universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Advanced Machine Learning Techniques for Gravitational Wave Detection: Overcoming High-Noise Challenges via Convolutional Autoencoders

verfasst von | submitted by

Kristyna Vitulova

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 861
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Astronomy
Betreut von | Supervisor:	Assoz. Prof. Mag. Dr. Josef Pradler

Acknowledgements

I would like to sincerely thank everyone who supported me throughout the process of not only writing this thesis but also conducting the research. My deepest gratitude goes to the Gravitational Waves research group at the Austrian Academy of Sciences, as well as to the kind professors at the University of Vienna. I would also like to thank the Einstein Telescope Collaboration for providing all the necessary data for my research. Above all, I am immensely grateful to my wonderful family, who once again supported me every step of the way.

Special thanks belong to my workplace, Inprasie Systems, for providing me with invaluable experience in rocket engineering projects while at the same time allowing me to fully dedicate myself to both this thesis and my studies abroad. For this, I am truly and forever thankful.

Abstract

Gravitational wave astronomy is a powerful tool for probing extreme astrophysical phenomena, yet current detection pipelines — relying on matched filtering — face limitations due to their dependence on predefined waveform templates. This can lead to up to 10% of detected events being missed and to potential mismatches in assigning incorrect gravitational waveforms to detected signals. The matched filtering method is also computationally demanding.

These challenges are expected to intensify with the upcoming third generation of gravitational wave observatories, such as the Einstein Telescope (ET), which is anticipated to operate with unprecedented low-frequency sensitivity but also increased susceptibility to complex noise artefacts.

This thesis focuses on the possibility of using convolutional autoencoders (CAEs) for the unsupervised detection of gravitational wave signals in high-noise environments, using realistic mock data from the ET collaboration. The proposed methodology is to train CAEs exclusively on noise-only spectrograms, allowing the model to learn the pattern of the detector's background and to flag any deviations via reconstruction error. In addition to fully unsupervised training, a weak supervision stage is introduced, in which a limited set of low-SNR injected signals guides hyperparameter optimization towards higher sensitivity to subtle anomalies.

The experiments presented in this thesis are conducted in two stages: (i) training and optimization on pure ET noise, and (ii) evaluation during parametrization with injected binary black hole waveforms of varying masses and luminosity distances. Hyperparameters are tuned using the Optuna framework, combining reconstruction accuracy on noise-only data with the maximization of the *gap* metric — enlarging the relative separation between noise and anomaly reconstruction errors.

The results of this research show that while unsupervised CAE achieves robust noise reconstruction, introducing weak supervision significantly improves the separability between data containing noise and anomalies, achieving up to a 100% detection rate and a 0% false alarm rate for high-SNR signals.

These findings present that convolutional autoencoders, especially when paired with weak supervision approach, can provide an effective and efficient detection method, offering adaptability to non-predefined waveforms and rare sources. This work also highlights the possibility of deep learning-based methods for anomaly detection in the data-rich, noise-dominated environments of next-generation detectors.

Kurzfassung

Die Gravitationswellenastronomie ist ein leistungsstarkes Instrument zur Erforschung extremer astrophysikalischer Phänomene. Dennoch stoßen die derzeitigen Detektionspipelines — die auf dem *Matched Filtering* beruhen — an ihre Grenzen, da sie auf vordefinierte Wellenform-Templates angewiesen sind. Dies kann dazu führen, dass bis zu 10% der Ereignisse nicht erkannt werden und potenziell falsche Gravitationswellenformen einem detektierten Signal zugeordnet werden. Darüber hinaus ist die Methode des *Matched Filtering* rechnerisch sehr aufwendig.

Diese Herausforderungen werden sich mit der bevorstehenden dritten Generation von Gravitationswellenobservatorien, wie dem Einstein-Teleskop (ET), voraussichtlich verschärfen. Das ET wird eine beispiellose Empfindlichkeit im Niederfrequenzbereich erreichen, ist jedoch auch anfälliger für komplexe Rauschartefakte.

Diese Arbeit befasst sich mit der Möglichkeit, Convolutional Autoencoders (CAEs) zur unüberwachten Detektion von Gravitationswellensignalen in stark verrauschten Umgebungen einzusetzen. Hierbei werden realistische Simulationsdaten der ET-Kollaboration verwendet. Die vorgeschlagene Methodik besteht darin, CAEs ausschließlich mit Rausch-Spektrogrammen zu trainieren, sodass das Modell das Muster des Detektorhintergrunds erlernt und Abweichungen über den Rekonstruktionsfehler erkennt. Zusätzlich zum vollständig unüberwachten Training wird eine Phase der schwachen Überwachung (*Weak Supervision*) eingeführt, in der ein begrenzter Satz von injizierten Signalen mit niedrigem Signal-Rausch-Verhältnis die Hyperparameter-Optimierung auf eine höhere Empfindlichkeit für subtile Anomalien ausrichtet.

Die in dieser Arbeit beschriebenen Experimente werden in zwei Phasen durchgeführt: (i) Training und Optimierung mit reinem ET-Rauschen und (ii) Auswertung während der Parametrisierung mit injizierten binären Schwarzen-Loch-Wellenformen unterschiedlicher Massen und Leuchtweitendistanzen. Die Hyperparameter werden mit dem Optuna-Framework optimiert, wobei die Rekonstruktionsgenauigkeit für reine Rauschdaten mit der Maximierung der *Gap*-Metrik kombiniert wird, um die relative Trennung zwischen Rausch- und Anomalie-Rekonstruktionsfehlern zu vergrößern.

Die Ergebnisse dieser Untersuchung zeigen, dass der unüberwachte CAE eine robuste Rekonstruktion des Rauschens erreicht, während die Einführung der schwachen Überwachung die Trennschärfe zwischen Rausch- und Anomaliedaten erheblich verbessert. Für Signale mit hohem Signal-Rausch-Verhältnis wird eine Erkennungsrate von bis zu 100% bei einer Falschalarmrate von 0% erzielt.

Diese Ergebnisse verdeutlichen, dass Convolutional Autoencoders, insbesondere in Kombination mit schwacher Überwachung, eine effektive und effiziente Detektionsmethode darstellen können, die Anpassungsfähigkeit an nicht vordefinierte Wellenformen und seltene Quellen bietet. Darüber hinaus hebt diese Arbeit das Potenzial tiefenlernender Methoden

Kurzfassung

zur Anomalieerkennung in datenreichen, rauschdominierten Umgebungen der nächsten Detektorgeneration hervor.

Contents

List of Figures

List of Tables

1. Introduction

Gravitational waves, first predicted by Albert Einstein in 1916 as a consequence of his General Theory of Relativity, were directly observed nearly a century later, on September 14, 2015. These waves represent perturbations in the curvature of spacetime that propagate at the speed of light, carrying information about some of the most energetic and distant astrophysical events in the universe. Currently, the standard method for detecting gravitational waves is matched filtering. This technique compares incoming data from detectors with a large bank of precomputed waveform templates, each corresponding to a specific source configuration. While highly effective for modeled sources, this method is computationally expensive and inherently limited—it can fail when the true signal deviates from the assumptions encoded in the template bank. It is estimated that up to 10 percent of gravitational-wave events may go undetected using this approach.

The next generation of gravitational wave observatories, such as the Einstein Telescope (ET), will improve detection sensitivity to higher levels. The ET is expected to measure signals at lower frequencies and from a wider range of sources, potentially even hours before the merger. However, this improvement in sensitivity also introduces significant challenges: not only larger volumes of measured data, but also an increased presence of complex noise artifacts, especially in the low-frequency regime.

To address these limitations, alternative methods are being explored. One of these approaches is the application of deep learning techniques — in case of this work convolutional autoencoders. These models are able to learn the underlying structure of the noise in an unsupervised training and identifying outliers by measuring reconstruction error. Since the model is trained exclusively on noise-only data, gravitational-wave signals — unseen during training — manifest as poorly reconstructed inputs, producing significantly higher reconstruction errors.

The goal of this thesis is to enhance gravitational wave detection using deep convolutional autoencoders. We investigate the use of unsupervised anomaly detection techniques on data derived from the Einstein Telescope Mock Data Challenge. In addition, we propose a weakly supervised optimization technique, when a small sets of injected signals guide the model to become more sensitive to subtle deviations from the noise baseline.

The structure of this thesis is organized as follows:

- **Chapter 2 – Theoretical Background:** The introduction of the fundamental concepts of gravitational wave physics, including sources, waveform morphologies, and current detection methods. It also provides an overview of machine learning techniques relevant to this work, with emphasis on convolutional autoencoders and anomaly detection.

- **Chapter 3 – Methodology:** Description of the experimental design, including the generation and preprocessing of white noise and Einstein Telescope mock data, the injection of synthetic gravitational waveforms, and the architecture of the convolutional autoencoder. It also introduces the optimization process and the introduction of weak supervision.
- **Chapter 4 – Results:** Presents the outcomes of the training process and evaluates the performance of the model, both in the fully unsupervised setting and after applying the weak supervision strategy.
- **Chapter 5 – Discussion:** Interprets the experimental results, highlights the strengths and limitations of the proposed approach, and discusses its broader implications.
- **Chapter 6 – Conclusion and Outlook:** Summarizes the key findings of the thesis and suggests potential directions for future research and improvements in gravitational wave detection methods.

2. Theoretical Background

2.1. Gravitational Wave Theory

Gravitational waves, first predicted by Albert Einstein in 1916 as a consequence of the General Theory of Relativity, are periodic distortions of spacetime that, like sound or electromagnetic waves, propagate outward from their source.

In the framework of general relativity, their propagation through a curved background spacetime can be described in the Lorenz gauge, which simplifies the Einstein field equations into a wave-like form.

Because of the conservation of mass-energy and momentum, gravitational waves cannot exhibit monopole or dipole radiation; instead, only variations in the quadrupole (or higher) mass-energy distribution can produce them. This requirement implies that the source must lack both point symmetry and perfect axial symmetry.

One of the simplest and most astrophysically relevant configurations satisfying these conditions is a compact binary system — for example, two neutron stars in mutual orbit. As they revolve around each other, bound within the curvature of spacetime, they are loosing orbital energy through gravitational-wave emission. This gradual energy loss causes the stars to spiral inward, while increasing their orbital velocity until they eventually merge. The wave amplitude decreases with distance from the source, following the inverse-square law [7].

2.1.1. Sources of Gravitational Waves

Simply said, gravitational waves are emitted whenever massive bodies undergo accelerated, non-spherically symmetric motion.

The sources of gravitational waves can be broadly categorized into four main classes:

- **Compact Binary Coalescences (CBCs)**
- **Isolated non-axisymmetric objects**
- **Stochastic Backgrounds**
- **Burst-like events (e.g. Supernovae)**

The detectability and morphology of these signals vary widely across the gravitational wave spectrum.

Compact Binary Coalescences (CBCs) are the most studied, understood and observationally confirmed sources of gravitational waves. These systems include several

types of mergers, such as binary neutron stars (BNS), binary black holes (BBH), and neutron star-black hole (NSBH) pairs. The signal emitted by such systems typically spans three phases: *inspiral*, *merger*, and *ringdown*.

The frequency evolution during inspiral is governed by the chirp mass $\mathcal{M}$, defined as:

$$\mathcal{M} = \frac{(m_1 m_2)^{3/5}}{(m_1 + m_2)^{1/5}} \tag{2.1}$$

where m_1 and m_2 are the component masses.

The gravitational-wave frequency increases with time. This produces a characteristic *chirp signal*. The rate of frequency change is given by the equation:

$$\frac{df}{dt} = \frac{96}{5}\pi^{8/3}\left(\frac{G\mathcal{M}}{c^3}\right)^{5/3} f^{11/3}. \tag{2.2}$$

The ranges of signals binary neutron stars systems generate are from tens to thousands of Hz and often last up to hundreds of seconds within the sensitivity band of the current generation of ground-based detectors such as aLIGO and VIRGO.

On the other hand, binary black hole mergers involving stellar-mass black holes are much shorter in duration and emit at lower frequencies.

Future detection of supermassive black hole binaries ($M \sim 10^6 - 10^9 M_\odot$) is mainly targeted by the space-based LISA observatory, as the signal emitted by these is in the millihertz regime.

Another type of signal is emitted by rapidly rotating **neutron stars (NSs** with asymmetric mass distribution - e.g. deviation from rounded shape in a form of "mountains" on their surfaces or magnetic distortion. Signals from such sources are expected to be nearly monochromatic and extremly stable phase-wise. Typical strain amplitudes are in order of $h \sim 10^{-26}$ to $h \sim 10^{-24}$. Despite the fact that such a source has not yet been detected and confirmed, targeted searches using known pulsars and all-sky searches for unknown neutron stars continue to place upper bounds on gravitational wave emission.

For a rotating triaxial neutron star, the expected strain amplitude is as follows:

$$h_0 = \frac{4\pi^2 G}{c^4}\frac{I_{zz}\epsilon f^2}{r}, \tag{2.3}$$

where I_{zz} is the moment of inertia, ϵ is the ellipticity, f is the rotational frequency, and r is the distance to the source.

Asymmetric **Core-collapse Supernovae** are also potential sources of burst-like gravitational wave emission. The morphology and amplitude of such signal is highly uncertain as a consequence of the complex and stochastic nature of the collapse. According to simulations performed, the gravitational wave signal may include several components: prompt convection, standing accretion shock instability (SASI), and rotational core bounce.

Characteristic frequencies are typically in the range of 100 to 1000 Hz and the duration of the signal is of 10 to 100 ms. Even though these events are rare, as they occur only a few times per century in Milky Way region or nearby galaxies (LMC, SMC) they are high-priority targets for multi-messenger astrophysics.

The last source of gravitational waves to be mentioned is **Stochastic Backgrounds**. In case of those, gravitational waves arise from the superposition of numerous unresolved sources or from early-universe processes. Possible cosmological origins include:

- Inflationary models and quantum fluctuations,
- first-order phase transitions
- cosmic strings or other topological defects.

