Deep Residual Networks for Gravitational Wave Detection

Paraskevi Nousi, Alexandra E. Koloniari, Nikolaos Passalis, Panagiotis Iosif, Nikolaos Stergioulas, Anastasios Tefas

Aristotle University of Thessaloniki

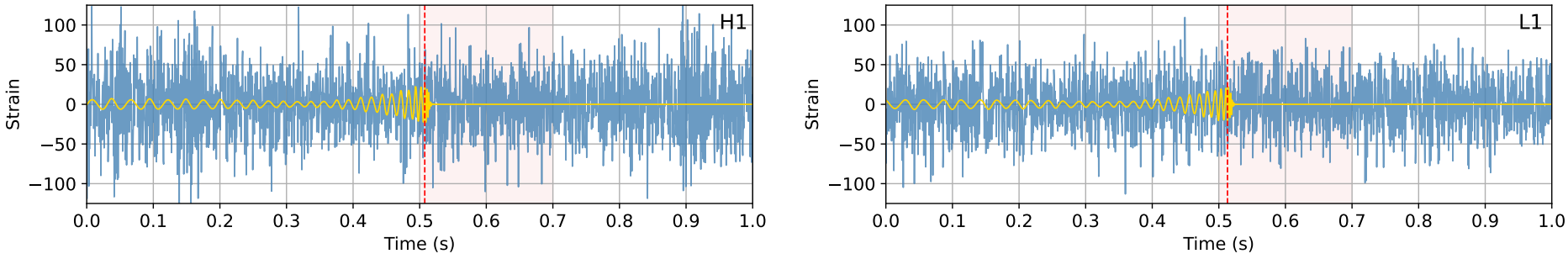
March 2023

Overview

1. GW Detection
2. MLGWSC
3. AresGW
4. Evaluation

GW Detection

GW Detection



- A representative 2-channel data segment 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

MLGWSC

MLGWSC-1

Open Access

First machine learning gravitational-wave search mock data challenge

Marlin B. Schäfer, Ondřej Zelenka, Alexander H. Nitz, He Wang, Shichao Wu, Zong-Kuan Guo, Zhoujian Cao, Zhixiang Ren, Paraskevi Nousi, Nikolaos Stergioulas, Panagiotis Iosif, Alexandra E. Koloniari, Anastasios Tefas, Nikolaos Passalis, Francesco Salemi, Gabriele Vedovato, Sergey Klimenko, Tanmaya Mishra, Bernd Brügmann, Elena Cuoco, E. A. Huerta, Chris Messenger, and Frank Ohme

Phys. Rev. D **107**, 023021 – Published 27 January 2023

https://arxiv.org/abs/2209.11146

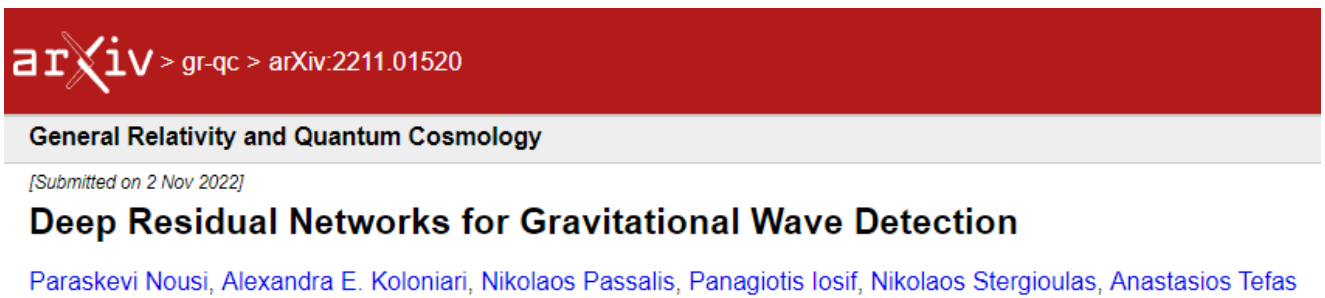
MLGWSC-1

- `github.com/gwastro/ml-mock-data-challenge-1`
- objective characterization of ML GW detection algorithms
- allow for easy comparison between different search algorithms
- 4 datasets of increasing complexity
- common code to generate data:
  - background (Gaussian or O3a noise)
  - foreground (same noise + injected waveforms)
  - parameters of injected signals (IMRPhenomXPHM model: non-aligned, spinning BBH waveforms)
- evaluation set generated with the same code, with random seed withheld from participants algorithms

