ECE 661 Homework 8

Rohan Contractor

October 2020

Name: Rohan Contractor
Email: rcontrac@purdue.edu

1 Introduction

The objective for this homework is to perform Texture characterization using the Gram matrix technique to represent the textures.

2 Implementation Description:

2.1 Gram Matrix:

For applying the Gram matrix technique of texture characterization, I have first resized all images to (256,256). After resizing, I have convolved the image with a (3,3) size kernel. The output of this convolution was then downsampled to size (128,128). The convolution was repeated for C=5,10,15 different kernels which were randomly generated. The kernels have float weights and lie in the interval [-1,1]. This process was done for both the train and the test datasets. This convolution generated 5,10,15 different channels. Thus for each image C channels were generated which were each vectorized to a column vector. These column vectors were then stacked into a matrix to represent one image. This is the feature map that we use to classify the images. The gram matrix was obtained by multiplying this stacked matrix with it's transpose. Since this gram matrix is symmetric, only the upper triangular part of the matrix was used to represent an image.

2.2 Support Vector Machine

For classifying the images of the test set with the help of their gram matrix representations, I used the Scikit-learn SVM model. The gram matrix representation of the training images was taken and the model was trained after splitting the data into a 75-25 train and validation sets. The program was iterated over 25 times generating random convolution kernels each time to get the best accuracy.

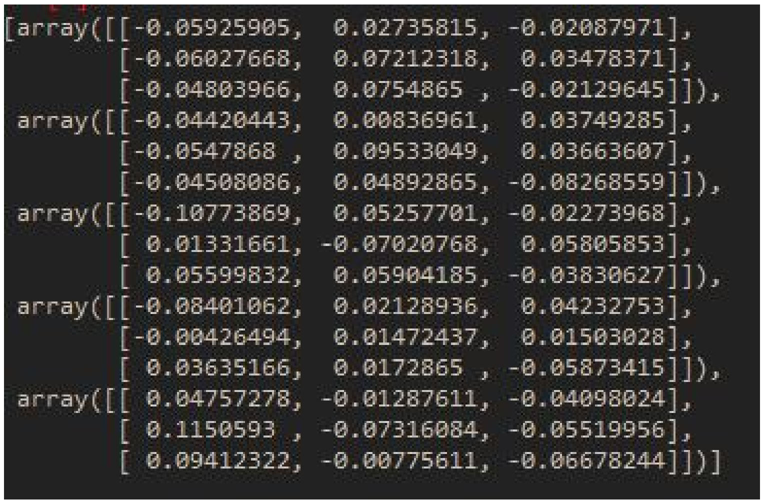
The model which had the highest accuracy on the validation set was stored using pickle and was run on the test set to get the classification of output images. The parameters that I have set are C = 0.1 and max_iter = 10000. The kernel used was the radial basis function and gamma was set to scale.

3 Observations:

I have run the program for C = 5,10,15 and have run the model 25 times, picking the iteration with the max value of the accuracy for the validation set. Since the kernel is always initialised with random weights it is hard to ascertain a a general trend. I have seen instances where there has been an increase in accuracy with increase in C, but have also seen a dip in accuracy in one such run before increasing again. Accuracy is dependant on how the weights of the kernel are set and how many iterations the program runs for. The more the number of iterations the program is run for the better the chances of getting a higher accuracy. Looking at the overall results I would say that with increase in C the accuracy would increase as more textures would be represented and there would be finer detail present which would aid classification. The amount of downsampling also affects the classification. The more the downsampling the more accuracy decreases. I had obtained worse accuracy when I had downsampled to (64,64) and hence raised it to (128,128). Radial Basis function kernel was the quickest and provided best results. I experimented with values of C and maximum iterations to see what provided a good result. I settled on setting C = 0.1 and iter = 10000. Validation set accuracy results were fairly good and often were close to that of the accuracy of the model on the test set.

