Testing Different Classification Approaches Based on Face Recognition

AHMED ABULILA
ECE539 FINAL PROJECT

UNIVERSITY OF WISCONSIN MADISON
Introduction

Identity verification solutions are one of the hot and critical topics today. The biometric based algorithms are the most powerful way to validate that the persons are who they say they are, because it is based on measurable biological and behavioral characteristics to recognize an individual, such as fingerprint, DNA, typing rhythm, and face recognition, which each one of them is unique for each person. The performance of this type is based on the robustness of the encryption system that is used, such as Advanced Encryption System (ADS), and the accurate of detection or classification algorithm.

The project has been concerned on the pattern classification part. The introduced application is a face recognition system, which is used to identify and provide access control on a group of people.

The first part of the project is about the data-set pre-processing. First data-set preprocessing step is images cropping algorithm which generates new images, all of them are with the same size (No. of features) by detecting the face and extracting it only, this step aimed to minimize the no. of features by omitting irrelevant features, such as backgrounds. The second pre-processing step is dividing the data-set to training, validating, and testing data-sets. The third step is generating different versions of the data-set each one with different type of compression and adding the appropriate labels.

The second part introduces different pattern recognition algorithms, such as Multi-Layer Perceptron (MLP), K-nearest neighbor classification (K-NN), and Support Vector Machine (SVM). The last section is a discussion about the result of each classifier and each version of compression level to determine which one is the best choice.
Data Set

The application is based on Caltech 101 face data-set, which is frontal face dataset, collected by Markus Weber at California Institute of Technology (Caltech). It consists of 450 face images for 27 or so unique people under different lighting, expressions, and backgrounds.

Each image of the data-set is an RGB image with 510 x 337 pixels. As shown in Fig. 1, the image has huge portion of unimportant area, which is the background. Long pre-processing phase is needed, because of the condition of the data-set.

Fig. 1 Sample Image of the Used Data-Set
Pre-Processing Data

One of the challenges of pattern recognition is the data pre-processing, because it affects in the performance of the classifier by changing the no. of feature. If it is huge, it will take a long time during the learning phase to give an acceptable result, on the other hand, if it is too small, it will not be able to learn. At the same time, a good feature transformation can difficult classification problem easier to solve, but a bad feature transformation can lead the classifier to not converge.

The pre-processing phase is the longest phase in the project. It consists of three steps. First one is images cropping to remove any unimportant features. Second step is dividing data to authorized data and non-authorized ones, in addition to dividing each group of them to training and testing groups. Final step is preparing the data to classification phase.

1- Image Cropping

Image cropping can’t be done manually, because the final no. of pixels is same for all images, also, the data-set is huge to be processed manually. Harpia Tool has been used to generate the cropped images. Fig. 2 shows the Harpia block diagram of the used design.

![Fig. 2 Harpia Tool Block Diagram](image-url)
Block 1 is used to call the images, like the one shown in Fig. 1. Block 2 is used to make a color conversion in the image from block 1, this block has been used sometimes when block 3, face detector, couldn’t detect the face right, color conversion can help the face detector to work as expected. Block 3 generates the coordination of the detected face and an image which is the same image from block 2 plus a highlight on the detected area, as shown in Fig. 3 using block 4. The coordinate from block 3 and block 1 image are applied to block 5 which crops block 3 detected coordinates from block 1 image, the result is shown in fig. 4 using block 6. As mentioned above, all data set images are needed to have the same size, this the responsibility of the rest of block diagram. A reference image is applied to block 7, block 8 gets the size of the reference image and provide it to block 9, which resize the image from block 5 to be the same size as the reference image.

