PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Deep Learning

Homework 2

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: January 25, 2023

1 Theory Question

If the pixel-value scaling by the piece of code in Slide 28 is on a per-image basis and if the same by the code shown on Slide 26 is on a batch basis, how come the two results are exactly the same?

In Slide 28, tvt.Totensor() is called on every single image of the batch and it scales the numpy array (the original tensor is reshaped and changed to a numpy array) in the range $[0, 255]$ to a tensor in the range $[0.0, 1.0]$. This scaling clearly does not depend on the value of the maximum pixel intensity of the image. In other words, all the pixels of the image are divided by 255.

In Slide 26, all the pixels in the all the images of the batch were divided by the maximum pixel value of the batch. As this value was 255, the results shown on Slide 26 were equal to the results shown in Slide 28.

2 Experiment

In this homework, we familiarize ourselves with the different transforms of the torchvision library. We also implement a custom Dataset class and create a DataLoader object.

2.1 Task 1

For the first experiment, I have taken the following two images of a stop sign.

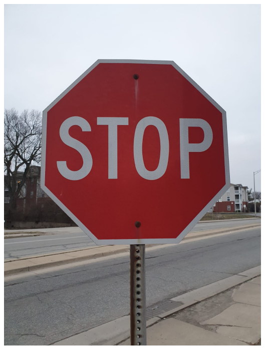


Figure 1: Straight on

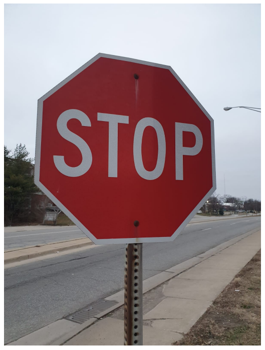


Figure 2: Oblique

2.2 Task 2

In this task we have to create our own custom Dataset class. For the dataset, I have taken 10 images of Cats from the Internet.

3 Results

3.1 Task 1

We use the tvt.functional.perspective() function to transform Figure 2 to Figure 1. The function has the parameters of start_pt and end_pt which are the lists containing of 4 corresponding points in the original image and the transformed image (These 4 corresponding coordinates are manually obtained). The function then uses these 4 points to calculate the Homography between the two images which is then applied to the original image to obtain the transformed image.

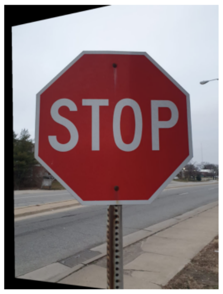


Figure 3: Transformed Figure 2. It looks similar to Figure 1

Two measure the similarity between two images we use the Wasserstein Distance. For the three channels between the original two images it is calculated as $[0.00135, 0.00127, 0.00099]$ and between the straight on original image and the transformed oblique image it is calculated as $[0.00122, 0.00100, 0.00089]$.

3.2 Task 2

In the figure below, three images of the dataset are shown before (left) and after (right) their augmentations. The augmentations chosen are:

1. transforms.Resize() - To make all the images of the dataset have the same size.

2. transforms.RandomRotation() - Rotates all the images of the dataset by a random degree.

3. transforms.GaussianBlur() - Smooths all the images of the dataset to reduce the noise in them.

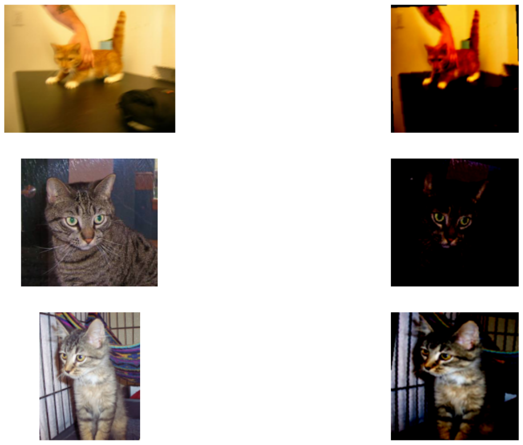


Figure 4: Images before and after transforms

The next figure shows the four images present in a single batch as returned by the dataloader.

