Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Harshad Rai
University of Illinois
Urbana Champaign
hrrai2@illinois.edu

Naman Shukla
University of Illinois
Urbana Champaign
namans2@illinois.edu

Ziyu Zhou
University of Illinois
Urbana Champaign
ziyuz2@illinois.edu

Abstract

The architecture introduced in this paper learns a mapping function $G : X \mapsto Y$ using an adversarial loss such that $G(X)$ cannot be distinguished from Y, where X and Y are images belonging to two separate domains. The algorithm also learns an inverse mapping function $F : Y \mapsto X$ using a cycle consistency loss such that $F(G(X))$ is indistinguishable from X. Thus, the architecture contains two Generators and two Discriminators. However, the major aspect in which this implementation truly shines is that it does not require the X and Y pairs to exist, i.e. image pairs are not needed to train this model. This is highly beneficial as such pairs are not necessarily always available or tend to be expensive monetarily. An application of this could be used in movies, where, if a movie crew was unable to shoot a scene at a particular location during the summer season and it is now winter, the movie crew can now shoot the scene and use this algorithm to generate scenes which look like they were shot during the summer. Other areas in which this algorithm can be applied include image enhancement, image generation from sketches or paintings, object transfiguration, etc. The algorithm proves to be superior to several prior methods.

1. Introduction

An image style transfer is a method to translate an image from one style or design to another keeping all the image characteristics intact. For example, transforming an image of a painting from Van Gogh [16] collection of paintings to an image of a Monet painting such that it is indistinguishable to the set of Monet painting collections. Changing textures of the objects within the painting also falls under the same style translation. In this report, we present one such method that can learn to do image to image style transfer : capturing special characteristics of a set of images and translating them to a different set of image collections, all in the absence of any pairing between the two image sets and with no human intervention.

Many years of research in pattern recognition, image processing and computer vision have produced algorithms that can translate images from one set to another [3] [7] [12]. Most of the processes require supervision. Additionally, obtaining paired training data can be difficult and expensive. Hence, this method proposes an alternative to using paired training data by assuming an existence of some underlying relationships between the domains - for example, there are two different renderings of the same underlying scene - and seek to learn that relationship. Additionally, this proposed method also asserts a "cycle consistency" between the translation from one image style to another and back. This makes sure that there exists only a single translation and not infinitely many maps to the same output domain.

Recently, algorithms have been introduced for image style transfer using generative models like Generative Adversarial Networks [4] (GAN) and Variational Auto Encoders [17] (VAE). GANs have been more popular due to their better perceptual results on various datasets as compared to algorithms based on likelihood maximization. The method proposed by Jun-Yan [19] is a network of GANs with additional cyclic consistency. The idea is that the optimal generator G will translate the image from domain X to a domain $\hat{Y}$ distributed identically to Y with an additional guarantee of recovering the original image X when an inverse mapping function F is applied on the generated image.

We implemented and applied the unpaired image to image translation with cycle constancy loss on five different datasets [1]: *horse2zebra*, *apple2orange*, *summer2winter*, *vangogh2photo* and *monet2photo* (Figure 1 presents several outputs from our implementation). We also performed detailed analysis of the performance and presented some of the unexpected as well as astounding results. Our code is available at `https://github.com/CycleGANS/CS543CycleGANsProject` and detailed blog and re-

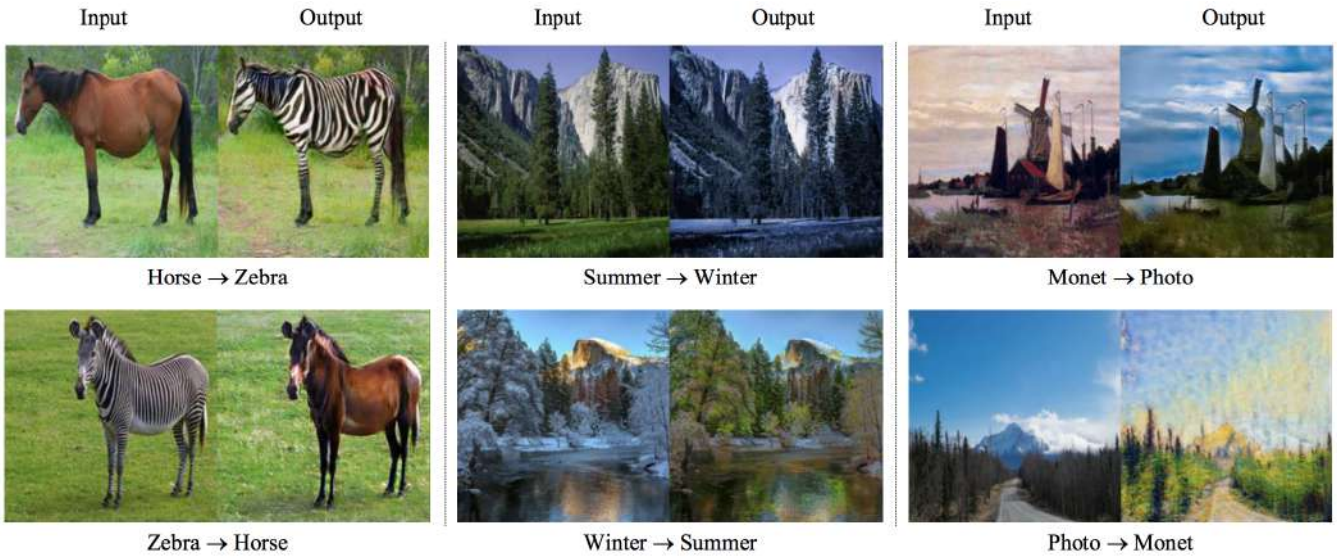

Figure 1. Our implementation of the CycleGAN algorithm introduced in paper [19] is able to perform unpaired image-to-image translation successfully.

sults snaps are available at `https://cyclegans.github.io`.

2. Details of Approach

In order to understand cycle GAN [19], it is important that we understand GANs [4] (Generative Adversarial Networks) first. GANs were introduced by Ian Goodfellow et. al. in their 2014 paper titled "Generative Adversarial Nets".

2.1. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are feedforward neural networks that create images from random noise that approximate real images.This is done using two neural networks: a generator and a discriminator.

Both, the generator and the discriminator, are multilayer perceptrons. An architecture that uses both of them is referred to as *Adversarial Nets*. Both the models are trained using backpropagation & dropout algorithms [18] and samples are obtained from the generator only using forward propagation. Adversarial nets is most straightforward to apply when both models are multilayer perceptrons.

To formulate this, let the prior on input noise variables be denoted by $p_z(z)$ and generator's distribution over data be p_g. Let generator, the multilayer perceptron with parameters θ_g, be the mapping function denoted by $G(z;\theta_g)$. Similarly, second multilayer perceptron (discriminator) with parameters θ_d, that outputs a single scalar, be denoted by $D(x,\theta_d)$. Also, the probability that x came from data rather than p_g be $D(x)$. The task for the Generator network is to approximate a function $G(z;\theta_g)$ that maps random noise to a range whose probability distribution p_g is the same as the probability distribution of the real data x. While the discriminator is tasked with differentiating bteween the images coming out of the Generator and real data.

Training

- D is trained to maximize the probability of assigning the correct label to both: training examples and samples from G
- Simultaneously, G is trained to $Minimize \log(1 - D(G(z)))$

Since, $D(G(z))$ is the discriminator's probability of classifying the fake image $G(z)$ as true image. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(z)))$. In other words, D and G play the following two-player minimax game with value function $V(G,D)$:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log(D(x))] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

In practice, the implementation is carried out in an iterative manner to avoid over-fitting and computational prohibition of optimizing D to completion in the inner loop of training. Instead, D and G are optimized alternately with k optimization steps of D followed by one optimization step of G. This allows D to be maintained near its optimal solution as long as G changes slowly.

The theory of this paper states that as long as D and G have enough capacity, p_g converges to p_{data} i.e. the original image space.

2.2. Cycle GANs

Cycle GANs was introduced by Jun-Yan Zhu et. al. in their 2017 paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [19]. They also have an amazing website `https://junyanz.github.io/CycleGAN/` that provides examples of their outputs, news articles and links to the implementation of their algorithm in different programming languages.

The goal of this implementation is to learn mapping functions between 2 domains $X \mapsto Y$ and vise versa. Considering the training examples from one domain $\{x_i\}_{i=1}^N$ where $x_i \in X$ distributed as $x \sim p_{data}(x)$ and similarly, other domain $\{y_i\}_{i=1}^N$ where $y_j \in Y$ distributed as $y \sim p_{data}(y)$. The two generators work as mapping functions : $G : X \mapsto Y$ and $F : Y \mapsto X$. The two discriminators D_X & D_Y work as classifiers aiming to distinguish between images $\{x\}$ & $\{y\}$ and translated images $F(y)$ & $G(x)$ respectively.

2.2.1 Objective

The Objective of this architecture contains two kinds of losses: One for matching the distribution of generated images to the data distribution in the target domain, called as *Adversarial Loss*. The other is to prevent the learned mappings G and F from contradicting each other, called as *Cycle Consistency Loss*.

