Why "CycleGAN"?

The reason it is called a cycleGAN is that we will use the two GAN models to construct a cycle for some purpose. To make such cycle, we need to make the two GAN models to do exactly the same work, that is to do *image-to-image translation*, but in reversed directions. Specifically, we want GAN A to transfer a horse image to a zebra image, while GAN B has the ability to transfer a zebra image to a horse image. Each of the two GAN models is formed by a Conditional Generator and a PatchGAN Discriminator. We will talk about the structures of the generator and discriminators in the following section. To sum up, in our tool box we have two GAN models. Each of them contains one generator and one discriminator.

Why two GANs?

The major advantage of a cycleGAN: it won't require pair samples anymore in the training process.

One merit of paired samples

They put strict restriction on the generated images so that except the desired adjustments (e.g. zebra pattern), the rest part of the generated images will be unchanged from the source images.

Although we take this restriction for granted when we have paired samples because it comes naturally with the strongly correlated paired images, it is not the case anymore when we switch to unpaired samples where there is no correlation between source images and target images.

Cycle Consistency

Put the restriction back on generated images through an additional loss function.

> We exploit the property that translation should be "cycle consistent", in the sense that if we translate, e.g., a sentence from English to French, and then translate it back from French to English, we should arrive back at the original sentence.
>
> — Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

All we need to do then is convert the difference between the two horse images to a loss function.

This process is called Forward Cycle, a half of the cycle structure which focuses on evaluating GAN A. The other half of the cycle, which is called Backward Cycle, will have exactly the same workflow of its counterpart but in a reversed direction so that it could evaluates GAN B.

Combining the two loss functions from Forward and Backward Cycles, we obtain a general **loss function** called *Cycle Consistency* Loss and we will involve it as a part of the final loss function which we will use to update the cycleGAN model.

Identity Loss

While the outcome zebras seem to have the same shape and other stuff with the input horses under cycle consistency restriction, the color profile consistency between inputs and outputs was not maintained very well by the model.

To solve this problem, they invented another structure called **Identity Mapping**.

> One of them focuses on the performance of GAN A and the other one evaluates GAN B.
>
> To evaluate GAN A, for example, instead of feeding it with a horse image, we will give it a zebra image.
>
> By its nature, GAN A will generate a new zebra image based on the original zebra. Ideally, all the stuff laying on the two images, including the colors, should look exactly the same.

Loss function

Again, we convert the difference between the two images to a loss function and combine it with the loss function from the other components to form an Identity Loss function, which will also be a part of the final loss function.

Discriminator

A fully CNN with totally five layer blocks where each block contains a 2D Convolution Layer with kernel size of 4x4 and stride of 2, an Instance Normalization Layer and a Leaky ReLU Layer (except the output block which uses a Sigmoid Layer as activation).

Why 16×16 matrix ?

This is because in cycleGAN, we are using a structure called PatchGAN for the discriminator.

Unlike a normal discriminator which takes an image as a whole to judge whether it is real or fake, a PatchGAN discriminator will first divide the input image into a number of "patches" and then make ***individual judgements*** on each of them.

In our case, the input image is divided into 16x16 patches in the end where each patch has a size of 70x70.

