ECE 66100 Computer Vision **Name:** *Nikita Ravi (ravi30)*
Homework 7
10/02/2022

1 Theoretical Questions

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

   (a) **Gray Scale Co-Occurence Matrix (GLCM)**: The core idea of GLCM is to estimate the joint probability distribution $P[x_1, x_2]$ for the grayscale values in an image where x_1 is the grayscale value at a randomly selected pixel from the image and x_2 is the grayscale value at another pixel that is at a distance d from the first pixel. After estimating $P[x_1, x_2]$, the texture can be characterized by the ***shape*** of the joint distribution. As the image is being raster scanned, the (m, n)-th element of the GLCM matrix records the number of times we have seen the grayscale value at the current pixel being m while the grayscale value at the *d*-displaced pixel being n. An interesting feature of the GLCM matrix, aside from the fact that it is a symmetric $M \times M$ matrix for an image with M gray levels is that if you sum the diagonal elements of a normalized GLCM matrix, the result yielded would be the probability that two pixels in the image that are separated by a displacement d will have identical grayscale values. GLCM is an example of texture characterizations based on their second-order statistical properties.

   (b) **Local Binary Pattern (LBM)**: The core idea of LBP is the notion that the local binary pattern to characterize the grayscale variations around a pixel through runs of 0s and 1s. Textures that have a single run of 0s and a single run of 1s carry the most discriminative information between different kind of textures. In other words, The LBP representation is constructed by comparing each pixel in the image with pixels in it's surrounding neighborhood. The LBP method is designed in such a way that the patterns are invariant to linear changes in image contrast, and stay invariant to in-plane rotations. LBP is an example of texture characterizations based on their second-order statistical properties.

   (c) **Gabor Filter Family**: Many image textures are composed of repetitively occurring micro-patterns thus making the Gabor filters ideal for these kind of images. This is because Gabor filters are highly localized fourier transforms where localization is achieved by applying a Gaussian decay to the pixels in an image with periodicity at different frequencies and in different directions. The technique based on Gabor filters is an example of the structural approach to texture characterizations.

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.
Answer: This is wrong because the transformation that exists to convert a color point from one space to another is not a linear relationship.

(b) The color space L*a*b* is a nonlinear model of color perception.
Answer: This is right because for example, to go from RGB to L*a*b*, you must locate the color value in the absolute color space XYZ and then apply nonlinear functions on the result values to get the L*, a*, and b* values.

(c) Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.
Answer: This is right because when illumination is not purely diffused, the color of the light reflecting off an object surface depends on the direction of the incident illumination and the true color of a surface is normally associated with just the diffuse light coming off the surface instead of the specular component.

2 Programming Section

2.1 Theoretical Background

2.1.1 Local Binary Pattern

Local Binary Pattern or LBP constructs a local representation of the texture of the input image. This representation is constructed by comparing each pixel in the image with pixels in it's surrounding neighborhood. The LBP computation must be invariant to linear changes in image contrast and in-plane rotations [1].

The neighborhood surrounding a specific pixel in the image is considered to be a unit circle and is calculated by

$$(\Delta k, \Delta l) = (R\cos(\frac{2\pi p}{P}), R\sin(\frac{2\pi p}{P})) \tag{1}$$

Where R is the radius of the unit circle, P is the number of neighbors a pixel might have, and p is thr p-th pixel in the collection of P neighboring pixels, where $p = 0$ is directly below the pixel under observation and increments in a counter-clockwise direction [1]. For this homework assignment, R was chosen to be equal to one and P was chosen to be equal to eight.

The grayscale values at the neighborhood points must be computed with a bilinear interpolation formula. This is achieved by assigning corners A,B,C, and D as the centers of four adjoining pixels and via interpolation we estimate the gray level of the pixel being analyzed inside the rectangle governed by the four corners as shown below

$$image(x + \Delta k, y + \Delta l) \approx (1 - \Delta k)(1 - \Delta l)A + (1 - \Delta k)\Delta l B + \Delta k(1 - \Delta l)C + \Delta k \Delta l D \tag{2}$$

Once the grayscale values are estimated at each p pixel on the unit circle, we compare the interpolated value at the point to the central pixel. If the interpolated value is greater than or equal to the central pixel then we set the point to one. Otherwise, we set it to zero. This thresholding will ultimately yield a binary vector [1].

Once we have obtained the binary vector, we need to figure out a way to make it invariant to all possible in-plane rotations. This is achieved by circularly rotating the binary vector until it acquires the smallest integer value. This is equivalent to saying until the largest number of zeroes occupy the most significant positions [1].

