# Neural networks for gravitational-wave trigger selection in single-detector periods

A. Trovato* , M. Bejger and E. Chassande-Mottin,
in collaboration with N. Courty, R. Flamary, H. Marchand
*APC, CNRS/IN2P3, Université de Paris

APC, CNRS, FRANCE

Université de Paris

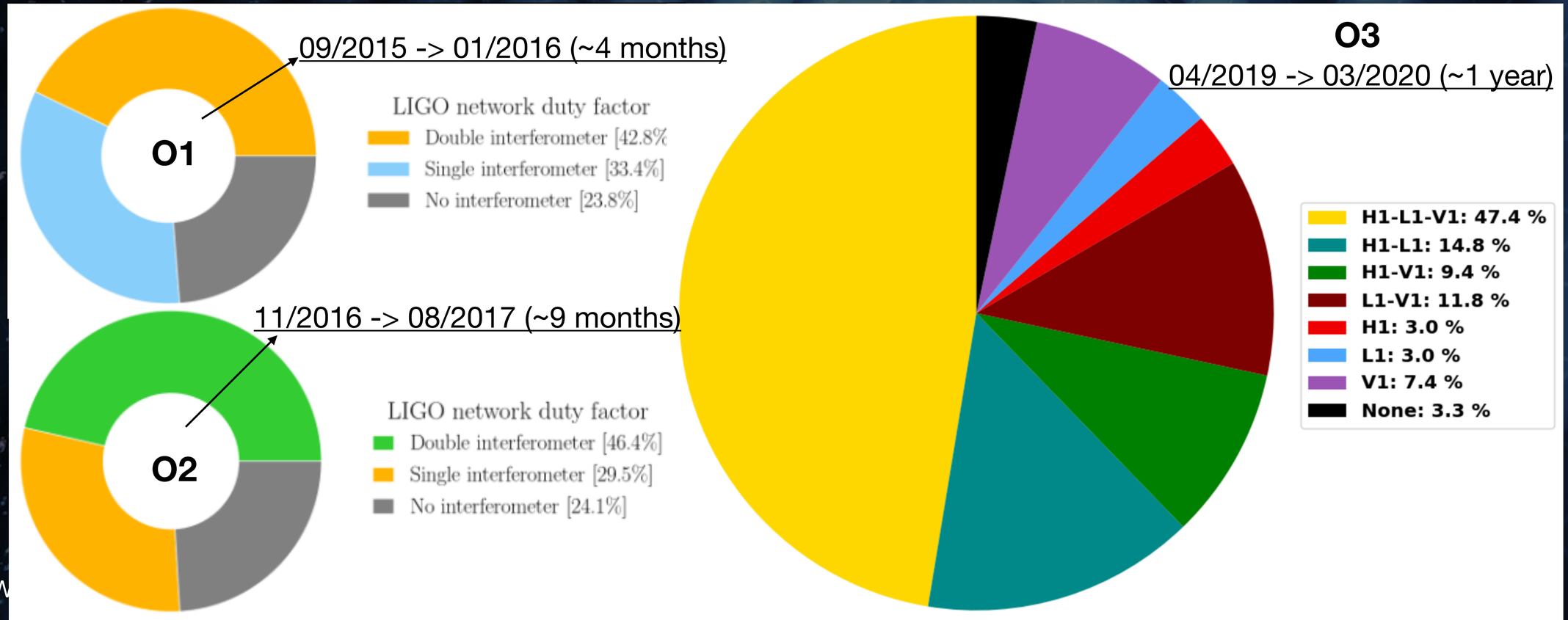
# ML used for GW signal detection

- **Data representation**
  - ✓ Spectrogram vs Time series
- **Pioneering works** (e.g. George et al.[1] or Gabbard et al.[2])
  - ✓ NN are capable to detect BBH (FAP ~ 1e-3 on a single-detector)
  - ✓ To be competitive with match filtering a low FAR is needed
- **Recent work** (Schäfer et al.[3])
  - ✓ Explored different training strategies and solution for softmax
  - ✓ FAR ~ 1/month on gaussian noise (FAP ~ 1e-7)
- **This work:**
  - ✓ time-series representation, real noise from single detector, trigger pre-selection

[1] Phys. Rev. D 97, 044039 (2018)
[2] Phys. Rev. Lett. 120, 141103 (2018)
[3] arXiv:2106.03741

# Single-detector time

- Glitch impact on sensitivity is larger during single-detector periods as coincidence with additional detector is impossible. Can machine learning help?
- Single-detector time:
  - ✓ 2.7 months in O1+O2; 1.6 month in O3



