**Off-line Signature Verification System Design Using Different Types of Neural Network and their Performance Analysis**

**CSE-448 Project Paper**

**MD.Rasheduzzaman**

**ID: 0910647044**

**(rashed091nsu@gmail.com)**

**Index**

1. **Introduction --------------------------------------------------------------------------------------3**
2. **Related Work -------------------------------------------------------------------------------------3**
3. **Problem Definition and Algorithm ---------------------------------------------------------4**
4. **Signature data set -------------------------------------------------------------------------------4**
5. **Per-processing stage----------------------------------------------------------------------------5**
6. **Feature Extraction ------------------------------------------------------------------------------5**
7. **Neural Network design -----------------------------------------------------------------------6**

**7.1. Perceptron Neural Network----------------------------------------------------------7**

**7.2. Back-propagation -----------------------------------------------------------------------9**

**7.3. Probabilistic Neural Network -------------------------------------------------------12**

1. **Time Vs accuracy -------------------------------------------------------------------------------14**
2. **Conclusion and Discussion ------------------------------------------------------------------14**
3. **References --------------------------------------------------------------------------------------14**

**11. Source Codes -----------------------------------------------------------------------------------15**

**Abstract**

In this paper we talked about off-line signature verification using different types of Neural Network and analysis their performance. Our network uses the global feature of a signature image to train itself. The networks that we used hare are Perceptron, Back-Propagation, and Probabilistic neural network. The final accuracy of Perceptron is 91.17%, Back-propagation is 71.83% and Probabilistic neural network is 71.00%.

**Keyword:** signature, binary image, global feature, neural network, Perceptron, Back-Propagation (BP), and Probabilistic Neural Network (PNN).

**1. Introduction**

A signature is a handwritten depiction of someone name or nickname that a person writes on documents as a proof of identity. For centuries, handwritten signature is used to validate the signer. Signature differs from person to person. Which gives it the distinctiveness that helps to identify the signer, but a person does not sign same way every time. Because of it signature can be easily forged. In order to sop the forging of we want design a signature recognition system that will identify the forge or original signature. There are two type of recognition system. 1. Online recognition and 2.Off-line recognition. Online recognition system three-dimension environment where, not only is the height and width of pen strokes measured, but also the amount of pressure applied in the pen stroke to measure the depth that would occur as if the stroke was made in the air. This helps to reduce the risk of forgery that can occur in two-dimensional signatures. One drawback to this form is that people do not always sign documents in exactly the same manner. The angle at which they sign may be different due to seating position or due to hand placement on the writing surface. Therefore, even though it is three-dimensional which adds to its ability to discern impostors, it is not as accurate as other forms of biometric verification. In Offline recognition case the signature appears as a 2D (gray level or binary) image. The static signature verification is considered to be much more difficult because timing and dynamic information are highly degraded in that case. The off-line method uses an optical scanner to obtain the handwriting data written on paper. In this mechanism the user signs on a piece of paper, which is read by a scanner or a camera. The image is then fed to a computer. The computer stores the image as specific to the signer. It is used to identify the user by the image. In our project we try to create an offline recognition system using different types of neural network.

**2. Related Work**

Hundreds of work has been done in the field of off-line signature verification system. Here we describe few neural network techniques which achieved very good accuracy on both training and testing data.

1. Mohammed A.Abdala and Noor Ayad Yousif [1] developed a system that uses two stage neural network classifiers. In the first stage classifier consists of three Back-Propagation neural networks and the second stage classifier consists of two Radial Basis function Neural Network. Three different types of feature extraction method are used. The global feature, grid information feature, and texture feature. In the first stage these three different features goes as an input to three Back Propagation neural networks NN1, NN2, and NN3. NN1 has 8 inputs, two outputs. NN2 has 48 input s and two outputs, and NN3 has 96 inputs and two outputs. In the second stage two radial basis functions (RBF) neural network takes outputs of the NN1, NN2 and NN3 as input and gives two outputs. RBF1 indicates (0 or 1) if the signature is recognized. RBF2 gives the data base number for the signature. For training 205 signatures are collected from 41 people (5 signatures from each person). From these 164 is used as a training data. In the testing phase 148 signatures is used to test the system. 66 Out of 148 signatures are forged signatures. The trained signature has 100% accuracy and tested signature has 95.955% accuracy.

