Convolutional neural networks: a magic bullet for gravitational-wave detection?

Timothy D. Gebhard,[1,2,∗] Niki Kilbertus,[1,3,∗] Ian Harry,[4,5] and Bernhard Schölkopf[1]

[1]*Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, 72076 Tübingen, Germany*
[2]*Max Planck ETH Center for Learning Systems, Universitätstrasse 6, 8092 Zürich, Switzerland*
[3]*Engineering Department, University of Cambridge, Trumpington Street, Cambridge, CB2 1PZ, UK*
[4]*Institute for Cosmology and Gravitation, University of Portsmouth, 1-8 Burnaby Road, Portsmouth, P01 3FZ, UK*
[5]*Max Planck Institute for Gravitational Physics, Am Mühlenberg 1, 14476 Potsdam, Germany*

(Dated: April 19, 2019)

In the last few years, machine learning techniques, in particular convolutional neural networks, have been investigated as a method to replace or complement traditional matched filtering techniques that are used to detect the gravitational-wave signature of merging black holes. However, to date, these methods have not yet been successfully applied to the analysis of long stretches of data recorded by the Advanced LIGO and Virgo gravitational-wave observatories. In this work, we critically examine the use of convolutional neural networks as a tool to search for merging black holes. We identify the strengths and limitations of this approach, highlight some common pitfalls in translating between machine learning and gravitational-wave astronomy, and discuss the interdisciplinary challenges. In particular, we explain in detail why convolutional neural networks alone can not be used to claim a statistically significant gravitational-wave detection. However, we demonstrate how they can still be used to rapidly flag the times of potential signals in the data for a more detailed follow-up. Our convolutional neural network architecture as well as the proposed performance metrics are better suited for this task than a standard binary classifications scheme. A detailed evaluation of our approach on Advanced LIGO data demonstrates the potential of such systems as trigger generators. Finally, we sound a note of caution by constructing *adversarial examples*, which showcase interesting "failure modes" of our model, where inputs with no visible resemblance to real gravitational-wave signals are identified as such by the network with high confidence.

Keywords: Convolutional Neural Networks, Gravitational Waves, Compact Binary Coalescences, Binary Black Holes, Deep Learning, Machine Learning, Binary Classification, Tagging

I. INTRODUCTION

Matched filtering techniques [1–4] have proven highly successful in discovering binary black hole coalescences from the recordings of the Advanced LIGO and Advanced Virgo gravitational-wave observatories [5–11]. Ten observations of merging black holes have now been made [12]. These observations have enabled population studies of the properties of stellar-mass black holes and allowed precision tests of general relativity to be carried out [12, 13]. The most important observation to date was arguably the detection of a binary neutron star inspiral together with a gamma-ray burst and other electromagnetic counterparts [14, 15]. This detection heralds the era of multimessenger gravitational-wave astronomy, has yielded an independent measurement of Hubble's constant, and probed the behavior of matter at the core of neutron stars [16, 17].

Additional observatories in Japan and India are expected to become operational in the next five years forming an evolving detector network capable of observing hundreds of sources every year [18, 19]. These sources will need to be rapidly observed, localized in the sky and this information quickly disseminated to electromagnetic partners to maximize the chance of multi-messenger observations [19]. This requires reliable, real-time identification of potential compact binary coalescences (CBCs) to provide a time window and basic parameter estimate for slower, but more accurate Bayesian inference techniques to follow-up [20, 21]. However, current matched filtering techniques are computationally expensive, with the computational cost scaling as a function of the broadness of the detector's sensitivity curve and the number of observatories; both of which are expected to increase in the coming years [19].

In this work, we investigate whether some of these challenges can efficiently be overcome by using deep convolutional neural networks (CNNs). CNNs are a machine learning technique that has been employed successfully on a wide variety of tasks, including image classification [22–24], natural language processing [25] and audio generation [26]. In the physics community, an early application of CNNs was [27]; Carleo *et al.* [28] provide a review of recent developments in this direction. In particular, CNNs have also been studied in the literature as a tool for gravitational-wave searches, and previous works have shown that they can indeed be effectively applied to this problem when treating it as a binary (i.e., two-class) classification task [29, 30].

However, despite these promising preliminary results, we believe that the precise role that machine learning can play within the larger scope of CBC searches and practical multi-messenger gravitational-wave astronomy has not yet been assayed in sufficient detail. The main goal

---
∗ These two authors contributed equally.
  Correspondence: {tgebhard,nkilbertus}@tue.mpg.de.

of this work is, therefore, to carefully and realistically analyze the practical potential of using CNNs to search for GWs from CBCs. Here, we pay particular attention to realistic data generation, an appropriate, task-specific architecture design and adequately chosen performance metrics. This results in the following main contributions:

1. We provide an in-depth analysis of the challenges one may expect machine learning to solve within the scope of a search for GWs from CBCs, and also discuss their limitations in replacing matched filtering or Bayesian parameter estimation techniques.

2. We extend the existing, binary classification-based approach of using CNNs to also handle inputs of varying length. This requires the introduction of new task-specific performance metrics, which we discuss and relate to the existing metrics.

3. We highlight potential challenges and subtle pitfalls in the data generation process that may lead to unfair comparisons. To facilitate further research and reproducibility in this area, we release the data generation workflow we have developed as a reusable open source software package.

4. Finally, the empirical results of our architecture indicate that deep convolutional neural networks are a powerful supplement to the existing pipeline for fast and reliable trigger generation. However, we also demonstrate that—like most deep neural networks—our architecture is also prone to *adversarial attacks*: We can construct inputs with no visible resemblance to gravitational-wave signals that are nevertheless identified as such by the model.

As a key aspect of this work, we aim to foster communication and understanding between disciplines: On the one hand, we hope to help physicists less acquainted with deep learning techniques understand the strengths and limitations of such methods in gravitational-wave searches and gain intuition towards how they function in this context. Simultaneously, for machine learning experts, we explicitly highlight some problem-specific subtleties—ranging from data generation to model architecture design and meaningful evaluation metrics—to help them to circumvent tempting pitfalls.

The rest of this paper is structured as follows. In section II, we revisit matched filtering (with a focus on the implementation by PyCBC). Furthermore, we discuss the existing literature on using CNNs in the context of gravitational-wave searches. In section III, we then continue by reviewing the previously used binary classification framework more principally, and discuss for which specific tasks CNNs may be useful and for which their output is insufficient. Consequently, after introducing our carefully designed data generation procedure and the corresponding open source software package in section IV, we suggest a fully convolutional network architecture suited for gravitational-wave trigger generation

in streaming data in section V. This architecture naturally gives way to novel performance metrics, which we develop in section VI, where we also explain their benefits and relation to current standard metrics. In section VII, we present and discuss the results of our model together with a note of caution concerning adversarial examples, highlighting the still not well-understood and unsettling brittleness of deep neural networks. Finally, we conclude with a summary and outlook in section VIII.

II. PROBLEM SETUP AND RELATED WORK

Observing compact binary coalescences has always been one of the primary goals of gravitational-wave astronomy. To date, searches for such systems rely on matched filtering using a large template bank (i.e., a set of simulated waveforms covering a carefully chosen parameter space). In the first part of this section, we will describe matched filtering with a specific focus on the implementation provided by the PyCBC software package [3, 31]. We explain the necessary components for a statistically sound search procedure and explain what it means to "detect" a gravitational wave. Readers familiar with the matched filtering search pipeline may wish to skip parts II A, II B and II C. In part II D, we then review the existing work using convolutional neural networks for gravitational-wave searches.

A. Matched filtering-based searches

Schutz [32] vividly describes the intuition behind the matched filtering technique as follows: "Matched filtering works by multiplying the output of the detector by a function of time (called the template) that represents an expected waveform, and summing (integrating) the result. If there is a signal matching the waveform buried in the noise then the output of the filter will be higher than expected for pure noise."

In the following, we will formalize this idea mathematically in order to provide the necessary background for a comparison between matched filtering and the outputs of deep learning-based systems later on. Readers interested in further details are referred to the excellent overview of matched filtering in the context of the LIGO and Virgo collaborations by Caudill [33] (and references therein).

The fundamental assumption of matched filtering is that the *strain* $s(t)$ measured by the interferometric detector is made up of two *additive* components, namely the *instrument noise* $n(t)$ and the (astrophysical) *signal* $h(t)$:

$$s(t) = n(t) + h(t) \tag{1}$$

For a given power spectral density S_n of n, we can then quantify the agreement between a given template $T(t)$ in the template bank and the recorded strain $s(t)$ at a time t_0 by computing the *signal-to-noise ratio* (SNR).

For an appropriate choice of normalization, the matched filtering signal-to-noise ratio is given by:

$$\mathrm{SNR}(t_0) := \int_{-\infty}^{\infty} \frac{\tilde{s}(f) \cdot \tilde{T}^*(f) \cdot e^{2\pi i f t_0}}{S_n(f)} \, \mathrm{d}f \,, \qquad (2)$$

where the tilde denotes the Fourier transform. For stationary Gaussian noise it can be shown that—by design— the SNR is indeed the optimal detection statistic for finding a signal $h(t)$ if the time-reversed template $T(-t)$ is equal to the signal [1]. This is called the *matched* filter. In practice, the template bank should therefore contain accurate simulated waveforms that cover the space of expected signals in the recorded data sufficiently densely. Computing the SNR for every waveform in the template bank and applying a threshold then produces a list of candidate event times.

In reality, however, the data is usually neither stationary nor exactly Gaussian. One particular challenge to the data analysis are so called *glitches*. Glitches are non-stationary noise transients, which comprise a range of different short-time phenomena that affect the quality of the data measured by the detectors. They occur frequently, at rates up to several times per minute [34]. Some of these effects are well understood, such as the signature of scattered light in the beam tube; others, however, remain enigmatic. For example, a certain common type of glitch named "blip", whose origin is only poorly understood, tends to mimic the signals that one would expect from the merger of two intermediate-mass black holes, thus limiting the sensitivity for this kind of event [35].

As a consequence of these non-Gaussian and non-stationary effects, the real distribution of the SNR (and thus the threshold value) is not known and must be determined empirically in order to obtain calibrated statistical results from the computed SNR. Allen *et al.* [1] provide a detailed account of the merits and challenges of matched filtering in practical gravitational-wave searches.

B. The PyCBC search pipeline

To understand the crucial components of a *full* search (which ideally results in a *detection*), we now outline the current PyCBC search pipeline [3]. The different steps of the search procedure are also illustrated schematically as a flowchart in Figure 1.

In a first step, a template bank containing simulated waveforms that cover the parameter space of interest is constructed; typically using the simulation routines provided by LALSuite [36], the central codebase that implements all waveform models used in Advanced LIGO and Advanced Virgo analyses. For more technical details we refer the reader to, for example, Capano *et al.* [37].