The stochastic background has spectral density characterized by the dimensionless quantity:

$$\Omega_{\mathrm{GW}}(f) = \frac{1}{\rho_c}\frac{d\rho_{\mathrm{GW}}}{d\ln f}, \tag{2.4}$$

where ρ_c is the critical density of the universe and ρ_{GW} is the gravitational-wave energy density per logarithmic frequency interval. Ground-based detectors constrain this quantity to $\Omega_{\mathrm{GW}}(f) < 10^{-9}$ in the tens to hundreds of Hz range [17].

2.1.2. Morphologies of waveforms

To gain insight into the physical properties of gravitational wave signals and their sources we need to understand their morphology. These variations of waveforms depend on the type of event, the mass configuration of the system, spin, orbital parameters and also the internal structure of the involved objects. For investigating such a signal, time-frequency analysis techniques are essential tools. These reveal hidden structures that are not visible in the time or frequency domain alone.

Gravitational waves from the compact binary systems exhibit a characteristic three-phase structure:

- **Inspiral**: The two bodies are orbiting each other while gradually losing energy to gravitational radiation. The increase in frequency and amplitude is quasi-monotonic and the waveform is commonly referred to as a *chirp*. This phase is very well described by post-Newtonian approximations.
- **Merger**: The final stage of orbital decay leads to the coalescence of the compact objects. This part of the evolution process is the most nonlinear and typically produces the peak amplitude of the waveform. Numerical relativity is required to model this regime.
- **Ringdown**: The remnant object (for example black hole) settles into a stable configuration and emits damped sinusoidal gravitational radiation.

The waveform of this evolution can be seen in 2.1

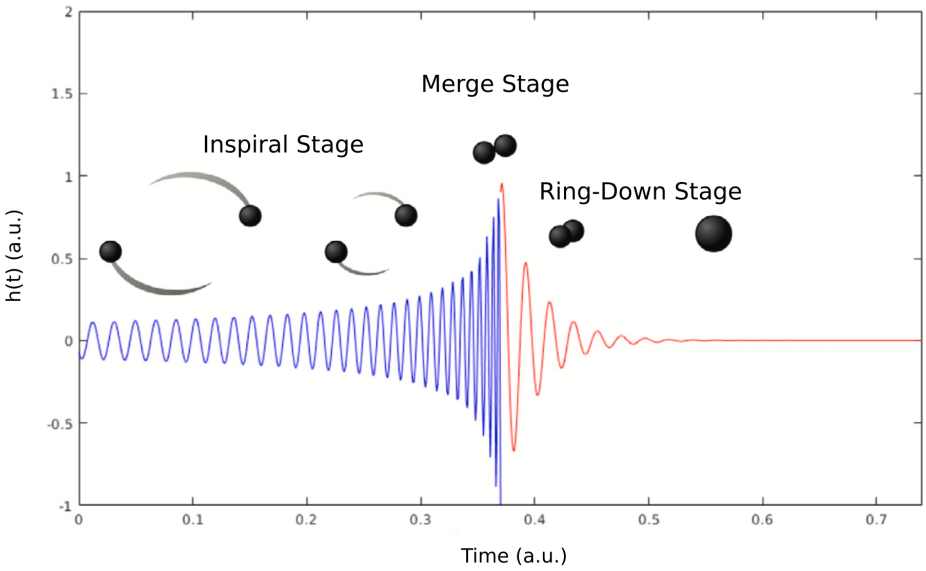

Figure 2.1.: Visualization of binary merger consisting of inspiral, merger and ring-down stage. Adapted from [12].

In time-frequency representation, they are clearly separable and are influenced by the total mass of the system and the mass ratio between the two bodies. Lower mass systems (such as binary neutron stars) are characterized by longer durations of the inspirals with domain power at lower frequencies.

Time-frequency representations, such as wavelet scalograms or spectrograms used in this work, reveal a rich variety of signal morphologies depending on the astrophysical origin:

- **Binary Black Hole Mergers**: Typically show a steep, rapidly rising chirp followed by a short ringdown. Systems with higher asymmetries or spin precession may display additional modulations or secondary peaks. However, for this work, only black holes without any spins or significant mass differences were used.
- **Binary Neutron Stars**: These systems usually produce longer inspiral phases and post-merger remnants can emit short-lived high-frequency oscillations (in range of several kHz). This can encode information about the neutron star's equation of state.
- **Eccentric Binaries**: These may exhibit repeated chirp-like bursts when going through each periastron. The resulting signal is made up of multiple chirps of varying duration and amplitude.
- **Core-collapse Supernovae**: These, along with other burst sources, tend to generate short-lived irregular broadband signals. Their morphology depends on the

internal dynamics of the collapse. It is well resolved by using high time-resolution transforms.

All these mentioned structures are influenced by waveform's *quality factor* $Q = \frac{f}{\Delta f}$, which explains the trade-off between time and frequency resolution. This means that higher Q values favor narrowband features as ringdown modes. Lower Q values, on the other hand, are better suited for capturing transient broadband phenomena.

Wavelet-based decompositions are allowing the extraction of the morphological features that can be hidden in the signal classification or detection algorithms. Discrete wavelet transforms (DWT) are a good tool to decompose the gravitational wave signal into the time-localized frequency bands, from these, the statistical features such as energy ratios, entropy and many more can be derived [10].

Another way to visualize data from interferometers is fixed-resolution time-frequency representations such as spectrograms, based on the short-time Fourier transform (STFT). While not offering the adaptive resolution of wavelets, spectrograms provide intuitive visualization of the spectral content over time and are widely used in gravitational wave data analysis for identifying noise transients, glitches, and signals with long duration [3].

These features serve as inputs to machine learning models for tasks like signal classification, parameter estimation and, in our case, anomaly detection. Specifically, they can improve the performance of unsupervised or weakly supervised pipelines by enhancing the separability between signal and noise distribution in latent or feature spaces [10].

2.2. Detection of Gravitational Waves

Gravitational waves are detected by observing the relative motion of freely falling objects, such as using interferometers or resonant mass detectors.

In our case, we only focus on laser interferometer detectors as a detection method used in detectors such as Advanced LIGO (aLIGO) and Advanced Virgo as well as in the future Einstein Telescope (ET), whose mock data will be used later on in this work. However, the final architecture of ET is not yet determined; therefore, for the scope of this work, we will describe the general idea of a laser interferometer used for already existing detectors.

Laser interferometer detectors are made to detect small differences in lengths of optical paths caused by the passing of gravitational waves. Such an interferometer emits light from a laser with coherent properties, and such light is then sent through a beam splitter and down to two orthogonal arms to distant mirrors and reflected to recombine and interfere. In a perfect scenario, once these two beams recombine, they vanish due to destructive interference, unless a gravitational wave passes through and causes a slight difference in the length of the arms. Simplified setup can be seen in 2.2 below.

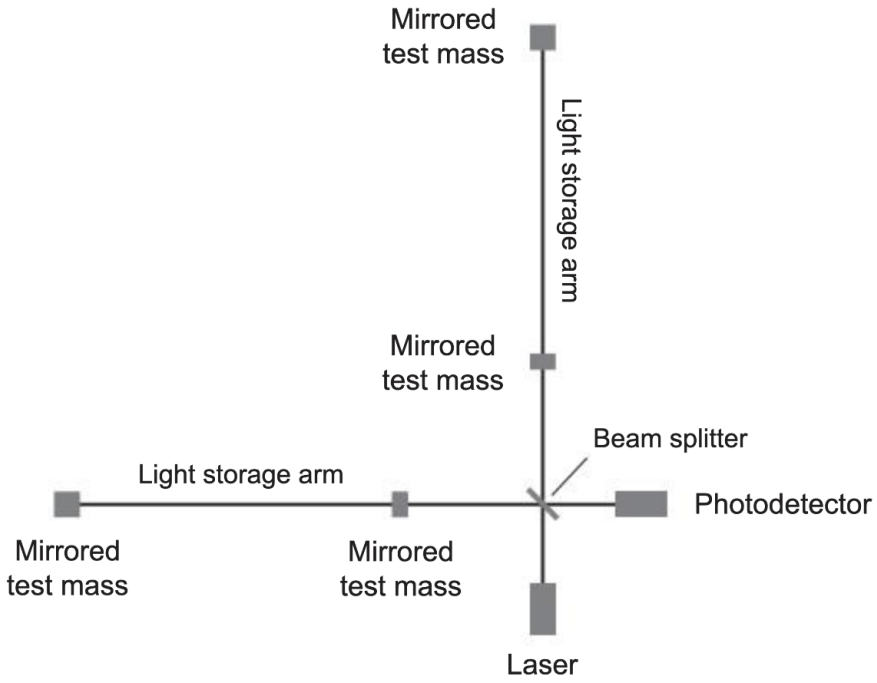

Figure 2.2.: A scheme of laser interferometer gravitational waves detector. Adapted from [8].

To effectively increase the optical path length—and thus enhance the sensitivity of the interferometer—each arm contains an optical cavity in which the laser light is reflected multiple times before being recombined at the beam splitter. This technique allows the light to traverse a much longer effective distance than the physical length of the arms would suggest, thereby amplifying the detector's response to passing gravitational waves.

2.2.1. Einstein Telescope

The Einstein Telescope (ET), a planned third-generation gravitational wave detector, plans to improve the sensitivity of current detectors like aLIGO, VIRGO, and KAGRA. The design, which was still under consideration as of July 2025, will likely include two separate but linked interferometric systems: one designed for low frequencies (ET-LF) and another for high frequencies (ET-HF). This setup, known as the xylophone configuration, is a key element in the next generation of detectors.

The ET-LF is expected to operate in the frequency range of about 2 to 30 Hz. It will use mirrors cooled to cryogenic temperatures, around 10–20 K, to reduce mechanical noise. Cooling the mirrors greatly cuts down on thermoelastic and Brownian noise, which are major issues at low frequencies. The laser system in ET-LF will use a relatively small

amount of power (about 3 W), but it will have very stable intensity and frequency to ensure accurate measurements.

On the other hand, the ET-HF will operate from around 30 Hz to several kilohertz. It will use a high-power laser (up to 500 W) and mirrors at room temperature, without the need for cryogenic cooling. This setup is specifically designed to detect gravitational waves from events like neutron star and black hole mergers.

Both interferometers will share the same tunnel for their optical arms, but they will work separately. By using this design, the ET avoids the trade-offs between high laser power and low thermal noise that are common in current single-channel detectors.

One major drawback of current detectors is their limited sensitivity to signals below 20 Hz, where seismic and Newtonian noise are dominant. The Einstein Telescope plans to extend the sensitivity down to about 2 Hz. To make this happen, the ET will be built about 200 meters underground. At this depth, seismic disturbances and Newtonian noise are greatly reduced. The underground location also ensures stable pressure and temperature while also reducing noise from human activities.

This enhanced sensitivity — particularly at low frequencies—will enable long-duration, in-band observations of binary systems, extending up to several hours before the merger of binary neutron stars. It will also improve the detection capabilities for more massive and distant sources, including intermediate-mass black holes (IMBHs) with masses ranging from 100 to 1,000 solar masses [5].

2.2.2. Noise Challenges

With the advent of next-generation gravitational wave detectors, such as the proposed Einstein Telescope, a significant improvement in instrumental sensitivity is anticipated. One of the key advancements lies in extending the lower frequency detection limit—potentially down to around 5 Hz. While this expansion of the observable frequency band opens new avenues for astrophysical discovery, it also introduces substantial challenges. Operating at such low frequencies requires overcoming a range of dominant noise sources, including seismic, Newtonian (gravity gradient), and thermal noise, which become increasingly significant in this regime [5].

As seen in 2.3, while the level of some type of noises is still uncertain, future operations are very likely to have to include a certain level of noise into their measurements and this level might be significantly higher than the one observed for aLIGO or Virgo detectors.

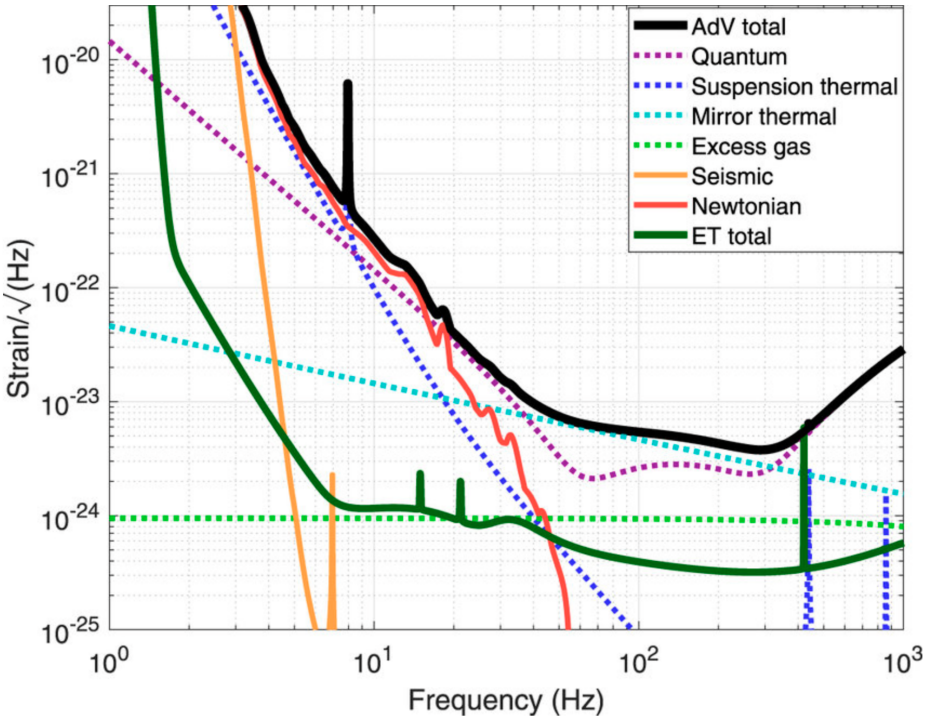

Figure 2.3.: Einstein Telescope noise budget. Adapted from [11].