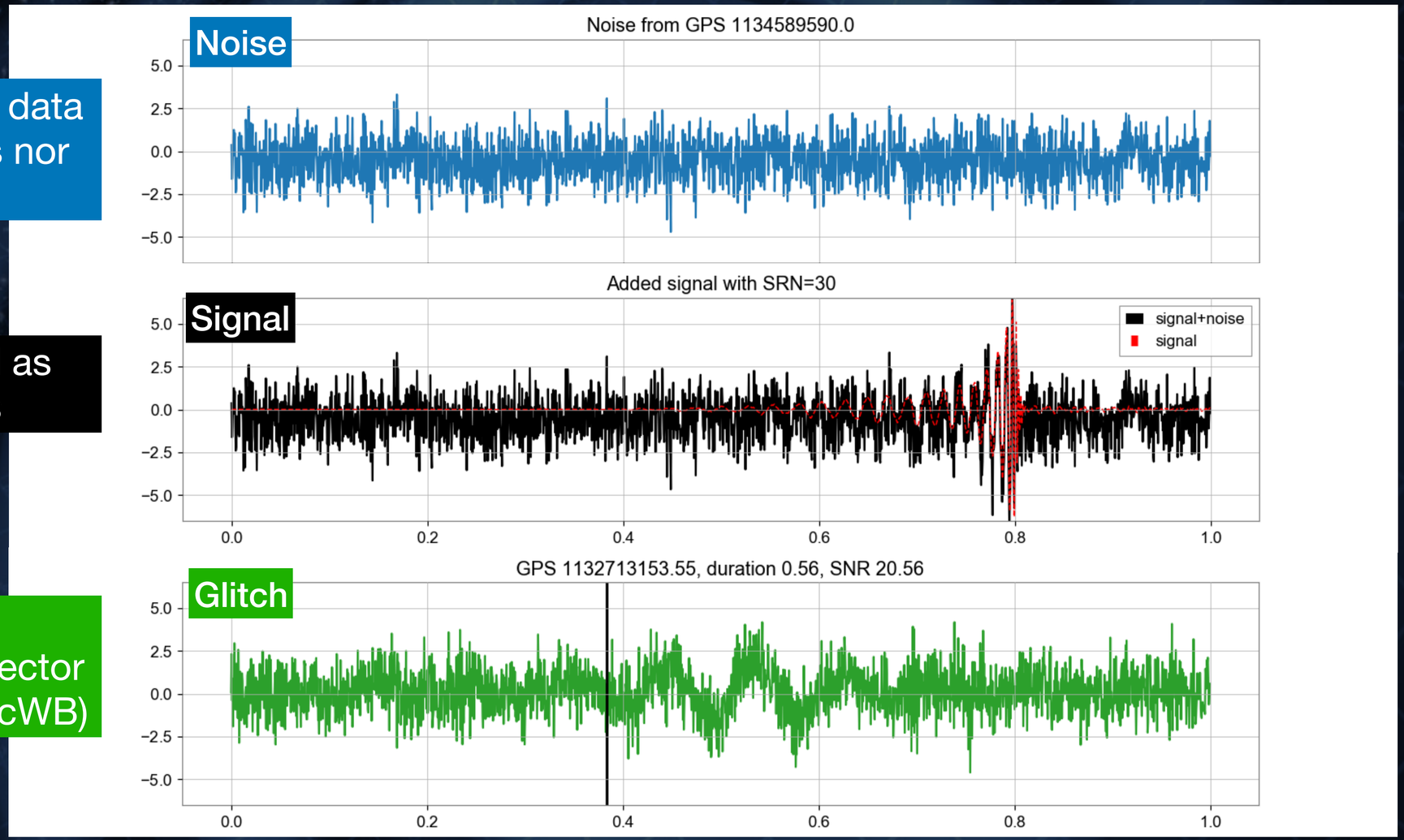
# Training data: 3 classes

Segments of glitches and “clean” noise data samples from the one month of LIGO O1 run (downsampled to 2048 Hz), whitened by the amplitude spectral density of the noise and bandpassed in [20,1000] Hz.

Real detector noise from real data when nor glitches nor signals nor injections are present

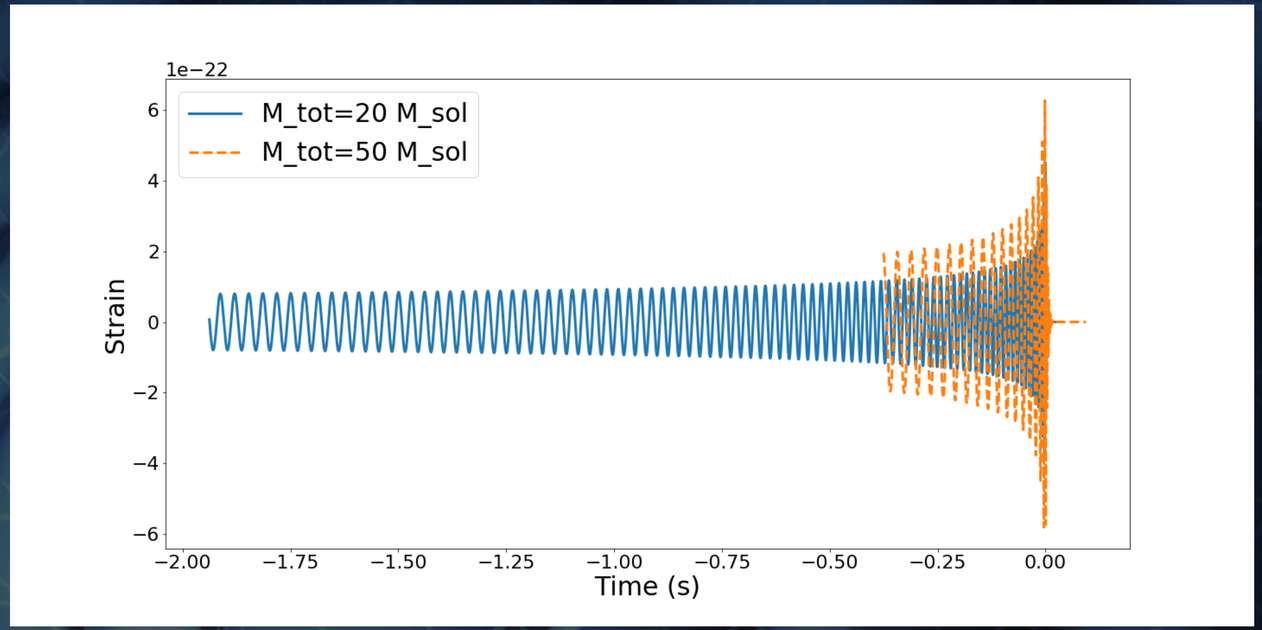
Real detector noise (selected as noise class) + BBH injections

Data containing glitches (glitches inferred from 2+ detector periods with gravity spy and cWB)



# Details on the dataset

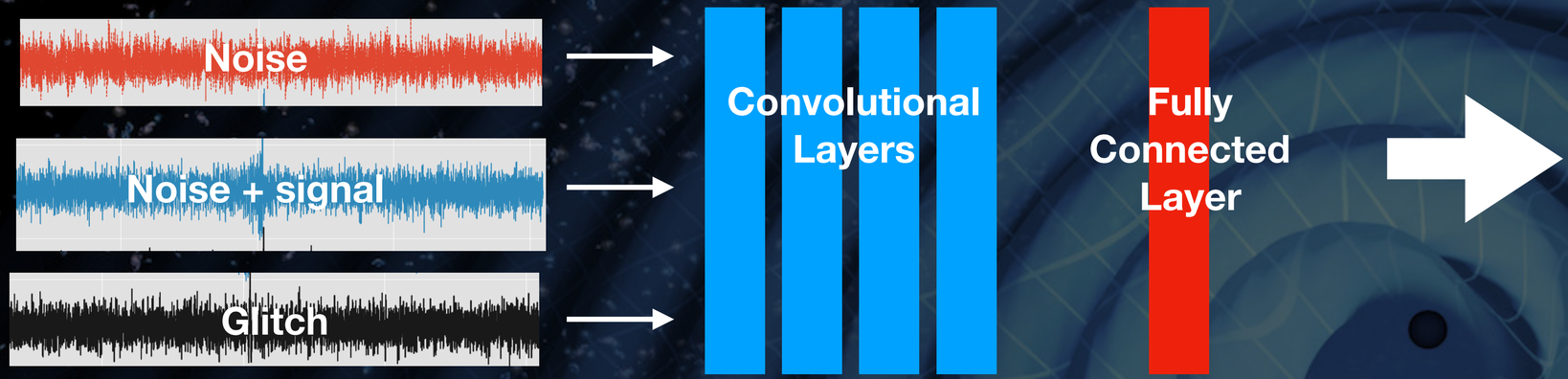
- **Noise class:**
  - **Non overlapping segments of 1 second**
  - Whitened and bandpass filtered in [20,1000] Hz
- **Glitch class:**
  - Glitch **position random** in the 1 s segment if short duration (fully contained) or tailing over multiple segments if duration > 1 s
  - Whitened and bandpass filtered in [20,1000] Hz
- **Signal class:**
  - GW signal injected in a segment of the noise class
  - Random position in the segment while ensuring containment
  - Type pf signal: non-spinning BBH
    - m1+m2 ∈ (33,60) M⊙
    - SNR ∈ (8,20)




