ICSC
Centro Nazionale di Ricerca in HPC,
Big Data and Quantum Computing

INFN
Istituto Nazionale di Fisica Nucleare

National Centre for HPC, Big Data and Quantum Computing – Spoke2 – WP2

Anomaly detection for Muon DQM/DC

Federica M. Simone
Dipartimento Interateneo di Fisica & INFN Bari

federica.simone@poliba.it, federica.simone@ba.infn.it

PNRR milestones

M9-M15	Landscape recognition of the state-of-the-art and technological investigation on the opportunity of the CN infrastructure - report submitted with detailed plan of work and selection of specific case studies.
M22-M26	Report on first implementations and tests.
M25-M36	Results from testbed and benchmarking activities; final report and evaluation.

Autoencoder

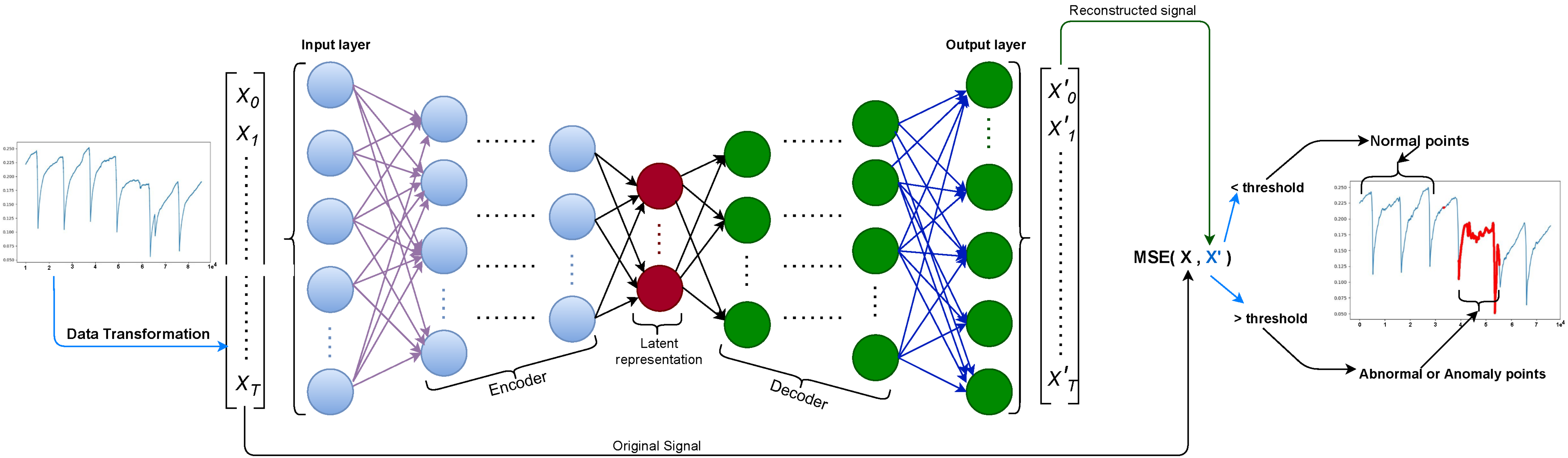
Architecture: artificial neural network

- Encoder: transforms input data in some enconded representation
- Decoder: recreates input data

Comparing the original input with the decoded data allows detecting anomalies.

Unsupervised learning: no need for labelled data.