2.3. Detection via Matched filtering

The current detection technique relies purely on *Matched Filtering*.

The signal from the detector is in practice translated in frequency domain and is composed of two parts: noise s and gravitational wave signal h.

$$\tilde{s}(f) = \tilde{n}(f) + \tilde{h}(f) \tag{2.5}$$

As in most cases of gravitational wave events, the noise is larger than the signal itself, we arrive at the classical problem in physics as well as in radio engineering – how to identify the signal of our interest, e.g., a gravitational wave.

The Matched filtering method is resolving such a problem with knowing the form of h(t) and have an approximate idea of the typical scales of variations of the noise.

The general idea then suggest to multiply the data s(f) by the template h(t), integrate over observation time T and average:

$$\frac{1}{T}\int_0^T s(t)h(t)\,dt = \frac{1}{T}\int_0^T h^2(t)\,dt + \frac{1}{T}\int_0^T n(t)h(t)\,dt \tag{2.6}$$

While the first term grows linearly with T and contributes to the actual signal, the second term behaves like a random walk and grows only as $T^{1/2}$. Therefore, for larger T, the signal dominates and can be detected even below the noise floor.

To formalize this idea, we introduce a linear filter $K(t)$ and define a filtered signal

$$\hat{s} = \int_{-\infty}^{\infty} s(t)K(t)\,dt. \tag{2.7}$$

The goal is to choose $K(t)$ in such a way that the signal-to-noise ratio (SNR) is maximized. In the frequency domain, this SNR can be written as

$$\frac{S}{N} = \frac{\int_{-\infty}^{\infty} \tilde{h}(f)\tilde{K}^*(f)\,df}{\left[\int_0^{\infty} S_n(f)|\tilde{K}(f)|^2\,df\right]^{1/2}}, \tag{2.8}$$

where $S_n(f)$ is the one-sided power spectral density (PSD) of the noise and tildes denote Fourier transforms.

The optimal filter that maximizes this expression is given by

$$\tilde{K}(f) \propto \frac{\tilde{h}(f)}{S_n(f)}, \tag{2.9}$$

which leads to the classical matched filter: the template is weighted by the inverse of the noise power at each frequency. This ensures that frequencies with lower noise contribute more strongly to the detection statistic.

Substituting this optimal filter into Equation (3) yields the optimal signal-to-noise ratio:

$$\left(\frac{S}{N}\right)^2 = 4\int_0^{\infty} \frac{|\tilde{h}(f)|^2}{S_n(f)}\,df. \tag{2.10}$$

This expression defines the theoretical limit of detectability for a given signal $h(t)$ in stationary Gaussian noise with known PSD $S_n(f)$ [14].

2.4. Machine Learning Techniques

What distinguishes machine learning from data analysis, as both processes the data, extract information, and make predictions, is that this process in machine learning techniques is automated. When computer programs are constructed with such abilities, the so-called training data are used and the computer program will become a compact representation of the main features captured in these data [13], [4].

2.4.1. Supervised vs. unsupervised learning

Machine learning techniques are commonly categorized into supervised and unsupervised learning. As the name of these branches suggests, for supervised learning the output - the so-called label - is already known. The model is being trained to predict those outputs for

new, unseen inputs. By using labeled inputs and outputs, model can measure its accuracy and learn over time. Such an approach uses two main types of problems, classification and regression.

Regression is modeling the relationship between two or more variables. When correlation of variables is present, the model can understand how one variable predicts the other and predict a real value output. The most common type of regression is linear regression, which aims to find a function that describes the relationship between x and y, and uses this function to predict values of y for given values of x. A visual example of such a relationship is shown in Figure 2.4. A linear regression model is a linear combination of coefficients and input variables. It can be expressed as:

$$y = a + Bx \tag{2.11}$$

which expands to:

$$y = a + b_0x_0 + b_1x_1 + \cdots + b_nx_n \tag{2.12}$$

where:

- a is the bias term,
- $B = [b_0, b_1, \ldots, b_n]$ is the vector of coefficients, and
- $x = [x_0, x_1, \ldots, x_n]^T$ is the input feature vector.

Alternatively, in matrix notation for multiple data points:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} \tag{2.13}$$

where:

- $\mathbf{y} \in \mathbb{R}^m$ is the vector of predicted values,
- $\mathbf{X} \in \mathbb{R}^{m\times(n+1)}$ is the design matrix (with a column of ones for the intercept),
- $\boldsymbol{\beta} \in \mathbb{R}^{n+1}$ is the parameter vector including the intercept.

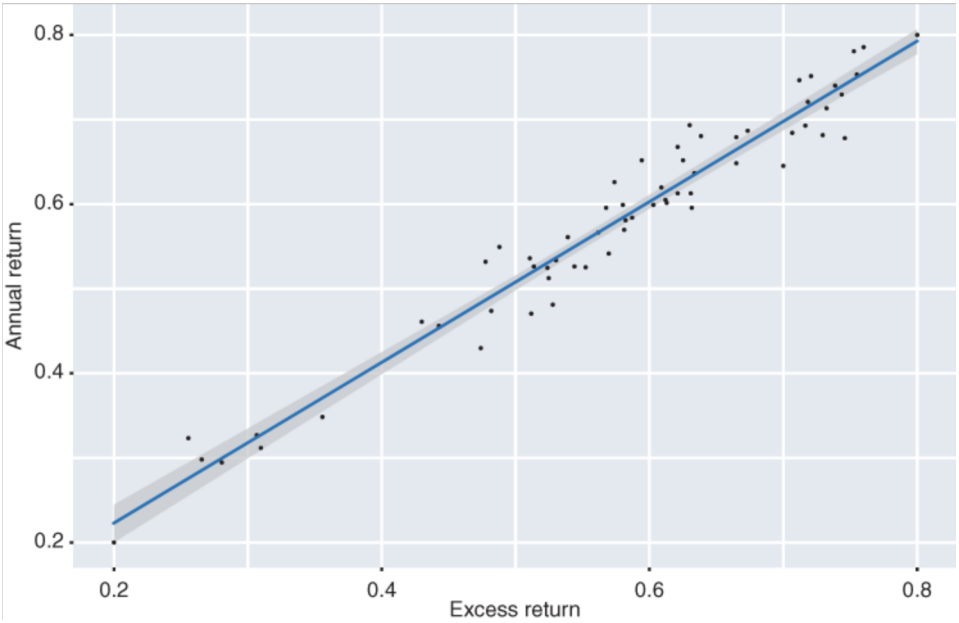

Figure 2.4.: An example from the stock portfolio dataset showing the relationship between annual return and excess return of stock using linear regression [18]

Examples of possible applications of linear regression include modeling the relationship between the floor area and the market price of a property to predict its value, or the relationship between study time and test results to estimate expected performance.

Classification, unlike regression, is the task of dividing data into distinct categories. The model assigns input data to one of several predefined classes based on a set of input features. Just as linear regression which is the simplest form of regression, the most basic form of classification is binary classification. In this case, the model produces a single output with two possible labels, commonly represented as 0 and 1. When the model is uncertain, the output may be a floating value between these two classes, reflecting the probability or confidence of classification. [6]

For unknown truth labels or correct values, unsupervised learning is used. With this technique we are able to learn the underlying structure in given data.

2.4.2. Introduction to Neural Networks

In this work, we are particularly interested in the subfield of machine learning called Deep Learning, neural networks with multiple layers of generalized linear regression models, inspired by neurons and their role in our brains.

Generally, neural networks are described by their type of activation function used to generate output as well as the architecture of the network.

2.4.3. Convolutional Neural Networks

A subclass of Deep Learning models are Convolutional Neural Networks (CNN), models specifically designed to work with data structured in regular grids - images, time series, or in our case spectrograms. CNNs take advantage of local spatial patterns and shared weights, which makes them efficient for recognizing patterns. Originally, they are designed by the visual cortex, where neurons respond to overlapping regions of input [9].

At the heart of a CNN is the convolution operation. In one dimension, the discrete convolution of an input signal x with a kernel k of size $2s + 1$ is defined as:

$$(x * k)(i) = \sum_{u=-s}^{s} k(u)\, x(i - u) \tag{2.14}$$

In the two-dimensional case—common in image and spectrogram analysis — the convolution becomes:

$$(x * k)(i, j) = \sum_{u=-s}^{s} \sum_{v=-s}^{s} k(u, v)\, x(i - u, j - v) \tag{2.15}$$

The kernel moves across the input and calculates weighted sums in small local regions. Settings like stride (how far it moves) and padding (how much border is added) affect how this works. After each convolution, a nonlinear function—usually ReLU—is used to add nonlinearity.

A standard CNN is build of repeated blocks of the following components:

- **Convolutional Layers**: Apply multiple learnable filters for extracting local features.
- **Activation Functions**: Introduce nonlinearity; ReLU is standard.
- **Batch Normalization**: Normalizes activations to stabilize and accelerate training.
- **Pooling Layers**: Reduce spatial resolution and computational load, often via max-pooling.

CCNs provide a hierarchical, learnable feature extraction mechanism of computing both low-level (e.g. edges and gradients) and high-level (e.g. shapes) patterns. They are efficient thanks to build-in features like local connections and shared weights and flexible enough to handle many types of data and tasks [9], [4].

2.4.4. Convolutional Autoencoders

One of the extensions of CNN is Convolutional Autoencoders (CAEs). They incorporate convolutional and transposed convolutional layers. While classical autoencoders compress and reconstruct input vectors through dense transformation, convolutional autoencoders preserve the spatial structure of input data and exploit the locality of its features.

Therefore, convolutional autoencoders are great tool for image-like inputs or time-frequency representations such as spectrograms.

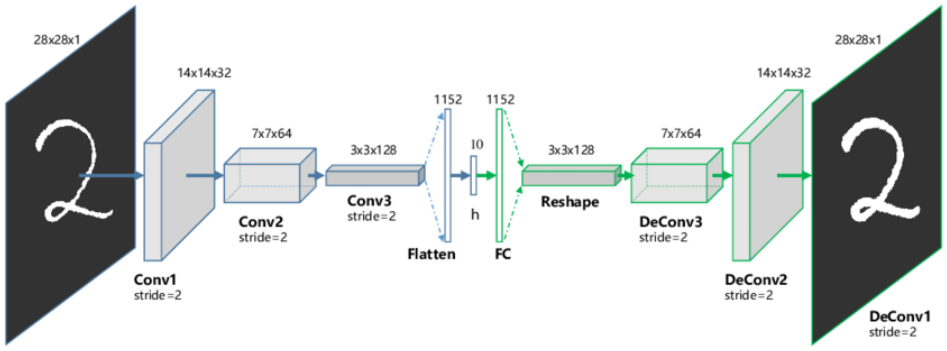

Figure 2.5.: Schematic representation of a convolutional autoencoder. The encoder compresses the input image into a low-dimensional latent space. Then the decoder reconstructs the image from the latent code. Adapted from [19]

Generally, the architecture of CAEs, as seen in 2.5 is made of two main components that mirror each other. The encoder, made of several convolutional layers, might be combined with nonlinear activations (such as ReLU) and downsampling operations such as max pooling or strided convolution. These layers reduce spatial resolution while increasing the number of feature maps. The result of this process is compact latent representation that captures the most important information of the input.

Then, from the compact latent representation, the decoder mirroring the encoder attempts to reconstruct the original input. Typically, the decoder part employs unsampling operations like transposed convolutions - also referred to as deconvolutions, to restore the original spatial dimension of the input. The model is trained so it minimizes a reconstruction loss - usually, as well as in our case - the mean squared error (MSE) between the original input x and the reconstructed output $\hat{x}$:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{2.16}$$

This approach enables the network to preserve key features while removing unnecessary noise or redundancies. Simply said, learning meaningful compressed representation in a fully unsupervised manner [19].

CEAs compared to fully connected autoencoders introduce several important advantages. Mainly, they allow to drastically reduce the number of trainable parameters through weight sharing, which is computationally more efficient as well as less prone to overfitting. Another important advantage is their use of convolutional filters. These make them naturally translation-equivariant, allowing them to recognize spatially shifted patterns. One last, but equally important advantage is preserving the 2D or 3D structure of the

input - this makes convolutional autoencoders well suited for high-dimensional structured data where local context is critical.

As to this date, CAEs have been applied in various domains, including image denoising, inpainting, super-resolution, and feature extraction. Specifically for science applications, they are increasingly used for tasks such as anomaly detection. Just like in our case, the model is trained solely on normal examples and signals that cannot be reconstructed well are flagged as outliers. This ability to learn task-independent features without requiring labeling the data makes them very useful for problems where labeled datasets are expensive to obtain or the potential anomalies are yet unknown.

Overall, the flexibility of CAEs architectures allows for many extensions. Different variants such as 1D, 2D or 3D CAEs adapt the convolutional operations to different data dimensionalities. Other variations add skip connections or regularization terms to improve performance. In more advanced approaches, convolutional variational autoencoders can use probabilistic latent space to better capture uncertainty in the data [19].

2.4.5. Anomaly detection

Patterns in data that do not align with expected behavior are anomalies. In gravitational wave physics, the anomalies in the telescope data can correspond to real astrophysical events - such as binary systems mergers or supernovae. These may be hidden within non-Gaussian and non-stationary detector noise.

Deep anomaly detection (DAD) methods are particularly well-suited to this settings as they have strong ability to learn rich, non-linear representation of high-dimensional data without requiring strong parametric assumptions [2].

There are several types of deep anomaly detection models, the four main categories are:

- **Reconstruction-based Methods**: Models such as convolutional or variational autoencoders are trained so that they reconstruct the non-anomalous data with the lowest reconstruction error possible. This method assumes that anomalies (data that has not been seen by the model during the training) will be hard to reconstruct.

