Université de Paris

Université de Montréal

Internship Report

Application of Deep learning methods for generation of ultra-realistic galaxy images

Lucas Goupil

Supervised by Laurence Perreault-Levasseur

21.09.2020 - 11.12.2020

Acknowledgements

I wish to thank my tutor Laurence Perreault-Levasseur for giving me this unique and enriching opportunity, for the countless discussions and debates over theories and practices and finally for her guidance that has taken me through so much more knowledge that I could have ever hoped for.

I would also like to thank Yashar Hezaveh for his precious insights and help throughout the project.

I thank everyone in the research team for the weeks of sharing our knowledge and their help with figuring out theories and ideas.

Finally, I would like to thank my pedagogical tutor Mahendra Mariadassou for his advice on the internship report, the defense and on post-bachelor orientation.

Abstract: Application of Deep Learning methods for generation of ultra-realistic galaxy images.

In the light of recent progress in the field of Machine Learning, research in physics can now benefit from the use of such a tool. Particularly efficient for typically difficult tasks where traditional algorithms are lacking a clear pathway to understanding the underlying protocol to follow, Deep Learning methods offer a valuable alternative. Such a task is, in the context of this project, the generation of realistic artificial images of galaxies. I explored the potential of Variational Auto-Encoders (VAEs) to conduct such a task. After careful application of the theory of Machine Learning on Convolutional Neural Networks (CNNs) and valuable statistical analysis for hyperparameter determination, I seeked to elaborate a VAE that would be able to generate new samples comparable in their simulation of density distribution, stellar light and stellar velocity fields to state-of-the-art Generative Adversarial Networks (GANs). With the addition of very recent enhancement techniques for VAEs I was able to generate samples that could almost compete with more advanced generative models.

Keywords: Machine learning, Variational Auto-Encoders, Galaxy light profiles, generative models, Generative Adversarial Networks

Abstract: Application de méthodes de Deep Learning pour la génération d'images hautement réalistes de galaxies.

Avec les récents progrès dans le domaine du Machine Learning, la recherche en physique peut grandement bénéficier d'un tel outil. Particulièrement efficaces pour des problèmes complexes où les algorithmes traditionnels peinent à définir un schéma détaillé dans leurs étapes vers une résolution, les méthodes de Deep Learning offrent une alternative intéressante. Une telle tâche est, dans le contexte de ce projet, la génération d'images artificielles de galaxies les plus réalistes possible. J'ai exploré le potentiel d'Auto-encodeurs Variationnels (VAE) pour arriver à bout d'une telle tâche. Après une soigneuse application de la théorie du Machine Learning sur un Réseau de Neurones Convolutionnel (CNN) et de précieuses analyses statistiques pour la détermination d'hyperparamètres, j'ai cherché à élaborer un VAE qui peut générer de nouveaux échantillons d'images de galaxies qui pourraient être comparables dans leurs simulation de la distribution de masse, la lumière stellaire et les champs de vélocités stellaires au sein d'une galaxie à un Réseau Génératif Adversarial (GAN). Avec l'ajout de très récentes techniques pour l'amélioration des VAE, j'ai été capable de générer des échantillons qui peuvent presque rivaliser avec ceux de modèles génératifs plus avancés.

Mots-clés: Machine learning, Variational Auto-Encoders, Profils de lumière stellaire, Modèles génératifs, Generative Adversarial Networks

Graphical abstract

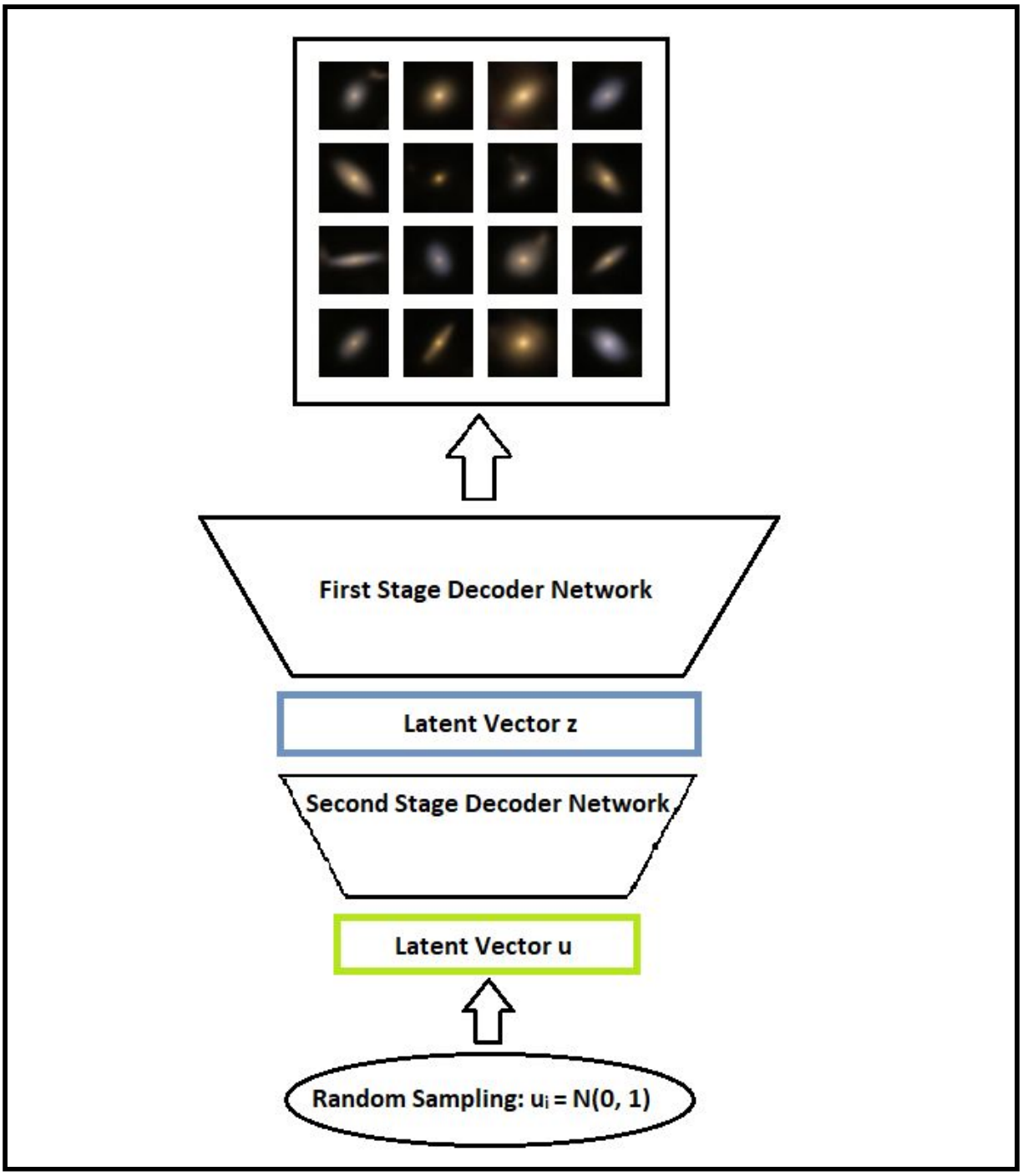

Table of contents

1. Introduction

1.1 The complexity of galaxy light profiles

Galaxies are incredibly complex systems. They are the sum of billions of years of physical interactions and aggregation of gasses into stars and planets. Each of these systems are unique and have a different history in their respective times and ways of formation. These facts combined with the knowledge of the collisions between multiple systems or inside a given system expresses themselves as giving a great visual disparity in the shapes and colours of galaxies. Hence, the challenge of building realistic images of galaxies lies in the definition of a realistic representation of how the light is distributed inside a galaxy. These definitions have to account not only for the shape of the galaxy itself but also for the distribution of the intensity of light, the distribution of mass, the velocities of objects inside the galaxy and even artifacts such as supernovas or nebulas. These representations are referred to as light profiles.

Figure 1: Image of the Messier 101 (NGC 5457), the Pinwheel galaxy, taken by the Hubble Space Telescope (1). We can see that the distribution of light in the arms of the spiral galaxy isn't continuous and presents brighter or darker spots.

Many attempts have been tried to define one or multiple mathematical equations that could capture this complexity as a function of one or a few parameters. An obvious example for such a function is the Sérsic light profile (2), first published in 1963 that has since taken the name of its inventor, J.L. Siersic. This light profile has been used countless times for the generation of simple-looking images of galaxies. However, it fails to capture the more detailed characteristics of galaxies that are essential for a realistic depiction of such astronomical objects and generalizes poorly to more than one type of galaxy. Adapting this light profile for very complex distributions is possible in practice but requires a lot of tuning on the part of the researcher. While I am indeed capable of fitting this profile to some extent to realistic images, it comes at the very high price of tuning time. We will soon see that there could very well exist a promising alternative to this method.

Since the 1970's, astronomers have refined their understanding of galaxies in part by the study of dark matter (3) and its effect on galactic velocity fields. The exact nature or even distribution of this dark matter is still an active field of research in physics as a few theories are still probable.
All these facts summed up justify the need for a tool that could transcend the very tight limits of standard mathematical equations and cover a wider range of galaxy types while still giving a lot of control over what to generate.

1.2 Machine learning and Deep Learning methods

Machine learning is a field of computer science that relies on the elaboration of programs that aim to be able to learn and adapt for a given task without being guided by explicit steps defined by a human. Deep Learning is a field of Machine learning that relies on the use of artificial neural networks to achieve this goal. These networks (collection of organized mathematical operations) adapt by the means of "backpropagation" of a gradient. This gradient is defined with regards to each trainable parameter in the network, called weight, and depends on a loss function that compares the output of the network to a desired optimal output. The weights are the constants of intertwined mathematical operations inside the network. For the remainder of this report, I shall use different terminology to refer to these terms. For example, the word "model" can be used as correctly as "networks". Please note that models are usually composed of "layers" and that the mathematical operations are usually done within the confines of these layers. Such a term is useful and sometimes necessary to bring higher-level interpretations as networks can be very complicated and intricate. The output of a layer can be referred to as an activation map.

Such networks need to be trained on large datasets representing the type of data you want your model to me efficient on. While training, the network goes through the dataset several times. One pass of the dataset is called an epoch. The standard practice is to use both a training and a testing set. You will see that these sets are used to draw conclusions on the efficiency of the network as will be discussed later.

