Theory Question

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).

   The three approaches mentioned above fall into two categories: Statistical and Structural. GLCM and LBP are statistical approaches. They compute the distributions of specific image properties and use it to estimate a texture in an image. Gabor Filter Family is a structural approach. This approach computes the periodicities in an image and characterize a texture with the relative spectral energy at different periodicities. All the approaches are invariant to rotation and contrast

   GLCM: This method calculates the joint probability $P(x_1, x_2)$ formed by the grayscale values in the image. The joint probability is calculated by finding the number of pairs of pixels that are a distance d apart, with one pixel at grayscale value $x1$ and the other at grayscale value $x2$. Once normalized, the shape of the joint probability $P(x_1, x_2)$ is used to estimate texture in an image.

   LBP: This approach calculates a local binary pattern to characterize the grayscale variation around a center pixel. It computes the binary pattern by thresholding the grayscale values of pixels in the neighborhood with respect to the grayscale value at the center pixel. If the grayscale value of the neighborhood pixel is equal or greater than the value at the central pixel, that pixel grayscale value is set to 1. Otherwise, it is set to 0. This binary pattern consists of "runs" of 0s and 1s. Based on the pattern of "runs", each pixel is given a encoding value, and based on the pattern of these encodings, the texture of an image is estimated.

   Gabor Filter Family: Gabor Filter Family is a structural approach. Gabor filters calculate the periodicities at different frequencies and in different directions. This is done by convoluting an image with a Gabor filter. The filter consists of highly localized Fourier transform which achieves localization by applying a Gaussian decay function to the pixels. After several filters are applied to an image, the combined outputs are used to estimate the texture.

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**: The transformation between RGB and HSI is nonlinear as points are mapped to angles.

(b)**Right**: L*a*b* color spaces are based on the Opponent Color Modeling of our color vision which is nonlinear.

(c)**Right**: There exist a number of spectrally pure colors corresponding to different wavelengths that cannot be synthesized by mixing pure R, G, and B components.

Programming Task

Implement an image classification framework that classifies images based on the LBP-based and the CNN-based texture descriptors.

1. Background

Brief Summary of the process:

- The process is divided into two parts: 1. Extract feature vectors from images and 2. Training and testing of SVM Classifier to classify images.
- Extract feature vectors from images:
    1. Iterate through training and testing data to load each image.
    2. For each image, reduce the size to (256,256).
    3. For LBP method, convert the image to grayscale and reduce the size to (64,64).
    4. For both LBP and Gram Matrix method, compute the feature vector and append it to their respective feature matrices. Each row of the feature matrix contains the feature vector of an image. Further details of computing the feature vectors for each method is given below. Similarly, the label or class of the image is appended to a "label" array.
    5. Save the LBP feature matrix and the Gram Matrix feature matrix, along with their respective label arrays to a compressed file. These matrices and arrays can be loaded later to train the SVM classifier and save time.
- Train and test SVM Classifier:
    1. Load the saved feature matrices and their corresponding label arrays.
    2. Create a SVM classifier and set the hyperparameters.
    3. Train the classifier with the training feature matrices and their corresponding label arrays.
    4. Using the test feature matrices, predict the labels by using the SVM classifier.

5. Compare the predicted labels with the actual labels and create a confusion matrix for each method.

Extracting Feature Descriptor Vectors:

1. **Local Binary Pattern:** LBP method uses statistical properties of a grayscale image to measure its texture. The feature vectors are histograms that are invariant to in-plane rotations and changes in contrast.

   To compute the feature vector of an image, a pattern of grayscale variations is computed in a circular neighborhood around each pixel in the image. The neighborhood around the target pixel is formed by finding the neighboring points. The radius R of the neighborhood and the number of neighbors P are used to calculate exact positions of the neighboring points around the target pixel.

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

   The grayscale values at the neighboring points are computed with bilinear interpolation since not all neighboring points will have coordinates that coincide exactly with existing pixel coordinates in the image.

   Once the grayscale values of all the neighboring points have been calculated, a threshold is applied to each of them such that if the neighboring point has a grayscale value greater than or equal to the grayscale value of the target pixel, the point is set to 1, otherwise it is set to 0. This forms a binary pattern of 1s and 0s. To make this invariant to rotations, the binary pattern is circularly rotated until it acquires the smallest integer value with the largest number of 0s occupying the most significant positions. This pattern characterizes the local texture at the target pixel. The BitVector module by Prof. Kak was used for this process.

   Similarly, for each pixel in the image a local binary pattern is computed that characterizes the local texture around that pixel. Each of this binary pattern is then encoded by a single integer to be used to create a image based texture characterization from the pixel based texture characterization. This is done by forming a histogram of all the encodings.

