UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Deep learning applied to automatic detection of gravitational waves

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Chorro Juan, Daniel

Tutor: Gómez Adrian, Jon Ander

ACADEMIC YEAR: 2022/2023

Resum

Les ones gravitacionals són una predicció fonamental de la teoria de la relativitat general d'Einstein, i la seua detecció ha obert noves vies per a estudiar l'univers. Els mètodes tradicionals per a detectar ones gravitacionals involucren l'anàlisi de les dades en brut a través de diverses tècniques de processament de senyals. No obstant, les tècniques d'aprenentatge profund han mostrat resultats prometedors en tasques de reconeixement de patrons i classificació, la qual cosa les converteix en un candidat adequat per a la detecció d'ones gravitacionals. En aquest treball, ens centrem en dissenyar i entrenar xarxes neuronals per a reconèixer senyals d'ones gravitacionals en les dades. Explorarem diferents arquitectures i hiperparàmetres per a optimitzar el rendiment de la xarxa. Els resultats mostren que les tècniques d'aprenentatge profund poden proporcionar un enfocament eficient i precís per a la detecció automàtica d'ones gravitacionals, la qual cosa pot beneficiar el camp de la física teòrica.

Paraules clau: Reconeixement de patrons, Aprenentatge profund, Aprenentatge automàtic, Xarxes neuronals, Ones gravitacionals, Detecció automàtica, Processament de senyal

Resumen

Las ondas gravitacionales son una predicción fundamental de la teoría de la relatividad general de Einstein, y su detección ha abierto nuevas vías para estudiar el universo. Los métodos tradicionales para detectar ondas gravitacionales implican el análisis de los datos en bruto a través de diversas técnicas de procesamiento de señales. Sin embargo, el aprendizaje profundo ha demostrado resultados prometedores en tareas de reconocimiento de patrones y clasificación, con resultados prometedores en detección de ondas gravitacionales como muestran algunos trabajos publicados hasta la fecha. En este trabajo, nos enfocamos en diseñar y entrenar redes neuronales para reconocer señales de ondas gravitacionales en los datos recogidos en los interferómetros. Exploramos diferentes arquitecturas e hiperparámetros para optimizar el rendimiento de la red. Los resultados muestran que las técnicas de aprendizaje profundo pueden proporcionar un enfoque eficiente y preciso para la detección automática de ondas gravitacionales, lo que puede beneficiar el campo de la física teórica.

Palabras clave: Reconocimiento de patrones, Aprendizaje profundo, Ondas gravitacionales, Procesamiento de señales, Redes neuronales

Abstract

Gravitational waves are a fundamental prediction of Einstein's theory of general relativity, and their detection has opened up new avenues for studying the universe. The traditional methods of detecting gravitational waves involve analyzing the raw data through various signal processing techniques. However, deep learning techniques have shown promising results in pattern recognition and classification tasks, making them a suitable candidate for detecting gravitational waves, as some published works show. In this work, we focus on designing and training neural networks to recognize gravitational wave signals in the data collected at the interferometers. We explore different architectures and hyperparameters to optimize the network's performance. The results achieved so far show that deep learning techniques can provide an efficient and accurate approach

to automatic detection of gravitational waves, which can benefit the field of theoretical physics.

Key words: Pattern recognition, Deep learning, Gravitational waves, Signal processing, Artificial Neural Networks

Contents

Appendix

List of Figures

List of Tables

CHAPTER 1
Introduction

1.1 Motivation

Gravitational waves are 'ripples' in space-time caused by some of the Universe's most violent and energetic processes. They were first proposed by Oliver Heaviside in 1893 and later by Henri Poincaré in 1905 as waves similar to electromagnetic waves but the gravitational equivalent. Albert Einstein predicted the existence of gravitational waves in 1916 in his general theory of relativity [1]. Einstein's mathematics showed that massive accelerating objects (such as neutron stars or black holes orbiting each other) would disrupt space-time so that 'waves' of undulating space-time would propagate in all directions away from the source. These cosmic ripples would travel at the speed of light, carrying information about their origins and clues to the nature of gravity itself. The strongest gravitational waves are produced by cataclysmic events such as colliding black holes, supernovae (massive stars exploding at the end of their lifetimes), and colliding neutron stars. Other waves are predicted to be caused by the rotation of neutron stars that are not perfect spheres, and possibly even the remnants of gravitational radiation created by the Big Bang.

The field of gravitational wave (GW) detection has seen tremendous progress in recent years with the help of advanced sensing technology and data analysis methods. One of the key challenges in GW detection is the need to distinguish the tiny signals from the noise in the data shows much more energy than the signal. Deep learning has proven to be an effective tool for solving complex problems in many fields, including computer vision, speech recognition, and natural language processing.

In this project, we aim to apply deep learning techniques to the problem of GW detection. The goal is to develop a deep-learning model that can accurately detect GW signals in the presence of noise without requiring manual feature extraction or thresholding. The resulting system will be compared to traditional GW detection methods to assess its performance and potential impact on the field [2].

1.2 Main goals

The main goals of this work can be summarized in these points:

- Design and develop deep-learning classifiers to distinguish between spectrograms including signals corresponding to gravitational waves and spectrograms without signals (i.e., background noise).

- Improve the performance of existing classifiers developed to detect gravitational waves.

1.3 Document structure

The following document is divided into seven chapters. The first chapter exposes the origin of gravitational wave events, explains the motivations of our work, and indicates the structure used. Following, Chapter 2 summarizes the state of the art of the two most widely used approaches for detection, the classic approach, which uses the matched filtering approach, and the new approach which uses deep-learning technologies. Next, Chapter 3 exposes the generations of the two datasets used, train and test datasets, and also describes the partitioning used for this data. Then, Chapter 4 describes the deep learning architectures used for our work, distinguishing mainly two types, the topologies of pre-trained models, and the topology of the tailored model. Next, Chapter 5 exhibits the technologies used to carry out the models' training, the data augmentation techniques used, and defines the deep-learning pipeline used to train every model tested. Following, Chapter 6 presents and analyzes the results obtained distinguishing between each architecture used, trying to find correlations between the models' hyperparameters and the performance. Finally, Chapter 7 concludes our work by comparing the results obtained with the goals proposed at the beginning of the work and outlines possible future lines of work.

CHAPTER 2
State of the art

The detection and analysis of gravitational wave signals are inherently challenging tasks due to their weak nature and the presence of background noise. Traditional detection methods rely on complex data processing techniques and waveform models to identify and extract the faint signals buried within the data. However, the emergence of deep learning has provided a promising alternative approach to automatic detection of gravitational waves, leveraging the power of artificial neural networks to learn and recognize patterns directly from the data.

2.1 Traditional Approaches to Gravitational Wave Detection

Prior to the advent of deep learning, the detection of gravitational waves mainly relied on matched filtering techniques, which involved cross-correlating the detector output with a set of theoretically predicted waveform templates. This approach required precise knowledge of the waveform models and was computationally intensive, limiting its scalability and adaptability to various astrophysical scenarios. Additionally, traditional methods often faced challenges in dealing with non-Gaussian noise and complex signal morphologies, leading to potential false positives or missed detections.

Matched filtering takes advantage of the fact that the gravitational wave signals are expected to closely resemble the waveform templates. By comparing the detector output with the templates, it is possible to identify the presence of a gravitational wave signal in the data. The process involves maximizing the signal-to-noise ratio by optimizing the alignment and phase of the templates with the data.

One of the key strengths of matched filtering is its ability to achieve optimal signal-to-noise ratio for signals that closely match the templates. This approach is highly effective for well-modeled and accurately predicted waveforms, such as those from binary black hole or binary neutron star mergers. In these cases, matched filtering can provide high detection sensitivity and low false positive rates.

However, the computational cost associated with matched filtering can be prohibitive, particularly when searching for gravitational waves in a wide range of astrophysical scenarios. The template bank used for cross-correlation can be vast, consisting of numerous templates representing different waveform parameters, such as masses, spins, and orientations, limiting the scalability and real-time capabilities of matched filtering notoriously.

Furthermore, matched filtering is highly dependent on the accuracy of the waveform models used to generate the templates. Any discrepancies between the true waveforms and the predicted templates can result in reduced detection sensitivity or missed detec-

tions. This greatly diminishes the ability of this method to detect gravitational waves with unfamiliar characteristics, thereby reducing its generalization capabilities.

Another limitation of matched filtering is its susceptibility to non-Gaussian and complex noise sources. Gravitational wave detectors are subject to various sources of noise, including instrumental noise, environmental disturbances, and astrophysical backgrounds. Non-Gaussian and non-stationary noise properties can impact the performance of matched filtering, leading to potential false positives or missed detections. The challenge of accurately characterizing and subtracting noise sources while preserving the weak gravitational wave signals is a significant hurdle for traditional detection methods.

Despite these limitations, matched filtering remains the benchmark technique for gravitational wave detection due to its theoretical optimality when the waveform models are accurate [3, 4, 5]. In some cases, matched filtering is even combined with AI techniques such as Convolutional Neural Networks (CNNs) [6]. However, the increasing availability of large datasets of labelled gravitational wave signals and advancements in computational resources have paved the way for exploring alternative approaches, such as deep learning, to complement and enhance the capabilities of matched filtering. Deep learning offers the potential to overcome the limitations of traditional methods by automatically learning discriminative features from raw data, addressing the challenges posed by noise and waveform variations in gravitational wave detection and potentially improving the computational efficiency.

2.2 Deep Learning in Gravitational Wave Detection

Deep learning has shown remarkable success in various fields, including computer vision, natural language processing, and speech recognition. Its application to gravitational wave detection has also gained significant attention in recent years. By utilizing neural networks, deep learning models can learn intricate features and patterns from the raw gravitational wave data, enabling the automated detection and analysis of signals.

While gravitational waves represent a relatively new domain of scientific exploration, the prevailing technique for detecting them has been matched filtering. However, deep learning is increasingly gaining popularity in this field due to its notable benefits, including enhanced generalization capabilities and its potential to uncover complex patterns that may be difficult to capture with traditional methods. As a result, deep learning has already been applied to signal detection, parameter estimation, noise reduction, and signal extraction. Following, we will discuss the most relevant applications associated with our work.