- **Predictive Models**: Models train, as their name suggest, to predict future evolvement of data, or masked parts of the input itself. Anomalies are detected via prediction loss, as the future states of anomalous inputs will be harder to infer.

- **Probabilistic generative Models**: Models like GANs learn the probability distribution of the input space. Samples with low likelihood under the learned distribution are considered anomalous. These models are more expressive but very often hard to train and evaluate.

- **Hybrid Models**: These approaches decouple anomaly scoring from representation learning. A deep network is used to embed the data, and anomaly detection is performed in the latent space through the classical methods such as k-means, one-class SVM or isolation forests.

Convolutional autoencoders - method used for this work falls into the first category.

Very important aspect of anomaly detection is the structure of the latent space. Ideally, embeddings of non-anomalous data should cluster within a compact, smooth regions, where anomalies are expected to be outlying of these regions. But in classical autoencoders, the latent space is not explicitly regularized. That can lead to overlaps between anomalous and normal embeddings. Some types of convolutional autoencoders - such as VAE address this by enforcing a prior, which is usually Gaussian, over the latent distribution via Kullback-Leibler divergence penalty

$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\mathrm{KL}}(q(z|x)\|p(z)) \tag{2.17}$$

where $q(z|x)$ is the approximate posterior, $p(z)$ is the prior (often $\mathcal{N}(0, I)$), and $p(x|z)$ is the decoder.

However, in practice, the performance of variational convolutional autoencoders in anomaly detection ha been mixed - especially when reconstructive fidelity is critical.

Some approaches are not using reconstruction at all but formulate anomaly detection as a one-class classification problem. This way of anomaly detection is based on a neural network $f(x)$ being trained such, that all inputs are mapped close to a predefined center $c \in \mathbb{R}^d$ in latent space, while maximizing the distance of anomalous samples or pushing unknown samples towards the boundary. The loss is calculated as:

$$\mathcal{L}_{\mathrm{OC}} = \frac{1}{N}\sum_{i=1}^{N} \|f(x_i) - c\|^2 \tag{2.18}$$

Such models are typically sensitive to representation collapse, if the latent space is missing expressive constraints and require careful inicialization [2].

When speaking about usage of neural networks for gravitational wave detection, we must identify the challenges connected to such task. Main challenges - especially for future third-generation detectors like Einstein Telescope - are:

- **Extreme class imbalance**: True signals are rare compared to all present noise.
- **Signal diversity**: Source morphology varies widely (binary black hole mergers, binary neutron stars mergers, black hole and neutron star mergers etc.).
- **Noise non-stationary**: Background characteristics change over time and across frequencies.

Considering these given challenges, and more, unsupervised and weakly supervised reconstruction-based shall introduce a viable solution. By training on noise-only data, the convolutional autoencoder should learn a data manifold that represents normal operation of the detector. Deviation from such manifold - like gravitational wave signal - manifest as low latent likelihood or - as in case of this work - as high reconstruction error.

2.4.6. Training and Its parameters

To minimize the reconstruction loss between the input and the output when using convolutional autoencoder requires optimizing networks parameters. Such process is iterative and driven by gradient descent algorithms and backpropagation. Several hyperparameters and how the model learns from them are critically influencing the quality and efficiency of training. We will now introduce the key elements involved in the training process and review the influence of core hyperparametes.

- **Learning rate**: The first parameter introduced is the learning rate, which is one of the most sensitive and influential parameters in deep neural network training. This parameter is controlling the magnitude of updates to the model's weights at each of iterations during training. Setting the learning rate too high may result in failing to converge or even diverge. This could lead to oscillatory behavior or exploding loss. When learning rate is set too low the convergence process could be too slow with potential risk of getting stuck in poor local minima saddle points.

  Typical values of this parameter lie in ranges of 10^{-2} to 10^{-5}, depending on used optimizer and network architecture. For convolutional autoencoders, especially when using adaptive optimizers like Adam, just as in this work, initial rates of 10^{-3} to 10^{-4} are commonly used. Learning rate is commonly dominant factor determinating the final model performance. Thus, its proper selection is crucial for stable and efficient training.

- **Dropout**: Another important parameter is dropout visualized in below in figure 2.6 which is a type of regularization method which avoid overfitting by randomly deactivating a fraction of neurons during training. This helps the system to be immune to becoming overly reliant on specific paths through the model and encourages it to become more robust with the ability to generalize features.

  The effectiveness of dropout is more significant in fully connected layers, while for convolutional layers it is more nuanced as convolutional layers already benefit from parameter sharing and spatial locality, which inherently act as type of regularization. Therefor, dropout in convolutional layers is often redundant or even detrimental. However, applying dropout in latent or dense layers of convolutional autoencoders can be beneficial. Especially, when the model is prone to overfitting due to a high-capacity latent space, just as ours.

  Typical values of dropout lie between 0.2 and 0.5. If higher dropout is used, the generalization can be better but may also impair convergence if applied too aggressively.

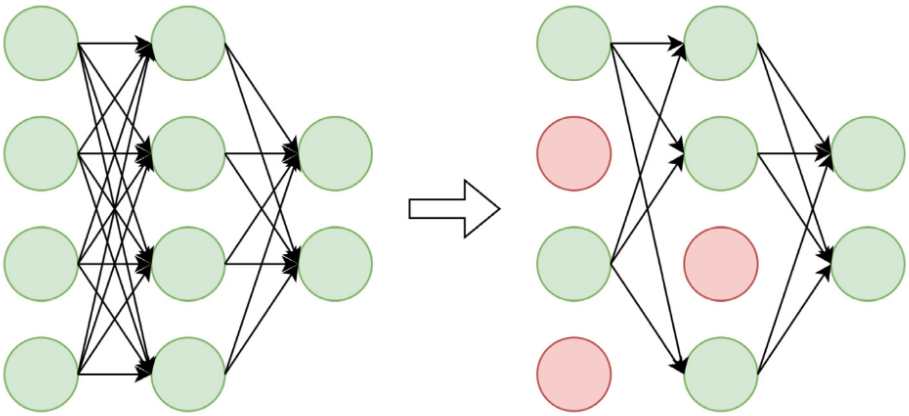

Figure 2.6.: Illustration of dropout regularization. During training, a subset of neurons is randomly deactivated (shown in red) to prevent co-adaptation and reduce overfitting. Adapted from [15].

- **Batch size**: determinate the number of training examples used to compute each gradient update. It has significant influence on the training dynamics as well as on the model's generalization capacity. If smaller batches are used, it introduces stochasticity into the gradient estimation. This can act as a form of regularization and might help escape sharp minima, but choosing larger batch size can provide more stable gradient estimates, which might accelerate convergence but can lead to worsening of generalization.

  The commonly used values for this parameter range from 32 to 256. The impact of batch size is task and dataset dependent, but moderate sizes (64 or 128) often provide a well trade-off between stability and generalization.

- **Weight decay**: (also known L2 Regularization) is a way of mitigating overfitting by penalizing large weight values. It modifies the loss function by adding a term proportional to the square of the model weights as

$$L_{\text{total}} = L_{\text{reconstruction}} + \lambda \sum_i w_i^2 \tag{2.19}$$

  where λ is weight decay coefficient and w_i donates the model weights. This approach forces the model to learn simpler, lower-magnitude weights that are in principal better in generalization.

  Moderate values of $\lambda \in [10^{-5}, 10^{-3}]$ are typically effective, but the optimal value depends on the model's size and richness of the dataset. Since the reconstruction task in convolutional autoencoders requires fine-grained detail preservation, overly strong regularization can degrade the quality of the output.

- **Number of epochs**: The number of training epochs defines how many complete passes is the model going to make through the entire training dataset. Increasing the number of epochs generally leads to an increase of the model capability of fitting the data, it also leads to larger risk of potential overfitting, especially when regularization techniques such as dropout or early stopping is missing in the architecture.

  As no universal number of epochs to reach perfect training exists, the training should be monitored by observing validation loss. The model should be then halted once the model starts to overfit - this manifests as lack of improvement in validation loss despite increasing the number of epochs.

- **Optimizers**: The optimizer determinates how gradients are used for updating the model parameters. The main known optimizers used for convolutional neural networks and convolutional autoencoders are Stochastic Gradient Descent (SGD) and Adam. The main differences between these two optimizers are, that SGD with momentum is capable of achieving competitive performance and is theoretically well understood, Adam is preferred as it has highly adaptive learning mechanism, which adjusts updates for each parameter individually based on the estimated first and second moments of the gradients.

  Adam is especially effective in deep convolutional autoencoders cause it accelerates convergence and reduces sensitivity to the choice of learning rate. The disadvantage of this optimizer compared to SGD is, that it may lead to solutions with poorer generalization, unless combined with suitable regularization techniques.

- **Learning Rate Scheduling**: Whenever dynamical adjusting of the learning rate during the training is needed, learning rate scheduling is employed. This may help the model to converge more quickly in the early stages of the training as well as to avoid overshooting or stagnation in later stages. Most known and used strategies are *step decay*, *exponential decay* and *cosine annealing*. Step decay reduces the learning rate by a fixed factor every few epochs. Exponential decay is continuously decreasing the learning rate according to an exponential schedule and finally, cosine annealing si decreasing the learning rate gradually, following a cosine function, often with restarts. This parameter is important and especially beneficial when using fixed learning rate optimizers like Stochastic Gradient Descent as well as for Adam-based training, particularly in avoiding early convergence to suboptimal minima [15].

2.4.7. Hyperparameter Tuning

All parameters mentioned above are limited by the proper set-up with each other, this is when hyperparameter tuning is stepping in.

Hyperparameter tuning serves as a tool to optimize machine learning models. As the complexity of deep learning architecture is increasing rapidly, so does the need for frameworks capable of an efficient and flexible way of searching large and possibly conditional hyperparameter spaces.

Hyperparameter optimization in this work was conducted using *Optuna*, an open-source framework that represents a new generation of optimization software that was designed to address limitations from the previous generation.

Three core design principles introduced by Optuna are:

- a define-by-run programming model for dynamic search space construction
- an efficient way of implementation of both sampling and pruning algorithms,
- and a lightweight and scalable system architecture suitable for both small and distributed experiments.

Normally, hyperparameter optimization libraries, such as *Hyperopt* or *SMAC* rely on the so-called *define-and-run* style. In practice, this means that the search space must be defined statically and completely prior to the beginning of any optimization. This becomes a major limitation for complex models with hierarchical and conditional parameters. Optuna, on the other hand, adopts the mentioned *define-and-run* approach. The search space is defined dynamically during the execution of the objective function by interaction with a trial object.

By using this dynamic construction, the use of standard Python control structures (e.g., loops and conditionals) is enabled. This makes it easier to express complex, nested, and conditional parameter spaces. As an example, the number of layers in a neural network can be sampled first, while the number of units per layer can be determined accordingly in a loop within the same trial.

Each of the Optuna trials corresponds to a single evaluation of the objective function. The optimizing process is then referred to as a *study*. The framework is supporting modular programming, which allows for the separation of model architecture and optimization logic. This makes it suitable for use in both research as well as production environments.

One of the major contributions of Optuna is its integrated support for both advanced sampling strategies and pruning of unpromising trials. These two components make Optuna a highly cost-effective tool for optimization.

Optuna provides different ways to explore the search space, including methods that treat each parameter separately and those that consider the relationships between parameters. For instance, the Tree-structured Parzen Estimator (TPE) is a method that functions by tuning each parameter without considering how it relates to others. Optuna also allows the use of methods that do account for these relationships, like CMA-ES and Gaussian Process-based methods. These methods are helpful when tuning parameters that influence each other. When using a define-by-run approach, where the parameter space is built as the optimization proceeds, identifying these parameter relationships becomes more complex. Optuna addresses this challenge by looking at the history of previous trials. It examines the data from these trials to understand how the parameters interact and then uses this understanding to apply the appropriate sampling strategy that takes these correlations into account.

The pruning mechanism is based on the Asynchronous Successive Halving Algorithm (ASHA). This algorithm evaluates intermediate results during training and aborts trials

that are underperforming early in the process. This helps the computational sources to be redirected towards more promising configurations. Unlike synchronous approaches, Optuna's asynchronous pruning works efficiently in distributed settings. Settings, where workers operate independently and do not need to wait for global synchronization.

Pruning decisions are made through calls to the `trial.report()` and `trial.should_prune()` Application Programming Interfaces (API)s. Whenever a trial shall fail to meet progress expectations as certain steps, based on validation accuracy, it is terminated. This leads to significant acceleration of the optimization process, especially for deep learning models with long training times.

Optuna is designed so that it works out-of-the-box for not only large distributed workloads, but also small-scale experiments. By default, it uses an in-memory data structure as a backend. This makes it ideal for use in interactive environments such as Jupyter Notebooks. For more robust or parallel deployments, which was out of the scope of this work, is also supports SQL backends like SQLite or PostgreSQL.

The results from optimization are stored in a central storage and and can queried, exported (for example via pandas DataFrames), or visualized through Optuna's build-in dashboard. These features allow for real-time tracking of trial progress, distribution of parameters, and convergence curves.

Through its define-by-run API, efficient sampling and pruning algorithms, and adaptable architecture, Optuna becomes a powerful toolkit for hyperparameter optimization. The diversity of its performance has been demonstrated across a wide range of domains, including image classification, database tuning, and video encoding. The features provided by Optuna were essential for the efficient and reproducible tuning of model used in this work, as well as for introducing a weak supervision into the gravitational wave detection mechanisms [1].

3. Methodology

3.1. White noise

In the initial phase of our study, we generated 2-second-long spectrograms from pure white Gaussian noise with varying standard deviations, σ_w, in the range of 1.0–4.0×10^{-21}. A total of 4×10^5 spectrograms were created for training and validation, using a 70:30 split. Additionally, an independent set of 10^5 spectrograms was generated for testing, both with and without gravitational wave (GW) signal injections.