As opposed to general algorithms designed to execute predefined and immuable tasks, Deep Learning networks are given the ability to modify the computing execution of their predefined tasks by their ability to change their weights. This ability is the cornerstone of Deep Learning methods and is the most valuable quality of these networks. It allows them to optimize themselves for the task they are intended to do.

Self-optimization has the advantage to free the algorithm of any human constraint. An evident constraint of this type to be thought of is the time required to elaborate the algorithm. Because the clear reasoning pathway for a given task can be incredibly hard to explicit for humans (e.g. in the field of image classification: how to recognize a cat or a dog?), Machine Learning aims to get around this problem by automating a majority of its steps. The time required to design these steps is then included in its training and can be ameliorated through better hardware or clever computing techniques. This brings light on another and perhaps major advantage of these methods: the algorithm is free to take any path it chooses in its abstraction to be optimal. Whereas (as cited above) traditional algorithms have human-designed steps for the realisation of a task, Machine Learning models are able to create their own and unique abstract reasoning in the context of a specific task. While these steps might at first glance be non-sense for the human mind, they form a complex abstractive network that enables them not only to perform exceptionally well (better than humans) but also to have a tremendous generalization potential (i.e. extend its understanding of concepts beyond specificities of the data it uses for training). These methods can also be an

incredible source of inspiration in the understanding of problem solving and abstraction. This last point is an active field of theory in mathematics and could very well have applications in the fields of learning.

Machine learning has experienced significant progress in recent years, benefiting highly from the overwhelming amount of data being produced and accessible in a lot of domains. Such progress has allowed tremendous breakthroughs in domains such as image recognition and more recently, image generation. The latter could not exist without the former.

The specific field of Deep Learning that will enable me to conduct my study is unsupervised learning: the study of data with no labels or categories to differentiate inside my dataset. As a contrast to supervised learning where data has to be processed by humans (e.g. categorization), unsupervised learning can be applied to raw data and even sometimes carry the goal of categorizing such data in the place of a human. This problem is at the heart of the project I will be joining. For a quick word on unsupervised learning, its theory relies on finding underlying structures inside the data and understanding unobserved probability distributions.

One milestone in the development of the field was the idea to use convolutions (4) in Deep Learning models to address the problem of an increasingly important number of parameters. This method introduced by a network known as AlexNet hinged on the use of filters (also called kernels) passed as convolutions on an image to detect oriented edges and, more generally, shapes. This technique only needs as many parameters as there are pixels in all the filters to be applied. This is a clear contrast to the then state-of-the-art models that relied on “fully-connected layers” (also called dense layers). A type of layer much inspired by the physiology of neurons in the brain. They were composed of computing cells called “neurons” that were each interconnected with previous and next layers. In order to give this kind of network more abstraction capabilities, it was required to either give the layers more neurons or to add layers. This addition is referred to as “deepening” the network (hence the term “Deep Learning”). Because of the nature of dense layers, the number of parameters could rapidly increase to overwhelming numbers. For example: say we decided to have a network composed of 3 dense layers, each composed of 4096 neurons, then the total number of parameters would be equal to $4096^3 = 68.7 * 10^9$ parameters. In other words, this simple network would have more than 68 billion parameters.

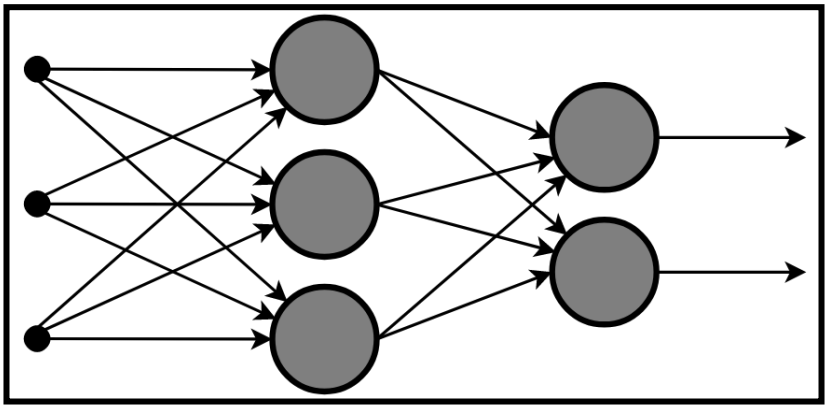

Figure 2: Multi-layer Neural Network representation (5). Each grey circle represents a neuron and each black arrow represents a connection between two neurons.

The abstractive power of dense layers cannot be directly compared to the one of convolutional layers as the two techniques are totally different. Thus, we cannot define a clear equivalence between the two networks by saying for example that a 3 layer dense network is equivalent to a 1

layer convolutional network. The stochastic nature of the optimization problem that is to find this optimal set of parameters that produces the minimum value for the loss function is preventing us from being able to make such comparisons even within the same category of networks. Nonetheless, there still exists quantitative values that bear meaning on how "well" a network performs, or even how efficient it is. These values are often the training time, the loss function value at a given epoch, the difference between training and testing loss value and a few other more problem dependent.

1.3 Exploring the frontier as a motivation

The interdisciplinary potential of Machine Learning is infinite and has already triggered small revolutions in many fields. I spent last year at University of Montreal as an exchange student, studying physics and mathematics. I came across Machine Learning very briefly in one of my courses and was immediately captivated by the idea. I was already looking for an experience at the frontier between physics and computer science and this really comforted me in my choice. This field of research is incredibly new and blooming and this carries for me a lot of motivation to take part in its exploration. This motivation carried me through a Stanford course of computer science on Machine Learning (6) that I took on Youtube before the beginning of my internship. This course has helped me a lot on the basics of Machine Learning and is still to this date a toolbox for me to go back to in order to refresh my memory.

This internship will have as a final goal to be able to produce realistic images of galaxies using Variational Auto-Encoders, a type of Deep Learning network able to generate images. In this matter, a blog by Agustinus Kristiadi (7) was of tremendous help for my understanding of the theory. As will be thoroughly explained in the next section, the motivation for this project blooms from an interdisciplinary perspective at its core. It would constitute a stone in a bigger edifice that could help a lot of scientists worldwide in a lot of fields inside the realm of physics and even further. I also wish to bring to the attention of the reader that my motivation for diving in the research on Machine Learning doesn't solely depend on its possible applications but also for what I would qualify as the intrinsic beauty of such methods. The very nature of how the theory behind the field is constructed and the meaning and interpretations it carries represents what has captivated me and, I believe, so many others. I would also add that this particular drive is what in the end constitutes what I love about science and hope to convey in this report.

2. Context

2.1 Two contestants for generation

In recent years, two candidates have led the race to be the most efficient type of network for image generation: Variational Auto-Encoders and Generative Adversarial Networks.

Generative Adversarial Networks (GANs for short) are a type of network based on game theory composed of two smaller networks inside a bigger network.

- The first network is in charge of generating images based on a training dataset. His goal is to generate samples of images that will be realistic enough to trick the following network into thinking they are real.
- The second network is a discriminatory network that takes two inputs: a real image and an image given by the generator network. The goal for this second network is to differentiate which image is real and which is a "fake".

Both networks are then trained in concert. The theory behind this type of architecture is that the contrasting goal for both of these networks will work as a drive for their cooperation to be always more refined. In practice however, the unusual structure of this type of networks can result in a complex training process with multiple arising instability problems.

Variational Auto-Encoder (VAEs for short) on the other hand offer a much easier training process. They are as well formed by two networks:

- An encoder network (or generator, or inference network). The base principle of its process is to "encode" an input image into a latent vector **z** supposedly representing the key features of the image
- A decoder network that then decodes this vector back into an image with a similar processing algorithm mirroring the encoder.

There lies, however, a specific detail that differentiates them from what would then be a normal auto-encoder: they rely on Bayesian inference to learn the unobserved distribution of the images of interest in both image and latent space. In practice, the encoder network will thus not produce a vector directly but rather parameters to a statistical distribution (say a Gaussian) from which the latent vector would be sampled. These networks suffer very mildly from the instability problems of GANs and offer both reliable training and, in my mind, a very interesting insight into the theory of image generation.

The state-of-the-art in galaxy image generation in the field of cosmology tends to favor the use of GANs over VAEs at the time of writing. However, studies (8)(9) suggest that VAEs could compete at the level where GANs stand, provided with some adjustments and enhancements that are still for the most part an actual field of research. As we will see, these adjustments can be fairly minor.

2.2 Generation and its advantages

Another motivation for the generation of realistic galaxy data from Deep Learning models emerges from the intrinsic versatility of these models and their ability to control parameters that impair drastically the quality of the real survey-based images. Among those parameters could be named:

- The Point Spread Function (PSF). This phenomenon is unavoidable with the use of telescope images but can be accounted for with the use of numerical techniques.
- The presence of artifacts such as foreground stars or other background galaxies.
- A similar potential exists for noise, a decisive limiting factor of modern imagery instruments that translates the limits of their resolution capabilities. This appears to be a very active area of research (10)(11).

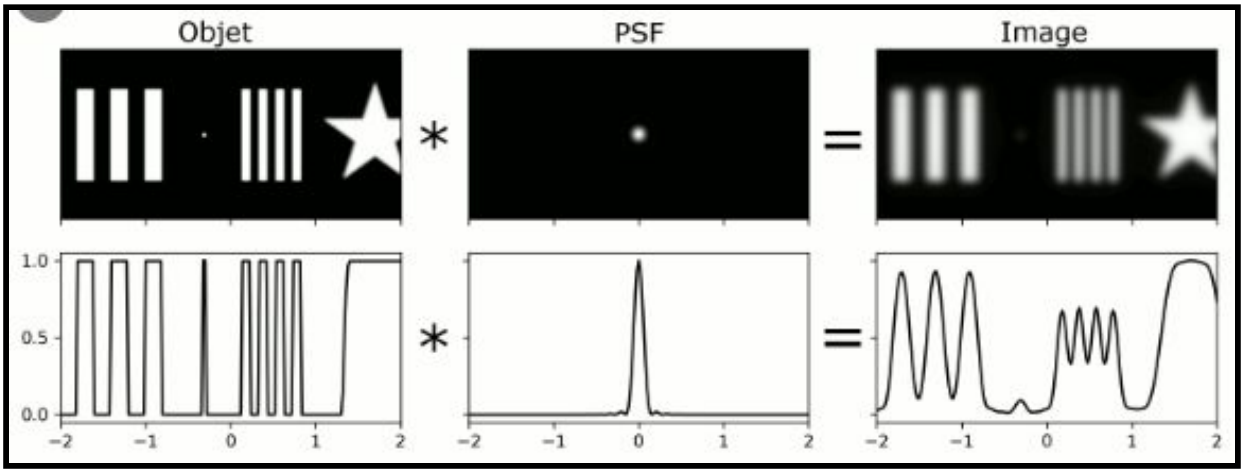

Figure 2: Representation of the Point Spread Function (PSF). (12)

Coming back again to the strong generalization potential that Machine learning offers in this matter, these problems can be alleviated by generating conditionally, meaning one could train a network to treat the presence of artifacts, PSF or noise as a parameter to be tuned. Therefore it appears feasible that images could be willingly sampled with or without said characteristics, even with the luxury of tuning their expressiveness.

