CNNs and GNNs for Tagging Anomalous Showers with ATLAS

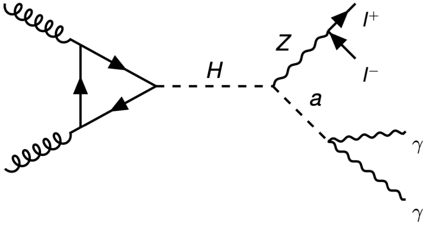
DESY FH-SciComp Workshop 2024

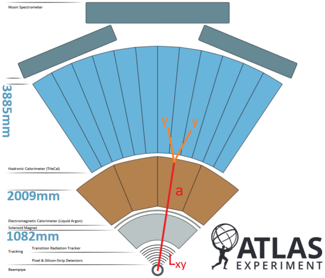
Lukas Bauckhage, Federico Meloni

Physikalisches Institut Universität Bonn, Deutsches Elektronen-Synchrotron (DESY) Hamburg

02.07.2024

Reconstructing long-lived ALP decay photons

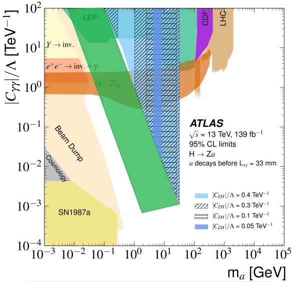




ATLAS-OUTREACH-2021-052

> What if the ALP is **long-lived**?

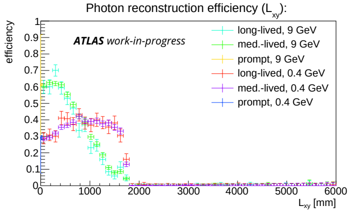
- Important difference:
  Due to long ALP lifetime $\Rightarrow$ large displacement of ALP decay vertex decreases reconstruction efficiency of photons
- New challenge: optimize reconstruction of displaced photons

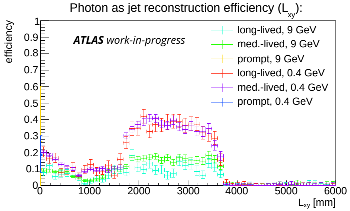


arXiv: 2312.01942

Reconstruction efficiency of displaced photons

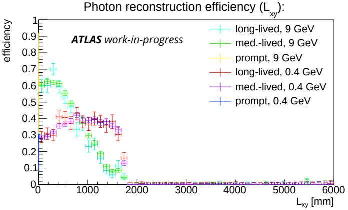
- L_{xy}: transverse distance between IP and ALP decay vertex
- When reaching HCAL, jets supersede photons
- photons/jets are matched to truth ALP decay photons (minimal ΔR)
- One approach: jet objects instead of photons?
- 2 main questions:
  - **How to optimize selection/cutflow?**
  - **How to identify displaced photon candidates?**

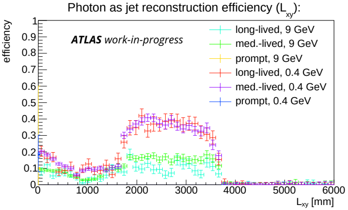




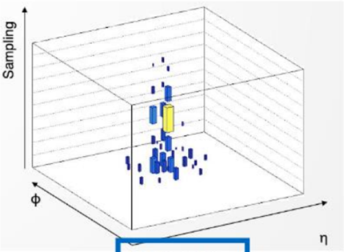
Reconstruction efficiency of displaced photons

- L_{xy}: transverse distance between IP and ALP decay vertex
- When reaching HCAL, jets supersede photons
- photons/jets are matched to truth ALP decay photons (minimal ΔR)
- One approach: jet objects instead of photons?
- 2 main questions:
  - How to optimize selection/cutflow?
  - **How to identify displaced photon candidates?**

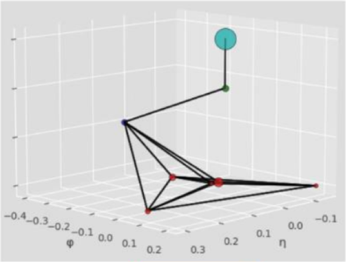




How to identify displaced photon candidates?



