FoCal-H

●●●

Bjartur í Túni Mortensen, Jens Peter Andersen,
and Julie Bojesen Koefoed

Goals and Motivation

Explore FoCal-H data with machine learning approaches

Classify and do regression on labels

Compute our own features

Use newest data to learn more about effect of angle

Treat showers as image data

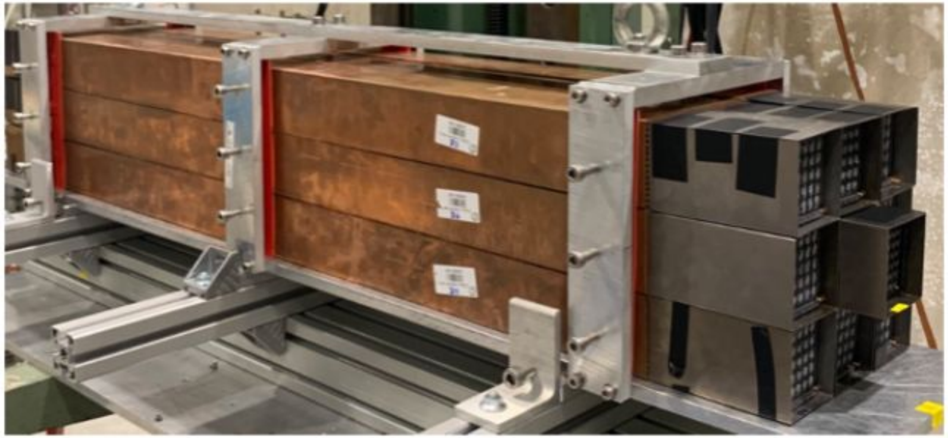
Detector

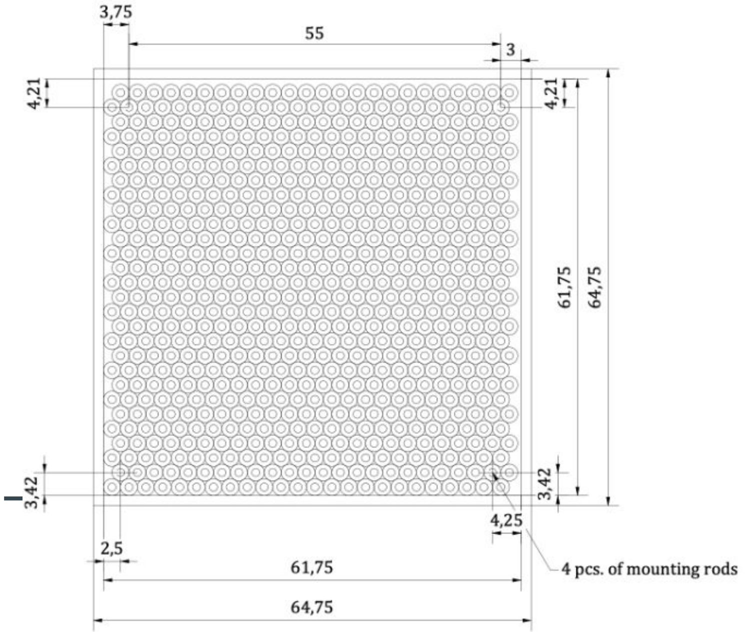
Hadronic calorimeter intended as an upgrade to the ALICE experiment in the Large Hadron Collider (LHC)

Calorimeters measure energy

Copper tubes enclosing scintillating fibers

Particle showers are generated and photons are emitted from scintillating fibers and bundled to 249 Silicon Photomultipliers (SiPM)





Data

A value for each photo-detector (channel) for each event (ideally a single particle)

256 channels (249 connected to photosensors)

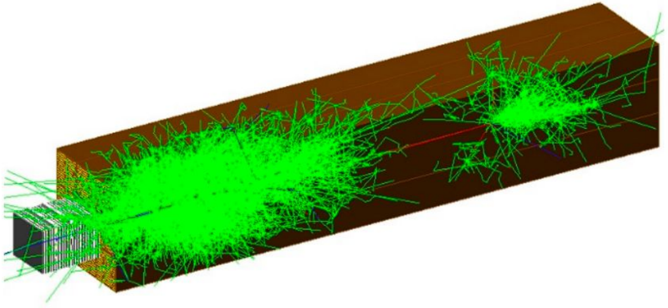
Testbeam event from September 2023

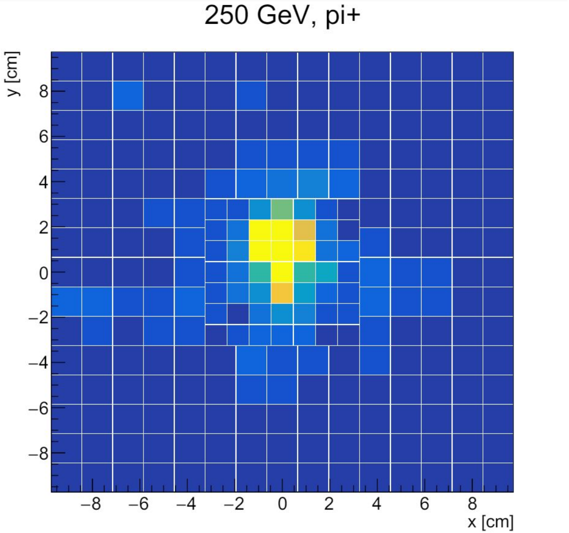
Dataset has 3 labels – Energy (20-350), Particle (electrons, hadrons) and Angle (0,2)

Also new data from May 2024 (4 angles)

256/249 spatially correlated features, otherwise "boring" data

But we compute shower characteristics (moments)





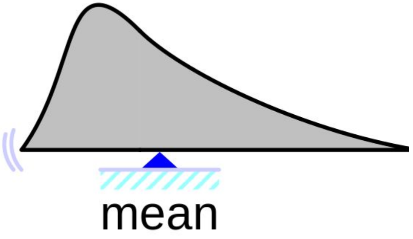
Features

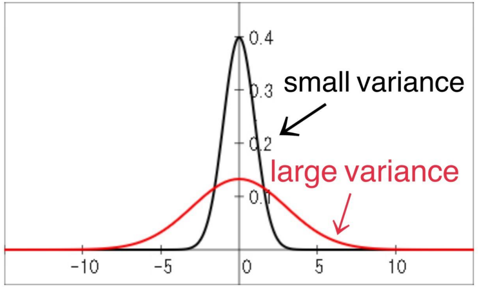
Center-of-mass (mean)

