Optimization in Gravitational Wave Detection Integrating Fourier Series, Topological Analysis and CNN

ÁNGEL AZAHEL RAMÍREZ CABELLO ⋆
ANNETTE PAMELA RUIZ ABREU †
AVRIL MICHELLE RUIZ MARTÍNEZ ‡
FRANCO MENDOZA MURAIRA §
JORGE RAÚL ROCHA LÓPEZ ◇
AND
LUIS ANGEL LÓPEZ CHÁVEZ △
IDM, Monterrey Institute of Technology and Higher Education, Av. Eugenio Garza Sada 2501 Sur, 64849, Nuevo León, México
⋆Corresponding author: A01383328@tec.mx †Corresponding author: A01423595@tec.mx
‡Corresponding author: A00833018@tec.mx §Corresponding author: A01383399@tec.mx
◇Corresponding author: A01740816@tec.mx △Corresponding author: A01571000@tec.mx

[07 June 2024]

Gravitational wave detection and classification represent pivotal endeavors in modern astrophysics, offering unprecedented insights into the dynamics of the universe. In this paper, we propose a novel approach leveraging Principal Component Analysis (PCA) of the Fourier discrete transform of signals and topological persistence images to enhance the accuracy and robustness of Convolutional Neural Network (CNN) models for gravitational wave classification. Through rigorous experimentation and analysis, we demonstrate the efficacy of our integrated approach, achieving an average classification accuracy of 0.8 across a range of signal-to-noise ratios (R). This significant improvement surpasses the limitations of conventional models trained solely on raw signal data, which often yield accuracy levels akin to random guessing. Our findings underscore the transformative potential of combining advanced preprocessing techniques, enabling more reliable and accurate models for gravitational wave detection, facilitating the discovery and characterization of celestial phenomena with unprecedented precision and reliability.

Keywords: CNN; fourier transform; gravitational waves; persistence images; XGBoost

MSC Codes

Primary: 54H30 (Topology to computer science)
Secondary: 68T07 (Neural networks), 83C35 (Gravitational waves), 68U03 (Digital topology)

1. Introduction

The detection of gravitational waves represents one of the most sophisticated and crucial challenges in modern physics. This phenomenon, whose existence was experimentally confirmed by the LIGO collaboration and honored with the Nobel Prize in Physics in 2017 (Thorne, 2018), has opened a new window for observing the universe, allowing the study of large-scale cosmic events such as black hole mergers. Gravitational waves, first theorized by Albert Einstein in his theory of general relativity a century ago, represent ripples in the fabric of space-time caused by the most violent and energetic

events in the cosmos (Rowan and Hough, 1999). The detection of these waves unveils a treasure trove of information about celestial phenomena such as black hole mergers, neutron star collisions, and other cataclysmic occurrences. By deciphering the gravitational wave signals, scientists can probe the deepest mysteries of the universe, including the nature of gravity itself and the elusive components of dark matter and dark energy. However, the precise identification of these signals is extremely challenging due to the low signal-to-noise ratio and the high presence of noise in the collected data.

Previous research in gravitational wave detection has predominantly focused on utilizing Convolutional Neural Networks (CNNs) as well as traditional signal processing techniques. Notably, Chan et al., 2020 successfully integrated CNNs with conventional methods to detect gravitational waves, showcasing the efficacy of machine learning in this field. Similarly, Li et al., 2020 demonstrated the capability of CNNs to classify gravitational wave signals amidst noisy data, highlighting their utility in real-world scenarios. Despite the advancements achieved through CNN-based approaches, limitations persist in capturing the intricate topological structures inherent in gravitational wave data. This gap has motivated researchers, such as George and Huerta, 2018, to explore complementary techniques such as Topological Data Analysis (TDA). In addition to TDA and CNNs, Fourier series analysis has also been employed in gravitational wave detection. For instance, Cornish, 2020 utilized Fourier series to decompose gravitational wave signals into their frequency components, enabling the identification of characteristic signatures. This approach offers a mathematical framework to analyze periodic signals, complementing the spatial feature extraction capabilities of CNNs and the topological insights provided by TDA.

Motivated by these developments, our study explores the integration of Fourier series analysis alongside TDA and CNNs for gravitational wave detection. By leveraging the strengths of these diverse methodologies, we aim to enhance the precision and efficacy of gravitational wave detection algorithms and reduce training time, ultimately advancing our understanding of the cosmos. The advent of gravitational-wave astronomy opens avenues for testing fundamental aspects of general relativity and exploring exotic physics, such as the nature of compact objects and the existence of extra dimensions.

The primary objective of our study is to develop a highly accurate CNN for detecting gravitational waves amidst high background noise, while optimizing its learning capacity and reducing training time through parameter optimization, including Takens embedding configurations. We will use topological descriptors like persistence diagrams and barcodes to capture essential features of gravitational wave signals and apply persistent homology to distinguish between noise and the actual chirp signal from black hole collisions. Additionally, we will implement interpretability techniques (Grad-CAM, SHAP, LIME) to understand the CNN's decision-making process and evaluate various neural network architectures to find the most effective model for classifying gravitational waves.

2. Theoretical Fundamentals

In this section, we delve into the theoretical underpinnings crucial for understanding the methodologies employed in our study.

2.1. *Gravitational waves*

Gravitational waves, first predicted by Albert Einstein in his theory of general relativity, are ripples in the fabric of spacetime caused by the acceleration of massive objects. These waves carry energy across the universe, offering a new window into the cosmos. Gravitational waves are analogous to ripples on a pond, propagating outward from their source at the speed of light. Unlike electromagnetic waves, which

arise from accelerating charges, gravitational waves emerge from the acceleration of mass distributions. This distinction makes them unique messengers of cosmic phenomena, capable of revealing insights into the dynamics of massive astrophysical systems (Cai et al., 2017).

The detection of gravitational waves presents a formidable challenge due to their exceedingly weak interactions with matter. Researchers outline two primary methods employed for their detection: interferometry and pulsar timing (Pitkin et al., 2011). Interferometric detectors, such as LIGO (Laser Interferometer Gravitational-Wave Observatory) and Virgo, utilize laser interferometry to measure minuscule changes in the length of perpendicular arms caused by passing gravitational waves. By comparing the phase shifts of laser beams, these detectors can discern the characteristic signature of gravitational wave signals. Pulsar timing arrays, on the other hand, exploit the precise timing of pulsar signals to infer the presence of gravitational waves. As gravitational waves pass through the pulsar-Earth system, they induce minute variations in the arrival times of pulsar signals. By monitoring an array of pulsars distributed across the sky, researchers can detect correlations indicative of gravitational wave interference. The detection of gravitational waves heralds a new era of observational astronomy, offering unprecedented insights into the most violent and energetic phenomena in the universe. From the mergers of black holes and neutron stars to the inflationary epoch of the early universe, gravitational waves provide a direct probe into phenomena beyond the reach of traditional telescopes.

2.2. *Fourier series*

Fourier series is a mathematical technique used to represent periodic functions as a sum of sinusoidal functions (sine and cosine) of different frequencies. It is named after the French mathematician Joseph Fourier, who first introduced the concept in the early 19th century.

