A Beginner's Guide to Cycle Consistent Adversarial Networks

James William Stonebridge [1]

Figure 1. Left four: Style Transfers from MNIST to SVHN. Right four: Style Transfers from SVHN to MNIST.

Abstract

CycleGAN is a generative method of Style Transfer originally designed by (Zhu et al., 2017). In this report, I detail my re-implementation of CycleGAN, along with the original method and outcomes of my implementation. Github:

1. Problem

Github

How does one measure "Style"? Is there a metric that could capture the "Van Goghness" of Starry Night or a value that could express how "wintery" a landscape is? If such values existed, how might they be used to apply a given style elsewhere? These questions lie at the heart of the problem tackled by "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" (aka CycleGAN) (Zhu et al., 2017).

More succinctly, CycleGAN seeks to solve the following problem: Given two styles, can an algorithm be designed to re-imagine an image of one style, in the second style? This problem, commonly referred to as style transfer, has been an area of research since the original style transfer paper by Leon Gatys et al. (Gatys et al., 2016) in 2016. Style Transfer models have allowed AI to imitate the creation of art by applying an artistic style to any image (see figure 2. Additionally, image segmentation tasks, such as highlighting cars and removing backgrounds are effectively solved using style transfer algorithms (Isola et al., 2017b).

Historically, training style transfer networks required datasets of paired images (Isola et al., 2017c). For example, consider training a model that restyles images of classical architecture into art deco equivalents. Each image in the classical set would require an accompanying photo of the same structure but built using an art deco design philosophy. In this example and many others, creating such a dataset would be impossible. In contrast, finding an unpaired dataset is significantly easier. Hence, CycleGAN's innovation, which allows the training of style transfer models without paired data, is a breakthrough.

Figure 2. Left four: Landscapes Re-imagined as Monet Paintings. Right four: Monet Paintings re-imagined as Real Photos.

2. CycleGAN relative to Similar Works

2.1. Style Transfer

"A Neural Algorithm of Artistic Style" by Leon Gatys et al. (Gatys et al., 2016) was the first to adequately demonstrate that image style[1] and structure[2] could be learned independently and reapplied elsewhere. As such, it is credited with the introduction of Style Transfer in the Image Domain. Their Algorithm learns from a single style and content image by feeding both to a pre-trained VGG network. The

[1]Image Style refers to aspects of an image such as color palette and texture. For example, Monet's paintings possess a unique texture

[2]Image Structure refers to the general shapes that compose an image. For example, in a photo of Mount Everest, the shape of the mountain peak is considered part of the image's structure

outputs from select layers within the VGG are then used to formulate an optimization problem that outputs the content image in the style of the content image.

This method has the clear advantage of only requiring 2 images. In contrast, CycleGAN demands large datasets for 2 distinct styles. Additionally, this method only requires one network in contrast to CycleGAN's four separate networks. This makes Gatys' method significantly easier and quicker to train. However, this method falls short of CycleGAN in its ability to generalize. This method requires a time-intensive training cycle for every image we want to perform style transfer on (**?**). In contrast, CycleGAN's algorithm creates generators that can be quickly applied to any image within its style domain(Zhu et al., 2017). Moreover, the style transfer offered by Gatys et al. is a one-way mapping. As the name suggests, CycleGAN offers a two-way mapping (figure 3).

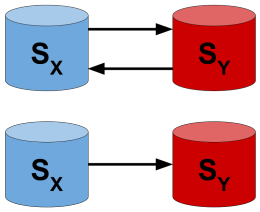

Figure 3. Top: CycleGAN allows style transfer to and from each style. Bottom: Gatys' method is one way

2.2. Generative Adversarial Networks

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. (Goodfellow et al., 2014) and have revolutionized the creative capacity of Artificial Intelligence. A GAN consists of a Generator $G(x)$ and a discriminator $D(y)$. The Generator's job is to create outputs that resemble some target dataset, typically a collection of images. It is the job of the Discriminator to determine whether an image is real (it is from the target dataset) or fake (it was created by the generator). Due to the nature of each job, the generator and discriminator will work against each other as they are trained simultaneously. In practice, this means the Generator must constantly become better at creating target images to fool a discriminator that is getting better at identifying fake images.

