BME 646/ ECE695DL: Homework 2

Bianjiang Yang

23 January 2022

1 Introduction

In this homework2. I will open my image files as PIL objects and subsequently process the images with a combination of Torch/Scipy/Torchvision/PIL/Matplotlib code. Overall, I learned how to use the PIL Image class for opening image files and for accessing their pixels. I also learned the differences between the PIL and tensor based representations of an image. During the pre-processing of the images, I was familiar with the pixel data scaling and normalization functionality in Torchvision.

2 Methodology

Packages: torch, pillow, torchvision, scipy.stats, matplotlib.pyplot
Functions used:
(1) Calculate histogram: torch.histc
(2) Display the several images separately but in a single composite display: matplotlib.pyplot.subplot
(3) Calculate Wasserstein Distance: scipy.stats.wasserstein_distance
Language: Python3
Images shot by: Oneplus7
Original Image resolution: 3000×3000
Cropped Image resolution: 1000×1000

3 Implementation and Results

```
import torch, torchvision, PIL
import matplotlib.pyplot as plt
from scipy.stats import wasserstein_distance
```

```
def main():

    ##Task 2
    img1 = PIL.Image.open("1.jpg")
    img2 = PIL.Image.open("2.jpg") #3000*3000
    transform2 = torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
    ]
    )
    img1 = transform2(img1)
    img2 = transform2(img2)
    img1 = img1[:, 1000:2000, 1000:2000]
    img2 = img2[:, 1000:2000, 1000:2000]
    print(img1.shape)

    #Calculating Histogram
    histogram1_0 = torch.histc(img1[0, :, :], bins=256, min = 0.0, max = 1.0)
    histogram1_1 = torch.histc(img1[1, :, :], bins=256, min = 0.0, max = 1.0)
    histogram1_2 = torch.histc(img1[2, :, :], bins=256, min = 0.0, max = 1.0)
    histogram2_0 = torch.histc(img2[0, :, :], bins=256, min = 0.0, max = 1.0)
    histogram2_1 = torch.histc(img2[1, :, :], bins=256, min = 0.0, max = 1.0)
    histogram2_2 = torch.histc(img2[2, :, :], bins=256, min = 0.0, max = 1.0)
    #Histogram Normalization
    histogram1_0 = histogram1_0.div(histogram1_0.sum())
    histogram1_1 = histogram1_1.div(histogram1_1.sum())
    histogram1_2 = histogram1_2.div(histogram1_2.sum())
    histogram2_0 = histogram2_0.div(histogram2_0.sum())
    histogram2_1 = histogram2_1.div(histogram2_1.sum())
    histogram2_2 = histogram2_2.div(histogram2_2.sum())

    ##Task 3
    ####################################
    # plot 1:
    f, axarr = plt.subplots(2, 2)
    axarr[0, 0].plot(histogram1_0)
    axarr[0, 1].plot(histogram1_1)
    axarr[1, 0].plot(histogram1_2)
    axarr[1, 1].imshow(img1.permute(2, 1, 0)*0.5+0.5)

    plt.suptitle("Histogram1")
    plt.savefig("Histogram1")
    ####################################
    # plot 2:
    f, axarr = plt.subplots(2, 2)
    axarr[0, 0].plot(histogram2_0)
```

```
axarr[0, 1].plot(histogram2_1)
axarr[1, 0].plot(histogram2_2)
axarr[1, 1].imshow(img2.permute(2, 1, 0)*0.5+0.5)

plt.suptitle("Histogram2")
plt.savefig("Histogram2")
####################################

##Task 4
distance_R = wasserstein_distance(histogram1_0, histogram2_0)
distance_G = wasserstein_distance(histogram1_1, histogram2_1)
distance_B = wasserstein_distance(histogram1_2, histogram2_2)
print("Wasserstein Distance R Channel Histogram: ", distance_R)
print("Wasserstein Distance G Channel Histogram: ", distance_G)
print("Wasserstein Distance B Channel Histogram: ", distance_B)

##Task 5
affine = torchvision.transforms.RandomAffine(degrees = 45)
affine_img1 = affine(img1*0.5+0.5)
    # torchvision.transforms.functional.affine(
    # img = img1, angle = -45, translate = (500, 500), scale = 1, shear = 0, fill = -1.0
histogram1_0_x = torch.histc(affine_img1[0, :, :], bins=256, min = 0.0, max = 1.0)
histogram1_1_x = torch.histc(affine_img1[1, :, :], bins=256, min = 0.0, max = 1.0)
histogram1_2_x = torch.histc(affine_img1[2, :, :], bins=256, min = 0.0, max = 1.0)
histogram1_0_x = histogram1_0_x.div(histogram1_0_x.sum())
histogram1_1_x = histogram1_1_x.div(histogram1_1_x.sum())
histogram1_2_x = histogram1_2_x.div(histogram1_2_x.sum())
# print("affine: ", histogram1_0_x)
histogram1_0_x = histogram1_0_x[1:] * (affine_img1.shape[-1] *