2.3 Strong Lensing

The specific topic of cosmology I am interested in is gravitational lensing. This physical phenomenon occurs as a result of the action of gravity on light. As known since the theory of general relativity formulated by Einstein, light is susceptible to be deflected in its trajectory much like one can imagine for moving objects. This deflection, however tiny, is always occurring around objects that have a mass. When dealing with massive celestial bodies like galaxies or clusters, these deflections can become significant and visibly noticeable. This change in direction of the light causes the observer to perceive background image light as coming from somewhere different from its actual location. As a result, background images are perceived as deformed and bear visuals resembling ones from images that passed through an optical lens, hence the term “gravitational lensing”. This distortion can be really important and even go to the point where multiple images of the source are produced, it is then referred to as “strong lensing”.

This effect can be studied in many different ways as it gives direct insight into the workings of relativity and mass-light interaction. One notable example of an application is for the prediction of where and when a supernovae was to appear in the sky (13). While we wish our understanding of theoretical stellar physics could enable us to make such accurate predictions, this was actually done with the use of the study of strong lensing and its theoretical properties. Researchers observed 4 images of the same supernovae in the sky. A supernovae happens at the end of the life cycle of massive stars where they collapse and eject violently their inner layers of plasma and gas. It is a very

bright and thus visibly noticeable event. Because of the dilation of time when light passes a massive object, they were able to predict where and when the same supernova would appear again in the sky with its light coming from another angle as the one they previously observed.

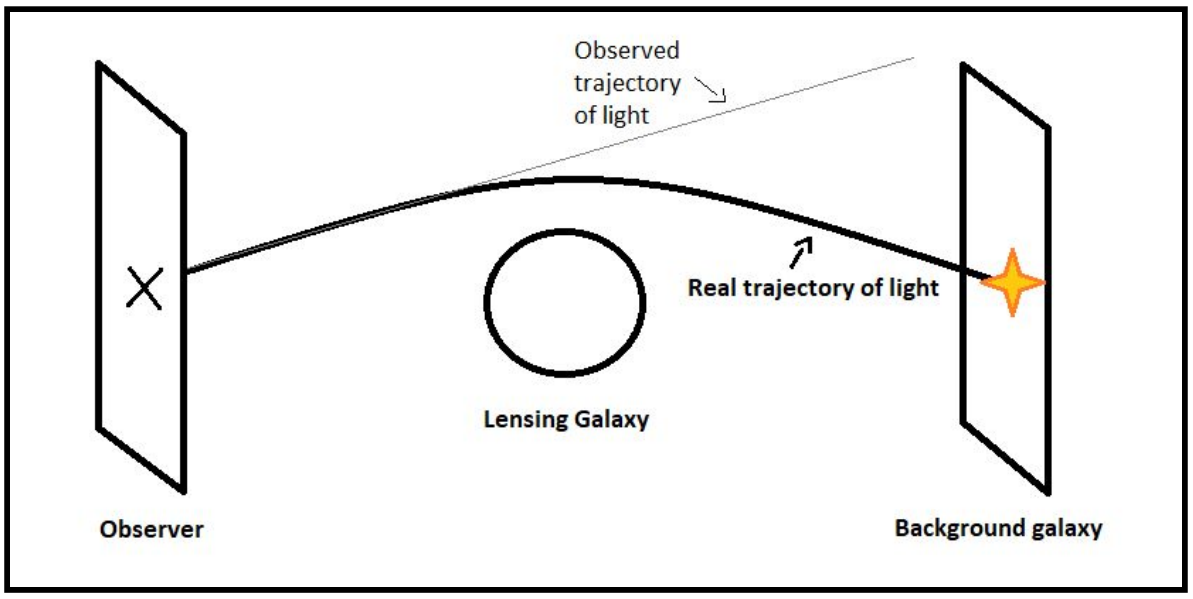

Figure 3: A schematic representation of the effect of strong lensing on light and observation. The massive galaxy bending light is called the lensing galaxy. We define a foreground (lensing galaxy) and a background (lensed galaxy) for convenience.

The research team I joined is more focused on the study of how gravitational lenses modify an image and their physical properties. The final goal of the unit of Laurence Pearrault-Levasseur and Yashar Hezaveh is to create a Machine Learning pipeline for strong lensing. The idea, in terms, would be to input any strong lensing data to this pipeline such as images of strong lenses and retrieve as an output parameters or data that researchers might be interested in. In theory, any given parameter could come out of this pipeline as long as it was trained for it. *Realistic images of galaxies that simulate accurate density distributions, stellar light and velocity fields could be used to break the mass-sheet degeneracy in H0-estimation applications of strong lensing.*

Its motivation primarily comes from the future large-scale surveys of the night sky (such as LSST (14) or Euclid (15)) which will produce very large quantities of real galaxy images. However, such data comes out in the form of simple images of astronomical bodies and their analysis would remain to be done. Such analysis is done in order to extract more data from these images. In the case of strong lensing images, this extracted data can be for example parameters of the lens, the mass distribution of the lensing galaxy or even the reconstructed background (lensed) galaxy. The ideal scenario would be to have frameworks ready by the moment data from these surveys will be accessible to classify and extract interesting parameters from this raw data. This could allow scientists to benefit from two major advantages:

- First, the technical analysis of the images could be totally automated and would relieve scientists of the burden of making these themselves. This could save some precious time for the scientists in charge of said analyses (as they can be pretty long to do, even more considering the quantity of data) but also for the entire scientific community to have access to a very large amount of data as soon as the surveys begin.
- Secondly, such a pipeline could be tuned in advance to output desired parameters. Designing this in advance of the surveys could allow us to take into account as many suggestions as possible for what type of data could be collected by the pipeline. This, in

term, would mean that scientists from ever so different fields would benefit from the work we do and take full advantage of the upcoming surveys.

However, such a pipeline requires training to be alimented by data as realistic as possible in order to be efficient on actual, real life data. This realistic data has to be produced by state-of-the-art techniques that could account for all the intricacies I explained in the introduction on the complexity of galaxy light profiles. Such promising techniques only come to my knowledge at the time of writing as a number of two. The first being large-scale simulations of the universe from its formation to a given epoch. TNG (16) is an example of such simulations that accounts for a large amount of physical processes vital to the accurate modelisation of realistic galaxies. The approach I decided to take however is through Machine Learning and the generative methods previously explained.

3. Material and Methods

A major part of my internship was focused on learning Machine Learning from bottom to actual applications to my field of interest. As a result, much of my time was spent reading articles, guides, tutorials and blog posts about the subject, experimenting with the tools I had. The process of trying and failing in computer science can be messy as a big part of the work is solving little problems such as syntax or logic. While this aspect of the research is unavoidable and normal, very little interesting information can actually be extracted from this process in the form of defined tasks and experimental protocol. However, we compensate for this in the understanding we make of the theory and the elaboration of efficient techniques and habits in the making of my programs. When it comes to restituting this experience, it may seem unclear as to what the day-to-day tasks might have been. As an explicit laying out of such activities would be confusing and unrepresentative of my work, I decided to take another route. I wish to give an overview of my conclusions and findings at different stages of my project and a depiction of my productions in the form of what could be close to an experimental methodology. These conclusions and numerical recipes of some sort will by their completeness and accuracy, I hope, give a truer picture of my implication in this project.

3.1 Python programming and hardware

As a numerical tool, Machine Learning requires the use of programming in order to write and operate networks. The first step to take is to choose from a wide variety of programming languages. Each language has its proper formalities mainly in its syntax but a lot of them are completely suitable for Machine Learning. Because of my previous studies, I grew very familiar with Python and decided to continue using it for this project. I decided to work with the Tensorflow (17) library along with the Keras API. A library is simply a collection of predefined functions and commands to use for certain types of tasks such as Deep Learning. APIs, however, grant a practical interface for the use of the library. Other libraries exist for Machine Learning in Python that differ in their format of writing. Tensorflow has a very object-oriented approach that seemed easier and cleaner to use for me.

The necessity of libraries and APIs resides in the convenience of their definitions. Because a single network is composed of a variety of different steps when it comes to its definition, its operation and its computing, using a library can allow for a much quicker access to building and training actual Machine Learning models. A library contains a lot of the functions and objects proper to Machine Learning problems that I would otherwise need to create myself. Without the help of the Tensorflow library, my work would have been drastically impaired by the development time needed for the definition of such things.

All the code I produced will be available on a GitHub repository (18) I created for this project. Github is a website designed to store and share code and related folders. It is a wonderful tool that plays a major role in collaboration, accessibility and reproducibility of science. I wish to ordonate this repository in a few different sections regrouping the code, the images produced and the explanation of how my project was structured.

Because of the nature of the computations used in Machine Learning networks, computing times can be drastically affected by the type of processing unit they are done on. I used my personal computer for the entirety of my internship so far and ran the computations on my Graphics processing unit (GPU) as these kinds of computations benefit greatly from being spread across a lot of different processing cells with low processing capabilities rather than a few very powerful ones.

My project can be separated into three different stages:

- First of all, I had to familiarize myself with the tool I would be using to write and train neural networks, namely Tensorflow and Keras. As the only knowledge I had on Machine Learning at the beginning of my internship was theoretical, I had to transfer this into practical knowledge with the help of the realisation of a simple task: the building and training of a Convolutional Neural Network (CNN) for image classification.

- Secondly, I refined my mastering of the subject by diving into the core project: image generation. From there I went through publications and blog posts about VAEs in order to understand how they worked and how to build one myself.

- Lastly, I looked for inspiration on how to ameliorate my previous results with novel techniques that could be applied to basic VAEs.