After computing the minimum binary vector, we need to encode the pattern by a single integer such that we can create an image-based characterization of the texture. The first step that needs to be achieved before encoding the pattern is to get the number of runs in the bitvector pattern. A run is defined as a group of bits with the same value [2]. For instance, a bitvector with the pattern 111001 has three runs: $111, 00, 1$ [2]. Using the number of runs, the encoding is given by [1]

$$E(\text{bitvector}) = \begin{cases} P+1 & \text{runs} > 2 \\ 0 & \text{runs = 1 and bv only has 0s} \\ P & \text{runs = 1 and bv only has 1s} \\ \text{Number of 1s in 2nd run} & \text{runs} = 2 \end{cases} \tag{3}$$

We create a feature vector with $P + 2$ bins containing the probability of each encoded value which can be visualized as a histogram as shown in figure 4. These histograms are compared with each other to classify the images.

The code written to extract the local binary pattern feature vectors was inspired by Professor Kak's implementation in his lecture notes [1]

2.1.2 VGG Network

VGG19 is a popular image classification deep learning network that was trained with over a million ImageNet images. As shown in figure 1, the network consists of a total of nineteen layers, where the first sixteen are convolutional layers and the last three are fully connected layers. The three fully connected layers are there to reshape the forward propagating information so that a judgement can be made on what the classification of the input is at the final layer.

2.1.3 Gram Matrices

The output from the VGG network is a feature map where each layer contains information on different styles (i.e. edges, lines, dots, curves, etc.) [3]. Each layer of the feature map is reshaped to one dimension and appended one top of each another forming a matrix called A. The Gram matrix G is then computed as

$$G = A^T A \tag{4}$$

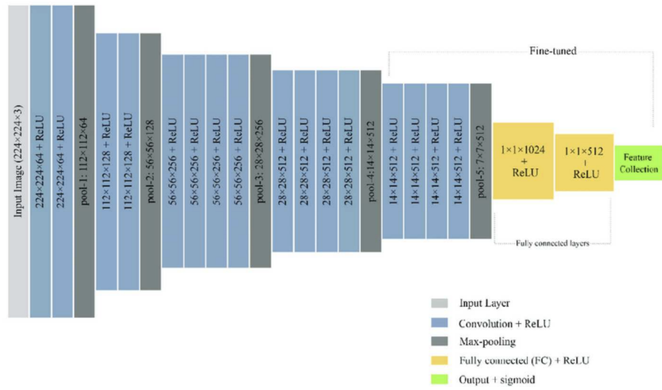


Figure 1: Architecture of the VGG Network [1]

The Gram matrix is therefore a layer-based characterization of the textures in the neural network. Since the Gram matrix was calculated by multiplying a matrix with its transpose, the result will be a symmetric matrix and also provides information on how the column vectors of A are correlated with one another [1]. Since the Gram matrix is symmetric, we only look at 1024 random samples of the upper triangle portion of the matrix to form our feature vector. The gram matrix is visualized in figure 5

2.1.4 Support Vector Machines

Support Vector Machine or SVM is a type of supervised machine learning algorithm. The objective of SVM is to find a hyperplane in an N-dimensional space that separates data points into its specific classifications [4].

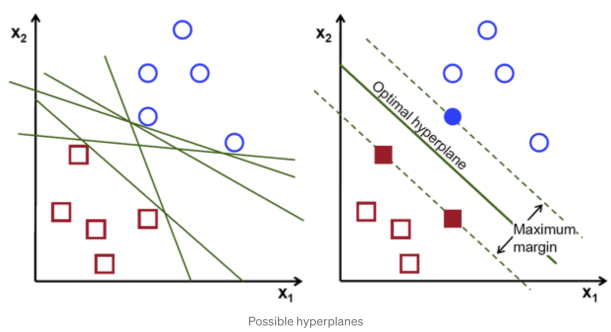


Figure 2: Support Vector Machine [4]

To choose the best hyperplane to distinguish between the classes, the objective is to find a hyperplane that has the maximum distance between data points of both classes [4] as shown in figure 2.

The two different kernels used for this homework assignment are linear for the gram matrices and RBF for the local binary patterns. The RBF kernel function for two points X_1 and X_2 computes their similarity and is computed as follows [5]:

$$K(X_1, X_2) = \exp(-\frac{\|X_1 - X_2\|^2}{2\sigma^2}) \tag{5}$$

The linear kernel is used if the data used can be modelled by a parametric machine learning model, which means the data must be linear.

