# Deep Learning - Final project

Caroline Boudier, Florian Schirrer

ENSAE, April 2021

In this report we present the results of our work around Generative Adversarial Networks as part of the Deep Learning course. The first section will consist in a short bibliographical review where two articles will be summarized : the DC-GAN[1] and the Wasserstein GAN[2]. Afterwards we will study the cycleGAN architecture and provide an implementation on the MNIST/USPS dataset.

## Table of Contents



[1] A Radford, L Metz, S Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks arXiv:1511.06434, 2015

[2] M Arjovsky, S Chintala, L Bottou, Wasserstein generative adversarial networks arXiv:1511.06434, 2016

# 1 Litterature review

## 1.1 Deep Convolutional Generative Adversarial Networks

### 1.1.1 Introduction

In computer vision, unsupervised learning consists in extracting image features, giving good intermediate representations of images, which can then be used for supervised learning problems such as image classification. In the literature, different options were developed to tackle the unsupervised learning issue but they present some difficulties. For instance, Adam Coates and Andrew Y. Ng [3] summarized several technical tricks to make K-means clustering more effective. But these methods appear to be very restrictive, mainly in term of quantity of data linked to their dimensionality, and cannot be applied in every case. The field of Generative Adversarial Networks (GANs) also has generated a strong interest. To mention just one example, in 2015 some authors [4] developed a GAN model with several generators. Despite getting better performances compared to traditional GANs, their model still suffered from being noisy and incomprehensible. In this context and in order to propose a more robust solution, Alec Radford, Luke Metz and Soumith Chintala define a new architecture based on CNN called Deep Convolutional Generative Adversarial Networks (DCGAN). They also implement several visualization experiences designed to enhance the network understandability.

### 1.1.2 Approach and Model Architecture

Historically, many articles highlight the inconveniences to use CNN architectures in order to scale up GAN models. Despite these difficulties, the authors succeed into finding an appropriate architecture allowing for a stable training of the GAN. This architecture is characterized by four main features.

- Replacing any pooling layers by **strided convolutions** for the discriminator and by fractional-strided convolutions for the generator. This change allows the network to learn more easily the downsampling of the discriminator and the upsampling of the generator.
- Using **batch normalization** in every layer (for both the generator and the discriminator), except the generator's output's layer and the discriminator's input's layer. It helps dealing with important training problems such as mode collapse (that we will define in more details in the summary of the second text), sample oscillation and model instability.
- **Removing the final fully connected layer**. Instead, the last convolutional layer is flattened and fed into a single sigmoid output.
- Using **ReLU** for all generators layers, except the last one. The use of this bounded activation function allows the model to learn faster and cover all the color space of the training distribution. For the discriminator, we switch to a Leaky-ReLU on all layers which appears well fitted for resolution modeling.

### 1.1.3 Empirical validation of DCGANs Capabilities

To evaluate the performances of their solution in the framework of unsupervised learning, the authors extract the features produced by DCGAN's feature extractor and use them on supervised issues based on labelled datasets (here CIFAR-10 and SVHN). They obtain a better accuracy than other traditional unsupervised models such as K-means. Only the Exemplar CNN[5] appears stronger. The success of Exemplar CNN is mainly due to the fact that the training process is done on special surrogate data.

[3] Coates, Adam and Ng, Andrew Y. Learning feature representations with k-means. In Neural Networks: Tricks of the Trade, pp. 561–580. Springer, 2012.

[4] Quan Hoang, Tu Dinh Nguyen, Trung Le, Dinh Phung - MGAN : Training GANs with multiple generators

[5] Dosovitskiy, Alexey, Fischer, Philipp, Springenberg, Jost Tobias, Riedmiller, Martin, and Brox, Thomas. Discriminative unsupervised feature learning with exemplar convolutional neural networks. In Pattern Analysis and Machine Intelligence, IEEE Transactions on, volume 99. IEEE, 2015.

### 1.1.4 Investigating and visualizing the internals of the network

Finally, to decrease the "black-boxness" of the CNNs used in GANs, the authors perform different explorations in the internals of the networks. The visualizations outline strong performances that can be demonstrated through three aspects.