2. Stephane A., Michael B., and Valipuram M. [3] used two Neural Network-based classifiers, a Resilient Back Propagation (RBP) neural network and a Radial Basis Function (RBF) neural network. The modified direction feature (MDF) and other techniques is used to extract the feature from the signature image. MDF uses two other feature extraction techniques, Direction Feature (DF) and the Transition Feature (TF). The 2106 signature is collected from publicly available database. From this 39 set is used, where 24 samples is genuine and 30 samples is forgeries. In the training phase 18 samples of genuine and 22 samples of forged signature is used for training. In the testing phase 6 samples of genuine and 8 samples of forged signature is used for testing. Four-fold cross validation methods is used is this system. For verification 1560 signature is used for turning and 546 are used for testing. Using RBP, MDF reached 86.08% accuracy and The RBF the system reached 91.12% accuracy. The RBF classifier is better than the RBP classifier.

3. H. Baltzakis and N. Papamarkos [2] developed a new techniques called two stage Perception OCON (one-class-one-network) classifier. The system uses global, grid and texture feature for each of this classifier. The output of the first stage classifier combine with the Euclidean distance of these features is feed a second-stage radial base faction, which gives the final decision. About 2000 signature is collected from SRVS database. The signature were taken from 115 persons (each person has 15-25 signature). For training the system, the data set is divided into two subsets of about 1000 and 500 signature. The first subset (TRS1) was used to train the first-stage classifier and (TSR2) was used to train second-stage classifier. Testing was done by remaining subset of (TS) of 500 signatures. To avoid local minima the ALOPEX algorithm (Pandya and Macy, 1995) was chosen for this task. The first-stage classification uses three Multi-layer Perceptron (MLPs) network and a Euclidean distance function. One for each group of features. The output of the three MLPs and the average Euclidean distance are taken as the input of the second-stage classifier. If the output of second-stage classifier is positive then the given signature belongs to a candidate person. If not, then it does not belong to a candidate person. The testing of system conducted with (TS) contained 500 signatures. The system achieve 404 correct classification (80.81%) and 96 false classification (19.19%).

**3. Problem definition and Algorithm:**

The main objective of this paper is to develop an off-line signature verification system that can identify original and forged signature. Here we used three different neural network system and analysis performance on signature verification. Our first neural net work is Single layer Perceptron, second is Back Propagation and final network is probabilistic neural network.

**4. Signature data Set:**

Sample signatures were collected from 30 different people. Each person has 100 signatures. So our total data set has 3000 thousand image.

Figure: Sample signature images

**5. Pre-possessing stage:**

The images may contain some noise or unnecessary information. This is not very helpful for our project. The prepossessing stage deals with this problem. The prepossessing stage consist of three filters: converting gray scale image into binary image, morphological filtering and size normalize.

**5.1. Convert Image to Binary Image:**

In first step of image prepossessing we convert the gray scale image into a binary image. First we calculate the gray threshold. Using the threshold we convert the image black (0) and white (1).

**5.2. Morphological filtering:**

The goal of image enhancement is to increase the visibility and perceptibility of an image. We apply the morphological filter which enhance the image and reduces the noise.

**5.3. Size Normalization:**

The original signature image may contain lot of white space, which is not very helpful as a feature. So we normalize the size of the image and extract only part of the image which has the signature.

**6. Feature Extraction:**

The dimension of our input image is. Global feature provides specific and useful information about images. This can be used as a feature. Using global feature of the input image we calculate the feature vector which is faded to our neural network. The length of the feature vector is 225.

**(a).Image Area:**

The area of binary (or gray) image is calculated by summing up all the pixel value. For our project we normalized the total area and used it as a feature.

**(b).Center of Gravity:**

The center of gravity of an image is a point from where all the all other points are symmetrically arranged.

**(c).Vertical slice:**

If slice the in different places regular intervals vertically and count the number of transition we have from black to white.

**(d).Horizontal slice:**

If slice the in different places regular intervals horizontally and count the number of transition we have from black to white.