The Discriminator, which resembles a binary classifier, defines loss through its accuracy:

$$L_{discriminator} = ||z_y - D(y)|| \tag{1}$$

y: Some Image

z_y: A label representing whether y is real or fake

In contrast, the Generator defines loss by its ability to fool the Discriminator. This is known as Adversarial Loss:

$$L_{adv} = ||1 - D(G(x))|| \tag{2}$$

In the original Goodfellow et al. (Goodfellow et al., 2014), x represents random noise.

CycleGAN builds on this basic setup but with two major alterations: Cycle Consistency (see3.2) and a Markovian Discriminator (see 2.3).

2.3. Markovian Discriminator

The idea of a Markovian Discriminator was formulated by Isola et al. in "Image-to-Image Translation with Conditional Adversarial Networks" (Isola et al., 2017a). Instead of outputting a single scalar to indicate the veracity of an image, a Markovian Discriminator outputs a matrix. Each component of this output matrix represents whether a patch of the input image is real or fake.

CycleGAN uses such a discriminator due to its many advantages. Primarily, it allows the discriminator, and by extension the generator, to learn from multiple parts of the image, instead of the image as a whole(Isola et al., 2017a). Finally, a Markovian Discriminator Architecture requires fewer parameters. This allows it to require less memory and train faster.

2.4. Generative Style Transfer

GANs have proven a popular method of performing style transfer. However, generators that are trained to style transfer on adversarial loss alone produce blurry and unstructured images (Figure 4). Paired data is a commonly proposed solution to this problem. For example, Pix2Pix (Isola et al., 2017a), proposes the following loss function:

$$||y - G_x(x)|| \tag{3}$$

x: Input image of style S_x

y: A manually recreated version of x in style S_y

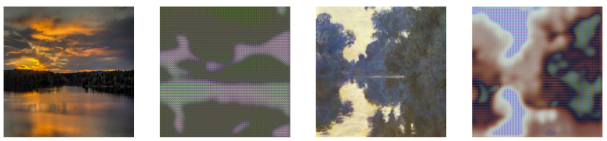

Figure 4. Landscapes ⇔ Monet style transfers from a model I trained using only adversarial loss. Note that the structure of the original images is destroyed as a result.

This method allows for sharper, more realistic images than CycleGAN (section 1). However, as previously described,

large paired datasets are very difficult to create. CycleGAN instead introduces a Cycle Consistency Loss function (see 3.2). This results in slightly more blurry images, but allows datasets to be unpaired as long as they each represent an individual style.

3. (Re)Implementation and Method

3.1. Data collection

I was able to get CycleGAN to successfully train on three datasets: Monet ⇔ Landscape, Aerial Photos ⇔ Google Maps, and MNIST ⇔ SVHN. The first two were reused from the original CycleGAN repository. My MNIST ⇔ SVHN dataset was downloaded from their respective websites(svh)(mni). In addition to these datasets, I was able to create a Human ⇔ Monkey and a Water ⇔Wine dataset. This was accomplished by adapting an image scraper(Ohyicong) that searched Google images based on given keywords. I found that giving the image scraper a single search term, like "man", ended up yielding unrelated images after the first 100 scraped images. I was able to resolve this issue by instead scrapping 100+ images through 10 related search terms (For example: "man", "boy", "hiker", etc.). Ultimately, this allowed the cultivation of 1000-1500 image datasets per style.

When inputting data, I re-implemented the image buffer strategy introduced by Shrivastava et al.(Shrivastava et al., 2017). As Shrivastava et al. demonstrate, this buffer helps stabilize the discriminator by regularly training the discriminator with a mix of new and old fake images. Lastly, I jitter the images to avoid overfitting.

