ECE661: Homework 7

Submission By: Joseph Wang
wang3450@purdue.edu

Prepared for Dr. Avinash Kak
GTA: Fangda Li
Purdue University
November 2, 2022

Contents

List of Figures

1 Theory Question

Question: The reading material for Lecture 16 presents three different approaches to characterizing the texture in an image: 1). using the gray level co-occurrence matrix (CLCM); 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.

Gray Level Co-occurence Matrix (GLCM)

- The basic idea of the GLCM method is to estimate the join probability distribution $P[x_1, x_2]$ for the grey-scale values in an image, where x_1 is the grey-scale value at any randomly selected pixel in the image and x_2 the grey-scale value at another pixel that is at a specific vector distance d from the first pixel.
- The algorithm entails performing a raster scan on the grey-scale image and populating the gray level co-occurrence matrix. The matrix is then normalized to reflect a joint probability distribution.
- The texture of the image can thus be characterized by the shape of the joint distribution. Other properties of the texture such as entropy, energy, and contrast can also be quantified using this method.

Local Binary Pattern (LBP)

- The basic idea of LBP is to characterize the grey-scale variations around a pixel through runs of 0s and 1s. All possible permutations of runs of 0s and 1s are then counted in the form of a histogram to serve as a rotational and grey-scale invariant characterization of image texture.
- A raster scan is performed on the grey-scale image, whereby at each pixel value a binary pattern is established and encoded into an integer.
- The frequencies of all the possible encodings then become the images feature descriptor.

Gabor Filter Family

- Gabor filters are spatially localized operators for analyzing an image for periodicities at different frequencies and in different directions. Unlike the aforementioned GLCM and LBP methods, techniques that leverage Gabor filters fall under the class of structure based texture characterizers.
- Gabor filters are highly localized Fourier transforms in which the localization is achieved by applying a Gaussian decay function to the pixel.
- Whereas the Gaussian weighting gives us the localization needed, the direction of the periodicities in the underlying Fourier kernel allows us to characterize a texture in that direction.

Question: With regard to representing color in images, answer Right or Wrong for the following questions and provide a brief justification for each:

(a) **RGB and HSI are just linear variants of each other**
Wrong. There does not exists a linear mapping from RGB to HSI. In order to derive a transformation between the RGB space and the HSI space, one must derive nonlinear transformation equations that relate the RGB vectors to the HSI space described by cylindrical coordinates.

(b) **The color space L*a*b* is a nonlinear model of color perception**
Right. L*a*b* is a nonlinear model of color perception where L stands for luminance, and a* and b* represent two color opponent dimensions. Since L*a*b* is a nonlinear color space, transformation from other spaces are quite complex.

(c) **Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.**
Right. It is not possible to create optical filters with exactly the same spectral responses as the three types of cone cells in the retina. That is why every color space model is simply an approximation.

2 VGG19 Based Style Classifier

This section describes the steps followed in order to perform style classification with the help of a pretrained VGG19 neural network. Those steps are as follows:

1. Given the encoder layers and pretrained weights of the VGG19 neural network, extract the feature maps F for all training and test images.

