FALL 22 ECE 661 – COMPUTER VISION

Homework 7

Sahithi Kodali – 34789866
kodali1@purdue.edu

THEORY QUESTION

1. The reading material for Lecture 16 presents three different approaches to characterizing the texture in an image:

1) using the Gray Level Co-Occurrence Matrix (GLCM);

2) with Local Binary Pattern (LBP) histograms; and

3) using a Gabor Filter Family.

Explain succinctly the core ideas in each of these three methods for measuring texture in images.

Gray Level Co-Occurrence Matrix (GLCM):

GLCM method of characterizing texture is a statistical method based on second order statistical properties. In GLCM, the texture is characterized using the joint probability distribution of the gray level pixels in an image. This is given as P (x_1, x_2) where x_1 is a gray level of a pixel and x_2 is the gray level of a pixel that is at a distance 'd' as an offset from x_1.

The images are scanned and if the two gray scale values at these pixels that are distance by 'd' are same, the GLCM matrix at the pixel value location (p_1, p_2) is incremented. The matrix position (p_2, p_1) is also updated depending on the importance of the order of the gray scale level of the pixels. Once the matrix is completely populated after the whole scan, the probability distribution is computed by normalizing the GLCM matrix.

This rotation invariant matrix is used in estimating the texture characteristics of an image like entropy, contrast, homogeneity and energy etc.

Local Binary Pattern (LBP) Histograms:

LBP method of characterizing texture is also a statistical method that estimates the texture using the neighbouring pixels of a specific pixel. LBP generates histogram-based feature descriptors that gives rotation invariant characteristics.

The method works by computing the gray levels of the 'P' number of neighbourhood pixels in a radius 'R' of each pixel in an image using bi-linear interpolation. The computed pixels are compared to the centre pixel as threshold and a binary vector of 0's and 1's is obtained.

The binary vector obtained is shifted in a circular direction to obtain the minimum integer value form since it contains the most discriminative texture information of the image and makes the image rotation invariant. The value obtained is converted to integer format as a pixel texture measure based on the runs in the binary vector and a histogram of all the texture information is built. The shape of the histogram stands as a feature vector of the image texture.

Gabor Filter Family:

Gabor Filter method is a structural method that utilizes image periodicities to characterize texture in an image. Gabor Filter Family utilizes a set of filters and frequency bands to determine the image periodicities as different directions and different frequencies. It utilizes the Fourier transform of cosine along with the Gaussian function for convolution of filters with the image. The impulse response obtained is a sinusoidal modulated Gaussian and is formulated for computing the impulse values.

Such Gabor filters are used to convolve an image in different orientations and frequencies to obtain the highest responding textural information. The results obtained from these several filter convoluted images are combined based on the filter to obtain a value that can characterize the texture of an image effectively.

2. With regard to representing color in images, answer Right or Wrong for the following questions and provide a brief justification for each (no more than two sentences):

(a) RGB and HSI are just linear variants of each other.

Wrong. The RGB and HSI transformation can be said as nonlinear. Such conversion from RGB to an HSI image is complex due to the transformation being controlled by white space alignment and requirement that the RGB vectors need to pass through three colors in HSI cylinder.

(b) The color space L*a*b* is a nonlinear model of color perception.

Right. These color spaces are based on the Opponent Color Modelling of human color perception like darkening different colors to get different colors making it non-linear based on color.

(c) Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.

Right. Based on changes in the illumination the color of the surfaces reflects in different ways making it harder to measure the true color of surface via camera or human perception since both are sensitive to color contrast/reflection due to illumination changes.

RESULTS

Using LBP Histogram:

The LBP Histogram feature vector for one image in each of the four classes are shown below.

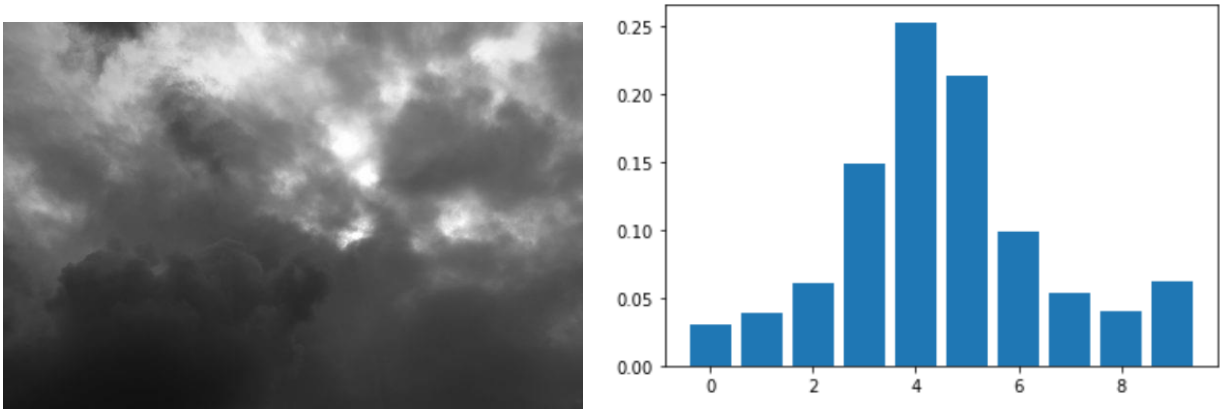

Figure – 1: LBP Histogram feature vector for an image in class 'cloudy'