**(e).Gradient Histogram:**

we divide our image into small 3 by 3 size window and check is the top 3 and rightmost indexes have 1 in them or not. If yes then we count them and save them as a vector.

**(f).Density Count:**

We divide the image into 4 regular windows and find the number of 1’s in each window and divide it with the total number of places in the windows.

**(g).Row Sum:**

The sum of pixels in each row of an image is called the row sum or horizontal projection.

**(h).Column Sum:**

The sum of pixels in each column of an image is called the column sum or vertical projection.

**7. Neural Network Design:**

Network used in our analysis is Perceptron, Back-propagation, and Probabilistic Neural Network. We design our network with simplest manner. For Perceptron network we used simple single layer Perceptron network. For Back-propagation we used Resilient Back-propagation (BP) algorithm, because it consume less memory when we conducted our simulation on PC. And last we used Probabilistic neural network.

**7.1. Perceptron:**

The Perceptron is binary classifier which maps a real value vector input x into a binary decision (0 or 1). The Perceptron architecture

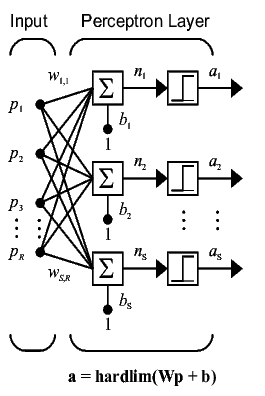


Figure: Simple multi-neuron single layer Perceptron network

used in our project has 225 input and the number of neuron is decided by the program based on target automatically.

**7.1.1. Algorithm:**

The procedure is as follows: During training an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections. The update rule apply this way, let out target be denoted by‘t’ and our network output is ‘a’. Then



1. If t=1 but a=0 then, 

2. If t=0 but a=1 then, 

3. If t=1 and a=1 then, 

**7.1.2. Training phase:**

In this phase, the 225 feature that we calculated in the feature extraction stage is fed to our Perceptron network. For training we divided our total data set into three sets. Our first set (Set1) contains 2400 sample images (80% of data), second set (Set 2) contains 1950 sample images (65% of data), and our last (Set 3) contains 1500 sample images (50 % of data). The purpose of dividing the data into three separate groups is to observe that what the effect of it on testing data is. Before going to testing phase the train network is tested on itself and the result is,

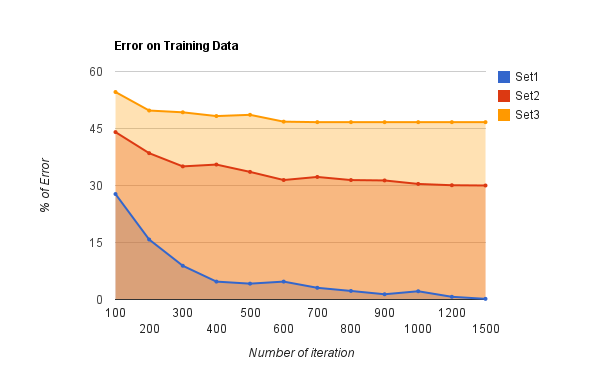


Figure: Graph of training phase

**7.1.3. Testing phase:**

After completing the training of Perceptron network on different data set, the network is then tested on single data which contains 600 sample images (20% of data). These 600 images are complete unknown to network. The testing result is



Figure: Graph of testing phase

**7.1.4.Result:**  
After training with three data set and testing with one. The best result we got is 99.83% accuracy on training data and 91.17% testing accuracy from the network which is train with data Set1. The table blow shows few testing result of different iteration. By analyzing the testing result we can that system train with more data has better accuracy over the system that was trained with less data.

|  |  |  |  |
| --- | --- | --- | --- |
| iteration | Testing accuracy of Set1 | Testing accuracy of Set2 | Testing accuracy of Set3 |
| 100 | 69.00% | 47.17% | 37.33% |
| 300 | 85.00% | 54.83% | 41.67% |
| 600 | 87.33% | 56.83% | 42.17% |
| 1000 | 89.00% | 57.67% | 42.33% |
| 1200 | 90.67% | 58.3% | 42.33% |
| 1500 | 91.17% | 58.17% | 42.33% |

**7.2. Back-Propagation:**

Back-propagation is a form of supervised training. When using a supervised training method, the network must be provided with both sample inputs and anticipated outputs. The anticipated outputs are compared against the actual outputs for given input. Using the anticipated outputs, the back-propagation training algorithm then takes a calculated error and adjusts the weights of the various layers backwards from the output layer to the input layer.

