Lab -2 Biometric identification using Neural Network

Introduction:

Biometrics allows a person to be identified and authenticated based on a set of recognizable and verifiable data, which are unique and specific to them.

Biometric authentication is the process of comparing data for the person's characteristics to that person's biometric "template" in order to determine resemblance. The reference model is first store in a database or a secure portable element like a smart card. The data stored is then compared to the person's biometric data to be authenticated. Here it is the person's identity which is being verified. Biometric identification consists of determining the identity of a person. The aim is to capture an item of biometric data from this person. It can be a photo of their face, a record of their voice, or an image of their fingerprint. This data is then compared to the biometric data of several other persons kept in a database.

Resources required: MATLAB: Neural Network Toolbox version R2018b.

Objective: To design the neural network for Biometric and Identify the target using various training algorithm with Back-propagation Neural Network.

Train the network without noise (1 sample) and with noise (4 samples**).** Once the network is trained well, demonstrate how the network identifies any of the learned object when the Boolean values of a character are input to the back-propagation network. Test the network using object with noise and without noise.

The available Training algorithms are as follows.

Trainbfg: BFGS quasi-Newton backpropagation.

Traincgb: Powell -Beale conjugate gradient backpropagation

Traincgf: Fletcher-Powell conjugate gradient backpropagation

Traingd: Gradient descent backpropagation

Traingdm: Gradient descent with momentum backpropagation

Trainlm: Levenberg-Marquardt backpropagation

Trainscg: Scaled conjugate gradient backpropagation

Fix the following parameters and comment on the training time and errors:

1. Epochs
2. Number of Hidden Layers
3. Number of Neurons in Each Hidden Layer
4. Transfer Function of Each Layer

Questions:

a) Prepare the data set such as inputs /output and Test /Test output similar with LAB-3.
b) Design neural network by varying Parameters such as number of layers, number of neuron in each layer and training algorithm.
c) Demonstrate the effectiveness of data handling through performance.
d) Comment your results in terms of RMS or Regression for the first training algorithm.
e) Use at least four training algorithm and vary training parameters 1 to 4 and comment your results.

Laboratory Activities:

Week 5 Session 2: Demonstration on "How to use Neural Network Toolbox" at the beginning of the class by tutor. Start working on the lab assignment. **Read carefully the sample demonstration as given.**

Week 6 Session 1: Continue working on the lab assignment.

Week7 Session 2: Report submission

Report Format

You are required to prepare a formal report for this lab assignment. Please make sure that your name and Student ID is included in the first page of your softcopy. Your report will be assessed based on the demonstration of the questions (a) to (e) as given above, however, the following aspects are essential:

1. Details of the objects (Biometric) chosen (with noise and without noise).
2. I/O preparation for training.
3. The network architecture of the working network.
4. Training parameters of the working network.
5. Comparative study on the performances with different training algorithms.

Sample demonstration: (Note: do not use any of the data as demonstrated below)

PART 1: Download and Save Fingerprint Images

Fingerprints samples can be obtained from various databases online or using Google Search. For this example, these two fingerprint samples are used and can be downloaded from the URL below.

https://drive.google.com/file/d/1VZUMLzrobQBFy2usbeX4lbg8jglkgnQr/view?usp=sharing

https://drive.google.com/file/d/1oHaQCjo-0tnzyGSwoh0lQf0hMKA1BOiS/view?usp=sharing

Save the images in MATLAB working directory before starting MATLAB sections. You can name the images anyway you want, but for this example, the images will be named as 'fingerprint1' and 'fingerprint2' respectively.

NOTE: You are highly encouraged to use your own images to try this example when you're confident of the procedures to train the network.

PART 2: MATLAB Codes

This part focuses on the codes required to import and process image, add noise, train network and test the network. Variable names can be changed to your preference, but be sure to be consistent with variable naming to prevent confusion and error in codes.

