School of Physics
and Astronomy

Machine Learning To Extract Gravitational Wave Transients

Vasileios Skliris

Submitted for the degree of Doctor of Philosophy
School of Physics and Astronomy
Cardiff University

30 September 2021

Summary of thesis

In this thesis we will present all the steps for creation of a pipeline that uses machine learning algorithms to detect sub-second gravitational wave transient signals. In the end we show the results of a search using the defined pipeline called MLy.

We begin with an introduction to gravitational waves where we show that gravitational waves are expected based on general relativity.

Then in Chapter 3 we make an introduction to machine learning an all its concepts and tools we are going to use during this work. We start with the definition of neuron and an overview of activation functions and gradient descend methods, to the parameters of a model and its layers and their optional and not, parameters.

In Chapter 4 we describe how we generate noise and injections, and we introduce the main functions and tools of the pipeline and we explain the reasons and condition under we use them. We start by defining our core objects DataPod and DataSet and continue with the generator function and all its extensions and schedulers , with detailed explanation of how we get the data and how we apply time-lags to the noise.

Following that in Chapter 5 we show our investigation on a model that detects gravitational wave transients. We show how we choose the parameters for the training and eventually the results of the best model on real noise from the second observing run.

Finally in the last Chapter 6 we present the application of that best model on an offline search for all three observing runs. In addition we show how we can extend an HLV trained model to an HL analysis without creating another model from scratch. In the end of that chapter we discuss our results and possible future investigations.

Acknowledgments

Those four years were an amazing experience for me, that exceeded any expectation I had for my PhD. This work is a result of hard work, a bit of luck and many people that supported and encouraged me.

First of all, I have to thank my supervisor Patrick Sutton, who was a professional and made me feel confident on my skills and work in every meeting. Thank you for focusing my inevitable stress where it was needed and never adding up to it. Also my secondary supervisor Bernard Schutz ,who I admired through his book which taught me general relativity, and who was always approachable and made me feel as an equal through our discussions. Thank you both for showing me that you can be professional and humble and there is no better way to do it.

I would like to thank Duncan Macleod for all the times that you figure out "why it wasn't working" and your amazing Gwpy that inspired all my codding skills the last four years. I would also like to thank Chris Messenger and his group in Glasgow that helped a lot on the direction of my project when I visited Glasgow and taught me how to use GPUs. I want to thank also Scott Coughlin for his suggestions and advice. I want to thank Shubhanshu Tiwari that provided me with extra information to complete this thesis. And of course want to thank all those people that spared some of their precious time to answer any probably stupid questions that I could actually google.

A special thanks to all my colleagues in the office, Rhys, Ali, Virginia, Charlie, Eleanor, Ed, Dave, Johnathan, Aldo, Alex, Seb, Ronaldas, Chinmay and even you Iain. Thank you for being there and made sure I will work half the time I was supposed to, so I don't get too tired. A special thanks to my fellow astrophysicists from the first floor that made sure Fridays were fun and always offered me tea.

I want to thank my special roommate genius Tanya Sign who was always looking at me suspiciously every time I passed by without any apparent reason and made me happy. Thank you for being that sister in the UK. My brain is yours.

I also want to thank Marika (my telescope) who has given me inspiration every time I was loosing mine and made me remember my roots and how it all started.

Finally I have to thank my Mum and Dad who supported my weirdness and know it all attitude from young age (and bought me Marika). I reached my dream because of you, this is your achievement as much is mine.

Contents

List of Figures

List of Tables

We call this pipeline MLy (Emily), in loving memory of my friend Emily Voukelatou who is not with us anymore. She was a fierce computer scientist and a master software engineer, who fought transphobia, sexism and misogyny all her life.

Chapter 1

Introduction

After the unexpected detection of the first gamma-ray bursts at the late 60s a new window of observation opened for astrophysics and nearly half a century later, gamma-ray astronomy is one of the most important fields of astrophysics. It made us understand a vast amount of new processes happening away from our own milky way. On September 14th of 2015 the first gravitational wave that was detectable from our detectors arrived, proving that from now on we would be able to "hear" the universe not only see it. After such a discovery a huge progress in the field of gravitational waves is expected and to the point of writing this we have more terrestrial detectors starting to be built and a future space gravitational wave detector - LISA. A lot of detectors means a lot of data and a lot of work for people that have to process those data.

Alongside the advances of astronomy in the new millennium we took more data that we could process on time. That created a demand for data scientist and most importantly the wide recognition of the term data scientist. Gravitational wave detectors produce a lot of data where most of them are auxiliary channels and environmental monitors. All these channels create signals and our job is to extract those signals and prove that they are real. A numerous amount of analytical methods have been produced to detect those signals that demand a descent amount of calculations, hence computational time. Although as the detectors are expected to become better and better with every upgrade the amount of detections will increase and alongside the demanded computational time will follow. From O2 it was clear that this will be a problem and another less costly method has do be developed.

Fortunately a promising tool for the task was already developed and it was called machine learning. Machine learning was developing along side the new data intensive age in computer science and it already had applications in our everyday life. Application of machine learning usually has quite different standards depending the problem it tries to solve and the discipline it belongs to. In our case this would be the extremely high standards for false alarm rates. Therefore the machine learning tool that will help with this issue has to be created from scientists that are adepts

in both fields of gravitational waves and machine learning.

Close to that point in history is when I started my PhD and the present work started building its foundations. During my first year I was mostly reading about machine learning and its current applications. I tried to recreate the results of Gabbard et al. 2018 [9] as a way to learn how machine learning algorithms work. It was a big surprise for me to see that the big challenge of this problem was not to find an algorithm that works but find the right processing of the data so that an algorithm can learn the important features from them. Furthermore at the end of my first year I realised the importance of working with real detector data and go beyond a proof of concept endeavour. In the end we want to apply an algorithm on real data, so it is important that it is tested in real data.

At the beginning of my second year I attended the Deep Learning for Multimessenger Astrophysics Conference in Urbana Illinois. Machine learning was in most of the projects presented there but alongside with it was a lot of mistrust about the results machine learning provides. The mistrust was created mainly from two factors. One was that the results back then were mostly proof of concepts that were not tested enough on real detector data and secondly the enthusiasm of machine learning researchers implying that match filter is outdated. It seemed to me that there was not room for middle ground at that point and I wanted to make it part of my endeavor to make machine learning for detections trustworthy.

To make machine learning trustworthy we needed to make it reproducible. If we ever have an algorithm that claims detections, we need to have a framework that is easy to use and with which other researchers would be able to recreate the results by themselves. That frameworks is the MLy-pipeline / package that we discuss in Chapter 4. The next year was mostly dedicated to built that framework. Further work would not be completed on time without it.

Furthermore into making a machine learning detection algorithm trustworthy, we need to be able to compare it's performance in exactly the same way with the analytical methods. Instead of recreating Gabbard et al. 2018 [9], I decided to try to do the same thing with real noise. Unfortunately the transition is and Odyssey that shows the big impact the data we use have in the performance, rather than the algorithm. Consequently I focused on finding the intricacies of such application on generic signals instead, given also that this was my goal from the start. This gave fruition to the paper we discuss in Chapter 5.

Finally in the last months of my PhD time I applied the successful algorithm of Chapter 5 to all the observing runs (Chapter 6), and I found possible caveats that different runs have for an algorithm.

Chapter 2

Gravitational Waves

In general relativity theory, gravity is not treated as a force, but as a result of curved space time. The gravitational "force" a body feels when is next to a massive object is *equivalent* to a case where this object is accelerated without the massive body present. This is known as the principle of equivalence. The curvature is present when gravity is present and objects that are made of energy create gravitational pull, even light. If the sun just disappeared out of thin air the Earth would still move around the sun for around eight minutes. This is because we know now that the information of gravitational force travels with the speed of light.

By the moment we realise that, the gravitational waves are inevitable. A wave can be any fluctuation, periodical or not, of information. This information can be anything including forces and people standing up in a stadium. When massive objects accelerate they create a wave in the space time. If the movement is periodical that ripple will be a periodical wave. The more massive the object the more intense the ripple will be.

In this chapter we will start by introducing the space-time metric, which describes the geometry of a space-time, and how the metric is determined by the Einstein equations and curvature. Then we will examine the behaviour of perturbations on flat space-time and show that those perturbations obey a wave equation, defining gravitational waves. Finally we will show some of the properties of gravitational waves and their interaction with matter.

This introductory chapter is based on Bernard Schutz book, "A first course in General Relativity" [10].

2.1 Gravity

2.1.1 Metrics and curvature

The metric fully describes the geometry of the space or space-time. It defines how do we measure the distance between any points. In flat three dimensional space we have the Cartesian metric which is a diagonal unit matrix, while in special relativity

we have the Minkowski space metric $diag(-1, 1, 1, 1)$. Both of those spaces are flat which means that if you move along a closed curve in those spaces and carry with you a vector that points towards a specific direction, when you come back to the origin point, that vector will point to the same direction. This is not true generally for a curved space, such as on a sphere, or in the space-time around a black-hole. Any curved space though is considered to be locally flat. This means that on sufficiently small scales we can always chose a coordinate system such that withing the neighborhood of a given point the metric will be approximately the flat metric. Earth is not flat but we walk on it like it is flat because the curvature doesn't interfere with our everyday life. Although when it comes to storms this curvature creates hurricanes and they do interfere.

To represent the metric in space-time we use $g_{\alpha\beta}$. When the indices are a letter of the Greek alphabet like here it means they include all the special plus time dimension $(t, x, y, z) \equiv (x^0, x^1, x^2, x^3)$. If they are a letter of the Latin alphabet they include only the special dimensions. When there two of the same letter in an expression, that means that we some over the parameters of the variables that have those indices.

$$x^\alpha y_\alpha \equiv \sum_{\alpha=0}^{3} x^\alpha y_\alpha$$

The curvature is defined as the effect of parallel transport in a loop, to a vector on a given space. For example when we apply the divergence on spherical coordinates there are many extra terms we need to add except the partial derivatives themselves. Those terms are an effect of curvature. This is called the covariant derivative and it is defined as:

$$\frac{\partial \vec{V}}{\partial x^\beta} = \frac{\partial V^a}{\partial x^\beta}\vec{e}_\alpha + V^\alpha \frac{\partial \vec{e}_\alpha}{\partial x^\beta}\,. \tag{2.1}$$

where x^β is the coordinates in which we want to find the derivative of $\vec{V}$ and $\vec{e}_a$ the basis vectors on witch we define that we define $\vec{V}$. For example in the Cartesian coordinates the basis vectors for x, y, z are dependent only their corresponding coordinate, hence the second term is zero for $\alpha \neq \beta$. On the contrary, in spherical coordinates the basis vectors for ϕ, θ depend on r coordinate. The term that is derived from the curvature of the space is the last term, which are called the Christoffel symbols:

$$\frac{\partial \vec{e}_a}{\partial x^\beta} = \Gamma^\mu{}_{\alpha\beta}\vec{e}_\mu\,. \tag{2.2}$$

Using the "," for partial derivative, equation 2.1 becomes:

$$\frac{\partial \vec{V}}{\partial x^\beta} = V^\alpha_{;\beta}\vec{e}_\alpha = \left(V^\alpha_{,\beta} + V^\mu \Gamma^\alpha{}_{\mu\beta}\right)\vec{e}_\alpha\,, \tag{2.3}$$

where $V^\alpha_{;\beta}$ is called the covariant derivative. When we are in a flat space covariant

derivative is identical with partial derivative, which means that Christoffel symbols will be zero in a flat space. It can be shown that the definition of curvature (where in flat space this is also zero) can be defined in terms of the Christoffel symbols:

$$R^{\alpha}{}_{\beta\mu\nu} = \Gamma^{\alpha}{}_{\beta\nu,\mu} - \Gamma^{\alpha}{}_{\beta\mu,\nu} + \Gamma^{\alpha}{}_{\sigma\mu}\Gamma^{\sigma}{}_{\beta\nu} - \Gamma^{\alpha}{}_{\sigma\nu}\Gamma^{\sigma}{}_{\beta\mu} \tag{2.4}$$

Now we will attempt to connect the curvature of space-time with the metric of that space-time. We begin from the Einstein's Equations in geometrized units (G=c=1). Which are 10 non-linear and coupled differential equations which describe how the metric of a space-time is related to the distribution of mass and energy in the space-time:

$$G^{\alpha\beta} = R^{\alpha\beta} - \frac{1}{2}R = 8\pi T^{\alpha\beta}\,. \tag{2.5}$$

where $R_{\alpha\beta}$ is the Ricci Tensor which is derived from the only non identical contractions of the curvature tensor:

$$R_{\alpha\beta} = g^{\mu\nu}R_{\mu\alpha\nu\beta}\,, \tag{2.6}$$

and R is the Ricci scalar which is also contractions of the curvature or the Ricci tensor with the metric of the system:

$$R = g^{\mu\nu}R_{\mu\nu} = g^{\mu\nu}g^{\alpha\beta}R_{\alpha\mu\beta\nu}\,. \tag{2.7}$$

All those algebrical notations are very useful so that we can manage to write the Einstein tensor ($G^{\alpha\beta}$) in relation only to the metric $g^{\alpha\beta}$. In a locally inertial frame the Christoffel symbols will be zero due to the nearly flat space. Although their derivatives will not be necessarily zero because the space is not flat by definition. It can be shown that in a locally flat coordinate system the curvature tensor can be written as:

$$R_{\alpha\beta\mu\nu} = \frac{1}{2}(g_{\alpha\nu,\beta\mu} - g_{\alpha\mu,\beta\nu} + g_{\beta\mu,\alpha\nu} - g_{\beta\nu,\alpha\mu}) \tag{2.8}$$

2.1.2 Linearised therory of gravity

If we assume a very weak gravitational field, we can translate that to a small deviation from the Minkowski metric and we can keep only first order contributions from this deviation. This is only valid for the cases where we are away from "strong" gravitational fields and we can assume that the metric is close to that of special relativity:

$$g_{\alpha\beta} = \eta_{\alpha\beta} + h_{\alpha\beta} \qquad |h_{\alpha\beta}| \ll 1\,. \tag{2.9}$$

If Equation 2.9 holds only away from "strong" gravitational fields, it cannot hold in any coordinate system, hence $h_{\alpha\beta}$ cannot be a tensor by itself.

We wish to write the Einstein equations in terms of the perturbation $h_{\alpha\beta}$, but

first we need to clarify the tensor behaviour $h_{\alpha\beta}$ has, while it is not a tensor. Assume a coordinate transformation $x^{\alpha'} = \Lambda^{\alpha'}{}_{\beta} x^{\beta}$. with a velocity v, $\gamma = 1/\sqrt{1-v^2}$ and for simplicity just do the transform to be into the x axis direction. The transformation matrix will be:

$$\Lambda^{\alpha'}{}_{\beta} = \begin{bmatrix} \gamma & -v\gamma & 0 & 0 \\ -v\gamma & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

In the boosted coordinate system the metric will be then:

$$g_{\alpha'\beta'} = \Lambda^{\mu}{}_{\alpha'} \Lambda^{\nu}{}_{\beta'} \eta_{\mu\nu} + \Lambda^{\mu}{}_{\alpha'} \Lambda^{\nu}{}_{\beta'} h_{\mu\nu} = \eta_{\mu'\nu'} + h_{\mu'\nu'} \tag{2.11}$$

This shows that the the perturbation term $h_{\alpha\beta}$ transforms like $g_{\alpha\beta}$ even if it is not a tensor. It is a part of $g_{\alpha\beta}$ that we decided to separate and gives us all the information regarding the small disturbance - curvature of our space.

This will help us apply simplifications based on tensor calculus to simplify Einstein's tensor. Any $h_{\alpha\beta}$ small enough will be valid, that means we have the freedom to choose some of it's characteristics.

One of these abilities is a specific type of coordinate transformation called the **gauge transformation**. We use a function ξ on the coordinates where:

$$x^{\alpha'} = x^{\alpha} + \xi^{\alpha}(x^{\beta}) \,. \tag{2.12}$$

The use of primed indices might confuse as they don't have their counterpart on the other side of the equation. These are not valid tensor equations, we just want to describe the relation between the components of the tensors in different coordinates.

This means that the components ξ^{α} are functions of the values of the initial coordinates. We want to make sure though that this transform will still keep $h_{\alpha\beta}$ small. Up to first order $\Lambda^{\alpha'}{}_{\beta} = \partial x^{\alpha'} / \partial x^{\beta} = \delta^{\alpha}{}_{\beta} - \xi^{\alpha}{}_{,\beta}$, which means we demand that:

$$|\xi^{\alpha}{}_{,\beta}| \ll 1 \,. \tag{2.13}$$

In the same way up to first order the transformation becomes:

$$g_{\alpha'\beta'} = \eta_{\alpha\beta} + h_{\alpha\beta} - \xi_{\alpha,\ \beta} - \xi_{\beta,\ \alpha} \tag{2.14}$$

$$h_{\alpha'\beta'} = h_{\alpha\beta} - \xi_{\alpha,\ \beta} - \xi_{\beta,\ \alpha} \tag{2.15}$$

As long as ξ follows the restriction 2.13 and the curvature is small we can also apply raising and lowering index like we have a Minkowski space: $\xi^{\alpha} = \eta^{\alpha\beta} \xi_{\beta}$. To this point the curvature tensor can be written in respect of $h_{\alpha\beta}$ without involving the gauge transformation

$$R_{\alpha\beta\mu\nu} = \frac{1}{2}(h_{\alpha\nu,\beta\mu} - h_{\alpha\mu,\beta\nu} + h_{\beta\mu,\alpha\nu} - h_{\beta\nu,\alpha\mu}) \tag{2.16}$$

Although when we attempt to write the Einstein tensor using equation 2.16 the result will not be trivial. For that reason we will use the **trace reverse** of $h_{\alpha\beta}$ assuming that it can have tensor behaviour as long we restrict ourselves to the coordinate transformations of the form given in the equation 2.12 subject to equation 2.13:

$$\bar{h}^{\alpha\beta} = h^{\alpha\beta} - \frac{1}{2}\eta^{\alpha\beta}h \qquad\qquad h^{\alpha\beta} = \bar{h}^{\alpha\beta} - \frac{1}{2}\eta^{\alpha\beta}\bar{h} \tag{2.17}$$

where $h = h^{\alpha}_{\ \alpha} = -\bar{h}$.

Including the previous transformation from equation (2.14)

$$\begin{aligned}
\bar{h'}^{\alpha\beta} &= h'^{\alpha\beta} - \frac{1}{2}\eta^{\alpha\beta}h'^{\nu}_{\ \nu} \\
&= h^{\alpha\beta} - \xi^{\alpha,\ \beta} - \xi^{\beta,\ \alpha} - \frac{1}{2}\eta^{\alpha\beta}(h^{\nu}_{\ \nu} - \xi^{\nu}_{\ ,\nu} - \xi_{\nu}{}^{,\nu}) \\
&= h^{\alpha\beta} - \frac{1}{2}\eta^{\alpha\beta}h^{\nu}_{\ \nu} - \xi^{\alpha,\ \beta} - \xi^{\beta,\ \alpha} + \eta^{\alpha\beta}\xi^{\nu}_{\ ,\nu} \\
&= \bar{h}^{\alpha\beta} - \xi^{\alpha,\ \beta} - \xi^{\beta,\ \alpha} + \eta^{\alpha\beta}\xi^{\nu}_{\ ,\nu}
\end{aligned}$$

By substituting the trace reverse metric into equation 2.16 and then substituting into the Einstein equations 2.5 it can be shown that the Einstein tensor gets simplified to:

$$G_{\alpha\beta} = \frac{1}{2}\left[\bar{h}_{\alpha\beta,\mu}{}^{,\mu} + \eta_{\alpha\beta}\bar{h}_{\mu\nu}{}^{,\mu\nu} - \bar{h}_{\alpha\mu,\beta}{}^{,\mu} - \bar{h}_{\beta\mu,\alpha}{}^{,\mu}\right] + O(\bar{h}^2_{\alpha\beta}) \tag{2.18}$$

To simplify the equation even more, we can demand from our free gauge functions xi^{α} to also satisfy that $\bar{h}^{\alpha\beta}_{\ \ ,\beta} = 0$, which is known as the Lorenz gauge and it is used also in electromagnetism. Given that $\bar{h}^{\alpha\beta}$ is symmetric and that the derivatives commute we will have only the first term left. The new $\bar{h}^{\alpha\beta}$ will be have to also follow the previous gauge conditions, restricting ξ:

$$\begin{aligned}
\bar{h}^{\alpha\beta}{}_{,\beta} - \xi^{\alpha,\beta}{}_{,\beta} - \xi^{\beta,\alpha}{}_{,\beta} + \eta^{\alpha\beta}\xi^{\nu}{}_{,\nu\beta} &= 0 \\
\bar{h}^{\alpha\beta}{}_{,\beta} - \xi^{\alpha,\beta}{}_{,\beta} - \xi^{\beta,\alpha}{}_{,\beta} + \xi^{\nu,\alpha}{}_{,\nu} &= 0 \\
\bar{h}^{\alpha\beta}{}_{,\beta} - \xi^{\alpha,\beta}{}_{,\beta} - \xi^{\beta,\alpha}{}_{,\beta} + \xi^{\beta,\alpha}{}_{,\beta} &= 0 \qquad\qquad \nu \to \beta \\
\bar{h}^{\alpha\beta}{}_{,\beta} &= \xi^{\alpha,\beta}{}_{,\beta}
\end{aligned}$$

$$\Box\xi^{\alpha} = \bar{h}^{\alpha\beta}{}_{,\beta}\,, \tag{2.19}$$

where:

$$\Box = -\frac{\partial^2}{\partial t^2} + \nabla^2\,, \tag{2.20}$$

is the d'Alembertian operator. From equation 2.19 we conclude that ξ^{α} must be a

solution of the wave equation with source $\bar{h}^{\alpha\beta}{}_{,\beta}$. It is proven that is always possible to invert the wave equation and solve it for any source. So we can always find a gauge transformation that takes us to Lorenz gauge. Moreover this transformation is not unique as any solution of the homogeneous wave equation $\Box\zeta^{\alpha}$ added to ξ^{α} can be as solution of 2.19.

For any $h_{\alpha\beta}$ that follows the Lorenz gauge condition equation (2.18) will be:

$$G^{\alpha\beta} = -\frac{1}{2}\Box\bar{h}^{\alpha\beta} \tag{2.21}$$

From which we can derive the **weak-field Einstein equations** to be:

$$\Box\bar{h}^{\alpha\beta} = -16\pi T^{\alpha\beta} \tag{2.22}$$

where $T^{\alpha\beta}$ is the stress-energy tensor.

2.2 Gravitational Waves

2.2.1 Propagation

Gravitational waves are produced from sources of gravitational fields that are not static, like the gravitational field of an accelerated mass. Assuming we are a way from such source of gravitational waves we can assume that the stress-energy is zero $T^{\alpha\beta} = 0$. Then the we only have to solve:

$$\left(-\frac{\partial}{\partial t} + \nabla^2\right)\bar{h}^{\alpha\beta} = 0 \tag{2.23}$$

The solution for a plane wave with a wave vector k_{ν} is:

$$\bar{h}^{\alpha\beta} = A^{\alpha\beta}e^{ik_{\nu}x^{\nu}}\,. \tag{2.24}$$

By applying the solution on the equation 2.23 we end up with the condition $k_{\nu}k^{\nu} = 0$, which means that the wave vector is a null vector $k_{\alpha} = (\omega, \vec{\mathbf{k}})$, where $\omega = |\vec{k}|$. Additionally if we apply the Lorenz gauge condition $\bar{h}^{\alpha\beta}{}_{,\beta} = 0$ we get $A^{\alpha\beta}k_{\beta} = 0$, which means that the direction of the wave is oscillation orthogonal to its direction of propagation, hence the wave is transverse.

Applying further gauge conditions we can choose $A^{\alpha}{}_{\alpha} = 0$ (traceless) and $A^{\alpha\beta}U_{\beta} = 0$, where U_{β} is the four-velocity of a fixed observer in the coordinate system and means that $\bar{h}^{\alpha\beta} = \bar{h}^{\beta\alpha}$. Therefore this is called the transverse-traceless gauge, where $h^{TT}_{\alpha\beta} = \bar{h}^{TT}_{\alpha\beta}$. By choosing the z direction as the direction of the wave it can be shown that the the general solution to the wave equation becomes:

$$h^{TT}_{\alpha\beta} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & A_{xx} & A_{xy} & 0 \\ 0 & A_{xy} & -A_{xx} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} e^{ik_\nu x^\nu} \tag{2.25}$$

where $k_\nu = (\omega, 0, 0, \omega)$. We will show later that those components of $A_{\alpha\beta}$ define the two main polarisations of gravitational waves. The plus polarisation (h_+) denoted by the symbol + and the cross polarisation $h_\times$ by symbol $\times$.

2.2.2 Interaction of gravitational waves and particles

Assume we have two free falling particles in the xy plane with a separation ϵ. We now investigate what will happen to the separation of the particles in the presence of a gravitational wave propagating in the z direction as we described above. We define the vector representing the separation of the two particles as the D^α. With the equation of geodesic deviation we can describe the distance between two particles as:

$$\frac{d^2}{d\tau} D^\alpha = R^\alpha{}_{\beta\mu\nu} U^\beta U^\mu D^\nu \,, \tag{2.26}$$

where $R^\alpha{}_{\beta\mu\nu}$ is the curvature created by the gravitational wave and U^β is a vector tangent to the geodesic. Assuming slow moving particles $d/d\tau \to d/dt$. Due to the facts that we want to approximate to first order in $h^{TT}_{\alpha\beta}$ and the curvature tensor is already first order we can simplify equation (2.26) in terms of $h^{TT}_{\alpha\beta}$. Additionally by using $h^{TT}_{0\beta} = h^{TT}_{z\beta} = h^{TT}_{\alpha 0} = h^{TT}_{\alpha z} = 0$, and equation (2.16) we will find that we have only the term $h^{TT\alpha}_{\beta,\mu,\nu}$ surviving:

$$\frac{d^2}{dt^2} D^\alpha = \frac{1}{2} h^{TT\alpha}{}_{\nu,00} D^\nu \,. \tag{2.27}$$

There are two obvious extreme cases from this result, that show the effects of each polarisation separately. (We will omit the exponent part for simplicity).

- $A_{xy} = 0$

  In this case the $A_{xy} = A_{yx} = 0$, only $\alpha = \nu$ terms will survive:

  $$\frac{d^2}{dt^2} D^x = \frac{1}{2} \left[\frac{\partial^2}{\partial t^2} A^x{}_x \right] D^x \qquad \to \qquad \frac{d^2}{dt^2} D^x = \frac{1}{2} \left[\frac{\partial^2}{\partial t^2} h_+ \right] D^x$$

  $$\frac{d^2}{dt^2} D^y = \frac{1}{2} \left[\frac{\partial^2}{\partial t^2} A^y{}_y \right] D^y \qquad \to \qquad \frac{d^2}{dt^2} D^y = -\frac{1}{2} \left[\frac{\partial^2}{\partial t^2} h_+ \right] D^y$$

  This is an oscillation on x axis and at the same time an opposite oscillation on y axis.

- $A_{xx} = 0$

  In this case the $A_{xx} = -A_{yy} = 0$, only $\alpha \neq \nu$ terms will survive:

$$\frac{d^2}{dt^2}D^x = \frac{1}{2}\left[\frac{\partial^2}{\partial t^2}A^x{}_y\right]D^y \qquad \rightarrow \qquad \frac{d^2}{dt^2}D^x = \frac{1}{2}\left[\frac{\partial^2}{\partial t^2}h_\times\right]D^y$$
$$\frac{d^2}{dt^2}D^y = \frac{1}{2}\left[\frac{\partial^2}{\partial t^2}A^y{}_x\right]D^x \qquad \rightarrow \qquad \frac{d^2}{dt^2}D^y = \frac{1}{2}\left[\frac{\partial^2}{\partial t^2}h_\times\right]D^x$$

In Figure 2.1 we show the effect of the two different polarisations on a ring of free falling particles. Note here that the particles do not move, but the space between them stretches and squeezes instead. We see that the two polarisations are rotated by 45^o in respect to each other.

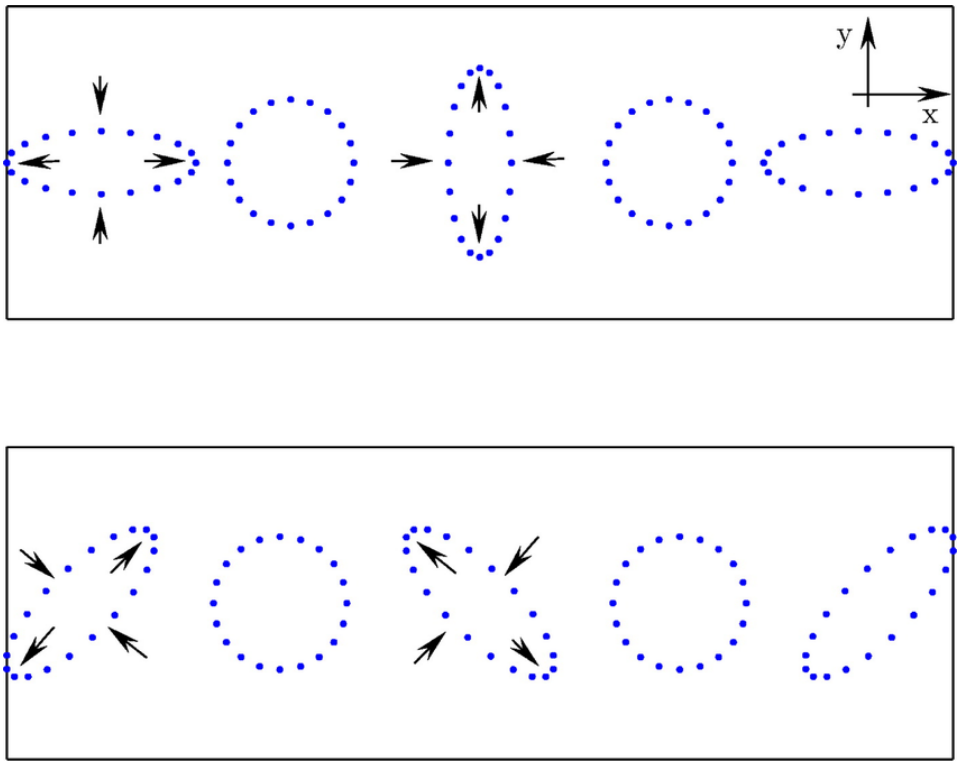

Figure 2.1: The effect of the main two polarisations of a gravitational wave moving in the z direction [1] over five different time snapshots. In both figures the you see a full period of a gravitational wave passing through a ring of free falling particles.(top) The effect of + polarisation. (bottom) The effect of $\times$ polarisation.

2.2.3 Generation of Gravitational waves by non-relativistic motion

In this section we will show the generation of gravitational waves from a non-relativistic oscillating mass, constrained to a spatial region much smaller than the wavelength itself. Equation 2.22 will be in that case:

$$\left(-\frac{\partial}{\partial t} + \nabla^2\right)\bar{h}_{\alpha\beta} = -16\pi T_{\alpha\beta}\,, \tag{2.28}$$

where we will chose the stress-energy tensor $T^{\alpha\beta}$ to be:

$$T_{\alpha\beta} = S_{\alpha\beta}(x^k)e^{-i\Omega t}\,, \tag{2.29}$$

where as we said $S_{\alpha\beta}$ is restricted to a region of size ϵ, much smaller than the wavelength of the wave eventually produced, $\frac{2\pi}{\Omega} \ll \epsilon$. Clearly 2.29 is not the most general form of a stress energy tensor. It simply is a plane wave, although a general stress energy tensor can be constructed out of an infinite sum of plane waves. Subsequently this approach, which is essentially a Fourier transform, will show us how each component wave of $T_{\alpha\beta}$ produce gravitational wave emission. We expect that the form of the solution will be:

$$\bar{h}_{\alpha\beta} = A_{\alpha\beta}(x^k)e^{-i\Omega t}\,, \tag{2.30}$$

and now we need to find the $A_{\alpha\beta}$ by applying all the restrictions. First of all by substituting into equation (2.28) we have:

$$(\Omega^2 + \nabla^2)A_{\alpha\beta} = -16\pi S_{\alpha\beta}\,, \tag{2.31}$$

where if we try to find a solution far away from the source where $S_{\alpha\beta} = 0$ we get a solution of the form:

$$A_{\alpha\beta} = \frac{B_{\alpha\beta}}{r}e^{i\Omega r} + \frac{C_{\alpha\beta}}{r}e^{-i\Omega r}\,, \tag{2.32}$$

where the first term represents an outgoing wave and the second term represents an ingoing wave. Since we are only interested in outgoing waves that are produced by the source, we can set $C_{\alpha\beta} = 0$.

Assuming that the mass of that source is inside a sphere of radius $\epsilon \ll 2/pi/\Omega$, we can integrate equation **??** over the volume of that sphere. Our goal is to express $A_{\alpha\beta}$ in terms of the source.

For the first part of the left hand side of the integral we can use the maximum value of $A_{\alpha\beta}$ to simplify the expression in terms of ϵ and Ω. The maximum value of that integral will be:

$$\Omega^2 \int A_{\alpha\beta}dx^3 \leq \Omega^2 \frac{|B_{\alpha\beta}|_{max}}{\epsilon}\frac{4}{3}\pi\epsilon^3 = \Omega^2|B_{\alpha\beta}|_{max}\frac{4}{3}\pi\epsilon^2\,.$$

In the second part of the left hand side of the integral we can apply Gauss' theorem:

$$\int \nabla^2 A_{\alpha\beta}dx^3 = \oint \mathbf{n}\nabla A_{\alpha\beta}dS = 4\pi\epsilon^2\left(\frac{dA_{\alpha\beta}}{dr}\right)_{r=\epsilon} \sim 4\pi B_{\alpha,\beta}\,.$$

We see that the first term is negligible compared to the second one. Finally by setting the right hand side of the integrated equation to be $J_{\alpha\beta} = \int S_{\alpha,\beta}dx^3$ we get:

$$4\pi B_{\alpha,\beta} + O(\Omega\epsilon)^2 = -16\pi J_{\alpha,\beta}, \tag{2.33}$$

So the solution for $\epsilon \to 0$ will be:

$$\bar{h}_{\alpha\beta} = \frac{4J_{\alpha\beta}}{r} e^{-i\Omega(t-r)}\,, \tag{2.34}$$

Now we will simplify $J_{\alpha\beta}$ by using it's relation to the stress-energy tensor and applying conservation rules on the stress energy tensor. The relation $J_{\alpha\beta}$ has with the stress energy tensor is:

$$J_{\alpha\beta} e^{-i\Omega t} = \int T_{\alpha\beta} dx^3\,. \tag{2.35}$$

By using the time derivative:

$$-i\Omega J^{\alpha\beta} e^{-i\Omega t} = \int T^{\alpha\beta}{}_{,0}\; dx^3\,. \tag{2.36}$$

Specifically for the case of $\beta = 0$ we have:

$$-i\Omega J^{\alpha 0} e^{-i\Omega t} = \int T^{\alpha 0}{}_{,0}\; dx^3\,, \tag{2.37}$$

which by using the conservation law of the stress-energy tensor:

$$T^{\alpha\beta}{}_{,\beta} = 0\,, \tag{2.38}$$

we can write 2.37 as:

$$i\Omega J^{\alpha 0} e^{-i\Omega t} = \int T^{\alpha k}{}_{,k}\; dx^3 = \oint T^{\alpha k}{}_{,k} n_k dS\,. \tag{2.39}$$

Since the oscillating masses are confined to the interior of the volume, that the stress energy tensor will be zero on the surface. Hence $J_{\alpha 0} = 0$ and due to the commuting of the derivatives $J_{0\beta} = 0$. Now there are only the spatial components J_{ab} to compute. Going back to equation 2.35 and keeping only the spatial components we have:

$$J_{ab} e^{-i\Omega t} = \int T_{ab} dx^3\,, \tag{2.40}$$

where from the tensor Virial theorem we have:

$$2\int T_{ab} dx^3 = \frac{d^2}{dt^2} \int T_{00} x_a x_b dx^3\,, \tag{2.41}$$

where for non-relativistic systems $T_{00} \sim \rho$. Furthermore:

$$I_{ab} = \int T_{00} x_a x_b dx^3 \tag{2.42}$$

is defined as the quadruple moment tensor of mass distribution. By substituting in equation (2.40) we have:

$$2J_{ab}e^{-i\Omega t} = \frac{d^2}{dt^2} I_{ab}\,. \tag{2.43}$$

Then by integrating twice on t and using the fact that if $J_{ab} = 0$ then also $I_{ab} = 0$:

$$I_{ab} = \frac{2}{\Omega^2} J_{ab}\, e^{-i\Omega t}\,. \tag{2.44}$$

That means that I_{ab} can be written as $D_{ab}e^{-i\Omega t}$. Finally the solution will be of the form:

$$\bar{h}_{ab} = -\frac{2\Omega^2}{r} D_{ab}\, e^{i\Omega(r-t)}\,. \tag{2.45}$$

It is convenient to find a gauge that is transverse in the direction of the wave. Additionally we will use a traceless form of the quadrapole moment tensor defined as:

$$\text{I}_{ab} = I_{ab} - \frac{1}{3}\delta_{ij} I^k_k\,. \tag{2.46}$$

Assuming the direction to be z, we can find a transverse traceless gauge that we will have $\bar{h}^{TT}_{zi} = 0$ and only the two polarisation terms will survive in the solution:

$$\begin{aligned} h_+ = \bar{h}^{TT}_{xx} = -\bar{h}^{TT}_{yy} &= -\frac{\Omega^2}{r}(\text{I}_{xx} - \text{I}_{yy})e^{i\Omega r} \\ h_\times = \bar{h}^{TT}_{xy} = \ \ \bar{h}^{TT}_{yx} &= -\frac{2\Omega^2}{r}\text{I}_{xy}e^{i\Omega r}\,. \end{aligned} \tag{2.47}$$

This solution is called the quadruple approximation of gravitational wave radiation.

2.3 Metrics of signal intensity

When discussing GWs we are often interested in characterizing the signal intensity (the power in the signal). To do this, we use two types of metrics to signify the intensity of a signal. Here we will discuss the Rooted Squared Sum (RSS) and the Signal to Noise Ration (SNR) metrics, and how we calculate them.

2.3.1 Rooted Squared Sum (RSS)

This metric uses all the intensity of the signal to calculate an average amplitude. The definition of h_{rss}, in continues and discrete form is:

$$h^2_{rss} = \int_{-\infty}^{+\infty} \left[h^2_+(t) + h^2_\times(t)\right]\ dt$$

$$h^2_{rss} = \frac{1}{f_s}\sum_{n=1}^{N} \left[h^2_+[n] + h^2_\times[n]\right]\,. \tag{2.48}$$

2.3.2 Signal to Noise Ratio (SNR)

The signal to noise ratio definition we use is not the peak over noise level one. The one we use for gravitational waves is the match filter SNR, which is an integration over frequencies of the ratio of the power spectrum of the signal $|\tilde{h}(f)|^2$ over the power spectral density $S(f)$ of the background, which refers to the sensitivity of the detector.

$$\rho = \sqrt{\int_{-\infty}^{+\infty} \frac{|\tilde{h}(f)|^2}{S(f)} df}$$

Given that we work with real data $S(f) = S(-f)$. For that reason it is a convention to use the one-sided PSD. To keep the normalisation correct we need to substitute $S_1(f) = 2S(f)$ and also the integral will be symmetrical, if we change the limits to be from 0 to $+\infty$ we get:

$$\rho = \sqrt{4\int_{0}^{+\infty} \frac{|\tilde{h}(f)|^2}{S_1(f)} df}$$

In our data analysis we only use the discrete Fourier transform. The integral is now a sum from $1/T$ to the niquest frequency $f_s/2$. Moreover the $df = f_s/N = f_s/(Tf_s) = 1/T$, where T is the duration in seconds.

$$\rho = \sqrt{4\sum_{n=1/T}^{f_s/2} \frac{|\tilde{h}_1[n]|^2}{S_1[n]}} \tag{2.49}$$

Comparison of h_{rss} and SNR The core difference between the two is that SNR uses the detector noise level of every frequency to normalise the signal. That will make a signal around $10Hz$ and a signal of $100Hz$ that both have the same h_{rss} to have a dramatic difference in SNR. The noise of the detectors is large around $10Hz$ compared to $100Hz$, which will diminish the the low frequency signal after normalisation.

Those two metrics can only be compared when the signal is in a reasonably narrow band of frequencies. Then the integral would be non zero only around that narrow band of frequencies, where the signal is present, and $S(f)$ will be practically a constant and could go out of the integral. Still the comparison though depends on the value of that constant. In any other case they cannot be directly compared.

Chapter 3

Machine Learning

The term machine learning is used to describe algorithms that try to create an "empirical" representation of a process. Humans use machine learning techniques all the time and we apply it effortlessly. A good example is learning to walk, where from our first attempts there is a natural system of reward if we manage to balance and a penalty of a fall if we fail. Every time we tried we failed better until we mastered the skill of walking. How do we do it though? If somebody ask you how do you walk the common answer is "I just do it". Years of practice and unfortunate falls created muscle memory that reacts to our balancing mechanisms in our ears. A plethora of processes and decisions we make are based on not really reasonable causes but rather a feeling, a hunch or an instinct. If we tried to calculate every move we do while walking based on position, wind and inclination of the floor we would never have enough time to think even one step given the time it takes to walk a step.

In 1950's when Arthur Samuel developed an algorithm to play checkers [11] and it was one of the first examples of an algorithm learning how to do something based on a system of reward and penalty. After that many people tried to beat games like chess and go using algorithms. Those algorithms were not calculating all the possible courses a game can have given the stage (analytical way). Instead Samuel developed a scoring system that represents the chance of winning for each player in every turn of the game. The algorithm was able to learn the right scoring after training. The most important feature of that achievement is that computer checkers wouldn't take much memory space compared to using an analytical algorithm.

Around the same time, two neurophysiologists were doing research on something that was going to develop a field totally different than their own. Their research was about how vision is actually perceived by the brain. By experimenting on the cortices of cats [12] they realised that they contain neurons that react only if light hits them around a specific area. Those areas are overlapping between neurons and all together they create cat's visual field. Following experimentation on monkeys in 1968 [13] they discovered that the visual cortex has a type of cell that can identify

edges and their orientation. This was the inspiration for Kunihiko Fukushima, a Japanese computer scientist to create the first convolutional layers, which we will describe in detail in subsection 3.2.4. The model he created called *Neocognitron*[14] had the conventional convolutional layers and also the feature of down sampling. This was the foundation of the Convolutional Neural Networks (CNN) as we know them today.

After years, CNNs proved to be the champions of what the first concept was trying to imitate. They are used widely in image recognition due to their ability to distinguish between two similar objects of the same class. They have outperformed humans on this task with the current champion being the *Inception* network [15]. To compare different models on a type of task like classification, there are specific data problems used called benchmark problems. A very famous benchmark problem of classification that is used to compare the performances of models like *Inception* network [15] is the MNIST dataset [16]. This dataset consists of images of ten different types of handwritten digits (0 to 9) and the goal is to distinguish them from each other.

