SAD2HAPPY AND FACE2CARTOON IMAGE TRANSLATION USING GENERATIVE ADVERSARIAL NETWORKS (GANS)

Ramzi MISSAOUI
M2 Data Science @ Ecole Polytechnque
ramzi.missaoui@telecom-paristech.fr

January 26, 2019

Abstract

Yann LeCun called adversarial training "the most interesting idea in the last 10 years in ML."

In this final project of Object Recognition and Computer Vision course, after explaining how the GANs and CycleGANs work, I will reproduce some of the results of the paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [6] using two of the provided datasets.

Then I will move on to use CycleGANs to perform face2cartoon and sad2happy transformations on two new datasets. I will conclude my experiments by highlighting the importance of each loss function during the training process.

1. Introduction

Generative adversarial networks aim at performing style transfer from an image to another. This task is usually performed "using a training set of aligned image pairs" [6] which is challenging as paired datasets are very hard to find and very costly to create. The CycleGAN is designed to perform this task without the need of paired images. The cycle idea consists in learning two mappings from a source domain X to a target domain Y and vice versa. (please refer to the CycleGANs section for more details).

2. GANs

Generative Adversarial Networks is an innovative generative model. The idea behind this model is to learn a loss function by training two adversarial neural networks: Generator and Discriminator. The generator is trained to generate data as close from the real distribution as possible, in order to fool the discriminator. The Discriminator is trained to distinguish between the fake generated data and the real ones. Eventually the generator ends up providing authentic real images, the discriminator, then, randomly guesses whether they are fake or real.(Cf figure 1)

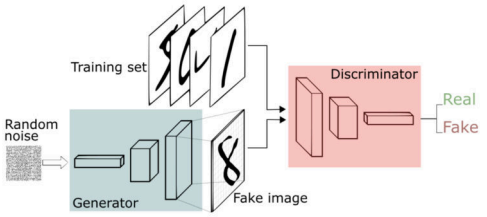


Figure 1: Generative Adversarial Networks: Basic Idea

3. CycleGANs

3.1. Reminder

Transferring characteristics from an image to an other is what CycleGANs are used for. The model learns two mapping functions: $G : X \rightarrow Y$ and $F : Y \rightarrow X$.
X and Y denote the two domain distributions between which we want to perform the style transformation. GAN architechture is used to improve the generated distribution G. In fact, a discriminator D_Y is associated with the domaine Y that works against G. F, in this case, is used to reconstruct the initial image from the generated one. The same mechanism is executed from the domain Y to the domain X.
The authors of the paper introduced a Cycle-consistency loss (Cf figure 2) to ensure that: $F(G(x_i)) = x_i;\ \forall x_i \in X$ and $G(F(y_i)) = y_i;\ \forall y_i \in Y$which adds more constrains to make the model more efficient.

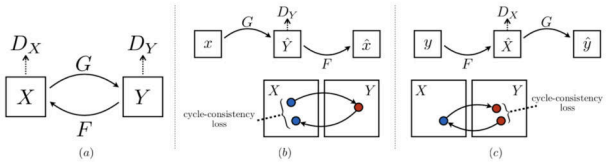


Figure 2: CycleGAN: Cycle Consistency Loss

3.2. Loss functions

The total loss function used to train the model is the weigthed sum of 4 loss functions (Here we suppose the weights are set to one for explanatory reasons):

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) \quad (1)$$
$$+ \mathcal{L}_{GAN}(F, D_X, Y, X) \quad (2)$$
$$+ \mathcal{L}_{cycle}(G, F) + \quad (3)$$
$$+ \mathcal{L}_{Identity}(G, F) \quad (4)$$

The terms (1) and (2) are the GAN losses, which are defined by:

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

In a simplified way, they tell the generator how close from the target distribution the generated images are with the help of the Discriminator.

The cycle consistency loss (3) is defined as follows:

$$\mathcal{L}_{cycle}(G, F) = \mathbb{E}\|G(F(y)) - y\|_1 + \|F(G(x)) - x\|_1$$

For more consistency, the authors added the identity loss to make the generator keep an image from the target distribution unchanged.

