Impact of filtering on Cygno’s images

Guilherme Lopes, Igor Abritta, Rafael Nóbrega

Motivation

- How to evaluate the performance of preprocessing algorithms for CYGNO?
- What is the impact of different preprocessing algorithms on energy estimation?
- Is it possible to get same clustering results decreasing the number of points sent to dbscan?

Objective

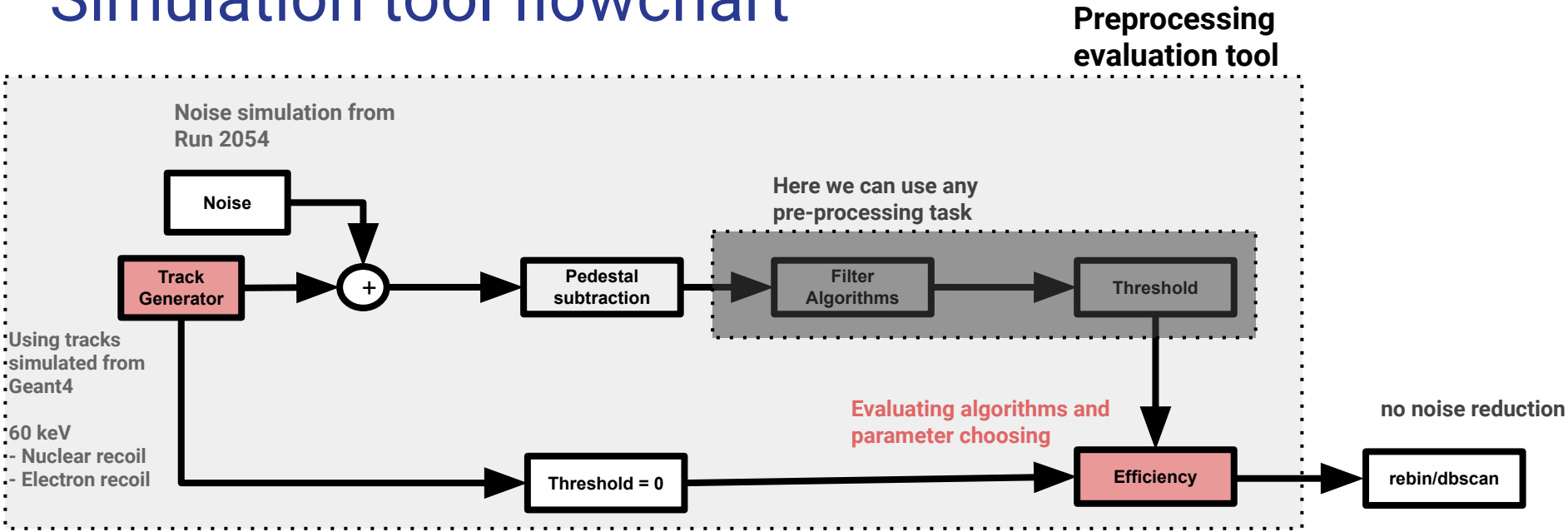
- Find a proper methodology to assess the performance of preprocessing algorithms for CYGNO → propose a test environment with this end
- Evaluate the impact of some filters on efficiency/false-alarm and energy estimation using simulated data
    - In this slides we are considering 60 keV simulated events to discuss and test the method used for performance assessment, with some preliminary results
- Evaluate the impact of some filters using real data (to be done)

Historical Overview

- Since the beginning of this work the need of a simulation tool was present (one year ago)
- Therefore, before thinking about filtering we worked on a simulation tool that would generate signal + noise, all based on the real Cygno's images:
  - The real physical signal itself was hard to simulate without using GARFIELD/GEANT packets, then we proposed a simple method that seemed to work **for the purpose of testing the filters.**
  - The noise process was simulated based on analysis made with real noise acquisition runs:
    - This noise simulation was though in a way that it could be integrated to any new signal simulation tool that could come up in the future.
- With this tool in hands we could implement and test the first filters for Cygno;
- Recently, a new simulation tool (from Roma) was available and then we used its data to converge with the filtering studies:
  - We are trying to create a proper methodology that will facilitate the process of proposing a paper to the collaboration about "impact of filtering on CYGNO"