                affine_img1.shape[-2] / (affine_img1.shape[-1] *
                affine_img1.shape[-2] - histogram1_0_x[0]))
histogram1_1_x = histogram1_1_x[1:] * (affine_img1.shape[-1] *
                affine_img1.shape[-2] / (affine_img1.shape[-1] *
                affine_img1.shape[-2] - histogram1_1_x[0]))
histogram1_2_x = histogram1_2_x[1:] * (affine_img1.shape[-1] *
                affine_img1.shape[-2] / (affine_img1.shape[-1] *
                affine_img1.shape[-2] - histogram1_2_x[0]))
####################################
# plot 1:
# print("affine: ", histogram1_0_x)
# print("original: ", histogram1_0)
f, axarr = plt.subplots(2, 2)
axarr[0, 0].plot(histogram1_0_x)
axarr[0, 1].plot(histogram1_1_x)
axarr[1, 0].plot(histogram1_2_x)
axarr[1, 1].imshow(affine_img1.permute(2, 1, 0))
```

```
plt.suptitle("Histogram3")
plt.savefig("Histogram3")
###################################
# plot 2:
f, axarr = plt.subplots(2, 2)
axarr[0, 0].plot(histogram2_0[1:])
axarr[0, 1].plot(histogram2_1[1:])
axarr[1, 0].plot(histogram2_2[1:])
axarr[1, 1].imshow(img2.permute(2, 1, 0)*0.5+0.5)

plt.suptitle("Histogram4")
plt.savefig("Histogram4")
###################################
distance_R_x = wasserstein_distance(histogram1_0_x, histogram2_0[1:])
distance_G_x = wasserstein_distance(histogram1_1_x, histogram2_1[1:])
distance_B_x = wasserstein_distance(histogram1_2_x, histogram2_2[1:])
print("After Affine Transformation, Wasserstein Distance R Channel Histogram: ",
      distance_R_x)
print("After Affine Transformation, Wasserstein Distance G Channel Histogram: ",
      distance_G_x)
print("After Affine Transformation, Wasserstein Distance B Channel Histogram: ",
      distance_B_x)

##Task 6
transformer_per = torchvision.transforms.RandomPerspective(
    distortion_scale=0.6, p=1.0)
perspective_img1 = transformer_per(img1*0.5+0.5)
histogram1_0_x = torch.histc(perspective_img1[0, :, :], bins=256, min=0.0, max=1.0)
histogram1_1_x = torch.histc(perspective_img1[1, :, :], bins=256, min=0.0, max=1.0)
histogram1_2_x = torch.histc(perspective_img1[2, :, :], bins=256, min=0.0, max=1.0)
histogram1_0_x = histogram1_0_x.div(histogram1_0_x.sum())
histogram1_1_x = histogram1_1_x.div(histogram1_1_x.sum())
histogram1_2_x = histogram1_2_x.div(histogram1_2_x.sum())
###################################
histogram1_0_x = histogram1_0_x[1:] * (perspective_img1.shape[-1]*
                 perspective_img1.shape[-2]/(perspective_img1.shape[-1]*
                  perspective_img1.shape[-2]-histogram1_0_x[0]))
histogram1_1_x = histogram1_1_x[1:] * (perspective_img1.shape[-1]*
                perspective_img1.shape[-2]/(perspective_img1.shape[-1]*
                perspective_img1.shape[-2]-histogram1_1_x[0]))
histogram1_2_x = histogram1_2_x[1:] * (perspective_img1.shape[-1]*
                perspective_img1.shape[-2]/(perspective_img1.shape[-1]*
                perspective_img1.shape[-2]-histogram1_2_x[0]))
# plot 1:
f, axarr = plt.subplots(2, 2)
```

```
    axarr[0, 0].plot(histogram1_0_x)
    axarr[0, 1].plot(histogram1_1_x)
    axarr[1, 0].plot(histogram1_2_x)
    axarr[1, 1].imshow(perspective_img1.permute(2, 1, 0))

    plt.suptitle("Histogram5")
    plt.savefig("Histogram5")
    ###################################
    # plot 2:
    f, axarr = plt.subplots(2, 2)
    axarr[0, 0].plot(histogram2_0[1:])
    axarr[0, 1].plot(histogram2_1[1:])
    axarr[1, 0].plot(histogram2_2[1:])
    axarr[1, 1].imshow(img2.permute(2, 1, 0)*0.5+0.5)

    plt.suptitle("Histogram6")
    plt.savefig("Histogram6")
    ###################################
    distance_R_x = wasserstein_distance(histogram1_0_x, histogram2_0[1:])
    distance_G_x = wasserstein_distance(histogram1_1_x, histogram2_1[1:])
    distance_B_x = wasserstein_distance(histogram1_2_x, histogram2_2[1:])
    print("After perspective Transformation, Wasserstein Distance R Channel Histogram: ",
          distance_R_x)
    print("After perspective Transformation, Wasserstein Distance G Channel Histogram: ",
          distance_G_x)
    print("After perspective Transformation, Wasserstein Distance B Channel Histogram: ",
          distance_B_x)


if __name__ == "__main__":
    main()
```