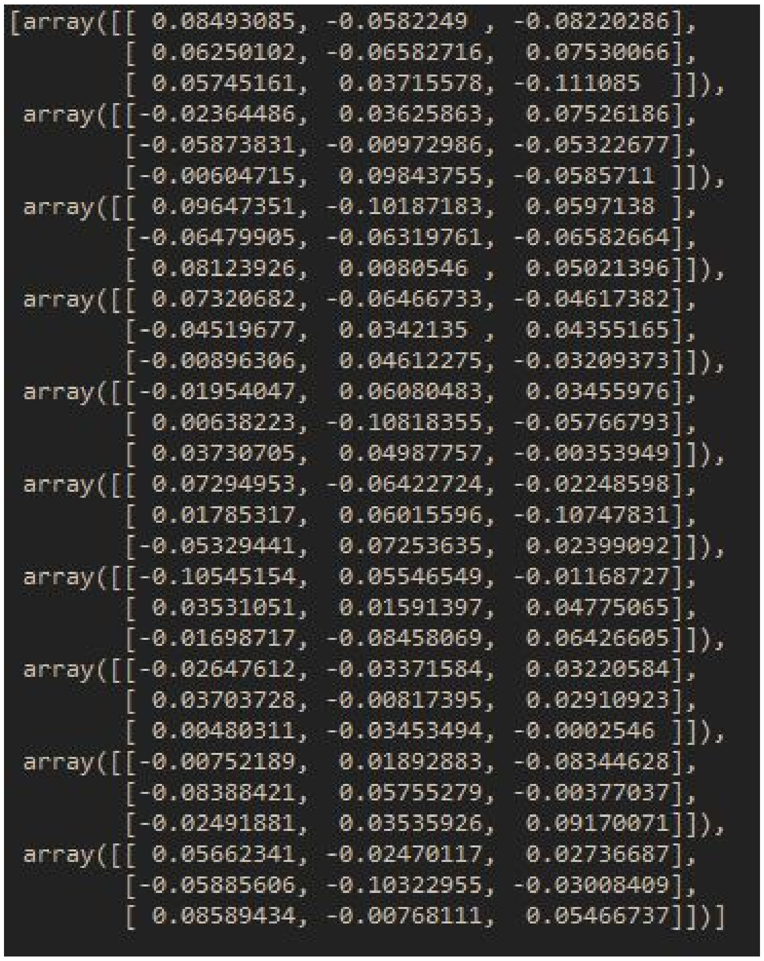
4 Performance Metrics:

The convolution kernels used for the various channels are shown here. Increasing the number of channels and the number of iterations leads to a better accuracy, number of iterations being the more dominant factor. For each channel the testing and validation accuracy's are printed along with the convolutional kernels. In some cases the validation accuracy was better than that of the test accuracy and the some cases he opposite.The accuracy of the model on the training sets is also close to that of the test sets. The classes are :-
1 - Cloudy
2 - Rain
3- Shine
4- Sunrise



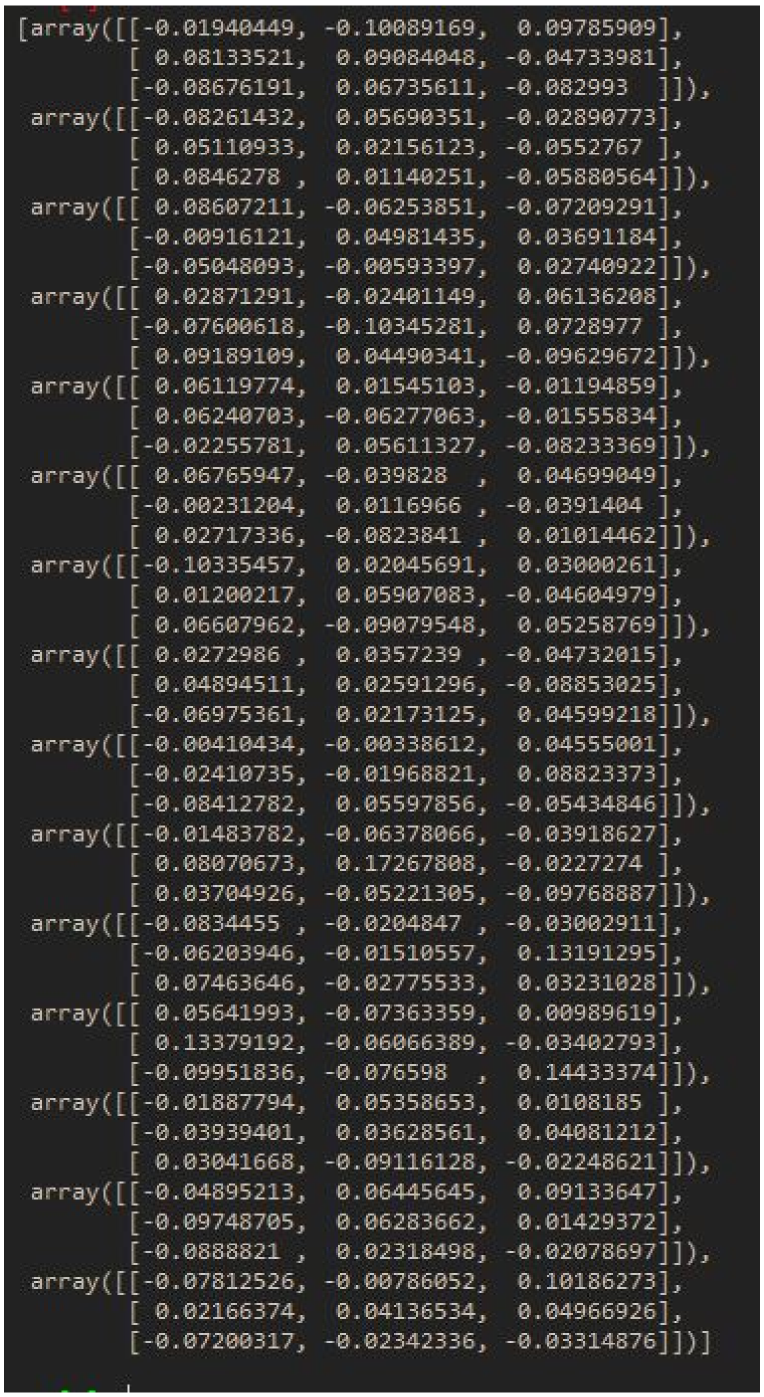
```
[array([[-0.05925905,  0.02735815, -0.02087971],
        [-0.06027668,  0.07212318,  0.03478371],
        [-0.04803966,  0.0754865 , -0.02129645]]),
 array([[-0.04420443,  0.00836961,  0.03749285],
        [-0.0547868 ,  0.09533049,  0.03663607],
        [-0.04508086,  0.04892865, -0.08268559]]),
 array([[-0.10773869,  0.05257701, -0.02273968],
        [ 0.01331661, -0.07020768,  0.05805853],
        [ 0.05599832,  0.05904185, -0.03830627]]),
 array([[-0.08401062,  0.02128936,  0.04232753],
        [-0.00426494,  0.01472437,  0.01503028],
        [ 0.03635166,  0.0172865 , -0.05873415]]),
 array([[ 0.04757278, -0.01287611, -0.04098024],
        [ 0.1150593 , -0.07316084, -0.05519956],
        [ 0.09412322, -0.00775611, -0.06678244]])]
```

