Deep residual networks for gravitational wave detection

Paraskevi Nousi,[1] Alexandra E. Koloniari,[2] Nikolaos Passalis,[1] Panagiotis Iosif,[2]
Nikolaos Stergioulas,[2] and Anastasios Tefas[1]

[1]*Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*
[2]*Department of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*

(Received 17 December 2022; accepted 14 June 2023; published 11 July 2023)

Traditionally, gravitational waves are detected with techniques such as matched filtering or unmodeled searches based on wavelets. However, in the case of generic black hole binaries with nonaligned spins, if one wants to explore the whole parameter space, matched filtering can become impractical, which sets severe restrictions on the sensitivity and computational efficiency of gravitational-wave searches. Here, we use a novel combination of machine-learning algorithms and arrive at sensitive distances that surpass traditional techniques in a specific setting. Moreover, the computational cost is only a small fraction of the computational cost of matched filtering. The main ingredients are a 54-layer deep residual network (ResNet), a deep adaptive input normalization (DAIN), a dynamic dataset augmentation, and curriculum learning, based on an empirical relation for the signal-to-noise ratio. We compare the algorithm's sensitivity with two traditional algorithms on a dataset consisting of a large number of injected waveforms of nonaligned binary black hole mergers in real LIGO O3a noise samples. Our machine-learning algorithm can be used in upcoming rapid online searches of gravitational-wave events in a sizeable portion of the astrophysically interesting parameter space. We make our code, *AResGW*, and detailed results publicly available at https://github.com/vivinousi/gw-detection-deep-learning.

DOI: 10.1103/PhysRevD.108.024022

I. INTRODUCTION

In the first three observing runs (O1-O3) of the LIGO-Virgo Collaboration [1,2] (recently joined by Kagra [3]) a number of $\mathcal{O}(90)$ confident gravitational wave (GW) detections were found in the data, including mostly binary black holes (BBH), but also a few binary neutron stars (BNS) and neutron star black hole systems (NSBH) [4–6]. The anticipated significant increase in the number of detections during the fourth observing run (O4), which is scheduled to start in spring of 2023 and even more so during O5 and the observing runs of the planned 3rd-generation detectors (e.g. Cosmic Explorer [7] and Einstein Telescope [8]) will make the application of traditional matched-filtering techniques increasingly costly or impractical [9]. This is both for reasons of computational efficiency, as the goal is to obtain near real-time detection triggers, as well as for reasons of accuracy, since the confident detection of near-threshold systems with random spin directions requires a much larger parameter space than the aligned-spin case. The situation will become even more challenging, if template banks with departures from general relativity (GR) will be included. Unmodeled search algorithms, on the other hand, naturally have a limited sensitivity, depending on the particular GW source.

An attractive solution to the above problem that has been investigated in the last few years is the implementation of machine-learning (ML) methods, such as convolutional neural networks (CNN) or auto-encoders, see, e.g., [10–30] and [31] for a review, but it has been difficult to evaluate the effectiveness of such efforts in a realistic setting. Recently, the first Machine-Learning Gravitational-Wave Mock Data Challenge (MLGWSC-1) was completed [32], defining an objective framework for testing the sensitivity and efficiency of ML algorithms on modeled injections in both Gaussian and real O3a detector noise, in comparison to traditional algorithms. Here, we present the leading ML algorithm in the case of injections of BBH template waveforms in real O3a noise and show that with further improvements it surpasses, for the first time, the results obtained with standard configurations of traditional algorithms in this specific setting. This is achieved for a component mass range between 7–50$M_\odot$ (which corresponds to 70% of the announced events in the cumulative GWTC catalog [6]) and a relatively low false-alarm rate (FAR) as small as one per month. We thus demonstrate that our ML algorithm is sufficiently mature to be implemented in GW search pipelines.

A key indicator for the effectiveness of a GW search algorithm is the sensitive distance it can achieve at a given FAR [14]. Our ML algorithm is a novel combination of several key ingredients, each of which boosts the sensitive distance to higher values. The base algorithm is a 54-layer one-dimensional (1D) deep residual network (ResNet) [33],

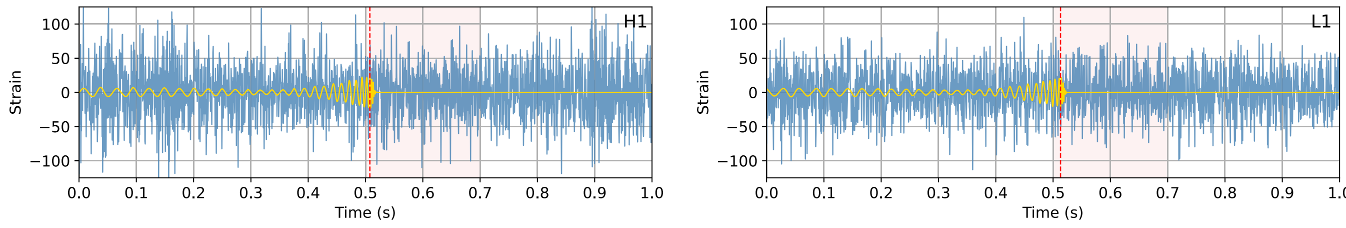

FIG. 1. A representative 2-channel data segment of the training set containing an injection in real O3a noise from the Hanford (H1) detector (left panel) and the Livingston (L1) detector (right panel). The *whitened strain* of a 1 s segment around the time of coalescence is shown. The coalescence times in the detector frames are within the 0.5s–0.7s range (shown as a shaded area). The injected waveform is shown scaled to match the difference between the whitened foreground and background segments. In this example, the component masses are $m_1 = 27.74M_\odot$, $m_2 = 11.50M_\odot$ and the luminosity distance is $d = 1497$ Mpc. The *non-aligned spins* have magnitudes of 0.624 and 0.008, respectively.

which allows for training of a much deeper network, in comparison to the simpler CNN. The second important ingredient that allowed the algorithm to work well at small FAR is the addition of a deep adaptive input normalization (DAIN) [34], to deal with the nonstationary nature of real O3a noise. Next, we obtained improvements by dynamically augmenting the dataset during training. The execution speed was significantly increased with the implementation of a framework-specific, module-based whitening layer, which computes the power spectral density (PSD) over a period of a few seconds in batched tensor format. Finally, we saw a large boost in sensitive distance employing curriculum learning, in which the network was learning the injected waveforms with highest signal-to-noise ratio (SNR) first.

The network was implemented in PYTORCH [35] and its training (including validation) on 12 days of training data was achieved within 31 hours on an A6000 GPU with tensor cores (for 14 epochs). The runtime for the evaluation of one month of test data on the same hardware was less than 2 hours.