After training the autoencoder exclusively on noise-only spectrograms, the model successfully learned to reconstruct pure noise. The resulting reconstruction error ranged from 10.1×10^{-4} to 27.3×10^{-4} across the considered range of σ_wW, indicating stable performance under different noise amplitudes.

We then proceeded to evaluate the detection performance by injecting synthetic GW signals into the test set. These signals were generated using the `IMRPhenomD` approximant, with component masses drawn from uniform distributions in the range of 100–500, $M_\odot$. All sources were placed at a fixed luminosity distance of 1 Gpc to simplify the analysis.

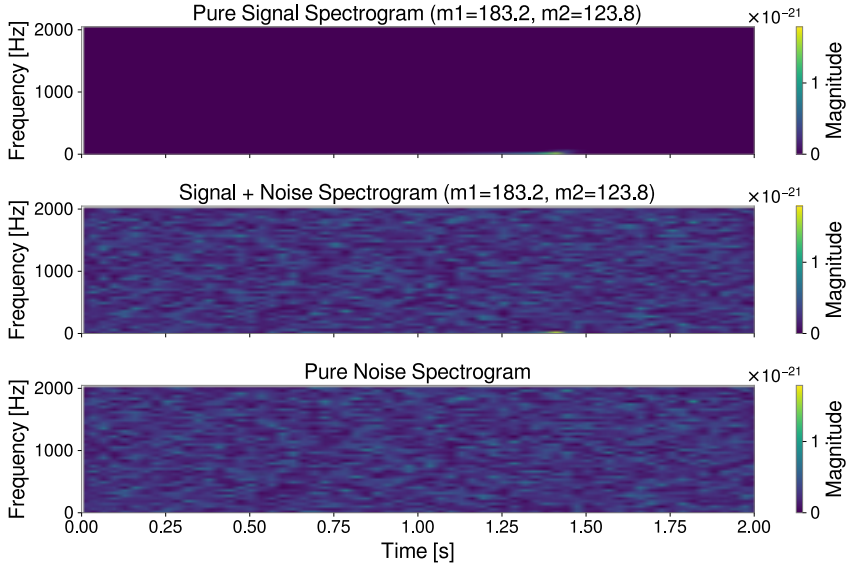

Figure 3.1.: Spectrograms of GW signal, signal in noise ($\sigma_w = 2.0 \times 10^{-21}$), and noise.

To avoid overfitting to specific coalescence times, each injected waveform was randomly time-shifted within the 2-second window, ensuring that the full signal remained fully contained within the spectrogram. This setup enabled a controlled assessment of the model's sensitivity as a function of the total source-frame mass, under varying noise conditions.

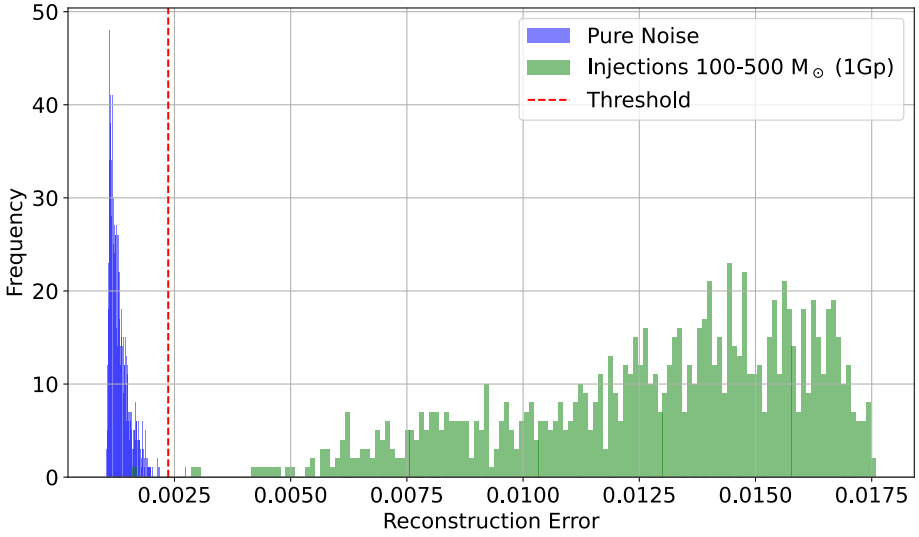

Figure 3.2.: Distribution of reconstruction errors for pure white noise with $\sigma_w = 2.0\times10^{-21}$ (blue) and for the same noise containing injected gravitational-wave signals (green). The vertical line marks the selected anomaly detection threshold. Separation between the two distributions indicates the model's ability to distinguish signal-containing data from noise-only inputs.

3.2. Einstein Telescope Data

3.2.1. Data Selection and Pre-processing of noise

In the second stage of our study, we employed mock data from the Einstein Telescope (ET) Collaboration, corresponding to approximately one month of simulated observation [16]. All noise data used for both training and gravitational wave (GW) injections are used from this dataset, which is publicly available in .gwf format. The data were generated by placing the ET detector at the Virgo site in Cascina, with the E_1 vertex aligned with the Virgo vertex, as described in [16]. The mock dataset includes realistic noise strains for the three ET interferometers (E_1, E_2, and E_3), arranged in the detector's proposed triangular configuration, and provides an additional null stream channel defined as $E_0 = (E_1 + E_2 + E_3)/\sqrt{3}$.

The full dataset contains a variety of simulated compact binary coalescence events:

59,540 binary neutron star (BNS), 6,578 binary black hole (BBH), and 1,977 neutron star–black hole (NSBH) mergers. For this phase, we focused exclusively on noise-only segments for training, ensuring the model learns the statistical structure of the background noise.

The pre-processing began by selecting the $E_1 : STRAIN$ channel and down-selecting the time series to a sampling rate of 8192 Hz. The continuous strain data were divided into 2048-second chunks and further split into non-overlapping 2-second segments, each containing 16,384 samples.

To whiten the strain data, we estimated the power spectral density (PSD) using Welch's method, with a segment length of 4096 samples and 50 percent overlap. The final PSD was interpolated to match the FFT frequency bins of each segment, and the whitening was performed in the frequency domain by dividing the FFT of each segment by the square root of the interpolated PSD. This procedure effectively flattened the frequency content, which reduces spectral features and enhancing the sensitivity to broadband anomalies.

The whitened time-domain segments were converted into PyTorch tensors of shape $[1, 16384]$, corresponding to single-channel inputs. These were grouped into batches of 1000 and saved as .pt files for efficient access during training. The autoencoder model was subsequently retrained and validated using the same procedure as in the first stage, but now on ET noise spectrograms derived from this more realistic, astrophysically motivated dataset.

While data with injections done by the collaboration itself were not part of this thesis scope, one of these events, detected by our architecture can be seen in Figure 3.3 below.

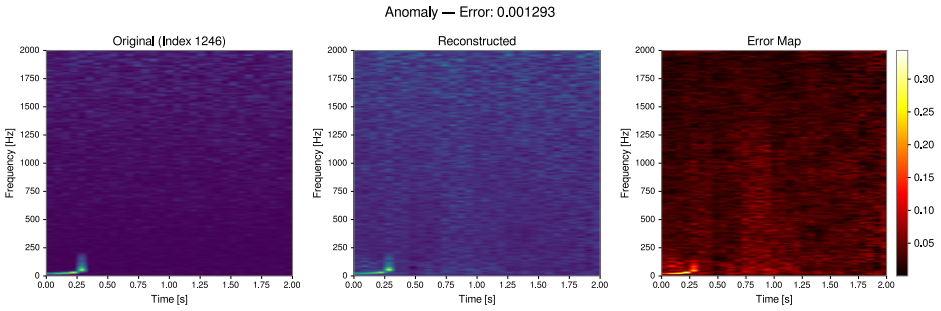

Figure 3.3.: Detection example from the Einstein Telescope mock dataset. Left: original spectrogram of ET noise containing an injected gravitational-wave signal provided by the ET Collaboration. Middle: reconstructed spectrogram produced by the convolutional autoencoder. Right: corresponding reconstruction error map, with brighter regions indicating higher deviation from the learned noise baseline (e.g detected anomaly). Data for this detection are coming from the ET collaboration [16].

3.2.2. Injection of gravitational waves

For evaluation of detection accuracy, during the parametrization as well as for the final testing of autoencoders performance the injections of gravitational waveforms at different masses and distances were injected into the Einstein telescope noise.

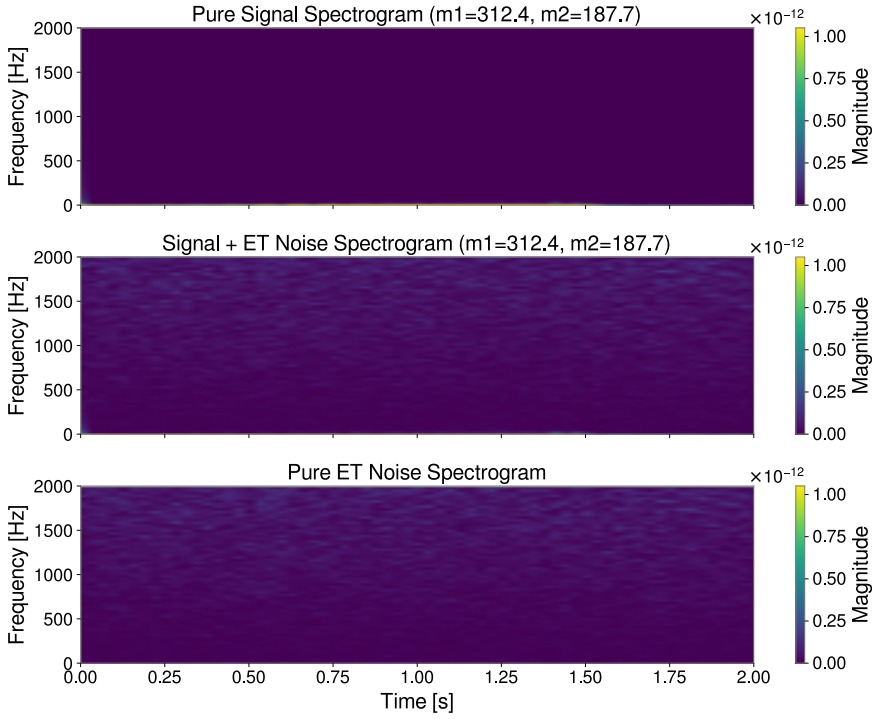

Figure 3.4.: Example of a gravitational-wave signal injection into simulated Einstein Telescope (ET) noise. Top: spectrogram of a pure IMRPhenomD waveform ($m_1 = 312.4\,M_\odot, m_2 = 187.7\,M_\odot$). Middle: the same signal embedded in realistic ET noise, representing the input to the autoencoder. Bottom: spectrogram of pure ET noise without any injection.

3.2.3. Model Architecture

As per the definition of a classical convolutional autoencoder, ours as well consists of two main components - encoder an decoder mirroring each other both with 3 layers.

Input, in our case a spectrogram of gravitational wave data, goes through each layer, lowering the dimension in each step. The architecture of the encoder is as follows:

- **ConvBlock1** Applies a 2D convolution with 1 input and 32 output channels using a 3×3 kernel (with padding=1), followed by batch normalization, dropout (p=0.079),

and 2×2 max pooling. This reduces the spatial resolution by half while increasing the number of feature maps.

- **ConvBlock2** Applies a convolutional layer that maps 32 to 64 channels, again followed by batch normalization, dropout, and 2×2 max pooling.
- **ConvBlock3** Performs the final encoding step with a 2D convolution from 64 to 128 channels, followed by normalization, dropout, and max pooling. The output of this block serves as the latent representation.

Once the input goes through these encoding layers all the way to the latent space, it is followed by a decoding process mirroring the encoding, aiming for reconstructing the original input into preferably as close to identical as possible output. This decoding process consists of layers:

- **DeconvBlock1** The decoder starts by unpooling the latent representation using the indices stored during the last max-pooling operation. A transposed convolution is applied to map 128 to 64 channels, followed by batch normalization and dropout.
- **DeconvBlock2** Performs a second unpooling operation and applies a transposed convolution from 64 to 32 channels, this is again followed by batch normalization and dropout.
- **DeconvBlock3** The final decoder block performs unpooling using the indices from the first encoder layer and is followed by a transposed convolution from 32 back to 1 channel to reconstruct the original input shape.

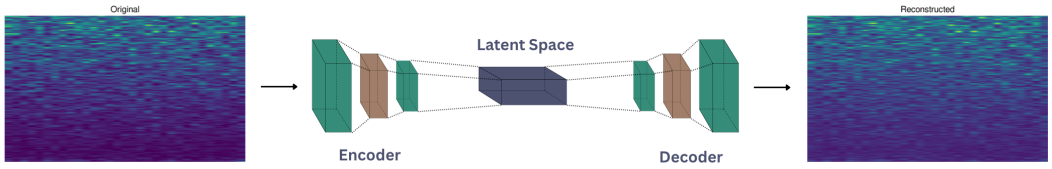

Figure 3.5.: Visual representation of a convolutional autoencoder architecture with three convolutional and three deconvolutional layers. The input consists of spectrograms of Einstein Telescope noise, and the output is corresponding to the reconstructed version of the input.

3.3. Optimization and Introduction of Weak Supervision

While the architecture of the model stayed the same, several hyperparameters were adjusted to improve performance. This tuning was done using Optuna which is a framework designed for hyperparameter optimization.

Optuna tested combinations of different parameters such as the number of epochs, dropout rate, learning rate, and others. After each training run, it reported the reconstruction error on spectrograms that contained only noise.

