ECE661 Computer Vision HW6

Akshita Kamsali, akamsali@purdue.edu

November 2, 2022

Note: bold lower case letters indicate vectors and bold upper case letters indicate matrices

1 Theory

1.1 Gray scaLe Co-occurrence Matrix - GLCM

- The basic idea of GLCM is to estimate the joint probability distribution between the grayscale values at any two selected pixels in the image separated specific vector distance d.
- This is a statistical method of detecting texture.
- Joint probability is created by a raster scan and updating a $N \times N$ matrix at $(i, j)^{th}$ position. Where N is the number of grayscale levels in the image and i, j are the grayscale values at two d-separated pixels.
- The $N \times N$ matrix generated in the previous step is divided by total number of pixels to obtain the joint probability.
- we use the statistical measures such as entropy, contrast, homogeneity etc from this joint probability to draw inferences about the texture.
- Entropy in general measures the randomness. Maximum when uniform distribution and minimum (0) when deterministic.
- Irrespective of the size of the image, the size GLCM depends only on the levels of grayscale.

1.2 Local Binary Points - LBP

- LBP is another statistical method for looking for texture in an image.
- Here, we look for pattern around every pixel and threshold it with respect to centre.
- The points around a pixel are chosen as follows:

$$Rcos(\frac{2\pi p}{P}), Rsin(\frac{2\pi p}{P})$$

  , where $p \in \{0, 1, 2, ..., P-1\}$
- R and P can be modified to obtained for different values.
- WE pick R=1 and P=8. Points to left, right, top and bottom can be picked directly. However, points diagonal need to be interpolated.
- We interpolate points as:
  $p_1 = (1-\Delta k)(1-\Delta l)img[x, y]+(1-\Delta k)\Delta limg[x+1, y]+\Delta k(1-\Delta l)img[x, y+1]+\Delta k\Delta limg[x+1, y+1]$
  similarly for p_3, p_5, p_7 then we threshold all points with respect to center to get the binary encoding.

- Once we have an encoding, we obtain a minimum bitvector representation and update out lbp histogram as mentioned. Figure 1 shows these histograms.
- The LBP histogram is used as an input feature vector to out SVM classifier.

1.3 Gabor Filter Family

- On contrast to above two methods, this a structural method. Although, professor mentions that it is a bit of a misnomer.
- They apply a Gaussian decay function which results in a highly localised Fourier transform of the image.
- The localisation is through the weights of the decay funstion and the characteristics of a texture are through periodicity of the kernel. Each kernel corresponding to a specific direction.
- It being part of ISO standard for videos, i.e., MPEG-7, conveys the significance it has.
- The convolutional operator for Gabor filter is given by

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

  we see sinusoids of u and v cycles per second on x and y-axis respectively.
- this kernel is flipped and multiplied with corresponding pixels and summed for a transformation.

1.4 True or False

1.4.1 RGB and HSI are just linear variants of each other.

False.

Reason RGB and HSI are not linear variants of each other. This is because to derive HSI from RGB, we need a complex non-linear relationship, that too is an approximation.

1.4.2 The color space L*a*b* is a nonlinear model of color perception.

True

Reason L stands for luminescence, and a* and b* stand for the two color opponent dimensions. If it were linear, it would have a non-complex linear relationship with a linear color model such as RGB

1.4.3 Measuring the true color of the surface of an object is made difficult by the spectral composition of the illumination.

True.

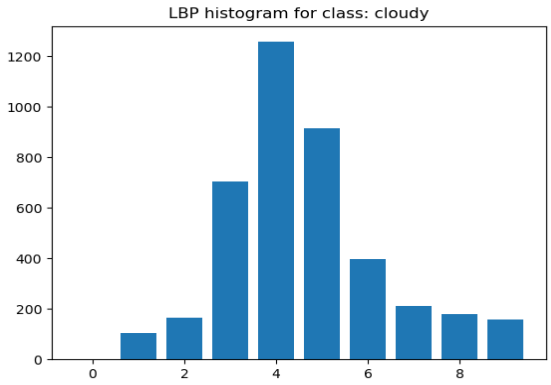
Reason It is because of the diffused light that is reflected off a surface (light we try to capture with camera for the color characterization). This diffused light has a cosine dependence on the angle of illumination to the perpendicular to the surface owing to fundamental laws of reflection and diffraction.

