Active Learning approach to Gravitational Waves Classification Algorithms

Author: Àlex Capilla Miralles
*Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.**

Advisor: Tomás Andrade Weber

Abstract: This project explores the integration of Bayesian Optimization (BO) algorithms into a base machine learning model, specifically Convolutional Neural Networks (CNNs), for classifying gravitational waves among background noise. The primary objective is to evaluate whether optimizing hyperparameters using Bayesian Optimization enhances the performance of the base model. For this purpose, a Kaggle [1] dataset that comprises real background noise (labeled 0) and simulated gravitational wave signals with noise (labeled 1) is used. Data with real noise is collected from three detectors: LIGO Livingston, LIGO Hanford, and Virgo. Through data preprocessing and training, the models effectively classify testing data, predicting the presence of gravitational wave signals with a remarkable score, 83.61%. The BO model demonstrates comparable accuracy to the base model, but its performance improvement is not very significant (84.34%). However, it is worth noting that the BO model needs additional computational resources and time due to the iterations required for hyperparameter optimization, requiring an additional training on the entire dataset. For this reason, the BO model is less efficient in terms of resources compared to the base model in gravitational wave classification.

I. INTRODUCTION

The detection of gravitational waves by the LIGO detectors in 2015 has been a significant discovery, confirming Einstein's prediction of gravitational waves in 1916 [2]. Detecting and analyzing these signals reveals a variety of physical properties of the objects emitting them (such as the distance, mass of the astrophysical objects). Nevertheless, there is large amount of data to be processed, which is a problem that has led to the development of machine learning based techniques that optimize data processing in gravitational wave studies.
Among the various data processing methods, the most important ones are Matched Filtering [3] an Machine Learning (ML). Matched filtering consists on correlating a known reference signal (template) with a noisy input signal for determining if the reference signal is present or not in the input signal. The template can be obtained solving and modelling Einstein's equations for gravitational waves. However, this method requires a vast amount of templates, and can be less powerful when the noise distribution in non-Gaussian.
ML offers one of the most effective and efficient approaches, as the selection of the appropriate architecture and proper preprocessing of the data can lead to robust and reliable outcomes.
For this reason, the objective of this study is to develop a base model using conventional Machine Learning techniques, along with convolutional neural networks (CNN), for the detection of gravitational waves amidst background noise. A brief explanation on the architecture and function of CNN will be provided, as well as the preprocessing method used for the signals. This base model will serve as a reference for a subsequent model, which will incorporate Bayesian Optimization algorithms to identify optimal hyperparameters. Further explanation of this approach will be covered later in the project as well. The aim is to determine whether this approach enhances accuracy and efficiency in gravitational wave detection.

A. Introduction to Gravitational Waves

Gravitational waves are ripples in spacetime that can propagate across vast distances, and come from fully relativistic sources that are changing rapidly, such as the collision of two black holes, neutron star mergers or supernovae.
As gravitational waves can be considered as region in spacetime in which the gravitational field is weak but not stationary, so weak-field equations are needed for studying them [4].
Weak-field theory assumes that a weak gravitational field consists in the flat spacetime metric, $\eta_{\mu\nu}$, plus a perturbation, $h_{\mu\nu}$, i.e., $g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}$.
The Einstein equations in vacuum ($T^{\mu\nu} = 0$) far outside the field are:

$$\left(-\frac{\partial^2}{\partial t^2} + \nabla^2\right) \bar{h}^{\mu\nu} = 0 \tag{1}$$

where the trace reverse tensor $\bar{h}^{\mu\nu} = h^{\mu\nu} - \frac{1}{2}\eta^{\mu\nu}h^{\mu}_{\ \mu}$ is used, defined with index-raised quantities. The solution for 1 has the following wave-like form:

$$\bar{h}^{\mu\nu} = A^{\mu\nu}\exp(\mathrm{i}k_\mu x^\mu) \tag{2}$$

These equations describe how gravitational waves propagate through spacetime, influencing the proper distances between objects. This distinction underscores the nature

*Electronic address: acapilmi7@alumnes.ub.edu

of gravitational waves as distortions of spacetime itself.
One of the most significant sources of gravitational waves (as mentioned before) is the merger of black hole binaries, systems that consist on black holes orbiting each other, gradually losing energy through emissions of gravitational waves, until they collapse. As the black holes become closer together, the frequency and amplitud of the gravitational wave increases, resulting in a quasi-circular inspiral waveform.