Adversarial Losses:

Adversarial losses need to be applied to both mapping functions. Where generator G tries to generate images that look similar to images from domain Y and discriminator D_Y aims to distinguish between translated samples $G(x)$ and real samples y. Hence, D_Y tries to maximize and G tries to minimize the below loss.

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}[\log(D_Y(y))] + E_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))]$$

Similarly, the second adversarial loss for mapping function $F : Y \mapsto X$ and it's discriminator D_X, the generator F tries to minimize and discriminator D_X tries to maximize the below loss.

$$L_{GAN}(F, D_X, X, Y) = E_{x \sim p_{data}(x)}[\log(D_X(x))] + E_{y \sim p_{data}(y)}[\log(1 - D_X(F(y)))]$$

Cycle Consistency Loss

In theory, adversarial training learns stochastic mapping functions G and F that produce outputs that are identically distributed as their target domains Y and X. However, with large enough capacity, these functions can map the same set of images to any random permutation of images in the target domain which have the same distribution as the target distribution. This basically means that using only the adversarial losses trains the generators to generate images that look like images from the target set but may not have the same structure as the input images.

Thus, forward cycle consistency and backward cycle consistency are needed as given below respectively,

$$x \mapsto G(x) \mapsto F(G(x)) \approx x$$

$$y \mapsto F(y) \mapsto G(F(y)) \approx y$$

Incentivizing this behavior using *cycle consistency loss*

$$L_{Cyc}(G, F) = E_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + E_{y \sim p_{data}(y)}[||G(F(y)) - y||_1]$$

This will result in full objective loss as given below, which we aim to solve by minimizing over each generator i.e. G & F and maximizing over each discriminator i.e. D_X & D_Y.

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{Cyc}(G, F)$$

where λ controls relative importance of the two objectives.

The pseudo code of the above algorithm is shown here 1

Algorithm 1 Cycle GANs

for number of epochs **do**
  for number of batches **do**
    Sample minibatch $\leftarrow \{x^{(i)}\}_{i=1}^m \in X$
    Sample minibatch $\leftarrow \{y^{(j)}\}_{j=1}^m \in Y$
    Generate m samples of $G(x)$ and $F(y)$
    Generate m samples of $F(G(x))$ and $G(F(y))$
      $X \mapsto G(x)$
      $Y \mapsto F(y)$
    Update the Discriminators D_X and D_Y
      $\max_{D_X} L_{GAN}(F, D_X, X, Y)$
      $\max_{D_Y} L_{GAN}(G, D_Y, X, Y)$
    Update the Generators G and F
      $\min_{G,F} L(G, F, D_X, D_Y)$

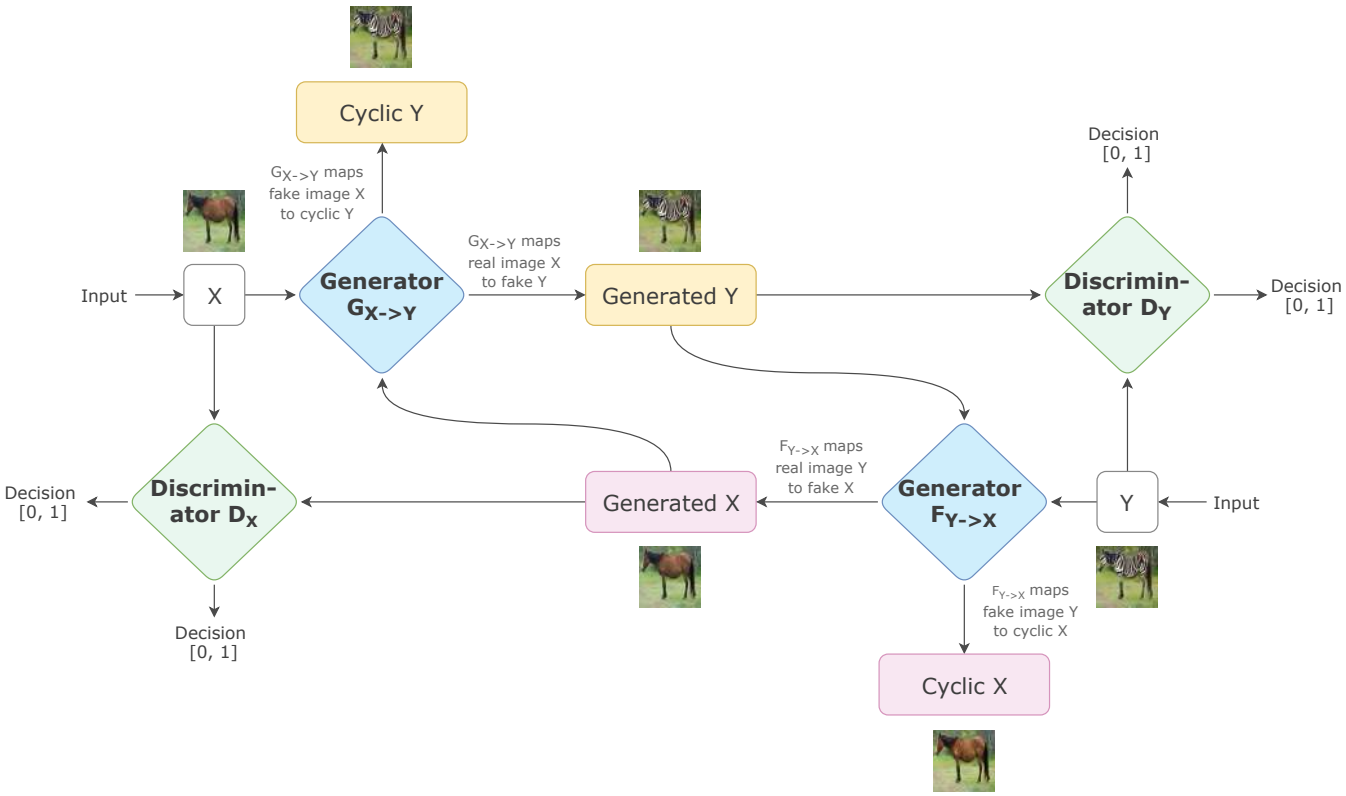

Figure 2. Overall CycleGAN Architecture of Our Implementation

2.3. Our Implementation

Now that we have understood the theory behind Cycle GANs, it is time for us to look at the implementation. Based on the theoretical analysis above, we implemented our CycleGAN architecture as shown in figure 2.

Basically, there are two generators and two discriminators. The input images from set X will be fed into a generator $G_{X \to Y}$ that maps X to Y to generate fake image Y, which will then be passed to the discriminator D_Y to determine whether it's fake or not. This fake image Y will also go to the other generator $F_{Y \to X}$ that maps image from Y to X to generate a cyclic image of X.

The input images from set Y follow similar procedures, except that they will be passed through the generators in the opposite order and a different discriminator.

2.3.1 Generator Network

In the implementation section of this paper, the authors stated that the generator architecture used, was obtained from the paper Justin Johnson et. al.[8]

Justin Johnson et. al. utilize perceptual losses to perform style transforms between two sets of images. They also perform single image super resolution. They claim that their implementation provides similar results when compared to optimization models while being three orders in magnitude faster.

They state that their architecture does not rely only on pixel losses but also on perceptual losses which is the most important, as small translations and rotations of the image give large pixel losses which is not the case when considering perceptual losses. This results in absolutely stunning outputs as shown in their paper.

However, we shall not get into the perceptual losses and their benefits here as that is not our current priority. Our motive to understand their architecture is limited to their generator design which generates the amazing outputs and also does this at a really fast pace.

The generator used in this implementation involves three parts to it: *in-network downsampling*, *several residual blocks* and *in-network upsampling*.

In-Network Downsampling

This part of the generator consists of two Convolution Networks [9], each followed by Spatial Batch Normalization and a ReLu activation function [18].

Each convolution network uses a stride of 2 so that downsampling can occur. The first layer has a kernel size of 9x9 while the second layer has a kernel size of 3x3.

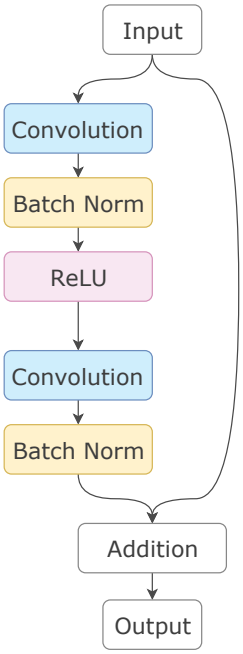

Figure 3. Residual Block

Residual Blocks

The concept of residual blocks was introduced by Kaiming He et. al.[6]. Each residual block consists of two convolution layers. The first convolution layer is followed by batch normalization and ReLu activation. The output is then passed through a second convolution layer followed by batch normalization. The output obtained from this is then added to the original input. The architecture of the residual block is shown in Figure 3.

Each convolution layer in every residual block has a 3x3 filter. The number of residual blocks depend on the size of the input image. For 128x128 images, 6 residual blocks are used and for 256x256 and higher dimensional images, 9 residual blocks are used.

In-network Upsampling

This part consists of two convolution layers. They are fractionally strided with a stride value of $\frac{1}{2}$. The first convolution layer has a kernel size of 3x3 while the last layer has a kernel size of 9x9. The first layer is followed by batch normalization and ReLU activation while the second convolution layer is followed by a scaled Tanh function [2] so that the values can fall between [0,255] as this layer is the output layer.

