CartoonGAN: Style Transferring Real-world Photos to Anime-style Images

Benny Cai *
Department of ECE
A16142137
j1cai@ucsd.edu

Jacky Wang *
Department of Mathematics
A16164359
jiw010@ucsd.edu

Abstract

We address the task of style transferring, which aims to render the semantic contents of the target images with a different style, given the contents of different semantic information. More specifically, our goal is to transfer real-world photos to anime style by re-implementing the "CartoonGAN" —– a Generative Adversarial Network for Photo Cartoonization. This report will include the cartoonization problem we want to solve, the method we would use to solve the problem, including image preprocessing, CartoonGAN architecture, loss function, and etc, the dataset, and the experiments we would perform.

1 Introduction

1.1 Problem

In general, we want to train a Generative Adversarial Network (GAN) that is able to perform anime-style transfer for real-world photos. Specifically for this project, we want our GAN to be able to transfer the style from an anime series called "Violet Evergarden" to real-world photos.

1.2 Motivation

Currently, a lot of anime series or movies would use real-world scenes as references to produce anime scenes. For example, the director of the famous anime movie "Your name", Makoto Shinkai, includes many realistic places in Tokyo into the movie. Therefore, as anime fans, we think being able to directly convert real-world photos into anime/cartoon style images can be very helpful for improving the efficiency of producing anime scenes. More significantly, training a robust GAN also facilitates other computer vision work in anime-style art.

1.3 Some key parts of the problem:

- In general we'll have two types of dataset: one for the real-world photos, the other for the anime images/screenshots.
- Our GAN should be able to reconstruct the input photo into a anime/cartoon style while keeping its semantic content as much as possible [1].
- Anime scenes tend to have very clear edges, so we need an edge-promoting adversarial loss to preserve the clear edges in our anime images dataset [1].
- We want our network to be able to accurately locate, recognize, and produce these clear edges. Therefore, we would manually smooth the edges of some anime images and train the GAN with these edge-smoothing images to improve its ability to recognize the feature of clear edges. This will be explained again later as well, since it's quite important.

*These two authors contributed equally.

2 Related Work

2.1 Style Transferring with GAN

The very early work applied CNN reveals the CNN's potential to generate high quality images [2]. Since GAN was first proposed in 2014, most style transferring networks depend on training both the discriminator network and the generator network [3]. One of the most representative one is CycleGAN, which introduces the cycle consistency loss to help training the inverse mapping for the unpaired image-to-image translation [4]. The network we are implementing is mostly based on the CycleGAN [1]. Most recently, the proposed GAN-based network architectures such as [5] and [6] are more light-weight or requiring fewer training samples.

2.2 Cycle Consistency

Cycle consistency aims to apply the transitivity of data as to supervise CNN training. CycleGAN has appeared in many tasks recently. The higher order cycle-consistency is shown the ability to apply for structure from motion [7] and 3D shaping [8]. Zhu et al. first takes this idea into GAN, known as CycleGAN [4]. In video action recognition, a self-supervised method using cycle-consistency over time is applied for learning visual correspondence from unlabeled video [9].

2.3 Neural Style Transfer

Other than CNN or GAN-based methods, another approach is neural style transfer [10, 2]. The significant difference is that these methods often learn the mapping between two image collections, rather than between two specific images. Most of these approaches are based on matching the Gram matrix statistics of pre-trained deep features.

3 Method

3.1 Image-preprocessing

As mentioned earlier, clear edges are an important feature of anime images, yet it only accounts for a small portion of the entire image. Therefore, we want to train the ability of the discriminator to correctly distinguish between anime images with and without clear edges —– essentially identify images with clear edges as real and images without clear edges as fake. This will push the generator to produce images with clearer edges, and hence obtain anime images with higher quality.