![Fig. 3 Block 3 Output Image](image-url)
The Generated Code of the block diagram:

```c
// Auto-generated C Code - S2i Harpia
/
* In order to compile this source code run, in a terminal window, the following
* command:
*   gcc sourceCodeName.c `pkg-config --libs --cflags opencv` -o outputProgramName
* the `pkg-config ... opencv` parameter is a inline command that returns the path to
both
* the libraries and the headers necessary when using opencv. The command also
returns other necessary compiler options.
*/
// header:
#include <stdio.h>
#include <stdlib.h>
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
#include <math.h>
#define PI 3.1415926535898
double rads(double degs)
{
    return (PI/180 * degs);
}
int main( int argc, char ** argv)
{
    //declaration block
    char block3_arg_Filename[] = "/abulila/Desktop/ECE539/Project
work/FacesModified/1/i1.jpg";
    char block10_arg_Filename[] = "/abulila/Desktop/ECE539/Project
work/FacesModified/1/2.png";
    IplImage * block3_img_o1 = NULL;
    IplImage * block10_img_o1 = NULL;
    IplImage * block5_img_i1 = NULL;
    CvPoint block5_point_o1 = cvPoint(0,0);
    CvRect block5_rect_o2 = cvRect( 0, 0, 1, 1);
    IplImage * block5_img_o3 = NULL;
```
double block5_double_o4 = 0.0;
static CvMemStorage* block5_storage = 0;
static CvHaarClassifierCascade* block5_cascade = 0;
const char* block5_cascade_name = "/usr/share/harpia/images/haarcascade_frontalface_alt2.xml";
IplImage* block9_img_i1 = NULL;
CvRect block9_rect_o1 = cvRect( 0, 0, 1, 1);
IplImage* block14_img_i1 = NULL;
IplImage* block14_img_o1 = NULL;
IplImage* block6_img_i1 = NULL;
IplImage* block6_img_o1 = NULL;
CvRect block6_rect_i2;
IplImage* block13_img_i1 = NULL;
IplImage* block13_img_o1 = NULL;
IplImage* block7_img_i1 = NULL;
IplImage* block7_img_o1 = NULL;
CvRect block7_rect_i2;
IplImage* block11_img_i1 = NULL;
IplImage* block11_img_o1 = NULL;
IplImage* block12_img_i1 = NULL;
IplImage* block12_img_o1 = NULL;

//execution block
//Weight: 1
block3_img_o1 = cvLoadImage(block3_arg_Filename,-1);
block6_img_i1 = cvCloneImage(block3_img_o1); // IMAGE conection
block5_img_i1 = cvCloneImage(block3_img_o1); // IMAGE conection
//Weight: 1
block10_img_o1 = cvLoadImage(block10_arg_Filename,-1);
block9_img_i1 = cvCloneImage(block10_img_o1); // IMAGE conection
//Weight: 2
if (block5_img_i1){
double scale = 1.3;
block5_cascade = (CvHaarClassifierCascade*)cvLoad(block5_cascade_name, 0, 0, 0);
IplImage* gray = cvCreateImage(cvSize(block5_img_i1->width,block5_img_i1->height), 8, 1);
IplImage* small_img = cvCreateImage(cvSize(cvRound(block5_img_i1->width/scale),cvRound(block5_img_i1->height/scale)),8,1);
cvCvtColor(gray, small_img, CV_BGR2GRAY);
cvResize(small_img, small_img, CV_INTER_LINEAR);
cvEqualizeHist(small_img, small_img);
if (block5_img_o3){
block5_img_o3 = cvCloneImage(block5_img_i1);
cvCopy(block5_img_i1,block5_img_o3,0);
block5_storage = cvCreateMemStorage(0);
cvClearMemStorage(block5_storage);
block5_rect_o2 = cvRect(0,0,1,1);
CvSeq* faces = cvHaarDetectObjects(small_img, block5_cascade, block5_storage,1.1, 2, 0/*CV_HAAR_DO_CANNY_PRUNING*/,
cvSize(30,30));
block5_double_o4 = faces->total;
if (faces){
  int i;
  for (i = 0; i < (faces ? faces->total : 0); i++) {
    CvRect* r = (CvRect*)cvGetSeqElem(faces, i);
    if (r)
CvPoint center;
int radius;
center.x = cvRound((r->x + r->width*0.5)*scale);
center.y = cvRound((r->y + r->height*0.5)*scale);
radius = cvRound((r->width + r->height)*0.25*scale);
cvCircle( block5_img_o3, center, radius, cvScalarAll(0), 3, 8, 0 );