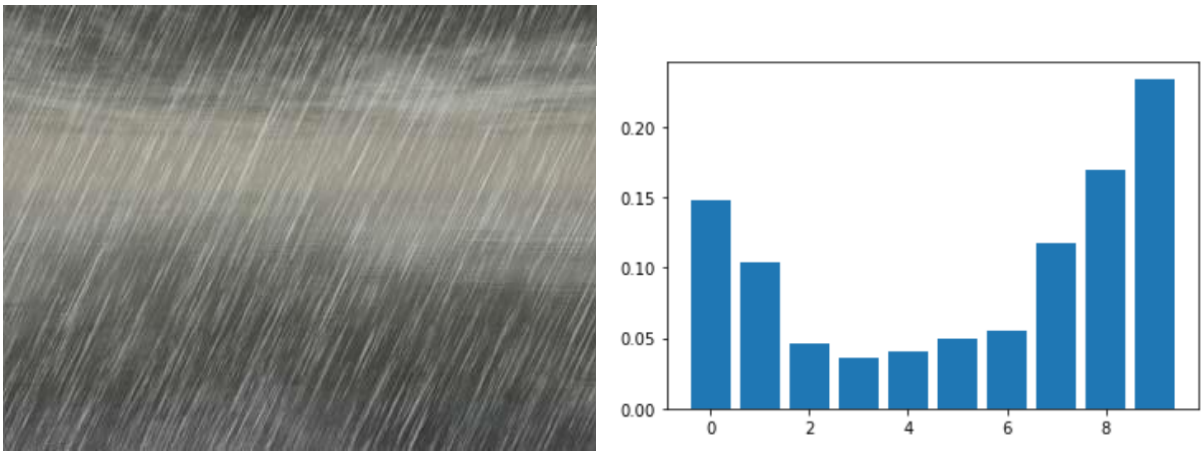

Figure – 2: LBP Histogram feature vector for an image in class 'rain'

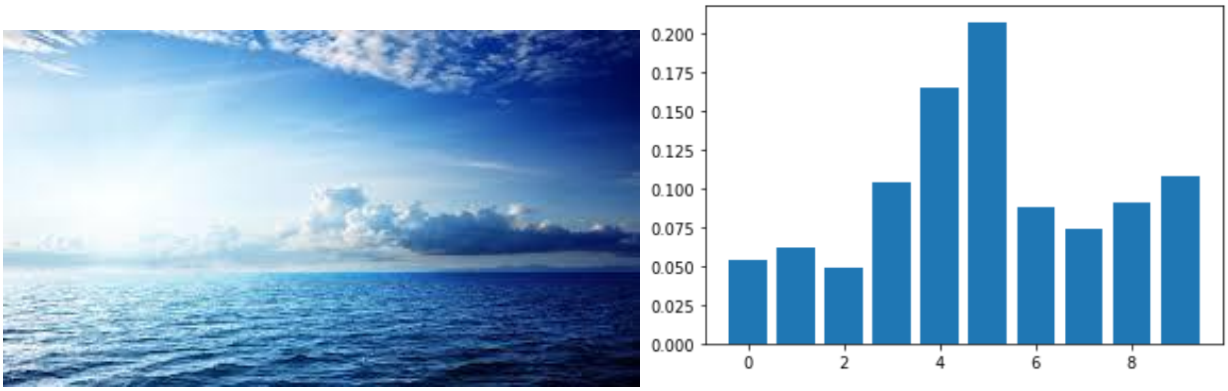

Figure – 3: LBP Histogram feature vector for an image in class 'shine'

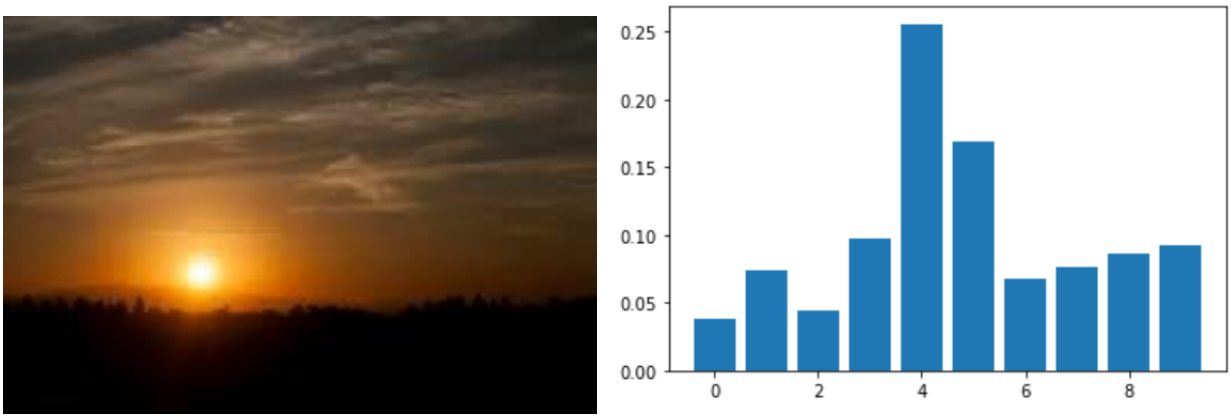

Figure – 4: LBP Histogram feature vector for an image in class 'sunrise'