2.1.5 Confusion Matrix

A confusion matrix provides information on the performance capability of the supervised machine learning algorithm for classification problems. It is a table consisting details of true positives, true negatives, false positives, and false negatives [6].

- True Positive: The predicted label is both positive and true
- True Negative: The predicted label is both negative and true
- False Positive: The predicted label is positive and false
- False Negative: The predicted label is negative and false [6]

2.2 Observations

As we can see from both the local binary pattern histograms and the gram matrices, the images part of the rain class do not follow the same trend as the other three classes. For instance, the LBP distribution of the cloudy, shine, and sunrise images are very similar to one another unlike the distribution of the histogram obtained for the image from the rain class. With respect to the gram matrices, the plot for images from the cloudy, shine, and sunrise classes are a bit darker than the plot for the image from the rain class. These differences are probably due to the fact that the texture for rain is very different from the other classes whereas there is some light visible in the cloudy, shine, and sunrise images.

Furthermore, the support vector machine model has an overall better accuracy and precision for the gram feature vector method than it did for the local binary pattern feature vector method as shown by the confusion matrices in figure 6

2.3 Source Code

```
# Name: Nikita Ravi
# Class: ECE 66100
# Homework #7
# Deadline: 11/02/2022

# Import Modules
import cv2
import math
import numpy as np
import os
import re
```

```
import BitVector as bv
from pprint import pprint
from vgg import VGG19
from sklearn import svm
from sklearn import preprocessing
from sklearn.metrics import confusionmatrix
from sklearn.metrics import plotconfusionmatrix, classificationreport
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import seaborn as sn


def getimages(path):
    images = –”cloudy”: [], ”rain”: [], ”shine”: [], ”sunrise”: []
    names = –”cloudy”: [], ”rain”: [], ”shine”: [], ”sunrise”: []

    pattern = re.compile(r”([A-Za-z]+)([0-9]+)”)
    for idx, imagename in enumerate(sorted(os.listdir(path)[::-1])):
        if(imagename != ’.DSStore’ and imagename != ”rain141.jpg” and
    imagename != ”shine131.jpg”):
            group = pattern.findall(imagename)[0][0]
            names[group].append(imagename)

            img = cv2.imread(os.path.join(path, imagename))
            images[group].append(img)

    return images, names

def displayimage(image):
    cv2.imshow(”window”, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    quit()

def getneighboringpixels(R=1, P=8):
    neighbors = np.zeros((P, 2))
    for p in range(P):
        neighbors[p][0] = R * math.sin(2 * math.pi * p / P)
        neighbors[p][1] = R * math.cos(2 * math.pi * p / P)

    return neighbors

def getimagepixelvalueatp(image, kbase, lbase, deltak, deltal):
    # This function was inspired by Professor Kak’s Local Binary Pattern Code
    from the Lecture Notes
    if(deltak ¡ 0.001 and deltal ¡ 0.001):
        imagepixelvalatp = float(image[kbase][lbase])

    elif(deltal ¡ 0.001):
        imagepixelvalatp = (1 - deltak) * image[kbase][lbase] + deltak
     * image[kbase + 1][lbase]

    elif(deltak ¡ 0.001):
        imagepixelvalatp = (1 - deltal) * image[kbase][lbase] + deltak
```

```
     * image[kbase][lbase + 1]

    else:
        imagepixelvalatp = (1 - deltak) * (1 - deltal) * image[kbase][lbase] + “
                    (1 - deltak) * deltal * image[kbase][lbase + 1] + “
                    deltak * deltal * image[kbase + 1][lbase + 1] + “
                    deltak * (1 - deltal) * image[kbase + 1][lbase]

    return imagepixelvalatp

def getencoding(bvruns, lbphist):
    # This function was inspired by Professor Kak’s Local Binary Pattern Code from the Lecture Notes
    if(len(bvruns) ¿ 2):
        lbphist[P + 1] += 1

    elif(len(bvruns) == 1 and bvruns[0][0] == ’1’):
        lbphist[P] += 1

    elif(len(bvruns) == 1 and bvruns[0][0] == ’0’):
        lbphist[0] += 1

    else:
        lbphist[len(bvruns[1])] += 1

    return lbphist

def obtainminimumbitvec(pattern):
    # Calculate the minimum binary vector for the pattern obtained to get maximal amount of information for discriminating texture features in the image
