HOMEWORK 7
Mythra Varun Balakuntala SM
October 30, 2018

1. Local Binary Pattern

For obtaining a rotation invariant texture feature for the image we use the Local Binary Pattern (LBP). The method presented here to compute the LBP follows the procedure presented in [**?**] The steps to compute LBP are as follows,

- First compute a P bit binary pattern at each pixel. This is done by generating points at a fixed distance R around the pixel using

$$\Delta u = R\cos\left(\frac{2\pi p}{P}\right), \quad \Delta v = R\sin\left(\frac{2\pi p}{P}\right) \ , \ p = 0, 1, ..7$$

  where P is number of points considered.

- For each of the points a interpolated gray level is computed using bilinear interpolation. If point lines on edge joining two pixels A and B then it is computed using

$$g(x) = A(1 - \Delta l) + B(\Delta l)$$

  where Δl is distance of point from pixel A.
  If the point lies inside the polygon ABCD as shown below,

$$\begin{array}{cccc} & \leftarrow & \Delta y & \rightarrow & \\ \uparrow & A & & & B \\ \Delta x & & & & \\ \downarrow & & & * & \\ & C & & & D \end{array}$$

  The gray level value at the point is computed using biliear interpolation as follows,

$$g(x) = A(1 - \Delta x)(1 - \Delta y) + B(1 - \Delta x)(\Delta y) + C(\Delta x)(1 - \Delta y) + D(\Delta x)(\Delta y)$$

- Threshold each of the computed gray levels with respect to the gray level at the reference pixel around which the points are located. Set gray levels above the reference as 1 and those below as 0.

- This set of P binary values gives a binary pattern. Now, we need to make this rotation invariant.

- This is done by stitching together all these P binary bits into a single binary number. Then circularly shift the bits until a minimum value of the number (represented by the bits) is obtained. Ex: If the binary pattern is $[0, 1, 1, 1, 1, 1, 0, 0]$ the minimum binary number we can get by doing a circular shift is 00011111. This gives the rotation-invariant binary pattern.

- Then we need to encode this into a single number, this is implemented as follow, Find number of bit flips on moving along the min bit array. Then the integer encoding for the array is given by - sum of the bit values of the array if number of flips is less than 2 and $P + 1$ otherwise.
  Ex: For min binary array $[0, 0, 0, 0, 1, 1, 1, 1]$, the number of flips is 1 and the sum is 4, so the integer encoding is 4

- Next, find the local binary patterns at each pixel and find the integer encoding at each pixel. Then compute the histogram of number of pixels over the $P + 2$ encodings (from 0 to $P + 1$).

2. kNN Classifier

For each of the training images we extract the feature vector(histogram described in previous section). Each feature vector is associated with a known label, given in the training dataset. For the set of test images we find the feature vectors following the procedure in previous section.

Next, we find the k closest neighbours in the training dataset which are the k closest vectors to the test vector based on some distance metric. The most frequent label of the k closest vector's labels gives the predicted label for the given test image.

Results

Here we evaluated two different distance metrics one was the Euclidean distance and other the Chebyshev distance. We get slightly better results using the Chebyshev distance. Table 1 shows the confusion matrix using Chebyshev distance for k=5.

True↓ Pred→	Beach	Building	Car	Mountain	Tree
Beach	5.	0.	0.	0.	0.
Building	0.	3.	0.	2.	0.
Car	0.	2.	2.	0.	1.
Mountain	1.	1.	0.	3.	0.
Tree	0.	1.	0.	0.	4.

Table 1: Confusion matrix for given dataset using Chebychev distance

The accuracy for table 1 is computed using

$$accuracy = \frac{\text{trace}(C)}{\sum_{i,j} C_{ij}}$$

where C is the confusion matrix. The accuracy of system is was found to be 68%.

On using the Eucliedean distance as the distance metric for nearest neighbours we get the confusion matrix using $k = 5$ as shown in Table 2

True↓ Pred→	Beach	Building	Car	Mountain	Tree
Beach	5.	0.	0.	0.	0.
Building	0.	3.	0.	2.	0.
Car	0.	1.	3.	0.	1.
Mountain	1.	2.	0.	2.	0.
Tree	0.	1.	0.	1.	3.

Table 2: Confusion matrix for given dataset using Euclidean distance

The accuracy for table 2 is found to be 64%.