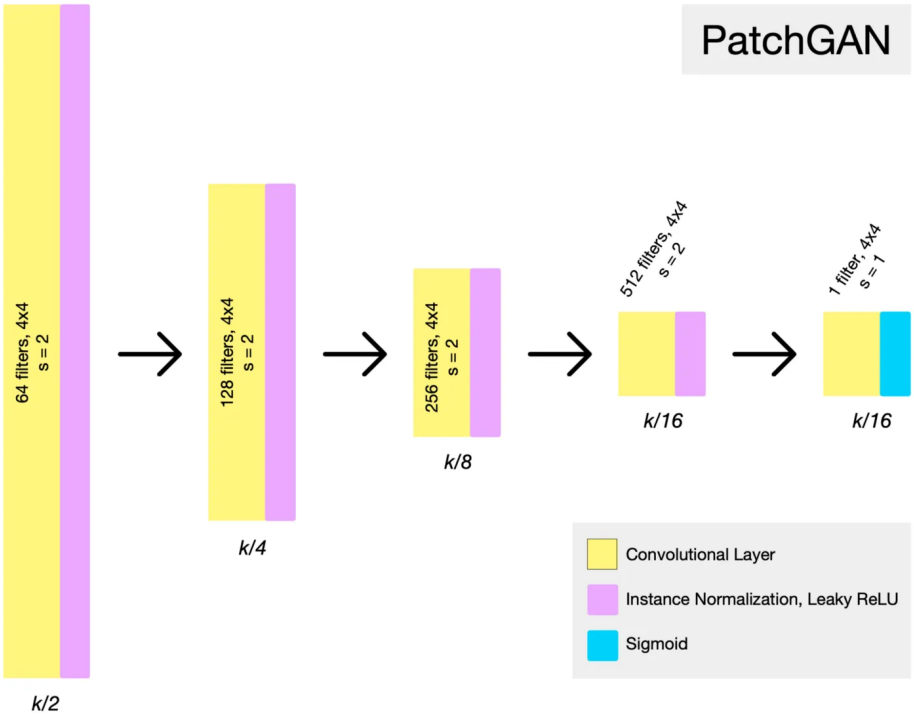

Figure 1: PatchGAN

Generator

A neural network structure which is formed by three sections:

- Encoder
  - Layer block × 3

    The first layer block doesn't change the size of image, it only creates 64 channels on the input. The next two layer blocks however, each shrinkages the input size by half while also double the number of channels.

    * 2D Convolution Layer × 1
    * Instance Normalization Layer × 1
    * Leaky ReLU Layer × 1
- Transformer
  - ResNet × 6

    each contains

    * Layer block × 2
      · 2D Convolution Layer (with stride=1) × 1
      · Instance Normalization Layer × 1
      · Leaky ReLU Layer. × 1

      and

      · 2D Convolution Layer (with stride=1) × 1
      · Instance Normalization Layer × 1
- Decoder

  The decoder enlarges the input back to the original size and collapses all channels into RGB to form the final output image.

  - Transpose Convolution Layers × 2

    Enlarge the size of input while decrease the number of channels

  - Output layer × 1

    Collapse the channels into RGB and thus output it as the final output image

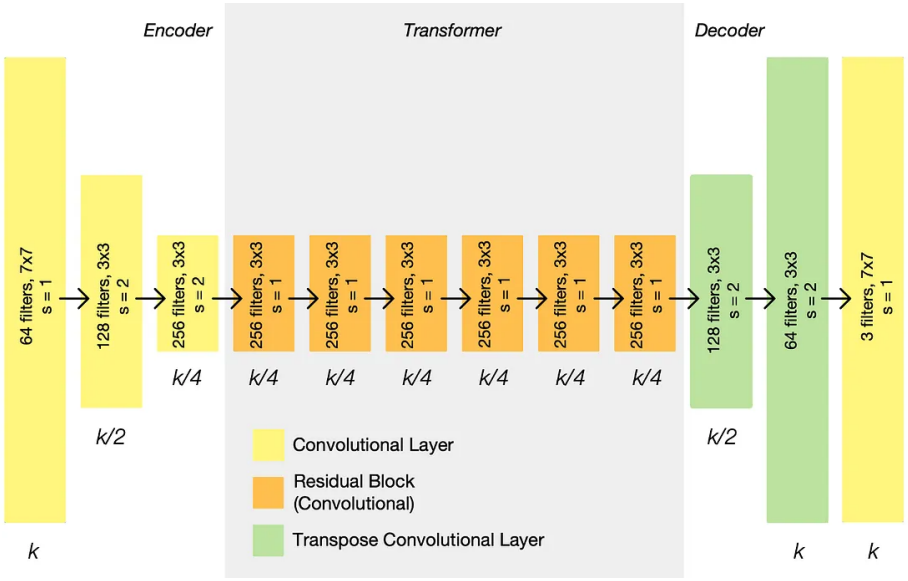

Figure 2: Generator

Update Weight for a Generator Through a Composite Model

> Traditionally, a composite model generates only one loss function — the ***adversarial loss*** which reflects how well the discriminator identifies fake images produced by the generator.

For a cycleGAN generator, however, there are three other loss functions should be involved — the ***identity loss*** and ***forward/backward consistency loss***.

There are 4 independent components which produces the 4 loss functions.