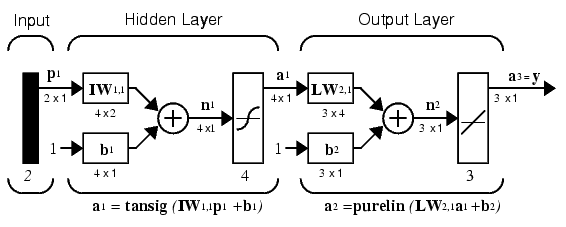


Figure: Simple back-propagation network with tansig and purelin as transfer function

Back-propagation network usually constructed with either tansig or logsig as a transfer function in the hidden layer and a purelin transfer function in the output layer.

**7.2.1. Algorithm:**

Consider a network with real input x and a transfer function. The derivatives is calculated in two phase,

Feed-forward: In feed-forward layer we fed input to our network. After few process networks produce output. Which is then comparing with our actual target? If the output and target matches then we move on to the next input. If not, then we calculate the error.

Back-propagation: In this layer we calculate the derivatives of the transfer function. Then by using the error we calculated in feed-forward layer and the derivatives we calculate the sensitivity of output and hidden layer. Using this sensitivity we then update the weight and bias of our network. We continue this as long as we don’t get the right output.

**7.2.2. Training phase:**

In this phase, the 225 feature that we calculated in the feature extraction stage is fed to our BP network. To keep our analysis simple we have only one set training data and one set of testing data. Our training data set contains 2400 sample images (80% of data), and our testing data set contains 600 sample images (20% of data). The BP network we are using here has only one hidden layer. First we set our hidden layer transfer function logsig and then tansig. The number of neuron in hidden layer vary from 1 to 100.The reason is to observe that which transfer function and what number of neuron gives us the best possible result. Before going to testing phase the train network is tested on itself and the result is,

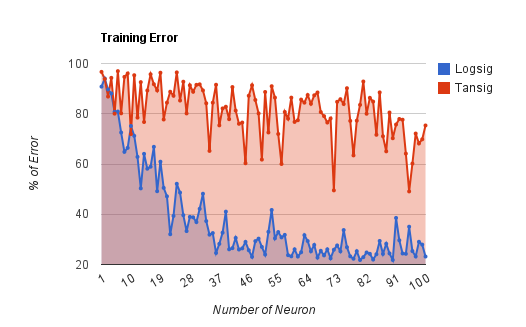


Figure: Graph of training phase

**7.2.3. Testing phase:**

After completing the training of BP network on different transfer function, the network is then tested on single data which contains 600 sample images (20% of data). These 600 images are complete unknown to network. The testing result is

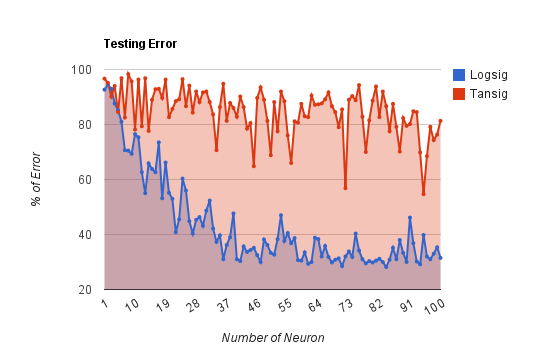


Figure: Graph of testing phase

**7.2.4. Result:**

After training, we tested out network with one data set. The best result we got is 78.44% accuracy on training data and 71.83% testing accuracy from the network which is train with data Set1. The table blow shows few testing result with different number of neuron. By looking at the table we say that logsig transfer function gives us better result than the tansig transfer function.

|  |  |  |
| --- | --- | --- |
| Number of Neuron | Testing with Logsig | Testing with Tansig |
| 1 | 7.44% | 3.44% |
| 20 | 37.44% | 17.44% |
| 50 | 67.50% | 31.17% |
| 65 | 70.83% | 11.44% |
| 72 | 70.17% | 33.17% |
| 88 | 71.83% | 29.83% |

**7.3. Probabilistic Neural Network:**

A probabilistic neural network (PNN) is used a classifier. It maps input pattern to a number of classification. A PNN is implementation of an statistical algorithm called kernel discriminant analysis.