To understand the way a network works we need to start from the smaller structures in it. In this chapter we will discuss all the parts that make a neural network be functional. Starting from the neurons in subsection 3.1.1, and they way they can learn simple functions in subsection 3.1.2. Going through different types of activation functions, in subsection 3.1.3, that help us introduce non-linearity in our models. Then we will discuss loss-functions in subsection 3.1.4 and how different gradient descent algorithms minimise the loss with different approaches, in subsection3.2.1. Finally we will talk in detail about the main two types of neural networks we use in this project starting with fully connected networks in subsection 3.2.3 and finishing with a discussion about how Convolutional Neural Networks (CNNs) are computing their input and back-propagation in subsection 3.2.4.

3.1 Foundations

3.1.1 The neuron

The neuron is the building block of neural networks. The name “neuron” is inspired from the simplified shape of organic neurons. Note though that the complexity of organic neurons is extremely higher that the complexity of the artificial neuron. Although it serves the purpose of explaining in a simple way how our artificial neurons work.

The artificial neuron is a process that applies a weighted sum on its inputs, and following that a function that digests this sum and gives another number as an output. The reason the artificial neuron evolved as it did is that in the beginning artificial neurons were simple decisions trees that were doing a simple, but most

importantly linear calculation. The function that the neurons apply at the end is called the activation function, and at the beginning was identity (linear) function. The problem with the neuron at this point is that it could simulate only linear functions because by itself it was a linear function. To fix this issue more types of activation functions were created that introduce a non linearity, as it is discussed later in subsection 3.1.3.

A general neuron has as many weights as inputs plus a variable called bias. So the final description of a neuron in a functional form is,

$$N(\vec{x}) = A\left(\sum_{i=1} w_i x_i + b\right) , \tag{3.1}$$

where $\vec{x} = (x_1, x_2, \ldots, x_i)$ is the input you have, $\vec{w} = (w_1, w_2, \ldots, w_i)$ the relative

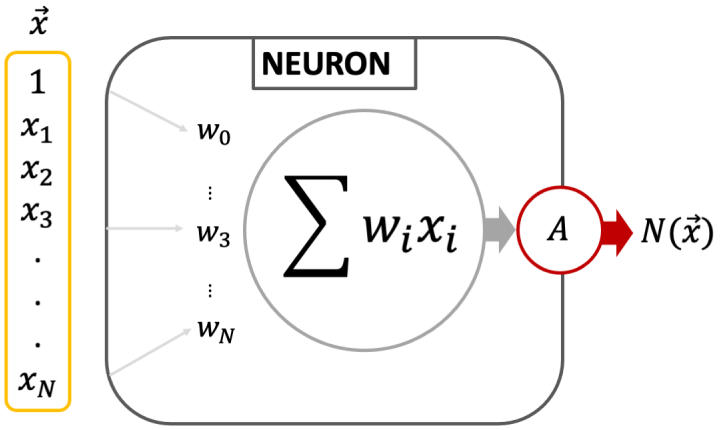

Figure 3.1: Diagram of the artificial neuron

weights, b the bias and $A()$ is the activation function. A better representation happens when we attach an $x_0 = 1$ in place of the bias term a so that $x_0 w_0 = x_0 b = b$. Eventually the function of the neuron is:

$$N(\vec{x}) = A\left(\sum_{i=0} w_i x_i\right) . \tag{3.2}$$

3.1.2 Training a single neuron

A neuron by itself can be trained to simulate a simple linear function. We will demonstrate an example of training a neuron with the simple function:

$$f(x_1, x_2, x_3) = a x_1 + b x_2 + c x_3 \, . \tag{3.3}$$

To do that we will need a simple neuron with three inputs and a linear activation function. So the weighted summation and activation function are defined as:

$$S(\vec{x}) = \sum_{i=0}^{3} w_i x_i \qquad A(S) = S\,. \tag{3.4}$$

Initially the weights $\vec{w}$ are uniformly distributed random numbers and our goal is to update them in a way that they become the weights of the equation $(0, a, b, c)$. For this to happen we need training data that we can put through the neuron and then see how close is the output to the true value. Then we have to quantify that closeness and finally use it to update the weights so the result comes even closer to the true one. To define how bad or good our neuron performed in each trial we have to define a loss function. For simplicity we will use a quadratic

$$L(y) = (y - y_{true})^2\,, \tag{3.5}$$

where $y_{true} = f(\vec{x})$ and $y = A(S)$ is the output of the neuron. We chose $A(S) = S$ for simplicity but as we will see later $A(S)$ is not equal to S because activation functions don't have to be identities. The loss function is related to the weights, hence we can calculate the derivative of the loss over each weight and figure out how much a weight is affecting the loss. So lets calculate this derivative:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial A}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial A}{\partial S}\frac{\partial S}{\partial w_i}\,. \tag{3.6}$$

Let's see all the those parts of the derivative. For every neuron $\frac{\partial S}{\partial w_i} = x_i$ by definition $\frac{\partial A}{\partial S} = 1$ and as you can see the derivative of such activation function is independent of S, hence independent of the weights themselves this makes linear functions not useful in more complicated cases of neural networks. Finally the last term:

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial y} = 2(y - y_{true})\,,$$

is the only term dependent of w_i through y. Now we have the derivative of loss over each weight:

$$\frac{\partial L}{\partial w_i} = 2x_i(y - y_{true})\,. \tag{3.7}$$

This process of finding those derivatives is called back-propagation and here we see the simplest example of it. We will encounter it again later in a more advanced level where more than one neurons is involved. The only thing left is to upgrade the weights proportionally to their influence ($\frac{\partial L}{\partial w_i}$). To control how radical those changes are going to be we multiply this influence with a learning rate η and then update the weights. Although there are two ways to update the weights, one by adding the change and one by subtracting it. As we show in Figure 3.2 the correct answer is subtracting, so that the new value of the weight will make the loss smaller.

$$w_i \rightarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

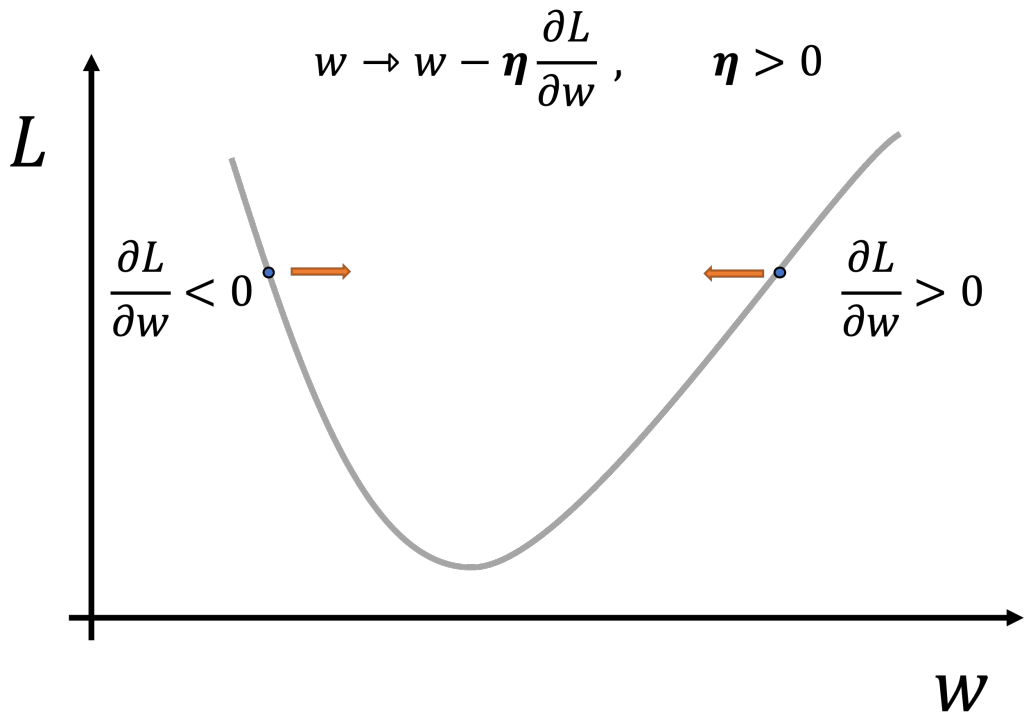

Figure 3.2: Graphic explanation of why we use the minus to update the weights. The point on the left has a negative gradient, and the best value for w precedes the current value. Hence the update of the weight should be positive and the only way to do that is to use the minus. The opposite case applies to the other point where the gradient is positive and the true value is behind the current value so the minus still updates the weight to the right direction.

A whole cycle of calculating the gradient and updating the weights accordingly is called a training **epoch**. To this point we have completed one epoch of training for this neuron. To fully train it we can repeat this process again as many times as we want to minimise the loss. There is no best activation or loss function. For each problem the ones that work better are often identified empirically by trying different ones and comparing them for a given problem.

3.1.3 Activation functions

Activation functions are a crucial component that can make a model learn successfully. The reason we use them at the end of a neuron is to restrict the result between an interval and prevent from small or big numbers that could create updates which could be insignificant or far from the real value respectively. By such a restriction we normalise the outputs of neurons. If we didn't do that a division with a very small

number might create a huge derivative that will make our weight orders of magnitude away from the real value. That is why from early one the activation functions were applied. No one activation is optimal for all applications. Most of them are good and some of them have great performance for specific types of problems. We will go through some of the most famous activation functions here. In the previous subsection we used the symbol S to denote the weighted sum of the input, although we will just use x here for simplicity because we focus on the activation function and not the whole process.

Linear activation is the simplest activation that can be applied.

$$A(x) = kx \qquad k \neq 0\,. \tag{3.8}$$

In the case where $k = 1$ we have a trivial activation function, which cannot work successfully in non trivial neural networks because the back propagation derivative will always return a constant. The linearity gives no power to the activation function to influence the weights in a non linear way. It can usually be applied in cases where the solution can also be derived analytically, which makes it redundant.

Sigmoid activation follows the sigmoid function:

$$A(x) = \sigma(x) = \frac{1}{1 + e^{-kx}}\,. \tag{3.9}$$

This function restricts the output inside the interval $(0, 1)$. The parameter k is called the gain and it controls how steep the transitioning is from 0 to 1.

$$A'(x) = \sigma'(x) = k\sigma(x)(1 - \sigma(x))\,. \tag{3.10}$$

The sigmoid function derivative is restricted between $[0, k/4]$. In the case of one neuron when we try to update the weights, we can see that from equation (3.10) $\partial A/\partial x \to 0$ for $|x| \gg 1$. Which means that $\partial L/\partial w_i \to 0$. While in the area around $x = 0$ the values are proportional to the gain and they "survive", hence the update to the corresponding weight is significant.

Having a derivative close to zero makes the gradient to go to zero. This phenomenon is called **vanishing gradient** and we will see later how serious it is when we have many layers of neurons. The more layers the more probable for this to happen. In general we try to avoid it by making sure all components of a model do not produce really small or really large numbers, because both cases correspond to gradients close to zero.

Tanh is a function similar in form to the sigmoid. This function restricts the output to the interval $[-1, 1]$. It can be shown that $\tanh(kx)$ can be written as:

$$A(x) = \tanh(kx) = 2\sigma(2kx) - 1\,. \tag{3.11}$$

The derivative is of the same nature as that of the sigmoid. Both have the same problem of the vanishing gradient for high magnitude inputs.

ReLU (Rectified Linear Unit) is the first of a series of activations that are based on the linear function. Although half linear, this function is non-linear as it discriminates between negative and positive input.

$$f(x) = \begin{cases} kx & : x \geq 0 \\ 0 & : x < 0 \end{cases}\,. \tag{3.12}$$

During back propagation the derivative is k for any positive result and zero for any negative. A characteristic of ReLU is that it converges very quickly because it is not asymptotic in its range like sigmoid and tanh are. A big loss will be translated to a big gradient proportionate to the loss. Although like sigmoid and tanh the gradient vanishes for negative values.

Leaky ReLU is the first obvious modification of ReLU to keep it non-linear but fix the vanishing gradient problem.

$$f(x) = \begin{cases} Kx & : x \geq 0 \\ ax & : ax < 0 \end{cases}\,, \tag{3.13}$$

where $a \ll 1$. A common choice is $a = 0.1\,, k = 1$ in general. This creates a constant derivative for negative values that is not zero, preventing the vanishing effect.

Parametric ReLU solves the problem of selecting the parameters of leaky ReLU. Instead of deciding the values a, k, they are treated as trainable parameters.

Softmax is a special case of activation function because it has more than one input values which are also equal to the number of the output values. We will denote those inputs as an array $\mathbf{z}$,

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}\,. \tag{3.14}$$

This function is the generalisation of the sigmoid function in more than one dimensions and it is used as final activation in classification problems. Softmax activation translates numbers of arbitrary size into a range (0,1). More importantly the sum of the output values of this function is equal to 1. This makes possible to treat the those values as probabilities. Although it is important to remember those outputs cannot be treated as probabilities without enough justification from statistics.

3.1.4 The loss function

Until now we discussed how a neuron is able to learn the right weights to describe a function. This training process is not happening in one step. It is a repetitive process that requires an algorithm with appropriate finesse.

A neural network algorithm has two main states. One is the initial state where we created the architecture and we have initialised the weights to values of our choice (usually random). In this state we haven't calculated any loss, we haven't even introduced the data. The second state is when the network has been through any kind of weight updates. After each epoch of training there is different version of a trained state. We can train a network as many times (epochs) as we want. At the end of final training we are at the fully trained neural network state. While we are still in the first state, we have to define the two pillars of training, the **loss function** and the **gradient descent algorithm**.

As we saw before on the single neuron example we needed a metric that checks how far from the correct answer the model is for a given input. In regression problems like a least squares fit, the output parameters are combined through a function to give a final result which then we compare with the expected value from this function. The way we compare the real value with the predicted one is called the loss function. We already saw the use of the quadratic loss function 3.5 when we were trying to use a single neuron to fit a function. The work we describe in chapter 5 is a classification problem and we will focus on components that are made for classification problems. Here we will discuss the most commonly used loss functions in classification problems.

In a real training scenario we will have many data to train with. As we will discuss further later, we don't use those training data one by one to calculate how far we are from the real value because the data are noisy, and noise can create outliers that can derail our training. It is common in the machine learning community divide our data to small datasets which we call batches. From now one batch is a subgroup of a dataset and batch size is the number of data instances this batch has.

Categorical cross-entropy Assume we have a classification problem where we want to classify a type of data to M different classes. After we feed data to the algorithm it will return M scores for each class. If we use the softmax activation function as the final activation, we will have an array with values in a probabilistic form. We expect that all the predicted scores will have a non zero number however small, because machine learning algorithms are not perfect. The correct scores of the array given that we know the correct class are always zero, except for the score of the correct class which is one. For one given data instance sample, we can describe the probability of that sample belonging to class m with the categorical distribution

which is the generalisation of Bernoulli distribution:

$$P(x) = \prod_{m=1}^{M} p_m^{\chi}\,, \tag{3.15}$$

where p_m is the probability of belonging in the m class and χ is a parameter that helps represent the distribution in the same way we do it for the Bernoulli distribution where $M = 2$. It takes values according to m as shown below:

$$\chi = \begin{cases} 1 \ : \ x = m \\ 0 \ : \ x \neq m \end{cases} .$$

Note here that we could also write this distribution as:

$$P(x) = \begin{cases} p_m \ : \ x = m \\ 0 \ : \ x \neq m \end{cases} ,$$

although the reason we use the previous format is to help with the following calculus.

Assuming that we will split our training in batches and for a given batch size of training N the likelihood of a data instance selected for a class at a given batch, will be the multiplication of all individual probabilities because they are independent.

$$l = \prod_{n}^{N} P_n(x) = \prod_{n}^{N} \prod_{m}^{M} p_{nm}^{\chi_n} \tag{3.16}$$

Our goal now is maximise the likelihood and to do that we need to take the derivative of the likelihood over all p_{nm} values. Due to the amount of multiplications needed we will take advantage of the monotonic nature of the logarithmic function. By maximising the negative log-likelihood we are maximising the likelihood. Defining the log-likelihood:

$$\log(l) = \sum_{n}^{N} log\left(\prod_{m}^{M} p_{nm}^{\chi_n}\right) = \sum_{n}^{N}\sum_{m}^{M} \log\left(p_{nm}^{\chi_n}\right) = \sum_{n}^{N}\sum_{m}^{M} \chi_n \log(p_{nm}) \tag{3.17}$$

To define a loss function L out of the the log-likelihood we will attempt to maximise the likelihood for each class. Maximising the likelihood is the same as minimising the negative log-likelihood as we already said, although many to be consistent with the fact that a loss function needs to be minimised we will try to minimise the negative log-likelihood instead. Assuming that we treat all training examples equally, we take the average loss over the training batch and we define the cross-entropy loss as:

$$L = -\frac{1}{N}\sum_{n}^{N}\sum_{m}^{M} \chi_n \log(p_{nm}) \tag{3.18}$$

Let's revise what this expression means. The χ_n parameter symbolises the correct

class, where $p_{nm} = p_n(m)$ is the predicted probability of the algorithm for class m. If we have a very good algorithm that gives very high probabilities to the correct class ($\chi_n = 1\,, p_n(m) \sim 1$) then we will have many parameters in the form $1\log(1) \to 0$, and the loss will be a sum of numbers close to zero. If the the algorithm gives very low probabilities to the correct class ($\chi_n = 1\,, p_n(m) \sim 0$) then the parameters will be in the form $1\log(0) \to -\infty$, which means the loss will quickly become a very large number. Similarly if a high score is given to the wrong class ($\chi_n = 0\,, p_n(m) \sim 1$) the contribution would be zero, although at the same time the correct class will have a low score due to $\sum_{m=1}^{M} p_n(m) = 1$ resulting to a high loss. Finally note how important is that we use log-likelihood and not just likelihood. A big deviation from the correct answer creates a huge loss, while if we didn't have the log, the loss would be proportionate to the deviation.

In the real application of the loss function, each class has its own loss calculated separately as you see below and eventually we add all the individual losses L_m, to acquire the total loss.

$$L_m = -\frac{1}{N}\sum_{n}^{N} \chi_n \log(p_{nm})\,. \tag{3.19}$$

Sometimes we might want to be more accurate in our predictions for one specific class. It is a common situation for different problems that false positives need to be minimised while false negatives are not such a big issue. For example a cancer diagnosing model would create big distress if it was allowed to have even 1% false positives. As we will also see later in chapter 5 one false positive instantly limits the efficiency of our algorithm as we will see later. In such cases we can weight the the loss of each individual class m with a parameter a_m so that it penalises differently the losses of each class:

$$L_m = -\frac{a_m}{N}\sum_{n}^{N} \chi_n \log(p_{nm}), \quad \sum_{m} a_m = 1\,. \tag{3.20}$$

Finally to come to agreement with the symbols commonly used in the literature for categorical cross-entropy [1].

$$L_m = -\frac{1}{N}\sum_{n}^{N} \hat{y}_n \log(y_{nm}) \tag{3.21}$$

Binary cross-entropy is a special case of categorical cross-entropy, where we have only two classes, $M = 2$. Instead of y_{nm} we have y_{n0} and $y_{n1} = 1 - y_n$. Similarly $\hat{y}_n$ can be either 1 or 0. The loss function becomes:

$$L = -\frac{1}{N}\sum_{n}^{N} [\hat{y}_n \log(y_n) + (1 - \hat{y}_n)\log(1 - y_n)] \tag{3.22}$$

[1] $\chi_n = \hat{y}_n$ the correct class score and $p_{nm} = y_{nm}$ the predicted class score

Binary cross-entropy is used when we have a binary classification problem. Our project is a binary classification problem where will try to classify signals embedded in noise from just noise. In our case we try the weighting of the loss to penalise falsely classified noise, as signals.

Connection of Activation function and Loss function

In the previous examples of the categorical and binary cross-entropy the output that is expected from the neurons lies in the interval $[0, 1]$. That means that it would be wrong for example to use ReLU activation as our final activation for a classification case. ReLU can give values above unity and this would go against the "rules" we defined for the cross-entropy loss. Therefore is important to understand the loss function we have before we choose the last activation function in a neural network. Common pairs are shown below:

- Sigmoid - Binary cross-entropy

  Note here that sigmoid return one parameter in (0,1) and binary cross-entropy demands also one parameter because the second is just $y_{n1} = 1 - y_{n0}$.

- Softmax - Categorical cross-entropy

- ReLu - Regression

3.2 From random numbers to features

3.2.1 Loss minimisation - gradient descent

Given any model our goal is train it appropriately so that it can mimic a multi-parameter undefined function and provide good prediction of what that function returns. The way to do this is to update the weights of the model so that the predicted result is as close as possible to the expected one, hence minimise the loss by evaluating on training examples. Assuming we have a training example that went through our model and received a prediction, we can calculate the loss. By calculating the loss we are able to update our weights in a way that the next prediction will be more accurate. This doesn't happen in one step because in a real case algorithm the number of the weights are a lot and not independent. By updating one weight we might change another one in a way that it doesn't help the prediction. The more the weights the higher the dimensionality our optimisation space has.

In the simple case of finding the perfect linear fit of some data we have two weights A and B to describe the function $y = Ax + B$. Using the least squares method we can find the best choices for those parameters. In this case the least squares is our loss function that we want to minimise and given that we have only

two parameters and the loss value, our optimisation space is a surface in a three dimensional space. We try to find the lowest point of this surface.

For this problem there is an analytic solution to find the A and B parameters that correspond to the minimum loss. Although let's assume for a moment that there is not an analytic solution and we need to try different A,B to find the minimum loss. A way to do it is to create a grid of values of A and B, for example 10 for A and 10 for B . Then by calculating the loss for each of those $10 \times 10 = 100$ combinations we select the pair with the minimum loss. This method works but if we had more parameters or dimensions this would be computationally impractical. Assuming a similar grid of 10 different trial values for each parameter, the amount of calculations of loss is 10^D where D is the dimensions/number of parameters. Moreover with more dimensions a number of 10 samples for each parameter will probably become insufficient as the dimensions grow. This is also called the "curse of dimensionality", and practically means that the more dimensions we have the more difficult it becomes to sample the parameter space. We need an method that is not critically affected from the amount of parameters.

A method of this type, whose computational cost is proportional to the number of parameters, is the **gradient descent**. First we select a random point in the parameter space and calculate the loss L and the gradient of the loss ∇L over each parameter. By doing that we know in which direction we need to move in the parameter space to reduce the loss even further. Something that is not defined is how far in that direction we need to go. This is called the learning rate η, and is one of the most important parameters of the training. So to update our parameters, the simplest gradient descend we can use is:

$$\vec{w} \to \vec{w} - \eta \nabla_w L \,. \tag{3.23}$$

Parameters like learning rate η that affect the training and the algorithm as a whole are called **hyper-parameters**. We will encounter this term more often, because many things that define a model structure are hyper-parameters. Gradient descent has has two main attributes, the type of the gradient descent and the algorithm that does the updates of the weight.

The process of choosing an appropriate learning rate is not trivial. Assume we have the case of the two dimensional parameter space and the linear fit above. The parameter space of the loss function will be a convex surface. If we are at some specific point on this surface but not on the minimum we can calculate the divergence on that point. The divergence will direct us towards the area where the loss will be minimised. If we choose a "small" learning rate, the loss will be decrease and the divergence will still point towards the same area. If we use a "large" learning rate there is a chance that we will go to the other side of this "well" where the the new divergence will be pointing to the opposite way. This is what we call "overshooting".

Usually, as we will see below, the learning rate rarely stays the same during training. A good depiction of what we discussed is in figure (3.3).

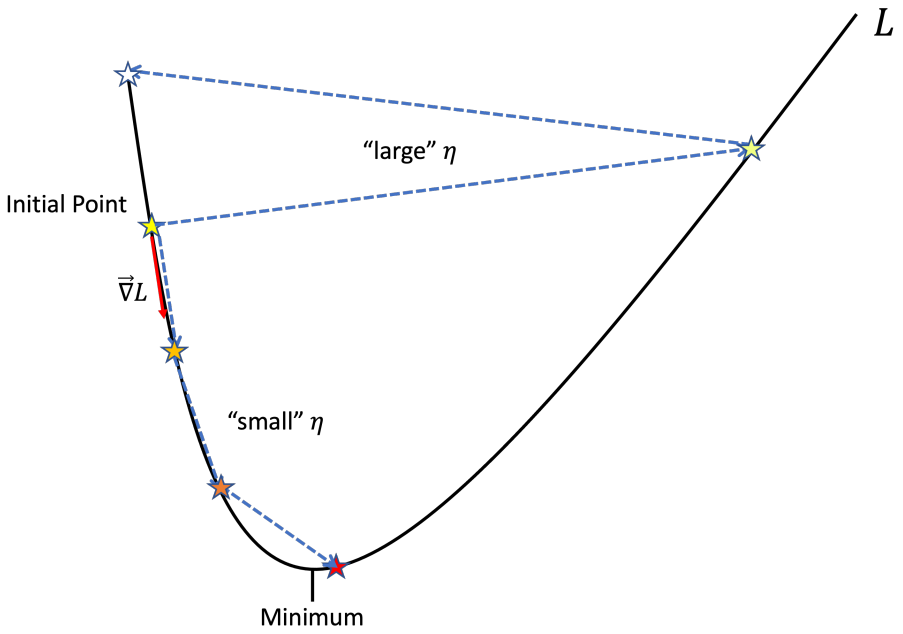

Figure 3.3: Simple example with two cases of different learning rates. We see that if η is rather small we can go close to the minimum, although we won't reach exactly the minimum due to the fact that the learning rate will be considered large for the distance we have left towards the minimum. When η is "large" or larger than we need it to be we actually can go even further from the minimum. Such phenomenon is called "overshooting".

Types of Gradient descent

There are three common types of gradient descent and their main difference is how they treat the training data. Below we will represent the training data as $\vec{X} = (\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots, \vec{x}_N)$, where N is size of the training data.

Stochastic gradient descent - STG When we described in the example above with the line fit, we talked about using one training example to update the weights. This is the main characteristic of the stochastic gradient descent. It is called stochastic because in cases where you have many classes the training example can be from any randomly selected class. This type can make a model train quite quickly but the training can be quite unstable. Each example can make the model get more accurate on that example's class. This is usually used for regression problems and in it's basic form it follows equation (3.23).

Batch gradient descent The exact opposite of STG is to use the whole dataset to train the model. This type is usually limited by the amount of memory we have.

It tends to converge very slow but it is very stable. Again we use equation (3.23), although we calculate an average over the whole dataset of the gradient and then update the parameters.

Mini-batch gradient descent This type can help achieve the golden ratio between stability and speed. Mini-batch gradient descent breaks the training data in batches of size M resulting on $b = N/M$ batches of data. Then it calculates the loss on each mini-batch and updates the weights after every mini-batch. The gradient of the loss used to update the weights is the average of all calculations in a given batch. The equation (3.23) is used again for this but the average of the gradient is on the batch.

Reaching the bottom

Gradient descent seems quite simple as a process. Although the process of actually reaching the lowest loss in the parameter spaces is difficult, mainly because we cannot easily visualise the parameter space. Here we will go through some of the most common issues the gradient descent methods encounter.

The correct learning rate Lets try to imagine again a parameter space with only two parameters, which gives a three dimensional surface like a golf course. Usually parameter space is not smooth and it has many local minima and maxima. Here are some problematic cases during training:

- If we calculate the gradient in a very steep part of the space, the step of the gradient descent will be big and it might jump over the minima.
- If we calculate the gradient in a plateau of the parameter our step will be small if not zero and we will be forever trapped there, especially if we are in multidimensional plateau.
- If we have a big (bigger that we should) learning rate, the step of each update will be bigger than the size of the minimum will are looking for. Therefor even if we get close to it we will overshoot out of it.
- If we have a small (smaller that we should) learning rate, the first minimum that we encounter will be the final one. It is very difficult to escape a local minimum with a small learning rate.

It is apparent from the above cases that the best learning rate for a model can be found. It is also profound though that a fixed learning rate or update is not the best idea. That is why many other algorithms with flexible learning rates have been developed. We will discuss some of them here.

Momentum When we first try to visualise the parameter space we use geographical concepts as hills, ravines and plateaus. Momentum is a gradient descent method that uses that concept. If the descent is in steep point of the parameter space and it heads downwards, in the next evaluation will have already enough speed to go even further, like it has momentum. Similarly if it heads upwards (something we don't want) it will loose momentum and will update the parameters less. The update at each step depends on the previous update, making it a recursive sequence. Defining t as the current update and γ as the parameter controlling the momentum, the sequence is as below:

$$\begin{aligned} \vec{v}_t &= \gamma \vec{v}_{t-1} + \eta \nabla L_(\vec{w}) \\ \vec{w}' &= \vec{w} + \vec{v}_t \end{aligned} \tag{3.24}$$

Note here that this method doesn't change the learning rate but it changes the effect on the updates, which are a linear combination of the previous step and the gradient of the current step. The suggested value of γ is 0.9.

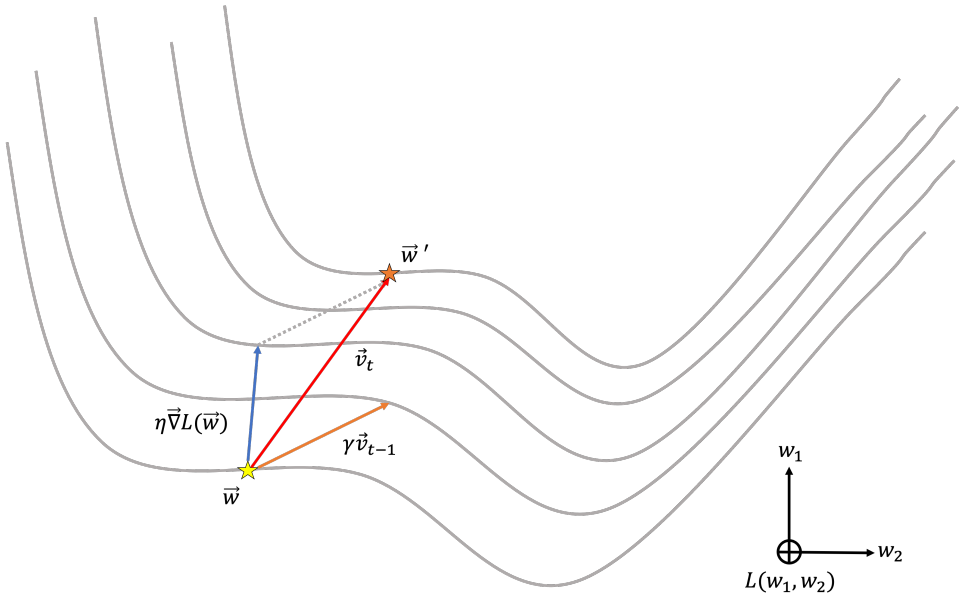

Figure 3.4: The momentum method on a simplified two dimensional loss parameter space. The isolines represent points where the loss has the same value. Instead of having the default update rule that depends on the gradient, the update now is also affected by the previous update $\vec{v}_{t-1}$. The amount of the effect is controlled by γ.

Nesterov momentum Nesterov momentum [17] works like momentum but with the only difference that that it doesn't evaluate the loss on the current point. Instead it uses the momentum to approximate the next point and calculate the gradient there $\nabla L(\vec{w} - \gamma v_{t-1})$. In that way it checks how the gradient behaves in the current direction of the momentum. If the gradient changes direction a lot compared to

the current point, the change is penalised, if it stays similar or tends to align with momentum the change increases.

$$\begin{aligned} \vec{v}_t &= \gamma \vec{v}_{t-1} + \eta \nabla L_(\vec{w} + \gamma \vec{v}_{t-1}) \\ \vec{w}' &= \vec{w} + \vec{v}_t \end{aligned} \tag{3.25}$$

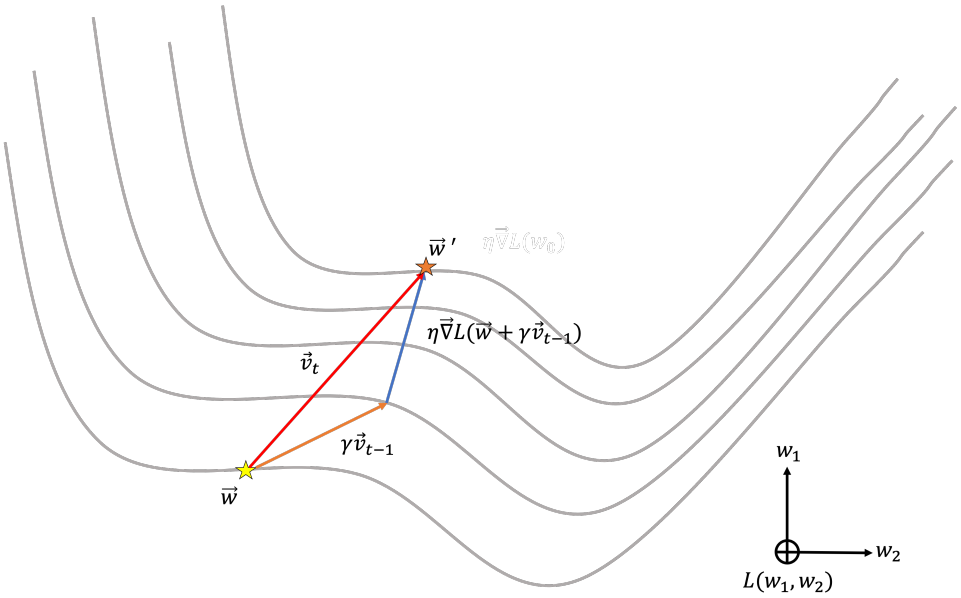

Figure 3.5: The Nesterov momentum method on a simplified two dimensional loss parameter space. Now the gradient is calculated in the point that the previous update directs to. If the directions don't agree the update is reduced and the opposite if they do.

Adagrad The previous two algorithms use information from previous updates in a recursive way to calculate the next one. Adaptive gradient (Adagrad) [18] takes this concept further by updating each parameter based on all the past values and most importantly it uses a different learning rate for each parameter at each time step. The index i refers to the different weights and the index t refers to each state of this weight throughout the training.

$$\eta_{i,t} = \frac{\eta}{\sqrt{G_{i,t} + \epsilon}} , \tag{3.26}$$

where:

$$G_{i,t} = \sum_{k=0}^{t-1} (\nabla L(w_{i,k}))^2 , \tag{3.27}$$

is a squared sum of the past gradients of each weight, η is a fixed value of learning rate and ϵ is just a very small number to avoid division with zero. The update of the parameters will be:

$$w_{i,t} = w_{i,t} + \eta_{i,t} \nabla L(w_{i,t})$$

,

and by using 3.26 we have the final update rule of Adagrad:

$$w_{i,t} = w_{i,t} + \eta \frac{\nabla L(w_{i,t})}{\sqrt{G_{i,t} + \epsilon}} \,. \tag{3.28}$$

As you can see, in the end we don't have to update the learning by hand. It is automatically updated by the past gradients through $G_{i,t}$. Adagrad performs very well in datasets with many important parameters that demand accuracy. Although there is a caveat that due to fact that the summation $G_{i,t}$ increases with every step, the learning rates tends to vanish, and the training to stall.

ADAptive momentum estimation -ADAM ADAM [19] uses decaying average of gradients and at the same time implements a similar technique to momentum. In machine learning we can treat the gradient as a random value due to the unexpected nature of the gradient descent. For any given random value we can define **moments**, which are the expectation values of those random values to a power. If we define g as the gradient, then the mean of g, is the first moment and variance is the second moment.

Adam wants to estimate those moments and use them on the update rule of the weights. The estimation of the mean gradient ($m_{i,t}$) will be used for the gradient and the estimation of the variance of the gradient ($v_{i,t}$) will be used to update the learning rate. Those equations are defined recursively as:

$$m_{i,t} = \beta_1 m_{t-1} + (1 - \beta_1) g_{i,t} \,, \tag{3.29}$$

$$v_{i,t} = \beta_2 v_{t-1} + (1 - \beta_2) g_{i,t}^2 \,, \tag{3.30}$$

where parameters β_1, β_2 are commonly fixed to 0.9 and 0.999 respectively. With the initial condition of $m_{i,0} = 0$ and $v_{i,0} = 0$, we can write equations 3.29, 3.30 as:

$$m_{i,t} = (1 - \beta_1) \sum_{k=1}^{t} \beta^{t-k} g_{i,k} \,, \tag{3.31}$$

$$v_{i,t} = (1 - \beta_2) \sum_{k=1}^{t} \beta_2^{t-k} g_{i,k}^2 \,. \tag{3.32}$$

Our goal was for those two parameters to be approximations of the moments of the gradient. Let's evaluate this on equation 3.31:

$$E[m_{i,t}] = (1-\beta_1)E\left[\sum_{k=1}^{t}\beta_1^{t-k}g_{i,k}\right].$$

Given the random nature of the gradient we can approximate $g_{i,k}$ with $g_{i,t}$ plus an overall error δg added to the equation.

$$E[m_{i,t}] \sim (1-\beta_1)E[g_{i,t}]\sum_{k=1}^{t}\beta_1^{t-k} + \delta g\,. \tag{3.33}$$

From the sum of a finite geometric series we get:

$$\sum_{k=1}\beta_1^{t-k} = \beta_1^t\left(\sum_{k=0}^{t}\left(\frac{1}{\beta_1}\right)^{-1} - 1\right) = \beta_1^t\left(\frac{1-\beta_1^{-1}}{\beta_1 - 1}\right) = \frac{1-\beta_1^t}{1-\beta_1}$$

By using this on equation 3.33 and doing the same procedure for $E[v_{i,t}]$ we get:

$$E[m_{i,t}] \sim g_{i,t}(1-\beta_1^t) + \delta g E[m_{i,t}] \sim g_{i,t}^2(1-\beta_2^t) + \delta g \tag{3.34}$$

This shows that after each update the older the gradient is the less it contributes to the parameters. That is why we consider it to be a biased estimator as it is. To overcome this we will divide by $(1-\beta_1^t)$ and $(1-\beta_2^t)$ respectively to remove this bias.

$$\begin{aligned}\hat{m}_{i,t} &= \frac{m_{i,t}}{1-\beta_1^t}\\ \hat{v}_{i,t} &= \frac{v_{i,t}}{1-\beta_2^t}\end{aligned} \tag{3.35}$$

Finally the parameter update becomes:

$$w_{i,t} = w_{i,t-1} - \hat{m}_{i,t}\frac{\eta}{\sqrt{\hat{v}_{i,t}+\epsilon}} \tag{3.36}$$

where η is the learning rate as we said before. Adam is the optimiser we used mostly in our project so I will restrict my overview of optimisers up to this point.

3.2.2 General structure of neural networks

We talked already about the building blocks of the machine learning process. The neuron, the activation function, and the gradient descent algorithm are all parts we can combine in different ways to train a machine learning algorithm. We need now, to define what a machine learning algorithm actually is. Most of the terms we discussed above are parts used in machine learning algorithms and some of them are considered as hyper-parameters, although we have had no discussion about the structure of the network itself.

Layers

If the neuron is the building block of neural networks, layers are the "beams" of the structure. Layers can have quite different functions from each other. Most of the time this depends on the way they connect to the previous layer (if any) and the next layer (if any). The right combination of layers can organise the neurons in such a way to create a specific method computationally feasible.

All neural networks start with an **input layer**, or more specifically at least one input layer. This type of layer assembles the training data in the way we want the network to receive them. Any type of input goes through an input layer or many of them depending how complicated the structure is. An input layer has no activation function or weights, it is just the data provided.

In the same way neural networks start with an input layer, they always have at least one **output layer**. Again an algorithm might have more than one output layers. Sometimes an output layer might be put in the middle of a structure to "monitor" the in between layers in a large algorithm. Output layers usually are an assembly of neurons in the shape of the output. If the output is N parameters it will be a layer with N neurons, or if the output a timeseries with three channels and size N it will be a two dimensional layer with size $N \times 3$. The important difference from other layers is that the output usually has a special activation function like softmax in classification problems.

Assembly of layers

The simplest assembly of layers is the **sequential** one, where we have one input, some hidden layers and one output layer. The output of each layer is the input of the next layer. This fact creates a compatibility demand between layers. Depending on the type of one layer or the shape of that layer we need to be careful on the choice of structural parameters of the next layer. A simple example is when we have a two dimensional layer and then the output layer is one dimensional. In that case we need to flatten the two dimensional layer, and if it was of shape $N \times M$ now it will be of shape $1 \times (NM)$.

The flattening is actually considered a layer called flattening layer. We call layers such as this formatting layers. Some of them just reshape the input, but most of them are calculating an average or a max and in general change the output and the shape at the same time. We will see some examples of them later in subsection 3.2.5. Those types of layers are usually an add-on on each hidden layer and they are a secondary feature.

Before training all weight parameters have an initial value. The process of choosing an the initial values for the trainable parameters is called **initialisation**. If we chose all weights to be zero before training, we choose a specific point in the parameter space without any reason. It would be theoretically equivalent if we chose

everything to be equal to unity. Although this depends on the problem and the model. The most common and "safe" practice is to initialise all weights by sampling a Gaussian distribution of a certain sigma, usually $\sigma = 1$. The random choice of the initial parameters makes every instance of training give a slightly different result. This can create confusion to us physicists when what we want from a method is to be consistent. The truth is that the power of machine learning lies on that random choice of starting point in parameter space. We will discuss further how we approach those different results in our application.

3.2.3 Fully connected layers and neural networks

Here we will discuss the simplest assembly of neurons, fully connected layers. These layers are also called **dense layers** and a network that is made from dense layers it is called **fully connected network**. Those models are called fully connected because each neuron of a layer is connected with all the neurons of the previous layer. This is the simplest type on network and it is in principle a very powerful tool.Those models are called fully connected because each neuron of a layer is connected with all the neurons of the previous layer. Although due the amount of connections it has between layers it is very computationally expensive to train. Furthermore it tends to have poor performance without any extra add-ons.