Figure 1: Convolution kernels for C= 5



```
[array([[ 0.08493085, -0.0582249 , -0.08220286],
        [ 0.06250102, -0.06582716,  0.07530066],
        [ 0.05745161,  0.03715578, -0.111085  ]]),
 array([[-0.02364486,  0.03625863,  0.07526186],
        [-0.05873831, -0.00972986, -0.05322677],
        [-0.00604715,  0.09843755, -0.0585711 ]]),
 array([[ 0.09647351, -0.10187183,  0.0597138 ],
        [-0.06479905, -0.06319761, -0.06582664],
        [ 0.08123926,  0.0080546 ,  0.05021396]]),
 array([[ 0.07320682, -0.06466733, -0.04617382],
        [-0.04519677,  0.0342135 ,  0.04355165],
        [-0.00896306,  0.04612275, -0.03209373]]),
 array([[-0.01954047,  0.06080483,  0.03455976],
        [ 0.00638223, -0.10818355, -0.05766793],
        [ 0.03730705,  0.04987757, -0.00353949]]),
 array([[ 0.07294953, -0.06422724, -0.02248598],
        [ 0.01785317,  0.06015596, -0.10747831],
        [-0.05329441,  0.07253635,  0.02399092]]),
 array([[-0.10545154,  0.05546549, -0.01168727],
        [ 0.03531051,  0.01591397,  0.04775065],
        [-0.01698717, -0.08458069,  0.06426605]]),
 array([[-0.02647612, -0.03371584,  0.03220584],
        [ 0.03703728, -0.00817395,  0.02910923],
        [ 0.00480311, -0.03453494, -0.0002546 ]]),
 array([[-0.00752189,  0.01892883, -0.08344628],
        [-0.08388421,  0.05755279, -0.00377037],
        [-0.02491881,  0.03535926,  0.09170071]]),
 array([[ 0.05662341, -0.02470117,  0.02736687],
        [-0.05885606, -0.10322955, -0.03008409],
        [ 0.08589434, -0.00768111,  0.05466737]])]
```

Figure 2: Convolution kernels for C= 10



```
[array([[-0.01940449, -0.10089169,  0.09785909],
        [ 0.08133521,  0.09084048, -0.04733981],
        [-0.08676191,  0.06735611, -0.082993  ]]),
 array([[-0.08261432,  0.05690351, -0.02890773],
        [ 0.05110933,  0.02156123, -0.0552767 ],
        [ 0.0846278 ,  0.01140251, -0.05880564]]),
 array([[ 0.08607211, -0.06253851, -0.07209291],
        [-0.00916121,  0.04981435,  0.03691184],
        [-0.05048093, -0.00593397,  0.02740922]]),
 array([[ 0.02871291, -0.02401149,  0.06136208],
        [-0.07600618, -0.10345281,  0.0728977 ],
        [ 0.09189109,  0.04490341, -0.09629672]]),
 array([[ 0.06119774,  0.01545103, -0.01194859],
        [ 0.06240703, -0.06277063, -0.01555834],
        [-0.02255781,  0.05611327, -0.08233369]]),
 array([[ 0.06765947, -0.039828  ,  0.04699049],
        [-0.00231204,  0.0116966 , -0.0391404 ],
        [ 0.02717336, -0.0823841 ,  0.01014462]]),
 array([[-0.10335457,  0.02045691,  0.03000261],
        [ 0.01200217,  0.05907083, -0.04604979],
        [ 0.06607962, -0.09079548,  0.05258769]]),
 array([[ 0.0272986 ,  0.0357239 , -0.04732015],
        [ 0.04894511,  0.02591296, -0.08853025],
        [-0.06975361,  0.02173125,  0.04599218]]),
 array([[-0.00410434, -0.00338612,  0.04555001],
        [-0.02410735, -0.01968821,  0.08823373],
        [-0.08412782,  0.05597856, -0.05434846]]),
 array([[-0.01483782, -0.06378066, -0.03918627],
        [ 0.08070673,  0.17267808, -0.0227274 ],
        [ 0.03704926, -0.05221305, -0.09768887]]),
 array([[-0.0834455 , -0.0204847 , -0.03002911],
        [-0.06203946, -0.01510557,  0.13191295],
        [ 0.07463646, -0.02775533,  0.03231028]]),
 array([[ 0.05641993, -0.07363359,  0.00989619],
        [ 0.13379192, -0.06066389, -0.03402793],
        [-0.09951836, -0.076598  ,  0.14433374]]),
 array([[-0.01887794,  0.05358653,  0.0108185 ],
        [-0.03939401,  0.03628561,  0.04081212],
        [ 0.03041668, -0.09116128, -0.02248621]]),
 array([[-0.04895213,  0.06445645,  0.09133647],
        [-0.09748705,  0.06283662,  0.01429372],
        [-0.0888821 ,  0.02318498, -0.02078697]]),
 array([[-0.07812526, -0.00786052,  0.10186273],
        [ 0.02166374,  0.04136534,  0.04966926],
        [-0.07200317, -0.02342336, -0.03314876]])]
```