This template bank is then used to compute an SNR time series for every possible combination of templates and recordings (i.e., we match every template with every observatory). We then find the times of peaks within all
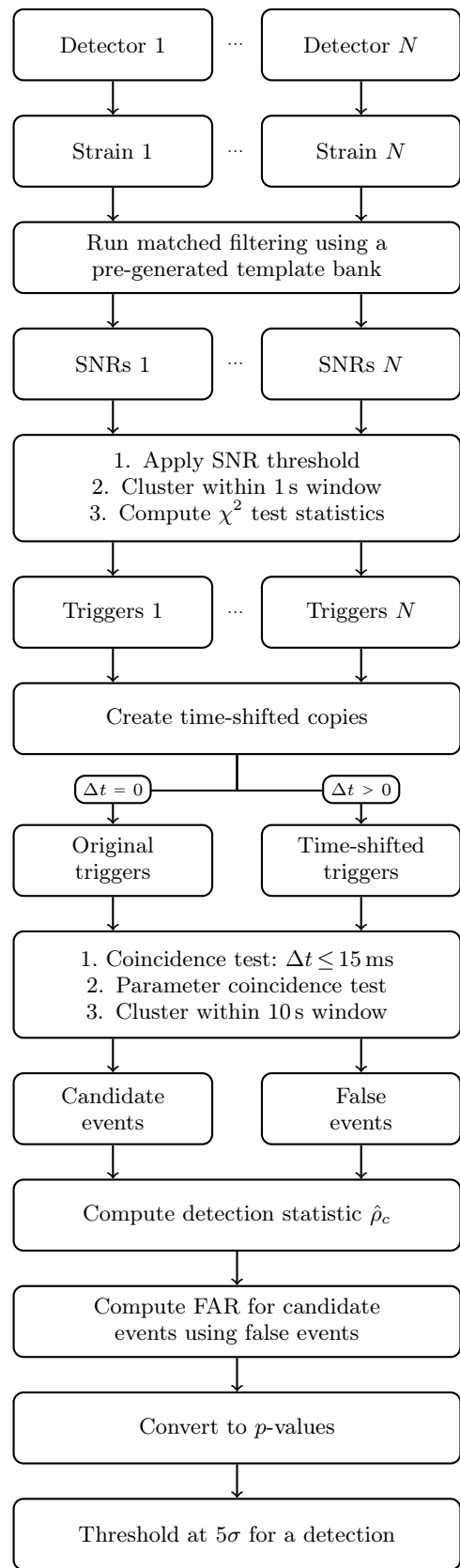

Figure 1. Flowchart of the PyCBC search pipeline, which shows the full process of going from the recordings of the different observatories to the detection of a gravitational wave.

these SNR time series that exceed a certain pre-defined threshold. Next we cluster these times to keep only the times of largest SNR within a 1-second window and then store the remaining times alongside the parameters of the template that caused the match. Each of these recordings is called a *trigger*.

Consequently, we obtain a list of single detector triggers for each observatory independently. Furthermore, a set of signal consistency tests—χ^2 tests—are computed for every trigger, which help to discriminate between real events and triggers that were caused by noise transients [38]. More precisely, these χ^2-test values are used to compute a re-weighted single detector SNR which serves as a ranking statistic. In a subsequent stage, several coincidence tests (for both the event time and the estimated event parameters) are conducted: the single detector triggers are combined if the same template matched at compatible times (i.e., within light distance of each other) in all detectors. The resulting coincident triggers are called *candidate events*. Finally, each candidate event is assigned a combined ranking statistic, informally called *loudness*, which is computed from the parameters of the triggers in each observatory. The precise mathematical definitions of the individual and combined ranking statistics are hand-tuned and regularly adjusted (see, for example, Nitz *et al.* [39] or Nitz [40]).

Note that while the loudness is designed to intuitively correspond to our confidence of the candidate being a real event (higher scores indicating higher confidence), the raw numerical values have no significance. Instead, we are interested in the relative ordering of the candidate events that is induced by the loudness score. To claim a *detection*—that is, to say that a candidate event with a given loudness in fact corresponds to a true gravitational-wave signal—we must perform the following statistical test: within our model assumptions, what is the probability that we observe this loudness purely by chance, if in reality there is no gravitational-wave signal present? This probability measures the statistical significance of the detection, that is, the confidence with which we can reject the null hypothesis, namely "there was no real signal in the data".

At this point, it is crucial to contrast this with deep learning based machine learning classifiers. The output of such classifier on a single example—for example, from a softmax or sigmoid output layer—is also between 0 and 1 and thus at times interpreted as a probability. However, these "probabilities" only reflect the "degree of confidence" of the network regarding its prediction. Therefore, they must not be interpreted as the statistical significance of a detection (see also section III).

In PyCBC, the probability of obtaining a given loudness from only noise is estimated via frequentist inference over a given time period. To this end, a matched filtering search is performed on a recording of given length T that is known to not contain any gravitational-wave signals. We then count the number of resulting candidate events that are at least as loud as the candidate event.

To obtain data that is guaranteed to not contain any gravitational-wave signals but still shares characteristics of real detector recordings, PyCBC makes use of time shifts. It shifts the recordings of the detectors relative to each other by a time period that is larger than the light travel time between them (see again Figure 1 for where this fits in the pipeline). Assuming that gravitational waves above the detection threshold of the instrument are sparse in time (i.e., further apart than the time shift), this ensures that no real signal will pass the coincidence tests and give rise to a candidate event. Instead, any candidate event found for a time-shifted input must be due to triggers caused by the random detector noise. Therefore, the loudness scores of candidate events found in time-shifted data can be used to estimate the frequency of false positives. This further allows us to derive false alarm rates for candidate events in the non time-shifted data and ultimately assign a statistical significance to a claimed detection. For a slightly more detailed yet compact description of how to estimate these probabilities in practice, we again refer to Caudill [33].

C. Injections

To conclude this introduction to the existing search pipeline, we note that due to the relatively small number of events detected so far, a proper performance evaluation of *any* search approach hinges on so called *injections*. An injection is a simulated waveform that is added into a piece of background noise (either synthetic or real) to emulate a real gravitational-wave signal as it would be observed by an actual detector. The search performance can then be evaluated by searching for a large variety of such injections added to the recorded strain data. Because in this case we know the precise location of the injections, we have access to the ground truth required to evaluate the detection rate and false alarm rate of the search pipeline for a given template bank, real recordings, and injections.

In the previous paragraphs, we have glanced over the fact that we can only compute false alarm probabilities and detection rates *within our model assumptions*. These assumptions include—among other factors—the parameter ranges and distributions of simulated waveforms both for the template bank and injections. Since the true physical distribution of gravitational-wave sources in the Universe (not only in terms of location, but also in terms of the parameters of their constituents) is unknown, these choices will not only affect how the obtained performance results transfer to real searches, but also influence the sensitivity towards various sources. In section IV, we comment on this in a little more detail. However, a full discussion of how to properly incorporate such ad hoc choices in the statistical analysis of the method is beyond of the scope of this work.

D. Existing CNN-based approaches

The idea of using convolutional neural networks (CNNs) to process time series information goes back to the early days of deep learning itself, more than twenty years ago [41]. Ever since, the community has established CNNs as one of the major work horses for processing images as well as time series data like audio (or various time-frequency representation thereof), which is structurally similar to the strain data produced by gravitational-wave observatories. CNNs have been particularly successful in supervised classification or regression tasks, where they are typically trained to map inputs in $\mathbb{R}^d$—for example, images of a fixed resolution or fixed-length audio snippets—to either a finite set of labels (classification) or a typically low-dimensional real vector (regression).

All previous work applying convolutional neural networks to the detection of gravitational-wave signals in interferometric detector data has adopted a classification-based approach. George and Huerta [29] generate white Gaussian noise examples with a fixed length of 1 s and, for a subset of them, add simulated gravitational-wave signals from binary black hole mergers similar to the injections in the PyCBC search. The maximum of the signal (which corresponds to the coalescence time) is randomly located in the last quarter of the sample. Using these data, they train a deep neural net, consisting of a common combination of convolutional and fully-connected layers with a final sigmoid layer, to output a value between 0 and 1, indicating the confidence of the network about the absence or presence of a gravitational-wave signal in each 1 s example. The network output can be thresholded to obtain a binary response. In addition, they train a second neural network, which estimates some basic parameters of the corresponding binary merger whenever the first network claims to have found a signal. In this setup, the CNN's task is to detect non-Gaussianities of a specific form in white Gaussian noise, where the non-Gaussianities fall within a specific region of the input snippet.

In later works, they also evaluate this method on 1 s snippets of real LIGO recordings, and on an enlarged dataset which also includes waveforms for binary black hole mergers with precessing spins and non-vanishing orbital eccentricities [42, 43]. Longer samples are processed by a sliding-window approach: recordings are split into overlapping 1 s-windows to each of which the trained network is applied. Multiple detectors are accounted for by processing each recording separately first and then combining the binary outputs at each time via a logical AND function. Notably, the authors suggest that their method can be used for gravitational-wave detection as well as parameter estimation and that it beats matched filtering in terms of errors and computational efficiency while retaining similar sensitivity [43]. We will explain in section III why we believe that a more careful and nuanced interpretation of such claims is essential to understanding the practical merits of CNN based approaches.

Gabbard *et al.* [30] employ a similar approach: the authors also use a deep neural network consisting of both convolutional and fully connected layers to perform a binary classification task on 1 s samples of Gaussian noise which either do or do not contain a simulated GW signal. The focus of their work, however, is the comparison with matched filtering. They conclude that their method is indeed able to closely reproduce the results of a matched filtering-based search on these 1 s samples.

A somewhat different approach was presented by Li *et al.* [44]. In their method, they use a wavelet packet decomposition to pre-process the data before feeding it into a convolutional neural network, which then operates on a frequency representation. They also work with a sliding-window approach to apply their network to samples of variable length. Ultimately, the practical conclusions of their work are limited by the fact that they use Gaussian noise for the background and an unrealistically simplified damped sinusoid as an analytical waveform model.

Finally, there is also a growing body of work which uses CNNs for various tasks that are different from but related to a gravitational-wave *search*, such as *glitch classification* (e.g., [45–49]) or *parameter estimation* (e.g., [50]).

III. GOING BEYOND BINARY CLASSIFICATION

In this section, we develop our main conceptual contributions, namely that a) convolutional neural networks are not suited to claim statistically significant detections of gravitational waves, however, b) they can still be useful tools for real-time trigger generation.

Our core argument for claim a) hinges on the fact that the "false alarm rate" which can be derived from machine learning-based classifiers is directly linked to the training dataset. As a consequence, there is only a single significance level that one can assign to every claimed detection, without being able to distinguish particularly loud events from fainter ones. Additional difficulties stem from the fact that in a real search, the task at hand is not to perform binary classification on fixed-length examples, but to identify the temporal location of potential signals in time series data of arbitrary length, or even in streaming data. The significance level obtained in the example-based binary classification setup does not transfer easily to sliding-window based approaches for streaming data.

