Handwritten Digits Recognition

ECE462 – Multimedia Systems | Project Report
University of Toronto

Gaurav Jain, Jason Ko
11/21/2008

Table of Contents

1 Introduction

The aim of this project is to implement a classification algorithm to recognize handwritten digits (0-9). It has been shown in pattern recognition that no single classifier performs the best for all pattern classification problems consistently. Hence, the scope of the project also included the elementary study the different classifiers and combination methods, and evaluate the caveats around their performance in this particular problem of handwritten digit recognition. This report presents our implementation of the Principal Component Analysis (PCA) combined with 1-Nearest Neighbor to recognize the numeral digits, and discusses the other different classification patterns. We were able to achieve an accuracy rate of 78.4%.

2 Background and Motivation

Hand writing recognition of characters has been around since the 1980s.The task of handwritten digit recognition, using a classifier, has great importance and use such as – online handwriting recognition on computer tablets, recognize zip codes on mail for postal mail sorting, processing bank check amounts, numeric entries in forms filled up by hand (for example - tax forms) and so on. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins. Our goal was to implement a pattern classification method to recognize the handwritten digits provided in the MINIST data set of images of hand written digits (0-9). The data set used for our application is composed of 300 training images and 300 testing images, and is a subset of the MNIST data set [1] (originally composed of 60,000 training images and 10,000 testing images). Each image is a 28 x 28 grayscale (0-255) labeled representation of an individual digit.

The general problem we predicted we would face in this digit classification problem was the similarity between the digits like 1 and 7, 5 and 6, 3 and 8, 9 and 8 etc. Also people write the same digit in many different ways - the digit '1' is written as '1', '1', '1' or '1'. Similarly 7 may be written as 7, 7, or 7. Finally the uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digits.

3 System Overview

Our approach to solve this problem of handwritten numeral recognition can be broadly divided into three blocks:

i) Pre-Processing/Digitization
ii) Feature Extraction using PCA
iii) Classification using 1-Nearest Neighbor algorithm

The block diagram for the system is shown below (Fig. 2):

Fig.1. System Diagram of the implemented pattern classifier

4 Pre-Processing

During our initial review of the various classifier methods, we undertook the approach of processing the training set images to reduce the data by thresholding the given image to a binary image. Fig.2 represents the sample images taken from the MNIST database [1].

Fig. 2 Sample digits used for training the classifier

We also looked at various processing methods such as edge-detection, and thinning of the digit image [2] to get a skeleton of the digit. This approach of acquiring the skeleton of the digit is widely used in the classifiers which mainly reply upon a well-defined input image for their accuracy. PCA is the holistic approach that extracts eigendigits based on the overall information contained in the image. So,

the information such as grayscale values and thickness of the digits actually assist in providing more information. Therefore, extensive pre-processing was not required in the implementation of the system. The training set was segmented into 10 groups – one for each digit, and then each group was individually passed into the PCA algorithm for the extraction of eigendigits.

5 Feature Extraction using Principal Component Analysis

Principal component analysis (PCA) is a fundamental multivariate data analysis method which is encountered into a variety of areas in neural networks, signal processing, and machine learning. It is an unsupervised method for reducing the dimensionality of the existing data set and extracting important information. PCA does not use any output information; the criterion to be maximized is the variance.

PCA can be applied to economically represent the input digit images by projecting them onto a low-dimensional space constituted by a small number of basis images [3]. These basis images or the "eigendigits" are derived by finding the most significant eigenvectors of the pixel wise covariance matrix, after mean centering the data for each attribute. After projection, we use the 1-NN classifier to classify the digit in the low dimensional space. PCA reduce the dimensions of the dataset from 784 to a lower value – for ease of computation.

5.1 PCA – The Algorithm

The algorithm used to implement PCA is described in detail below:

1) **Mean center the each training digit:** The empirical mean or the average digit is calculated for each group of digits ('0','1'…'9'). After that, it is subtracted from each training digit.

$$u[m] = \frac{1}{N}\sum_{n=1}^{N} X[m,n]$$

$$\mathbf{B} = \mathbf{X} - \mathbf{uh}$$

2) **Form the Covariance Matrix:** Find the empirical covariance matrix from the outer product of matrix **data set matrix** with itself.

$$\mathbf{C} = \mathbb{E}\left[\mathbf{B} \otimes \mathbf{B}\right] = \mathbb{E}\left[\mathbf{B} \cdot \mathbf{B}^*\right] = \frac{1}{N}\mathbf{B} \cdot \mathbf{B}^*$$

3) **Eigen decomposition:** Get the eigenvectors (columns of Vectors) and eigenvalues (diag of Values). Also normalize the eigenvalues to relate the values to the transpose of the covariance matrix. These basis vectors are also labeled as the eigendigits.