$$\mathcal{L}_{Identity}(G, F) = \mathbb{E}\|G(y) - y\|_1 + \|F(x) - x\|_1$$

Figure 3 illustrates the ue of the loss functions

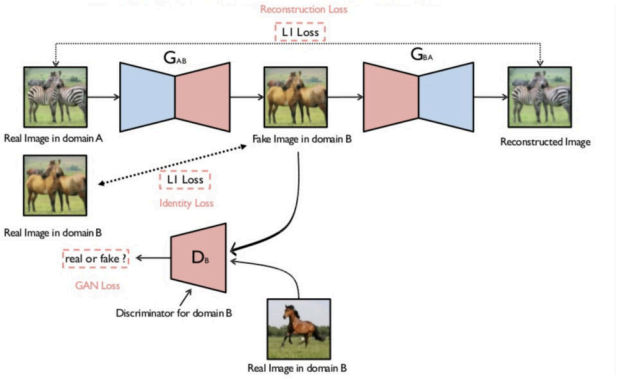


Figure 3: CycleGAN: Overview

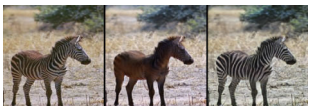
4. Experiments

One of the main challenges with GANs is the huge training time that it takes. That's why I used two GPUs to perform two training tasks in parallel: Tesla V100 and Tesla K80 GPU provided by my school. I chose to use the pytorch implementation of Jun-Yan-Zhu [3]. I run a visdom server in parallel of the training in order to keep track of the model results during its traing. The results are shown in the appendices section.

4.1. Reproducing some results

I trained the provided model using horse2zebra and apple2orange datasets. More results can be found in the appendices section, figure 4 shows one of my best results on horse2zebra dataset.

(a) From horse to zebra



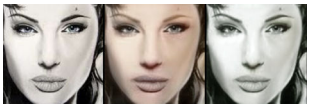
(b) From zebra to horse

Figure 4: From left to right: original, generated, reconstructed image

4.2. Using new datasets

I tested the original code on two different datasets to transform real faces into cartoon faces and to transform sad cartoon faces into happy cartoon faces.

- **face2cartoon:** I used the "IIIT-CFW database for the cartoon faces in the wild", containing real faces and cartoon faces collected for tasks as such Photo2Cartoon retrieval. The dataset can be found in this reference [2]. It took quite a while to start having good results, see figure 5 for some of the best results and please refer to the appendices for more observations.



(a) From cartoon to real face

(b) From real face to cartoon

Figure 5: From left to right: original, generated, reconstructed

- **sad2happy:** I used the annotated dataset created by Facial Expression Research Group Database (FERG-DB) which can be found here [5]. The dataset contains several facial expression images of six characters, I took the sad and the happy images of those characters to perform the emotion transfer task from sad to happy using the cycleGAN.

- Case 1: At first, I used the sad images of the 6 characters as the source domain X and the happy images of all the charaters as the target domain Y. The model ended up to successfully perform the task and perfectly output the opposite emotion of a given image for the given character. 7

-Case 2: I tried to use one character either for the sad or the happy domain. I used sad images of Malcom, Mery and Ray for the source domain X, and I used happy images of Aia, Jules and Bonnie for the target domain Y. The results are less satisfying than the first case.

In fact, for the first case, the Discriminator of the target distribution will lead the generator to the correct distribution about the given character. Whereas, for the second

(a) From sad to happy

(b) From happy2sad

Figure 6: sad2happy case 1 best results. From left to right: original, generated, reconstructed image

case, since the generated images should be as close as possible from the target domain, the model ends up generating a totally new charater which is a combination of the source image and the one of the target images. 7

(a) From sad to happy

(b) From happy to sad

Figure 7: sad2happy case 2 best results. From left to right: original, generated, reconstructed image

4.3. Loss modification

The authors of the paper used three types of loss function: GAN, Cycle and identity loss for both domain directions. What if we keep only the GAN and the cycle loss? what if we assure the cycle consistency in one direction and not the other?
I did some experiments to show how important each loss function is. I trained the modified model on horse2zebra and sad2happy, as described in the case 1 in the previous subsection, datasets. I used the default parameters for most of the experiments, $\lambda_{cycle_A} = \lambda_{cycle_B} = 10$ and $\lambda_{idt} = 0.5$. A few words about the results, I noticed that the cycle loss helps maintain the shape and the edges of the output image and makes it more realistic. The identity loss helps maintain the color distribution of the output image.

For the happy2sad case 2 dataset, I experimented with different values of $\lambda_{idt} \in \{0.5, 10, 50\}$.

Please refer to the appendice section to see the results of the different cases.

5. Quantitative results

In order to have quantitative results, in addition to the qualitative observations, I used the inseption score [1] to evaluate the outcomes of the model.

5.1. Inception Score

As cited in the paper "A Note on the Inception Score' [1], inception score is a widely used evaluation metric for generative models for images. It uses two criteria to measure the performance of GANs: the quality of the generated images and their diversity. The higher the score is, the better the results are. [1]

This score holds its sense by comparing different generative models in terms of output quality. Since I chose to experiment with the cycleGAN model, it doesn't really make sense to have one score without comparing it with something else, so I chose to compare it with the outcome of the cycleGAN with the modified loss. I used this implementation of inception score [4]. The table below summarizes the results I obtained.

