# Manga-to-Anime Translation Using Cycle-Consistent Generative Adversarial Networks

**Jonathan Griffin Department of Computer Science**
Stanford University
jgriffi2@stanford.edu

## Abstract

This paper presents GANime: a generative adversarial network that uses cycle-consistency to perform on the manga-to-anime task. The architecture used is that of CycleGAN. The main contribution of this paper is the effort to translate one page of manga that contains multiple panels into an equivalent animated version. This task proves difficult due to the lack of a manga dataset that contains single images as opposed to multiple panels.

## 1 Introduction

Over the past few years, manga and anime have became widely popular. Therefore, there has been an increase in production of both manga and anime. One major part of the industry is adapting widely popular manga series into anime. Unfortunately, it is often the case that these animators are overworked and underpaid.

For this reason, I introduce GANime: a manga-to-anime translator that uses cycle-consistent generative adversarial networks (GANs). The input to the network are images from the manga and anime domain, and the output are translated images into the anime and manga domain.

## 2 Related Work

CycleGAN is one of the main pieces of work related to GANime [14]. It introduces the idea of translating images from domain to domain while maintaining consistency. In addition to CycleGAN, [6] and [12] discuss interesting concepts pertaining to consistency within GAN networks.

There are also other works that focus on manga or anime-style GAN networks [2], [7], [4], [13], [11], [10], [3]. These works tackle areas such as Style Transfer, Colorization, Character Generation, and Sketching. However, none of them try to use cycle-consistency within their network architecture for their specific task and only one covers domain-to-domain translation in the manner I wish to address [10].

## 3 Data

The dataset used for GANime was divided into two domains: manga and anime. The detail about each of the datasets are given in Sections 3.1 and 3.2. Each image is of shape $256$x$256$ and each domain uses a training-validation-testing split of $80\%$-$10\%$-$10\%$. Each domain has $2000$ images, so each training set has $1600$ images, each validation set has $200$ images, and each testing set has $200$ images.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Figure 1: Example images from **Manga109**.


Figure 2: Example images from **Nico-Illust** and **Danbooru2017**.

### 3.1 Manga

The manga dataset is a subset of **Manga109** [9], [8]. **Manga109** consists of manga pages from 109 manga volumes drawn by professional manga artists. Domain $X$ consists of random images from **Manga109**. Examples of images from the manga dataset are shown in Figure 1.

### 3.2 Anime

The anime dataset is a subset of **Nico-Illust** and **Danbooru2017** [1], [5]. Together, **Nico-Illust** and **Danbooru2017** have over 3 million hand-drawn anime-style images taken from a public website. Domain $Y$ consists of random images from **Nico-Illust** and **Danbooru2017**. Examples of images from the anime dataset are shown in Figure 2.

## 4 Method

This section describes the architecture (Section 4.1), the loss functions (Section 4.2) and the implementation (Section 4.3). Each of them are discussed in more detail in their corresponding section below.

### 4.1 Architecture

The architecture used for the manga-to-anime task was the same architecture that was used in CycleGAN. There are two domains, $X$ and $Y$, two generators, $G$ and $F$, and two discriminators $D_X$ and $D_Y$. Generator $G$ translates an image from $X$ to $Y$ and Generator $F$ translates an image from $Y$ to $X$. Discriminator $D_X$ tries to determine if an image is an image generated from $F$ or is an image originally from $X$. Discriminator $D_Y$ tries to determine if an image is an image generated from $G$ or is an image originally from $Y$.