To push the optimization further, we introduced a dataset with weak gravitational wave signals of binary black hole mergers injected into the noise after the first phase of training. From that point on, each optimization trial was guided by two goals: minimize the reconstruction error on noise, and at the same time maximize the reconstruction error on data that contained weak anomalies.

$$\text{gap} = \frac{\mu_{\text{anomaly}} - \mu_{\text{noise}}}{\mu_{\text{noise}}}. \tag{3.1}$$

This metric, called the gap, expresses how much higher the reconstruction error is on anomalous data compared to clean noise. It helped guide the optimizer to models that were more sensitive to small deviations caused by weak gravitational wave signals. This approach adds a simple form of weak supervision — not through labels, but by rewarding models that perform differently on noise and anomalous data.

3.3.1. Reconstruction Error and Its Meaning

The quality of the reconstruction of the input produced by an autoencoder is typically evaluated using a *reconstruction error*, which calculates the difference between the input data and its reconstruction done by the autoencoder. The most commonly used metric for this purpose is the *mean squared error* (MSE), defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2,$$

where x_i denotes the i-th element of the original input, $\hat{x}_i$ is the corresponding reconstructed output and N is the total number of input elements. The term $(x_i - \hat{x}_i)^2$ represents the squared difference between the original and reconstructed value for each dimension, and the loss is averaged across all dimensions.

A low reconstruction error means that the model is able to accurately reproduce the input, this is suggesting that the data sample is similar to those seen during training. In contrast, a high reconstruction error implies a poor reconstruction and may signal the presence of an anomaly. This makes autoencoders particularly effective for *anomaly detection*, as they tend to reconstruct familiar (normal) inputs well, and struggling to accurately reproduce new or unusual patterns.

4. Results

The performance of the developed architecture of the convolutional autoencoder will be evaluated before and after introducing the weak-supervision method and compared in the following *Discussion*.

4.1. Detection ability without using Weak Supervision

Originally, the given architecture was subject to a process of parameterization to find the best parameters set-up for the lowest calculated reconstruction error on pure noise. Simply put, the goal of our output was to tune the system so that it could reconstruct the noise spectrograms with the highest precision possible. During this optimization, the model was trained using the Adam optimizer with mean squared error (MSE) as the loss function.

This optimization had tested parameters in the range of:

- **Number of epochs: 45 - 75**
- **Learning rate: $\mathbf{1\,e^{-6}}$–$\mathbf{6\,e^{-4}}$** (log scale)
- **Dropout 0.005 - 0.10**
- **Weight Decay: $\mathbf{1\,e^{-9}}$–$\mathbf{1\,e^{-4}}$** (log scale)

Over 70 trials. Such an optimization process over given 260 000 spectrograms took approximately 96 hours; however, this process was not explicitly timed. The best parameters found gave the loss on pure ET noise as **0.000317**. After adjusting these parameters and final training, files with injections of binary black hole mergers of various masses at 1 and 3 GPc were tested for anomalies as well as files with yet-unseen pure noise from Einstein Telescope without an event injected.

Parameters chosen by the parametrization process were as follows:

- *learning rate:* 0.0003428457490522765
- *dropout:* 0.005742095271666759
- *weight decay:* $4.082344822641397\text{e}^{-9}$
- *number of epochs:* 59

As shown in Figure 4.1, the reconstruction errors for injected events are not clearly distinguishable from those of the pure noise data, regardless of the threshold chosen. While this may not pose a critical issue for the current dataset, it presents a fundamental limitation: if these relatively loud events become weaker due to increased distance or less favorable sky location, the autoencoder is likely to fail at detecting them.

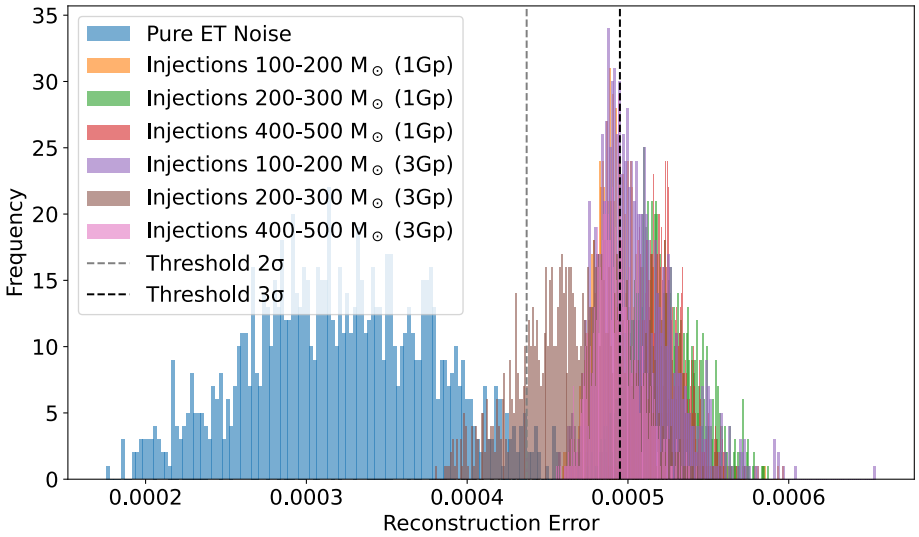

Figure 4.1.: Reconstruction error distributions for pure Einstein Telescope (ET) noise and ET noise containing injected binary black hole signals of various total masses and luminosity distances. Vertical dashed lines indicate anomaly detection thresholds at 2σ and 3σ above the noise mean. Overlap between noise-only and signal-containing distributions highlights the limited separability achieved by using a purely unsupervised setup.

For numerical results of the detection ability of this system, for this specific dataset, two types of thresholds were defined:

$$\text{Threshold } 2\ \sigma = \mu + 2\sigma,$$

and

$$\text{Threshold } 3\ \sigma = \mu + 3\sigma,$$

where μ and σ represent the mean and standard deviation of the reconstruction errors over the noise-only test set.

Table 4.1 below shows numerical results of detection ability and false rate alarm for data of pure noise for both these thresholds.

While these results — particularly when using the 2σ threshold — are not unsatisfactory, it is important to stress that the evaluation was carried out on relatively simplified and

Table 4.1.: Detection rates of injected signals across different mass ranges and distances, compared for 2σ and 3σ anomaly thresholds.

	Threshold 2σ	**Threshold 3σ**
False alarm rate for pure ET noise	2.60%	0.30%
Detection 100–200 $M_\odot$ (1 Gpc)	100%	49.10%
Detection 200–300 $M_\odot$ (1 Gpc)	100%	92.60%
Detection 400–500 $M_\odot$ (1 Gpc)	100%	97.80%
Detection 100–200 $M_\odot$ (3 Gpc)	100%	60%
Detection 200–300 $M_\odot$ (3 Gpc)	82.60%	21.00%
Detection 400–500 $M_\odot$ (3 Gpc)	100%	40.30%
Detection Ability	97%	60.13%

thus more easily detectable waveforms with lower luminosity distances. These conditions represent a best-case scenario in which the model would be expected to achieve its highest performance. Introducing weaker signals with additional physical parameters, such as spins or orbital eccentricity, would very likely lead to a decrease in detection performance, indicating potentially greater challenges during real observational runs.

Furthermore, the observed 2.6% false alarm rate is relatively high for a system that has not yet reached optimal detection capability. Such a level of spurious triggers would require additional downstream processing to reliably distinguish genuine astrophysical events from instrumental glitches, thereby increasing computational demands — an undesirable outcome for real-time detection pipelines.

4.2. Results: Influence of Weak Supervision

Following the completion of the unsupervised optimization phase, the model architecture remained unchanged, and the hyperparameter settings was:

- **Number of epochs: 45 - 70**
- **Learning rate: $1e^{-5}$–$5e^{-4}$** (log scale)
- **Dropout 0.005 - 0.10**
- **Weight Decay: $1e^{-9}$–$1e^{-4}$** (log scale)

This parametrization ran over 50 trials. The optimization process ran also over the same given 260 000 spectrograms and took approximately 86 hours. However, one simple modification was present. In this second phase, each model, trained solely on pure noise data, was subsequently evaluated on two distinct validation sets: one containing only noise spectrograms and another with injected weak anomalies. After this run of parametrization, the best trial had loss on pure ET noise: 0.000840 and on noise with injected anomalies: 0.002327. The score, that was calculated after each trial in the form of:

```
gap = (anomaly_loss - noise_loss) / (noise_loss)
print(f"Trial {trial.number} | Noise Loss: {noise_loss:.6f} | Anomaly
    Loss: {anomaly_loss:.6f} | Score: {-gap:.6f}")
```

was for the best trial -1.771847.
And the parameters chosen by the parametrization process were as follows:

- *learning rate:* 0.0005950527652410784
- *dropout:* 0.08028083975596854
- *weight decay:* $7.216242101650261e^{-7}$
- *number of epochs:* 70

The anomalous dataset included binary black hole (BBH) merger signals, with component masses ranging from 30 to 100 solar masses, placed at randomly distributed luminosity distances between 3 and 5 Gpc. This particular dataset was selected because, in earlier experiments, it consistently exhibited the lowest detection performance among all evaluated data sets. Its inclusion, therefore, served as a challenging benchmark for testing the model's generalization ability and sensitivity to weak signals.

After each training iteration, the reconstruction error was computed for both the noise and anomaly datasets. The primary goal of this extended by using optimization strategy which was to maximize the separation between these two error distributions — that is, to minimize the reconstruction error on pure noise while simultaneously increasing it for the anomalous data. A larger gap between these two metrics indicates shows an improved ability of the model to detect subtle deviations from the expected noise patterns.

To evaluate the effectiveness of this weak supervision approach, the final model was tested on the same validation datasets as those used in the experiment described in *Detection ability without using Weak Supervision*. This allowed for a direct comparison of the models' detection performance and an assessment of whether the integration of weak supervision during optimization leads to measurable improvements in gravitational wave detection.

The final reconstruction errors for both validation datasets are presented in Figure 4.2.A he anomaly detection threshold shown was defined as:

$$\text{Threshold} = \mu + 5\sigma,$$

where μ and σ represent the mean and standard deviation of the reconstruction errors over the noise-only test set. For the threshold of this 5σ, false rate alarm is 1,2 % and detectability of all given data is 100 %. However, in this ideal case, where the reconstruction errors for noise-only and anomalous spectrograms are completely separated without overlap, the threshold could, in principle, be placed to achieve a zero false alarm rate while maintaining the detectability of 100 % of the introduced anomalies.

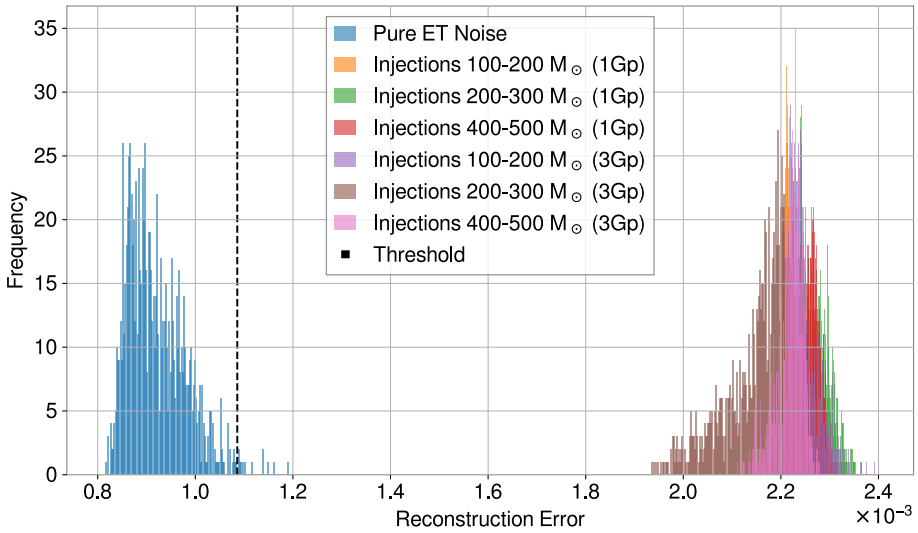

Figure 4.2.: Reconstruction error distributions for pure Einstein Telescope (ET) noise and ET noise with injected weak gravitational-wave signals after introducing weak supervision. Compared to the unsupervised case, the distributions show increased separation, indicating improved sensitivity of the convolutional autoencoder to low-SNR anomalies. The vertical dashed line marks the selected anomaly detection threshold.

5. Discussion

5.1. Interpretation of Results

The results provided in Chapter 4 show the capability and limitations of using a Convolutional Autoencoder for the detection of gravitational waves in an environment with high noise present. The objective of this thesis was to determine whether the use of unsupervised training on noise-only data from the Einstein Telescope (ET) collaboration may result in distinguishing anomalies from pure noise and therefore allowing detection of not only predefined waveforms in highly noisy data to be detected but also avoiding mismatches of these predefined waveforms with detected anomalies, with less computational power than the current *Matched Filtering* method. This approach aims to be more efficient, while being a less computationally heavy approach to detect gravitational waves, than the current method.

Limitations of this research came primarily from using a computer with a GPU — this limited the size of the data used for training. While on the available GPU 260,000 spectrograms were used, 70% of which were used for training itself, this corresponds to approximately 4 days of available noise data from the overall 30-day mock data challenge from the Einstein Telescope collaboration. Using all available data should increase the detection ability. However, the whole parametrization process would require repetition, as different data would lead to different optimal parameters.

The results of this work not only show that convolutional autoencoders have strong predispositions to detect anomalies in high-noise environments but also that the introduced methodology of using weak supervision can increase the performance of the given architecture.

The parametrization process with no supervision ran over 70 trials for 96 hours and allowed detection of 97% of introduced anomalies for a threshold of 2σ and 60.13% for a threshold of 3σ. While this performance is not unsatisfactory and may possibly be improved — with a larger dataset of noise-only signals introduced to the training in later stages — the 2.6% false alarm rate is alarmingly high for imperfect detection of introduced, simplified anomaly waveforms.

