Style-transfer using CycleGAN and its Variants

CHEN Xiaoqi
xchenfb@connect.ust.hk
20784638

CHOI Jang Hyeon
jhchoiac@connect.ust.hk
20783567

LU Fei
fluac@connect.ust.hk
20797386

Contributions	
Data augmentation	CHEN Xiaoqi
U-network with residual blocks	LU Fei
BCE, MSE, Huber loss	
Weight clipping	LU Fei & CHOI Jang Hyeon
Wasserstein, identity loss	CHOI Jang Hyeon
Gradient penalty	
Hyperparameter tuning	

***Abstract* – The painter Monet's painting style can be transferred on to photos using style-transfer via Cycle Generative Adversarial Networks. CycleGAN's network structure and multiple loss functions have been experimented with to find the optimum architecture.**

Keywords – data augmentation, U-net, identity loss, binary cross entropy loss, mean squared error loss, Wasserstein loss, Huber loss, gradient penalty, weight clipping

I. TASK DESCRIPTION

Human beings can create art, so can Neural Networks. We use GANs to create pictures in the style of Claude Monet (1840 –1926), a French painter and founder of impressionist painting. In this project, our task is to transfer photos into his unique artistic style and trick classifiers into believing these photos are true Monet paintings.

This task is inspired by a Kaggle competition: "I'm Something of a Painter Myself". We learned the baseline model CycleGAN and made improvements by data augmentation and by using different Neural Network structures, different loss functions, and comparing Adam and RMSProp, and also tuning many other hyperparameters as well.

II. DATASET

The dataset contains 300 Monet paintings sized 256x256, which are critical to help models to learn the unique style of Monet. And there are also 7,028 photos of the same size in the dataset which are normal pictures. In the experiment, we make use of these 7,028 pictures and transform them into Monet-style images.

Figure 1 Sample Monet paintings

III. INTRODUCING GAN AND CYCLEGAN

Generative Adversarial Networks(GAN) is composed of two parts, a generator and a discriminator.

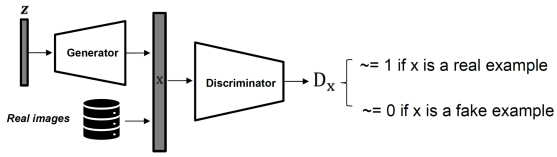


Figure 2 Generative Adversarial Networks

Generator tries to create images, while the discriminator will try to recognize whether the images are real ones or fake ones. The output D(x) is the probability of x being a real example. If x is a real example, D(x) will be close to 1, otherwise, D(x) will be close to 0. Cost function for the discriminator is to use the cross entropy cost:

$$J = -\frac{1}{m}\sum_{i=1}^{m} logD(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} log\,(1 - D(g(z^{(i)}))$$

In this formula, x are real examples, and D(x) should be one, while g(z) are fake examples, so D(g(z)) should be 0. So to train Discriminator, cost function should be minimized.

And for Generator, it tries to generate images that are very similar to the real ones, so that they can fool the Discriminator. In other words, the generator wants to make D(g(z)) to be 1, and this cost function should also be minimized:

$$\sum_{i=1}^{m} log\,(1 - D(g(z^{(i)})))$$

For Cycle GAN, there are two mapping functions G and F.

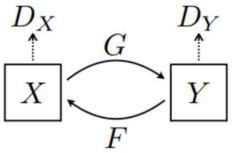


Figure 3 CycleGAN

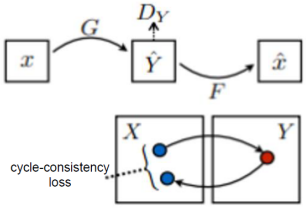
For the mapping function G, and its discriminator D_y, G tries to generate images that are similar to the real images, while D_y aims to distinguish between generated examples and real examples. So, this part is very similar to GAN. Adversarial loss is as below:

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

The goal for G is to minimize this adversarial loss while D wants to maximize it. For mapping function F, it is also similar. What's different in CycleGAN, is that CycleGAN introduces cycle-consistency loss:

