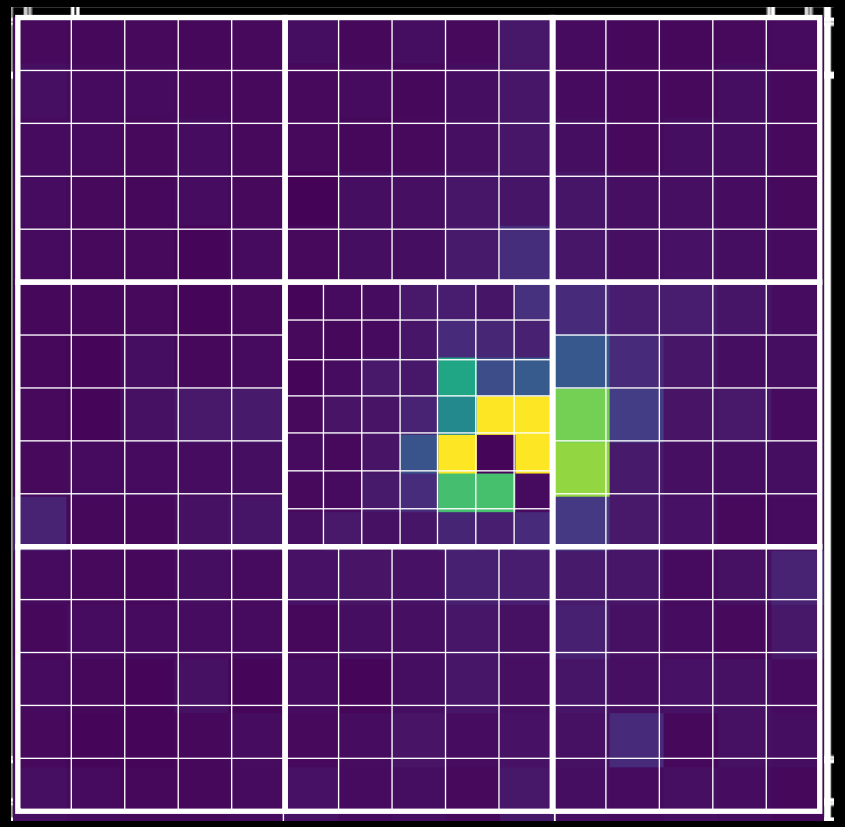
Particle classification on data from a calorimeter, using GNN

By
Zakarias, Mark and Frederik

All contributed evenly to the project

Present the data

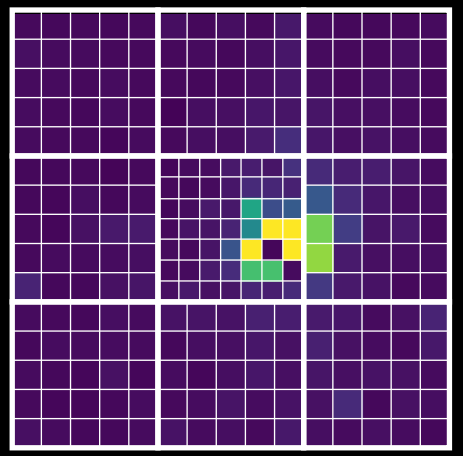
- 19 files of 100k+ events each.
- 10 files of hadrons, 9 files of electrons.
- Each file a new energy.
- Data has no NaNs
- Format is 256 ints (where 249 is used, i.e non-zero) per event. Each int is 13 bits
- Each int represents a pixel in a non-uniform image, where the pixel density in the center module 7x7 and the density in the surrounding 8 modules is 5x5



Due to this uneven pixel granularity, we couldn't just use a traditional CNN.

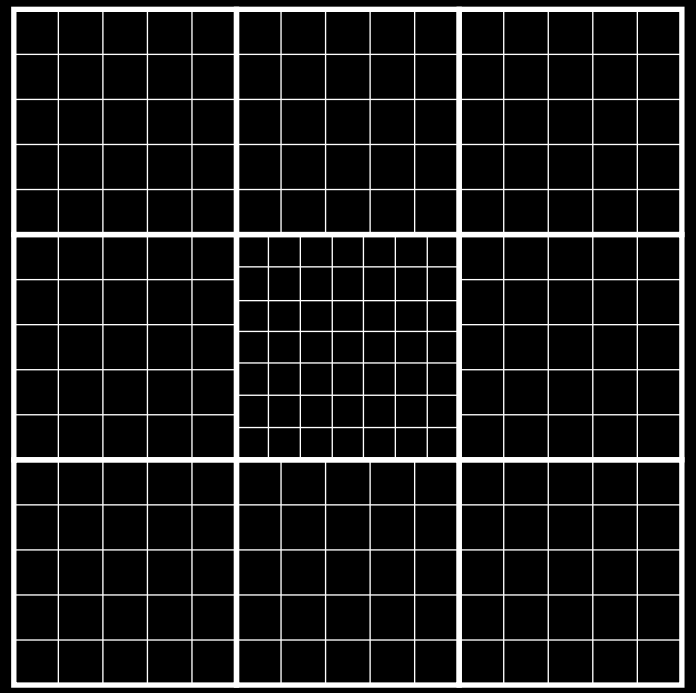
Seeing the bigger picture

[6132, 214, 332, ... , 165] →



- The data itself came as a 3D NumPy array.
  - Type, Energy and Position
- We focused on High Gain events, so technically 2D.
- So we had to go from a list to a “picture”/grid somehow.

- Which required a mapping, thank you to Ian for providing this! (Shout out Ian Pascal)
- We could then map the raw data to the correct pixel.



- More or less like this toy example:

[3 , 10 , 7 , 9 , 1 , 2 , 1 , 3 , 1 , 5 , ...]

Present the data

- The objective is to determine if the particle that hit was an electron or a hadron, based on the shape of the impact.
- Hadrons should interact more and leave a wider impact
- There is a wide variety of impact patterns even in the same runs. The impact patterns change character with energy and particle type but not dramatically.

