ECE661: Homework 8

Fall 2016
Vinoth Venkatesan
venkat26@purdue.edu

November 7, 2016

Task 1.

***Problem description*:** Implement the Local Binary Pattern (LBP) algorithm for texture characterization of images for the given classes of training images to obtain the texture histograms. Provide labels to histograms of pictures in the same class.

***Solution*:**

The texture characterization of images can be done using various methods like GLCM (Gray Level Co-occurence Matrix) and LBP (Local Binary Pattern). The implementation of LBP is explained below:

- **Finding coordinates and intensities of neighboring points:**

  The neighbors of the pixel under construction is calculated using the change in distance along two orthogonal axis as:

$$(\Delta u, \Delta v) = \left(Rcos(\frac{2\pi p}{P}), Rsin(\frac{2\pi p}{P}) \right) \quad p = 0, 1, 2, .., 7$$

  Here, the neighbors are found in a unit circle around the center pixel. The number of points is set by setting the value of $P = 8$. After finding the position of these pixels, we need to find the intensity of these point locations.

  - If these points coincide with an existing pixel value, then the intensity of that pixel value is set to that point.
  - If the point does not coincide with any pixel, we can use an interpolation technique. Here, a weighted mean of the intensity values of the four surrounding pixels was used to find the value of the current pixel as follows (here the weights are the inverse of the euclidean distances from the point):

$$I_i = \frac{I_{1i}w_1 + I_{2i}w_2 + I_{3i}w_3 + I_{4i}w_4}{w_1 + w_2 + w_3 + w_4}$$

- **Representing the intensities using a minimum value pattern**

  In order to get the binary pattern which will act as a representation of these values, we use the following logic:

  - If the intensity value is greater than or equal to the value at the center, we give it a label 1
  - If the value is lesser, we give it a label 0.

  This way, we get the binary pattern for the each pixel in the image. Now, in order to make the algorithm robust enough to reject in-plane rotations, we rotate the obtained binary pattern one bit at a time and keep doing it until we get the maximum number of zeros in the most significant bit (MSB) positions. This can done as follows:

  - We rotate the pattern one bit at a time and find the unsigned integer value corresponding to that pattern.
  - We use the pattern which has the least integer value and this pattern will represent the local texture of the pixel.

- **Encoding the binary pattern values:**

  After we have acquired the binary pattern, we encode or represent these texture patterns based on the number of runs of 1's and 0's as follows:

  - If there are exactly two runs, a run of 0's followed by 1's, represent the pattern by the number of 1's in the second run.
  - If the pattern has all 0's, then represent it with 0.
  - If the pattern has all 1's, represent it with the value $P(= 8)$.
  - If the pattern has more than two runs, encode it with the value of P.

  Once we have encoded the pixel, we update the histogram for the image which has 8-bins (equal to the value of P). This histogram then represents the image and will be used to compare with the test images to determine the label for the test image.

The LBP algorithm is implemented for all the training images in the four classes and each of these histograms are associated with a label depending on the class they belong to.

Task 2.

***Problem description*:** Using the NN-Classifier, we find the label or the class that each of the test images belong to.

***Solution*:**

The NN-Classifier uses the euclidean distance measure to classify the test images. The flow is explained as follows:

- For each of the test images, find the histogram texture for each of them using the LBP algorithm.
- Using Euclidean distance metric (2 - norm), we find the k-nearest neighbors by finding the metric value with each of the histograms of the training images. Here, the value of k was set to 5.
- We assign the corresponding label of the training image to the test image and hence we end up with k-labels. Out of these, we choose the label which occurs the most number of times and assign it to the test image.

Task 3.

***Problem description*:** Based on the results obtained using the k-nearest neighbor algorithm, we build the confusion matrix to study the performance of the algorithm.

***Solution*:**

The confusion matrix is built based on the classifications done for the test images. Hence, there are four rows and four columns (since we have four classes). The rows correspond to the actual labels and the columns correspond to the predicted labels. The confusion matrix obtained for the implementation is shown below:

	Car	**Building**	**Mountain**	**Tree**
Car	3	2	0	0
Building	0	4	1	0
Mountain	0	1	4	0
Tree	1	1	0	3

The accuracy of the algorithm was calculated to be 70%. The metric that was used is as follows:

$$Accuracy = \frac{Correct\_Classifications}{Total\_Number\_of\_Classifications}$$

Observations on performance of the ICP algorithm:

- The image recognition algorithm developed based on LBP and the NN-Classifier performed comparatively well given the accuracy of 70%.
- However, there were some wrong classifications and this could due to a lot reasons like:
    - The number of nearest neighbors which we considered were too small. This is a crucial one because it was observed that in a lot of cases, there were equal number of wrong neighbors and correct neighbors it is difficult to automate the decision making process in these cases. Hence, increasing the number k might result in a better performance.
    - The resolution of the pictures in both the training and test images were small and this could have resulted in wrong classifications.
    - The encoding algorithm which we used seems to be biased towards the last condition (where there are more than two runs) which was apparent from the histograms. If we use a better encoding technique, then we might get better results.