To substantiate b), we highlight the benefits of CNNs in terms of computational complexity and devote the remaining sections of this paper to developing a modified CNN architecture which can overcome many of the pitfalls of the binary classification approach.

A. True / false positive rate and class imbalance

Standard performance metrics for classification tasks are the *true positive rate* (TPR; also called *recall*) and the *false positive rate* (FPR), which are defined as:

$$\text{True Positive Rate (TPR)} := \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{False Positive Rate (FPR)} := \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

Here, TP are true positives (i.e., examples correctly classified as positives), FP are false positives (i.e., examples falsely classified as positive; Type I error), TN are true negatives (i.e., examples correctly classified as negative) and FN are false negatives (i.e., examples falsely classified as negative; Type II error).

Indeed, all previous comparisons of CNNs use a binary classification framework and compare the true positive rate at fixed false positive rate directly to matched filtering results at a given false alarm rate [30, 42, 43]. To obtain this measure in practice, for threshold-based binary classifiers, one usually sweeps the threshold from 0 to 1, recording the true positive rate and the false positive rate for each threshold value to produce the receiver operator characteristic (ROC) curve, that is, the true positive rate over the false positive rate. Since the false positive rate is maximal for threshold 0 and minimal (zero) for threshold 1, we can then simply read off the true positive rate for any given false positive rate. However, there is a subtle difference between the generalization properties of this population level false positive rate and the false alarm rate in matched filtering.

Intuitively, we may interpret the CNN as an implicit abstract representation of all the examples—with and without simulated waveforms—which it has seen during training. In that sense it does not directly capture a compressed version of the template bank alone, but the entire training distribution including the ratio of positive and negative examples. Therefore, unlike matched filtering, the network's output on new inputs depends also on the relative frequencies of positive and negative examples in the training set and the above performance measures only transfer to unseen examples following the exact same distribution. Consequently, performance evaluations of CNNs on the training distribution (many examples with injections) do not transfer to the test distribution (real recordings with few signals) as is the case for matched filtering, where the output depends only on the template bank. For efficient and stable training, the number of positive and negative examples should be on a similar order of magnitude, which is a clear misrepresentation of the true distribution and calls for caution when interpreting the FPR on hand-crafted training or validation sets as false alarm probability in a full search on real data. We note that in [43], the authors have computed an estimate of their FPR by applying their trained network to a continuous stretch of real LIGO data.

B. Performance vs. detection

The core task of gravitational-wave searches is not a population-level performance rating of the search pipeline on synthetic data, but to ascertain the *individual* statistical significance of a given candidate event. Hence, we must ask ourselves the question: What would be our level of confidence that there is a real event in the data when a binary classifier outputs a 1? Here is the problem: If we were to use the false positive rate as a level of confidence for a claimed detection of the CNN (output 1), we would assign the same confidence to every candidate! In particular, we would have no way of distinguishing particularly significant detections from fainter ones. This is due to the fact that the false positive rate is a statistic of the network output on the entire dataset, not any given example. Furthermore, as described above, the interpretation of the false positive rate as a confidence is only valid if the test distribution (actual detector recordings) comes in well-defined, distinct fixed-length examples which follow the same distribution (including the frequencies of positive and negative examples) as the training set. Therefore, while the false positive rate may seem like a tempting, convenient measure for the false alarm probability of CNNs, it must not be interpreted as a statistical significance. Consequently, CNNs alone cannot be used to properly claim gravitational-wave detections.

C. Classification vs. tagging

In a real search, we must identify and annotate those parts of an arbitrarily long input time series that contain a signal. The existing works extend the binary classification-based approach to longer inputs via a sliding window approach. In addition to the fixed input size of the classifier, this requires yet another parameter choice, namely the step size of the sliding window.

Both of these parameters influence the performance metrics directly and in ways that are hard to interpret. First, the tempting conversion of "FPR × example length = temporal rate of false positives" becomes invalid due to the overlap between neighboring windows. Second, depending on the step size of the sliding window, waveforms may lie only partially within the input window, which can then not be labelled as one or zero in a principled fashion. Moreover, there is no natural interpretation of the sequence of outputs. For example, assume the CNN outputs the sequence $1 - 1 - 0 - 1 - 1 - 0 - 1$, where the coalescence happens roughly at the center value. How should these labels be counted as true (false) positives (negatives)? The interpretation would perhaps also depend on the time step, that is the *temporal resolution*, and the window size. Finally, while a high temporal resolution (small step size) would be desirable in order to localize the signal in time, it also leads to computational redundancy as we will further elaborate in section V.

All in all, the metrics derived from the example-based binary classification setup do not easily transfer to the sliding window approach on streaming data; a fact which has largely been overlooked in the literature so far.

D. Overfitting

We have seen that in the example-based approach, we cannot easily process inputs with partially contained waveforms. Previous works have therefore positioned injections only in specific regions within the examples, usually such that the coalescence is located towards the end.

Deep learning systems are known to pick up unintentional quirks in the training data which correlate with the labels. This can result in an undesirable behavior called *overfitting*, where a classifier learns to perform well on training data, but fails on new examples in the real application. In the above example, the CNN may overfit on the location of the coalescence within the training examples. In particular, the final, fully connected layer(s) can learn location-sensitive features. Since the coalescence is the most pronounced part of the waveform, if it is always located in the same region, a network containing fully connected layers may focus exclusively on high amplitude, high frequency oscillations in this region, ignoring other parts of the input.

One crucial measure to avoid overfitting is to make the training set as representative as possible of the context in which the model will be deployed to reduce its potential to adapt to irrelevant characteristics of the training data. In sections IV and V, we discuss a data generation process and network architecture that pay particular attention to minimizing the danger of overfitting.

E. Use-case for deep learning

To conclude this section, let us discuss how CNNs can still complement matched filtering-based searches (instead of replacing them). Looking into the future, various upcoming challenges of matched filtering concern growing computational needs. For example, as more detectors come online, the computational complexity of matched filtering scales at least linearly in the number of detectors (recall that the search for triggers is performed independently for each detector first). Moreover, this trigger generation scales linearly also in the number of waveforms in the template bank. As template banks grow, matched filtering becomes increasingly expensive, causing real-time online trigger generation to become computationally challenging and prohibitive.

Such computational considerations are a key part of the motivation to look into alternative search methods in the first place. Convolutional neural networks are natural candidates, because inference—evaluating the network on new strain data after it has been trained—can be substantially faster than matched filtering. Our architecture (see section V A) scales to an arbitrary number of detectors with almost no computational overhead. Furthermore, once an architecture is fixed, it can in principle be trained on any distribution of simulated waveforms. Thus, we can view the network training as building an abstract, constant size representation of the template bank. Note that the computational cost of inference is independent of the size of the training data. The expensive training of the network is performed only once up front.

The benefit of fast inference of CNNs—they analyze detector recordings much faster than real-time—makes them natural candidates for trigger generators. Real-time alarms can provide useful hints for follow up searches of electromagnetic counterparts as well as for focused analysis with matched filtering and Bayesian parameter estimation [51]. Arguably, a straightforward extension to also provide rough first parameter estimates could further decrease the computational cost of subsequent analysis by narrowing down the parameter space.

Moreover, while CNNs do not enjoy theoretical guarantees for stationary Gaussian data like matched filtering, one may speculate that they can, in principle, incorporate mechanisms to better deal with common non-Gaussianities in the data by learning internal models not only of waveforms, but also of transient glitches. Testing and quantifying this hypothesis is left for future work.

In the remainder of this work, we develop a promising proof of concept implementation for such a use-case that avoids many pitfalls presented earlier in this section.

IV. DATA GENERATION PROCESS

In this section, we describe the steps we have taken to generate realistic, synthetic data which can be used to train and evaluate a CNN-based model. We discuss our design choices and explain steps where we found a need to compromise between realistically modeling physics on the one hand and the requirements for efficient and reliable machine learning on the other hand. For reasons of transparency and reproducibility, as well as to foster further research in the area, we will make our data generation code publicly available online at [52].

A. Choice of background data

When choosing background data, one has essentially two options: *simulated* Gaussian noise, which is then *colored* using the Power Spectral Density (PSD) of the detectors, or actual detector recordings (in which the existing matched filtering pipeline did not find any gravitational-wave signals). While the first option yields background data that has on average the correct frequency distribution, it will not contain glitches. However, as discussed before, glitches are one of the major challenges for the data analysis. Therefore, we have decided

to use real LIGO recordings from the first observation run (O1) to emulate the background noise. O1 included the first three discoveries of gravitational waves: GW150914, GW151012 and GW151226 [7, 8, 12]. The exact detector configuration during O1 is described in detail in [53–55].

The data from O1 is publicly available through the Gravitational Wave Open Science Center (GWOSC; see also [56, 57]). In our study, we limited ourselves to a subset of the data, specified by the following criteria:

- **Data available:** Both H1 and L1 must have data available (due to different times when the detectors are operating, this is not always the case).

- **Minimum data quality:** For the scope of this study, the data needs to pass all vetoes for CBC searches, meaning that only recording segments with data quality at least `CBC_CAT3` (using the GWOSC definitions) are used.

- **No hardware injections:** The data on GWOSC does already contain a small number of simulated transient signals called *hardware injections* [58]. We exclude all segments containing such signals.

- **No real signals:** We also exclude the *real* events in O1 (i.e., GW150914, GW151012, GW151226).

B.   Generating a Data Set

In this section, we give a detailed account of our data generation process, which is visualized in Figure 2.

In order to generate a new example, we first need to select a piece of LIGO recording to be used as background. To this end, we keep drawing a GPS time t_{GPS} between the start and end of O1 uniformly at random until we find a valid time. A time t_{GPS} is considered valid when the symmetric Δt interval around it fulfills the four criteria defined above. To save memory, this interval is then down-sampled from the original sampling rate of 4096 Hz of the GWOSC data to 2048 Hz. Note that Δt should be chosen larger than half the desired sample length, because we will later compute the (discrete) Fourier transform as part of a whitening procedure. This corrupts the edges at both ends, which need to be cropped off.

In parallel, a set of parameters for the waveform simulation is sampled from the joint distribution over the entire parameter space. This study is limited to waveforms from mergers of binary black holes, which are simulated using the effective-one-body model `SEOBNRv4` in the time-domain [59]. Therefore, we need to randomly sample values for the masses of the black holes, the z-components of their spins, the right ascension, declination, polarization, inclination, and coalescence phase angle (which together specify the location and orientation of the source in the sky), as well as the injection SNR. For more details about these parameters, see appendix A.