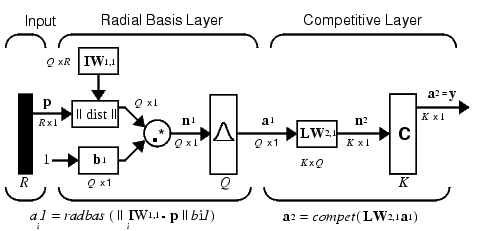


Figure: Simple PNN architecture.

**7.3.1. Algorithm:**

The first layer computes distances from the input vector to the training input vectors and produces a vector whose elements indicate how close the input is to a training input. The second layer sums these contributions for each class of inputs to produce as its net output a vector of probabilities. Finally, a *compete* transfer function on the output of the second layer picks the maximum of these probabilities, and produces a 1 for that class and a 0 for the other classes.

**7.3.2. Training phase:**

In this phase, the 225 feature that we calculated in the feature extraction stage is fed to our PNN network. Our training data set contains 2400 sample images (80% of data). For PNN training we used different spread value and tried to find which spread value gives us the best result. Before going to testing phase the train network is tested on itself and the result is,

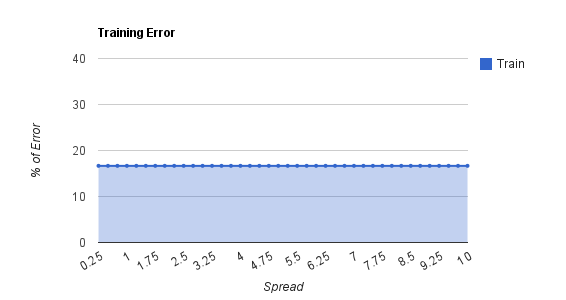


Figure: Graph of training phase

**7.3.3. Testing phase:**

After completing the training of BP network on different transfer function, the network is then tested on single data which contains 600 sample images (20% of data). These 600 images are complete unknown to network. The testing result is

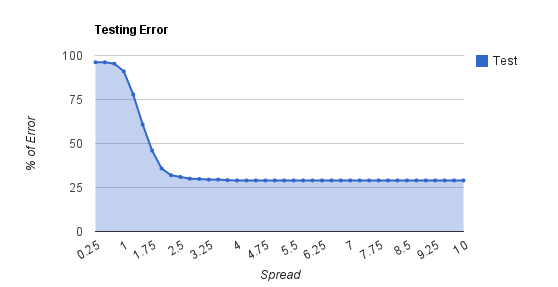


Figure: graph of testing phase

**7.3.4. Result:**

After training we tested our network with one test data set. From the graph of training phase we see that even if we increase the spread value it shows no improvement and on the other hand in the testing phase after spread value 4 the network does not shows any improvement on accuracy. The best result we got is 83.60% accuracy on training data and 71.00% on testing data. The table blow shows few testing result of different iteration.

|  |  |
| --- | --- |
| Spread Value | Test Accuracy |
| 0.25 | 3.83% |
| 1.00 | 9.00% |
| 2.00 | 64.17% |
| 3.75 | 70.83% |
| 4.00 | 71.00% |

