ECE661 Computer Vision HW7

Chengjun Guo
guo456@purdue.edu

2 November 2022

1 Theory question

1.1

The reading material for Lecture 16 presents three different approaches to characterizing the texture in an image: 1) using the Gray Lcale 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. (You are not expected to write more than a dozen sentences on each).

GLCM method is characterizing by estimating the joint probability $\mathrm{P}(x_1, x_2)$ where pixel with value x_2 exists using displacement vector with x_1 as a starting pixel value . GLCM will use a matrix of N by N where N is the number of possible values. This matrix will keep track of the occurrence of (x,y) at N_{xy} for any starting pixel value x and corresponding pixel value y. This matrix will be symmetric. The GLCM matrix will be normalized to represent the probability distribution. The entropy, energy and homogeneity of the matrix will provide characterization of it.

LBP histogram is another way to create rotationally and grayscale invariant characterization of texture. Firstly, we define a P that stands for the number of evenly distributed sample points around the center pixel. Then we define a R as the radius from the center to the surrounding sample points. The value at the sample points is calculated by bilinear interpolation. We record a 1 if the value at sample point is larger than the center point and a 0 if it is less than the center point. This is rotated to get minimum decimal value and this will make it invariant to rotation. Then it's encoded into P+2 levels where 0 means there is no 1, 1 to P means the number of 1 and P+1 means there are more than 2 runs. The histogram will keep track of the texture with the P+2 bins.

Unlike the first two statistical methods, Gabor filter method is structural method. A gabor filter is a highly localized fourier transform in which the localization is achieved by Gaussian decay function. Gabor filter will localize periodicity and

orientation. The convolutional operator for the continuous form a gabor filter:

$$h(x, y; u, v) = \frac{1}{\sqrt{\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} e^{-j2\pi(ux+vy)}$$

All in all, the gaussian weighting gives the localization needed, the direction of the periodicities in the fourier kernel will characterize texture in that direction.

1.2

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. (b) The color space L*a*b* is a nonlinear model of color perception. (c) Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.

(a) Wrong. HSI space is cylindrical where H is the azimuthal angle around the vertical axism S is the outward radial distance and I is the height along vertical. It is more close to how human perceive colors while RGB is more close to how hardware device for displaying images.
(b)Right. For example, if we want to convert from RGB to L*a*b*, then we need XYZ color value and then apply nonlinear functions on it to get L*a*b* values.
(c)Right. The difficulties could include the direction of illumination, the reflection properties of surface, the color of illumination, the intensity of illumination, color filtering by the hardware.

2 Logic

The code includes two parts. The first one is the function defined called LBP that is used to get the LBP histogram of each image with changeable P and R. The second part is the main function that includes getting LBP histogram and gram matrix after relu5_1 for first image in each class and getting feature map for each image then save them. Then we load the saved data for a SVM that can classify the images. For extra credit, the main function includes AdaIN method for feature extracting and save and load the feature map for SVM.

2.1 Algorithms and implementation

2.1.1 LBP histogram

For the LBP histogram, as mentioned before, we define a P that stands for the number of evenly distributed sample points around the center pixel. Then we

define a R as the radius from the center to the surrounding sample points. Take p=8 as example:

$$(\Delta u, \Delta v) = \left(R\cos\left(\frac{2\pi p}{P}\right), R\sin\left(\frac{2\pi p}{P}\right)\right) \quad p = 0, 1, 2, \ldots, 7$$