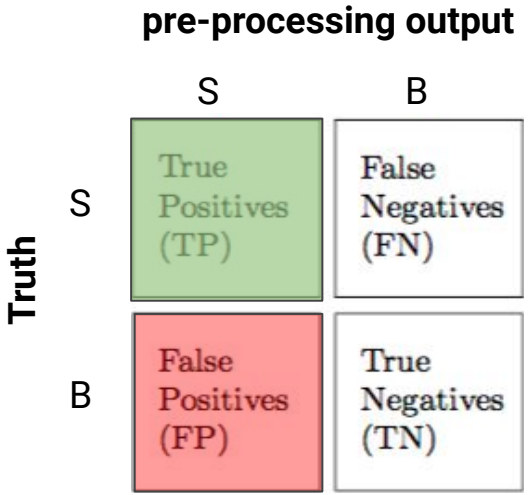
Flowchart overview

Simulation tool flowchart



Evaluating the efficiency

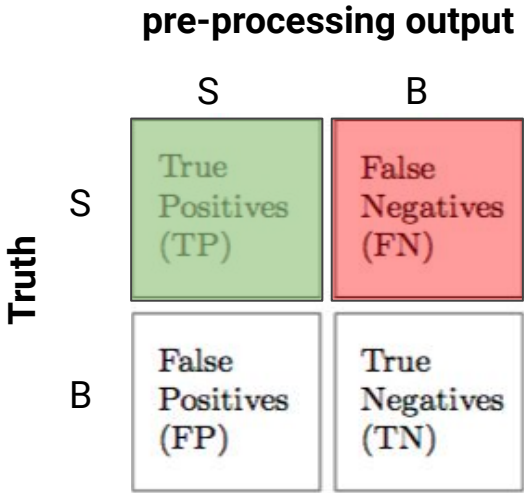
- ***Precision-recall curve;***
- ***Used for imbalanced datasets;***
- ***For each threshold value we have a precision value at that point;***
- ***Precision -> #Classified as signal and really signal / #Classified as signal***



$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Evaluating the efficiency

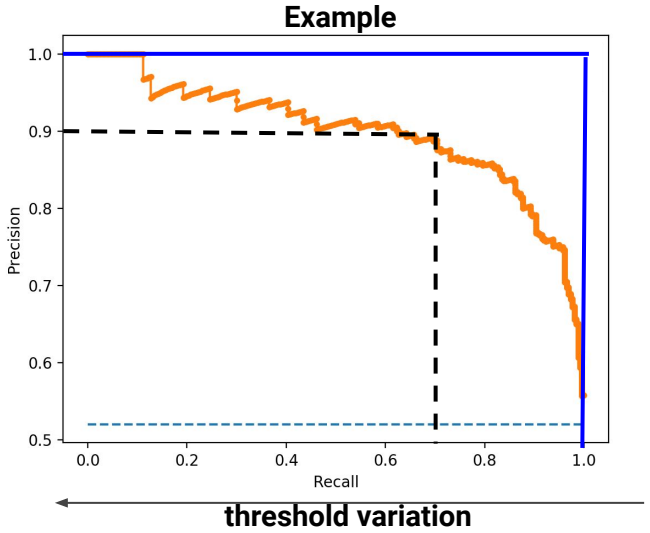
- ***Precision-recall curve;***
- ***Used for imbalanced datasets;***
- ***For each threshold value we have a recall value at that point;***
- ***Recall -> Signal detection efficiency.***



$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Ways to Choose a filter

- ***We can determine if a curve is better than another by using some metrics;***
  - ***f1-score***



$$F1 = 2.\frac{Precision \times Recall}{Precision + Recall}$$

max(f1) = (2*(0,9*0,7))/(0,9+0,7) = 0.7875

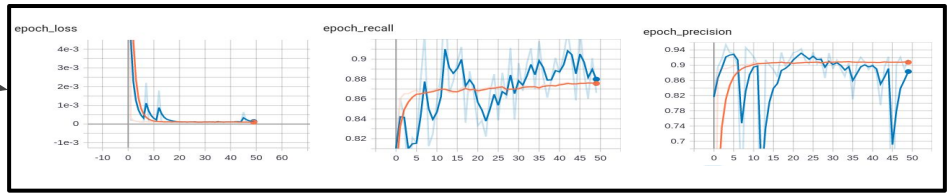
Best case -> max(f1) = 1

Each pair (filter, parameter) has a precision x recall curve that will be summarized using f1-score.

