Tau Decay Mode Classification using Neural Network

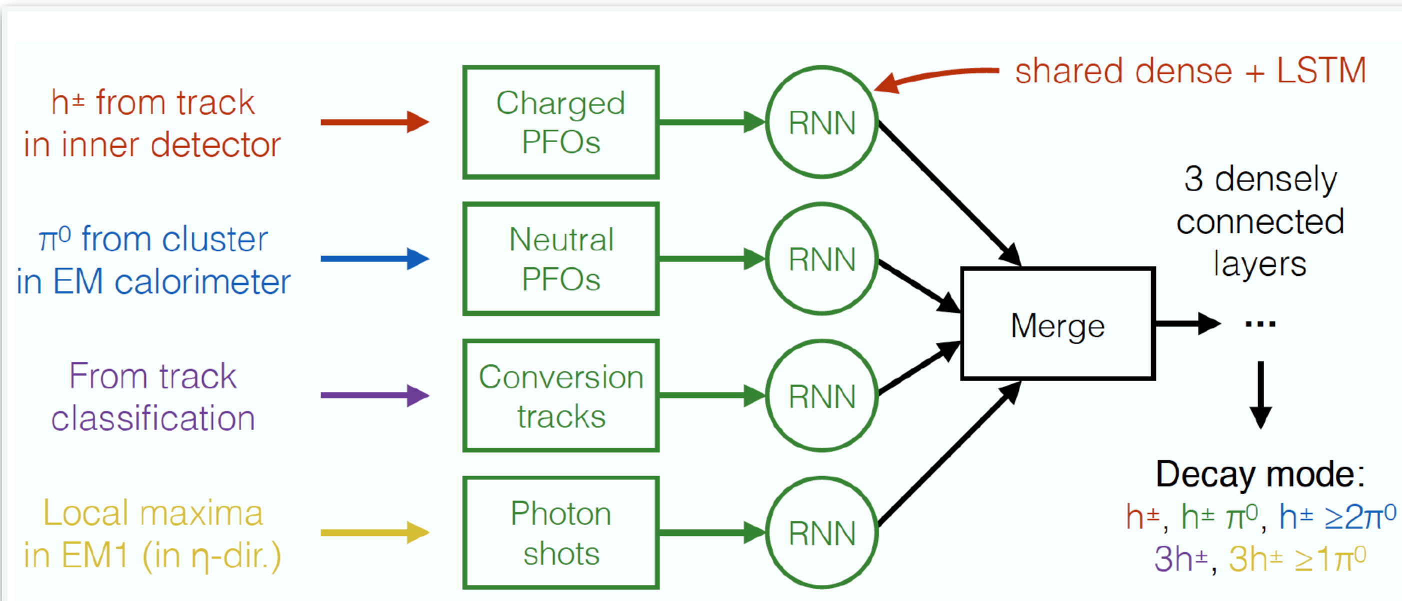
Bowen Zhang

