

ÉCOLE POLYTECHNIQUE
MSc&T Artificial Intelligence and Advanced Visual Computing
Academic Year 2020-21
LORANCHET Guillaume

INTERNSHIP REPORT

« Deep learning applications in Visual Effects »

NON CONFIDENTIAL REPORT

Referent teacher: Erwan Scornet
Internship tutor: Vincent Champsiaux
Internship dates: 29/03/2021 – 30/07/2021

Declaration of Academic Integrity

Hereby I, Guillaume Loranchet, confirm that:

1. The results presented in this report are my own work.
2. I am the author of this report
3. I have not used the work of others without clearly acknowledging it, and quotations and paraphrases from any source are clearly indicated.

Guillaume Loranchet, 03/09/2021

Signature

A handwritten signature in black ink, consisting of a stylized 'G' followed by a horizontal line and a small loop.



Deep learning applications linked to visual effects

Guillaume Loranchet

April to July 2021

Abstract

When I arrived, MacGuff had precise needs for their upcoming TV-production "L'Hotel Du Temps": they had to de-age actors and to make talking pictures. They already had some promising results but were looking for ways to improve their workflow and the rendered quality. While they knew from previous productions that they could achieve a great result by hand, they were looking for new methods to speed up the process. Rodolphe Chabrier, president of the company, knew a lot of research was done in Artificial intelligence in this area and is convinced that IA will change the way visual effects are made. Generative models, and in particular a relatively new method called GAN (Generative Adversarial Networks) quickly came to mind. During my internship, I thus closely studied GANs and tested some of the most recent papers published in this domain. While MacGuff is most interested in concrete results, I tried to add a more theoretical approach. In this report, I first explain the overall functioning of GANs, from the main issues encountered when training one, to some of the most important objective functions used throughout the years. I then present a state of the art method known as StyleGAN before finally explaining the methods I tried and showing the results I obtained.

1 Presentation of the internship

1.1 MacGuff

MacGuff Ligne is a VFX production company founded in 1986. They were first doing effects for advertising and music video before making it to movie, with notably *La Haine* by Mathieu Kassovitz in 1995. They are considered as pioneers in France regarding 3D animation and digital effects. After directing *Despicable Me* in 2010, the studio split in two: Illumination MacGuff, affiliated to Universal Studio that specialized in animated movies and MacGuff Ligne (or simply MacGuff), which kept doing visual effects for ads, series or

feature films. They are also still making animation with for example *Dilili à Paris* that won the Cesar award for best animated film in 2019. MacGuff is also really interested in research and in particular in IA. They, for instance, won the Cesar Technique innovation price in 2021 for their new tool: Face Engine. While it is often reserved to huge productions, MacGuff managed to make a more simple but efficient instrument for facial transformations like de-aging. It was notably used in the last season of the series *Le Bureau des Légendes*. Recently MacGuff were responsible for the visual effects in *Oxygène* by Alexandre Aja and *Titane* by Julia Ducournau who won the Palme d'Or at Canne festival for best movie, in 2021 also.

1.2 IA and visual effects

IA is gaining more and more importance in the visual effects production. Every major company invests a lot in machine learning research. For example, Disney published several innovative papers on rejuvenation: for instance a young Marc Hamill in the *Mandalorian* season 2 or a young Carrie Fisher in *Rogue One*. Both methods, with a lot of manual fine tuning, relied heavily on Artificial Intelligence. Another more concrete example directly concerns MacGuff, who had to de-age several actors in 2016 for the movie *Eternité*. It took them more than 30 different people over the course of several weeks to manually remove all facial traits to make people look younger. Even if IA models are still a work in progress and often require a long and expensive training, in the end, the economic and energetic cost is much lower. One more instance of IA use in visual effects is rotoscoping (the fact of outlying a silhouette to separate it from the background). Even for green or blue screens sequence, most of the time, the job had to be done manually, necessitating days of tedious effort. Nowadays, most softwares have plug-ins to efficiently track an object or person, and do most of the work. Nonetheless, IA does not aim at replacing every digital artists but rather give them more time to work on actual artistic improvement.

As often in its history, MacGuff tries to stay on the bleeding edge of new technologies and strongly believes that the democratisation of IA will be a revolution in visual effects, like digital was compared to manual effects. However, the market for visual effect in France is far from super-production made in the US (with ILM, Disney studios...) or New Zealand (Weta Digital), and thus impacting the state of research. Very few visual effects company in France, MacGuff included, can afford to put resources in it, as it requires a lot of time and computing power.

1.3 My experience

As said previously, MacGuff is quite small compared to huge American corporations but still is one of the most important visual effect studio in France and in Europe. The research and development team I was working with pretty small as most of the people working there are graphists and visual effects artists. I mainly interacted with Vincent Champsiaux, the head of the team and responsible for almost all the technical problems at MacGuff, and Yann Le Paih, the only other person working on IA with me. The first 2 months of my internship was dedicated to test all kind of models that looked interesting. Due to the large number of models to test, we had to be pragmatic and we could not afford to spend too much on each paper: in most cases, if after a few days of trying, we did not get any results, we moved on to the next one. It was an enriching experience, as I was able to discover many papers and gave me a good overview of the state of research in the area of generative models. Progressively, I started to fill up a Notion document to note the state of advancement in each project. In the end, we tested more than 18 different models. In parallel, Yann was writing a weekly report on the news concerning IA. Nonetheless, it was also sometimes frustrating as we did not have time to dive into each subject and had to face a lot of failures. Indeed, most of the time,

either the code of the paper was not online, we also often had trouble installing all the requirements or the pretrained weights was missing.

During the last month and a half, I focused more on re-training some models, in particular First Order Motion Model [17] to improve the resolution and InterpGaze [14] where the weights were missing. It was quite challenging for me as it was the first time I had to train huge models almost from scratch. I was also limited by the computing power available, as most of the big machine were used by 3D artists or for rendering. Due to this, the long training (sometimes taking more than a week) and the short time I had, I could only do a few different testing. Even if some results were encouraging, it was not enough for production usage. Nevertheless, it was a great experience and I had a lot of fun of testing all the different models. It also taught me how hard and how long it takes to produce a great machine learning algorithm, especially when dealing with generative models.

Disclaimer: all the results that we discuss in the following do not reflect the overall quality of one's paper but simply what we managed to outcome. We would also like to thank all the researchers for sharing their code to the public.

2 GAN

2.1 Overview

GAN (Generative Adversarial Network) is a class of machine learning methods and more specifically of generative models. Data generation, often images but not only, is a wild field and existed way before the introduction of GANs. The goal of a generative model is to create new content that does not already exist, based on a training data set that we provide. Most of the time, the parameters of the training data distribution are unknown: we are only given data points and it is the role of the model to find a **statistical distribution** of parameters W that fits the data the most. More precisely, we want to find the parameter W that maximizes the likelihood of real samples. A **likelihood function** measures the goodness of fit of a statistical model for data samples. It depends only on the parameter (the random variables are fix). Thus, finding the maximum likelihood is equivalent to finding the parameter value that best describes the observed data. Or in other words, minimize the distance between our generated distribution and the actual data.

Generative models then separate into two groups: the **explicit** and the **implicit** distribution models. The first one needs to explicitly defines a distribution (for example Gaussian). The parameters of the distribution are then improved by training the model on the real data set. On the other hand, we have implicit models like GANs. They sample vector from a latent space which is a high dimension space where each point generates a unique image.

GANs have first been introduced by Ian Goodfellow et al. [2] in 2014. The main idea behind GAN is to have two networks, a **generator** and a **discriminator** confronting each other, hence the name. The generator creates images while the discriminator has to accurately separate fake from real images coming from a training set (see figure 1). To fool the discriminator, the generator must create the best images possible. . In other words, if the model is presented only images from dogs, it will try to recreate dogs. GANs are thus completely unsupervised learning methods. Unsupervised learning has a lot of advantages, mainly the fact that no labeling is needed which is truly handy when working with thousands of images. However, the unsupervised part makes GAN quite complicated to train.

The original **objective function** (i.e the goal of the model) was defined as follow:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))] \quad (1)$$

where G is the generator, D the discriminator, x an image sampled from real dataset $q_{data}(x)$ and z a vector sampled from the latent space $p(z)$. Thus, to maximize this function, the discriminator has to output a huge number (in this case, 1 is the maximum) when it thinks the image comes from the real data and a small number (0 is the minimum) when it comes from the generator. Conversely, the generator can only minimize the right part by fooling the discriminator and making it output a 1.