Therefore, we would like to manually create another anime-style dataset without clear edges based on the original anime-style dataset we have using image-processing techniques. Essentially, our goal is to smooth, or blur, the clear edges of the anime images. We basically follow the steps proposed by the original paper [1]. We use the Canny edge detector to detect the edges and dilate these edges [11]; then, we apply a Guassian smoothing in these edge regions. Two sample images after the pre-processing are shown below:

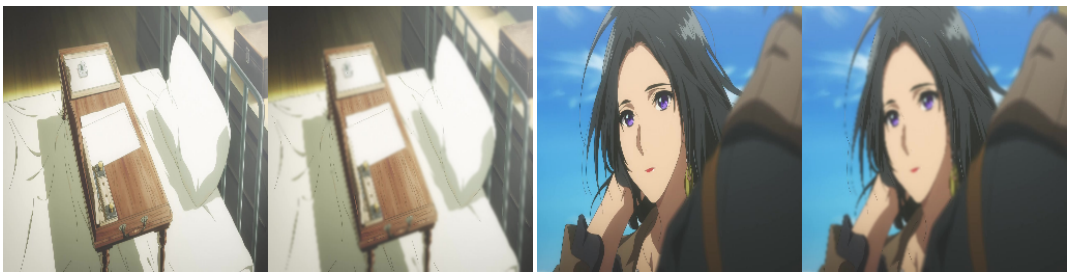


Figure 1: Comparison of original anime image (left) and edge-smoothed version (right)

3.2 CartoonGAN architecture

Discriminator Network

We want to train the ability of the discriminator to correctly classify the input images as real or fake images. In our case, the discriminator takes three types of anime images as input:

1. original anime images/screenshots, classified as real images.
2. edge-smoothed version of anime images, classified as fake images.
3. anime images generated by the generator network, classified as fake images.

As mentioned earlier, training discriminator's ability would push the generator to keep improving its performance of generating high-quality anime images with clear edges.

The discriminator network is rather simple and shallow compared to the generator network, considering the fact that it has a much easier task than the generator and hence is much easier to train. The network starts with a flat-convolutional layer, followed by two down-convolutional blocks to reduce the resolution and encode features; then, a flat-convolutional block and a convolutional layer are applied to construct the features and accordingly obtain the classification result.

Generator Network

We want to train the generator to produce anime images by applying the anime style to real-world photos such that the generated images can fool the discriminator. The generator takes real-world photos as input, and outputs the anime-style version of the photos. Its ability to transfer the style is improved by the push from the discriminator.

The generator network is much more complex. It starts with a flat-convolutional layer, followed by two down-convolutional blocks to compress the images and encode features. Then, eight residual blocks are utilized to re-construct the semantic content and apply the anime style. Lastly, two up-convolutional blocks, followed by a final convolutional layer, are used to decode the features and construct the final result.