Examples of feature vectors obtained, The feature vector for *training/beach/*01 is

$$[9226., 17256., 8397., 23195., 37019., 39285., 17099., 24347., 37988., 30702.]$$

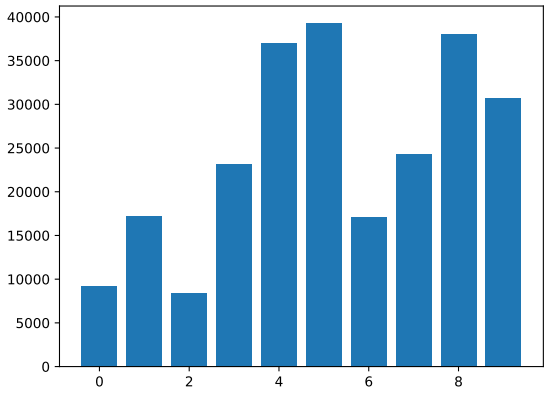

Figure 1: Histogram for beach/01

The feature vector for *training/building*/01 is

$$[2656., 5208., 1746., 4016., 6488., 6905., 2192., 5306., 7107., 7894.]$$

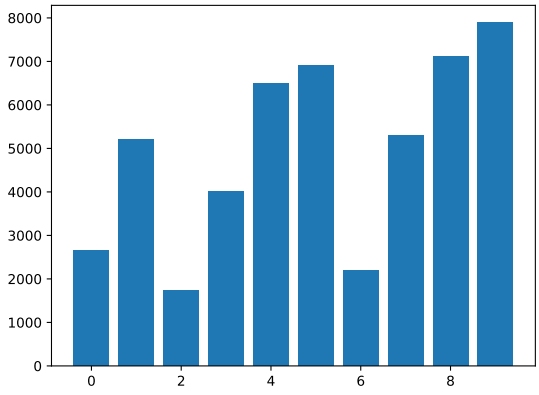

Figure 2: Histogram for building/01

The feature vector for *training/car*/01 is

$$[1769., 3332., 1414., 3081., 6190., 5595., 2234., 3986., 6128., 15684.]$$

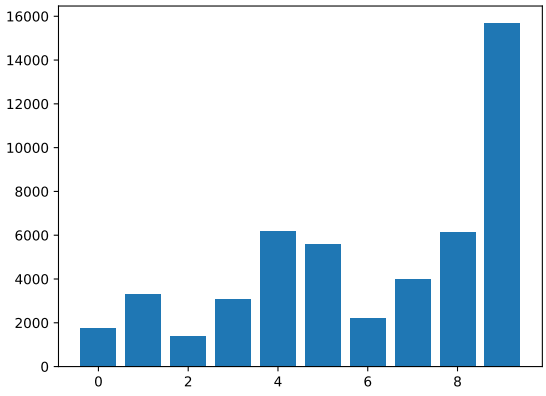

Figure 3: Histogram for car/01

The feature vector for *training/mountain*/01 is

$$[2926., 3314., 2175., 3102., 3681., 5515., 2771., 4979., 13973., 6908.]$$

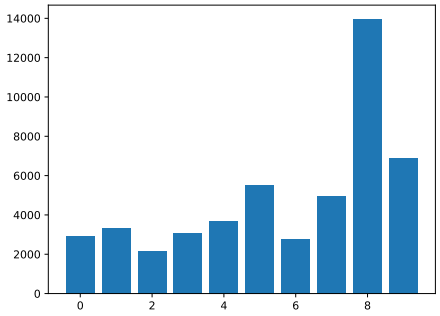

Figure 4: Histogram for mountain/01

The feature vector for *training/tree*/01 is

$$[4859., 4908., 3413., 4327., 5048., 3839., 3143., 4878., 5853., 9145.]$$

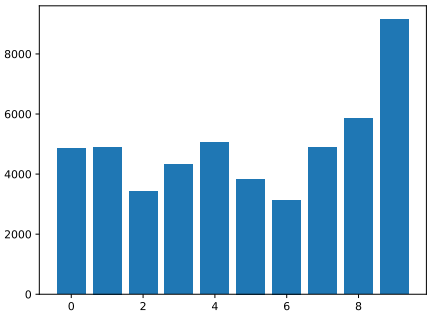

Figure 5: Histogram for tree/01

3. Code

```
    #!/usr/bin/env python3
################################################################################
# Script for homework 7 ECE661
# Image classification file
#  - Contains functions to perform LBP
#  - Functions to perform knn using different distances
# Author: Mythra Balakuntala
################################################################################

import cv2, numpy as np, math, time
import matplotlib.pyplot as plt
import time
from joblib import Parallel, delayed
import multiprocessing

class ImageClassify():
    def __init__(self):
        self.train_img_pth = { 1:'imagesDatabaseHW7/training/beach/',
                               2:'imagesDatabaseHW7/training/building/',
                               3:'imagesDatabaseHW7/training/car/',
                               4:'imagesDatabaseHW7/training/mountain/',
                               5:'imagesDatabaseHW7/training/tree/',
                               }
        self.test_pth = { 1:'imagesDatabaseHW7/testing/beach_',
                               2:'imagesDatabaseHW7/testing/building_',
                               3:'imagesDatabaseHW7/testing/car_',
                               4:'imagesDatabaseHW7/testing/mountain_',
                               5:'imagesDatabaseHW7/testing/tree_',
                               }
        self.labels = {1:'beach', 2:'building', 3:'car', 4:'mountaun', 5:'tree'}
        # self.sample =  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
        # ,[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]])
        self.P = 8          # Number of neighbours
        self.R = 1          # Radius of neighbours
        self.zer = 1e-5
        self.nimgs = 20     # Number of images per class in training
        self.nlabels = 5    # Number of classes
        self.ntest = 5      # Number of test images per class

# Function for bilenear interpolation
    def bl_interp(self, val, dx, dy):
        dx -=  np.floor(dx)
        dy -=  np.floor(dy)
        m,n = val.shape
        if m==2 and n==2:
            return(val[0,0]*(1-dx)*(1-dy) + val[0,1]*(1-dx)*dy + val[1,0]*(1-dy)*dx \
            + val[1,1]*dx*dy)
        elif m == 1:
            return(val[0,0]*(1-dy) + val[0,1]*dy)
        elif n == 1:
            return(val[0,0]*(1-dx) + val[1,0]*dx)

# Function to find runs
    def b_runs(self, gval):
        flips = 0
```

```
        for i in range(1,gval.shape[0]):
            if gval[i] != gval[i-1]:
                flips +=1
        if flips < 2:
            return(int(sum(gval)))
        else:
            return(self.P+1)