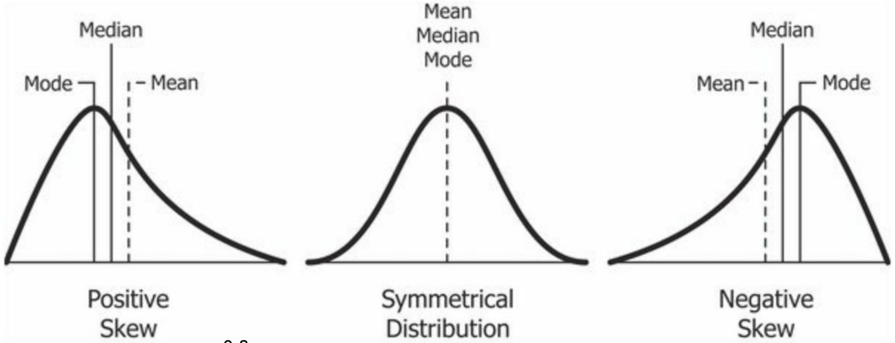
Variance

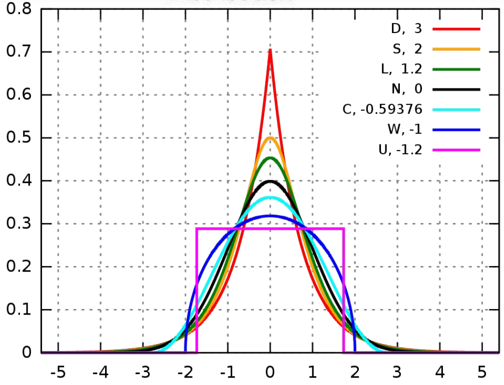
Skewness

Kurtosis



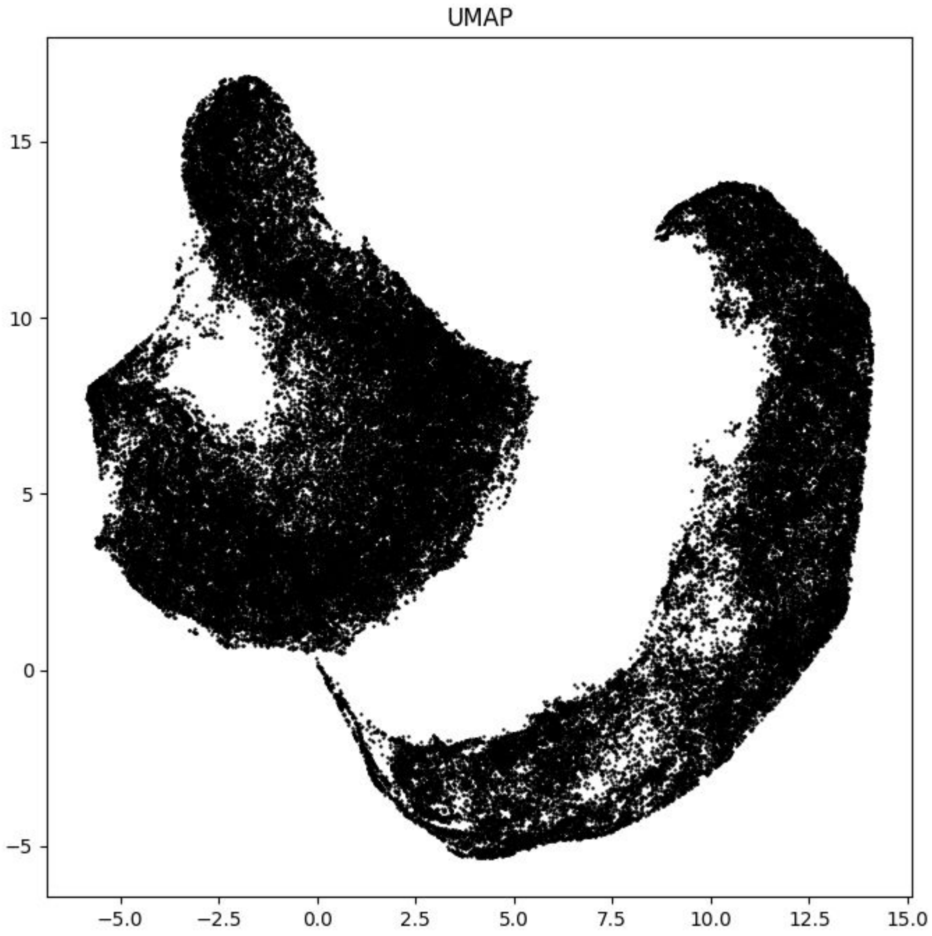






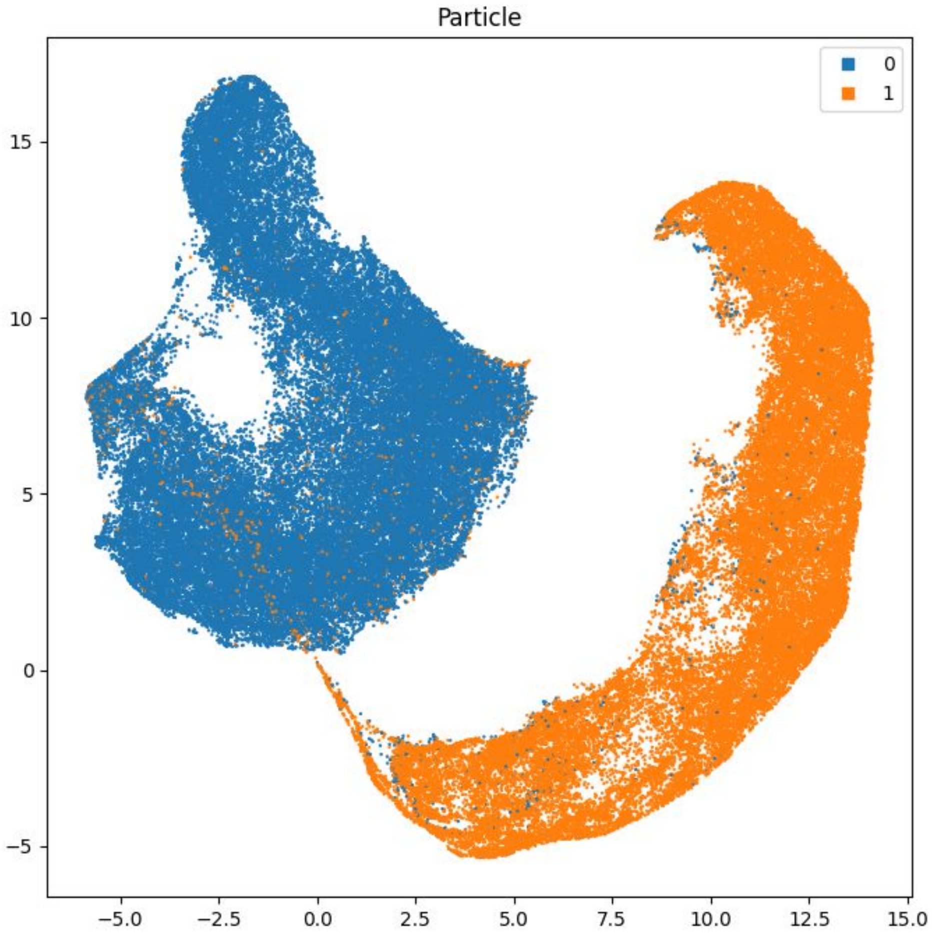
UMAP

Dimensionality reduction on a random sample of events with moment feature data only. Reduction to 2D with overlapping energies between particles and angles



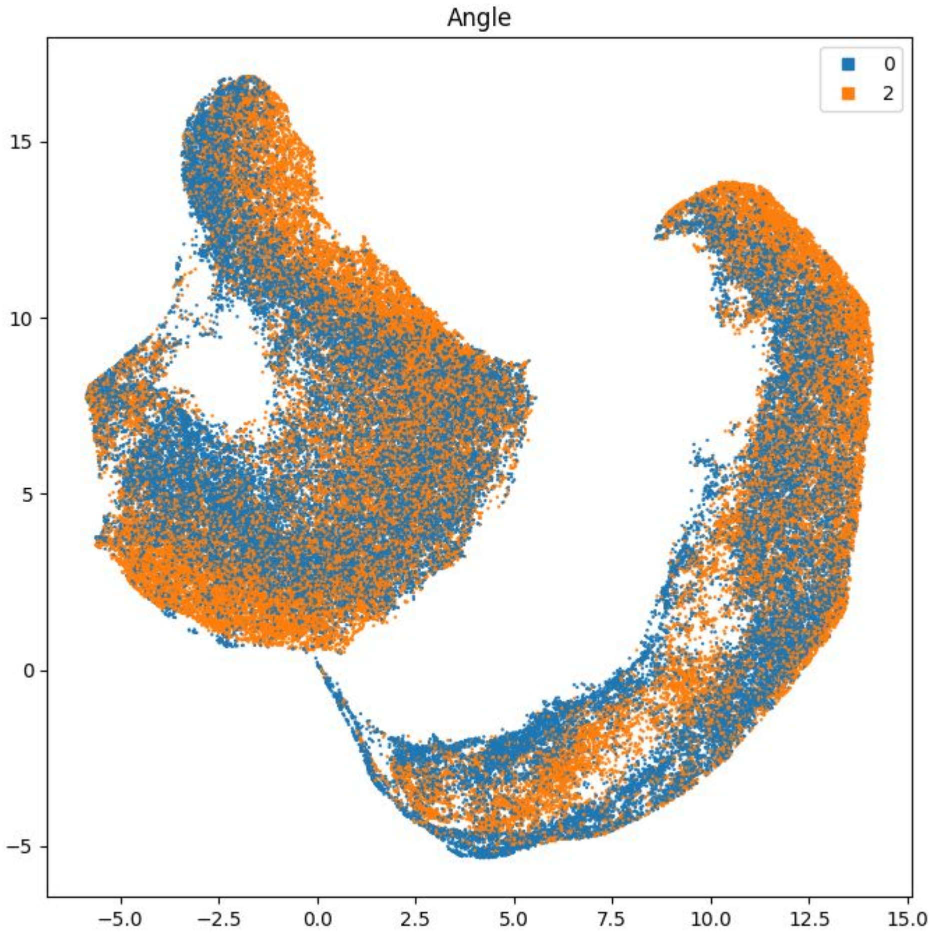
UMAP

Color coding according to particle. Looks like the data will be separable by particle