AresGW

Our method

- Four basic components:
  1. Large training set
  2. Whitening process implemented as neural layer
  3. Adaptive input normalization
  4. Deep Residual networks
  5. Data augmentation
  6. SNR approximation based curriculum learning



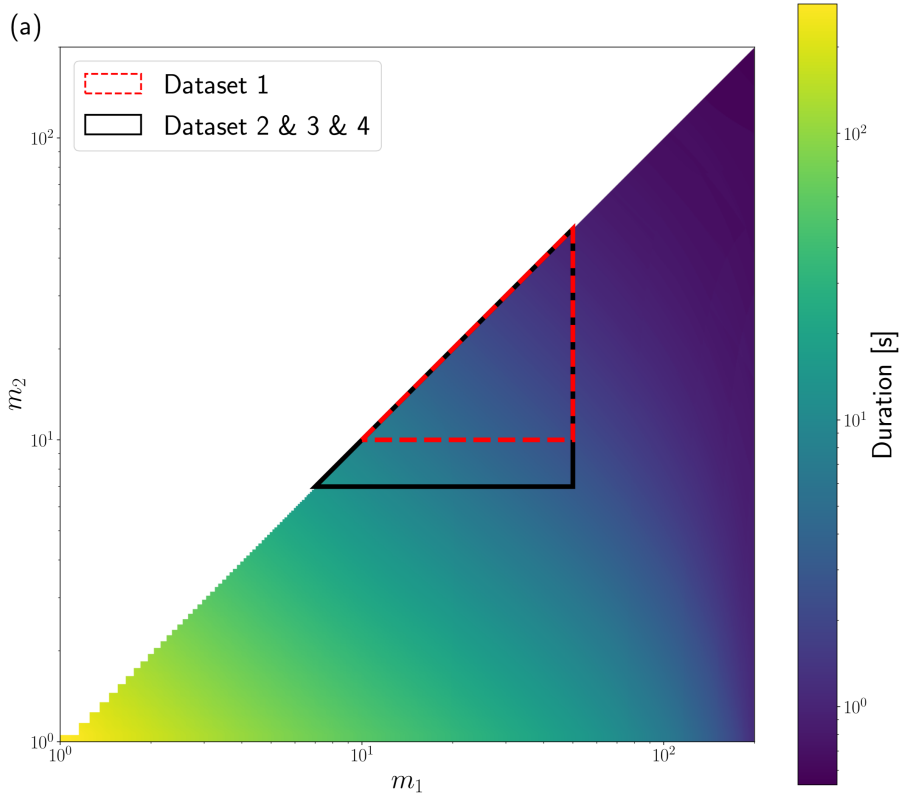
https://arxiv.org/abs/2211.01520

1 - Training set

- Dataset 4 only
- We start by generating about 740k noise segments of 1.25s duration each, over the span of 12 days
- We generate a large number (about 38k) of waveforms with parameters within the given ranges
- Our training set then consists of about 1.5M samples, half noise only, half noise + waveform
- Noise and waveforms are combined once, before training A validation set with about 1.8M noise samples is generated in a similar way, from four weeks of data The noise samples are combined with about 96k waveforms, resulting to 3.6M validation samples Our test sets are generated using the provided code (1 month)