Figure 3: Convolution kernels for C= 15

5 Comparison with LBP (Extra Credit):

From the confusion matrix we see that the LBP method performs better. I have used the KNN method to compare the images according to the LBP texture representations and have found that for k=5 a maximum accuracy of 70 percent is achieved. As for the gram matrix method we have a maximum accuracy of 55 percent for C = 5. This is due to the weights used in the gram matrix method being randomly selected and this would require a large number of iterations to get a good accuracy. With an increase in C and the number of iterations we see that the accuracy increases. In the case of the gram matrix method we see that the cloudy and shine images are getting mis-classified which is causing the accuracy to reduce. For LBP as well the shine image is getting mis-clasified as either cloudy or sunshine.

6 Sample Input Images:

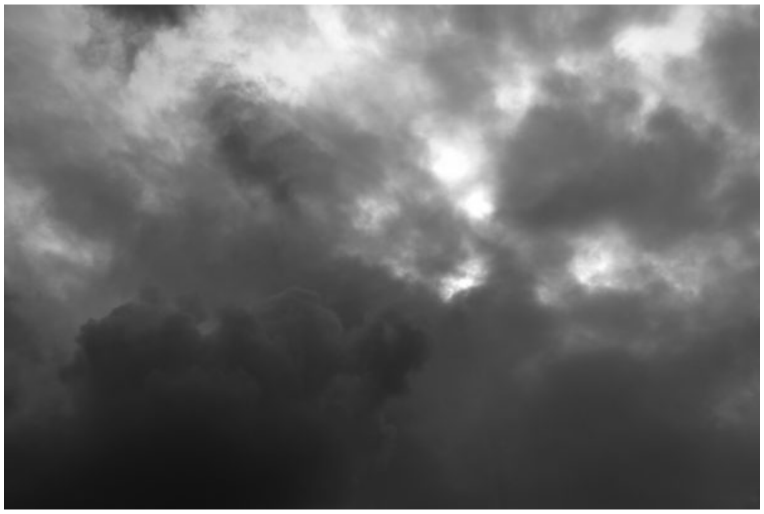


Figure 4: Cloudy

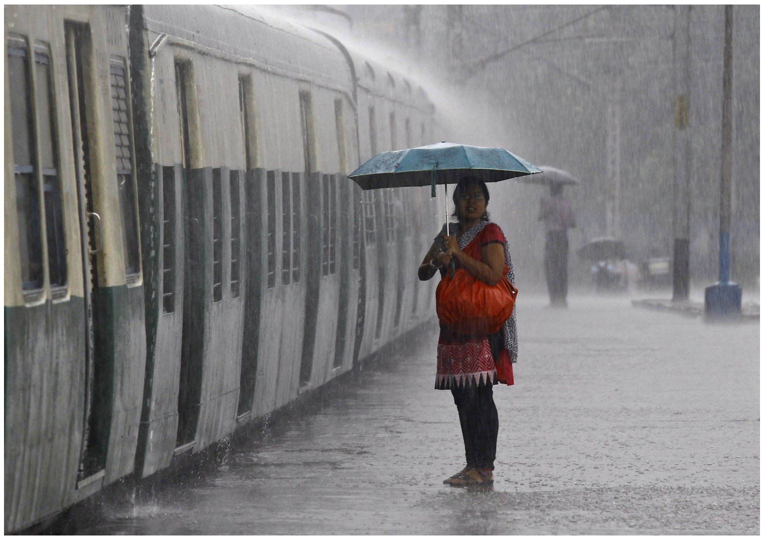


Figure 5: Rain

Figure 6: Shine

Figure 7: Sunrise

7 Output:

7.1 Gram Matrix

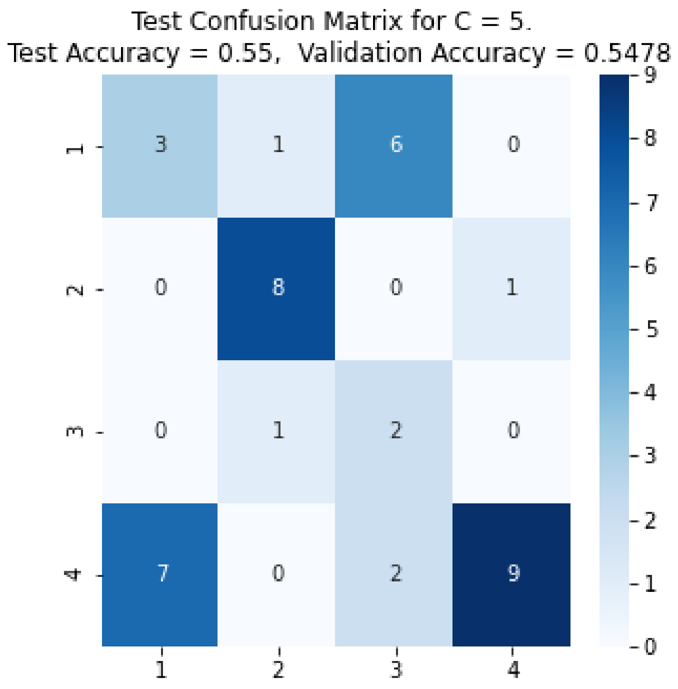


Figure 8: Confusion matrix for C=5

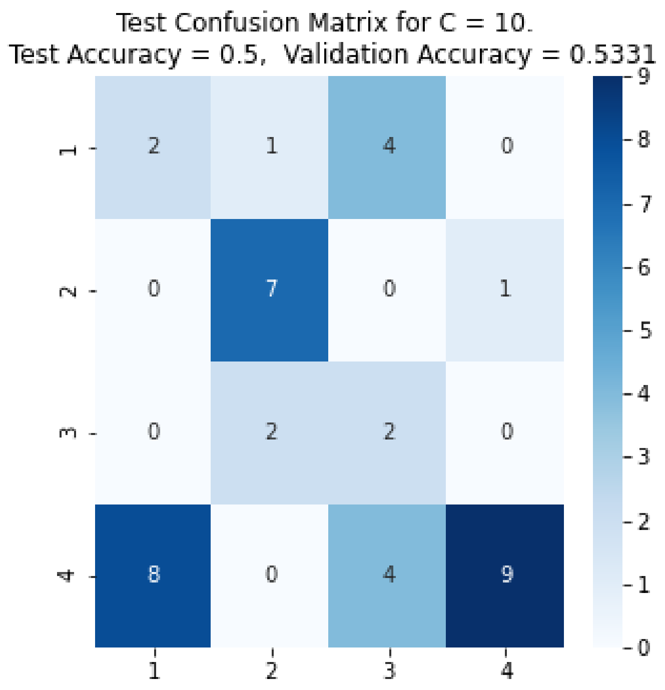
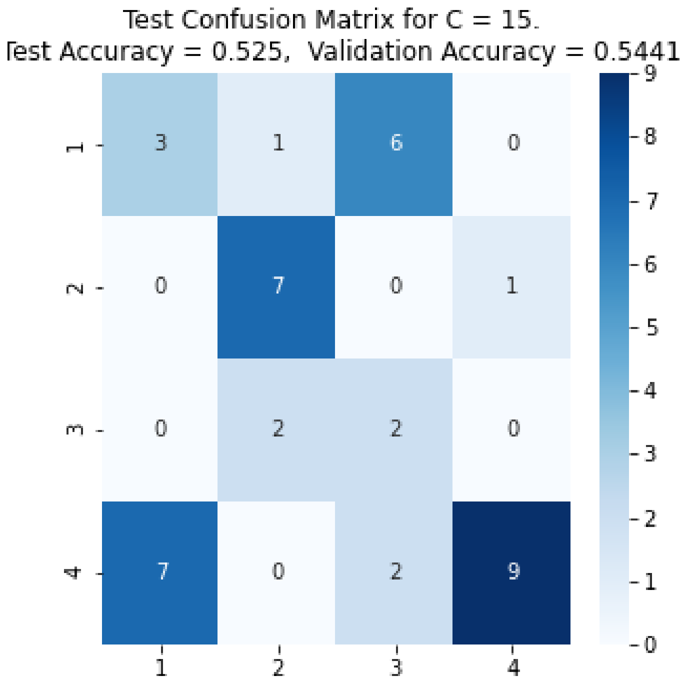


