Assignment-3
Deep Generative Modeling

Name:	Muhammad Ibrahim
Student ID:	8029808

(Graduate Student)

I, Muhammad Ibrahim attest that the work. I am submitting my own work and that it has not been copied / plagiarized from online or other sources. Any sourced material used for completing this work has been properly cited. Ibrahim

Problem-1:

As discussed in class we decided that WGAN-GP should be a significant improvement to DCGAN. Your job is to try and show that this is indeed the case by applying both models to the same problem. The codes provided in Lectures 8 and 9 are applied to two different problems. Make appropriate changes so that the models are effectively the same with the exception of the loss function, and are applied to the same data. Then:

1. Train and evaluate both models for the same choice of hyperparameters. Comment on training performance and support your arguments with training curve plots.
2. (**Graduate students only**) For either DCGAN or WGAN-GP (or both if you are feeling adventurous) change a hyperparameter (your choice) and comment on the effect that it has on training. Is the training sensitive or not? Justify your answer.

---

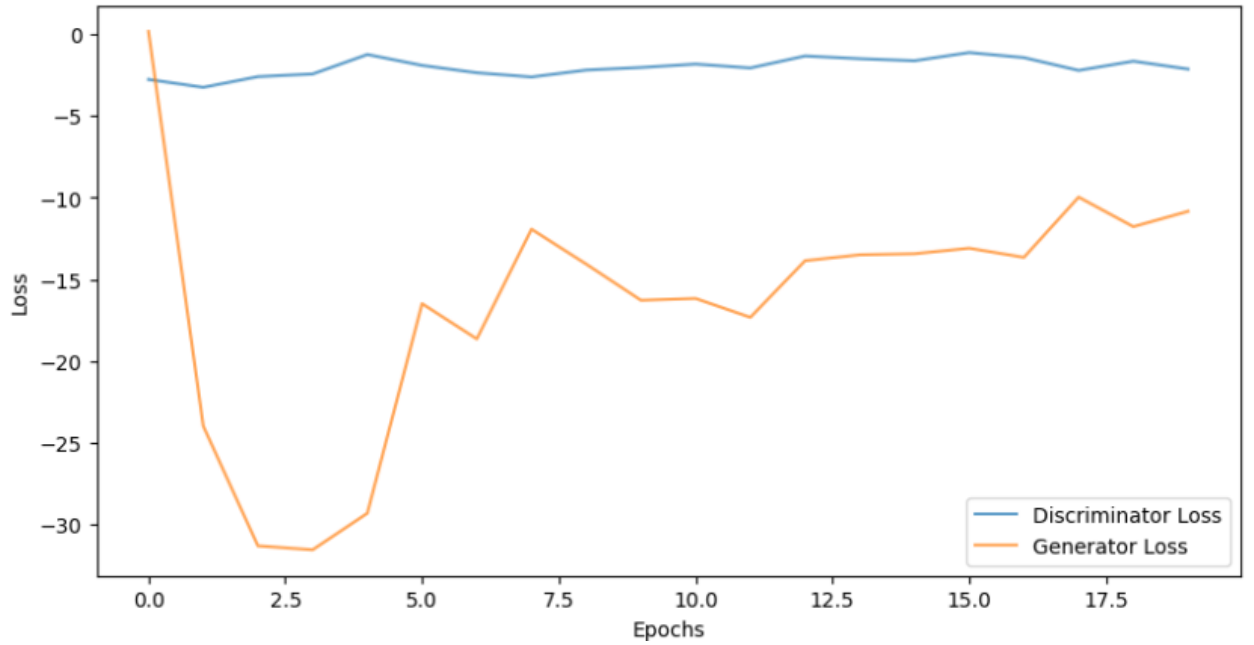
a)

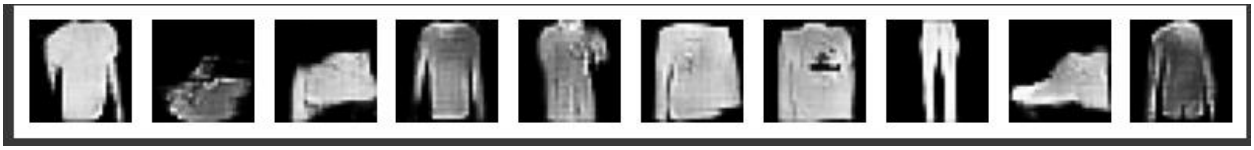
First both models are tested at learning rate of 0.0001 and batch size 512 with same architecture. Due to limited resources I tested on small dataset fashion mnist with each image being 28,28 size.

The training time of WGAN (45sec per epoch) was triple the amount of DCGAN (15sec) and DCGAN converged faster than WGAN. However in DCGAN on some occasions my generator could not improve further and was stuck somewhere.

