VISION AND PERCEPTION

# CYCLEGAN

RICCARDO GOZZOVELLI

1849977

ACCADEMIC YEAR 2018-2019

CycleGANs are a particular implementation of the Generative Adversial Networks. They are used in situations where there is a lack of paired training data. In GANs the task is to learn the distribution of some training data (X) given a random distribution,generally noise (Z):

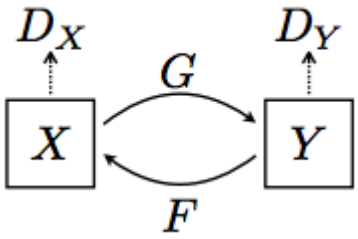
$$f : Z \rightarrow X$$

This is done by exploiting the labels associated to the training data. In CycleGANs instead we want to learn an invertible mapping between two different data distribution (X and Y):

$$g : X \rightarrow Y$$

$$f : Y \rightarrow X$$

In this case the label for each training sample will be a correspondent image in the other domain.
As a consequence, the architecture of the CycleGANs is slightly different from the one of simple GANs. Here we exploit pairs of Generators and Discriminators:




The generator G learns the mapping from the data distribution X to the data distribution Y, while the generator F learns the inverse mapping. At the same time two discriminators, Dx and Dy, verify how good are the samples generated by the two generators. Therefore the losses that the model uses to improve its perfomances are the same of the ones used in GANs, the loss on the real data and the loss on the generated data. The key feature of CycleGANs is the presence of an additional loss called “Cycle Consistency Loss”. Given an image X, the generator G will produce as output a new image $Y^I$ . If we then feed this new image to the other generator F, we obtain another image $X^I$ that should be exactly equal to the first one:

$$g : X \rightarrow Y' \rightarrow \rightarrow f : Y' \rightarrow X' \approx X$$

The difference between X and $X^I$ (and therefore Y and $Y^I$ ) is the Cycle Consistency Loss. Summarizing, the losses that we compute are:

- $L_{GAN}(G, D_Y, X, Y) = E_{y \sim pdata(y)}[\log(D_y(y))] + E_{x \sim pdata(x)}[\log(1 - D_y(G(x)))]$ which is the loss of the discriminator $D_Y$ with respect to fake and real input Y.

- $L_{GAN}(F, D_X, Y, X) = E_{x \sim pdata(x)}[\log(D_x(x))] + E_{y \sim pdata(y)}[\log(1 - D_x(G(y)))]$ which is the loss of the discrminator Dx with respect to fake and real input X.

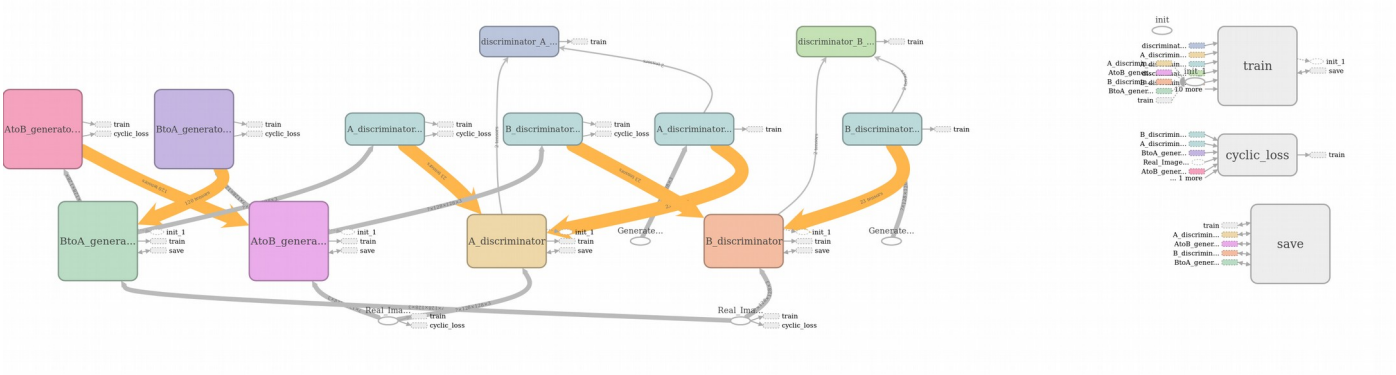
- $L_{cyc}(G, F) = E_{x \sim pdata(x)}[norm_1(F(G(x)) - x)] + E_{y \sim pdata(y)}[norm_1(F(G(y)) - y)]$ which is the Cycle Consistency Loss.

- The total loss is
  $L(G, F, D_x, D_y) = L_{GAN}(G, D_y, X, Y) + L_{GAN}(F, D_x, Y, X) + lambda * L_{cyc}(G, F)$
  where lambda is a weight parameter with value 10.

The objective function used for CycleGANs is:

$$G', F' = arg\,min_{G,F}\, max_{D_x, D_y} L(G, F, D_x, D_y)$$

We are therefore playing a min-max game where the discriminators try to master how to distinguish between fake and real images, while generators try to learn how to produce images as close as possible to the real ones. The graph of the CycleGAN architecture that we implemented can be seen in the following figure:




Concerning the training data, we used the “summer2winter_yosemite” dataset. The task, as the name of the dataset hints, is to transform images of the Yosemite National Park from a summer setting to a winter setting and viceversa. Those are some of the best images that the model has been able to produce after training it for 100 epochs:

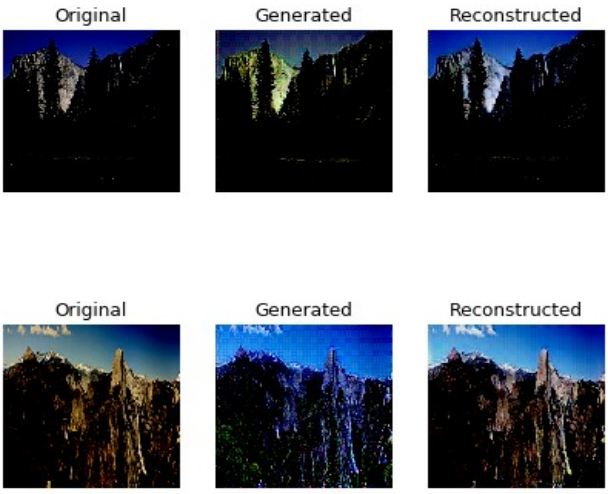



Image 1. The generated image on the second row is more “bluish” than the originial one but still it presents trees with green leaves.

