BME 646/ ECE695DL: Homework 2

Varun Aggarwal

January 25, 2022

1 Introduction

This project was aimed at utilizing PIL to read images and PyTorch libraries to manipulate them. Overall, this HW had six tasks.

2 Methodology

The six tasks in this assignment were tackled as follows:

Task 1

Two images were captured using smartphone camera. They are shown in Figure 1

Figure 1: Images captured for this homework

Task 2

For this task, the two images were read using "PIL Image" Class. Next, the images were converted into tensor and normalized so the pixel values lie between -1.0 and 1.0 .

Task 3

For this task, a function was used to first generate a histogram was Red-Green-Blue channels. The function was called *histc* which was part of *Torch* Library. Next, the histogram was plotted for each of the individual channel. The output of this task is given in Figure 2

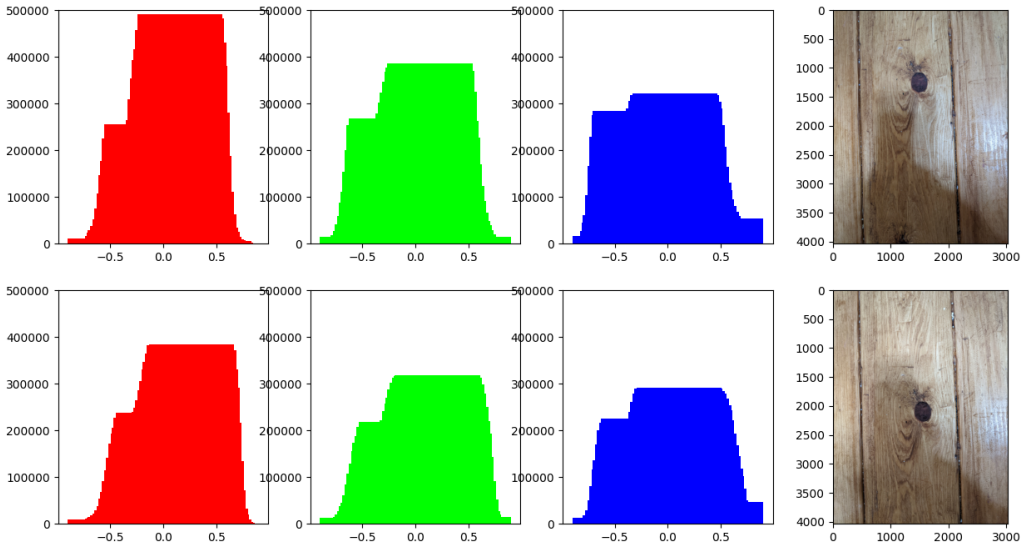

Figure 2: Histogram for Each Image. The color for each histogram corresponds to the color channel of the image i.e. RGB

Task 4

In this task, per channel distance between each image was calculated. The distance matrix used for this task was Wasserstein distance. The reason was choosing this particular matrix was not completely arbitrary. Wasserstein Distance is popularly used to calculate distance between two distributions. And since channel histograms are essentially pixel intensity distribution for each color, the Wasserstein distance was applicable here. The resulting histograms are given in Table 1.

Table 1: Wasserstein Distance between unmodified images

	Red	**Green**	**Blue**
Wasserstein Distance	17667.02	13950.28	12329.28

Task 5

In this task, random affine transformation was applied to each image. Next, the histograms were recomputed again and the distance between individual channels was recalculated. For transformation, the fill value was set to -2.0 . The reason behind the value was so that any pixel which wasn't filled due to transformation, would be ignored in histogram generation (histogram generation was restricted to pixels with value between -1.0 and 1.0). On comparing the histograms, it was observed that compared to original histogram, the histogram for transformed images were scaled down. The result for this task are shown in Figure 3 and Table 2.

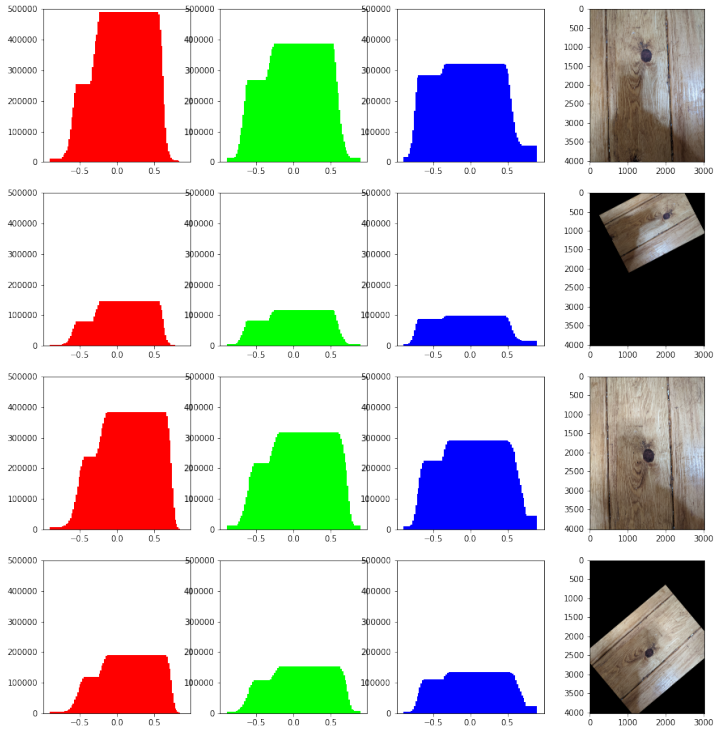

Figure 3: Histogram following affine tranformation

Table 2: Wasserstein Distance after Affine Tranformation

	Red	**Green**	**Blue**
Wasserstein Distance (Orginal)	17667.02	13950.28	12329.28
Wasserstein Distance (Affine)	22883.68	22883.68	22941.72