5.2. Results

Model	mean	std
standard CycleGAN	2.384	0.435
Absent Cycle and idt loss	1.344	0.334

Table 1: Inception Score

6. Conclusion and further work

Working with generative adversarial networks was a very fruitful experience to me. On the technical level: I practiced to use remote servers, launch parallel trarining tasks in the background, monitor the use of GPUs and manage datasets using linux command lines. On the computer vision knowledge level, GANs is a very interesting and have a very promising future, and the fact that I could observe good results and understand the failure cases helped me go deeper into understanding the paper.

One of the improvements that could be performed, is to improve the loss functions. It could be interesting to experiment with Wasserstein loss and see if we get better results.

GANs are based on the idea of generating images as close as possible from the target distribution, what if we add an other criteria, generate images as far as possible from an other distribution. This additional constrain may help refine the results by adding more specificies to the target domain.

References

[1] S. Barratt and R. Sharma. A note on the inception score, 2018.

[2] IIIT-CFW. Face to cartoon dataset.

[3] Jun-Yan-Zhu. github repository for pytorch-cyclegan-and-pix2pix implementation, 2018.

[4] Thomas Eboli. github repository for gan metrics, 2018.

[5] Facial Expression Research Group Database (FERG-DB). Facial expression dataset.

[6] J.-Y. Zhu, Taesung, P. P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.

Appendices

For all the following images here is the display code

real_A	fake_B	rec_A	idt_A
real_B	fake_A	rec_B	idt_B

Figure 8: results display code. From left to right: real, fake, reconstructed, identity image*

