Assignment 13

# Running DL26B.py several more times (2 runs)

## Output

### DL26A.py

Text

Description automatically generated

### DL26B.py

A picture containing text, clipart

Description automatically generated Text

Description automatically generated with low confidence

Qr code

Description automatically generated with medium confidenceQr code

Description automatically generated

Logo

Description automatically generated with medium confidence A picture containing qr code

Description automatically generated Qr code

Description automatically generated with low confidence A black and white image of a person's face

Description automatically generated with low confidence

Qr code

Description automatically generated Graphical user interface

Description automatically generated with medium confidence

## Code

### DL26B.py

# DL26B.py CS5173/6073 2023 cheng  
# MNIST edge detection using the loaded DL26Aepoch1.zip  
# Usage: python DL26B.py  
  
import torch  
import torchvision  
from torchvision import transforms  
import torch.nn as nn  
import torch.nn.functional as F  
import numpy as np  
import matplotlib.pyplot as plt  
  
testset = torchvision.datasets.MNIST('/data/', train=False,  
 transform=torchvision.transforms.ToTensor())  
test\_size = len(testset)  
  
class Model(nn.Module):  
 def \_\_init\_\_(self):  
 super(Model, self).\_\_init\_\_()  
 self.cnn1 = nn.Conv2d(1, 64, 5, stride=2, padding=2)  
 self.cnn2 = nn.Conv2d(64, 128, 5, stride=2, padding=2)  
 self.conv1 = nn.ConvTranspose2d(128, 64, 5, stride=1, padding=2, bias=False)  
 self.conv2 = nn.ConvTranspose2d(64, 32, 4, stride=2, padding=1, bias=False)  
 self.conv3 = nn.ConvTranspose2d(32, 1, 4, stride=2, padding=1, bias=False)  
  
 def forward(self, x):  
 x = F.relu(self.cnn1(x))  
 x = self.cnn2(x)  
 x = F.relu(self.conv1(x))  
 x = F.relu(self.conv2(x))  
 return self.conv3(x)  
  
model = Model()  
model.load\_state\_dict(torch.load("DL26Aepoch1.zip"))  
  
for i in range(5):  
 index = np.random.randint(test\_size)  
 sample, t1 = testset[index]  
 s = sample.view((1, 1, 28, 28))  
 y = model(s)  
 plt.subplot(1, 2, 1)  
 plt.imshow(s[0][0], cmap='gray')  
 plt.xticks([])  
 plt.yticks([])  
 plt.subplot(1, 2, 2)  
 plt.imshow(y[0][0].detach().numpy(), cmap='gray')  
 plt.xticks([])  
 plt.yticks([])  
 plt.show()

# Changing Filter to Gaussian Blur

## Output

### DL26A.py

Text

Description automatically generated

### DL26B.py

A picture containing graphical user interface

Description automatically generated Qr code

Description automatically generated with medium confidence

A picture containing logo

Description automatically generated A picture containing text

Description automatically generated

A picture containing logo

Description automatically generated

## Code

### DL26A.py

# DL26A.py CS5173/6073 2023 cheng  
# MNIST edge detection training with encoder-decoder  
# ImageFilter.FIND\_EDGES used to produce image-to-image translation  
# in the training set and trained model is saved in DL26Aepoch1.zip  
# The application of ImageFilter.FIND\_EDGES to all training images may take a few minutes.   
# Usage: python DL26A.py  
  
import torch  
import torchvision  
from PIL import Image, ImageFilter  
from torchvision import transforms  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
import numpy as np  
  
dataset = torchvision.datasets.MNIST('/data/', train=True)  
train\_size = len(dataset)  
totensor = transforms.ToTensor()  
d2 = []  
for i in range(train\_size):  
 data, target = dataset[i]  
 target = data.filter(ImageFilter.GaussianBlur)  
 d2.append((totensor(data), totensor(target)))  
  
batch\_size\_train = 256  
n\_epochs = 1  
  
train\_loader = torch.utils.data.DataLoader(  
 d2,  
 batch\_size=batch\_size\_train, shuffle=True)  
  
class Model(nn.Module):  
 def \_\_init\_\_(self):  
 super(Model, self).\_\_init\_\_()  
 self.cnn1 = nn.Conv2d(1, 64, 5, stride=2, padding=2)  
 self.cnn2 = nn.Conv2d(64, 128, 5, stride=2, padding=2)  
 self.conv1 = nn.ConvTranspose2d(128, 64, 5, stride=1, padding=2, bias=False)  
 self.conv2 = nn.ConvTranspose2d(64, 32, 4, stride=2, padding=1, bias=False)  
 self.conv3 = nn.ConvTranspose2d(32, 1, 4, stride=2, padding=1, bias=False)  
  