Choosing the distributions of these parameters is a good example for the contradicting requirements of correctly modeling physics on the one hand and the practical concerns of the ML side. In reality, most of the GW signals are expected to be very faint, because their sources are comparatively far away: If we assume the sources to be distributed isotropically and uniformly in space over the whole (spherical) search volume, approximately half of all sources will be at 80% or more of the maximum sensitive distance. However, if this r^3-dependency is modelled correctly when sampling parameters for simulating training data, a large fraction of the data will be barely above the detectability threshold. This makes it hard for the machine learning methods to actually learn anything. One common approach in deep learning to address this kind of problem is to split the training into different phases, first training on "easy" examples (in this case events with strong GW signals), and then gradually replacing or complementing the training set with "harder" (i.e., fainter) examples. In our experiments, however, this so-called *curriculum learning* [60] did not lead to relevant improvements of the final performance.

The simulation routines in LALSUITE return two time series for given parameter settings, namely the two polarization modes of the gravitational wave, h_+ and $h_\times$. These are then transformed according to the interferometer *antenna patterns*, which are functions that describe the directional sensitivity of the detector. [61] PyCBC provides methods to calculate the projection onto the antenna patterns for the detectors in Hanford and Livingston for a given source location in the sky and a corresponding polarization angle. Finally, the simulated detector signals also need to be corrected for the time offset between the detectors, based again on the relative source location in the sky. This gives us the "pure" signals that the detectors would observe in the absence of noise.

Next, these signals are *injected* into the noise that we selected in the beginning. For comparison later on, we would like to know how "loud" the injection was. This can be measured by the *optimal matched filter SNR* (e.g., [62, 63]) of the injection, which is the maximal SNR possible resulting from using the time-inverted signal itself as a filter. This is achieved by a two-step process:

1. First, we simply add the two time series (noise and signal) in such a fashion that the peak of the signal amplitude in H1 is centered within the noise interval. Afterwards, we compute the optimal matched filtering signal-to-noise ratio in both detectors, and subsequently also the network optimal matched filtering SNR (NOMF-SNR). The latter is then used to determine a scaling factor by which the waveform needs to be multiplied to ensure that the injected signal has the desired *injection SNR*. This is possible because multiplying the waveforms of *both* detectors by a factor λ results in a network SNR that has been scaled by the same factor λ. From an astrophysical perspective, rescaling simply
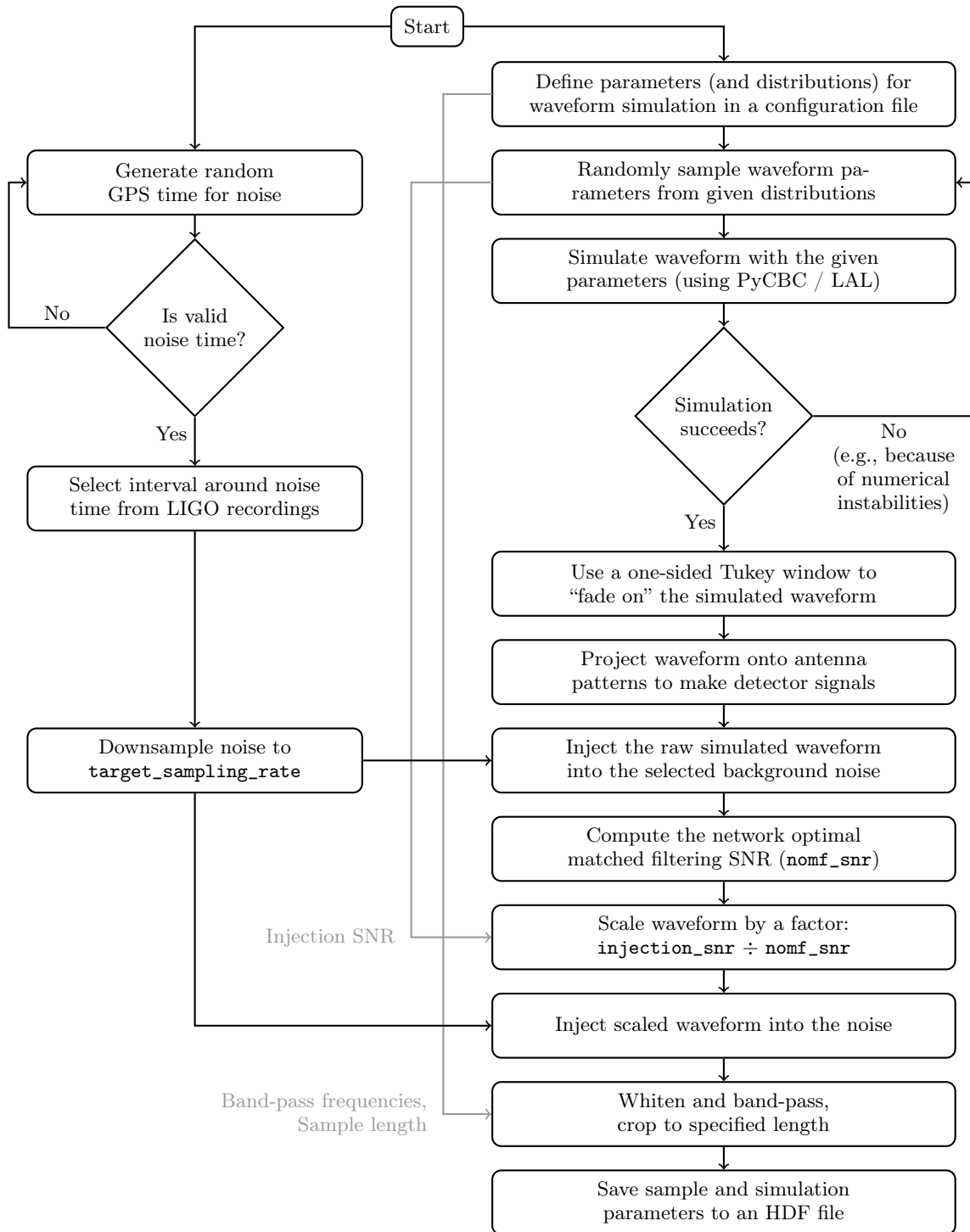
Figure 2. This flowchart visualizes the process that was used to generate synthetic training and testing data by injecting simulated waveforms into background noise comprised of real LIGO recordings.

corresponds to moving the source closer or further away from the detectors.

2. Now we can add the rescaled waveform to the noise, which guarantees that the sample has the desired network SNR.

The result is then whitened with PyCBC using a local estimate of the power spectral density, and high-passed at 20 Hz to remove some of the non-physical turn-on artifacts from the simulation. Finally, the example is cropped to the desired length (which was chosen as 8 s) in such a fashion that the maximum of the signal always

ends up at the same (relative) location within the sample. This is permitted, because our particular choice of model architecture (see below) is not sensitive to the position of the signal within a sample. The choice of 8 s for the length was governed by memory considerations: training a neural network efficiently requires that both a mini-batch of training examples and the network parameters (together with their gradients) fit into memory of a graphical processing unit (GPU).

C. Training and testing data sets

For this work, we created three data sets: a training dataset with 32 768 examples, a validation set with 4096 examples, and a testing dataset with 16 384 examples. The parameters for the waveform simulation were drawn independently from the same joint distribution over the parameter space (see appendix A) for all three data sets. All data sets are mutually disjoint, that is, no single example is used for both training and testing / validation.

To ensure that during training the net is also exposed to sufficient data which do not contain any signals, a number of examples not containing any injections is generated by simply skipping the injection step. We use three times as many examples that contain an injection than pure noise examples.

In section VII, we also evaluate our trained model on real signals from LIGO's first observation run, which have undergone analogous pre-processing (whitening, bandpassing) like the training data.

V. MODEL AND TRAINING PROCEDURE

In this section, we develop our specific neural network architecture (which aims to avoid some of the previously mentioned problems of CNNs) and document the training procedure. A high-level schematic drawing of the model architecture is depicted in Figure 3.

A. Model architecture

In order to achieve a model that is agnostic to the length of the input time series, we choose a *fully convolutional* architecture. This means there are no fully connected (or dense) layers. Instead, the neural network only learns convolutional filters (or kernels), which make no assumptions about the size of their input data.

This has two major advantages. First, if the size of the *receptive field* of the network is r, we can directly evaluate our model on a time series of n time steps for any $n > r$, resulting in an output time series of length $n - r + 1$. The receptive field of a network refers to the number of time steps on the input layer that affect a single time step on the output layer. Typically, an architecture should be chosen such that the receptive field is large enough to cover a substantial part of the signal. Secondly, it is more computationally efficient than a sliding window approach, which—due to the overlap of neighboring windows—performs redundant computations. A fully convolutional architecture avoids this overhead.

Moreover, instead of evaluating the network for each detector separately, we stack the recordings from all observatories and treat them as different channels of a single, multi-dimensional input. This means that when the number of detectors changes, we only need to adjust the number of input channels of the first layer, while the rest of the architecture remains fixed. While retraining is required after such an extension, the computational complexity of our approach at test time is virtually constant in the number of detectors.

In practice, we use a stack of 12 *(convolutional) blocks*, each based on a dilated convolutional layer with 512 convolutional kernels of size 2. Empirically, we found that increasing the number of channels used in the convolutional blocks generally improves the overall performance. However, memory limitations during training upper-bounded the number of channels to 512. Within each block, the convolutional layer itself is followed by a non-linear activation function, namely a rectified linear unit (ReLU). We did not use any regularization techniques such as dropout or batch normalization.

The difference between the twelve convolutional blocks is the dilation of the kernels, which increases exponentially in powers of two (i.e., 1, 2, 4, ..., 2048) with the block number. This simple trick yields a relatively large receptive field of 2 seconds with a moderate depth of only 12 blocks while avoiding loss of resolution or coverage. This was considered sufficient to cover the relevant region around the coalescence for all signals of interest. Other modifications of the kernel, such as strides, were not used.

The stack of convolutional blocks is preceded by an *input* convolutional layer with kernel size of 1, which maps the input data from two channels (the strains from H1 and L1) to 512 channels. On the output side of the network, the last convolutional block is succeeded by an *output* convolutional layer, which again has a kernel size of 1 and serves to reduce the number of channels from 512 back to 1. The now one-dimensional network output is then passed through a sigmoid layer [64], which maps it into the interval $(0, 1)$.

Our implementation (in Python 3.6.7) is based on the PYTORCH deep learning framework (version 1.0.1) [65]. All code that was used to obtain the results presented in this work will be made available online at [66].