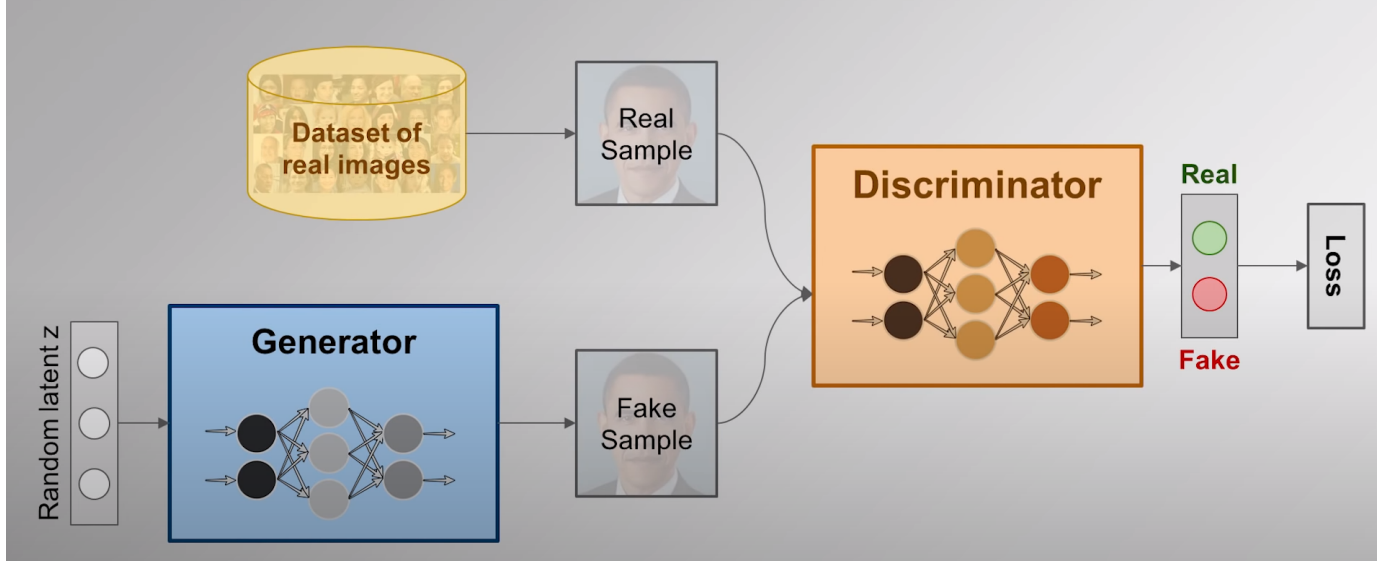


Figure 1: GAN overall architecture ([source](#))

In the original paper by Ian Goodfellow et al. [2] both the discriminator and the generator are multilayer perceptrons and are thus fully differentiable. Which means that we don't need to use approximation scheme and can do backpropagation across the entire network. The pseudo code describe in the paper goes as follow:

- For each epoch **do**
 - For k steps **do** (k is a hyperparameter, chose beforehand)
 - * sample $x_1, \dots, x_n$ images from the training set at random
 - * sample $z_1, \dots, z_n$ random vectors from the latent space
 - * update the discriminator by doing stochastic gradient ascent (as we want to maximize the objective function with respect to the discriminator)

$$\nabla_{\theta_d} \frac{1}{n} \sum_{i=1}^n [\log D(x_i) + \log (1 - D(G(z_i)))]$$

- For k steps **do**
 - * sample $z_1, \dots, z_n$ random vectors from the latent space

* update the generator by doing stochastic gradient descent

$$\nabla_{\theta_g} \frac{1}{n} \sum_{i=1}^n [\log (1 - D (G (z_i)))]$$

The important part to remember is that the generator and the discriminator are **trained separately**. when one of them is being updated, the other one is fixed. However, to spare some computational resources, Goodfellow et al. [2] chose to set $k = 1$. This means that they both update only once before the next iteration.

2.2 Improvement

Training a GAN could turn out difficult. Indeed if the generator and the discriminator are not balanced well enough, the training may become too easy or too hard for one of the networks. For example, if the discriminator guesses correctly each time, the generator will have no information to improve itself. The three main issues when training a GAN are:

- **Non-convergence:** the model doesn't converge or is unstable
- **Mode collapse:** lack of diversity and all images look the same (see example in figure 2)

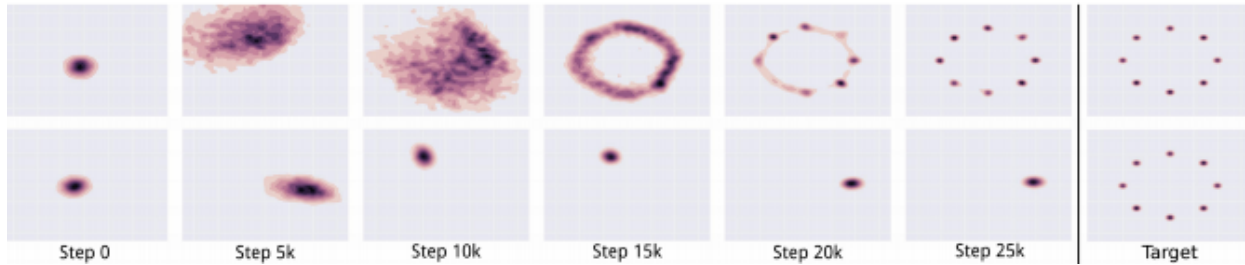


Figure 2: Top: a model that successfully recreated the 8 modes of the target, Bottom: the generator and the discriminator are trained for too long before alternating. For a fixed discriminator, the generator finds the mode that fools it the most. When the discriminator discover the mode used, the generator can simply use another one resulting in a never-ending cat and mouse game. ([source](#))

- **Slow training:** cases when the gradient vanishes

Nonetheless, throughout the years, a lot of solutions have been found to fix those issues. We will present seven ways of improving GANs that mainly come from the paper *Improve Techniques for Training GANs* by Salimans et al. [1].

- **Feature matching:** Feature matching is a new objective function to regulate the generator so that it does not overfit for a given discriminator. It is especially useful to stabilize the training. Rather than simply maximizing the discriminator loss, feature matching also aims at minimizing the distance (L2 norm in this case) between features of generated and real images:

$$\| \mathbb{E}_{x \sim p_{data}} \mathbf{f}(\mathbf{x}) - \mathbb{E}_{z \sim p_z(\mathbf{z})} \mathbf{f}(G(\mathbf{z})) \|_2^2$$

where f is a feature vector extracted from an intermediate layer of the discriminator. Indeed, before outputting a binary result (fake or real), the discriminator has to simplify the image via a neural network to find features he deems important. Hence, we are not comparing fake and real images directly but rather features selected by the discriminator. The generator therefore knows in which direction to go to better fool the discriminator. Furthermore, the features of real image are selected from a minibatch at each iteration adding randomness to the process, which is often good to avoid overfitting.

- **Minibatch discrimination:** Minibatch discrimination is used to limit mode collapses. We first feed two distinct mini-batches to the discriminator: one of real images and one of generated images. Then, for each image, we compute the similarities with all other images of the same batch. The similarity, written $o(x_i)$ between image x_i and all other images, is computed as shown below:

- We first multiply the features with a transformation matrix T : $M_i = f(x_i) * T$
- Then the similarity $c_r(x_i, x_j)$ between image x_i and x_j is computed as the negative exponential of the L1-distance for each row r of the matrices M_i and M_j :

$$c_r(x_i, x_j) = \exp(-||M_{r,i} - M_{r,j}||_1) \in \mathbb{R}$$
- Finally, the similarity $o(x_i)$ is the sum and concatenation of all pairwise similarities.

If the generator doesn't make images diverse enough, the similarity score will be lower than for real images. The discriminator can then use this information to better detect fake images and punish the generator. The generator will hence be forced to add more diversity. Salimans et al. [1] commented that thanks to its quickness to generate visually appealing images, minibatch discrimination was superior to feature matching. However, the latter works way better to obtain a classifier using a semi-supervised approach.

- **One sided label smoothing:** One-sided label smoothing serves to reduce overconfidence for the discriminator. In some cases, it uses only a small set of features to classify images. The generator can use this flaw to emphasis specific features to trick the discriminator without improving or changing the overall quality of the image (see image below). This process is called **adversarial example** and is a very active research topic as it represents a serious threat in security domains.

It was recently shown that smoothing the labels, i.e change the target from 1 to 0.9 for example, helps counter adversarial examples. It is only done for the discriminator and real images. By reducing the reward, it penalizes the discriminator for being overconfident and outputting a 1 for real images.

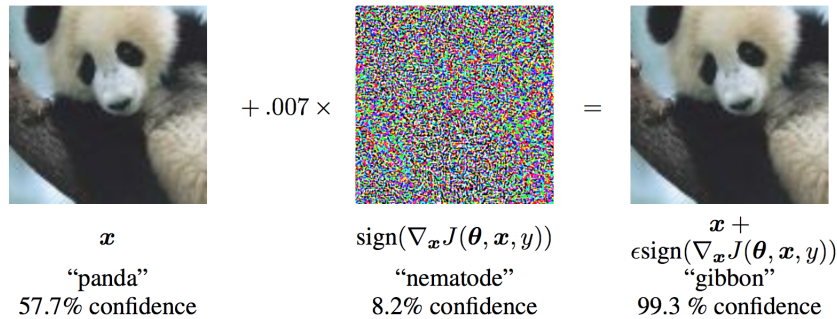


Figure 3: adversarial example: the two images look exactly the same but by adding noise at the right place, the model is fooled

- **Historical average:** Historical average consists of keeping in memory previous parameter values and penalizing the model if it's moving too much away. The parameters are averaged over time to avoid taking too much importance over time. Historical average helps low dimensional and non-convex models to converge, rather than turning around the equilibrium point. The following equation is added to the cost functions:

$$\|\sigma - \frac{1}{t} \sum_{i=1}^t \sigma[i]\|_2^2$$

where $\sigma[i]$ is the parameter values at time i

- **Add label:** Adding labels to either or both the generator and the discriminator, often helps the model to train faster. Indeed, it gives more information for it to learn from. This class of GAN are called conditional GANS and are thus semi-supervised models, which mean more work on the human side to prepare the data.

2.3 Objective function

The underlying goal of GAN is to reduce the difference between two data distributions, the real data set on one hand and the latent space generated by the model on the other. To measure the distance between them, we use an objective function. The choice of the right objective function has been the subject of many studies as it is a key element of a GAN. We will briefly present the main objective functions that have been used throughout GAN's history. In the following, let $\mathbf{p}$ be the real data and $\mathbf{q}$ the generated data distributions.

