# Detection of gravitational waves using topological data analysis and convolutional neural network

Gerardo Juárez Hernández - A01732799
Andres Villareal A00833915
Salvador Vidal Torres - A01732983

04 de Junio 2024

**Abstract**

Detecting gravitational waves has revolutionized astrophysics, revealing new insights into massive cosmic events. However, identifying these faint signals amidst noise is challenging. This project enhances gravitational wave detection by integrating Topological Data Analysis (TDA) with Convolutional Neural Networks (CNN). We use persistent homology to extract topological features from sliding window embeddings of time-series data, combining them with raw signals for CNN input vectors. Our multi-step process includes data generation, PCA for dimensionality reduction, persistence vector calculation, and deep learning model training. This approach improves detection robustness and accuracy, showcasing the synergy between topological features and neural networks to address low signal-to-noise ratios in gravitational wave astronomy.

## 1 Introduction

As a form of preamble and to support the central idea of the project presented in the article *"Detection of gravitational waves using topological data analysis and convolutional neural network: An improved approach"* it is imperative for us to highlight the importance of the topic of gravitational waves, mainly due to the importance that this topic holds for the general understanding of the theoretical and mathematical arguments that we will present below.

The detection of gravitational waves marks a monumental advancement in astrophysics, providing unprecedented insights into the dynamics of massive cosmic events such as black hole mergers and neutron star collisions. These ripples in spacetime, first predicted by Einstein's theory of general relativity, offer a new observational window into the universe, allowing scientists to probe the fundamental nature of gravity and the behavior of matter under extreme conditions. However, the challenge lies in detecting these incredibly faint signals against a backdrop of substantial noise, which arises from both terrestrial and cosmic sources. Improving the robustness and accuracy of gravitational wave detection is therefore critical for advancing our understanding of the universe.

Traditional methods of gravitational wave detection have relied heavily on matched filtering techniques, which compare observed data with a set of theoretical templates. While effective, these methods can be computationally intensive and may struggle with signals that deviate from expected patterns. In recent years, convolutional neural networks (CNNs) have emerged as a powerful alternative due to their ability to automatically extract features from raw data and identify complex patterns. CNNs have been successfully applied to gravitational wave detection, demonstrating the potential to outperform traditional methods in terms of speed and accuracy. Studies have shown that convolutional neural networks can effectively detect simulated gravitational wave signals in noisy data, highlighting their promise for real-time detection in operational observatories.

Despite the success of CNNs, the challenge of distinguishing faint gravitational wave signals from noise remains significant. This study proposes integrating Topological Data Analysis (TDA) with CNNs to enhance detection capabilities. TDA, and specifically persistent homology, offers a novel approach to feature extraction by capturing the intrinsic topological structures within data. By applying TDA to sliding window embeddings of time-series data, we can extract robust topological features that complement the raw signal information. Combining these topological features with CNNs leverages the strengths of both techniques: the deep learning capabilities of CNNs and the structural insights provided by TDA. This integrated approach aims to improve the detection accuracy and robustness, particularly in low signal-to-noise scenarios, thereby advancing the field of gravitational wave astronomy.

# 2 Theoretical Fundamentals

## 2.1 Gravitational waves

Gravitational waves are disturbances in the fabric of spacetime caused by massive astronomical events, such as the collision of black holes or the merger of neutron stars. Predicted by Albert Einstein in his general theory of relativity in 1916, these waves travel at the speed of light, carrying information about their origin and the nature of gravity.

The most potent examples of gravitational wave sources include supernovae (exploding stars), binary systems of black holes or neutron stars, and galaxy mergers. As these massive objects move, they generate waves that propagate through the universe, distorting spacetime as they pass. [1]

The first direct detection of gravitational waves was made in 2015 by the Laser Interferometer Gravitational-Wave Observatory (LIGO), confirming the collision of two black holes 1.3 billion light-years away. This detection not only validated a key prediction of Einstein but also inaugurated a new era in astronomy, allowing scientists to observe cosmic events that cannot be detected by traditional means such as light. [3]

To detect these waves, extremely sensitive instruments like LIGO and Virgo are used, employing laser interferometry. This method measures the minute changes in distance between mirrors located in large perpendicular arms when a gravitational wave passes, stretching and compressing spacetime. These detectors can register changes smaller than the size of an atomic nucleus, demonstrating the precision and sensitivity

required to observe these phenomena. [4] [3]

## 2.2 Sliding window embedding

Sliding window embedding, also known as time-delay embedding, is a technique used in time series analysis and dynamic systems. This method allows for transforming a time series into a set of points in a higher-dimensional space, facilitating the geometric and topological analysis of the underlying behavior of the system.