**8. Time and Accuracy:**

Based on the result we can say that Perceptron gives us better result than the other two networks Back-propagation and Probabilistic neural network. Now if look at this time (minute) vs. accuracy table we will find that though Perceptron gives us better result, but it takes a lot of time to converge to a good accuracy. In the other hand Back-propagation and Probabilities achieve around 70.00% of accuracy in less than a minute. If modify our BP and PNN architecture then we can probably get 90.00% of accuracy out these two network also.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (minute) | PNN | Time (minute) | Perceptron | Time (minute) | BP |
| 0.626571 | 3.83% | 3.995799 | 69.00% | 0.041785 | 5.50% |
| 0.611714 | 3.83% | 7.777996 | 77.17% | 0.462752 | 14.83% |
| 0.622956 | 4.67% | 11.61151 | 85.00% | 0.106656 | 19.00% |
| 0.622956 | 9.00% | 11.61151 | 87.33% | 0.342233 | 37.33% |
| 0.612706 | 22.17% | 18.87924 | 88.00% | 0.205553 | 45.00% |
| 0.612677 | 39.17% | 22.711 | 87.33% | 0.172378 | 47.00% |
| 0.62293 | 54.00% | 26.56489 | 88.00% | 0.162654 | 51.33% |
| 0.615258 | 64.17% | 32.82911 | 89.17% | 0.160887 | 57.83% |
| 0.615335 | 68.00% | 40.87507 | 89.83% | 0.33222 | 60.33% |
| 0.616335 | 69.00% | 49.59182 | 89.00% | 0.223043 | 64.33% |
| 0.621419 | 70.17% | 57.48658 | 90.67% | 0.615565 | 66.33% |
| 0.621134 | 71.00% | 65.76476 | 91.17% | 0.484591 | 71.83% |

Figure: Time (minute) Vs Accuracy

**9. Conclusion and Decision:**

After completing the analysis of different network the result we got is quiet convincing, because we used the simplest form of our network with simple global feature as our input. It easy can be said that that using more sophisticated network architecture and more advance feature extraction method we can easily increase the performance of our off-line handwritten signature verification system.

**10. Reference:**

1**.** Mohammed A. Abdala and Noor Ayad Yousif, Offline Signature Recognition and Verification Based on Artificial Neural Network.

2. H. Baltzakisa, N. Papamarkos, A new signature verification technique based on a two-stage neural network classifier.

3.Stephane Armand, Michael Blumenstein and Vallipuram Muthukkumarasamy, Off-line Signature Verification based on the Modified Direction Feature.

**11. Code:**

All the code used in these project are given below.

11.1. Feature extraction:

|  |
| --- |
| Image Area |
| function A=areaI(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %img=im2double(img);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  A=bwarea(img);  A=sqrt(A);  End |

|  |
| --- |
| Center of Gravity |
| function [cgi,cgj]=CG(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %img=im2double(img);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  [r,c] = size(img);  sum1=0;  sum2=0;  for i=1:r,  for j=1:c,  sum1 = sum1 + j\*img(i,j);  end  end  for i=1:r,  for j=1:c,  sum2 = sum2 + i\*img(i,j);  end  end  A=areaI(img);  cgi=sqrt(sum1/A);  cgj=sqrt(sum2/A);  end |

|  |
| --- |
| Row sum |
| function [ result ] = rowSum(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  result=sum(~img);  end |

|  |
| --- |
| Column sum |
| function [ result ] = colSum(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %I=im2double(I);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  result=sum(~img');  end |

|  |
| --- |
| Gradient Histogram |
| function [array] = gradientHistogram(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %img=im2double(img);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  A = ~img;  [r,c]=size(A);  r1 = floor(r/4);  c1 = floor(c/4);  array = zeros(4,4\*4);  cnt =1;  u1=0; u2=0; u3=0; u4=0;  for i=1:r1:r  for j=1:c1:c  if((i+r1-1)>r || (j+c1-1)>c)  break;  end  b = A(i:i+r1-1,j:j+c1-1);  for k=2:size(b,1)-1  for q=2:size(b,2)-1  if(b(k,q)==1)  if(b(k,q+1)==1)  u1 = u1 + 1;  array(1,cnt) = u1;  elseif(b(k-1,q+1)==1)  u2 = u2 + 1;  array(2,cnt) = u2;  elseif(b(k-1,q)==1)  u3 = u3 + 1;  array(3,cnt) = u3;  elseif(b(k-1,q-1)==1)  u4 = u4 + 1;  array(4,cnt) = u4;  end  end  end  end  cnt = cnt + 1;  end  end  array = array(:)';  end |