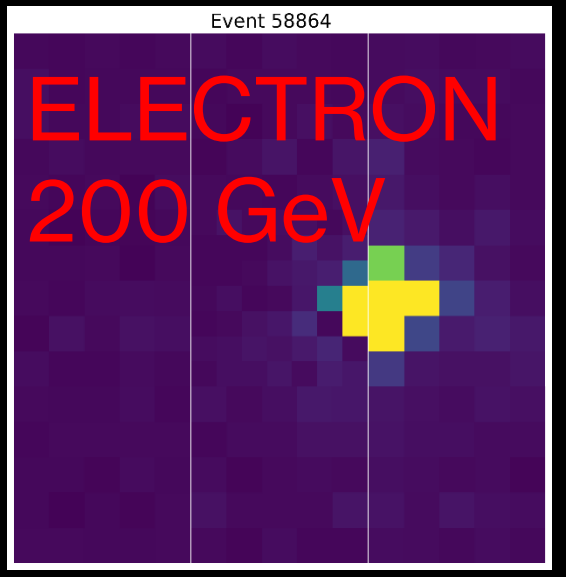
Let us demonstrate by playing a game

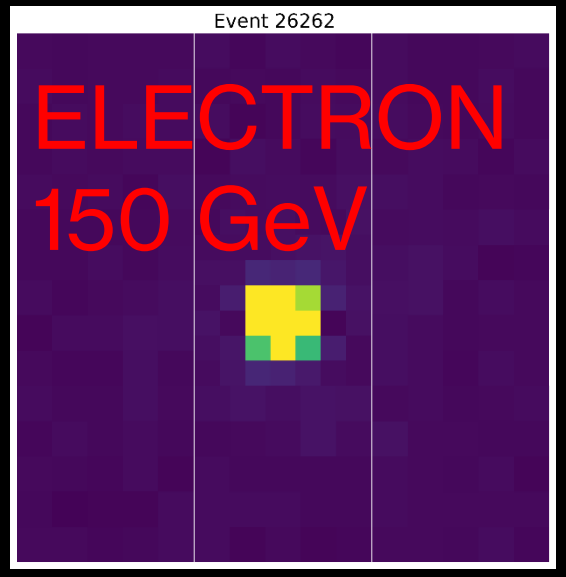
Zak

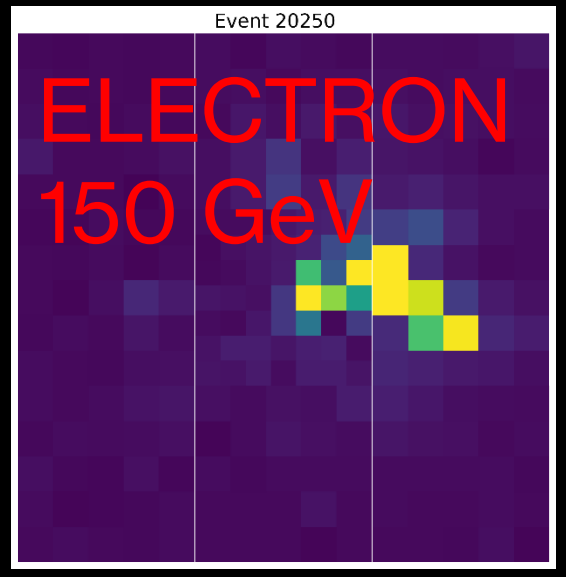

Event 13206
HADRON
80 GeV


Event 84436
HADRON
100 GeV


Event 53750
HADRON
100 GeV


Event 58864
ELECTRON
200 GeV


Event 26262
ELECTRON
150 GeV

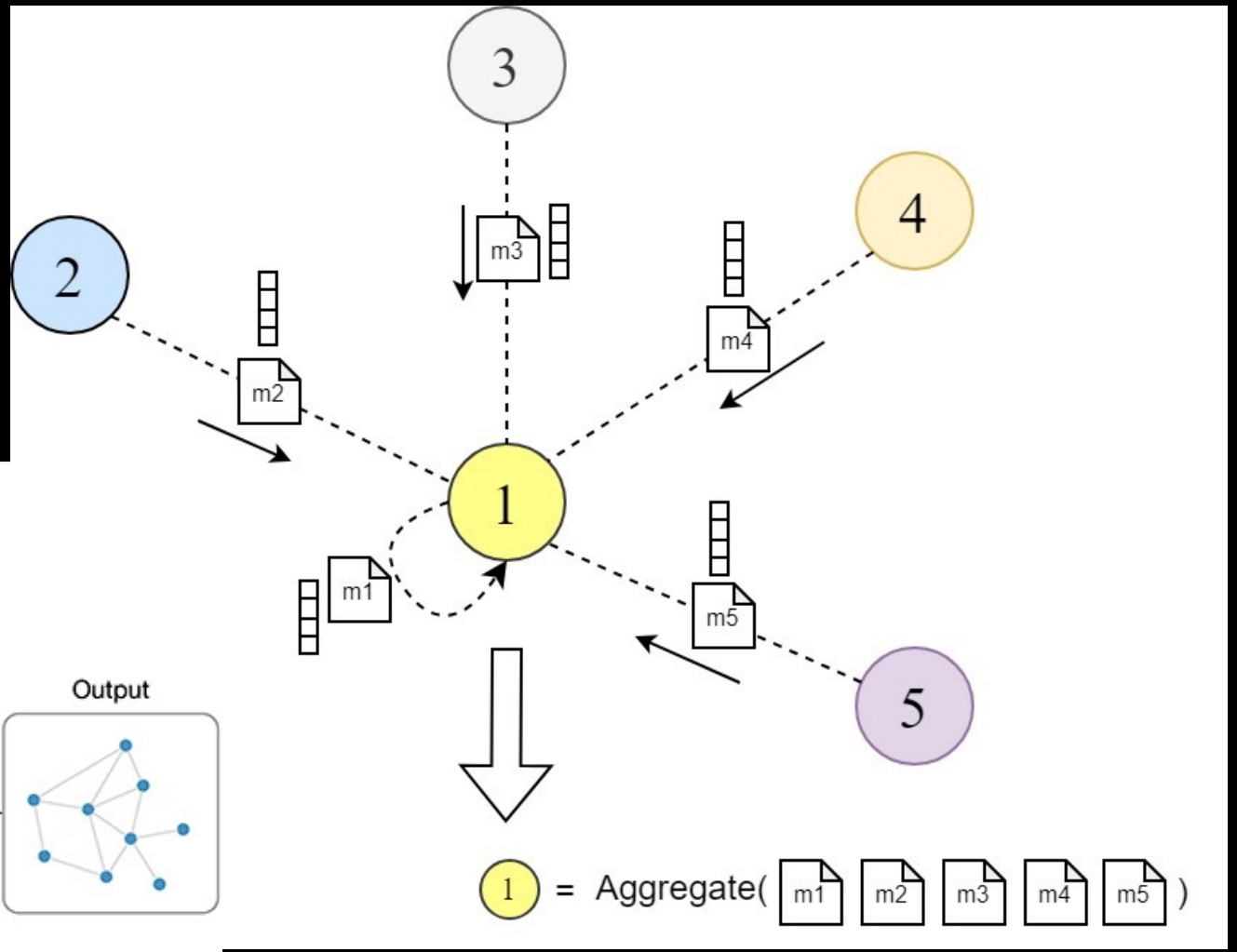

Event 20250
ELECTRON
150 GeV