# Local Binary Pattern histogram function
    def lbp_fvec(self, img):
        #------------------------------------------------------------
        # Computes the local binary histogram from gray levels at each pixel
        # Then reduces it to feature vector and returns the feature vector
        # Inputs: img - Gray image
        # Outputs: Feature vector for image
        #------------------------------------------------------------
        hist = np.zeros(self.P+2)
        imgh, imgw = img.shape
        nbrs_x = self.R*np.cos(2*np.pi*np.arange(self.P)/self.P)
        nbrs_y = self.R*np.sin(2*np.pi*np.arange(self.P)/self.P)
        # Compute gray histograms for each pixel
        for i in range(1,imgh-1):
            for j in range(1,imgw-1):
                glvls = -1*np.ones(self.P)  # Gray histogram vector init
                # Compute gray histograms
                for k in range(self.P):
                    # If integral point
                    if(abs(nbrs_x[k] - int(nbrs_x[k])) < self.zer and\
                    abs(nbrs_y[k] - int(nbrs_y[k])) < self.zer):
                        glvls[k] = img[i+int(nbrs_x[k]),j+int(nbrs_y[k])]
                    # Bilear interpolate gray values for mid pixel values
                    else:
                        vals  = img[int(np.floor(i+nbrs_x[k])):int(np.ceil(i+nbrs_x[k]))+1\
                        ,int(np.floor(j+nbrs_y[k])):int(np.ceil(j+nbrs_y[k]))+1]
                        glvls[k] =  self.bl_interp( vals, nbrs_x[k], nbrs_y[k])
                # Binary representation from gray levels
                glvls = np.where(glvls>=img[i,j], 1, 0)
                # print(glvls)
            # Rotation invariant representation
                mbin = np.packbits(glvls)[0]
                mk = 0
                for k in range(1,8):
                    # Circular shift
                    bin = np.packbits(np.roll(glvls,k))[0]
                    if bin < mbin:      # find min
                        mk = k
                        mbin = bin
                glvls = np.roll(glvls,mk)   # Min bin
                # print(glvls)
                # print(self.b_runs(glvls))
            # Number encoding for min int
                hist[self.b_runs(glvls)] += 1
        return(hist)

# Function to compute distance
    def dist(self, v1, v2, dmetric = 0):
```

```
        if dmetric == 0:    # Euclidean distance
            return(np.linalg.norm(v1-v2))
        elif dmetric ==1:   # Dot product
            return(abs(abs(np.dot(v1/np.linalg.norm(v1),v2/np.linalg.norm(v2))) - 1))
        elif dmetric ==2:   # manhattan
            return(max(abs(v1-v2)))

# Function to find k nearest neighbours
    def knn(self, q, ds, k=5, dmetric = 0):
    #-------------------------------------------------------------
    # Computes the k nearest neighbours for the given query vector q
    # Inputs: q -query vector, ds -dataset, k - number of neighbours,
    #   dmetric - type of distance metric
    # Outputs: k nearest neigbours according to dmetric
    #-------------------------------------------------------------
        dvec = np.zeros(ds.shape[1])    #distance to data set
        for i in range(ds.shape[1]):
            dvec[i] = self.dist(q, ds[:,i], dmetric)
        # Find k nearest neighbours
        knn = np.argsort(dvec)[0:k]
        # Fine lablel of test image based on most frequent label of nn
        return(np.argmax(np.bincount((knn/self.nimgs).astype(int)))+1)

#  Testing function
    def test(self, i,j):
    #-------------------------------------------------------------
    # Compute feature vectors for test images
    # Inputs: i - class label, j - imaage number