2 Tasks

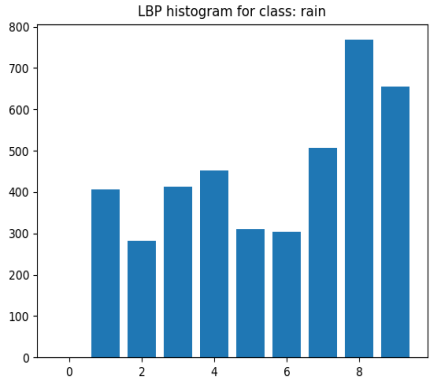
2.1 LBP

My implementation of LBP uses $R = 1$ and $P = 8$, therefore has 8 points around and interpolation equations are as follows:

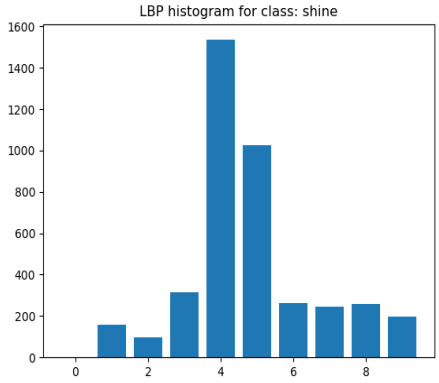
LBP histograms for different classes are in the figure1



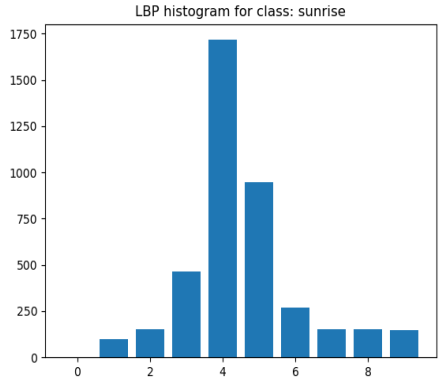
(a) Cloudy image



(b) Rain Image



(c) Shine Image



(d) Sunrise image

Figure 1: A random LBP histogram from each class

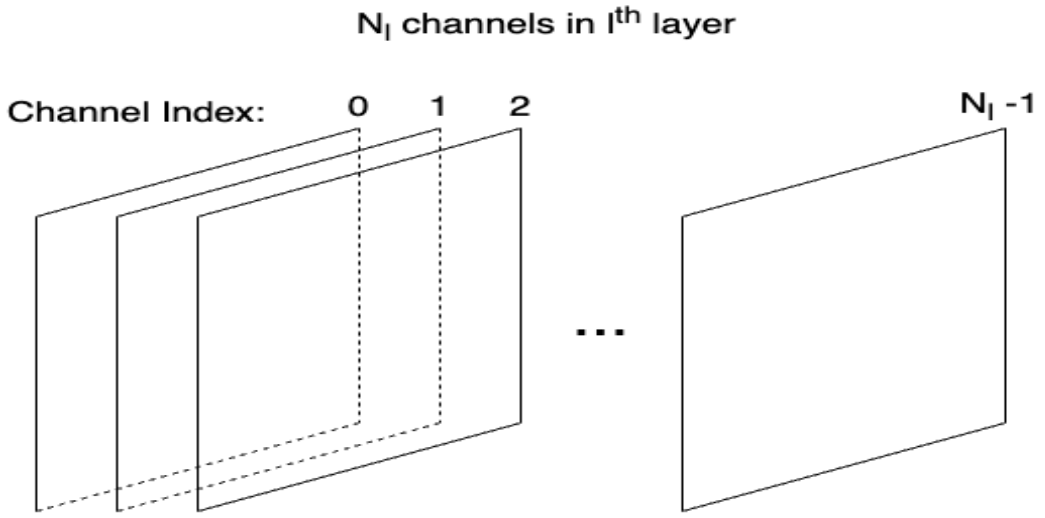


Figure 2: Features in lth layer (ReLU for us) and $N_l = 512$, each slab is of 16×16