Assumption: most of the data is good and anomalies are rare



Step1: load data (DQM histograms) and filter

```
In [2]: ### read the data
        # note: this cell assumes you have a csv file stored at the specified location,
        #       containing only histograms of the specified type;
        #       see the tutorial read_and_write_data for examples on how to create such files!

        histname = 'FEDTotalEventSize'
        filename = 'nanodqmio_2023C_Muon0_'+histname+'_mod.csv'
        datadir = '/eos/user/f/fsimone/auto_DQM/output_nanodqm/'

        dloader = DataLoader.DataLoader()
        df = dloader.get_dataframe_from_file( os.path.join(datadir, filename) )
        print('raw input data shape: {}'.format( dfu.get_hist_values(df)[0].shape ))
```

```
INFO in DataLoader.get_dataframe_from_file: loading dataframe from file /eos/user/f/fsimone/auto_DQM/output_nanodq
m/nanodqmio_2023C_Muon0_FEDTotalEventSize_mod.csv...
INFO in DataLoader.get_dataframe_from_file: sorting the dataframe...
INFO in DataLoader.get_dataframe_from_file: loaded a dataframe with 2263 rows and 13 columns.
raw input data shape: (2263, 102)
```

```
In [3]: ### filtering: select only DCS-bit on data and filter out low statistics

        df = dfu.select_dcson(df)
        print('number of passing lumisections after DCS selection: {}'.format( len(df) ))

        df = dfu.select_highstat(df, entries_to_bins_ratio=10)
        print('number of passing lumisections after high statistics selection: {}'.format( len(df) ))
```

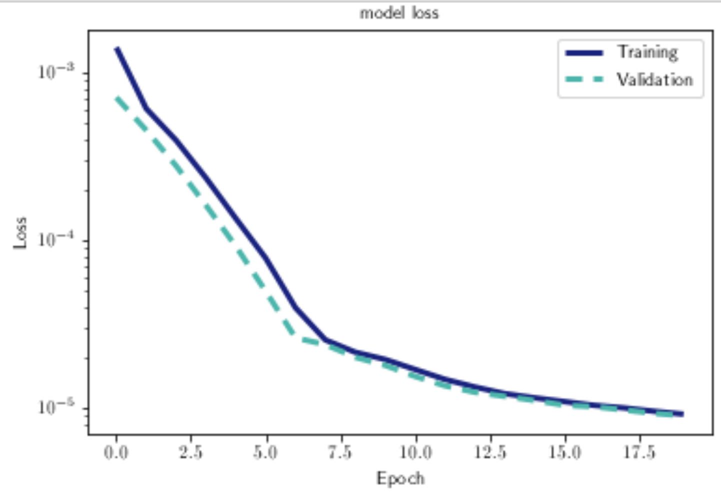
```
number of passing lumisections after DCS selection: 1943
number of passing lumisections after high statistics selection: 1943
```

+ rebinning, re-shaping if needed

Step2: build the model and train it

```
### build the model and train it
from keras.layers import LeakyReLU

    # - input_size: size of vector that autoencoder will operate on
    # - arch: list of number of nodes per hidden layer (excluding input and output layer)
    # - act: list of activations per layer (default: tanh)
    # - opt: optimizer to use (default: adam)
    # - loss: loss function to use (defualt: mseTop10)

    # - mseTop10: mean squared error between y_true and y_pred,
    #   where only the 10 bins with largest squared error are taken into account.

input_size = X_train.shape[1]
arch = [int(X_train.shape[1]/2.)]
act = ['tanh']*len(arch)
opt = 'adam'
loss = aeu.mseTop10
autoencoder = aeu.getautoencoder(input_size,arch,act,opt,loss)
history = autoencoder.fit(X_train, X_train, epochs=20, batch_size=500, shuffle=False, verbose=1, validation_split=0.
pu.plot_loss(history, title = 'model loss')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 51)                5253      
                                                                 
 dense_1 (Dense)             (None, 102)               5304      
                                                                 
=================================================================
Total params: 10,557
Trainable params: 10,557
Non-trainable params: 0
```



Step2: build the model

```
### build the model and train it
from keras.layers import LeakyReLU

    # - input_size: size of vector that autoen
    # - arch: list of number of nodes per hidd
    # - act: list of activations per layer (de
    # - opt: optimizer to use (default: adam)
    # - loss: loss function to use (defualt: m

    # - mseTop10: mean squared error between y
    #   where only the 10 bins with largest sq

input_size = X_train.shape[1]
arch = [int(X_train.shape[1]/2.)]
act = ['tanh']*len(arch)
opt = 'adam'
loss = aeu.mseTop10
autoencoder = aeu.getautoencoder(input_size,ar
history = autoencoder.fit(X_train, X_train, ep
pu.plot_loss(history, title = 'model loss')
```