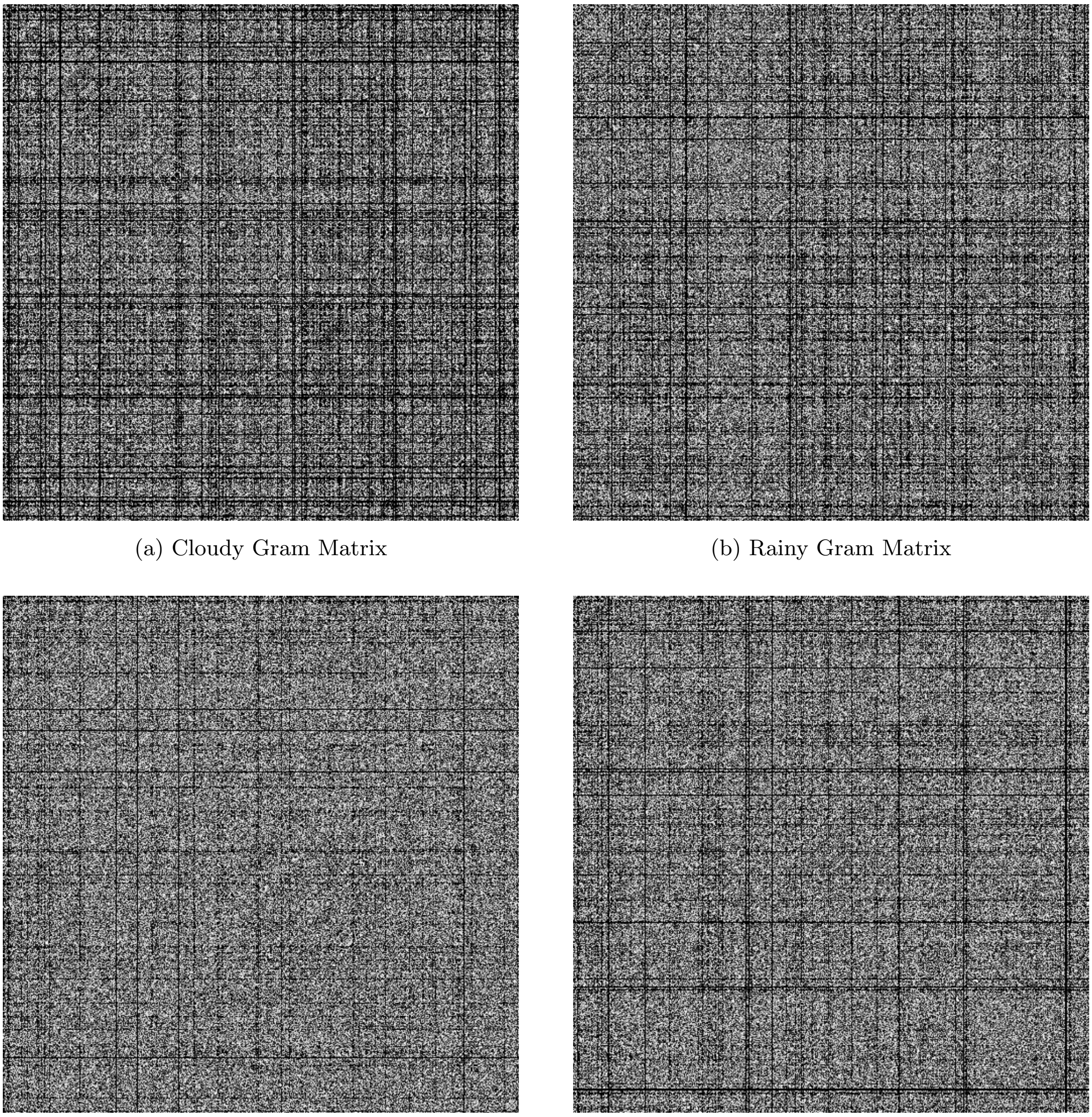
2. Having those feature maps, we can then compute the Gram matrix for each image as $G = F \cdot F^T$

3. From the Gram matrix, we then sampled 1024 features to constitute a feature descriptor for our image.
Note: A single feature descriptor for an image (train/test) is a tensor of shape $(1, 1024)$

4. Having constructed two feature matrices (one for the train images and one for the test images), we then fitted a support vector machine with our train feature matrix.
Note: The train feature matrix is of shape $(920, 1024)$ and the test feature matrix is of shape $(200, 1024)$. From these dimensions alone, it can be seen that our train data consisted of 920 images and our test data consisted of 200 images.

5. Lastly, we tested the SVM on our test data to see how well the algorithm performed.
Note: We used an SVM from the scikit-learn library that implemented a 'One versus One' strategy

Shown on the following page in Figure 1 is a 2x2 subplot displaying visualizations of gram matrices for input images in each of the four classes. Those classes were cloudy, rain, shine, and sunrise.



(a) Cloudy Gram Matrix

(b) Rainy Gram Matrix

(c) Shine Gram Matrix

(d) Sunrise Gram Matrix

Figure 1: Gram Matrices of Images From All Classes

The test accuracy of the VGG19 based style classifier was **95.5%** as reflected below in the confusion matrix shown in Figure 2. Class 0, 1, 2, 3 and 4 correspond to cloudy, rain, shine, and sunrise respectively.