II. TRAINING AND TEST DATASETS

The training dataset had a duration of 12 days and comprised real noise from the O3a LIGO run and injections of non-aligned BBH waveforms (corresponding to the assumptions of dataset 4 in [32]). The noise was sampled from parts of O3a that are available from the Gravitational Wave Open Science Center (GWOSC). Only segments with a minimum duration of 2h, where both LIGO detectors registered good quality data, excluding 10s around detections listed in GWTC-2 were included (see Ref. [32] fore more details). Applying these criteria, leaves a dataset with noise for each of the two aLIGO detectors, Hanford (H1) and Livingston (L1), with a total duration of 11 weeks, with a sampling rate of 2048 Hz.

The injected BBH waveforms were generated using the waveform model IMRPhenomXPHM [36], in which a lower-frequency cutoff of 20 Hz was applied. The masses m_1, m_2 of individual components were in the range $7M_\odot - 50M_\odot$ (corresponding to a maximum signal duration of 20s). The signals were uniformly distributed in coalescence phase, polarization, inclination, declination and right ascension (see Ref. [32] for details). The waveforms were not injected uniform in volume, but the *chirp distance* d_c was sampled (instead of the luminosity distance d). This choice increases the number of low-mass systems which can be detected. The chirp distance is defined as

$$d_c = d\left(\frac{\mathcal{M}_{c,0}}{\mathcal{M}_c}\right)^{5/6}, \tag{1}$$

where $\mathcal{M}_c = (m_1 m_2)^{3/5}/(m_1 + m_2)^{1/5}$ is the chirp mass and $\mathcal{M}_{c,0} = 1.4/2^{1/5}M_\odot$ is a fiducial value. The spins of the individual components had an isotropically distributed orientation with a magnitude between 0 and 0.99 (hence, precession effects are present). All higher-order modes up to $(4, -4)$ available in IMRPhenomXPHM are included. A taper window was applied to the start of each waveform. Figure 1 illustrates a representative data segment of the training set.

The training dataset comprised the first 12 days of the 11-week dataset, resulting in 740k noise segments (for each of the two detectors) of duration 1.25s, which constituted the background noise. We then generated a set of 38k waveforms and each one was injected randomly into about 19 different background segments, resulting in 740k foreground segments that contain injections, leading to a balanced training set. The time of coalescence is chosen to randomly fall within the 0.625s and 0.825s mark in each segment. This means that, regardless of the actual duration of the injected signal, only the last cycles corresponding to less than 1s duration are kept for training for any choice of parameters.

Similarly, we created a validation set, based on weeks 4 to 7 of the 11-week dataset, with a total duration of one month, comprising 1850k background noise samples and 1850k foreground samples containing injections (generated from 96k different injections, each injected randomly into about 19 different segments). The injections in the validation set were generated with a different random seed in comparison to the training set.

The main test dataset comprises noise from weeks 8-11 of the 11-week dataset, with a duration of one month[1] and injections with merger times separated by a random time between 24 s to 30 s (which translates to about 96k injections over the whole duration). The code for creating the different datasets was provided by the organizers of MLGWSC-1 and is publicly available. For the main test dataset we used the same random seed of 2514409456 and offset of 0 (first 4 weeks) as in [32]. We created additional test datasets with the same total duration, but different random seeds and offset, in order to assess the variance of our main results.

III. COMPUTATIONAL METHODS

A. Data preprocessing and normalization

Following approaches from traditional matched filtering as applied in GW detections, we first *whiten* the training data [10,37]. As is common in deep learning (DL) methods, we then use an input normalization layer [34,38], before feeding the data to the neural network for classification.

Whitening. The data segments were whitened using the Welch method [39] for computing the PSD and an inverse-spectrum truncation method [40] for smoothing. Both methods were implemented in PYTORCH [35], to allow for batch processing of multiple input segments in a single forward pass, which gave a significant speed up with respect to a CPU implementation. First, we crop 4.25s segments from the original timeseries, for both the training and validation sets, with a stride of 1.4s. An injection is done in a random quarter segment (of 1.25s duration), which is whitened using the 4.25s noise segment surrounding it. After whitening, the first and last 0.125s (0.25s in total) are removed from each sample, *leaving a 1 s sample as input for the neural network.*

Adaptive normalization. As the real noise of the H1 and L1 detectors is nonstationary and affected by a multitude of factors, we opted for an *adaptive input normalization method*, specifically deep adaptive input normalization (DAIN) [34], which is applied before feeding the data to the network. DAIN has been successfully used in tasks that involve non-stationary timeseries, such as financial timeseries forecasting [41].

The goal of this normalization layer is to *learn* how the input timeseries should be normalized. DAIN is used before feeding the input to the subsequent layers and *it is trained by back-propagating the network's gradients* to its parameters, as shown in Fig. 2. Furthermore, DAIN differs from other normalization schemes, since it can dynamically

[1]A smaller test set of one day duration was generated to facilitate a faster comparison of different augmentation and training strategies.

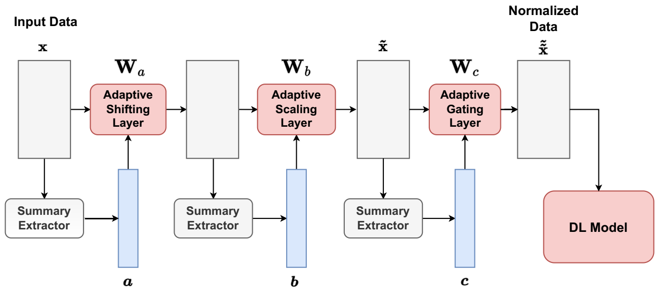

FIG. 2. Deep adaptive normalization (DAIN) is applied before feeding the data to the employed DL model. DAIN involves three adaptive normalization steps, i.e., adaptive shifting, adaptive scaling and adaptive gating, increasing the ability of DL models to handle nonstationary data.

adjust the applied normalization scheme based on the input *at inference time*, allowing for handling nonstationary data. More specifically, DAIN aims to learn how the measurements $\mathbf{x} \in \mathbb{R}^{2048\times 2}$ fed to the neural network should be normalized by appropriately shifting and scaling them:

$$\tilde{\mathbf{x}}_j = (\mathbf{x}_j - \boldsymbol{\alpha}) \oslash \boldsymbol{\beta}, \tag{2}$$

where $\mathbf{x}_j \in \mathbb{R}^2$ refers to the jth observation (out of 2048 included in the current window), $\oslash$ is the Hadamard (entrywise) division operator and $\boldsymbol{\beta}$ is a scaling operator. To this end, we first build a summary representation of the current window as:

$$\mathbf{a} = \frac{1}{2048}\sum_{j=1}^{2048} \mathbf{x}_j \in \mathbb{R}^2. \tag{3}$$

This average is used to estimate the mode of the distribution that generated the observed data. Then, DAIN learns how to appropriate shift the data based on the observed mode by estimating the value for the shifting operator $\boldsymbol{\alpha}$ as:

$$\boldsymbol{\alpha} = \mathbf{W}_a \mathbf{a} \in \mathbb{R}^2, \tag{4}$$

where $\mathbf{W}_a \in \mathbb{R}^{2\times 2}$ are the trainable parameters of the shifting operator.

