ECE661 Fall 2024: Homework 7

By – Aayush Bhadani

abhadani@purdue.edu

1. Theory Question

In my approach I have tried to combine image pyramids, Gabor filtering and Gram Matrix based approaches to create a texture detector. It tries to exploit the benefits of multi-scale texture representation with robust feature extraction and adaptability.

1.For a given image, we start by creating an image pyramid with several octave layers at various scales. The number of layers depend on the desired level of detail to be captured and computation overload.

2.Next, for each layer in the image pyramid we do Gabor Filtering. We apply a set of Gabor filters to the layer images to produce filtered response maps, where each map corresponds to a specific orientation and scale.

3.Considering these response maps as different channels, we construct Gram Matrix and then vectorize it to obtain the texture descriptor.

In order to reduce the computational load, we can also consider aggregating the response maps at different scale intervals before Gram Matrix computation. This aggregation can be done using averaging, summation, or max pooling over the filtered response maps.

The texture descriptor vector thus produced will be of lower dimension as compared those generated from VGG and ResNet-50 based feature maps. This reduction in dimensionality should assist in faster training of the classifier model. Although the performance may be somewhat inferior to the pure Gram Matrix based texture descriptor derived from VGG/ResNet-50 feature maps.

The input RGB image can either be converted to grayscale for this process or else we can perform step1 and step2 over R,G,B channels of the image simultaneously and finally combine all the response maps across all channels for step3 in which we extract gram matrix based texture descriptor.

5. Tasks

Dataset: Multiclass outdoor weather image data. Categories: [‘cloudy’, ‘rain’, ‘shine’, ‘sunrise’]

RGB to HSI conversion:

Given an image with R,G,B channels we use the following transformation to convert to HSI format:

$$M = \max(R, G, B)$$

$$m = \min(R, G, B)$$

$$c = M - m$$

$$I = \frac{R + G + B}{3}$$

$$H = \begin{cases} 60\left(\frac{G - B}{c} \ mod\ 6\right) & if\ M = R, c \neq 0 \\ 60\left(\frac{B - R}{c} + 2\right) & if\ M = G, c \neq 0 \\ 60\left(\frac{R - G}{c} + 4\right) & if\ M = B, c \neq 0 \\ 0 & if\ c = 0 \end{cases}$$

$$S = \begin{cases} 0 & if\ c = 0 \\ 1 - \frac{m}{I} & if\ c \neq 0 \end{cases}$$

Local Binary Pattern (LBP):

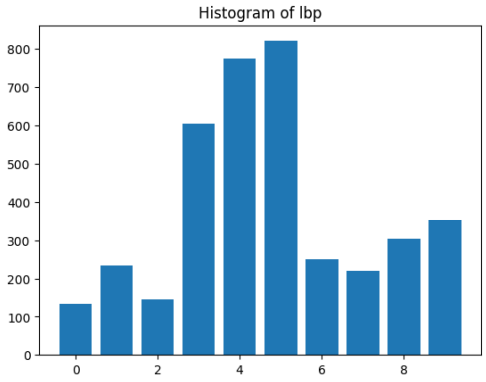
For each image we extract a histogram feature vector. We start by converting image to HSI format and then use Hue channel. To make computation more feasible, image is downsized to (64x64).

In the implementation we set P=8, which would mean 8 neighbouring points around. Radius R is 1.

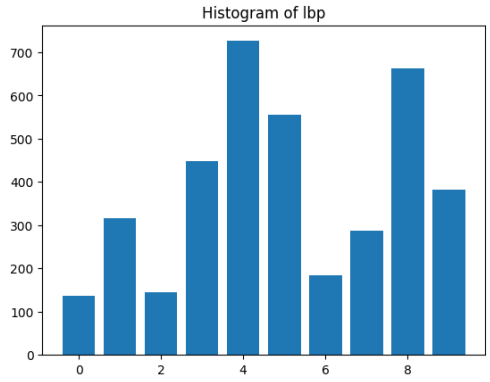
R = 2,3 were also explored. Best classification accuracy on the test set for the task using lbp features was obtained with R = 2.

LBP Histograms of images belonging to different classes are presented below:

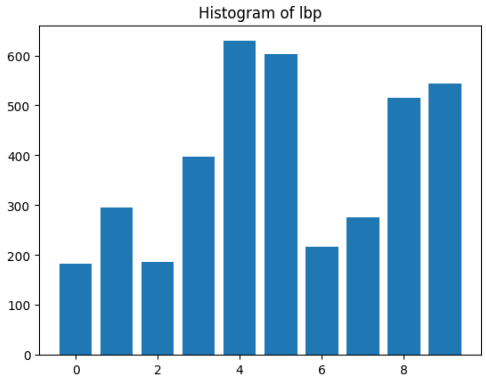
(a) Cloudy



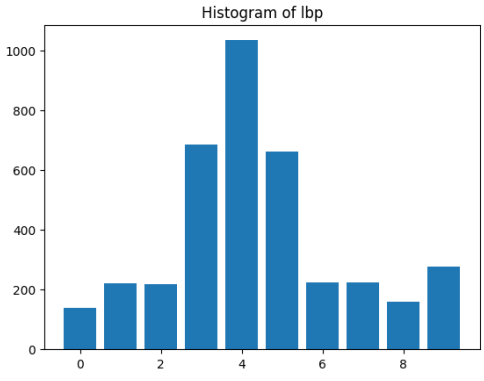
(b) Rain



(c) Shine



(d) Sunrise



Gram Matrix (GM):

To compute the Gram Matrix, we start from a feature map F_l of the shape (N_l, M_l) where N_l is the number of channels and $M_l = W_l \times H_l$ is the number of spatial locations. The Gram matrix is given as, $G = F_l \cdot F_l^T \in \mathbb{R}^{N_l \times N_l}$. As this is a symmetric matrix we can just retain upper triangular matrix and vectorize it get out Gram matrix based description vector $v_{gram} \in \mathbb{R}^{N_l(N_l+1)/2}$.

