Evaluation of GAN architectures for the characterization of transient noise in the Virgo interferometer

Master's degree thesis in physics of complex systems

Edoardo Grasso

Supervisor: Massimo Masera
Co-supervisor: Federica Legger, Sara Vallero

Università degli studi di Torino

Physics Department

A.A. 2022/2023

Abstract

Since the first detection of a *gravitational wave* (GW) in 2015, about 100 years after they were theorized by Einstein's General Relativity, GW research has become one of the most active fields of research in astrophysics. This new messenger allow us to explore a much larger portion of the Universe than, e.g., electro-magnetic radiation. A current-day GW detector is called *interferometer* and it requires a very fine level of precision. There are numerous disturbances that can affect the interferometer's sensitivity, some are continuous and add background noise localized at specific frequencies, others are transient.
When a transient noise event is sufficiently loud and has a time duration comparable to a typical GW signal, we call it *glitch*. Glitches can mask or mimic the passage of a real GW. Both are usually loud in a specific portion of the sensitive frequency spectrum of the interferometers. A visual representation of the time-frequency spectrum of a signal is called *spectrogram* and it is very useful for the analysis of GW data, where the final goal would be to discern glitches from real events of astrophysical origin.
Over the years, different ways to deal with glitches have been proposed, one of these is to exploit data gathered by the auxiliary channels of the interferometers. While the main channel (called *strain* channel) is susceptible to GWs and is responsible for detecting them, many auxiliary channels are not, since they gather data from other sensors. Nonetheless there are also auxiliary channels that are more closely related to the strain one.
In this thesis, I will explore how a particular type of machine learning based generative models, called *Generative Adversarial Networks* (GANs), can be used for glitch characterization through auxiliary channels. Among the tasks that GAN models can tackle there is *Image-to-Image translation* (I2I), that consists in a controlled conversion of a given source image to a target image. This feature is relevant for my work, because the spectrograms of glitches can be seen as images. I will apply a GAN trained for I2I where the source image is a glitch viewed in a specific auxiliary channel and the target image is the same glitch viewed in the strain channel.
The motivation behind this thesis is that a real GW signal as seen in the strain channel would be related only to the auxiliary channels that are themselves related to the strain one, since what we call "strain" is actually a composition of different auxiliary channels that measure, e.g., the differential length of the interferometer's arms. Other channels might witness specific types of glitches, but are not sensitive to the astrophysical signal. The network should learn the mappings that transfer glitches images from a subset of relevant auxiliary channels to the strain channel. When an event is detected all these transfers are attempted: if only the transfers from strain-related auxiliary channels succeed, the event is probably a GW signal, if also transfers from some non-strain related auxiliary channels succeed, the event is probably a glitch.
In this context, I want to investigate whether the idea of using GANs to transfer images from the auxiliary channel domain to the strain domain is valid. I will apply these models to a relatively small set of glitches that have been observed in the Virgo interferometer during observation run O3a, using data both from the strain channel and a subset of auxiliary channels.

Contents

Chapter 1

Introductory Concepts

In this chapter I will report introductory concepts about gravitational waves, including the basics of the detection mechanism, emission, and the analysis that permits spotting a signal in the interferometer's data. I will also talk about continuous noise sources that hamper GW detection by producing background noise that must be filtered out from the data before proceeding with any further analysis.

1.1 Gravitational waves

On September 14, 2015, the advanced Laser Interferometer Gravitational-Wave Observatory (LIGO) observed for the first time a signal of gravitational waves. The signal provides the first observational evidence of the existence of black holes with a mass larger than $25M_{\odot}$ [1]. The direct observation of gravitational waves was a further confirmation of Einstein's theory of general relativity. It also launched the use of GWs as a tool to explore the Universe, taking advantage of some unique features of this new messenger.

1.1.1 Generalities

According to general relativity, gravity is a local property of the space occupied by a test mass, which is curved by another source mass. Information on the distribution of mass is communicated throughout the Universe by the gravitational field. In Newtonian physics, any change in this distribution is propagated by a corresponding change in the gravitational field everywhere instantaneously. But Einstein's theory of relativity limits the speed at which anything, including information carried on fields, can travel to the speed of light. Therefore, a change in the position of a mass is communicated to the rest of the Universe through a propagating change in the gravitational field traveling in every direction at the speed of light [2], that is a gravitational wave (see figure 1.1).

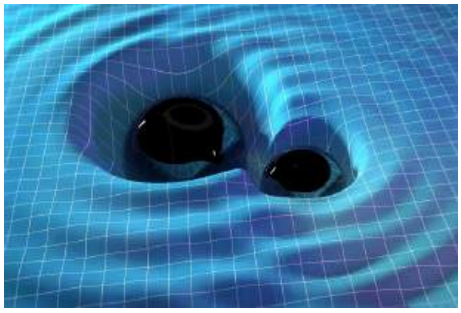


Figure 1.1: *Illustration of two orbiting black holes warping spacetime and generating gravitational waves.*

Gravitational waves manifest as ripples of curvature in the fabric of spacetime, transverse to the propagation direction and with two possible polarizations. The effect of a GW traveling along e.g., the z axis, is to squeeze and stretch spacetime along the x axis while stretching and squeezing it along the y axis (see figure 1.2).

Some important properties [3] of gravitational waves are:

- Detectable GW signals are generated by the motion of star-scale masses, in contrast to electromagnetic emissions like light and radio waves, which are emitted by the acceleration of subatomic-scale particles.
- GWs are very weakly interacting, and are effectively not scattered or blocked by intervening matter, unlike light and radio waves.
- GW detectors sense the amplitude of the signal, rather than its power, so that the detected signal falls linearly in strength as $1/r$ rather than as $1/r^2$ for many astronomical telescopes (where r is the distance of the detector from the source), allowing a cosmological reach.

1.1.2 Detection mechanism

The ultimate goal of a GW detector is to measure the amount of stretching and squeezing of spacetime induced by a GW, or *strain* h, as the difference ΔL in length L between two orthogonal directions. In the aforementioned example of the ring of test masses, let L be the diameter of the ring. The strain experienced by the ring along the x, y axis will be $h = \Delta L/L$ where $\Delta L = (L + \delta L_x) - (L + \delta L_y) = \delta L_x - \delta L_y$.
To measure this strain we use an experimental setup similar to a *Michelson and Morley interferometer*. Inside an interferometer, a beam is shot by a laser and encounters the beam splitter, an optical component that divides it in two. The two beams travel inside the interferometer's arms of length respectively L_x and L_y, at the end of both arms there is a mirror that reflects the beams. The two beams travel back and get rejoined by the beam splitter. The rejoined beam arrives at the output photodiode which senses the light intensity. Based on the difference between the travel length of the two light beams ΔL, they will have a phase difference, sensed by the photodiode as a change in light intensity [3] (see figure 1.3). Note that real detectors include several enhancements to achieve sufficient sensitivity to measure gravitational waves, as we will see in section 1.1.5.
Let λ be the wavelength of the laser light, we can express the expected change in phase ϕ, due to incident strain, h as:

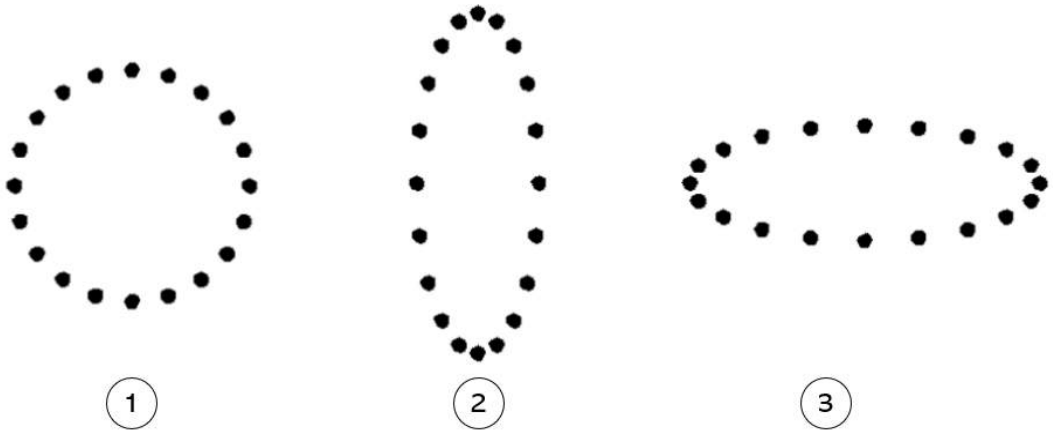
$$\Delta\phi = 4\pi\frac{Lh}{\lambda} \tag{1.1}$$

We can then relate the power measured at the output photodiode, P_{out}, to this change in phase measured by the detector, $\Delta\phi$, and power at the input of the interferometer, P_{in}:

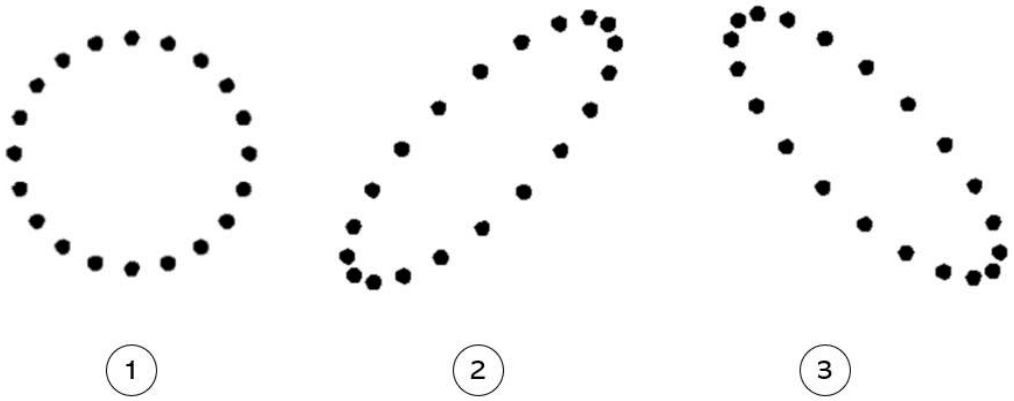
$$P_{out} = P_{in}\cos^2(\Delta\phi) = P_{in}\cos^2(4\pi\frac{Lh}{\lambda}) \tag{1.2}$$

The variation of output power ΔP_{out} due to a change in GW strain Δh is approximately linear, since $\Delta h \sim 10^{-21}$ (see figure 1.4). We can write:

$$\Delta P_{out} = \frac{2\pi L}{\lambda}P_{in}\Delta h \tag{1.3}$$



(a) Plus polarized (+) gravitational wave



(b) Cross polarized (×) gravitational wave

Figure 1.2: *Effect of a gravitational wave on a ring of test masses lying on the x, y plane according to the polarization of the wave. The GW is traveling along the z axis, perpendicular to the x, y plane, entering into the sheet.*

From this formula we see that making the detector arms longer and the laser power higher, will provide a bigger signal, allowing us, in theory, to detect a strain with a smaller magnitude. Unfortunately, making such adjustments will also enhance some particular frequencies of the background noise, so a balance must be found. Modern interferometers have a sensitivity that allows us to detect gravitational waves that produce a strain signal of the order of $h \sim 10^{-21}$.

1.1.3 Emission

In first approximation, we can understand the strain h observed by one of our detectors in terms of the gravitational constant, G, the speed of light, c, the distance between the detector and the source, r, and the source's quadrupole moment, I [3]:

$$h(t) \propto \frac{G}{c^4}\frac{1}{r}\ddot{I} \tag{1.4}$$

Note that I is a tensor, where each element is the summation over the mass of each system component multiplied by the distance from that component to a coordinate origin

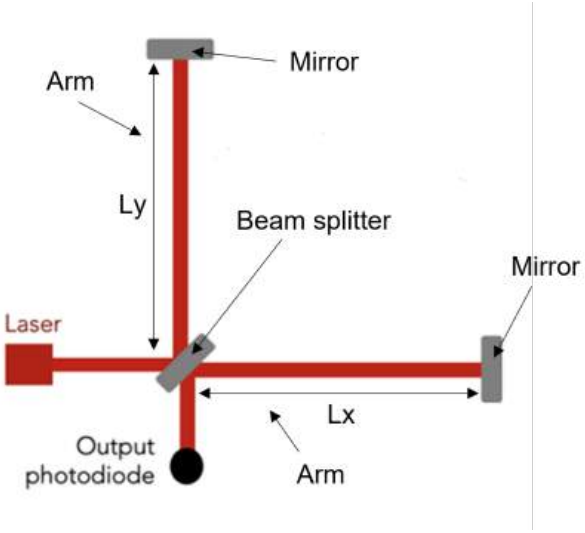


Figure 1.3: *Simplified diagram of an interferometer.*

in the directions described by that tensor element (i.e. I_{xx} would be the sum over mx^2 for each mass component). A system with perfect spherical symmetry would have zero quadrupole moment, and therefore would not produce any measurable GWs.
We see that the generated spacetime strain is proportional to the second time derivative of the system's quadrupole moment denoted $\ddot{I}$, which is the acceleration of some asymmetric mass distribution. Note that the constant G/c^4 is a very small number, on the order of 10^{-45} $s^2/(kg\ m)$. Moreover, as most GW sources are very far (millions or billions of light years) from the Earth, the $1/r$ term is quite small as well. The remaining term $\ddot{I}$ must be very large to generate even extremely small fractional changes in length, on the order of 10^{-21}. Detectable sources are not possible to generate on Earth, these sources are on the same mass scale as stars, and often moving at relativistic speeds.

To this day, detectable GW sources are essentially of two categories:

- Compact binary coalescence (CBC): binary systems composed of two compact objects (e.g. black holes, neutron stars) that are so close to each other to merge via GW emission within the age of the Universe [4]. CBCs are composed of three phases:

  1. *Inspiral*: the orbit of the two objects gradually shrinks;
  2. *Merger*: after the innermost complete orbit, the system transits to the merger phase, where the two black holes meet and there is the peak in GW emission;
  3. *Ringdown*: the now single black hole settles down to a stable spherical form.

  The first GW signal ever discovered came from a CBC (see figure 1.5).

- Burst type: the most representative are core-collapse supernovae (CCSN), which result from the rapid collapse and violent explosion of a massive star [5].

1.1.4 Basic data analysis

Since we don't expect weakly interacting gravitational wave signals to be altered by dust, galaxies, or any other mass between Earth and the source, we can use a technique known

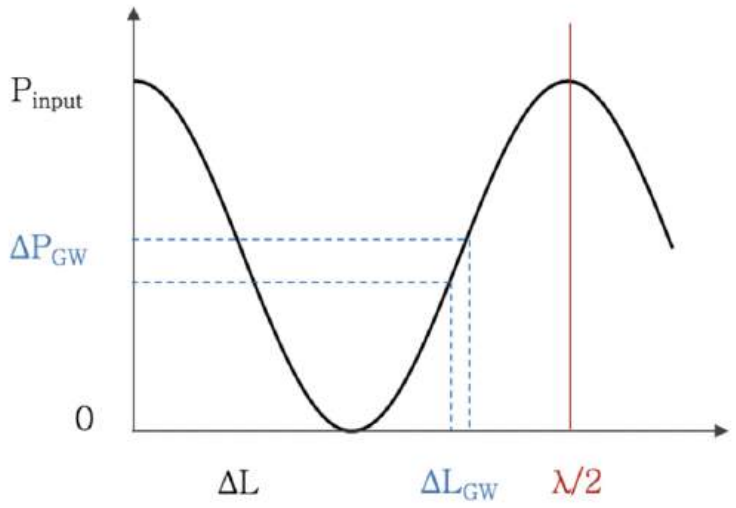


Figure 1.4: *A plot of how the laser power, that has value P_{input} when entering the interferometer, varies at the interferometer's output as the arm length difference changes (λ is the wavelength of the laser light). The blue dots show how a small differential change in arm length due to a passing gravitational wave, ΔL_{GW}, would induce a small (approximately linear) change in circulating power, ΔP_{GW}.*

as *matched filtering* to compare the detector data directly to well-modeled GW signals [3]. This technique consists in comparing a template for one of our expected signals (coming from a mathematical model of a specific GW source) to our detector data, we can calculate the cross-correlation between them as the signal-to-noise ratio (SNR). The SNR (ρ) is the sliding inner product between the template, h, and the data, s, normalized by the frequency content of the noise, S_n integrated over frequency f.

$$\rho =< s|h >= 4Re \int_0^{inf} \frac{s(f)h^*(f)}{S_n(f)} e^{2\pi i f t} df \tag{1.5}$$

If there is sufficient GW signal power that is well described by our template present in the data, our cross-correlation will yield a high enough ρ that is unlikely due to a Gaussian noise fluctuation. Usually, if ρ exceeds roughly 8, the signal will be flagged as a candidate GW event [3]. Moreover, if we have a network of interferometers located in different places on the Earth, we can use the powerful noise rejection constraint of multiple detectors to look only for coincident triggers due to true signals which propagate through the detector network.
Among the different GW sources, there are some for which we don't have a mathematical model as shown in figure 1.6. In these cases we can't use matched filtering, we use instead coherency methods: instead of assuming a template model for the waveform, we search for coherent gravitational wave power across our detector network.

1.1.5 The Virgo interferometer

The Virgo interferometer is hosted in Italy, near Pisa, in the town of Cascina, and it is one of the most prominent gravitational wave interferometers globally. It works alongside KAGRA, situated in Japan, and LIGO, which comprehends two detectors located in the USA: one in Livingston, Louisiana, and the other in Hanford, Washington. The Virgo

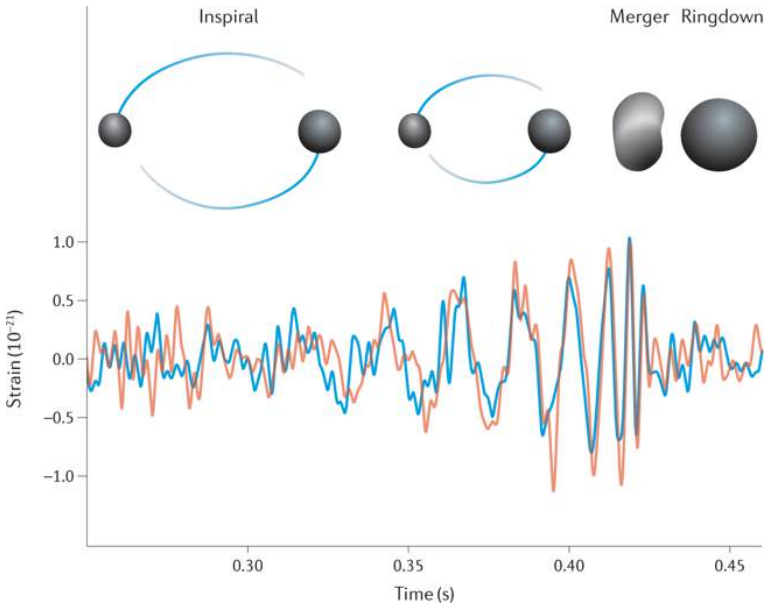


Figure 1.5: *The first GW signal ever revealed by the Advanced LIGO interferometer, which is composed of two detectors: LIGO Hanford, and LIGO Livingston. The orange and blue curves show the matching of the signal registered by the two detectors. The signal came from a compact binary coalescence (CBC) of two black holes. note that in the figure the three phases of the CBC are highlighted: in the inspiral phase, the orbit of the two black holes gradually shrinks until they collide in the merger phase, where we have the peak in GW emission, followed by the last phase of ringdown, where the now single black hole settles down to a stable spherical form.*

Collaboration, together with the LIGO Scientific Collaboration and the KAGRA Collaboration form the LIGO-Virgo-KAGRA (LVK) Collaboration: a network of scientists who work together on GW detection.
Virgo is an instrument designed to detect gravitational wave signals and, as so, it is basically a Michelson interferometer with two 3 Km long longitudinal arms, but features several enhancements beyond the basic Michelson interferometer design. Some of these are [10]:

- *Pre-Stabilized Laser (PSL)*: Virgo uses a high-power laser source called the Pre-Stabilized Laser. The PSL generates a highly stable laser beam with precise wavelength and frequency control.

- *Input Mode Cleaner (IMC)*: before entering the arms of the interferometer, the laser passes through the input mode cleaner, a system that refines the quality of the laser beam to ensure that only the desired transverse modes are used.

- *Fabry-Perot Cavities*: each arm is composed of two mirrors, the input mirror, and the end mirror. The end mirror is completely reflective, it bounces the laser beam back towards the beam splitter; the input mirror is highly reflective but also partly transmissive. Adding the input mirror between the beam splitter and the end mirror makes the laser light bounce back and forth multiple times. The effect of a gravitational wave on the phase difference between the laser beams in the two arms will be additive for each round trip, the result being as if the arms were much longer than they actually are. Note that Fabry-Perot Cavities are used also in other places of the interferometers and not only inside the arms, for instance in the Power

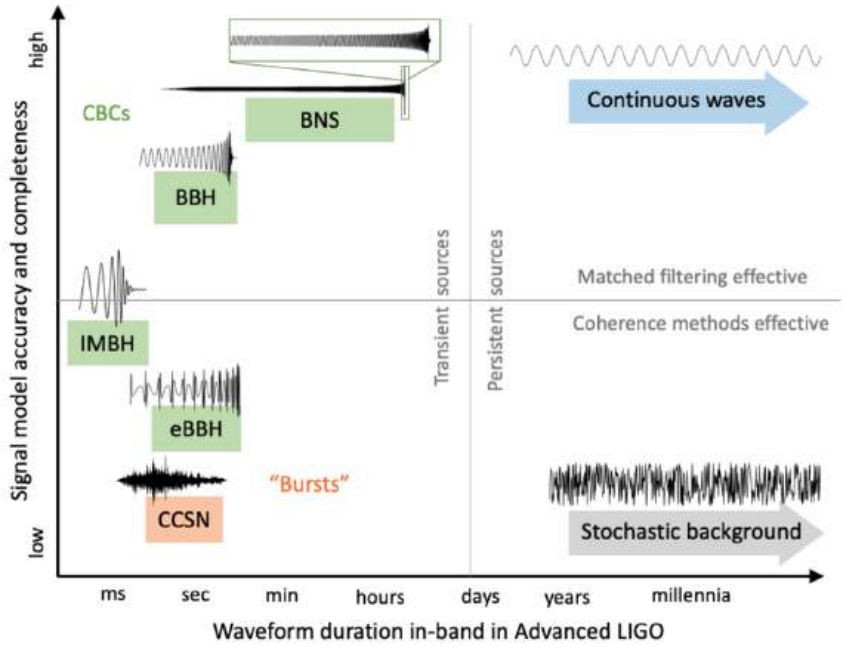


Figure 1.6: *examples of target GW sources shown by how well current waveform models capture the signal and how much time the signal would spend in the detectable band of the detectors, or "in-band". Note that BNS stands for "binary neutron star", BBH for "binary black hole", IMBH for "intermediate-mass black hole", and eBBH for "eccentric binary black hole", these are all types of CBCs where what varies is the kind of compact objects that compose the binary system.*

Recycling Cavity (PRC), placed between the IMC and the beam splitter, which has the purpose of augmenting the laser power circulating inside the interferometer.

- *Output Mode Cleaner (OMC)*: it is placed between the beam splitter and the output photodiode, it has a similar functioning to the IMC but its purpose is to avoid spurious modes that might have been generated by inaccuracies in the mirrors' alignment to reach the output photodiode.

Auxiliary channels of the Virgo interferometer

The Virgo detector records data streams from a large number of subsystems controlling different aspects of the instrument and monitoring their state. Some auxiliary channels are directly correlated with the main strain channel but others include data from a variety of instrumental and environmental sensors, such as photodetectors and seismometers. These sensors can witness noise sources that couple to the interferometer. Their data can be used to diagnose and mitigate non-astrophysical couplings [15]. An example is shown in figure 1.7.
I will talk more extensively about auxiliary channels in chapter 3, as we will see, the final goal of my thesis is to train a machine learning model to understand the mapping from various auxiliary channels to the strain channel.

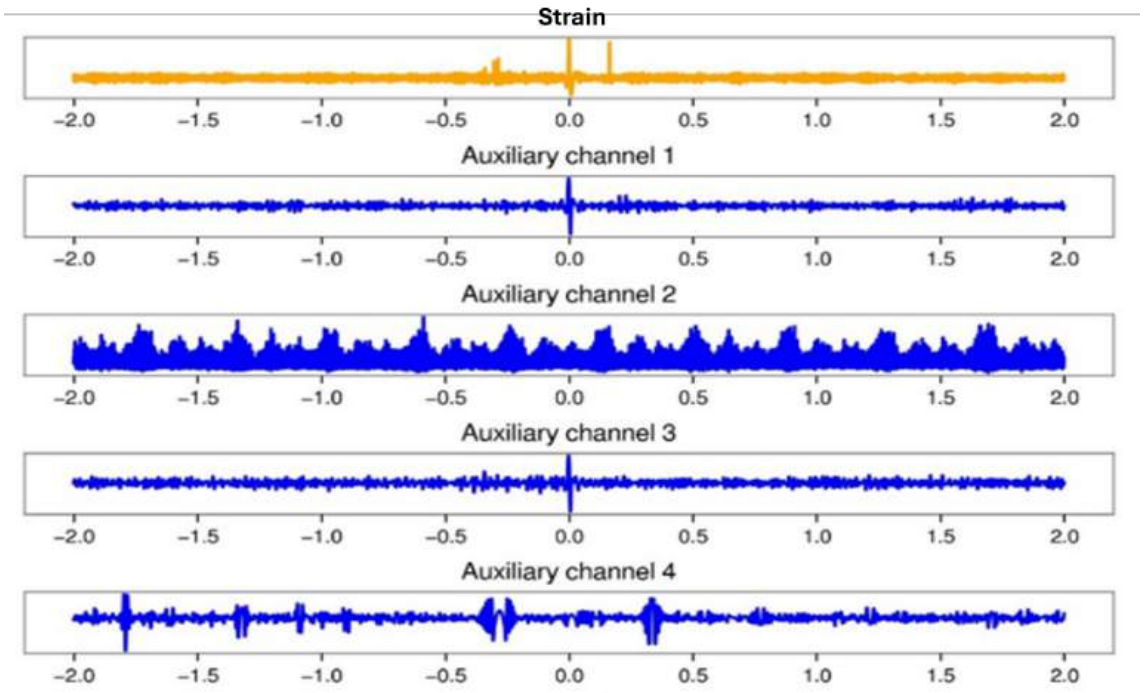


Figure 1.7: *A qualitative illustration of how auxiliary channels can help determine the non-astrophysical nature of detector triggers. The top time series is $h(t)$. The other time series are from detector auxiliary channels that monitor different sources that are not sensitive to GWs. The spike in the strain time series at $t = 0$ occurs also in auxiliary channels 1 and 3. This may indicate that the trigger is not of astrophysical origin [15].*

1.2 Noise sources

Since the interferometers have to detect an incredibly small quantity like the spacetime strain produced by a gravitational wave, any kind of noise source can be a big problem and must be dealt with. In this section, I will talk first about what is referred to as continuous *stationary* noise, i.e., noise that is continuous in time and for which the frequency content does not change over time, so it can be easily subtracted from the detector data with a procedure called "whitening", described later in chapter 3.
When searching for a gravitational wave signal we look for non-stationary behavior in strain data, this kind of behavior will not be filtered out by the whitening procedure, but unfortunately, GW data can exhibit also some types of non-stationary noise that is, therefore, much more difficult to identify and remove. Section 1.2.2 will introduce *glitches*, which are the most relevant type of non-stationary noise that can hamper gravitational wave detection, for which identification is very challenging and can take advantage of machine learning methods.

1.2.1 Stationary noise

I will start with a brief description of the most important stationary noise sources [3].

1) Photon shot noise

This type of noise has to do with a fundamental property of light that limits our ability to determine the light intensity exactly, and as a consequence adds uncertainty to the measurement of the difference in the lengths of the arms, ΔL, exactly. Light can be described as a stream of photons, and they arrive at the output photodiode randomly in time. The rate of arrival fluctuates and the fluctuation is described by the Poisson statistics. E.g., if our laser would emit only 10 photons per second on average, our fluctuation would be of $\sqrt{10} \sim 3$, and the fractional fluctuation would be $\sqrt{10}/10 = 1/\sqrt{10} \sim 30\%$. Our uncertainty in the actual position of the masses due to this effect scales as $1/\sqrt{P}$ so it *decreases* with increasing laser power.

2) Radiation pressure force noise

Each photon hitting a mirror in the interferometer transfers a bit of momentum to that mirror and this effect is known as radiation pressure, it can cause the mirrors to jitter in position and potentially mask the very subtle motions due to gravitational waves. The uncertainty due to this effect *increases* with increasing laser power P, since higher P means higher momentum transferred to the mirror, and this leads to making mirrors as massive as possible to help reduce the motion due to this momentum.

The two effects just described are usually referred to as "quantum sensing noise" or "*quantum vacuum noise*". The photon shot noise is more present at high frequencies and it decreases with higher laser power, radiation pressure force noise is problematic at low frequencies and decreases with lower laser power, this helps us establish the optimum laser power to use for a measurement, based on what is the desired observation frequency.

3) Thermal noise

Thermal noise is observed as small-scale random motion in all objects due to their physical temperature. Even in our 40 Kg optics, we can observe this effect, where thermal energy can induce real physical motion of the mirrors. In a resonant system, thermal motion decreases at high frequencies and peaks at frequencies close to resonance. To reduce this effect we seek materials and construction methods that minimize the mechanical losses, this, despite not reducing the integrated motion of the object, concentrates it at the frequency of resonance making it negligent above and below the resonance. Note that we usually classify thermal noise in different categories based on the component of the interferometer that is affected by it or the effect it has on a certain component:

- Suspension thermal noise: due to motion of the pendulum suspension of the mirror;
- Substrate Brownian noise: due to excitation of the mirrors themselves;
- Coating Brownian noise: due to the motion of the very thin reflective coating on the mirrors.
- Substrate thermo-elastic noise: due to thermo-elastic dissipation of vibration energy possessed by the mirrors;
- Coating thermo-optic: due to mechanical loss in the coatings of the mirrors.

4) Ground motion noise

Ground motion can be due to human activity, wind, water waves, and Earth's seismic activity. It is generally large at low frequencies and falls as $\sim 1/f^2$. An important effect of seismic motion is due to what is referred to as Newtonian noise. Basically, the mirror, suspended as a pendulum, is attracted to the distribution of mass around it but the distribution dynamically changes due to, e.g., seismic waves. This random force on the mirrors dominates at around 7 Hz and lower frequencies, hiding any gravitational wave signal in this portion of the spectrum. However, exploiting very efficient filtering techniques, seismic noise becomes negligent above roughly 10 Hz.

Figure 1.8 shows graphically the noise sources I just described, pointing out their relevance at different observation frequencies.