if(i == 0)
{
  block5_point_o1 = center;
  block5_rect_o2.x = (r->x)*scale;
  block5_rect_o2.y = (r->y)*scale;
  block5_rect_o2.width = (r->width)*scale;
  block5_rect_o2.height = (r->height)*scale;
}

block6_rect_i2 = block5_rect_o2; // RECT conection
block14_img_i1 = cvCloneImage(block5_img_o3);// IMAGE conection
//Weight: 2
if(block9_img_i1)
{
  block9_rect_o1 = cvRect( 0, 0, block9_img_i1->width, block9_img_i1->height);
}
block7_rect_i2 = block9_rect_o1; // RECT conection
//Weight: 3
if(block14_img_i1){
  block14_img_o1 = cvCloneImage(block14_img_i1);
  cvSaveImage("block14_OUT.png",block14_img_i1);} //Weight: 4
if(block6_img_i1){
  block6_rect_i2.x = MAX(0,block6_rect_i2.x);//Check whether point is negative
  block6_rect_i2.y = MAX(0,block6_rect_i2.y);
  block6_rect_i2.x = MIN(block6_img_i1->width-1,block6_rect_i2.x);//Check whether point is out of the image
  block6_rect_i2.y = MIN(block6_img_i1->height-1,block6_rect_i2.y);
  block6_rect_i2.width = MIN(block6_img_i1->width-
   block6_rect_i2.x,block6_rect_i2.width);//Check whether rect reaches out of the image
  block6_rect_i2.height = MIN(block6_img_i1->height-
   block6_rect_i2.y,block6_rect_i2.height);
  block6_img_o1 = cvCreateImage(cvSize(block6_rect_i2.width,block6_rect_i2.height),
   block6_img_i1->depth,block6_rect_i2.height),
   block6_img_i1->depth-
   block6_rect_i2.y,block6_rect_i2.height);
  cvSetImageROI(block6_img_i1,block6_rect_i2);
  cvCopyImage(block6_img_i1,block6_img_o1);
}
block7_img_i1 = cvCloneImage(block6_img_o1);// IMAGE conection
block13_img_i1 = cvCloneImage(block6_img_o1);// IMAGE conection
//Weight: 7
if(block13_img_i1){
block13_img_o1 = cvCloneImage(block13_img_i1);
cvSaveImage("block13_OUT.png", block13_img_i1);
//Weight: 9

if(block7_img_i1){
    block7_img_o1 = cvCreateImage(cvSize(block7_rect_i2.width, block7_rect_i2.height), block7_img_i1->depth, block7_img_i1->nChannels);
    cvResize(block7_img_i1, block7_img_o1, CV_INTER_LINEAR);
}
block11_img_i1 = cvCloneImage(block7_img_o1); // IMAGE connection
block12_img_i1 = cvCloneImage(block7_img_o1); // IMAGE connection

//Weight: 20
if(block11_img_i1){
    block11_img_o1 = cvCloneImage(block11_img_i1);
cvSaveImage("block11_OUT.png", block11_img_i1);
//Weight: 20
block12_img_o1 = cvCloneImage(block12_img_i1);

if(block12_img_i1)
cvSaveImage("/abulila/Desktop/ECE539/Project work/FacesModified/1/1", block12_img_i1);

//deallocation block
cvReleaseImage(&block3_img_o1);
cvReleaseImage(&block10_img_o1);
cvReleaseImage(&block5_img_o3);
cvReleaseImage(&block5_img_i1);
cvReleaseMemStorage(&block5_storage);
cvReleaseImage(&block9_img_i1);
cvReleaseImage(&block14_img_o1);
cvReleaseImage(&block14_img_i1);
cvReleaseImage(&block6_img_o1);
cvReleaseImage(&block6_img_i1);
cvReleaseImage(&block13_img_o1);
cvReleaseImage(&block13_img_i1);
cvReleaseImage(&block7_img_o1);
cvReleaseImage(&block7_img_i1);
cvReleaseImage(&block11_img_o1);
cvReleaseImage(&block11_img_i1);
cvReleaseImage(&block12_img_o1);
cvReleaseImage(&block12_img_i1);

return 0;
} //closing main()
2- Data Grouping

The data-set is for 27 different person, therefore, the cropped data-set has been divided into two groups one for authorized persons and the other one is for non-authorized persons, then each group is divided to a training set and a testing set. The authorized group includes 8 persons, who are shown in Fig. 5.