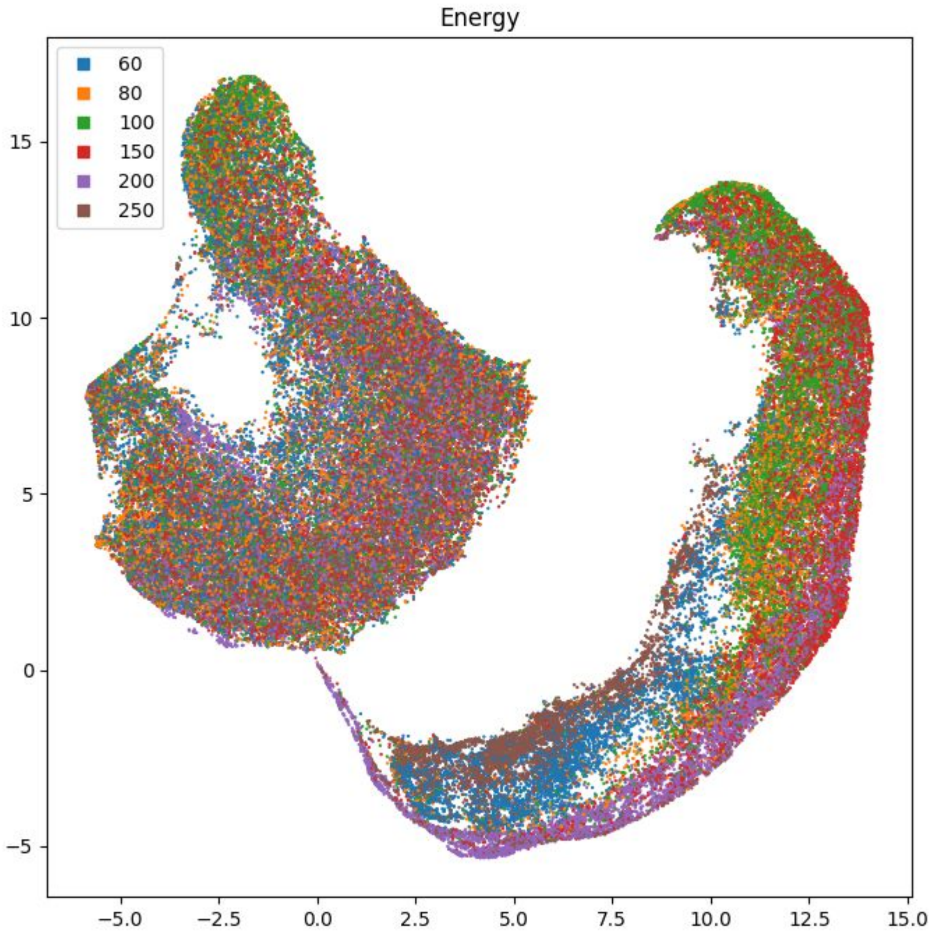
UMAP

Color coding according to angle. Seems mixed, but there are trends



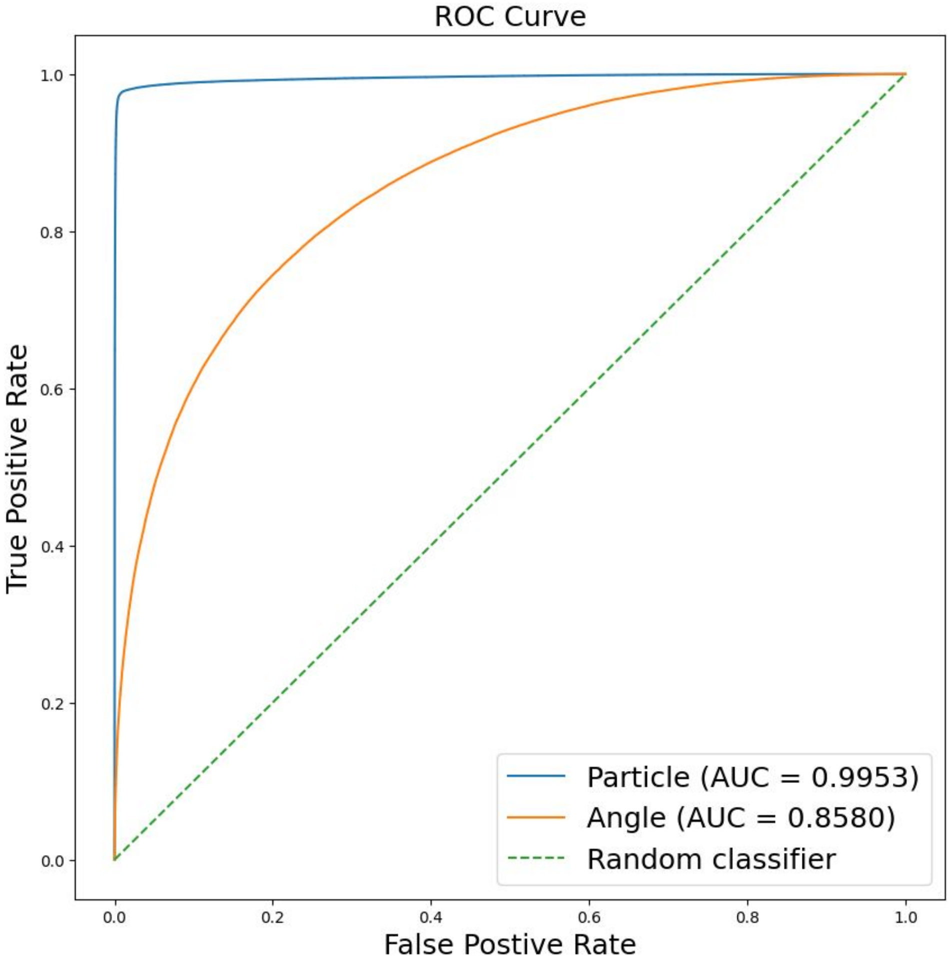
UMAP

Color coding according to energy. Since it's only on moment data we don't expect a separation



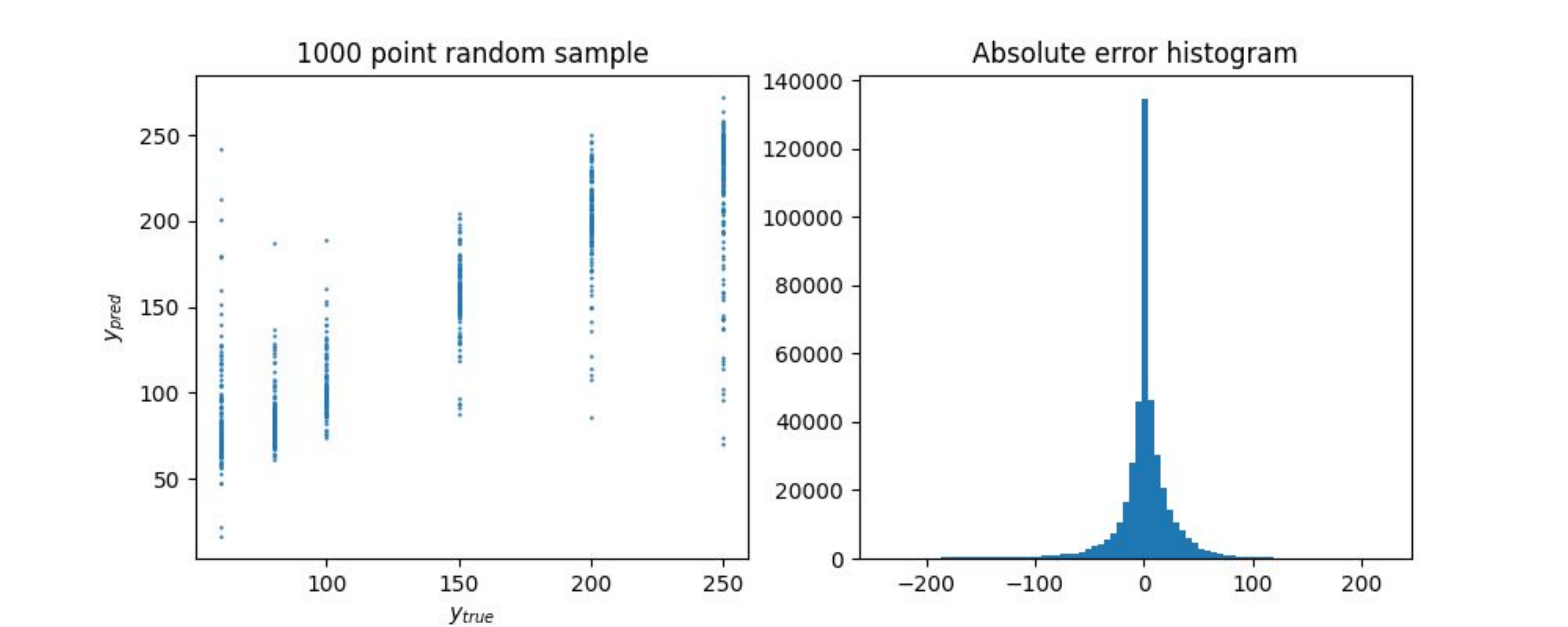
Classification on Particle and Angle

Particle acc: 0.985
Angle acc: 0.772



Regression on Energy

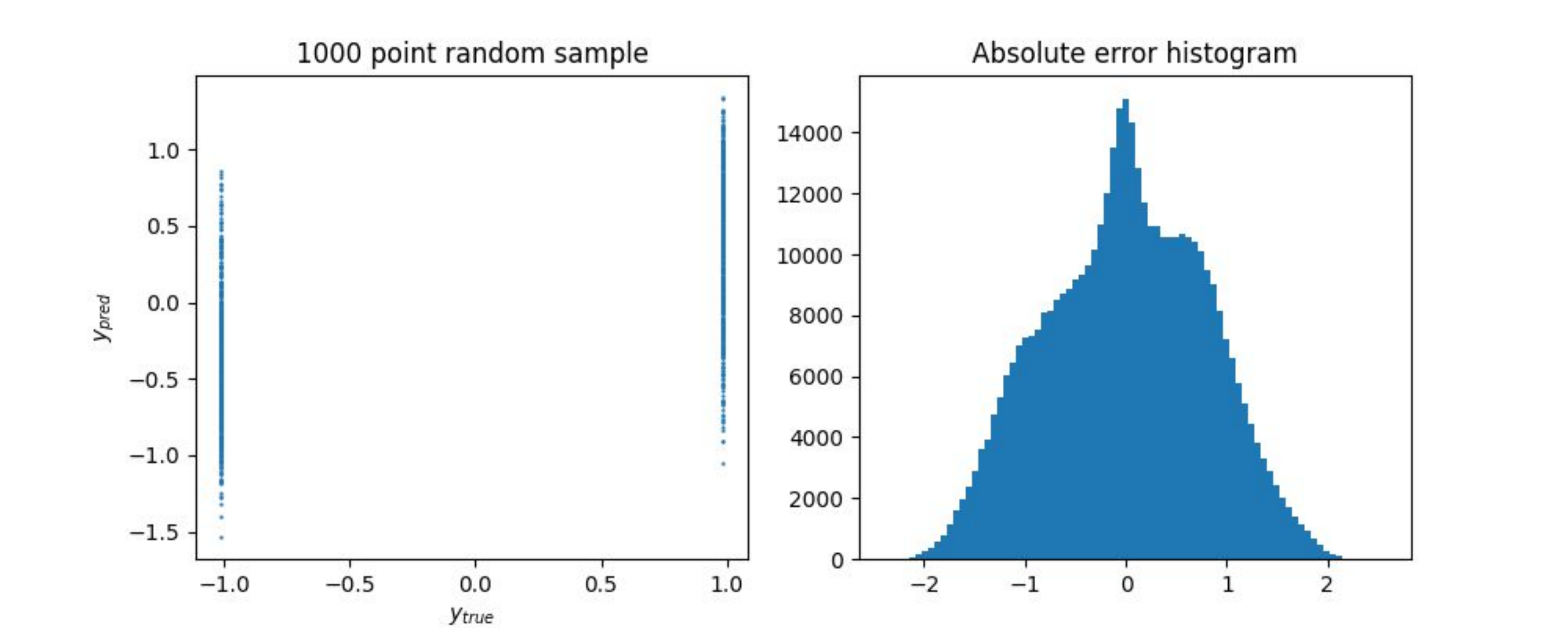
Mean Absolute Error: 16.5
Relative Mean Absolute Error: 0.141



Regression on Angle

Mean Absolute Error: 0.646
Relative Mean Absolute Error: 0.646

Two angles 0 and 2 degrees (scaled to -1 and 1)



