UNIVERSITY OF BALAMAND

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING

CPEN 425

Neural Network Design

Final Project

Submitted to: Dr. Issam Dagher

Submitted by: Tania Aoude (A2010001)

Hala Msalem (A2020119)

I. Introduction

The purpose of this project is to extract the Local Binary Pattern (LBP) features of the Our Database of Faces (ORL) dataset which consists of 40 people with 10 images for each person. These extracted features are then used as inputs for a multilayer Perceptron network whose weights are updated using backpropagation. Once the network is trained on the training dataset it is then tested using the testing dataset in order to classify the image as one of the 40 people.

II. Solution

1.Our Database of Faces (ORL)

The ORL dataset consists of 400 images of 40 different people, each person has 10 images in which they have different facial expressions, facial details and are looking at the camera through multiple angles under different lighting. The images all feature a monotone dark background against which the people are photographed, with 256 grey levels per pixel and an image size of 92x112.

Figure 1: Sample of Four Photos for Four People from the ORL Dataset

Figure 1 shows a sample of 4 photos for 4 people in which the diversity of facial expressions and camera angle/position can be seen.

2.Local Binary Pattern (LBP) Feature Extraction

LBP is used in various applications ranging from texture classification to object recognition thanks to its simplicity and resistance to variations in brightness. The way it works is by comparing the pixel intensities of neighboring pixels to the intensity of a central pixel in a captured frame. Each one of these pixels is given a binary value based on its intensity relative to the central pixel which forms sort of a threshold for the pixels being compared to it (similar to the concept of a detector).

$$neighboring\ pixel\ intensity > central\ pixel\ intensity \longrightarrow' 1' \ else\ '0'$$

These values are then strung together (anti clockwise) to form a binary number which describes the captured frame, the latter is then converted into a decimal value. This process is repeated over various frames forming a distribution represented by a histogram of textures.

In MATLAB there is a built-in function called "extractLBPFeatures" to which the user passes a grayscale image and is returned the extracted uniform Local Binary Pattern.

3.MATLAB Code Explanation and Results

The ORL dataset is downloaded in **pgm** format and saved into a folder in which each person has a unique label. This folder is then read into MATLAB, and the content is iterated over in order to read each image and extract its **lbpFeatures** (1 by N vector) which are saved in **LBPFeatures**. Additionally, the labels of the images in the folder are extracted and saved in **Labels**. The expected number of classes, which is 40 for 40 different people, is set and used to check the label range to ensure that the extracted labels fall in the range of 1 to 40.

```
% Path to ORL dataset
datasetPath = 'C:\Users\Tania\Desktop\Neural';
imagesList = dir(fullfile(datasetPath, '*.pgm'));
nbrImages = length(imagesList);

% Initialize
Images = {};
LBPFeatures = [];
Labels = [];

% Load images, extract LBP, and assign labels
for i = 1:nbrImages
    filename = imagesList(i).name;
```

```
    img = imread(fullfile(datasetPath, filename));
    Images{end+1} = img;

    % Extract LBP features based on a circular patter for neighbor
    % selection
    lbpFeatures = extractLBPFeatures(img, 'CellSize', [32 32], 'Radius',
 2);
    LBPFeatures = [LBPFeatures; lbpFeatures];

    % Extract subject ID from filename using regex
    tokens = regexp(filename, 'p(\d+)', 'tokens');
    if ~isempty(tokens)
        subjectID = str2double(tokens{1}{1});
        Labels = [Labels; subjectID];
    else
        disp(['No subject ID found for file: ', filename]);
    end
end

% Set expected number of classes
numClasses = 40;

% Check label range
if any(Labels < 1 | Labels > numClasses)
    error('Some labels are outside the expected range [1, %d].',
 numClasses);
end
```

Afterwards, the extracted LBPFeatures are normalized by subtracting the minimum value of the set and dividing the answer by the maximum value minus the minimum value in the set. The Labels are then encoded to obtain the Targets for each person in the dataset.

```
% Normalize LBP features
LBPFeatures = double(LBPFeatures);
LBPFeatures = (LBPFeatures - min(LBPFeatures)) ./ (max(LBPFeatures) -
 min(LBPFeatures));

% One-hot encode labels
Labels = Labels(:);
Targets = full(ind2vec(Labels'))';  % [samples x classes]
```

