

Style Transfer with CycleGANs and Autoencoders

Jan-Henner Roberg, S3228851 Charlie Albietz, S3058735
Panagiotis Ritas, S3152529

*Faculty of Science and Engineering
University of Groningen*

Abstract

This project investigates different deep-learning techniques for style transfer. Cycle GANs are the state-of-the-art technique for this task, we extend this by expanding by using the Wasserstein loss with and without gradient clipping. In addition, autoencoders using content and style loss are presented, as well as autoencoders trained using the images created by GANs. The results show that cycle GANs subjectively perform best, however, the style transfer autoencoder gave clearer images. Using regular autoencoders with a Transfer loss managed to perform as well as the model that supplied the images.

1 Introduction

In this project different deep-learning techniques to perform style transfer are investigated. In particular, the task is to take a photo and transfer the style to make it look like a Monet painting. There are several ways of using a neural network to achieve this task.

Generative Adversarial Networks (GANs) were first proposed by Goodfellow et al. [3]. GANs are generative models that consist of two networks engaged in a zero-sum game against each other: the Generator tries to learn the distribution of the input data, while the discriminator tries to maximize the loss, and therefore maximize the difference between the original input and generated distributions [3]. A generative model outputs data that is as close to the distribution of the input data as possible.

Wasserstein GANs, as proposed by Arjovsky et al [1] aim to tackle convergence problems of regular GANs: This is done by implementing the Wasserstein distance function as the model's loss function for the discriminator, as well as changing the discriminator/generator training ratio to 5:1. Doing so ensures convergence stability and a smoother training process for the GAN.[1] In the original paper, weight clipping is enforced after each critic mini-batch training to enforce Lipschitz continuity. Gulrajani et al. [4] argue that this is a non-optimal way of enforcing continuity and can lead to instability/vanishing gradient problems. They proposed imposing a gradient penalty on the critic, based on penalizing the norm of the gradient of the critic with respect to its input.

CycleGANs was first proposed by Zhu et al.,[8]. CycleGANs was proposed as a potential solution towards training GANs with "unpaired" data. In this architecture, GANs learn a function that maps an input image from one source domain to the other. In a CycleGAN a cycle consistency loss function is also introduced that tries to enforce constraints of one-to-one mapping from one domain to another, and then back to the original domain.

Autoencoders In general, an autoencoder is a neural network that is trained to take an input image encode it to some representation and then decode the original image from this representation. As the representation is in a smaller dimensional space than the input and output dimensions, the autoencoder is unable to just learn to copy the image [7]. Usually, the task of an autoencoder is to learn to replicate

an image that is identical to the input image and thus learn the central representation. For this reason, the loss function measures the distance between the input image and the generated image. Compared to this task, style transfer using autoencoders is a rather unexplored area of research.

One way to use an autoencoder for style transfer is to train the network to take the original as input and output the same image but in the desired style. This method is usually not possible as one would need pairs of images that only differ in their style and is often not given as training data. Another way to use an autoencoder network for style transfer is to have one image in the desired style, e.g. a Monet painting, and some images that should be transformed. To make sure that the generated images match both the content of the input image and the style of the model image two loss functions are used; one for content and one for style [5]. This is further explained in the method section.

Problem Description As mentioned in the previous section, it is notoriously difficult to work with GANs. Especially when we aiming for style transfer the most common method is using a CycleGAN. In this project, it is our aim to explore different methods of style transfer in order to change photographs into Claude Monet paintings. We first implement different GANs: we implement a well-performing CycleGAN and then develop our own CycleGAN variations of Wasserstein CycleGANs, one with weight clipping and another with gradient penalty. To the best of our knowledge, there are no Wasserstein implementations of CycleGANs in the scientific literature. Additionally, we extend these models by introducing the gradient norm penalty as proposed by Gularjani et al., in[4].

After implementing the CycleGANs we intend to explore the possibilities of style transfer using autoencoders by implementing models in accordance with those previously mentioned. We aim to obtain a suitable set of image training pairs by using the best performing CycleGANs to generate the Monet-style photos.

2 Methods

Dataset We trained our models on the Dataset given in the Kaggle competition "I'm something of a painter myself". The dataset is comprised of two sets: one set of pictures with images taken from a photograph, as well as a set of Monet paintings, with dataset sizes of 7038 and 300, respectively (total of 7338). In specific, this dataset was used in all CycleGAN models. Figure 1 shows some examples of photos taken from both sets of the dataset.

