear the reconstructed image will be. As the energy preserved goes up, we expect the classification accuracy to increase and then decrease due to Hughes Phenomenon (See Figure. 2).

The results for the SVM, LDA, KNN are the same as we expected. As the threshold increases from 50% to 90%, the accuracy rate increases; and then it decreases when threshold goes above 90%. For Naive Bayes, the accuracy rate increases as the threshold increases. Thus, reconstructing data matrix with approximately 90% energy preserved will yield the best performance.

	Threshold				
Accuracy	0.5	0.7	0.9	0.95	0.99
SVM	0.42188	0.6875	0.71875	0.64062	0.64062
LDA	0.21875	0.70312	0.8125	0.82812	0.35938
KNN	0.71875	0.76562	0.78125	0.78125	0.76562
Naive Bayes	0.25	0.60938	0.625	0.625	0.67188

Table 1: Accuracy for SVM, LDA, KNN, Naive Bayes

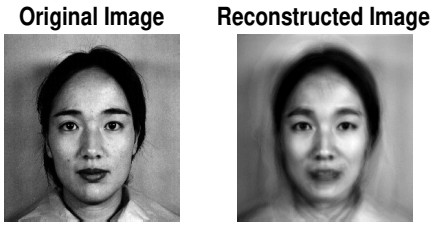

Figure 6: Reconstructed image with 50% energy

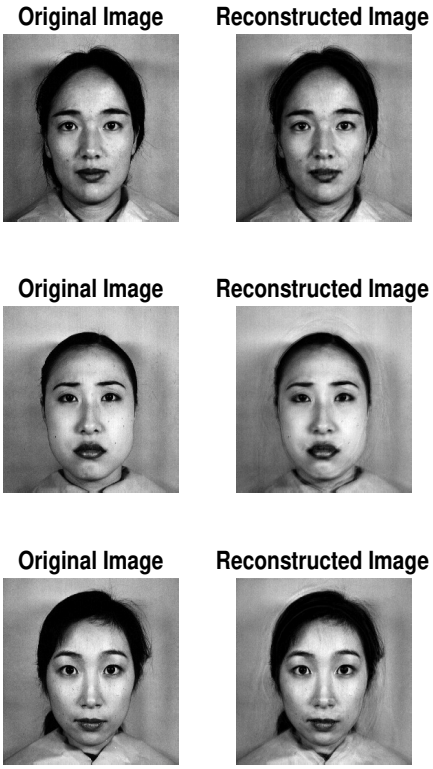

Figure 7: Reconstructed image with 90% energy

4.2 Neural Network

From the result in 4.1, we found that reconstructing data matrix with 90% energy preserved had the best performance. Thus we used this setting in our neural network experiment. Then we built neural network with different activation functions (tanh and sigma) and different layer structure (1, 2 or 3 hidden layers). Note: [250 150 100] in Table. 4.2 means 3 hidden layers with $250, 150$ and 100 nodes each.

	Hidden Layer Structure			
Accuracy	[]	[50]	[150 100]	[250 150 100]
tanh	0.75156	0.80781	0.79063	0.72344
sigm	0.73281	0.76562	0.84531	0.80781

Table 2: Accuracy for neural network with different activation functions/ structures

As shown in Table. 4.2, we found that one or two hidden layers slightly improved the accuracy but the performance stopped improving as we added more layers. Tanh and sigma activation functions did not show much difference in terms of performance. With proper layer structure, the neural network classification method can reach 85% accuracy. Thus, simple neural network has a better performance than LDA, SVM, KNN and Naive Bayes classifier models.

5 Summary and Conclusions

In this project, we successfully reduced the number of features/dimensions with PCA to help increase our model's performance on JAFFE dataset [5]. We found an approximate optimal solution (dimension reduced with 90% energy preserved) for our machine learning models such as SVM, LDA, KNN and Naive Bayes and reached a classification accuracy around 80%. We then built and trained simple neural networks with various hidden layer structures and activation functions, which further improved our classification accuracy to 85%. An interesting future direction is to use a more complex neural network on JAFFE dataset [5], for example AlexNet [4].

6 Appendix A: MATLAB functions

1. *randperm*: Randomly permute numbers from 1 to n. We used this command to build training and test set.
2. *fitcecoc*, *fitcdiscr*, *fitcnb*, *fitcknn*, *predict*: We used these functions to train SVM, LDA, Naive Bayes and KNN models respectively and to predict with the trained models.
3. *learnNN*, *predictNN*: We used these functions to build and train easy neural network and to predict with the trained neural network model. Imported from [6].