2.2 Gram Matrix from VGG

Implementation details:

- I use the given VGG model architecture along with the pretrained weights to extract features for every image.
- Size of feature vector ($\mathbf{F}$) is $C \times H \times W$, where $C = 512$. $H = 16,\ W = 16$, as seen in figure 2, compress this to $C \times HW$, making shape 512x256.
- I compute gram matrix as, $\mathbf{G} = \mathbf{F}\mathbf{F}^{\mathbf{T}}$. shape 512x512. We can see them in figure 3
- I randomly sample 1024 samples from 512x512. Making sure I pick same indices from all images.
- After pruning the gifs and other images from training set, I get 920 images. I have 920x1024 size of train data for my SVM along with labels.
- similarly, for test I get 200x1024 size data.

2.3 SVM Classifier

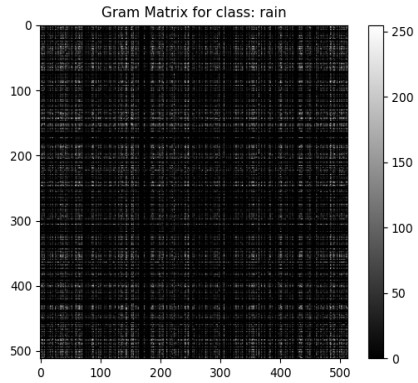
I use sklearn for my SVM. I use `fit` function to train my classifier and `predict` to get prediction labels on test data. I use `score` to get my accuracy. We see the prediction results through confusion matrix in figures 4, 5 and 6.

The accuracy computed through `score` is in table 1

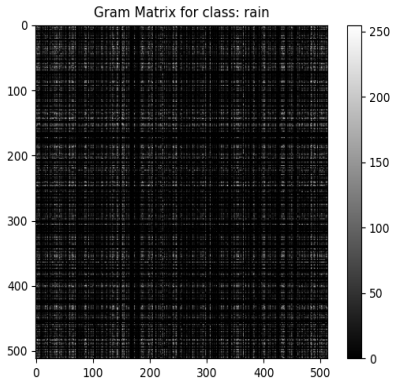
3 Extra credit

For AdaIn, we use channel-wise normalisation parameters, i.e. mean and variance of 16×16 matrix for all 512 channels. This gives us a feature vector of 1024 for every image.

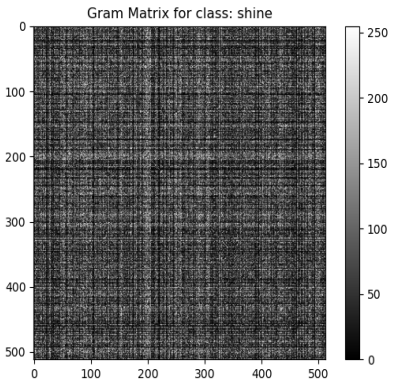
Observation I think this 1024 size feature vector captures the underlying statistics of all channels instead of randomly sampling 1024 elements. This improved the performance from 94% for simple sampling to 97% for normalised parameters.



