UCLouvain

École polytechnique de Louvain

Deployment of a GAN architecture to improve the realism of an urban traffic network

Author: **Laurent Moutier**
Supervisor: **Benoît Macq**
Readers: **Jonathan Samelson, Victorien Sonneville, Christophe De Vleeschouwer**
Academic year 2022–2023
Master [120] in Computer Science and Engineering

Acknowledgement

I would like to express my deepest thanks and appreciation to my master thesis supervisor Professor Benoît Macq for giving me the opportunity to work under his kind supervision. Moreover, I would like to express my deepest gratitude to Jonathan Samelson for all his guidance, his endless support, and his valuable advice throughout the year.

I would also like to sincerely thank Victorien Sonneville, Benoît Gérin, Nikita De Broux, and Lucas El Raghibi for guiding me through this all year.

Finally, I take this opportunity to express my sense of gratitude to my parents and family, without whom all this will not have been possible. My sincere thanks for all their financial and unceasing moral support throughout all those years.

I sincerely hope you will enjoy reading this master thesis or at least provide you with some insightful information about Generative Adversarial Networks.

Abstract

In recent years, supervised learning has made significant results in the domain of computer vision, progressively changing our daily life with the help of Convolutional Neural Networks. Comparatively, unsupervised learning has received less attention, yet the invention of Generative Adversarial Network in 2014 provided a new attracting alternative to traditional representation learning. In this master thesis, we will first try to understand the original GAN structure, its improvements with WGAN-GP, and its limits, and successfully propose qualitative generated samples of size $(128, 128)$ for classical data augmentation problem. About its limitations, we show the mode collapse and computation time problems such networks have. We then tackle this work's primary goal, i.e., deploying such architecture to improve the photorealism of images taken in a virtual urban traffic environment. We try to run a specific GAN that modifies the typical architecture to take as input metadata available from the game engine, pixel-level annotations, and the normal synthetic data. Even though we failed to make such a visual shift due to problems with the annotation, we propose and describe several potential solutions for future work, including CycleGAN and its synthetic-to-photorealistic data translation version CyCADA.

Contents

1 Introduction **3**

2 Theoretical Backgrounds **5**
2.1 Generative Adversarial Network 5
2.1.1 Generator and Discriminator Architecture 8
2.1.2 Problems with GAN 10
2.2 Related work 10
2.2.1 Map the latent space to the data domain 11
2.2.2 Change the discriminator 12
2.3 Wasserstein-GAN 13
2.4 Improved training of WGAN 14
2.4.1 Generator and Critic Architecture 15
2.5 GAN in enhanced photorealism 16
2.5.1 Gbuffer Encoder 18
2.5.2 Image Enhancement Network 20
2.6 Alternatives to enhanced photorealism 21
2.6.1 CycleGAN 22
2.6.2 Cycle-Consistent Adversarial Domain Adaptation 25
2.7 Summary 27

3 Metrics used in GANs **29**
3.1 Inception Score 29
3.2 Fréchet Inception Distance 30
3.3 Other Metrics in GANs 30

4 Methodology and tools **31**
4.1 Tools used 31
4.2 Datasets used in GAN 32
4.3 Datasets used in WGAN-GP 33
4.4 Retrieval of data for the EPE-GAN architecture 34
4.4.1 Extracting the G-buffers 34

4.4.2 Cleaning the screenshots and reordering the data 35
4.5 Performing the Segmentation 36

5 Experiments and Results **38**
5.1 Results with GAN 38
5.2 Results with WGAN-GP 41
5.3 Results for enhanced realism 45

6 Discussion and perspectives **46**
6.1 Introspectives 46
6.2 Questions 47
6.3 Future works 47

7 Conclusion **49**

A Annexes : Theoretical Part **55**
A.1 Algorithm of the GAN framework 55

Chapter 1

Introduction

Since John McCarthy in 1956, artificial intelligence has gone through multiple waves of optimism followed by disappointments. In the last decades, deep learning, a part of the broader family of machine learning methods, made some pivotal breakthroughs in domains such as image classification [39], natural language processing [32], speech recognition [1], and computer vision [46]. The potential scale of deep learning's impact on society with applications across many industries led to astonishing investments from tech giants like Amazon, Google, Meta Plateforms, and Microsoft. From the United States Department of Commerce's estimation [2], the global artificial intelligence market is expected to grow from an estimated $ 157 billion in 2020 to $15 trillion by 2030.

While deep learning seems to be the hot topic for machine learning algorithms, according to McKinsey [6], we can divide this domain into three categories: supervised learning, unsupervised learning, and semi-supervised learning. In this master thesis, we will focus on the unsupervised learning category and, more precisely, on the generative Adversarial Networks entitled as "the most interesting idea in the last ten years in machine learning" [24] by Yann Lecun, Facebook AI research director. This type of neural network enables one to produce images never seen before by imitating the style of a given dataset.

By definition, unsupervised algorithms will look for patterns in a dataset without pre-existing labels. To get the dataset, we were given access to the Digital Twin Project, a 3D game project realized with the Unreal Engine software. In this project, automated cars pass a crossroad following basic traffic rules. The goal of this master thesis is to take images from Digital Twin, pass them through a GAN architecture, and transform them into photo-realistic images.

To structure this work, we will first give an overview of the famous vanilla GAN

to make a data augmenter for Digital Twin images. We will then try to apply a GAN-based algorithm to improve the photo-realism of the synthetic images from the Unreal Engine project.

Chapter 2

Theoretical Backgrounds

This section will overview the mathematical concepts and architectures we need to know to understand all the frameworks used. At first, we will see the architecture of a Generative Adversarial Network (GAN). To continue, we will explain the issue of mode collapse and how to solve it using Wasserstein GAN (WGAN). In addition, we will discover some significant improvements we can make to WGAN to stabilize the network using gradient penalties (WGAN-GP). Finally, we will see an architecture that can transform synthetic images into real images and its alternatives.

2.1 Generative Adversarial Network

In 2014, Ian J. Goodfellow et al. [16] created a new field in machine learning for unsupervised learning: Generative Adversarial Network, which we will called GAN throughout this master thesis. While supervised learning requires more human preparation due to the labeling process, the unsupervised learning field contains algorithms to analyze and cluster unlabeled informations according to patterns, similarities, and differences without any prior knowledge of the data.

In the original framework, we have on one side a generative-oriented deep neural network G (called generator) trained to fool an adversary deep neural network D, called discriminator. On the other side, this network D aims at learning whether the generator created the proposed sample or if it is a real sample from a given dataset. In the beginning, the generator only receives as input a random vector that will try to match a plausible real data sample throughout the iterations. Throughout this master thesis, we will call the space from which the input comes the z-space or latent space while a sample from this space, i.e., the generator's input, is a noise. By introducing this noise, we can get the generator to produce a wide variety of data.

An important aspect is that both generator and discriminator learn simultaneously and try to adapt from each other, as shown in figure 2.1. Throughout this paper, we will call this training a *minmax* training as both generator and discriminator compete against each other. Moreover, this name *minmax* arises because each model minimizes the maximum payoff for the other and thus minimizes their maximum loss since the game is a zero-sum.

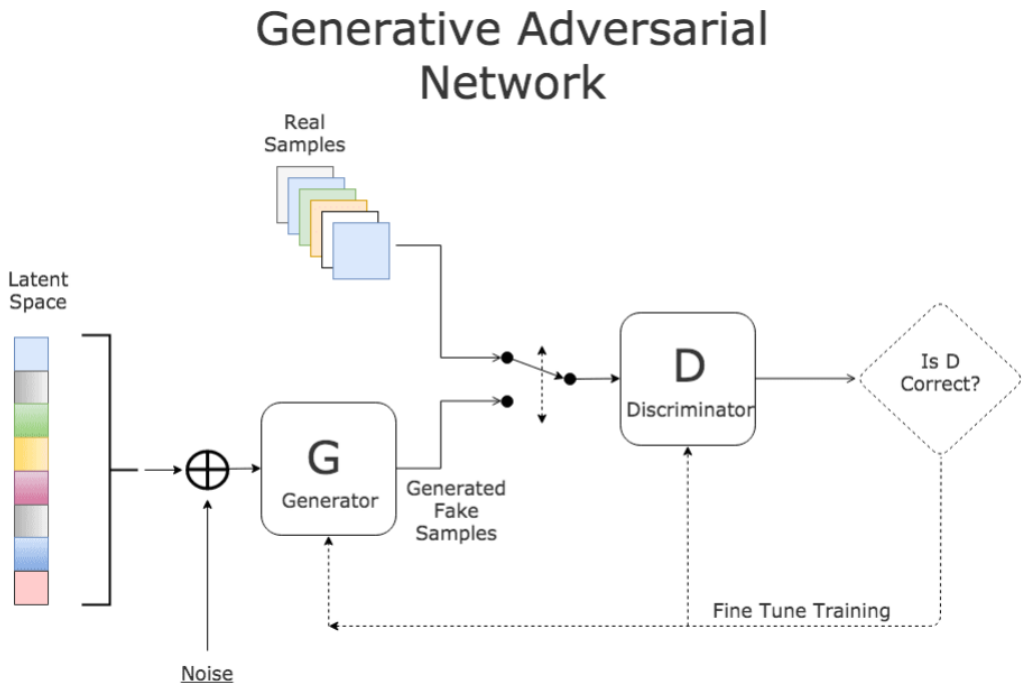


Figure 2.1: Architecture of a GAN [47] (Image source : KDnuggets [15])

Intuitively, we want to be able to analyze, capture and copy the variations within a given dataset. In order to achieve such a feat, we will need to find a vector representing the direction of the real data distribution. As we will see later on section 3, more or less pertinent metrics were created to evaluate how well a model performs. Nevertheless, it is worth mentioning that the objective evaluation of GAN generator models remains an open problem.

The duality between generator and discriminator can be formally defined. From a given dataset, we want to approach p_{data} the probability distribution of its samples with p_g, the probability distribution of the samples created from the generator G. This multilayer perceptron will receive as input the noise variables $p_z(z)$ and

outputs a vector x representing an image. Here, $G(z)$ corresponds to this data space mapping. Parallelly, we define the second multilayer perceptron $D(x)$ that represents the probability that the vector x comes from the data rather than p_g. We then train D to maximize the probability of assessing the labels correctly while G has to minimize $log(1 - D(G(z))$.

The first equation in Ian's paper [16] gives us a first criterion both networks have to follow to try to maximize their gains :

$$\min_G \max_D \mathcal{L}(G, D) = \quad \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[log(D_G(\mathbf{x}))] + \quad \mathop{\mathbb{E}}_{\mathbf{x}\sim p_g}[log(1 - D_G(\mathbf{x}))] \tag{2.1}$$

This equation is called an adversarial loss and can be solved for $D_G(\mathbf{x})$ using simple mathematical tools. Indeed, from their **Proposition 1**, we have :

For G fixed, the optimal discriminator D is

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})} \tag{2.2}$$

Proof. For any $(a, b) \in \mathbb{R}\backslash 0, 0$, we have that the function

$$f(x) = a \ log(x) + b \ log(1 - x)$$

achieves its maximum at $\frac{a}{a+b}$ between 0 and 1. As we work with probabilities, we will not exceed this interval. □

Finally, the second proposition in the paper gives us the final criterion for the minimax training:

If G and D have enough capacity, and at each step of Algorithm 1 [1]*, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\min_G \max_D \quad \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[log(D^*_G(\mathbf{x}))] + \mathop{\mathbb{E}}_{\mathbf{x}\sim p_g}[log(1 - D^*_G(\mathbf{x}))] \tag{2.3}$$

then p_g converges to p_{data}.

[1] see Appendix A.1

2.1.1 Generator and Discriminator Architecture

After understanding how a generative adversarial network works, we must implement it. In this section, we will detail the structure of the vanilla GAN we created with the help of an article [21] from A. Fernàndez Jauregui, a senior data scientist, and professor in the university of Deusto, Spain. We used Python and the free library Keras to make the layers of the machine learning algorithm.

Generator

Firstly, in order to make a deep neural network that can output a fake image from scratch, we need to load functions from Keras:

- **Dense**: This layer is the first one of the network and serves as a noise layer for our generator.
- **BatchNormalization**: This function applies a transformation that maintains the mean output near 0 and the output standard deviation near 1. It serves to normalize the input.
- **LeakyReLU**: It is the leaky version of a Rectified Linear Unit. Even though the unit is not active, it will still leak a small gradient depending on the argument **alpha**, the negative slope coefficient. We do not use simple ReLU functions to tackle the *dying ReLU problem* [28] that can appear in deep neural networks. It refers to the problem when ReLU neurons become inactive and only output zero for any output.
- **Reshape**: This layer will help us form a three-dimensional array (i.e., an image) out of the dense layer's single-dimensional vector.
- **Conv2DTranspose**: The two-dimensional transposed convolutional layer enables us to convolve backward, meaning we want to upsample and generate an output feature map with higher spatial dimensions than that of the input feature map. One should note that this layer differs from a deconvolution layer even though the output's spatial dimensions are the same. Here, we do not reverse the operation of a standard convolution and thus do not get back the original output. Both can be used in a GAN architecture.
- **Conv2D**: This layer applies a two-dimensional convolution and will help us have a three-dimensional tensor representing the fake image. It is the last layer before passing the output to the discriminator.