The technique is based on Takens theorem, which establishes conditions for reconstructing the attractor of a dynamic system from observations of a function. By sliding a fixed window over the data, a "point cloud" is created that can be analyzed using persistent homology to extract important topological features, such as periodicity and hidden patterns in the data. [7]

## 2.3 Singular Value Decomposition

: The Singular Value Decomposition (SVD) is a technique in linear algebra that allows any matrix $A$ to be decomposed into three specific matrices: $U$, $\Sigma$, and $V^T$. Mathematically, this is expressed as $A = U\Sigma V^T$. [6]

- $U$ is an orthogonal matrix of dimension $m \times m$ called the "left singular vectors matrix". [6]
- $\Sigma$ is a diagonal matrix $m \times n$ that contains the non-negative singular values of $A$ on its diagonal entries. [6]
- $V^T$ is the transpose of an $n \times n$ orthogonal matrix called the "right singular vectors matrix". [6]

## 2.4 Persistent Homology

Persistent homology is a fundamental technique in topological data analysis (TDA) that facilitates the study of topological features of data across various scales. This method is utilized to identify and quantify features such as connected components, holes, and cavities within the data, providing profound insights into the underlying structure. [8]

- Construction of Simplicial Complexes: The initial data is represented as a point cloud. To analyze its topological structure, simplicial complexes are constructed. These complexes consist of sets of vertices, edges, triangles, and their higher-dimensional generalizations that connect nearby points in space. [8]
- Filtration: A sequence of nested simplicial complexes is created by incrementing a parameter (such as the radius in the construction of Vietoris-Rips complexes). Each complex in this sequence captures the structure of the data at a particular scale. [8]

- Homology Computation: For each simplicial complex in the sequence, homology groups are calculated, identifying the topological features present (such as connected components and holes). Persistent homology tracks how these features appear and disappear throughout the filtration process. [8]
- Persistence Diagrams: The results are visualized in persistence diagrams, where each point represents a topological feature with its appearance (birth) and disappearance (death) throughout the filtration. These diagrams allow the identification of persistent features that are robust to perturbations in the data. [8]

### 2.4.1 Relevance in Topological Data Analysis

- Persistent homology is robust to perturbations in the data, allowing the extraction of significant features even in the presence of noise. [8]
- This technique does not depend on the dimensionality or specific coordinates of the data, making it applicable to a wide variety of datasets and problems. [8]
- Persistent homology has been successfully used in multiple domains, including image analysis, neural networks, computational biology, and financial data analysis, among others. Its ability to capture topological structure at different scales makes it extremely versatile and powerful. [8]
- It provides a multiscale summary of the data, which is useful for identifying patterns and structures at different levels of detail, offering a deeper and more comprehensive understanding of the analyzed data. [8]

## 2.5 Convolutional Neural Networks

CNNs are a class of deep neural networks specifically designed to process data with a grid-like structure, such as images. They are characterized by their ability to learn hierarchical representations of data through convolutional and pooling layers, allowing them to identify complex patterns with high precision.

- Convolutional Layers: These layers apply filters (kernels) to the input to produce feature maps. Each filter slides over the input and performs a dot product operation, extracting specific features such as edges, textures, etc. The filter weights are adjusted during training through backpropagation. [11]
- Pooling layers: Also known as subsampling layers, they reduce the dimensionality of the feature maps through aggregation operations such as max pooling or average pooling. This helps to decrease computational complexity and the risk of overfitting while preserving the most relevant features. [11]
- Fully Connected (FC) Layers: These layers are found at the end of the network and perform the classification task. They connect each node of the previous layer with each node of the current layer, producing the network's final output. They often use activation functions such as softmax to generate classification probabilities. [11]

#### 2.5.1 Applications of CNNs in Signal Detection

- CNNs are effective for tasks such as voice recognition and audio signal classification. They can learn discriminative features from audio waveforms, allowing them to distinguish between different sounds or words. [13]
- In industrial and monitoring applications, CNNs are used to detect anomalies in continuous signals, such as mechanical vibrations or electrical signals. This is crucial for predictive maintenance and fault prevention.
- In the medical field, CNNs are applied to the classification of electrocardiograms (ECGs), helping to identify cardiac arrhythmias and other health conditions. The networks can learn complex patterns in ECG signals, improving diagnostic accuracy. [12]

# 3 Methodology

To address the problem of detecting gravitational waves in a noisy environment, we will use a methodology that integrates Topological Data Analysis (TDA) and Convolutional Neural Networks (CNN). The implementation is divided into several key stages:

## 3.1 Generation of Synthetic Data

### 3.1.1 Data Acquisition

Gravitational wave signals were sourced from a pre-existing dataset, `gravitational_wave_signals.npy`, containing multiple recorded waveforms. The dataset comprises signals sampled at a high resolution, with each signal corresponding to a different astrophysical event.