   To form the image based texture characterization, the creators of LBP suggested that only the uniform patterns be used to form the histograms. Uniform patterns are patterns that consist of a single run of 0's followed by a single run of 1's. Therefore the following encoding is used:

   - If the binary pattern has two runs: a run of 0's followed by a run of 1's, the encoding for that pattern is set to the number of 1's in the second run.
   - If the binary pattern consists of all 0's, the encoding is set to 0.
   - If the binary pattern consists of all 1's, the encoding is set to integer value P.
   - If the binary pattern consists of more than two runs, the encoding is set to the integer value P+1.

The python code given in Texture And Color Tutorial by Prof. Kak was used as inspiration for the code for this assignment.

The histogram formed from the encodings represents the feature vector for the image. The feature vector of each image is appended to the feature matrix which is then used to train the SVM classifier.

2. **Gram Matrix**: For this assignment, a pre-trained VGG-19 based network is given as the encoder (vgg.py). The output features from the layer relu5-1 are used to form the Gram matrix descriptors.

   The output of the relu5-1 layer of the VGG-19 network is a (512,16,16) feature map. To compute the Gram matrix, each row of the feature map, which is a 2D (16,16) matrix, is flattened. This process reshapes the feature map into (512,216) matrix $\mathbf{F}$.

   The Gram matrix is then computed by finding the inner product of $\mathbf{F}$ with its transpose.

   $$\mathbf{G} = \mathbf{F} \cdot \mathbf{F}^T$$

   This gives a symmetric $\mathbf{G}$ which is (512,512), which can be computationally expensive to be used as a feature descriptor to train a classifier. Since the Gram Matrix captures the inter-correlations between all the individual feature channels, a smaller subset of this correlation can be used train a classifier. To form the feature vector, first the upper triangle of the Gram Matrix is taken into consideration since it is symmetric. Then, a subset of 1024 element is formed by taking the elements from the middle of the upper triangle Gram matrix. The feature vector for each image is therefore 1024 elements long and is then appended to the feature matrix which is then used to train the SVM Classifier.

Implementation Notes:

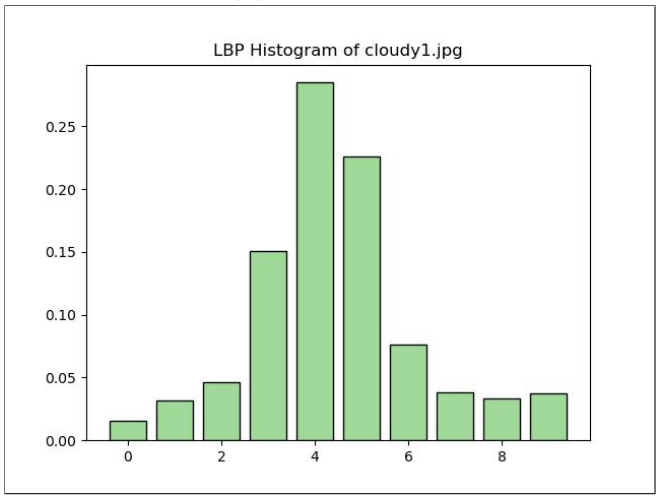
1. The Gram Matrix was scaled and normalized using the log scale, and then mapped on a grayscale color map with the Matplotlib modules. This allowed the Gram Matrix to be displayed as an image with each pixel representing a level of grayscale.

2. For the SVM classifier, CV2's SVM classifier is used to classify the image textures using the feature descriptors. The CV2 SVM classifier offers multiple kernels to choose for the classifier. The kernel that produced the best results was used.

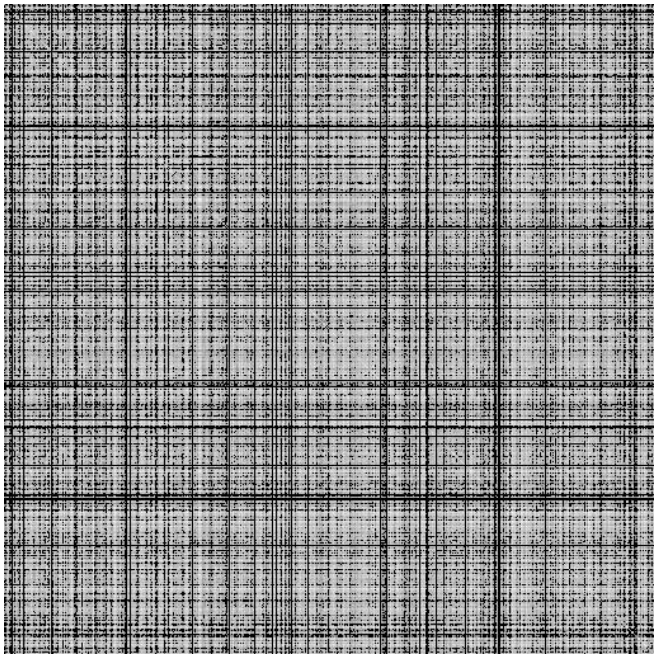
3. To display the Confusion Matrix, the Seaborn library is used. The matrix values are transformed to a heat map with the highest value closer to the darker color.