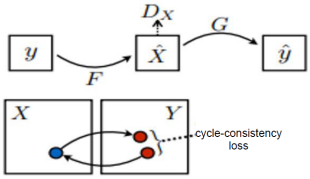
$$L_{cyc}(G, F) = E_{x\sim p_{data}(x)}[\|F(G(x)) - x\|] + E_{y\sim p_{data}(y)}[\|G(y) - y\|]$$

For each image x using mapping function G generates an image from domain Y. And then using mapping function F, it should create images back to domain X. This is called forward cycle consistency. Similarly, for each image y from domain Y, the image translation cycle should be able to bring y back to the original image. And this process is called backward cycle consistency. We use a cycle consistency loss to implement both forward and backward cycle consistency. F(G(x)) should be similar to x. G(F(y)) are supposed to be close to y. So this loss function could be minimized to 0.



Forward cycle consistency:

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



Backward cycle consistency:

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

Figure 4 Forward and backward cycle consistency

IV. DATA AUGMENTATION

While the size of the dataset for Monet was 1,073 in Zhu's experiment, our dataset with only 300 Monet images is very small. Consequently, the performance of our models may deteriorate once the discriminator memorizes the exact training set. Facing this problem, data augmentation is a good choice to increase training data, which is efficient and improves the performance of our models.

Data augmentation is essential to help increase the diversity of the dataset by applying random transformations, such as rotation, flipping, cropping, noise injection etc. In the experiment, we mainly do the following two operations for data augmentation.

For one thing, we do random cropping for each image, that is, randomly removing the outer areas of each image. Firstly, extend each image to be 282x282, and then randomly crop it back to its original size: 256x256. After this processing, while most of the image pattern does not change, some areas are removed, and the models will take transformed images as new ones to learn.

Another augmentation operation is flipping. Each image will be flipped along the width dimension. Although the image pattern is changed from left to right after flipping, the intensity of each pixel remains the same, which means it is almost lossless for the image.

Both flipped images and cropped images are added to the original dataset. And thus the dataset becomes 3 times as large as before.

V. NETWORK STRUCTURE

Cycle GAN contains two generators and two discriminators.

A. Generator

In our work, the U-net [Ronneberger, O] structure is utilized as the generator of cycle-GAN. U-net was first developed for biomedical image segmentation in 2015. It consists of a contracting path that follows the typical architecture of a convolutional network and an expansive path that contains up-convolution and concatenation. Our network is constructed in a similar way which is shown as follow:

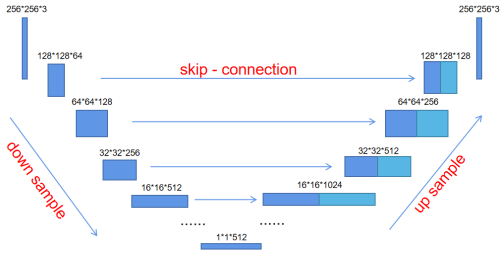


Figure 5 Generator Structure

First, the input image is down-sampled through several convolution layers, and instance normalization is applied after each Conv-layer except the first one. Then the (1, 1, 512) size output of down-sampling is up-sampled through transposed convolution layers. Skip-connection is performed during the up-sampling, which means the output of up-sampling is concatenated with the output of former down-sampling in a symmetrical mode. This helps to combine the low-level features with high-level features to improve the accuracy of the generator.

Besides, we also tried to add residual blocks in our generator model. A residual block [He, K.] is constructed with two Conv-layer and ReLU in between, and the identity input is added after two convolution transformations before another ReLU activation. The Residual block helps with the vanishing gradient problem. The structure is shown as follow:

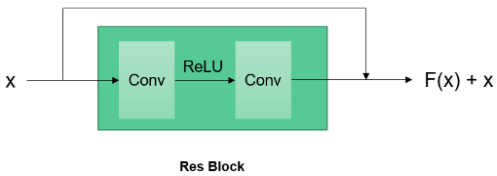


Figure 6 Residual Block Structure

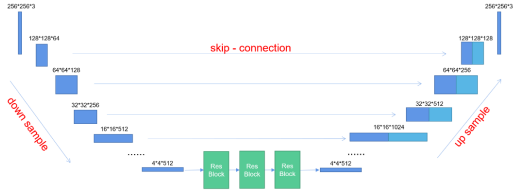


Figure 7 Generator Structure with Res block

As shown in the generator structure, the output of down-sampling is passed by 3 Residual blocks and up-sampled back to the original size.