### 3.1.2 Signal Processing and Downsampling

To simulate realistic conditions, the original signals were downsampled by a factor of two. This process reduces the data size while preserving essential features of the waveform, enabling more efficient simulation and analysis.

$$N = \frac{N_{\text{orig}}}{\text{downsample\_factor}}$$

Where $N_{\text{orig}}$ is the original length of the signal, and downsample_factor $= 2$.

### 3.1.3 Noise Generation

Noise is a critical component in the simulation, representing the various interferences encountered in actual gravitational wave detection. Noise coefficients were calculated using the following formula:

$$n_{\text{coeff}} = 10^{-19} \left( \frac{1}{R_{\text{coef}}} \right)$$

Here, $R_{\text{coef}}$ is a set of Signal-to-Noise Ratio (SNR) values linearly spaced between 0.075 and 0.65. This range simulates varying noise levels across different signals.

#### 3.1.4 Signal and Noise Integration

For each simulated signal, a decision was made randomly to determine if the signal would contain a gravitational wave (signal-present) or only noise (signal-absent). The corresponding label was assigned as 1 for signal-present and 0 for signal-absent.

- **Signal-Present**: The gravitational wave signal was combined with noise, and random padding was added to both ends to mimic real-world data imperfections.

$$\text{rawsig} = \text{padrand}(\text{signal} + \text{noise}, N_{\text{pad}}, n_{\text{coeff}})$$

- **Signal-Absent**: Only noise was generated with padding, simulating the absence of gravitational waves.

$$\text{rawsig} = \text{padrand}(\text{noise}, N_{\text{pad}}, n_{\text{coeff}})$$

The function `padrand` introduced random padding by splitting the signal at a random index, adding Gaussian noise before and after the split.

### 3.2 Data preprocessing

Analyzing time series data, such as signals from gravitational wave detectors, often requires transforming the data into a format suitable for visualization and further analysis. This study employs sliding window embeddings and Principal Component Analysis (PCA) to reduce the dimensionality of time series data and visualize its structure.

#### 3.2.1 Sliding Window Embedding

Sliding window embedding is a technique used to convert a one-dimensional time series into a higher-dimensional space, enabling the capture of temporal patterns within the data. This method involves creating overlapping segments (windows) of a fixed size from the time series.

#### 3.2.2 Principal Component Analysis (PCA)

To reduce the dimensionality of the sliding window embeddings, PCA is employed. PCA identifies the principal components that capture the most variance in the data, projecting the high-dimensional data onto a lower-dimensional space. This reduction facilitates the visualization and analysis of the data.

## 3.3 TDA application

Persistent homology will be used to analyze the topological features of the embedded data. This process involves constructing simplicial complexes and creating persistence diagrams that represent the topological features across different scales.

In the context of topological data analysis (TDA), $H_0$ and $H_1$ refer to homology groups which capture different types of topological features within data:

- $H_0$: The 0th homology group captures the connected components of the data. Each element in $H_0$ represents a distinct connected piece of the data. In other words, $H_0$ indicates how many separate pieces there are in your data set.
- $H_1$: The 1st homology group captures one-dimensional holes or loops within the data. These are features that cannot be shrunk to a point within the space.

### 3.3.1 Interpretation in the Context of Gravitational Waves

Given the context of gravitational wave signals:

- **Gravitational Waves (GWs)**: These signals typically exhibit periodic-like behaviors due to oscillations caused by astronomical events like binary black hole mergers. Such signals can create loops or cyclic structures in the feature space when embedded using sliding windows and visualized in a reduced-dimensionality space. Hence, in topological terms, the presence of gravitational waves might be effectively captured by $H_1$, which detects these loops or circular patterns.
- **Noise**: Random noise, especially Gaussian noise, tends to scatter data points without forming coherent structures or cycles. This would typically result in many disconnected components (each noisy segment might appear as an isolated point) rather than forming loops. Therefore, noise would be more related to $H_0$, as it would represent many disconnected components in the data.

## 3.4 Convolutional Neural Network

The obtained topological features, along with the original signals, will be used as inputs to train a CNN. The network will be trained to classify the presence of gravitational waves in noisy data.

Regarding the hyperparameters chosen for the CNN:

- **Kernel Size**: The size of the convolutional filters determines the spatial extent of features that the network can learn. A smaller kernel size may capture finer details, while a larger kernel size may capture more global features.
- **Number of Filters**: The number of filters in each convolutional layer determines the complexity of the learned features. More filters allow the network to learn a greater variety of features but also increase the computational cost.