2. Results and Outputs

(a) Cloudy1.jpg



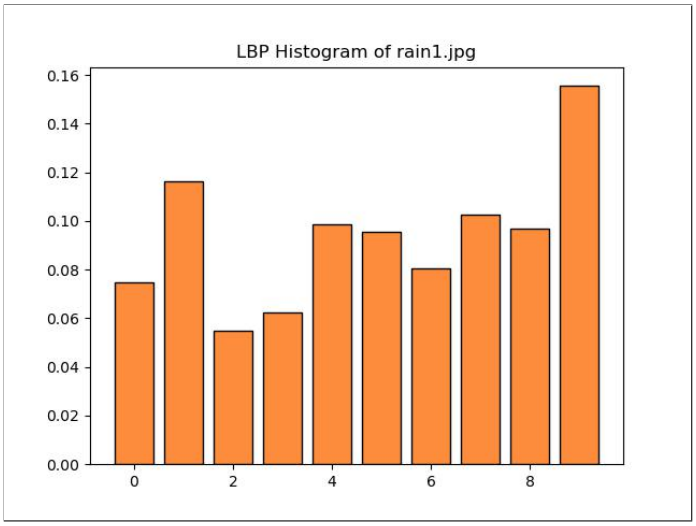
(b) LBP Histogram



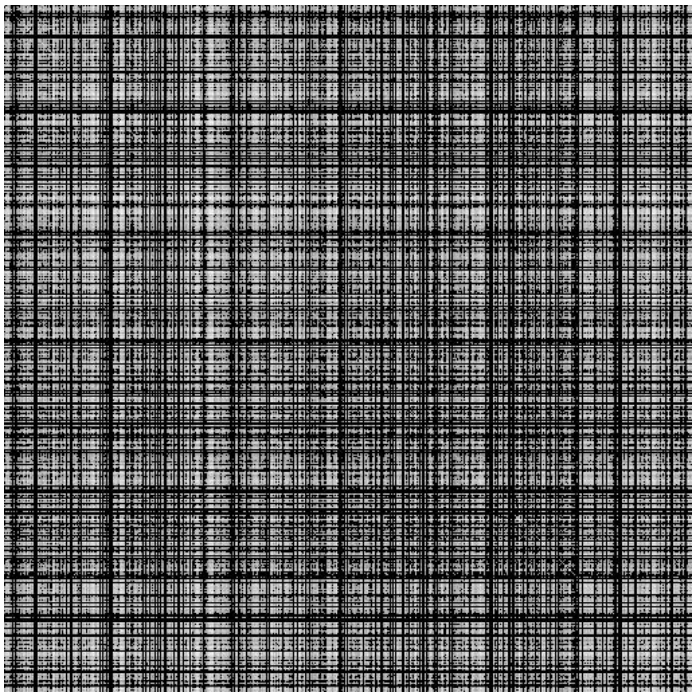
(c) Gram Matrix

Figure 1

(a) rain1.jpg



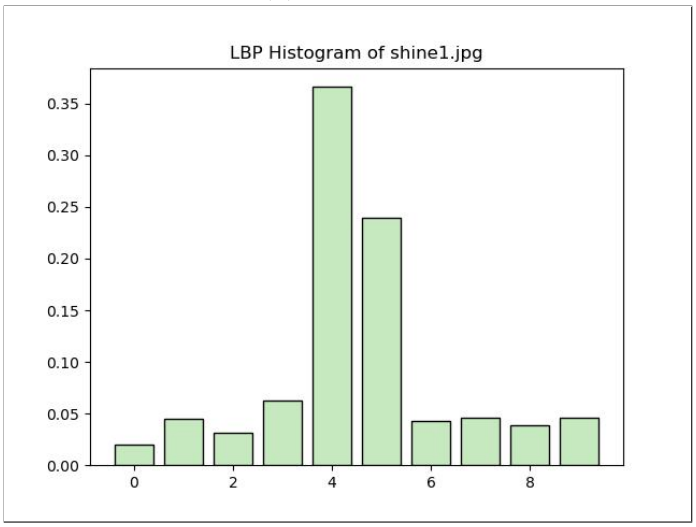
(b) LBP Histogram



