ECE 66100 HW7 Report

Zhengxin Jiang
(jiang839@purdue.edu)

November 15, 2022

1 Theory Questions

Explanation of texture measure methods

Gray Lcale Co-Occurrence Matrix: The idea of GLCM is to estimate the joint probability distribution $P[x1, x2]$, where $x1$ is a random pixel and $x2$ is at a specific vector from $x1$. The GLCM matrix has the size of $M \times M$ where M is the gray levels of the image. GLCM is a statistical method.

Local Binary Pattern: The LBP method looks to find a rotation-invariant representation of local patterns. Let R be the radius from the center pixel and P be the number of neighboring points, the point positions of a local pattern can be expressed as

$$(\Delta u, \Delta v) = (Rcos(\frac{2\pi p}{P}), Rsin(\frac{2\pi p}{P})) \quad p = 0, 1, 2, ..., P-1$$

The local pattern is then turned into binaries by comparing the values with the center pixel. The corresponding histogram bin of the pattern is determined by the minIntVal of its binary pattern. LBP is also a statistical method.

Gabor Filter Family: Gabor filters analyze images based on periodicities at different frequencies and in different directions. It's like a localized Fourier transform with localization provided by a Gaussian decay function. Gabor Filter is a structural method.

Color Representation

(a) Wrong. RGB and HSI are in different coordinates. HSI has coordinates refer to angle while RGB has coordinates only for values.

(b) Right. $L*a*b*$ is an opponent color model and also nonlinear.

(c) Right. Light coming off the object surface has a specular component and a diffuse component. The direction, color and intensity of illumination all influence the color measurement.

2 Implementation

LBP descriptor

The algorithm of getting the LBP histogram vectors is described in the theory question. For this homework, the parameters of $R = 1$ and $P = 8$ are used.

Gram descriptor

First, the gram matrix is calculated using the feature map F^l form the the layer $relu5_1$ in the vgg network:

$$G^l = F^l F^{l^T}$$

Then we take the upper triangle of G^l and randomly select $C = 1024$ elements to form the gram vector.

AdaIN descriptor

Taking the same feature map as used for gram matrix, we now do channel normalization by calculating

$$\mu_i^l = \frac{1}{M_l} \sum_{k=0}^{M_i - 1} x_{i,k}^l$$

$$\sigma_i^l = \sqrt{\frac{1}{M_l} \sum_{k=0}^{M_i - 1} (x_{i,k}^l - \mu_i^l)^2}$$

where μ_i^l is per-channel mean, σ_i^l is per-channel variance and $x_{i,k}^l$ denotes the feature value at channel i location k. The AdaIN vector is then given by