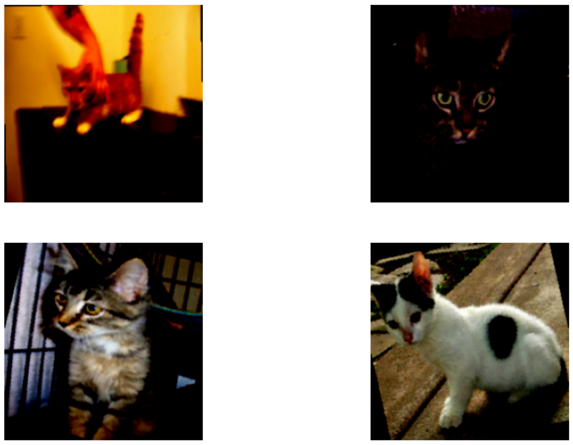


Figure 5: All the images in a batch

Finally in the table below, the time taken by the dataloader instance to process 1000 images for different number of workers and batch sizes are given.

Time Elapsed in seconds	**For 0 workers**	**For 1 worker**	**For 2 workers**
For batch size of 1	125.85	191.71	210.52
For batch size of 4	31.18	46.43	52.41
For batch size of 10	12.09	19.31	24.11
For batch size of 25	4.99	8.01	9.56
For batch size of 100	1.24	1.93	3.89

For reference, the time taken by the custom dataset instance to process 1000 images was 22.83 seconds. Therefore, for larger batchsizes the dataloader performs better than the dataset instance.

In [1]:

```
#importing the libraries
from PIL import Image
import matplotlib.pyplot as plt
import torch
import torchvision.transforms as transforms
import numpy as np
from scipy.stats import wasserstein_distance
```

In [2]:

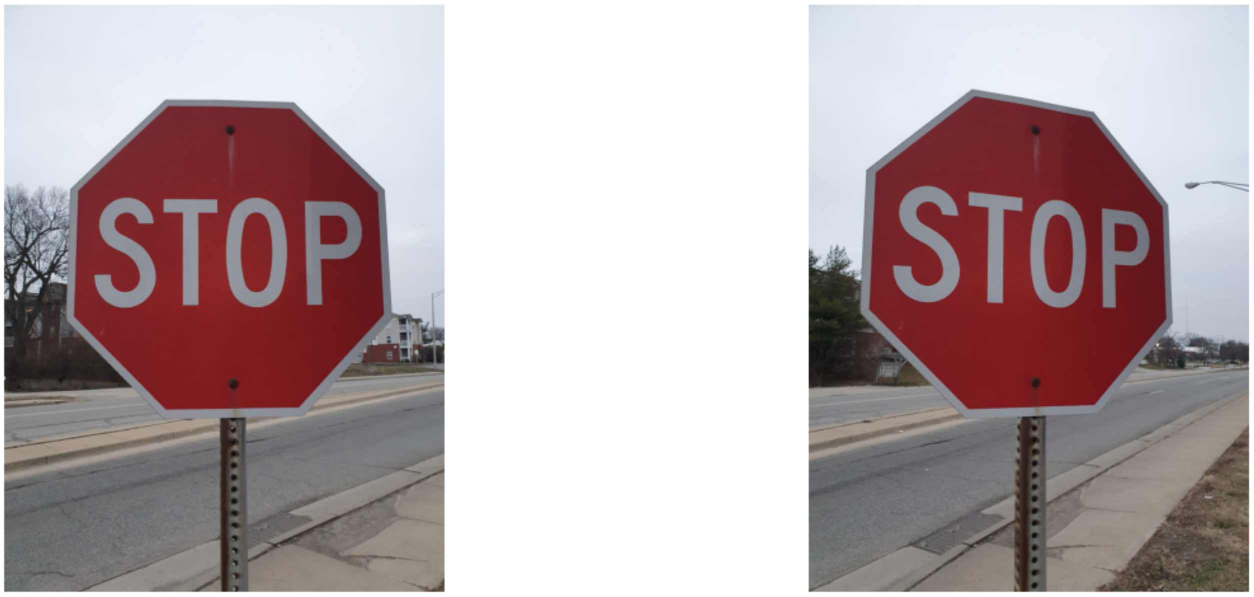
```
#Computing the Wasserstein Distance between two images
def dist(im1,im2):
    #Converting PIL to Tensor
    transform = transforms.Compose([transforms.ToTensor()])
    img1  = transform(im1)
    img2  = transform(im2)

    lc_img1 = [img1[c] for c in range(3)]
    lc_img2 = [img2[c] for c in range(3)]

    #Computing the histogram for each channel of the two images
    h_img1 = [torch.histc(lc_img1[c], bins=100, min=0.0,max=1.0) for c in range(3)]
    h_img2 = [torch.histc(lc_img2[c], bins=100, min=0.0,max=1.0) for c in range(3)]

    dist = []
    #Computing the Wasserstein Distance for each channel between two images
    for c in range(3):
        a = h_img1[c].div(h_img1[c].sum())
        b = h_img2[c].div(h_img2[c].sum())
        d = wasserstein_distance(a,b)
        dist.append(d)
    return dist
```