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D}$$

4) **Sort:** Sort the eigenvectors by eigenvalues, and select the 'k' most significant eigenvectors

5) **Projection:** Generate the projection map by projecting each digit onto the k-dimensional eigenvector space.

$$\mathbf{Y} = \mathbf{W}^* \cdot \mathbf{Z} = \mathbb{KLT}\{\mathbf{X}\}.$$

Below, we represent the sample k (=25, in this case) eigendigits generated by our PCA algorithm for the digits '0' and '9'. (Fig. 3)

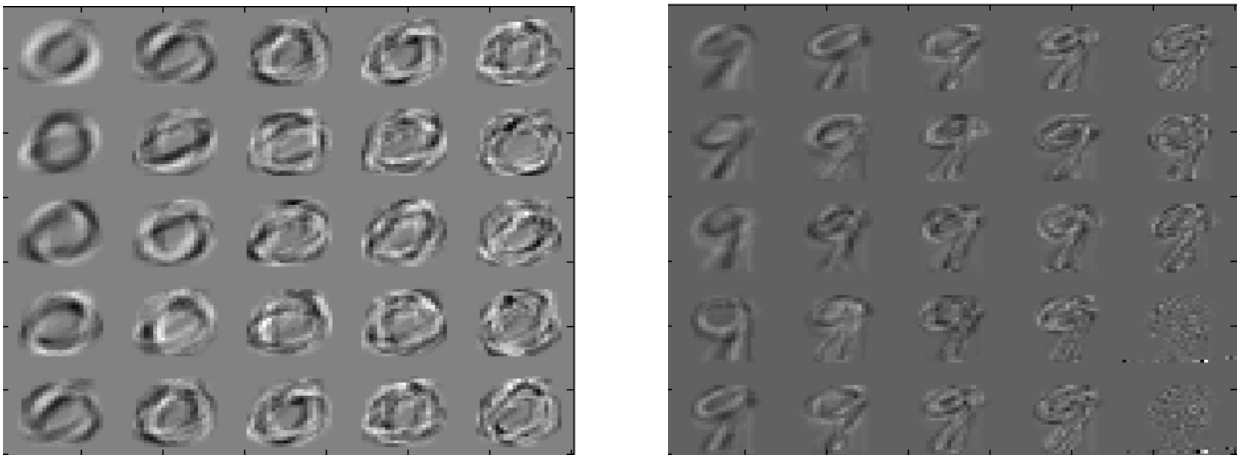


Fig.3. Sample eigendigits for the numerals 0 and 9, generated by the PCA classifier

6 K-nearest neighbor classifier

This classifier is one of the simplest classifier to implement because it is a brute force method [4]. The algorithm is used to find the nearest match for a given case to the known cases stored in memory. The K is used to signify how many votes are used for decision making. It is most optimal to choose a value of K that is odd so it eliminates a tie between two sets. There are three steps to implement for this classifier:

First step: compare known samples with the test sample using a distance metric. The distance metric is used to find the nearest neighbor. The most widely used distance metric is the Euclidean distance, because it gives a normalized value. Other distance metrics include City-block, and Chebychev, refer to (Appendix A, Fig. 1). The Euclidean distance is:

$$\vec{D} = \sum^{i} (known_i - testcase_i)^2$$

Second Step: Once we have the distances of the test subject to known subjects, we can rank them accordingly. If we are given 1000 known samples, we can rank distances of the results from 1 to 1000. The value K denotes the number of ranks to use. For example if K is equal to 99 then the top 99 distance vector value is considered.

Third Step: Considering a case where it is either true or false, within the 99 results the one with the greatest number is the value of the test case. Then if there were 50 results that point to true and 49 results point to false, then the test sample is true. This is why an odd value is chosen so there would be no ties. It is important to note that there could be more than 2 cases, so a tie could result even with odd value of K, for example if there were three sample spaces. A triangle, a circle and a square object, there could be 40 votes for circle and square out of 99 which would result in a tie. For this case it would be best to consider their ranking number as well.