3.2. Architecture

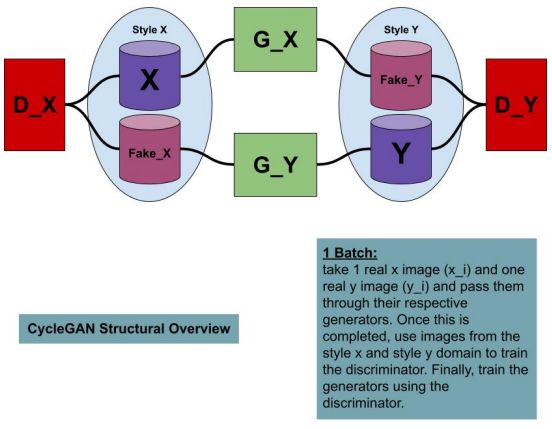

Figure 5. A visualization of the combined CycleGAN Architectures

Instead of using a single Generator and Discriminator, CycleGAN uses 2 of each. D_x and D_y seek to determine how real images of style S_x and S_y are respectively. Meanwhile, G_x re-imagines images from S_x in S_y. Likewise, The opposite is performed by G_y.

My generator architecture follows the structure of Johnson et al's (Johnson et al., 2016) auto-encoder. It is composed of a convolutional layer, two downsampling layers, a chain of residual blocks, two upsampling layers, and a final convolutional layer. The original CycleGAN recommended six residual blocks, but I found it useful to alter this number during experimentation. Additionally, I found the findings of Radford et al. (Radford et al., 2016) helpful. Per their recommendations, all normalization layers were changed to Batch Norm. This reduced blur in the final images. Finally, all layers used a ReLU for activation, except the last layer which used Tanh.

The Discriminator follows the Markovian Structure of Isola et Al(Isola et al., 2017a). It consists of 4 convolutional layers. Each layer is normalized using batch normalization, followed by a leaky ReLU of slope 0.2. The first layer has 64 filters and each subsequent layer doubles the number of filters. The convolution layer maps back to 1 filter and is followed by a Sigmoid layer.

3.3. Losses

L_{adv} (eq. 1) is reused by CycleGAN to optimize each Generator. However, CycleGAN also introduces two new loss functions for each generator/discriminator pair:

$$L_{cyc\_x} = ||G_y(G_x(x)) - x|| \tag{4}$$

The namesake of CycleGAN, Cycle-Consistency Loss (L_{cyc}), is most notable. Reusing the style transfer dichotomy from section 2.1, we can describe L_{adv} as a method of emphasizing style transfer, while L_{cyc} emphasizes the structure of the original image. This is accomplished through the recovery of generated images using the opposing generator, $x_{recovered} = G_y(G_x(x))$. $x_{recovered}$ is then compared pixel by pixel against the original input image x. This method preserves the structure of the original image with minimal impact on the stylistic transfer.

$$L_{idt\_x} = ||G_x(y) - y|| \tag{5}$$

L_{idt} exists to encourage the color profile of the desired style transfer. This is achieved by feeding a generator an image from its target style domain. The output is then compared pixel by pixel with the input. In practice, this discourages any deviation from the general color scheme of the target style. This is not always necessary but proves helpful when the coloration of the original image is desired in its style

transfer. For example, Monet ⇔ Landscape benefited from this Loss.

Finally, each Loss is multiplied by a scalar coefficient λ. The final cumulative loss function for the Generators is:

$$\begin{aligned} L_{G_x,G_y} =& \lambda_{adv}(L_{adv_x} + L_{adv_y}) + \\ & \lambda_{cyc}(L_{cyc_x} + L_{cyc_y}) + \\ & \lambda_{idt}(L_{idt_x} + L_{idt_y}) \end{aligned} \tag{6}$$

The Discriminators' loss function is simply:

$$L_{D_x} = \frac{1}{2}(||1 - D_x(G_x(x))|| + ||D_x(y)||) \tag{7}$$

$$L_{D_y} = \frac{1}{2}(||1 - D_y(G_y(y))|| + ||D_y(x)||) \tag{8}$$

4. Training

I follow the original CycleGAN's method of training over a period of 200 epochs with a batch size of 1. The learning rate is linearly decayed to zero from the 100th to 200th epoch. Additionally, loss coefficients of $\lambda_{adv} = 1$, $\lambda_{cyc} = 10$ and $\lambda_{idt} = 0.5$ are used.