$(\Delta u, \Delta v)$ is the displacement vector from the centerpoint to the surrounding points. For each point surrounding, we calculate $(\Delta u, \Delta v)$ to get their coordinates. With those true coordinates, we calculate the value using bilinear interpolation:

$$\begin{aligned}\text{image } (x) \approx &(1-\Delta k)(1-\Delta l)A + (1-\Delta k)\Delta l B \\ &+ \Delta k(1-\Delta l)C + \Delta k \Delta l D\end{aligned}$$

where Δk and Δl means the x axis and y axis distance from left top pixel to the true coordinate.
After getting the value at each points, we get 0 if it's less than the center value and 1 if it's more than the center value.
This is rotated to get minimum decimal value and this will make it invariant to rotation. Then it's encoded into P+2 levels where 0 means there is no 1, 1 to P means the number of 1 and P+1 means there are more than 2 runs. For example: 0 0 1 1 1 1 1 1 encode as 6, 0 0 0 0 0 0 0 0 encode as 0, 0 0 0 0 0 1 0 1 encode as 9.
With these encoding, looping through all the points will gives a histogram with P+2 bins.

2.1.2 Gram matrix

Given a output from relu5_1, it is a $512 \times 16 \times 16$ matrix. We first flatten each 16×16 matrix as a row. This will become a 512×256 matrix A. Gram matrix is calculated by $A \cdot A^T$. The weaker the correlation ship between the row i and row j, the value at (i,j) is more likely to become 0.

2.1.3 AdaIN

For the Adaptive normalization, it is calculating the mean and square root of variance for each lane in matrix A mentioned in Gram matrix section and put them in a list. The new list V_{norm} would be with length of $2N_l$ where N_l means the number of channels.

2.1.4 Image classification pipeline

For each image in train and test dataset, we record the label in its name and the feature descriptor as the data. For image in train dataset, I have train_x, train_y lists as the input for training SVM. With the trained SVM, test_x will

generate a predicted y label. With the predicted y label and the test_y, we can generate a confusion matrix and calculate the accuracy of it.

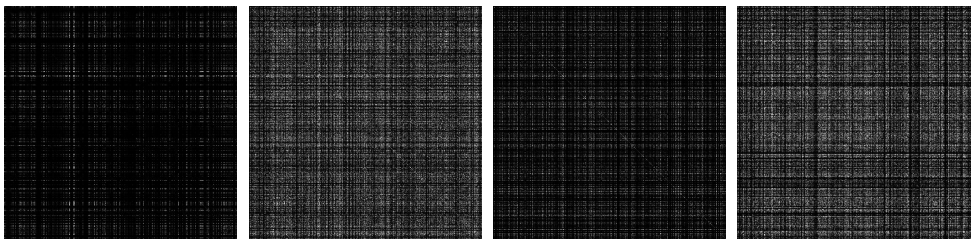
3 Data and images:

Accuracy:

method	accuracy
gram matrix	0.945
lbp histogram	0.73
adain	0.975

3.1 gram matrix

The sample gram matrix for each class:



(a) rain1 (b) shine1 (c) cloudy1 (d) sunrise1

Figure 1: gram matrix for each class

Confusion matrix of gram matrix:

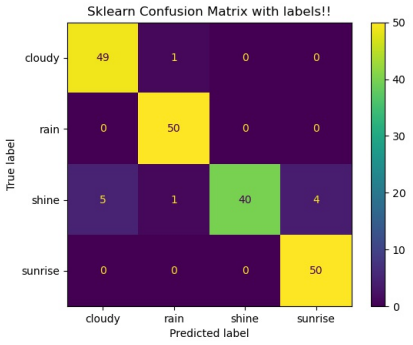
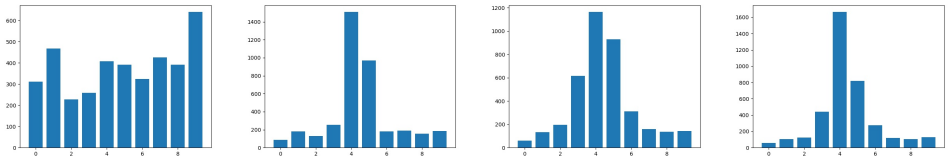


Figure 2: Confusion matrix of gram matrix

3.2 lbp histogram

The sample lbp histogram for each class:



(a) rain1 (b) shine1 (c) cloudy1 (d) sunrise1

Figure 3: lbp histogram for each class

Confusion matrix of lbp histogram:

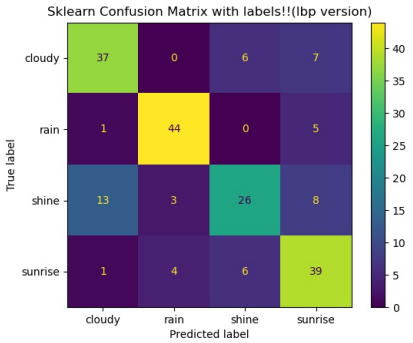


Figure 4: Confusion matrix of lbp histogram

3.3 AdaIN

Confusion matrix of AdaIN:

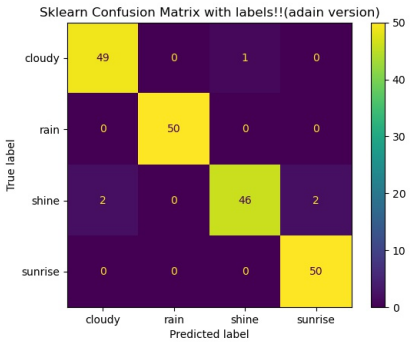


Figure 5: Confusion matrix of AdaIN

4 Comment on gram matrix visualization

For all of the gram matrices, shine and sunrise have more correlated channels based on the gram matrix. Sunrise1 and shine1 looks similar to each other is reasonable because the only difference in the image is the sunrise looks red while shine looks white. Rain1 gram matrix is the most distinct one among all four classes and it is reasonable to be most accurate class among all four classes.

5 Comment on performance of different descriptors

Among all the methods, the AdaIN has the highest accuracy. The lbp hist has the lowest accuracy. Both AdaIN and gram matrix methods have higher accuracy than 90%. For all of the three methods, they got lowest accuracy on shine images. Also, all of them have highest accuracy on raining. I think the result is reasonable, rain has the most evident texture with the rain drops. For shine images, they are recognized as cloudy and sunrise. Take shine 167 as example, it would be reasonable to call it cloudy even as human eyes. Compared to rain, shine is more difficult to distinguish is reasonable.