- **Learning Rate**: The learning rate controls the step size during optimization. A higher learning rate may lead to faster convergence but can also cause the optimizer to overshoot the minimum. A lower learning rate may lead to slower convergence but more stable training.
- **Batch Size**: The batch size determines the number of samples processed before updating the model's parameters. A larger batch size may lead to more stable updates but requires more memory.
- **Number of Epochs**: The number of epochs defines the number of times the entire dataset is passed forward and backward through the network during training. More epochs allow the network to learn more complex patterns but may also increase the risk of overfitting.

The CNN model architecture and hyperparameters are carefully chosen to effectively classify homologies using persistence diagrams as input. By extracting relevant features and leveraging regularization techniques, the model aims to achieve high accuracy in homology classification tasks.

### 3.5 Test Statistic

A permutation test is a non-parametric statistical method used to determine the significance of a particular statistic by comparing it against the distribution of values obtained by randomly permuting the data. The key steps involved in a permutation test are as follows:

- Calculate the Observed Statistic: Compute the statistic of interest (e.g., accuracy, mean difference) using the original data.
- Permute the Data: Randomly shuffle the labels of the data multiple times (thousands of permutations).
- Calculate the Permuted Statistics: For each permuted dataset, compute the same statistic of interest.
- Compare Observed Statistic to Permuted Distribution: Determine where the observed statistic falls within the distribution of permuted statistics to calculate the p-value.

The p-value indicates the probability of obtaining a statistic as extreme as, or more extreme than, the observed statistic purely by chance. A low p-value suggests that the observed effect is unlikely to be due to random chance.

## 4 Implementation

### 4.1 Generation of Synthetic Data

The following Python code was used to generate reference signals and embed them in noise, this specific code was made in reference from Giotto's Github, however we changed it in a way the data wouldn't change every time it runs, this by adding a seed:

```python

import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

# Fijar la semilla para reproducibilidad
np.random.seed(42)

# Par metros
sample_rate = 2048  # Frecuencia de muestreo
duration = 2.0  # Duraci n de la se al en segundos
n_samples = int(sample_rate * duration)  # N mero de muestras
n_signals = 416  # N mero de se ales
downsample_factor = 2
r_min = 0.075
r_max = 0.65
n_snr_values = 10
N = int(n_samples / downsample_factor)
Rcoef = np.linspace(r_min, r_max, n_snr_values)

# Funci n para generar una se al de onda gravitacional desde un archivo
def load_gw_signal(path_to_data, index, downsample_factor):
    gw = np.load(path_to_data / "gravitational_wave_signals.npy", allow_pickle=True)
    Norig = len(gw["data"][0])
    signal = gw["data"][index % len(gw["data"])][range(0, Norig, downsample_factor)]
    return signal

# Funci n para a adir ruido a una se al
def add_noise(signal, noise_std):
    noise = noise_std * np.random.randn(len(signal))
    return signal + noise

# Generar se ales de referencia
def generate_signals(path_to_data, n_signals, downsample_factor, n_snr_values, r_min,
    r_max):
    signals = []
    Rcoef = np.linspace(r_min, r_max, n_snr_values)
    ncoeff = [10 ** (-19) * (1 / Rcoef[i % n_snr_values]) for i in range(n_signals)]

    for i in range(n_signals):
        signal = load_gw_signal(path_to_data, i, downsample_factor)
        signals.append(signal)

    return signals, ncoeff

# Generar se ales con ruido
```

```python
def generate_noisy_signals(signals, ncoeff):
    training_data = []

    for i, signal in enumerate(signals):
        noise_std = ncoeff[i]
        sigp = int((np.random.randn() < 0))
        if sigp == 1:
            noisy_signal = add_noise(signal, noise_std)
        else:
            noisy_signal = np.random.randn(len(signal)) * noise_std
        training_data.append(noisy_signal)

    return training_data

# Ruta a los datos
path_to_data = Path("./data")  # Reemplaza con la ruta a tus datos

# Generar se ales y a adir ruido
signals, ncoeff = generate_signals(path_to_data, n_signals, downsample_factor,
    n_snr_values, r_min, r_max)
training_data = generate_noisy_signals(signals, ncoeff)

print(f"Generated {len(training_data)} signals with noise.")
```