3.2 Image classification with a basic CNN

The major part of my inspiration was found in the Stanford youtube course for the theory and in the very well designed Tensorflow guides for the practice. When dealing with Machine Learning, the central aspect is networks. It appeared right to me to then start with the definition of a basic CNN architecture. To achieve this, I needed to first think about what I wanted my network to do. I had a dataset of 70,000 grayscale images of articles of clothing, each image having a size of 28x28 pixels. This dataset (the Fashion MNIST dataset) was separated into 10 categories that are as follows:

- T-shirt/top
- Trouser
- Pullover
- Dress
- Coat
- Sandal
- Shirt
- Sneaker
- Bag
- Ankle Boot

My network was to look at images in this dataset and make predictions about the category a given image belonged to. I decided to opt for 4 convolutional layers, composed respectively of 16, 32, 64 and 128 4x4 filters, followed by a 64-neuron dense layer and finally a 10-neuron dense layer. The idea was that the input image would be convolved at 3 different layers then analysed through dense layers to finally give out a score in each of the 10 categories as the activation map of the 10-neuron layer.

We can take a minute to think about two, very connected, of these numbers: the choice of dimension for the filters and the depth (number of layers). Because I want my network to detect shapes in the image, it is important that it will be able to capture shapes up to the same order of magnitude as the object of interest. It is easily understood as to why this aspect is important when thinking about all the parts of an image that could define what it represents. For an image of a cat for example, the general outlined shape of the animal could be interesting but also smaller ones such as the shape of the eyes or stripes it could have on its fur. As a result, I need to make sure that my network can run convolutions that have an effective size in the same order of magnitude as my

object. Because of the downsampling aspect of the computations done, this need not to be done with wider filters. Smaller filters, applied on deeper layers could achieve the same effect as bigger ones applied in earlier layers as the area they will effectively cover, when thought of in the context of the input image, will be bigger than their actual size. In my case, the image is downsampled by a factor of 2 at each layer because of the pooling layers. Because I have 3 of such layers, and because the input images are 28x28 pixels, I downsample the image 3 times by a factor of 2, resulting in a layer activation map of a size of 4x4 pixels before the last convolution (the last pooling being applied after the last convolution). Because my objects usually take up the whole image, I can be comforted in my choice of size for the filters.

It is worth noting that no real guideline exists on the exact choice for all the numbers cited above. Apart from the qualitative example before, they could theoretically all be considered as hyperparameters and depend highly on the task. This however can be quickly very overwhelming. In reality, there are not that many hyperparameters that actually have a deep impact on training and this justifies, in the end, the importance of a good choice of hyperparameters. We have to keep in mind however that this search is much more often qualitative than quantitative. There is no single number for a given hyperparameter that represents an absolute optimal choice as there exist many different local minima in the parameter space that are roughly equivalent, as the literature suggests (19). The idea is to give insights into what those hyperparameters should approximately look like.

Such important hyperparameters are in this case the sizes of filters and the depth of network that can be determined as described above. Another example could be the starting learning rate because this may influence greatly how the parameter space is searched. To understand the learning rate, we can think of the parameter space we want to optimize as a 2D landscape with hills and valleys, the height of the ground representing the loss value of my network. To optimize the network, we would need to walk to the lowest part of the landscape. In this analogy, the learning rate would represent the size of the steps we took to go down a certain path. This is in direct link to the principle of an optimizer that will be defined later.

Standard practice in Machine Learning is to conduct this hyperparameter search with the help of different splits of my data, called datasets. I would need to have 3 different datasets: training, validation and test datasets. This first one would be used for actual computations of gradients and backpropagation. In brief, this set is used by the network to understand the data and try to refine its efficiency. The next dataset would be used to compute the loss of the network after training given a certain set of hyperparameters. Repeated application of this process with different sets of hyperparameters can then be quantitatively compared to decide which to be chosen. Finally, the test dataset would only be used at the end of my training process, after the hyperparameters have been set and the network is trained. The loss result produced by the forward pass of the test data through the network constitutes an insight on how well the network would perform on unseen data.

When thinking about the difference between the input and output of the network, it appears pretty clear that some kind of shrinking occurs to go from a 28x28 image to a vector of size 10. Such shrinking indeed happens and is called “pooling”. Different techniques exist to achieve this but the result does not depend, in this case, on the exact type chosen. As a consequence, I decided to use MaxPooling, a technique that looks at local maximums and shrinks the image representing only these maximums. What’s called pooling layers are inserted in between each convolutional layer to down sample the processed image. Pooling is an essential step for Deep Learning networks as it forces them to focus only on a smaller part of the information received in input.

Now that the network is built, I need to import the dataset on which my network will be trained. Fortunately, Keras gives easy access to the Fashion MNIST dataset defined above. A simple line of

code can import all the data in a vector that will be stored for later use. A contributing factor in the efficiency of networks is what I call “data preprocessing”. As the name suggests, I modify the raw data to firstly be understood by the network but also for optimizing its treatment. I applied batching to my data, separating it in batches of 32 images. This allowed the network to look at 32 images and record all 32 gradients before applying their sum on the weights. This saves valuable time with large datasets as compared to applying gradients for each image that passes through the network. Because my data pixel values are integers, I normalize them between 0 and 1. This step is crucial for the network to understand the data as it cannot treat integers. Lastly, I separate my data into a training set validation and test sets respectively of 50,000, 10,000 and 10,000 images.

For this network to be trained, I need to define which optimizer and loss function will be used. An optimizer defines how the gradients are applied to the weights with a mathematical function and has for parameter the learning rate (and a few others in some cases). For this I chose the Adam (20) optimizer, a common practice that gives minor optimization problems. The loss function is much more problem-dependent than the optimizer. Because my problem is to classify in 10 categories, I chose to use the Sparse Categorical Cross-entropy as a loss function. This will compute the cross-entropy between the output vector and the real classification vector (encoding to which category an image belongs to) for every image.

The network is now ready for training. Knowing that I would need to dive into lower-level handling of the training in future networks, I chose to write myself the training loop, as opposed to using pre-written training functions of the Keras API. The training is defined as follows:

1. The network takes a batch of images as inputs.
2. It computes what’s called a forward pass to output a prediction vector for each image.
3. It computes a loss, thought the loss function, for every prediction vector and an average of all the losses in the batch.
4. This average is then passed to the optimizer that will backpropagate it to the trainable weights in the network.
5. Each 200 steps (a step being a whole batch computing), the network computes the forward pass and loss for the entirety of the test set. However, it doesn’t propagate this loss to the trainable weights. Before continuing to step 6, it prints the average value of the training loss so far and the test loss to give an idea as to how the network is training.
6. When all the training set has been passed into the network, an epoch has been completed. The network shuffles the dataset and starts again at step 1 as many times as I defined the number of epochs to be.
7. Each of the values of batch losses and average test losses are stored to be used later for plotting graphs.

As I quickly understood, many tricks exist to enable the network to train faster or better. One of those tricks is called regularization. There are again many different ways to do regularization. One of them is to apply batch normalization. In between each convolutional layer, I normalize and scale the activation maps of the previous layer by some average and variance across the batch. This has the effect of granting the network a better generalization potential as now, each activation map integrates a little bit of all other activation maps in the batch, effectively reducing its peculiarities and enabling the network to better focus on bigger aspects of the data as a whole.

3.3 Image generation with Variational Auto-Encoder

Now that I was a bit more familiar with Tensorflow and had for the first time applied my theoretical knowledge, I went on to a more difficult task: building and training a VAE.

The first major difference would be the dataset I was training my network on. I could then start working on images of galaxies, more specifically the GalaxyZoo dataset. This dataset is composed of RGB images of 424x424 pixels. This means that each pixel has 3 channels (Red, Blue and Green) that encode the colour of the given pixel.

I first had to download this dataset myself on the GalaxyZoo contest website, Kaggle (21). However, the images were mainly composed of black space with occasional artifacts and the actual galaxy only making up a small portion of the total image. Because of this aspect and the size of the images, I had to crop and resize the whole dataset in order to reduce the influence of undesired artifacts, help the network to focus more on the galaxy itself and at a smaller resolution as a starting point. A band of a hundred pixels was removed from the edges of the image and then reduced to 64x64 pixels using the OpenCV library, designed to help with computer vision-oriented problems such as this one.

Figure 4: An Image from the GalaxyZoo dataset. On the left a 424x424 original image in the dataset, on the right the same cropped and resized image to a resolution of 64x64 as described in the previous paragraph.

Even with rescaled images to 64x64 pixels, this dataset still represented a much bigger space on my computer as compared to the Fashion MNIST dataset. As opposed to MNIST which I loaded into a vector that I could get my batches from directly, I had to go into my data folder for every batch and create a temporary vector containing the images of the batch. This resulted in a small bottleneck for my training time as the many readings on the disk took an irreducible portion of training. For the reading and formatting of my dataset into a vector, I used the matplotlib.image library.

Another major difference from the last task is the approach itself. The task is now not to classify the images into categories but to generate new images. This required the loss function to be totally different since I would not be comparing vector encoding classes but actual images. A proper tool for such comparisons is the Mean Squared Error (MSE), an average of the squared difference between each corresponding pixel in the two images. This method, which can be thought of as the square of the Euclidean distance between pixels in the image space, benefits from a lot of theoretical

motivations. As explained in the introduction, VAEs take an image as input and outputs a reconstructed image. These are the images that are compared for each step in the loss function.

In addition to this loss function giving an insight on the reconstruction loss, VAE theory requires the definition of another loss: the KL-divergence loss. This will compute, as it states, the KL-divergence between the statistical distribution defined by the output of the encoder and a unit Gaussian function (Gaussian of zero mean zero and unit variance). For the computation of gradients, these two losses will simply be summed before they are passed to the optimizer.