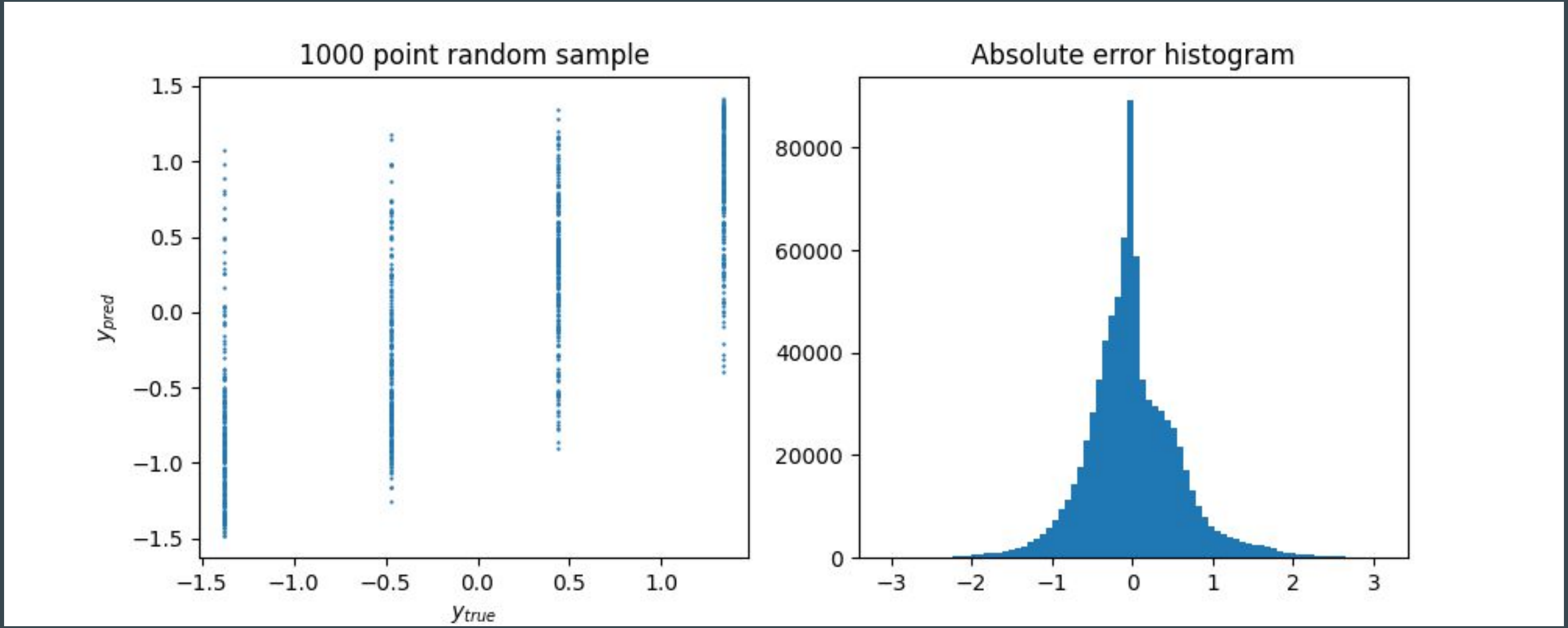
1000 point random sample

Absolute error histogram

Regression on Angle from May Data

Mean Absolute Error: 0.419
Relative Mean Absolute Error: 0.564

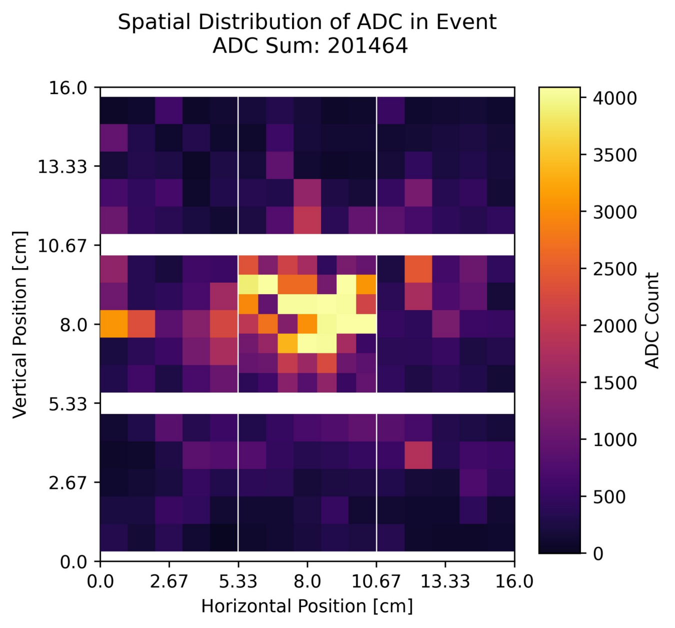
Four angles 2, 0, -2, -4 degrees (scaled to between ~ -1.4 to 1.4)



Treating Data as an Image

Going from 249 sensors → 441 pixels by upscaling the 5x5 areas

1 x 7 x 7 + 8 x 5 x 5



21 x 21

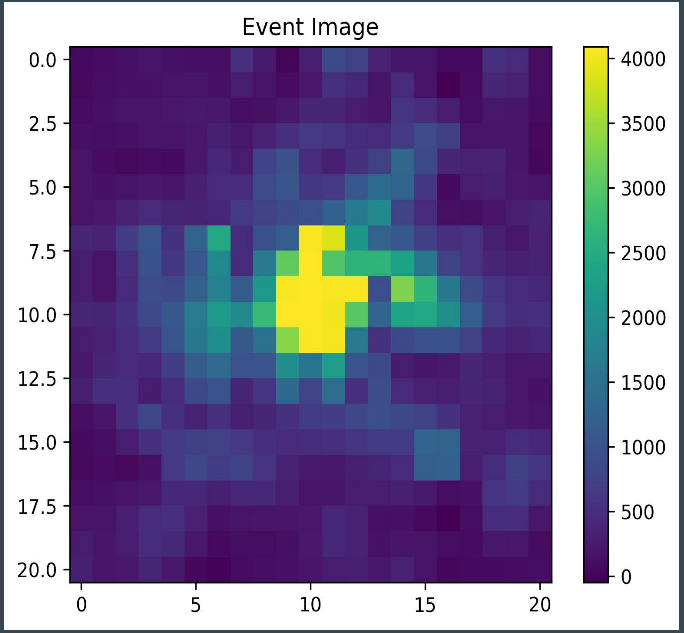
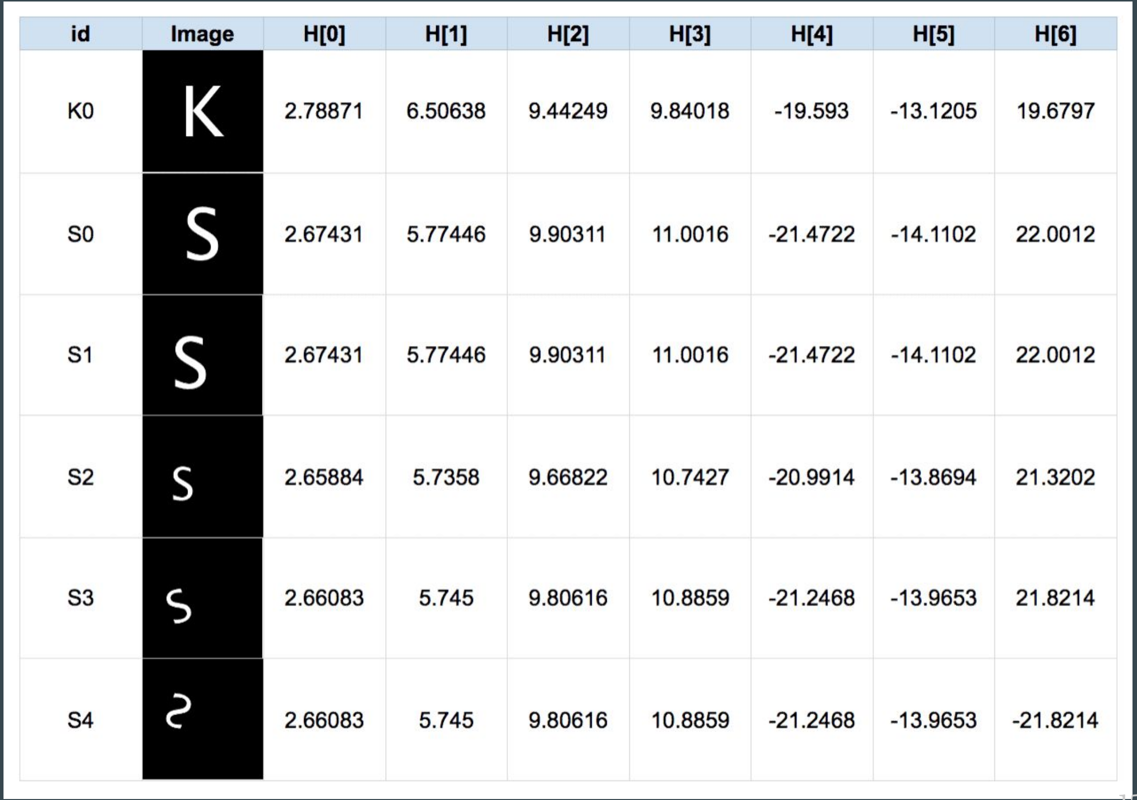


Image Moments and the 7 Hu Moments

Instead of treating the 441 pixel image as 441 variables we calculate 7 features

Image moments: a particularly useful type of moments derived from “ordinary moments”: Mean, Variance, skewness and so on.