- Training:
  - Noise: 250 k - Glitch: 70 k - Signal: 250 k
- Testing:
  - Noise: 520 k - Glitch: 80 k - Signal: 500 k

# CNN used as starting point

- CNN used: small network with 4 convolution layers (with dropouts and pooling) used as classifier to distinguish the 3 classes: noise, noise+signal, glitches




**Output:** probability of belonging to each class

| Layer # | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Type | Conv | Conv | Conv | Conv | Dense |
| Filters | 64 | 32 | 16 | 8 | - |
| Kernel | 16 | 8 | 8 | 4 | - |
| Strides | 4 | 2 | 2 | 1 | - |
| Activation | relu | relu | relu | relu | softmax |
| Dropout | 0.5 | 0.5 | 0.25 | 0.25 | - |
| Max Pool | 4 | 2 | 2 | 2 | - |

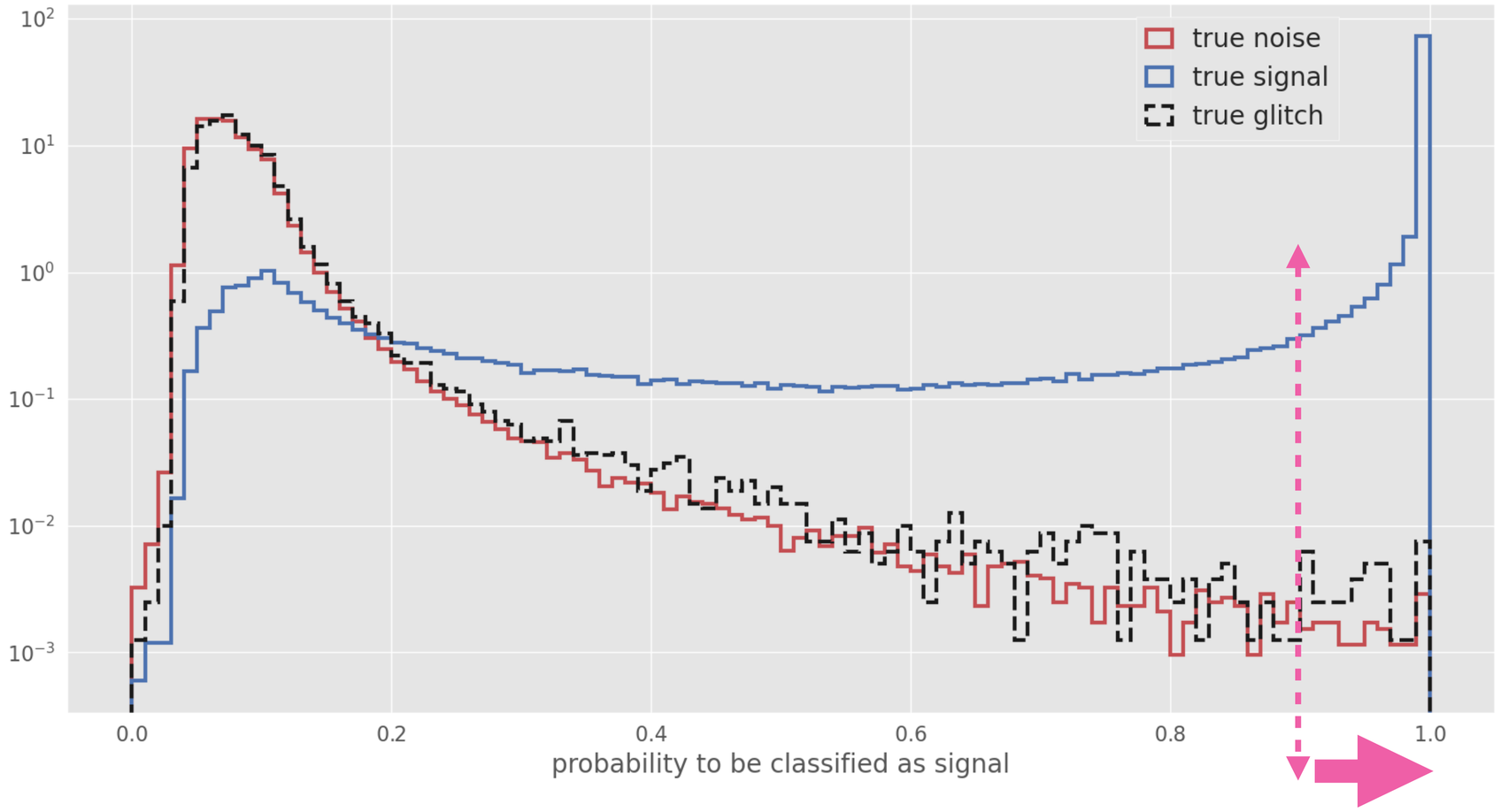
**Optimiser:** Adam (except otherwise indicated)