For our project we chose K to be 1 so we have implemented a 1-nearest neighbor classifier. We retrieve the results for 20 samples of each digit so there are a total of 120 votes to consider. We choose the minimal value of the 20 samples of each digit, so in the end we have 10 values to compare for the final result, which is the minimal distance vector for the digits 0 to 9. By choosing the minimum value out of the ten we effectively chosen the most significant vote for what this sample value is.

7 Alternative Designs

From the literature we reviewed we found that the highest accuracies achieved by classifiers on the MNIST database were 99.05%, by individual classifiers and multiple classifier systems, and 99.3% (boosted), by multiple neural networks. Below listed are the main schemes that we reviewed, and their advantages and disadvantages, and the caveats around their implementation.

7.1 Neural Networks

Multilayer Neural Networks trained with the back-propagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Convolutional Neural Networks are a special kind of multi-layer neural networks used to recognize visual patterns with extreme variability (such as handwritten digits), and with robustness to distortions and simple geometric transformations. LeNet-5 is the latest convolutional network [4], and below is the architecture diagram (Fig. 4). LeNet-5 comprises 7 layers, all of which contain weights as trainable parameters. The values of the input pixels are normalized to make the mean input roughly 0, and the variance roughly 1, which accelerates learning. Layer C1, C3 and C5 are convolutional layers with 6, 16 and 120 feature maps respectively. Each unit in each feature maps are successively connected to the neighboring 5x5 feature maps, assisting training by localization. The layers S2 and S4 are subsampling layers with 6 feature maps of size 14x14 and 16 feature maps of size 28x28 respectively.

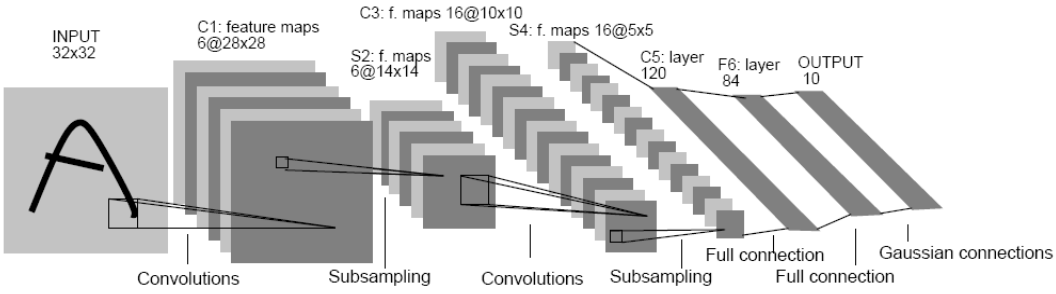


Fig.4 Architecture of LeNet-5, a convolutional neural network, for digits recognition

Given the high accuracy achieved by this technique, this was our initial proposed algorithm for this project. However, the implementation of the algorithm is highly complex. Convolutional neural networks are particularly well suited to hardware implementations because of their regular structure and low memory requirements for the weights. They also perform more efficiently and are well suited to the extensively large database of digits. However, from the reviews it has been found that the training time for these networks are expectedly longer, amounting to 2 to 3 days of CPU time in some cases for 10 to 20 passes through the training set. Given the limited time scope of the project, and major emphasis being on being able to review the literature and implement a basic algorithm, we decided to proceed with the PCA implementation instead.

7.2 Lottery Digit Recognition

Lottery Digit Recognition Based Multi-feature [5], is an alternate implementation of digit recognition compared to neural network and support vector machine methods. One of the chief advantages of the system is that it doesn't require any learning for the system as compared to neural network recognition system. Their implementation consists of the same steps as our implementations, pre-process, feature-extraction and classification (multi-features). The main difference is their feature-extraction and multi-features classification. The features they extract are the end points of the digits, and depending on the distances of two endpoints they could be connected or deleted. An example of this system is to consider the circle of nine that is unclosed from the image, depending on how far the two endpoints are they could be connected to create a closed loop for better classification. Using the number of endpoints they can classify into smaller sets of digits to compare with eventually ending into a result. The problem with this implementation is the number of algorithms we need to implement; therefore we did not choose this solution.

