Robust machine learning algorithm to search for continuous gravitational waves

Joe Bayley, Chris Messenger, and Graham Woan
SUPA, University of Glasgow, Glasgow G12 8QQ, United Kingdom

(Received 20 July 2020; accepted 21 September 2020; published 21 October 2020)

Many continuous gravitational wave searches are affected by instrumental spectral lines that could be confused with a continuous astrophysical signal. Several techniques have been developed to limit the effect of these lines by penalizing signals that appear in only a single detector. We have developed a general method, using a convolutional neural network, to reduce the impact of instrumental artifacts on searches that use the SOAP algorithm Bayley et al. [Phys. Rev. D **100**, 023006 (2019)]. The method can identify features in corresponding frequency bands of each detector and classify these bands as containing a signal, an instrumental line, or noise. We tested the method against four different datasets: Gaussian noise with time gaps, data from the final run of Initial LIGO (S6) with signals added, the reference S6 mock data challenge dataset Walsh et al. [Phys. Rev. D **94**, 124010 (2016)] and signals injected into data from the second advanced LIGO observing run (O2). Using the S6 mock data challenge dataset and at a 1% false alarm probability we showed that at 95% efficiency a fully automated SOAP search has a sensitivity corresponding to a coherent signal-to-noise ratio of 110, equivalent to a sensitivity depth of 10 $\mathrm{Hz}^{-1/2}$, making this automated search competitive with other searches requiring significantly more computing resources and human intervention.

DOI: 10.1103/PhysRevD.102.083024

I. INTRODUCTION

Gravitational-wave detectors such as the Laser Interferometer Gravitational-Wave Observatory (LIGO) [1,2] and Virgo [3,4] are sensitive to signals from many types of astrophysical sources. One type, compact binary coalescences (CBCs), has already been observed in quantity [5–7], however, other promising source types, including sources of continuous gravitational waves (CWs), remain undetected. CWs are well-modeled quasisinusoidal signals with a duration much longer than observing times of detectors. The sources of these signals are thought to be rapidly rotating neutron stars, which will emit gravitational waves (GWs) if there is some asymmetry around the rotation axis [8]. The signals will have small amplitudes compared to CBCs, and only detectable with sensitive algorithms and observing times of months or years. These search algorithms are generally classed as "targeted," "directed," or "all-sky" searches, dependent on how much is known *a priori* about the source from electromagnetic observations.

Targeted searches can be performed on sources with known sky position and spin evolution. If only the sky position is known one can perform a directed search, and if nothing is known one is forced to perform an all-sky search covering sky position and source rotational frequency (and usually frequency derivative). The most sensitive of these are targeted searches which can employ variants on coherent matched filtering [9,10]. These use template waveforms which are generated using the information already known about the source, then this is correlated with the data. Directed and all-sky searches have a much broader parameter space to search, therefore, many templates are needed to sufficiently cover the parameter space. Coherent matched filter methods have a high computing burden in broader parameter space searches and become unfeasible. This led to the development of semicoherent searches, in which the data is divided into segments that are analyzed separately and their results combined incoherently [11,12]. Semicoherent searches are generally tuned to deliver maximum sensitivity for a given computing time. An overview of current CW searches can be found in [13,14].

The analysis presented here applies to an existing semicoherent search algorithm called SOAP [15,16]. This is a fast and largely model-independent search which uses a Viterbi-like algorithm to find continuous tracks of excess power in time-frequency spectrograms. When applied to multiple detectors using a line-aware statistic, SOAP looks for frequency bins which have consistent high power in each detector. This means that, at a given frequency and a given time, SOAP will penalize signals which are not seen consistently in the detector network. The algorithmic details are summarized in Sec. II.

The sensitivity of SOAP, and many other GW searches, is limited by noise artifacts known as "instrumental lines" which have been investigated in [17] for advanced LIGO. This generic term covers a range of artifacts, including long-duration, fixed-frequency or wandering lines to fixed-frequency transients. There have been many techniques to mitigate the effect of these lines on searches including [18–20]. For the SOAP search, there are certain types of instrumental lines that are difficult to distinguish from an astrophysical signal even with the development of a "line-aware" statistic [16]. Currently these require one to manually examine the problematic subbands to determine whether they are contaminated. This process is slow and requires significant human input and judgement, and for full-band, long-duration searches it would become impractical.

We have therefore automated this process by employing convolutional neural networks (CNNs). These have been used extensively in image classification problems, and we explain their use in more detail in Sec. III. CNNs have already been shown to detect gravitational wave signals from CBCs in [21–23], have been used in searches for burst signals [24–26] and various deep learning techniques have been used in searching for CW signals in [26–29]. An overview of machine learning techniques used in GW science can be found in [30].

In Sec. II we will summarize how the SOAP algorithm works. In Sec. III we explain how CNNs operate, and we show how we generate data to train the CNN in Sec. IV. We describe the entire search, from raw data to results, in Sec. V and finally in Sec. VI we show our results from real searches, comparing them to corresponding results using other techniques.

II. SOAP

SOAP [16] is a search algorithm for unmodeled long-duration signals based on the Viterbi algorithm [31]. In its most simple form SOAP analyzes a spectrogram to find the continuous time-frequency track which gives the highest sum of fast Fourier transform power. If a signal is present this is the track which is most likely to correspond to that signal. In [16] the algorithm was expanded to include multiple detectors as well as a statistic to penalize artifacts in the data from instrumental lines.

Figure 1 shows an example of a time-frequency spectrogram and the corresponding outputs from SOAP; the three main output components are the frequency track, the Viterbi statistic, and the Viterbi map, described below:

Viterbi track. The Viterbi track is the most probable track through time-frequency data for given a choice of statistic [i.e., summed short Fourier transform (SFT) power].

Viterbi statistic. The Viterbi statistic is the sum of the individual statistics along the Viterbi track. In the analysis that follows we use the line-aware Viterbi statistic. This is the sum of the log-odds ratios, $p_{\mathrm{signal}}/(p_{\mathrm{line}} + p_{\mathrm{noise}})$ along the track [16].

Viterbi map. The Viterbi map shows the value of the Viterbi statistic for every time-frequency bin in the spectrogram, corresponding to the log-probability that the track passes through each time-frequency bin. Each time slice in the map is normalized individually, i.e., each vertical slice is adjusted so that the sum of their exponentiated values is unity. Each pixel in the image can therefore be interpreted as a value related to the log-probability that the signal is in that frequency bin at that time.

In [16] we used the Viterbi line-aware statistic (described above) to determine whether the signal had an astrophysical origin. This statistic reduces the effect of instrumental lines on the analysis, but certain types of line are not picked up by it. For example, the statistic is affected by broad, wandering, common lines as they offer high power tracks in both detectors. To reduce the effect of these instrumental lines in [16], we examined the spectrograms and Viterbi maps of individual bands by eye, as in Fig. 1. Bands which appeared to be contaminated were then manually removed from the search.

In this paper we show that we can exploit additional information in the spectrograms and Viterbi map, in combination with the Viterbi statistic, to perform the process of removing contaminated bands automatically. The tool which we use to classify this extra information is a convolutional neural network.

III. CONVOLUTIONAL NEURAL NETWORKS

CNNs are a type of deep neural network which are primarily used in image processing and recognition [32–35]. A CNN is designed to take in data, identify different features within that data and classify what those features or combinations of those features mean. In the context of this work the input data is a time-frequency spectrogram which may contain a (simulated) CW signal. The output is then a single number which represents the confidence that a signal is present. Values closer to 1 represent the presence of a signal and closer to 0 represent its absence. A CNN can learn how to identify features by training on many labeled examples of the input data where the output is known. For example, an input spectrogram with a CW signal would have a label of 1. Given the set of training examples, the many parameters of the CNN can be updated such that it gives the best result for any new spectrogram image. The many parameters of the CNN relate to the key building block of neural networks: the neuron.

A. Neurons

A neuron converts any number of inputs into a single output value and can perform three operations which are applied to N inputs $\boldsymbol{x}$: multiplying each input by a "weight" w, adding a "bias" b and passing them through, and