At the end of the generator network, we need to pay attention to having the same shape between the generated sample and the real image from the dataset, as it will serve as input to the discriminator network. In order to manage the shapes inside the network, three parameters of the transposed convolutional layer help us know the layer's output. For a given size of the input $\boldsymbol{i}$, kernel $\boldsymbol{k}$, padding $\boldsymbol{p}$, and stride $\boldsymbol{s}$, the size of the output feature map $\boldsymbol{o}$ generated by a transposed convolutional layer is given by the formula :

$$o = (i - 1) \times s + k - 2p \tag{2.4}$$

while with the same input, we have the following formula for the output $\boldsymbol{o}$ of a standard convolutional layer :

$$o = \frac{i + 2p - k}{s} + 1 \tag{2.5}$$

By convention, after the dense layer and its reshaping, we used four blocks that consist of a Conv2DTranspose, a BatchNormalization, and a LeakyReLU layer. Later on, we try to suppress the BatchNormalization layer without changes. The last layer is a Conv2D layer with a tangent activation function $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ to get the output values between the interval $[-1, 1]$. We use non-linear activation functions in deep neural networks to approximate better physical world phenomenon, which rarely follows linearity. More specifically, we prefer the tangent one to normalize the output image instead of the sigmoid function $\mathrm{S}(z) = \frac{1}{1+e^{-z}}$ that gives values in the $[0, 1]$ range. Indeed, its output is zero-centered, and it has a stronger gradient, as explained in this paper [14].

Discriminator

The discriminator network can be seen as a normal convolutional neural network even though certain modifications have been made. To build this neural network that takes as input an image and outputs a binary response, we need to load other functions from Keras :

- **Dropout**(**rate**): This layer randomly sets inputs to 0 with a frequency of **rate** at each step of the training. As stated by a paper from Nitish Srivastava et al. [43], this should prevent overfitting.
- **Flatten**: This layer serves to flatten the output before we give it a binary response.

In the same way than with the generator, we will use four blocks with each a Conv2D, a LeakyReLU, and a Dropout layer. Finally, we add a Flatten layer and

a Dense layer with a sigmoid activation function to get the binary response. The optimizer we used here is the Adam, with a lower learning rate and lower beta than the default values. Indeed, from the paper [37] written by Alec Radford et al., they tuned their hyperparameters and proposed 0.0002 for the learning rate while reducing the beta to 0.5 helped stabilize the training. After some tries, we also decide to definitively suppress the Dropout layer due to the result not being impacted by the suppression.

2.1.2 Problems with GAN

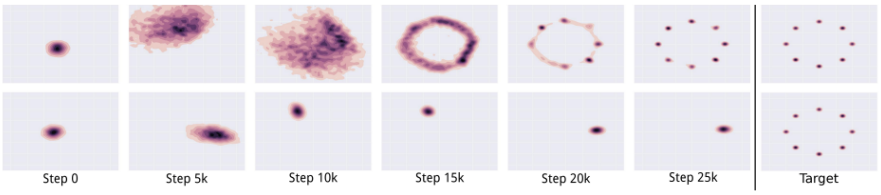
One of the main issues of GAN is its difficulty in training simultaneously two neural networks. Accordingly to the principle of a zero-sum game, improving the generator means lower our expectations on the discriminator and vice-versa, which is terrible for the performance. Moreover, it takes more time to train two neural networks than one, so the computation time can be a big factor depending on the machine we have access to. Furthermore, like any deep learning model, GAN has to face the non-convergence and instability challenges.

Finally, as observed by GAN's paper [16] and highlighted later in the Unrolled GAN's paper [31], a last inherent problem in the use of this type of network is the *mode collapse* problem. One of the goals we want to achieve with a GAN model is obtaining a generator able to produce a wide variety of new samples, and during training, the generator will indeed need to try to find the sets of samples that seem the most plausible to the discriminator. However, when the generator produces a small set of outputs at every iteration, the discriminator's best strategy should always be to learn to reject the generator's propositions. The *mode collapse* problem appears when the discriminator is stuck in a local minimum and cannot learn that strategy. Over-optimized, the generator can then easily fools the discriminator by rotating through small sets of output types.

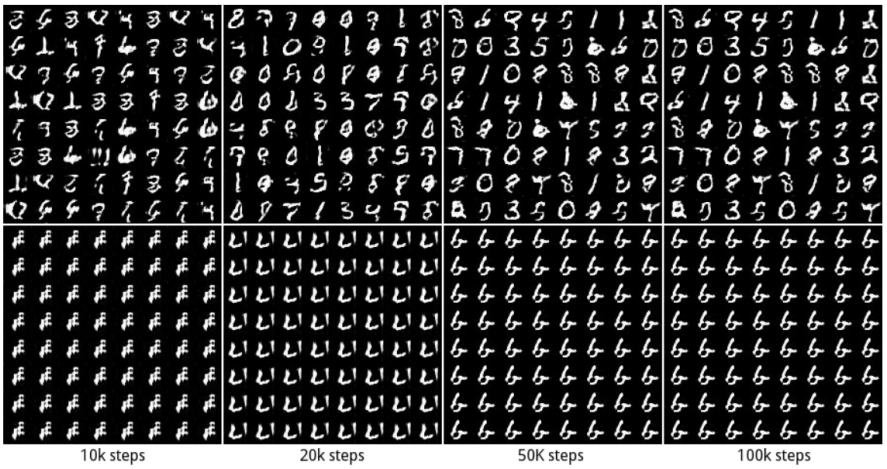
Furthermore, besides not generating diverse data samples, the found distribution can sometimes make no visual sense from a human point of view. To see the impact of such failure on the generation of images, we show below an example 2.2 from the Unrolled GAN's paper [31] with the MNIST dataset.

2.2 Related work

In order to tackle the *mode collapse* problem, several solutions exist and we will categorize them into two groups [5].



(a) On the rightmost image, we can see a heatmap of the source dataset's distribution with ten nodes as the numbers in the MNIST dataset vary between zero and nine. The first row presents what an Unrolled GAN model achieved, while the second row shows the impact of mode collapse on a vanilla GAN model.



(b) The first row shows the evolution of samples made by an unrolled GAN model. We can note a net stabilization throughout the iterations while maintaining a diverse pool of examples. In the second row, we can see samples made from a vanilla GAN model that suffers from the mode collapse problem. All the samples seem similar and do not represent anything from a human perspective.

Figure 2.2: Example of the mode collapse problem with the Unrolled GAN architecture (Image source : Unrolled GAN paper [31]).

2.2.1 Map the latent space to the data domain

On one side, we can try to learn to map the latent space to the data domain, and such a solution has been proposed in the Adversarial Feature Learning paper, more commonly called BiGAN's paper[2]

[2]One can note that the same model has been proposed independently in a concurrent paper, Adversarial Learned Inference [13], with similar conclusions.

Adversarial Feature Learning In BiGAN's paper, the generator possesses two components: the decoder $G(\mathbf{z})$ that maps the prior $p(z)$ (a source of noise) to the input space in the same way than in a vanilla GAN model and its inverse the encoder $E(\mathbf{x})$, that maps data samples x into the $\mathbf{z}$-space. The discriminator will then have to distinguish between samples created from the pair $(\mathbf{x}, \hat{\mathbf{z}} = G_z(\mathbf{x}))$ from the encoder and samples from the pair $(\hat{\mathbf{x}} = G_x(\mathbf{z}), \mathbf{z})$. We can see the network structure in figure 2.3. Unfortunately, as shown in this analyst paper [5] it can not avoid either training instability or a tradeoff issue between image diversity and image quality.

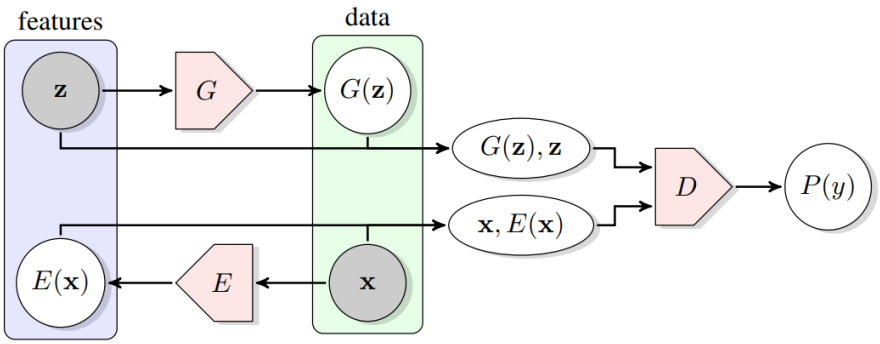


Figure 2.3: Architecture of a Bidirectional Generative Adversarial Networks (BiGAN) (Image source : BiGAN's paper [12]).

2.2.2 Change the discriminator

On the other side, the approach tends to change the discriminator, whereas papers such as WGANs [3], Unrolled GAN [31] or LSGAN [30] can propose different solutions to solve mode collapse.

Unrolled Generative Adversarial Networks The main issue with the mode collapse problem rests in the discriminator being trapped in a local minimum and not being able to pass to the next generation of discriminator the information to change in the strategy's direction. An unrolled GAN model [31] tackles this issue by unrolling the objective function for the K previous steps. By taking into account, the preceding gradients of both generator and discriminator, the next generations of discriminator will then be able to learn their way out of the local minima.

Wasserstein Generative Adversarial Networks In the WGAN's paper [3], the main issue behind mode collapse can is solved by changing the role of the

discriminator in the architecture. Indeed, the binary response given back to the generator network does not contain much information to reduce the distance between the distribution of the latent space and the actual distribution of the data. In order to resolve the mode collapse effect, the discriminator will use a new metric to compare samples that will help guide the generator in the same direction as the real data distribution.

Least Squares Generative Adversarial Networks In the vanilla GAN shown in a previous section 2.1.1, we adopted the sigmoid cross entropy activation function as the output of our discriminator. As stated in the LSGAN's paper [30], this loss function causes the mode collapse problem as it does not give enough information for the samples correctly classified yet far from the real data distribution. A possible remedy to the problem can be to change the objective function with a least squared approach :

$$\begin{aligned}
\min_D \quad & V_{\text{LSGAN}}(D) = \frac{1}{2}\,\mathbb{R}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[(D(\mathbf{x})-b)^2\right] + \frac{1}{2}\mathbb{R}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[(D(G(\mathbf{z}))-a)^2\right] \\
\min_G \quad & V_{\text{LSGAN}}(G) = \frac{1}{2}\,\mathbb{R}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[(D(G(z))-c)^2\right]
\end{aligned}$$

Where a, b are respectively the labels for fake and real data and c denotes the value G wants D to believe for fake data. The idea behind using a double minimization problem instead of a minimax problem is to separate the training of the two sets of parameters. When we update the generators, the parameters for the discriminator are already fixed, which means the decision boundary is fixed. As long as this boundary reflects the distribution of the real data, penalizing the correctly classified samples lying far away from this boundary will help the generator generate samples towards it.

Even though each architecture has its pros, in the following sections, we will try to solve the mode collapse problem with WGAN as it is one of the most promising.

2.3 Wasserstein-GAN

According to the original WGAN paper, the generator winning a turn does not correlate with an actual reduction in the distance between the two distributions. The zero-sum game ties discriminator and generator together and is one reason behind this mode collapse problem. In order to improve the generator, we no longer give it a binary response but rather a value of its actual distance to a real sample. As it no longer classifies the two samples, we replace the discriminator with a critic,

and the new metric to compare samples is the Earth Mover distance (Also called Wasserstein-1) between p_g and p_{data}.

$$EM(\mathbb{P}_{data}, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_{data},\ Pe_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|], \tag{2.6}$$

One of the main advantages of this distance is that it is continuous everywhere and almost differentiable everywhere. To know in which direction the generator needs to go, we need to modify the last criterion :

$$\min_G \max_{D\in\mathcal{D}} \quad \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}}[D(x)] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_g}[D(y)] \tag{2.7}$$

where $\mathcal{D}$ is the set of all 1-Lipschitz functions. As a reminder, a function f is a M-Lipschitz if there exists a real constant $M \geq 0$ such that for all x and y, we have :

$$|f(x) - f(y)| \leq M|x - y|$$

In our case, M is here equal to one. In order to enforce this Lipschitz constraint on the critic, the paper proposes to clip its weights within a compact space $[-c, c]$ with c the clipping parameter. Unfortunately, this could lead to vanishing gradients or a too heavy computation time for weights to reach their limits if c is too small or too large. Nevertheless, by guiding the generator in the same direction as the real data distribution, we can solve the mode collapse problem and create more diverse samples.

2.4 Improved training of WGAN

Since weight clipping was a temporary solution to an open problem, it can lead to massive drawbacks in terms of optimization if poorly parameterized. To correct this weakness, Ishaan Gulrajani et al. [17] proposed an alternative solution to weight clipping to enforce the Lipschitz constraint on the training objective. In this proposition, the gradient norm of the critic function will be constrained with respect to its input. Throughout this master thesis, we will call this variation WGAN-GP.