For generating feature map, custom VGG-19 and dual resolution ResNet-50 based networks have been used. So, we have feature maps from:

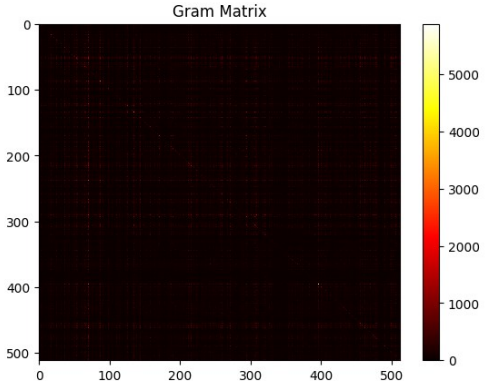
- VGG-19 (vgg): output – [C=512, h=H/16, w=W/16]
- ResNet-50 Coarse (resnet_coarse): output – [C=1024, h=H/16, w=W/16]
- ResNet-50 Fine (resnet_fine): output – [C=512, h=H/8, w=W/8]

Images are pre-processed before being fed to these abovementioned networks where these images get resized to (256X256).

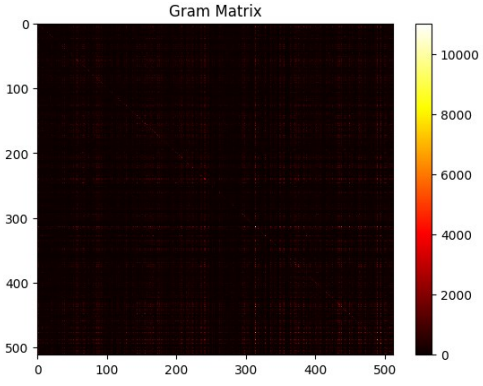
Gram Matrices (un-normalized) of images belonging to different classes are presented below:

For VGG:

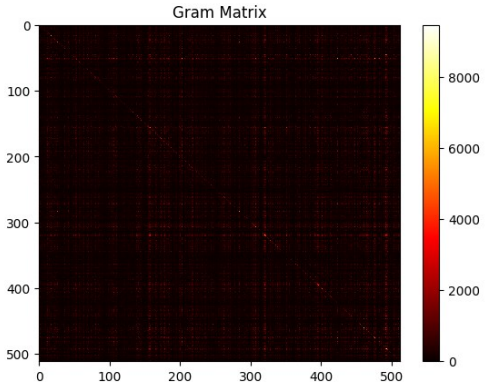
(a) cloudy



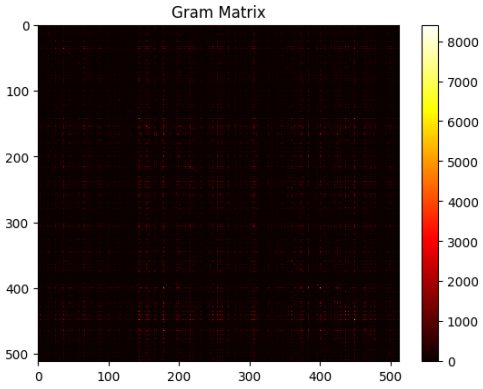
(b) rain



(c) shine

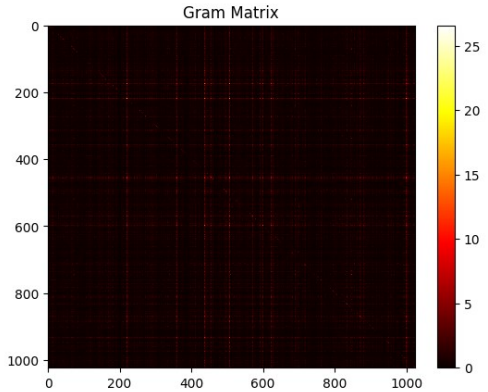


(d) sunrise

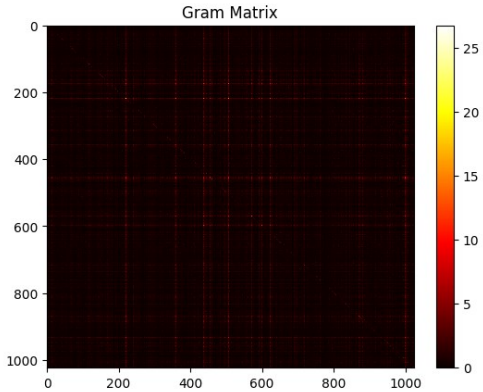


For ResNet_coarse:

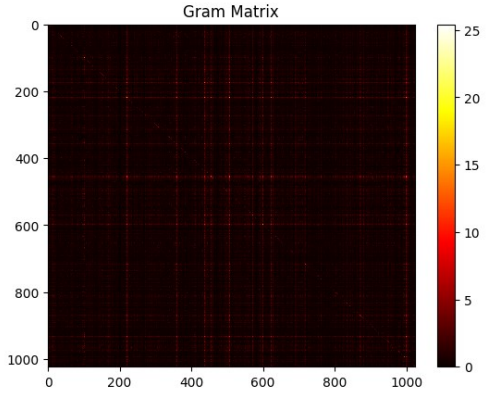
(a) Cloudy



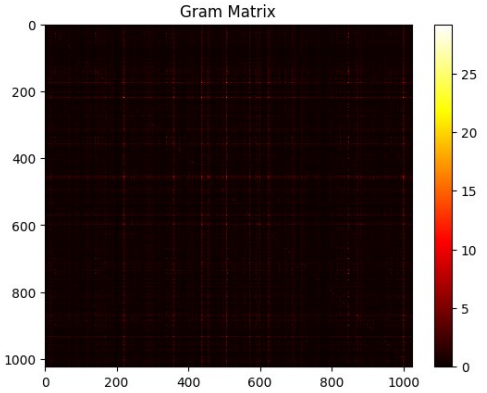
(b) rain



(c) shine

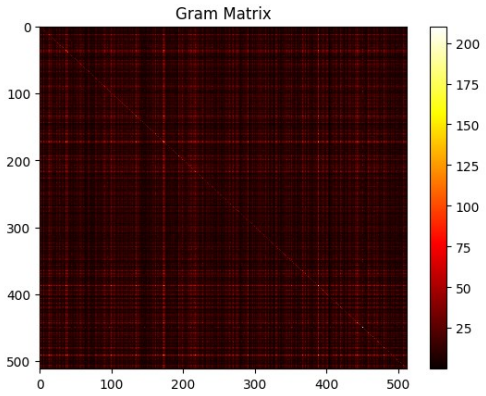


(d) sunrise

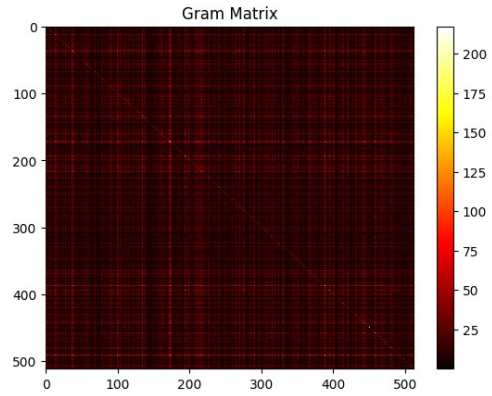


For ResNet_fine:

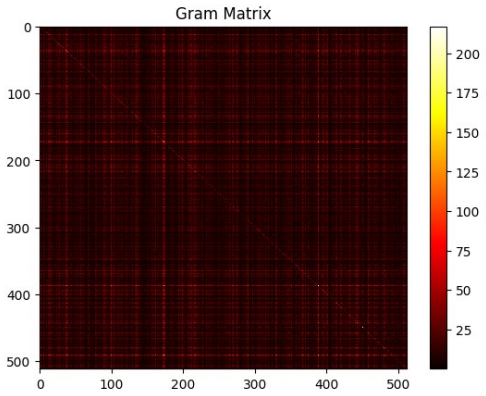
(a) cloudy



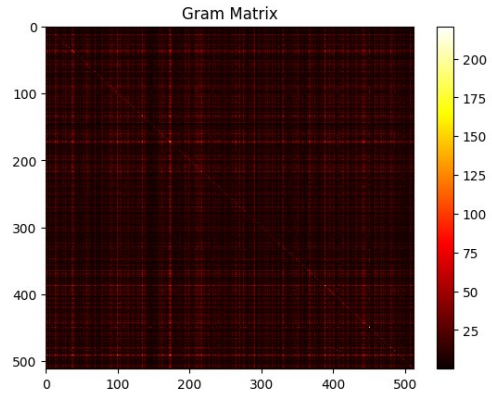
(b) rain



(c) shine



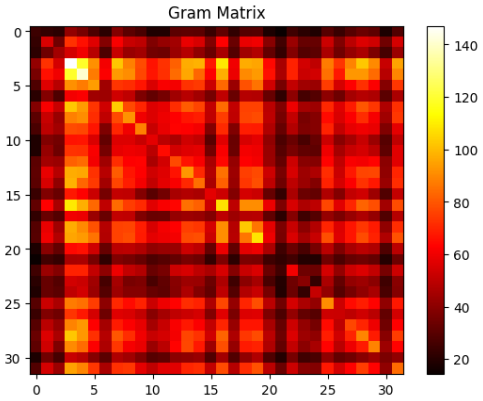
(d) sunrise



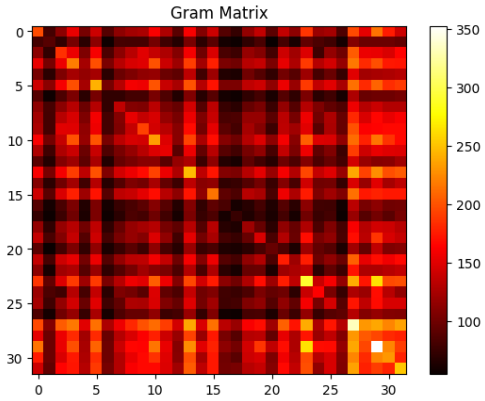
Down-sampled versions of Gram Matrices (32X32): used block averaging

For VGG:

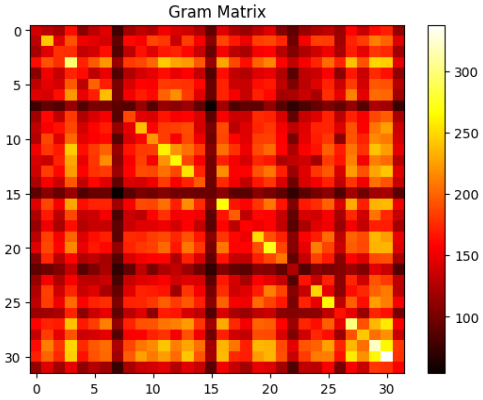
(a) cloudy



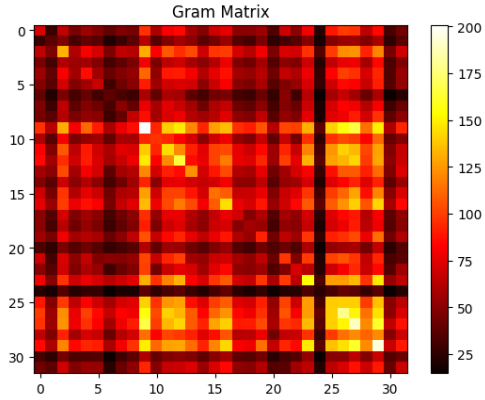
(b) rain



(c) shine

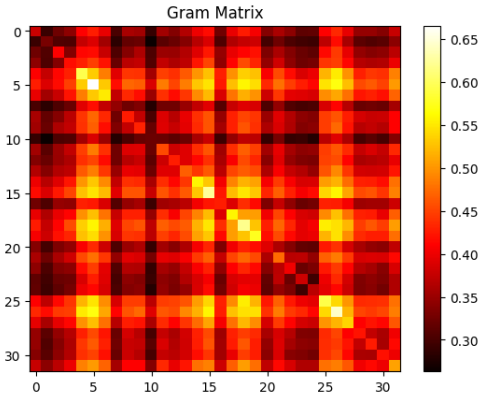


(d) sunrise

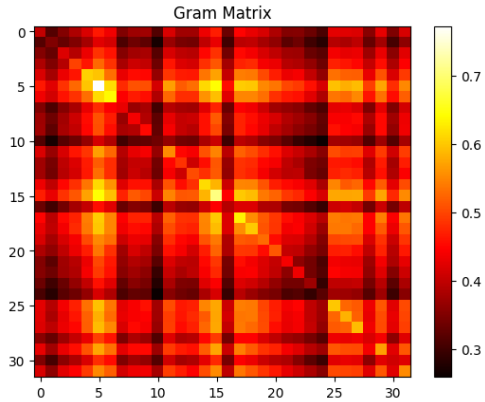


For ResNet_coarse:

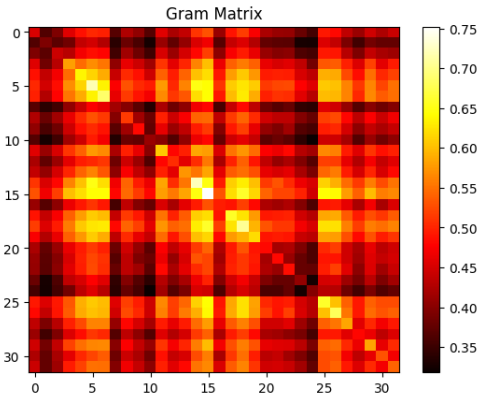
(a) cloudy



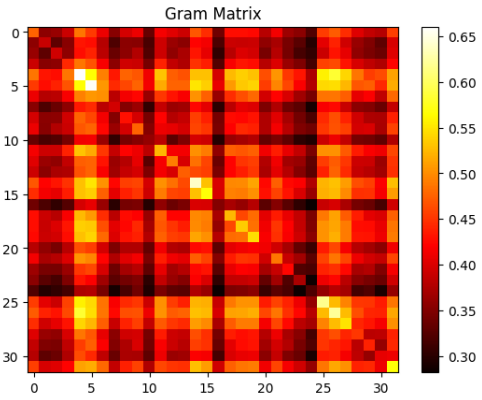
(b) rain



(c) shine

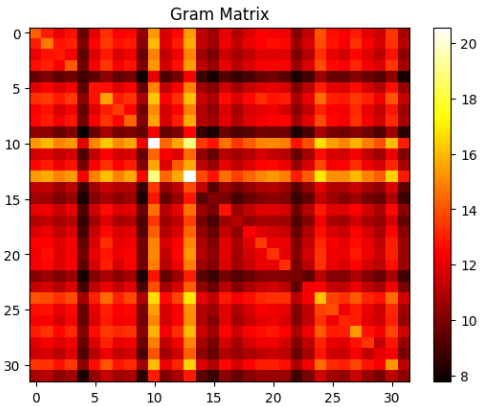


(d) sunrise

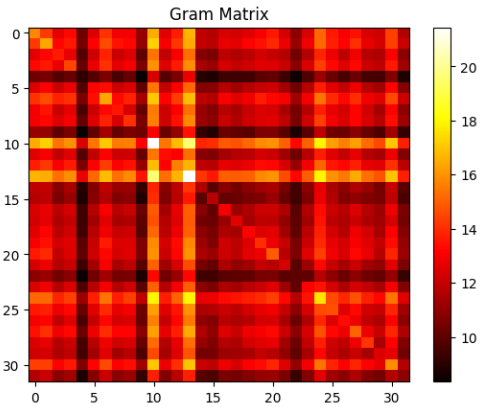


For ResNet_fine:

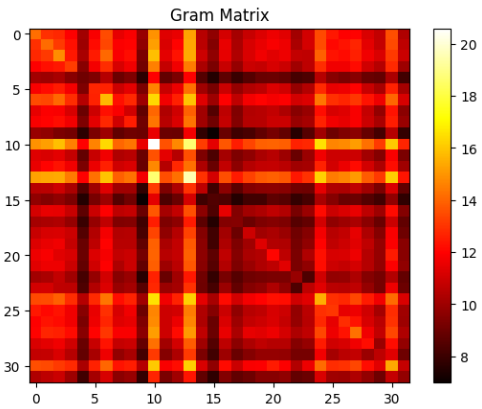
(a) cloudy



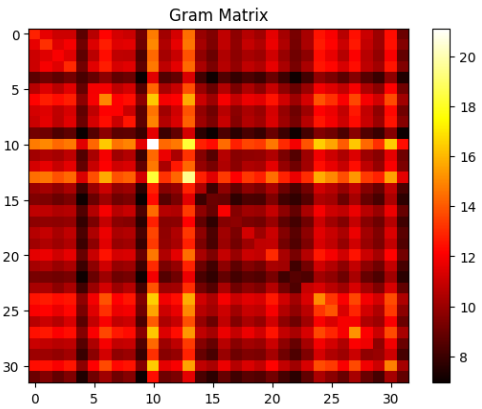
(b) rain



(c) shine



(d) sunrise



A range of correlation levels is observed among the feature channels across different classes. Some have higher correlation while others have low. This suggests that certain feature channels are more closely related than others. Thus, not all feature channels are strongly correlated.

Extra: Adaptive Instance Normalization (AdaIN):

We use feature maps generated by VGG-19 and ResNet-50 (coarse/fine) – same as prev. GM section

For a given feature map F_l, the channel normalization parameters are given as per channels mean (μ) and variance (σ) values:

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

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

Finally, by stacking all mean and variance values across all feature channels we obtain our feature descriptor $v_{norm} = (\mu_0, \sigma_0, \mu_1, \sigma_1, \ldots., \mu_{N_l}, \sigma_{N_l}) \in \mathbb{R}^{2N_l}$

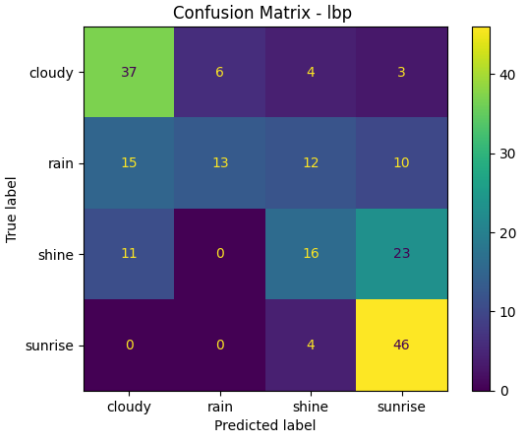
Classification:

SVM based multi-class classifier is trained to fit the training set and make predictions on testing set. For this purpose, scikit-learn based implementation has been used. Linear kernel is selected due to high dimensionality of the features. For the given multi-class SVM problem, one-vs-one (ovo) strategy has been opted.

