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

Bastien Ponchon
MVA ENS Paris Saclay
January 2017
bastien.ponchon@ens-paris-saclay.fr

Abstract

This final report present my work on paper [7]. I will first present the Generative Adversarial Networks framework, as well as some of its common issues. Then I will explain the CycleGAN model, presented in [7] to realize Unpaired Image-to-Image translation, and how the authors manage to overcome the above mentionned issues. Finally, I will present the experiments I undertook on the CycleGAN framework, in order to test its limits and abilities.

1. Introduction to Generative Adversarial Networks

Generative Adversarial Networks (GANs) were introduced by [2] as a deep generative model able to learn to represent an estimate, either explicitly or implicitly, of some distribution $p_{data}(Y)$ on some space Y, given a training set composed of samples y drawn from this distribution.

1.1. The GAN framework

A GAN can be seen as a two player game between two parametrized and differentiable (with regards to their inputs and their parameters) functions: a generator G and a discriminator D.

The **Generator** takes as input some x drawn from a a prior distribution p_X on a hidden space X and tries to produce an output $G(x) \in Y$ drawn from a probability distribution p_{model} such that that p_{model} is close from p_{data}, that is to say such that $G(x)$ seems to have been drawn from the distribution $p_{data}(Y)$.

The **Discriminator**, on the other hand takes as input an element $y' \in Y$ and outputs the probability $D(y')$ that y' comes from the training set, that is to say, has been drawn from the true distribution $p_{data}(Y)$, rather than having been generated by G, and therefore drawn from the distribution p_{model}. This translate by the maximization of the loss function $\mathcal{L}_{GAN}$ given by the following log likelihood (or the minimization of the corresponding negative log likelihood):

$$\begin{aligned}\mathcal{L}_{GAN}(G, D, X, Y) =& E_{y\sim p_{data(Y)}}\left(logD(y)\right) \\ &+ E_{x\sim p_X}\left(log(1 - D(G(x)))\right)\end{aligned}$$

The generator tries to fool the discriminator, and therefore tries to minimize this very value $\mathcal{L}_{GAN}(G, D, X, Y)$. The solution of this minimax game (or zero-sum game, as the generator and the discriminator try to minimize two opposite cost functions) is given by: $min_G max_D \mathcal{L}_{GAN}(G, D, X, Y)$.

Both the generator and the discriminator are usually represented as deep neural networks and are trained by applying Stochastic Gradient Descent to update them alternatively at each step of the training phase. In order to prevent the generator gradients to vanish whenever the discriminator rejects successfully the generated samples $G(z)$, and hence preventing the generator from being properly updated, the cost function that the Generator aims at minimizing can be replaced by $-E_{x\sim p_X}\left(log(D(G(x)))\right)$, while the discriminator keeps trying to minimizing the same cost functions.

1.2. Some major issues with GANs

Although, [2] has proven that the GAN training procedure converges $p_{model} = p_{data}$ in the space of probability density functions, that is to say if the models have infinite capacity (infinite number of parameters), the implementation of GANs as deep neural networks with a finite set of parameters for both the discriminator and the generator leave the GAN framework facing several problems.

1.2.1 Instability and Non Convergence

As stated in [1], it is the major issue of the GAN framework. Indeed, as both players (the generator and the discriminator) tries to minimize opposite cost functions, the updates of one can sometimes undo the progress made by the other when

updating its own parameters. These alternative updates may lead to oscillation in the loss function $\mathcal{L}$ (one player trying to maximize it while the other try to minimize it) slowing down the learning procedure which may not even end up any close to an equilibrium state.

1.2.2 Mode collapsing

Mode collapsing is a form of non-convergence of the adversarial network, where the generator learns to map different input value z to a single output value x in the target domain. Indeed, by doing so the generator can yields samples that will be classified as real samples by the discriminator, until the latter learns to reject the mode, forcing the generator to change the mode to map its inputs to. This phenomenon is illustrated in Figure 1, taken from [1].

1.2.3 The side effects of Batch Normalization

Batch Normalization, introduced in 2015 by [3], has become a common practice in deep neural networks. The idea is to reduce the *Internal Covariate Shift*, that is to say the change in the distribution of the inputs of each layer of the network due to the iterative updates of the parameters of the previous layers during training. This is done by introducing normalization layers in the architectures before non-linear layers (such as ReLU and Sigmod) which fix the mean and the variance of each feature in the layer over the batch of training inputs to the network. These normalization layers have themselves parameters that are trained with the other parameters of the network, in order not to change its representation ability. This practice enables to speed up the training by using higher learning rates and be less careful about optimization. It also sometimes eliminate the need for Dropout to prevent overfitting. However this practice in GAN can lead the generator to learn features specific to each learning batch.

2. Unpaired Image-to-Image Translation with CycleGAN

We will now see how the authors of [7] applied the GAN framework to the task of unpaired image-to-image translation.

2.1. Objectives

Image-to-image translation is a common task in computer vision where one wants to learn the mapping between images from two different domain X and Y, on which the training data follows two different distribution p_X and p_Y. The goal is therefore to learn a mapping $G : X \longrightarrow Y$, matching inputs from the source domain X with output that seems to follow the same distribution as the target domain Y. As the generator will be trained using training images from both domains but without any example of mappings between any target images and source images, this image-to-image translation is qualified as unpaired.