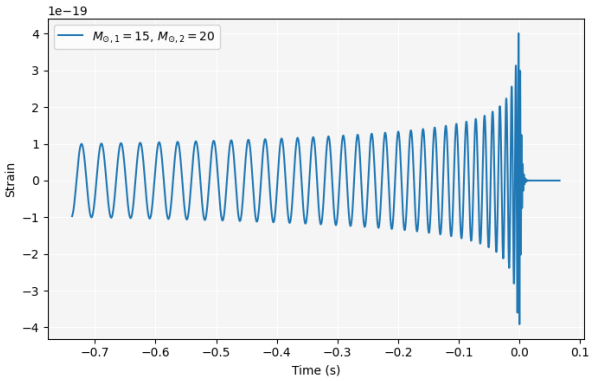

FIG. 1: Waveform of a binary merger, created with PyCBC [5]. The masses of the binary are 15 and 20 solar masses.

II. CONVOLUTIONAL NEURAL NETWORKS (CNNS)

Gravitational waves signals are time series collected from three detectors: LIGO Livingston, LIGO Hanford and Virgo.
Every file of the dataset has three time series corresponding to data from each detector. After the preprocessing of the data (which will be elaborated upon later), the resulting training data are images. For this reason, CNNs will be used, because they are domain-aware neural networks specifically designed for images.
In CNNs, images are represented as tensors with dimensions corresponding to length, width, and depth, where depth represents the number of color channels [6]. Each layer of the CNN is three-dimensional to match the dimensionality of the input image. The depth of the layer corresponds to the image's color channels: it is 3 for colored images (RGB channels) and 1 for grayscale images. For the CNN in this TFG, grayscale is enough for differentiating between a signal and noise. The input image will be a tensor that can be represented as a matrix, while the layers remain three-dimensional.
The convolution operation that every layer undergoes is based on a filter that maps the activations from one layer to the next one, where the filter has weights of the same depth as the current layer but with a smaller spatial extent, resulting into a hidden state in the next layer determined by the dot product between the filter weights and a spatial region (that matches the filter's dimensions), and this process is repeated for all possible positions in the input. Precisely, the convolution procedure can be expressed as follows [7]:

$$y_{i^{l+1},j^{l+1},d} = \sum_{i=0}^{H}\sum_{j=0}^{W}\sum_{d^l=0}^{D^l} f_{i,j,d^l,d} \times x^l_{i^{l+1}+i,j^{l+1}+j,d^l} \quad (3)$$

Which is reapeated for all D, and for any spatial location (i^{l+1}, j^{l+1}) satisfying $0 \leq i^{l+1} < H^l - H + 1 = H^{l+1}$, $0 \leq j^{l+1} < W^l - W + 1 = W^{l+1}$, where H is the number of rows and W the number of columns. x^l corresponds to the input tensor. $f_{i,j,d^l,d}$ represents the set of convolutional kernels.
Additionally, the choice of the loss function is critical, as it calculates a numerical value that represents the discrepancy between the prediction that has been made by the CNN for a signal and the true result (the label of that signal). The output of the loss function is this value. Thus, minimizing this value is the goal.

III. DATA VISUALIZATION AND PREPROCESSING

The dataset used in this TFG comes from the 'G2-Net Gravitational Wave Detection' competition on Kaggle [1], and includes data from three interferometers: LIGO-Hanford, LIGO-Livingston, and Virgo. It comprises two types of data: signals with real noise combined with a simulated gravitational wave signal (labeled 1), and signals containing solely real noise (labeled 0). Both categories span 2 seconds and have a sampling frequency of 2048Hz. Using the raw data directly for training the model (further explanation of the model will be explained later) is inefficient and will not yield positive results. This is primarily due to the fact that gravitational wave signals are weak, their order is comparable to the noise.

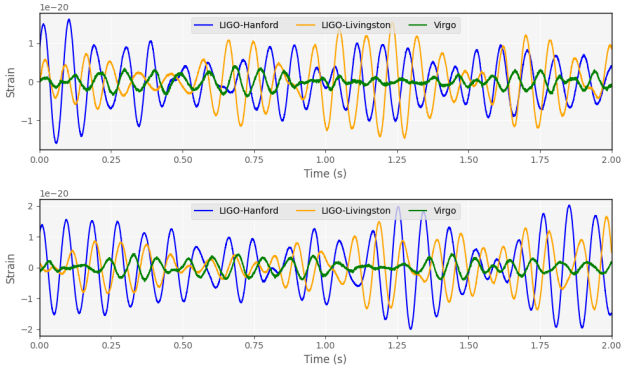

FIG. 2: Signals with simulated gravitational wave (upper) and real noise (lower).

