COMPUTER VISION

ECE 661

HOMEWORK 7

Tanzeel U. Rehman

Email: *trehman@purdue.edu*

0. Theory Questions:

Q: The reading material for Lecture 15 presents three different approaches to characterizing the texture in an image: 1) using the Grayscale Co-Occurrence Matrix (GLCM); 2) with Local Binary Pattern (LBP) histograms; and 3) using a Gabor Filter Family. Explain succinctly the core ideas in each of these three methods for measuring texture in images. (You are not expected to write more than a dozen sentences on each).

Ans: The GLCM characterize the texture by estimating the joint probability distribution P(i,j) for grayscale values of two neighboring pixels separated by distance vector *d* with first one having gray value of "*i*" and the other one being "*j*". The final shape of the GLCM matrix will be $G \times G$, where *G* represents the grayscale values in a gray image. After estimating the joint probability distribution, the texture can be characterized by the shape of GLCM matrix. The matrix needs to be normalized by dividing its individual elements with sum of all elements. The matrix is usually developed with multiple distance vectors. After developing the GLCM, texture is usually characterized through a set of features such as entropy, energy, contrast and homogeneity. These features can then be used with different classification algorithms to perform the classification task. For details and interpretation of these features, see the tutorial material provided as a part of class lectures [1)].

The LBP algorithm extracted the texture by characterizing the local grayscale variations around pixels through binary pattern runs of 0s and 1s. These runs were obtained by defining a circular neighborhood controlled by the radius *R* and the number of pixels *P* on the circle. The position of points lying on the neighborhood circle was computed (Eq. 1) and bilinear interpolation was performed to estimate the gray values for neighbors whose coordinates do not coincide with the center of circle. These gray values in the neighborhood were thresholded with respect to the value at the center pixel to obtain the binary patterns. These local binary pattern was made rotationally invariant by circularly rotating a binary pattern until it acquires the smallest integer value. Finally, the runs were achieved in the range of 0 to P+2 (see details in the section A) and were encoded to develop LBP histograms. These histograms represent the features and are invariant to rotation and any to any monotonic transformations of the grayscale.

Q: With regard to representing color in images, answer Right or Wrong for the following questions:

(a) RGB and HSI are just linear variants of each other.

Ans: Wrong.

(b) The color space L*a*b* is a nonlinear model of color perception.

Ans: Right**.**

(c) Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.

Ans: Right.

1. Introduction:

This homework performs image classification on 5 different classes using the textural features extracted from local binary pattern (LBP) algorithm combined with the nearest neighbor classifier. The performance of the classification task on the test data was reported in terms of confusion matrix and overall classification accuracy.

1.1. Description of Methods:

A. LBP feature extraction:

The LBP algorithm extracted the texture by characterizing the local grayscale variations around pixels through run of 0s and 1s. The LBP algorithm was implemented based on Dr. Avi Kak's tutorial [1)] the following steps:

1) For each pixel location in the grayscale image, we defined a circular neighborhood controlled by the radius R and the number of pixels P on the circle.

2) The position of points lying on the neighborhood circle was computed using Eq. 1.

$$(\Delta u, \Delta v) = \left(Rcos\left(\frac{2\pi p}{P}\right), Rsin\left(\frac{2\pi p}{P}\right)\right) \qquad \text{(Eq.1)}$$

Where, $p = 0, 1, 2, \ldots, P\text{-}1$ and the position coordinates displacement along x and y-axis from the pixel coordinates of the center pixel. While $p = 0$ corresponds to the bottom neighbor, $p = 2$ corresponds to the right neighbor with respect to the center pixel, and so on.

3) For the neighbors, whose coordinates do not coincide with the center of pixel coordinates in the image, we performed bilinear interpolation to compute the grayscale value (Eq. 2).

$$image\ (x) = (1 - \Delta u)(1 - \Delta u)A + \ (1 - \Delta u)\Delta vB + \Delta u(1 - \Delta v)C + \Delta u\Delta vD \qquad \text{(Eq.2)}$$

Where, A, B, C and D are centers of four adjoining pixels representing a rectangle around the pixel location x. The bilinear interpolation assumes that the inter-pixel sampling interval is a unit distance along X and Y directions.