- **Kullback-Leibler Divergence (KLD):** The KL divergence was used for the **Maximum Likelihood Estimation** (MLE) of generative models before GANs existed. As described above, MLE consists of finding the best model parameters for data with unknown parameters. It can be shown that maximizing the log-likelihood is equivalent to minimizing the KL divergence (sometimes the word distance is used but the KL divergence doesn't define a metric in a mathematical sense). It is defined as follow

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

The KL divergence measures how much our estimated distribution q differs from the real-life distribution p . However, it is not symmetrical, i.e $D_{KL}(p||q) \neq D_{KL}(q||p)$ (see example below). Thus, using one or the other has different implications for the model. For example, if we use $D_{KL}(p||q)$, we can see it is high when p is high and q is low (around $x = -2$). It means the model is highly penalized for not making images (described by the q distribution) that are represented a lot in the original dataset (p distribution). The divergence is then zero when $p = q$ and close to 0 for positive values, which means the model is not penalized much for making images that are far from the dataset. In other words, it generates diverse but low-quality images. If we use $D_{KL}(q||p)$ instead, the inverse happens: the model creates realistic but not diverse images.

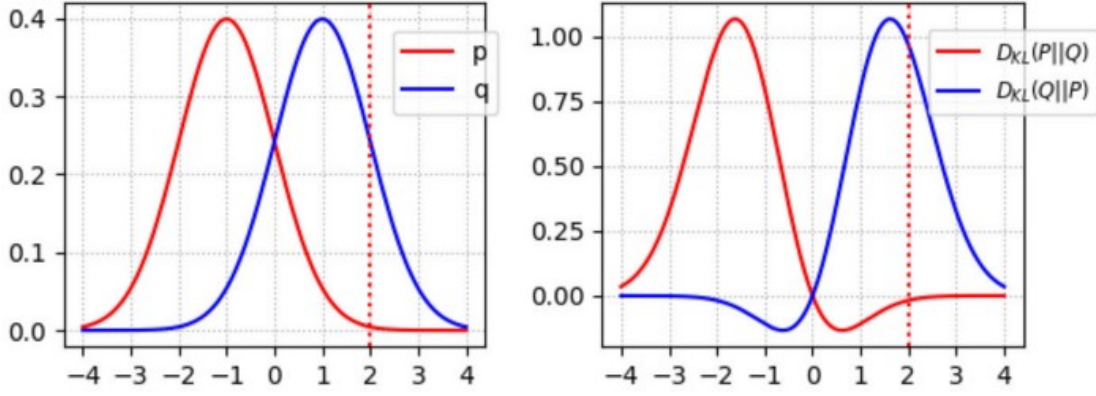


Figure 4: left: two distributions, right: KL divergence of p w.r.t q and the inverse ([source](#))

- **Jensen-Shannon Divergence (JS-D):** The JS divergence, also called the total divergence to the average is based on the KL divergence with few more proprieties: it is symmetrical and defines a metric.

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||m) + \frac{1}{2}D_{KL}(q||m) \quad \text{with} \quad m = \frac{p+q}{2}$$

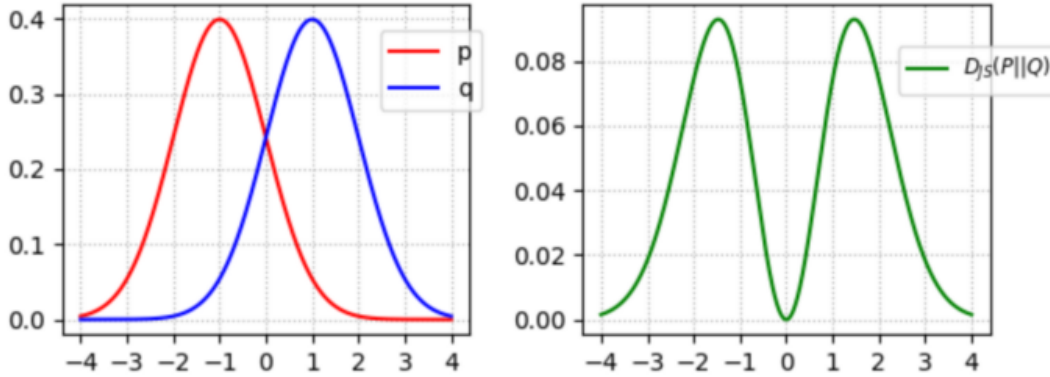


Figure 5: right: JS divergence between p and q ([source](#))

The JS divergence, unlike the KL one, can penalize both bad quality and not enough diversity at the same time. It was the original objective function used by Goodfellow et al. [2] in 2014. They also showed that optimizing the JS divergence is equivalent to optimizing the min max objective function of a GAN (see equation 1). If we denote $C(G) = \max_D V(G, D)$ the function that give the optimal discriminator for any given generator, we obtain that $C(G) = -\log(4) + 2 \cdot D_{JS}(p||q)$ (See the proof in the appendix). Thus, as the JS divergence is non negative and 0 only when the two distributions are equal ($p = q$), minimizing $C(G)$ is equivalent to minimizing the JS divergence and the optimal value is $C^* = -\log(4)$

3 StyleGAN

3.1 Version 1

StyleGAN, or Style-Based Generator Architecture for GAN, by Karras et al [4] at Nvidia, was introduced in 2019 as a new method for image generation. It is based on another paper by Karras et al [3] published a few years earlier that presented **Progressive Growing** to improve image quality. Indeed, generating high-resolution images has always been a challenging task for two reasons. The first one is simply an exponential use of the memory to train the model as every time we double the resolution, the number of pixels is multiplied by 4. The other reason is that the discriminator has more details to differentiate real from fake images. As said previously, if the discriminator is too good, the generator can't know in which direction he should go to improve. Progressive growing consists of starting with low-resolution images (4x4) for both the discriminator and the generator until convergence. The model then gradually increases the resolution to reach the desired quality. The low-resolution start makes the training harder for the discriminator as it has fewer details to differentiate real from fake images. According to their paper, this method greatly improves the stability of the generation and the speed of the training.

One of the most important change StyleGAN brought was to change the initial latent vector $z \in Z = \mathcal{N}(\mu, \sigma^2)$ sample from a Gaussian to a *style code* $w \in W \subsetneq \mathbb{R}^{512}$ via a mapping network f . The main difference is that the distribution of W is unknown and constructed by f . Yet, not all feature distributions are sample uniformly (see figure 6). But as Z is, it needs to warp the space to fill the gaps. W on the other hand, does not have such constraint and can better describe the distribution of the features. This small change eases a lot the training of the generator as it has a more one-to-one connection between the latent and the real data distributions. This principle is called **disentanglement**.



Figure 6: Example of a Gaussian distribution over data with a gap (source)

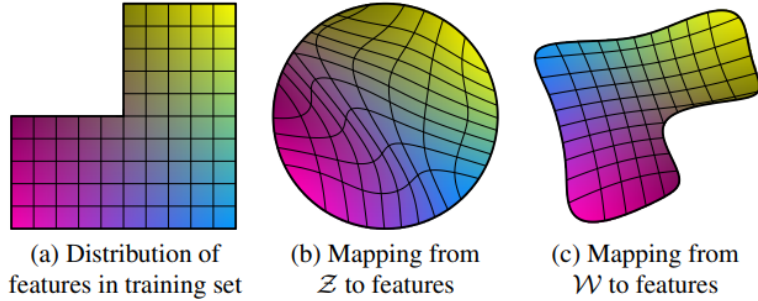


Figure 7: (a) Example of data with a gap, (b) which would result in a deformation of the space to fit a Gaussian distribution, (c) while the mapping network better capture the distribution (source)

But then, the w vector is not directly fed to the generator. Rather than using a randomly sampled vector, the generator starts from a constant one that is optimized during the training (i.e like a fixed seed). w is fed throughout the network via Adaptive Instance Normalisation (AdaIN) (see figure 8 (b)). AdaIN was originally created for style transfer and is defined as follow by Huang et al. [7]:

$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

where x is the content input and y is the style input. AdaIN simply normalize the channel wise mean and variance of x to match those of y

AdaIN is closely related to Batch Normalization, with the difference that, unlike BN, AdaIN does not have learnable parameters computed from batch statistics. Instead, it computes the affine parameters from the input style features stored in each channel. For example, one channel could be responsible for a specific brushstroke in the style image and is then transfer to the original image via normalization. In StyleGAN [4], the vector w has the role of the style input y after going through a learned affine transform (A in figure 8 (b)). The authors added another learned parameter B responsible for channel-wise scaling. This noise allows for more detailed training.