The entire feedforward generator network starts off with downsampling, followed by residual blocks and ends with upsampling. See figure 4.

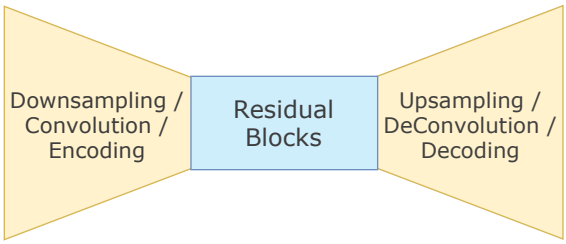

Figure 4. Generator Network

The benefits of using such a network is that it is computationally less expensive compared to the naive implementation and provides large effective receptive fields that lead to high quality style transfers in the output images.

For our implementation, we used least squared losses as the adversarial loss instead of cross entropy as suggested by Mao, Xudong, et al.[14] since cross entropy leads to vanishing gradient issues. We also replaced the first convolution layer having 9x9 filter size by two convolution layers having 7x7 and 3x3 sized filters respectively as this provides the same receptive field size with smaller number of parameters. We did the same with the output layer but with 3x3 followed by 7x7 filter sized convolution layers. Also, [13] was used to get the outputs within the 0-255 range.

The exact details of the layers in the Generator of our implementation is shown in Table 1 below:

Table 1. Generator Layers

Layer Number	Layer Type	Kernel Size	Stride	Input—Output Dimension	Input—Output Channels
1	Conv2d	7	1	256—256	3—64
2	BatchNorm2d	-	-	256—256	64—64
3	ReLU	-	-	256—256	64—64
4	Conv2d	3	2	256—128	64—128
5	BatchNorm2d	-	-	128—128	128—128
6	ReLU	-	-	128—128	128—128
7	Conv2d	3	2	128—64	128—256
8	BatchNorm2d	-	-	64—64	256—256
9	ReLU	-	-	64—64	256—256
10	Conv2d	3	1	64—64	256—256
11	BatchNorm2d	-	-	64—64	256—256
12	ReLU	-	-	64—64	256—256
13	Convd2d	3	1	64—64	256—256
14	BatchNorm2d	-	-	64—64	256—256
15	9+14	-	-	64—64	256—256
16	Conv2d	3	1/2	64—128	256—128
17	BatchNorm2d	-	-	128—128	128—128
18	ReLU	-	-	128—128	128—128
19	Conv2d	3	1/2	128—256	128—64
20	BatchNorm2d	-	-	256—256	64—64
21	ReLU	-	-	256—256	64—64
22	Conv2d	7	1	256—256	64—3
23	TanH	-	-	256—256	3—3

Note: Layers 9 to 15 represent a residual block and are repeated 9 times.

2.3.2 Discriminator Network

The CycleGAN paper uses the architecture of 70×70 PatchGANs introduced by the paper of Isola, Phillip, *et al.* in

2017 [7]. This architecture is also applied in [10] and [12].

The main difference between a PatchGAN and a regular GAN discriminator is that the latter maps an input image to a single scalar output in the range of $[0, 1]$, indicating the probability of the image being real or fake, while PatchGAN provides an array as the output with each entry signifying whether its corresponding patch is real or fake.

According to [7], using a PatchGAN is sufficient because the problem of blurry images caused by failures at high frequencies like edges and details can be alleviated by restricting the GAN discriminator to only model high frequencies, which is what PatchGAN is designed for.

The reason why the CycleGAN paper implements PatchGAN as its discriminator is that it has fewer parameters than a full-image discriminator, and thus runs faster, being able to work on arbitrarily large images. The experimental results also show that it can produce high quality results even with a relatively small patch size.

Due to the limitation of computing resources, we implemented a slightly simpler discriminator network considering the suggestions of [11] and [5] based on the original network in the paper.

The main difference is that instead of using PatchGAN, we applied a general discriminator structure. The other parts of the architecture are similar to the original one. Figure 5 shows our basic idea. Besides the convolution layers in the figure, our discriminator model also supports batch normalization and leaky ReLU layers in between by making them optional when needed.

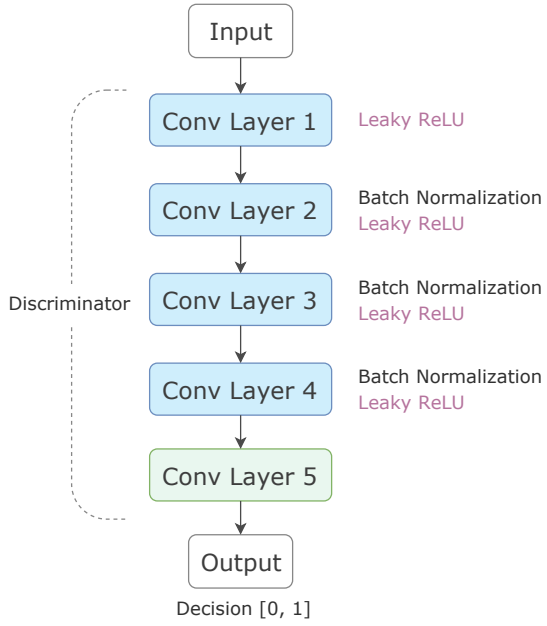

Figure 5. Discriminator Network

Table 2. Discriminator Layers

Layer Number	Layer Type	Kernel Size	Stride	Input—Output Dimension	Input—Output Channels
1	Conv2d	4	2	256—128	3—64
2	LReLU	-	-	256—128	3—64
3	Conv2d	4	2	128—64	64—128
4	BatchNorm2d	-	-	128—64	64—128
5	LReLU	-	-	128—64	64—128
6	Conv2d	4	2	64—32	128—256
7	BatchNorm2d	-	-	64—32	128—256
8	LReLU	-	-	64—32	128—256
9	Conv2d	4	1	32—31	256—512
10	BatchNorm2d	-	-	32—31	256—512
11	LReLU	-	-	32—31	256—512
12	Conv2d	4	1	31—30	512—1

The detailed information of the layers that we use in our final model are in table 2. We decide to use leaky ReLU to allow a small gradient α when the unit is not active, i.e.,

$$f(x) = \alpha x \text{ for } x < 0$$

The value of α is 0.2 in our case.

Note that the output image of the last layer has a size of $30 \times 30 \times 1$, with the value of each pixel representing for the possibility of how likely the corresponding section of the input image is real.

From our tests and results, the architecture is further simplified. However, it is still able to produce descent results (see the following sections for details).

3. Results

The table below displays our implementation choices for running Cycle GANs:

Table 3. Hyperparameters

Name	Epochs	Batch size	Cyclic loss rate λ	Learning rate
Value	200	1	10	0.002

The number of epochs were reduced for monet2photo and vangogh2photo.

Table 4. Dataset Details

Data set	Space	Train Size (A)	Train Size (B)	Test Size (A)	Test Size (B)
horse2zebra	111M	1067	1334	120	140
apple2orange	75M	995	1019	266	248
summer2winter	126M	1231	962	309	238
vangogh2photo	292M	400	6287	400	751
monet2photo	291M	1072	6287	121	751

3.1. Selected Generated Images

We will present some of our best results for datasets *horse2zebra*, *winter2summer* and *monet2photo* here. The entire results of all the datasets that we have tested can be found in the Appendix, and the corresponding discussion and analysis can be found in the following sections.

Figure 6. Horse to Zebra

Figure 7. Zebra to horse

Figure 8. Summer to Winter

Figure 9. Winter to Summer

Figure 10. Monet to Photo

Figure 11. Photo to Monet

3.2. Quantitative Analysis

To monitor the analytics, we have used Tensorboard (3.2) from Tensorflow, Google [15]. Trends in total generator loss and the variation in the two discriminator losses are those attributes that measure model performance.

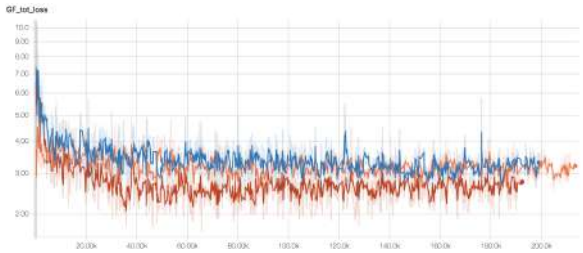

Figure 12. Total Generator loss profile for datasets : *horse2zebra*, *apple2orange* and *summer2winter* - 200 epochs

Figure 12 shows the generator total loss profile across all datasets. The trend is apparent that the loss is reduces till a certain point and then saturates for rest of the iterations. The saturated values in Figure 12 for the respective datasets are directly proportional to their apparent output image quality.

