Unpaired Image to Image Translation using CycleGAN

Jules Youngberg
Department of Computer Science
University of Victoria
jules@uvic.ca

Abstract

Image to image translation is the transformation of images from one set to another. Generative adversarial networks (GANs) are a powerful type of neural network for generating data, and can be used for translating images. The CycleGAN architecture uses two GANs to learn functions that form a cycle. The added cycle constraint improves learning stability for image translation. Like all deep neural network models, there are many hyper parameters available to tune the system for a particular problem. Bayesian optimization is a technique that can find a near optimal configuration of hyper parameters without testing all possible configurations. Training CycleGANs proves to be difficult due to the inherent instability of the GAN architecture. The results are mixed, though interesting and promising. With enough computing power, this technique could be very effective for particular problems.

1 Introduction

Image to image translation refers to the problem of translating images from one set to another. There are various applications for this, such as image and video filters, drawing applications, super resolution, and map generation. For example, image filters could apply a painter's style to a photograph. Structurally similar objects can be transformed, for example horses can be converted to zebras and apples can be converted to oranges. Drawing applications are a particularly interesting use case, where a user could sketch an object and a filled in version of their drawing could be generated.

Typically when training machine learning models, there is a target output for each input, commonly referred to as the label. Clearly, it may be difficult to obtain target output images for all input images in the applications described previously. This report studies the problem of unpaired image to image translation. If target images were included, the problem would be considered paired image to image translation. To solve the problem at hand, a function G must be learned that maps images from set X to set Y. The output $G(x)$ for $x \in X$ should be indistinguishable from images in the target set Y. While learning G, another function D_Y can be learned that determines whether an image belongs to Y or not.

Functions G and D_Y can be implemented as a Generative Adversarial Network (GAN), which is a machine learning architecture composed of two neural networks. One generates data, and is called the generator, while the other distinguishes generated data from real data, and is called the discriminator [1]. This architecture is commonly used for generating images from random vectors, and produces very convincing results. With the complexity of image translation, this definition of the problem is often not restrictive enough. Problems commonly occur such as mode collapse, where all input images map to the same output [2].

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

To remedy the problems with GANs, Zhu, Park, Isola, and Efros (2017) introduce the CycleGAN architecture which includes an additional constraint of cycle consistency. Their reasoning for this is that when translating a sentence from English to French, it should translate back to the same English sentence [3]. They implement this by introducing a second generator H that maps images from set Y to set X, as well as a second discriminator, D_X, for determining whether images belong to X. Cycle consistency is enforced by adding new losses that encourage $F(G(x)) \approx x$ and $G(F(y)) \approx y$ [3]. The way these functions interact is illustrated in Figure 1.

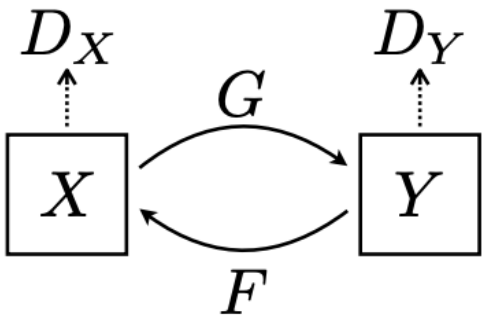

Figure 1: CycleGAN architecture diagram from the initial publication [3].

2 Architecture

GANs consist of 2 types of networks, the generators and discriminators. The generator's goal is to generate data indistinguishable from the target data, meanwhile the discriminator's goal is to determine whether data is authentic or not [1]. The generator typically takes random data as a vector, generally much smaller than the output. In this way, the generator converts data from a latent space to the image space. However, in the case of image to image translation, the generator must take an image as input. To perform the transformation, the generator acts like an auto-encoder [3]. First, the input image is downsampled to a latent space using a series of convolution layers. Then the latent space is upsampled using transposed convolution layers. The discriminator takes an image as input, and downsamples with a series of convolution layers to a single value that represents the discriminator's confidence that the input image is authentic.