Firstly, one interesting application of deep learning that is utilized in this field is signal extraction. This task, also known as denoising, consists of removing the noise from a gravitational wave signal and isolating the signal corresponding to the merger. Regarding this task, the following three papers describe their methodologies and present their results using DL models such as recurrent neural networks (RRNs), CNNs or Denoising Auto-Encoders (DAEs) [7, 8, 9].

Moreover, DL has also been applied to parameter estimation tasks. The goal is to find the set of parameters, such as the mass or distance of the source, that best fits the data obtained from the detectors, to enable profounder understanding of the underlying phenomena. Dedicated to this subject, these three papers present highly promising findings that offer a glimpse into a future full of great potential [10, 11, 12].

Furthermore, DL is also gaining popularity in applications including noise reduction. This task focuses on decreasing noise in gravitational wave signals while preserving the

integrity of the signal. In this paper [13], this is approached by synthetic signal injection and subsequent regeneration of the parameters. Researchers found that the wanted signal is indeed easier to detect while parameters being consistent.

In addition, DL has also been used to try to mitigate glitches in gravitational signals. Glitches are the product of short-lived linear and non-linear couplings among the detector control systems. This paper uses DL to decrease the effect of glitches in signals to improve data quality and detectability [14].

Finally, signal detection is perhaps one of the primary applications of deep learning in the realm of gravitational waves. Essentially, this task involves determining whether a given signal contains a gravitational wave or is simply background noise. Numerous deep learning models have been employed for this purpose, such as CNNs [15, 6, 16], residual neural networks [17], Wave Net based architectures [18], AutoEncoders [19] or Transformer based models [20].

CHAPTER 3

Data Collection and Preprocessing for Neural Network Training

This chapter discusses the data provided by the "Gravitational Wave Open Science Center" (GWOSC) and the generation of a synthetic dataset consisting of binary black hole (BBH) merger spectrograms to train a neural network-based image classifier. Additionally, it will delve into the technical details of how this dataset was used to train the classifier. The creation of this artificial data collection is crucial to efficiently and accurately classify natural gravitational waves from noise signals and other types of interference.

3.1 GWOSC data

The "Gravitational Wave Open Science Center" (GWOSC), formerly known as the LIGO Open Science Center, was created to provide public access to gravitational-wave data products. The collaborations running LIGO, Virgo, GEO600, and KAGRA have all agreed to use GWOSC services as the primary access points for public data products. This collaborative approach benefits users by creating a uniform interface to access data from multiple observatories and provides cost savings to the various observatories by sharing the tools, services, and human resources [21].

In this study, we utilized data obtained from the Gravitational Wave Open Science Center (GWOSC), specifically the GWTC1 and GWTC2 catalogs, which contain information on binary black hole (BBH) and binary neutron star mergers. Our signal acquisition method primarily involved the utilization of the ***fetch_open_data*** function from the GWPY[1] library, which facilitates access to the GWOSC data archive. Specifically, we extracted a time window of ± 4 seconds centered on the time of the merger event.

An alternative approach to signal acquisition involved direct download from the GWOSC website utilizing the "Timeline Queries" feature. This approach involved specifying the desired time range, in our case, a ± 4 second window around the merger event. However, it should be noted that this approach results in the acquisition of a file covering a broader time range (i.e., 4096 seconds) instead of just the specified window of interest. Furthermore, it is possible that the desired time window may not be fully contained in a single file, necessitating the download of multiple files.

After obtaining an 8-second signal, the signal is further segmented by dividing it into 0.2 second windows, with the window shifted by 0.1 seconds. This segmentation yields a total of 79 slices from the original signal. For each segmented slice, the spectrogram's time resolution and frequency resolution are calculated.

[1] https://gwpy.github.io/

Subsequently, a set of signal processing techniques is applied to the data obtained from each detector to prepare it for the q-transform. Specifically, the amplitude spectral density (ASD) is computed for the data coming from each detector using the Welch method. The data is then whitened using the computed ASD, followed by the application of band-pass and notch filters to remove noise in determined frequencies. Next, the q-transform is performed on the signal, and subsequently, the spectrogram matrix of each detector is normalized.

By utilizing this approach, we created a testing data set that included a total of 3792 images comprising 11 BBH merger signals from GWTC1 and 37 merger signals from GWTC2. We label as signal the images generated 0.1 seconds before and after the merger and the image precisely at the merger time. Due to this, we obtain an unbalanced test set since, for every 79 images generated from an event, three are labeled as a signal, and 76 are labeled as noise. Therefore, we have 144 signal images and 3648 noise images.

3.2 Synthetic data set generation

In our case, an artificial data set is vital to develop and train an efficient and accurate classification model that can distinguish gravitational waves from noise signals. It has been observed that data is an essential part of the training of a deep-learning model. If we control the generation of data, we control quality and quantity. Managing quality enables us to provide the model with varied data for a correct generalization. Controlling quantity allows us to provide the model with a large enough dataset to have enough samples to learn.

We begin by describing the generation of the data set used in our analysis. The synthetic dataset generation was extracted from the GitHub of a researcher participating in this paper [12]. Moreover, all waveforms used in the classification data set were obtained using pyCBC[2] and GWpy libraries, using the SEOBNRv4[3] approximant.

In order to combine the data from all three detectors (Hanford, Livingston, and Virgo) we select coincident segments of 500 s of noise from all detectors starting at a certain t_{GPS} time. Defining τ to be some time from the start of our noise segment, we randomly select $\tau_0 \in$ [len(signal), 500-len(signal)] s and isolate the window $[\tau_0 - 4\text{ s}, \tau_0 + 4\text{ s}]$. Then, the resulting background strain data, n_H, n_L and n_V, for Hanford, Livingston and Virgo interferometers respectively, are whitened through inverse spectrum truncation, using their own amplitude spectral density (ASD). Then, we apply a band pass filter from 20 Hz to 300 Hz, as well as notch filters at the individual frequencies 60 Hz, 120 Hz and 240 Hz.

For the generation of the waveform signal strain h, a random pair of black hole masses $m_1 \in [4, 50]\ M_\odot$[4] and $m_2 \in [m_1, 50 - m_1]\ M_\odot$ are selected for a BBH merger with luminosity distance $d_L \in [325, 675]$ Mpc and inclination $\iota \in [0,\ 2 * \pi]$. Once the signal waveform has been generated, we incorporate each detector's antenna power and other characteristics into the time series. h_H, h_L, and h_V.

After generating the signal waveform and noise, we align them such that the maximum amplitude of the signal waveform occurs at τ_0. This aligned waveform is then injected into the background noise, resulting in the composite signal $S = h + t$. We per-

[2] https://pycbc.org/

[3] SEOBNRv4 approximant is the improved version of SEOBNRv2. It is a waveform model that predicts the gravitational waves emitted by the binary system as it spirals towards merger.

[4] A solar mass is equal to 1.989e30

form this alignment and injection process for each detector, obtaining three composite signals: S_H, S_L, and S_V.

The injected signal is whitened using the ASD of the selected noise strain segments, n_H, n_L and n_V and the same filtering process is undertaken. Following this, the multi-Q transform is calculated for the $[\tau_0 - 0.16\text{ s}, \tau_0 + 0.04\text{ s}]$ interval in the composite signals S_H, S_L, and S_V, and a spectrogram is produced.

Later, we emulate the sky position for the signal by randomly choosing one of three detectors as a reference, and shift the beginning of the other two time series according to their time delay with respect to the reference detector. Once the three spectrograms are produced, they are combined into a 3-dimensional array in such a way that each of them is represented by a certain color channel in an RGB image. Specifically, Hanford, Livingston, and Virgo datasets are mapped into the Red, Green, and Blue channels respectively. A second spectrogram without the injected signal is also generated for the same interval. Both spectrograms are saved as images and appropriately labelled as "signal" and "background". This process is iterated 10000 times to build our dataset, generating a total of 20000 images, 10000 noise images and 10000 signal injected images[12]. Generated images have size 275x275 with 3 channels, therefore having shape 3x275x275.

In the figure below, we compare an image with pure noise versus an image with the same frame of noise, but with an injected gravitational wave signal. The above-mentioned image pre-processing enables us sometimes to see a noticeable spike, signifying the presence of a gravitational wave within the data.

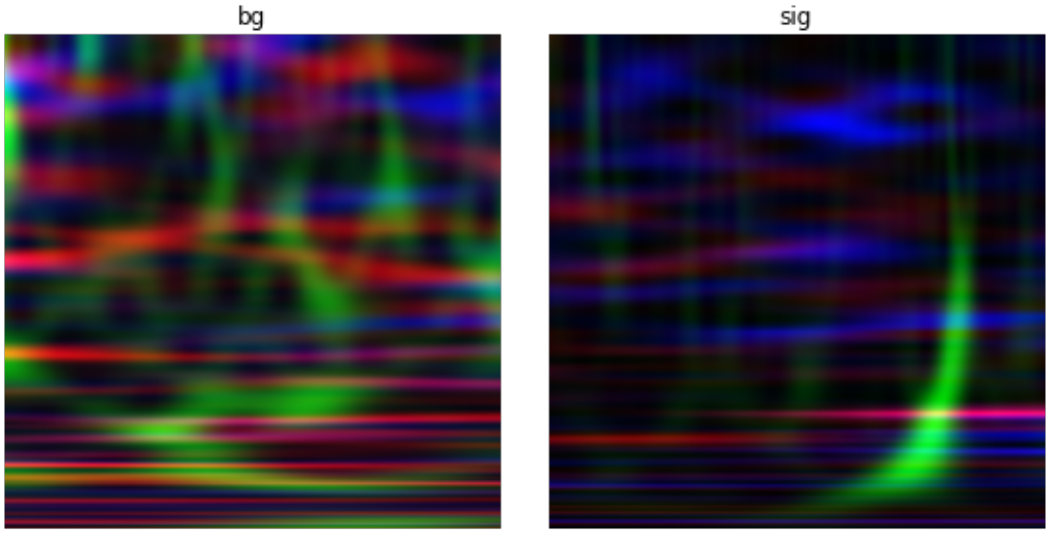

Figure 3.1: RGB image from background labeled spectrogram (left) as compared with a spectrogram where a GW waveform was injected into real conditions noise (right).
https://arxiv.org/pdf/2011.10425.pdf

3.3 Train, Validation, and Test set Partitioning

The final configuration of the train, validation, and test sets is depicted in the figure below. The train and validation sets are derived from the synthetic dataset, a 20000 image set with a size around 1 GB. In this set, 85%, or 17000 images, is allocated to the training of the model, and the remaining 15%, or 3000 images, is allocated to the validation set, used to test the model at the end of each epoch.