For this end, the data from all detectors is stacked and

then a *Q-transform* [8] is employed on the data. In this case, from the python library *nnAudio.Spectrogram CQT1992v2* [9] is used.
The Q-transform originates from a discrete Fourier Transform (DFT). Q is the ratio of the center frequency to bandwidth. It returns a tensor of 3D spectrograms, in a image format. This format makes the difference between a gravitational wave signal and a noise signal clearer, and the model becomes better at distinguishing the difference after the training with a decent accuracy, achieving a higher level of accuracy (further details on this approach elaborated later).
An example of the image obtained after applying the method for a gravitational wave signal (in the upper image), and for noise (in the lower image) is represented in FIG.3:

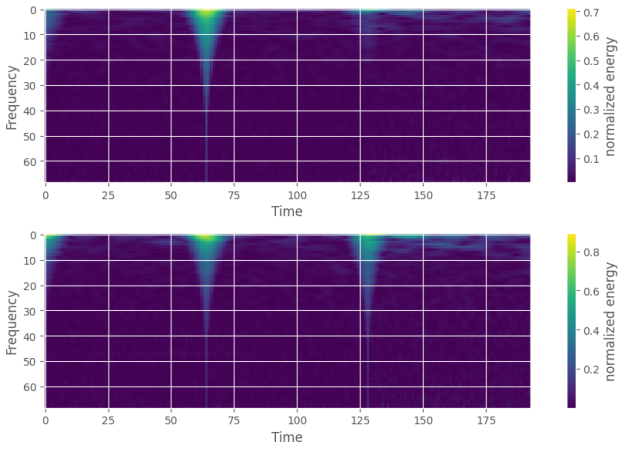

FIG. 3: Q-transformed signals with simulated gravitational wave (upper) and real noise (lower). The difference of signals with or without gravitational waves can not be found with the naked eye.

However, the upside down yellow cones that can be seen in FIG. 3 must not be misunderstood as gravitational waves. Those are mathematical artefacts that appear because strains are stacked one next to each other, resulting into a discontinuity in the intersection of the signals.
Eliminating these mathematical artefacts was tested, yet it did not result effective in terms of score. Instead, it increased the traning time, as it consisted on implementing the CQT for the signals of each detector separately (and then, stacking the images), which resulted in an increase up to three times the training time.
A comparison between spectograms obtained by both methods is shown in FIG.4:

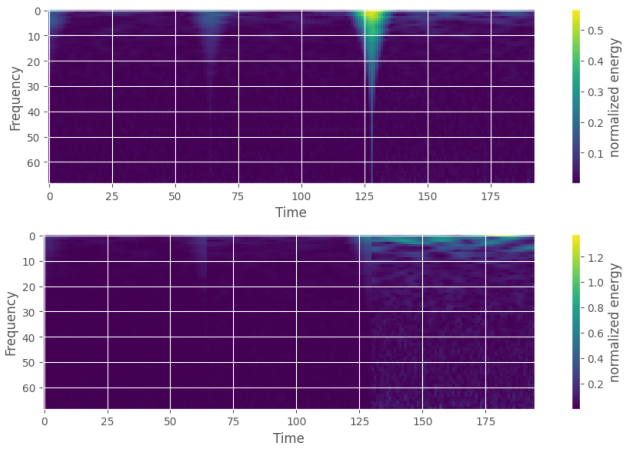

FIG. 4: Signals stacked before CQT (upper) and stacked after CQT (lower).

IV. BASE MODEL

The dataset is divided in two parts: the training set and the testing set.

- **Training Set:** In the training set, the signals are labeled according to whether they contain a signal (label 1) or not (label 2), and are used to train the model. This data is used for teaching the model how to distinguish between signals with gravitational waves, or noise.
- **Testing Set:** The testing set consists of signals that are all labeled with the same neutral value (0.5), since its purpose is to evaluate the performance of the trained model on unseen data. For this reason, the label is chosen as 0.5 to ensure that model's predictions are not biased. After the testing, the model will return a value in-between 0 and 1, indicating the presence of a gravitational wave if the value is higher than 0.5, or the absence of it if the value is lower than 0.5.