    # This function was inspired by Professor Kak’s Local Binary Pattern Code from the Lecture Notes
    bitvec = bv.BitVector(bitlist = pattern)
    intervalsforcircularshifts = [int(bitvec ¡¡ 1) for   in range(P)]
    minbitvec = bv.BitVector(intVal = min(intervalsforcircularshifts), size=P)

    return minbitvec

def generatefeaturevectorlbp(image, R=1, P=8):
    # This function was inspired by Professor Kak’s Local Binary Pattern Code from the Lecture Notes
    image = cv2.cvtColor(image, cv2.COLORBGR2GRAY)
    height, width = image.shape
    totalnumpixels = height * width

    neighbors = getneighboringpixels(R, P) # Get all the pixels around the pixel at the origin at a specific radius R
    lbphist = -t:0 for t in range(P + 2)  # LBP Histogram

    rowmax, colmax = width - R, height - R
    for i in range(R, rowmax):
```

```
        for j in range(R, colmax):
            pattern = []
            for p in range(P):
                dell, delk = neighbors[p][0], neighbors[p][1]
                delk = 0 if abs(delk) ¡ 0.001 else delk
                dell = 0 if abs(dell) ¡ 0.001 else dell

                k, l = i + delk, j + dell
                kbase, lbase = int(k), int(l) # Corner coordinates for A, B, C, D
                deltak, deltal = k - kbase, l - lbase

                imagepixelvalatp = getimagepixelvalueatp(image, kbase, lbase, deltak, deltal)

                if(imagepixelvalatp ¿= image[i][j]):
                    pattern.append(1)
                else:
                    pattern.append(0)

            minbitvec = obtainminimumbitvec(pattern)
            bvruns = minbitvec.runs() # number of groups of runs

            lbphist = getencoding(bvruns, lbphist)

    probabilityhist = –key: value / totalnumpixels for key, value in lbphist.items()
    return probabilityhist

def generatefeaturevectorgram(image):
    featuremapmatrix = []

    # Initialize the vgg model
    vgg = VGG19()
    vgg.loadweights(r”hw07/HW7-Auxilliary/vggnormalized.pth”)

    featuremaps = vgg(image) # The feature maps contains layers each containing different pieces of information on style (edges, lines, dots, curves, etc.)
    for idx, stylemapmatrix in enumerate(featuremaps):
        featuremapmatrix.append(stylemapmatrix.flatten())

    featuremapmatrix = np.array(featuremapmatrix)
    grammatrix = np.matmul(featuremapmatrix, featuremapmatrix.T) # 512x512 matrix

    # Randomly select 1024 samples using the upper triangle
    uppertrianglegramindices = np.triuindices(grammatrix.shape[0])
    uppertrianglegram = grammatrix[uppertrianglegramindices]

    mididx = len(uppertrianglegram) // 2
    featurevector = uppertrianglegram[mididx:(mididx + 1024)]

    return grammatrix, featurevector
```

```python
def plothistogram(featurevector, extraction, imagename):
    plt.figure()
    colors = plt.cm.getcmap('tab20c')
    randomcoloridx = np.random.rand()
    plt.bar(range(len(featurevector)), featurevector, width=0.8, color=colors(randomcoloridx), edgecolor = "black")
    plt.title(extraction + "Histogram for " + imagename)
    plt.savefig("hw07/histogramplots/" + extraction + " " + imagename)

def plotgrammatrix(grammatrix, imagename):
    grammatrix += 0.001
    mapping, lognorm = plt.cm.gray, colors.LogNorm()
    gramplot = mapping(lognorm(grammatrix))
    plt.imsave(fname=r"hw07/grammatrixplots/" + imagename, arr=gramplot, format="jpg")

def generatefeaturematrix(imagedict, imagenames, extraction, path, R=1, P=8, train=True):
    featurematrix, groupmatrix = [], []
    for key, images in imagedict.items():
        for idx, image in enumerate(images):
            imagename = imagenames[key][idx]
            if(extraction == "lbp"):
                imageresized = cv2.resize(image, dsize=(64, 64))
                featurevector = [val for key, val in generatefeaturevectorlbp(imageresized, R, P).items()]

                if(idx == 1 and train):
                    plothistogram(featurevector, extraction, imagename)

            elif(extraction == "gram"):
                imageresized = cv2.resize(image, dsize=(256, 256))
                grammatrix, featurevector = generatefeaturevectorgram(imageresized)

                if(idx == 1 and train):
                    plotgrammatrix(grammatrix, imagename)

            featurematrix.append(featurevector)
