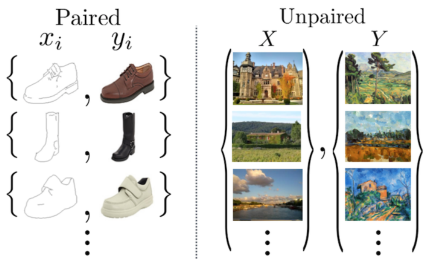
Cycle-Gan for image to image translation

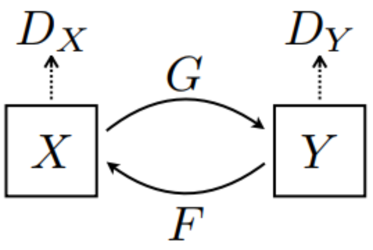
Jacopo Dapueto
s4345255@studenti.unige.it

Abstract

Image-to-image translation is the problem of leaning a mapping between an input image (belonging to the domain X) and an output one (belonging to the domain Y) using a training set of paired images. However in many tasks the paired images are not available or available in a limited number. For this reasons a method has been developed which used two conditional Generative and Adversarial architectures: the first Gan learns a map $G : X \rightarrow Y$ and the second one learns the map $F : Y \rightarrow X$, then the mapping funtions are coupled with a cycle consistency so that $F(G(X)) \approx X$. I tried to implement such image-to-image translation with different architectures for both generators and discriminators, and using different datasets.



(a) Figure 1. Paired and Unpaired datasets.



(b) Figure 2. The model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and the corresponding discriminators D_x and D_y. D_y make G to translate X into outputs indistinguishable from domain Y, and vice versa for D_x and F.

1 Introduction

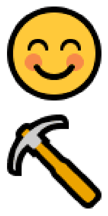
1.1 Paired and Unpaired image

Paired data (see Figure 1, left) are sample $\{x_i, y_i\}_{i=1}^{N}$, where the relationship between x_i and y_i is well known. In the other hand *unpaired* data (see Figure 1, right) are two sets $\{x_i \in X, x = 1...N\}$ and $\{y_i \in Y, y = 1...N\}$ where the relationship between x_i and y_i is not provided. For the following experimets we assume there is some kind of underlying relationship between the two domains.

1.2 Datasets

I used two unpaired datasets for my experiments:

- The first 2 datasets are the MNIST dataset of handwritten digits and the Fashion-MNIST which is a dataset of Zalando's article.The method should transfer an image from the *Digit* domain to *Fashion*, and vice versa.
- The other two datasets are made up of rgb images [4] of Apple and Windows's emojis: The method should transfer the style from one domain to the other. Notice that there is no a 1 to 1 matching between the emojis from one domain and the another one, in fact the windows set is smaller than the Apple one. The images from Windows are more stylized and all of them are surrounded with a thick black line, instead the ones from the other domain are more detailed and shaded.


(a) Windows

(b) Apple

1.3 Architecture

The method is based on 2 generative adversarial networks (Figure 2): The generators take as input an image and produces as output another image of the same size, the discriminators take as input the result of the corresponding generator and learn if such image is taken from the original dataset or it is fake.

We need to appy an adversarial loss to both the mapping functions G and F:

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_y[logD(y)] + \mathbb{E}_x[1 - logD(G(x))]$$
$$\mathcal{L}_{GAN}(F, D_X, X, Y) = \mathbb{E}_y[logD(x)] + \mathbb{E}_x[1 - logD(F(y))]$$

However the adversarial losses alone cannot garantee that the learn function maps the input x_i to the desired output y_i, so must be introduced a cycle-consistency loss which should garantee the ability of the network to go back to the original image.

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

Such loss can be define as:

$$\mathcal{L}_{cyc}(G, F, X, Y) = \mathbb{E}_x[\|F(G(x)) - x\|_1] + \mathbb{E}_y[\|G(F(y)) - y\|_1]$$

The L_1 distance is used because is the one that best reconstruct the images, since since the L_2 tends to make the images blurred.

