Homework 8

Fangda Li
li1208@purdue.edu
ECE 661 - Computer Vision

November 8, 2016

1 Methodology

1.1 Local Binary Pattern (LBP) Feature Extraction

In this section, the steps to extract LBP histogram from an image given R and P are described, where R is the radius and P is the number of samples on the circumference. Note that all following operations

1. **Sampling**: given a pixel p, we draw a circle of radius R around it and take P samples on the circumference, as shown in Figure **FIXME**. Next, we form a binary sequence based on the sampled value. We assign a 1 if the sampled pixel value is greater or equal to the pixel value at p and a 0 otherwise. As a result, the final binary sequence S will be a combination of 1s and 0s of length P.

2. **Encoding**: now that we have the binary sequence S, we do bit-wise rotation on S until S starts with the longest continuous run of 0s. For example, if S is 00011110, the result of the rotation will be 00001111. Note that this manipulation is identical to encoding all the possible rotations of S with the minimum integer representation m, which guarantees rotation invariance to LBP features.

3. **Histogram**: so far we have obtained a minimum integer representation m for each pixel. The next step is to construct a histogram with $P+2$ bins to represent the whole image. Each m is assigned to a specific bin based on the following rule:

   (a) If m contains all 0s, assign m to bin 0.

   (b) If m contains all 1s, assign m to bin P.

   (c) If m only contains one run of 0s followed by one run of 1s, assign m to bin n, where $n \in [1, P-1]$ is the length of the second run.

   (d) Otherwise, assign m to bin $P+1$.

Now we have extracted LBP histogram for a given image.

1.2 Classification with Nearest Neighbor

In the training phase, we extract LBP histogram from all the training images and organize them in a K-D tree data structure. While in the testing phase, to classify a test image, we first extract the LBP histogram from the test image. Then, we find the k nearest neighbors of the test image LBP histogram in the K-D tree based on Euclidean distance. Next, two separate voting scheme are proposed: majority voting and weighted voting. In majority voting, each of the k neighbors casts a vote on its own class and finally whichever class with the most votes becomes the prediction for the test image. While in weighted voting, vote from each of the k nearest neighbor has a weight of distance inversed, i.e. $1/Euclidean(query, neighbor)$.

2 Results and Discussion

2.1 Results

Note that all the performance measurements used only weighted voting. Also, in confusion matrix, rows correspond to actual class labels while columns correspond to predicted class labels.

Table 1: Confusion matrix with $R = 1$ and $P = 8$; Overall accuracy is 60%.

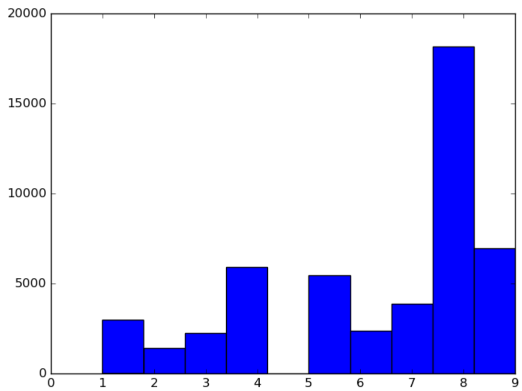
	Mountain	Building	Tree	Car
Mountain	4	1	0	0
Building	2	2	0	1
Tree	1	1	3	0
Car	0	2	0	3

Table 2: Confusion matrix with $R = 2$ and $P = 16$; Overall accuracy is 50%.

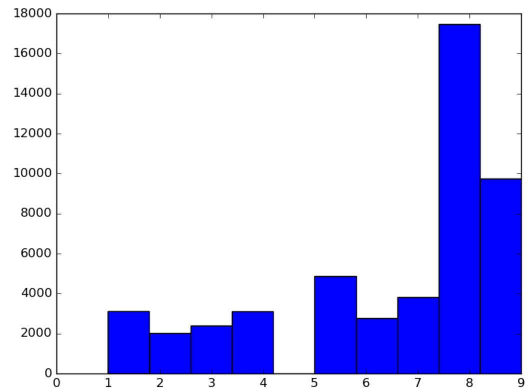
	Mountain	Building	Tree	Car
Mountain	3	2	0	0
Building	3	2	0	0
Tree	2	1	2	0
Car	0	2	0	3

Table 3: Confusion matrix with $R = 4$ and $P = 16$; Overall accuracy is 50%.