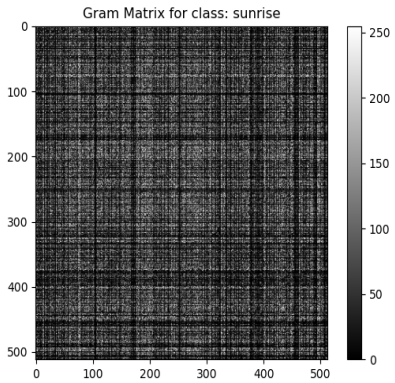
(a) Cloudy image



(b) Rain Image



(c) Shine Image



(d) Sunrise image

Figure 3: Gram Matrix from each class

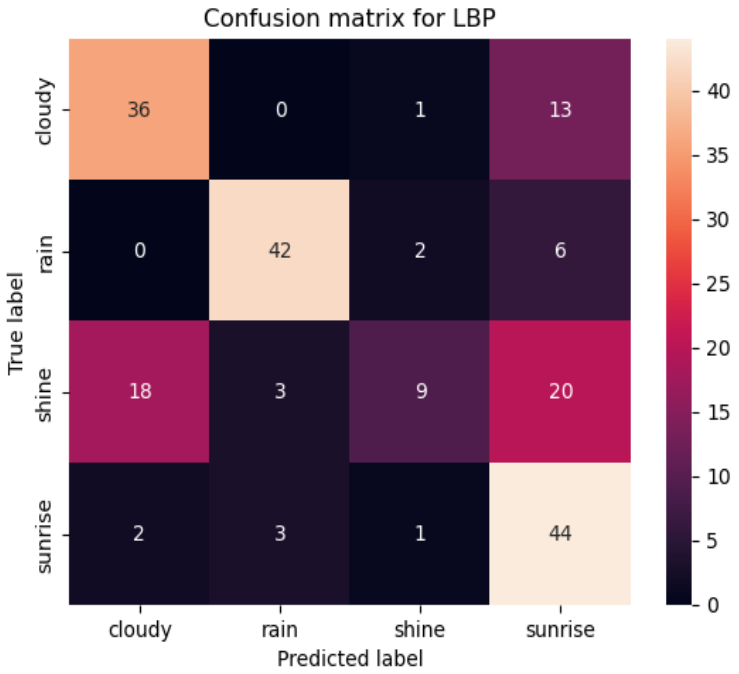


Figure 4: Confusion matrix for LBP

4 Results and observations

I see that all cases, shine is the most confused category. Upon inspecting figures provided in HW pdf, the shine has many similarities to cloud (since it has clouds) and sunrise (as it has sun). I think this could be an explanation for the results we see. LBP performs the worst as it is a pixel level. I think Neural Networks capture more information owing to information being pushed into channels (3 -¿ 512), increasing the capacity of model to learn valuable features which help discriminate between a sun in a shiny day vs sun in sunrise or scattered clouds in a cloudy day vs scattered clouds in a shiny day.

Accuracies are tabulated as follows:

Method	**Accuracy**
LBP	65.5 %
VGG	94 %
AdaIN	97.5 %

Table 1: Accuracy for different methods

The figure 5 has confusion matrix for classifier results on test data for LBP histograms. The figure 5 has confusion matrix for classifier results on test data for sampled elements from GRAM matrix. The figure 6 has confusion matrix for classifier results for channelwise normalisation parameters.