Listing 1: Synthetic Data

## 4.2 Data Preprocessing

Now we'll show the code used to implement the methodology we wanted to preprocess the data:

```python

from sklearn.decomposition import PCA

# Funci n para realizar el embebido de ventana deslizante
def sliding_window_embedding(signal, window_size):
    n_windows = len(signal) - window_size + 1
    embedding = np.array([signal[i:i + window_size] for i in range(n_windows)])
    return embedding

# Tama o de la ventana
window_size = 200

# Realizar el embebido de ventana deslizante para todas las se ales
embedded_signals = [sliding_window_embedding(s, window_size) for s in training_data]

# Reducir dimensionalmente con PCA
pca = PCA(n_components=3)
```

```
reduced_signals = [pca.fit_transform(e) for e in embedded_signals]
```

Listing 2: Data Preprocessing

## 4.3 TDA

```

import pickle
import numpy as np
from sklearn.preprocessing import StandardScaler
import glob

# Funci n para cargar los diagramas de persistencia de un archivo
def load_persistence_diagrams(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)

# Funci n para extraer vectores de persistencia
def extract_persistence_vectors(diagrams, max_length):
    persistence_vectors = np.zeros((2, max_length))
    for dim in range(2):
        for i, (birth, death) in enumerate(diagrams[dim]):
            if i < max_length:
                persistence_vectors[dim, i] = death - birth
    return persistence_vectors.flatten()

# Funci n para determinar la longitud m xima de los vectores de persistencia
def determine_max_persistence_length(diagrams, percentile=95):
    lengths = []
    for diagram in diagrams:
        for dim in range(len(diagram)):
            lengths.append(len(diagram[dim]))
    return int(np.percentile(lengths, percentile))

# Cargar todos los diagramas de persistencia guardados
all_persistence_diagrams = []
for file in sorted(glob.glob('persistence_diagrams_block_*.pkl')):
    diagrams = load_persistence_diagrams(file)
    all_persistence_diagrams.extend(diagrams)

# Determinar el max_persistence_length basado en los diagramas de persistencia cargados
max_persistence_length = determine_max_persistence_length(all_persistence_diagrams,
    percentile=95)
print(f"Longitud m xima de persistencia determinada: {max_persistence_length}")

# Limitar manualmente la longitud m xima de persistencia si es demasiado grande
```

```
max_persistence_length = min(max_persistence_length, 500)  # Ajusta este valor seg n sea necesario

# Extraer y normalizar los vectores de persistencia
persistence_vectors = [extract_persistence_vectors(d, max_persistence_length) for d in all_persistence_diagrams]

# Filtrar valores grandes e infinitos
persistence_vectors = np.array(persistence_vectors)
persistence_vectors[np.isinf(persistence_vectors)] = np.nan
persistence_vectors = np.nan_to_num(persistence_vectors, nan=0.0, posinf=0.0, neginf=0.0)

# Normalizar los vectores de persistencia
scaler = StandardScaler()
normalized_persistence_vectors = scaler.fit_transform(persistence_vectors)

# Supongamos que las se ales crudas ya est n normalizadas y almacenadas en 'training_data'
# Normalizar las se ales crudas (si no lo est n)
normalized_signals = [scaler.fit_transform(s.reshape(-1, 1)).flatten() for s in training_data]

# Concatenar los vectores de persistencia y las se ales crudas
input_data = [np.concatenate((psv, sig)) for psv, sig in zip(normalized_persistence_vectors, normalized_signals)]

# Convertir los datos de entrada a formato numpy array
input_data = np.array(input_data)

# Crear etiquetas ficticias para la demostraci n
labels = np.array([1 if i % 2 == 0 else 0 for i in range(len(input_data))])

# Dividir en conjuntos de entrenamiento y prueba
train_size = int(0.8 * len(input_data))
x_train, x_test = input_data[:train_size], input_data[train_size:]
y_train, y_test = labels[:train_size], labels[train_size:]

# Redimensionar los datos para la entrada de la CNN
x_train = x_train[..., np.newaxis]
x_test = x_test[..., np.newaxis]
```

Listing 3: Diagrams Code

## 4.4 CNN

With the diagrams received by the last code we needed it to be in zip so we could access it anytime, but for the code now we needed it to be extracted

```

from tensorflow.keras.optimizers import Adam

# Definir el modelo
model = Sequential([
    Conv1D(filters=64, kernel_size=16, strides=1, activation='relu',
        input_shape=(x_train.shape[1], 1)),
    MaxPooling1D(pool_size=4, strides=4),
    Conv1D(filters=128, kernel_size=16, strides=1, activation='relu'),
    MaxPooling1D(pool_size=4, strides=4),
    Conv1D(filters=256, kernel_size=16, strides=1, activation='relu'),
    MaxPooling1D(pool_size=4, strides=4),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')  # Cambiar la activaci n a sigmoid para
        clasificaci n binaria
])

# Compilar el modelo con una tasa de aprendizaje m s baja
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy',
    metrics=['accuracy'])  # Cambiar la p rdida a binary_crossentropy

# Resumen del modelo
model.summary()

# Entrenar el modelo con m s  pocas
history = model.fit(x_train, y_train, epochs=30, batch_size=32,
    validation_data=(x_test, y_test))

# Evaluar el modelo
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")
```