2.2. Formulation in the GAN framework

The formulation of the objective of unpaired image-to-image translation obviously calls for the use of Generative Adversarial Network, as we are trying to train a model to generate outputs in a target domain from inputs in a source domain. As presented in section 1, the aims is thus to learn a generator $G_X : X \longrightarrow Y$ and a discriminator $D_Y : Y \longrightarrow [0, 1]$ to approach the objective $min_{G_X} max_{D_Y} \mathcal{L}_{GAN}(G_X, D_Y, X, Y)$ by Stochastic gradient descent.

Note that even though we do not know any mapping between training images from X and training images Y, the model use the knowledge of which images are actual training images from Y and which images have been generated by G_X, in order to compute the previous GAN loss.

2.3. Cycle Consistency

We have seen that one of the major issues of GANs is non convergence, and especially the case of mode collapsing. To overcome this issue, the authors introduce another GAN, that is to say another generator $G_Y : Y \longrightarrow X$ and another discriminator $D_X : X \longrightarrow [0, 1]$, and train them similarly to the other GAN, that is to say by sotchastic gradient descent to approach the objective $min_{G_Y} max_{D_X} \mathcal{L}_{GAN}(G_Y, D_X, Y, X)$. Moreover, the author introduce a cycle consistency loss to enforce each generator to be the inverse of the other: $G_Y(G_X(x)) \approx x$ for all x in the source domain (forward cycle consistency) and $G_X(G_Y(y)) \approx y$ for any y in the target domain (backward cycle consistency). The total cycle consistency loss is given by the sum of two cycle consistency losses:

$$\begin{aligned}\mathcal{L}_{cyc}(G_X, G_Y, X, Y) =& E_{y \sim p_{data}(Y)} [\| G_X(G_Y(y)) - y \|_1] \\ &+ E_{x \sim p_{data}(X)} [\| G_Y(G_X(x)) - x \|_1]\end{aligned}$$

The full loss of the model is therefore the (weighted) sum of the six losses functions: two generator losses, two discriminator losses and two cycle consistency losses. As shown in [7], the ablation of one of these losses leads to poor results. For instance, the ablation of the cycle consistency can lead to two different images translated in one same image in the target domain, even though this image indeed look indistinguishable from training images of the target domain.

2.4. Dealing with oscillations

The authors of [7] apply two techniques to stabilize the model training procedure. First they replace

Figure 1. An illustration of mode collapsing in two dimensions. Here the target distribution is a mixture of Gaussians. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. Images from Metz et al. (2016)

the negative log likelihood objective in the expression of $\mathcal{L}_{GAN}$ by a least square (l_2) loss, that does not suffer from the same saturation issues as the log function does, as explained in [4]. In our case, it boils down to minimizing $E_{y\sim p_{data}(Y)}\left((D_Y(y)-1)^2\right) + E_{x\sim p_{data}(X)}\left[D_Y(G_X(x))^2\right]$ (that is to say, minimizing the error of the discriminator) instead of minimizing $-E_{y\sim p_{data}(Y)}(logD_Y(y)) - E_{x\sim p_{data}(X)}[log(1-D_Y(G_X(x)))]$ (that is maximizing the log likelihood of the discriminator parameters). Similarly, for the generator, it boils down to minimizing $E_{x\sim p_{data}(X)}\left[(1-D_Y(G_X(x)))^2\right]$ instead of minimizing $-E_{x\sim p_{data}(X)}log(D_Y(G_X(x)))$.

The other method used to improve the stability of the training procedure comes from [5]. In the basic GAN framework, the discriminator is updated using only the gradient of its negative log likelihood (or the least square error in our case) on a (random) set of training images and a random set of images generated by the last version of the generator. However, this may obstruct the model convergence as the discriminator has no memory of the images generated by the previous versions of the generator, allowing the latter to re-introduce artifacts that had already caused previous rejection by the discriminator. In the modifications proposed in [5], a buffer of B previously generated images is kept and the discriminator is updated using both images generated by the current generator version and images from the buffer, a fraction of which being replaced with new generated images after the update.

2.5. Instance Normalization

The authors chose to implement both batch normalization and instance normalization in their architecture, with the latter being the default setting (that can be changed thanks to a option tag when calling the training and the testing methods of the authors code). As explained in [6], Instance Normalization aims at improving the training efficiency by removing instance-specific contrast information from the content image. The means and covariance of the inputs to the normalization layer are no longer computed across the mini batch of input images, but are computed across only one instance (image). The training process will hence be less prone to taking into account features specific to each input mini batch.

3. Experimenting the limits and abilities of CycleGAN

After trying to reproduce the authors results, I performed several experiments on the CycleGAN architecture in order test the limits and abilities of the network. All the following trainings and tests of the architecture were performed using an AWS p2.xlarge instance with 2Gb GPU capacity. I used the authors' Pytorch implementation of CycleGAN proposed on their project web page.

