ECE661: Computer Vision (HW7)

Risi Jaiswal
rjaiswa@purdue.edu

October 20, 2020

Theory Task

1:

Grayscale Co-occurence Matrix(GLCM):

- GLCM method for texture extraction is based on second order statistical properties. In GLCM method joint probability distribution $P(x_1, x_2)$ is estimated where x_1 is randomly selected pixel value in image and x_2 is is pixel value at a distance d.
- Texture is characterized by shape of this joint probability distribution. GLCM is formulated as matrix of order LXL where L is number of levels in which image intensities have been quantized.
- Whole image is traversed from left to right and top to bottom and grayscale value is examined at current pixel and pixel at distance d. While scanning if we find same pair of values (x, y) , GLCM matrix at location (x, y) is incremented. At the end of scan matrix is normalized to represent probability distribution. Using GLCM matrix various quantities can be computed such as entropy,energy, contrast and homogeneity, which characterize the texture of image.We can see GLCM is not rotation invariant because changing rotation $P(x_1, x_2)$ will change

Local Binary Pattern (LBP):

- LBP is statistical method to find rotation invariant texture characteristics in image. In LBP method rotation invariant histogram based feature vectors are generated.
- In LBP method representation is constructed by comparing each pixel with its surrounding neighborhood of pixels which lie at circle of radius R.

Neighboring pixel coordinate with respect to center is given by following formula

$$(\Delta u, \Delta v)_p = (R(\frac{cos(2\pi p)}{P}), R(\frac{sin(2\pi p)}{P})) \qquad p = 1, 2, 3 \dots P$$

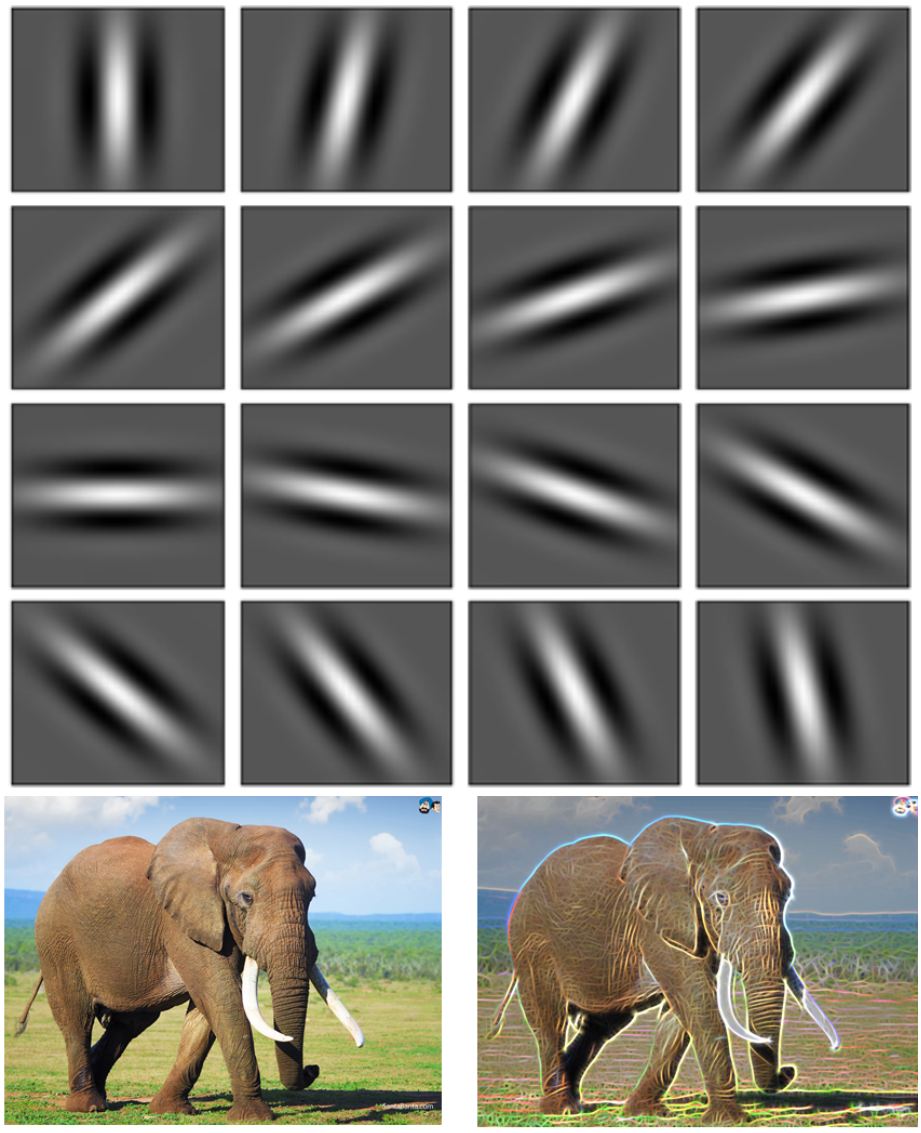
- Since Image is rectangular in nature to get neighboring pixel value bi-linear interpolation is used.
- All those neighboring pixels which are less than center are assigned as 0 and rest as 1. To make it rotation invariant this local binary pattern is rotated until we get minimum decimal value. Then in this rotated local binary pattern runs of zeros and ones are calculated and encoded into P+2 levels where 1 to P levels are correspond to number of ones in runs of zeros and ones, 0 level is when only zero runs and if more than two runs found it's assigned value is P+1.