The labeling of the dataset is crucial because it conditions the architecture of the model. In this case, it forces the output to be a number comprised among (0,1). Using the *Keras* library, it is possible to construct a CNN using pre-built layers (in this TFG only sequential architectures are implemented). The input layer of the model is a convolutional layer where the input is a tensor shaped with the same size as the Q-transformed signal. It employs 256 filters (the number of filters is important for the BO part).
Following the convolutional layer, a Batch normalization layer is implemented (it adjusts the presentation of the data, for a better compatibility in the next stages of the model), and then flattened, for transforming the output into a one-dimensional array.
Then, a dense layer with the same filters as the convolutional layer is implemented, and the number of dense

layers implemented are a important parameter for the BO part (along with the number of filters), since it will be modified to increase the accuracy of the model.
Finally, the output layer consists in a single dense layer with a sigmoid activation function, and generates an output comprised among (0, 1), which is a probability score that indicates the presence of a gravitational wave.

V. BAYESIAN OPTIMIZATION (BO) MODEL

Once a base model is established, it is possible to improve it optimizing its hyperparamters. The difference between hyperparameters and normal parameters is that normal parameters are modified during training, while hyperparameters are set before training begins and are not modified during training by the CNN.
The strategy that this current TFG is studying is Bayesian optimization [10]. The goal of this method is to find the global maximum or minimum of a function (in this case, the accuracy function), with the following steps:

1. First, a surrogate model is defined (it is called surrogate because it acts as a stand-in or substitute for the final model that you ultimately want to build or optimize), in which a set of initial hyperparameters are chosen. This set is the **prior**.

2. Secondly, the function (the performance metric we want to optimize, namely the loss or accuracy function) is evaluated (using the Bayes rule), in order to determine the posterior. The posterior is a distribution that provides a probabilistic summary of the true function.

3. Then, an acquisition function $\alpha(x)$ (which is a function of the posterior) will decide the next sample point, $x_t = \text{argmax}_x \alpha(x)$.

4. This process is repeated (from step 2) with new sampled data until the method converges.

Acquisition functions are crucial to Bayesian Optimization. For this TFG, the function used will be *probability of improvement* (PI). It chooses the next query point as the one which has the highest *probability of improvement* over the current max $f(x^+)$. Mathematically:

$$x_{t+1} = \text{argmax}(P(f(x) > f(x^+) + \varepsilon)) \qquad (4)$$

where $P(\cdot)$ indicates probability and ε is a small positive number that represents the exploration of the method (high values of ε correspond to more exploration).
In the context of CNN, a the python library *scikit-optimize* will be used, as it provides the tools to implement this optimization in the model.
The code initially declares a search space for the optimization. The dimension of the search space is 2:

- Dense layers. This hyperparameter sets the number of dense layers in the neural network. The search space for the possible number of dense layers consists of integer values from 1 to 4.

- Number of filters (width of a CNN). In a CNN each filter (also known as channel) generates a feature map for identifying patterns. Numbers among 2^i $\forall\ 5 \leq i \leq 8$ will be searched (i being an integer number).

To perform the optimization, *gp-minimize* [11] from *scikit-optimize* will be used. This function applies BO method explained earlier in this section, maximizing the value of accuracy (it minimizes the negative accuracy value). Further explanation of its implementation can be found in the Appendix.
It is important to mention that this function calls the model with a different set of hyperparameters each time, until it reaches the number of calls specified in the function (we will use $n\_calls = 16$). However, every call uses all the training data for the training, which is costly. In order to reduce the training time, the approach used in this TFG consists on dividing the dataset in 16 parts, and every call (namely, every exploration of a set of hyperparameters) will use one part of the dataset. This vastly reduces cost of this method, but the fact that the trainings are not made with the entire dataset might have a negative influence in the result of the accuracy obtained for each sampling point.
The resulting hyperparameters found with the BO method will be used for training the model, and its performance will be compared to the base model, which used a not optimized set of hyperparameters.
The next figure shows an example of the exploration of the hyperparameter space, finding the hyperparameters that lead to the maximum accuracy.