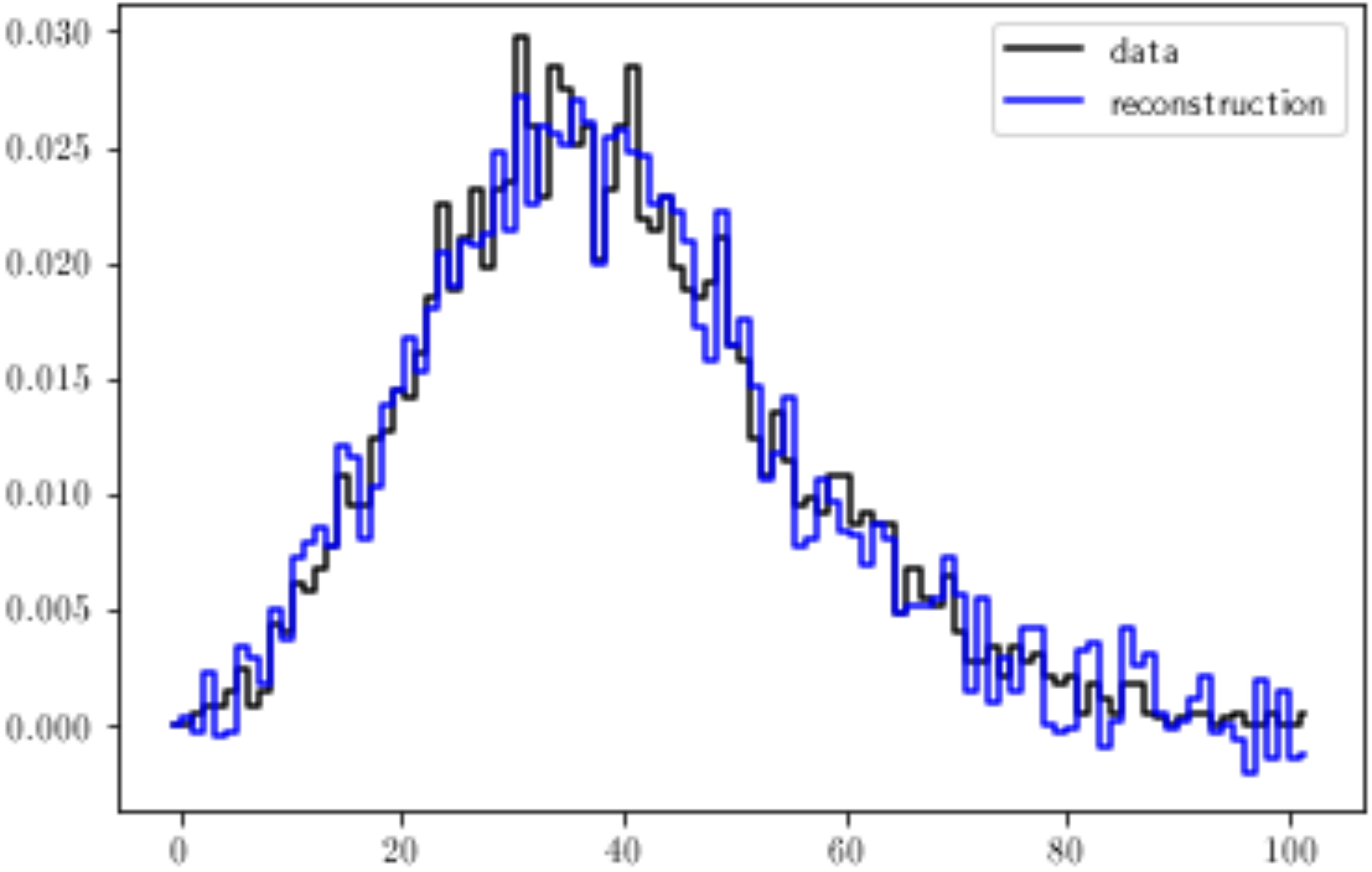
Documentation:

https://www.tensorflow.org/tutorials/generative/autoencoder

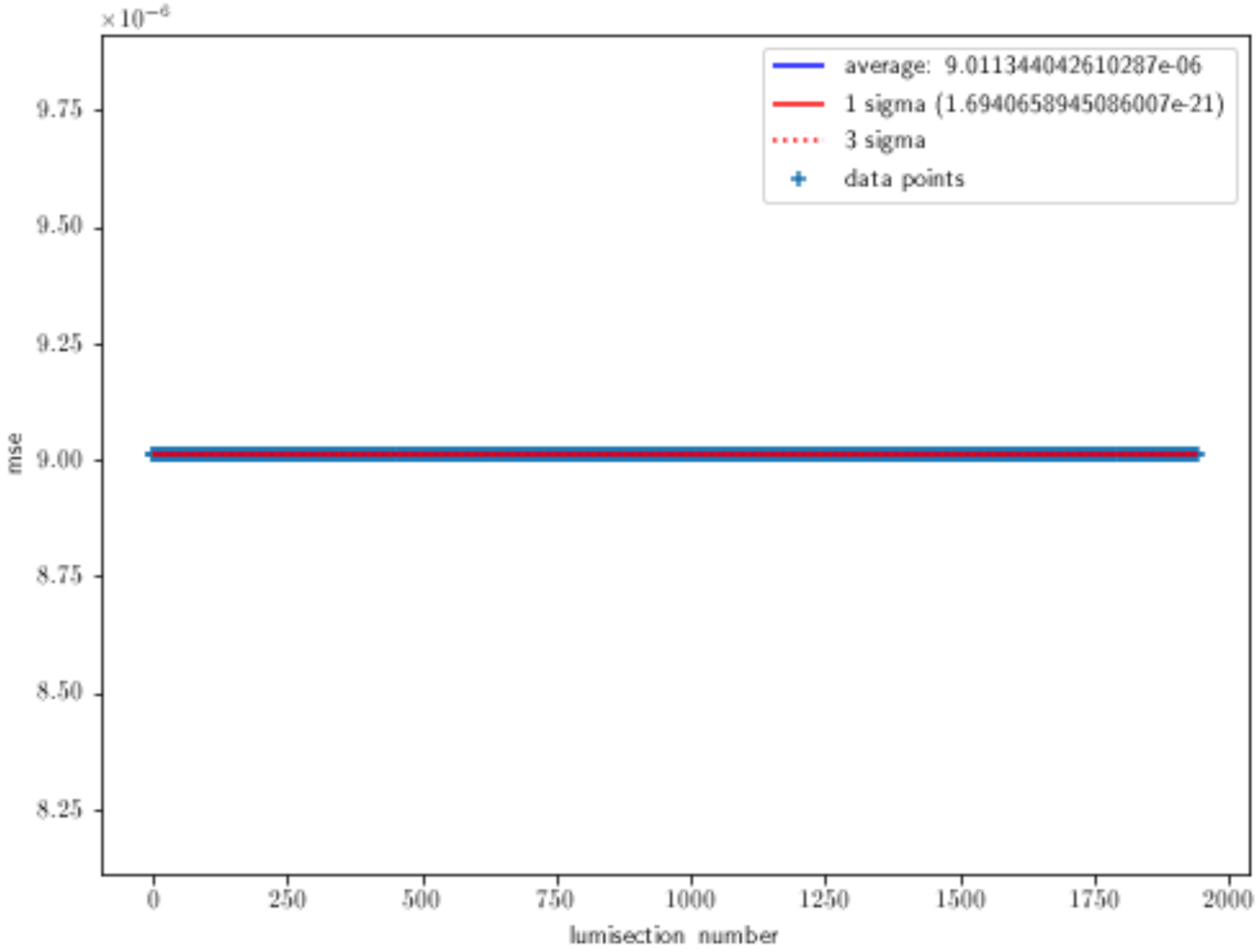
```
=================================================
Total params: 10,557
Trainable params: 10,557
Non-trainable params: 0
```

```
### getting a keras model ready for training with minimal user inputs

def getautoencoder(input_size,arch,act=[],opt='adam',loss=mseTop10):

    import math
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
    from tensorflow.keras.layers import Input, Dense
    from keras.layers import PReLU
    #from keras.layers.advanced_activations import PReLU
    from tensorflow.keras.models import Model, Sequential, load_model
    from keras import backend as K

    if len(act)==0: act = ['tanh']*len(arch)
    layers = []
    # first layer manually to set input_dim
    layers.append(Dense(arch[0],activation=act[0],input_dim=input_size))
    # rest of layers in a loop
    for nnodes,activation in zip(arch[1:],act[1:]):
        layers.append(Dense(nnodes,activation=activation))
    # last layer is decoder
    layers.append(Dense(input_size,activation='tanh'))
    autoencoder = Sequential()
    for i,l in enumerate(layers):
        #l.name = 'layer_'+str(i)
        autoencoder.add(l)
    autoencoder.compile(optimizer=opt, loss=loss)
    autoencoder.summary()
    return autoencoder
```