Gabor Filter Family:

- Unlike LBP and GLCM , Gabor filter is structural method for extracting texture. Texture of an image can be characterized by spatial frequency and directionality by it's orientation. Gabor filters have this ability of localizing periodicity, micro patterns and it's orientation. It's impulse response is defined by sinusoidal modulated Gaussian which is given by:

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

- In frequency domain it would be convolution of impulse (Fourier transform of Cosine) with Gaussian function (Since frequency response of Gaussian is same) which can be visualized as two shifted Gaussian at the location of Cosine frequency.
- Bank Gabor filters are convolved with image at different orientation (Equation above just represent 0 degree orientation for different orientation-θ x will be transformed accordingly) which gives the highest response at edges at different orientation (As we can see the textures at following elephant image after applying Gabor filter banks of 16 filters).



a b

2:

a)Wrong.
b)Right.
c)Right.

Programming task

In this homework we need to extract Local Binary pattern based (LBP) feature vectors for each image and then using KNN classifier we need to classify images into 5 classes.

Calculating P -neighbors in the image:

In this step each pixel of image is taken and then compared with its surrounding neighborhood of P pixels which lie at circle of radius R. Neighboring pixel coordinate with respect to center is given by following formula

$$(\Delta u, \Delta v)_p = (R(\frac{cos(2\pi p)}{P}), R(\frac{sin(2\pi p)}{P})) \qquad p = 1, 2, 3 \dots P$$

Since Image is rectangular in nature to get neighboring pixel value bi-linear interpolation is used to calculate pixel value which does not on exact pixel coordinate:

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

Generating rotation invariant binary pattern:

All those neighboring pixels which are less than center are assigned as 0 and rest as 1. To make it rotation invariant this local binary pattern is rotated until we get minimum integer value corresponding to local binary pattern.

Encoding the rotation-invariant binary pattern

In this step rotated local binary pattern runs of zeros and ones are calculated and encoded into P+2 levels. Where 1 to P levels are correspond to number of ones in runs of zeros and ones only, 0 level is when only zero runs and if more than two runs found it's assigned value is P+1.

Histogram based Feature preparation:

To prepare a feature vector P+2 bins are created and corresponding bins is incremented while scanning the image. After full scan histogram is normalized so that it's independent of image shape.

KNN classifier on Euclidean metric:

Once LBP based histogram feature vectors are extracted for each training and test images, Euclidean distance is calculated for test vector with all the training vectors in space and K training vectors are chosen which are at minimum distance.Final classification of test vector is made by majority of class labels from K labels.

Comments

1- Overall accuracy is 68% for K =1 .Classwise accuracy for different K values is mentioned at the end of source code in inline jupyter text
2- We can see LBP histograms based feature are not much different from each other that's why getting accuracy more than 70 percent is difficult.
3- Another view to see why accuracy is not very good is , in terms of vector spaces images are very high dimensional and we have reduced it's dimension to just 10 which has cause loss of many feature characteristics.

Note

1- All images(histograms and corresponding images) are included below in Jupyter Inline plots and Observations are also mentioned.

