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

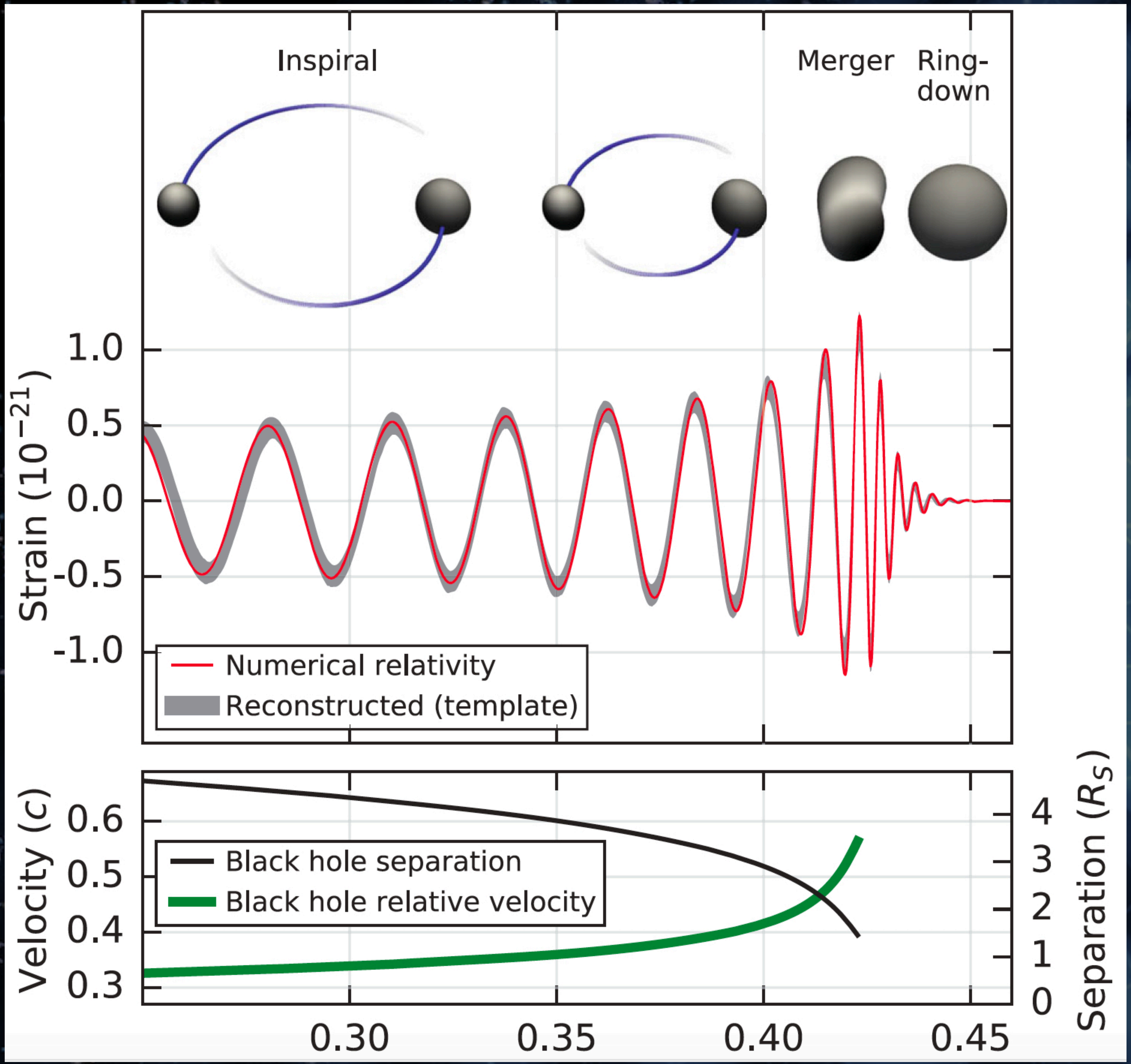
A. Trovato* with M. Bejger and E. Chassande-Mottin, N. Courty, R. Flamary, H. Marchand
*Università di Trieste, INFN-Sezione Trieste

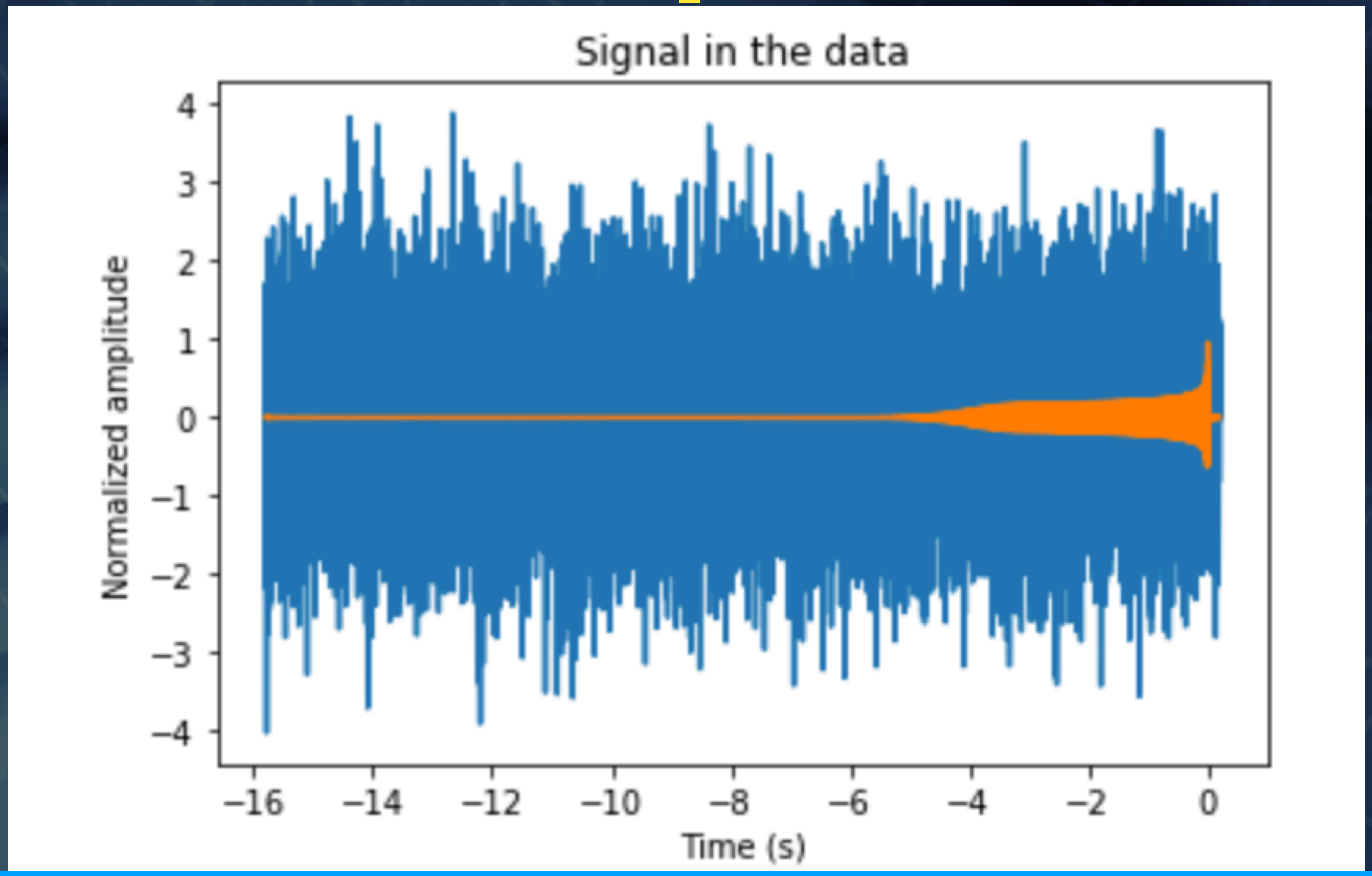
UNIVERSITÀ DEGLI STUDI DI TRIESTE

G2NET

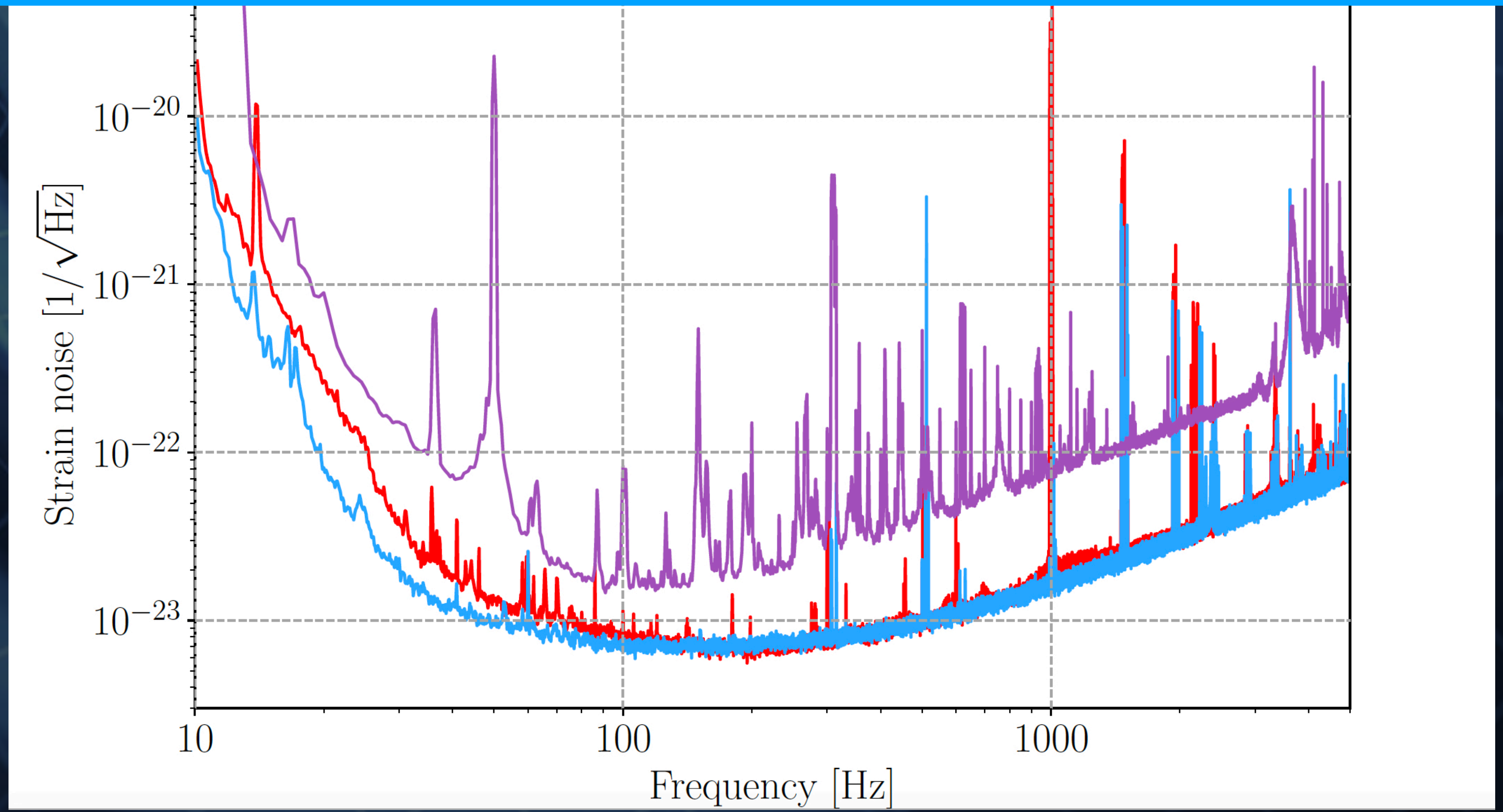
INFN Istituto Nazionale di Fisica Nucleare