7 Appendix B: MATLAB codes

7.1 SVM, LDA, Naive Bayes, KNN

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Load Images %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
directory = 'jaffe/*.tiff';
npixels = 65536; x = 256; y = 256;


imgFiles = dir(directory);
numFiles = length(imgFiles);
% X: n by p — n # of pixels in each image, p # of images
X = zeros(npixels, numFiles);
% Y: 1 by p — labels
Y =[];

% Load each face image as a column vector to form a data matrix X.
% Extract labels to form a label matrix Y.
for k = 1:numFiles
    img = imread(strcat(imgFiles(k).folder,'/',imgFiles(k).name));
    img_vector = reshape(img, npixels, 1);
    label = imgFiles(k).name(4:5);
    if label == "AN"
        label = 1;
    elseif label == "DI"
        label = 2;
    elseif label == "FE"
        label = 3;
    elseif label == "HA"
        label = 4;
    elseif label == "SA"
        label = 5;
    elseif label == "SU"
```

```
        label = 6;
    elseif label == "NE"
        label = 7;
    end
    X(:,k) = img_vector;
    Y = [Y;label];
end

% Separate training and testing set
P = 0.7;
idx = randperm(numFiles);
X_train = X(:,idx(1:round(P*numFiles)));
X_test = X(:,idx(round(P*numFiles)+1:end));
Y_train = Y(idx(1:round(P*numFiles)),:);
Y_test = Y(idx(round(P*numFiles)+1:end),:);

%%%%%%%%%%%%%%%%%%% SVM/LDA/KNN/Naive Bayes with PCA %%%%%%%%%%%%%%%%%%%%%%%%%
[u,s,v] = svd(X_train,'econ');
figure(4)
plot(diag(s)/sum(diag(s)),'ro','LineWidth',[2])
xlabel('Images')
ylabel('Energy')
title('Singular value spectrum for images')
print(gcf,'-dpng','s_plot.png');

e = diag(s)/sum(diag(s));

threshold = [0.5 0.7 0.9 0.95 0.99];

sample_idx = [2,50,100];
for j = 1:length(threshold)
    th = threshold(j);
    E = 0;
    for i = 1:length(e)
        E = E + e(i);
        if (E > th)
            break
        end
    end
    recon_faces = u*s(:,1:i)*v(:,1:i)';
    recon_train = u(:,1:i)'*X_train;
    recon_test = u(:,1:i)'*X_test;
    % reconstruction of faces
    figure('position', [0, 0, 500, 800]);
    for i = 1:length(sample_idx)
        subplot(3,2,2*i-1);
        imagesc(uint8(reshape(X_train(:,sample_idx(i)),256,256)));
        colormap(gray);
        axis off;
        title('Original Image');
        set(gca,'FontSize',18);
        subplot(3,2,2*i);
        imagesc(uint8(reshape(recon_faces(:,sample_idx(i)),256,256)));
        colormap(gray)
        axis off;
        title('Reconstructed Image');
        set(gca,'FontSize',18);
    end

    %%% SVM
    svm_mod2 = fitcecoc(recon_train.',Y_train);
    svm_label2 = predict(svm_mod2,recon_test.');
    wrong_predicts = find(Y_test - svm_label2);
    svm_pca_accuracy = (length(Y_test) - length(wrong_predicts))...
        /length(Y_test);
```

```
    svm_pca_A = ['Accuracy for using SVM after dimension reduction for',...
        ' a threshold of ',num2str(th),' is ', num2str(svm_pca_accuracy)];
    disp(svm_pca_A)

    %%% LDA
    lda_mod = fitcdiscr(recon_train.',Y_train);
    lda_label = predict(lda_mod,recon_test.');
    wrong_predicts = find(Y_test ~= lda_label);
    lda_accuracy = (length(Y_test) - length(wrong_predicts))...
        /length(Y_test);
    lda_A = ['Accuracy for using LDA after dimension reduction for ', ...
        'a threshold of ', num2str(th),' is ', num2str(lda_accuracy)];
    disp(lda_A)

    %%% Naive Bayes
    nb_mod = fitcnb(recon_train.',Y_train);
    nb_label = predict(nb_mod,recon_test.');
    wrong_predicts = find(Y_test ~= nb_label);
    nb_accuracy = (length(Y_test) - length(wrong_predicts))...
        /length(Y_test);
    nb_A = ['Accuracy for using Naive Bayes after dimension reduction '...
        'for a threshold of ',num2str(th),' is ', num2str(nb_accuracy)];
    disp(nb_A)