The fully connected network is characterised by arrays of neurons that belong in the same dense layer. For example as we present in Figure 3.6 in the group of functions $f(x, y, z), g(x, y, z), h(x, y, z)$, the output layer (f, g, h) is connected with the same pool of inputs in the input layer (x, y, z) but there is no dependency between them. If we make this simple network we have two (3×1) layers, the input and the output layer respectively. Machine learning starts when we decide to add another layer between them. As we said before the layers between input and output are the hidden layers. If for example we have one hidden layer, it is getting input from the input layer (x_i) and its output will be the input for the output layer. So, assume we have a hidden layer h with neurons which have an output value h_i. The value for each neuron will be

$$h_i = A\left(\sum_{k=0}^{3} {}^{i} w_k x_k\right) , \tag{3.37}$$

where A is the activation function of this neuron.The activation function will be the same for all the neurons of the layer. Note also that the summation includes $k = 0$ which is the bias $(w_0 x_0)$. In the general case of a fully connected network with N layers (We assume N plus the input layer, we separated because it is not a trainable layer.) we will have the following recursive equation for the neuron i in the hidden

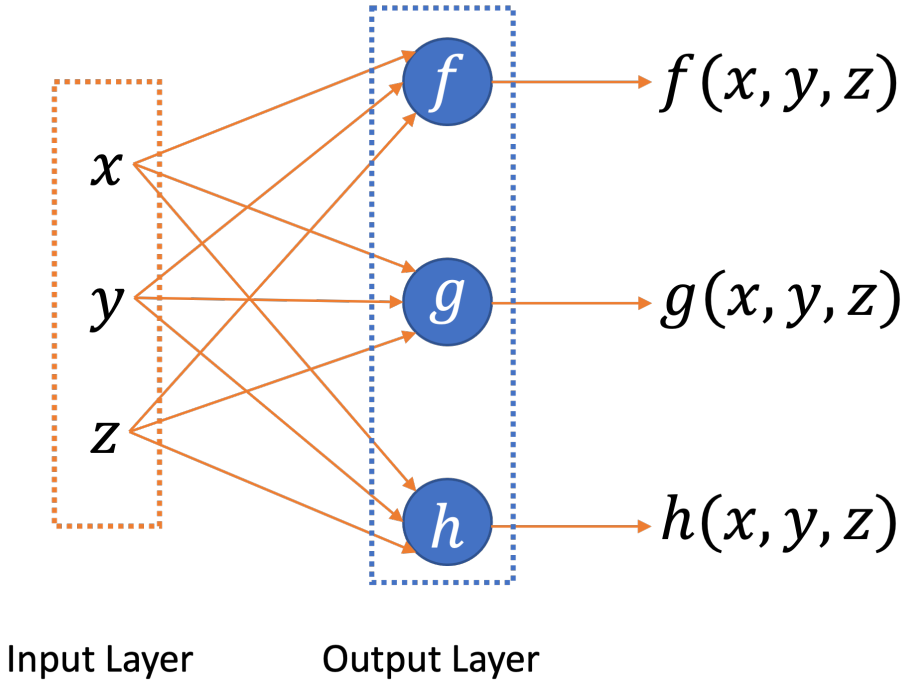

Figure 3.6: The simplest fully connected network with three inputs (x, y, z) and three outputs (f, g, h). As you can see in the output layer, all neurons are connected with all the neurons of the previous layers.

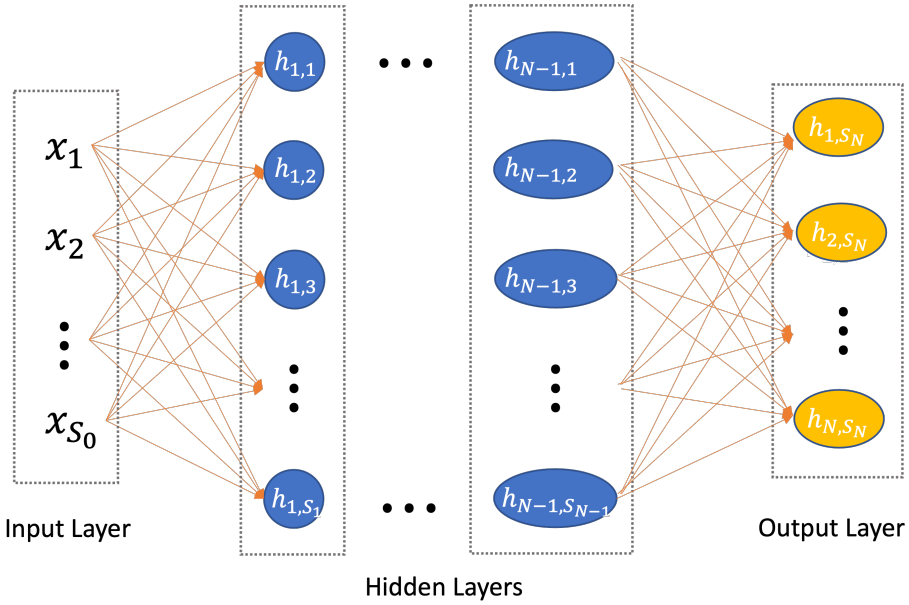

Figure 3.7: A fully connected network, with N layers plus the input layer (layer 0). Again you can see that all neurons of one layer are connected with all the neurons of the previous layer.

layer n:

$$h_{n,i} = A\left(\sum_{k=0}^{S_{n-1}} {}^{i}w_{n,k} \cdot h_{n-1,k}\right) , \tag{3.38}$$

where S_n is the size of the $\mathrm{n_{th}}$ layer and S_0, S_N are the input and output layer sizes respectively. The calculation of the gradient for such a model is quite straight forward or an even better description would be "straight backwards". Similar to the simple neuron the loss will be a linear combination of the weights and input, although this time it will have one more step for every layer. In the output layer, the output of a neuron is $h_{N,i}$ and the loss of that particular value is L_i. We want to calculate the gradient with respect to each weight.

$$\begin{aligned}
\frac{\partial L_i}{\partial w_{n,i,j}} &= \frac{\partial L_i}{\partial h_{N,i}} \frac{\partial h_{N,i}}{\partial\, ^i w_{n,j}} \\
&= \frac{\partial L_i}{\partial h_{N,i}} \frac{\partial h_{N,i}}{\partial A} \frac{\partial A}{\partial\, ^i w_{n,j}} \qquad\qquad \frac{\partial h_{n,i}}{\partial A} = 1 \ , \forall\ n \\
&= \frac{\partial L_i}{\partial h_{N,i}} \sum_{k_1=0}^{S_{N-1}} \left(^i w_{N,k_1} \cdot \frac{\partial h_{N-1,k_1}}{\partial A} \frac{\partial A}{\partial\, ^i w_{n,j}} \right) \\
&= \frac{\partial L_i}{\partial h_{N,i}} \sum_{k_1=0}^{S_{N-1}} \left(^i w_{N,k_1} \cdot \frac{\partial A}{\partial\, ^i w_{n,j}} \right) \\
&= \frac{\partial L_i}{\partial h_{N,i}} \sum_{k_1=0}^{S_{N-1}} \left(^i w_{N,k_1} \cdot \sum_{k_2=0}^{S_{N-2}} \left(^i w_{N-1,k_2} \cdot \frac{\partial A}{\partial\, ^i w_{n,j}} \right) \right) \\
&= \frac{\partial L_i}{\partial h_{N,i}} \sum_{k_1=0}^{S_{N-1}} \left(^i w_{N,k_1} \cdot \sum_{k_2=0}^{S_{N-2}} \left(^i w_{N-1,k_2} \cdot \sum_{k_3=0}^{S_{N-3}} \left(\cdots \sum_{k_{N-1}=0}^{S_1} ^i w_{1,k_{N-1}} \frac{\partial A}{\partial\, ^i w_{n,j}} \cdots \right) \right) \right) \ .
\end{aligned} \tag{3.39}$$

where n refers to the layer, i to the neuron of the layer n and j to the weight j of the neuron i of the layer n.

Depending on the loss function $\frac{\partial L_i}{\partial h_{N,i}}$ will be different, so we have left it as a free parameter. Due to the fact the layers are fully connected, for N layers the number of parameters scales as $\mathcal{O}(N)$. This makes fully connected networks computationally expensive. Moreover it is noticed that as you back propagate the further in you go from $N, N-1, ...n$ you have to multiply with the weights of each layer. There are many cases where a weight might get a very small value or even zero. That prevents the next weight from being updated properly and stays small or zero. In cases where many small weights are multiplied in a row we have the phenomenon of the vanishing gradient. The more layers fully connected a network has, the more prone is the vanishing gradient effect.

It is a very common practice that in the end of any type of model we see a dense layer. The reason for that is that they are good for summarising what features a network learned in an non spacial way, like independent voters.

3.2.4 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) [14] are a revolutionary approach to machine learning that instead of a black box approach that you don't know what exactly your neurons learn, they try to identify features in the data that you can actually verify. Those features are not prefixed but they are inferred by the data themselves. They are very successful when applied on either regression or classification of images or timeseries data. Convolution as an operation needs two inputs. In the one dimensional case the convolution function $c(t)$ is the integral of the product of two functions $h(t), f(t)$, while one of them is "time-shifted", assuming the variable is time.

$$c(t) = (h * f)(t) = \int_{-\infty}^{\infty} h(\tau) f(t - \tau) d\tau \,. \tag{3.40}$$

In this definition $f(t)$ is said to be the "filter" that is applied to $h(t)$ where later we will also refer too as kernel. In our case we want to use discrete convolution of

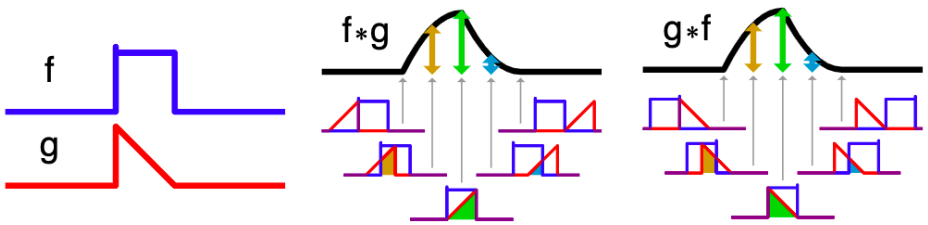

Figure 3.8: Example of two function f, g in continuous convolution.

unitary steps m.

$$c[n] = (h * f)[n] = \sum_{m} h[m] f[n - m] \tag{3.41}$$

By applying to a data series h a filter f with delay n we get the convolution $c[n]$. This process defines a convolutional layer. In convolutional networks we use such layers with small filter functions - kernels. By small we mean that the length of the kernel is much smaller than the length of the layer. Kernels carry all the trainable parameters and their features eventually emerge during training.

Imagine a timeseries $t[n]$ with $N = 1024$ values. We want to figure out if this timeseries has Gaussian like narrow peaks using convolution. So we will define a kernel function that looks like the feature we are looking for:

$$k[n] = [0, 0.1, 0.5, 1.0, 0.5, 0.1, 0.0] \,.$$

By convolving t and k we will get a new timeseries that shows us where the feature of a narrow peak appears. To do that we need to multiply the two functions and get the sum of their elements. The kernel size is $K = 7$, so for every step we need to pad the kernel with zeros from both sides to create the shifting effect.

$$k_m[n] = m \times x[0.0] + [0, 0.1, 0.5, 1.0, 0.5, 0.1, 0.0] + (N - m - 7) \times [0.0] \quad m \in [0, N - K]$$

The final function $c[n]$ will have a reduced size of $1024 - 7 + 1 = 1018$. The main difference from the default convolution is that we don't want to shift our kernel around the edges because the edge points have no feature relation to each other. So instead we end up with an output that has reduced size.

$$c[m] = \sum_{n=0}^{N-1} t[n] k_m[n]$$

or instead of using the zero padding we can just do the convolution locally:

$$c[m] = \sum_{n=0}^{K-1} t[n + m] k[n]$$

Note here that after this calculation, each neuron of the new layer puts the result through an activation function. Here we showed the process for one kernel/filter. The input usually has more than one feature that characterises it and every time we define a convolutional layer we also define the number of those filters which set the number of features we want it to learn during training. In the example above we already defined in advance a feature to be a narrow peak to show how the layer would work. Although when we build a model we initialise all the kernel parameters to random values, which means that in the beginning kernels are looking for random features. This is a common misconception in convolutional network, the assumption that the features are fixed, while instead they are only fixed after the training has finished.

Its time now to show a real example of a convolutional layer. Assume the same case as above with input $X[n]$ of size 1×1024 but this time we will have 4 kernels W_1, W_2, W_3, W_4 with size 1×7 that will do a calculation process and give back 4 new arrays. Our output will have now two dimensions 4×1018 now but only the second will be spatial.

$$C[f, m] = \sum_{n=1}^{K} X[n + m] W_f[n] \,. \tag{3.42}$$

Comparing to the dense layer notation h_i in equation 3.37, we can see that convolutional layers have an extra parameter, the amount of kernels (amount of filters) f. So in a similar notation for a sequential convolutional network we could

write the equation 3.42 as:

$$c_{f,i} = A\left(\sum_{j=1}^{K} x_{j+i}\ {}^{i}w_{[f,j]}\right) . \tag{3.43}$$

In dense layers we have as many weights as input parameters and a bias where in convolutional layers we have as much weights as defined by the the kernel size. Hence CNNs have few parameters to train.

Multi-channel convolutional layers

Following the layer above we saw that the result was two dimensional with only one of the dimensions to be spatial. What if there is another convolutional layer following? Then the next layer will face the previous one as a multi-channel layer, in that case of four channels. The channel dimension is usually called the depth of a layer and the main difference from the spatial dimensions is that the kernel doesn't shift over it, i.e. there is no convolution along the channel dimension.

Lets say now that we have our convolutional layer n, $c_{n,f,i}$, where f indicates the channel and i the neuron of that channel. We want the next layer to have a kernel size K_{n+1} and a number of filters F_{n+1}. The calculation of the output $c_{n+1,f,i}$ will then be:

$$c_{n+1,f,i} = A\left(\sum_{\phi=1}^{F_n}\sum_{j=1}^{K_{n+1}} c_{n,\phi,j+i} w_{n+1,f,j}\right) . \tag{3.44}$$

What it does is convolve on each of the F_n channels and then adds them up to create the output. In the previous first example there was only one channel so the summation over the input channels didn't apply. We see that this sequence creates an output that allows for every layer to have as many filters we want. As we will discuss later in chapter 5 the input of the network a three channel input layer that is fed to convolutional layers. We will not discuss the two dimensional convolutional layers here because they are out of the scope of this project.

Connection with other layers

It is common for the final convolutional network of a CNN to be connected with a dense layer. When this happens we usually have a flattening type of layer that arranges the filters of the last convolutional layer in one dimension. All neurons are arranged in one array ignoring any spacial information between them. By that point the filters of the successive convolutional layers have already down-sampled the input enough that dense layers can handle the information. Although we still use other techniques to avoid that.

3.2.5 Pooling and secondary layers

When we were discussing how machine learning came to be and more specifically CNNs, we saw that Kunihiko Fukushima, along with the convolutional layer developed and down-sampling layer [14]. Those type of layers have evolved over the years and are recognised as pooling layers. What they do is reduce the size of their input data using different methods like the average of inputs or their maximum value.

Average Pooling

Average pooling calculates the average of a subgroup of the input data. As with one dimensional convolution it has a filter that calculates the average of the input inside it's window:

$$p_i = \frac{1}{K} \sum_{j=1}^{K} x_{j+i} \,. \tag{3.45}$$

As with convolution the size of the output will be reduced depending on the size K of the average pooling along with the size of the optional stride (see below).

Max Pooling

Max pooling returns the maximum of a subgroup of the input data. It is much more common than average pooling due to the fact that it has less computations and it effective when it is used properly. It's formula is:

$$p_i = \max(x_{i:i+K}) \,. \tag{3.46}$$

Strides

The two pooling layers we discussed above are reducing the input size proportional to their window. This is not enough for down-sampling. Strides are not a type of layer rather than an extra option of layers with sliding windows, including the convolutional layers. By stride we mean how many steps will the window do for every calculation. The default is unity, while if we have stride two we apply the window for every other element of the array. Assuming a window with size K, an input with size N and a stride S. The output N' of this process will be of size:

$$N' = (N - K) \div S + 1 \,.$$

As you can see sometimes that division is not complete. In practice this means that we might omit input because the filter or the stride are too big. To solve that issue we usually use zero-padding appropriate for each layer to make use of all the input. Then the output will be:

$$N' = (N - K + 2P) \div S + 1 \,,$$

where P is the amount of padding we add on each side. Additionally we might want to keep the size exactly the same for compatibility reasons. To do that we only have to choose the appropriate padding being:

$$P = \frac{1}{2}[S(N-1) - K] \tag{3.47}$$

Those parameters are very important for convolutional neural networks. We need to use the stride with caution because it is very easy to reduce the parameters of a network very quickly when we use stride in consequent layers. Such thing can reduce the training time but also reduce the performance if the parameters are also reduced significantly.

3.3 Machine Learning applied for Gravitational Wave detection

The fields of gravitational-wave astronomy and machine learning have both advanced significantly in recent years. As such, there has been a confluence of research into their combination and a considerable body of work has developed. Artificial neural networks, including CNNs, autoencoders, classifiers and various other architectures, have been applied to a variety of problems within gravitational-wave astronomy.

The first obvious question was: Is it possible that machine learning can outperform match filtering when it comes to CBC detections? The first order approach on this conflict was that machine learning cannot surpass match filter because match filter is the analytical way to find modeled signals. This answer has two major problems. The first is that match filter would have been the best way if the detector noise was Gaussian and stationary. As we will discuss in the last two chapters, detector noise is anything but stationary or Gaussian, so there is a chance machine learning could fill that unpredictable gap where analytical methods fail. The second problem is that a detection is as good as it's false alarm probability. As we will see later in chapter 5, we need a low false alarm rate and the calculation of a low false alarm rate is computationally expensive. After we take in account the above facts, we can change our approach to that machine learning it can actually help boost the detection performance, but even if it doesn't it might make it less computationally expensive if not faster.

After people saw potential using machine learning in gravitational-wave astronomy (and also some peer pressure because many other disciplines were already using it for a long time now) the first proof of concept research came to be. The first obvious model to be created is one that distinguishes CBC signals from noise in Gaussian noise. The paper from Gabbard et al.[9] used a CNN classifier and their goal was to compare the performance of the classifier and matched filter at different false alarm probabilities up to 0.001. Those models were trained and tested on Gaussian

noise. A similar investigation, mostly focused on how the network structure affects the performance has been also developed by George et al. [20]. There were many other similar investigations [21, 22, 23, 24, 20, 25, 26, 27, 28], but most of them were focused on binary black hole or neutron star binary signal detection. It is proven that machine learning can have similar performance with match filtering, although most of the time those algorithms have not generated enough background tests to be able to have a low false alarm rate that limits the confidence of any detections made.

Another group of investigation that started shortly after was the parameter estimation algorithms. One of the most time consuming procedures of the post detection analysis is the calculation of the parameters and their margins of error. Those investigations assumed that there is a CBC signal detected and confirmed and they attempt to find the sky localisation[29, 28], the masses and spins of the initial and final masses or other combination of parameters specifically targeted on known signals [30, 31, 32].

Artificial Neural Networks (ANNs) provided a speed boost into the generation of modeled waveforms as shown to Khan et al. [33]. Not only they increased the speed of generating one waveform out of parameters but made possible, by using GPUs, to generate 10^4 waveforms in 163 ms. This is an excellent example how machine learning can decrease the time of even the analytical methods by helping creating the theoretical templates in much less time.

Machine learning seemed also promising to help reduce the false alarm rates or understanding better the glitch behaviour of the detectors. Gravity Spy [34] and other applications [35, 36] create classifiers of glitches that help us understand how the detectors change over time, and use those classes of glitches to simulate the behaviour of the detectors. LIGO and Virgo detectors have a huge number of auxiliary channels, that monitor different part of the detector and environment variations. If we search through those channels we can see how glitches are correlated to signals in other channels (making them to be of non-astrophysical origin). There have been some projects that try to train algorithms to identify what would we expect to see in the strain based on the auxiliary channels or just identify when there is a signal that is correlated to auxiliary channels [37, 38, 39, 40].

The first BNS signal [41] ever detected was accompanied by a very loud glitch, this brought a question to the surface about how often this might happen and how do separate the glitch from the signal. There have been many attempts to de-noise signals that are contaminated with glitch like non-gaussian signals [42, 43, 44] using auto-encoders.

There has been somewhat less interest in the application of machine learning to the detection of other, as yet undetected, gravitational-wave sources, although there have been a number of studies into the efficacy of using ANNs to detect specific supernovae waveforms In this paper [45] they attempt to use the spectrogram of three

detectors as RGB colours and the really good performance of CNNs in picture recognition. This focuses on phenomenological templates that follow a specific frequency curve and it doesn't generalise to more types. An investigation of the performance of similar models in the future when we will be using the Einstein Telescope was performed by this paper [46], where they have trained and tested models on different core colapse super nova (CCSN) waveforms.

The main issue with machine learning applied in gravitational wave detection pipelines is the false alarm rate. Most projects focus on their efficiencies showing promise but there was not a big effort to reduce the false alarm rate to a significant level. The drive to my project was exactly that. To make a machine learning pipeline that can be a candidate for low latency detection.

Chapter 4

MLy ("emily") pipeline

In this project we are investigating how we can make a machine learning algorithm capable to distinguish unmodelled signals from noise. The performance of the best algorithm can be as good as the dataset that it is trained with. To assure that our datasets and training procedures are consistent, we have put all the tools needed for this project into one library and into one pipeline. MLy is this python library. MLy pipeline is the the combination of the functions of the MLy library that will help use produce our results as we will see in the last two chapters. In this chapter we will describe all the details about the generation process of our datasets. Specifically, we will look at how the appropriate time periods for analysis are determined, how the data is located (or generated, in the case of simulated detector data), how time lags are selected for background studies, and how artificial GW signals ("injections") are simulated for training and sensitivity studies.

4.1 Obtaining detector data

In all our investigations we have to obtain background noise. There are two types of noise currently used - artificial noise and real noise from the detectors. Real noise needs only pre-processing, like whitening and band-passing. Although artificial noise needs more work to be generated. We describe how to obtain each type of noise below.

4.1.1 Artificial detector noise

Artificial detector noise is used for training our models. This might be surprising at first but we will show later, that training with simulated data is a better choice than training with real data. The reason for that is that with simulated data we will be able to generate artificial glitches at a controlled rate, that is higher than the rate at which real glitches occur in real data. It is important to know though that we test our models with real noise, otherwise the models would not be practically useful. Given that all models we create are used and verified on real detector data, we want the

artificial noise to follow the detector noise curves and simulate a stationary version of the real detector noise. In Figure 4.1 we can see the PSD of the theoretical curves of Advanced LIGO and Virgo that we use as a base for the PSD of our generators.

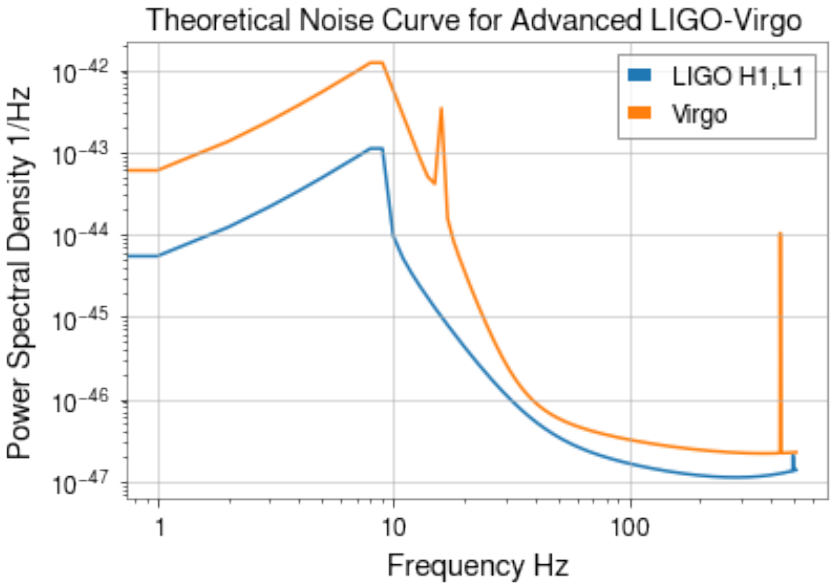

Figure 4.1: Theoretical curves of Advanced LIGO and Virgo as used by the generators

A real detector noise segment will have fluctuations around a mean value. To simulate these fluctuations we generate white Gaussian noise in the frequency domain (real and imaginary parts are generated independently as Gaussian distributed random numbers). The the noise value in each frequency bit k is multiplies by $\sqrt{S[k]}$ where S is the PSD of the theoretical curves. For every generation of noise we do this process to create different instances of the same type of noise. Additionally we use smooth cut-offs on the frequencies we want to exclude from the generated segment. The frequencies outside the two thresholds respectively will exponentially go towards zero. The code for this process was adapted into python from X-Pipeline [47]. Noise will not be further processed depending on it's use. We will talk about the final processing in the end in subsection 4.2.3.

4.1.2 Real detector noise

Real detector noise doesn't need to be generated because it already exists. We only need to find it and load it. To access real detector noise we have to figure out the GPS times where the noise is appropriate for our analysis. GW detectors need to be in a precisely controlled state to operate, in which their optical cavities are in resonance. This is known as "science mode". LIGO and Virgo are able to run stably in science mode for periods of hours to days, with a typical overall duty cycle of around 0.7(70%) [5]. We use gwpy [48] and dqsegdb2 [49] to determine the times

of science mode operation during the observing run of interest.

Not all science mode data is of a quality that allows us to confidently detect GW signals. Background noise fluctuations with durations of order of one second are common. These fluctuations are known as "glitches" and can confuse algorithms searching for GW transients. Glitches may be caused by a variety of environmental disturbances, electronics noise, limitations in the control feedback systems, etc. [50] Many glitches appear in both the $h(t)$ channel and also auxiliary channels used to monitor the detector's environment and control systems. Science mode times that coincide with glitches in selected auxiliary channels are tagged by "data quality flags". Candidate GW events occurring at the time of "not appropriate for analysis", DQ flags are typically discarded. We use the real detector data for false alarm tests and off-line searches and we want to exclude parts of the noise where any type of hardware injection (simulated GW signal added to the feedback control signals for the detector mirrors) is present. Those can be CBC, burst, stochastic, transients or detector characterisation related injections and are characterised by the related data quality flags. Note though that real data also contain some true GW signals. We include the times of real signals in the processing as we want to see if the pipeline can detect them. The dqsegdb2 [49] packages provides the tools to exclude those wanted segments and help us to create new appropriate segments without injections. More specifically we look for all the coincident data when all three detectors (or two depending on the search) are online.

All GW detector data are stored in a particular file format called frame files, with copies in all LSC computing clusters. After getting the segments we want to use, we can use the gwdatafind [51] package to get the locations of the frame files containing the real detector strain data. Although at this point, a computational problem emerges. Some segments are too long to fit in memory, or we might need only a very small fraction of them. We therefore fetch the detector data in intervals that they don't exceed one hour in duration.

Finally, by using gwpy [48], we down-sample the strain data to the appropriate sample frequency we use in our analysis. This is the last stage of getting the real noise ready to be processed to create the dataset. We will discuss the final details in the end in subsection 4.2.3.

4.1.3 Implementing time-lags in the data

In detecting candidate GW signals in low-latency analysis for the generation of alerts, we target false alarm rates of 1 per year or less. The once per year threshold is a requirement for a potential low-latency search pipeline. Confident offline detection of GW signals requires even lower false alarm rates, typically 1 per century or less [52, 5]. A typical LIGO-Virgo observing run will produce less than 1 year of coincident data, which is not sufficient for determining such low false alarm rates.

Therefore we need a method to generate additional independent sets of background (signal-free) data to use in measuring false alarm rates. The solution to that is using time lagged streams of data.

When we test for false alarms, each second of noise in one detector is considered independent of any other second in any other detector (unless there is a real signal of course). We artificially shift the time stamps of the data from each detector relative to the other. The time shift is by an amount that is larger than the light travel time between the defectors and equal to or larger than the time interval used to produce each instance of data that we feed in the neural network, which in our case is one second (in practice multiples of seconds). Then any real data cannot appear in coincidence in the shifted data, so any coincidences we do measure must be due to random coincidence of background noise events. By using multiple time shift values and repeating the analysis for each value we effectively generate many independent copies of the data that are suitable for measuring the properties of the background.

Usually when we apply time-lags we have a lot of data to process. Therefore we separate them in to smaller jobs that process small segments. Time-lags are applied first on big chunks of data creating time-shifts of typically 1024 seconds. Those are called **external** lags (or superlags). One external lag will consist by typically 1024 seconds of data for each detector. If we need more instances of noise then we apply time-shifts of one second for each external-lag. Those types of lags are called **internal** lags (or circular lags). They are both applied in the same way, with the only difference that external lags never use the zero-lag. We will talk more about the implementation in the scheduler functions subsection **??**.

There are some basic rules about time-lags that need to be followed:

- We don't want to include in the false alarm rate any real signals that might be present, because a real signal will not be a false alarm. For that reason we don't use noise at zero lag, which means data as they were collected from the detectors in real time.

- The lag time between two detectors should not be more than an hour or two. Detector behaviour changes with time as they are not stationary. For example for some detectors there are specific ours of the day that are noisier due to human activity (such as traffic or train activity). Therefore if we use two instantiations with a really big difference in time, it might not be a realistic representation of the noise.

- We should exclude times of real known GW events. These are known to be able to produce biases in the false alarm estimation. We could do this in post-processing, vetoing background events where unshifted time is any of the detectors is the time of know GW events.

- Last and most important rule is that we don't want to repeat the same time

Detector 1, lag=0	0	1	2	3	4	5	6
Detector 2 ,lag=1	6	0	1	2	3	4	5
Detector 2 ,lag=2	5	6	0	1	2	3	4
Detector 2 ,lag=3	4	5	6	0	1	2	3
Detector 2 ,lag=4	3	4	5	6	0	1	2
Detector 2 ,lag=5	2	3	4	5	6	0	1
Detector 2 ,lag=6	1	2	3	4	5	6	0

Table 4.1: An example of the order of segments in the case of two detector time-lags. Detector 1 stays the same for every combination and only Detector 2 shifts one second at a time. Note that for Detector 2 lag=7 would be the same with lag=0 and this is not allowed. Any further lags would also be repetitions of other lags.

lag between any pair of detectors for a given data segment, because then they won't be independent samples of the background. That is the most tricky part of this calculation.

The last rule of not repeating pairs is quite simple in this example in two detectors. We leave one detector in zero lag and we move the other one step at a time ignoring the case where lag is zero. If a segment duration has N seconds and we use a step of one second then we will get out $N(N-1)$ time-lagged new instantiations of one second.

The case of three detectors is more complicated. We must not repeat the same pair of any two detectors and to do that we use the following train of thought.

- The first detector is always at zero lag.
- The other two detectors should never repeat the same lag (including zero), because they will repeat the same combinations of instantiations with the first detector.
- The difference of lags between two detectors should never be repeated because it will repeat the same combinations of instantiations between them.

To make this process of lags more understandable I will use lag tables. A simple example of size 5 is the following: Three detectors A, B, C with data segment of duration 5 $\times$ the time lag step size (in our case one second). A is always kept at zero lag by convention. The horizontal and vertical axes indicate possible choices of time lags of B and C. Grey boxes marked X indicate choices that are not allowed because they keep at least one pair of detectors at zero lag with each other. Any other choice is allowed. Although by the time we select a combination we restrict the number of combinations that are left. For example lets follow the selection at the following tables 4.3 step by step. When we select the combination (0,1,4) at the

	4	X				X
	3	X			X	
	2	X		X		
C	1	X	X			
	0	X	X	X	X	X
		0	1	2	3	4
				B		

Table 4.2: Three detectors A, B, C with data segment of duration 5 × the time lag step size (in our case one second). A is always kept at zero lag by convention. The horizontal and vertical axes indicate possible choices of time lags of B and C. Grey boxes marked X indicate choices that are not allowed because they keep at least one pair of detectors at zero lag with each other. Any other choice is allowed.

first table referring to the lags of every detector, we can never use 1 for B or 4 for C because the combination AB-(0,1) and AC-(0,4) will repeat themselves and that is forbidden. So the column and line that selection belongs becomes unavailable. Additionally the diagonal that passes through (1,4) and (0,3) extends to the other side on (4,2) and (3,1) making them also unavailable because the combinations of the same diagonal have the same distance in lags, and this is not allowed to repeat after our selection. Then we choose the combination (0,2,3) and the column and line of that combination become unavailable. The diagonal is already unavailable. We now choose the combination (0,3,2) and the column and line are already unavailable as is the diagonal. Finally we choose the last option left, combination (0,4,1).

Eventually we see that by following the diagonal of that table we get all the possible combinations. Moreover the particular way of doing the lags gives us a simple algorithm, which is that for every step we lag detector B by +1 and detector C by -1. Finally we get that for size N we get again $N(N-1)$ instantiations of data.

It is important to notice here that this method works when the segment size is an odd number of instantiations. When we have an even number things get trickier. If he method we use to do the lags for the odd numbers is used here, we can see that one of the combinations will lie on the B=C diagonal which is forbidden. For that reason we start already with an additional one-step lag offset in one of the detectors, +1 for B for -1 for C. See Table 4.4. If the size N is an even number, then we get $N(N-2)$.

More than 3 detectors The generalisation of this to more than three detectors and for any number of instantiations is a simple but difficult to grasp algorithm from Patrick Sutton []that goes as follows.

Assume you have N instantiations and you want to combine D detectors. We will avoid the term time-lag here because we will just describe the combinations of segments however long in duration.

For each combination of instantiations we want to guaranty that no instantiation

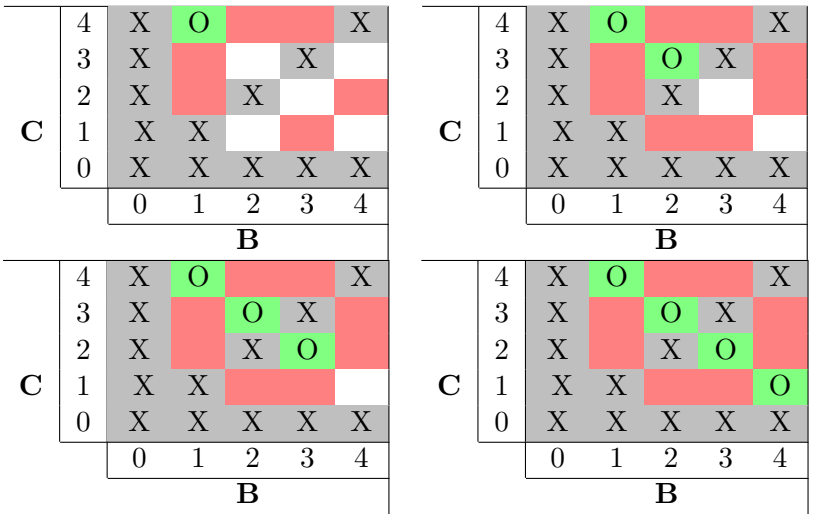

Table 4.3: **(Up-Left)** When we select the combination (0,1,4) at the first table referring to the lags of every detector, we can never use 1 for B or 4 for C because the combination AB-(0,1), AC-(0,4) will repeat themselves and that is forbidden. So the column and line that selection belongs becomes unavailable. Additionally the diagonal that passes through (1,4) and (0,3) extends to the other side on (4,2) and (3,1) making them also unavailable because the combinations of the same diagonal have the same distance in lags, and this is not allowed to repeat after our selection. (Up-Right) We choose the combination (0,2,3) and the column and line of that combination become unavailable. The diagonal is already unavailable. **(Down-Left)** We now choose the combination (0,3,2) and the column and line are already unavailable as is the diagonal.**(Down-Right)** Finally we choose the last option left, combination (0,4,1).

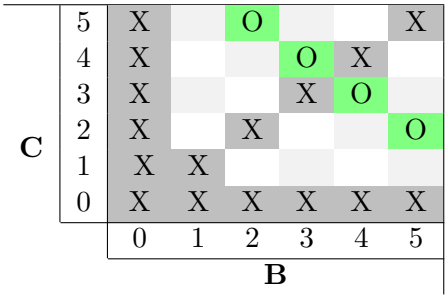

Table 4.4: The case of even number of instantiations create an odd number of maximum lags, hence an asymmetry. We need to move the diagonal method by one and loose one instance.

pairs will repeat in any pair of detectors. This can be achieved by shifting each detector i by $(i-1)$ steps while at the same time making sure that N is appropriate. By appropriate we mean that the N instantiations don't allow any of the shifts to "synchronise" resulting to repeating pairs. More specifically, since each detector uses different multiple of the step, the lag between each pair of detectors increases with each step. The total shifts $i-1$ for each detector i should satisfy $N/(i-1) \geq 1 for i \in [1, D]]$, where D is the number of detectors. That means that N should be

at least the lowest common multiple between the shifts $i-1 for i \in [1, D]$. Although, as we shown in the simple case of three detectors, having an even number for N results to repetitions. Furthermore, if we can use the lowest common multiple of N, let's name it m, and then add 1 to have an appropriate number of instances, we will still have an appropriate number for N if we use any multiple of m and then add 1. In conclusion given the number of instances we have N we have to choose an appropriate number N_{new}, where $N_{new} = M + 1$, where M is the largest multiple of the lowest common multiple of integer numbers between $[1, (D-1)]$ such that $M + 1 \leq N$. If N_{new} is defined as the largest common multiple of all the step sizes plus 1, no detector repeats a previously used shift upon "wrapping" by modulo N_new. This prevents repetition of any lag between any pair.

Let's check some examples about that. Assuming three detectors $D = 3$ and $N = 128$, the lowest common multiple in $[1, 2]$ is 2. So the largest multiple of 2 for which if we add one, it will be ≤ 128 is $126 = 2 \times 63$. So $N_{new} = 127$. If we do the same but for $D = 4$ the lowest common multiple between $[1, 2, 3]$ is 6 and the largest multiple of 6 that if we add one will be ≤ 128 is $126 = 6 \times 21$, so again $N_{new} = 127$. The bigger the number of detectors, the biggest the minimum N_new we need to do the shifts.

When we apply this on a segment of data that has N instances and data from D number of detectors, we can create a shifting matrix. Each number on this matrix represent the amount of steps a segment is shifted. To create this matrix, we apply the train of thought we describe above. We use the cross product of the array of detectors D^T and the array $(0, 1, 2, N_{new})$. The first row of this matrix will be zeros because we don't apply shifts on the first detector. The row will be one step at a time, the third will be two steps at a time. Eventually the shifts will become much greater than the number of instances used (N_{new}) which will still be valid but unnecessary. Shifting an array of size 10 by 104 steps is the same of shifting just 4 steps. For that reason we apply modulo N_{new} at the end to avoid such redundancy.

$$\mathbb{L} = \left(\begin{pmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ D-1 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 2 & \cdots N_{new} - 1 \end{pmatrix} \right) \mod N_{new} \tag{4.1}$$

Let's see an example of the shift matrix in the case of four detectors ($D = 4$) and $N = 128$. We can't use N as it is, we need to find an appropriate N which as we shown above it will be $N_{new} = 127$:

$$\mathbb{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & \dots & 124 & 125 & 126 \\ 0 & 2 & 4 & 6 & 8 & 10 & \dots & 121 & 123 & 125 \\ 0 & 3 & 6 & 9 & 12 & 15 & \dots & 118 & 121 & 124 \end{bmatrix}$$

Each row represents a detector and each column the lags in respect of the first detector.

Finally we need to need to note here that this procedure is independent of the duration of the segment or the instances. These are to be derived depending on the problem.

4.2 Injection generation

Our goal is to be able to detect tranisents of GW signals without prior knwoledge of their waveform, and distinguish them from background noise glitches (also potentially of unknown waveforms) that occur simultaneously or near-simultaneously in two or more detectors. The complexity of this task demanded a plethora of signal types and ways of injections. In this section we will discuss the main waveform morphologies we will use and why we use them, then in subsection **??** we will explain the different ways of creating injections and projecting them in the the detectors, and finally in subsection 4.2.3 we will talk about the final assembly of noise and injection and how we create the different types of data to simulate generic signals and glitches.

4.2.1 Gravitational-Wave morphologies

The main signal type we used in our project was the White Noise Burst (WNB). This type was used for all training, although we generated other types of signals to test with, such as Compact Binary Coalescence (CBC), Circularly polarised Sine Gaussian (CSGs), Core Collapse SuperNova (CCSN) and Cusp signals. All of those signals were of maximum duration of one second and all injections have two polarisations components h_+ and $h_\times$, with the only exception of CUSPs which are linearly polarised and have only h_+ component.

WNBs are a type of signal that attempts to be as random and unstructured within its parameters as possible. To create a WNB we start by generating white noise on the frequency domain with two frequencies $[f_{min}, f_{max}]$. They are described by a central frequency and a frequency range $f_0, \delta f$ where $f_{min} = f_0 - \delta f/2$ and $f_{max} = f_0 + \delta f/2$. Then we inverse Fourier transform to get the injection in a time-series format and we apply a sigmoid window on both edges of the signal to make it non-zero only over the desired time interval The steepness of the sigmoid window

depends on the duration of the injection. Both sigmoids on the sides go from 0.01 to 0.99 and in reverse, using 2.5% of the signal duration on each side. We desire the final signal to have two independent polarisations, so we generate two independent WNBs with the same parameters. A physical interpretation of this is a source in which matter is undergoing random motion over some range of length/time scales, such as turbulent flow. WNBs are the closest type of signal to represent generic signals.

CBCs are compact binary coalescence signals generated with IMRPhenomD [53]. These represent the inspiral, merger and ringdown of binaries consisting of black holes and/or neutron stars. We restricted the generation only to binary black hole signals with the black-hole masses and spins selected randomly and uniformly over the intervals [10,100] and [-1,1] respectively, with the restriction that the signal maximum frequency has to stay below the Nyquist frequency of the dataset. The two different polarisations also assume a random angle of inclination. We use this type of signals because they are the only type of GW signal detected to dated, so ti makes sense to test the pipeline sensitivity to this type. We don't use them for training.

CSGs are an *ad hoc* but standard waveform for testing GWB analyses [7, 5]. They are sinusoidal signals of a fixed frequency in a Gaussian envelope. Each polarisation has sine and cosine oscillation respectively with the $h_\times$ having also an optional ellipticity parameter. Here is the waveform analytically:

$$h_+ = h_0 \cos(2\pi f_0 t + \phi) e^{-\left(\frac{t-T/2}{T/\sigma}\right)^2}$$

$$h_\times = h_0 \sin(2\pi f_0 t + \phi) e^{-\left(\frac{t-T/2}{T/\sigma}\right)^2} \sqrt{(1-\epsilon^2)}$$

Where t is a time array from 0 to T with step one over the sample frequency, f_0 the frequency, ϕ the phase offset and σ is the number of standard deviations we want to be included in both sides of the Gaussian envelope. The default is 5. The parameters f, T, ϕ, ϵ are all randomised within $[20, 480]$ Hz, $[0.0625, 0.9]$ sec, $[0, 2\pi)$ and $[0, 1]$ respectively.

CCSN (Core colapse super nova) is the most popular type of burst type we expect to see, as it is the next expected type of signal to be detected with an electromagnetic counterpart. There are many CCSN simulations that return the two polarisations of the expected signal. In our project we used only the N20-2 waveform of [54], from a 3D simulation of a neutrino-driven core-collapse supernova (CCSN) explosion for testing. We only use that one waveform because we are doing an early study of the type and not comprehensive study of CCSN detectability. Hence one waveform is

enough for a sanity check. This specific waveform is used in previous LIGO searches [7, 5, 4]

Cusp signals are the type of signals that are expected from cosmic string cusps.[**?**, 55, 56] These signals have a rather simple equation:

$$h(t) = A|t - t_*|^{1/3},$$

where A includes physical parameters regarding the strings and t_* is the peak time of the signal. Cusp signals are linearly polarised.