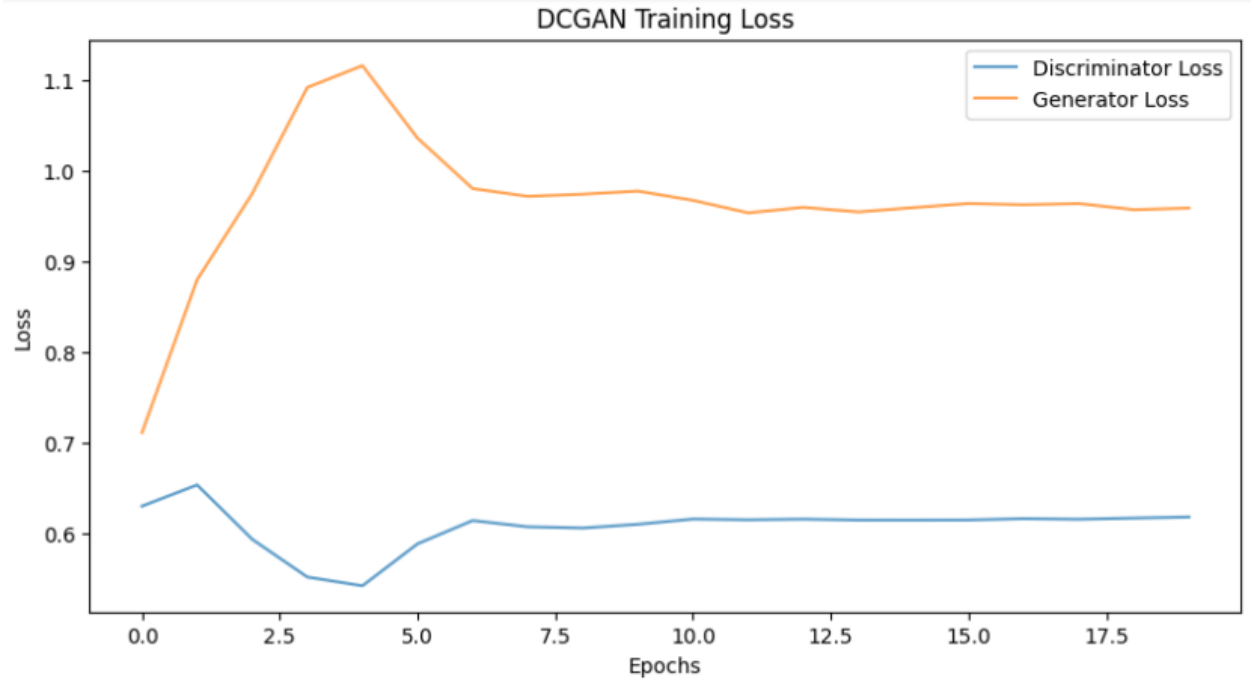
WGAN was way more stable on most of the parameters. However on some parameters DCGAN was a bit unstable. In WGAN I was getting a bit shaky graph but still the loss was significantly decreased while DCGAN graph was smooth.

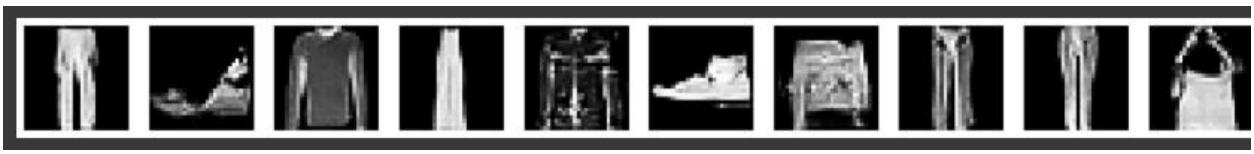
WGAN


0
−5
−10
−15
−20
−25
−30
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
Discriminator Loss
Generator Loss



DCGAN


DCGAN Training Loss
1.1
1.0
0.9
0.8
0.7
0.6
Loss
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
Discriminator Loss
Generator Loss



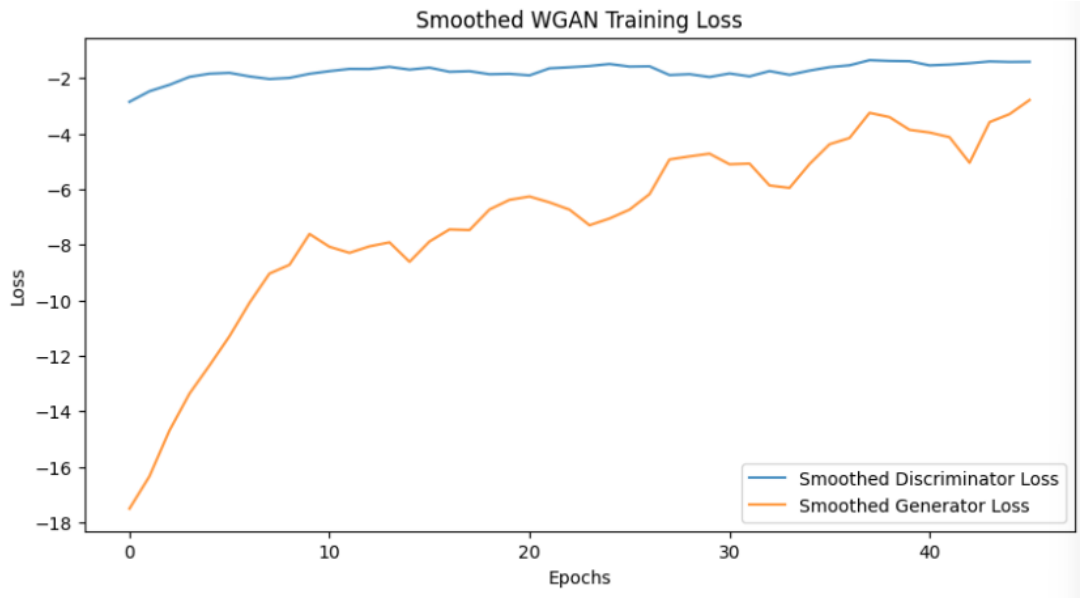
B)

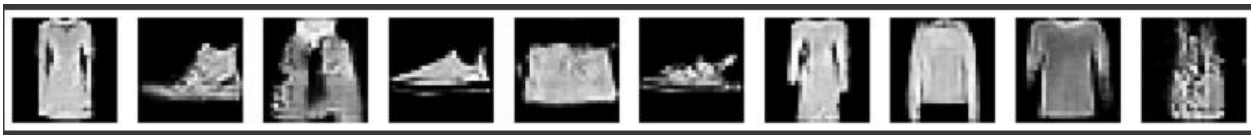
According to my observations, WGAN is a bit more sensitive but stable and does not diverge. However, DCGAN diverges but it does not show shaky behavior.