Gravitational waves detection problem

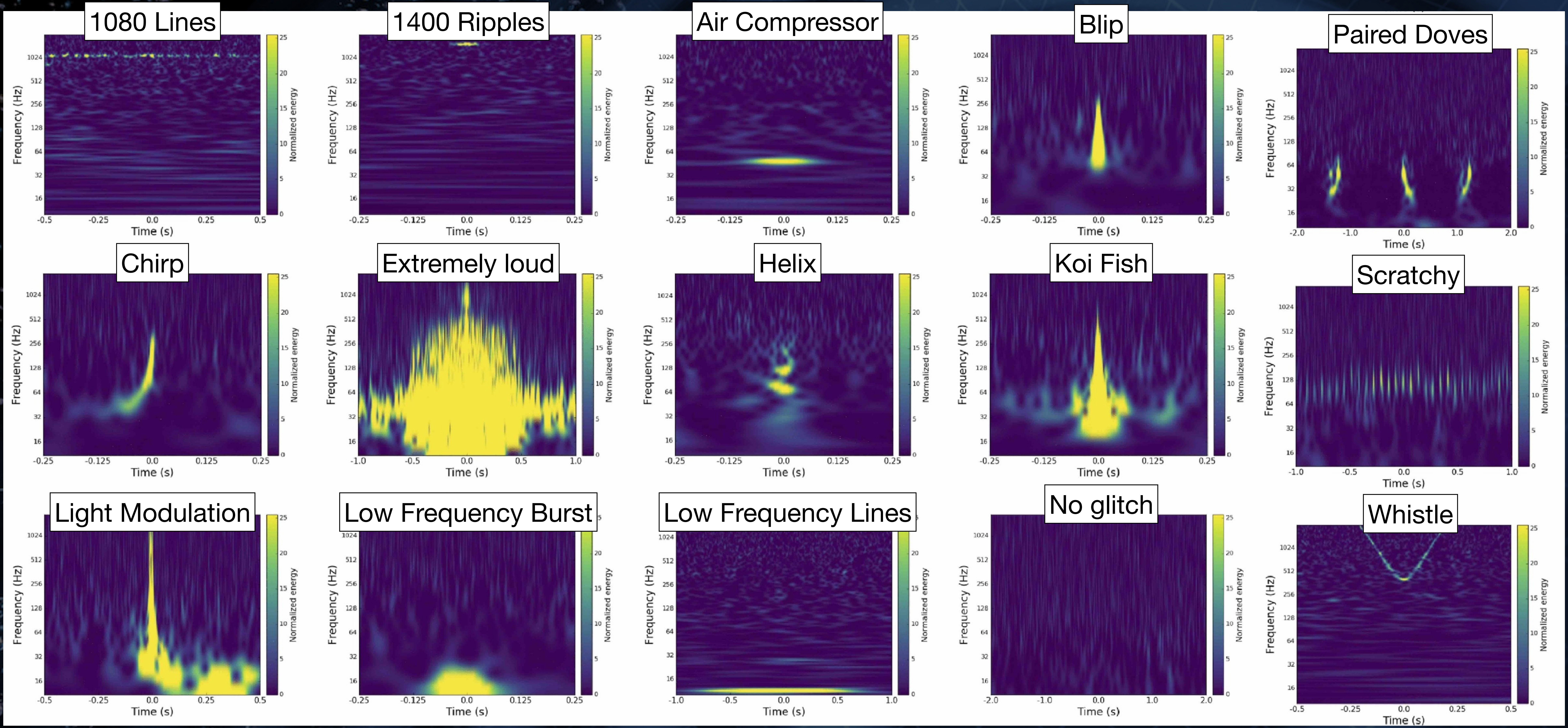




Rare and weak signals in complex background: non-Gaussian non-stationary



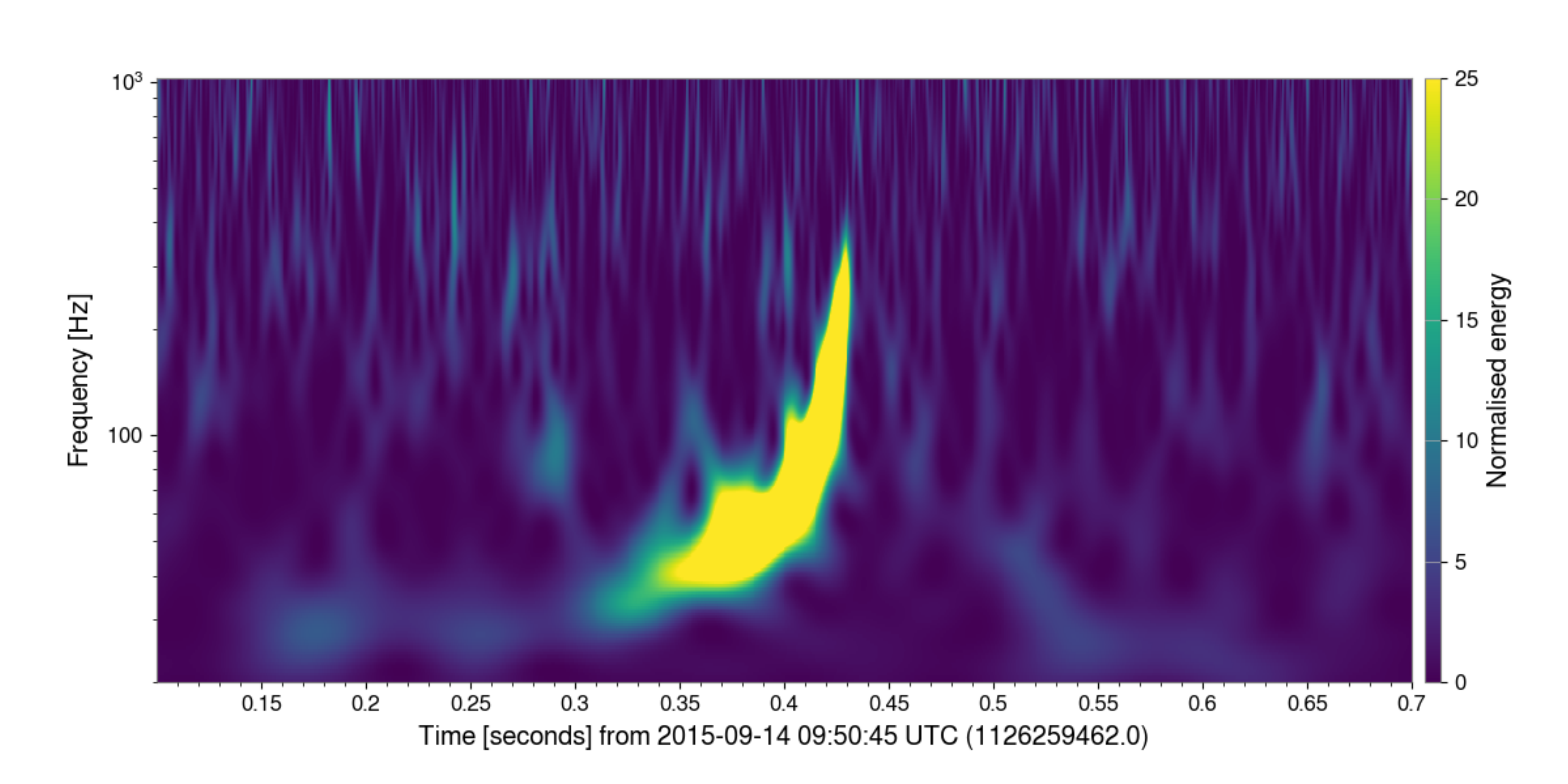
Glitches zoo

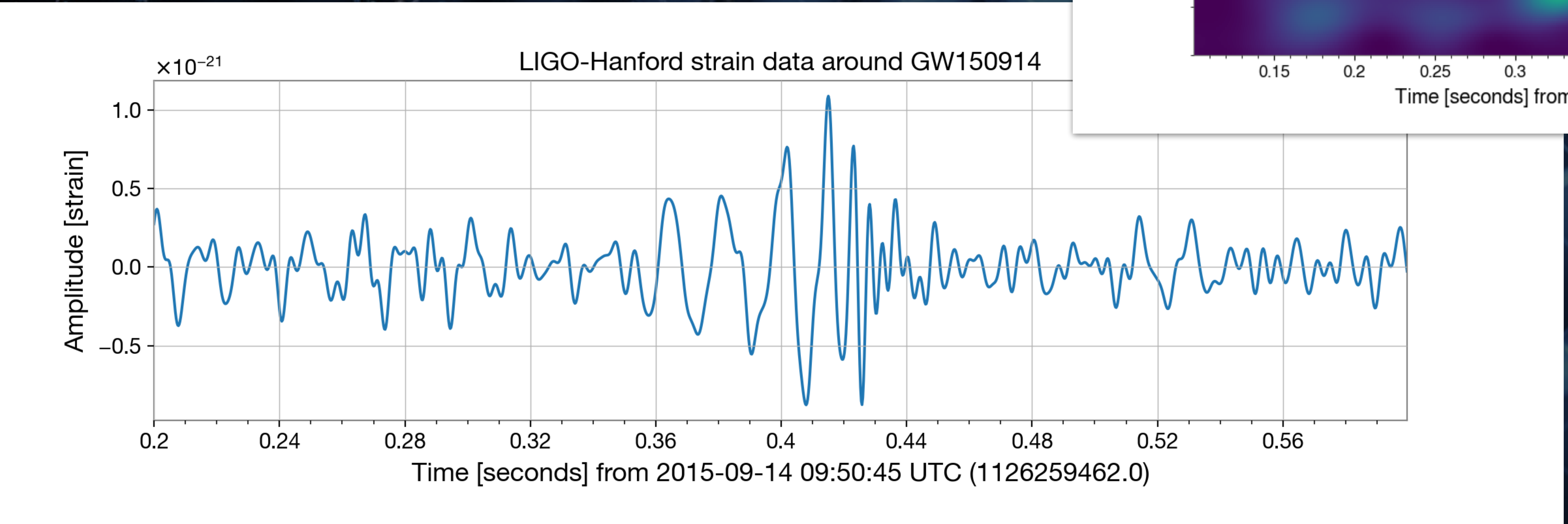


⋆ **Credits: Gravity Spy dataset**

ML used for GW signal detection

- **Data representation**
  - ✓ Spectrogram vs Time series





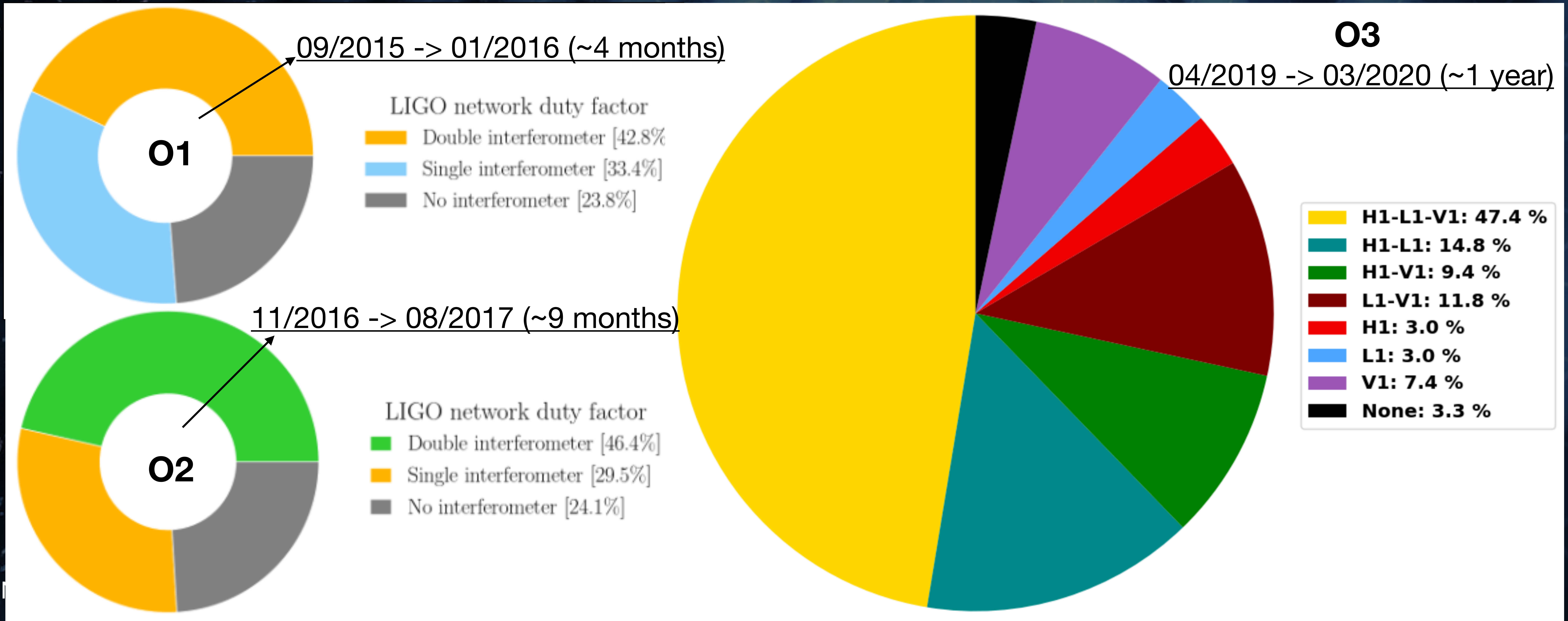
ML used for GW signal detection

- **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 usable a lower FAR is needed
- **Recent work** (Schäfer et al.[3])
  - ✓ Explored different training strategies and solution for softmax
  - ✓ FAR ~ 1/month but on gaussian noise
- **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