- **Smooth transitions in the latent space**: Through interpolations between 9 random points, the latent space seems smooth, meaning that the model has not only memorized the data but learnt a coherent representation of the latent space.
- **Vizualizing the Discriminator Features** : In order to measure the performances of the discriminator, one can visualize the features learnt by the discriminator using guided backpropagation. With random filters no relevant discrimination was learnt while our learnt filters succeed into learning interesting discriminative filters (such as beds for example).
- **Manipulating the Generator Representation** : A generator can also be evaluated on its capacity to learn specific objects representations (such as windows, door). Two approaches were retained to assess the model's performances. First, specific item filters (such as windows filters) were removed. The generated images keep a high resolution quality and the general composition of the image is not affected. This means that the generator is able to distinguish between global composition and object composition. Second, consistency is again verified by performing arithmetics on the latent space. These experiments once again show that the latent space seems coherent and linear.

To conclude, this article suggests a new innovative and very performing GAN architecture based on convolutional networks that proves to be efficient when fed into supervised problems. The authors also implement useful vizualisations that contribute to reduce the inherent opacity carried by these very complex models.

## 1.2 Wasserstein GAN

The second article takes a more theoretical point of view by digging deeper into the workings of generative models and more specifically focusing on the topic of distances.

### 1.2.1 Introduction

In generative models and in the framework of unsupervised learning, one of the main assumption is that the real collected data comes from an unknown probability distribution $\mathbb{P}_r$. Consequently, we want to learn a distribution $\mathbb{P}_\theta$ that approximates the real distribution from a parametric family where $\theta$ is the parameter to optimize. A first classical approach is to directly learn the parametric distribution by optimizing the maximum likelihood estimation $\underset{\theta \in \mathbb{R}^d}{max} \frac{1}{m} \sum_{i=1}^{m} log \mathbb{P}_\theta(x^{(i)})$, which is equivalent to minimizing the KL-divergence $KL(\mathbb{P}_r || \mathbb{P}_\theta)$. However in the case where $\mathbb{P}_\theta$ has low dimensional support, the KL divergence is likely to be undefined.
To overcome this dimensional limit, another approach consists in adding some noise to the parametric distribution to ensure it to lie everywhere. But it introduces errors and it may be very costly to sample $\mathbb{P}_\theta$. These drawbacks motivate us to use a different method. We will define a Random Variable $Z$ with a fixed distribution $p(z)$ and pass it through a parametric function $g_\theta$ that would typically be a neural network. This function should generate samples from $\mathbb{P}_\theta$, with $\mathbb{P}_\theta$ being close to the real distribution of data. However, to train $g_\theta$, we need to define a metric that measures the distance between $\mathbb{P}_r$ and $\mathbb{P}_\theta$ without adding noise. The authors in the article thus introduce the *"Wasserstein-GAN"* : a generative model that minimizes an approximation of the Earth-Mover (EM) distance. They demonstrate that this model gives better performances than other forms of GANs and that this comes from the chosen distance metric.

### 1.2.2 Different Distances : specificity of Earth Mover distance

Other authors outlined problems that arise when using generative models for learning a specific unknown distribution with traditional distances. Martin Arjovsky and Leon Bottou [6] for example, already showed that

[6] Martin Arjovsky, Leon Bottou - Towards Principled Methods for Training Generative Adversarial Networks

using KL and Jensen-Shannon distances brings instability in GANs and gave first theoretical intuitions on the fact that the Earth Mover distance appears to be a better tool.
To better understand this Earth Mover metric, let's take two different distributions $\mathbb{P}_r, \mathbb{P}_g$ and assume that we want to move the mass of the former to the mass of the latter. In this context we define a transport plan $\gamma \in \Pi$ that will move the mass from $x$ to $y$. The Earth Mover distance scores the minimal effort spent to transport the amount of mass that leaves from $x$, $\int_y \gamma(x,y)\mathrm{d}y = \mathbb{P}_r(x)$ to the amount of mass that enters $y$, $\int_x \gamma(x,y)\mathrm{d}x = \mathbb{P}_g(y)$. We can define it as follows:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \int_x \int_y \gamma(x,y)||x-y||\mathrm{d}y\mathrm{d}x = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)}[||x-y||]$$