The best metrics for classification tasks using LBP method are obtained for P=8 and R=1 parameters with an accuracy of 71% and the confusion matrix as shown in the figure 5.

```
accuracy = accuracy_score(pred, test_labels_LBP.ravel())
print(accuracy)
```

```
0.715
```

```
conf_mat = confusion_matrix(pred, test_labels_LBP)
sns.heatmap(conf_mat, annot = True, cmap = 'Blues')
```

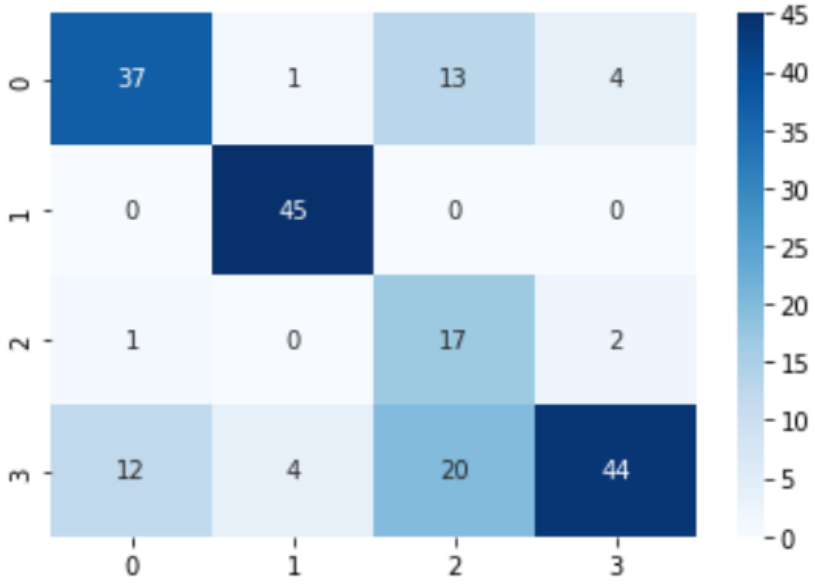

Figure – 5: Accuracy and confusion matrix obtained using LBP descriptor and SVM classifier

Using Gram Matrix:

The Gram matrix feature vector for one image in each of the four classes are shown below.

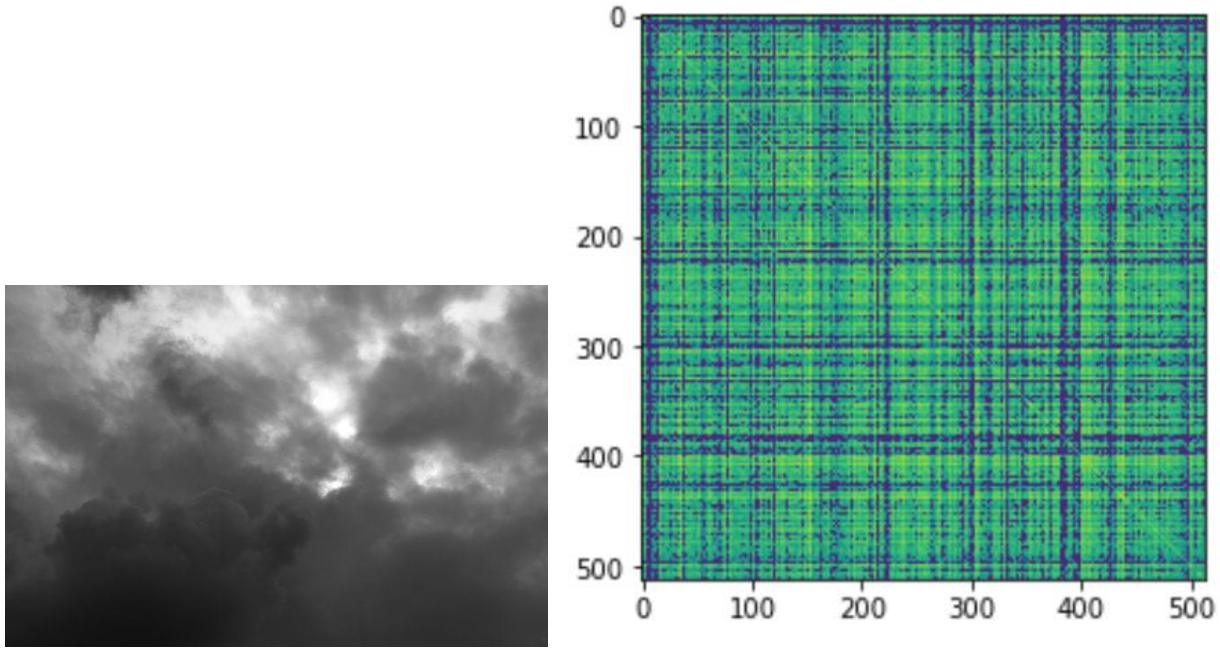

Figure – 6: Gram matrix feature vector for an image in class 'cloudy'