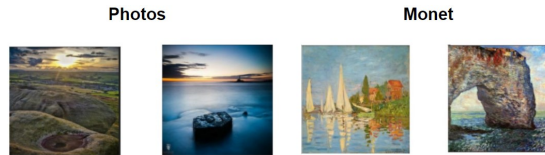


Figure 1: example images of the "I'm something of a painter myself" competition dataset, found on kaggle.

2.1 Models

CycleGAN architecture formulation: Our goal is to learn mapping functions between two domains X and Y , given training samples $x_{i=1..N} \in X$ and $y_{i=1..M} \in Y$. Two Generators are presented that learn the mapping between domains G and F , where $G : X \rightarrow Y$ and $F : Y \rightarrow X$. Additionally, two Discriminator functions are introduced, D_x and D_y . D_x distinguishes between images from the dataset $x_{i=1..N} \in X$ and generated images $F(y)$; D_y distinguishes between images from the dataset $y_{i=1..M} \in Y$ and generated images $G(x)$.

Both Wasserstein CycleGAN models are built on top of the basic CycleGAN loss function explained below. The model's function consists of three different elements:

First, Two adversarial losses evaluate the distance between the generated distributions and the original distributions: $Loss_{adv} = \frac{1}{m} \sum_{i=1}^m (1 - D_y(G(x_i)))^2 + \frac{1}{m} \sum_{i=1}^m (1 - D_x(F(y_i)))^2$

The function has two components, one for each GAN; each generator-discriminator pair of the CycleGAN gets evaluated by first mapping an input image from one domain to the other with function

G or F, and then are passed through the discriminator Dy or Dx. The results are evaluated by averaging scores across all mini batch inputs of size m. The loss is then evaluated by the Mean Squared Error function (MSE).

The second part of the loss function is Cycle consistency loss. This loss is described by the following equation: $Loss_{cyc} = \lambda(\frac{1}{m} \sum_{i=1}^m [F(G(xi)) - xi] + [G(F(yi)) - yi])$

Cycle consistency loss enforce mapping constraints $F(G(x)) \approx x$ and $G(F(y)) \approx y$. This is done by mapping a given image x with function G into domain Y. It is then mapped back into domain X by the learned function F, and compared with the original image x again by applying L1 loss. The same is done on the second part of the equation across domain Y. λ controls the magnitude of the regularization effect of this loss.

The last component of the loss function is the identity loss $Loss_{idt} = \lambda_{idt} \frac{1}{m} \sum_{i=1}^m (|G(yi) - yi| + |F(xi) - xi|) * 0.5$.

Identity loss is added as an extra constraint of the two G and F mappings. specifically, the functions learn the identity mapping $G(y) \approx y$ and $F(x) \approx x$. This loss regularizes the generator to be near an identity mapping when real samples of the target domain are provided. This loss is evaluated by L1 loss and is a constraint meant to preserve colour features when mapping from one domain to the other. Finally, the total loss is given by combining all three losses into an objective function.

Wasserstein CycleGAN As a next step, we took the original CycleGAN architecture mentioned above and implemented changes guided by Arjovsky's implementation of the original Wasserstein GAN [1]. Specifically, the discriminator was first turned into a critic by changing the adversarial loss into the following Wasserstein loss function :

$$Loss_{adv} = (\frac{1}{m} \sum_{i=1}^m (Dx(xi) - \frac{1}{m} \sum_{i=1}^m Dx(F(yi))) + (\frac{1}{m} \sum_{i=1}^m (Dy(yi) - \frac{1}{m} \sum_{i=1}^m Dy(G(xi))))).$$

Here, for each generator-network pair in the Wasserstein CycleGAN, we take the average score of the discriminator's Dx between a real image xi and the average score of the critic over the generated images translated by function F into domain X. The distance between these two scores should be maximized. Similarly, for the second model, the average critic Dy scores between a real image y the discriminators Dy score of the generated counterpart by generator G into domain Y is being maximized, as seen in the second part of the equation.

The critic's last layer is also modified to have a linear convolutional output layer. The critic now outputs a score that shows the "realness" or "fakeness" of an image, hence the name critic, and not if the image is fake or not. This score helps for better stabilization of training. Weight clipping was also enforced on the WassersteinCycleGAN in the range [-0.01,0.01] to prevent vanishing/exploding gradient problems. Finally, in contrast to Arjovsky's et al., suggestions [1], we set the optimizer of all generator and discriminator networks to Adam as it provided better results than the RMSprop optimizer.