I have smoothed the graph of **WGAN** a bit to show the trend. When I increased the learning rate to 0.003 and reduced the batch size to 256. It was giving me good results at 25 epoch.

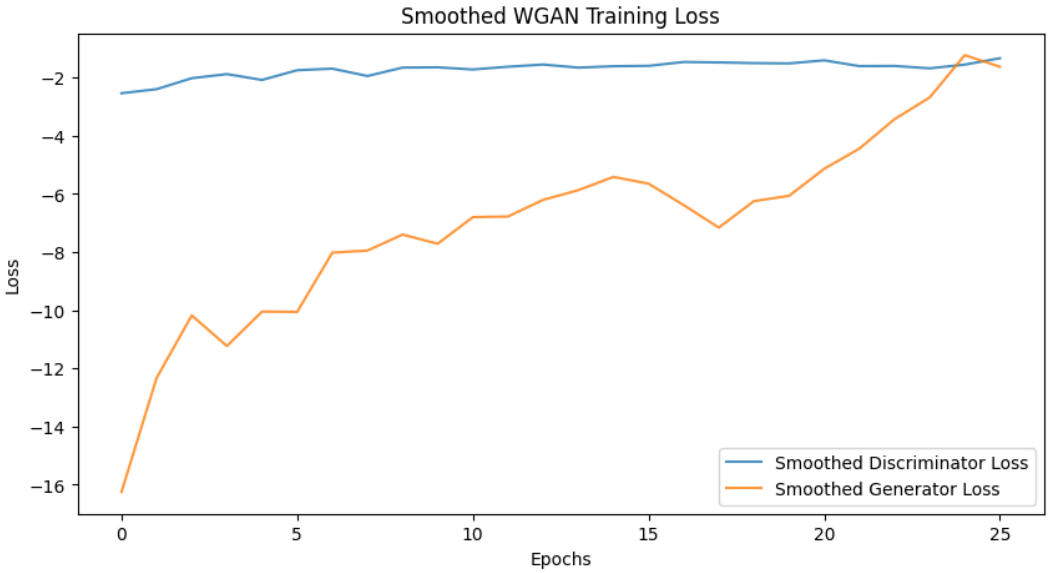
WGAN:

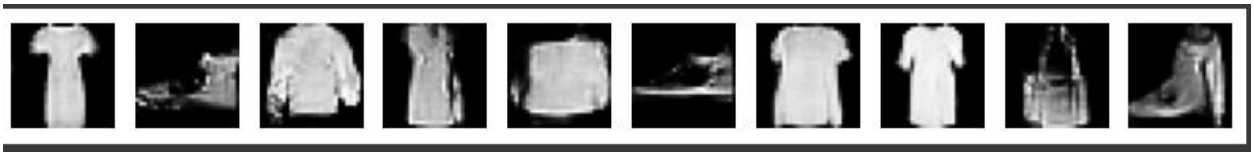
Learning rate = 0.0002, batch size = 512