With those few but important changes, along with some other adjustments, StyleGAN became a reference in image generation. Most papers that followed, started from or compared themselves to StyleGAN. However, even though the incorporation of AdaIN is an ingenious idea that produces great results, its justification is still somewhat unclear. Indeed, it seems surprising that only a style transfer input could produce better images. Karras et al [4] even quoted “We find it quite remarkable that the synthesis network is able to produce meaningful results even though it receives input only through the styles that control the AdaIN operations”

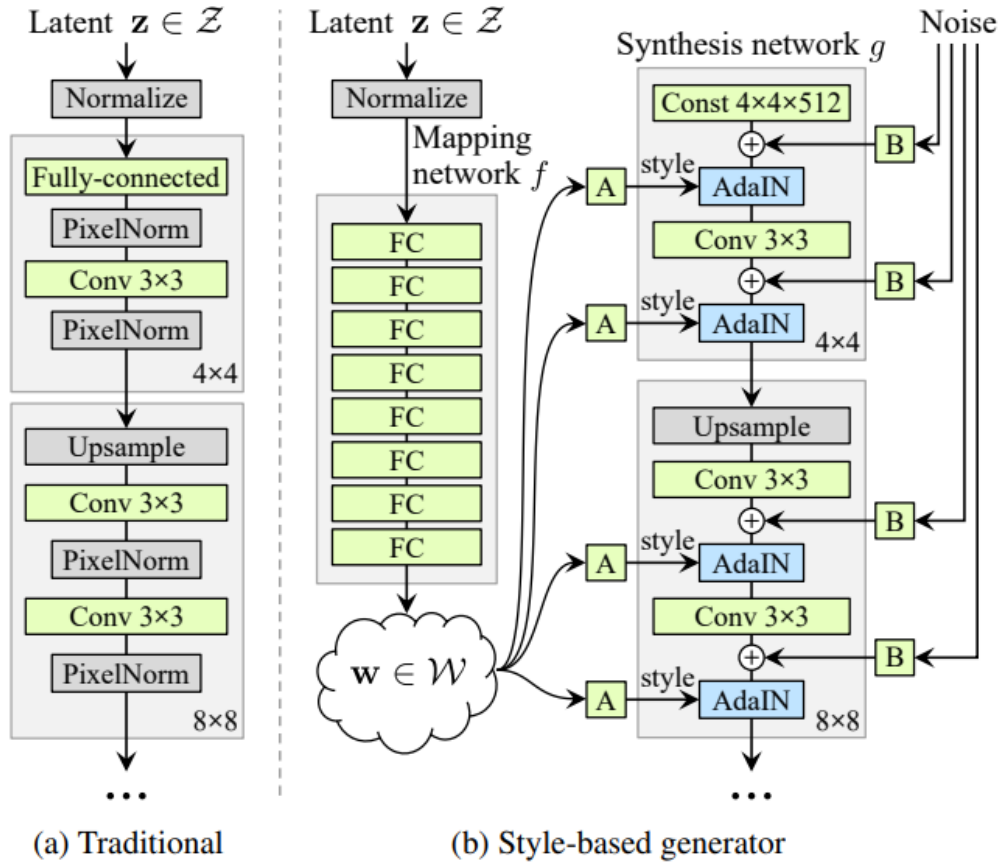


Figure 8: (a) Traditional GAN progressive growing architecture where the input vector z is sampled from a Gaussian distribution, (b) StyleGAN architecture with the mapping network f and the blending layer AdaIN ([source](#))

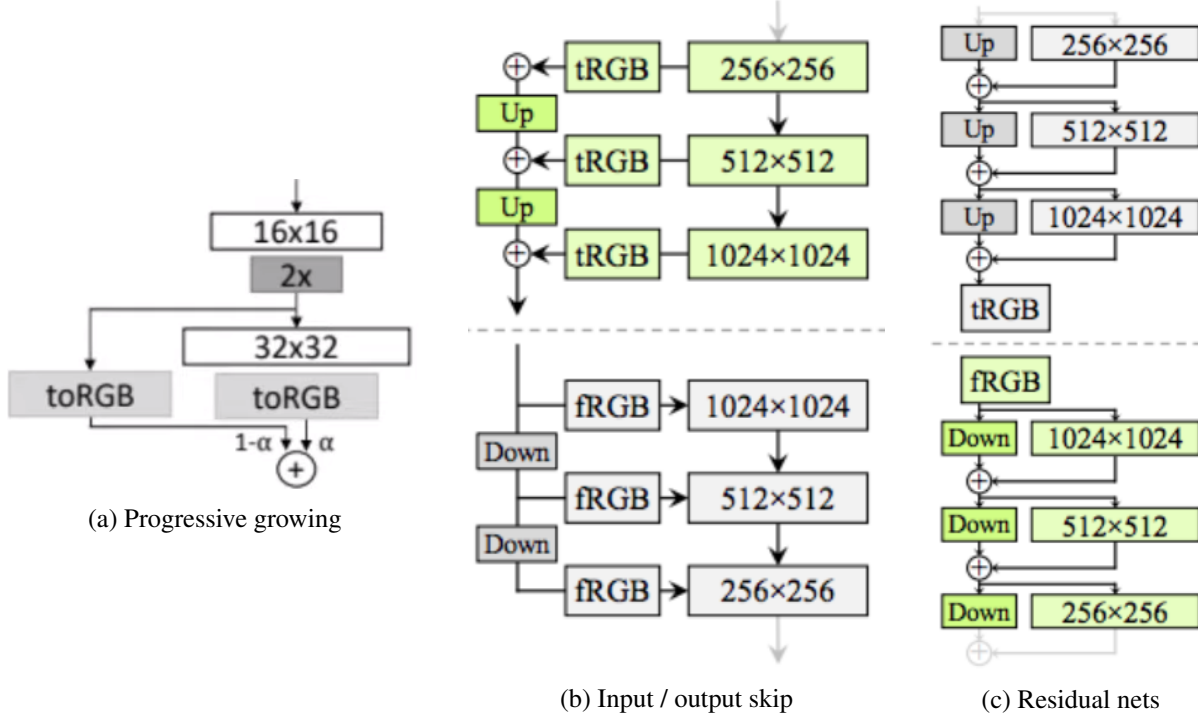


Figure 9: (a) Progressive growing example that needs a hyperparameter α to blend the two images, but has been replaced in the new version by two new methods: (b) input/ output skip and (c) residual nets. Even though the results are similar, they found out that the best configuration (in green) is input / output skip for the generator and residual nets for the discriminator ([source](#))

3.2 Version 2

Strong of their success, Karras et al decided to improve few elements of the first version and released a new paper: Analyzing and Improving the Image Quality of StyleGAN in 2019, also called StyleGAN2 [5].

One of the biggest changes was to modify the progressive growing architecture. According to them, it causing StyleGAN1 to have a location preference for facial features (i.e always having the eyes or the nose at the place). Inspired by the paper MSG-GAN [6], they introduced two methods: input/ output skip and residual nets (see figure 9). The first one first converts each feature map of high dimension to RGB images with convolution. It is then Up or Down sampled with a bilinear filtering operation to the next level. The second applies the upsampling directly on the feature map and only converts it to an image at the end.

The second aspect they wanted to improve the latent space interpolation: a small change in the latent vector should result in a small change in the generated image. A smoother latent space also facilitates projection onto it (i.e finding the latent vector corresponding to a given image). To do so, they added a perceptual path length regularization to the loss function of the generator. The regularization is defined as follow:

$$\mathbb{E}_{\mathbf{w} \sim \mathbf{W}, \mathbf{y} \sim \mathcal{N}(0, \mathbf{I})} \left(\|J_{\mathbf{w}}^T \mathbf{y}\|_2 - a \right)^2, \quad \text{with Jacobian matrix } \mathbf{J}_w = \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}$$

They also found out the regularization could be applied less often without reducing the overall quality, which is a great help considering the heavy calculations required to compute the Jacobian matrix. Thus,

they chose to only perform it every 16 step. This technique is called **lazy regularization**

Finally, they discovered as well that the previous version was not targeting the full 1024x1024 resolution (see figure 10), Instead, it was outputting sharp 512 images. To improve the quality and force the model to create real 1024² images, they increased the size of the network from 25 million parameters to 30 million by doubling the number of feature maps on higher resolutions.

With those few improvements, the Nvidia team reinforce the position of StyleGAN2 as a state-of-the-art model for GANs. Almost all the models we tried and will discuss in the following, are based on StyleGAN in some way. Most of them used the StyleGAN latent space and tried to improve either the encoding or the editing.

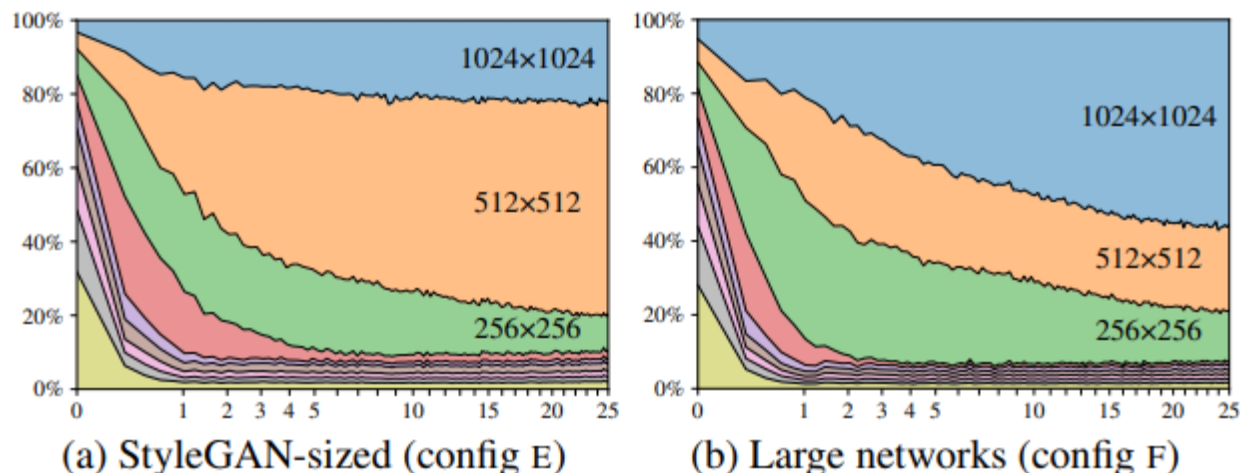


Figure 10: Contribution of each resolution on the final image. The y-axis is the relative standard deviation that measure how much each resolution weight at any given point. The x-axis is the number of images showed to the discriminator in millions. On the left the previous version and on the right the new one ([source](#))

4 Testing

4.1 Encoders

One important aspect of my research was to be able to modify the facial features of real persons, in particular aging. Thanks to GAN and its latent space, we know how to smoothly change specific features. However, one aspect that is often overlooked is how to encode an existing image in the latent space, also called **GAN Inversion**. Indeed, most papers that show impressive facial transformations, are using images already generated by the model. StyleGAN2 introduced a method to go from a generated image to its latent vector (i.e the point is the latent space that after going through the generator, output the desired image). However, finding a latent vector for real images is not as simple because we can only get an approximated version of it. One way could be of starting from a random vector in the latent space, compare the generated image with the real images with pixel-wise L2 loss and do gradient descent. Though a better comparison would be to first do a CNN (Convolutional Neural Networks) to efficiently capture the most important features of the image. However, even with CNN, when starting from a random position, the training is really slow. To speed up the process, we could generate a data set of faces beforehand quite fast to train a ResNet whose

goal is to efficiently find the respective latent vector of the generated images. We can now feed our initial image to the ResNet, have a good initial guess, and do gradient descent with the CNN to find a close image in the latent space.

