ECE 661 Homework 7

Suyash Ail
sail@purdue.edu

October 20, 2020

The task of this homework is to implement a simple image classifier using the Local Binary Pattern (LBP) feature extraction and Nearest Nearest (NN) classifier.

1 Theory Questions

Question1

This is a very good chart explaning all the texture based characterization techniques:

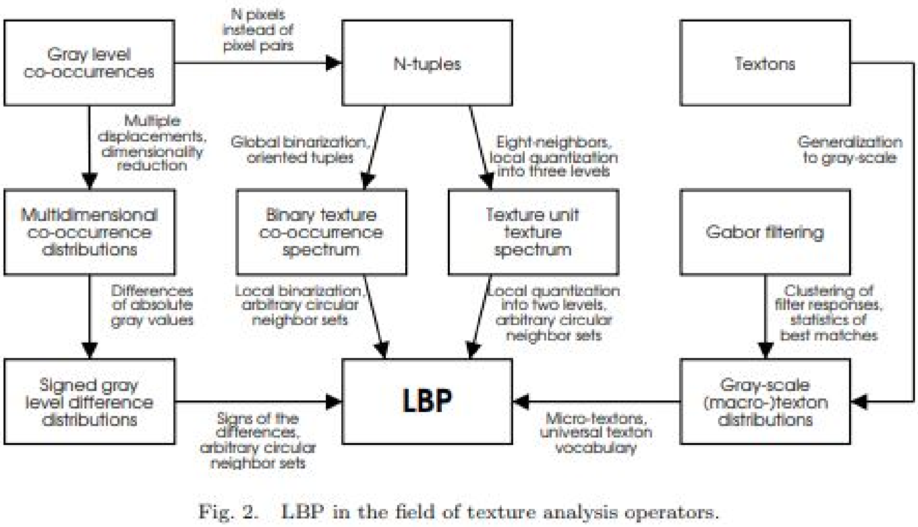

Figure 1: Texture based techniques [1]

1.0.1 Gray Level Co-Occurrence Matrix (GLCM)

- In case of (GLCM), we try to estimate the joint probability distribution $P[x_1, x_2]$ for grayscale values in an image where x_1 is grayscale value of any randomly selected pixel in the image and x_2 is the grayscale value of another pixel at a specific vector distance d from x_1

- We examine grayscale values at two different pixels that are separated by a distance vector d. After this, we count the number of pairs that are d distance apart but exhibit the grayscale values (x_1, x_2). We normalize this count to get $P[x_1, x_2]$, we can characterize the texture based on the shape of $P[x_1, x_2]$

- Using this joint probability distribution (microscopic view of texture) we can compute values of Entropy, Energy, Contrast and Homogeneity (macroscopic view of textures) that help with texture characterizations

1.0.2 Local Binary Pattern (LBP)

LBPs compute a local representation of texture. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels.

1. The fundamental idea of LBP is the notion of a binary pattern to characterize the grayscale variations around the pixels using 0s and 1s.

2. The pixels are assumed to lie on the circumference of a circle of radius R with the centre pixel as the center of the circle.

3. Not all points fall at pixel locations and hence bilinear interpolation is used to generate their pixel values.

4. Once these pixel values are obtained, we threshold them with the center pixel. If the center pixel is higher than the neighbour pixels, they are set to 0, else 1. Hence we obtain a P dimensional vector representation of the local texture of the image. The vector can be converted into an integer value which replaces the center pixel.

5. The binary pattern should be invariant of scale and rotation. This is obtained by circular shifting the feature vectors until the minimum integer representation is obtained.

The binary pattern and the texture represented by them can be visuaized as below(White points represent 1s and black points represent 0s) :

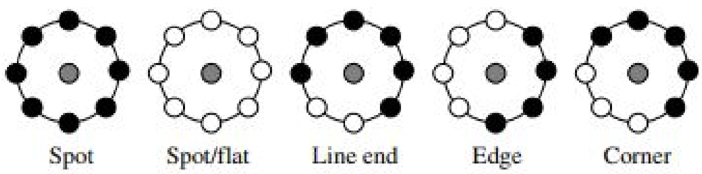

Fig. 3. Different texture primitives detected by the LBP.