The normalized LBPFeatures are transposed and assigned to the variable inputs to be used as inputs for the neural network (the same process is applied to the targets).

```
% Transpose for neural network input format
inputs = LBPFeatures';     % [features x samples]
targets = Targets';        % [classes x samples]
```

The network is now ready to be created. The number of hidden layer neurons were specified with a relu activation function and patternnet was used (classification problem) to create the network with scaled conjugate gradient for the update process.

```
% Define and configure the MLP
hiddenLayerSize = 55;
net = patternnet(hiddenLayerSize, 'trainscg');  % One hidden layer, SCG
 optimizer
net.layers{1}.transferFcn = 'poslin';           % ReLU activation
net.performParam.regularization = 0.1;          % Regularization
```

The dataset splitting ratios are specified before the training so that the train function can randomly split the data and then train without the testing set. The training is performed by passing the net, the inputs and the targets to the train function which returns the splitting indices for the 3 sets (training, validation and testing).

```
% Let the network automatically divide data
net.divideParam.trainRatio = 0.7;   % 70% training
net.divideParam.valRatio = 0.15;    % 15% validation
net.divideParam.testRatio = 0.15;   % 15% testing

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

The trained network ‘s performance is then evaluated using the test inputs and test targets which are extracted based on their indices from inputs and targets. Finally, the accuracy is calculated by getting the mean of the test outputs that are equal to their test targets, and the confusion matrix is plotted using the test targets and test outputs.

```
% Evaluate performance on the test set
testInputs = inputs(:, tr.testInd);
testTargets = targets(:, tr.testInd);
testOutputs = net(testInputs);

% Convert output to class labels
trueLabels = vec2ind(testTargets);
predLabels = vec2ind(testOutputs);

% Calculate classification accuracy
accuracy = mean(predLabels == trueLabels) * 100;
fprintf('Test Accuracy (Auto Split): %.2f%%\n', accuracy);

% Plot confusion matrix
figure;
```

```
plotconfusion(testTargets, testOutputs);
title(sprintf('Confusion Matrix (Auto Split Accuracy: %.2f%%)',
 accuracy));
```

```
tr =
  struct with fields:

        trainFcn: 'trainscg'
      trainParam: [1×1 struct]
      performFcn: 'crossentropy'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 4 5 8 10 14 15 16 17 19 20 22 23 24 25 27 28 29 31 34 35 36 38 39 40 41 42 43 44 47 48 49 50 53 54 55 56 57 59 60 61 62 63 ... ] (1×280 double
          valInd: [6 7 13 21 26 32 33 51 58 80 84 87 100 104 124 125 128 131 149 150 153 160 167 176 178 188 190 194 203 204 213 217 242 244 246 247 ... ] (1×60 double
         testInd: [9 11 12 18 30 37 45 46 52 64 66 67 75 76 82 86 89 98 110 112 118 121 142 152 155 161 166 177 184 207 209 212 218 220 221 236 237 ... ] (1×60 double)
            stop: 'Training finished: Met validation criterion'
      num_epochs: 135
       trainMask: {[40×400 double]}
         valMask: {[40×400 double]}
        testMask: {[40×400 double]}
      best_epoch: 129
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 ... ] (1×136 double)
            time: [0.7090 0.7790 0.8090 0.8370 0.8540 0.8800 0.9070 0.9300 0.9550 0.9780 1.0060 1.0260 1.0500 1.0710 1.0890 1.1300 1.1590 1.1800 ... ] (1×136 double)
            perf: [0.7000 0.5554 0.4785 0.3838 0.1872 0.1323 0.1170 0.1058 0.0923 0.0819 0.0744 0.0709 0.0663 0.0625 0.0582 0.0552 0.0521 0.0474 ... ] (1×136 double)
           vperf: [0.6716 0.5543 0.4864 0.4060 0.2048 0.1496 0.1426 0.1282 0.1235 0.1119 0.1039 0.1031 0.0987 0.0953 0.0926 0.0895 0.0872 0.0822 ... ] (1×136 double)
           tperf: [0.7219 0.5974 0.5221 0.4102 0.2107 0.1450 0.1320 0.1178 0.1165 0.1049 0.0982 0.1023 0.1004 0.0933 0.0888 0.0892 0.0893 0.0862 ... ] (1×136 double)
        gradient: [0.1423 0.1529 0.1304 0.1586 0.0926 0.0285 0.0228 0.0147 0.0292 0.0197 0.0133 0.0170 0.0250 0.0179 0.0105 0.0100 0.0135 0.0130 ... ] (1×136 double)
        val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 2 3 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 2 3 0 1 2 3 4 0 0 0 0 1 2 0 0 0 0 0 ... ] (1×136 double)
       best_perf: 0.0034
      best_vperf: 0.0063
      best_tperf: 0.0059
```

Figure 2: Neural network training record (tr) in MATLAB

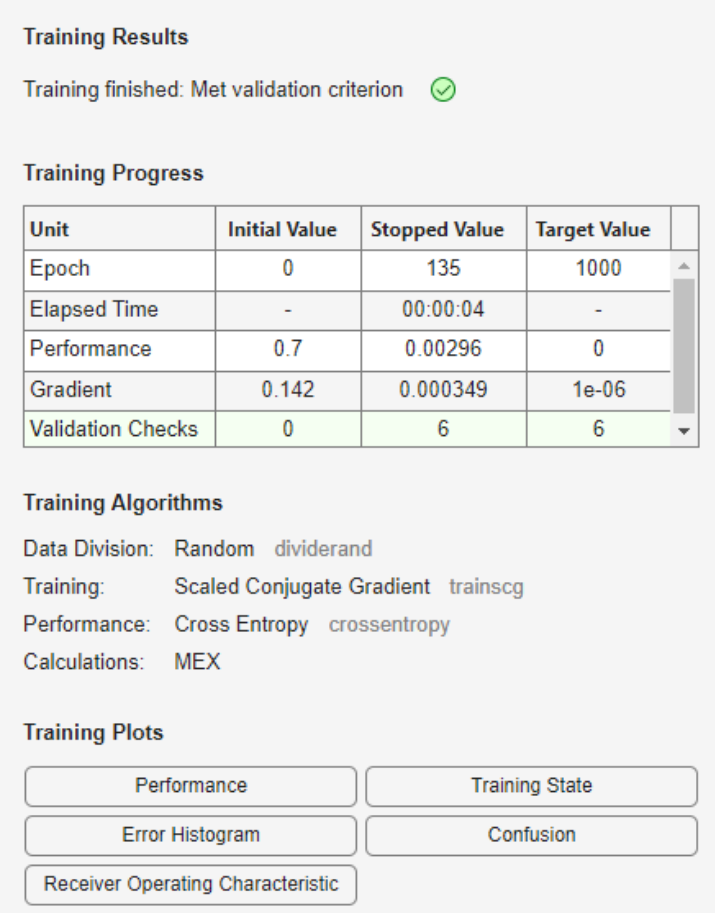

Figure 3: Network training results

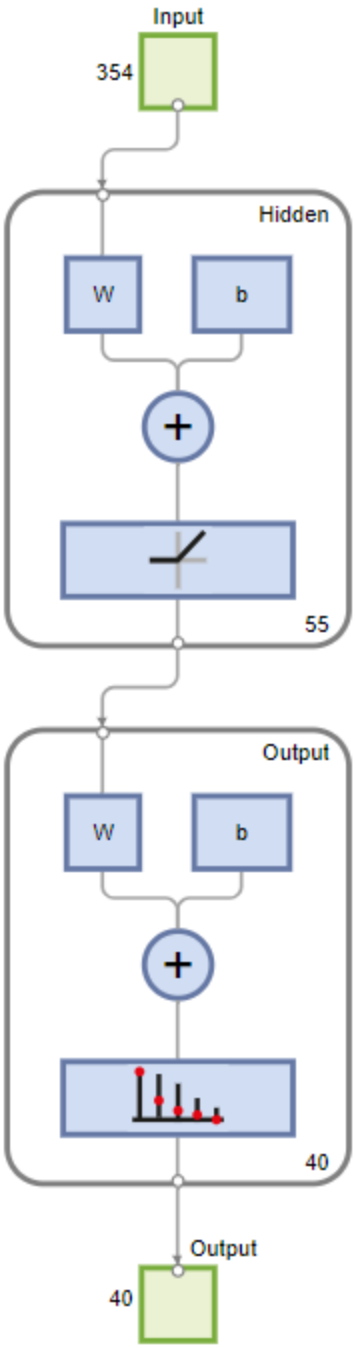

Figure 4: Network Architecture