Source Code, Images and few Observations (HW7)

Importing Libraries

In [71]:
```
from matplotlib import pyplot as plt
import numpy as np
import os
import scipy
import cv2
import pdb
from BitVector import BitVector
import time
import pickle
import scipy.stats
import seaborn as sn
import matplotlib.pyplot as plt
import itertools
```

Creating training pair with labels for all training images

In [125]:
```
training_path = "imagesDatabaseHW7/training"
testing_path = "imagesDatabaseHW7/testing"
P = 8

training_set = []
tranining_colot_set= []
class_dict={}
for label,class_dir in enumerate(os.listdir(training_path)):
    print("class of Image is {0}: label is {1}".format(class_dir,label))
    class_dict.update({class_dir:label})                                                         #Building dicti
onary for class labels
    for image in os.listdir(os.path.join(training_path,class_dir)):
        img = cv2.imread(os.path.join(training_path,class_dir,image))
        gray_image = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        training_set.append((gray_image,label))
        tranining_colot_set.append(img)
```

```
class of Image is beach: label is 0
class of Image is building: label is 1
class of Image is car: label is 2
class of Image is mountain: label is 3
class of Image is tree: label is 4
```

Creating LUT (of LBP encoding) indexed by decimal value of binary array for all binary permutaions of P bits

In [93]:
```
# Creating all binary pattern from 00000000 to 11111111
binary_pattern = list(map(list, itertools.product([0, 1], repeat=P)))

#Creating lookup table for Encoding values for all 256 (=2^P) Binary patterns so that don't need to rec
alculate same thing again and again
#Using bit vector module by Prof Avinash Kak for calculating number of runs and then assigning value fo
r each binary pattern

lut_encode = np.zeros((len(binary_pattern),1))
for index,item in enumerate(binary_pattern):
    bv = BitVector(bitlist = item)
    min_bit_int = min([int(bv<<1) for _ in range(P)])
    min_bit_vec = BitVector(intVal = min_bit_int, size = P)
    bit_vec_runs = min_bit_vec.runs()

    #Encoding Value
    if(len(bit_vec_runs)==1):
        lut_encode[index] = int(bit_vec_runs[0][0])*P
    elif(len(bit_vec_runs)>2):
        lut_encode[index] = P+1
    else:
        lut_encode[index] = len(bit_vec_runs[1])
LUT = np.squeeze(lut_encode).astype(int)
```

Function for local binary pattern of image

In [101]:
```
center_weight = (1-0.707)*(1-0.707)
opposite_weight = 0.707*0.707
side_weights = (1-0.707)*0.707

def create_lbp_hist(image):
    start_fun_time = time.time()
    h,w = image.shape
    hist = np.zeros((1,P+2))
    hist1 = np.zeros((1,P+2))
    for x in range(1,w-1):
        for y in range(1,h-1):
            frame = image[y-1:y+2,x-1:x+2]
            #Bilenear Interpolation
            center_val = center_weight*frame[1,1]
            p2 = center_val+side_weights*frame[1,2]+side_weights*frame[2,1]+opposite_weight*frame[2,2]
            p4 = center_val+side_weights*frame[1,2]+side_weights*frame[0,1]+opposite_weight*frame[0,2]
            p6 = center_val+side_weights*frame[1,0]+side_weights*frame[0,1]+opposite_weight*frame[0,0]
            p8 = center_val+side_weights*frame[1,0]+side_weights*frame[2,1]+opposite_weight*frame[2,0]
            lbp = np.array([frame[2,1],p2,frame[1,2],p4,frame[0,1],p6,frame[1,0],p8])

            #Making those value zero which are less than center value
            lbp = np.where(lbp>=frame[1,1],1,0)
            #Creating decimal value from binary pattern
            lookup_index = lbp.dot(2**np.arange(P)[::-1])
            hist[0,LUT[lookup_index]]+=1
    print("--- each image seconds --- {0} h,w {1},{2}".format((time.time() - start_fun_time),h,w))
    return hist
```

Creating histogram for each training image pair

In [ ]:
```
feature_vector_label_list = np.empty((0,2))
start_time = time.time()
for image,label in training_set:
    feature_vector_label_list = np.append(feature_vector_label_list,np.array([create_lbp_hist(image),la
bel]))
print("--- %s seconds ---" % (time.time() - start_time))
```