Figure 9: Percentage confusion matrix for C=10



	1	2	3	4
1	3	1	6	0
2	0	7	0	1
3	0	2	2	0
4	7	0	2	9

Figure 10: Confusion matrix for C=15

7.2 Local Binary Pattern

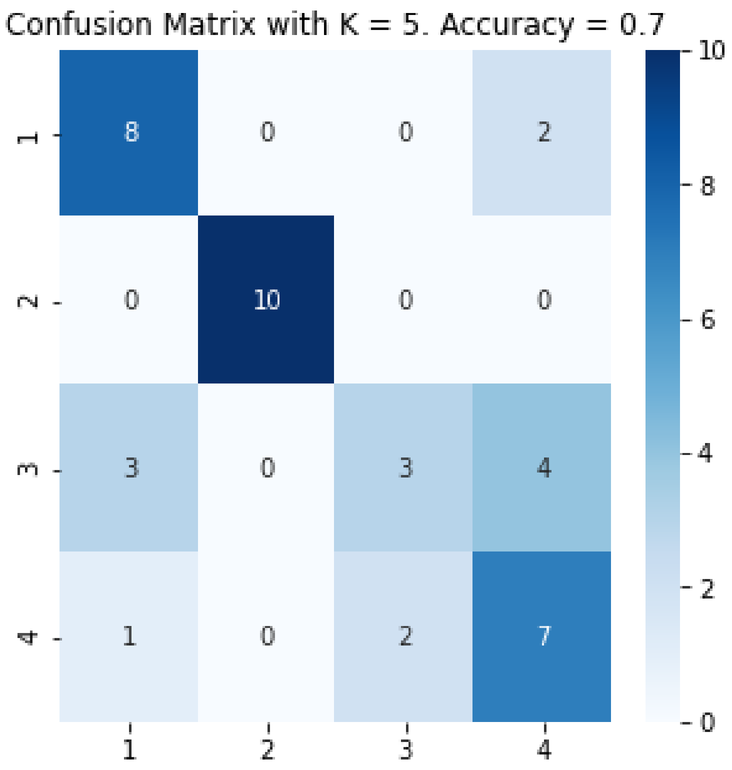


Figure 11: Confusion matrix for K=5

8 Source Code:

8.1 Gram Matrix Output:

```
import numpy as np
import cv2
import os
import pdb
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
import re
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import pickle

# Generating random convolutional operator
def gen_kern(C):
    C_list = []
    for i in range(C):
        a = np.random.randint(1,1000000,(3,3))
        b = a/np.sum(a)
        b = b-1/9
        C_list.append(b)
    return C_list

path = os.getcwd()
testing = os.path.join(path + "\\imagesDatabaseHW8\\testing")
training = os.path.join(path + "\\imagesDatabaseHW8\\training")
max_accuracy = 0

# Looping to get all images
K = 256 # Resizing to ensure same size for all images

#Resizing the images and creating the train/test images
trainlist = []
train_class_list=[]
for path in os.listdir(training):
    a = cv2.imread(os.path.join(training +'\\' + path))
    gray = cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)
    try:
        b = cv2.resize(gray,(K,K), interpolation = cv2.INTER_NEAREST)
    except:
        print(os.path.join(training +'\\'+ path))
    match = re.match(r"([a-z]+)([0-9]+)",path,re.I)
    if match:
        train_class_list.append(match.groups()[0])
    trainlist.append(b)

testlist = []
test_class_list=[]
for path in os.listdir(testing):
    a = cv2.imread(os.path.join(testing +'\\' + path))
    gray = cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)
    try:
        b = cv2.resize(gray,(K,K), interpolation = cv2.INTER_NEAREST)
    except:
        print(os.path.join(testing +'\\'+ path))
    match = re.match(r"([a-z]+)([0-9]+)",path,re.I)
    if match:
        test_class_list.append(match.groups()[0])
    testlist.append(b)

train_class_list =
     np.asarray(train_class_list).reshape(len(train_class_list),1)
test_class_list =
     np.asarray(test_class_list).reshape(len(test_class_list),1)
train_class_list[np.where(train_class_list == 'cloudy')] = 1
train_class_list[np.where(train_class_list == 'rain')] = 2
train_class_list[np.where(train_class_list == 'shine')] = 3
train_class_list[np.where(train_class_list == 'sunrise')] = 4

test_class_list[np.where(test_class_list == 'cloudy')] = 1
test_class_list[np.where(test_class_list == 'rain')] = 2
test_class_list[np.where(test_class_list == 'shine')] = 3
test_class_list[np.where(test_class_list == 'sunrise')] = 4


C = 15 #No of channels # Change to 5,10,15,20
K1 = 128 #Resize amount for downsampling
accuracy = 0
for i in range(25):
    c_list = gen_kern(C)
    seperate_images_train = []
    for images in trainlist:
        train_conv_out = []
        for kernel in c_list:
            temp = convolve2d(images,kernel)
            downsamp = cv2.resize(temp,(K1,K1),interpolation =
                cv2.INTER_NEAREST)
            downsamp = downsamp.flatten().reshape(K1**2,1)
            train_conv_out.append(downsamp)
        seperate_images_train.append(train_conv_out)

    gram_train = []
    for array in seperate_images_train:
        mat = np.asarray(array).reshape(C,K1**2)
        gram = np.dot(mat,mat.T)
        gram = gram[np.triu_indices(C)]
        gram_train.append(gram)

    #Validation
    x_train,x_valid,y_train,y_valid =
        train_test_split(gram_train,train_class_list, test_size =
        0.25,random_state=42)
    clf = svm.SVC(kernel = 'rbf', gamma = 'scale', C = 0.1,max_iter =
        10000)
    clf.fit(x_train,y_train.ravel())
    pred = clf.predict(x_valid)
    acc = accuracy_score(pred,y_valid.ravel())
    param = np.array((c_list,clf,acc),dtype = object)

    if acc > accuracy:
        accuracy = acc
        test_file_name = 'param.pkl'
        with open(test_file_name,'wb') as f:
            pickle.dump(param,f)

with open('param.pkl','rb') as f:
    param = pickle.load(f)

c_list = param[0]
seperate_images_test = []
for images in testlist:
    test_conv_out = []
    for kernel in c_list:
        temp = convolve2d(images,kernel)
        downsamp = cv2.resize(temp,(K1,K1),interpolation =
            cv2.INTER_NEAREST)
        downsamp = downsamp.flatten().reshape(K1**2,1)
        test_conv_out.append(downsamp)
    seperate_images_test.append(test_conv_out)

gram_test = []
for array in seperate_images_test:
    mat = np.asarray(array).reshape(C,K1**2)
    gram = np.dot(mat,mat.T)
    gram = gram[np.triu_indices(C)]
    gram_test.append(gram)

clf = param[1]
pred = clf.predict(gram_test)
acc = accuracy_score(pred,test_class_list.ravel())
cmat = confusion_matrix(pred,test_class_list)
total_accuracy = np.trace(cmat)/len(test_class_list)
plt.figure(figsize=(5,5));
plt.title("Test Confusion Matrix for C = {}. \n Test Accuracy = {},
    Validation Accuracy = {:1.4f}".format(C,total_accuracy,param[2]));
sns.heatmap((cmat),annot = True,cmap = "Blues",xticklabels =
    list([1,2,3,4]),yticklabels = list([1,2,3,4]),fmt='g');
plt.savefig('Test Confusion Matrix for C = {}'.format(C))
```