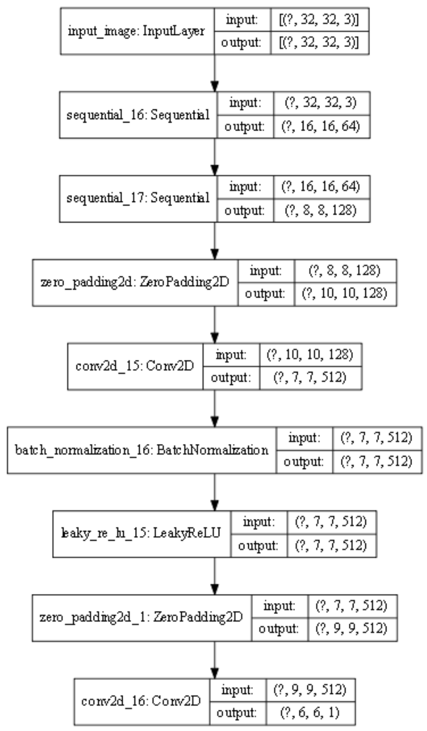
In order to further reduce the space of possible mapping functions can be introduce the *Identity loss*: this is based on the idea that if the generator G is responsible for translating an image X to an image Y, then if the generator G is fed with an image Y then it should yeld something close to the image Y. So it can be defined the loss function:

$$\mathcal{L}_{id}(G, F, X, Y) = \|G(Y) - Y\|_1 + \|F(X) - X\|_1$$

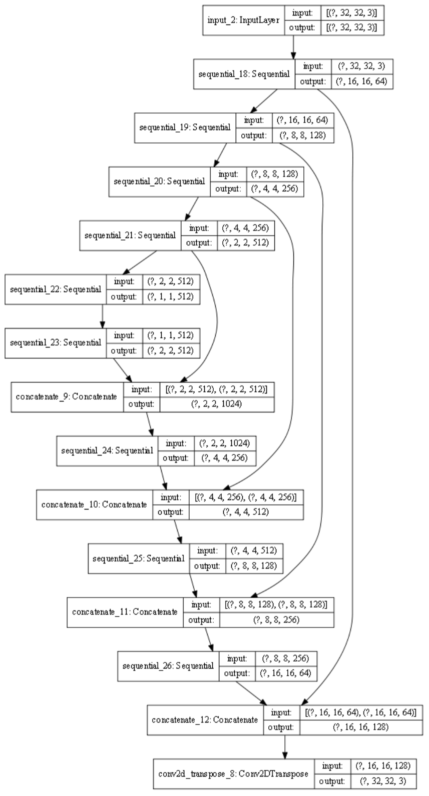
At the end the full objective function to solve is the following:

$$G^*, F^* = \min_{G,F} \max_{D_x, D_y} \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, X, Y) + \lambda\mathcal{L}_{cyc}(G, F, X, Y) + \mathcal{L}_{id}(G, F, X, Y)$$

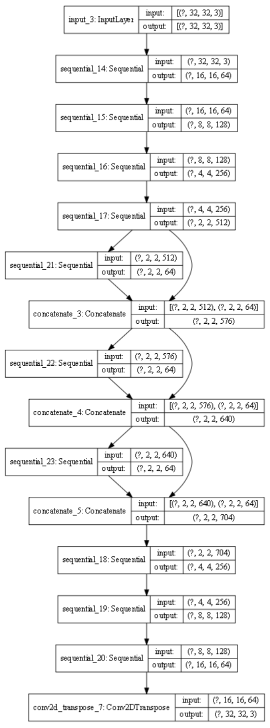
The λ control the importance of the cycle consistency constraint.


input_image: InputLayer | input: | [(?, 32, 32, 3)] | output: | [(?, 32, 32, 3)]
sequential_16: Sequential | input: | (?, 32, 32, 3) | output: | (?, 16, 16, 64)
sequential_17: Sequential | input: | (?, 16, 16, 64) | output: | (?, 8, 8, 128)
zero_padding2d: ZeroPadding2D | input: | (?, 8, 8, 128) | output: | (?, 10, 10, 128)
conv2d_15: Conv2D | input: | (?, 10, 10, 128) | output: | (?, 7, 7, 512)
batch_normalization_16: BatchNormalization | input: | (?, 7, 7, 512) | output: | (?, 7, 7, 512)
leaky_re_lu_15: LeakyReLU | input: | (?, 7, 7, 512) | output: | (?, 7, 7, 512)
zero_padding2d_1: ZeroPadding2D | input: | (?, 7, 7, 512) | output: | (?, 9, 9, 512)
conv2d_16: Conv2D | input: | (?, 9, 9, 512) | output: | (?, 6, 6, 1)