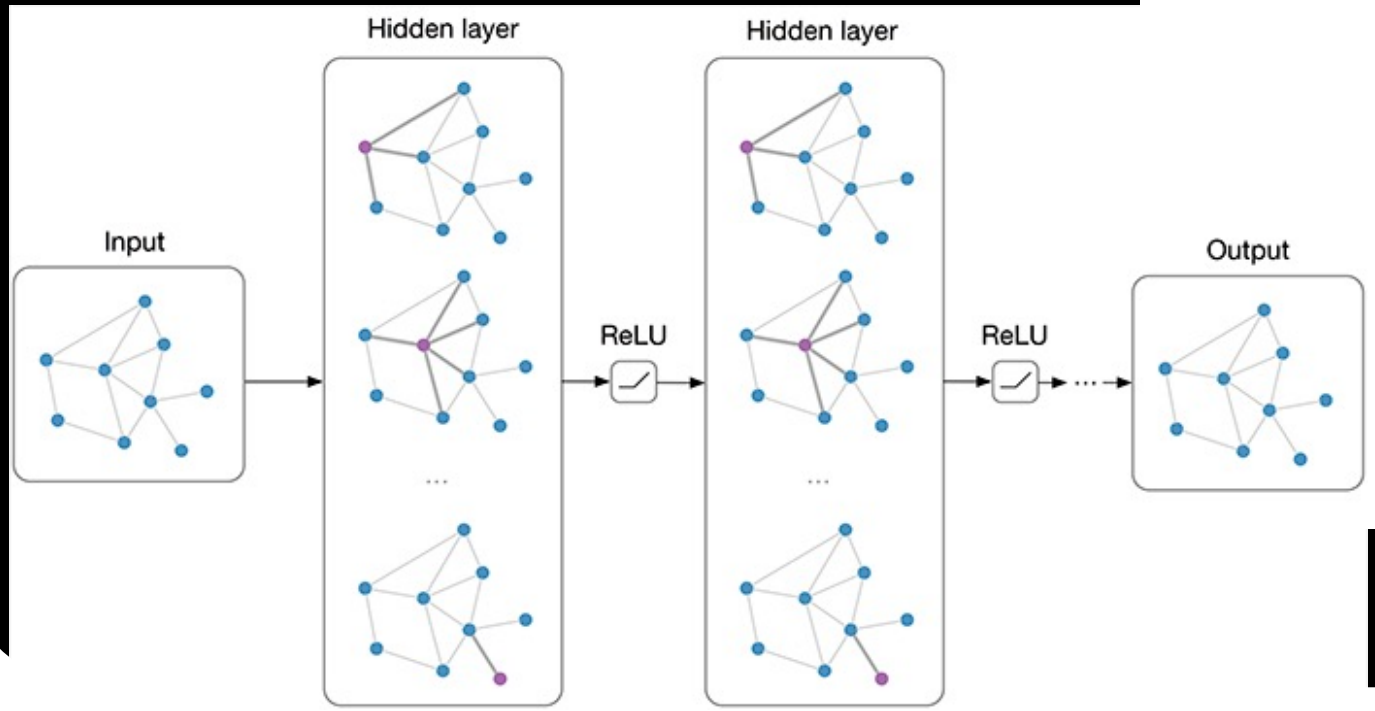

Event 85643
HADRON
200 GeV


Event 78366
HADRON
200 GeV

GNN

Message Passing



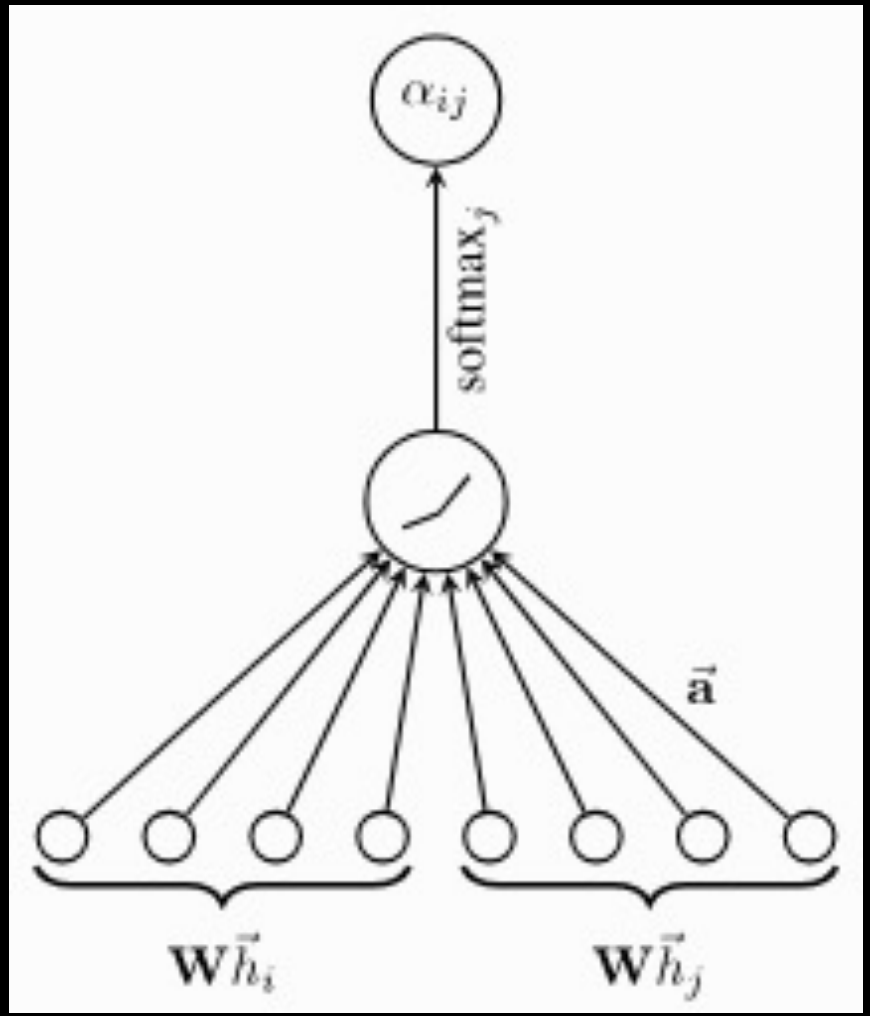


Thomas Kipf: ” Graph Convolutional Networks”. 2016, tkipf.github.io

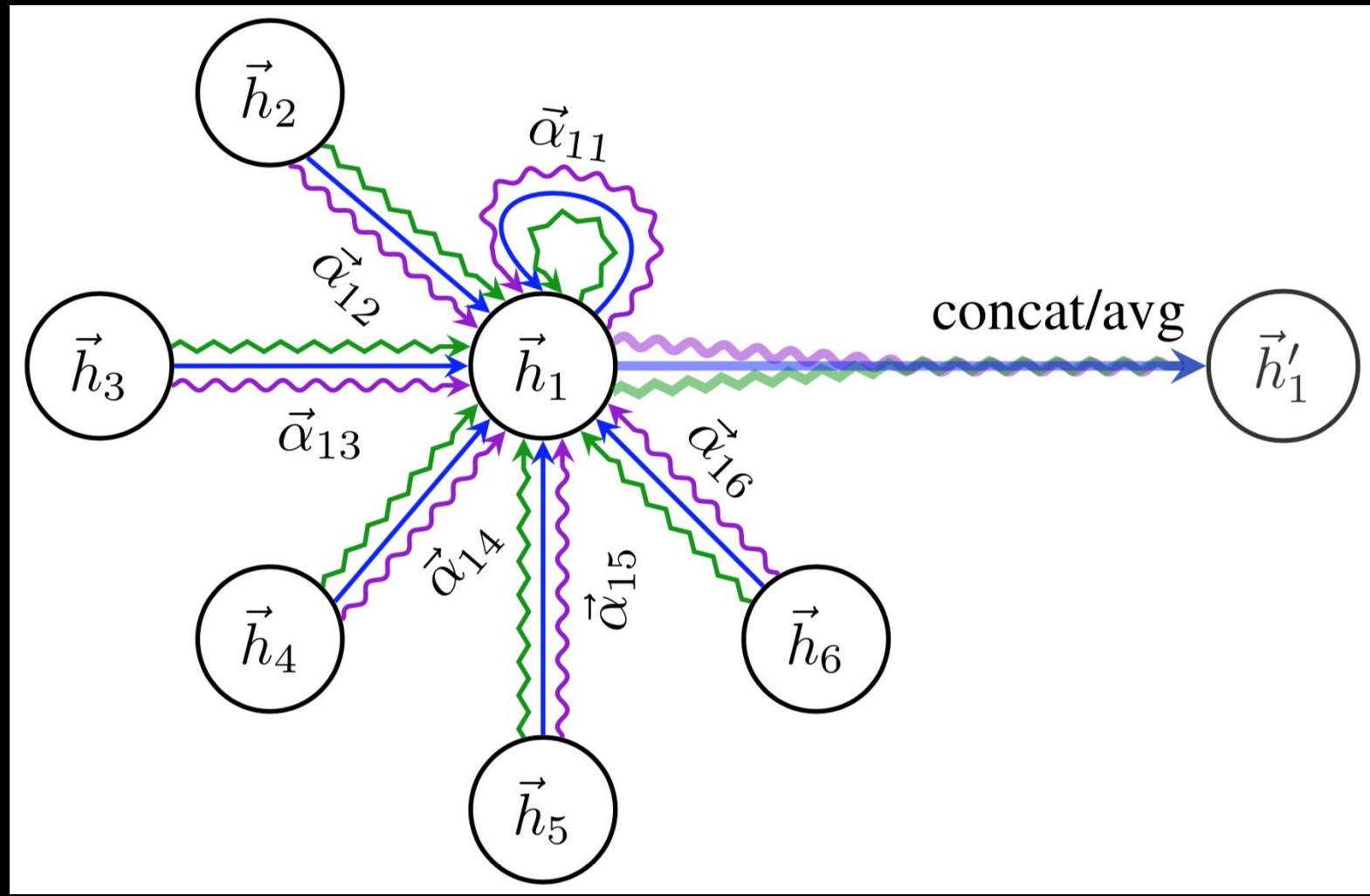
Mark

GNN

Graph Attention

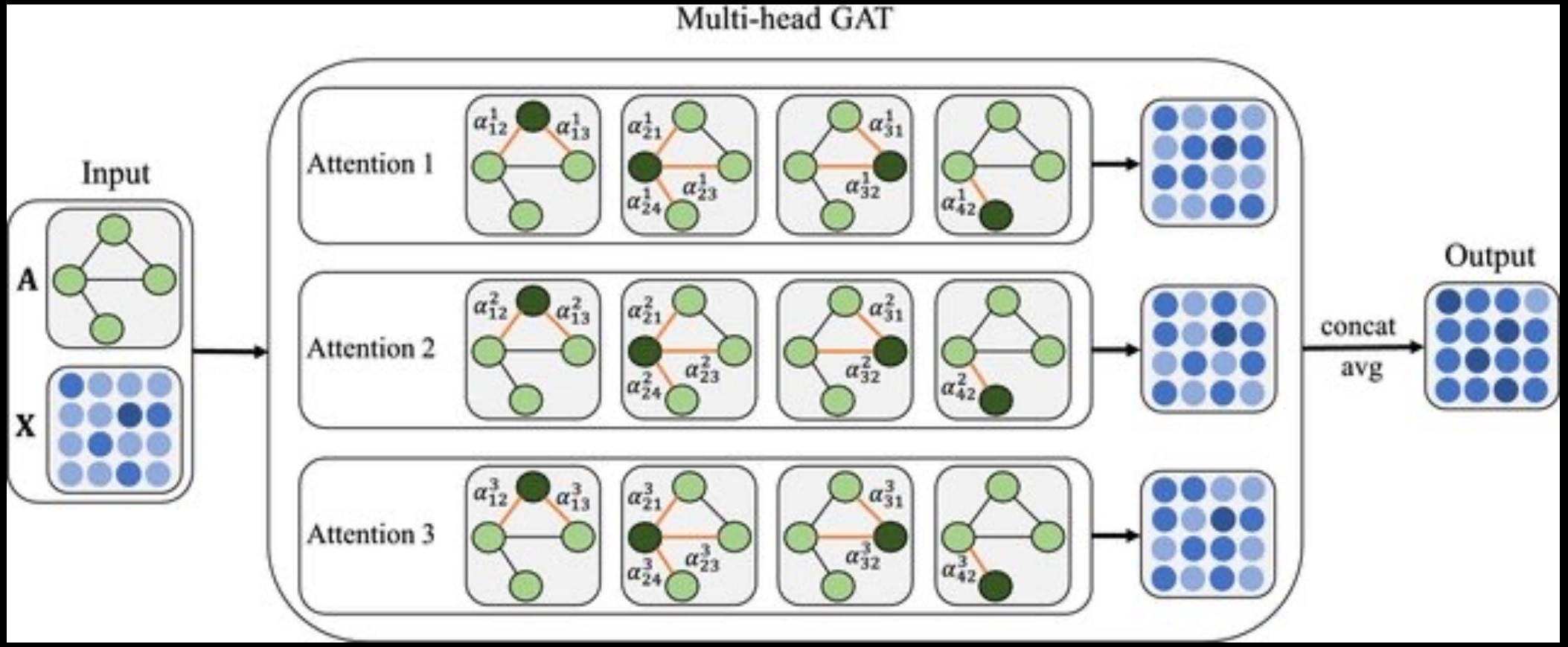


Multi-head Attention



arXiv:1710.10903 [stat.ML]