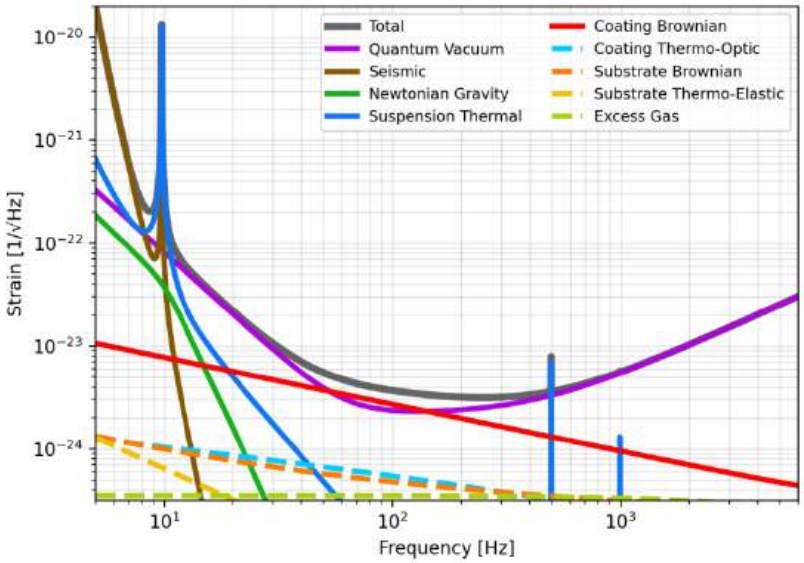


Figure 1.8: *A schematic representation of what we call a sensitivity curve, that is the amplitude spectral density, or ASD (see section 3.2.1), of the noise content in a generic interferometer. As I discussed above, at low frequencies, the most relevant noise source is seismic noise, while at high frequencies, quantum vacuum noise and, in particular, photon shot noise, is the most problematic. We can also appreciate vertical lines, known as "spectral lines", placed at resonant frequencies of the system resulting from thermal noise. Note that the "excess gas" noise, which I did not mention, is essentially due to the residual molecules of gas in the arms of the interferometer.*

Figure 1.9 shows the real sensitivity curve of the Virgo detector acquired at a specified time. One could notice the presence of additional vertical lines (or spectral lines) at frequencies of 50 Hz and harmonics, compared to the schematic sensitivity curve of figure 1.8. These are due to electromagnetic couplings with the AC power grid in Europe: the sensitivity curve for LIGO Hanford or LIGO Livingston (two interferometers that are located in the USA), has similar lines at 60 Hz and harmonics.

1.2.2 Glitches

The term "glitch" refers to transient, short-duration non-stationary noise or disturbance in the interferometer data that can mimic or obscure the detection of actual gravitational

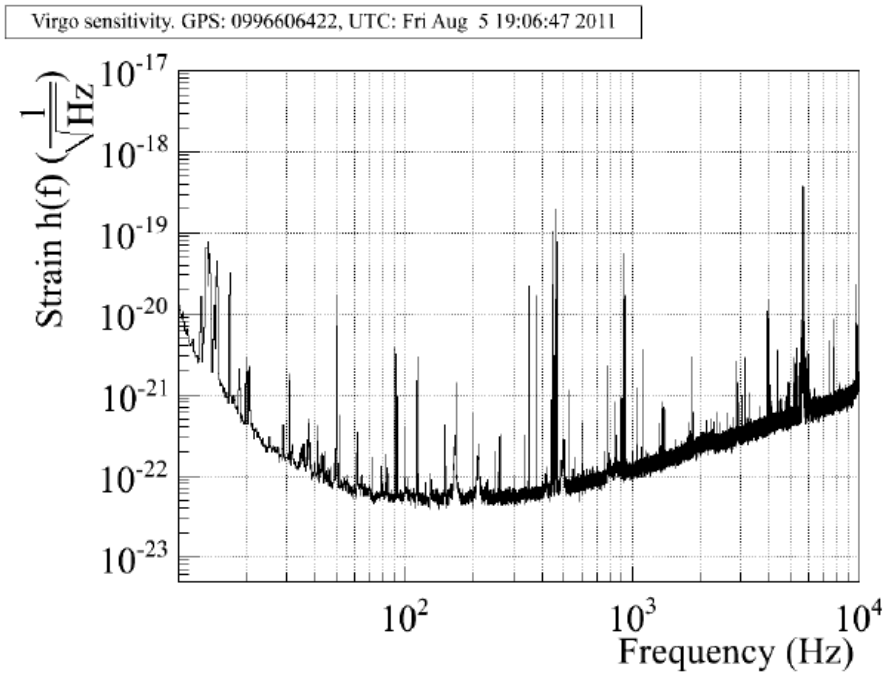


Figure 1.9: *Sensitivity curve of the VIRGO detector, acquired at the time specified above the plot.*

wave signals. Glitches are unwanted artifacts that can complicate the analysis and interpretation of data.
Glitches can originate from a variety of sources of different nature [42], for example:

- *Seismic*: Ground movements such as earthquakes or local vibrations can introduce transient seismic noise into the interferometer data;
- *Instrumental*: Various mechanical and electronic components of the detectors can produce transient noise, such as fluctuations in laser power or electrical interference;
- *Environmental*: External factors like wind, temperature variations, or even wildlife can cause disturbances and produce transient noise;
- *Calibration errors*: Also inaccurate calibration of the detectors can lead to glitches in the data.

A useful way to visualize a glitch is given by its spectrogram, i.e., a visual representation of the spectrum of frequencies of the signal as it varies with time (more details about the algorithm used to create spectrograms, called *Q transform*, in chapter 3). The shape of a glitch, that is portrayed in its spectrogram, is an indication of the evolution over time of the frequency spectrum of the glitch and so it is related to the physical origin of that particular glitch. In fact, both glitches and real gravitational wave signals can have disparate physical origins and so can produce a large variety of different shapes: a comprehensive classification and characterization of glitches could allow their origin to be identified and their root cause to be removed from the instruments. That is why there are active projects like *Gravity Spy* [17] that aim to provide a classification of glitches based on their shape. Figure 1.10 shows one example for each glitch class that has been identified by the Gravity Spy project, the name of the class sometimes refers to what

glitches in that class look like, e.g., "Paired Doves", and sometimes to what is their suspected physical origin, e.g., "Scattered Light". Note that the glitches present in the Gravity Spy dataset have been observed in the LIGO interferometer data, but the same types of glitches also occur in Virgo.
The glitch classes on which I focused in my thesis are:

- *Scattered Light* glitches consist of low-frequency, relatively long-duration, humpy glitches that can have several lines stacked on top of each other. This glitch type is caused by the laser light scattering off of dust or other particles which can produce a bright, narrow peak in the detector output that can be mistaken for a gravitational wave signal.

- *Blip* are short glitches with usually a symmetric "teardrop" shape. The origin is unknown.

- *Koi Fish* are similar to blip glitches but resemble a fish with a head, fins, and thin tail. The cause of koi fish glitches is not yet fully understood, but they may be related to the mechanical properties of the mirrors.

- *Low-Frequency Lines* occur at very low frequencies and do not vary in frequency over long durations. This glitch type is caused by low-frequency vibrations in the detectors, which can produce a broad, low-frequency noise pattern in the detector output. Low-frequency noise can be caused by a variety of factors, including seismic activity, wind, and temperature changes.

1.2.3 Machine learning for glitch analysis

Machine learning (ML) is a subset of artificial intelligence that focuses on the development of algorithms that enable computers to learn and make predictions based on data, without being explicitly programmed. It is used in a wide range of applications, including image and speech recognition, language processing, autonomous vehicles, medical diagnostics, and much more. A subfield of machine learning is deep learning, which combines the power of ML algorithms with *neural networks*, which are architectures composed of interconnected layers of artificial neurons. The artificial neuron, being schematized in figure 1.11, is the basic computational unit: it receives as input multiple values and each input is associated with a weight; the neuron computes a weighted sum of the inputs and weights which is then passed through a function commonly known as "activation function" (e.g., it can be a sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$); the output of the neuron is the result of the activation function applied to the weighted sum. Many neurons placed one on top of the other form a layer, many layers stacked one after the other and interconnected with each other form a neural network, as shown in figure 1.12. A neural network for which every node in a certain layer is connected to every node in the next one is called "*feedforward neural network*".
More details about the functioning of neural networks can be found in chapter 2, for now I will say that they are widely employed in glitches analysis, being extremely useful for their ability to learn complex patterns and representations from data. Moreover, there exist numerous types of neural networks that are modifications of the standard architecture shown in figure 1.12, e.g., *convolutional neural networks* (CNNs) [22] which are designed to extract information from images and are used, for instance, for automatic

glitch classification [16]. Also, in chapter 2 I will talk extensively about *generative adversarial networks* (GANs), a class of neural networks used to generate synthetic data, which have already been employed in glitches analysis, for instance, to create a synthetic population of blip glitches [41].

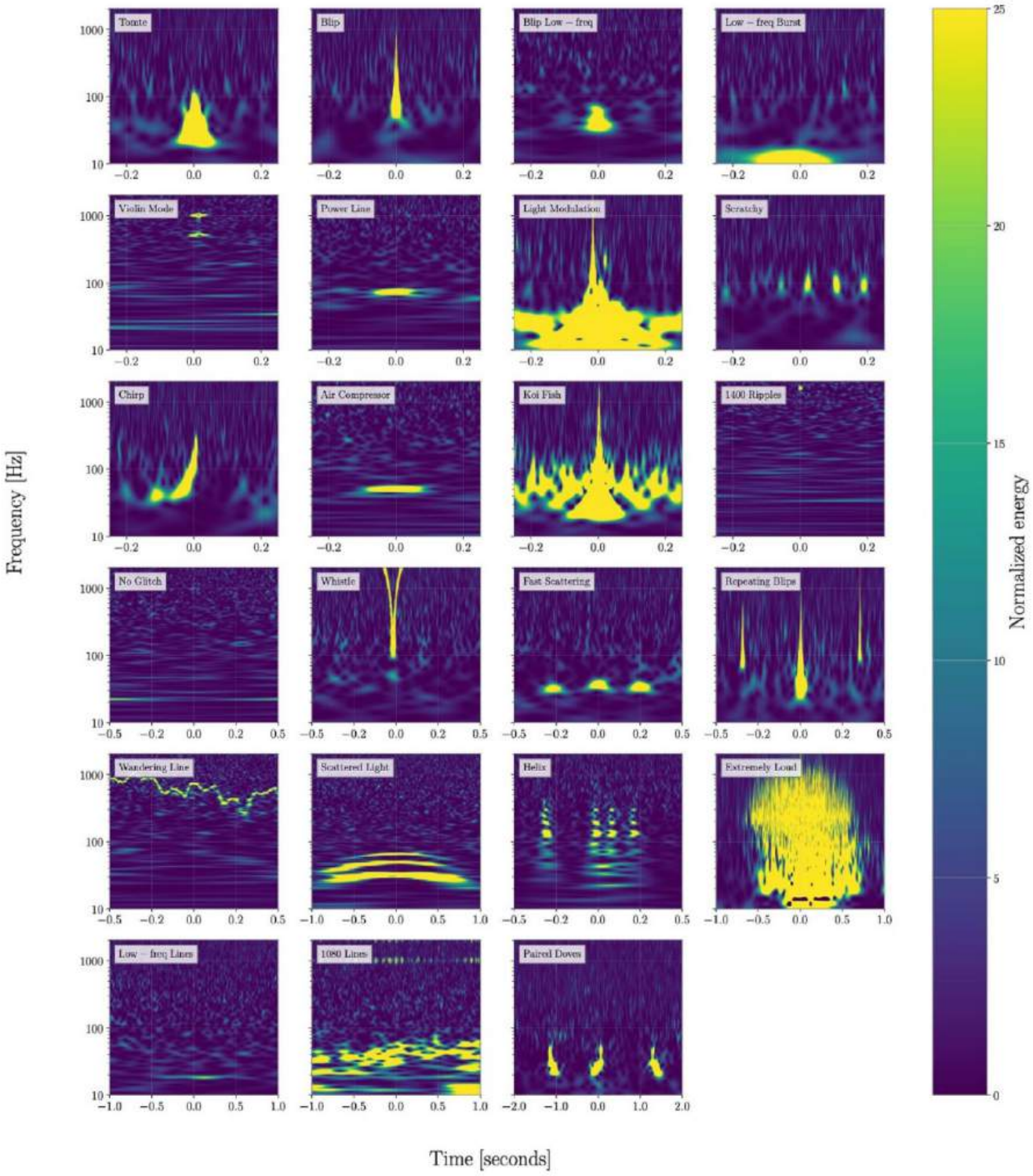


Figure 1.10: *An example for each glitch class. These glitches have been detected by the LIGO interferometer. Each figure is a time vs. frequency plot where the color map indicates the normalized energy, i.e., the weight of a specific frequency in the signal at a specific time [18].*

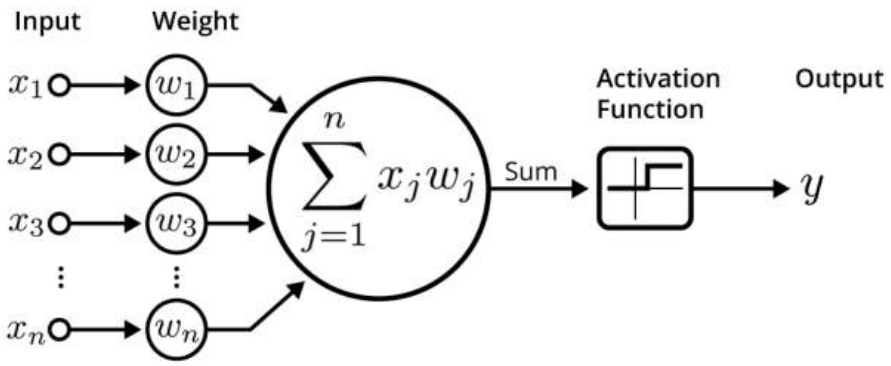


Figure 1.11: *The artificial neuron: each input x_i is multiplied by a specific weight w_i and all these products are summed together; the result of this summation is fed to a function called in jargon "activation function"; the result is the output of the neuron*

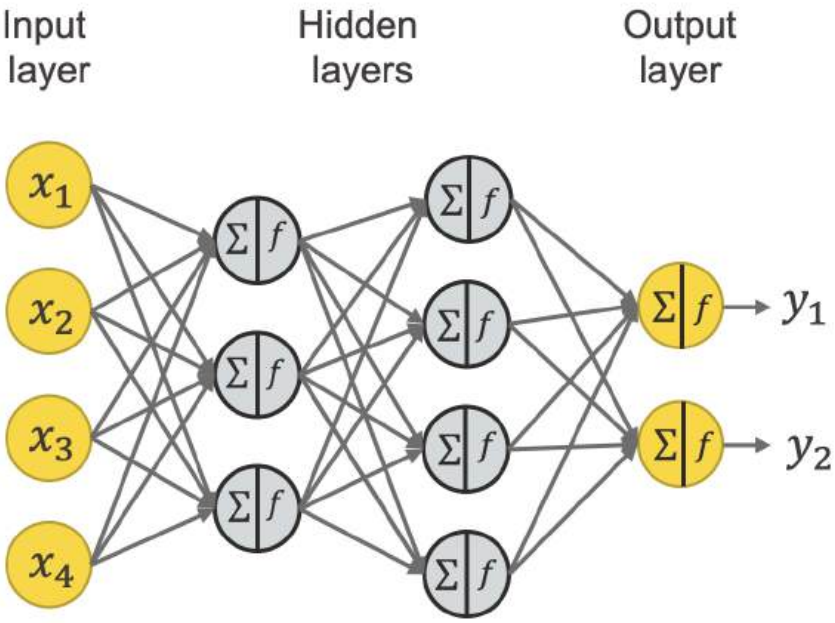


Figure 1.12: *Artificial neural network: each neuron is represented by the summation symbol and the activation function; each column of neurons is called layer; all the layers apart from the input and output ones are called hidden layers; in this example, every neuron of a certain layer is connected to every neuron in the following layer, this type of architecture is called "feedforward neural network".*

Chapter 2

Generative Adversarial Networks

Generative Adversarial Networks (GANs) belong to the class of generative models, they provide a way to learn useful representations of data that may be used in a variety of applications. The most straightforward application of GANs is generating synthetic samples that resemble a given set of already existing ones (e.g., generating portraits of human faces that do not exist given a set of real portraits of existing people), this is what vanilla GANs were built for. Many variations of the original GAN architecture have been developed in the years, and some of these can address different tasks like *image-to-image translation*: the process of transforming an image from one domain to another (e.g., transforming a landscape photo to an image of the same landscape as if it were painted by Van Gogh). In this chapter I will first give some theoretical background, then describe some important types of GANs, focusing more on models that perform Image-to-Image translation, in particular, the models *Pix2Pix* and *CycleGAN*, which I implemented myself and used for my analysis.

2.1 Introduction on generative modeling

The goal of generative models is to study a collection of training samples and learn the probability distribution that generated them. Technically speaking, training samples x are drawn from an unknown distribution p_{data}. A generative model wants to learn a p_{model} that approximates p_{data} as closely as possible, so that new samples z drawn from p_{model} will resemble the original training samples x.
A straightforward way to learn an approximation of $p_{data}(x)$ is to explicitly write a function $p_{model}(x;\theta)$ controlled by parameters θ and search for the values of the parameters that make p_{data} and p_{model} as similar as possible. This kind of approach is known as "explicit density modeling", it assumes a particular distribution for the data and utilizes true data to fit the distribution parameters [23]. In particular, one popular approach to generative modeling is the *maximum likelihood estimation*, consisting in maximizing the Kullback-Leibler divergence[1], which estimates the distance between two distributions [27], between p_{data} and p_{model}:

$$\max_{\theta} D_{KL}\Big(p_{data}(x) \parallel p_{model}(x;\theta)\Big) \tag{2.1}$$

This approach becomes unfeasible when the underlying distribution that generated our training samples is so complicated that it becomes computationally intractable.

[1] $D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$, where $P(x)$ is the probability of an event x occurring according to the distribution P; $Q(x)$ is the probability of the same event x occurring according to the distribution Q; and $\mathcal{X}$ is the set of all possible events.

Generative Adversarial Networks belong to the category of "implicit density models" because they can extract data from the learned p_{model} distribution without explicitly representing it.

2.2 Theoretical background

A GAN achieves the goal of inferring the probability distribution of the training examples through an adversarial process in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data. Figure 2.1 shows a schematic representation of the functioning of a GAN. Essentially the training procedure for G is to *maximize* the probability of D making a mistake while for D it is to *minimize* that same probability. This framework has been inspired by what in game theory is known as a two-player minimax game [25].

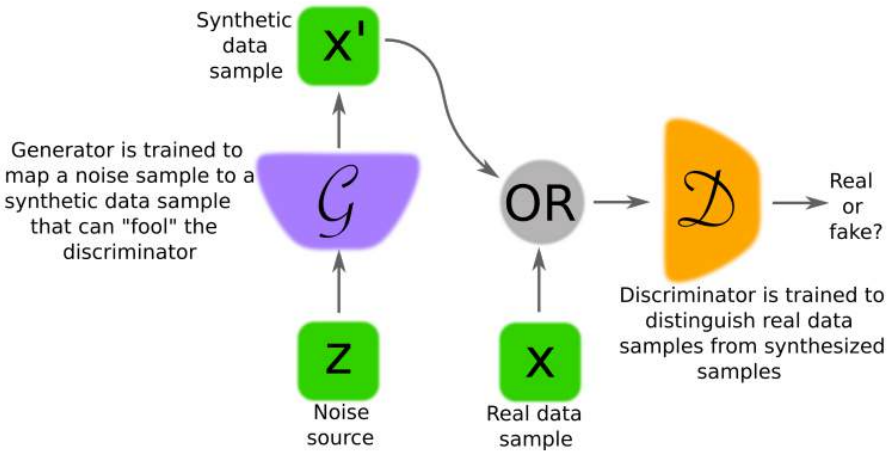


Figure 2.1: *Schematic representation of how a GAN works [24].*

2.2.1 Minimax game

Let us call the generated data distribution p_g and the real (unknown) data distribution p_{data}. As it can be seen in figure 2.1, we must also define a distribution from which we sample random noise to feed to the generator, we call it p_{noise}.
We introduce the generator model as $G(z;\theta_g)$, G is a differentiable function with parameters θ_g, which takes as input z, a noise vector sampled from p_{noise}. The discriminator will be $D(x;\theta_d)$ and it will output a single scalar that represents the predicted probability that x came from p_{data} rather than p_g.
Now take a dataset of n samples, this dataset will be used to evaluate the performance of the discriminator and contains both real samples (extracted from p_{data}) and fake (or generated) samples (extracted from p_g). Also, this dataset is labeled, meaning that we know which samples are real and which are fake. What we need is a function that measures the difference between the values predicted by the discriminator and the actual target values for samples inside our dataset, such a function is called *loss function*. Let y_i be the true

label of the i-th sample, it is 1 if it is a real sample or 0 if it is fake (generated); let $\hat{y}_i$ be the prediction of the discriminator for the label of the i-th sample, it can be any number between 0 and 1. An appropriate loss function for the discriminator would be the binary cross-entropy (BCE) [28]:

$$BCELoss = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \log(\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\Big] \tag{2.2}$$

We can write the equation for the discriminator loss noticing that for each sample:

- If $y_i = 1$: we have only the first part of the equation and the discriminator is evaluating a real sample, that is $D(x)$ with x extracted from p_{data};
- If $y_i = 0$: we have only the second part of the equation and the discriminator is evaluating a fake sample, that is $D\big(G(z)\big)$ with z extracted from p_{noise}.

Then we can write[2]:

$$L_D = -\mathbb{E}_{x\sim p_{data}(x)}\Big[\log\big(D(x)\big)\Big] - \mathbb{E}_{z\sim p_{noise}(z)}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big] \tag{2.3}$$

The discriminator wants to minimize this function by adjusting its parameters θ_d.
The generator, instead, wants all fake samples to be labeled as true ones. So it wants to maximize (by adjusting its parameters θ_g) the second part of the discriminator loss equation, or equivalently, to minimize the loss function:

$$L_G = \mathbb{E}_{z\sim p_{noise}(z)}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big] \tag{2.4}$$

These two losses can be summarized in one single "objective function":

$$O(G,D) = \mathbb{E}_{x\sim p_{data}(x)}\Big[\log\big(D(x)\big)\Big] + \mathbb{E}_{z\sim p_{noise}(z)}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big] \tag{2.5}$$

the generator and the discriminator play a minimax game where one aims to minimize it and the other to maximize it, we can write:

$$G^* = \arg\min_G \max_D \Big\{\mathbb{E}_{x\sim p_{data}(x)}\Big[\log\big(D(x)\big)\Big] + \mathbb{E}_{z\sim p_{noise}(z)}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big]\Big\} \tag{2.6}$$

Where G^* is the optimal generator.

2.2.2 GAN training

Training the GAN means making a series of steps where in each step the parameters θ_d get updated to maximize the objective function (equation 2.5), and the parameters θ_g get updated to minimize it. In order to train the GAN we need to have a *training set*, that contains all our available samples extracted from the real data distribution p_{data}. Let us say that there are N training samples in our training set.

[2]Note that: $\mathbb{E}_{x\sim p(x)}f(x) \equiv \int_x f(x)p(x)dx$ with x integrated over the support of $p \rightarrow$ is the mean value of $f(x)$ over distribution $p(x)$

Minibatch stochastic gradient descent

A commonly used algorithm to perform the updates of the parameters is called *minibatch stochastic gradient descent/ascent*. "Standard" gradient descent is a technique that consists of estimating the gradient of the loss function with respect to the parameters that we want to optimize using N points (every point in the training set) and then updating the parameters in the opposite of the direction of the estimated gradient. Applying gradient descent to update the generator would mean doing:

$$\theta_g = \theta_g - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_g} \log(1 - D(G(z^{(i)}))) \tag{2.7}$$

Where, as usual, $z^{(i)}$ are sampled from p_{noise}. η is called the "learning rate" and scales the largeness of the steps that we take each time we update the parameters, a commonly used value is $\eta = 0.001$. When updating the discriminator, we must remember that we want to maximize an objective function and not minimize a loss function: we apply gradient *ascent* instead, which just means that we update the parameters in the same direction of the estimated gradient and not the opposite one:

$$\theta_d = \theta_d + \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_d} \Big[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \Big] \tag{2.8}$$

A *minibatch* refers to a small, randomly selected subset of the training data: minibatch stochastic gradient descent is a variation of gradient descent where, instead of estimating the gradient using the whole training data, we estimate it on a minibatch. The advantage is that the updates are much quicker and an element of stochasticity is added, which improves training.

Learning algorithm

The best theoretical way to train both networks, as described in [25], would be to optimize D to completion every time we do an update of G. This is computationally prohibitive, instead, we alternate between k steps of optimizing D and one step of optimizing G (in standard implementations $k = 5$), this way we can maintain D near its optimal solution, so long as G changes slowly enough.
A thing to point out is that early in learning when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $D(G(z))$ is close to 0 and so $\log(1 - D(G(z)))$ saturates, slowing down the training process. Rather then training G to minimize $\log(1-D(G(z)))$ we can train G to maximize $\log D(G(z))$. This objective function results in the same fixed point of the dynamics of G and D but prevents saturation.
Figure 2.2 shows a non-formal explanation of the learning algorithm.
Pseudo code for a minibatch stochastic gradient descent training of GANs, as can be found in [25], is presented in algorithm 1.

Backpropagation

Note that a common architecture for the generator and discriminator model is a feed-forward artificial neural network (that I introduced in section 1.2.2). In this case, the

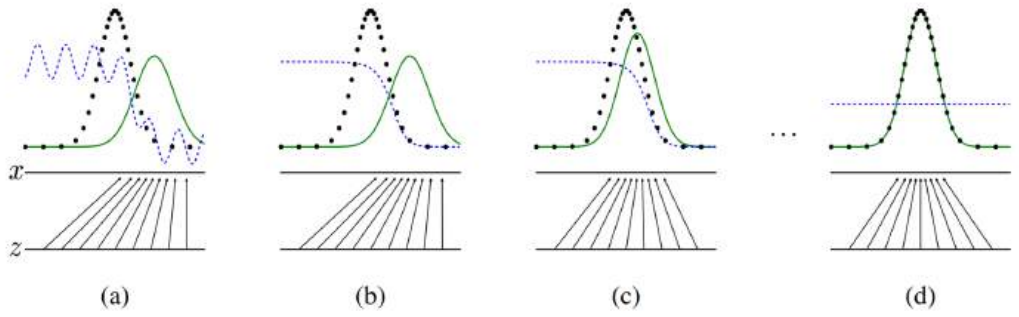


Figure 2.2: *Schematic representation of the learning algorithm of a GAN. The thick dotted line is p_{data} and the thin green line is p_g. The blue dotted line represents the discriminator, in the last picture it is flat indicating that it can no longer discriminate real from fake samples. The figure also shows z, i.e., the domain from which random noise is sampled (in this case uniformly) and the mapping $x = G(z)$ [25].*

parameters (θ_g and θ_d) that we must update at each step, also called "learnable parameters", are the weights associated with every link of the network. Calculating the gradient of the loss function with respect to all the weights of the network is hard and involves an algorithm called *backpropagation*.
The backpropagation algorithm can be divided into two phases: the forward pass and the backward pass.

In the **forward pass**, the input data is fed through the neural network to make predictions. The following steps are involved:

1. Input data, represented as a vector x, is passed to the first layer of the neural network. Each neuron in this layer performs a weighted sum of its inputs:

$$z_i^{(1)} = \sum_j w_{ij}^{(1)} x_j$$

   where $z_i^{(1)}$ is the weighted sum for neuron i in the first hidden layer, $w_{ij}^{(1)}$ is the weight connecting neuron i to input j.

2. The weighted sum is then passed through an activation function σ:

$$a_i^{(1)} = \sigma(z_i^{(1)})$$

3. The same process is repeated for every other layer:

$$z_i^{(l)} = \sum_j w_{ij}^{(l)} a_j^{(l-1)}$$

$$a_i^{(l)} = \sigma(z_i^{(l)})$$

4. The final layer's output is used to make predictions. For the discriminator, the last layer is composed of a single neuron and its activation function is usually a sigmoid, which produces an output in the range $[0, 1]$, representing the predicted probability of the sample being real or synthetic. For the generator, the output has the same

Algorithm 1 Minibatch stochastic gradient descent training of GANs [25].

for number of training iterations **do**

for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \Big] \tag{2.9}$$

end for

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \tag{2.10}$$

end for

number of neurons as the dimensionality of samples extracted from p_{data}, and the activation functions of these neurons could also be sigmoids, if, e.g., the generator is trying to generate pictures for which every pixel has an activation value in the range $[0, 1]$.

The **backward pass**, often referred to as error backpropagation, is where the network updates its parameters to minimize the prediction error. This involves the following steps:

1. Using the output predicted in the forward pass, calculate the value of the loss function (or objective function), here indicated with J.

2. Compute the gradient of J with respect to the model's parameters (weights). This is done using the chain rule of calculus. For weights in layer l:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

where $\delta_i^{(l)}$ is what we call the "error signal" for neuron i in layer l:

$$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} = \sum_k \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

Using this algorithm we can obtain the gradient of the loss with respect to each parameter. We will update the parameters using minibatch stochastic gradient descent as we saw above.

2.3 Basic types of GANs

2.3.1 Vanilla GAN and conditional GAN

The GAN that I described so far is usually referred to as "vanilla GAN", introduced in the original paper by Goodfellow et al. published in June 2014 [25]. This model, after being trained using, e.g., algorithm 1 should be able to generate new data that look like the ones in the training dataset. An example where the GAN was trained using the famous MNIST dataset, containing images of handwritten digits, is shown in figure 2.3.