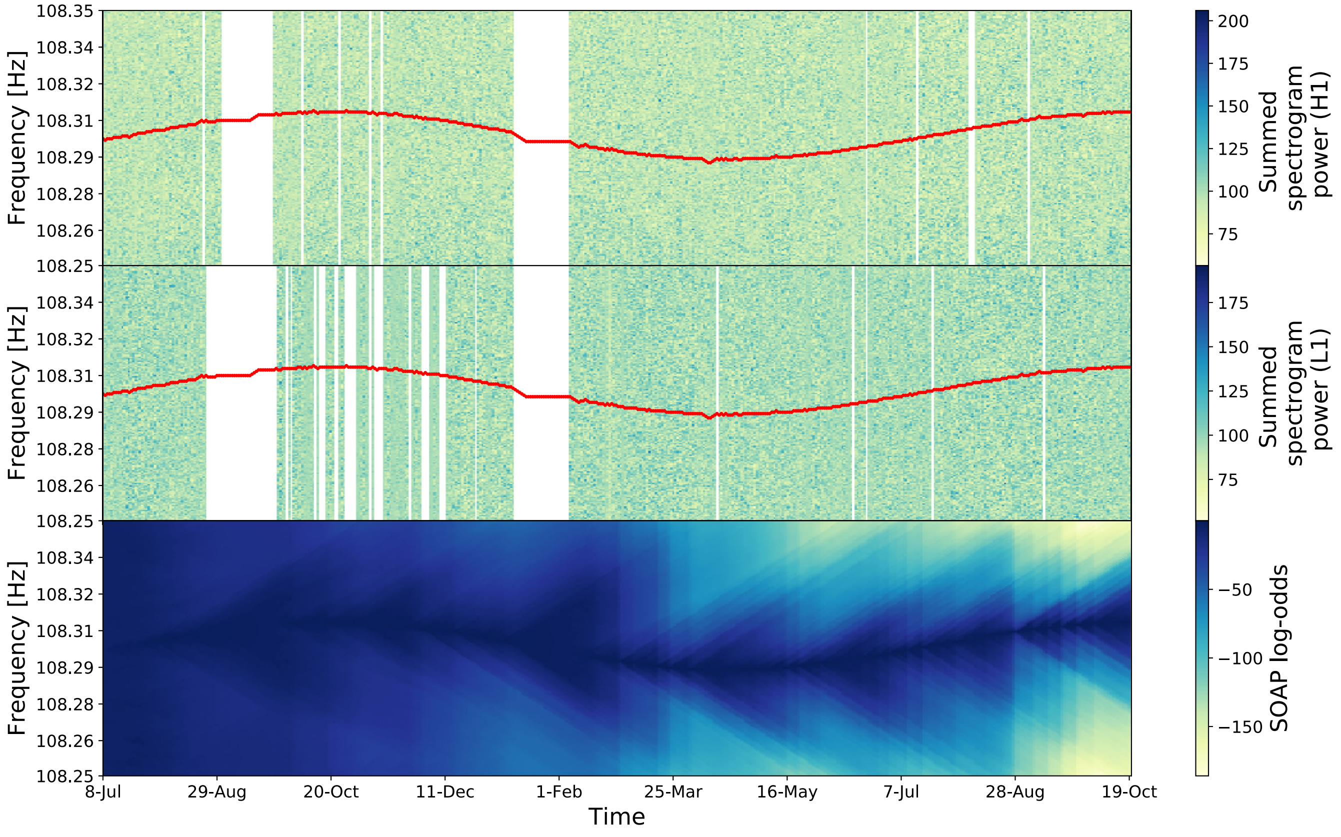

FIG. 1. An example of a SOAP search. The top two panels show time-frequency spectrograms, preprocessed according to Sec. V and representing a 0.1 Hz-wide frequency band from the LIGOs S6 observing run. The data includes a simulated CW signal. The white areas in the spectrograms are gaps in data when the corresponding detector was not operating. The optimal track found by SOAP is overlaid in both cases. The bottom panel shows the normalized Viterbi map with the pixel intensity showing the log-probability that the track falls in a particular frequency bin as a function of time.

applying a nonlinear "activation function" f. Figure 2 shows this basic operation, where there is one weight for every input and a single bias for each neuron. The output O is therefore related to the inputs by

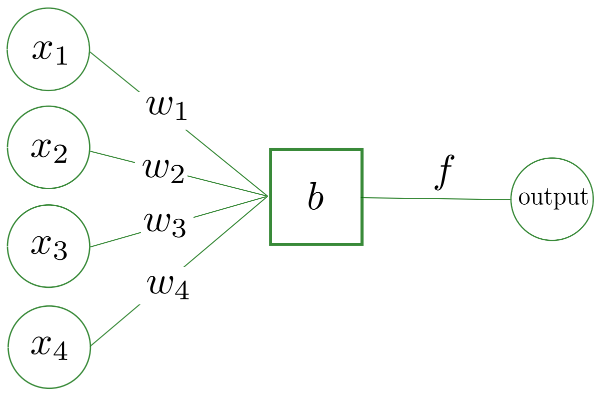

FIG. 2. Equation (1) can be visualized as above. Here the inputs x_i are multiplied by the corresponding weights w_i, the sum of these and the bias b are then passed through an activation function f to the output. This example has four inputs but there can be any number.

$$O = f\left(b + \sum_{i=1}^{N} w_i x_i\right). \tag{1}$$

The weights and bias are the parameters which the neural network "learns" during training and we consider this further in Sec. III D. The activation function is there to explicitly impose a nonlinearity to the calculation.

B. Neural networks

To create a neural network, many of these neurons are connected together into "layers." Layers comprise a set of neurons, each of which takes the same data as input but which applies a different set of weights and biases. The output of each neuron then acts as the input to another set of neurons or another layer. Neural networks combine many of these layers together to learn abstract representations of the data. For classification, this abstract representation is distilled down to a simple output. Neural networks can be made from many different types of layers. We described a "fully connected" layer above, in which each neuron in the layer takes in all the data points or the previous layer's outputs. However for certain types of problems, such as identifying features in images, another type of layer called a convolutional layer is better suited.

C. Convolutional layers

Convolutional layers are an adaptation of the fully connected layers described above in Sec. III B, where the input data is generally image pixels. For this type of layer there is not a separate weight for each input data point (pixel). Rather, there is a fixed number of weights defined by a "filter" size. This filter is convolved with the input image such that the output of the layer is a filtered image. This operation is shown in Fig. 3. The convolutional layers equivalent to Eq. (1) are

$$O_{ij} = f\left(b + \sum_m \sum_n F_{mn} x_{i-m,j-n}\right), \qquad (2)$$

where O is the output image, b is the bias, x is the input image, F is the convolutional filter and f is the activation function. The indices m and n iterate over the filter rows and columns and the indices i and j iterate over the input image rows and columns. The convolutional layer can learn to identify features within an image by changing the weights and bias of a filter. A convolutional layer can apply a number of these filters as defined by the user. If the layer has 10 filters then the output is 10 filtered images. Each of these filters can then be trained to identify different features within the input image [33,35].

The output of a convolutional layer comprises a number of filtered images, so potentially there is a lot of data to feed to the next layer. A method called max pooling can be used to reduce the size of the output while retaining the important information within the images. A max-pooling layer splits the image into blocks of fixed size and takes the maximum pixel value in each block as the output. So if the size of the max-pooling block is 2×2, the output image will have $1/4$ the number of pixels of the input.

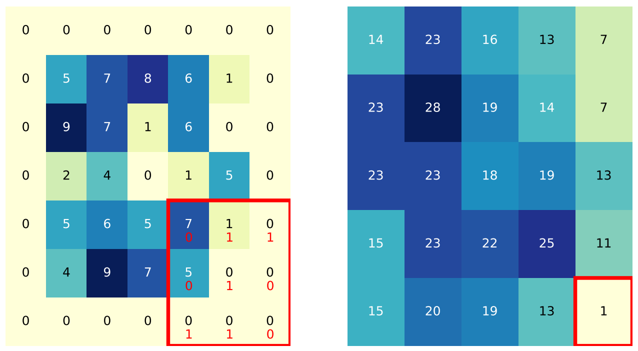

FIG. 3. The convolutional layers convolve a filter with the input image and output a convolved image the same size as the input to pass to the next layer. Here we show a simple 5×5 image with a 3×3 filter, the input is padded with zeros such that the output is the same size. When the network is trained, the values within the filter (the red values below the inputs) are updated.

D. Training

Once the structure of the network is decided, the network needs to be trained by adjusting the weights and biases to give the desired performance. To achieve this the networks classify the input images (the spectrograms and Viterbi maps) using a single output neuron with a sigmoid activation function which restricts the output between 0 and 1. The CNN is trained using a supervised learning process in which the class of each input example is known. We assign a label of 1 when the input is a time-frequency spectrogram which includes a simulated CW signal and 0 when there is no signal.

The performance of the network can be improved by increasing the number of input examples which is seen during training. This helps it learn the underlying features within the data and prevents it from overfitting to specific examples.

Each of the training examples is then propagated though the network to its corresponding single output value, which lies between 0 and 1. This output is then compared to the label of the input data using a loss function. For our two-class network the loss function, L, is the binary cross entropy [36]

$$L = -y \log(p) + (1 - y) \log(1 - p), \qquad (3)$$

where p is the network's output, which has any value in the range [0, 1] and y is the true output which has the binary label 0 or 1. The loss function is minimized when the output matches the truth. Its current value is used to train the network by updating the weights and biases through the process of "back propagation," typically using the derivative of the loss function with respect to a weight to update that weight [37].