Setup

- ***Input***
  - ***For now, we are using 60 keV electron and nuclear recoil (200 images);***
- ***Noise simulation using ecdf algorithm for run 2054;***
- ***Filters used:***
  - ***Convolutional Neural Network (U-net):***
    - ***Trained using 70% train, 15% test and 15% validation;***
    - ***50 epochs;***
    - ***Metrics: Precision and recall.***
  - ***Mean using windows from 3 to 23;***
  - ***Median using windows from 3 to 23;***
  - ***Gaussian using windows from 3 to 23;***
  - ***Cygno (n*sigma threshold using std map);***

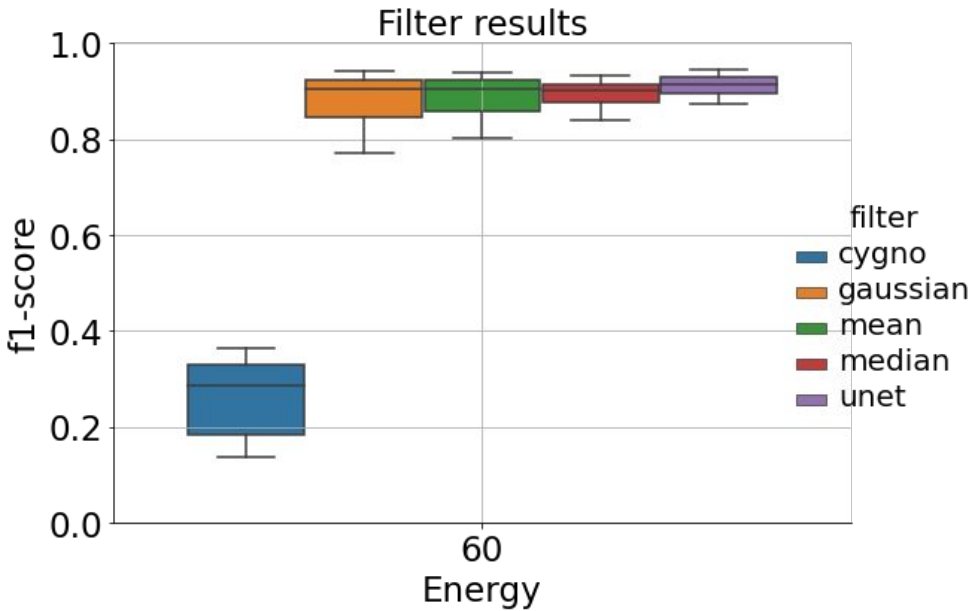
Training step



Results

Results

- ***F1-score filters***



best parameters

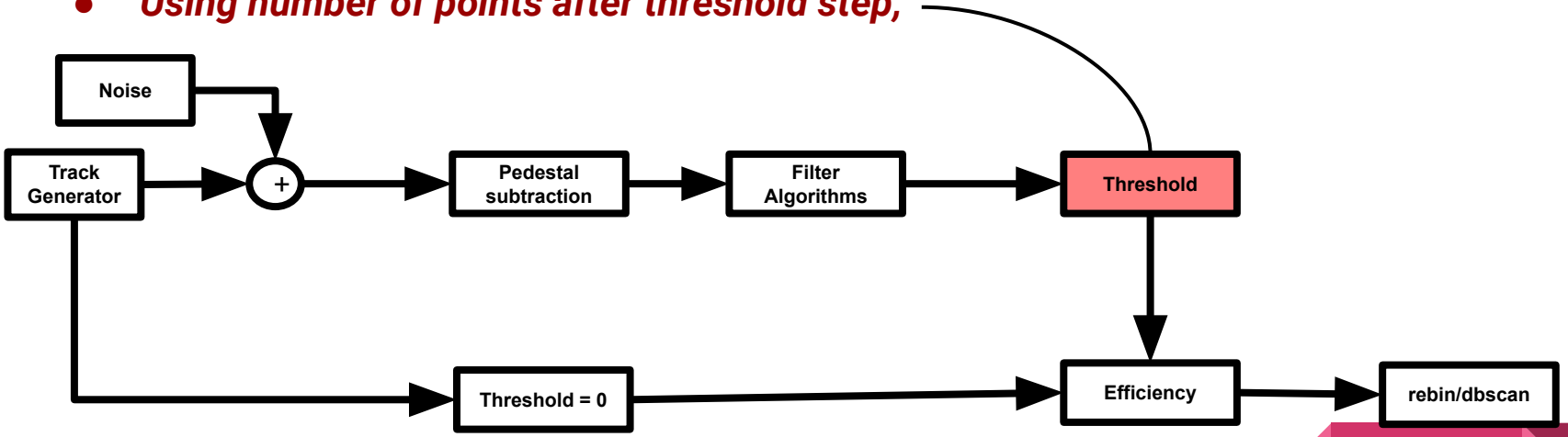
filter	parameter	f1_median	f1_25q	f1_75q
unet	0	0.915000	0.895840	0.928599
mean	15	0.903572	0.855331	0.921199
gaussian	13	0.903410	0.843906	0.922620
median	17	0.897056	0.874514	0.911014
cygno	0	0.286066	0.181680	0.329005

