COMP - 6731
Pattern Recognition

Instructor: Dr. Adam Krzyżak

Team : (9)

Project Title : Unpaired Image-to-Image Translation using Cycle-consistent Adversarial Networks

Team Members

Student Id	**Name**
40071883	Keyurbhai Hasmukhbhai Patel
40071095	Manan Prajapati
40071098	Khyatibahen Chaudhary
40091993	Himen Sidhpura
40081593	Dhvani Agola
40092777	Yash Seth
40092281	Jenny Mistry
40086580	Divyansh Thakar

Contents

1 Abstract

This report will explain Cycle Consistent Generative Adversarial Network in the context of the Image-to-Image Translation problem. The CycleGAN is one of the artificial neural networks which is used to execute translations between different images. We have made the idea of CycleGAN more clear and also include the result of the Pokemon2Pokemon. We start with the basic idea of the GAN and eventually made some changes as we found the drawbacks of the used method. Then we will explain Mode Collapse and Cyclic Loss as a technique to overcome this problem.

2 Introduction

Nowadays, Generative Adversarial Network is very popular for its versatility and astonishing results. They have been used in several applications such as text-to-image, image-to-image translation and many more.

This report will focus on one of the types of GAN, which is a cycle-consistent generative adversarial network (CycleGAN). CycleGAN is designed architecture that involves the simultaneous training of two generator models and two discriminator models to perform unpaired image-to-image translation. A discriminative model D that tries to classify if the output image is real or fake, and a generative model G that captures the data distribution. Two neural networks are contesting with each other in a game.

For instance, Consider two collections of Photographs, and they are unpaired, i.e. different locations and different times. The first collection of winter images and another is of summer images from different locations. To implement this concept, we use the architecture of two GANs, and each GAN has a discriminator and a generator model, meaning there are four models in total in the architecture. Now, GAN-1 will translate photos of summer to winter and GAN-2 will translate photos of winter to summer. In this project, we will try to replicate the results of CycleGAN.

3 Methods

This section includes the detailed description of CycleGAN used to translate Image without using paired training data, and the following sub-section narrates the generator and discriminator of CycleGAN and loss function. The CycleGAN was built using PyTorch, so the data was encapsulated in tensors. The Images used were in RGB channels with the dimension of 256 X 256.

3.1 Generative Adversarial Network

Generative Adversarial Network allows specifying the high-level goal. For example, "make the generated images indistinguishable from real images." GANs simultaneously learn two models: a discriminative model D that tries to classify if the output image is real or fake, and a generative model G that captures the data distribution. Two neural networks are contesting with each other in a game resulting in zero-sum. Yann LeCun described GANs as "the coolest idea in machine learning in the last twenty years."

