ECE 661 – Homework 7

Ran Xu

xu943@purdue.edu

10/30/2018

1. Algorithms

1.1 Feature Extraction – Local Binary Patterns (LBP)

The purpose of LBP is to generate a rotation-invariant feature vector of an image so that a classifier can use this feature vector to classify the image. 5 steps to extract Local Binary Patterns (LBP) from a grey-scale image are shown as follows,

Step 1: Compute P-neighbors in the image

In this step, for each pixel (x, y) in the image, I select P neighbors that are R away with rotation angle $\theta = 0, \frac{2\pi*1}{P}, \dots, \frac{2\pi*(P-1)}{P}$. The pixel coordinate of the i-th neighbor is shown as follows,

$$x_i = x + R \cdot \cos\left(\frac{2\pi * i}{P}\right)$$

$$y_i = y + R \cdot sin(\frac{2\pi * i}{P})$$

To compute the pixel value whose coordinate is not integer, I use bilinear interpolation. Denote the integer part and fractional part of the i-th neighbor as $(\widetilde{x_i}, \widetilde{y_i})$ and (du, dv). Denote the surrounding pixels with integer coordinate of the i-th neighbor as A, B, C, and D in the top-left, top-right, bottom-left and bottom-right directions. The coordinate of A, B, C, and D are shown as follows,

$$A: (\widetilde{x_i}, \widetilde{y_i}), B: (\widetilde{x_i}, \widetilde{y_i} + 1)$$

$$C: (\widetilde{x_i} + 1, \widetilde{y_i}), D: (\widetilde{x_i} + 1, \widetilde{y_i} + 1)$$

The pixel value of the i-th neighbor in terms of A, B, C, D, du and dv is as follows,

$$Value(x_i, y_i) = (1 - du)(1 - dv)A + (1 - du)dv \cdot B + du(1 - dv)C + du \cdot dv \cdot D$$

*** Computation-optimized version (1000x faster than pixel-wise computation)**

Instead of iterating among all pixels, I use vector operations to speed up the processing, I first construct a matrix with shape $(H * W, P * 4)$ which includes the reference of all 4 integer neighbor of the P neighbors of each pixel. I then right-multiply this matrix with a transformation matrix as follows and directly get the pixel values of the H*W*P neighbors.

$$TransMatrix = \begin{bmatrix} (1-du_0)(1-dv_0) & 0 & \cdots & 0 \\ (1-du_0)dv_0 & 0 & \cdots & 0 \\ du_0(1-dv_0) & 0 & \cdots & 0 \\ du_0 \cdot dv_0 & 0 & \cdots & 0 \\ 0 & (1-du_1)(1-dv_1) & \cdots & 0 \\ 0 & (1-du_1)dv_1 & \cdots & 0 \\ 0 & du_1(1-dv_1) & \cdots & 0 \\ 0 & du_1 \cdot dv_1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & (1-du_{P-1})(1-dv_{P-1}) \\ 0 & 0 & \cdots & (1-du_{P-1})dv_{P-1} \\ 0 & 0 & \cdots & du_{P-1}(1-dv_{P-1}) \\ 0 & 0 & \cdots & du_{P-1} \cdot dv_{P-1} \end{bmatrix}$$

Step 2: Compute the binary P-neighbors in the image

In this step, the pixel values of the P neighbors are compared with the original pixel value in the image and end up with 1 if greater and equal and 0 otherwise.

Step 3: Compute rotation-invariant integer representation

In this step, the P 1s and 0s are firstly encoded to an integer but concatenate them together. To against the rotation, I shift the integer left by x bits (x=0, 1, … P-1). Note that the left x bits are concatenated to the rightmost of the integer. I select the minimum integer among the P shiftings. This integer is a rotation-invariant feature of the image.

Step 4: Encode the rotation-invariant integer using 0-1 runs

In this step, I further encode the rotation-invariant integer by its 0-1 runs. If the binary representation is all 0, the encoding is 0; if the binary representation is 0s and followed by 1s, the encoding is the number of the 1s; if the binary representation is all 1, the encoding is P; otherwise the encoding is P+1

Step 5: Compute the histogram among the image

Finally, I compute the histogram with P+2 bins on the 0-1 runs among the image. The histogram count is normalized by the sum so that it represents the occurrence of each 0-1 run.