4.2.2 Computing the detector response

The next step is the projection of the injection waveforms, where we project the waveforms as they come from the source to the detectors and we calculate the strain data that will appear in the detectors. The waveform that will appear in a detector is a combination of two polarisations with weighting parameters F_+ and $F_\times$. For a given detector we have the following signal as a response [57]:

$$d = F_+ h_+ + F_\times h_\times \,, \tag{4.2}$$

The antenna response parameters F_+ and $F_\times$, depend on the incident direction and polarisation angle to the GW with respect to the detector, see [57] for explicit formulae. To simplify the calculation and make sure we distribute our injections uniformly throughout the sky, we fix the time (practically freeze the earth rotation) and generate random spherical polar coordinates of the for each generated waveform. Additionally we randomise the polarisation angle of the incoming signal so that there is no preference of polarisation during the training.

After the projection is done the initial two polarisation components of the waveform h_+ and $h_\times$ will transform to D projected waveforms where D the number of detectors we projected to. A very important detail we need to add is the arrival time difference of the detectors. The signal will arrive to each detector with a difference of some milliseconds, so we need to time-shift the projected signals in respect to each other. Assuming that the signal comes from direction $\hat{\Omega}$, the time delay will be [47]:

$$\Delta t_\alpha(\hat{\Omega}) = \frac{1}{c}(\vec{r_0} - \vec{r_\alpha})\hat{\Omega} \,, \tag{4.3}$$

where α is a specific detector and $\vec{r_0}$ a reference point, which we select to be the Hanford detector. In combination to the antenna response, we have that for a given detector the projected signal will be described as [47]:

$$d_\alpha(t + \Delta t_\alpha(\hat{\Omega})) = F_\alpha^+ h_+(t) + F_\alpha^\times h_\times(t) + n_\alpha(t + \Delta t_\alpha(\hat{\Omega})) \,. \tag{4.4}$$

In the case where the signal duration is quite close to the duration of the injection duration (noise and signal, in our case one second) we might have edge effects after shifting. To resolve this we pad with zeros equal to the maximum time delay possible calculated based on our sample frequency f_s. Then we calculate the Fourier transform of the waveform and add a phase that will create this time-shift in the time domain.

$$\tilde{d_\alpha}(f) \rightarrow \tilde{d_\alpha}(f) e^{-i2\pi f \delta t_\alpha} \tag{4.5}$$

We follow this approach because if the time delay is not an exact multiple of the time step in the time domain, there will be side effects in higher frequencies that will add extra noise. To do this calculation shown in equation 4.5 we use the first detector as reference detector, where we don't apply the shift (δt_1) and δt_α is calculated in respect to that detector. After the shift we crop the padding. The waveform will not stay at the center of the injection. In the next subsection, we will describe how we randomize the central times of the waveforms when possible.

4.2.3 Final processing of the data

In the previous subsections we detailed how we generate noise samples and injections. Now we will explain how we combine these into the the final instances of our datasets.

When we first started experimenting with machine learning and data types, the first obvious choice was to have noise, and injected signals as we expect them to appear in the detectors. We found that CNNs trained only on real detector noise and simulated signals performed poorly, and performance was improved by creating simulated glitch populations as well. In this subsection we will describe the procedure and go through them one by one.

Coherent Signal Injections These are the gravitational-wave signals as we expect to see them when they arrive in the detectors. Depending of what metric of intensity we use we change their amplitude in a different way. As we described before we can have a target SNR ρ or a target h_{rss} value h, that we want the signal to have. In both cases we pad first the injection s_{0_α} with zeros equally from both sides to have the same length as the noise segment n_α and we add it to the noise.

$$d_{0_\alpha} = s_{0_\alpha} + n_\alpha \,. \tag{4.6}$$

The zero indexed values indicate the signal or injection before re-scaling. In the case of a target SNR value ρ we have to calculate the PSD $S(f)_\alpha$ of the noise and then calculate the current SNR value ρ_{0_α} the signal has on each detector α using equation 2.49. The initial network SNR will be:

$$\rho_0 = \sqrt{\sum_{\alpha=1}^{D} \rho_{0_\alpha}^2} \tag{4.7}$$

Then we re-scale the amplitude of the signal using the target and the initial SNRs using:

$$d_\alpha = \frac{\rho}{\rho_{0_\alpha}} s_{0_\alpha} + n_\alpha \,.$$

In the case of a target h_{rss} value h we don't have to calculate a PSD. We simply calculate the initial RSS ampitued h_0 from the injection using equation (2.48) and then re-scale in the same way:

$$d_D = \frac{h}{h_0} s_{0_D} + n_D \,.$$

Note here that the h_{rss} value is independent of the detectors.

The signals we inject have different durations, from 0.0625s to nearly 1s. We don't use signals of exactly 1 second because they would be cropped when we apply time delays between detectors. The rest of the interval will be zero and we take advantage of that to move the injection around. Having an injection always at the center would easily make a machine learning algorithm over-fit and try to find signals only in the center.

Let us denote the maximum time delay between detectors by ΔT. The maximum time delay depends on which detectors we use (based on their distance on the globe). Given the duration T_s of a signal, the duration of the instance T we can shift the injections in all detectors by a random value from the interval $[-\Delta T/2, \Delta T/2]$, without any danger of the injection being cropped by the edges of the data instance. We perform these shifts independently for each injection.

We treat BBH signals as a special case, given that the beginning of the signal is not clearly defined (only based on the minimum frequency the detector can see), we usually have signals that already have duration equal or bigger to the instance duration. Although we also need to move the loudest part of the signal around without any important information. For this reason the shift is allowed to be only backwards $[-(\Delta T)/2, 0]$, so that the merger appears always within the second half of the instance (in our case 1s).

Whitening and Bandpassing All instances have a basis of noise, artificially made or real. The loaded or generated noise segment is 16 times the duration of the final instance in the beginning. The reason of such length of the segment is that it will be used to calculate a power spectral density (PSD). When a signal is present, it might contribute to the PSD. The larger the segment of noise we use compared to the signal the smaller the contribution to the PSD it will have. We choose to

use a background of duration 16 times the instance duration. This is because it is big enough to suppress that contribution in the PSD and small enough so that it is not computationally expensive. There is not a correct option for that of course, we just make a balanced choice between efficiency and time. We suggest that an investigation about the contribution of the noise segment size would be very useful addition to the analysis.

After the noise and (optional) injection are combined we need to finalise the instance to its size. First we use the whole interval (16 second in our case) of combined signal and noise to calculate a PSD. Then we divide the Fourier transform of the instance with the PSD to normalise the frequency levels. This will make the signal more apparent. This will then be a whitened timeseries. Additionally we apply an IIR band-passing filter to crop all the frequencies below 20 Hz. Finally we crop and keep the central second of our segment which is the final form of the data instance.

4.2.4 Simulating glitches

The use of simulated waveforms for glitches is a really important point that the reader should elaborate (it is an unusual choice). we use simulated glitches for two reasons. First, real glitches don't occur in coincidence frequently enough to be useful for training - if we used real data only a small fraction of our training samples would contain coincident glitches and the CNN would not learn to reject them. We would instead use another algorithm (such as Gravity Spy [34] to identify times that contain glitches and preferentially select those fore training samples to get a good enough ratio of signals vs glitches so that they have an impact on the training. But then there is a danger that we are training the CNN to recognise and reject only those specific glitch mythologies that are know by Gravity Spy. This may leave us vulnerable to as-yet unclassified/unknown glitches - exactly the same danger as using specific signal morphologies for gravitational wave bursts (GWBs). What we prefer to do is to find a way to force the CNN to learn how to distinguish coherent events from incoherent events, rather than classifying morphologies. By coherent event we mean a signal that appears in more than one detectors and has the same morphology (projected differently), and incoherent signal would be a case of signals appearing in more than one detectors and being of a different morphology. To focus on the coherency information during training, we use the same WNB waveform sets for both GWB and glitch simulations. The CNN cannot rely on the signal morphology for tits classification, only on the coherence between detectors.

Single detector glitches To simulate the glitchy behaviour of one detector, we first choose a detector which we call the "lucky" detector. We follow the same procedure with coherent injections but we inject the projected waveform only into the lucky detector. The target SNR or h_{rss} goes directly to that lucky detector.

All other detectors remain as they are without injection. In this way we simulate the common case of single-detector glitches. As with coherent signals we apply a time-shift to the signal so the glitches are not always centered in the analysis interval.

Multi-detector glitches Random coincidence of glitches in multiple detectors constitute a challenge to the confident detection of GWBs. We force our model to learn to handle this scenario by injecting incoherent glitches into the data streams of each detector. The incoherent injections are a type of dataset that is used to confuse any model by introducing different signals in all detectors to make present the idea of uncorrelated coincident signals to the model training. We do this by selecting the WNB parameters independently for each detector. No antenna response projection is required as the glitch is not an astrophysical signal. We simply normalise the total SNR to the desired value.

Regarding the time-shifts, in this case if we have D number of detectors then we will have D different injections with D different durations each. Glitches are not correlated between detectors, so we have to train the analysis for the case where transients are present in multiple detectors in a given analysis segment (instance, 1 second), but not necessarily coincident to within the light travel time. Given that they are simulated and are used to complicate the noise representation, the more complex and randomly placed they are the better. Injections in each detector will have a random shifting on their central times. This shifting will have a range that will depend on the size of each individual waveform. To help with this we define a value called **disposition** $\mathcal{D}$ which is the maximum difference between the two central times of the glitches in any pair of detectors.

When we define a disposition, depending on the number of the detectors D, central times are created with which we shift our signals. Assuming duration of instance $T = 1$s we can have the following central times for each signal given the number of detectors D.(Note the difference in symbols D: number of detectors, $\mathcal{D}$: disposition (seconds)).

- $D = 2 : [(T - \mathcal{D})/2\ ,\ (T + \mathcal{D})/2]$
- $D = 3 : [(T - \mathcal{D})/2\ , T/2,\ (T + \mathcal{D})/2]$
- $D = 4 : [(T - \mathcal{D})/2,\ (T - \mathcal{D}/2)/2,\ (T + \mathcal{D}/2)/2,\ (T + \mathcal{D})/2]$

Therefore, the central times of the glitches are equally spaced in time over an interval of length $\mathcal{D}$, which is centered on the middle of the analysis interval. A central time shift is given randomly to each detector until all of them are allocated. Additionally we also add a random variation of maximum 5% of $\mathcal{D}$ on each central time to make sure the positioning retain a randomness, independent for each detector. A valid question would be why not just randomly shift the glitches instead of using disposition. This could also be done, but then we would have no control over how

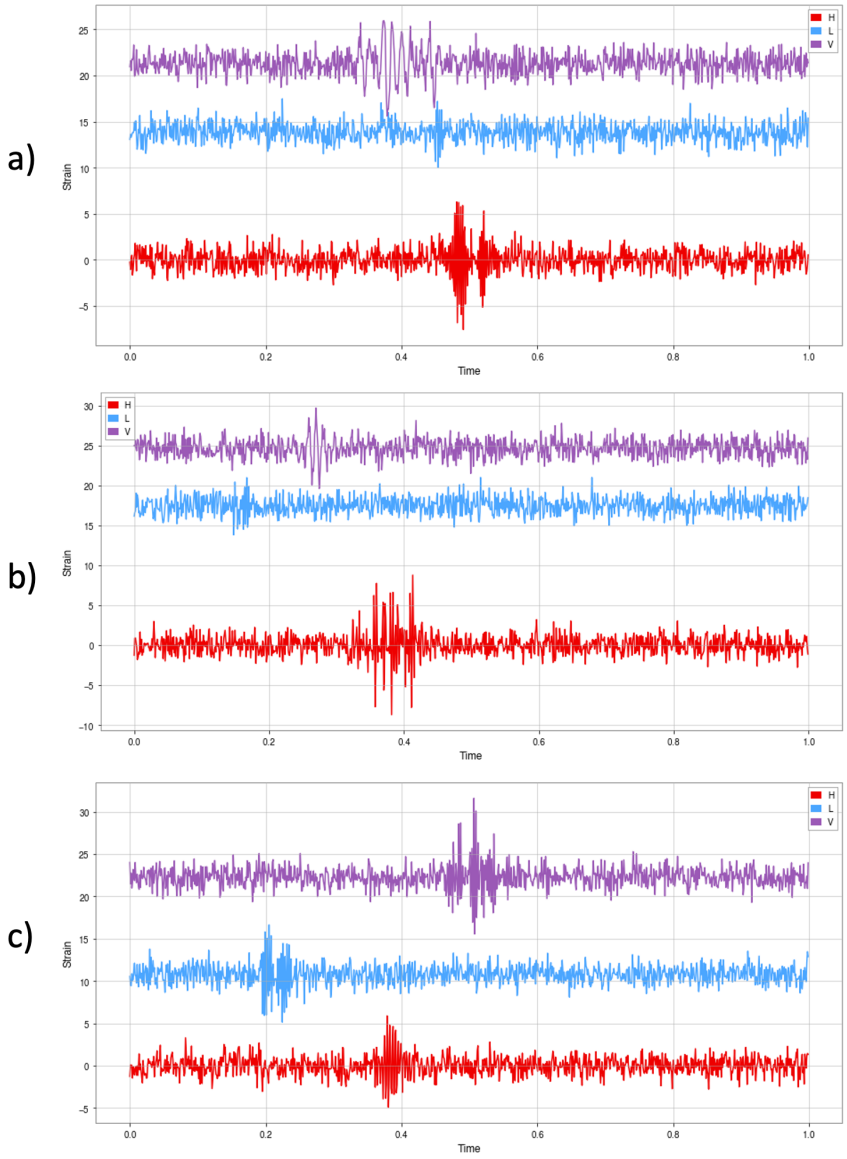

Figure 4.2: Examples of disposition between different glitch signals. We present the timeseries for each detector with an offset that depends of the intensity of each signal so that they are distinguishable. a) Disposition of 0.1 s. You can see that the central time of Virgo signal (magenta) is 0.1 s away from the central time of the LIGO H1 signal (red). b) Disposition of 0.2 s. c) Disposition of 0.3 s.

simultaneous the glitches are. Having the central times of the glitches very close or at the same place poses a much bigger challenge for the network and forces it to understand coherency. By using disposition we can control the distance between the central times of the signals and create datasets that will focus the training into coherency identification.

When we use disposition with a bank of waveforms with duration within one second, we might come into conflict. For example if the glitches we use are of duration 0.7 seconds, the disposition cannot be more than 0.3 seconds, because the

signals could get cropped. Therefore, when we generate such datasets we restrict the duration of the injections used based on the disposition we want to have. So for example if we choose a disposition of $\mathcal{D} = 0.4s$ the maximum duration of the signals is chosen to be $T_s(max) = 0.6s$.

After this is done and given that there is still enough room to shift the glitches around we apply also a common random shift to all glitches central times (together) to avoid always having the group of glitches centered in the interval. We present three examples of disposition in figure (4.2).

4.3 Main pipeline tools

4.3.1 DataPod and DataSet

Our application as already described has a lot of different terms and attributes for each data instance or injection: duration of segment, duration of injection, number of detectors, SNR for each detector, type of noise, sample frequency, disposition. These are only some of the parameters that need to describe our datasets. It is not uncommon that an unusual result demands investigation and to do that we need all those metadata. Therefore we need to make it so that each data instance carries it's metadata with it. This object we called a **DataPod**. DataPods carry with them the main data used in a data instance and also all the important details of it's history and creation. For example the waveforms that are used for injections carry information about their generation by being inside a DataPod. By the time we add a waveform on the noise background to make an injection, it is impossible to calculate it's h_{rss}. DataPods helps preserve this information for us. When the waveform DataPod is used and added to a noise DataPod, it creates an injection DataPod that inherits information from the waveform (type, duration, h_{rss}) and the noise segment (detector, GPS time etc.). Additionally it can create new information out of this combination such as the SNR value and stop conflicts of information which prevents bugs, like having zeros in the timeseries or signals cropped due to lack of information about the waveform duration.

A collection of DataPods with the same data type is called a **DataSet**. This object is an organising tool that helps us filter information out of the DataPods collectively. For example when we use those data for training, the DataSet can export the data in a shape appropriate for the algorithm to digest and also any other parameter in the same order, such as labels. That organisation prevents mix-up of the order of data and true values. DataSets can be fused into one when we combine different DataSets with different parameters and can check themselves for any inconsistencies that could create bugs. Their collection structure makes it possible to apply modifications to each DataPod of a DataSet at the same time. Our training datasets are DataSet objects.

Generator function

In the previous sections we described in detail how we combine noise and injections to create different data instances. The process of this generation is the backbone of this pipeline. The main function that is used for creating DataSets for training and testing is called the **Generator** function. The generator is used in all processes we describe below and has many parameters and keywords to provide a plurality of types of datasets. The output of the generator function is a DataSet object.

4.3.2 PlugIns

As we will see later, in our project we use both strain data and also the Pearson correlation between detectors as inputs to the algorithm. Correlation data were calculated from the strain data. This gave us an idea of developing an object that acts externally on DataPods and DataSets. Those objects we called **PlugIns**.

A PlugIn consists of a function plus a list of attributes that already exist in a DataPod. For example the strain data, the sample frequency and the number of detectors are attributes already existing in a DataPod. In the case of the correlation, the function will return the Pearson correlation for each pair of detectors and add it as an attribute to the DataPod so that it is callable as the strain data are. Again we don't need a DataPod to define a PlugIn but we need to know the attribute names that we expect the PlugIn to use from the DataPod. Assume we have a DataSet that has only strain data in it. We can use the already defined PlugIn for the correlation and add it to the DataSet. Now the DataSet will be able not only to export strain data but also correlation data.

PlugIns don't need to be functions, they can also be simple values. Another example of use for PlugIns could be sky-localisation parameters. The source of an injection can be described from right ascension and declination, which will be parameters accompanying any DataPod that has an injection in noise. If we had an algorithm that estimates the sky position of a signal, we could use PlugIns to define our metric of location in the sky (numbered tiles for example) without having to hard-code it in the generator.

Plugins give a flexibility that could make any generated DataSet evolve accordingly to the type of problem and model is going to be used for.

4.3.3 Validation functions

As we will see in chapters 5 and 6, after a model is trained we need to test it to verify how well it performs. For that procedure we have three main validation functions, **falseAlarmTest**, **efficiencyTest** and **zeroLagTest**. Each of these functions validate a different aspect of performance of each model, by using the same base tool, the generator function.

FalseAlarmTest

This function tests the models for false positives. In the case of real data, it takes a number of GPS time ranges equal to the number of detectors. For a false alarm test we don't want to use the zero-lag data because they might have real signals present. So it is expected that those GPS time segments are from different times. Also it takes as an argument, the specified number of tests N to be performed, where one test is analysis of 1 second of data. If the size of tests N is bigger than the GPS interval can provide, we use internal time lags to expand the amount of test performed, using the duration as a step (in our case 1 second). It uses the generation function to create a DataSet of processed real noise and then feeds that DataSet into the models and saves the output of the model for each instance into a table along with the GPS times for each detector.

EfficiencyTest

This function calculates the true positives, for a specific class or parameter, that are predicted from the models out of a number of N trials for different M values of intensity. As values of intensity we use either different SNR values or h_{rss} as we describe them in subsection 2.3. The main parameters given to this function are the waveform bank list that has different waveforms of the type we want to test, a list of values for the intensity of the signal and the interval of GPS times to use for background noise. For each intensity value m it uses a randomly selected waveform from the list and creates an injection with that intensity. It repeats the same procedure using the generator function N times for each m intensity value. Then it saves all the scores in a dictionary with the intensity value as a key. It is up to the user then to select a threshold (corresponding to a specific false alarm rate) above which we consider an injection to be detected or not. This function provides the scores and leaves it to us to use the trials as we see fit to calculate an efficiency value for the type of waveform tested.

ZeroLagTest

This function is used to create an offline search. It follows the same procedure as falseAlarmTest, except that all segments are in zero-lag, as we can see them in the detectors. Furthermore an important additional parameter that we describe more in the offline search Chapter 6 is the strides. Note that those are not the same strides that are used in CNN searches. Strides help us "scan" the data with more detail by using overlapping segments. When stride S is defined, the function runs the same generator function S times. Each time it adds a small offset to the GPS time equal to d/S, with d being the duration of the instance (in our case 1 second). All S DataSets are then fused and fed to the models as described above. Again the way we will process the results depends on what we will consider a detection. In practice

we find that injections tend to produce high model scores over multiple consecutive strides. Therefore, given the threshold, for a detection to occur we need the model to fire in at least two consecutive strides to consider a candidate. This helps us avoid single high false positives that are a result of the imperfections of the model.

Chapter 5

Real-time detection of Unmodelled Gravitational wave transients using CNNs

This chapter is adapted from a submitted paper, by me, Michael Norman and Patrick Sutton. I worked on the pipeline creation and the analysis of the results, Michael Norman worked on the creation of the genetic algorithm that helped choose the correct model and Patrick Sutton was supervising the project offering direction and corrections.

5.1 Introduction

Gravitational-wave (GW) astronomy is now an established field of observational science. To date, the LIGO [58] and VIRGO[59] collaborations have published the details of 90 detection candidates [52, 60, 61, 62, 63, 64, 65], over the first three observing runs. The detected signals originate from the binary inspiral and merger of two black holes [66], two neutron stars [41], or one object of each type [67].

Low-latency detection of candidate signals offers arguably the greatest potential scientific payoff, as the GW observations can trigger followup observations in other channels; i.e , multi-messenger astronomy. For example, combined GW and electromagnetic observations of GW170817 - GRB 170817A [68] have yielded novel insights into the origin of heavy elements [69], neutron-star structure [70], GRB astrophysics and host environments [71], and the Hubble constant [72]. Electromagnetic followup of gravitational-wave signals requires very low latency analysis of the GW data - preferably at the second scale to capture the highest energy emissions (e.g., the prompt gamma and x-ray emission of GRBs), so that it produces accurate sky localisation results to make the follow up possible. Current low-latency GW analysis techniques rely on hundreds of dedicated CPUs and achieve minute-scale latency for automated alerts [73].

CNNs have demonstrated potential for the real-time analysis of data from gravitational-wave detector networks for the specific case of signals from coalescing compact-object binaries such as black-hole binaries. Unfortunately, training these CNNs requires a precise model of the target signal; they are therefore not applicable to a wide class of potential gravitational-wave sources, such as core-collapse supernovae and long gamma-ray bursts, where unknown physics or computational limitations prevent the development of comprehensive signal models.

Recent work by a number of authors [74, 9, 21, 22, 29] has shown that a fundamentally different approach using CNNs has the potential to analyse detector data for GW signals in real time ($\sim$1 s latency) using a single dedicated GPU. However, CNNs demonstrated to date are only capable of detecting signals with a precisely defined signal model (*i.e.*, compact binary coalescences) that is used to train the network. Many potential sources are governed by physics which is either unknown (eg. the neutron star equation of state) [75]) and/or computationally intractable (eg. the modelling of core-collapse supernovae [76] and accretion-disk instabilities [77]); their transient signals are commonly known as gravitational wave bursts (GWBs). While the unknown physics governing GWBs makes the study of such signals exciting, it also poses a challenge: to fully explore the new GW window we need to be able to detect signals from the widest possible variety of sources without relying on precise models for training.

We address this challenge by proposing a novel CNN architecture that analyses not only the detector strain data directly but also the cross-correlation timeseries between detectors. By training the CNN with 'featureless' randomised signals, we are able to construct a neural network that detects coherence (amplitude and phase consistency) between detectors rather than specific signal shapes in individual detectors. We test the trained CNN with real data from the LIGO-Virgo network and show that it is capable of detecting a variety of simulated GWB signal morphologies without being specifically trained for them, at sensitivities close to that of standard GWB searches, but at much lower latency and a tiny fraction of the computational cost.

In section 5.2 we present the architecture of our CNN and describe the analysis and training procedures. In Section 5.3 we present the performance of the trained CNN on both simulated and real LIGO-Virgo data. We discuss the implications of these results and next steps in Section 5.4.

5.2 A CNN for Unmodelled Burst Detection

5.2.1 Network Architecture

Our goal is to be able to detect sub-second-duration GWBs in data from the three detectors of the LIGO-Virgo network, without prior knowledge of the signal mor-

phology. A significant challenge is to distinguish real signals from the background noise transients, "glitches", that are common in these detectors [7, 5, 50].

Typical GWB detection algorithms [78, 79, 47, 80] do this by requiring candidate signals to be seen simultaneously in multiple detectors (simultaneously up to the light travel time between the detectors) and to be correlated between detectors. We follow this logic in our analysis by using a network architecture that combines the outputs of two different CNNs: one that detects coincident signals in multiple detectors (Coincidence model - Model 1), and a second that detects correlation in phase and amplitude between the detectors (Coherence model - Model 2).

Coincidence Model - Model 1

The first CNN is a single-input single-output residual neural network whose main goal is to identify real signals from noise. More specifically it tries to identify coincident signals that appear in at least two detectors. This model takes as input the whitened timeseries data from each of the three LIGO-Virgo detectors. The output is a score on $[0, 1]$, where high values indicate signal and low values no signal.

Residual neural networks are proven to boost the performance of simpler CNNs by reducing the effect of vanishing gradients that makes deep CNNs loose contributions from their first layers and causes their efficiency to saturate. We adapt a network from [81] that compares different methods of using machine learning on timeseries data. In addition to that we optimise it using a genetic algorithm (see below). We find that the resulting residual neural networks outperforms "ordinary" deep CNNs in our case.

More specifically we optimised a simple CNN model with a relatively good performance (overall accuracy $> 95\%$) and then used it as a "residual block", where we feed the output of the block to its input. To find the hyper-parameters of the model we used a genetic algorithm that trained many generations of different randomly initialised models and found which hyper-parameters seem to increase the performance. The main two differences from the original model of [81] was the kernel size that is quite bigger in our application and the reduction of one layer, from three layers to two, in the residual blocks. We also tried to see how many residual blocks are optimal and we saw that three are enough, as it is in the original model. Variations on the number of filters had not obvious effect on the performance and we chose to follow the original sizes on this to make the model less computationally expensive. The final addition was the use of a cyclical learning rate [82] that boosted our performance by 2%.

This residual network as we present in Figure 5.1 has three residual blocks. The first has filter size of 64 and the last two 128. At the end of each residual block we add the filters of the first layer of that block to the output. By doing this the gradient can skip the intermediate layer reducing the vanishing effect. Due to the

fact that we use convolution, all layers are zero-padded to maintain the same size with the input and make this addition feasible. At the end of each layer we apply ReLU activation followed by batch-normalisation.

Finally after the last residual block we flatten all the parameters using global average pooling. Following that we used two fully connected layers with batch-normalisation before the output layer. The output layer has a sigmoid activation and for the calculation of the loss we used binary cross-entropy.

As we discussed before this model is trained to identify signals that are present to at lease two detectors. We trained with simulated noise, signals and glitches with proportions appropriate to focus on that goal. We discuss this more in the training methods 5.2.3

Coherency Model - Model 2

The second model has two inputs and one output. The first input is the same whitened timeseries data fed to the first model, while the second input is the Pearson correlation of each pair of detectors, generated using the **PlugIn** functionality described in subsection 4.3.2. The reason we decided to feed correlation in the input is because after a lot of experimentation there was no clear proof that a model can figure out correlation information (which is crucial for this problem) by just giving as input the strain data. Furthermore, even if this was possible, it is an unnecessary computational burden as the Pearson correlation is simple to compute and more easy to digest by a feature detecting algorithm.

$$r_{\alpha\beta}[n] = \frac{\sum_{i=1}^{N}(d_\alpha[i] - \bar{d}_\alpha)(d_\beta[i+n] - \bar{d}_\beta)}{\sqrt{\sum_{j=1}^{N}(d_\alpha[j] - \bar{d}_\alpha)^2 \sum_{k=1}^{N}(d_\beta[k] - \bar{d}_\beta)^2}}\,. \tag{5.1}$$

Here $d_\alpha[i]$ is the whitened data timeseries for detector α, $\bar{d}_\alpha$ is the mean over N samples, and n is an integer time delay between detectors. The correlation is computed for all n corresponding to time delays of up to $\pm 30\,\text{ms}$ in respect to the first detector. This is the maximum arrival time delay a signal can have between the three detector network. These two inputs have their own separate branches and eventually merge their features in one as we present in Figure 5.2.

Due to the small size of the input the network didn't need to be as deep as the coincidence model. We also used a genetic algorithm that tries different hyper-parameters and trains many generations of models. In the end it keeps the models with the best performances. An interesting observation was that in this model the number of filters is a sensitive parameter that can prevent the model from training at all if it increased or decreased at any of the convolutional layers. On the contrary, variations on the kernel size had surprisingly no big effect on performance. They affected stability of training in some cases though. The choice for kernel size was made based on how often those numbers appeared in the gene pool of last generation

of successful models, where as gene pool we define the values of different hyper-parameters that are present in a generation of models.

There has been an attempt to use residual neural network for the "Strain" branch of the model but is seemed to be an unnecessary complication because the performance is the same. That shouldn't surprise us because the problem that the model is trying to address is different. The two branches of the coherency model are flattened using global average pooling which as in model 1 showed to increase the performance slightly.

This model has also a binary classification output that returns a measure of coherency among any detectors [0,1]. To achieve that we use a different group of datasets than the coincidence model. We use simulated noise and signals but also a combination of coincident and not coincident uncorrelated signals.

5.2.2 Analysis Procedure

In our analysis we use two types of background noise. The one we use for training is Gaussian noise that follows the design curve for the LIGO detectors [83]. For testing we use publicly available data from the LIGO-Virgo detectors. They are read from files accessible from the GW open science center (GWOSC)[84] (https://www.gw-openscience.org/O2/). Important mention again that real detector noise is used only for testing and not for training. The three detectors are denoted H (Hanford), L (Livingston), and V (Virgo). The power spectral density $S_\alpha(f)$ for each detector α is computed using Welch's method, and used to whiten the corresponding data stream. The Pearson correlation is computed between each pair of detectors. The bandpassed [20,512]Hz - whitened data series and the correlation series are then fed into the two models. Finally, the scores from the two models are multiplied together to give a combined score on $[0, 1]$. In practice we find the scores of both models tend to be strongly peaked around 0 for noise and weak signals, and strongly peaked around 1 for strong signals. In this way a candidate signal needs to score highly for both models; i.e. showing both coincidence in multiple detectors and correlation between the detectors.

The LIGO-Virgo data used, from [84], were sampled at 4096 Hz. We downsample to 1024 Hz, allowing us to detect signals up to 512 Hz; this covers the most sensitive frequency range of the detectors and is sufficient for the purpose of demonstrating our CNN. (Since the trained network can process data much faster than real time, we could extend the analysis to higher sample rates.) We chose to focus on signal durations <1 s by analysing data in 1 s segments. This covers many plausible signal models, including for example core collapse supernovae [76], perturbed neutron stars and black holes [67], and cosmic string cusps [56]. We could extend to longer durations as well (with a corresponding increase in latency). At our chosen sampling rate this means that the timeseries data is fed to the CNN in an array of size [1024

x 3]. The Pearson correlation is computed for time delays in steps of one sample over ± 30 ms, for a total of 60 samples; this data is therefore in an array of size [60 x 3]. We use H as the reference and apply the time delays to the L and V timeseries.

To estimate the distribution of scores of the background noise, we repeat the analysis many times after time-shifting the data between detectors by an integer number of seconds. This procedure is discussed excessively in chapter 4. Since the time shift is much larger than the largest possible time-of-flight delay between detectors, it prevents a real GW signal from appearing in coincidence between multiple detectors. All coincident events in the time-shifted series can therefore be assumed to be uncorrelated and treated as background noise. This is a standard procedure in GW analysis; see eg. [7].

To estimate the sensitivity to GWBs, we repeat the analysis after adding simulated signals to the data. In General Relativity, a GW has two polarisations, denoted $h_+(t)$ and $h_\times(t)$. The received signal $h_\alpha(t)$ of a given detector α is the combination

$$h_\alpha(t) = F_\alpha^+ h_+(t) + F_\alpha^\times h_\times(t) \tag{5.2}$$

where the antenna response functions $F_\alpha^{+,\times}$ are determined by the position and orientation of the source relative to the detector. We characterise the strength of the received signal by its network optimal signal-to-noise ratio

$$\rho = \sqrt{\sum_\alpha \; 4 \int_0^\infty \frac{|\tilde{h}_\alpha(f)|^2}{S_\alpha(f)}\, df}\,. \tag{5.3}$$

Generating simulated signals distributed isotropically over the sky and rescaling to different ρ values allows us to measure the distribution of CNN scores as a function of the signal-to-noise ratio of the signal.

5.2.3 Training

The choice of data used to train a CNN is a critical factor for the CNN's performance. For the signal population we use white-noise bursts (WNBs) [47, 7]; these are signals where the h_+ and $h_\times$ polarisations are independent timeseries of Gaussian noise that is white over a specified frequency range, multiplied by a sigmoid envelope. We select these as our training sample as they are effectively featureless. The bandwidth of each injection is selected randomly and uniformly over the range $[40, 480]$ Hz. The duration of each injection, is selected randomly and uniformly over the range $[0.05, 0.9]$ s. Given their duration their central time is positioned randomly inside this 1 second interval in a way that they don't get cropped. The injections are distributed uniformly over the sky and projected onto the detectors using equation (5.2). Finally, the signal is rescaled to a desired network signal-to-noise ratio ρ as defined in equation (5.3). In the rest of this section we will discuss different

aspects of our training data and methods, that eventually gave us the current best model.

Code

All training data was generated using the MLy pipeline [85] as discussed in chapter 4 and its generator function, which uses elements of the PyCBC [86] and GWpy[48] package to project the signal onto the detectors and apply time-of-flight differences to the signals arriving at the various detector locations. For training the models we used Keras [87].

Data types

Each model is trained with different datasets that focus on challenging the models to achieve their best performance on each of their tasks. All data instances have background noise and if applicable a type of injection. For the background samples, we find in practice that the best CNN performance is obtained by training with *simulated* detector noise and glitches, rather than real glitchy detector noise. We therefore use Gaussian noise with a power spectrum that follows the design curve for the LIGO detectors [83] to model the stationary component of the LIGO-Virgo background noise. The reason the performance is better is that by using real noise the occurrence of glitches is not enough for the model to learn to recognise them. In our application we created noise instances with glitches and we can control the frequency they appear. As it will be shown later this helps a lot with the reduction of false alarms.

For the coincidence model we use three types of data instances: noise, glitches and signals. Noise follows the stationary component of LIGO-Virgo background with addition of random fluctuations. For the glitches we use the same waveform bank as we do with signals and we do a single detector injection to a randomly chosen detector. Those injections will have a variety of SNR values ranging between [6,70]. Noise and glitch datasets are both labeled as “noise” they belong in the same classification class. Signals are the only type labeled as “signal” and they are injected in all three detectors as described above with SNR values inside the range [12,30]. The limits of this range are chosen based on the fact that lower SNR values in training increase the false alarm rates dramatically and that higher SNR than 30 is quite obvious to be detected by the simplest pipeline and as proven later the detection efficiency holds to higher SNR values than trained ones.

For the coherency model we also use three types of data. Noise and signals are as before with the difference that the range of SNR values now is [10,50], because after trials on minimum and maximum SNRs included, this was shown to be best. The third type are the incoherent signals. This type is also labeled as ”noise” and they represent the extreme cases where glitches or non gaussianities are present in

at least two detectors. To make this we randomly select three different injections from our bank and we inject them in the detectors like they are three projections of the same signal. Their network SNR is in the range [10,70].

There are three subcategories of incoherent signals that are themselves three extra datasets. Pearson correlation is calculated only around the window of $\pm 30s$. We expect that even different signals will have some kind of excess correlation, although this correlation will not be the same in all three pairs of detectors (HL,HV,LV). Moreover each injection can have different duration which will affect the overlap of those signals. To control their relative positions we created a metric called "disposition", described in detail in section 4.2.4. Disposition is the range of the central times of those signals. In the case of three detectors and a disposition of 0.5 seconds one randomly selected signal will be put at 0 seconds the next at 0.25 seconds and the last one at 0.5 seconds. All together will be randomly positioned in the 1 second interval so that no signal gets cropped. Finally given the disposition there is a restriction imposed on the injections duration, so that the don't go outside of the range of 1 second. Meaning that injection duration and disposition are complementing each other to 1 second. We explain this in detail on section 4.2.4. We generate three datasets with disposition options. The first one has range of [100,500] **(3a)** milliseconds, the second [0,100] **(3b)** and the last one has zero disposition **(3c)** to challenge the model with coincident incoherent signals.

The list below summarises all the types of data we used. All the types are described in detail in Chapter 4:

- **Type 1:** Gaussian LIGO-Virgo noise, with no injections. Labeled as noise.
- **Type 2:** Gaussian LIGO-Virgo noise with coherent WNB injections in all detectors, simulating a real GWB. Labeled as signal.
- **Type 3:** Gaussian LIGO-Virgo noise with different (incoherent) WNB injections in each detector, simulating unrelated glitches or excess noise in each detector. The injections may be simultaneous or offset in time from each other to simulate simultaneous or nearly simultaneous glitches. Labeled as noise.
- **Type 4:** Gaussian LIGO-Virgo noise with a single WNB injection in a randomly chosen detector, simulating a glitch in a single detector. Labeled as noise.

Reproducibility

It is crucial for one method or model to have reproducible results if it is going to be compared with another. The weights of each model are initialised with random normal numbers before training starts. Furthermore the path that the gradient decent will follow during training is never the same, even with identical initialisation.

This can lead to different results if we train the same model with the same data two independent times. To address this issue we train a given model multiple times to see how important is this effect on the variance of the results. This is a crucial issue for the next subsections where we compare different methods of training.

The variation is present on the false alarm rate test and the efficiency curve (see section 5.3). For the false alarm rate what varies is the score of the loud events. For the efficiency the variation depends on how sensitive is the model to a signal but it also depends on the the loudest false alarm rate event because it defines the threshold.

In Figure 5.3, we trained a pair of models 1 and 2, ten times each. This provides a good enough estimation of the variance and mean of the false alarm curve. Additionally by combining all the possible pairs we get 100 independent tests of the final model. This practice is followed in the next subsections where we investigate different methods of training.

Input ratios

In more common classification problems the training data have the same amount of examples in each case, because all classes are equal and there is not preference in the performance. In this problem we prioritise the reduction of false alarms (false positives) and the ability of the network to understand how noise looks like, instead of what is a signal. Even with our best model, a 50-50 ratio gives back a false alarm rate of 1/day. Furthermore we have more than just simple noise and injection in our data types, and we try to simulate a real noisy detector with Gaussian noise and WNBs, so we also needed to investigate what are the optimal proportions of each type. We trained both models with different ratios of the data types to check which one provides the best performance.

For model 1 we tried 12 different proportions of different training data. We tested combinations of 3,6, 3,6,12 and 3,6 of 10^4 instances of Type 2, 4 and 1 respectively. For model 2 we have 16 combinations. The tested combinations are of Type 2, 3a & b together, 3c and 1 with choices being (5,10) 10^4 for each of them. For all those different proportions we train 7 models and we use the mean false alarm curve and efficiency test for comparisons. We get in total 192 different combinations of model 1 & 2. As is described in more detail in section 5.3, the ranking of a model is defined by the 50% threshold of its efficiency on a fixed false alarm rate. After training, all models were tested for false alarms with the same background noise from O2 LIGO-Virgo scientific run during August 2017. Additionally we run an efficiency test on various waveform types to all models, using again the same exact background data and injections for all models so that the comparison is robust.

The combination of proportions with the best performance from each type is the one that follows 3-6-3 (Type 2 - Type 4 - Type 1) proportion for model 1 and

5-10-5-5 (Type 2 - Type 3a&b - Type 3c - Type 1) for model 2. In both cases it is interesting to see that for the best training we need much more glitch like data than plain noise or signals. That supports the idea that noise by itself is not enough to train the algorithm to ignore glitches.

Virgo elevation

As it is stated, before we train with artificial Gaussian noise that follows the PSD of the advanced LIGO and Virgo detectors. Although that PSD is not representative of the real data, especially when the real data are from the O2 observing run. More specifically, the distance between the Virgo and LIGO detector noise level is smaller in the artificial data, than what it is in real data. For that reason we experimented on training the best model from the previous investigation (5.2.3) with different noise level elevations for Virgo, by multiplying with powers of two the background of the artificial data: [0.5,1,2,4,8,16,32]. The case of 1 is the same models used in the previous section and the case of 0.5 is for a sanity check on the method. Again for each model we trained seven independent models with the same parameters and used the mean to compare the methods.

As we present in Figure 5.4 the higher the elevation the better the false alarm rate. That is expected because if Virgo is much louder than LIGO detectors most of the SNR contribution will go to Hanford and Livingston strains. Eventually a lower false alarm threshold, makes more injections detectable. As we present in Figure 5.5, making Virgo around 10 times louder than the other detectors will increase the probability of a randomly localised injection to be detected. This is a controversial situation where we practically teach the model to ignore Virgo, unless the injection is really loud. It is not the first time [4] where HL analysis is better than HLV, assuming that our case of a loud Virgo resembles an HL analysis.

5.3 Evaluation

Following the investigation to find the best working model we found one with the best performance compared to others. The evaluation below describes how we calculated the values we talked about in previous sections but focusing on the best current model for the final results.

5.3.1 False Alarm Rate

A standard means to assess the performance of a GWB detection algorithm (see eg. [7]) is to measure the detection efficiency for various signal morphologies as a function of the signal amplitude (signal-to-noise ratio) at a fixed false alarm rate.

We calculate the FAR as a function of score by analysing background data samples that are independent from those used in the training. For the final model we

measured the FAR using real LIGO-Virgo data from the second observing run during times when all three detectors were operating (1 - 25 Aug 2017), with time shifts applied as discussed in Section 5.2.2. For this calculation we used the **falseAlarmTest** function and its scheduler function as they were discussed in section 4.3.

We consider that 1 per year for unmodelled searches is a good starting threshold to see if our models could be relevant for triggering electromagnetic follow-up observations. This choice is motivated by LIGO-Virgo for issuing public alerts of candidate signals [88]. We calculate the FAR up to 10 years in our results to show the limits of the current models and methods.

From this distribution we can find our false alarm rate threshold (1/year) and calculate the detection efficiency. In Figure 5.6 you see the false alarm rate for the current best model found from the above investigation.

5.3.2 Detection Efficiency

We calculate the detection efficiency of our model for a selection of different possible GWB signals, both for the case of simulated Gaussian background noise and for real LIGO-Virgo data from the second observing run. In each case we generate new sets of signal injections (different from the training data) and add them to a noise background from the O2 observing run. We measure our sensitivity to five distinct waveform morphologies:

WNB These are the same type of signal as the Type II used for training.