Jet image



Jet graph

- CNN jet tagger already used in a dark photon analysis (arXiv: 2206.12181)
- **New**: Transform clusters of calo cells into graphs and use GNN to tag jets
- Jet Images mostly empty $\Rightarrow$ conversion to graphs very convenient (less storage/memory, faster I/O, faster training, ...)
- CNN vs. GNN comparison (as fair as possible)

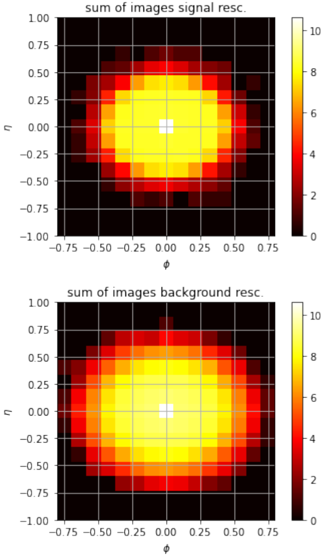
CNN framework

Image Processing

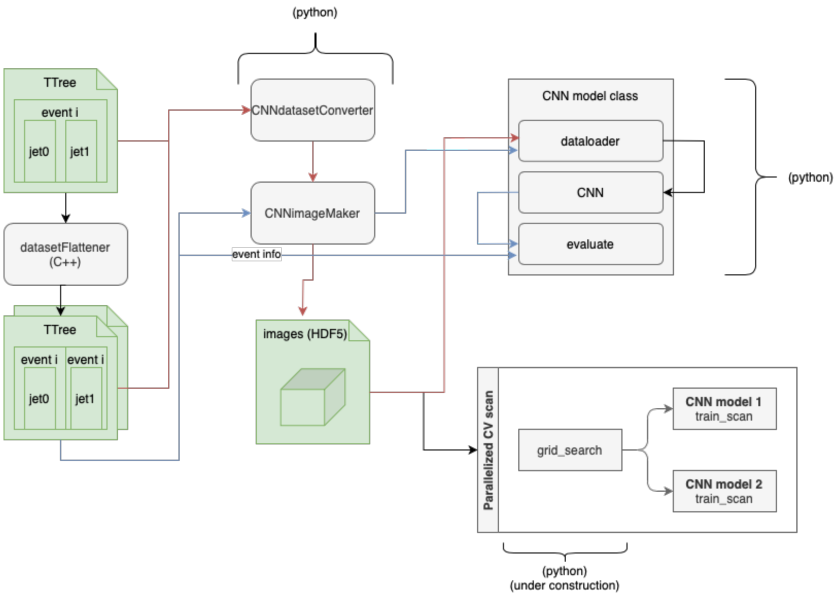
- Find highest energy cluster of jet
- Convert positions of all other clusters to relative coordinates w.r.t. highest energy cluster

- Each cluster energy filled in $\eta \times \phi \times$ layer $= 15 \times 15 \times (4/5/3)$ histograms
- 3 different histograms for barrel, endcap and barrel extension (different # of layers)

- Clusters below threshold $E_{\text{min}} = 400\,\mathrm{MeV}$ are not used
- Histograms are rescaled by total energy of all clusters (i.e. normalized to 1)

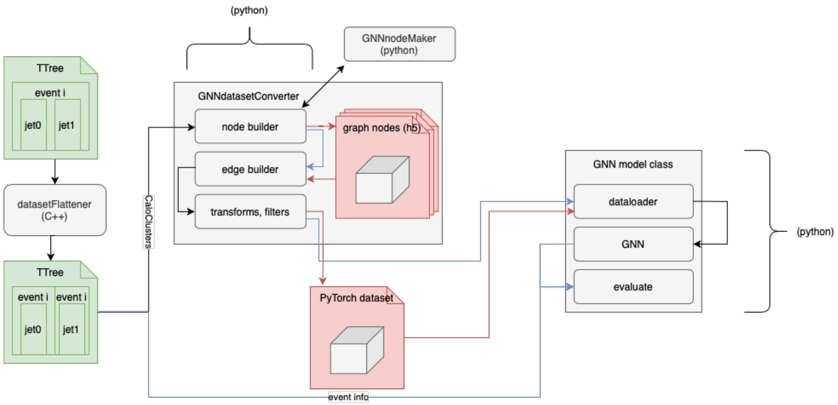


CNN framework



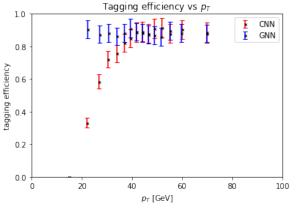
GNN framework

GNN framework



Comparison

Comparison of Performance



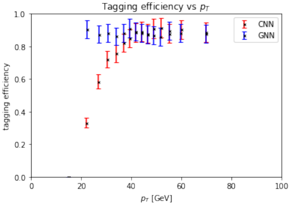
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	[(None, 15, 15, 12, 1)]	0
conv3d_0 (Conv3D)	(None, 15, 15, 12, 60)	1680
conv3d_1 (Conv3D)	(None, 15, 15, 12, 60)	97260
max_pool_0 (MaxPooling3D)	(None, 7, 7, 12, 60)	0
conv3d_2 (Conv3D)	(None, 7, 7, 12, 60)	97260
max_pool_1 (MaxPooling3D)	(None, 2, 2, 12, 60)	0
flatten (Flatten)	(None, 2880)	0
dense_0 (Dense)	(None, 100)	288100
dense_1 (Dense)	(None, 1)	101