(a) Figure 3. PatchGan architecture. Each *Sequencial* node is made up of a Convolutional layer + Batch normalization + Relu


input_2: InputLayer | input: | [(?, 32, 32, 3)] | output: | [(?, 32, 32, 3)]
sequential_18: Sequential | input: | (?, 32, 32, 3) | output: | (?, 16, 16, 64)
sequential_19: Sequential | input: | (?, 16, 16, 64) | output: | (?, 8, 8, 128)
sequential_20: Sequential | input: | (?, 8, 8, 128) | output: | (?, 4, 4, 256)
sequential_21: Sequential | input: | (?, 4, 4, 256) | output: | (?, 2, 2, 512)
sequential_22: Sequential | input: | (?, 2, 2, 512) | output: | (?, 1, 1, 512)
sequential_23: Sequential | input: | (?, 1, 1, 512) | output: | (?, 2, 2, 512)
concatenate_9: Concatenate | input: | [(?, 2, 2, 512), (?, 2, 2, 512)] | output: | (?, 2, 2, 1024)
sequential_24: Sequential | input: | (?, 2, 2, 1024) | output: | (?, 4, 4, 256)
concatenate_10: Concatenate | input: | [(?, 4, 4, 256), (?, 4, 4, 256)] | output: | (?, 4, 4, 512)
sequential_25: Sequential | input: | (?, 4, 4, 512) | output: | (?, 8, 8, 128)
concatenate_11: Concatenate | input: | [(?, 8, 8, 128), (?, 8, 8, 128)] | output: | (?, 8, 8, 256)
sequential_26: Sequential | input: | (?, 8, 8, 256) | output: | (?, 16, 16, 64)
concatenate_12: Concatenate | input: | [(?, 16, 16, 64), (?, 16, 16, 64)] | output: | (?, 16, 16, 128)
conv2d_transpose_8: Conv2DTranspose | input: | (?, 16, 16, 128) | output: | (?, 32, 32, 3)

(b) Figure 4. U-Net architecture. Each *Sequencial* node is made up of a Convolutional (or Conv2DTranspose) layer + Batch normalization + Relu


input_3: InputLayer | input: | [(?, 32, 32, 3)] | output: | [(?, 32, 32, 3)]
sequential_14: Sequential | input: | (?, 32, 32, 3) | output: | (?, 16, 16, 64)
sequential_15: Sequential | input: | (?, 16, 16, 64) | output: | (?, 8, 8, 128)
sequential_16: Sequential | input: | (?, 8, 8, 128) | output: | (?, 4, 4, 256)
sequential_17: Sequential | input: | (?, 4, 4, 256) | output: | (?, 2, 2, 512)
sequential_21: Sequential | input: | (?, 2, 2, 512) | output: | (?, 2, 2, 64)
concatenate_3: Concatenate | input: | [(?, 2, 2, 512), (?, 2, 2, 64)] | output: | (?, 2, 2, 576)
sequential_22: Sequential | input: | (?, 2, 2, 576) | output: | (?, 2, 2, 64)
concatenate_4: Concatenate | input: | [(?, 2, 2, 576), (?, 2, 2, 64)] | output: | (?, 2, 2, 640)
sequential_23: Sequential | input: | (?, 2, 2, 640) | output: | (?, 2, 2, 64)
concatenate_5: Concatenate | input: | [(?, 2, 2, 640), (?, 2, 2, 64)] | output: | (?, 2, 2, 704)
sequential_18: Sequential | input: | (?, 2, 2, 704) | output: | (?, 4, 4, 256)
sequential_19: Sequential | input: | (?, 4, 4, 256) | output: | (?, 8, 8, 128)
sequential_20: Sequential | input: | (?, 8, 8, 128) | output: | (?, 16, 16, 64)
conv2d_transpose_7: Conv2DTranspose | input: | (?, 16, 16, 64) | output: | (?, 32, 32, 3)

(c) Figure 5. Encoder-Decoder + Residual blocks. Each *Sequencial* node is made up of a Convolutional (or Conv2DTranspose) layer + Batch normalization + Relu

1.4 PatchGan

In some of the following experiments I used a PatchGan [1] discriminator (Fig. 3): the classical discriminators learn if an entire image is real or fake, instead this architecture aims to classify individual *NxN* image patches as real or generated. First such architecture has the advantage of having few parameters to be estimated with respect to the classical discriminators and then it allows the generator to focus on such patches of the image that required some changes, because there could be the case where only a small part of the image must change and what remain must be untouched.

1.5 U-Net

The U-Net [2] is a generative architecture (Fig. 4) similar to a standard Encoder-Decoder structure, but in addiction each layer in the Encoder is connected even with a layer in the Decoder and each connection concatenates all the channels in the econder layer to those present in the decoder layer. This kind of connections are useful under the assumption that the input and the output images share an undelying structure and so the features extracted from the input image can help the construction of the output.