B. Training procedure

As usual for CNNs, before feeding an example time series x as input during training, validation, and test time, we normalize it via $(x - \mu)/\sigma$, where μ and σ are computed as the medians of the mean and standard deviation of each individual example in the training set. During
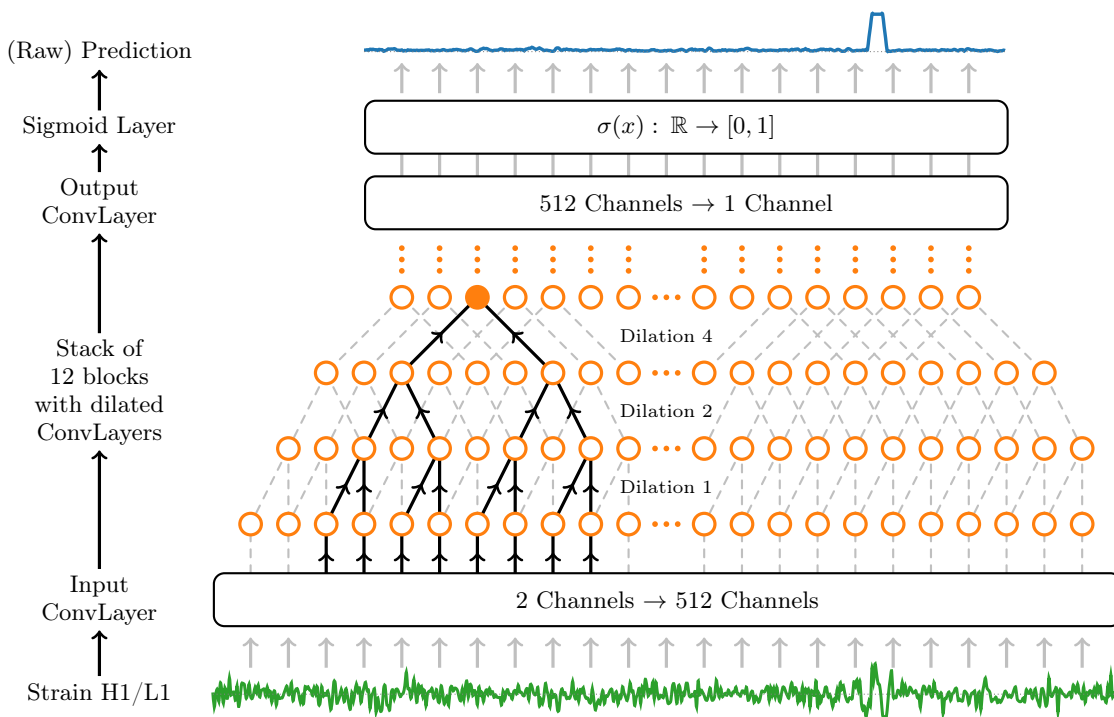
Figure 3. Schematic visualization of the proposed architecture to illustrate the effect of dilated convolutions on the receptive field: the highlighted (solid orange) value in the fourth layer depends on exactly 8 values in the input layer. It therefore has an receptive field of size 8. The figure also shows how the length of the time input is successively reduced with each convolutional layer: the output of layer i is $r_i - 1$ time steps shorter than the original input, where r_i denotes the receptive field of that layer.

training, we monitor the generalization performance by regularly evaluating the model on the validation set. For the actual training, we first use the *Kaiming* initialization scheme as introduced in [67] to assign initial random values to the network parameters (i.e., the convolutional kernels). During training, the kernel entries are optimized using stochastic gradient descent using *Adam* [68] with the *AMSgrad* modification proposed in [69]. To this end, within every *epoch* (i.e., a full pass over all training data) the entire training dataset is randomly shuffled and divided into a fixed number of *mini-batches*. We use *binary cross-entropy* (BCE) as the loss function. The *batch loss* is calculated as the average of the BCE losses at every time step of every example in the mini-batch and its corresponding label value. This batch loss is then automatically differentiated with respect to all kernels, and error back-propagation is used to update the kernel values.

At the end of every epoch, the performance of the network with its current parameter values is evaluated both on the full training and validation data set. The current loss (as well as other metrics, see below) are logged and a *checkpoint* of the model is created. This means that a copy of the model parameters is saved to disk such that the current training state can later be loaded again. We end training after a fixed number of epochs and retrieve the checkpoint corresponding to the lowest validation loss as the final *trained model*. This is a form of *validation-based early stopping*, which helps to avoid overfitting.

By default, we choose an initial learning rate of $\eta_\mathrm{init} = 3 \times 10^{-4}$. During training, the learning rate is reduced whenever the loss on the validation set has not decreased by more than a certain threshold over a given number of epochs (default value: 8). This behavior is controlled by PyTorch's `ReduceLROnPlateau` method.

In practice, we have trained our network for 64 epochs on the full training set using 5 NVIDIA Tesla V100 GPUs, each with 32 GB of memory. In total, training the model took approximately 30 hours on our hardware. This was deemed sufficient, as the network started to show signs of overfitting after approximately 30 epochs. As mentioned above, however, at test time (i.e., for all evaluation experiments) we only used the model checkpoint with the lowest validation loss.

C. Post-processing

Finally, we apply two post-processing steps to the raw network output: *smoothing* and *thresholding*.

To *smooth* the output time series, we apply a rolling average as a convolution with a rectangle function. The window size (i.e., width) of this rectangle function can be tuned depending on the metric we want to optimize (see next section). Smoothing removes short spikes, which otherwise could be confused with the presence of signals. By default, we choose a window size of 256 time steps.

In the subsequent *thresholding* step, the smoothed output is mapped from $(0, 1)$ to $\{0, 1\}$ depending on whether it exceeds a threshold t. This allows for stable and efficient peak-finding (see next section). Again, the choice of the threshold t depends on the metric that one ultimately wants to optimize. By default, we used $t = 0.5$.

Both post-processing steps are only applied at *test time*, and we evaluate the effect of the parameter choices on the final performance in section VII. To compute the loss during training, we only use the raw, non-processed output of the network.

VI. PERFORMANCE METRICS

A. Design and creation of labels

Let us now explain how we generate the *labels*, that is, the desired network output for a given input. In our case, the labels are also time series: Ideally, the network should mark the exact locations of coalescences. A natural way to represent this is a time series which is zero everywhere except at the event time where the signal in H1 reaches its maximum amplitude (where the label takes on a value of 1).

From a practical machine learning point of view, however, this is problematic: such sparse signals do not contribute sufficiently to the average loss to keep the network from simply *always* predicting zero. To prevent this failure mode, instead of labelling a single time point, we choose a fixed-width interval centered around the time when the injected signal in the H1 channel reaches its maximum amplitude. By construction of our data generation pipeline, this position is fixed for all examples. (Recall that our fully convolutional network architecture is by design unable to overfit to specific locations within input examples.) Thus, labels need not be pre-generated or stored, but can be computed on the fly during training or testing. By default, the labels width (i.e., the length of the symmetric interval around the event time in which the label time series takes on a value of 1) is $0.2\,\text{s}$.

B. Evaluation metrics at test time

In section II we discussed the drawbacks of the true positive rate and the false positive rate as performance measures for gravitational-wave searches in the example based binary classification setup. The fact that our data generation pipeline also generates individual examples is merely to make training convenient. Our model could equivalently be trained on a single time series (of sufficient length) containing any number of injections at arbitrary locations. This is possible because our architecture does not perform binary classification on an example level, but outputs yet another time series. As a consequence, different performance metrics are required.

Our objective is to tag signals in the data by outputting a peak *close* to the actual coalescence time. This intuition can be formalized to obtain interpretable performance metrics using the following evaluation procedure:

1. We identify all intervals of value 1 in the smoothed and thresholded network output.

2. The interval centers are stored as *candidate times*.

3. For each candidate time t_c, we test for coincidence with the ground truth injection time t_i, that is, if $|t_c - t_i| \leq \Delta t$. By default, we used $\Delta t = 0.05\,\text{s}$. Note that Δt is another free, tuneable hyperparameter whose value will directly affect the performance metrics defined below.

4. If a candidate time passes this coincidence check, we count it as a *true positive* or *detection* (see note below); otherwise, it is a *false positive*.

5. Per example, there can only be one true positive. If multiple candidate times pass the coincidence test for a single example, only one of them is counted as a detection, while the others are false positives.

> **Note:** We use the term *detection* in this context to refer to an injected signal which was successfully recovered (in the sense of the procedure described above) by the network. This is, however, purely for ease of terminology. A "detection" by the CNN cannot be compared to and must not be confused with the (statistically significant) detection of a gravitational wave as described in section II B. Similarly, the *false positive rate* (see below) cannot directly be compared to a *false alarm rate*.

We can now discuss the network performance on the test set in terms of the *detection ratio* and the *false positive ratio*. The detection ratio is simply the number of injected signals in the test set that the network could recover, divided by the total number of injected signals. We therefore also call it *sensitivity*. The false positive ratio is the number of false positives divided by the number of all produced candidate times. It is thus an estimate of the error probability; the probability that any given candidate time does not coincide with an actual signal.

Additionally, we can also define the *false positive rate*: the total number of false positives divided by the combined duration of all samples in the test set. Its inverse is more intuitive: the *inverse false positive rate* is the average time between two false positives. Naturally, this number should be as high as possible, meaning false positives should be as infrequent as possible.

Again, note that our metrics do not rely on the existence of distinct examples, but could equally be evaluated on a single time series of arbitrary length containing multiple signals. To illustrate this key difference further, let us go back to our argument why the true positive rate and the false positive rate can not be used to evaluate example based binary classification approaches in

the sliding window mode of operation considering the output sequence $1 - 1 - 1 - 0 - 1 - 1 - 0$. First, previous approaches do not explain how to interpret such an undesirable situation. Moreover, their performance metrics are blind to these occurrences, because they are derived only from fixed-length examples, which all have an unambiguous binary label. Taking into account the continuous nature of the task, our metrics acknowledge this issue by counting at least one of the two positive intervals as a false positive if there was only one real signal within the corresponding time interval.

VII. EXPERIMENTS AND RESULTS

A. Performance evaluation

When evaluated on our full test set, our trained model is able to successfully recover approximately 89% of all injections, while on average producing a false positive about once every 19.5 minutes.

For a more fine grained analysis, we then split the positive examples (i.e., the ones that do contain an injection) in the test set into 30 bins based on their respective injection SNR. The bins are distributed equidistantly and cover the full injection SNR range of $(5.0, 5.5), (5.5, 6.0), \ldots, (19.5, 20)$. On average, every bin therefore contains $0.75 \cdot 16\,384/30 \approx 410$ examples. We then compute the detection ratio independently for each of these bins to investigate how the sensitivity of our method scales with the faintness of the signals. The results in Figure 4 show that the detection ratio increases steeply with the injection SNR and achieves essentially 100% roughly at an SNR of 11.

For comparison, the threshold for a coincident trigger to even be analyzed within the PYCBC search pipeline is $\sqrt{5.5^2 + 5.5^2} = 7.79$. At this injection SNR, our model already recovers more than 80% of all injected signals. Furthermore, the first ten real binary black hole mergers observed so far had network SNRs between 9.5 and 30.9 [12], which is well within the region in which our model has a virtually perfect detection ratio.

