ECE661: Homework 7

Rahul Deshmukh
deshmuk5@purdue.edu
PUID: 0030004932

October 30, 2018

Introduction The goal of this homework is to carry out image classification using Local binary pattern (LBP) features and Nearest Neighbor(NN) classifier. We are given two sets of images- one for training and other for testing. We are required to generate LBP features of the training images as a first task and then compare these features for any test image feature and identify the class of the test image using its k closest neighbors.

Thereafter generate a confusion matrix to evaluate the performance of the method.

1 Logic and Methodology

The task of this homework can be further divided into the following sub-tasks:

a) Generating Local Binary Pattern (LBP)

b) Generate Training data

c) Classification using Nearest Neighbors.

d) generate Confusion Matrix

1.1 Local Binary Pattern (LBP)

As the name suggests, LBP estimates a binary pattern at every pixel location of an image using the information of its surrounding pixels and then finds a *minintval* from the binary pattern which makes the method rotation invariant. Then using an encoding scheme we generate a feature vector for the image.

The detailed steps involved are as follows:

1. For each pixel location $\vec{x}$ find its P closest equally-spaced neighbors at a radial distance of R using the following displacement values:
$$\Delta u, \Delta v = Rcos(\frac{2\pi p}{P}), Rsin(\frac{2\pi p}{P})$$
$$\forall p = (0, 1, 2...., P)$$
Therefore we get any i^{th} neighboring point will be at $P_i = \vec{x} + (\Delta u, \Delta v)$

2. At the identified P neighbors we estimate the pixel values using Bilinear interpolation given by:
$$image(P_i) = (1 - \Delta k)(1 - \Delta l)A + (1 - \Delta k)\Delta lB + \Delta k(1 - \Delta l)C + \Delta k\Delta lD$$
Where the points A,B,C,D are at the locations upper left, upper-right,lower-right and lower-left respectively with respect to the pixel location P_i. Also, $(\Delta k, \Delta l) = \vec{P_i} - \vec{A}$

3. We then append the pixel values for all the P neighbors into a single vector. Using this vector we generate a binary pattern using the center pixel(ie value at $\vec{x}$) as a threshold. This means that any pixel value in the vector whose value is greater than or equal to the threshold will get a 1 in the pattern else 0.

4. We then estimate the *minintval* of the binary pattern using circular permutations. I am doing this using string manipulations. The idea is to generate all circular permutations of the pattern and its corresponding integer value and then return the pattern with minimum interger value.

   The easiest way to do this is by repeating the pattern word twice and then sequentially take slices of same sizes moving one unit every time. For eg, if our pattern was '101', the repeated word will be '101101' and the possible permutations will be '101','011','110' and the corresponding integer value will be 5,3,6, Therefore the *minintval* pattern will be '011'.

5. After obtaining the *minintval* word we then find a scalar value using the encoding scheme suggested by the creators of LBP. The encoding scheme requires us to find the number of runs in the *minintval* word. To find the number of runs we find the number of times the string value changes from 0 to 1 or vice versa as we move along the word and we add 1 to this to get the number of runs.

6. After getting the number of runs we then proceed with the encoding with the following rules:

   i if number of runs=2 then encoding is given by the number of 1's in the *minintval* word

   ii Else, if *minintval* consists of all 0's then encoding is 0.

   iii Else, if *minintval* consists of all 1's then encoding is P.

   iv Else, if number of runs ¿2 then encoding is P+1

   Therefore we will require a size P+2 vector for storing the encoding.

7. Repeat steps 1-6 for all pixel locations and update the encoding vector. The resulting vector at the end of the loop will be our final LBP feature.

1.2 Generating Training Data

For this task we are given several images of different classes, we find the LBP feature vector for all images in the training set and store this data in a dictionary so as to easily refer back to it later.