    %%% KNN
    knn_mod = fitcknn(recon_train.',Y_train);
    knn_label = predict(knn_mod,recon_test.');
    wrong_predicts = find(Y_test ~= knn_label);
    knn_accuracy = (length(Y_test) - length(wrong_predicts))...
        /length(Y_test);
    knn_A = ['Accuracy for using KNN after dimension reduction for ', ...
        'a threshold of ',num2str(th),' is ', num2str(knn_accuracy)];
    disp(knn_A)
end
```

7.2 Neural Network

```
directory = 'jaffe/*.tiff';
npixels = 65536; x = 256; y = 256;


imgFiles = dir(directory);
numFiles = length(imgFiles);
% X: n by p - n # of pixels in each image, p # of images
X = zeros(npixels, numFiles);
% Y: 1 by p - labels
Y =[];

% Load each face image as a column vector to form a data matrix X.
% Extract labels to form a label matrix Y.
for k = 1:numFiles
    img = imread(strcat(imgFiles(k).folder,'/',imgFiles(k).name));
    img_vector = reshape(img, npixels, 1);
    label = imgFiles(k).name(4:5);
    if label == "AN"
        label = 1;
    elseif label == "DI"
        label = 2;
    elseif label == "FE"
        label = 3;
    elseif label == "HA"
        label = 4;
    elseif label == "SA"
```

```
        label = 5;
    elseif label == "SU"
        label = 6;
    elseif label == "NE"
        label = 7;
    end
    X(:,k) = img_vector;
    Y = [Y;label];
end

% Separate training and testing set
P = 0.7;
idx = randperm(numFiles);
X_train = X(:,idx(1:round(P*numFiles)));
X_test = X(:,idx(round(P*numFiles)+1:end));
Y_train = Y(idx(1:round(P*numFiles)),:);
Y_test = Y(idx(round(P*numFiles)+1:end),:);

% Apply PCA to reduce data matrix size
[u,s,v] = svd(X_train,'econ');
e = diag(s)/sum(diag(s));

threshold = 0.9;
E = 0;
for i = 1:length(e)
    E = E + e(i);
    if (E > threshold)
        break
    end
end
recon_train = u(:,1:i)'*X_train;
recon_test = u(:,1:i)'*X_test;

% Neural Network setup
hidden_layer = [250 150 100];
func = 'tanh';
nnOptions = {'maxIter', 500, 'activationFn', func, ...
    'hiddenLayers', hidden_layer};
% Train a simple Neural Network, average 10 results
avg_accuracy = 0;
for i = 1:10
    modelNN = learnNN(recon_train', Y_train, nnOptions);
    % Predict test data with trained Neural Network
    nn_label = predictNN(recon_test', modelNN);
    wrong_predicts = find(Y_test - nn_label);
    nn_accuracy = (length(Y_test) - length(wrong_predicts))...
         /length(Y_test);
    avg_accuracy = avg_accuracy + nn_accuracy;
end
avg_accuracy = avg_accuracy/10;
nn_A = ['Accuracy for using Neural Network with activation function '...
        func, ' and hidden layer structure ', ...
        mat2str(hidden_layer), ' is ', num2str(avg_accuracy)];
disp(nn_A)
```

References

[1] Matthew N. Dailey et al. “Evidence and a Computational Explanation of Cultural Differences in Facial Expression Recognition”. In: vol. 10(6):874-93. Dec. 2010. DOI: 10.1037/a0020019.

[2] John Daugman. “Face and Gesture Recognition: Overview”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 19.7 (1997), pp. 675–676. DOI: 10.1109/34.598225. URL: https://doi.org/10.1109/34.598225.

[3] Italo José. *KNN (K-Nearest Neighbors)*. https://towardsdatascience.com/@italojs.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] Michael Lyons et al. “Coding Facial Expressions with Gabor Wavelets”. In: May 1998, pp. 200–205. ISBN: 0-8186-8344-9. DOI: 10.1109/AFGR.1998.670949.

[6] Vahe Tshitoyan. *Simple Neural Network*. https://www.github.com/vtshitoyan/simpleNN, GitHub.

[7] Andreas Zell. “Simulation Neuronaler Netze”. In: (Aug. 1994).