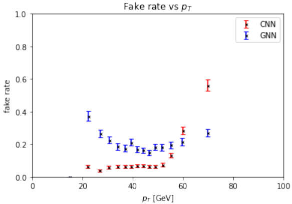
Total params: 484401 (1.85 MB)
Trainable params: 484401 (1.85 MB)
Non-trainable params: 0 (0.00 Byte)

	Name	Type	Params
0	loss	BCEWithLogitsLoss	0
1	norm	BatchNorm	8
2	conv1	ARMAConv	203 K
3	conv2	ARMAConv	590 K
4	conv3	ARMAConv	590 K
5	lin0	Linear	65.8 K
6	lin	Linear	257

1.5 M Trainable params
0 Non-trainable params
1.5 M Total params
5.803 Total estimated model params size (MB)

Comparison of Performance





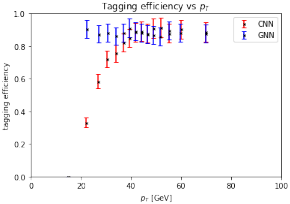
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	[(None, 15, 15, 12, 1)]	0
conv3d_0 (Conv3D)	(None, 15, 15, 12, 60)	1680
conv3d_1 (Conv3D)	(None, 15, 15, 12, 60)	97260
max_pool_0 (MaxPooling3D)	(None, 7, 7, 12, 60)	0
conv3d_2 (Conv3D)	(None, 7, 7, 12, 60)	97260
max_pool_1 (MaxPooling3D)	(None, 2, 2, 12, 60)	0
flatten (Flatten)	(None, 2880)	0
dense_0 (Dense)	(None, 100)	288100
dense_1 (Dense)	(None, 1)	101

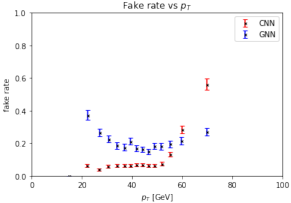
Total params: 484401 (1.85 MB)
Trainable params: 484401 (1.85 MB)
Non-trainable params: 0 (0.00 Byte)

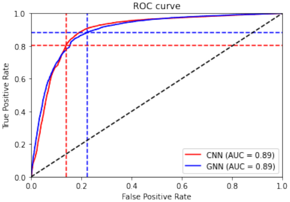
	Name	Type	Params
0	loss	BCEWithLogitsLoss	0
1	norm	BatchNorm	8
2	conv1	ARMAConv	203 K
3	conv2	ARMAConv	590 K
4	conv3	ARMAConv	590 K
5	lin0	Linear	65.8 K
6	lin	Linear	257

1.5 M Trainable params
0 Non-trainable params
1.5 M Total params
5.803 Total estimated model params size (MB)

Comparison of Performance







Layer (type)	Output Shape	Param #
input_layer (InputLayer)	[(None, 15, 15, 12, 1)]	0
conv3d_0 (Conv3D)	(None, 15, 15, 12, 60)	1680
conv3d_1 (Conv3D)	(None, 15, 15, 12, 60)	97260
max_pool_0 (MaxPooling3D)	(None, 7, 7, 12, 60)	0
conv3d_2 (Conv3D)	(None, 7, 7, 12, 60)	97260
max_pool_1 (MaxPooling3D)	(None, 2, 2, 12, 60)	0
flatten (Flatten)	(None, 2880)	0
dense_0 (Dense)	(None, 100)	288100
dense_1 (Dense)	(None, 1)	101

Total params: 484401 (1.85 MB)
Trainable params: 484401 (1.85 MB)
Non-trainable params: 0 (0.00 Byte)

	Name	Type	Params
0	loss	BCEWithLogitsLoss	0
1	norm	BatchNorm	8
2	conv1	ARMAConv	203 K
3	conv2	ARMAConv	590 K
4	conv3	ARMAConv	590 K
5	lin0	Linear	65.8 K
6	lin	Linear	257