* $\|G_{A \leftarrow B}(img_B) - img_B\|_1 \approx 0$

A. Horse2zebra

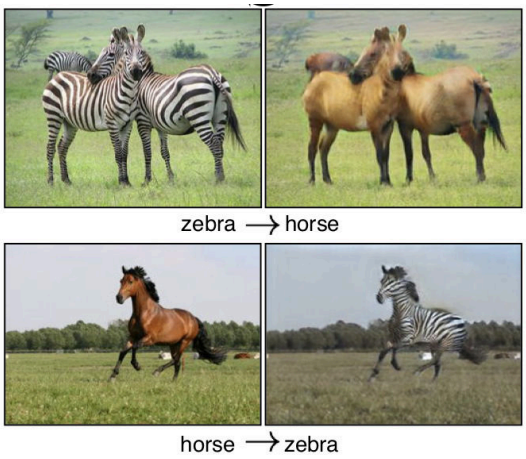


Figure 9: Paper results

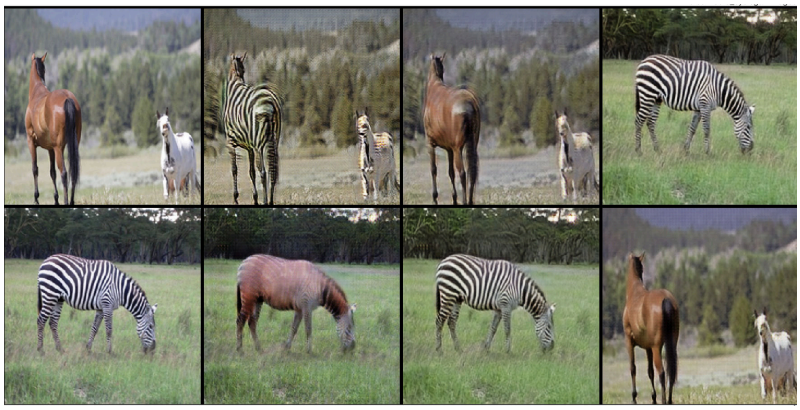


Figure 10: epoch_35

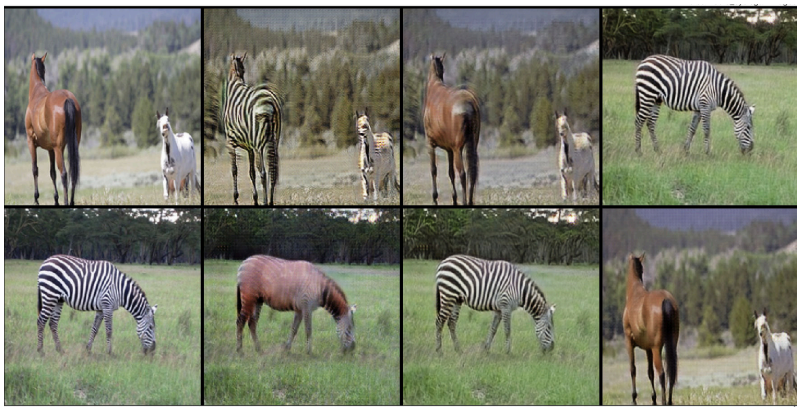


Figure 11: epoch_35

Figure 12: epoch_63

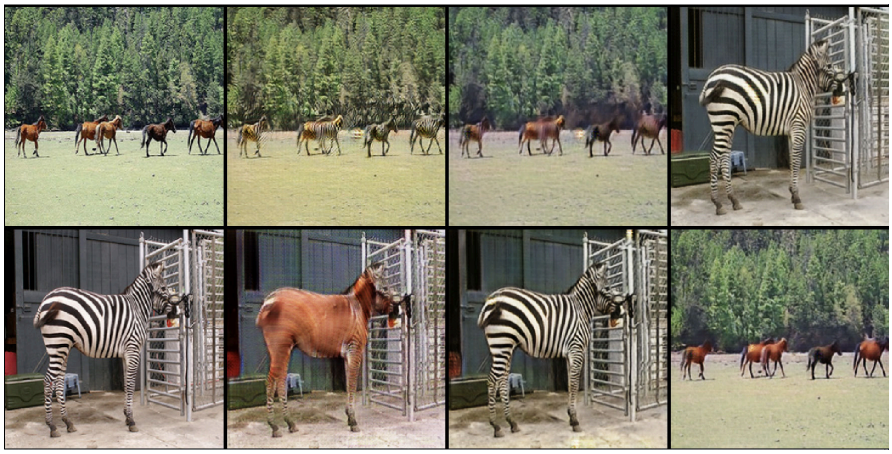
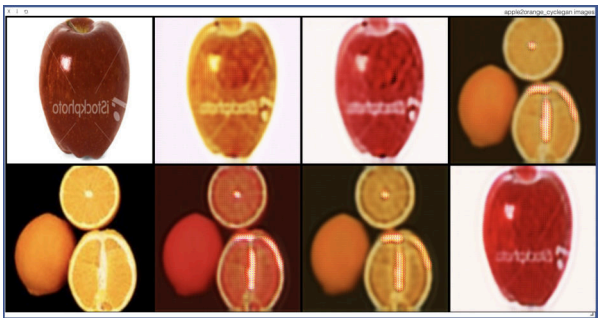