$$v_{norm} = (\mu_0, \sigma_0, \mu_1, \sigma_1, ..., \mu_{N_l}, \sigma_{N_l})$$

3 Plot of LBP Histogram

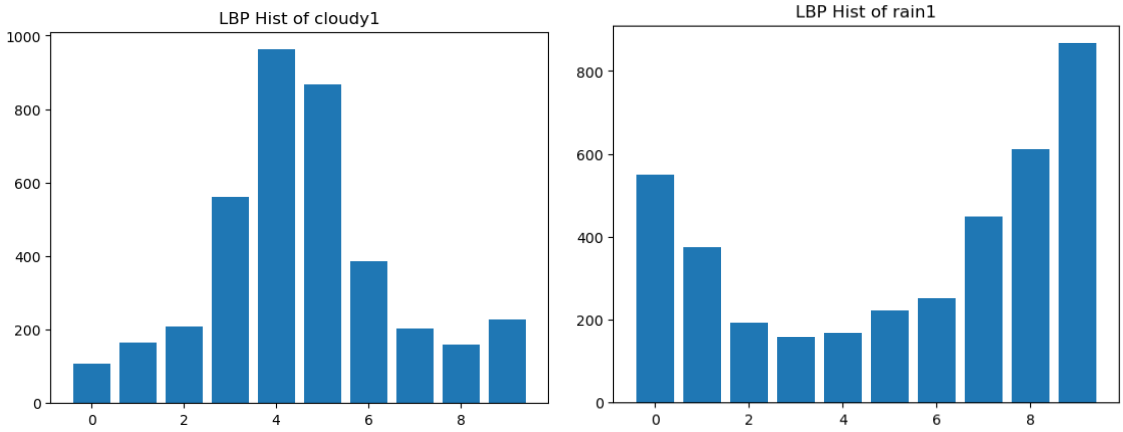

Figure 1: LBP Histogram of cloudy1 and rain1

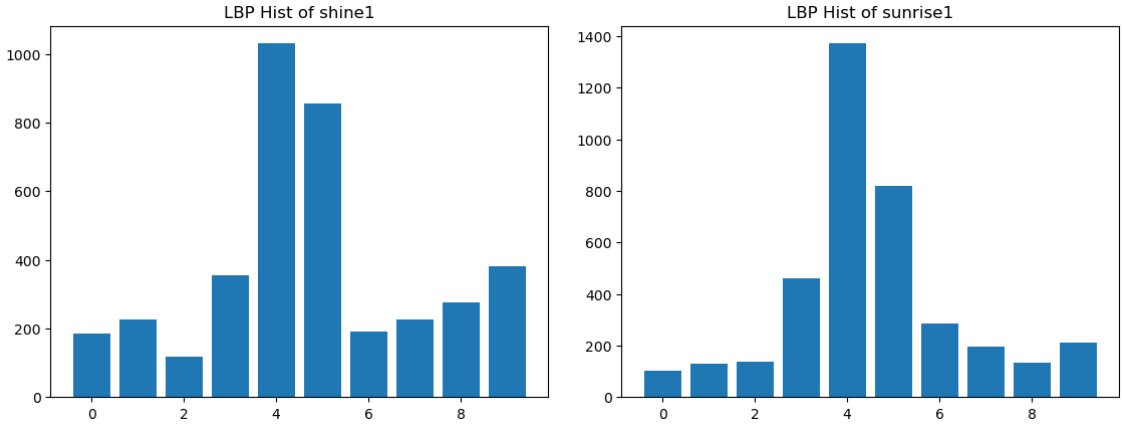

Figure 2: LBP Histogram of shine1 and sunrise1

4 Plot of Gram Matrix

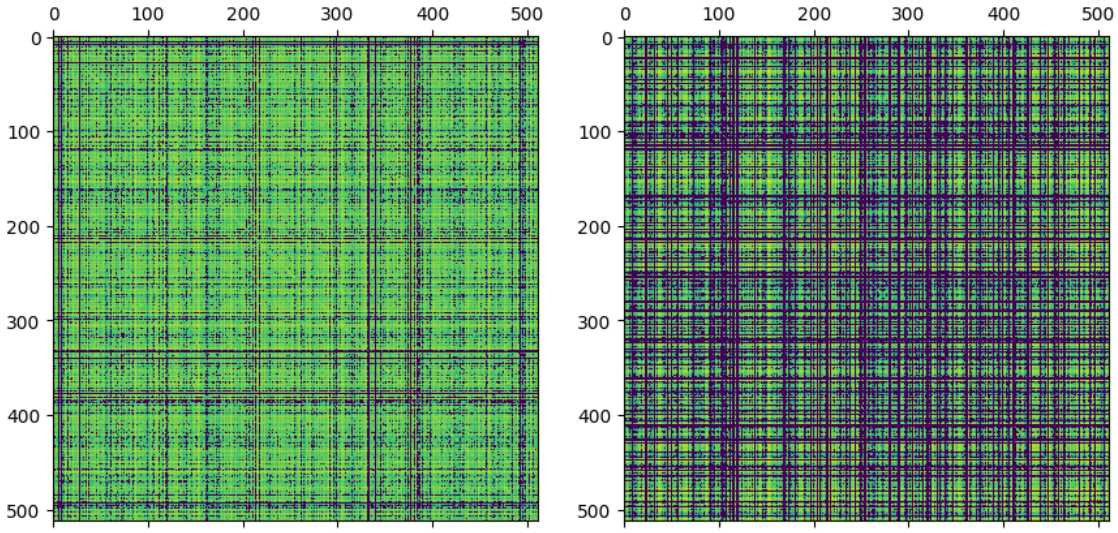