The difference between the fully unsupervised and weakly supervised versions of parametrization lies in introducing an additional dataset during parametrization, but not during training itself. The Optuna was tasked with not aiming for the lowest reconstruction error on noise-only data, but to have the lowest reconstruction error on noise-only and the highest for data containing the anomaly. So, that gap between reconstruction errors for both datasets is the biggest possible. This desired separation can be seen below in Figure 5.1.

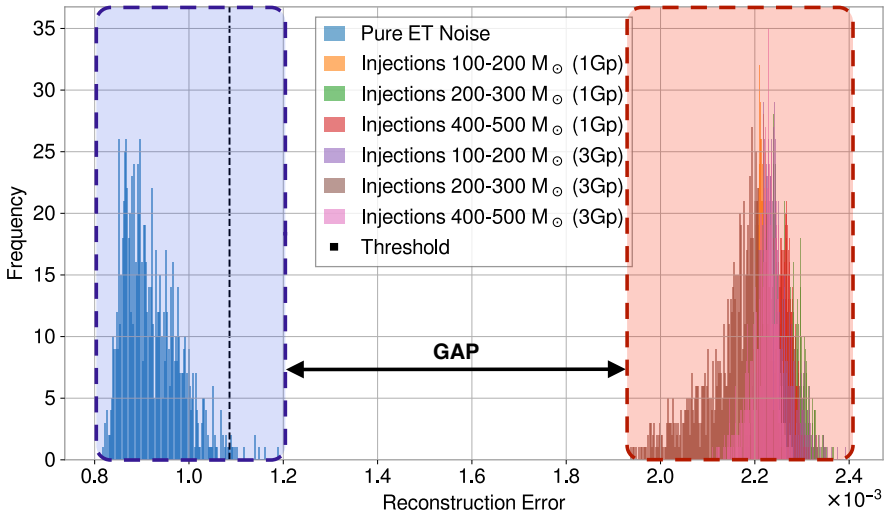

Figure 5.1.: Visualization of the desired gap between reconstruction errors for noise-only data and data containing gravitational-wave anomalies. A larger gap indicates better separability in the model's latent feature space, allowing more reliable discrimination between normal and anomalous inputs.

The parametrization process where weak supervision was introduced ran on the same dataset over 50 trials for 86 hours. Compared to parametrization with no supervision, each trial was slightly longer, which is most likely caused by the system preferring a higher number of epochs, as can be deduced from the final parameters chosen. However, this process overall required less computational power while leading to perfect separation of noise-only data from data containing anomalies. The anomaly dataset used during parametrization was identical to that used for training the systems with the best-found parameters, this allowed for a direct comparison under equivalent conditions.

Introducing the weak supervision technique in this work allowed the system to achieve 100% detection while having a 0% false alarm rate.

5.2. Potential Applications

While the primary aim of this work is its integration into gravitational wave detection systems, convolutional autoencoders — particularly when employed in anomaly detection settings — possess a much broader range of potential applications. Their inherent ability to identify subtle and often temporally or spectrally localized deviations in high-noise datasets make them well suited for tasks in a variety of domains that face similar

signal-to-noise challenges.

For example, in medical imaging, such architectures could help in the early detection of rare pathologies that manifest as small, localized irregularities within large datasets, such as MRI or CT scans. In industrial monitoring, they could be deployed to identify precursors to mechanical failures by detecting atypical vibration patterns or spectral signatures within sensor networks. Within other branches of experimental physics, these methods could assist in spotting rare particle interaction events or subtle deviations in detector outputs, where the volume of acquired data renders exhaustive manual inspection infeasible.

By adapting the network architecture and preprocessing pipelines to the statistical and morphological characteristics of the target domain, convolutional autoencoders are a new, flexible, data-driven approach capable of generalizing to a wide range of anomaly detection problems beyond the scope of gravitational wave astronomy.

6. Conclusion

6.1. Summary of Findings

This work's primary objective was to investigate whether using convolutional autoencoders - a type of neural network - could be used as a computationally effective alternative to the *Matched Filtering* method while also allowing to detect broader range of waveforms of gravitational waves. This study focused on training three-layered convolutional autoencoder (CAE) exclusively on simulated noise-only data from the Einstein Telescope (ET) and determinating, whether or not, this approach would enable it to distinquish between pure noise and inputs containing anomalies, such as gravitational waves.

For this research, spectrograms were derived from the Einstein Telescope Collaboration's mock data challenge. This data consisted of realistic, non-Gaussian and non-stationary noise segments, into which, in the end, gravitational wave signals were injected.

These injected signals, for determining the ability of the trained architecture to detect anomalies, varied in component masses and luminosity distances. The majority of these signals had high SNR and were morphologically simple waveforms.

This approach allowed for evaluation of the first versions of these architectures used for these specific versions and comparison in unsupervised and supervised training before more challenging applications could be assessed. Due to hardware limitations - specifically using a single GPU machine - the total dataset was restricted to 260 000 spectrograms, of which 70% were allocated for training. This data corresponds to approximately four days of Einstein Telescope noise data, a subset of the thirty days available within Mock Data Challenge 1.

Two parametrization strategies were investigated:

1. **Fully Unsupervised parametrization**, in which optimization was based solely on reconstruction error for noise-only inputs.

2. **Weak-supervision parametrization**, in which the optimization process additionally incorporated reconstruction error separation between noise-only and anomaly-containing inputs.

In the fully unsupervised configuration version, our convolutional autoencoder achieved a detection rate of 97% for a 2σ threshold and 60.13% for a 3σ threshold, with a false alarm rate of $2.6\% for$ given anomaly dataset. These results show, that the model was able to identify a significant fraction of injected anomalies in the idealized high-SNR scenarios, but also revealed some limitations in noticeable overlap between reconstruction error distributions for noise and signal-containing samples, which suggest that the per-

formance would likely degrade for more complex and lower-SNR waveforms, such as those incorporating spins or orbital eccentricity, or events placed in larger luminosity distances.

The second proposed method to employ convolutional autoencoder architecture into the detection systems was to use weakly supervision approach. This brings a dramatic improvement in separation between two classes.

Conducted over 50 trials and 86 hours of total runtime, this parametrization process slightly increased the duration of individual trails (likely due to a preference for higher number of epochs) but resulted in perfect class separation under identical data conditions. The final trained model achieved a 100% detection rate with a 0% false alarm rate. Also, despite the longer duration of individual trials, the overall computational cost over the weak supervision approach was lower than the one of fully unsupervised parametrization.

The findings of this study shows that convolutional autoencoders, especially when enhanced with weak supervision during hyperparameter optimization, can provide a robust and computationally efficient framework for gravitational wave detection in high-noise environment. This approach offers key advantages such as:

- It is essentially data-driven and does not require explicit assumptions about the waveform morphology, making it adaptable to a wide range of signal types.
- It achieves significant gains in detection accuracy and false alarm reduction compared to a purely unsupervised approach.
- It maintains flexibility for integration into real-time detection pipelines, where computational efficiency is critical.

This methodological framework might give potential applicability also beyond gravitational wave astronomy. Possibly in other domains, where rare or weak signals must be detected in large volumes of noisy data.

In summary, this work provides a proof of concept that convolutional autoencoders, especially when used with a weak supervision method, can be used as a powerful anomaly detection tool for gravitational wave data. These results lay the groundwork for the future research incorporating more diverse and astrophysically realistic signal models, larger datasets and integration into the multi-stage pipelines required for operational gravitational wave observatories.

6.2. Future Work

This work should be followed by several steps to become ready for application to real gravitational wave data—not only from future generations of detectors but also from newly acquired data from current observatories. One of the still unresolved challenges in detection is distinguishing gravitational wave signals from potential glitches, which are commonly present in the data. To address this, an additional post-detection pipeline should be developed. Another issue encountered during this work was that, for supermassive black hole mergers—where the entire signal lies below the 15 Hz limit — the developed

autoencoder was able to detect that an anomaly was present within the spectrogram, but was not able to properly localize it. Resolving this problem, which most likely arises from the logic of power spectral density (PSD) computation, is a key step for future parameter estimation of such events. These enhancements would not only improve the detection of low-frequency events, but also expand the method's applicability in pre-merger signal characterization and multi-messenger follow-up analyses.

A. Appendices

A.1. Code and Algorithm Details

A.1.1. Preparing ET Noise into Spectrograms

Description: This script processes raw Einstein Telescope (ET) noise data in GWF format, extracts fixed-length segments, applies whitening, and saves the resulting 1D time-domain waveforms as PyTorch tensors. These waveforms are used as input to an autoencoder model for anomaly detection.

Dependencies: numpy, scipy, torch, gwpy, os, pathlib

Input: Directory structure with .gwf files containing ET strain data (channel "E1:STRAIN").

Output: Batches of one thousand preprocessed waveform segments saved as .pt files in PyTorch format.

```
    # Settings
    base_data_dir = "/path_to_.gwf_data"
    channel = "E1:STRAIN"
    sample_rate = 8192
    segment_duration = 2
    segment_length = sample_rate * segment_duration

    output_folder = "/home/output_folder"
    os.makedirs(output_folder, exist_ok=True)
    batch_size = 1000

    # Whitening
    def whiten(data, f_psd, psd, fs):
        psd = np.maximum(psd, 1e-20)
        freqs = np.fft.fftfreq(len(data), d=1/fs)
        psd_interp = np.interp(freqs, f_psd, psd)
        white_fft = np.fft.fft(data) / np.sqrt(psd_interp)
        return np.real(np.fft.ifft(white_fft))

    # Loading GWF files
    gwf_files = []
    for root, _, files in os.walk(base_data_dir):
        for file in sorted(files):
            if file.endswith(".gwf"):
                gwf_files.append(os.path.join(root, file))

    print(f"Found {len(gwf_files)} .gwf files to process.")

    # Processing
    current_batch = []
```

```
    batch_counter = 0

    for idx, data_file in enumerate(gwf_files, 1):
        print(f"[{idx}/{len(gwf_files)}] Processing file: {data_file}")

        try:
            strain = TimeSeries.read(data_file, channel=channel)
            f_psd, psd = welch(strain.value, fs=sample_rate, nperseg
=4096)
            num_segments = int((strain.times.value[-1] - strain.times.
value[0]) // segment_duration)

            for i in range(num_segments):
                start = strain.times.value[0] + i * segment_duration
                end = start + segment_duration
                if end > strain.times.value[-1]:
                    break

                segment = strain.crop(start, end).value
                segment = highpass_filter(segment)
                segment = whiten(segment, f_psd, psd, sample_rate)

                tensor = torch.tensor(segment, dtype=torch.float32).
unsqueeze(0)  # [1, 8192]
                current_batch.append(tensor)

                if len(current_batch) >= batch_size:
                    batch_tensor = torch.cat(current_batch, dim=0)  # [
batch_size, 8192]
                    save_path = os.path.join(output_folder, f"
waveform_batch_{batch_counter:03d}.pt")
                    torch.save(batch_tensor, save_path)
                    print(f"Saved batch {batch_counter} with {len(
current_batch)} waveform segments")
                    batch_counter += 1
                    current_batch = []

        except Exception as e:
            print(f"Error processing {data_file}: {e}")

    # Saving Last Batch
    if current_batch:
        batch_tensor = torch.cat(current_batch, dim=0)
        save_path = os.path.join(output_folder, f"waveform_batch_{
batch_counter:03d}.pt")
        torch.save(batch_tensor, save_path)
        print(f"Saved final batch {batch_counter} with {len(current_batch
)} waveform segments")

    print("Done.")
```

A.1.2. Injecting Gravitational Waves into ET Noise

Description: This script injects simulated gravitational wave (GW) signals into pre-processed Einstein Telescope (ET) noise segments. The injected signals are generated using the IMRPhenomD model from PyCBC, of binary black hole (BBH) with random (within desired range) masses and distance parameters. The combined waveforms are then whitened and stored for use in training the anomaly detection autoencoder.

Dependencies: numpy, scipy, torch, pycbc, gwpy, os, pathlib.

Input: Raw ET strain data in `.gwf` format and a configured preprocessing pipeline including segmenting and whitening.

Output: Batches of 1D waveform segments containing injected GW signals, saved as `.pt` PyTorch tensors in the specified output directory.

```
    # Settings and Whitening is unchanged
    for data_file in gwf_files:
        try:
            strain = TimeSeries.read(data_file, channel=channel)
            full_data = strain.value

            # Calculating PSD
            f_psd, psd = welch(full_data, fs=sample_rate, nperseg=
sample_rate * 4)
            num_segments = int((strain.times.value[-1] - strain.times.
value[0]) // segment_duration)

            for i in range(num_segments):
                start = strain.times.value[0] + i * segment_duration
                end = start + segment_duration
                if end > strain.times.value[-1]:
                    break

                segment = strain.crop(start, end).value.copy()
                injected_segment = segment.copy()

                # Wave injection
                m1 = np.random.uniform(30, 100)
                m2 = np.random.uniform(30, 100)
                distance = np.random.uniform(3000, 5000)

                hp, _ = get_td_waveform(
                    approximant="IMRPhenomD",
                    mass1=m1, mass2=m2,
                    delta_t=1 / sample_rate,
                    f_lower=3,
                    distance=distance,
                    spin1z=0.0, spin2z=0.0,
                    eccentricity=0.0,
                    inclination=0.0
                )

                gw_tensor = torch.tensor(hp.numpy(), dtype=torch.float32)
                peak = torch.argmax(gw_tensor).item()
```

```
                new_peak = np.random.randint(int(0.2 * segment_length),
    int(0.8 * segment_length))
                shift = new_peak - peak
                gw_tensor = torch.roll(gw_tensor, shifts=shift)

                start_idx = max(0, new_peak - segment_length // 2)
                end_idx = start_idx + segment_length
                gw_tensor = gw_tensor[start_idx:end_idx]

                if len(gw_tensor) < segment_length:
                    gw_tensor = F.pad(gw_tensor, (0, segment_length - len
    (gw_tensor)))

                # Tapering using Tukeyova window
                taper = tukey(segment_length, alpha=0.2)
                gw_tapered = gw_tensor.numpy() * taper

                gw_filtered = highpass_filter(gw_tapered)
                injected_segment += gw_filtered

                whitened = whiten(injected_segment, f_psd, psd,
    sample_rate)
                whitened_tensor = torch.tensor(whitened, dtype=torch.
    float32).unsqueeze(0)
                current_batch.append(whitened_tensor)

                if len(current_batch) >= batch_size:
                    batch_tensor = torch.cat(current_batch, dim=0)
                    save_path = os.path.join(output_folder, f"spec_batch_
    {batch_counter:03d}.pt")
                    torch.save(batch_tensor, save_path)
                    print(f"Saved batch {batch_counter} with {len(
    current_batch)} segments")
                    batch_counter += 1
                    current_batch = []

            except Exception as e:
                print(f"Error processing {data_file}: {e}")

        if current_batch:
            batch_tensor = torch.cat(current_batch, dim=0)
            save_path = os.path.join(output_folder, f"spec_batch_{
    batch_counter:03d}.pt")
            torch.save(batch_tensor, save_path)
            print(f"Saved final batch {batch_counter} with {len(current_batch
    )} segments")

        print("Done.")
```

A.1.3. Model Architecture

Description: This script defines the core convolutional autoencoder architecture used throughout the anomaly detection experiments. It consists of three convolutional and max-pooling layers for encoding, and three corresponding unpooling and transposed convolutional layers for decoding. Batch normalization and dropout are applied after each layer to improve generalization and training stability.