1.6 Encoder-Decoder + Residual blocks

Another kind of generator (Fig. 5) I'm going to use is similar to one proposed in the paper "Image-to-Image Translation with Conditional Adversarial Networks": it takes as input an image and they implement three stages of computation: the first stages *encode* the input image using a series of convolutional layers; the second stage is a set of *residual blocks* to *transform* features extracted in the previous stage; the third and last stage *decode* the feature of the previous stage and it produces an output of the same size of the input. The residual block used in the transformation stage consists of a convolutional layer plus a batch normalization and a relu activation function, where the input is added to the output of the convolution. This is done so that the characteristics of the output image (e.g., the shapes of objects) don't change too much from the input.

1.7 Preprocessing

Each dataset is first divided into the training set and the test set which is only used to evaluate the performances of the model. Then as suggested in the paper " Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" each image in the training dataset is:

- Resize to bigger height and width, I decide to inflate the image of about the 20% of the original size;
- The image is randomly cropped to the target size, more and less half of the images are cropped;
- The image is randomly flipped horizontally (i.e left to right).

This is done to avoid overfitting that can easely affect these kind of models, in particular without these operations the discriminators can learn very fast if an image is real or generated and the generator stucks on that.

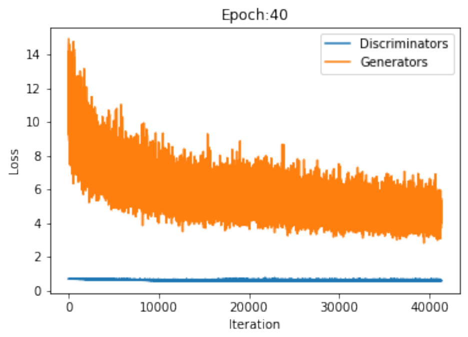
2 Experiments on Emojis

In the following experiments the generative model is trained to transfer the images from the dataset *Apple* to the dataset *Windows*. The model is trained using as $\mathcal{L}_{GAN}$ a *Binary cross entropy Loss* and with an Adam optimizer, setting the λ equal to 10 and using 0.0002 as leaning rate for both the generators and disciminators optimizer, as suggested in the paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". I always trained the model with 40 epochs.

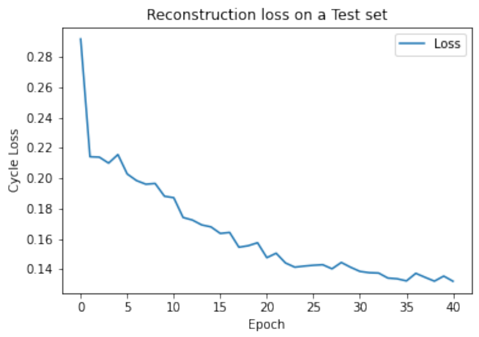
2.1 First experiment

For this experiment the Discriminator is a simple architecture that takes as input an image and with series of convolutional layers it fires if the image is fake or real, instead as generators I used the one at the figure 5.

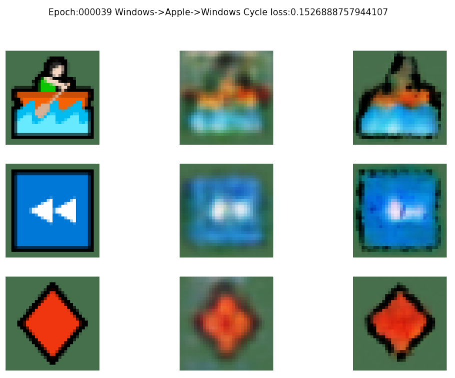
I trained the model with a batch size of 32 to make the training session fast.



(a) Figure 6. Discriminatiors and Generators losses.



(b) Figure 7. Cycle loss on a test set along the epoch.