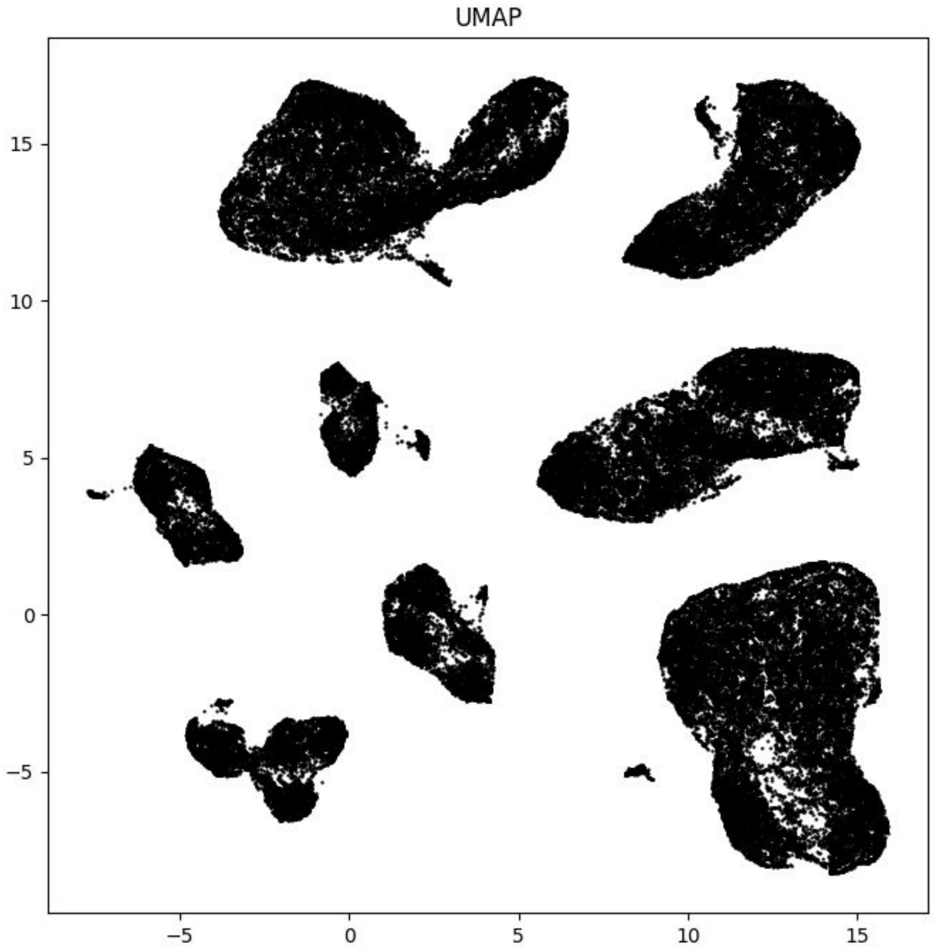
The 7 Hu moments are invariant to:
translation, rotation, scaling, and reflection (6/7)



id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0	K	2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2	S	2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3	S	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4	Ƨ	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

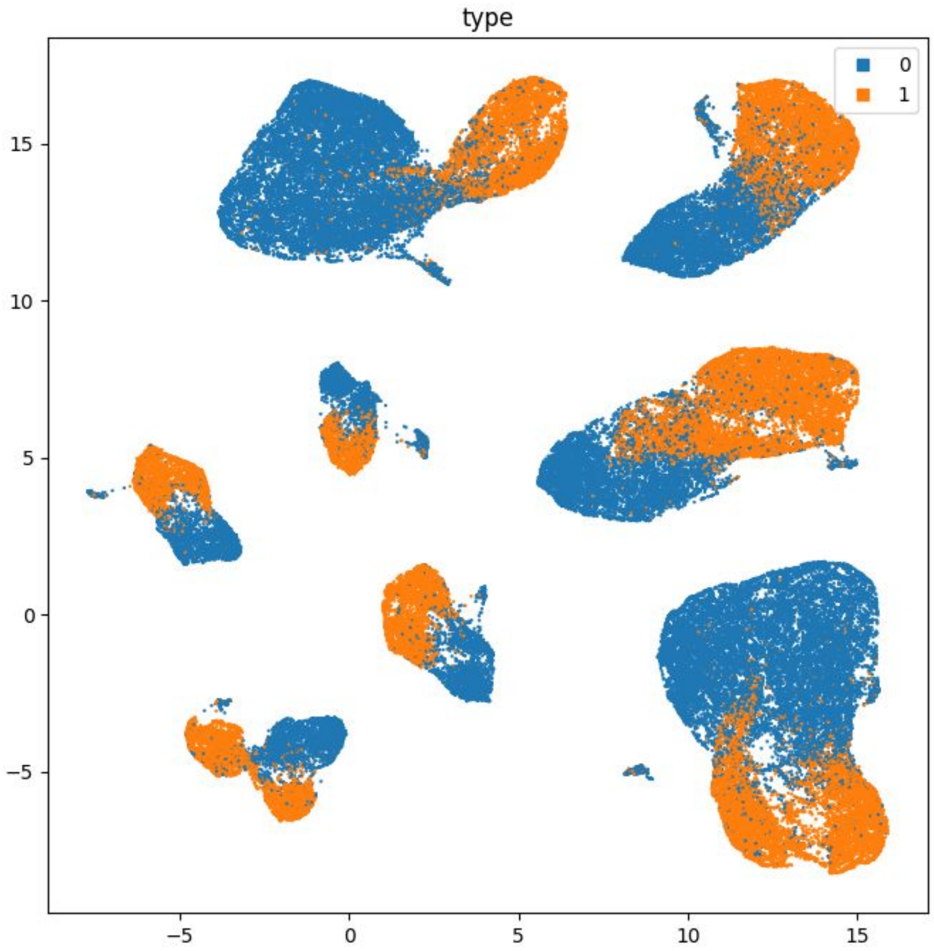
UMAP

Dimensionality reduction on a random sample of events with hu moment feature data only. Reduction to 2D with overlapping energies between particles and angles.



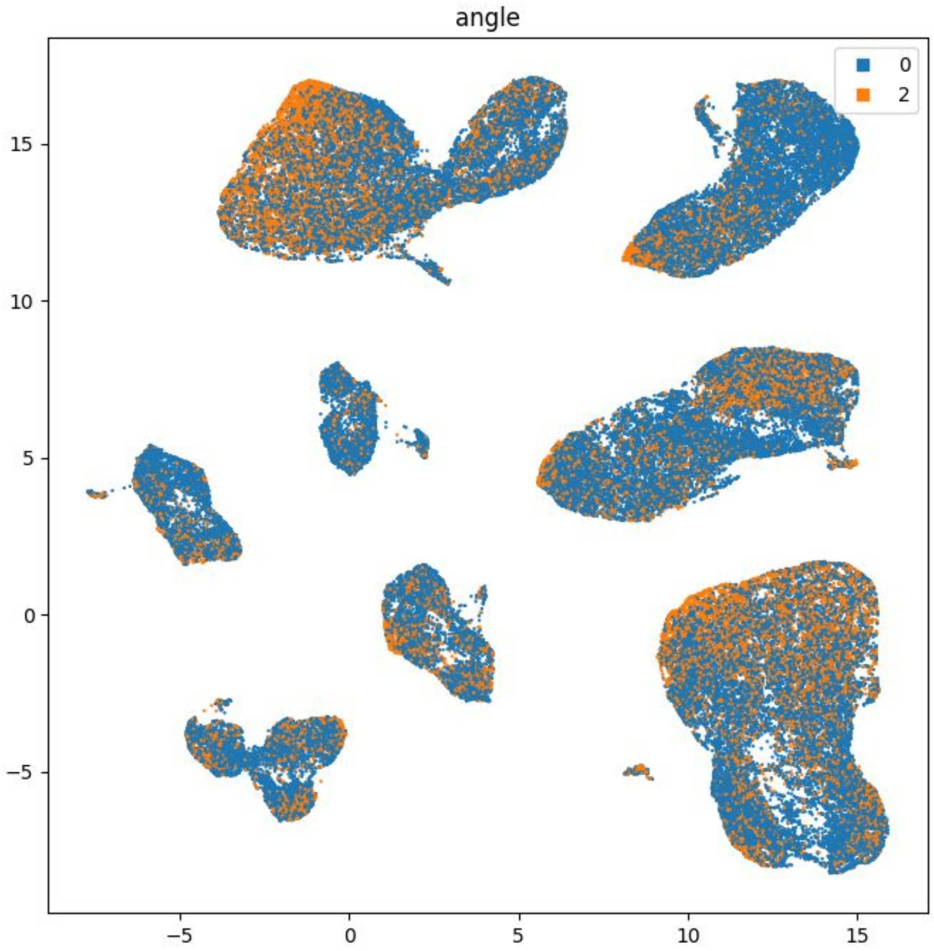
UMAP

Color coding according to particle.