Before starting to code, it is always good practice to clear workspace, close all windows and clear the command window. These can be accomplished using the following codes.

```
clc;clear all;close all;
```

Step 1: Load Fingerprint Images

The first step is to load the images into MATLAB workspace. This is done using the built in function *imread* as shown below.

```
pic1 = imread('fingerprint1.jpg');
pic2 = imread('fingerprint2.jpg');
```

Step 2: Convert Image to Black and White (Binary)

Next step is to convert the image to black and white (1 and 0). This is done using the built in function, ***im2bw***. A threshold is selected from 0 to 1. This example uses a threshold of 0.5. Depending on the image used, the threshold might need to be changed.

Output Image = im2bw(I,Thresh)

I = image file needed to be thresholded

Thresh = Threshold value (0 – 1)

```
bw1 = im2bw(pic1,0.5);
bw2 = im2bw(pic2,0.5);
figure('Name','Fingerprint Images after Thresholding')
subplot(1,2,1),imshow(bw1,'InitialMagnification','fit'),axis square;
subplot(1,2,2),imshow(bw2,'InitialMagnification','fit'),axis square;
% Image size (For resizing purpose)
imrow = 192;
imcol = 112;
```

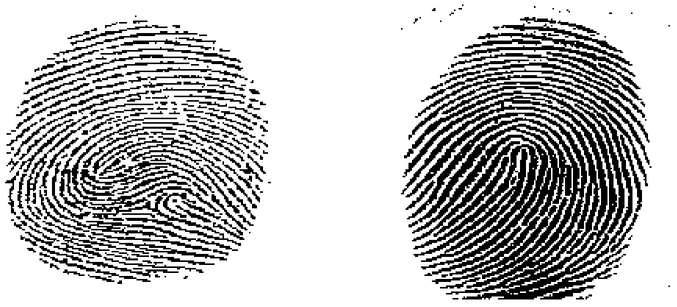


Figure 1:Fingerprint Images (After Thresholding)

Step 3: Create Duplicate of Specified Region with Randomized Pixels

There are many ways to produce noisy images. In this example, the images are produced using a custom function ***AddNoise***. This function randomly selects N number of pixels and inverts the values, thus producing noisy versions of the original fingerprint image to train.

The code checks for the 'FingerprintSamples.mat' file in the directory and loads this file. If it doesn't exist, then it proceeds to create and save the required variables in the same file. This is done so that the variables do not get erased accidentally during initialization of the workspace.

```
if exist('FingerprintSamples.mat','file')
    fprintf('The Variables Exist. Loading...\n');
    load('FingerprintSamples.mat');
else
    fprintf('Variables Do Not Exist. Creating Variables...\n');
    N = 100; % Number of pixels to be inverted
```

N represents number of pixels to be inverted. This value is used in the custom function, **AddNoise**. The following set of codes involve resizing images imported into a common size based on the *imrow* and *imcol* and selecting a window (range of pixels) from these images to use for neural network training. A window is selected because usage of the entire image would utilize too much memory.

However, you are welcome to try to use the entire image should you want to see what happens.

```
% Fingerprint 1
    f11_og = bw1; % Original Fingerprint
    f11_resize = imresize(f11_og,[imrow,imcol]);
    f11 = f11_resize((96:116),(56:76));
    f12 = AddNoise(f11,N);
    f13 = AddNoise(f11,N);
    f14 = AddNoise(f11,N);
    f15 = AddNoise(f11,N);

     % Fingerprint 2
    f21_og = bw2; % Original Fingerprint
    f21_resize = imresize(f21_og,[imrow,imcol]);
    f21 = f21_resize((96:116),(56:76));
    f22 = AddNoise(f21,N);
    f23 = AddNoise(f21,N);
    f24 = AddNoise(f21,N);
    f25 = AddNoise(f21,N);


save('FingerprintSamples.mat','f11','f12','f13','f14','f15','f21','f22','f23','f24','f25');
% Save variables for future training usage
end
```