In contrast, the "Real Dataset", consisting of images obtained from actual measured signals extracted using the ***fetch_open_data*** function in the GWPY library, comprises a total of 3792 images. The total size of the "Real Dataset" is approximately 150 MB and is entirely dedicated to the test set.

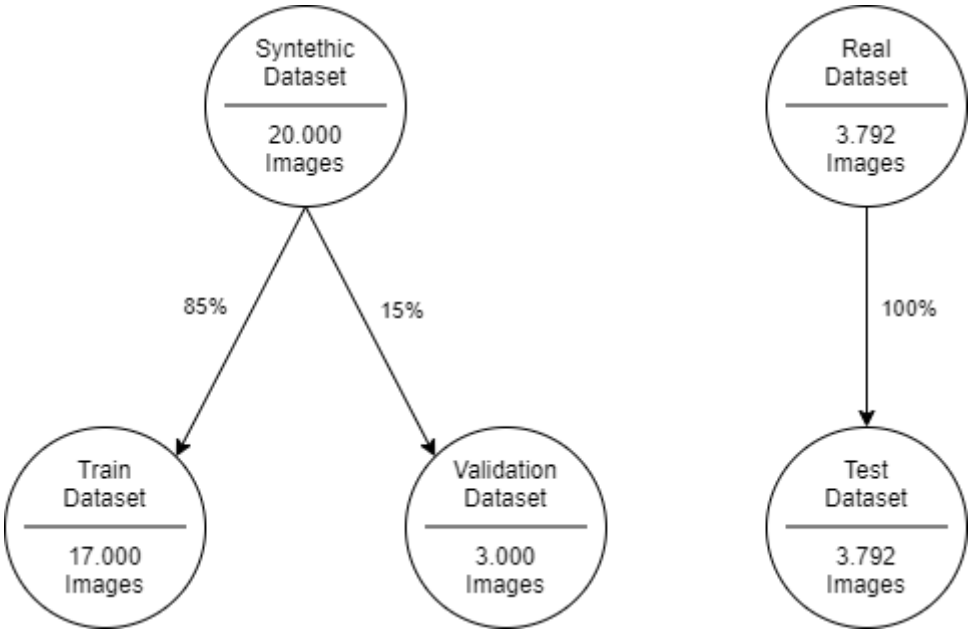

Figure 3.2: Dataset Partitioning

CHAPTER 4
Deep Learning Topologies

In this chapter, we are going to comment about the neural network architectures we have used to solve this task.

Concretely, Section 4.1 introduces the concept of deep learning and provides insights into its current applications. Following, Section 4.2.1 provides an analysis of the ResNet architecture, stating its main characteristics and its advantages over conventional CNNs. Next, Section 4.2.2 explains the Transformer architecture and its use in the vision field. Later, in Section 4.3, we comment on the tailored topology we built for this problem. Lastly, in Section 4.4 we present the accuracy and loss functions used across all the models.

4.1 Introduction

In recent years, deep learning has emerged as a powerful tool for solving complex problems in a variety of fields. Fields including image recognition, natural language processing, and speech recognition have undergone revolutionary changes as a result of its capacity to automatically learn and extract complex patterns from vast datasets. Broadening the range of deep learning applications, this section focuses on its application to the automatic detection of gravitational waves.

In this section, we present a comprehensive overview of various deep learning topologies employed in the automatic detection of gravitational waves. We examine different network topologies, their traits, and each one's advantages and disadvantages. The presented topologies serve as a foundation for the subsequent chapters, where we dig into the experimental setup, and performance evaluation of each model.

This section aims to provide a thorough grasp of the deep learning frameworks and approaches that are applied to autonomous gravitational wave detection.

4.2 Topologies of Pretrained Models

In this section, we are going to comment about the architecture of the pretrained models we have used. The methodology used in these pretrained networks involved freezing the weights of the model and subsequently change the last fully connected layer. Moreover, as the models we used were trained on the ImageNet dataset, a dataset consisting of 1000 different classes, we needed to reduce the output of the final layer to 1, adapting it to our needs in binary classification. Later, we included the option to determine the epoch at

which the model weights could be unfrozen as a configurable parameter in our training pipeline. At this point, the whole model was set to be trained on our data.

4.2.1. The ResNet Architecture

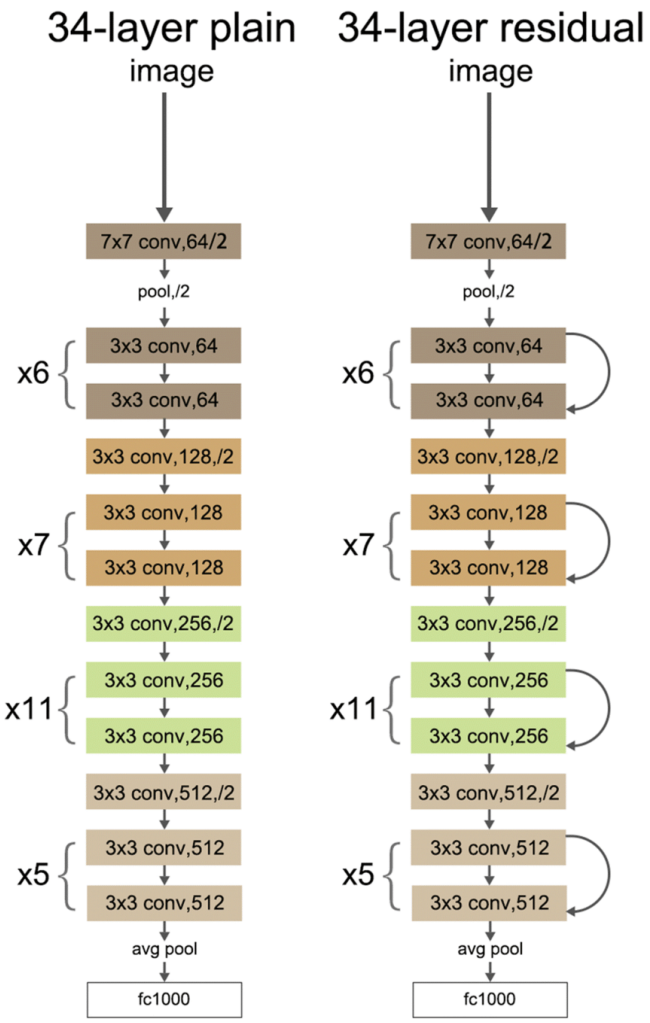

Figure 4.1: Comparison of CNN architecture vs ResNet architecture.
https://www.researchgate.net/figure/Two-structures-are-for-ResNet-34-left-and-ResNet-50-101-152-r
fig3_344392249

ResNet (Residual Neural Network) is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper "Deep Residual Learning for Image Recognition" [22]. Primarily used in computer vision problems, in 2015 ResNet achieved remarkable success by winning the ImageNet competition.

Traditional deep neural networks often suffer from the problem of vanishing or exploding gradients, which makes it challenging to train networks with numerous layers. This can be seen in Figure 4.2, where we observe that the scaling of CNNs can lead to worse results. The ResNet's main characteristic is focused on improving the ability of neural networks to be deeper. It is characterized by incorporating skip connections that conform residual blocks. By stacking these blocks, the architecture enables to go beyond the depth restrictions of conventional CNNs.

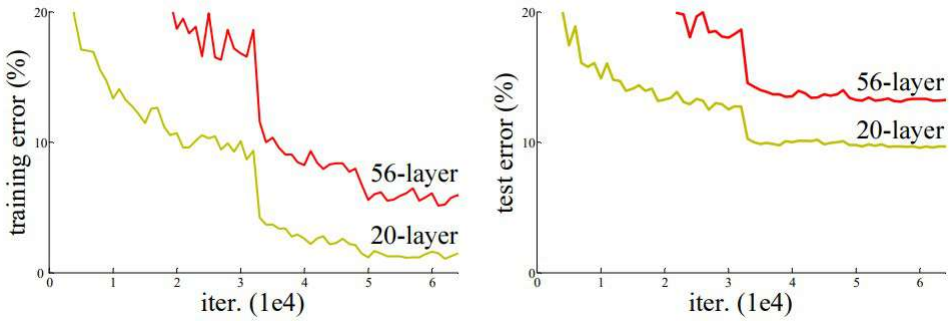

Figure 4.2: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks.
https://arxiv.org/abs/1512.03385

Residual learning is a fundamental idea behind the ResNet architecture. Traditional deep neural networks aim to learn direct mappings from input to output, which can become challenging as the network depth increases. Residual learning introduces the concept of residual mappings, where the network is trained to learn residual functions rather than direct mappings. Utilizing residual mappings allows the network to concentrate on understanding changes or discrepancies between the input and the desired output, which helps in mitigating the degradation issue. Figure 4.1 shows the structure difference between classical CNNs and ResNets, where skip connections are clearly visible.

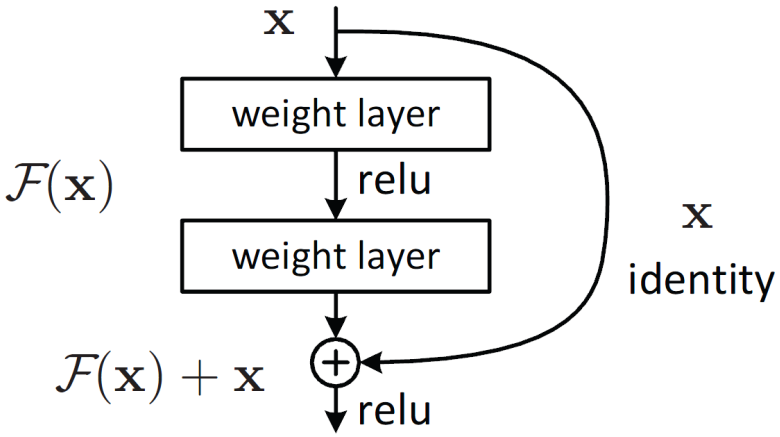

Figure 4.3: Residual learning: a building block.
https://arxiv.org/abs/1512.03385

The degradation issue arises when the accuracy of a deep network saturates or even degrades as the network depth increases. This behavior is counterintuitive because a deeper network should presumably be able to capture more complex features. Residual learning addresses this problem by introducing identity mappings and skip connections. The skip connections allow the network to transmit information directly from earlier layers to later layers, enabling the network to learn residual functions effectively.

