Implementation of Cycle GAN

Yilin Zhu
yilinz10@illinois.edu

Yifan Shi
yifans16@illinois.edu

April 2020

1 Brief Introduction of GAN

GAN is a form of generative model composed of two competing parts , a generative network and a discriminative network. Generative network produces fake samples from some random noise. Discriminative network, performing as a binary classifier, is responsible for distinguishing candidates synthesized by the generative network from the real data distribution.

To be specific, suppose we have some real data $\mathbf{x} \in \mathcal{X}$, of which the probability distribution function is $Pr(\mathbf{x})$. For generative network, we are going to construct a function $\mathcal{G}$ with the parameter set θ, mapping from a predefined space $\mathcal{Z}$ to the sample space $\mathcal{X}$, and we want the result $\mathcal{G}(\mathcal{Z};\theta)$ follows the real sample distribution $Pr(\mathbf{x})$. A very common choice for $\mathcal{Z}$ is multivariate normal distribution $\mathcal{N}(0, I)$. For discriminative network, the constructed function $\mathcal{D}$ with the learnt parameter set ϕ would suggest the probability that the input comes from the real sample. In other words, if inputs from the real sample are labeled 1, we should have

$$Pr(y=1|x) = \mathcal{D}(x;\phi), \text{ and } Pr(y=0|x) = 1 - \mathcal{D}(x;\phi)$$

According to different functions of the two differing networks, the optimization objective functions could be built respectively as follows,
for discriminative network,

$$\arg\max_{\phi} \left\{ \mathbb{E}_{x\sim Pr(x)} \left[\log \mathcal{D}(x;\phi)\right] + \mathbb{E}_{z\sim Pr(z)} \left[\log(1-\mathcal{D}(x;\phi)\right] \right\}, \tag{1}$$

and for generative network,

$$\arg\max_{\theta} \left\{ \mathbb{E}_{z\sim Pr(z)} \left[\log \mathcal{D}(\mathcal{G}(z;\theta);\phi)\right] \right\} \tag{2}$$

As we can see from objective functions above, the two networks compete with each other. Discriminative network spares no effort to identify the source of samples. While the generative network aims to synthesize samples which could be identified as real data by discriminative network. The overall objective function for GAN could be written as

$$\arg\min_{\theta}\max_{\phi} \left\{ \mathbb{E}_{x\sim Pr(x)} \left[\log \mathcal{D}(x;\phi)\right] + \mathbb{E}_{z\sim Pr(z)} \left[\log(1-\mathcal{D}(\mathcal{G}(z;\theta);\phi))\right] \right\} \tag{3}$$

where the equation inside the curly brackets could be denoted as $\mathcal{L}_{GAN}(\mathcal{G}, \mathcal{D}, \mathcal{Z}, \mathcal{X})$, the adversarial loss.

2 Discussion of the Paper

2.1 Research purpose

One prevalent application of GAN would be synthesizing output identical to the target picture, which generally requires a set of paired training samples, $i.e.\{x_i, y_i\}_{i=1}^N$, where $x_i \in X$ and $y_i \in Y$ for each $i \in \{1, 2, \cdots, N\}$. X and Y are called source domain and target domain respectively. This process could be regarded as a supervised learning task. However, sometimes the paired training data could be unavailable, such as the task of translating Monet's paintings to a a photo of the scene he painted, under which the target photo is nearly infeasible to obtain. To generate the expected output from the source data without the existence of the corresponding target, this paper proposes Cycle-Consistent Adversarial Networks (Cycle GAN). [1]

2.2 Illustration on the theory of Cycle GAN

In the situation mentioned above, although we lack supervision in terms of paired examples, we could explore the supervision at the level of overall training sets. We could instead consider a source set $\{x_i\}_{i=1}^N, x_i \in X$ and a target set $\{y_j\}_{j=1}^N, y_j \in Y$, between which there is no correspondent relationship. In other words, we don't have the information on which x_i matches which y_j.

We consider to train the two sets cyclically. To be specific, we aim to construct a translator from X to Y and a translator conversely from Y to X, and then train them simultaneously. The two translators should be inverses of each other. Mathematically, our goal is to construct the translator $\mathcal{G} : X \to Y$ as well as another translator $\mathcal{F} : Y \to X$, and force them to satisfy $\mathcal{F}(\mathcal{G}(x)) \approx x$ and $\mathcal{G}(\mathcal{F}(y)) \approx y$.