Compared to the other metrics (KL distance, Total Variation, Jenson-Shannon divergence), the EM distance presents two main advantages that are described in Theorem 1 and 2. First in the GAN framework, only the EM distance is guaranteed to be continuous and differentiable (Theorem 1) which is a desired property for a loss function. Second, the EM distance is the weakest amongst these metrics : every distribution that converges under the KL, JS and TV divergence also converges under the EM distance.
Here we can add that other metrics developed in statistical reviews can give performances as convincing as the EM distance. The Maximum Mean Discrepancy (MMD)[7], an integral probability metric where functions belong to a Reproducing Kernel Hilbert Space, provides excellent results in the framework of GANs. However, computing such a metric is very costly and grows quadratically with the amount of samples. In conclusion, the Earth Mover distance today appears to be a really efficient metric for optimization problems. This explains why researchers started to focus on how to develop optimized algorithms to compute this distance[8].

### 1.2.3 Wasserstein GAN

The next challenge is the Earth Mover distance computation. Because the infimum is intractable (making it difficult to compute both the distance and its gradient), the authors suggest an approximation based on the Kantorovich-Rubinstein duality defined as follows:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \approx \sup_{||f||_L <= K} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \approx \max_{w\in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z)]$$

where f is a K-Lipschitz function for some unknown K, parameterized by $w \in W$. Using this approximation we assume that the supremum is reached for some $w \in W$. The authors introduce a third theorem stating that if $g_\theta$ is a locally Lipschitz function then the gradient of the EM distance, using the previous approximation, can be deduced by $\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim(z)}[\nabla_\theta f(g_\theta(z))]$.

With this in mind, the training process can be defined as follows:

- For a given $\theta$ we optimize the parameter $w$ of $f_w$ by computing the gradient $\nabla W(\mathbb{P}_r, \mathbb{P}_\theta)$
- Once $f_w$ has been optimized, we compute the gradient of $\theta$ by sampling a random variable $z$ (with a fixed distribution $p(z)$) and using the new weights $w$ to update
- $\theta$ is updated and the process is repeated until its convergence

It is important to note that, the algorithm can converge only if the function family $f_{w\,w\in W}$ is K-Lipschitz. To ensure this, the authors chose to constrain the weights $w$ within a clipping interval $[-c, c]$. Finally a key difference with classical GANs approaches concerns training. Usually when implementing traditional GANs, we do not train the discriminator up to convergence as the discriminator would become too strong. In our model and because the Wasserstein distance has good differentiability properties, the discriminator is trained up to convergence before each generator update.

[7] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Scholkopf,and Alexander Smola. A kernel two-sample test. J. Mach. Learn. Res., 13:723 773, 2012

[8] Aude Genevay, Marco Cuturi, Gabriel Peyre, and Francis Bach. Stochastic optimization for large-scale optimal transport. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29

#### 1.2.4 Results

WGAN solve two classical GAN problems : vanishing gradients and mode collapse.

- **Vanishing gradients** (gradients coming close to zero) are usual causes of failures of the optimization process. WGANs prevent vanishing gradients thanks to the weight clipping process.
- **Mode collapse** is one well-known form of GANs failure in literature. It happens when the discriminator gets stuck in a local minimum and does not find the best strategy. Thus, it is easy for the next generator to find the next output. As a consequence, the GAN does not explore a wide variety of outputs because each iteration of generator over-optimizes for a particular discriminator. WGAN prevents mode collapse due to the fact that the discriminator is trained until optimality : the generator is thus forced to try new outputs.

Finally, the article presents results on empirical experiments. These illustrations verify the two previous improvements. Moreover, the EM distance appears to be well correlated to the visual quality of the generated images. Said differently, the lower is the loss, the higher the resolution of the image. Finally WGANs also improve training stability and robustness when combined with changing architectures.

# 2 Cycle - GAN

In this second part we will study a different GAN architecture : the Cycle-GAN. We will first provide a brief overview and general summary of the paper that introduced this technique[9] and then present an extensive implementation to translate between MNIST and USPS datasets.

