hw2_ChengjunGuo

Chengjun Guo

January 2023

1 Explanation to the mystery

Based on the slide 23, the max value in the data is 255. The code on Slide 26 would divide all the images by the max value which is 255. The tvt totensor function will convert the value from [0,255] to [0.0,1.0] that is divide each batch by 255. Thus, two pieces of code is doing the same thing to the whole data by dividing it with 255. The results would be the same.

2 Output

Task 3.2

The front image and side image:

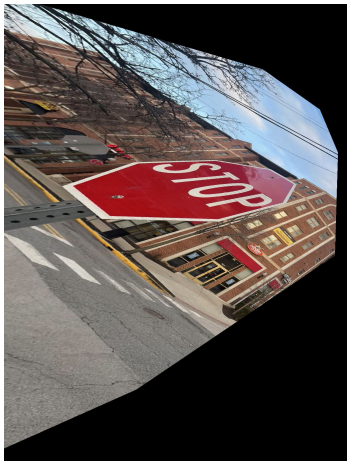
(a) front (b) side

Figure 1: input stop sign

The best transformed image For this task, I used the torchvision.transform.functional.perspective

Figure 2: transformed image

function to map from points to points. First, I used the GIMP to record the points location and then I used the list of points of the front image and the list of corresponding points on the side image to perform the transform. The results of randomaffine and the randomperspective are not as good as the direct warping. For reference, the unsuccessful transformed images are:



(a) 1_transformed

(b) 2_transformed

Figure 3: random outputs

Task 3.3

Original dataset:

(a) index 0 (b) index 5 (c) index 9

Figure 4: Original 0,5,9

Augmented:

(a) index 0 (b) index 5 (c) index 9

Figure 5: Transformed 0,5,9

Rationale behind the augmentation:For this task, I chose four steps for the augmentation. I first transformed the PIL image type to tensor with ToTensor function for future processing. Then I use Resize and CenterCrop to standardize the size of all images. This step will ensure a square shape of input image.(Some neural network requires the images to be a specific shape) The last step is to normalize the data that is to ensure the data to have similar data distribution.

Task 3.4

The four images in one batch:

(a) 2

(b) 6

(c) 7

(d) 8

Figure 6: batch images

time without batch		9.964999675750732	
		num worker	
batch size	4	8	16
4	0.0019996166229248047	0.004999399185180664	0.00599980354309082
8	0.006000041961669922	0.005000114440917969	0.005000114440917969
20	0.004999399185180664	0.005000114440917969	0.004999399185180664

3 Code

```
import os
import torch
import torchvision
import torchvision.transforms as tvt
from torchvision.io import read_image
from torch.utils.data import DataLoader, Dataset
from PIL import Image
import numpy as np
import random
import time


# task 3.2
def task3_2():
    im = Image.open("front.jpg")
    side = Image.open("side1.jpg")
    startpoints =
        [[434,702],[424,909],[722,560],[718,866]]
    endpoints = [[365,577],[367,781],[855,572],[859,776]]
    for i in range(3):
        # height = random.randint(24, 32) * 2
        # width = random.randint(24, 32) * 2
        transform = tvt.Compose([tvt.RandomAffine((40,80)
            )])
        img_transformed = transform(side)
        # img_transformed = side
        # perp = tvt.RandomPerspective()
        # startpoints, endpoints = perp.get_params(width,
            height, 0.5)
        img_transformed = tvt.functional.perspective(
            img_transformed, startpoints, endpoints)
        img_transformed.save("./transformed_images/"+str(
            i+1)+".jpg")
# img_transformed.show()
# pts_src = np.array([])
# pts_dst = np.array([])
# h, status = cv2.findHomography(pts_src, pts_dst)


# task 3.3
# transforms: resize,centercrop, normalize?
```

```
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, root):
        super().__init__()
        self.transform = tvt.Compose([tvt.ToTensor(),tvt.
            Resize(256), tvt.CenterCrop(256),tvt.Normalize
            ([0.5,0.5,0.5],[0.5,0.5,0.5])])
        self.images = []
        for image in os.listdir(root):
            if (image.endswith(".jpg")):
                img = Image.open(os.path.join(root,image)
                    )
                self.images.append((img,image.split(".")
                    [0]))


    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        if index >= self.__len__():
            raise IndexError('list index out of range')
        return self.transform(self.images[index][0]),
            self.images[index][1]

def task3_3():
    my_dataset = MyDataset("./images")
    print(len(my_dataset))
    index = 0
    print(my_dataset[index][0].shape, my_dataset[index
        ][1])
    torchvision.utils.save_image(my_dataset[index][0],
        my_dataset[index][1]+'_augmented.jpg')
    index = 5
    print(my_dataset[index][0].shape, my_dataset[index
        ][1])
    torchvision.utils.save_image(my_dataset[index][0],
        my_dataset[index][1] + '_augmented.jpg')
    index = 9
    print(my_dataset[index][0].shape, my_dataset[index
        ][1])
    torchvision.utils.save_image(my_dataset[index][0],
        my_dataset[index][1] + '_augmented.jpg')
```

```
# task 3.4
def task3_4():
    my_dataset = MyDataset("./images")
    data_loader = torch.utils.data.DataLoader(dataset=
        my_dataset, batch_size=4, shuffle=True,
        num_workers=0)
    # for i, data in enumerate(data_loader):
    #     img, label = data
    #     for j in range(4):
    #         torchvision.utils.save_image(img[j], label[
        j] + '_batch.jpg')
    #     break
    for i, data in enumerate(data_loader):
        print(data[0].shape,data[1])
    # start = time.time()
    # print(start)
    # for i in range(1000):
    #     x = my_dataset.__getitem__(0)
    # end = time.time()
    # print(end)
    # print('time without batch:',end - start)

    my_dataset = MyDataset("./images")
    my_dataset.images = my_dataset.images * 100

    start = time.time()
    data_loader = torch.utils.data.DataLoader(dataset=
        my_dataset, batch_size=20, shuffle=True,
        num_workers=16)
    end = time.time()
    print('time with batch:',end - start)


# task3_2()
# task3_3()
task3_4()
```