Image translation using generative adversarial networks: Review and Test

Pirashanth RATNAMOGAN
MVA ENS Paris-Saclay
pirashanth.ratnamogan@ens-paris-saclay.fr

Othmane SAYEM
MVA ENS Paris Saclay
othmane.sayem@ens-paris-saclay.fr

Abstract

In the paper, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" (Zhu, Jun-Yan, et al.) propose a Generative Adversarial Network based approach that allows to translate an image from a source domain to a target one. In this final project of the Object Recognition and Computer Class, we will first go through the basic ideas of this article. Then we will try to apply it on a totally new dataset before trying to use the Wasserstein distance as a better cost function.

1. Introduction

Style transfer from an image to another one is a really hot topic. It aims at learning the distribution that represent a class of images and transfer it to another set (transforming horses to zebras for instance). However, for now, most of the efficient approaches focus on supervised learning tasks where you have pairs of input and output data. This approach is impossible in various problems where we don't have that kind of pairs. The Cycle GAN (Zhu, Jun-Yan, et al.) [3] tries to tackle this problem. It uses the Generative Adversarial Networks (GAN) approach to learn two mappings. The first one G learns a map from a source domain X to another domain Y such that the distribution $G(X)$ is really close to the distribution given by Y. The second one F learns in the same way a mapping from Y to X. The article brings the "cycle constency" in their training that means they try to impose $F(G(X)) \simeq X$. This approach brings pretty good outcomes. Tests on a new dataset, that aims to use style transfer to create pokemon from animals and reversely, has been performed. We have also tried to improve the cost function used for the cycle consistency using the Wassertein distance.

2. Generative Adversarial Networks

Generative Adversarial Networks is a popular method to train a generative model. It uses an adversarial training between a discriminator and a generator that aims at providing an efficient loss to the generator. The discriminator is trained to distinguish images that are from the original dataset from the one that has been generated by the generator. The generator is trained to fool the discriminator and generate samples that cannot be detected as generated samples by the discriminator. This method has been a big improvement in training generative models. Discriminator and generator networks architecture must be adapted depending on the wanted task. Hence, in [3], it uses an adaptation of the article [2].

3. CycleGAN for Image translation

3.1. First explanations

CycleGAN aims at transfering style from an image to another one. It denotes by X and Y the two domains from which we want to learn the mapping functions. It gives training sample $\{(x_i)\}_{i=1}^{N}$,$\{(y_j)\}_{j=1}^{M}$ with $x_i \in X, y_j \in Y$. It aims at finding the distribution G that allows to go from the original inputs $\{(x_i)\}_{i=1}^{N}$ to a set of images with a distribution close to the target one $\{(y_j)\}_{j=1}^{M}$. To compute such efficient generators, the author uses a GAN architecture that allows to iteratively improve by learning the loss function used to train the generator G. In order to improve the quality of this mapping G the author trains in the same way a mapping F that should allow to go from the original inputs $\{(y_j)\}_{j=1}^{M}$ to images with a distribution close to $\{(x_i)\}_{i=1}^{N}$. This two generators are trained to do the exact reverse operation, hence the authors add a cycle consistency loss that should enforce the operation $F(G(x_i)) \simeq x_i$ and $G(F(y_j)) \simeq y_j$ for all x_i in the domain X and y_j in the domain Y.

3.2. Computed Loss and original implementation

As described in the previous section the loss used to compute our models will be given by a sum of three term :

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) \quad (1)$$
$$+ \mathcal{L}_{GAN}(F, D_X, X, Y) \quad (2)$$
$$+ \lambda \mathcal{L}_{cyc}(G, F) \quad (3)$$

Terms (1) and (2) are the GAN based generator loss. As we can see it depends on the trained discriminators D_X, D_Y that are trained to find images which have distribution close to the one of the domain X or Y. The term (3) is the described cycle consistency loss that is defined as follow:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] \quad (4)$$
$$+ \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1] \quad (5)$$

For the generators architecture, authors used a network that has provided good outcome in neural style transfer [2]. For the discriminator, they used the so-called PatchGANs with size 70x70 that aims at detecting if the image is fake or not from 70x70 patches.