B. Discriminator

As for the discriminator, Patch-GAN is utilized. Instead of outputting a single value that indicates if the input image is real or generated, the discriminator output an N*N matrix. Every point in the output matrix indicates a patch of the input image is real or generated. The structure is shown as follow:

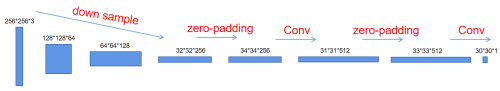


Figure 8 Discriminator Structure

The input image is down-sampled to a (32, 32, 256)-size matrix, after two times zero-padding and convolution transformation, the final output of the discriminator is a (30, 30, 1)-size matrix. Every point in the output matrix indicates a patch of the input image is real or generated.

VI. U-NET RESULTS

In our work, we applied the two structures mentioned above and compared the results under the same 25 epochs training. The outputs are shown as follows: (we evaluate our model through the Kaggle score system (the lower score indicates the better performance).

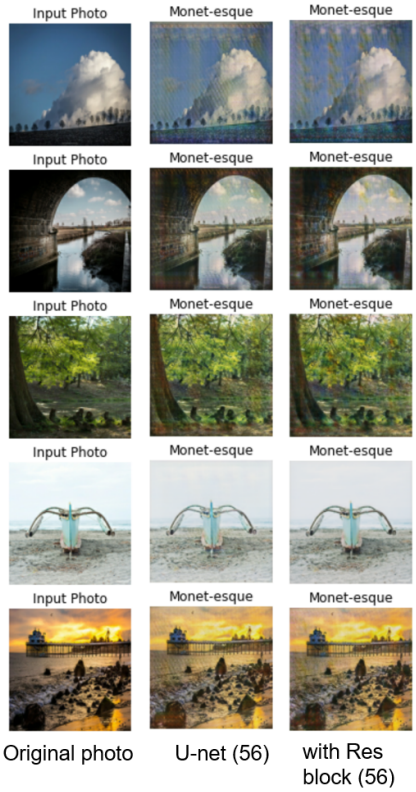


Figure 9 Outputs of two network structures

It seems the residual block added did not improve the final score, more training iterations and parameter tuning should be applied in the future work.

VII. TESTING DIFFERENT LOSS FUNCTIONS

In our work, different loss functions are tested and the performance are compared.

A. Mean Square Error

MSE is tested to calculate the loss of generator and discriminator both. After 25 epochs of iteration, it gets 66 points in the end.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

B. Huber loss

Huber loss is an improvement of MSE, it enhances the robustness of noises. After 25 epochs of training, the result is 64.5 points. It works slightly better than MSE.

$$\text{Huber} = \begin{cases} \frac{1}{2}(y-y')^2, |y-y'| \le \delta \\ \delta(|y-y'| - \frac{1}{2}\delta), otherwise \end{cases}$$

C. Binary Cross Entropy

Binary Cross Entropy is calculated for both generator and discriminator. After 25 epochs, the result is 56.

$$BCE = ylog\,(p(y)) + (1-y)log\,(1-p(y))$$

Loss between the generated photo and a same size all-ones matrix is treated as the generator loss. (F indicates different loss functions)

$$L_{Gen} = F(1, Fake)$$

Loss between the real photo and a same size all-ones matrix, plus the loss between the generated photo and a same size all-zeros matrix is seen as the discriminator loss.

$$L_{Real} = F(1, Real)$$
$$L_{Fake} = F(0, Fake)$$
$$L_{Disc} = \frac{L_{Real} + L_{Fake}}{2}$$

(Note: the 1 and 0 here indicate all-zeros matrix and all-ones matrix)