# Overview on the following results

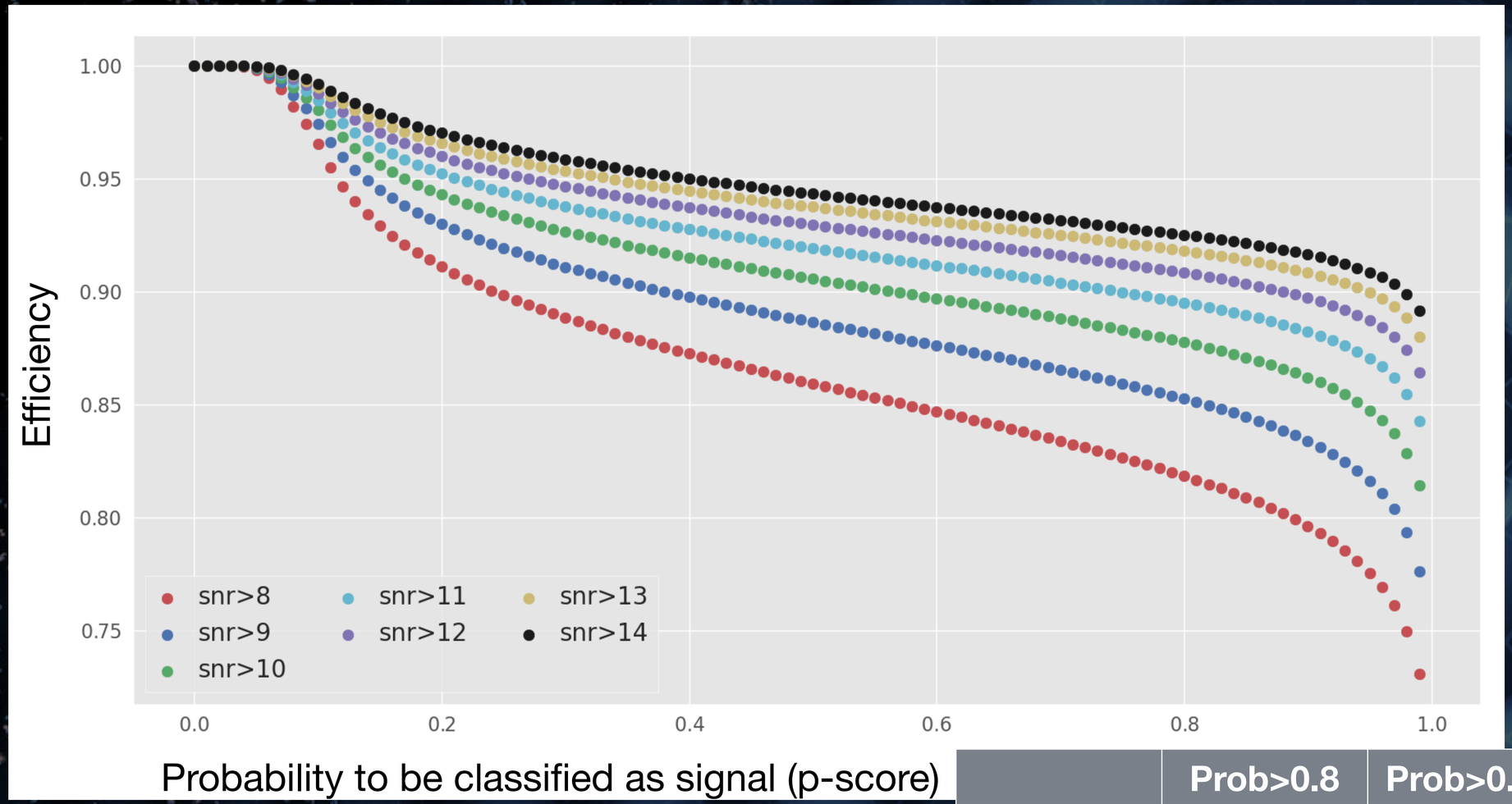
- Evaluate performances of the reference CNN
  - ✓ For each class the network gives as output the probability of belonging to each class
  - ✓ For all samples we use the probability of belonging to the signal class as p-score
- Explore hyper-parameters tuning (number of filters, kernel sizes, optimizer)
- Test network architecture designed for time series classification (TCN)
- Spoiler alert: softmax activation is used in all the results and we noticed the same problem discussed in Schäfer et al.

