GLANCE

Better inspections at a glance

Matt MacDonald

Insight AI - 2019

In repetitive manufacturing there are many cyclical processes.

Regular inspections are common, but they are slow and expensive.

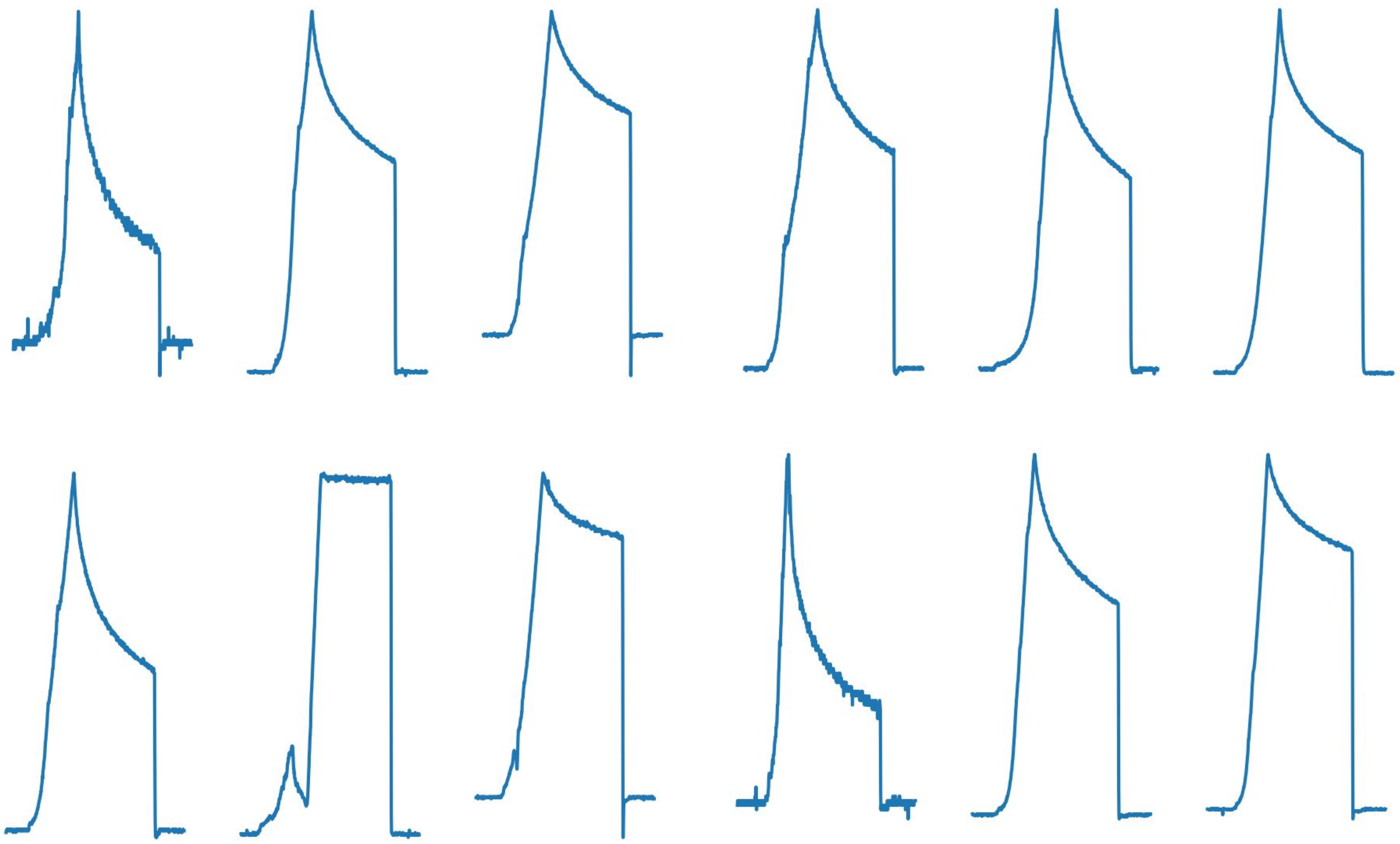
Can we do better?

- *Electronics*
- *Food products*
- *Automotive parts*
- *Appliances*
- *Consumer goods*
- *Oil and gas*
- *Mining equipment*
- *HVAC*
- *and many more..*

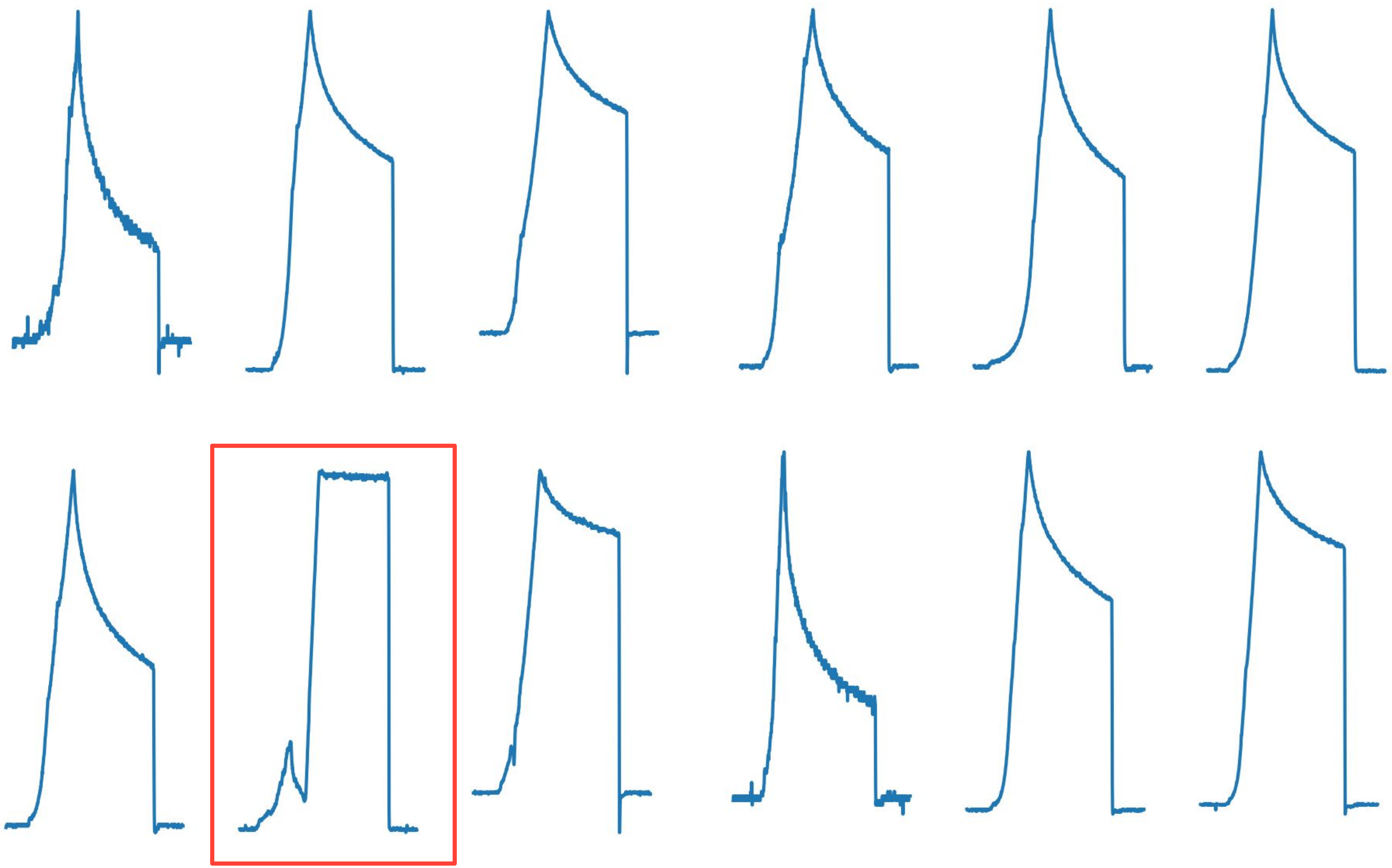
We want:

1. Automatic monitoring (not necessarily diagnosing!)
2. Easy setup and low cost
3. Work for *unknown* anomalies

Which is the anomaly?



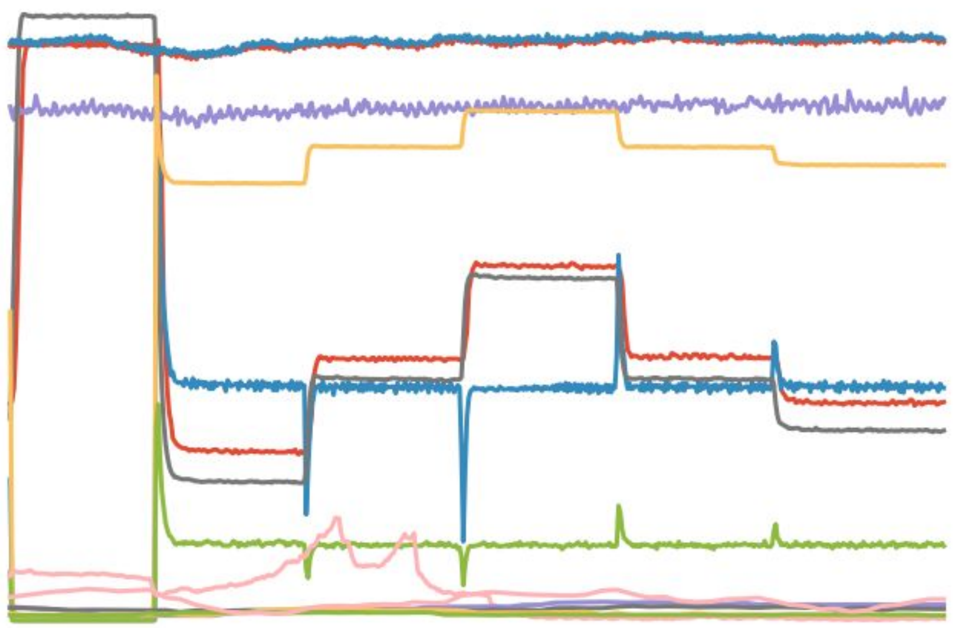
Easy right?



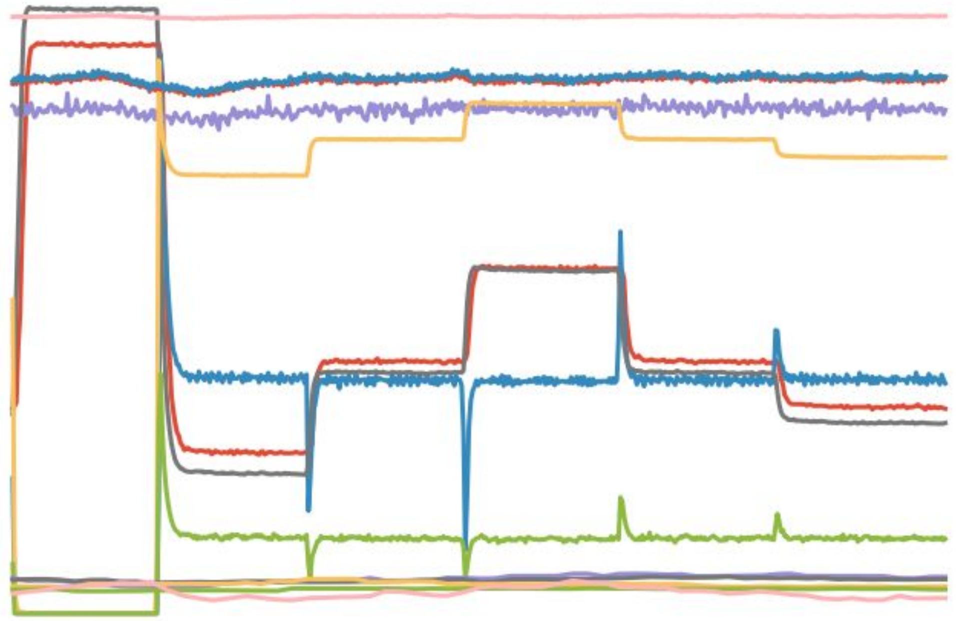
Not really...

The real world can be noisy and complex.

Close to total failure!!



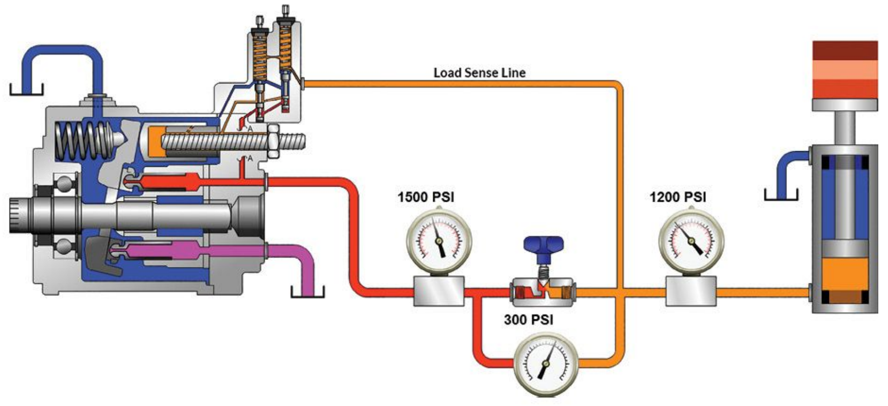
Perfectly normal :)



	LDA	ANN	SVM (linear)	SVM (RBF)
Cooler	100	100	100	100
Valve	100	100	100	95.7
Pump	73.6	80.0	72.4	64.2
Accumulator	54.0	50.4	51.6	65.7
Mean	**81.9**	**82.6**	**81.0**	**81.4**