VIII. LOSS FUNCTION RESULTS

The outputs and corresponding scores of using different loss functions are shown as follows:

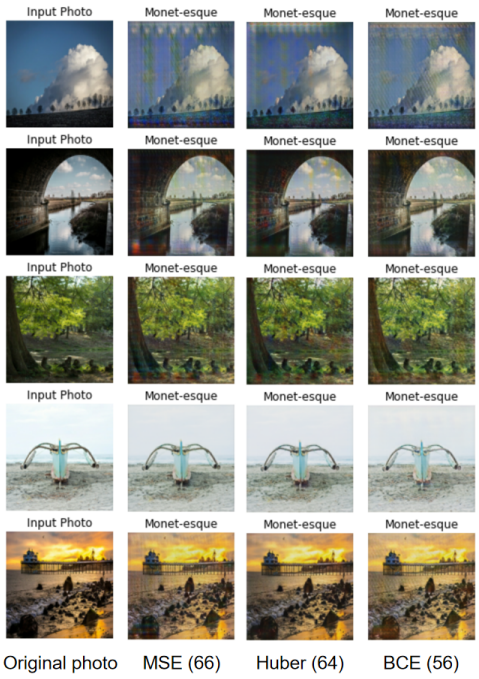


Figure 10 Outputs of different loss functions

A. Analysis

MSE and Huber loss focus on the difference between the predicted value and the true value of each output point, and Huber loss is less sensitive to outlier data, thus it is more robust and shows a better performance than MSE. BCE focuses on the accuracy of prediction. It converges faster since it only focuses on the right estimate points, and when a wrong estimate is made, the high BCE gives a penalty to the model. That is why after the same round of training, the model with BCE as loss

function gets better performance.

IX. WASSERSTEIN LOSS FUNCTION

The paper [Hu et al.] modifies the loss function from the summation of expected values of a logarithmic function to simply the difference between expected values of $D_Y(y)$ and $D_Y(G_x(x))$, where y is the original dataset image and $G_x(x)$ is the generated image:

$$L_{WGAN}(G, D_Y, X, Y) = E_{y \sim p_Y(y)}[D_Y(y)] - E_{x \sim p_X(x)}[D_Y(G_x(x))]$$

$$L_{WGAN}(G, D_X, X, Y) = E_{y \sim p_Y(y)}[D_X(x)] - E_{x \sim p_X(x)}[D_X(F_Y(y))]$$

The corresponding Python code to reflect the Wasserstein Loss is:

```
def discriminator_loss(real_monet, generated_monet):
        return backend.mean(real_monet)-
        backend.mean(generated_monet)

def discriminator_loss(real_photo, generated_photo):
        return backend.mean(real_photo)-
        backend.mean(generated_photo)
```

A. Complete Adversarial Loss

The complete adversarial loss is, thus, the Wasserstein loss plus the cyclic loss:

$$L(G, F, D_X, D_Y) = L_{WGAN}(G, D_Y, X, Y) + L_{WGAN}(G, D_X, X, Y) + \lambda_0 L_{cyc}(G, F)$$

X. IDENTITY MAPPING LOSS

Including identity loss helps the neural network to adjust for any deviation it takes from the original image [Zhu et al.]. A CycleWGAN without identity loss tends to output images with a different tint as shown in the outputs below under "Weight clipping versus gradient penalty". Based on the paper of Zhu et al., the formula for the identity loss is the following:

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

To elaborate, the identity loss is equal to the expected value of the difference between a generated image and the real image in addition to the expected value of the difference between a generated photo and the real photo, where G(x) = y and F(y) = x. In Python, the identity equation would translate to:

```
def identity_loss(real_image, same_image, LAMBDA):
            loss = tf.reduce_mean(tf.abs(real_image -
            same_image))
            return LAMBDA * 0.5 * loss
```

(The *same_image* is the generated image and the *same_photo* is the generated photo.)

During experimentation, the identity loss is derived by taking the mean of the difference between the input image and a generated image. The loss is then multiplied by a constant (0.5 in this case) for the best results.