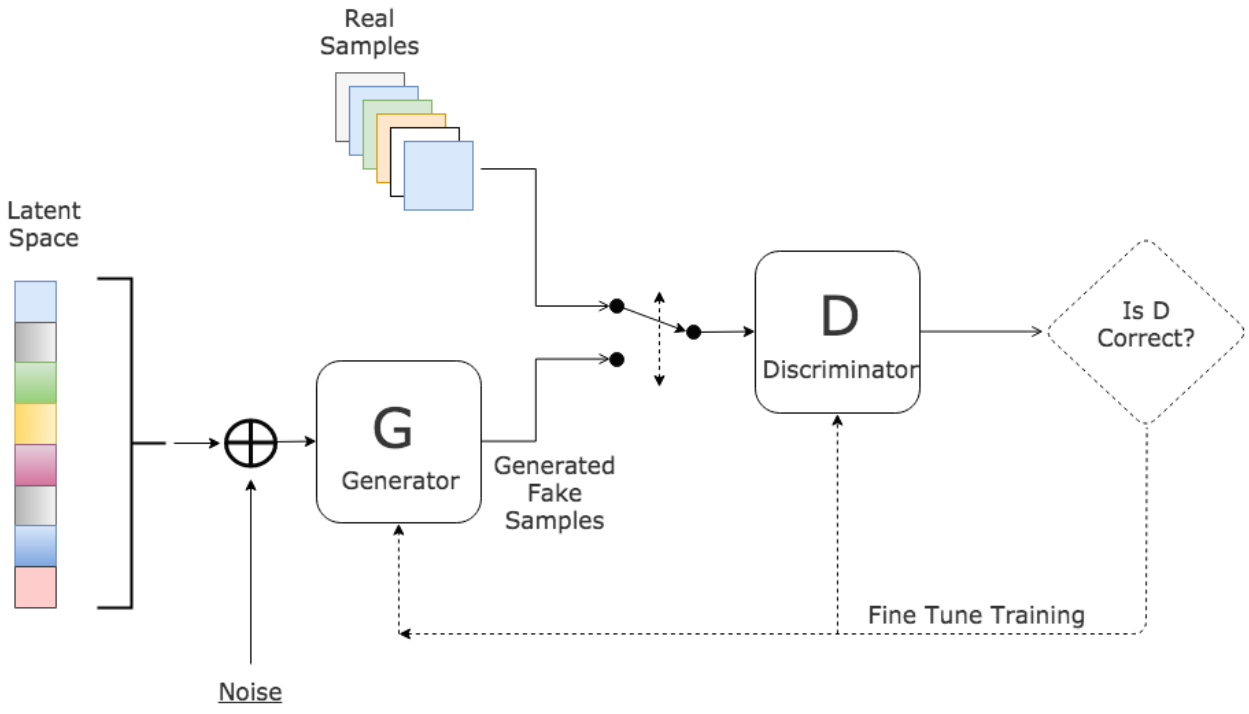

Figure 1: Architecture of Generative Adversarial Network

3.1.1 Mathematical Intution Behind GANs

We want to learn a generator's distribution p_g over data x:

$$x = G(\boldsymbol{z}; \theta_g) \tag{1}$$

where p_z is distribution over noise input z. G is represented by multilayer perceptron with parameters θ_g.

Similarly, We define another multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ that represents the probability that x is from the real data, not p_g.

Given a real data, we want D to maximize $\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]$. Meanwhile, given a fake sample $G(z)$, we want D to maximize $\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ where $D(G(\boldsymbol{z}))$ is the discriminator's estimate of the probability that a fake instance is real. However, G is trained to minimize $\log(1 - D(G(\boldsymbol{z})))$ simultaneously.

In short, G and D are playing zero-sum game with following loss function:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2}$$

3.2 CycleGAN

A lot of recent studies provided state-of-the-art results for image-to-image translation problems under a supervised setting, where training data provides the mapping between input and output images.

However, In real life, paired training data is difficult to obtain. For object transfiguration (e.g., zebra↔horse), the desired output is not even well-defined.

CycleGAN is a type of GAN which is aimed at solving this problem. CycleGAN can translate images from domain X to domain Y under an unsupervised setting where no information is available as to which input image matches which output Image.

3.2.1 CycleGAN Formulation

Our goal is to learn mapping functions between two domains X and Y given training samples $\{x_i\}_{i=1}^N$ where $x_i \in X$ and $\{y_j\}_{j=1}^M$ where $y_j \in Y$. We denote the data distribution as $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$.

CycleGAN consists of two generators:

$$G : X \to Y$$
$$F : Y \to X$$

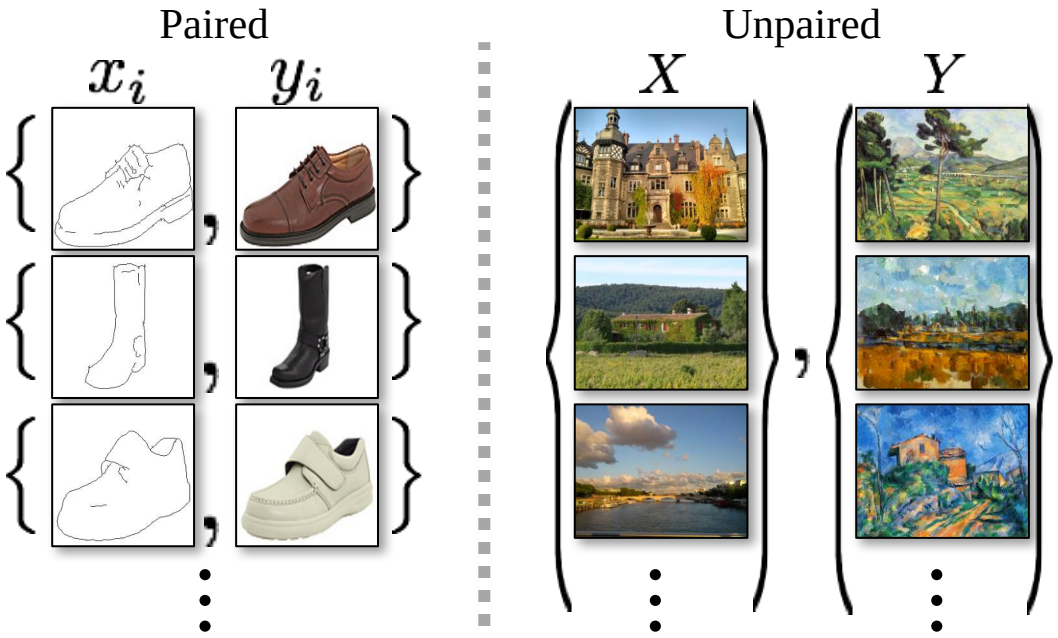

Figure 2: Paired Training Data vs Unpaired Training Data

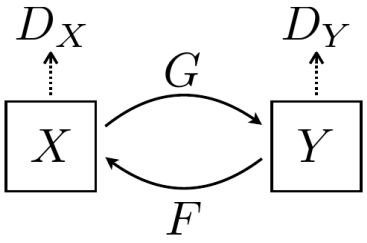

Figure 3: CycleGAN consists of two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$

Along with, two adversarial discriminators:

$$D_X : X \rightarrow \mathbb{R}$$
$$D_Y : Y \rightarrow \mathbb{R}$$

D_X distinguishes between images from domain X and fake images generated by $F(y)$; in the same way, D_Y aims to discriminate between images from domain Y and fake images generated by $G(x)$

For the mapping function $G : X \rightarrow Y$ and its Discriminator D_Y, we can write adversarial

loss as:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \tag{3}$$

where G will try to generate images indistinguishable from Y, on the other hand D_y learns to distinguish between fake and real images.

Mode Collapse is a widespread problem with GANs. GANs do **not** force output to correspond to input. Given an input, we want to generate output that matches the input. Although, If there is an image that can minimize loss for Generator, then it merely maps any given input to the same output image. Adversarial Loss alone is not enough to overcome this problem. To overcome this problem, Cycle Consistency Loss is added to the final loss function.

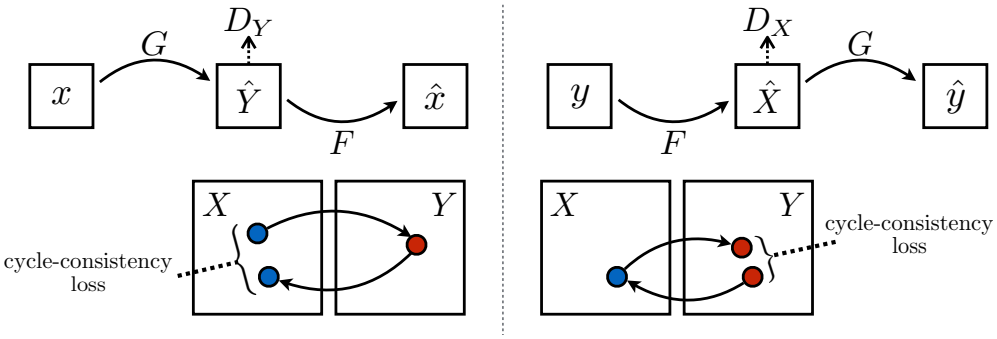

Figure 4: Cyclic Loss

Here, F maps back images generated by G to the original domain X. F should be able to bring back the orignal image. i.e., $x \to G(x) \to F(G(x)) \approx x$.