E. Network structure

In this section we describe the structure of the networks used in our analysis. There are three possible inputs for each CNN: a spectrogram, a Viterbi map and the Viterbi statistic. Each of these are different representations of the raw detector data. We proceed by training a separate CNN for each input separately and then a further three CNNs which use combinations of inputs: Viterbi map + spectrogram, Viterbi map + Viterbi statistic and Viterbi map + Viterbi statistic + spectrogram. With the exception of the output layer, all the CNN layers use the "leakyRELU" activation function [38] in Eqs. (2) and (1). We use a sigmoid activation function for the output neuron so that, for a given input, a CNN generates an output a value between 0 and 1. The closer this output value is to 1 the greater the probability that the input contains a signal, so this output value can then be treated as a detection

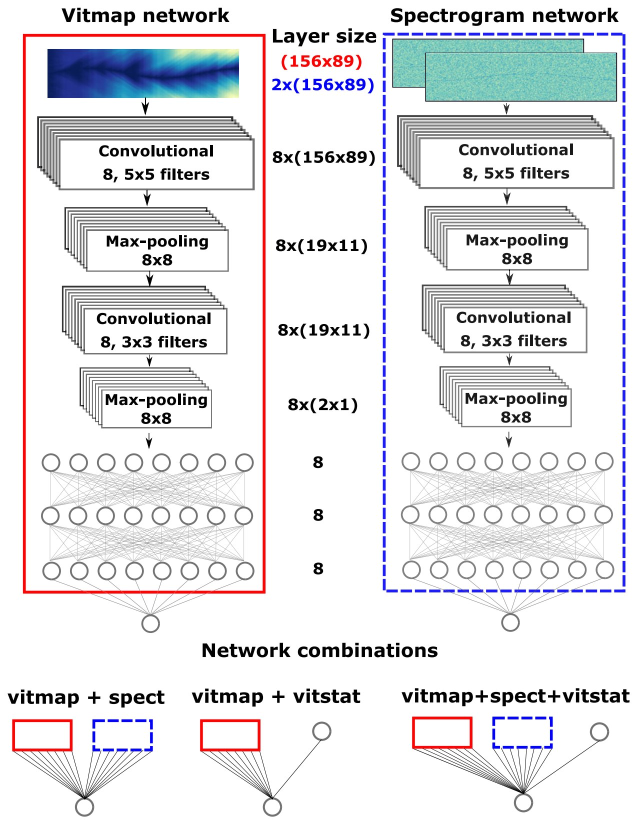

FIG. 4. The structure of the Viterbi map (vitmap) and spectrograms CNNs and the arrangement of the combined CNNs. The Viterbi map and spectrogram CNNs are identical other than the input to the spectrogram CNN is two images. They each use two convolutional layers and 3 fully connected layers before they are output to a single neuron which represents the probability of belonging to the signal class. The size of the layers as the image progresses through the network is shown, where the image size is in parentheses. The Viterbi statistic network is a single neuron that transforms the statistic into a number between 0 and 1 representing the probability of belonging to the signal class. When multiple networks are combined, the final output neuron and the weights connecting to the previous layer are removed, i.e., in the vitmap network the components inside the red box are used. In the vitmap + spect case, each network then has 8 output neurons which are combined to a single neuron using 16 new weights.

statistic. The structure of the networks for the Viterbi map (vitmap), spectrograms and their combinations are shown in Fig. 4 and the components are described below:

Viterbi statistic. This is the simplest of the networks and comprises a single neuron which takes in the Viterbi statistic, applies a weight and bias and passes the result through a sigmoid function. This would give the same sensitivity as the Viterbi statistic on its own, however can now easily be combined with other networks.

Viterbi map. The Viterbi map CNN takes in a down-sampled Viterbi map of size (156,89) as input, described in Sec. IV C. As shown in Fig. 4, this CNN consists of two convolutional layers and three fully connected layers. The first layer has 8 filters which have a size of 5×5 pixels, the second layer has 8 filters with a size of 3×3 pixels. After each of these layers we use a max-pooling layer with a size of 8×8 pixels. This is then passed into three fully connected layers which all have 8 neurons and used leakyRELU activation functions. Finally these lead to an output neuron which uses a sigmoid activation function.

Spectrogram. The spectrogram CNN takes down-sampled spectrograms of size (156,89) as inputs (see Sec. IV C). It has an identical structure to the Viterbi map CNN but takes the spectrograms of two different detectors as inputs.

The next three networks are constructed from combinations of these single CNNs:

Viterbi map and spectrogram. To combine the spectrogram and Viterbi map network we remove the final output neuron and its 8 weights from each of the networks and combine these to a single sigmoid neuron with 16 new weights.

Viterbi map and Viterbi statistic. In this network we combine the Viterbi statistic with the Viterbi map. As before, this uses the pretrained Viterbi map and Viterbi statistic CNNs. Again, the output sigmoid neuron and corresponding weights are removed from each network. The 8 neurons from the Viterbi map network and the single neuron from the Viterbi statistic network are then combined to a single neuron with 9 new weights.

Viterbi map, Viterbi statistic and spectrogram. This combination takes all component CNNs from above. As before the final sigmoid output and the corresponding weights from each network are removed. The 8 neurons from the Viterbi map and spectrograms CNNs and the single neuron from the Viterbi statistic are then joined into a single output neuron with 17 new weights.

To combine CNNs we use "transfer learning" [39] by taking the pretrained weights of the networks as a starting point for further training. In our examples we found that we could fix the weights inside the pretrained networks and just train the final 16 output weights from the neurons as in Fig. 4. We chose to investigate combinations of networks because different representations of the data should contain slightly different information on the presence of a signal. For example, the Viterbi statistic contains no information on the structure of the track in the time-frequency plane, and the Viterbi maps lose some information about multiple lines in the band. The spectrograms contain the most information but in an unprocessed form. When used in combination, the resulting CNN should be able to pick the important features from each of these representations.

IV. DATA GENERATION

To train the CNNs we need to generate many examples of labeled data corresponding to the three data inputs used

TABLE I. The upper and lower limits bounding the random signal parameter. The parameters α, $\sin(\delta)$, f, $\log(\dot{f})$, $\cos(\iota)$, ϕ_0, ψ were sampled uniformly between these ranges. The frequencies f_l and f_u refer to the lower and upper frequency of the band into which each signal is injected. Excluding the distribution of frequencies f, all the injections parameters are sampled from the same distributions as the S6 MDC [41].

	α [rad]	$\sin(\delta)$ [rad]	f [Hz]	$\log_{10}(\dot{f}[\mathrm{Hz/s}])$	$\cos\iota$ [rad]	ϕ [rad]	ψ [rad]
Lower bound	0	-1	$f_l + 0.25$	-9	-1	0	0
Upper bound	2π	1	$f_u - 0.25$	-16	1	2π	$\pi/2$

above, i.e., time-frequency spectrograms, Viterbi maps and Viterbi statistics. The training data needs to include many examples of possible features which could appear, such as Gaussian noise, non-Gaussian artifacts and CW signals. Non-Gaussian artifacts are difficult to simulate, but it is possible to use artifacts in real data as part of the training set. Therefore, for the majority of the analysis that follows, the time-frequency spectrograms used to generate the Viterbi data come from real LIGO observing runs (see Sec. VI).

Overall, we need to consider three sets of data, labeled "training data," "test data" and "search data." Training data contains a set of augmented (see Sec. IV B) time-frequency spectrograms containing simulated signals and is used to train each of the networks. Test data is a separate set of simulations which are not augmented. These are used to generate efficiency curves and test the network. Search data does not contain any simulated signal injections and is used to search for real signals within the data.

When training and testing a network it is important that the networks are not trained and tested on the same data. Otherwise the CNNs can learn specific features of the training data and not the underlying distribution of features. To avoid this, the spectrograms are split into subbands of width 0.1 Hz. Alternating bands are designated as "odd" or "even," so that bands starting at 100.1, 100.3 Hz are odd and those starting at 100.2, 100.4 Hz are even etc. The networks can then be trained on the odd bands and tested on the even bands and vice versa. When we search over data we will therefore have two trained networks, one for the even bands and one for the odd bands.

A. Signal simulations

To inject simulated signals into real data we first generate a set of signals with parameters drawn randomly from prior distributions defined in Table I. The signal-to-noise-ratio (SNR) of the simulations is uniformly distributed between 50 and 150, where the SNR is the coherently "recovered" SNR defined in Eq. (5). This is calculated for each time segment using the definition of optimal SNR in [40], the total SNR is then the sum of the squares of these. The GW amplitude h_0 is scaled based on the noise power spectral density (PSD) to achieve this SNR. The power spectrum of the signal can then be simulated in each time segment of a time-frequency spectrogram. This is done by assuming that the spectrogram is χ^2 distributed. The antenna pattern functions are taken into account for the given source parameters and detector such that the SNR for each time segment is calculated. This SNR is spread over neighboring frequency bins dependent on its location in frequency. The power spectrum values can then be drawn from a noncentral χ^2 distribution with the noncentrality parameter equal to the square of the SNR. Each signal is simulated in two detectors: LIGOs H1 and L1. The SNRs reported below are then the sum of the squares of the SNRs from each detector. The simulation code used in this analysis can be found in [15].

B. Augmentation

To train a neural network, many examples of data from each class are needed to avoid overfitting. Simply using data between 40 and 500 Hz and splitting the data into 0.1 Hz-wide subbands does not give enough data for the networks to be trained effectively. We therefore use the technique of data augmentation [42,43] to artificially increase the number of training examples. Augmentation is the process of transforming existing data so that, to the network, it appears to be "new" data. For example, by reversing a time-frequency band in time we get a new realization of noise in that frequency band. This gives two noise realizations for each frequency band and would double the size of the training dataset, reducing the likelihood of overfitting to the training data.