The Fourier series representation of a periodic function $f(x)$ with period T is given by:

$$f(x) = a_0 + \sum_{n=1}^{\infty} (a_n \cos(nx) + b_n \sin(nx))$$

where:

- a_0 is the average value of the function over one period,
- a_n and b_n are the Fourier coefficients, which represent the amplitudes of the cosine and sine terms, respectively, at integer multiples of the fundamental frequency $\frac{2\pi}{T}$.

The Fourier coefficients are calculated using the following formulas:

$$a_0 = \frac{1}{T} \int_0^T f(x)\, dx$$

$$a_n = \frac{2}{T} \int_0^T f(x) \cos(nx)\, dx$$

$$b_n = \frac{2}{T} \int_0^T f(x) \sin(nx)\, dx$$

The Fourier series expansion allows us to approximate a wide range of functions, including periodic and non-periodic ones, by representing them as combinations of sinusoidal waves. This representation is particularly useful in signal processing, where it enables the analysis and manipulation of signals in the frequency domain (Wolfram Research, n.d.). In the context of gravitational wave detection, Fourier series analysis can be employed to decompose gravitational wave signals into their frequency components, facilitating the identification of characteristic signatures associated with astrophysical

events. By analyzing the frequency content of gravitational wave data, researchers can extract valuable information about the sources and properties of these waves.

2.3. *Convolutional neural networks*

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid data, such as images or time series. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers, that learn hierarchical representations of the input data. Convolutional Neural Networks (CNNs) are structured architectures composed of interconnected layers designed to extract hierarchical representations from input data. At the core of CNNs are convolutional layers, which perform convolution operations utilizing learnable filters or kernels. These operations extract features such as edges, textures, or temporal patterns from the input data while preserving spatial or temporal relationships. Following convolutional layers, pooling layers downsample the feature maps, reducing spatial or temporal dimensions while retaining salient features. Techniques like max pooling or average pooling are commonly employed for this purpose. Subsequently, fully connected layers connect every neuron in one layer to every neuron in the next, facilitating the learning of complex, nonlinear relationships within the data. Positioned after the convolutional and pooling layers, these fully connected layers are instrumental in high-level feature learning and classification. This cohesive architecture enables CNNs to automatically extract spatial and temporal features from raw data and learn intricate representations of complex patterns within the signals (IBM 2021).

Convolutional Neural Networks (CNNs) are highly effective in image processing tasks, particularly in identifying gravitational wave signals through persistence images. Their proficiency stems from their innate capacity to extract intricate spatial patterns and hierarchical representations from images. As persistence images encode multi-scale features representing signal persistence over time, CNNs hierarchically learn these features across successive convolutional layers, capturing both local and global characteristics of the signal. Leveraging spatial contextual understanding, CNNs exploit spatial relationships within persistence images to discern patterns relevant to wave persistence, thus differentiating between genuine signals and noise or artifacts. Moreover, CNNs demonstrate robustness to variability in signal morphology, frequency content, and noise levels, enabling adaptive feature learning for accurate signal identification (spectric-labs, 2023).

2.4. *Decision Trees*

A decision tree is a supervised machine learning algorithm used for both classification and regression inference (De Ville, 2013). The tree structure consists of nodes and branches, where each internal node represents a decision on an attribute, each branch represents the outcome of the test, and each leaf node represents the output of the inference, either being a label for classification tasks or a continuous value for regression tasks.

The top node of the tree that represents the entire dataset and is divided into subsets is called the root node, the branches that connect each of the nodes as edges represent a binary decision. For the decision tree to be created it has as a main objective to maximize the information gain as it represents the reduction of entropy by establishing a certain decision to be made. The entropy of a dataset S is given by the following equation, where it measures the *impurity or disorder* in the dataset with p_i being the proportions of each label respectively:

$$E(S) = \sum_{i=1}^{C} -p_i \log_2(p_i)$$

For binary classification, the output of a model consists of a probability given to an object of being a member of the positive class. By default, the threshold p of assignation to the positive class is 0.5. However, this threshold can be modified to a value of between $p \in (0,1)$

2.5. *XGBoost*

Extreme Gradient Boosting (Chen and Guestrin, 2016), more commonly known as XGBoost, is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable for the application of Gradient Boosting algorithms. This model has the capabilities of making regression inferences, and binary and multi-class classification, making it versatile for many scenarios. Although the framework is known for applying gradient-boosting trees, it also supports gradient-boosting linear regressions.

The underlying algorithm mainly consists of building decision trees sequentially (Chen and Guestrin, 2016), each new tree correcting errors made by the previous ones. It uses the gradient of the loss function to make the corrections. Shrinkage is also commonly applied as a form of regularization that reduces the influence of each individual tree by scaling down the weights of leaf nodes controlled by a learning rate, one of its main parameters.

Other parameters of this implementation gradient boosting mainly consist on the characteristics of each of the trees to be built to make the ensemble. This include the number of trees to be built (also called estimators), the maximum depth of each of them commonly being less than ten, and the number of features and training data to be taken into account for each one (Chen and Guestrin, 2016).

Another important parameter of gradient-boosting trees is *gamma* (γ), which indicates the minimum loss reduction threshold needed to split a leaf node, meaning that a higher threshold will imply shorter decision trees preventing possible overfitting. As for most models, there can be regularization methods most commonly known as L1 and L2 regularization, which are added to the loss function of the model scaled by λ and α respectively. In the case of binary classification, the log loss function is shown below:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}[-y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)] + \lambda L_1 + \alpha L_2$$

$$L_1 = \frac{1}{n}\sum_{j=1}^{m} ||\theta_j||$$

$$L_2 = \frac{1}{n}\sum_{j=1}^{m} (\theta_j)^2$$

In this regularization method, the features of the decision trees may be *punished* depending on the values of λ and α. By increasing λ, the L1 regularization value will increase and as a result, some features will be reduced to zero eliminating features that are not relevant enough for the model's output. By increasing α, the L2 regularization value will increase and as a result, some features will be reduced to near zero but will not be eliminated. With these terms, the following boosting trees will aim to decrease the loss function taking into account the predicted output $\hat{y}_i$ during the validation metric.

2.6. *TDA*

Topological Data Analysis (TDA) is an emerging field of data analysis that applies techniques from topology, the mathematical study of shapes and their properties, to extract meaningful information from complex datasets. TDA provides a robust framework for identifying, characterizing, and visualizing the underlying topological structure of data, offering insights that are often not apparent through traditional data analysis methods.

At the core of TDA is the concept of using topological invariants to describe the shape of data. These invariants are properties that remain unchanged under continuous deformations, such as stretching or bending. One of the primary tools in TDA is persistent homology, which allows for the computation of topological features across multiple scales. By leveraging TDA, specifically persistent homology, we can gain a deeper understanding of the topological structures present in high-dimensional datasets. This approach complements traditional methods such as Fourier analysis and machine learning models like CNNs, offering a more holistic view of the data and enhancing our ability to identify and interpret complex patterns (Chazal and Michel, 2021).