B. Effects of post-processing

Next, we systematically investigate the effect of both the smoothing and thresholding parameters. To this end, we post-process the raw network output on the test set with different sizes of the smoothing window (1, 2, 4, 8, 16, 32, 64, 128, and 256) and different thresholds (0.1, 0.3, 0.5, 0.7, and 0.9). In the parametric plot in Figure 5, we show the detection ratio and the inverse false positive rate averaged over the entire test set for each combination of parameter settings (meaning up and right are better). While there is no single best option, this plot shows that our two parameters provide clearly interpretable tuning
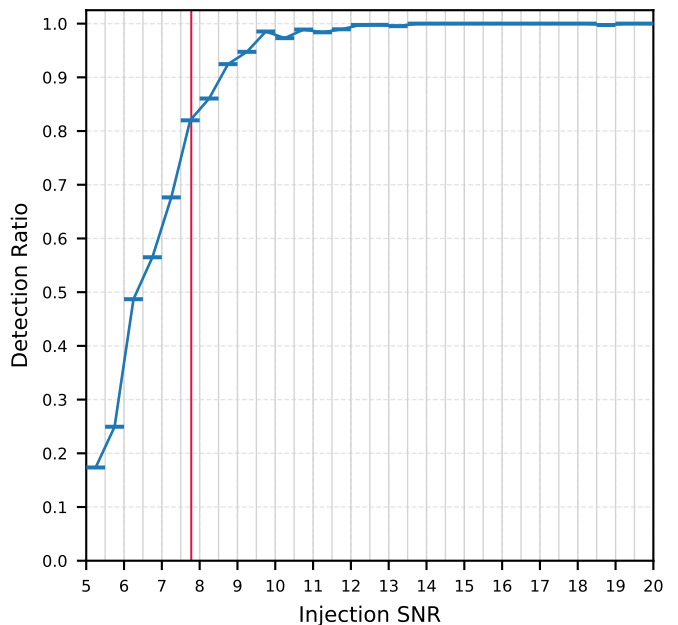

Figure 4. The (non-cumulative) detection ratio (DR) for positive examples binned by their network injection SNR. The DR increases steeply and plateaus at essentially 100% for an SNR $\gtrsim 11$. The red line indicates the network SNR threshold above which PYCBC considers events for further analysis.
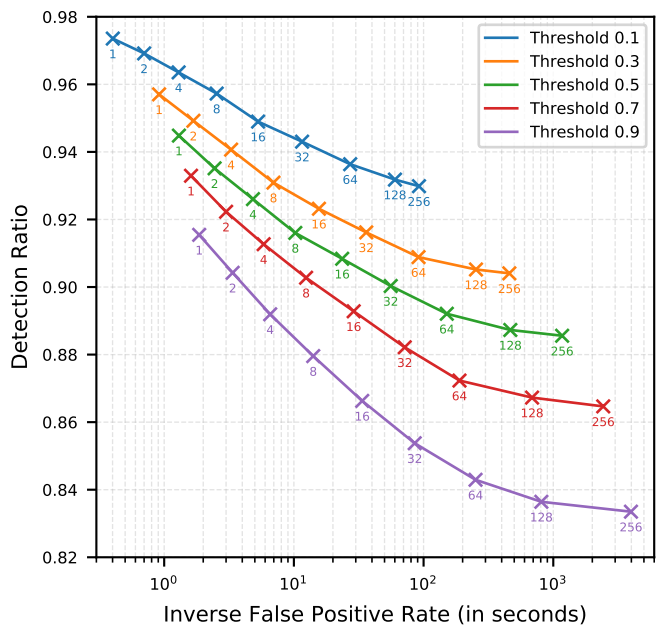

Figure 5. This figure shows the effect of the smoothing and thresholding parameters used during the post-processing step on the detection ratio and the inverse false positive rate. Colors encode the different threshold values, while the number next to the data points indicates the size of the smoothing window. The plot shows that these two parameters provide interpretable tuning knobs to choose an operating point.
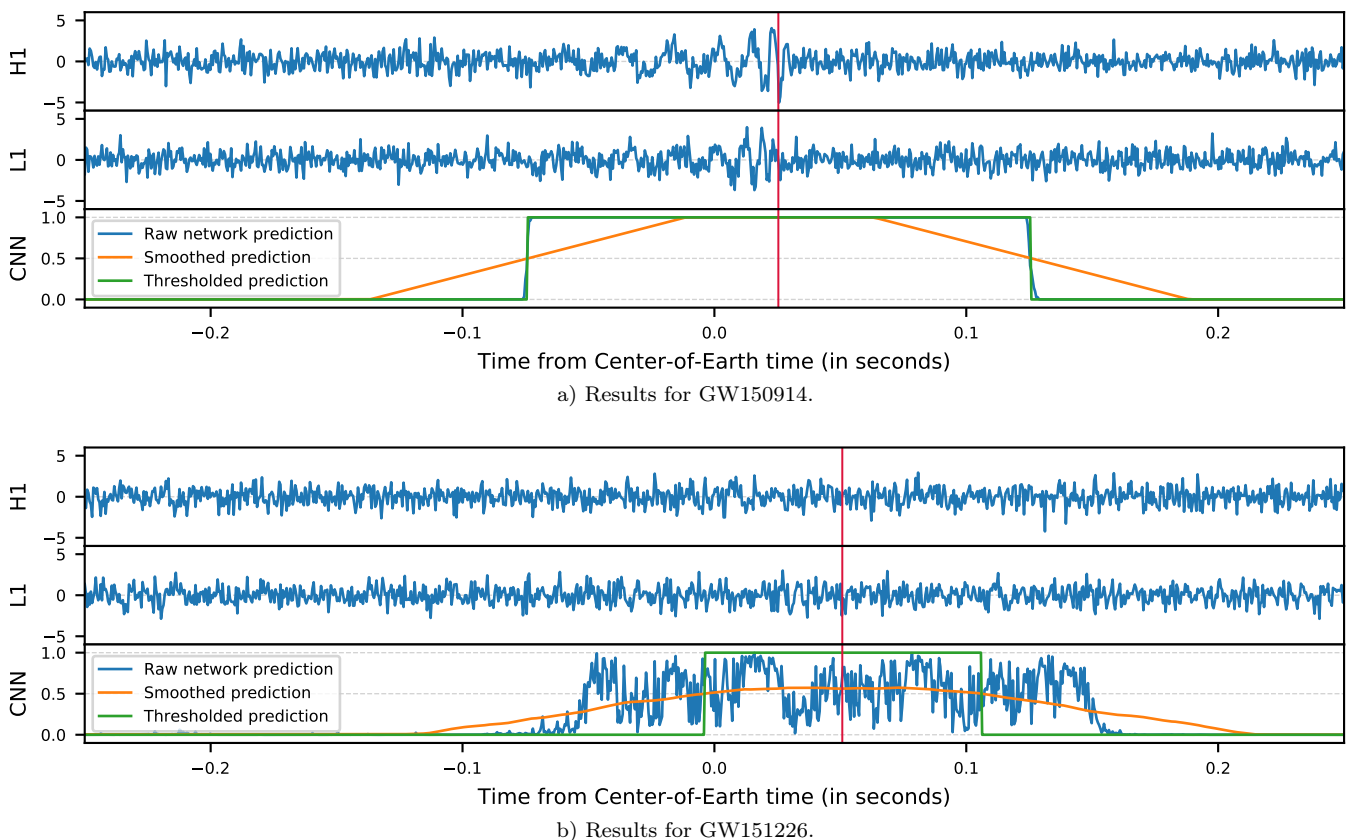
a) Results for GW150914.

b) Results for GW151226.

Figure 6. Results for recovering the first two confirmed real events in O1, GW150914 and GW151226. The top two panels of each plot show the whitened, normalized strain for H1 and L1, centered around the time at which the peak of the gravitational-wave amplitude passed through the center of the Earth. The last panel shows the different post-processing stages, namely, the raw, smoothed and thresholded network output (smoothing window size 256, threshold 0.5). The vertical red line indicates the predicted position of the event, calculated as the center of the interval of ones in the thresholded output.

knobs to choose an operating point by trading off the sensitivity and the false positive rate. Depending on the application requirements one may use this plot to optimize detection ratio at fixed false positive rate or vice versa.

C. Recovering real gravitational-wave events

In the next experiment, we evaluate our model's ability to generalize from synthetic training data to real events. The first two observations announced during LIGO's first observation run were GW150914 and GW151226 [7, 8]. These real signals were not included in the training data. At test time, we select an interval centered around the event times from the original recordings for both events, and apply the established whitening and band-passing procedure. Both samples are then cropped to 16 s, again centered around the event time. After normalizing and passing them through the network, we apply our usual post-processing steps, using a window size of 256 time steps for the smoothing and thresholding the result at 0.5.

The results in Figure 6 show that in both cases, the model was able to successfully recover the real GW signal

at the correct position despite being slightly less accurate on the fainter event GW151226 (with a network SNR of 13) than the first observed event GW150914 (with a network SNR of 24) [7, 8]. The fainter example highlights the effect of post-processing: Instead of causing multiple false positives when thresholding the raw network output directly, the additional smoothing step yields a single connected interval (i.e., a single predicted event time).

Finally, we also apply our trained network to all other events in the GWTC-1 catalog [12], which consists of 11 confirmed binary mergers from both the first and second observation run of LIGO. Using the event data available from the GWOSC (which was pre-processed in the same way as before), we find that our network can indeed recover all known events, with the exception of GW170817. This is, however, not a surprise: While all other events are binary *black hole* mergers, and we also trained our model using simulated BBH waveforms, GW170817 is the only confirmed binary *neutron star* merger [14].

Lastly, the fact that we are able to also successfully recover the events from O2 after using only recordings from O1 to train also indicates that the model is, to a certain extent, robust to changes in the detector characteristics.

D. A note of caution

In a final experiment, we once more want to emphasize our call for caution when interpreting CNNs in the context of gravitational-wave searches. To address the question "What has the model actually learned?", we use techniques inspired from *activation maximization* or *feature visualization* (see, e.g., [70, 71]), as well as *adversarial examples* or *adversarial attacks* (see, e.g., [72]), which are currently active areas of research within the machine learning community. Specifically, we perform the following test in which we make use of the differentiability of our model to find examples of inputs which cause the network to produce a given target output:

1. We randomly select a noise-only example (i.e., an example that does not contain an injection) from our testing set and crop it from the end to a length of 3 s. This is our initial network input.

2. Next, we generate a target label, which is about 1 s long (3 s minus the receptive field of the model) and zero everywhere except for the interval from 0.45 s to 0.55 s, where it takes on a value of 1.

3. If applicable, we enforce additional constraints on the inputs. For example, we pass the input through a $\max(x, 0)$-function to create the physically nonsensical scenario of a strain that is strictly non-negative (see first example in Figure 7 c).

4. We pass the constrained network input through the trained model from the previous experiments. We then compute a weighted sum of a binary cross-entropy and a mean squared error loss between the network prediction and the target. The exact weighting depends on the optimization target.