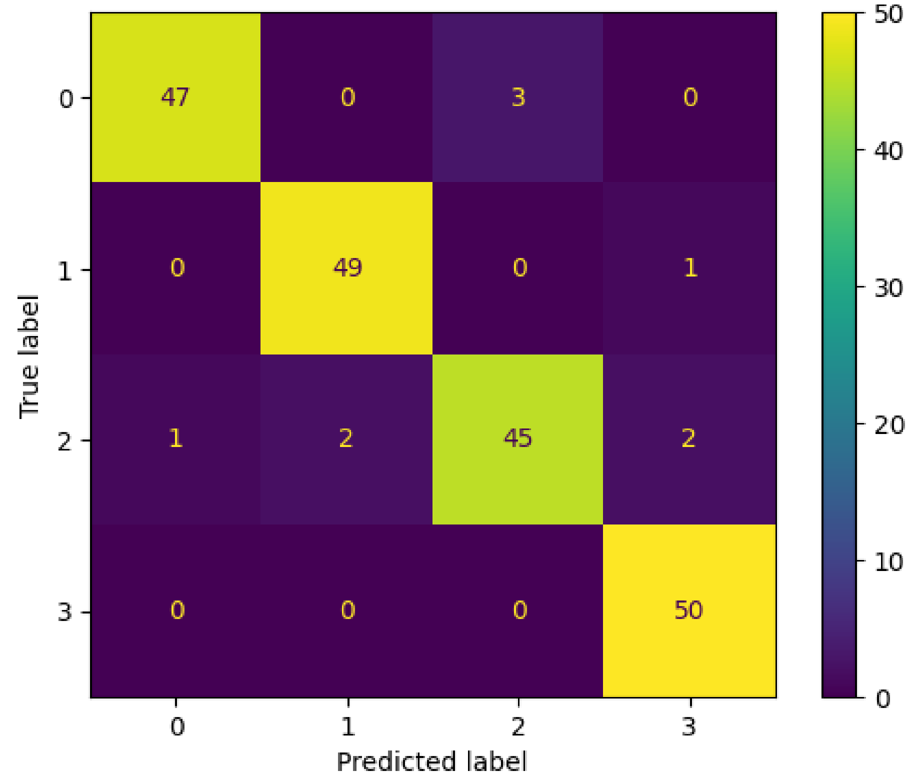


Figure 2: VGG19 Style Classifier Confusion Matrix

3 LBP Based Style Classifier

This section describes the steps followed in order to perform style classification with the help of the local binary pattern algorithm. Those steps are as follows:

1. For every pixel in the grey-scale image, consider a unit circle with 8 points centered at the pixel in question.
2. For these 8 points determine their grey-scale values. The four points directly up, down, left, and right of the center assume the neighboring pixel's grey-scale values. The other four points assume grey-scale values from the following bi-linear interpolation function: $(1-\Delta k)(1-\Delta l)A+(1-\Delta k)\Delta lB+\Delta k(1-\Delta l)C+\Delta k\Delta lD$ where A, B, C, and D are the grey-scale values of the 4 neighboring pixels.
3. Having calculated the grey-scale values for all 8 points, threshold these values against the value of the center of the unit circle. More specifically, assign 1 to a point if its grey-scale value is greater than that of the center. Otherwise assign 0 to it.
4. The orientation of 1s and 0s that yields the smallest value thus forms the binary pattern with respect to that particular pixel in question.

5. The binary pattern is then encoded and a histogram of these encoded binary patterns is computed to form the feature descriptor of that image.
6. Similar to the VGG19 based style classifier, we fitted an SVM with our train feature matrix to classify our test feature matrix.
   Note: Our train and test feature matrix had shapes $(920, 10)$ and $(200, 10)$ respectively. Our SVM had the same parameters as the one used in section 2.

Listed below are LBP histogram feature vectors from images of all four classes:

Cloudy Histogram: {0: 146, 1: 205, 2: 270, 3: 597, 4: 1113, 5: 707, 6: 342, 7: 212, 8: 190, 9: 314}
Rain Histogram: {0: 444, 1: 454, 2: 252, 3: 287, 4: 324, 5: 348, 6: 242, 7: 438, 8: 473, 9: 834}
Shine Histogram: {0: 517, 1: 411, 2: 211, 3: 283, 4: 318, 5: 410, 6: 290, 7: 402, 8: 454, 9: 800}
Sunrise Histogram: {0: 111, 1: 273, 2: 160, 3: 566, 4: 1184, 5: 818, 6: 248, 7: 254, 8: 224, 9: 258}

The test accuracy of the LBP based style classifier was **73%** as reflected below in the confusion matrix shown in Figure 3. Class 0, 1, 2, 3 and 4 correspond to cloudy, rain, shine, and sunrise respectively.