4.1. Hardware Limitations and Visualization

My model was trained on the standby queue of Purdue's Gilbreth Cluster. As such, the first issue to confront was the 4-hour training limit. Due to the immense size of my Model (14,136,212 parameters) I was limited to 50 epochs per training session. I was able to resolve this issue by introducing a timer to my script that would save the model right before the four-hour mark. This allowed me to continue training when necessary.

I was able to visualize my model by keeping graphs of every loss in every network during training (Figure 6). I set these graphs to continuously update so that issues could be caught early as soon as they occurred. Additionally, I ensured consistency in the quality of outputs by regularly updating a separate figure of 16 style transfers.

4.2. What does it mean for a GAN to Train?

When training most artificial intelligence models the goal is to get some loss to converge to zero. However, when training GANs neither L_{adv} nor $L_{Discriminator}$ should reach 0. This is undesirable because generative results are only improved by the constant competition between the generator and discriminator. Thus, if L_{adv} ever reaches 0, it means the generator has "won" and will no longer improve the quality of its generated outputs. Conversely, if $L_{discriminator}$ ever reaches 0, it means the Discriminator has "won" and will forbid the generator from improving. So what is desired? In my research, I was not able to find clear answers to this question as it appears to still be an area of study. In general, the best advice I found was to keep L_{adv} fluctuating (Dwivedi, 2023) and, most importantly, to avoid network collapse.

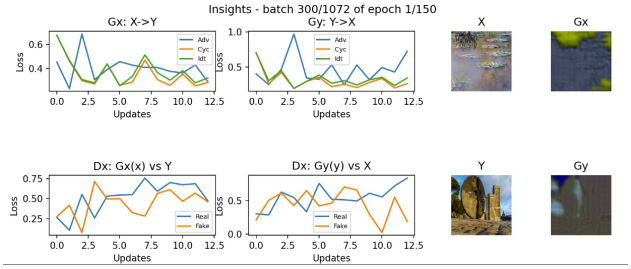

Figure 6. The Analytic panel after a few batches of training. When training on Jupyter, it is designed constantly update the graphs and images throughout training. When working through slurm, this figure is stored as a jpg and constantly updated.

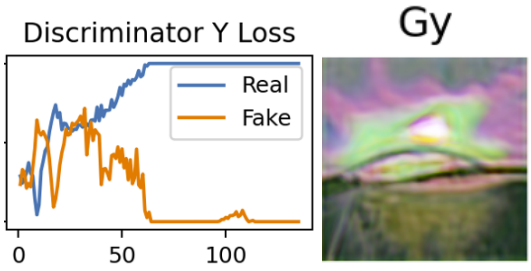

Figure 7. This Discriminator Collapsed around update 60. As you can see it has 0 loss on fake images and 1 loss on real images. It is stuck at a local minimum in which it classifies all input as fake. The corresponding output image is of poor quality due to the discriminator's collapse

Early in training, I found that my discriminators were very unstable. Through observation of my discriminator's L_{fake} (how well it identifies fake images) and L_{real} (how well it identifies fake images), I identified that one loss would sometimes "overpower" the other and cause collapse (see figure 7). In other words, the discriminator would get trapped in a local minimum of classifying everything as real or fake.

I was able to resolve this issue by monitoring the stability of each discriminator during training. If the L_{real} or L_{fake} exceeded 0.8, I would multiply the higher loss by 1.8 and multiply the lower loss by 0.2. This heavily encouraged the gradient to favor minimizing the higher loss (figure 8).

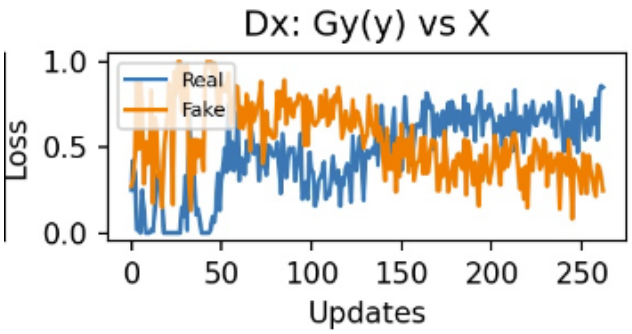

Figure 8. This Discriminator was unstable around update 40. However, when L_{fake} exceeded 0.8, it was multiplied and the optimizer punished it until the discriminator was stable again.