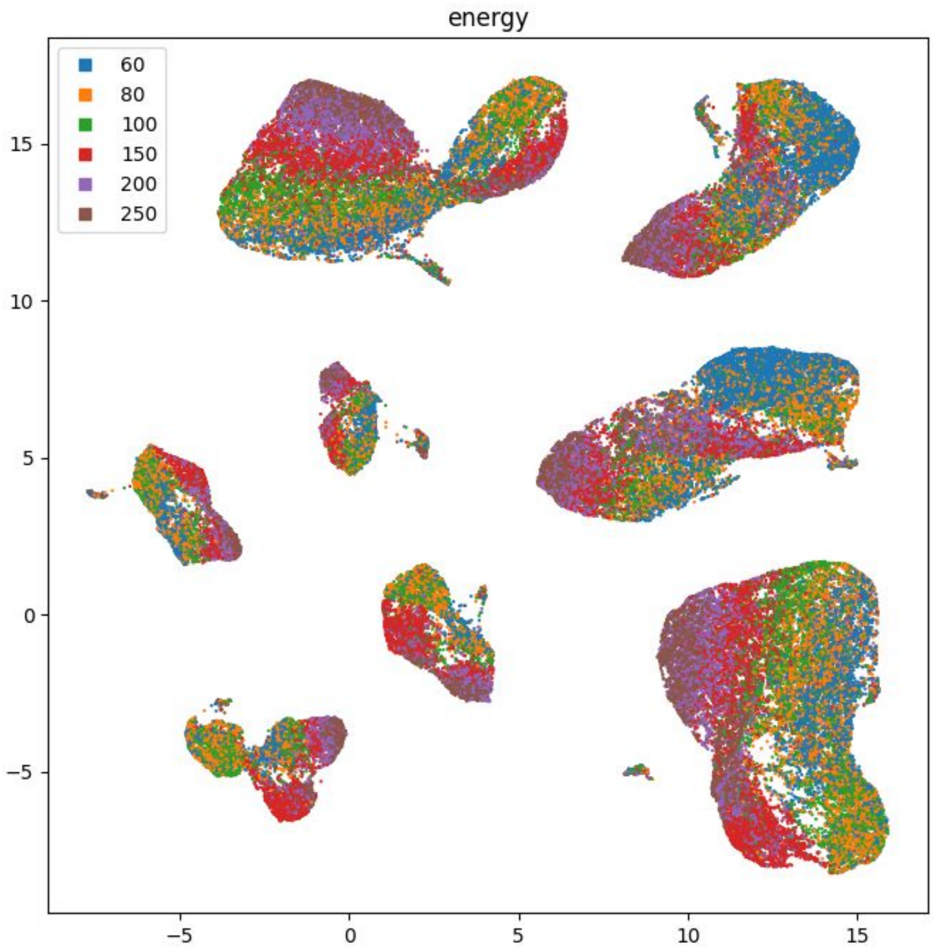
UMAP

Color coding according to angle.



UMAP

Color coding according to energy.



Performance Hu Moments

Binary Classification BDT:	Accuracy	logloss
Angle:	0.724	0.522
Particle type:	0.978	0.0707
Regression NN :	MAE	MAE (Linear Reg.)
Energy	19.5	29.27
	RelMAE	RelMAE (Linear Reg.)
Energy	0.1274	0.189

Regression on Energy: trained on everything but one energy

We trained on energies: 60, 80, 100, 150, 250, 300, 350

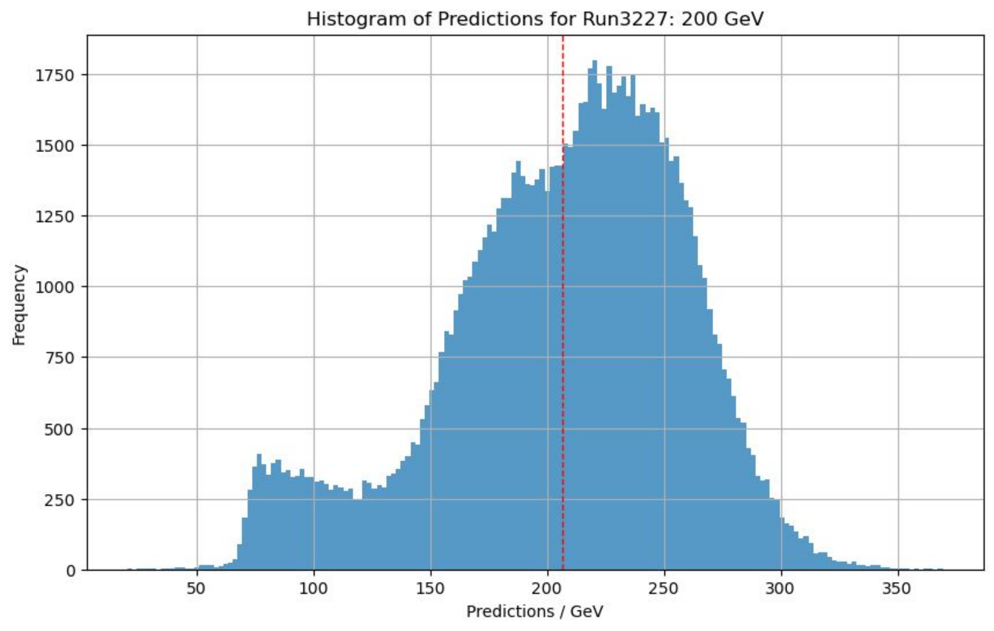
Executed on an all 200's run.
Small dip on 200.

With a mean absolute error of 42.2

Worse than the linear regression.

Implies:
model is only "good" when trained
on data resembling the data to test.

Could definitely be an issue when we
want to use calorimeter on real data.



Detector Angles: can machine see a difference between -2 and 2 degrees

Using only 6 first Hu moments.

Invariances: translational, rotational, scaling, reflection

Model discriminates between the two angles with
Accuracy: 0.74 and logloss 0.51

Why? Asymmetry in detector or data, wrong angle, or third?

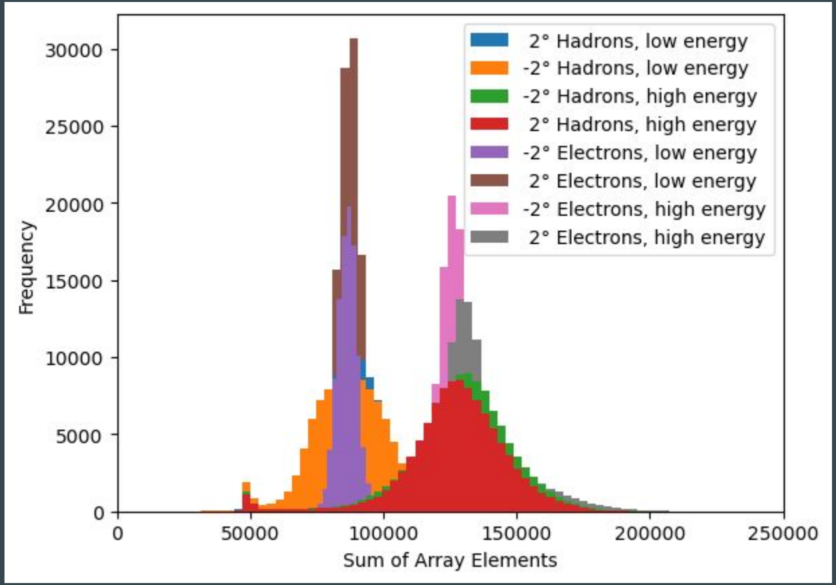
Training on One Energy, Running on Another