| Layer | Activation Size |
|---|---|
| Input | 256x256x3 |
| 4x4x64 convolution, stride 2 | 128x128x64 |
| 4x4x128 convolution, stride 2 | 64x64x128 |
| 4x4x256 convolution, stride 2 | 32x32x256 |
| 5 4x4x512 convolution, stride 2 | 1x1x512 |
| 4 4x4x512 deconvolution, stride 2 | 16x16x1024 |
| 4x4x256 deconvolution, stride 2 | 32x32x512 |
| 4x4x128 deconvolution, stride 2 | 64x64x256 |
| 4x4x64 deconvolution, stride 2 | 128x128x128 |
| 4x4x3 deconvolution, stride 2 | 256x256x3 |

| Layer | Activation Size |
|---|---|
| Input | 256x256x3 |
| 7x7x64 convolution, stride 1 | 128x128x64 |
| 3x3x128 convolution, stride 2 | 64x64x128 |
| 3x3x256 convolution, stride 2 | 32x32x256 |
| 9 Residual block, 256 filters | 32x32x256 |
| 3x3x128 deconvolution, stride 2 | 64x64x128 |
| 3x3x64 deconvolution, stride 2 | 128x128x64 |
| 7x7x3 convolution, stride 1 | 256x256x3 |

Table 1: Generators, (left) U-Net, (right) Res-Net.

| Layer | Activation Size |
|---|---|
| Input | 256x256x3 |
| 4x4x64 convolution, stride 2 | 128x128x64 |
| 4x4x128 convolution, stride 2 | 64x64x128 |
| 4x4x256 convolution, stride 2 | 32x32x256 |
| 4x4x512 convolution, stride 1 | 32x32x512 |
| 4x4x1 convolution, stride 1 | 32x32x1 |

Table 2: Discriminator Architecture.

#### 4.1.1 Generators

There were two types of generator architectures that this paper tested: U-Net and Res-Net. Their architectures are shown in Table 1.

Instance normalization is performed for all layers of the U-Net architecture aside from the last layer. The convolution layers use a leakyReLU activation function characterized by $\alpha = 0.2$. The deconvolution layers use a ReLU activation function. The first three deconvolution layers use dropout with a probability of $0.5$. Aside from the last deconvolution layer, concatenation is performed with the opposite convolution layer (deconvolution layer $1$ is concatenated with convolution layer $7$, $2$ with $6$, and so on). The last step for this architecture is to use a tanh activation function.

Instance normalization is performed for all layers of the Res-Net architecture and the ReLU activation function is used for all layers. The residual block consists of two 3x3 convolutions with a stride of $1$ with instance normalization and ReLU activation in between. The last step for this architecture is to use a tanh activation function.

#### 4.1.2 Discriminators

Discriminators $D_X$ and $D_Y$ are the same for the purposes of this task. The architecture for the discriminators is shown in Table 2.

LeakyReLU is used for each layer and is characterized by $\alpha = 0.2$. Instance normalization is performed for all layers except the first and last layer.

### 4.2 Loss Functions

GANime is trained on a few different loss functions, which are described below.

#### 4.2.1 GAN Loss

The standard GAN loss that was used in CycleGAN is the saturating loss, which is why GANime does not use it. Instead, I test using the loss functions in Equations 1 and 2. Equation 1 is the non-saturating loss function for a generator, so training occurs much more quickly at the beginning since the generator initially performs poorly at generating an image in the target domain and the

| | LSGAN, Res-Net | LSGAN, U-Net | GAN, Res-Net |
|---|---|---|---|
| **FID Score** | 298.98 | 252.80 | 287.24 |

Table 3: FID scores of models.

gradient of 1 is largest at 0. Equation 2 is tested because [14] states that the LSGAN loss function produces the smallest loss given enough time to train.