            groupmatrix.append(key)

    np.savezcompressed(path, featurematrix=featurematrix, groups=groupmatrix)

def loadsavedmatrix(path):
    loaded = np.load(path)
    featurematrix, groupmatrix = np.matrix(loaded["featurematrix"], dtype=np.float32), np.array(loaded["groups"])

    return featurematrix, groupmatrix

def nominalencoding(labels):
    le = preprocessing.LabelEncoder()
```

```
    le.fit(labels)
    return le

def trainsupportvectormachine(featurematrix, groupencoded, kernel=”rbf”):
    clf = svm.SVC(kernel=kernel)
    clf.fit(featurematrix, groupencoded)
    return clf

def predictgroupname(clf, featurematrix):
    yHat = clf.predict(featurematrix)
    return yHat

def plotconfusionmatrix(cnfmatrix, classes, title=”LBP”):
    ax = sn.heatmap(cnfmatrix, annot=True, cmap=’Blues’)

    ax.settitle(”Confusion Matrix for ” + title +”“n“n”);
    ax.setxlabel(’“nActual Values’)
    ax.setylabel(’Predicted Values ’)

    ax.xaxis.setticklabels(classes)
    ax.yaxis.setticklabels(classes)

    plt.show()
    plt.savefig(”hw07/confusionmatrix/” + title + ”.jpg”)

def createconfusionmatrix(ytest, yHat, le, title):
    ytestdecoded, yHatdecoded = list(le.inversetransform(ytest)), list(le.inversetransform(yHat))
    classes = list(set(ytestdecoded + yHatdecoded))

    cnfmatrix = confusionmatrix(ytestdecoded, yHatdecoded, labels=classes).T
    plotconfusionmatrix(cnfmatrix, classes, title=title)

def createclassificationreport(ytest, yHat, le):
    ytestdecoded, yHatdecoded = list(le.inversetransform(ytest)), list(le.inversetransform(yHat))
    classes = list(set(ytestdecoded + yHatdecoded))

    report = classificationreport(ytestdecoded, yHatdecoded, labels=classes)
    print(report)


if  name  == ” main ”:
    trainpath = r”hw07/HW7-Auxilliary/data/training”
    testpath = r”hw07/HW7-Auxilliary/data/testing”

    trainimages, trainnames = getimages(trainpath)
    testimages, testnames = getimages(testpath)

    mode = ”Classify”

    if(mode == ”Extract”):
```

```
        ### Task 1 - Local Binary Pattern
        P = 8 # Number of neighboring pixels
        R = 1 # Radius of circle neighborhood

        generatefeaturematrix(trainimages, trainnames, "lbp", r"hw07/lbpmatrix/trainedfeaturematrix", R=1, P=8)
        generatefeaturematrix(testimages, testnames, "lbp", r"hw07/lbpmatrix/testingfeaturematrix", R=1, P=8, train=False)

        ### Task 2 - Gram Matrix Generation
        generatefeaturematrix(trainimages, trainnames, "gram", r"hw07/grammatrix/trainedfeaturematrix")
        generatefeaturematrix(testimages, testnames, "gram", r"hw07/grammatrix/testingfeaturematrix", train=False)

    elif(mode == "Classify"):
        ### Task 3 - Building an Image Classifier Pipeline
        trainlbpfeaturematrix, trainlbpgroupmatrix = loadsavedmatrix(r"hw07/lbpmatrix/trainedfeaturematrix.npz")
        testlbpfeaturematrix, testlbpgroupmatrix = loadsavedmatrix(r"hw07/lbpmatrix/testingfeaturematrix.npz")
        traingramfeaturematrix, traingramgroupmatrix = loadsavedmatrix(r"hw07/grammatrix/trainedfeaturematrix.npz")
        testgramfeaturematrix, testgramgroupmatrix = loadsavedmatrix(r"hw07/grammatrix/testingfeaturematrix.npz")

        #=======================================================TRAIN==========================================================================

        labelencoderlbp = nominalencoding(trainlbpgroupmatrix)
        labelencodergram = nominalencoding(traingramgroupmatrix)

        trainlbpgroupencoded = labelencoderlbp.transform(trainlbpgroupmatrix)
        traingramgroupencoded = labelencodergram.transform(traingramgroupmatrix)

        svmlbp = trainsupportvectormachine(trainlbpfeaturematrix, trainlbpgroupencoded)
        svmgram = trainsupportvectormachine(traingramfeaturematrix, traingramgroupencoded, kernel="linear")

        #=======================================================PREDICT=======================================================================
        yHatlbp = predictgroupname(svmlbp, testlbpfeaturematrix)
        yHatgram = predictgroupname(svmgram, testgramfeaturematrix)

        createconfusionmatrix(labelencoderlbp.transform(testlbpgroupmatrix), np.array(yHatlbp), labelencoderlbp, title="LBP")
        print("Classification for LBP")
        createclassificationreport(labelencoderlbp.transform(testlbpgroupmatrix), np.array(yHatlbp), labelencoderlbp)