Accuracy of the trained classifier model along with confusion matrix is presented for various cases. One correctly classified and one incorrectly classified image for each model is also displayed.

LBP feature based:

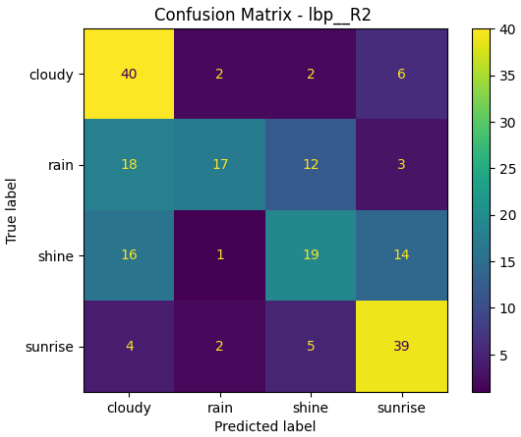
- SVM Classifier based on LBP features with R =1: Accuracy = 56%



Correctly Classified: shine

Misclassified as: cloudy (Actual: rain)

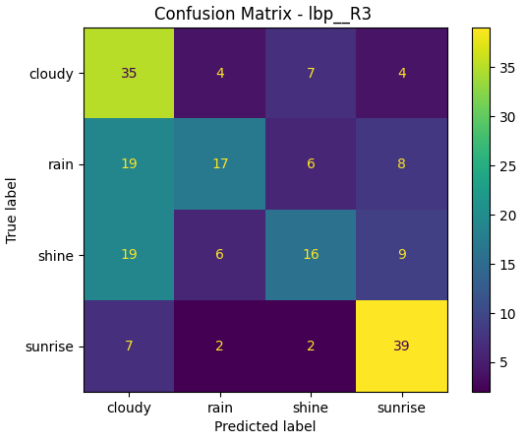
- SVM Classifier based on LBP features with R =2: Accuracy = 57.5%



Correctly Classified: cloudy

Misclassified as: shine (Actual: sunrise)

- SVM Classifier based on LBP features with R =3: Accuracy = 53.5%

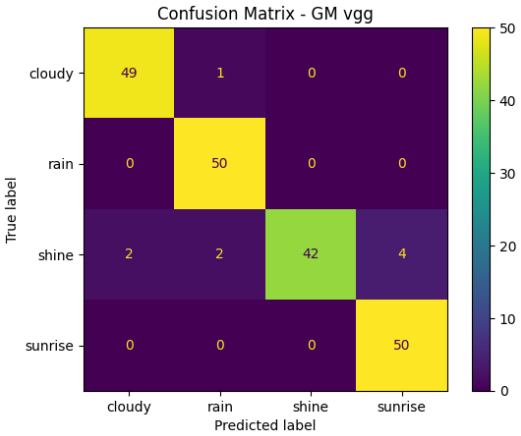


Correctly Classified: shine

Misclassified as: cloudy (Actual: sunrise)

GM feature based:

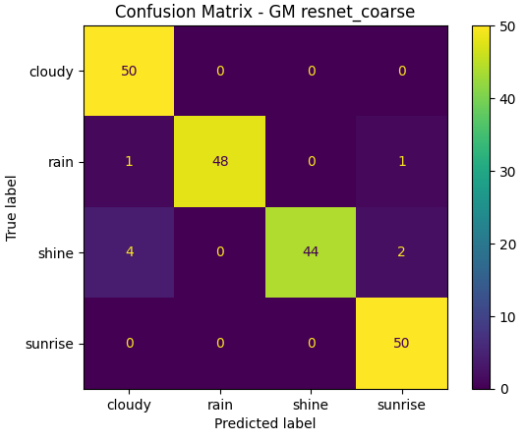
- SVM Classifier based on GM features from VGG: Accuracy=95.5%



Correctly Classified: cloudy

Misclassified as: cloudy (Actual: shine)

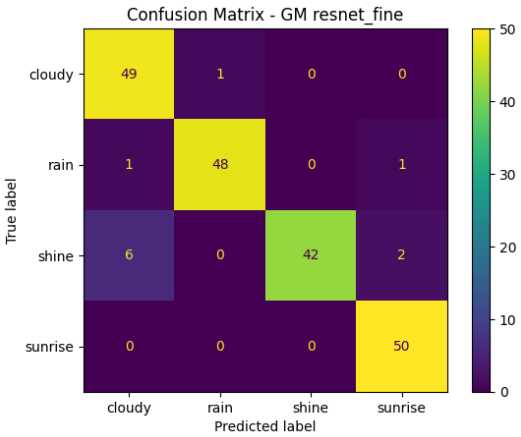
- SVM Classifier based on GM features from ResNet-50 coarse: Accuracy=96%



Correctly Classified: shine

Misclassified as: sunrise (Actual: rain)

- SVM Classifier based on GM features from ResNet-50 fine: Accuracy=94.5%

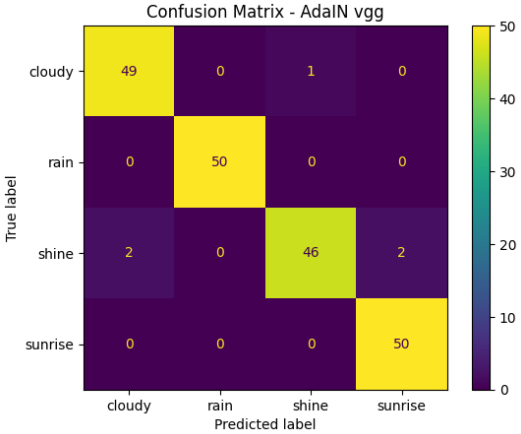


Correctly Classified: shine

Misclassified as: rain (Actual: cloudy)

AdaIN feature based:

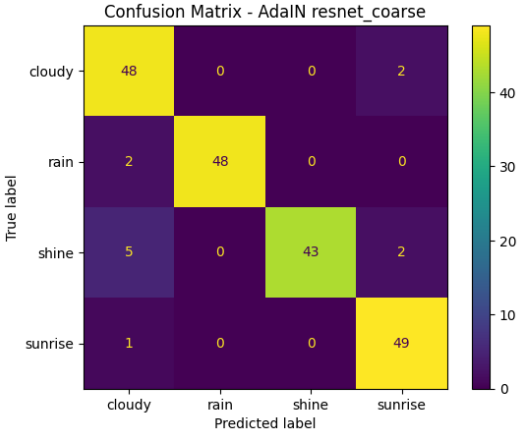
- SVM Classifier based on AdaIN features from VGG: Accuracy=97.5%



Correctly Classified: shine

Misclassified as: sunrise (Actual: shine)

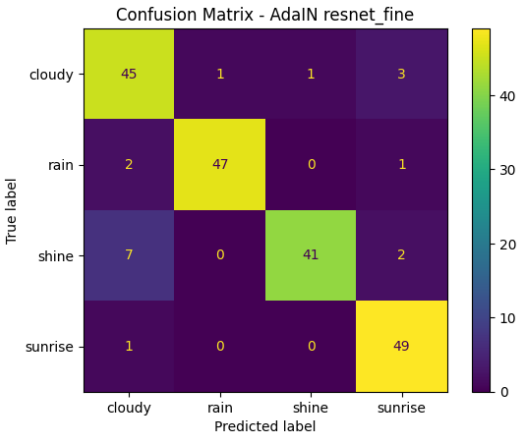
- SVM Classifier based on AdaIN features from ResNet-50 coarse: Accuracy=94%



Correctly Classified: sunrise

Misclassified as: cloudy (Actual: shine)

- SVM Classifier based on AdaIN features from ResNet-50 fine: Accuracy=91%



Correctly Classified: cloudy

Misclassified as: cloudy (Actual: sunrise)

Discussion:

Classifier based on LBP features performed poorly compared to those based on gram matrix and channel normalization (AdaIN). This can be attributed to lbp inability to capture broader, higher-level features as it is focuses only in a small neighbourhood around each central pixel. The generated Lbp histogram feature vectors were not very different for images belonging to different classes and thus caused difficulty for the model to correctly classifying the classes accordingly. In testing lbp feature based classifiers with R=1,2,3, highest accuracy of about 57.5% was obtained with R=2. Good amount of overfitting also observed for such lbp based models.

Gram Matrix based classifiers performed quite well, reaching accuracy of 96% with ResNet-50 coarse feature map. However, Gram Matrix descriptor is of high dimension which makes training of the classifier model quite slow and difficult. AdaIN based descriptor vectors are shorter than Gram Matrix descriptor vectors. This does not affect the accuracy of the classifiers much as we only see a slight dip in the performance of the classifiers which are based on AdaIN descriptors from ResNet feature maps as compared to those using Gram matrix descriptors. Although, VGG feature map based AdaIN descriptor classifier was able to achieve accuracy of 97.5%.

Acknowledgement:

https://engineering.purdue.edu/kak/computervision/

Code

```
# !pip install BitVector-3.4.9.tar.gz
# !unzip HW7-Auxilliary.zip


import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
import random
import math
import os
import sys
import BitVector
sys.path.append('/content/HW7-Auxilliary')
import vgg_and_resnet
from skimage import io
from tqdm import tqdm
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import shuffle



#data location and categories
training_data_path = '/content/HW7-Auxilliary/data/training'
testing_data_path ='/content/HW7-Auxilliary/data/testing'

categories = ['cloudy','rain','shine','sunrise']



#VGG and RESNET models:
vgg = vgg_and_resnet.VGG19()
vgg.load_weights('/content/HW7-Auxilliary/vgg_normalized.pth')

encoder_name='resnet50' # Valid options ['resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152']
resnet = vgg_and_resnet.CustomResNet(encoder=encoder_name)



#RGB to HSI
def RGB_2_Hue(image):
    image = image.astype('float')
    image /= 255.0
    R,G,B = image[:,:,0], image[:,:,1], image[:,:,2]
    I = (R+G+B)/3
    M = np.maximum(np.maximum(R,G),B)
    m = np.minimum(np.minimum(R,G),B)
    c = M-m
    S = np.zeros_like(I)
    S[c!=0] = 1 - m[c!=0]/I[c!=0]
    H = np.zeros_like(I)
    crit1 = (M==R) & (c!=0)
    H[crit1] = 60*(((G[crit1]-B[crit1])/c[crit1])%6)
    crit2 = (M==G) & (c!=0)
    H[crit2] = 60*(((B[crit2]-R[crit2])/c[crit2])+2)
    crit3 = (M==B) & (c!=0)
    H[crit3] = 60*(((R[crit3]-G[crit3])/c[crit3])+4)

    return H,S,I



#LBP histogram

def BV_encoding(BV_runs,P):
    if len(BV_runs) == 1 and BV_runs[0][0] == '0':
        encoding = 0
    elif len(BV_runs) == 1 and BV_runs[0][0] == '1':
        encoding = P
    elif len(BV_runs) > 2:
        encoding = P+1
    else:
        encoding = len(BV_runs[1])
```

```
    return encoding

def lbp(image_H, draw_plot=False):
    R = 1
    P = 8
    eps = 0.001
    hist_lbp = np.zeros(P+2)
    # hist_lbp_list = []
    image_H =  cv.resize(image_H, (64,64), interpolation=cv.INTER_AREA)
    H,W = image_H.shape
    for i in range(R,H-R):
        for j in range(R,W-R):
            pattern = []
            # print(f'pixel at {(i,j)}')
            for p in range(P):
                del_vals =  np.array([R*math.cos(2*math.pi*p/P), R*math.sin(2*math.pi*p/P)])
                del_vals[abs(del_vals) < eps] = 0
                k,l = int(i + del_vals[0]), int(j+del_vals[1])
                del_k, del_l = (i + del_vals[0]) - k, (j+del_vals[1]) - l
                if del_l < eps and del_k < eps:
                    imgVal_at_P = image_H[k,l]
                else:
                    imgVal_at_P = (1-del_k)*(1-del_l)*image_H[k,l] + (1-del_k)*del_l*image_H[k,l+1] + del_k*(1-del_l)*image_H
                pattern.append(1) if imgVal_at_P >= image_H[i,j] else pattern.append(0)
            # print(pattern)
            bv = BitVector.BitVector(bitlist=pattern)
            min_BV = BitVector.BitVector(intVal = min([int(bv << 1) for _ in range(P)]), size=P)
            BV_runs = min_BV.runs()
            hist_lbp[BV_encoding(BV_runs,P)]+=1

    if draw_plot:
        print('hist_lbp : ',hist_lbp)
        plt.bar(range(len(hist_lbp)), hist_lbp, )
        plt.title('Histogram of lbp')
        plt.show()

    return hist_lbp

# pattern_listA = lbp(image_H)


# Gram Matrix
def GramMatrix_feature(image, model='vgg', draw_plot=False, og_plt=True):
    image = vgg_and_resnet.transform.resize(image,(256,256))
    if model == 'vgg':
        feature = vgg(image)
    elif model == 'resnet_coarse':
        feature = resnet(image)[0]
    elif model == 'resnet_fine':
        feature = resnet(image)[1]
    c,h,w = feature.shape
    F = feature.reshape(c,-1)
    G = F@F.T

    if draw_plot:
        if og_plt:
          plt.imshow(G,cmap='hot', aspect='auto', interpolation='nearest')
        else: #downsampled plot
          if model == 'resnet_coarse':
            plt.imshow(G.reshape(32, 32, 32, 32).mean(axis=(1, 3)),cmap='hot', aspect='auto', interpolation='nearest')
          else:
            plt.imshow(G.reshape(32, 16, 32, 16).mean(axis=(1, 3)),cmap='hot', aspect='auto', interpolation='nearest')
        plt.colorbar()
        plt.title('Gram Matrix')
        plt.show()

    G /= c*h*w  # G /= np.max(G)
    V_gram = (G[np.triu_indices(G.shape[0])])
    # print(V_gram)
    return V_gram

# GramMatrix_feature(image)


# AdaIN - Channel normalization
def AdaIN_feature(image, model='vgg'):
```

```
    image = vgg_and_resnet.transform.resize(image,(256,256))
    if model == 'vgg':
        feature = vgg(image)
    elif model == 'resnet_coarse':
        feature = resnet(image)[0]
    elif model == 'resnet_fine':
        feature = resnet(image)[1]
    c,h,w = feature.shape
    F = feature.reshape(c,-1)
    u_l = np.mean(F,axis=1)
    sig_l = np.std(F,axis=1)
    V_norm = np.array(sum(zip(u_l,sig_l),()))

    return V_norm

# AdaIN_feature(image)


#load and label data
def data_load(categories, data_path=training_data_path):
    data_dir = sorted(os.listdir(data_path))
    categories_label = []
    img_data = []
    for img_name in data_dir:
        if img_name == '.DS_Store':
            continue
        img_path = os.path.join(data_path,img_name)
        img = io.imread(img_path)
        if len(img.shape) == 3 and img.shape[2]==3:
            if 'cloudy' in img_name.lower():
                categories_label.append(categories.index('cloudy'))
            elif 'rain' in img_name.lower():
                categories_label.append(categories.index('rain'))
            elif 'shine' in img_name.lower():
                categories_label.append(categories.index('shine'))
            elif 'sunrise' in img_name.lower():
                categories_label.append(categories.index('sunrise'))
            img_data.append(img)

    return categories_label, img_data

train_categories_label, train_img_data = data_load(categories, data_path=training_data_path)
test_categories_label, test_img_data = data_load(categories, data_path=testing_data_path)


# extract and save lbp features
def extractSave_lbp_features(img_data_src = train_img_data):
    hist_lbp_list = []
    for img in tqdm(img_data_src):
        img_H = RGB_2_Hue(img)[0]
        hist_lbp_list.append(lbp(img_H))
    data_src = 'train' if img_data_src == train_img_data else 'test'
    np.savez_compressed(f'{data_src}_lbp_features.npz',*hist_lbp_list)

extractSave_lbp_features(img_data_src=train_img_data)
extractSave_lbp_features(img_data_src=test_img_data)



#extract and save GM features based on diff. networks
def extractSave_GM_features(img_data_src=train_img_data, model='vgg'):
    V_gram_list = []
    for img in tqdm(img_data_src):
        V_gram_list.append(GramMatrix_feature(img,model))
    data_src = 'train' if img_data_src == train_img_data else 'test'