Figure 3: Gram Matrix of cloudy1 and rain1

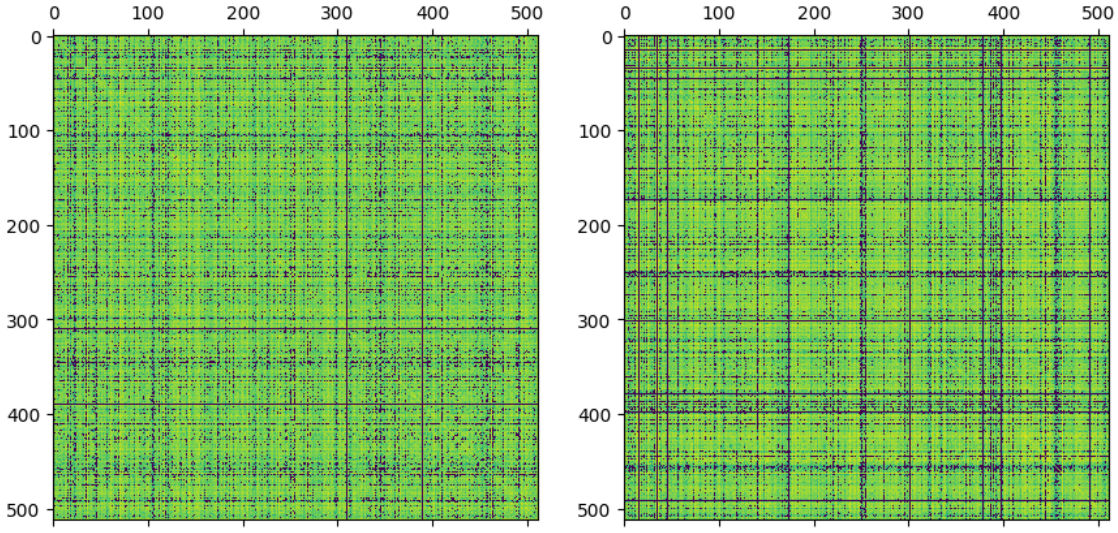

Figure 4: Gram Matrix of shine1 and sunrise1

5 Classification Results of All Descriptors

All extracted feature vectors for all descriptors are classified by SVM. In my homework implementation the Scikit-learn library is used.