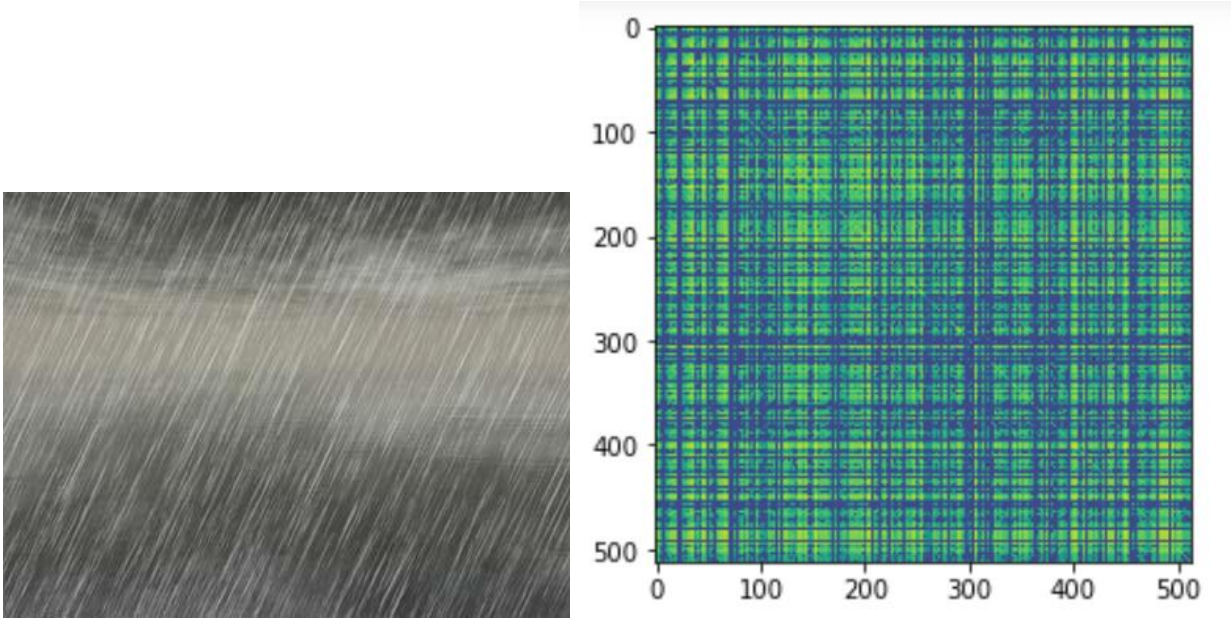

Figure – 7: Gram matrix feature vector for an image in class 'rain'

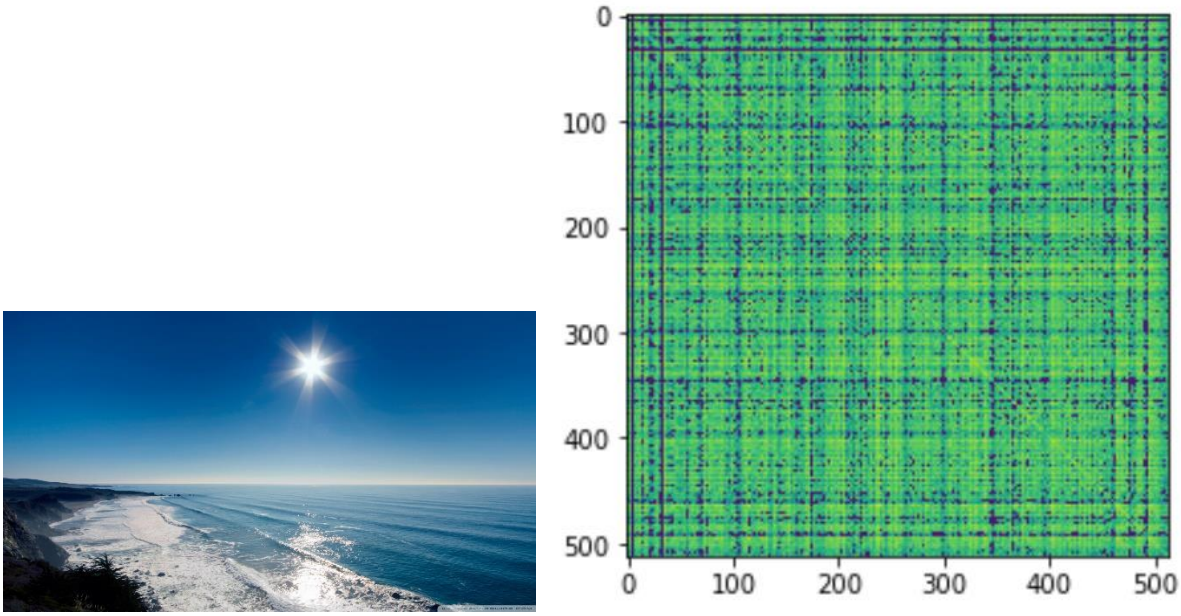

Figure – 8: Gram matrix feature vector for an image in class 'shine'