2.6.1. Persistent homology

Persistent homology is an advanced mathematical framework originating from algebraic topology, used to analyze the topological features of complex datasets. This technique examines how homology groups evolve across multiple scales, allowing researchers to identify and characterize persistent topological structures. Its application is particularly significant in high-dimensional datasets, including those derived from gravitational wave signals. Persistent homology extends traditional homology by considering the persistence of topological features over a range of spatial resolutions. It provides a multiscale description of the dataset's topological structure, identifying features that are robust to changes in scale. Persistent homology tracks the birth and death of homological features (such as connected components, loops, and voids) throughout the filtration. This information is summarized in a persistence diagram or a barcode, where each feature is represented by a point or a line segment corresponding to its lifespan (birth and death scale). Persistent homology can distinguish between significant topological features and those arising from noise, making it a powerful tool for analyzing noisy datasets (Aktas et al., 2019).

2.6.2. Sliding window embedding

Sliding window embedding involves partitioning the time series into overlapping sub-sequences, or windows, of a fixed length. These windows slide across the data sequence with a specified stride, creating overlapping segments that capture local temporal patterns. This method enhances the model's ability to learn from the data by maintaining temporal coherence and context within each window (Perea and Harer, 2015).

- **Window Size and Stride:** The window size, denoted as W, determines the length of each sub-sequence, while the stride, S, defines the step size by which the window moves along the time series. A smaller stride results in more overlap between consecutive windows, capturing finer temporal details.
- **Overlapping Windows:** Overlapping is crucial as it ensures that each data point is considered multiple times within different temporal contexts, enhancing the model's capacity to recognize patterns that span across adjacent windows. This overlap can be mathematically expressed as:

$$\text{Number of windows} = \left\lfloor \frac{N-W}{S} \right\rfloor + 1$$

where N is the length of the time series.

- **Embedding Transformation:** Each window serves as an independent input instance for the CNN, allowing the network to learn features specific to the local segment of the time series. This approach is particularly beneficial for detecting local anomalies or events within the data.

Sliding window embedding significantly enhances gravitational wave detection by allowing convolutional neural networks (CNNs) to focus on local features within the data, which is crucial for identifying the short, transient signals characteristic of gravitational waves. This technique maintains temporal dependencies within the data through overlapping windows, ensuring that the sequential nature of the waveforms is preserved. Additionally, by increasing the effective size of the training dataset and improving the signal-to-noise ratio, sliding window embedding helps the CNN distinguish true gravitational wave signals from noise. This method can be applied in various stages of gravitational wave detection, including preprocessing raw time series data, extracting temporal features, and classifying segments of data as either containing gravitational waves or being noise, thereby enhancing the overall accuracy and robustness of the detection process.

2.6.3. Singular value decomposition

Singular Value Decomposition (SVD) is a linear algebra technique that decomposes a matrix into three fundamental components, providing a comprehensive summary of the matrix's structure and facilitating various data analysis tasks.

SVD decomposes an $m \times n$ matrix $\mathbf{A}$ into three matrices:

$$\mathbf{A} = \mathbf{UEV}^T$$

where:

- $\mathbf{U}$ is an $m \times m$ orthogonal matrix containing the left singular vectors.
- $\mathbf{E}$ is an $m \times n$ diagonal matrix with non-negative real numbers on the diagonal, known as the singular values.
- $\mathbf{V}$ is an $n \times n$ orthogonal matrix containing the right singular vectors.

The singular values in $\mathbf{E}$ are ordered in descending magnitude, and they provide a measure of the significance of each corresponding singular vector pair.

SVD helps identify dominant modes in time-series data, facilitating the detection of underlying patterns and structures within signals. This is particularly useful in fields like seismology and acoustics, where identifying such modes is critical. By decomposing a signal matrix and truncating the less significant singular values, SVD effectively reduces noise, enhancing the clarity and interpretability of the signal. Therefore, SVD is employed to filter noise from gravitational wave signals, isolating the significant components that correspond to astrophysical events.

2.6.4. Principal Component Analysis

Principal Component Analysis (PCA) is a linear transformation technique that transforms a set of correlated variables into a set of uncorrelated variables called principal components. These principal components capture the directions of maximum variance in the data, facilitating dimensionality reduction and feature extraction.

PCA begins by computing the covariance matrix of the data. Given a dataset with n observations and p variables, the covariance matrix $\mathbf{C}$ is a $p \times p$ matrix that captures the variance and covariance between the variables.

$$\mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

where $\mathbf{x}_i$ is the i-th observation and μ is the mean vector of the dataset.

The next step involves performing eigenvalue decomposition on the covariance matrix $\mathbf{C}$. This yields eigenvalues and corresponding eigenvectors.

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v}$$

where λ is an eigenvalue and $\mathbf{v}$ is the corresponding eigenvector.

The eigenvectors of the covariance matrix form the principal components. The eigenvectors corresponding to the largest eigenvalues represent the directions of maximum variance in the data. The data is then projected onto these principal components, resulting in a transformed dataset with reduced dimensions (IBM 2023).

Gravitational wave signals can be represented as high-dimensional time-series data. By applying PCA, the dataset can be reduced to a lower-dimensional space while preserving the most significant features. This reduction not only simplifies the analysis but also helps in identifying the dominant modes of variability in the data.

2.6.5. SHAP values

SHAP (SHapley Additive exPlanations) values are used to explain the output of machine learning models. This method is based on the concept of Shapley values from cooperative game theory, which are used to fairly distribute the ”payout” among the ”players” based on their contributions to the total payout. In the context of machine learning, SHAP values distribute the prediction of a model among the input features, attributing the contribution of each feature to the final prediction (Trevisan, 2022).

SHAP values have several key properties that make them particularly useful for model interpretation. One such property is additivity, which ensures that the explanation model is additive, meaning the sum of the contributions of all features equals the prediction of the model for a specific instance.

Additionally, SHAP values provide local accuracy, meaning they offer local explanations for individual predictions. The sum of the SHAP values for all features, plus the model's baseline expectation, equals the model's output for a given instance.

The SHAP value is calculated as a weighted average of these marginal contributions across all subsets. Mathematically, the SHAP value for a feature i is given by:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [f(S \cup \{i\}) - f(S)]$$

where ϕ_i is the SHAP value for feature i, N is the set of all features, S is a subset of N not containing i, and $f(S)$ is the model prediction for the subset S.

A high positive SHAP value for a feature indicates that the feature strongly pushes the prediction towards a higher value. In other words, this feature significantly contributes to increasing the model's output for that instance. Conversely, a high negative SHAP value signifies that the feature strongly pushes the prediction towards a lower value, meaning it has a substantial negative impact on the model's

output. Essentially, positive SHAP values indicate features that increase the predicted value, while negative SHAP values indicate features that decrease it.

3. Methodology

3.1. *Data generation*

To generate the signals, it was based on the code used in Bresten and Jung's paper (Li et al., 2020), where gravitational waves were generated from non-spinning binary black hole mergers. A noise with a probability of 0.5 in random time is incorporated to the signal. The result of that is a time series which has the following form:

$$s = g + \varepsilon \frac{1}{R} \xi$$

Where g represents the signal of a gravitational wave of the reference set (the data generated without noise), *xi* represents the Gaussian noise, ε is a constant epsilon = 10^{-19} which scales the amplitude of the noise to the signal and the R coefficient $\in (0.075, 0.65)$, is a parameter that controls how much noise there is in the generated measurements (the signal-to-noise ratio). With a higher value of R, the signals will have less noise and vice versa. To generate the data, 1500 signals with different R values were generated and the measurements and their corresponding labels were saved. The different values of R for creating the models were $R = \{0.065, 0.15, 0.2, 0.25, 0.3, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65\}$.