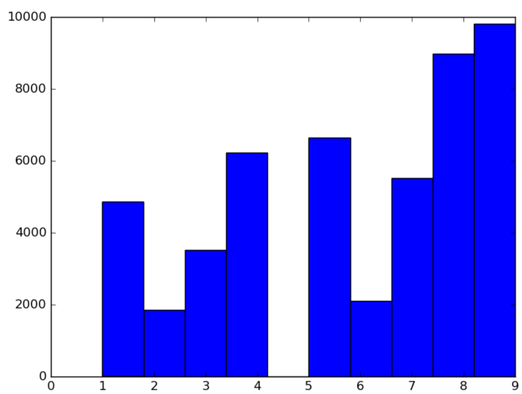
	Mountain	Building	Tree	Car
Mountain	2	3	0	0
Building	3	2	0	0
Tree	1	2	2	0
Car	0	1	0	4



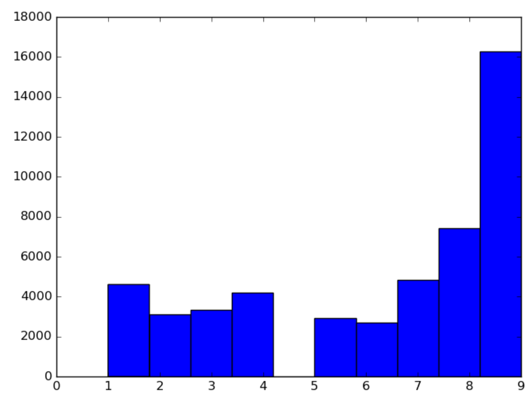
(a) Car



(b) Mountain



(c) Building



(d) Tree

Figure 1: LBP histograms of images from four sample images from different classes

2.2 Discussion

1. LBP classifier is very limited and its best accuracy only capped at 60% in this experiment. However, possible improvements may include:
   (a) Use bi-linear interpolation for non-integer pixel coordinates.
   (b) Normalize all LBP histograms.
   (c) Instead of find a single histogram for the whole image region, we can divide the image into a grid of sub-regions, find LBP histogram for each sub-region and concatenate all the histograms together as our final feature vector.
2. Histograms from different classes may look very similar, as shown in Figure 1.
3. LBP histogram doesn't capture the spatial information of the image. Instead, it only focus on local pattern or texture.

3 Source Code

3.1 lbp.py

```
#!/usr/bin/python
from pylab import *
import cv2
import BitVector

def get_lbp_hist(image, R, P):
        '''
                Find the normalized LBP histogram of an image.
                @image: np.ndarray of gray scale input image
                @R: int of number of LBP sampling circle radius
                @P: int of number of samples to take on each circle
                @return: 1-D array of histogram of the image
        '''
        h, w = image.shape[0], image.shape[1]
        H = zeros(P+2)
        # Iterate through the image
        for r in range(R, h-R):
                for c in range(R, w-R):
                        # Obtain the pixel values on the circle
                        sample_coords = get_sample_coords((r,c), R, P)
                        samples = [image[coord] for coord in sample_coords]
                        # samples = [bilinear_interpolate(image, coord[0], coord[1]) fo
                        samples = array(samples)
                        # Obtain the pattern
                        pattern = zeros(P)
```

```python
                        pattern[ samples >= image[(r,c)] ] = 1
                        # Obtain the encoding and add to histogram
                        H[ encode(pattern) ] += 1
                        temp.append(encode(pattern))
        return H / float(sum(H))

def bilinear_interpolate(image, y, x):
        '''
                Bilinear interpolation.
        '''
        fx, fy = int(floor(x)), int(floor(y))
        cx, cy = int(ceil(x)), int(ceil(y))
        dx, dy = x-fx, y-fy
        a1 = array([1-dy, dy])
        a2 = array([ [image[fy,fx], image[fy,cx]],
                          [image[cy,fx], image[cy,cx]] ])
        a3 = array([1-dx, dx])
        return dot(dot(a1,a2), a3.transpose())

def get_sample_coords(curr, R, P):
        '''
                Given the current pixel coordinate, find the coordinates of samples on
                @curr: tuple of (r,c)
                @R: int of number of LBP sampling circle radius
                @P: int of number of samples to take on each circle
                @return: list of (r,c) coordinate tuples
        '''
        coords = []
        dTheta = 2*pi / P
        theta = 0.
        for i in range(P):
                dX = np.cos(theta) * R
                dY = np.sin(theta) * R
                new_coord = ( int(curr[0]+0.5+dY), int(curr[1]+0.5+dX) )
                # new_coord = ( curr[0]+0.5+dY, curr[1]+0.5+dX )
                coords.append(new_coord)
                theta += dTheta
        return coords

def encode(pattern):
        '''
                Given pattern, find the integer LBP encoding (0 to P+1).
        '''
        pattern = list(pattern)
        P = len(pattern)
```

```
        bv = BitVector.BitVector( bitlist = pattern )
        ints = [int(bv << 1) for _ in range(P)]
        minbv = BitVector.BitVector( intVal = min(ints), size = P )
        bvruns = minbv.runs()
        if len(bvruns) == 1:
                # Single run of all 0s
                if bvruns[0][0] == 1:
                        return 0
                # Single run of all 1s
                else:
                        return P
        elif len(bvruns) == 2:
                # 0s followed by 1s
                return len(bvruns[1])
        else:
                # Mixed runs of both 0s and 1s
                return P+1

def main():
        # test = np.array([[5, 4, 2, 4, 2, 2, 4, 0],
        #                  [4, 2, 1, 2, 1, 0, 0, 2],
        #                  [2, 4, 4, 0, 4, 0, 2, 4],
        #                  [4, 1, 5, 0, 4, 0, 5, 5],
        #                  [0, 4, 4, 5, 0, 0, 3, 2],
        #                  [2, 0, 4, 3, 0, 3, 1, 2],
        #                  [5, 1, 0, 0, 5, 4, 2, 3],
        #                  [1, 0, 0, 4, 5, 5, 0, 1]])
        fpath = './imagesDatabaseHW8/training/tree/01.jpg'
        image = imread(fpath)
        image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        print get_lbp_hist(image, 1, 8)

if __name__ == '__main__':
        main()
```