ps: If 0, don't have parameters

Evaluating preprocessing in clustering algorithm

How ???

- ***Using number of points after threshold step;***



How ???

- ***Using number of points after threshold step;***

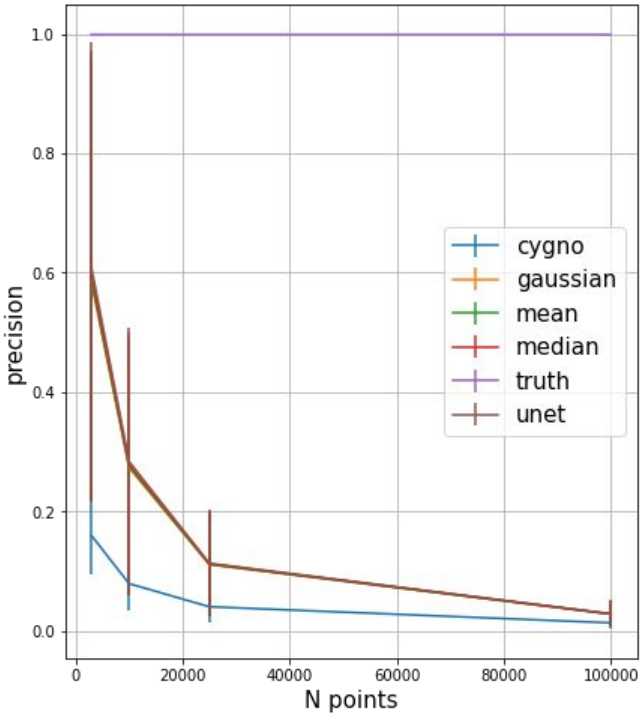
N	Precision	Recall	DBSCAN	Energy estimation
Many	Low	High	Slow	good (depending)
Few	High	Low	Fast	Poor

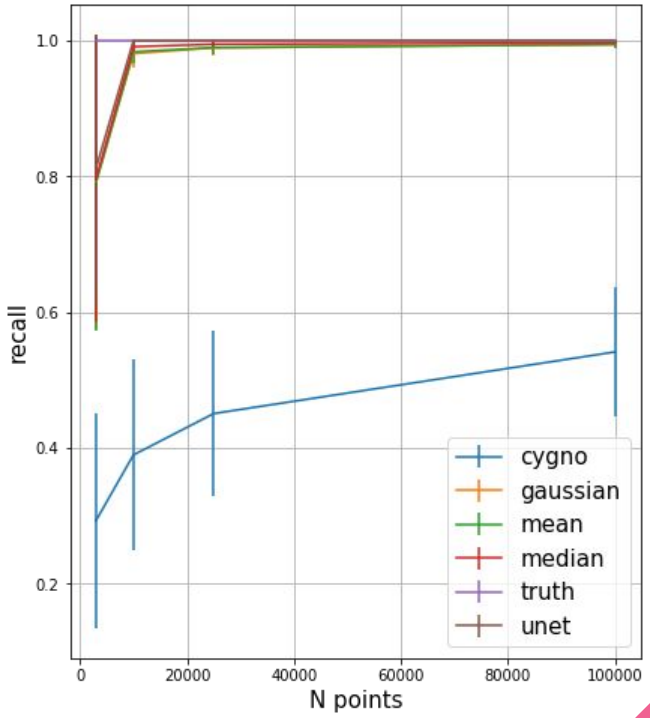
- ***Is there a filter that can provide high precision, high recall, good energy estimation using few points (and fast dbscan) ?***