66%

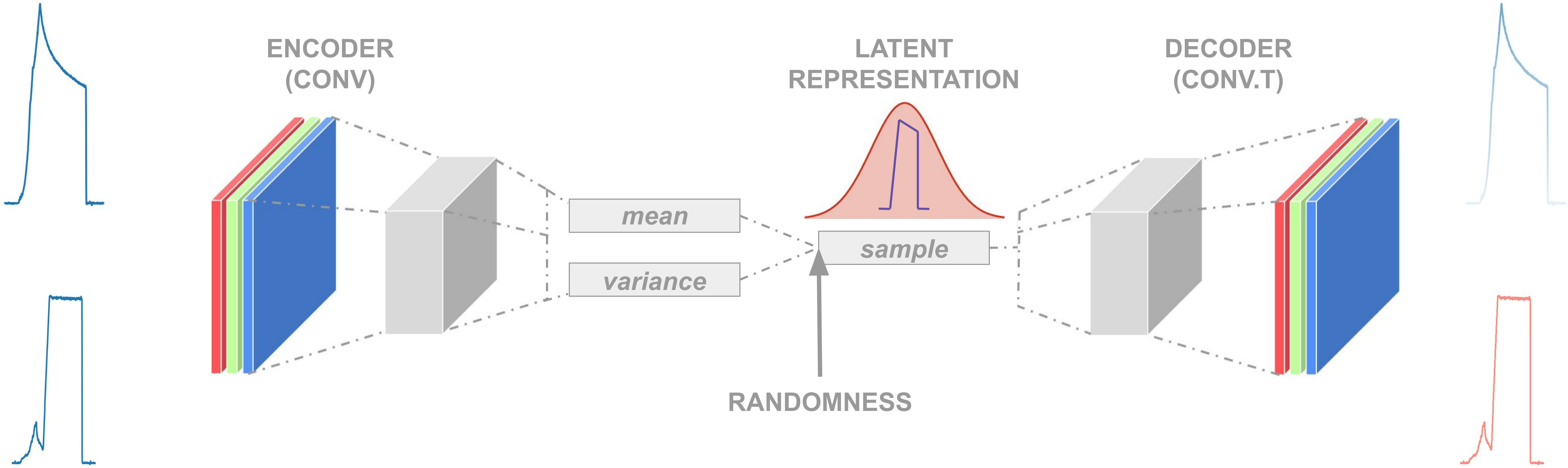
* May 2015 - https://ieeexplore.ieee.org/document/7151267



Hydraulic System

EPS1	Motor power	100 Hz	W
FS1	Volume flow	10 Hz	l/min
FS2	Volume flow	10 Hz	l/min
PS1	Pressure	100 Hz	bar
PS2	Pressure	100 Hz	bar
PS3	Pressure	100 Hz	bar
PS4	Pressure	100 Hz	bar
PS5	Pressure	100 Hz	bar
PS6	Pressure	100 Hz	bar
TS1	Temperature	1 Hz	C
TS2	Temperature	1 Hz	C
TS3	Temperature	1 Hz	C
TS4	Temperature	1 Hz	C
VS1	Vibration	1 Hz	mm/s

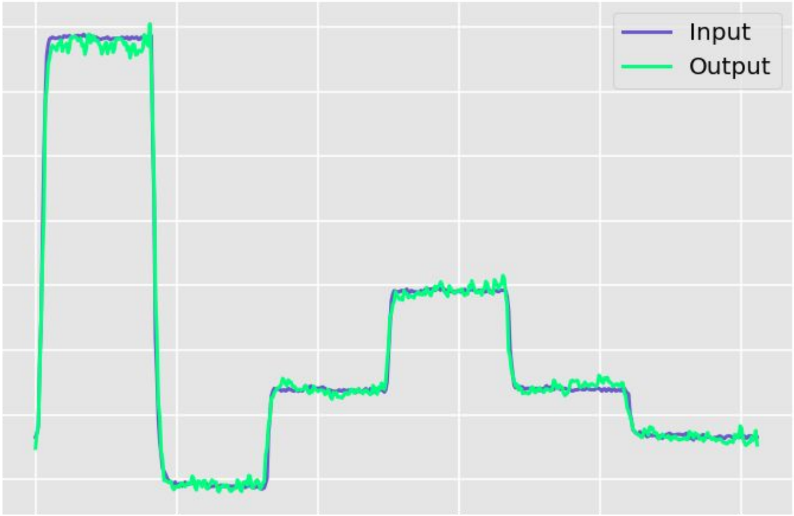
Recent research* suggests that convolutional variational autoencoders (VAE) can do this. The idea is to use the *error of the recreation* as a metric for detecting anomalies.