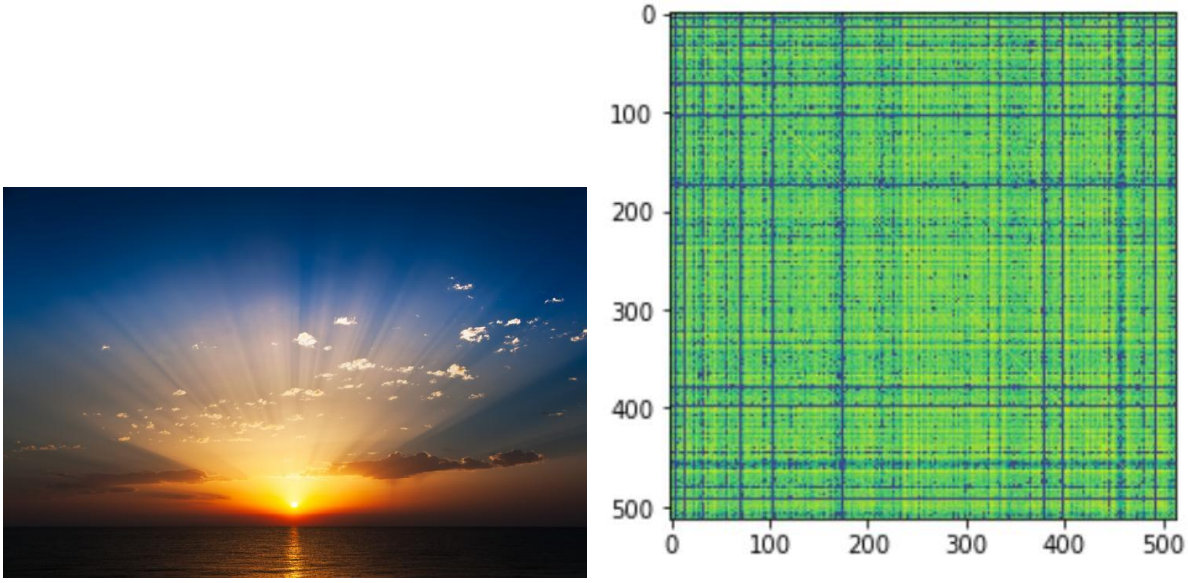

Figure – 9: Gram matrix feature vector for an image in class 'sunrise'

The best metrics for classification tasks using Gram matrix are obtained for parameter C = 320 with an accuracy of 98% and the confusion matrix as shown in the figure 10.

```
print(accuracy)
```

```
0.98
```

```
conf_mat_gram320 = confusion_matrix(pred_gram_320, test_lab
sns.heatmap(conf_mat_gram320, annot = True, cmap = 'Blues')
```

```
<AxesSubplot:>
```

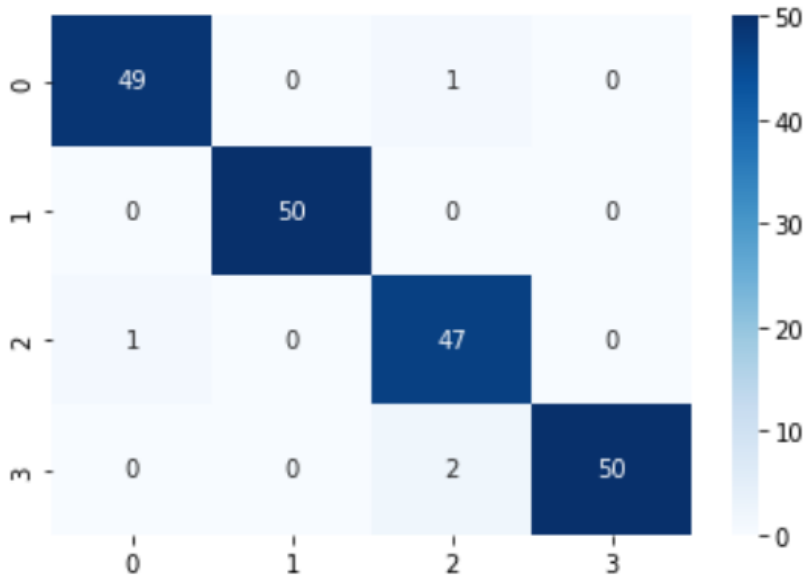

Figure – 10: Confusion matrix obtained using gram matrix and SVM classifier for C = 320

We can also observe all the diagonal values of the matrix close to 50 or 50 showing the effective results using Gram matrix method.

For the other C value experimented the metric results are as follows,

C = 15, Accuracy = 76% and confusion matrix as in Figure 11

C = 120, Accuracy = 94% and confusion matrix as in Figure 12

```
print(accuracy)
```

```
0.765
```

```
conf_mat_gram15 = confusion_matrix(pred_gram_15
sns.heatmap(conf_mat_gram15, annot = True, cmap
```

```
<AxesSubplot:>
```