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[log(D_Y(y))] \\ + \mathbb{E}_{x \sim p_{data}(x)}[-log(D_Y(G(x)))]\end{aligned} \tag{1}$$

$$\begin{aligned}\mathcal{L}_{LSGAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[(D_Y(y) - 1)^2] \\ + \mathbb{E}_{x \sim p_{data}(x)}[D_Y(G(x))^2]\end{aligned} \tag{2}$$

#### 4.2.2 Cycle Loss

The cycle loss, as described in [14], is shown in Equation 3. The intuition behind this loss function is that if an image from one domain is translated into another domain and then back into the original domain, the translated image should be the same as the original image.

$$\begin{aligned}\mathcal{L}_{CYC}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] \\ + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(x)) - y\|_1]\end{aligned} \tag{3}$$

#### 4.2.3 GANime Loss

The loss function of GANime is the combination of the GAN loss and the Cycle loss. The two loss functions optimized on are shown in Equations 4 and 5.

$$\begin{aligned}\mathcal{L}_1 = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \\ \mathcal{L}_{GAN}(F, D_X, X, Y) + \\ \lambda \mathcal{L}_{CYC}(G, F)\end{aligned} \tag{4}$$

$$\begin{aligned}\mathcal{L}_2 = \mathcal{L}_{LSGAN}(G, D_Y, X, Y) + \\ \mathcal{L}_{LSGAN}(F, D_X, X, Y) + \\ \lambda \mathcal{L}_{CYC}(G, F)\end{aligned} \tag{5}$$

### 4.3 Implementation

Each of the models were trained through Amazon AWS with $4$ GPUs for $200$ epochs, taking $4$ days to train per model. Using random learning rate choices from $10^{-5}$ to $10^{-1}$, I found that a learning rate of $0.0002$ was optimal. Using a similar style of choosing $\lambda$, I found that $\lambda = 10$ was an optimal choice.

The network was made using tensorflow. The github repository can be found at `https://github.com/jgriffi2/CS230-Project`.

## 5 Results

The results for the various models are shown in Figure 3. As can be seen, generator $G$ wants to create a single image as opposed to panels of images, which causes an undesired output.

### 5.1 Evaluation

The FID scores of each of the models is shown in Table 3. The LSGAN+U-Net model performed the best. However, none of the models produced an output that was desired and each of the FID scores were very large, meaning the generators did not perform well.

### 5.2 Limitations

The main obstacle of the models of GANime are that the images in domain $X$ contain multiple panels of images and the images in domain $Y$ contain, usually, a single image. Therefore, the discriminators