LBP histogram feature vector

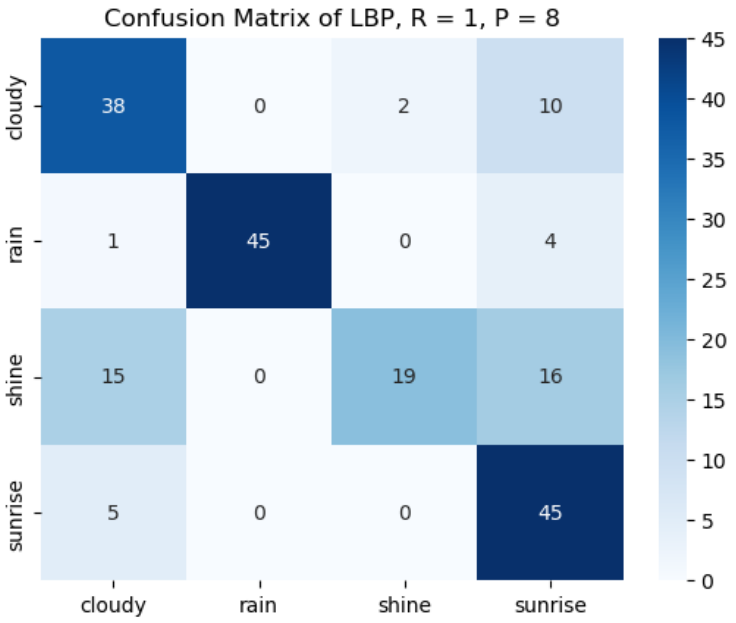

Figure 5: Confusion matrix of LBP

The test accuracy of LBP descriptor is: 0.735

Gram matrix feature vector

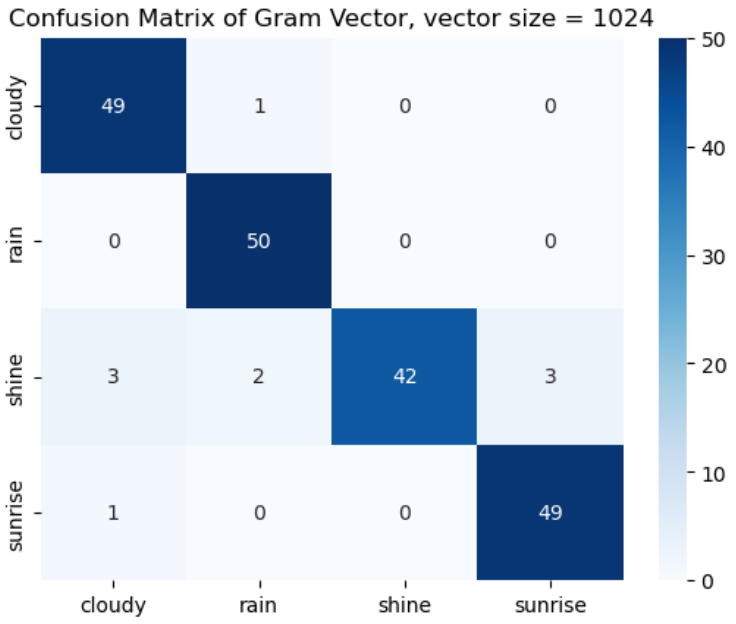

Figure 6: Confusion matrix of gram matrix

The test accuracy of gram descriptor is 0.95.

AdaIN feature vector (Extra credit)

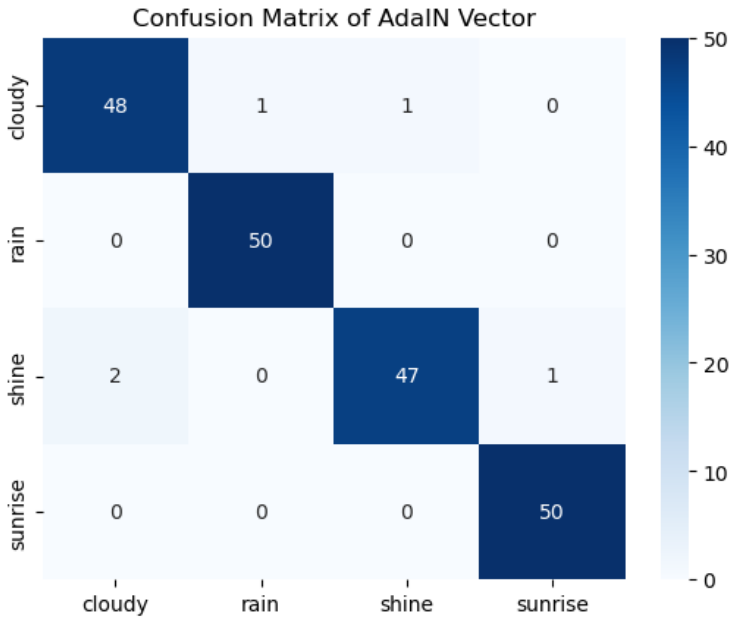

Figure 7: Confusion matrix of AdaIN

The test accuracy of AdaIN descriptor is 0.975.

6 Comments on Results

First of all, the deep network feature based descriptors have significantly higher test accuracy than the LBP descriptor. The LBP descriptor has the test accuracy of 0.735 while gram and AdaIN descriptors both have test accuracy over 0.9.

For the LBP descriptor, the accuracy of the shine class is significantly lower than other classes. The same applies to the gram descriptor. For the AdaIN descriptor, all four classes have similar accuracy.