1.2 Euclidean Distance Based Classifier

The purpose of an Euclidean Distance Based Classifier is to classify a test image given a dataset of training images. Denote the LBP histogram of test image as LBP_{test} and that of $N_{training}$ training images as $LBP_{training0}, LBP_{training1}, \dots, LBP_{training(N_{training}-1)}$

The Euclidean Distance (ED) between the test image and the i-th training image is defined as

$$ED = \sqrt{||LBP_{test} - LBP_{training\,i}||^2}$$

where $||\cdot||$ represents the l2-norm. The 5 smallest ED between the test image and all training images are picked. The classification is firstly made based on the majority of the class labels of the 5 training images. If multiple classes have a tie, we sum the ED inside each classes and choose the class label of the smallest summed ED as final prediction on the test image.

2. Observations

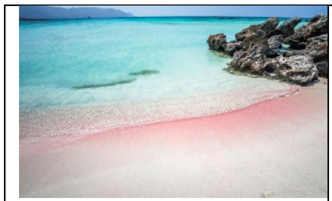
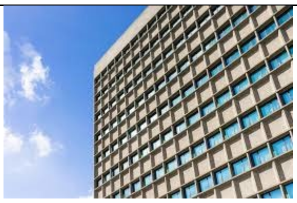
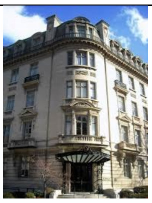
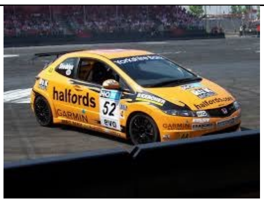
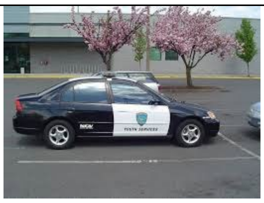
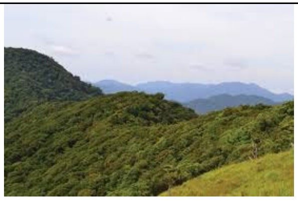
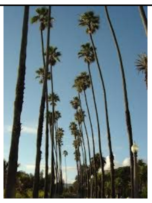
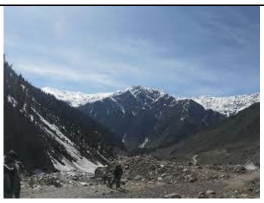
Firstly, the LBP histograms of the first images in each class in the training dataset are shown as follows,

Image	LBP histograms, where $R = 1,\ P = 8$
imagesDatabaseHW7/training/building/01.jpg	[0.134 0.099 0.038 0.100 0.099 0.056 0.024 0.083 0.076 0.292]
imagesDatabaseHW7/training/car/01.jpg	[0.114 0.307 0.034 0.099 0.093 0.058 0.027 0.056 0.063 0.148]
imagesDatabaseHW7/training/mountain/01.jpg	[0.208 0.191 0.041 0.071 0.057 0.050 0.036 0.056 0.083 0.206]
imagesDatabaseHW7/training/tree/01.jpg	[0.145 0.081 0.054 0.079 0.071 0.073 0.050 0.082 0.145 0.222]
imagesDatabaseHW7/training/beach/10.jpg	[0.190 0.111 0.043 0.124 0.094 0.080 0.037 0.060 0.084 0.177]

Secondly, using $R = 1, P = 8$, I get the confusion matrix as follows, the overall test accuracy is 68%. Wrong predictions are labelled in bold and underline. The overall test accuracy is much better than random guessing (20% accuracy) while far less than a production system. The accuracy can be improved a lot if using better features. However due to the fundamental limitation of LBP, the accuracy cannot be as high as 90%.

		Prediction				
		building	car	mountain	tree	beach
Ground truth	building	3	0	0	**2**	0
	car	0	3	0	0	**2**
	mountain	0	0	3	0	**2**
	tree	**1**	0	0	4	0
	beach	0	0	**1**	0	4

Finally, the 8 of 25 test images that are wrongly classified are shown as follows,

 Ground truth: beach Prediction: mountain	 Ground truth: building Prediction: tree	 Ground truth: building Prediction: tree	 Ground truth: car Prediction: beach
 Ground truth: car	 Ground truth: mountain	 Ground truth: tree	 Ground truth: mountain

Prediction: beach	Prediction: beach	Prediction: beach	Prediction: building

Table 1: Wrongly classified test images

As can be seen in Table 1, the first misclassified image maybe because the rocks on the beach misleads the classifier. However, in other misclassified images, I cannot even reason why they get wrongly classified. The reason is that LBP histogram cannot well distinguish the classes, and thus we cannot easily achieve higher accuracy than 68%.