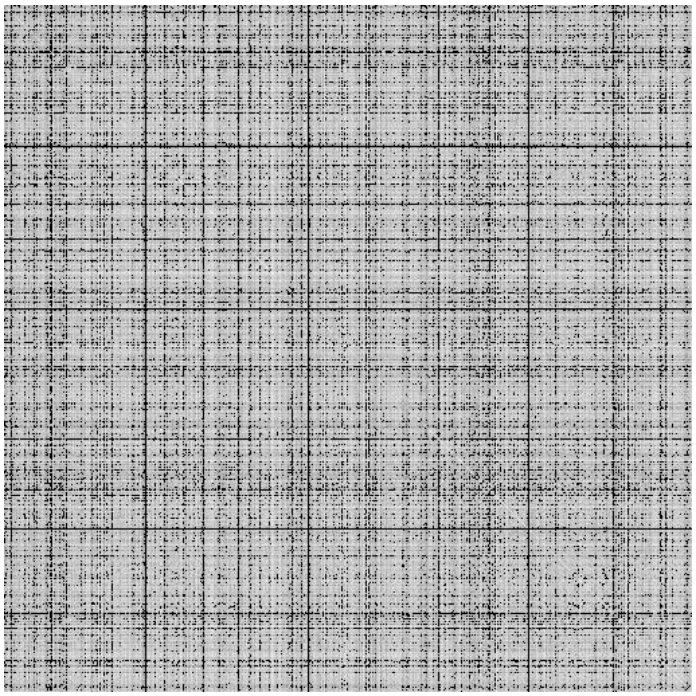
(c) Gram Matrix

Figure 2

(a) shine1.jpg



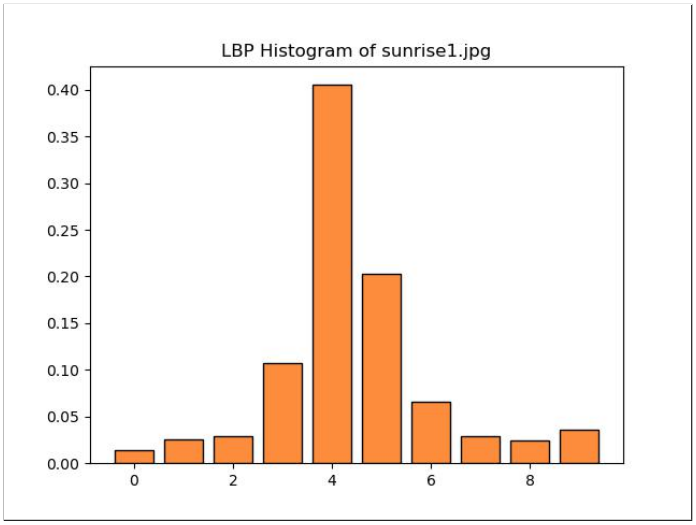
(b) LBP Histogram



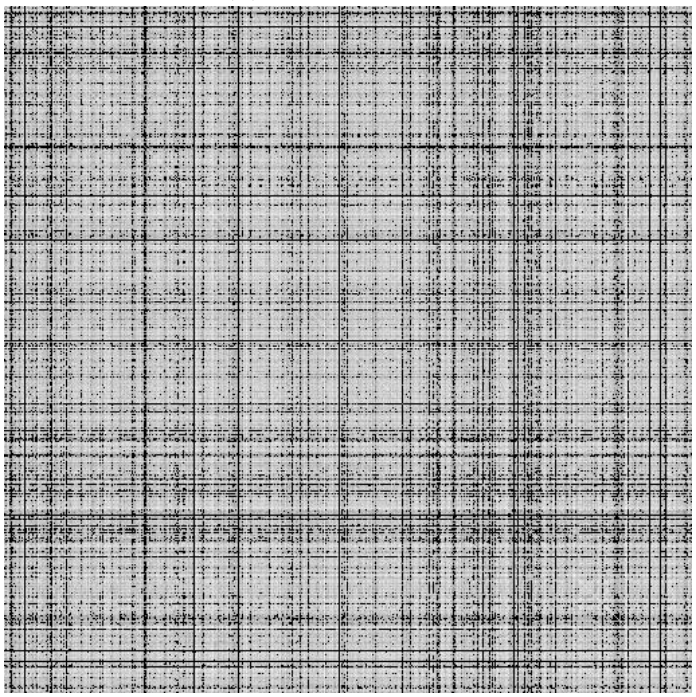
(c) Gram Matrix

Figure 3

(a) sunrise1.jpg

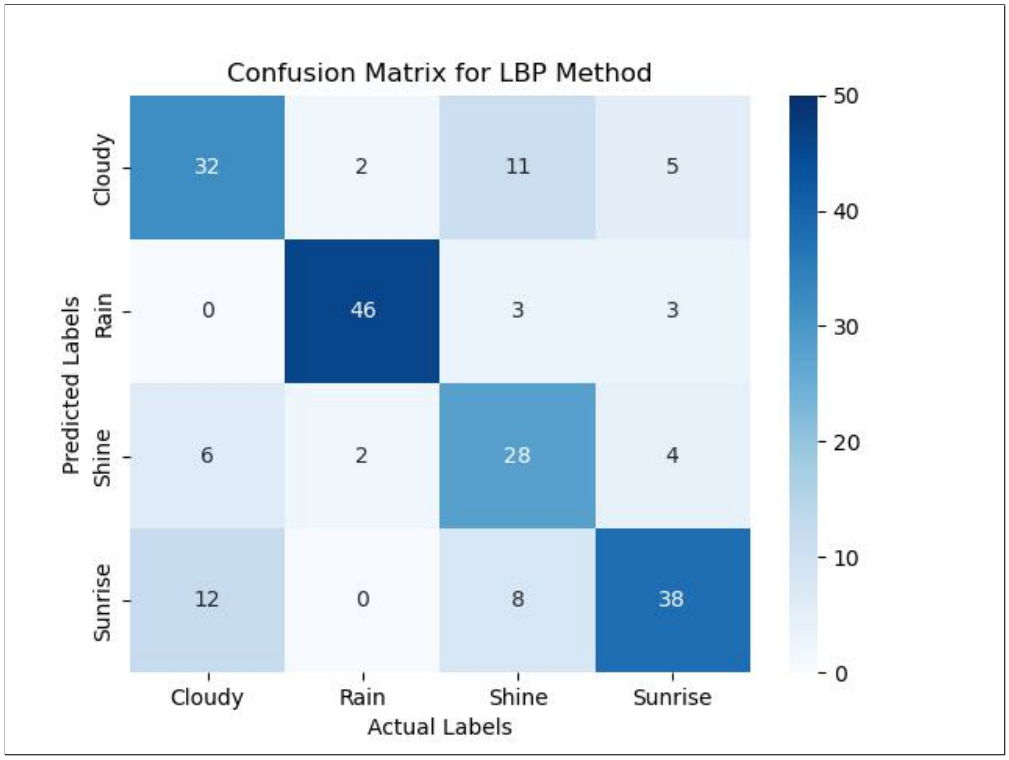