## 2.1 Idea and theory behind the Cycle-GAN method

#### 2.1.1 Context and objective

Cycle GANs tackle the topic of image-to-image translation : a class of computer vision problem where the goal is to convert images from one source domain $X$ to a target domain $Y$ (for example converting a day picture into a night picture or an impressionist painting into a realistic one). The first idea to approach this task is to use aligned training sets (datasets where each image $x_i \in X$ has a corresponding translation $y_i \in Y$ in the target domain). But these datasets are in practice very tedious or even impossible to build and when they do exist, they often lack data to train massive models. In the Cycle-GAN paper the authors thus present a new approach to translate an image from a source domain to a target domain without any paired samples.
They assume that there exists a "meaningful" relationship between both distributions that the model will try to identify. The goal of the model can thus be summarized as learning a mapping function G : X $\rightarrow$ Y such that the output $\hat{y} = G(x)$ is almost impossible to distinguish from images $y \in Y$ by an adversary (adversarial loss). However, only resorting to an adversarial loss cannot guarantee that a "meaningful" translation will be learnt. Indeed there would be many different ways to generate images in $Y$ starting from $X$. This is why the main challenge of image-to-image translation is to ensure "cycle-consistency". "Cycle consistency" is the idea that if we translate an image to the target domain then translate it back to the source domain, we would like to obtain something similar to our source image. This condition creates the need to put some control on an inverse mapping : F : Y $\rightarrow$ X. Both G and F will be trained simultaneously and controlled by a cycle-consistency loss ensuring that $F(G(x)) \approx x$ and $G(F(y)) \approx y$.

#### 2.1.2 Details on the agents and loss function

To be more specific : given training samples $x_i \in X$ and $y_j \in Y$, we want the model to learn two mappings $F$ and $G$ (that translate between domains X and Y). On top of these two mappings we introduce two adversarial discriminators : $D_X$ (aims to distinguish between $x$ and $F(y)$) and $D_Y$ (aims to distinguish between $y$ and $G(x)$).

[9] Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros, 2017, arXiv:1703.10593

Concerning our target functions, the final mappings should satisfy : $G^*, F^* = \arg\min_{G,F} \max_{D_x,D_y} \mathcal{L}(G, F, D_X, D_Y)$ where our full objective $\mathcal{L}(G, F, D_X, D_Y)$ can be decomposed into three terms :

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

The two first terms represent the adversarial loss for the mapping functions $F$ and $G$. More precisely

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{y \hookrightarrow p_{data(y)}}[log(D_Y(y))] + \mathbb{E}_{x \hookrightarrow p_{data(x)}}[log(D_Y(G(x)))]$$

where G tries to generate images $G(x)$ that look similar to $y$ and $D_Y$ tries to discriminate real from fake samples. The last term represents the cycle consistency loss. It is composed of two parts : the forward cycle consistency term and the backward consistency term :

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

.

### 2.1.3 Detailed architecture and training procedure

Both generators $G : X \to Y$ and $F : Y \to X$ have the same structure and can be divided into three main blocks. The first stage encodes the input via a series of convolutional features that extract the image features. Then, the second stage transforms the features by passing them through one or more residual blocks. These residual blocks consist of a convolutional layer where the input is added to the output of the convolution. This is done so that the characteristics of the output do not differ too much from the input. Finally, in the third stage the transformed features are decoded using a series of transposed convolutional layers and an output image (same size as the input) is produced. The detailed architecture is printed out below (Figure 1).

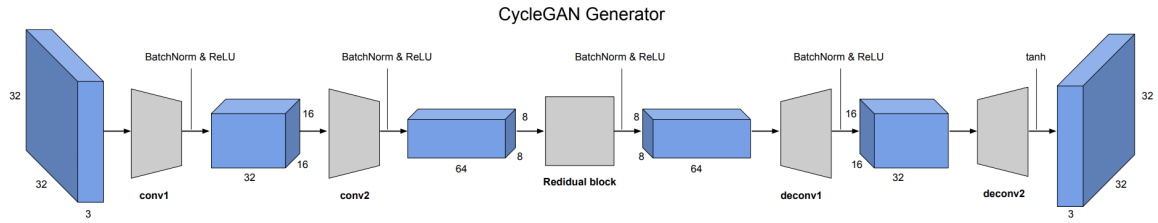


Figure 1: Cycle GAN Generator Architecture

In order to stabilize the training the authors suggest two measures : first the negative log-likelihood objective is replaced by a least-squares loss and second the discriminator is trained on a history of stored generated images (and not only on the images created by the latest generator). Regarding the optimizer, the authors chose to use Adam and the initial learning rate of 0.0002 linearly tends to 0 after the first 100 epochs. $\lambda$ is set to 10. The training is done with mini-batches of size 1, each training loop being divided into two parts. First the "discriminator loss" is computed on real and fake images and the discriminators are updated. The goal at this stage is to improve their capacity to detect real from fake samples in both distributions. Second we compute the generator cycle-consistency loss and update the generators. Here the objective is to reduce the difference between the input images and their reconstructions obtained by passing through both generators sequentially.