1.3 Nearest Neighbor Classifier

In this task we are given a test image and are required to identify its class using its nearest neighbors in the training data, the distance between the the test image and all images in training set is calculated using the L2 norm of the LBP feature vectors. The steps involved are as follows:

1. Find the LBP feature vector for the test image.

2. Find the distance of the test image feature vector with all the feature vectors in the training set.

3. Find the 'n' closest neighbors based on the smallest 'n' distances.

4. Identify the class of the i^{th} nearest neighbor and keep a tally of the identified class.

5. The class which has the maximum number in the tally is the predicted class of the test image

1.4 Generating Confusion Matrix

After the identification of predicted classes for all testing images we can then generate a confusion matrix. The confusion matrix serves as a performance measure of the algorithm. Confusion matrix consists of True levels in rows and Predicted levels in columns.

2 Results

2.1 LBP feature vectors

The feature vectors for one image for each class in the training set is as follows:
beach
1.jpg – [9246. 17242. 8400. 23554. 37111. 40087. 17491. 21920. 41563. 27900.]

---

building
01.jpg – [2653. 5191. 1754. 4010. 6494. 6960. 2196. 5304. 7133. 7823.]

---

car
01.jpg – [1781. 3352. 1386. 3091. 6211. 5978. 2341. 3594. 17379. 4300.]

---

mountain
01.jpg – [2924. 3313. 2176. 3110. 3680. 5710. 2799. 3649. 16294. 5689.]

---

tree
01.jpg – [4857. 4899. 3417. 4327. 5052. 3845. 3145. 4884. 5846. 9141.]

2.2 Confusion Matrix

The confusion matrix for the testing set with n=5 nearest neighbors came out as:

	Beach	Building	Car	Mountain	Tree
Beach	5	0	0	1	0
Building	0	3	2	3	1
Car	0	0	3	0	0
Mountain	0	2	0	1	1
Tree	0	0	0	0	3

The overall accuracy = sum of correct Classification/Total number of Classification
= (5+3+3+1+3)/25 = 60%

3 Observations

a) The algorithm performs well in the sense that it gives an accuracy of 60% which is much more than the probability of randomly picking any class i.e. 20% (1/5).

b) The algorithm performs the best(100%) for the class 'beach' even when we have buildings and mountains in some of the testing images for beaches.

c) It performs decently for Buildings, cars and Tree with 3 out of 5 correct classifications each. Also for the false classifications, for instance in the case of cars, some cars are wrongly classified as buildings. This is because of both cars and buildings have windows and therefore the LBP pattern might resemble the same. Which indicates that when using LBP if our images have similar looking features, then ther performance is poor.

d) It performs the worst for the class Mountain with only 1 out of 5 correct classification. This can be because

4 Source Code:

4.1 Testing file

```
"""
ECE661: hw7 main file
@author: Rahul Deshmukh
email: deshmuk5@purdue.edu
PUID: 0030004932
"""

#import libraries
import cv2
import os
import numpy as np
import scipy.io as sio
import sys
sys.path.append('../../')
import MyCVModule as MyCV

# define training and testing path
mainpath='../imagesDatabaseHW7/'

# read training data from pickle file
import pickle
with open('training.pickle','rb') as f:
    P,R,class_names,class_sizes,class_LBPs=pickle.load(f)

#------------------Testing-------------------------#
# define testing path
testing_path=mainpath+'testing/'

test_img_list=os.listdir(testing_path)

#define number of nearest neighbors for NNclassifier
n=1

#loop over each image
for i in range(len(test_img_list)):
    print(test_img_list[i]) # testing image name
    # read image
    img=cv2.imread(testing_path+test_img_list[i],0)#gray scale image
    # find LBP vector of the testing image
    test_vec=MyCV.LBP(img,P,R)
    print(test_vec)
    # find predicted class using NNclassifier
    identified_class=MyCV.NNClassifier(test_vec,class_LBPs,
                                       class_names,
                                       class_sizes,n)
    print('identified class was '+identified_class)
    print('-----------------------')
#-------------------------------------------------#
```