3.1. Reproducing the author results

I first tried to train the Vangogh to photographs image translation model. I used instance normalization just like the authors, but, due to the limited amount of GPU resources I had at my disposal (even for 128x128 images), I only trained it for 20 epochs, 10 with a constant learning rate and 10 epochs with a decaying learning rate (to 0). The training set was the one used and provided by the authors, with 400 Vangogh paintings and 6287 landscape paintings. I reported some of my results in Figures 2 and 3, along with the images obtained from applying the pre-trained from the author website. Surprisingly, my results compare with those from the author, especially in the Vangogh to photograph translation. I think there may be an error in the pre-trained model, it seems to turn Vangogh paintings into even more Vangogh-like paintings.

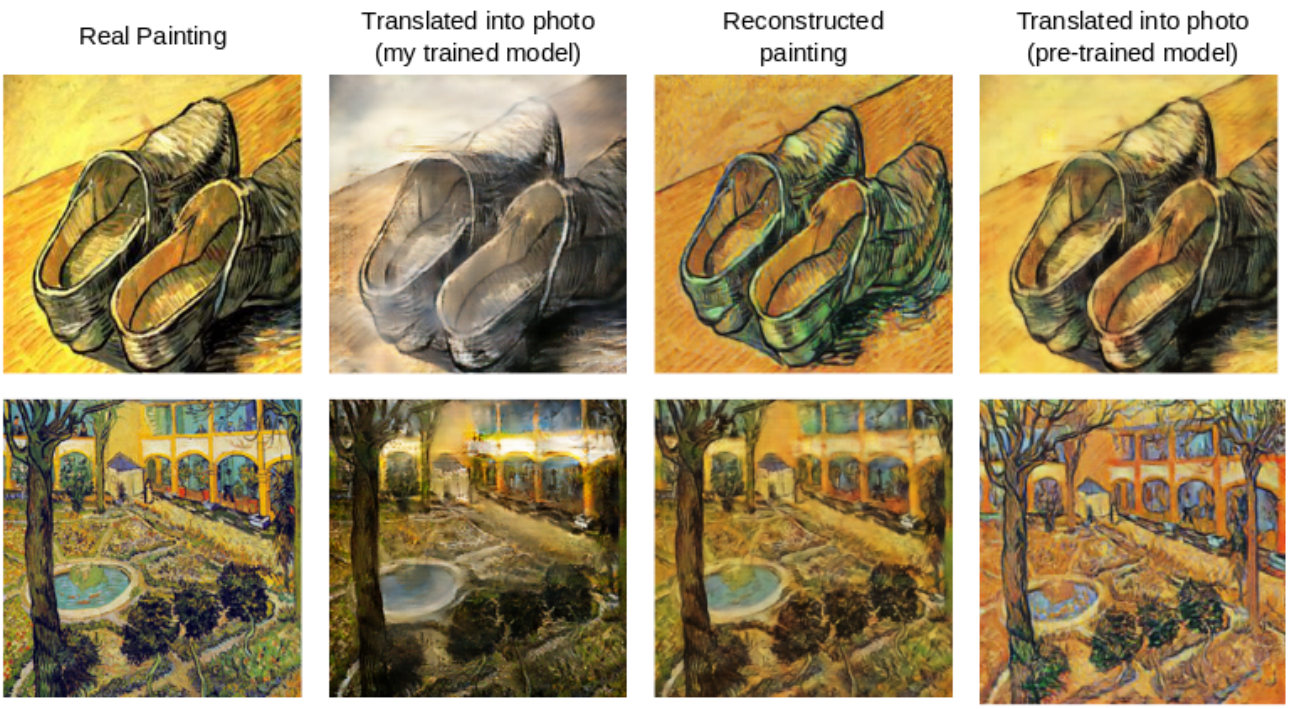

Figure 2. Image to image translation from Vangogh paintings to photographs.