GNN



10.1007/s00138-021-01251-0

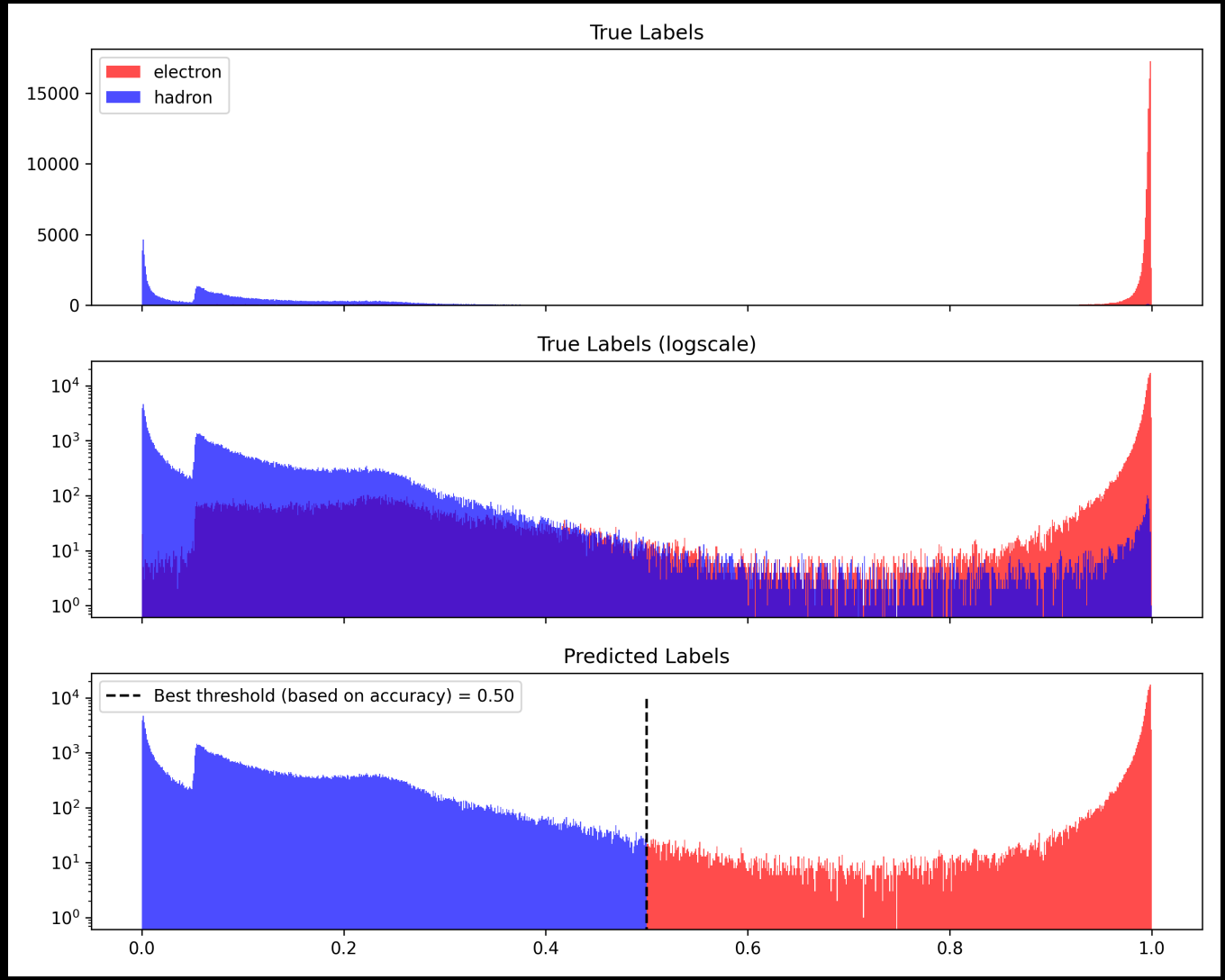
GNN

When training on the full set, the best cutting point turned out to in fact be the default 0.5

We can also clearly see some mixing, though we still have good general separation.

It is clear that this data is just not forgiving as we saw.

A curious hump is also visible in the hadron predictions.



GNN

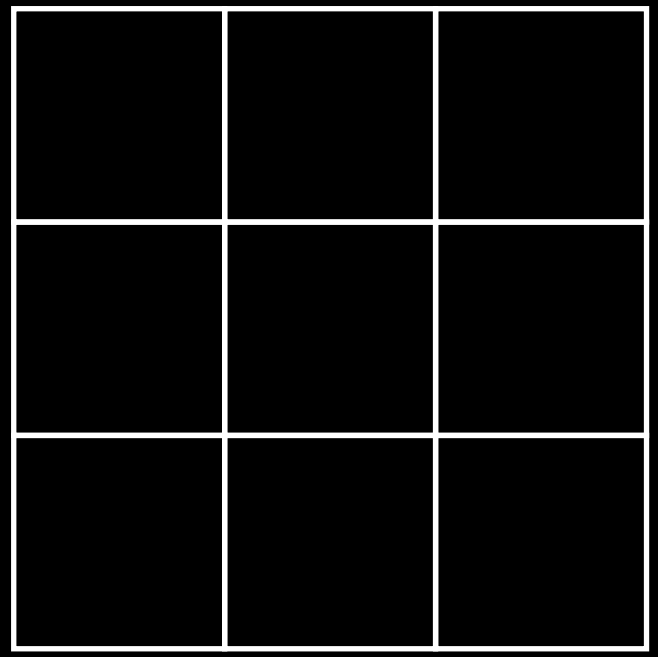
- We used 25% of the data when hyperparameter optimizing (HPO) in order to speed things up.
- The data used was a randomized mix of all energies and of course both particle types.
- Training still took several days
- Lastly, we then used only the best hyperparameters and retrained the model on all the data (again this took days)
- Going from 25% to 100% of the data did not increase the accuracy, hinting to something else being the issue
  (we will return to this)

Alternative approach

- A member of the group, is a good example of being a Jedi.
- The detector is a 3 x 3 grid, with different granularities
- A traditional CNN doesn’t work because of the granularity difference.
- However, the granularity shouldn’t be an issue if it were to be ... removed?

A Jedi:

- *Someone who wields the force.*
- *Someone who can say stuff about a domain, without any prior knowledge, using ML.*