Setup

- ***Input***
  - ***For now, we are using 60 keV electron and nuclear recoil (200 images);***
- ***Using filter parameters from efficiency evaluation;***
- ***Set threshold to get N points, where N is 3000, 10000, 25000 and 100000;***
- ***DBSCAN parameters***
  - ***eps = 5.8***
  - ***min_samples = 70***
- ***Evaluate cluster integral from reconstruction.py python algorithm (without noise reduction).***

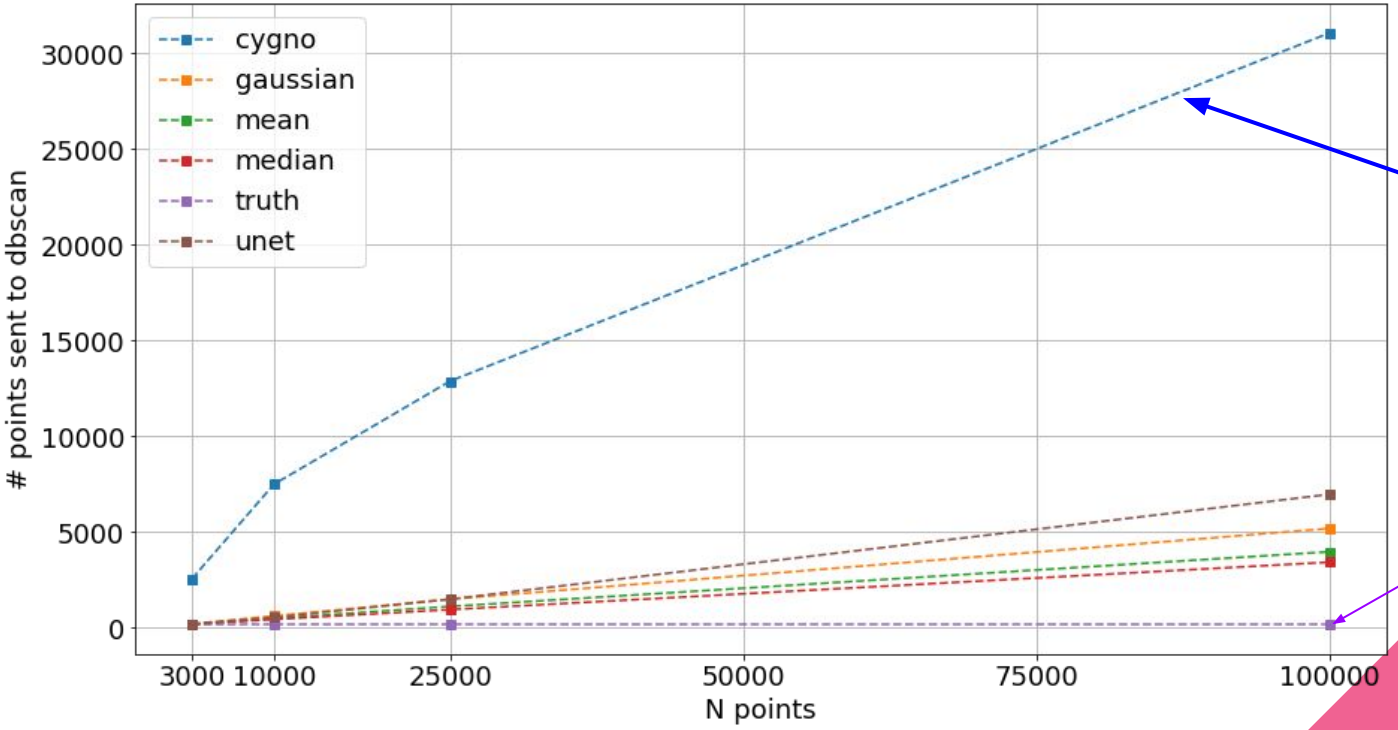
Precision and recall





- ➔ **Precision decreases when N increases;**
- ➔ **For N too big, the recall is closer to 1;**