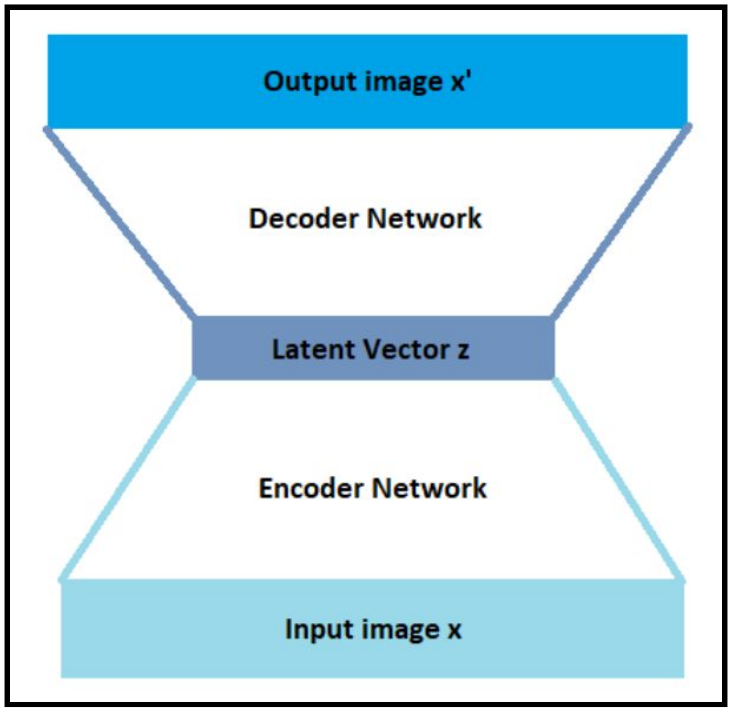

Figure 5: Representation of the architecture of VAEs

Because of the shape of the VAE network, it was more convenient to define the encoder and decoder networks separately as layers that would then be added inside a bigger VAE network. To design the encoder and decoder, I had to do so using the "layer" class of the keras API. This pre-built class contained a lot of the automations needed for it to work. The only thing for me to do was to define the architecture of each model and the order in which it would do its computations when it would be called by the VAE.

The architecture I decided to adopt for the encoder was at first a 3 convolutional layer followed by a single dense layer for each parameter of the output, as vectors, statistical distribution parameters (here of the number of 2, the mean (μ) and variance (Σ) of a normal distribution). These vectors have the same dimension as the latent vector **z** and have independent dimensions. However, because of the level of accuracy I wanted to have for my generated images and on advice from my teaching tutor, I decided to change to a deeper model of 8 convolutional layers followed by 2 dense layers for each output.

The decoder network has a mirrored architecture to what was presented above and introduces deconvolutional layers. These are the exact "opposite" computations compared to convolutional layers. What links the two networks is a sampling function that takes as inputs the mean (μ) and variance (Σ) from the encoder and samples a latent vector **z** from a gaussian distribution.

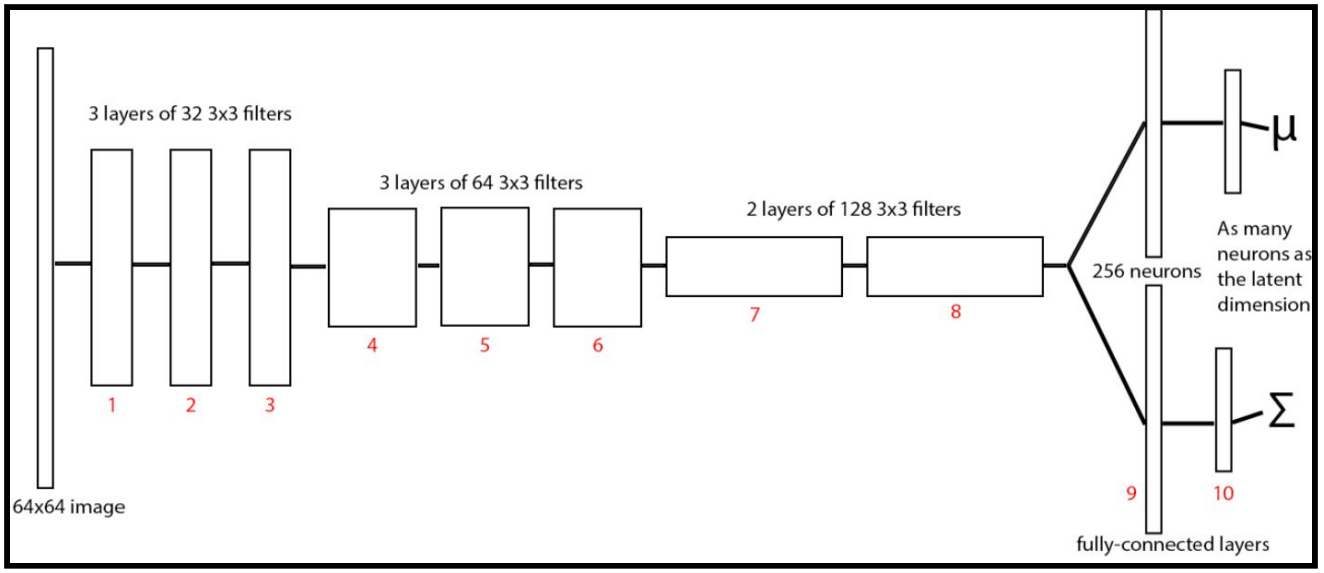

Figure 6: Representation of the encoder network for better visualisation. In red, the number of each layer. The width of rectangles represent the depth of the activation functions created by the corresponding layer. The height of a rectangle represents the size (height and width) of the activation maps.

As can be seen on figure 6, a kind of shrinking is applied again at certain points in the network. This, as opposed to the previous task, is not done by the use of MaxPooling but with the use of strides. This simply means that instead of moving the filters over the image one by one pixel, for a striding of 2, the filters will move 2 by 2 pixels. This striding parameter is applied at 3 different stages: the very first layer, the 4th and the 7th. The only reason max-pooling was used before was that its operation is easier to understand and thus better suited for a beginner network. Striding, however, allows for better accuracy as it conserves spatial information regarding the features (22).

Another important feature of the structure of neural networks is the choice of activation functions that are applied at the exit of each convolutional or dense layer. Activations are a key pillar of the mathematical theory behind Machine Learning models. They are nonlinear functions that are indispensable because they prevent the linearity of all the other chained computations to otherwise collapse to a single linear function. I was previously using the ReLU function for this matter, which is a very good first choice when building a network. However, because such function prevents the gradient from flowing through it in some cases, training could suffer from its use. For the VAE, I switched to using Leaky-ReLU functions, an amelioration of the previous ReLU that accounts for the cited problem it poses.

The last piece that was added was an embedded function inside the VAE to be called for generating new images. This worked by sampling a random latent vector **z** from a normal law of mean zero and variance 1. Because this vector can (and should) have more than one dimension, this process is done for each of its dimensions. This new vector is then passed into the decoder network that will convert it into an image. To make sure this image is composed of pixels of values ranging from 0 to 1, I normalize the output image by its maximum pixel value. This generation is done 16 times in order to show multiple samples of generated galaxies in a 4 by 4 manifold that is then stored in a dedicated folder.

One of the only few hyperparameters of VAEs is the dimension of the latent vector **z**. I conducted this search once the previous steps cited above were done as a way to refine the accuracy of the generated samples.

As my training process got more precise and efficient, I designed new, deeper networks that would work on higher resolution images (128x128). This was a logical step into closing the gap between a beginner model and a more serious model that could produce sharp images.

3.4 Enhancements for the VAE

The search for a way to alleviate the major problem of VAEs, blurriness, was a long and thorough endeavour in the form of reading articles and discussing with my teaching tutor. Because my understanding of the mathematical theory behind VAEs was meager, I had to educate myself on Bayesian statistics before even starting to deal with practical enhancements. My subsequent research into the literature had brought light on two potential techniques to achieve my goal: normalizing flow and double staging.

Both techniques could give similar results and aim to achieve the same thing in theory. However, their application is completely different, with the normalizing flow being (I thought) way harder to implement. As a consequence, I decided to use the second technique found in a 2019 paper presented at the ICLR (International Conference on Learning Representations) cited in introduction (8).

The elaboration of a "Double-Stage VAE" changed drastically the way generation of images was to happen while conserving the shape of a traditional VAE that I had familiarized myself with. Simply put, the idea was to separate the training process into two stages:

- The first being a normal VAE training as described in section 3.3.
- The second would consist of a smaller VAE, with a different latent vector called **u**, that would not train on images, but on latent vectors.

In the same fashion as for the CVAE, the dimension of this latent vector **u** can also be treated as a hyperparameter.

For convenience reasons, I shall refer to the first VAE as the "Convolutional VAE" (or CVAE) and the second as the "Latent VAE" (or LVAE). The first being the same network that has been defined at section 3.3 and the second a new network with a different architecture, this time not composed of convolutional layers but dense layers.

When the first training stage had been completed, I had to create a new dataset derived from the image dataset that was actually the same dataset but passed into the encoder of the CVAE to produce a dataset of latent vectors **z**. This dataset was then used for training the LVAE in a much similar fashion as the training of the CVAE.

The LVAE, working on vectors and not complicated images, needed not to be as complex and deep as the CVAE. While convolutions have a clear interpretation when applied to images, their use for the study of a vector is harder to motivate. As a consequence, I chose the architecture recommended in the paper, being a 3 dense layer with 1024 neurons for each layer. It is worth noting that, despite my research being done on images as for the one conducted in the paper, the data I am using (galaxies) is not as complex as the one used in the paper. This could indicate that the complexity of the network could actually be even smaller, for example in its number of neurons per layer. This theory however was not tested as my latest results with this network are still in their earliest stages, coming way before such optimization would actually be indicated.

I struggled for weeks with a problem usually only attributed to GANs: something that resembled mode collapse. This problem is defined as the network choosing to generate only one single type of data. In my case, the LVAE would never produce latent vectors that would give anything else than Gaussian noise when passed into the CVAE decoder and retrieved as a reconstructed image.

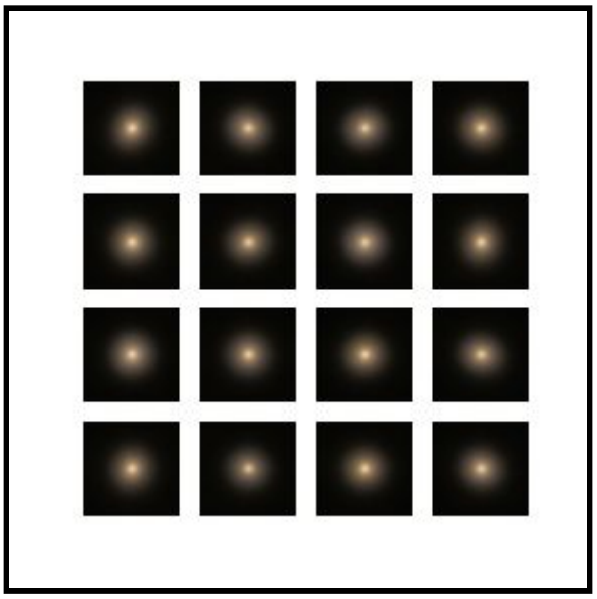

Figure 7: 16 generated images (128x128 pixels) from the Double-Stage VAE that depict the generation problem.