4. Basic tests and implementation

In order to improve our undersanding of what the author did we tried several really simple tests.

4.1. Apply the code on existing dataset

We have struggled to apply this costly implementation multiple time but finally we used both a floydhub account (that gives some hours of free GPU just by signing up) and a computer with a 12Go Nvidia Titan X that we have managed to use during the end of the project. For the project we will use the original Torch implementation. We have tried the original implementation on the 'horse2zebra dataset' that aims at making horses looking alike zebras (we have also worked with apple2oranges dataset). The best training that we have achieved was done by applying the algorithm with the parameters (epoch = 32 (original = 200) , batch_size= 1, discriminator= 'basic' i.e simple convolutions, generator = 'resnet_6blocks'). The author claims that this kind of architecture well-suits the texture transfer problem. Here you have an image that we obtain during the unsupervised training time:

Figure 1. Left original image , center generated image, right reconstructed image (cycle) - epoch=32

4.2. Naively change the architecture of the neural network

The code that we used naively allows to use state-of-the-art architectures for both generators and discriminator. As generator we can easily switch between Resnet, Unet and an encoder-decoder or tune what we want We have performed various tests in order to see what performs best. It's hard to say which architecture performs best because outcome depends on initialization and the way of comparing and judging visually outputs can be really different. However from what we saw, in the datasets horse2zebra and orange2applen, where the goal was to transfer color and texture, Resnet based generator seems to provide the best outcomes. We have tried to run the bi-gan model removing the cycle consistency loss. Cycle consistency loss appears to be really important. One can find some graphics that we have obtained in the annex 6.4.

5. Algorithm in a new dataset

We have tested the original code on a totally new task: to generate Pokemons from animal and reversely. For the Pokemon domain images we used the dataset provided in the video "Generating Pokemon with a Generative Adversarial Network" by Siraj Raval on Youtube (100 images). We also added 100 extra images manually downloaded from Google. We haven't found any interesting dataset that provides animals without backgrounds. Hence we have manually download 200 images in Google/Bing images search. You can find the built datasets here: *https://drive.google.com/drive/folders/17A1IXkaxP9krNxAiGOmQUZxiJaLE0fVp?usp=sharing*.
Our datasets are small in comparison with the ones used in the article tests. It's because the dataset have been created manually and we have quickly been in lack of relevant images in Google/Bing. We have used pictures without backgrounds to reduce the task difficulty. What we thought was that the network could learn kind of texture and style that makes how pokemons are and how animals are. Hence we kept the original architecture with a Resnet based generator. After several tests we found the setting that provides the best outcome from our point of view (visual). As a generator we have used a Resnet with 9 blocks as a generator. We used a training time of 200 epochs ($\lambda_{cyc} = 10$, and Lsgan architecture for GAN). The final outcome looks quite good: the texture seems realistic but the shapes haven't been changed. One can imagine that training during more epochs and with a wider dataset could allow us to improve the transformation. You can find some more outcome in the dedicated annex 6.4.

6. Improve the loss function using the Wasserstein distance

In order to improve our results, and to test other loss functions, we implemented a new version of CycleGAN, where the loss function of the two GAN's is computed using Wasserstein Distance (Earth Moving) [1].

6.1. Definition of Wasserstein distance

By introducing $\mathcal{X}$ a compact metric set, and Prob($\mathcal{X}$) the set of probability measures defined on it, we define the *Wasserstein-1* or *Earth-Moving* distance, for $p, q \in \mathcal{X}$, as following :

$$W_1(p,q) = \inf_{\gamma \in \pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x-y||\right]$$

Using the Kantorovich-Rubinstein duality results, we can deduce that the Wasserstein-1 distance can also take the following form :

$$W_1(p,q) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim p}\left[f(x)\right] - \mathbb{E}_{x\sim q}\left[f(x)\right]$$

Where f are 1-lipschitz functions defined on : $\mathcal{X} \to \mathbb{R}$
In the case of $\mathbb{P}_r$ a random distribution, and $\mathbb{P}_\theta$ the distribution of $g_\theta(z)$ a random variable of density p, we can prove that there exists f^* a solution to this optimization problem, which verifies :

$$\Delta_\theta W_1(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}\left[\Delta_\theta f^*(g_\theta(z))\right]$$

Using a neural network parametrized with weights lying in a space $\mathcal{W}$, we can estimate f^* by training it and backpropagating using the gradient defined above..