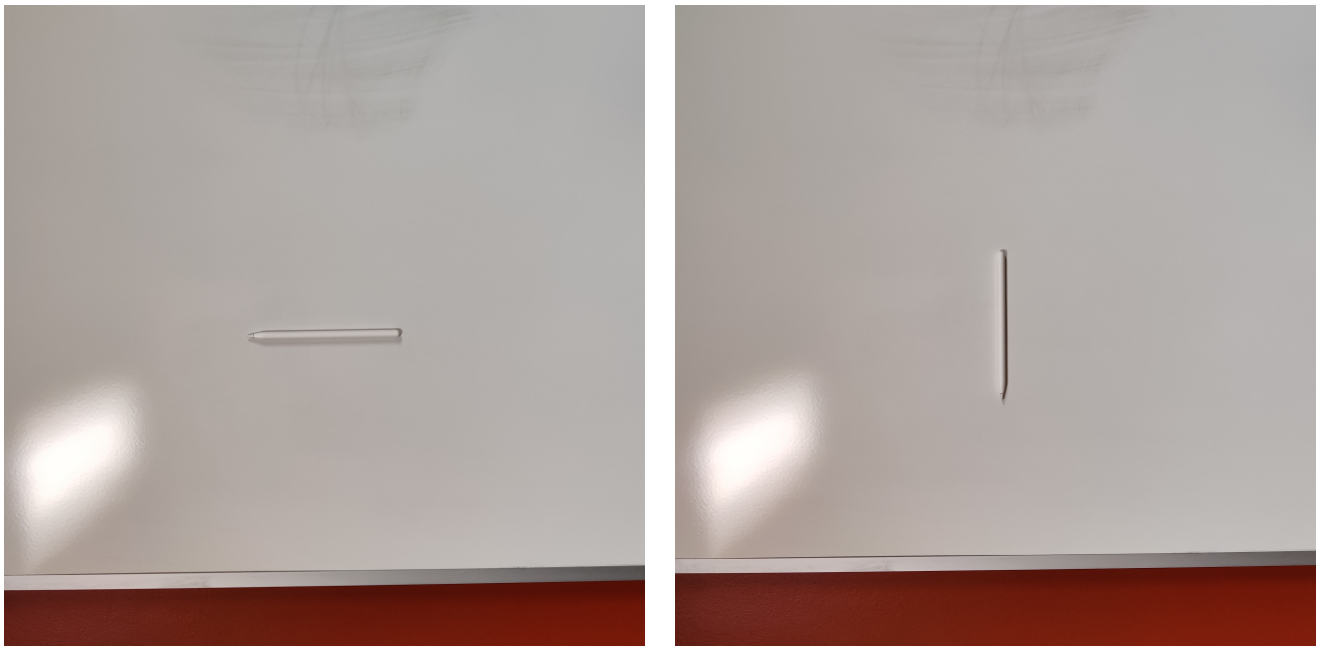

Figure 1: Original Images before cropping.