Batch size = 256, learning rate = 0.0003

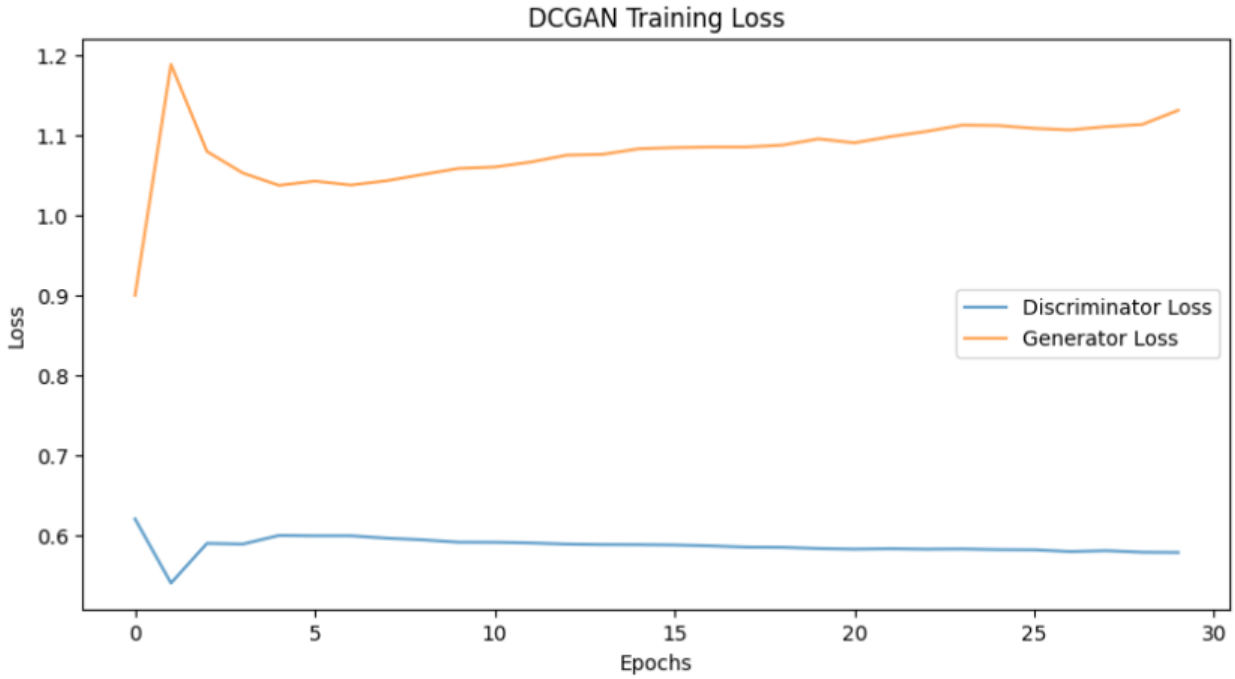


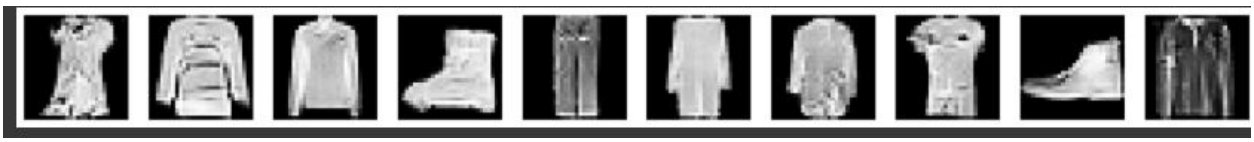


Learning rate = 0.002, batch size = 256

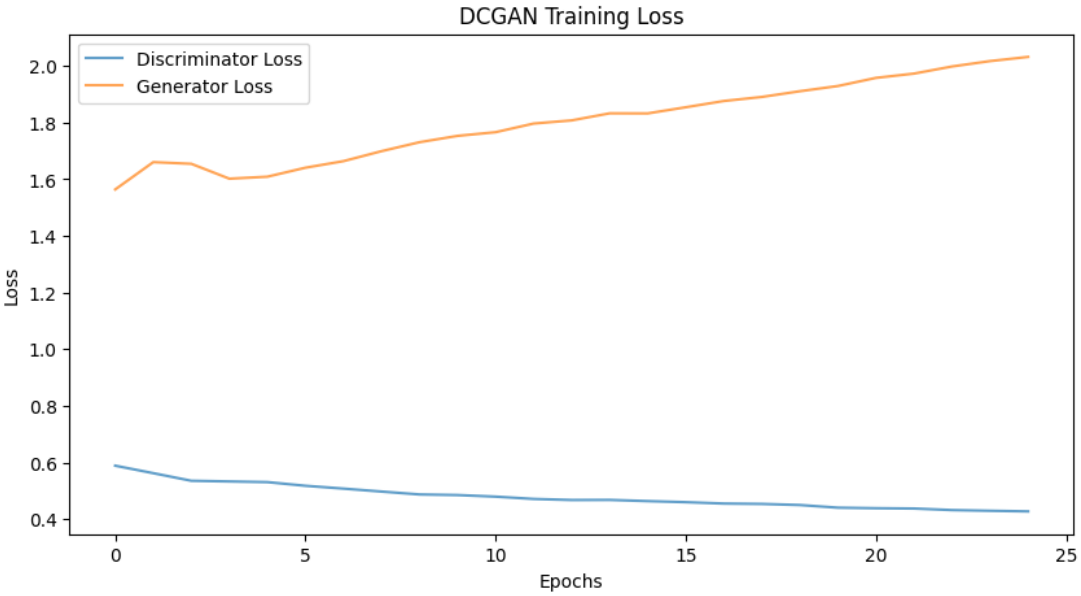
DCGAN:

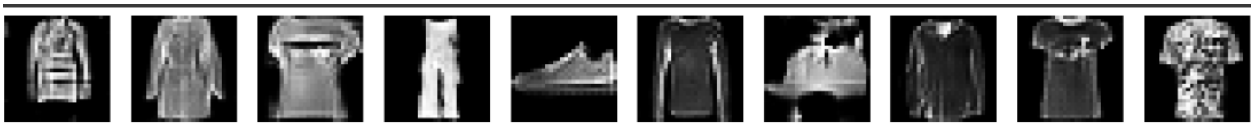
Learning rate = 0.0003, batch size = 256





Learning rate = 0.002, batch size =256





When I increased the learning rate of DCGAN to 0.002 it started the generator loss started to diverge, but when I changed the parameter of WGAN, it showed very unstable behavior which shows that it is very sensitive to the hyperparameter. Even for other parameter the graph was a bit shaky. But still it does not diverged.

Problem-2:

Given the Conditional GAN notebook (Lecture 10 Program a) explain, at a high level, how the conditional GAN is implemented focusing on the loss function and how attributes are handled. What flavour of Conditional GAN is this?

In simple GANs, there was no input to the model for generating a specific type of image. However, in conditional GANs, we enforce the model to generate a specific type of image. This allows us to train a single GAN to generate images based on a given condition, rather than training separate models for each class.

In terms of the loss function, the following was the standard loss function used in a simple GAN:

$$E_x[\log D\,(x)] + E_z\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Now, when we introduce conditioning, the loss function is modified to incorporate the given condition. This ensures that both the generator and discriminator take the class label into account during training.

$$E_x[\log D\,(x \mid y)] + E_z\left[\log\left(1 - D\big(G(z \mid y)\big)\right)\right]$$

Previously, the input to the generator was only latent noise. In the conditional GAN given in the code, we concatenate the class labels (one-hot encoded vectors) with the latent noise before feeding them into the generator.

For example, if the latent noise was originally 128-dimensional and we use the MNIST dataset with 10 classes, the generator's input will now be 138-dimensional (128 noise + 10 class labels). Similarly, for the discriminator, we concatenate this attribute vector with both real and fake images.

Suppose the original image shape was (28, 28, 1). After concatenation with the one-hot encoded label (broadcasted across the spatial dimensions), the new input shape becomes (28, 28, 11). This means that both the generator and the discriminator now depend on the attribute vector.

This modification helps the discriminator determine whether an image is real or fake based on the given attribute. Additionally, it allows the generator to produce samples conditioned on specific attributes.