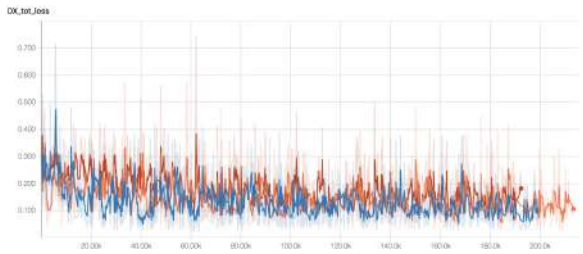

Figure 13. Discriminator (X) loss profile for datasets : *horse2zebra*, *apple2orange* and *summer2winter* - 200 epochs

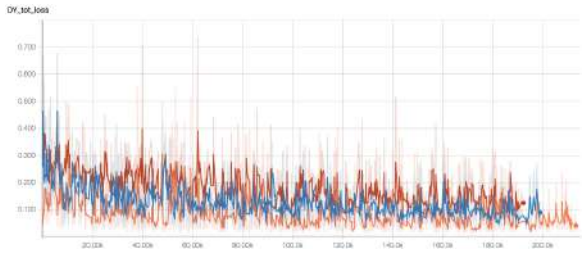

Figure 14. Discriminator (Y) loss profile for datasets : *horse2zebra*, *apple2orange* and *summer2winter* - 200 epochs

The discriminator profiles shown in Figure 13 and Figure 14 are denoised by a smoothing factor of 0.65. The profile suggests that the variance is decreasing as iterations increase. However, the mean value gets saturated around 0.2 instead of 0.5. Hence, we draw the conclusion that the discriminators accept fake images as true images 20% of the time.

Due to early saturation of the discriminator variance and generator loss, we have tuned down total epochs to 75 for next two datasets: *monet2photo* and *vangogh2photo*

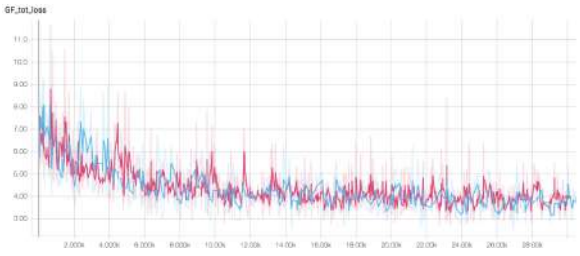

Figure 15. Total Generator loss profile for datasets : *monet2photo* and *vangogh2photo* - 75 epochs

Similar trend as Figure 12 is observed in Figure 15. The total generator loss is exponentially decreasing with respect to iterations. However, the loss in Figure 12 is lower than in Figure 15, might be due to the initialization effect.

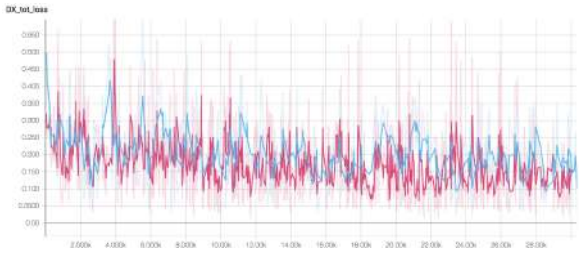

Figure 16. Discriminator (X) loss profile for datasets : *monet2photo* and *vangogh2photo* - 75 epochs

Figure 17. Discriminator (Y) loss profile for datasets : *monet2photo* and *vangogh2photo* - 75 epochs

There exists a high variance in painting to photo translation as seen in Figure 16 when compared to photo to painting translation shown in Figure 17. This can be explained intuitively as it is easier to learn the conversion from photo to painting but its much more troublesome to do the other way round. This is also apparent in the observed outcome of the painting to photo translation for both *monet2photo* and *vangogh2photo* datasets.

3.3. Testing Analysis

The summarized result for all of the datasets, namely, *horse2zebra*, *apple2orange*, *winter2summer*, *monet2photo*, *vangogh2photo*, are compiled in the *APPENDIX* section with good results and bad results as two main categories.

horse2zebra

We have a successful implementation of the *horse2zebra* dataset with rare failure cases. We were able to replicate the results presented by Jun-Yan [19]. Through observation, the failure cases are more for zebra to horse than horse to zebra translation. This dataset is also prone to initialization trap but fortunately we have managed to re-initialize the network graph to avoid such failure cases. See the good results for converting horse to zebra and zebra to horse in Figure 18 and Figure 19. The bad results are shown in Figure 28.

apple2orange

The outputs for this dataset are not as desired for both converting apples to oranges (Figure 22) and oranges to apples (Figure 23) compared to the *horse2zebra* dataset. The bad results look even worse (Figure 30).

The reason is that this dataset suffers form a bad initialization trap. The initialization trap is when the weights of the generative network initialized in such a way that it produces the negatives of the complementary colors to the target colors. For example, the background of the apples to orange and orange to apple translation comes out to be negatives of the original. The same happened with the *horse2zebra* dataset. The re-initialization of the network graph is one way to avoid this trap. Unfortunately, due to restricted walltime in 'Blue Waters', several re-initializations are infeasible. Hence, we have presented the results with bad initialization and only categorize the images based on target object itself. But for *horse2zebra* dataset, we have tried multiple re-initializations and thus obtained descent results.

winter2summer

This dataset produced the most astonishing results for both way translations (see Figure 21 for winter to summer and Figure 20 for summer to winter). The network is able to not only translate but also maintain or sometimes enhance the overall perceptual quality of the image. Most of the good translations occur when input images have compositions of variety of elements like trees, leaves, mountains/rocks and

snow/sand in the landscape. We have rare failure cases when there is a portrait style image present, high percentage of single entity present (ex : snow, sand or mountain) and presence of specular reflection. This failure could additionally cause acyclic behavior as well in the translation. Details can be found in Figure 29.

monet2photo

In general, it is difficult to convert a painting to photo as compared to photo to painting. Similar difficulty is observed in the training for this dataset. Although, there exists a non-decreasing variance, the testing results have better quality than expected. Most of the images are successfully translated to the target domain.

Details can be found in Figure 24 for converting monet to photo and Figure 25 for photo to monet. Results that are relatively not that good are shown in Figure 31.

vangogh2photo

Like *apple2orange* dataset, this dataset also suffers from the initialization trap. Additionally, due to the presence of comparatively less number of images to train (i.e. 400 training images) it is a challenging task to alter the hyperparameters to adjust the network again to avoid the trap. Due to restricted walltime we could only attempt the re-initialization twice but unfortunately could not avoid the initialization trap. Details can be found in Figure 26 for vangogh to photo and Figure 27 for photo to vangogh. Some bad results that reflect the problem of bad initialization can be found in Figure 32.

4. Discussions

Our implementations seem to provide good results overall. The results obtained from horse2zebra, summer2winter_yosemite and monet2photo show a lot of promise. However, there are certain cases in which our implementation fails. In the apples2oranges dataset, several of the output images have an effect of negative photographs. This is probably due to bad initialization as similar results were coming up in the horse2zebra implementation. But since we ran horse2zebra on Google Cloud, it was easier for us to re-run the network as soon as we figured out the issue, which solved the problem. The apples2oranges was implemented on Bluewaters. Despite, rerunning it 3 times, we still could not find a good initialization. Similar effects are seen in vangogh2photo outputs.

Since, we tested our network on test images for every 400^{th} iteration, we found that the horse2zebra worked best between 75-80 epochs. This motivated us to use fewer epochs in our implementation for monet2photo and vangogh2photo. Another issue that we noticed in the horse2zebra outputs is that different number of epochs provide best results for different colored horses. This is a typical evidence against the "One size fits all" concept.

Observing the generator loss from the figure above, it is evident that the summer2winter and horse2zebra datasets are among the curves that come close to low losses. This means that the apparent results are better for these dataset as compared to others. Hence, we have observed the pleasing results for horse2zebra and summer2winter as compared to other datasets.

As seen in the graphs for the generator and discriminator losses, the generator losses reduce with the number of iterations, and the discriminator losses stagnate around 0.2. The generator losses are as desired but a good loss graph for the discriminator would hover around 0.5 as discussed in GANs. The possible reason for the bad discriminator loss could be that our discriminator is not too sophisticated. Further improvements in the discriminator such as an implementation of PatchGAN could provide us with better results.

5. Conclusion

Our work in this project involved learning and summarizing the theoretical details of CNN, GANs, CyCleGANs along with the architectures of it's generator and discriminator networks, implementing the entire CycleGANs algorithm from scratch on TensorFlow, training the model with different training sets and hyperparameters, testing the trained model with multiple test sets, performing quantitative analysis on the loss trends, and presenting detailed discussions on both our positive and negative results.

Experimental results show that our CycleGAN model performs well in the task of two-way unpaired image-to-image translation, such as transforming horses to zebras and zebras to horses. Most of the outputs are quite reasonable.

Based on our work and results, we believe that we have achieved the desired goal that motivated us to take up this project. Specifically, we stated in our proposal that our minimal goal was to implement the method proposed in the paper and test it on test images such as *horse2zebra* and *apple2orange* and self provided images. We have actually achieved more than that. Besides the two datasets mentioned above, we have also tested our model on *winter2summer*, *monet2photo* and *vangogh2photo* datasets.

It is worth mentioning that this project also provided us the opportunity to understand the implementation of neural networks on cloud platforms such as Google Cloud. There are still some limitations of this project. Although we have achieved a lot more than our minimum goal, we did not have enough time to make our model work on videos, which was our maximum goal. Another problem is that some of our datasets suffer from the bad initialization trap and we weren't able to find the good ones.

Future work may contain solving the above two problems and exploring more about CycleGANs by further testing on different configurations, such as modifying the structures of the generator and discriminator.

