Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW07

Local Binary Pattern Feature Extraction

For each pixel apart from the layer of pixels at the border, the location of P surrounding pixels is calculated as follows:

Delta u = R*cos(2∏p)/P, Delta v = R*sin(cos2∏p)/P

This gives us locations of P surrounding pixels in a circle of radius R. We have set R = 1, and P = 8. p varies from 0 to P-1. The coordinate given by p = 0 is the surrounding point right below the pixel, the coordinate given by p = 1 is to the right of the coordinate given by p = 0, and so on.

Because these coordinates may not correspond to actual pixels in the image, the gray value at that location is bilinearly interpolated using the following formula:

Value=(1-Delta u)*(1-Delta v)*A+(1-Delta u)*(Delta v)*B+(Delta u)*(1-Delta v)*C+(Delta u)*(Delta v)*D

where A, B, C and D are the gray levels at the pixel on the top left, top right, bottom left and bottom right respectively of the surrounding location.

Once gray levels for all P surrounding locations of a pixel are obtained, they are converted into either a 0 or a 1 value. If the gray level at the surrounding location is greater than or equal to the gray level at the center pixel, then the value is 1. Otherwise the value is 0.

Once we obtain a bit string of P bits, we rotate it, considering all rotated versions, to obtain the bit string that represents the least value in decimal. To find the smallest value in decimal, each rotated version of the bit string was converted into an unsigned integer in base 10. Once the bit string corresponding to the lowest decimal value is obtained, the runs in this bit string are analyzed.

If there are only two runs: a run of 0s followed by a run of r 1s, then the rth bin of the histogram is incremented.

If there is only one run of all 0s, then the 0^{th} bin of the histogram is incremented.

If there is only one run of all 1s, then the P^{th} bin of the histogram is incremented.

If there are more than two runs, then the $(P+1)^{st}$ bin of the histogram is incremented.

This gives a histogram containing (P+2) bins, which is the characteristic histogram or ‘feature vector’ of the image.

Nearest Neighbors Classification

Once the feature vector for all 20 training images for all 5 classes is calculated and stored in a file, when we want to classify an image, we compute its feature vector and find the Euclidean distance between that feature vector and the feature vector of each of the training images. We select k training images with the least Euclidean distance from the test image. k is set to 5 in the experiments. The class label that occurs most frequently in these k selected images is the predicted class label of the test image.

Confusion Matrix as a Performance Measure

A confusion matrix was computed to measure the performance of this image classification system. The size of this matrix is N x N, where N is the number of classes. In our case it is a 5 x 5 matrix. The rows represent actual class labels while the columns represent predicted labels. The diagonal elements indicate

how many test samples were classified correctly. The non-diagonal elements tell us how many images of a particular class were misclassified into a certain different class. This enables us to observe if images of a certain class tend to get misclassified into another specific class frequently.

LBP histogram feature vectors

Histogram of first beach image:

[23195,17315,22104,41861,31728,34276,24161,14531,16814,18529]

Histogram of first building image:

[6904,4049,3143,7450,5140,7518,3188,3040,7819,1267]

Histogram of first car image:

[3976,3350,13633,5922,4604,5875,3682,2099,3183,3089]
Histogram of first mountain image:

[3738,9058,5005,6502,5177,4462,3655,2490,4540,4717]

Histogram of first tree image:

[4526,2780,5361,7022,6363,6621,5229,2472,4650,4389]

In this list representation of the histogram, the bin is represented by the index in the list.

Results
beach_1.jpg is correctly classified as beach
beach_2.jpg is correctly classified as beach
beach_3.jpg is incorrectly classified as car
beach_4.jpg is correctly classified as beach
beach_5.jpg is correctly classified as beach
building_1.jpg is correctly classified as building
building_2.jpg is incorrectly classified as mountain
building_3.jpg is incorrectly classified as car
building_4.jpg is incorrectly classified as mountain
building_5.jpg is incorrectly classified as car
car_1.jpg is correctly classified as car
car_2.jpg is incorrectly classified as beach
car_3.jpg is correctly classified as car
car_4.jpg is correctly classified as car
car_5.jpg is correctly classified as car
mountain_1.jpg is correctly classified as mountain
mountain_2.jpg is correctly classified as mountain
mountain_3.jpg is incorrectly classified as car
mountain_4.jpg is incorrectly classified as car
mountain_5.jpg is correctly classified as mountain
tree_1.jpg is incorrectly classified as mountain
tree_2.jpg is correctly classified as tree
tree_3.jpg is incorrectly classified as mountain
tree_4.jpg is correctly classified as tree

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW07

tree_5.jpg is incorrectly classified as building

Performance measures

This is the confusion matrix:

[[4. 0. 1. 0. 0.]
 [0. 1. 2. 2. 0.]
 [1. 0. 4. 0. 0.]
 [0. 0. 2. 3. 0.]
 [0. 1. 0. 2. 2.]]
The order of labels is: beach, building, car, mountain, tree.