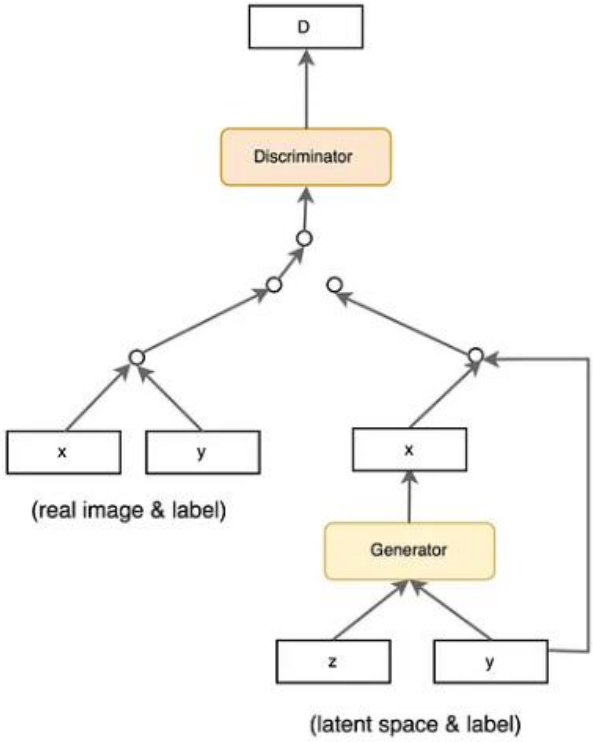


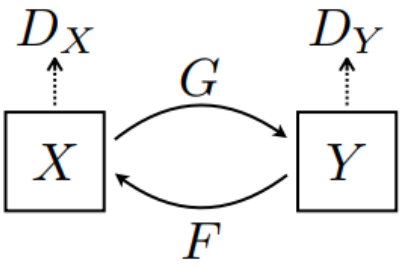
Fig. Condtional GANs (Image taken from [1])

In the given code it was vanilla conditional GAN as the output of the discriminator was only the probability of image being real or fake.

Problem - 3

1. Explain how the loss function is implemented, including what terms are evaluated, why they are included, and how they are weighted. Note that there is a discussion of the loss function in Lecture 10 that was not covered in detail in class.

Before explaining loss terms first let me explain Cycle GAN model. In cycle GAN we do not need a pair of data in training. So, we use two generators and two discriminators as shown in the following figure



In above image, we have two generator G that generates Y from image X and generator F that regenerates X from Y and vice versa. While to see if X and Y are real we have two discriminators Dx and Dy.

For each generator we have three types of losses.

- Cycle consistency loss
- Identity loss
- Adversarial loss

As we do not have a pair of data so to enforce that the network learns correct mapping we use cycle consistency loss. Let's consider the following image:

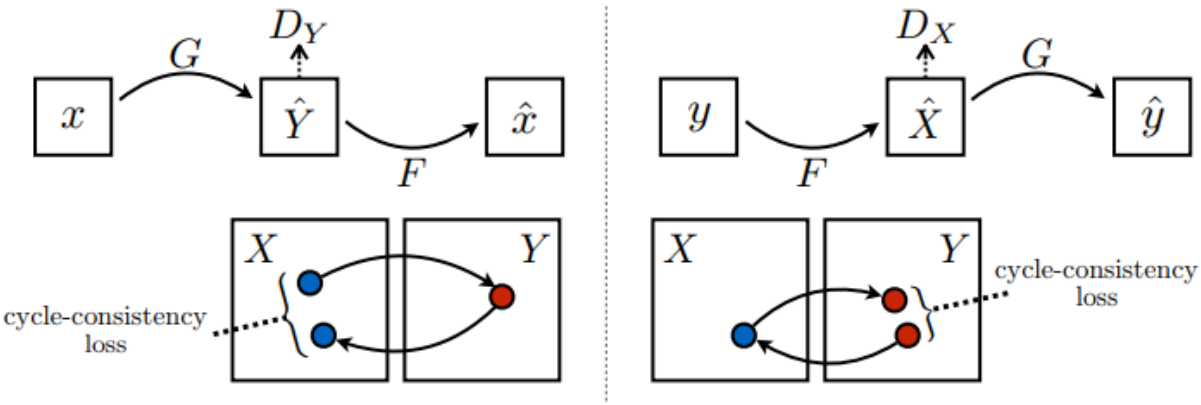


Image taken from [2]

First, we have forward consistency in which X is regenerated in cycle using G and F generators. Similarly, for backward cycle consistency, y is regenerated using F and G generators. Then for both X estimate and Y estimate. We take mean square errors to compute cycle consistency loss.

$$L_{cycle}(G,F) = || F\big(G(x)\big) - x ||_1$$

$$L_{cycle}(G,F) = || \mathrm{G}\big(\mathrm{F}(\mathrm{y})\big) - \mathrm{y} ||_1$$

```
# Generator cycle loss
cycle_loss_G = self.cycle_loss_fn(real_y, cycled_y) * self.lambda_cycle
cycle_loss_F = self.cycle_loss_fn(real_x, cycled_x) * self.lambda_cycle
```

Where lambda is multiplied to give weight to this loss.