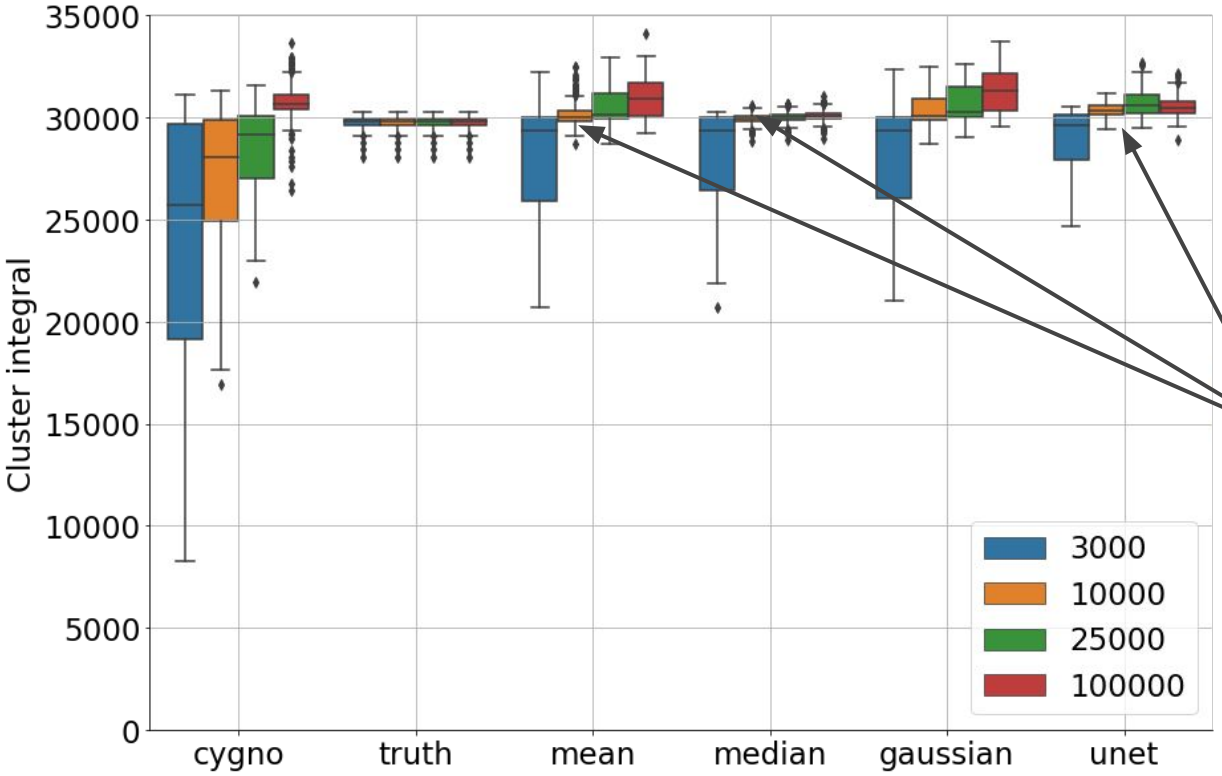
Points sent to dbscan



Indicates many background points away from target cluster

Ideal case

Evaluating clustering



It would be also important to see how resilient are the filters (e.g. when many fakes are present - low precision and high recall - how much the energy estimation get worse?)

Using only 10000 points, we can estimate cluster_integral closer to truth

Conclusion and next steps

Conclusions

- ***The results show that the presented methodology might be useful to evaluate different preprocessing algorithms (suggestions are welcome);***
- ***The presented results seemed coherent, indicating that the codes are working properly;***
- ***This preliminary study shows that some filters might help to decrease the number of points sent to dbscan;***
  - ***It might help to remove the rebinning operation if needed***

Next steps

- ***A time evaluation of those algorithms;***
- ***Include low energy particles as input to produce results closer to reality;***
- ***Evaluate filters performance using real data;***