7.3 Stroke based Characterization

User-Independent Online Handwritten Digit Recognition [6], this system uses a local based recognition system. It considers the stroke of the digit and characterizes them for the classifier, the different strokes are assigned a number therefore each digit can be encoded by the starting stroke to ending stroke, such as a 2 is encoded as "78123451" [b]. (Fig. 5)

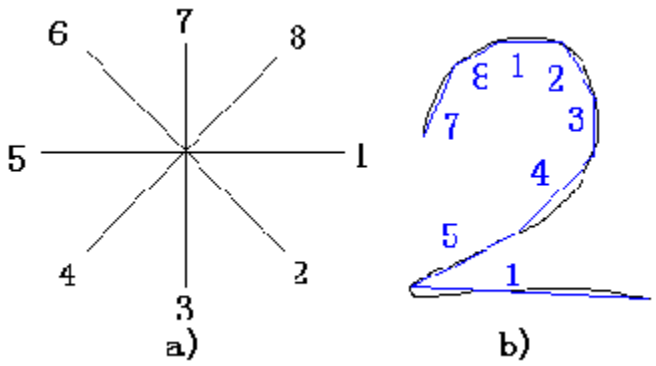


Fig. 5 Directional Stroke algorithm feature set

With this feature code, it can be used in simple classifiers. This implementation is good because it doesn't require a training system but the main disadvantage is the problem of classifying the strokes. There are problems with clockwise and counter-clockwise digits. Our solutions is simpler implementation because it uses a global approach instead of local approach which still gives accurate recognitions without the amount of work required for localized approach.

8 Test Results

From the test images given to us, we are able to obtain a maximum accuracy of 78.4%. This is varied by the number of eigenvectors values we consider in comparison of the test sample. To achieve this result our program considers all 784 eigenvector values in order to get an accuracy of 78.4%; where as, if we only consider 12 most significant eigenvector values we only get 40.2% accuracy.

As the number of eigenvector values considered are decreased the accuracy of the system decreases. The diagram below shows the results of the accuracy based of the number of eigenvector values. (Fig. 6)

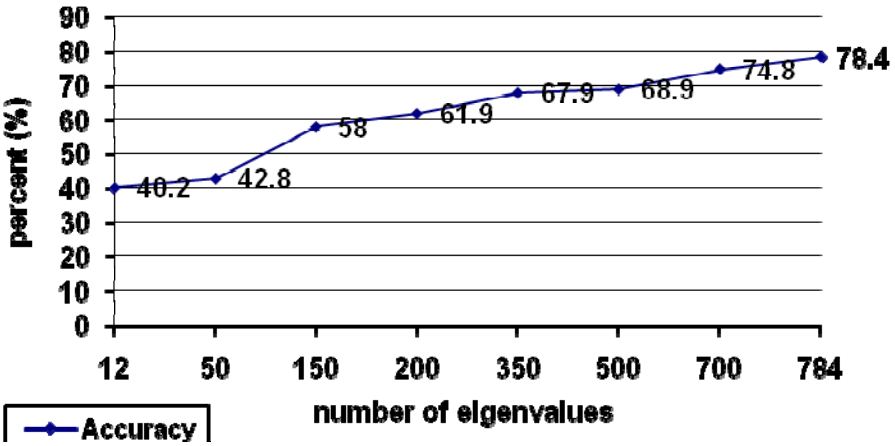
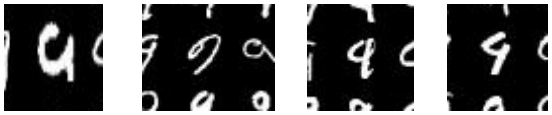
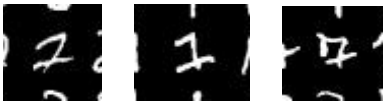


Fig. 6 Accuracy achieved plotted against the number of eigenvalues used

There are several reasons for the 21.6% inaccuracy: The main reason for the system inaccuracy is because the number of training sample for a digit was not enough. From the results of other documentation of the same type of project [7], we find that when they use more training sample the more accurate they can get. But because the scope and time frame of this project is small therefore we limited the amount samples to test and compare. Another limiting factor is the basic setup of this project, we are simply using the most basic algorithms for feature extraction and classification therefore it is hard to have good results compared to a more elaborate designed system. A factor of the system that changes the accuracy are the number of eigenvector values we consider, increases the number of significant values we increase the accuracy in a linear fashion. This make sense because the values from the eigenvectors represents the feature of the digit, the more we use to compare to our test case the more information we can get from the distance metrics. There are other reasons for inaccuracies such as poorly written digits, for example:



These are all examples of the digit 9. For the last one it is actually hard for a person to figure out it's a 9 instead of a 4. Other convoluted examples are:



The actual digits are in the following order 2, 1, 7. Given their similar features visually, it would be difficult to have a mathematical expression for them that is perfectly unique to each digit they belong to. The increase of samples in a training algorithm will definitely help in the accuracy of the results but in some cases where the digit looks like something else it would be hard to find a good mathematical expression for them to be compared by a computer without errors.