Listing 4: Extract Diagrams

With the diagrams in our hands now we needed to calculate persistence entropy and finally create the CNN.

```


```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical


# Function to load diagrams from JSON files
def load_diagrams_from_json(file_paths):
    diagrams = []
    for file_path in file_paths:
        with open(file_path, 'r') as f:
            diag = json.load(f)
            h0 = np.array(diag[0])
            h1 = np.array(diag[1])
            diagrams.append((h0, h1))
    return diagrams

# Load the diagrams
diagram_file_paths = [os.path.join(extracted_path, file) for file in extracted_files if
    file.endswith('.json')]
diagrams = load_diagrams_from_json(diagram_file_paths)

print(f"Loaded {len(diagrams)} diagrams.")

# Function to calculate persistence entropy
def persistence_entropy(diagram):
    if len(diagram) == 0:
        return 0
    persistence = diagram[:, 1] - diagram[:, 0]
    persistence = persistence / np.sum(persistence)
    entropy = -np.sum(persistence * np.log(persistence))
    return entropy

# Calculate the persistence entropy for H0 and H1
entropies_h0 = [persistence_entropy(h0) for h0, h1 in diagrams]
entropies_h1 = [persistence_entropy(h1) for h0, h1 in diagrams]



# Convert the list of entropies to a numpy array
X = np.array(list(zip(entropies_h0, entropies_h1))).reshape(-1, 2, 1)

# Dummy labels (replace with actual labels if available)
y = np.random.randint(0, 2, len(entropies_h0))

```

```
# Preprocess the data
scaler = StandardScaler()
X = scaler.fit_transform(X.reshape(-1, 2)).reshape(-1, 2, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Convert labels to categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Define the CNN model
model = Sequential()
model.add(Conv1D(64, kernel_size=1, activation='relu', input_shape=(2, 1)))
model.add(BatchNormalization())
model.add(Conv1D(64, kernel_size=1, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

# Compile the model with a lower learning rate
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
    metrics=['accuracy'])