6. Statement of individual contribution

Harshad Rai

Harshad was responsible for implementing the generator network. He also implemented the training algorithm with Ziyu and Naman contributing with their respective work. While training the network, he was also responsible for setting up Google cloud due to insufficient walltime for large training data in Blue Waters.

Naman Shukla

Naman was incharge of the website for collaboration and dataset management. He also wrote the scripts for training and testing with dataset in place along with Harshad and Ziyu's implementations. Blue waters setup and runtime visual analysis is also done by Naman with continuous support from the other two team members.

Ziyu Zhou

Ziyu implemented the discriminator network for the cycle GAN module. She was also responsible for writing the testing module for the network. Along with Harshad and Naman, she worked on visualizations of the results and dataset analysis of all results.

References

[1] Taesung park cyclegan datasets. `http://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/`.

[2] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[3] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] Hardikbansal. Cyclegan-layers, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[8] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint*, 2016.

[11] Leehomyc. Tensorflow implementation of cyclegans (for some layers), 2017.

[12] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision*, pages 702–716. Springer, 2016.

[13] LynnHo. Function used so that output images are within the 0-255 range. `https://github.com/LynnHo/CycleGAN-Tensorflow-PyTorch/blob/master/image_utils.py#L9-L17`, 2018.

[14] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821. IEEE, 2017.

[15] A. Saxena. Convolutional neural networks: an illustration in tensorflow. *XRDS: Crossroads, The ACM Magazine for Students*, 22(4):56–58, 2016.

[16] M. Schapiro, P. Cézanne, G. Courbet, V. van Gogh, G. Seurat, P. Picasso, M. Chagall, A. Gorky, and P. Mondrian. *Modern Art: 19th & 20th Centuries: Selected Papers*. Braziller, 1978.

[17] S. Semeniuta, A. Severyn, and E. Barth. A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*, 2017.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.

Appendix

In this section we present the collage of images for all the datasets. This collection is arranged in a sequential manner into two major category : good images and bad images.