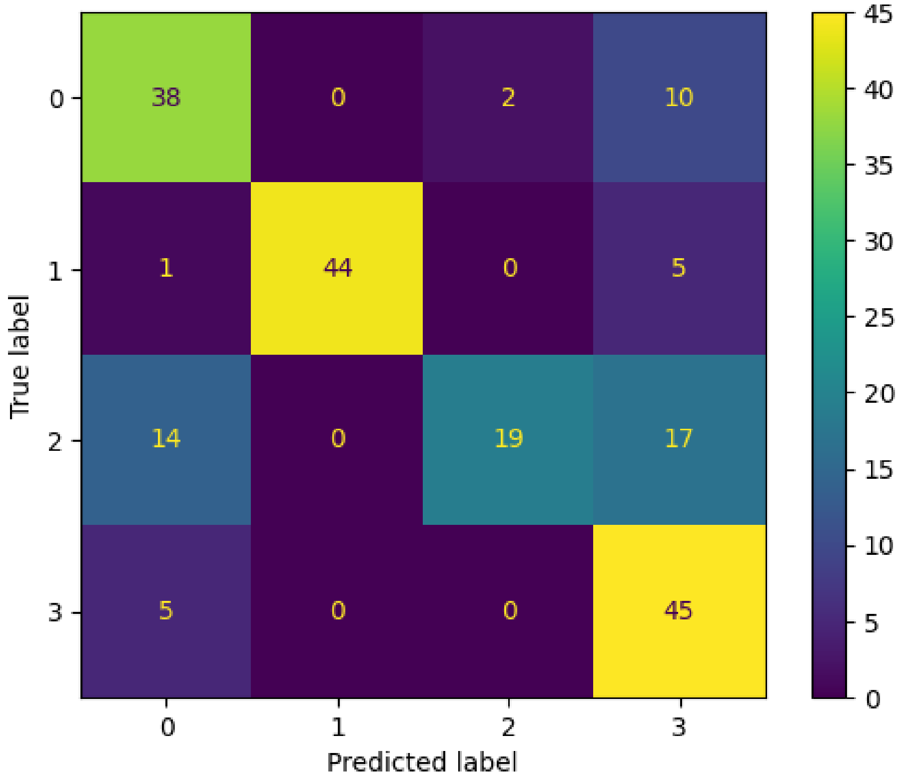


Figure 3: LBP Style Classifier Confusion Matrix

4 Adaptive Instance Normalization (AdaIN)

Another way of extracting style from convolutional features is through the channel normalization parameters i.e. the per-channel means and variances. More specifically, by aligning the channel normalization parameters of a content image with those of a style image, style transfer can be achieved. Subsequently, the steps to perform this task are detailed below.

- Given a feature map F^l of shape (N_l, M_l), the channel normalization parameters can be written as the following per channel mean and variance values:

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

  where $x_{i,k}^l$ denotes the feature value at channel i location k of the feature map F^l

- The concatenations of the above per-channel mean and variance values constitute the feature descriptor for one image. i.e.

$$v_{norm} = (\mu_0, \sigma_0, \mu_1, \sigma_1, \ldots, \mu_{N_l}, \sigma_{N_l}) \in \mathbb{R}^{2N_l}$$

The test accuracy of the AdaIN based style classifier was **97.5%** as reflected below in the confusion matrix shown in Figure 4. Class 0, 1, 2, 3 and 4 correspond to cloudy, rain, shine, and sunrise respectively.