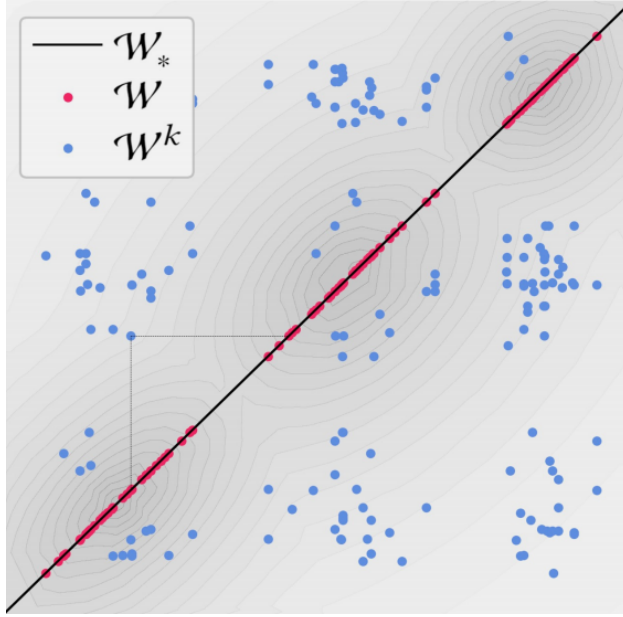


Figure 11: Example of a 1 dimensional style space with $k = 2$. W points are sampled from W , composed of 3 1D Gaussian distributions, W^k points are sampled from 2 points in W (1 for each axis). W_* is the black line and W_*^2 the entire gray region, that belongs in $\mathbb{R}^2$ (source)

What makes GAN inversion so hard is that we need to face two trade-offs at the same time. The first one is between **distortion** and **perception**: the best would be to have a small distortion (the real and generated images have a similar shape, posture...) and high perceptual quality (the generated image looks realistic). However, it is almost impossible to have both at the same time. The second trade-off is between **distortion** (again) and **editability**. Indeed, more accurate images are tempted to behave badly when modified. To explain those two trades off, we need to introduce the W_*^k space. Recall that in StyleGAN, we $z \in Z = \mathcal{N}(\mu, \sigma^2)$ sampled from a Gaussian distribution converted to a *style code* $w \in W \subsetneq \mathbb{R}^{512}$ via a mapping network. It has been proven that not every image can be inverted in a StyleGAN latent space. In order to facilitate the inversion, we can extend the space to a higher dimension and we denote $W_*^k \subsetneq \mathbb{R}^{k \times 512}$. This is done because w can control the normalization at different positions in the network to increase the expressiveness of the generator. The StyleGAN space has 18 different style inputs.

It is well-documented [8], that images in W_*^k have a lower distortion (i.e better quality), due to its higher dimension. Furthermore, images in W , because it's the space the model (StyleGAN2) was trained on, are easier to edit and have better perceptual quality.

- **e4e**: Encoder4Editing [8], or more formally called Designing an Encoder for StyleGAN Image Manipulation was released in 2021 by Tov et al. [8]. Among all the methods we tried for encoding, e4e yielded the best results by far, but still a bit short to be used in a production (see figure 13). The other aspect of e4e is that it tries to find a vector close to W to allow better editing.

Additionally to **Encoder4Editing** [8] we also test **ReStyle** by Alaluf et al. [10] and **TediGAN** by Xia et al. [9]. Some results can found in the appendix, see figure 18, but were not as good as e4e.

Overall, even though some results looked interesting, the quality was never high enough. Most methods could generate almost perfect human faces, but when trying to recreate famous faces, it always felt a bit short. We called it the sibling effect: we could recognize the person but it was not him. Which is not acceptable for professional films. One idea could be to train the model again with a lot of images of the person we wanted to recreate (which is quite easy considering it was famous people) but we first did not have time to try and was not even sure it would have worked.



Figure 12: For each pair of image, on the left: real image, on the right: image generated by e4e. Even if they look really similar, it was still far from being an exact replica

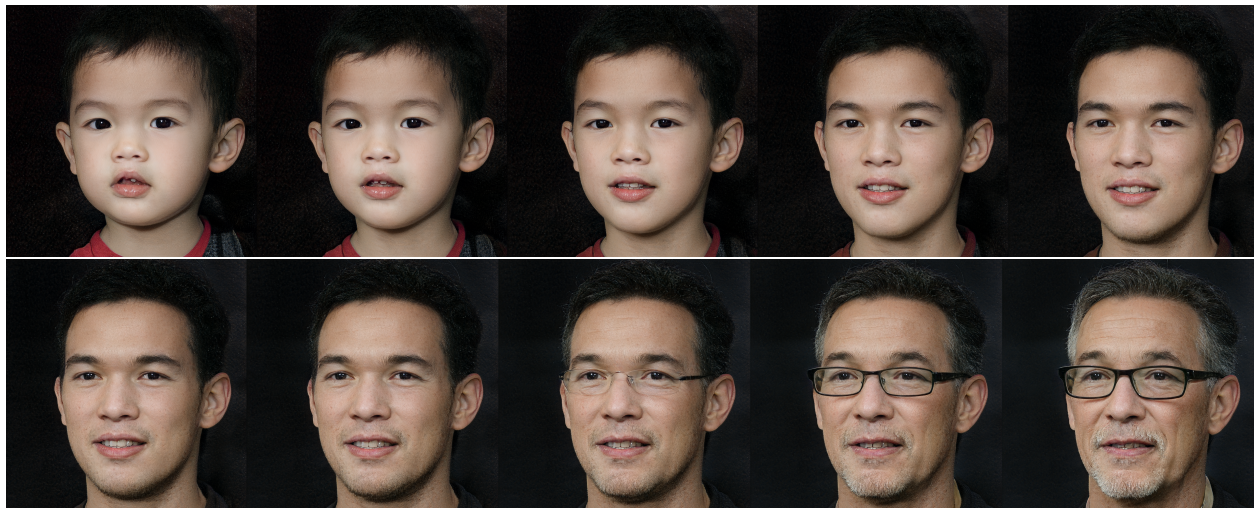


Figure 13: Example of aging editing with e4e ([source](#))

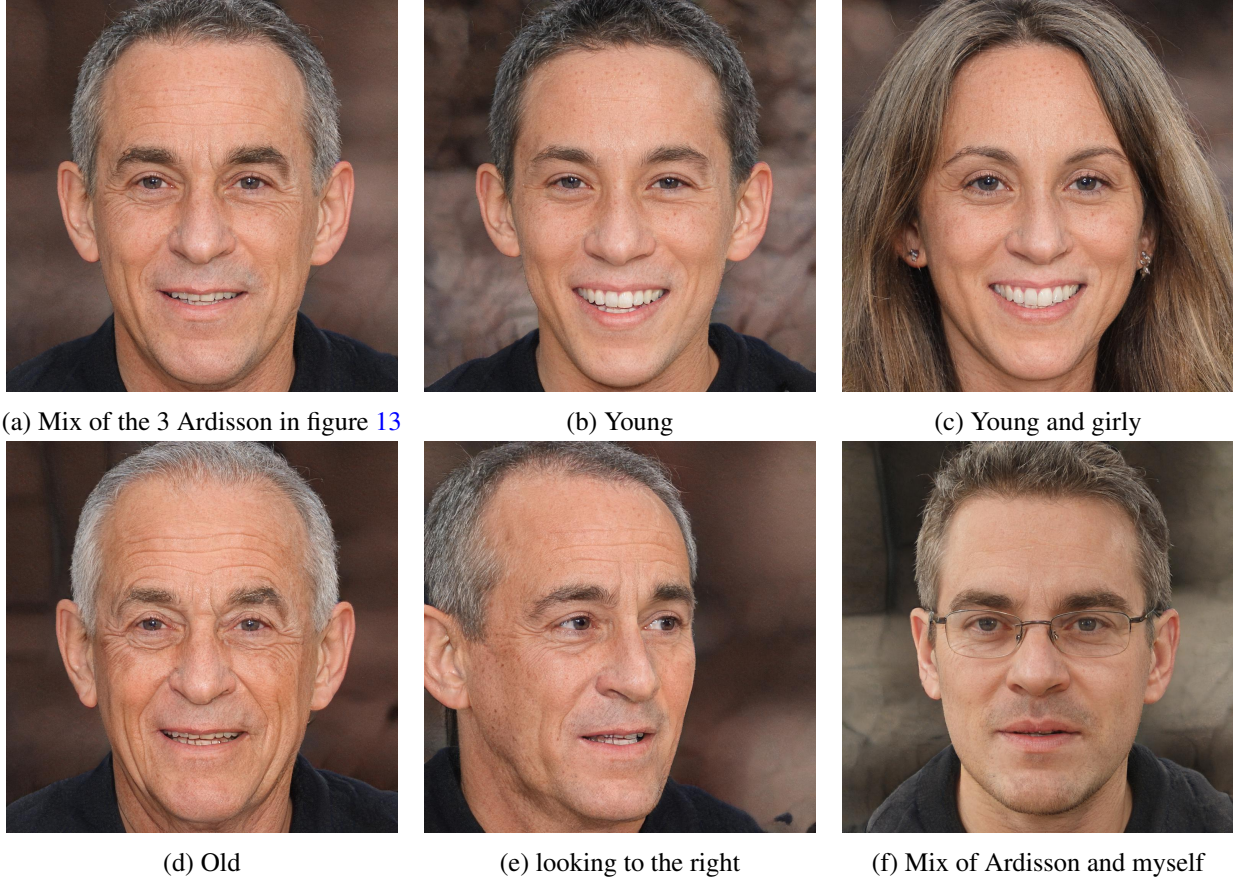


Figure 14: (a) Reference image generated by e4e with the 3 Ardisson picture presented in figure [8] (Top left, top right and bottom left), (b,c,d,e) transformations obtained with StyleFlow, (f) mix of all the 4 picture in figure [8]