Step 4: Plot Fingerprint Images

The resulting images are plot in the same plot using the subplot built in function. The syntax for the subplot function is as follows:

subplot(r,c,n),…

r = No. of rows of pictures

c = No. of columns of pictures

n = Picture No. (for placement)

```
figure('name','Input Images')
subplot(2,5,1),imshow(f11,'InitialMagnification','fit'),axis square;
subplot(2,5,2),imshow(f12,'InitialMagnification','fit'),axis square;
subplot(2,5,3),imshow(f13,'InitialMagnification','fit'),axis square;
subplot(2,5,4),imshow(f14,'InitialMagnification','fit'),axis square;
subplot(2,5,5),imshow(f15,'InitialMagnification','fit'),axis square;

subplot(2,5,6),imshow(f21,'InitialMagnification','fit'),axis square;
subplot(2,5,7),imshow(f22,'InitialMagnification','fit'),axis square;
subplot(2,5,8),imshow(f23,'InitialMagnification','fit'),axis square;
subplot(2,5,9),imshow(f24,'InitialMagnification','fit'),axis square;
subplot(2,5,10),imshow(f25,'InitialMagnification','fit'),axis square;
```

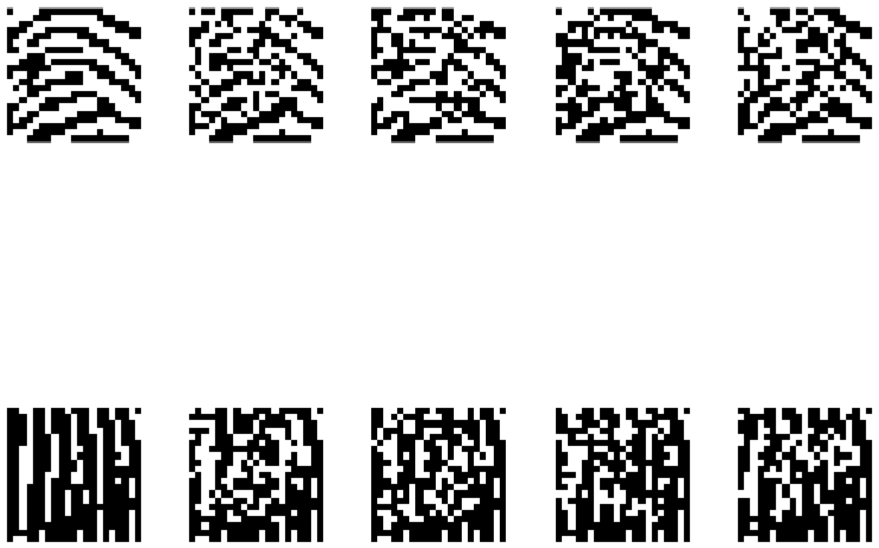


Figure 2: Input Images Incl. Noise

Step 5: Prepare Input for Training

The input images are then prepared to be trained. The images are reshaped to be 1 column and certain number of rows (denoted by [] in code below), with each image occupying one column of the training input.

Similar to previous examples, the training input is then transposed (switch row to column) using the transpose syntax.

```
biometric_in =
[reshape(f11,1,[]);reshape(f12,1,[]);reshape(f13,1,[]);reshape(f14,1,[]);reshape(f15,1,[]);

reshape(f21,1,[]);reshape(f22,1,[]);reshape(f23,1,[]);reshape(f24,1,[]);reshape(f25,1,[])];
 traininput = biometric_in';
figure('Name','Training Input')
imagesc(traininput)
```