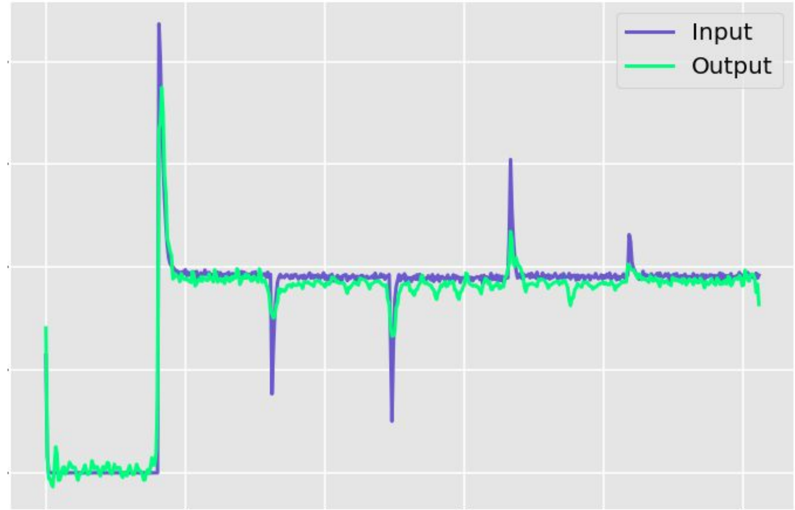


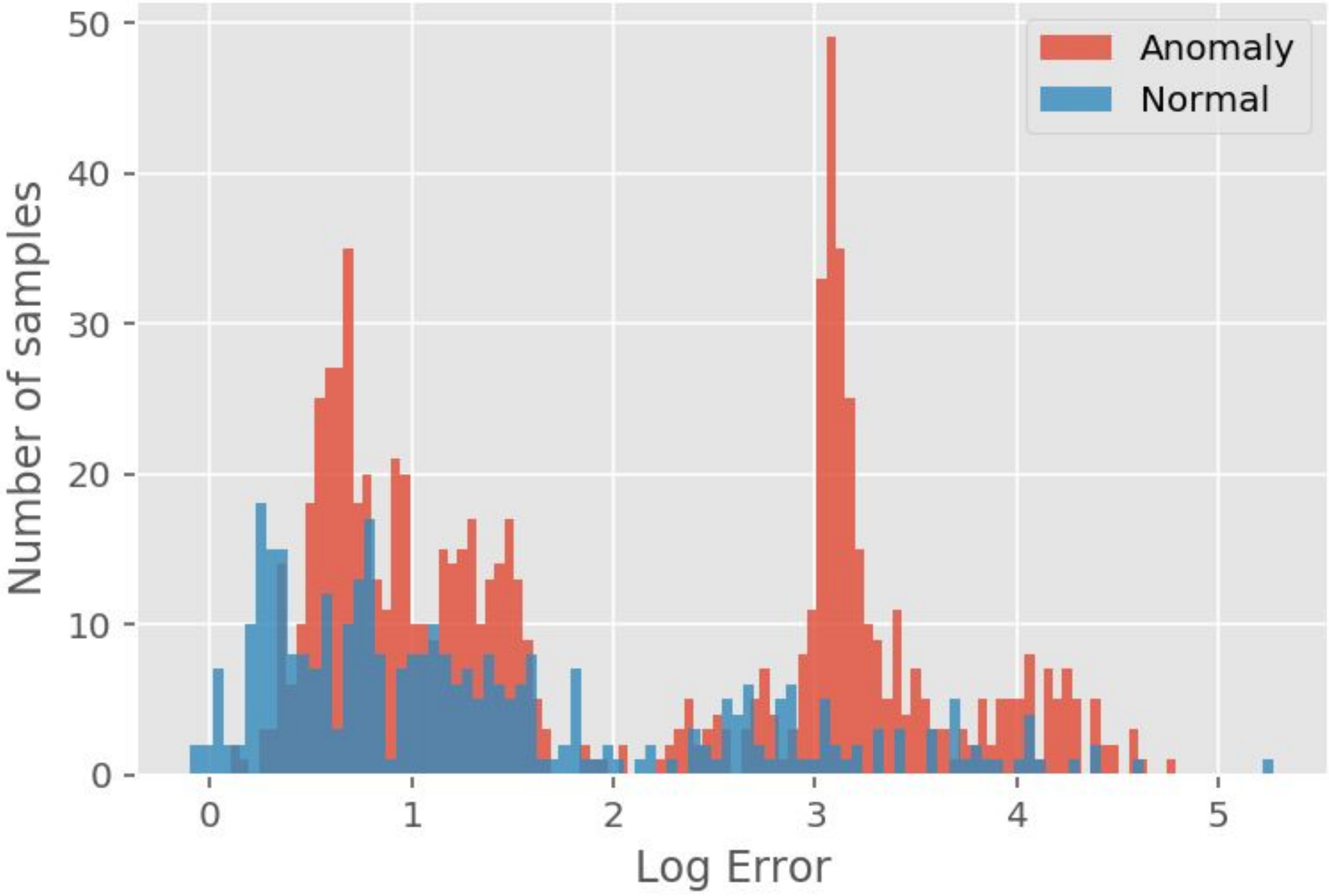
I adapted the approach from 2D images to 1D time-series sensor readings and trained it on the 14 sensors monitoring the hydraulic system.

*July 2018 - https://arxiv.org/pdf/1807.01349.pdf

The VAE can recreate input data well enough but using the error term to detect anomalies isn't all it's cracked up to be...