The above-pictured Figure 4.3 illustrates that by incorporating skip connections, the ResNet topology experiences a transformation of certain input functions from $H(x) = f(x)$ to $H(x) = f(x) + x$. This modification effectively facilitates the smoothing of the gradient flow, thereby enabling the network to reach scalability beyond the threshold of 100 layers [23].

4.2.2. The Visual Transformer Architecture

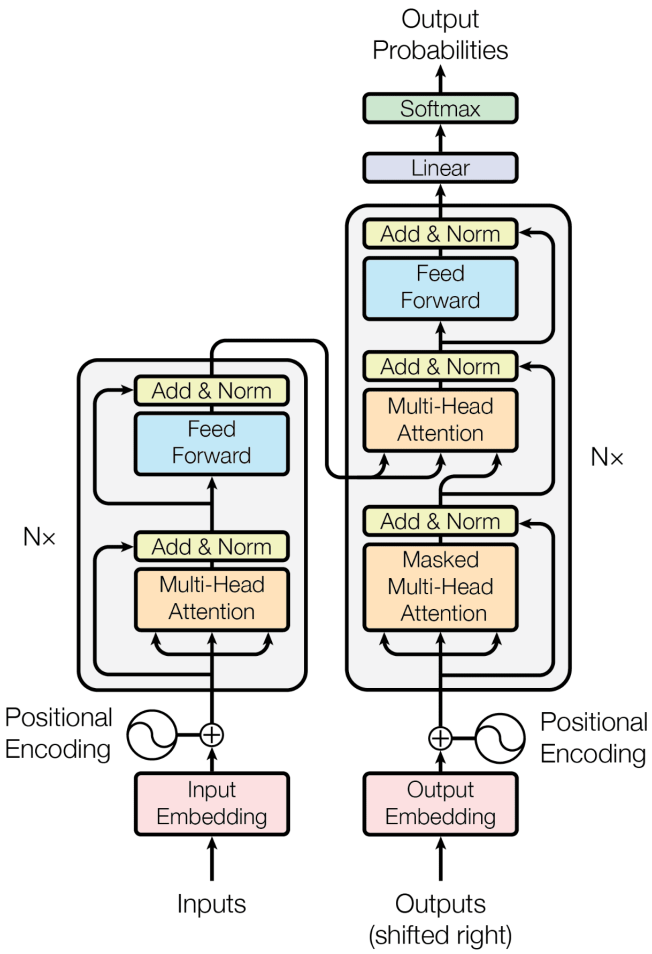

Figure 4.4: The Transformer - model architecture.
https://arxiv.org/abs/1706.03762

The transformer architecture [24], initially introduced for natural language processing tasks, revolutionized sequence modeling by substituting recurrent neural networks (RNNs) with self-attention mechanisms. Unlike traditional sequential models that process inputs sequentially, transformers can simultaneously capture dependencies over the entire sequence. They can identify long-range relationships effectively because their parallelism allows for efficient training and inference.

The transformer architecture consists of two main components: the self-attention mechanism and the feed-forward layers, both depicted in Figure 4.4.

The self-attention mechanism allows each token in the input sequence to attend to all other tokens, capturing their relationships. It determines the value or relevance of tokens to one another by computing attention weights between them. It can also perceive local and global dependencies by incorporating information from various positions in the sequence. Thanks to this mechanism, vanishing gradient problems with previous architectures, such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTMs), disappeared. Also, self-attention mechanisms implemented both in the encoder and in the decoder are, in fact, multi-head attention, which means that attention vectors are computed in parallel. This solves the time-consuming aspect of the architectures mentioned above.

Following the self-attention mechanism, transformer architectures commonly apply feed-forward layers to process the tokens and capture complicated relationships, applying non-linear alterations to the token representations.

Transformers are designed in an encoder-decoder architecture. Here, the encoder maps an input sequence of symbol representations x to a sequence of continuous representations z. Given z, the decoder generates an output sequence y of symbols one element at a time [24].

The Vision Transformer (ViT) [25] is simply an adaptation of this architecture to enable it to work with images. However, given the quadratic complexity of computing the attention matrix, researchers in this work propose a method to alleviate this computation by splitting images into smaller patches, such as 14x14 or 16x16. Next, these patches are flattened and passed through a trainable linear projection that maps these vectors into the dimensions employed by the architecture. Given that Transformers do not encode any positional information, a positional encoding layer is introduced which assigns each patch a position in the original image. At this point, the embeddings obtained are fed into a stack of Transformer encoder layers to capture local and global image dependencies. The output of the last encoder layer is then fed into a classification head, usually a fully connected layer. With this slight modifications, Vision Transformers are now obtaining state-of-the-art results in image classification benchmarks such as ImageNet or CIFAR-100[1].

A significant point worth noting, to be discussed in more detail in subsequent chapters regarding our findings, is that the authors of the Vision Transformers paper mention that this architecture performs exceptionally well when a vast amount of data is available. In contrast, ResNets tend to achieve better results with a smaller amount of data. In our particular situation, the results obtained with Transformers are comparable to those of ResNets. However, it is possible that if we were to train Transformers with a significantly larger dataset, they could potentially outperform ResNets significantly.

4.3 Tailored Topologies

In this section, we are going to explain the architecture we designed for this project. First, we are going to explain each layer separately. Then, we'll zoom out and provide an overview of how all the layers come together to form the complete architecture.

4.3.1. Convolutional Layer

A convolutional layer consists of a set of learnable filters known as kernels. Each filter is a small matrix of weights that slides over the input data, performing element-wise multiplications and adding the results to produce a single value, as shown in Figure 4.5. This operation is known as the convolution operation. The convolutional layer's purpose is to identify local patterns and spatial correlations in the input data. The layer learns to recognize various features such as edges, textures, and shapes by convolving the filters across the input. These features are essential for subsequent layers to learn higher-level input representations.

[1] On this website we can check the best-performing architectures used in a broad set of different benchmarks https://paperswithcode.com/task/image-classification

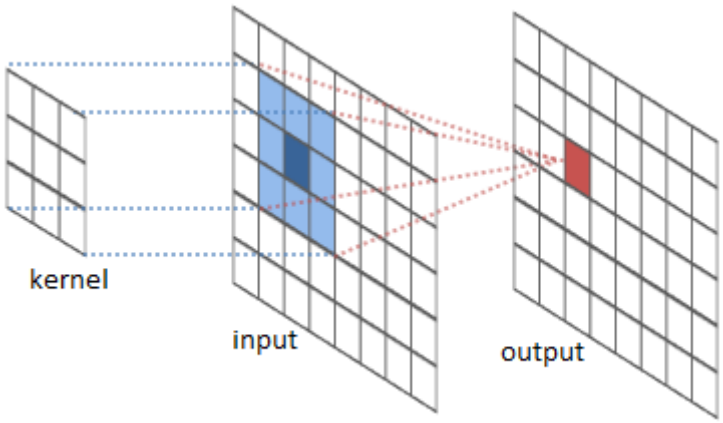

Figure 4.5: Convolutional Kernel.
http://intellabs.github.io/RiverTrail/tutorial/

4.3.2. Dropout layer

A dropout layer is a regularization technique commonly used in deep learning models, including convolutional neural networks (CNNs), to prevent overfitting and improve generalization. It works by randomly turning off a fraction of the neurons in a neural network during training.

During the forward pass of training, a dropout layer randomly sets a fraction of neurons to zero with a certain probability. This probability is typically set between 0.2 and 0.5, and affects different neurons with each model update. Furthermore, it is turned off during the evaluation, and the full network with all its neurons is used.

Dropout mitigates the network from relying too heavily on specific neurons, encouraging it to learn more robust and generalized features, as no single neuron can dominate the training process.

4.3.3. Pooling Layers

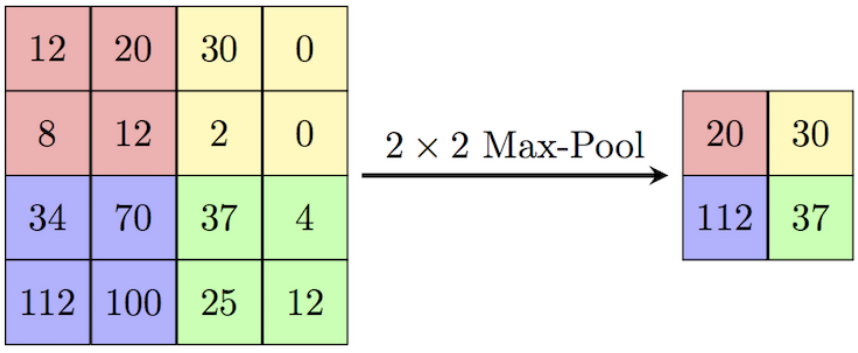

Figure 4.6: Max Pooling Layer.
https://paperswithcode.com/method/max-pooling

A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g., ReLU) has been applied to the feature maps output by a convolutional layer [26]. Its main purpose is to downsample the spatial dimensions of the input, reducing their size while retaining important features. The most commonly used are average pooling or max pooling, which we chose.

A sliding window moves across the input with a predefined stride value to perform max pooling. The maximum value is selected within each window, while the remaining values are discarded. This behavior can be seen in the figure above.

4.3.4. ReLU Activation Function

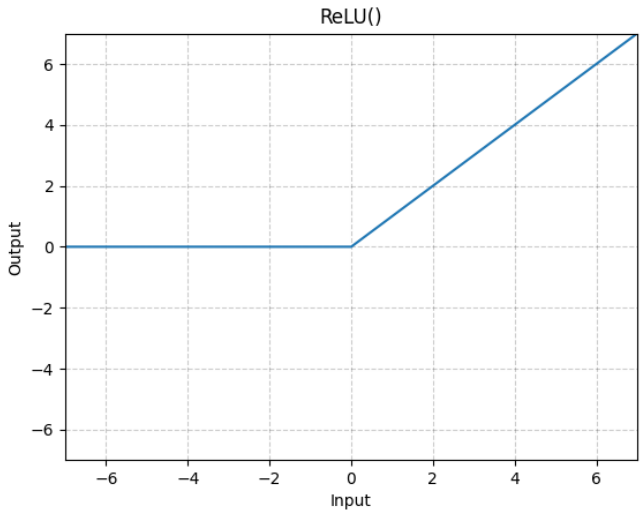

Figure 4.7: ReLU activation function.
https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

In a neural network, the activation function transforms the summed weighted input from the node into the node's activation or output for that input.