Figure 2: LBP patterns and corresponding textures [1]

1.0.3 Gabor Filter Family

Gabor filters are used to extract localized features using the convolution operators. It is a part of the MPEG-7 standard. The convolution kernels are Gaussian weighted 2D Fourier kernels.

$$h(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(\frac{-1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right) . \cos 2\pi u_0 x$$

The Gaussian kernel helps in achieving localization while the Fourier transform helps in extracting spatial frequencies from the image. Multiple frequencies at multiple angles are used to extract an array of Gabor filtered channels which form the feature vector.

Question2

1. RGB and HSI are just linear variants of each other : True
2. The color space L*a*b* is a nonlinear model of color perception.: True
3. Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.: True

2 Logic

2.1 LBP feature extraction

The LBP feature extraction is a texture based feature extraction technique which is used to compute translation and rotation invariant features.

2.1.1 Creating binary patterns for inter-pixel variations

First step is define the neighbourhood of each pixel X. The neighbourhood is defined by two parameters - radius (R) and number of neighbours(P).
The P neighbours lie along a circle of radius R. Their position can be computed by

$$(\Delta u, \Delta v) = (Rcos(\frac{2\pi p}{P}, Rsin(\frac{2\pi p}{P})$$

where $p = 0, 1, 2...P$ and Δu and Δv indicate the displacement of the point p in the X and Y direction. Let A be the center pixel with coordinates (x,y). Then the neighbouring points are given by $x + \Delta u$ and $y + \Delta v$. $p = 0$ corresponds to the bottom pixel and $p = 4$ corresponds to the top pixel. All the P points will not lie on pixel coordinates. Hence to determine their location, we need bilinear interpolation.

$$I_p = (1 - \Delta u)(1 - \Delta v)A + (1 - \Delta u)(\Delta v)B + (\Delta u)(1 - \Delta v)C + (\Delta u)(\Delta v)D$$

where A,B,C,D represent the four corners of the rectangle around the center pixel.

Next, these intensity values are threshold with respect to the center pixel in order to obtain the binary pattern. If $I_p > A$, the value is 1 else 0.

2.1.2 Rotational Invariance in LBP

Given the binary pattern, the goal is to create rotation invariation patterns. To achieve this, the binary sequence is circularly shifted to obtain minimum integer when the binary sequence is converted to integer.

I have made use of professor Avinash Kak's BitVector module to obtain the binary sequence corresponding to minimum integer values.

2.1.3 Image Encoding

Next, the rotationally invariant binary pattern is encoded by a single integer ranging from 0 to P+2. The order of the binary pattern is used for encoding. The methodology is as follows:

1. If there are exactly two runs, a run of 0s followed by a run of 1s, then represent the pattern by number of ones in the sequence.
2. If the pattern has all 0s, represent the pattern by 0
3. If the pattern has all 1s, encode the pattern by 1
4. if the pattern has more than two runs, ecode the pattern by P+1.

For each point, we compute the binary encoding using the above steps. One thing to make sure is that all images are of the same size. Or else the histogram for a high resolution image and a low resolution image of the same class would differ heavily. We wish to keep the method invariant of the size of the image. Hence the image is first resized to (256,256) and then the computed histogram is normalized.

2.2 K-Nearest Neighbours

KNN is a supervised classification algorithm that requires both the feature and their corresponding class labels.KNN requires a distance metric to evaluate the closeness between the two feature vectors. In this homework I have used Euclidean distance and Manhattan distance.

There is no training as such for the KNN algorithm. During the testing phase, each new instance is compared with every feature in the train dataset. The class labels corresponding to the K closest points are chosen to determine the final class using a voting method. The class that occurs maximum times is selected as the predicted class.