A differentiable function is 1-Lipschitz if and only if everywhere its domain is defined, the gradients are of norm less or equal to one. Using this property, we will choose particular points of the input space where we will evaluate the gradient of the critic $\nabla_{\hat{x}} D(\hat{x})$. Subsequently, we will penalize its squared distance from 1 in the critic loss function to enforce the 1-Lipschitz constraint. To summarize, the paper proposes a soft yet easier tractable version of the constraint by considering a

subset of points rather than the whole input space. The new criterion we will want to optimize takes the form of :

$$\min_G \max_D \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}}[D(x)] - \mathop{\mathbb{E}}_{\mathbf{y}\sim\mathbb{P}_g}[D(y)] + \lambda \mathop{\mathbb{E}}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\| \nabla_{\hat{x}} D(\hat{x}) \|_2 - 1)^2] \tag{2.8}$$

Whereas the notation employed is a bit similar to the one displayed in the WGAN section 2.7, we will add the hyperparameter λ to control the penalty term. His role is to correct the balance between the penalty term and the original objective. Similar to Ishaan's paper, we will use arbitrarily $\lambda = 10$.

2.4.1 Generator and Critic Architecture

After understanding the major improvements the WGAN-GP gives to a GAN, we can overview the difference in implementation. Whereas we implemented the GAN employing the Keras Library, we will now employ PyTorch's library in python. Indeed, creating a GAN is not an easy feat, and to help us go through the architecture's technical implementation, we used Github's repository [34] based on that specific machine learning library.

Generator

In a similar manner to the GAN's generator, this neural network needs some functions from PyTorch documentation, such as :

- **Linear**: It is the noise layer of our generator.
- **BatchNorm2D** : This function is the PyTorch implementation of Keras's BatchNormalization function explained above 2.1.1.
- **ReLU** : It is the rectified linear unit function. Mathematically, it is defined by $\text{ReLU}(x) = \max(0, x)$.
- **Conv2D** : This convolutional layer will serves for multiple purposes. Using this function, we will create the final three-dimensional tensor that will be used as input for the critic. Furthermore, we will build a new module **UpSampleConv** using this convolutional layer that will serve as an upsampling function[3].

[3]for more information, refer to the repository

The global structure of WGAN-GP's generator works again similarly to GAN's generator: a noise layer, four blocks with each the same layers, and a last convolutional layer followed by a tangent function to get the output between -1 and 1. The main difference here is the size of the blocks that consist of a BatchNorm2D, a ReLU, a UpSampleConv layer, followed by a BatchNorm2D, a ReLU, and finally, a Conv2D layer.

Critic

The structure of the critic follows the same pattern as the generator. Indeed, we begin with a layer of Conv2D followed by four blocks composed of the same layers and finally a Linear layer to flatten the output. A single block contains six layers: LayerNorm, ReLU, Conv2D, LayerNorm, ReLU, and ConvMeanPool.

The **LayerNorm** function here applies normalization over the inputs as described by this paper [4]. The reason behind using LayerNorm instead of BatchNormalization comes from the WGAN-GP paper [17] that did not see any differences when omitting the BatchNorm2D layer. On the other hand, they discovered that normalization that did not introduce correlations between examples in a minibatch, such as a layer normalization, weight normalization, or instance normalization, works better with the gradient penalty technique.

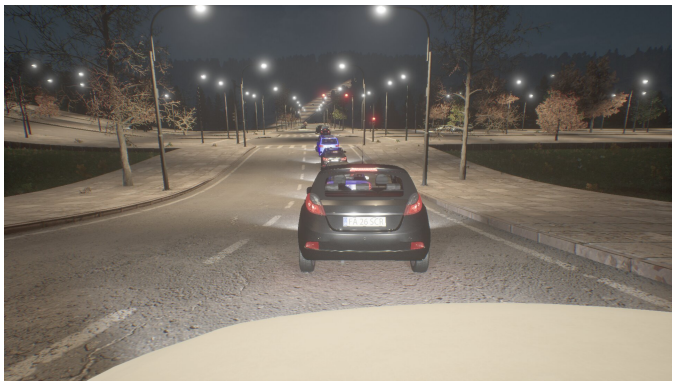
Another change comes from the **ConvMeanPool** module, which is a self-made variation of a standard Conv2D layer and is again a choice from the paper. Finally, we can point out that here we need to optimize both neural networks. For this purpose, we will use an Adam optimizer with a learning rate of 0.0001 and a beta of 0.5 [37].

2.5 GAN in enhanced photorealism

As a reminder, the main goal we want to achieve through this master thesis is to give realism to pictures taken from the DigitalTwin project created from the game engine Unreal. It is composed of automated cars going through a crossroads without bursting into one another, respecting the traffic light's rule. From this project, we will retrieve the pictures shown below in figure 2.4 that either come from a camera fixed at the crossroad or a camera placed inside a moving car.

After seeing GAN, WGAN, and then a way to improve it, we will now see another modification of the GAN's structure proposed by Richter et al. in their paper [38]. The researchers did not define its name clearly, so throughout this master thesis,


(a) Fixed crossroad's camera

(b) Conductor's Camera

Figure 2.4: Example of pictures taken from the DigitalTwin project

we will call the architecture EPE-GAN as it stands for enhanced photorealistic enhancement and use the idea of GAN's architecture.

The main difference between EPE-GAN and GAN subsists in the goal it wants to achieve. Rather than creating an image from scratch, the idea here is to enhance the realism of a given synthetic image. The generator's input will no longer be a noise vector but a synthetic image coupled with additional metadata from a game engine project. The architecture we will employ was first based on the popular game GTA-V.

As one could doubt, having only screenshots of a game is not enough to make a real lookalike photo. The pipeline will need to receive three other essential inputs. The first one concerns metadata (G-Buffers) that we will need to extract to provide

pieces of information about geometric structure (surface normals, depth), materials (metallic, opacity, specular), and lightning. The second involves the semantic labels of the synthetic images we need to extract from the MSEG architecture. The last one is about images taken from a real dataset such as Cityscape, for example [11] that will help the GAN renders photorealistic modified synthetic images.

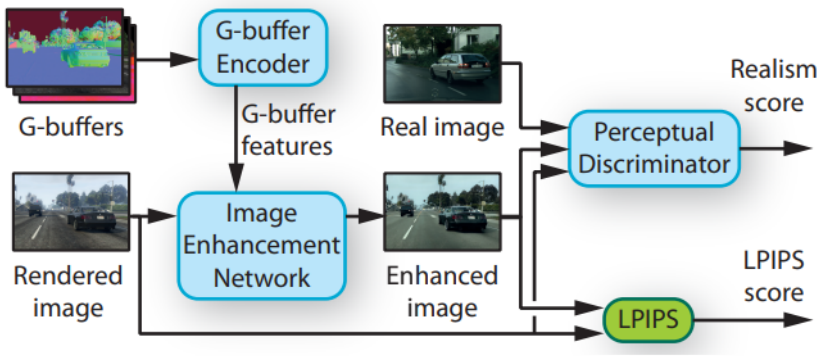


Figure 2.5: Architecture of an EPE-GAN [38] (Image source : Vladlen [38])

From the figure 2.5, we can see that the architecture works the same way as a GAN with two networks and metrics that will fine-tune the hyperparameters of both networks. Here the image enhancement network can be seen as the generator of the EPE-GAN[4], but it is trained with two objectives.

First, a **Learned Perceptual Image Patch Similarity** (*LPIPS*) loss penalizes significant structural differences by computing the activations of two image patches. A low LPIPS score means the image samples are intuitively similar (for more information, refer to the first equation of this paper [49]). Secondly, a perceptual discriminator is trained to distinguish images enhanced by the image enhancement network from the real images. We will now see more details about each of the three blocks we can see in figure 2.5 above.

2.5.1 Gbuffer Encoder

This block illustrated in more detail in figure 2.6 is neither a discriminator nor a generator but is used to feed the image enhancement network with the G-buffers inputs. It consists of multiple network streams that process each the same set of

[4]The EPE-GAN name is defined to propose a continuity in the architecture names seen throughout this master thesis. One should note that it does not generate images but rather transforms existing ones.

G-Buffers.

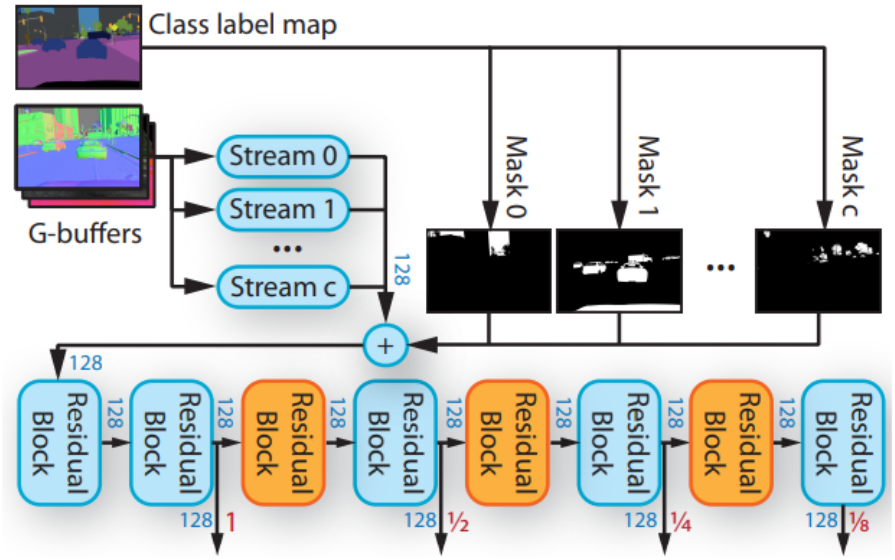


Figure 2.6: Architecture of the gbuffer encoder [38] of an EPE GAN (Image source : Vladlen). In red we can see the output scale of the feature tensors. In addition, the numbers in blue gives informations about the size of channel width.

As we can see from figure 2.6, the G-buffers encoder takes not only as input G-buffers but also the class labels map from the MSEG segmentation. Those labels come from the synthetic images that went through a pre-trained MSEG model we can get from a Github's repository MSEG [23]. In brief, the model will assign to each pixel of the image one of its 194 labels (see more in section 4.5). Those labels will then be used as masks in the G-buffers encoder to identify parts of the image.

Moreover, we can see that the G-buffers encoder uses convolutional layers in its residual block, as shown in the figure 2.7 below. The outputs will then be progressively merged into the image enhancement network.

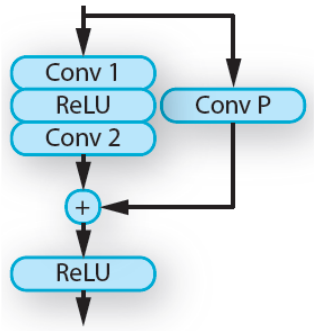


Figure 2.7: Architecture of a single Residual Block from the Gbuffer Encoder (Image source : Vladlen [38]). In case of downscaling, the Conv P layer will be used otherwise the layer is omited.

2.5.2 Image Enhancement Network

The image enhancement network is the "generator" of our EPE-GAN and aims to enhance a synthetic image into a photorealistic one. From figure 2.8 we can see that the G-Buffers features will be progressively added to the neural network. One should note that the features streamed into the network from the G-Buffers encoder corresponds to the processed image.

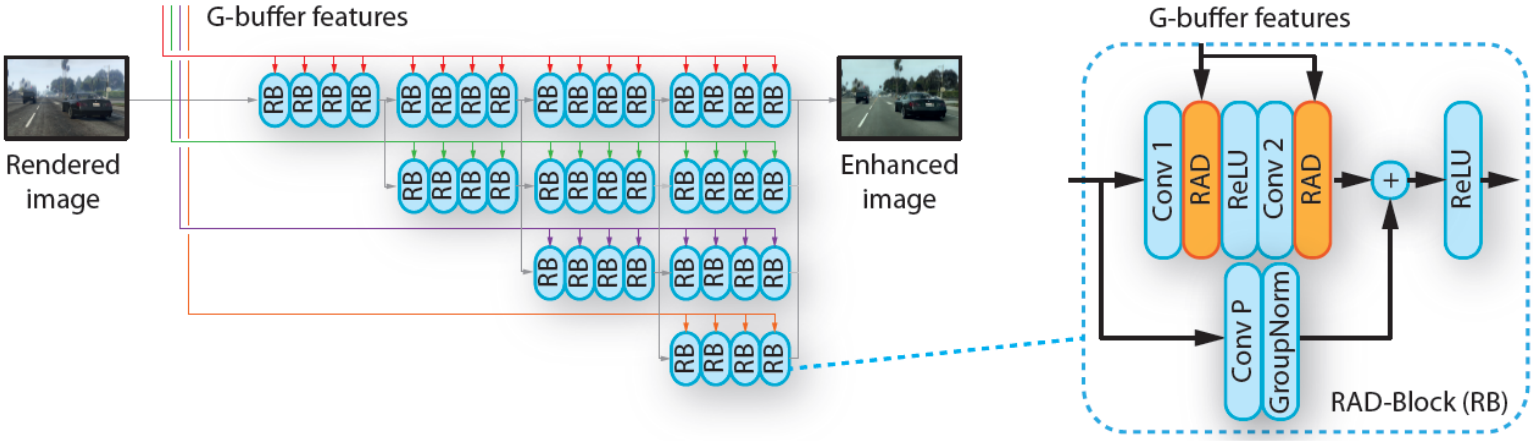


Figure 2.8: Architecture of the image enhancement network (Image source : from the paper paper [38]).