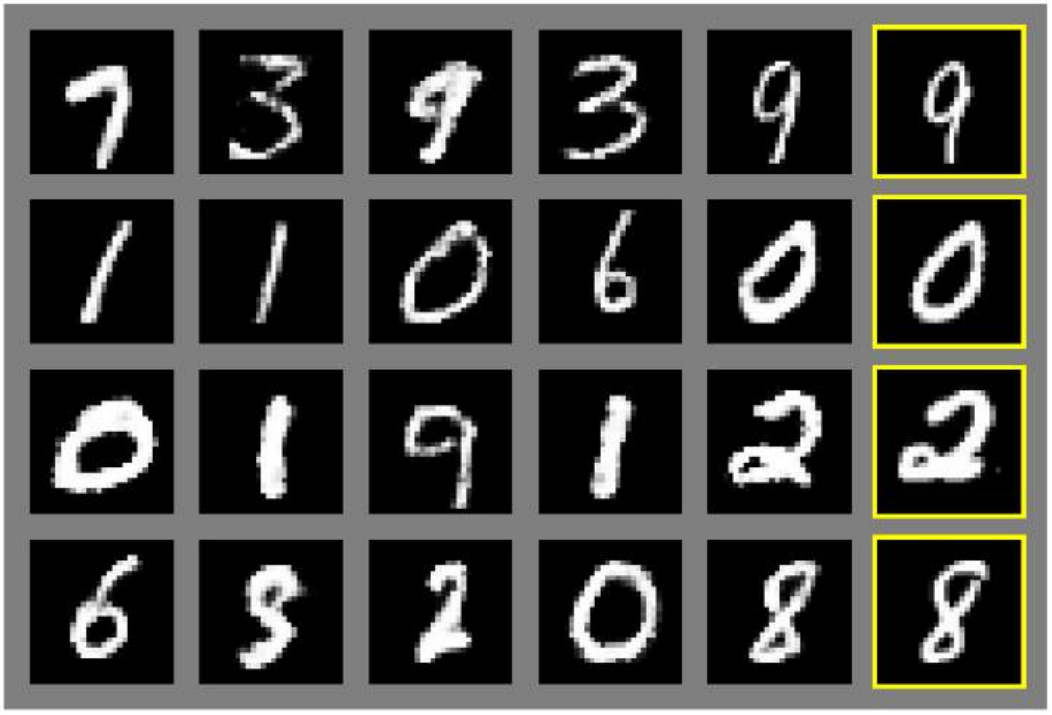


Figure 2.3: *Example taken from [25], in this case, the GAN had been trained with the MNIST dataset. The rightmost column shows the nearest training example of the neighboring sample, to demonstrate that the model has not memorized the training dataset. Every other picture is an original generation of the GAN, not present in the training dataset.*

In the years, a considerable amount of types of GANs have been introduced by numerous papers, some are improvements of the architecture of already existing GANs, and others are slight modifications that allow these networks to perform different tasks other than generating synthetic samples.
A simple improvement from vanilla GAN is a conditional GAN [30], where both generator and discriminator are conditioned on some extra information c, which could be, e.g., class labels. Conditioning is performed simply by feeding c into both generator and discriminator as additional input layer (see figure 2.4).
The minimax equation in this case is just slightly different than for the simple GAN [30] (confront with equation 2.6):

$$G^* = \arg\min_G \max_D \Big\{ \mathbb{E}_{x\sim p_{data}(x)} \Big[\log\big(D(x|c)\big) \Big] + \mathbb{E}_{z\sim p_{noise}(z)} \Big[\log\Big(1 - D\big(G(z|c)|c\big)\Big) \Big] \Big\} \tag{2.11}$$

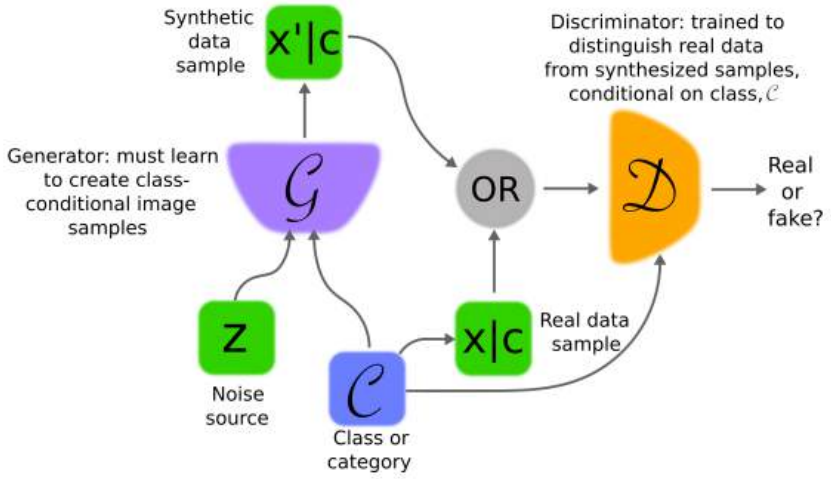


Figure 2.4: *Structure of a conditional GAN [24].*

With this architecture we can, for instance, train the GAN on the MNIST dataset where each training sample contains also the label indicating which digit is represented in the image. After the training, we can condition the generator to provide only images of digits associated with one label (e.g. only images of hand-written nines). We see an example in figure 2.5.

Figure 2.5: *Generated MNIST digits, each row now conditioned to one label [30].*

2.3.2 Deep Convolutional GAN

When a neural network is supposed to extract information from images, the favored architecture is not a feedforward type, but rather one based on convolutional layers [34]. Convolutional neural networks (CNNs) provide impressive results in most image-related tasks, like image classification or image segmentation (i.e., identifying the objects that are present in pictures). If we want to create a GAN able to perform Image-to-Image translation we should consider turning both generator and discriminator into CNNs. Now

I will introduce the basic concepts of CNNs and then we will see how they are applied in Deep Convolutional GANs (DCGANs).

Convolutional layers

The core operation of a convolutional layer is the convolution. It involves taking a filter (also known as kernel), which is basically a three-dimensional matrix, and sliding it over the input to perform element-wise multiplications and summations (see figure 2.6).

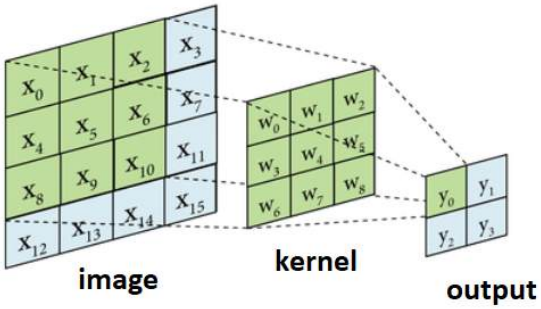


Figure 2.6: *Convolution operation, applied in this case to a single-channel image (black & white), so both the image and the kernel are represented by two-dimensional matrices.*

$y_0 = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 + w_5x_6 + w_6x_8 + w_7x_9 + w_8x_{10}$

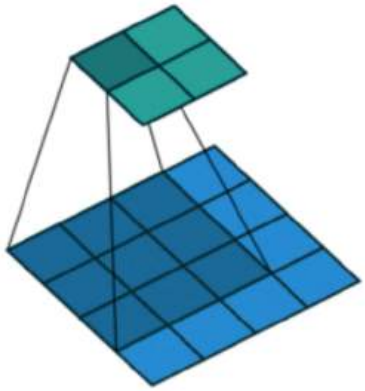
$y_1 = w_0x_1 + w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 + w_5x_7 + w_6x_9 + w_7x_{10} + w_8x_{11}$

$y_2 = w_0x_4 + w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 + w_5x_{10} + w_6x_{12} + w_7x_{13} + w_8x_{14}$

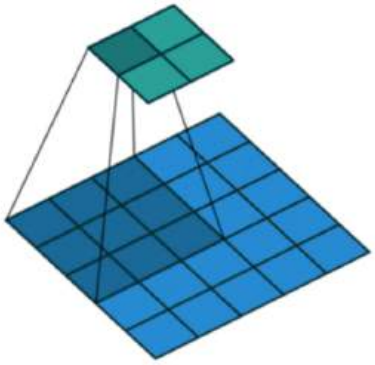
$y_3 = w_0x_5 + w_1x_6 + w_2x_7 + w_3x_9 + w_4x_{10} + w_5x_{11} + w_6x_{13} + w_7x_{14} + w_8x_{15}$

The output of the convolution operation is a feature map. Each feature map represents the presence of a specific feature or pattern in the input data. By using multiple filters, a convolutional layer can learn to detect various features, such as edges, textures, or more complex shapes. The elements of the filters' matrices are our new learnable parameters, which replace the weights of the links between neurons in the feedforward architecture and are trained using the same backpropagation algorithm as we saw in section 2.2.2.
An important parameter of the convolution operation is the *stride*, which determines how much the filter moves between steps. The standard value of stride is 1 and it means that the filter moves one pixel to the left (or to the bottom). A convolution with a stride value of 1 is sometimes referred to as "non-strided convolution", while if the stride is 2 or more it is called "strided convolution" (see figure 2.7). A "fractional-strided convolution" is one where the filter moves only of one pixel after each step, but a circle of zeros is added around every pixel, this operation is also known as "transposed convolution" or also, erroneously, as "deconvolution" (see figure 2.8). Note that the larger the stride, the more the loss in size of the extracted feature map compared to the original input, on the other hand, a fractional-strided convolution will usually produce a feature map that is larger than the input.
Another important parameter is the padding, it adds additional zeros around the whole input matrix to control the size of the output feature map (see figure 2.9).
Note that the discriminator gets fed an image as input and outputs a scalar, so, across the network, the dimensionality of the input must be reduced drastically. By contrast, the generator usually gets fed a low-dimensional noise vector and must produce a high-dimensional image as output, so its input must be enlarged. This up-and-down scaling

of the network's input is possible thanks to padding and the various types of strided convolutions, as well as the "pooling layers" described below.



(a) *Convolution with standard stride value (non-strided convolution). The filter gets shifted by 1 pixel after every step.*



(b) *Convolution with stride value of 2 (strided convolution). The filter gets shifted by two pixels after every step.*

Figure 2.7: *Non-strided vs strided convolution.*

Pooling layers

The preferred way to implement convolutional layers is the one where padding and stride are adjusted so as to have a fixed size, meaning that the output feature map must have the same size as the input. The most popular way to reduce the dimensionality of the input is, instead, by using pooling layers [34]. A common pooling layer is the 2×2 max pooling, where the image gets divided into 2×2 squares and, within each square, only the largest pixel value is selected (see figure 2.10).

Fully connected layers

After having done feature extractions with convolutional layers, it is common practice to flatten out the last feature map we obtain and feed it to fully connected layers (that are the same type of layers that we have in the feedforward architecture) as the last portion of the network (see figure 2.11).

Activation functions

In the case of convolutional layers, the non-linear activation function is applied after every convolution operation. The most popular activation functions to use are the hyperbolic tangent, the sigmoid, the rectified linear unit (or ReLU), defined by the following equation:

$$ReLU(x) = \max(0, x) \tag{2.12}$$

and the LeakyReLU:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha \cdot x, & \text{if } x < 0 \end{cases} \tag{2.13}$$

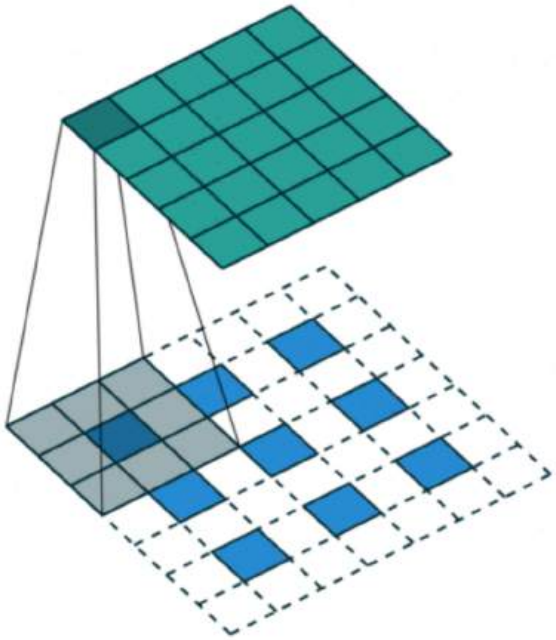


Figure 2.8: *Fractional-strided convolution. The filter gets shifted by one pixel after every step, but a circle of zero is added around each pixel. The size of the output feature map is larger than that of the input image.*

with α small (usually $\alpha = 0.01$).

Many attempts to scale up GANs using CNNs as base architecture for both the generator and the discriminator have been unsuccessful. In the paper [31] the authors list a series of guidelines to respect to obtain a stable DCGAN. These are:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator);
- Use Batch Normalization (a technique used to improve training stability and speed by normalizing the input of each layer within a mini-batch [33]) in both the generator and the discriminator;
- Remove fully connected layers;
- Use ReLU activation in the generator for all layers except for the output, which uses hyperbolic tangent;
- Use LeakyReLU activation in the discriminator for all layers.

Figure 2.12 shows the architecture of a DCGAN generator following the above guidelines. DCGANs have been tested successfully in image generation and other tasks like Image-to-Image translation, as we will see in the following section.

2.4 GANs for Image-to-Image translation

We define *image-to-image translation* as a controlled conversion (or mapping) of a given source image to a target image. In this section, I will describe the two architectures that

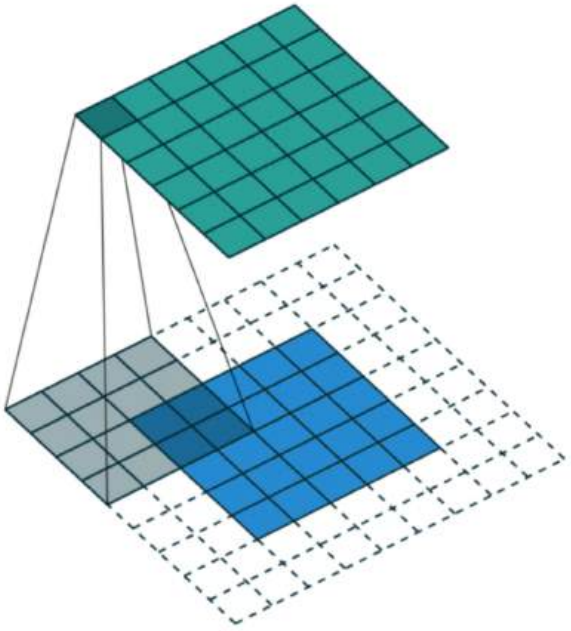


Figure 2.9: *Non-strided convolution with padding. The padding value is 2 in this case since two circles of zeros are added around the input image.*

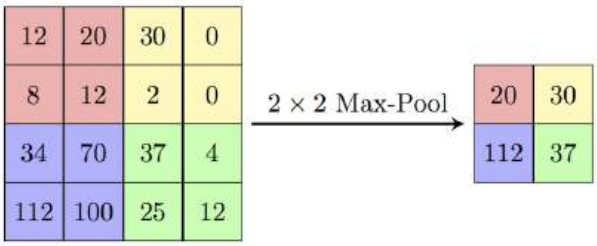


Figure 2.10: 2×2 *max pooling. The* 4×4 *image is divided into four* 2×2 *squares, within each square only the largest value is selected. After applying the max pooling layer, the output image is half-sized.*

I implemented trying to achieve this task.

2.4.1 Pix2Pix

The Pix2Pix model [35] is a deep convolutional GAN where both the generator and the discriminator are conditioned on some extra input. This kind of model is suitable for image-to-image translation tasks.
In the previous section, we have seen a GAN that learns a mapping from random noise vector z to output image y, $G : z \to y$. In contrast, this GAN learns a mapping from observed image x and random noise vector z, to y, $G : \{x, z\} \to y$. In the jargon of Image-to-Image translation, we call x the *observed image* and y the *ground truth*, while $G(x, z)$ is the *generated image*. A generation is successful if the generated image is "close to" the ground truth (we will see later how to decide whether two images are close to each other).
Important to note is that without providing z to the generator, the net could still learn a mapping from x to y, but it would produce deterministic outputs, and therefore fail to

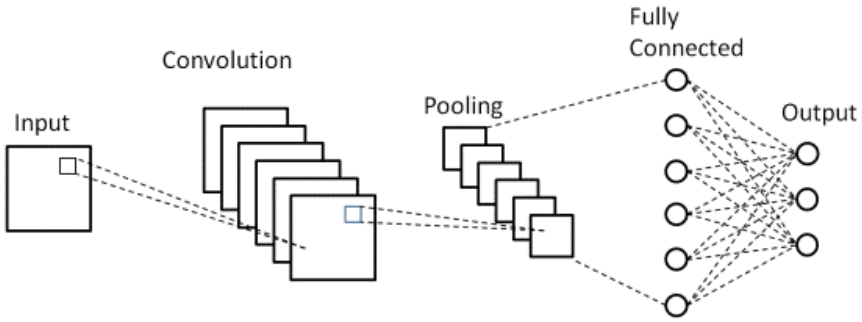


Figure 2.11: *Schematized structure of a standard convolutional neural network.*

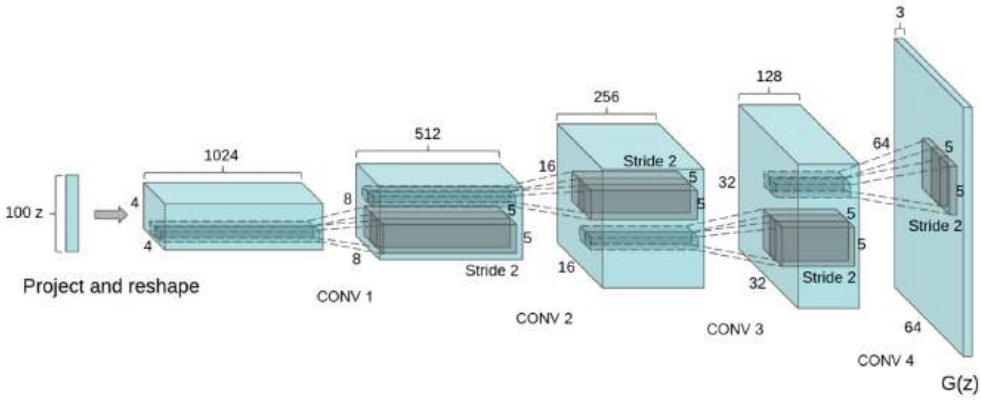


Figure 2.12: *Example of DCGAN generator. A 100-dimensional uniform distribution z is projected to a small spatial extent convolutional representation with many feature maps. A series of fractional-strided convolutions convert this high-level representation into a* 64×64 *pixel image. Notably, no fully connected or pooling layers are used [31].*

match any distribution other than a delta function. However, giving z as additional input to the generator is found to be not effective since the Pix2Pix generator simply learns to ignore the noise. The solution is to provide noise only in the form of dropout, a technique that consists of setting to zero some randomly selected elements of feature maps to add stochasticity, and it is applied on several layers of the generator at both training and test time. I will still use the notation $G(x, z)$ to point out that noise is still provided in some form to the generator. Empirical results showed that the network learns better if also the discriminator is conditioned to the observed image x, so rather than learning to discern y from $G(x, z)$, it is better if it learns to discriminate tuples: $\{x, y\}$ vs. $\{x, G(x, z)\}$. Figure 2.13 shows a schematic representation of the input structure of Pix2Pix.
With this knowledge, and remembering equation 2.3.1. We can write the objective function of a conditional GAN (cGAN) as:

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y}\Big[\log\big(D(x, y)\big)\Big] + \mathbb{E}_{x,z}\Big[\log\Big(1 - D\big(x, G(x, z)\big)\Big)\Big] \tag{2.14}$$

G tries to minimize this objective function and D tries to maximize it. The best possible generator is, as usual, given by $G^* = \arg\min_G \max_D L_{cGAN}(G, D)$.
To obtain the loss for our Pix2Pix generator we must add another term that forces the generator's output to be near the ground truth y. Usually for this loss term the $L1$

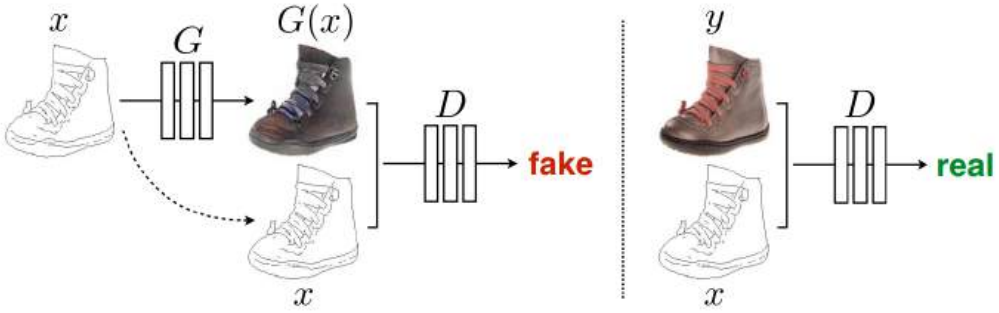


Figure 2.13: *Training a Pix2Pix to map edges→photo. The discriminator D learns to classify between fake and real {edge, photo} tuples. The generator G learns to provide a photo picture from an edge picture and fool the discriminator.*

distance[3] is chosen rather then $L2$[4] as $L1$ encourages less blurring:

$$L_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1] \tag{2.15}$$

So the final objective is:

$$G^* = \arg\min_G \Big[\max_D L_{cGAN}(G,D) + \lambda L_{L1}(G) \Big] \tag{2.16}$$

Where λ defines the relative importance of the two objectives (the standard value is $\lambda = 100$).

Generator Architecture

The architecture used for the generator is called U-Net (see figure 2.14):

- Input passes through a series of layers that progressively downsample, until a bottleneck layer (*encoder*);
- The process is reversed, successive layers progressively upsample up to the original dimension (*decoder*);
- The network is allowed to shuttle part of the information directly across the net (without passing through the bottleneck) thanks to *skip connections*.

Note that the rules defined in section 2.3.2 still apply: in the encoder section we use modules that implement strided convolution, batch normalization, and LeakyReLU as activation functions; in the decoder section the modules implement fractional-strided convolution, batch normalization, and ReLU as activation function.

Discriminator Architecture

The chosen discriminator architecture is called PatchGAN:

[3] The $L1$ distance between two images x and y is defined as: $\|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$ where n is the number of pixels of both images. x_i and y_i indicate the value of the i-th pixel of image x and image y respectively.

[4] With the same definitions of footnote 3, the L2 distance is defined as: $\|x - y\|_2 = \sum_{i=1}^{n} |x_i - y_i|^2$

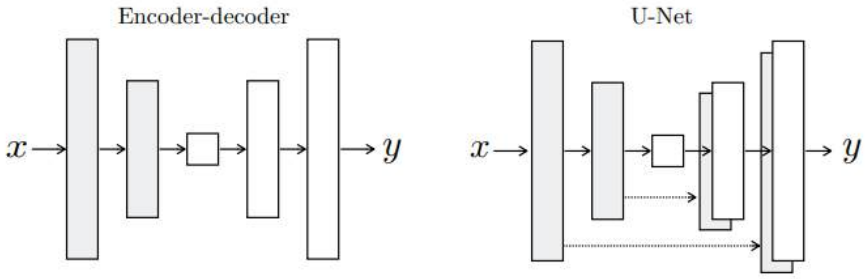


Figure 2.14: *The U-Net is an encoder-decoder architecture with skip connections between mirrored layers in the encoder and decoder stacks.*

- The image is divided into $N \times N$ patches;
- The discriminator is run convolutionally across the image, producing a value for each patch;
- Responses are averaged to give the ultimate output of D.

This discriminator's architecture is chosen because the $L1$ loss forces low-frequency correctness. To model high frequencies we restrict our attention to local image patches.

Figures 2.15 show an example of the application of this architecture.

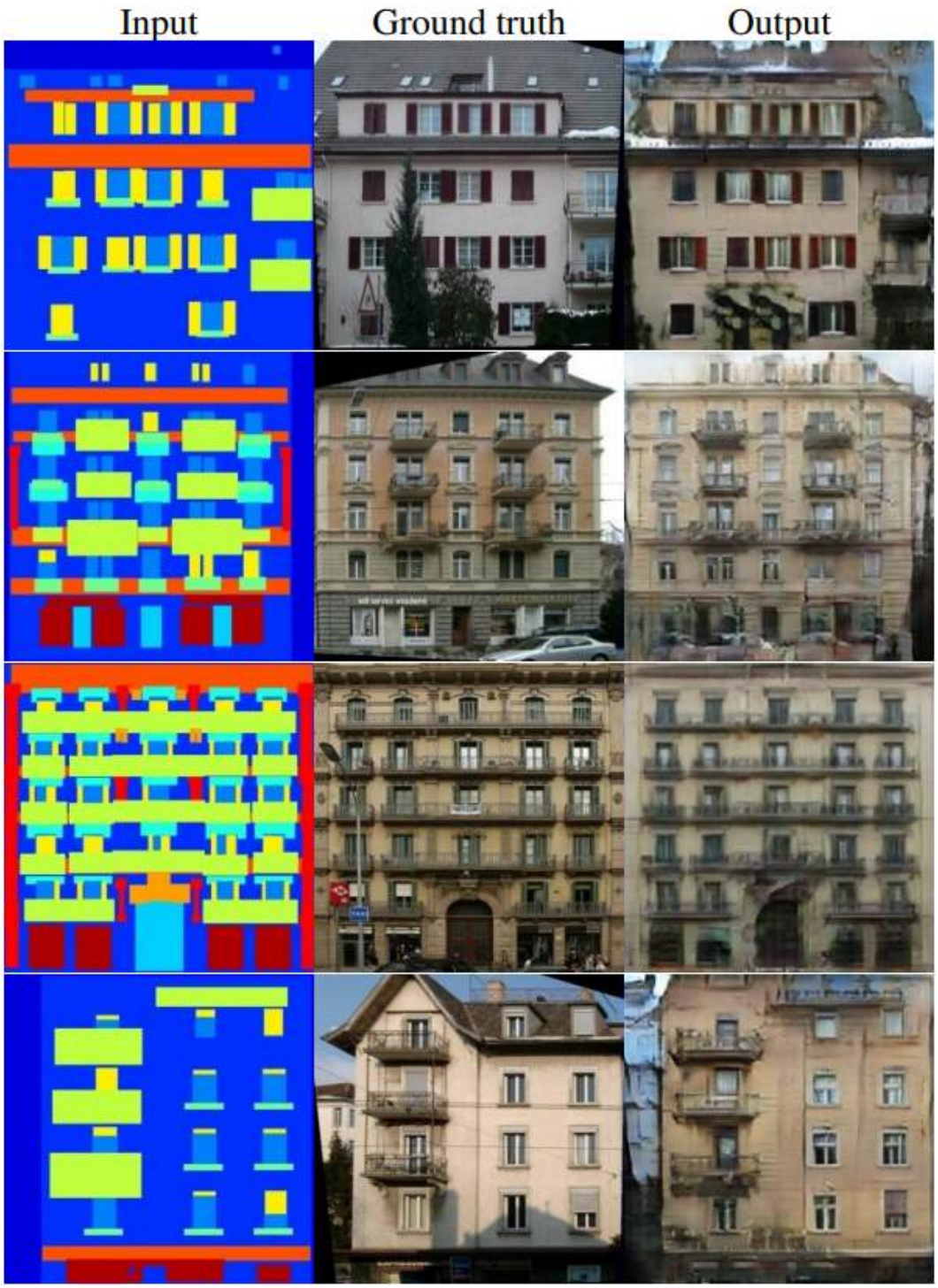


Figure 2.15: *Example results of Pix2Pix applied on facades labels→photo [35].*

2.4.2 CycleGAN

CycleGAN [36] is a model that allows to translate an image from a source domain X to a target domain Y in the *absence of paired samples*, as opposed to Pix2Pix (see figure 2.16).
The idea is to let the model learn a mapping $G : X \to Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y (we write $G(X) \approx Y$),

and also the inverse mapping $F : Y \to X$. Then we introduce a *cycle-consistency loss* to push $F(G(X)) \approx X$ and $G(F(Y)) \approx Y$.

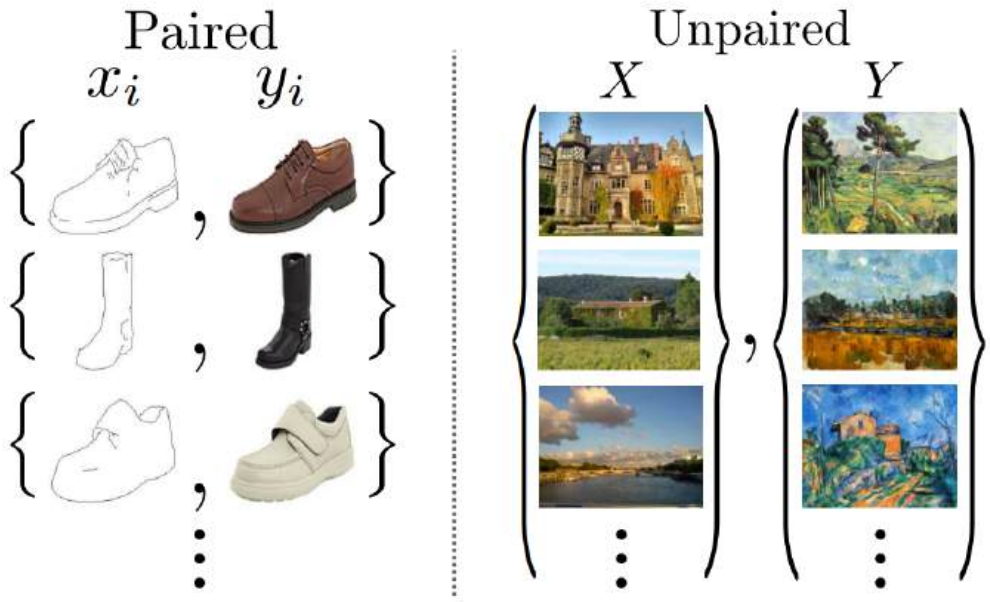


Figure 2.16: *Paired training data (left) consist of training examples* $\{x_i, y_i\}_{i=1}^{n}$ *where the* y_i *that corresponds to each* x_i *is given. We instead consider unpaired training data (right), consisting of a source set* $\{x_i\}_{i=1}^{N} \in X$ *and a target set* $\{y_j\}_{j=1}^{M} \in Y$*, with no information provided as to which* x_i *matches which* y_j*.*

Since the model implements two generators, G and F, we must introduce two adversarial discriminators D_X and D_Y, where D_X aims to distinguish between real images $x \in X$ and fake translated images $F(y), y \in Y$ and D_Y aims to distinguish real $y \in Y$ from $G(x), x \in X$. The architectures of the discriminators and generators are very similar to the ones in Pix2Pix, the discriminators are PatchGANs and the generators are again based on U-Net [37].
We apply adversarial losses to both mapping functions:

$$L_{GAN}(G, D_Y) = \mathbb{E}_{y \sim p_{data}(y)}\Big[\log\big(D_Y(y)\big)\Big] + \mathbb{E}_{x \sim p_{data}(x)}\Big[\log\Big(1 - D_Y\big(G(x)\big)\Big)\Big] \tag{2.17}$$

$$L_{GAN}(F, D_X) = \mathbb{E}_{x \sim p_{data}(x)}\Big[\log\big(D_X(x)\big)\Big] + \mathbb{E}_{y \sim p_{data}(y)}\Big[\log\Big(1 - D_X\big(F(y)\big)\Big)\Big] \tag{2.18}$$

And use the $L1$ distance to introduce the cycle consistency loss:

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}\Big[\|F\big(G(x)\big) - x\|_1\Big] + \mathbb{E}_{y \sim p_{data}(y)}\Big[\|G\big(F(y)\big) - y\|_1\Big] \tag{2.19}$$

So we can write the final objective function as:

$$L(F, G, D_X, D_Y) = L_{GAN}(G, D_Y) + L_{GAN}(F, D_X) + \lambda L_{cyc}(G, F) \tag{2.20}$$

where λ, again, controls the relative importance of the different objectives (this time the standard value is $\lambda = 10$). As usual, generators G and F will try to minimize the objective function while discriminator D_X will try to maximize $L_{GAN}(F, D_X)$ and discriminator D_Y will try to maximize $L_{GAN}(G, D_Y)$:

$$G^*, F^* = \arg\min_{G,F} \max_{D_X, D_Y} L(G, F, D_X, D_Y) \tag{2.21}$$

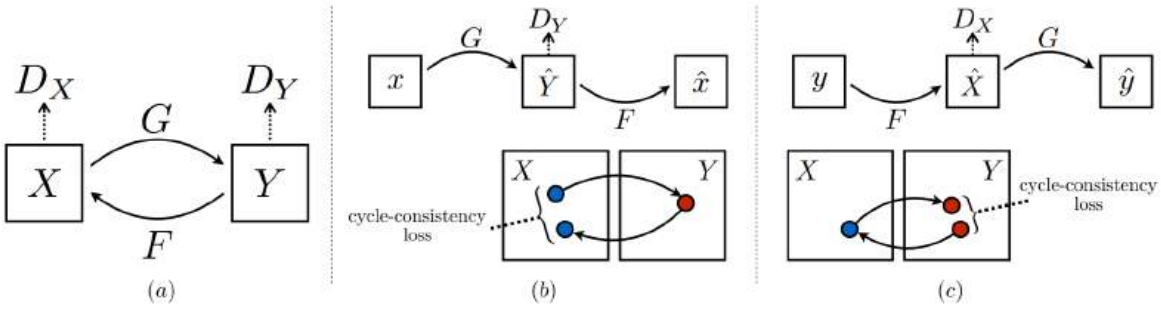


Figure 2.17: *(a) The model contains two mapping functions (generators) $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X outputs indistinguishable from domain Y, and vice versa for D_X, F and X. To further regularize the mappings, two "cycle-consistency losses" are introduced, that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$.*

Figure 2.17 shows a schematic representation of the model.
Figure 2.18 shows an example of the application of a CycleGAN.