Task 6

For this task, first a random perspective transform was applied to input images. Next, the histograms were generated for transformed images followed by calculation of Wasserstein Distance. It was observed that histogram of transformed image was not only scaled down but also, the distribution of pixels in each bin changed around i.e. bin with maximum number of pixels was different for original image vs the transformed image. This was in stark contrast to affine transformation where relative pixel distribution remained the same. The result for this task are shown in Figure 4 and Table 3.

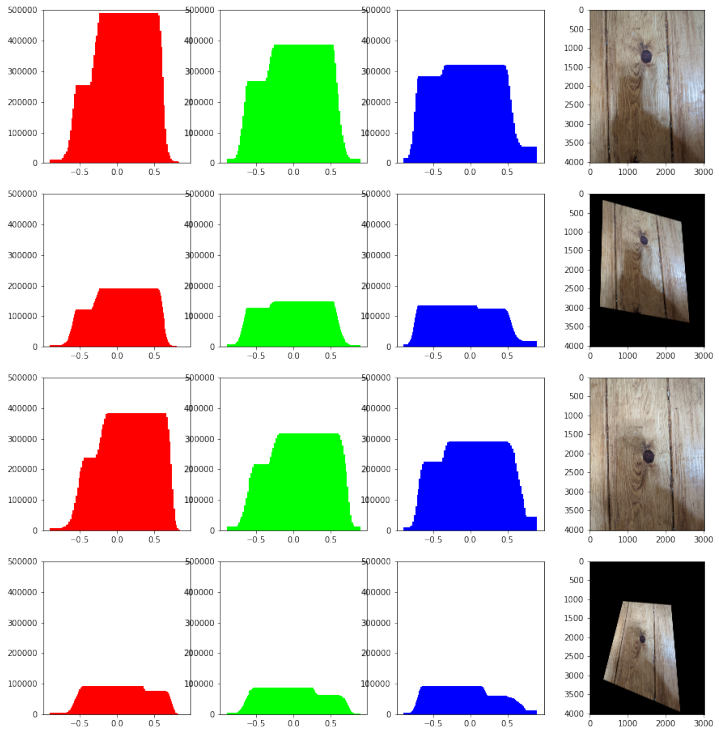

Figure 4: Histogram following perspective tranformation

Table 3: Wasserstein Distance after Perspective Transformation

	Red	**Green**	**Blue**
Wasserstein Distance (Orginal)	17667.02	13950.28	12329.28
Wasserstein Distance (Perspective)	20432.92	19943.31	19294.20