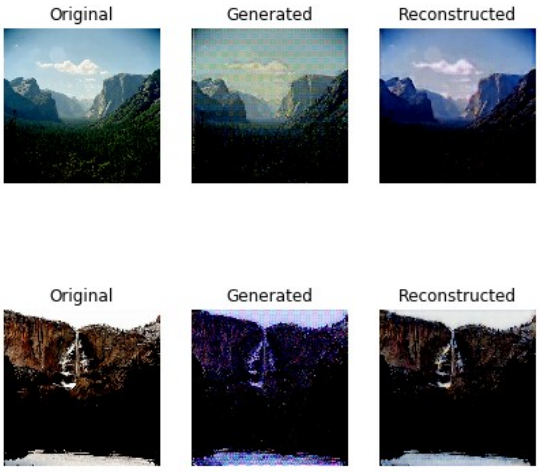



Image 2. The generated image on the presents more cold colors than the original ones.

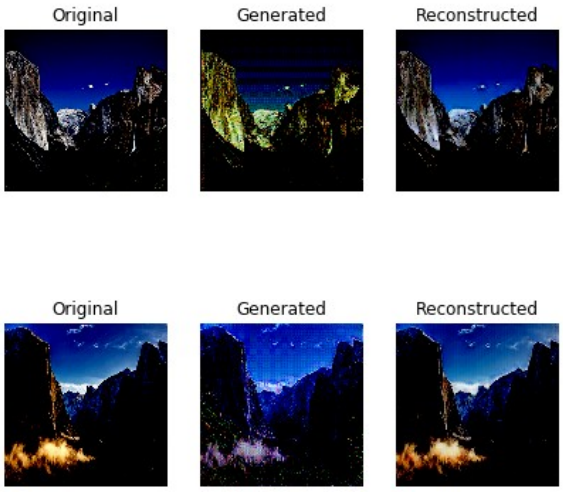



Image 3. The generated image on the second row is again colder than the original one.

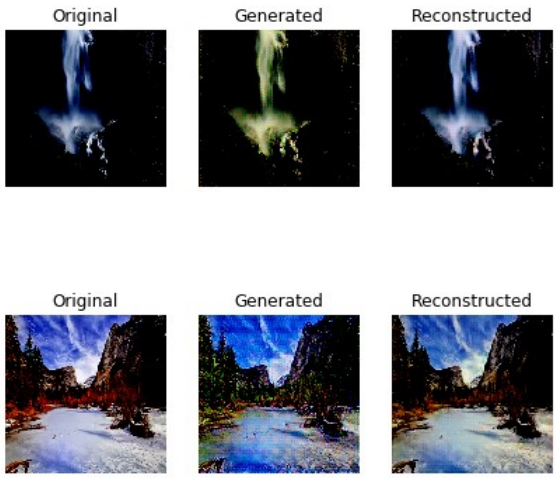



Image 4. The generate image on the second row starts to present hotter colors than the original one.

Those results have been obtained by following different techniques that are very common in the field of GANs. First of all we limited for the first 60 epochs the number of updates related to the parameters of the discrimators, which where set to 2 updates per 10 epochs. This is because the discriminators were too powerful, they were able to distinguish very easily any product created by the generators thus restricting the possibility for the latters to learn. But this has also a disadvantage, which is that a generator might learn to produce wrong images. So after the 60th epoch we increased the number of updates for the discriminators to 5 per 10 epochs. A second technique that we used is again related to reduce the efficiency of the discriminators. We added to the inputs of the discriminators some Gaussian noise with 0 mean and 0.1 standard deviation in order to mislead them. Next we flipped the labels for computing the losses, generated images were associated to label 1, while original images to label 0. It is still not known why this helps the learning but it worked for our case. Finally we avoided the use of a sigmoid activation function for the last layer of the discriminators. The reason why we did this is because the sigmoid function converts every label to either 1 or 0, thus the information contained in very low and very high samples is truncated.

Concerning the results obtained, we can see from the figures previously showed that not all the generated images are correct. Some of the generated images on the first row are far from being associable to a winter setting. Even in the “correct” images we can sometimes see that there is something wrong. For instance some images are not completely smooth and presents visible shapes of the kernels used for the various layers. Some other instead are completely identical to the original, and this is true in particular for images that are already associated to a winter setting. The causes of this poor results can be be grouped under different basis. First of all GANs requires a lot of training in order to obtain significant results. We didn’t proceeded over 100 epochs because the model started to diverge, that is the generator loss was increasing while the discriminator loss decreased. Finally, we might have not tune correctly the hyperparameters for the various components of the model (filters, learning rate, number of layers, and so on). We did tried different combination of those hyperparameters, but the only one that improved our perfomances was to increase the size of the input images passing from 128x128x3 to 256x256x3 and adding 3 additional transformation blocks in the generator (reaching a total of 9 transformation blocks).