Now, let's suppose generator G produce Y from X. If we pass Y into generator G it should generate Y.

$$\text{Identiy loss G} = |G(Y) - Y|$$

Similarly, if we pass X into generator F it should generate X which gives us identity loss for F.

$$\text{Identiy loss F} = |F(X) - X|$$

```
# Generator identity loss
id_loss_G = (
    self.identity_loss_fn(real_y, same_y)
    * self.lambda_cycle
    * self.lambda_identity
)
id_loss_F = (
    self.identity_loss_fn(real_x, same_x)
    * self.lambda_cycle
    * self.lambda_identity
)
```

Thirdly, we compute adversarial loss for the generator by comparing the contents of X and Y as we want to ensure both have similar content.

```
# Define the loss function for the generators
def generator_loss_fn(fake):
    fake_loss = adv_loss_fn(tf.ones_like(fake), fake)
    return fake_loss


# Define the loss function for the discriminators
def discriminator_loss_fn(real, fake):
    real_loss = adv_loss_fn(tf.ones_like(real), real)
    fake_loss = adv_loss_fn(tf.zeros_like(fake), fake)
    return (real_loss + fake_loss) * 0.5
```

Lastly, we add all these terms to get the final generator loss.

$$G_{loss} = \lambda_{cc} L_{cycle} + \lambda_I * L_{identity} + \lambda_c * L_{\text{GAN}}$$

In the above equation lambda is applied to weight losses. Although in given code identity loss is multiplied by both λ_{cc} and λ_I. But at the end values are chosen such that they are in proposed paper (10 for cc loss and 5 for Identity loss).

```
# Total generator loss
total_loss_G = gen_G_loss + cycle_loss_G + id_loss_G
total_loss_F = gen_F_loss + cycle_loss_F + id_loss_F
```

Lambda was already applied in code previously so here in code it's not shown.

The discriminator loss is like previous GANs. It computes mean square error for both real and fake images and then take average of both. It wants to discriminate real and fake images. For cycle GAN we have two discriminators Dx and Dy which discriminates real and fake images of X and Y respectively.

2. Explain the training process, focusing on the relationship between generators and discriminators.

Let's suppose we have two generators, G that translates an image from domain X (e.g., horses) to domain Y (e.g., zebras) and G that translates an image from domain Y back to domain X. Both generators are trained to fool the discriminators while maintaining image content.

For two generators we have two discriminators Dx that distinguish real images in domain X from fake images generated by F and Dy that distinguishes real images in domain Y from fake images generated by G. They are trained to classify real and fake images.

For training first we train the both discriminators such that it classify real images as 1 and fake images as 0. In code the following loss functions are used.

$$L_{D_X} = \frac{1}{2}\left(MSE(D_X(x), 1) + MSE\big(D_X\big(G(y)\big), 0\big)\right)$$

$$L_{D_Y} = \frac{1}{2}\left(MSE(D_Y(y), 1) + MSE\big(D_Y\big(F(x)\big), 0\big)\right)$$

Now we train generators G and F as we have already explained the three types of losses that we used in generators in previous part. So I am just mentioning the loss equations that we used in training:

First we compute the adversarial loss from the following equation

$$L_{\text{GAN}}(G) = \text{MSE}\big(D_Y\big(G(x)\big), 1\big)$$

Then we compute the forward and backward cycle loss using L1 norm as follows:

$$L_{cycle}(G, F) = || \, F\big(G(x)\big) - x \, ||_1$$

Now we compute the identity loss fo both F and G as follows:

$$L_{identity}(G) = || \, G(y) - y \, ||_1$$

To compute the final generator loss, we add all losses with multiplied weight to it.

$$G_{loss} = \lambda_{cc} L_{cycle} + \lambda_I * L_{identity} + \lambda_c * L_{\text{GAN}}$$

Apart from this, the algorithm is similar, generators try to fool the discriminators and discriminator try to correctly classify real and fake images.

3. Augment the code to monitor each component of the loss function and plot the loss function through training. Which loss term(s) dominate? Are there general trends? If so can you justify why?

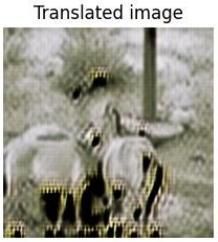
After epoch 1

Input image

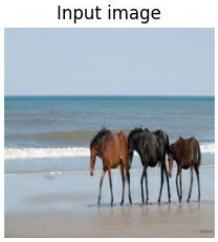
Translated image

Input image

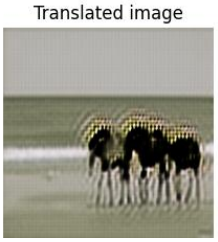
Translated image



Input image



Translated image



After 6 epoch:

Input image

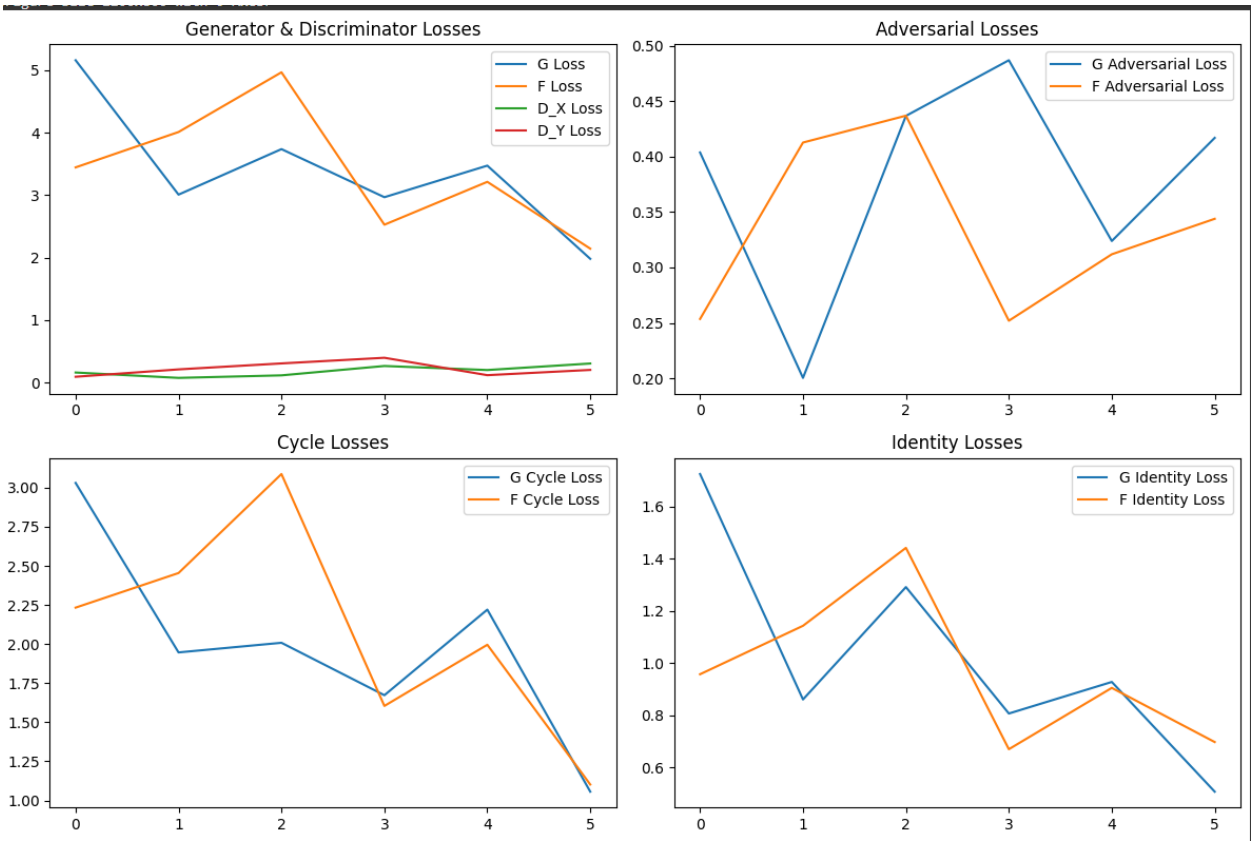
Translated image

Input image

Translated image

Input image

Translated image



Initially, the cycle consistency loss dominates both the identity loss and adversarial loss, indicating that the model prioritizes reconstructing the original images when enforcing cycle consistency. However, after six epochs, the cycle loss starts decreasing significantly, suggesting that the generators are improving in maintaining consistency between domains.

The identity loss for both G and F started high but is significantly decreased after 6 epoch. This suggests that the model is learning to map images closer to their original forms but still retains some transformation artifacts.

The adversarial loss exhibits instability, fluctuating throughout training, which shows that the generators and discriminators are still competing and have not yet fully converged. This is expected, as adversarial training involves dynamic adjustments between the two networks.

Regarding the overall loss trends, the generator loss is initially much higher than the discriminator loss, indicating that the generators are struggling to produce realistic images. However, as training progresses, the generator loss decreases from 5 to 2, approaching the discriminator loss. This suggests that the model is improving in generating realistic images while maintaining balance between the generators and discriminators.

Due to computational constraints, training was limited to six epochs, meaning that full convergence was not observed. However, within this range, the cycle loss remains the most

dominant term. Further training would be needed to analyze long-term trends and determine if the adversarial loss stabilizes.

4. (**Graduate students only**) Modify the loss function by dropping a term (your choice). Before running the code, predict what effect it will have on performance (consider both sides of the cycle). Run the code, plot the loss functions throughout training, and comment on the trends and final results.

I removed identity loss in the computation of generator loss for both F and G. As we use identity loss to preserve the style of the input image. So, we can predict that now the image style will not be preserved.

```
total_loss_G = gen_G_loss +  cycle_loss_G
total_loss_F =  gen_F_loss + cycle_loss_F
```

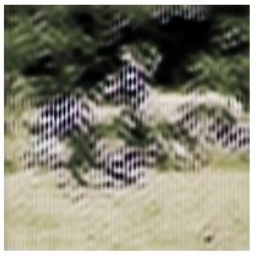
After epoch 1:

Input image

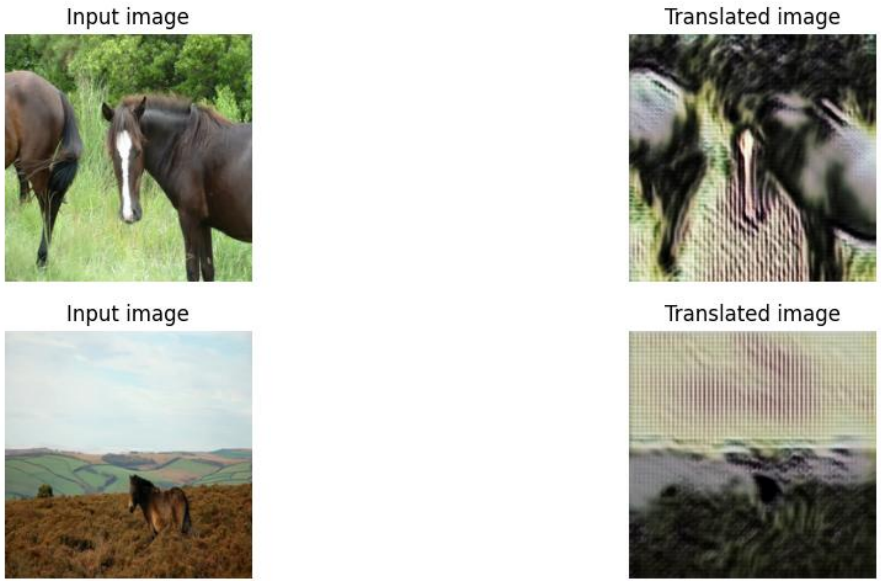
Translated image

Input image

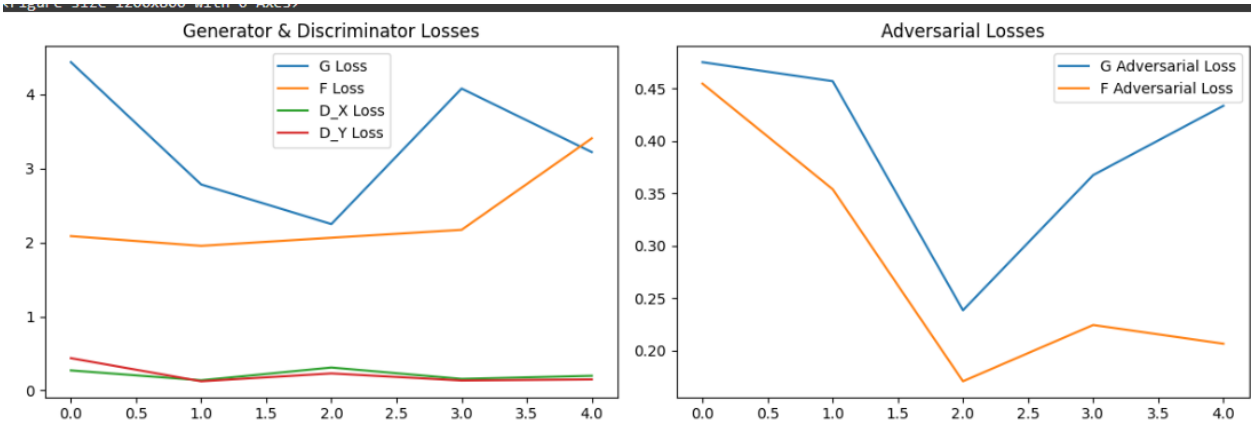
Translated image

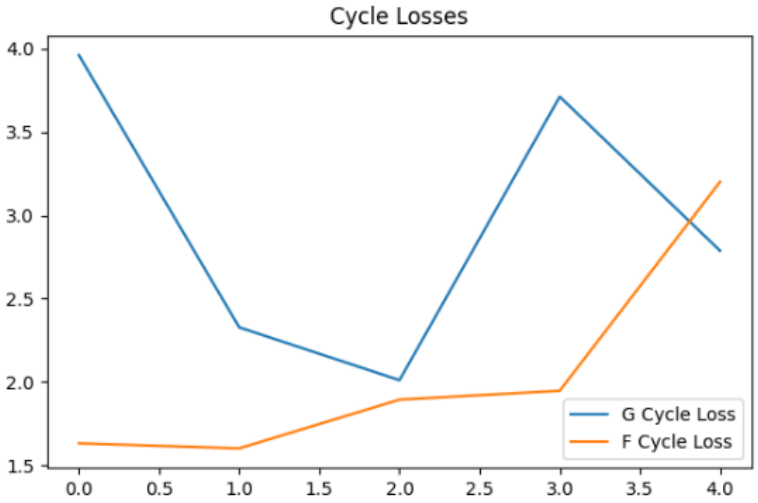


After epoch 5:



We can see that slightly, its losing the style of input image due to removal of identity loss.



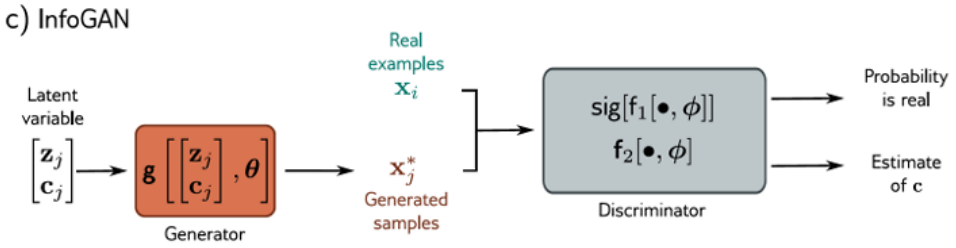


As now the generator loss relies much on cycle loss so, the adversarial loss becomes a bit unstable and is unable to fool the discriminator. However, we cannot conclude anything until we run a few more epoch which I couldn't do due to less time and computational power.

Additional Discussion Questions

1. In class we discussed InfoGAN and what it means to randomly generate inputs and attributes. For the sake of solidifying our understanding, write a short (300 word max) explanation of why InfoGAN should/could work to give meaningful information about sample features. You may want to refer to Prince or the original paper for insight.

In a vanilla conditional GAN, we explicitly pass a known attribute vector into the generator and discriminator. However, InfoGAN can learn these attributes in an unsupervised manner by predicting the latent code rather than relying on labeled data. This makes it particularly useful when labels are unavailable.



The key idea behind InfoGAN is to maximize the mutual information between the latent code c and the generated output $G(z, c)$. This ensures that each dimension of c corresponds to a distinct, interpretable feature in the generated data. For example, in a dataset of handwritten digits, one latent dimension might control digit rotation, another stroke thickness, and another the digit itself.

To achieve this, InfoGAN introduces an auxiliary network $Q(c \mid x)$, which predicts c from the generated sample x. The network is trained to maximize the mutual information $I(c; G(z, c))$, preventing the generator from ignoring c. Without this constraint, the generator might map different values of c to the same outputs, making the latent codes meaningless.

Compared to standard GANs, InfoGAN is more structured and interpretable. Unlike vanilla GANs, where latent space representations are often entangled and difficult to control, InfoGAN enforces meaningful variation in the generated samples.

By learning latent factors without explicit supervision, InfoGAN enables better control over generative models, making it a powerful approach for analyzing and synthesizing complex data distributions.

References:

[1] https://medium.com/analytics-vidhya/conditional-generative-adversarial-networks-f8f1ce025c5d

[2] https://www.tensorflow.org/tutorials/generative/cyclegan