ML for time resolution of MuX HPGe detector

Techniques for optimizing the timing resolution of HPGe detectors

ELET (Extrapolated Leading-Edge timing) algorithms are using for improving the time resolution of the HPGe detectors;

Cons:

- **Single set of parameters** for predefined function for whole energy range
- **Manual optimization**

Another possible approach: **Deep learning**

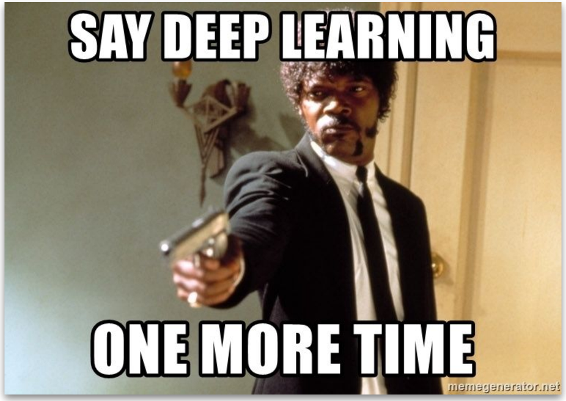
Techniques for optimizing the timing resolution of HPGe detectors

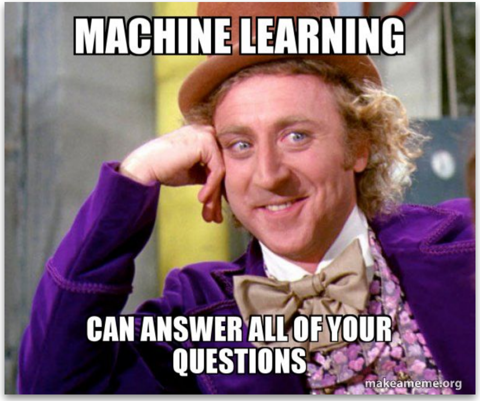
Deep learning

- learning relationship/correlation between signal shape and time of signal rising (t0)

Pros:

- Automated process
- More generalized approach: the builded network can be used for different different detectors with different electrical characteristics (gain, etc.)

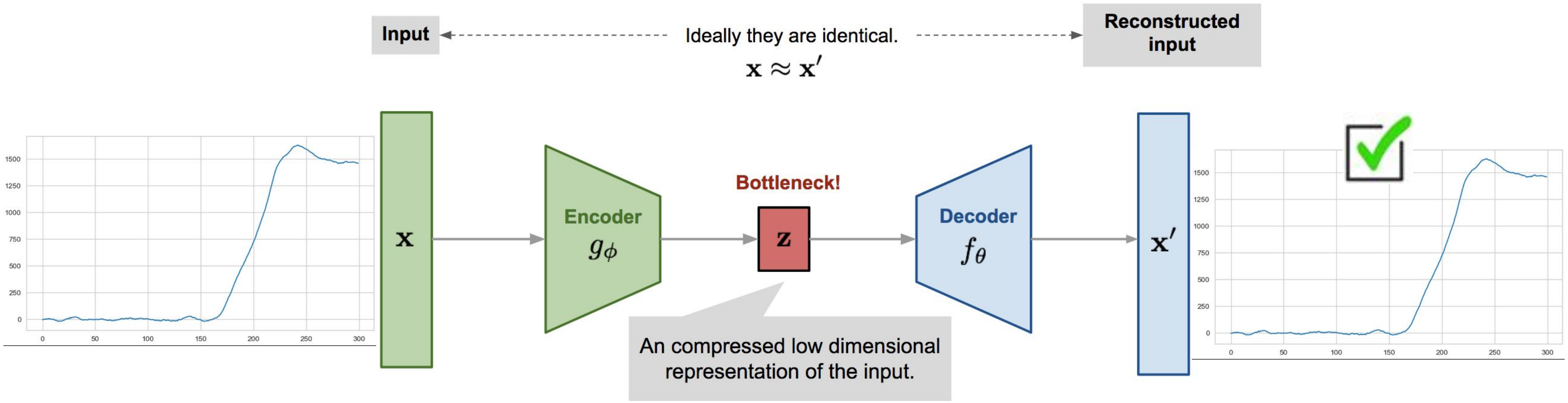




The idea

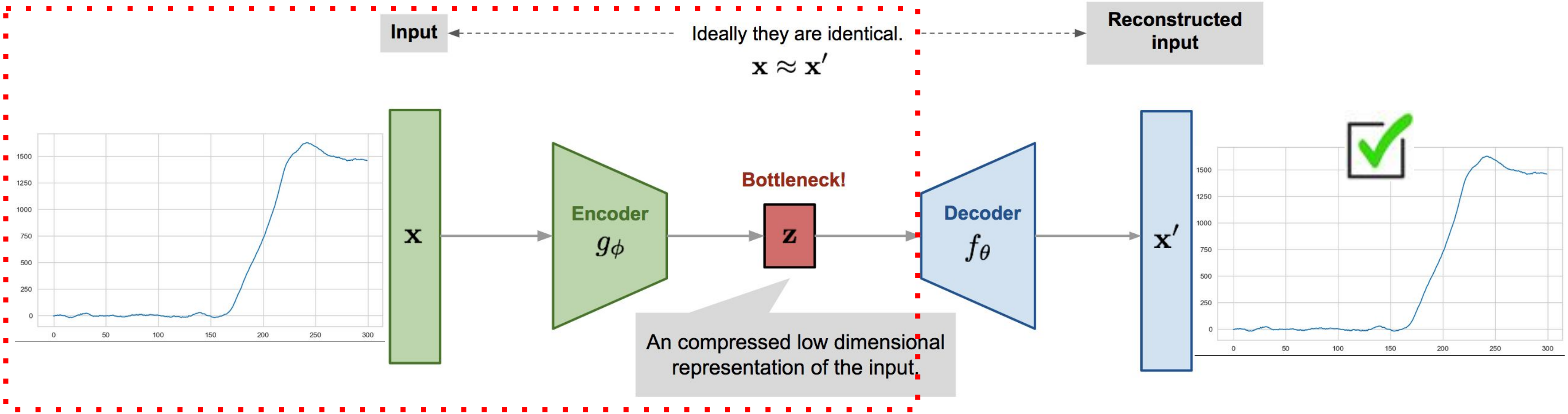
(based on Alex Skawran (PSI) thesis/draft,
with couple of valuable advices from Dr. Jean-Roch Vilmant, Caltech)

- Autoencoder CNN at first stage:



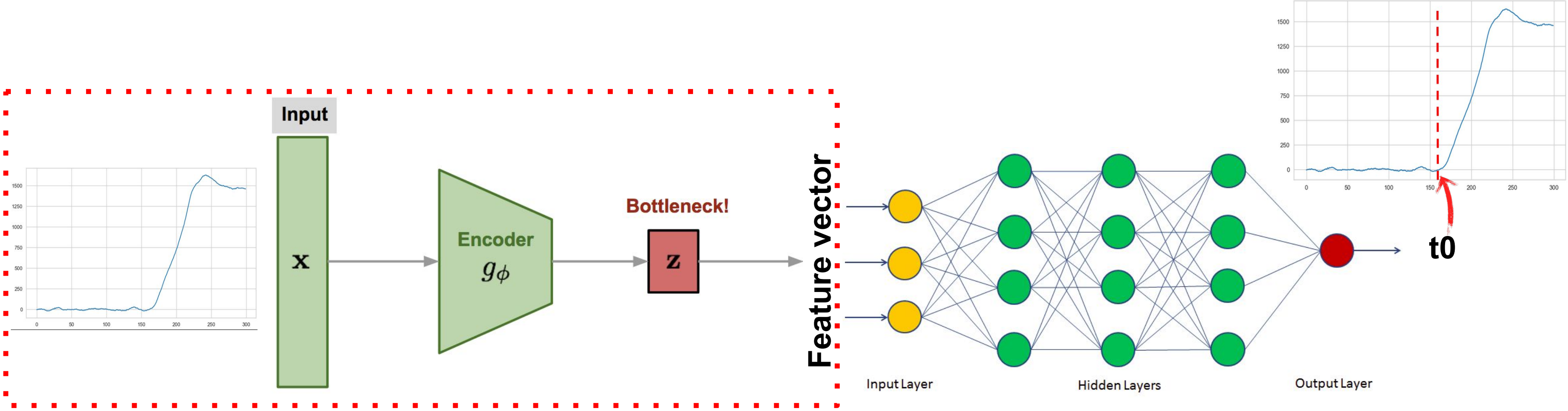
The idea

- Take pre-trained Encoder part...



The idea

- ...and add the NN with dense layers (w/ fully connected neurons):



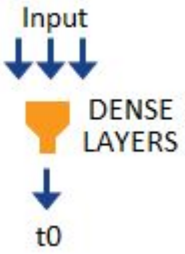
Pre-trained Encoder → feature vector (“compressed” representation) → t0 detecting ANN

Why Encoder + time_det structure?

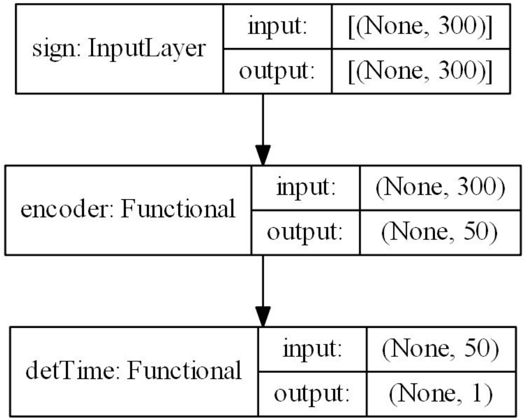
Just Dense layers

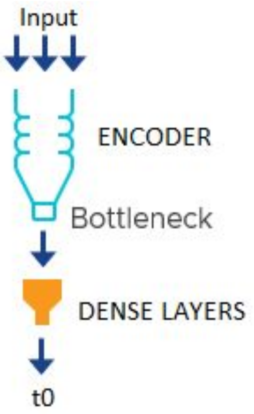
Layer (type)	Output Shape	Param #
encoded_sign (InputLayer)	[(None, 300)]	0
dense (Dense)	(None, 128)	38528
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 1)	65

Total params: 46,849
Trainable params: 46,849
Non-trainable params: 0



CNN Autoencoder + t0 Dense part





Total params: 20,790
Trainable params: 14,849
Non-trainable params: 5,941

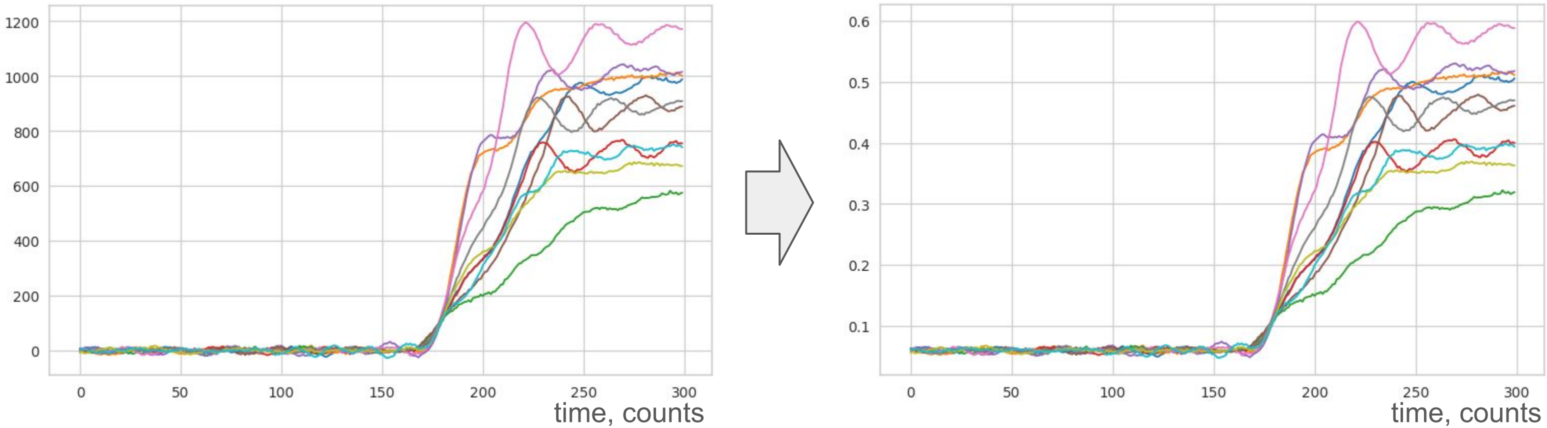
**Initially: ~198k parameters (wrong concept)*

+Advantage of CNN: convolution filters => highlighting the features

Data preprocessing

- Data scaling:

Too large spread of "features"