XI. WEIGHT CLIPPING VERSUS GRADIENT PENALTY

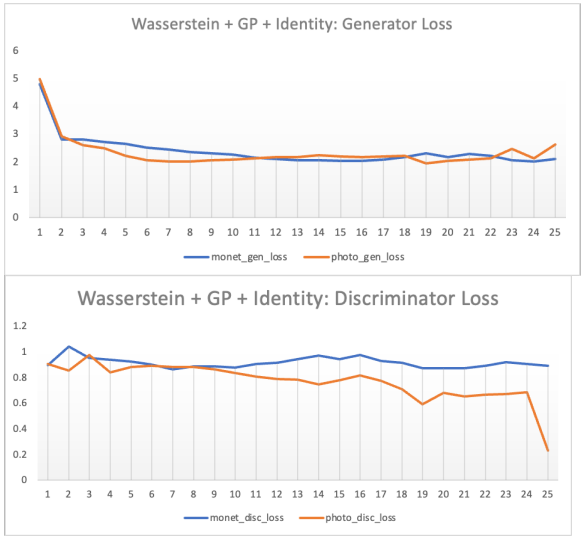
Weight clipping was used by Hu et al. with the Wasserstein loss function. It implements a 1-Lipshitz constraint to improve the robustness of the neural network and stabilizes the GAN training [https://link.springer.com/article/10.1007/s00521-020-04954-z]. It has been observed during experimentation that, without weight clipping, gradient explosion would occur, dramatically increasing losses during training and worsen the network's performance.

However, according to another paper, [Gulrajani et al.], weight clipping did not bring satisfactory performance and introduced the concept of gradient penalty for the discriminator. The following is the equation for gradient penalty:

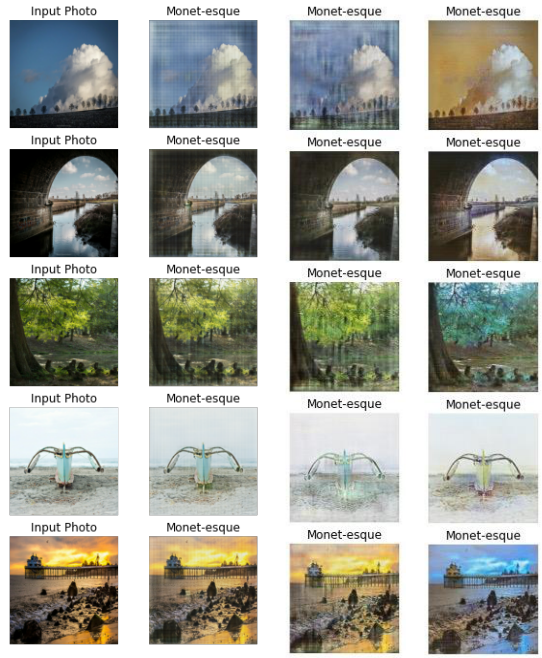
$$E_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D_Y(\hat{x})\|_2 - 1)^2]$$

where $\hat{x}$ is a selection from a distribution of both real and generated images $P_{\hat{x}}$.

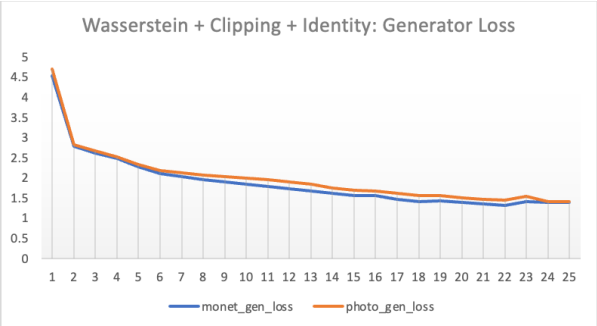
Cycle WGAN was experimented with both weight clipping and gradient penalty.