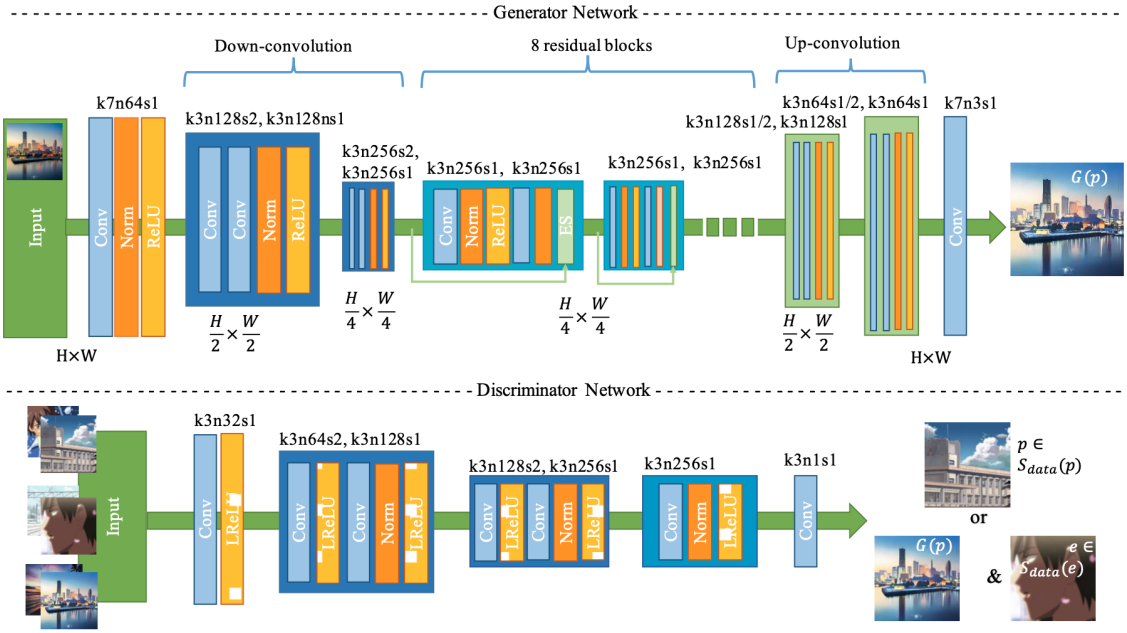


Figure 2: CartoonGAN architecture proposed in the original CartoonGAN paper [1]

3.3 Loss

Discriminator loss

The discriminator loss is consist of three parts, where each part corresponds to a type of input that it takes in. Denote $S_{data}(o) \subset O$ as the original anime images, $S_{data}(e) \subset E$ as the edge-smoothed anime images, $S_{data}(p) \subset P$ as the real-world photos, and $G(p_i)$ as the generated anime image. Hence, the discriminator loss can be formulated as follows:

$$\begin{aligned}\mathcal{L}_D = -(&\mathbb{E}_{o_i \sim S_{data}(o)} [\log D(o_i)] \\ &+ \mathbb{E}_{e_j \sim S_{data}(e)} [\log (1 - D(e_j))] \\ &+ \mathbb{E}_{p_k \sim S_{data}(p)} [\log (1 - D(G(p_k)))])\end{aligned}$$

Essentially, we want to maximize the probability of correctly classifying the aforementioned three types of input. In terms of the formula, we want $D(o_i) = 1$, $D(e_j) = 0$, and $D(G(p_k)) = 0$ to achieve the optimization. We compute the discriminator loss by maximizing these probabilities and negating their sum, and we let the back-propagation take care of minimizing the loss.

Content loss

The network's ability to retain the semantic content of the original images is rather crucial when performing style transfer. Therefore, as proposed in the original paper, it's necessary to include a content loss as part of the training. CartoonGAN adopts the high-level feature maps in the pretrained VGG network, which is believed to have good object preservation ability [1]. Using the same set of notations, the content loss can be formulated as follows:

$$\mathcal{L}_{\text{con}} = \mathbb{E}_{p_i \sim S_{data}(p)} [\|\text{VGG}_l (G(p_i)) - \text{VGG}\, l (p_i)\|_1]$$

We compute the L1 distance between the semantic contents of the original real-world photo and the same photo after applying the anime style, where the semantic contents are extracted by the pretrained VGG network [12]. We minimize the content loss by minimizing the computed L1 distance, which could push the generator to produce images that retain as much semantic content as possible from the original photos.

Generator loss

For the generator, we want to maximize the probability of generating an anime-style image that fools the discriminator. In terms of the formula, we want $D(G(p_i)) = 1$, indicating that the discriminator would classify the generated image as a real image. Moreover, the aforementioned content loss is also included as part of the generator loss, because of the same reason for retaining semantic content in the generated images. Hence, the content loss is formulated as follows:

$$\mathcal{L}_G = -\left(\mathbb{E}_{p_i \sim S_{data}(p)} [\log (D(G(p_i)))]\right) + \mathcal{L}_{\text{con}}$$

Similar to the discriminator loss formula, the first part of the generator loss is computed by maximizing the probability of fooling the discriminator and negating the result. After adding the content loss, we can let the back-propagation take care of minimizing the loss.

3.4 Generator network pretrain / warm-up phase

As the original paper mentioned, GAN is highly non-linear. Hence, with random initialization, the optimization might be easily trapped at sub-optimal local minimum [1]. To improve the convergence, we put a warm-up phase before the official start of the GAN training process, aiming to prepare the generator network with good parameter initialization.