# Train the model with more epochs and a larger batch size
model.fit(X_train, y_train, epochs=100, batch_size=64, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy * 100:.2f}%")
```

Listing 5: Persistence Entropy CNN

## 4.5 Mapper Classification

Mapper is a Topological Data Analysis (TDA) tool used for dimensionality reduction and data visualization. This section describes how Mapper is used to classify data in the context of the project.

This technique provides a simplified representation of a dataset by mapping it to a lower-dimensional space. The dataset is divided into clusters, which are then represented as nodes in a graph. Adjacent clusters

are connected by edges if they share data points.

**Using Mapper for Classification**

To classify data using Mapper, we follow these steps:

1. **Select a filter function**: This function is used to project the data into a lower-dimensional space.
2. **Cover the range of the filter function**: The range of the filter function is divided into overlapping intervals.
3. **Apply clustering within each interval**: Within each interval, a clustering algorithm is applied to group the data.
4. **Construct the simplicial complex**: The resulting clusters are represented as nodes and connected with edges if they share data points.

# 5 Results

In this section, we present the results obtained so far in the project on gravitational wave detection using topological data analysis (TDA) and convolutional neural networks (CNN).

## 5.1 Synthetic Data Obtained

The first step in our analysis involved generating synthetic gravitational wave signals. These signals were created to mimic the characteristics of actual gravitational waves detected by LIGO and Virgo. The synthetic data serves as a controlled environment to test and refine our methods before applying them to real-world data.

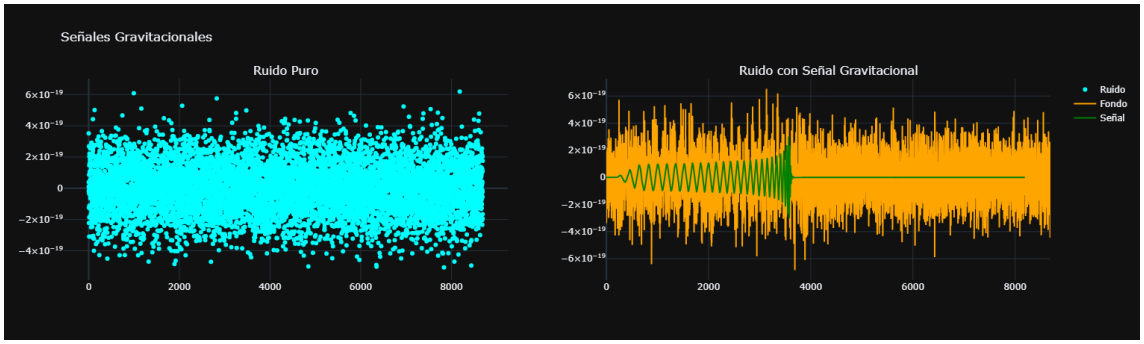


Figure 1: Pure noise (left) and noise with gravitational signal (right)

These synthetic signals, are crucial for developing and validating our TDA and CNN methods.

## 5.2 Visualization of Sliding Window Embedding

Below, we show the embeddings images obtained from the sliding window technique and dimensionality reduction using Principal Component Analysis (PCA).

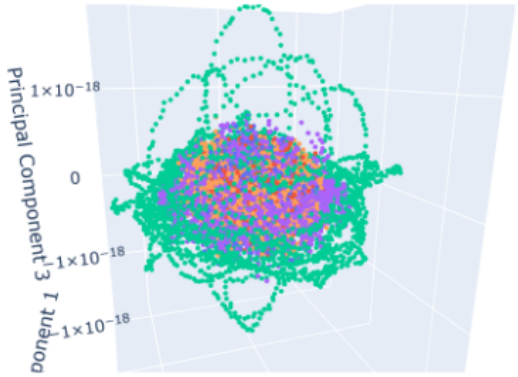


Figure 2: Visualization of the sliding window embedding in reduced dimensionality

The image above shows the visualization of the sliding window embedding in reduced dimensionality. This visualization helps in understanding the structure of the data after preprocessing.

## 5.3 Persistence Diagram

To further analyze the topological features of the gravitational wave signals, we generated a persistence diagram using the TDA techniques.

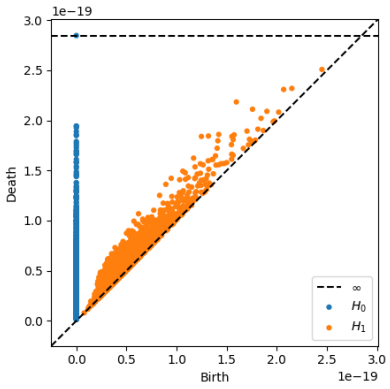


Figure 3: Persistence diagram showing the birth and death of topological features. Blue points ($H_0$) represent connected components, and orange points ($H_1$) represent loops or cycles.

The persistence diagram allows us to visualize the significant topological features of the data. In this diagram: - The x-axis (Birth) represents the parameter value at which a topological feature appears. - The y-axis (Death) represents the parameter value at which the feature disappears. - Points further from the

diagonal (dashed line) indicate features with longer lifespans, which are considered more significant.

## 5.4 Convolutional Neural Network Results

To classify the gravitational wave signals, we implemented a convolutional neural network (CNN). The training process and results are shown in Figure.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.82 | 0.86 | 0.84 | 42 |
| 1 | 0.85 | 0.81 | 0.83 | 42 |
| accuracy | | | 0.83 | 84 |
| macro avg | 0.83 | 0.83 | 0.83 | 84 |
| weighted avg | 0.83 | 0.83 | 0.83 | 84 |

Figure 4: CNN results

The final test accuracy of the CNN was 0.83, this accuracy indicates that the model has the ability to generalize to unseen data, although improvements are necessary to enhance its performance and stability during training.

## 5.5 Statistical Significance Testing

To evaluate the statistical significance of our CNN model's predictions, we performed a permutation test. This test helps determine whether the observed prediction accuracy could have occurred by chance.