### 2.1.4 Results

The cycle GAN achieves far better performance than other unsupervised techniques (CoGAN, BiGAN,ALI, SimGA, Feature loss + GAN) and produces translations that are often of similar quality to the fully supervised pix2pix (trained on paired data). Moreover removing any part of the loss shows a substantial degradation in the results demonstrating the power of this combined adversarial/cycle-consistency objective. Nevertheless,

the authors point at a few limits of the model. CycleGANs appear extremely convincing when translation turns around evolving textures and colours but have harder times converting one geometry to another (for example turning a cat into a dog). Moreover, the performance gap with models trained on aligned datasets seems impossible to fully bridge.

## 2.2 Implementation

In this section we implement a CycleGAN architecture in order to translate from the MNIST dataset to the USPS dataset. We follow Roger Grosse's detailed assignment. The full code is available here [10].

### 2.2.1 Objective and Parameters

USPS and MNIST are both handwritten digits datasets but they differ by their resolution. USPS images are $16 \times 16$ while MNIST images are $28 \times 28$. In order to feed our Cycle GAN we will need our input and output to have the same size. This is why we resize USPS images to a $28 \times 28$ format, thus lowering their resolution. In the end, our assignment can be summarized as translating low-resolution figures into higher-resolution figures. The architecture of the two generators and the two discriminators follows the original paper (described in 2.1.3). The only differences are the necessary adjustments in padding and stride in order to match our $28 \times 28$ inputs/outputs, the use of a fixed learning rate (0.0003) instead of a linearly decaying one and a batch size of 16 (instead of 1). With this parametrization we implement two experiments that we execute on GPU in order to make calculation faster.

- In the first experiment we compare two training results based on two different Python random seeds (10 and 11). In both cases we train on 10 000 iterations, with a cycle-consistency loss ($\lambda = 10$ as suggested in the CycleGAN Paper).
- In the second experiment we study the impact of $\lambda$ on the quality of our results by changing its value and training five models ($\lambda = 0, 3, 6, 10, 15$). The scenario where $\lambda = 0$ corresponds to the case where no cycle-consistency loss is used.

In order to analyse our network's convergence we frequently (every 200 iterations) test our two generators $G : X \to Y$ and $F : Y \to X$ on fixed data samples. This qualitative analysis is essential to assess our network performance and convergence during training as the loss curves do not reveal much information when training GANs.

### 2.2.2 Results

**First experiment**
As described above, in the first experiment we repeat twice the same training only changing the random-seed. This random seed plays a crucial role as it will impact the weights initialisation of our network. The results can be seen in Figure 2 where we display the result from our generators at different time-steps.
We can highlight several points. First our model seems to converge pretty well. While at iteration 200, the generators do not produce satisfying and sharp results, it seems that they stabilize and converge as soon as iteration 1000. Indeed the outputs aspect does not vary much in the last iterations. Then, the initialization of the weights plays a key role : we obtain a superior result with random seed fixed to 10 (compared to random seed equal to 11). Lastly we detect a difference in quality depending on the translation direction. If we observe the samples obtained via random seed 10 we can clearly see that the MNIST $\to$ USPS generator performs better at imitating USPS style than the USPS$\to$ MNIST generator. This seems logical as we would naively imagine that passing from a high-resolution image to a low-resolution image is easier than the reverse path.