3.2. *Data preprocessing*

In the exploratory data analysis process, two distinct behaviors were identified in the Fourier series of the signals under study. When examining the magnitudes of the Fourier transforms, it was observed that some signals exhibited significant peaks at frequencies near zero, while others showed uniform magnitudes across all frequencies. To assess the usefulness of the Fourier series magnitudes in distinguishing between waves and noise, a topological mapping algorithm (mapper) was implemented. This involved reducing the dimensions of the Fourier series magnitudes using PCA and creating 50 components. This approach allowed for visualizing the data distribution in clusters and evaluating its ability to discriminate between wave characteristics and noise. The results, depicted in figure 1, revealed that the signals could be grouped into clusters; however, to achieve a precision of 94%, a minimum of three clusters was required. Specifically, two clusters associated with waves and one with noise were identified. It is important to note that this differential behavior in the magnitudes of the Fourier series only manifested when the correlation coefficient (R) exceeded the threshold of 0.15. This finding indicates that as the presence of noise increases, the magnitudes of the Fourier series become less effective in distinguishing between waves and noise.

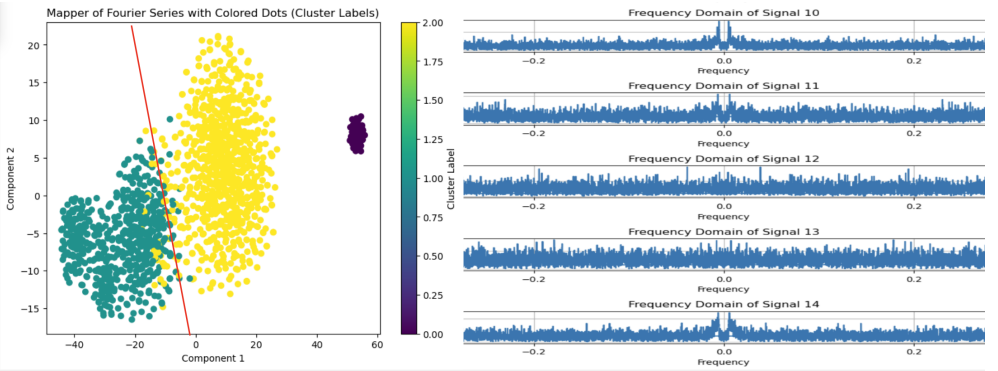

FIG. 1. Mapper of the 50 PCA components of the magnitudes of the Fourier series of each signal.

To prepare the data for analysis and training, we employed several preprocessing techniques aimed at enhancing the data's suitability for subsequent modeling. Firstly, the raw signals underwent transformation using the discrete Fourier transform (DFT) to convert them into Fourier series representations. This transformation enabled the conversion of the signals from the time domain to the frequency domain, thereby revealing their frequency components and spectral characteristics. Subsequently, to reduce the dimensionality of the Fourier series data and extract its essential features, we applied Principal Component Analysis (PCA). Specifically, we applied PCA only to the magnitudes of the Fourier series within the frequency range from -0.05 to 0.05. By decomposing the data into its principal components, PCA facilitated the extraction of the most salient patterns and variations present in the Fourier series while discarding redundant or noise-dominated components. We retained the top 50 principal components, which encapsulated the majority of the variance within the data. This phenomenon suggests the need to explore other distinctive features to improve the discrimination capacity between the signals of interest and noise under conditions of greater interference.

3.3. *TDA application*

After the data preprocessing it was evident that the signals had critical differences wether it was a gravitational wave or pure noise, so the methodology proposed by Perea and Harer, 2015, that suggests an outstanding way to classify signals using the persistent vectors of each signal, by describing the topological form of each wave and finding similarities between groups, may find key variables to solve the gravitational waves problem using machine learning.

To start with this process it is necessary to apply the sliding window embedding to each signal in order to truly find their persistent homology, to achieve this goal it is possible to use the giotto-tda library in python (Tauzin et al., 2021), which returns the global optimal parameters (time delay and dimension) of the embedding. After applying this algorithm to many signals in various noise levels it was found out that the process takes too long and the parameters are quite different in each case, so in order to save resources the NoLiTSA python library was applied, which is useful to calculate the local optimal parameters of the embedding and with this tool,that executes much faster, it was revealed that almost all of the signals at various noise levels have an optimal time delay of 2 and a dimension of 4, which can be due to the fact that the data is manufactured and not actual signal recordings.

So after applying the sliding window embedding to each signal on the dataset with giotto-tda, it was possible to calculate the persistent vectors of each signal, using the ripser python library (Bauer, 2021),

which is just a translation of a C++ algorithm of the Vietoris–Rips complex, the visual representation of this is a persistence diagram (figure 2) , for the context of this research only the β_0 and β_1 homologies were used because of limited time and computational power, nevertheless it was found out that each vector for the 1500 signals had a different length so this variables could not be used to feed a machine learning algorithm without a padding what could affect the accuracy of the model.

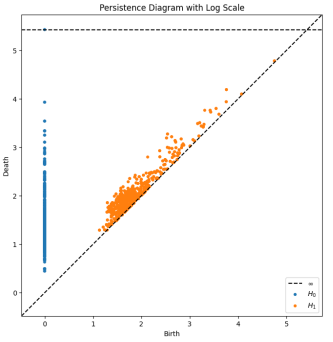

FIG. 2. Persistence diagram of a gravitational wave signal with noise

To assure the same length of predictor variables for a machine learning model it is possible to use the persistent images instead as proposed by (Deng and Duzhin, 2022), to address this problem the **PersistenceImager** function from the Persim python library was used, which generates a matrix with colors depending on the density of points in the persistence diagram of a simplicial complex, with this tool it is possible to generate a vector of consistent length with the persistence homology of the signal (by flattening the matrix $n \times m$ to a vector of dimension $1 \times (n \cdot m)$), the example of the persistence image of β_1, generated from same signal used in figure 2 can be seen in figure 3, where the color is closer to yellow if there is a greater density of dimension-2 holes, the election of using β_1 only was applied to all the other signals, because it shows more differences between a pure noise signal and a GW, although to increase this difference and really differentiate between these two type of signals the Kernel Parameters of the function that generates the image can be adjusted to a lower value of standard deviation (which actually paints the matrix cells with the same color where they have a similar density), setting the **kernel params** parameter lower to 0.0001, which results in a better representation of the image as seen on figure 4, which is the same persistence image of 3, but with better localization of the areas where the dimension-2 holes accumulate.

FIG. 3. Persistence image of a signal with a gravitational wave

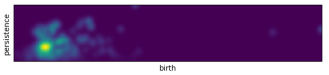

FIG. 4. Persistence image of a signal with a gravitational wave and a smaller standard deviation for the kernel

3.4. *Model training*

3.4.1. CNN 1

Using the persistence images obtained through the TDA application, we proceed to extract 500 waves with an R of 0.65 which was used to create and train our first CNN model, using Keras, which followed the following process:

Data Preparation:

- **Splitting the dataset:** The dataset is divided into training and test sets using an 80-20 split with `train_test_split`, training 80% of the data while the remaining 20% is used for evaluation.