After this shifting process, the data are also scaled by employing the scaling operator $\boldsymbol{\beta}$, as shown in (2). Again, we calculate a summary representation as:

$$b_k = \sqrt{\frac{1}{2048}\sum_{j=1}^{2048} (x_{j,k} - \alpha_k)^2}, \qquad k = 1, 2, \tag{5}$$

and then define the scaling operator as:

$$\boldsymbol{\beta} = \mathbf{W}_b \mathbf{b} \in \mathbb{R}^2, \tag{6}$$

where $\mathbf{W}_b \in \mathbb{R}^{2\times 2}$ are its parameters.

Finally, the shifted and scaled observations are fed to an *adaptive gating layer* which aims to appropriately modulate the features according to their usefulness for the task at hand as:

$$\tilde{\tilde{\mathbf{x}}}_j = \tilde{\mathbf{x}}_j \odot \boldsymbol{\gamma}, \tag{7}$$

where $\odot$ is the Hadamard (entrywise) multiplication operator and

$$\boldsymbol{\gamma} = \mathrm{sigm}(\mathbf{W}_c \mathbf{c} + \mathbf{d}) \in \mathbb{R}^2, \tag{8}$$

$\mathrm{sigm}(x) = 1/(1+\exp(-x))$ is the sigmoid function, $\mathbf{W}_c \in \mathbb{R}^{2\times 2}$ and $\mathbf{d} \in \mathbb{R}^2$ are the parameters of the gating layer that are learned through back-propagation. The updated summary representation $\mathbf{c}$ is calculated as:

$$\mathbf{c} = \frac{1}{2048} \sum_{j=1}^{2048} \tilde{\mathbf{x}}_j \in \mathbb{R}^2. \tag{9}$$

The nonlinearity introduced by this layer allows for the suppression of the normalized features during inference. This can help to reduce the effect of features that could harm the generalization abilities of the network. The parameters introduced by the DAIN layer are fitted using gradient descent:

$$\Delta(\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c, \mathbf{d}, \mathbf{W}) = -\eta\left(\eta_a \frac{\partial \mathcal{L}}{\partial \mathbf{W}_a}, \eta_b \frac{\partial \mathcal{L}}{\partial \mathbf{W}_b}, \eta_c \frac{\partial \mathcal{L}}{\partial \mathbf{W}_c}, \eta_c \frac{\partial \mathcal{L}}{\partial \mathbf{d}}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right) \tag{10}$$

where $\mathcal{L}$ denotes the final loss function of the network and $\mathbf{W}$ its weights. Separate learning rates can be used for the parameters of each sublayer, i.e., η_a, η_b, and η_c, if necessary, to ensure the stability of the training process.

B. Deep residual networks

Residual neural networks [33] use so-called *skip connections* to improve training, by allowing gradients to better reach the earliest layers of a neural network architecture, effectively solving the vanishing gradient problem [42] and leading to more effective training as the number of layers increases, especially when *paired with carefully designed training methods* [43]. This allows for much deeper networks to be trained, in comparison to simple CNN. Depth in neural networks has been linked to higher levels of semantic information, and better performance in tasks like image recognition [44], detection [45], etc.

We designed a deep residual network, based on 1D convolutions, for the purpose of the binary classification of 1s long (i.e. 2×2048-dimensional)[2] segments into positive (containing an injection) or negative (pure noise) segments. Our network has a depth of 54 layers, grouped into 27 *blocks* comprising two convolutional layers with a varying number of filters. The network's backbone (except for the final two convolutional layers) is summarized in Table I, where D denotes the dimensionality of the input tensor. Blocks 5, 8, 11, 14 and 17 are *2-strided*, meaning that dimensionality is halved and an additional layer is used in the residual connection.

[2]We remind that the sampling rate was 2048 Hz and there are two channels, one for each of the aLIGO detectors.

TABLE I. Feature extraction backbone of our ResNet54-like architecture. The first column lists the number of residual blocks, totalling 27, each consisting of *two* convolutional layers. The second column lists the number of filters in each corresponding block. The third column indicates those blocks that are 2-strided (see text for details) and the last column displays the dimensionality D of the input tensor for each block.

Residual blocks	Filters	Strided	Input D
4	8		2×2048
1	16	✓	8×2048
2	16		16×1024
1	32	✓	16×1024
2	32		32×512
1	64	✓	32×512
2	64		64×256
1	64	✓	64×256
2	64		64×128
1	64	✓	64×128
2	64		64×64
5	32		64×64
3	16		32×64

A graphical depiction of the network architecture is displayed in Fig. 3. The output of a residual block of layers can be written schematically as:

$$\mathbf{g} = f(\mathbf{x}) + h(\mathbf{x}), \tag{11}$$

where $f(\mathbf{x})$ is a block of two convolutional layers, and $\mathbf{x}$ the input (i.e. the 2×2048-dimensional timeseries) to its first layer. The *residual function* $h(\mathbf{x})$ is either the identity function, $h(\mathbf{x}) = \mathbf{x}$, or a strided convolutional layer. Each convolutional layer is followed by batch normalization and a ReLU activation function. Two final individual convolutional layers gradually reduce the output to a binary outcome (noise plus injected waveform vs. noise only).