```

```
    createconfusionmatrix(labelencodergram.transform(
testgramgroupmatrix), np.array(yHatgram), labelencodergram, title="
Gram")
    print("Classification for Gram")
    createclassificationreport(labelencodergram.transform(
testgramgroupmatrix), np.array(yHatgram), labelencodergram)
```

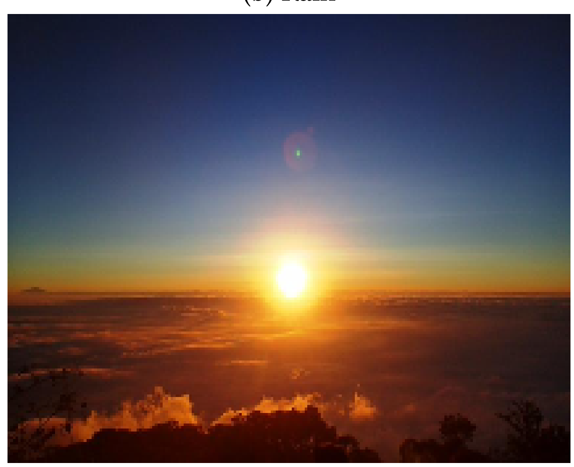
Listing 1: The Source Code

2.4 The Inputs

(a) Cloudy

(b) Rain

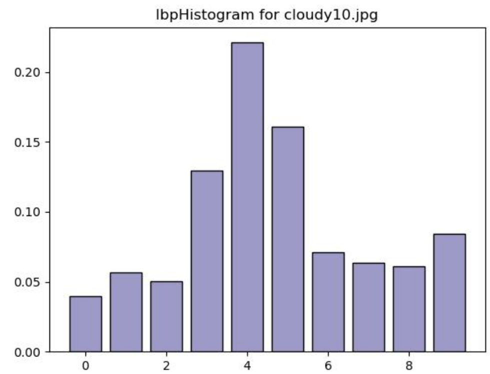
(c) Shine



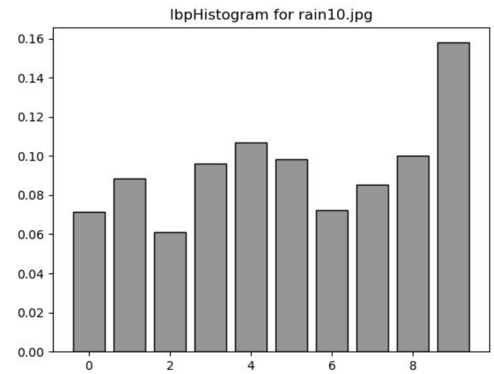
(d) Sunrise

Figure 3: The Input Classes

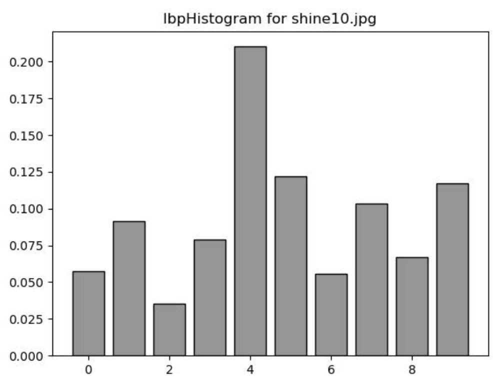
2.5 The Outputs



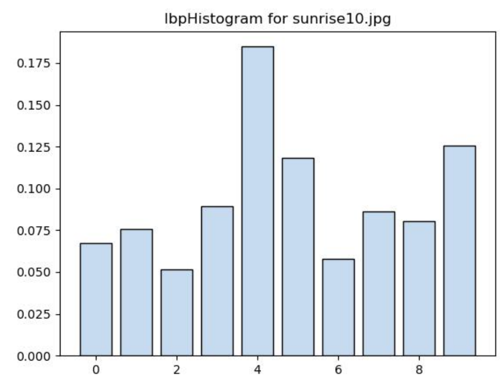
(a) LBP Histogram for Cloudy 10 Image



(b) LBP Histogram for Rain 10 Image

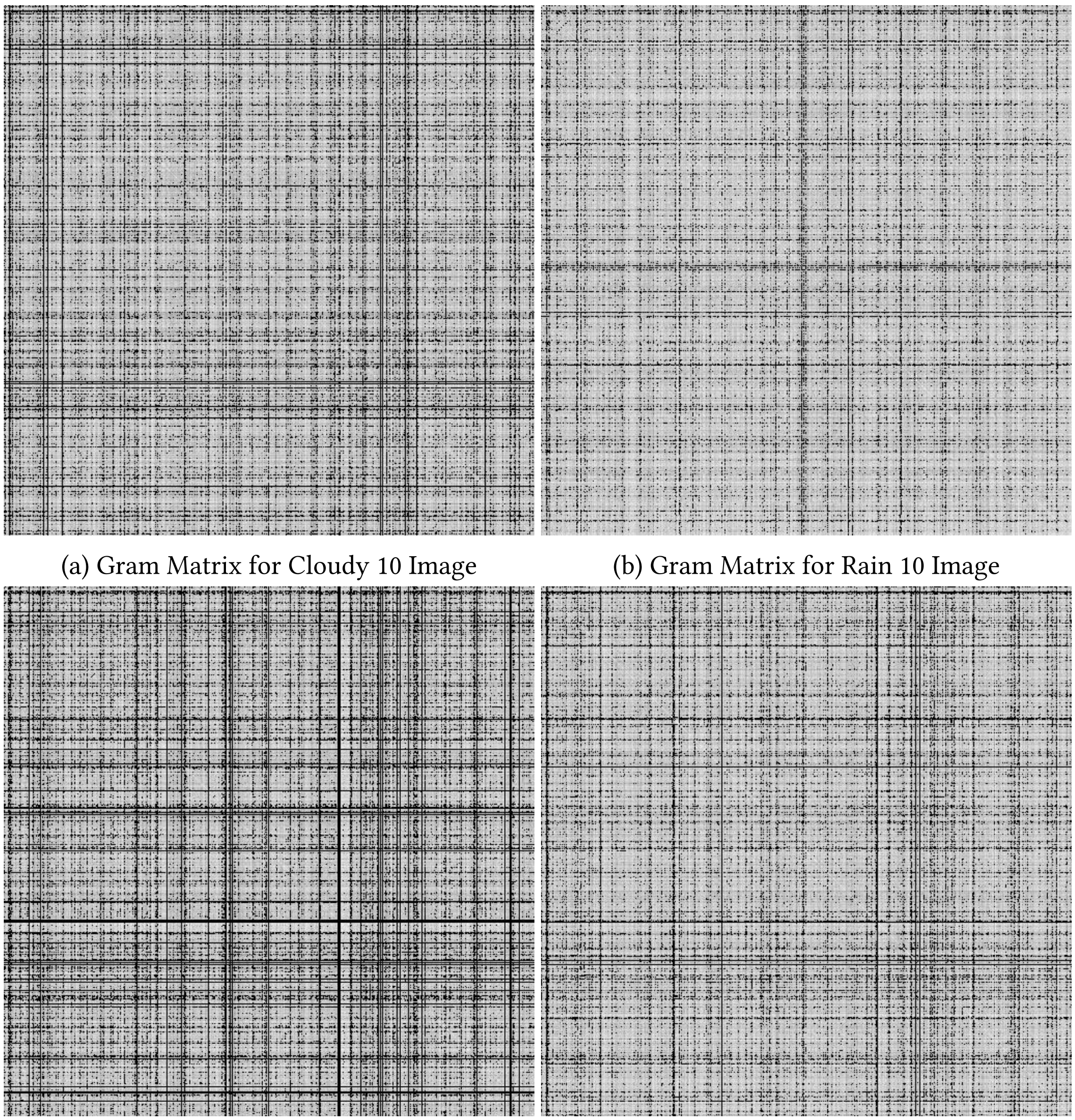


(c) LBP Histogram for Shine 10 Image



(d) LBP Histogram for Sunrise 10 Image

Figure 4: LBP Histograms for the 10th image of each class