6.2. Implementing the algorithm

Inspired by various papers and code, we coded a CycleWGAN on top of the original Cycle GAN implementation. The main changes that we did were, for each GAN training of our cycle were:

- Replace the probabilistic output of the discriminator (sigmoid) by simply the difference between the real and fake outputs.
- As explained in the algorithm of the WGAN paper, train the critic n times before each generation.
- Clipping the weights of the critic function (to have the parameters laying in a compact space).
- Change Adam Optimizer to non momentum optimizers : here we chose to use RMSprop and set a low learning rate.

The code is available in the project drive: https://drive.google.com/drive/folders/17A1IXkaxP9krNxAiGOmQUZxiJaLE0fVp?usp=sharing

6.3. Advantages of CycleWGAN

In addition to the fact that Wasserstein distance provides clean gradients on all space, and allows to avoid situations of vanishing gradient (Which is the case for JS distance for example), the use of Wasserstein distance in GAN models has two main advantages. First of all, it provides a convergence stability to the algorithm, since the differentiability property allows us to train the critic till optimality, and doesn't quickly saturate as other distance discriminators. Moreover, using EM distance also allows us to have a loss function that is correlated with the visual quality of the results (decreases when the results are visually good), which provides us a tool to evaluate generative models.

6.4. Results of experiments:

We run the CycleWGAN on the Horse2zebra and Apple2Orange datasets. Some of the results are presented in section 6.4. We see that the outcome is pretty fair, regarding the number of epochs used.
As explained above, one of the main advantages of using EM distance is providing a meaningful loss function that correlates with the visual quality of the output. We have plotted the different loss functions for each GAN components of our Cycle GAN and for Cycle WGAN while training on horse2zebra datasets, the plot images can be seen on the annex in the Figure 2 and Figure 3.
As we can see, in Cycle GAN with LSGAN, the generator losses of A and B saturated quickly (just after one epoch), while in the CycleWGAN, the Wasserstein loss decreases with the number of epochs, and the critic trains till optimality (after 6 epochs in the presented case). This confirms the theoretical result.
Even if our critic converges to an optimal loss, the generated images using CycleWGAN aren't necessarily good. Actually, we noticed that this output presents a lot of noise compared to the cycleGAN output. This could be due to the clipping done to represents the Lipschitz constraint that seems to play quite an important role. For instance, when we changed the clipping interval from $[-0.01, 0.01]$, to $[-0.1, 0.1]$, we noticed changes of quality.