Results:

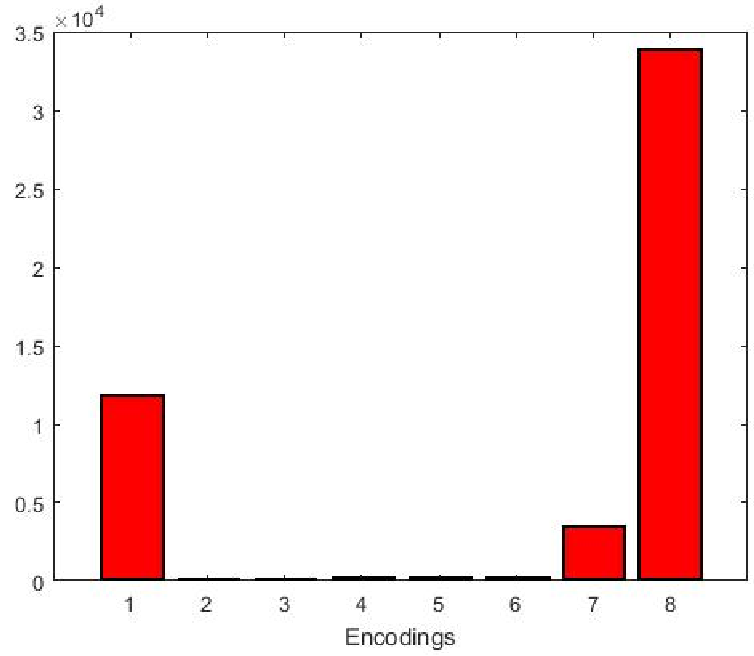

Figure 1: LBP histogram for a car image

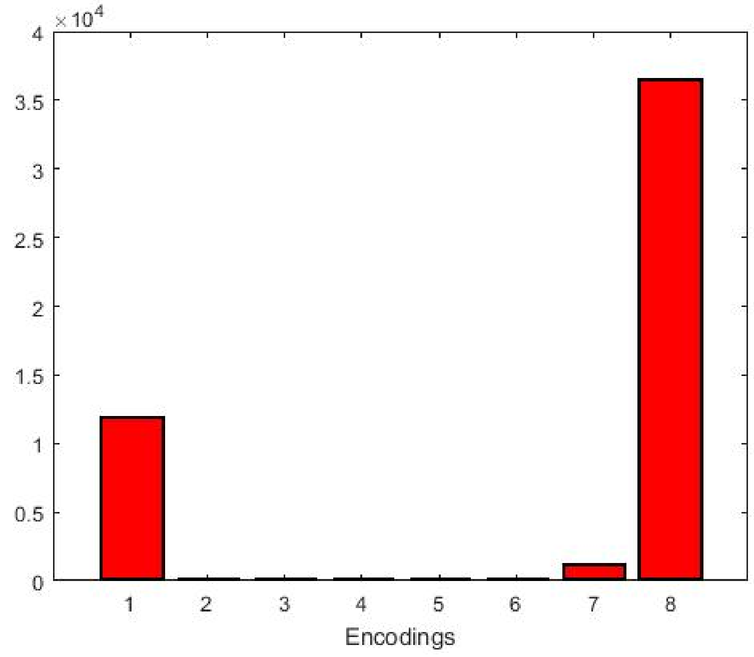

Figure 2: LBP histogram for a building image

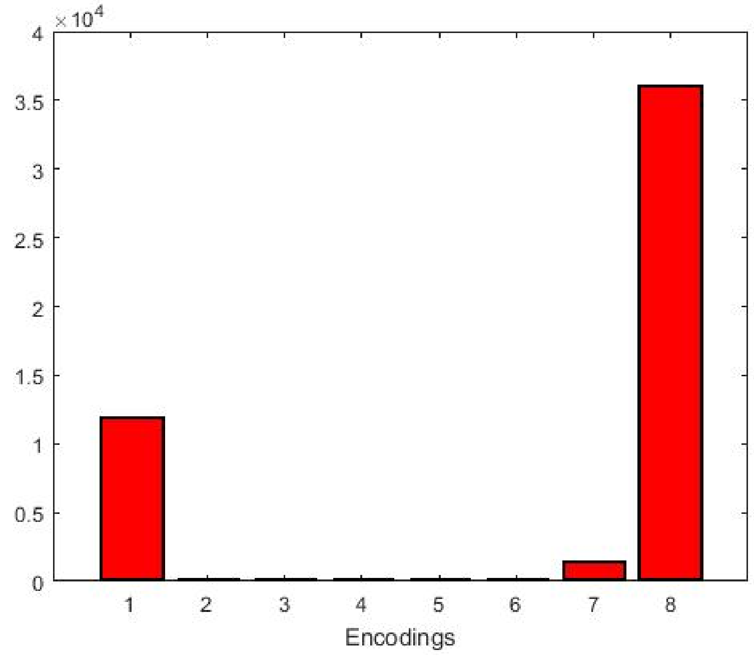

Figure 3: LBP histogram for a mountain image

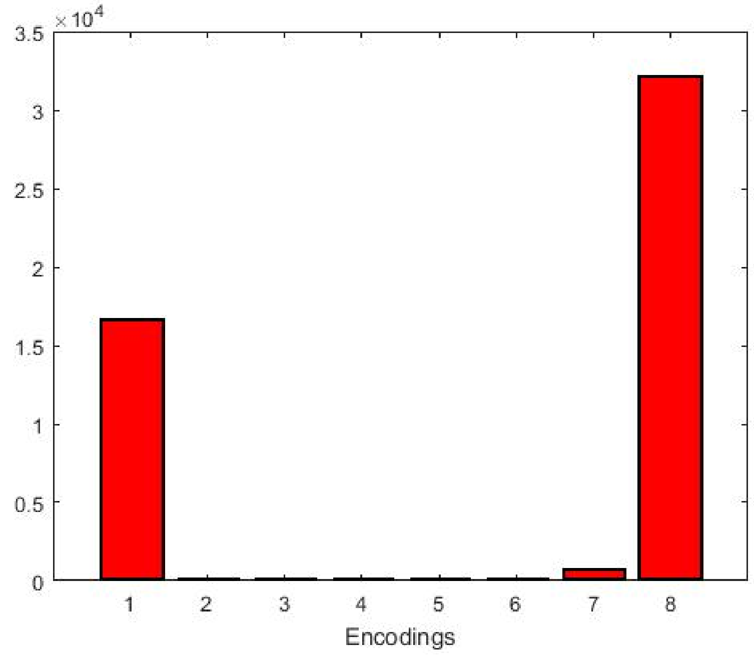

Figure 4: LBP histogram for a tree image

Source Code(Main):

```
clear all
close all
clc

for i = 1:20

    % Load i-th image in the training set
    str = sprintf('01_car (%d).jpg', i);
    A = imread(str);
    % Convert to grayscale
    A = rgb2gray(A);
    % Call to LBP_texture to get the texture histogram
    histogram_car(i,:) = LBP_texture(A);

    str = sprintf('01_building (%d).jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_building(i,:) = LBP_texture(A);

    str = sprintf('01_mountain (%d).jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_mountain(i,:) = LBP_texture(A);

    str = sprintf('01_tree (%d).jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_tree(i,:) = LBP_texture(A);

end

% Saving the training results
save('training_results_LBP');
```

LBP_texture

```

function histogram = LBP_texture(A)

R = 1; % Radius for examination
P = 8; % Number of sampled
histogram = zeros(1,8);

for i = 2:size(A,1)-1
    for j = 2:size(A,2) - 1

    % Pixel coordinates of the neighbours