4.2 Editing

After inverting our image (finding the right latent vector), we can now start the editing. To modify an image, we can simply move along the latent space. The hard part is to find in which direction we should move along, which is far from easy considering the high dimension (512) of the space. A simple method consists of generating new data of faces and doing simple binary classification according to a list of important features (e.g smiling/not smiling, old/young, woman/man...). This can be done manually but we now have a lot of machine learning classifiers that could do it automatically with a negligible error. Once we have classified our data, we can separate them in two using a linear hyperplane (for example young people on one side and old people on the other) and walk orthogonally to it. If the training of the GAN was done correctly, we should have smooth modification when going in a given direction. However, it is harder in practice to find the perfect direction as some features intersect each other. For example, if we want to add a bear, the model tends to make the overall face more masculine as those two features are generally correlated. The editing part has been well documented over the years and modern GANs can make incredibly realistic transformations. Nonetheless, the overall quality of real faces is still limited by the encoding part.

- **Style Flow:** Styleflow, released by Abdel et al. [11] is an editor based on StyleGAN generated images

and was the best one we tried. Additionally, to doing great transformations, StyleGAN had a lot of different attributes to choose from, 17 in total: gender, glasses, yaw, pitch (for face orientation), baldness, beard, age, expression (smiling or not), and 9 for lightning redirection. It also provided an interface that facilitates testing for non-developers people. As MacGuff is a cinema-oriented company, part of my job was to make tools easy to use. For example, I added buttons in the interface to process a list of images and apply the same modification to all of them. It let us try to experiment with video rather than just images. But as one could expect, the lack of temporal coherence made the result unusable. One idea we had to improve the resemblance was to mix several images of the same person. However, even though the overall quality was great, we did not gain in distortion compared to the real image.

- **Style Clip:** StyleClip [13] seemed like an interesting tool as it was using **Contrastive Language-Image Pre-training** (CLIP) models. In other words, they develop a text-based interface for StyleGAN image manipulation. If the idea is compelling and the results acceptable, the number of applications is quite limited. For example, it did not work with face orientation. Additionally, the installation was fairly complicated having to write the modification turned out not much useful.
- **EigenGAN:** The idea of EigenGAN was to find the main face attributes automatically using **Principal Component Analysis** (PCA). However, we did not find how to add our own inputs and the results could quickly become out of bound (see figure [12]).

4.3 Talking pictures

Another important aspect of my research was to be able to create talking pictures. The idea is to have a **source image** on one side, a **driving video** on the other, and replicate the movements of the video onto the image. In particular, we want to apply it to faces in order to make old picture talk. The main principle is to segment or attach key points to both the image and the video to be able to identify common parts. Once the motion is effectively estimated, a generative module create a new video.

- **First Order Motion Model (FOMM):** First Order Motion Model for Image Animation, by Siarohin et al. [17] has been the state of the art model concerning image animation. By recreating an entirely new video, the model was able to add elements that are not even on the source image (for example the back of a person turning around). This can be done thanks to an occlusion mask that tells which part in the video can be warped from the original image and which one should be recreated from the surrounding context. Even today, the talking faces result by FOMM are one of the best. However, it is still subject to one limitation: its resolution. For now, the official repository only provides 256x256 pre-trained weights. Unfortunately, 256x256 is just too small for **production** usage. Our first idea was to force the model to output a 512x512 video, which it did, but at the cost of the animation (see figure 15 (b)).

As it was an important project for the company, we trained the model again from scratch but this time in 512x512. However, the results were not as good as expected (see figure 15 (c)) both in terms of animation and image quality. The training data set was not given with the paper but they recommended two datasets: VoxCeleb and UvaNemo. The second one is a large data set of people smiling with good video quality (1980x1080) but we thought it would be too restrictive so we chose the first one. VoxCeleb is composed of thousands of celebrities' videos downloaded from Youtube. We followed the steps explained [here](#) to help us process the data. One thing we found out afterward

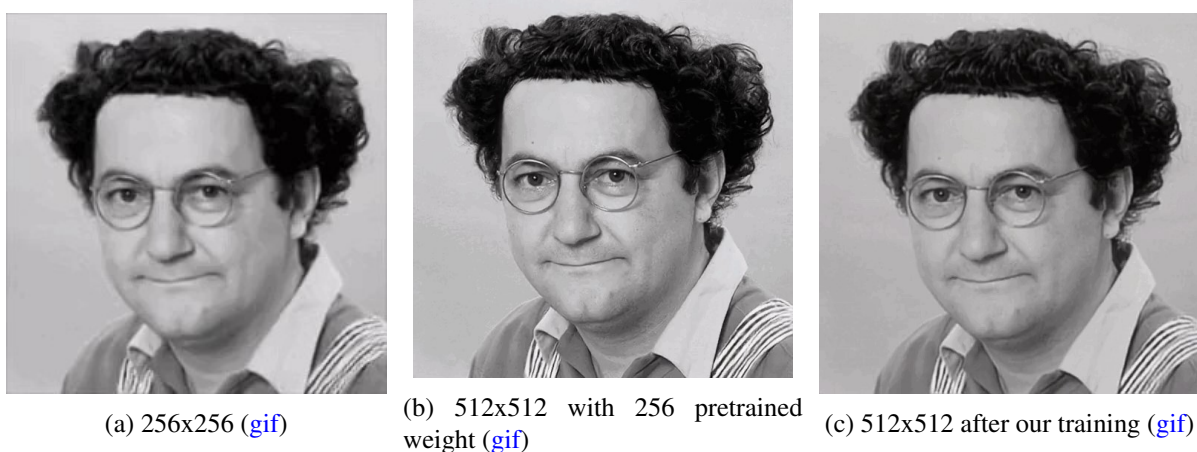
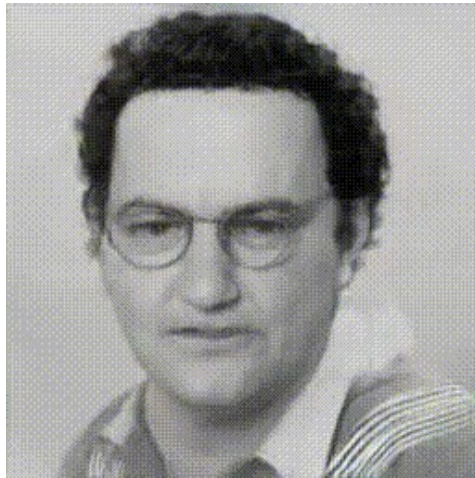


Figure 15: Same Coluche video with 3 different settings. On the left, the original weights given by Siarohin et al. [17], in the middle, our brute force 512x512 image and on the right, our 512x512 result after retraining

that Siarohin et al. [17] did but not us was to remove all videos that are below their required resolution. Instead, we only rescaled the videos to reach 512x512. Even if we are not sure our model failed due to this, it probably partially explains why their model trained on 256x256 images has better quality than ours trained on 512x512 images (with identical parameters). The second hypothesis concerns the batch size. Due to memory limitations, we had to reduce our batch size to only 2 while they used one of 36. Anyway, after getting more than 20 thousand videos for our data set, we started the training and tested several combinations: mainly changing the number of epochs, repeats, and facial key points. All our measure done during the training can found in the appendix (see figure 21, 22 and 23). Our final result, obtain after 300 hours of training is showed in figure 15 (c). We can see it has better animation than the 512x256 with still a huge issue for the mouth and image quality just a bit better than the 256x256.

- **Articulated Animation** Motion representation for articulated animation by Siarohin et al [18] was released in 2021 and is based on first-order model. The results looked great to animate very different objects and poses: they had demonstrations of animals drawing, TED talks presentations, people doing Tai Chi... The only negative point is that the model could only be used with videos belonging to the same environment as the training. We still tried to animate faces with the TED talk pretrained weights but the results were, as expected, quite bad (see figure 16). We also tested on people standing, close to the position of the training set, but it was still too far away from the original (see figure 17). Nonetheless, we could still get curious outcomes when mixing with the wrong training weights.
- **Motion co-segmentation** Motion-supervised Co-Part Segmentation by Siarohin et al. [19] allowed for partial swapping. By segmenting the face (or the body) in a more meaningful way, they were able to change local parts with precision, for example, the hair, or the lips. As we were not interested in part swaps, we selected the full face and test it with different numbers of segmentation. Although this is a good idea to explore, the results were not as good as First Order Model.