This major problem forced me to review thoroughly again the basics of VAEs and their theory. Fortunately for me, the authors had linked their github repository that contained all the source code used to write the article. Even though the code was written in an older version of Tensorflow and differed quite a bit from the syntax I was used to, I was able to understand the precise steps taken by their network. This, in term, allowed me to understand the differences between what the intended application was supposed to be and what the code I had written from my understanding of the paper.

As I evoked earlier, relevant results have been produced only very recently and haven't been processed and fully understood yet. My thoughts on the matter will be explicitly laid out in the result and discussion section.

This new stage, once trained, offers a totally different pipeline for generating images. Instead of sampling a random latent vector **z** from a unit Gaussian and passing it through the CVAE decoder, a random vector **u** is to be sampled through the same process, then passed to the LVAE decoder to produce a corresponding latent vector **z** that would then be passed to the CVAE decoder to finally generate an image.

This, in theory, allows the network to not only understand which images in the image space are galaxies but also the unobserved likelihood of a given sample. This should grant the network a better definition of what a galaxy is and the limits of their visual representation that reflects, in essence, the real physical boundaries of galaxy shapes.

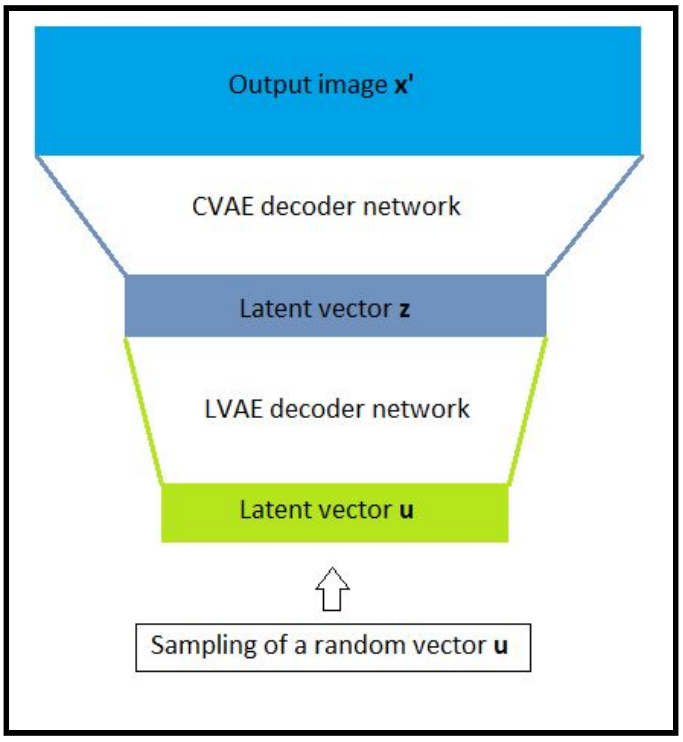

Figure 8: Pipeline for image generation in a Double-Stage VAE.

4. Results and discussion

4.1 CNN performance

As described in the last section, when I was done defining my network architecture, the next step is to conduct a hyperparameter search. Here I targeted the base learning rate for the Adam optimizer.

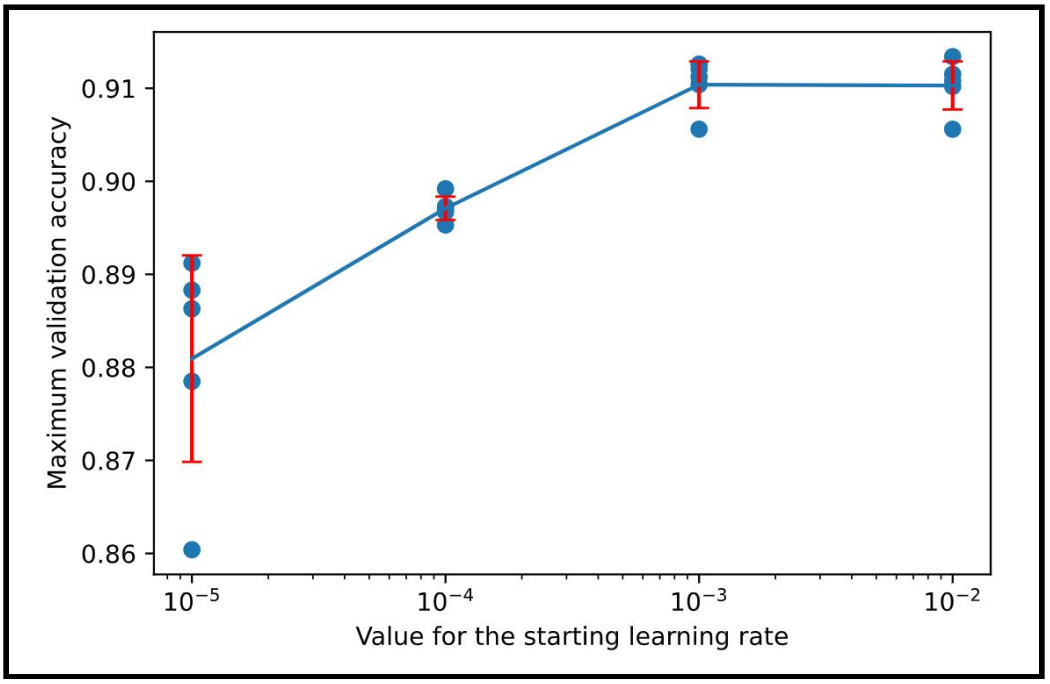

Figure 9: Plot of the maximum accuracy achieved at epoch 5 for a given simulation of the CNN on the Fashion MNIST dataset as a function of the value for the starting learning rate. Each learning rate was evaluated 5 times. In blue the points representing the accuracies, the blue line passes through the mean of the accuracies and in red the standard deviation for the distribution for a given learning rate.

The accuracy is defined as the fraction of correctly categorized images over all the images categorized. This data is very representative of the performance of the network since it gives a direct insight into how often it can manage to be right. This accuracy can be viewed as a percentage of the correctly classified data if multiplied by 100. The accuracy was calculated on a validation dataset composed of 10,000 images taken from the training dataset before training. As a result, I made sure that the network would never treat these images as training examples and never record their produced gradients.

A starting learning rate of 10^{-5} produces not only the lower mean accuracy (0.8806) but also the sparsest results ($\sigma \simeq 0.01$). An interesting result is the standard deviation for a learning rate of 10^{-4} ($\sigma \simeq 0.001$), even if this result doesn't produce the lowest accuracy, it produces very consistent training accuracy. While sometimes overlooked, this quality of neural networks can be important for reproducibility. Because deeper networks could take days and even weeks to train, having such consistency can allow for the training of the network only a few times. Otherwise, I would need to conduct multiple training to be quite certain of the maximum accuracy that the network can produce.

As we can see, a starting learning rate between 10^{-3} and 10^{-2} seems suitable in order to produce a maximum accuracy. We can note however that 10^{-3} seems to produce a slightly more consistent training ($\sigma \approx 0.0019$) as compared to 10^{-2} ($\sigma \approx 0.0021$).

One other possible choice for the learning rate that was not cited in the previous section is the use of a learning rate decay. Instead of having a fixed learning rate, I can define my learning rate to follow a decay schedule such as an exponential decay. Here I tried to explore the impact of such a choice again on validation accuracy.

The decay I chose is defined by the following equation: $X = X_0 * R^{(s/N_s)}$ where X is the learning rate, X_0 is the base learning rate, R the decay rate, s the step (number of batch that passed the network) and N_s the number of step for a decay equal to the rate R is applied. Note that this function is continuous but could also be defined as a "staircase decay" of the rate, decaying by a given factor at a fixed number of steps.

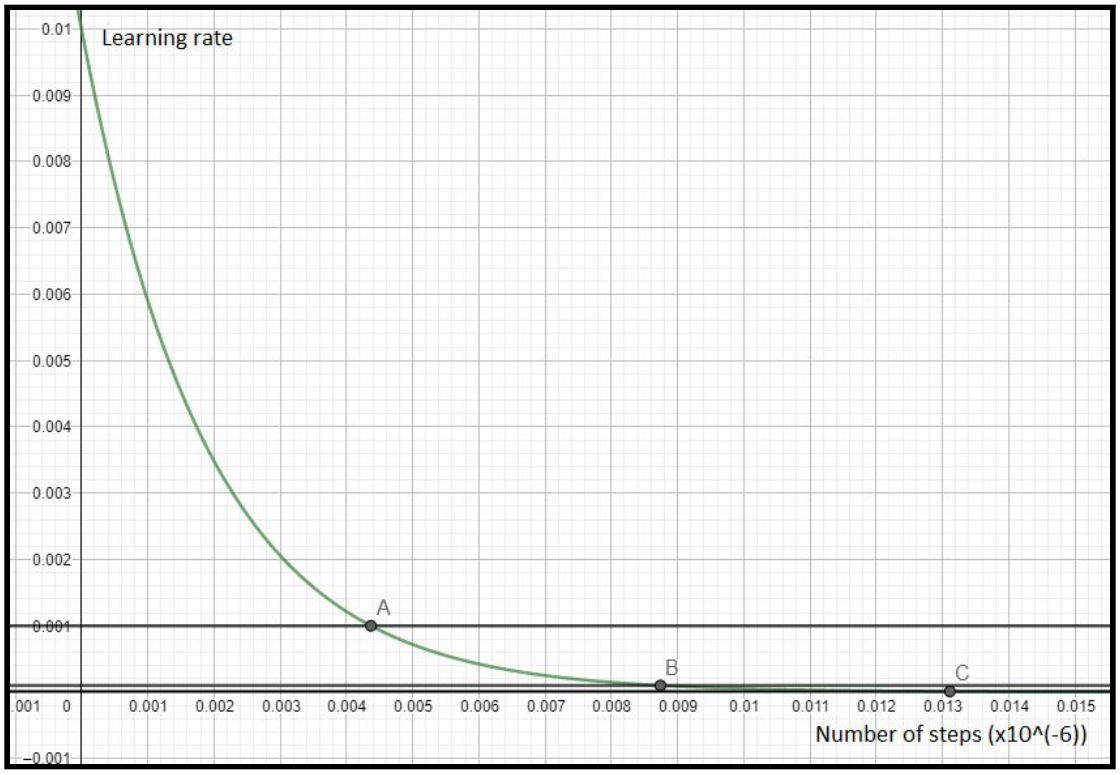

Figure 10: Y axis: Learning rate, X axis: 10^{-6} times the number of steps. This is a graphical representation of the decay function with parameters $X_0 = 10^{-2}$, $R = 0.9$ and $N_s = 200$ done with GeoGebra Calculator (https://www.geogebra.org/calculator). Points A, B and C represent respectively the intersection between the decay curve and values of the learning rate equal to 10^{-3}, 10^{-4} and 10^{-5}.