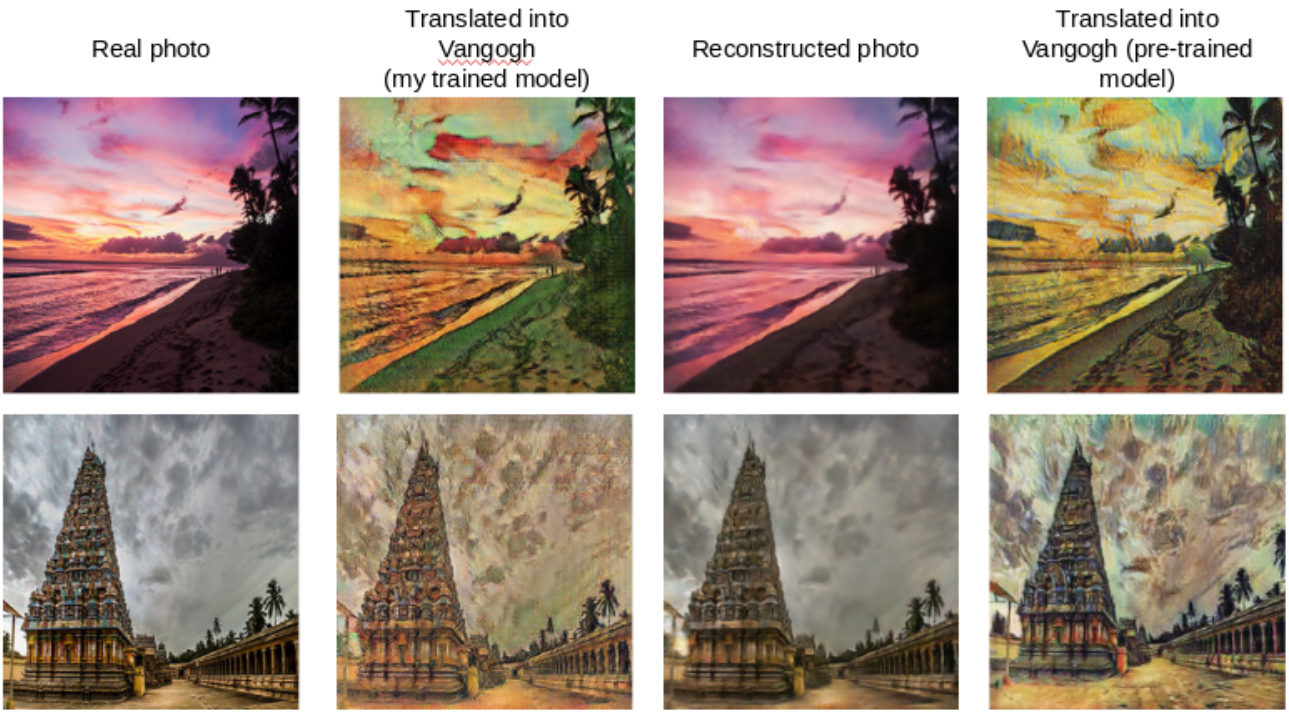

Figure 3. Image to image translation from photographs to Vangogh paintings.

3.2. Shape modification in Image-To-Image translation

The authors noted that their architecture usually failed in tasks requiring geometric changes between the two domains. I thus trained their model on two toy data sets: binary images of circles (for one domain) and squares (for the other). Both training sets hold 1000 images. I trained the model for 100 epochs (50 with a constant learning rate, 50 with a decaying learning rate). Figure 4 shows some of the results I got. We see that even though the results can be quite compelling for the training images (although a too big circle will systematically tend to be mapped to a black image), we almost never get credible translation in the test set. I assume this may be due to an overfitting issue that could be solved either by resorting to drop out or to use larger training sets. As a matter of time and resources, I did not get to verify one of these two assumptions.

3.3. Color to Gray translation

I finally applied the CycleGAN architecture to a common task in computer vision: gray scale image colorization. I trained the model with 1000 images (500 gray scale images and 500 color ones).

3.3.1 Base results

Figure 5 and 6 shows training and test results I got when training the architecture with instance normalization for 100 epochs (50 with a constant learning rate, 50 with a decaying learning rate). Note how compelling the gray to color translation on the test image is (more that the training one). Color to gray translation usually works well, as it is an easy task (the generator is actually only trying to average the actual function giving the gray scale values as a linear combination of the color values).

3.3.2 Impact of the training schedule

In Figure 7 and 8 is illustrated the impact of the modification of the training schedule on the quality of the results. Indeed, the fourth column of both figures were obtained by having a constant learning rate during all the training of the model (instead of having 50 epochs with a constant learning rate, and 50 epochs with a decreasing learning rate), while the fifth column were obtained by decaying the learning rate all along the training. We can see that the model trained with no learning rate decay tends to apply brightest colors (especially blue) than the default implementation in the gray-to-color translation, while the model trained with a decay of the learning rate from the very beginning of the training tends to apply dimmer colors. In the color-to-gray translation, the no-decay model usually leads to the apparition of straight lines close to the border of the image, which is a phenomenon pointed out in [6]. While the model with no learning rate decay usually gave unrealistic results, the model without a phase of constant learning rate outperformed the default implementation.

3.3.3 Impact of Instance Normalization

In Figure 7 and 8 is illustrated the impact of the replacement of Instance Normalization (default implementation) by Batch Normalization. Indeed in the third column of both figures, we can see that batch normalization leads to slightly less realistic images than instance normalization, with less realistic colors in the gray-to-color translation and with occasional apparition of straight lines close to the border of the images in the color-to-gray translation, to a lesser extent that with the model learned with no decay of the learning rate.

3.3.4 Impact of the least square loss and generated image buffer

The images of the last column of Figures 7 and 8 were obtained by training the model without using the two techniques used in section 2.4. We can see that it indeed leads to images with artifacts in both translation directions.

4. Conclusion

I have presented my work on Image-To-Image translation using CycleGAN and shown what this model was capable of and why.

References

[1] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[4] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang. Multiclass generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[5] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016.

[6] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[7] 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.