(a) (click to see the [gif](#))



(b) (click to see the [gif](#))

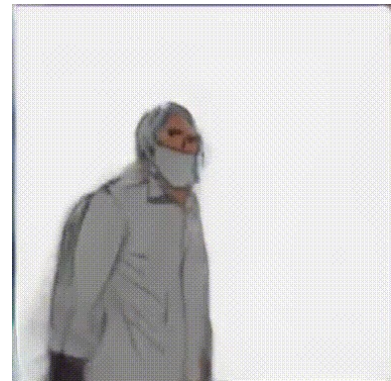
Figure 16: Talking picture of Coluche on the left and Johnny Hallyday on the right with Motion representation for articulated animation



(a) Driving video ([gif](#))



(b) Reference image drawn over the first frame



(c) Result video ([gif](#))

Figure 17: Test of Motion representation with someone walking. We can see that the model manage at the beginning but quickly lose tracking

4.4 Gaze Redirection

Another need was to be able to redirect the gaze on images. Indeed, most dataset cruelly lack pictures of people looking on extreme positions, thus limiting the output result. Several papers discussed gaze redirection with GAN but it is not a subject as discussed as face transformation for example. Moreover, most projects did not have pre-trained weight available, making them impossible to quickly test. We found three papers that related to our problematic: Controllable Continuous Gaze Redirection (or **InterpGaze**) by Xia et al. [14], Unsupervised Gaze Correction and Animation in the Wild (**Gaze Animation** in short) by Zhang et al. [15] and Photo-Realistic Monocular Gaze Redirection Using GAN (or **Gaze redirection**) by He et al. [16]. Gaze animation [15] yielded the worst result out of all three and could only recenter the eyes (which was the opposite of our needs). Gaze redirection [16] had the best results, it allows to move the eyes while also taking care of movements of secondary parts like cheeks. However, the training was apparently taking several months for some people so we decided to focus on Interp Gaze [14] instead. I first spent the first week trying to debug the code available on github. At the beginning, it was only outputting black images and could not read correctly our inputs. It was also supposed to do gaze redirection (as mentioned in the title) but I was only able to get interpolation. I decided to still retrain the model but once again, due to memory issues, I had to decrease the batch size from 64 to only 16, which could have affected the final results. After almost a week of training, the results obtained while running looked great but for some unknown reasons, the final outputs had a lot of noise. As it was not the most important project to work on, it was decided to leave it and move on.

Conclusion

Throughout my report, I explain the theoretical functioning of GANs (Generative Adversarial Network) and their usage in the industry, in particular in the field of visual effects. GANs, due to their adversarial nature are by essence unstable and challenging to train. However, when done correctly, they can produce unprecedented results in terms of image generation. They can be used for a wide variety of subjects and the focus of my work only explored a small amount of them, in particular GAN inversion to encode images in the latent space like **Encoder4Editing**, and GANs for image transformation such as **StyleFlow** and **First Order Motion Model**. Even if the results obtained were not as good as expected, mainly due to its complexity, GANs are a promising technology. A lot of research, in particular in VFX studios, is done across the world. They still require a lot of manual tuning but they already provide a solid base that keeps on improving. I truly believe that new software based on this technology will become an essential tool to the creative community.

Acknowledgements

I would like to thank the research and development team in particular Yann Le Pailh, the supervisor of the IA project and Vincent Champsiaux the director of the research and development team for helping me and answering all my questions and needs while also giving me a lot of freedom in my work. I would as well like to thank MacGuff and especially Rodolphe Chabrier for letting me do my internship with them.

References

- [1] Tim Salimans and Ian Goodfellow and Wojciech Zaremba and Vicki Cheung and Alec Radford and Xi Chen
Improved Techniques for Training GANs.
cs.LG, arXiv: 1606.03498, 2016
<https://arxiv.org/abs/1606.03498>
- [2] Ian J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu and David Warde-Farley and Sherjil Ozair and Aaron Courville and Yoshua Bengio
Generative Adversarial Networks.
stat.ML, arXiv: 1406.2661, 2014
<https://arxiv.org/abs/1406.2661>
- [3] Tero Karras and Timo Aila and Samuli Laine and Jaakko Lehtinen
Progressive Growing of GANs for Improved Quality, Stability, and Variation.
CoRR, arXiv: 1710.10196, 2017
<http://arxiv.org/abs/1710.10196>
- [4] Tero Karras and Samuli Laine and Timo Aila
A Style-Based Generator Architecture for Generative Adversarial Networks.
CoRR, arXiv: 1812.04948, 2018
<http://arxiv.org/abs/1812.04948>
- [5] Tero Karras and Samuli Laine and Timo Aila
Analyzing and Improving the Image Quality of StyleGAN.
CoRR, arXiv: 1912.04958, 2019
<http://arxiv.org/abs/1912.04958>
- [6] Animesh Karnewar and Oliver Wang
Multi-Scale Gradient GAN for Stable Image Synthesis (MSG-GAN).
CoRR, arXiv: 1903.06048, 2019
<http://arxiv.org/abs/1903.06048>
- [7] Xun Huang and Serge Belongie
Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization.
CoRR, arXiv: 1703.06868, 2017
<http://arxiv.org/abs/1703.06868>
- [8] Omer Tov and Yuval Alaluf and Yotam Nitzan and Or Patashnik and Daniel Cohen
Designing an Encoder for StyleGAN Image Manipulation.
CoRR, arXiv: 2102.02766, 2021
<https://arxiv.org/abs/2102.02766>
- [9] Weihao Xia and Yujiu Yang and Jing-Hao Xue and Baoyuan Wu
TediGAN: Text-Guided Diverse Face Image Generation and Manipulation.
CoRR, arXiv: 2012.03308, 2021
<https://arxiv.org/abs/2012.03308>

- [10] Yuval Alaluf and Or Patashnik and Daniel Cohen-Or
ReStyle: A Residual-Based StyleGAN Encoder via Iterative Refinement.
 CoRR, arXiv: 2104.02699, 2021
<https://arxiv.org/abs/2104.02699>
- [11] Rameen Abdal and Peihao Zhu and Niloy Mitra and Peter Wonka
StyleFlow: Attribute-conditioned Exploration of StyleGAN-Generated Images using Conditional Continuous Normalizing Flows.
 CoRR, arXiv: 2008.02401, 2020
<https://arxiv.org/abs/2008.02401>
- [12] Zhenliang He and Meina Kan and Shiguang Shan
EigenGAN: Layer-Wise Eigen-Learning for GANs.
 CoRR, arXiv: 2104.12476, 2021
<https://arxiv.org/abs/2104.12476>
- [13] Or Patashnik and Zongze Wu and Eli Shechtman and Daniel Cohen-Or and Dani Lischinski
StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery.
 CoRR, arXiv: 2103.17249, 2021
<https://arxiv.org/abs/2103.17249>
- [14] Xia, Weihao and Yang, Yujiu and Xue, Jing-Hao and Feng, Wensen
Controllable Continuous Gaze Redirection.
 CoRR, arXiv: 2010.04513, 2020
<https://arxiv.org/abs/2010.04513>
- [15] Jichao Zhang and Jingjing Chen and Hao Tang and Wei Wang and Yan Yan and Enver Sangineto and Nicu Sebe
Dual In-painting Model for Unsupervised Gaze Correction and Animation in the Wild.
 CoRR, arXiv: 2008.03834, 2020
<https://arxiv.org/abs/2008.03834>
- [16] Zhe He and Adrian Spurr and Xucong Zhang and Otmar Hilliges
Photo-Realistic Monocular Gaze Redirection Using Generative Adversarial Networks.
 CoRR, arXiv: 1903.12530, 2019
<https://arxiv.org/abs/1903.12530>
- [17] Aliaksandr Siarohin and Stéphane Lathuilière and Sergey Tulyakov and Elisa Ricci and Nicu Sebe
First Order Motion Model for Image Animation.
 CoRR, arXiv: 2003.00196, 2020
<https://arxiv.org/abs/2003.00196>
- [18] Aliaksandr Siarohin and Oliver J. Woodford and Jian Ren and Menglei Chai and Sergey Tulyakov
Motion Representations for Articulated Animation.
 CoRR, arXiv: 2104.11280, 2021
<https://arxiv.org/abs/2104.11280>
- [19] Aliaksandr Siarohin and Subhankar Roy and Stéphane Lathuilière and Sergey Tulyakov and Elisa Ricci and Nicu Sebe

Motion-supervised Co-Part Segmentation.
CoRR, arXiv: 2004.03234, 2020
<https://arxiv.org/abs/2004.03234>

Other sources

- <http://hunterheidenreich.com/blog/what-is-a-gan/>
- <http://hunterheidenreich.com/blog/gan-objective-functions/>
- advanced topic on objective functions: <https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-adversary-networks-819a86b3750b>
- <https://towardsdatascience.com/stylegan2-ace6d3da405d>
- <https://www.youtube.com/watch?v=dCKbRCUyop8t=1s>

Appendix

4.5 Proof

In this section, we want to prove that $C(G) = -\log(4) + 2 \cdot D_{JS}(p_d||p_g)$, with $C(G)$ the training criterion for the discriminator for any generator G and defined as:

$$C(G) = \max_D V(G, D) = \max_D \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \mathbf{d}x$$

Recall that for $(a, b) \in \mathbb{R}_+^2$, the function $f(x) = a \log(x) + b \log(1 - x)$ reaches a maximum between $[0, 1]$ at $\frac{a}{a+b}$

Thus, the optimal discriminator can be expressed as $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$