8.2 Extra Credit

```
import cv2
import numpy as np
import itertools
from BitVector import BitVector
import pdb
import re
import matplotlib.pyplot as plt
import os
import seaborn as sns
import pickle
from scipy import stats

# Generating a table for encoding all 256 possible combos
binary = list(map(list,itertools.product([0,1], repeat = 8)))
table = np.zeros((len(binary),1))
for ind,vec in enumerate(binary):
    bitvec = BitVector(bitlist = vec)
    temp = [int(bitvec << 1) for j in range((len(vec)))]
    minbitvec = BitVector(intVal = min(temp), size =len(vec))
    multi = minbitvec.runs()
    if len(multi)>2:
        table[ind] = 9
    elif(len(multi) == 1 and multi[0][0] == '1'):
        table[ind] = 8
    elif(len(multi) == 1 and multi[0][0] == '0'):
        table[ind] = 0
    else:
        table[ind] = len(multi[1])
lookup = np.squeeze(table.astype(int))

def lbp(img):
    hist = np.zeros((1,10))
    for x in range(1,img.shape[1]-1):
        for y in range(1,img.shape[0]-1):
            kern = img[y-1:y+2, x-1:x+2]
            #Direct Values
            p1 = kern[2,1]
            p3 = kern[1,2]
            p5 = kern[0,1]
            p7 = kern[1,0]
            # Bilinear Interpolation
            p2 = ((1-0.707)**2)*kern[1,1] + (1-0.707)*0.707*kern[1,2] +
                (1-0.707)*0.707*kern[2,1]+kern[2,2]*(0.707)**2
            p4 = ((1-0.707)**2)*kern[1,1] + (1-0.707)*0.707*kern[1,2] +
                (1-0.707)*0.707*kern[0,1]+kern[0,2]*(0.707)**2
            p6 = ((1-0.707)**2)*kern[1,1] + (1-0.707)*0.707*kern[1,0] +
                (1-0.707)*0.707*kern[0,1]+kern[0,0]*(0.707)**2
            p8 = ((1-0.707)**2)*kern[1,1] + (1-0.707)*0.707*kern[1,0] +
                (1-0.707)*0.707*kern[2,1]+kern[2,0]*(0.707)**2
            #Generating binary vector
            vec = (np.array((p1,p2,p3,p4,p5,p6,p7,p8)) >=
                kern[1,1]).astype(int)
            ind = vec.dot(2**np.arange(8)[::-1])
            hist[0,lookup[ind]] += 1
    return hist[0]

def training():
    K = 256
    path = os.getcwd()
    training = os.path.join(path + "\\imagesDatabaseHW8\\training")
    trainlist = []
    train_class_list = []
    for path in os.listdir(training):
        a = cv2.imread(os.path.join(training +'\\'+ path))
        gray = cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)
        gray = cv2.resize(gray,(K,K), interpolation = cv2.INTER_NEAREST)
        b = lbp(gray)
        match = re.match(r"([a-z]+)([0-9]+)",path,re.I)
        if match:
            train_class_list.append(match.groups()[0])
        trainlist.append(b)

    # Changing the class types to integers from strings
    train_class_list =
        np.asarray(train_class_list).reshape(len(train_class_list),1)
    index1 = np.where(train_class_list == 'cloudy')
    index2 = np.where(train_class_list == 'rain')
    index3 = np.where(train_class_list == 'shine')
    index4 = np.where(train_class_list == 'sunrise')
    train_class_list[index1] = 1
    train_class_list[index2] = 2
    train_class_list[index3] = 3
    train_class_list[index4] = 4
    (train_class_list).astype(int)
    train = tuple(zip(trainlist,train_class_list))
    return train

#for testing
def testing():
    K = 256
    path = os.getcwd()
    testing = os.path.join(path + "\\imagesDatabaseHW8\\testing")
    testlist = []
    test_class_list=[]