- Possible options

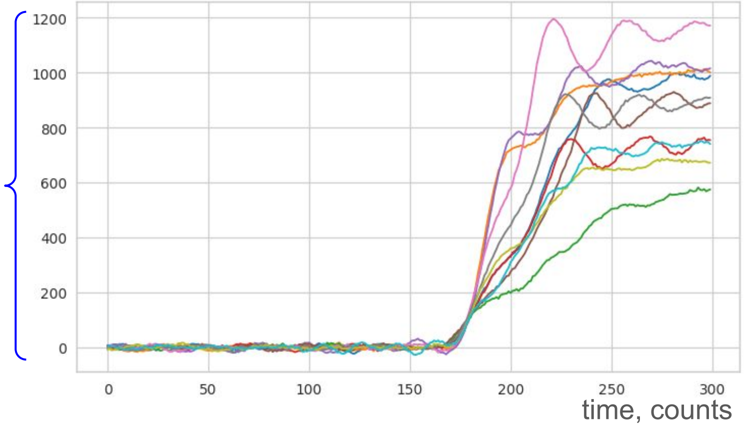
Linear scalers:

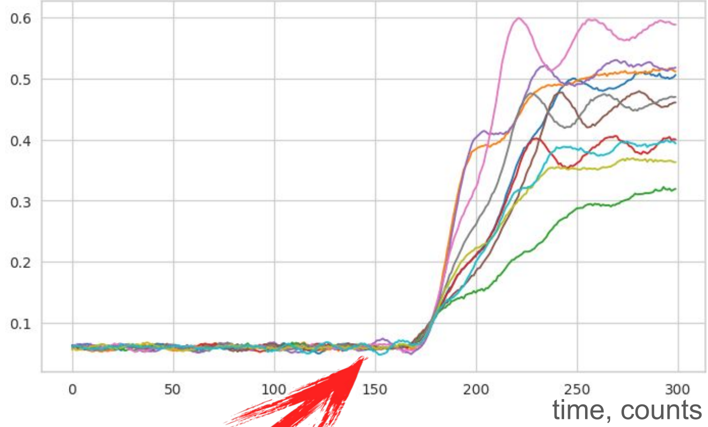
➔ StandardScaler: removes the mean and scales to unit variance
➔ MinMaxScaler: scale in the range [0,1]
➔ MaxAbsScaler: values are mapper in the range [0,1]
➔ RobustScaler: removes median and scales to the quantile range

Data preprocessing

- Data scaling:

Too large spread of "features"





- Possible options

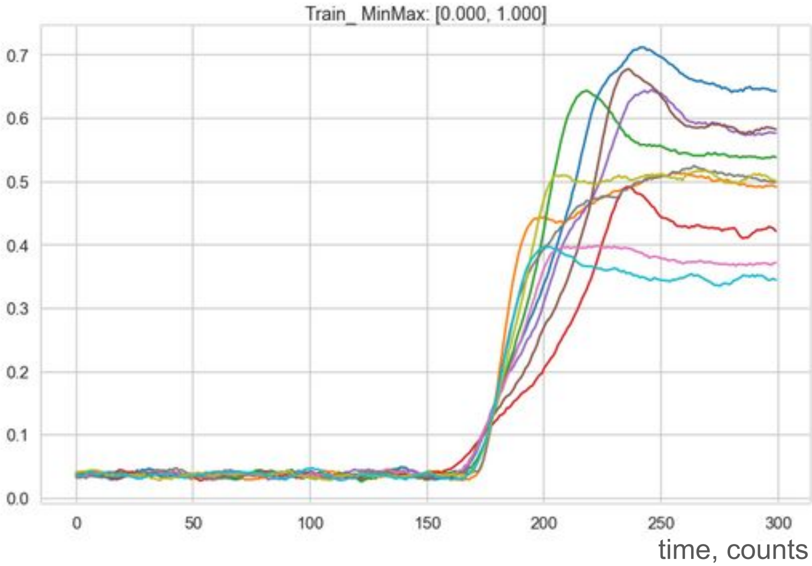
    Linear scalers:

    - ➔ StandardScaler: removes the mean and scales to unit variance
    - ➔ MinMaxScaler: scale in the range [0,1]
    - ➔ MaxAbsScaler: values are mapper in the range [0,1]
    - ➔ RobustScaler: removes median and scales to the quantile range

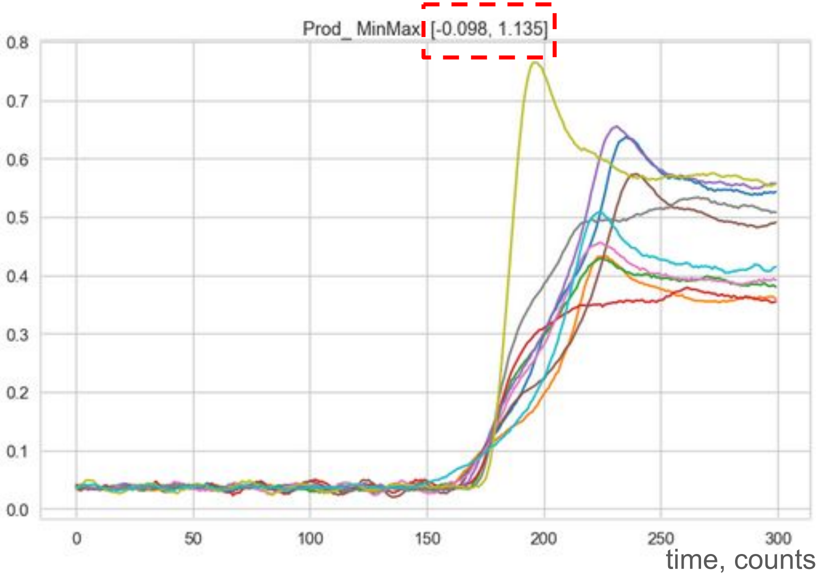
Issues of the MinMaxScaler

- Potential **outliers** for different datasets, if the scaler was fitted once:

Original dataset, scaler was fitted once



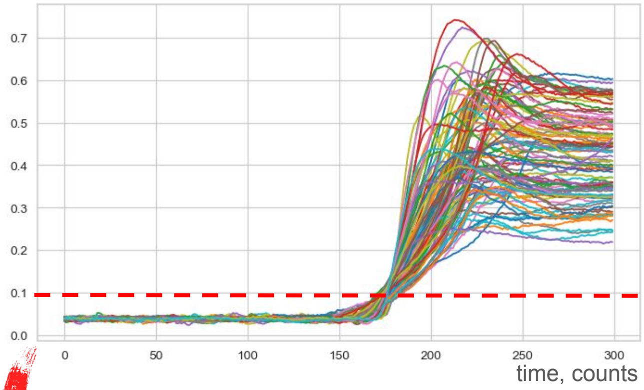
Another dataset, scaler was just applied