Figure 18. Good Results for Converting Horse to Zebra

Figure 19. Good Results for Converting Zebra to horse

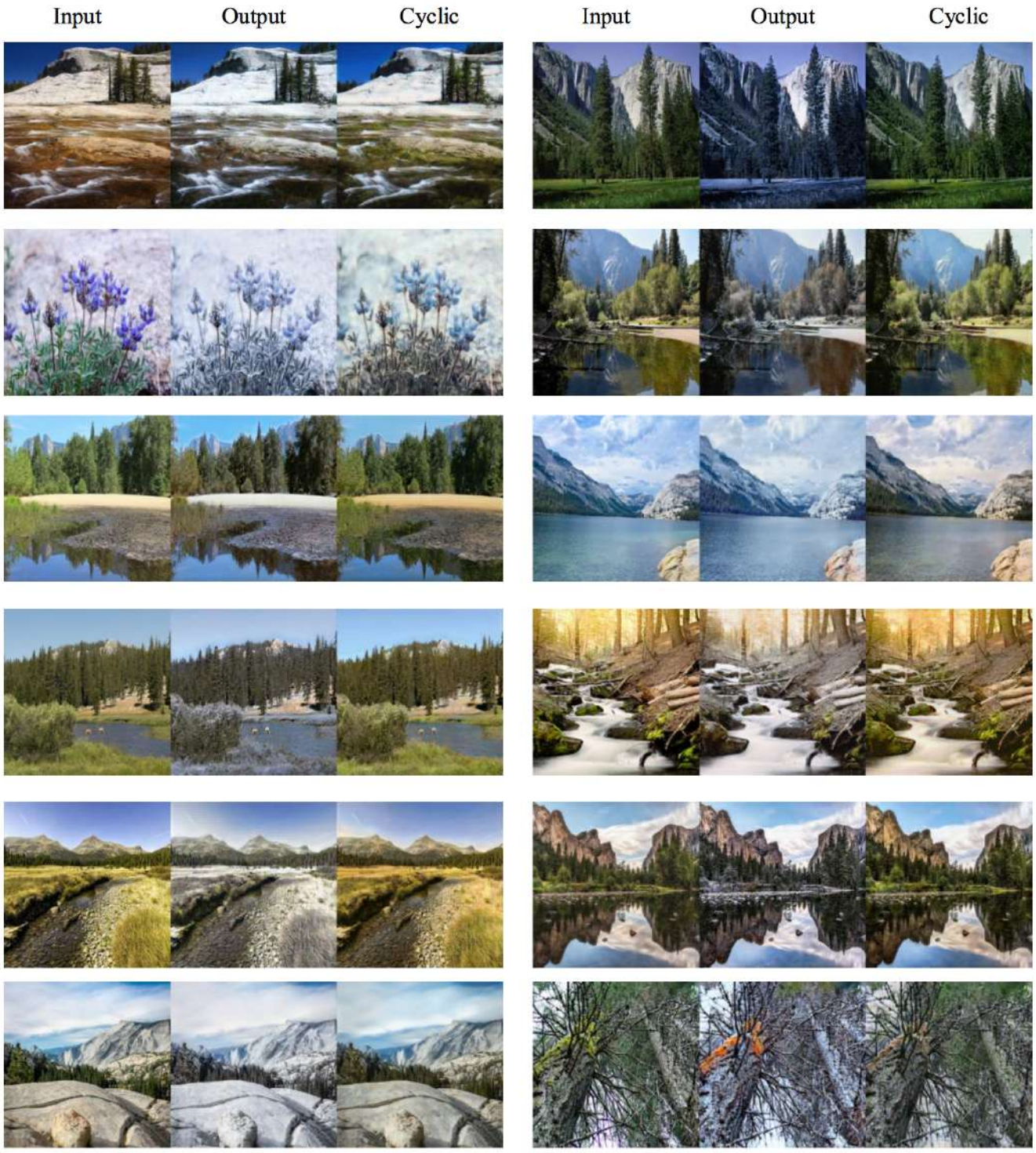

Figure 20. Good Results for Converting Summer to Winter

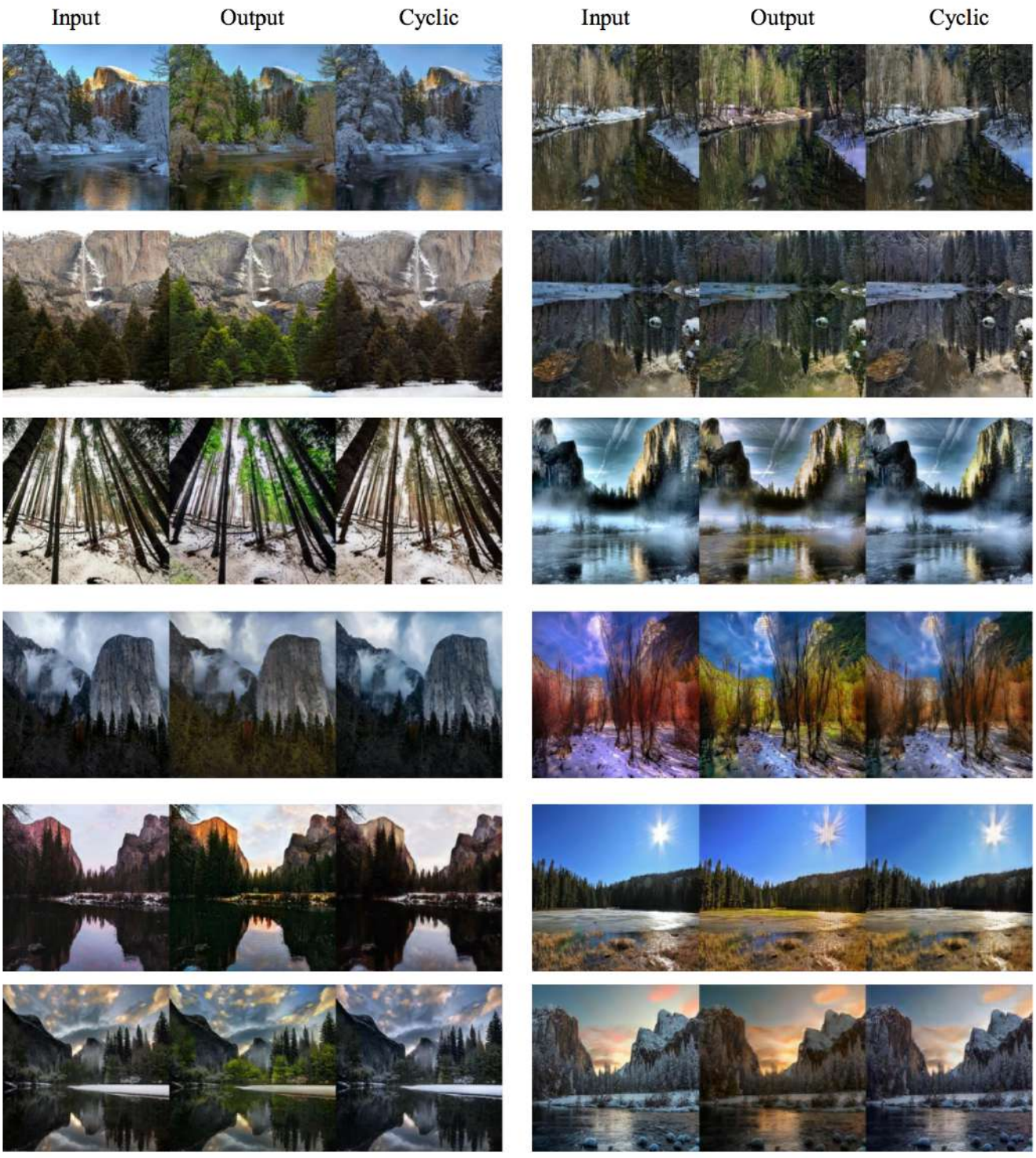

Figure 21. Good Results for Converting Winter to Summer

Figure 22. Good Results for Converting Apple to Orange

Figure 23. Good Results for Converting Orange to Apple