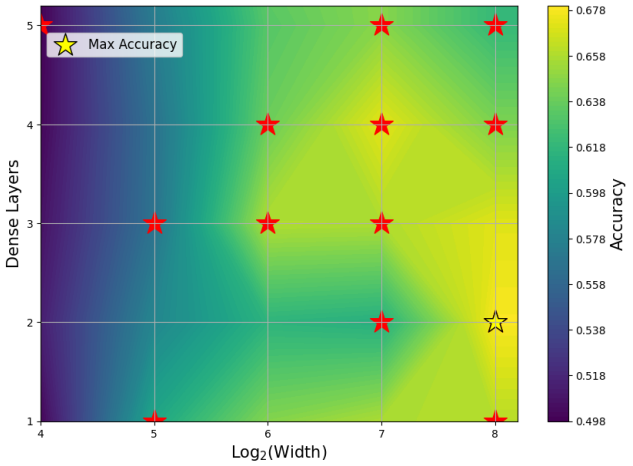

FIG. 5: Exploration of the hyperparameter space, in which the color bar represents the accuracy. The yellow star corresponds to the best result

The greatest value of accuracy is obtained when the num-

ber of dense layers is 2, and the number of filters is 256 (base 2 logarithm of the number of filters equals to 8), which represents the point (8,2) in the hyperparameter space. With a low number of filters, CNNs do not capture all the important features present in the spectrograms, and the accuracy diminishes considerably, as shown in the color map.

VI. RESULTS

It is important to mention that hyperparameters that were not optimized, such as the learning rate, batch size, and activation functions are identical in the base model and the BO model.
Once the model has predicted the entire test dataset, a file with the predictions is submitted to Kaggle. Submissions are evaluated [1] on area under the ROC curve between the predicted probability and the observed target, and it provides a score from 0 to 1. For the base model, the score has been 0.83612. The base model used as default hyperparameters 1 dense layer and 2^8 neurons. The BO model occupied double the base model training time (it required two trainings with the entire dataset). With the first training, the best hyperparameters were obtained: 2^8 neurons and 2 dense layers. With this hyperparameters, the score of the model became 0.84339. Comparing to the score of the base model (0.83612), the difference is approximately 0.007. As a reference, the difference between the first and the second place in this competition is 0.003.
As seen in FIG.3, the greatest number of filters studied lead to the maximum accuracy, so greater values of width were also studied in different tranings. However, the model became unstable for values greater than 2^8, so the possibility of studying greater values was discarded. This is because every point in the hyperparameter space, as explained previously, was studied with a reduced set of training data, which might be the cause for the instability of the model for high values of width. In this current TFG, we do not dispose of sufficient computational resources to train the hyperparameter optimization part with the totality of the training set for every step.

VII. CONCLUSIONS

This TFG has explored that models based on a convolutional neural networks, with the appropriate preprocessing of the data, lead to a great performance for gravitational waves classification.
Optimizing the hyperparameters with Bayesian Optimization seem to have a slight increase of performance in terms of score (as mentioned before, it can be the difference between winning and loosing the competition). However, the optimal hyperparameters obtained are very similar to the hyperparameters of the base model, which might be the reason to this minimal increase. Nevertheless, this algorithm vastly increases the computational cost of the training, and becomes unstable for great values of the number of filters. Better results can be obtained using a greater amount of data for the exploration, yet the model becomes even more expensive.

Acknowledgments

I am sincerely grateful to my tutor, Dr. Tomás Andrade, for his invaluable support, dedication and for giving me the opportunity to be part of the NR group.
I also want to thank my parents for affording me with this opportunity, as well as to Lucía, for her support.

[1] European Gravitational Observatory, 2021, *G2Net Gravitational Wave Detection*, Kaggle. https://kaggle.com/competitions/g2net-gravitational-wave-detection
[2] Einstein, A. (1916). *Approximate Integration of the Field Equations of Gravitation.* Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften (Berlin), 1, 688–696.
[3] Yan, J. et al, (2022). *Generalized approach to matched filtering using neural networks.* Physical Review D, 105(4), 043006.
[4] B. Schutz., 2009, *A First Course in General Relativity.* Cambridge University Press
[5] Niels J., 2021. *PyCBC: Gravitational-wave data analysis toolkit [Software].* https://github.com/gwastro/pycbc
[6] Charu C. Aggarwal, 2018, *Neural Networks and Deep Learning: A Textbook*, Springer.
[7] Wu, Jianxin, 2017, *Introduction to Convolutional Neural Networks.* Nanjing University, China.
[8] Brown, J. C. & Puckette, M. S., 1992, *An efficient algorithm for the calculation of a constant Q transform.*
[9] nnAudio, 2022, *nnAudio.Spectrogram.CQT1992v2.* Retrieved from https://github.com/KinWaiCheuk/nnAudio
[10] Apoorv, Agnihotri & Batra, Nipun (2020). *Exploring Bayesian Optimization.*
[11] Scikit-optimize contributors, 2020 *skopt.**gp_minimize**.* https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html

VIII. APPENDIX

The codes used for preprocessing the data and for the models are shown in this Appendix. Every line of code will begin with - to differentiate among the other lines.