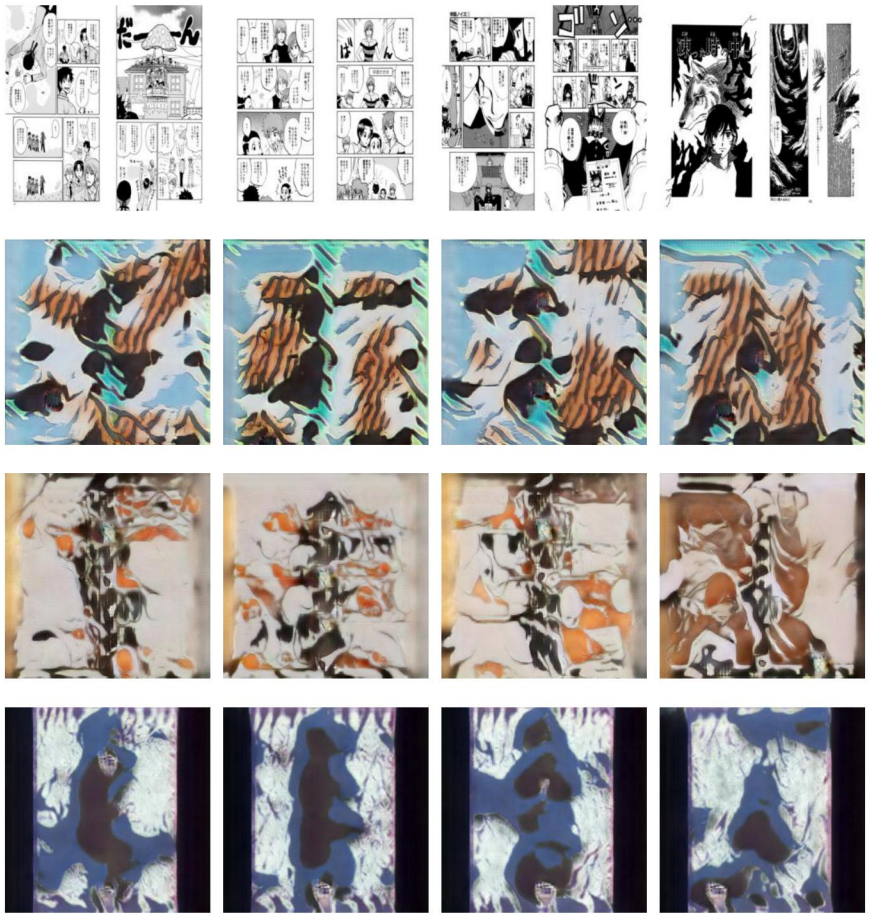

Figure 3: Results from experiments; (top) input, (2nd row) LSGAN + Res-Net, (3rd row) LSGAN + U-Net, (4th row) GAN + Res-Net.

can tell very easily the generated image apart from the target domain if it maintains the qualities of the original image. This causes the generator $G$ to generate images that look like a single conglomerated image and generator $F$ to generate images that look like an image that contains multiple panels. This is, unfortunately, not the desired output of the model.

### 5.3 Future Work

The best course of action for future work is to use individual manga panels as opposed to whole manga pages as the domain of $X$. As discussed in 5.2, the models wanted to produce a single images instead of multiple self-contained images like manga pages have. Doing this would lead to generated images that favored the likeness of the original input image more than the current model does.

## 6 Conclusion

This paper presents GANime, a GAN that uses cycle-consistency to perform on the manga-to-anime task. Unfortunately, due to available datasets, the manga and anime domains were not similar enough to one another which caused issues with generating images. However, this work can be expanded upon in the future to generate better results given a better manga dataset.

## 7 Contributions

All work was done by Jonathan Griffin.

## References


[1] Anonymous, the Danbooru community, Gwern Branwen, and Aaron Gokaslan. Danbooru2018: A large-scale crowdsourced and tagged anime illustration dataset. `https://www.gwern.net/Danbooru2018`, January 2019. Accessed: 22 February.

[2] Y. Ci, X. Ma, Z. Wang, H. Li, and Z. Luo. User-guided deep anime line art colorization with conditional adversarial networks. 2018.

[3] M. Eitz, J. Hays, and M. Alexa. How do humans sketch objects? 2012.

[4] K. Hamada, K. Tachibana, T. Li, H. Honda, and Y. Uchida. Full-body high-resolution anime generation with progressive structure-conditional generative adversarial networks. 2018.

[5] Hikaru Ikuta, Keisuke Ogaki, and Yuri Odagiri. Blending texture features from multiple reference images for style transfer. In *SIGGRAPH Asia Technical Briefs*, 2016.

[6] P. Isola, J. Zhu, T. Zhou, and A. Efros. Image-to-image translation with conditional adversarial networks. 2018.

[7] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Zhihao Fang. Towards the automatic anime characters creation with generative adversarial networks. 2017.

[8] Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki, and K. Aizawa. Sketch-based manga retrieval using manga109 dataset, multimedia tools and applications. 2017.

[9] T. Ogawa, A. Otsubo, R. Narita, Y. Matsui, T. Yamasaki, and K. Aizawa. Object detection for comics using manga109 annotations.

[10] E. S. Serra, S. Iizuka, and H. Ishikawa. Mastering sketching: Adversarial augmentation for structured prediction. 2018.

[11] S. Xiang and H. Li. Anime style space exploration using metric learning and generative adversarial networks. 2018.

[12] J. Yi, H. Zhang, P. Tan, and M. Gong. Dualgan: Unsupervised dual learning for image-to-image translation. 2017.

[13] L. Zhang, Y. Ji, and X. Lin. Style transfer for anime sketches with enhanced residual u-net and auxiliary classifier gan. 2017.

[14] J. Zhu, T. Park, P. Isola, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. 2017.