4) In the next step, we thresholded the grayscale values of the P neighbors with respect to the grayscale value at the center pixel. Any neighbor having the grayscale value greater than center pixel was set to 1, otherwise to 0. This helped us to obtain a local binary pattern around each pixel.

5) The local binary pattern was made rotationally invariant by circularly rotating a binary pattern until it acquires the smallest integer value. This homework uses **BitVector** module implemented by Dr. Avi Kak to get the pattern with minimum integer value [2)].

6) Next, we encoded the rotationally invariant binary patterns around each pixel location based on the number of run to an integer ranging from 0 to P + 2 using the following conditional scheme:

   - If the patterns is all 0s, encode = 0.
   - If the patterns has all 1s, encode = 1.
   - If the pattern has only two runs (a run of 0s followed by a run of 1s), encoding = number of 1s in pattern.
   - If the pattern has more than 2 runs, encoding = P + 1.

The encoded pixels were used to compute the histogram having $P + 2$ bins. These histogram features characterized the texture and were used with k-nearest neighbor classifier.

B. K-Nearest Neighbor Classifier:

K-nearest neighbor (KNN) performs the classification by using the LBP features and classification labels. The KNN algorithm keeps a copy of the training features along with labels in order to make predictions of class labels of test data. In this homework, we used Euclidean distance to compare the LBP features (histograms) of an instance of the test dataset to the LBP features of the training dataset. This comparison helped us to identify the closest *k* training samples (k = 5) to an instance of the test dataset. Finally, we assigned a label to the instance of test dataset based on the labels of closest (having smallest Euclidean distance) *k* training samples.

1.2. Observations:

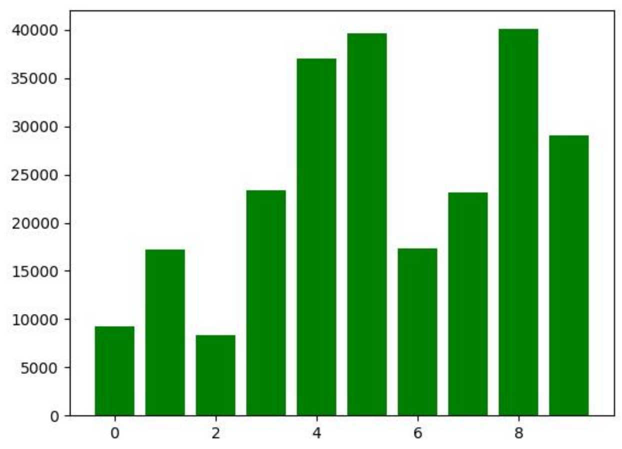
a) It can be observed that the LBP histograms of different classes have considerable difference (Fig 1).

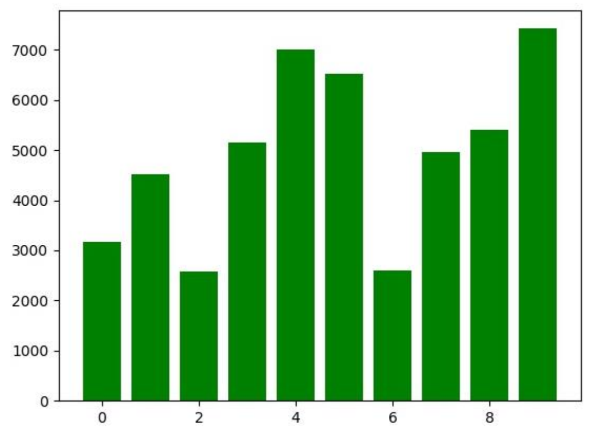
b) On an overall, the LBP features with KNN classifier achieved an accuracy of 68% on the test dataset (Fig 2). Given the fact that, KNN is very simple linear classification algorithm, I believe that this is relatively decent accuracy.

c) The best classification accuracy was obtained for the beach images, which could be because images in this have relatively higher spatial resolution.