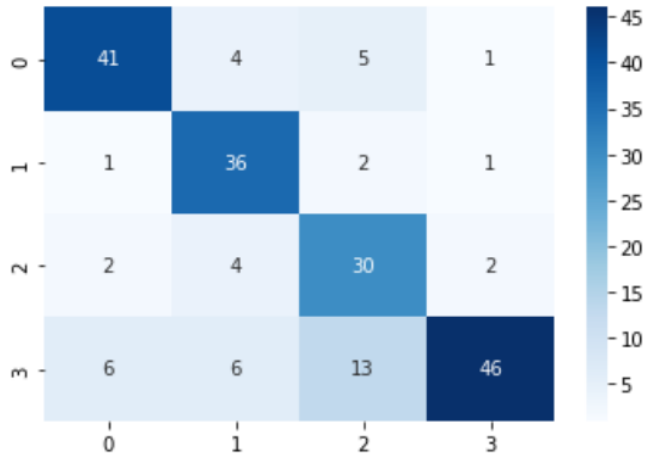

Figure – 11: Confusion matrix obtained using gram matrix and SVM classifier for C = 15

```
print(accuracy)
```

```
0.94
```

```
conf_mat_gram120 = confusion_matrix(pred_gram_
sns.heatmap(conf_mat_gram120, annot = True, cm
```

```
<AxesSubplot:>
```

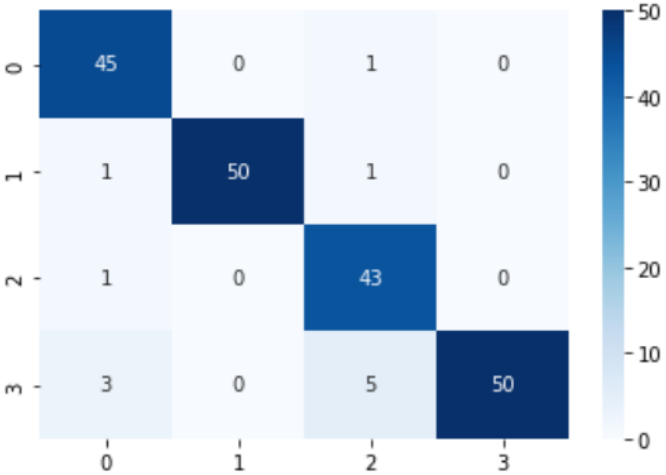

Figure – 12: Confusion matrix obtained using gram matrix and SVM classifier for C = 120

Observations:

We can observe that the LBP histograms for each class image have differences in the texture plots. The histograms obtained within same class have similar texture plots when analysed. Similarly, the Gram matrix of the images in different classes have different texture plots while similar class images have a slightly similar plot.

The accuracy of 71% is good enough for the LBP, however still less compared to even the worst accuracy (when C = 15) obtained using Gram Matrix. A 98% accuracy has been achieved (when C = 320) using Gram Matrix and further finetuning can result in better accuracy as well. This shows how the neural network has dominated the results of LBP in predicting the texture characteristics accurately.

The confusion matrix can also be looked into above, which shows the almost accurate results obtained using Gram matrix in all cases when compared to the LBP method.