The warm-up phase can be considered as a pretrain process for the generator, in which the goal is to train the generator to retain as much semantic content as possible to help with convergence during later training. Therefore, the only loss involved here is the content loss, which has been defined in the previous section. Two sample images produced by the generator after pretraining for 10 epochs are shown below, which turns out to have pretty nice initialization results:

Figure 3: Comparison of original photos (left) and generated images after 10 epochs' pretrain (right)

4 Experiments

4.1 Dataset

Real-world photos - ADE20K dataset

There isn't any strict requirement for real-world photos dataset for CartoonGAN as long as it's a clearly taken photo, because one should expect any kind of scene to appear in an anime. We decided to choose ADE20K dataset, because it is believed to contain a variety of real-world scenes, including natural landscapes, constructions and buildings, indoor scenes, and etc. This dataset contains roughly 20,000 photos of different sizes, which are later all resized to 256 x 256 for training purpose.

Anime - "Violet Evergarden"

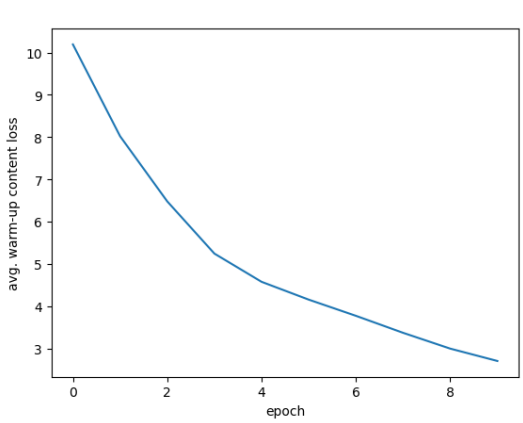
We choose the anime series "Violet Evergarden" as the anime style to transfer because its painting style is known to be very beautiful. Besides, it also contains a variety of scenes similar to the ADE20K dataset. Admittedly, one limitation of this anime is that it contains many human faces, which might hinder the GAN's training process a little bit. We downloaded the original anime screenshots from the Internet as our dataset, and collected roughly 9,000 screenshots.

4.2 Results

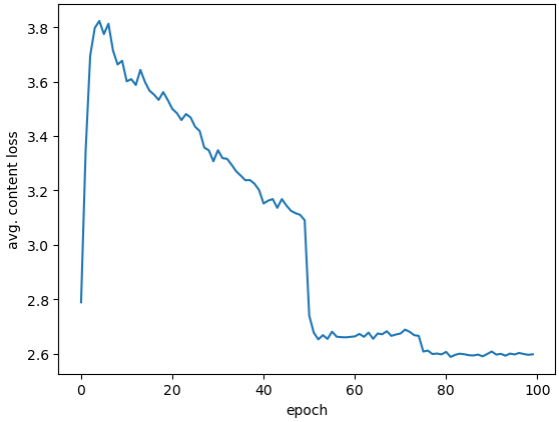
Analysis of losses (Figure 4 and 5)

Essentially, we pretrain the generator for 10 epochs for the warm-up phase, and then train the CartoonGAN for 100 epochs. From graph(a) in Figure 4, we can see the warm-up phase is very successful, where the average content loss has been steadily decreasing. Two sample images in Figure 3 in section 2.4 also demonstrates the good result of the warm-up phase, where the semantic content of the input image is effectively retained.

During training, the discriminator loss has been generally decreasing, and its loss curve is pretty smooth and flat. On the other hand, although the content loss and the generator loss are also decreasing in general, the losses fluctuate a lot, especially the generator loss. This is expected because the discriminator is much easier to train than the generator as previously mentioned. Also, we apply a learning-rate decay at epoch 50 and 75 for both discriminator and generator, which is shown to be effective as we can observe that for all three losses during training, there is an obvious drop of loss at epoch 50 and 75. Nonetheless, the fluctuation in the generator loss indicates that our hyper-parameter tuning might not be optimal, and there is definitely still room for improvement.