1 - Training Set

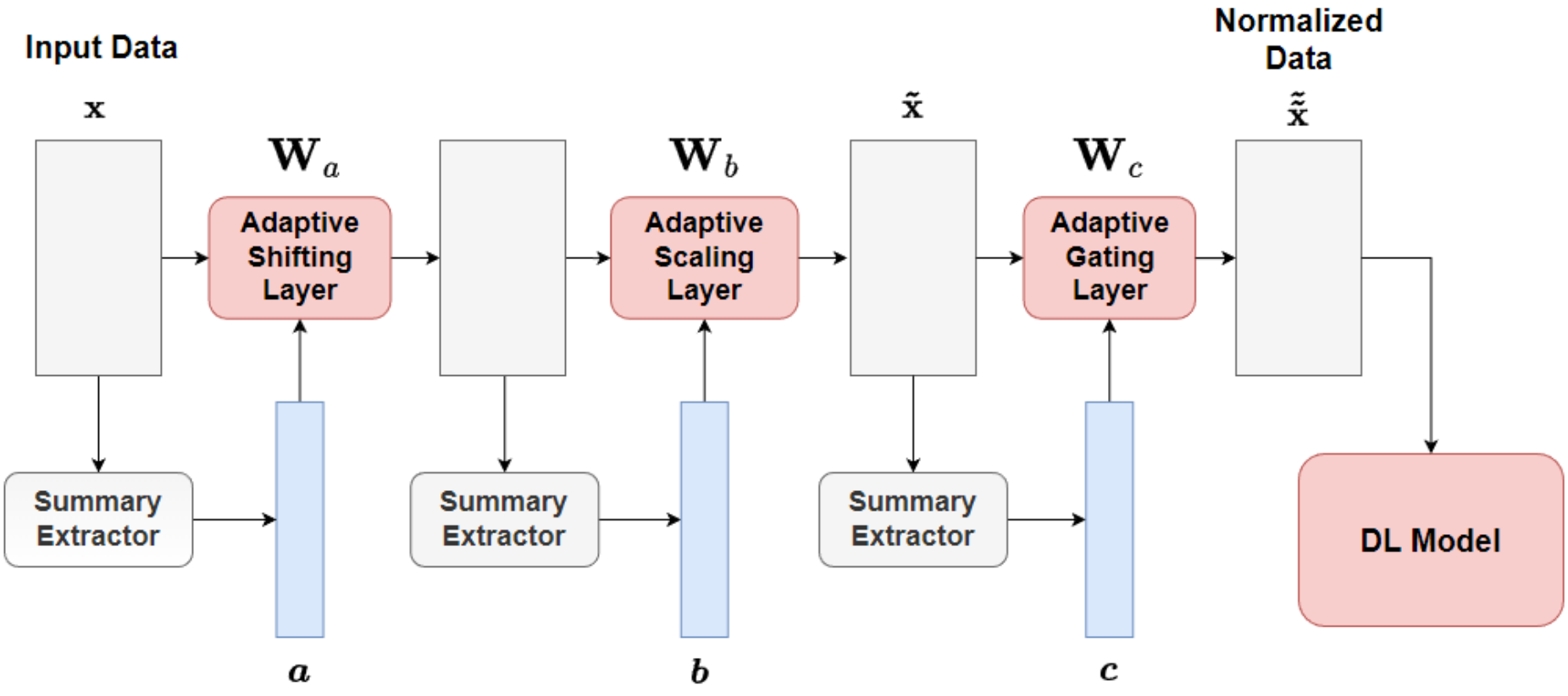
- Mass space for dataset



2 - Whitening

- We found whitening to be an important preprocessing step - in line with other relevant works
- But PyCBC's method is CPU-based, and processes each sample by itself instead of as a batch So we implement whitening using PyTorch functions
  - Welch method, following PyCBC's methods including inverse spectrum truncation
  - Operations performed on GPU directly, no need for CPU utilization or CPU to GPU data transfer
  - Some operations can be performed as batched
- After whitening the 1.25s samples they are cropped unilaterally to 1s total duration We are left with samples $\mathbf{x} \in \mathbb{R}^{2\times 2048}$

3 - Deep Adaptive Input Normalization



Passalis et al. `https://arxiv.org/abs/1902.07892`

3 - Deep Adaptive Input Normalization

Learnable data normalization, avoids squishing of samples with low levels of noise DAIN aims to learn how the measurements $\mathbf{x} \in \mathbb{R}^{2\times 2048}$ should be normalized by appropriately shifting and scaling them

$$\tilde{\mathbf{x}}_j = (\mathbf{x}_j - \alpha) \oslash \beta \tag{1}$$

where $\mathbf{x}_j \in \mathbb{R}^2$ refers to the j-th observation (out of 2048 included in the current window), $\oslash$ is the Hadamard (entrywise) division operatorwe first build a summary representation of the current window as:

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

DAIN learns how to appropriate shift the data based on the observed mode by estimating the value for the shifting operator α as:

$$\alpha = \mathbf{W}_a \mathbf{a} \in \mathbb{R}^2, \tag{3}$$

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

3 - Deep Adaptive Input Normalization

- For data scaling
  - we calculate a summary representation as:

$$b_k = \sqrt{\frac{1}{2048}\sum_{j=1}^{2048}(x_{j,k} - \alpha_k)^2}, \quad k = 1, 2, \tag{4}$$

  - then define the scaling operator as:

$$\beta = \mathbf{W}_b \mathbf{b} \in \mathbb{R}^2, \tag{5}$$

    where $\mathbf{W}_b \in \mathbb{R}^{2\times 2}$ are its parameters
  - the shifted and scaled observations are fed to an *adaptive gating layer*

$$\tilde{\tilde{\mathbf{x}}}_j = \tilde{\mathbf{x}}_j \odot \gamma, \tag{6}$$

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

$$\gamma = \text{sigm}(\mathbf{W}_c \mathbf{c} + \mathbf{d}) \in \mathbb{R}^2, \tag{7}$$

    $\mathbf{W}_c \in \mathbb{R}^{2\times 2}$ and $\mathbf{d} \in \mathbb{R}^2$ are the parameters of the gating layer, and the updated summary representation $\mathbf{c}$ is calculated as:

$$\mathbf{c} = \frac{1}{2048}\sum_{j=1}^{2048}\tilde{\mathbf{x}}_j \in \mathbb{R}^2 \tag{8}$$

4 - Deep Residual Networks

- Originally proposed for 2D image analysis (recognition, detection, etc.) `https://arxiv.org/abs/1512.03385`
- The residual connections allow for effectively training deeper networks, alleviating the gradient vanishing problem
- Let $\mathbf{x} \in \mathbb{R}^{2\times 2048}$ be the input of a residual block, then the output is given as:

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

  where f is a block of two convolutional layers followed by ReLU activation functions and BN layers, and h is the *transfer* function
- h can be either a convolutional layer or an identity mapping (i.e., $h(\mathbf{x}) = \mathbf{x}$), depending on whether or not f changes the dimensionality of its input

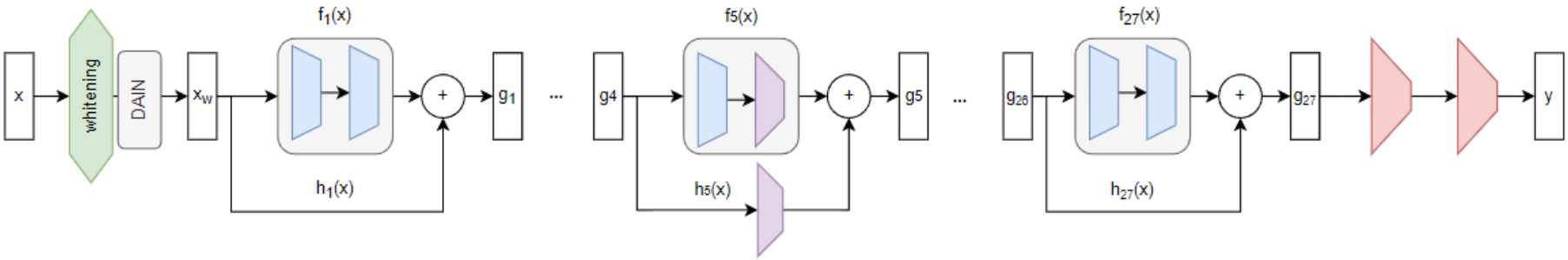
4 - Deep Residual Networks

```python
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        if out_channels != in_channels or stride > 1:
            self.x_transform = nn.Conv1d(in_channels, out_channels, kernel_size=1, stride=stride)
        else:
            self.x_transform = nn.Identity()

        self.body = nn.Sequential(
            nn.Conv1d(in_channels, out_channels, kernel_size=3, stride=1, padding='same'),
            nn.BatchNorm1d(out_channels),
            nn.ReLU(),
            nn.Conv1d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1),
            nn.BatchNorm1d(out_channels)
        )

    def forward(self, x):
        x = F.relu(self.body(x) + self.x_transform(x))
        return x
```