Preprocessing: The steps followed for the preprocessing of the data are explained in section III:

```
- waves = np.hstack(raw_signals)
- waves = waves / np.max(waves)
- waves = torch.from_numpy(waves).float()
- transform=CQT1992v2(sr=2048, fmin=20,
fmax=1024, hop_length=64, verbose = False)
- image = transform(waves)
- image = np.array(image)
- image = np.transpose(image,(1,2,0))
```

Base model: It consists on a basic one-dimensional CNN with one dense layer with 256 neurons, and the rectified linear unit (ReLU) activation function. Its input shape matches with the dimensions of the Q-transformed signal. Additionally, it has one dense layer with the same number of neurons and ReLU, and the output layer, with sigmoid activation function.
The compilation is done with Adam optimizer with learning rate 0.0002.

```
- from keras.models import Sequential
- from keras.layers import Dense, Dropout,
Flatten, Conv1D, MaxPool1D, BatchNormalization
- from keras.optimizers import RMSprop,Adam
- from nnAudio.Spectrogram import CQT1992v2

- model = Sequential()
- model.add(Conv1D(256, input_shape=(69, 195,),
kernel_size=3, activation='relu'))
- model.add(BatchNormalization())
- model.add(Flatten())
- model.add(Dense(256, activation='relu'))
- model.add(Dense(1, activation='sigmoid'))
- model.compile(optimizer = Adam(lr=2e-4),
loss='binary_crossentropy',metrics=['acc'])
```

Bayesian Optimization model: First of all, the dataset is split in 16 divisions which correspond to the files that comprise it, and they are set to be cycled; every time this vector is called, it will return the next string (referring to the next file):

```
- from itertools import cycle
- next_dir = cycle(['0', '1', '2', '3', '4',
'5', '6', '7', '8', '9', 'a',
'b', 'c', 'd', 'e', 'f'])
```

Then, the hyperparameters to be optimized are defined:

```
- import skopt
- from skopt import gp_minimize
- from skopt.space import Real, Integer,
Categorical
- dim_log_width = Integer(low=4, high=8,
name='log_width')
- dim_layers = Integer(low=1, high=5,
name='layers')
- dimensions = [dim_log_width, dim_layers]
```

gp_minimize requires a function that includes the architecture of the CNN, the training data and the value to be minimized. It was built as follows:

```
- def train(params):
-    dir = next(next_dir)
-    train_idx = labels[labels['id']
.str.startswith(dir)]['id'].values
-    y = labels[labels['id'].isin(train_idx)]
['target'].values
-   log_width, layers = params

-    train_idx, train_Valx = train_test_split
(list(labels[labels['id'].str.startswith(dir)]
.index), test_size=0.2, random_state=2021)
-    train_generator = DataGenerator
('/kaggle/input/g2net-gravitational-wave-
detection/train/', train_idx, labels
[labels['id'].str.startswith(dir)], 256)
-    val_generator = DataGenerator
('/kaggle/input/g2net-gravitational-wave-
detection/train/', train_Valx,
labels[labels['id'].str.startswith(dir)], 256)

-    model = Sequential()
-    model.add(Conv1D(2**log_width, input_shape=
(69, 193,), kernel_size=3, activation='relu'))
-    model.add(BatchNormalization())
-    model.add(Flatten())
-    for i in range(layers):
-        model.add(Dense(2**log_width,
activation='relu'))
-    model.add(Dense(1, activation='sigmoid'))
-    model.compile(optimizer=Adam(lr =2e-4),
loss='binary_crossentropy', metrics=['accuracy'])

-    history = model.fit(train_generator,
validation_data=val_generator, epochs=1)

-    val_loss, val_accuracy = model.evaluate
(val_generator)
-    return -val_accuracy
```

The function **DataGenerator** will not be shown in this Appendix. It covers the preprocessing of the data, Q-transforming the signals and stacking the data in batches. Finally, the prior is defined and the function *gp_minimize* is called:

```
- prior = [5, 1]

- result = gp_minimize(
    func=train,
```

```
    dimensions=dimensions,
    acq_func='PI',
    n_calls=16,
    x0=prior)
```

The optimal hyperparameters are packed inside of the vector *result* and are used to train the model, which has the same structure as the base model, the only changing part are that now it will use the optimal width and dense layers.