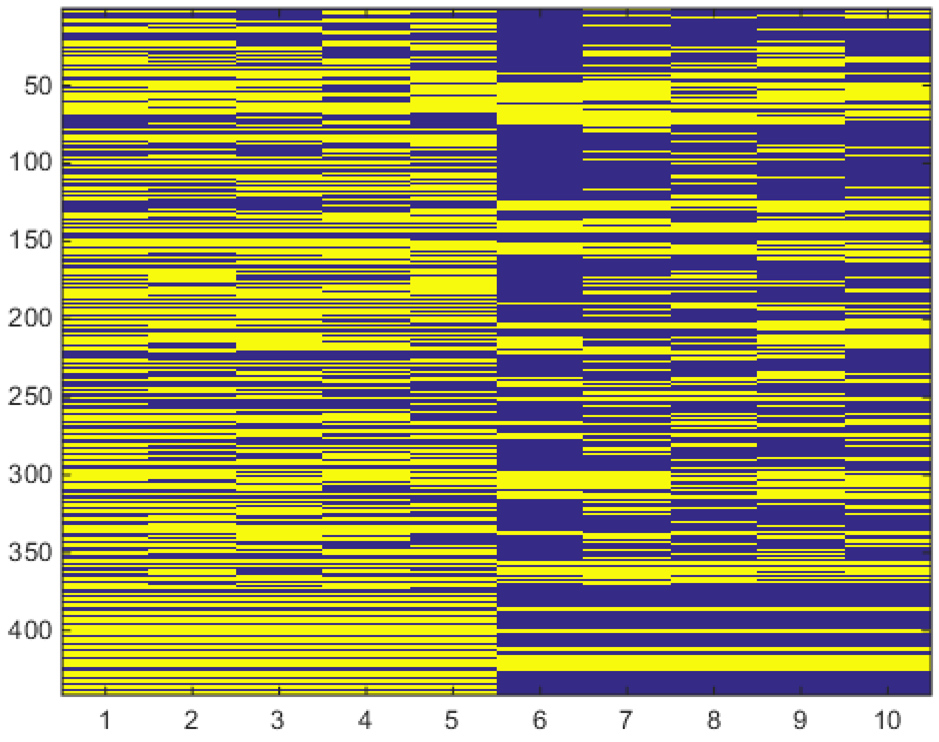


Figure 3: Image after Reshaping

Step 6: ANN Target

A target value is created for the images. This target is to determine which image set corresponds to which fingerprint. The first five images correspond to the first fingerprint sample while second five correspond to second sample. This is done by creating an array with zeros and inverting sections of array representing the first and second fingerprint images.

```
figure('Name','Target Output')
target = zeros(2,10);
target(1,1:5) = 1;
target(2,6:10) = 1;
imshow(target,'InitialMagnification','fit');
```

Figure 4: Target Output (White is output, black is background)

Step 7: Train Network

The network is trained using the train function. Before training, the network is created using the newff function. The syntax explanation for the newff function is shown below:

train1 = newff(traininput, target, [25,25], {'tansig','tansig', 'tansig'}, 'trainscg');