Model Architecture:

- **Building the CNN Model**:
  - **Sequential Model**: A sequential model is initialized.
  - **First Convolutional Layer**:
    - 32 filters, kernel size of 3, 'tanh' activation function.
    - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.5).
  - **Second Convolutional Layer**:
    - 64 filters, kernel size of 3, 'sigmoid' activation function.
    - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.5).
  - **Third Convolutional Layer**:
    - 128 filters, kernel size of 3, 'sigmoid' activation function.
    - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.5).
  - **Flatten Layer**:
    - Flattening of the 3D output from convolutional layers to 1D.
  - **Dense Layer**:
    - A single neuron with 'sigmoid' activation function for binary classification.

Model Compilation and Training

- **Compiling the Model**:
  - The model is compiled with the Adam optimizer and binary cross-entropy loss function, suitable for binary classification tasks.
  - The performance metric is set to 'accuracy'.

- **Training the Model**:
  - The model is trained for 300 epochs with a batch size of 50.
  - 20% of the training data is used as a validation set to monitor and validate the training process.

3.4.2. XGBoost 1

Using the 50 principal components given by PCA of the discrete Fourier Transformation of the signals using different of $R \in \{0.65, 0.5, 0.4, 0.3, 0.25, 0.2, 0.15, 0.065\}$ same split proportion as the CNN.

Data Preparation:

- **Splitting the dataset:** The dataset is divided into training and test sets using an 80-20 split with `train_test_split`, training 80% of the data while the remaining 20% is used for evaluation.

Model Architecture:

- **Building the XGBoost Model**:
  - **Output Type**: Binary classification
  - **Number of estimators**: 3
  - **Learning Rate**: 0.1
  - **Booster**: *gbtree* (gradient-boosting trees)
  - **Gamma (γ)**: 0
  - **Maximum tree depth**: 6
  - **Objective**: binary:logistic
  - **Lambda (λ) (L1 regularization)**: 1
  - **Alpha (α) (L2 regularization)**: 0

Model Compilation and Training

- **Compiling the Model**:
  - **Device**: CPU
- **Training the Model**:
  - 20% of the training data is used as a validation set to monitor and validate the training process.

3.4.3. CNN 2

After obtaining the Discrete Fourier Transforms of each signal in a sample of 1500 waves, we produce another CNN arquitecture using Keras, which followed a simple process:

Data Preparation:

- **Splitting the dataset:** The dataset is divided into training and test sets using an 80-20 split with `train_test_split`, training 80% of the data while the remaining 20% is used for evaluation.

Model Architecture:

- **Building the CNN Model**:
  - **Sequential Model**: A sequential model is initialized.

- **First Convolutional Layer**:
  - 32 filters, kernel size of 3, 'relu' activation function.
  - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.25).
- **Second Convolutional Layer**:
  - 64 filters, kernel size of 3, 'relu' activation function.
  - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.25).
- **Third Convolutional Layer**:
  - 128 filters, kernel size of 3, 'relu' activation function.
  - Followed by a max-pooling layer (pool size of 2) and a dropout layer (dropout rate of 0.5).
- **Flatten Layer**:
  - Flattening of the 3D output from convolutional layers to 1D.
- **Dense Layer**:
  - 32 neurons with 'relu' activation function with a regularizer of 0.01 to stop overfitting.
- **Dense Layer**:
  - A single neuron with 'sigmoid' activation function for binary classification.

Model Compilation and Training

- **Compiling the Model**:
  - The model is compiled with the Adam optimizer and binary cross-entropy loss function, suitable for binary classification tasks.
  - The performance metric is set to 'accuracy'.
- **Training the Model**:
  - The model is trained for 20 epochs with a batch size of 32.
  - 20% of the training data is used as a validation set to monitor and validate the training process.

3.4.4. XGBoost 2

Using the persistence images obtained through the TDA application, we proceed to extract 500 waves with an R of 0.65 with the same process as the previous CNN model and the same split proportion.

Data Preparation:

- **Splitting the dataset:** The dataset is divided into training and test sets using an 80-20 split with `train_test_split`, training 80% of the data while the remaining 20% is used for evaluation.

Model Architecture:

- **Building the XGBoost Model**:
  - **Output Type**: Binary classification

- **Number of estimators**: 500
- **Learning Rate**: 0.1
- **Booster**: *gbtree* (gradient-boosting trees)
- **Gamma (γ)**: 0
- **Maximum tree depth**: 6
- **Objective**: binary:logistic
- **Lambda (λ) (L1 regularization)**: 1
- **Alpha (α) (L2 regularization)**: 0

Model Compilation and Training

- **Compiling the Model**:
  - **Device**: CPU
- **Training the Model**:
  - 20% of the training data is used as a validation set to monitor and validate the training process.

3.5. *Model evaluation*

3.5.1. CNN 1:

After training, the model's performance is evaluated on the test set. The key metrics include precision, recall, f1-score, and accuracy, as shown in the following evaluation report:

	Precision	**Recall**	**F1-Score**	**Support**
0.0	0.84	0.81	0.82	58
1.0	0.75	0.79	0.77	42
Accuracy		0.80		100
Macro Avg	0.79	0.80	0.80	100
Weighted Avg	0.80	0.80	0.80	100

TABLE 1 *Classification report for CNN 1 model*

		Predicted	
		0	**1**
Actual	**0**	47	11
	1	9	33

TABLE 2 *Confusion matrix for CNN 1 model*

3.5.2. XGBoost 1:

With a threshold of $p = 0.5$ for assigning the positive label, the following metrics are given.

	$R = 0.65$	$R = 0.5$	$R = 0.4$	$R = 0.3$	$R = 0.25$	$R = 0.2$	$R = 0.15$	$R = 0.065$
Accuracy	0.97	0.96	0.95	0.84	0.72	0.62	0.55	0.5
Recall	0.94	0.94	0.92	0.92	0.94	0.93	0.93	0.86

TABLE 3 *Accuracy and Recall metrics of XGBoost given different R values*

3.5.3. CNN 2:

	$R = 0.65$	$R = 0.5$	$R = 0.4$	$R = 0.3$	$R = 0.25$	$R = 0.2$	$R = 0.15$	$R = 0.065$
Accuracy	0.9	0.88	0.80	0.75	0.71	0.7	0.57	0.51
Recall	0.9	0.9	0.81	0.72	0.7	0.66	0.55	0.49

TABLE 4 *Accuracy and Recall metrics of CNN 2 given different R values*

3.5.4. XGBoost 2:
The threshold of $p = 0.76$ was selected for assigning the positive label after finding the one that maximizes accuracy, the following metrics are given.