There have been many implementations of the CycleGAN since it's initial publishing, some with slightly different parameters, loss function, or generator architecture. The original CycleGAN uses a res-net for the generator, though a u-net is commonly used instead because it is less computationally expensive [4]. With so much room for customization, it is hard to know what is best for a given problem. From the resources online [4,5,6,7], I built a highly configurable CycleGAN class using Tensorflow Keras. It supports both res-net and u-net generators, instance and batch normalization, binary cross entropy and least squares loss, as well as various other hyper parameters that are passed to the generators and discriminators. A couple interesting flag parameters are $flip\_labels$ and $soft\_labels$. The former trains the discriminator to output a 0 for authentic data instead of a 1, and the latter softens labels with random numbers between 0 and 0.1, moving 1 to 0.9-1.0, and 0 to 0.0-0.1 [7]. This appears to help the models converge in some cases.

Though there are two types of generators and losses, during the final training sessions, least squares was used with a u-net generator. Least squares was used in the original paper's training because it produced better and more stable results than binary cross entropy [3]. The preliminary experiments ran during this project showed similar results. The u-net was used in favour of the res-net due to computational costs and time constraints.

2.1 Network Architectures

The u-net generator consists of 8 encoding layers, each with stride 2, kernel size (3, 3), and all but the first with the configured normalization. Next, there are 7 decoder layers, which are transposed

convolution layers, each with stride 2, kernel size (3, 3), the configured normalization, and the first 3 layers with the configured dropout. Each of the first 7 layers' output is fed to the next layer as well as the corresponding encoder layer, forming the u-net architecture. Finally, there is one last upsampling layer, with $tanh$ activation function. Each layer uses padding same and the number of filters corresponds to what is used in [4].

The discriminator is 5 convolution layers, each with kernel size (3, 3), the first 4 with leaky ReLU activation, and the middle 3 with the configured normalization. Each layer uses padding same and the number of filters corresponds to what is used in [4].

2.2 Loss Functions

The losses will be described for a particular generator G and discriminator D, since they are symmetric for the different GANs within the CycleGAN. The generator's adversarial loss is expressed as:

$$\mathcal{L}_{adv_G} = (D(G(x)) - 1)^2 \tag{1}$$

This trains the generator to fool the discriminator. That is, the generator's output should cause the discriminator to output 1. On the other hand, the discriminator's adversarial loss is expressed as:

$$\mathcal{L}_{adv_D} = (D(y) - 1)^2 + D(G(x))^2 \tag{2}$$

This trains the discriminator to correctly distinguish between real and fake data, classifying real images as 1 and fake as 0. This is clearly in contradiction with the generator's loss. This contradiction is a characteristic of GANs that both makes them produce such impressive results as well as makes training them difficult and unstable. Next, the generators are encouraged to learn the inverse of each other, forming a cyclic pair of functions. This is done by applying the cycle loss, which is given by:

$$\mathcal{L}_{cyc} = \lambda(|F(G(x)) - x| + |G(F(y)) - y|) \tag{3}$$

Where λ is 10 [3].

3 Training

With an easily configurable class, the question becomes what values should the hyper parameters be set to. Testing all possible combinations of the hyper parameters can take a very long time, especially when training 4 deep convolutional networks. To keep training times reasonable, Keras Tuner was used with Bayesian optimization. This type of optimization is able to find a near optimal configuration much faster than grid search because it infers meaning from each trial [8]. Each trial consisted of only 3 epochs, with a maximum of 10 trials, and the optimal model was trained for 50. It would have been nice to allow for longer trials, but the priority was testing different datasets.

Even without hyper parameter optimization, training these networks takes a very long time. To improve training times, training was done on Google Colab, using a Google Cloud Storage bucket for the results. Much of the effort involved in this project was devoted to setting up an efficient training workflow to run experiments and iteratively improve the code.