Step3: look at the reconstructed (decoded) distributions



Step4: define threshold on MSE(x,x')




```
mean mse: 9.011344042610287e-06
std mse: 1.6940658945086007e-21
```

You would typically set a cut value of mean+3std.

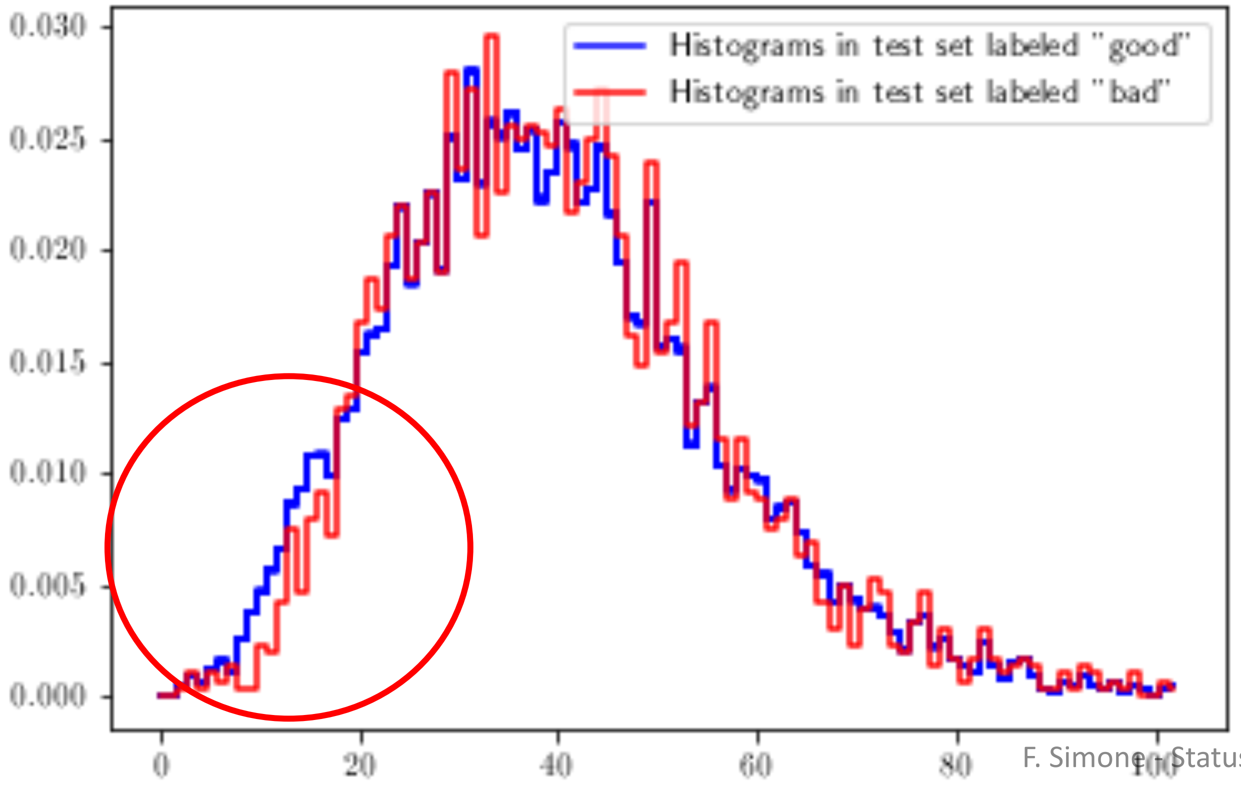
In this case, the MSE distribution is way too flat (std too small) → all histograms are good and no anomaly has been found.

