50.039 Theory and Practice of Deep Learning HW11
Loh De Rong (1003557)

The complete code and printouts can be found in the hw11.ipynb.

1. Copy and paste the code for your Critic class. Briefly explain your choice of architecture.

```
# Critic
class Critic(nn.Module):

  def __init__(self, image_size):
    """
    Only forced parameter will be the image size, set to 28.
    """
    # Init from nn.Module
    super().__init__()

    # Conv layers
    self.D = nn.Sequential(nn.Conv2d(1, 128, 7, stride = 2),
                nn.LeakyReLU(0.2),
                nn.Conv2d(128, 256, 5, stride = 2, padding = 1),
                nn.LeakyReLU(0.2),
                nn.Conv2d(256, 1, 5, stride = 1))

  def forward(self, x):
    return self.D(x)
```

For the first two layers, the number of channels progressively increases while the spatial dimension decreases. This is to capture more high-level concepts in different activation maps. The use of LeakyRelu could help introduce non-linearity as well as ensure the invertibility property, in the hopes of more effective learning.

50.039 Theory and Practice of Deep Learning HW11
Loh De Rong (1003557)

2. Copy and paste the code for your Generator class. Briefly explain your choice of architecture.

```
# Generator
class Generator(nn.Module):

  def __init__(self, latent_size, image_size):
    """
    Only forced parameters will be the image size, set to 28,
    and the latent size set to 64.
    """
    # Init from nn.Module
    super().__init__()

    # ConvTranspose layers
    self.G = nn.Sequential(nn.ConvTranspose2d(latent_size, 32, 7, stride = 1),
                nn.LeakyReLU(0.2),
                nn.ConvTranspose2d(32, 16, 3, stride = 2, output_padding = 1, padding = 1),
                nn.LeakyReLU(0.2),
                nn.ConvTranspose2d(16, 1, 3, stride = 2, output_padding = 1, padding = 1),
                nn.Tanh())

  def forward(self, x):
    return self.G(x)
```

The number of channels progressively decreases while the spatial dimension increases through upsampling. In particular, the use of deconvolutional layers has learnable parameters that can be updated during backpropagation. The use of LeakyRelu could help introduce non-linearity as well as ensure the invertibility property, in the hopes of more effective learning. The use of Tanh activation function in the final layer is to normalize the image.

3. For how many iterations did you have to train when using Wasserstein with Conv/TransposeConv layers to get plausible images from the generator? Is it training faster than the Fully Connected Wasserstein/Vanilla GAN?

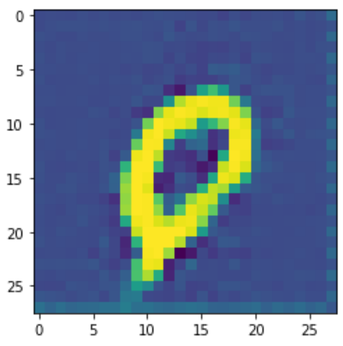
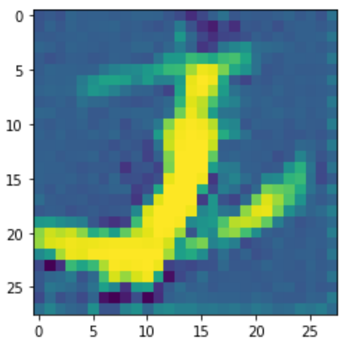
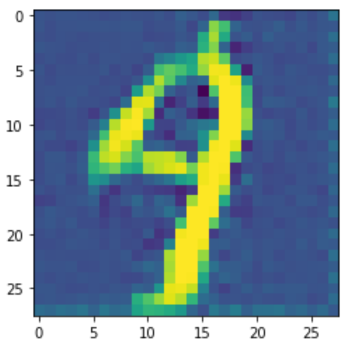
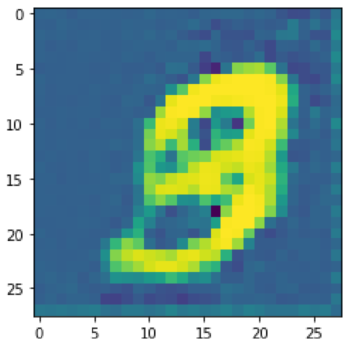
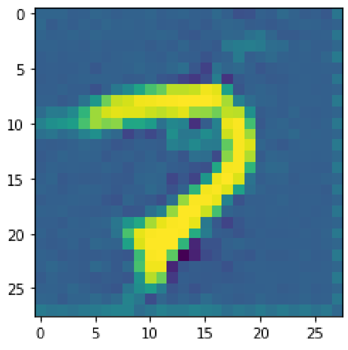
I had to train for around 20 epochs. In terms of the amount of training time required for convergence, training using Wasserstein with Conv/TransposeConv layers is faster than with Fully Connected Wasserstein /Vanilla GAN. This is likely because the convolutional layers can perform better as feature extractor.

However, in terms of training time per epoch, it takes longer to train with Conv/TransposeConv layers. This is because there are more parameters for the convolutional layers (due to the large number of input and output channels) than the Fully Connected layers, as well as more convolutions (and hence scalar multiplications) that need to be performed, resulting in longer time for each pass.

During my training, I realize that my Generator model was overpowering my Critic model, so it was very difficult for my Critic model to learn and discriminate the fake images in this kind of 1-to-1 interleaved training mode. As a result, there was no incentive for the Generator model to produce more plausible images to trick the Critic model. Therefore, I also increased the number of training iterations for my Critic model to give it more training time, as compared to the Generator model. This evens out the learning capability of each model, which helps in the training of the Generator model to produce more plausible images, but at the expense of more training time for every epoch.

4. Display some samples generated by your trained generator. Do they look plausible?

The images are plausible, but not as great as the Fully Connected Wasserstein/Vanilla GAN.

50.039 Theory and Practice of Deep Learning HW11
Loh De Rong (1003557)

5. Let us assume we use Conv2d layers in the Critic. We do NOT use Transposed Conv2d layers, but only Fully Connected layers in the Generator. Would the GAN still be able to train both models or would it encounter difficulties? Discuss.

Yes, the GAN would still be able to train both models. Fully connected layers can approximate any functions, and should be able to generate plausible images with sufficient learning. However, the models using Fully Connected layers might face more difficulty in picking up local features, and take a time longer time to converge.