3.1 Data

Tensorflow provides a wide range of images for training neural networks, even including some of the collections used in the CycleGAN paper [9]. These were the obvious first choice for testing. There are many other image collections available online, and there are virtually endless possibilities with the CycleGAN. Unfortunately, due to time constraints and GPU usage limits on Google Colab, there was not enough time for further experimentation. Every trial used 256x256 images with RGB channels. Training images were also randomly rotated and cropped since that is supposed to help reduce over-fitting [4].

4 Results

A combination of quantitative and qualitative methods are used for evaluating GANs. The loss computed during training can be examined to determine how well the model is improving over time. Secondly, the results of the generators can be examined to see if they are actually producing desirable results. A better solution would be to use a another network to evaluate the content loss of the results [3], though that is beyond the scope of this project. Several trials were performed once an efficient workflow was developed. Results were mixed. In some cases outputs were quite desirable, though many inputs do not work well. Figures 2 and 3 show the training and validation losses for the cezanne2photo dataset, respectively.

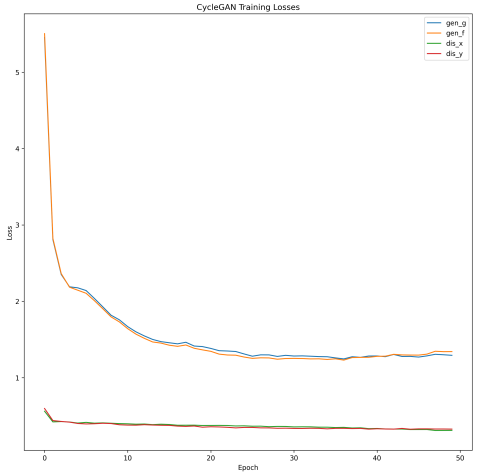

Figure 2: Cezanne2photo training losses.

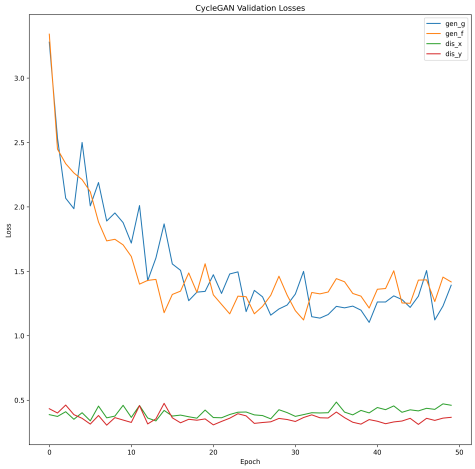

Figure 3: Cezanne2photo validation losses.

There is an okay curve to the generators. The discriminators have little to no improvement each epoch. Over-fitting was bad with the maps dataset, as illustrated by the loss graphs in Figures 4 and 5.

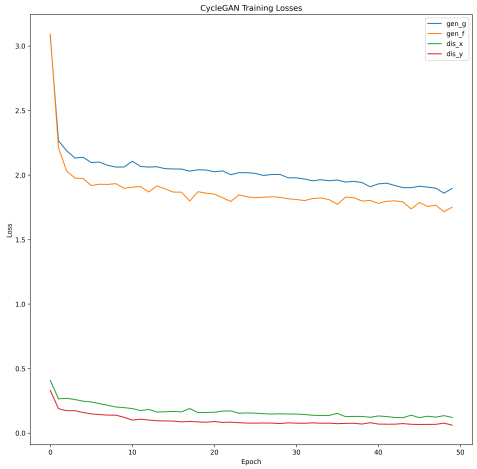

Figure 4: Maps training losses.

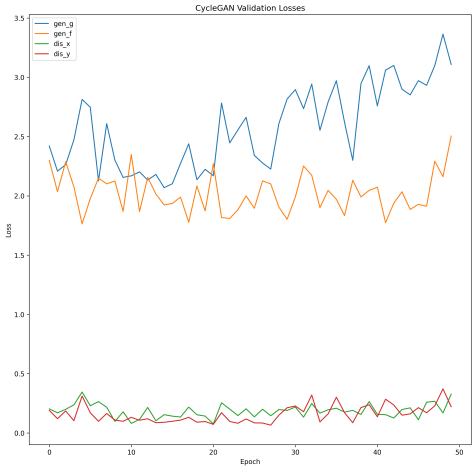

Figure 5: Maps validation losses.