References

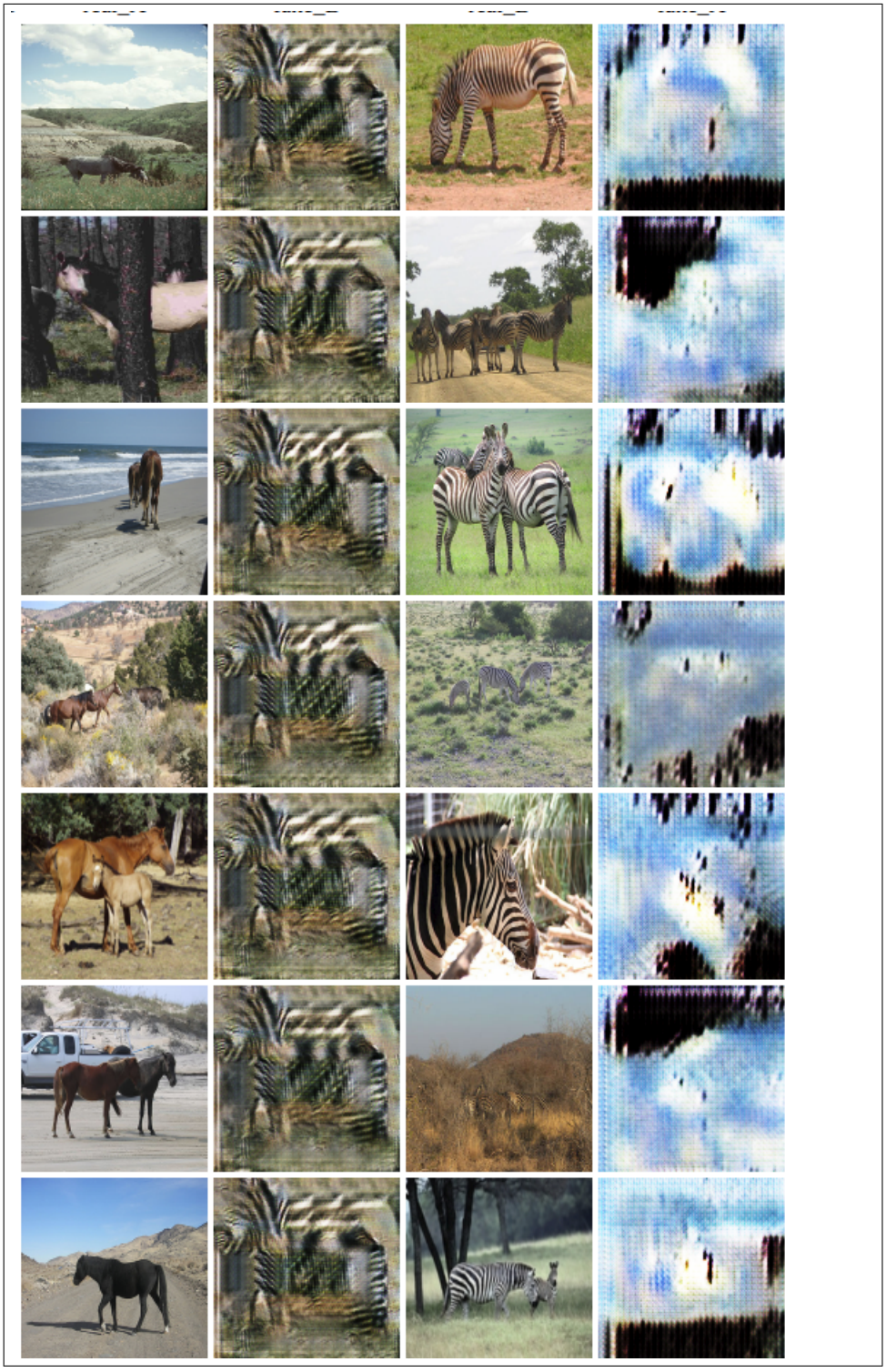
[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[3] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.