(b) LBP Histogram

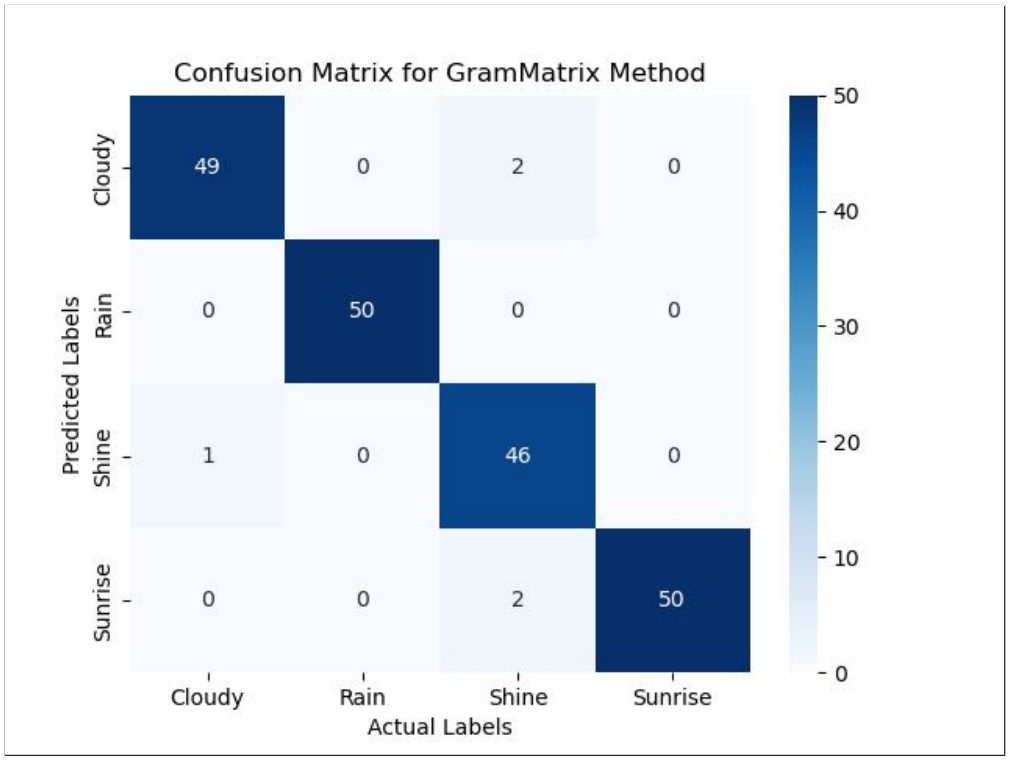


(c) Gram Matrix

Figure 4



(a) Confusion Matrix for LBP



(b) Confusion Matrix for Gram Matrix

Figure 5

	LBP	Gram Matrix
Cloudy	64%	98%
Rain	92%	100%
Shine	56%	92%
Sunrise	76%	100%
Overall	72%	97.5%