# P-score

**Use the probability of the signal classification as statistic to distinguish signal vs noise+glitches**

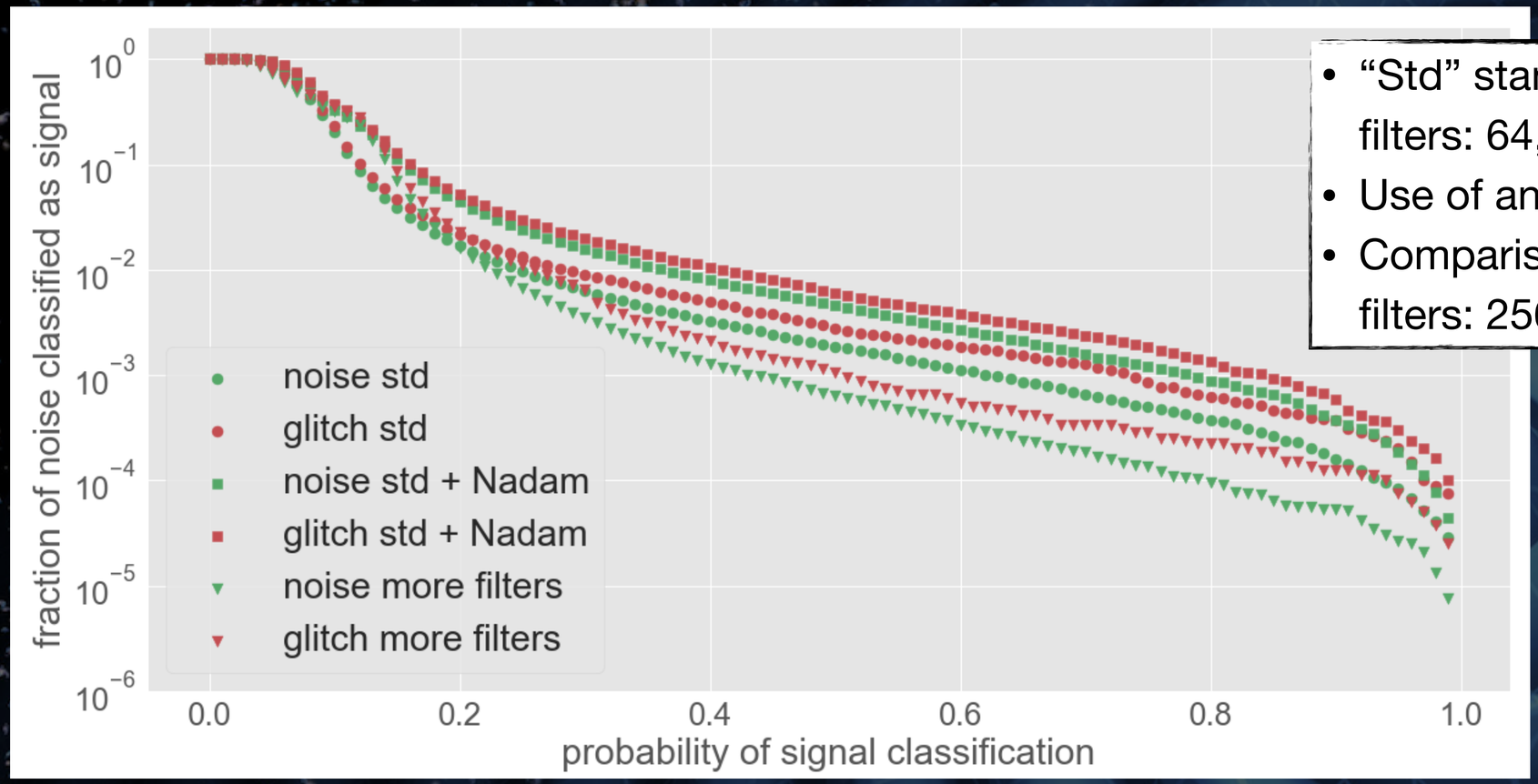


# Efficiency vs p-score




| | Prob>0.8 | Prob>0.85 | Prob>0.9 | Prob>0.95 |
|---|---|---|---|---|
| **SNR>8** | 85% | 84% | 82% | 79% |
| **SNR>10** | 90% | 89% | 88% | 86% |
| **SNR>14** | 94% | 94% | 93% | 92% |

# FAP vs p-score




- “Std” standard architecture used for reference filters: 64,32,16,8
- Use of another optimiser Nadam
- Comparison with a similar architecture with more filters: 256,128, 64,64

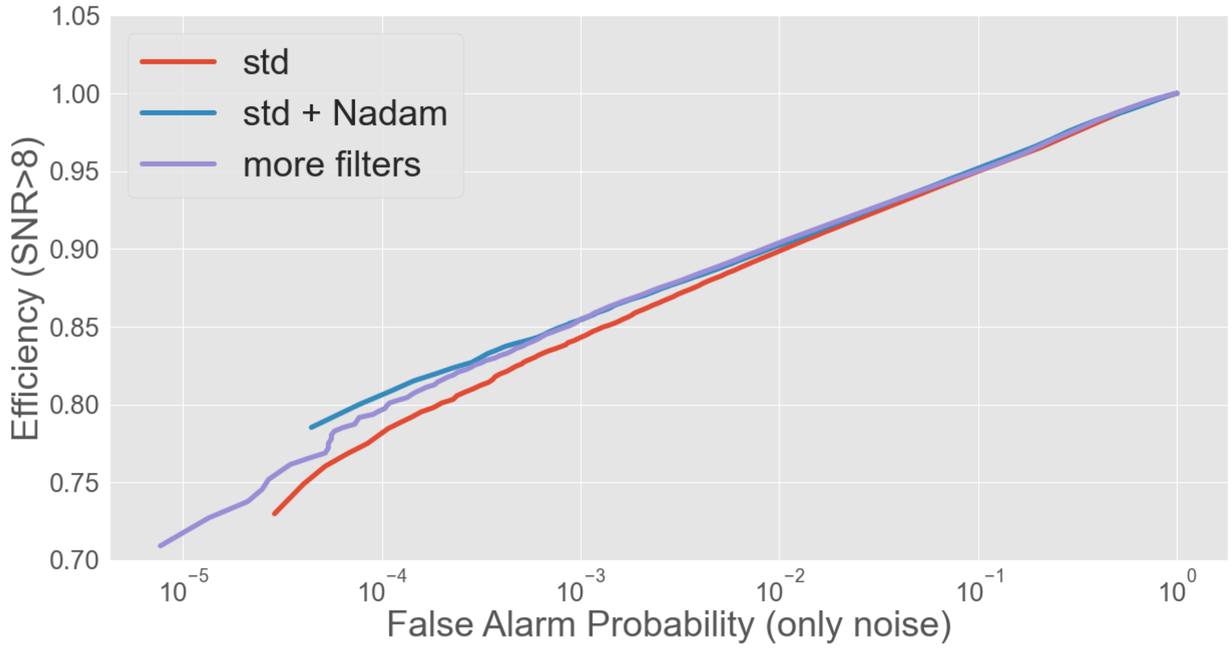
**Efficiency at SNR=8 when noise FAP<1e-4**

| | Cut | Efficiency | Glitch FAP |
|---|---|---|---|
| **Standard** | 0.94 | 78% | 2.37e-04 |
| **Std + Nadam** | 0.98 | 80% | 1.62e-04 |
| **More filters** | 0.80 | 80% | 2.25e-04 |