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1], \tag{4}$$

So, the Final Loss Function can be written as:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F), \tag{5}$$

4 Experimental Details

We have implemented CycleGAN using the PyTorch framework. We have provided source code along with this report.

4.1 CycleGAN Architecture

CycleGAN consists of only two components named Generator and Discriminator.

4.1.1 Generator

The Generator takes Image as an input, which has a dimension of $256 * 256 * 3$. Here 3 represents a number of channels in our case RGB channels.

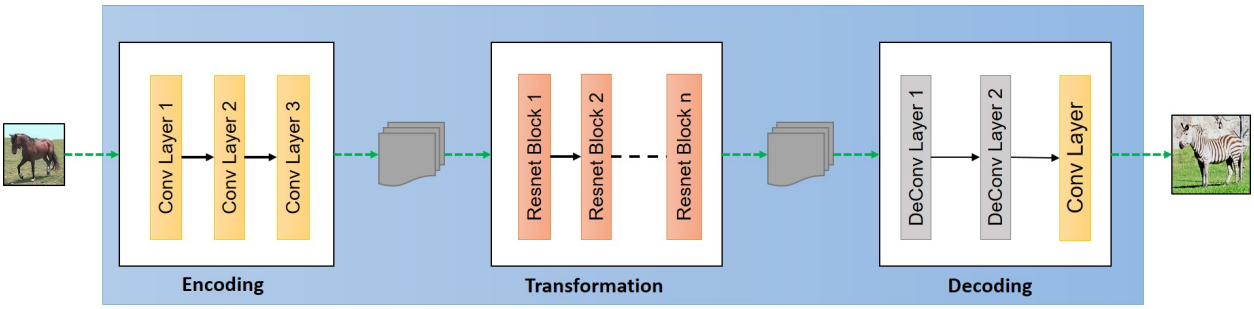

Figure 5: Generator Architecture

The Generator has three phases.

- Downsampling (Encoding)
- Transformation
- Upsampling (Decoder)

4.1.1.1 Downsampling

The task of Encoder is to downscale Image to $1 * 1 * 256$ latent space. It starts with 64 filters and doubles the number of filters with every layer up to 256 filters. The detailed architecture of this phase is provided below.

```
(0): ReflectionPad2d((3, 3, 3, 3))
(1): Conv2d(3, 64, kernel_size=(7, 7), stride=(1, 1))
(2): InstanceNorm2d(64, eps=1e-05, momentum=0.1)
(3): ReLU(inplace=True)
(4): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
```

```
(5): InstanceNorm2d(128, eps=1e-05, momentum=0.1)
(6): ReLU(inplace=True)
(7): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(8): InstanceNorm2d(256, eps=1e-05, momentum=0.1)
(9): ReLU(inplace=True)
```

Here, Encoder consist 3 convolution layers with *kernel_size* and *stride* as specified. Each convolution layer is followed by the *ReLU* activation function.

4.1.1.2 Tranfomation

The resulting activation from Encoder is passed to the transformer, which composed of 9 Residual Blocks, which consists of two convolution layers where the residue is being added to the output from input. This part connects the Encoder and decoder, allowing a model to learn the effective transformation of one domain to another domain through residual functions while maintaining less deviation of output from original input helping us to minimize cyclic loss.