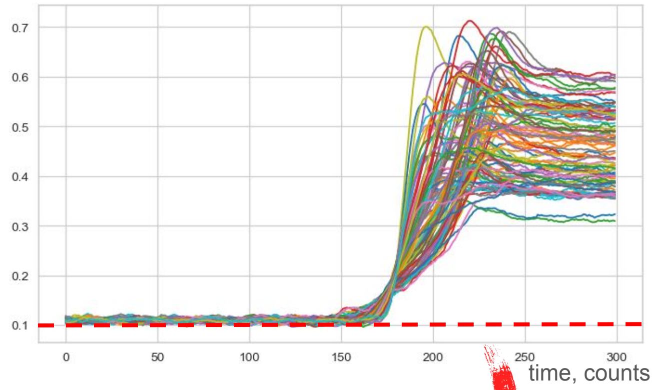
Issues of the MinMaxScaler

- **Different signal baselines** when <u>scaling separately</u>:

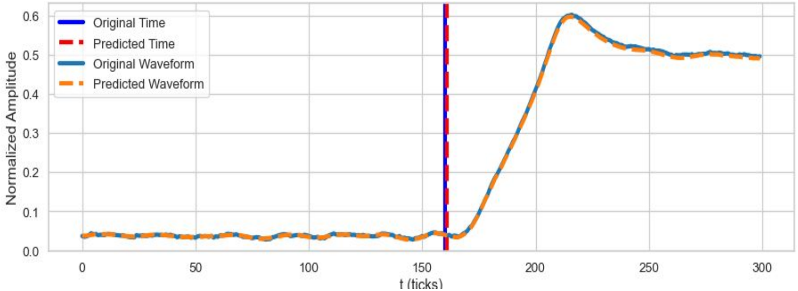
Original dataset, scaler was fitted

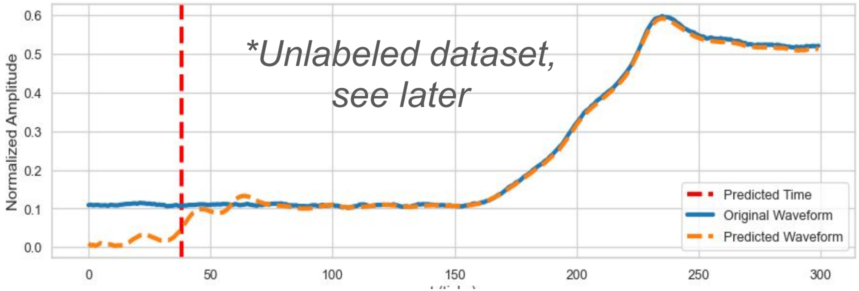


Another dataset, scaler was fitted separately



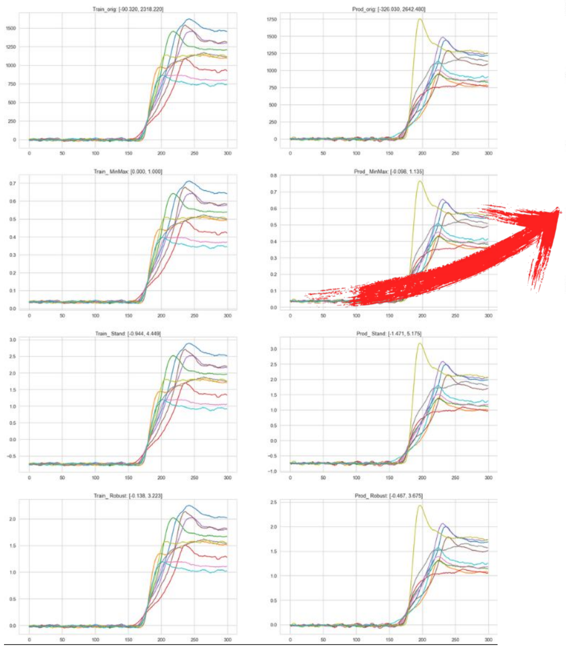
- Totally bad performance in second case:

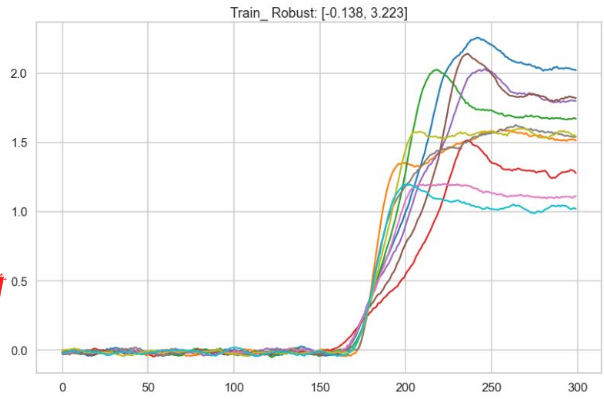


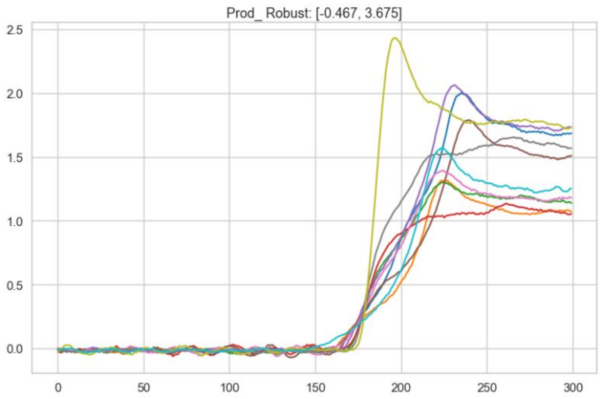


Another option: RobustScaler

Comparison different scalers







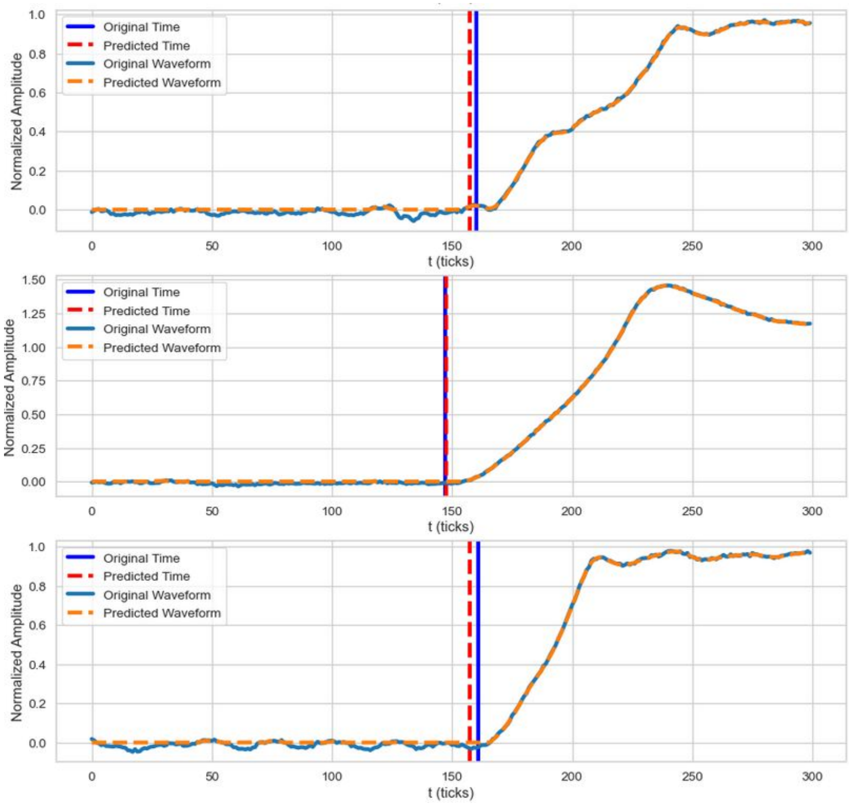
- RobustScaler: removes median and scales to the quantile range (not in the range [0,1])
- Robust to outliers: centering and scaling happen independently on each feature

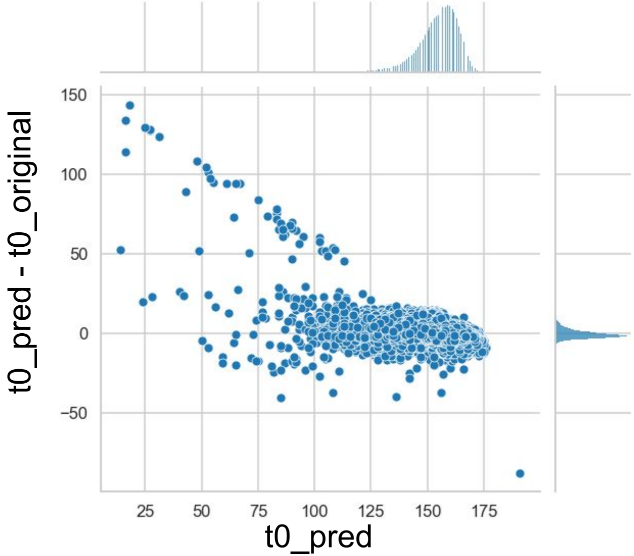
Current results: datasets

- Dataset #10800-11099:
  - solid Kr target
  - low energy range (0.5 - 1 MeV) was used for training (~724k events)
  - “production” unlabeled subset for checking (~462k events)
- Dataset #10188-10358:
  - H target (no muH->muKr transfer)
  - low energy range (0.5 - 1 MeV) (~284k events) and
  - extended energy range (0.3 - 2.5 MeV) (~801k events) were used
- Dataset #27000-27099:
  - Different run
  - solid Zn target and different detectors
- NN was trained on this dataset using RobustScaler
- In all cases of NN fitting (training) datasets were splitted to:
  - training and validation subsets (80%)
  - test subset (20%)

Current results

- Dataset [#10800-11099]() (low_en)
- **NN was trained**



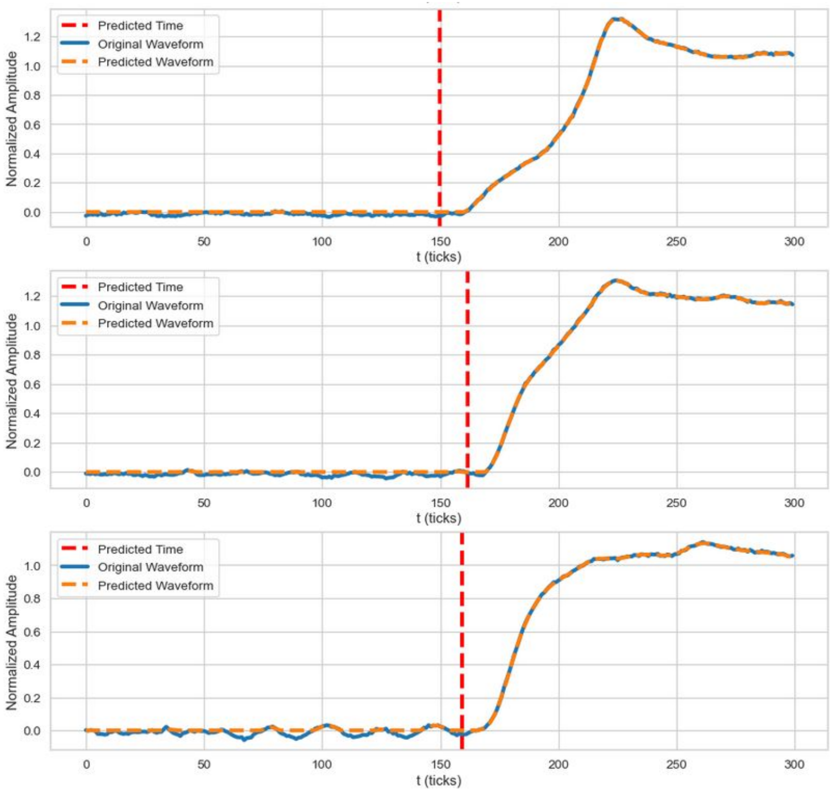


Metrics:

MAE: 2.354980576445645
MSE: 11.514306991941039
RMSE: 3.393273786764198
R^2: 0.863038920402409

Current results

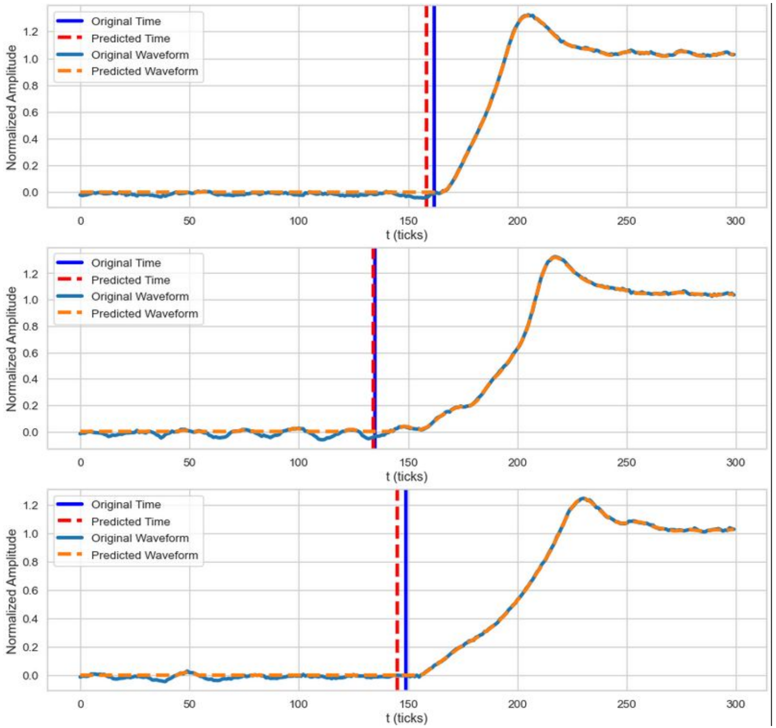
- Then **pre-trained NN** was applied to “production” datasets
- **unlabeled** “production” sub-dataset #10800-11099 (low-en) :

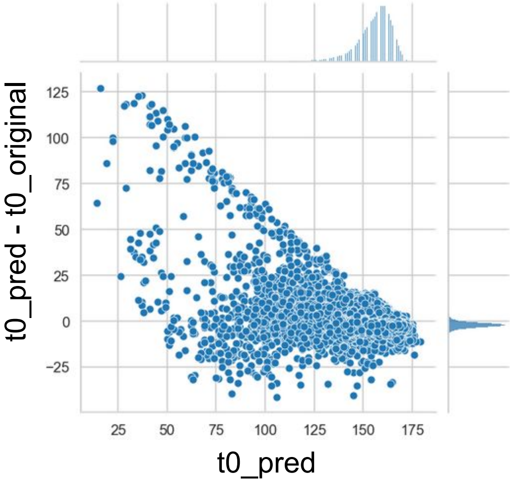


- Issue with baseline went away
- See Frederik’s part with analysis and comparing with ELET algorithm

Current results

- In order to check the NN model generalization, the same **pre-trained NN** was applied also to other datasets
- Dataset #10188-10358 (tlow-en), **W/O separate fitting** the scaller





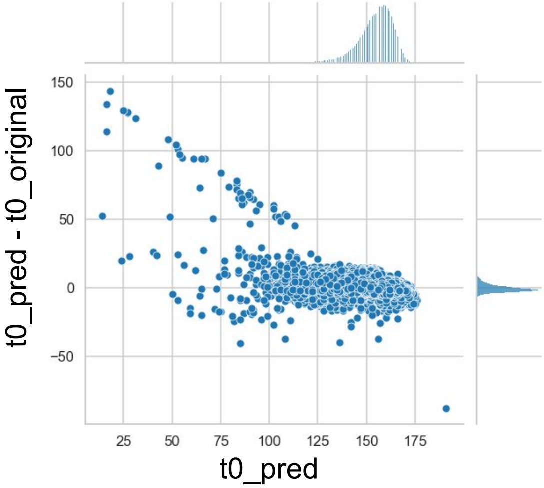
Metrics:

MAE: 2.5964176759765865
MSE: 12.57420742485626
RMSE: 3.5460128912422553
R^2: 0.8545213196793375

Current results

- Comparing different results (low_en)

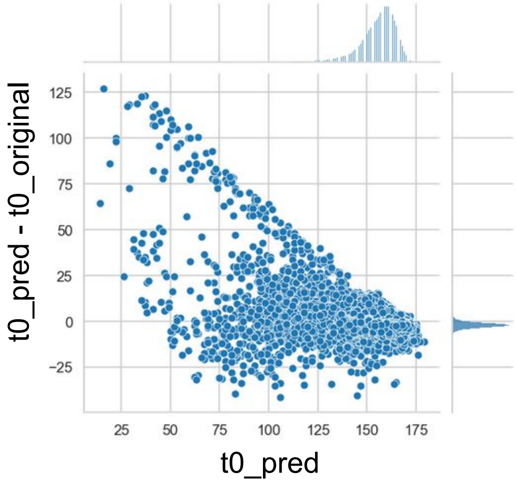
Det #10800-11099
(NN was trained)



Metrics:

MAE: 2.354980576445645
MSE: 11.514306991941039
RMSE: 3.393273786764198
R^2: 0.863038920402409

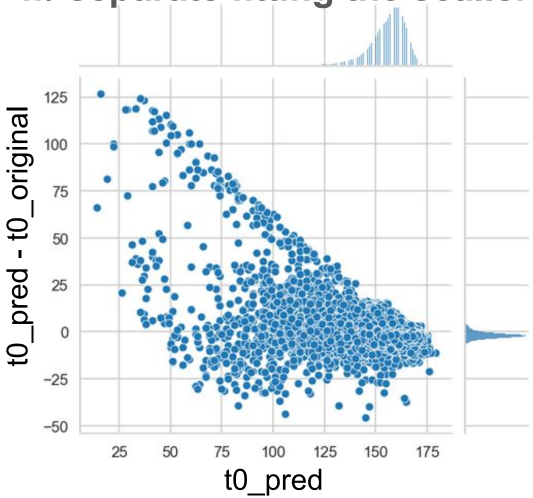
Dataset #10188-10358
w/o separate fitting the scaller



Metrics:

MAE: 2.5964176759765865
MSE: 12.57420742485626
RMSE: 3.5460128912422553
R^2: 0.8545213196793375

Dataset #10188-10358
w/ separate fitting the scaller



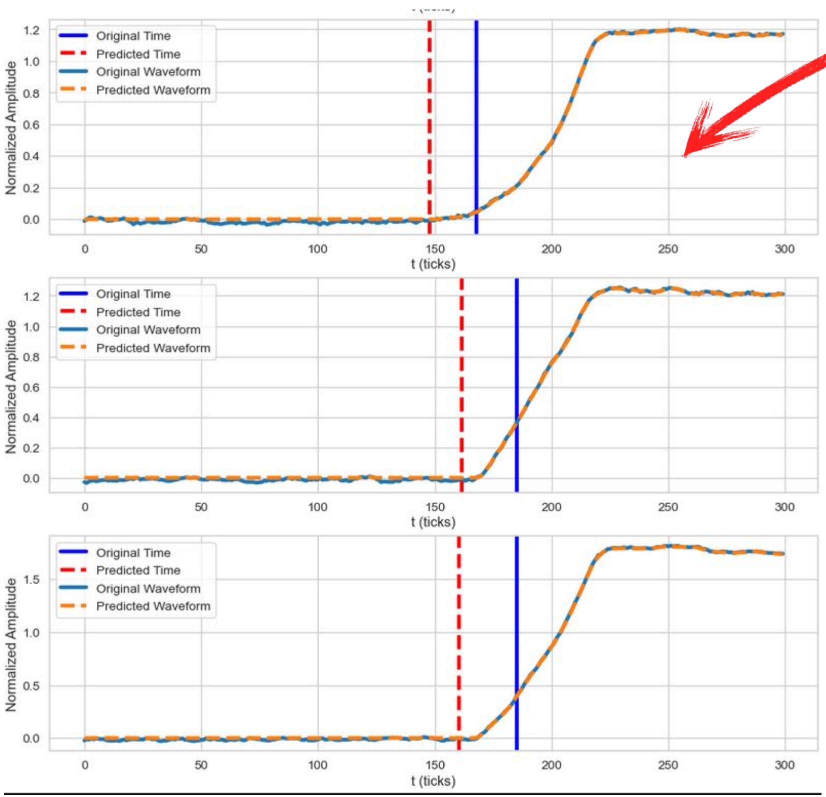
Metrics:

~10 ns

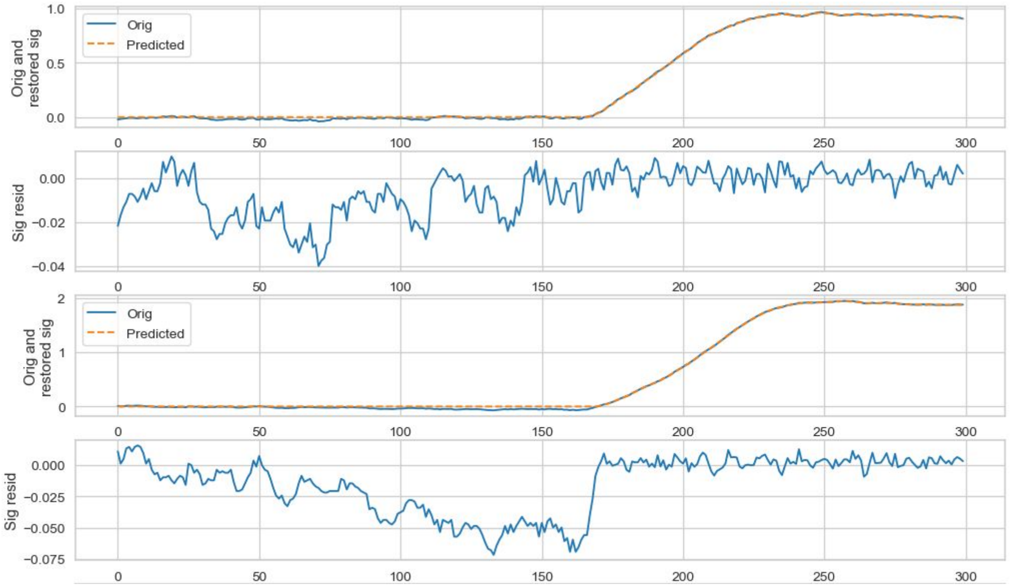
MAE: 2.4518733128336394
MSE: 12.022302841409712
RMSE: 3.4673192586506527
R^2: 0.860906640658218

Current results

- Dataset #27000-27099 (low-en) **W/O fitting the scaller:**
  - Data from different run, with different detectors, but the NN still seems to work

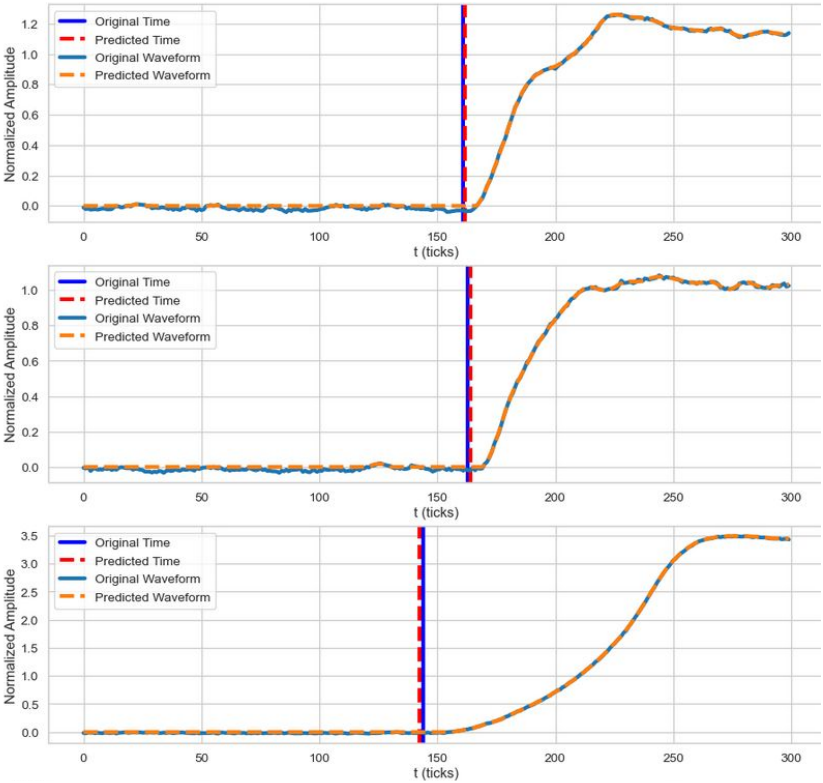


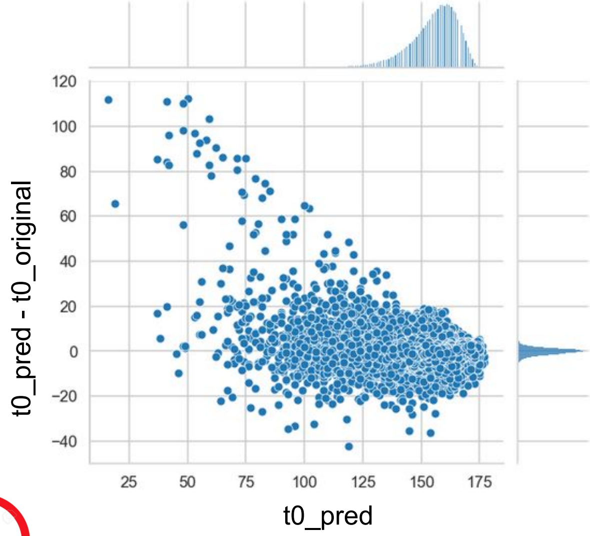
Issue with original t0 values (?)