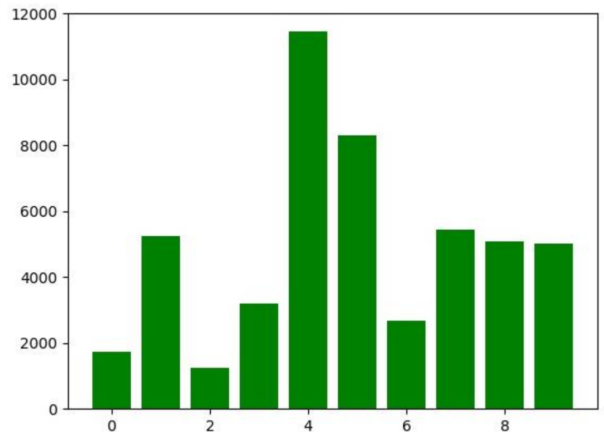
d) Class building and mountain have some confounding, and this could probably be due to probably relative similarity in the textures of these classes.

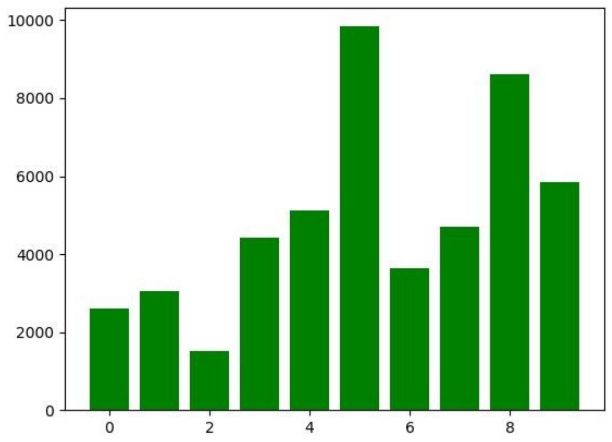
3. Results:



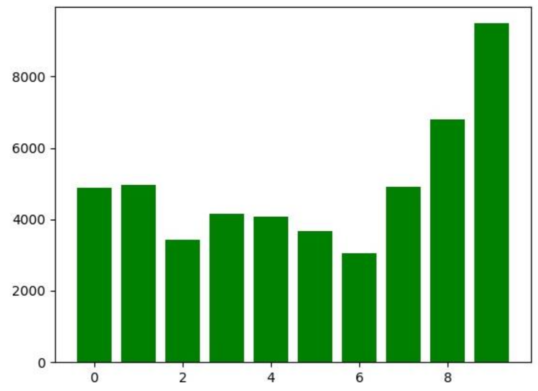


(a) (b)





(c) (d)



(e)

Fig 1: LBP histograms of the instances randomly picked from training and test data of all classes, (a) beach image, (b) building image, (c) car image, (d) mountain image and (e) tree image.