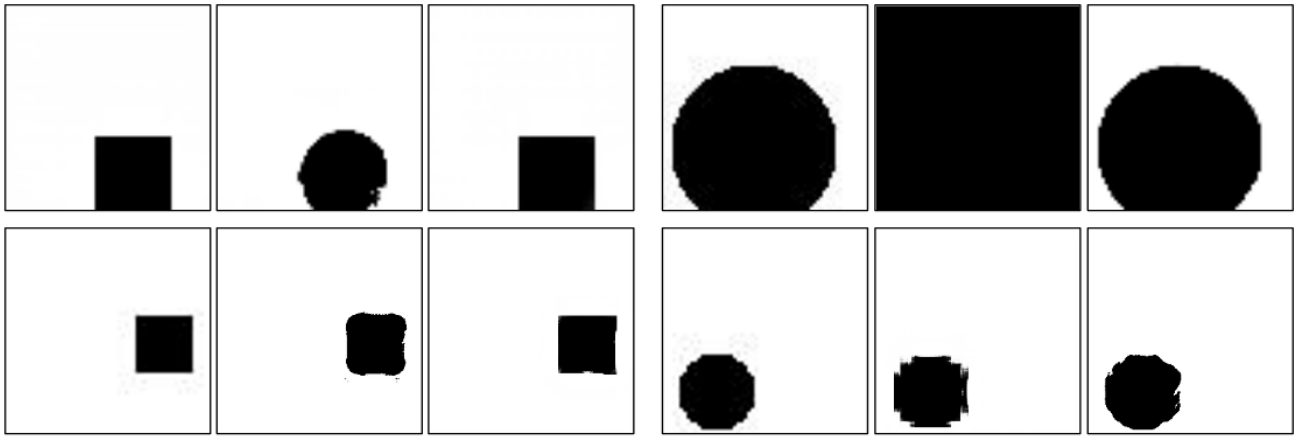

Figure 4. Image to image translation between square and circle binary images. First row: actual, generated and reconstructed (obtained by applying the second generator to the generated image) of **training** square and circle images respectively. Second row: actual, generated and reconstructed (obtained by applying the second generator to the generated image) of **test** square and circle images respectively.

