Prateek Gupta

gupta198@purdue.edu

ECE 661 Homework 7

**Theoretical Questions**

1. The reading material for Lecture 15 presents three different approaches to characterizing the texture in an image: 1) using the Grayscale CoOccurrence 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.

Grayscale CoOccurrence Matrix (GLCM): In this method, we construct a square symmetric matrix based on the portability distribution of several gray levels in the grayscale image. The amount of gray values used should be much less than 256, typically 16. We then normalize the matrix by dividing each element by the sum of the elements. Now, we can figure out the texture measures of the matrix. The most popular way is to compute the entropy value. The value tells you how random the texture in the image are.

Local Binary Pattern (LBP) histograms: This method relies on encoding every pixel in the binary image representation. The details for this method are described in the rest of this report, but at a high level, each pixel gets a value depending on its, and the surrounding pixels’, intensities. Then, the pixel is encoded into an integer based on the rotational invariant. This produces a vector, which can be turned into a histogram.

Gabor Filter Family: In this method, we measure the local periodicity in 30 bins, such that each bin is an octave. That is, the largest bin has a size double that of the previous bin. We will take the Fourier transform of an image, and then assign each frequency to a bin. This distribution helps find the textures.

1. 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. **Right**

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

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

**Overview**

In this assignment, we will attempt to sort images based on the scene they depict. We will use the Local Binary Pattern (LBP) algorithm to obtain a histogram feature vector for each image in a training set. Then, we will use the nearest neighborhood (NN) classifier to determine the appropriate label on a testing set.

**Local Binary Pattern**

The LBP algorithm generates computes rotational and translational invariants of the input greyscale images, and then extracts texture-based features. The algorithm is explained in the steps below.

Step 1: Create a binary image

We will start by computing the neighborhood points of a particular pixel. In order to do so, we need to decide a radius R and the number of points P around a particular pixel. We will utilize the following equation to calculate the coordinates of the neighbors

Setting the radius to 1 and P to 8 gives us the eight points in a circle such that point 1 is directly right of the center pixel A and the remaining points are in an circle of radius one counterclockwise from point 1. If the coordinates of A are (x, y) then each surrounding pixel’s coordinates are given by (x + Δu, y + Δv).

These points are not likely to correspond to the exact center of any particular pixel, so we will need to adjust by using bi-linear interpolation to estimate what the intensity at the surrounding point should be based on the intensity of its neighbors. Letting A be the intensity of the center pixel, B be this intensity of the horizontal pixel, C be the intensity of the vertical pixel, and D be the intensity of the diagonal pixel, the estimated intensity would be

Using this intensity, we will construct a binary image. If it is larger than the intensity of A, we will set a value of 1, otherwise it will be 0.

Step 2: Compute Rotational Invariants

Now that we have a binary image, we want to create some rotations. This will create a more regularized training set. We are assuming the rotation does not affect the label. Looking at the labels, it should not. This will be done using Professor Avi Kak’s BitVector module. Here, we essentially rotate the binary image at several angles and keep the image with the minimum decimal value for the binary sequence.

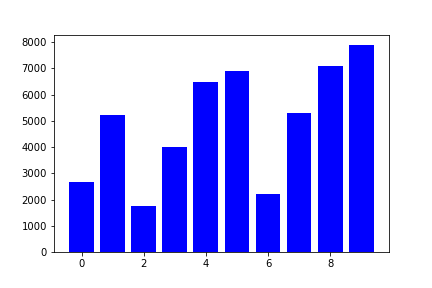
Step 3: Encoding the Image

We will now convert the pixels of the rotation invariant image into single integers. We use the BitVector module to get the 0-1 runs, then apply the following logic. If the binary representation is completely 0s, the encoded output is 0. If it is 0s then 1s, the output is 1. If it is all 1s, the output is P. And finally, the output will be P+1 otherwise. Once we have an encoded value for each pixel, we will compute the histogram for the pixels to act as the feature vector for each image.

**Nearest Neighborhood Classifier**

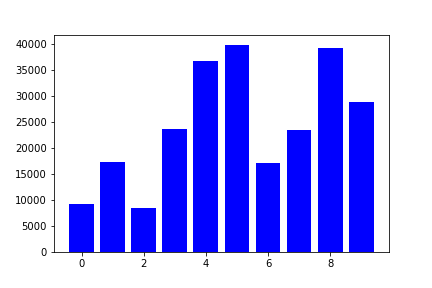
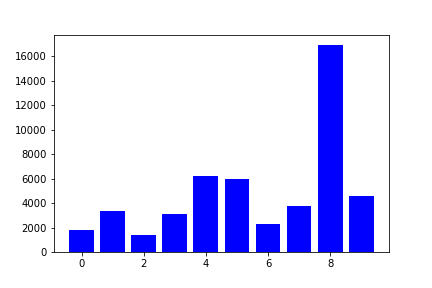
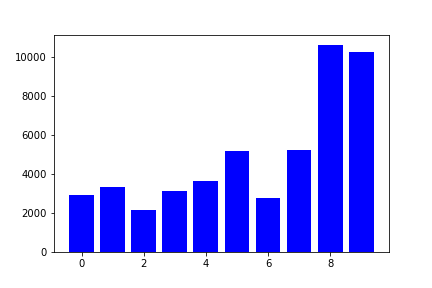
We will now attempt to classify a test image based on the training images. We will compute the Euclidian distance between the feature vector for each test image, and every training image in the following way

Here, we are computing the square root of the sum of the squares of the L2 norm. Vtest is the feature vector of the test image, and is the ith training image feature vector. Once we compute all the Euclidean distances, the pair that yields the smallest distance will give us the predicted label.