    neighbours = [i + R*(cos((2*pi*(0:7))/P)); j + R*(sin((2*pi
       *(0:7))/P))];
    intensity = zeros(1,8);

    % Finding ther intensity of the neighbouring pixels
    for k = 1:size(neighbours,2)
        if (mod(k,2))
            % When the point coincides with a pixel coordinate
            if(A(round(neighbours(1,k)),round(neighbours(2,k)))
               >= A(i,j))
                intensity(k) = 1;
            else
                intensity(k) = 0;
            end
        else
            % Performing weighted mean to find the intensity
            if (weighted_mean(neighbours(2,k),neighbours(1,k),A)
                >= A(i,j))
                intensity(k) = 1;
            else
                intensity(k) = 0;
            end
        end
    end

    % Initializations
    final_num = intensity;
    bin_num = intensity;
    final_decimal = num2str(intensity);
    final_decimal(isspace(final_decimal)) = '';
    final_decimal = bin2dec(final_decimal);

    % Loop to find the minimum value pattern
    for k = 1:8
        % Rotating one bit at a time
        bin_num = circshift(bin_num,1,2);
        str_x = num2str(bin_num);
        str_x(isspace(str_x)) = '';
        bin_num = bin2dec(str_x);

        % Finding the minimum value
        if bin_num < final_decimal
            final_decimal = bin_num;
            final_num = de2bi(bin_num,8,'left-msb');
        end