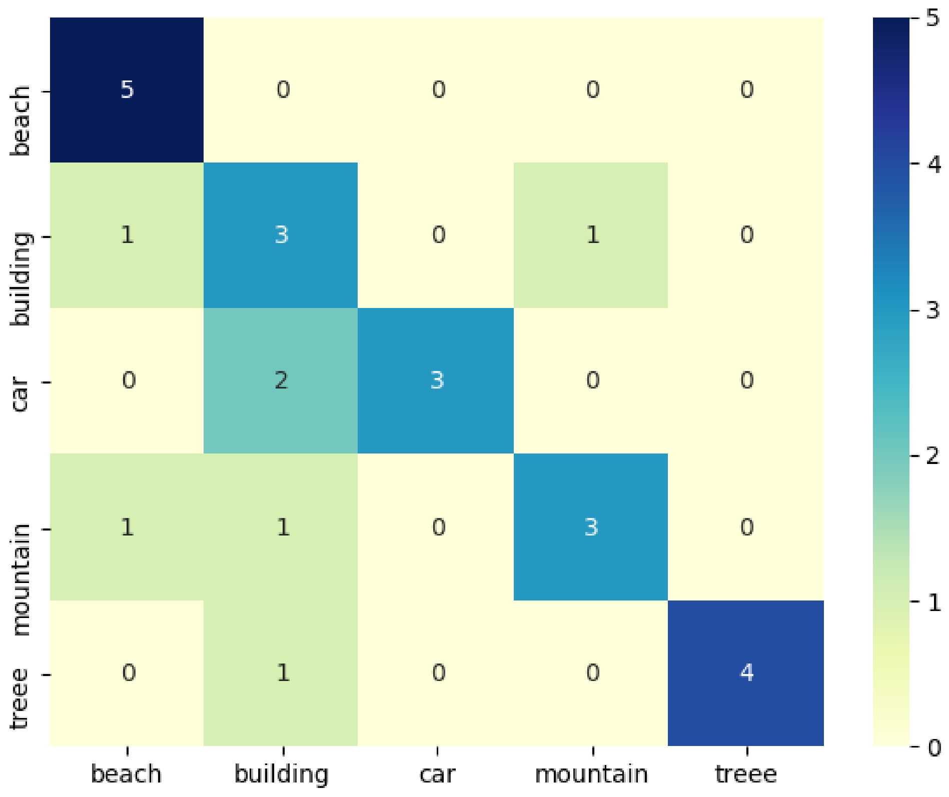


Fig 2: Confusion matrix of test dataset obtained with LBP features and the kNN classifier with k = 5

4. References:

1) Avinash Kak. Measuring Texture and Color in Images accessed on October 19, 2020 available at https://engineering.purdue.edu/kak/Tutorials/TextureAndColor.pdf.

2) Avinash Kak. Python module named Bitvector accessed on October 19, 2020 available at https://engineering.purdue.edu/kak/dist/BitVector-3.4.9.html.

5. Source Code:

5.1.Function calls for classification:

```
'''----------Main Code for training and testing a classifier based on LBP and KNN-------'''
n_train_imgs = 20
n_claases = 5
n_test_imgs = 5

R = 1
P = 8

#k Nearest-Neighbor Classifier Parameters
K_NN = 5

# Collect the LBP features of entire training data
print("Training the Classifier Begins")
Beach_features = Class_LBP_features('beach', P, R,n_train_imgs,True)
Building_features = Class_LBP_features('building',P, R,n_train_imgs,True)
Car_features = Class_LBP_features('car', P, R,n_train_imgs,True)
Mountain_features = Class_LBP_features('mountain', P, R,n_train_imgs,True)
Tree_features = Class_LBP_features('tree', P, R,n_train_imgs,True)

#Combine all these histogram features into a 2d array with first column being the index of calss
hist_train                                                                                    =
np.concatenate((Beach_features,Building_features,Car_features,Mountain_features,Tree_feature
s),axis=0)
idx = np.repeat(np.arange(0,n_claases,1),n_train_imgs)
hist_train = np.insert(hist_train,0,idx,axis=1)

# Working on the test images:
# Get the LBP features of entire test
Beach_features_test = Class_LBP_features('beach',P, R,n_test_imgs,False)
Building_features_test = Class_LBP_features('building',P, R,n_test_imgs,False)
Car_features_test = Class_LBP_features('car',P, R,n_test_imgs,False)
Mountian_features_test = Class_LBP_features('mountain',P, R,n_test_imgs,False)
Tree_features_test = Class_LBP_features('tree',P, R,n_test_imgs,False)

# Create an empty confusion table to hold the final results
Confusion_Table = np.zeros((n_claases, n_claases))
#Count the correctly classified instances and fill the confusion table
unique_idx,counter                                                                            =
PredictAndCount(Beach_features_test,hist_train,n_test_imgs,n_train_imgs,K_NN)
Confusion_Table[0,unique_idx] = counter
unique_idx,counter                                                                            =
PredictAndCount(Building_features_test,hist_train,n_test_imgs,n_train_imgs,K_NN)
Confusion_Table[1,unique_idx] = counter
unique_idx,counter                                                                            =
PredictAndCount(Car_features_test,hist_train,n_test_imgs,n_train_imgs,K_NN)
```

```
Confusion_Table[2,unique_idx] = counter
unique_idx,counter                                                    =
PredictAndCount(Mountian_features_test,hist_train,n_test_imgs,n_train_imgs,K_NN)
Confusion_Table[3,unique_idx] = counter
unique_idx,counter                                                    =
PredictAndCount(Tree_features_test,hist_train,n_test_imgs,n_train_imgs,K_NN)
Confusion_Table[4,unique_idx] = counter
#Show the confusion Table as a colored heatmap
df_cm     =     pd.DataFrame(Confusion_Table,     index     =     [i     for     i     in
["beach","building","car","mountain","treee"]],
                columns = [i for i in ["beach","building","car","mountain","treee"]])
sn.heatmap(df_cm, annot=True,cmap="YlGnBu")
```