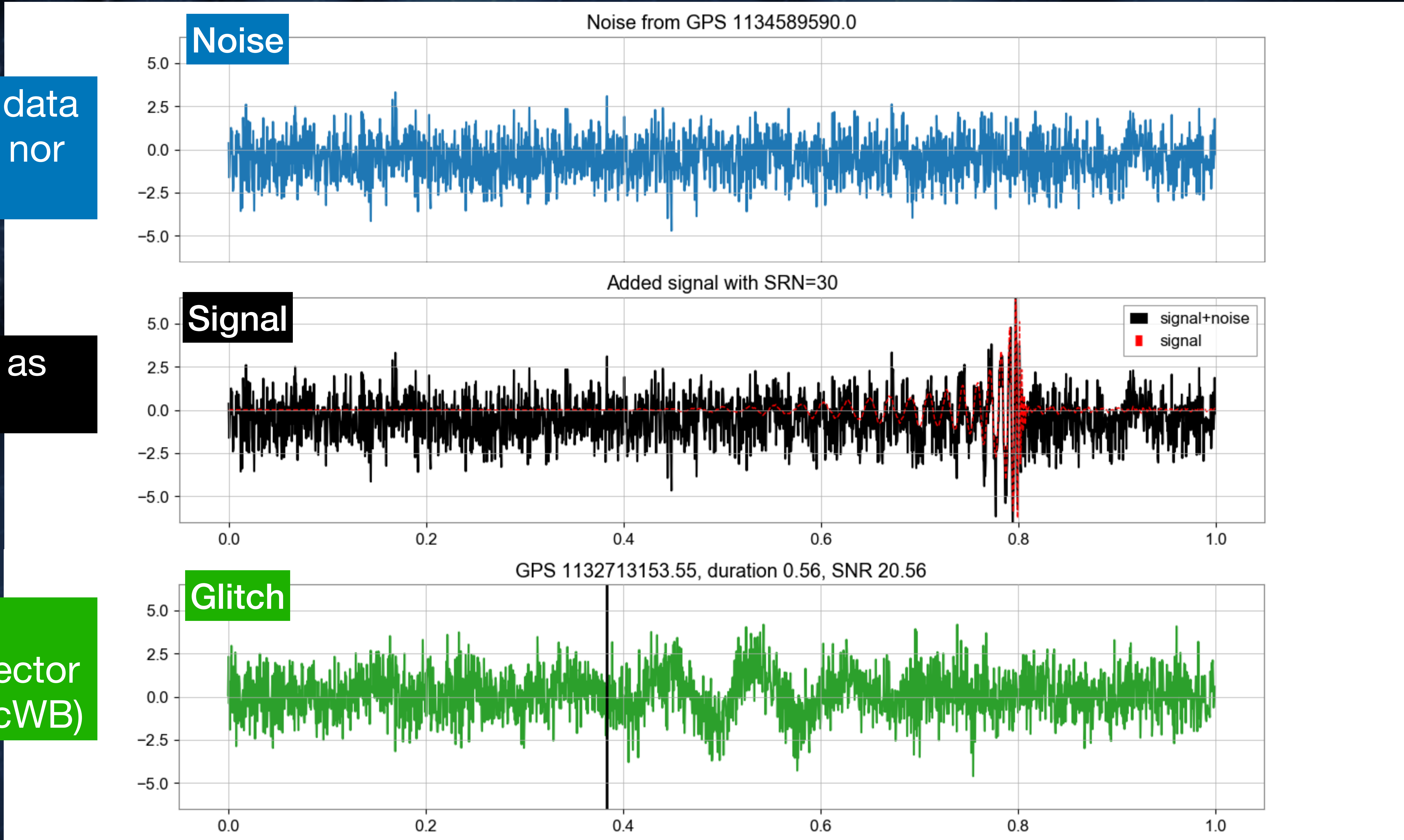
Training data: 3 classes

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.

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)

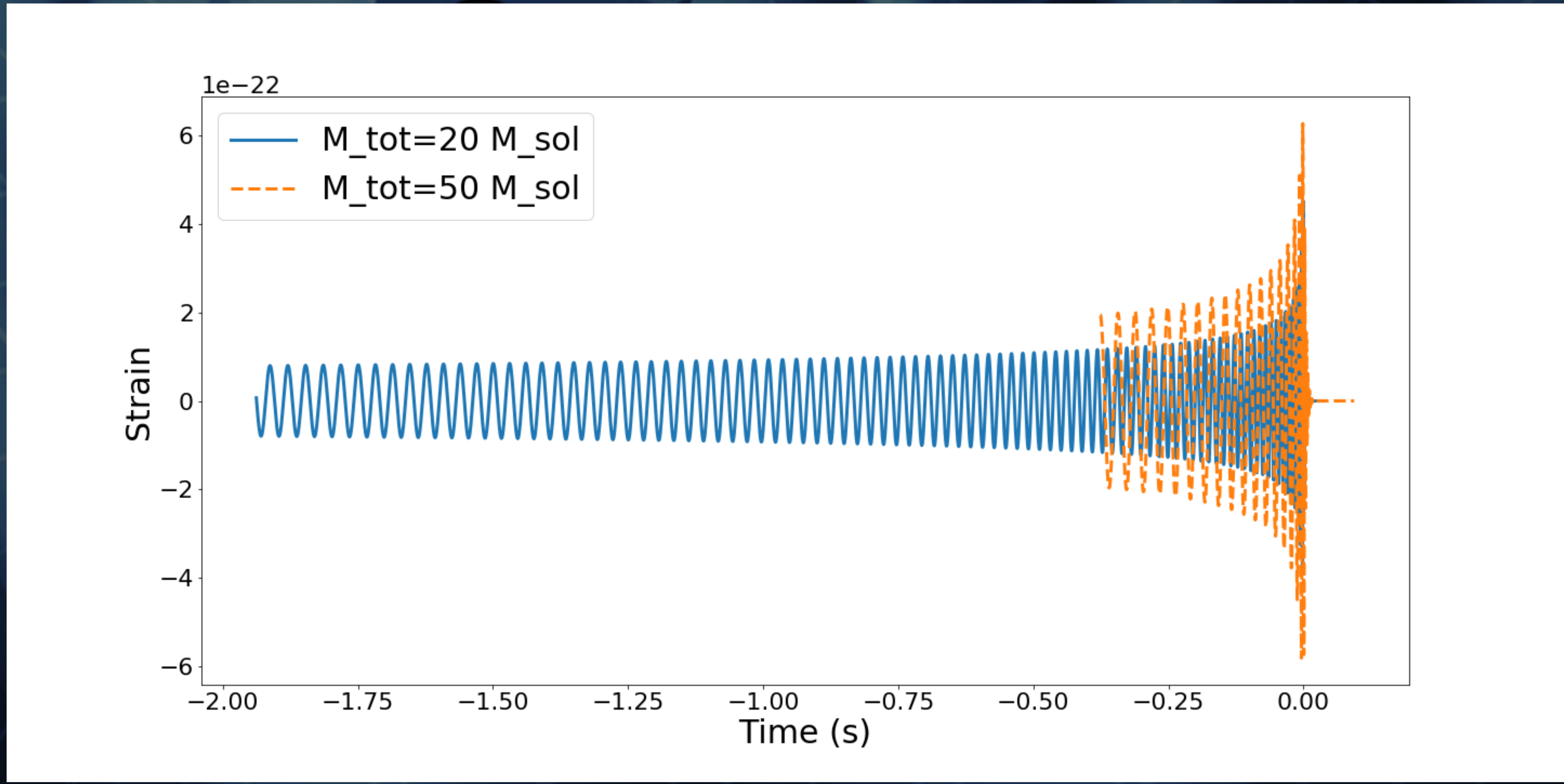


Details on the dataset

- Segments of fixed duration: **1 second**
- **Bandpass filter [20,1000] Hz**
- **No superposition** between segments in 1 month dataset
- Glitch **position random** in the segment (if short duration, fully contained) or tailing over multiple segments if duration > 1 s
- Samples for training:
  - Noise: 2.5e5
  - Signal: 2.5e5
  - Glitch: 0.7e5
- Samples for testing:
  - Noise: 5.2e5
  - Signal: 2.5e5
  - Glitch: 0.8e5

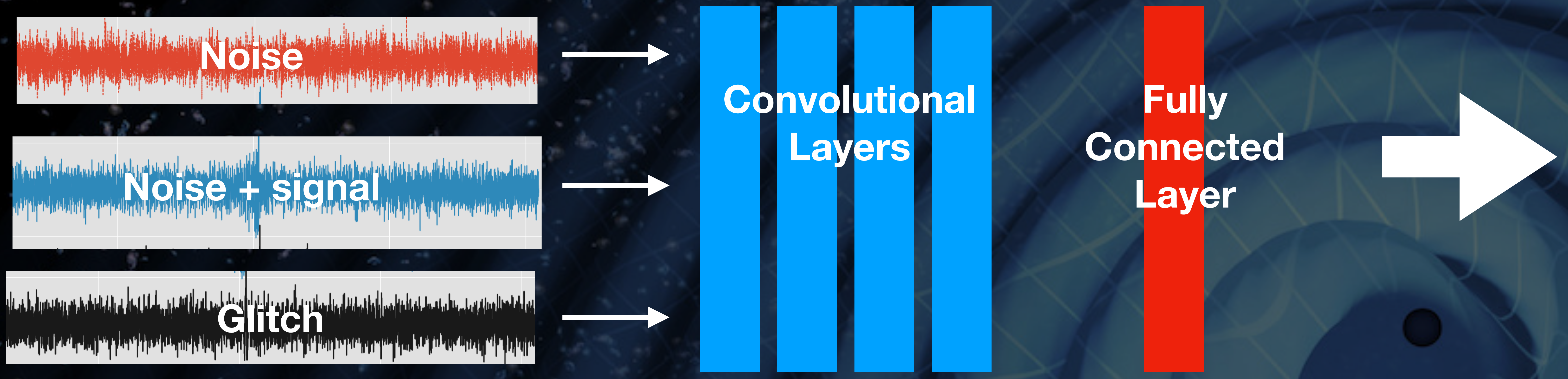
Signal injection:

- **Position random** in the segment but almost fully contained
- Type pf signal: (BBH)
  - **m1+m2** ∈ **(33,60) M⊙**
  - **SNR** ∈ **(8,20)**



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



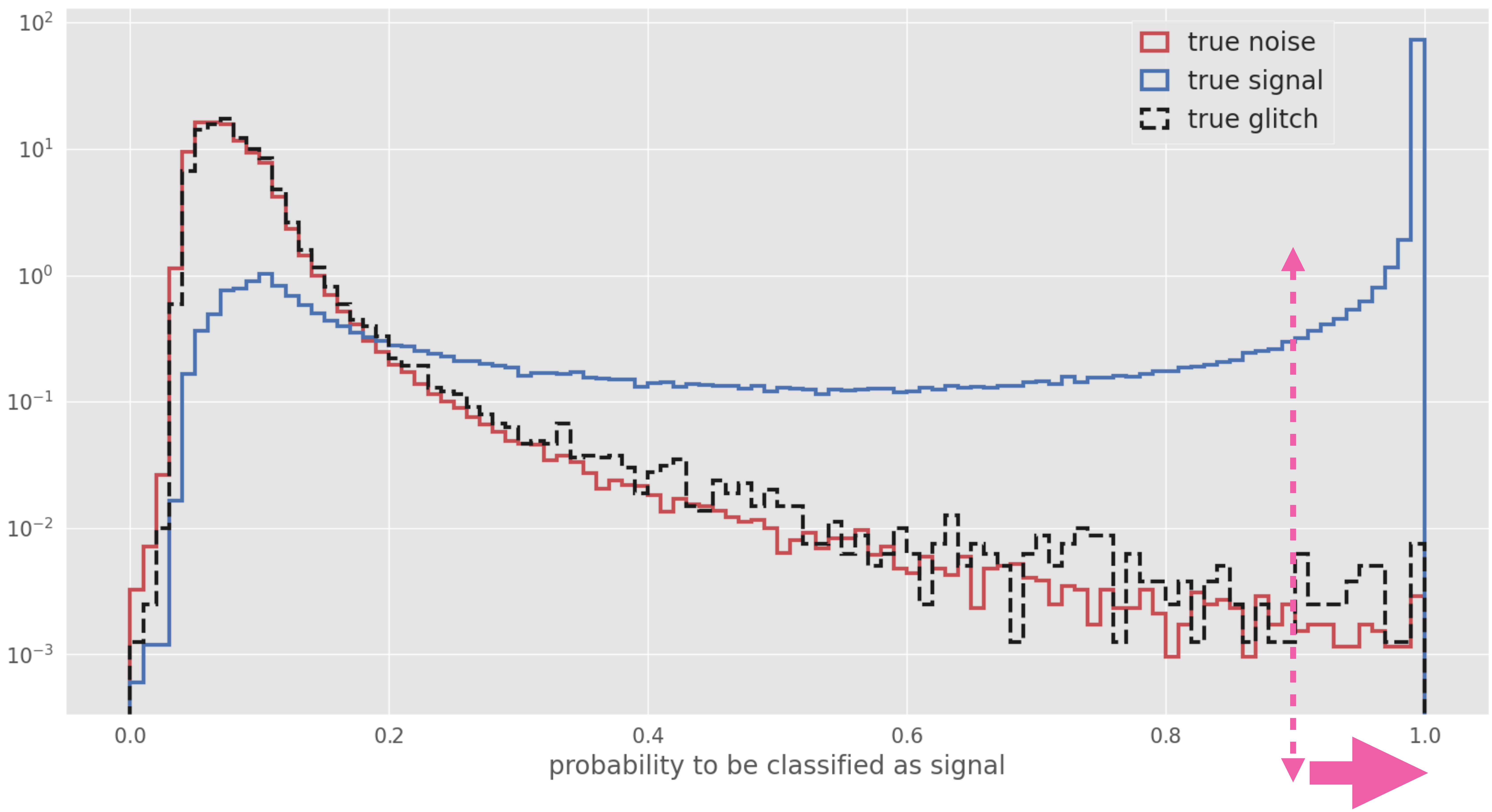
Output: probability of belonging to each class

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	-

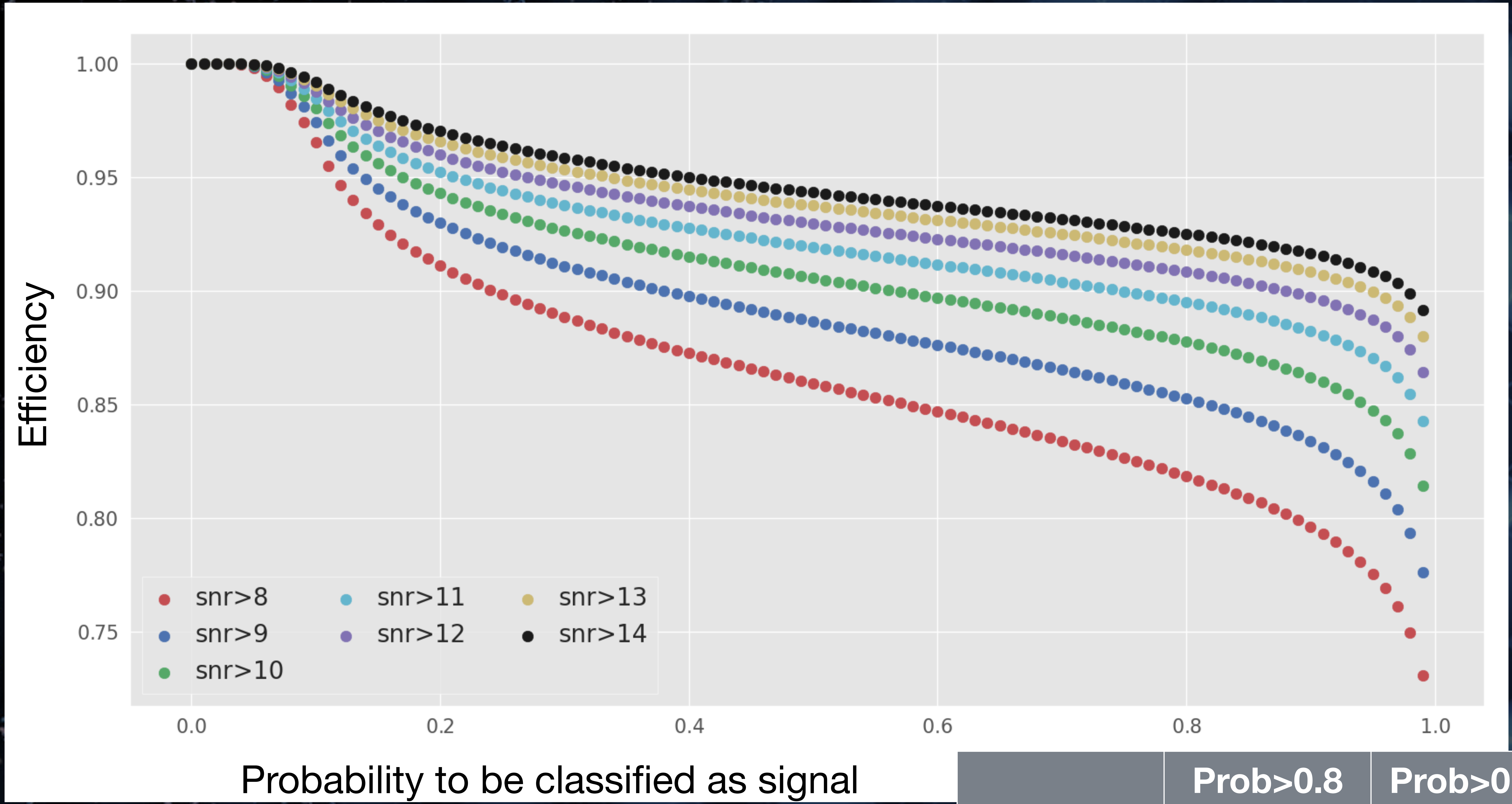
Optimiser: Adam (except otherwise indicated)