Dependencies: torch, torch.nn

Input: Single-channel spectrograms or time-domain segments formatted as 2D tensors with shape `[1, H, W]`.

Output: Reconstructed output tensor with the same shape as the input. The reconstruction error is used to identify potential anomalies in the input data.

```
class DeeperAutoencoderWithRegularization(nn.Module):
    def __init__(self, d):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.drop1 = nn.Dropout(d)
        self.pool1 = nn.MaxPool2d(2, return_indices=True)

        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.drop2 = nn.Dropout(d)
        self.pool2 = nn.MaxPool2d(2, return_indices=True)

        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.drop3 = nn.Dropout(d)
        self.pool3 = nn.MaxPool2d(2, return_indices=True)

        self.unpool1 = nn.MaxUnpool2d(2)
        self.deconv1 = nn.ConvTranspose2d(128, 64, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(64)
        self.drop4 = nn.Dropout(d)

        self.unpool2 = nn.MaxUnpool2d(2)
        self.deconv2 = nn.ConvTranspose2d(64, 32, 3, padding=1)
        self.bn5 = nn.BatchNorm2d(32)
        self.drop5 = nn.Dropout(d)

        self.unpool3 = nn.MaxUnpool2d(2)
        self.deconv3 = nn.ConvTranspose2d(32, 1, 3, padding=1)

    def forward(self, x):
        orig = x.size()
        x = self.conv1(x); x = self.bn1(x); x = self.drop1(x); x, i1 =
    self.pool1(x); s1 = x.size()
        x = self.conv2(x); x = self.bn2(x); x = self.drop2(x); x, i2 =
    self.pool2(x); s2 = x.size()
        x = self.conv3(x); x = self.bn3(x); x = self.drop3(x); x, i3 =
    self.pool3(x); s3 = x.size()
```

```
        x = self.unpool1(x, i3, output_size=s2); x = self.deconv1(x); x = self.bn4(x); x = self.drop4(x)
        x = self.unpool2(x, i2, output_size=s1); x = self.deconv2(x); x = self.bn5(x); x = self.drop5(x)
        x = self.unpool3(x, i1, output_size=orig); x = self.deconv3(x)
        return x
```

A.1.4. Loading Spectrograms

Description: This script loads preprocessed waveform data stored as PyTorch `.pt` tensors, converts each time-domain waveform into a normalized spectrogram using the short-time Fourier transform (STFT), and interpolates the resulting spectrograms to a fixed size. The final dataset is split into training, validation, and test sets and loaded into PyTorch `DataLoader` objects for use during model training.

Dependencies: numpy, scipy, torch, sklearn, os

Input: Directory containing batches of waveform tensors in `.pt` format, previously preprocessed from ET strain data.

Output: Train and validation `DataLoader` objects containing batches of 2D spectrograms with shape `[1, 256, 31]`.

```
        def compute_spectrogram(waveform, nperseg=512, noverlap=256):
        f, t, Zxx = stft(waveform, fs=4096, nperseg=nperseg, noverlap=noverlap)
        spec = np.abs(Zxx)
        if spec.shape[1] > 4:
            spec = spec[:, 1:-1]
        return spec

    def preprocess_waveforms_to_spectrograms(waveforms):
        processed = []
        for waveform in waveforms:
            spec = compute_spectrogram(waveform.numpy())
            min_val, max_val = np.min(spec), np.max(spec)
            norm_spec = (spec - min_val) / (max_val - min_val) if max_val > min_val else np.zeros_like(spec)
            processed.append(norm_spec)
        return torch.tensor(processed, dtype=torch.float32)

    def load_all_from_folder(folder_path):
        all_waveforms = []
        for filename in sorted(os.listdir(folder_path)):
            if filename.endswith('.pt'):
                path = os.path.join(folder_path, filename)
                data = torch.load(path)
                waveforms = data.get('waveforms', data) if isinstance(data, dict) else data
                all_waveforms.append(waveforms)
        return torch.cat(all_waveforms)

    # Defining path to the noise only data
    noise_folder = "Path/Noise_Only_Data"
```

```
29
30      noise_data = load_all_from_folder(noise_folder)
31      noise_specs = preprocess_waveforms_to_spectrograms(noise_data)
32      noise_specs = torch.nn.functional.interpolate(noise_specs.unsqueeze
        (1), size=(256, 31))
33
34      # Spliting data into training, validation and test
35      train_data, temp = train_test_split(noise_specs, test_size=0.3,
        random_state=42)
36      val_data, _ = train_test_split(temp, test_size=0.5, random_state=42)
37
38      train_loader = DataLoader(TensorDataset(train_data), batch_size=16,
        shuffle=True)
39      val_loader = DataLoader(TensorDataset(val_data), batch_size=16,
        shuffle=False)
```

A.1.5. Parametrization Process not including Weak Supervision

Description: This script defines the hyperparameter optimization routine for the convolutional autoencoder using the Optuna framework in traditional way. It samples values for learning rate, dropout rate, weight decay, and number of training epochs. The model is trained using only spectrograms of noise (without injected gravitational waves), and the objective function minimizes the reconstruction error (MSE) on the validation set.

Dependencies: torch, optuna, sklearn

Input: Spectrogram batches of noise-only data provided via `train_loader` and `val_loader`, along with a defined model architecture.

Output: Optimized hyperparameter set minimizing the average validation loss on pure noise data. The best parameters are printed after the optimization process has been completed.

```
1       def objective(trial):
2           lr = trial.suggest_float("lr", 1e-6, 6e-4, log=True)
3           dropout = trial.suggest_float("dropout", 0.005, 0.10)
4           weight_decay = trial.suggest_float("weight_decay", 1e-9, 1e-4,
        log=True)
5           epochs = trial.suggest_int("epochs", 45, 75)
6
7           model = DeeperAutoencoderWithRegularization(dropout).to(device)
8           optimizer = torch.optim.Adam(model.parameters(), lr=lr,
        weight_decay=weight_decay)
9           criterion = nn.MSELoss()
10
11          for _ in range(epochs):
12              model.train()
13              for batch in train_loader:
14                  x = batch[0].to(device)
15                  optimizer.zero_grad()
16                  y = model(x)
17                  loss = criterion(y, x)
18                  loss.backward()
19                  optimizer.step()
```

```
20
21          model.eval()
22          noise_loss = 0.0
23          with torch.no_grad():
24              for batch in val_loader:
25                  x = batch[0].to(device)
26                  y = model(x)
27                  noise_loss += criterion(y, x).item()
28
29          noise_loss /= len(val_loader)
30          print(f"Trial {trial.number} | Noise Loss: {noise_loss:.6f}")
31
32          return noise_loss  # Optimalizujeme pouze noise loss
33
34      study = optuna.create_study(direction="minimize")
35      study.optimize(objective, n_trials=70)
36
37      print("Best trial:", study.best_trial.params)
```

A.1.6. Parametrization Process including Weak Supervision

Description: This script extends the hyperparameter optimization process by including a weakly supervised objective. In addition to minimizing the reconstruction error on noise-only data, it evaluates the model's ability to distinguish noise from anomaly by comparing the reconstruction losses for both. The final objective function maximizes the reconstruction gap between the two classes.

Dependencies: torch, optuna, sklearn

Input: `train_loader` and `val_loader` containing noise-only spectrograms, and anomaly_loader containing spectrograms with injected gravitational wave signals.

Output: Optimized hyperparameters that maximize the normalized loss gap between anomaly and noise, returned as the best trial result from Optuna.

```
1       def objective(trial):
2           lr = trial.suggest_float("lr", 1e-5, 6e-4, log=True)
3           dropout = trial.suggest_float("dropout", 0.005, 0.10)
4           weight_decay = trial.suggest_float("weight_decay", 1e-9, 1e-4,
    log=True)
5           epochs = trial.suggest_int("epochs", 45, 70)
6
7           model = DeeperAutoencoderWithRegularization(dropout).to(device)
8           optimizer = torch.optim.Adam(model.parameters(), lr=lr,
    weight_decay=weight_decay)
9           criterion = nn.MSELoss()
10
11          for _ in range(epochs):
12              model.train()
13              for batch in train_loader:
14                  x = batch[0].to(device)
15                  optimizer.zero_grad()
16                  y = model(x)
17                  loss = criterion(y, x)
```

```
                loss.backward()
                optimizer.step()

        model.eval()
        noise_loss, anomaly_loss = 0.0, 0.0
        with torch.no_grad():
            for batch in val_loader:
                x = batch[0].to(device)
                y = model(x)
                noise_loss += criterion(y, x).item()
            for batch in anomaly_loader:
                x = batch[0].to(device)
                y = model(x)
                anomaly_loss += criterion(y, x).item()

        noise_loss /= len(val_loader)
        anomaly_loss /= len(anomaly_loader)

        gap = (anomaly_loss - noise_loss) / (noise_loss + 1e-8)
        print(f"Trial {trial.number} | Noise Loss: {noise_loss:.6f} |
    Anomaly Loss: {anomaly_loss:.6f} | Score: {-gap:.6f}")

        return -gap

    study = optuna.create_study(direction="minimize")
    study.optimize(objective, n_trials=50)

    print("Best trial:", study.best_trial.params)
```

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *arXiv preprint arXiv:1907.10902*, 2019.

[2] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[3] Neil J Cornish. Time-frequency analysis of gravitational wave data. *arXiv preprint arXiv:2009.00043*, 2020.

[4] Iddo Drori. *The Science of Deep Learning*. Cambridge University Press, 2022.

[5] Einstein Telescope Collaboration. The science of the einstein telescope. Technical Report ET-0036D-25, ET Collaboration, 2023. Technical Design Study Document.

[6] Adam Gibson and Josh Patterson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, 2017.

[7] Mike Guidry. *Modern General Relativity: Black Holes, Gravitational Waves, and Cosmology*. Cambridge University Press, 2019.

[8] Mike Guidry. Chapter 22: Gravitational waves. In *Modern General Relativity: Black Holes, Gravitational Waves, and Cosmology*, pages 465–488. Cambridge University Press, 2022.

[9] Wei-Che Hsieh, Ziqian Bi, Chuanqi Jiang, Junyu Liu, Benji Peng, Sen Zhang, Xuanhe Pan, Jiawei Xu, Jinlang Wang, Keyu Chen, Pohsun Feng, Yizhu Wen, Xinyuan Song, Tianyang Wang, Ming Liu, Junjie Yang, Ming Li, Bowen Jing, Jintao Ren, Junhao Song, Hong-Ming Tseng, Yichao Zhang, Lawrence K.Q. Yan, Qian Niu, Siilin Chen, Yunze Wang, and Chia Xin Liang. A comprehensive guide to explainable ai: From classical models to llms, 2024.

[10] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Soumen Koley, Maria Bader, Johannes van den Brand, Xander Campman, Henk Bulten, Frank Linde, and Bjorn Vink. Surface and underground seismic characterization at terziet in limburg—the euregio meuse–rhine candidate site for einstein telescope. *Classical and Quantum Gravity*, 01 2022.

[12] Aman Kumar, Deepika Joshi, Abhishek Kumar, and Sanjay Singh. Wavelet-based feature extraction for gravitational wave signals. *Expert Systems with Applications*, 202:117172, 2022.

[13] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning: A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[14] Michele Maggiore. *Gravitational Waves: Volume 1: Theory and Experiments*. Oxford University Press, Oxford, 2008.

[15] Nasar Raiaan, Imran Razzak, and Muhammad Imran. A systematic review of hyperparameter optimization techniques in convolutional neural networks. *Decision Analytics Journal*, 10:100123, 2024.

[16] Tania Regimbau and Jishnu Suresh. A mock data challenge for next-generation detectors. *arXiv preprint*, 2025.

[17] B.S. Sathyaprakash and Bernard F. Schutz. Gravitational-wave astronomy: observational results and their impact. *arXiv preprint arXiv:1511.00231*, 2015.

[18] Chirag Shah. *A Hands-On Introduction to Machine Learning*. Cambridge University Press, 2022.

[19] R. Shaikh, R. Madan, and D. B. Kulkarni. Convolutional autoencoders for image denoising. In Nilanjan Dey and Debashree Mukherjee, editors, *Deep Learning for Image Processing Applications*, pages 69–93. Springer, 2018.