4.2 Training file

```
"""
ECE661: hw7 main file
@author: Rahul Deshmukh
email: deshmuk5@purdue.edu
PUID: 0030004932
"""

#import libraries
import cv2
```

```
import os
import numpy as np
import sys
sys.path.append('../../')
import MyCVModule as MyCV

# define training and testing path
mainpath='../imagesDatabaseHW7/'

# constants
P=8 # neighbors
R=1 # radius

#--------------------Training---------------------------#
training_path=mainpath+'training/'
# find class names from folder names
class_names=os.listdir(training_path) # list of names
class_sizes={}
class_imgNames={}
for i in range(len(class_names)):
    temp=os.listdir(training_path+class_names[i]+'/')
    class_sizes[class_names[i]]=len(temp)
    class_imgNames[class_names[i]]=temp
# find LBP vector for all classes  and images
class_LBPs={}
for i in range(len(class_names)):
    print(class_names[i])
    LBP_vec=np.zeros((class_sizes[class_names[i]],P+2))
    for j in range(class_sizes[class_names[i]]):
        print(class_imgNames[class_names[i]][j])
        # find image name
        img_path=training_path+class_names[i]+'/'+\
                 class_imgNames[class_names[i]][j]
        # read image
        img=cv2.imread(img_path,0) # gray scale image
        # find LBP vector
        LBP_vec[j,:]=MyCV.LBP(img,P,R)
        print(LBP_vec[j,:])
        print('-----------------------------------')
    # put LBP_vec into the class_LBPs dictionary
    class_LBPs[class_names[i]]=LBP_vec # rows as LBP of jth image
    print('###########################################')

#save data to pickle file
import pickle
with open('training.pickle','wb') as f:
    pickle.dump((P,R,class_names,class_sizes,class_LBPs),f)
#---------- Training ends-----------------------------#
```