We applied augmentations to the spectrograms from each of the detectors. The augmentations that are used on each subband are: reversing the data in time, flipping the data in frequency, rolling the data in time by a small number of segments and shifting the data in frequency by a small number of bins. As we use real data, there are gaps in time where the detectors were not operating. We preserve the location of these gaps when augmenting the data. When shifting the data in frequency we shift each band up and down by 30 frequency bins (0.016 Hz) and up and down by 60 frequency bins (0.032 Hz). When rolling the data in time, we roll each subband by 100 time segments (100 days). Figure 5 shows examples of the original data, a flip in frequency, a roll in time and a flip in time. For each frequency shift, we flip the subband in time and frequency and roll the subband in time. This then gives us three transformations for each of the four frequency shifts, which

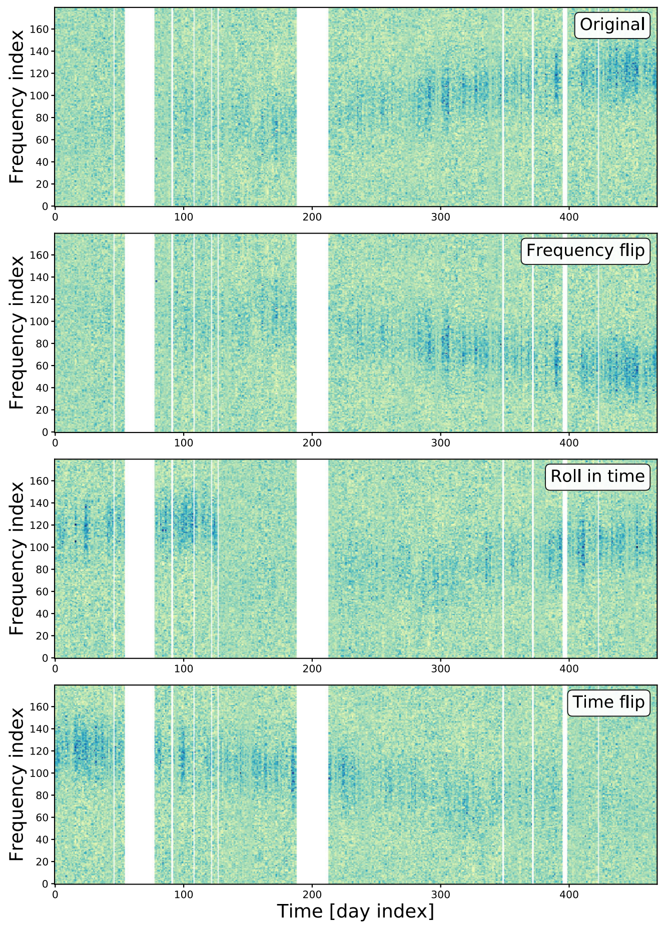

FIG. 5. The data is transformed by flipping the data in frequency (panel 2), rolling the data in time by 100 bins (panel 3) and flipping the data in time (panel 4). The original summed spectrogram is show in panel 1. Simulated signals can then be injected using this data as noise. The plots above show a broad wandering line to demonstrate the changes to the data when it is augmented, however, the majority of subbands contain almost Gaussian noise.

including the original data gives 15 augmentations of each band and therefore 15 times the number of training examples.

C. Downsampling

The raw spectrograms contain a large number of pixels and, as the spectrograms pass through the network, there are a correspondingly large number of computations to perform and a significant burden on memory. To reduce their size, the spectrograms are binned in time over one day, i.e., every 48 time segments, as in [16]. As well as reducing the size of the spectrogram, this increases the SNR within a given time-frequency bin assuming that the signal remains within the frequency bin for the majority of the time segment. To reduce the size of the data further we used the "resize" package from scikit-image [44] to interpolate and resize the summed spectrograms to 156 time segments by 89 frequency bins. This size was defined based on the summed spectrograms of the S6 dataset. This is $1/3$ the number of summed segments in time and $1/2$ the number in frequency. This downsampling is applied to the spectrograms and Viterbi maps. In [16] we demonstrated that summing spectrograms can increase the speed and sensitivity of our search. When downsampling the image, we found that reducing the amount of data had a small affect on the sensitivity of the CNNs used.

V. SEARCH PIPELINE

The components described above were combined to form a single search pipeline with a flow diagram shown in Fig. 6. We ran this pipeline in three modes, to train the CNN, test the search and run a search on real data. The elements of the flow diagram are described below:

(1) *SFTs.* These are 1 800 s SFTs generated from the detector strain time-series data. This is the standard SFT length for a number of CW searches.

(2) *Normalizing.* The SFTs are then divided by their running median with a window width of 100 frequency bins. If we assume the resulting SFTs to be χ^2 distributed, we can apply a correction factor using LALSuite code XLALSFTtoRngmed [45] such that their power spectrum has a mean of $\sim$1. By then multiplying this by 2, the noiselike component of the spectrum is distributed as expected.

(3) *Narrow banding.* The computational efficiency can be improved if the data are divided into frequency bands so the analysis can be completed on each band using separate CPU nodes. In this search the spectrograms are split into 2.1 Hz-wide bands every 2 Hz, i.e., 100.0 to 102.1 Hz, 102.0 to 104.1 Hz etc. The analysis on each node will further split the data into 0.1 Hz-wide subbands. The overlap then allow the subband from 1.95–2.05 to be calculated on one node. The band size was chosen based on the available computational memory at the time.

(4) *Band splitting.* A CNN should not be trained on the same data that it will be tested on, so each of the 0.1 Hz-wide subbands are split into odd or even bands. A CNN can then be trained on even bands and tested on odd bands, and vice versa.

(5a) *Training data generation.* The training data generation is described in Sec. IV. Each of the 0.1 Hz subbands is "augmented" (Sec. IV B). For each of the augmented bands, the data is duplicated and signals are injected into the copy with SNRs in the range 50–150 to give an example of a noise class member and a signal class member. There are two of these sets, one for even bands and one for odd.

(5b) *Testing data generation.* The signals in the testing data following the parameters in Table I are injected into 50% of the 0.1 Hz subbands. These signals have a SNR in the range 20–200. The SNR range here is wider than the training set to show how the trained

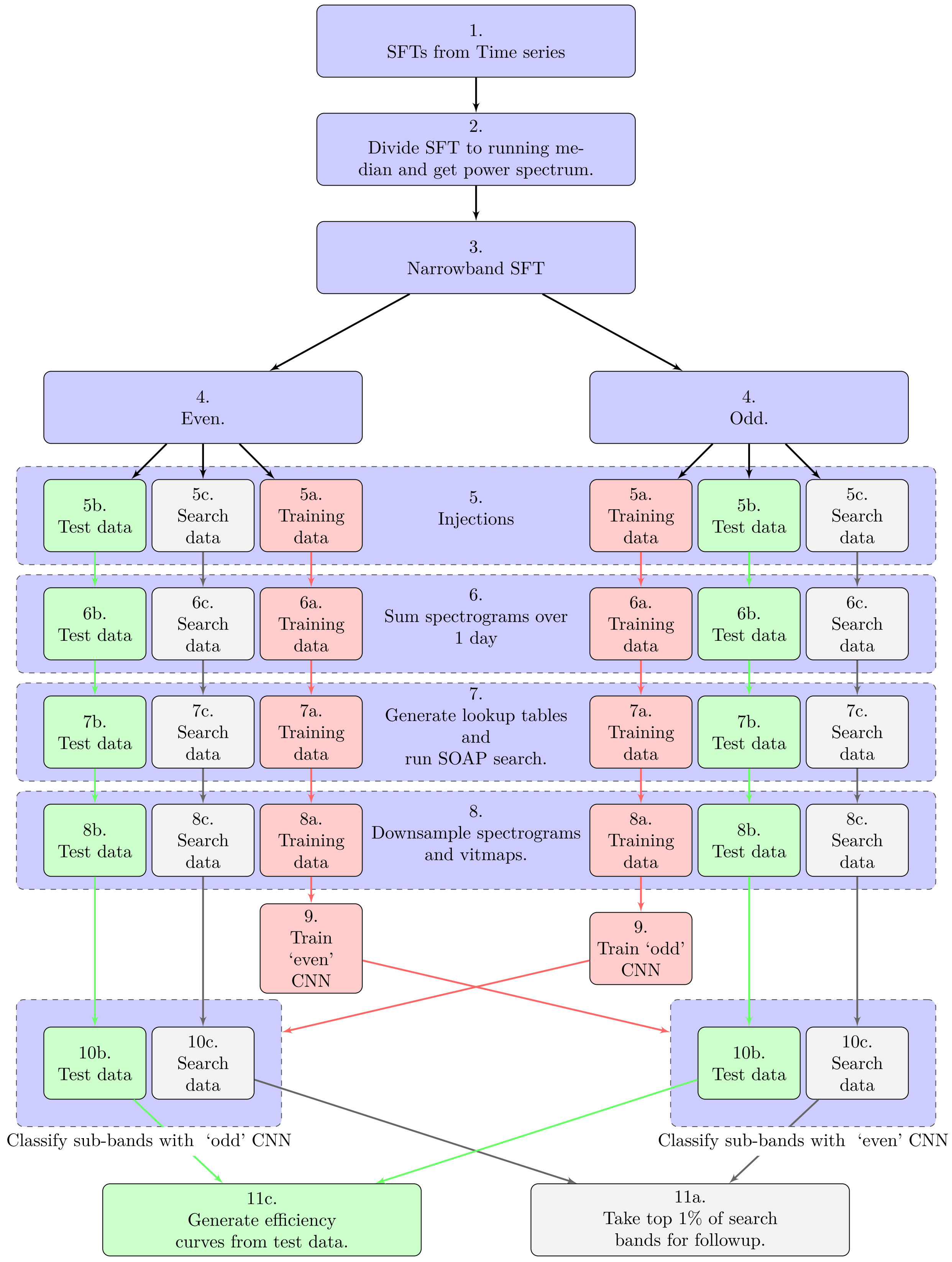

FIG. 6. The SOAP pipeline from start to finish. There are three main sections: Training (red), testing (green) and searching (grey) for both the odd and even bands. The blue sections surrounding these indicate that the same operation is applied to each of the training, test and search data. The blue sections mean that the same operations are applied to all data in that section, for example, injections are made into training, test and search data for both odd and even bands in step 5.

networks perform on this wider range. Again we have a set for odd and a set for even subbands.