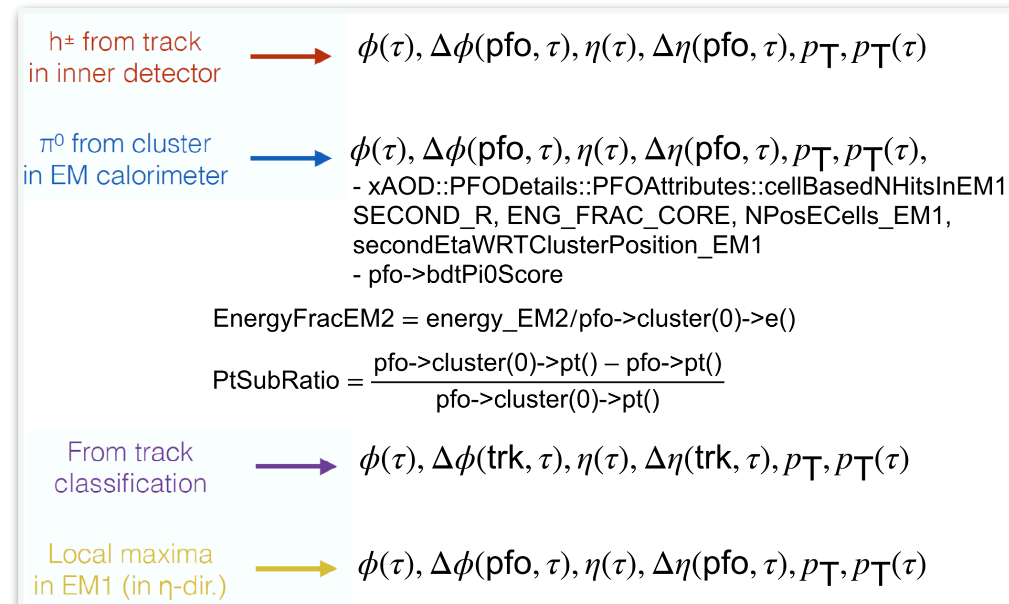
09/12/2019

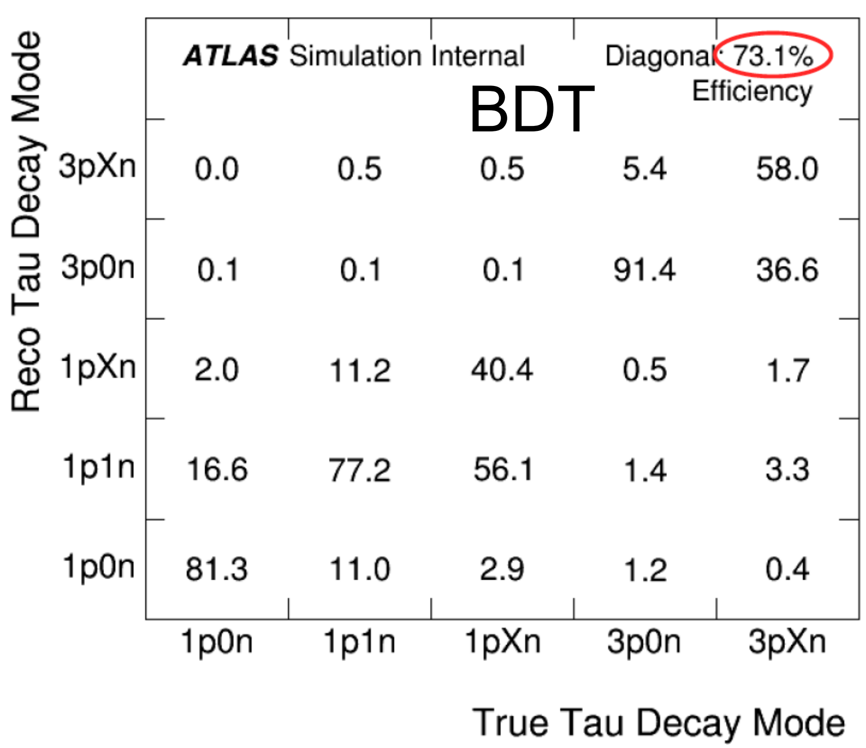
NJU-TAU Meeting