The rectified linear activation function, or ReLU for short, is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero $f(x) = max(0, x)$ [27]. Figure 4.7 is an illustration of this function.

It introduces non-linearity into the network by transforming the input signal simply and computationally efficiently.

The simplicity of the ReLU function allows for faster computation compared to more complex activation functions, such as sigmoid or hyperbolic tangent, and often, models using it achieve better performance.

4.3.5. Fully Connected Layer

A fully connected layer, or dense layer, is often used in the final layers of a neural network to map the learned features to the desired output, such as class probabilities or regression values.

The differential characteristic of this layer resides in that all its neurons are connected to all the neurons in the next layer. This is shown in Figure 4.8.

4.3.6. Complete architecture

For this problem, we have devised a convolutional neural network (CNN), a well-established architecture widely used in the scope of image recognition. In the former subsections, we

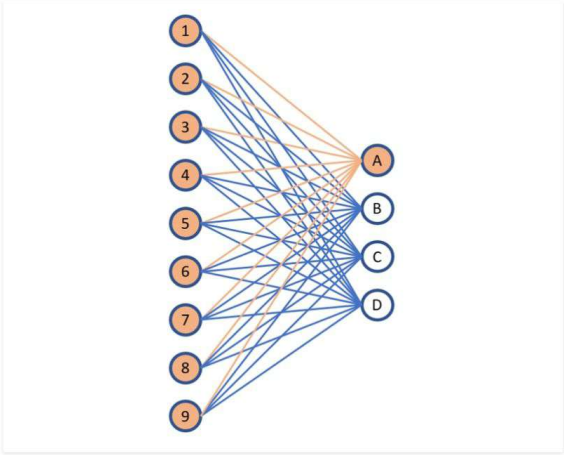

Figure 4.8: Fully Connected Layer.
https://builtin.com/machine-learning/fully-connected-layer

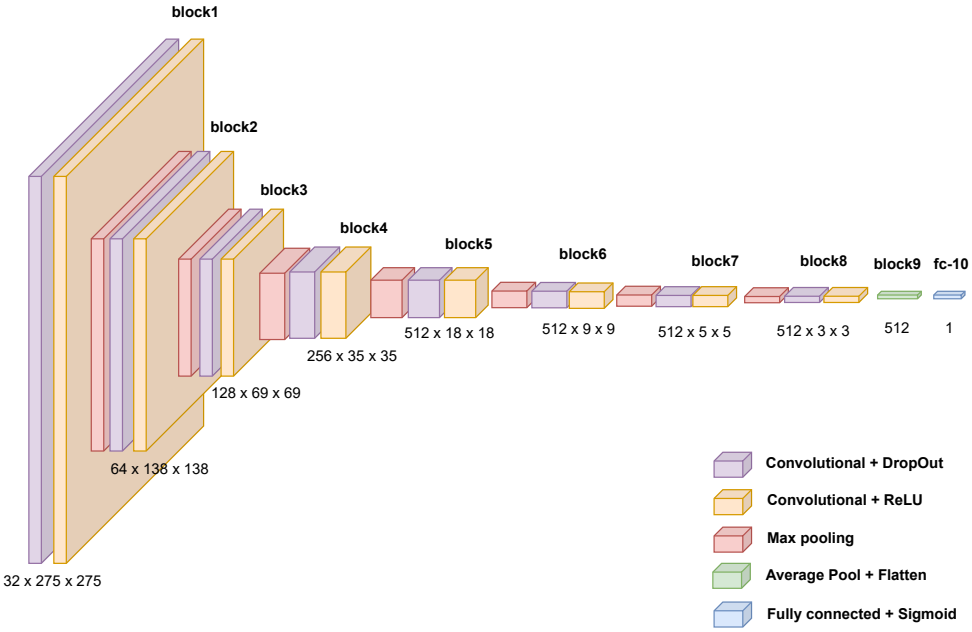

Figure 4.9: Architecture designed in this work.

explained every component of our architecture, so we won't delve too deeply into the specifics of each element.

Our architecture consists of nine layers, with the first eight exhibiting an identical structure. In contrast, the ninth and final layer presents a distinct configuration designed to reduce the input dimensions derived from the preceding layers. This ultimately yields a singular value, interpretable as a probability.

As can be seen in the diagram, our architecture follows the following structure: first, a convolutional layer followed by a dropout layer, and subsequently, another convolutional layer. Next is applied the ReLU activation function, following a max pool layer, and finally, a dropout layer. This structure is repeated eight times, increasingly reducing the dimensions. Finally, a fully connected layer is employed to reduce the dimensions

from 512 to 1, followed by the application of a sigmoid function to enable the interpretation of the output as a probability.

We adapted the original diagram [28] representing the VGG-16 architecture to our needs in order to present the custom architecture built in this work.

4.4 Common Accuracy and Loss functions

For both pre-trained models and customized architectures, we used the same evaluation and loss functions. In terms of the evaluation function, we utilized accuracy as the primary metric, which is determined by the ratio of accurate predictions to the total number of predictions made.

$$\frac{1}{N}\sum_{i=1}^{N} 1(y_i = \hat{y}_i)$$

where y_i expresses the target value, $\hat{y}_i$ the prediction, and N the number of instances.

It is important to note that the accuracy function we utilized in our research accepted the logits of the neural networks as an input. These logits were then subsequently passed through a sigmoid layer, resulting in an output that fell from 0 to 1. We employed a threshold value of 0.5 to assign class labels, meaning that any value higher than this threshold was labeled as class 1.

When it comes to the loss function, we chose for binary cross-entropy with logits, which combines a Sigmoid layer and the binary cross-entropy loss (BCELoss) into a single class. Compared to applying a single Sigmoid layer and then a BCELoss, this method provides more numerical stability [29]. We chose binary cross-entropy since it is well-suited for binary classification problems because it accurately measures the difference between the predicted probability and the actual class labels.

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where N is the total number of samples in the dataset, y_i is the true binary label of the i-th sample, which can be either 0 or 1, $\hat{y}_i$ is the predicted probability of the i-th sample belonging to class 1, and log is the natural logarithm function.

CHAPTER 5
Experimentation and evaluation

This chapter aims to present an in-depth analysis of the approaches used in the deep learning pipeline for training models to detect gravitational waves produced by binary black hole mergers. The chapter will address aspects including the libraries that have made the development of this work possible, the use of data augmentation, or the incorporation of approaches like unfreezing model weights for transfer learning models.

5.1 Used Technologies

In the context of employing a machine-learning framework, our study opted for PyTorch[1], a popular and flexible deep-learning platform with a Python-based interface. As part of the PyTorch ecosystem, we utilized associated packages such as torchvision[2], a package that includes data augmentation techniques and from which we have obtained the weights of the pretrained models, and torchmetrics[3] a collection of more than 90 PyTorch metrics from which we have used accuracy for binary classification. Furthermore, we leveraged sublibraries like torch.utils for efficient data loading and management. We added Matplotlib[4], a robust Python toolkit for producing static, animated, and interactive visualizations, to enhance our visuals. As a result, we were able to produce high-quality charts and figures for our analysis.

Furthermore, the image dataset employed for training our models was created using two libraries: PyCBC[5] and GWpy[6]. These libraries enabled the simulation of gravitational wave signals with specific attributes, such as the masses of both black holes, the distance to the merger, and positional parameters like right ascension and declination. By introducing more variation to these parameters, we can generate a diverse set of images, exposing our model to a wider range of scenarios. This is beneficial, as we do not want our model to be limited to images with particular characteristics; otherwise, it may struggle to accurately label images with variations, which would translate to a worse generalization.

To select a logical range of parameters, we have generated the images using the code from the paper "Exploring gravitational-wave detection and parameter inference using Deep Learning methods" [12] written by physicists from the University of Valen-

[1] https://pytorch.org/
[2] https://pytorch.org/vision/stable/index.html
[3] https://torchmetrics.readthedocs.io/en/stable/
[4] https://matplotlib.org/
[5] https://pycbc.org/
[6] https://gwpy.github.io/

cia, among others. This code is publicly available on the GitHub of one of the researchers, as already mentioned in Section 3.2.

5.2 Data Augmentation

Data augmentation is a technique for artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points [30]. This technique is used to help models generalize better, avoiding overfitting. It is crucial when available data is scarce, also enabling the gain of a few accuracy percentage points. This technique is not perfect, as we won't avoid biases present in the original dataset, and in some cases depending on the transformations we apply to our data, it can be computationally demanding. After experimenting with different kinds of data augmentation techniques, these are the ones we chose.

1. Data Augmentation 0.1: In this approach, the image is resized, transformed into a tensor, and subsequently normalized using the ImageNet mean and standard deviation parameters $mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]$. Using these parameters is a common practice, since they are calculated based on the dataset's millions of images. Moreover, the dataset used on the pre-training of every pre-trained model was normalized using these constants.

2. Data Augmentation 1.1: This method extends the previous technique, adding a random rotation between -45º and 45º. We developed both this method and the former.

3. Data Augmentation 2.0: Concerning this data augmentation, we also include image resizing and normalization, but additionally, we utilize a method from the PyTorch library called TrivialAugmentWide, characterized by being dataset-independent. This approach involves applying a unique transformation to an image by employing a collection of functions, including rotation, brightness adjustment, or contrast modification. It caught our attention because the PyTorch team used it to achieve state-of-the-art results in training a ResNet50. [31].

Our approach involves applying data augmentation 0.1 over the validation and test set, since we are not interested in artificially increasing the images in these sets, just normalizing them.

Conversely, we evaluate every data augmentation technique mentioned during training, and 1.1 and 2.0 yield superior outcomes, as expected. As these techniques increment images in the test set, the model benefits from the advantages mentioned above, such as better generalization and a few percentage points increment in the accuracy.