Probability to be classified as signal

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

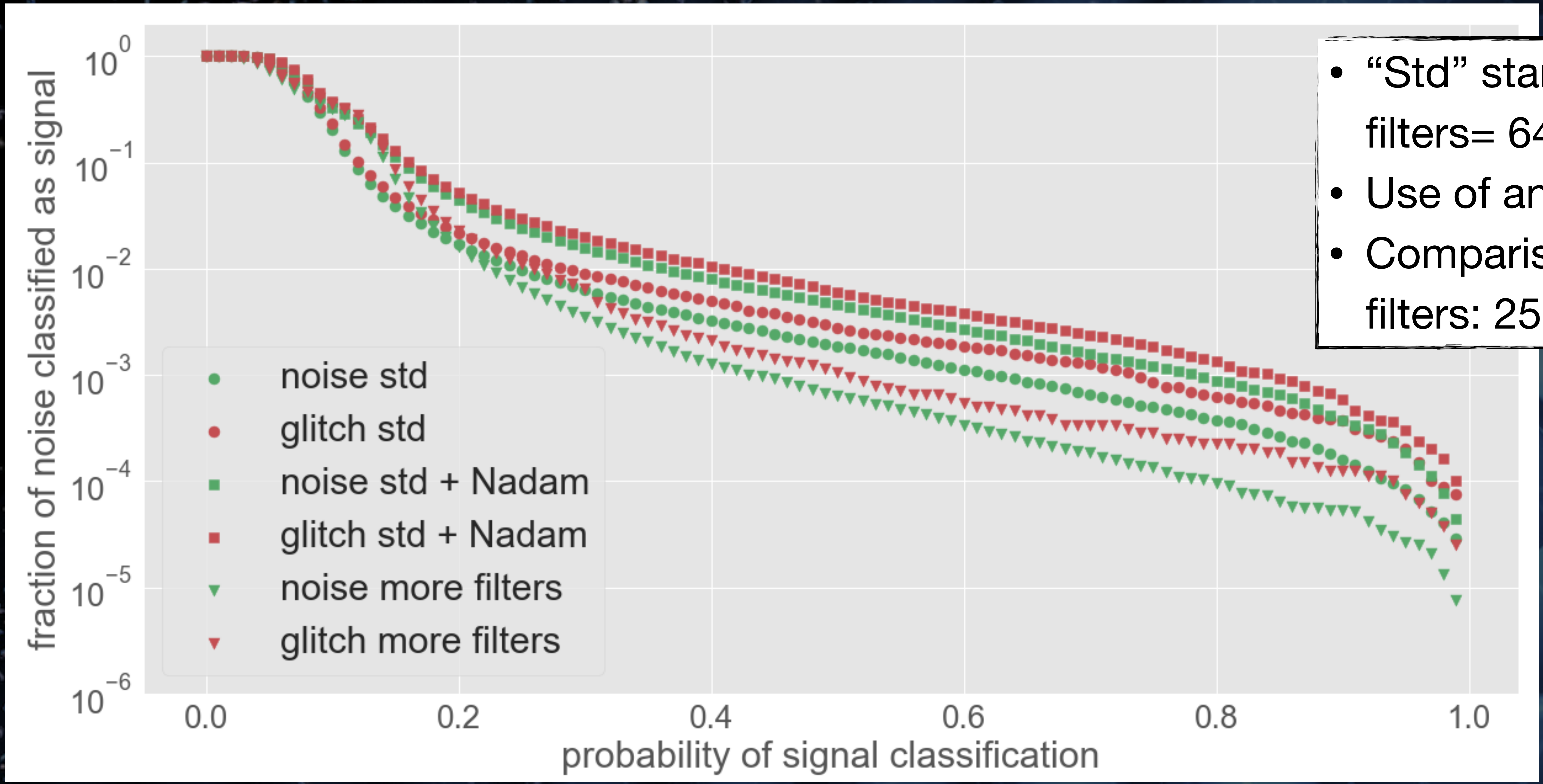


Efficiency vs probability



	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 Probability



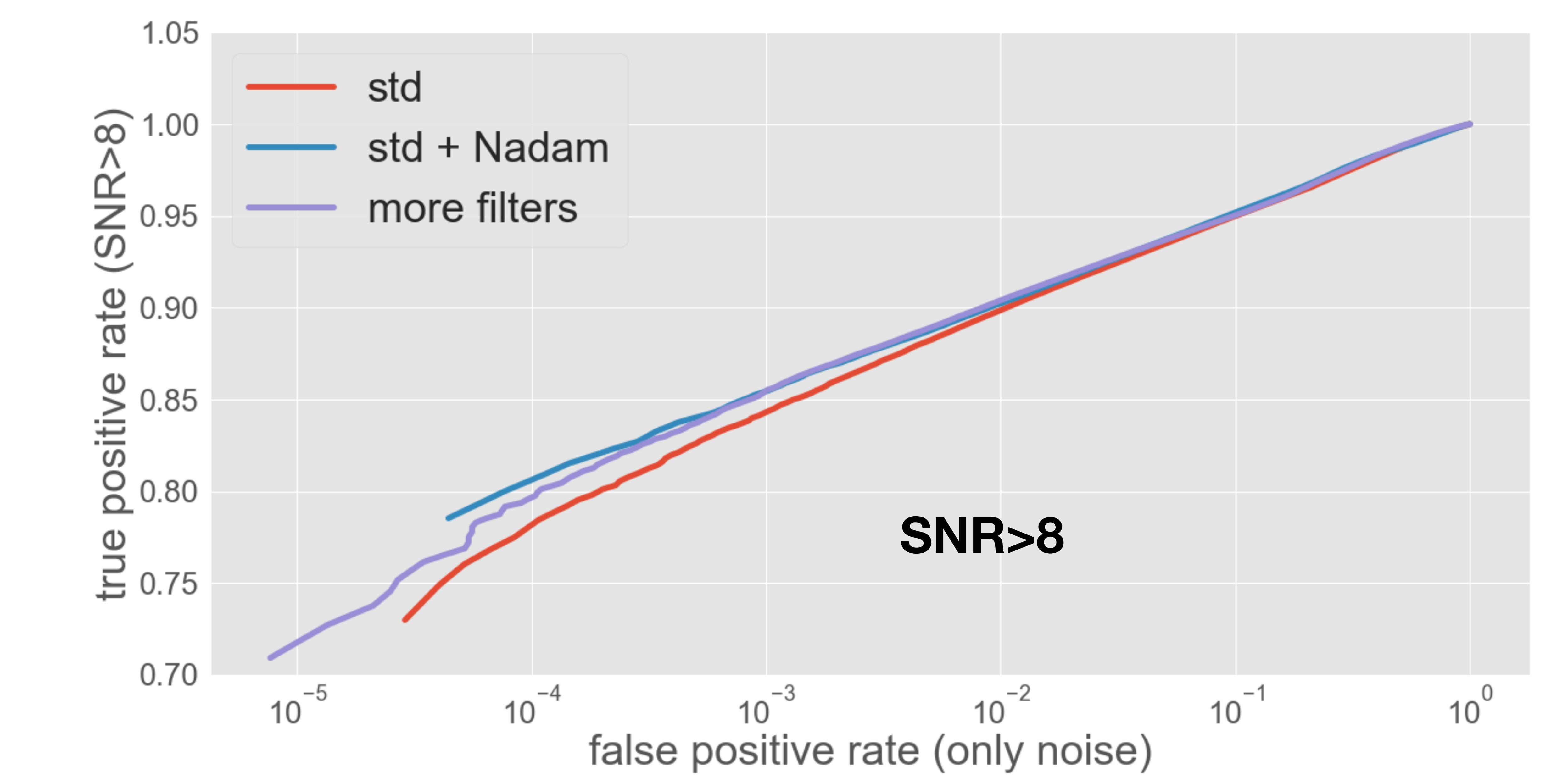
- "Std" standard architecture used for reference filters= 64,32,16,8
- Use of another optimiser Nadam
- Comparison with a similar architecture with more filters: 256,128, 64,64

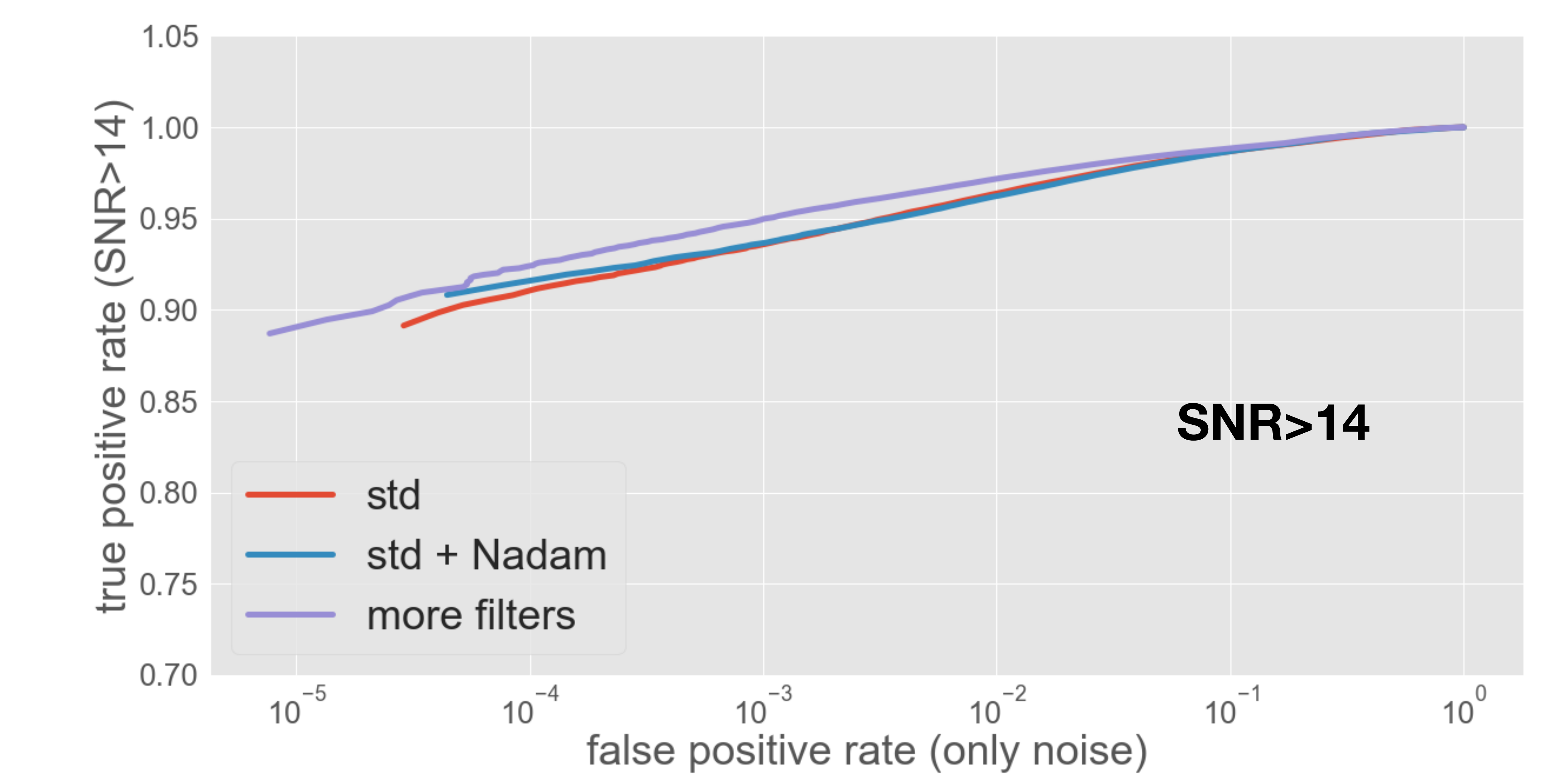
Efficiency at SNR=8 when noise FAP<1e-4

	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





NN architectures for time series

- Literature of NN architectures for time series
- TCN: Temporal Convolutional Network (next slides)
- IT: Inception Time (https://arxiv.org/abs/1909.04939)
- g2net kaggle competition: a lot of results used EfficientNet (arXiv: 1905.11946v5)
  - ✓ new scaling method that uniformly scales all dimensions of depth/ width/resolution using a simple yet highly effective compound coefficient
  - ✓ NOT TRIED (YET)