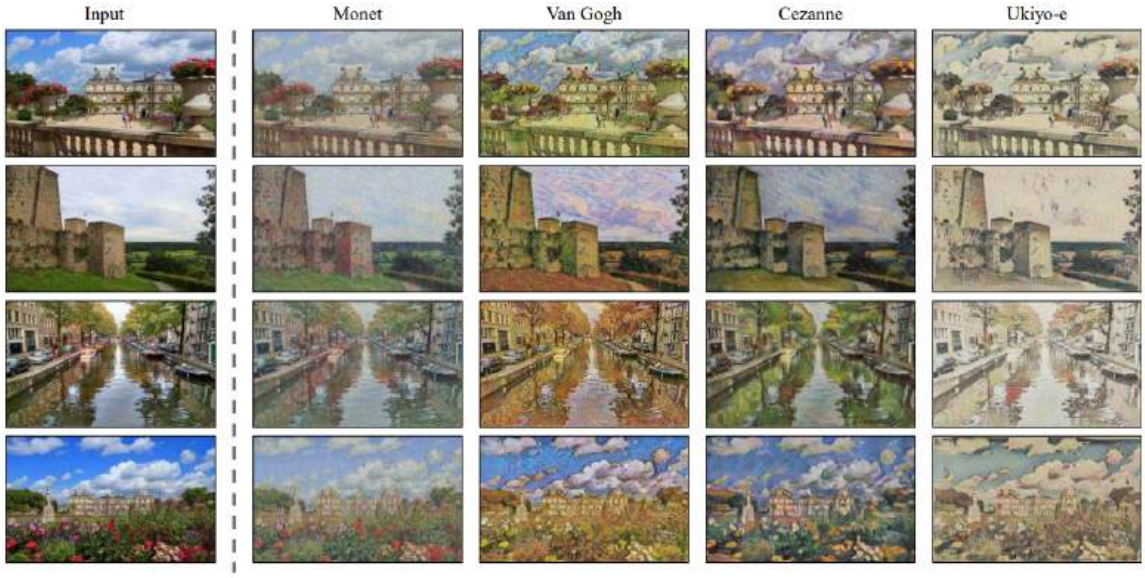


Figure 2.18: *Transfer of input images into different artistic styles done with CycleGAN [36].*

Chapter 3

Methodology

In this chapter, I will introduce the actual work I did, based on the so far described theory and motivation. In sections 3.1 and 3.2 I will talk about the data I had at my disposal and the preprocessing I needed to perform before using them to train the Pix2Pix and CycleGAN models. Section 3.3 will present some details on the training process, including the computational resources I had available and the metric I used to evaluate the performances of the models during training.

3.1 Available data

For my analysis, I had available a dataset of glitches detected by the Virgo interferometer during observation run O3a. In particular: 855 glitches of the "Scattered Light" class; 370 glitches of the "Blip" class; 389 glitches of the "Koi Fish" class; and 335 glitches of the "Low-Frequency Line" class.
Data for each glitch event consists of 8 different time series, each lasting 16 seconds with the glitch centered in the middle of the time axis. One time series is for the strain channel and the others are for 7 auxiliary channels, which are:

1. "*V1:INJ_IMC_TRA_DC*": Monitors the transmitted power, specifically the direct current component, of the laser beam within the Injection Mode Cleaner (IMC). The IMC is an optical cavity within the interferometer used to enhance the laser beam quality before it enters the main interferometer arms.

2. "*V1:LSC_DARM_ERR*": Monitors the deviation in the differential arm length measurement.

3. "*V1:LSC_MICH_ERR*": Monitors the error signal associated with the Michelson Interferometer. It provides information about the discrepancy between the desired and actual state of the interferometer's mirrors, aiding in their control and alignment for gravitational wave detection.

4. "*V1:LSC_NE_CORR*": Monitors the noise correlation between the two interferometer arms, which are also referred to as North and East arms since one is aligned in the North-South direction and the other in the East-West direction.

5. "*V1:LSC_PRCL_ERR*": Monitors the deviation in the length of the Power Recycling Cavity (PRC). The PRC is a component of the interferometer that enhances the sensitivity by recycling and amplifying the laser power.

6. "*V1:LSC_PR_CORR*": Represents a signal that provides correction information to control the length of the PRC. By continuously monitoring and adjusting this

cavity length, the interferometer can optimize the coupling of the laser light, enhancing the overall performance and sensitivity.

7. "*V1:Sc_MC_MIR_Z_CORR*": Refers to the correction signal related to the position along the Z axis[1] of mirrors within the Mode Cleaner (MC). The MC consists of optical components such as lenses, mirrors, and other elements designed to select and clean a specific spatial mode of the laser beam. This process removes higher-order modes and ensures that the beam is in a well-defined, stable spatial mode.

Moreover, each data sample comes with metadata regarding the peak frequency of the glitch, its duration, the GPS time at which it happened, the confidence with which it has been classified into its glitch class, and other less interesting variables. These metadata can provide us with useful information: for instance, we can plot the distribution of the peak frequency of Scattered Light glitches (figure 3.1) and see that the peak frequency is always in the range $[10, 100]$ Hz and the 99.5-th percentile is found at 50 Hz, indicating that these types of glitches are confined at low frequencies.

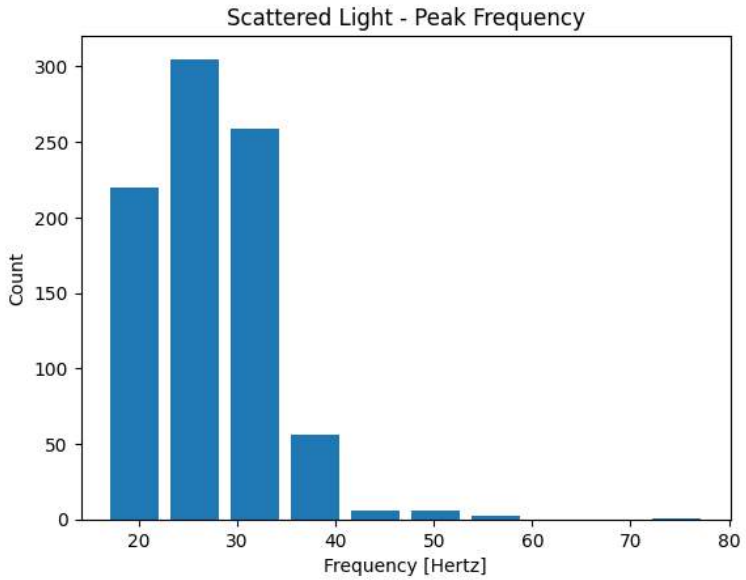


Figure 3.1: *Scattered Light glitches, peak frequency distribution. The 95-th percentile is* $P_{95} = 50$ *Hz.*

3.2 Preprocessing

In this section I will explain in detail what are all the steps I used to construct the training and testing set for the Generative Adversarial Network models, consisting of images representing spectrograms of glitch events, starting from the time series that I had available as raw data.

[1]In the context of Virgo's coordinate system, the Z axis does not refer to the vertical direction but rather to the direction longitudinal to one of the interferometer's arms.

3.2.1 Whitening

The combination of all the noise sources in a detector produces a time series $n(t)$ that can be represented by a vector $\mathbf{n}$, with components given by the discrete time samples $n_i = n(t_i)$. The noise is described as a stochastic process with statistical properties given by the joint probability distribution $p(\mathbf{n})$. The noise in the Virgo detector is, with isolated exceptions, *Gaussian* and approximately *stationary*. Gaussian means that the joint probability distribution follows a multivariate normal distribution:

$$p(\mathbf{n}) = \frac{1}{\det(2\pi\mathbf{C})^{1/2}} \exp\left[-\frac{1}{2}\sum_{ij}(n_i - \mu)(n_j - \mu)C_{ij}^{-1} \right] \tag{3.1}$$

Where $\mathbf{C}$ is the covariance matrix and μ is the mean value.
Stationary means that C_{ij} depends only on the lag $|i-j|$, so stationary noise can be characterized by the correlation function $C(\tau)$ where $\tau = |t_i - t_j|$ is the time lag. Transforming to the Fourier domain, where the labels i, j now refer to frequencies f_i, f_j, stationary noise has a diagonal covariance matrix $C_{ij} = \delta_{ij}S_n(f_i)$, which defines the power spectral density $S_n(f)$. The power spectral density is given by the Fourier transform of the correlation function $C(\tau)$. Amplitude spectral density is the square root of power spectral density and has units of $\mathrm{Hz}^{-1/2}$.
Stationary Gaussian noise is uncorrelated between frequency bins and the noise $\tilde{n}(f)$ in each bin follows a Gaussian distribution with random phase and amplitude $S_n^{1/2}(f)$.
The first step in GW data analysis is most often to divide the Fourier coefficients by an estimate of the amplitude spectral density of the noise, which ensures that the data in each frequency bin has equal significance by down-weighting frequencies where the noise is loud. This process is called *whitening* because it leaves us with white noise[2]. Let $d(t)$ be our strain vs. time data, the pipeline for this first step will be (FFT stands for Fast Fourier Transform [6], a fast algorithm to calculate the Fourier transform):

$$d(t) \xrightarrow{\text{FFT}} \tilde{d}(f) \xrightarrow{\text{Whiten}} \tilde{d_w}(f) = \frac{\tilde{d}(f)}{S_n^{1/2}(f)} \xrightarrow{\text{iFFT}} d_w(t) \tag{3.2}$$

Note that the FFT assumes that the stretch of data being transformed is periodic in time: a window function[3] must be applied to the data before taking the FFT in order to suppress spectral leakage, which causes spurious frequency components to appear in the spectrum. The whitening function provided by the python library GWpy [20], that I used, automatically applies a Hann window function [12] to the input signal.
Figure 3.2 shows a time series of the strain channel from the Scattered Light glitches dataset, plotted in its full length of 16 seconds.

[2]Noise is referred to as white if $C_{ij} = \delta_{ij}\sigma^2$, in both the frequency domain and the time domain, i.e. every frequency is equally represented in the noise spectrum

[3]A window function is a mathematical function that is zero-valued outside of a chosen interval, it is usually symmetric around the middle of the interval with maximum in the middle point. Its effect is to taper off the edges of a signal, erasing the discontinuity that could be present if that signal is a time slice of a longer one.

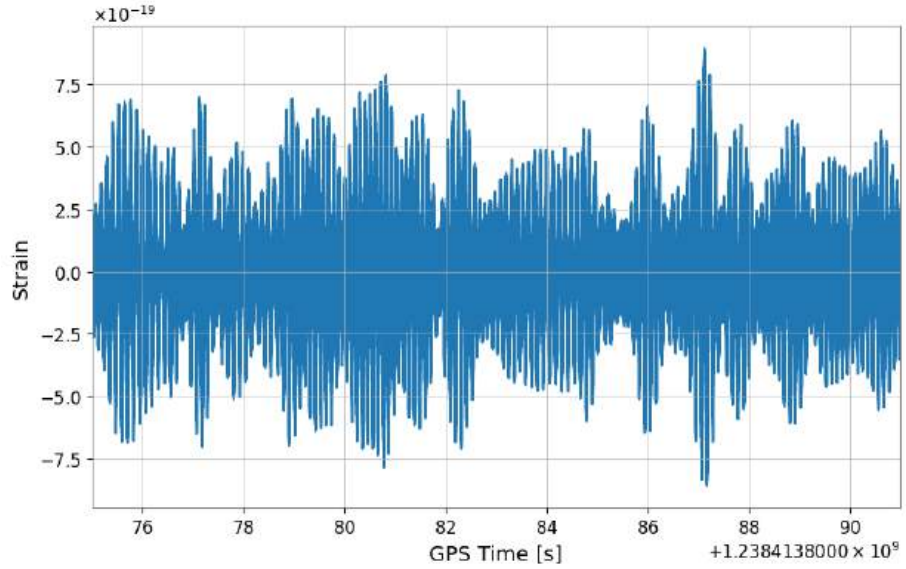


Figure 3.2: *Time series from the Scattered Light glitches dataset, full length. Sampled at 4096 samples per second.*

We cannot identify the glitch from this raw time series. First of all, knowing that the glitch is centered in the time axis, we should plot a time slice of (e.g.) one second in the middle of the time series, as shown in figure 3.3.

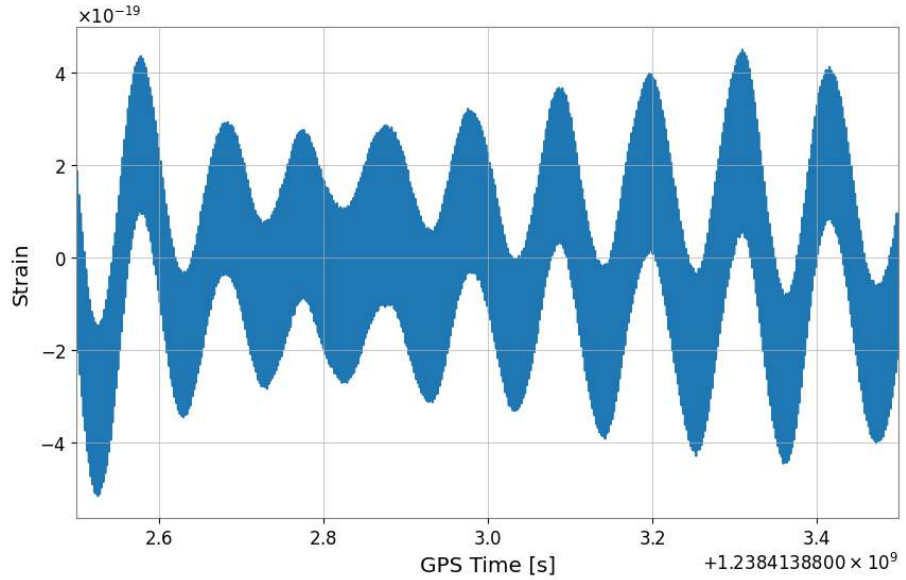


Figure 3.3: *Time series from the Scattered Light glitches dataset, 1 second long, centered on time axis.*

As we can see, the oscillations are dominated by low-frequency noise. Now we perform whitening to cut off the dominant noise frequencies from the signal (see figure 3.4).

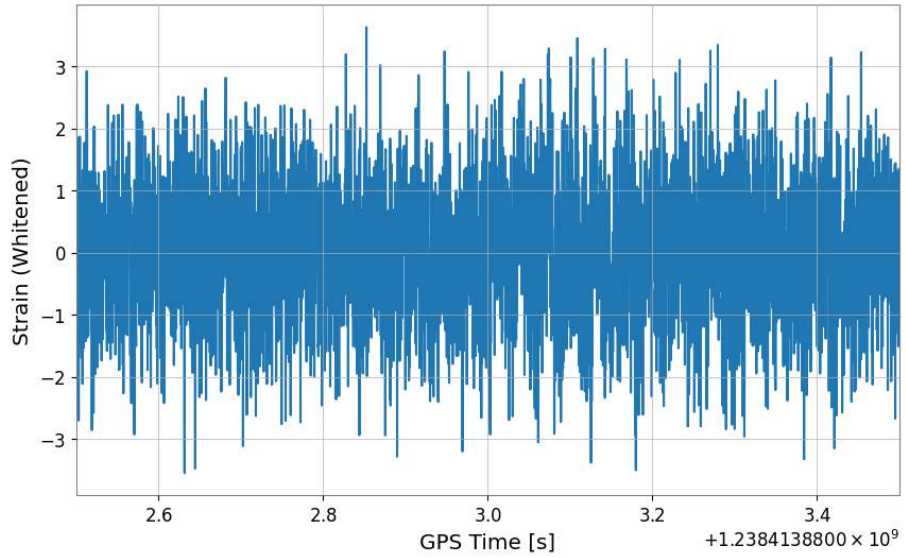


Figure 3.4: *Time series from the Scattered Light glitches dataset, 1 second long, centered on time axis. Windowed and whitened.*

Now we remain with signal and white noise, if we want to be able to spot the glitch we just need to bandpass the time series to extract the frequency band where we expect the signal to be. Since the glitch in question is of the class Scattered Light and we know them to be concentrated at low-frequencies, as I discussed in section 3.1, we try with the band $[10\text{Hz}, 50\text{Hz}]$ (see figure 3.5).

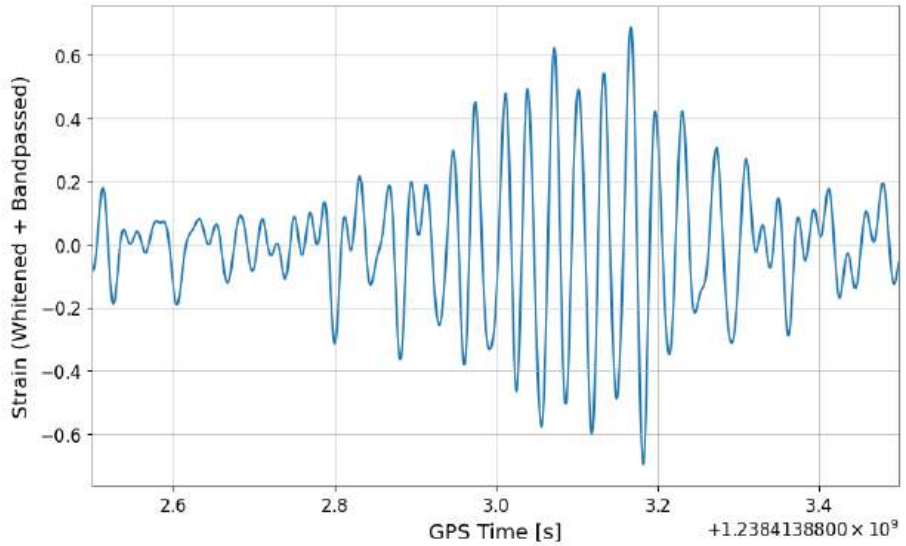


Figure 3.5: *Time series from the Scattered Light glitches dataset, 1 second long, centered on time axis. Whitened and bandpassed in frequency band* $[10Hz, 50Hz]$.

Now, having filtered all the noise outside the frequency band of interest, the glitch is finally visible. One thing that we should notice is that it is not quite in the center but it is slightly moved to the right. Later we will see why this is quite a big problem and how I solved it.

3.2.2 Creating spectrograms

As I said in section 1.2.2, we are especially interested in the time evolution of the frequency spectrum of glitches, i.e., in their spectrogram. The simplest algorithm for creating spectrograms is called the Short Time Fourier Transform, but in the context of glitch analysis, usually we prefer to use the Q transform. Since the second algorithm is based on the first one, I will now briefly describe both.

Short Time Fourier Transform

We know that the Fourier transform (FT) is a powerful tool that can provide information about what frequencies are represented in a certain signal. Take for example an audio signal of some sound played by a musical instrument, if we want to know which notes are played by the instrument we can apply the Fourier transform to that audio signal as we can see in figure 3.6.

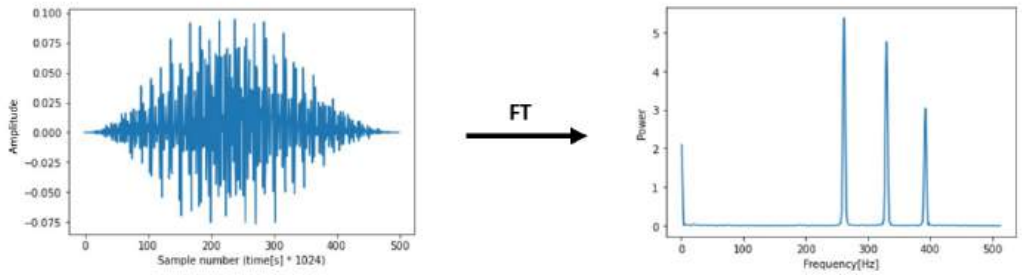


Figure 3.6: *On the left a windowed audio signal (sampled at 1024 samples per second) of a grand piano playing a C major chord, i.e. playing the notes C (*261 *Hz), E (*329 *Hz), G (*392 *Hz) simultaneously. On the right the spectrum analysis of that signal done with an FT.*

The Fourier transform correctly identifies the frequencies played by the instrument. Notice that in the case of figure 3.6 the audio file is of a piano playing a C major chord (i.e., playing three notes simultaneously), if instead, the audio was of a piano playing a C major arpeggio (i.e., playing three notes one after the other) the result of the FT would be the one shown in figure 3.7.

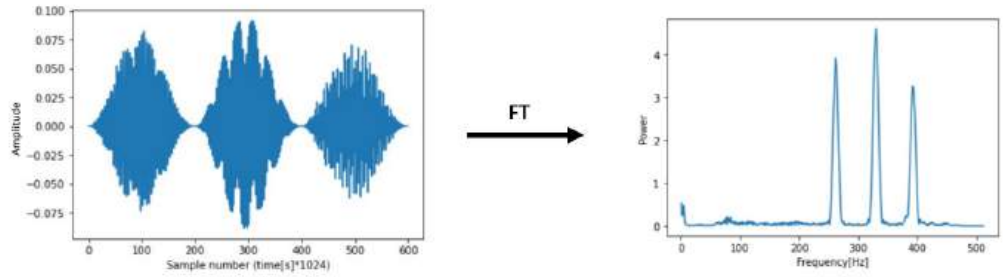


Figure 3.7: *On the left is an audio signal of a grand piano playing a C major arpeggio. On the right the spectrum analysis of that signal, done with an FT. Note that to produce a less noisy spectrum each note signal in the audio file has been separately windowed.*

As we can see the output of the Fourier transform is almost identical, except for being a bit more noisy. This is because the FT provides us with information about the frequencies

that compose a signal but does not tell us anything about the time at which each of these frequencies is present.
If we want to obtain that kind of information, the most straightforward algorithm is called Short Time Fourier Transform or STFT. In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length, which is done by sliding a short-time window function on the signal and then computing the FT separately on each segment. This reveals the Fourier spectrum on each shorter segment, one can then plot the changing spectra as a function of time, i.e. the *spectrogram* as can be seen in figure 3.8.

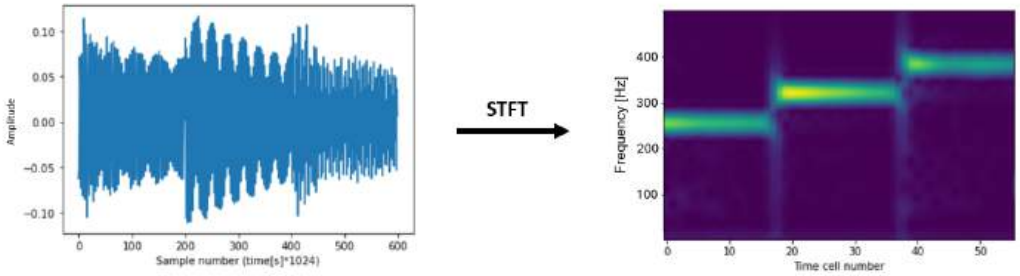


Figure 3.8: *On the left is an audio signal (sampled at 1024 samples per second) of a grand piano playing a C major arpeggio. On the right is the spectrogram of that signal, obtained with an STFT. The colormap in the spectrogram indicates the weight of each frequency at a certain time. Note that in this case the audio signal on the left has not been windowed in advance, this is because we already apply a window function at each step when calculating the STFT.*

Mathematically, in the continuous-time case, the Short Time Fourier Transform calculated at time $t = \tau$ and frequency $f = \phi$ can be written as [11]:

$$X(\tau, \phi) = \int_{-\infty}^{+\infty} x(t) w(t - \tau) e^{-i2\pi\phi t} dt \tag{3.3}$$

Where $w(t)$ is commonly a Hann window [12] or Gaussian window (section 3.2.2) centered around zero. This formula will be helpful when, just below, I will introduce the Q transform.

Q transform

A downside of the Short Time Fourier Transform is that it has a fixed resolution. The width of the windowing function relates to how the signal is represented: a wide window gives better frequency resolution but poor time resolution, and vice versa a narrower window gives good time resolution but poor frequency resolution. We would prefer an algorithm able to give good time resolution for high-frequency events and good frequency resolution for low-frequency events. One possible solution that can be implemented is to have a window that adapts its length based on the frequency it is calculating. This is the strategy adopted by the Q transform algorithm. Indeed the mathematical formula for the continuous-time case of the Q transform calculated at time τ and frequency ϕ is [14]:

$$X(\tau, \phi, Q) = \int_{-\infty}^{+\infty} x(t) w(t - \tau, \phi, Q) e^{-i2\pi\phi t} dt \tag{3.4}$$

Comparing it to equation 3.3, the only difference, other than the additional parameter Q to which the transform depends, is that now the window function (actually the *width* of the window function) depends also on the frequency ϕ that it is calculating. In particular, the window chosen is Gaussian and will have the equation:

$$w(t-\tau,\phi,Q) = \frac{C}{\sqrt{2\pi}\sigma_t(Q,\phi)} \exp\left[-\frac{(t-\tau)^2}{2\sigma_t(Q,\phi)^2}\right] \tag{3.5}$$

Where C is a normalization constant and $\sigma_t(Q,f)^2 = \frac{Q^2}{8\pi^2 f^2}$ [14]. The window defines both the resolution in time domain and frequency domain since the Fourier transform of the window function will still be a real-valued Gaussian with variance (dropping the explicit dependence notation) $\sigma_f^2 = \frac{2f^2}{Q^2}$. Note that in frequency domain $\sigma_f \propto f$ while in time domain $\sigma_t \propto 1/f$.
In practice, we can think of the Q transform as the application of a series of filters with central frequency f_k and spectral width δf_k. At low f, σ_f is small, meaning that filters are narrow and allow to distinguish between close frequencies, on the contrary, σ_t is large so those filters scan the signal for these frequencies at wide chunks, i.e. the time resolution is low and the frequency resolution is high. Vice versa at high f. This is ideal since short transient features are more likely to happen in the high-frequency region.
It is also important to talk about the quality factor Q. $Q = \frac{f_k}{\delta f_k}$ is the number of cycles processed at a center frequency f_k [13]. It determines the trade-off between frequency and time resolution. Lower values of Q mean smaller σ_t and better characterization of features with fast time evolution at the cost of a bad frequency resolution and vice versa. Actual implementations of this algorithm perform what is known as "omega scan", which consists of scanning a range of Q values and automatically choosing the value that provides the highest signal-to-noise ratio and so the best characterization of features [17].

3.2.3 Glitches images

Now, as I said in section 3.2.1, glitches are not perfectly centered, meaning that if we try to take a Q transform in a time interval centered in the middle of the time series, the glitch will result slightly off to the right or to the left in the final picture, and this problem is common for every glitch in my available dataset. Moreover, if we take the Q transform of the whole 16-second long time series (figure 3.9), we notice something else.