```

```
            bin_num = de2bi(bin_num,8,'left-msb');
        end

        % Initialize runs to zero
        runs = 0;

        % Loop to find the number of runs
        for k = 1:7
            if final_num(k + 1) ~= final_num(k)
                runs = runs + 1;
            end
        end

        % Updating the histogram vector based on the number of runs
        if ((runs == 0) && (final_num(1) == 1)) || runs > 1
            histogram(8) = histogram(8) + 1;
        elseif (runs == 0) && (final_num(1) == 0)
            histogram(1) = histogram(1) + 1;
        else
            temp = nnz(final_num == 1);
            histogram(temp) = histogram(temp) + 1;
        end

    end
end


end
```

main_testing.m

```
clear all
close all
clc

% Load the training results
load training_results_LBP
histogram_all = cat(1,histogram_car,histogram_building,
    histogram_mountain,histogram_tree);

for i = 1:5

    % Loading the i-th test image
    str = sprintf('car_%d.jpg', i);
    A = imread(str);
    % Convert to grayscale
    A = rgb2gray(A);
    % Finding the histogram of the test image
    histogram_car_test(i,:) = LBP_texture(A);
    % Classification based on the NN-Classifier
    label_test_car(i) = NN_classifier(histogram_car_test(i,:),
       histogram_all);

    str = sprintf('building_%d.jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_building_test(i,:) = LBP_texture(A);
    label_test_building(i) = NN_classifier(histogram_building_test(i
       ,:),histogram_all);

    str = sprintf('mountain_%d.jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_mountain_test(i,:) = LBP_texture(A);
    label_test_mountain(i) = NN_classifier(histogram_mountain_test(i
       ,:),histogram_all);

    str = sprintf('tree_%d.jpg', i);
    A = imread(str);
    A = rgb2gray(A);
    histogram_tree_test(i,:) = LBP_texture(A);
    label_test_tree(i) = NN_classifier(histogram_tree_test(i,:),
       histogram_all);

end

% Confusion matrix development
confusion_matrix = [nnz(label_test_car == 1)    nnz(label_test_car
    == 2)    nnz(label_test_car == 3)    nnz(label_test_car == 4)
    nnz(label_test_building == 1)    nnz(label_test_building == 2)
          nnz(label_test_building == 3)    nnz(label_test_building
       == 4)
    nnz(label_test_mountain == 1)    nnz(label_test_mountain == 2)
          nnz(label_test_mountain == 3)    nnz(label_test_mountain
       == 4)
    nnz(label_test_tree == 1)    nnz(label_test_tree == 2)    nnz(
       label_test_tree == 3)    nnz(label_test_tree == 4)];

% Accuracy = correct_classifications/total_number_of_classifications
Accuracy = trace(confusion_matrix)/sum(sum(confusion_matrix));
```

NN_classifier.m

```
function label = NN_classifier(histogram_test, histogram_all)

% Initialize the distance matrix
```

```
euclidean_dist = zeros(1,size(histogram_all,1));

% Finding the euclidean distance
for i = 1:size(histogram_all,1)
    euclidean_dist(i) = norm(histogram_test - histogram_all(i,:));
end

% Finding the k-nearest neighbours (for 5 neighbours)
[minimum, indices] = sort(euclidean_dist);
labels = zeros(1,5);

for i = 1:5
    if indices(i) <= 20
        labels(i) = 1;
    elseif indices(i) <= 40
        labels(i) = 2;
    elseif indices(i) <= 60
        labels(i) = 3;
    else
        labels(i) = 4;
    end
end

% Finding the label for the test image
temp = [nnz(labels == 1) nnz(labels == 2) nnz(labels == 3) nnz(
   labels == 4)];
% label that appears the most
[maximum, label] = max(temp);

end
```

weighted_mean.m

```
function X = weighted_mean(x,y,B)

    % Finding the surrounding pixels
    x1 = floor(x);
    x2 = ceil(x);
    y1 = floor(y);
    y2 = ceil(y);

    % Finding the weights (euclidean distance)
    dist1 = norm([x - x1, y - y1]);
    dist2 = norm([x - x2, y - y1]);
    dist3 = norm([x - x1, y - y2]);
    dist4 = norm([x - x2, y - y2]);

    y_act1 = size(B,1) - y1 + 1;
    y_act2 = size(B,1) - y2 + 1;
    x_act1 = x1;
    x_act2 = x2;

    % Calculating the intensity value using weighted mean
    X = (B(y_act1,x_act1,:)./dist1 + B(y_act1,x_act2,:)./dist2 + B(
       y_act2,x_act1,:)./dist3 + B(y_act2,x_act2,:)./dist4)...
        ./(1/dist1 + 1/dist2 + 1/dist3 + 1/dist4);

end
```