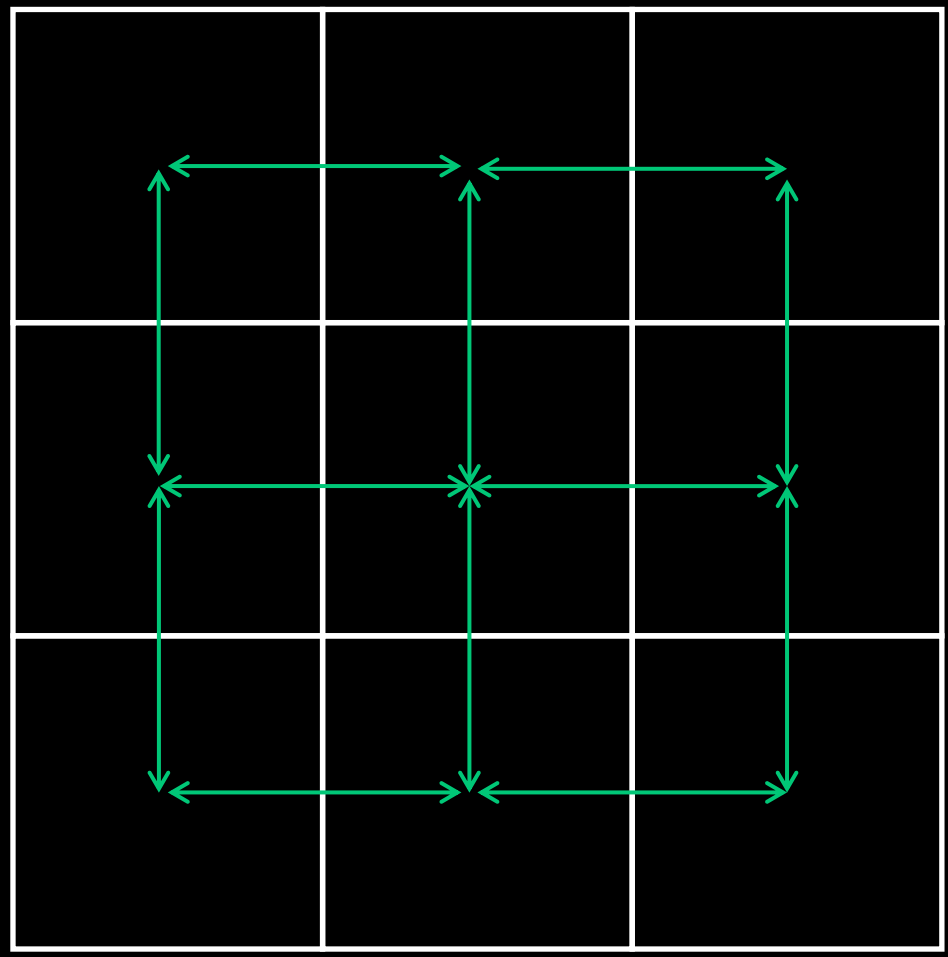
A new hope/model

- Instead of treating the detector as the a whole, it was split up into the 3 x 3 sub grid.
- We then trained a CNN on each sub grid, and used the output of each of these sub models for the final layer of the model.

CNN 1	CNN 2	CNN 3
CNN 4	CNN 5	CNN 6
CNN 7	CNN 8	CNN 9

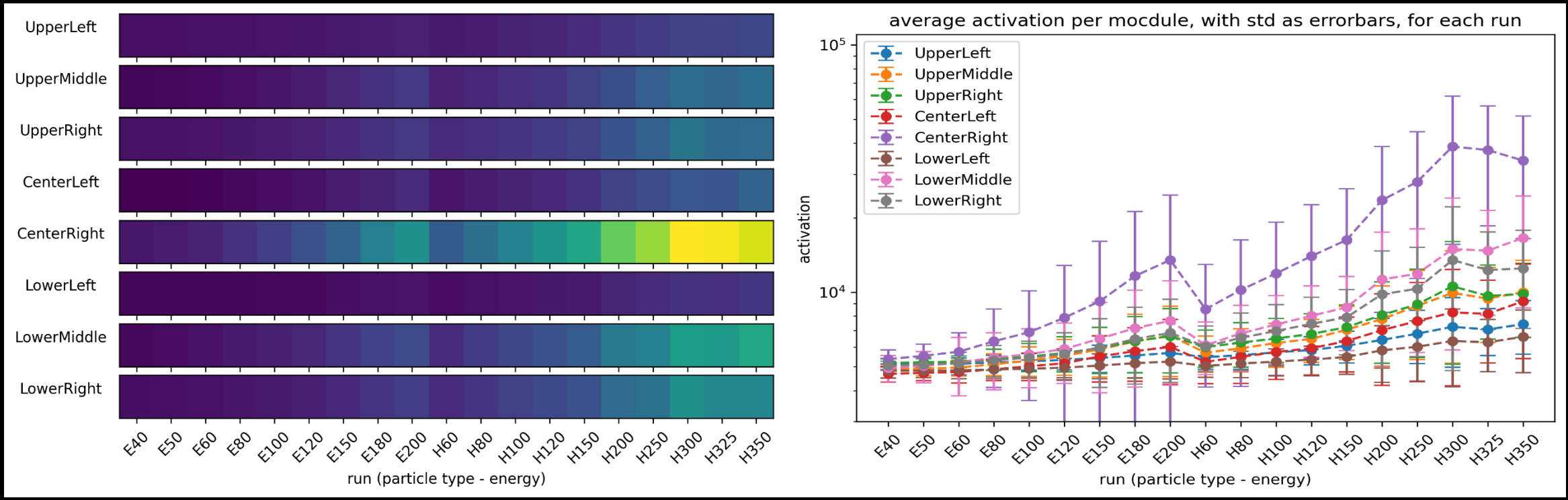
Return of the GNN

- The final layer of the model was a GNN, which related the sub grids to each other and took their outputs and used them as inputs.
- This hybrid model could then also be used to classify electrons vs. hadrons.
- This final layer could also have been a CNN, but twice the algorithms – double the learning!



Issues and observations

We also found that the activation of the different modules was not uniform. The beam was targeted on the center, yet we found that the impacts drifted much more to the right of the detector than any other. We think this is due to the fact that the detector was turned 2° with respect to the beam, so the impact would mainly shower onto the one side of the detector.



Issues and observations

We know that we have some mislabeling. The test beam is never 100% pure, so some interactions in the hadron folder might really be electrons, and so some of the “false predictions” might really be true predictions with bad labels. This could explain why more data does not improve performance