1.5 M Trainable params
0 Non-trainable params
1.5 M Total params
5.803 Total estimated model params size (MB)

Conclusion

- Some (computational) challenges to overcome
- CNN and GNN frameworks set up and ready for optimizing and comparing models

Conclusion

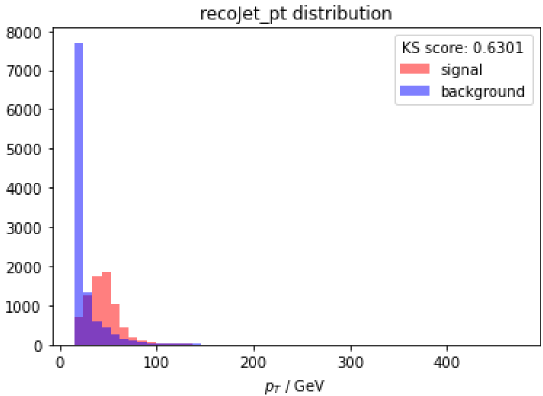
- Some (computational) challenges to overcome
- CNN and GNN frameworks set up and ready for optimizing and comparing models

"Feature Request":
Run LCG `Jupyter`/`IPython` kernels on NAF JupyterHub Server

- It *should* be possible by specifying the correct kernel in the `Jupyter` configuration file
- Need to set all necessary environment variables to `CVMFS` paths
- If someone has done this $\Rightarrow$ please let me know

Backup

p_T distribution



Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4+5+3))$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB}))$
- recreate images on-the-fly at every epoch not the best solution either

Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4+5+3))$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB}))$
- recreate images on-the-fly at every epoch not the best solution either

$\Rightarrow$ preproduce images and store to disk

Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4 + 5 + 3)$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB}))$
- recreate images on-the-fly at every epoch not the best solution either

$\Rightarrow$ preproduce images and store to disk

- A complete dataset still too large to load into memory at once

Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4 + 5 + 3))$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB})$)
- recreate images on-the-fly at every epoch not the best solution either

$\Rightarrow$ preproduce images and store to disk

- A complete dataset still too large to load into memory at once

$\Rightarrow$ load images in (variable-size) **batches** (`tf.data.Dataset` and `tf.data.Dataset.from_generator`)

Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4 + 5 + 3)$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB}))$
- recreate images on-the-fly at every epoch not the best solution either

$\Rightarrow$ preproduce images and store to disk

- A complete dataset still too large to load into memory at once

$\Rightarrow$ load images in (variable-size) **batches** (`tf.data.Dataset` and `tf.data.Dataset.from_generator`)

- How to maintain the linking between images and the original ROOT event data? (e.g. to investigate performance, efficiencies, etc. in dep. of kinematic variables)
- Each event can have multiple jets
- Some events/jets filtered out in pre-selection
- Definition of Signal/Background might depend on complicated criteria (e.g. truthmatching)

Training Procedure

- datasets of $\mathcal{O}(10^5) - \mathcal{O}(10^6)$ **3D** images won't fit into memory
  $(15 \times 15 \times (4+5+3))$ pixel images $\approx 173\,\mathrm{kB}$
  $\Rightarrow$ batch of 5000 images $\approx 864\,\mathrm{MB}$
  $\Rightarrow$ whole dataset $\approx \mathcal{O}(100\,\mathrm{GB})$)
- recreate images on-the-fly at every epoch not the best solution either

$\Rightarrow$ preproduce images and store to disk

- A complete dataset still too large to load into memory at once

$\Rightarrow$ load images in (variable-size) **batches** (`tf.data.Dataset` and `tf.data.Dataset.from_generator`)

- How to maintain the linking between images and the original ROOT event data? (e.g. to investigate performance, efficiencies, etc. in dep. of kinematic variables)
- Each event can have multiple jets
- Some events/jets filtered out in pre-selection
- Definition of Signal/Background might depend on complicated criteria (e.g. truthmatching)

$\Rightarrow$ For performance tests create images on-the-fly from ROOT data and read event data along with CaloCluster data
"flatten" ROOT tree (i.e. 1 n-jet-event $\rightarrow$ n 1-jet-events)