Pickling training feature to save time

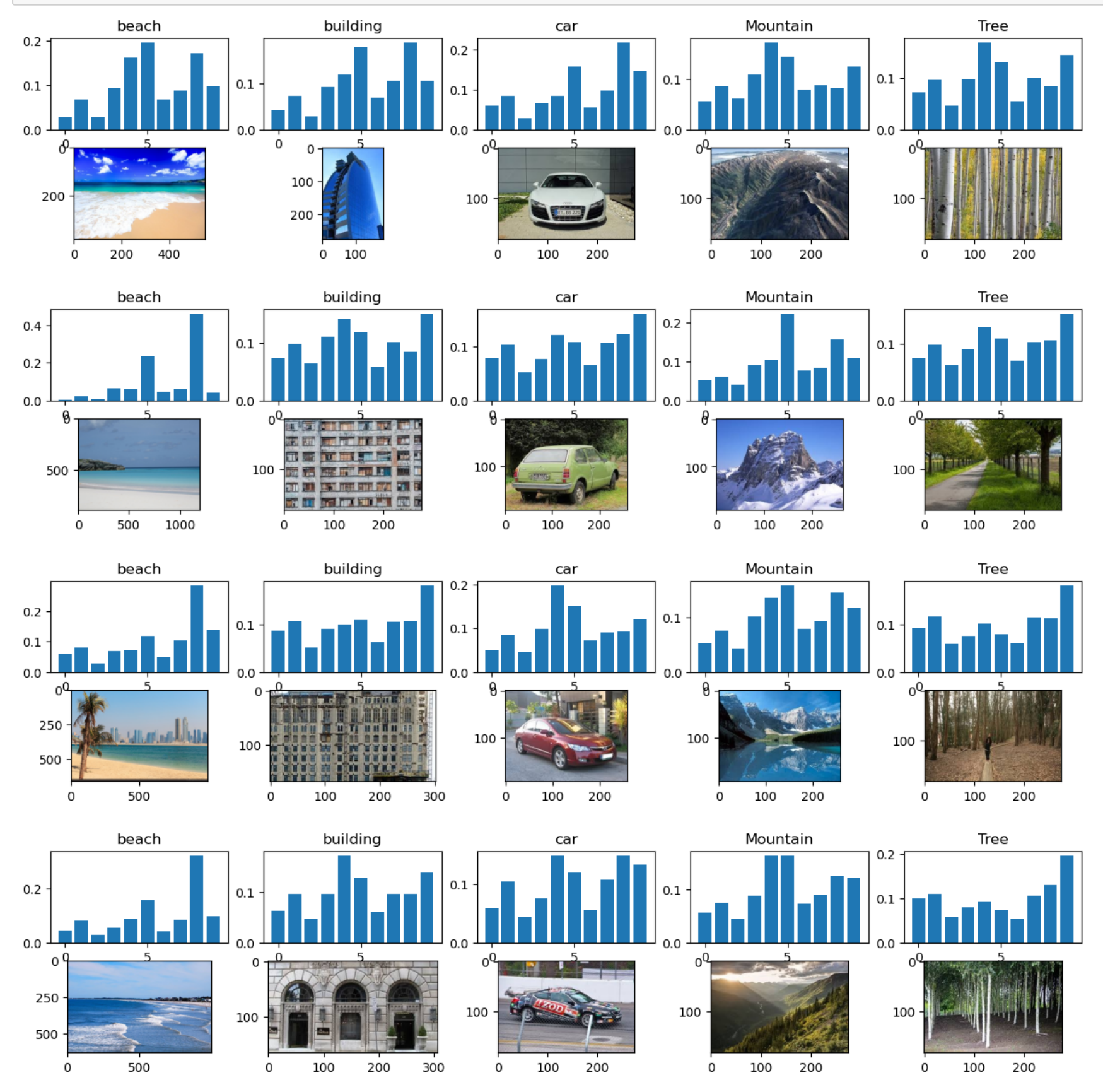
In [63]:
```
training_feature_file = "lbp_hist_data.pkl"
with open(training_feature_file,'wb') as f:
    pickle.dump(feature_vector_label_list, f)
```