C. Training

The network is fed with the training dataset (925k 1s segments of noise and an equal number of 1s segments containing injections) using a minibatch size of 400 (each minibatch also contains an equal number of only noise and noise plus injection segments, randomly chosen from the whole dataset). The Adam optimizer [46] was used during

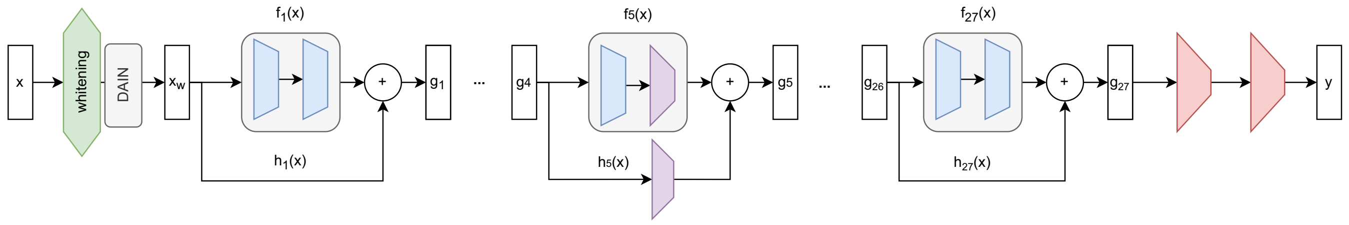

FIG. 3. Description of our residual network architecture. The input **x** is 2×2048-dimensional. There are 27 blocks comprising two convolutional layers. In five of these blocks, the dimensionality is halved (stride 2, shown in purple) and an additional layer is used in the residual connection. Finally, there are two individual convolutional layers which gradually decrease the number of channels down to two, corresponding to the binary classification targets (noise plus injected waveform vs. noise only) in the output.

backpropagation[3] and the *regularized* binary cross entropy (a variant of the cross entropy loss function that is designed to be finite [27]) was used as the *objective function*.

Dynamic augmentation. During training, we also employed dynamic augmentation, by randomly replacing a noise segment for the L1 channel, by another segment. This augmentation is performed with a random probability of 40% and virtually increases the total number of samples that the network sees during training.

SNR-based curriculum learning. We implemented a learning strategy, so that the network is first trained on the loudest injections and only at later epochs it is trained on weaker injections. To achieve this, we used the optimal signal-to-noise ratio (SNR) of the injected signal

$$\mathrm{SNR} = 2\sqrt{\int_0^\infty df \frac{|\tilde{h}(f)|^2}{S_n(f)}}, \tag{12}$$

where $\tilde{h}(f)$ is the amplitude of the Fourier transform of the injected signal and $S_n(f)$ is the PSD of the detector's noise. An accurate computation of the SNR of each segment during training would add some computational overhead, so we chose to construct an empirical relation that gives an approximate prediction of the SNR. Specifically, by varying only the chirp mass, the distance and the inclination ι (and setting all other parameters to some fixed values), we arrived at the following approximate empirical relation:

$$\mathrm{SNR} = \frac{1261\ \mathrm{Mpc}}{D}\left(\frac{\mathcal{M}_c}{M_\odot}\right)^{5/7}[0.7 + 0.3\cos(2\iota)]. \tag{13}$$

Figure 4 shows a comparison of the distribution of the optimal SNR (obtained through Eq. (12) using the PSD of the Hanford detector) for a subset of 10000 randomly

[3]Note that the first two training epochs are a warm up period, during which the learning rate is annealed from 10^{-8} to 4.5×10^{-3}.

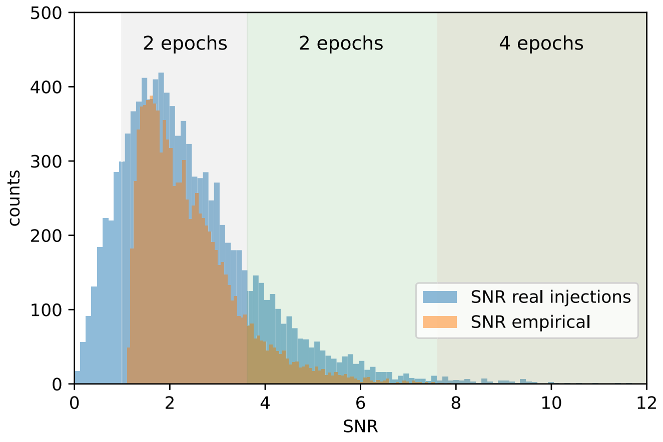

FIG. 4. Comparison of the SNR histogram computed with the empirical relation Eq. (13) to the optimal SNR of the injections (using a random subset of 10000 injections). A similar distribution is obtained. The shaded areas correspond to the restricted values of the SNR used in the first eight epochs of the *learning strategy* (from right to left, see Table II).

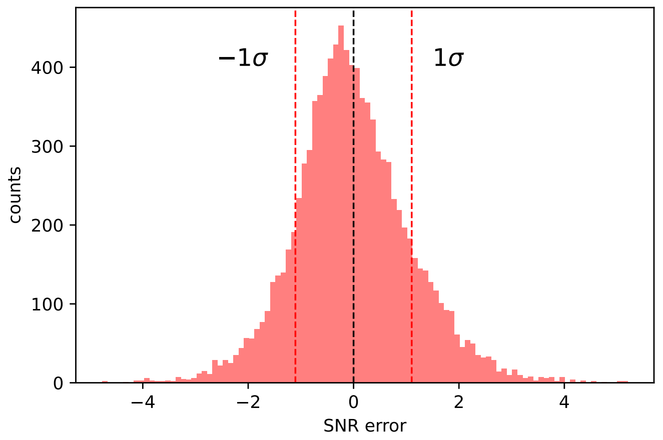

FIG. 5. Histogram of the errors between the SNR computed with the empirical relation Eq. (13) to the optimal SNR of the same injections as in Fig. 4. The errors follow a normal distribution with standard deviation $\sigma = 1.1$.

TABLE II. Learning strategy: During the first 4 epochs the network learns only strong signals with estimated optimal SNR larger than 7.63. Next, the network gradually learns weaker signals (for the subsequent epochs the estimated SNR range is shown). After the tenth epoch, the network learns all signals.

Epochs	Min(SNR)	Max(SNR)
4	7.63	100
2	3.63	100
2	1	3.63
2	1	7.63
Remaining	0	∞

chosen injections to the approximate distribution as obtained through Eq. (13). The two distributions are similar, their difference stemming from the fact that the actual SNR depends on many more parameters (sky location, spins etc.). To quantify their similarity, in Fig. 5 we show the histogram of the errors between the two distributions (actual and empirical SNR) of Fig. 4. The errors follow a normal distribution with a standard deviation of $\sigma = 1.1$, which we found to be adequate for our purpose.

We start the training with easily recognizable signals with large estimated SNR, for the first four epochs. Next, the network is trained on gradually weaker signals and after the tenth period it learns all signals in the training set. The detailed SNR-based training strategy is shown in Table II and the first eight epochs are also depicted in Fig. 4 in relation to the distribution of the SNR.

Fig. 6 shows the evolution of the loss function as a function of the training epoch. Due to the adopted learning strategy, initial losses are small. When the network is finally learning all signals (after ten epochs), the loss of the training set converges with that of the validation set. The corresponding evolution of the accuracy as a function of the training epoch is shown in Fig. 7.