Epoch:000039 Windows->Apple->Windows Cycle loss:0.1526888757944107

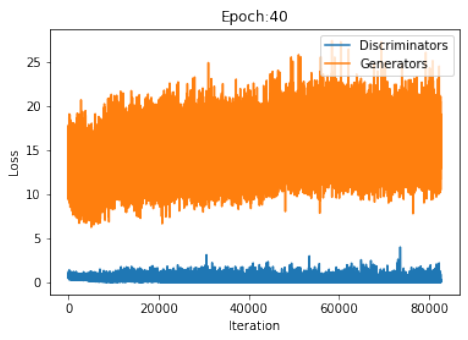
(c) Figure 8. Results at the 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Windows -> Apple -> Windows

the Fig. 8 show 3 images from the Windows dataset mapped into the Apple domain and then mapped back into the original domain: the column in the middle shows the transformed images and they are particular smoothed and blurred and not even well shaped as the original, however they don't have or only slightly the black line around the object (in particular the first and second image) which is not a characteristic of the Apple domain but I don't know if it's learning how to go from one distribution to the another or the results are like so because they are very low quality . Instead the reconstruction (third column) is less smoothed but without some level of detail, the mean $\mathcal{L}_1$ loss between the original images and recontructed is 0.15 .

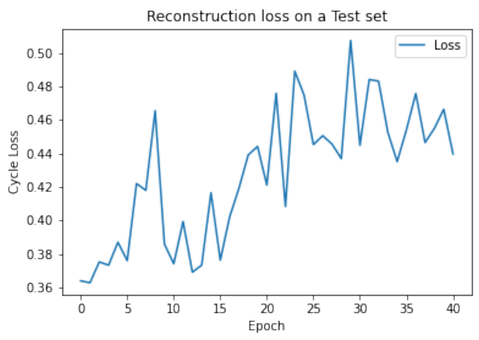
From the Fig. 6 and the Fig. 7 it seems the reconstruction of the images leads the leaning of the model until the 40th epoch because the generators loss tends to decreases and the discriminators loss is almost flat and this means that the discriminators and the generators are not yet behaving as adversarial , meanwhile the reconstruction loss shown in the figure 7 decreases more and more slowly. Probably with more epochs the model could achieve better results in terms of the reconstruction and mapping functions.

2.2 Second experiment

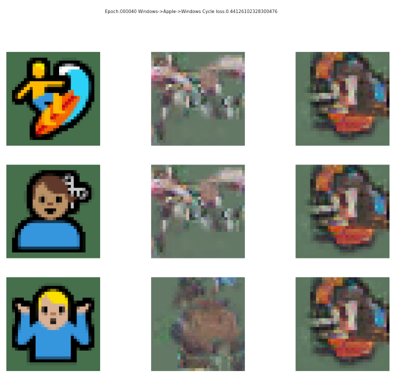
In this case I used the same kind of generator and a PatchGan discriminator (see Fig. 3) described before, in order to allow the generator to focus on the patches of the image that need to change. The model is trained with a batch size of 1 as it is suggested in the paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", this makes the training more slowlier but these choices should allow to achieve better results.



(a) Figure 9. Discriminatiors and Generators losses.



(b) Figure 10. Cycle loss on a test set along the epoch.



Epoch:000040 Windows->Apple->Windows Cycle loss:0.44126102328300476

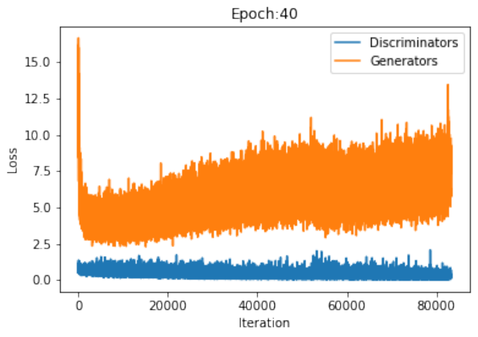
(c) Figure 11. Results at the 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Windows -> Apple -> Windows

If we look at the figure 11 can be seen this kind of model doesn't work at all: the model after 40 epochs doesn't produce something meaningful and even original image is not reconstructed in any part. The Fig. 9 shows the Generators loss which only tends to increases without minimizing the cycle consistency, as can be seen from the Fig. 10 which shows the reconstruction loss that at the first epoch is relatively high (almost 0.36) and only tends to encrease. I don't think training with further epochs could help to improve the results since the model has produced nonsense images from the first epoch to the last one.

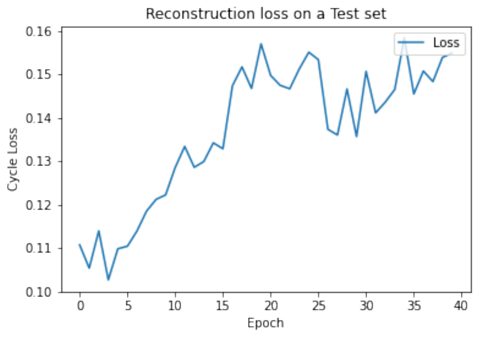
2.3 Third experiment

Since the previous experiment doesn't work, here I trained the model with the U-Net generator (see Fig. 4) and the PatchGan discriminator with a batch size of 1, this kind of choice should provide the best results so far .