Improved Wasserstein CycleGAN with Gradient Penalty Taking Gulrajani's et al., advice in [4], the last WCycleGAN-GP model is implemented in exactly the same way as mentioned in the Wasserstein CycleGAN section, with one difference: instead of weight clipping, an extra gradient penalty is added on top of the Wasserstein Adversarial loss. In the loss function, the following gradient penalties are added, one for each discriminator network: $\lambda_1(||\nabla_{\hat{x}} D\hat{x} - 1|| - 1)^2 + \lambda_2(||\nabla_{\hat{y}} D\hat{y} - 1|| - 1)^2$, Where $\hat{x} = \epsilon x + (1 - \epsilon)G(y)$, and $\hat{y} = \epsilon y + (1 - \epsilon)G(x)$.

Here the gradient is taken in accordance to \hat{x} and \hat{y} , which are comprised of interpolation between the real image of the networks input domain and a generated image mapped from the target domain to the original domain. ϵ takes a random value between [0-1] every time. A soft length constraint is then imposed on the gradients of \hat{x} and \hat{y} by taking the norm of the gradient, subtracting 1 from it and squaring the result. This ensures that the gradient is stabilized in a much smoother way than the forceful weight clipping of the WCycleGAN model. Finally, λ_1 and λ_2 are the regularization parameters for each gradient penalty.

Standard Autoencoder with Transfer Loss One common way to train an autoencoder is to use MSE loss. In this paper, we attempt to use a regular autoencoder to learn to change the style of an input image. As mentioned earlier, this requires each of the input images to have a counterpart in the desired style. As this is not usually available this technique had not been done before, however, after running the CycleGANs and producing style-transferred versions of each of the input images, this method became

possible. The new loss function is thus $L = \frac{1}{n} \sum_{i=0}^n (\hat{x}_i - y_i)^2$ where \hat{x}_i remains a reconstructed input image and y_i represents a stylised version of the input image x_i .

Fast Style Transfer Autoencoder When no input-output pairs of images and their corresponding style transfer are available, there is another way to do style transfer using autoencoders. This works by having two different losses a content-loss and a style-loss, first introduced in [2]. The content-loss makes sure that the structure of the input photo is kept in the output image, while the style-loss rewards the network for having a similar texture (style) compared to a reference style image. To compute these losses the input and the corresponding output of the network (or the style template) are feed into a pre-trained VGG network [6] which then extracts features from the images.

For the content loss, the feature maps after the ninth layer of both input and output image are used, the loss is then computed using the *MSE* between these representations. The intuition behind this is that after nine layers there is enough information about the structure of the images and therefore this makes sure that the content of the input image is kept in the output. Simply comparing both images using the *MSE* loss would not work, since then not only the structure but also the style (pixel colours) would influence the loss.

For the style loss, the generated image and the style template are fed into VGG, for training the models the image in Figure 2 was used as a style template. The style loss is then given by the correlations between different channels at different levels in the VGG. Here the basic intuition is that features that are active together in the style template give information about the texture and stylistic features but not about the content of the image. For instance, if vertical edges tend to activate with curly edges this would be specific to Van Gogh style. To calculate these correlations Gram matrices were used. Please see [5] for an explanation. In our implementation the Gram matrices after the 4_{th}, 9_{th}, 16_{th} and 23_{rd} are compared and the loss is then averaged.

Both the style loss and the content loss are then combined to give the total loss for one forward pass thru the network. This is done by multiplying both the style and the content loss by some constant and then adding them together [5]. The advantage of using this technique is that it is a lot faster than GANs since only one image is used as a style template.

Network	Conv Layers	Type of padding	Max image compression
CycleGAN generator	12	Reflection	62x62
CycleGAN discriminator	5	Reflection	-
Our Autoencoder	25	Zero	26x26
TransformerNet	16	Reflection	64x64




Figure 2: Left: Description of the model architectures Right: Image used as style template

Model architectures A brief description of all model architectures can be found in table 2. All CycleGAN models used the same generator and discriminator architecture, with the only difference being that we removed the tanh output activation from the Wasserstein GANs. The CycleGAN generator and the TransformerNet both use residual blocks in the middle of the model. The important distinction between the models is their depth and the maximal compression of the image dimensions. Our Autoencoder compresses the images a lot more than the other models, however, it is deeper than the other models. All models were optimized using the Adam optimizer, furthermore, the inner activation function is always ReLU while the output activation function is always Tanh.

Experiment Design To test the performance of each of the different models, they were each trained for 30 epochs on their corresponding data sets. After training each of the models was given 4 more photos that were not contained in the training dataset. The performance of each of the models was then evaluated based on how well the style of the Monet painting was transferred onto the photos.