SOURCE CODE:

```
#!/usr/bin/env python
# coding: utf-8

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 7 </center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# In[1]:


# Import libraries needed

import numpy as np
import matplotlib.pyplot as plt
import skimage.io as sk
import cv2
import math
import copy
import pandas as pd
import os
import PIL
import re
import itertools
import BitVector
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
```

```
# In[159]:


#P and R values for LBP
P = 8
R = 1
# R = 2


# In[160]:


#data paths
folder_path = "D:\Purdue\ECE661_CV\HW7\HW7-Auxilliary\data"
train_path = os.path.join( folder_path + "\\training")
test_path = os.path.join( folder_path + "\\testing")


# In[161]:


#load images from the folder
def load_images_from_folder(path, img_size):
    images = []
    images_class = []

    #read image
    for img_name in os.listdir(path):
        img = cv2.imread(os.path.join(path + '\\' + img_name))

        #resize image
        try:
            img_resized = cv2.resize(img, (img_size, img_size), interpolation =
cv2.INTER_NEAREST)
        except:
            print(os.path.join(path + '\\' + img_name))

        #match the images name to regex to extract labels and names of images
        img_name_match = re.match(r"([a-z]+)([0-9]+)", img_name, re.IGNORECASE)

        if img_name_match:
            images_class.append(img_name_match.groups()[0])
            images.append(img_resized)
```

```
    return images, images_class


# In[162]:


#get train and test data for LBP
train_data_LBP, train_class_LBP = load_images_from_folder(train_path, 64)
test_data_LBP, test_class_LBP = load_images_from_folder(test_path, 64)


# In[163]:


#get train and test data for VGG-Gram matrix method
train_data_VGG, train_class_VGG = load_images_from_folder(train_path, 256)
test_data_VGG, test_class_VGG = load_images_from_folder(test_path, 256)


# In[164]:


#convert class names to numerical labels
def convert_labels(class_list):
    class_list = np.asarray(class_list).reshape(len(class_list),1)
    class_list[np.where(class_list == 'cloudy')] = 1
    class_list[np.where(class_list == 'rain')] = 2
    class_list[np.where(class_list == 'shine')] = 3
    class_list[np.where(class_list == 'sunrise')] = 4

    return class_list


# In[165]:


#get train and test data labels
train_labels_LBP = convert_labels(train_class_LBP)
test_labels_LBP = convert_labels(test_class_LBP)
train_labels_VGG = convert_labels(train_class_VGG)
test_labels_VGG = convert_labels(test_class_VGG)


# ### LBP method

# In[166]:
```

```
#Function to get the binary vector
def get_binaryVector(R, P, frame):

    #set neighbourhood offsets
    deltaVec = np.zeros((P,2))

    for p in range(P):
        deltaVec[p][0] = R*math.sin(2*math.pi*p/P)
        deltaVec[p][1] = R*math.cos(2*math.pi*p/P)

    deltaVec[abs(deltaVec) < 0.0001] = 0

    #compute the binary vector
    bv = np.zeros(len(deltaVec))
    center_thresh = frame[1,1]

    #perform rotation and computer interpolation binary vector for each pixel
point in the neighbourhood
    rotate = 0
    for i in range(len(deltaVec)):
        delta_x = deltaVec[i][0]
        delta_y = deltaVec[i][1]

        if (i%2 == 0 and i != 0):
            rotate -= 1

        #perform interpolation for the adjacent pixels using offset
        p1 = frame[math.floor(1+delta_y), math.floor(1+delta_x)]
        p2 = frame[math.floor(1+delta_y), math.ceil(1+delta_x)]
        p3 = frame[math.ceil(1+delta_y), math.floor(1+delta_x)]
        p4 = frame[math.ceil(1+delta_y), math.ceil(1+delta_x)]

        adj_mat = np.array([[p1,p2], [p3,p4]])
        rot_mat = np.rot90(adj_mat, rotate)

        dx = abs(delta_x)
        dy = abs(delta_y)

        bv[i] = ( (1-dy)*(1-dx)*rot_mat[0,0] + (1-dy)*(dx)*rot_mat[0,1] +
                 (dy)*(1-dx)*rot_mat[1,0] + (dy)*(dx)*rot_mat[1,1] )

    #return the boolean vector with 1 and 0's obtained using the center pixel
value as threshold
```

```
    return (bv >= center_thresh).astype(int)


# In[167]:


#compute the histogram probability distribution for an image
def LBP_hist(image, R, P):

    #convert to gray scale
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

    h,w = image.shape

    #give histogram size
    hist = np.zeros(P + 2)
    num_pix = 0

    for i in range(1, w-1):
        for j in range(1, h-1):

            frame = image[j-1:j+2, i-1:i+2]

            #compute the binary vectors for the window frame
            bv_pattern = get_binaryVector(R, P, frame)

            #find the minimum binary vector
            bv = BitVector.BitVector(bitlist = bv_pattern)
            min_circular_shifts_intval = min([int(bv<<1) for _ in range(P)])
            min_bv = BitVector.BitVector(intVal = min_circular_shifts_intval,
size = P)

            #find the encoding of minimum binary vector
            runs = min_bv.runs()

            if len(runs) == 2:
                encoding = len(runs[1])
            elif len(runs) > 2:
                encoding = P + 1
            elif sum(min_bv) == 0:
                encoding = 0
            elif sum(min_bv) == P:
                encoding = P
            else:
                encoding = None
```

```
            #update histogram based on encoding and the count of pixels
            hist[encoding] += 1
            num_pix += 1

    #get histogram distribution probabilities
    lbp_hist = hist/num_pix
    return lbp_hist



# In[197]:


#plot histogram for any image (change according to the image that needs to be
visualized)
exp = LBP_hist(train_data_LBP[450], R, P)
plt.bar(range(10),exp)


# In[172]:


#obtain histogram based feature vectors for train data
train_featVect_LBP = np.zeros((len(train_data_LBP),10))

for i in range(len(train_data_LBP)):
    hist = LBP_hist(train_data_LBP[i], R, P)
    train_featVect_LBP[i,:] = hist


# In[173]:


#save the feature vectors
np.save('D:\Purdue\ECE661_CV\HW7\\feature.npy', train_featVect_LBP)
# train_LBP_loaded = np.load('D:\Purdue\ECE661_CV\HW7\\feature.npy')


# In[174]:


#obtain histogram based feature vectors for test data
test_featVect_LBP = np.zeros((len(test_data_LBP),10))

for i in range(len(test_data_LBP)):
```

```
    hist = LBP_hist(test_data_LBP[i], R, P)
    test_featVect_LBP[i,:] = hist


# In[105]:


#save the feature vectors

np.save('D:\Purdue\ECE661_CV\HW7\\feature_test.npy', test_featVect_LBP)


# In[102]:


#import, initialize and fit/train the SVM classifier
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(train_featVect_LBP, train_labels_LBP.ravel())