Appendix 1 - Horse 2 zebras bigan model (without cycle consistency loss

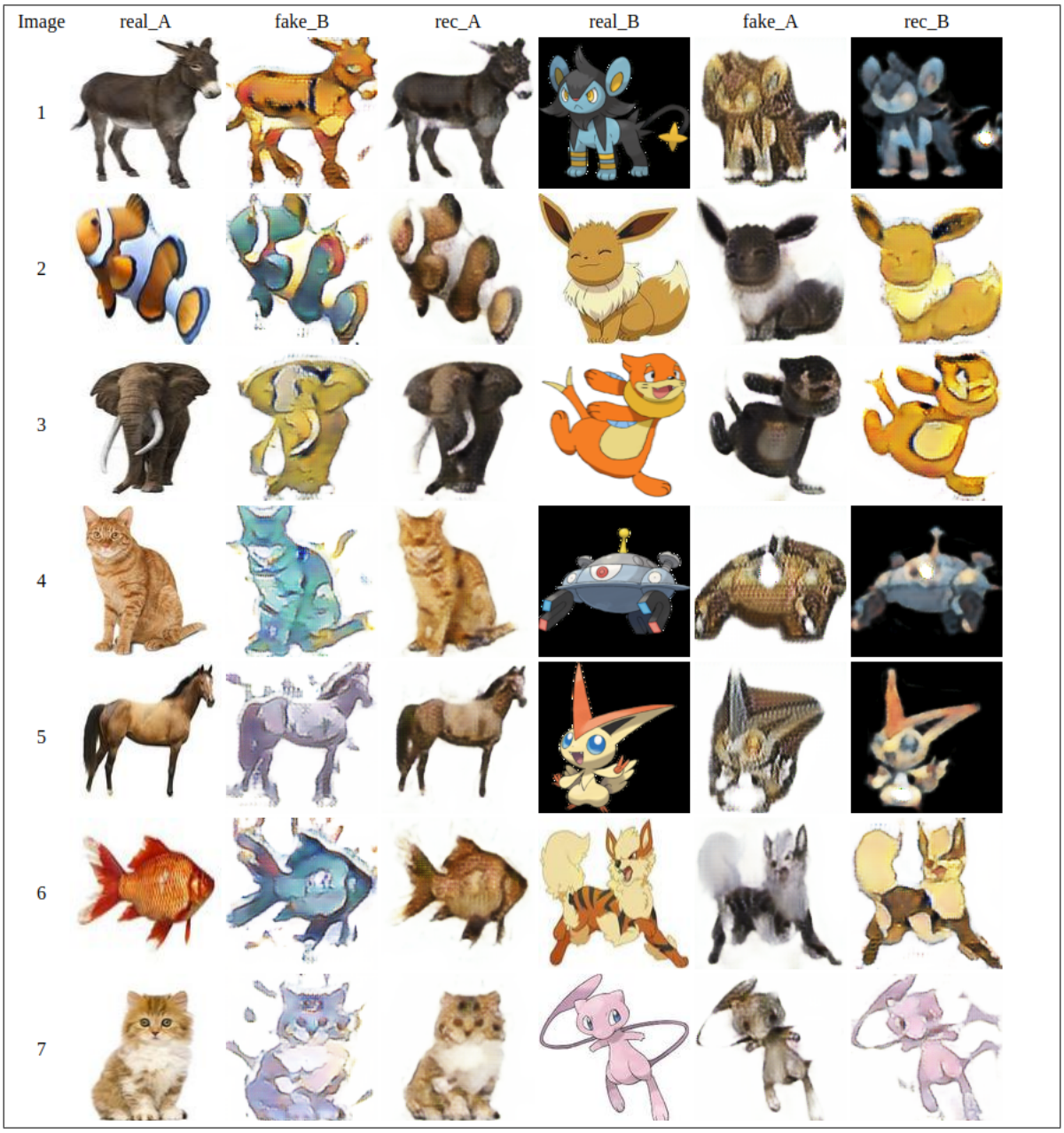
We have trained the network during 30 epochs.