(5c) *Search data.* This data is generated to assess the performance of the trained network with real signals. The subbands described in part 4 are now overlapping by 0.05 Hz. This means that if there is an astrophysical signal it should be fully contained within at least one subband. We do assume that the signal frequency does not change by more than 0.1 Hz over a year, which is reasonable for isolated neutrons stars <500 Hz. There are both odd and even versions of this search data.

(6) *Summing spectrogram.* Following the practice in [16] the spectrograms are summed over one day, i.e., we sum 48 contiguous 30-minute time segments of the spectrogram to give one time segment per day. This is done separately for each of the six datasets (three for odd, three for even).

(7) *Generate lookup tables and run SOAP search.* When the SOAP search is run using the line-aware statistic, lookup tables which contain values of the statistic as a function of the spectrogram power in each detector [16] are used to increase the speed of the analysis. These lookup tables are generated in advance of the search. Once done, we run separate SOAP search for each of the six datasets (three odd, three even) separately.

(8) *Downsample data.* At this stage there are four saved elements for each of the six datasets: two spectrograms, the Viterbi map and the Viterbi statistic. The spectrograms and the Viterbi map are downsampled to a size of (156×89) using interpolation from scikit-image's resize [44]. This size was chosen based on the S6 MDC dataset, where this is $1/3$ the length in time and $1/2$ the width in frequency of the summed spectrograms. This was chosen such that the CNNs trained efficiently and still achieved a reasonable sensitivity.

(9) *Train networks.* The downsampled training data is then used to train the CNNs. One CNN is trained on odd bands and another CNN with the same structure is trained on even bands.

(10b) *Run search on testing data.* The trained CNNs from part 9 are then used to classify each subband in the testing data with injections. This returns a statistic in the range [0, 1], where values closer to one imply that an astrophysical signal is likely to be present in the data. Here the CNN trained on the 'odd' bands is tested using the 'even' bands and vice versa. The algorithms are run on this test data to asses the sensitivity of the analysis.

(10c) *Run search on real data.* The trained CNNs from part 9 are then used to classify each subband in the search data, returning a statistic in [0, 1] as in part 10b. Once again the CNN trained on the odd bands is run on the even bands and vice versa.

(11a) *Signal candidates.* The subbands which return a statistic in the top 1% of all subbands can be taken as potential candidates. This can then be followed up with other CW search methods.

(11c) *Efficiency curves.* The output statistics from the test dataset (11b) are examined to assess how well the network has classified signals as a function of the injected signal SNR. A range of efficiency curves are generated, as detailed in Sec. VI A.

VI. RESULTS

The networks described in Sec. III E were trained and tested on four different datasets: the S6 MDC as in [16,41], our own injections into O2 data, Gaussian noise with the same time gaps and noise floor as the S6 dataset, and our own injections into real S6 data. Each of the searches uses training and testing data in the frequency range of 100–400 Hz, except the S6 MDC which uses data in the range 40–500 Hz for consistency with other searches in the challenge. All of the networks were trained using the Adam optimizer [37] with a learning rate of 0.001. For each training epoch the training data was split into random batches of size 1000, where the network weights are updated after each batch. The networks were trained for 400, 200 and 4000 epochs for the vitstat, vitmap and spectrogram networks respectively.

A. Sensitivity

To investigate the sensitivity of the pipeline we use two measures: the sensitivity depth $\mathcal{D}$ [40] and the optimal SNR ρ [46], both described in [16]. The sensitivity depth is defined as

$$\mathcal{D}(f) = \frac{\sqrt{S_h(f)}}{h_0}, \tag{4}$$

where $S_h(f)$ is the single-sided noise PSD and h_0 is the GW amplitude. The optimal SNR is defined as

$$\rho^2 = \sum_X 4\Re \int_0^\infty \frac{\tilde{h}^X(f)\tilde{h}^{X*}(f)}{S^X(f)} df, \tag{5}$$

where X indexes over detectors and $\tilde{h}(f)$ is the Fourier transform of the time series of the signal $h(t)$. ρ^2 is defined in [40] for a double-sided PSD but here we have defined it for the more common single-sided case.

The sensitivity curves shown in Figs. 8–10 were generated using a 1% false alarm probability, which we use to set our detection threshold. This threshold is the statistic value exceeded by just 1% of subbands that do not contain an injection. The efficiency is defined as the fraction of events which exceed the false alarm threshold for any given SNR. The SNR is sampled uniformly between the range 20–200 as described in Sec. V. Instead of having multiple

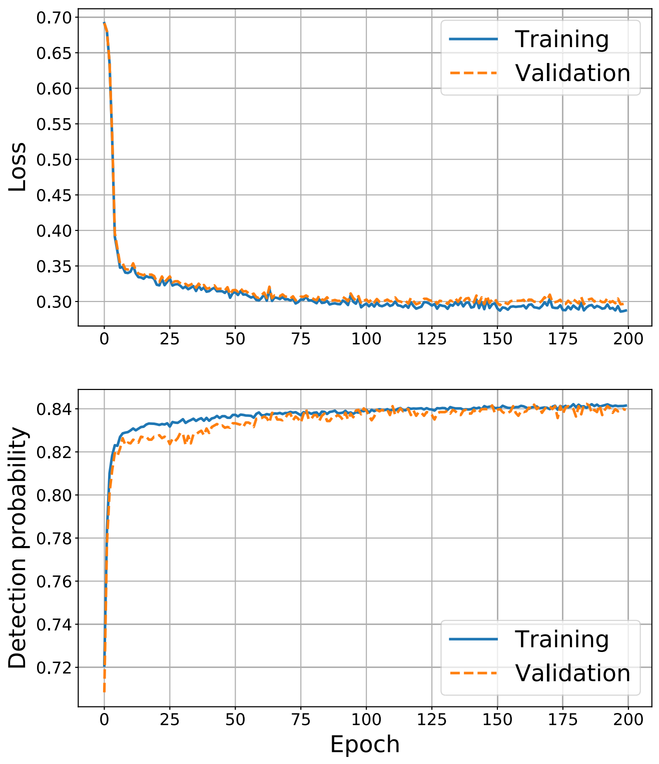

FIG. 7. The loss and detection probability for training and validation sets as a function of training epoch for the vitmap network in the O2 dataset. The loss decreases for both datasets with each epoch where the validation set converges after ~100 training epochs. The detection probability is calculated as the fraction of all signal simulations which exceed the 1% false alarm value.

simulations for a discrete set of SNR we define a window around a point in SNR and count the fraction of statistics which exceed the threshold determined by the false alarm probability within that window. We define the window as a Gaussian with a standard deviation of 2, which is wide enough to contain enough injections at a given SNR to return a reliable value. The detection efficiency $y(\rho)$ is

$$y(\rho) = \frac{\sum_i H(O_i - O^{1\%})\mathcal{G}(\rho_i; \mu = \rho, \sigma = 2)}{\sum_i \mathcal{G}(\rho_i; \mu = \rho, \sigma = 2)}, \quad (6)$$

where O_i is the output statistic from the CNN, $O^{1\%}$ is the statistic value corresponding to a 1% false alarm probability, H is the Heaviside step function which has a value of 1 for positive input arguments and 0 for negative arguments. The SNR of a simulation with output O_i is defined in Eq. (6) using ρ_i. The center of the window in SNR is then ρ. The window is a Gaussian with a mean of the current SNR and a standard deviation of 2, $\mathcal{G}(\rho_i, \mu = \rho, \sigma = 2)$. The sensitivity curves for each of the described datasets are shown in Figs. 8–10.

For the first test, injections were made into the O2 dataset as described in Sec. IV between 100 and 400 Hz. Each of the six networks described in Sec. III E were then trained and tested on this data. Figure 7 shows an example of the training and validation loss and detection probability as a function of training epoch for the vitmap network trained on simulations in O2 data. One epoch is when the entire training set has been passed through the network. Figure 7 shows that for both the training and an independent validation set, the loss and detection probability both converge and perform similarly on each dataset, implying that the network does not overfit.