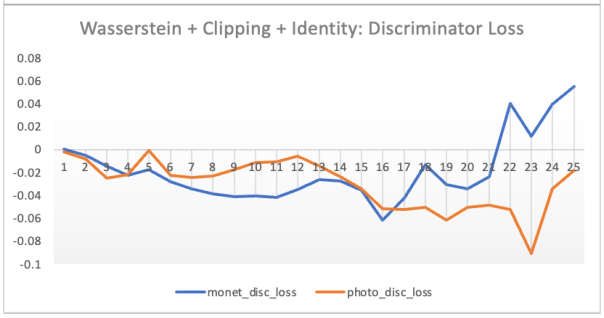


(Loss per epoch for the Cycle WGAN with gradient penalty and identity loss)

Input	Weight Clipping + identity loss	Gradient Penalty + identity loss	Gradient penalty – identity loss
			
Scores (lower is better)	70	62	121

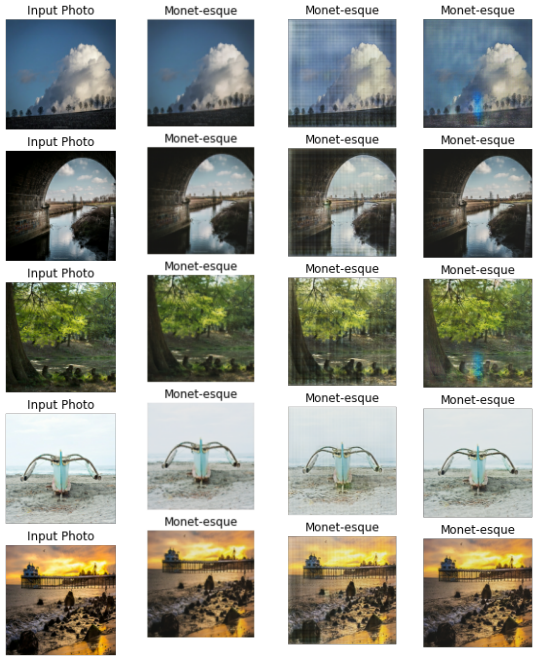
The loss line-graph shows how the general trend of the Monet generator network loss decreases throughout the epochs. The photo generator network initially decreased but did not improve much after the 10th epoch. On the other hand, the Monet discriminator loss remained stable, indicating that it has reached a local minimum, and the photo discriminator started to decrease only after the 10th epoch.





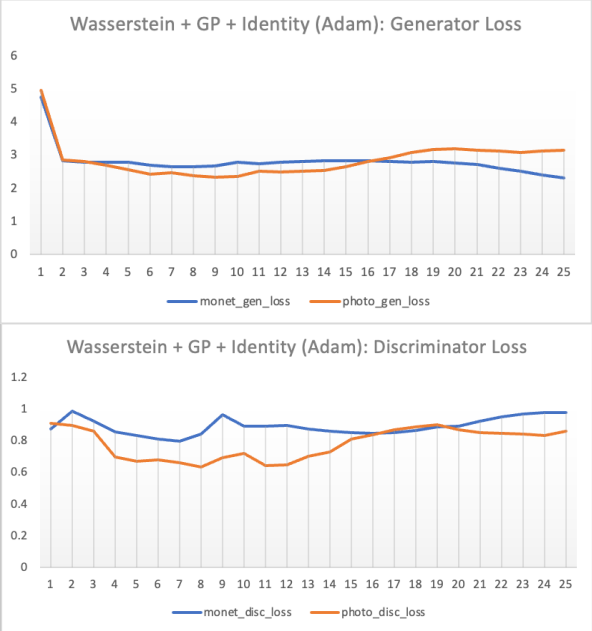
(Loss per epoch for the Cycle WGAN with weight clipping and identity loss)

Both the Monet generator and photo generator networks showed a decrease in their losses, while their discriminator counterparts showed relatively smaller changes. In general, applying gradient penalty instead of weight clipping provided a better loss value and, thus, better performance of the network model.