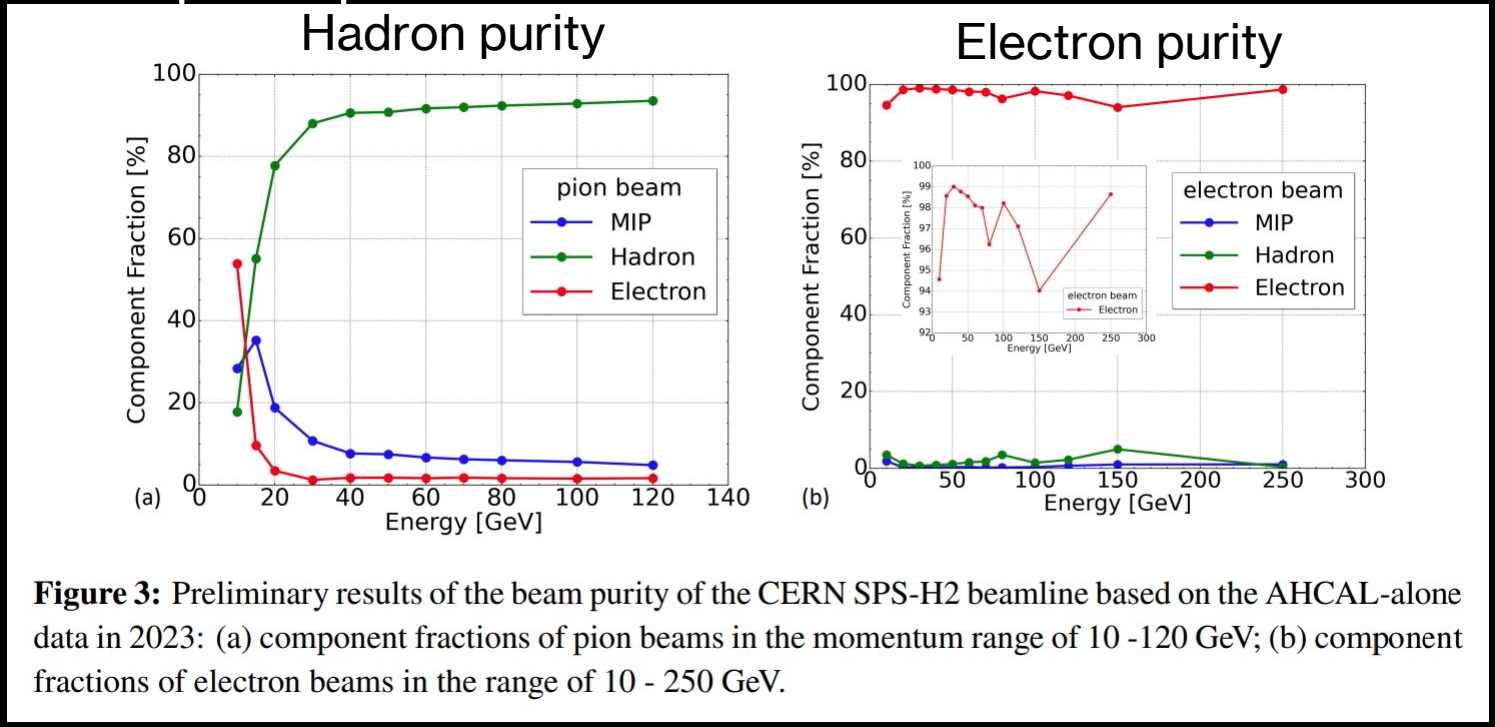
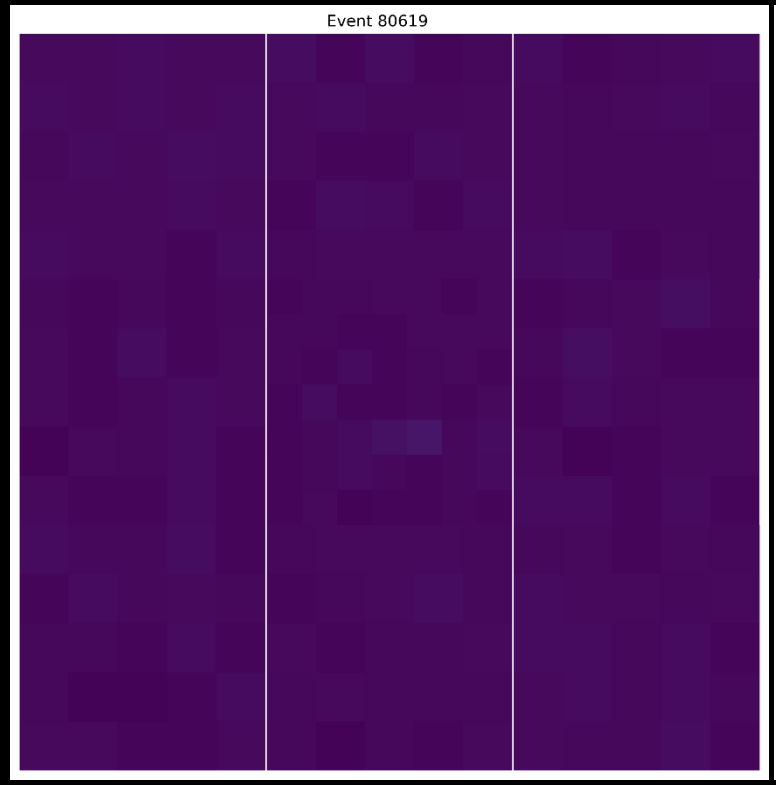
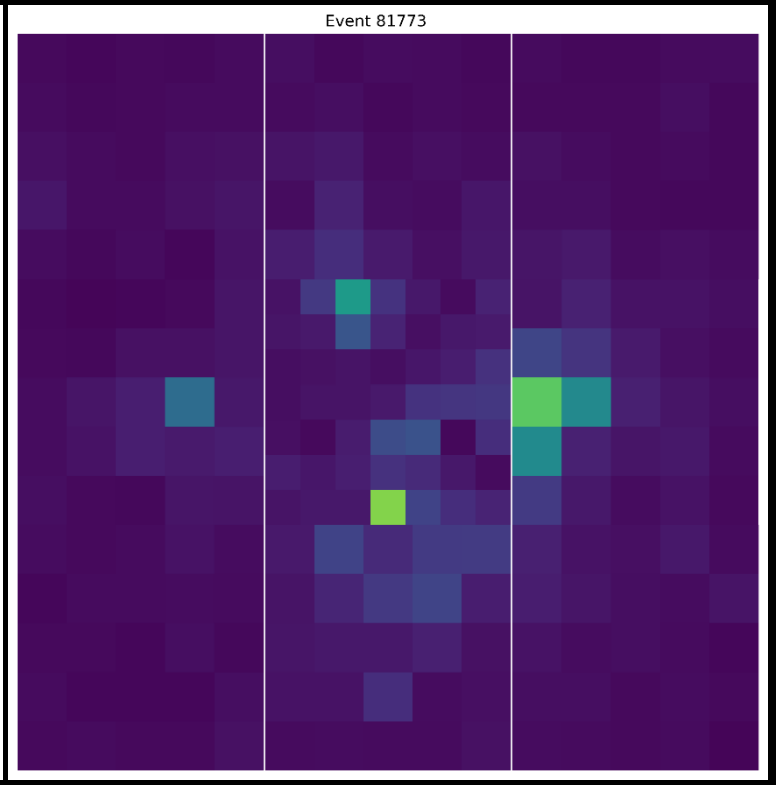


Figure 3: Preliminary results of the beam purity of the CERN SPS-H2 beamline based on the AHCAL-alone data in 2023: (a) component fractions of pion beams in the momentum range of 10 -120 GeV; (b) component fractions of electron beams in the range of 10 - 250 GeV.

Issues and observations

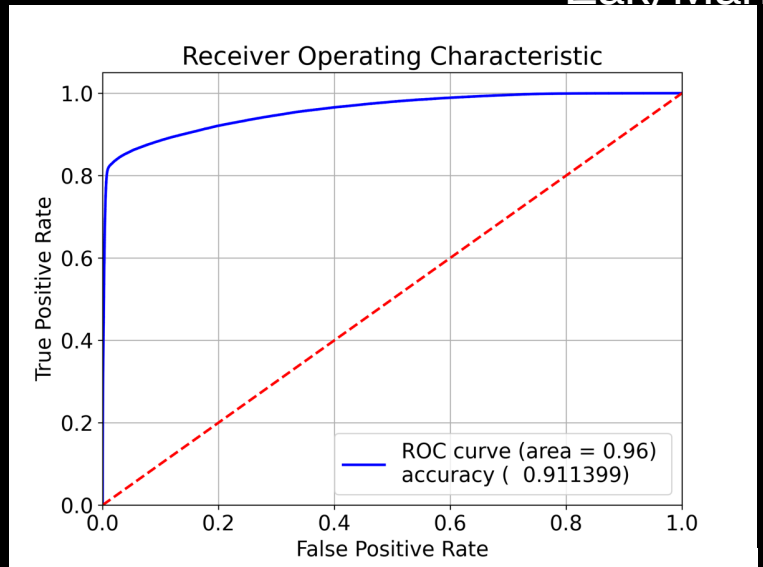
Additionally, we also found out too late that some of the events seem to be false triggers, as they have no impacts or such sparse activation that they seem to be impossible.
Had we found this earlier we would have pruned these out by simply removing outliers with too low total activation

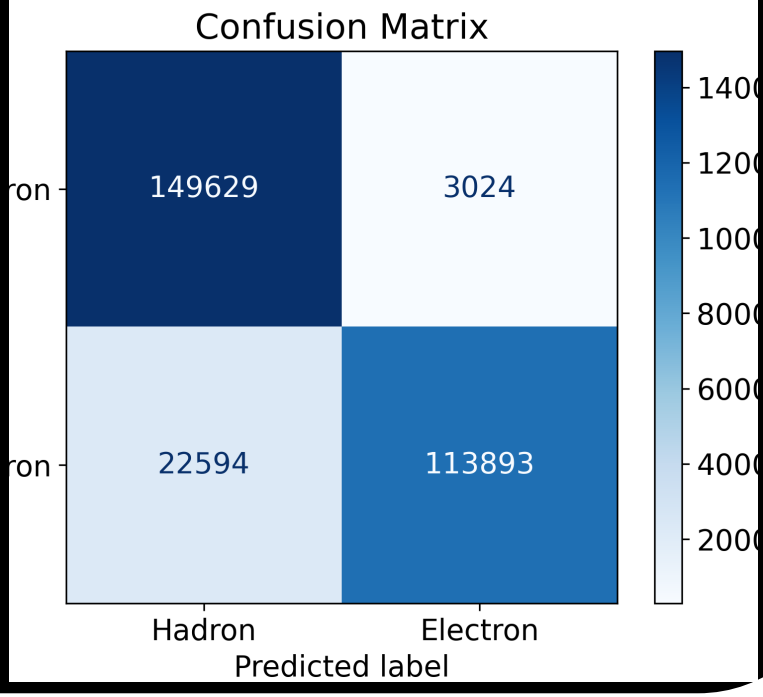




Performance GNN

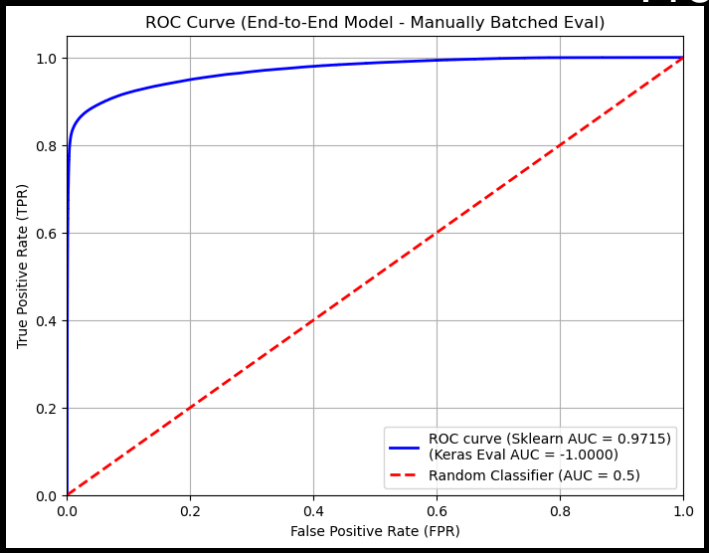
- HP Optimization: Yes
- Optimized using CUDA: We tried, but didn't make it work in time
- Final training time (for 36 epochs): 27.68 hours
  not including HPO epochs
- Validation accuracy: 0.9114
- Validation AUC: 0.9581