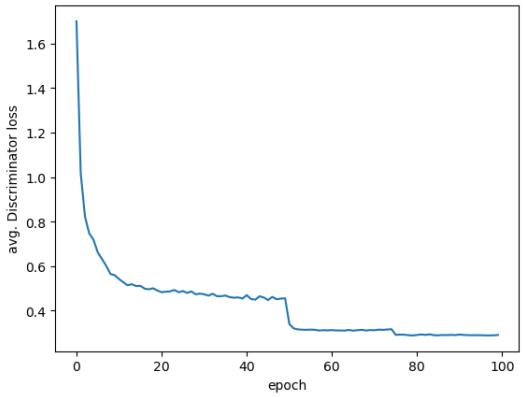


(a) Average content loss during warm-up

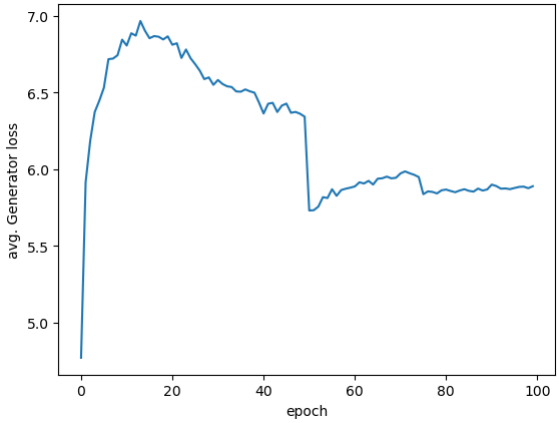


(b) Average loss during training

Figure 4: Content loss



(a) Average discriminator loss



(b) Average generator loss

Figure 5: Discriminator and generator loss

Analysis of generated results (Figure 6 and 7)

We've chosen a few sample generated results displayed down below. In general, we can see changes in colors, lightness, edge smoothness, and etc, indicating the effectiveness of style transfer. More specifically, for some certain examples, we can observe that the grass or tree in the real-world photo is applied a brighter green color in the generated counterpart; the sky is applied a brighter blue color, and the cloud in the sky becomes whiter and more distinct. Very interestingly, for some real-world photos that contain a relatively empty sky, the generated images might add a patch that look like sun to the sky. We think this may be due to the fact that most of our anime screenshots that contain blue sky also contain a sun, and hence this feature is also applied to the real-world photos after learning. Moreover, if we look more closely, we can observe that some of our generated images have clearer edges than the original photos do, although it might not be very obvious. We believe that if we tune the hyper-parameter better, this feature would become more obvious, because it's an essential part of this project.

Admittedly, one flaw in the generated results we can observe is that some objects and buildings are somehow also applied a little bit of green and blue colors. We think the reason is probably also about our anime screenshots dataset, in which the dataset might have too many images that contain green plants and blue sky. As a result, the network might tend to apply the green and blue colors to many objects no matter what the objects are.

Figure 6: Comparison of original photos (left) and generated images (right)

Figure 7: Comparison of original photos (left) and generated images (right)

5 Distribution of work

- Benny: model/network
- Jacky: dataset
- The experiment/training/validating section is written together by us.

References

[1] Yang Chen, Yu-Kun Lai, and Yong-Jin Liu. "CartoonGAN: Generative Adversarial Networks for Photo Cartoonization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "Image Style Transfer Using Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`.

[4] Jun-Yan Zhu et al. “Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[5] Jie Chen, Gang Liu, and Xin Chen. “AnimeGAN: A Novel Lightweight GAN for Photo Animation”. In: *Artificial Intelligence Algorithms and Applications*. Ed. by Kangshun Li et al. Singapore: Springer Singapore, 2020, pp. 242–256. ISBN: 978-981-15-5577-0.

[6] Guoxian Song et al. “AgileGAN: Stylizing Portraits by Inversion-Consistent Transfer Learning.” In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* (July 2021).

[7] Christopher Zach, Manfred Klopschitz, and Marc Pollefeys. “Disambiguating visual relations using loop constraints”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 1426–1433. DOI: 10.1109/CVPR.2010.5539801.

[8] Qi-Xing Huang and Leonidas Guibas. “Consistent Shape Maps via Semidefinite Programming”. In: *Computer Graphics Forum* 32.5 (2013), pp. 177–186. DOI: https://doi.org/10.1111/cgf.12184. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12184. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12184.

[9] Xiaolong Wang, Allan Jabri, and Alexei A. Efros. “Learning Correspondence From the Cycle-Consistency of Time”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[10] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 405–421.

[11] John Canny. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[12] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].