traininput = Input variables for network

target = Target output

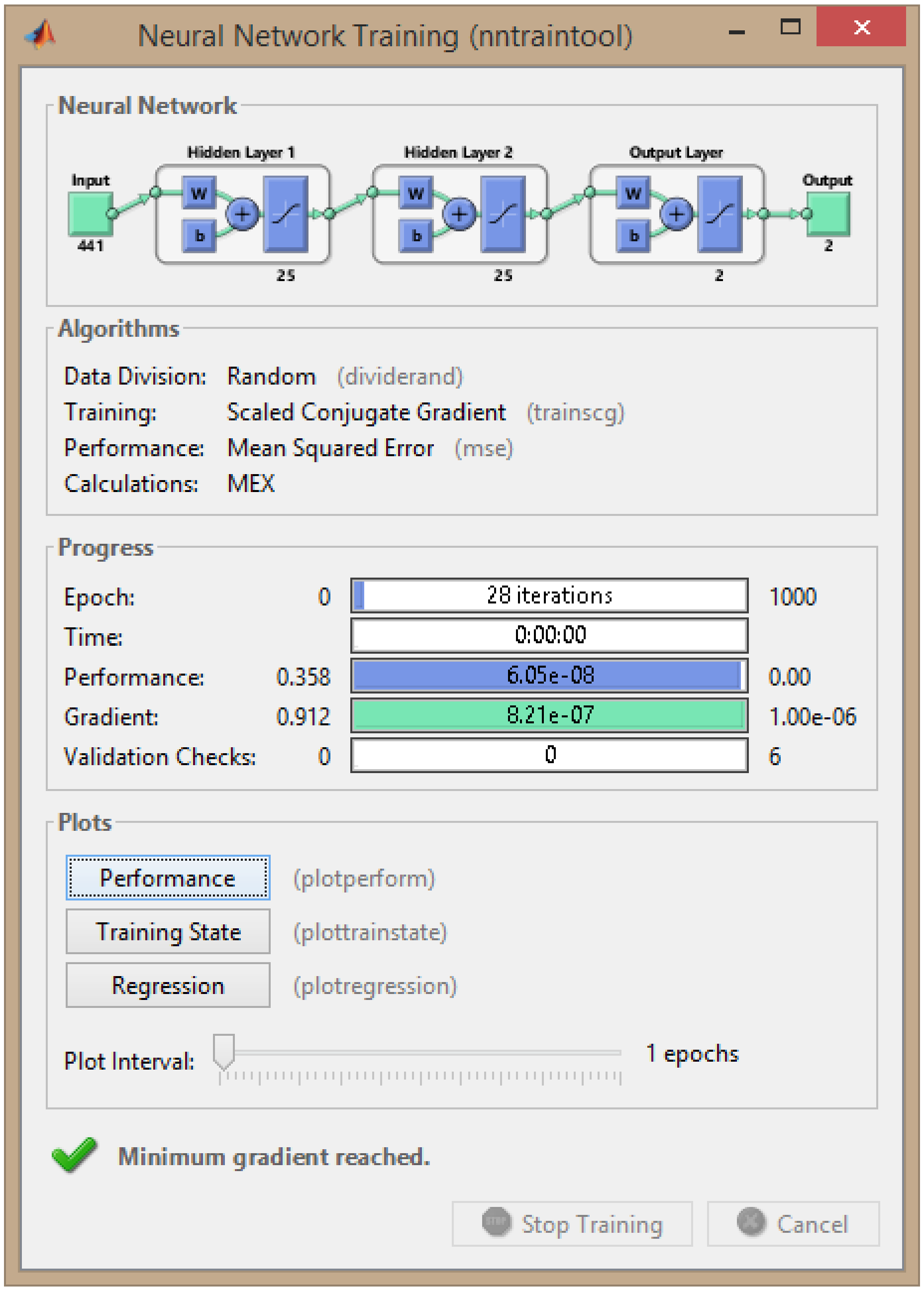
[25,25] = Number of hidden layers (how many numbers) and number of neurons in each layer

{‘tansig’,…} = Transfer function for each layer

‘trainscg’ = Training algorithm used for the netowork. Refer types of algorithms available (trainlm, trainbr, trainscg, trainbfg, etc.)

```
train1 = newff( traininput, target, [25,25], {'tansig','tansig', 'tansig'}, 'trainscg' );
train1.trainParam.show = 100;
train1.trainParam.epochs = 1000; % Number of epochs
train1.trainParam.lr = 0.08; % Learning Rate
train1.trainParam.goal = 0; % Goal
[net_out,train_rec,train_out,nn_err] = train(train1,traininput,target);
```

Training parameters can be set using the codes above as well. These include the number of epochs, learning rate (lr) and required goal. Running the codes above will open the nntool and perform training. The following window will appear to signify training process.