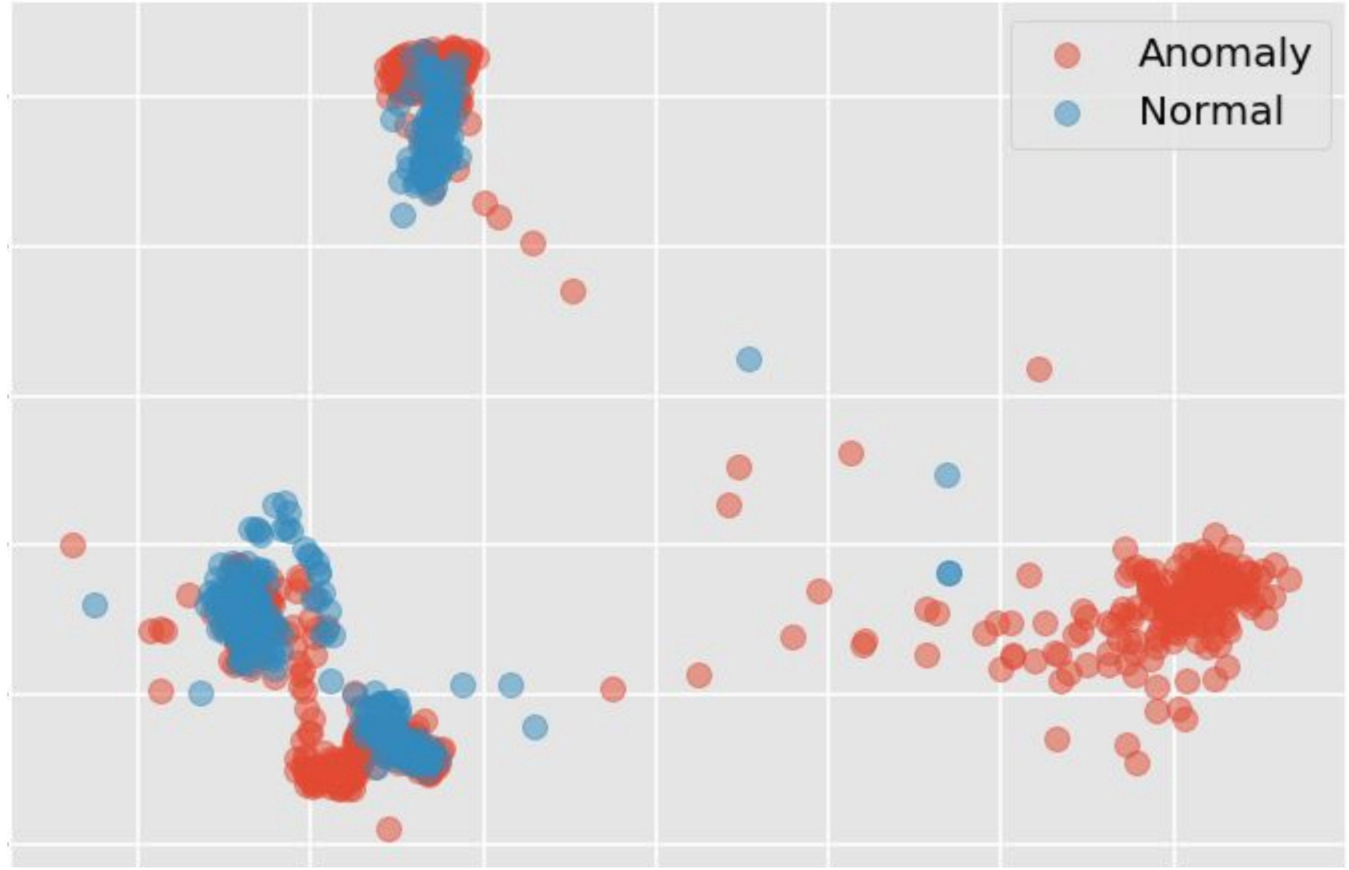


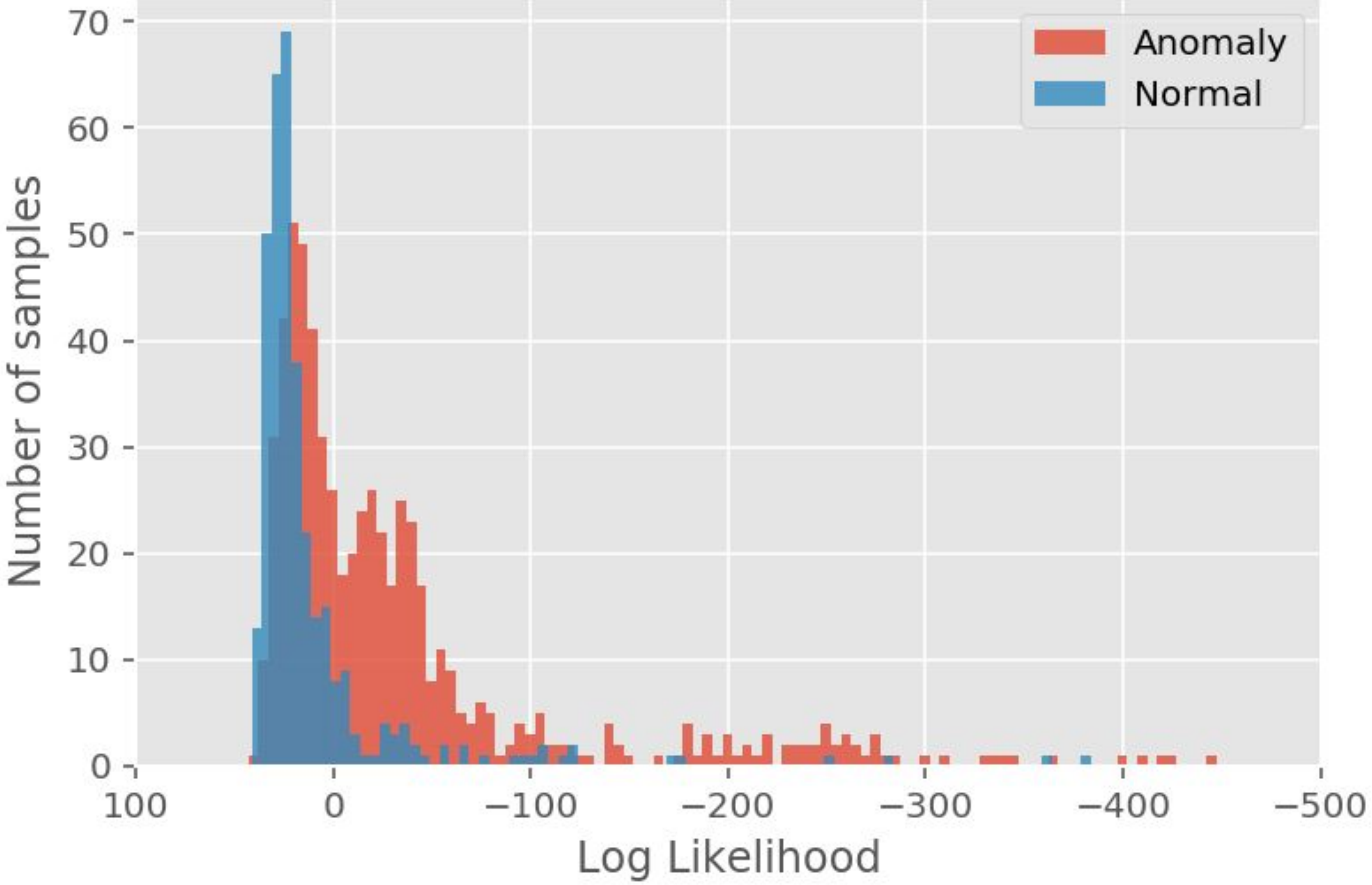




Accuracy = 53%
F1 score = 56%

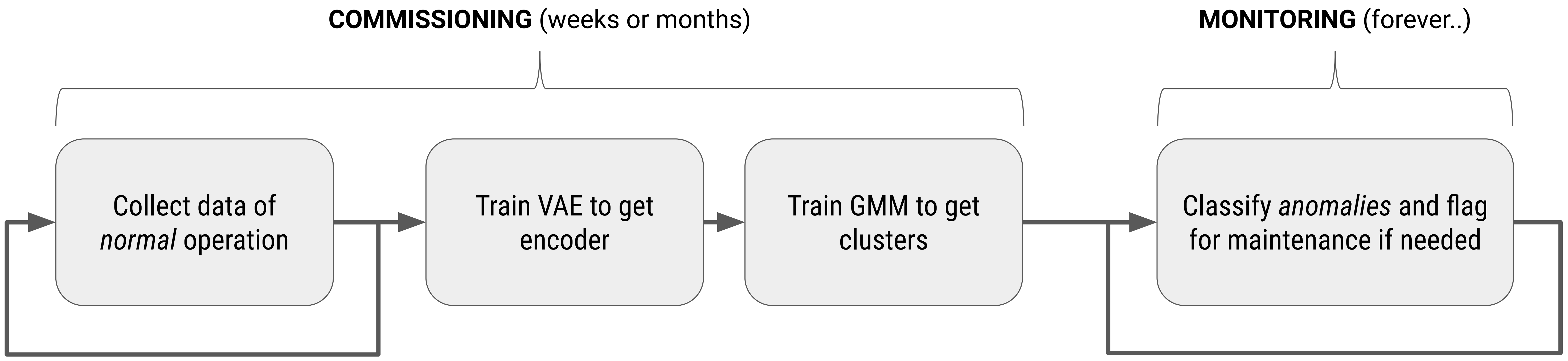
But the latent space has more information than a single number error term. Clustering using a Gaussian Mixture Model provides a much more meaningful outlier metric: <u>Likelihood</u>





Accuracy = 81%
F1 score = 87%

This approach improves performance over some of the best supervised methods (SVM) and it does it **without ever having to see an anomaly**.



Once running, access through a **dashboard.**

Who Am I?