Figures 6 and 7 show some validation outputs for these experiments. The results in Figure 6 are quite good. Generator G has learned to remove the brush strokes but also add more definition. Generator F adjusts the colors to fit into Cezanne's palette and removes a lot of the detail that makes it look real. With some more tweaks and longer training times this could produce quality images. The results in Figure 7 are not bad either. Generator G learns to translate the basics like land, water, and some roads, but on closer examination it is clear that many roads and ground textures are incorrect. Additionally,

this sample is early in the training, before over-fitting started setting in. Figures 8 and 9 show some failure examples.

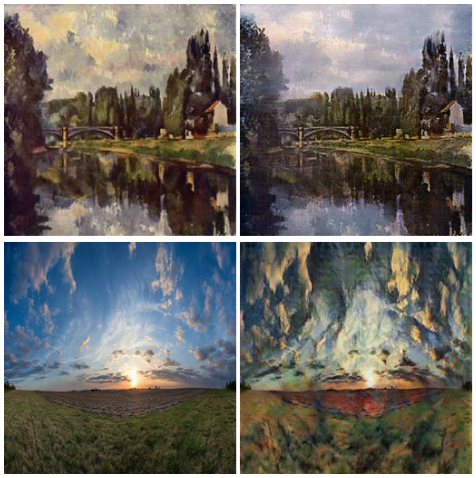

Figure 6: Validation input/output pairs from epoch 44 of cezanne2photo. Top: Cezanne to photo (G). Bottom: photo to Cezanne (F)

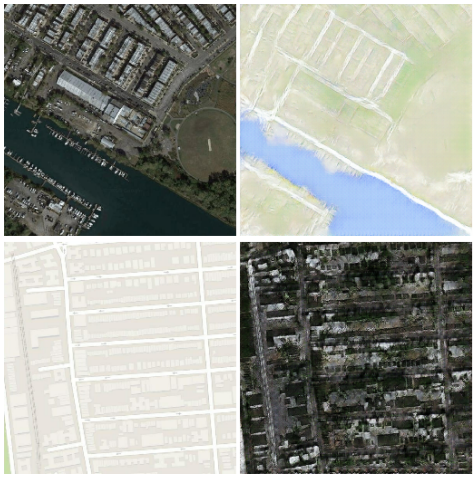

Figure 7: Validation input/output pairs from epoch 17 of maps. Top: satellite to map (G). Bottom: map to satellite (F).

Figure 8: Validation input/output pairs from epoch 47 of horse2zebra. Top: horse to zebra (G). Bottom: zebra to horse (F).

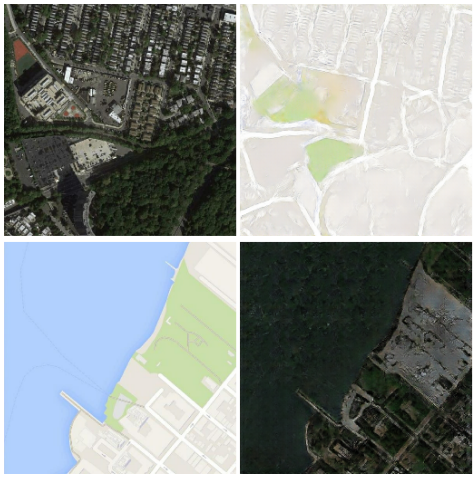

Figure 9: Validation input/output pairs from epoch 17 of maps. Top: satellite to map (G). Bottom: map to satellite (F).

Unfortunately, the generators often learn to invert or severely alter colors in an undesirable way, as in Figure 8. This could possibly be due to the generators over-fitting to some subtle color differences between the image sets, or perhaps it is simply due to instability. Regardless, it could potentially be fixed by introducing identity loss, encouraging $G(y) \approx y$ and $F(x) \approx x$ [4]. In Figure 9 it is clear the generators do not translate different types of ground correctly, since where it is grass in the bottom left becomes something else entirely, perhaps a roof, in the bottom right.