3 Results

Figure 3: Beach

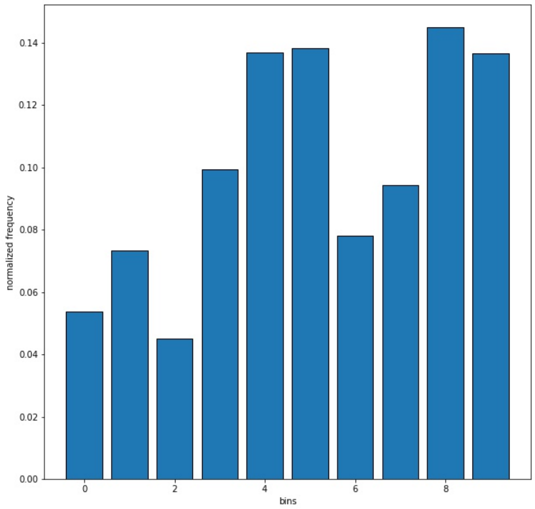

Figure 4: Beach histogram

Figure 5: Building

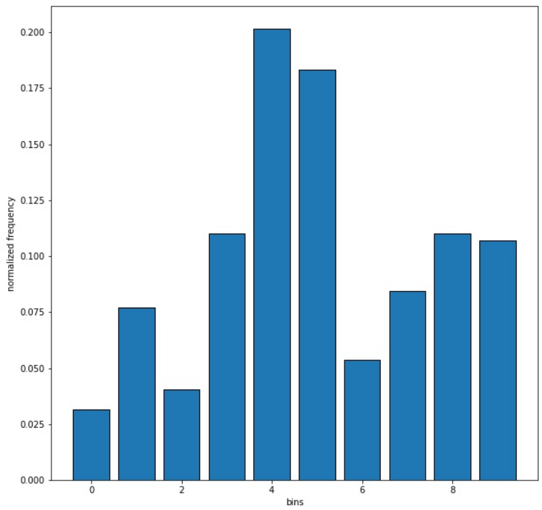

Figure 6: Building histogram

Figure 7: car

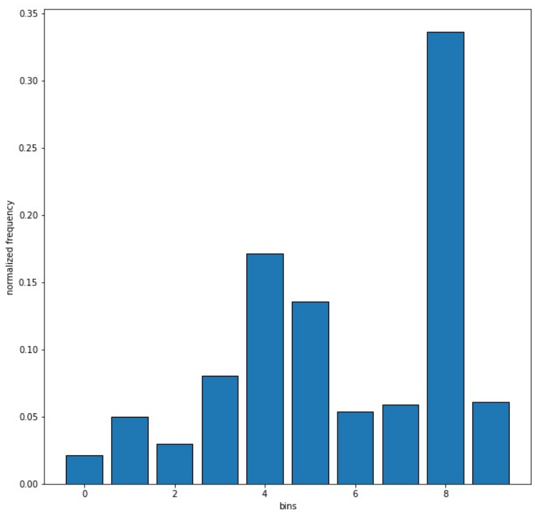

Figure 8: Car histogram

Figure 9: Mountain

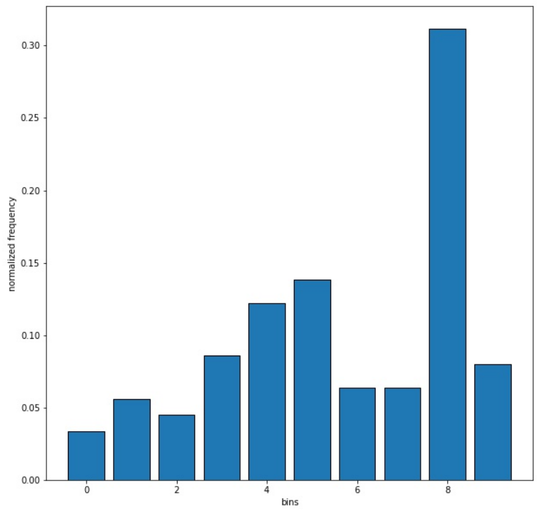

Figure 10: Mountain histogram

Figure 11: Tree

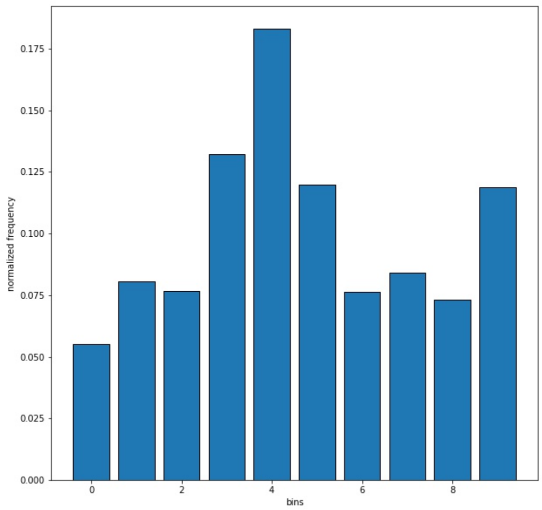

Figure 12: Tree histogram

3.1 Confusion Matrix

$$\text{ConfMat} = \begin{bmatrix} True/Predicted & Tree & Beach & Mountain & Building & Car \\ Tree & 3 & 0 & 1 & 1 & 0 \\ Beach & 0 & 5 & 0 & 0 & 0 \\ Mountain & 0 & 2 & 2 & 1 & 0 \\ Building & 1 & 0 & 1 & 2 & 1 \\ Car & 0 & 0 & 0 & 2 & 3 \end{bmatrix}$$

Accuracy of the classifier =

$$\frac{3+5+2+2+3}{25} = 0.6$$

The classifier has correctly classified all the Beach images. It performed well on tree and car but had trouble with mountain and building images.

Varying the distance metric did not change the accuracy.By increasing the K to 7, i got a higher accuracy.

$$\text{ConfMat} = \begin{bmatrix} True/Predicted & Tree & Beach & Mountain & Building & Car \\ Tree & 3 & 0 & 1 & 1 & 0 \\ Beach & 0 & 5 & 0 & 0 & 0 \\ Mountain & 0 & 1 & 3 & 1 & 0 \\ Building & 1 & 0 & 1 & 2 & 1 \\ Car & 0 & 0 & 0 & 2 & 3 \end{bmatrix}$$

Accuracy of the classifier =

$$\frac{3+5+3+3+3}{25} = 0.68$$

Further increasing the K value resulted in decrease of accuracy.

4 References

[1] Maenpaa, Topi Pietikainen, Matti. (2005). Texture analysis with local binary patterns. Handbook of Pattern Recognition and Computer Vision. 10.1142/9789812775320$_0$011.

5 Code Listings

```
# -*- coding: utf-8 -*-
"""hw7_suyash_ail.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1uRQot1tPBL6wfsXMYHkGMDVlxXuJVxSB
"""

import os
os.chdir('/content/drive/My Drive/BitVector-3.4.9/BitVector')

import numpy as np
import matplotlib.pyplot as plt
import glob
import cv2
from BitVector import BitVector
import pickle

'''
def n_neighbour(image,radius=1,num_neighbours=8):
  R = radius
  P = num_neighbours

  dx_list = []
  dy_list = []
  for p in range(P):
    du = R*np.cos(2*np.pi*p/P)  #get x-coord
    dv = R*np.sin(2*np.pi*p/P)  #get y-coord

    if abs(du)< 1e-4: du=0
    if abs(dv)< 1e-4: dv=0