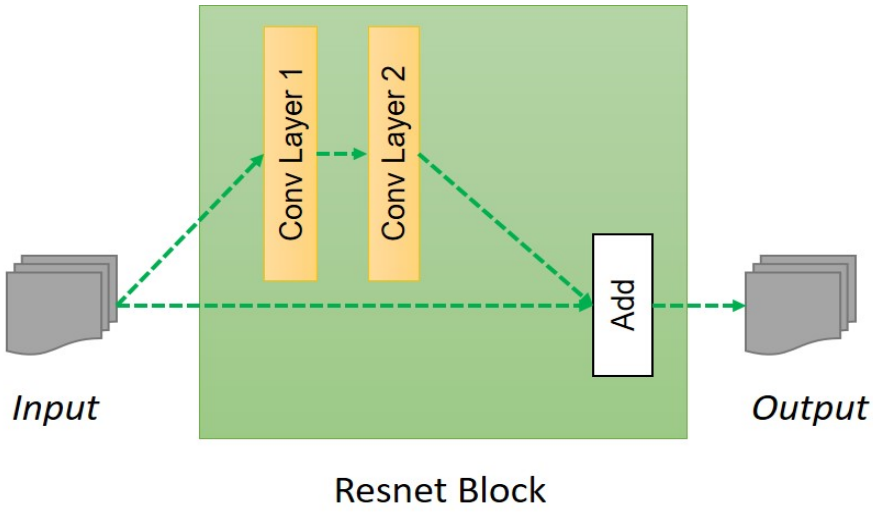

Figure 6: Residual Block Architecture

4.1.1.3 Upsampling

In this phase, the latent space is upscaled back to the original dimensions by applying transposed convolution. It starts with 256 filters and halving the number of filters with every layer up to 64 filters resulting in a $256 * 256 * 3$ image. The detailed architecture of this phase is provided below.

```
(0): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
```

```
(1): InstanceNorm2d(128, eps=1e-05, momentum=0.1)
(2): ReLU(inplace=True)
(3): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
(4): InstanceNorm2d(64, eps=1e-05, momentum=0.1)
(5): ReLU(inplace=True)
(6): ReflectionPad2d((3, 3, 3, 3))
(7): Conv2d(64, 3, kernel_size=(7, 7), stride=(1, 1))
(8): Tanh()
```

4.2 Discriminator

Discriminator will take Image as an input and predicts the probability of it being real. The detailed architecture of Discriminator is provided below.

Discriminator uses four convolution layers to extract features from the input image. One final convolution layer will produce 1-dimensional output.

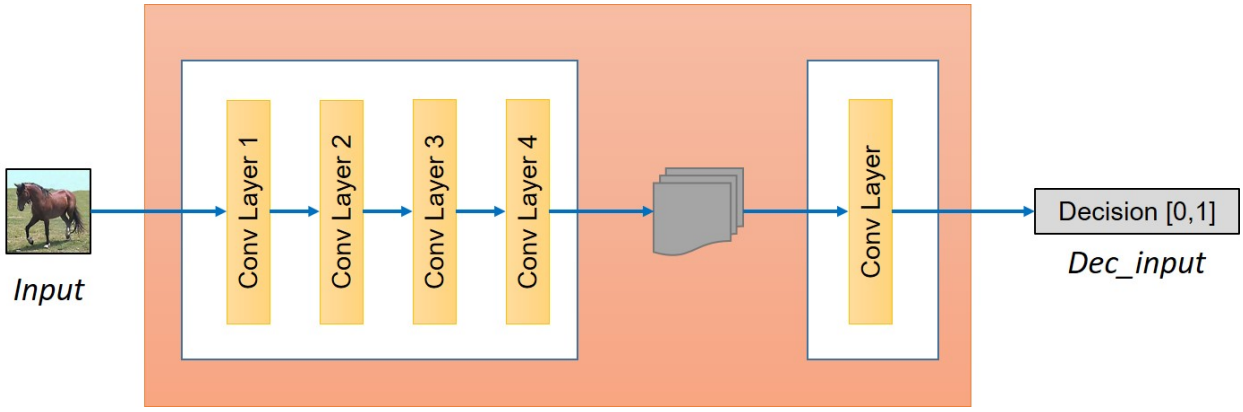

Figure 7: Discriminator Architecture

4.3 Datasets

We have trained CycleGAN to generate Pokemon as different elemental types. We have obtained dataset from https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types. This dataset contains 809 images of Pokemon in $256 * 256$. To categorize them by elemental types, we used script provided here.

4.4 Training Details

Authors of paper trained CycleGAN for 200 epochs. Due to limited resources, we have trained for only 50 epochs. We use StepLR from PyTorch to decay the learning rate. We

start with 0.0002 as the learning rate for the first 100 epochs and decay learning rate to zero for the next 100 epochs. Weights are randomly initialized from Normal Distribution with mean of 0 and variance of 0.02.