6 Code

```
import cv2
import numpy as np
import BitVector
```

```
import math
import os
import glob
import matplotlib.pyplot as plt
import time
from skimage import data, io, filters, transform
from vgg import VGG19
from sklearn import svm,metrics


def LBP(image,P=8,R=1):
    '''
    ref: https://engineering.purdue.edu/kak/Tutorials/
       TextureAndColor.pdf
    This is calculating the LBP histogram of the image
    '''
    lbp_hist = {t:0 for t in range(P+2)}
    for i in range(R,image.shape[0]-R):
        for j in range(R,image.shape[1]-R):
            pattern = []
            for p in range(P):
                #find the delta u and delta v
                u, v = R * math.cos(2 * math.pi * p / P),
                    R * math.sin(2 * math.pi * p / P)
                real_y, real_x = i + u, j + v
                y_tl, x_tl = int(real_y), int(real_x)
                delta_y, delta_x = real_y - y_tl, real_x
                   - x_tl
                # when x becomes 63, delta is 0, it will
                    go out of boundary
                if delta_x == 0 and delta_y == 0:
                    image_val_at_p = image[y_tl][x_tl]
                elif delta_x == 0:
                    image_val_at_p = (1 - delta_y) * (1 -
                        delta_x) * image[y_tl][x_tl] +
                       delta_y * (1 - delta_x) * \
                                   image[y_tl + 1][x_tl
                                       ]
                elif delta_y == 0:
                    image_val_at_p = (1 - delta_y) * (1 -
                        delta_x) * image[y_tl][x_tl] + (1
                        - delta_y) * delta_x * \
                                   image[y_tl][x_tl +
                                       1]
                else:
```

```
                    image_val_at_p = (1 - delta_y) * (1 -
                        delta_x) * image[y_tl][x_tl] + (1
                        - delta_y) * delta_x * \
                                    image[y_tl][x_tl +
                                        1] + delta_y *
                                        delta_x * image[
                                        y_tl + 1][x_tl +
                                        1] + \
                                    delta_y * (1 -
                                        delta_x) * image[
                                        y_tl + 1][x_tl]
                if image_val_at_p >= image[i][j]:
                    pattern.append(1)
                else:
                  pattern.append(0)
            #reform the pattern
            bv = BitVector.BitVector(bitlist=pattern)
            intvals_for_circular_shifts = [int(bv << 1)
                for _ in range(P)]
            minbv = BitVector.BitVector(intVal= min(
                intvals_for_circular_shifts), size=P)
            bvruns = minbv.runs()
            if len(bvruns) > 2:
                lbp_hist[P + 1] += 1
            elif len(bvruns) == 1 and bvruns[0][0] ==
                "1":
                lbp_hist[P] += 1
            elif len(bvruns) == 1 and bvruns[0][0] ==
                "0":
                lbp_hist[0] += 1
            else:
                lbp_hist[len(bvruns[1])] += 1
    return lbp_hist


if __name__ == '__main__':
    classes = ['cloudy','rain','shine','sunrise']
    vgg = VGG19()
    vgg.load_weights('vgg_normalized.pth')

    # 4.1 & 4.2
    # for cls in classes:
    #     image = io.imread("data/training/"+cls+"1.jpg")
    #     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #     downsize = cv2.resize(gray,(64,64),
        interpolation = cv2.INTER_AREA)
```

```
#     lbp = LBP(downsize)
#     plt.figure()
#     plt.bar(lbp.keys(), lbp.values())
#     plt.savefig(cls+"1_lbp_hist.jpg")
#
#     x = io.imread("data/training/"+cls+"1.jpg")
#     x = transform.resize(x, (256, 256))
#     ft = vgg(x)
#     F_l = ft.reshape(ft.shape[0], ft.shape[1] * ft.
    shape[2])
#     G = np.dot(F_l, F_l.T)
#     io.imsave(cls+"1_gram_matrix.jpg",G.astype(np.
    uint8))

# preprocess train data and test data by gram matrix
x_train = []
y_train_true = []
x_test = []
y_test_true = []
# feature_extraction = np.random.choice(512 ** 2,
    1024)
dict = {'cloudy':0,'rain':1,'shine':2,'sunrise':3}
# for cls in classes:
#     for img in glob.glob("data/training/"+cls+"*.
    jpg"):
#         print(img)
#         start = time.time()
#         x = io.imread(img)
#         if len(x.shape) != 3:
#             continue
#         if x.shape[2] != 3:
#             continue
#         x = transform.resize(x, (256, 256))
#         ft = vgg(x)
#         F_l = ft.reshape(ft.shape[0],ft.shape[1]*ft
    .shape[2])