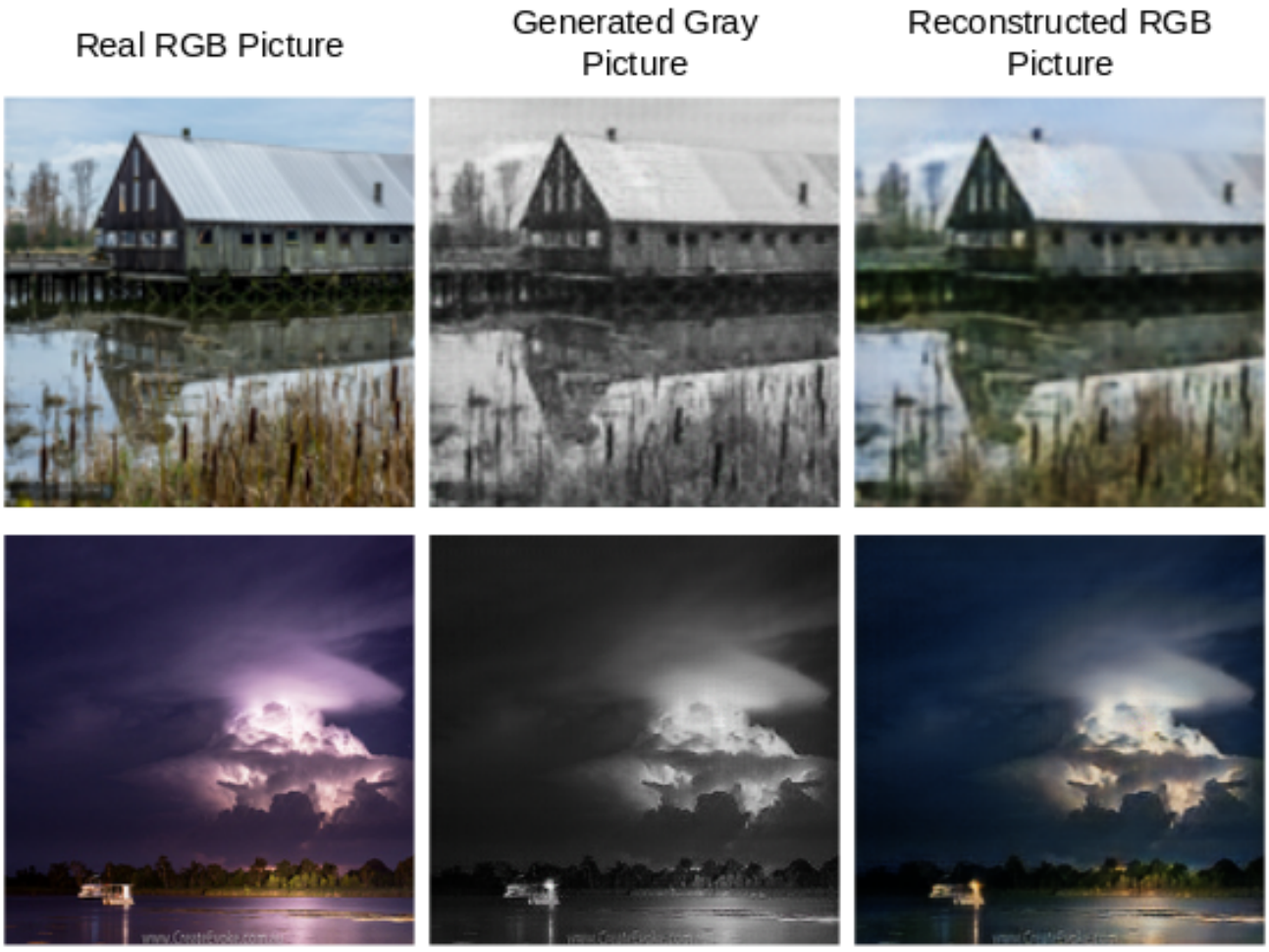

Figure 5. Image to image translation from color to gray images. First row: results on **training** image. Second row: results on **test** image

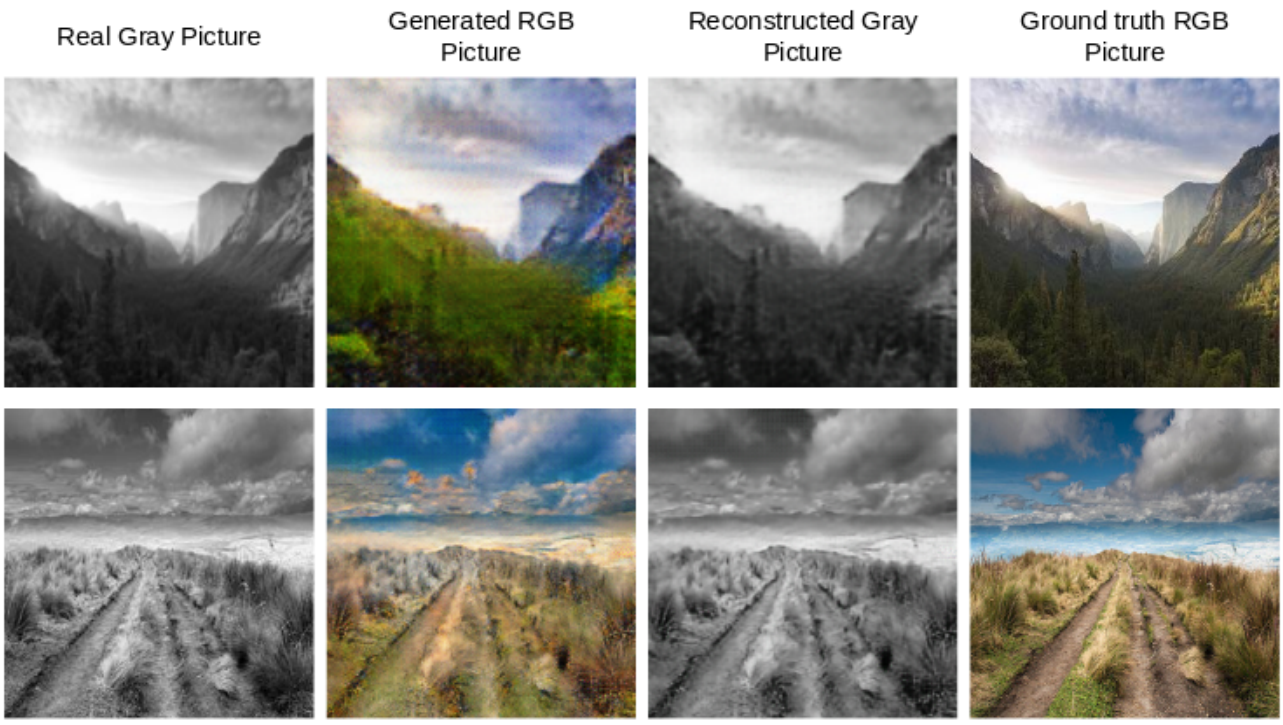

Figure 6. Image to image translation from gray to color images. First row: results on **training** image. Second row: results on **test** image