7 Source code

```
# ECE661 HW7
# Zhengxin Jiang
# jiang839

import cv2
import numpy as np
import matplotlib . pyplot as plt
import math
from BitVector import *
from vgg import *
from skimage import io, transform
from sklearn import svm
import seaborn as sn


num_trainc = 250
num_trainr = 165
num_trainsh = 203
num_trainsr = 307
num_testc = 50
num_testr = 50
num_testsh = 50
num_testsr = 50


# load train and test images
def loadImages():

    train_imgset = []
    train_label = []
    test_imgset = []
    test_label = []

    # load images
    for i in range(1, num_trainc+1):
        img = cv2.imread('data/training/cloudy'+str(i)+'.jpg')
        train_imgset.append(img)
        train_label.append(0)

    for i in range(1, num_testc+1):
        img = cv2.imread('data/testing/cloudy'+str(i+num_trainc)+'.jpg')
        test_imgset.append(img)
        test_label.append(0)

    for i in range(1, num_trainr+1):
        img = cv2.imread('data/training/rain'+str(i)+'.jpg')
        train_imgset.append(img)
        train_label.append(1)

    for i in range(1, num_testr+1):
        img = cv2.imread('data/testing/rain'+str(i+num_trainr)+'.jpg')
        test_imgset.append(img)
```

```
        test_label.append(1)

    for i in range(1, num_trainsh+1):
        img = cv2.imread('data/training/shine'+str(i)+'.jpg')
        train_imgset.append(img)
        train_label.append(2)

    for i in range(1, num_testsh+1):
        img = cv2.imread('data/testing/shine'+str(i+num_trainsh)+'.jpg')
        test_imgset.append(img)
        test_label.append(2)

    for i in range(1, num_trainsr+1):
        img = cv2.imread('data/training/sunrise'+str(i)+'.jpg')
        train_imgset.append(img)
        train_label.append(3)

    for i in range(1, num_testsr+1):
        img = cv2.imread('data/testing/sunrise'+str(i+num_trainsr)+'.jpg')
        test_imgset.append(img)
        test_label.append(3)

    return train_imgset, train_label, test_imgset, test_label


# create a histogram using LBP
def lbpHist(image):

    img = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (64,64))

    del_l, del_k = 0.707, 0.707
    w_a = (1-del_l)*(1-del_k)
    w_d = del_l*del_k
    w_bc = (1-del_l)*del_k

    lbp_hist = np.zeros(10)

    # go through image pixels
    for i in range(1, img.shape[1]-1):
        for j in range(1, img.shape[0]-1):

            block = img[j-1:j+2, i-1:i+2].astype(int)

            # create pattern
            p = np.zeros(8)

            p[0] = block[2,1]
            p[1] = w_a*block[1,1] + w_bc*(block[2,1]+block[1,2]) + w_d*block[2,2]
            p[2] = block[1,2]
            p[3] = w_a*block[1,1] + w_bc*(block[0,1]+block[1,2]) + w_d*block[0,2]
            p[4] = block[0,1]
            p[5] = w_a*block[1,1] + w_bc*(block[0,1]+block[1,0]) + w_d*block[0,0]
            p[6] = block[1,0]
```

```
            p[7] = w_a*block[1,1] + w_bc*(block[2,1]+block[1,0]) + w_d*block[2,0]

            pattern = np.where(p>=block[1,1],1,0)

            # minintval representation
            bv = BitVector(bitlist = list(pattern))
            intvals_for_circular_shifts = [int(bv << 1) for _ in range(8)]
            minbv = BitVector(intVal = min(intvals_for_circular_shifts), size=8)
            bvruns = minbv.runs()

            # increment the corresponding bin
            if len(bvruns) > 2:
                lbp_hist[9] += 1
            elif len(bvruns) == 1 and bvruns[0][0] == '1':
                lbp_hist[8] += 1
            elif len(bvruns) == 1 and bvruns[0][0] == '0':
                lbp_hist[0] += 1
            else:
                lbp_hist[len(bvruns[1])] += 1

    return lbp_hist


# Calculate the gram matrix using the given image and model,
# return the vector of the gram matrix using the given indices
def gramVec(image, model, indices):