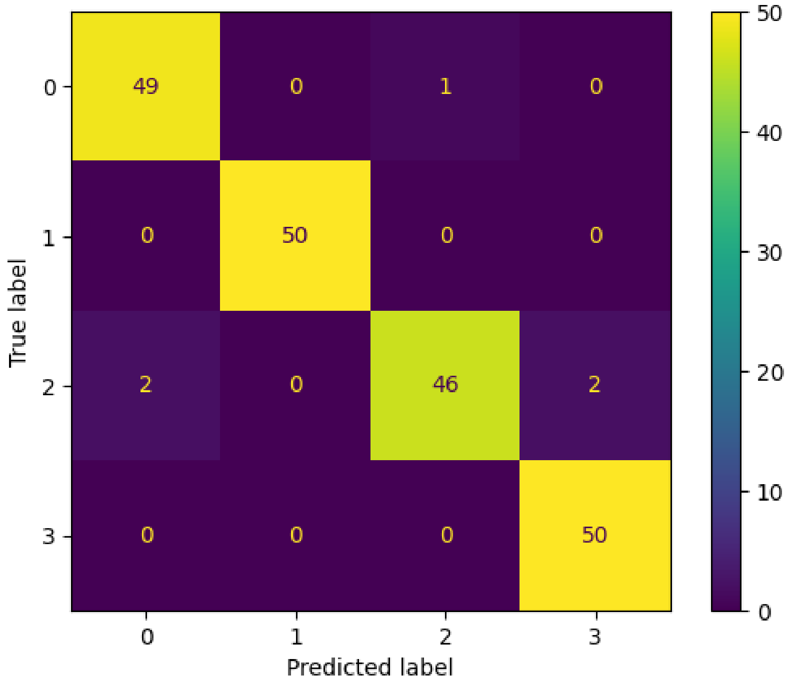


Figure 4: AdaIN Style Classifier Confusion Matrix

5 Discussion of Results

Overall, the three implemented algorithms yielded significantly "better than chance" test accuracies. In the scope of the assignment, classification by random choice would yield 25% test accuracies. Considering that the lowest test accuracy was around 70%, we can justify that all algorithms performed at a satisfactory level. In terms of relative accuracy (comparing the three against each other), VGG19 and AdaIN significantly outperformed LBP. The justification for this is that both VGG19 and AdaIN leverage a pretrained CNN that is better able to extract meaningful features than the purely statistical LBP method.

6 Code Listings

6.1 Helper Functions

```python
import numpy as np
import torch
import torch.nn as nn
from BitVector import *
import os
import cv2

'''VGG19(nn.Module)
Input: (256 x 256) image tensor
Output: (512 x 16 x 16) image tensor
Purpose: given an image tensor, extract feature map'''
class VGG19(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            # encode 1-1
            nn.Conv2d(3, 3, kernel_size=(1, 1), stride=(1, 1)),
            nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 1-1
            # encode 2-1
            nn.Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 2-1
            # encoder 3-1
            nn.Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),

            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            nn.Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 3-1
            # encoder 4-1
```

```
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 4-1
            # rest of vgg not used
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            nn.ReLU(inplace=True),  # relu 5-1
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            # nn.ReLU(inplace=True),
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            # nn.ReLU(inplace=True),
            # nn.Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode='reflect'),
            # nn.ReLU(inplace=True)
        )

    def load_weights(self, path_to_weights):
        vgg_model = torch.load(path_to_weights)
        # Don't care about the extra weights
        self.model.load_state_dict(vgg_model, strict=False)
        for parameter in self.model.parameters():
            parameter.requires_grad = False

    def forward(self, x):
        # Input is numpy array of shape (H, W, 3)
        # Output is numpy array of shape (N_l, H_l, W_l)
        x = torch.from_numpy(x).permute(2, 0, 1).unsqueeze(0).float()
        out = self.model(x)
        out = out.squeeze(0).numpy()
        return out


'''loadImages
Input: file directory
```

```python
Output: 2 lists
Purpose: read all train and test images'''
def loadImages(dir_path):
    img_list = list()
    label_list = list()
    for filename in os.listdir(dir_path):
        filepath = os.path.join(dir_path, filename)
        if os.path.isfile(filepath) and ('.jpg' in filepath or '.jpeg' in filepath):
            img = cv2.imread(filepath, cv2.IMREAD_UNCHANGED)
            resize_img = cv2.resize(img, (256, 256), cv2.INTER_AREA)
            assert(resize_img.shape == (256, 256, 3))
            img_label = -1
            if 'cloudy' in filename:
                img_label = 0
            elif 'rain' in filename:
                img_label = 1
            elif 'shine' in filename:
                img_label = 2
            elif 'sunrise' in filename:
                img_label = 3
            try:
                assert (img_label >= 0)
                img_list.append(resize_img)
                label_list.append(img_label)
            except AssertionError:
                pass
    return img_list, label_list


'''loadImages
Input: file directory
Output: 2 lists
Purpose: read all train and test images'''
def loadGrayImages(dir_path):
    img_list = list()
    label_list = list()
    for filename in os.listdir(dir_path):
        filepath = os.path.join(dir_path, filename)
        if os.path.isfile(filepath) and ('.jpg' in filepath or '.jpeg' in filepath):
            img = cv2.imread(filepath, 0)
            resize_img = cv2.resize(img, (64, 64), cv2.INTER_AREA)
            resize_img = np.pad(resize_img, 1)
            assert(resize_img.shape == (66, 66))
            img_label = -1
            if 'cloudy' in filename:
                img_label = 0
            elif 'rain' in filename:
                img_label = 1
            elif 'shine' in filename:
                img_label = 2
            elif 'sunrise' in filename:
                img_label = 3
            try:
                assert (img_label >= 0)
                img_list.append(resize_img)
                label_list.append(img_label)
            except AssertionError:
                pass
    return img_list, label_list