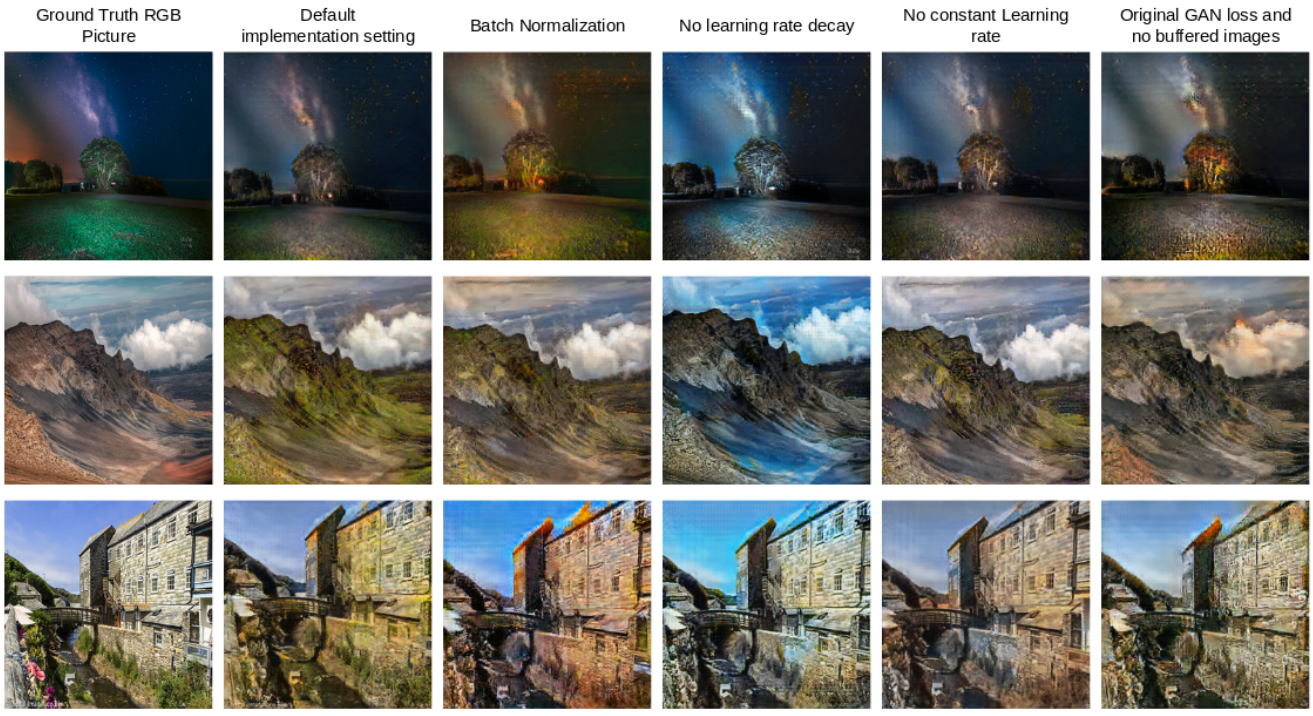

Figure 7. Image to image translation from gray to color images for various modification of parameters in the default authors' implementation (second column).

Figure 8. Image to image translation from color to gray images for various modification of parameters in the default authors' implementation (second column).