```
Test Accuracy (Auto Split): 98.33%
```

Figure 5: Results accuracy

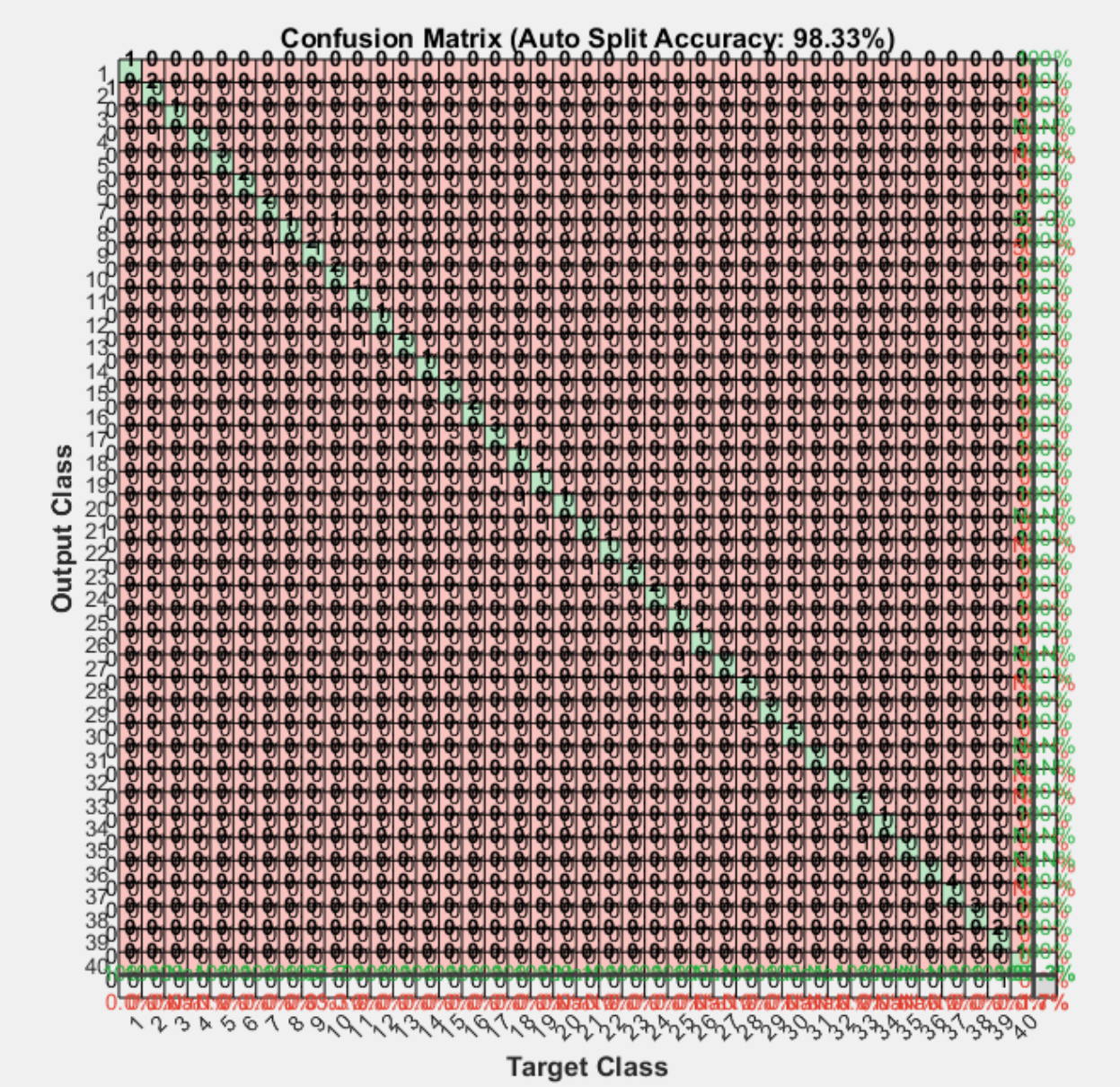

Figure 6: Confusion matrix

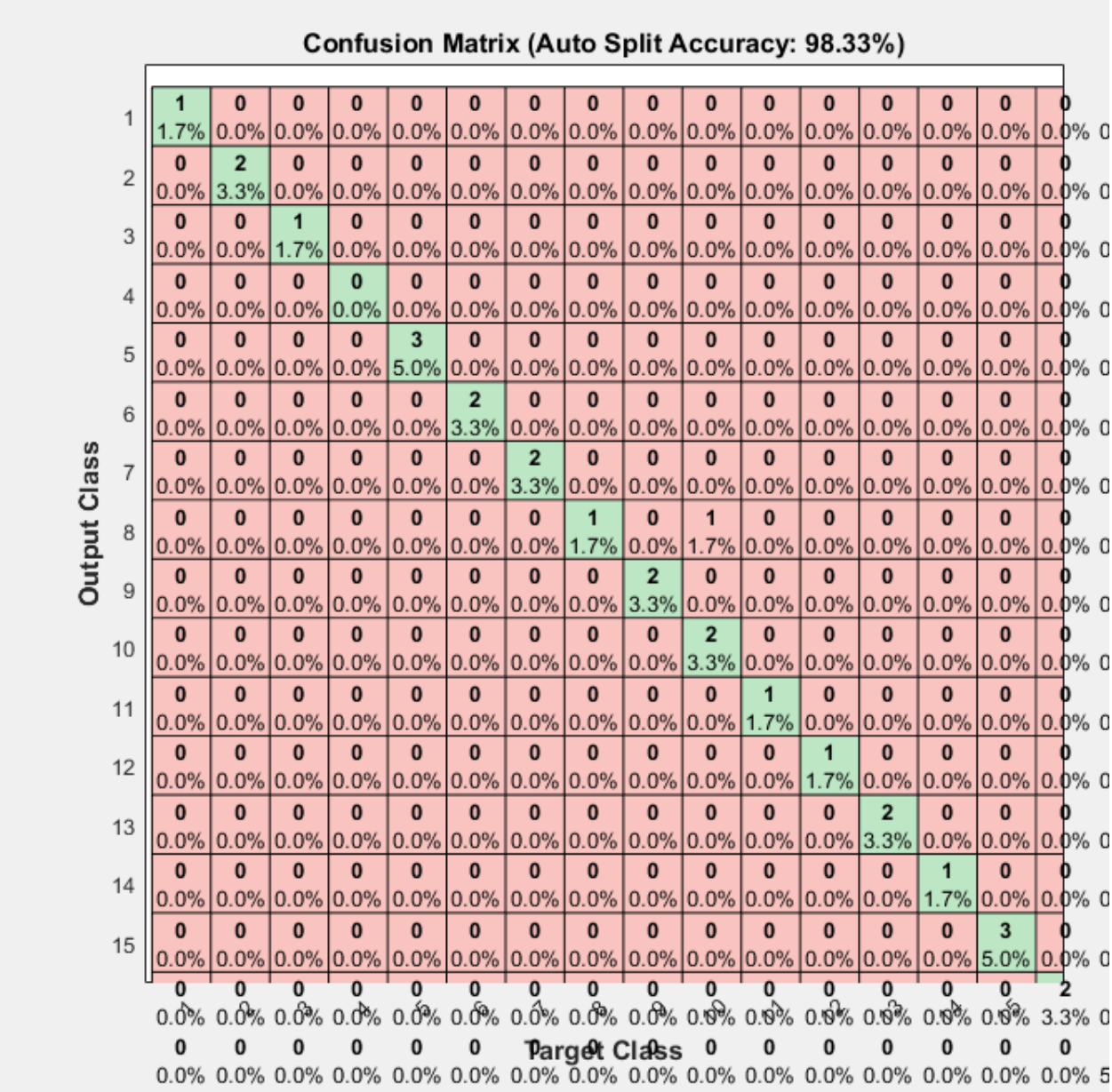

Figure 7: Confusion matrix (close-up view))

III. Conclusion

In conclusion, in this project, a face recognition system was successfully developed using Local Binary Pattern (LBP) features extracted from the ORL face dataset. These texture-based features were normalized and used to train a Multi-Layer Perceptron (MLP) neural network with a single hidden layer and ReLU activation. The system utilized automated data splitting to divide the dataset into training, validation, and test sets, allowing for a fair evaluation of model performance.

The use of the LBP descriptor provided robust facial feature representation, while the neural network achieved accurate classification of individuals across 40 subjects. The system attained a test accuracy of **98.33%**, demonstrating its effectiveness for the given dataset and feature representation method.

APPENDIX