3.2 main.py

```
#!/usr/bin/python
import os
from pylab import *
import cv2
from sklearn.neighbors import NearestNeighbors
from lbp import get_lbp_hist
from mpl_toolkits.mplot3d import Axes3D
```

```python
def train(R, P):
	'''
		Extract LBP features from training images.
		@R: int of number of LBP sampling circle radius
		@P: int of number of samples to take on each circle
	'''
	tdir = os.path.join(os.getcwd(),'imagesDatabaseHW8','training')
	classes = os.listdir(tdir)
	hists = None
	labels = None
	print "Training started..."
	for c in classes:
		sdir = os.path.join(tdir,c)
		samples = os.listdir(sdir)
		label = classes.index(c)
		for s in samples:
			# Read the image and convert to gray scale
			fpath = os.path.join(sdir,s)
			image = imread(fpath)
			image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
			# Get the LBP histogram
			hist = get_lbp_hist(image, R, P)
			if hists != None:
				hists = vstack( (hists, hist) )
				labels = append(labels, label)
			else:
				hists = hist
				labels = array([label])
			print "Training finished on...", fpath
	savetxt("train_hists.out", hists)
	savetxt("train_labels.out", labels)
	print "Trained histogram and labels saved..."

def test(R, P):
	'''
		Extract LBP features from testing images.
		@R: int of number of LBP sampling circle radius
		@P: int of number of samples to take on each circle
	'''
	tdir = os.path.join(os.getcwd(),'imagesDatabaseHW8','training')
	classes = os.listdir(tdir)
	tdir = os.path.join(os.getcwd(),'imagesDatabaseHW8','testing')
	samples = os.listdir(tdir)
	hists = None
	labels = None
```

```
        print "Testing started..."
        for s in samples:
                # Read the image and convert to gray scale
                fpath = os.path.join(tdir,s)
                image = imread(fpath)
                image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
                # Obtain ground truth index
                label = classes.index( s.split("_")[0] )
                # Get the LBP histogram
                hist = get_lbp_hist(image, R, P)
                if hists != None:
                        hists = vstack( (hists, hist) )
                        labels = append(labels, label)
                else:
                        hists = hist
                        labels = array([label])
                print "Testing finished on...", fpath
        savetxt("test_hists.out", hists)
        savetxt("test_labels.out", labels)
        print "Testing histogram and labels saved..."

def evaluate(n, weightedNN=True):
        '''
                Evaluate our classifier, calculate accuracy and confusion matrix.
                @n: int of number of nearest neighbors for majority voting
        '''
        trainHists = loadtxt("train_hists.out")
        trainLabels = loadtxt("train_labels.out").astype(uint8)
        testHists = loadtxt("test_hists.out")
        testLabels = loadtxt("test_labels.out").astype(uint8)
        nSamples = testLabels.size
        # Construct NN
        # Each sample hist is a row vector
        NN = NearestNeighbors(n_neighbors=n, metric='euclidean').fit(trainHists)
        # Get classes information
        tdir = os.path.join(os.getcwd(),'imagesDatabaseHW8','training')
        classes = os.listdir(tdir)
        print "Classes are...", classes
        nClasses = len(classes)
        # Initialize confusion matrix
        confusion = zeros((nClasses,nClasses)).astype(uint8)
        nCorrect = 0.
        print "Evaluation started..."
        for i in range(nSamples):
                # Obtain ground truth index
```

```
                ground_truth = testLabels[i]
                # Get the LBP histogram
                hist = testHists[i]
                # Obtain the indices of NNs
                _, indices = NN.kneighbors(hist.reshape(1,-1))
                indices = squeeze(indices)
                preds = zeros(len(classes))
                if weightedNN:
                        # Weighted voting
                        weights = zeros(n)
                        for j in range(n):
                                weights[j] += 1000. / linalg.norm(hist - trainHists[ ind
                        for j in range(n):
                                preds[ trainLabels[ indices[j] ] ] += weights[j]
                else:
                        # Majority voting
                        for j in indices:
                                preds[ trainLabels[j] ] += 1
                print preds
                pred = argmax(preds)
                confusion[ground_truth, pred] += 1
                if pred == ground_truth: nCorrect += 1.
                print "Ground Truth:", classes[ground_truth], "Classification:", classes
        print "Overall accuracy...", nCorrect / nSamples
        print "Confusion matrix..."
        print confusion

def main():
        R = 1
        P = 8
        n = 5
        train(R, P)
        test(R, P)
        evaluate(n, weightedNN=True)

if __name__ == '__main__':
        main()
```