CSG A circularly polarised sinusoidal signal with Gaussian envelope. These *ad hoc* waveforms are standard for testing GWB analyses [7, 5].

CCSN The N20-2 waveform of [54], from a 3D simulation of a neutrino-driven core-collapse supernova (CCSN) explosion.

Cusp The GW emission expected from cosmic string cusps [**?**].

BBH The GW signal from a black-hole binary merger. We used the IMRPhenomD waveform model [53][89]. The black-hole masses and spins were selected randomly and uniformly over the intervals $[10, 100]\,M_{\odot}$ and $[-1, 1]$, with the restriction that the spectrum of the signal from the binaries producing those waveforms does not exceed 512 Hz.

To create the efficiency curve we select a range of SNR values and for each one of those SNR values we generate 1000 injections (Type 2) where each one of them is rescaled to match this value. Each injection is embedded to the background timeseries and it is processed by the models giving a score. The injection is considered to be detected if the output score is larger than that of the false alarm rate test threshold. We test those 1000 signals of each morphology at each value of the range $\rho =$

$0, 1, 2, \ldots, 50$, where the $\rho = 0$ case corresponds to pure noise (where we expect small efficiency for a small FAR). For the efficiency calculation we used the **efficiencyTest** function and its scheduler function as they were discussed in section 4.3.

Fig.5.8 shows the detection efficiency for each case. We see that we are able to detect >50% (>90%) of all signals that have amplitudes $\rho \geq 19$ ($\rho \geq 27$), with the exception of cusps for which the sensitivity is lower. The performance for CCSN and BBH signals is very similar to that for WNBs, even though the signal morphologies are quite different. The performance for CSGs is even better, which we attribute to the very small time-frequency volume that it occupies. The performance for cusps is poorer, which probably occurs because cusps are linearly polarised and WNB training injections are elliptically polarised. This is also apparent in the difference in performance of SGE153Q8D9 compared to SGL153Q8D9 where a similar reduction of performance is detected for the linearly polarised signals. This is an interesting demonstration of where our training could potentially be improved to handle a yet broader range of signals.

We tested the models also on the five BBH signals that appear in August 2017 on the O2 run. Although as expected, due to their SNR being below our 50% threshold, we were not able to detect them.

5.3.3 Comparison with the all-sky search in O2

Except for the performance test for the various waveforms above we also do a comparison with the all-sky search results for short duration bursts in the O2 observing run [90]. In this performance test we follow exactly the same procedure as before but instead we use the h_{rss} as a metric of intensity of the signal. Note here that the h_{rss} calculation happens before the whitening of the data. Hence signals with frequencies of low detector sensitivity will result to a reduced representation in SNR. There is no clear matching between SNR and h_{rss} due to that fact. The reason we use h_{rss} is because we need to have the same metric to compare the waveforms. In Figure 5.9 we show the efficiency curves of all the waveforms we compare, with the addition of one linearly polarised one that we don't compare with.

In Table 5.1 we see that the 50% efficiency thresholds are very close. Although this shows the potential of our models. SGL1538D9 are the same type of waveforms with SGE1538D9 with the only difference that they are linearly polarised. Their performance is worse than the elliptical polarisation as it was also observed by the Cusps performance.

5.3.4 Inference Time

We note that the CNN analysis of data is very fast: we find that the average time required to process 1 second of data, given that the data are already downloaded and processed, is 51 ms on a 3.5 GHz Xeon E3-1240v5 quad-core CPU with 32 GB

Performance comparison on the O2 HLV data set		
Morphology	cWB [5] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	2.8	3.5
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	1.5	1.9
f_0=153 Hz, Q=8.9	1.3	1.7
f_0=235 Hz, Q=100	0.9	1.4
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	1.2	1.7
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	1.4	1.8

Table 5.1: The values shown refer to h_{rss}, in units of $10^{-22}\mathrm{Hz}^{-1/2}$, at the 50% efficiency threshold. Comparing waveforms from cWB results [5] but modified for iFAR of 1 per year. We were provided with the efficiencies for FAR values of 1 per year for cWB by Shubhanshu Tiwari, a member of cWB team. We restrict the comparison to injections with frequencies and durations that are in the range for which we have trained the MLy models (duration $<$ 1s , frequencies in the 20 Hz - 480 Hz band).

of RAM. This is much faster than the minute-scale latency typical of current LIGO-Virgo low-latency searches [73]. Also the time to train both models on a Tesla V100-SXM2-16GB GPU is 4 hours.

Important to note is that due to the fact we train with artificial noise the training is independent of the observing run. A false alarm rate calculation would need to be repeated only to make sure that it stays stable along the run once in a while. By the time it is trained and tested, inference is the only thing we need to do.

Finally, given the low latency inference of those models, we suggest that it's combination with sky-localisation pipelines [91, 29] can be a complete set for an unmodelled signal search. Although we understand that this needs more work to be generalised to unmodelled signals except CBCs.

5.4 Discussion

Unlike previous CNN-based analyses, our pipeline is capable of detecting waveforms with morphologies that are not included in the training set while rejecting real detector noise glitches. The analysis is shown to be sensitive to a variety of waveform morphologies at signal-to-noise ratios and false alarm rates relevant for issuing rapid alerts to the astronomical community, with very low latencies and computing requirements.

Our pipeline uses a multi-component architecture where one CNN detects transients that are simultaneous in multiple detectors while a second detects correlation between the detectors to eliminate coincident background glitches. The second CNN takes as input both the whitened detector timeseries data and the Pearson correlation between detectors computed for all physically allowed light travel time delays between detectors, allowing the CNN to detect signal correlation rather than signal shape.

We suggest using separate models to identify different aspects or properties of the desired signal may be a useful approach generally for GW detection with machine learning methods. While our model already has sensitivity approaching that of standard low-latency analyses, we consider this investigation to be a promising first attempt with potential for improvement. For example, training with more sophisticated simulated and real glitches may reduce the threshold required for the target false alarm rate, while training over a wider range of parameter space (like polarisations) may increase sensitivity to transients similar to cosmic string cusps.

In the next chapter we will use this model to run a search on O1, O2 and O3 observing runs.

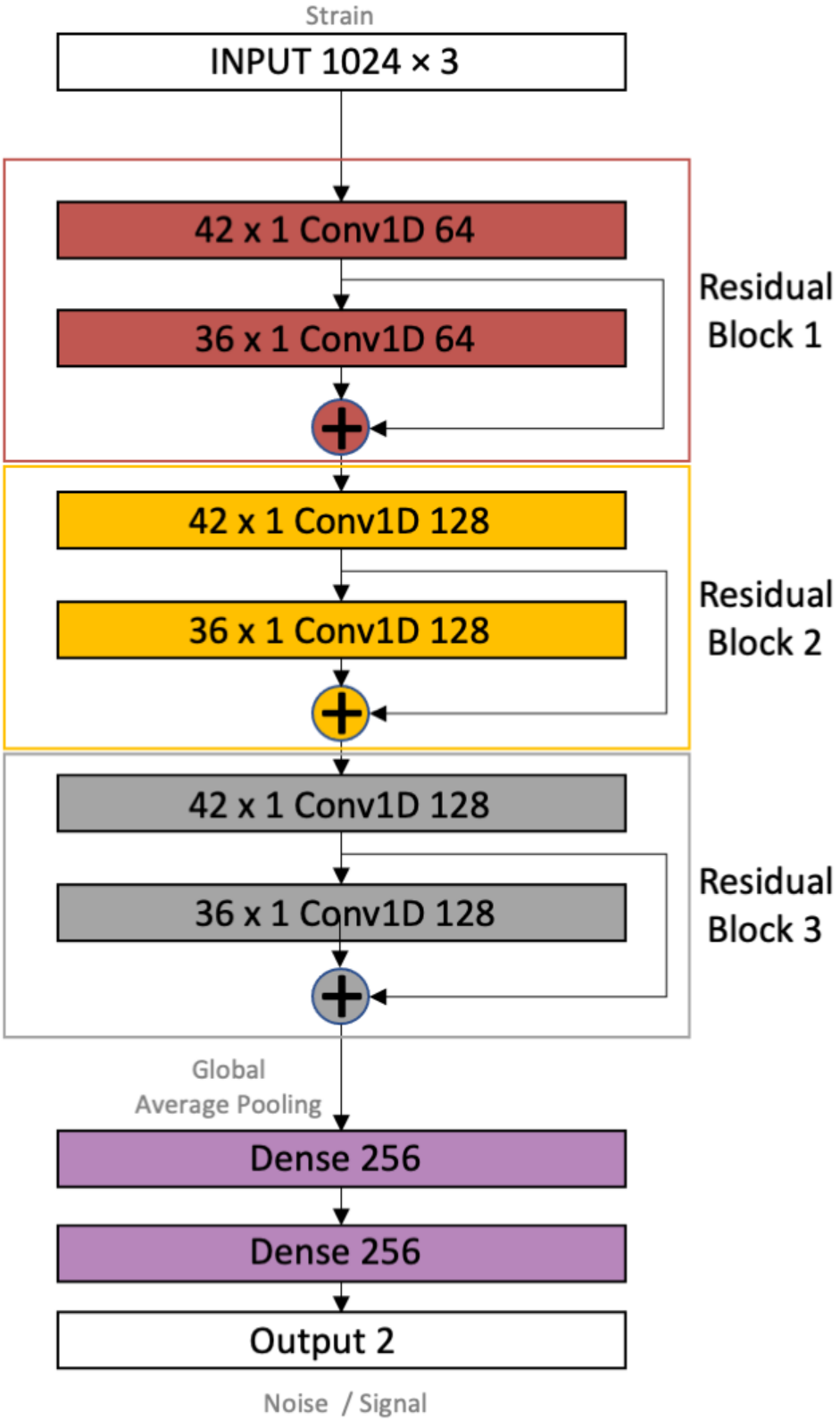

Figure 5.1: Coincidence model architecture (model 1). For each layer we show the kernel size on left (if applicable) and the number of filters on the right. The plus symbol indicates the summation of weights between outputs of different layers.

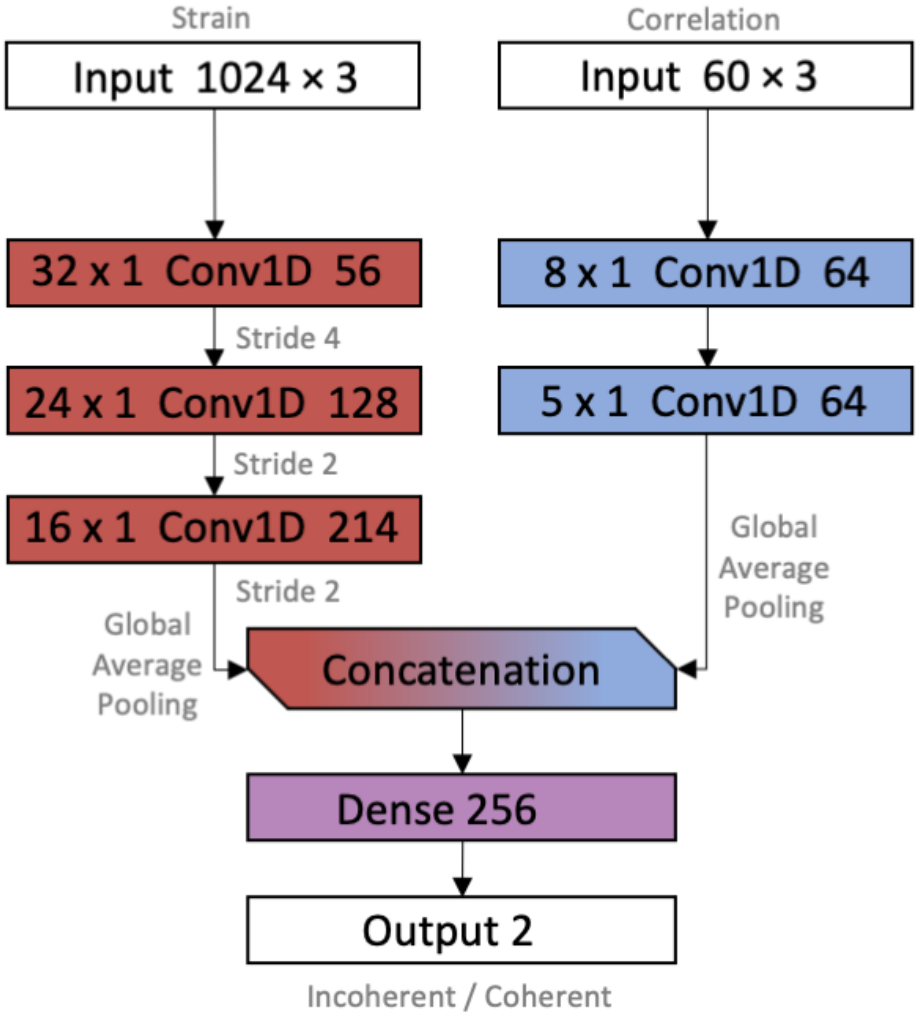

Figure 5.2: Coherency model architecture (model 2)

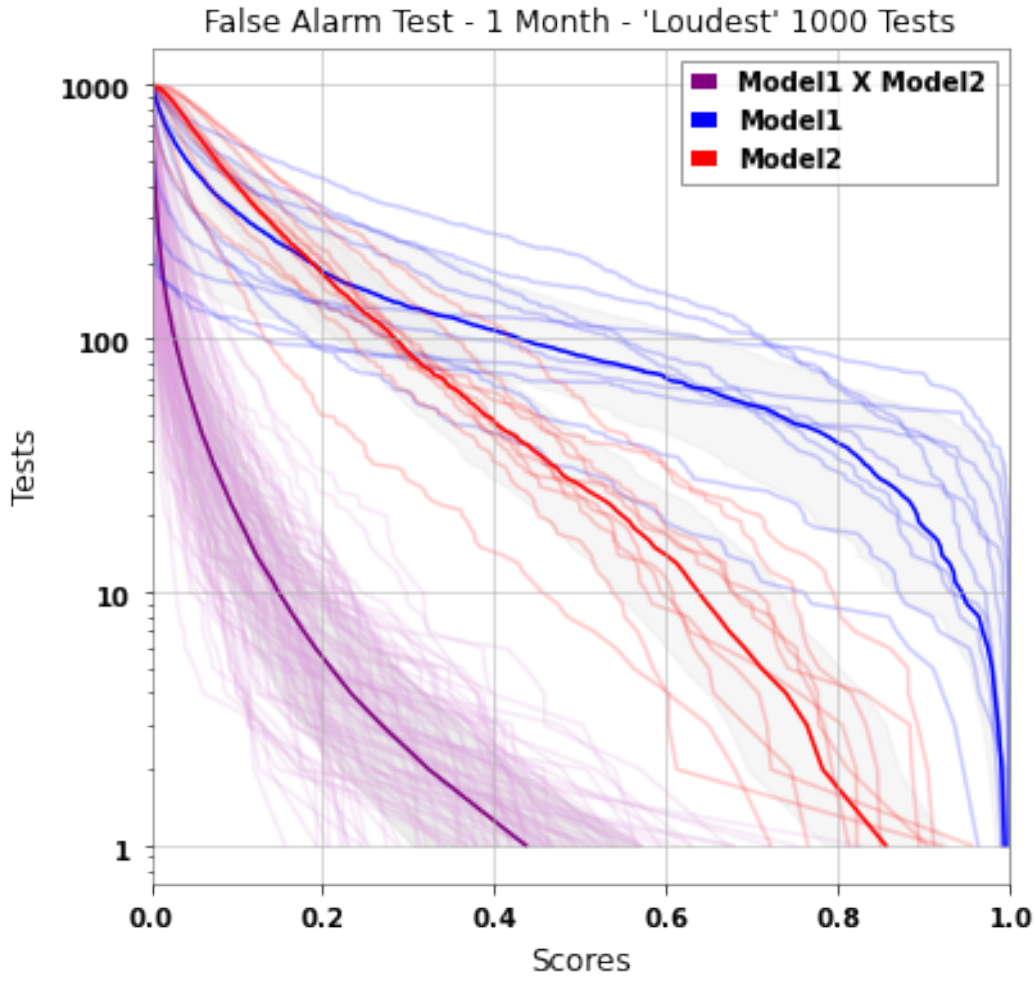

Figure 5.3: False alarm test results for 10 different trainings of Model1 (blue) and Model2 (red) and their combination (100 models - magenta) of scores. For the false alarm tests we used the same instances among the models. We tested 1 month of data and we present here the 1000 loudest events for each model. We highlight with a bold line the mean result for each model. The gray area represent one standard deviation for each case.

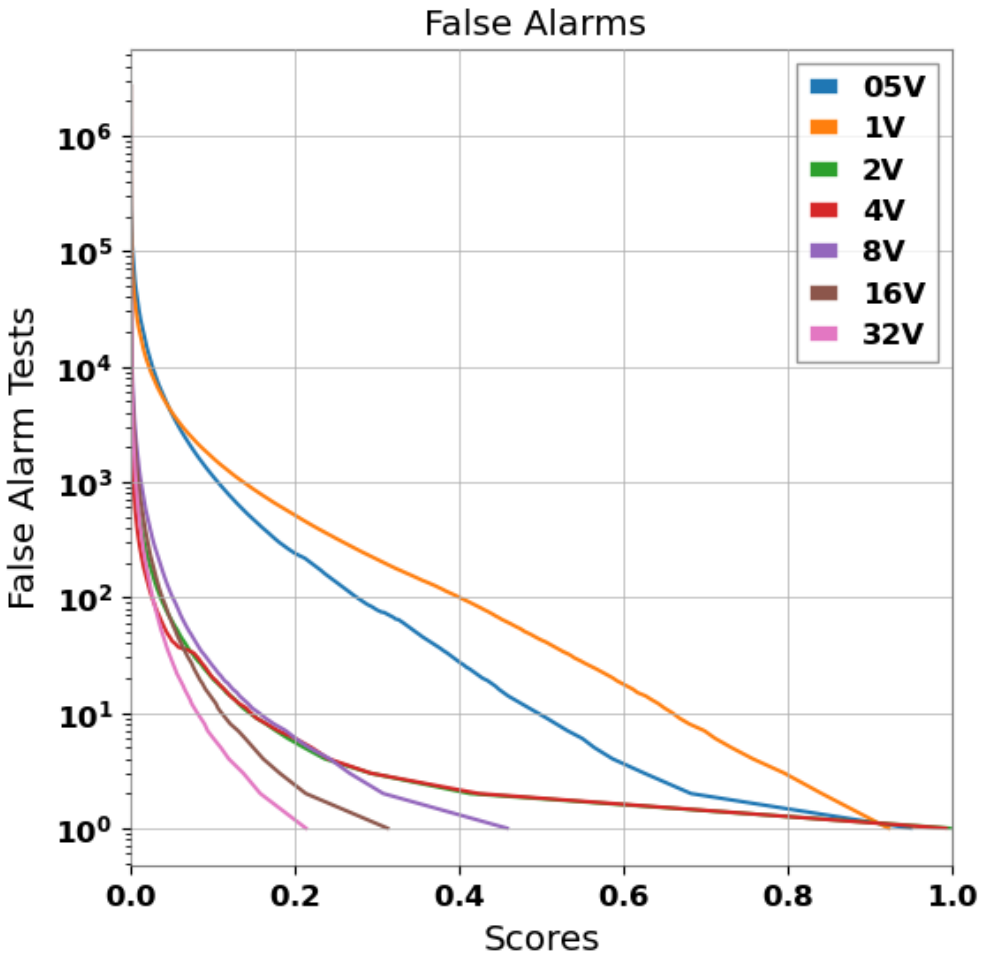

Figure 5.4: Average false alarm rates of models trained with elevated Virgo level. For each average curve we calculated the false alarm rate of 49 models (7 model1, 7 model2 combinations), using the same data instances.

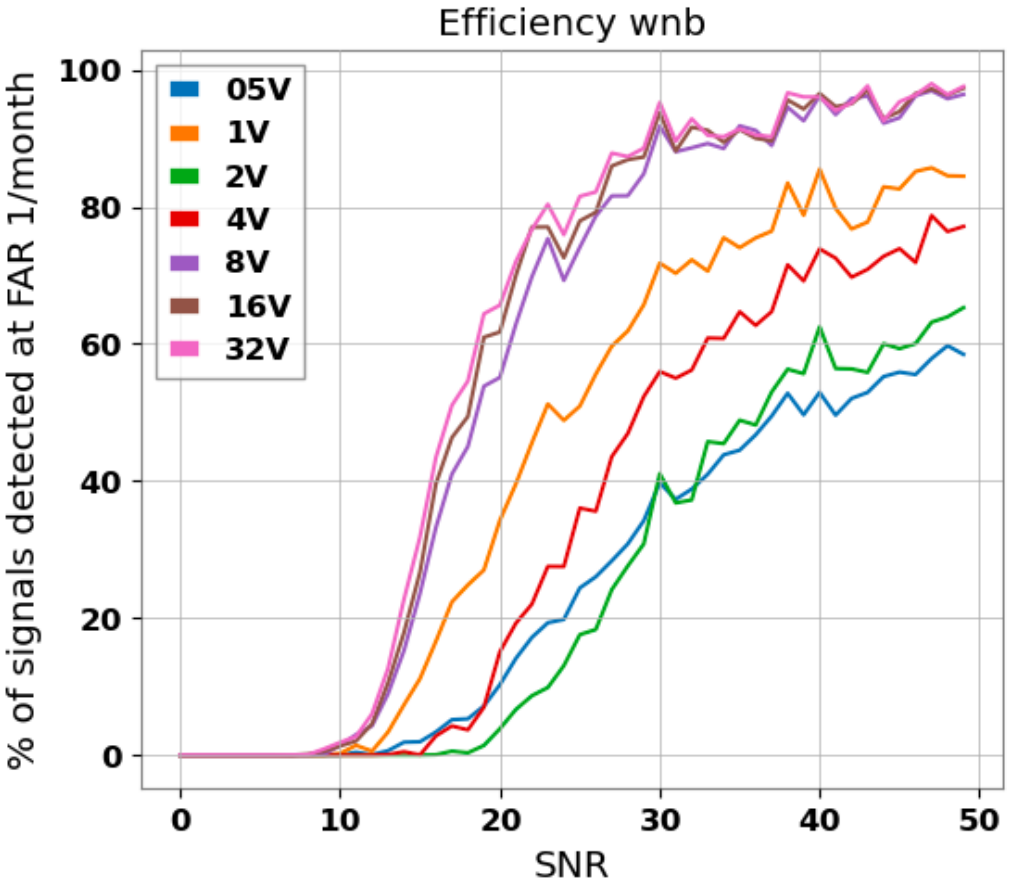

Figure 5.5: The average efficiency curve (out of 49 different combinations of model1 and model2) for WNB signals based on thresholds of 1 per month false alarm rates for each case.

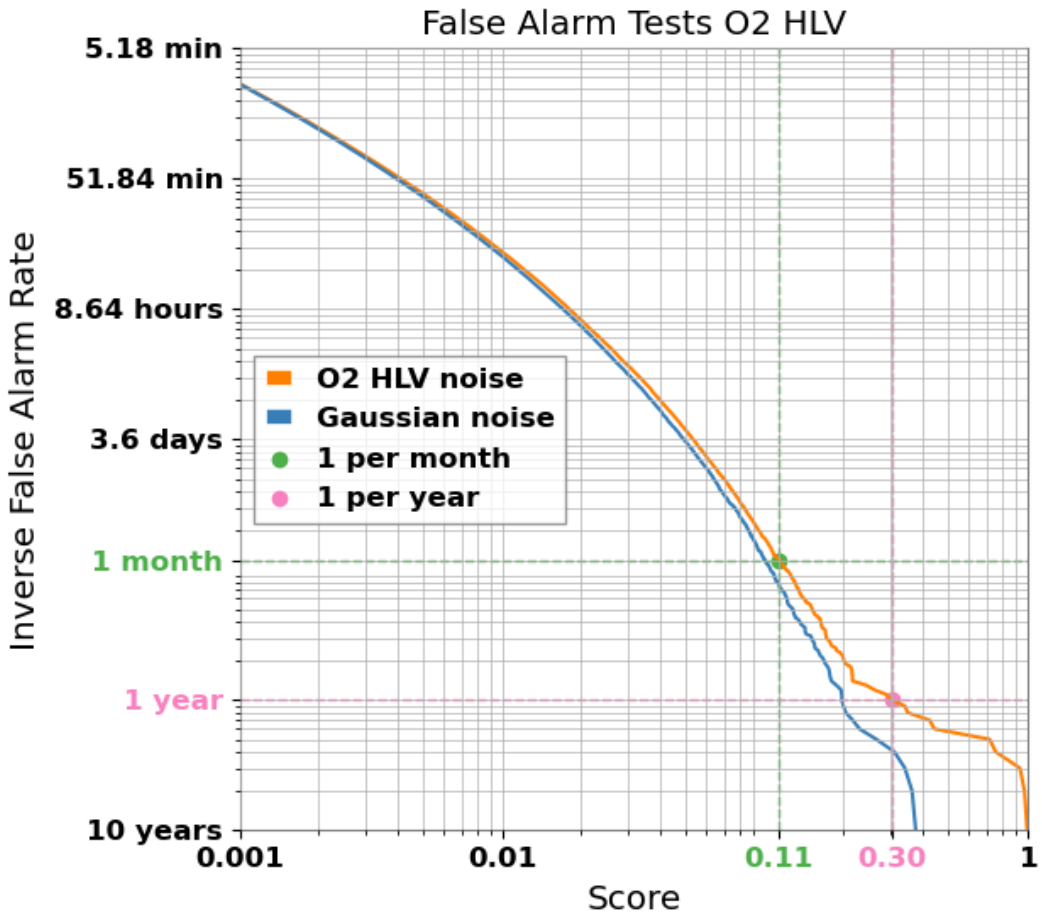

Figure 5.6: False alarm rate of the current best model, up to once per 10 years. We compare the performance of Gaussian Noise and O2 real noise. The score thresholds of 0.11 and 0.30 false alarm rates of 1/month and 1/year are also shown respectively.

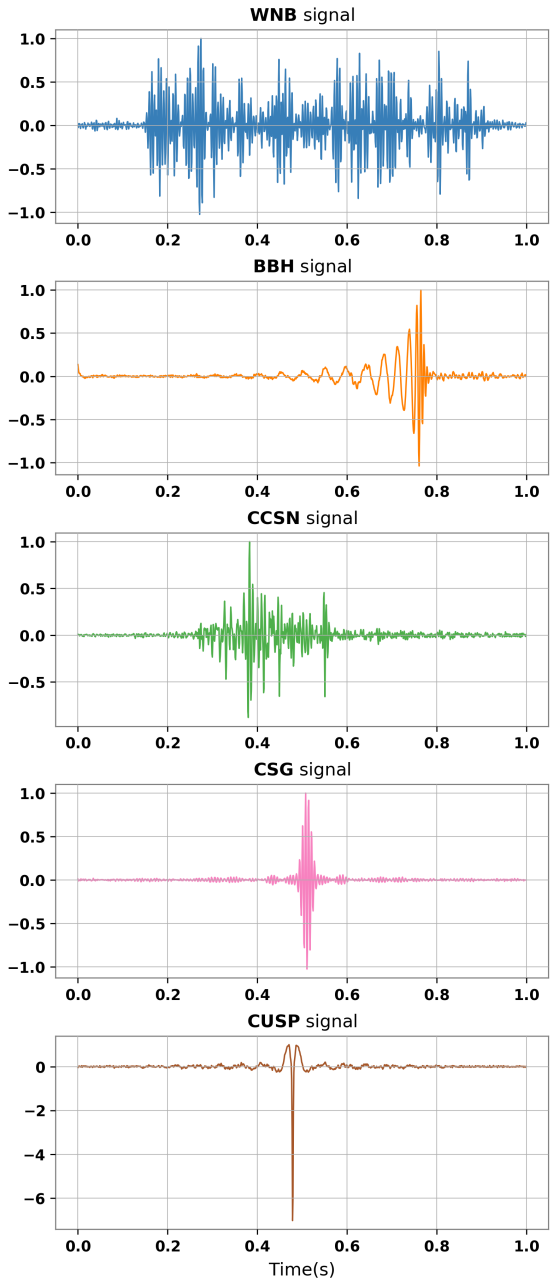

Figure 5.7: Examples of testing signals of high SNR embeded in Gaussian noise: white noise burst (WNB), binary black hole merger (BBH), core-collapse supernova (CCSN), circularly polarised sine-Gaussian (CSG) and cosmic string cusp.

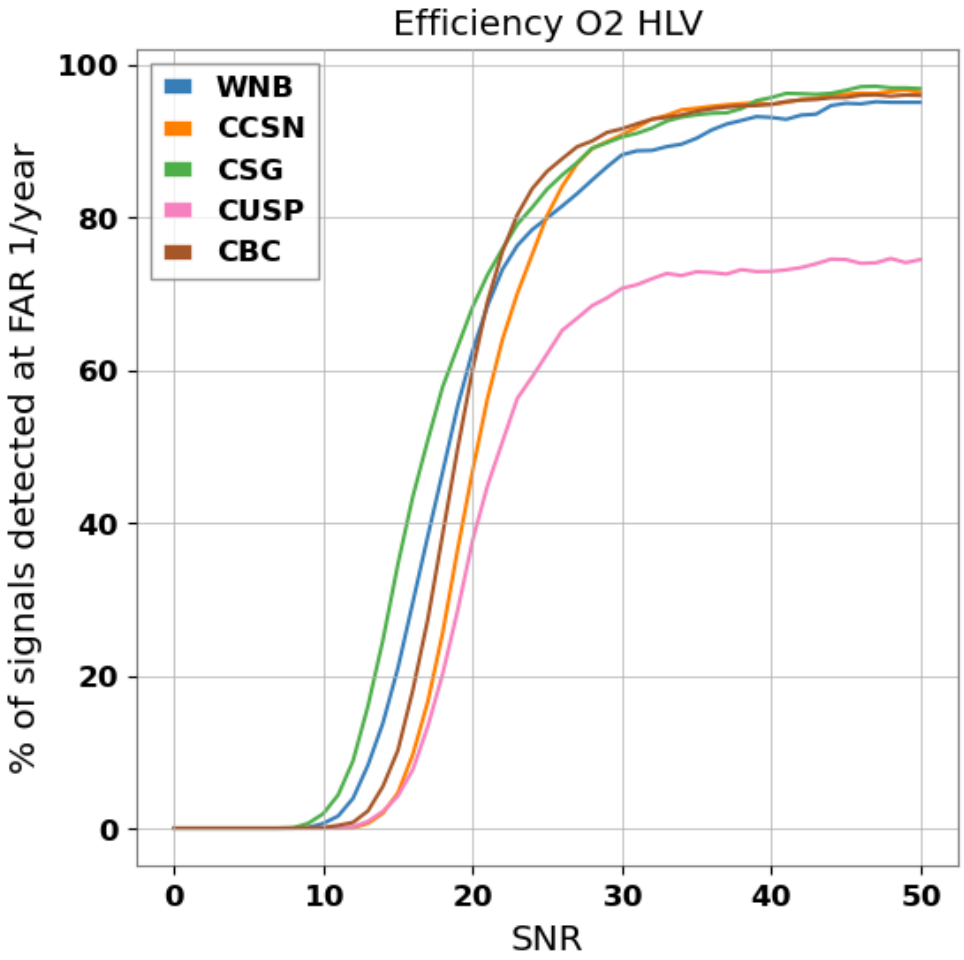

Figure 5.8: Detection efficiency : The percentage of simulated signals that are detected at a false alarm rate of 1 event per year versus the signal-to-noise ratio (SNR) defined by equation 5.3. The waveform morphologies are white noise burst (WNB), core-collapse supernova (CCSN), circularly polarised sine-Gaussian (CSG), cosmic string cusp, and binary black hole merger (BBH).

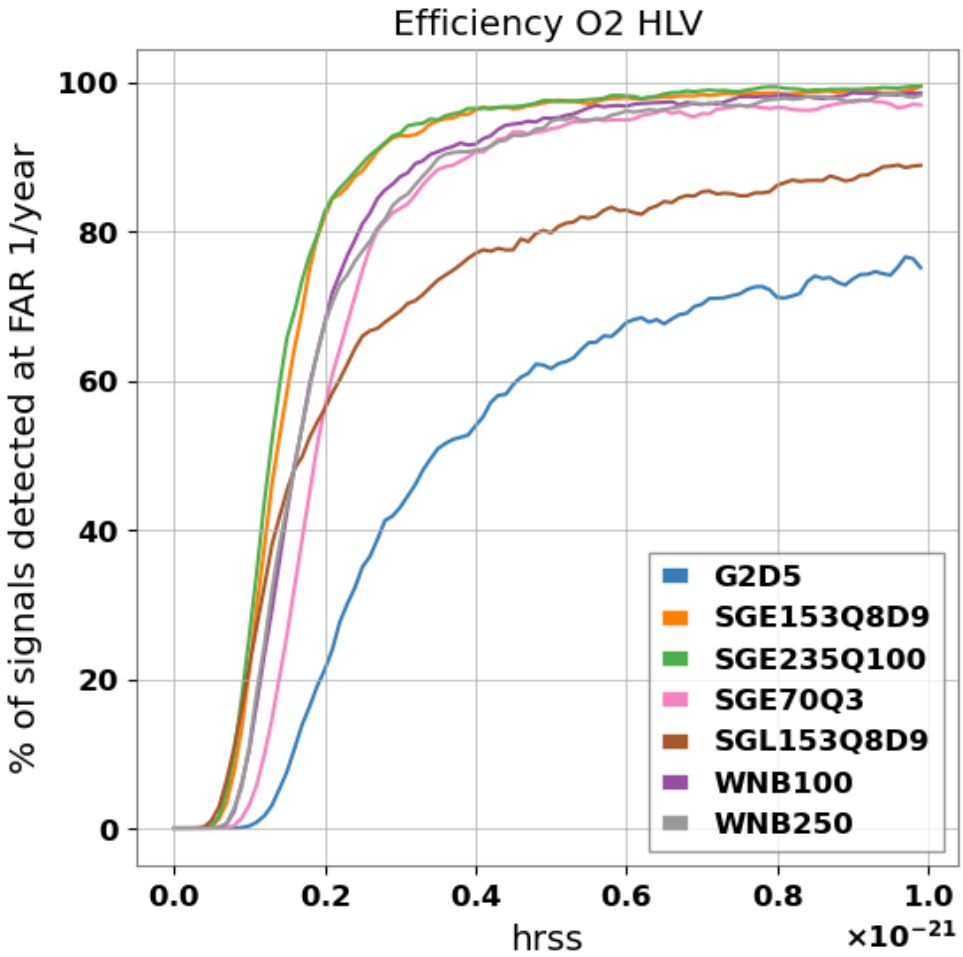

Figure 5.9: Detection efficiency for generic sub-second bursts for iFAR 1/year.

Chapter 6

Offline search using CNNs for generic subsecond waveforms

In the previous chapter we described our pipeline for real-time detection of unmodelled gravitational wave transients. We showed in Figure 5.8 that the current best model can detect WNB sub-second generic waveforms with a false alarm rate of once per year when SNR values exceed ~ 19. In this chapter will extend our results by applying our best model to an off-line search on O1, O2 and O3 observing run datasets. In section 6.1 we discuss two modifications to the analysis procedure required for performing a search for real signals in the O1, O2 and O3 observing run datasets.

6.1 Process

For each off-line search we follow the same process with small differences in some details related to how we access the data. We use the same code to process the data that we use to calculate the false alarm rates and the efficiency curves as described in Chapter 4 and 5. In this off-line analysis we don't use time-lags. All our data are processed in zero lag as they were observed from the detectors. Finally we will add an extra feature to make the search more efficient and to make sure we don't loose any signals. This feature is called the time-strides.

A time-stride is the time step between two successive data instances that we process. In the false alarm testing we use the default stride of 1 second because we don't want any data to be repeated during the tests. During a search we want to make sure that any signal will have the opportunity to pass through the center of our process window. For that reason we use a stride of $\mathcal{S} = 1/8 = 0.125\,s$. The choice for that number is based on computational costs. In theory we can use a stride of $1/f_s$, although the computational cost would be excessive. Using 0.125 seconds as a stride is enough to give us an idea of how long the algorithm reacts to each trigger. We expect that a short-duration trigger will have a continuous high

score as it passes through the one-second processing window.

By using the strides a short duration signal might be inside the analysis window up to $T/\mathcal{S}$ times, where T the duration of the analysis window. In our case $T = 1\,s$ so $\mathcal{S} = 1/8\,s$. Triggers that have a distance in time equal to the stride, are grouped together in a **super-event**. For each super-event we keep the highest score as the overall score of the super event and the GPS time of that highest score as the time of the super event.

6.1.1 An HL search with an HLV trained model. How?

The MLy pipeline demonstrated in Chapter 4 is designed to analyse data from the three-detector HLV network. However, for a large fraction of the O1, O2 and O3 runs (including all of O1) only the HL pair were operating. As you will see below we do an HLV analysis, but also an HL analysis using the same model for each observing run. We will explain here how is this possible. At the beginning we attempted to see how the model would react if the data from one of the detectors is only zeros and found that the performance was reduced dramatically. This is expected because the models were never trained with zeros in their input. We note however that by applying the Virgo elevation technique in the training we effectively taught the model to ignore Virgo data provided there is not a loud correlated signal in them. We take advantage of that and instead of creating a whole new model, that is trained with only two detector data, we add an elevated simulated Virgo noise whenever Virgo is not in observing mode. It is important to note here that even with a very elevated Virgo, after whitening the data, there is no difference in the noise level from the other detectors. In this way we feed a familiar representation of a detector without signal, so that the model doesn't react unpredictably. Finally we have to note that for the moment we chose not to analyse data when only the HV or LV detectors were operating. We do this because without two LIGO detectors operating the overall sensitivity of the network is much lower. Testing the performance on these networks while substituting Gaussian noise for the missing H or L detector is a subject we leave to future study.

To test this theory we created a series of 200 injections of WNB signals and we embedded them in 200 different real LIGO-Virgo noise instances. First we injected the signal in the HLV detectors and noted the scores. We then used exactly the same pairs of injection and noise but omitted to inject the signals into Virgo. Note that we did that without adjusting the intensity of the signal in the LIGO detectors. We want to prove that in most cases, if the signal is missing in Virgo, there will be a minimal effect to the score compared to the HLV score. In addition to that we repeat the same procedure but now we zero out the Livingston injection to demonstrate that if we omit any other detector except Virgo, the model will fail.

In Figure 6.1 we present the comparison of HLV scores, HL (Virgo missing) scores

and HV (Livingston missing) scores for three different h_{rss} values. The first value of $1.8 \times 10^{-22}\text{Hz}^{-0.5}$ corresponds to the 50% threshold for WNBs in the frequency range 250-350 Hz in the O2 search as presented in Table 5.1 and Figure 5.9. The second is $4 \times 10^{-22}\text{Hz}^{-0.5}$ which the 90% threshold of the same type and the third value is $2 \times 10^{-21}\text{Hz}^{-0.5}$ which is just an really high intensity of the signal.

It can be seen that for the 50% threshold value the HL search (magenta) stays in mostly the same scores with HLV, with a $\sim 10\%$ of them being outliers scattered around. The outliers are expected due to the random sensitivity machine learning can show to unknown features. All outliers are equally distributed to both sides of the dashed line. On the contrary HV values (blue) are all zero, showing that the model will not detect signals if Livingston is empty. For the 90% threshold value we see a similar behaviour, although HL scores are closer to 1 this time. In the last exaggerated intensity case we see nearly all HL values to 1, with the expected outliers. The HV case continues to be zero.

This test demonstrates that we can expect to be able to analyse times when the Virgo detector is offline using our three-detector model by substituting simulated Gaussian noise for the Virgo data stream. We stress that to test the validity of an HL analysis we still have to run a separate false alarm and efficiency tests for that kind of data. We will show below that this technique makes it possible for our model to detect HL-only events such as GW150914 with high confidence. This approach will have the further benefit in the future scenario of an online analysis, in that only one model will be needed and no switching between three-detector and two-detector models will be required. In the following applications of the HL search, we will run the search along all the available coincident times of H1 and L1 and we will not exclude times were Virgo is online for convenience. Although it might be advantageous to combine the analysis of data with both HLV and HL-not V technique. For example if an event is detected in both versions of the search, it could possibly reduce its false alarm rate. Although due to the fact we use the same model for both searches we cannot treat HLV and HL-not V version as totally independent and more research needs to be conducted about that matter. Hence this search we will treat HLV and HL-not V as separate searches without combining their findings.

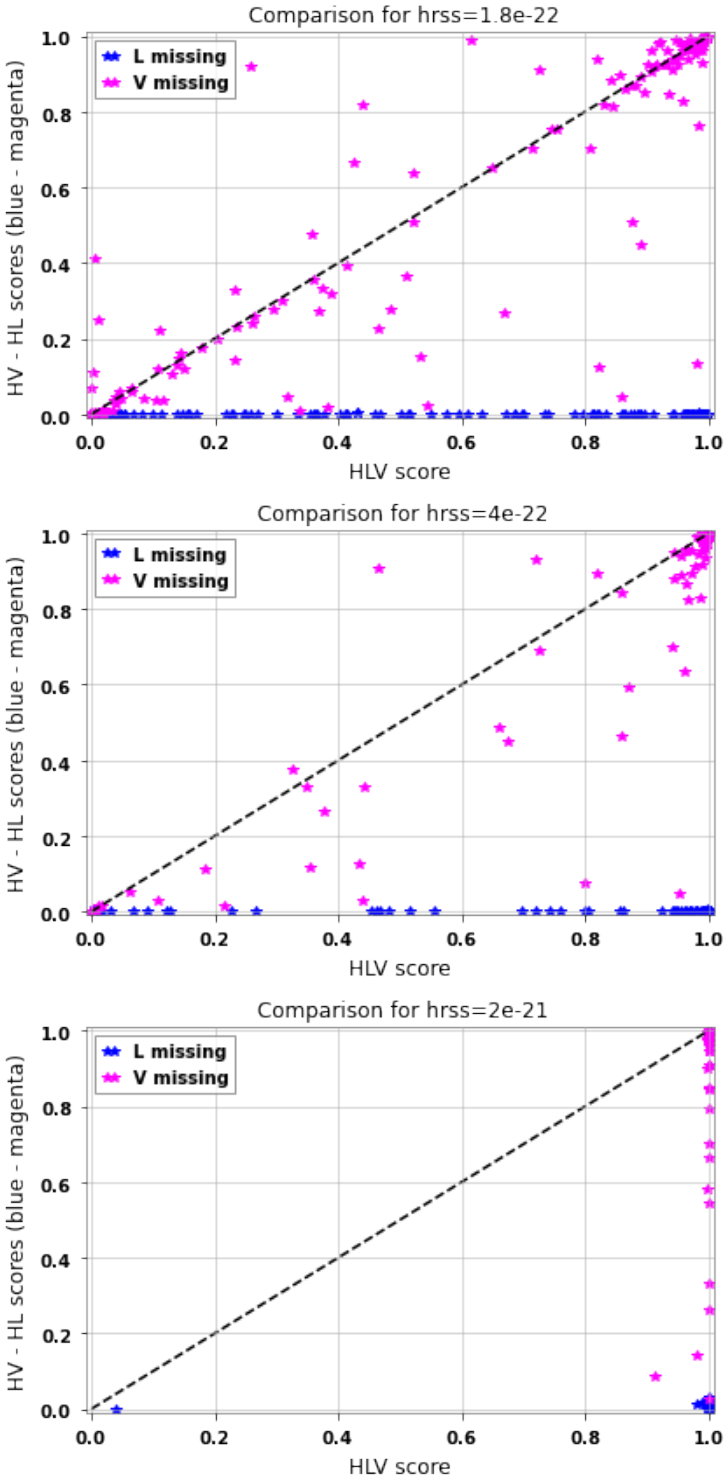

Figure 6.1: Test of the importance of the L and V detectors to the score assigned to 200 WNB injections by the three-detector CNN model. Horizontal axis: ordinary HLV score. Vertical axis: score assigned to the injection when the injection data stream for L ("L missing" - blue) or V ("V missing" - magenta) is replased by zeros. Top plot is for injections with h_{rss} value of $1.8 \times 10^{-22}\mathrm{Hz}^{-0.5}$ which corresponds to the 50% detection efficiency for this signal type, the middle plot value corresponds to 90% and the bottom plot is a very high value for reference $2 \times 10^{-21}\mathrm{Hz}^{-0.5}$.