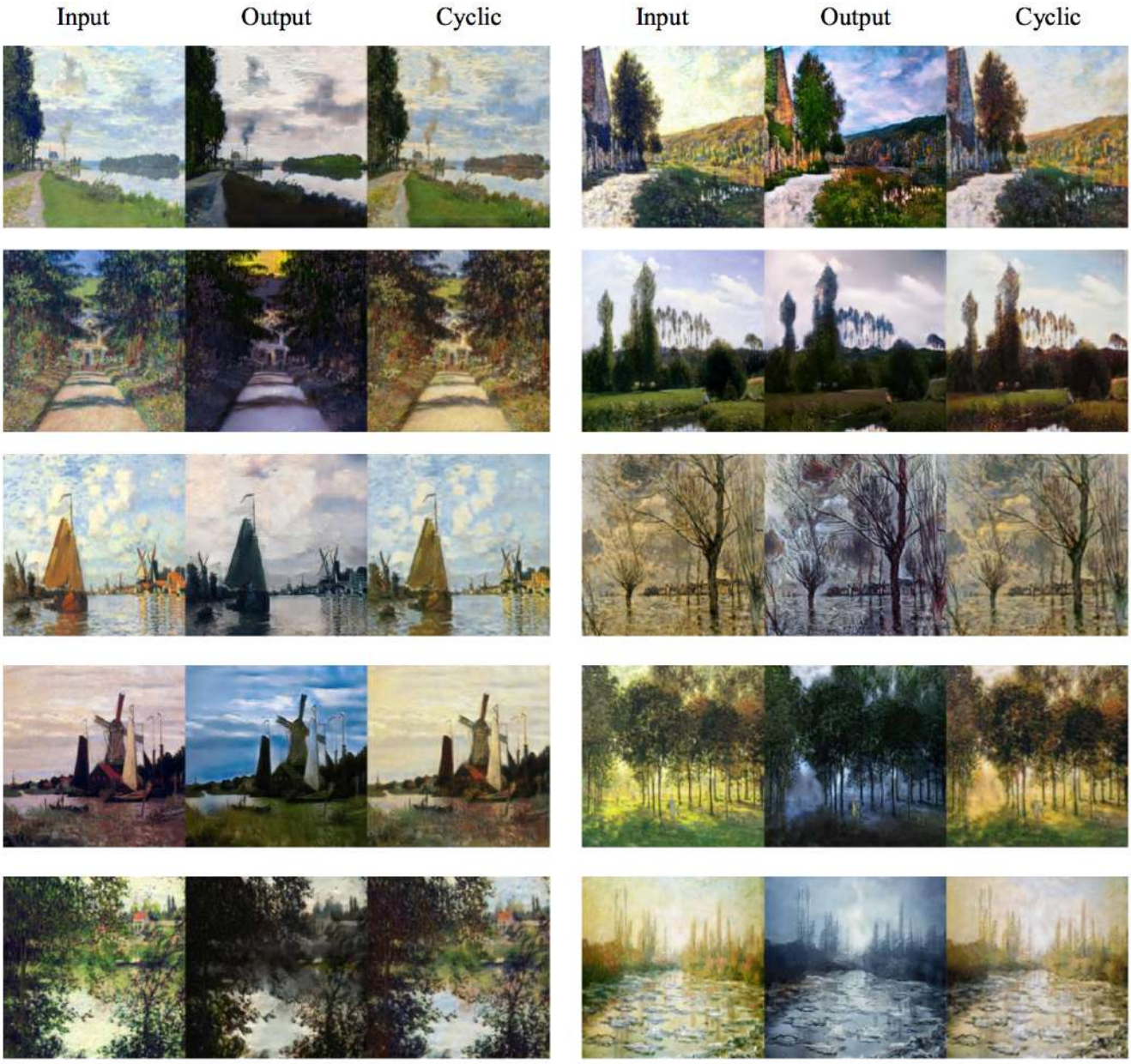

Figure 24. Good Results for Converting Monet to Photo

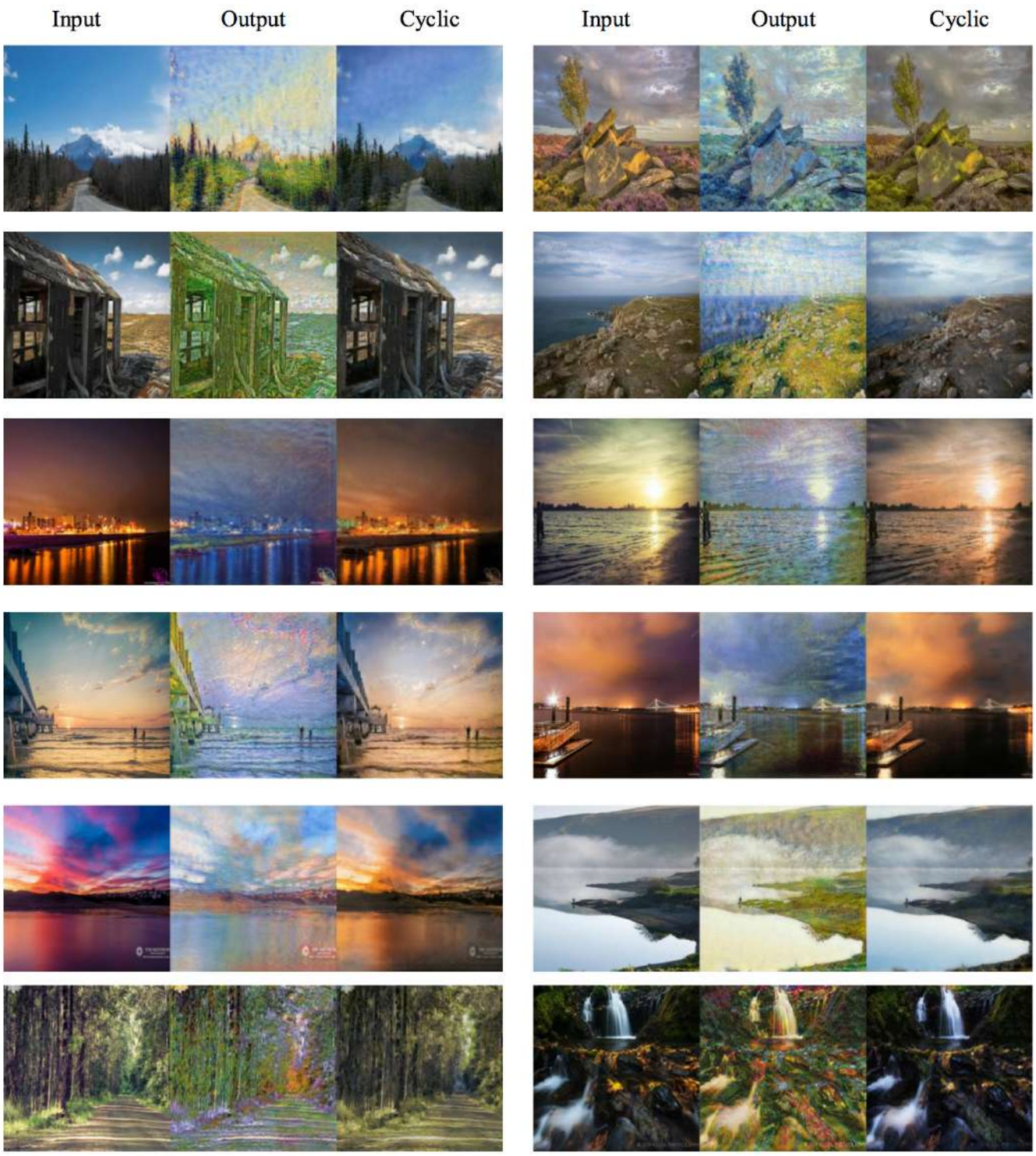

Figure 25. Good Results for Converting Photo to Monet

Figure 26. Good Results for Converting Vangogh to Photo

Figure 27. Good Results for Converting Photo to Vangogh

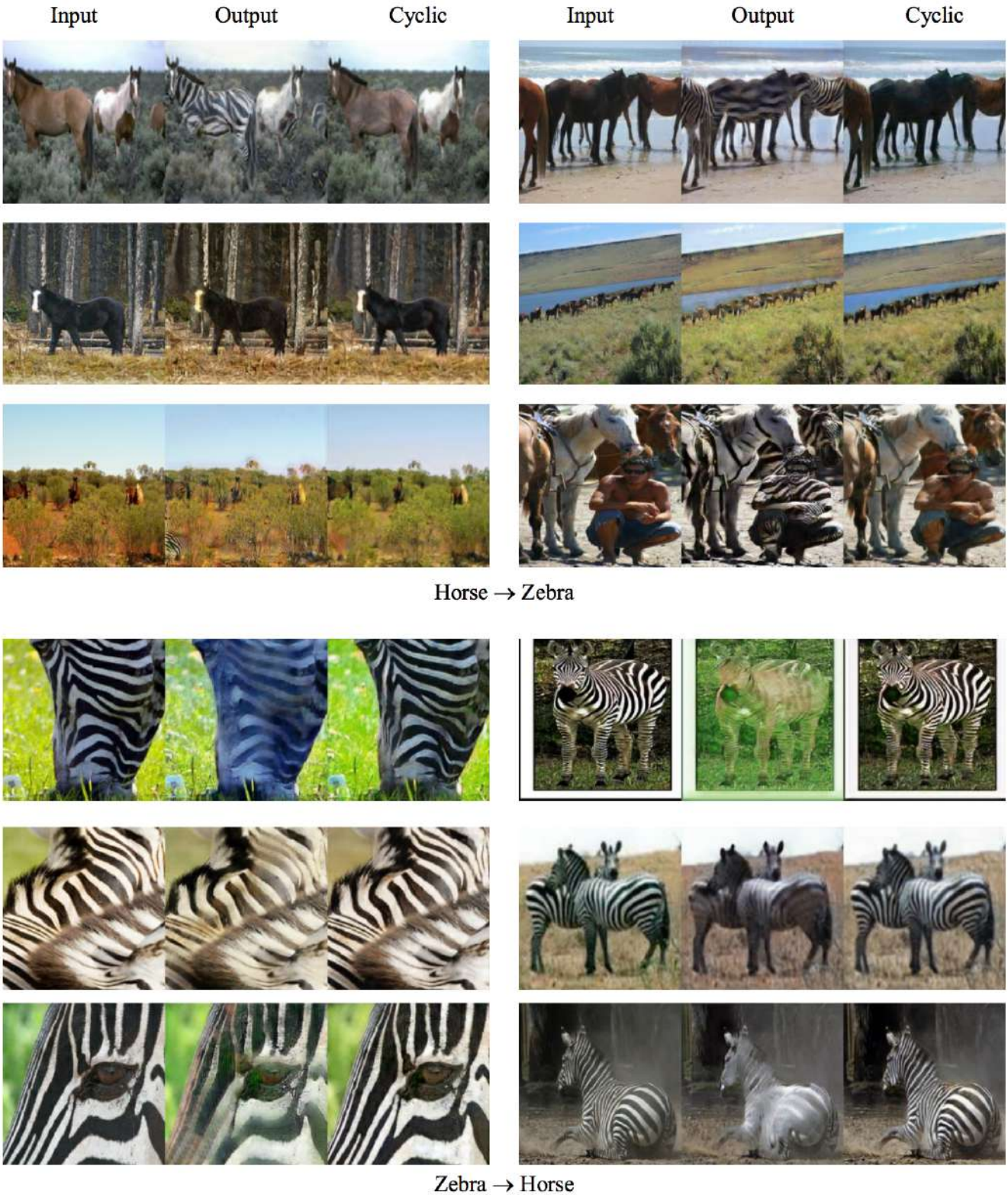

Figure 28. Bad Results for Converting Horse to Zebra and Zebra to Horse

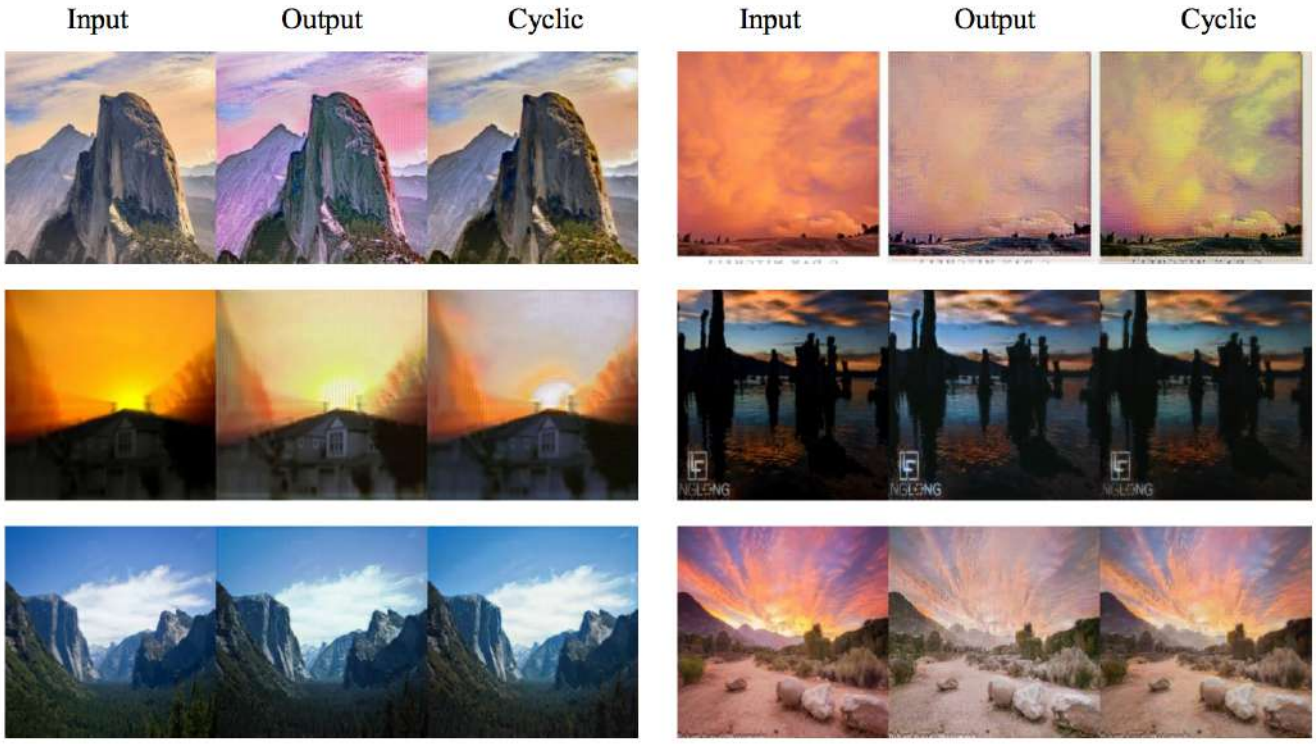