4.3 Functions

```
#====================HW7=================================#
#%%
# function for Nearest Neighbor classifier
def NNClassifier(test_vec,class_LBPs,class_names,class_sizes,n):
    """
    Input: class_names= dictionary of class names
           class_LBPs: dictiory of LBPs for all classes
           class_LBP[class_names[i]]= np array of size j,P+2: j is size of image
           class_sizes: dictionary of number of training images in one class
           test_vec: LBP vector for testing image
           n: number of nearest neighbors to be found
    Output: identfied_class: name of identified class (string format)
    """
    P=len(test_vec)-2
```

```
    numClass=len(class_names)
    # make a vector containing all class_LBPs using vstack
    all_vec=np.zeros((1,P+2))
    for i in range(numClass):
        all_vec=np.vstack((all_vec,class_LBPs[class_names[i]]))
    all_vec=all_vec[1:,:] # remove the first row of zeros
    # find euclidean distance
    d= all_vec-np.kron(np.array(test_vec),np.ones((all_vec.shape[0],1))) # difference
    d=np.square(d) # element wise squaring
    d= d@np.ones((P+2,1))# sum all cols: squared distances
    # find smallest n entries in d
    sorted_d=np.sort(d[:,0])
    count=np.zeros(numClass,dtype=int)
    # make cumulative size array: to be used for comparison
    cum_size=np.zeros(numClass)
    cum_size[0]=class_sizes[class_names[0]]
    for i in range(1,numClass):
        cum_size[i]=cum_size[i-1]+class_sizes[class_names[i]]
    #loop over number of nearest neighbors n
    for i in range(n):
        # find ith smallest distance
        dmin=sorted_d[i]
        # find index of dmin in d
        i_dmin=np.where(d[:,0]==dmin)[0][0]
        # find class type for this index
        for k in range(numClass):
            if i_dmin<cum_size[k]:
                count[k]+=1
                break
    # find the class which has max count
    i_max=np.argmax(count)
    identified_class=class_names[i_max] # string output
    return(identified_class)
#%%
#function for making the LBP from given image
def LBP(img,P,R):
    """
    Input: img: grayscale image mxn
           P: number of points in the circular neighbour for making the
           bit vector at any point
           R: radius of the neighbour circle
    Output: hist:  Feature vector from histogram of LBP of size P+2
    """
    m=img.shape[0];n=img.shape[1]
    hist=np.zeros(P+2,dtype=int)#LBP histogram with P+2 bins:0-P+1
    # loop over image pixels removing one row and col on both sides
    for i in range(m-2):
        for j in range(n-2):
            # find coordinate of current pixel: shifted by one unit
            x=np.array([j+1,i+1]) # row vector
            # find P neighbours at a radius of R
            p=np.arange(P)
            x_nbor=np.kron(x,np.ones((P,1)))
            x_nbor=x_nbor.T+np.round(R*np.array([np.cos(2*np.pi*p/P),np.sin(2*np.pi*p/P)])
                ,6) #stacked as col vectors
            # find pixel values at the neighbors using Bilinear
            p_nbor=[]
            for k in range(P):
                    p_nbor.append(Bilinear_gray(x_nbor[:,k],img))
            # convert neighboring pixel values to binary using center threshold
            binvec=binary_vec(p_nbor,img[x[1],x[0]])
            # convert binary vector to minIntval: to make Rotation Invariant
            min_bv=minbv(binvec) # is a string of len P
            #find encoding of minintvalbinary pattern
            encoding=encoding_LBP(min_bv)
```

```
            #update corresponding histogram range
            hist[encoding]+=1
    return(hist)
#function for finding the encoding of minintval pattern as developed by
# creators of LBP
def encoding_LBP(x):
    """
    Input: x: is a string of pattern made of 1 and 0, will be of size P
    Output: op: integer value for the pattern x using rules of LBP creators
    """
    P=len(x)
    #initialize runs
    runs=1
    # find runs for the pattern
    old_str=x[0] # first string in the pattern
    for i in range(P-1):
        if x[i+1]!=old_str:
            old_str=x[i+1]
            runs+=1
    # assign value for output to op
    if runs>2:
        op=P+1
    elif runs==1 and old_str=='1':
        op=P
    elif runs==1 and old_str=='0':
        op=0
    else:
        #runs=2; count the number 1s in the pattern
        op=0
        while True:
            if x[P-1-op]=='1':
                op+=1
            else:
                break
    return(op)
# function for finding the minimum binary vector
def minbv(x):
    """
    Input: x: array of 0 or 1s
    output: min_bv: str format minimum binary vector ...
    from all circular rotations
    """
    n=len(x)
    # make repeated str list from x for cicular rotations
    whole_str=''
    for i in range(n): whole_str=whole_str+str(x[i])
    repeat_str=whole_str+whole_str
    str_store=[]
    num_store=[]
    for i in range(n):
        # find circular rotation string slice
        str_store.append(repeat_str[i:i+n])
        num_store.append(int(str_store[i],2)) #integer value of the circular rotated string
    # find minintval
    imin=np.argmin(num_store)
    min_bv=str_store[imin] #string format
    return(min_bv)

#function to make binary vector from neighboring pixel values list/array
def binary_vec(x,c):
    """
    Input: x: array of pixel values(scalars) around the neighbors
           c: center pixel values
    Output: binvec: array of size(x) st any element in x if greater than c
    gets 1 else 0
```

```
    """
    n=len(x)
    binvec=np.zeros(n,dtype=int)
    for i in range(n):
        if x[i]>=c:
            binvec[i]=1
    return(binvec)

#function for Bilinear Interpolation of Gray Levels
def Bilinear_gray(x,img):
    """
    Input: x: [x,y] coordinates of pixel:np.array format
        img: gray scale image of size mxn
    Output:p: interpolated pixel value at X
    """
    # find nearest four pts
    A=[int(np.floor(x[0])),int(np.floor(x[1]))]
    B=[int(np.ceil(x[0])),int(np.floor(x[1]))]
    C=[int(np.floor(x[0])),int(np.ceil(x[1]))]
    D=[int(np.ceil(x[0])),int(np.ceil(x[1]))]
    # find pixel values at these pts
    a=img[A[1],A[0]]
    b=img[B[1],B[0]]
    c=img[C[1],C[0]]
    d=img[D[1],D[0]]
    #use formula for bilinear
    del_l=x[0]-A[0]
    del_k=x[1]-A[1]
    p=(1-del_k)*(1-del_l)*a + (1-del_k)*del_l*b + del_k*(1-del_l)*c + del_k*del_l*d
    p=round(p,3) #roundiing to three digits will take care of cases when del_k,de_l are
        close to 0 or 1
    return(p)
#=====================HW7=============================#
```