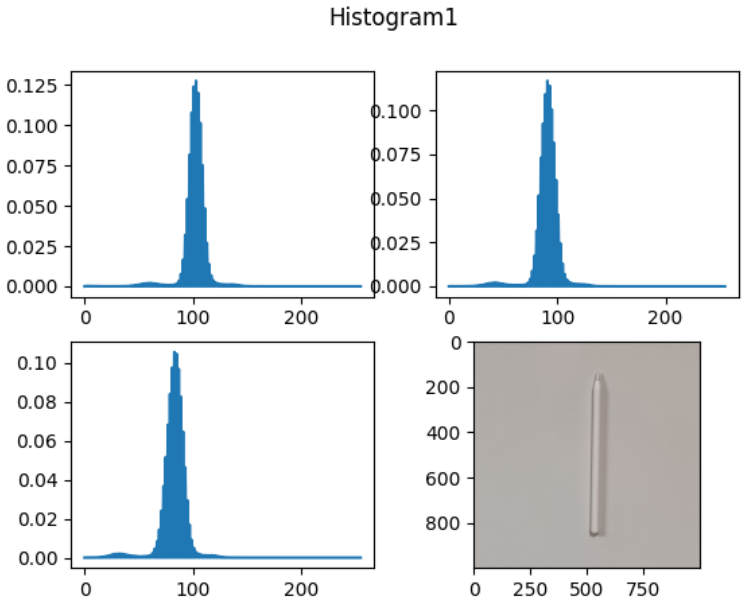

Figure 2: Histogram1 & Image 1

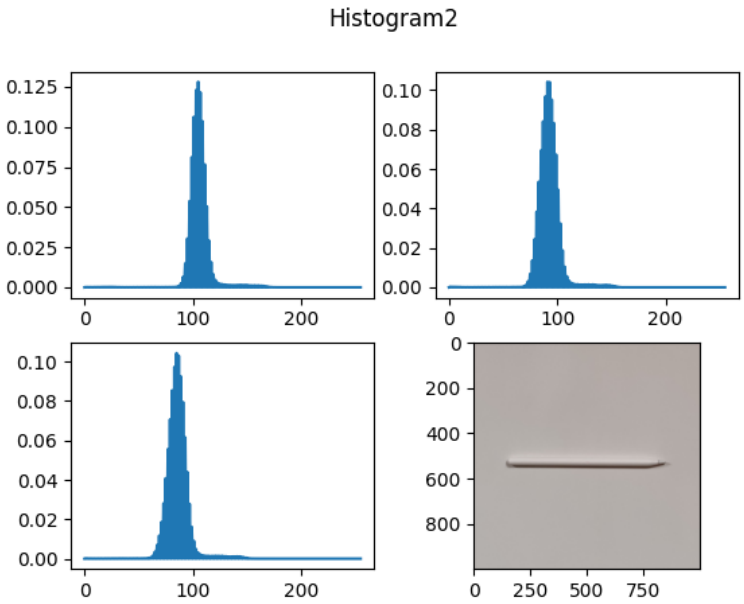

Figure 3: Histogram2 & Image 2

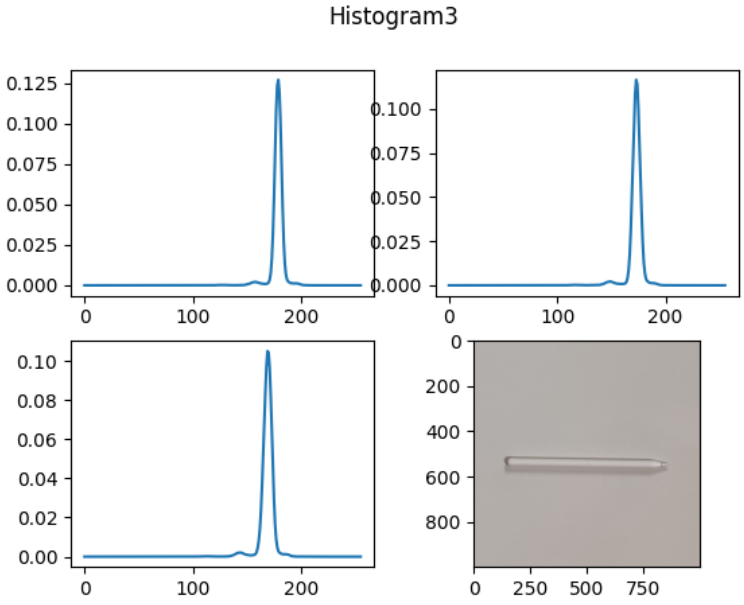

Figure 4: Histogram3 & Image 3

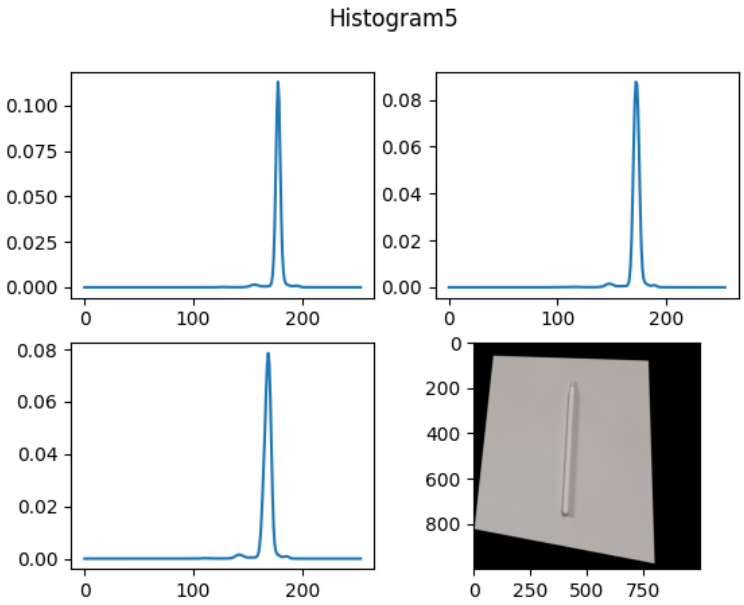

Figure 5: Histogram5 & Image 5

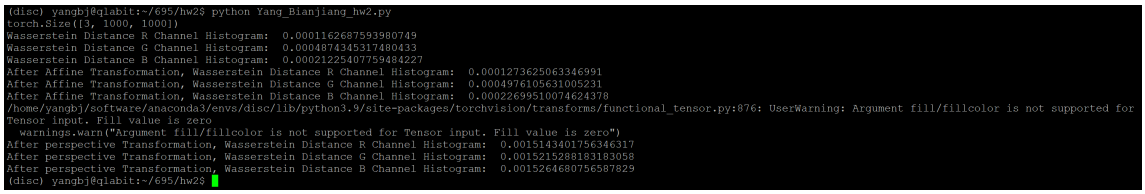

Figure 6: Printed Results

The results above shows that:
(1)After affine transformation(e.g. rotation with -90), the images typically will preserver all information, which will result in the distribution of the histogram for each channel being remained. Then the Wasserstein Distance will be very similar.
(2)After the random perspective transformation, the images will be distorted, which will result in the proportion of each pixel value being changed, then the distribution of the histogram for each channel will be changed. So the Wasserstein Distance will be different.

4 Lessons Learned

The hurdles faced and the techniques employed to overcome them:
(1) When using the torch.hisc function, setting the min and max is necessary;
(2) After Affine and Perspective transformation, the images saved are filled with black background (pixel value=0). To calculate the new histogram, I tried to first scale the image to [0,1] then calculate the histogram, then replace the histogram with: $histogram[1:] \times img\_width \times img\_height/(img\_width \times img\_height - histogram[0])$
(3)When trying to subplot 3 plots and 1 image into on composite image, I need to use firstly plot three plots and then imshow one image with scale [0,1].

5 Suggested Enhancements

I don't know why to do Normalization here because it is useless and confusing when try to show the images/calculate histogram. Maybe when in deep learning models, such normalization is necessary. But in this project, it seems unnecessary.