A. Different functions for extracting the LBP features and performing the KNN classification:

```
import numpy as np
import cv2
import glob
import BitVector
import seaborn as sn
import pandas as pd
import pickle
import os.path
from collections import Counter
import matplotlib.pyplot as plt

def Neighboorhood_position(P = 8, radius= 1):
   '''
   Function for computing the circular position of the neighboring pixels with reference to
   center pixel.
   '''
   u,v = [],[]
   for p in range(P):
      du = radius * np.cos(2*np.pi*p/P)
      dv = radius * np.sin(2*np.pi*p/P)
      # du and dv are not yielded as exact zero, therefore change to zero
      if abs(du) < 1e-3:
         du = 0.0
      if abs(dv) < 1e-3:
         dv = 0.0
      u.append(du),v.append(dv)
   return u,v

def Interpolation_Bilinear(frame, dx, dy, x0, y0):
   #skip interpolation on 2,1/1,2,0,1/1,0
   if dx == 1.0 or dx == 0.0 or dx == -1.0 or dx ==-0.0:
        return float(frame[x0][y0])
    else:
        A = frame[x0][y0]
        B = frame[x0+1][y0]
        C = frame[x0][y0+1]
        D = frame[x0+1][y0+1]
        return (1-dx)*(1-dy)*A + dx*(1-dy)*B + (1-dx)*dy*C  + dx*dy*D

def LBP_pattern(frame,P,radius):
    '''
    Function for computing the binary vector from a provided window of size controlled by
Radius
    '''
    pattern_vc=[]
    du, dv = Neighboorhood_position(P,radius)

    for p in range(P):
        dx = du[p]-np.floor(du[p])
        dy = dv[p]-np.floor(dv[p])
        pattern_vc.append(Interpolation_Bilinear(frame, dx, dy, int(1+du[p]), int(1+dv[p])))
        #print(dx,dy)
    pattern_vc = np.array(pattern_vc)
    pattern_vc = pattern_vc >= frame[1,1]

    return pattern_vc
def minIntVal(pattren_vec, P):
    '''
    Function for acheiveing the smallest integer value upon rotation of binary pattern.
    This is done to acheive rotation invariance. This function uses the module written by
    Dr. Avi Kak from Purdue University. https://engineering.purdue.edu/kak/dist/BitVector-
3.4.9.html
    '''
    bv =  BitVector.BitVector( bitlist = pattren_vec)
    ints  =  [int(bv << 1) for _ in range(P)]
    minbv = BitVector.BitVector( intVal = min(ints), size = P )
    return minbv.runs()

def LBP_hist(Gray_img, P = 8, R = 1):
    '''
    Function for creating the LBP histograms
    '''
    h, w = Gray_img.shape[0]-R, Gray_img.shape[1]-R
    hist = [0]*(P+2)
    for i in range(R,h):
        for j in range(R,w):
            # Create a window of size 3x3
            window = Gray_img[i-1:i+2,j-1:j+2]
            pattren_vec = LBP_pattern(window,P, R) #Generate the pattern vector
            minruns = minIntVal(pattren_vec, P) # Find the min values agains patten vec
            # Populate thy histogram
            if len(minruns) > 2:
                hist[P+1] += 1
            elif len(minruns) == 1 and minruns[0][0] == '1':
                hist[P] += 1
            elif len(minruns) == 1 and minruns[0][0] == '0':
                hist[0] += 1
            else:
                hist[len(minruns[1])] +=1
    return np.array(hist)


def Class_LBP_features(img_class,P,R,num_images,Train_Flag=True):