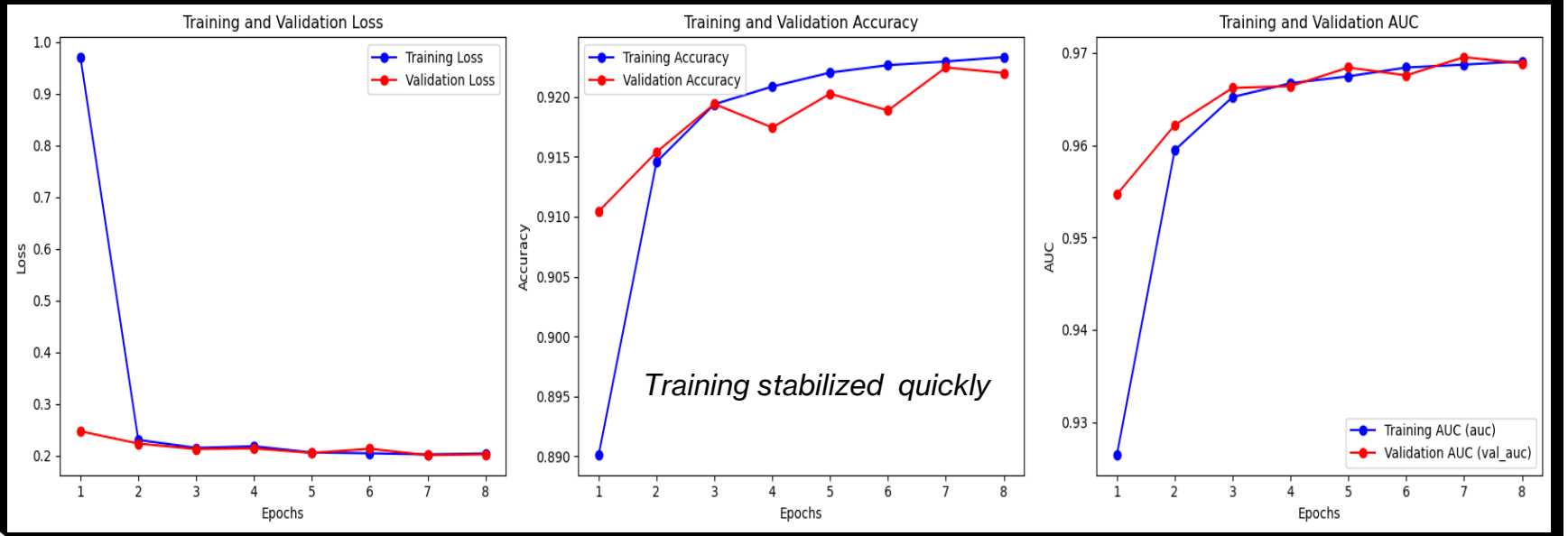


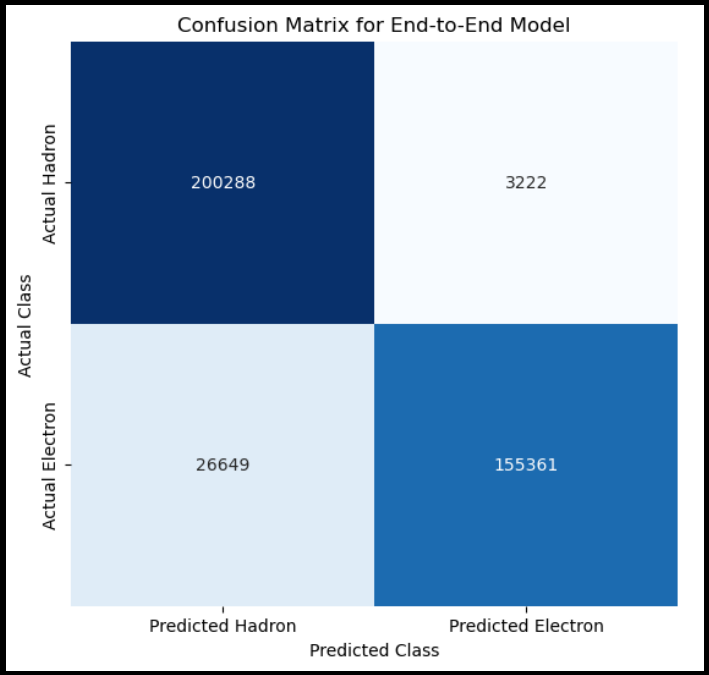


Performance Hybrid

- HP Optimization: No, but very possible
- Optimized using CUDA: No, but also very possible
- Training time (for 10 epochs): 140 minutes
- Validation accuracy: 0.9226
- Validation AUC: 0.9715







?

Conclusion

- The GNN model and the Hybrid model both performed so well that we might almost call it a photo finish, though the Hybrid model did beat the GNN by a tiny margin.
- It is important to have well labeled data.
- This was a highly iterative process
- A fun challenge of using different ML tools to solve a very real problem. Great example of a problem which can be solved in many ways!

?

Possible changes

To fix the issue of improper labels, one might preprocess the data, using unsupervised methods to remove outliers. Perhaps clustering or autoencoding could be utilized.

Alternatively, we maybe could have written our own loss function, that uses the graph of the beam purity as weights, to lessen the impact of guessing wrong when the energy is low and impurities high.

We could also have used a non-binary classifier, a trinary classifier, that sorted electrons, hadrons and "other" which might just be bad events but could perhaps also catch MIPs. For this model we assumed the data to only contain two types so not electron = hadron, though this is not entirely true.

APPENDIX

Project is in 3 parts

- S00 – Batch processing multiple .npy (Numpy) data files into combined HDF5 data files.
- S01 – Pre-processes the output files from S00 into graphs to input into S02
- S02 – Contains the actual GNN model; optional Optuna HPO and training/validation/testing

The Librarian s00_convert_npy_to_hdf5.py

Main purpose of this script is to collect the data from both electron beam and hadron beam files, for all energies, and make it into one easy to use file. With labels depending on their directory of origin.

At the same time this discards any unused data that is given with the set (low gain, TOT and TOA), thus we now have a collection of single flat vectors for each event

We also generate a sha256 checksum file to make sure there are no corrupted files

The file has two output modes, known and unknown, as we had 3 folders, one for hadron runs, one for electron, and one simply marked 'forgor' which was dubious to assume (we did not have access to the logs for most of the project)

The Architect s01_prepare_gnn_data.py

This file takes the flat vector-events from s00 and reconstructs them, using a channel mapping file given to us from Ian Pascal, into their true 2D form

It identifies "hits“ - areas of high energy intensity - these are then nodes in the graph, connected by edges to their k nearest neighbors. The final result is a graph for each event, describing the event

Each node has then also data on the hit it represents, such as location, how many pixels it covers, what module it is in and intensity sum, thus even in a small shower that leaves only one node, there is still data to learn from.

The Model
s02_train_gnn.py

The GNN utilizes a model called Graph Attention Network (GAT). This model introduces attention to the GNN, enabling it to discriminate between the neighbors of a graph, paying more attention to the ones that prove to be important.

We also utilized Optuna to automate HPO, we gave it a parameter space to search, it then chose promising parameters using a search algorithm, TPEsampler is default, but other options are pre-configured. It has early stopping patience 15, so if loss does not improve enough over 15 epochs it times out. Max epochs was 75. This enabled us to run the training over days on a PC we got to borrow.

Finally after the best parameters is found the model is trained a last time and evaluated on a test set. Then the model weights are written to a .pth file (it is also written out during training at regular intervals in case of crashes or mishaps).

Attempt at an even better model

- Mark this one is also mostly for you

We also tried with no luck to optimize our model further by implementing a terminator and pruner into it.