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

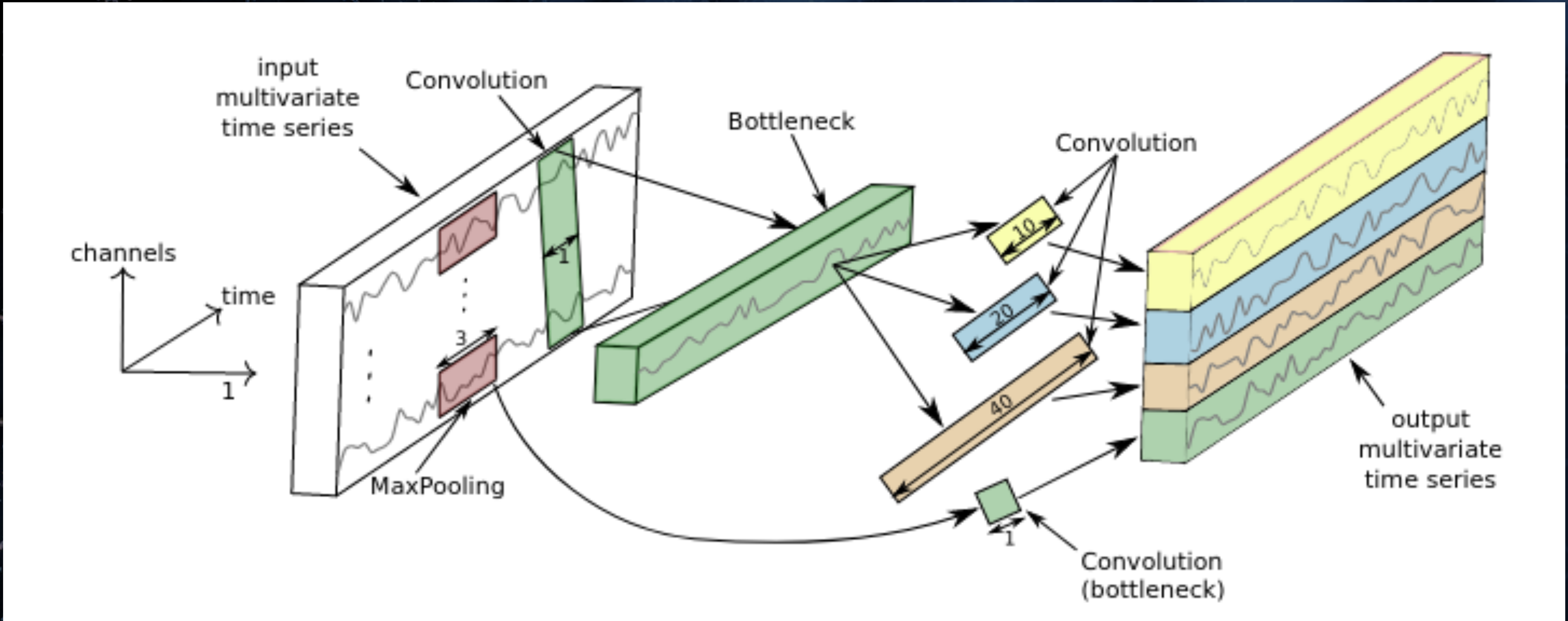
```
TCN(
    nb_filters=64,
    kernel_size=3,
    nb_stacks=1,
    dilations=(1, 2, 4, 8, 16, 32),
    padding='causal',
    use_skip_connections=True,
    dropout_rate=0.0,
    return_sequences=False,
    activation='relu',
    kernel_initializer='he_normal',
    use_batch_norm=False,
    use_layer_norm=False,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

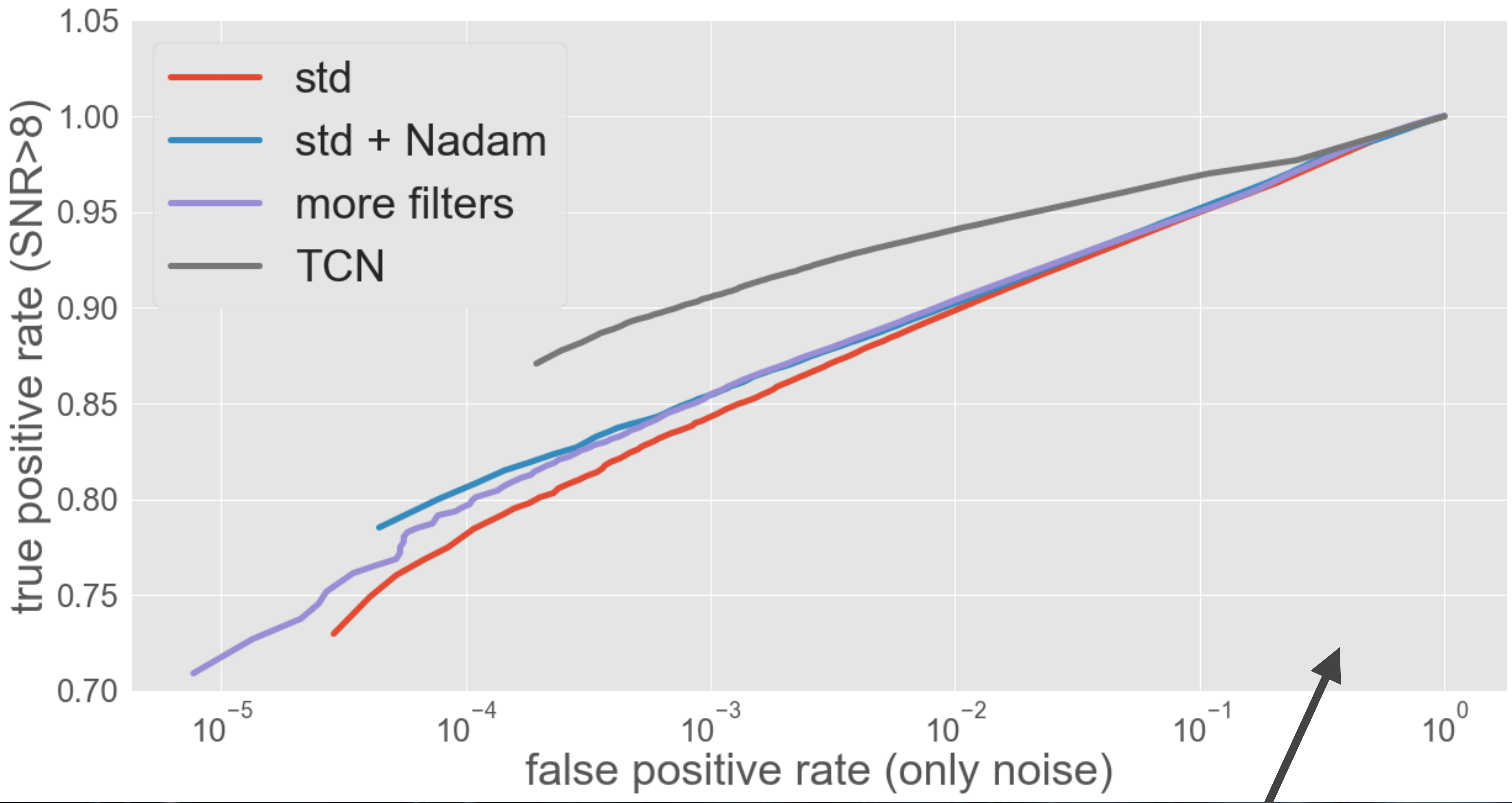
Same number of filters and kernel size in all the layers

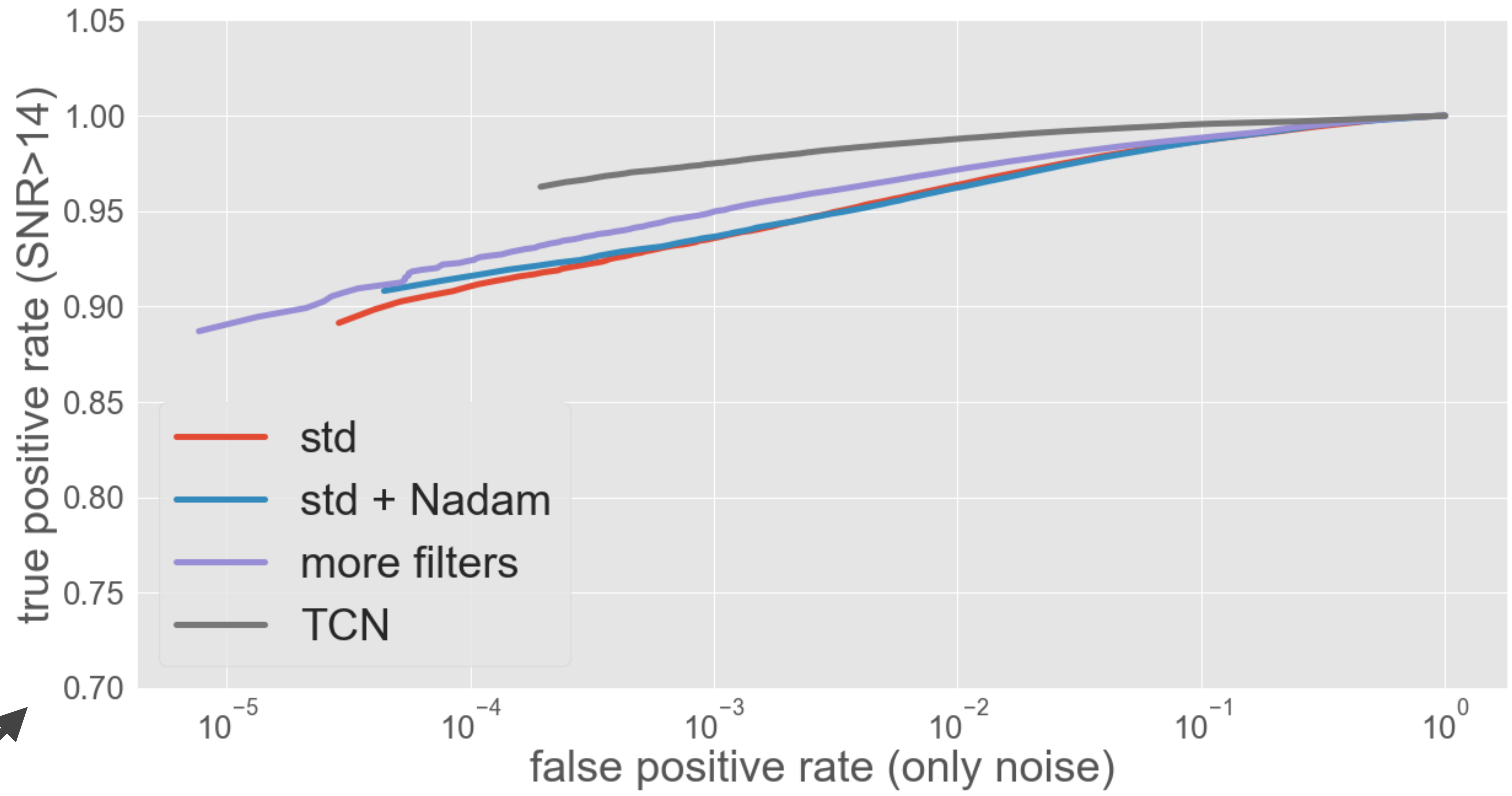
By default 6 layers

Results given here: nb_filters=32, kernel_size=16

Inception time

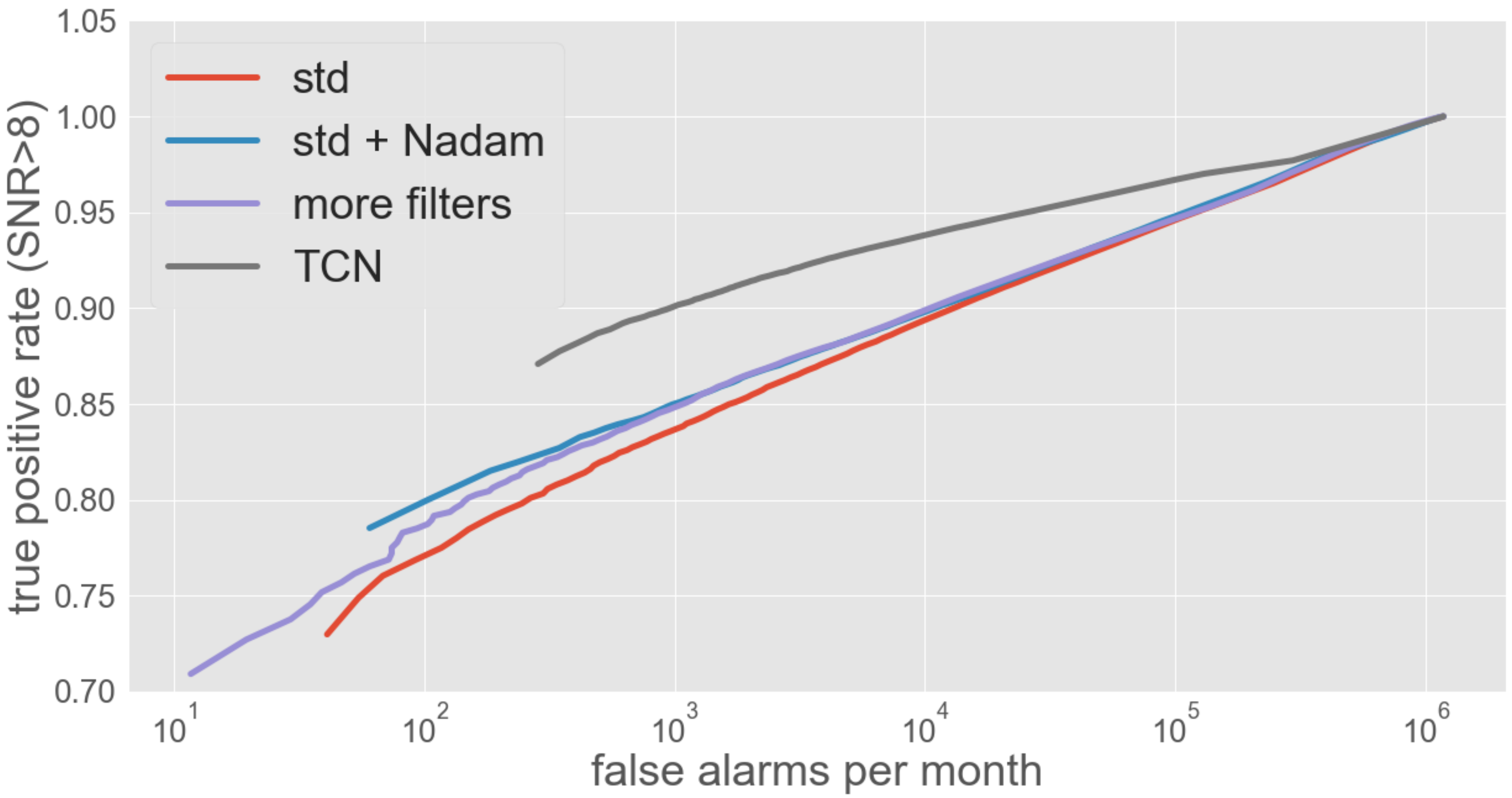




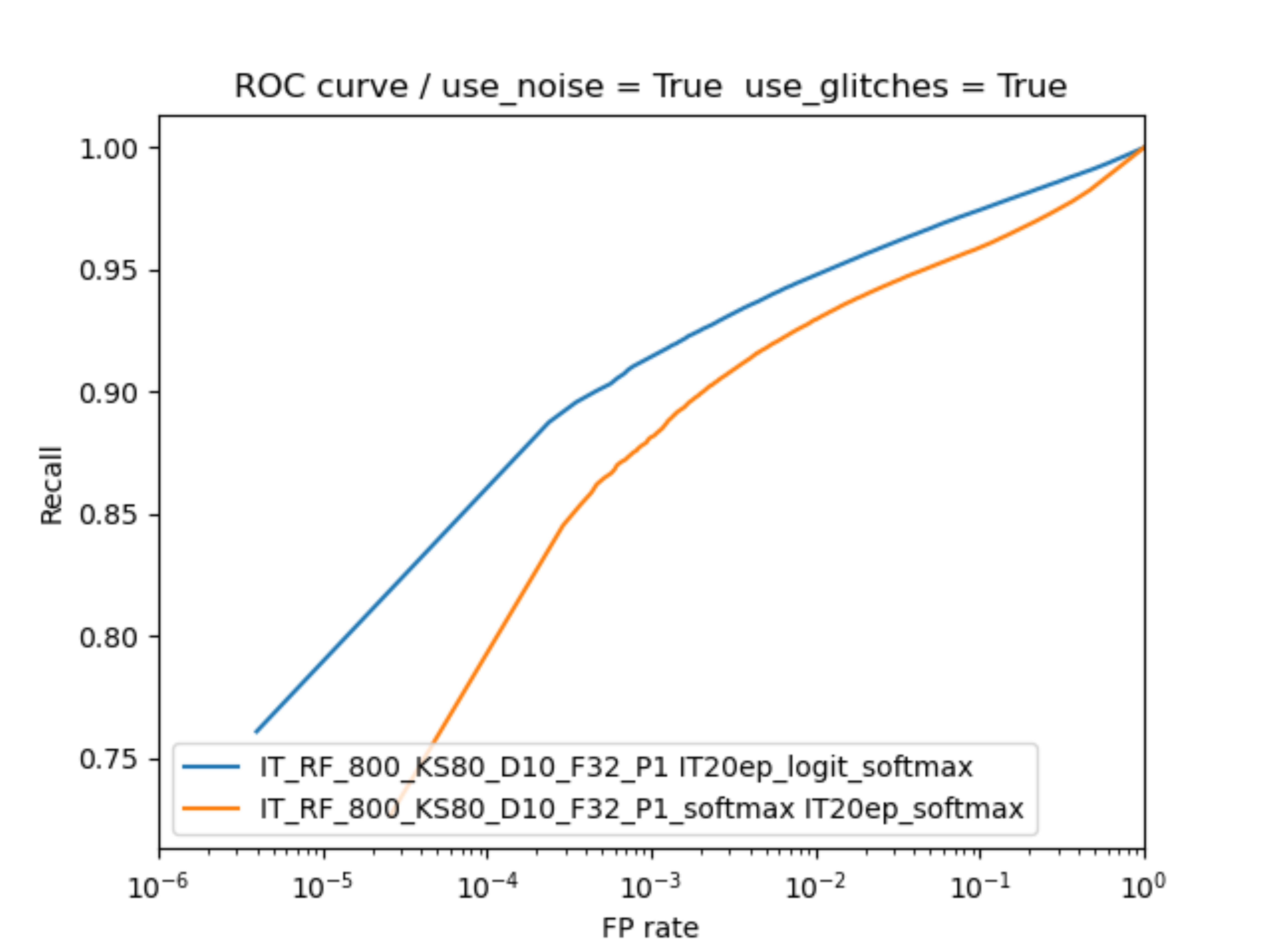


TCN: good ratio efficiency vs FAP but doesn't allow to reduce the minimum FAP

false alarms per months obtained by:
FAP_noise * #_1sec_noise_seg_1month_O1 +
FAP_glitch * #_1sec_glicth_seg_1month_O1
(rough estimate…)