![Authorized Persons](image)

Fig. 5. Authorized Persons

3- Feature Extracting

Matlab Code has been developed to prepare the data set for the classification phase. The code read all data set, and has provided two different versions of the data set one with compression scale of 2 to be (79x79) and the other with 4 to be (40x40). Also, the code has generated two types of labels, one suitable with SVM, and the other suitable with both MLP and K-NN. The final process was randomizing the data-set to get a good learning performance.
The developed Matlab codes:

(i) Code for non-authorized data-set.

```matlab
% Clearing workspace and directory paths
clear all
myfolder = '/abulila/Desktop/ECEFinalProj/FaceNoOk/Tarining';
filePattern = fullfile(myfolder, '*.png');
facetrainfiles = dir(filePattern);
imagesTrainBy2 = [];
imagesTrainBy4 = [];
for k = 1:length(facetrainfiles),
    baseFileName = facetrainfiles(k).name;
    fullFileName = fullfile(myfolder, baseFileName);
    imageArray = imread(fullFileName);
    imageArray1 = rgb2gray(imageArray);
    imageArray = imresize(imageArray1, 0.5);
    imagesTrainBy2 = [imagesTrainBy2; imageArray(:)'];
    imageArray = imresize(imageArray1, 0.25);
    imagesTrainBy4 = [imagesTrainBy4; imageArray(:)'];
end

% Directory path for testing files
myfolder1 = '/abulila/Desktop/ECEFinalProj/FaceNoOk/Testing';
filePattern1 = fullfile(myfolder1, '*.png');
facetrainfiles1 = dir(filePattern1);
imagesTestBy2 = [];
imagesTestBy4 = [];
for k = 1:length(facetrainfiles1),
    baseFileName1 = facetrainfiles1(k).name;
    fullFileName1 = fullfile(myfolder1, baseFileName1);
    imageArray = imread(fullFileName1);
    imageArray1 = rgb2gray(imageArray);
    imageArray = imresize(imageArray1, 0.5);
    imagesTestBy2 = [imagesTestBy2; imageArray(:)'];
    imageArray = imresize(imageArray1, 0.25);
    imagesTestBy4 = [imagesTestBy4; imageArray(:)'];
end

% Storing features and labels for training and testing
DataNotFaceTrainBy2 = [imagesTrainBy2];
DataNotFaceTrainBy4 = [imagesTrainBy4];
DataNotFaceTestBy2 = [imagesTestBy2];
DataNotFaceTestBy4 = [imagesTestBy4];

h = length(imagesTrainBy2(:, 1));
LabelTrain = [ones(h, 1) zeros(h, 1)];
LabelTrainSVM = [-1*ones(h, 1)];

h = length(imagesTestBy2(:, 1));
LabelTest = [ones(h, 1) zeros(h, 1)];
LabelTestSVM = [-1*ones(h, 1)];
```

clear all
myfolder = '/abulila/Desktop/ECEFinalProj/FaceOk/Training';
filePattern = fullfile(myfolder, '*.png');
facetrainfiles = dir(filePattern);
imagesTrainBy2 = [];
imagesTrainBy4 = [];

for k = 1:length(facetrainfiles),
    baseFileName = facetrainfiles(k).name;
    fullFileName = fullfile(myfolder, baseFileName);
    imageArray = imread(fullFileName);
    imageArray1 = rgb2gray(imageArray);
    imageArray = imresize(imageArray1, 0.5);
    imagesTrainBy2 = [imagesTrainBy2; imageArray(:)'];
    imageArray = imresize(imageArray1, 0.25);
    imagesTrainBy4 = [imagesTrainBy4; imageArray(:)'];
end