 def forward(self, x):  
 x = F.relu(self.cnn1(x))  
 x = self.cnn2(x)  
 x = F.relu(self.conv1(x))  
 x = F.relu(self.conv2(x))  
 return self.conv3(x)  
model = Model()  
optimizer = optim.Adam(model.parameters())  
loss\_fn = nn.MSELoss()  
  
for epoch in range(n\_epochs):  
 for batch\_idx, (data, target) in enumerate(train\_loader):  
 y = model(data)  
 train\_loss = loss\_fn(y, target)  
 if batch\_idx % 10 == 0:  
 print('train', epoch, batch\_idx, float(train\_loss))   
 optimizer.zero\_grad()  
 train\_loss.backward()  
 optimizer.step()  
  
torch.save(model.state\_dict(), "DL26Aepoch1.zip")

### DL26B.py

# DL26B.py CS5173/6073 2023 cheng  
# MNIST edge detection using the loaded DL26Aepoch1.zip  
# Usage: python DL26B.py  
  
import torch  
import torchvision  
from torchvision import transforms  
import torch.nn as nn  
import torch.nn.functional as F  
import numpy as np  
import matplotlib.pyplot as plt  
  
testset = torchvision.datasets.MNIST('/data/', train=False,  
 transform=torchvision.transforms.ToTensor())  
test\_size = len(testset)  
  
class Model(nn.Module):  
 def \_\_init\_\_(self):  
 super(Model, self).\_\_init\_\_()  
 self.cnn1 = nn.Conv2d(1, 64, 5, stride=2, padding=2)  
 self.cnn2 = nn.Conv2d(64, 128, 5, stride=2, padding=2)  
 self.conv1 = nn.ConvTranspose2d(128, 64, 5, stride=1, padding=2, bias=False)  
 self.conv2 = nn.ConvTranspose2d(64, 32, 4, stride=2, padding=1, bias=False)  
 self.conv3 = nn.ConvTranspose2d(32, 1, 4, stride=2, padding=1, bias=False)  
  
 def forward(self, x):  
 x = F.relu(self.cnn1(x))  
 x = self.cnn2(x)  
 x = F.relu(self.conv1(x))  
 x = F.relu(self.conv2(x))  
 return self.conv3(x)  
  
model = Model()  
model.load\_state\_dict(torch.load("DL26Aepoch1.zip"))  
  
for i in range(5):  
 index = np.random.randint(test\_size)  
 sample, t1 = testset[index]  
 s = sample.view((1, 1, 28, 28))  
 y = model(s)  
 plt.subplot(1, 2, 1)  
 plt.imshow(s[0][0], cmap='gray')  
 plt.xticks([])  
 plt.yticks([])  
 plt.subplot(1, 2, 2)  
 plt.imshow(y[0][0].detach().numpy(), cmap='gray')  
 plt.xticks([])  
 plt.yticks([])  
 plt.show()

# Changing Filter to Sharpen

## Output

### DL26A.py

Text

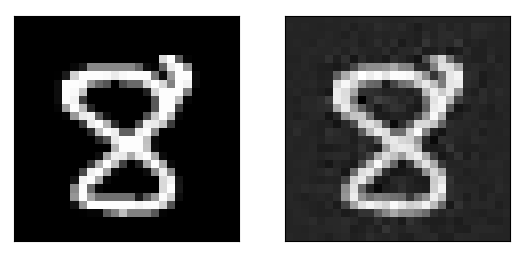
Description automatically generated

### DL26B.py

A picture containing text, black, clipart

Description automatically generated Qr code

Description automatically generated

 A black and white image of a person's face

Description automatically generated with low confidence

Text

Description automatically generated with low confidence

## Code

### DL26A.py

Only changed this line, rest of the code is same as the original

target = data.filter(ImageFilter.SHARPEN)

### DL26B.py

Same Code as above two.

# Changing Filter to SmoothMore

## Output

### DL26A.py

Text

Description automatically generated

### DL26B.py

Text

Description automatically generated Logo

Description automatically generated with medium confidence

Text

Description automatically generated with low confidence Qr code

Description automatically generated with medium confidence

A picture containing qr code

Description automatically generated

## Code

### DL26A.py

Only changed this line, rest of the code is same as the original

target = data.filter(ImageFilter.SMOOTH\_MORE)

### DL26B.py

Same Code as above.

# Comments

As you can observe in the output above, different images filters produce different results. The original filter makes the image grainier, while gaussian blur, blurs out the image very well. Sharpen and Smooth More also apply their respective filters to the images. Overall, Gaussian Blur has the least training loss, while Sharpen and the original have the highest.