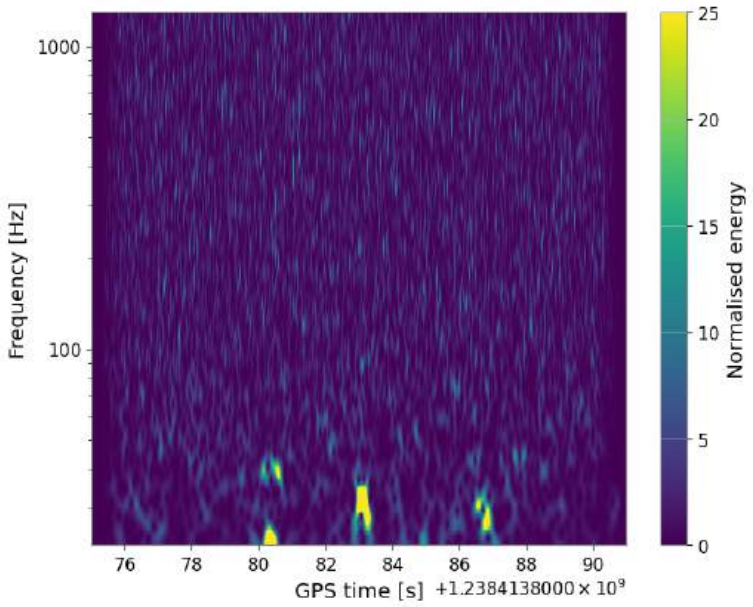


Figure 3.9: *Q transform of same time series of figure 3.2.*

We have a central, relatively large, "yellow shape", but also some other, smaller, shapes located near the central one. This happens because some glitches, in particular those of the class Scattered Light, have a periodic behavior, and different instances of the glitch can occur within the time window of 16 seconds [19]. Note that the algorithm that calculates the Q transform is heavy, especially for large time windows to process. Since one of the goals for future improvements of the glitch analysis that I am presenting would be to execute the whole process in low latency, it is not feasible to take 16-seconds-long Q transforms every time an event detection is triggered in the interferometer, so I decided to always ignore any glitch outside the central one in every time series. The way I center the central glitch in the proper time window is:

- First I take the Q transform of the time series in a one-second-long window around the central time, this will always include the central glitch:

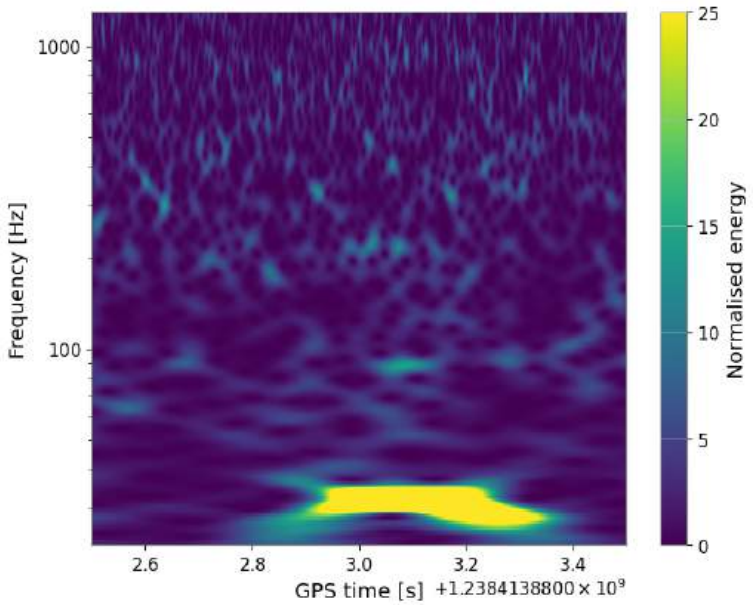


Figure 3.10: *Q transform of same time series of figure 3.2. 1 second long, centered on the time axis.*

- Then I calculate the time at which the maximum value of the normalized energy appears, and I perform another Q transform using this central time and a 0.5 seconds long time window, this will have the glitch well centered in the image:

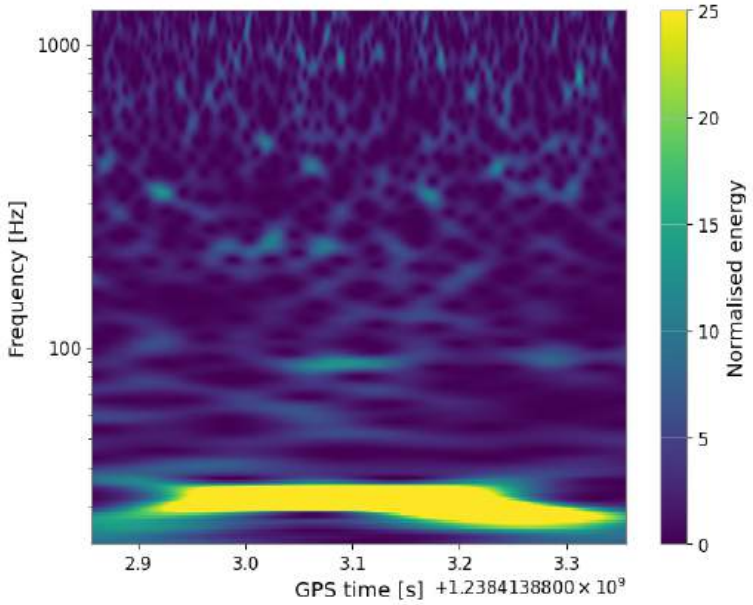


Figure 3.11: *Q transform of same time series of figure 3.2. 0.5 seconds long, centered on the time axis around the glitch.*

- The previous steps are applied for every glitch class. For Scattered Light and Low-Frequency Lines glitches I also select a limited range in frequencies, that is $[10\text{Hz}, 100\text{Hz}]$ because these glitches always happen within this frequency band, and doing so I have the glitch picture well centered also in the frequency axis:

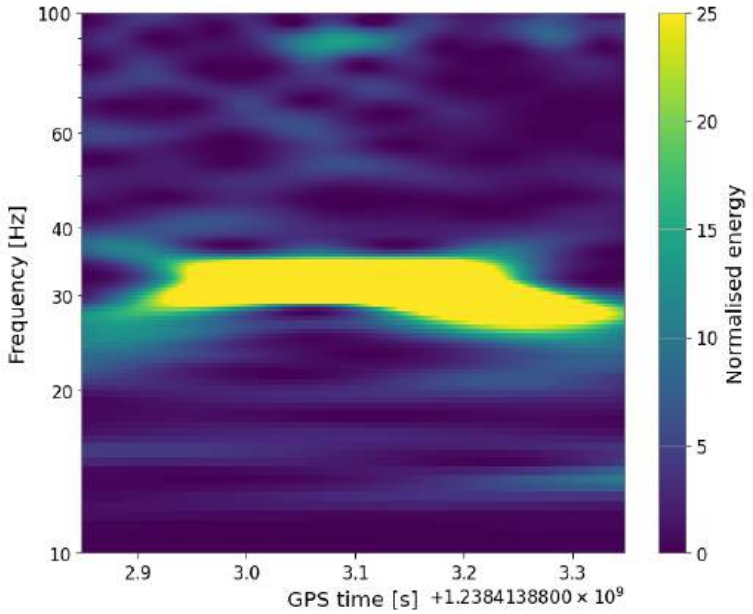
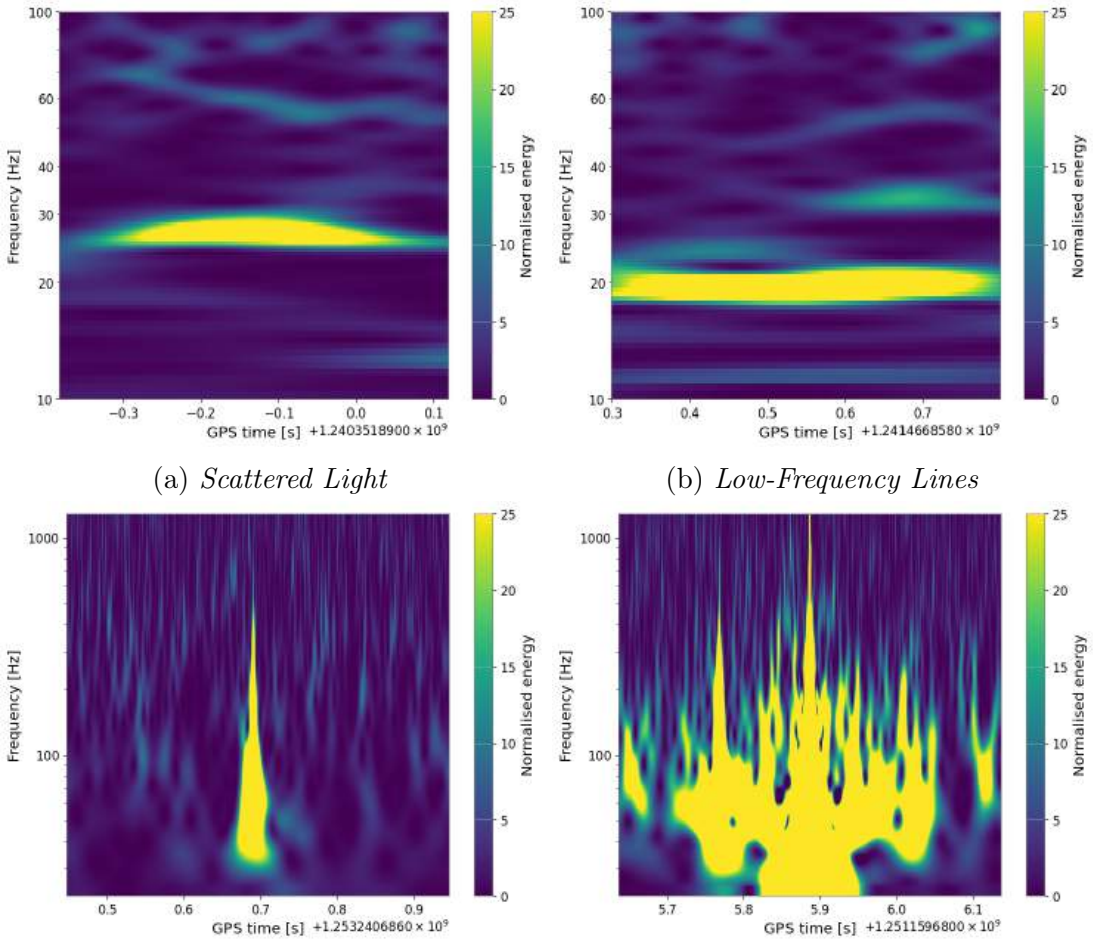


Figure 3.12: *Q transform of same time series of figure 3.2. 0.5 seconds long, centered on the time and frequency axis around the glitch.*

Figure 3.13 shows an example of a plot for every glitch class, created following the step described so far.



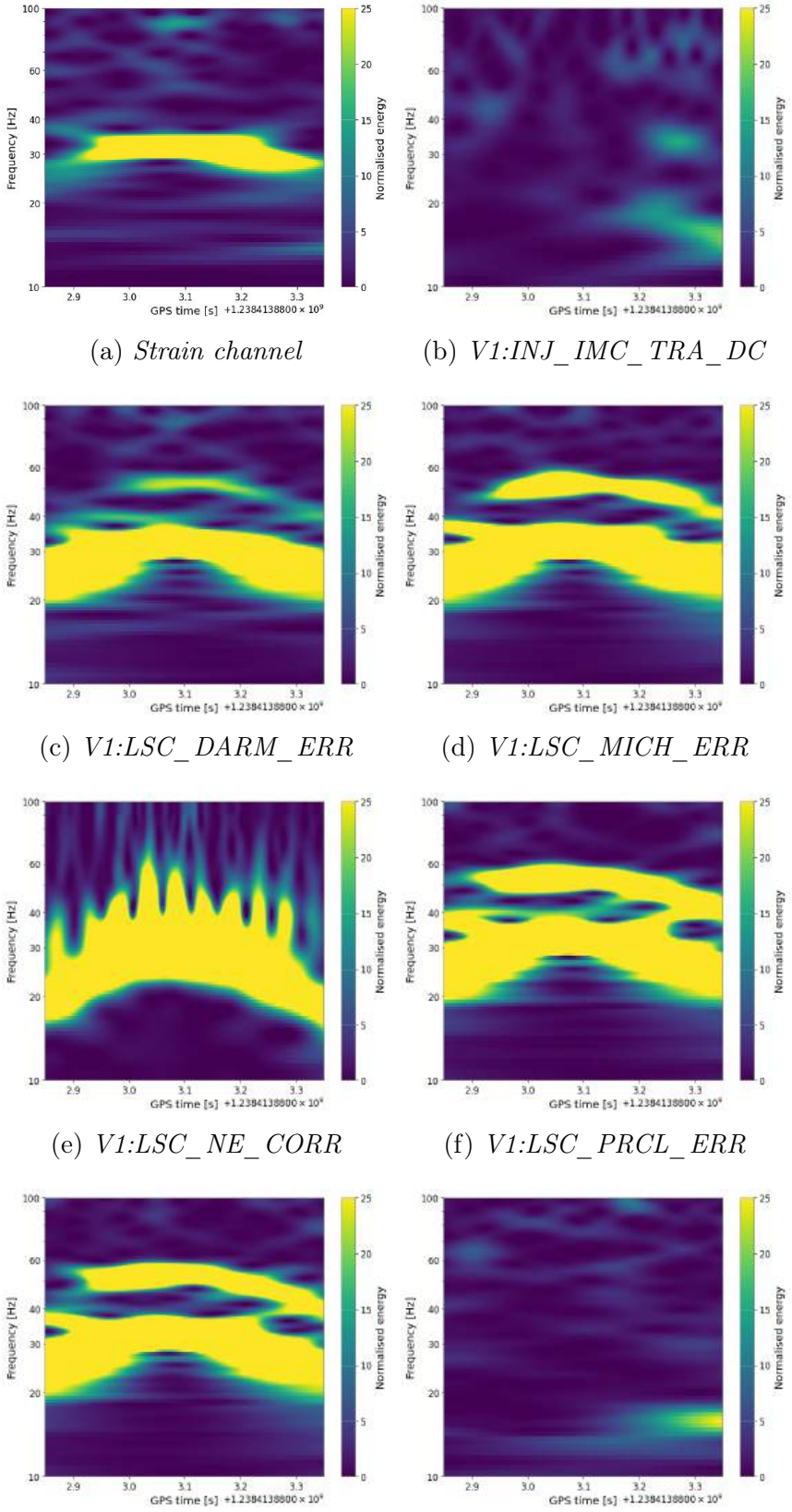
(a) *Scattered Light* (b) *Low-Frequency Lines*

(c) *Blip* (d) *Koi Fish*

Figure 3.13: *Example of Q transform plots for every glitch class available in my dataset.*

All the processing so far described must be applied not only to the strain channel but also to all the corresponding auxiliary channels, with the only difference that the glitch gets centered only in the strain channel and the central time calculated in that case gets used also for the Q transform of all the auxiliary channels. Taking for example the glitch of figure 3.12, the Q transform for all the channels is shown in figure 3.14.
Now I want to create pictures to be fed into a Generative Adversarial Network. What I do is to render the Q transform graphics into single-channel images of size 256×256 pixels. I use the python library "matplotlib" to perform the rendering, the lowest possible value of normalized energy, 0, is mapped to a pixel value of -0.76 and the highest value, 25, to 0.69[4]. The Q transforms from figure 3.14 turned into images and printed one after the other are shown in figure 3.15. The first step in creating the training dataset for the GANs was to create images like this for every glitch of every class in my available data.

[4]The range $[-0.76, 0.69]$ was automatically chosen by the rendering function in the matplotlib library. In a different implementation, I tried to force the rendering to map the normalized energy values $[0, 25]$ to pixel values $[-1, 1]$ but it turned out that this way the networks had a harder time during the training process, which was often more unstable, and the overall results were worse.



(a) *Strain channel* (b) *V1:INJ_IMC_TRA_DC*

(c) *V1:LSC_DARM_ERR* (d) *V1:LSC_MICH_ERR*

(e) *V1:LSC_NE_CORR* (f) *V1:LSC_PRCL_ERR*

(g) *V1:LSC_PR_CORR* (h) *V1:Sc_MC_MIR_Z_CORR*

Figure 3.14: *Q transforms for every auxiliary channel concerning a Scattered Light glitch.*

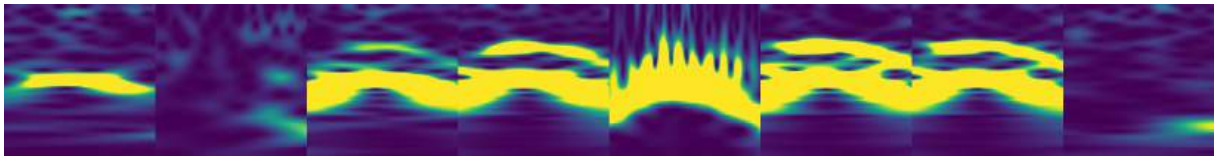
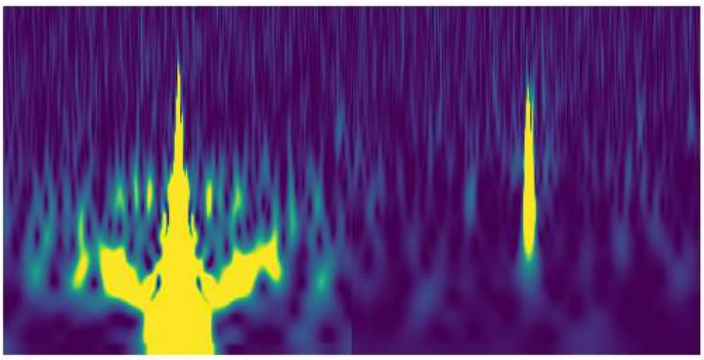


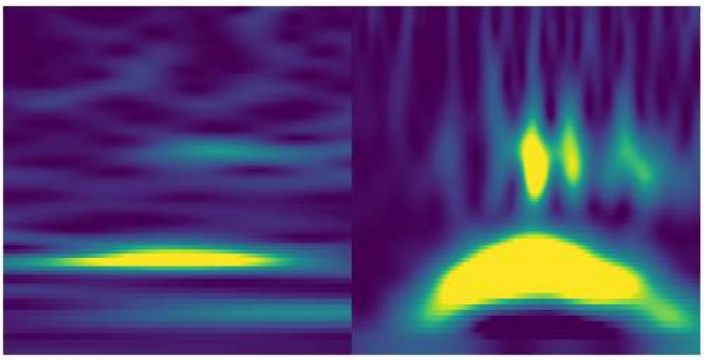
Figure 3.15: *Q transforms for every auxiliary channel concerning a Scattered Light glitch, turned into* 256×256 *images, printed one after the other.*

3.2.4 Training and testing dataset

As we discussed in section 2.4, Pix2Pix requires data to be paired while CycleGAN does not. To create both training and testing datasets for Pix2Pix we print the measured strain image with one of the auxiliary channel images, obtaining a picture like the ones shown in figure 3.16. This needs to be done separately for each combination of glitch class and auxiliary channel, the network will learn independently on every different combination. For CycleGAN, instead, we just need to create (for each combination of glitch class and auxiliary channel) two sub-folders, one containing all images of the strain channel and the other all images of the auxiliary channel.



(a) *Example for glitch class Koi Fish*



(b) *Example for glitch class Low-Frequency Lines*

Figure 3.16: *Example of training data for the Pix2Pix model. On the left, there is the true strain image, which is the target (also called "ground truth"). On the right there is, in this case, the respective image for auxiliary channel number 4, which is the starting image (also called "observed image").*

Having a relatively small-sized dataset, I decided to perform the train-test set split like so:

- Scattered Light: 100 test samples, 755 train samples;
- Low-Frequency Lines: 50 test samples, 285 train samples;
- Blip: 50 test samples, 320 train samples;
- Koi Fish: 50 test samples, 339 train samples.

As I will discuss in section 3.3, the main goal of the testing procedure is to estimate the average $L1$ distance between generated images and ground truths. 50 points should be enough for this purpose, and having fewer training data could more easily lead to overfitting by the models.

3.3 Training and testing

3.3.1 Model training on the Yoga cluster

Pix2Pix and CycleGAN were implemented on machine learning framework PyTorch 2.0.0, as described respectively in [35] and [36]. Pix2Pix's generator consists of a U-Net, and the discriminator is a PatchGAN (see section 2.4.1), both are trained using Adam optimizer, an improved version of the minibatch stochastic gradient descent [43], as well as both generators and both discriminators of CycleGAN. The learning rate chosen is $\eta = 0.0002$ and the minibatch size is 1.
All computing resources were provided by the Yoga cluster, hosted at the INFN Torino Computing Centre. The cluster comprises eight servers for a total of 240 cores, 2 TB of RAM, and 11.1 TB of disk. Moreover, some servers are equipped with Nvidia GPUs, for a total of six Tesla T4 and one A100. The servers are linked by a 1 Gbps Ethernet connection. Task scheduling across the cluster is managed by an orchestration layer (Kubernetes [45]), leveraging Docker [46] containers to define and isolate the runtime environment for several virtual clusters (one for each user). The virtual clusters are accessed through a web interface based on Jupyter Hub [47]. When a user authenticates on the Hub through GitHub OAuth [48], a personal notebook server is created as a containerized application. Particular server profiles can be chosen to have direct access to one of the available GPUs. For the training of the Pix2Pix model I used a Tesla T4, for the CycleGAN (more computationally costly) I used the A100.
A difference from the standard implementation is that, in my version of the training of Pix2Pix, for each update of the discriminator, the generator updates 5 times. This has been done because, in the first runs, I noticed the discriminator was much stronger than the generator, and its loss used to quickly fall to zero without ever rising again, slowing down the learning process of the generator. Ideally, during training we would like the generator loss to continuously decrease and the discriminator loss to approach a value of 0.5, which would mean that it is no longer able to discern real images from fake ones.
As I said in section 3.2.4 multiple instances of the models must be created: one of Pix2Pix and one of CycleGAN for each combination of auxiliary channel and glitch class. This leads to a total of 28 instances of Pix2Pix and as many of CycleGAN. To limit the

amount of computational time needed I decided to train each instance of the models for 100 epochs[5] at first, then prolong the training only when it seemed necessary.

3.3.2 Image quality control metric

The trend of generator and discriminator losses can give us a hint on whether the learning went well or not, but it is not necessarily related to the final quality of the generated pictures. In fact, having an objective criterion to evaluate the goodness of the generations is still quite an open and discussed topic [38] [39]. When it is possible we can rely on manual evaluation, i.e., let the researcher themselves evaluate the quality and diversity of the images produced by the generator. Similar approaches rely instead on citizen scientists, for example, we could ask people to decide which picture looks more realistic given a pair of images one of which is real and the other is generated. The model in this case obtains a score according to these decisions. Problems of these manual qualitative approaches are:

- Time consuming;
- Non-objective;
- Require knowledge of what is realistic and what is not for the target domain;
- Costly;
- Human performance is not constant and tends to improve over time.

Instead, we would like a quantitative approach, something that involves the calculation of a specific numerical score used to summarize the quality of generated images. As described in section 2.4, during training, Pix2Pix forces generated images to be close to the ground truth, and the similarity between real and fake images is imposed thanks to the $L1$ distance. Also in CycleGAN, to introduce the cycle consistency loss, we exploit the $L1$ distance between the original image and the cycled image. Therefore I decided to use the $L1$ distance as a base for my evaluation metric. To evaluate the performance of the generator I define the $L1$ error as the average over the testing set of the $L1$ distance between generated images and corresponding ground truths:

$$L1 \text{ error} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \|\hat{y}_i - y_i\|_1 \tag{3.6}$$

where N_{test} is the number of images in the testing set, y_i is the i-th ground truth and $\hat{y}_i$ is the i-th generated image. We should note that this metric is biased when describing the goodness of the generator since the $L1$ distance enters into the definition of the loss that we minimize during training. In section 4.1.3, I will discuss some of the problems that occur when using the $L1$ distance and explain what should be a more appropriate metric to evaluate the performances of the models (and why it is much more complicated to implement it). Nonetheless, in the next chapter, we will see how the $L1$ error works pretty well in describing the overall quality of generated images in the testing dataset, and so, it is a good metric for my purposes.

[5] A epoch refers to a complete cycle through the entire training dataset during model training.

3.3.3 Metric's trend

I was interested in understanding how the actual performances of the models were improving during training, so I decided to evaluate them after every epoch. For this purpose, after each epoch of training:

- Training stops;
- The so far trained generator is fed every observed image in the testing set;
- The $L1$ error is calculated, averaging the $L1$ distance between every pair of generated image and ground truth (equation 3.6);
- Training resumes with the next epoch.

This way I can keep track of the $L1$ error over the whole training process.
To understand what is the meaning behind a certain value of the $L1$ error consider that, as I said in section 3.2.3, pixel values for glitch images are in the range $[-0.76, 0.69]$, where -0.76 represent the minimum of normalized energy, while 0.69 represent the maximum. The length of this range is 1.45. So, if, e.g., the $L1$ distance between a ground truth and a generated image is 0.2, it means that the generated image is about $\frac{0.2}{1.45} \approx 14\%$ off from the ground truth. Hence, if the $L1$ error of the generator is 0.2, it means that, on average, generated images in the testing set are 14% off from their ground truths.

Chapter 4

Results

In this chapter, I will describe the results obtained with the abovementioned methodology. For brevity's sake and to avoid repetitions, I will limit the discussion mostly to glitch class scattered light and to auxiliary channels number 2: V1:LSC_DARM_ERR; and 5: V1:LSC_PRCL_ERR.
The choice of the scattered light class over the others is not only because it is the one with the most samples (about double that of every other class), allowing the model to train better with less risk of overfitting, but also because the seven available auxiliary channels are likely to be correlated to scattered light events[1].
The 2nd and 5th auxiliary channels are interesting because we would like to see the performance of the models both when the auxiliary channel is correlated to the strain one and when it is not. In section 3.1 we have seen that auxiliary channel 2 monitors the deviation in the differential arm length measurement, that is the difference in length between the two interferometer's arms: apart from some correction signals this is the strain itself, meaning that this channel will be highly correlated to the strain one. Auxiliary channel number 5 monitors instead the deviation in length of the power recycling cavity, these kinds of deviations are among those that can cause the laser light to scatter off its original path, meaning that this channel is likely to be correlated to scattered light glitches while not being directly correlated to the strain channel. Note that from now on, I will refer to auxiliary channels with their number: the correlated auxiliary channel will be "auxiliary channel 2", and the uncorrelated one will be "auxiliary channel 5".
In this discussion, I will focus more on the results obtained with Pix2Pix (section 4.1) since they were much more convincing than what I could achieve with CycleGAN, of which I will talk about in section 4.2.

4.1 Pix2Pix

Pix2Pix requires training images to be paired, as we discussed in section 3.2.4, it implements a single generator, which I will refer to as G, that is supposed to learn the mapping from a particular auxiliary channel to the strain channel, and a discriminator D that aims to distinguish generated (fake) strain images from true ones. I will discuss the results obtained on scattered light glitches, first on auxiliary channel 2 (section 4.1.1), then on auxiliary channel 5 (section 4.1.2). In section 4.1.3, I will talk about other auxiliary channels and glitch classes, and see what we can learn from them.

[1]This is because, originally, the dataset included only the scattered light class, and the auxiliary channels were chosen accordingly among the ones that are more likely to pick a signal when such an event occurred. Later we were provided with data from the other cited glitch classes, but the same auxiliary channels are generally less correlated to these other types of events.

4.1.1 Scattered Light - Auxiliary Channel 2 (Correlated)

Figure 4.1 shows the trend of generator and discriminator loss vs. epoch. These are what we would expect to see in the case of good-performing training: the generator loss is decreasing uniformly and the discriminator loss has a pretty smooth trend that never falls to 0 nor rises to 1. This results in the fact that the generator can learn quickly,

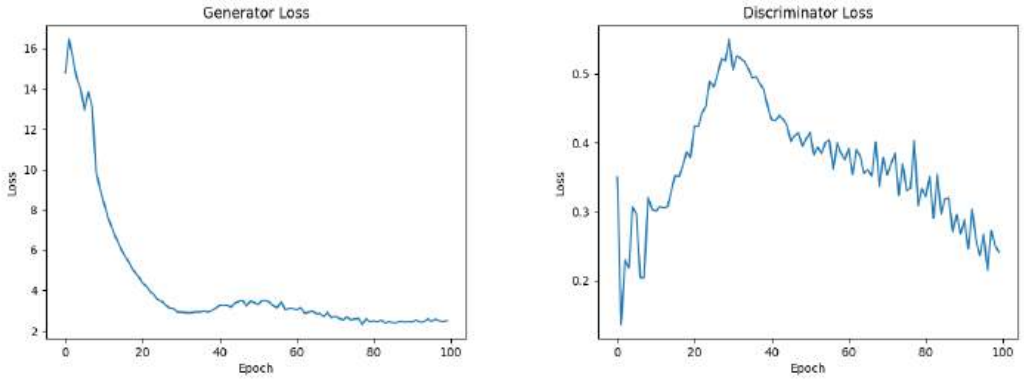


Figure 4.1: *Generator and discriminator loss vs epoch for Pix2Pix (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

as we can see from the $L1$ error trend in figure 4.2. In particular, we can notice that

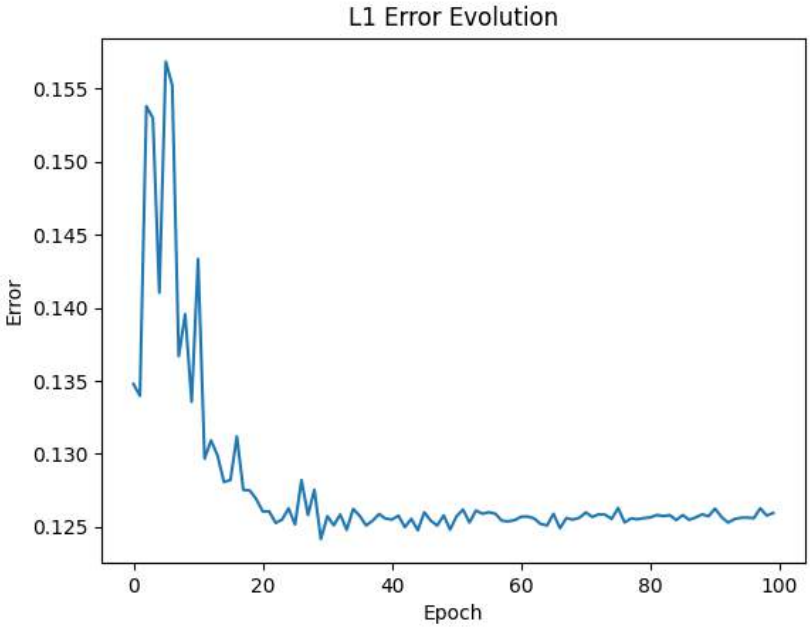
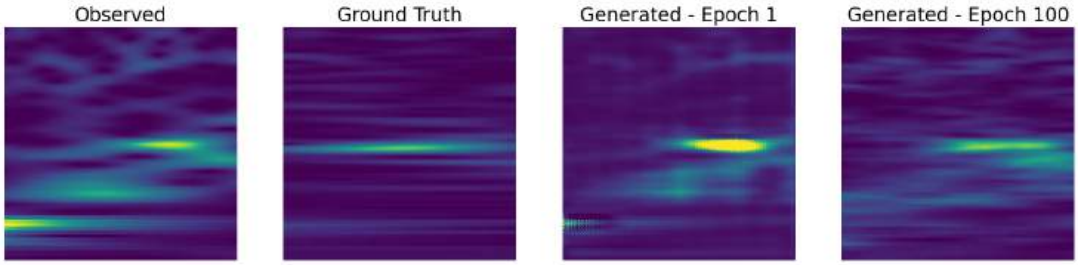
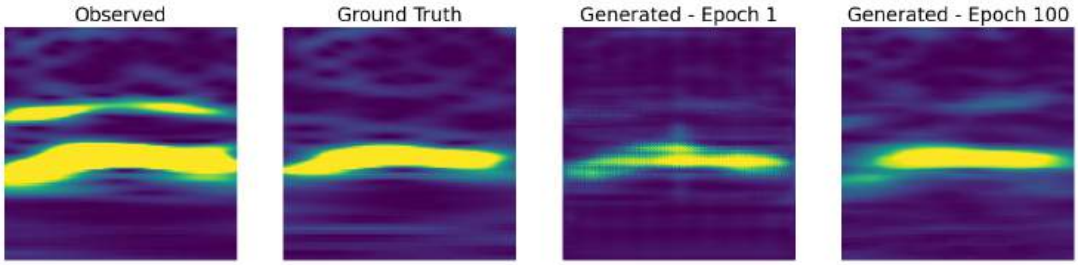


Figure 4.2: *$L1$ error vs epoch for Pix2Pix (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

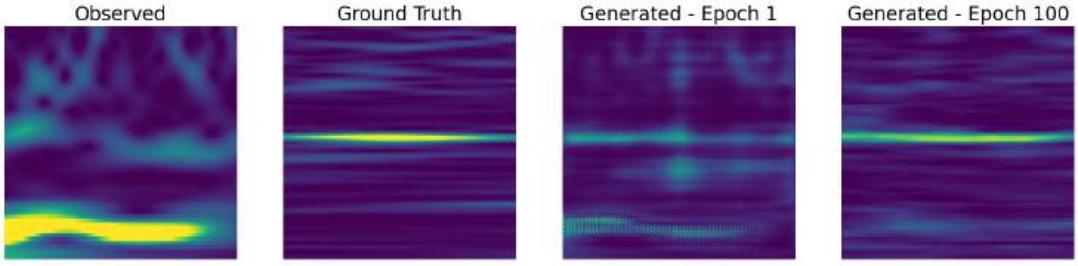
the $L1$ error drops and stabilizes after about 20 epochs at ≈ 0.125, implying an average distance of 9% between generated images and ground truths in the testing set. Also, in the very first epochs, the value of the $L1$ error is a little above this minimum: notice for instance that at epochs number 1 and 2 it is about 0.135. Figure 4.3 shows examples of generations after 1 epoch and after 100 epochs of training. What we realize is that



(a) *Example 1*



(b) *Example 2*



(c) *Example 3*

Figure 4.3: *Generation examples for Pix2Pix at epoch 1 and epoch 100 (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

from the very first epochs, the generator can produce pretty convincing images, probably staying close to the identity at first, realizing that it is the easier way to get a sufficiently low loss. Notice for instance that the generated images at epoch 1 still show the remains of the shape that is present in the observed image. This is particularly noticeable in figure 4.3c, wherein the bottom part of the generated image at epoch 1, we can see a wavy shape about to be removed. At epoch 100 not only the shapes of glitches but also the backgrounds are more well defined and precise. The generator, indeed, is motivated to look for a better mapping than the identity and improve its performance. After some failed attempts that cause the $L1$ error to slightly increase up to 0.155, it can find a good combination of weights that produces a better mapping than what it started with and lowers the metric to 0.125. This behavior of the generator, consisting of an initial state where it has a relatively low loss, then a first learning phase where it explores different strategies allowing the metric to increase and change abruptly from epoch to epoch, to then stabilize in a last phase after about 20 epochs, can be observed in a large number of instances of Pix2Pix.