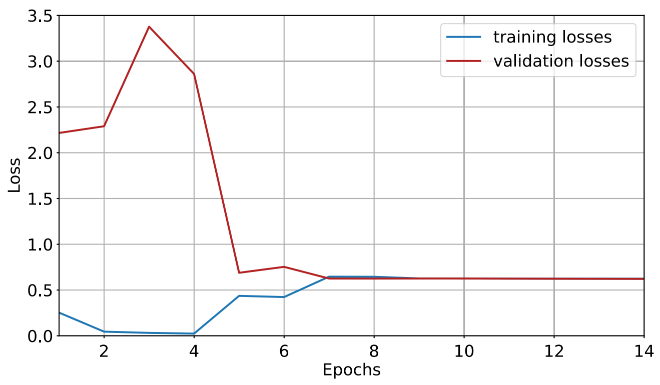

FIG. 6. Loss as a function of epoch. Due to the adopted learning strategy (strong signals first), initial losses are small. After ten epochs, when the network is learning all signals, the loss of the training set converges with that of the validation set.

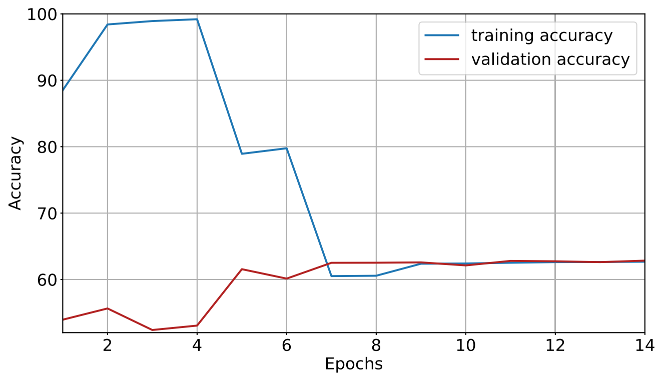

FIG. 7. Accuracy as a function of epoch. Due to the adopted learning strategy (strong signals first), initial accuracy is high. After ten epochs, when the network is learning all signals, the accuracy of the training set converges with that of the validation set.

IV. DETECTION OF BBH INJECTIONS IN REAL NOISE

The trained network is run on segments of the test dataset described in Sec. II and produces a binary output, corresponding to the probabilities that the waveform is present or that the segment contains only noise. The first output (waveform present) serves as a ranking statistic $\mathcal{R}$ with values between 0 and 1 for each 1s segment. When $\mathcal{R} > 0.5$, a positive outcome is recorded.

During deployment on the test dataset, we crop an amount of 4.25 seconds of data, from which we compute the PSD in real time. To increase speed, the Welch method for computing the PSD was implemented in PYTORCH [35], whereas whitening is implemented as the first layer of the final detection module. Then, the input is reorganized as a batch of 1.25s segments with a stride of 0.1s, i.e., with an overlap of 1.15s. This batch is whitened with a single forward pass of the implemented whitening module. The last valid segment starts at the 3s mark, which leads to 31 segments of duration 1.25s. After whitening, the 1.25s whitened segments are cropped unilaterally to 1s total duration. Notice that when an injection is present, multiple overlapping segments can have $\mathcal{R} > 0.5$. Because of this, we cluster positives and report any positives that are detected within a time span of 0.3s as a *single detection* (see Fig. 8 for a representative example).

After the deployment, the output is evaluated every 0.1s comparing it to the known injection times in the test dataset. If a positive output differs less than 0.3s from the nominal merger time for a particular injection, then it is classified as a true positive, otherwise as a false positive.

To evaluate the effectiveness of the search algorithm, we first need to determine the false alarm rate $\mathcal{F}$ as a function of the ranking statistic, i.e., the function $\mathcal{F}(\mathcal{R})$. Next, the sensitive distance of the search is determined as a function of the ranking statistic, which finally produces a relation

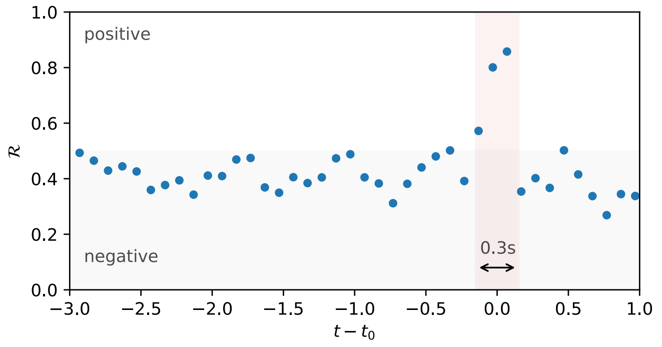

FIG. 8. Representative example of the ranking statistic for overlapping segments (with a stride of 0.1s). Segments with $\mathcal{R} < 0.5$ are classified as negatives. Segments with $\mathcal{R} > 0.5$ that cluster within 0.3s of a known injection at time t_0 are reported as a single true positive.

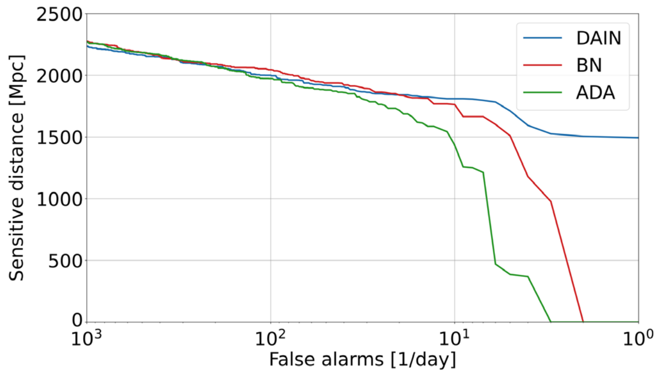

FIG. 9. Sensitive distance vs. false alarm rate for three different input normalizations: DAIN [34], batch normalization (BN) [38] and adaptive batch normalization (ADA) [47] (for a shorter test dataset of duration of one day). In contrast to BN and ADA, DAIN maintains a good sensitive distance down to the lowest FAR.