Graph Processing

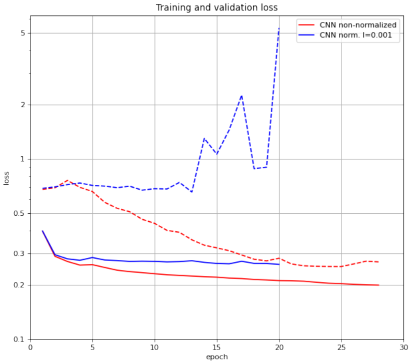
- Create nodes from CaloClusters with features η, ϕ, E_{abs}, E_{norm} (for each layer)
  η, ϕ relative to highest energy cluster
  store to disk (`.h5`)

- Create `PyTorch` dataset (`GNNdataset` class inherits from `torch_geometric.data.InMemoryDataset`):
  - Load nodes from `.h5` files and combine them
  - Build graphs from list of nodes, filter out nodes below threshold $E_{\text{min}} = 400\,\text{MeV}$ and remove E_{abs} as a feature

- Apply filters ($\rightarrow$ `jetgraphs` library)
  - pre-filters (e.g. $n_{\text{nodes}} \geq 2$)
  - pre-transform: build edges according with tunable thresholds (threshold for distance between nodes in same layer, consecutive layer, self-loop weights, ...)
  - transform: add layer information
  - post-filters

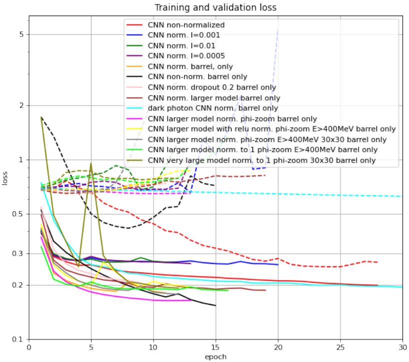
- Store dataset to disk (`.pt`)
- No manual batch-wise training routine necessary
- (Possibility to create graphs on-the-fly from ROOT data for tests of dependence of performance, efficiencies, etc. on kinematics)

CNN normalization



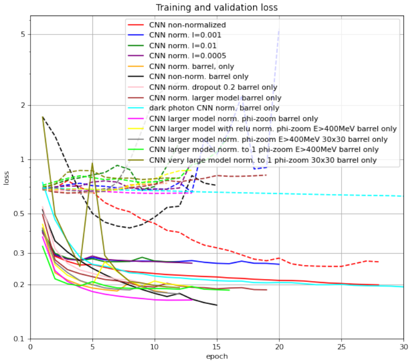
- After normalization of images was introduced, CNN was not able to improve on the validation or test set

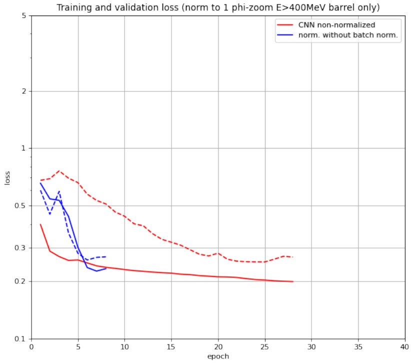
CNN normalization



- After normalization of images was introduced, CNN was not able to improve on the validation or test set
- After many tests, it was found that batch normalization layers were the cause of the problem:

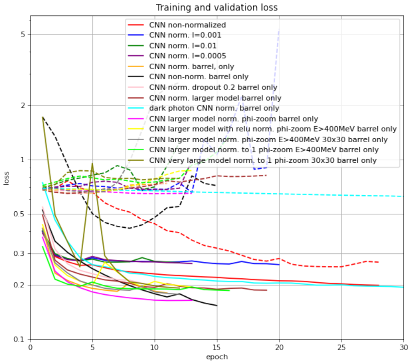
CNN normalization

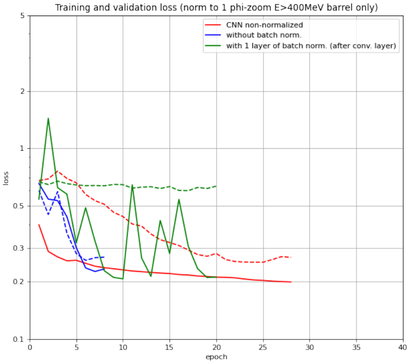




- After normalization of images was introduced, CNN was not able to improve on the validation or test set
- After many tests, it was found that batch normalization layers were the cause of the problem:

CNN normalization





- After normalization of images was introduced, CNN was not able to improve on the validation or test set
- After many tests, it was found that batch normalization layers were the cause of the problem:
- As soon as a single batch norm. layer is introduced in model $\Rightarrow$ CNN does not improve on validation set anymore
- We were puzzled by this behavior, batch normalization should do the opposite