In [3]:

```
#Printing the two input images
fig = plt.figure(figsize = (15,6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.axis('off')
ax2.axis('off')

im1 = Image.open('I1.jpeg')
img1 = np.asarray(im1)
ax1.imshow(img1)

im2 = Image.open('I2.jpeg')
img2 = np.asarray(im2)
ax2.imshow(img2)
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x2282c35bee0>
```



In [4]:

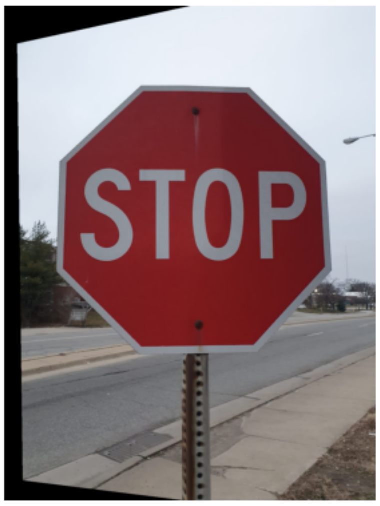
```
#Performing projective transform on the right image to return the left image
end_pt = [[180,503],[1037,504],[169,859],[1057,857]]
start_pt = [[158,450],[979,546],[140,840],[991,869]]

transf_im2 =  transforms.functional.perspective(im2,start_pt,end_pt)
```

In [5]:

```
#The transformed right image
plt.imshow(np.asarray(transf_im2))
plt.axis("off")
```

Out[5]:

```
(-0.5, 1199.5, 1599.5, -0.5)
```



In [6]:

```
#Wasserstein Distance between the original two images for all the three channels
dist(im1,im2)
```

Out[6]:

```
[0.0013495103830609876, 0.0012694791786270801, 0.0009966562065164906]
```

In [7]:

```
#Wasserstein Distance between the transformed image for all the three channels
dist(im1,transf_im2)
```

Out[7]:

```
[0.0012242603535764826, 0.0010014792433685217, 0.0008933854644601524]
```

In [ ]:

In [1]:

```
#importing the libraries
import torch
from torch.utils.data import Dataset,DataLoader
import os
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import time
```

In [2]:

```
#Creating the custom dataset class
class MyDataset(Dataset):
    def __init__(self,root): #Initializer
        super().__init__()
        self.filenames = [os.path.join(root, x) for x in os.listdir(root)]
        #Augmenting the dataset
        self.transforms = transforms.Compose([transforms.Resize((224,224)),
                                              transforms.RandomRotation(15),
                                              transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
                                              transforms.ToTensor(),
                                              transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])])

    def __len__(self): #Returns the number of images in the dataset
        return len(self.filenames)

    def __getitem__(self,index): #Loads the image and applies the transforms on it
        img = Image.open(self.filenames[index])
        n_img = self.transforms(img)
        cls = torch.randint(0,10,(1,)).item()
        return n_img,cls
```

In [3]:

```
#Creating an object of the custom class
my_dataset = MyDataset('/content/drive/MyDrive/Cat_images')
len(my_dataset)
```

Out[3]:

```
10
```

In [4]:

```
#Printing the size and the class of the image of the dataset
index = 9
print(my_dataset[index][0].shape,my_dataset[index][1])
```

```
torch.Size([3, 224, 224]) 9
```