Matt MacDonald

Professional engineer

Masters in Mechanical and Industrial Engineering from U of T

Worked on numerical analysis models and control systems for jet engines

Worked on software, products and research for surgical devices

Questions?

mattmacdc@gmail.com
github.com/ought

BACKUP

Home

GLANCE

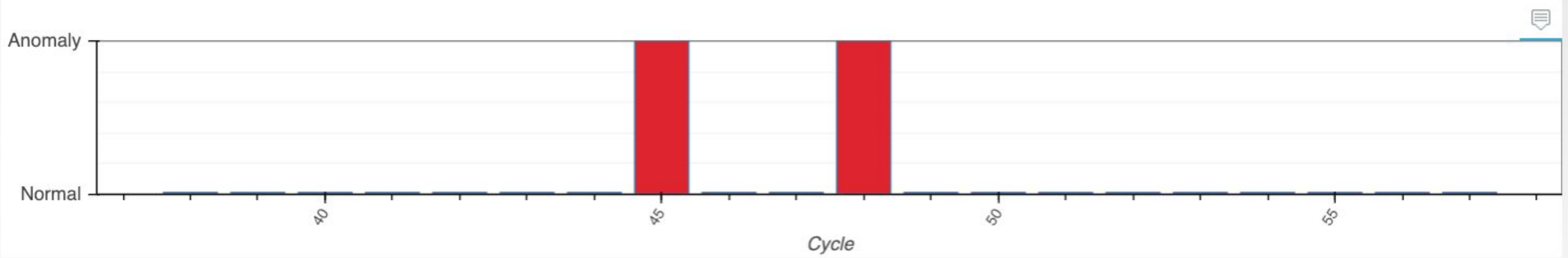
Better inspections at a glance

Monitoring Dashboard

Hydraulic-FX5

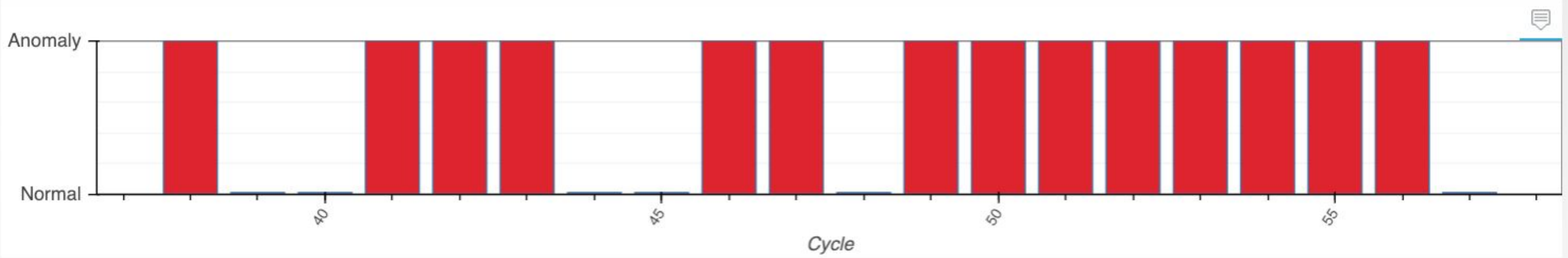
Reading from sensors: EPS1, FS1, FS2, PS1, PS2, PS3, PS4, PS5, PS6, TS1, TS2, TS3, TS4, VS1
Commissioned on 2019-01-30 (17 hr 30 min collected)