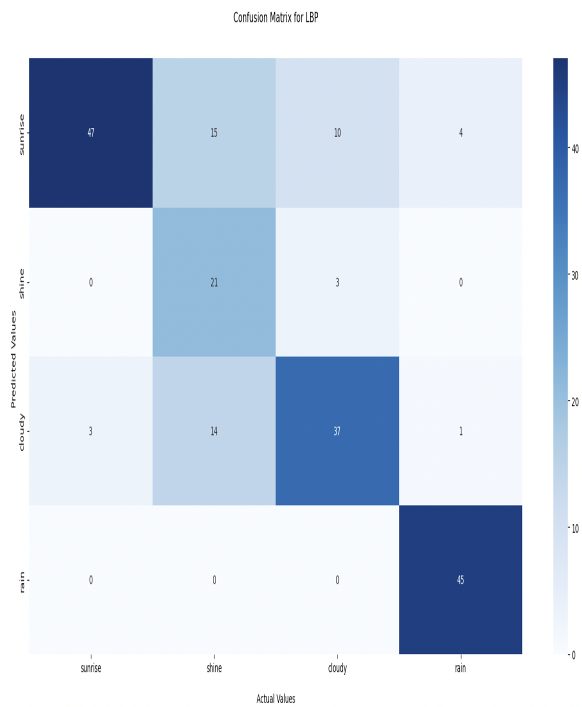
(a) Gram Matrix for Cloudy 10 Image

(b) Gram Matrix for Rain 10 Image

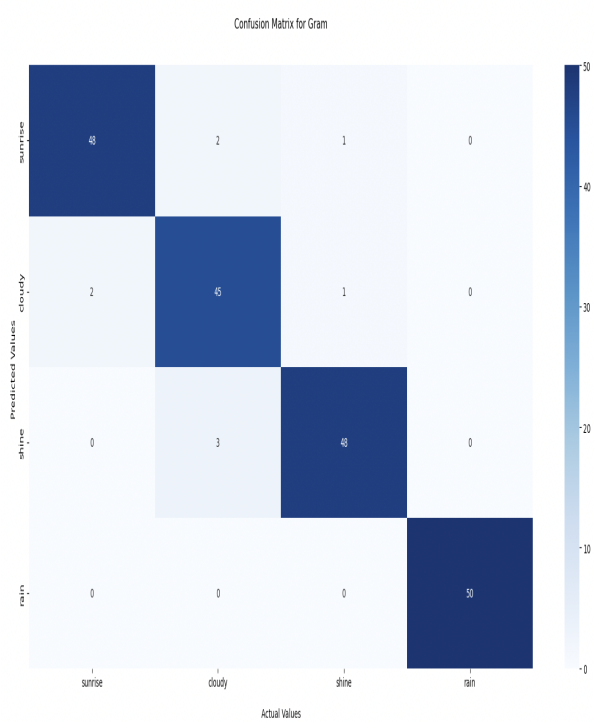
(c) Gram Matrix for Shine 10 Image

(d) Gram Matrix for Sunrise 10 Image

Figure 5: Gram Matrix for the 10th image of each class



(a) Confusion Matrix for Local Binary Pattern Method



(b) Confusion Matrix for Gram Matrix Method

Figure 6: Confusion Matrix

Classification for LBP

	precision	recall	f1-score	support
shine	0.88	0.42	0.57	50
rain	1.00	0.90	0.95	50
cloudy	0.67	0.74	0.70	50
sunrise	0.62	0.94	0.75	50
accuracy			0.75	200
macro avg	0.79	0.75	0.74	200
weighted avg	0.79	0.75	0.74	200

(a) Classification Report for Local Binary Pattern Method

Classification for Gram

	precision	recall	f1-score	support
shine	0.94	0.96	0.95	50
rain	1.00	1.00	1.00	50
cloudy	0.94	0.90	0.92	50
sunrise	0.94	0.96	0.95	50
accuracy			0.95	200
macro avg	0.95	0.95	0.95	200
weighted avg	0.95	0.95	0.95	200

(b) Classification Report for Gram Matrix Method

Figure 7: Classification Report

References

[1] A. Kak, "Lecture 16: Texture and Color", https://engineering.purdue.edu/kak/Tutorial
2022.

[2] A. Kak, "BitVector", https://engineering.purdue.edu/kak/dist/BitVector-3.4.8.
2018.

[3] T. C. Reader, "Gram or Gramian Matrix – Explained with Example in Python", https://www.theclickreader.com/gram-or-gramian-matrix-explained-in-p

[4] R. Gandhi, "Support Vector Machine — Introduction to Machine Learning Algorithms", https://towardsdatascience.com/support-vector-machine-introduction-t
2018.

[5] S. Sreenivasa, "Radial Basis Function (RBF) Kernel: The Go-To Kernel", https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-
2020.

[6] S. Narkhede, "Understanding Confusion Matrix", https://towardsdatascience.com/unders
2018.