	Precision	**Recall**	**F1-Score**	**Support**
0.0	0.71	0.91	0.80	56
1.0	0.82	0.52	0.64	44
Accuracy		0.74		100
Macro Avg	0.76	0.72	0.72	100
Weighted Avg	0.76	0.72	0.73	100

TABLE 5 *Classification report for XGboost 2 model*

		Predicted	
		0	**1**
Actual	**0**	51	5
	1	21	23

TABLE 6 *Confusion matrix for XGBoost 2 model*

4. Implementation

The code for the Fourier PCA transformation is structured in two main functions: *compute_frequency_domain* and *procesamiento_pca*. The function *compute_frequency_domain* calculates the Fast Fourier Transform

(FFT) of the given signals, filters the frequencies within a specified range, and extracts the corresponding magnitudes. The function *procesamiento_pca* processes the frequency domain results, normalizes the data, and applies Principal Component Analysis (PCA).

```
def compute_frequency_domain(signals, start_idx, end_idx, sampling_rate=1.0):
    frequency_domain_data = []
    for i in range(start_idx, end_idx):
        # Compute the FFT of the signal
        signal_fft = fft(signals[i])
        # Compute the corresponding frequencies
        frequencies = fftfreq(len(signal_fft), d=1/sampling_rate)
        # Filter frequencies and magnitudes in the range -0.05 to 0.05
        filtered_indices = np.where((frequencies >= -0.05) & (frequencies <= 0.05))
        filtered_frequencies = frequencies[filtered_indices].tolist()
        filtered_magnitudes = np.abs(signal_fft)[filtered_indices].tolist()
        # Store filtered frequencies and magnitudes
        frequency_domain_data.append({
            'Signal': i+1,
            'Frequency': filtered_frequencies,
            'Magnitude': filtered_magnitudes
        })
    return frequency_domain_data

# Example usage: Compute frequency domain for the specified range of signals
frequency_domain_results = compute_frequency_domain(datos_np, start_idx=0, end_idx=len(datos_np),
     sampling_rate=1.0)

# Convert the results into a DataFrame
frequency_domain_df = pd.DataFrame(frequency_domain_results)
fouriertable = frequency_domain_df.copy()
frequency_domain_df.head()
```

The following function *crearimagen* generates the n amount of persistence images of the same length, by first applying the sliding window embedding to the signal, and then calculating the persistent vector for β_1, this vector will be saved on a list for the algorithm to loop in all the other signals what where selected, after calculating all this vectors (using samples of the signal to reduce execution time), they are processed individually to calculate their persistence image that is allocated to a numpy.array, each of this matrix can be later flattened to be presented as a dataframe to train the machine learning models.

```
def crearimagen(amount:int):
    lista_vectores=[] #save all the  persistence diagrams
    for i in range(amount):
        x_periodic = np.linspace(0, 10, 869)
        y_periodic = X[i] #Select all the values in the time series

        tau=2
        d=4
        embedder_periodic = SingleTakensEmbedding(
            parameters_type="fixed",
            time_delay=tau,
            dimension=d,
```

```
        )
        y_periodic_embedded = embedder_periodic.fit_transform(y_periodic) #Embed the time series


        #Generate an array of random indexes from y_periodic_embedded
        np.random.seed(0)
        random_indexes = np.random.randint(0, y_periodic_embedded.shape[0], 300)
        #Sort the random indexes array
        random_indexes.sort()
        #Generate the persistence diagram with the selected samples
        y_periodic_sel = y_periodic_embedded[random_indexes]
        ripserperiod = ripser.ripser(y_periodic_sel, distance_matrix=False)["dgms"]
        diagrams= ripserperiod[1].copy() #Select only the 2-dimensional holes

        lista_vectores.append(diagrams)







    # Generate the images for all the persistence diagrams at once to assure the same pixel size
    pimgr = PersistenceImager(pixel_size=0.01)
    pimgr.kernel_params = {'sigma': 0.000000001}
    pdgms = lista_vectores
    pimgr.fit(lista_vectores, skew=True)
    pimgs = pimgr.transform(pdgms,skew=True)


    return pimgs
```

After developing our models, such as tree-based models like XGBoost and on the other hand, the Convolutional Neural Network (CNN), we use the following code to compute and visualize SHAP values, which help us understand the contribution of each feature to the model's predictions. For the XGBoost model, we initialize an explainer and calculate the SHAP values for the dataset X. These SHAP values are then visualized using a beeswarm plot and a bar plot to show the distribution and average impact of each feature. For the CNN, we define a custom prediction function and utilize KernelExplainer to estimate SHAP values. The SHAP values are reshaped to match the dimensions of the input data and visualized similarly with beeswarm and bar plots.

```

# Import SHAP library
import shap

# Compute SHAP values for Tree-based models
# Create an explainer object for the given model
explainer = shap.Explainer(model)

# Compute SHAP values for the dataset X
shap_values = explainer(X)

# Create a beeswarm plot to visualize the distribution of SHAP values
```

```
shap.plots.beeswarm(shap_values)

# Create a bar plot to show the average impact of each feature
shap.plots.bar(shap_values)

# Compute SHAP values for Neural networks

# Define a function that uses the model to predict outputs for the given input x
def f(x):
    return model.predict(x)

# Create a KernelExplainer object for the model using the function f and dataset X
explainer = shap.KernelExplainer(f, X)

# Compute SHAP values for the dataset X using a specified number of samples (n)
shap_values_multiple = explainer.shap_values(X, nsamples=n)

# Reshape the computed SHAP values to match the dimensions of the input dataset X
shap_values_dim_corrected = shap_values_multiple.reshape(X.shape[0], X.shape[1])

# Set the maximum number of features to display in the plots
max_features = 10

# Create a beeswarm plot to visualize the distribution of SHAP values
# Displaying the top max_features features
shap.summary_plot(shap_values_dim_corrected, X, max_display=max_features)

# Create a bar plot to show the average impact of each feature
# Displaying the top max_features features
shap.summary_plot(shap_values_dim_corrected, X, plot_type="bar", max_display=max_features)
```

To see the complete documentation of the scripts used on the project, enter the following link.

5. Results and Analysis

5.1. *Model comparison*

Models	**R = 0.65**	**R = 0.5**	**R = 0.4**	**R = 0.3**	**R = 0.25**	**R = 0.2**	**R = 0.15**	**R = 0.065**
RandomForest	0.97	0.96	0.96	0.84	0.70	0.60	0.54	0.48
SVM	0.95	0.91	0.95	0.85	0.77	0.69	0.56	0.50
XGBoost	0.97	0.96	0.95	0.84	0.72	0.62	0.55	0.50
Logistic Regression	0.95	0.87	0.94	0.85	0.74	0.66	0.55	0.50
Neural Network (MLP)	0.94	0.86	0.94	0.84	0.75	0.70	0.55	0.50

TABLE 7 *Accuracy of the machine learning models performance across different R values*

The table 7 presents the accuracy results of the models under varying noise levels in the measurements. These models were trained using PCA with 50 components derived from the Discrete Fourier Transform. The table illustrates the increasing difficulty models face as the noise coefficient (R)

rises. Models generally maintain acceptable performance up to $R = 0.2$. Through experimentation, it was determined that using the training data with $R = 0.4$ was the best choice for training our models. To obtain these results, the models were trained with 1200 signals out of 1500 (80% of the data with $R = 0.4$). According to these results, the SVM and XGBoost 1 models show the best performance. However, since this is a classification problem, accuracy alone is not sufficient to fully evaluate model performance.