9 REFERENCES

[1] MNIST Database of Handwritten digits: http://yann.lecun.com/exdb/mnist/

[2] Claude Chouinard, Rejean Plamondon “Thinning and Segmenting Handwritten Characters by Line Following”, Ecole Polytechnique de Montreal, Machine Vision and Applications, 1992

[3] M. Turk and A. Pentland (1991) ``Eigenfaces for recognition'', Journal of Cognitive Neuroscience, 3(1)

[4] By Kardi Teknomo, PhD. “K-Nearest Neighbors Tutorial”
http://people.revoledu.com/kardi/tutorial/KNN/index.html

[5] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner “Gradient-Based Learning Applied to Document Recognition”, IEEE, November 1998

[6] Decong Yu, Lihong Ma and Hanqing Lu “Lottery Digit Recognition Based Multi-feature”

[7] Wen-Li Jiang, Zheng-Xing Sun, Bo Yuan, Wen-Tao Zheng and Wen-Hui Xu “User-Independent Online Handwritten Digit Recognition”, Nanjing University, 13-16 August 2006

[8] Geoffrey F Hinton, Michael Revow and Peter Dayan “Recongnizing Handwritten Digits Using Mixtures of Linear Models”, http://www.cs.utoronto.ca/~hinton/absps/pancake.pdf

APPENDIX A

Other distance measures used to find the nearest neighbor:

City-block: $D(x,p) = Abs (x – p)$

Chebychev: $D(x,p) = Max (| x – p |)$

Fig. 1

APPENDIX B – SOURCE CODE

[PCA CODE – pcaimg.m]

```
function [base,mean,projX] = pcaimg(X,k)

% pca analysis of image data
% input: X: vectorized image data
%        k: number of eigenvectors you want to keep
% output: base: eigenvectors
%         mean: mean of data
%         projX: the projected data in the low-dimensional space.

disp('eigendecomposition...');

[xdim,ndata] = size(X);

mean = sum(X,2)/ndata;  % compute mean of data

X = X-repmat(mean,1,ndata); % substract the mean

cov = X*X'/ndata;  % form the covariance matrix

[ev,ed]=eig(cov);  %eigendecomposition

ed = diag(ed);

% Sort eigenvectors by eigenvalues
[foo,p]=sort(-ed);

ed = ed(p);

ev = ev(:,p);

% Take the top k eigenvectors
base = ev(:,1:k);

% project the data into low-dim space
projX = base'*X;
```

[Nearest Neighbor Classifier – NNclassifier.m]

```
function digit = NNclassifier (TestImage, Vecs, Mean, Projec0, Projec1
,Projec2,Projec3,Projec4,Projec5,Projec6,Projec7,Projec8,Projec9)
% This classifier, uses a given test image from the test space
% we need to project the test sample into digit's 1,2,3,4,5,6,7,8,9,0 space
% individually thant we get the euclidean distance
%
% finally find the minimal distance to specify the digit for the test case
%
    testsample = TestImage(:,1);
    test = testsample - Mean(:,1);
    ProjT0 = Vecs(:,:,1)'*test;
    test = testsample - Mean(:,2);
    ProjT1 = Vecs(:,:,2)'*test;
    test = testsample - Mean(:,3);
    ProjT2 = Vecs(:,:,3)'*test;
    test = testsample - Mean(:,4);
    ProjT3 = Vecs(:,:,4)'*test;
    test = testsample - Mean(:,5);
    ProjT4 = Vecs(:,:,5)'*test;
    test = testsample - Mean(:,6);
    ProjT5 = Vecs(:,:,6)'*test;
    test = testsample - Mean(:,7);
    ProjT6 = Vecs(:,:,7)'*test;
    test = testsample - Mean(:,8);
    ProjT7 = Vecs(:,:,8)'*test;
    test = testsample - Mean(:,9);
    ProjT8 = Vecs(:,:,9)'*test;
    test = testsample - Mean(:,10);
    ProjT9 = Vecs(:,:,10)'*test;
    for i=1:20
        a0(i) = norm(ProjT0 - Projec0(:,i));
        a1(i) = norm(ProjT1 - Projec1(:,i));
        a2(i) = norm(ProjT2 - Projec2(:,i));
        a3(i) = norm(ProjT3 - Projec3(:,i));
        a4(i) = norm(ProjT4 - Projec4(:,i));
        a5(i) = norm(ProjT5 - Projec5(:,i));
        a6(i) = norm(ProjT6 - Projec6(:,i));
        a7(i) = norm(ProjT7 - Projec7(:,i));
        a8(i) = norm(ProjT8 - Projec8(:,i));
        a9(i) = norm(ProjT9 - Projec9(:,i));
    end
    minv(1) = min(a0);
    minv(2) = min(a1);
    minv(3) = min(a2);
    minv(4) = min(a3);
    minv(5) = min(a4);
    minv(6) = min(a5);
    minv(7) = min(a6);
    minv(8) = min(a7);
    minv(9) = min(a8);
    minv(10) = min(a9);
    mm = min(minv);
    digit = find (minv==mm) - 1;
```