    dx_list.append(du)
    dy_list.append(dv)

  return dx_list,dy_list
'''

def bilinear_intp(A,B,C,D,dx,dy):
  '''
  Function that returns the bilinear interpolation of a point with respect to
    its 4 neighbour points.
  A------B
  |      |
  |    X |
  C------D

  A is the CENTER pixel
  dx : x distance from A
  dy : y distance from A
  '''
  return (1-dy)*(1-dx)*A + (1-dy)*dx*B + dy*(1-dx)*C + dy*dx*D

def lbp (A):
  # A is the n neighbourhood of the pixel
```

```
    '''
    a00  a01  a02
    a10 'a11' a12
    a20  a21  a22

    V vector starts from from a21 and moves counter-clockwise
    v6 v5 v4
    v7    v3
    v8 v1 v2

    v1,v3,v5,v7 are the same as original pixels

    for now hard-coded the distances..
    '''
    v1 = A[2,1]
    v2 = bilinear_intp(A[1,1], A[1,2], A[2,1], A[2,2], 0.707, 0.707)
    v3 = A[1,2]
    v4 = bilinear_intp(A[1,1],A[1,2],A[0,1],A[0,2],0.707,0.707)
    v5 = A[0,1]
    v6 = bilinear_intp(A[1,1],A[1,0],A[0,1],A[0,0],0.707,0.707)
    v7 = A[1,0]
    v8 = bilinear_intp(A[1,1],A[1,0],A[2,1],A[2,0],0.707,0.707)

    vec = np.array([v1,v2,v3,v4,v5,v6,v7,v8])
    vec = vec >= A[1,1]


    ############ Using prof. Avinash Kak's BitVector module to get the lbp value
        ################
    # https://pypi.org/project/BitVector/
    bv = BitVector(bitlist = vec)
    intvals = [int(bv<<1) for _ in range(P)]  # integer values for circular
        shift
    #print('intvals=',intvals)
    minbv = BitVector(intVal = min(intvals), size = P)
    #print('min bit vector=',minbv)
    bvruns = minbv.runs()
    #print('bit vector runs=',bvruns)
    if (len(bvruns) > 2): return P + 1
    elif (len(bvruns) == 1 and bvruns[0][0] == '1'): return P
    elif (len(bvruns) == 1 and bvruns[0][0] == '0'): return 0
    else: return len(bvruns[1])

def get_lbp_hist(img_path, radius=1, num_neighbours=8):
    P=num_neighbours
    R=radius
    img = cv2.imread(img_path,0)
    img = cv2.resize(img,(256,256))
    #img = cv2.resize(img,(40,40))
    #print(img.shape)
    hist =[0]*(P+2)
    for x_idx in range(R, img.shape[1]-R):
        for y_idx in range(R, img.shape[0]-R):
            window = img[y_idx-1:y_idx+2, x_idx-1:x_idx+2]
            pvalue = lbp(window)

            hist[pvalue] += 1