$L1$ distance distribution

Now I will show how the $L1$ distance between the ground truth and the generated image is correlated with the actual quality of the generation.
Figure 4.4 shows the distribution of the $L1$ distance between ground truths and corresponding Pix2Pix generated images, calculated on every sample of the testing set. Notice that the $L1$ error with which I evaluate the generator is nothing but the mean of this distribution.

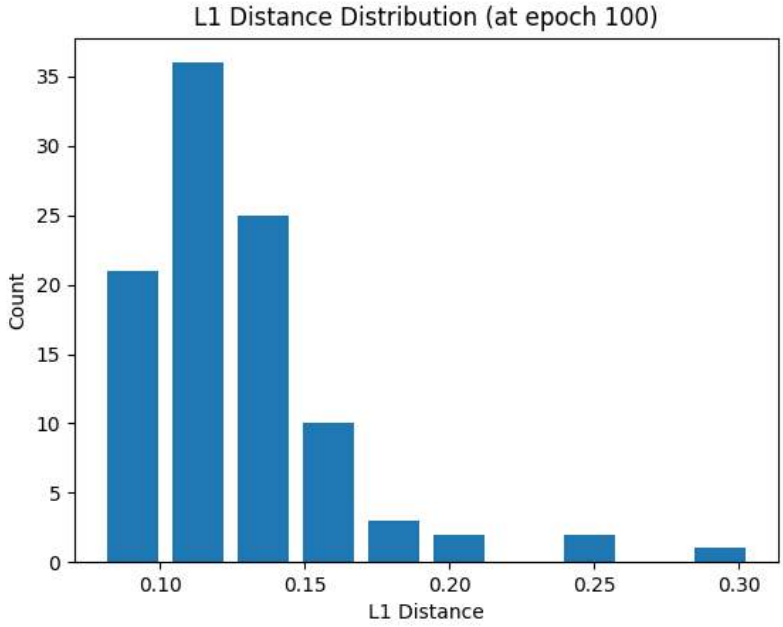


Figure 4.4: *Distribution of $L1$ distance between Pix2Pix generated images and ground truths took from the testing set, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2). The 95-th percentile is $P_{95} = 18.8$, while $P_{80} = 14.4$.*

Figure 4.6 shows examples of generations with various values of $L1$ distance. When the $L1$ distance reaches a value above 0.20, the generations are usually bad, but in this case, most of the bad generations seem to be due to a presumable misplacement of non-scattered-light glitches in the Scattered Light class, e.g., the shape that we see in the ground truth in figure 4.6d does not resemble a scattered light glitch, and indeed, if we look at the confidence metadata for the scattered light glitches dataset (figure 4.5) we see that there are a good amount of samples for which the confidence of classification is pretty low.

When looking at the examples of figure 4.6, if we keep in mind the $L1$ distance distribution in figure 4.4, and that its 95-th percentile is $P_{95} = 18.8$, we can conclude that the overall quality of the generations is good.

Glitch placement error

The shape of scattered light glitches in the strain channel is basically a yellow line placed at a certain height, the height being a fundamental property since it is related to the frequency content of the glitch itself. We want to ensure that the network understands the importance of placing the line at the proper frequency and is not just drawing random

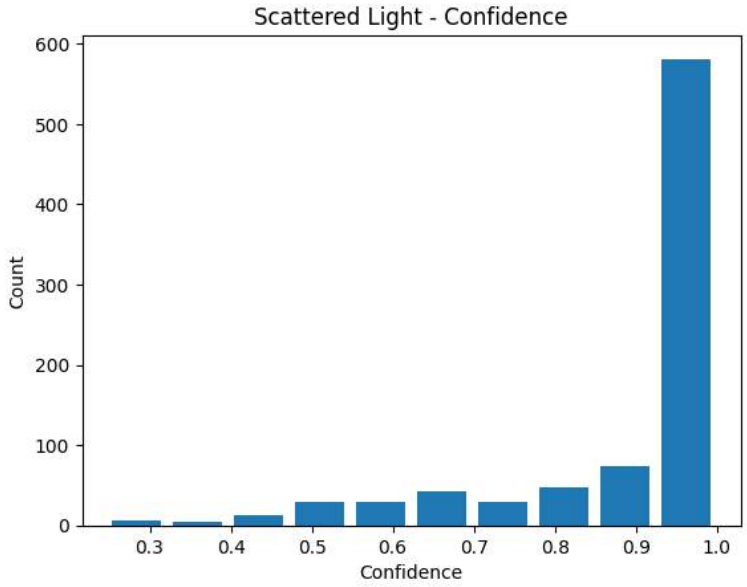
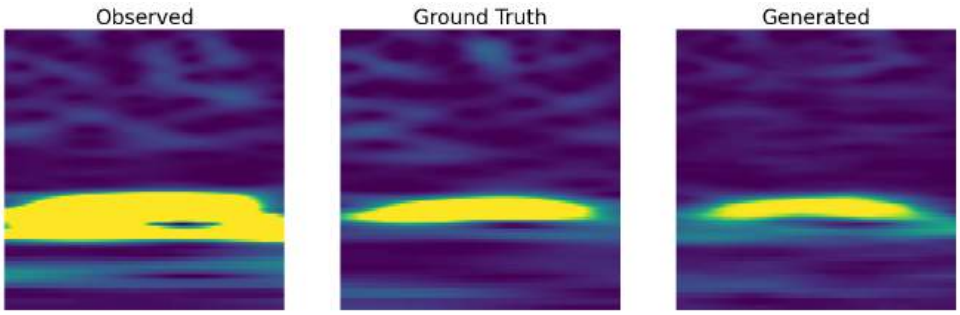
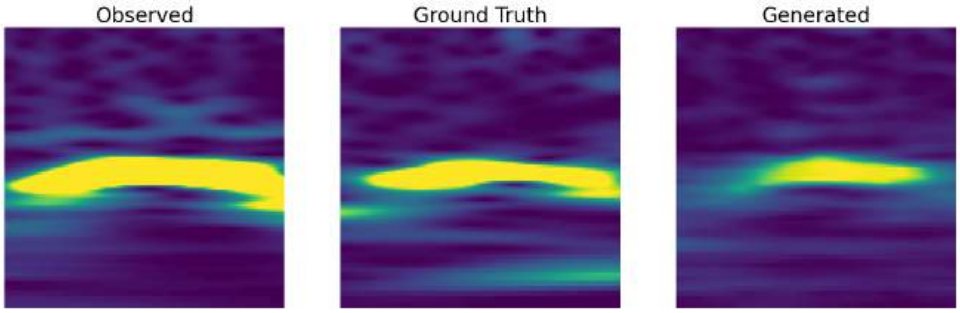


Figure 4.5: *Scattered Light glitches; confidence of classification.*

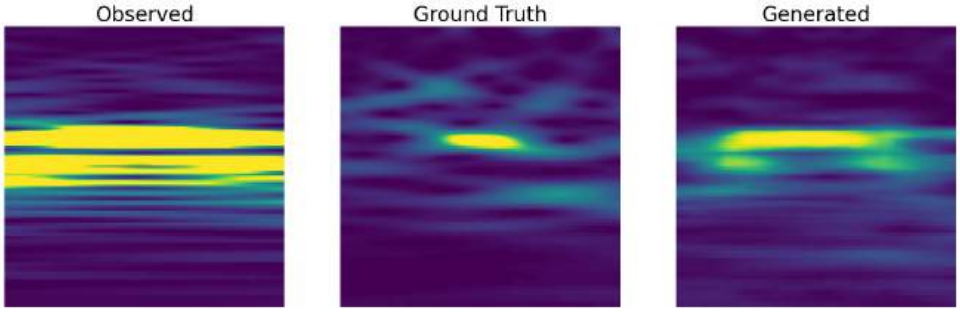
yellow shapes on blue canvases. I estimated the frequency of the glitch in an image as the mean of the height of the highest pixel value in each pixel column, and once I had the height in terms of pixels I mapped that value into the frequency domain.
This estimated frequency was calculated for the whole testing dataset for both ground truths and generated images. Figure 4.7 shows the two distributions of the height of glitches for ground truths and generated images. On a qualitative level, we can say that the two distributions look pretty similar, which means that the network can generate glitches in a proper range of frequencies and does not always place them in the same position. More quantitatively, the mean value for the ground truths distribution is $\mu_{f-true} = 38.1$ Hz and its standard deviation is $\sigma_{f-true} = 8.2$ Hz; for the generated images distribution, these values are $\mu_{f-gen} = 39.1$ Hz and $\sigma_{f-gen} = 7.9$ Hz. Moreover, figure 4.8 shows the distribution of the errors in the placement of glitches by Pix2Pix, the mean value is 5.2 Hz and the 90-th percentile is $P_{90} = 13.2$ Hz. From these values, we can conclude that the model is very precise in positioning the glitches at the right frequencies.



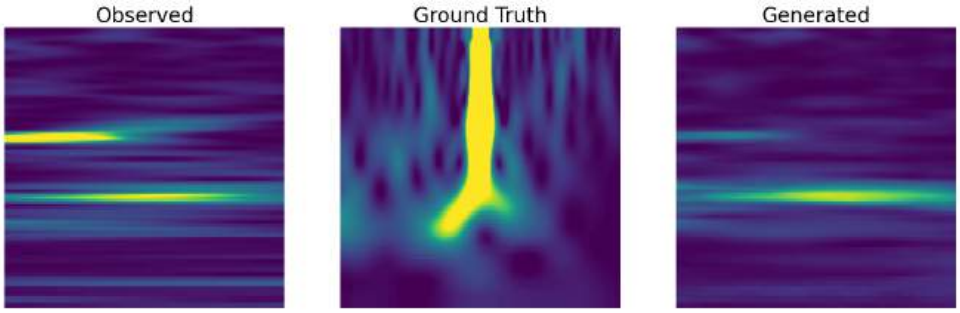
(a) *Example 1:* $L1$ *distance* $= 0.098$



(b) *Example 2:* $L1$ *distance* $= 0.146$



(c) *Example 3:* $L1$ *distance* $= 0.178$



(d) *Example 4:* $L1$ *distance* $= 0.243$

Figure 4.6: *Generation examples for Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

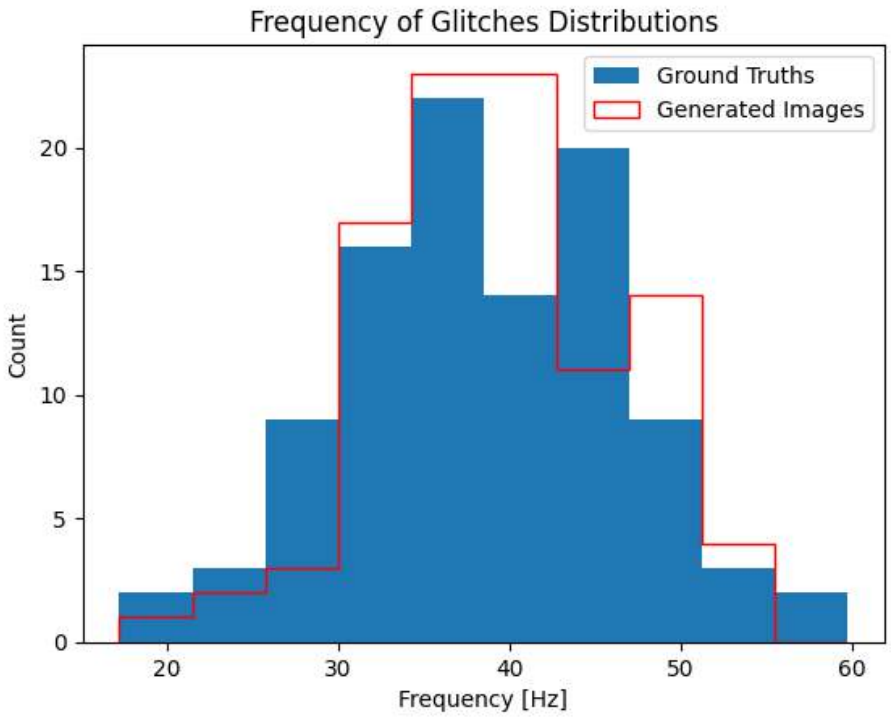


Figure 4.7: *Distributions of the frequency of glitches in the testing set for ground truths and Pix2Pix generated images, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2). The ground truths distribution has mean value* $\mu_{f-true} = 38.1$ *Hz and standard deviation* $\sigma_{f-true} = 8.2$ *Hz, for the generated images distribution they are* $\mu_{f-gen} = 39.1$ *Hz and* $\sigma_{f-gen} = 7.9$ *Hz.*

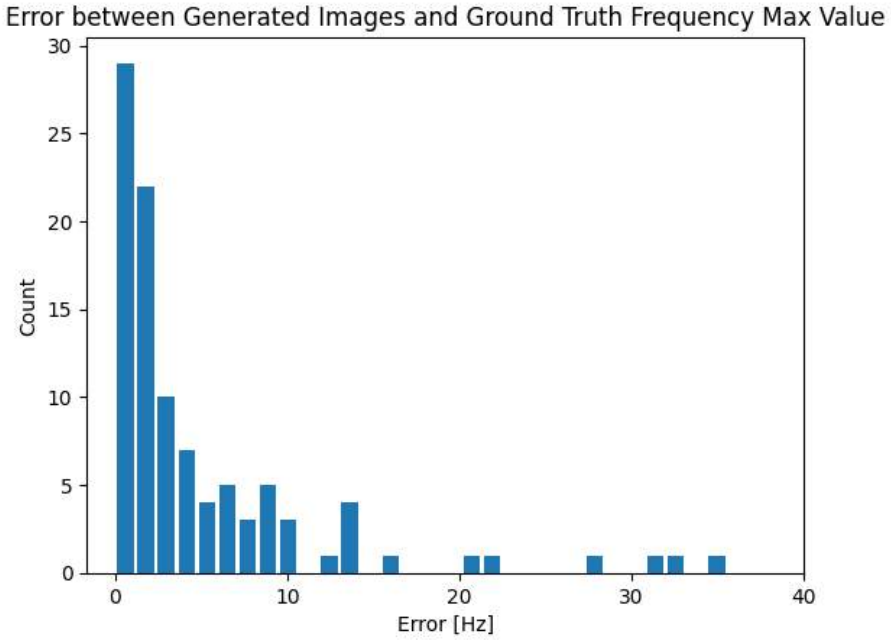
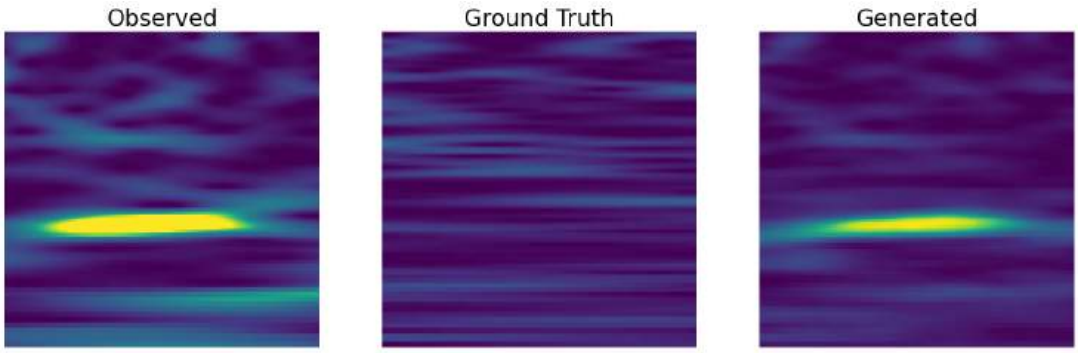


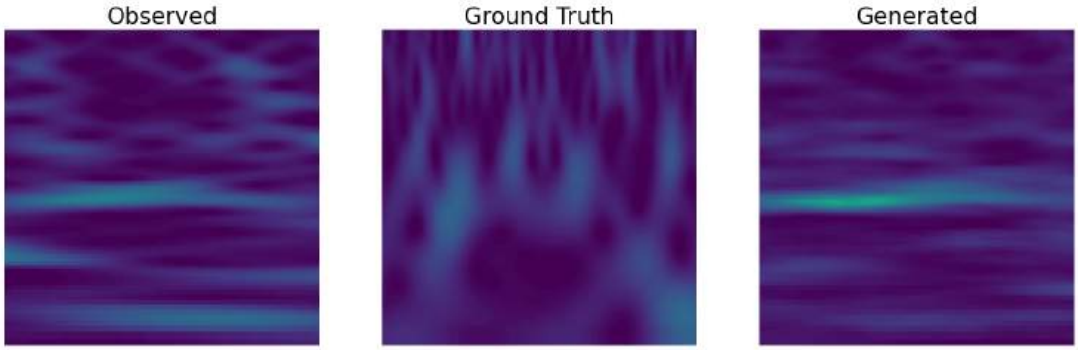
Figure 4.8: *Distribution of the error in the placement of glitches by Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2). The mean value is 5.2 Hz and the 90-th percentile is* $P_{90} = 13.2$ *Hz.*

No-glitch generations

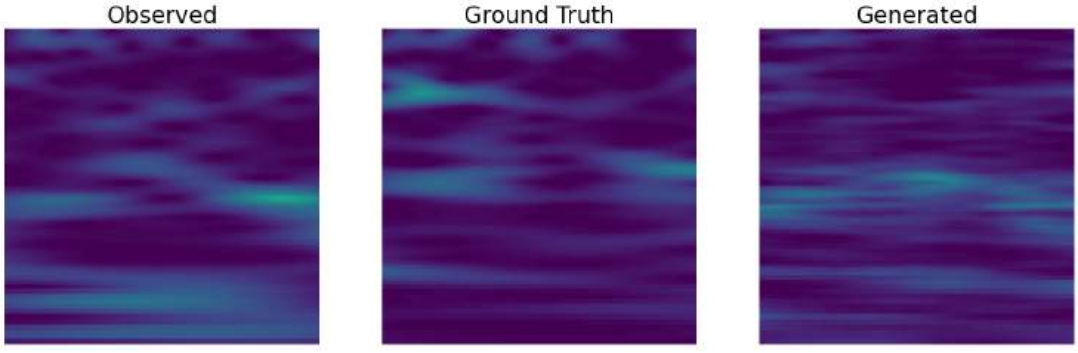
Since the time series in the glitch dataset are 16 seconds long (see chapter 3), we can easily create additional testing samples taken from areas where no glitch is present in the strain channel and see how the network behaves with them. Note that in the training set, there are no samples without glitch in the strain channel.
Some examples of these generations are shown in figure 4.9 and the results are what I expected: when the glitch is not visible both in the ground truth and in the observed image, the generated image also does not show any glitch (figure 4.9b and 4.9c); but when the glitch is visible in the observed image, the network does not remove it and it is still present in the generated image (figure 4.9a). We can conclude that it is probably necessary to provide a good amount of no-glitch images in the training set if we want the network to be reliable in the no-glitch scenario.



(a) *Example 1*



(b) *Example 2*



(c) *Example 3*

Figure 4.9: *Generation examples for Pix2Pix after 100 epochs of training, for testing images where the ground truth does not show a glitch.*

4.1.2 Scattered Light - Auxiliary Channel 5 (Uncorrelated)

I will start once again by showing the trend of the generator and discriminator loss vs. epoch (figure 4.10). We can see that it is a bit more unstable than what we had with

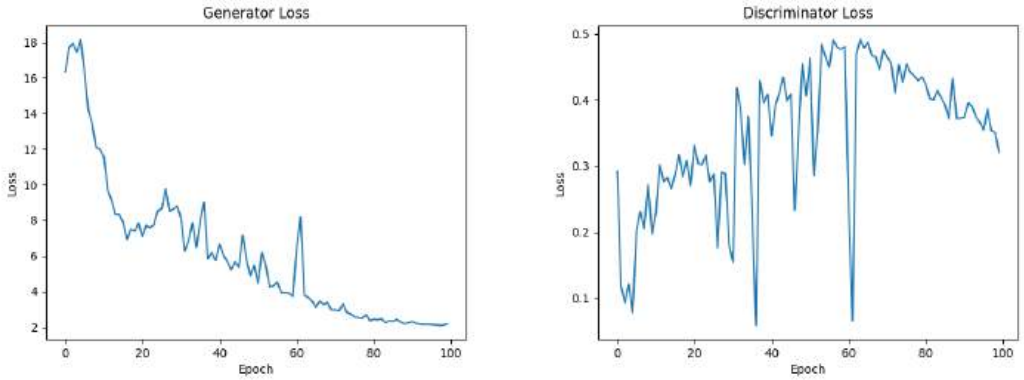


Figure 4.10: *Generator and discriminator loss vs epoch for Pix2Pix (glitch class = scattered light; auxiliary channel = auxiliary channel 5).*

auxiliary channel 2 (see figure 4.1). The $L1$ error evolution is shown in figure 4.11, in the first epochs it floats from values around 0.150 to 0.165, and then stabilizes at about 0.147 after 60 epochs, following a similar trend to what we discussed in the previous section.

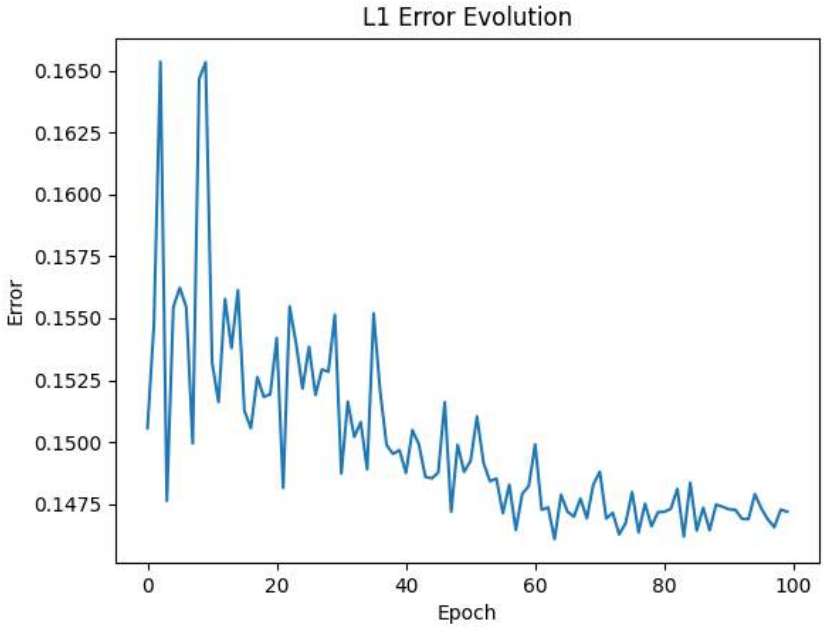


Figure 4.11: *L1 error vs epoch for Pix2Pix (glitch class = scattered light; auxiliary channel = auxiliary channel 5).*

The $L1$ distance distribution after 100 epochs of training is shown in figure 4.12. The 95-th percentile is $P_{95} = 22.9$, meaning that, this time, we have a relevant amount of bad generations. The 80-th percentile is $P_{80} = 18.2$. As before, I will show some examples of generations taken from various points of this distribution.
Figure 4.13 shows examples of generations with different values of $L1$ distance between

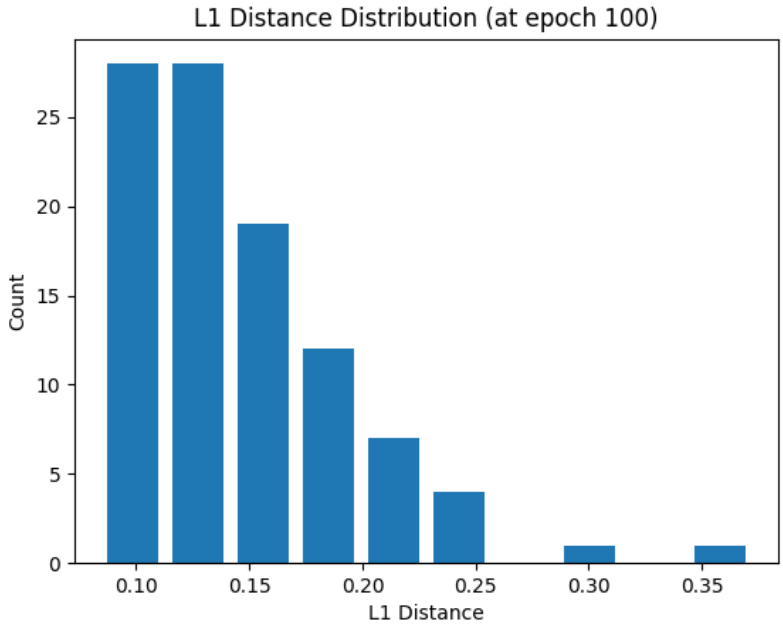
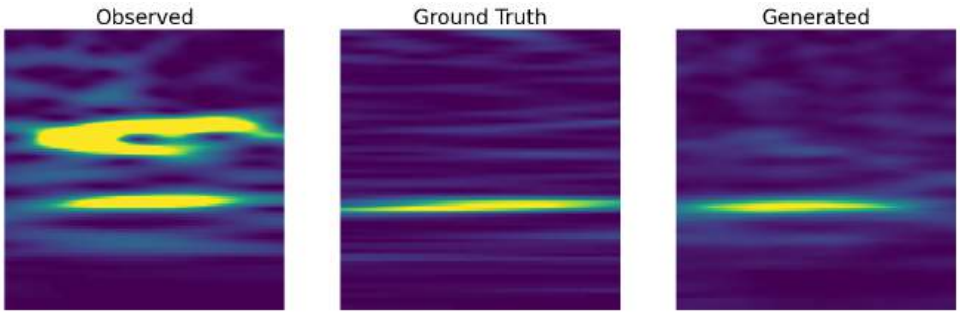
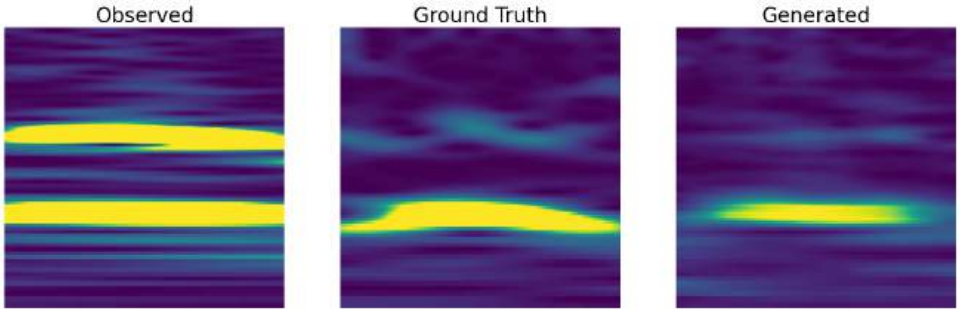


Figure 4.12: *Distribution of L1 distance between Pix2Pix generated images and ground truths taken from the testing set, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 5). The 95-th percentile is* $P_{95} = 22.9$*, while* $P_{80} = 18.2$*.*

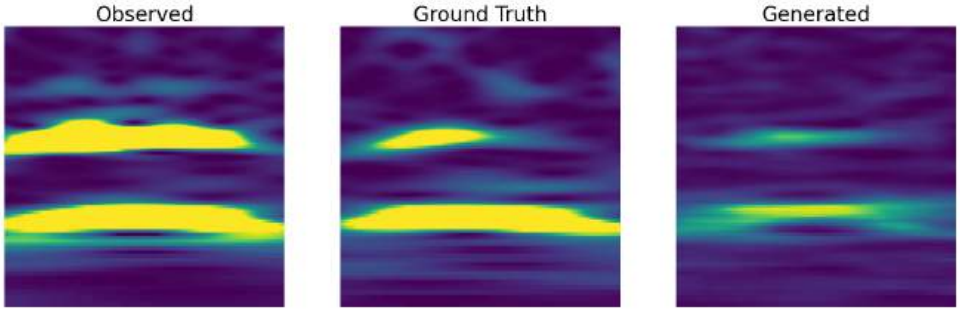
the generated image and the ground truth. However, as I said in the introduction of this chapter, auxiliary channel 5 might be correlated to scattered light glitches but is not directly correlated to the strain channel, this results (as we can see in figure 4.13d) in the fact that we have a lot of samples where the observed image has no information on what the ground truth should look like because, for instance, the glitch is not visible in the auxiliary channel but is only present in the strain one. It is not a problem if the model fails to generate glitches for which the ground truth is not correlated to the observed image, what I am interested in, is whether the model can find a good mapping for those glitches that are correlated to this particular auxiliary channel. To understand this, I calculated the glitch placement error as we saw in the previous section. Figure 4.14 shows the distribution of the frequencies of glitches for ground truths and generated images; the mean value for the generated images distribution is $\mu_{f-gen} = 39.1$ Hz, and its standard deviation is $\sigma_{f-gen} = 5.6$ Hz (remember that for the ground truths distribution we had $\mu_{f-true} = 38.1$ Hz, $\sigma_{f-true} = 8.2$ Hz). In figure 4.15 we can see the glitch placement error distribution; the mean value, in this case, is 6.0 Hz, and the 90-th percentile is $P_{90} = 14.2$ Hz, just slightly higher than what we had for auxiliary channel 2, indicating that the network is mostly able to understand the proper mapping from glitches in auxiliary channel 5 to glitches in the strain channel, at least in the case where the glitch is actually related to this particular auxiliary channel.