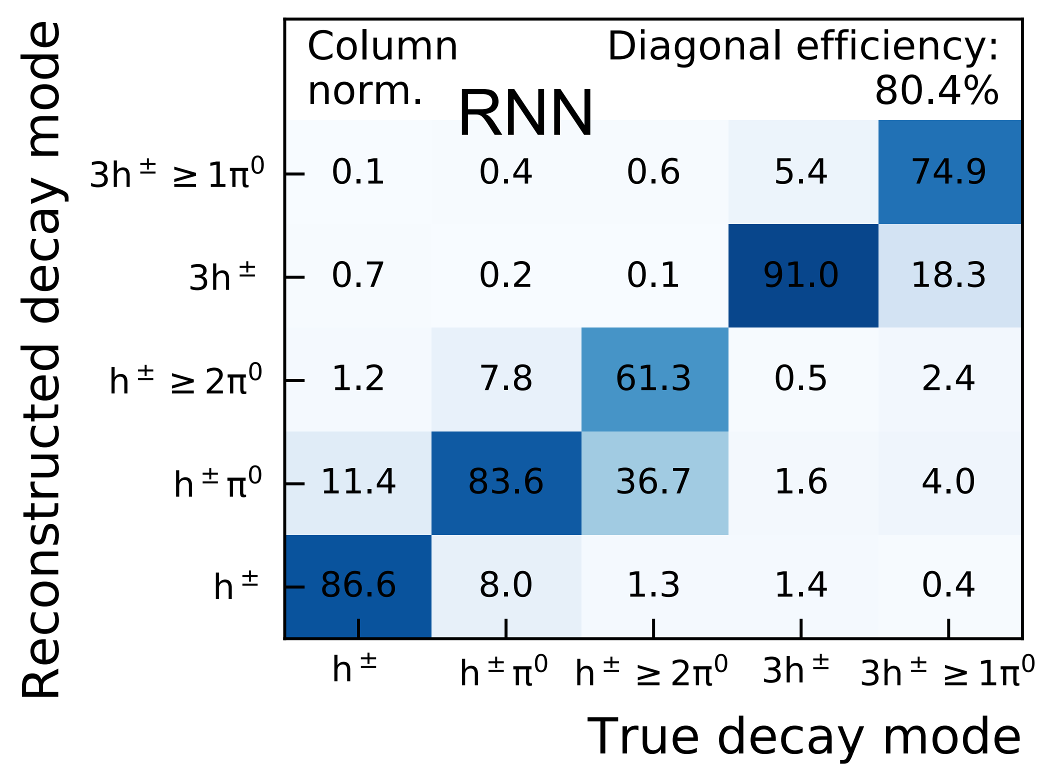
Recap

- Presented RNN approach a while ago