Unpickling Training features

In [105]:
```
with open(training_feature_file,'rb') as f:
    x = pickle.load(f)

train_feature_vect = np.array([np.squeeze(item) for item in x[0::2]])
train_feature_vect = train_feature_vect/np.sum(train_feature_vect,axis=1)[:,np.newaxis]
train_label = np.array([np.squeeze(item) for item in x[1::2]])
```

Plotting Histograms

In [135]:
```
#Randomly plotting 3 set of Histograms
for i in [2,9,10,11]:
    fig=plt.figure(figsize=(15,3), dpi= 100, facecolor='w', edgecolor='k')

    plt.subplot(2, 5, 1)
    plt.bar(range(10),train_feature_vect[i])
    plt.title('beach')

    plt.subplot(2, 5, 2)
    plt.bar(range(10),train_feature_vect[20+i])
    plt.title('building')

    plt.subplot(2, 5, 3)
    plt.bar(range(10),train_feature_vect[40+i])
    plt.title('car')

    plt.subplot(2, 5, 4)
    plt.bar(range(10),train_feature_vect[60+i])
    plt.title('Mountain')

    plt.subplot(2, 5, 5)
    plt.title('Tree')
    plt.bar(range(10),train_feature_vect[80+i])


    img = tranining_colot_set[i]
    image_name = "image.jpg"
    cv2.imwrite(image_name,img)
    plot_image = cv2.imread(os.path.join(image_name))
    plot_image = cv2.cvtColor(plot_image,cv2.COLOR_BGR2RGB)
    plt.subplot(2, 5, 6)
    plt.imshow(plot_image)

    img = tranining_colot_set[20+i]
    image_name = "image.jpg"
    cv2.imwrite(image_name,img)
    plot_image = cv2.imread(os.path.join(image_name))
    plot_image = cv2.cvtColor(plot_image,cv2.COLOR_BGR2RGB)
    plt.subplot(2, 5, 7)
    plt.imshow(plot_image)


    img = tranining_colot_set[40+i]
    image_name = "image.jpg"
    cv2.imwrite(image_name,img)
    plot_image = cv2.imread(os.path.join(image_name))
    plot_image = cv2.cvtColor(plot_image,cv2.COLOR_BGR2RGB)
    plt.subplot(2, 5, 8)
    plt.imshow(plot_image)


    img = tranining_colot_set[60+i]
    image_name = "image.jpg"
    cv2.imwrite(image_name,img)
    plot_image = cv2.imread(os.path.join(image_name))
    plot_image = cv2.cvtColor(plot_image,cv2.COLOR_BGR2RGB)
    plt.subplot(2, 5, 9)
    plt.imshow(plot_image)

    img = tranining_colot_set[80+i]
    image_name = "image.jpg"
    cv2.imwrite(image_name,img)
    plot_image = cv2.imread(os.path.join(image_name))
    plot_image = cv2.cvtColor(plot_image,cv2.COLOR_BGR2RGB)
    plt.subplot(2, 5, 10)
    plt.imshow(plot_image)
```



Observation

- We can in above hitograms they have some similarity within class
- In first row image as we can see building have sky and lot's of beach kind of similarity that's why it's histogram is similar to beach but in all other three example it is not.

Creating list of test images

In [67]:
```
testing_set = []
for image in os.listdir(testing_path):
        img = cv2.imread(os.path.join(testing_path,image))
        gray_image = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        testing_set.append((gray_image,class_dict[image.split("_")[0]]))
```

Creating histogram for each testing image pair

In [ ]:
```
test_feature_vector_label_list = np.empty((0,2))
start_time = time.time()
for image,label in testing_set:
    test_feature_vector_label_list = np.append(test_feature_vector_label_list,np.array([create_lbp_hist
(image),label]))
```

Pickling testing feature to save time

In [69]:
```
testing_feature_file = "lbp_hist_data_test.pkl"
with open(testing_feature_file,'wb') as f:
    pickle.dump(test_feature_vector_label_list, f)
```