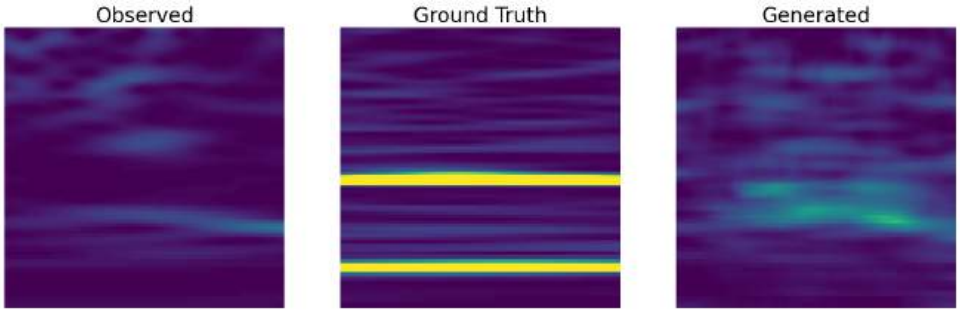
(a) *Example 1:* $L1$ *distance* $= 0.089$



(b) *Example 2:* $L1$ *distance* $= 0.128$



(c) *Example 3:* $L1$ *distance* $= 0.157$



(d) *Example 4:* $L1$ *distance* $= 0.230$

Figure 4.13: *Generation examples for Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 5).*

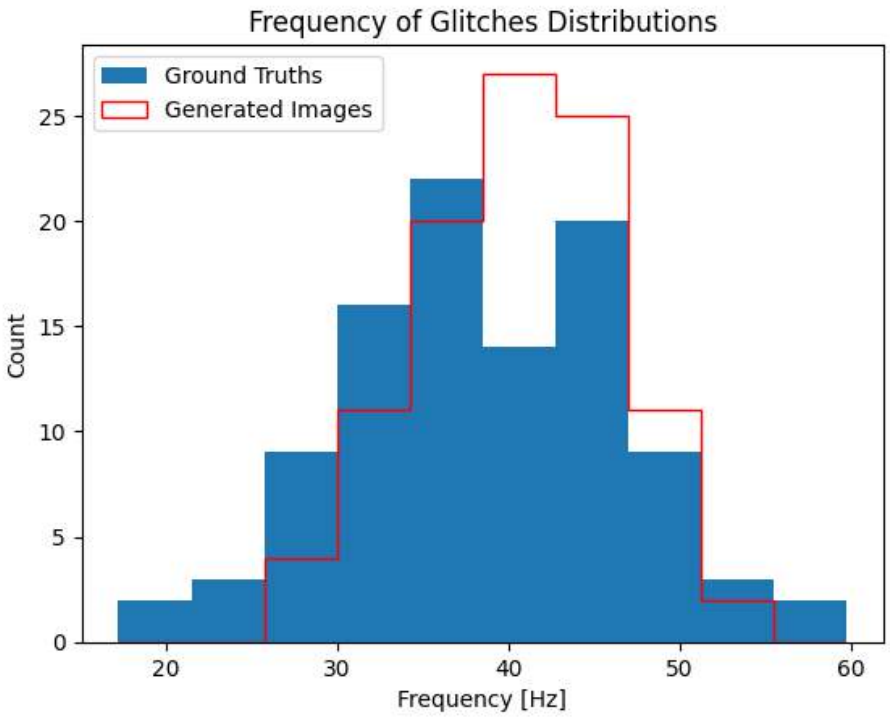


Figure 4.14: *Distributions of the frequency of glitches in the testing set for ground truths and Pix2Pix generated images, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 5). The ground truths distribution has mean value* $\mu_{f-true} = 38.1$ *Hz and standard deviation* $\sigma_{f-true} = 8.2$ *Hz, for the generated images distribution they are* $\mu_{f-gen} = 39.1$ *Hz and* $\sigma_{f-gen} = 5.6$ *Hz.*

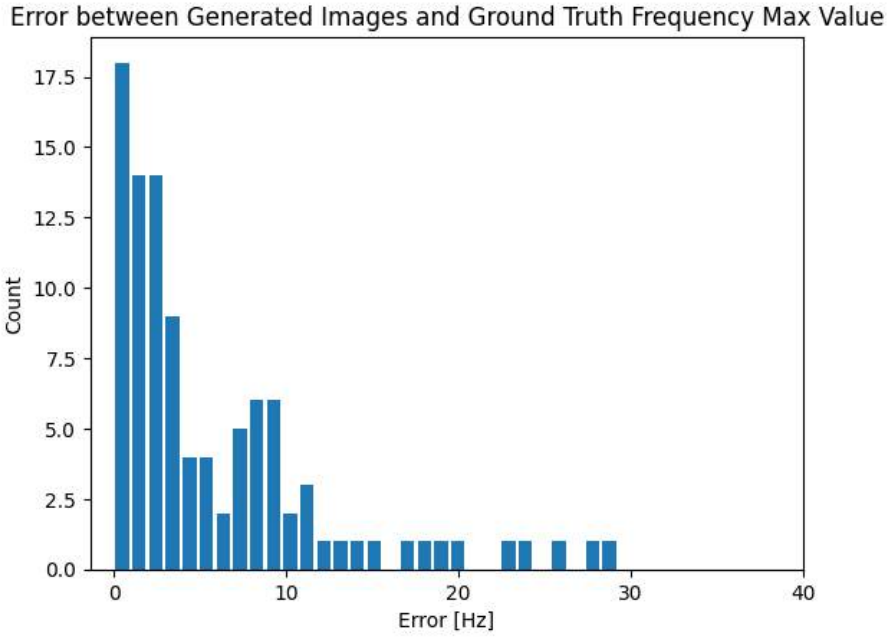


Figure 4.15: *Distribution of the error in the placement of glitches by Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2). The mean value is 6.0 Hz and the 90-th percentile is* $P_{90} = 14.2$ *Hz.*

4.1.3 Other auxiliary channels and glitch classes

Still focusing on scattered light glitches, the same behavior described at the end of the previous section is common to almost every auxiliary channel, for instance, auxiliary channel 3: figure 4.16 shows the $L1$ distance distribution. Some of the bad generations

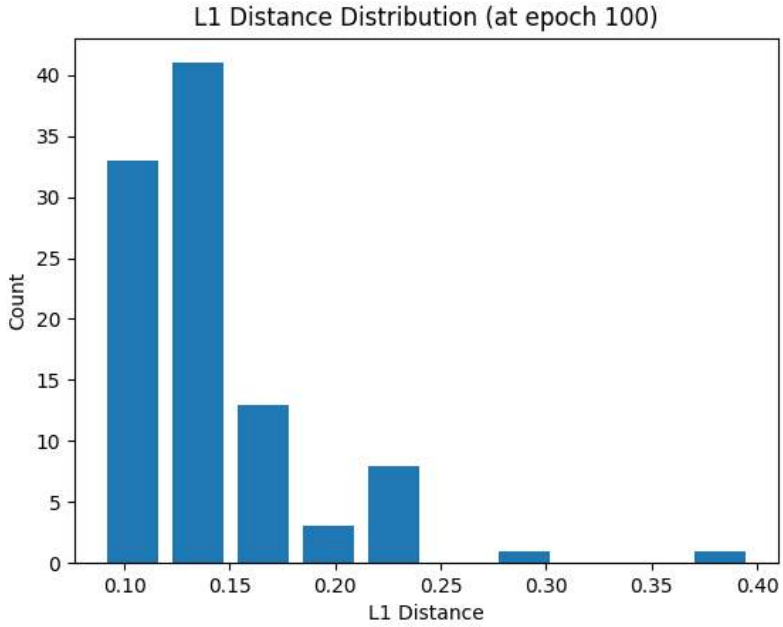
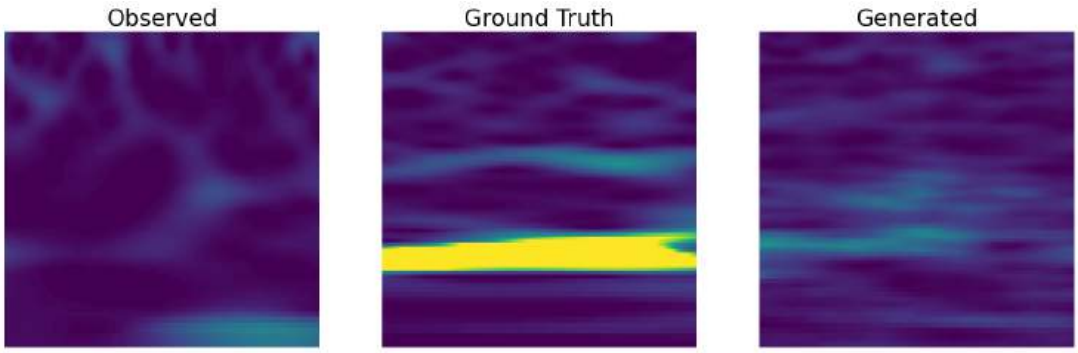


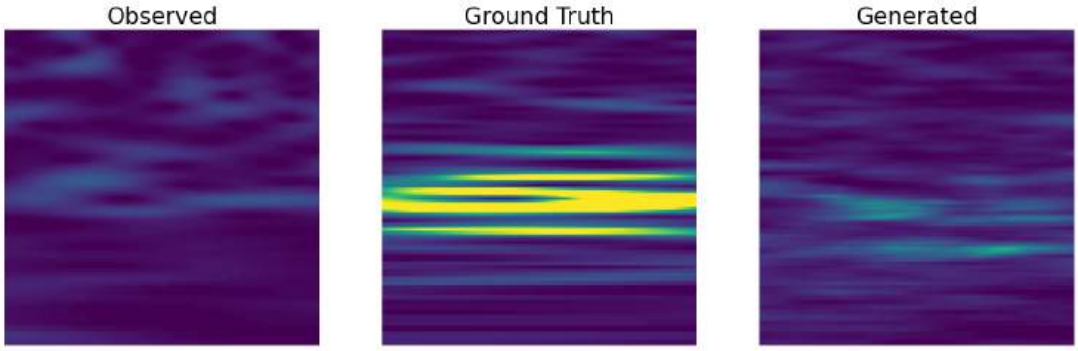
Figure 4.16: *Distribution of $L1$ distance between Pix2Pix generated images and ground truths took from the testing set, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 3).*

(with $L1$ distance above 0.2) are shown in figure 4.17 and as usual, they are either due to a glitch of another class misplaced in the scattered light dataset or to a non-existent correlation between the observed image (which is blank) and the ground truth. Some examples of good Pix2Pix generations for this auxiliary channel are shown in figure 4.18.

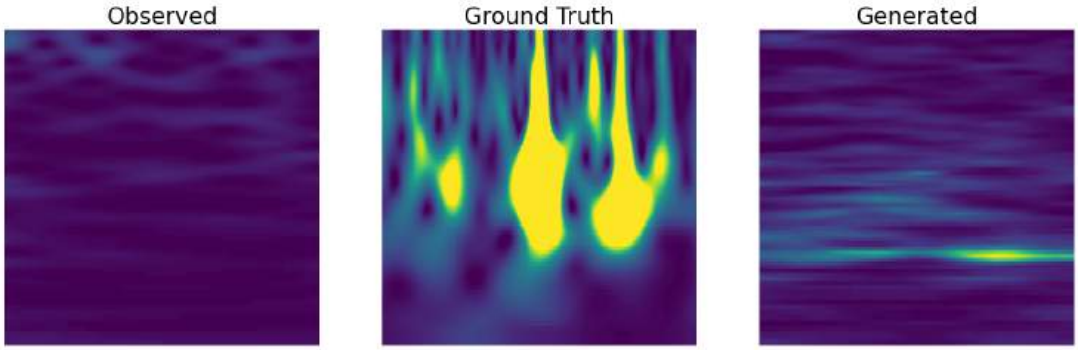
Nonetheless, there are some more problematic channels, for instance, auxiliary channel 7, that is totally uncorrelated to scattered light glitches and for which almost every observed image is blank. However, if we look at the $L1$ distance distribution (figure 4.19) we see something unexpected: there is a large number of testing samples for which the $L1$ distance is below 0.2, and indeed, the mean of the distribution is about 0.16, just slightly higher than what we have for other, more correlated, auxiliary channels. Some generation examples with different values of $L1$ distance are shown in figure 4.20
What we realize is that the $L1$ distance between the generated image and the ground truth, and hence, the $L1$ error, other than being biased (as I discussed in section 3.3.2) is a flawed metric, because placing a glitch at the wrong height will always produce a higher $L1$ distance than not placing the glitch at all. Moreover, the glitch shape usually takes up a relatively small portion of the image, so, the $L1$ distance between an image with no glitch and one that shows a tiny glitch shape will be small. Indeed, as we see in figure 4.20, the value of the $L1$ distance depends essentially on the size of the glitch in the ground truth image.
To overcome this problem we should find a better metric to evaluate the goodness of the



(a) *Example 1:* $L1$ *distance* $= 0.223$

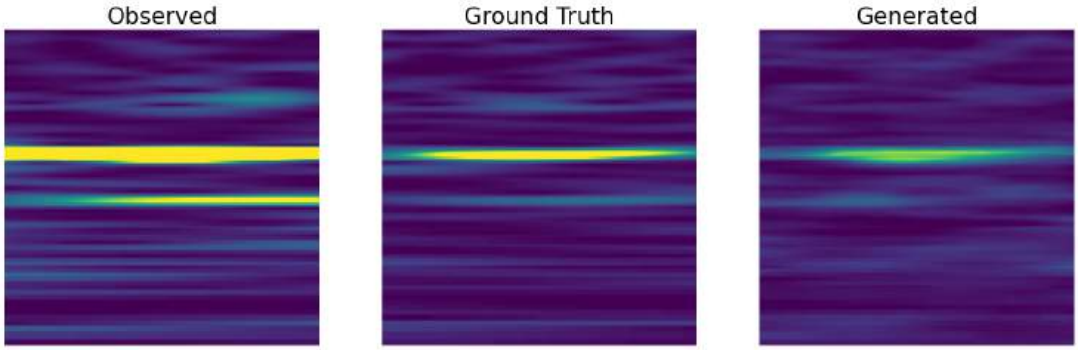


(b) *Example 2:* $L1$ *distance* $= 0.231$

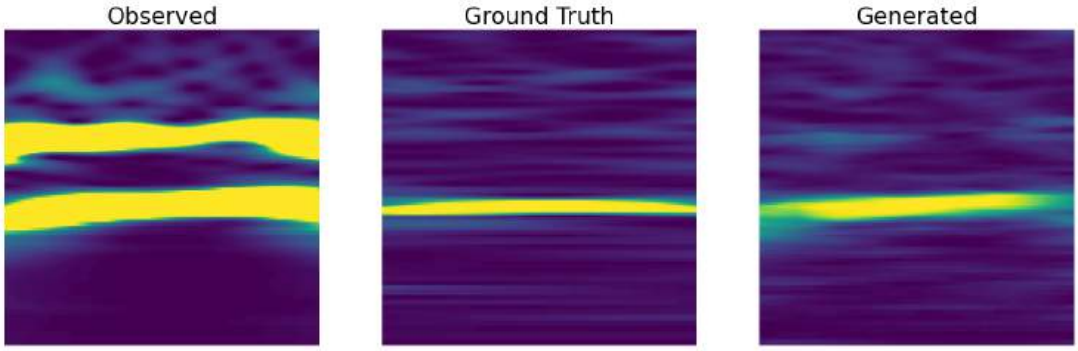


(c) *Example 3:* $L1$ *distance* $= 0.398$

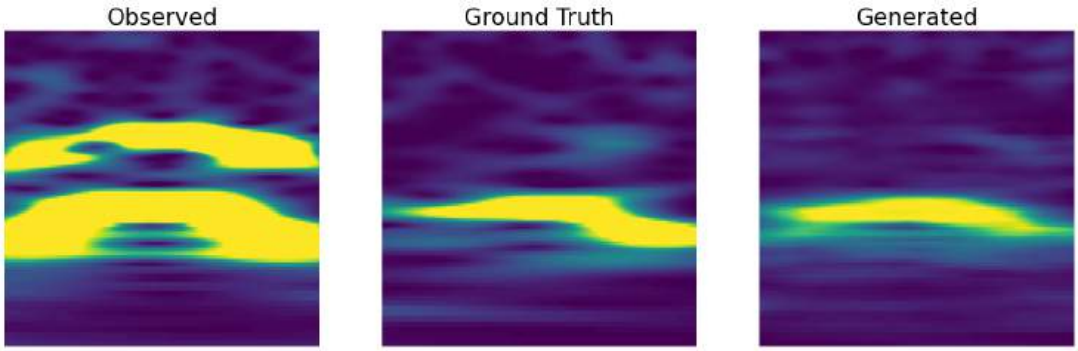
Figure 4.17: *Generation examples for Pix2Pix after 100 epochs of training, the* $L1$ *distance between observed images and ground truths is greater or equal than 0.20 (glitch class = scattered light; auxiliary channel = auxiliary channel 3).*



(a) *Example 1:* $L1$ *distance* $= 0.088$



(b) *Example 2:* $L1$ *distance* $= 0.110$



(c) *Example 3:* $L1$ *distance* $= 0.111$

Figure 4.18: *Generation examples for Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 3).*

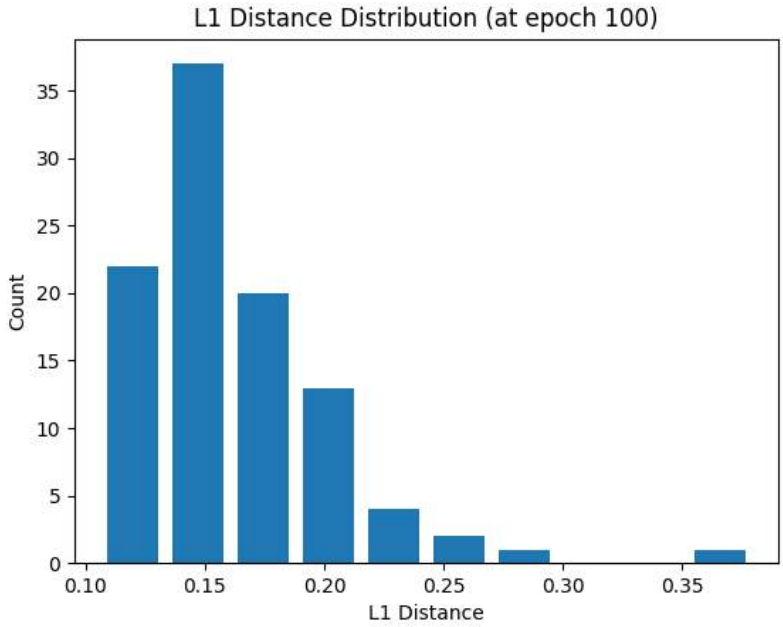
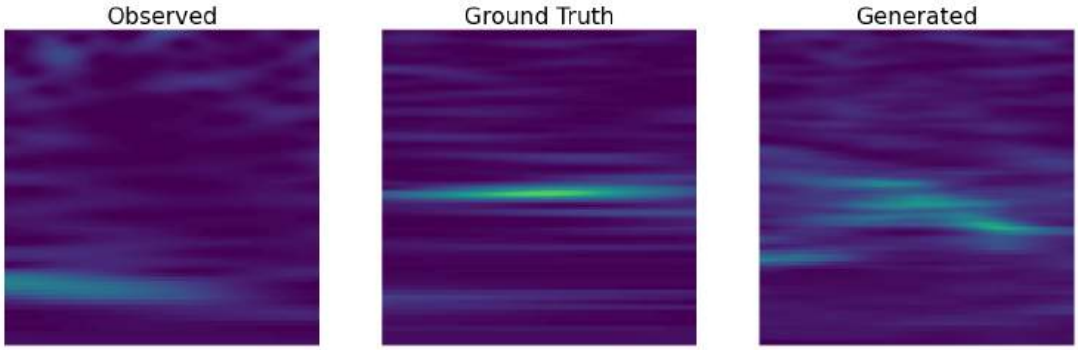


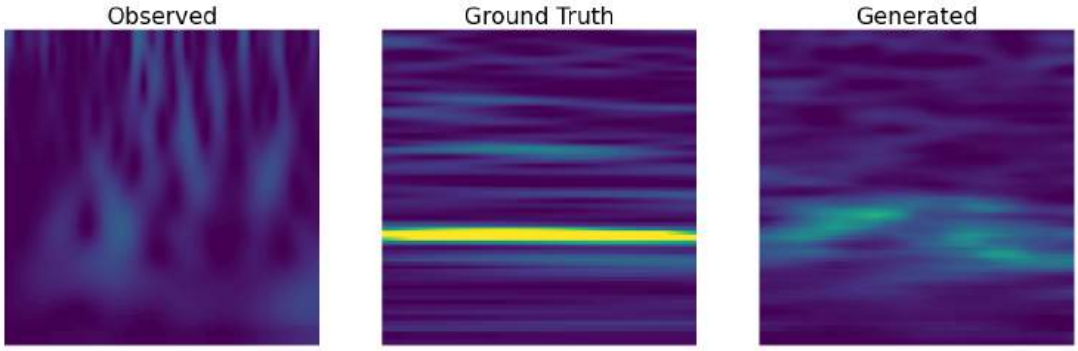
Figure 4.19: *Distribution of $L1$ distance between Pix2Pix generated images and ground truths took from the testing set, after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 7).*

generated images. Finding a quantitative, objective criterion to evaluate the generator's performance is not easy, the most popular is probably the "Inception Score", which takes its name from Google's Inception image classification model (based on convolutional neural networks) [38]. Indeed, the idea behind this score is to let a pre-trained image classifier evaluate the GAN's generations, and then the score of the generator is basically equal to the confidence with which the classifier places the generated image (in this case the glitch) in the correct class. Exploring this more advanced evaluation metric is beyond the purpose of this thesis and will be left for future works.

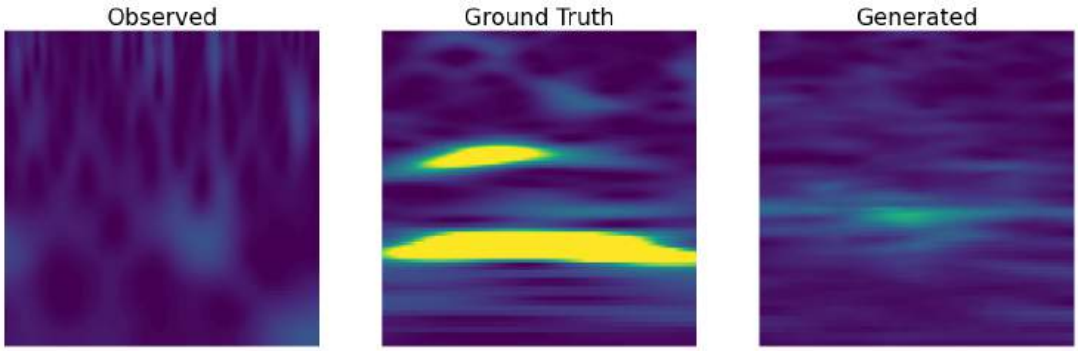
As I said in the introduction of this chapter, the auxiliary channels at our disposal are not correlated to glitches belonging to the other classes: low-frequency lines, koi fish, and blip. So, in these cases, the behavior of Pix2Pix is very similar to what we just saw for scattered light glitches with auxiliary channel 7, except, of course, for auxiliary channel 2, which is always correlated to the strain channel. Figure 4.21 shows the $L1$ distance distributions for koi fish, blip, and low-frequency lines glitches with auxiliary channel 2. Note that the mean of these distributions is lower than what we had for scattered light glitches, this is because, for these other classes, the glitch is always pretty much in the same position, making them easier to generate. Figure 4.22 shows a generation example for each class.
The network, based on the results we have seen so far, seems to be always able to figure out a good mapping between two channels when a correlation is present, even with few training samples. This happens both when the auxiliary channel is correlated with the strain channel, and when the auxiliary channel is correlated with the glitch class.



(a) *Example 1:* $L1$ *distance* $= 0.112$

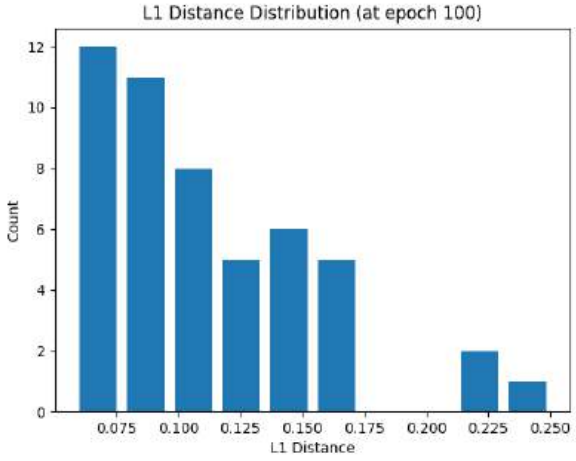


(b) *Example 2:* $L1$ *distance* $= 0.149$

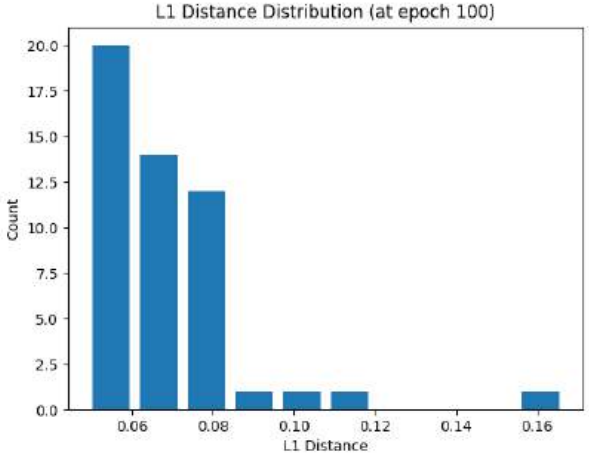


(c) *Example 3:* $L1$ *distance* $= 0.245$

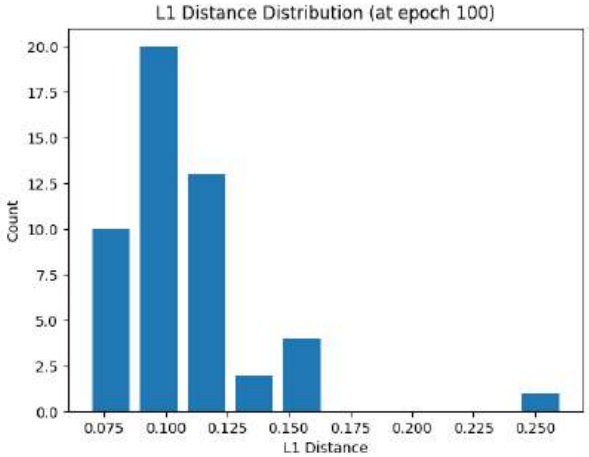
Figure 4.20: *Generation examples for Pix2Pix after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 7).*



(a) *Koi fish:* $\mu = 0.10$*;* $P_{95} = 0.15$.

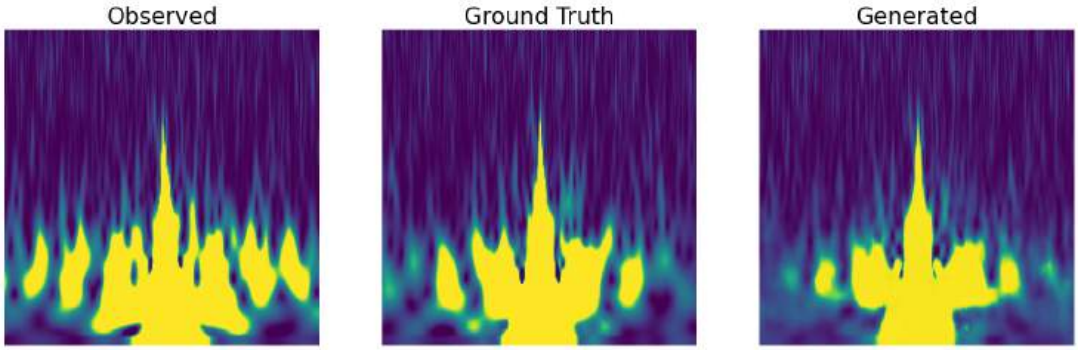


(b) *Blip:* $\mu = 0.07$*;* $P_{95} = 0.09$.

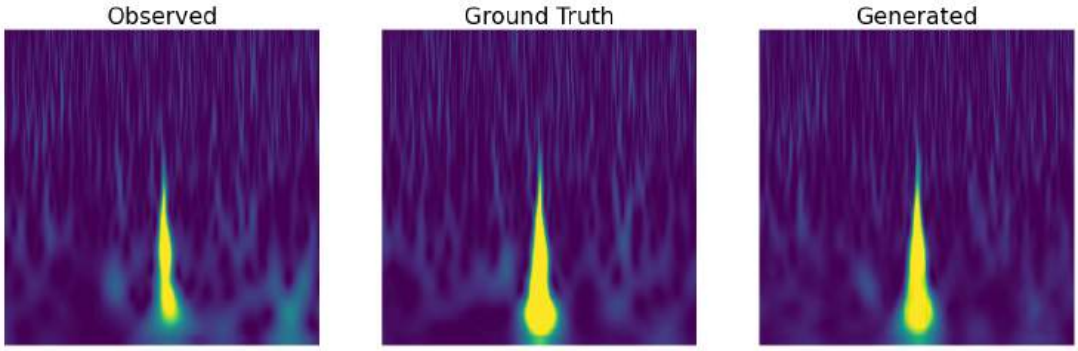


(c) *Low-frequency lines:* $\mu = 0.11$*;* $P_{95} = 0.16$.