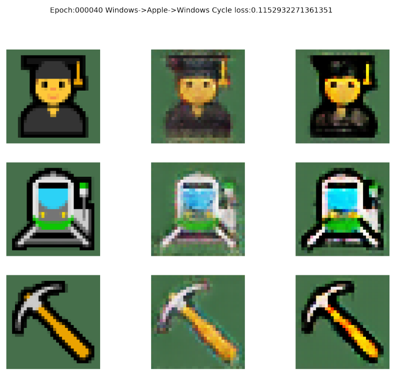
The results shown in the figure 14 are the best obtained so far: the second column respresents the transition between the Windows domain to the Apple domain and the images generated are more detailed with respect to those in the previous experiments, they don't have the thick and black line present in the original images but preserving the meaningful parts of the objects. They present artifacts mainly along the borders because those are the image patches involved in the transformation. In addiction the reconstruction achieves good performances with 0.11 of $\mathcal{L}_1$ loss, even if the small details are not reconstructed well and such detail are not present (or only slightly) even in the second column.



(a) Figure 12. Discriminatiors and Generators losses.



(b) Figure 13. Cycle loss on a test set along the epoch.



Epoch:000040 Windows->Apple->Windows Cycle loss:0.1152932271361351

(c) Figure 14. Results after 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Windows -> Apple -> Windows

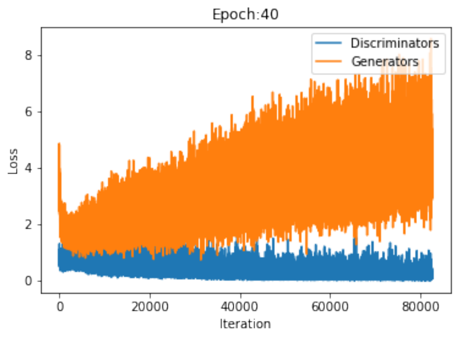
The generators loss (Fig. 12) goes immediately down because the minimization of the reconstruction loss leads completely the first iterations of the training procedure, in fact looking at the figure 13 the reconstruction loss within the first epochs is small (about 0.11) and as the training goes on it increases but still remaining in a small range from 0.10 and 0.16. The increase of generators losses and so the the reconstruction loss on the test set could be the symtom that the model is learning the translation functions, and it seems that in order to learn how to map an image from a domain to another it should first learn how to reconstruct the images to preserve as much as possible the useful informations between the two datasets.

2.4 Without cycle consistency

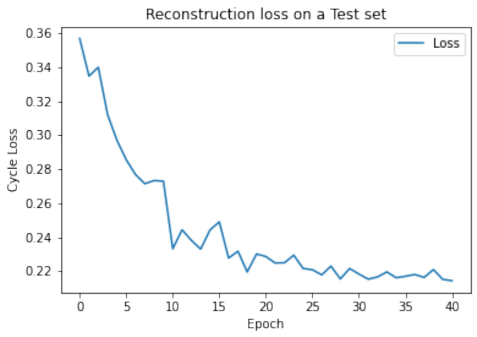
As seen in the previous experiments the cycle consistency loss seems to play a very important role in how the model learns the image-to-image translation: what if the λ is set to zero? I trained the same architecture without computing the cycle consistency loss and a batch size of 1.

The behaviors of the generators and discriminators losses are similar to those in the previous experiment (Fig. 15): the generators loss first decreases and then encreases while the discriminators loss seems to decreases very slowly. Instead the cycle loss on the test set has a completely different behavior (Fig. 16), it starts high (about 0.36) and it descrease more and more slowly.

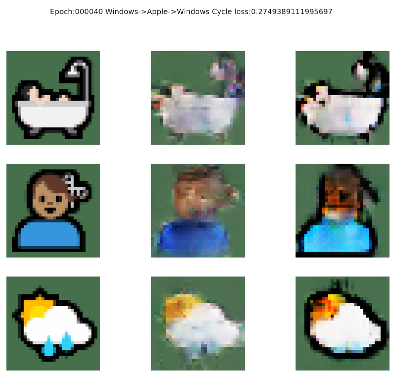
Indeed if we look at the figure 17 and 18 some kind of the structures seem to be more and less reconstructed even if the cycle consistency is not present in the system, in particular the reconstruction into the Apple domain reach the better perfomances with 0.15 of averaged cycle loss and the reconstruction into the Windows domain gets 0.27 of loss. The generator $G_{WindowsToApple}$ works better at reconstructing the images and that's the reason why the generator $G_{AppleToWindows}$ seems