The global structure of the image enhancement network is based on the HRNetV2 framework [44] that processes the image into four stages with each its own resolution. Nonetheless, the EPE-GAN's paper proposed two main changes to the

HRNetV2 architecture. The first one concerns the stridden convolutions that will remove some important details and went replaced by regular convolutions. In addition to this, the batch normalization layers were replaced by rendering-aware denormalization modules to include the G-Buffers features as shown by the figure 2.9 below :

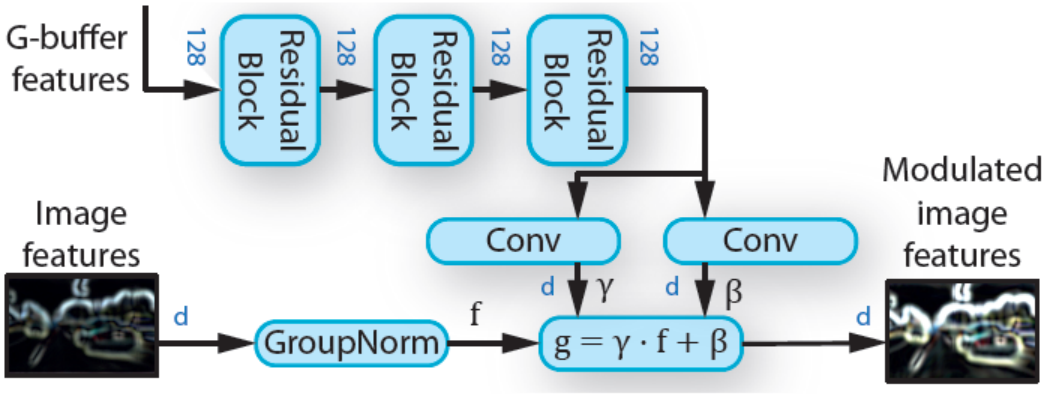


Figure 2.9: Architecture of the RAD modules (Image source : from the paper paper [38]).

Here, the three residual blocks have the same structure as the one in the G-buffers encoder architecture. They help to learn the weights γ and β and modulate the image features through metadata.

2.6 Alternatives to enhanced photorealism

Even though EPE-GAN seems to be the perfect solution to transform the screenshots from the DigitalTwin project into lookalike real images, it is not easy to gather the inputs correctly. As a reminder, we require a dataset containing real-world images, another dataset with synthetic images, their respective semantic labels, and their set of metadata called G-Buffers. Collecting those data is not the only tricky part. Indeed, we then need to train the network that uses a modified HRNet architecture and four different discriminators. For those reasons, we will need to look at other possible solutions in the literature.

Fortunately, the EPE-GAN paper [38] is not the first to propose a solution to unsupervised adaptation from synthetic to real worlds images. In this section, we will overview two other solutions. The first one, CycleGAN, can be seen as

a pioneer in unsupervised image-to-image translation with an architecture that achieved remarkable results. Even though the solution is not photorealism-oriented, it only needs two datasets to perform a domain shift. The second one, CyCADA, uses synthetic data and high-level semantic knowledge to propose an adaptation of those data into real-world image which tackle our initial problem.

2.6.1 CycleGAN

When we think about unpaired Image-to-Image translation, lots of papers propose solutions with adversarial networks such as the CoGAN [27] architecture that introduces two generators, one for each dataset with shared-weight in the first layers to learn a common representation between the two domains. Another unsupervised image-to-image model worth mentioning is the PixelDA model [9] with GAN-based techniques that add a noise image to the synthetic image before entering the generator and T, a task-specific classifier. T assigns a label to both the original and modified image. Both proposed innovations to the GAN's paper [16] but the resulting models have shown to work only for small image sizes, with a limited domain shift while being less stable than what was announced.

The first paper that introduced a proper and useable solution to an unsupervised image-to-image translation was the CycleGAN paper [51] in 2017. The model can propose a tranformation of a Monet's painting into a photorealistic image or the winter's version of a summer's landscape as shown in the figure 2.10 below.

The idea behind the paper is that the transformation should work in both senses. In the same way that the translation of a text from English to french can be translated back into English with minor modifications to the original text or that $\arcsin(\sin(x)) = x + 2k\pi$, the unsupervised image-to-image translation models should be able to transform a zebra into a horse and then back into a zebra. For this reason, there will not be one generator but two associated each with a discriminator.

Formally, we can define the two mapping functions between the source images X and the target images Y:

$$\begin{array}{lcccll} G & : & X & \longrightarrow & Y & \textit{with the associated discriminator } D_Y \\ F & : & Y & \longrightarrow & X & \textit{with the associated discriminator } D_X \end{array}$$

It is important to understand that G does not monopolize all the data from the dataset, and F will only have as input the generated image from G. As we work with unpaired image-to-image translation, we receive two datasets with each a latent

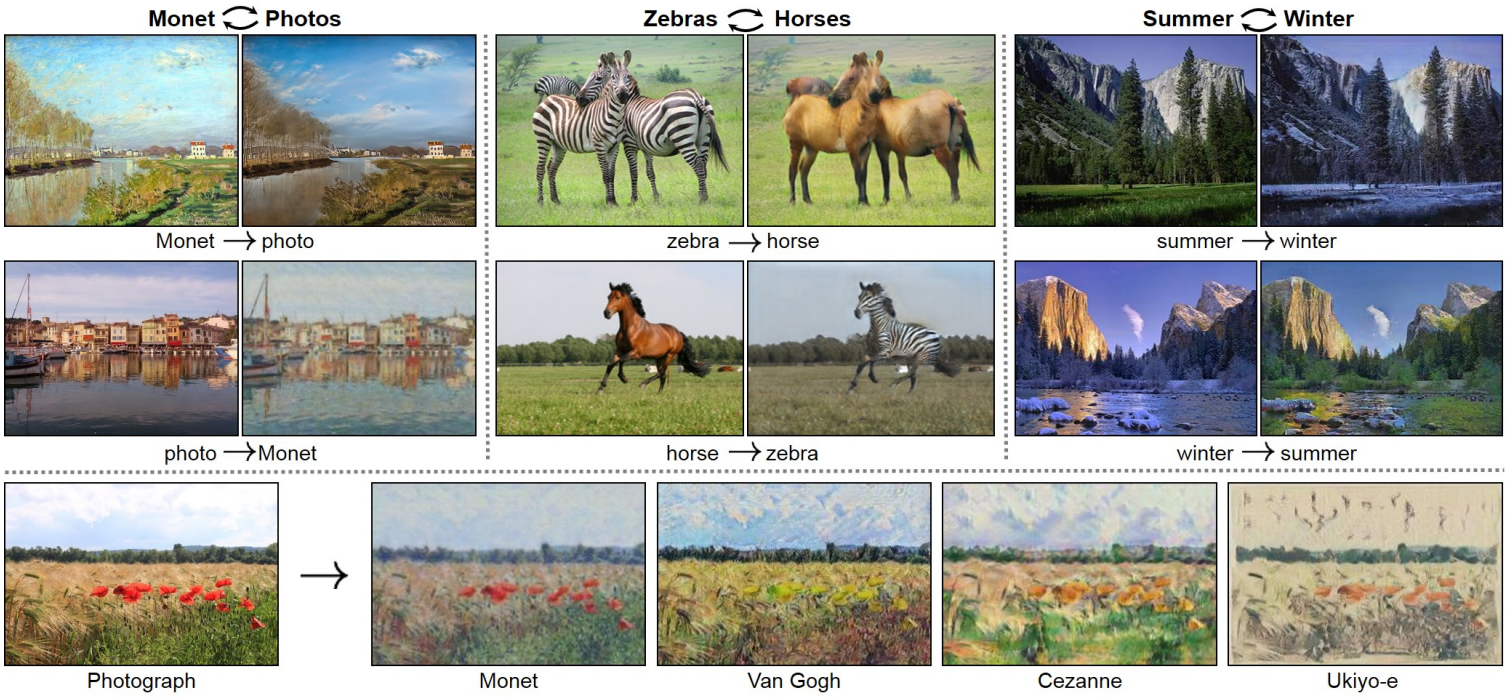


Figure 2.10: Examples of high resolution's transformations using a CycleGAN model (Image source : Junyanz [10]).

domain. We may be interested in only the transformation from one to another, but both translations should be considered, as argued by the paper. Using the equation 2.1 we saw in the section 2.1, we can slightly modify it to adapt to our two new functions. We will have to min-max the following adversarial losses :

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) &= \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[log(D_Y(\mathbf{x}))] + \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[log(1 - D_Y(G(\mathbf{x})))] \\ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) &= \mathop{\mathbb{E}}_{\mathbf{y}\sim p_{data}}[log(D_X(\mathbf{y}))] + \mathop{\mathbb{E}}_{\mathbf{y}\sim p_{data}}[log(1 - D_X(F(\mathbf{y})))] \end{aligned} \tag{2.9}$$

Furthermore, we must consider that G and F should form a bijection as one is the inverse of the other. Indeed, we need to force the *forward cycle consistency*, i.e., $x \to G(x) \to F(G(x)) \approx x$ and its inverse the *backward cycle consistency* : $y \to F(y) \to G(F(y)) \approx y$. Mathematically, the CycleGAN paper proposes the *cycle consistency loss* :

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{data}}[\| F(G(x)) - x \|_1] + \mathop{\mathbb{E}}_{\mathbf{y}\sim p_{data}}[\| G(F(y)) - y \|_1] \tag{2.10}$$

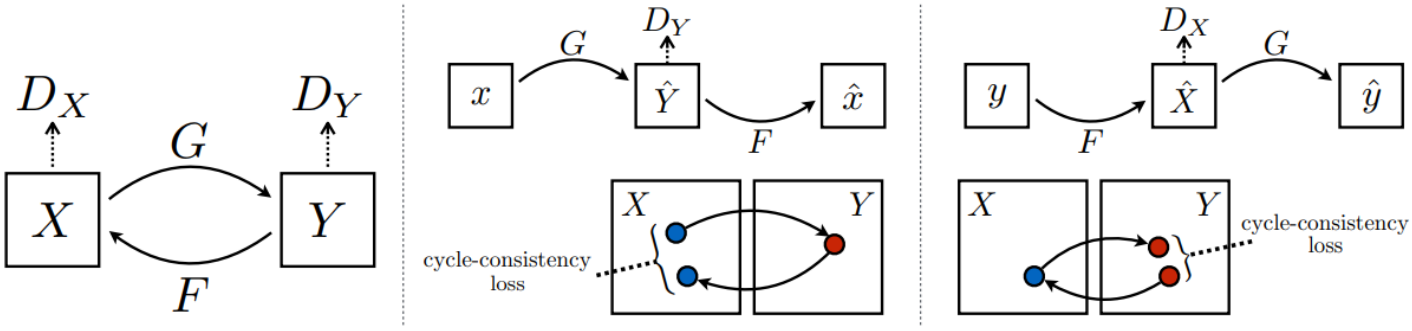


Figure 2.11: Training procedure of a CycleGAN. (Image source: Junyanz [10]). On the left, we can see the conceptual representation of the framework and the two adversarial losses. In the middle and right schemas, we can see the representation of respectively the *forward cycle-consistency loss* and the *backward cycle-consistency loss* that one needs to minimize. For example, if X is a dataset of horse images and Y is a dataset containing zebras images, we want to minimize the damage done to the reconstitution of the horse from its zebra's transformation but also minimize the damage done to the reconstitution of a zebra from its horse's transformation.

In figure 2.11 we can see a more visual resume of the reason behind the losses of our model. Finally, we can put those three losses together to form our objective :

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) \quad = \quad & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \quad + \quad \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \quad \lambda \mathcal{L}_{\text{cyc}}(G, F)\end{aligned}$$

With λ, the hyperparameter that controls the importance of the cycle consistency loss. In most cases he is set to $\lambda = 10$.

One phenomenon we can point out is that inside the transformed image, the cycle-GAN will hide the necessary pieces of information to build the reconstructed image as explained in *CycleGAN, a Master of Stenography* [10].In order to address the constraint of *cycle consistency*, the model will reduce the quality of the intermediate result. It reminds us how a GAN will try to fulfill in any way the loss function even at the cost of the quality result. While it may not seems like a big problem, it can be the reason behind a source of failure for the model.

Problems of CycleGAN

With two generators and two discriminators, the case of failure can be hard to solve because it is hard to understand where the problem stands. Furthermore, CycleGAN is deeply inspired by the GAN structure with a discriminator and a

generator but not from the Wasserstein-GAN as they are both from the same period (2017). Unfortunately, the mode collapse problem is thus one of the main problems one has to face when dealing with the training of a CycleGAN architecture.

2.6.2 Cycle-Consistent Adversarial Domain Adaptation

When we want to make a domain shift on an image, we want to ensure that elements rest in place, which can be difficult, specifically when we have unsupervised data between the source (i.e., synthetic) and target domain. While the CycleGAN's paper [51] was able to achieve remarkable results, we want to take it a step further [19] as we work mainly with driving scenes by including high-level semantics knowledge. The key idea here is to force semantic consistency before and after the adaptation. While we do not have access to such labels, we will have a second purpose for our GAN, which will try to predict those labels.