PyTorch implementation of residual block

4 - Deep Residual Networks

- We tried various network depths, ranging from 10 to 54, the latter being our final model
- This network takes as input the whitened, normalized 1s samples and outputs 2 values corresponding to the two possible outcomes: noise only vs. noise + waveform



4 - Deep Residual Networks

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

6 - SNR-based Curriculum Learning

- We imeplement a training strategy such that the network is first trained on the loudest injections and only at later epochs it is trained on weaker injections
- Using the signal-to-noise ratio (SNR) of the injected signal, with respect to the PSD of the noise
- An accurate computation of the SNR of each segment during training would add a lot of computational overhead we chose to construct an empirical relation that gives an approximate prediction of the SNR

6 - SNR-based Curriculum Learning

- Varying only the chirp mass, the distance and the inclination ι we arrived at the following approximate empirical relation:

$$\mathrm{SNR} = \frac{1261\ \mathrm{Mpc}}{D}\left(\frac{\mathcal{M}_\mathrm{c}}{M_\odot}\right)^{5/7}[0.7 + 0.3\cos(2\ \iota)] + 4.87, \qquad (9)$$

  where the last term was added to calibrate the empirical relation to have zero mean error with respect to the actual distribution of SNR values in the training set

6 - SNR-based Curriculum Learning

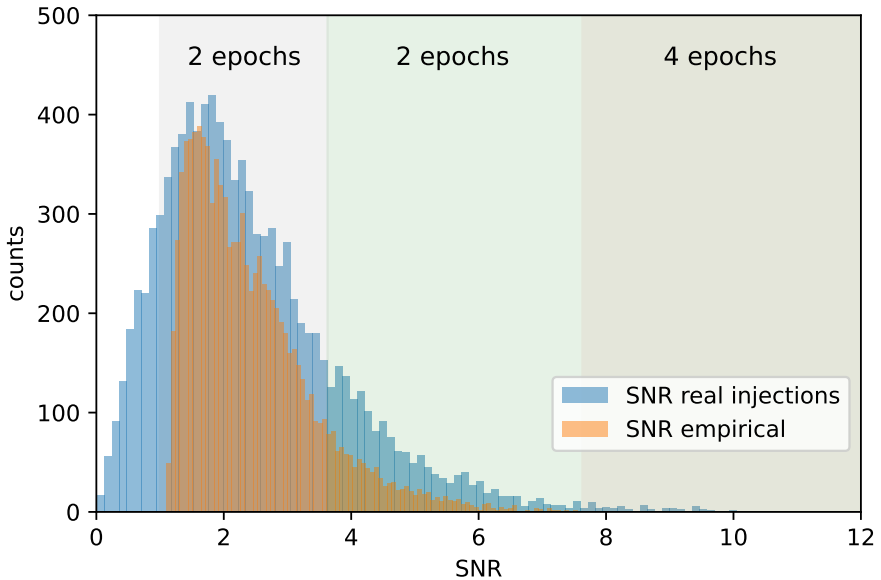


Figure: Comparison of the SNR histogram computed with the empirical relation Eq. (9) to the actual SNR of the injections (using 5000 representative injections and the noise of the Hanford detector). A similar distribution is obtained. The shaded areas correspond to the restricted values of the SNR used in the first eight epochs epochs of *learning strategy* (from right to left.