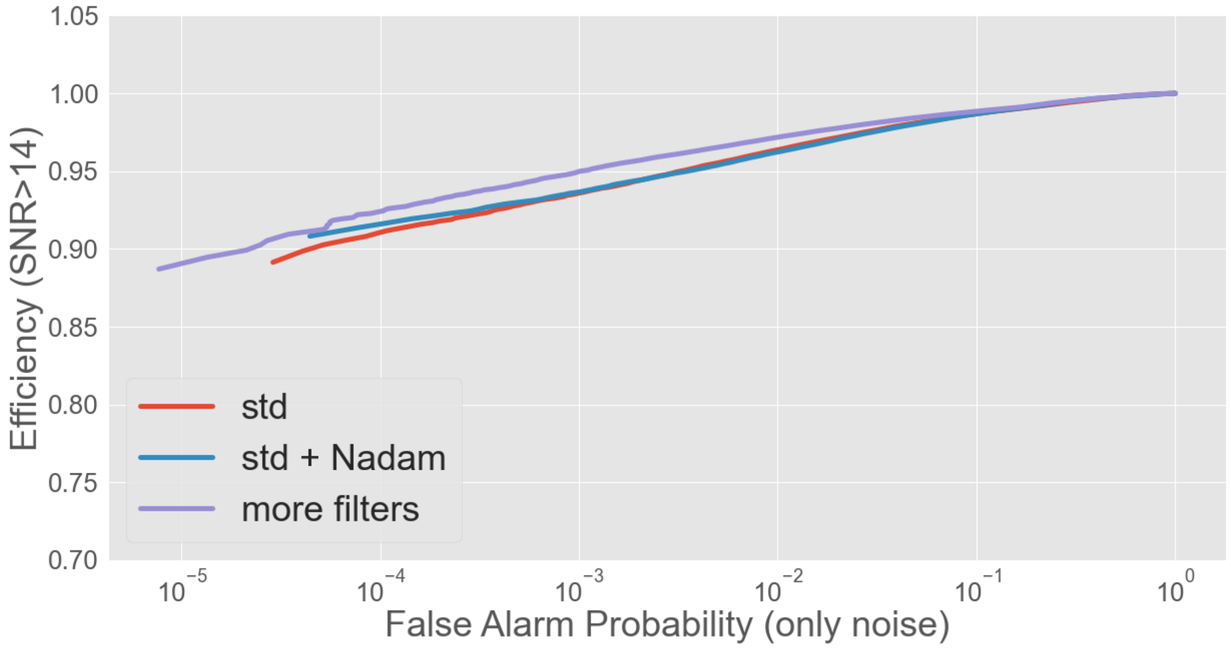
# ROC: efficiency vs FAP

- Nadam optimiser allows to get an improvement
- Increasing the number of filters goes also in the right direction and the improvement is more evident at higher SNR

**SNR>8**




**SNR>14**



# Temporal Convolutional Network

- Web page: https://github.com/philipperemy/keras-tcn
- Paper: https://arxiv.org/abs/1803.01271
- Easy to install: *pip install keras-tcn*

2017).) The distinguishing characteristics of TCNs are: 1) the convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past; 2) the architecture can take a sequence of any length and map it to an output sequence of the same length, just as with an RNN. Beyond this, we emphasize how to build very long effective history sizes (i.e., the ability for the networks to look very far into the past to make a prediction) using a combination of very deep networks (augmented with residual layers) and dilated convolutions.

Pay attention to the **receptive field** (you how far the model can see in terms of timesteps)

$$R_{field} = 1 + 2 \cdot (K_{size} - 1) \cdot N_{stack} \cdot \sum_i d_i$$

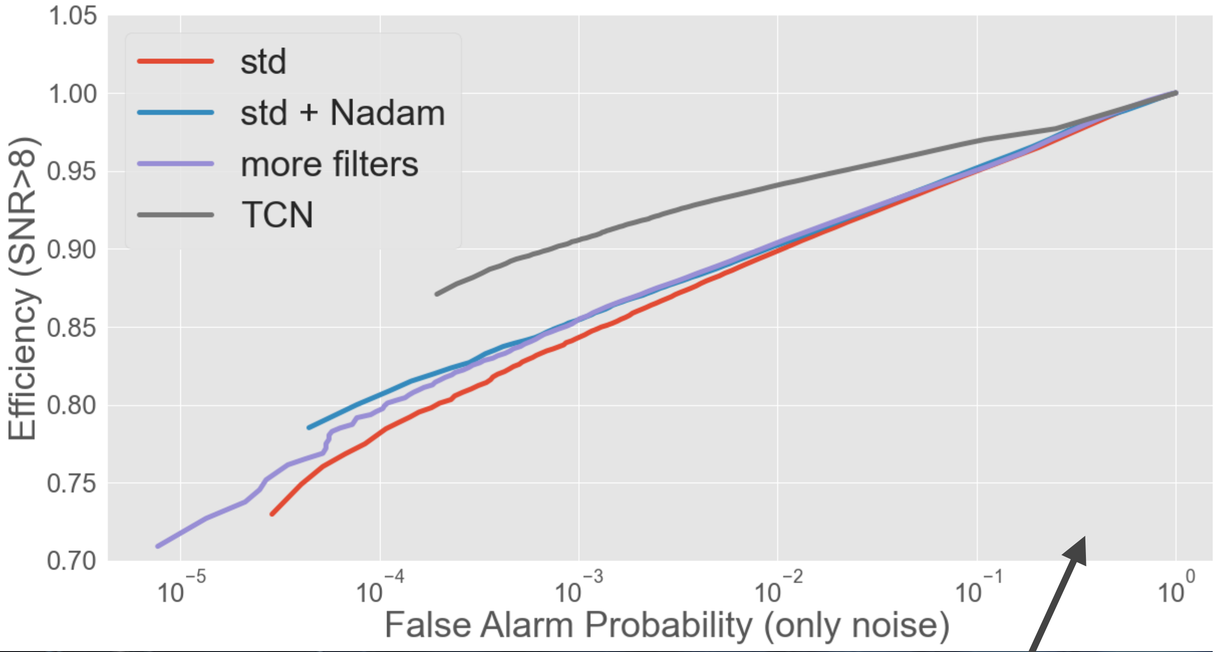
Arguments of the TCN

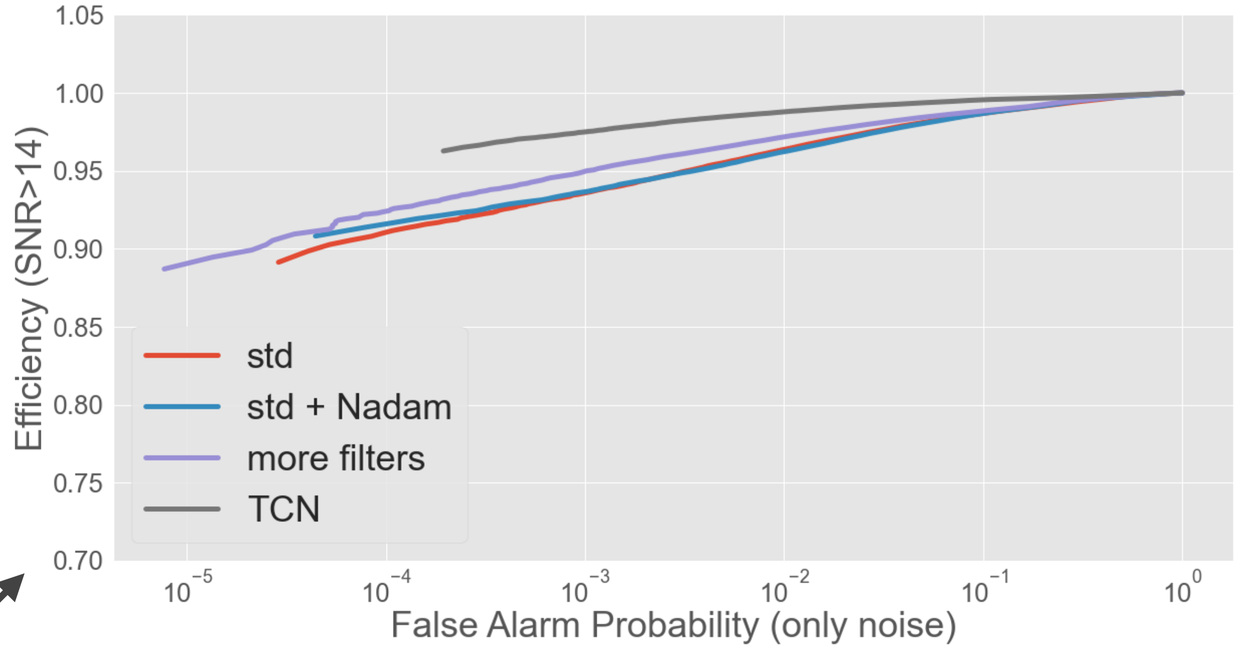
```
TCN(
        nb_filters=64,
        kernel_size=3,
        nb_stacks=1,
        dilations=(1, 2, 4, 8, 16, 32),
        padding='causal',
        use_skip_connections=True,
        dropout_rate=0.0,
        return_sequences=False,
        activation='relu',
        kernel_initializer='he_normal',
        use_batch_norm=False,
        use_layer_norm=False,
        use_weight_norm=False,
        **kwargs
)
```