```

```
  hist = np.array(hist).astype(float) / np.sum(hist)
  return hist

#get 1 image from each class and plot the histogram

im1_path = '/content/drive/My Drive/imagesDatabaseHW7/training/beach/1.jpg'
im1 = cv2.imread(im1_path)
im2 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
plt.imshow(im2)

hist = get_lbp_hist(im1_path)
plt.figure(figsize=(10,10))
fig=plt.bar(range(10),hist,edgecolor='black')
plt.xlabel('bins')
plt.ylabel('normalized frequency')
cv2.imwrite("beach.jpg",im1)
#cv2.imwrite("beach_hist.jpg",fig)
plt.savefig("beach_hist.jpg")


im1_path = '/content/drive/My Drive/imagesDatabaseHW7/training/building/01.jpg'
im1 = cv2.imread(im1_path)
im2 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
plt.imshow(im2)



hist = get_lbp_hist(im1_path)
plt.figure(figsize=(10,10))
fig=plt.bar(range(10),hist,edgecolor='black')
plt.xlabel('bins')
plt.ylabel('normalized frequency')
cv2.imwrite("building.jpg",im1)
plt.savefig("building_hist.jpg")

im1_path = '/content/drive/My Drive/imagesDatabaseHW7/training/mountain/01.jpg'
im1 = cv2.imread(im1_path)
im2 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
plt.imshow(im2)

hist = get_lbp_hist(im1_path)
plt.figure(figsize=(10,10))
fig=plt.bar(range(10),hist,edgecolor='black')
plt.xlabel('bins')
plt.ylabel('normalized frequency')
cv2.imwrite("mountain.jpg",im1)
plt.savefig("mountain_hist.jpg")


im1_path = '/content/drive/My Drive/imagesDatabaseHW7/training/tree/01.jpg'
im1 = cv2.imread(im1_path)
im2 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
```

```
plt.imshow(im2)

hist = get_lbp_hist(im1_path)
plt.figure(figsize=(10,10))
fig=plt.bar(range(10),hist,edgecolor='black')
plt.xlabel('bins')
plt.ylabel('normalized frequency')
cv2.imwrite("tree.jpg",im1)
plt.savefig("tree_hist.jpg")

im1_path = '/content/drive/My Drive/imagesDatabaseHW7/training/car/01.jpg'
im1 = cv2.imread(im1_path)
im2 = cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
plt.imshow(im2)

hist = get_lbp_hist(im1_path)
plt.figure(figsize=(10,10))
fig=plt.bar(range(10),hist,edgecolor='black')
plt.xlabel('bins')
plt.ylabel('normalized frequency')
cv2.imwrite("car.jpg",im1)
plt.savefig("car_hist.jpg")



# do this for every image in the folder
def train():
  #### loading images from folder code taken from :https://www.codegrepper.com/code-examples/objectivec/how+to+load+images+from+folder+in+python
  #training_dir='/content/drive/My Drive/imagesDatabaseHW7/training/'+classname
  training_dir='/content/drive/My Drive/imagesDatabaseHW7/training/'
  classnames = os.listdir(training_dir)
  i=0
  for classname in classnames:
    #print(classnames)
    images = []
    hist_list=[]
    count=0
    image_dir = training_dir+classname
    for filename in os.listdir(image_dir):
      hist = get_lbp_hist(os.path.join(image_dir,filename))
      count+=1
      print("histogram of " + classname + " number: "+str(count))
      hist_list.append((hist,i))
    with open(classname+'_features.pickle','wb') as fp:
      pickle.dump(hist_list,fp)

    with open(classname+'_features.pickle','rb') as fp:
      pickle.load(fp)
    i+=1
  #return hist_list

train()

'''
class labels:
```

```
  tree=0
  beach=1
  mountain=2
  building=3
  car=4
'''
with open('tree_features.pickle', 'rb') as fp:
  tree_features = pickle.load(fp)
with open('beach_features.pickle', 'rb') as fp:
  beach_features = pickle.load(fp)
with open('mountain_features.pickle', 'rb') as fp:
  mountain_features = pickle.load(fp)
with open('building_features.pickle', 'rb') as fp:
  building_features = pickle.load(fp)
with open('car_features.pickle', 'rb') as fp:
  car_features = pickle.load(fp)
#print(beach_features[0][-1])

train_features = tree_features + beach_features + mountain_features +
    building_features + car_features

def test():
  #### code reference- Naveen Madapana (best solution1 2018) ########
  testing_dir='/content/drive/My Drive/imagesDatabaseHW7/testing'
  test_img_paths = glob.glob(os.path.join(testing_dir, '*.jpg'))
  out_features = {os.path.basename(test_img_path): None for test_img_path in
    test_img_paths}
  #print(out_features)
  for test_img_path in test_img_paths:
    out_feat_vec = get_lbp_hist(test_img_path)
    out_features[os.path.basename(test_img_path)] = out_feat_vec
  with open('test_features.pickle', 'wb') as fp:
    pickle.dump(out_features, fp)

  with open('test_features.pickle', 'rb') as fp:
    out_features = pickle.load(fp)
  class_dict = {"tree":0, "beach":1, "mountain":2, "building":3, "car":4}
  test_features = []
  for fname, test_inst in out_features.items():
    cname = os.path.splitext(fname)[0].split('_')[0]
    c_idx = class_dict[cname]
    test_features.append((test_inst,c_idx))

  return test_features

test_features=test()

def most_frequent(List):
    #https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list
    /
    return max(set(List), key = List.count)

def get_confusion_matrix(pred,true,count):
  conf_mat = np.zeros((5,5))
  correct=0
  for i in range(len(true)):
    if pred[i]==true[i]:
      correct+=1
```

```python
      conf_mat[true[i],pred[i]]+=1
  accuracy = correct/len(pred)
  print("Accuracy: ",accuracy)
  print()
  print("confusion matrix:\n",conf_mat)


def knn(train_input,test_input,metric='Euclidean',K=5):
  train_input = np.array(train_input)
  test_input = np.array(test_input)
  train_data = train_input[:,0]
  train_label = train_input[:,1]
  test_data = test_input[:,0]
  test_label = test_input[:,1]
  pred_labels=[]
  for i in range(len(test_input)):
    dist =[]
    for j in range(len(train_input)):
      if metric =='Euclidean':
        distance = np.linalg.norm(train_data[j]-test_data[i])
      elif metric =='Manhattan':
        distance = np.sum(np.abs(train_data[j]-test_data[i]))
        #distance =0
      dist.append((distance,train_label[j]))
      #print(dist)
    dist.sort(key=lambda x:x[0])
    #print(dist)
    labels=[x[1] for x in dist]
    topK=labels[:K]
    pred_labels.append(most_frequent(topK))

  get_confusion_matrix(pred_labels,test_label,len(test_label))
  return pred_labels

labels =knn(train_features,test_features,metric='Manhattan',K=7)
```