Table 1: Accuracy

3. Observations

1. From the images shown above, it can be seen that the Rain class has a texture that is very different from others. The Gram Matrix for Rain class is darker than other showing that it each column has a high correlation with other columns. Similarly, the histogram of Rain class shows high values for each type of encodings. Since it is distinctly very different from other classes, the results show that Rain was predicted more accurately than other classes.

2. From table 1, it can be seen that Gram Matrix has a higher accuracy than LBP since the feature vector are bigger and contain more information about the texture.

3. When the kernel for the SVM classifier was set to "RBF" or "CHI2", which gave an accuracy of 65% for LBP method but 10% accuracy for Gram Matrix method. Similary, when the kernel was set to "Linear", it gave 68% for LBP Method and 97.5% for Gram Matrix. Finally, the kernel set to "INTER" gave 72% for LBP Method and 97.5% for Gram Matrix.

4. Source Code

```
import cv2
import math
import numpy as np
import BitVector as bt
from vgg import VGG19
from skimage import io
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import pandas as pd
import seaborn as sn


#---------------------------Functions-------------------------------#
def getFeatureMatrix_LBP(image, R, P):
    '''
    This function is inspired by Prof. Kak's
    python code to get LBP descriptor vector
    Source code found at:
    https://engineering.purdue.edu/kak/Tutorials/TextureAndColor.pdf
```

```
    BitVector Module by Prof. Kak is also used to compute
    the encodings for the patterns
    '''
    img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (64,64),interpolation = cv2.INTER_AREA)

    binaryHist = np.zeros((P+2))
    totalPixels = img.shape[0]*img.shape[1]

    for i in range(R, img.shape[1]-R):
        for j in range(R, img.shape[0]-R):
            pattern = []
            for p in range(P):
                del_k,del_l = R*math.cos(2*math.pi*p/P), R*math.sin(2*math
                                                        .pi*p/P)
                if abs(del_k) < 0.001: del_k = 0.0
                if abs(del_l) < 0.001: del_l = 0.0
                k, l = i + del_k, j + del_l
                k_b,l_b = int(k),int(l)
                delta_k,delta_l = k-k_b,l-l_b
                if (delta_k < 0.001) and (delta_l < 0.001):
                    pVal = float(img[k_b][l_b])
                elif (delta_l < 0.001):
                    pVal = (1 - delta_k) * img[k_b][l_b] + \
                        delta_k * img[k_b+1][l_b]
                elif (delta_k < 0.001):
                    pVal = (1 - delta_l) * img[k_b][l_b] + \
                        delta_l * img[k_b][l_b+1]
                else:
                    pVal = (1-delta_k)*(1-delta_l)*img[k_b][l_b] \
                        + (1-delta_k)*delta_l*img[k_b][l_b+1] \
                        + delta_k*delta_l*img[k_b+1][l_b+1] \
                        + delta_k*(1-delta_l)*img[k_b+1][l_b]
                if pVal >= img[i][j]:
                    pattern.append(1)
                else:
                    pattern.append(0)
            bv = bt.BitVector(bitlist = pattern)
            intVals = [int(bv<<1) for _ in range(P)]
            minbv = bt.BitVector(intVal = min(intVals), size = P)
            runs = minbv.runs()
            if len(runs)>2:
                binaryHist[P+1] += 1
            elif len(runs)==1 and runs[0][0]=='1':
                binaryHist[P] += 1
            elif len(runs)==1 and runs[0][0]=='0':
                binaryHist[0] += 1
            else:
                binaryHist[len(runs[1])] += 1
    featureVector = binaryHist/totalPixels
    return featureVector

def getFeatureMatrix_Gram(img, vgg, cl, i):
```

```
    featureMat = []
    featureMap = vgg(img)
    for r in range(featureMap.shape[0]):
        mat = featureMap[r]
        featureMat.append(mat.flatten())
    bigGramMat = np.matmul(featureMat, np.transpose(featureMat)) # 512x512
                                                                  matrix
    uppertriangleIdx = np.triu_indices(bigGramMat.shape[1])       # Get
                                                                  upper-triangle indices
    upperTriangleGramMat = bigGramMat[uppertriangleIdx]           # Get
                                                                  upper-triangle of Gram Matrix
    upperTriangleGramMat = np.ravel(upperTriangleGramMat)         # Flatten
                                                                   the matrix
    indx = int(len(upperTriangleGramMat)/2)
    featureVector = upperTriangleGramMat[indx:(indx+1024)]       # Select