myfolder1 = '/abulila/Desktop/ECEFinalProj/FaceOk/Testing';
filePattern1 = fullfile(myfolder1, '*.png');
facetrainfiles1 = dir(filePattern1);
imagesTestBy2 = [];
imagesTestBy4 = [];

for k = 1:length(facetrainfiles1),
    baseFileName1 = facetrainfiles1(k).name;
    fullFileName1 = fullfile(myfolder1, baseFileName1);
    imageArray = imread(fullFileName1);
    imageArray1 = rgb2gray(imageArray);
    imageArray = imresize(imageArray1, 0.5);
    imagesTestBy2 = [imagesTestBy2; imageArray(:)'];
    imageArray = imresize(imageArray1, 0.25);
    imagesTestBy4 = [imagesTestBy4; imageArray(:)'];
end

DataFaceTrainBy2 = [imagesTrainBy2];
DataFaceTrainBy4 = [imagesTrainBy4];
DataFaceTestBy2 = [imagesTestBy2];
DataFaceTestBy4 = [imagesTestBy4];

h = length(imagesTrainBy2(:, 1));
LabelFaceTrain = [zeros(h, 1) ones(h, 1)];

LabelFaceTrainSVM = [ones(h, 1)];

h = length(imagesTestBy2(:, 1));
LabelFaceTest = [zeros(h, 1) ones(h, 1)];

LabelFaceTestSVM = [ones(h, 1)];
(iii) Putting all together.

clear all;
clc;

load('FaceData.mat');
load('NoFaceData.mat');

DataTrainBy2=randomize([double(DataNotFaceTrainBy2)
LabelTrain;double(DataFaceTrainBy2) LabelFaceTrain]);

DataTrainBy2SVM=randomize([double(DataNotFaceTrainBy2)
LabelTrainSVM;double(DataFaceTrainBy2) LabelFaceTrainSVM]);

DataTrainBy4=randomize([double(DataNotFaceTrainBy4)
LabelTrain;double(DataFaceTrainBy4) LabelFaceTrain]);

DataTrainBy4SVM=randomize([double(DataNotFaceTrainBy4)
LabelTrainSVM;double(DataFaceTrainBy4) LabelFaceTrainSVM]);

DataTestBy2=randomize([double(DataNotFaceTestBy2)
LabelTest;double(DataFaceTestBy2) LabelFaceTest]);

DataTestBy2SVM=randomize([double(DataNotFaceTestBy2)
LabelTestSVM;double(DataFaceTestBy2) LabelFaceTestSVM]);

DataTestBy4=randomize([double(DataNotFaceTestBy4)
LabelTest;double(DataFaceTestBy4) LabelFaceTest]);

DataTestBy4SVM=randomize([double(DataNotFaceTestBy4)
LabelTestSVM;double(DataFaceTestBy4) LabelFaceTestSVM]);
Classification Phase

Three different classifiers have been used to detect the authorization of each person on the data set. The three classifiers are SVM, K-NN, and MLP.

1- Support Vector Machine Classifier

Support Vector Machine (SVM) attempts to divide a feature space with a hyper-plane so that the feature vectors with different labels are on one side or the other. SVMs are different from other linear classifiers in that they maximize the margin between the hyper-plane and the data points. If the data is not linearly separable, SVMs use a smudge factor and minimize the error.

SVM classifier has been used with radial basis function as kernel function. Also, a scaling factor (sigma) has been needed to adjust to make it work with the huge number of features.

2- K-Nearest Neighbor Classifier

K-Nearest Neighbor classifier assigns a test data point a label that is the same as the majority of the K nearest neighboring training points. It is a very simple pattern classifier but also very effective.

K-NN classifier has been used based on the implemented code, which is available in the course web-site.

3- Multi-Layer Perceptron

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis, etc.