**Second experiment**
In the second experiment we repeat the same training only changing the $\lambda$ parameter. As explained before, this

[10] https://colab.research.google.com/drive/1UNmAQkcQzbsyuzjwB1xndZIyJ3acPTM7?usp=sharing

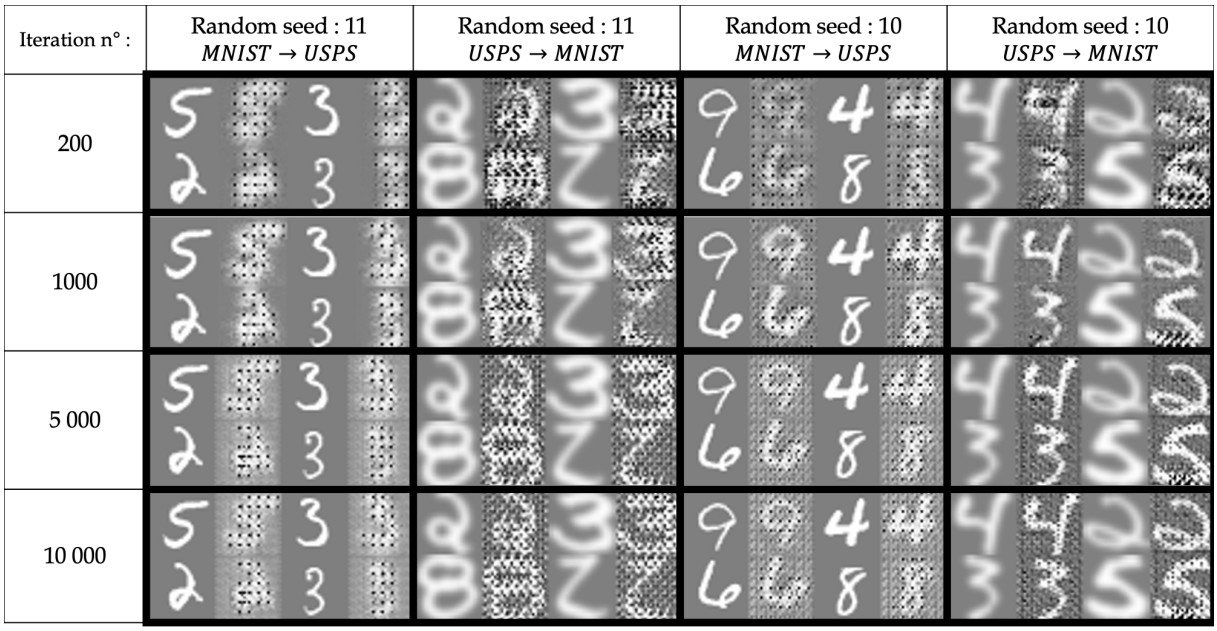


Figure 2: Results for the first experiment

parameter defines the relative weight of the *cycle-consistency loss* in the total generator loss. The results can be seen in Figure 3 where we display the results from our generators at iteration 200 and 10 000 for different values of $\lambda$.