5.3 Model Training

In this section, we present a thorough explanation of the standardized pipeline used to evaluate deep learning models. Our objective was to have a common workflow applicable to every model, facilitating a better comparison of the differences in performance without the influence of external factors. To achieve this, we used four files, each one in charge of a different aspect, ***train.py***, ***data_setup.py***, ***model_builder.py*** and ***engine.py***.

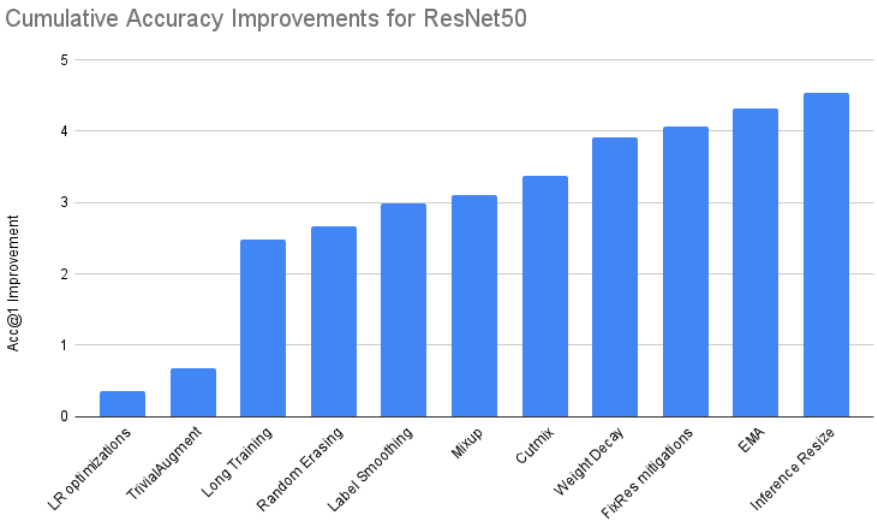

Figure 5.1: Effect of TrivialAugmentWide on ResNet50 accuracy (second bar). https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives#break-down-of-key-accuracy-improvements

The first file, ***train.py*** is the in charge of setting up everything for the model's training, carried out in the file ***engine.py***. It receives a set of hyperparameters such as the model's name, learning rate, optimizer, image size, batch size, number of epochs, and data augmentation, among others. It then communicates with ***data_setup.py*** and ***model_builder.py*** to set up data augmentation and model before passing the whole configuration to the file***engine.py***.

The following file is ***data_setup.py*** which is responsible for configuring the selected data augmentation type and creating the corresponding train, validation, and test data loaders. The chosen data augmentation is exclusively applied to the train data loader since for validation and test data loaders it is consistently applied data augmentation 0.1, which only normalizes the images as explained in the section before. Furthermore, as explained in section 3.3, the train and validation sets come from the same set of synthetic images using an 85% and 15% distribution respectively, whereas in the test set we use images derived from real detections.

The third file, ***model_builder.py***, is tasked with preparing the models for training. In the case of our custom CNN, it builds the model and configures the dropout parameter. In the case of the pre-trained models, it loads the pre-trained weights, freezes them, and adjusts the output fully connected block to be compatible with binary classification, since usually these models are trained with dataset ImageNet, which has one thousand different classes.

Finally, ***engine.py*** manages the training of the models and primarily consists of two functions, ***train_step*** and ***validation_step***.

train_step is employed to train one step of our model, a step corresponding to an epoch. However, as fitting an entire epoch within the model is not feasible, we iterate through batches, which are subdivisions of the whole set. We have executed every experiment with a batch size of 32 images; that is, for each pass of the ***train_step*** function, it will see 32 images at a time until it sees them all. During the function, we update the weights of our model, as long as they are not frozen, and once the function ends, we calculate the accuracy and the loss of the epoch.

validation_step follows the same structure, differing only on the weight update. Weights are not updated in this function since it helps us evaluate the model's performance trained on the current epoch to see if, as epochs progress, the model is improving or getting worse.

Once an epoch has passed through both functions, we compare the resulting validation accuracy against the best validation accuracy obtained up to that point. If the last epoch improves, we update our best accuracy so far and save the model dictionary.

It is worth noting that if we use pre-trained models, it could be the case during a specific epoch where we have to unfreeze the model weights, so the whole model weights start updating and not just the last fully connected layer.

Once this loop ends, the best model is loaded and evaluated on the test dataset, which consist of data the model has never seen. Accuracy and loss metrics extracted from this data tend to be the most relevant metrics when evaluating the performance of a model.

5.3.1. Utilized Hardware

Our work comprised two main stages, each utilizing distinct hardware solutions. The first phase, data generation, used Google Colab as the only hardware platform. Despite its limitations such as time and resources, which we were affected by, it had an important advantage, which was the ease of installation and use of PyCBC and GWpy libraries. At the time, these libraries had no Windows support, and installing Linux packages was problematic due to dependency and version conflicts. We did not have this problem in Google Colab, so we generated the needed data there.

It is worth mentioning that Google Colab makes use of notebooks, which have gained popularity recently. Notebooks structure as cells, each containing small code snippets. Thanks to this setup, the output generation tends to be quick which enables users to iteratively develop and test their code, making notebooks an excellent tool for prototyping and debugging.

The second phase involved training the model, for which we employed a private server equipped with an Nvidia GTX 3060 graphics card. All experiments were conducted using this hardware configuration. Conversely, when training the model on our private server, we opted for conventional Python files. In our opinion, the structure of a Python file was better suited for designing a machine-learning pipeline, and the use of notebooks did not offer any additional benefits in this context.

CHAPTER 6
Results

This chapter comments on the results obtained using different machine-learning models. It is crucial noting that although we performed various experiments with the same hyperparameters, the represented results are the best obtained. This is mainly due to the total number of epochs, which sometimes were too insufficient, so we had to train it for longer durations.

6.1 Presentation of Results

The table below represents the results obtained with pre-trained ResNet18 models. We can see that, although decreasing the image size could theoretically lead to worse outcomes due to the information loss, it seems like it is not an impediment to the model. Concerning the learning rate, we tried 1e-3, 1e-4, and 1e-5, and the best results were obtained with 1e-4. This impacted future experiments, since we tried many more times with a learning rate = 1e-4, which generally gave us better results. Regarding data augmentation, we see a slight correlation of better performance using type 2.0.

Size	Data_Aug	LR	Train_Acc	Val_Acc	Test_Acc
128x128	0.1	1e-4	91.85	94.02	92.88
128x128	1.1	1e-3	93.26	94.13	85.61
128x128	1.1	1e-4	94.44	93.55	83.53
128x128	1.1	1e-5	94.41	93.85	85.87
128x128	2.0	1e-4	95.09	92.83	93.09
275x275	0.1	1e-3	94.62	93.79	87.82
275x275	0.1	1e-5	96.68	93.34	87.08
275x275	1.1	1e-3	94.52	94.48	87.18
275x275	1.1	1e-4	95.98	94.27	84.32
275x275	1.1	1e-5	95.32	94.13	85.16
275x275	2.0	1e-4	94.98	92.86	94.17

Table 6.1: Results of Pretrained_ResNet18 with frozen_epochs=10.

On the following table, we can see the results given by the pre-trained ResNets34 models. In these results, we can see the same thing commented on the previous table about image size, which appears to have no effect in validation and test accuracy. It can be inferred that some weight decay has a positive effect on the test accuracy, since two of the best three results have some, with an optimal range likely between 1e-2 to 1e-3.

Size	Data_Aug	LR	Weight_Decay	Train_Acc	Val_Acc	Test_Acc
275x275	1.1	1e-4	0	94.13	93.96	92.65
275x275	2.0	1e-4	0	96.37	93.23	92.46
275x275	2.0	1e-4	1e-2	92.36	92.96	91.96
128x128	2.0	1e-4	0	95.07	93.54	93.91
128x128	2.0	1e-4	1e-2	94.05	92.83	94.3
128x128	2.0	1e-4	1e-3	93.96	93.27	94.38
128x128	2.0	1e-4	1e-4	93.84	93.03	93.12
128x128	2.0	1e-4	1e-5	94.74	93.48	89.63
64x64	1.1	1e-4	0	93.35	93.9	94.64
64x64	1.1	1e-4	1e-2	94.08	94.09	90.91

Table 6.2: Results of Pretrained_ResNet34 with frozen_epochs=10.

Regarding the table with data from our custom CNN, first of all, we can see that the results are slightly worse than those from ResNets. We can see an inverse correlation regarding dropout, clearly visible in the configuration with data augmentation 1.1, where the more we increment dropout, the worse results it gives.

It is essential to highlight that a learning rate of 1e-3 or higher is too high for this architecture, and models cannot learn. This can be seen in train and validation accuracy, although test accuracy is higher than any other model. This is because these models always label images as noise, the predominant class in the test set, containing 96% of images.

Data_Aug	LR	Dropout	Train_Acc	Val_Acc	Test_Acc
0.1	1e-3	0,25	49.22	49.94	96.19
0.1	1e-4	0.25	93.49	94.51	92.88
0.1	1e-4	0.5	92.57	93.34	92.94
0.1	1e-5	0,25	93.62	93.58	94.14
1.1	1e-3	0,25	50.43	51.07	96.22
1.1	1e-4	0.25	92.97	93.34	92.49
1.1	1e-4	0.3	92.09	93.65	90.39
1.1	1e-4	0.5	91.6	92.93	87.95
1.1	1e-5	0,25	91.71	92.38	91.07
2.0	1e-4	0,25	92.06	93.17	91.78
2.0	1e-4	0.5	88.84	90.21	90.86

Table 6.3: Results of small_custom_net with size=275x275.

The next table provides data associated to the experiments performed with Vision Transformers (ViT). The model ***vit-base-patch16-224*** was extracted from the transformers library from Hugging Face, while models ***vit_b_16*** and ***vit_b_32*** were extracted from the PyTorch torchvision pre-trained models. Upon analyzing the results, it seems that the best data augmentation for the latter two models was 0.1, which, as previously mentioned, consists of image normalization using ImageNet parameters. For the ***vit-base-patch16-224*** model, data augmentation techniques 1.1 and 2.0 seem to be the most effective. Based on the table's data, it could be inferred that weight decay has either no effect or a slightly negative effect. Nonetheless, it should be noted that the model configuration with the best results used weight decay, surpassing the results of the same configuration without any weight decay.