Same number of filters and kernel size in all the layers

By default 6 layers

**Results given here: nb_filters=32, kernel_size=16**



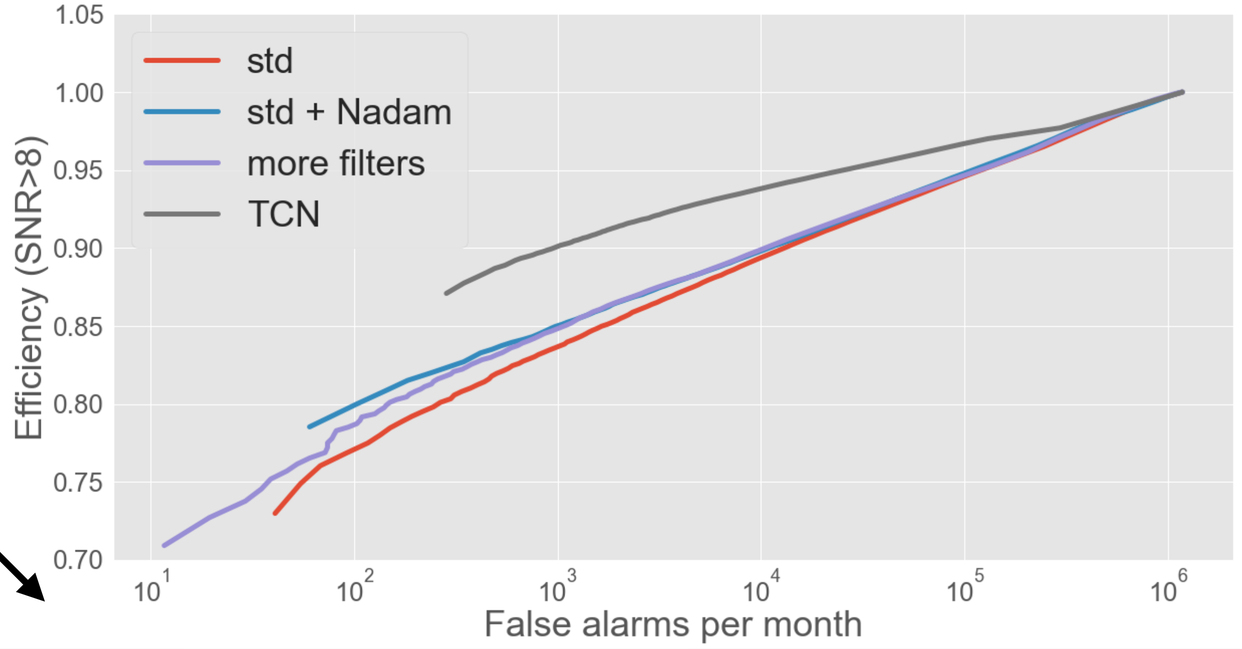

TCN: good efficiency vs FAP but doesn't allow to reduce the minimum FAP

# false alarms per month:

FAP_noise * #noise + FAP_glitch * #glitch

#noise and #glitch is the number of 1 s segments of glitch and noise in 1 month of O1

(rough estimate…)