Training model to detect particle type only on low energy beams

Running model only on high energy beams

Statistical significance:
Accuracy: 0.61
Logloss: 0.87

Conclusion:
Shape of event is correlated to particle type



Autoencoder and Anomaly Detection

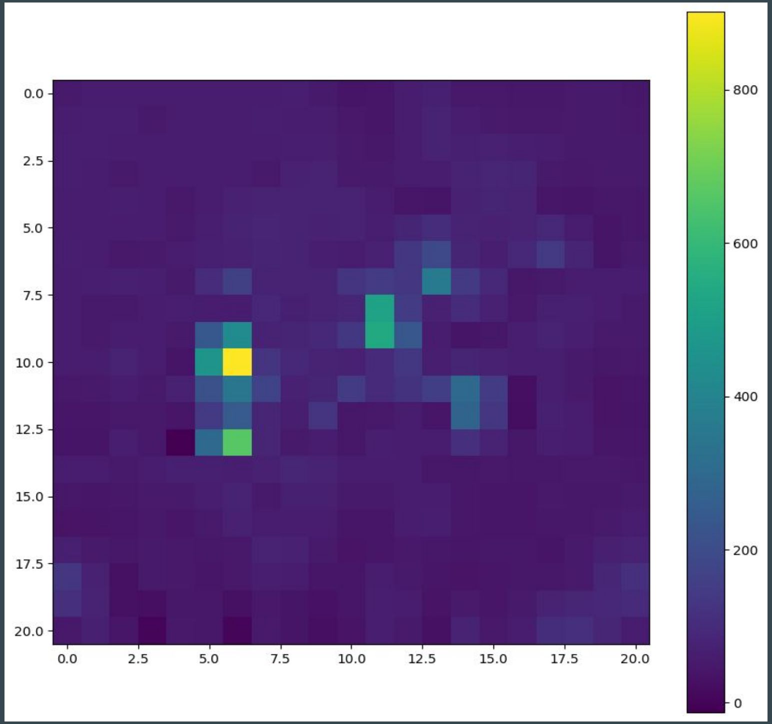
Autoencoder with an CNN

An event where an particle does not hit the detector

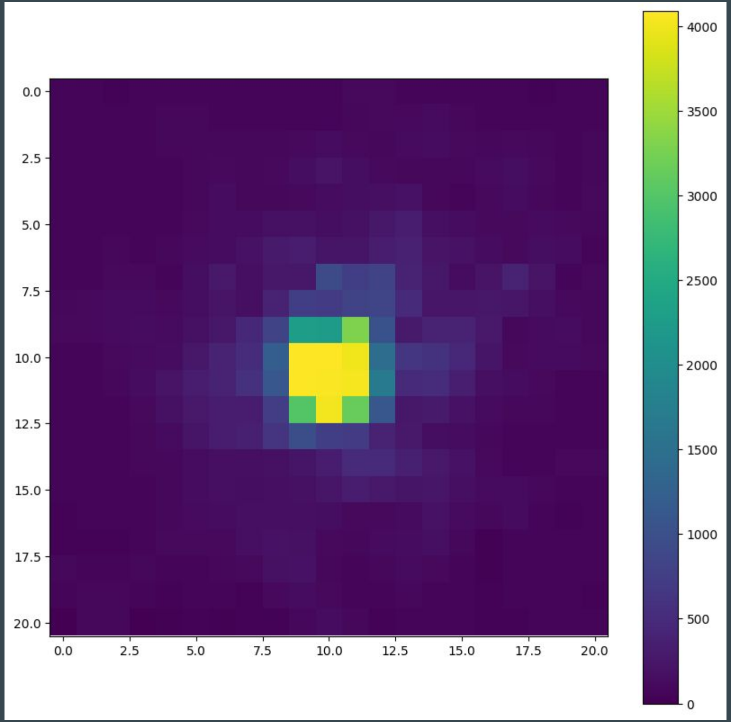
Did detect few anomalies

Threshold for anomalies: 5%

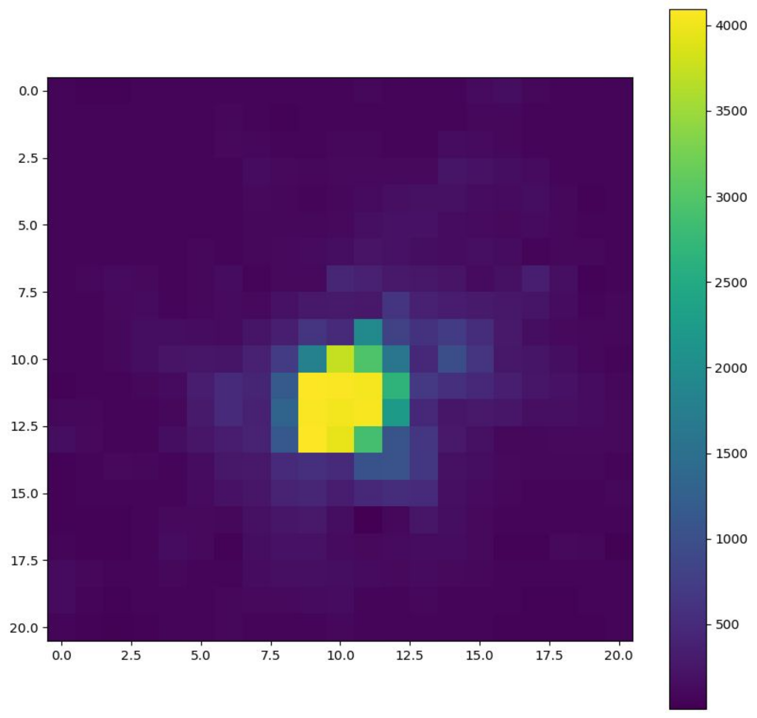
Detected events as anomalies, that are not anomalies



Anomaly Detection

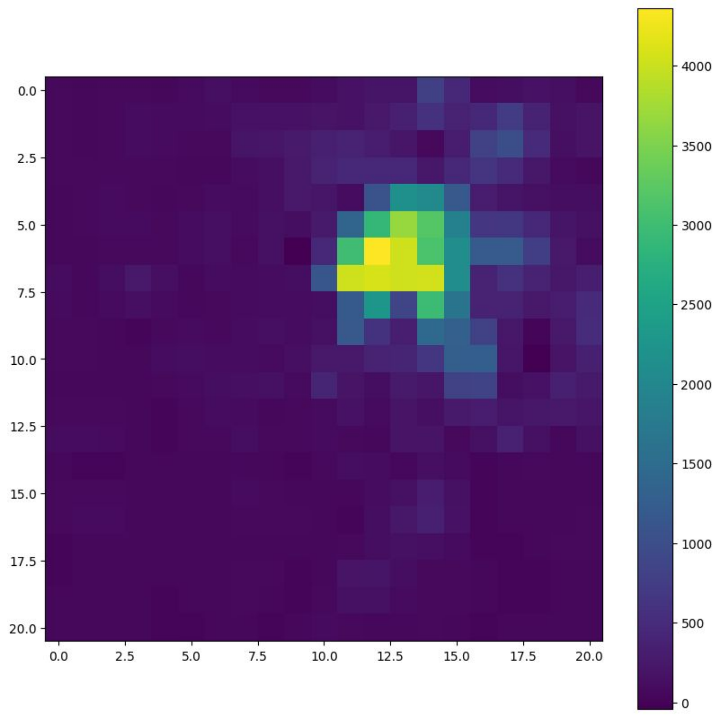


Detected as an anomaly

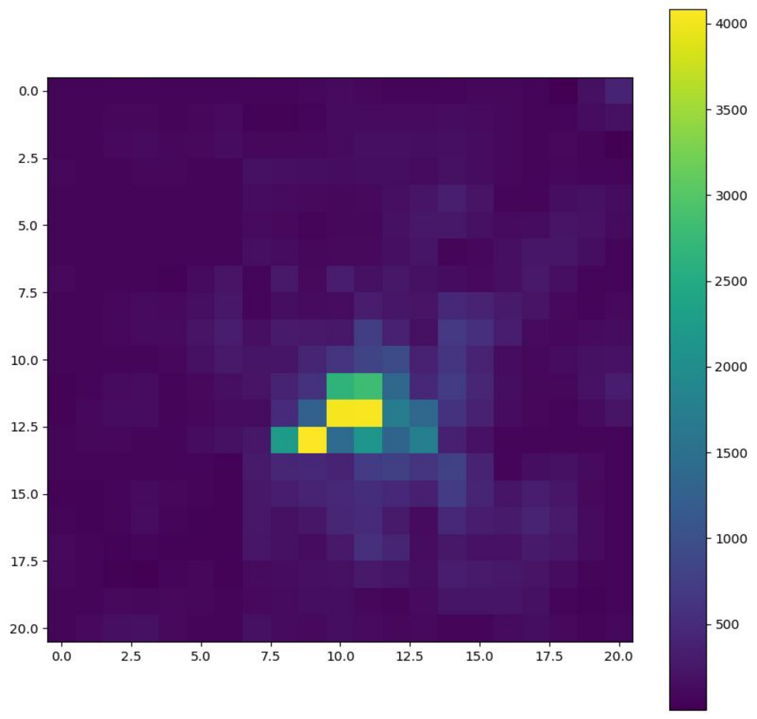


Not an anomaly

Anomaly Detection



Detected as an anomaly



Not an anomaly

Trying to Beat the Autoencoder: “semi-labelled” anomaly detection

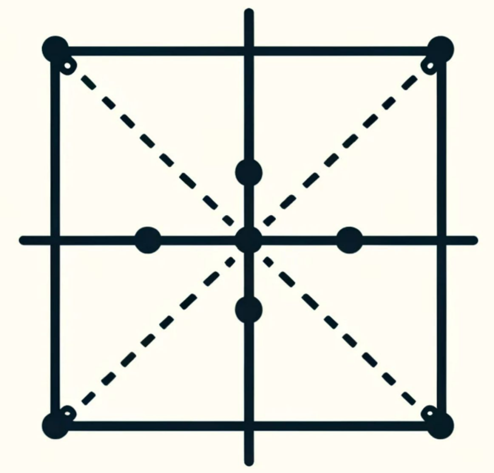
Looked through 15000 images. Filtered out anomalies (158).

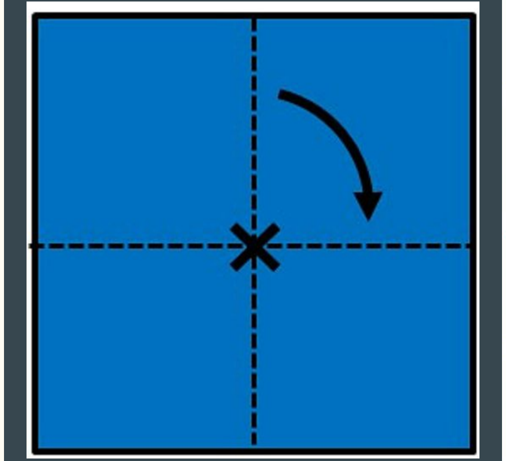
Autoencoder, convolutional neural network. No good.

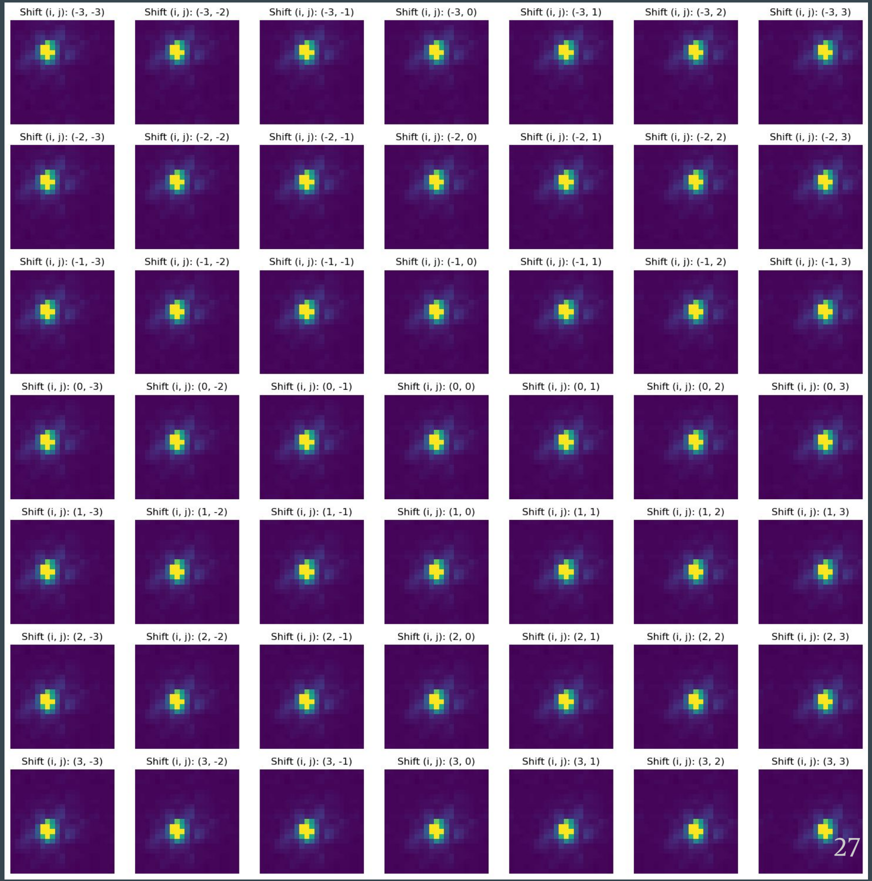
mirror about 4 + 1 axis, rotated each of those 0, 90, 180, 270