The Neural Network Pattern Recognition Tool included in Matlab has been employed to classify personal authority.
Results and Discussion

In this section, codes, parameters, and result for each classifier is going to be discussed.

1- Support Vector Machine Classifier

SVM is used for both version of data-sets. For both data-set, SVM with radial basis kernel function has been used. It didn't work at the first attempts but after increasing sigma scale it has converged with high classification rates. In the case of data-set with compression of 2, the sigma scale is 40, and the classification rate has reached 96.49%. On the other side, the classifier has reached the same classification rate with the 4 scale compressed data set using sigma value of 20.

The developed code:

(i) Data-set with compression of 2

```matlab
Feature=DataTrainBy2SVM(:,1:6241);  
Out=DataTrainBy2SVM(:,6242);  
Ftst=DataTestBy2SVM(:,1:6241);  
Otst=DataTestBy2SVM(:,6242);  
[svmStructure] = svmtrain(Feature,Out,'KERNEL_FUNCTION','rbf','rbf_sigma',30);  
[predictClass] = svmclassify(svmStructure,Ftst);  
Classification_rate=0;  
L=length(Otst);  
for i=1:L,  
   if(predictClass(i)==Otst(i)),  
      Classification_rate=Classification_rate+1;  
   end  
end  
Classification_rate=Classification_rate*100/L;
```
(ii) Data-set with compression of 4

```matlab
Feature=DataTrainBy4SVM(:,1:1600);
Out=DataTrainBy4SVM(:,1601);
Ftst=DataTestBy4SVM(:,1:1600);
Otst=DataTestBy4SVM(:,1601);
[supportVectors, SVMStruct] = svmtrain(Feature,Out,'KERNEL_FUNCTION','rbf','rbf_sigma',20);
[predictClass] = svmpredict(svmStruct,Ftst);
Classification_rate=0;
L=length(Otst);
for i=1:L,
    if(predictClass(i)==Otst(i)),
        Classification_rate=Classification_rate+1;
    end
end
Classification_rate=Classification_rate*100/L;
```

2- K-Nearest Neighbor Classifier

K-NN classifier has been used for both data-sets. With compressed data-set by 4, it reached a classification rate of 62.28% with 2 nearest neighbors used. On the other hand, with compressed data-set by 2, it reached a classification rate of 64.91% with using 2 nearest neighbors.

The developed code:

(i) Data-set with compression of 2

```matlab
Features=DataTrainBy2(:,1:6241);
Out=DataTrainBy2(:,6242:6243);
Ftst=DataTestBy2(:,1:6241);
Otst=DataTestBy2(:,6242:6243);
N=length(Features(:,1));
Q=length(Otst(:,1));
Pr=Features; Tr=Out;
Pt=Ftst;
fileID=fopen('P3b.txt','wt');
```
for k=1:7,
    Class=knn(Pr,Tr,Pt,k);
    disp([int2str(k) 'Output = '])
    Class
fprintf(fileID,'For K=%f :

',k);
fprintf(fileID,'The Confusion Matrix is:

');
    for i=1:Q,
        fprintf(fileID,'%d   %d   %d
',Class(i,:));
    end
    if k==1,
        Classk1=Class;
    elseif k==2,
        Classk2=Class;
    elseif k==3,
        Classk3=Class;
    elseif k==4,
        Classk4=Class;
    elseif k==5,
        Classk5=Class;
    elseif k==6,
        Classk6=Class;
    else
        Classk7=Class;
    end
end
fclose(fileID);

(ii) Data-set with compression of 4

Features=DataTrainBy4(:,1:1600);
Out=DataTrainBy4(:,1601:1602);
Ftst=DataTestBy4(:,1:1600);
Otst=DataTestBy4(:,1601:1602);

N=length(Features(:,1));
Q=length(Otst(:,1));

Pr=Features; Tr=Out;
Pt=Ftst;

fileID=fopen('P3b.txt','wt');

for k=1:7,
    Class=knn(Pr,Tr,Pt,k);
    disp([int2str(k) 'Output = '])
    Class
fprintf(fileID,'For K=%f :