Activation effect + IT



Network: Inception Time

- Biggest kernel size = 80
- Depth (number of modules) = 10
- Number of filters = 32

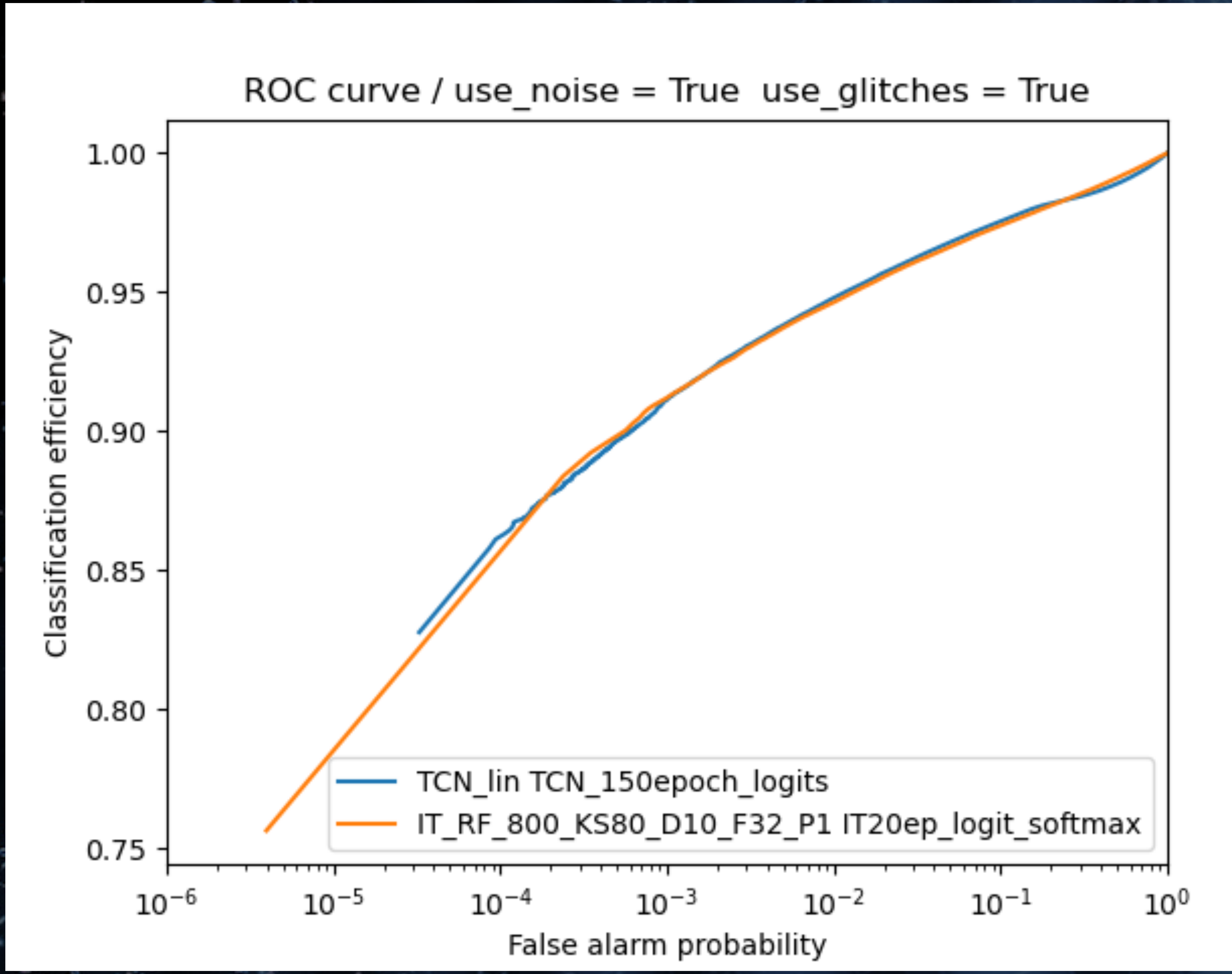
Blu line:

- *activation=None* in the output layer of the network
- *keras.losses.CategoricalCrossentropy(from_logits=True)* as loss in model.compile
- Softmax applied at the end to get the predictions

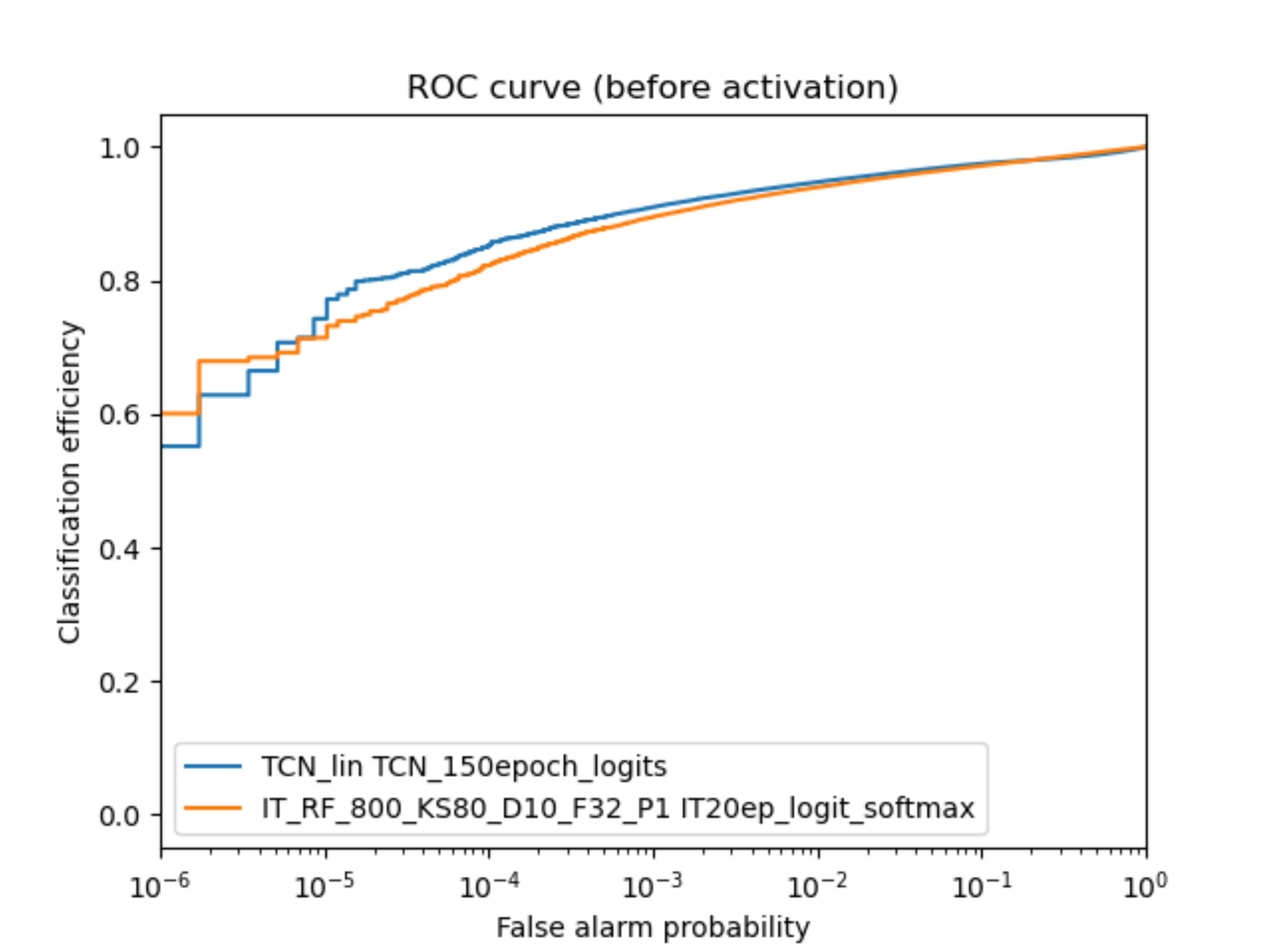
Orange line:

- *activation='softmax'* in the output layer of the network
- *keras.losses.get('categorical_crossentropy')* as loss in model.compile

TCN vs IT



ROC curve / use_noise = True use_glitches = True
Classification efficiency
False alarm probability
TCN_lin TCN_150epoch_logits
IT_RF_800_KS80_D10_F32_P1 IT20ep_logit_softmax



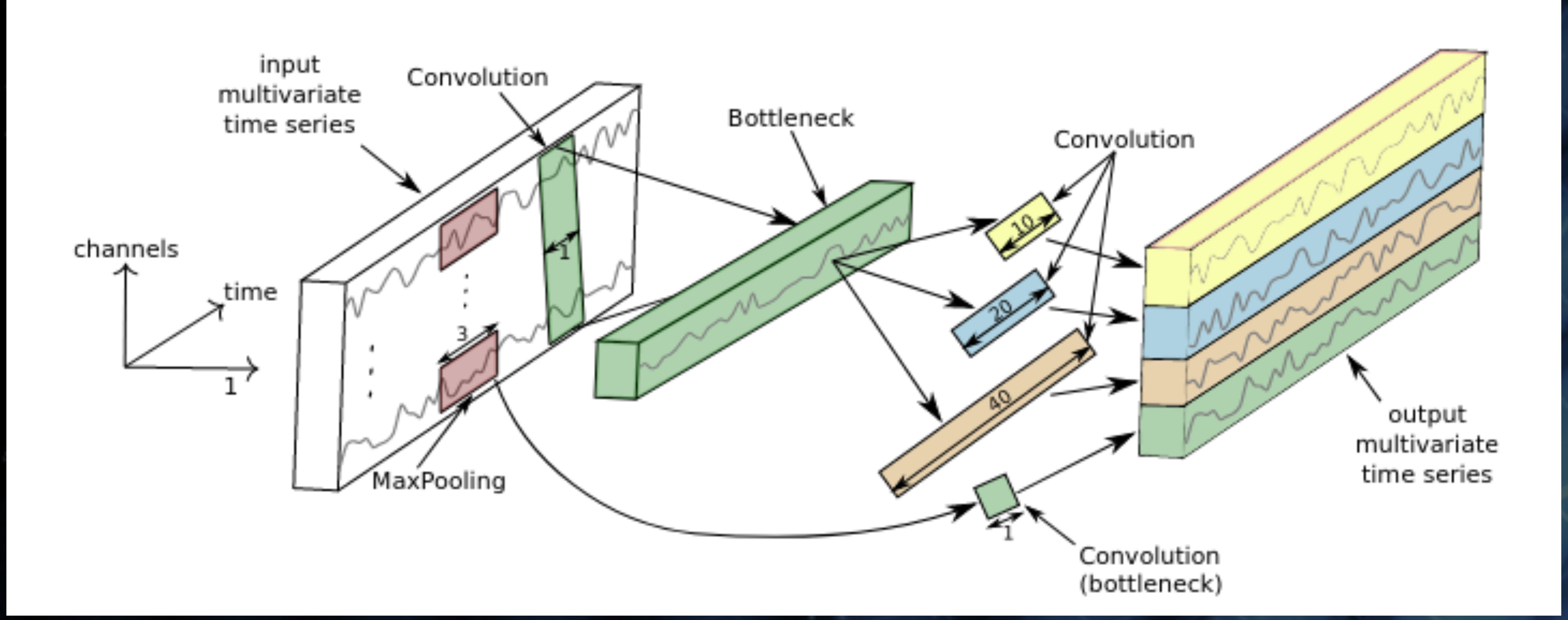
ROC curve (before activation)
Classification efficiency
False alarm probability
TCN_lin TCN_150epoch_logits
IT_RF_800_KS80_D10_F32_P1 IT20ep_logit_softmax

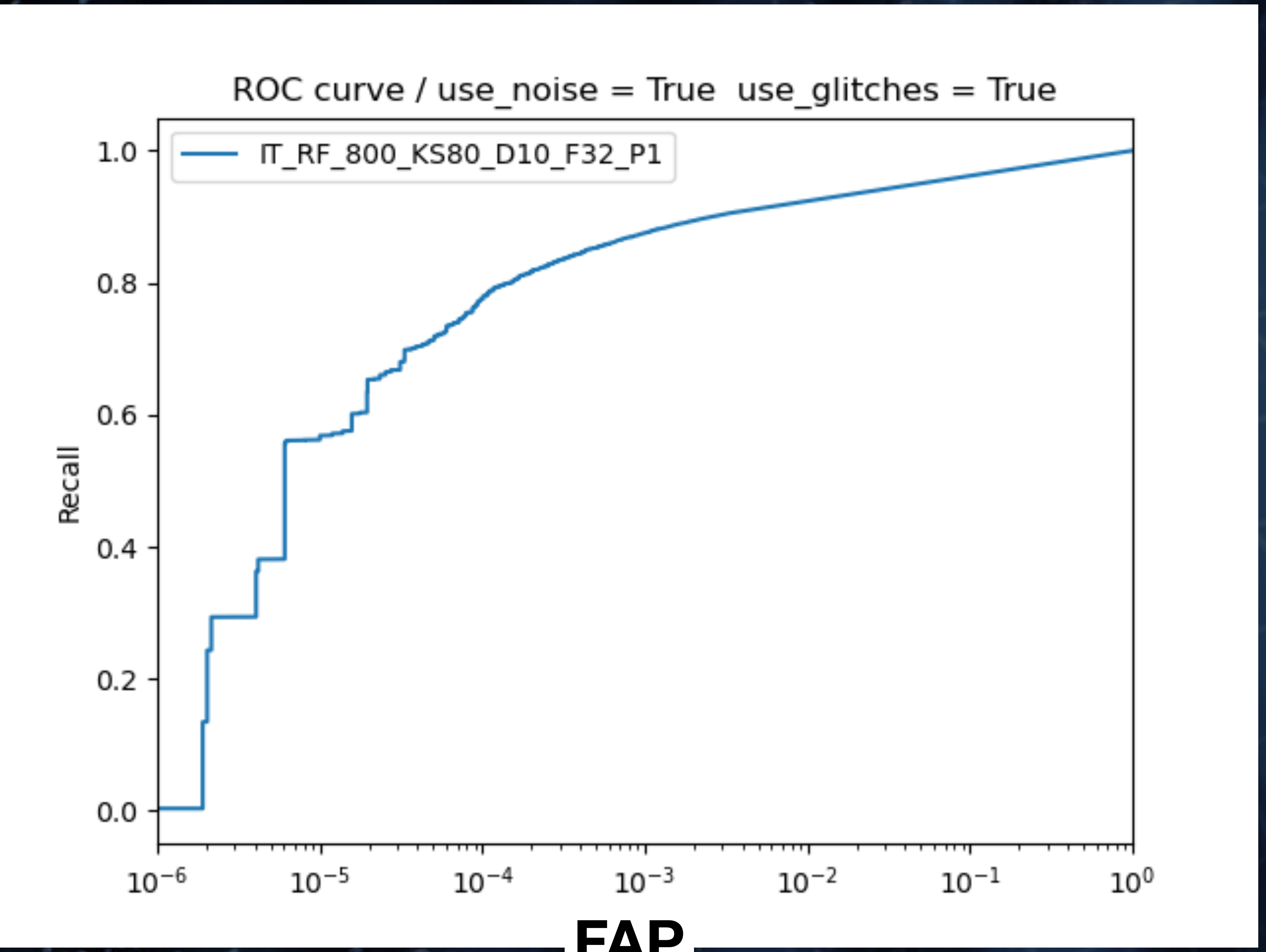