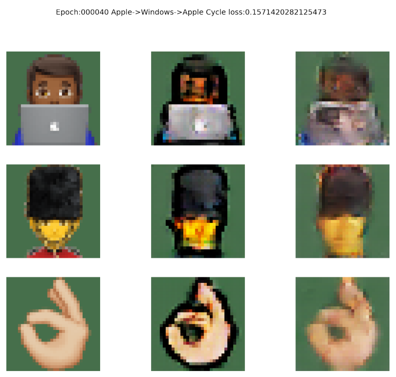
(a) Figure 15. Discriminatiors and Generators losses.



(b) Figure 16. Cycle loss on a test set along the epoch.

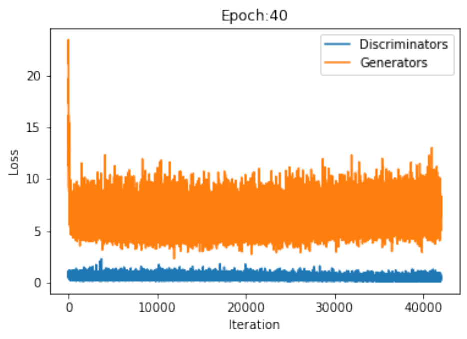

Epoch:000040 Windows->Apple->Windows Cycle loss:0.2749389111995697

(c) Figure 17. Results after 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Windows -> Apple -> Windows

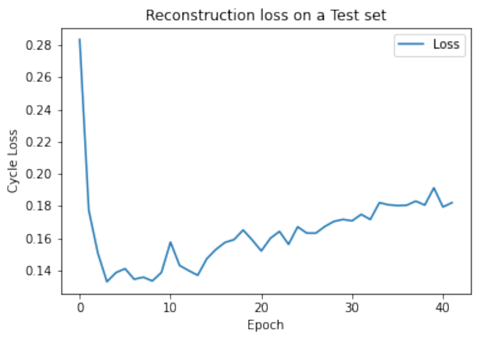

Epoch:000040 Apple->Windows->Apple Cycle loss:0.1571420282125473

(d) Figure 18. Results after 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Apple -> Windows -> Apple the best to translate an image from one domain to the other, since the generated images are less faded and with less noise than the images generated with the other generator.

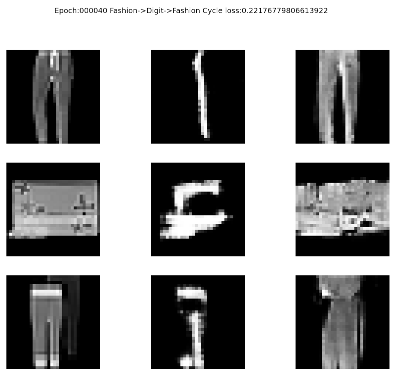
From such results we can get that to learn an image-to-image translation function the model should learn also to reconstruct back the meaningful and commom structures of the image, on this purpose the cycle consistency loss can help the model but notice that It doesn't need to learn to reconstruct perfectly the image so the λ shouldn't be very high.



(a) Figure 19. Discriminatiors and Generators losses.

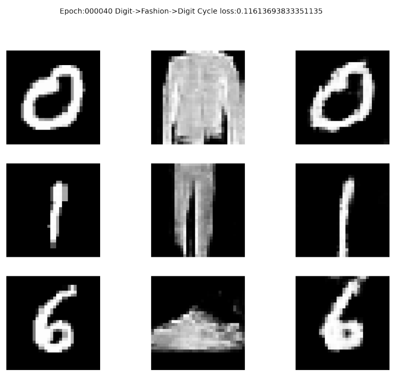


(b) Figure 20. Cycle loss on a test set along the epoch.