6.2 O1: The first observing run analysis

During the first observing run there have been three gravitational wave detections [6] including the first confirmed gravitational wave signal ever detected: GW150914 observed at 09:50:45 UTC [92], which was the merger of two black holes with masses 36 $M_{\odot}$ and 29 $M_{\odot}$, SNR of 23 and luminosity distance of 420 Mpc. The other two signals were GW151226 at 03:38:53 UTC [93] and GW151012 at 09:54:43 UTC [94] had pairs of masses $14-7$ $M_{\odot}$, $23-13$ $M_{\odot}$, SNR of 13-9.7 and luminosity distances of $850-1600$ Mpc respectively. Those events were detected using two independently implemented match filter analyses, PyCBC [95, 86] and GstLAL[96, 97, 98]. There were no non-BBH signals [99, 100] detected in O1 and the only unmodelled search was that of cWB [7]. O1 run spanned from the 12th of September 2015 until the 19th of January 2016. This was the first observing run of Advanced LIGO. During this run only the two LIGO detectors in Hanford (H1) and Livingston (L1) were online. From those 130 (calendar) days, an equivalent of 75 days were in analysis mode for H1 and 65 days for L1. The times we could utilise for this search were the coincident times of the two detectors which were 48 days of data. The sensitivity of both detectors during O1 is shown in Figure 6.2.

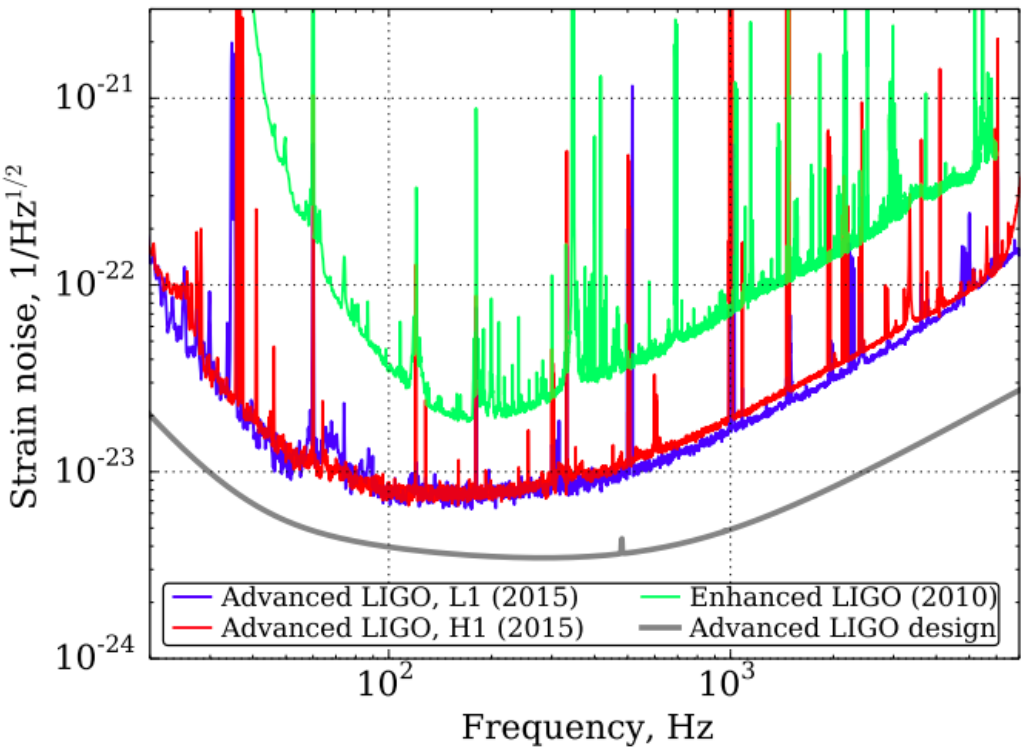

Figure 6.2: The sensitivity of LIGO detectors during O1 along with the final design sensitivity for Advanced LIGO and also the sensitivity of the last science run of the initial LIGO detectors, S6 [2]

In this run we have only two detectors and to actually run a search with our models we will have to add strain data in the Virgo input that is made from simulated Gaussian noise that follows the exaggerated Virgo sensitivity curve that we also used for the training data. We described the process in section 6.1. For the false alarm rate during the time of the search we used the real H1 and L1 data and applied time-lags to create independent noise segments of equivalent duration of 10 years. We present the false alarm rates for O1 in Figure 6.3.

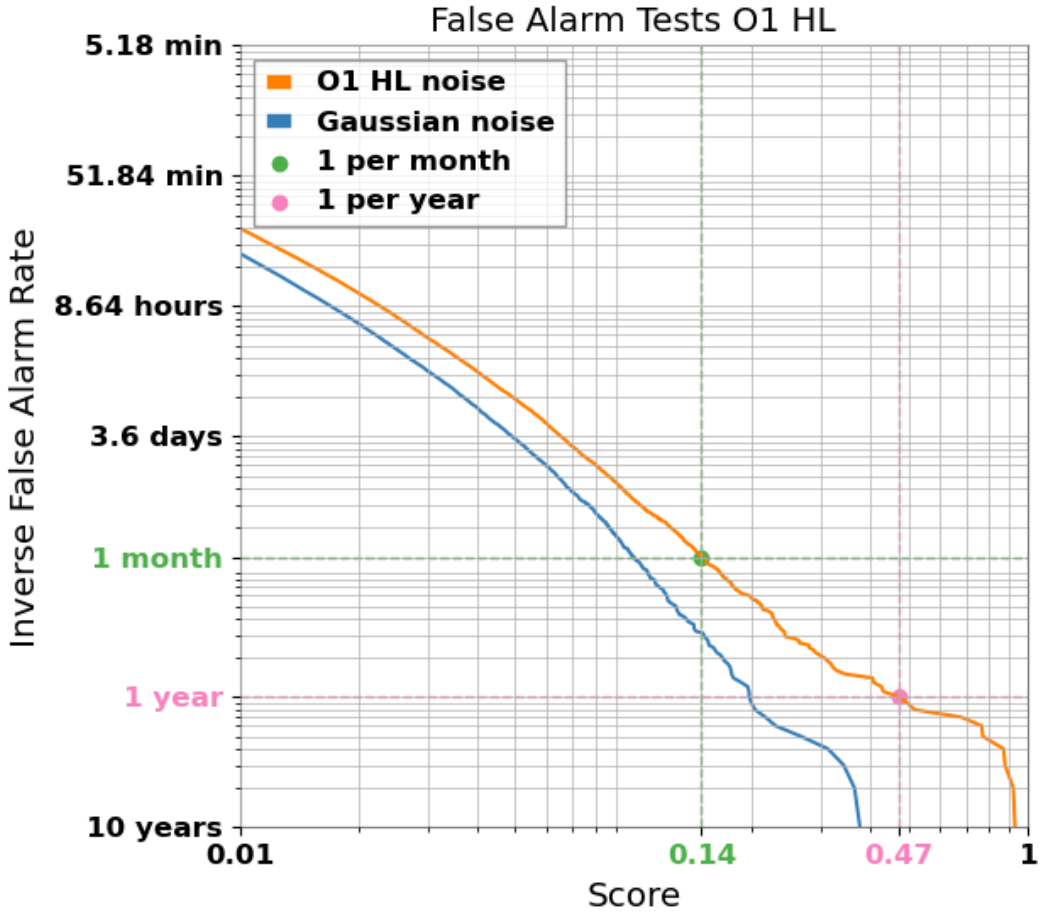

Figure 6.3: The false alarm rate of the HL network in comparison with the false alarm rate of Gaussian noise during O1 observing run. The score thresholds of 0.14 and 0.47 false alarm rates of 1/month and 1/year are also shown respectively.

We choose the false alarm rate of 1 per year, which corresponds to score threshold score value of 0.47. Based on that threshold we will calculate the expected efficiencies to other known various waveform morphologies. In Figure 6.4 we present the efficiency curves for the main groups of waveforms as described in section 4.2. During the first observing run we had three gravitational wave events, including **GW150914**, the first gravitational wave detection. Our search on the first observing run, had only one trigger and it was on **GW150914**, with a score 0.986 which corresponds to a false alarm smaller than 1 per 10 years. Our loudest false alarm

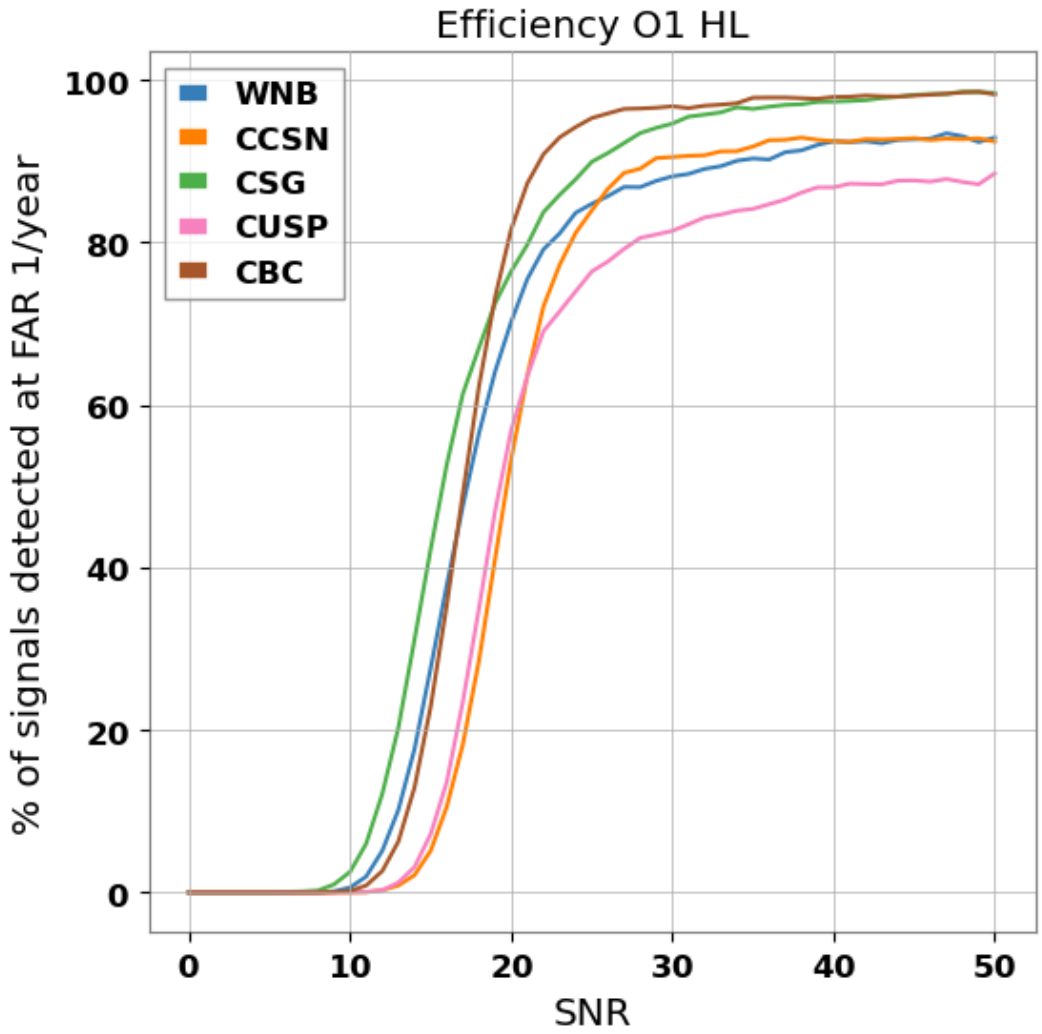

Figure 6.4: The detection efficiency of the MLy pipeline algorithm as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O1 observing run.

Event	UTC time	PyCBCB	GstLAL	cWB	MLy*
GW150914	09:50:45.4	23.6	24.4	25.2	0.986
GW151012	09:54:43.4	9.5	10.0	...	0.002
GW151226	03:38:53.6	13.1	13.1	11.9	0.002

Table 6.1: The official transient search pipelines for O1 [6] and their network SNR as reported from PyCBC, GstLAL and cWB. In comparison we show the score of MLy pipeline that correspond to detection. MLy pipeline has a threshold of 1/year (0.47) and events below that threshold are not present here.

in the 10 year test had score 0.931. The other two gravitational wave events were not detected. These results are consistent with the efficiency on the MLy pipeline to detecting BBH injections. From Figure 6.4 we see that the detection probability

at SNR 13(10) is $\sim$ 10%($\sim$ 1%), while at SNR 24 it is $\sim$ 93%. These estimated efficiencies are consistent with MLy detecting the GW150914 event.

Finally we calculated the efficiency thresholds for specific gravitational wave signals used in the all-sky search for short gravitational-wave bursts in the first Advanced LIGO run [7]. In Figure 6.5 we present the efficiency curves of those waveforms from which we inferred their 50% detection efficiency thresholds, and in Table 6.2 we present the 50% detection efficiency thresholds in comparison to what cWB inferred for the first observing run [7] for iFAR of 1 year. It is important to note that the G2D5 and SGL152Q8D9 injection sets were linearly polarised, while our training set consisted exclusively of unpolarised signals (WNBs with equal amplitudes in the two polarisations but no relation between the signals except duration and frequency parameters). We believe this to be the reason for MLy's poor limiting performance at high amplitudes for these injection sets.

In the cWB O1 search paper [7], the results are in FAR 1/100 years. To be able to compare our results we need the efficiencies for two orders of magnitude higher false alarm rate. For the next two subsection for O2 and O3 we managed to produce this calculation with the help of a member of cWB team, Shubhanshu Tiwari. Although unfortunately we were not able to produce the 1 per year results for O1. We will use the ratio between 1 year and 100 years of 50% efficiency thresholds of O2 HL analysis as a way to calculate the cWB thresholds in table 6.2 for O1. This is a rough estimate.

Performance comparison on the O1 data set		
Morphology	cWB [7] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	8.3	4.0
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	N/A	2.4
f_0=153 Hz, Q=8.9	1.3	1.5
f_0=235 Hz, Q=100	3.8	1.4
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	2.0	1.8
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	1.7	1.6

Table 6.2: The values shown refer to h_{rss}, in units of $10^{-22}\mathrm{Hz}^{-1/2}$, at the 50% efficiency threshold. Comparing waveforms from cWB results [7] and MLy pipeline at FAR of 1 per year. We restrict the comparison to injections with frequencies and durations that are in the range for which we have trained the MLy models (duration $<$ 1s , frequencies in the 20 Hz - 480 Hz band) The cWB thresholds are only an estimation base on the behaviour in O2 observing run.

Based on the results in Table 6.2, we cannot claim (and we do not) that the MLy pipeline is more sensitive than cWB in O1. However we can see that out of almost

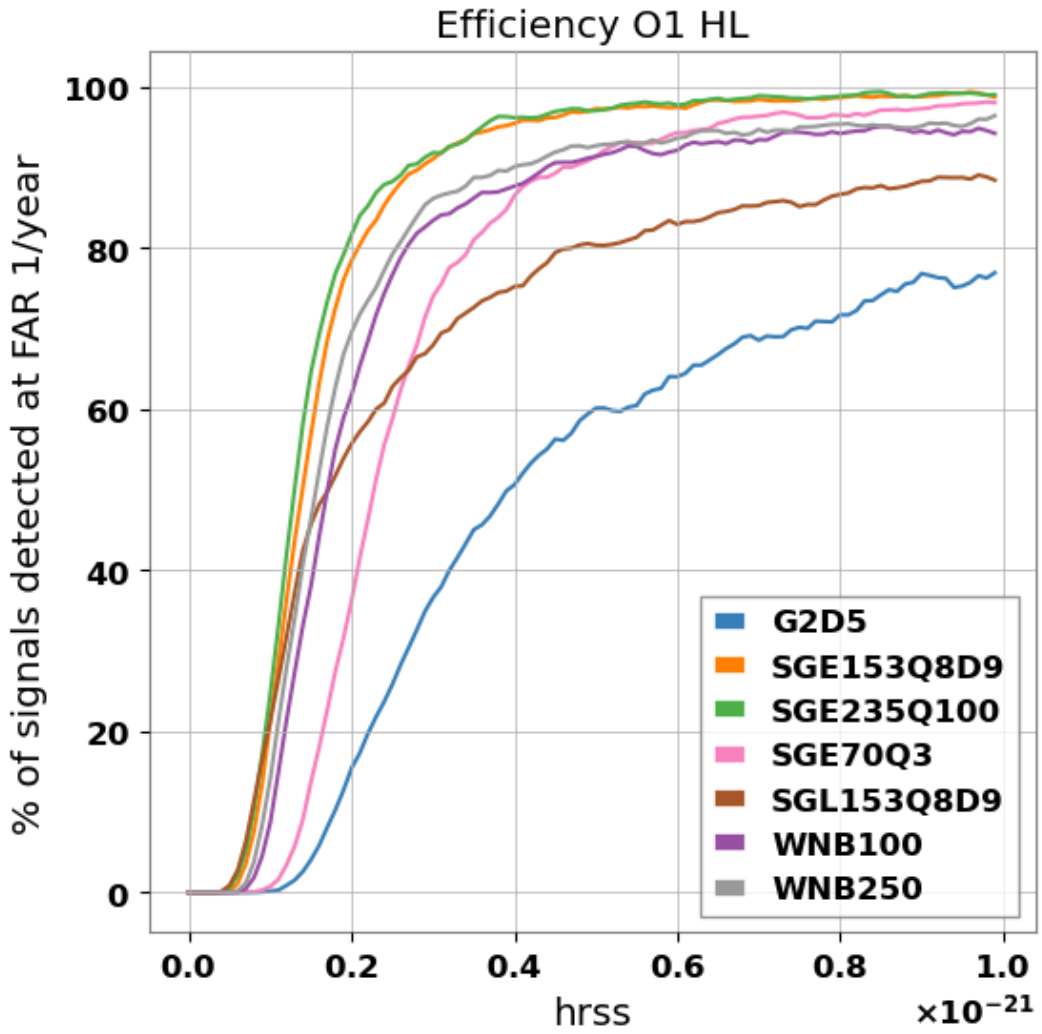

Figure 6.5: The detection efficiency of the MLy pipeline as a function of the h_{rss} for generic sub-second burst signals and for a false alarm rate of 1 per year, during the O1 observing run.

all tested morphologies , any signal loud enough to be confidently detected by the cWB, would have shown up in the MLy search as a trigger loud enough to trigger an alert under or close to the alert threshold of FAR $<$ 1/year.

6.3 O2: The second observing run analysis

During the second observing there have been detected 7 BBH signals (GW170104 [101], GW170608 [102], GW170729 [103], GW170814 [104], GW170809, GW170818, GW170823 [6]) and the first binary neutron star merger signal (BNS) GW170817 [105]. Those events were detected using two independently implemented analyses, PyCBC [95, 86] and GstLAL [96, 97, 98, 106]. There were no burst or long duration signals as reported by [107, 108] and the unmodelled search of cWB [5].

The first BNS signal GW170817 was also the event that was the first example of multi-messenger astronomy utilising gravitational wave astronomy.. This merger not only emitted a gravitational wave signal with a duration $\sim 60\ s$, but also had an electromagnetic counterpart that was identified and observed by a collaboration of teams from different observatories around the world [109, 110]. The implications of that detection affected different parts of physics, from particle physics and the equation of state of a neutron star [75], to the Hubble constant [111] and testing general relativity [112]. The existence of such an event demonstrated the need for fast reactions to events with a possible electromagnetic counterpart, such as join super-novae and any other unmodelled signal we try to detect with the MLy pipeline.

The second observing run spanned from the 1 November 2016 until 25 August 2017. At the beginning of the run only LIGO Hanford and Livingston detectors were in observing mode. During the last month of the run, on 1st of August 2017, Virgo joined for the first time creating the first three detector network.

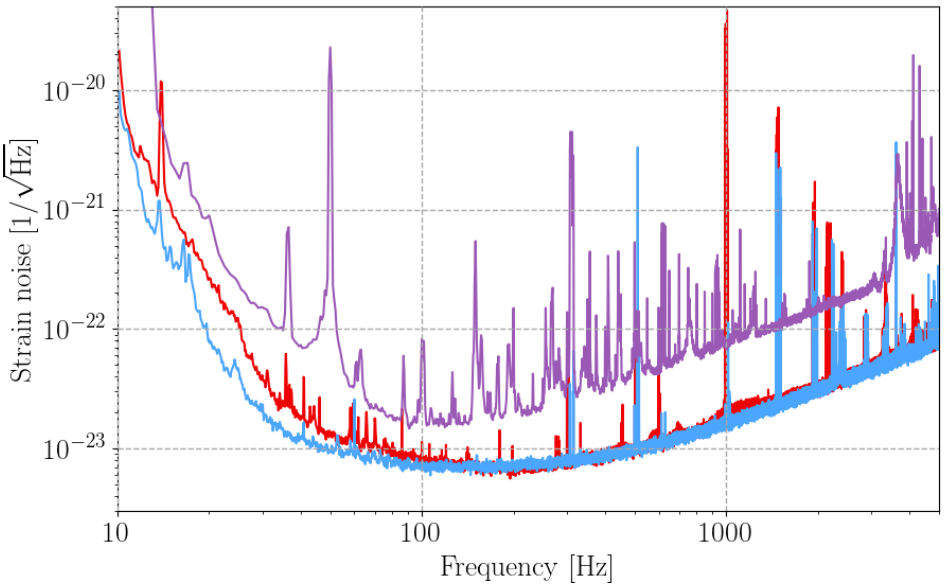

Figure 6.6: Sensitivity curves of the three detectors H1 (red), L1 (blue) and V1 (purple) during the O2 observing run in August 2017. Plot taken from [3].

As can be seen from Figure 6.6, H1 and L1 had very similar sensitivities while Virgo was 2 to 3 times less sensitive depending the frequency. H1 and L1 were in observing mode at the same time on around 75% (118 days) of the total time they were on. Virgo was in observing mode 85% of the 25 days it was on, and 51% of that time it was in triple coincidence with the H1 and L1 detectors providing 15 days of three detector data.

6.3.1 HLV analysis

For the HLV analysis we used all the coincident times of H1 L1 and V1. We generated an equivalent of 10 years of noise out of those 15 days to calculate our false alarm rate. We have shown in the previous chapter, in Figure 5.6 the false alarm rates down to 10 years for the coincident times of three detectors in O2. We use as a detection threshold the false alarm rate of once per year and we find it to be 0.30. For this analysis there were no zero-lag triggers that exceeded that threshold.

The efficiency expected for known groups of waveforms is shown in Figure 5.8. During the second observing run we had six gravitational wave events, listed in Table 6.3 that happened while all three detectors were active. None of these events were

Event	UTC time	PyCBCB	GstLAL	cWB	MLy
GW170729	18:56:29.3	9.8	10.8	10.2	0.002
GW170809	08:28:21.8	12.2	12.4	...	$< 10^{-3}$
GW170814	10:30:43.5	16.3	15.9	17.2	$< 10^{-3}$
GW170817	12:41:04.4	30.9	33.0	...	0.003
GW170818	02:25:09.1	...	11.3	...	$< 10^{-3}$
GW170823	13:13:58.5	11.1	11.5	10.8	$< 10^{-3}$

Table 6.3: The transient events for O2 present in all three detectors [6], and their network SNR as reported from PyCBC, GstLAL and cWB and MLy pipeline. *MLy pipeline has a threshold of 1/year and events below that threshold are not present here.

detected from our pipeline or they had an insignificant score and hence very high false alarm rate. The non-detection of those BBH events is expected given their SNR values as estimated from the three pipelines in Table 6.3, are lower than 19 which is the calculated 50% threshold on BBH signals, shown in Figure 5.8. The highest SNR event GW170817, was a loud BNS merger and was not detected because its SNR was spread over nearly 60 s and our pipeline is not trained on long-duration signals. Hence we didn't expect to detect it.

6.3.2 HL analysis

For the HL analysis we used all the coincident times of H1 and L1. During O2 the coincident times of H1 and L1 were 118 days and from them we calculated an

equivalent of 10 years of noise. H1 and L1 have very similar sensitivities. We remind here that the HL analysis uses all HL data and in the place of Virgo detector input it feeds Gaussian noise the same distribution as that used for training as described in the beginning of this chapter.

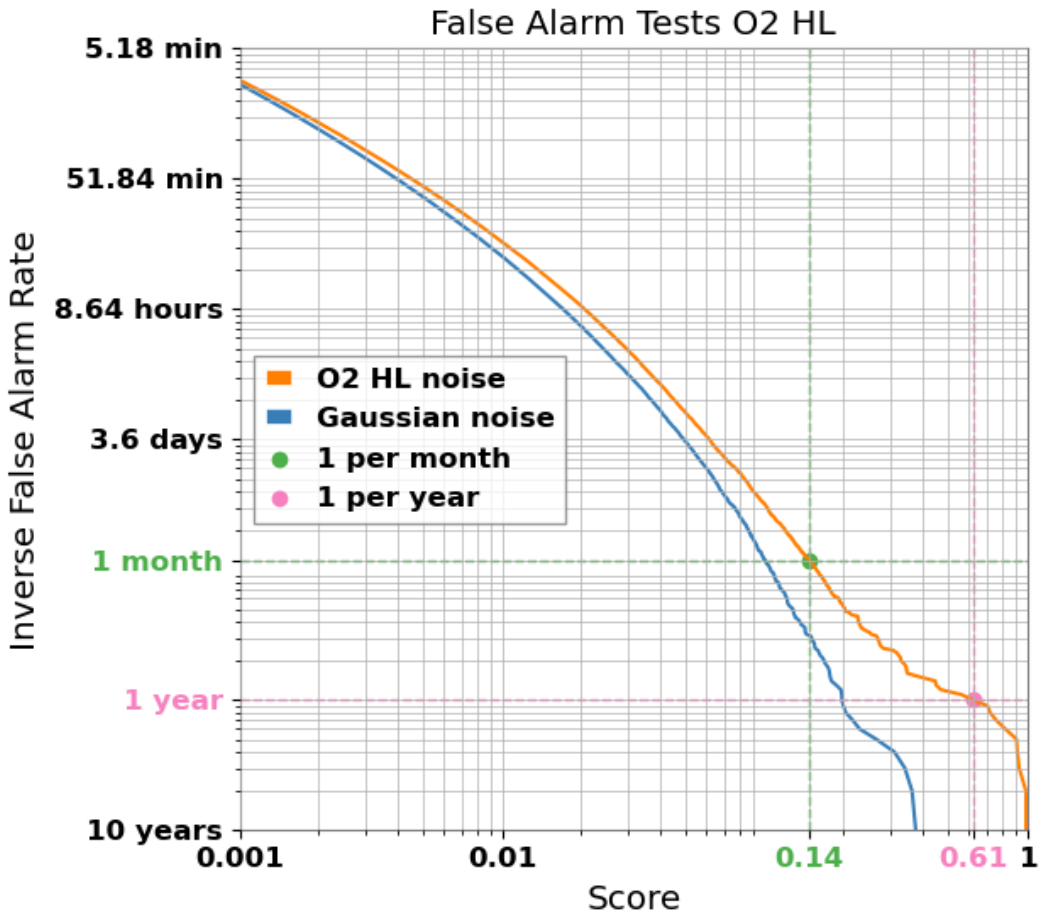

Figure 6.7: The false alarm rates of the HL network in comparison with the false alarm rates of Gaussian noise during O2 observing run. The score thresholds of 0.14 and 0.61 false alarm rates of 1/month and 1/year are also shown respectively.

From Figure 6.7 that shows the false alarm rates, we select as detection threshold 0.61 that represents the threshold for once per year. In this search we found one trigger that exceeded the threshold at GPS 1168270760.375 (UTC: 2017-01-12 15:39:02.375). This time is the start time of the one second interval that provided the highest score. The specific trigger was the loudest glitch of the hour and it was identified by GravitySpy [35] as a loud Koi-fish and a blip glitch [113] in very close coincidence that is within the time delay of the two detectors.

The efficiency expected for our sample groups of waveforms for the HL search is shown in Figure 6.8. During the second observing run we had six gravitational wave events that happened while all detectors were observing (shown in table 6.3 and two events while only the two LIGO detectors were active. In table 6.4 we present the

results for those two events.

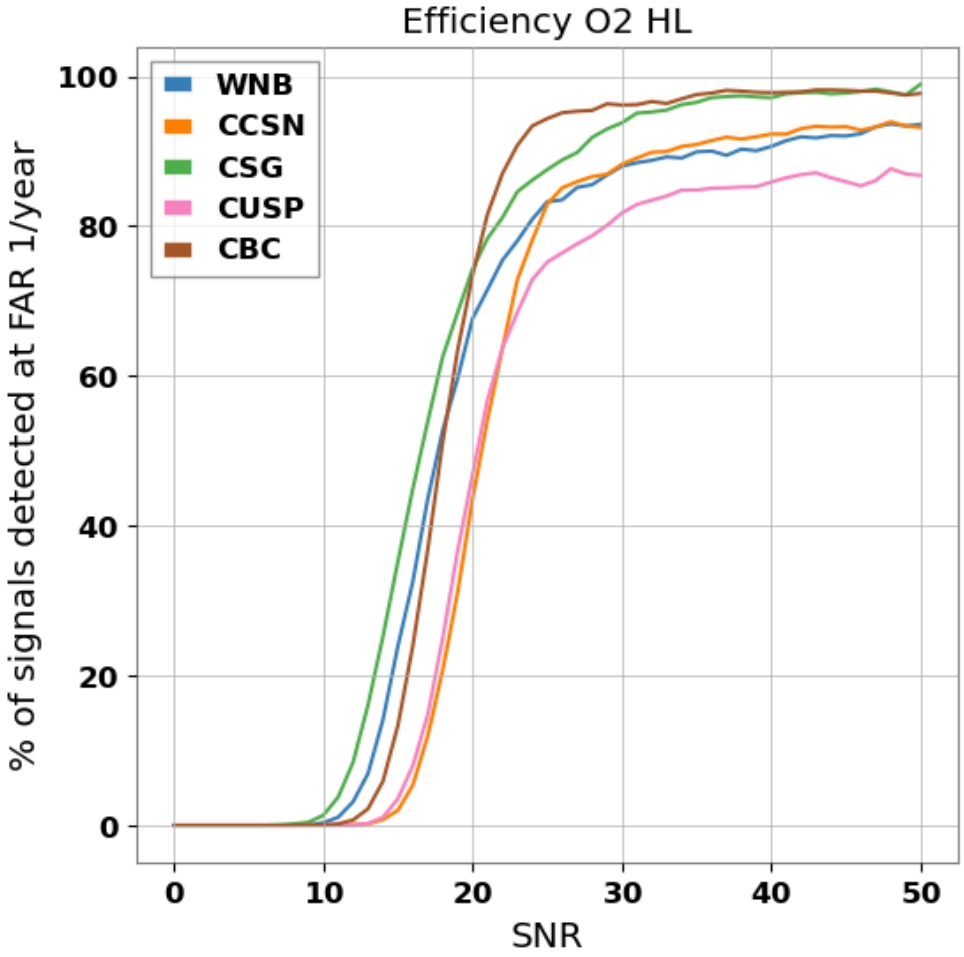

Figure 6.8: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O2 HL observing run.

Event	UTC time	PyCBCB	GstLAL	cWB	MLy*
GW170104	10:11:58.6	13.0	13.0	13.0	$< 10^{-3}$
GW170608	02:01:16.5	15.4	14.9	14.1	0.015

Table 6.4: The official transient events for O2 present only in detectors H1 and L1, and their network SNR as reported from PyCBC, GstLAL and cWB [6].*MLy pipeline has a threshold of 1/year and events below that threshold are not present here.

Our O2 HL search didn't detect any of the events in Tables 6.3 and 6.4. The predicted 50% detection threshold for BBH signals is SNR 19 as it is shown in Figure 6.8. All BBH signals in O2 had an SNR $\leq$ 19. With a rough estimation of their

probabilities of not being detected based on Figure 6.8 and by multiplying all of these probabilities, we find that the probability of not detecting any signal based on our efficiency is $\sim 50\%$. Therefore we cannot verify the consistency of our efficiency tests with the detected events for the O2 results.

Finally we also calculated the 50% detection thresholds for generic sub-second burst signal groups, shown in Figure 6.9. The 50% detection thresholds are shown in the table 6.5. It is important to note that the G2D5 and SGL152Q8D9 injection sets were linearly polarised, while our training set consisted exclusively of unpolarised signals (WNBs with equal amplitudes in the two polarisations). We believe this to be the reason for MLy's poor limiting performance at high amplitudes for these injection sets.

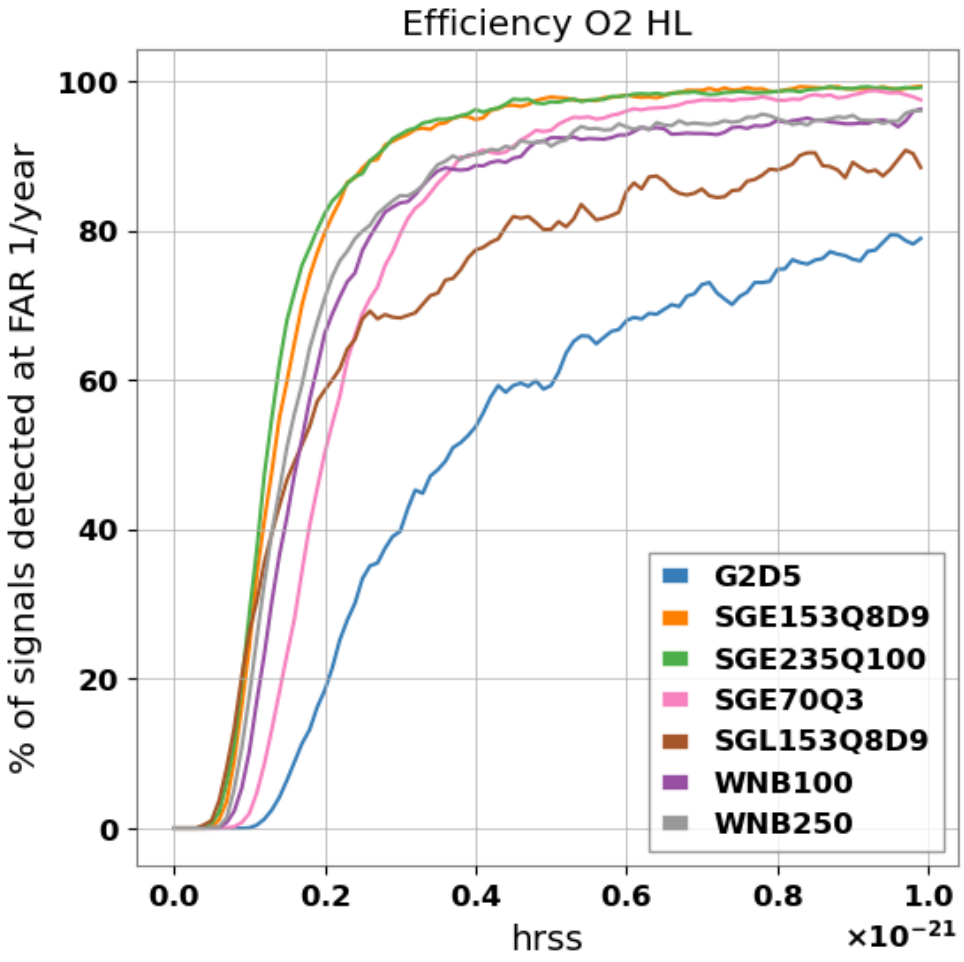

Figure 6.9: The HL detection efficiency of the MLy pipeline as a function of the h_{rss} for generic sub-second burst signals and for a false alarm rate of 1 per year, during the O2 observing run.

The results in Table 5.1 and 6.5, show that MLy pipeline is not better than cWB.

Performance comparison on the O2 HL data set		
Morphology	cWB [7] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	2.8	3.7
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	1.5	2.1
f_0=153 Hz, Q=8.9	1.3	1.4
f_0=235 Hz, Q=100	0.9	1.3
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	1.2	1.7
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	1.4	1.6

Table 6.5: The values shown refer to h_{rss}, in units of $10^{-22}\mathrm{Hz}^{-1/2}$, at the 50% efficiency threshold. Comparing waveforms from cWB results [7] and the MLy pipeline at FAR of 1 per year. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team. We restrict the comparison to injections with frequencies and durations that are in the range for which we have trained the MLy models (duration $<$ 1s , frequencies in the 20 Hz - 480 Hz band)

The results show the potential of the pipeline even if the FAR is increased in both HL and HLV searches. The HL analysis comparison to cWB is slightly better than the HLV analysis, which could be due to the fact that cWB analysis is also just HL and not HLV.

We can also observe an increase in false alarm rate from O1 results. Along with that increase, we have an increase in our threshold. The most obvious reason for that increase is the occurrence of glitches. O2 has more glitches that O1 because the noise level is higher. Hence glitches that were not identified in O1 are seen in O2. Moreover, the more glitches in the noise the higher the chance for them to appear close to coincidence when we apply time-lags. In the end of this analysis we will also discuss why does this affect our analysis with more details.

6.4 O3: The third observing run analysis

During the third observing run the amount of gravitational wave detections increased significantly. During its first half we had 47 CBC events [8] that we also show in Table 6.10. We had a lot of interesting events. Starting by GW190425 [114], a CBC signal with total mass $\sim 3.4 M_{\odot}$. The components of this merger could be neutron stars or black holes and there was no electromagnetic counterpart. We also had two BBH mergers with quite asymmetric masses. GW190412 [115] was a black hole with mass 30 $M_{\odot}$ with a companion of only 8 $M_{\odot}$, and GW190814 [116] a 23 $M_{\odot}$ black hole with a companion of 2.5 $M_{\odot}$ which could be the first black hole - neutron star merger ever detected. The last special event for the first part of O3 is the largest BBH ever detected GW190521 [117], making it's post merger black hole to be the first intermediate mass black hole (IMBH) to be observed. Those were detected using four matched-filter independently implemented analyses, PyCBC [118, 86, 119, 120, 121], GstLAL [98, 106, 122], MBTA [123] and SPIIR [124]. There were no non-BBH signals [125, 126] identified in the unmodelled search. The only unmodelled search pipeline was cWB [4].

The third observing run started on 1 April 2019 and ended on 27 March 2020, it was stopped earlier than expected due to the Covid-19 pandemic. During this run LIGO Hanford, LIGO Livingston and Virgo detector were all online. In the middle of the run, from 1st of October to 1st of November 2019 there was a commissioning break. This, and the fact that all three detectors were online, gives us the opportunity to separate the search into "O3a" and "O3b" periods so that the background representation is more accurate. During summer months we expect a less noisy detector due to the fact that the bad weather affects the detector background noise. Many of the LIGO-Virgo collaboration searches were broken up this way too [8, 127, 4]. In Figure 6.10 we show the relative background levels of noise for all three detectors during O3. For each part of the O3 observing run we run separate HL and HLV analyses.

6.4.1 O3a HLV analysis

The first part of the third observing run, O3a lasted from the 1 of April 15:00 UTC to 1 October 2019 15:00. Each detector was online for 130.2, 138.5 and 139.5 days in total for H1, L1 and V1 respectively. Out of those times only 81 days were of triple coincidence between H1, L1 and V1.

First step of the analysis is again to calculate the 10 year equivalent false alarm rates of those six months of O3a. In Figure 6.11 we see the false alarm rates for O3a for the HLV network. It is obvious that the false alarm rate is elevated compared to O2 and O1. We are going to discuss later that this is due to having selected the optimal models based on their FAR performance for the O2 HLV data set, without having tested their performance on O3. Based on this our threshold score will be

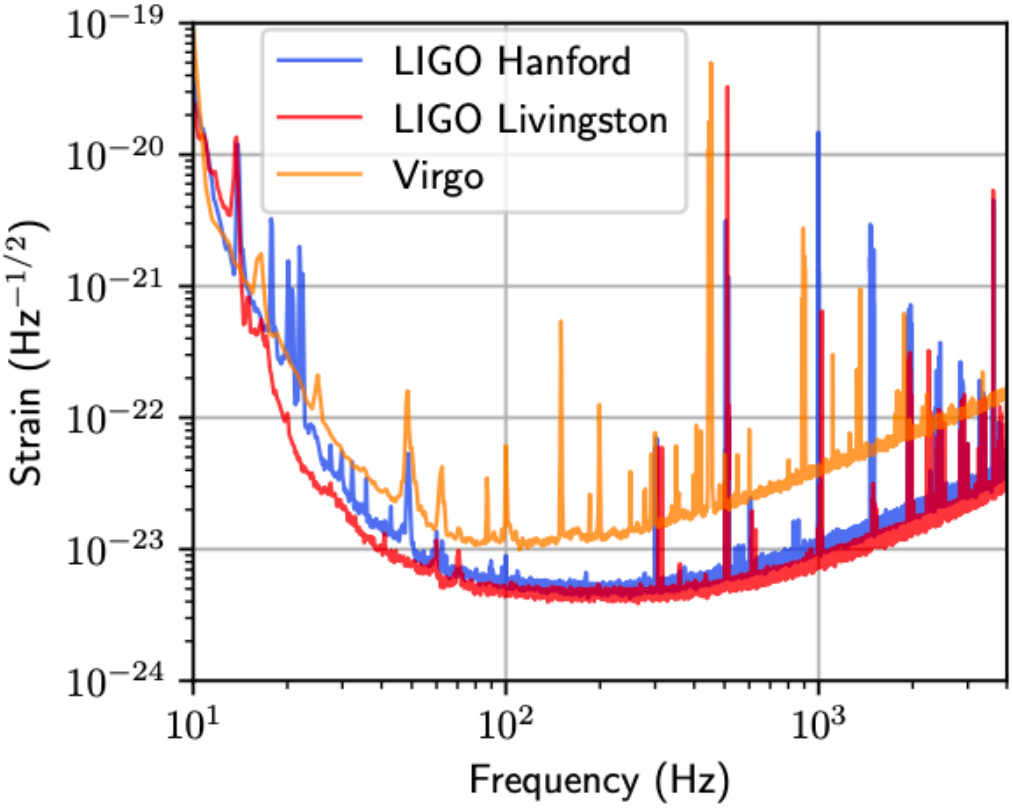

Figure 6.10: The representative sensitivity of LIGO-Virgo detectors during O3a. Plot taken from [4].