In order to achieve the two approximation, we hereby introduce the cycle consistency loss, which is a core concept in the paper,

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim Pr(x)} \left[\|\mathcal{F}(\mathcal{G}(x)) - x\|_1 \right] + \mathbb{E}_{y \sim Pr(y)} \left[\|\mathcal{G}(\mathcal{F}(y)) - y\|_1 \right]. \tag{4}$$

The equation 4 is incentivized by forward cycle consistency $x \to \mathcal{G}(x) \to \mathcal{F}(\mathcal{G}(x)) \approx x$ and backward cycle consistency $y \to \mathcal{F}(y) \to \mathcal{G}(\mathcal{F}(y)) \approx y$.

Furthermore, inspired by the principle of GAN naturally, we should introduce two adversarial disciminators, $\mathcal{D}_X$ for $\mathcal{G}$ and $\mathcal{D}_Y$ for $\mathcal{F}$. Therefore, the total objective function contains two types of loss: adversarial losses, for matching the distribution of synthesized images to the real data distribution in the target domain, and cycle consistency losses, for preventing the learned mappings $\mathcal{G}$ and $\mathcal{F}$ from contradicting each other. In other words, cycle consistency loss could guarantee us to obtain the original image by applying the composite mappings $\mathcal{F} \circ \mathcal{G}$ or $\mathcal{G} \circ \mathcal{F}$.

We hereby present the full objective function for Cycle GAN:

$$\mathcal{L}(\mathcal{G}, \mathcal{F}, \mathcal{D}_X, \mathcal{D}_Y) = \mathcal{L}_{GAN}(\mathcal{G}, \mathcal{D}_Y, X, Y) + \mathcal{L}_{GAN}(\mathcal{F}, \mathcal{D}_X, Y, X) + \lambda \mathcal{L}_{cyc}(\mathcal{G}, \mathcal{F}), \tag{5}$$

where λ controls the relative importance of the two types of losses. The target could be written as a min-max structure as follows

$$\arg \min_{\mathcal{G}, \mathcal{F}} \max_{\mathcal{D}_X, \mathcal{D}_Y} \mathcal{L}(\mathcal{G}, \mathcal{F}, \mathcal{D}_X, \mathcal{D}_Y).$$