```

```python

'''lbp_encode
Input: grey scale image
Output: histogram of lbp encodings
Purpose: Given an image, extract the lbp feature descriptor'''
def lbp_encode(img):
    encoded_image = np.zeros((64,64))
    lbp_hist = {t:0 for t in range(10)}
    k = 0.707
    l = 0.707
    img = np.transpose(img, (1,0))
    for x in range(1, img.shape[0] - 1):
        for y in range(1, img.shape[0]-1):
            p0 = img[x,y+1]
            p2 = img[x+1,y]
            p4 = img[x,y-1]
            p6 = img[x-1,y]
            p1 = (1-k) * (1-l) * img[x,y] \
                + (1-k) * l * img[x+1,y] \
                + k * (1-l) * img[x, y+1] \
                + k * l * img[x+1, y+1]
            p3 = (1-k) * (1-l) * img[x,y] \
                + (1-k) * l * img[x,y-1] \
                + k * (1-l) * img[x+1, y] \
                + k * l * img[x+1, y-1]
            p5 = (1-k) * (1-l) * img[x,y] \
                + (1-k) * l * img[x-1,y] \
                + k * (1-l) * img[x, y-1] \
                + k * l * img[x-1, y-1]
            p7 = (1-k) * (1-l) * img[x,y] \
                + (1-k) * l * img[x,y+1] \
                + k * (1-l) * img[x-1, y] \
                + k * l * img[x-1, y+1]

            p0 = 1 if p0 >= img[x,y] else 0
            p1 = 1 if p1 >= img[x,y] else 0
            p2 = 1 if p2 >= img[x,y] else 0
            p3 = 1 if p3 >= img[x,y] else 0
            p4 = 1 if p4 >= img[x,y] else 0
            p5 = 1 if p5 >= img[x,y] else 0
            p6 = 1 if p6 >= img[x,y] else 0
            p7 = 1 if p7 >= img[x,y] else 0

            pattern = [p0, p1, p2, p3, p4, p5, p6, p7]

            bv = BitVector(bitlist=pattern)
            intvals_for_circular_shifts = [int(bv << 1) for _ in range(8)]
            minbv = BitVector(intVal=min(intvals_for_circular_shifts), size=8)

            bvruns = minbv.runs()
            encoding = None
            if len(bvruns) > 2:
                lbp_hist[9] += 1
            elif len(bvruns) == 1 and bvruns[0][0] == '1':
                lbp_hist[8] += 1
            elif len(bvruns) == 1 and bvruns[0][0] == '0':
                lbp_hist[0] += 1
            else:
                lbp_hist[len(bvruns[1])] += 1