                                                                  random 1024 elements

    # Plot the Gram Matrix for each class
    if i==1:
        # scale the values for better visual
        bigGramMat = bigGramMat+0.001
        cmap = plt.cm.gray
        norm = colors.LogNorm()
        plot = cmap(norm(bigGramMat))
        # Save plot with the name of the image
        filename = 'HW7/plots/'+str(cl)+str(i)+'.jpg'
        plt.imsave(fname=filename, arr=plot, format = 'jpg')
    return featureVector

def extractFeature_LBP(dataSetnums, path, labels):
    featureMatrix = []
    labelMatrix = []
    # Iterate through all training images
    for key,num in dataSetnums.items():
        cl = labels[key]
        print("Working on class:", key, cl)
        for i in range(num[0],num[1]+1):
            imgName = str(cl)+str(i)
            img = io.imread(str(path)+imgName+".jpg")
            if img is None or len(img.shape)!=3 or (len(img.shape)==3 and
                                                    img.shape[2]!=3):
                continue
            img = cv2.resize(img, (256,256), interpolation = cv2.
                                                    INTER_AREA)

            # LBP feature Extraction
            featureVector = getFeatureMatrix_LBP(img, R = 1, P = 8)
            print(imgName)
            featureMatrix.append(featureVector)
            labelMatrix.append(key)

            # Plot the histogram
            if i == 1:
                print(featureVector)
```

```
                plt.figure()
                cm = plt.cm.get_cmap('tab20c')
                x = np.random.rand()
                plt.bar(range(len(featureVector)), featureVector, width =
                                                    0.8,color = cm(x),
                                                    edgecolor = 'black')
                plt.title('LBP Histogram of '+str(imgName)+'.jpg')
                plt.savefig('HW7/plots/'+str(imgName)+'_hist.jpg')

    return featureMatrix, labelMatrix

def extractFeatures_Gram(dataSetnums, path, labels):
    gramMatrix = []
    labelMatrix = []
    # Load the model and the provided pretrained weights
    vgg = VGG19()
    vgg.load_weights('HW7/vgg_normalized.pth')
    # Iterate through all training images
    for key,num in dataSetnums.items():
        cl = labels[key]
        print("Working on class:", key, cl)
        for i in range(num[0],num[1]+1):
            imgName = str(cl)+str(i)
            img = io.imread(str(path)+imgName+".jpg")
            if img is None or len(img.shape)!=3 or (len(img.shape)==3 and
                                                    img.shape[2]!=3):
                continue
            img = cv2.resize(img, (256,256), interpolation = cv2.
                                                    INTER_AREA)

            # Gram Matrix feature Extraction
            gramFeatureVector = getFeatureMatrix_Gram(img, vgg, cl, i)
            print(imgName)
            gramMatrix.append(gramFeatureVector)
            labelMatrix.append(key)
    return gramMatrix, labelMatrix

def trainSVM(trainingData, labels):
    # Train SVM multi-class classifier
    svm = cv2.ml.SVM_create()
    svm.setType(cv2.ml.SVM_C_SVC)
    svm.setKernel(cv2.ml.SVM_INTER)
    svm.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 110, 1e-6))
    svm.train(trainingData, cv2.ml.ROW_SAMPLE, labels)
    return svm

def predictLabels(trainedSVM, data):
    labels = trainedSVM.predict(data)[1]
    return labels

def getConfusionMatrix(predictedLabels, actualLabels):
    confusionMatrix = np.zeros((4,4))
    for i in range(len(predictedLabels)):
        predictedLabel = int(predictedLabels[i,0]) - 1
```

```
        actualLabel = actualLabels[i] - 1
        confusionMatrix[predictedLabel, actualLabel] += 1
    return confusionMatrix

def drawConfusionMatrix(cmat, method):
    plot_cm = pd.DataFrame(cmat, index = ["Cloudy", "Rain", "Shine", "
                                          Sunrise"], columns = ["Cloudy", "
                                          Rain", "Shine", "Sunrise"])
    plt.figure()
    sn.heatmap(plot_cm, vmax = 50, annot=True, cmap=sn.color_palette("
                                          Blues", as_cmap=True))
    plt.xlabel("Actual Labels")
    plt.ylabel("Predicted Labels")
    plt.title("Confusion Matrix for "+str(method)+" Method")
    plt.savefig('HW7/plots/'+str(method)+'.jpg')
    return 0

#---------------------------Main Code-----------------------------#
labels = {1: "cloudy", 2: "rain", 3:"shine", 4:"sunrise"}
numTrainingImages = {1: [1,250], 2: [1,165], 3: [1,203], 4: [1,307]}