We also recall that

$$\begin{aligned} \text{Thus we have: } D_{JS}(p_{data}||p_g) &= \frac{1}{2} D_{KL}\left(p_{data}||\frac{p_{data} + p_g}{2}\right) + \frac{1}{2} D_{KL}\left(p_{data}||\frac{p_{data} + p_g}{2}\right) \\ &= \frac{1}{2} \int_x p_d(x) \log(2 \cdot D_G^*) \mathbf{d}x + \frac{1}{2} \int_x p_g(x) \log(2 \cdot (1 - D_G^*)) \mathbf{d}x \\ &= \frac{1}{2} \left(\log 2 + \int_x p_d(x) \log(D_G^*) \mathbf{d}x + \log 2 + \int_x p_g(x) \log(1 - D_G^*) \mathbf{d}x \right) \end{aligned}$$

$$\text{As } \int_x p_d(x) \mathbf{d}x = \int_x p_g(x) \mathbf{d}x = 1$$

$$\begin{aligned} &= \frac{1}{2} \left(\log 4 + \int_x p_d(x) \log(D_G^*) + \int_x p_g(x) \log(1 - D_G^*) \mathbf{d}x \right) \\ &= \frac{1}{2} \log 4 + \frac{1}{2} V(G, D_G^*) \end{aligned}$$

$$\text{Therefore } C(G) = -\log(4) + 2 \cdot D_{JS}(p_d||p_g)$$

4.6 Various results



(a) Reference image



(b) encoding with ReStyle



(c) encoding with TediGAN

Figure 18: Test of encoding with different methods. ReStyle is not too far from e4e, but the results we obtained with TediGAN are not usable



(a) Reference image encoded by e4e



(b) Image transform with StyleClip with only indication being "old face"

Figure 19: Test of transformation with Style CLIP



Figure 20: Two examples obtained with TediGAN. The middle image seems to be the reference and the one on the left and right obtained by going in the certain direction. However, as soon as we move too far from the original point, the results start getting weird

4.7 Analytical results for First Order Model training

Perceptual loss

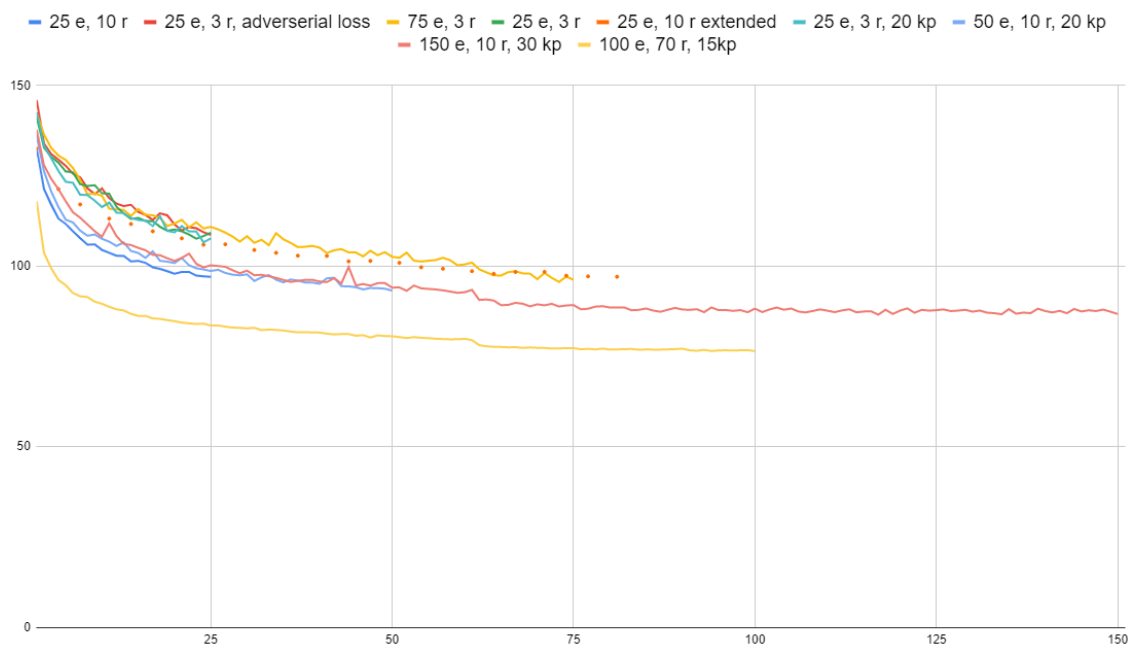


Figure 21: Plot of the perceptual loss w.r.t the number of epochs

Equivariance value

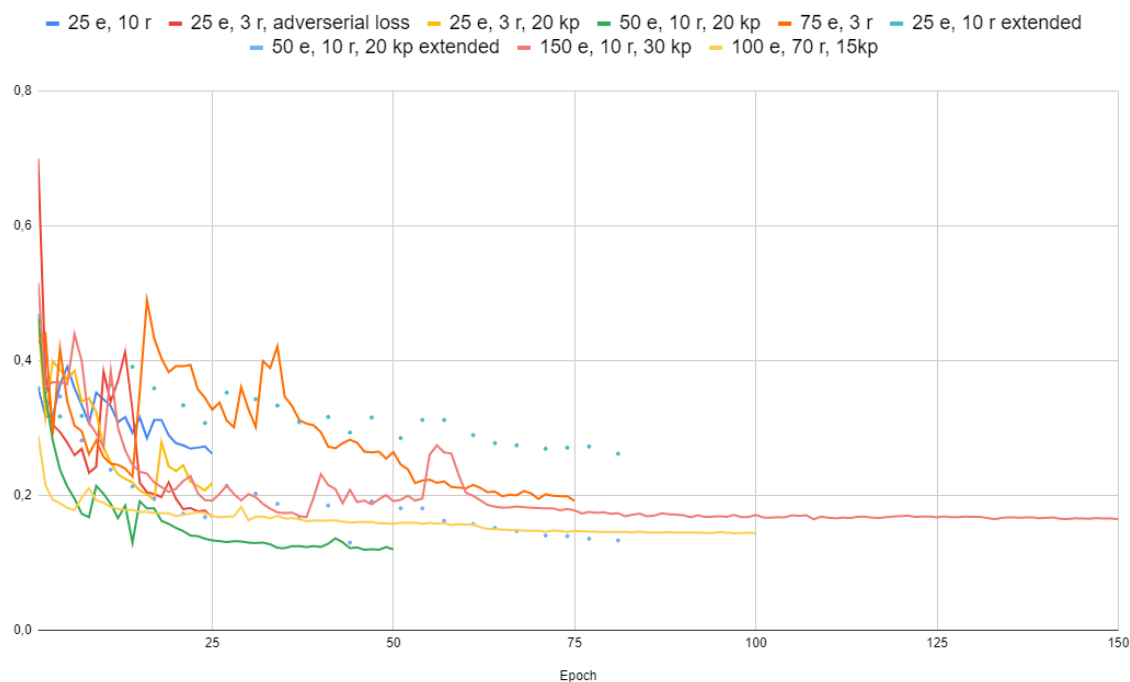


Figure 22: Plot of the equivariance value w.r.t the number of epochs

Equivariance jacobian

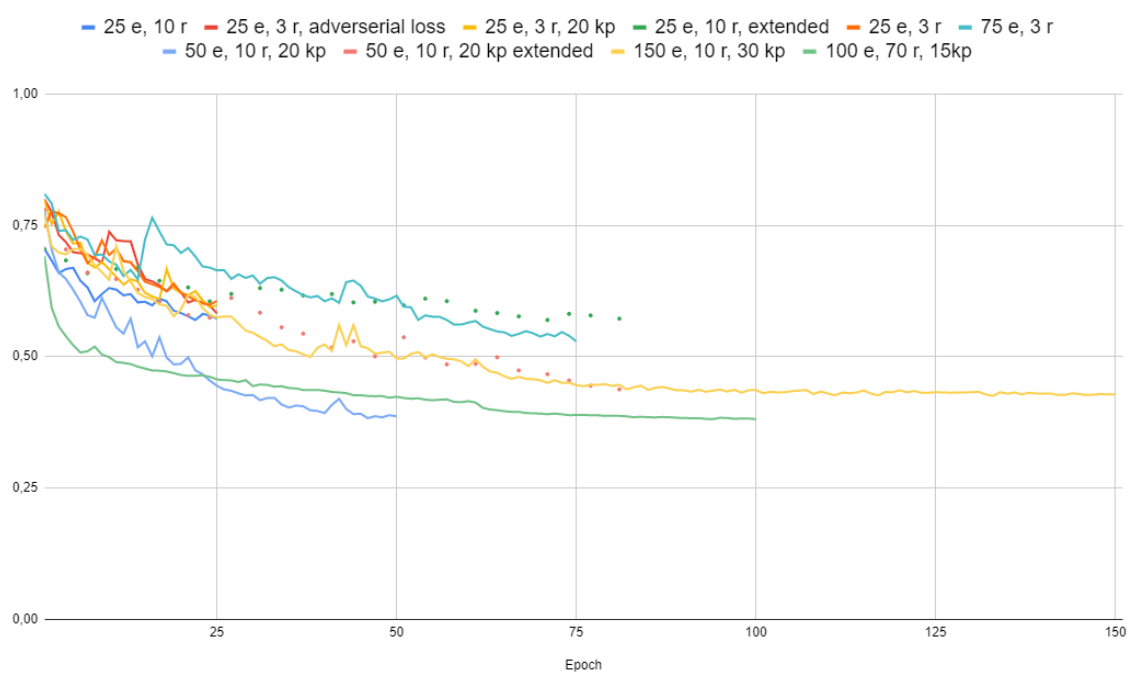


Figure 23: Plot of the equivariance jacobian w.r.t the number of epochs