    # Outputs: feature vector for image
    #-------------------------------------------------------------
    # Load testing dataset
        # Array to store feature vectors
        fvecs = np.zeros((self.P+2))
        img_rgb = cv2.imread(self.test_pth[i]+str(j)+'.jpg')  # RGB
        img_gry = cv2.cvtColor(img_rgb,cv2.COLOR_BGR2GRAY)  # GRAY
        fvecs = self.lbp_fvec(img_gry)
        return(fvecs)

# Training function
    def train(self, i, j):
        #-------------------------------------------------------------
        # Compute feature vectors for training images
        # Inputs: i - class label, j - imaage number
        # Outputs: feature vector for image
        #-------------------------------------------------------------
        fvecs = np.zeros((self.P+2))
        st = time.time()
        # Get image
        img_rgb = cv2.imread(self.train_img_pth[j]+str(i).zfill(2)+'.jpg')  # RGB
        img_gry = cv2.cvtColor(img_rgb,cv2.COLOR_BGR2GRAY)  # GRAY
        # Get features
        fvecs = self.lbp_fvec(img_gry)
        endt = time.time()
        print(i, ' Time: ',endt-st)
        return(fvecs)
```

```
# The main function
    def main(self, dtrain = 0, dtest = 0):
        #-----------------------------------------------------------------------
        # Compute features and then compute the confusion matrix based on knn
        # Inputs: dtrain - flag to indicate whether to train or use existing
        #   feature vectors from folder
        # dtest - flag to indicate computing features for test set or to
        #   load from folder
        # Outputs: Confusion matrix
        #-----------------------------------------------------------------------
        if dtrain:  # Check if training is selected
            inp = range(1,self.nimgs+1)
            num_cores = multiprocessing.cpu_count()
            for j in range(1,self.nlabels+1):
                results = Parallel(n_jobs=num_cores)(delayed(self.train)(i,j) for i in inp)
                np.save('trn'+str(j)+'.npy',results)
        # Read the feature vectors
        if dtest:
            inp = range(1,self.ntest+1)
            num_cores = multiprocessing.cpu_count()
            for j in range(1,self.nlabels+1):
                results = Parallel(n_jobs=num_cores)(delayed(self.test)(j,i) for i in inp)
                np.save('tst'+str(j)+'.npy',results)
        #-------------------------------------------------------------------------
        # Load the data
        ds = np.zeros((self.P+2, self.nimgs*self.nlabels)) # trained dataset
        ts = np.zeros((self.P+2, self.ntest*self.nlabels)) # trained dataset
        for j in range(1,self.nlabels+1):
            # Read train data features
            res = np.load('trn'+str(j)+'.npy')
            ds[:,self.nimgs*(j-1):self.nimgs*j] = res.T
            # Read test data features
            res = np.load('tst'+str(j)+'.npy')
            ts[:,self.ntest*(j-1):self.ntest*j] = res.T
        # Compute predicted labels
        tst_lbl = np.zeros((self.nlabels,self.ntest))
        cmat = np.zeros((self.nlabels,self.nlabels))
        for i in range(self.nlabels):
            for j in range(self.ntest):
                plbl = int(self.knn(ts[:,self.ntest*i+j],ds,5,0)) #predicted label
                tst_lbl[i,j] = plbl
                cmat[i,plbl-1] += 1
        accuracy = np.trace(cmat)/(self.nlabels*self.ntest)
        print(cmat)
        print(np.trace(cmat)/(self.nlabels*self.ntest))



if __name__ == '__main__':
    nimgs = 20
    nlabels = 5
    img_clf = ImageClassify()
    img_clf.main(0,0)
    # See description for main
    # img_clf.main(1,1)   # Call if you want to run training and testing
```