In the same way that we did with CycleGAN, we will have two generators between X_S the source dataset (i.e., synthetic images) and X_T the target dataset (i.e., real-world images) :

$$\begin{array}{lcccll} G & : & X_S & \longrightarrow & X_T & \textit{with the associated discriminator } D_T \\ F & : & X_T & \longrightarrow & X_S & \textit{with the associated discriminator } D_S \end{array}$$

With the purpose of producing convincing samples, from equations 2.9 we need to have the two losses $\mathcal{L}_{\text{GAN}}(G, D_T, X_S, X_T)$ and its symmetry $\mathcal{L}_{\text{GAN}}(F, D_S, X_T, X_S)$ as we also want to transform real images into synthetic samples. Furthermore, in order to preserved contents while transforming the images from one style to the other, we will need to restrain the two generators such that $x_s \to G(x_s) \to F(G(x_s)) \approx x_s$ and $x_t \to F(x_t) \to G(F(x_t)) \approx x_t$. As proposed by CycleGAN in the equation 2.10, we will use the *cycle consistency loss* with our two generators : $\mathcal{L}_{\text{cyc}}(G, F)$.

At this stage, we have the same objective as CycleGAN, but we need to add some terms to account for the semantic labels. In this second part, we tackle the problem of predicting the target labels Y_T while working with the synthetic data X_S, the synthetic labels Y_S and target data X_S. In order to learn the model f_T that predicts Y_T, we will try to find the best model f_S that is able to predict Y_S using X_S :

$$\mathcal{L}_{\text{task}}(f_S, X_S, Y_S) = -\,\mathbb{E}_{(\mathbf{x_s},\mathbf{y_s})\sim(X_S,Y_S)} \sum_{k=1}^{K} \mathbb{1}_{[k=\mathbf{y_s}]} \log\left(\sigma(f_S^{(k)}(\mathbf{x_s}))\right) \tag{2.11}$$

We find such a model by solving the equation 2.11 which is nothing less than a K-way cross-entropy loss. Here, K means the number of different labels, whereas cross-entropy is a widespread loss function for classification tasks in machine learning. It measures the average number of bits needed to encode data from a source with distribution p when we use model q. Furthemore, σ denotes the softmax function[5] that gives an output between zero and one, which explains the minus sign at the front of the formula. Indeed, the log of such interval gives a negative output, whereas it is more common to minimize a function than maximize it.

Using the generator to create target data instead of the original one, we know now that we will need to take into account in the final objective the loss: $\mathcal{L}_{\text{task}}(f_T, G(X_S), Y_S)$.

Secondly, we want to encourage high semantic consistency before and after image translation. For this purpose, we will use the imperfect model f_S to serve as a labeler that will ensure the same label is given to the image after its transformation. This *semantic consistency loss* can be defined as follows :

$$\begin{aligned}\mathcal{L}_{\text{sem}}(G, F, X_S, X_T, f_S) = &\mathcal{L}_{\text{task}}\left(f_S, F(X_T), \text{argmax}(f_S(X_T))\right) \\ &+ \mathcal{L}_{\text{task}}\left(f_S, G(X_S), \text{argmax}(f_S(X_S))\right)\end{aligned}$$

We use the *argmax* function to ensure the most probabilistic label is given when numerous candidates are possible. Finally, the two discriminators D_S and D_T based their decision on a pixel level whether the image has been transformed or not. Note that we could make that decision on a feature level whether the label seems plausible or not. From this idea, we will add a new discriminator D_{feat} and a final adversarial loss :

$$\mathcal{L}_{\text{GAN}}(f_T, D_{\text{feat}}, f_S(G(X_S), X_T)$$

Finally, we can resume our final objective :

$$\begin{aligned}\mathcal{L}_{\text{CyCADA}}(f_T, X_S, X_T, G, F, D_S, D_T) = &\mathcal{L}_{\text{GAN}}(G, D_T, X_S, X_T) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_S, X_T, X_S) \\ &+ \mathcal{L}_{\text{cyc}}(G, F) \\ &+ \mathcal{L}_{\text{sem}}(G, F, X_S, X_T, f_S) \\ &+ \mathcal{L}_{\text{task}}(f_T, G(X_S), Y_S) \\ &+ \mathcal{L}_{\text{GAN}}(f_T, D_{\text{feat}}, f_S(G(X_S), X_T)\end{aligned}$$

[5]The standard softmax function $\sigma : \mathbb{R}^M \to [0,1]^M$ is defined when $M \geq 1$ and follows the formula : $\sigma(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{M} e^{\mathbf{z}_i}}$ for $i \in [1, M]$ and $z \in \mathbb{R}^M$.

The complete CyCADA architecture is shown in figure 2.12. One can note from the final loss equation ?? that the CyCADA architecture will take more time to train than a CycleGAN as it extends its objective while taking more inputs. Furthermore, we will not train two discriminators but three which augment the number of training parameters.

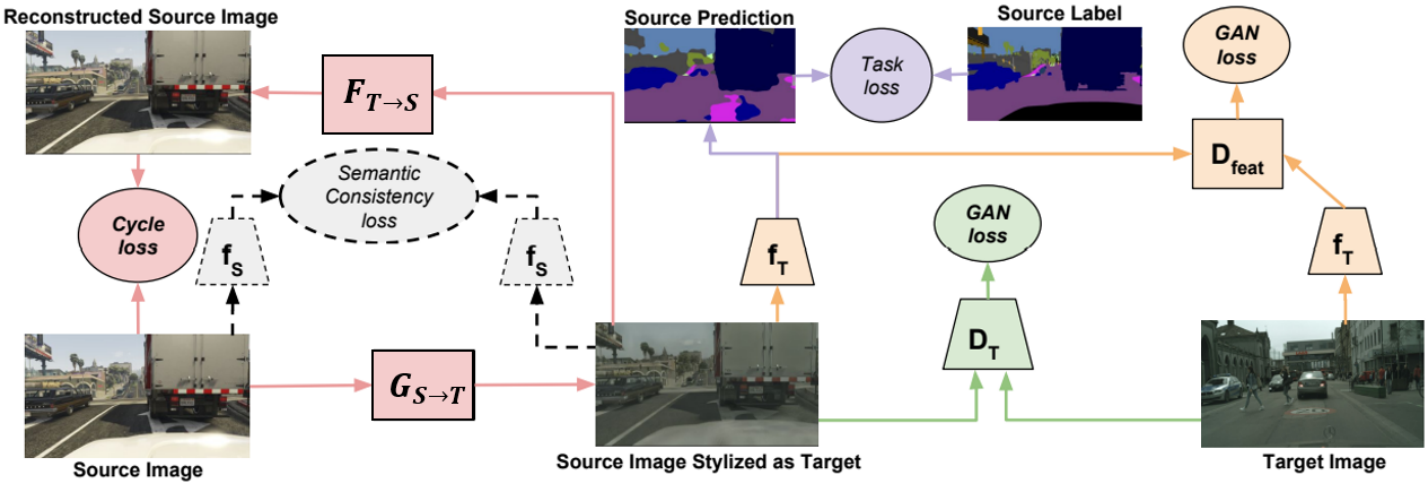


Figure 2.12: Architecture of a CyCADA model (Image source : from the paper [19]). In red on the left part of the image, we can see one of the two cycles that enabled the reconstruction of an image as controlled by the *cycle loss*. The other cycle should go from a target image to a source image style and then back to its reconstruction using the generator G. By comparing the semantic labels between the source image and its transformed version, we can see another gray loss we need to minimize, i.e., the *semantic loss*. This loss will help the consistency before and after image translation. In green, we show one of the GAN losses between a transformed source image and a sample of the real-world dataset. The other GAN loss will naturally compare a transformed target image with a sample from the synthetic images dataset. To continue, in purple we will need to minimize the difference between the source image prediction and the source label using the *task loss*. Finally, in yellow we will use the third discriminator to compare how plausible the guessed labels are on a transformed image.

2.7 Summary

To summarise this part of the master thesis, we first began explaining the mathematical foundations behind a vanilla generative adversarial network. Furthermore,

we discovered one of its main issues: the mode collapse problem. In order to solve this problem, we proposed different solutions and chose one of them. More specifically, the Wasserstein-distance solution measures the distance in distributions between the z-space and the real data distribution. We continue by discovering a significant improvement that relaxes the constraint on the weight variables by imposing a maximum value for the gradient on particular points of the input space.

After looking for a way to use the GAN framework to augment a dataset without learning supervision, we looked at how we can use adversarial networks to perform an unpaired image-to-image translation. In order to transform synthetic data into real-world images, we deepen into the EPE-GAN architecture that achieved impressive results in photorealism enhancement. We then proposed less complex architectures such as the CycleGAN framework, which does not require as many inputs but is not photorealism-oriented, or the CyCADA model that improves CycleGAN in terms of translation between synthetic data the real-world imagery.

Chapter 3

Metrics used in GANs

In this section, we will see utilized metrics to evaluate GANs architecture. The metrics explained here were not included in any architecture for optimization but rather served as a post-training analysis tool to quantify how well a model performs.

3.1 Inception Score

The Inception Score is one of the first widely adopted metrics for evaluating generated images. Proposed by Tim Salimans et al. [40], this score measures two important properties a GAN model should possess: the clearness of samples and the variety of outputs the model can produce. It uses an Inception model pre-trained on ImageNet to evaluate the images produced by the GAN. The better the two conditions are met, the higher the score will be, thanks to the following equation :

$$\mathrm{IS}(G) = \exp\left(\mathop{\mathbb{E}}_{\mathbf{x}\sim p_G}\left[\mathrm{KL}(p(y|\mathbf{x})||p(y))\right]\right) \tag{3.1}$$

Where G is the GAN model we try to evaluate, $\mathbf{x}$ is a generated image from the learned generator distribution p_G and KL is the Kullback-Leibler divergence to measure the dissimilarities between two probability distributions. Here, $p(y|\mathbf{x})$ stands for the conditional probability distribution of the label y knowing the image $\mathbf{x}$ and $p(y)$ is simply the probability of label y.

As one can note, it does not consider the real images part of the GAN and cannot determine how well the generator approximates the real distribution. As a result, it may give a high score to models that produce diverse samples while not matching the features of the real dataset. Furthermore, it can detect neither overfitting nor mode collapse, as stated by this paper [8]. Unfortunately, it is thus limited for our

application as we need to evaluate how well our architecture produces new data useable in a dataset.

3.2 Fréchet Inception Distance

The Frechet Inception Distance (FID) can be seen as an extension of the Inception Score seen above. Proposed by Martin Heusel et al. in 2017 [18], it compares the distribution of the generated images with the distributions of the images from the real dataset used to train the generator. In the same way as the Inception Score, it relies on an ImageNet pre-trained Inception network to produce a score. The lower the score, the closer the distance between the two distributions and the less difference there should be between the two sets of samples.

To be more precise, it will compare this network's last pooling layer (called coding layer) when given generated images or real samples. In order to make such a comparison, they will approximate the units of this coding layer as a multi-dimensional Gaussian parameterized by a mean and a covariance. The Frechet Inception distance we want to measure is thus a function of the mean and covariance of two distributions, and its formula is given by :

$$d^2((\mathbf{m}_r, \mathbf{C}_r), (\mathbf{m}_g, \mathbf{C}_g)) = ||\mathbf{m}_r - \mathbf{m}_g||_2^2 + \mathrm{Tr}(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r\mathbf{C}_g)^{1/2}) \tag{3.2}$$

Where $(\mathbf{m}_r, \mathbf{C}_r)$ denotes the mean and covariance of the real image distribution and $(\mathbf{m}_g, \mathbf{C}_g)$ represents the same for the generated image distribution. Unfortunately, even though the FID is more stable than the Inception Score, we can not track whether a high FID means a low-grade image quality or a poor diversity of generated images. It is up to the human eye to make a visual inspection and draw conclusions.

3.3 Other Metrics in GANs

Another metric that is worth mentioning is the Kernel Inception Distance. In 2018, Mikołaj Binkowski et al. [7] wrote a conference paper to analyze the Inception Score and the Fréchet Inception Distance and proposed their metric: the Kernel Inception Distance (KID). To be more precise, KID is the squared *Maximum Mean Discrepancy* between Inception representations with a polynomial kernel $k(x, y) = \left(\frac{1}{d}x^T y + 1\right)^3$. Compared to FID, it is unbiased but much more complex to implement.

Chapter 4

Methodology and tools

In this part of the master thesis, we will resume the technical aspect of the tools we used and the preparation of the datasets. To be more precise, we will overview the datasets used in the GAN and WGAN-GP architecture. Subsequently, we will look at the gathering of the inputs for the EPE-GAN model and, more precisely, at the inputs of the G-Buffer Encoder2.6.

4.1 Tools used

The main problem with GAN and other deep learning architecture is the computation time that ties together the performance of the computer used and the quality of the result we can obtain. For this master thesis, we were given access to a machine that combines four NVIDIA GeForce RTX 2080 Ti graphic cards possessed by the *Université Catholique de Louvain*. After some tests on the machine with different architectures, we realized that only a single RTX was used simultaneously for every simulation we made. We try combining those with the help of the environment variable *CUDA_VISIBLE_DEVICES*, but it continues working on a single device even when we give it access to multiple ones.