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random (dividerand)
Training:	Scaled Conjugate Gradient (trainscg)
Performance:	Mean Squared Error (mse)
Calculations:	MEX

Progress

Epoch:	0	28 iterations	1000
Time:		0:00:00	
Performance:	0.358	6.05e-08	0.00
Gradient:	0.912	8.21e-07	1.00e-06
Validation Checks:	0	0	6

Plots

Performance	(plotperform)
Training State	(plottrainstate)
Regression	(plotregression)

Plot Interval: 1 epochs

Minimum gradient reached.

Figure 5: nntool Training

Once training is completed, the window can be minimized and the trained network can be used to simulate. Unlike using the nntool, usage of the code **does not** require exporting the trained network to the workspace.

‘**net_out’** is the trained network required for simulation. ‘**train_out**’ is the output of the training. Next, the trained network from ‘net_out’ will be tested using the images to see if proper recognition of thumbprint can be performed.

The performance graph and regression can also be obtained by clicking the appropriate buttons in the window above. The figures below show examples of these two metrics.

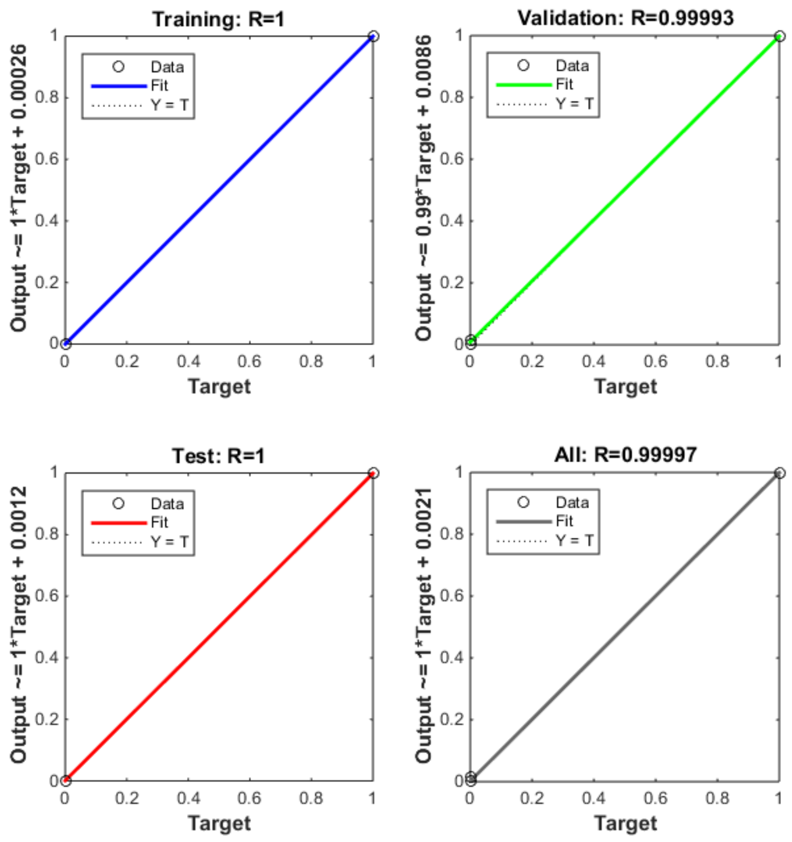