Shifted each event -3 to 3 pixels in both x and y

N = (15.000 - 158) · 5 · 4 · 7 · 7 = 14.545.160 (1000 times more than before)





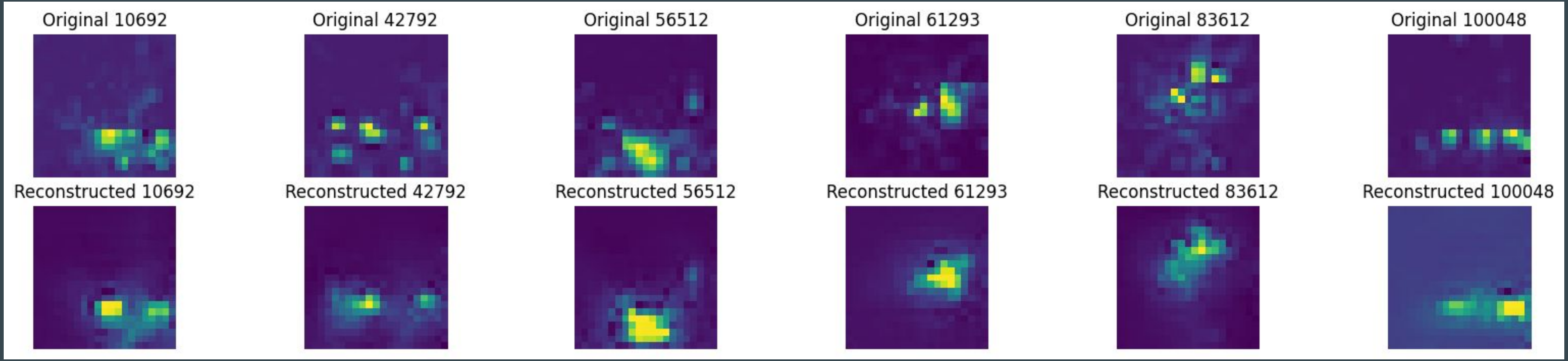


Trying to Beat the Autoencoder: “semi-labelled” anomaly detection

It finds good events that aren’t anomalies, and real anomalies.

Idea for future: Use the found anomalous good events for retraining, might converge to good model.

Could save a lot of time labeling data. When I went through 15000 images I saw very few “good” anomalies



Conclusions

Particle classification worked well

Energy regression was ok

Angle regression was poor

Anomaly detection was hard

Acknowledgements

Thank you to the FoCal-H group for allowing us to use the data.

Special thanks to Ian Pascal Møller for helping out and providing processed data ready for use in Python.

Appendix

Overview of methods

We do a mix of supervised and unsupervised training using both BDTs and NNs, and we compute our own features.

The BDT models are XGBoost and LightGBM.

Regression using vanilla neural network using PyTorch.

Autoencoder with CNN using TensorFlow.

Autoencoder with CNN using PyTorch.

UMAP for dimensionality reduction for data exploration and visualization.

Classification with moment features (particle type and angle)

xgboost.XGBClassifier with binary:logistic objective. Otherwise default parameters:

- learning rate: 0.3
- max_depth: 6
- min_child_weight: 1

Classified on moment features, no shower sum used.

Only events where electrons and hadrons share energy are used.

Notebook: BDT.ipynb

Regression with moment features (energy and angle)

xgboost.XGBRegressor with reg:squarederror objective. Otherwise default parameters:

- learning rate: 0.3
- max_depth: 6
- min_child_weight: 1

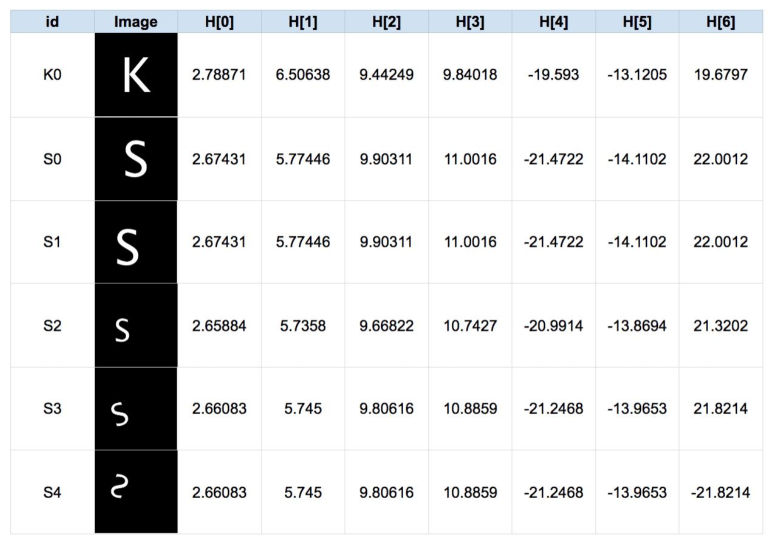
Classified on moment features, no shower sum used.

Only events where electrons and hadrons share energy are used.

Notebook: BDT.ipynb

7 Hu moments (classification for angle and particle type)

When looking at the Hue moment table here, it does look a little bit different for the scaled and the rotated images, this is due to the resolution not being infinite. When an image is scaled, if not done perfectly, it might take up just a few pixels more or less in different places than originally, because the pixels themselves are arranged in a square arrangement. The same goes for rotation. Those values for the scaled and rotated are almost the same, and completely fine for our use.



id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0	K	2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2	S	2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3	S	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4	S	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

7 Hu moments (classification for angle and particle type)

Used cv2 from opencv (open computer vision library) to calculate the 7 hu moments

Used both BDT (LGBM) and NN (Pytorch), LGBM gave, as expected, the best results for the classification, so I ended up using that.

I used SHAP values for feature importance, just for fun. 0th moment was most important for both classifications, but after that it became pretty mixed up. Running only on 0th moment was not good. Using all the moments was definitely the best.

Used optuna for hyper parameter optimization.

7 Hu moments (classification for angle and particle type)

Hyper Parameters:

Objective:	binary
Metric:	binary_logloss
Num_leaves:	31
Learning_rate:	0.1
Feature fraction:	1.0

7 Hu moments (regression)

Used cv2 from opencv (open computer vision library) to calculate the 7 hu moments

Used Neural Network (PyTorch)

Used built in feature importance

Used Optuna for hyper parameter optimization

Hyperparameters:

Learning rate:	0.001
Batch size:	64
Epochs:	10
First hidden layer	64
Second hidden layer	32
Activation function	ReLU
Loss Function	MAE

Trying to beat anomaly detection with an auto encoder with CNN

Took 15000 events from 15 different runs (1000 from each)

Looked through all 15000, and filtered out all the bad-event anomalies (158)
I took the ones left, and for each of those I mirrored them around the four reflection symmetry axis of a square, and left one untouched. Then I took each of those and rotated them 0, 90, 180, and 270 degrees. Each Of those I then translated from -3 to 3 pixel in both x and y. So instead of having just 14842 different events I had “synthesized” 14.545.160 events, which is about three orders of magnitude more than to begin with.

Then trained the model.

I then ran the model on a completely new run, that hadn’t been used for training, I tweaked the threshold, and it found quite a few anomalies(540), some actual anomalies, some good-event anomalies (which we want to keep). There was definitely a trend when looking at those found anomalies.

I then looked through all the newfound anomalies, filtered the bad ones out, added them to the old training set, and now all that would be needed would be to train the model again with both the old and the new events. Then I’d run it on a completely new run again, to see how it performs there.

Trying to beat anomaly detection with auto-encoder and CNN

Hyperparameters

Batch_size = 256
Auto Encoder architecture:
Encoder:

```
nn.Conv2d(1, 16, 3, padding=1)
nn.ReLU()
nn.MaxPool2d(2, padding=1)
nn.Conv2d(16, 8, 3, padding=1)
nn.ReLU()
nn.MaxPool2d(2, padding=1)
```

Decoder:

```
nn.Conv2d(8, 8, 3, padding=1)
nn.ReLU()
nn.Upsample(scale_factor=2)
nn.Conv2d(8, 16, 3, padding=1)
nn.ReLU()
nn.Upsample(scale_factor=2, mode='nearest')
nn.Conv2d(16, 1, 3)
nn.ReLU()
nn.Conv2d(1, 1, 2)
```

Epochs: 20
Loss function: Mean squared error
optimizer = optim.Adam(model.parameters(), lr=0.001)

Run on GPU in google colab. A100

Threshold for anomaly detection =
np.mean(reconstruction_errors) + np.std(reconstruction_errors) * 3 =
0.00158

Autoencoding with CNN

CNN used for autoencoding on the image data

Hyperparameters:

Input shape:	(21, 21, 256)	size of image, and number of channels
Batch size:	32	
Epochs:	tested on different values, and found 50 to be best	
Loss function:	binary_crossentropy	
Optimization:	Adam, with a learning rate of 0.001	

DataGenerator used to generate batches of training and validation data

Normalised the data by dividing by 4095, the max value for the ADC count

Split the data 80/20 with 80% being the test data and the 20% being the validation data

Anomaly Detection with the Autoencoder

A function to calculate reconstruction error were the mean absolute error between the original data and the reconstructed data across the channel dimensions was calculated. This function also return an error for each of the event.

The data was processed in batches, by reshaping batches of data to match the input shape from the autoencoder. By using the function for reconstruction error, and error for each batch was calculated, and lastly the threshold of 5 % was set

The anomalies was identified by comparing the error with the threshold, but the model was able to identify the true anomalies

Hyperparameters:

Batch size: 256, the number of channels

Threshold: 5%

Julie