0.96 and we can use that to define triggers in our search. Out of the six months of O3a we found one trigger on GPS 1253732203.875, which was caused by two coincident glitches, a Koi-fish glitch in Hanford and a Tomte glitch in Livingston as classified by GravitySpy [35, 113]. We believe that the reason that the model reacted to them was due to their similar peak frequency (42 , 49 Hz).

In Figure 6.12 we see the performance on the selected waveforms for the three detector case for O3a. It is obvious that the performance is poorer compared to O1 and O2 with the 50% detection efficiency thresholds to be lower than the previous observing runs and also the SNR values above 50 are still at $\sim 90\%$. This is due to the higher score threshold for FAR=1/year.

In addition to that we tested all the official triple coincidence events for O3a that are shown in Table 6.10 [8]. Out of the 27 triple coincidence events we had four sub-threshold detection events that corresponded to those events. Sub-threshold events are defined as all the known gravitational wave events with scores at least once per month and less than once a year. In table 6.6 we present those events in one table along with the scores of the individual models. It is observed that Model 2 tends to provide lower score than Model 1. Model 2 works are a regulative model that reduces the confidence of Model 1.

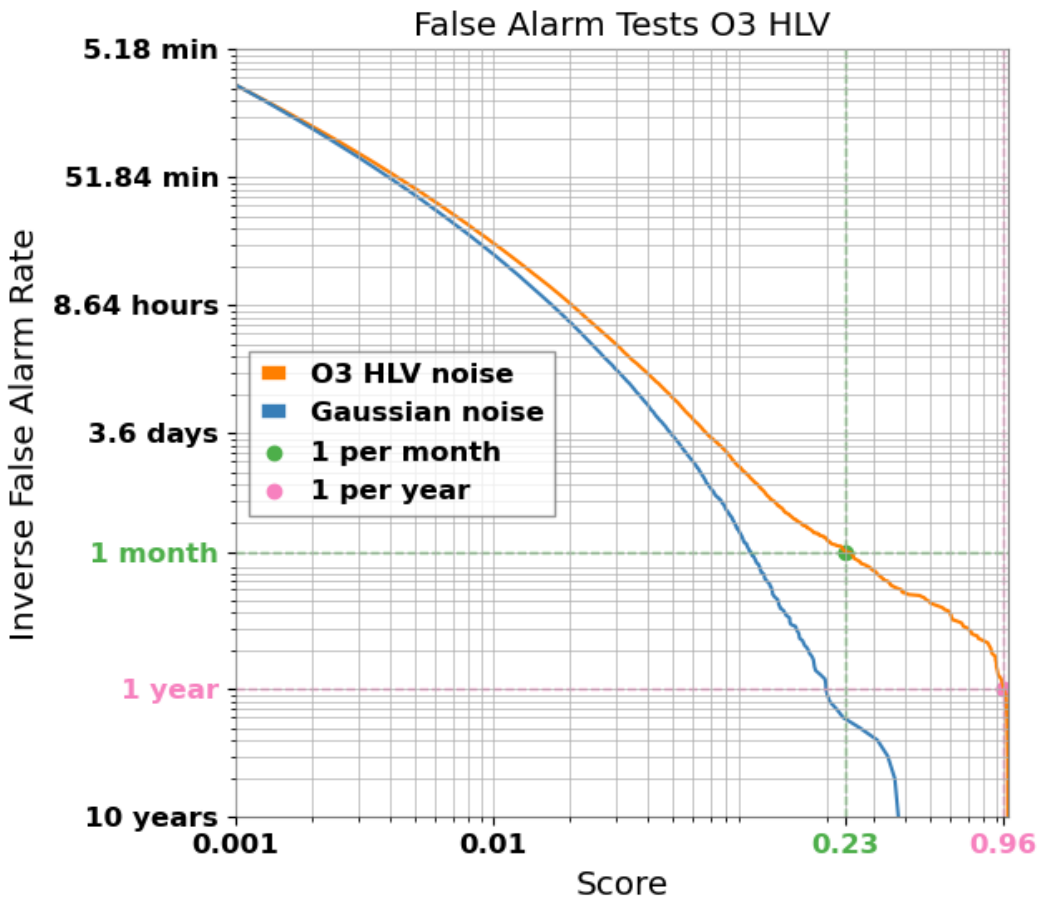

Figure 6.11: The false alarm rates of the HLV network in comparison with the false alarm rates of Gaussian noise for O3a. The score thresholds of 0.23 and 0.96 false alarm rates of 1/month and 1/year are also shown respectively.

Event Name	Model1	Model2	Final Score	iFAR(years)
GW190412	0.998	0.702	0.839	0.920
GW190519	0.993	0.639	0.634	0.720
GW190521	0.992	0.873	0.866	0.950
GW190828	0.997	0.283	0.282	0.170

Table 6.6: Scores and false alarm rates for sub-threshold official BBH events in O3a. We also show the scores for the two individual models.

We compare the performance of the model to the generalised short duration search for O3a [4]. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team. In Figure 6.13 we present the performance of all generic burst types used for the comparison and in Table 6.7 the direct comparison of the 50% efficiency detection thresholds for each burst class. We observe that the G2D5 and SGL152Q8D9 injection sets under-perform as they did in previous runs.

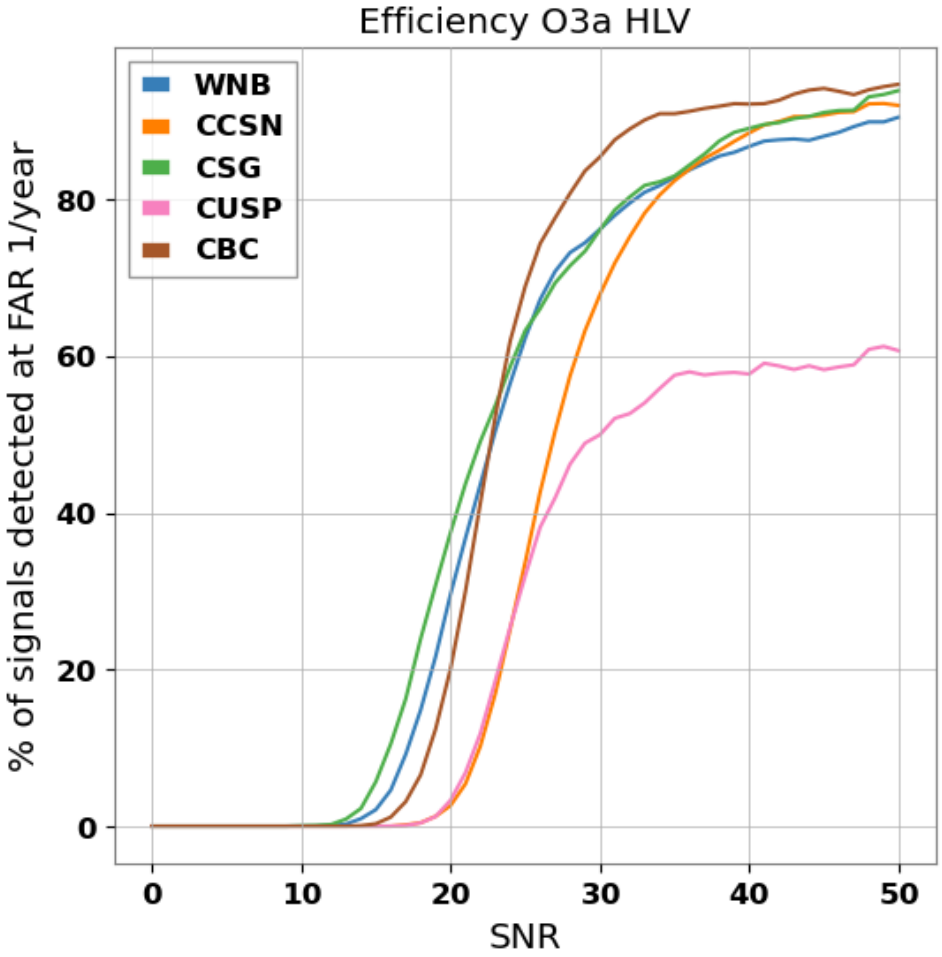

Figure 6.12: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O3a HLV observing run.

If we compare the performance of the signals from O1 in table 6.2 and O2 in table 6.5, we see that the 50% efficiency threshold is not significantly different than O3a. Although if we see the efficiency plot of various waveforms (figure 6.12) and the comparison waveforms (figure 6.13), we see that the performance in higher SNR values starts reducing. This suggests that higher SNR values are affected more from the FAR thresholds than smaller SNR values.

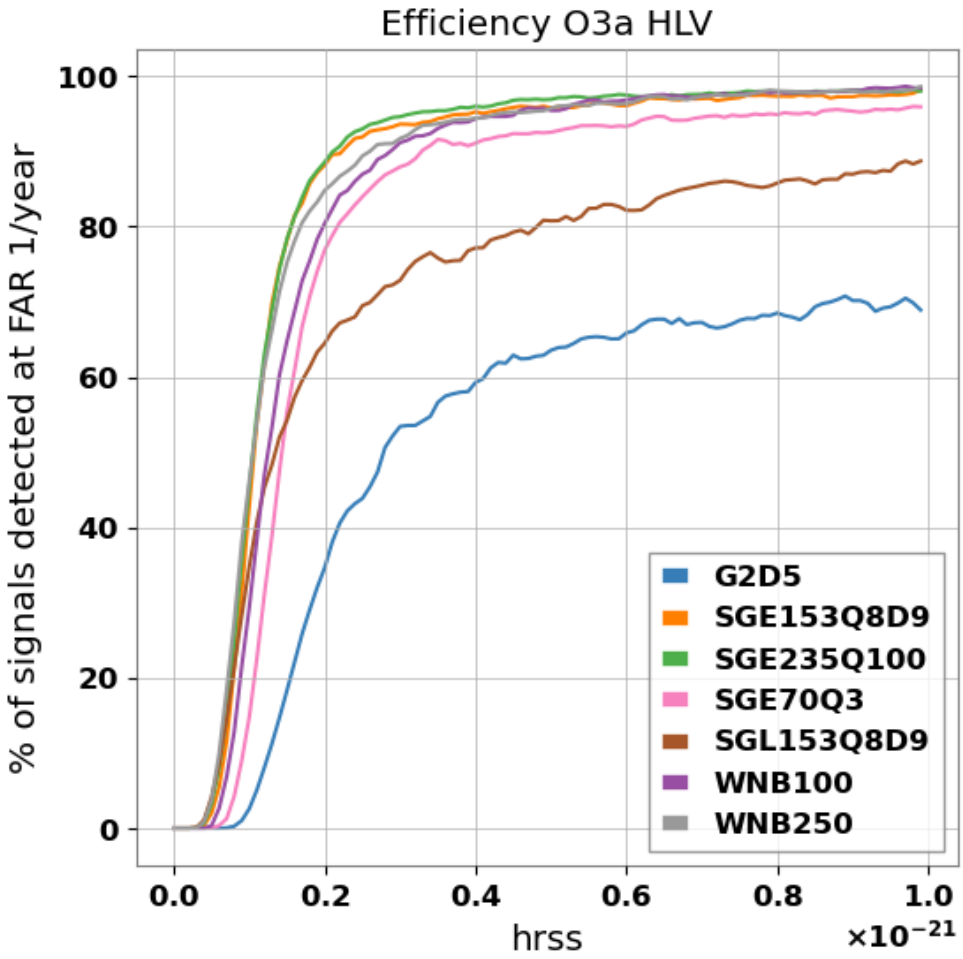

Figure 6.13: The HLV detection efficiency of the MLy pipeline as a function of the h_{rss} for generic sub-second burst signals and for a false alarm rate of 1 per year, during the O3a observing run.

Performance comparison on the O3a HLV data set		
Morphology	cWB [4] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	1.6	2.9
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	0.9	1.5
f_0=153 Hz, Q=8.9	0.6	1.2
f_0=235 Hz, Q=100	0.6	1.1
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	0.8	1.3
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	0.8	1.1

Table 6.7: Same comparison like table 5.1, but this time for O3a three detector search. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team.

6.4.2 O3a HL analysis

During O3a, for the two detector network analysis, we had 104 days of coincidence for H1 and L1. The first step of the analysis is to calculate the 10 year equivalent false alarm rates for the HL network using those 104 days.

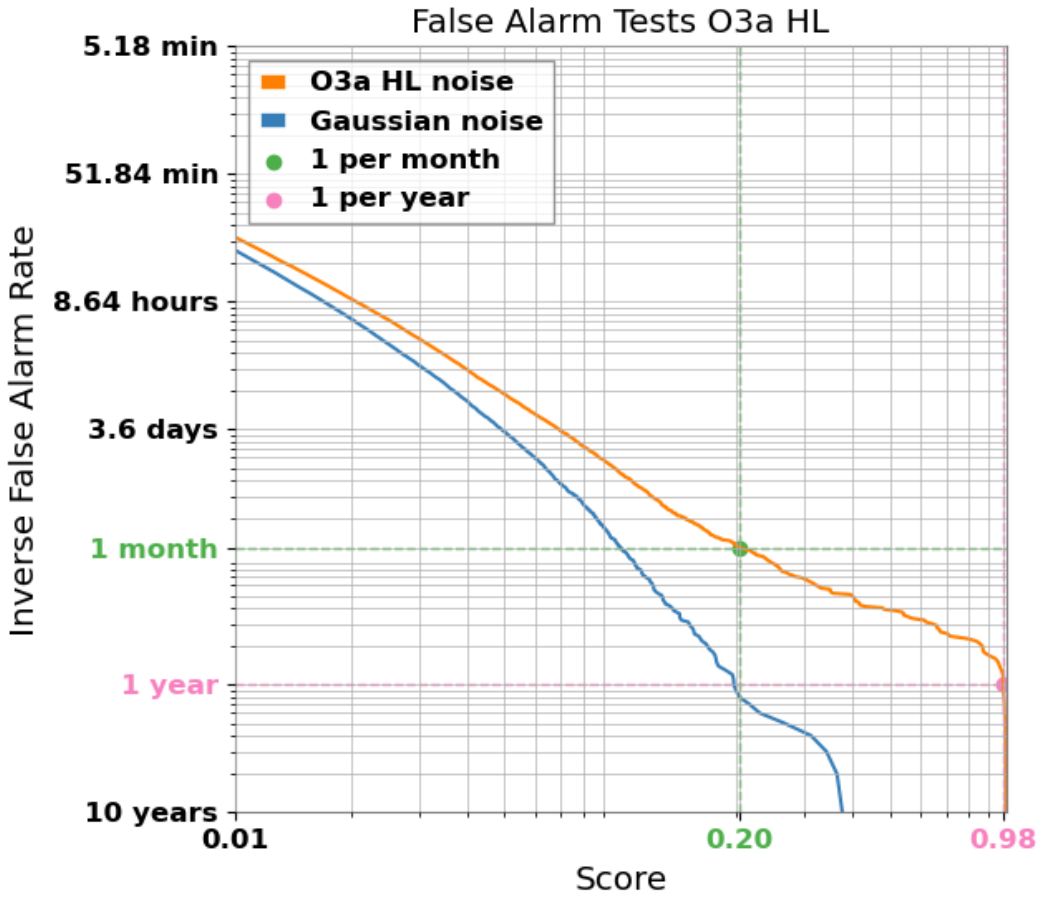

Figure 6.14: The false alarm rates of the HL network in comparison with the false alarm rates of Gaussian noise for. The score thresholds of 0.20 and 0.98 false alarm rates of 1/month and 1/year are also shown respectively.

In Figure 6.14 we see the false alarm rates for O3a for the HL network analysis. It is obvious that the false alarm rate is at high scores compared to O2 and O1 for the HL case too. Based on this our threshold score will be 0.96 and we can use that to define triggers in our search. Out of the six months of O3a we found one glitch trigger on GPS 1253732203.875, which were two coincident signals, a Koi-fish glitch in Hanford and Tomte glitch in Livingston according to GravitySpy [35, 113]. This glitch is the same we found with the O3a HLV analysis. We also see that the 1 per year threshold is much higher than it was in O2. We will discuss the reasons behind this in the end of this section.

In Figure 6.15 we see the performance on the other various waveform morphologies for the three detector case of O3a. It is obvious here too, that the performance

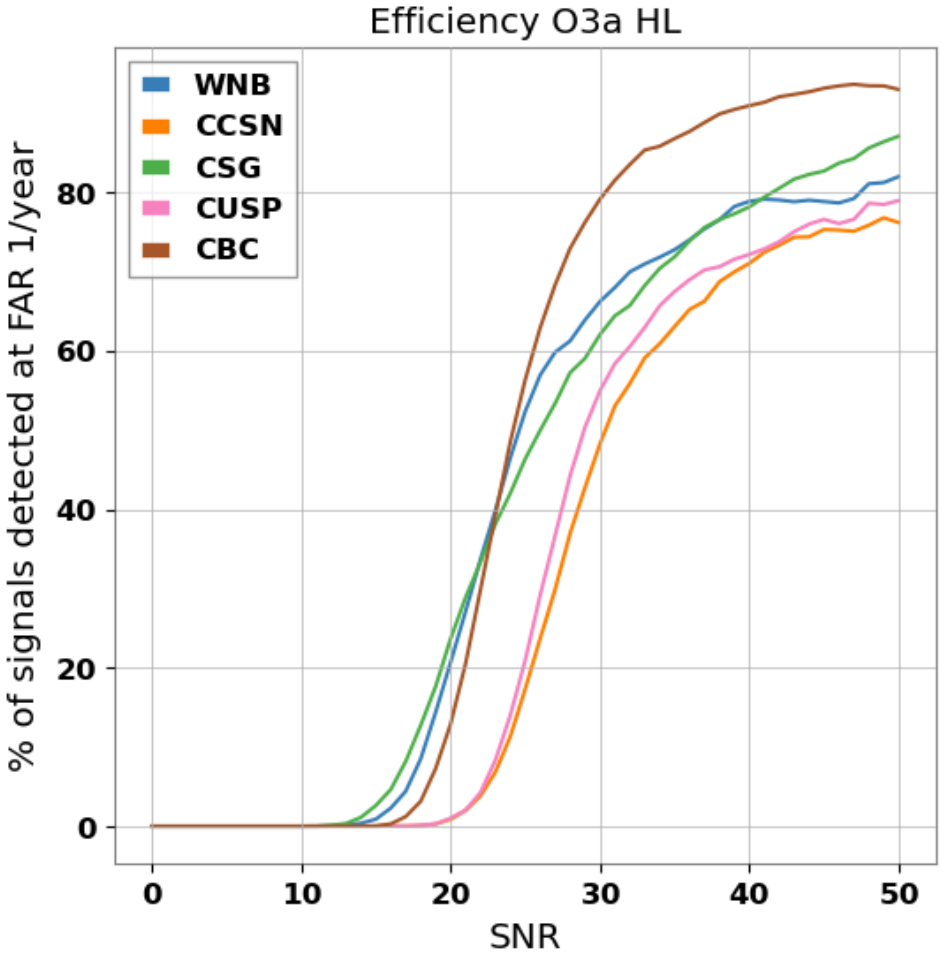

Figure 6.15: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O3a HL observing run.

is poorer compared to O1 and O2 with the 50% detection efficiency thresholds being higher and also SNR values above 50 are still at $\sim 90\%$. This is due to the higher score threshold for FAR=1/year.

We also tested all the double coincidence events on H1 and L1 for O3a that are shown in Table 6.10. Out of the 39 HL coincidence events we had five sub-threshold detection events that corresponded to those events. A reminder here that sub-threshold events we defined as all the gravitational wave detection events with score at least once per month and less than once a year. We see that all the sub-threshold events that are detected are the same with the HLV ones with the addition of one more, which wasn't loud enough in the HLV analysis.

In Figure 6.16 we present the detection efficiency of MLy for each of the generic burst types used for the comparison with cWB and in Table 6.9 the direct comparison

Event Name	Model1	Model2	Final Score	iFAR(years)
GW190412	0.999	0.849	0.848	0.910
GW190519	0.991	0.678	0.673	0.840
GW190521	0.989	0.877	0.877	0.930
GW190521b	0.999	0.956	0.956	0.970
GW190828	0.997	0.360	0.359	0.550

Table 6.8: Scores and false alarm rates for sub-threshold official BBH events in O3a HL analysis.

of the 50% thresholds for each burst class for MLy and cWB.

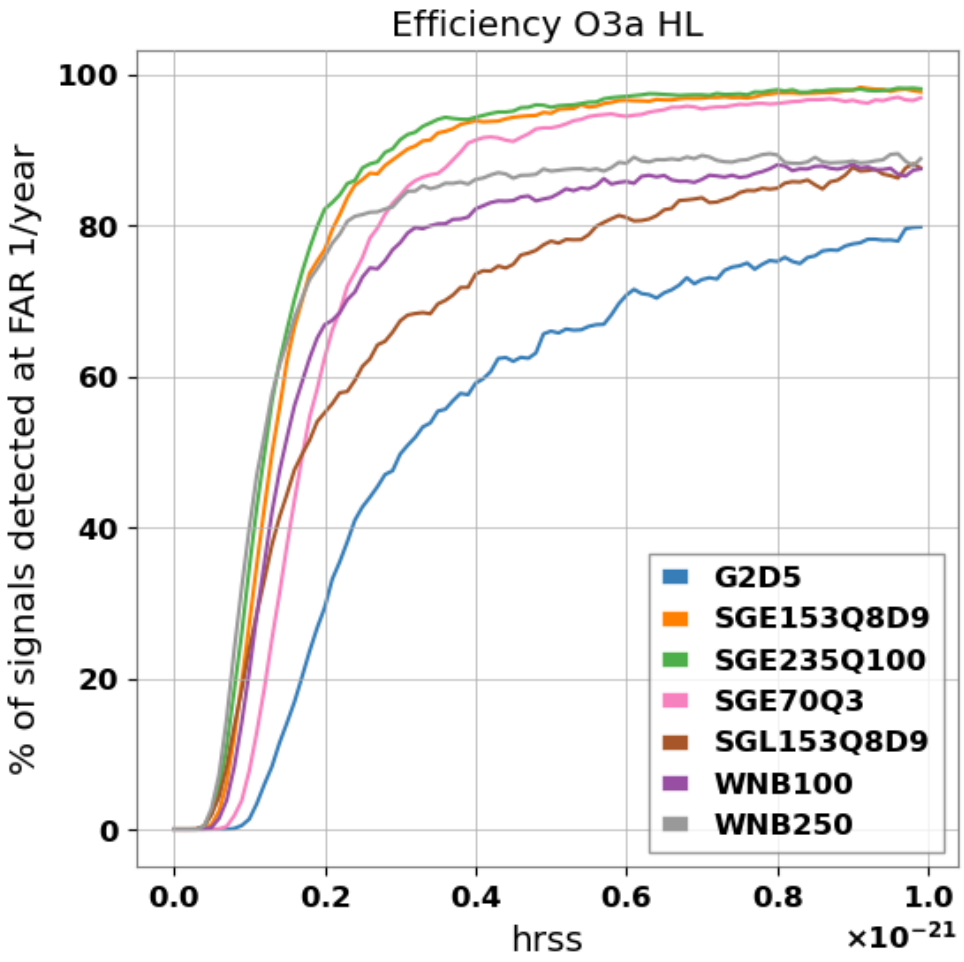

Figure 6.16: The HL detection efficiency of the MLy pipeline as a function of the h_{rss} for generic sub-second burst signals and for a false alarm rate of 1 per year, during the O3a observing run.

Performance comparison on the O3a HL data set		
Morphology	cWB [4] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	1.6	3.1
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	0.9	1.8
f_0=153 Hz, Q=8.9	0.6	1.4
f_0=235 Hz, Q=100	0.6	1.3
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	0.8	1.6
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	0.8	1.3

Table 6.9: Same comparison like table 6.5, but this time for O3a two detector search. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team.

Event Name	UTC time	Detectors	SNR	MLy HLV	MLy HL
GW190403	05:15:19	HL	8.0	N/A	$< 10^{-3}$
GW190408	18:18:02	HLV	14.8	0.078	0.206
GW190412	05:30:44	HLV	19.7	0.839	0.848
GW190413	05:29:54	HLV	8.6	$< 10^{-3}$	$< 10^{-3}$
GW190413b	13:43:08	HLV	10.0	$< 10^{-3}$	$< 10^{-3}$
GW190421	21:38:56	HL	10.6	N/A	0.002
GW190424	18:06:48	L	10.0	N/A	N/A
GW190425	08:18:05	LV	13.0	N/A	N/A
GW190426	19:06:42	HL	9.6	N/A	$< 10^{-3}$
GW190426b	15:21:55	HLV	10.1	$< 10^{-3}$	$< 10^{-3}$
GW190503	18:54:04	HLV	12.2	$< 10^{-3}$	0.07
GW190512	18:07:14	HLV	12.2	$< 10^{-3}$	$< 10^{-3}$
GW190513	20:54:28	HLV	11.9	$< 10^{-3}$	$< 10^{-3}$
GW190514	06:54:16	HL	8.3	N/A	$< 10^{-3}$
GW190517	05:51:01	HLV	10.6	$< 10^{-3}$	$< 10^{-3}$
GW190519	15:35:44	HLV	13.0	0.634	0.673
GW190521	03:02:29	HLV	14.4	0.866	0.868
GW190521b	07:43:59	HL	24.7	N/A	0.956
GW190527	09:20:55	HL	8.9	N/A	$< 10^{-3}$
GW190602	17:59:27	HLV	12.1	0.010	0.016
GW190620	03:04:21	LV	10.9	N/A	N/A
GW190630	18:52:05	LV	15.6	N/A	N/A
GW190701	20:33:06	HLV	10.2	$< 10^{-3}$	$< 10^{-3}$
GW190706	22:26:41	HLV	12.7	0.008	0.009
GW190707	09:33:26	HL	13.0	N/A	$< 10^{-3}$
GW190708	23:24:57	LV	13.1	N/A	N/A
GW190719	21:55:14	HL	8.0	N/A	$< 10^{-3}$
GW190720	00:08:36	HLV	11.7	$< 10^{-3}$	$< 10^{-3}$
GW190725	17:47:28	HLV	9.8	$< 10^{-3}$	$< 10^{-3}$
GW190727	06:03:33	HLV	12.3	$< 10^{-3}$	$< 10^{-3}$
GW190728	06:45:10	HLV	13.6	$< 10^{-3}$	$< 10^{-3}$
GW190731	14:09:36	HL	8.5	N/A	$< 10^{-3}$
GW190803	02:27:01	HLV	9.0	$< 10^{-3}$	$< 10^{-3}$
GW190805	21:11:37	HL	8.3	N/A	$< 10^{-3}$
GW190814	21:10:39	LV	22.2	N/A	N/A
GW190828	06:34:05	HLV	16.6	0.282	0.359
GW190828b	06:55:09	HLV	11.1	$< 10^{-3}$	$< 10^{-3}$

Event Name	UTC time	Detectors	SNR	MLy HLV	MLy HL
GW190909	11:41:49	HL	8.5	N/A	$< 10^{-3}$
GW190910	11:28:07	LV	13.4	N/A	N/A
GW190915	23:57:02	HLV	13.1	$< 10^{-3}$	$< 10^{-3}$
GW190916	20:06:58	HLV	8.2	$< 10^{-3}$	$< 10^{-3}$
GW190917	11:46:30	HLV	9.5	$< 10^{-3}$	$< 10^{-3}$
GW190924	02:18:46	HLV	13.2	$< 10^{-3}$	$< 10^{-3}$
GW190925	23:28:45	HV	9.9	N/A	$< 10^{-3}$
GW190926	05:03:36	HLV	9.0	$< 10^{-3}$	$< 10^{-3}$
GW190929	01:21:49	HLV	9.9	$< 10^{-3}$	$< 10^{-3}$
GW190930	13:3541	HL	10.0	N/A	$< 10^{-3}$

Table 6.10: The extended catalogue of transient events [8] for O3a and network SNR from the pipeline with the highest significance, as reported from MBTA, GstLAL or PyCBC or PyCBC-BBH [8] UTC time refers to the merger time. Detectors indicate which detectors were operational at the time of the event. When a signal is present only in HL, the HLV model is non-applicable (N/A). The same is happening in the case when H or L detectors are not observing. In such case non of our models are applicable. In this work we analyse only HL and HLV times.

6.4.3 O3b HLV analysis

The second part O3b of the third observing run was from the 1st of November 2019 to 27st of March 2020. Each detector was online for 115.7, 115.5 and 113.2 days in total, for H1, L1 and V1 respectively. Out of those times only 75 days were of triple coincidence between H1, L1 and V1. The step of the analysis is to calculate the 10 year equivalent false alarm rates of O3b.

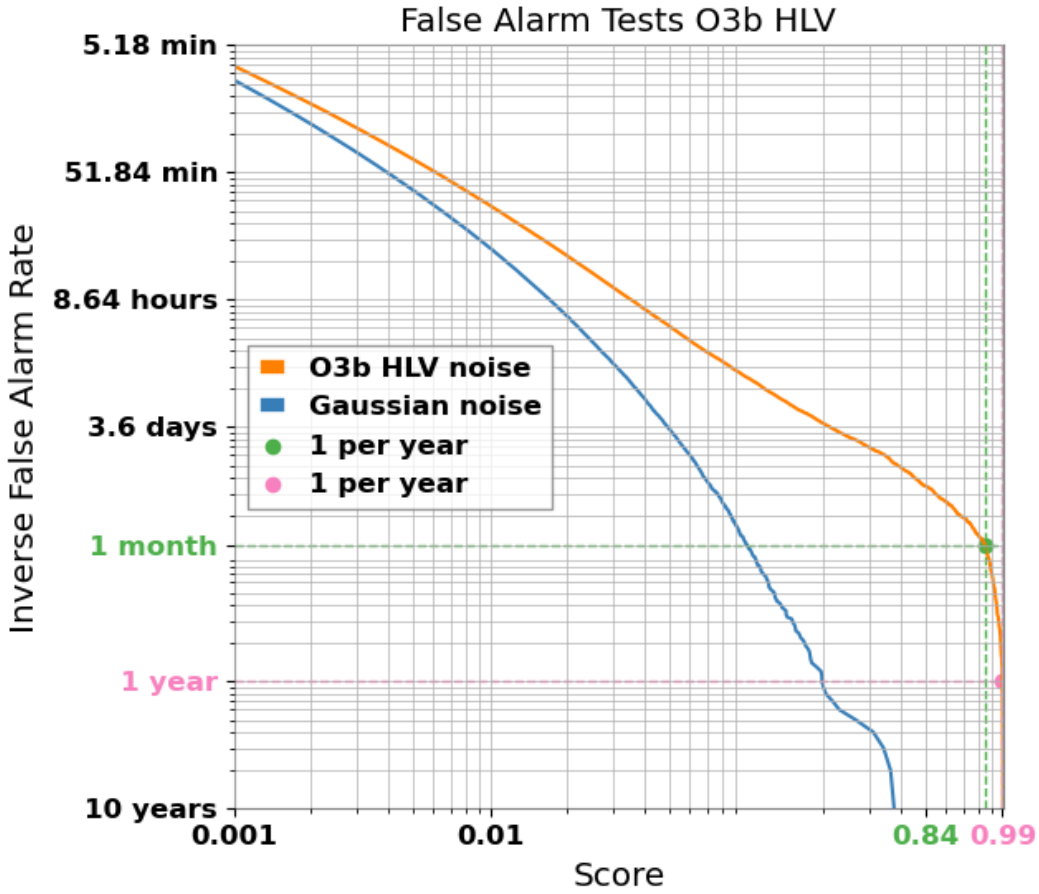

Figure 6.17: The false alarm rates of the HLV network in O3b in comparison with the false alarm rates of Gaussian noise. The score thresholds of 0.84 and 0.99 false alarm rates of 1/month and 1/year are also shown respectively (the 1/month score is shown slightly to the left to avoid overlapping).

In Figure 6.17 we see the false alarm rates for O3b for the HLV network.It is obvious that the false alarm rate is at high scores compared to O2 and O1 for the HL case too. Based on this our threshold score will be 0.9943 and we can use that to define triggers in our search. A threshold that high, hints that our training has a fault and we need to investigate a better method. We will discuss the fault in more detail a the end of the section.

In Figure 6.18 we see the performance on the various waveform morphologies for the three detector case of O3b. It is clear that the high threshold value increases a

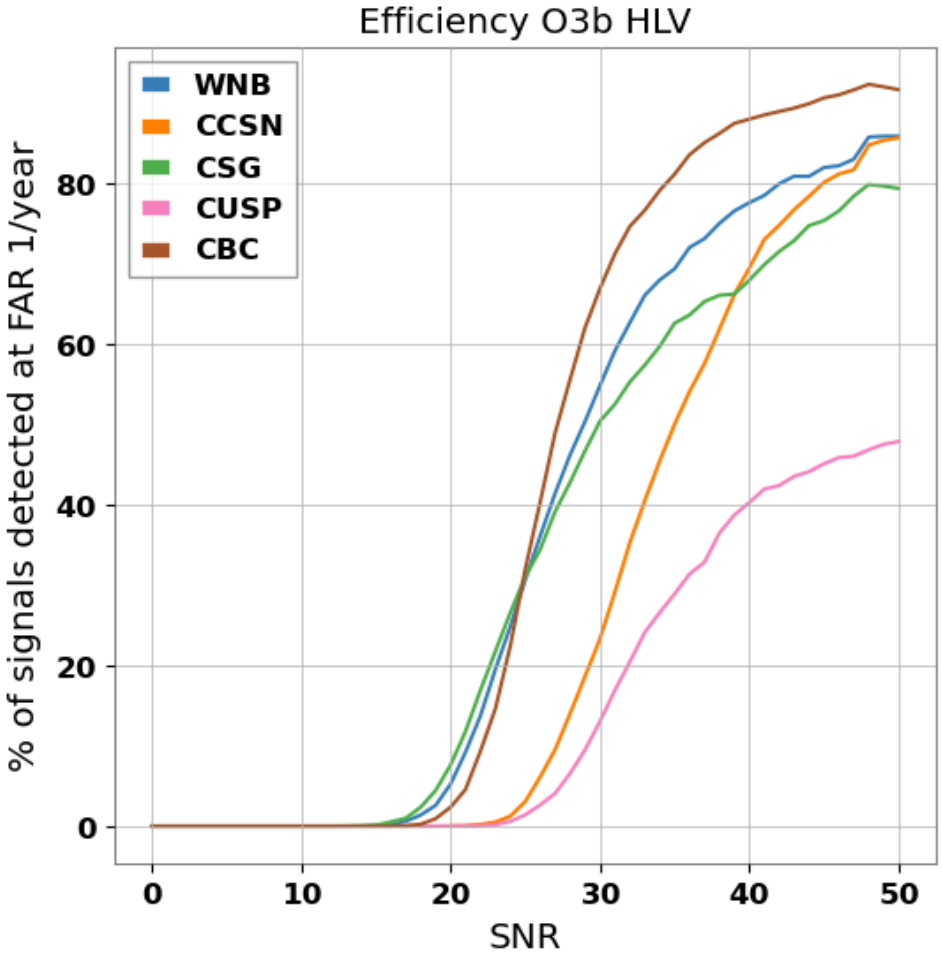

Figure 6.18: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O3b HLV observing run.

lot the 50% efficiency detection threshold for all waveforms. We also compare the performance of the model to the general short duration search for O3b [4].

In Figure 6.19 we present the performance of all generic burst types used for the comparison and in Table 6.12 the direct comparison of the 50% thresholds for each burst class.

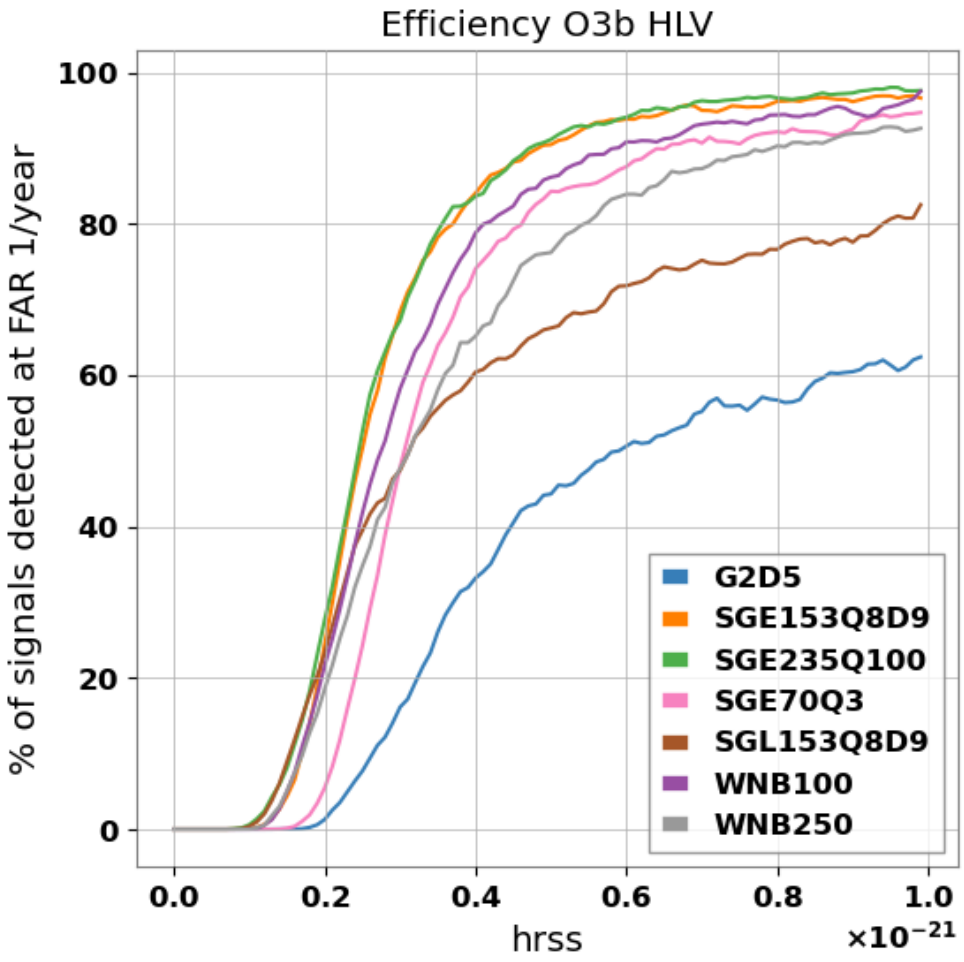

Figure 6.19: The HLV detection efficiency of the MLy pipeline as a function of the h_{rss} for generic sub-second burst signals and for a false alarm rate of 1 per year, during the O3b observing run.

Performance comparison on the O3b HLV data set		
Morphology	cWB [4] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	1.5	6.0
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	0.9	3.1
f_0=153 Hz, Q=8.9	0.6	2.6
f_0=235 Hz, Q=100	0.6	2.5
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	0.8	2.8
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	0.9	3.2

Table 6.11: Same comparison like table 5.1, but this time for O3b three detector search. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team.

6.4.4 O3b HL analysis

During O3b, for the two detector network, we had 93.4 days of coincidence for H1 and L1. First step of the analysis is to calculate the 10 year equivalent false alarm rates for the HL network in O3b.

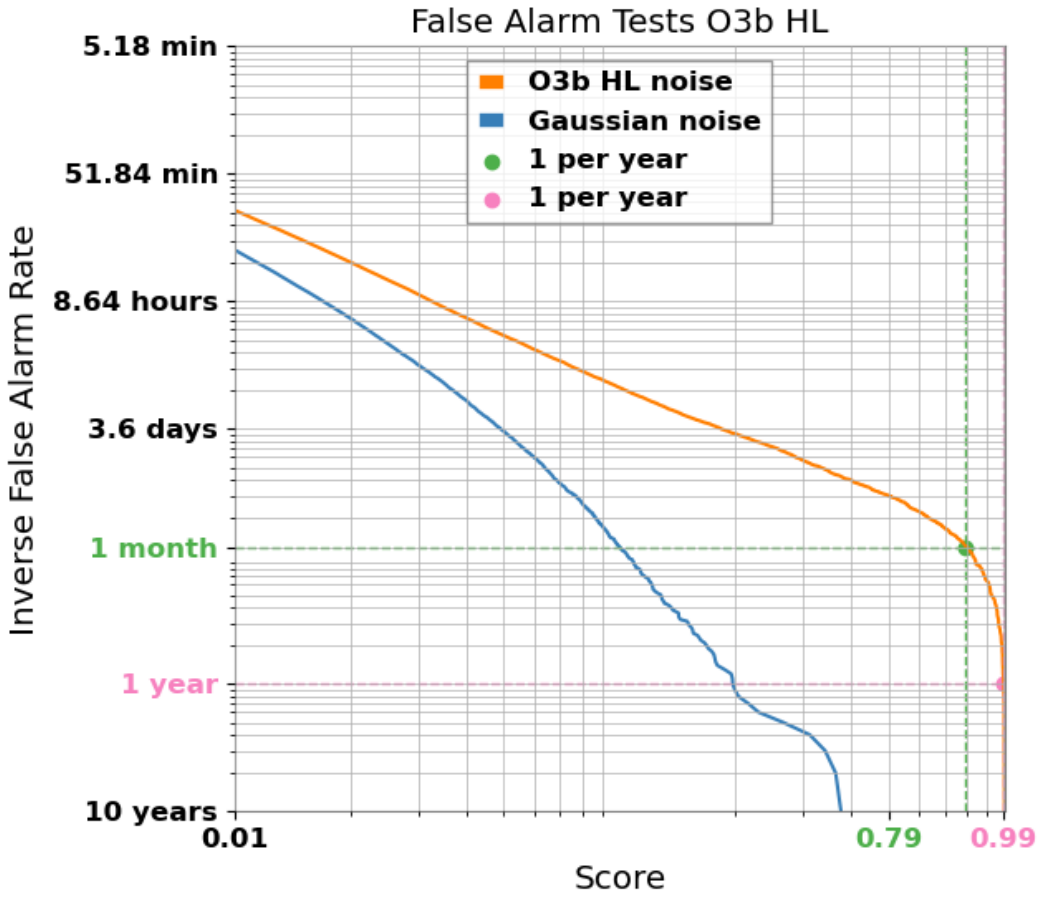

Figure 6.20: Same comparison like table 6.5, but this time for O3a three detector search. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team.