Conclusion

- GW signal classifier from single-detector time-series
  - ✓ FAR ~ few/month can be achieved
- Can noise rejection be improved further to reach 1/month?
  - ✓ investigating other architectures specialised for time-series
- Can we optimize the CNN with this objective specifically?
  - ✓ Working on alternative loss functions

Backup slides

Inception time





- **RF** is the receptive field. It is determined by the two following parameters, roughly my multiplication
- **KS** is the biggest kernel size in each module (InceptionTime uses kernels of different sizes at each step)
- **D** is the depth (number of modules)
- **F** is the number of filters for each kernel size with each module
- **P1** indicates that the model uses pooling after each residual connection, that is every 3 modules

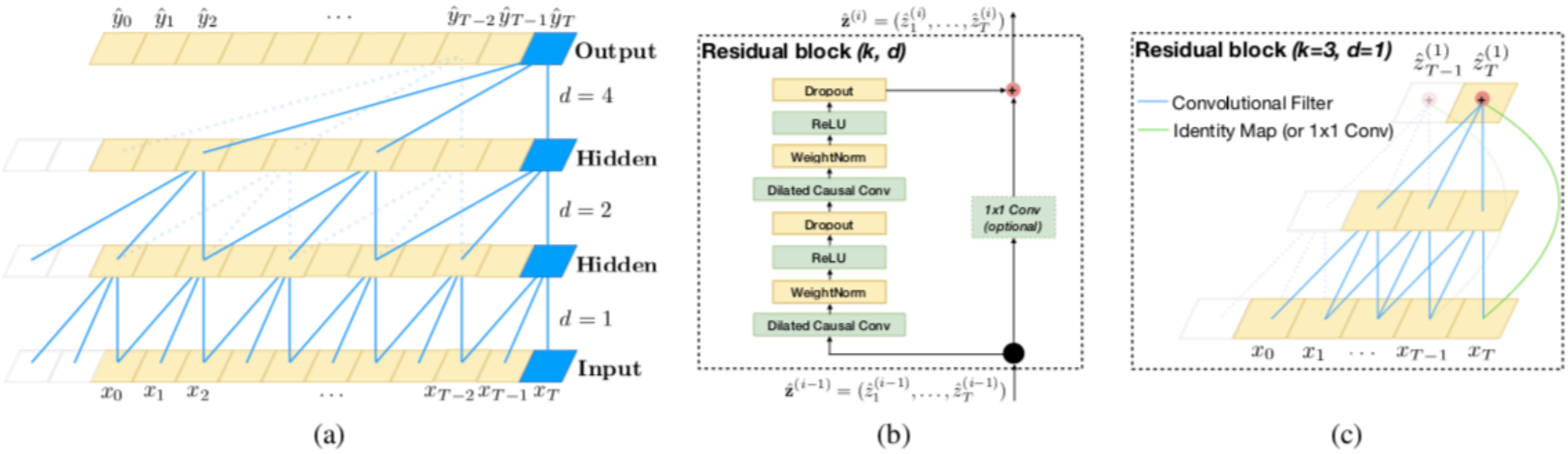


Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different 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