Figure 8 shows the sensitivity curves for the tests in O2 for both SNR and sensitivity depth for each of the six networks. Focusing on Fig. 8(a), the least sensitive of the CNNs is the Viterbi statistic (vitstat), and this is expected. We know that, despite the line-aware component to the Viterbi statistic calculation, it can still fail to distinguish between some instrumental lines and astrophysical signals. The spectrogram CNN has an improved sensitivity over the Viterbi statistic; this importantly does not involve the SOAP search but is run entirely on downsampled and summed spectrograms. This network is approaching the most sensitive of the examples in Fig. 8. The difference in sensitivity between the spectrogram and the Viterbi map CNN is most likely due to the Viterbi map providing a distilled representation of the spectrograms which is easier for a CNN to interpret. It is possible that the spectrogram CNN could reach the same sensitivity as the Viterbi map CNN by changing its structure or the dataset resolution. However, as explained in more detail in Sec. VI B, this network takes ~10 times longer to train than the Viterbi map network.

The remaining four networks contain the Viterbi map (vitmap) as one of their inputs (or their only input) and all achieve similar sensitivities. It appears therefore that the dominant effect on sensitivity is from the Viterbi maps component. In the following tests our focus will be on the Viterbi map CNN as in all cases this is competitively the most sensitive. For the O2 dataset we show that, with a false alarm probability of 1%, the Viterbi map CNN achieves a sensitivity of SNR 95 and sensitivity depth of 12 $\mathrm{Hz}^{-1/2}$ with 95% efficiency. In Fig. 8(a) the sensitivity of the spectrogram CNN drops after a SNR of 150. This is most likely due to the training set containing simulations between and SNR of 50 and 150, and therefore has not seen signal simulations of higher SNR. The dip in sensitivity in Fig. 8(b) at lower depths is due to the same effect.

For the second test we simulate the S6 dataset with Gaussian noise, retaining the same gaps in the data present in S6 but without including instrumental artifacts such as lines. The noise floor of S6 was also replicated by scaling the SNR of each injection by an estimate of the PSD at that frequency. Figure 9 shows the SNR and depth sensitivity curves for the Viterbi statistic and Viterbi map CNN for both the Gaussian noise run with S6 gaps and for injections into the real S6 dataset. In the Gaussian noise dataset the

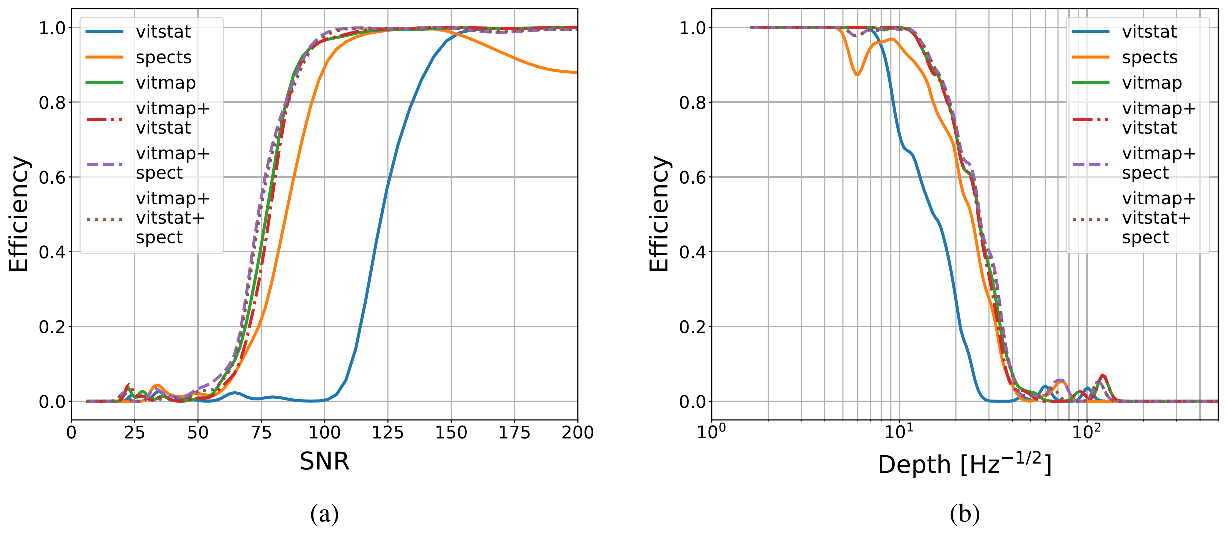

FIG. 8. Tests of the six CNNs with the O2 dataset between 100–400 Hz. The efficiency plots above are for a 1% false alarm probability. Figure 8(a) shows the efficiency of the search as a function of SNR and Fig. 8(b) shows the efficiency as a function of sensitivity depth. The efficiency here is a measure of the fraction of events which exceed the 1% false alarm probability for any given SNR. These plots both show that the sensitivity of the Viterbi statistic is significantly below that of the different CNNs. The others group with a similar sensitivity.

curves for both the Viterbi map CNN and the Viterbi statistic show very similar results. This is to be expected as the main use of the CNN was to reduce the effect of instrumental lines, and there are none in this dataset. The advantage of using the Viterbi maps in a CNN becomes clear when it is tested on simulations into real S6 data with many instrumental lines. The two curves corresponding to simulations in real S6 data in Fig. 9(a) show the sensitivity as a function of SNR in these tests. It becomes clear that the Viterbi map CNN reduces the effect of instrumental lines and increases the searches sensitivity to SNR. A similar feature can be seen in Fig. 9(b) where the use of a CNN again greatly improves sensitivity.

These tests on S6 data also show that the effect of instrumental lines was far greater in this run than in O2. This is shown in Fig. 8(a) where the separation between the

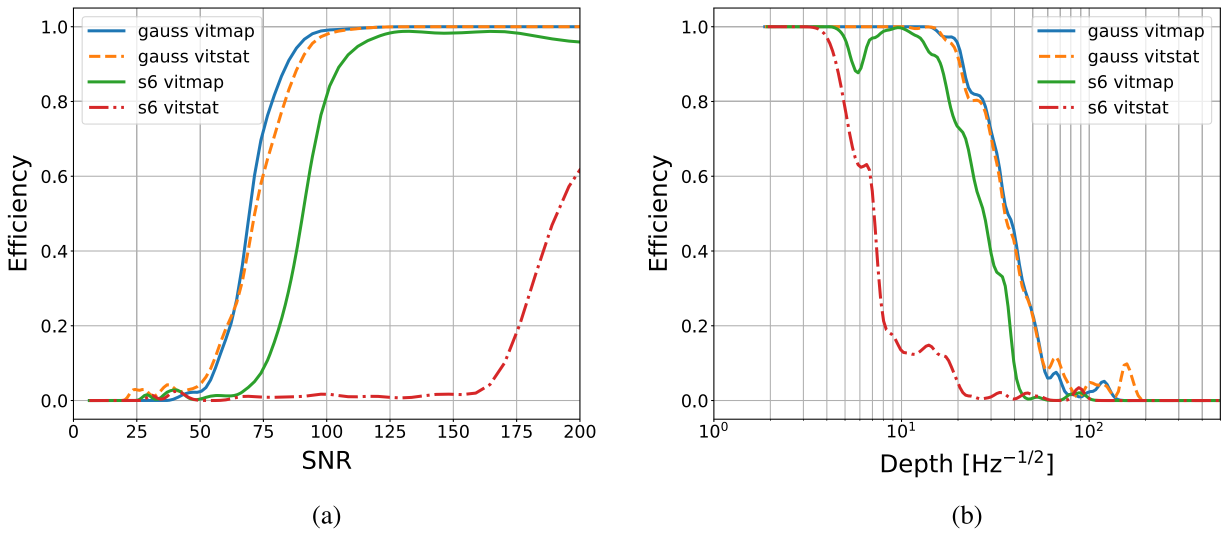

FIG. 9. The sensitivity of the search for simulations in real S6 data (s6) compared to simulations in Gaussian noise (gauss) between 100–400 Hz. Figure 9(a) shows the efficiency of the search as a function of SNR and Fig. 9(b) shows the efficiency as a function of sensitivity depth. The efficiency is the fraction of events which exceed the 1% false alarm threshold for a given SNR or depth. The Gaussian noise injections included the same gaps in data as the S6 dataset and the SNR of the simulated signal in Gaussian noise was adjusted to replicate the expected SNR in S6 data. In the Gaussian noise simulations the searches achieve an efficiency of 95% with 1% false alarm at an SNR ~ 85 and ~ 90 for the Viterbi map and Viterbi statistic respectively. In the real S6 noise simulations the searches achieve an efficiency of 95% with 1% false alarm at an SNR ~ 108 and >200 for the Viterbi map and Viterbi statistic respectively.