Epoch:000040 Fashion->Digit->Fashion Cycle loss:0.22176779806613922

(c) Figure 21. Results after 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Fashion -> Digit -> Fashion



Epoch:000040 Digit->Fashion->Digit Cycle loss:0.11613693833351135

(d) Figure 22. Results after 40 epoch: the first colum: images X, second: $G(X)$, third: $F(G(X))$, Digit -> Fashion -> Digit

3 Experiments on MNIST datasets

In the following experiment the model will be trained to transfer the images from the *Digit* dataset to the *Fashion* one. The translation of images from the Apple to Windows (and vice versa) seems to work even because the two datasets have some kind of structures that are somehow retaled, by contrast the Digit and the Fashion datasets are completely different and are not related at all. This experiment aims to find out if the method presentend so far can work with this kind of translation. I used the U-Net architecture for the generator and the PatchGan discriminator as in the previous experiment. The model is trained using as $\mathcal{L}_{GAN}$ a *Binary cross entropy Loss* with an Adam optimizer, using 0.0002 as leaning rate for both the generators and disciminators optimizer with the cycle consistency setting the λ equal to 10.

The images show the results after 40 epochs with a batch size of 1 image.

As the previuos experiments the generators loss (Fig. 19) first decreases and then encreases very slowly meanwhile it seems that the disciminators learn very fast how to recognise fake and real images, since the disciminators loss is very close to zero and as the epochs go on it further decreases. The Fig. 20 shows the $\mathcal{L}_1$ loss on the test set, at the first epochs it decreases very fast and then it slowly encreases as the model learn the translations. Comparing the Fig. 20 and 13, refering to the experiment under the same conditions (architecture, cycle consistency, batch size, etc.), the model in this experiment needs some epochs to minimize the loss meanwhile in the third experiment (Fig. 13) the loss at the first epoch is already at the minimum and only encreases with the epochs, even though the two plots have differents scales it can be seen that the two losses encrease more and less with the same speed. This could mean that the two models have worked in a similar way.

The figure 21 and 22 shows the results after 40 epochs: the reconstruction of the images into the Digit domain seems to work better since it has 0.11 of reconstruction loss, and also the translation from Digit to Fashion seems to work better. From the translated images can be recognise a shoe, a pair of pants and a jacket, instead the images translated in the Digit domain are less rognizable: in fact the reconstruction loss in the Fashion domain is less accurate, having on average 0.22 of $\mathcal{L}_1$ loss.

Although the two generators are trained together with the same steps, it seems that a generator is always better at translating the image and the other one is better at reconstructing back the original image: this may depends on the characteristics of one of the domains which are easier to learn. This can be observed also in the previous experiments.

All in all this experiment shows how this kind of method can even work with datasets without anything in common. It only needs more epochs to improve the results for both the domains.

4 What remains

In the experiments I tried to provided numerical evaluations about the performances, in particular with the plots of the generators and discriminators losses and the plots of reconstructions as the number of iterations encreases. But I didn't provide any number about the generated samples, originally I tried to implement a way to evalute the generated images. There are not a lot of studies about such evaluation and the kind of metrics may differ depending on what you want evaluate, in particular the aim of the cycleGan is to generate samples with the same style of a particular domain and so the generated should have similar features to the images belonging to the training set. This aspect can be evaluated with the **Fréchet Inception Distance (FID)**[6]: It uses the pretrained Inception network without the final layer that fires the output label, but the output is provided by the latest features extractor layer.

Let $F_r = \varphi(R)$, $F_g = \varphi(G)$ be two groups of features vectors extracted from the real and generated image sets and $F_r \sim N(\mu_r, \Sigma_r)$ $F_g \sim N(\mu_g, \Sigma_g)$ be two Gaussians.

The distance is computed like so:

$$FID = \|\mu_r - \mu_g\|^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

In fact the **FID** computes the distance between the distributions of the generated and real images. such distance must be minimize to have samples with the same style of the real images.

I'm not able to run such measure because my limited amount of computational resources doesn't let me load the Inception network. However my idea was to compute the distance between the generated images and training set of the relative domain for each epoch: I expect the distance to decrease as the epochs encreases but still remaining relatively high because of the low quality of the images.

References

[1] Jun-Yan Zhu & Taesung Park & Phillip Isola &Alexei A. Efros (2018) *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* Berkeley AI Research (BAIR) laboratory, UC Berkeley.

[2] Olaf Ronneberger & Philipp Fischer & Thomas Brox (2015) *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Emoji dataset: https://unicodey.com/emoji-data/

[5] Jun-Yan Zhu & Tinghui Zhou & Phillip Isola &Alexei A. Efros (2018) *Image-to-Image Translation with Conditional Adversarial Networks* Berkeley AI Research (BAIR) laboratory, UC Berkeley.

[6] Shuyue Guan & Murray Loew A Novel Measure to Evaluate Generative Adversarial Networks Based on Direct Analysis of Generated Images Department of Biomedical Engineering, the George Washington University, Washington DC, USA