Normal operation



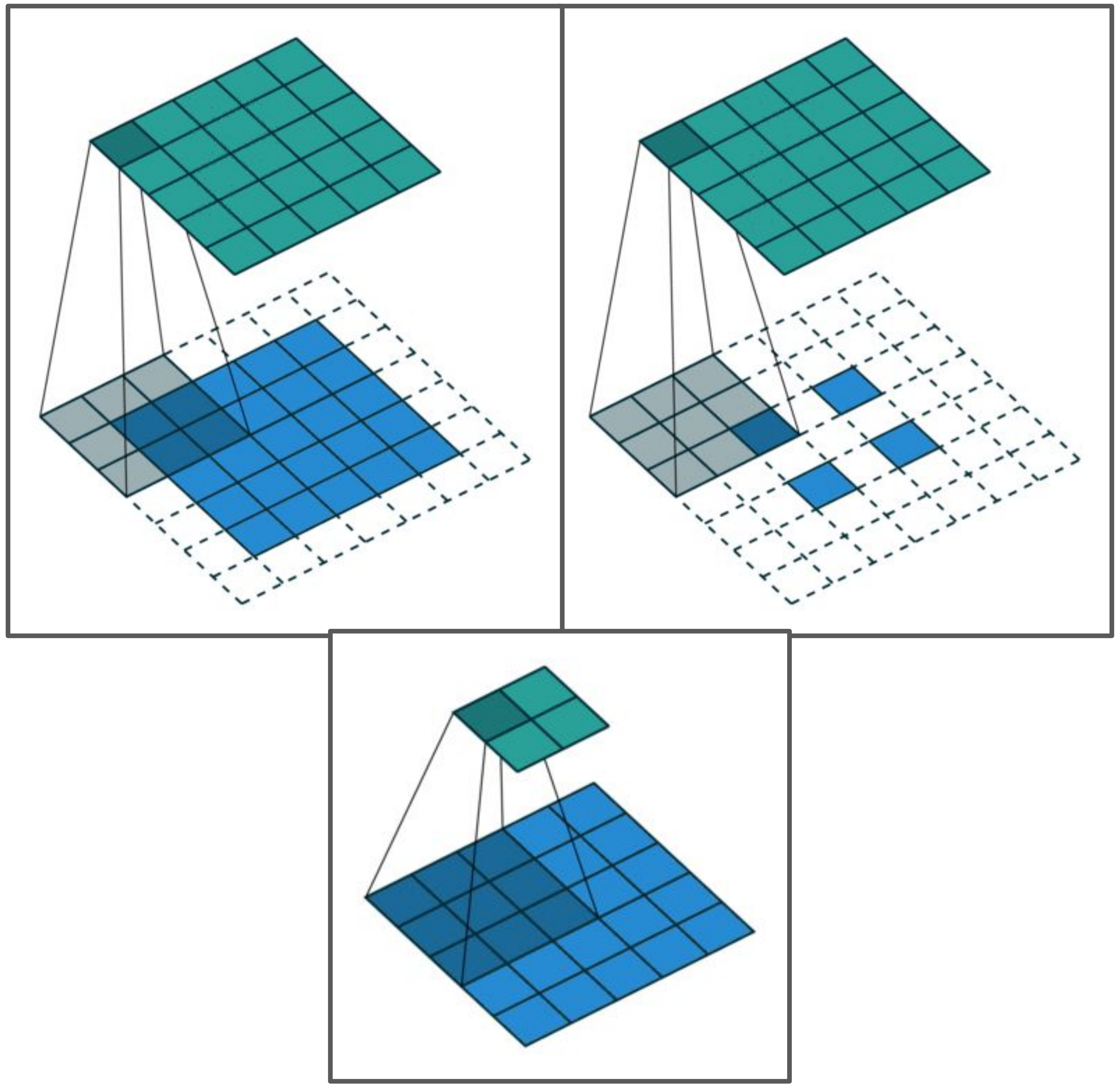
Faulty operation

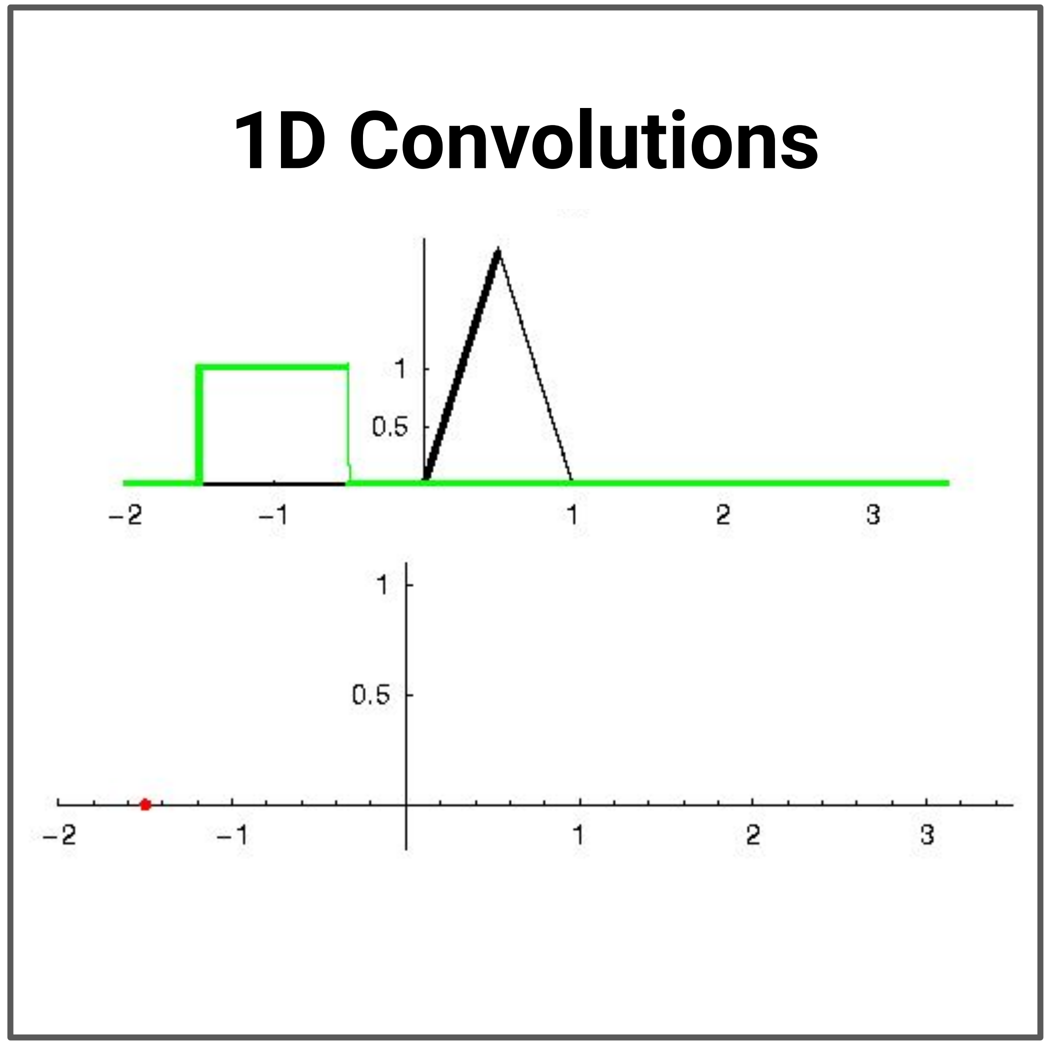
Danger! High anomaly rate detected. Maintenance required for installation Hydraulic-FX5.