5. Unlike when training a neural network, this loss is then not back-propagated to the weights of the network, which stay fixed during this experiment. Instead, the loss is back-propagated to the *input*, which is updated in order to minimize the loss.

6. We repeat this procedure (starting with enforcing possible constraints on the inputs) for 256 iterations, again using *Adam* as the optimizer, with an initial learning rate of $\eta = 0.3$. PyTorch's default cosine annealing scheduler is used to gradually decrease the learning rate every epoch.

7. Finally, we compute the difference between the original network input and the optimized input. This can be interpreted as the hypothetical "signal", which—when added into the pure noise example—makes our network produce the target output.

We repeat this procedure for different initial inputs and manually inspect the results in form of the hypothetical "signals" to check if they match our expectation:

If the network had truly learned to respond only to gravitational waves, we would expect these hypothetical "signals" to closely resemble gravitational-wave signals.
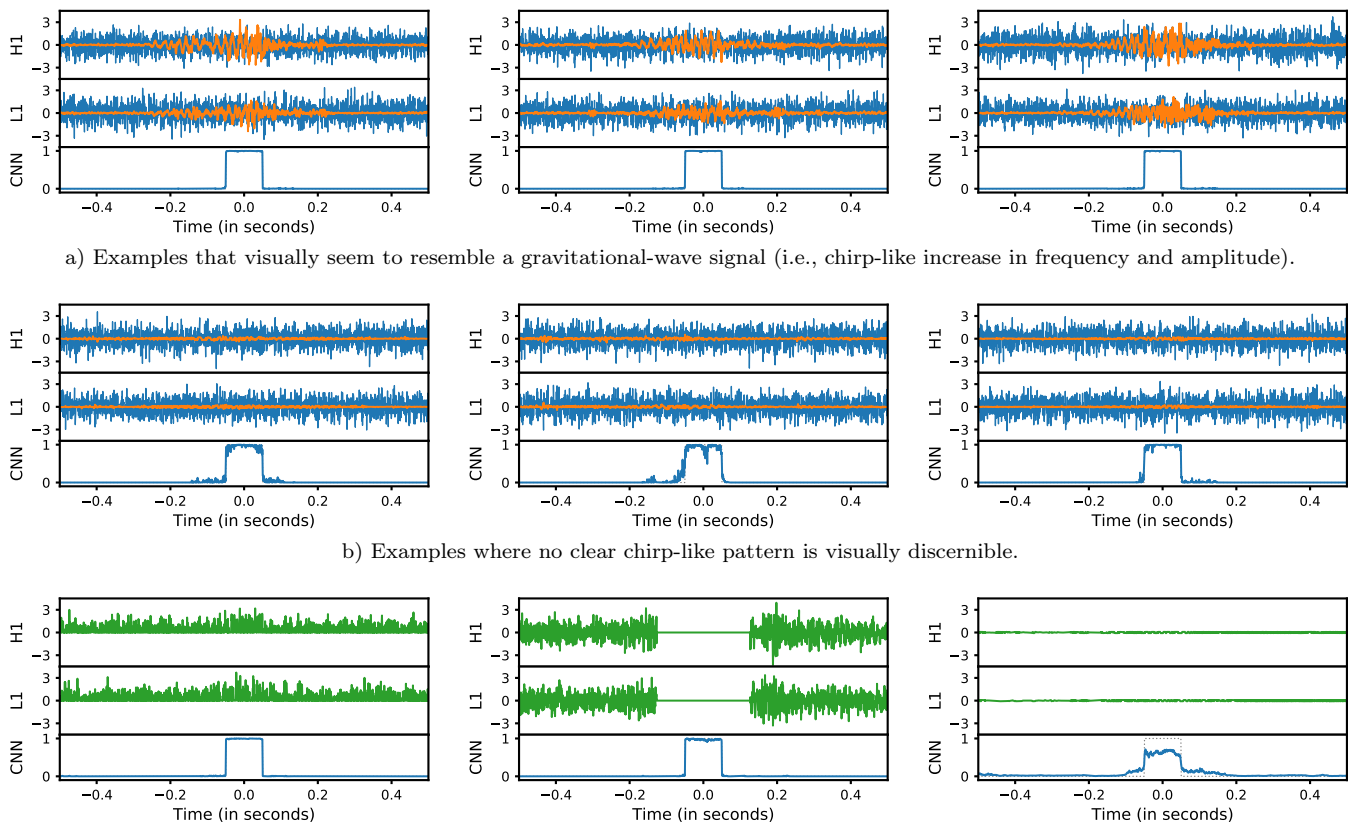
However, while some of the inputs that have undergone the described optimization procedure do exhibit a chirp-like structure (i.e., oscillations increasing in both amplitude and frequency), we find that this is not always the case; see panel a) and b) of Figure 7. Worse yet, we can also achieve the desired output even when imposing non-physical constraints on the inputs. We investigate three types of such constraints: Firstly, we allow only non-negative strain values. Secondly, we enforce the strain to be zero in a 0.25 s-interval covering the interval in which the target output is one. Thirdly, we clip the network input values to a small interval around zero to minimize the overall amplitude. In all three cases, we can still find examples that obey the constraints and, when passed through the network, yield the desired target output. Examples for this are shown in panel c) of Figure 7.

Since we crafted these examples in a supervised fashion, one may argue that the cases in panel c) are *unrealistically* out of distribution, that is, they would never occur in real detector recordings and therefore do not lead to complications in practice. However, in particular the unconstrained examples in panel b) of Figure 7 are unsettling, because they illustrate just how easily the network can be fooled even by small changes in the inputs. These results suggest a detailed quantification of how contrived these hypothetical "signals" really are (measured by how likely they are to occur accidentally in future detector recordings) to assess whether one must account for them in the false positive rate. Without such an analysis the worry of overconfident positive CNN output on pure noise or faint non-Gaussian transients remains.

VIII. DISCUSSION AND CONCLUSION

In this work we provide an interdisciplinary, in-depth analysis of the potential of deep convolutional neural networks (CNNs) as part of the effort around searching for gravitational waves from binary coalescences in strain data. First, we critically scrutinize both the methods as well as the contributions of existing works on this topic by carefully analyzing how standard machine learning approaches and metrics map to the specific task at hand. This analysis yields two major conclusions: 1. CNNs alone can not be used to claim statistically significant gravitational-wave detections. 2. Fast inference times, favorable computational scaling in the number of detectors, and a compact internal representation of a large number of waveforms presented during training still make CNNs a useful and promising tool to produce real-time triggers for detailed analysis and follow up searches.

As part of these key conceptual insights, we hope to foster further interdisciplinary research on this topic by highlighting important subtleties of GW searches to ma-

a) Examples that visually seem to resemble a gravitational-wave signal (i.e., chirp-like increase in frequency and amplitude).

b) Examples where no clear chirp-like pattern is visually discernible.

c) Examples which satisfy unphysical constraints, yet still cause the network to predict the presence of a signal. In the first example, the input strain is constrained to only non-negative values. In the second example, the input strain is constrained to 0 in the 0.25 s-interval around predicted event time. In the last example, the entire example is constrained to have a minimal strain amplitude.

Figure 7. This figures shows different example results where we—using a fixed pre-trained model—optimized the network inputs (starting from noise-only examples) in order to produce a given desired output. The top and middle panel show the strain for the two detectors, H1 and L1. The original inputs (i.e., the pure background noise) are shown in blue, and the *difference* between the original and the optimized input is shown in orange. This is the component that is added to the noise in order to make the network predict the presence of a "signal". Ideally, we would therefore expect the orange component to look like a gravitational-wave waveform. For the examples in subfigure c), only the the effective (i.e., optimized and constrained) inputs to the network are shown (in green). The bottom panel of every figure shows the desired output (i.e., the optimization target) in dotted gray, and the raw network prediction in blue (i.e., without any post-processing).

chine learning experts and exposing some potential pitfalls and surprising properties of CNNs to physicists.

Building on these insights, we have designed a flexible data generation pipeline which we make publicly available as an open source package. We use a novel network architecture which is more tailored to the physical task at hand than a binary classification-based approach and also overcomes some subtle pitfalls, such as the danger of overfitting to some particular properties of the training data. We evaluate this approach on real LIGO recordings and demonstrate the potential of such a system as a trigger generator by achieving a detection ratio of 86% with a false positive on average once every 40 minutes. Two tuneable post-processing parameters allow us to intuitively trade off between the detection ratio and the false positive rate without having to retrain the model.

Finally, as part of our effort for cross-disciplinary understanding, we showcase a selection of "failure modes" of our model which are typical for deep convolutional neural networks. We contrive inputs which the network believes to contain gravitational-wave signals with high confidence, even though they are structurally very different from real detector signals for compact binary coalescences. While some of these inputs are physically unrealistic and thus unlikely to be observed in practice, others appear quite plausible (e.g., tiny modifications of pure noise examples). Because the detector noise properties change on an hourly timescale, the rate of false triggers due to such failures may be hard to predict even for a well-tuned CNN. We leave the required quantitative analysis of how such incidences may affect the performance on real-world recordings under changing detector characteristics for future research, and conclude this work with a note of caution: CNNs are a promising tool for gravitational-wave data analysis; however, their exact interpretation requires great care and attention.

ACKNOWLEDGMENTS

Appendix A: Data generation parameters

The following list explains the different parameters and the distributions from which their values are randomly sampled before being passed as inputs to the `SEOBNRv4` waveform model in order to simulate synthetic gravitational-wave signals. Because the true astrophysical distributions for compact binary coalescences are unknown, we choose the following generic values:

- `mass1` and `mass2`: The masses of the two merging black holes, chosen independently and uniformly at random between 10 and 80 solar masses.

- `spin1z` and `spin2z`: The z-component of the spin of the merging black holes, chosen independently and uniformly at random between 0 and 0.998 (to improve the numerical stability).

- `ra` and `dec`: The right ascension of declination defining the position of the source in the sky. Both values are sampled together from a uniform distribution over the sky.

- `polarization`: The polarization angle is one of the three Euler angles relating the radiation frame, which is the reference frame in which the gravitational wave propagates in the z-direction, to the reference frame of the detector. It is sampled uniformly at random from the interval $[0, 2\pi]$.

- `coa_phase` and `inclination`: To understand the significance of the *coalescence phase* and the *inclination*, one needs to introduce a third reference frame beside the detector and radiation frame, namely, the reference frame of the source itself. In the case of a binary coalescence, this source reference frame is chosen such that its z-axis is perpendicular to the plane in which the two black holes orbit each other. Then, the `coa_phase` and the `inclination` are the two angles that specify the location in the sky of the detector as seen from this source frame. Their values are sampled jointly from a uniform distribution over a sphere.

- `injection_snr`: For evaluation purposes, it is useful to generate samples with a pre-defined signal-to-noise ratio. This can be achieved by rescaling the waveform, which is physically equivalent to moving the source closer or further from the detector. The `injection_snr` is the desired network SNR for the example, which is sampled uniformly from [5, 20]. It is not directly passed to the simulation routine, but only used later when adding the simulated signal into the background noise.