3 Model Architecture and Objective Function

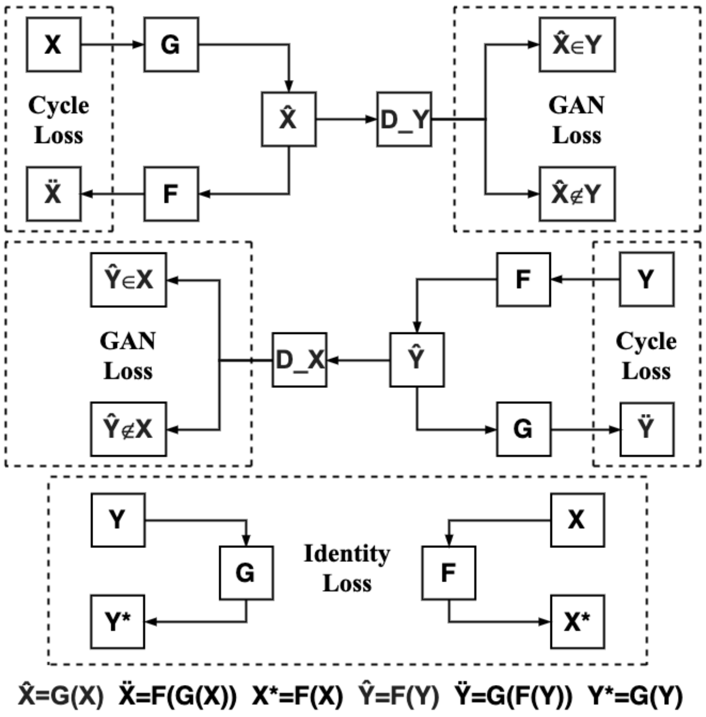


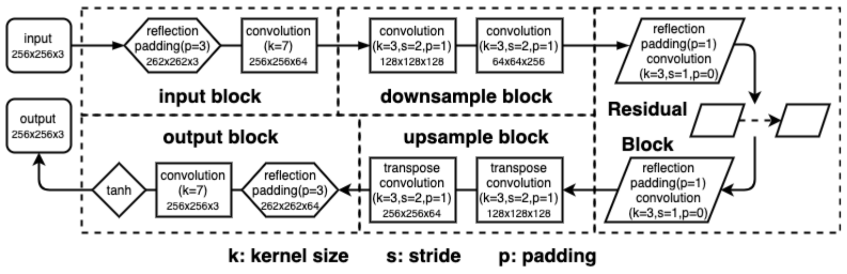
Figure 1: Cycle-GAN Structure

We display the model structure in Figure 1. The first and second parts of the picture (from top to bottom) explicitly present the structure of the cycle loss and GAN loss. The third part demonstrates the identity loss, a new type of loss that we have not mention before,

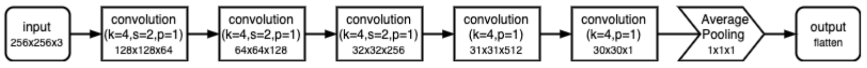
$$\mathcal{L}_{identity}(G, F) = \mathbb{E}_{x\sim Pr(x)}\left[\|\mathcal{F}(x) - x\|_1\right] + \mathbb{E}_{y\sim Pr(y)}\left[\|\mathcal{G}(y) - y\|_1\right].$$

The identity loss prevent the generators changing the tint of input images freely and unnecessarily, which is extremely useful for photo generation from paintings. As we could see from the formation of the identity loss, it encourages mappings to preserve original composition as much as possible.

The structure for generative network and discriminative network could be summarized in the Figure 2.



(a)



(b)

Figure 2: Generator & Discriminator Structure

4 Training Methods

- Optimizer: Adam Algorithm. [2]
- Activation Function: ReLU for the generator and Leaky ReLU for the discriminator.
- Normalization: The paper that we were trying to implement stated to use instance normalization. We used batch normalization since the batch size here was 1 and these two methods would be the same.
- Residual Network: In the generator, we included a residual block containing deep residual network [3] between the down-sampling and up-sampling blocks.
- Transpose Convolution: We introduced transpose convolution for the up-sampling block in the generator.
- Reflection Padding: The method of reflection padding was used in the input, output and residual blocks of generator. This method performed padding with existing values around the edges the image.
- Learning Rate Scheduler: The scheduler allowed the learning rate to start decaying after the decay epoch.
- Data Augmentation:
  - Resized 1.12 times larger with bicubic interpolation.
  - Randomly cropped into initial size of image.
  - Randomly horizontal flip with probability of 0.5
  - Normalization.

5 Hyperparameters

Dataset	**apple2orange**	**horse2zebra**	**summer2winter**	**monet2photo**	**vangogh2photo**
LR	0.0002	0.0002	0.0002	0.0002	0.0002
Batch Size	1	1	1	1	1
Epochs	150	152	199	80	80
Decay	120	80	110	75	75
$\lambda_{Identity}$	3	3	5	10	10
λ_{Cycle}	10	10	10	10	10
β in Adam	$(0.5, 0.999)$	$(0.5, 0.999)$	$(0.5, 0.999)$	$(0.5, 0.999)$	$(0.5, 0.999)$

Table 1: Parameters for each Dataset

6 Description of the Datasets

We accessed the datasets from the website of EECS department of UC Berkeley. [4]

Dataset	**apple2orange**	**horse2zebra**	**summer2winter**	**monet2photo**	**vangogh2photo**
X	Apple	Horse	Yosemite Summer	Monet Artwork	Vangogh Artwork
Y	Orange	Zebra	Yosemite Winter	Real Photo	Real Photo
X:train	996	1068	1232	1073	401
Y:train	1020	1335	963	6288	6288
X:test	267	121	310	122	401
Y:test	249	141	239	752	752

Table 2: Composition of Datasets

7 Computational cost

We did all the computation by using the NVIDIA GK110 (K20X) Kepler with 6GB of GPU memory on Blue Waters.