Appendix 2 - Animal2pokemon outcome

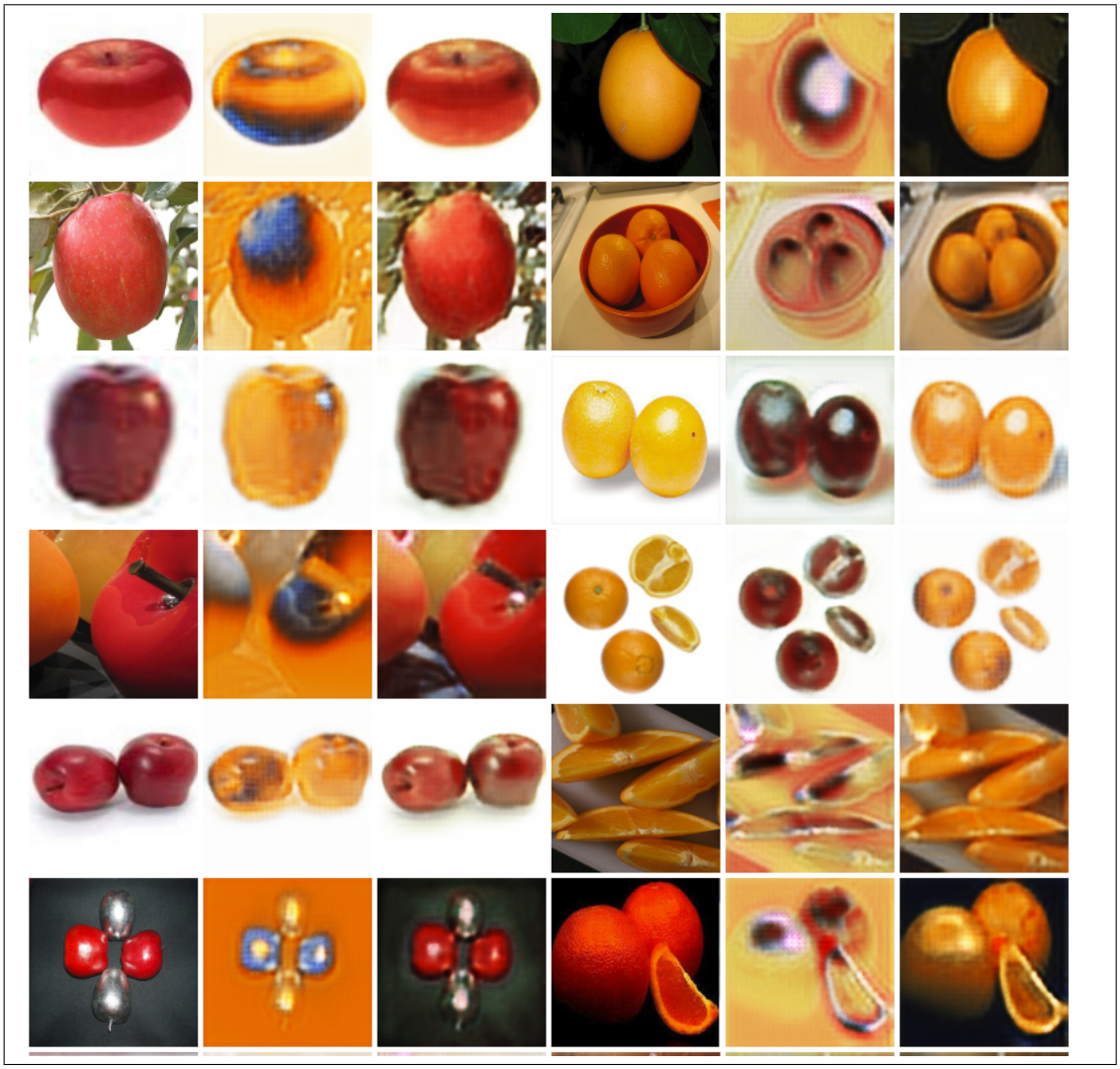
You can see complete tests here: https://drive.google.com/drive/folders/17A1IXkaxP9krNxAiGOmQUZxiJaLE0fVp
Some funny results are in the slides as well

For animal2pokemon database after 200 epochs :



Appendix 3 - Orange2apple outcome with WGAN

We have trained the network during 30 epochs. We have clipped weights in [-0.01,0.01]. One can compare the outcome obtained with the normal cyclegan present in the project drive. Visually, the outcome that we have obtained with cycle GAN with lsgan loss seems more realistic. However, the outcome seems faire.