[1] B. Allen *et al.*, FINDCHIRP: An Algorithm for Detection of Gravitational Waves from Inspiraling Compact Binaries, Physical Review D **85** (2012).

[2] S. Babak *et al.*, Searching for gravitational waves from binary coalescence, Physical Review D **87** (2013).

[3] S. A. Usman *et al.*, The PyCBC search for gravitational waves from compact binary coalescence, Classical and Quantum Gravity **33** (2016).

[4] C. Messick *et al.*, Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data, Physical Review D **95** (2017).

[5] J. Aasi *et al.*, Advanced LIGO, Classical and Quantum Gravity **32** (2015).

[6] F. Acernese *et al.*, Advanced Virgo: a second-generation interferometric gravitational wave detector, Classical and Quantum Gravity **32** (2014).

[7] B. P. Abbott *et al.*, Observation of Gravitational Waves from a Binary Black Hole Merger, Physical Review Letters **116** (2016).

[8] B. P. Abbott *et al.*, GW151226: Observation of Gravitational Waves from a 22-Solar-Mass Binary Black Hole Coalescence, Physical Review Letters **116** (2016).

[9] B. P. Abbott *et al.*, GW170104: Observation of a 50-Solar-Mass Binary Black Hole Coalescence at Redshift 0.2, Physical Review Letters **118** (2017).

[10] B. P. Abbott *et al.*, GW170814: A Three-Detector Observation of Gravitational Waves from a Binary Black Hole Coalescence, Physical Review Letters **119** (2017).

[11] B. P. Abbott *et al.*, GW170608: Observation of a 19 Solar-mass Binary Black Hole Coalescence, The Astrophysical Journal **851** (2017).

[12] B. P. Abbott *et al.*, GWTC-1: A Gravitational-Wave Transient Catalog of Compact Binary Mergers Observed by LIGO and Virgo during the First and Second Observing Runs (2018), arXiv:1811.12907.

[13] B. P. Abbott *et al.*, Binary Black Hole Population Properties Inferred from the First and Second Observing Runs of Advanced LIGO and Advanced Virgo (2018), arXiv:1811.12940v3.

[14] B. P. Abbott *et al.*, GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral, Physical Review Letters **119** (2017).

[15] B. P. Abbott *et al.*, Multi-messenger Observations of a Binary Neutron Star Merger, The Astrophysical Journal **848** (2017).

[16] B. P. Abbott *et al.*, A gravitational-wave standard siren measurement of the Hubble constant, Nature **551** (2017).

[17] B. P. Abbott *et al.*, GW170817: Measurements of Neutron Star Radii and Equation of State, Physical Review Letters **121** (2018).

[18] Y. Aso *et al.*, Interferometer design of the KAGRA gravitational wave detector, Physical Review D **88** (2013).

[19] B. P. Abbott *et al.*, Prospects for observing and localizing gravitational-wave transients with Advanced LIGO, Advanced Virgo and KAGRA, Living Reviews in Relativity **21** (2018).

[20] J. Veitch *et al.*, Parameter estimation for compact binaries with ground-based gravitational-wave observations using the LALInference software library, Physical Review D **91** (2015).

[21] L. P. Singer and L. R. Price, Rapid Bayesian position reconstruction for gravitational-wave transients, Physical Review D **93** (2016).

[22] Y. LeCun *et al.*, Backpropagation applied to handwritten zip code recognition, Neural Computation **1** (1989).

[23] Y. Lecun *et al.*, Gradient-based learning applied to document recognition, Proceedings of the IEEE **86** (1998).

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems (NeurIPS)* (2012).

[25] Y. Kim, Convolutional neural networks for sentence classification (2014), arXiv:1408.5882.

[26] A. van den Oord *et al.*, WaveNet: A Generative Model for Raw Audio (2016), arXiv:1609.03499.

[27] W. Zhu *et al.*, Searching for pulsars using image pattern recognition, The Astrophysical Journal **781**, 117 (2014).

[28] G. Carleo *et al.*, Machine learning and the physical sciences (2019), arXiv:1903.10563.

[29] D. George and E. A. Huerta, Deep Neural Networks to Enable Real-Time Multimessenger Astrophysics (2016), arXiv:1701.00008v1.

[30] H. Gabbard *et al.*, Matching matched filtering with deep networks for gravitational-wave astronomy, Physical Review Letters **120** (2018).

[31] A. H. Nitz *et al.*, PyCBC Release v1.13.5 (2019).

[32] B. F. Schutz, Gravitational wave astronomy, Classical and Quantum Gravity **16**, A131 (1999).

[33] S. Caudill, Techniques for gravitational-wave detection of compact binary coalescence, in *26th European Signal Processing Conference (EUSIPCO)* (2018).

[34] B. P. Abbott *et al.*, Characterization of transient noise in Advanced LIGO relevant to gravitational wave signal GW150914, Classical and Quantum Gravity **33** (2016).

[35] M. Cabero *et al.*, Blip glitches in Advanced LIGO data (2019), arXiv:1901.05093.

[36] LIGO Scientific Collaboration, LIGO Algorithm Library – LALSuite, Free Software (GPL) (2018).

[37] C. Capano *et al.*, Implementing a search for gravitational waves from binary black holes with nonprecessing spin, Physical Review D **93** (2016).

[38] B. Allen, χ^2 time-frequency discriminator for gravitational wave detection, Physical Review D **71** (2005).

[39] A. H. Nitz *et al.*, Detecting Binary Compact-object Mergers with Gravitational Waves: Understanding and Improving the Sensitivity of the PyCBC Search, The Astrophysical Journal **849**, 118 (2017).

[40] A. H. Nitz, Distinguishing short duration noise transients in LIGO data to improve the PyCBC search for gravitational waves from high mass binary black hole mergers, Classical and Quantum Gravity **35** (2018).

[41] Y. LeCun and Y. Bengio, Convolutional networks for images, speech, and time-series, in *The Handbook of Brain Theory and Neural Networks* (The MIT Press, 1995).

[42] D. George and E. A. Huerta, Deep neural networks to enable real-time multimessenger astrophysics, Physical Review D **97** (2018).

[43] D. George and E. A. Huerta, Deep Learning for real-time gravitational wave detection and parameter estimation: Results with Advanced LIGO data, Physics Letters B **778** (2018).

[44] X. Li, W. Yu, and X. Fan, A method of detecting gravitational wave based on time-frequency analysis and convolutional neural networks (2017), arXiv:1712.00356.

[45] M. Zevin *et al.*, Gravity Spy: integrating advanced LIGO detector characterization, machine learning, and citizen science, Classical and Quantum Gravity **34** (2017).

[46] S. Bahaadini *et al.*, Deep multi-view models for glitch classification, in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017).

[47] M. Razzano and E. Cuoco, Image-based deep learning for classification of noise transients in gravitational wave detectors, Classical and Quantum Gravity **35** (2018).

[48] S. Bahaadini *et al.*, Machine learning for Gravity Spy: Glitch classification and dataset, Information Sciences **444** (2018).

[49] S. B. Coughlin *et al.*, Classifying the unknown: discovering novel gravitational-wave detector glitches using similarity learning (2019), arXiv:1903.04058.

[50] H. Shen, E. A. Huerta, and Z. Zhao, Deep Learning at Scale for Gravitational Wave Parameter Estimation of Binary Black Hole Mergers (2019), arXiv:1903.01998.

[51] A. H. Nitz *et al.*, Rapid detection of gravitational waves from compact binary mergers with PyCBC Live, Physical Review D **98** (2018).

[52] https://github.com/timothygebhard/ggwd.

[53] B. P. Abbott *et al.*, GW150914: The Advanced LIGO Detectors in the Era of First Discoveries, Physical Review Letters **116** (2016).

[54] D. V. Martynov *et al.*, Sensitivity of the advanced LIGO detectors at the beginning of gravitational wave astronomy, Physical Review D **93** (2016).

[55] B. P. Abbott *et al.*, Calibration of the Advanced LIGO detectors for the discovery of the binary black-hole merger GW150914, Physical Review D **95** (2017).

[56] M. Vallisneri *et al.*, The LIGO Open Science Center, Journal of Physics: Conference Series **610** (2015).

[57] LIGO Scientific Collaboration, O1 Data Release (2018).

[58] C. Biwer *et al.*, Validating gravitational-wave detections: The Advanced LIGO hardware injection system, Physical Review D **95** (2017).

[59] A. Bohé *et al.*, Improved effective-one-body model of spinning, nonprecessing binary black holes for the era of gravitational-wave astrophysics with advanced detectors, Physical Review D **95** (2017).

[60] Y. Bengio *et al.*, Curriculum Learning, in *International Conference on Machine Learning (ICML)* (2009).

[61] For further details, see, for example, section 9.5.3 (b) in [73], section 9.2.3 in [74], or section 6.1.11 in [75].

[62] C. Cutler and É. E. Flanagan, Gravitational waves from merging compact binaries: How accurately can one extract the binary's parameters from the inspiral waveform?, Physical Review D **49** (1994).

[63] R. J. E. Smith, I. Mandel, and A. Vecchio, Studies of waveform requirements for intermediate mass-ratio coalescence searches with advanced gravitational-wave detectors, Physical Review D **88** (2013).

[64] The sigmoid activation function $\sigma : \mathbb{R} \to (0, 1)$ is defined as $\sigma(x) := 1/(1 + e^{-x})$.

[65] A. Paszke *et al.*, Automatic differentiation in PyTorch, Accepted at the "Autodiff Workshop" at NeurIPS (2017).

[66] https://github.com/timothygebhard/magic-bullet.

[67] K. He *et al.*, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification (2015), arXiv:1502.01852.

[68] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2014), arXiv:1412.6980.

[69] S. J. Reddi, S. Kale, and S. Kumar, On the convergence of Adam and beyond, in *International Conference on Learning Representations (ICLR)* (2018).

[70] M. D. Zeiler and R. Fergus, Visualizing and understanding convolutional networks, in *European Conference on Computer Vision (ECCV)* (2014) pp. 818–833.

[71] C. Olah, A. Mordvintsev, and L. Schubert, Feature visualization, Distill **2** (2017).

[72] C. Szegedy *et al.*, Intriguing properties of neural networks (2013), arXiv:1312.6199.

[73] K. S. Thorne, Gravitational radiation, in *Three hundred years of gravitation*, edited by S. W. Hawking and W. Israel (Cambridge University Press, Cambridge, 1987) pp. 330–458.

[74] M. Maggiore, *Gravitational Waves: Volume 1: Theory and Experiments* (Oxford University Press, New York, 2008).

[75] J. D. E. Creighton and W. G. Anderson, *Gravitational-Wave Physics and Astronomy: An Introduction to Theory, Experiment and Data Analysis* (Wiley-VCH, Weinheim, 2011).