3. Source code

3.1 Helper functions (LBP feature extraction, bit shifting operation and so on)

```
import numpy as np
import subprocess
import cv2
import time
import matplotlib.pyplot as plt

def LoadImages(Dataset):  # Load images from training or testing dataset
    ClassEncs = ["building", "car", "mountain", "tree", "beach"]
    if (Dataset=="Train"):
        Paths, Imgs = ([], [])
        for ClassEnc in ClassEncs:
            # List all images
            ImgDir = "imagesDatabaseHW7/training/%s/" %ClassEnc
            out, _ = subprocess.Popen("ls "+ImgDir, shell=True,
                                      stdout=subprocess.PIPE,
                                      stderr=subprocess.PIPE).communicate()
            Paths = Paths + [ImgDir + x.decode("utf-8") for x in out.split()]
        GT = [0]*20 + [1]*20 + [2]*20 + [3]*20 + [4]*20
    else: # Dataset=="Test"
        Paths, Imgs = ([], [])
        for ClassEnc in ClassEncs:
            # List all images
            cmd = "ls imagesDatabaseHW7/testing/%s_*" %ClassEnc
            out, _  = subprocess.Popen(cmd, shell=True,
                                       stdout=subprocess.PIPE,
                                       stderr=subprocess.PIPE).communicate()
            Paths = Paths + [x.decode("utf-8") for x in out.split()]
        GT = [0]*5 + [1]*5 + [2]*5 + [3]*5 + [4]*5
    for Path in Paths:
        # Read images
        Img = cv2.imread(Path, cv2.IMREAD_GRAYSCALE) # (H, W) uint8 np ndarray
        Imgs.append(Img)
    return Imgs, Paths, GT # list of (2D uint8, str, int)
def LBP_Parameters(R = 1, P = 8): # Init parameters in LBP
    dh = np.zeros((P*4, )).astype("int") # 1st dim
    dw = np.zeros((P*4, )).astype("int") # 2nd dim
    # Each 4 colomns is the multipler in bilinear interpolatation
    BLI_Matrix = np.zeros((P*4, P))
    for idx_P in range(P):
        h = R * np.cos(2*np.pi*idx_P/P)
        w = R * np.sin(2*np.pi*idx_P/P)
        dh[4*idx_P] = np.floor(h).astype("int")         # A
        dh[4*idx_P+1] = np.floor(h).astype("int")       # B
        dh[4*idx_P+2] = (np.floor(h)+1).astype("int")   # C
        dh[4*idx_P+3] = (np.floor(h)+1).astype("int")   # D
        dw[4*idx_P] = np.floor(w).astype("int")         # A
        dw[4*idx_P+1] = (np.floor(w)+1).astype("int")   # B
        dw[4*idx_P+2] = np.floor(w).astype("int")       # C
        dw[4*idx_P+3] = (np.floor(w)+1).astype("int")   # D
        dv, du = (h-np.floor(h), w-np.floor(w))
        BLI_Matrix[4*idx_P, idx_P] = (1-du)*(1-dv)      # A
        BLI_Matrix[4*idx_P+1, idx_P] = (1-du)*dv        # B
        BLI_Matrix[4*idx_P+2, idx_P] = du*(1-dv)        # C
        BLI_Matrix[4*idx_P+3, idx_P] = du*dv            # D
    return BLI_Matrix, dh, dw
def LBP(Imgs, R = 1, P = 8): # Img is 2D grey-scale image
    # Init with parameters
    BLI_Matrix, dh, dw = LBP_Parameters(R, P)
    ImgHists = np.zeros((P+2, len(Imgs)))

    for idx, Img in enumerate(Imgs):
        # Step 1a: Construct a (H*W, P*4) matrix of the 4-neighbor of each 8 points
        ImgInterp0 = np.zeros((int((Img.shape[0]-2*R-2)*(Img.shape[1]-2*R-2)), P*4))
        for idx_p0, (dh0, dw0) in enumerate(zip(dh, dw)):
            ImgInterp0[:,idx_p0] = Img[R+1+dh0:Img.shape[0]-R-1+dh0,
                                       R+1+dw0:Img.shape[1]-R-1+dw0].flatten()

        # Step 1b: ImgInterp0 * BLI_Matrix, then unflatten and get the raw LBP
        #          Output shape: (H-2R-2, H-2R-2, P)
        ImgInterp = np.matmul(ImgInterp0, BLI_Matrix)
        ImgInterp = np.reshape(ImgInterp, (int((Img.shape[0]-2*R-2)),
                                           int((Img.shape[1]-2*R-2)), P))
        # Step 2: Compare ImgInterp with Img, output shape: (H-2R-2, H-2R-2, P)
        RawImg = np.expand_dims(Img[R+1:Img.shape[0]-R-1,
                                    R+1:Img.shape[1]-R-1], axis=-1)
        ImgBin = (ImgInterp - RawImg) > 0

        # Step 3: Get MinIntVal
        ImgIntVal = np.zeros((int(Img.shape[0]-2*R-2), int(Img.shape[1]-2*R-2)))
        ImgRotIntVal = np.zeros((int(Img.shape[0]-2*R-2), int(Img.shape[1]-2*R-2), P))
        ImgMinIntVal = np.zeros((int(Img.shape[0]-2*R-2), int(Img.shape[1]-2*R-2)))
        for idx_P in range(P):
            ImgIntVal = ImgIntVal + ImgBin[:, :, idx_P] * (2**idx_P)
        ImgIntVal = ImgIntVal.astype("int")
        for idx_P in range(P):
            ImgRotIntVal[:, :, idx_P] = rol(ImgIntVal, idx_P, P)
        ImgMinIntVal = np.min(ImgRotIntVal, axis = -1)

        # Step 4: Encode the MinIntVal using "runs"
        ImgEnc = np.zeros(ImgMinIntVal.shape)
        ImgEncOcc = np.zeros(ImgMinIntVal.shape)
        for idx_P in range(P+1): # There is idx_P 1's followed by 0
            MaskNumber = (2**idx_P)-1
            ImgEnc = ImgEnc + (ImgMinIntVal==MaskNumber)*idx_P
            ImgEncOcc = ImgEncOcc + (ImgMinIntVal==MaskNumber)
        ImgEnc = ImgEnc + (1-ImgEncOcc)*(P+1)

        # Step 5: Get histogram
        histogram_bin =  np.arange(-0.5, P+1+0.5+1e-3, 1) # (P+3)-long, P+2 bins
        hist, _ = np.histogram(ImgEnc, bins = histogram_bin)
        ImgHists[:, idx] = hist/sum(hist)
    return ImgHists
# Bit-wise rotate left: 0b00001001 --> 0b00010010
rol = lambda val, l_bits, max_bits: \
    (val << l_bits) & (2**max_bits-1) | \
    ((val & (2**max_bits-1)) >> (max_bits-l_bits))
```