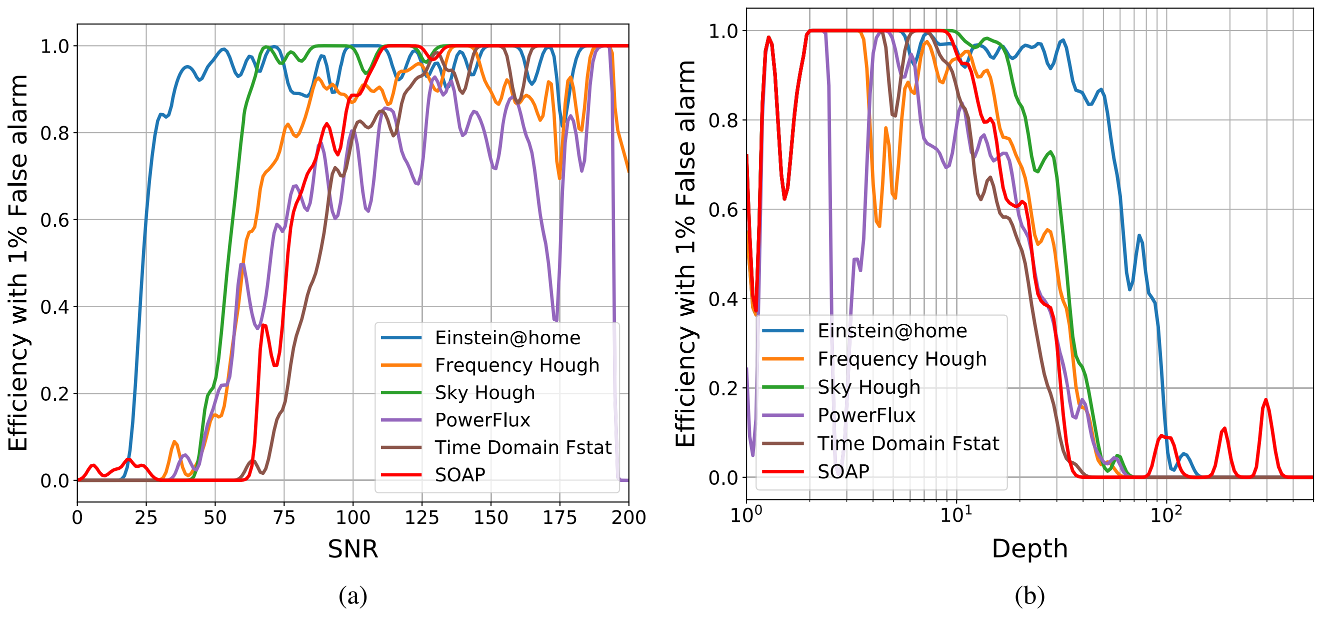

FIG. 10. A comparison of the SOAP + CNN search with other CW searches through a standard set of injections used in the S6 MDC [41]. We have taken the published list of detected sources for each search [41] and replotted using the method in Sec. VI A to compare the sensitivities to the SOAP + CNN search. This includes results for all source simulations between 40 and 500 Hz. The efficiency curves are generated with a 1% false alarm probability. The curves are substantially more noisy than in Figs. 8 or 9 as there are smaller number simulations in a given SNR range.

Viterbi statistic curves and the Viterbi map curves is significantly smaller than the S6 curves in Fig. 9(a). For simulations into Gaussian noise following S6 gaps we show that with a false alarm of 1% the Viterbi map CNN achieves a sensitivity of SNR 85 and sensitivity depth of 20 $\mathrm{Hz}^{-1/2}$ with 95% efficiency. For injections into real S6 data the search achieves a sensitivity of SNR 115 and sensitivity depth of 11 $\mathrm{Hz}^{-1/2}$ with 95% efficiency and 1% false alarm. We can also see from Fig. 9(a) that the sensitivity of the vitmap CNN in Gaussian noise with S6 gaps is better than in real S6 data.

The final test also uses the S6 dataset, however, in this case we use the standard set of injections used in previous CW analysis pipeline comparisonsMDC [41]. Figure 10 shows the resultant sensitivity curves derived from these injections. In both Figs. 10(a) and 10(b) the sensitivity curves are substantially more noisy than in Figs. 8 or 9, mainly due to the smaller size of the testing set. The standard set of simulations in Fig. 10 contained ~900 signal simulations between 40 and 500 Hz where the majority of these signals are distributed between an SNR of 0 and 150. Figures 8 and 9 are generated using 2300 simulations between 40 and 500 Hz and SNRs of 20 and 200 as described in Sec. V. Figure 10(b) shows the direct comparison in depth of the results in [41] with the results from the SOAP search with the Viterbi map CNN. This shows that we achieve a sensitivity consistent with that of other semicoherent searches with the exception of the Einstein@home search [47]. While we are not the most sensitive search by this measure, the SOAP + CNN search offers a greatly reduced computational cost (see Sec. VI B).

This particular test was limited to signals from isolated neutron stars. However, unlike some other semicoherent searches, SOAP is sensitive to a broad range of continuous-wave signals, including binary sources and sources with limited coherence times. The inclusion of the CNN does introduce some dependency on the signal model, as the training set for the CNN contains simulations of isolated neutron stars. However, this is not a limitation of the method: new training sets can be readily generated using a different signal models. For tests in the S6 MDC we show that with a false alarm of 1% the Viterbi map CNN achieves a sensitivity in SNR of ~110 and sensitivity depth of ~10 $\mathrm{Hz}^{-1/2}$ with 95% efficiency.

B. Computational time

A key parameter for any CW search is the computational time it takes to run. Table II shows the timings for different sections of the search using the S6 dataset. This is split into three sections: data generation, CNN training and CNN testing. The majority of the computational time taken to get from raw SFTs to results occurs is the data generation step. The timings shown Table II are for the S6 observing run where each section is run on a single central processing unit (CPU) or graphics processing unit (GPU), however, in practice the generation of the data is run on multiple CPUs on a computing cluster. The training and testing of a CNN is done on a single GPU, this substantially decreases the

TABLE II. Approximate timings for training and testing using the S6 dataset (the longest run we tested), starting from 22 538 SFTs each of duration 1 800 s. The frequency range covered is 40–500 Hz. In the training, testing and search data sections we averaged the SFTs over one day to give 469 time segments as input to the CNNs. The data generation times quoted are for a single CPU however, in reality this will be split across many separate CPUs. The training and testing is completed on a single GPU.

Generating data on single CPU		
	Time [hrs]	
Narrowbanding	~9	
Training data	~240	
Testing data	~75	
Search data	~40	
Training CNNs on single GPU		
	Training time [hrs]	Loading time [hrs]
Viterbi statistic	0.03	0.2
Viterbi map	0.8	0.7
Spectrogram	9	1
Viterbi map + Viterbi statistic	1	0.7
Viterbi map + spectrogram	1.4	1.6
Viterbi map + Viterbi statistic + spectrogram	1.5	2
Testing CNNs on real data on GPU		
	Testing [s]	Loading [s]
All CNNs	5	60–160

training time compared to a CPU due to the intrinsically parallel nature of neural networks.

Starting with raw SFTs covering the 40–500 Hz band of the S6 dataset (i.e., 22 538 time segments each 1 800 s long, giving 828 000 frequency bins in total) and without any trained networks, this search would have a total computing time of ~386 hours on a single CPU and GPU. However, the majority of this time is used making the simulated data. The generation of training, testing and search data can be easily parallelized, and in practice this is split over 200 CPUs and takes just ~2 real-time hours (~355 CPU hours). After this parallelization, if one was given S6 data without any trained networks, the search would then take approximately 13 hours to get an efficiency curve and a list of candidates. In this case I assume that only the Viterbi map network is trained and tested based on the conclusions from Sec. VI.

The computational cost can be reduced further if a network has been trained on a previous observing run, negating the need for new training data and the training itself. This reduces the total run time on S6 to ~9.5 hours (on 200 CPUs). The reduction is significant but not drastic, as the majority of the time is spent narrowbanding the SFTs.

To reduce the time taken to generate results at the end of an observing run, one could narrowband the SFTs periodically as the data is taken during an observing run. This would allow the results to be generated within ~3.5 hours of the end of the run. SFTs generated on a regular basis would allow results to be generated during an observing run. This could be done, for example, on a weekly basis by adding 7 days of pixels to a spectrogram, then retraining a CNN and generating results.

The computational cost of this search is small when compared to other existing CW searches. In [41] the expected computational cost for the first 4 months of O1 for each search is shown, where the fastest search takes 0.9 million core hours (Hough searches) and the slowest is 100–170 million core hours (Einstein@Home). The equivalent cost of the SOAP + CNN search is ~100–200 core hours which is ~5–10 thousand times faster.

VII. SUMMARY

In this paper we summarize an extension of the SOAP algorithm [16]. The extension makes use of a CNN to limit the effect of instrumental lines in searches for astrophysical CW signals. The SOAP search has a number of outputs and in this paper we focused on two of these: the Viterbi statistic and the Viterbi map. The Viterbi statistic has previously been used as a measure of whether a given frequency band contains an astrophysical signal [16]. The Viterbi map is an output map with the same shape as the input spectrogram, with a value related to the probability that a signal passes through any particular time-frequency bin. We use both the Viterbi map and spectrogram as input images to a CNN and classify each frequency band as containing an astrophysical signal or not. This approach removes the need to manually look through frequency bands and remove those which are contaminated with nonastrophysical (instrumental) features.