Models	**R = 0.65**	**R = 0.5**	**R = 0.4**	**R = 0.3**	**R = 0.25**	**R = 0.2**	**R = 0.15**	**R = 0.065**
RandomForest	0.95	0.94	0.91	0.93	0.96	0.95	0.96	0.89
SVM	0.85	0.85	0.90	0.84	0.85	0.95	0.85	0.85
XGBoost	0.94	0.94	0.92	0.92	0.94	0.93	0.93	0.86
Logistic Regression	0.93	0.93	0.91	0.92	0.91	0.77	0.90	0.85
Neural Network (MLP)	0.91	0.90	0.93	0.90	0.90	0.89	0.85	0.80

TABLE 8 *Recall of the machine learning models performance across different R values*

However, accuracy is not the only metric for evaluating the performance of a classification model. Recall measures a model's ability to identify all positive instances in the dataset, answering the question: of all the actual positive instances, what proportion does the model correctly identify? Table 8 shows that even with lower values of R (indicating more noise), the models were still able to accurately predict the positive labels. Regardless of the noise level, the vast majority of our models correctly identified at least 85% of the instances related to gravitational waves.

XGBoost	**10 different R values from 0.15 to 0.65**	**500 different R values from 0.05 to 0.65**	**100 different R values from 0.05 to 0.25**
Recall	0.81	0.77	0.58
Accuracy	0.85	0.82	0.60
Training time	10s	12s	10s

TABLE 9 *XGBoost performance metrics across different R ranges*

Given that one of our best models was built using XGBoost, we conducted a series of tests by varying different random R values to observe the model's performance. The results of these tests are presented in Table 9. This model was trained using the principal component analysis (PCA) of the Fourier transform of the signals. We can observe that even with small R values ($R = (0.05, 0.25)$), the results remain acceptable.

Considering the excellent metrics obtained by the XGBoost model and our objective of finding a classifier that performs well across various signal-to-noise ratios, we chose to further explore this model. Specifically, we aimed to ensure it maintains good performance with different levels of noise. To gain deeper insights into the decision-making process of the XGBoost model, which was trained using observations of 50 components of Principal Component Analysis of the Fourier transform of signals with $R = 0.4$, we implemented SHAP values. This analysis helps us better understand the factors influencing the model's predictions.

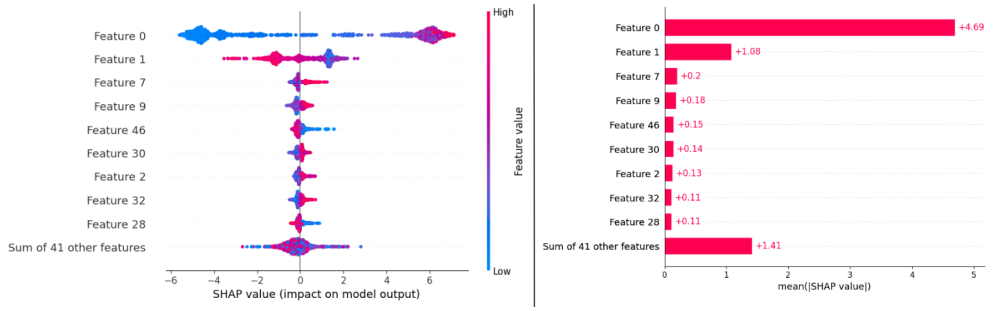

FIG. 5. Beeswarm SHAP plot (left) and Mean SHAP plot (right) of the XGBoost model trained using PCA observations of the Fourier transform of signals with $R = 0.4$.

The SHAP value analysis revealed several key findings. The beeswarm plot indicated that Feature 0 (the first principal component) is the most influential, contributing significantly to both positive and negative values in the model's predictions. Higher values of this component increase the model's prediction, while lower values decrease it. Conversely, Feature 1 (the second principal component) exhibited an inverse relationship, where higher values reduced its SHAP value, thereby decreasing the model's prediction. The mean SHAP plot reinforced these insights, showing that Feature 0 had the highest average absolute SHAP value, highlighting its importance in explaining the variability of the Fourier magnitudes. These findings underscore the critical role of the first few principal components in driving the model's performance.

CNN 1	**R = 0.65**	**R = 0.5**	**R = 0.25**
Recall	0.80	0.70	0.50
Accuracy	0.85	0.73	0.49
Training time	2m 16s	2m 03s	22s

TABLE 10 *CNN 1 performance metrics across different R values*

To assess the potential of topology in aiding signal label prediction, we utilized persistence images derived from the signals and trained a convolutional neural network (CNN) for prediction. The dataset consisted of 1200 persistence images representing the signals. Table 10 presents the performance of these models across various R values. It is noteworthy that the model exhibits high sensitivity to noise; specifically, when $R \leq 0.25$, its predictive accuracy diminishes significantly.

Model comparison	**Accuracy**	**Recall**	**Execution time**
CNN 1 Input: This was done using signal persistence imaging	85%	80%	2m 16s
XGBoost 2 Input: This was done using signal persistence imaging	75%	55%	5s

TABLE 11 *Model comparison: CNN 1 vs XGBoost 2*

Table 11 highlights the performance disparities between the convolutional neural network (CNN) and the XGBoost model. Several key distinctions emerge: the CNN achieved superior metrics in recall and accuracy, albeit with significantly longer training times. Conversely, while the XGBoost model exhibited lower metrics, its computational cost was notably lower, yet yielded comparable results

Given the great performance of the CNN trained with persistence images from the signals, we now turn to SHAP values to analyze this model. By applying SHAP, we aim to gain a better understanding of the CNN's decision-making process and identify which features are most influential in its predictions.

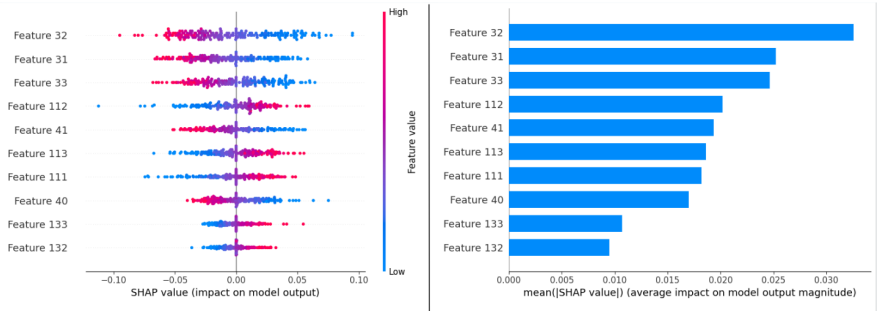

FIG. 6. Beeswarm SHAP plot (left) and Mean SHAP plot (rigth) of the CNN model trained with persistence images from the signals.

The SHAP value analysis for the CNN model reveals several key findings about its decision-making process. Feature 32 stands out as the most influential feature, contributing significantly to both positive and negative values in the model's predictions. Higher values of Feature 32 decrease the model's predictions, while lower values increase the SHAP value, leading to higher predictions. Notably, Feature 32 exhibits a larger value for class 0 and a much lower value for class 1, making it a crucial factor in distinguishing between these classes. Features 31 and 33 exhibit similar behavior, where their SHAP values also meaningfully impact the model's output. The mean SHAP plot further emphasizes the importance of these features, showing that Features 32, 31, and 33 have the highest average impact on the model's output. These features are particularly effective in differentiating between noise and actual signal data, making them vital for the model's performance.

CNN 2	**R = 0.65**	**R = 0.45**	**R = 0.25**
Recall	0.90	0.80	0.72
Accuracy	0.92	0.81	0.69
Training time	4m 12s	6m 03s	10m 12s

TABLE 12 *CNN 2 performance metrics across different R values*

Building on the previous results, we developed a new CNN model trained with persistence images obtained from the results of the fast Fourier transform. Table 12 displays the performance of this model

across various R values, revealing an improvement over the initial model, although with an increase in training time.