    for path in os.listdir(testing):
        a = cv2.imread(os.path.join(testing +'\\' + path))
        gray = cv2.cvtColor(a,cv2.COLOR_BGR2GRAY)
        gray = cv2.resize(gray,(K,K), interpolation = cv2.INTER_NEAREST)
        b = lbp(gray)
        match = re.match(r"([a-z]+)([0-9]+)",path,re.I)
        if match:
            test_class_list.append(match.groups()[0])
        testlist.append(b)

    # Changing the class types to integers from strings
    test_class_list =
         np.asarray(test_class_list).reshape(len(test_class_list),1)
    index1 = np.where(test_class_list == 'cloudy')
    index2 = np.where(test_class_list == 'rain')
    index3 = np.where(test_class_list == 'shine')
    index4 = np.where(test_class_list == 'sunrise')
    test_class_list[index1] = 1
    test_class_list[index2] = 2
    test_class_list[index3] = 3
    test_class_list[index4] = 4
    (test_class_list).astype(int)
    test = tuple(zip(testlist,test_class_list))
    return test

def knn(train,test,k):

    #For train unpacking:
    hist1 = np.empty((1,10))
    classes1 = np.empty((1,1))
    for i in train:
        hist1 = np.vstack((hist1,i[0]))
        classes1 = np.vstack((classes1,int(i[1][0])))
    hist1 = hist1[1:]
    classes1 = classes1[1:]

    #For test unpacking:
    hist2 = np.empty((1,10))
    classes2 = np.empty((1,1))
    for i in test:
        hist2 = np.vstack((hist2,i[0]))
        classes2 = np.vstack((classes2,int(i[1][0])))
    hist2 = hist2[1:]
    classes2 = classes2[1:]

    # Euclidean Distance knn
    t = np.empty((1,1))
    p = np.empty((1,1))
    for ind,item in enumerate(hist2):
            diff = hist1 - item
            dist = np.linalg.norm(diff, axis = 1)
            sorted_diff = np.argsort(dist)
            nearest = sorted_diff[0:k]
            prediction = classes1[nearest]
            pred_class = stats.mode(prediction)[0][0]
            actual_class = classes2[ind]
            t = np.append(t,pred_class)
            p = np.append(p,actual_class)
    t = t[1:].reshape(len(t[1:]),1)
    p = p[1:].reshape(len(p[1:]),1)
    final = (t==p).astype(int)

    #confusion matrix generation
    cmat = np.zeros((4,4))
    final = np.hstack((p,t)).astype(int)
    for iteration in final:
        cmat[iteration[0]-1,iteration[1]-1] += 1
    total_accuracy = np.trace(cmat)/40
    plt.figure(figsize=(5,5));
    plt.title("Confusion Matrix with K = {}. Accuracy =
        {}".format(k,total_accuracy));
    sns.heatmap((cmat),annot = True,cmap = "Blues",xticklabels =
        list([1,2,3,4]),yticklabels = list([1,2,3,4]),fmt='g');
    return None

# ##For training. Have used pickle so have commented
# lbp_texture_train = training()
# test_file_name = 'trainfile256.pkl'
# with open(test_file_name,'wb') as f:
#     pickle.dump(lbp_texture_train,f)

# ##For testing. Have used pickle so have commented
# lbp_texture_test = testing()
# test_file_name = 'testfile256.pkl'
# with open(test_file_name,'wb') as f:
#     pickle.dump(lbp_texture_test,f)

with open('trainfile256.pkl','rb') as f:
    x_array = pickle.load(f)
with open('testfile256.pkl','rb') as f:
    y_array = pickle.load(f)

for k in range(1,6):
    knn(x_array,y_array,k)
```

9 References

1. Avinash Kak, *Measuring Texture and Color in Images.* Oct 13,2020.
2. Avinash Kak, BitVector, https://engineering.purdue.edu/kak/dist/BitVector-3.4.9.html