Figure 13: epoch_72

B. Apple2orange



(a) epoch_1

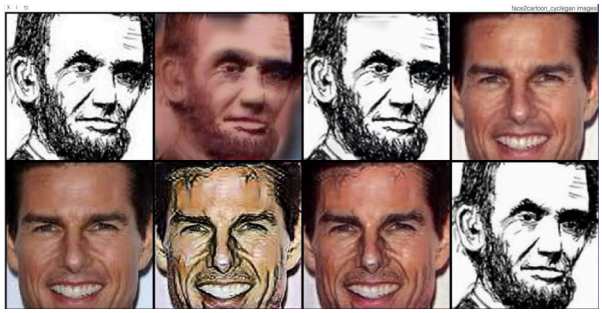
(b) epoch_8

(c) epoch_50

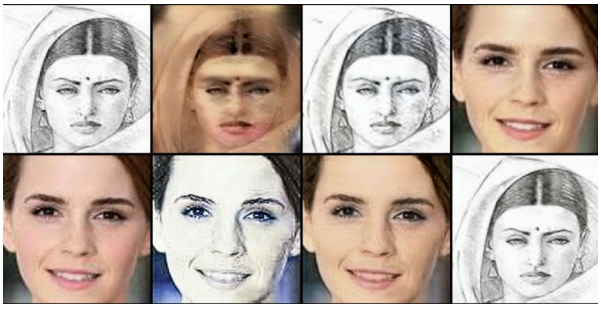
(d) epoch_200

C. Face2cartoon

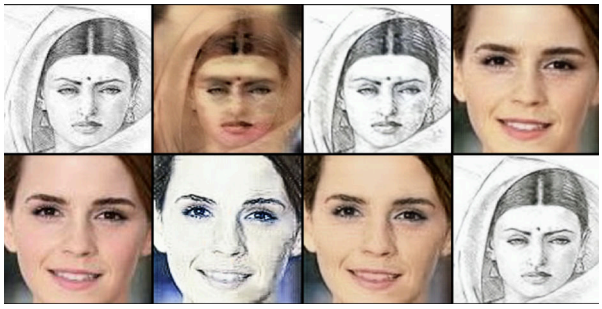
(a) epoch_41



(b) epoch_68



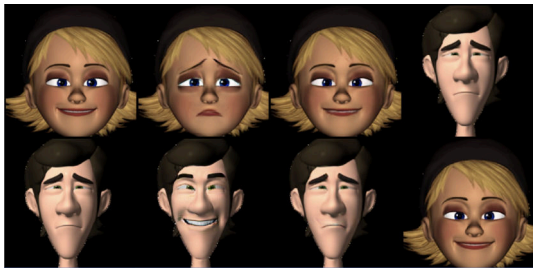
(c) epoch_70



(d) epoch_70

D. sad2happy