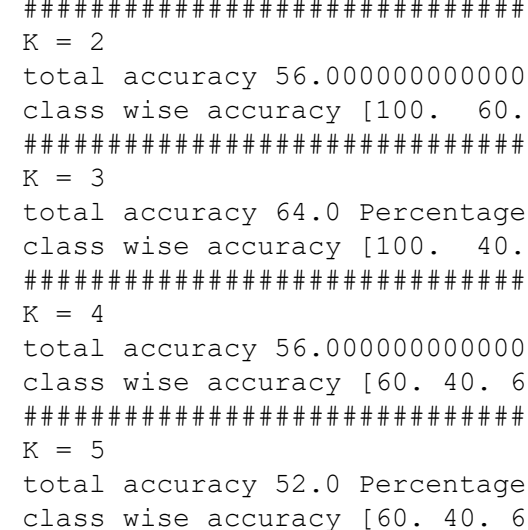
Unpickling Testing features

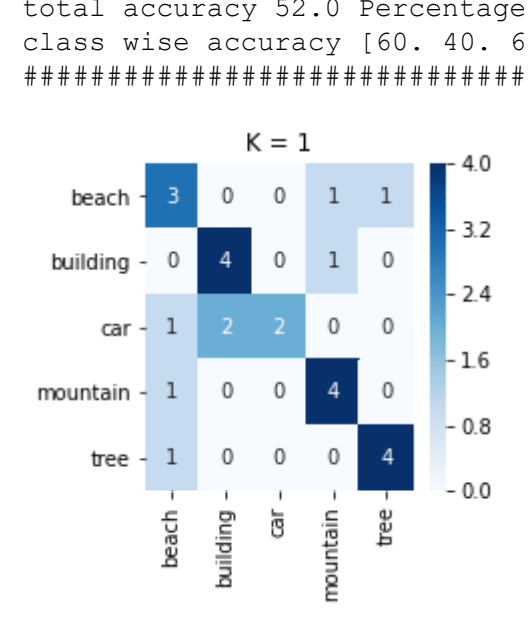
In [ ]:
```
with open(testing_feature_file,'rb') as f:
    x_test = pickle.load(f)

test_feature_vect = np.array([np.squeeze(item) for item in x_test[0::2]])
test_feature_vect = test_feature_vect/np.sum(test_feature_vect,axis=1)[:,np.newaxis]
test_label = np.array([np.squeeze(item) for item in x_test[1::2]])
```