5.2. *Discussion*

The results of our study provide valuable insights into the efficacy of combining Principal Component Analysis (PCA) of the Fourier discrete transform of signals with topological persistence images in the creation of a Convolutional Neural Network (CNN) model for the classification of gravitational waves and noise. Our findings underscore the significance of leveraging both techniques to enhance the model's performance and robustness. Notably, while PCA of the Fourier series serves as a powerful input for the model, we observed that its effectiveness diminishes significantly when the signal-to-noise ratio (R) falls below 0.2. It is noteworthy to highlight the remarkable improvement in classification accuracy achieved through the combination of PCA and topological persistence images. Our integrated approach yielded an average accuracy of 0.8 across all signal-to-noise ratios (R), underscoring the robustness and efficacy of the proposed model. In contrast, traditional models trained solely on raw signal data, such as CNNs or other conventional techniques, often struggle to achieve meaningful classification accuracy, typically hovering around 50%. This accuracy level is akin to random guessing and emphasizes the inherent complexity and challenges associated with gravitational wave signal classification. This highlights the importance of considering the limitations of individual preprocessing techniques and the need for complementary approaches to address varying signal characteristics. By incorporating Topological Data Analysis (TDA) through persistence images, we augment the model with a more comprehensive understanding of signal morphology and temporal persistence. This integration enables the CNN to capture complex spatial and temporal patterns inherent in gravitational wave signals, thereby enhancing its robustness and accuracy. Crucially, our results demonstrate that the combination of TDA with Fourier-based PCA results in a more resilient and accurate model, particularly when faced with diverse and random R parameters. This synergistic approach not only improves the model's classification performance but also enhances its ability to generalize to unseen data and adapt to varying signal conditions.

6. Conclusions and Recommendations

In conclusion, our study presents a novel approach for enhancing the accuracy and robustness of Convolutional Neural Network (CNN) models in gravitational wave classification. By integrating Principal Component Analysis (PCA) of Fourier discrete transform signals with topological persistence images, we achieved a significant improvement in classification accuracy, with an average accuracy of 0.8 across various signal-to-noise ratios (R). This surpasses the limitations of conventional models trained solely on raw signal data, which often yield accuracy levels akin to random guessing. Our findings highlight the transformative potential of combining advanced preprocessing techniques, underscoring the importance of leveraging complementary methods for improved model performance.

A significant limitation of the study pertains to the artificial nature of the signals analyzed, which were generated by us rather than occurring naturally. While this approach enables controlled experimentation and systematic evaluation of models, it introduces a degree of artificiality that may impact the generalizability of the findings to real-world scenarios. Specifically, the reliance on artificially generated signals may lead to model performance being overestimated due to the inherent predictability and regularity of these signals. In contrast, real-life signals are characterized by inherent variability, noise, and unpredictability, posing challenges that artificial signals may not fully capture. As

a result, while models trained on artificial signals may exhibit proficiency in detecting patterns within these controlled environments, their performance may falter when faced with the erratic behavior and diverse characteristics of real-life signals.

Moving forward, several recommendations can be made for future studies or improvements in gravitational wave classification:

- Explore Additional Preprocessing Techniques: Investigate the efficacy of integrating other preprocessing techniques, such as wavelet transforms or time-frequency analysis, to further enhance the model's performance and adaptability to diverse signal characteristics.
- Incorporate Temporal Dynamics: Consider incorporating temporal dynamics and sequential information into the model architecture, potentially through recurrent neural networks (RNNs) or attention mechanisms, to capture temporal dependencies and improve classification accuracy.
- Evaluate Real-world Data: Validate the proposed approach using real-world gravitational wave data from observatories such as LIGO or Virgo to assess its applicability and generalization to practical scenarios.

Overall, our study lays the foundation for future research in gravitational wave classification, offering valuable insights and directions for further improvements in model performance and applicability. By addressing these recommendations and continuing to innovate in the field, we can advance our understanding of gravitational wave phenomena and unlock new discoveries in astrophysics.

References

Aktas, M E Akbas, E & Fatmaoui, A E (2019) Persistence homology of networks: methods and applications. *Applied network science*, *4*(1) https://doi.org/10.1007/s41109-019-0179-3

Bauer, U (2021) Ripser: Efficient computation of Vietoris-Rips persistence barcodes. *J. Appl. Comput. Topol.*, *5*(3) 391–423. https://doi.org/10.1007/s41468-021-00071-5

Cai, R G Cao, Z Guo, Z K Wang, S J & Yang, T (2017) The gravitational-wave physics. *National Science Review*, *4*(5) 687–706.

Chan, M L Heng, I S & Messenger, C (2020) Detection and classification of supernova gravitational wave signals: A deep learning approach. *Physical Review D 102*(4) 043022.

Chazal, F & Michel, B (2021) An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists. *Frontiers in artificial intelligence*, *4*. https://doi.org/10.3389/frai.2021.667963

Chen, T & Guestrin, C (2016) Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939785

Cornish, N J (2020) Time-frequency analysis of gravitational wave data. *Physical Review D 102*(12) 124038.

De Ville, B (2013) Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, *5*(6) 448–455.

Deng, R & Duzhin, F (2022) Topological data analysis helps to improve accuracy of deep learning models for fake news detection trained on very small training sets. *Big Data and Cognitive Computing*, *6*(3) 74.

George, D & Huerta, E A (2018) Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B 778*, 64–70.

IBM (2021) What are Convolutional Neural Networks? — IBM https://www.ibm.com/topics/convolutional-neural-networks

IBM (2023) What is Principal Component Analysis (PCA)? — IBM https://www.ibm.com/topics/principal-component-analysis

Li, X R Yu, W L Fan, X L & Babu, G J (2020) Some optimizations on detecting gravitational wave using convolutional neural network. *Frontiers of physics*, *15*, 1–11.

Perea, J A & Harer, J (2015) Sliding windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, *15*, 799–838.

Pitkin, M Reid, S Rowan, S & Hough, J (2011) Gravitational wave detection by interferometry (ground and space). *Living Reviews in Relativity*, *14*, 1–75.

Rowan, S & Hough, J (1999) *The detection of gravitational waves*. CERN

spectric-labs. (2023, March) Using Deep Learning for Signal Detection and Classification. https://www.spectric.com/post/using-deep-learning-for-signal-detection-and-classification

Tauzin, G Lupo, U Tunstall, L Pérez, J B Caorsi, M Medina-Mardones, A M Dassatti, A & Hess, K (2021) Giotto-tda: A topological data analysis toolkit for machine learning and data exploration. *Journal of Machine Learning Research*, *22*(39) 1–6. http://jmlr.org/papers/v22/20-325.html

Thorne, K S (2018) Nobel lecture: Ligo and gravitational waves iii. *Reviews of Modern Physics*, *90*(4) 040503.

Trevisan, V (2022) Using SHAP Values to Explain How Your Machine Learning Model Works. https://towardsdatascience.com/using-shap-values-to-explain-how-your-machine-learning-model-works-732b3f40e137

Wolfram Research, I (n.d.) Fourier Series – from Wolfram MathWorld. https://mathworld.wolfram.com/FourierSeries.html