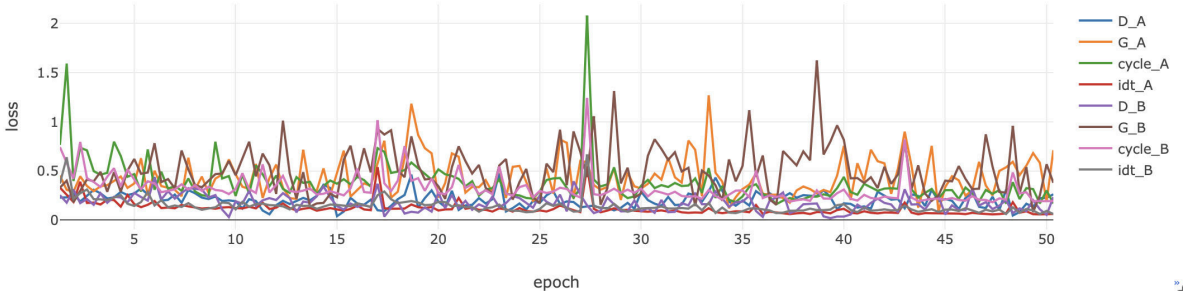
D.1. sad2happy case 1



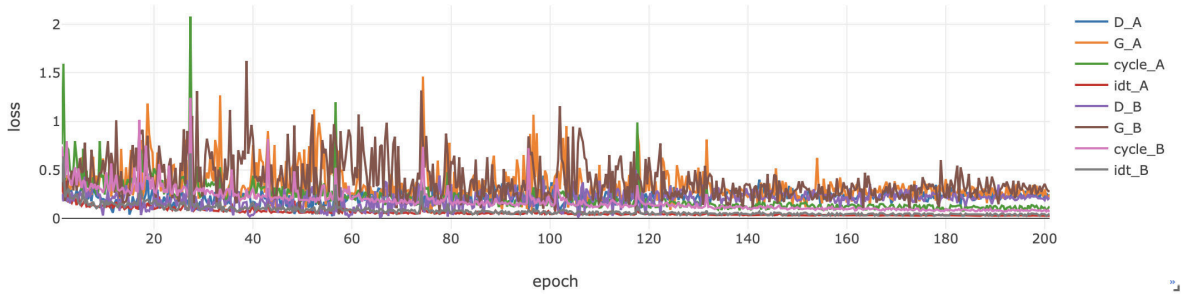
(a) epoch_26

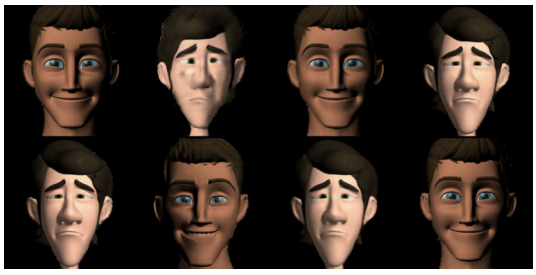
(b) epoch_24

D.2. sad2happy case 2

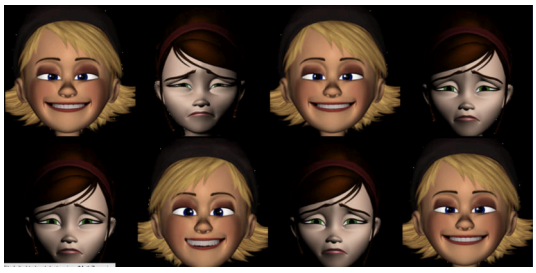


(a) epoch_50





(b) epoch_50



(c) epoch_200

E. Loss graphs over time

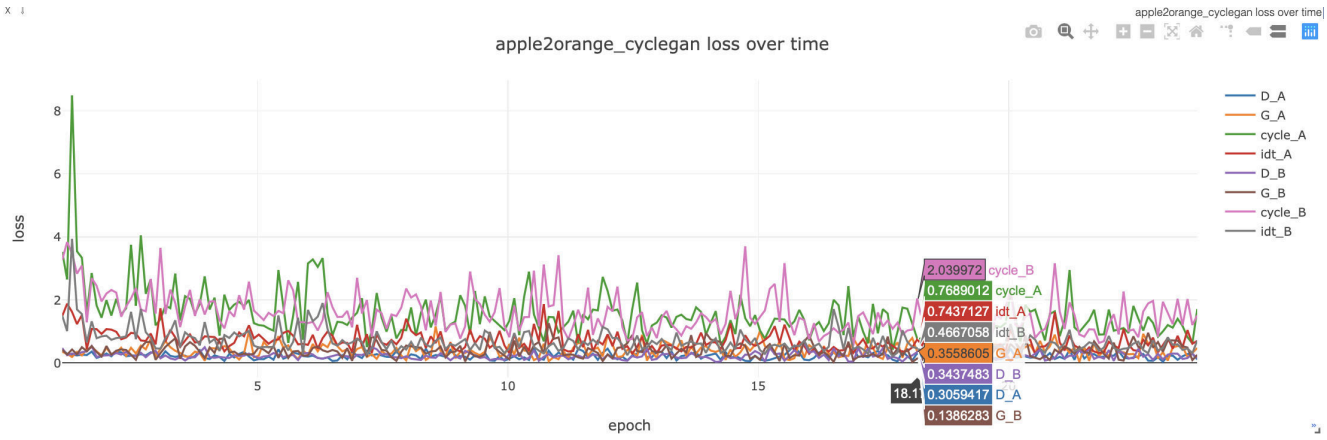