```
from netneurotools.stats import permtest_rel

# Perform the permutation test
pval, null_dist = permtest_rel(y_pred, y_test, n_perm=10000)

print(f'Valor p: {pval}')
print(f'Distribuci n nula: {null_dist}')
```

```
Valor p: 0.023809523809523808
Distribución nula: 0.7997200279972003
```

Figure 5: P-value results

The permutation test yielded a p-value of 0.02, indicating that there is a probability that the observed prediction accuracy could have occurred by chance.

Since the p-value is less than the conventional significance level of 0.05, we have enough evidence to reject the null hypothesis. This suggests that the model's performance is significantly better than what could be expected by chance, indicating that the model is likely capturing meaningful patterns in the data.

# 6 Conclusions and Recommendations

## 6.1 Summary of Findings

The study aimed to develop a convolutional neural network (CNN) model to classify gravitational wave signals using persistence diagrams derived from sliding window embeddings. The model was trained on synthetic gravitational wave signals with added noise to test its ability to distinguish between different classes of signals. After several iterations of model training and adjustments, the final model achieved a test accuracy of 83%.

Key findings include:

- The use of persistence diagrams and sliding window embeddings to preprocess the signals provided a novel approach to feature extraction in the context of gravitational wave signal analysis.
- The CNN model demonstrated a reasonable ability to classify the signals, achieving an accuracy of 83%, indicating that the model was able to learn and generalize from the training data to some extent.

## 6.2 Limitations of the Study

Several limitations were identified during the study:

- **Data Generation:** The synthetic data generation process, while ensuring reproducibility, may not fully capture the complexity and variability of real gravitational wave signals. This limits the applicability of the model to real-world scenarios.
- **Model Complexity:** The model's architecture, despite adjustments, may still be insufficiently complex to capture all the nuances of the data, leading to a plateau in accuracy.
- **Computational Resources:** The training process was computationally intensive, and interruptions during long training sessions posed challenges. This may have affected the model's performance due to incomplete training cycles.
- **Limited Feature Engineering:** The features extracted from persistence diagrams were limited to basic differences between birth and death times. More sophisticated feature engineering techniques could potentially improve model performance.

## 6.3 Suggestions for Future Work

Based on the results obtained and the limitations encountered, the following recommendations are made for future research:

- **Enhanced Data Generation:** Incorporate more realistic simulations of gravitational wave signals, including a wider variety of noise and interference patterns to better mimic real-world conditions.
- **Model Architecture Exploration:** Experiment with more advanced neural network architectures, such as deeper CNNs, recurrent neural networks (RNNs), or hybrid models combining CNNs and RNNs, to better capture temporal dependencies and complex patterns in the data.
- **Feature Engineering:** Explore advanced feature extraction techniques from persistence diagrams, such as using persistence images, landscapes, or other topological summaries that may capture more informative aspects of the data.
- **Transfer Learning:** Investigate the application of transfer learning, using pre-trained models on similar tasks to improve model performance and reduce training time.
- **Regularization Techniques:** Apply regularization techniques such as dropout, batch normalization, and data augmentation to improve model generalization and prevent overfitting.
- **Robust Training Pipelines:** Develop more robust training pipelines that can handle interruptions and continue training from checkpoints, ensuring that the model training process is not disrupted and can be completed effectively.

By addressing these areas, future work can build upon the findings of this study to develop more accurate and robust models for the classification of gravitational wave signals.

# 7

# References


[1] NASA. (2020). *What is a gravitational wave?*. NASA Space Place. Retrieved from https://spaceplace.nasa.gov/gravitational-waves/en/

[2] Sentinel Mission. (2024). *Gravitational Wave - Definition & Detailed Explanation*. Sentinel Mission. Retrieved from https://sentinelmission.org/astronomical-phenomena-glossary/gravitational-wave/

[3] LIGO Caltech. (2024). *What are gravitational waves?*. LIGO Lab — Caltech. Retrieved from https://www.ligo.caltech.edu/page/what-are-gw

[4] Encyclopaedia Britannica. (2024). *Gravitational wave — Detectors, Discovery, & Speed*. Britannica. Retrieved from https://www.britannica.com/science/gravitational-wave

[5] ArXiv. (2024). *Sliding Windows and Persistence: An Application of Topological Methods to Signal Analysis*. Retrieved from https://arxiv.org/abs/1307.6188

[6] Wikipedia. (2024). *Singular value decomposition*. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/Singular_value_decomposition

[7] ArXiv. (2024). *Introduction to Persistent Homology*. Retrieved from https://arxiv.org/abs/2209.05134

[8] EPJ Data Science. (2024). *A roadmap for the computation of persistent homology*. Retrieved from https://epjdatascience.springeropen.com/articles/10.1140/epjds/s13688-019-0213-7

[9] Applied Network Science. (2024). *Persistence homology of networks: methods and applications*. Retrieved from https://appliednetsci.springeropen.com/articles/10.1007/s41109-020-00297-1

[10] DataCamp. (n.d.). *An Introduction to Convolutional Neural Networks: A Comprehensive Guide to CNNs in Deep Learning*. Retrieved from https://www.datacamp.com

[11] IBM. (2024). *What are Convolutional Neural Networks?*. Retrieved from https://www.ibm.com

[12] Liu, J., Chen, S., Wu, L., Zhang, R., Chen, Z., Guo, L., Li, H. (2023). *Structural insights into small molecule SARM1 inhibitors with in vivo neuroprotective efficacy*. Nature Communications, 14, 3191. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10542398/

[13] Tsalera, E., Papadakis, A., Samarakou, M. (2021). *Comparison of Pre-Trained CNNs for Audio Classification Using Transfer Learning*. JSAN, 10(4), 72. Retrieved from https://www.mdpi.com