To motivate the use of such decay, we could think again of the hills and valley analogy cited in the previous section. While we walk the landscape to find the deepest point, we might encounter very narrow valleys. As a result, if our steps were too big, we could run the risk of stepping over the valley and never being able to go down inside it. This is why taking smaller and smaller steps could be desirable.

In reality, the Adam optimizer already accounts for a kind of decay in the steps it takes. This, however, depends solely on the topology of the parameter space it is exploring. Such topology could potentially prevent this optimizer from reaching the best possible minima because of the very way it operates. The decay of the learning rate is thus less commonly used with the Adam optimizer as compared to other simpler optimizers. Nothing, however, prevents us from using it anyway with Adam as complex parameter spaces could benefit from the two techniques working in concert.

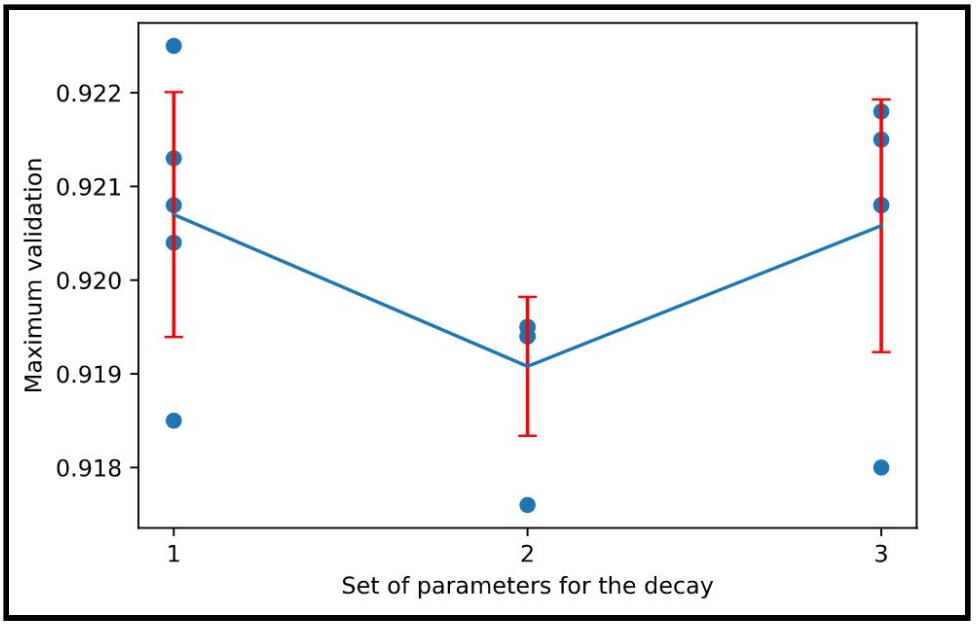

Figure 11: Maximum validation score and error bars at epoch 5 for each different set of parameters for the decay. The blue line passes through the mean for each set. Set 1 corresponds to parameters $X_0 = 10^{-2}$, $R = 0.93$ *and* $N_s = 200$, *set 2:* $X_0 = 10^{-2}$, $R = 0.9$ *and* $N_s = 200$ *and set 3:* $X_0 = 10^{-3}$, $R = 0.93$ *and* $N_s = 200$.

As we can see, in all of the cases above, the use of a learning rate decay offers better accuracy than a fixed rate regardless of the set of parameters used within the reasonable limits around the parameters used here. Qualitatively, the second set of parameters decays much faster than the first one. They start, however, from the same base learning rate. The third set starts at a base learning rate 10 times smaller than the first two but describes the same decay as the first set. Even though the error bar might look bigger than on figure 9, it is worth noting that the scale of the y axis has changed. For the first and third sets, the standard deviation of their distributions are around $\sigma \simeq 0.0012$ while the second is around $\sigma \simeq 0.0008$.

We can conclude from these findings that a decaying learning rate is more often than not best suited for optimization and that a fast decaying learning rate, despite providing a slightly more consistent training, seems to harm the training process. Moreover, learning rate decay allows for some increase in consistency for training.

These accuracy results are very conclusive when compared to the state-of-the-art for the Fashion MNIST dataset. My best result is an accuracy of 92.25% as compared to 96.91% for DARTS (23).

4.2 VAE generated images

A key aspect of the hyperparameter search for Deep Learning models is to conduct such a search on a shallower network and use the results to give a general indication of what might constitute an optimal choice for deeper networks. Thus, the search for the optimal latent vector dimension for **z** was done on a 3 convolutional layer for the encoder and the decoder. While a very similar search was conducted in the last sub-section, I wish to concentrate here more on the results of the generation of galaxy images. The research suggested that more dimensions would always result in better accuracy in term. However, until recently, problems in computation capabilities for the KL loss function prevented me from using more than 20 dimensions for the latent vector **z**. The literature, however, suggests (8) the use of around 64 dimensions for **z** when applied on datasets such as Fashion MNSIT or CIFAR-10. As discussed earlier, the GalaxyZoo dataset carries less complexity compared to those datasets. As a result, the use of fewer dimensions should not have a deep impact on the sharpness of the images.

In the case of VAEs, a few hyperparameters can actually be defined from my previous search on CNNs. Because VAEs are composed of two CNNs, the results I found can give direct insights into what can be a good learning rate to use. The basic construction of the VAE was done with a fixed learning rate of 10^{-3} and was later adapted to a decaying learning rate with parameters $X_0 = 10^{-3}$, $R = 0.97$ and $N_s = 200$. This learning rate covered 3 powers of 10 in the span of 20 epochs to allow a sparse and coarse exploration of the parameter space.

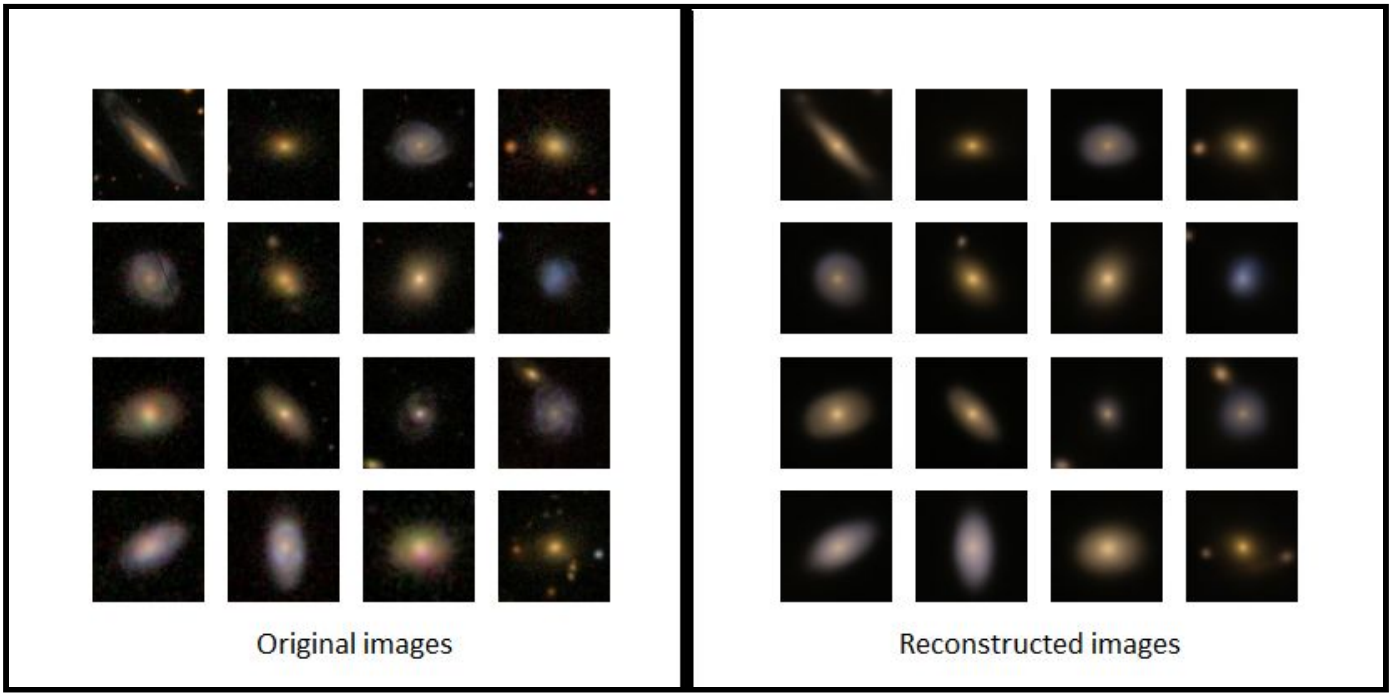

Figure 11: On the left: 16 random original images from the GalaxyZoo dataset. On the right: Same 16 images passed into the fully trained VAE with 20 latent dimensions at epoch 20 with MSE = 61.79 and KL loss = 17.6.

This result can inform us firstly on the abstractive power of my network. As we can see in quite a few images, the number of latent dimensions I am using are enabling us to encode the presence of artifacts and their position. Despite still lacking the power to clearly define small details such as the

arms of spiral galaxies, what strikes the eye is the exceptional capacity of the network to reconstruct the colours accurately. This is the major problem that general mathematical equations have. Other than this, the overall shape of each galaxy seems to also be very well understood, comforting us in the choice of architecture I discussed in theory.

The MSE for this simulation is 61.79 at the end of epoch 20. This means that on average, the network can reproduce each pixel with a value error of $\sqrt{61.79} \approx 7.9$. The KL loss at 17.6 informs us that there still exists quite a bit of difference between the real distribution of the data in the latent space and a unit Gaussian. This result is expected to be ameliorated by the double-stage VAE.

When comparing the final training and testing losses, I observed a difference of only 2.3%. This is a really good indicator that my network is able to greatly generalize and perform well on unseen data.

The next results are totally new images generated from the sampling of a random latent vector **z**.