5. Experiments

5.1. Baselines

I compare my implementation with 4 other algorithms.

CycleGAN (Original): I was able to use the original CycleGAN repository to generate some style transfer for qualitative comparison. Additionally, various images are taken from the CycleGAN website (Zhu et al., 2017). Finally, I use the AMT scores reported in the original paper for qualitative comparison.

SimGAN: (**?**) Another style transfer GAN that utilizes cycle consistency. Shrivista et Al. uses $L_{cyc} = ||G_x(x) - x||$ as opposed to CycleGAN's $L_{cyc} = ||G_x(x) - y||$. Images and AMT scores are used for comparison.

Feature Loss + GAN: (Simonyan & Zisserman, 2014) Another Generative method. However, instead of learning from the output image, this network learns from the outputs of layers within its Generator architecture. In theory, this allows it to learn from high-level image features. Images and AMT scores are used for comparison.

Pix2Pix: Pix2Pix is described in section 2.4. Since Pix2Pix utilizes paired data, its images will be viewed as the high bar for this project.

5.2. Normalization Functions

I found many recommendations from "Unsupervised Representation Learning with DCGANS" (Radford et al., 2016) to be very intriguing. Radford et Al argues that instance normalization layers destabilize GANs because they lack the scale and shift parameters found within batch norm layers. In this experiment I tested 6 CycleGANs on the Monet⇔Landscape dataset. 3 CycleGANs used instance norm in their generator and 3 used batch norm.

Figure 9. three style transfers from three different model's trained using instance normalization.

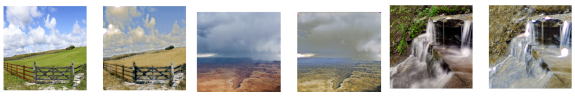

Figure 10. three style transfers from three different model's trained using batch normalization.

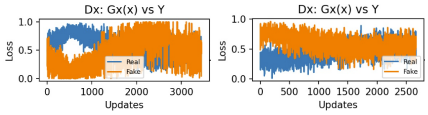

Figure 11. Left: Generator loss over time with instance norm layers. Right: Generator loss over time with batch norm layers

As figure 9 and 10 demonstrate, the models trained on batch norm are sharper than those trained on Instance Normalization. Additionally, the bottom two transfers in figure 9 possess undesirable artifacts in random sections of the image. When observing the loss graphs of each model, the instance norm Generators had more noisy losses during training (figure 11).

5.3. Perceptual Studies

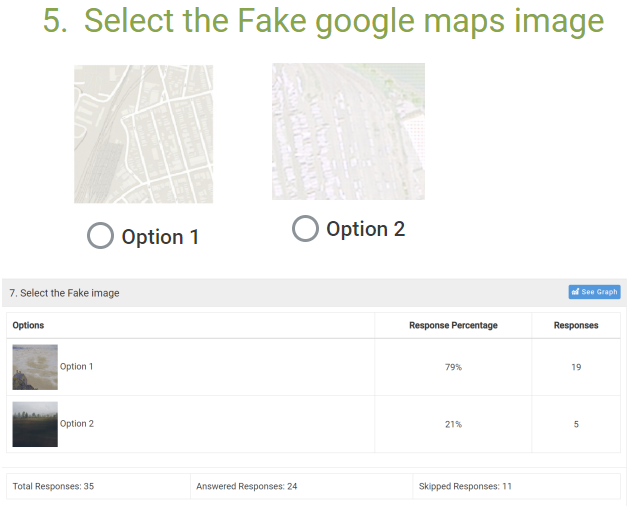

Figure 12. Top: An example of how the survey questions were displayed to participants (Right is fake). Bottom: The results of a Monet ⇔ Landscape Style transfer (Top is fake).