While the ***adversarial loss*** is represented by L2 loss function (Mean Square Error, MSE), the rest three are represented by L1.

The four losses are weight averaged to obtain the final loss function: The ***cycle consistency loss*** has the highest weight while the effect from ***adversarial loss*** is downplayed.

At the end of each iteration, we record the final loss function and use Adam SGD to update model weight of the generator.

Adversarial Loss

For the mapping function $G : X \to Y$ and its discriminator D_Y, we express the objective as:

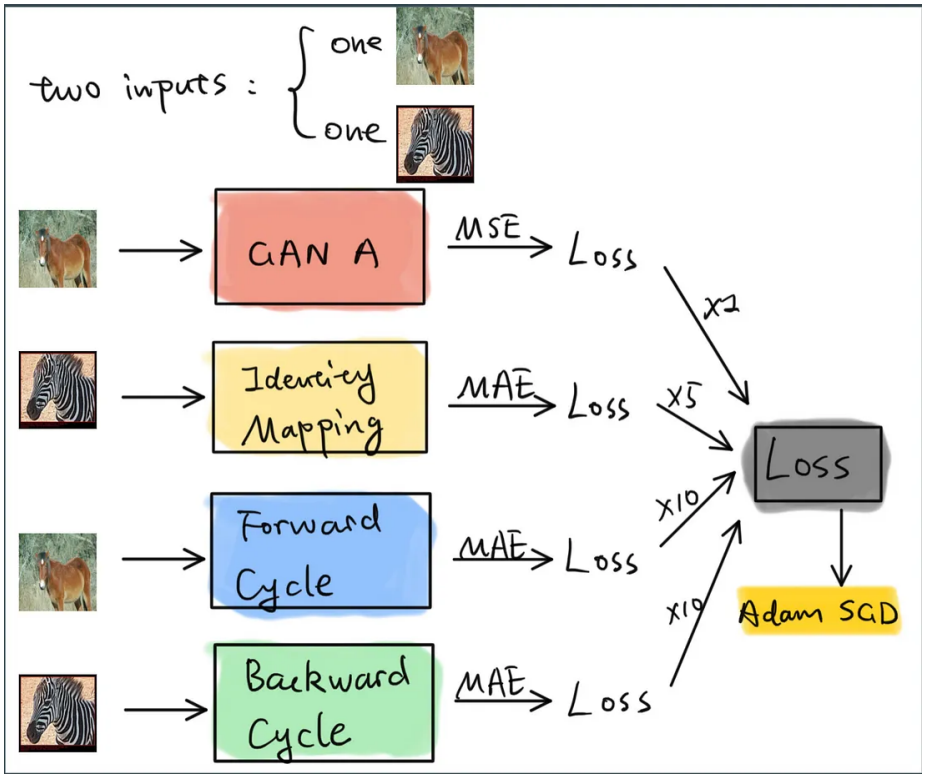

Figure 3: Update Weights

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

where G tries to generate images $G(x)$ that look similar to images from domain Y, while D_Y aims to distinguish between translated samples $G(X)$ and real samples y.