… Let's add anomalous histograms "by hand" →

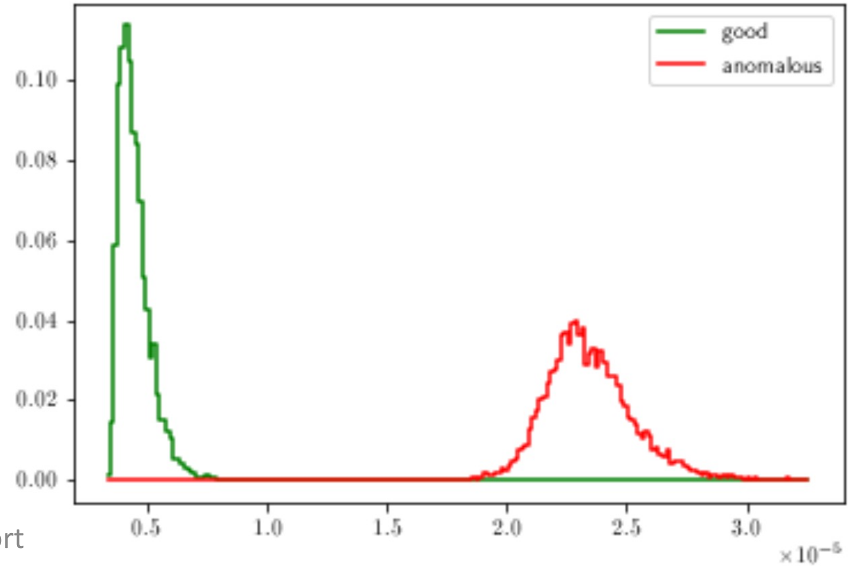
Evaluation of labelled dataset

```
INFO in DataLoader.get_dataframe_from_file: loading dataframe from file /eos/user/f/fsimone/auto_DQM/output_nanodqm/nanodqmio_2023C_Muon0_FEDTotalEventSize_mod.csv...
INFO in DataLoader.get_dataframe_from_file: sorting the dataframe...
INFO in DataLoader.get_dataframe_from_file: loaded a dataframe with 2263 rows and 13 columns.
shape of good test set: (15, 102)
shape of bad test set: (1, 102)
```

```
average mse on good set: 3.791672874866125 6e-06
average mse on bad set: 2.250084308584567e-05
```



In this case, MSE is potentially sensitive to anomaly!



Workflow and to-dos

- **Input**: set of 1-D or 2-D plots (monitoring elements, ME) specific for the muon system
  - occupancies, DAQ flags
  - mix good and problematic runs/lumis (each run contains O(1k) lumisections)
  - might need to resample the histograms (removing outliers)

- **Single training:** train one network for each ME

- **Composite training:** fed the autoencoder outputs into one neural network

- **Evaluation:** compare model output with labelled data

To-dos:

- define large training dataset for my use-case, possibly with labelled anomalies for performance evaluation

- implement notebook on HPC test-bed