3 Results

The transformed images can be seen in figure 3.

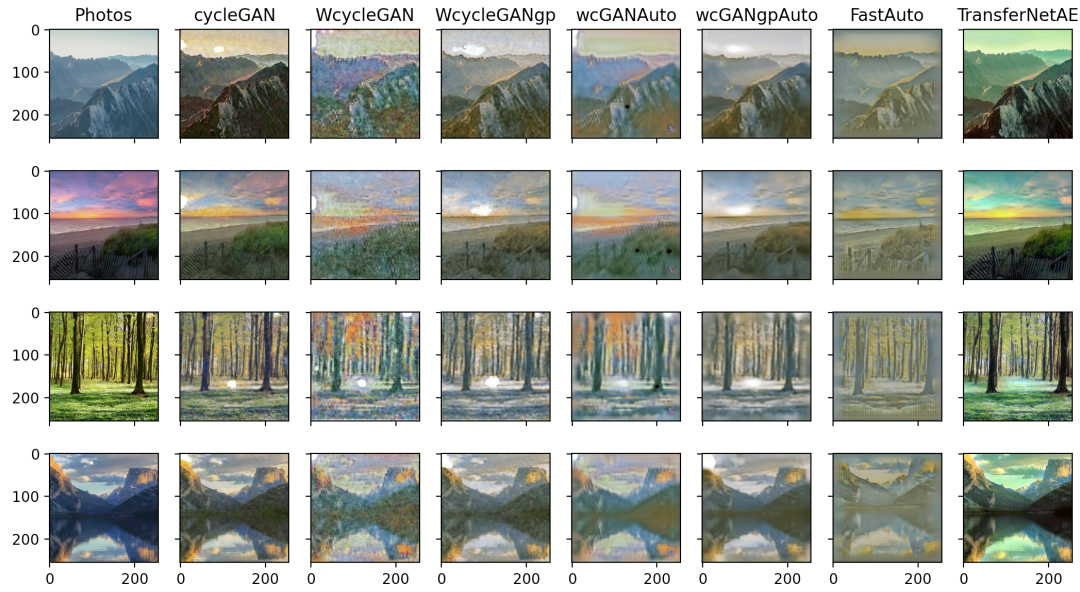


Figure 3: Resulting style transformations by the different models

CycleGANs Looking at CycleGAN model output first in figure 3, it seems that the colour of the photos to the Monet domain has transferred over nicely. On the other hand, WcycleGAN also seems to transfer the Monet feel over to the test images, and it also "intensifies" photo images, giving an even more "painting" feel. Lastly, the WCycleGAN with gradient penalty seems to have washed out the colours slightly, but the Monet-like texture is kept.

Standard Autoencoders with Transfer Loss Using the standard autoencoder with transfer loss we can see that it was indeed possible to train the model to learn the Monet style. We can see that the resulting images follow a similar style to those obtained from the model corresponding to their training data. The quality of their images, compared to their GAN counterparts is less defined, especially texture-wise, using a better performing GAN though would, of course, results in a better performing autoencoder.

Transfer Autoencoders We observe that both of the autoencoder models are heavily dependant on the given reference image. We observe that the texture of the image was not transferred well to the test photos, that being said, the improved version of this model was able to recreate the input image with very high detail.

4 Discussion

Given our results from Table 3, it seems that CycleGANs can produce promising results. In specific, CycleGANs are exceptionally good at picking features from the training Monet images. Wasserstein-CycleGAN seems more prone to instability problems between the generators and discriminators, as hyperparameter optimization proved to be tricky. In comparison, WassersteinCycleGAN with gradient penalty managed to produce more "stable" results which are reflected in the output colours of the test images. Hyperparameter optimization was indeed smoother in the optimization of this model, as the loss did not tend to get stuck in local minima.

Autoencoders using input-output pairs can work very well for style transfer, however, they are entirely dependent on the quality of these pairs. Fast style transfer autoencoder also works quite well for this task, however, the style does not come thru that much and is mostly reflected in the colours. This is probably due to only one template image being used for the style, which can be hard to generalize. In our case the style template is a sunset above the ocean, from this it is hard to infer in which style Monet would paint a forest. The fact that the TransformerNet works better than our Autoencoder is probably

due to a combination of the following factors: too much image compression, no residual blocks and no symmetry between encoder and decoder.

References

- [1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.
- [2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.
- [3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.
- [5] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.
- [6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [7] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*, 2018.
- [8] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.