Training

- 4.25s of noise are taken around each input sample to compute the PSDs for each channel/detector
- Each sample is whitened using its computed PSD, then cropped to 1s
- If positive (i.e., noise + waveform), the sample is cropped around the reference time of coalescence, such that if falls within the 0.5s to 0.7s mark
- The entire network is optimized using a regularized cross entropy objective function
- Final validation accuracy is around 63%

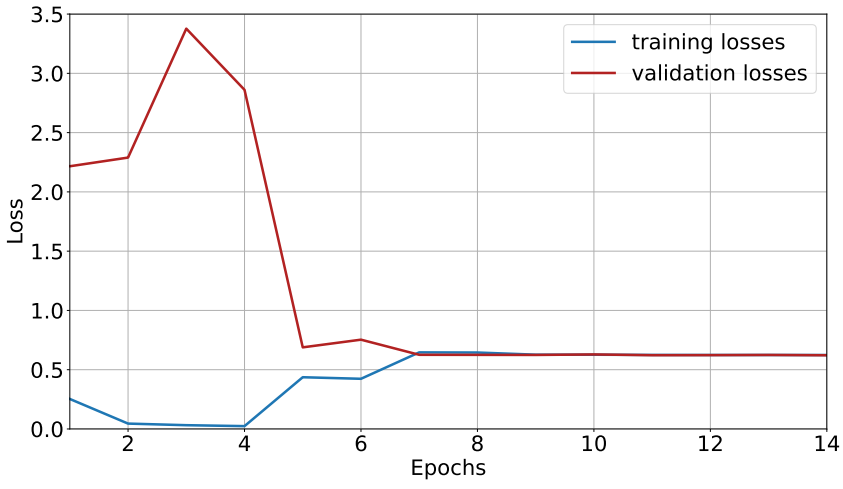
Training

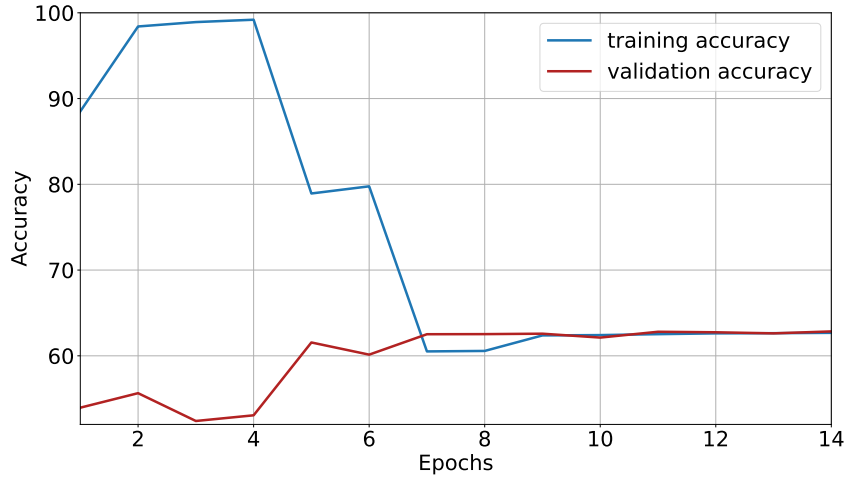
- Curriculum:

Table: Learning strategy: During the first 4 epochs the network learns only strong signals with estimated SNR larger than 12.5. 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	12.5	100
2	8.5	100
2	1	8.5
2	1	12.5
remaining	0	∞