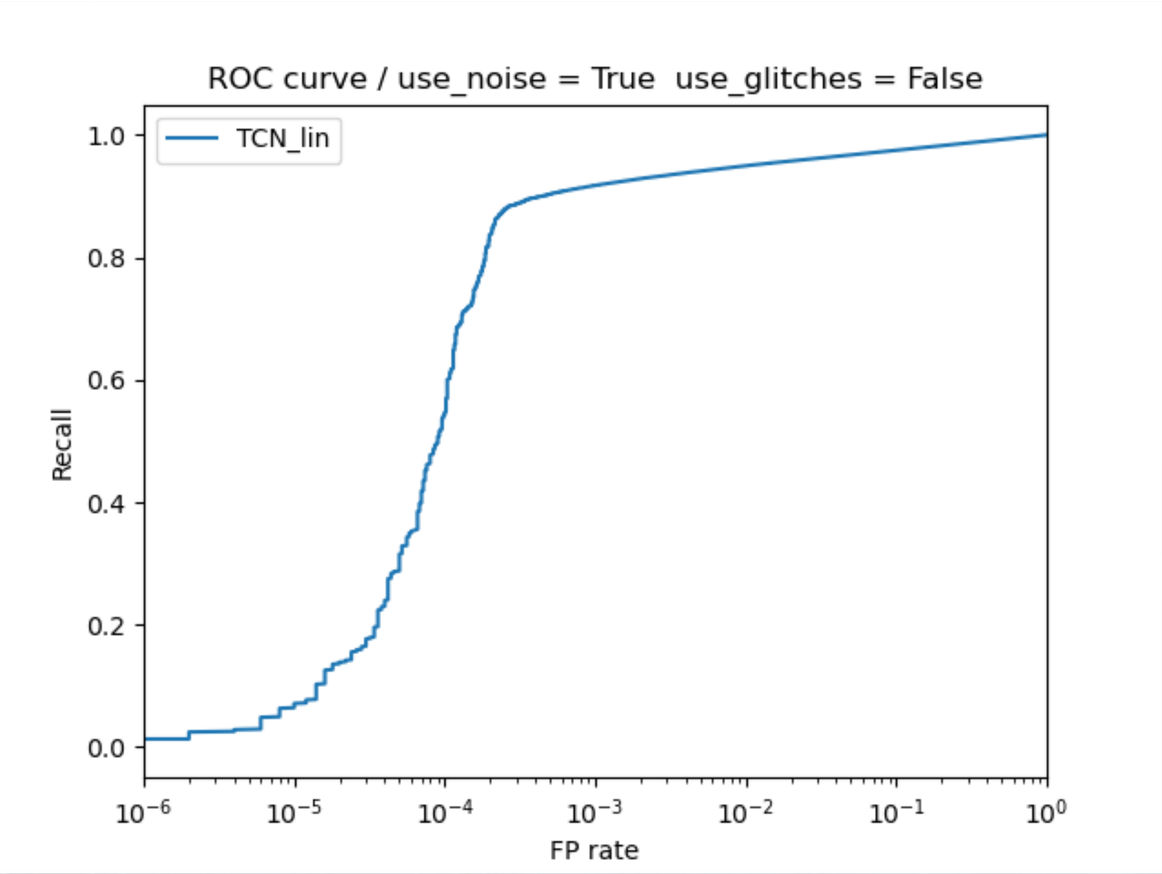


# Conclusion

- Tested CNN on real LIGO noise including glitch classificaton
- Increasing the number of filters and using Nadam as optimiser improve the results
- TCN needs further tuning
- CNN can reach 70 % efficiency at a false alarm rate of 10/months (for 1 month of O1)
- Probably can go at lower false alarm rate by removing the endmost softmax activation as suggested by Schäfer et al.
- Deep NN for time-series classification look very promising!

# Backup slides

# TCN + no softmax



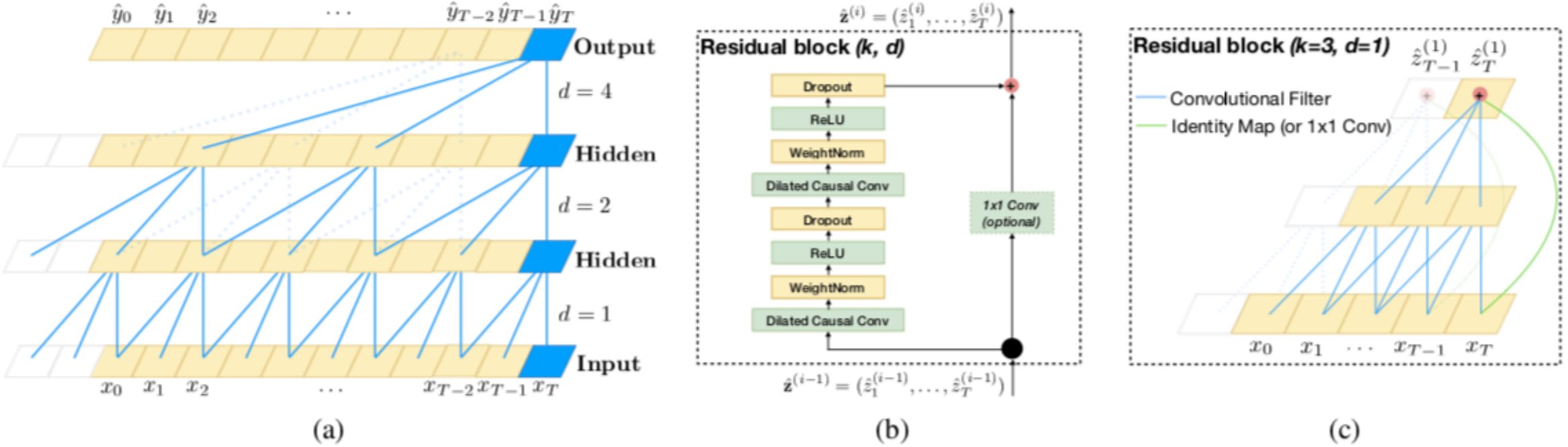



*Figure 1.* Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

George et al.

Phys. Rev. D 97, 044039 (2018)

| | | |
|---|---|---|
| | Input | vector (size: 8192) |
| 1 | Reshape | matrix (size: 1×8192) |
| 2 | Convolution | matrix (size: 16×8177) |
| 3 | Pooling | matrix (size: 16×2044) |
| 4 | ReLU | matrix (size: 16×2044) |
| 5 | Convolution | matrix (size: 32×2016) |
| 6 | Pooling | matrix (size: 32×504) |
| 7 | ReLU | matrix (size: 32×504) |
| 8 | Convolution | matrix (size: 64×476) |
| 9 | Pooling | matrix (size: 64×119) |
| 10 | ReLU | matrix (size: 64×119) |
| 11 | Flatten | vector (size: 7616) |
| 12 | Linear Layer | vector (size: 64) |
| 13 | ReLU | vector (size: 64) |
| 14 | Linear Layer | vector (size: 2) |
| | Output | vector (size: 2) |

Gabbard et al.

Phys. Rev. Lett. 120, 141103 (2018)

| | Layer | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Parameter (Option) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Type | *C* | *C* | *C* | *C* | *C* | *C* | *H* | *H* | *H* |
| No. Neurons | 8 | 8 | 16 | 16 | 32 | 32 | 64 | 64 | 2 |
| Filter size | 64 | 32 | 32 | 16 | 16 | 16 | Not applicable | Not applicable | Not applicable |
| Max pool size | Not applicable | 8 | Not applicable | 6 | Not applicable | 4 | Not applicable | Not applicable | Not applicable |
| Drop out | 0 | 0 | 0 | 0 | 0 | 0 | 0.5 | 0.5 | 0 |
| Activation function | Elu | Elu | Elu | Elu | Elu | Elu | Elu | Elu | SMax |

# Nadam Optimization Algorithm

- The Nesterov-accelerated Adaptive Moment Estimation, or the Nadam, algorithm is an extension to the Adaptive Movement Estimation (Adam) optimization algorithm to add Nesterov's Accelerated Gradient (NAG) or Nesterov momentum
- 2016: https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ
- Momentum adds an exponentially decaying moving average (first moment) of the gradient to the gradient descent algorithm. This has the impact of smoothing out noisy objective functions and improving convergence.