More accurately, we had memory shortage crashes when we wanted to allocate more than the 11 GB GDDR6 memory each graphic card possesses, even though the simulation had access to two of them. One can notice that it does not question the utility of having four graphic cards instead of one, as we can then work in parallel instead of in series. As we will see later, it will greatly impact the size of images we can work with. In order to solve this problem, we try to use google collab sessions to have more freedom in the size of images we could use. Indeed, while we had access to 11 GBytes on the personal machine of UCL for each session, this free service of Google proposes a GPU of 17 GBytes for anyone to use. Unfortunately,

after a few runs, we need to have a professional account for free usage. Furthermore, it discretely restrained the power of the GPU we have been accessed to when having a free account. All the results shown in the following section were thus obtained with a single NVIDIA GeForce RTX 2080 Ti.

4.2 Datasets used in GAN

In order to test a first Generative Adversarial Networks implementation, we need to begin with small images. Indeed, we will train two deep neural networks, and input size matters to reduce the computation time needed. For this purpose, our first dataset will be the CIFAR-10 dataset, proposed by Krizhevsky et al. in 2009 [22]. This image collection is widely used for machine learning research and consists of labeled 60.000 colored images of size 32 by 32 pixels. Those tiny images are labeled in 10 different classes as represented in the figure 4.1.

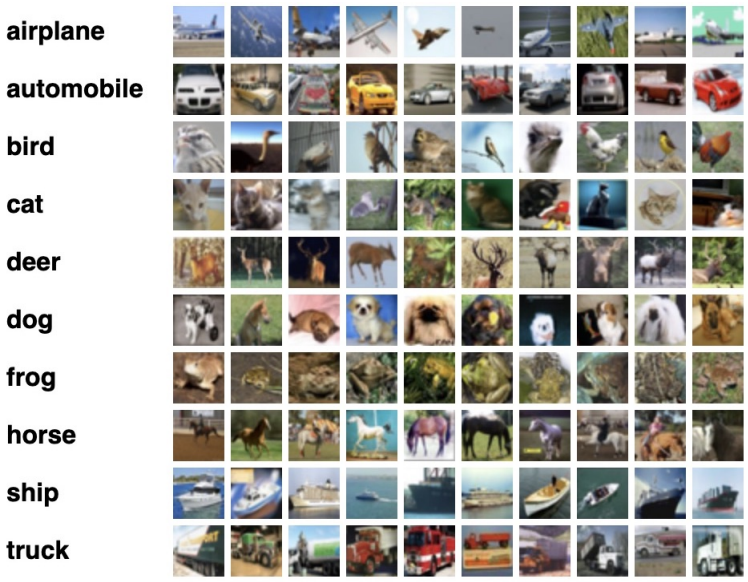


Figure 4.1: Example of diverse samples from each of the 10 labels from the CIFAR-10 dataset (Source : Krizhevsky [22]).

Alternatively, to download the dataset and load it in two steps which can take time, one can load them from the Keras library in python. In our case, we only use the cat-labeled images from this dataset to emphasize that we will be using the same type of data, i.e., road scenes.

After using CIFAR-10 data, we will use the MIO-TCD dataset [29] which consists of colored images acquired from 8.000 traffic cameras deployed all over Canada and the United States. The idea behind using such a dataset was to train the model with something close to what we will have with the Digital Twin project, but more importantly, the set of images is diverse and collected at different times of the day for different periods of the year as we can see from figure 4.2.

Figure 4.2: Example of diverse samples from the MIO-TCD dataset.

While the original dataset contains 600.000+ images, we only used a subset of 25.000 images. Intending to reuse the same GAN tested on the CIFAR-10 dataset, we cropped the images using the *resize* function from PIL's library in python.

4.3 Datasets used in WGAN-GP

In order to test our WGAN-GP architecture, we reuse the same subset of MIO-TCD images and begin with images of size $(32, 32)$. We then switched to images from the Digital Twin project. Those images came from either a driver's camera or a camera fixed along the road, as shown in the figure 2.4. Those pictures mainly were full HD, i.e., of size $(1920, 1080)$, so we again cropped them using the *resize* function from PIL's library in the same manner we did to rescale MIO-TCD images.

4.4 Retrieval of data for the EPE-GAN architecture

This section will focus on the difficulty of saving the G-Buffers from an Unreal Engine project. We want to get the G-Buffers related to a picture made by a camera placed on the driver's side to imitate the inputs the EPE-GAN paper worked with. We will then need to clean the obtained data.

4.4.1 Extracting the G-buffers

In order to fit all the information the EPE-GAN architecture needed, we were committed to finding a way to extract the G-buffers from the DigitalTwin project. In unreal, a few functions gives the ability to take a high-resolution screenshot, i.e., a screenshot with G-Buffers. In our case, we did not want to take a screenshot but rather used cameras to make those pictures. After testing different pre-built unreal nodes, we used a combination of methods to achieve our goal.

Firstly, we execute the command line "r.BufferVisualizationDumpFrames 1" that will automatically save the G-buffers for the following screenshots. Secondly, we ask for the three-dimensional coordinates and rotations of the car and place the camera approximatively above the hood of the car, on the driver's side. During the following step, we use the pre-built function "Take Automation Screenshot at Camera" (see figure 4.3) to take the screenshot with the rightful camera.

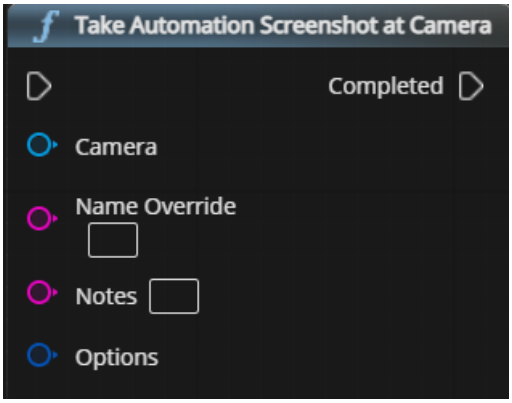


Figure 4.3: Built-in Unreal Engines node that takes a screenshot from a specific point of view

While saving all the G-Buffers, we simultaneously take a normal picture of the same scene to link the G-Buffers with its normalized image. In figure 4.4 we can

see all metadata we were able to retrieve.

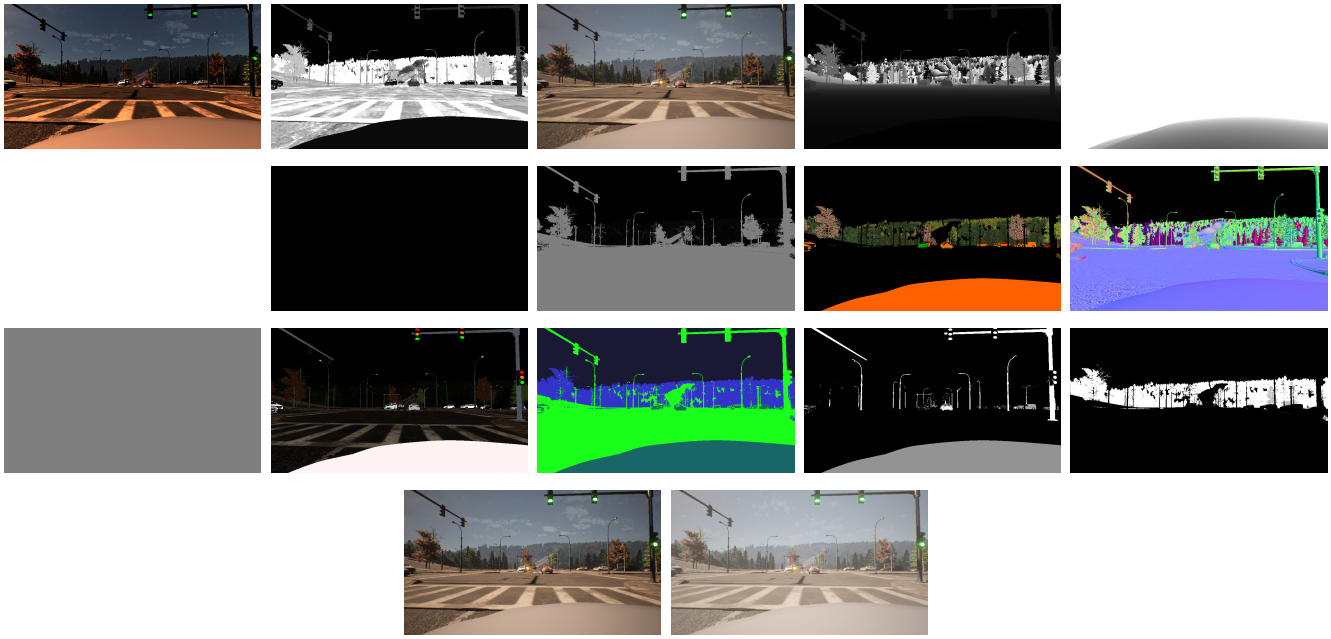


Figure 4.4: Those are the G-Buffers we were able to collect, from left to right, top to bottom : BaseColor, LightingModel, Metallic, Opacity, PostTonerMapHDRColor, PreTonerMapHDRColor, Roughness, SceneColor, SceneDepth, SceneDepthWorldUnits, SeparateTranslucencyA, SeparateTranslucencyRGB, Specular, SubsurfaceColor, WorldNormal. The last image is the original picture taken by the camera.

While for their project about the famous game GTA V, the pieces of information that they retrieved were different [1], one should note that it should not have an impact on the feasibility of adapting the framework to our data. Finally, we renamed each G-Buffers by adding to their name the car's number and the picture it is related to. A specific counter per car increments after every picture taken.

4.4.2 Cleaning the screenshots and reordering the data

Unfortunately, not all screenshots we saved were useable. Firstly, we can see from figure 4.4 that four G-buffers do not contain viewable information. Since the rendering stays the same from one screenshot to the other, we suppress them. Unfortunately, the suppression of 25% of the G-buffers only reduced 1% of the size

[1]Their sets of G-Buffers were: surface normals, depth, shaderIDs, albedo, specular intensity, glossiness, transparency, and lightning (approximate irradiance and emission, sky, bloom).

of all G-Buffers on the disk.

Secondly, we can also remove all the screenshots that are not complete. We need 13 images for a screenshot to be valid, but it is not always the case. Indeed, with the growing number of cars appearing on the map, after a dozen minutes of simulation, the game engine (and thus the graphic card) will be overloaded with tasks to do such as: calculating trajectories, avoiding collisions, respecting traffic lights, computing the weather behavior, making the screenshots for each car every x seconds depending on the frequency, etc. It results in sometimes incomplete G-Buffers. Those incomplete data constitute 20 to 25% of the dataset, and removing them is thus essential.

Finally, after cleaning all the data, we must reorganize them. As explained in the EPE-GAN Github, to prepare the datasets, we will need to compress the 12 G-buffers with the help of NumPy's *npz_compressed* function and put them in a directory while all the raw pictures need to be put in another directory. Even though it is compressed, surprisingly, it takes about 8 to 10 % more memory.

4.5 Performing the Segmentation

Computer vision is a thriving engineering field in informatics and deals with the high-level understanding of computers when given digital images or videos. This field began in 1966 with a summer project proposed by S. Papert to his students [35]. Five decades later, we are going to talk about the MSEG [23] project that should give us the class label maps we need in the G-Buffer encoder part of the EPE-GAN (2.6) architecture.

MSEG is a composite dataset that aims to unify different taxonomies and align the pixel-level annotations. To achieve such a feat, they reconcile the taxonomy of COCO [26], ADE20K [50], Mapillary [33], IDD [45], BDD [48], Cityscapes [11], SUN RGB-D [42] and merge them into 194 categories. In order to test their so-called universal taxonomy, they used an HRNet-V2 with a width of 48 for the high-resolution convolution. Whereas all their architecture is available on Github, the output of this model with the DigitalTwin synthetic data will give us the class labels we need to run the G-buffer Encoder. Unfortunately, this architecture blocked us as we could not make their version of HRNet work with our input on our machine. However, we attested the performance of their architecture with a demo available on Google Collab, as shown in the figure 4.5 while not being able to retrieve the exact segmentation of the pictures.