3 Implementation

CODE

```
import torch
import torchvision.transforms as tvt

import os
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import wasserstein_distance


# Setup Data
data_path = "images"
image1_name = "img1.jpg"
image2_name = "img2.jpg"


# load images wiht PIL
img1 = Image.open(os.path.join(data_path, image1_name))
img2 = Image.open(os.path.join(data_path, image2_name))


# setup transform
transform_img = tvt.Compose(
    [
        tvt.PILToTensor(),
        tvt.ConvertImageDtype(torch.float),
        tvt.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ]
)

# create image tensor
```

```
img1_tensor = transform_img(img1)
img2_tensor = transform_img(img2)


# create histogram
def get_hist(tensor_img, bins=100):
    return list(
        map(
            lambda x: torch.histc(tensor_img[x, :, :], bins=bins, min=-1.0,
                ↪ max=1.0),
            [0, 1, 2]
        )
    )


# get histogram from each image
bins = 100
hist_img1 = get_hist(img1_tensor, bins)
hist_img2 = get_hist(img2_tensor, bins)


# plot images and thier respective histograms
plt.figure(figsize=(15, 8))
X_axis = np.linspace(-0.5, 0.5, bins)
colors = [(1.0, 0.0, 0.0, 1.0), (0.0, 1.0, 0.0, 1.0), (0.0, 0.0, 1.0, 1.0)]
for i in range(1, 4):
    plt.subplot(2, 4, i)
    plt.bar(X_axis, hist_img1[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)
    plt.subplot(2, 4, i + 4)
    plt.bar(X_axis, hist_img2[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)

plt.subplot(2, 4, 4)
plt.imshow(img1)

plt.subplot(2, 4, 8)
plt.imshow(img2)

plt.savefig("orginalHistogram.png")
```

```
# apply affine transformation - https://pytorch.org/vision/master/
    ↪ auto_examples/plot_transforms.html#randomaffine
affine_transfomer = tvt.RandomAffine(
    degrees=(30, 70), translate=(0.1, 0.3), scale=(0.5, 0.75), fill=-2.0
)
img1_tensor_affine = affine_transfomer(img1_tensor)
img2_tensor_affine = affine_transfomer(img2_tensor)


# apply perspective transformation - https://pytorch.org/vision/master/
    ↪ auto_examples/plot_transforms.html#randomaffine
perspective_transformer = tvt.RandomPerspective(distortion_scale=0.6, p=1.0,
    ↪  fill=-2.0)
img1_tensor_perspective = perspective_transformer(img1_tensor)
img2_tensor_perspective = perspective_transformer(img2_tensor)


# plot images after affine and their respective histograms
plt.figure(figsize=(15, 16))
X_axis = np.linspace(-0.5, 0.5, bins)
colors = [(1.0, 0.0, 0.0, 1.0), (0.0, 1.0, 0.0, 1.0), (0.0, 0.0, 1.0, 1.0)]
for i in range(1, 4):
    plt.subplot(4, 4, i)
    plt.bar(X_axis, hist_img1[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 4)
    plt.bar(X_axis, get_hist(img1_tensor_affine)[i - 1], color=colors[i -
        ↪ 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 8)
    plt.bar(X_axis, hist_img2[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 12)
    plt.bar(X_axis, get_hist(img2_tensor_affine)[i - 1], color=colors[i -
        ↪ 1])
    plt.ylim(0, 5e5)

plt.subplot(4, 4, 4)
plt.imshow(img1)
plt.subplot(4, 4, 8)
plt.imshow((img1_tensor_affine.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(4, 4, 12)
```

```
plt.imshow(img2)
plt.subplot(4, 4, 16)
plt.imshow((img2_tensor_affine.permute(1, 2, 0) + 1.0) / 2.0)

plt.savefig("affineHistogram.png")


# plot images after perspective and their respective histograms
plt.figure(figsize=(15, 16))
X_axis = np.linspace(-0.5, 0.5, bins)
colors = [(1.0, 0.0, 0.0, 1.0), (0.0, 1.0, 0.0, 1.0), (0.0, 0.0, 1.0, 1.0)]
for i in range(1, 4):
    plt.subplot(4, 4, i)
    plt.bar(X_axis, hist_img1[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 4)
    plt.bar(X_axis, get_hist(img1_tensor_perspective)[i - 1], color=colors[i
        ↪  - 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 8)
    plt.bar(X_axis, hist_img2[i - 1], color=colors[i - 1])
    plt.ylim(0, 5e5)
    plt.subplot(4, 4, i + 12)
    plt.bar(X_axis, get_hist(img2_tensor_perspective)[i - 1], color=colors[i
        ↪  - 1])
    plt.ylim(0, 5e5)

plt.subplot(4, 4, 4)
plt.imshow(img1)
plt.subplot(4, 4, 8)
plt.imshow((img1_tensor_perspective.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(4, 4, 12)
plt.imshow(img2)
plt.subplot(4, 4, 16)
plt.imshow((img2_tensor_perspective.permute(1, 2, 0) + 1.0) / 2.0)

plt.savefig("perspectiveHistogram.png")


# Display Image with scaling
plt.subplot(2, 3, 1)
plt.imshow((img1_tensor.permute(1, 2, 0) + 1.0) / 2.0)
```

```
plt.subplot(2, 3, 2)
plt.imshow((img1_tensor_affine.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(2, 3, 3)
plt.imshow((img1_tensor_perspective.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(2, 3, 4)
plt.imshow((img2_tensor.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(2, 3, 5)
plt.imshow((img2_tensor_affine.permute(1, 2, 0) + 1.0) / 2.0)
plt.subplot(2, 3, 6)
plt.imshow((img2_tensor_perspective.permute(1, 2, 0) + 1.0) / 2.0)

plt.savefig("allTransforms.png")


# calculate distances between histograms
def get_dist(hist1, hist2):
    return list(map(lambda x: wasserstein_distance(hist1[x], hist2[x]), [0,
        ↪ 1, 2]))


dist_without_transform = get_dist(hist_img1, hist_img2)
dist_with_affine = get_dist(get_hist(img1_tensor_affine), get_hist(
    ↪ img2_tensor_affine))
dist_with_perspective = get_dist(
    get_hist(img1_tensor_perspective), get_hist(img2_tensor_perspective)
)

print(dist_without_transform)
print(dist_with_affine)
print(dist_with_perspective)
```

4 Lessons Learned

Once again, this programming assignment was straightforward and covered the basics of image transformation. beign familiar with PyTorch made it was to manipulate the images in this assignment. I did not find any issues with the implementation. Although, certain things were unclear and I took my best shot to address them.

- It was unclear whether the images had to be part of same tensor or two separate tensor
- It was unclear what distance had to be used for comparing Histograms

- It was unclear what affine and perspective transformation had to be applied to the images

5 Suggested Enhancements

The assignment could direct student to explore systematic group transformation to observe changes in histogram rather than randomly performing a transformation.