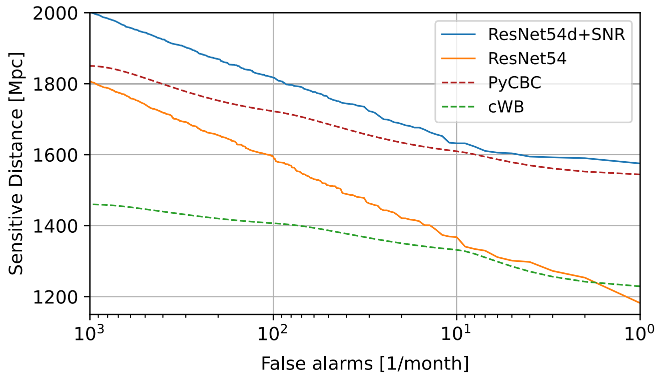

FIG. 10. Sensitive distance vs. false alarm rate for our best model (ResNet54d + SNR), in comparison with the simpler setup (ResNet54) used in [32] and two widely used algorithms for GW detection, coherent waveburst (cWB) and PYCBC (the latter using algned-spin templates only, see text for details). All codes were run on the same test dataset established in [32]. Our best model surpasses the performance of the other algorithms at all FAR in this setting (notice that the PYCBC run was based on a template bank with only aligned-spin waveforms).

between the sensitive distance and the $\mathcal{F}$ (see Ref. [32] for definitions and the detailed methodology).

In Fig. 9 we demonstrate the importance of using DAIN [34] as the input normalization, by comparing the sensitive distance at different FAR (for a shorter test dataset of duration of one day) to the corresponding results obtained with either batch normalization (BN) [38] or adaptive batch normalization (ADA) [47]. Whereas the three methods achieve similar sensitive distances at FAR > 30/d, the performance of BN and ADA deteriorates, dropping to 0 at FAR of several per day. In contrast, the input normalization with DAIN allows the network to maintain a good sensitive distance down to the lowest FAR (notice that for the comparison shown in Fig. 5 we did not activate the SNR-based curriculum learning, to isolate the effect of the input normalization).

Fig. 10 displays the sensitive distance as a function of FAR for our best model (ResNet54d + SNR), in comparison with a simpler setup (ResNet54[4]) and two widely used algorithms for GW detection, coherent waveburst (cWB), a waveform model-agnostic search pipeline for GW signals based on the constrained likelihood method [48–50] and PYCBC [51], based on a standard configuration of the archival search for compact-binary mergers [52]. The results of cWB and PYCBC shown in Fig. 10 are taken from [32] (where they were obtained on the same test dataset). cWB uses wavelets instead of specific models of the waveforms it searches for, which naturally does not allow it to reach ideal fitting factors. It was recently enhanced with machine-learning techniques [53] (for details on the particular cWB setup used on the test dataset, see Ref. [32]). PYCBC implements matched filtering of waveform templates, but in [32,52] only aligned-spin templates were used (for more general waveforms, the method could become computationally too costly). Since the injections in the test dataset are based on more general waveforms, this particular PYCBC search cannot reach ideal fitting factors, leaving room for other algorithms to surpass it.

As seen in Fig. 10, our best model, which includes the SNR-based curriculum learning, surpasses the PYCBC results at all FAR. The sensitive distance as a function of FAR is nearly level at the lowest FAR of 1 per month, indicating that our algorithm may maintain good performance even when extended to lower FAR. Our best model also significantly exceeds the sensitivity of the unmodeled cWB search.

[4]This setup has half the number of filters, as was the case in [32], and does not include the SNR-based curriculum learning.

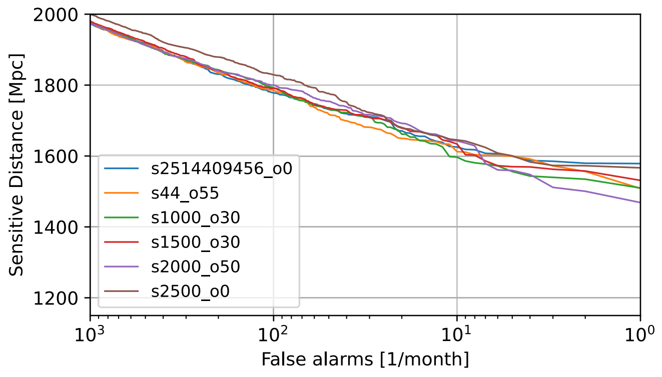

FIG. 11. Variance of the sensitive distance vs. false alarm rate for our best model, using various test datasets of one month duration, that differ by the random seed used for the injections and the offset in the starting time. The first curve (seed 2514409456 and offset 0) corresponds to the case of the test dataset of the MLGWSC-1 challenge (blue curve in Fig. 10). At a FAR of 1/month, the variance is only $\pm 3\%$ of the average.

V. SUMMARY AND DISCUSSION

We used a novel combination of ML algorithms and arrived at sensitive distances for injected BBH GW signals that surpass traditional algorithms, even at small false alarm rates. The main ingredients are a 54-layer deep residual network (ResNet), a deep adaptive input normalization (DAIN), a dynamic dataset augmentation, and curriculum learning, based on an empirical relation for the signal-to-noise ratio. Our best ML model surpasses the sensitive distance achieved with traditional algorithms in a specific setting that uses a dataset consisting of a large number of injected nonaligned spin waveforms in real LIGO O3a noise samples. The matched-filtering PYCBC run on the same dataset included only aligned-spin templates and could therefore not reach optimal sensitivity.

We examined the variance of our main result using additional test datasets with different offsets and starting times, see Fig. 11. The different sensitivity curves we obtain are within a few percent from the average, demonstrating that our trained neural network runs robustly on differently datasets, with consistent sensitivity.

Our ML model operates at a fraction of the computational cost of matched filtering (see Ref. [32] for detailed comparisons) and it can thus be deployed in upcoming rapid online searches of gravitational-wave events in a sizeable portion of the astrophysically interesting parameter space.

The performance of our ML model could be further improved using a more accurate empirical relation for the signal-to-noise ratio and a more fine-tuned curriculum learning. We are planning to extend the training dataset to include lower or higher black hole masses. It is likely that dedicated networks will need to be trained to cover the edges of the parameter space. Furthermore, we are planning to include additional channels for the operating advanced Virgo [2] and Kagra [3] detectors. In the future, additional detectors, such as the planned LIGO-India [54] detector could be included in the training of the network.

Our code, *AResGW*, and detailed results are publicly available at [55].