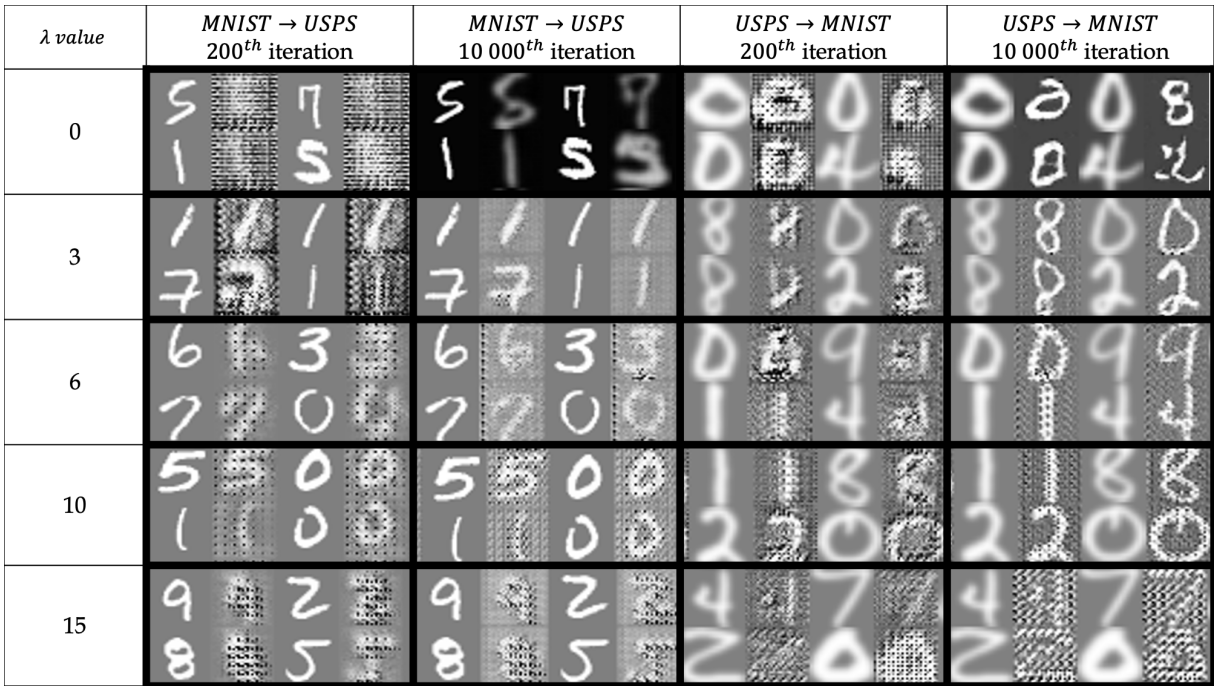


Figure 3: Results for the second experiment

Several conclusions can be drawn. At first sight the network trained without consistency loss might seem better than the others. Indeed the background in the final result is really sharp without any grey spots. But if we look closely at the result we can see that this generator fails to genuinely *translate* from one dataset to another. The outputs are coherent with the destination domain but lack consistency with their inputs (a 0 in USPS becomes an 8 in MNIST, a 5 in MNIST becomes a 9 in USPS etc). The lack of meaningful relation between input and

output completely disappear when we introduce cycle-consistency loss. It is interesting to see that when $\lambda$ increases, the *"link"* between input and ouput is better and better conserved but the coherence with the target domain decreases. This observation is striking on the USPS→MNIST translation where we clearly see that the outputs from the generator get thicker and thicker (ie getting further from a sharp MNIST digit and closer to a low-resolution USPS figure). This shows the need to properly fine-tune the $\lambda$ hyperparameter. In our example $\lambda = 10$ seems to offer the best compromise between cycle-consistency and output domain adaptation. This empirically confirms the choice of $\lambda = 10$ in the CycleGAN paper.

### 2.2.3 Next steps and conclusions

In a nutshell, with good fine-tuned hyper-parameters the Cycle-GAN succeeds well at reproducing USPS images from MNIST ones and the reverse path. To go further, several works could be added to our experiments. First it could be interesting to test different learning-rates. More precisely, using a linear decreasing learning-rate through iterations (as described in the Cycle-GAN paper) could improve the performances of the Cycle-GAN. In fact, such a mechanism would make the convergence of the Cycle-GAN easier after the first iterations. Secondly, as we already noticed when testing different random seeds, the weights initialization is really important. Trying several other methods with different probability laws and parameters to initialize them could improve performances. Finally, it is to be noted that we trained our model on very standard black-and-white images. This is a good start to understand how to implement such models, however it is difficult to make real and serious conclusions on the results. It would be more relevant to apply the same architecture on more complex images (e.g. facial images) and evaluate them. This would nevertheless require much more computation power in order to increase the number of iterations.

## 3 Conclusion

To conclude, this study allowed us to dig deeper in the very promising field of GANs. The two first papers helped us better understand the specificities of these models in terms of architecture and metrics used. The third paper about Cycle GANs made us discover a slightly different approach. Indeed in the classical GANs framework, the main goal it to generate images, whatever their shape, and make them relatively similar to the target domain to deceive the discriminator. The Cycle GAN approach is a bit different as it is a one-to-one mapping. In other words, the generated image must match the target domain while staying faithful to their source image.

Despite this difference, implementing a Cycle GAN enabled us to get a stronger sense of the challenges and limits that come with the use of GANs despite their astounding results. The difficulty to quantitatively evaluate their performance and the lack of stability of the training (especially the high sensitivity to hyperparameters and weights initialization) make GANs giants with clay feet.

Finally, even if GANs today appear as a gold standard when it comes to image generation, they can be challenged by other approaches. In particular VAE models have the advantage of being less opaque than GANs thanks to the mapping they provide between a latent space and the output domain. However their results are usually of lower quality compared to the ones from GAN models. One potential prospect for improvement might thus be to combine these two approaches.