```
clc;
clear;
close all;

% Path to ORL dataset
datasetPath = 'C:\Users\Tania\Desktop\Neural';
imagesList = dir(fullfile(datasetPath, '*.pgm'));
nbrImages = length(imagesList);

% Initialize
Images = {};
LBPFeatures = [];
Labels = [];

% Load images, extract LBP, and assign labels
for i = 1:nbrImages
    filename = imagesList(i).name;
    img = imread(fullfile(datasetPath, filename));
    Images{end+1} = img;

    % Extract LBP features based on a circular patter for neighbor
    % selection
    lbpFeatures = extractLBPFeatures(img, 'CellSize', [32 32], 'Radius',
2);
    LBPFeatures = [LBPFeatures; lbpFeatures];

    % Extract subject ID from filename using regex
    tokens = regexp(filename, 'p(\d+)', 'tokens');
    if ~isempty(tokens)
        subjectID = str2double(tokens{1}{1});
        Labels = [Labels; subjectID];
    else
        disp(['No subject ID found for file: ', filename]);
    end
end

% Set expected number of classes
numClasses = 40;

% Check label range
if any(Labels < 1 | Labels > numClasses)
    error('Some labels are outside the expected range [1, %d].',
numClasses);
end

% Normalize LBP features
LBPFeatures = double(LBPFeatures);
LBPFeatures = (LBPFeatures - min(LBPFeatures)) ./ (max(LBPFeatures) -
min(LBPFeatures));
```

```
% One-hot encode labels
Labels = Labels(:);
Targets = full(ind2vec(Labels'))';  % [samples x classes]

% Transpose for neural network input format
inputs = LBPFeatures';      % [features x samples]
targets = Targets';         % [classes x samples]

% Define and configure the MLP
hiddenLayerSize = 55;
net = patternnet(hiddenLayerSize, 'trainscg');  % One hidden layer, SCG
optimizer
net.layers{1}.transferFcn = 'poslin';           % ReLU activation
net.performParam.regularization = 0.1;          % Regularization

% Let the network automatically divide data
net.divideParam.trainRatio = 0.7;   % 70% training
net.divideParam.valRatio = 0.15;    % 15% validation
net.divideParam.testRatio = 0.15;   % 15% testing

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

% Evaluate performance on the test set
testInputs = inputs(:, tr.testInd);
testTargets = targets(:, tr.testInd);
testOutputs = net(testInputs);

% Convert output to class labels
trueLabels = vec2ind(testTargets);
predLabels = vec2ind(testOutputs);

% Calculate classification accuracy
accuracy = mean(predLabels == trueLabels) * 100;
fprintf('Test Accuracy (Auto Split): %.2f%%\n', accuracy);

% Plot confusion matrix
figure;
plotconfusion(testTargets, testOutputs);
title(sprintf('Confusion Matrix (Auto Split Accuracy: %.2f%%)',
accuracy));
```