In the original CycleGAN paper, workers from Amazon

Mechanical Turk (AMT) are used to assess the realism of various images. I replicated these results by organizing a survey of 24 participants. Each participant was given 1 minute to look through pairs of images of the Aerial ⇔ Street view and Landscape ⇔ Monet dataset. Each pair contained one real and one fake image. Respondents were shown 10 pairs of style transfers. Active participation in the survey was incentivized by offering a small cash prize to whoever could guess the most correct fake images. Ultimately, participants guessed incorrectly 22.1% of the time on average. This is immensely better than SimGAN and Feature Loss + GAN; however, it is slightly behind the original CycleGAN by 4%.

Model	% who were fooled by GAN
SimGAN	0.7
Feature Loss + GAN	1.2
CycleGAN (Me)	22.1
CycleGAN (Original)	26.8

Table 1. AMT results

It is worth noting this study differed from the original in a few ways. I gave my respondents more time to view the pictures (2.5 seconds versus 1). I also did not include screening questions to filter out respondents who did not pay attention. Finally, my survey was a mix of Aerial ⇔ Street view and Landscape ⇔ Monet transfers, whereas Zhu et al. only used Aerial ⇔ Street view in their survey.

5.4. Personal Datasets

In addition to Google Maps ⇔ Aerial and Landscape ⇔ Monet datasets, I trained on three datasets of my own creation. These were: MNIST ⇔ SVHN, Water ⇔ Wine and Man ⇔ Monkey.

5.4.1. MNIST ⇔ SVHN

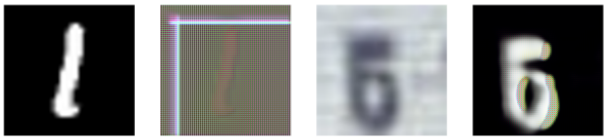

Figure 13. Left: MNIST unsuccessfully transfers to SVHN. Right: SVHN successfully transfers to MNIST.

Figure 14. Left: A successful MNIST to SVHN transfer. Right: A successful SVHN to MNIST transfer.

Using the basic setup outlined in section 4, I was able to transfer mnist's style onto svhn by epoch 30-50. However, my model struggled to transfer svhn's style onto mnist, even after 200 epochs. As seen in figure 13, transfers to svhn were grainy and discolored. This was likely caused by the variance in number coloration in the svhn dataset. Since L_{adv} behaves as the conduit of the target style, I suspected that raising it might yield better results. In my experiments, setting L_{adv} to 1.5 allowed successful style transfer to svhn by epoch 100 (figure 14).

5.4.2. Water ⇔ Wine

Figure 15. Top: Wine to Water. Bottom: Water to Wine

Using the setup in 4 proved unsuccessful for either style transfer in this dataset. However, increasing $L_{idt}$to 1.5 yielded better results when transferring to water. I found it helpful to train with L_{adv} set to 1.5 to encourage the transfer to wine.

5.4.3. Human ⇔ Monkey

Figure 16. Two images from the monkey dataset and two images from the human dataset. Note that the monkey's have different facial structures and fur colors. Also note that the monkey's are in poses and environments that might never appear in the human dataset.

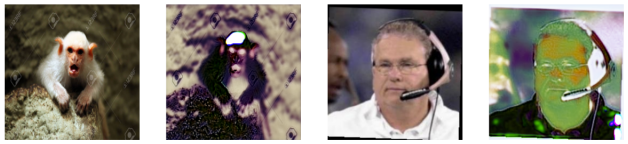

Figure 17. A typical result from training this dataset. I found that the background was often distorted and the subject would be tinted.