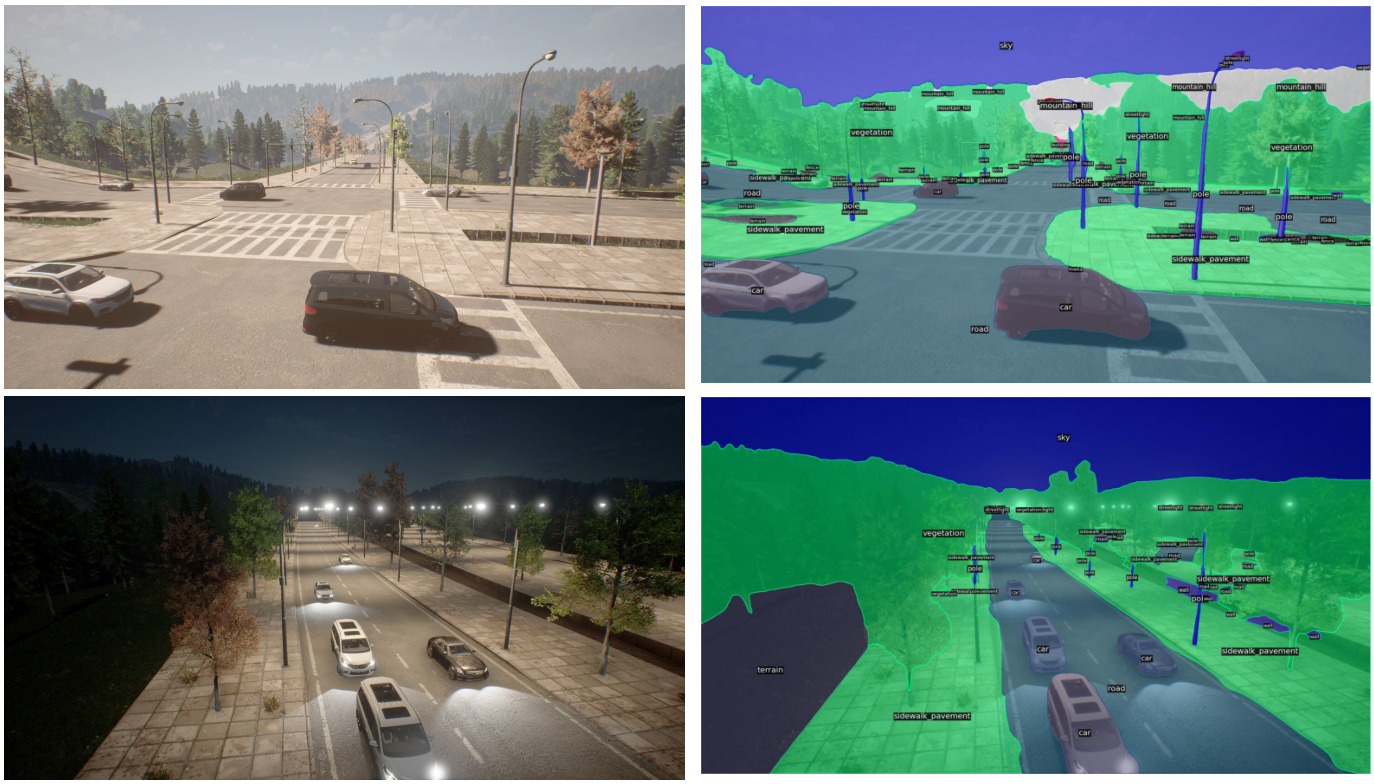


Figure 4.5: Examples of images taken from the Digital Twin project segmented using the MSEG model. We can see the synthetic image on the left, while the segmentation result is on the right.

The failure to achieve this segmentation greatly impacts this master's thesis since we can not use the EPE-GAN structure to yield results if the G-Buffer Encoder is not working.

Chapter 5

Experiments and Results

In this chapter, we will overview the different experiments we made and show their results for the different GAN frameworks. To be more precise, we will begin by examining the experiments we made with the GAN architecture. In addition, we will look at the experiments we made on the WGAN-GP framework and analyze the evolution of the computation time and the metric along the way.

5.1 Results with GAN

In order to test the GAN architecture, we first use the CIFAR datasets [22] which consists of 60.000 tiny color images of 32x32 pixels. As the dataset is divided into ten categories, we decided only to take the cat category and see how the architecture responds after training. In figure 5.1, we can see generated samples of a vanilla GAN model trained for 30.000 iterations.

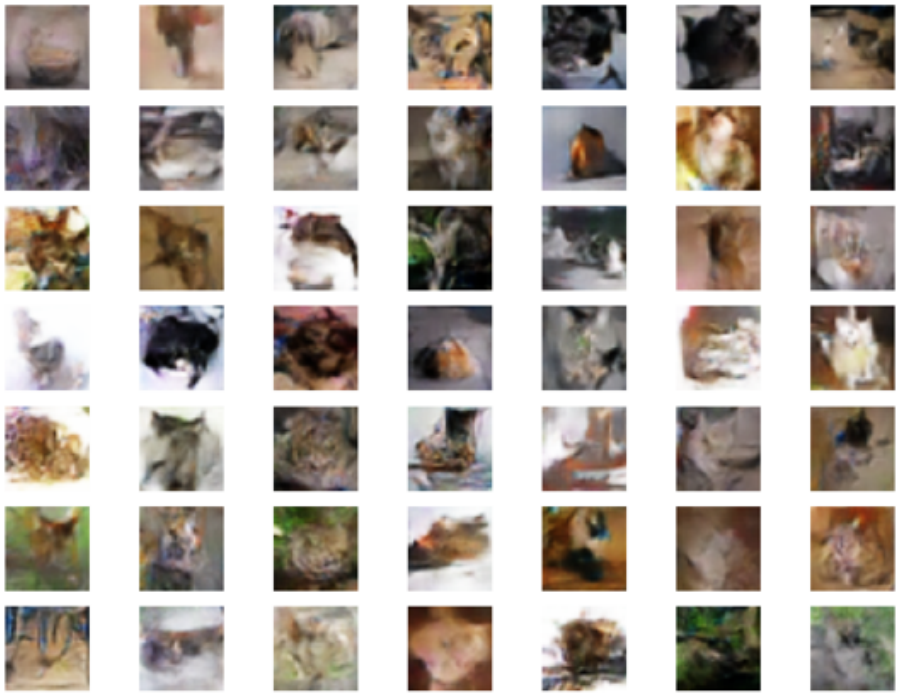


Figure 5.1: Generated images made by a GAN architecture trained with the cat images 32x32 from CIFAR dataset. [$Batchsize = 64, iter = 30.000, dataset_size = 6.000$images]

One should note that the generated images we show in figure 5.1 and later figures are continuous, which means they were non-selected to show more remarkable results in this master thesis. Apart from this, in terms of computation time, a single iteration takes around 0.95 seconds when training with images of size $(32, 32)$ with this model, which means it took eight hours to get the result we see in figure 5.1. We then try to do the same with the MIO-TCD dataset after resizing its input to the size $(32, 32)$ to match the input we gave the discriminator when using the CIFAR dataset.

Several conclusions can be drawn from figure 5.2. Firstly, we can see that even though we ran the experiment three times independently, we failed three times to get the diversity we were able to get from the previous experiment with the CIFAR dataset. As we explained in the section 2.1.2, this problem comes from the discriminator being stuck in a local minimum and unable to learn to reject GAN's proposition. One can notice that while the three experiments were running with three different models, the figure 5.2 gives three different outputs. It means at least three different distributions can fool the discriminator and thus find three different local minima. As we supposed the issue comes from the mode collapse problem, we will now try to train another architecture that suffered less from this problem.

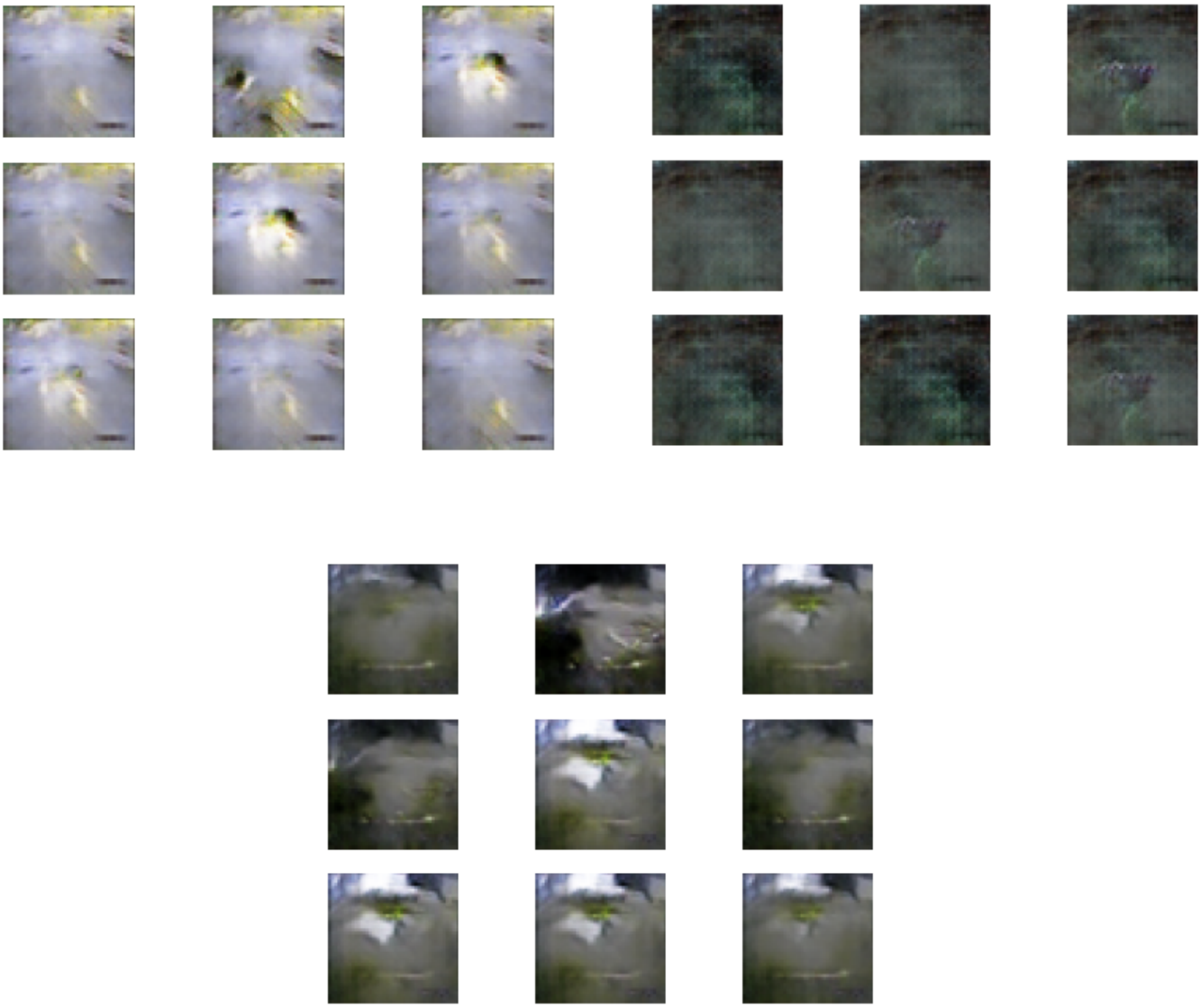


Figure 5.2: Samples generated by a vanilla GAN architecture trained with MIO-TCD images of size $(32, 32)$. Hyperparameters : $[Batchsize = 64, iter = 30.000, dataset_size = 25.000$images$]$. The experiment was run three times, giving three independent models.

5.2 Results with WGAN-GP

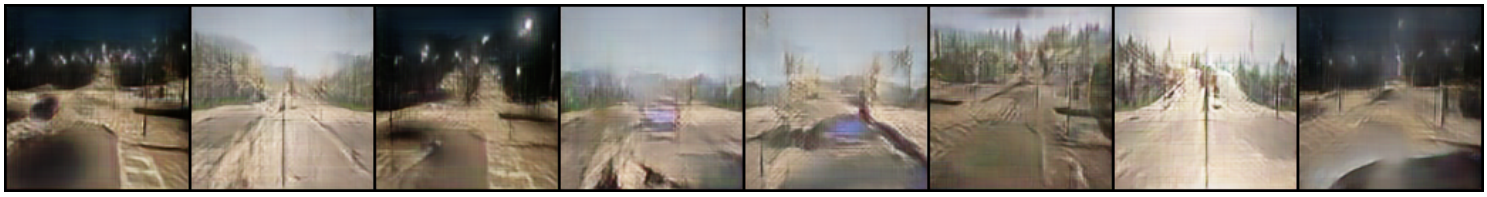
After grasping the limitation of the vanilla GAN model, we can now try to use the WGAN-GP architecture explained in the section 2.4.1 on our data. We first apply the WGAN-GP architecture on the MIO-TCD dataset with small images of size $(32, 32)$ as we did in the previous section 5.1. In figure 5.3 we can see some samples generated with a model trained during 30.000 iterations :

Figure 5.3: Samples generated by a WGAN-GP model trained with MIO-TCD images of size $(32, 32)$. Parameters : $[Batchsize = 64, iter = 30.000, dataset_size = 25.000$images$]$

From figure 5.3, we can remark a wider diversity of samples that do not seems to suffer from the mode collapse problem compare to the figure 5.2. Furthermore, from a human point of view, we can notice the resemblance between those samples and the road camera images from MIO-TCD we fed the model. For instance, from the MIO-TCD samples shown in figure 4.2, we can notice a small black rectangle on the bottom right of most images indicating the date and time of the camera shot. Indeed, the dataset contains images taken from traffic cameras deployed over Canada and the United States. This detail in the image can also be seen in the samples we show in figure 5.3.

Now that we know that the model can achieve noticeable results with images of size $(32, 32)$ from the MIO-TCD dataset, we can try to increase the input's size. For the simulation, we will manipulate images from the DigitalTwin project with a size of $(128, 128)$. Such results can be seen in figure 5.4 :

After 500 iterations



After 10.000 iterations

After 50.000 iterations

After 90.000 iterations

Figure 5.4: Evolution of a WGAN-GP model throughout the iterations. Each row contains samples from the same generator for 500, 10.000, 50.000, and 90.000 iterations. Parameters : $[Batchsize = 4, dataset\_size = 25.000\text{images}]$

From the simulation illustrated in figure 5.4, we first observe that the samples do not seem to suffer from the mode collapse problem. From a human perspective, there is a clear evolution between the 10.000th iteration and the 90.000th, where we better recognize inspiration from the Digital Twin project as shown in figure 2.4. Furthermore, we can note that it still has trouble producing valid generated samples for images taken from the driver's camera. Indeed, in figure 5.5, for the same simulation we did previously, we only look at the generated samples that represent shots taken from a fixed camera:

Figure 5.5: Samples generated by a WGAN-GP model trained with Digital Twin images of size $(128, 128)$. Parameters : $[Batchsize = 64, iter = 90.000, dataset_size = 25.000$images$]$

From the illustration 5.5, we can conclude that the model prefers more stable input images, which means we will have less diverse data. We could try to augment the number of iterations, but in terms of computation time, each iteration already took 2.72 seconds, as we can see in table 5.1 which means the simulation lasted nearly three days (68 hours) on the machine we had accessed to.