    x = transform.resize(image, (256, 256))

    # feature map
    ft = vgg(x)
    ft = ft.reshape(512, 256)

    gram = ft.dot(ft.T)
    gram = gram[np.triu_indices(gram.shape[0])] # upper triangle

    gramvec = gram[indices]

    return gramvec


# Calculate the AdaIN vector using the given image and model,
def adainVec(image, model):

    x = transform.resize(image, (256, 256))

    # feature map
    ft = vgg(x)
    ft = ft.reshape(512, 256)

    mu = np.mean(ft, axis=1)
    mu_expand = np.tile(mu,(256,1)).T
    sigma = np.sqrt(np.mean(np.power(ft-mu_expand, 2), axis=1))
```

```
    vector = np.zeros(1024)
    for i in range(512):
        vector[i*2] = mu[i]
        vector[i*2+1] = sigma[i]

    return vector


# train a svm classifier and return the test confusion matrix
def svmConfusionMatrix(train_vec, test_vec, train_label, test_label):

    clf = svm.SVC()
    clf.fit(train_vec, train_label)

    predict = clf.predict(test_vec)

    cm = np.zeros((4,4))

    for i in range(len(test_vec)):
        cm[test_label[i], predict[i]] += 1

    return cm


if __name__ == '__main__' :

    result_path = 'C:/Users/jzx/OneDrive␣-␣purdue.edu/ECE661/hw7/result␣images/'

    # load images
    train_imgset, train_label, test_imgset, test_label = loadImages()


    ### Task LBP ###
# train_lbpvec = []
# test_lbpvec = []

# for i in range(len(train_imgset)):
# train_lbpvec.append(lbpHist(train_imgset[i]))

# for j in range(len(test_imgset)):
# test_lbpvec.append(lbpHist(test_imgset[j]))

# np.savez_compressed('lbp', train=train_lbpvec, test=test_lbpvec)

    loaded = np.load('lbp.npz')
    train_lbpvec = loaded['train']
    test_lbpvec = loaded['test']

    cm = svmConfusionMatrix(train_lbpvec, test_lbpvec, train_label, test_label)

    plt.figure()
    plt.title("Confusion␣Matrix␣of␣LBP,␣R␣=␣1,␣P␣=␣8")
    sn.heatmap(cm, annot=True, cmap="Blues",xticklabels=['cloudy', 'rain', 'shine', '
        sunrise'], yticklabels=['cloudy', 'rain', 'shine', 'sunrise'])
```

```
    ### Task Gram ###
# train_gramvec = []
# test_gramvec = []

# vgg = VGG19()
# vgg.load_weights('vgg_normalized.pth')

# rand_indices = np.random.randint(131328, size = 1024)

# for img in train_imgset:
# train_gramvec.append(gramVec(img, vgg, rand_indices))

# for img in test_imgset:
# test_gramvec.append(gramVec(img, vgg, rand_indices))

# np.savez_compressed('gram', train=train_gramvec, test=test_gramvec)

    loaded = np.load('gram.npz')
    train_gramvec = loaded['train']
    test_gramvec = loaded['test']

    cm_gram = svmConfusionMatrix(train_gramvec, test_gramvec, train_label, test_label)

    plt.figure()
    plt.title("Confusion␣Matrix␣of␣Gram␣Vector,␣vector␣size␣=␣1024")
    sn.heatmap(cm_gram, annot=True, cmap="Blues",xticklabels=['cloudy', 'rain', 'shine',
        'sunrise'], yticklabels=['cloudy', 'rain', 'shine', 'sunrise'])


    ### Extra Credit AdaIN ###
# train_adainvec = []
# test_adainvec = []

# vgg = VGG19()
# vgg.load_weights('vgg_normalized.pth')

# for img in train_imgset:
# train_adainvec.append(adainVec(img, vgg))

# for img in test_imgset:
# test_adainvec.append(adainVec(img, vgg))

# np.savez_compressed('adain', train=train_adainvec, test=test_adainvec)

    loaded = np.load('adain.npz')
    train_adainvec = loaded['train']
    test_adainvec = loaded['test']

    cm_adain = svmConfusionMatrix(train_adainvec, test_adainvec, train_label, test_label)

    plt.figure()
    plt.title("Confusion␣Matrix␣of␣AdaIN␣Vector")
```

```
sn.heatmap(cm, annot=True, cmap="Blues",xticklabels=['cloudy', 'rain', 'shine', '
    sunrise'], yticklabels=['cloudy', 'rain', 'shine', 'sunrise'])
```