In detail, we tested six separate CNNs which take in a combination of three representations of the input data: the Viterbi statistic, the Viterbi map and a normalized spectrogram. The objective here was to combine these different representations of the data to increase the robustness and sensitivity of the search. The tests found that the CNN which uses the Viterbi map alone as input was more sensitive than any other which used a single data type as input. Each of the CNNs that used a combination of input data types had a similar sensitivity to the Viterbi map CNN. Therefore we concluded that the Viterbi map provides the most useful summary of information for detecting a signal. Given that the main aim of this investigation was to reduce the effect of instrumental lines on the SOAP search, it is unsurprising that tests with Gaussian noise data (with no such lines) showed the CNN search achieved a similar sensitivity to the Viterbi statistic alone. The tests in Gaussian noise with S6 gaps showed that at a 95% efficiency and a 1% false alarm probability the Viterbi statistic and Viterbi map achieved a sensitivity of SNR 95 and 90 respectively. When the same test was run in real S6 data at a 95% efficiency and a 1% false alarm probability the Viterbi statistic and Viterbi map achieved corresponding sensitivities of SNR 300 and 120 respectively. This demonstrates that the CNN approach adds robustness to SOAP and regains much of the sensitivity that would otherwise be lost to the effects of instrumental lines in real detector data.

These tests were repeated using a standard set of injections into S6 data to make a direct comparison with other CW search pipelines. At a 95% efficiency and a 1% false alarm probability the Viterbi map CNN achieved a sensitivity of SNR ~ 110 and a sensitivity depth $\sim 10\ \mathrm{Hz}^{-1/2}$. We have shown that the SOAP + CNN approach can achieve a similar sensitivity to other semicoherent CW search algorithms but with a greatly reduced computational cost.

This search also offers significant flexibility in the type of signal that is searched for. In the above examples the focus is on isolated neutron stars, largely to make a straight comparison with other CW searches that are tuned for these sources. However, the search framework itself is largely model-free and nonparametric. By changing the training sets, the same pipeline can be optimized for different signal types, and in future work we aim to test its ability to identify other sources of GW such as neutron stars in binary systems. Additionally, we will apply a more advanced Bayesian analysis to estimate basic source parameters which would then provide crucial information for a more sensitive search by fully coherent pipelines.

ACKNOWLEDGMENTS

We would like to acknowledge the continuous-wave working group of THE LIGO-Virgo Collaboration for their assistance during this project. We would also like to acknowledge Sinéad Walsh for providing us with the data from the S6 MDC paper. This research is supported by the Science and Technology Facilities Council., J. B. G. W. and C. M. are supported by the Science and Technology Research Council (Grant No. ST/L000946/1). C. M. is also supported by the European Cooperation in Science and Technology (COST) action CA17137. The authors are grateful for computational resources provided by the LIGO Laboratory supported by National Science Foundation Grants No. PHY-0757058 and No. PHY-0823459.

[1] B. P. Abbott, R. Abbott, R. Adhikari, P. Ajith, B. Allen, G. Allen, R. S. Amin, S. B. Anderson, W. G. Anderson, M. A. Arain *et al.*, Rep. Prog. Phys. **72**, 076901 (2009).
[2] J. Aasi, B. P. Abbott, R. Abbott, T. Abbott, M. R. Abernathy, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari *et al.*, Classical Quantum Gravity **32**, 115012 (2015).
[3] F. Acernese, M. Agathos, K. Agatsuma, D. Aisa, N. Allemandou, A. Allocca, J. Amarni, P. Astone, G. Balestri, G. Ballardin *et al.*, Classical Quantum Gravity **32**, 024001 (2015).
[4] F. Acernese, M. Alshourbagy, P. Amico, F. Antonucci, S. Aoudia, P. Astone, S. Avino, L. Baggio, G. Ballardin, F. Barone *et al.*, Classical Quantum Gravity **25**, 114045 (2008).
[5] B. P. P. Abbott, R. Abbott, T. D. D. Abbott, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. X. Adhikari, V. B. B. Adya *et al.*, Phys. Rev. Lett. **119**, 161101 (2017).
[6] B. P. P. Abbott, R. Abbott, T. D. D. Abbott, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. X. Adhikari, V. B. B. Adya *et al.*, Phys. Rev. Lett. **119**, 141101 (2017).
[7] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari *et al.*, Phys. Rev. Lett. **116**, 061102 (2016).
[8] R. Prix, in *Neutron Stars Pulsars* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2009), pp. 651–685.
[9] R. J. Dupuis and G. Woan, Phys. Rev. D **72**, 102002 (2005).
[10] B. Schutz, Phys. Rev. D **58**, 063001 (1998).
[11] B. P. Abbott, R. Abbott, T. D. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, R. X. Adhikari, V. B. Adya, C. Affeldt *et al.* (LIGO Scientific and Virgo Collaborations), Phys. Rev. D **100**, 024004 (2019).
[12] T. Creighton, Phys. Rev. D **61**, 082001 (2000).
[13] K. Riles, Mod. Phys. Lett. A **32**, 1730035 (2017).
[14] M. Sieniawska and M. Bejger, Universe **5**, 217 (2019).
[15] J. C. Bayley, Soapcw, https://git.ligo.org/joseph.bayley/soapcw (2020).
[16] J. Bayley, G. Woan, and C. Messenger, Phys. Rev. D **100**, 023006 (2019).
[17] P. B. Covas, A. Effler, E. Goetz, P. M. Meyers, A. Neunzert, M. Oliver, B. L. Pearlstone, V. J. Roma, R. M. Schofield, V. B. Adya *et al.* (LSC Instrument Authors), Phys. Rev. D **97**, 082002 (2018).
[18] D. Keitel, R. Prix, M. A. Papa, P. Leaci, and M. Siddiqi, Phys. Rev. D **89**, 064023 (2014),
[19] P. Leaci, Phys. Scr. **90**, 125001 (2015).
[20] S. J. Zhu, M. A. Papa, and S. Walsh, Phys. Rev. D **96**, 124007 (2017).
[21] H. Gabbard, M. Williams, F. Hayes, and C. Messenger, Phys. Rev. Lett. **120**, 141103 (2018).
[22] D. George and E. A. Huerta, Phys. Lett. B **778**, 64 (2018).
[23] T. D. Gebhard, N. Kilbertus, I. Harry, and B. Schölkopf, Phys. Rev. D **100**, 063015 (2019).
[24] M. L. Chan, I. S. Heng, and C. Messenger, Phys. Rev. D **102**, 043022 (2020).

[25] A. Iess, E. Cuoco, F. Morawski, and J. Powell, Mach. Learn. Sci. Technol. **1**, 025014 (2020).

[26] P. Astone, P. Cerdá-Durán, I. Di Palma, M. Drago, F. Muciaccia, C. Palomba, and F. Ricci, Phys. Rev. D **98**, 122002 (2018).

[27] C. Dreissigacker, R. Sharma, C. Messenger, R. Zhao, and R. Prix, Phys. Rev. D **100**, 044009 (2019).

[28] A. L. Miller, P. Astone, S. D'Antonio, S. Frasca, G. Intini, I. La Rosa, P. Leaci, S. Mastrogiovanni, F. Muciaccia, A. Mitidis *et al.*, Phys. Rev. D **100**, 062005 (2019).

[29] F. Morawski, M. Bejger, and P. Ciecieląg, Mach. Learn. Sci. Technol. **1**, 025016 (2020).

[30] E. Cuoco, J. Powell, M. Cavaglià, K. Ackley, M. Bejger, C. Chatterjee, M. Coughlin, S. Coughlin, P. Easter, R. Essick *et al.*, arXiv:2005.03745.

[31] A. J. Viterbi, IEEE Trans. Inf. Theory **13**, 260 (1967).

[32] Y. Lecun, Y. Bengio, and G. Hinton, Nature (London) **521**, 436 (2015).

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998).

[34] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, IEEE Trans. Acoust. Speech Signal Process. **37**, 328 (1989).

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., New York, 2012), Vol. 25, pp. 1097–1105.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[37] D. P. Kingma and J. Ba, in *3rd Int. Conf. Learn. Represent. ICLR 2015—Conf. Track Proc.* (2015) [arXiv:1412.6980].

[38] A. L. Maas, A. Y. Hannun, A. Y. Ng, *Rectifier Nonlinearities Improve Neural Network Acoustic Models* (2013), https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.693.1422.

[39] L. Y. Pratt, in *Advances in Neural Information Processing Systems*, edited by S. J. Hanson, J. D. Cowan, and C. L. Giles (Morgan-Kaufmann, San Fransisco, 1993), Vol. 5, pp. 204–211,

[40] R. Prix, Phys. Rev. D **75**, 023004 (2007).

[41] S. Walsh, M. Pitkin, M. Oliver, S. D'Antonio, V. Dergachev, A. Królak, P. Astone, M. Bejger, M. Di Giovanni, O. Dorosh *et al.*, Phys. Rev. D **94**, 124010 (2016).

[42] S. Patrice, V. Bernard, L. Yann, and D. John S, *Tangent Prop: A Formalism for Specifying Selected Invariances in Adaptive Networks* (Morgan Kaufmann, 1991).

[43] H. S. Baird, in *Struct. Doc. Image Anal.* (Springer Berlin Heidelberg, Berlin, Heidelberg, 1992), pp. 546–556.

[44] S. Van Der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, PeerJ **2014**, e453 (2014).

[45] LIGO Scientific Collaboration, LIGO Algorithm Library - LALSuite (2018), https://lscsoft.docs.ligo.org/lalsuite/.

[46] B. Behnke, M. A. Papa, and R. Prix, Phys. Rev. D **91**, 064007 (2015).

[47] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari *et al.* (LIGO Scientific and Virgo Collaborations), Phys. Rev. D **94**, 102002 (2016).