```

```python
    return lbp_hist
```

style_classifier_helper.py

6.2 VGG19 Style Classifier Source Code

```python
#!/usr/bin/env python
# coding: utf-8

# # VGG19 Based Style Classifier

# ### Import Statements

# In[1]:


from style_classifier_helper import *
from tqdm import tqdm
import random
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay


# ### Load Training and Testing Image Data
# * train_img_list is a list of all the training images stored as np.ndarry
# * train_label_list is a list of the labels for the training images
# * test_img_list is a list of all the testing images stored as np.ndarry
# * test_label_list is a list of the labels for the test images

# In[2]:


training_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/training"
test_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/testing"

train_img_list, train_label_list = loadImages(training_directory)
test_img_list, test_label_list = loadImages(test_directory)

assert(len(train_img_list) == len(train_label_list))
assert(len(test_img_list) == len(test_label_list))
assert(len(train_img_list) == 920)
assert(len(test_img_list) == 200)


# ### Obtain Feature Maps of all Training Images
# 1. Create an instance of the VGG19 class
# 2. Load the pre-trained weights
# 3. Iterate across both the test and train data
# 4. Extract feature map from the CNN
# 5. Compute the gram matrix for each image and store in the respective list
# 6. Display gram matrix plots for one image in each class

# In[3]:


# Load the model and the provided pretrained weights
vgg = VGG19()
```

```python
vgg.load_weights('/home/jo_wang/Desktop/ECE661/HW07/vgg_normalized.pth')

train_gram_matrix = list()
for i in tqdm(range(len(train_img_list))):
    ft = vgg(train_img_list[i])
    ft = np.resize(ft, (512, 256))
    gram_matrix = ft@ft.T
    train_gram_matrix.append(gram_matrix)

test_gram_matrix = list()
for i in tqdm(range(len(test_img_list))):
    ft = vgg(test_img_list[i])
    ft = np.resize(ft, (512, 256))
    gram_matrix = ft@ft.T
    test_gram_matrix.append(gram_matrix)

cloudy_idx = train_label_list.index(0)
rain_idx = train_label_list.index(1)
shine_idx = train_label_list.index(2)
sunrise_idx = train_label_list.index(3)

cv2.imwrite('cloudy_gram_matrix.jpg', train_gram_matrix[cloudy_idx].astype('uint8'))
cv2.imwrite('rain_gram_matrix.jpg', train_gram_matrix[rain_idx].astype('uint8'))
cv2.imwrite('shine_gram_matrix.jpg', train_gram_matrix[shine_idx].astype('uint8'))
cv2.imwrite('sunrise_gram_matrix.jpg', train_gram_matrix[sunrise_idx].astype('uint8')
    )
assert(len(train_gram_matrix) == len(train_img_list))
assert(len(test_gram_matrix) == len(test_img_list))


# ### Train Support Vector Machine
# 1. For every image in the train and test data set, sample 1024 features randomly
# 2. Build train and test features matrix
#     * train: (920 x 1024)
#     * test: (200 x 1024)
# 3. Fit the SVM model with the train data
# 4. Compute the accuracy on the test data
# 5. Display the confusion matrix

# In[4]:


train_features = np.zeros((1,1024))
test_features = np.zeros((1,1024))

for gram in tqdm(train_gram_matrix):
    random.seed(5283)
    gram_as_list = gram.flatten().tolist()
    sampled_features = random.sample(gram_as_list, 1024)
    sampled_features = np.resize(sampled_features, (1,1024))
    train_features = np.vstack((train_features, sampled_features))

for gram in tqdm(test_gram_matrix):
    random.seed(5283)
    gram_as_list = gram.flatten().tolist()
    sampled_features = random.sample(gram_as_list, 1024)
    sampled_features = np.resize(sampled_features, (1,1024))
    test_features = np.vstack((test_features, sampled_features))

assert(train_features[1:,:].shape == (920, 1024))
```

```
assert(test_features[1:,:].shape == (200, 1024))

clf = svm.SVC(decision_function_shape='ovo')
clf.fit(train_features[1:,:], train_label_list)
texture_predict = clf.predict(test_features[1:,:])
print("Accuracy:",metrics.accuracy_score(test_label_list, texture_predict))
ConfusionMatrixDisplay.from_estimator(clf, test_features[1:,:], test_label_list)
```

vgg19_style_classifier.py

6.3 LBP Style Classifier Source Code

```
#!/usr/bin/env python
# coding: utf-8

# # LBP Based Style Classifer

# ### Import Statements

# In[1]:


import numpy as np

from style_classifier_helper import *
from tqdm import tqdm
import random
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
from BitVector import *


# ### Load Training and Testing Image Data
# * train_img_list is a list of all the training images stored as np.ndarry
# * train_label_list is a list of the labels for the training images
# * test_img_list is a list of all the testing images stored as np.ndarry
# * test_label_list is a list of the labels for the test images

# In[2]:


training_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/training"
test_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/testing"

train_img_list, train_label_list = loadGrayImages(training_directory)
test_img_list, test_label_list = loadGrayImages(test_directory)

assert(len(train_img_list) == len(train_label_list))
assert(len(test_img_list) == len(test_label_list))
assert(len(train_img_list) == 920)
assert(len(test_img_list) == 200)

cloudy_idx = train_label_list.index(0)
rain_idx = train_label_list.index(1)
shine_idx = train_label_list.index(2)
sunrise_idx = train_label_list.index(3)

```

```python
cloudy_histogram = lbp_encode(train_img_list[cloudy_idx])
rainy_histogram = lbp_encode(train_img_list[rain_idx])
shine_histogram = lbp_encode(train_img_list[shine_idx])
sunrise_histogram = lbp_encode(train_img_list[sunrise_idx])

print(f'Cloudy Histogram: {cloudy_histogram}')
print(f'Rain Histogram: {rainy_histogram}')
print(f'Shine Histogram: {shine_histogram}')
print(f'Sunrise Histogram: {sunrise_histogram}')


# ### Train Support Vector Machine
# 1. For every image in the train and test data set, extract the lbp feature
    descriptors
# 2. Build train and test features matrix
#     * train: (920 x 10)
#     * test: (200 x 10)
# 3. Fit the SVM model with the train data
# 4. Compute the accuracy on the test data
# 5. Display the confusion matrix

# In[3]:


train_features = np.zeros((1,10))
test_features = np.zeros((1,10))

for img in tqdm(train_img_list):
    histogram = lbp_encode(img)
    histogram_as_list = list()
    for key, value in histogram.items():
        histogram_as_list.append(value)
    features = np.asarray(histogram_as_list)
    features = np.resize(features, (1,10))
    train_features = np.vstack((train_features, features))

for img in tqdm(test_img_list):
    histogram = lbp_encode(img)
    histogram_as_list = list()
    for key, value in histogram.items():
        histogram_as_list.append(value)
    features = np.asarray(histogram_as_list)
    features = np.resize(features, (1,10))
    test_features = np.vstack((test_features, features))

assert(train_features[1:,:].shape == (920, 10))
assert(test_features[1:,:].shape == (200, 10))


# In[4]:


clf = svm.SVC(decision_function_shape='ovo')
clf.fit(train_features[1:,:], train_label_list)
texture_predict = clf.predict(test_features[1:,:])
print("Accuracy:",metrics.accuracy_score(test_label_list, texture_predict))
ConfusionMatrixDisplay.from_estimator(clf, test_features[1:,:], test_label_list)
```