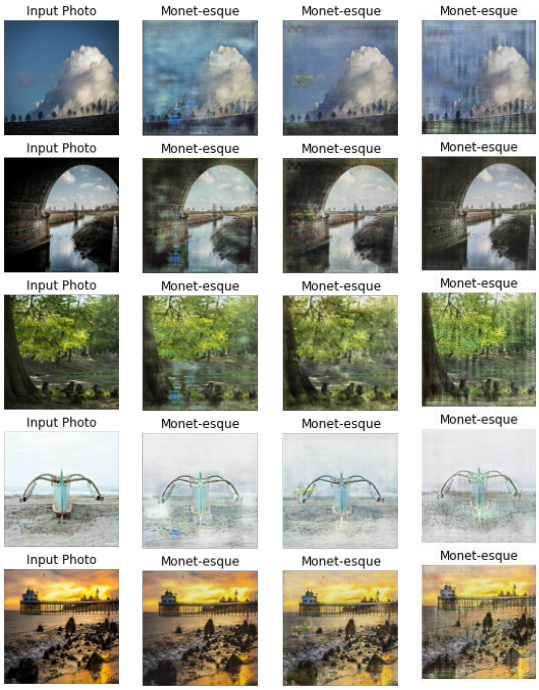
Input	No Clipping	Clipping = 0.01	Clipping = 0.5
			
Scores (lower is better)	Not scored	70	72

XII. OPTIMIZERS: ADAM VERSUS RMSPROP

Hu et al. suggested using the RMSProp optimizer when implementing Cycle GAN with Wasserstein loss and weight clipping, while the Adam optimizer was more suitable for the original GAN and WGAN with gradient penalty architecture. After multiple trials, using the Adam optimizer with a learning rate of 5e-6 and a beta_1 value of 0.8 gave the best results on the Cycle WGAN with gradient penalty. Higher learning rates resulted in negative losses (e.g. -2000).



(Loss record using the Adam optimizer (d=5).)

Input	Adam (d=1)	Adam (d=5)	RMSProp (d=5)
			
Scores (lower is better)	60.4	56	62

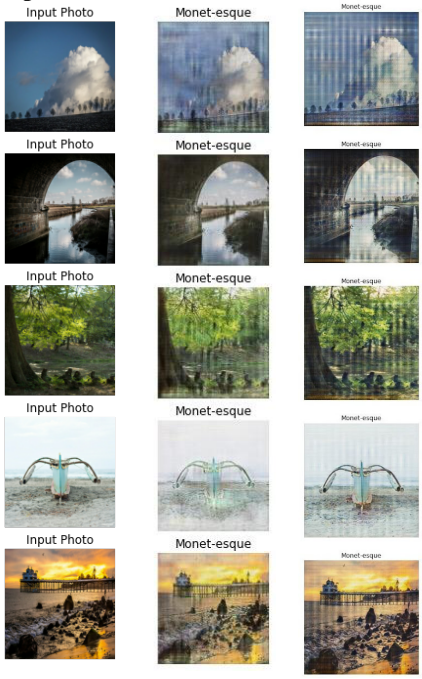
The value *d* is the number of times the discriminator network is trained every time the generator network is trained. When the value of d is 5, then this means that the discriminator has been trained five times before the generator network was trained once.

According to the experiment results, the CycleWGAN with gradient penalty had the best score using the Adam optimizer and when the discriminator was trained for five times more than the generator was. When the *d* value was further increased to 10, the losses were very negative.

The performance difference between the three settings is most evident in the first and fourth images in a column. Adam (d=5) shows the pictures with the best clarity in Monet-style.

XIII. LAMBDA CONSTANT

The lambda constant is a coefficient that is multiplied to various losses in the network, i.e. forward cycle loss, backward cycle loss, and identity loss. The lambda value helps put emphasis on losses that are important to the learning of a network, and a value of 10 showed the best results. A network with the lambda value of 30 was not able to modify the original input image but only add an overlay of some sort.

Input	Lambda = 10	Lambda = 30
		
Scores (lower is better)	62	130

XIV. CITATIONS

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio. Y. (2014). Generative adversarial nets. In NIPS.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Hu, W., Li, M., & Ju, X. Improved CycleGAN for Image-to-Image Translation.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In Advances in Neural In- formation Processing Systems, pages 5769–5779, 2017.

Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).