# In[112]:


#predict the class labels and chech the accuracy with test data labels
from sklearn.metrics import accuracy_score, confusion_matrix

pred = classifier.predict(test_featVect_LBP)

accuracy = accuracy_score(pred, test_labels_LBP.ravel())
print(accuracy)


# In[121]:


#plot a confusion matrix as a heat map
conf_mat = confusion_matrix(pred, test_labels_LBP)
sns.heatmap(conf_mat, annot = True, cmap = 'Blues')


# ### VGG - Gram matrix

# In[17]:
```

```
#import libraries and modules
import numpy as np
import torch
import torch.nn as nn


# In[18]:


#the VGG19 architecture and weights provided are imported here
class VGG19(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            # encode 1-1
            nn.Conv2d(3, 3, kernel_size=(1, 1), stride=(1, 1)),
            nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 1-1
            # encode 2-1
            nn.Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False),

            nn.Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 2-1
            # encoder 3-1
            nn.Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False),
            nn.Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 3-1
            # encoder 4-1
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
```

```
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False),

            nn.Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 4-1
            # rest of vgg not used
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False),

            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 5-1
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            # nn.ReLU(inplace=True),
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            # nn.ReLU(inplace=True),
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), padding_mode='reflect'),
            # nn.ReLU(inplace=True)
        )

    def load_weights(self, path_to_weights):
        vgg_model = torch.load(path_to_weights)
        # Don't care about the extra weights
        self.model.load_state_dict(vgg_model, strict=False)
        for parameter in self.model.parameters():
            parameter.requires_grad = False

    def forward(self, x):
```

```
        # Input is numpy array of shape (H, W, 3)
        # Output is numpy array of shape (N_l, H_l, W_l)
        x = torch.from_numpy(x).permute(2, 0, 1).unsqueeze(0).float()
        out = self.model(x)
        out = out.squeeze(0).numpy()
        return out


# In[207]:


if __name__ == '__main__':
    # Load the model and the provided pretrained weights
    vgg = VGG19()
    vgg.load_weights('D:\Purdue\ECE661_CV\HW7\HW7-
Auxilliary\\vgg_normalized.pth')
    # Read an image into numpy array
    from skimage import io, transform
    x = io.imread('D:\Purdue\ECE661_CV\HW7\HW7-
Auxilliary\data\\training\\sunrise1.jpg')

    # Resize the input image
    x = transform.resize(x, (256, 256))
    # Obtain the output feature map
    ft = vgg(x)
    print(ft.shape)

    #reshape the feature map
    ft = ft.reshape(*ft.shape[:-2], -1)
    print(ft.shape)

    #compute gram matrix
    gram = np.dot(ft,ft.T)
    print(gram.shape)
    gram = np.where(gram > 0.00000001, gram, 1e-4)

    #apply logarithmic to gram
    gram_log = np.log(gram)

    #plot the gram matrix
    plt.imshow(gram_log, interpolation='nearest')
    plt.show()


# In[135]:
```

```
#compute gram matrices for train and test data for C = 15. Save the gram matrices
features
for img in train_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
    gram = gram[np.triu_indices(15)]

    gram_train_15.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_train_gram.npy', gram_train_15)

for img in test_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
    gram = gram[np.triu_indices(15)]

    gram_test_15.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_test_gram15.npy', gram_test_15)


# In[199]:


#train and predict the accuracy using SVM classifier for C = 15
classifier.fit(gram_train_15, train_labels_VGG.ravel())
pred_gram_15 = classifier.predict(gram_test_15)

accuracy = accuracy_score(pred_gram_15, test_labels_VGG.ravel())
```

```
print(accuracy)


# In[200]:


#plot the confusion matrix as heat map for C = 15
conf_mat_gram15 = confusion_matrix(pred_gram_15, test_labels_VGG)
sns.heatmap(conf_mat_gram15, annot = True, cmap = 'Blues')


# In[138]:


#compute gram matrices for train and test data for C = 120. Save the gram
matrices features
gram_train_120 = []
gram_test_120 = []

for img in train_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
    gram = gram[np.triu_indices(120)]

    gram_train_120.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_train_gram120.npy', gram_train_120)

for img in test_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
```

```
    gram = gram[np.triu_indices(120)]

    gram_test_120.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_test_gram120.npy', gram_test_120)


# In[140]:


#train and predict the accuracy using SVM classifier for C = 120
classifier.fit(gram_train_120, train_labels_VGG.ravel())
pred_gram_120 = classifier.predict(gram_test_120)

accuracy = accuracy_score(pred_gram_120, test_labels_VGG.ravel())
print(accuracy)


# In[141]:


#plot the confusion matrix as heat map for C = 120
conf_mat_gram120 = confusion_matrix(pred_gram_120, test_labels_VGG)
sns.heatmap(conf_mat_gram120, annot = True, cmap = 'Blues')


# In[142]:


#compute gram matrices for train and test data for C = 320. Save the gram
matrices features
gram_train_320 = []
gram_test_320 = []

for img in train_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
    gram = gram[np.triu_indices(320)]
```

```
    gram_train_320.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_train_gram320.npy', gram_train_320)

for img in test_data_VGG:

    # Obtain the output feature map
    ft = vgg(img)
    ft = ft.reshape(*ft.shape[:-2], -1)

    #compute gram matrix
    gram = np.dot(ft,ft.T)

    #get the top triangular elements
    gram = gram[np.triu_indices(320)]

    gram_test_320.append(gram)

np.save('D:\Purdue\ECE661_CV\HW7\\feature_test_gram320.npy', gram_test_320)


# In[143]:


#train and predict the accuracy using SVM classifier for C = 320
classifier.fit(gram_train_320, train_labels_VGG.ravel())
pred_gram_320 = classifier.predict(gram_test_320)

accuracy = accuracy_score(pred_gram_320, test_labels_VGG.ravel())
print(accuracy)


# In[144]:


#plot the confusion matrix as heat map for C = 320
conf_mat_gram320 = confusion_matrix(pred_gram_320, test_labels_VGG)
sns.heatmap(conf_mat_gram320, annot = True, cmap = 'Blues')
```

****