Observations on performance of image recognition system

The algorithm has an accuracy of 56%, since 14 of the 25 test images were correctly classified. This is much better compared to the baseline accuracy of 20%.

Beaches and cars were quite well-classified with an accuracy of 80%.

Two of the mountains were incorrectly classified as cars. Two of the buildings were incorrectly classified as mountains, so were two of the trees. Two of the buildings were also misclassified as cars. This is probably due to similarity in texture.

Using more training images may increase the accuracy. In that case, increasing the value of k would also make it more likely to find more similar training images. Using higher resolution pictures may also result in better matching of textures.

Source Code

```
import numpy as np

import cv2, os, time

from scipy import spatial

path = '../Users/rmahfuz/Desktop/661/HW07/'

#=============================================================================
============================

def create_hist(img):

    '''returns the histogram as a list'''

    if len(img.shape) == 3:

        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    hist = [0]*10

    P = 8

    R = 1
```

```
    for i in range(1, img.shape[0]-1):
        for j in range(1, img.shape[1]-1): #iterating through each pixel
            intensities = []
            for p in range(P):
                del_u = R*np.cos(2*np.pi*p/P)
                del_v = R*np.sin(2*np.pi*p/P)

                intensities.append((1-del_u)*(1-del_v)*img[i-1,j+1] + (1-del_u)*del_v*img[i+1,j+1] \
                                + del_u*(1-del_v)*img[i-1,j-1] + del_u*del_v*img[i+1,j-1])
            intensities = np.array(intensities)
            intensities = (intensities >= img[i][j]).astype(int)
            #finding minimum value pattern
            min_val = np.packbits(intensities)
            for k in range(len(intensities)):
                pattern = list(map(lambda x: (x+k)%8, [0,1,2,3,4,5,6,7]))
                cur_val = np.packbits(intensities[pattern])
                if cur_val < min_val:
                    min_val = cur_val
            code = np.binary_repr(min_val.item(), width = 8)
            #analyzing runs
            if code[0] == '1': #if all are 1
                hist[8] += 1
            else:
                k = 7
                while(code[k] == code[k-1] and k > 0):
                    k-=1
                if k == 0: #if all are 0
                    hist[0] += 1
                else:
                    break1 = k
```

```
                k-=1
                while(code[k] == code[k-1] and k > 0):
                    k-=1
                if k == 0: #if exactly 2 runs
                    hist[break1] += 1
                else: #if more than 2 runs
                    hist[9] += 1
    assert sum(hist) == (img.shape[0]-2) * (img.shape[1]-2)
    return hist
#================================================================================
==============================
def train(className):
    '''appends to hists.txt the histograms for a particular class' training images'''
    hists = []
    for file in os.listdir(path + 'imagesDatabaseHW7/training/{}/'.format(className)):
        img = cv2.imread(path + 'imagesDatabaseHW7/training/{}/'.format(className) + file)
        start = time.time()
        hists.append(create_hist(img))
        print('img {} of {} took {} seconds'.format(file, className, time.time() - start))
    with open(path + 'hists.txt', 'a') as fh:
        for hist in hists:
            to_write = ''
            for i in range(len(hist) - 1):
                to_write += str(hist[i]) + ','
            to_write += str(hist[-1])
            fh.write(to_write + '\n')
#================================================================================
==============================
def classify(img):
    hist = create_hist(img)
    with open(path + 'hists.txt', 'r') as fh:
```

```
        lines = fh.readlines()
        dists = []
        i = 0
        for line in lines:
            cur_hist = list(map(lambda x: int(x), line.split(',')))
            dists.append((spatial.distance.euclidean(cur_hist, hist), int(i/20)))
            i+=1
        dists.sort()
        count = [0]*5
        for item in dists[:5]:
            count[item[1]] += 1
        prediction = count.index(max(count))
        return prediction
#================================================================================
==============================
def train_all():
    train('beach')
    train('building')
    train('car')
    train('mountain')
    train('tree')
#================================================================================
==============================
def classify_all():
    classNames = ['beach', 'building', 'car', 'mountain', 'tree']
    confusion = np.zeros((5,5))
    i = 0
    for file in os.listdir(path + 'imagesDatabaseHW7/testing/'):
        img = cv2.imread(path + 'imagesDatabaseHW7/testing/{}'.format(file))
        ans = classify(img)
        if classNames[ans] == file[:-6]:
```

```
            print('{} is correctly classified as {}'.format(file, classNames[ans]))
        else:
            print('{} is incorrectly classified as {}'.format(file, classNames[ans]))
        confusion[int(i/5), ans] += 1
        i += 1
    print('Confusion matrix: ', confusion)

#================================================================================
==============================

def main():
    train_all()
    classify_all()

#================================================================================
==============================

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