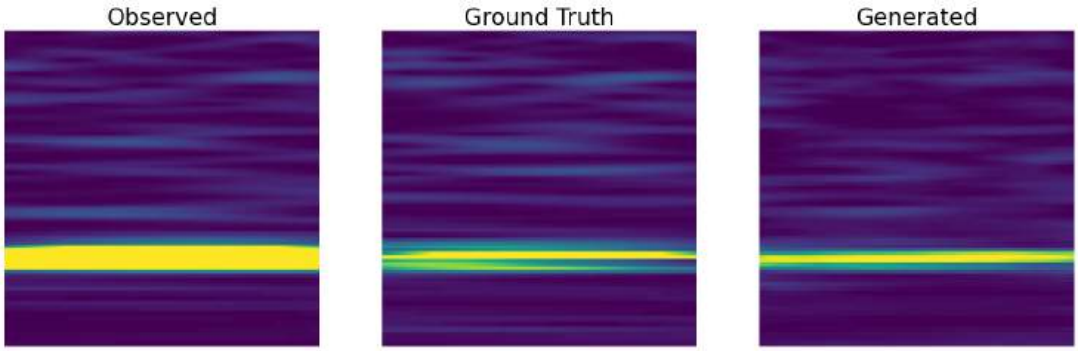
Figure 4.21: *Distribution of L1 distance between Pix2Pix generated images and ground truths took from the testing set, after 100 epochs of training (auxiliary channel = auxiliary channel 7).* μ *refers to the mean of the distribution, and* P_{95} *refers to its 95-th percentile.*



(a) *Koi Fish example*



(b) *Blip example*



(c) *Low-Frequency Lines example*

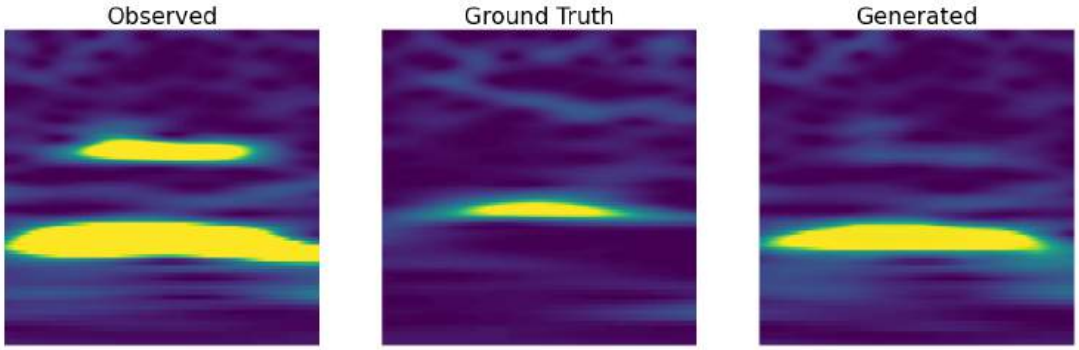
Figure 4.22: *Generation examples for Pix2Pix after 100 epochs of training (auxiliary channel = auxiliary channel 2).*

4.2 CycleGAN

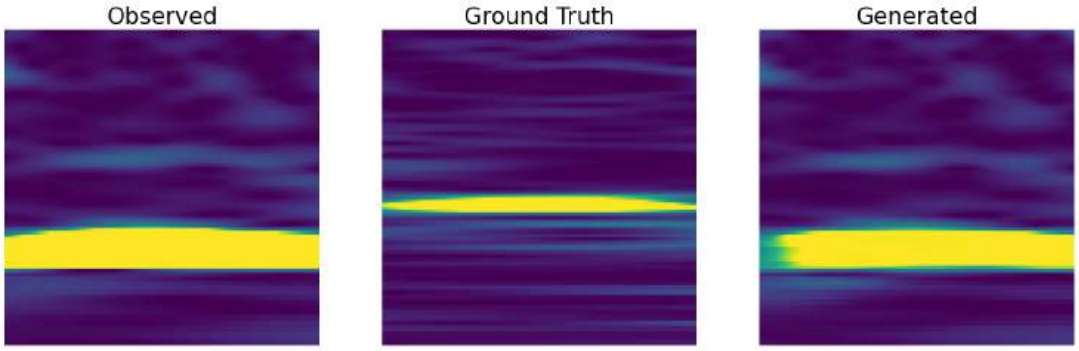
CycleGAN deals with unpaired Image-to-Image translation. To do so it requires two generator models and two discriminator models, making it much heavier in terms of computational cost than Pix2Pix. This is already a downside. Now I will show its behavior on scattered light glitches both on auxiliary channel 2 (section 4.2.1) and on auxiliary channel 5 (section 4.2.2). Note that I will call G_X the generator responsible for translating images in the auxiliary channel domain, Y, into images in the strain channel domain, X, and G_Y the generator that translates images in X into images in Y. Also, $L1_X$ refers to the $L1$ error of generator G_X, so it is the average $L1$ distance between real strain and generated strain images, calculated on the testing set; on the contrary, $L1_Y$ refers to the error of generator G_Y. Finally, D_X is the adversarial discriminator of generator G_X, so it discriminates between real strain images and generated strain images; on the contrary, D_Y is the adversarial discriminator of generator G_Y.
Note that, for clarity's sake, I decided to show in the examples only the images generated by G_X, which produces the mapping that we are interested in. To maintain the notation consistent with the previous section, I will indicate images from the strain channel domain as "Ground Truth", from the auxiliary channel domain as "Observed", and the images generated by G_X as "Generated".

4.2.1 Scattered Light - Auxiliary Channel 2 (Correlated)

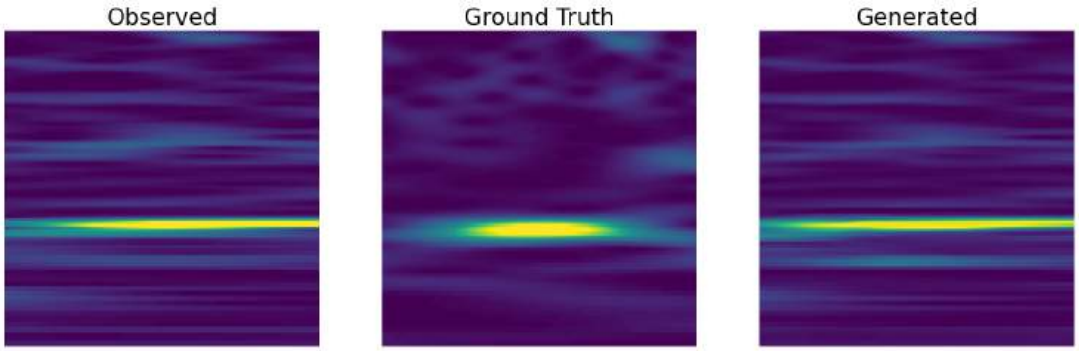
Some examples of generated images vs. ground truths taken from the testing set are shown in figure 4.23. We see that it seems like the model has essentially learned to produce the identity: usually the generated images are very similar to the observed images. There can be some adjustments made by the network, for example in figure 4.23a we see that generator G_Y removes the smaller yellow shape from the observed image to make the generated image more similar to the ground truth. The fact that the mappings learned by the generators are so close to the identity could be because the two domains are highly correlated, so the pictures of strain and auxiliary channel are already pretty similar, this could lead the generators always to produce the identity since it gives already a reasonably low loss. Indeed, figure 4.24 shows the trend of the losses vs. epoch for both generators and we can see that they, apart from some fluctuations, remained static during the whole training. Also the $L1$ error remained pretty much constant during the training (see figure 4.25), fluctuating around 0.22 for G_X and 0.24 for G_Y, that are pretty high values compared to the Pix2Pix standards. For completeness' sake, figure 4.26 shows the trend of the two discriminators. The large fluctuations, compared to what we had with Pix2Pix, are symptoms of unstable training (see section 4.2.3 for the discussion).



(a) *Example 1*



(b) *Example 2*



(c) *Example 3*

Figure 4.23: *Generation examples for CycleGAN after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

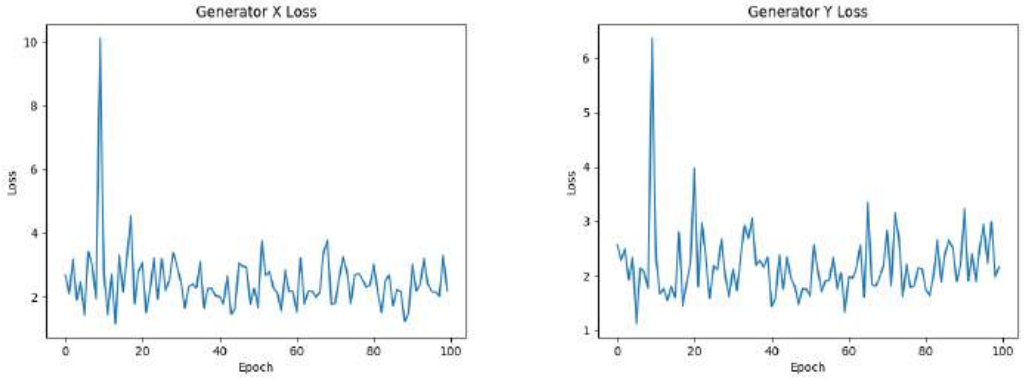


Figure 4.24: *Loss vs epoch for both generators of the CycleGAN (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

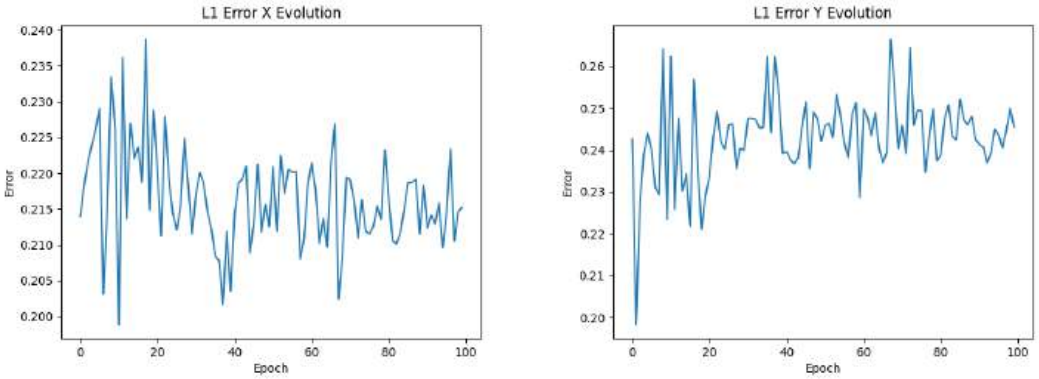


Figure 4.25: *L1 error vs epoch for CycleGAN (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

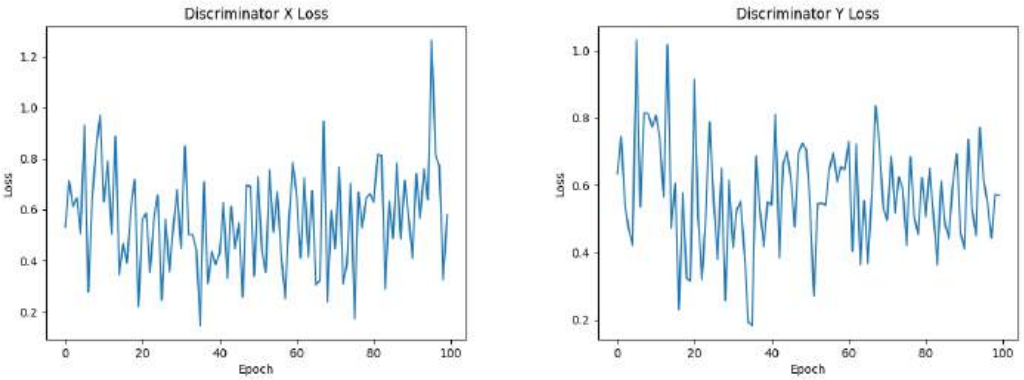


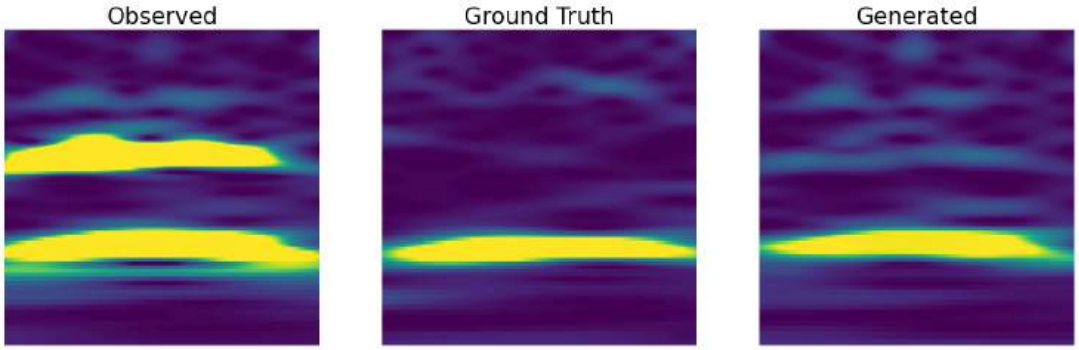
Figure 4.26: *Loss vs epoch for both discriminators of CycleGAN. (glitch class = scattered light; auxiliary channel = auxiliary channel 2).*

4.2.2 Scattered Light - Auxiliary Channel 5 (Uncorrelated)

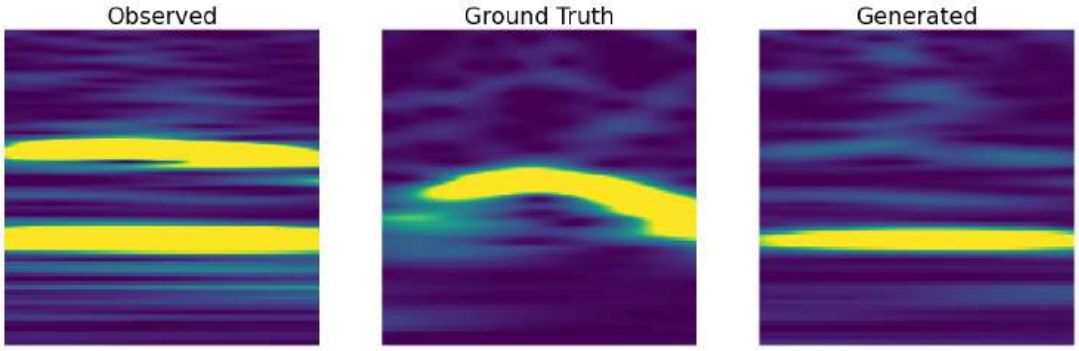
As before, figure 4.27 shows some examples of generated images vs. ground truths taken from the testing set. This time, the generations are more varied in terms of quality. There are some very good examples like figure 4.27a, but in the majority of cases, the results are pretty off, like in figure 4.27b and 4.27c. Anyway, it is interesting to notice that the mapping learned by the generators this time is not so close to the identity, in agreement with the fact that the two domains are not directly correlated in this case, so providing the identity would yield a too high loss for the generators. Nonetheless, the training is still very unstable, as we can see from the graphs in figure 4.28. The generators' losses seem to have a slightly downward trend, as well as the $L1_X$ error, on the other hand, the $L1_Y$ error is slightly increasing. I tried to extend the training for this particular instance of CycleGAN up to 300 epochs to see if these trends were significant in the longer run. The answer is no: the $L1_X$ error stabilizes around 0.22 and the $L1_Y$ error around 0.34 and the overall quality of generated images does not improve.

4.2.3 Conclusions on CycleGAN

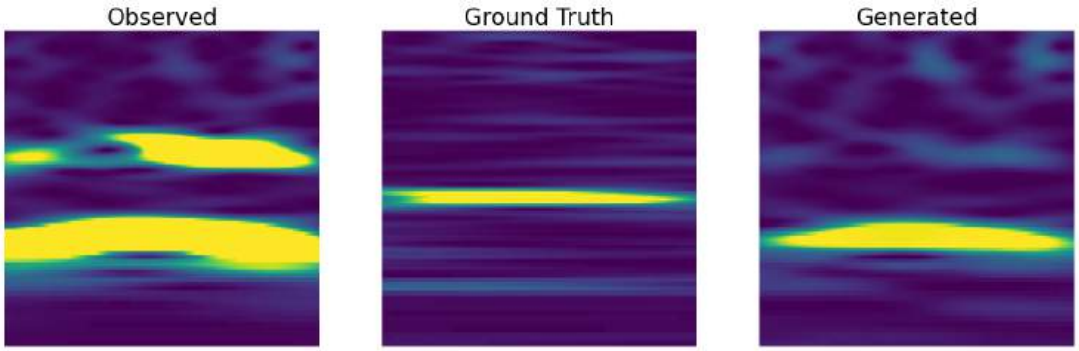
In conclusion to this section, I should point out that what I did is not a complete study of the CycleGAN model; this would require, e.g., trying to train it with a larger dataset and for a longer time and perform a thorough analysis of the model's hyperparameter space, which requires a large amount of time and computational resources. Due to the limited time I had, I decided to focus more on the Pix2Pix model, which provided quickly some very promising results, however, the poor achievements I got with CycleGAN are not, in principle, a sufficient reason to say that this model is not suited for this task. Nonetheless, we should consider that Image-to-Image translation with unpaired data is a much more complex task than the corresponding one with paired training data. It requires building two generators that should learn both the mapping from domain Y to X and the one from X to Y because this is what allows us to implement the cycle-consistency loss. This makes CycleGAN a slow-learning model and it is actually a waste of computational resources since, in the end, we are interested only in one mapping: the one that translates auxiliary channel images to strain channel images. Considering the way our dataset is built, providing paired training data, it is reasonable to think that we should drop the requirement of two generators and two discriminators, and use instead a simpler and more stable model like Pix2Pix.



(a) *Example 1*



(b) *Example 2*



(c) *Example 3*

Figure 4.27: *Generation examples for CycleGAN after 100 epochs of training (glitch class = scattered light; auxiliary channel = auxiliary channel 5).*

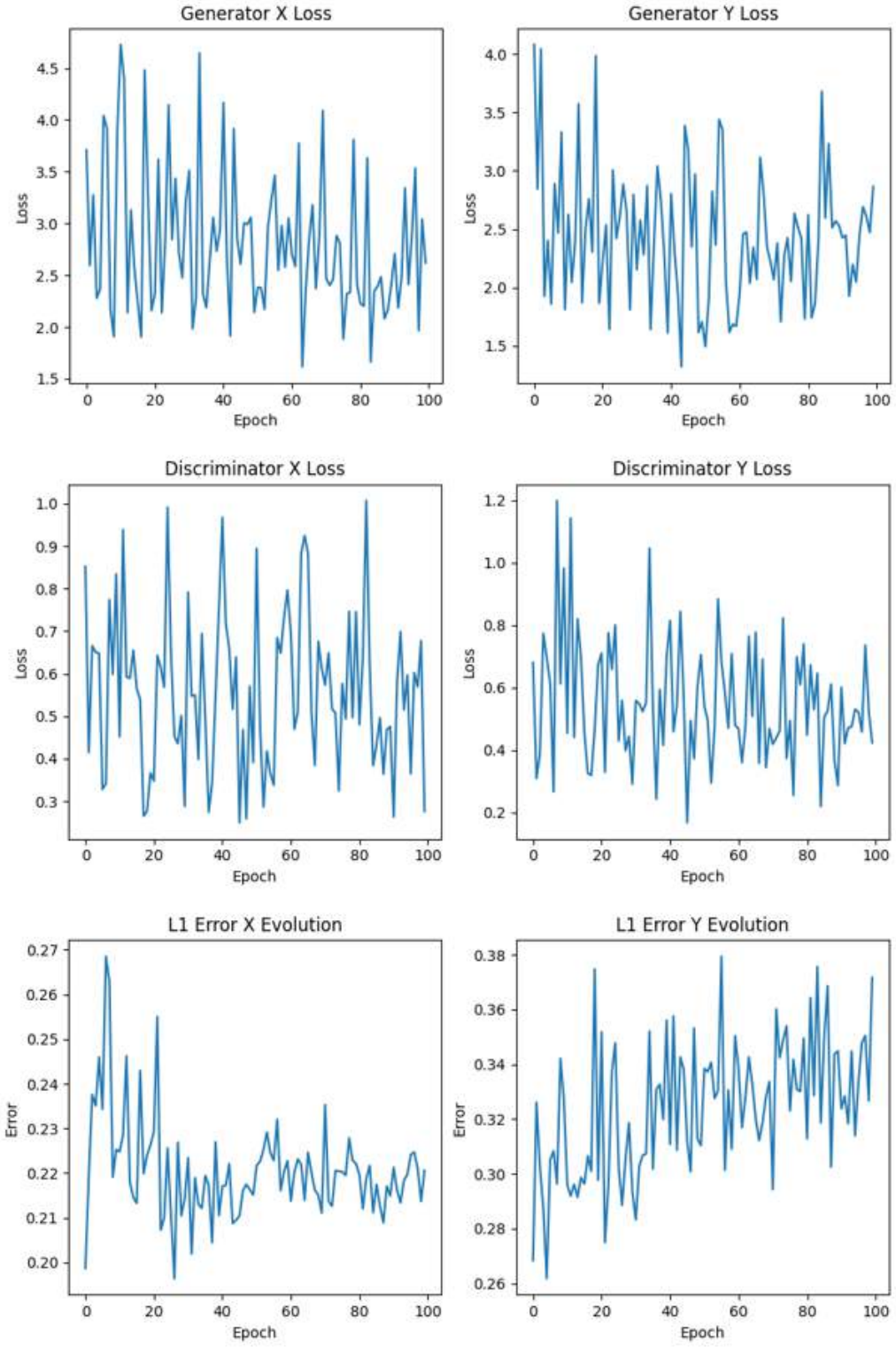


Figure 4.28: *Evolution of losses and L1 error for cycleGAN (glitch class = scattered light; auxiliary channel = auxiliary channel 5)*

Chapter 5

Conclusions

The goal of this thesis was to understand whether GANs could be appropriate models to translate glitches Q transform images from a particular auxiliary channel domain to the strain channel domain. Hence, I implemented two common GAN architectures that perform image-to-image translation: the Pix2Pix and the CycleGAN. After applying the appropriate preprocessing to turn the time series I had available as my dataset into glitch images, I could train and test the models. Note that some auxiliary channels are actually correlated to the strain one while others are not, and I was interested in understanding the performances of the models both in the case where the two channels were correlated and when they were not, so I chose to focus the study on two auxiliary channels, that I referred to as: number 2, being the correlated channel; and number 5, being the uncorrelated one.
The CycleGAN's performances were poor: being a heavier model than Pix2Pix, training required a much larger computational time, and results were not satisfying. As we have seen: when trained with data coming from the correlated auxiliary channel, the model was stuck in producing the identity of the observed image; and with the uncorrelated channel, the images generated by the model were also bad.
Pix2Pix's results, on the other hand, were satisfying in most cases. It provided good generations for every glitch class when trained with data of auxiliary channel 2. Also for other auxiliary channels that were correlated to scattered light glitches but not to the strain channel, Pix2Pix was able to translate correctly the glitch in the auxiliary channel into the glitch in the strain channel, indicating that, when there is a correlation between the two domains, the model can understand a proper mapping, without limiting itself to producing the identity of its input. Results obtained with Pix2Pix are summarized in table 5.1. However, Pix2Pix was not able to understand the case of no-glitch, when a signal present in the auxiliary channel does not result in a glitch in the strain channel.
I believe that training the network, providing samples of the no-glitch case in the training dataset could solve this problem and make the model generalize better in this case, but this is left for future works. Future works should also consider implementing a better metric for evaluating the performances of the generators since the $L1$ distance that I

	$L1$ - P_{95}	$L1$ - P_{80}	μ_{f-true}	μ_{f-gen}	σ_{f-true}	σ_{f-gen}	$\delta_f - P_{90}$
Aux 2	18.8	14.4	38.1 Hz	39.1 Hz	8.2 Hz	7.9 Hz	13.2 Hz
Aux 5	22.9	18.2	— " —	39.1 Hz	— " —	5.6 Hz	14.2 Hz

Table 5.1: *Summary of results for Pix2Pix model.* $L1$ - P_{95} *is the 95-th percentile of the* $L1$ *distance distribution and* $L1$ - P_{80} *is its 80-th percentile;* μ_{f-true} *and* σ_{f-true} *are the mean value and the standard deviation of the glitch frequency distribution for the ground truths while* μ_{f-gen} *and* σ_{f-gen} *are the same but for generated images;* δf - P_{90} *is the 90-th percentile of the distribution of errors in glitch frequencies.*

used is biased, as I discussed in section 3.3.2. Moreover, I decided to focus on Pix2Pix and CycleGAN, since they are the prototypes of image-to-image translation GANs for the case of paired training images and unpaired training images respectively, but there is a wide variety of GAN architectures and other kinds of models, like variational autoencoders and diffusion models, that should be explored to find the best fitting one for this task.
With my thesis, I think I proved the effectiveness of using generative models to transfer glitch images from an auxiliary channel domain to the strain domain, and that, after training such a model, the failure or the success of a glitch transferring is a good indication of whether the glitch is related to the auxiliary channel or not. Therefore, this method could be useful to understand which glitch classes are more related to which auxiliary channels, and hence, to figure out the physical origin of glitches. Finally, having well-trained models that work for the most relevant glitch classes and auxiliary channels, could be useful for confidently classifying events revealed in gravitational wave interferometers as glitches or astrophysical signals.

Bibliography

[1] Branchesi, Marica. "Multi-messenger astronomy: gravitational waves, neutrinos, photons, and cosmic rays." Journal of Physics: conference series. Vol. 718. No. 2. IOP Publishing, 2016.

[2] Amber L. Stuver. "Gravitational Waves" PhysicsWorld Discovery - January 2019

[3] McIver, Jess, and D. H. Shoemaker. "Discovering gravitational waves with Advanced LIGO." Contemporary Physics 61.4 (2020): 229-255.

[4] Spera, Mario, Alessandro Alberto Trani, and Mattia Mencagli. "Compact binary coalescences: Astrophysical processes and lessons learned." Galaxies 10.4 (2022): 76.

[5] Ott, Christian D. "The gravitational-wave signature of core-collapse supernovae." Classical and Quantum Gravity 26.6 (2009): 063001.

[6] Nussbaumer, Henri J., and Henri J. Nussbaumer. The fast Fourier transform. Springer Berlin Heidelberg, 1982.

[7] Roy, Tushar Kanti, and Monir Morshed. "Performance analysis of low pass FIR filters design using Kaiser, Gaussian and Tukey window function methods." 2013 2nd international conference on advances in electrical engineering (icaee). IEEE, 2013.

[8] Abbott, Benjamin P., et al. "Observation of gravitational waves from a binary black hole merger." Physical review letters 116.6 (2016): 061102.

[9] Abbott, Benjamin P., et al. "A guide to LIGO–Virgo detector noise and extraction of transient gravitational-wave signals." Classical and Quantum Gravity 37.5 (2020): 055002.

[10] Nardecchia, Ilaria. "Detecting gravitational waves with Advanced Virgo." Galaxies 10.1 (2022): 28.

[11] Durak, Lutfiye, and Orhan Arikan. "Short-time Fourier transform: two fundamental properties and an optimal implementation." IEEE Transactions on Signal Processing 51.5 (2003): 1231-1242.

[12] Podder, Prajoy, et al. "Comparative performance analysis of hamming, hanning and blackman window." International Journal of Computer Applications 96.18 (2014): 1-7.

[13] Schörkhuber, Christian, and Anssi Klapuri. "Constant-Q transform toolbox for music processing." 7th sound and music computing conference, Barcelona, Spain. 2010.

[14] Robinet, Florent, et al. "Omicron: a tool to characterize transient noise in gravitational-wave detectors." SoftwareX 12 (2020): 100620.

[15] Cuoco, Elena, et al. "Enhancing gravitational-wave science with machine learning." Machine Learning: Science and Technology 2.1 (2020): 011002.

[16] Razzano, Massimiliano, and Elena Cuoco. "Image-based deep learning for classification of noise transients in gravitational wave detectors." Classical and Quantum Gravity 35.9 (2018): 095016.

[17] Bahaadini, Sara, et al. "Machine learning for Gravity Spy: Glitch classification and dataset." Information Sciences 444 (2018): 172-186.

[18] Glanzer, J., et al. "Data quality up to the third observing run of advanced LIGO: Gravity Spy glitch classifications." Classical and Quantum Gravity 40.6 (2023): 065004.

[19] Tolley, Arthur E., et al. "ArchEnemy: Removing scattered-light glitches from gravitational wave data." arXiv preprint arXiv:2301.10491 (2023).

[20] Macleod, Duncan M., et al. "GWpy: A Python package for gravitational-wave astrophysics." SoftwareX 13 (2021): 100657.

[21] Abbott, Benjamin P., et al. "Characterization of transient noise in Advanced LIGO relevant to gravitational wave signal GW150914." Classical and Quantum Gravity 33.13 (2016): 134001.

[22] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).

[23] Gui, Jie, et al. "A review on generative adversarial networks: Algorithms, theory, and applications." IEEE transactions on knowledge and data engineering (2021).

[24] Creswell, Antonia, et al. "Generative adversarial networks: An overview." IEEE signal processing magazine 35.1 (2018): 53-65.

[25] Goodfellow, Ian J., et al. "Generative adversarial networks (2014)." arXiv preprint arXiv:1406.2661 1406 (2014).

[26] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.

[27] Van Erven, Tim, and Peter Harremos. "Rényi divergence and Kullback-Leibler divergence." IEEE Transactions on Information Theory 60.7 (2014): 3797-3820.

[28] Hurtik, Petr, et al. "Binary cross-entropy with dynamical clipping." Neural Computing and Applications 34.14 (2022): 12029-12041.

[29] Menéndez, M. L., et al. "The jensen-shannon divergence." Journal of the Franklin Institute 334.2 (1997): 307-318.

[30] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[31] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

[32] Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

[33] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

[34] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).

[35] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[36] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017.

[37] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14. Springer International Publishing, 2016.

[38] Borji, Ali. "Pros and cons of gan evaluation measures." Computer Vision and Image Understanding 179 (2019): 41-65.

[39] Xu, Qiantong, et al. "An empirical study on evaluation metrics of generative adversarial networks." arXiv preprint arXiv:1806.07755 (2018).

[40] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks." International conference on machine learning. PMLR, 2017.

[41] Lopez, Melissa, et al. "Simulating transient noise bursts in LIGO with generative adversarial networks." Physical Review D 106.2 (2022): 023027.

[42] Abbott, Benjamin P., et al. "Characterization of transient noise in Advanced LIGO relevant to gravitational wave signal GW150914." Classical and Quantum Gravity 33.13 (2016): 134001.

[43] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[44] Davies, Alex, and Alessandro Orsaria. "Scale out with GlusterFS." Linux Journal 2013.235 (2013): 1.

[45] Kubernetes, T. "Kubernetes." Kubernetes. Retrieved May 24 (2019): 2019.

[46] Docker, Inc. "Docker." lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker (2020).

[47] Kluyver, Thomas, et al. "Jupyter Notebooks-a publishing format for reproducible computational workflows." Elpub 2016 (2016): 87-90.

[48] Hardt, Dick. The OAuth 2.0 authorization framework. No. rfc6749. 2012.