    np.savez_compressed(f'{data_src}_{model}_GM_features.npz',*V_gram_list)

extractSave_GM_features(train_img_data, 'vgg')
extractSave_GM_features(test_img_data, 'vgg')

extractSave_GM_features(train_img_data, 'resnet_coarse')
extractSave_GM_features(test_img_data, 'resnet_coarse')

extractSave_GM_features(train_img_data, 'resnet_fine')
extractSave_GM_features(test_img_data, 'resnet_fine')
```

```
#extract and save AdaIN features based on diff. networks
def extractSave_AdaIN_features(img_data_src=train_img_data, model='vgg'):
    V_norm_list = []
    for img in tqdm(img_data_src):
        V_norm_list.append(AdaIN_feature(img,model))
    data_src = 'train' if img_data_src == train_img_data else 'test'
    np.savez_compressed(f'{data_src}_{model}_AdaIN_features.npz',*V_norm_list)

extractSave_AdaIN_features(train_img_data, 'vgg')
extractSave_AdaIN_features(test_img_data, 'vgg')

extractSave_AdaIN_features(train_img_data, 'resnet_coarse')
extractSave_AdaIN_features(test_img_data, 'resnet_coarse')

extractSave_AdaIN_features(train_img_data, 'resnet_fine')
extractSave_AdaIN_features(test_img_data, 'resnet_fine')


# SVM Classifier model -> accuracy, confusion matrix
def SVM_classifer_model(Xtrain,Xtest,Ytrain=train_categories_label,Ytest=test_categories_label,categories=categories,fType='
    Xtrain, Ytrain = shuffle(Xtrain, Ytrain, random_state=42)
    model = SVC(kernel='linear', decision_function_shape='ovo')
    model.fit(Xtrain,Ytrain)
    pred = model.predict(Xtest)
    accuracy = accuracy_score(Ytest,pred,normalize=True)
    report = classification_report(Ytest,pred)

    predTrain = model.predict(Xtrain)
    accuracyTrain = accuracy_score(Ytrain,predTrain,normalize=True)
    print(f'Train Accuracy: {accuracyTrain:.4f}')

    print(f'Test Accuracy: {accuracy:.4f}')
    print('Classification Report:')
    print(report)

    ## index to display
    # correct_index = (np.where(pred == Ytest)[0])[0]
    # misclassified_index = (np.where(pred != Ytest)[0])[0]

    correct_index = random.choice(np.where(pred == Ytest)[0])
    misclassified_index = random.choice(np.where(pred != Ytest)[0])

    # Display one correct classification
    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    plt.imshow(test_img_data[correct_index])
    plt.title(f'Correctly Classified: {categories[Ytest[correct_index]]}')
    plt.axis('off')

    # Display one misclassification
    plt.subplot(1, 2, 2)
    plt.imshow(test_img_data[misclassified_index])
    plt.title(f'Misclassified as: {categories[pred[misclassified_index]]} (Actual: {categories[Ytest[misclassified_index]]})'
    plt.axis('off')
    plt.show()

    # Confusion Matrix
    conf_matrix = confusion_matrix(Ytest, pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=categories)
    disp.plot()
    plt.title(f'Confusion Matrix - {fType} {net}')
    plt.show()
```

```
#load saved lbp features
loaded_train_lbp = np.load('train_lbp_features.npz')
loaded_test_lbp = np.load('test_lbp_features.npz')

train_lbp = [loaded_train_lbp[key] for key in loaded_train_lbp]
test_lbp = [loaded_test_lbp[key] for key in loaded_test_lbp]


# load saved AdaIN features
loaded_train_AdaIN_vgg = np.load('train_vgg_AdaIN_features.npz')
loaded_test_AdaIN_vgg = np.load('test_vgg_AdaIN_features.npz')
loaded_train_AdaIN_resnet_coarse = np.load('train_resnet_coarse_AdaIN_features.npz')
loaded_test_AdaIN_resnet_coarse = np.load('test_resnet_coarse_AdaIN_features.npz')
loaded_train_AdaIN_resnet_fine = np.load('train_resnet_fine_AdaIN_features.npz')
loaded_test_AdaIN_resnet_fine= np.load('test_resnet_fine_AdaIN_features.npz')

train_AdaIN_vgg = [loaded_train_AdaIN_vgg[key] for key in loaded_train_AdaIN_vgg]
test_AdaIN_vgg = [loaded_test_AdaIN_vgg[key] for key in loaded_test_AdaIN_vgg]
train_AdaIN_resnet_coarse = [loaded_train_AdaIN_resnet_coarse[key] for key in loaded_train_AdaIN_resnet_coarse]
test_AdaIN_resnet_coarse = [loaded_test_AdaIN_resnet_coarse[key] for key in loaded_test_AdaIN_resnet_coarse]
train_AdaIN_resnet_fine = [loaded_train_AdaIN_resnet_fine[key] for key in loaded_train_AdaIN_resnet_fine]
test_AdaIN_resnet_fine = [loaded_test_AdaIN_resnet_fine[key] for key in loaded_test_AdaIN_resnet_fine]



# load saved GM features
loaded_train_GM_vgg = np.load('train_vgg_GM_features.npz')
loaded_test_GM_vgg = np.load('test_vgg_GM_features.npz')
loaded_train_GM_resnet_coarse = np.load('train_resnet_coarse_GM_features.npz')
loaded_test_GM_resnet_coarse = np.load('test_resnet_coarse_GM_features.npz')
loaded_train_GM_resnet_fine = np.load('train_resnet_fine_GM_features.npz')
loaded_test_GM_resnet_fine= np.load('test_resnet_fine_GM_features.npz')

train_GM_vgg = [loaded_train_GM_vgg[key] for key in loaded_train_GM_vgg]
test_GM_vgg = [loaded_test_GM_vgg[key] for key in loaded_test_GM_vgg]
train_GM_resnet_coarse = [loaded_train_GM_resnet_coarse[key] for key in loaded_train_GM_resnet_coarse]
test_GM_resnet_coarse = [loaded_test_GM_resnet_coarse[key] for key in loaded_test_GM_resnet_coarse]
train_GM_resnet_fine = [loaded_train_GM_resnet_fine[key] for key in loaded_train_GM_resnet_fine]
test_GM_resnet_fine = [loaded_test_GM_resnet_fine[key] for key in loaded_test_GM_resnet_fine]



#function to plot lbp histograms and GM for different classes
def plot_Hist_n_GM(og_plt):

  ## Class Histograms
  #lbp
  print('-----------------Histogram of LBP-----------------------')
  for i in range(len(categories)):
    print('Class : ',categories[i])
    idx = test_categories_label.index(i)
    test_img = test_img_data[idx]
    img_H = RGB_2_Hue(test_img)[0]
    hist_lbp = lbp(img_H,draw_plot=True)

  print('')
  ## Gram Matrix
  # vgg #resnet_coarse #resnet_fine
  print('--------------------GRAM MATRIX--------------------------')
  for net in ['vgg','resnet_coarse','resnet_fine']:
    print('Model : ',net)
    print('--------------------------------------------------------------------------------')
    for i in range(len(categories)):
      print('Class : ',categories[i])
      idx = test_categories_label.index(i)
      test_img = test_img_data[idx]
      V_gram = GramMatrix_feature(test_img,model=net,draw_plot=True,og_plt=og_plt)
      print('')

plot_Hist_n_GM(og_plt=True)




SVM_classifer_model(train_lbp,test_lbp,fType='lbp')
```

```
SVM_classifer_model(train_AdaIN_vgg,test_AdaIN_vgg,fType='AdaIN',net='vgg')

SVM_classifer_model(train_AdaIN_resnet_fine,test_AdaIN_resnet_fine,fType='AdaIN',net='resnet_fine')

SVM_classifer_model(train_AdaIN_resnet_coarse,test_AdaIN_resnet_coarse,fType='AdaIN',net='resnet_coarse')

SVM_classifer_model(train_GM_vgg,test_GM_vgg,fType='GM',net='vgg')

SVM_classifer_model(train_GM_resnet_fine,test_GM_resnet_fine,fType='GM',net='resnet_fine')

SVM_classifer_model(train_GM_resnet_coarse,test_GM_resnet_coarse,fType='GM',net='resnet_coarse')
```