numTestingImages = {1: [251, 300], 2: [166,215], 3: [204,253], 4: [308,357
                                      ]}

#============Actions=============#
ExtractFeature = False
TrainAndTest = True
#================================#

if ExtractFeature:
    # Extracting Texture Descriptors for training data
    featureMatrix_Training_lbp, labelMatrix_Training_lbp =
                                          extractFeature_LBP(
                                          numTrainingImages, "HW7/data/
                                          training/", labels)
    np.savez_compressed('HW7/data/featureMatrix_Training_lbp', featureMat
                                          = featureMatrix_Training_lbp,
                                          labels = labelMatrix_Training_lbp
                                          )

    gramMatrix_Training_cnn, labelMatrix_Training_cnn =
                                          extractFeatures_Gram(
                                          numTrainingImages, "HW7/data/
                                          training/", labels)
    np.savez_compressed('HW7/data/featureMatrix_Training_cnn', featureMat
                                          = gramMatrix_Training_cnn, labels
                                           = labelMatrix_Training_cnn)

    # Extracting Texture Descriptors for testing data
    featureMatrix_Testing_lbp, labelMatrix_Testing_lbp =
                                          extractFeature_LBP(
                                          numTestingImages, "HW7/data/
                                          testing/", labels)
    np.savez_compressed('HW7/data/featureMatrix_Testing_lbp', featureMat =
                                           featureMatrix_Testing_lbp,
```

```
                                        labels = labelMatrix_Testing_lbp)

    gramMatrix_Testing_cnn, labelMatrix_Testing_cnn = extractFeatures_Gram
                                        (numTestingImages, "HW7/data/
                                        testing/", labels)
    np.savez_compressed('HW7/data/featureMatrix_Testing_cnn', featureMat =
                                         gramMatrix_Testing_cnn, labels =
                                         labelMatrix_Testing_cnn)


if TrainAndTest:
    #============================== TRAIN ==============================#
    # load the feature matrices for lbp method
    trainingData_lbp = np.load('HW7/data/featureMatrix_Training_lbp.npz')
    featureMatrix_Training_lbp = np.matrix(trainingData_lbp['featureMat'],
                                         dtype=np.float32)
    labelMatrix_Training_lbp = np.array(trainingData_lbp['labels'])

    # Load the feature matrices for gram matrix method
    trainingData_cnn = np.load('HW7/data/featureMatrix_Training_cnn.npz')
    gramMatrix_Training_cnn = np.matrix(trainingData_cnn['featureMat'],
                                        dtype=np.float32)
    labelMatrix_Training_cnn = np.array(trainingData_cnn['labels'])

    # Get trained SVM using training data and labels
    svm_lbp = trainSVM(featureMatrix_Training_lbp,
                                        labelMatrix_Training_lbp)
    svm_cnn = trainSVM(gramMatrix_Training_cnn, labelMatrix_Training_cnn)


    #============================== TEST ==============================#
    # Get the test data for lbp method
    testingData_lbp = np.load('HW7/data/featureMatrix_Testing_lbp.npz')
    featureMatrix_Testing_lbp = np.matrix(testingData_lbp['featureMat'],
                                        dtype=np.float32)
    labelMatrix_Testing_lbp = np.array(testingData_lbp['labels'])

    # Get the test data for gram matrix method
    testingData_cnn = np.load('HW7/data/featureMatrix_Testing_cnn.npz')
    gramMatrix_Testing_cnn = np.matrix(testingData_cnn['featureMat'],
                                        dtype=np.float32)
    labelMatrix_Testing_cnn = np.array(testingData_cnn['labels'])

    # Predict labels for test data
    predictedLabels_lbp = predictLabels(svm_lbp, featureMatrix_Testing_lbp
                                        )
    predictedLabels_cnn = predictLabels(svm_cnn, gramMatrix_Testing_cnn)

    # Build the confusion matrix
    confusionMatrix_lbp = getConfusionMatrix(predictedLabels_lbp,
                                        labelMatrix_Testing_lbp)
    confusionMatrix_cnn = getConfusionMatrix(predictedLabels_cnn,
                                        labelMatrix_Testing_cnn)
```

```
    print(confusionMatrix_lbp)
    print(confusionMatrix_cnn)

    drawConfusionMatrix(confusionMatrix_lbp, "LBP")
    drawConfusionMatrix(confusionMatrix_cnn, "GramMatrix")
```