In Figure 6.20 we see the false alarm rates for O3b for the HL network.It is obvious that the false alarm rate is at high scores compared to O2 and O1 for the HL case too. Based on this our threshold score will be 0.9953 and we can use that to define triggers in our search.

In Figure 6.21 we see the performance on the various waveform morphologies for the HL detector analysis of O3b. It is clear that the high threshold value increases a lot the 50% detection threshold for all waveforms in this case too.

We also compare the performance of the model to the generalised short duration search for O3b [4]. In Figure 6.22 we present the performance of all generic burst types used for the comparison and in Table 6.12 the direct comparison of the 50% thresholds for each burst class. It is important to note that the G2D5 and

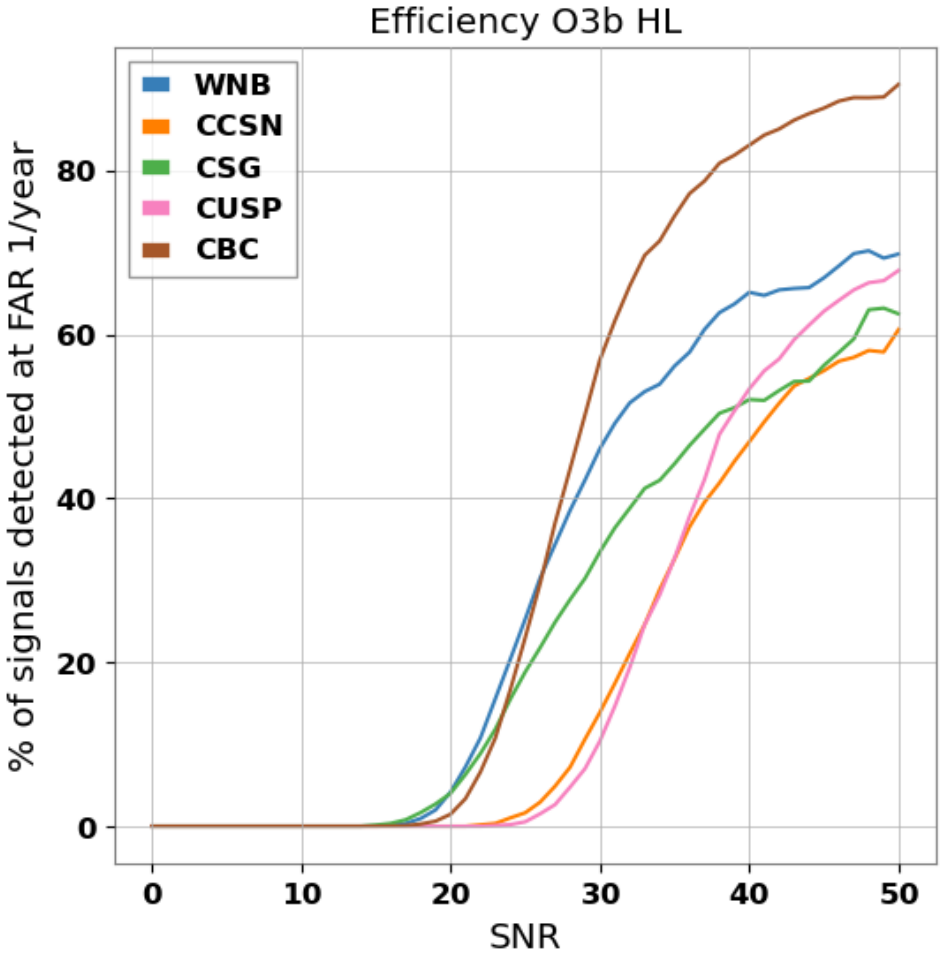

Figure 6.21: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O3b HL observing run.

SGL152Q8D9 injection sets were linearly polarised, while our training set consisted exclusively of unpolarised signals (WNBs with equal amplitudes in the two polarisations). We believe this to be the reason for MLy's poor limiting performance at high amplitudes for these injection sets.

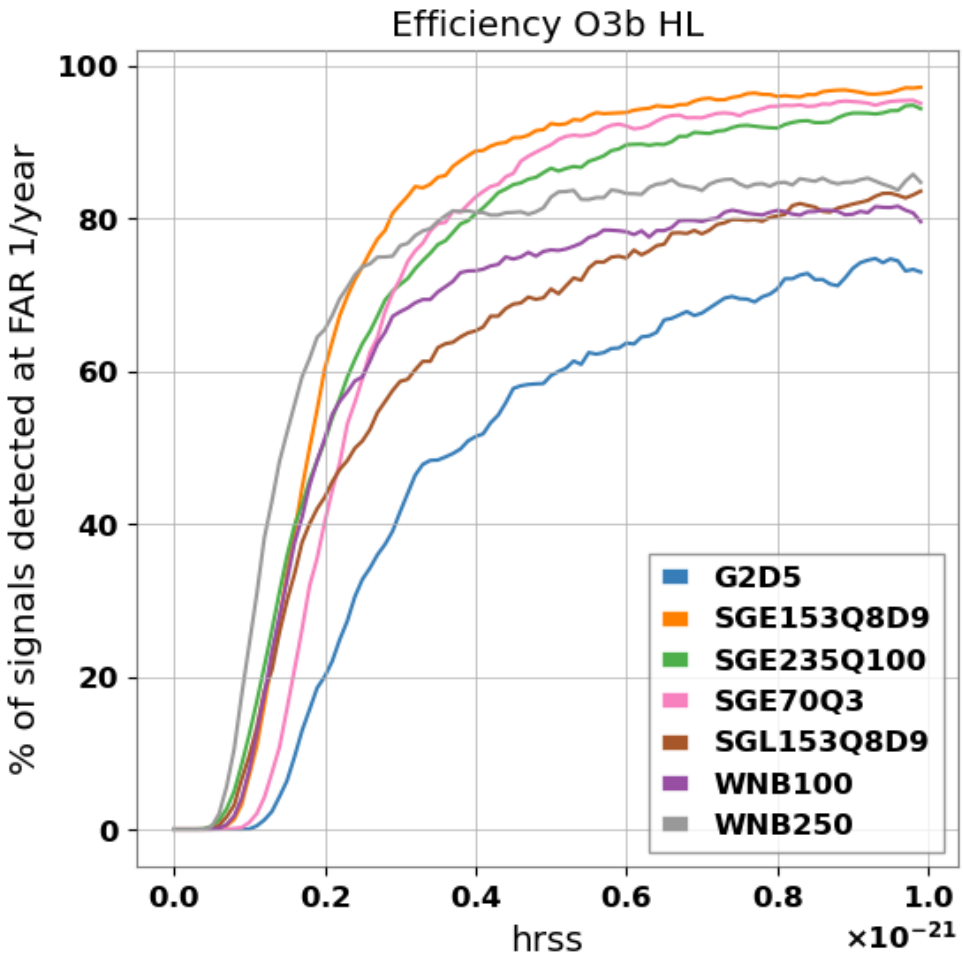

Figure 6.22: The detection efficiency of the MLy pipeline as a function of the network SNR for signals from various waveform morphologies described in subsection 4.2.1 and for a false alarm rate of 1 per year, during the O3b HL observing run.

Performance comparison of the O3b HL data set		
Morphology	cWB [4] ($10^{-22}\mathrm{Hz}^{-1/2}$)	MLy ($10^{-22}\mathrm{Hz}^{-1/2}$)
Gaussian pulses		
t=2.5 ms	1.5	3.8
Sine-Gaussian wavelets		
f_0=70 Hz, Q=3	0.9	2.3
f_0=153 Hz, Q=8.9	0.6	1.9
f_0=235 Hz, Q=100	0.6	2.0
White-Noise Bursts		
f_{low}=100 Hz, Δf=100 Hz, t=0.1 s	0.8	2.1
f_{low}=250 Hz, Δf=100 Hz, t=0.1 s	0.9	1.5

Table 6.12: Same comparison like table 6.5, but this time for O3b two detector search. We were provided with the efficiencies for FAR values of 1 per year by Shubhanshu Tiwari, a member of cWB team.

6.5 Discussion

We conducted an off-line search on the O1 O2 and O3 observing run data-sets using two CNN models trained on simulated Gaussian noise that follow the PSD curves of the detectors. We used the same models for all observing runs to actually test the generality of the model among runs. It is obvious from the results, especially when we compare the performance of O2 and O3, that the model is not performing the same on each run. Apparently the FAR for all thresholds (1/month, 1/year) increase with each run. As a result the efficiency of detecting various waveforms is decreasing overall. Here we will discuss the possible reasons behind those issues and investigations to fix them.

The first possible reason was the fact that we chose the best model based on O2-HL and O2-HLV performance. In Chapter 5, we discussed how we trained 7 different Model1 instances and 7 Model2 instances. In total those are 49 independent models when we combine their scores. Based on their performance on real noise from three detectors for August 2017, we chose the best model. We hoped that given that the model is not trained on real noise the dependency on the observing run will be small. Although the FAR of O1 is still smaller than O2, which suggests that this might not be the case. We repeated the procedure of model selection based on the performance of FAR for all the runs (O1, O2 ,O3a O3b) individually. For each run we have a best performing model. Although the differences were minimal and in some cases there was a common best model between runs (O2 and O1). This indicated to us that the issue has to be in the background noise.

Specifically in O3a and O3b the amount of loud false alarms makes the search impractical and raises concern on how would this model generalise in an even more advanced observing run like O4. Therefore we had to investigate in detail what type of triggers make MLy pipeline give such high scores. It was not a surprise to find some coincident glitches among those high scores. Most of those glitches had very similar morphologies and SNR values above 50. Among those triggers though a big proportion had no signal within the process window of one second. Those specific triggers were not making any sense, so we investigated the background that was used to create the PSD used by the whitening of those triggers. In our surprise those "empty" triggers had loud glitches included on their PSD calculation.

After some further testing we realised there was a bug in the outdated (at the time) version of GWpy we were using. That bug made the default version of PSD calculation to be labeled as the "median" method (using the median of the FFTs of overlapping segments), but instead it was the Weltch methdod (using the mean of the FFTs of overlapping segments). So practically we were using the mean method while we thought we were using the median. That had two consequences:

- If two loud glitches appear at any point in the background used for the calculation of a PSD in different detectors, their overlapping frequencies will create

background noise that looks to be coherent. The louder the glitches the more intense the coherency between the "noise" backgrounds. This creates conditions under which our pipeline will react with a high score, creating those triggers.

- During the creation of the training data, if a signal has a high SNR, due to the fact that we use the mean method for the PSD, it will have a considerable contribution to the PSD and hence it will reduce the intensity of the signal itself. This will prevent the creation of accurately loud signals of SNR 40 and above. As a result we cannot claim that we trained our model with really loud glitches, and that might be a reason it recognises really loud coincident glitches as signals. Additionally this might make all the signals we are generating appear of lower than specified SNR. This won't have an important effect in lower SNR values as it is proportionate the intensity of the signal.

We attempted to run all our FAR and efficiency calculations using the median method when we process the data, only to find that the FAR was significantly increased. Which makes sense from the perspective that we test with data that are processed differently from the training data. Then we attempted to generate new training data using the median method and retrain the models with the same parameters from scratch. Unfortunately the FAR was increased dramatically again indicating that we need to optimise the models based on the new way of processing the data. This is a quite interesting result. It demonstrates how sensitive the parameters of the models are to the data processing we use and that we need to be absolutely consistent with the training data, and for what are we optimising them for. Optimising the models for the "correct" type of data format is a matter of future investigation.

Although after conducting an investigation on the glitches that created those triggers we noticed that all of them are identified by GravitySpy [35] to be of different specific classes. Assuming that we have a glitch classification algorithm parallel to MLy pipeline, those glitches would have been identified and not produce triggers. This is of course a very optimistic statement, but it allows us to demonstrate how the FAR would look like in such case.

In figure 6.23 we present how the "loud tail" of FAR in O3a would look like in the case of a loud glitch classification running in parallel and ignoring triggers that involve coincident glitches within the 16 seconds used for the creation of the noise instance. As it is shown the cleaned FAR tail is smooth, following the Gaussian noise FAR curve closely.

A fair argument would be that we claim that our models are good on ignoring glitches, so hasn't the pipeline failed to do its job? It would be probably impossible to have a single model that ignores any glitch, especially when we do a generic search as is this. Our model is ignoring single glitches successfully, although coincident

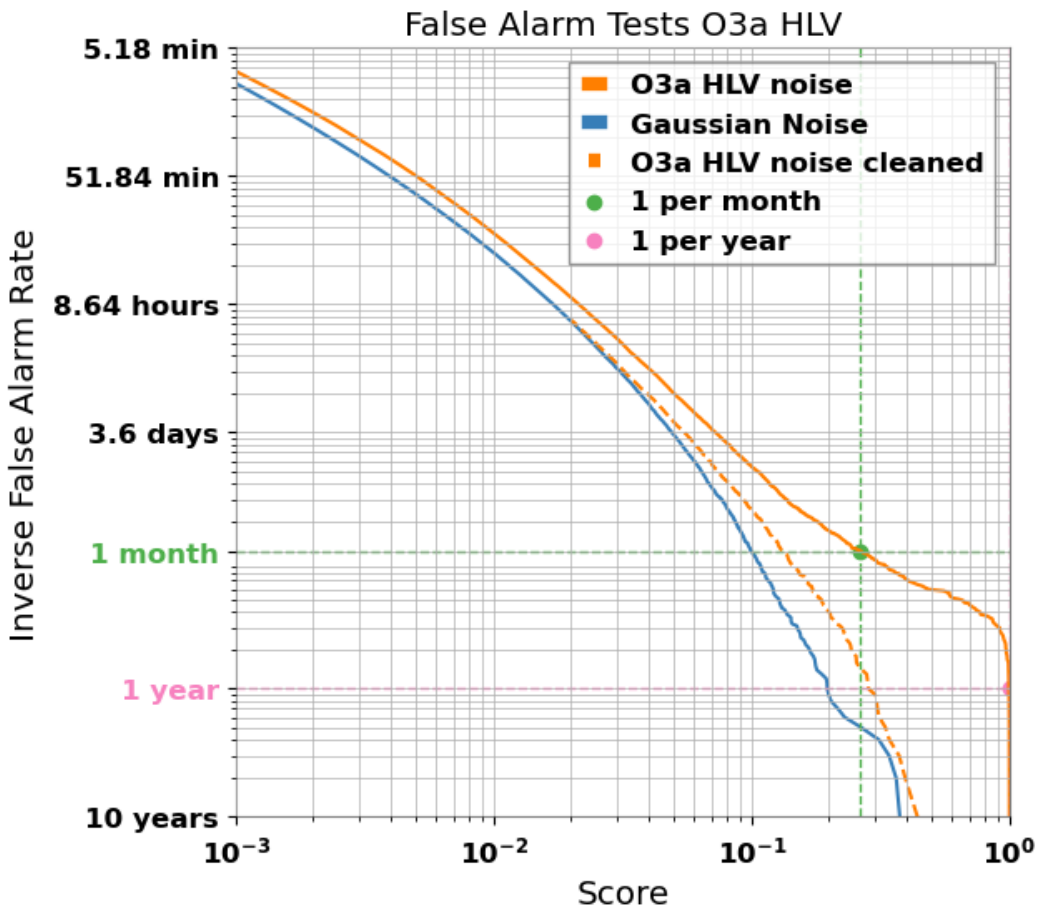

Figure 6.23: The false alarm rates of the HLV network in O3a in comparison with the the same false alarm rates when we ignore instances that involve coincident glitches.

glitches will always be a challenge that we will try to get better at. At the current state, this model has a limit at once per year FAR rates, which is enough for a low latency online search similar to O2 (for searches similar to O3 we have work to do).

Other than the issues to be addressed there are some interesting observations to be made from our results. For example the same seasonal effect that makes the winter background more noisy than the summer one, is present in our search too. O3a and O3b have the same average noise level, although O3b is much more noisy than O3a due to environmental disturbances during winter from wind, waves and storms. That affects the background. O3b is much more noisy than O3a as is shown in FAR tests for both HL and HLV searches.

In all false alarm test plots, we always included the Gaussian noise performance along with the real noise case. We did that to show the expected limitations of our model. We can confirm, at least for the O2 case, that by simulating the intricacies of the real noise using Gaussian noise and artificial glitches, we can achieve a background distribution of the search statistic that is very similar to that for Gaussian

noise, down to FARs or approximately 1/year.

Overall MLy pipeline seems to be sensitive equally to most types of signals. In the efficiency plots we observed specific signal types to consistently under-perform. Those signal types are all linearly polarised, in comparison to other signals that are either unpolarised or elliptically polarised.

We were able to detect (FAR $<$ 1/year) only one event **GW150914** in O1. We also detected five sub-threshold events (1/month $<$ FAR $<$ 1/year) in O3a. Those detections confirm that our estimated detector efficiencies, at least on BBH signals are consistent with our results. Our performance comparison to the cWB search on generic burst signals shows a lot of potential for this pipeline for possible off-line searches if the FAR is reduced by two orders of magnitude. Given the FAR thresholds and the efficiency comparisons we believe MLy pipeline to be a promising pipeline for low-latency analysis, as long as it continues to evolve with the observing run demands.

Chapter 7

Conclusion

In the research on which we base this thesis, we investigated many different model configurations to make a machine learning algorithm detect gravitational waves signals. We started by describing the two subjects we try to connect, first we had a simple introduction on gravitational wave astronomy based on Bernard Schutz book on general relativity [10] and then an introduction to the basic concepts in machine learning and how it works and giving a description of all the techniques used. Later we described the fundamental tools we created during this research, to generate data-sets and tools to validate the models we train but most importantly to make the process reproducible and easy to expand in other problems. Finally we used those tools to find out the best configuration for a model so that it can detect generic sub-second gravitational wave transients with low enough false alarm rate to be useful for an online search. Then we presented the results of that model on offline searches in all three observing runs.

To get the results we present here we went through a lot of trials of performance but also a need to create a protocol that lead us to creating a pipeline. Earlier year research we focused on learning the basics of machine learning by trying replications of investigations on CBC signals. When we were confident enough we stared building a consistent pipeline with the goal to make similar investigations much easier without spending 90% of the time on data generation and preparation. Now it is much easier for us and anyone deciding to work on similar problems to work on improving the models.

The state of our current best model is good enough to show that there is potential on using machine learning for short duration burst searches. Even more it removes the obstacle of using real noise, by trying to simulate it for training. We have also shown that having one model trained cannot justify which approach of training is the best, and we need to treat models as we do test samples. This provides us with the tools to rely less on the black-box mentality which is usually how machine learning algorithms are used.

This research is not finished and the models are not optimised in all ways possible.

In Chapter 6 we saw that our performance is reduced with each observing runs due to the frequency of the occurrence of glitches in coincidence. In addition to that we can see from the different scores on events that model 2 seems to always score less than model 1, acting like a moderator in the result. This was an example of the optimisation still need. We need to find a balance between the two so that the contribute equally to the final score, and possibly this will be done by finding the optimal lowest SNR limit for training data in both models, which is something we haven't investigated as much as we have with input ratios. Another aspect we need to address is how general the model is. We saw that CUSP signals and SGL153Q8d9 signals under-perform. The common thing they have between them is that they are both lineally polarised. So we need to investigate the impact of polarisation in the training, which is something we found out recently.

In the near future we need to integrate the model creation with a complete genetic algorithm that would make the parameter selection a non-manual process. Further more we need to focus our optimisation on the new data whitening method (using the median of the FFTs) to hope to overcome the glitch issue. We don't expect to overcome the glitch issue completely so if MLy-pipeline wants to continue being successful in the future runs, it needs to have a glitch classification running in parallel, to help remove the glitches that appear in the detectors for off-line and low-latency searches. This is only one of the few important additions needed for MLy pipeline. The main goal for the current algorithm is to decrease the false alarm rate and find clever ways to have reduced reactions on non signals. Our metric of intensity (score) is ranging between [0,1]. As we discussed before this restricts our results and makes it impossible to differentiate between loud and week signals. Having a quantified measure of intensity will be a very important step to help differentiate different intensity of signals.

To eventually have a complete search pipeline, we also need a basic parameter estimation. We need independent models that will return a signal duration, frequency range and other useful generic parameters. Most importantly we need to develop a sky-localisation follow-up for the triggers of this algorithm. All those might be projects distributed to other future students or collaborators.

Focusing again to the classification problem we discussed in this thesis we need to extend those models to bigger range of parameters. A potential first extension is the duration of the signals. We could also use models that identify low intensity short duration signals and use them to provide continues triggers. Potentially this could lead to long duration burst search. The other extension of parameters could be the frequency range. Such an extension would require a lot more examples of injections and much more space for the training datasets. Both those extension of parameters are not trivial and they will need a functional optimisation algorithm.

Finally until all the above future plans come to fruition, we need to change MLy code itself and the way it calculates the false alarm rates and efficiencies, to

be compatible with low-latency. Due to technical issues we cannot calculate a FAR more that 1 per 10 years in a reasonable time, and this is really restricting our power to compare with other pipelines. This technical issue will be solved by creating a file system with processed data ready to be used for FAR calculation.

Bibliography

[1] Giles Hammond, Stefan Hild, and Matthew Pitkin. Advanced technologies for future ground-based, laser-interferometric gravitational wave detectors. *Journal of Modern Optics*, 61(sup1):S10–S45, June 2014.

[2] D. V. Martynov, E. D. Hall, and et al. Sensitivity of the advanced ligo detectors at the beginning of gravitational wave astronomy. *Physical Review D*, 93(11), Jun 2016.

[3] LIGO Scientific Collaboration and Virgo Collaboration. Ligo virgo strain data from observing run o2 (nov 2016 - aug 2017) https://www.gw-openscience.org/o2/, 2019.

[4] The LIGO Scientific Collaboration the Virgo Collaboration and the KAGRA Collaboration. All-sky search for short gravitational-wave bursts in the third advanced ligo and advanced virgo run. *Phys. Rev. D*, 104:122004, Dec 2021.

[5] All-sky search for short gravitational-wave bursts in the second advanced LIGO and advanced Virgo run. *Phys. Rev. D*, 100:024017, Jul 2019.

[6] B. P. Abbott, R. Abbott, and et al. Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs. *Physical Review X*, 9(3), Sep 2019.

[7] All-sky search for short gravitational-wave bursts in the first advanced LIGO run. *Phys. Rev. D*, 95:042003, Feb 2017.

[8] The LIGO Scientific Collaboration and the Virgo Collaboration. Gwtc-2.1: Deep extended catalog of compact binary coalescences observed by ligo and virgo during the first half of the third observing run, 2021.

[9] Hunter Gabbard, Michael Williams, Fergus Hayes, and Chris Messenger. Matching matched filtering with deep networks in gravitational-wave astronomy. *Physical Review Letters*, 120(14):141103, April 2018. arXiv: 1712.06041.

[10] Bernard Schutz. A first course in general relativity, 2009.

[11] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[12] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, October 1959.

[13] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968.

[14] Kunihiko Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

[15] Christian Szegedy, Wei Liu, and et al. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[16] NYU Corinna Cortes Yann LeCun, Courant Institute and et al. The mnist database of handwritten digits http://yann.lecun.com/exdb/mnist/.

[17] Y. NESTEROV. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. *Doklady AN USSR*, 269:543–547, 1983.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *1412.6980 arXiv*, 2017.

[20] Daniel George and E. A. Huerta. Deep Learning for Real-time Gravitational Wave Detection and Parameter Estimation with LIGO Data. *arXiv:1711.07966 [astro-ph, physics:gr-qc]*, November 2017. arXiv: 1711.07966.

[21] Christopher Bresten and Jae-Hun Jung. Detection of gravitational waves using topological data analysis and convolutional neural network: An improved approach. *arXiv:1910.08245 [astro-ph, physics:gr-qc, physics:physics]*, October 2019. arXiv: 1910.08245.

[22] He Wang, Zhoujian Cao, Xiaolin Liu, and et al. Gravitational wave signal recognition of O1 data by deep learning. *arXiv:1909.13442 [astro-ph, physics:gr-qc]*, September 2019. arXiv: 1909.13442.

[23] Alexander Schmitt, Kaiyu Fu, Siyu Fan, and Yuan Luo. Investigating deep neural networks for gravitational wave detection in advanced ligo data. In *Proceedings of the 2nd International Conference on Computer Science and Software Engineering*, CSSE 2019, page 73–78, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Timothy D. Gebhard, Niki Kilbertus, Ian Harry, and Bernhard Schölkopf. Convolutional neural networks: a magic bullet for gravitational-wave detection? *Physical Review D*, 100(6):063015, September 2019. arXiv: 1904.08693.

[25] Chayan Chatterjee, Linqing Wen, Foivos Diakogiannis, and Kevin Vinsen. Extraction of binary black hole gravitational wave signals from detector data using deep learning. *Phys. Rev. D*, 104:064046, Sep 2021.

[26] Dwyer S. Deighan, Scott E. Field, Collin D. Capano, and Gaurav Khanna. Genetic-algorithm-optimized neural networks for gravitational wave classification. *Neural Computing and Applications*, April 2021.

[27] Hua-Mei Luo, Wenbin Lin, Zu-Cheng Chen, and Qing-Guo Huang. Extraction of gravitational wave signals with optimized convolutional neural network. *Frontiers of Physics*, 15(1):14601, November 2019.

[28] XiLong Fan, Jin Li, Xin Li, YuanHong Zhong, and JunWei Cao. Applying deep neural networks to the detection and space parameter estimation of compact binary coalescence with a network of gravitational wave detectors. *Science China Physics, Mechanics & Astronomy*, 62(6):969512, February 2019.

[29] Chayan Chatterjee, Linqing Wen, Kevin Vinsen, Manoj Kovalam, and Amitava Datta. Using deep learning to localize gravitational wave sources. *Physical Review D*, 100(10), Nov 2019.

[30] Hongyu Shen, E. A. Huerta, Zhizhen Zhao, and et al. Deterministic and Bayesian Neural Networks for Low-latency Gravitational Wave Parameter Estimation of Binary Black Hole Mergers. *arXiv:1903.01998 [astro-ph, physics:gr-qc, stat]*, March 2019. arXiv: 1903.01998.

[31] Stephen R. Green, Christine Simpson, and Jonathan Gair. Gravitational-wave parameter estimation with autoregressive neural network flows. *Physical Review D*, 102(10), Nov 2020.

[32] Hunter Gabbard, Chris Messenger, Ik Siong Heng, Francesco Tonolini, and Roderick Murray-Smith. Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy. *Nature Physics*, 18(1):112–117, Dec 2021.

[33] Sebastian Khan and Rhys Green. Gravitational-wave surrogate models powered by artificial neural networks. *Physical Review D*, 103(6), Mar 2021.

[34] S. B. Coughlin, S. Bahaadini, N. Rohani, and et al. Classifying the unknown: discovering novel gravitational-wave detector glitches using similarity learning. *Physical Review D*, 99(8):082002, April 2019. arXiv: 1903.04058.

[35] M Zevin, Coughlin, and et al. Gravity spy: integrating advanced ligo detector characterization, machine learning, and citizen science. *Classical and Quantum Gravity*, 34(6):064003, Feb 2017.

[36] Massimiliano Razzano and Elena Cuoco. Image-based deep learning for classification of noise transients in gravitational wave detectors. *Classical and Quantum Gravity*, 35(9):095016, apr 2018.

[37] R Essick, L Blackburn, and E Katsavounidis. Optimizing vetoes for gravitational-wave transient searches. *Classical and Quantum Gravity*, 30(15):155010, Jun 2013.

[38] Joshua R Smith, Thomas Abbott, Eiichi Hirose, Nicolas Leroy, and et al. A hierarchical method for vetoing noise transients in gravitational-wave detectors. *Classical and Quantum Gravity*, 28(23):235005, Nov 2011.

[39] Robert E. Colgan, K. Rainer Corley, Yenson Lau, and et al. Efficient gravitational-wave glitch identification from environmental data through machine learning. *Phys. Rev. D*, 101:102003, May 2020.

[40] Rahul Biswas, Lindy Blackburn, Junwei Cao, and et al. Application of machine learning algorithms to the study of noise artifacts in gravitational-wave data. *Phys. Rev. D*, 88:062003, Sep 2013.

[41] Multi-messenger observations of a binary neutron star merger. *The Astrophysical Journal Letters*, 848:L12:59, 2017.

[42] Hongyu Shen, Daniel George, E. A. Huerta, and Zhizhen Zhao. Denoising Gravitational Waves using Deep Learning with Recurrent Denoising Autoencoders. *arXiv:1711.09919 [astro-ph, physics:gr-qc]*, November 2017. arXiv: 1711.09919.

[43] Wei Wei and E.A. Huerta. Gravitational wave denoising of binary black hole mergers with deep learning. *Physics Letters B*, 800:135081, 2020.

[44] Rich Ormiston, Tri Nguyen, Michael Coughlin, Rana X. Adhikari, and Erik Katsavounidis. Noise reduction in gravitational-wave data via deep learning. *Phys. Rev. Research*, 2:033066, Jul 2020.

[45] P. Astone, P. Cerdá-Durán, I. Di Palma, and et al. New method to observe gravitational waves emitted by core collapse supernovae. *Phys. Rev. D*, 98:122002, Dec 2018.

[46] Alberto Iess, Elena Cuoco, Filip Morawski, and Jade Powell. Core-collapse supernova gravitational-wave search and deep learning classification, 2020.

[47] Patrick J Sutton, Gareth Jones, Shourov Chatterji, and et al. X-Pipeline: an analysis package for autonomous gravitational-wave burst searches. *New Journal of Physics*, 12(5):053034, may 2010.

[48] Duncan Macleod, Alex L. Urban, Scott Coughlin, and et al. gwpy/gwpy: 2.0.1, 2020.

[49] Duncan Macleod. duncanmmacleod/dqsegdb2: 1.0.1, 2019.

[50] Characterization of transient noise in advanced LIGO relevant to gravitational wave signal GW150914. *Classical and Quantum Gravity*, 33(13):134001, Jun 2016.

[51] Gwdatafind https://gwdatafind.readthedocs.io/en/stable/index.html.

[52] GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and Virgo during the first and second observing runs. *Physical Review X*, 9(3), Sep 2019.

[53] Sascha Husa, Sebastian Khan, Mark Hannam, and et al. Frequency-domain gravitational waves from nonprecessing black-hole binaries. i. new numerical waveforms and anatomy of the signal. *Phys. Rev. D*, 93:044006, Feb 2016.

[54] Müller, E., Janka, H.-Th., and Wongwathanarat, A. Parametrized 3d models of neutrino-driven supernova explosions - neutrino emission asymmetries and gravitational-wave signals. *A&A*, 537:A63, 2012.

[55] M B Hindmarsh and T W B Kibble. Cosmic strings. *Reports on Progress in Physics*, 58(5):477–562, may 1995.

[56] Thibault Damour and Alexander Vilenkin. Gravitational wave bursts from cosmic strings. *Phys. Rev. Lett.*, 85:3761–3764, Oct 2000.

[57] Warren G. Anderson, Patrick R. Brady, Jolien D. E. Creighton, and Éanna É. Flanagan. Excess power statistic for detection of burst sources of gravitational radiation. *Physical Review D*, 63(4), Jan 2001.

[58] J Aasi et al. Advanced LIGO. *Classical and Quantum Gravity*, 32(7):074001, mar 2015.

[59] F Acernese et al. Advanced virgo: a second-generation interferometric gravitational wave detector. *Classical and Quantum Gravity*, 32(2):024001, dec 2014.

[60] GW190425: Observation of a compact binary coalescence with total mass $\sim 3.4\,\mathrm{M}_\odot$. *The Astrophysical Journal*, 892(1):L3, Mar 2020.

[61] GW190412: Observation of a binary-black-hole coalescence with asymmetric masses. *Physical Review D*, 102(4), Aug 2020.

[62] GW190814: Gravitational waves from the coalescence of a 23 solar mass black hole with a 2.6 solar mass compact object. *The Astrophysical Journal*, 896(2):L44, Jun 2020.

[63] GW190521: A binary black hole merger with a total mass of 150 $M_{\odot}$. *Phys. Rev. Lett.*, 125:101102, Sep 2020.

[64] https://gracedb.ligo.org/superevents/public/O3/.

[65] The LIGO Scientific Collaboration, the Virgo Collaboration, and the KAGRA Collaboration. Gwtc-3: Compact binary coalescences observed by ligo and virgo during the second part of the third observing run, 2021.

[66] Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, Feb 2016.

[67] Koutarou Kyutoku, Fujibayashi, and et al. On the Possibility of GW190425 Being a Black Hole-Neutron Star Binary Merger. *apj*, 890(1):L4, February 2020.

[68] GW170817: Observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett.*, 119:161101, Oct 2017.

[69] Daniel Kasen, Brian Metzger, Jennifer Barnes, Eliot Quataert, and Enrico Ramirez-Ruiz. Origin of the heavy elements in binary neutron-star mergers from a gravitational-wave event. *Nature*, 551(7678):80–84, October 2017.

[70] GW170817: Measurements of neutron star radii and equation of state. *Phys. Rev. Lett.*, 121:161101, Oct 2018.

[71] Estimating the contribution of dynamical ejecta in the kilonova associated with GW170817. *The Astrophysical Journal*, 850(2):L39, dec 2017.

[72] A gravitational-wave standard siren measurement of the Hubble constant. *Nature*, 551(7678):85–88, October 2017.

[73] Low-latency gravitational-wave alerts for multimessenger astronomy during the second advanced LIGO and Virgo observing run. *The Astrophysical Journal*, 875(2):161, apr 2019.

[74] Daniel George and E. A. Huerta. Deep Neural Networks to Enable Real-time Multimessenger Astrophysics. *Physical Review D*, 97(4):044039, November 2017. arXiv: 1701.00008.

[75] B. P. Abbott, R. Abbott, Abbott, and et al. Gw170817: Measurements of neutron star radii and equation of state. *Physical Review Letters*, 121(16), Oct 2018.

[76] Jeremiah W. Murphy, Christian D. Ott, and Adam Burrows. A model for gravitational wave emission from neutrino-driven core-collapse supernovae. *The Astrophysical Journal*, 707(2):1173–1190, dec 2009.

[77] Anthony L. Piro and Eric Pfahl. Fragmentation of collapsar disks and the production of gravitational waves. *The Astrophysical Journal*, 658(2):1173–1176, apr 2007.

[78] Sergey Klimenko, Vedovato, and et al., 2020.

[79] Ryan Lynch, Salvatore Vitale, Reed Essick, Erik Katsavounidis, and Florent Robinet. Information-theoretic approach to the gravitational-wave burst detection problem. *prd*, 95(10):104046, May 2017.

[80] Neil J Cornish and Tyson B Littenberg. Bayeswave: Bayesian inference for gravitational wave bursts and instrument glitches. *Classical and Quantum Gravity*, 32(13):135012, Jun 2015.

[81] Hassan Ismail Fawaz, Germain Forestier, and Jonathanand et al. Weber. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Mar 2019.

[82] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[83] Prospects for observing and localizing gravitational-wave transients with advanced LIGO, advanced Virgo and KAGRA. *Living Rev Relativ*, 21(3), 2013.

[84] LIGO Scientific Collaboration and Virgo Collaboration. LIGO-Virgo strain data from observing run O2 (Nov 2016 - Aug 2017), 2019.

[85] Vasileios Skliris. MLy, gravitational waves and machine learning library. https://git.ligo.org/vasileios.skliris/mly.

[86] Samantha A Usman, Alexander H Nitz, Harry, and et al. The pycbc search for gravitational waves from compact binary coalescence. *Classical and Quantum Gravity*, 33(21):215004, Oct 2016.

[87] Martín Abadi, Ashish Agarwal, Paul Barham, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[88] https://emfollow.docs.ligo.org/userguide/analysis/index.html.

[89] Sebastian Khan, Sascha Husa, and et al. Frequency-domain gravitational waves from nonprecessing black-hole binaries. ii. a phenomenological model for the advanced detector era. *Phys. Rev. D*, 93:044007, Feb 2016.

[90] B. P. Abbott, R. Abbott, and et al. All-sky search for short gravitational-wave bursts in the second advanced ligo and advanced virgo run. *Physical Review D*, 100(2), Jul 2019.

[91] Leo P. Singer and Larry R. Price. Rapid bayesian position reconstruction for gravitational-wave transients. *Physical Review D*, 93(2), Jan 2016.

[92] B. P. Abbott, Abbott, and et al. Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, 116(6), Feb 2016.

[93] B. P. Abbott, Abbott, and et al. Gw151226: Observation of gravitational waves from a 22-solar-mass binary black hole coalescence. *Physical Review Letters*, 116(24), Jun 2016.

[94] B. P. Abbott, Abbott, and et al. Binary black hole mergers in the first advanced ligo observing run. *Physical Review X*, 6(4), Oct 2016.

[95] Tito Dal Canton, Alexander H. Nitz, and et al. Implementing a search for aligned-spin neutron star-black hole systems with advanced ground based gravitational wave detectors. *Phys. Rev. D*, 90:082004, Oct 2014.

[96] Kipp Cannon, Romain Cariou, Adrian Chapman, and et al. TOWARD EARLY-WARNING DETECTION OF GRAVITATIONAL WAVES FROM COMPACT BINARY COALESCENCE. *The Astrophysical Journal*, 748(2):136, mar 2012.

[97] Stephen Privitera, Mohapatra, and et al. Improving the sensitivity of a search for coalescing binary black holes with nonprecessing spins in gravitational wave data. *Phys. Rev. D*, 89:024003, Jan 2014.

[98] Cody Messick, Blackburn, and et al. Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data. *Physical Review D*, 95(4), Feb 2017.

[99] B. P. Abbott, Abbott, and et al. Upper limits on the stochastic gravitational-wave background from advanced ligo's first observing run. *Physical Review Letters*, 118(12), Mar 2017.

[100] B. P. Abbott, R. Abbott, and et al. All-sky search for periodic gravitational waves in the o1 ligo data. *Physical Review D*, 96(6), Sep 2017.

[101] B. P. Abbott, Abbott, and et al. Gw170104: Observation of a 50-solar-mass binary black hole coalescence at redshift 0.2. *Physical Review Letters*, 118(22), Jun 2017.

[102] B. P. Abbott, Abbott, and et al. Gw170608: Observation of a 19 solar-mass binary black hole coalescence. *The Astrophysical Journal*, 851(2):L35, Dec 2017.

[103] Gravitational Wave Open Science Center. Strain data release for gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs, 2018.

[104] B. P. Abbott, R. Abbott, T. D. Abbott, and et al. Gw170814: A three-detector observation of gravitational waves from a binary black hole coalescence. *Physical Review Letters*, 119(14), Oct 2017.

[105] B. P. Abbott, Abbott, and et al. Gw170817: Observation of gravitational waves from a binary neutron star inspiral. *Physical Review Letters*, 119(16), Oct 2017.

[106] Surabhi Sachdev, Sarah Caudill, and et al. The gstlal search analysis methods for compact binary mergers in advanced ligo's second and advanced virgo's first observing runs, 2019.

[107] B. P. Abbott, R. Abbott, and et al. Directional limits on persistent gravitational waves using data from advanced ligo's first two observing runs. *Physical Review D*, 100(6), Sep 2019.

[108] B. P. Abbott, R. Abbott, and et al. All-sky search for long-duration gravitational-wave transients in the second advanced ligo observing run. *Physical Review D*, 99(10), May 2019.

[109] B. P. Abbott, R. Abbott, and et al. Multi-messenger observations of a binary neutron star merger. *The Astrophysical Journal*, 848(2):L12, Oct 2017.

[110] B. P. Abbott, Abbott, and et al. Gravitational waves and gamma-rays from a binary neutron star merger: Gw170817 and grb 170817a. *The Astrophysical Journal*, 848(2):L13, Oct 2017.

[111] A gravitational-wave standard siren measurement of the hubble constant. *Nature*, 551(7678):85–88, Oct 2017.

[112] B. P. Abbott, R. Abbott, Abbott, and et al. Tests of general relativity with gw170817. *Physical Review Letters*, 123(1), Jul 2019.

[113] Gravityspy database.

[114] B. P. Abbott, R. Abbott, Abbott, and et al. Gw190425: Observation of a compact binary coalescence with total mass 3.4 $m_{\odot}$. *The Astrophysical Journal*, 892(1):L3, Mar 2020.

[115] R. Abbott, T. D. Abbott, Abraham, and et al. Gw190412: Observation of a binary-black-hole coalescence with asymmetric masses. *Physical Review D*, 102(4), Aug 2020.

[116] R. Abbott, T. D. Abbott, Abraham, and et al. Gw190814: Gravitational waves from the coalescence of a 23 solar mass black hole with a 2.6 solar mass compact object. *The Astrophysical Journal*, 896(2):L44, Jun 2020.

[117] R. Abbott, T. D. Abbott, Abraham, and et al. Gw190521: A binary black hole merger with a total mass of $150m_{\odot}$. *Physical Review Letters*, 125(10), Sep 2020.

[118] Bruce Allen, Warren G. Anderson, Patrick R. Brady, Duncan A. Brown, and Jolien D. E. Creighton. Findchirp: An algorithm for detection of gravitational waves from inspiraling compact binaries. *Physical Review D*, 85(12), Jun 2012.

[119] Bruce Allen. χ^2time-frequency discriminator for gravitational wave detection. *Physical Review D*, 71(6), Mar 2005.

[120] Tito Dal Canton, Alexander H. Nitz, and et al. Implementing a search for aligned-spin neutron star-black hole systems with advanced ground based gravitational wave detectors. *Physical Review D*, 90(8), Oct 2014.

[121] Alexander H. Nitz, Thomas Dent, Tito Dal Canton, Stephen Fairhurst, and Duncan A. Brown. Detecting binary compact-object mergers with gravitational waves: Understanding and improving the sensitivity of the pycbc search. *The Astrophysical Journal*, 849(2):118, Nov 2017.

[122] Chad Hanna, Sarah Caudill, and et al. Fast evaluation of multidetector consistency for real-time gravitational wave searches. *Physical Review D*, 101(2), Jan 2020.

[123] F Aubin, F Brighenti, Chierici, and et al. The mbta pipeline for detecting compact binary coalescences in the third ligo–virgo observing run. *Classical and Quantum Gravity*, 38(9):095004, Apr 2021.

[124] Qi Chu. *Low-latency detection and localization of gravitational waves from compact binary coalescences*. PhD thesis, The University of Western Australia, 2017.

[125] R. Abbott, T. D. Abbott, and et al. Constraints on cosmic strings using data from the third advanced ligo–virgo observing run. *Physical Review Letters*, 126(24), Jun 2021.

[126] The LIGO Scientific Collaboration, the Virgo Collaboration, and the KAGRA Collaboration. All-sky search for long-duration gravitational-wave bursts in the third advanced ligo and advanced virgo run, 2021.

[127] The LIGO Scientific Collaboration, the Virgo Collaboration, and the KAGRA Collaboration. Search for subsolar-mass binaries in the first half of advanced ligo and virgo's third observing run, 2021.