Figure 18: Apple2orange

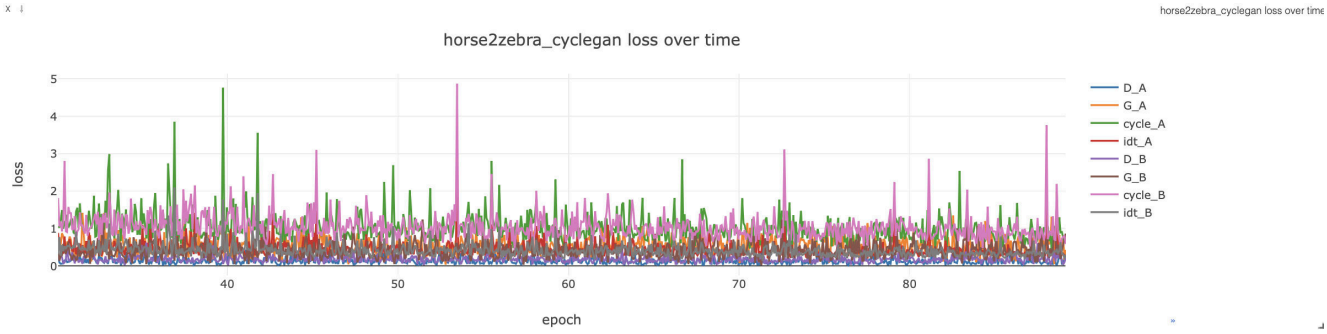


Figure 19: Horse2zebra

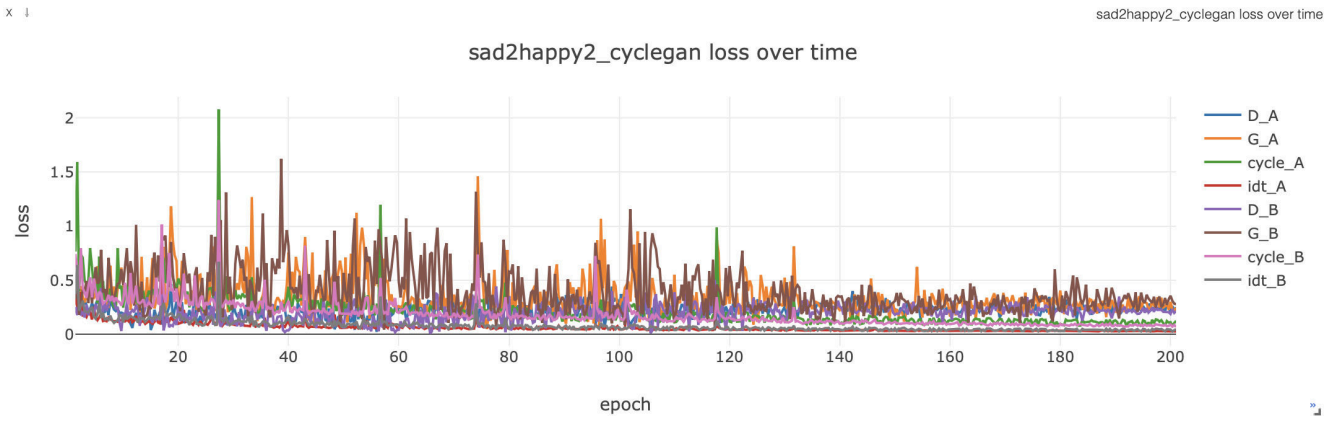


Figure 20: Sad2happy case 1

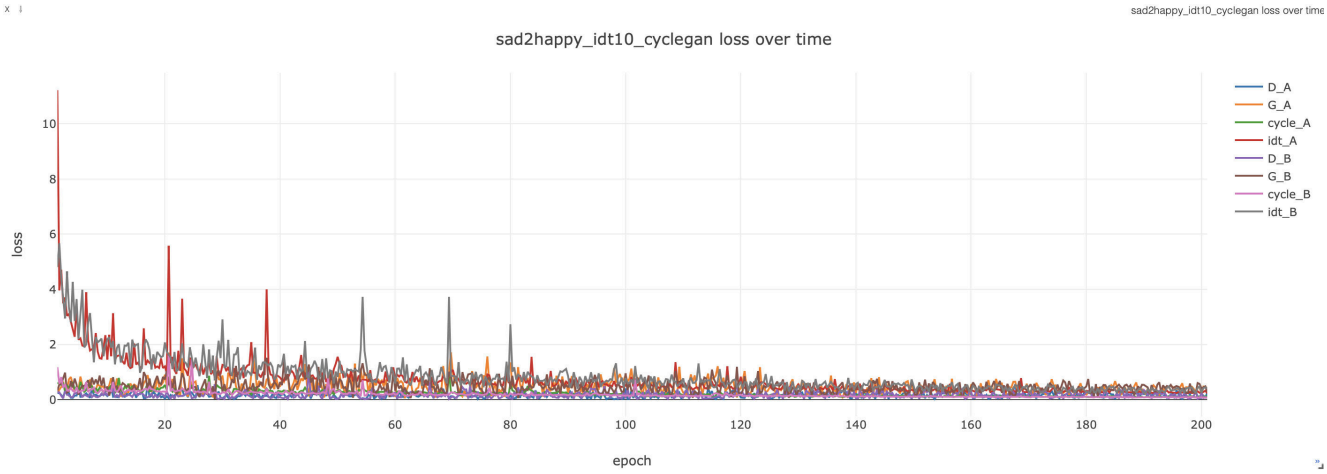


Figure 21: Sad2happy case 2 $\lambda_{idt} = 10$

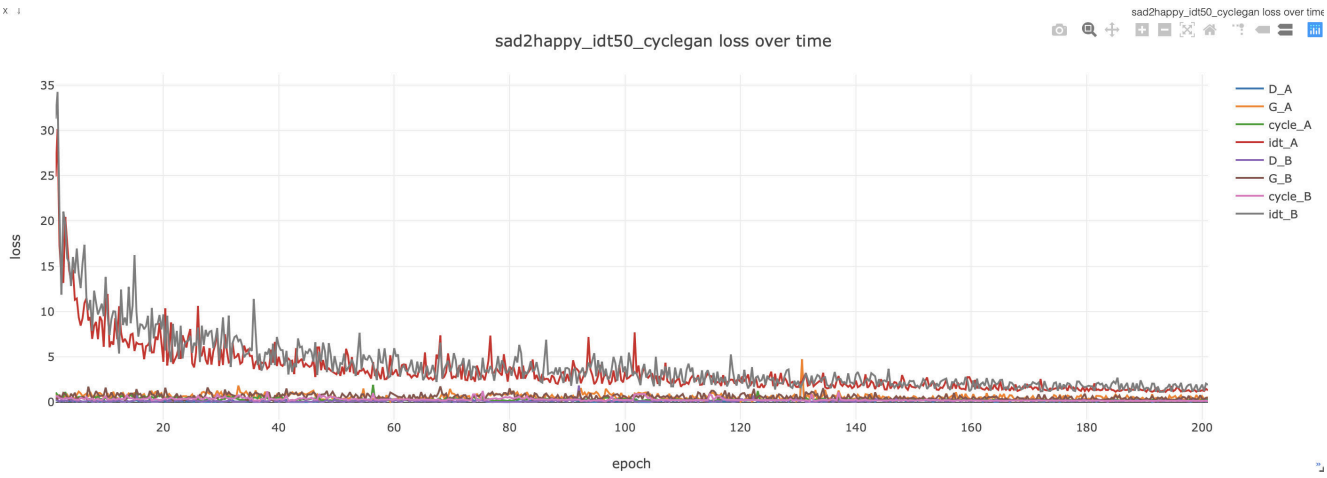
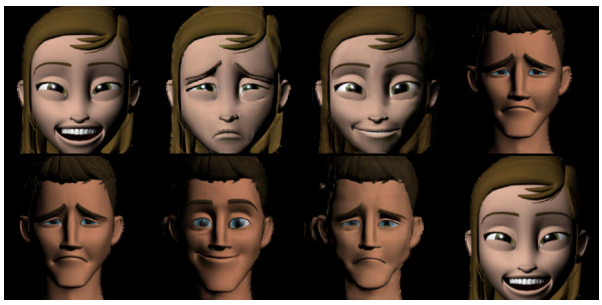