Table 1: Hyperparameters

Number of Epochs	50
β_1	0.5
β_2	0.999
λ_{cyc}	10
ImagePool Size	50
Input Image Size	$256 * 266$
Resized Image Size	$1.12 * InputImageSize$
Cropped Image Size	$256 * 256$

5 Results and Discussions

We have applied CycleGAN to Pokemon Transfer. These images are selected as relatively successful results. In the figure 8, we show results on style transfer between different elemental types of pokemon, trained on 114 images from the *water* pokemon class and 78 images from the *grass* pokemon class for just 52 epochs.

Figure 8: Pokemon2Pokemon

Authors trained for 200 epochs on various datasets. Even though we were only able to train for 50 epochs on the Pokemon2Pokemon dataset, we obtained relatively successful results on mapping pokemon to different elemental types. We find that CycleGAN is very sensitive to hyperparameters. You may have to train multiple times to obtain the best results possible.

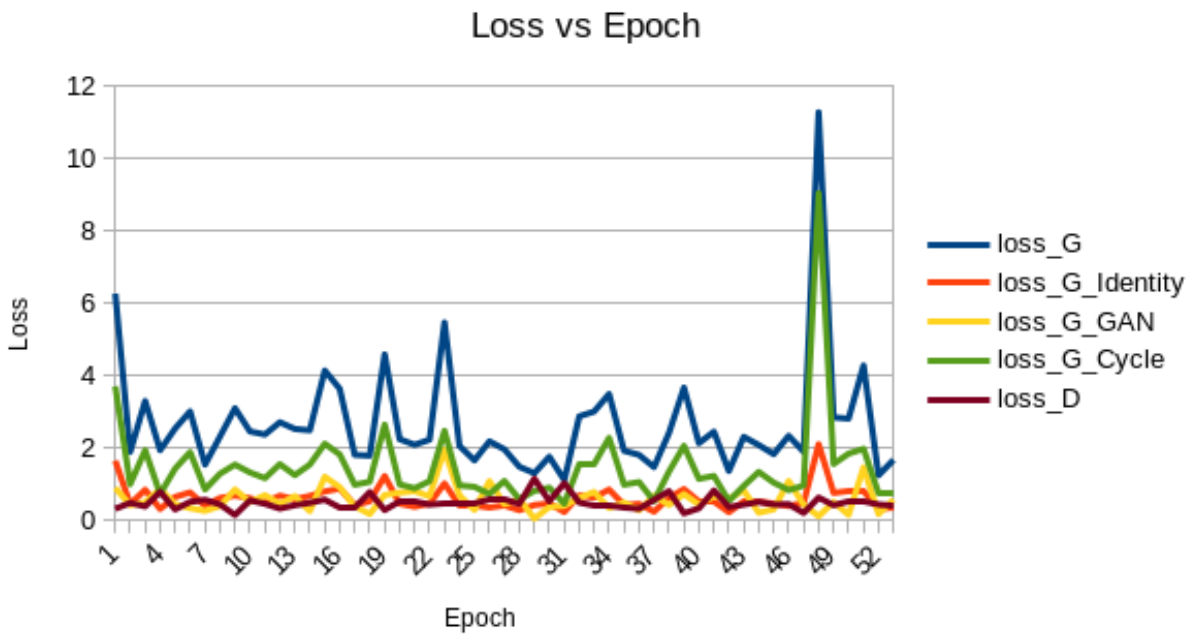

Figure 9: CycleGAN Loss : Pokemon2Pokemon ($Water <-> Grass$)

Figure 10: Season Transfer : Summer2Winter trained on winter and summer photos of Yosemite from Flickr

6 Conclusion

In this report, we have presented the results of the replication of CycleGAN on Pokemon2Pokemon transfer. We found that CycleGAN performs really well despite smaller datasets. Best results are obtained when the input and output domain only involves colour and texture change. Sometimes, We have to adjust hyperparameters to obtain relatively successful results.

References

[1] https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d.

[2] https://hardikbansal.github.io/CycleGANBlog/.

[3] https://machinelearningmastery.com/cyclegan-tutorial-with-keras/.

[4] https://github.com/rileynwong/sort-pokemon-images-by-type.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[7] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.