Dataset	**apple2orange**	**horse2zebra**	**summer2winter**	**monet2photo**	**vangogh2photo**
Time	$83.6h$	$118.2h$	$143.7h$	$258.2h$	$294.9h$

Table 3: Computational Cost for the Datasets

8 Results

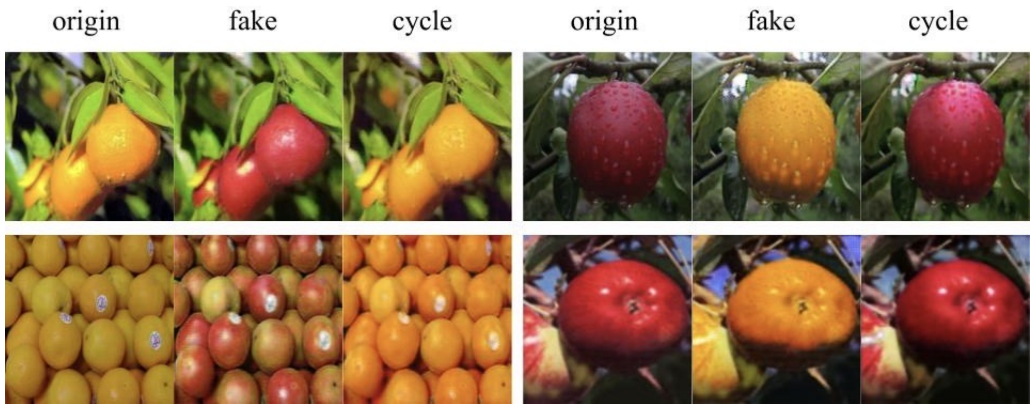


Figure 3: apple2orange

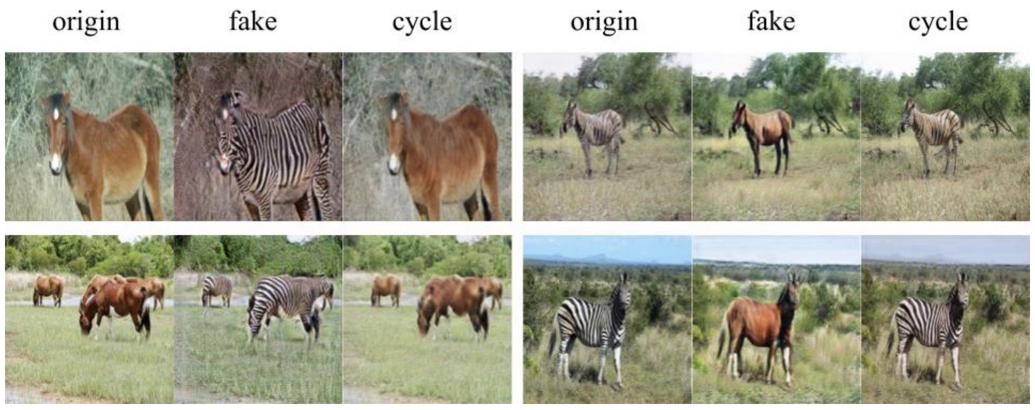


Figure 4: horse2zebra

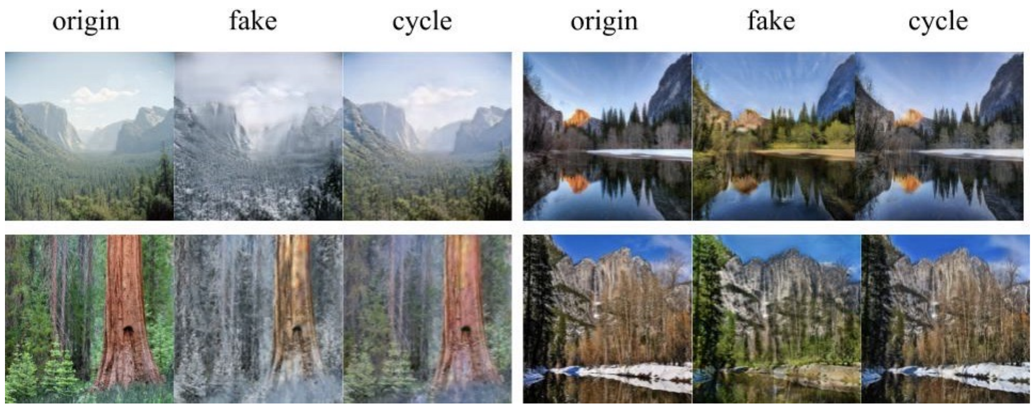


Figure 5: summer2winter_yosemite

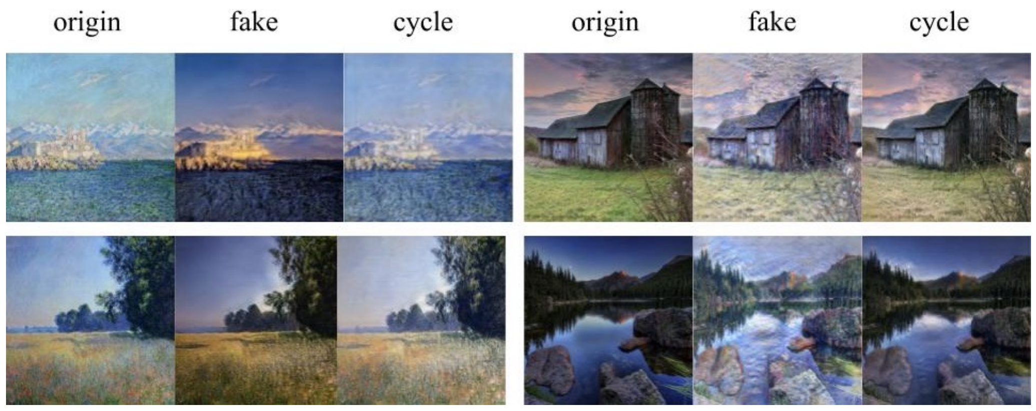


Figure 6: monet2photo

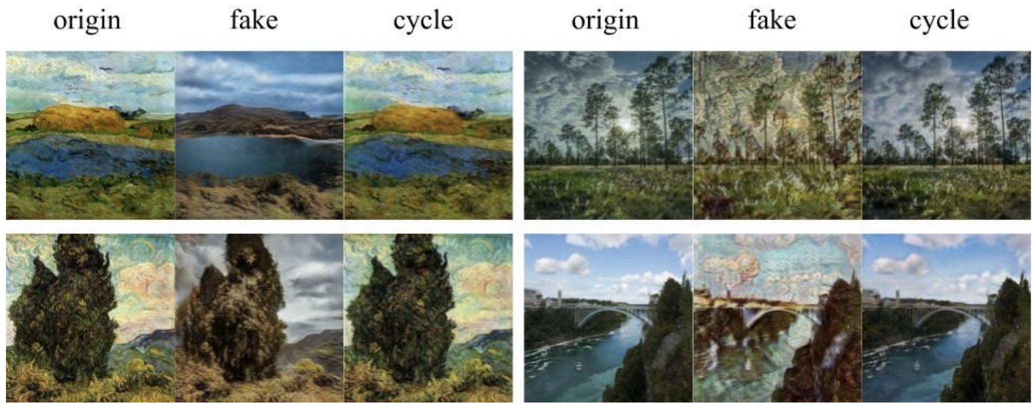


Figure 7: vangogh2photo

9 Description of Code

- **gen_model.py**

  It defines the class of the residual network and the class of the generator network model.

- **dis_model.py**

  It defines the class of the discriminator network model.

- **function.py**

  It defines three functions as classes: The 1^{st} one, is used for importing the images; the 2^{nd} one is used for creating replay buffer; and the 3^{rd} one is used for defining the learning rate scheduler which takes effect after the decay epoch.

- **trn.py**

  It imports all the classes from **gen_model.py**, **dis_model.py** and **function.py** in order to perform the training process. At the beginning, it defines the values for the hyperparameters and a function for initializing weights in the networks. Then it imports the training data along with data augmentation, and defines G, F as the generators and D_X, D_Y as the discriminators. Before the training, it also defines the optimizers and the loss. Since we train the $G\&F, D\_X, D\_Y$ separately, we also turned off the gradient requirements for those parameters which are not in training in order to save memory on GPU. For instance, The gradient requirement for D_X and D_Y are both turned off when $G\&F$ is in training.

- **tst.py**

  It imports some of the classes from **gen_model.py**, **dis_model.py** and **function.py** in order to perform the testing process. At first, it imports testing data in the same way as **trn.py** without data augmentation. Then it loads the trained G, F and applies G, F on variables in group X, Y to get fake X and fake Y. It also performs F, G on fake X, Y to get the cycle X and cycle Y. Lastly, all of these variables are converted back into original, fake and cycle images, and saved in different groups.

10 Public Sources

We borrowed ideas from:
`https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`
`https://github.com/aitorzip/PyTorch-CycleGAN`
`https://cyclegans.github.io/`
And rewrote most parts into our own version.

References

[1] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Taesung Park. Cyclegan dataset. Accessed on 2020-04-29, `https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/`.