ACKNOWLEDGMENTS

We are grateful to Agata Trovato, Marlin Schaefer and Ondrej Zelenka for useful comments on the manuscript. We are indebted to the organizers of the MLGWSC-1 challenge, for all their work and for providing routines for creating different datasets. We acknowledge the support provided by the IT Center of the Aristotle University of Thessaloniki (AUTh) as our results have been produced using the AUTh High Performance Computing Infrastructure and Resources. We also acknowledge support from COST Action (European Cooperation in Science and Technology) CA17137 (G2Net). This research has made use of data or software obtained from the Gravitational Wave Open Science Center (GWOSC) a service of LIGO Laboratory, the LIGO Scientific Collaboration, the Virgo Collaboration, and KAGRA. LIGO Laboratory and Advanced LIGO are funded by the United States National Science Foundation (NSF) as well as the Science and Technology Facilities Council (STFC) of the United Kingdom, the Max-Planck-Society (MPS), and the State of Niedersachsen/Germany for support of the construction of Advanced LIGO and construction and operation of the GEO600 detector. Additional support for Advanced LIGO was provided by the Australian Research Council. Virgo is funded, through the European Gravitational Observatory (EGO), by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale di Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by institutions from Belgium, Germany, Greece, Hungary, Ireland, Japan, Monaco, Poland, Portugal, Spain. The construction and operation of KAGRA are funded by Ministry of Education, Culture, Sports, Science and Technology (MEXT), and Japan Society for the Promotion of Science (JSPS), National Research Foundation (NRF) and Ministry of Science and ICT (MSIT) in Korea, Academia Sinica (AS) and the Ministry of Science and Technology (MoST) in Taiwan.

[1] J. Aasi *et al.* (LIGO Scientific Collaboration), Advanced LIGO, Classical Quantum Gravity **32**, 074001 (2015).
[2] F. Acernese *et al.* (Virgo Collaboration), Advanced Virgo: A second-generation interferometric gravitational wave detector, Classical Quantum Gravity **32**, 024001 (2014).
[3] T. Akutsu *et al.*, Overview of KAGRA: Detector design and construction history, Prog. Theor. Exp. Phys. **2021**, 05A101 (2020).
[4] B. P. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), Gwtc-1: A Gravitational-Wave Transient Catalog of Compact Binary Mergers Observed by LIGO and Virgo during the First and Second Observing Runs, Phys. Rev. X **9**, 031040 (2019).
[5] R. Abbott *et al.* (LIGO Scientific and Virgo Collaborations), GWTC-2: Compact Binary Coalescences Observed by LIGO and Virgo During the First Half of the Third Observing Run, Phys. Rev. X **11**, 021053 (2021).
[6] R. Abbott *et al.* (The LIGO Scientific, the Virgo and the KAGRA Collaborations), GWTC-3: Compact Binary Coalescences Observed by LIGO and Virgo During the Second Part of the Third Observing Run, arXiv:2111.03606 [Phys. Rev. X (to be published)].
[7] D. Reitze *et al.*, Cosmic Explorer: The U.S. contribution to gravitational-wave astronomy beyond LIGO, Bull. Am. Astron. Soc. **51**, 035 (2019), https://baas.aas.org/dash/pub/2020n7i035/overview?collectionSlug=vol-51-issue-7.
[8] M. Maggiore, C. Van Den Broeck, N. Bartolo, E. Belgacem, D. Bertacca, M. A. Bizouard, M. Branchesi, S. Clesse, S. Foffa, J. García-Bellido, S. Grimm, J. Harms, T. Hinderer, S. Matarrese, C. Palomba, M. Peloso, A. Ricciardone, and M. Sakellariadou, Science case for the Einstein telescope, J. Cosmol. Astropart. Phys. 03 (2020) 050.
[9] P. Couvares, I. Bird, E. Porter, S. Bagnasco, M. Punturo, D. Reitze, S. Katsanevas, T. Kajita, V. Kalogera, H. Lueck, D. McClelland, S. Rowan, G. Sanders, B. S. Sathyaprakash, D. Shoemaker, and J. van den Brand, Gravitational wave data analysis: Computing challenges in the 3G Era, arXiv:2111.06987.
[10] H. Gabbard, M. Williams, F. Hayes, and C. Messenger, Matching Matched Filtering with Deep Networks for Gravitational-Wave Astronomy, Phys. Rev. Lett. **120**, 141103 (2018).
[11] D. George and E. A. Huerta, Deep neural networks to enable real-time multimessenger astrophysics, Phys. Rev. D **97**, 044039 (2018).
[12] T. D. Gebhard, N. Kilbertus, I. Harry, and B. Schölkopf, Convolutional neural networks: A magic bullet for gravitational-wave detection?, Phys. Rev. D **100**, 063015 (2019).
[13] R. Corizzo, M. Ceci, E. Zdravevski, and N. Japkowicz, Scalable auto-encoders for gravitational waves detection from time series data, Expert Syst. Appl. **151**, 113378 (2020).
[14] M. B. Schäfer, F. Ohme, and A. H. Nitz, Detection of gravitational-wave signals from binary neutron star mergers using machine learning, Phys. Rev. D **102**, 063015 (2020).
[15] H. Wang, S. Wu, Z. Cao, X. Liu, and J.-Y. Zhu, Gravitational-wave signal recognition of LIGO data by deep learning, Phys. Rev. D **101**, 104003 (2020).
[16] P. G. Krastev, Real-time detection of gravitational waves from binary neutron stars using artificial neural networks, Phys. Lett. B **803**, 135330 (2020).
[17] V. Skliris, M. R. K. Norman, and P. J. Sutton, Real-time detection of unmodelled gravitational-wave transients using convolutional neural networks, arXiv:2009.14611.
[18] Y.-C. Lin and J.-H. P. Wu, Detection of gravitational waves using Bayesian neural networks, Phys. Rev. D **103**, 063034 (2021).
[19] E. A. Huerta, A. Khan, X. Huang, M. Tian, M. Levental, R. Chard, W. Wei, M. Heflin, D. S. Katz, V. Kindratenko, D. Mu, B. Blaiszik, and I. Foster, Accelerated, scalable and reproducible AI-driven gravitational wave detection, Nat. Astron. **5**, 1062 (2021).
[20] T. Marianer, D. Poznanski, and J. X. Prochaska, A semi-supervised machine learning search for never-seen gravitational-wave sources, Mon. Not. R. Astron. Soc. **500**, 5408 (2021).
[21] W. Wei, A. Khan, E. A. Huerta, X. Huang, and M. Tian, Deep learning ensemble for real-time gravitational wave detection of spinning binary black hole mergers, Phys. Lett. B **812**, 136029 (2021).
[22] S. Jadhav, N. Mukund, B. Gadre, S. Mitra, and S. Abraham, Improving significance of binary black hole mergers in Advanced LIGO data using deep learning: Confirmation of GW151216, Phys. Rev. D **104**, 064051 (2021).
[23] P. Chaturvedi, A. Khan, M. Tian, E. A. Huerta, and H. Zheng, Inference-optimized ai and high performance computing for gravitational wave detection at scale, Front. Artif. Intell. **5**, 828672 (2022).
[24] S. Choudhary, A. More, S. Suyamprakasam, and S. Bose, SiGMa-Net: Deep learning network to distinguish binary black hole signals from short-duration noise transients, Phys. Rev. D **107**, 024030 (2023).
[25] M. B. Schäfer and A. H. Nitz, From one to many: A deep learning coincident gravitational-wave search, Phys. Rev. D **105**, 043003 (2022).
[26] F. P. Barone, D. Dell'Aquila, and M. Russo, A novel multi-layer modular approach for real-time gravitational-wave detection, arXiv:2206.06004.
[27] M. B. Schäfer, O. Zelenka, A. H. Nitz, F. Ohme, and B. Brügmann, Training strategies for deep learning gravitational-wave searches, Phys. Rev. D **105**, 043002 (2022).
[28] G. Baltus, J. Janquart, M. Lopez, H. Narola, and J.-R. Cudell, Convolutional neural network for gravitational-wave early alert: Going down in frequency, Phys. Rev. D **106**, 042002 (2022).
[29] M. Andrews, M. Paulini, L. Sellers, A. Bobrick, G. Martire, and H. Vestal, DeepSNR: A deep learning foundation for offline gravitational wave detection, arXiv:2207.04749.
[30] C. Verma, A. Reza, G. Gaur, D. Krishnaswamy, and S. Caudill, Can convolution neural networks be used for detection of gravitational waves from precessing black hole systems?, arXiv:2206.12673.
[31] E. Cuoco *et al.*, Enhancing gravitational-wave science with machine learning, Mach. Learn. **2**, 011002 (2020).