Appendix 4 - Losses comparison

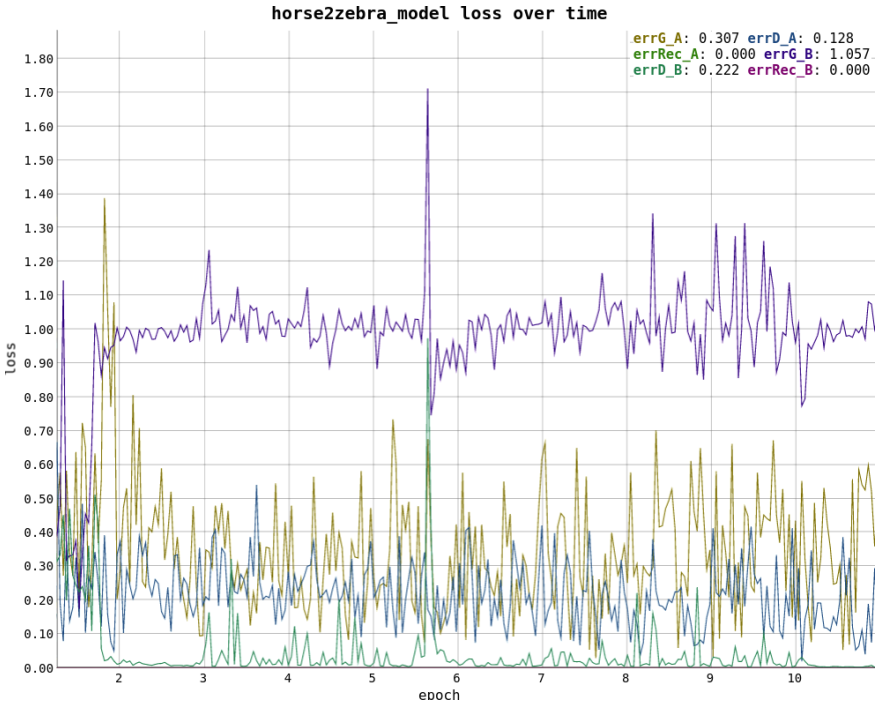


Figure 2. Loss function for the Generator and Discrimator of GAN A and B on horse2zebra dataset, The reconstruction error isn't visualized here.

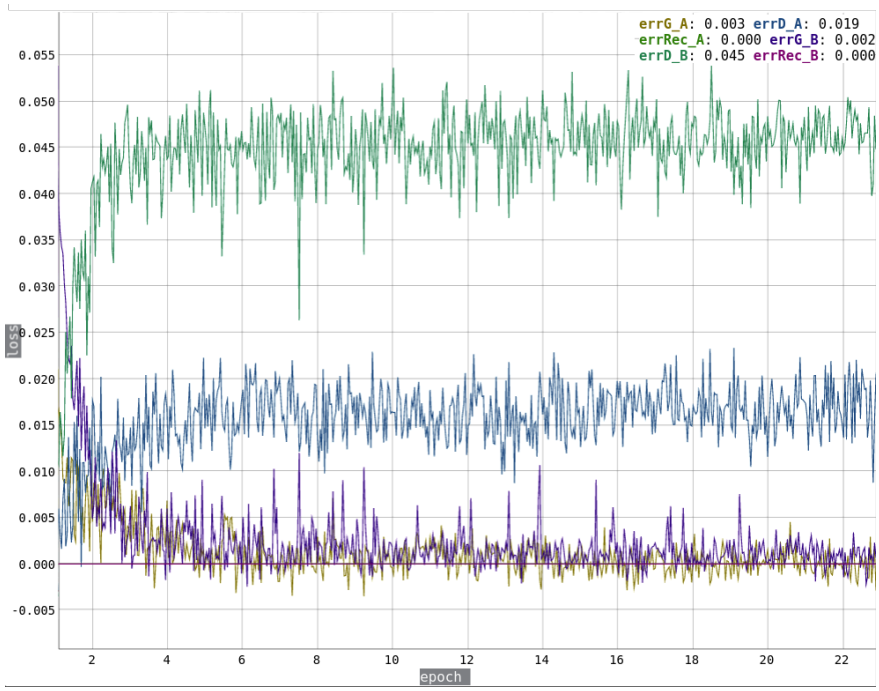


Figure 3. CycleWGAN :Loss function for the Generator and the critic of WGAN A and B on horse2zebra dataset, The reconstruction error isn't visualized here.