#         G = np.dot(F_l, F_l.T)
#         elapsed = (time.time() - start)
#         print("time elapsed:" + str(elapsed))
#         y_train_true.append(dict[cls])
#         x_train.append(G.flatten()[
    feature_extraction])
# for cls in classes:
#     for img in glob.glob("data/testing/"+cls+"*.jpg
    "):
#         print(img)
```

```
#         start = time.time()
#         x = io.imread(img)
#         if len(x.shape) != 3:
#             continue
#         if x.shape[2] != 3:
#             continue
#         x = transform.resize(x, (256, 256))
#         ft = vgg(x)
#         F_l = ft.reshape(ft.shape[0],ft.shape[1]*ft
    .shape[2])
#         G = np.dot(F_l, F_l.T)
#         elapsed = (time.time() - start)
#         print("time elapsed:" + str(elapsed))
#         y_test_true.append(dict[cls])
#         x_test.append(G.flatten()[
    feature_extraction])
#
# np.savez_compressed('featuremap', train_x=x_train,
    train_y=y_train_true,test_x=x_test,test_y=
    y_test_true,feature_extraction=feature_extraction)

# preprocess train data and test data by lbp
# for cls in classes:
#     for img in glob.glob("data/training/"+cls+"*.
    jpg"):
#         print(img)
#         start = time.time()
#         x = io.imread(img)
#         if len(x.shape) != 3:
#             continue
#         if x.shape[2] != 3:
#             continue
#         gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
#         downsize = cv2.resize(gray,(64,64),
    interpolation = cv2.INTER_AREA)
#         lbp = LBP(downsize)
#         elapsed = (time.time() - start)
#         print("time elapsed:" + str(elapsed))
#         y_train_true.append(dict[cls])
#         x_train.append(list(lbp.values()))
# for cls in classes:
#     for img in glob.glob("data/testing/"+cls+"*.jpg
    "):
#         print(img)
#         start = time.time()
#         x = io.imread(img)
```

```
#         if len(x.shape) != 3:
#             continue
#         if x.shape[2] != 3:
#             continue
#         gray = cv2.cvtColor(x, cv2.COLOR_BGR2GRAY)
#         downsize = cv2.resize(gray,(64,64),
    interpolation = cv2.INTER_AREA)
#         lbp = LBP(downsize)
#         elapsed = (time.time() - start)
#         print("time elapsed:" + str(elapsed))
#         y_test_true.append(dict[cls])
#         x_test.append(list(lbp.values()))
#
# np.savez_compressed('featuremap_lbp', train_x=
    x_train, train_y=y_train_true,test_x=x_test,test_y
    =y_test_true)


## preprocess train data and test data by adain
# for cls in classes:
#     for img in glob.glob("data/training/"+cls+"*.
    jpg"):
#         print(img)
#         start = time.time()
#         x = io.imread(img)
#         if len(x.shape) != 3:
#             continue
#         if x.shape[2] != 3:
#             continue
#         x = transform.resize(x, (256, 256))
#         ft = vgg(x)
#         F_l = ft.reshape(ft.shape[0], ft.shape[1] *
     ft.shape[2])
#         Vnorm = []
#         for i in range(F_l.shape[0]):
#             Vnorm.append(np.mean(F_l[i]))
#             Vnorm.append(np.sqrt(np.var(F_l[i])))
#         x_train.append(Vnorm)
#         y_train_true.append(dict[cls])
# for cls in classes:
#     for img in glob.glob("data/testing/"+cls+"*.jpg
    "):
#         print(img)
#         start = time.time()
#         x = io.imread(img)
#         if len(x.shape) != 3:
```

```
#             continue
#         if x.shape[2] != 3:
#             continue
#         x = transform.resize(x, (256, 256))
#         ft = vgg(x)
#         F_l = ft.reshape(ft.shape[0],ft.shape[1]*ft
    .shape[2])
#         Vnorm = []
#         for i in range(F_l.shape[0]):
#             Vnorm.append(np.mean(F_l[i]))
#             Vnorm.append(np.sqrt(np.var(F_l[i])))
#         y_test_true.append(dict[cls])
#         x_test.append(Vnorm)
# np.savez_compressed('featuremap_adain', train_x=
    x_train, train_y=y_train_true,test_x=x_test,test_y
    =y_test_true)

# loaded = np.load('featuremap.npz')
# loaded = np.load('featuremap_lbp.npz')
loaded = np.load('featuremap_adain.npz')
train_x = loaded['train_x']
train_y = loaded['train_y']
test_x = loaded['test_x']
test_y = loaded['test_y']


clf = svm.SVC()
clf.fit(train_x, train_y)
y_test_pred = clf.predict(test_x)
print("Accuracy:", metrics.accuracy_score(test_y,
    y_test_pred))
cm = metrics.confusion_matrix(test_y, y_test_pred)
cmd_obj = metrics.ConfusionMatrixDisplay(cm,
    display_labels=classes)
cmd_obj.plot()
cmd_obj.ax_.set(title='Sklearn Confusion Matrix with
    labels!!(adain version)')
plt.savefig('confusion_matrix_adain.jpg')
```