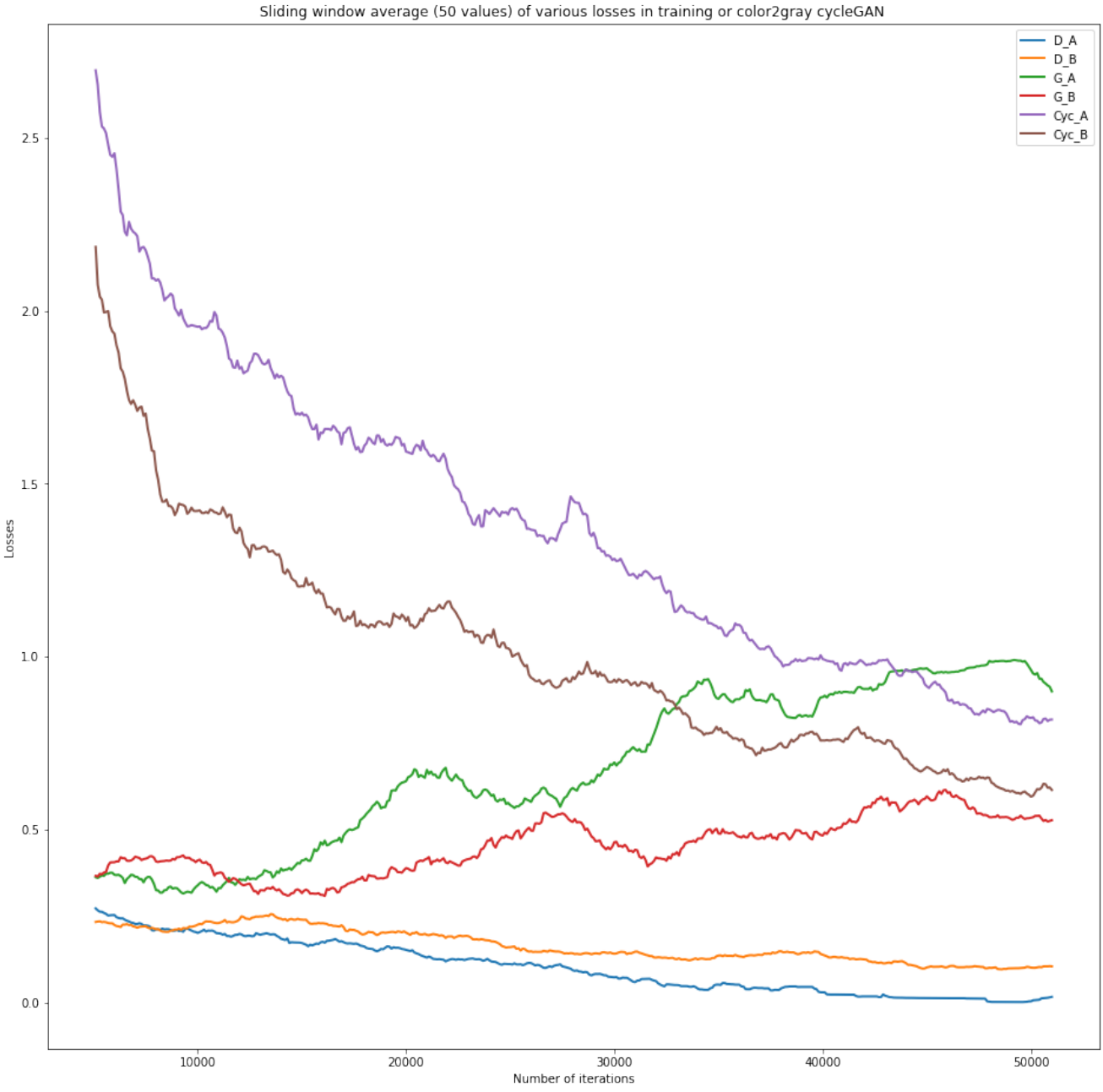

Figure 9. Sliding window average of the loss functions during the training of the Cycle on the Color-To-Gray Translation task.

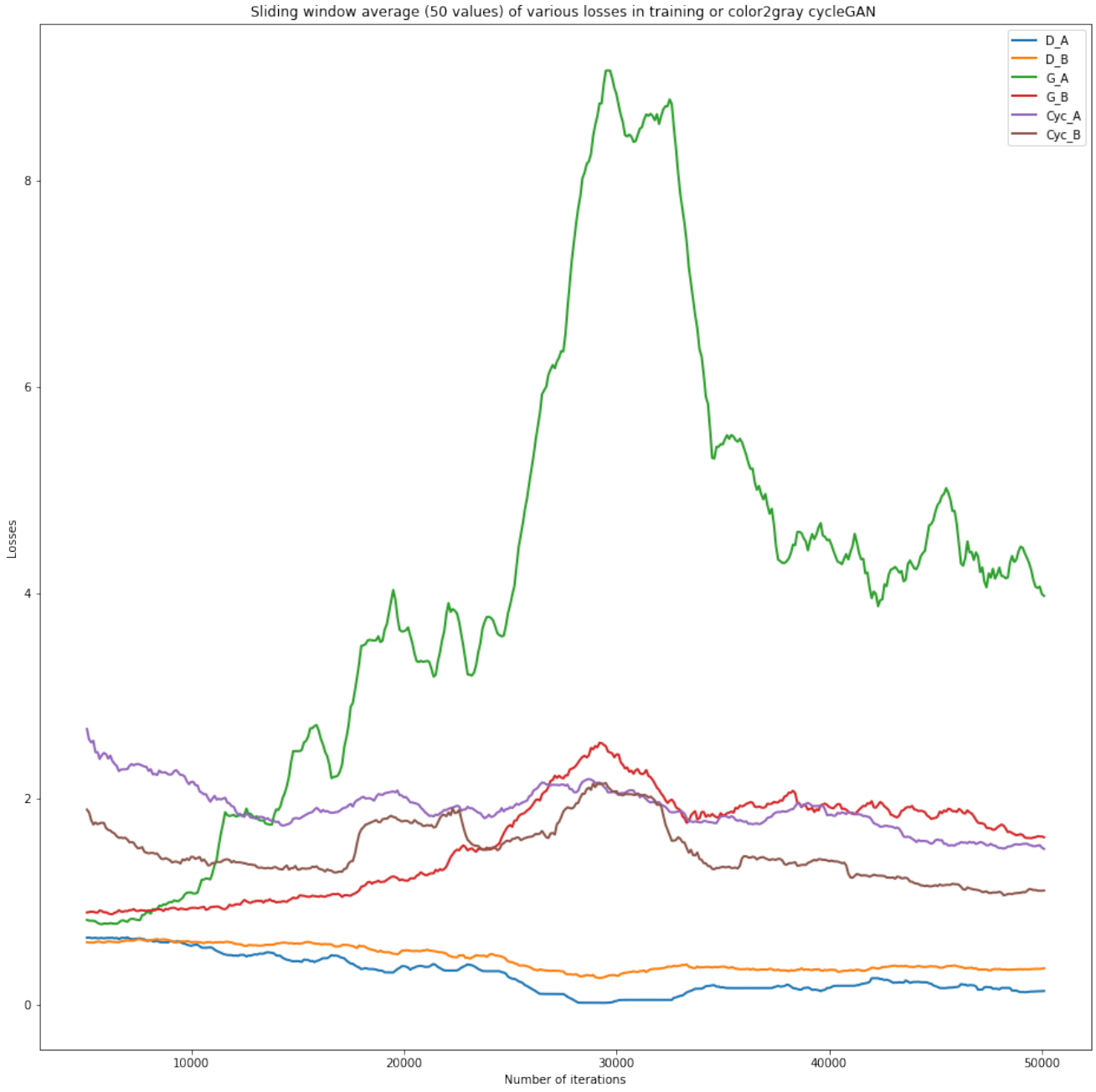

Figure 10. Sliding window average of the loss functions during the training of the Cycle on the Color-To-Gray Translation task with the negative log likelihood losses and no buffer of previously generated images. Note that the training process is clearly less stable than in the authors' implementation.