|  |
| --- |
| Dot Density |
| function [sample] = dotcnt(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %img=im2double(img);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  I=img;  dim=[11 32];  T = zeros(8, 8);  for k = 1:8  for l = 1:8  W = I((k-1)\*floor(dim(1)/8)+1:k\*floor(dim(1)/8),(l-1)\*floor(dim(2)/8)+1:l\*floor(dim(2)/8));  T(k,l) = sum(sum(W==0));  end  end  I = T;  sample=I(:)';  end |

|  |
| --- |
| Vertical slice |
| function [D] = vertSlice(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  B = img;  [r,c]= size(B);  r1 = floor(r/4);  c1 = floor(c/4);    t=0;  D = zeros(1,4);  E = [0 1];  e=0;  count = 1;  for i=r1:r1:r  C = B(i:i,:);  j=1;  t=0;  while(j<=c)  if(C(:,j)==0)  while(j<=c && C(:,j)~=1)  j=j+1;  end  if(j<=c && C(:,j)==1)  t=t+1;  end  end  if(j<=c && C(:,j)==1)  %t=t+1;  while(j<=c && C(:,j)~=0)  j=j+1;  end  if(j<=c && C(:,j)==0)  t=t+1;  end  end  end  D(1,count)=t;  count = count + 1;  end  end |

|  |
| --- |
| Horizontal slice |
| function [D] = horiSlice(img)  %img=imresize(img,[84 256]);  [r,c,k]= size(img);  if(k==1)% the image is already in single channel (gray or binary)  img = im2double(img);  elseif(k==3)  img = im2double(rgb2gray(img));  end    %img=im2double(img);  g=graythresh(img);  img=im2bw(img,g);  img=bwmorph(img,'clean');  B = img;  [r,c]= size(B);  %r1 = floor(r/4);  c1 = floor(c/4);  t=0;  D = zeros(1,4);  E = [0 1];  e=0;  count = 1;  for i=c1:c1:c  C = B(:,i:i);  j=1;  t=0;  while(j<=r)  if(C(j,:)==0)  while(j<=r && C(j,:)~=1)  j=j+1;  end  if(j<=r && C(j,:)==1)  t=t+1;  end  end  if(j<=r && C(j,:)==1)  %t=t+1;  while(j<=r && C(j,:)~=0)  j=j+1;  end  if(j<=r && C(j,:)==0)  t=t+1;  end  end  end  D(1,count)=t;  count = count + 1;  end  end |

11.2. Data generation:

|  |
| --- |
| Data generation |
| function [D]=genData()  D=ones(161,3000);  for i=1:3000  I=imread(sprintf('%d.png',i));  temp=GAF(I);  D(:,i)=temp';  end  end |

|  |
| --- |
| Feature mapping |
| function F = GAF(img)  f1=areaI(img);  [f2,f3]=CG(img);  f4=horiSlice(img);  f5=vertSlice(img);  f6=gradientHistogram(img);  f7=rowSum(img);  f8=colSum(img);  f9=dotcnt(img);  F=[f1 f2 f3 f4 f5 f6 f7 f8 f9];  end |

|  |
| --- |
| Training data generation |
| function P=trainD(D)  P=zeros(225,2400);  num=1;  for i=1:100:2400  for j=i:((i+80)-1)  P(:,num)=D(:,j);  num=num+1;  end  end  end |

|  |
| --- |
| Testing data generation |
| function P=testD(D)  P=zeros(225,600);  num=1;  for i=81:100:3000  for j=i:((i+20)-1)  P(:,num)=D(:,j);  num=num+1;  end  end  end |

11.3. Back-propagation

|  |
| --- |
| Training and Testing of BP |
| function [tr,ts,Time]=trainB(P,T,test)  tr=zeros(100,2);%train  ts=zeros(100,2);%test  Time=zeros(100,1);%time  num=1;  for j=1:100  %disp('with j number of neuron in 1st hidden layer');j=j  tic;  N=startSim(P,T,j,1000);  [w,x]=headCount(N,P);  tr(num,1)=w;  tr(num,2)=x;  [y,z]=hCount(N,test);  ts(num,1)=y;  ts(num,2)=z;  Time(num)=toc;  num=num+1;  if(w==0)  disp('End of training and testing');  j=j  disp('with j number of the system was able to learn train data with no error');  break;  end  end  end |