LBP_style_classifier.py

6.4 AdaIN Style Classifier Source Code

```
#!/usr/bin/env python
# coding: utf-8

# # Adaptive Instance Normalization (adaIN) Style Classifier

# ### Import Statements

# In[1]:


from style_classifier_helper import *
from tqdm import tqdm
import random
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay


# ### Load Training and Testing Image Data
# * train_img_list is a list of all the training images stored as np.ndarry
# * train_label_list is a list of the labels for the training images
# * test_img_list is a list of all the testing images stored as np.ndarry
# * test_label_list is a list of the labels for the test images

# In[2]:


training_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/training"
test_directory = "/home/jo_wang/Desktop/ECE661/HW07/data/testing"

train_img_list, train_label_list = loadImages(training_directory)
test_img_list, test_label_list = loadImages(test_directory)

assert(len(train_img_list) == len(train_label_list))
assert(len(test_img_list) == len(test_label_list))
assert(len(train_img_list) == 920)
assert(len(test_img_list) == 200)


# ### Obtain Feature Maps of all Training Images
# 1. Create an instance of the VGG19 class
# 2. Load the pre-trained weights
# 3. Iterate across both the test and train data
# 4. Extract feature map from the CNN
# 5. Compute the gram matrix for each image and store in the respective list
# 6. Display gram matrix plots for one image in each class

# In[4]:


# Load the model and the provided pretrained weights
vgg = VGG19()
vgg.load_weights('/home/jo_wang/Desktop/ECE661/HW07/vgg_normalized.pth')

train_extracted_feature = list()
for i in tqdm(range(len(train_img_list))):
    ft = vgg(train_img_list[i])
    ft = np.resize(ft, (512, 256))
```

```
    train_extracted_feature.append(ft)

test_extracted_feature = list()
for i in tqdm(range(len(test_img_list))):
    ft = vgg(test_img_list[i])
    ft = np.resize(ft, (512, 256))
    test_extracted_feature.append(ft)

assert(len(train_extracted_feature) == len(train_img_list))
assert(len(test_extracted_feature) == len(test_img_list))


# ### Peform Adaptive Instance Normalization
# 1). Compute the mean and standard deviation of each row in the extracted feature
    map
# 2). Build a (920 x 1024) train and a (200 x 1024) test feature vector
# 3). Fit an SVM model with the train feature vector
# 4). Evaluate the SVM on the test feature vector
# 5). Compute accuracy and display the confusion matrix

# In[45]:


train_features = np.zeros((1,1024))
test_features = np.zeros((1,1024))

for ft in train_extracted_feature:
    mean = np.mean(ft, axis=1)
    std = np.std(ft, axis=1)
    temp = np.hstack((mean, std))
    temp = np.reshape(temp, (1,1024))
    train_features = np.vstack((train_features, temp))

for ft in test_extracted_feature:
    mean = np.mean(ft, axis=1)
    std = np.std(ft, axis=1)
    temp = np.hstack((mean, std))
    temp = np.reshape(temp, (1,1024))
    test_features = np.vstack((test_features, temp))

assert(train_features[1:,:].shape == (920, 1024))
assert(test_features[1:,:].shape == (200, 1024))

clf = svm.SVC(decision_function_shape='ovo')
clf.fit(train_features[1:,:], train_label_list)
texture_predict = clf.predict(test_features[1:,:])
print("Accuracy:",metrics.accuracy_score(test_label_list, texture_predict))
ConfusionMatrixDisplay.from_estimator(clf, test_features[1:,:], test_label_list)
```

AdaIN_style_classifier.py