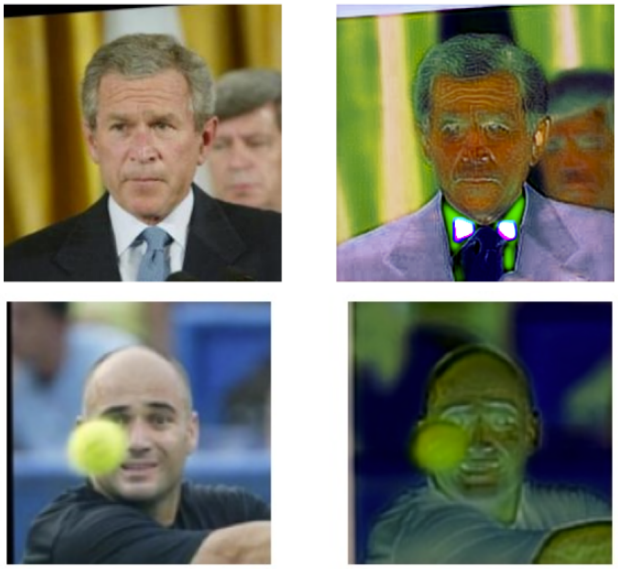

Figure 18. Top: The style transfer alters the bone structure around the eyes and emphasizes forehead wrinkles. The upper lip is also enlarged. Bottom: The eye structure is altered by the style transfer to appear more "monkey-like".

Training on this dataset was extremely difficult. Adversarial Losses were consistently unstable during training which lead to frequent generator and discriminator convergence. I suspect much of this instability stemmed from the variance within my Monkey dataset. I realized too late that the aesthetic variety between monkey species is profound. For example, spider monkey's are more lankey and small than swamp monkey's. Additionally, I believe the complexity of this transfer was overly ambitious. Notably, the backgrounds and poses of monkey's significantly differed from those in the human dataset. Ultimately, I was unable to create any convincing style transfers from this dataset. Nonetheless, I provide some more successful results in figure 18.

5.5. Qualitative Comparisons

On the next page (Figure 20), I've compiled a list of various Images from the other style transfers, including the original CycleGAN. These images are placed next to the outputs of my implementation for qualitative comparison.

6. Limitations and Conclusion

As I have demonstrated, my model can effectively output style transfers on multiple datasets. However, it often fails if a dataset involves a style transfer that is too complex (see section 5.4.3). This can also be caused by variance within the individual style's themselves.

It is also worth noting that this algorithm is very inconsistent. For example, I found the same set of hyper-parameters would sometimes yield excellent results, just to fail during the next experiment. Even with the discriminator stabilization algorithm I outlined in section 4.2, the generator or discriminator still are susceptible to collapse. This is particularly common on the Google Maps dataset.

In this paper, I explained the process of re-implementing CycleGAN from its paper and related writings. Tangential works are examined, explained, and compared to the original paper. I elaborate on the underlying method of CycleGAN and how I deviated from it. Finally, I provide experiments that demonstrate my implementation is capable of training on the original CycleGAN datasets and other low-complexity datasets.

Acknowledgements: I would like to thank Ruqi Bai and Wei Chen for the hours of assistance they gave me over the semester. This is the second neural network I've ever built and they were both critical in helping me overcome the learning curve. Professor Qi Guo also offered guidance in the development of this project. In particular, I found his explanation of auto-encoders helpful. My classmate, Ken Wrong pointed me toward the MNIST SVHN dataset. Lastly, I thank Jun-Yan Zhu and his colleagues, without whom this project would not be possible.

References

MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. URL http://yann.lecun.com/exdb/mnist/.

The Street View House Numbers (SVHN) Dataset. URL http://ufldl.stanford.edu/housenumbers/.

Dwivedi, H. Understanding GAN Loss Functions. *neptune.ai*, 4 2023. URL https://neptune.ai/blog/gan-loss-functions.

Gatys, L. A., Ecker, A. S., and Bethge, M. A Neural Algorithm of Artistic Style. *Journal of Vision*, 16(12): 326, 8 2016. doi: 10.1167/16.12.326. URL https://doi.org/10.1167/16.12.326.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. GANGenerative Adversarial Nets. *Journal of*