5 Challenges and Improvements

Training GANs is inherently difficult because the generators and discriminators are in a competition with each other, similar to a game of minimax [1]. Due to this competition, GANs are prone to instability. Additionally, with such complex models, training takes many hours making it time

consuming to run experiments. Most impressive GAN results require heavy computing power and long training times.

The most difficult part of this work was training the model in a timely manner in order to test all parts of the program and get results. With so many hyper parameters to configure, and multiple deep networks, testing was a very slow feedback loop at first. Additionally, attempting to perform grid search made every test even slower, and introduced unnecessary development overhead since the CycleGAN had to satisfy the Sklearn Estimator interface. Eventually, grid search was replaced with Bayesian optimization via Keras Tuner, and Google Colab was introduced for training.

Some hyper parameter values caused particularly undesirable effects during the training process. After many trials, it was possible to narrow down the domain of some hyper parameters. Even still, it seams sometimes configurations may produce a lower loss even if the results are not better, or the networks seem to over-fit to another difference between the image sets. The Bayesian optimization uses the average of the losses of the 4 networks, perhaps this is not the best way to score the overall success of a configuration. The range of the generators' and discriminators' losses are different, so the generators will more heavily weight the score. This may be desired or may not. It would probably be better to normalize them and be able to control the weighting. An even better solution would be to use another model to compute the content loss as in the original research paper [3]. Another possible problem with the current implementation is that the learning rate was part of the optimizable hyper parameters. This may be okay, though with a limited number of epochs this produces a strong bias towards faster learning, possibly more unstable, configurations.

6 Conclusions

Image to image translation is a problem with some fascinating applications in image processing. Objects or even entire images can be transformed as long as the general shape of the objects are maintained. The CycleGAN architecture makes it possible to better learn image transformation functions by enforcing a cycle loss, which encourages the generators to learn the inverse of each other. Training these models can be difficult, and is prone to stability issues, but can produce impressive results when tuned correctly. CycleGANs, and GANs in general, are an extremely interesting technology with incredible capabilities, given enough computing power.

Code for this project is available at: https://github.com/julesyoungberg/img-gen

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative adversarial nets*. In NIPS, 2014.

[2] I. Goodfellow. NIPS 2016 tutorial: *Generative adversarial networks*. arXiv preprint arXiv:1701.00160, 2016.

[3] J. Zhu*, T. Park*, P. Isola, and A. Efros. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, in IEEE International Conference on Computer Vision (ICCV), 2017. (* indicates equal contributions)

[4] *CycleGAN | Tensorflow Core*. Tensorflow. June 17, 2021. Accessed on: Aug 8, 2021. Available: https://www.tensorflow.org/tutorials/generative/cyclegan

[5] S Theiler. *Transforming the World Into Paintings with CycleGAN*. Analytics Vidhya. Nov 19, 2019. Accessed on: Aug 8, 2021. Available: https://medium.com/analytics-vidhya/transforming-the-world-into-paintings-with-cyclegan-6748c0b85632

[6] J. Brownlee. *How to Implement CycleGAN Models From Scratch With Keras*. Machine Learning Mastery. Aug 7, 2019. Accessed on: Aug 8, 2021. Available: https://machinelearningmastery.com/how-to-develop-cyclegan-models-from-scratch-with-keras/

[7] U. Desai. *Keep Calm and train a GAN. Pitfalls and Tips on training Generative Adversarial Networks*. Utkarsh Desai. Apr 29, 2019. Accessed on: Aug 8, 2021. Available: https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9

[8] A. Gozzoli. *Practical Guide to Hyperparameters Optimization for Deep Learning Models*. Floydhub. Sept 5, 2018. Accessed on: Aug 8, 2021. Available: https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

[9] *cycle_gan | Tensorflow Datasets*. Tensorflow. Jun 30, 2021. Accessed on: Aug 8, 2021. Available: https://www.tensorflow.org/datasets/catalog/cycle_gan