Figure 29. Bad Results for Converting Summer to Winter and Winter to Summer

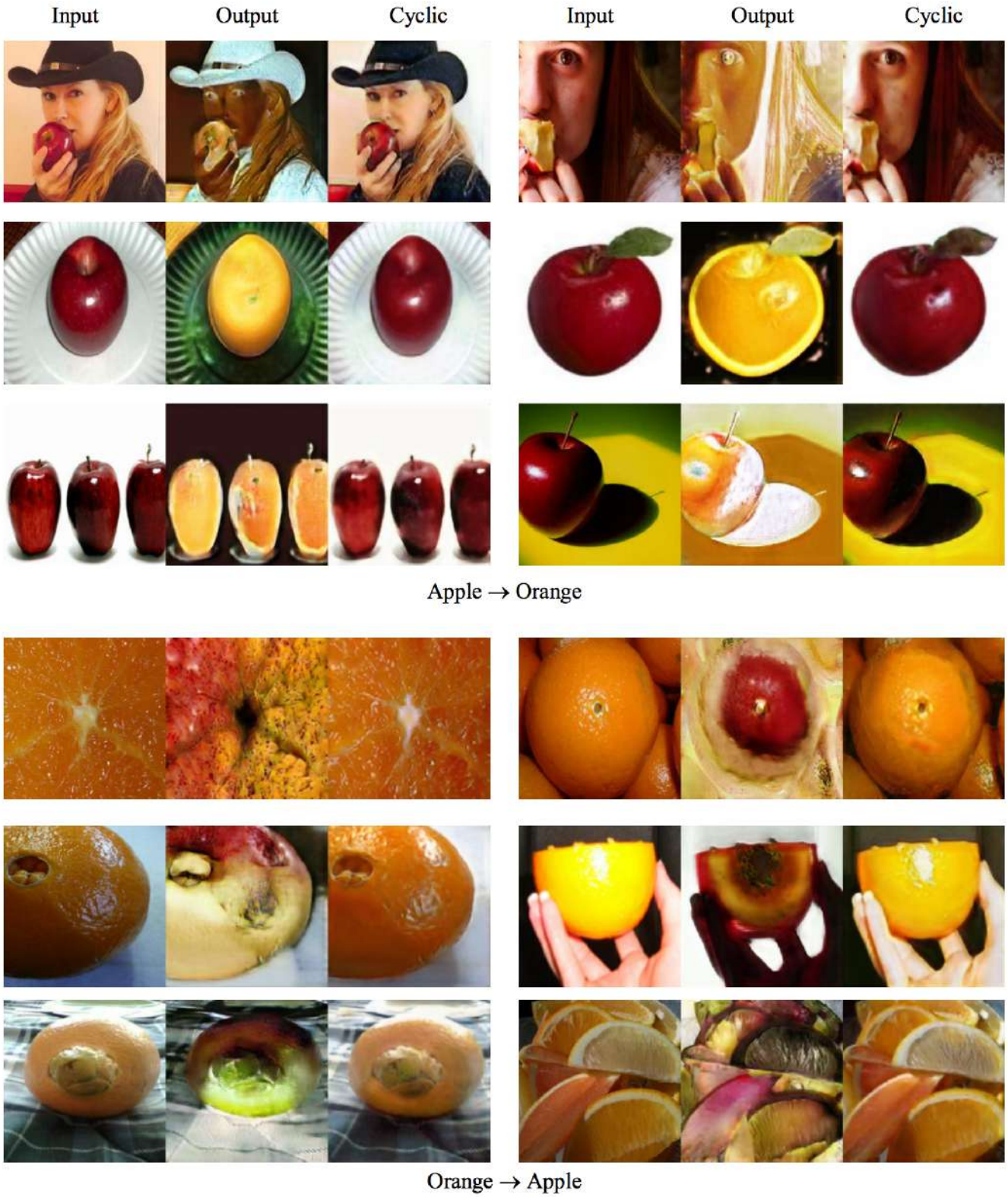

Figure 30. Bad Results for Converting Apple to Orange and Orange to Apple

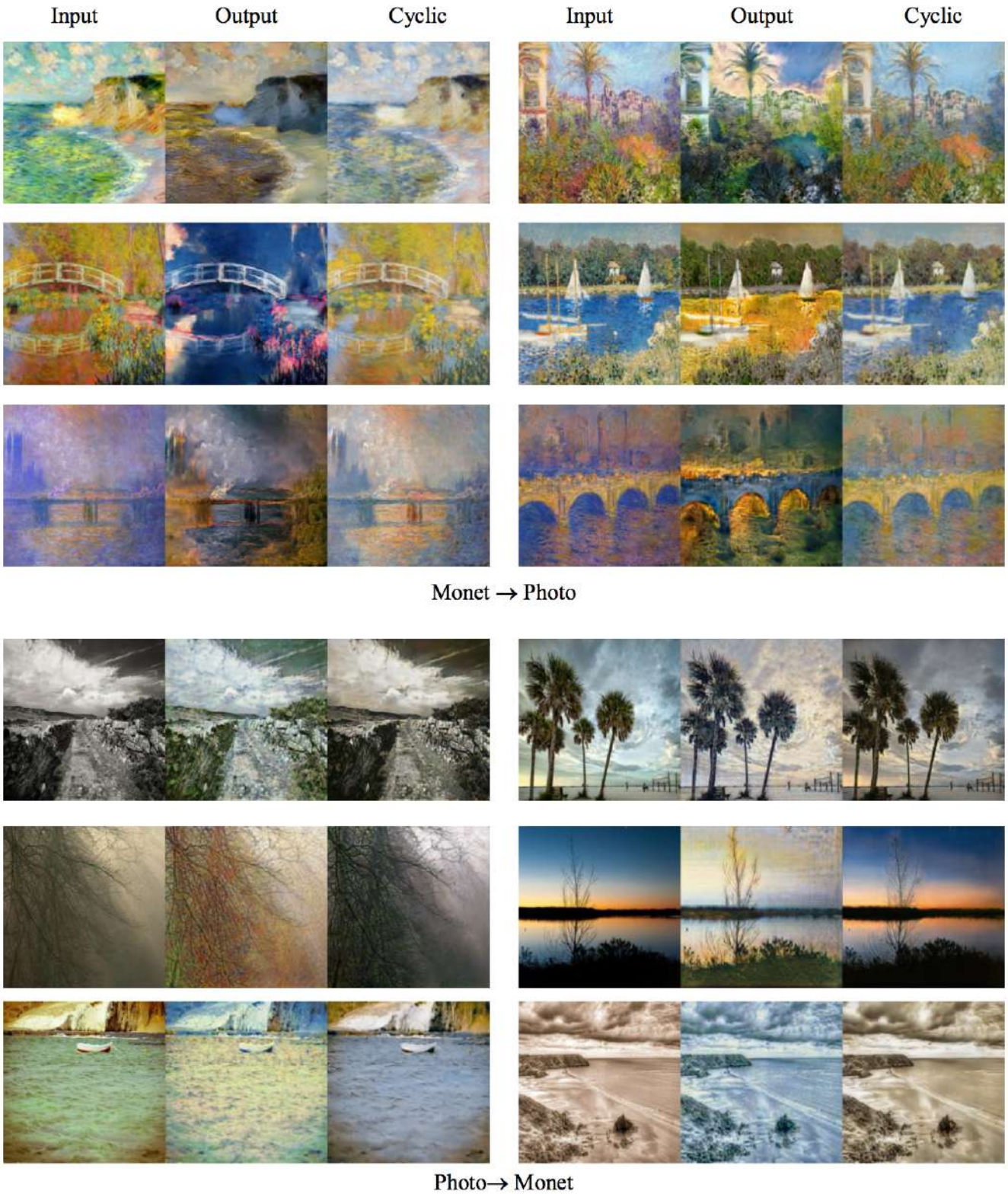

Figure 31. Bad Results for Converting Monet to Photo and Photo to Monet

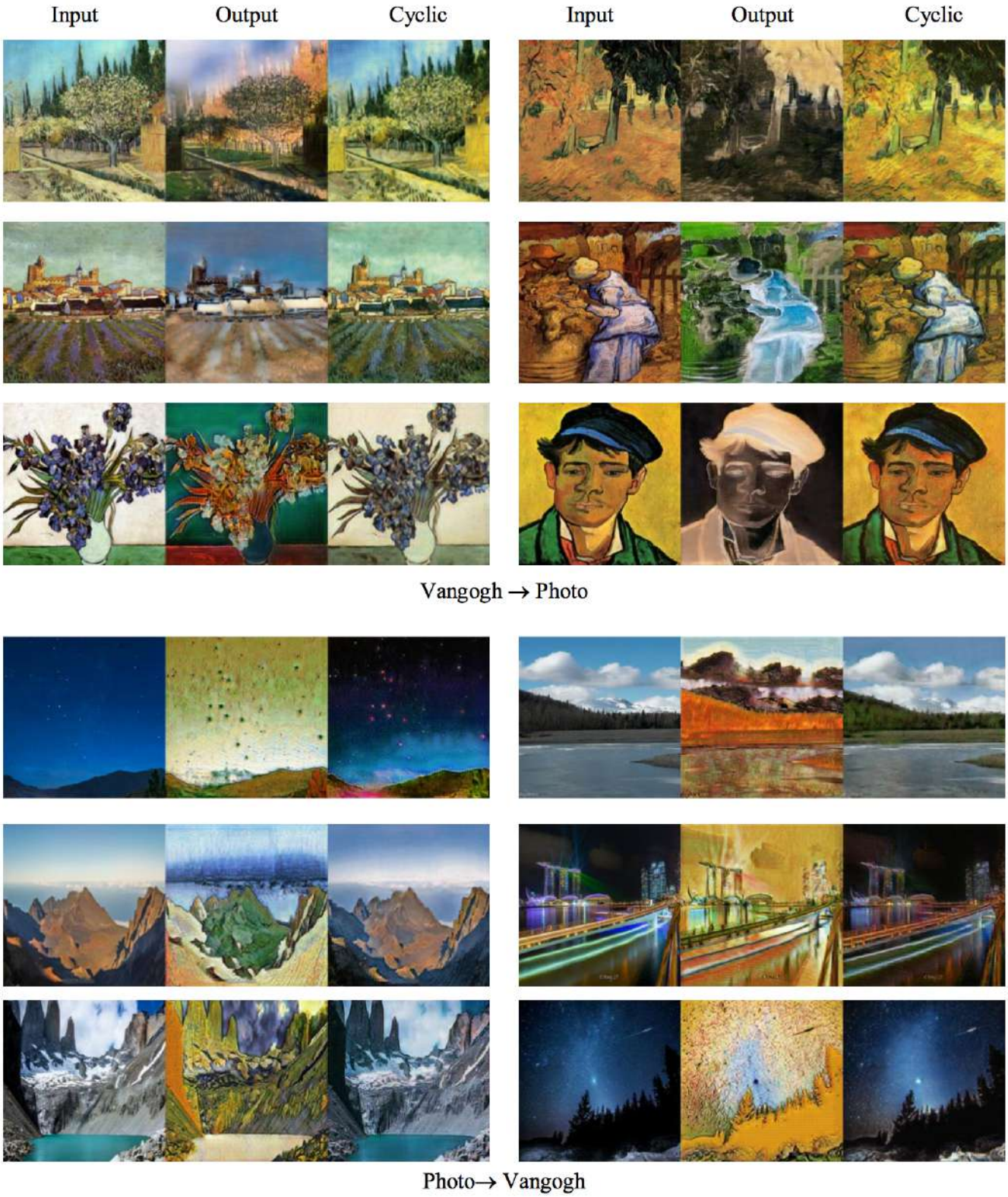

Figure 32. Bad Results for Converting Vangogh to Photo and Photo to Vangogh