    histograms = []
    if Train_Flag ==True:
        st = 'Training'
        print ('Extracting LBP features from training data for class: {}'.format(img_class))
        images = glob.glob('../imagesDatabaseHW7/training/{}'.format(img_class) + '/*.jpg')
    else:
        st = 'Testing'
        print ('Extracting LBP features from Testing data for class: {}'.format(img_class))
        images = glob.glob('../imagesDatabaseHW7/testing/{}'.format(img_class) + '_*.jpg')

    for i in range(num_images):
        image_color = cv2.imread(images[i])
        image_gray = cv2.cvtColor(image_color,cv2.COLOR_BGR2GRAY)
        hist = LBP_hist(image_gray,P, R)
        histograms.append(hist)

    #Make a plot of last image in the class sequence for showing in report
    plt.bar(range(len(hist)), hist, color='g')
    plt.savefig('LBPHist_Class_{}_'.format(img_class) + '{}_'.format(int(i)) + '{}_'.format(st)
+ '.jpg')
    plt.close()

    return np.array(histograms)

def Class_LBP_Read_Write(filename,Class_features,img_class,Train_Flag=True):
    #if training write the LBP features
    if Train_Flag ==True:
        handle = open(filename +'.obj',"wb")
        pickle.dump(Class_features,handle)
        handle.close()
    else:
        if not os.path.exists(filename +'.obj'):
            print("The Image Classifier has not been trained on {} Images".format(img_class))
            exit()
        handle = open(filename +'.obj',"rb")
        hist_arr = pickle.load(handle)
        handle.close()
        return hist_arr

def NearestNeighborClassifier(LBP_Hist_test, LBP_Hist_train, n_test_imgs, nTrainImgs,
KNN):
    #Find the Eucledian distance between LBP features of test image with that of training
images
    n_train_instances = LBP_Hist_train.shape[0]
    test_img_labels = np.zeros((n_test_imgs,KNN),dtype='int')
    Euclidean_dist = np.zeros((n_test_imgs,n_train_instances))
    label = np.zeros(n_test_imgs,dtype='int')

    for i in range(n_test_imgs):
        for j in range(n_train_instances):
            Euclidean_dist[i,j] = np.linalg.norm(LBP_Hist_test[i,:]-LBP_Hist_train[j,1:])
        idx = np.argsort(Euclidean_dist[i,:])

        for k_idx in range(KNN):
            if(idx[k_idx]<(nTrainImgs*1)):
                test_img_labels[i, k_idx] = 0
            elif (idx[k_idx]<(nTrainImgs*2)):
                test_img_labels[i, k_idx] = 1
            elif (idx[k_idx]<(nTrainImgs*3)):
                test_img_labels[i, k_idx] = 2
            elif (idx[k_idx]<(nTrainImgs*4)):
                test_img_labels[i, k_idx] = 3
            elif (idx[k_idx]<(nTrainImgs*5)):
                test_img_labels[i, k_idx] = 4

        label[i],freq = Counter(list(test_img_labels[i,:])).most_common(1)[0]

    return label

def PredictAndCount(test_features,hist_train,n_test_imgs,n_train_imgs,k_nn):
        #Classify A set of test images of specific class against 5 classes
        predicted_idx = NearestNeighborClassifier(test_features, hist_train, n_test_imgs,
n_train_imgs, k_nn)
        unique_idx, counter = np.unique(predicted_idx, return_counts=True)
        return unique_idx,counter
```