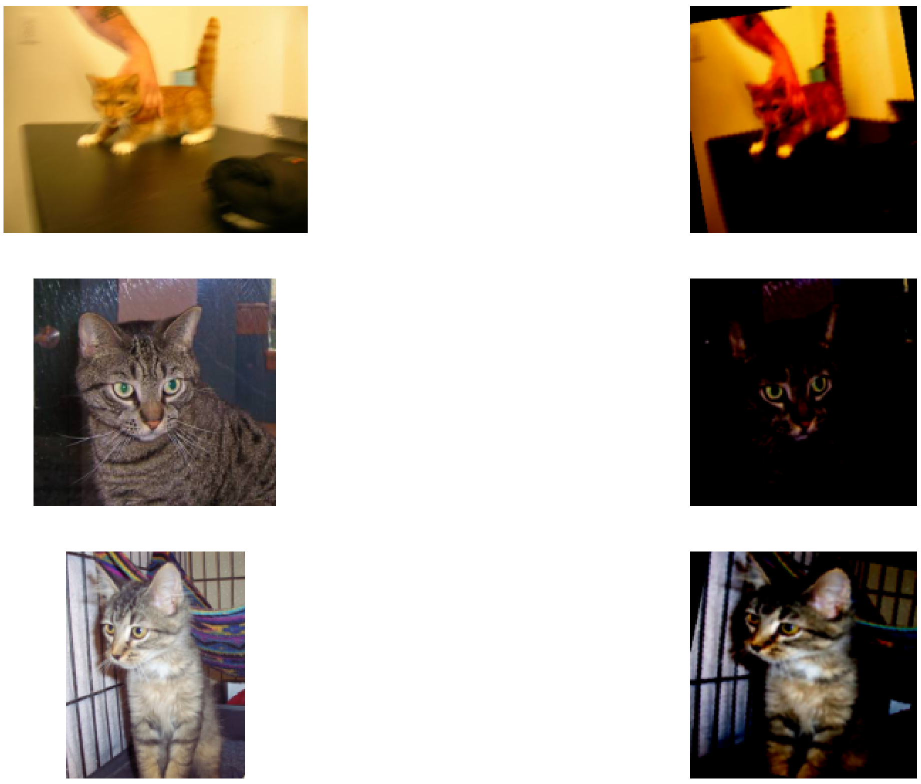
In [5]:

```
#Printing 3 images of the dataset before and after transforms
fig = plt.figure(figsize = (15,10))
ax1 = fig.add_subplot(321)
ax2 = fig.add_subplot(322)
ax3 = fig.add_subplot(323)
ax4 = fig.add_subplot(324)
ax5 = fig.add_subplot(325)
ax6 = fig.add_subplot(326)

ax1.axis('off')
ax2.axis('off')
ax3.axis('off')
ax4.axis('off')
ax5.axis('off')
ax6.axis('off')

img1 = Image.open('/content/drive/MyDrive/Cat_images/0.jpg')
img1 = np.asarray(img1)
ax1.imshow(img1)
ax2.imshow(np.transpose(my_dataset[0][0].numpy(),(1,2,0)))

img3 = Image.open('/content/drive/MyDrive/Cat_images/1.jpg')
img3 = np.asarray(img3)
ax3.imshow(img3)
ax4.imshow(np.transpose(my_dataset[1][0].numpy(),(1,2,0)))

img5 = Image.open('/content/drive/MyDrive/Cat_images/2.jpg')
img5 = np.asarray(img5)
ax5.imshow(img5)
ax6.imshow(np.transpose(my_dataset[2][0].numpy(),(1,2,0)))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x7fe019e896a0>
```



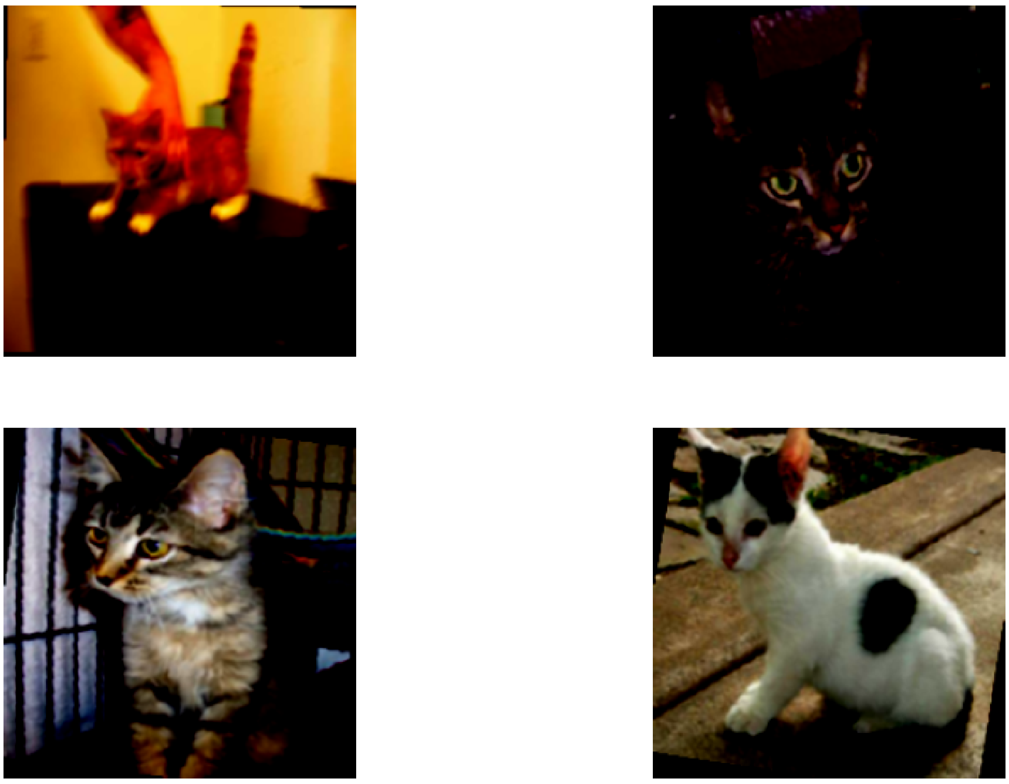
In [6]:

```
#Wrapping an instance of the custom dataset class in the DataLoader class
my_dataloader = DataLoader(my_dataset, batch_size= 4, shuffle= False, num_workers=2)
```