[Main Script – rec.m]

```
% every image is 28x28 in gray scale (0-255)
% train_images is 300x784 which means that it has 300 training images
% and every image has been comverted in to a vector (1x784) for convenience

clear all
nPrinc = 12;
load train_images.txt;

% Gathering all the zero images
% Open the training images label file
  labelfile = 'train_labels.txt';
  fid = fopen(labelfile,'r');
  if fid < 0
    error(['Cannot get list of images from file "' labelfile, '"']);
  end;

  ImgZero = [];  ImgOne = [];  ImgTwo = [];  ImgThree = [];  ImgFour = [];
  ImgFive = [];  ImgSix = [];  ImgSeven = [];  ImgEight = [];  ImgNine = [];
  Imgs = [];
  ImgCount = [0 0 0 0 0 0 0 0 0 0];

  i=1;
  % Pre-process the images
  while 1
    imgname = fgetl(fid);
    if ~isstr(imgname)            % EOF is not a string
      break                      % Exit from loop on EOF
    end
    imgnum = str2num(imgname);
    A=vec2mat(train_images(i,:),28);
    ImgCount(imgnum+1) = ImgCount(imgnum+1) + 1;

    if (imgnum == 0)
        ImgZero(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 1)
        ImgOne(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 2)
        ImgTwo(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 3)
        ImgThree(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 4)
        ImgFour(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 5)
        ImgFive(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 6)
        ImgSix(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 7)
        ImgSeven(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 8)
        ImgEight(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
    elseif (imgnum == 9)
        ImgNine(:,ImgCount(imgnum+1)) = (reshape(A',1,784)');
```

```
    end
    i = i+1;
  end

[Vecs(:,:,1),Mean(:,1),Projec0] = pcaimg(ImgZero,nPrinc);
[Vecs(:,:,2),Mean(:,2),Projec1] = pcaimg(ImgOne,nPrinc);
[Vecs(:,:,3),Mean(:,3),Projec2] = pcaimg(ImgTwo,nPrinc);
[Vecs(:,:,4),Mean(:,4),Projec3] = pcaimg(ImgThree,nPrinc);
[Vecs(:,:,5),Mean(:,5),Projec4] = pcaimg(ImgFour,nPrinc);
[Vecs(:,:,6),Mean(:,6),Projec5] = pcaimg(ImgFive,nPrinc);
[Vecs(:,:,7),Mean(:,7),Projec6] = pcaimg(ImgSix,nPrinc);
[Vecs(:,:,8),Mean(:,8),Projec7] = pcaimg(ImgSeven,nPrinc);
[Vecs(:,:,9),Mean(:,9),Projec8] = pcaimg(ImgEight,nPrinc);
[Vecs(:,:,10),Mean(:,10),Projec9] = pcaimg(ImgNine,nPrinc);

accuracy = 0;
load test_images.txt;
load test_labels.txt;
for check = 1:1000

TestImage=(test_images(check,:))';
Testvalue=test_labels(check);

value =
NNclassifier(TestImage,Vecs,Mean,Projec0,Projec1,Projec2,Projec3,Projec4,Proj
ec5,Projec6,Projec7,Projec8,Projec9);

if Testvalue == value %|| Testvalue == (value + 1) || Testvalue == (value -
1)
    accuracy = accuracy + 1;
  end
end

testResult = accuracy/1000 * 100
```