[32] M. B. Schäfer *et al.*, MLGWSC-1: The first machine learning gravitational-wave search mock data challenge, Phys. Rev. D **107**, 023021 (2023).

[33] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778, 10.1109/CVPR.2016.90.

[34] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, Deep adaptive input normalization for time series forecasting, IEEE Trans. Neural Netw. Learn. **31**, 3760 (2019).

[35] A. Paszke *et al.*, PYTORCH: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., New York, USA, 2019), pp. 8024–8035.

[36] G. Pratten, C. García-Quirós, M. Colleoni, A. Ramos-Buades, H. Estellés, M. Mateu-Lucena, R. Jaume, M. Haney, D. Keitel, J. E. Thompson, and S. Husa, Computationally efficient models for the dominant and subdominant harmonic modes of precessing binary black holes, Phys. Rev. D **103**, 104056 (2021).

[37] S. A. Usman, A. H. Nitz, I. W. Harry, C. M. Biwer, D. A. Brown, M. Cabero, C. D. Capano, T. Dal Canton, T. Dent, S. Fairhurst *et al.*, The PYCBC search for gravitational waves from compact binary coalescence, Classical Quantum Gravity **33**, 215004 (2016).

[38] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *International Conference on Machine Learning* (PMLR, 2015), pp. 448–456.

[39] S. Villwock and M. Pacas, Application of the welch-method for the identification of two-and three-mass-systems, IEEE Trans. Ind. Electron. **55**, 457 (2008).

[40] B. Allen, W. G. Anderson, P. R. Brady, D. A. Brown, and J. D. Creighton, Findchirp: An algorithm for detection of gravitational waves from inspiraling compact binaries, Phys. Rev. D **85**, 122006 (2012).

[41] N. Passalis, J. Kanniainen, M. Gabbouj, A. Iosifidis, and A. Tefas, Forecasting financial time series using robust deep adaptive input normalization, J. Signal Process. Syst. **93**, 1235 (2021).

[42] B. Hanin, Which neural net architectures give rise to exploding and vanishing gradients?, Adv. Neural Inf. Process. Syst. **31** (2018), https://proceedings.neurips.cc/paper/2018/hash/13f9896df61279c928f19721878fac41-Abstract.html.

[43] R. Wightman, H. Touvron, and H. Jégou, Resnet strikes back: An improved training procedure in timm, arXiv:2110.00476.

[44] N. Takahashi, M. Gygli, and L. Van Gool, Aenet: Learning deep audio features for video analysis, IEEE Trans. Multimedia **20**, 513 (2017).

[45] H. Xu, X. Lv, X. Wang, Z. Ren, N. Bodla, and R. Chellappa, Deep regionlets for object detection, in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 798–814, 10.1007/978-3-030-01252-6_49.

[46] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[47] Q. Chen, J. Xu, and V. Koltun, Fast image processing with fully-convolutional networks, in *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2516–2525, 10.1109/ICCV.2017.273.

[48] S. Klimenko, G. Vedovato, M. Drago, F. Salemi, V. Tiwari, G. A. Prodi, C. Lazzaro, K. Ackley, S. Tiwari, C. F. Da Silva, and G. Mitselmakher, Method for detection and reconstruction of gravitational wave transients with networks of advanced detectors, Phys. Rev. D **93**, 042004 (2016).

[49] M. Drago, S. Klimenko, C. Lazzaro, E. Milotti, G. Mitselmakher, V. Necula, B. O'Brian, G. Prodi, F. Salemi, M. Szczepanczyk, S. Tiwari, V. Tiwari, G. V, G. Vedovato, and I. Yakushin, coherent waveburst, a pipeline for unmodeled gravitational-wave data analysis, SoftwareX **14**, 100678 (2021).

[50] S. Klimenko, G. Vedovato, V. Necula, F. Salemi, M. Drago, R. Poulton, E. Chassande-Mottin, V. Tiwari, C. Lazzaro, B. O'Brian, M. Szczepanczyk, S. Tiwari, and V. Gayathri, cwb pipeline library: 6.4.1 (2021).

[51] A. Nitz *et al.*, gwastro/pycbc: v2.0.5 release of pycbc (2022).

[52] A. H. Nitz, S. Kumar, Y.-F. Wang, S. Kastha, S. Wu, M. Schäfer, R. Dhurkunde, and C. D. Capano, 4-OGC: Catalog of gravitational waves from compact-binary mergers, Astrophys. J. **946**, 59 (2023).

[53] T. Mishra, B. O'Brien, M. Szczepańczyk, G. Vedovato, S. Bhaumik, V. Gayathri, G. Prodi, F. Salemi, E. Milotti, I. Bartos, and S. Klimenko, Search for binary black hole mergers in the third observing run of Advanced LIGO-Virgo using coherent waveburst enhanced with machine learning, Phys. Rev. D **105**, 083018 (2022).

[54] M. Saleem, J. Rana, V. Gayathri, A. Vijaykumar, S. Goyal, S. Sachdev, J. Suresh, S. Sudhagar, A. Mukherjee, G. Gaur, B. Sathyaprakash, A. Pai, R. X. Adhikari, P. Ajith, and S. Bose, The science case for LIGO-India, Classical Quantum Gravity **39**, 025004 (2022).

[55] AResGW code (2022), https://github.com/vivinousi/gw-detection-deep-learning.