Comparing the results obtained using ViT with ResNet's results, there is no substantial difference, or even Transformers' performance is worse. As stated in section 4.2.2, ViT architecture works better with vasts amounts of data, around the millions or tens of millions of samples, but with a smaller dataset, its performance may be on par with alternative architectures, such as ResNets. This factor could explain the divergence in performance in competitions like ImageNet, where nowadays ViT holds the top results, achieving higher accuracies than ResNets, and our experiments, where they obtain similar results.

Model	Data_Aug	Weight_Decay	Train_Acc	Val_Acc	Test_Acc
vit-base-patch16-224	0.1	1e-3	90.58	91.35	80.93
vit-base-patch16-224	1.1	1e-3	90.58	91.35	80.93
vit-base-patch16-224	1.1	1e-4	93.52	93.78	89.89
vit-base-patch16-224	2.0	1e-4	92.65	93.24	89.5
vit_b_16	0.1	0	93.71	94.3	92.12
vit_b_16	0.1	1e-5	93.22	93.68	94.59
vit_b_16	1.1	0	92.71	93.1	91.86
vit_b_16	1.1	1e-5	92.19	93.17	91.02
vit_b_16	2.0	1e-4	91.26	91.32	89.94
vit_b_32	0.1	0	97.34	93.37	93.3
vit_b_32	0.1	1e-4	93.02	93.68	91.54
vit_b_32	1.1	0	92.5	93.1	91.96
vit_b_32	2.0	0	91.5	92.1	91.39
vit_b_32	2.0	1e-4	91.06	91.66	88.68

Table 6.4: Results of visual transformers with size=224x224, LR=1e-4.

In the following table, we present the best configuration of each model based on test accuracy. We can see that learning rates around 1e-4 yield the best results, as seen in the results' tables presented above for each model. As also commented in the results' table for each architecture, different data augmentations seem best suited for each model. Still, one thing that stands out is that while ResNet architecture benefits from data augmentation setups that artificially increase the number of samples, it seems like for Vision Transformers and Convolutional Neural Networks, that is not the case. These architectures benefit the most from data augmentation 0.1, which is just image normalization, while on ResNet architecture, data augmentations 1.1 and 2.0 tend to yield better results.

As noted above, we cannot select the model obtaining 96% test accuracy as the best small_custom_net since it is not a real accuracy. It results from predicting the most abundant class in the test dataset. Those models obtain this test accuracy while having 50% train and test accuracies, which is nonsense.

Model	Size	Data Aug	LR	Weight Decay	Dropout	Train Acc	Val Acc	Test Acc
Pretrained_ResNet18	275x275	2.0	1e-4	0	n/a	94.98	92.86	94.17
Pretrained_ResNet34	64x64	1.1	1e-4	0	n/a	93.35	93.9	94.64
small_custom_net	275x275	0.1	1e-5	0	0,25	93.62	93.58	94.14
vit_b_16	224x224	0.1	1e-4	1e-5	n/a	93.22	93.68	94.59

Table 6.5: Hyperparameters of the best model for each architecture.

6.2 Visualization of Predictions

In the following images, we can grasp the difficulty of the task presented to these models, contextualizing the results obtained, which are far from bad from our point of view.

The first figure is composed of three signal injected in noise images, being the next three pure noise. The first image of the first figure can be easily characterized as a signal image. It contains an ascending curve, which is the representative signal shape, so it is easy for the model to predict with a 93% probability that the image corresponds to the signal class.

But if we look closely at the following two images and compare them to the pure noise images, we could see they resemble more to pure noise, lacking any attributes that would imply the presence of a signal. As a result, the model fails these predictions and classifies them as signal with a 5% probability, or inversely, as background with 95% probability.

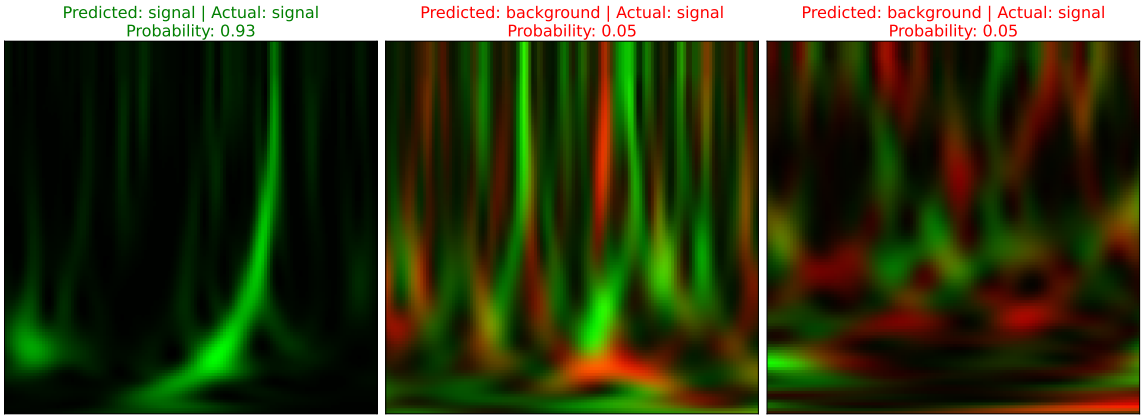

Figure 6.1: Signal images with the model's prediction.

Likewise, when analyzing the noise images, specially the last one, the slight upwards curve at the right of the image could remind us to a signal image. That is why the neural network gets confused and predicts signal instead of background with a 79% probability. The model handles the remaining two images perfectly, assigning a 99% and 98% probability for noise.

To sum up, the model usually performs a correct inference where images are distinguishable at first sight. Although the model tends to surpass human performance when the task becomes more challenging, it gets confused when signal and noise images present features of the opposite class.

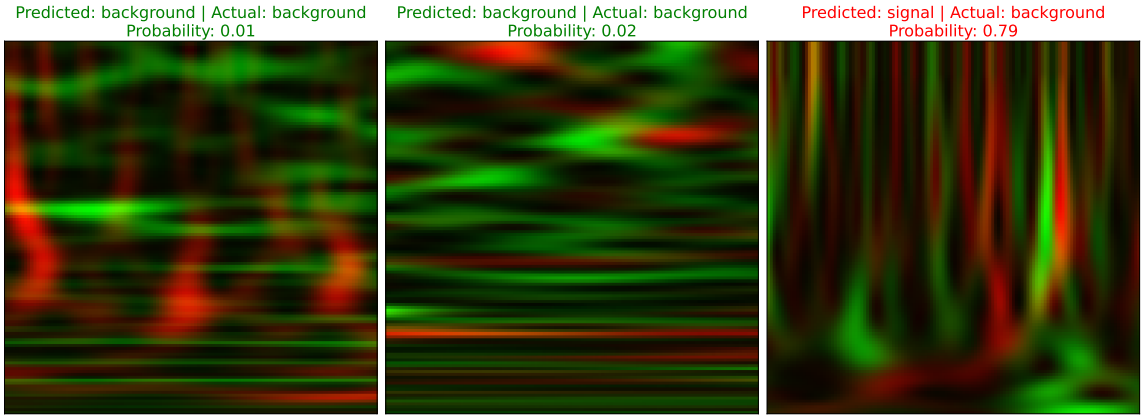

Figure 6.2: Noise images with the model's prediction.

6.3 Precision and Recall Analysis

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p).

$$precision = \frac{T_p}{T_p + F_p}$$

Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n).

$$recall = \frac{T_p}{T_p + F_n}$$

These quantities are also related to the (F_1) score, which is defined as the harmonic mean of precision and recall.

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly. [32]

Precision, recall, and F_1 are bound in the range [0,1], indicating that values closer to one correspond to superior model performance. As F_1 relates both precision and recall, a number nearing one means the model is more accurate, since it implies precision and recall are also close to one. Generally, the most common metrics to assess model quality are

accuracy and loss, but they may not be entirely representative of unbalanced datasets. In such cases, precision and recall represent better the neural network's actual performance.

In the context of our study, a high-recall, low-precision system would detect a large percentage of signals but would also mislabel many noise images as signals, thus causing many false positives. This model would need a lot of double-checking to ensure every image labeled as a signal is indeed a signal and not an erroneous prediction.

On the other hand, a high-precision, low-recall system would mean that a high percentage of images predicted as a signal are, in fact, signals, but would also misidentify many signal images as noise. This would also force double-checking since many signal images would be labeled as noise, although the chances of a correctly predicted signal are higher than in the former case.

We will compare the following models based on the metrics in class 1, as our primary interest lies in signal detection rather than noise detection. Nonetheless, in cases where models are very similar, we will also take noise prediction metrics into account.

The first model, the best-performing small_custom_net, exemplifies a high-precision, low-recall system compared to the second model. This model is expected to have fewer errors when predicting signal images, therefore requiring less double-checking for mislabeled images. However, it is also likely to recover a lower percentage of the total signal images, leaving out a more significant proportion than the alternative model.

Class	Precision	Recall	f1-score	support
0	0.97	0.98	0.97	3648
1	0.34	0.28	0.31	144

Table 6.6: Classification report of the best small_custom_net obtained. Data_Aug=0.1, Dropout=0.25, LR=1e-5.

In contrast, the second model, the top-performing Pretrained_ResNet34, represents a high-recall, low-precision system compared to the alternative. This model is expected to retrieve a more significant proportion of total signal images, but mislabel more noise images as signal, leading to an increase in false positives. As a result, this model would need double-checking images marked as signals due to its lower precision.

Class	Precision	Recall	f1-score	support
0	0.99	0.60	0.74	3648
1	0.07	0.78	0.13	144

Table 6.7: Classification report of the best Pretrained_ResNet34 obtained. Size=64x64, Data_Aug=1.1, LR=1e-4, Weight_Decay=0.

Presented the difference in characteristics between these two models, the election could be done based on the F_1 score, which allows a better comparison when these metrics are disparate. Based on this metric, we would select the small_custom_net model since it better balances precision and recall.