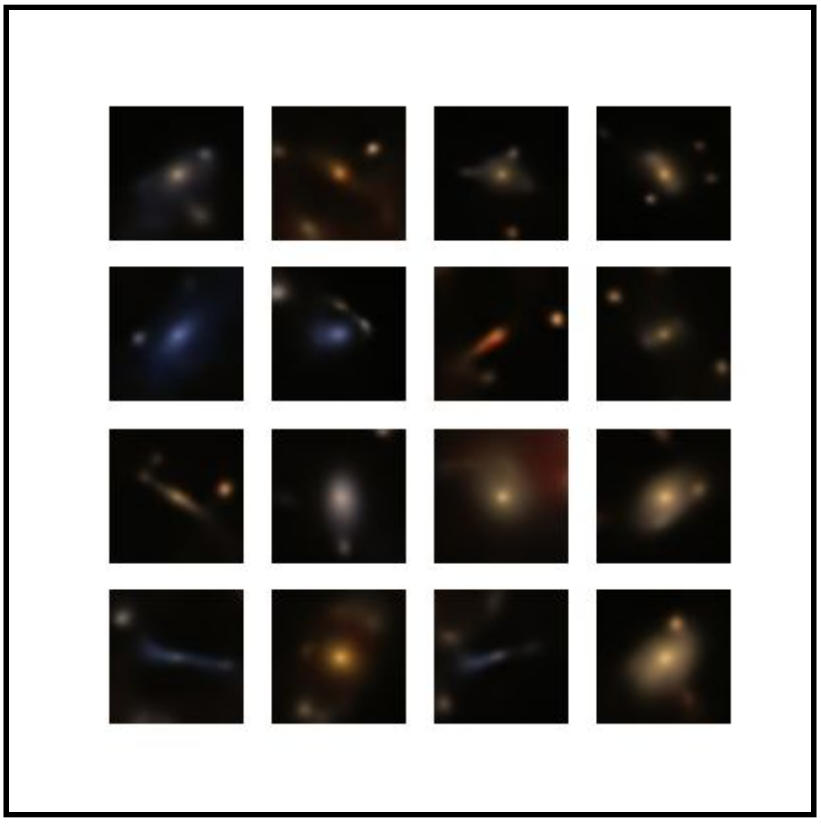

Figure 11: 16 generated images from a random vector z at epoch 20.

As we can see, these images are quite far, for the majority, of clear and sharp images of galaxies. The network seems to have focused too much on artifacts and have generalized poorly what it had seen. As a result, the generation produces messy images with very few shapes that were actually resembling galaxies seen in the dataset. This meant that somehow, my network could reproduce images of galaxies pretty well but failed to generate them from my method. This led me to conclude that my network actually had a pretty bad understanding of what the unobserved distribution of galaxies was in the latent space.

After a discussion with Yashar Hezaveh, a director of my research group, he gave me very valuable ideas on minor changes to make to my network. One of those was the definition of a better

activation function. Because of the definition of ReLU, some computations can lead to an activation of 0. This prevents the gradient from flowing in backpropagation and limits the training process. I defined a new activation function that corrected this flaw in ReLU that enabled negative activations and thus eased backpropagation.

4.3 A qualitative analysis of advanced-VAE generated samples

With minor modifications applied to my network, I was able to very recently produce images from the Double Stage VAE.

Figure 12: 16 generated 128x128 images of galaxies from the Double-Stage VAE pipeline explained in section 3.3.

As explained before, these results (figure 12) are still in their earliest stages, a hyperparameter search for the optimal dimension of latent vector **u** still has to be conducted. However, these results clearly appear way superior to the previous ones. Samples produced are notably more explicit on galaxy shape and much less on artifacts. In addition to this, the images are notably more diverse in their shape and colour than before and give a much better overview of the whole dataset.

While, of course, drawing conclusions from such a small sample is unwise, these images still carry information that indicates I am on the right track to producing samples comparable to the state-of-the-art. Such samples are illustrated in figure 13.

Even if these images have a much better resolution than the 128x128 pixels generated by my network, there still exists a clear difference in terms of blurriness and detail. Spiral galaxies such as the central image in figure 13 have much better definition of their spiral arms in terms of bright and dark spots. Moreover, artifacts are too much better resolved and spread in the generated images.

Figure 13: Generated samples of 424x424 images from the network trained on the GalaxyZoo dataset designed by Lanusse et al. (10), all credit to the authors of the paper. Image adapted from (24).

Conclusion and perspectives

GANs are very powerful tools that produce images that are still very hard to beat in terms of quality and diversity. While VAEs were until very recently slowed by practical hardships in the race for the title of best generator network, they promise to be a very fruitful theoretical object of research for many years to come (25). I have good faith, what's more, in the ameliorations that are yet to be done on my network and hope very much to achieve results comparable to GAN-generated samples within the time left of my internship. Preliminary results of the Double-Stage VAE indicate that this network has a much better understanding of the complexity of galaxy light profiles than a "vanilla" VAE.

As for the project I worked on, much is left to be done. A simple and logical continuation of my project would be to conduct the necessary analysis on the Double-Stage VAE generated data and to be able to generate at the level of state-of-the-art GANs. In addition to this, a very interesting enhancement would be to make my network conditional (26) on galaxy types and morphologies. This could be a tremendous plus for my network that could allow me to pinpoint the type of data I want to generate as a contrast to random generation. One other very exciting perspective and very precious to me would be the elaboration of a network capable of reconstructing the background image in the context of an image of strong lensing. Such a goal might be achievable with the use of a VAE coupled with a network with an architecture resembling the one of AlexNet. This idea for a next project needs this present one to work in order to exist. If VAEs do not produce the expected results, the training of a GAN could be an alternative.

My project was somewhat crucial to the bigger project of my research group. However, the new project I explicitly described in the last paragraph would be a cornerstone of the bigger picture. This would represent one of the many possibilities in terms of output that the pipeline could be able to produce. As mentioned in the introduction, the results of my research might have an impact on a much larger portion of the scientific community than just the research group I integrated.

References

1. The galaxy Messier 101. [internet] ESA and NASA/Hubble [cited 2020 Nov 27] Available from: https://www.spacetelescope.org/news/heic0602/
2. J.L. Sérsic “Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy”. *Boletín de la Asociación Argentina de Astronomía*, Vol. 6, pp. 41-43. February 1963
3. Freeman, K.C. (June 1970). "On the Disks of Spiral and S0 Galaxies". *The Astrophysical Journal*. **160**: 811–830. Bibcode:1970ApJ...160..811F. doi:10.1086/150474.
4. Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks". *Communications of the ACM*. **60** (6): 84–90. doi:10.1145/3065386. ISSN 0001-0782.
5. Multi-layer neural network [internet] Wikipedia Commons [cited 27 Nov 2020] Available from: https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetwork_english.png
6. Stanford course CS231n, spring 2017. [internet] Youtube and Stanford [cited 2020 Nov 27] Available from: https://www.youtube.com/playlist?list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk
7. Variational Autoencoders: Intuition and implementation [internet] Agustinus Kristialdi’s blog [cited 2020 Nov 27] Available from: https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/
8. Bin Dai; David Wipf. (27 Sep 2018). “Diagnosing and Enhancing VAE models”. *ICLR 2019 Conference Blind Submission. Available from: https://arxiv.org/pdf/1903.05789v2.pdf*
9. Shengjia Zhao; Jiaming Song; Stefano Ermon (July 2019). “Balancing Learning and Inference in Variational Autoencoders”. *Proceedings of the AAAI Conference on Artificial Intelligence.* Doi:10.1609/aaai.v33i01.33015885
10. François Lanusse, Rachel Mandelbaum, Siamak Ravanbakhsh, et al. (9 Aug 2020) “Deep Generative Models for Galaxy Image Simulations” *arXiv preprint arXiv:2008.03833*
11. Herbel, J., Kacprzak, T., Amara, et al. (2018). “Fast point spread function modeling with Deep Learning”. *Journal of Cosmology and Astroparticle Physics*, *2018*(07), 054 doi:10.1088/1475-7516/2018/07/054
12. Snippet of the Point Spread Function [internet] user Ellande [cited 2020 Nov 27] Available from: https://commons.wikimedia.org/wiki/File:Point_Spread_Function.gif
13. KELLY, Patrick L., RODNEY, Steven A., TREU, Tommaso, *et al.* “Deja vu all over again: the reappearance of supernova Refsdal”. *The Astrophysical Journal Letters*, 2016, vol. 819, no 1, p. L8
14. The LSST large scale survey [internet] Rubin Observatory [cited 2020 Nov 27] Available from: https://www.lsst.org
15. The Euclid Survey [internet] European Space Agency [cited 2020 Nov 27] Available from: https://www.cosmos.esa.int/web/euclid/euclid-survey
16. The IllustrisTNG project [internet] The TNG collaboration [cited 2020 Nov 27] Available from: https://www.tng-project.org
17. TensorFlow [internet] Google Corporation [cited 2020 Nov 27] Available from: https://www.tensorflow.org
18. My github repository [internet] Available from: https://github.com/lgoupil/2020Internship
19. BALDASSI Carlo, BORGS Christian, CHAYES Jennifer T., *et al.* “Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes”. *Proceedings of the National Academy of Sciences*, 2016, vol. 113, no 48, p. E7655-E7662.
20. KINGMA, Diederik P. et BA, J. “Adam. A method for stochastic optimization”. arXiv 2014. *arXiv preprint arXiv:1412.6980*, 2019, vol. 434

21. GalaxyZoo contest dataset page [internet] Kaggle [cited 2020 Nov 27] Available from: https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/data
22. SPRINGENBERG, Jost Tobias, DOSOVITSKIY, Alexey, BROX, Thomas, *et al.* "Striving for simplicity: The all convolutional net". *arXiv preprint arXiv:1412.6806*, 2014.
23. Suhaib Tanveer M., Umar Karim Khan M., Chong-Min K. (june 2020) "Fine-tuning DARTS for image classification", available from: https://arxiv.org/pdf/2006.09042v1.pdf
24. "This AI system can produce images of artificial galaxies" [internet] Venturebeat [cited 2020 Nov 27] Available from : https://venturebeat.com/2018/11/08/this-ai-system-can-generate-images-of-artificial-galaxies/
25. KLUSHYN, Alexej, CHEN, Nutan, KURLE, Richard, *et al.* "Learning hierarchical priors in VAEs" In : *Advances in Neural Information Processing Systems*. 2019. p. 2870-2879.
26. ENGEL, Jesse, HOFFMAN, Matthew, et ROBERTS, Adam. "Latent constraints: Learning to generate conditionally from unconditional generative models". *arXiv preprint arXiv:1711.05772*, 2017.