Training



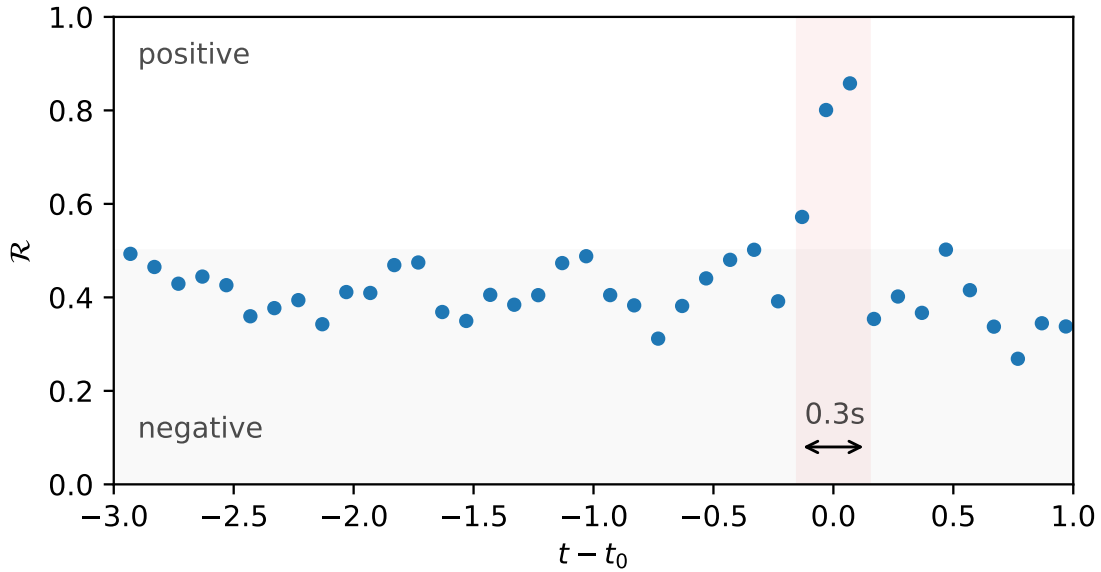


- We don't apply the SNR masking to the validation set

Evaluation

Deployment

- The test set is split in 4.25s long segments with a step size of 3.1s
- The PSDs for each segment are computed
- Each segment is split into 31 samples, using an internal step size of 0.1s, which are processed as a batch
- This results in triggers as shown below, which are then clustered in time