CHAPTER 7
Conclusions

7.1 Conclusions

This work focused on building an image classifier to distinguish between signal injected in noise spectrograms and simple noise. To achieve this, we needed a large enough dataset to train our deep-learning model to accomplish this. We chose to generate it ourselves since, in this way, we could control every aspect, including the number of samples, which was very important for the training phase to obtain a diverse enough set. This constituted the training set. On the other hand, a real signal-based image dataset was also needed to test our model correctly. This testing dataset consisted of images obtained from actual measured signals extracted using the fetch_open_data function in the GWPY library. After obtaining the necessary data, we built the required pipeline to train the models. For this purpose, we leveraged cutting-edge technologies such as PyTorch, a wide-used technology in the deep-learning ecosystem. Utilizing this foundation, we experimented with various models, ranging from ResNets and Vision Transformers to a custom-designed neural network we developed from scratch. Each of these architectures achieved an accuracy of over 90%, without too many differences in discrimination power.

Concerning the goals we set ourselves in Section 1.2, we successfully met one: developing a deep-learning model that can accurately distinguish spectrometers containing pure noise from those containing noise with an injected signal. It can be demonstrated in Chapter 6, where we comment on our results, that we have been successful in this objective.

About our second goal, which was to enhance the performance of existing classifiers designed for the detection of gravitational waves, we did not fully meet the desired outcome, although we achieved state-of-the-art performance. We believe this objective was highly ambitious, even though we may achieve a better result with a different approach or a deeper understanding of the physics related to this phenomenon. We comment more on this in the future work section.

7.2 Future work

In our opinion, to boost significantly the accuracy obtained, we must take a different approach. To achieve this, and based on some papers mentioned in 2, we think it may be further improved on along these lines:

- To use alternative input formats for gravitational wave signals. The developed system uses images to predict whether or not the signal contains a gravitational wave.

It could be interesting to try to change the model's architecture to use as input the raw signal. In this way, the model may grasp slighter differences in the signal, thus obtaining higher accuracy and lower loss.

- To properly leverage the Vision Transformer architecture by generating more synthetic samples. As commented in Section 4.2.2, ViT benefits from large amounts of data. Our dataset used for train and validation has only ten thousand images. At the same time, comments made by researchers participating in the design of this architecture suggest a boost in performance around the millions and tens of millions of samples.

Bibliography

[1] Wikipedia. Gravitational wave. https://en.wikipedia.org/wiki/Gravitational_wave, 2023. [Online; accessed 06/02/2023].

[2] LIGO. What are gravitational waves? https://www.ligo.caltech.edu/page/what-are-gw, 2023. [Online; accessed 06/02/2023].

[3] Alexander H. Nitz, Sumit Kumar, Yi-Fan Wang, Shilpa Kastha, Shichao Wu, Marlin Schäfer, Rahul Dhurkunde, and Collin D. Capano. 4-ogc: Catalog of gravitational waves from compact-binary mergers, 2022.

[4] Samantha A Usman, Alexander H Nitz, Ian W Harry, Christopher M Biwer, Duncan A Brown, Miriam Cabero, Collin D Capano, Tito Dal Canton, Thomas Dent, Stephen Fairhurst, Marcel S Kehl, Drew Keppel, Badri Krishnan, Amber Lenon, Andrew Lundgren, Alex B Nielsen, Larne P Pekowsky, Harald P Pfeiffer, Peter R Saulson, Matthew West, and Joshua L Willis. The PyCBC search for gravitational waves from compact binary coalescence. *Classical and Quantum Gravity*, 33(21):215004, oct 2016.

[5] 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. *Phys. Rev. D*, 85:122006, Jun 2012.

[6] He Wang, Shichao Wu, Zhoujian Cao, Xiaolin Liu, and Jian-Yang Zhu. Gravitational-wave signal recognition of LIGO data by deep learning. *Physical Review D*, 101(10), may 2020.

[7] Chayan Chatterjee, Linqing Wen, Foivos Diakogiannis, and Kevin Vinsen. Extraction of binary black hole gravitational wave signals from detector data using deep learning. *Physical Review D*, 104(6), sep 2021.

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

[9] Hongyu Shen, Daniel George, Eliu. A. Huerta, and Zhizhen Zhao. Denoising gravitational waves with enhanced deep recurrent denoising auto-encoders. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, may 2019.

[10] 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.

[11] Maximilian Dax, Stephen R. Green, Jonathan Gair, Jakob H. Macke, Alessandra Buonanno, and Bernhard Schölkopf. Real-time gravitational wave science with neural posterior estimation. *Phys. Rev. Lett.*, 127:241103, Dec 2021.

[12] João D Álvares, José A Font, Felipe F Freitas, Osvaldo G Freitas, António P Morais, Solange Nunes, Antonio Onofre, and Alejandro Torres-Forné. Exploring gravitational-wave detection and parameter inference using deep learning methods. *Classical and Quantum Gravity*, 38(15):155010, jul 2021.

[13] Rich Ormiston, Tri Nguyen, Michael Coughlin, Rana X. Adhikari, and Erik Katsavounidis. Noise reduction in gravitational-wave data via deep learning. *Physical Review Research*, 2(3), jul 2020.

[14] Kentaro Mogushi. Reduction of transient noise artifacts in gravitational-wave data using deep learning, 2021.

[15] Hunter Gabbard, Michael Williams, Fergus Hayes, and Chris Messenger. Matching matched filtering with deep networks for gravitational-wave astronomy. *Physical Review Letters*, 120(14), apr 2018.

[16] 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), sep 2019.

[17] Paraskevi Nousi, Alexandra E. Koloniari, Nikolaos Passalis, Panagiotis Iosif, Nikolaos Stergioulas, and Anastasios Tefas. Deep residual networks for gravitational wave detection, 2022.

[18] Wei Wei, Asad Khan, E.A. Huerta, Xiaobo Huang, and Minyang Tian. Deep learning ensemble for real-time gravitational wave detection of spinning binary black hole mergers. *Physics Letters B*, 812:136029, jan 2021.

[19] Eric A. Moreno, Jean-Roch Vlimant, Maria Spiropulu, Bartlomiej Borzyszkowski, and Maurizio Pierini. Source-agnostic gravitational-wave detection with recurrent autoencoders, 2021.

[20] Tianyu Zhao, Ruoxi Lyu, Zhixiang Ren, He Wang, and Zhoujian Cao. Space-based gravitational wave signal detection and extraction with deep neural network, 2022.

[21] GWOSC. Gravitational wave open science center. https://www.gw-openscience.org/about/, 2023. [Online; accessed 16/02/2023].

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[23] datagen. Resnet: The basics and 3 resnet extensions. https://datagen.tech/guides/computer-vision/resnet/, 2023. [Online; accessed 25/05/2023].

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[25] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[26] Jason Brownlee. A gentle introduction to pooling layers for convolutional neural networks. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/, 2023. [Online; accessed 04/06/2023].

[27] Jason Brownlee. A gentle introduction to the rectified linear unit (relu). https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, 2023. [Online; accessed 05/06/2023].

[28] kennethleungty. Neural-network-architecture-diagrams. https://github.com/kennethleungty/Neural-Network-Architecture-Diagrams/blob/main/vgg16_xml.drawio, 2023. [Online; accessed 27/06/2023].

[29] PyTorch. Bcewithlogitsloss. https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html, 2023. [Online; accessed 10/06/2023].

[30] Abid Ali Awan. A complete guide to data augmentation. https://www.datacamp.com/tutorial/complete-guide-data-augmentation, 2023. [Online; accessed 10/06/2023].

[31] Vasilis Vryniotis. How to train state-of-the-art models using torchvision's latest primitives. https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/#break-down-of-key-accuracy-improvements, 2023. [Online; accessed 10/06/2023].

[32] Scikit-learn. Precision-recall. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html, 2023. [Online; accessed 27/06/2023].

APPENDIX A
Sustainable Development Goals

Degree of relevance of the work to the Sustainable Development Goals (SDGs).

Sustainable Development Goals	High	Medium	Low	Not applicable
SDG 1. **No poverty.**				X
SDG 2. **Zero hunger.**				X
SDG 3. **Good health and well-being.**				X
SDG 4. **Quality education.**		X		
SDG 5. **Gender equality.**				X
SDG 6. **Clean water and sanitation.**				X
SDG 7. **Affordable and clean energy.**				X
SDG 8. **Decent work and economic growth.**				X
SDG 9. **Industry, innovation and infrastructure.**	X			
SDG 10. **Reduced inequalities.**				X
SDG 11. **Sustainable cities and communities.**				X
SDG 12. **Responsible consumption and production.**				X
SDG 13. **Climate action.**		X		
SDG 14. **Life below water.**				X
SDG 15. **Life on land.**				X
SDG 16. **Peace, justice and strong institutions.**				X
SDG 17. **Partnership for the goals.**	X			

Reflexion on the relation of the TFG/TFM with the SDGs and with the most related SDG(s).

- **SDG 4: Quality Education**
  Our research contributes to advancing knowledge in theoretical physics and deep learning. By sharing our findings and methods, we help improve the quality of education in these areas, fostering innovation and promoting lifelong learning opportunities for all. Also, we help teachers by providing insights into cutting-edge techniques, which could help enhance students' learning experience.

- **SDG 9: Industry, Innovation, and Infrastructure**
  Our work on designing and training neural networks for gravitational wave detection is an example of innovation in the field of physics and deep learning. Although we share the objective with other researchers, we developed a slightly different approach to detecting gravitational waves, thus contributing to innovation in this field.

- **SDG 13: Climate Action**
  Although our research is not directly related to climate change, applying deep learning techniques to sciences could lead to developing complex climate systems and effective climate action strategies. By exploring the potential of deep learning in various scientific domains, we can collectively work towards mitigating the impacts of climate change and achieving a sustainable future. Moreover, along history, science, and engineering have played crucial roles in advancing various processes, making them more efficient and sustainable. Through continuous research and innovation, scientists and engineers have developed new technologies, improved methodologies, and optimized systems to achieve greater efficiency and effectiveness in various fields.

- **SDG 17: Partnerships for the Goals**
  Science builds on the discoveries of other scientists, thus making collaboration intrinsic to the scientific community. For an increasingly better society, scientific progress is necessary, and for this, collaboration between all agents such as the government, companies, and academia is crucial. Without the collaboration of these entities over time, we would not have achieved discoveries as important as the gravitational waves predicted by Einstein's theory of relativity. This theory, in turn, is built upon prior findings, remarking the importance of collaboration, because without it, we could not have carried out our work.