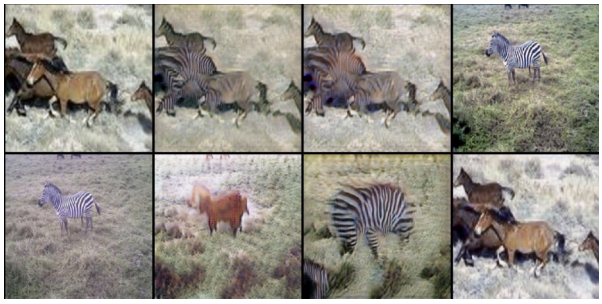
Figure 22: Sad2happy case2 $\lambda_{idt} = 50$

F. Experimenting with the loss functions

F.1. without cycle loss

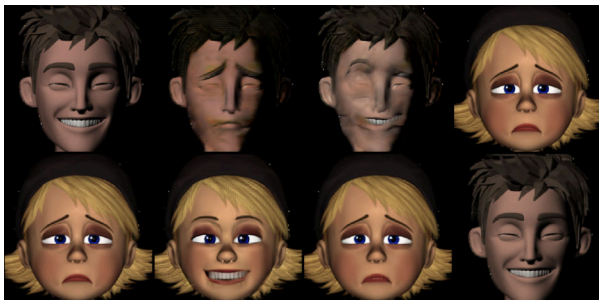


(a) epoch_65

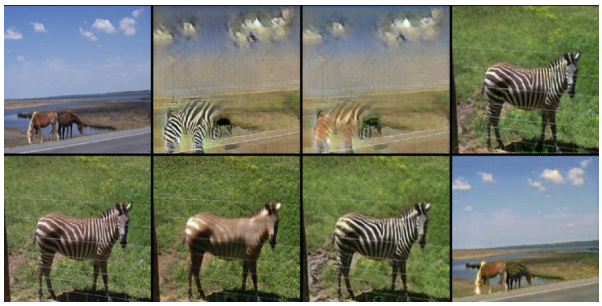


(b) epoch_33

F.2. without cycle_A loss

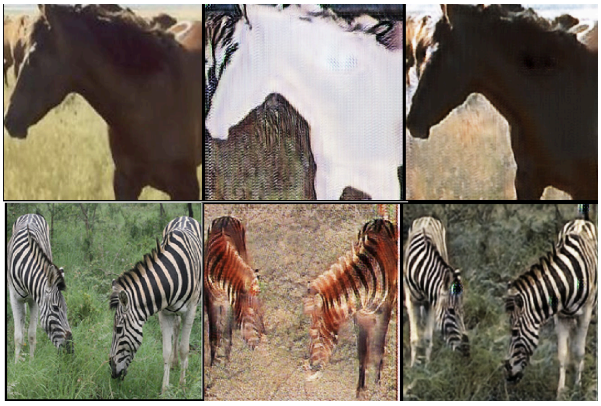


(a) epoch_44

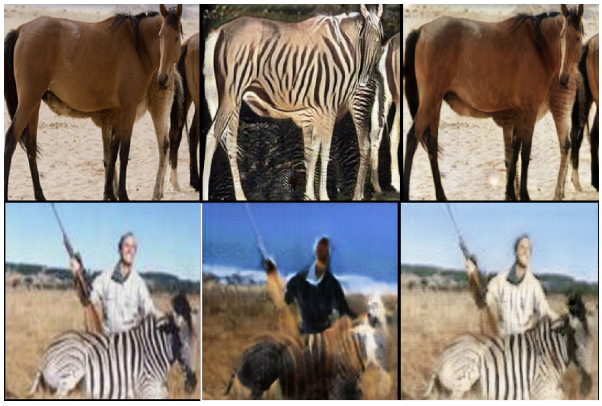


(b) epoch_22

F.3. without identity loss

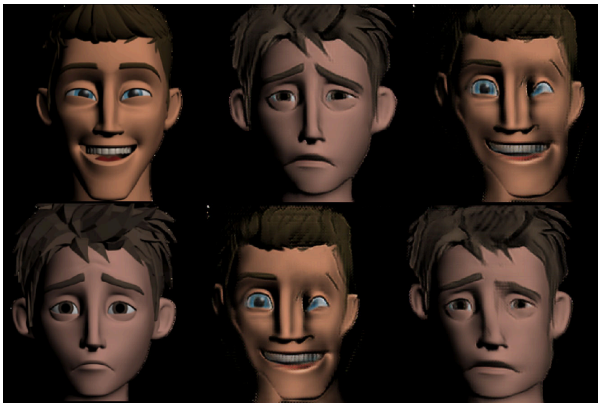


(a) epoch_17



(b) epoch_73

F.4. without Cycle and identity loss



(a) epoch_110

(b) epoch_5