We tried to parallelize the training and utilize the GPUs in the PC we borrowed by using CUDA, however even when the output log clearly said it recognized the GPU and was using CUDA, it only at most utilized some 25% of the GPU for seconds at a time, averaging at 3-4%.
We suspect it must have been a bottleneck that we could not see, possibly the RAM or physical CPU to GPU transfer speed was limiting it.

Hybrid model

- The next few slides will go over the more technical parts of the Hybrid CNN/GNN model
- Highlighting some of the design principles and core elements of the model
- Following this, it should be clear how the model was built, and possible reflections upon the different elements.

DATA PREPARATION

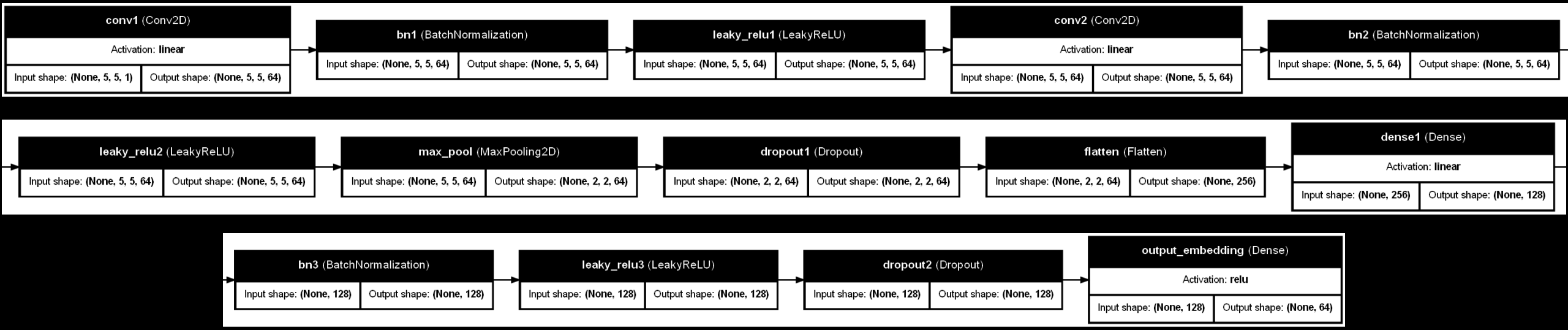
- First the relevant packages are loaded
- Next the data is loaded and concated into common lists, one for electrons and one for hadrons
- After preparing the data, the channel mapping is defined
- This is used for mapping the 1D data into 2D, such that it can be used by a CNN

DATA PREPROCESSING

- Data is then concated into a common list, and labels are added
- The data is then shuffled, as to not have the model learn any ordering (which there very much is at this point)
- The total energy deposition for each observation is calulated, and standardized using StandardScaler
- Lastly data is split into 80 / 20 for training and testing

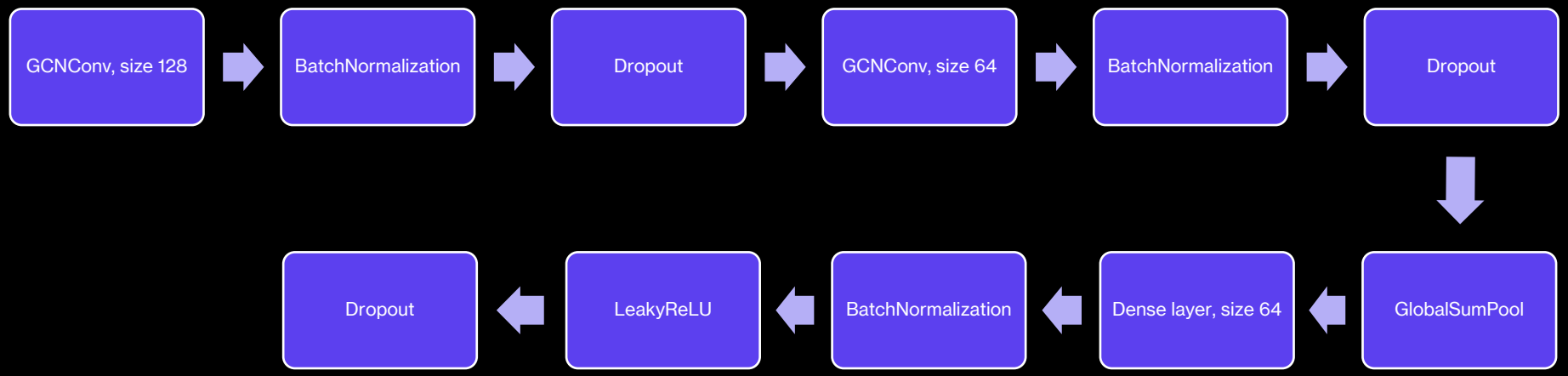
Building the CNN

- A Keras Sequential CNN is used for the sub grid models
- The model in short looks like the graph in the bottom (for a 5x5 grid)
  + For the first drop out a rate of 0.3 was used, and for the second 0.4
- A dictionary was then created to hold each of the CNN models



The Hybrid E2E model

- Next the data is input to each of their corresponding pre instantiated models
- The outputs which are 64 dimensional vectors are then stacked and passed to a 2 layer Spektral GNN along with the adjacency matrix
- The GNN layers are then aggregated into a single graph embedding, which is concated with the standardized total energy, created in the data preprocessing step
- This is then passed to a MLP for predicting the class



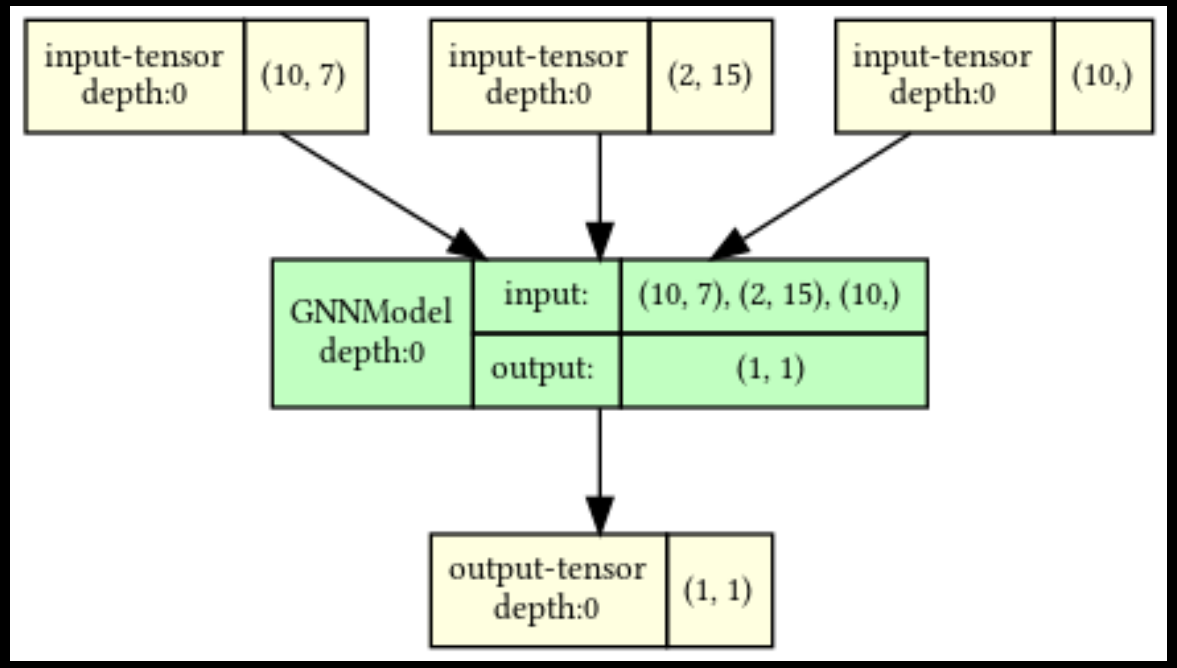
Custom data loading

- As this is a custom built model, we had to make a custom data loader
- The loader first processes the raw data, using the channel mapping, turning them into a Graph object
- The objects are then batched into a tuple, which has tuples of nine subgrid batches, a batch of adjacency matrices and lastly a batch of the total energy features. All this to ensure that it can be used for the models input.

Compiling and training

- We used ”adam” as our optimizer and binary cross entropy as our loss function.
- For training, checkpoints were created, to save the best weights after each epoch.
- Even though it wasn’t used in the project, early stopping was also implemented.

GNN


input-tensor
depth:0
(10, 7)
input-tensor
depth:0
(2, 15)
input-tensor
depth:0
(10,)
GNNModel
depth:0
input:
(10, 7), (2, 15), (10,)
output:
(1, 1)
output-tensor
depth:0
(1, 1)