|  |
| --- |
| Target generation |
| function T=genT(I,P,S)  target=zeros(S,I);  t=ones(P,1);  c=0;  num=1;  for i=1:P:S  c=c+P;  target(i:c,num)=t;  num=num+1;  end  T=target;  T=T';  End |

|  |
| --- |
| Training error count |
| function [e1,e2]=headCount(N,test)  Y=sim(N,test);  R=vec2ind(compet(Y));  V=validT(80,2400);  S=(R==V);  S=sum(S);  %disp('WRONG prediction on train data');  e1=(2400-S);  %disp('% of ERROR');  e2=(double((2400-S)/2400)\*100);  end |

|  |
| --- |
| Testing error count |
| function [e1,e2]=hCount(N,test)  Y=sim(N,test);  R=vec2ind(compet(Y));  V=validT(20,600);  S=(R==V);  S=sum(S);  %disp('WRONG prediction on test data');  e1=(600-S);  %disp('% of ERROR');  e2=(double((600-S)/600)\*100);  end |

|  |
| --- |
| Validity check |
| function V=validT(P,S)  tt=ones(1,S);  num=1;  c=0;  for i=1:P:S  c=c+P;  tt(:,i:c)=num;  num=num+1;  end  V=tt;  End |

11.4. Perceptron

|  |
| --- |
| Training and Testing of Perceptron |
| function [rTr,rTs,Time]=trainP(P,T,Test)  rTr=zeros(15,2);%train  rTs=zeros(15,2);%test  Time=zeros(15,2);%time  num=1;  for i=1000:200:1200  tic;  N=perT(P,T,i);  Y1=sim(N,P);  Y2=sim(N,Test);  [e1,e2]=cntC(Y1);  [e3,e4]=cntT(Y2);  rTr(num,1)=e1;  rTr(num,2)=e2;  rTs(num,1)=e3;  rTs(num,2)=e4;  Time(num,1)=toc;  Time(num,2)=i;  num=num+1;  end  end |

|  |
| --- |
| Target generation |
| function T=genT(I,P,S)  target=zeros(S,I);  t=ones(P,1);  c=0;  num=1;  for i=1:P:S  c=c+P;  target(i:c,num)=t;  num=num+1;  end  T=target;  T=T';  End |

|  |
| --- |
| Training error |
| function [e1,e2]=cntC(Y)  n=1;  nn=80;  sam=0;  num=0;  for i=1:80:2400  sam=sam+nn;  t=sum(Y(n,i:sam));  num=num+t;  n=n+1;  end  %disp('WRONG prediction');  e1=(2400-num);  %disp('% of ERROR');  e2=(double((2400-num)/2400)\*100);  end |

|  |
| --- |
| Testing error |
| function [x1,x2]=cntT(Y)  n=1;  nn=20;  sam=0;  num=0;  for i=1:20:600  sam=sam+nn;  t=sum(Y(n,i:sam));  num=num+t;  n=n+1;  end  disp('WRONG prediction');  x1=(600-num)  disp('% of ERROR');  x2=(double((600-num)/600)\*100)    end |

11.5. Probabilistic neural network

|  |
| --- |
| Training and Testing of PNN |
| function [E1,E2,Time]=trainPnn(P,T1,Test,T2)  E1=zeros(40,3);  E2=zeros(40,3);  Time=zeros(40,2);  num=1;  for i=1:20  T=ind2vec(T1);  tic;  %train  net=newpnn(P,T,i);  Y=sim(net,P);  R1=vec2ind(Y);  [e1,e2]=count(T1,R1);  E1(num,1)=e1;  E1(num,2)=e2;  E1(num,3)=i;  %test  Y=sim(net,Test);  R2=vec2ind(Y);  [e3,e4]=count(T2,R2);  E2(num,1)=e3;  E2(num,2)=e4;  E2(num,3)=i;  Time(num,1)=toc;  Time(num,2)=i;  num=num+1;  end    end |

|  |
| --- |
| Training and Testing error |
| function [e1,e2]=count(T,Y)  [R,C]=size(Y);  S=sum(T==Y);  %disp('WRONG prediction');  e1=(C-S);  %disp('% of ERROR');  e2=(double((C-S)/C)\*100);  end |