Current results

- Dataset #10188-10358 (ext_en)
- **the NN was trained**





Metrics:

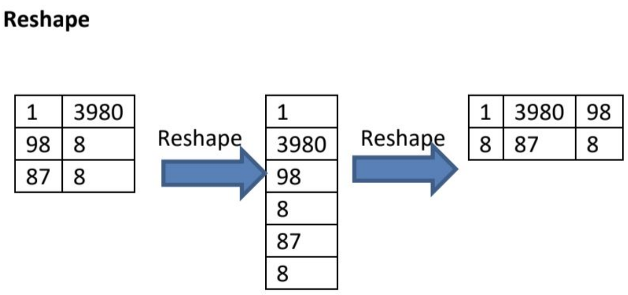
MAE: 1.4607229014786671 → **~5.8 ns**
MSE: 7.149763225980863
RMSE: 2.6739041168263427
R^2: 0.9399099430911559

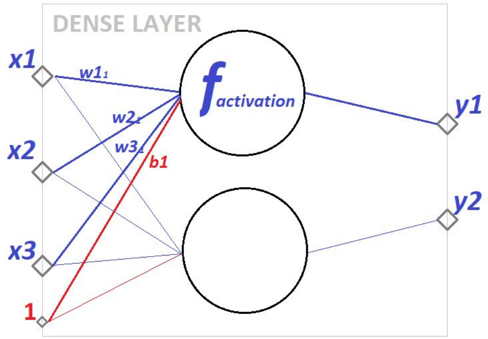
To do list:

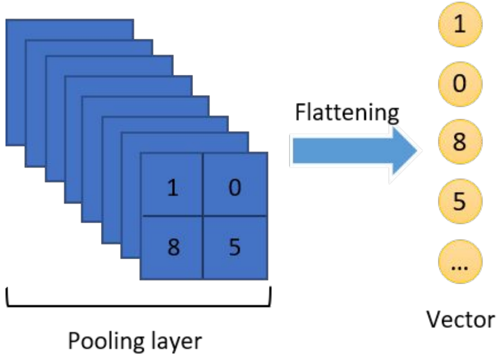
- Check other options for scaler
- Analize all results with double-checking
- Train the NN on different datasets, different energy ranges, and use more “production” ones
- Integration of the NN in terms of code for real experiment pipeline:
    - Python-scripts
    - reading saved file with trained NN (structure + parameters) with C++ code
    - etc.

Backup

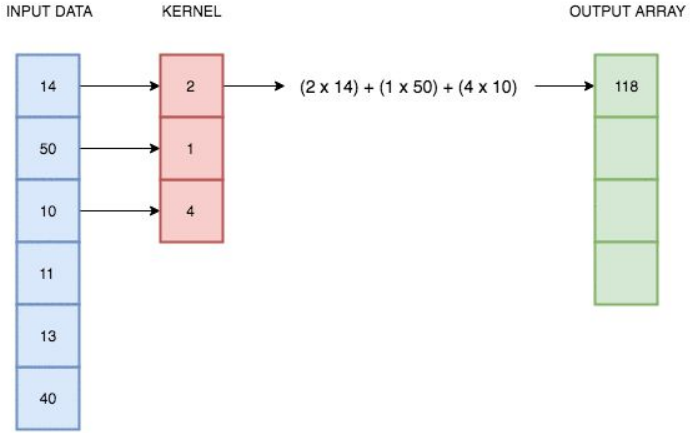
Basic NN layers



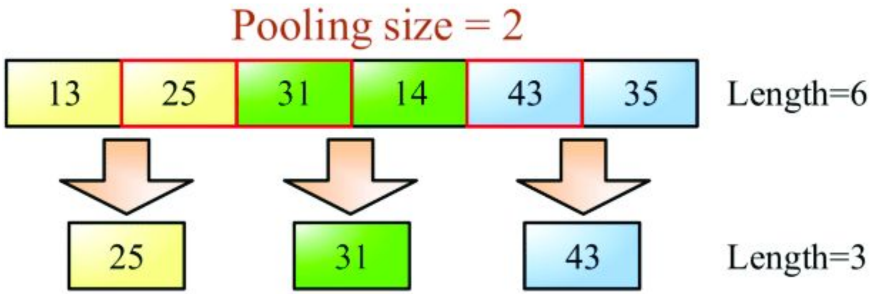




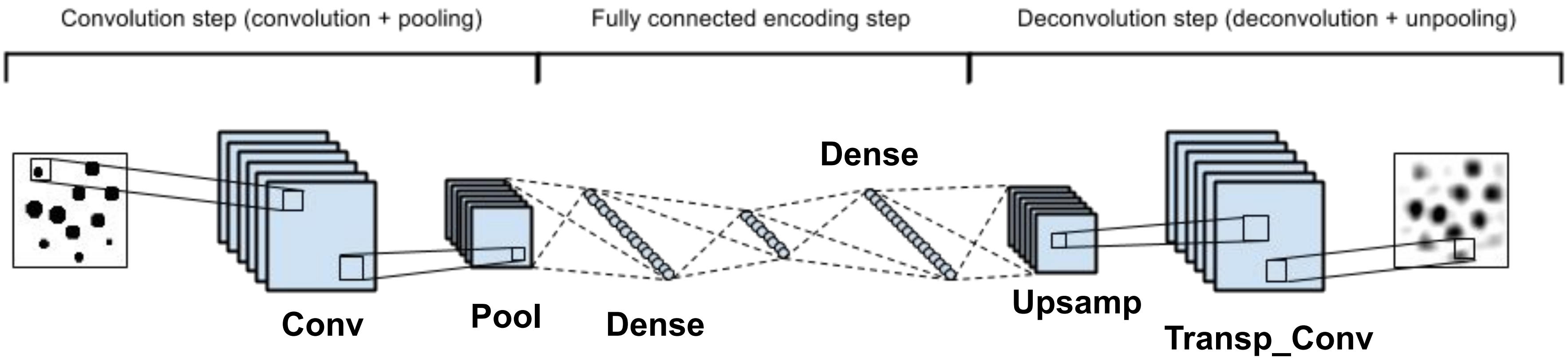
- CNN: Convolution layer



- CNN: Max/Avg Pooling



Example of Autoencoder CNN



Det t0 NN (curent)