',k);
fprintf(fileID,'The Confusion Matrix is:

');
    for i=1:Q,
        fprintf(fileID,'%d   %d   %d
',Class(i,:));
    end
end
```matlab
fprintf(fileID, '\n ');
fprintf(fileID, '%d %d %d', Class(i,:));
end
if k==1,
    Classk1=Class;
elseif k==2,
    Classk2=Class;
elseif k==3,
    Classk3=Class;
elseif k==4,
    Classk4=Class;
elseif k==5,
    Classk5=Class;
elseif k==6,
    Classk6=Class;
else
    Classk7=Class;
end
fclose(fileID);

3- Multi-Layer Perceptron

The Neural Network Pattern Recognition Tool included in Matlab has been trained to be able to classify authorized person and non-authorized person. The computer specification couldn’t handle training of a MLP for the compressed data set by 2 because the huge no. of features (6242), the computer main specifications are Intel i7 processor with 8 GB RAM, and 64-bit OS. However, it has worked fine with the other data-set and has provided a surprising classification rate of 100%, the training has been re-run many times to be sure about the result, Fig. 6 shows the training phase confusion matrices, and Fig. 7 shows the testing confusion matrix.
Fig. 6 MLP Training Confusion Matrices
The MLP generated code:

```matlab
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by NPRTOOL
% Created Fri Dec 20 17:59:47 CST 2013
%
% This script assumes these variables are defined:
% % Feature - input data.
% % Out - target data.

Feature=DataTrainBy4(:,1:1602);
Out=DataTrainBy4(:,1601:1602);
Ftst=DataTestBy4(:,1:1602);
Otst=DataTestBy4(:,1601:1602);

inputs = Feature';
targets = Out';

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.inputs{1}.processFcn = {'removeconstantrows','mapminmax'};
net.outputs{2}.processFcn = {'removeconstantrows','mapminmax'};
```
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% For help on training function 'trainscg' type: help trainscg
% For a list of all training functions type: help nntrain
net.trainFcn = 'trainscg'; % Scaled conjugate gradient

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse'; % Mean squared error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform', 'plottrainstate', 'ploterrhist', ...
'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% Recalculate Training, Validation and Test Performance
trainTargets = targets .* tr.trainMask{1};
valTargets = targets .* tr.valMask{1};
testTargets = targets .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,outputs)
valPerformance = perform(net,valTargets,outputs)
testPerformance = perform(net,testTargets,outputs)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotconfusion(targets,outputs)
%figure, plotroc(targets,outputs)
%figure, ploterrhist(errors)
Conclusion

Data-set preprocessing can enhance the training phase, and make it possible, as noticed in the case of training an MLP with the compressed data-set by 2, it wasn’t possible to be done on the available machine. In addition, after compressing data from (157x157) to (40x40) it still give the required data for making a successful classifier. All results are summarized in Table 1.

<table>
<thead>
<tr>
<th>Classifier Type</th>
<th>Classification Rate</th>
<th>Used Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM with compressed data set by 2</td>
<td>96.49%</td>
<td>Sigma=40</td>
</tr>
<tr>
<td>SVM with compressed data set by 4</td>
<td>96.49%</td>
<td>Sigma=20</td>
</tr>
<tr>
<td>K-NN with compressed data set by 2</td>
<td>64.91%</td>
<td>K=2</td>
</tr>
<tr>
<td>K-NN with compressed data set by 4</td>
<td>62.28%</td>
<td>K=6</td>
</tr>
<tr>
<td>MLP with compressed data set by 2</td>
<td>NA</td>
<td>Neurons=NA</td>
</tr>
<tr>
<td>MLP with compressed data set by 4</td>
<td>100%</td>
<td>Neurons=10</td>
</tr>
</tbody>
</table>

References

- Course web-site:  
  http://homepages.cae.wisc.edu/~ece539/

- Data-set web-site:  
  http://www.vision.caltech.edu/Image_Datasets/Caltech101/

- MLP tool:  