**Results**

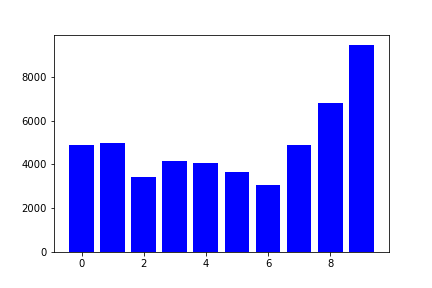
Building Image 1

Beach Image 1



Mountain Image 1

Car Image 1



Tree Image 1

Confusion Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Actual Class** | | | | |
| Beach | Building | Car | Mountain | Tree |
| **Predicted Class** | Beach | 5 | 0 | 0 | 0 | 0 |
| Building | 0 | 3 | 0 | 2 | 0 |
| Car | 0 | 1 | 4 | 0 | 0 |
| Mountain | 0 | 2 | 0 | 3 | 0 |
| Tree | 0 | 1 | 0 | 0 | 4 |

Accuracy

Based on the confusion matrix, we did not achieve a completely accurate model for classification. There was a total of 9 incorrectly predicted labels and 16 correctly predicted labels. Dividing the number of accurate predictions by the total number of predictions (25), we obtain an accuracy of **64%**. Unfortunately, this accuracy is not great. It is certainly significantly better than random guessing, but I am not sure how to make optimizations to this. The beach scene performed very well, but it seems the building and mountain got confused with each other.

import numpy as np

import cv2

import os

import subprocess

import matplotlib.pyplot as plt

import BitVector

import math

labels = ["beach", "building", "car", "mountain", "tree"]

def getCirclePts():

pts = []

for p in range(8):

pts.append([np.cos(0.25\*np.pi\*p), np.sin(0.25\*np.pi\*p)])

return pts

def LoadTrainImg():

imgs = [[],[],[],[],[]]

for i in range(20):

im = cv2.imread("Images/training/beach/%d.jpg" %(i+1))

imgs[0].append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(20):

im = cv2.imread("Images/training/building/%d.jpg" %(i+1))

imgs[1].append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(20):

im = cv2.imread("Images/training/car/%d.jpg" %(i+1))

imgs[2].append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(20):

im = cv2.imread("Images/training/mountain/%d.jpg" %(i+1))

imgs[3].append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(20):

im = cv2.imread("Images/training/tree/%d.jpg" %(i+1))

imgs[4].append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

return imgs

def LoadTestImg():

imgs = []

for i in range(5):

im = cv2.imread("Images/testing/beach\_%d.jpg" %(i+1))

imgs.append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(5):

im = cv2.imread("Images/testing/building\_%d.jpg" %(i+1))

imgs.append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(5):

im = cv2.imread("Images/testing/car\_%d.jpg" %(i+1))

imgs.append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(5):

im = cv2.imread("Images/testing/mountain\_%d.jpg" %(i+1))

imgs.append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

for i in range(5):

im = cv2.imread("Images/testing/tree\_%d.jpg" %(i+1))

imgs.append(cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY))

return imgs

def euclid(test, train):

tot = 0

for i in range(test):

tot += (test[i] - train[i])\*\*2

return math.sqrt(tot)

train = LoadTrainImg()

trainingLPBBeach = []

for im in train[0]:

trainingLPBBeach.append(LBP(im))

trainingLPBBuilding = []

for im in train[0]:

trainingLPBBuilding.append(LBP(im))

trainingLPBCar = []

for im in train[0]:

trainingLPBCar.append(LBP(im))

trainingLPBMountain = []

for im in train[0]:

trainingLPBMountain.append(LBP(im))

trainingLPBTree = []

for im in train[0]:

trainingLPBTree.append(LBP(im))

test = LoadTestImg()

for im in test:

testLBP = LBP(im)

minDist = 99999

bestLabel = ""

for i in trainingLPBBeach:

dist = euclid(testLBP, i)

if (dist < minDist):

minDist = dist

bestLabel = "Beach"

for i in trainingLPBBuilding:

dist = euclid(testLBP, i)

if (dist < minDist):

minDist = dist

bestLabel = "Building"

for i in trainingLPBCar:

dist = euclid(testLBP, i)

if (dist < minDist):

minDist = dist

bestLabel = "Car"

for i in trainingLPBMountain:

dist = euclid(testLBP, i)

if (dist < minDist):

minDist = dist

bestLabel = "Mountain"

for i in trainingLPBBuilding:

dist = euclid(testLBP, i)

if (dist < minDist):

minDist = dist

bestLabel = "Tree"

print(bestLabel)