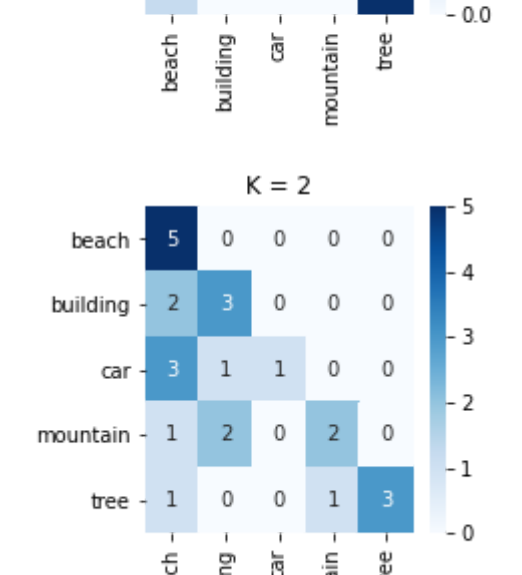
Running KNN

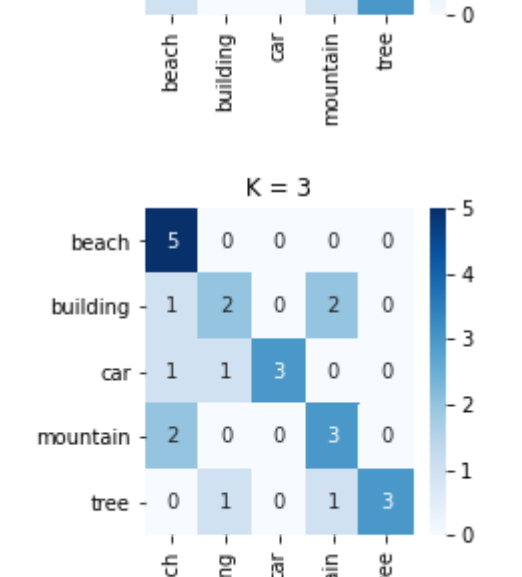
In [138]:
```
for K in range(1,6):
    pred_label_true_label_pair = np.empty((0,2))
    for test_vector,true_label in  zip(test_feature_vect,test_label):
        #Calculating Euclidean distance for test vector with each training vector
        dist= np.linalg.norm(train_feature_vect-test_vector[np.newaxis,:],axis=1)
        #Choosing K best neighbor
        k_nearest_index = np.argsort(dist)[:K]
        k_nearest_label = train_label[k_nearest_index]
        pred_label_true_label_pair = np.vstack((pred_label_true_label_pair,np.array([scipy.stats.mode(
k_nearest_label)[0][0],true_label])))

    #Creating confusion matrix
    confusion_matrix = np.zeros((5,5))
    for item in pred_label_true_label_pair.astype(int):
        confusion_matrix[item[1],item[0]]+=1
    #Plotting Confusion Matrix
    plt.figure(figsize = (3,3))
    plt.title("K = {0}".format(K))
    sn.heatmap(confusion_matrix, annot=True, cmap="Blues",xticklabels=list(class_dict.keys()), ytickla
bels=list(class_dict.keys()))

    total_accuracy = (np.sum(np.diag(confusion_matrix))/25)*100
    class_wise_accuracy = ((np.diag(confusion_matrix))/5)*100

    print("K = {0}".format(K))
    print("total accuracy {0} Percentage".format(total_accuracy))
    print("class wise accuracy {0} in percentage".format(class_wise_accuracy))
    print("###############################################################")
```

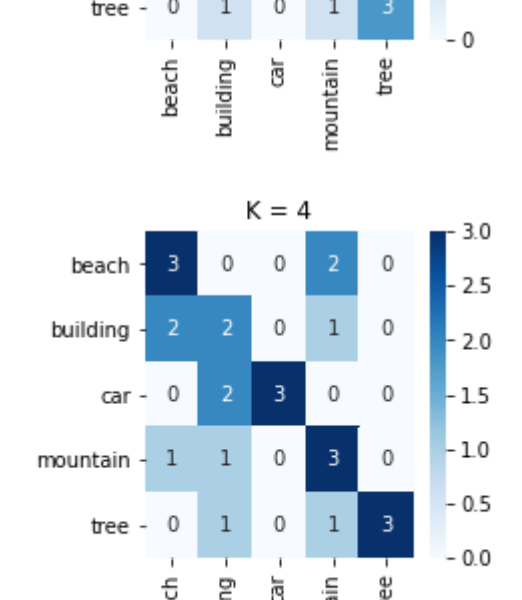
```
K = 1
total accuracy 68.0 Percentage
class wise accuracy [60. 80. 40. 80. 80.] in percentage
###############################################################
K = 2
total accuracy 56.00000000000001 Percentage
class wise accuracy [100.   60.   20.   40.   60.] in percentage
###############################################################
K = 3
total accuracy 64.0 Percentage
class wise accuracy [100.   40.   60.   60.   60.] in percentage
###############################################################
K = 4
total accuracy 56.00000000000001 Percentage
class wise accuracy [60. 40. 60. 60. 60.] in percentage
###############################################################
K = 5
total accuracy 52.0 Percentage
class wise accuracy [60. 40. 60. 40. 60.] in percentage
###############################################################
```











Observations

- We can see above few observations easily:
- Mountains are getting clssified as beaches due to stone and sky kind of objects in both
- In many cases buildings are clssified as beaches and mountains which are due to sky kind of charectristics
- We can see LBP histograms based feature are not much different from each other that's why getting accuracy more than 70 percent is difficult.
- Another view to see why accuracy is not very good is , in terms of vector spaces images are very high dimensional and we have reduced it's dimension to just 10 which has cause loss of many feature charectristics.

References

[1] Avinash Kak: Measuring Texture and Color in Images,
`https://engineering.purdue.edu/kak/Tutorials/TextureAndColor.pdf`

[2] Avinash Kak: Bit-Vector Module ,
`https://engineering.purdue.edu/kak/dist/BitVector-3.4.9.html`

[3] Texture Segmentation Using Gabor Filtersr,
`By Khaled Hammouda`

[4] Through The Eyes of Gabor Filter (Medium article for image),
`https://medium.com/@anuj`$_s hah/through - the - eyes - of - gabor - filter - 17d1fdb3ac97$