Figure 6: Regression Plots

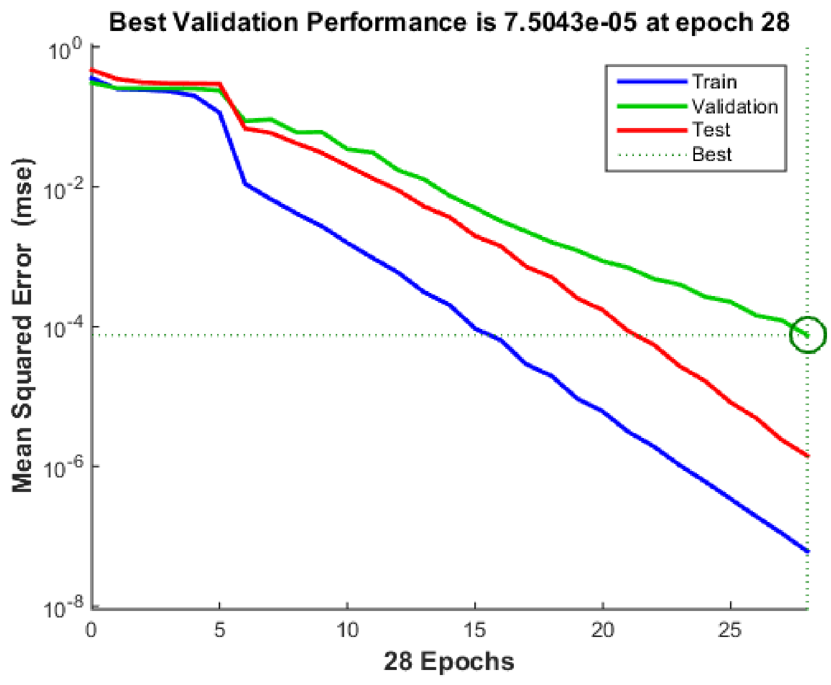


Figure 7: Performance Plots

Step 8: Simulate Output

The final step is to simulate the trained network with the inputs to see if the output of network is same as the target output. This is done using the ***sim*** built in function provided in MATLAB. If output is different, the network has not been trained properly and retraining is required (run the code again). Note that the output needs to be PERFECT.

```
NN_Out = sim(net_out,traininput);
 NN_Out_round = round(NN_Out);
 figure('name','Neural Network Output')
 imshow(NN_Out,'InitialMagnification','fit')
 figure('name','NN Output(Rounded)')
 imshow(NN_Out_round,'InitialMagnification','fit')
```

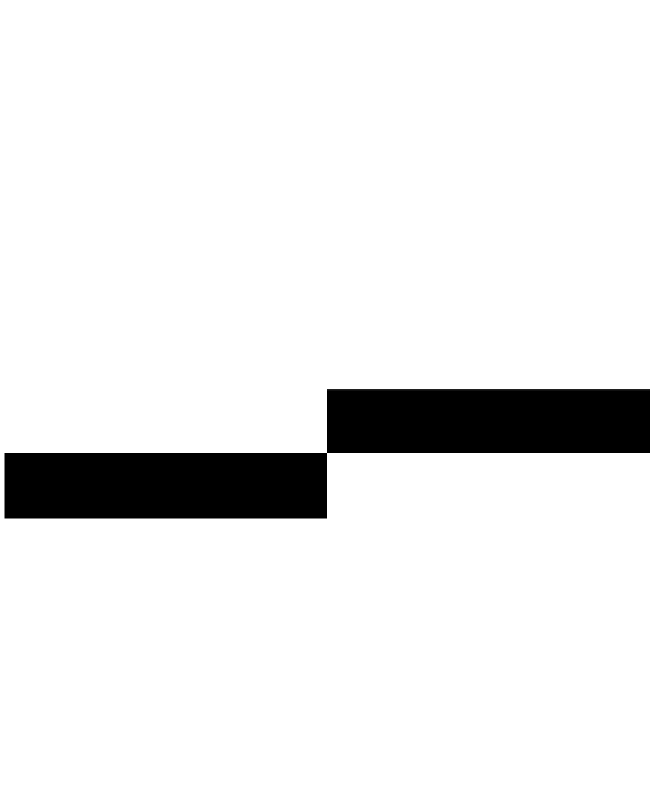


Figure 8: Simulated Output (White is output, black is background)

PART 3: Further Experimentation

Further experimentation can be done. These include adding more fingerprint samples to the data. Besides that, the performance of the networks can be gauged using the performance graph. You can also try changing the training parameters to see the effects on the network performance and effectiveness.