5 Code

```
    from vgg import VGG19

import os
from skimage import io, transform
```

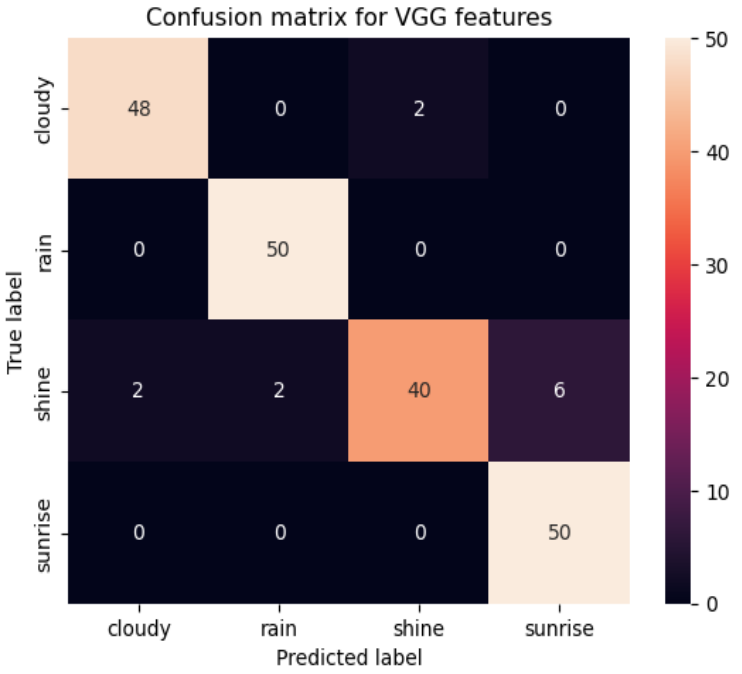


Figure 5: Confusion Matrix for VGG features

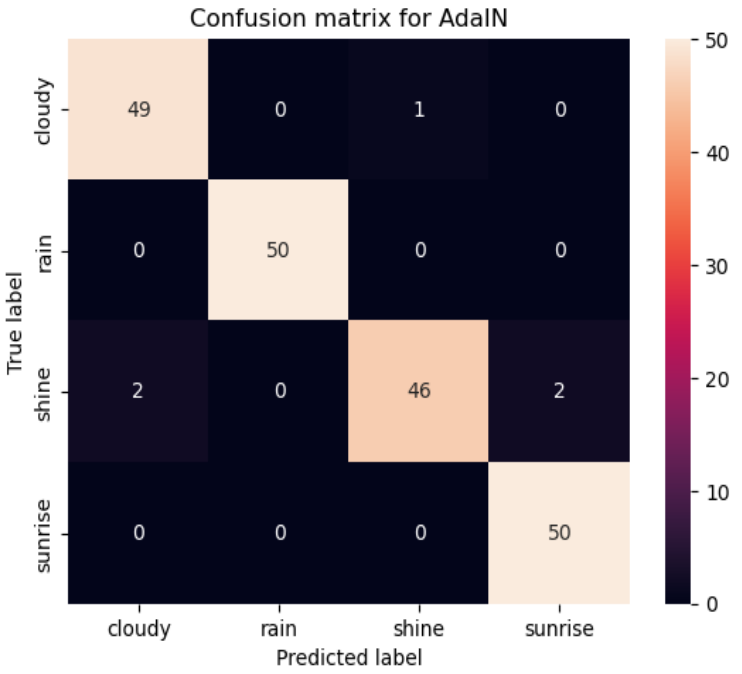


Figure 6: Confusion matrix for normalising parameters

```
from tqdm import tqdm
from BitVector import *
import cv2

import numpy as np


class GetFeatures:
    def __init__(self, dir, model_dir):
        self.model = VGG19()
        self.model.load_weights(model_dir)
        self.dir = dir

    def get_images(self, train=True):
        images = []
        labels = []
        classes = ["cloudy", "rain", "shine", "sunrise"]
        data_dir = self.dir + "/training" if train else self.dir + "/testing"
        for img in sorted(os.listdir(data_dir)):
            label = -1
            if img == ".DS_Store":
                continue
            if "cloudy" in img:
                label = classes.index("cloudy")
            elif "rain" in img:
                label = classes.index("rain")
            elif "shine" in img:
                label = classes.index("shine")
            elif "sunrise" in img:
                label = classes.index("sunrise")

            img = io.imread(os.path.join(data_dir, img))

            if img.shape[-1] == 3:
                labels.append(label)
                images.append(img)

        return labels, images

    def get_features(self, images, labels, token="train"):
        features = []
        for img in tqdm(images):
            img = transform.resize(img, (256, 256))
            feature = self.model(img)
            features.append(feature)
        ## saving features to use later instead of recomputing as it takes a very long time
        np.savez(f"data/{token}_features.npz", labels=labels, features=features)
        # return features

    def lbp(self, images):
        lbp_hists = []
        for img in tqdm(images):
            gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            gray_img = cv2.resize(gray_img, (64, 64), interpolation=cv2.INTER_AREA)
```

```
gray_img_pad = np.pad(gray_img, 1, "constant", constant_values=0)
assert gray_img_pad.shape == (66, 66)
lbp_hist = [0] * 10
# gray_img = gray_img.T
k = 0.707
l = 0.707
for i in range(1, gray_img_pad.shape[0] - 1):
    for j in range(1, gray_img_pad.shape[1] - 1):
        center = gray_img_pad[i][j]
        p = [0] * 8
        if gray_img_pad[i][j + 1] > center:
            p[0] = 1
        if gray_img_pad[i + 1][j] > center:
            p[2] = 1
        if gray_img_pad[i][j - 1] > center:
            p[4] = 1
        if gray_img_pad[i - 1][j] > center:
            p[6] = 1

        p[1] = (
            (1 - k) * (1 - l) * gray_img_pad[i][j]
            + (1 - k) * l * gray_img_pad[i + 1][j]
            + k * (1 - l) * gray_img_pad[i][j + 1]
            + k * l * gray_img_pad[i + 1][j + 1]
        )
        p[3] = (
            (1 - k) * (1 - l) * gray_img_pad[i][j]
            + (1 - k) * l * gray_img_pad[i][j - 1]
            + k * (1 - l) * gray_img_pad[i + 1][j]
            + k * l * gray_img_pad[i + 1][j - 1]
        )
        p[5] = (
            (1 - k) * (1 - l) * gray_img_pad[i][j]
            + (1 - k) * l * gray_img_pad[i - 1][j]