	Size of images			
Batch size	(32, 32)	(64, 64)	(128, 128)	(256, 256)
4	0.31	0.36	2.72	*crash*
16	0.32	0.87	7.5	/
32	0.32	1.63	*crash*	/
64	0.45	3.02	/	/

Table 5.1: Evolution of the computation time per iteration when we vary the size of the input images and the batch size for a WGAN-GP model.

From table 5.1, we can note that a simple GAN architecture with only a generator and a critic takes a long time to train. Furthermore, we are observably limited with the computation power available since when we train our model with bigger images, the machine crashes or takes too much time. One can note that to preserve

the machine we accessed, we did not try too many times to crash it to calculate the exact size of images we could afford. Importantly, knowing that the machine crashes already for such small images shows that the system can not endure huge data size and will not be able to serve as a dataset augmenter unless we significantly modify in some ways the tools we are working with (see section 6 below).

Finally, in figure 5.6, we show the evolution of the two metrics described in section 3 when a WGAN-GP model progresses through the iterations:

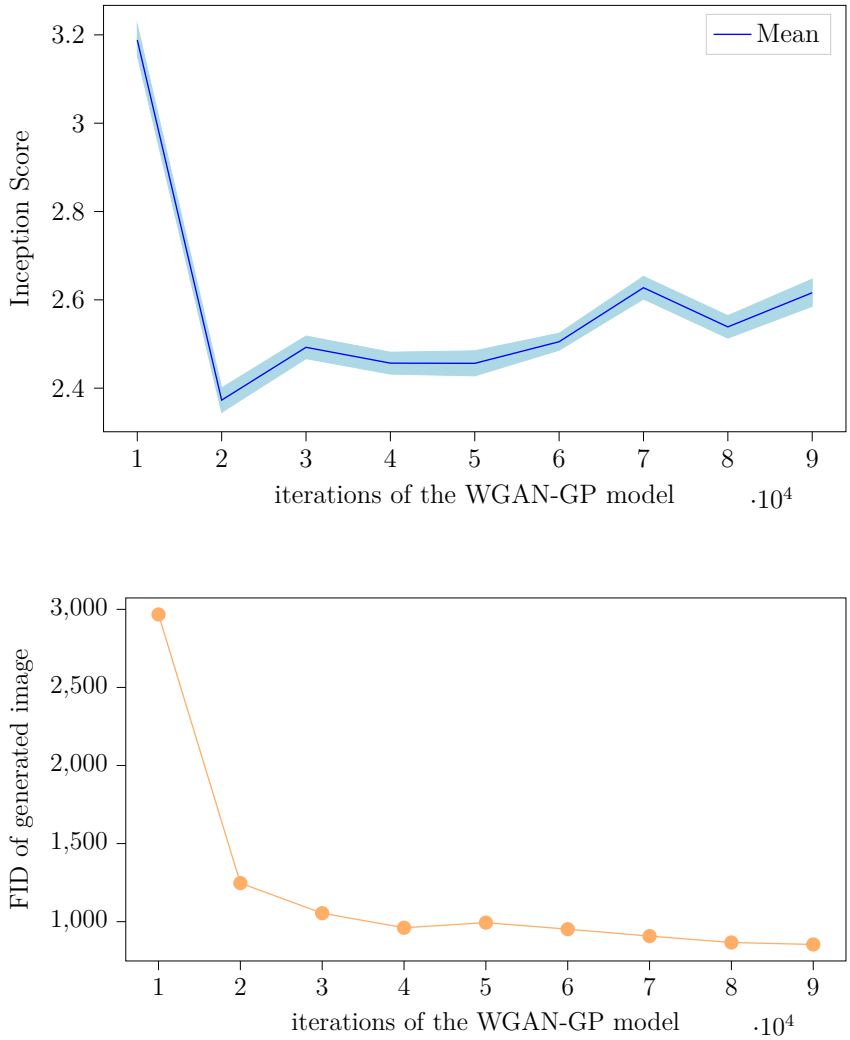


Figure 5.6: Evolution of the Inception Score and the Fréchet Inception Distance for the WGAN-GP model with input size $(128, 128)$. In the first graph, we also plotted the standard deviation among 10 sets of 3.000 generated samples.

Firstly, we can see that the Inception Score does not give much information about the evolution of the quality of the generated images that the WGAN-GP model can output. It seems to stagnate from the beginning around low values which may be an indication of a constant poor diversity of generated samples rather than their lousy quality throughout the iterations of the model. Here the term "low" refers to the value the Inception Score can take when evaluates on famous datasets shown in this ECCV 2018 paper [41].

On the other hand, the FID is more precise and shows that the two distributions get closer as the WGAN-GP improves. It correlates with what one can notice when looking at the evolution of the generated samples in figure 5.4. It improves quickly at the beginning, but we can barely see any differences between the samples at 50.000 iterations and those at 90.000.

5.3 Results for enhanced realism

In order to make a deep neural infrastructure work, one needs to have the correct input to pass to the network. While we succeeded at getting the synthetic data from Unreal Engine and for every image their respective G-Buffers, we could not get the proper semantic segmentation for those images. This semantic segmentation needs to be done using an MSEG model [23] as imposed by the EPE-GAN paper to fit the suitable class labels into the framework. Unfortunately, we could not fit the data from this architecture into the inputs we needed in the EPE-GAN G-buffer Encoder as explained in the section 4.5. Furthermore, we tried to suppress those semantic inputs in this encoder without success. Indeed, as we can see in the figure 2.6, failing to get the class labels impacts the whole framework.

Chapter 6

Discussion and perspectives

This master thesis was the occasion to discover the deep neural field created by Ian Goodfellow in 2014. After working with several adversarial networks, we will discuss deeper the work done, some questions one could wander when working with adversarial networks, and future research to continue this project.

6.1 Introspectives

Even though we implemented a generator that proposed decent results in specific cases, the outcome of our work was far below our expectations. Firstly, we could not benefit plainly from the machine composed of four NVIDIA 2080 Ti graphics cards we were given access to and ran the simulations on only a single GPU instead of the fours. This issue led to huge drawbacks where the computation time of each simulation was a decisive factor in the quality of generated samples we produced. Besides being unable to check if we reached a plateau in the experiment, the time constraint restricted us in the number of qualitative simulations we could run. Indeed, digging into the impact of the batch size on the quality of generated synthetic data is interesting, but one simulation will last numerous days before getting a valuable result.

Secondly, we failed to retrieve the segmentation labels data from our synthetic images. Being unable to correctly identify the source of the problem with the MSEG architecture led to failing the main objective of this master thesis. In parallel, acquiring the G-Buffers data took a noticeable amount of time while not being useful for its end purpose.

Lastly, the choice of architecture for the unsupervised image-to-image translation was poor, considering the result we could barely achieve with smaller inputs on a less

complex architecture. Importantly, the difficulties encountered in training a simple WGAN-GP model should have been a hint to begin the research for a photorealistic transformation of synthetic data with less demanding GAN architecture. Beginning such research with the CycleGAN framework could have been a better obtainable milestone considering the model only requires two datasets and no extra data from the project DigitalTwin.

6.2 Questions

When one wants to train an adversarial network, some questions about the batch size and minimum data required to train a GAN can emerge. In this section, we will try to respond to those questions using the literature to pave the way for future works.

What batch size should I use when training a GAN-based network ? As a reminder, the batch size of a neural network defines the number of propagated samples at every iteration. To our knowledge, there does not exist a batch size formula for all generative adversarial networks. From one paper to the other, each seems to make its own decision based on trials and the complexity of architecture. As an indication, we noted the batch size of popular GAN-related papers: Conditional GAN [20] (1-10), unrolled GAN [31] (28), Deep Convolutional GAN [36] (128), Super-Resolution GAN [25] (16), Wasserstein GAN [3] (64). One can notice that a single sample strategy was prioritized for both CycleGAN [10] and EPE-GAN.

How much data should I have to train a GAN ? In the CycleGAN [10] paper, the same architecture was used on several datasets, as illustrated in figure 2.10, and their sizes went from 400 to 50.000 training images. Moreover, a similar comparison can be drawn in the CyCADA paper. This example shows that the number of samples needed for a GAN-based network mostly depends on their availability, their variety, and the objective one wants to achieve rather than a fixed number of images.

6.3 Future works

Even though this field of machine learning is relatively new, around 50.000 papers cited Ian's paper in eight years. While not all will directly contribute to the development of the technology, it still shows an important interest for GANs whether we use them for data augmentation, text-to-image translations, image reconstitution, and other techniques. In the case of unsupervised synthetic to

real-world image translation, we can try to continue to analyze the CyCADA and EPE-GAN architecture and use it in a game engine environment. If such models propose convincing outputs, it should be possible to go a step further and propose a real-world live version filter for any game project that uses similar synthetic data. A first step could be to begin with an analysis of the CycleGAN architecture.

Chapter 7

Conclusion

This master thesis was the occasion to get a glimpse of the possibility a GAN-based network can achieve. By digging into the adversarial training domain, we witness some of its weaknesses, such as the difficulty of producing a GAN capable of generating diverse outputs. When trained with pictures taken from both fixed and moving cameras, the model could reproduce the images from the fixed camera properly while struggling to produce plausible images taken from a moving camera. It is thus rather sensible to the data and its hyperparameter, and trials-errors are necessary to find the correct model.

Another fragility that can affect the output's quality is the computation power available. Indeed, generative adversarial networks are computational intensive since two networks are trained instead of one. When we backpropagate, we will thus have twice the number of parameters and weights to update than a standard convolutional neural network.

Lastly, even though we failed to propose a working synthetic-to-photorealistic GAN-based model, we described two other simpler solutions than the one we investigated.

Bibliography

[1] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[2] L. Alvarez. Artificial intelligence. https://www.trade.gov/artificial-intelligence.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] D. Bang and H. Shim. Mggan: Solving mode collapse using manifold-guided training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2347–2356, 2021.

[6] G. Batra, A. Queirolo, and N. Santhanam. Artificial intelligence: The time to act is now. https://www.mckinsey.com/industries/advanced-electronics/our-insights/artificial-intelligence-the-time-to-act-is-now, Feb 2018.

[7] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018.

[8] A. Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[9] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[10] C. Chu, A. Zhmoginov, and M. Sandler. Cyclegan, a master of steganography. *arXiv preprint arXiv:1712.02950*, 2017.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[12] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[13] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[14] J. Feng and S. Lu. Performance analysis of various activation functions in artificial neural networks. In *Journal of physics: conference series*, volume 1237, page 022030. IOP Publishing, 2019.

[15] A. Gharakhanian. Generative adversarial networks - hot topic in machine learning. https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html, Jan 2017.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[18] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[19] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. Pmlr, 2018.

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[21] A. F. Jauregui. How to code a gan in python with google colab ?

[22] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[23] J. Lambert, Z. Liu, O. Sener, J. Hays, and V. Koltun. Mseg: A composite dataset for multi-domain semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2879–2888, 2020.

[24] Y. LeCun. What are some recent and potentially upcoming breakthroughs in deep learning? https://quorasessionwithyannlecun.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning-1, Apr 2016.

[25] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[27] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. *Advances in neural information processing systems*, 29, 2016.

[28] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[29] Z. Luo, F. Branchaud-Charron, C. Lemaire, J. Konrad, S. Li, A. Mishra, A. Achkar, J. Eichel, and P.-M. Jodoin. Mio-tcd: A new benchmark dataset for vehicle classification and localization. *IEEE Transactions on Image Processing*, 27(10):5129–5141, 2018.

[30] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. Oct 2017.

[31] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[32] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 2011.

[33] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 4990–4999, 2017.

[34] H. Nguyen. Jalola/improved-wgan-pytorch: Improved wgan in pytorch. https://github.com/jalola/improved-wgan-pytorch, Apr 2018.

[35] S. Papert. *The summer vision project*, 1966.

[36] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[37] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

[38] S. R. Richter, H. A. AlHaija, and V. Koltun. Enhancing photorealism enhancement. *arXiv:2105.04619*, 2021.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[40] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[41] K. Shmelkov, C. Schmid, and K. Alahari. How good is my gan? In *Proceedings of the European conference on computer vision (ECCV)*, pages 213–229, 2018.

[42] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[44] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.

[45] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker, and C. Jawahar. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1743–1751. IEEE, 2019.

[46] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[47] L. Weng. From gan to wgan. *arXiv preprint arXiv:1904.08994*, 2019.

[48] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2(5):6, 2018.

[49] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[50] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

[51] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Appendix A

Annexes : Theoretical Part

A.1 Algorithm of the GAN framework

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**