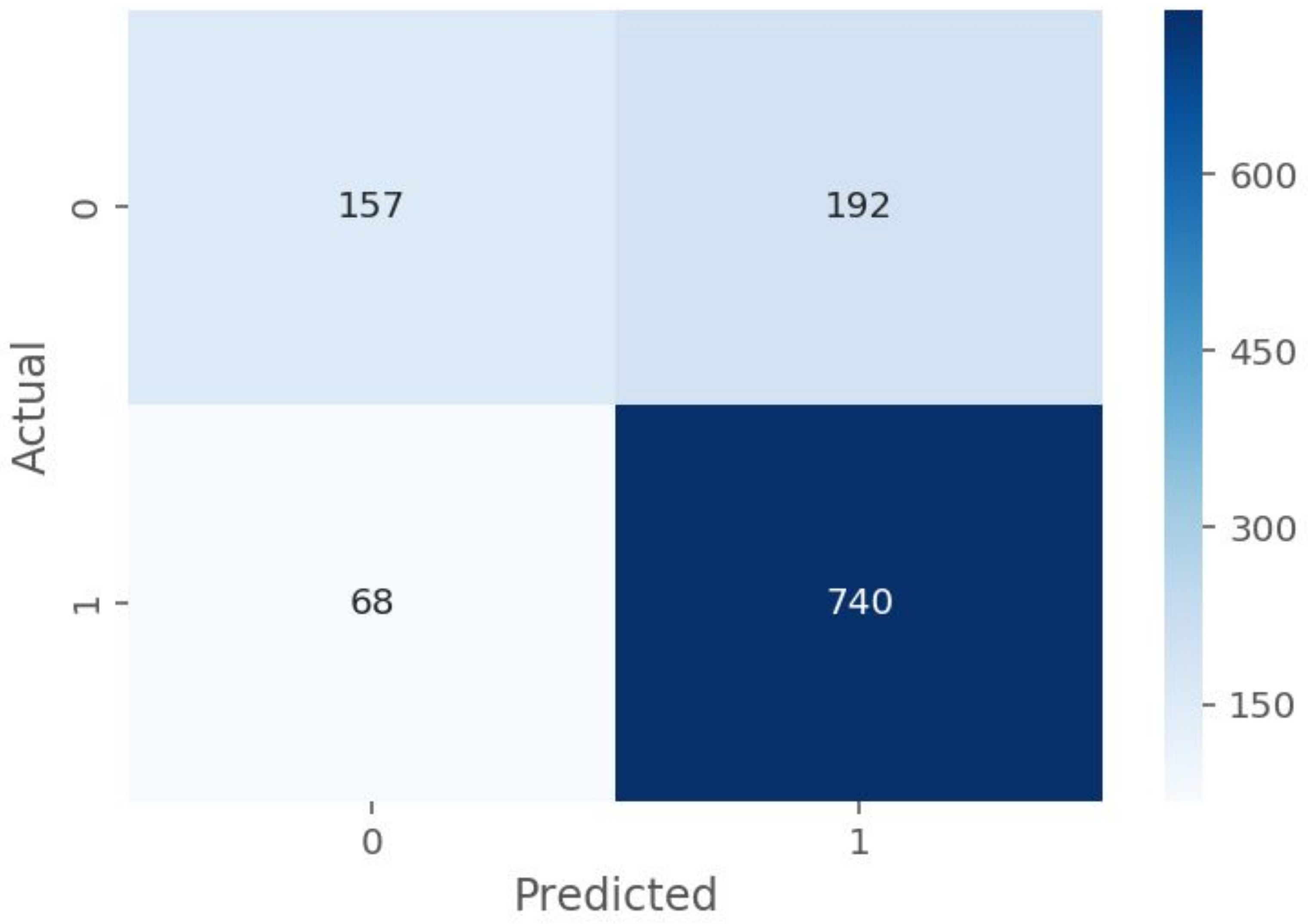
BACKUP

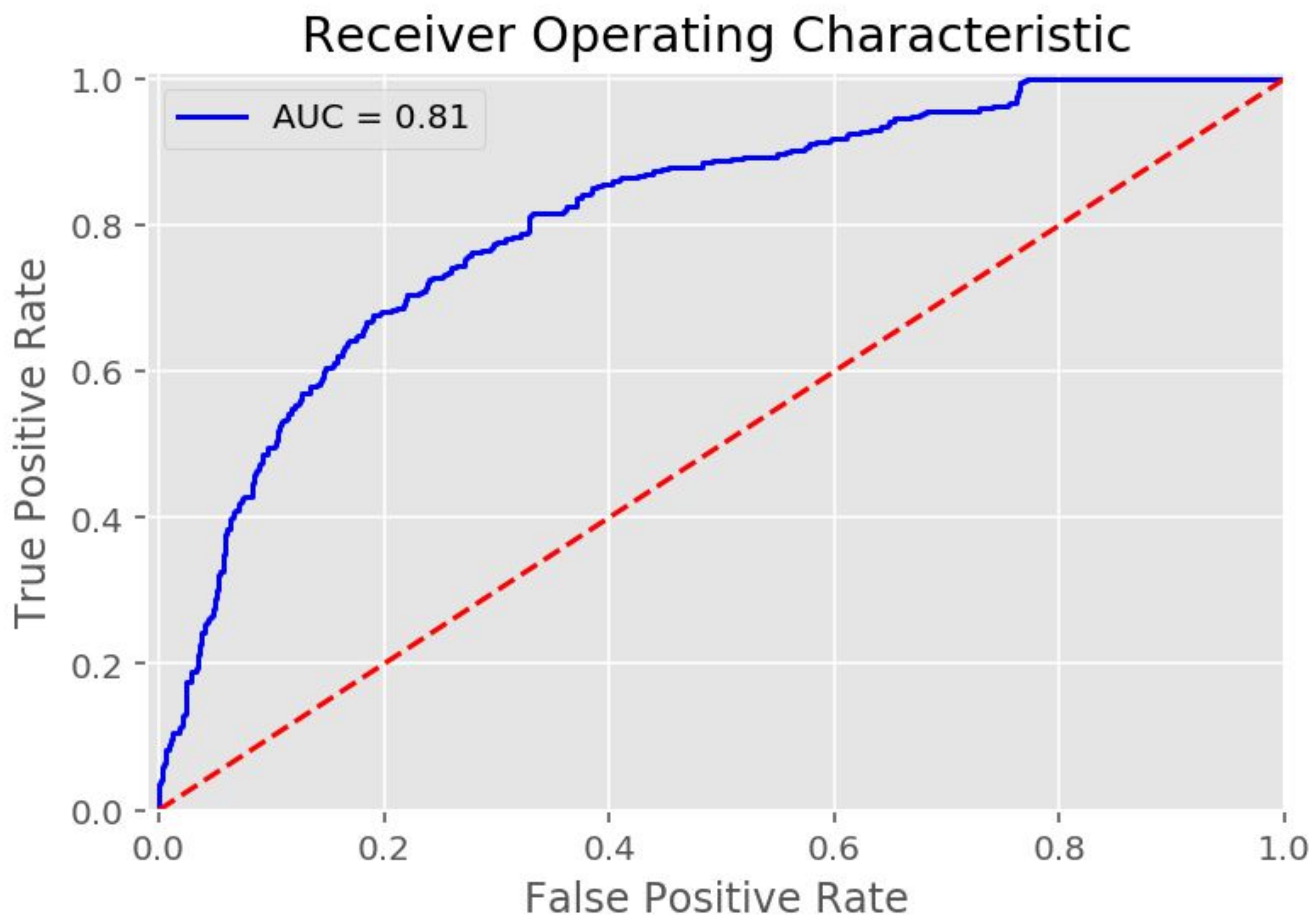
2D Convolutions





BACKUP


157
192
68
740
0
1
Actual
Predicted
600
450
300
150


Receiver Operating Characteristic
AUC = 0.81
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

BACKUP

VAE1D Architecture

Best results using:

- Learning rate cosine annealing
- No output activation
- Batch normalization and ReLU
- Equal KL divergence loss
- 7 convolutional layers
- 45 layers total
- 50 latent dimensions
- 300 epochs

```
VAE1D(
 (encoder): Sequential(
  (input-conv): Conv1d(14, 16, kernel_size=(4,), stride=(2,), padding=(1,))
  (input-relu): ReLU(inplace)
  (pyramid_16-32_conv): Conv1d(16, 32, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_32_batchnorm): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_32_relu): ReLU(inplace)
  (pyramid_32-64_conv): Conv1d(32, 64, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_64_batchnorm): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_64_relu): ReLU(inplace)
  (pyramid_64-128_conv): Conv1d(64, 128, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_128_batchnorm): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_128_relu): ReLU(inplace)
  (pyramid_128-256_conv): Conv1d(128, 256, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_256_batchnorm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_256_relu): ReLU(inplace)
  (pyramid_256-512_conv): Conv1d(256, 512, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_512_batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_512_relu): ReLU(inplace)
  (pyramid_512-1024_conv): Conv1d(512, 1024, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_1024_batchnorm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_1024_relu): ReLU(inplace)
 )
 (conv_mu): Conv1d(1024, 100, kernel_size=(4,), stride=(1,))
 (conv_logvar): Conv1d(1024, 100, kernel_size=(4,), stride=(1,))
 (decoder): Sequential(
  (input-conv): ConvTranspose1d(100, 1024, kernel_size=(4,), stride=(1,))
  (input-batchnorm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (input-relu): ReLU(inplace)
  (pyramid_1024-512_conv): ConvTranspose1d(1024, 512, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_512_batchnorm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_512_relu): ReLU(inplace)
  (pyramid_512-256_conv): ConvTranspose1d(512, 256, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_256_batchnorm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_256_relu): ReLU(inplace)
  (pyramid_256-128_conv): ConvTranspose1d(256, 128, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_128_batchnorm): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_128_relu): ReLU(inplace)
  (pyramid_128-64_conv): ConvTranspose1d(128, 64, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_64_batchnorm): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_64_relu): ReLU(inplace)
  (pyramid_64-32_conv): ConvTranspose1d(64, 32, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_32_batchnorm): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_32_relu): ReLU(inplace)
  (pyramid_32-16_conv): ConvTranspose1d(32, 16, kernel_size=(4,), stride=(2,), padding=(1,))
  (pyramid_16_batchnorm): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pyramid_16_relu): ReLU(inplace)
  (output-conv): ConvTranspose1d(16, 14, kernel_size=(4,), stride=(2,), padding=(1,))
 )
)
```

7 (+ 2) convolutional encoding layers

1 variational sampling layer

7 (+ 1) transpose convolutional decoding layers