            + k * (1 - l) * gray_img_pad[i][j - 1]
            + k * l * gray_img_pad[i - 1][j - 1]
        )
        p[7] = (
            (1 - k) * (1 - l) * gray_img_pad[i][j]
            + (1 - k) * l * gray_img_pad[i][j + 1]
            + k * (1 - l) * gray_img_pad[i - 1][j]
            + k * l * gray_img_pad[i - 1][j + 1]
        )

        if p[1] > center:
            p[1] = 1
        else:
            p[1] = 0
        if p[3] > center:
            p[3] = 1
        else:
            p[3] = 0
        if p[5] > center:
            p[5] = 1
```

```
                    else:
                        p[5] = 0
                    if p[7] > center:
                        p[7] = 1
                    else:
                        p[7] = 0
                    # Taken from Dr. Kak's tutorial
                    bv = BitVector(bitlist=p)
                    min_val = min([int(bv << 1) for _ in p])
                    min_bv = BitVector(intVal=min_val, size=len(p))

                    bv_runs = min_bv.runs()

                    if len(bv_runs) == 1 and bv_runs[0][0] == 0:
                        lbp_hist[0] += 1
                    elif len(bv_runs) == 1 and bv_runs[0][0] == 1:
                        lbp_hist[8] += 1
                    elif len(bv_runs) > 2:
                        lbp_hist[9] += 1
                    else:
                        lbp_hist[len(bv_runs[0])] += 1
            lbp_hists.append(lbp_hist)
        return np.array(lbp_hists)

    def get_normalised_parameters(self, features):
        ft_r = features.reshape(features.shape[0], features.shape[1], -1)
        train_mean = np.mean(ft_r, axis=2)
        train_std = np.std(ft_r, axis=2)
        train_new_ft = list(zip(train_mean, train_std))
        train_new_ft = np.transpose(np.array(train_new_ft), (0, 2, 1))
        return train_new_ft.reshape(features.shape[0], -1)
from sklearn import svm


class SVMClassifier:
    def __init__(self):
        self.clf = svm.SVC(decision_function_shape="ovo")

    def fit(self, features, labels):
        self.clf.fit(features, labels)

    def predict(self, features):
        return self.clf.predict(features)

    def score(self, features, labels):
        return self.clf.score(features, labels)
from feature_extractor import GetFeatures
from SVM_classifier import SVMClassifier

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import random
```

```
import numpy as np

classes = ["cloudy", "rain", "shine", "sunrise"]

gf = GetFeatures(dir="data", model_dir="vgg_normalized.pth")
train_labels, train_images = gf.get_images(train=True)
test_labels, test_images = gf.get_images(train=False)


## extract features through LBP hists
lbp_train = gf.lbp(train_images)
np.savez(f"data/lbp_train_net.npz", labels=train_labels, features=lbp_train)
lbp_test = gf.lbp(test_images)
np.savez(f"data/lbp_test_net.npz", labels=test_labels, features=lbp_test)

## extract feautures through VGG
gf.get_features(train_images, train_labels, token="train")
gf.get_features(test_images, test_labels, token="test")
train_data = np.load("data/train_feature.npz")
test_data = np.load("data/test_feature.npz")
train_features = train_data["features"]