Deployment

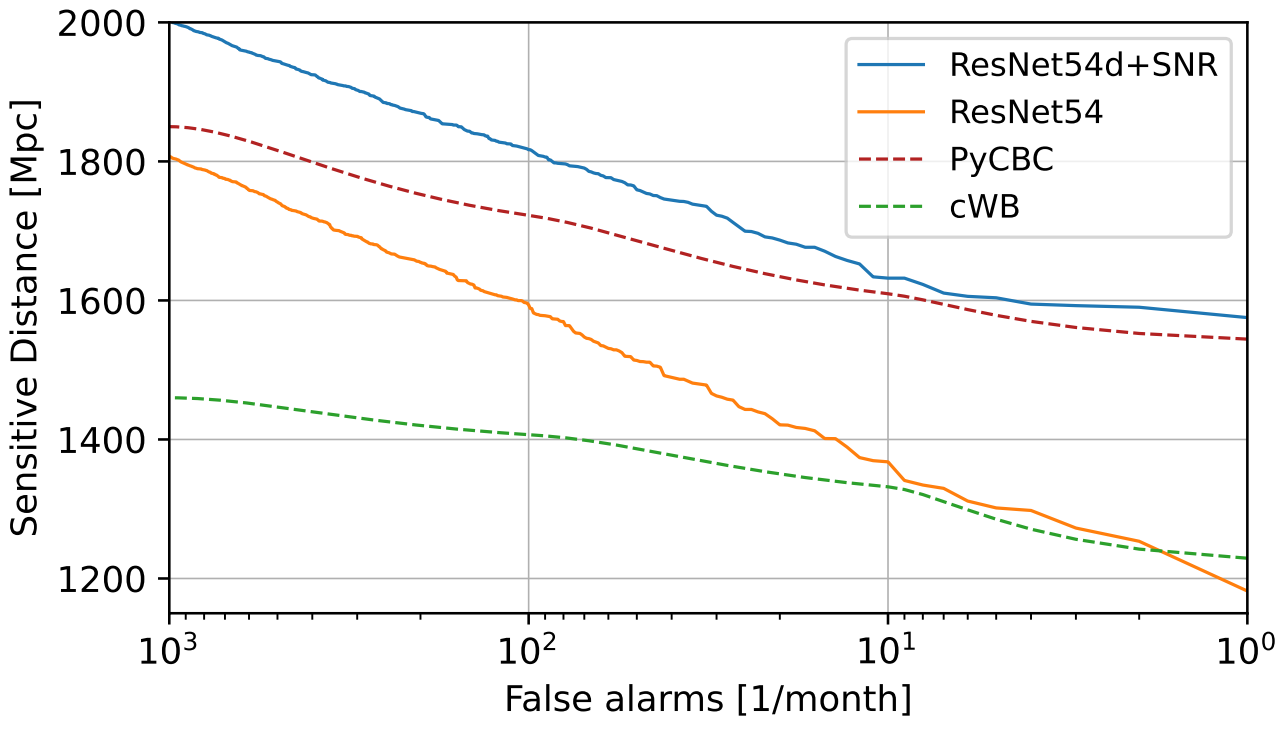


Figure: Sensitive distance vs. false alarm rate for our best model (ResNet54d+SNR), in comparison with the simpler setup (ResNet54) used in our challenge submission and two widely used algorithms for GW detection, Coherent WaveBurst (cWB) and PyCBC (the latter using aligned-spin templates only, see text for details). All codes were run on the same test dataset established in the challenge. 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).

Deployment

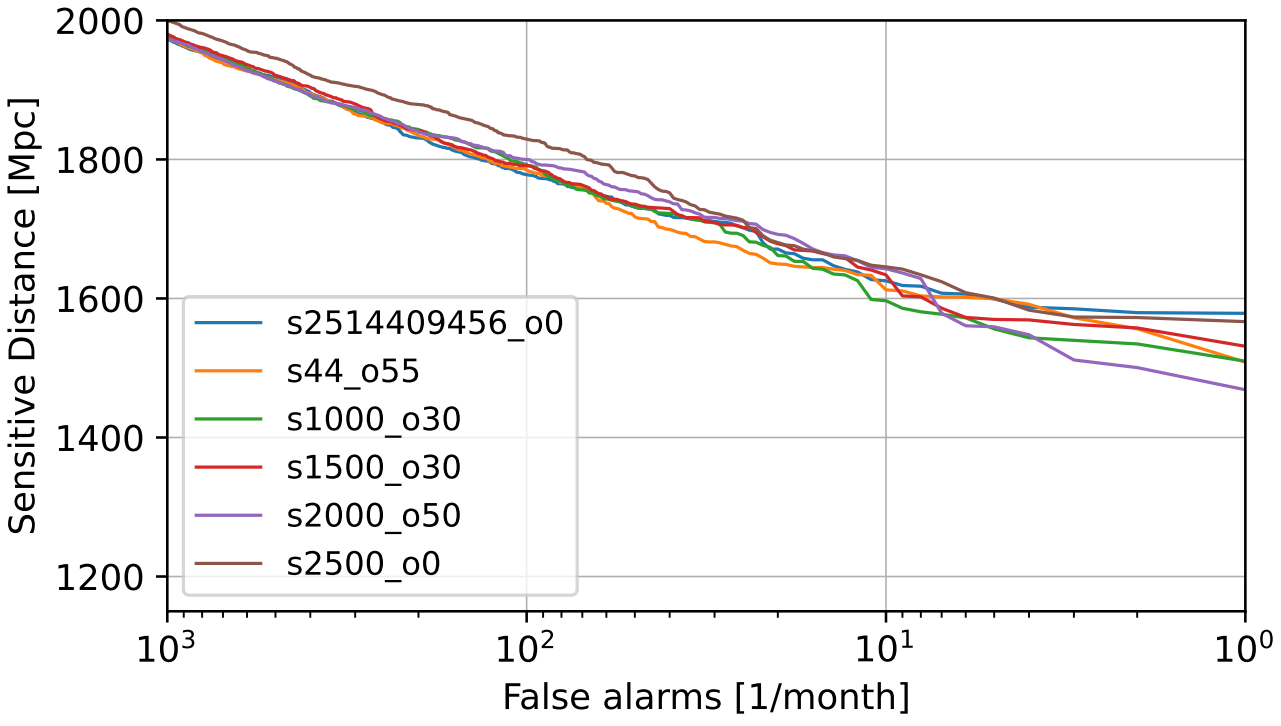


Figure: Variance of the sensitive distance vs. false alarm rate for our best model, using various test datasets of one month duration, that differ by the random seed used for the injections and the offset in the starting time. The first curve (seed 2514409456 and offset 0) corresponds to the case of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

Code

Our code is publicly available at:
github.com/vivinousi/gw-detection-deep-learning

Thank you - Questions?