3.2 Main function – training and testing

```
(R, P) = (1, 8)
# Train & Test
```

```python
TrainImgs, TrainPaths, TrainingGT = LoadImages("Train")
Patterns_Train = LBP(TrainImgs, R, P)
TestImgs, TestPaths, TestGT = LoadImages("Test")
Patterns_Test = LBP(TestImgs, R, P)

for x in range(5):
    print("%s: %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f"
          %(TrainPaths[x*20], Patterns_Train[0, x*20], Patterns_Train[1, x*20], Patter
ns_Train[2, x*20],
           Patterns_Train[3, x*20], Patterns_Train[4, x*20], Patterns_Train[5, x*20],
           Patterns_Train[6, x*20], Patterns_Train[7, x*20], Patterns_Train[8, x*20],
           Patterns_Train[9, x*20]))

# Matching patterns
ConfMatrix = np.zeros((5, 5))
N_Train, N_Test = (len(TrainImgs), len(TestImgs))
for idx_testimg in range(N_Test):
    # Compute Distance
    EU_Dis = np.zeros((N_Train,))
    for idx_trainimg in range(N_Train):
        EU_Dis[idx_trainimg] = np.sqrt(np.sum((Patterns_Test[:,idx_testimg] -
                                               Patterns_Train[:,idx_trainimg])**2))

    # Sort in increasing order (EU_Dis, TrainingGT)
    Sorted_EU_Dis, Sorted_TrainingGT = zip(*sorted(zip(EU_Dis, TrainingGT)))
    Sorted_EU_Dis = np.array(Sorted_EU_Dis)
    Sorted_TrainingGT = np.array(Sorted_TrainingGT)

    # The GroundTruth is TestGT[idx_testimg]
    ClassScore = np.zeros((5,))
    for idx_cls in range(5):
        Mask = (Sorted_TrainingGT[0:5]==idx_cls)
        ClassScore[idx_cls] = np.sum(Mask)*100 + 1 - np.sum(Sorted_EU_Dis[0:5]*Mask)
    Prediction = np.argmax(ClassScore)
    ConfMatrix[TestGT[idx_testimg], Prediction] = ConfMatrix[TestGT[idx_testimg], Pred
iction] + 1
ConfMatrixDiag = [ConfMatrix[x,x] for x in range(5)]
print("ConfMatrix = \n", ConfMatrix.astype("int"))
print("(R, P) = (%d, %d), Test accuracy = %.0f%%" %(R, P, np.sum(ConfMatrixDiag)/25*10
0))
```