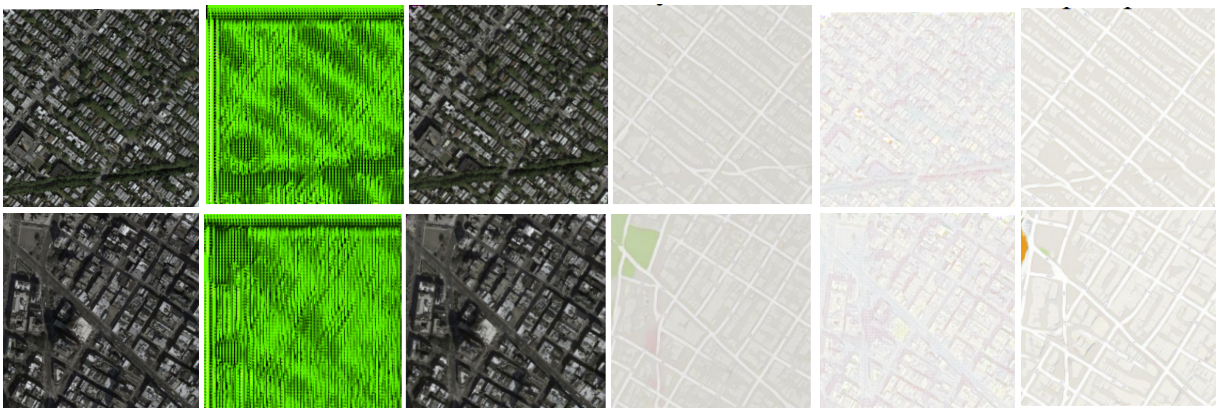

Figure 19. Columns from left to right: input image, Feature Loss + GAN, SimGAN, Original CycleGAN, My Implementation, Pix2Pix

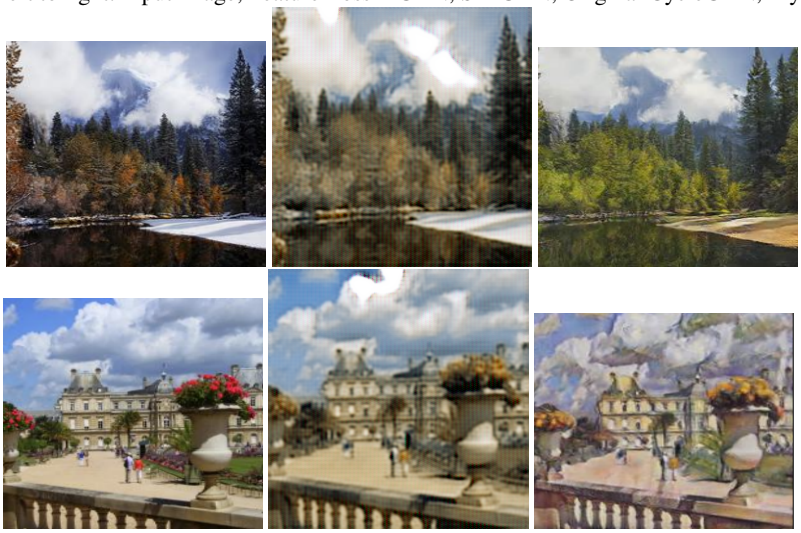

Figure 20. Columns from left to right: input image, my implementation's output, original CycleGAN

Japan Society for Fuzzy Theory and Intelligent Informatics, 29(5):177, 12 2014. doi: 10.3156/jsoft.29.5\{_}177\{_}2. URL https://www.jstage.jst.go.jp/article/jsoft/29/5/29_177_2/_pdf.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. 7 2017a. doi: 10.1109/cvpr.2017.632.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. 7 2017b. doi: 10.1109/cvpr.2017.632. URL https://doi.org/10.1109/cvpr.2017.632.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. 7 2017c. doi: 10.1109/cvpr.2017.632.

Johnson, J. C., Alahi, A., and Fei-Fei, L. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. Springer Science+Business Media, 10 2016. doi: 10.1007/978-3-319-46475-6\{_}43.

Ohyicong. GitHub - ohyicong/Google-Image-Scraper: A library to scrap google images. URL https://github.com/ohyicong/Google-Image-Scraper.

Radford, A., Metz, L., and Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 1 2016. URL https://arxiv.org/pdf/1511.06434.

Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J. M., Wang, W., and Webb, R. Y. Learning from Simulated and Unsupervised Images through Adversarial Training. 7 2017. doi: 10.1109/cvpr.2017.241. URL https://doi.org/10.1109/cvpr.2017.241.

Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 9

2014. URL http://export.arxiv.org/pdf/1409.1556.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 10 2017. doi: 10.1109/iccv.2017.244.