train_labels = train_data["labels"]
test_features = test_data["features"]
test_labels = test_data["labels"]

# extract features - normalisation parameters
normalised_train_ft = gf.get_normalised_parameters(train_features)
normalised_test_ft = gf.get_normalised_parameters(test_features)


def get_gm(features):
    gm_train = []
    for ft in features:
        random.seed(0)
        ft = ft.reshape(512, -1)
        gm = ft @ ft.T
        gm = random.sample(list(gm.flatten()), 1024)
        gm_train.append(gm)
    return gm_train


gm_train = get_gm(features=train_features)
gm_test = get_gm(features=test_features)

clf = SVMClassifier()
clf.fit(gm_train, train_labels)
pred_labels = clf.predict(gm_test)
print("train_sore: ", clf.score(gm_train, train_labels))
print("test_score: ", clf.score(gm_test, test_labels))


sns.heatmap(confusion_matrix(test_labels, pred_labels), annot=True)
plt.ylabel("True label")
plt.xlabel("Predicted label")
```

```
plt.xticks(np.arange(0.5, 4, 1), classes)
plt.yticks(np.arange(0.5, 4, 1), classes)
plt.title("Confusion matrix for VGG features")
plt.savefig("solutions/lbp_cm.png")


clf = SVMClassifier()
clf.fit(normalised_train_ft, train_labels)
pred_labels = clf.predict(normalised_test_ft)
print(f"Train score: {clf.score(normalised_train_ft, train_labels)}")
print(f"Test score: {clf.score(normalised_test_ft, test_labels)}")

sns.heatmap(confusion_matrix(test_labels, pred_labels), annot=True)
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.xticks(np.arange(0.5, 4, 1), classes)
plt.yticks(np.arange(0.5, 4, 1), classes)
plt.title("Confusion matrix for AdaIN")
plt.savefig("solutions/vgg_cm.png")


clf = SVMClassifier()
clf.fit(normalised_train_ft, train_labels)
pred_labels = clf.predict(normalised_test_ft)
print(f"Train score: {clf.score(normalised_train_ft, train_labels)}")
print(f"Test score: {clf.score(normalised_test_ft, test_labels)}")

sns.heatmap(confusion_matrix(test_labels, pred_labels), annot=True)
plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.xticks(np.arange(0.5, 4, 1), classes)
plt.yticks(np.arange(0.5, 4, 1), classes)
plt.title("Confusion matrix for AdaIN")
plt.savefig("solutions/AdaIN_cm.png")


clf = SVMClassifier()
clf.fit(lbp_train, train_labels)
pred_labels = clf.predict(lbp_test)
print("train_sore: ", clf.score(lbp_train, train_labels))
print("test_score: ", clf.score(lbp_test, test_labels))

sns.heatmap(confusion_matrix(test_labels, pred_labels), annot=True)

plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.xticks(np.arange(0.5, 4, 1), classes)
plt.yticks(np.arange(0.5, 4, 1), classes)
plt.title("Confusion matrix for LBP")
plt.show()

for i in range(4):
    a = np.array(lbp_train)[(np.array(train_labels) == i)][0]
    plt.figure()
```

```
    plt.bar(np.arange(0, 10), a)
    plt.title(f"LBP histogram for class: {classes[i]}")
    plt.savefig(f"solutions/{classes[i]}.png")


# plot gram matrices
for i in range(4):
    ft = train_features[train_labels == i][0].reshape(512, -1)
    g = ft @ ft.T
    plt.figure()
    plt.imshow(g.astype("uint8"), cmap="gray")
    plt.colorbar()
    plt.title(f"Gram Matrix for class: {classes[i]}")
    plt.savefig(f"solutions/gm_{classes[i]}.png")
```