In [7]:

```
#Plotting the 4 images of a batch together
for i, (img,cls) in enumerate(my_dataloader):
    img0 = img[0]
    img1 = img[1]
    img2 = img[2]
    img3 = img[3]

    fig = plt.figure(figsize = (15,10))
    ax1 = fig.add_subplot(221)
    ax2 = fig.add_subplot(222)
    ax3 = fig.add_subplot(223)
    ax4 = fig.add_subplot(224)

    ax1.axis('off')
    ax2.axis('off')
    ax3.axis('off')
    ax4.axis('off')

    ax1.imshow(np.transpose(img0.numpy(),(1,2,0)))
    ax2.imshow(np.transpose(img1.numpy(),(1,2,0)))
    ax3.imshow(np.transpose(img2.numpy(),(1,2,0)))
    ax4.imshow(np.transpose(img3.numpy(),(1,2,0)))

    break
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
```



In [8]:

```
#Time needed to load and augment 1000 images using the custom dataset class
st1 = time.time()
for i in range(1000):
    cl = my_dataset[i%10][1]

et1 = time.time()
print("Time taken is %f seconds"%(et1-st1) )
```

```
Time taken is 22.825076 seconds
```

In [9]:

```
#Time needed to load and augment 1000 images using the DataLoader class with different batch sizes and number of workers
workers = [0,1,2]
batch_sizes = [1,4,10,25,100]
for worker in workers:
  for batch_size in batch_sizes:
    temp_dataloader = DataLoader(my_dataset, batch_size=batch_size,shuffle= False,num_workers=worker)
    st2 = time.time()
    for j in range(int(1000/batch_size)):
      for i, (img,cls) in enumerate(temp_dataloader):
        continue
    et2 = time.time()
    print("For batch size of %d and number of workers = %d, time taken is %f seconds"%(batch_size,worker,et2-st2) )
  print("\n")
```

```
For batch size of 1 and number of workers = 0, time taken is 125.854827 seconds
For batch size of 4 and number of workers = 0, time taken is 31.187582 seconds
For batch size of 10 and number of workers = 0, time taken is 12.091074 seconds
For batch size of 25 and number of workers = 0, time taken is 4.992397 seconds
For batch size of 100 and number of workers = 0, time taken is 1.243059 seconds


For batch size of 1 and number of workers = 1, time taken is 191.716542 seconds
For batch size of 4 and number of workers = 1, time taken is 46.429814 seconds
For batch size of 10 and number of workers = 1, time taken is 19.314854 seconds
For batch size of 25 and number of workers = 1, time taken is 8.011915 seconds
For batch size of 100 and number of workers = 1, time taken is 1.927321 seconds


For batch size of 1 and number of workers = 2, time taken is 210.524373 seconds
For batch size of 4 and number of workers = 2, time taken is 52.407943 seconds
For batch size of 10 and number of workers = 2, time taken is 24.105997 seconds
For batch size of 25 and number of workers = 2, time taken is 9.563501 seconds
For batch size of 100 and number of workers = 2, time taken is 3.893192 seconds
```

In [ ]:

```

```