Cycle Consistency Loss

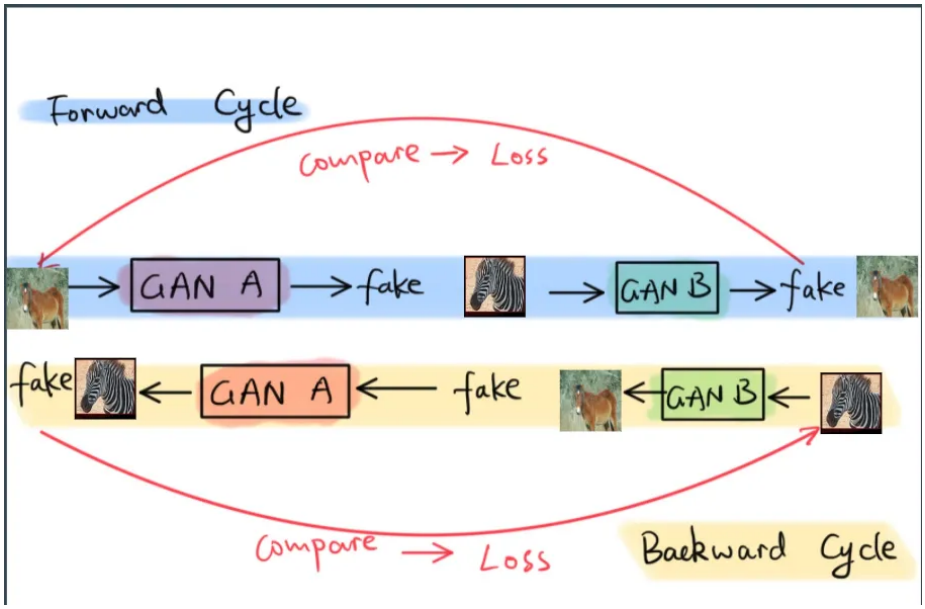

Figure 4: Cycle Consistency Loss

Forward cycle consistency

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(x)) - y\|_1]$$

Functions should be cycle-consistent: for each image x from domain X, the image translation cycle should be able to bring x back to the original image.

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

Backward cycle consistency

Similarly, for each image y from domain Y, G and F should also satisfy backward cycle consistency:

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

Identity Loss

It is an additional loss to encourage the mapping to preserve color composition between the input and output.

$$\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(x) - x\|_1]$$

Full object

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F)$$

Update Weight for a Discriminator Using Image Pool

An **image pool** is a set of at most 50 fake images produced by the generator in the past iterations.

The goal of image pools is to prevent cycleGAN model from changing drastically from iteration to iteration.

Once we have the two images ready, all we need to do is put them to the discriminator separately and sum up the two loss functions represented by L2. We record the loss sum and update the weight of the discriminator using Adam SGD.

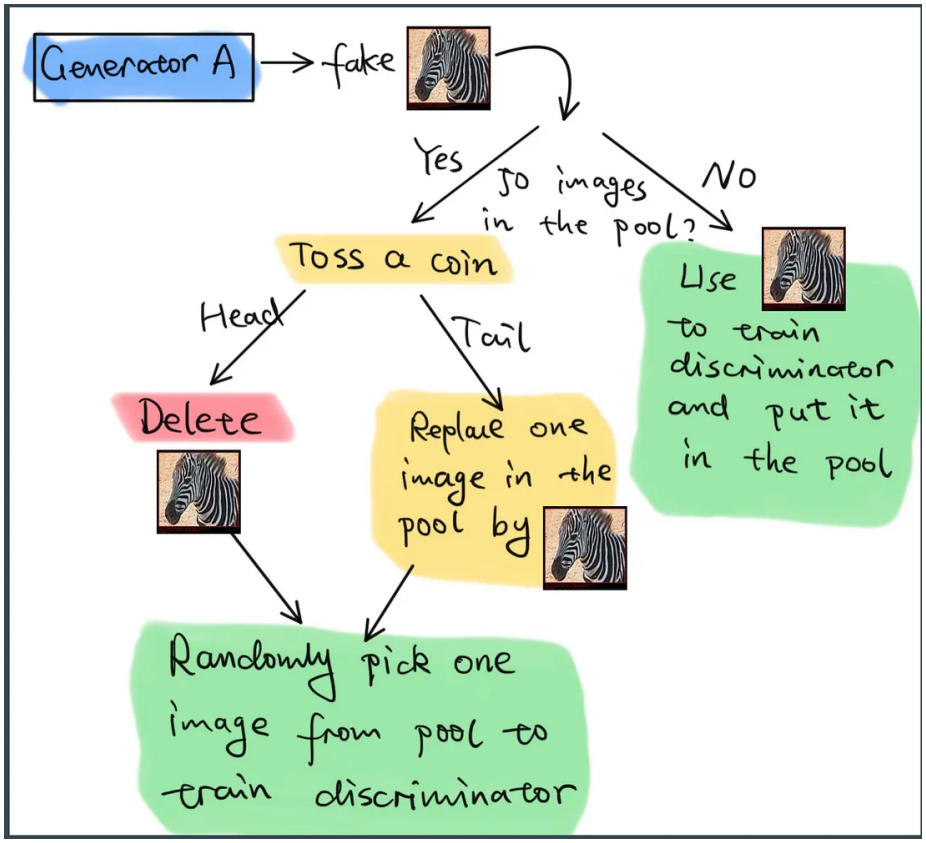

Figure 5: Update Weight for a Discriminator