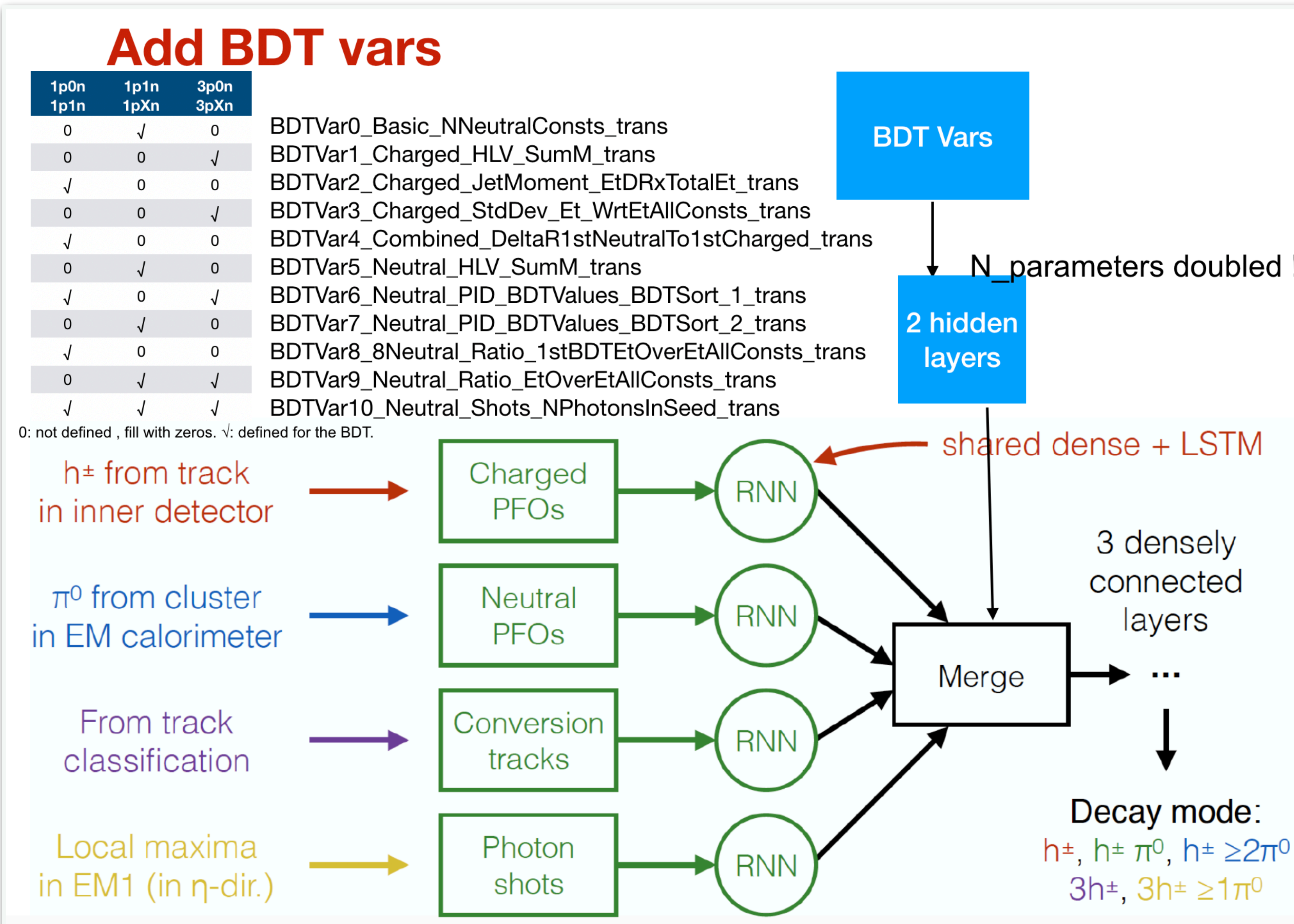






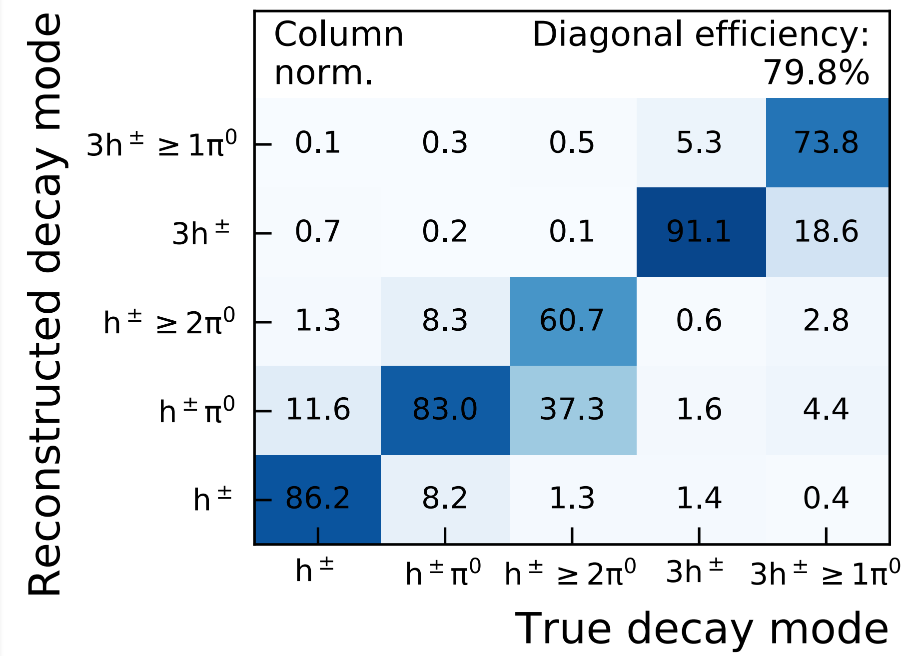
Recap

- Checked inslusion of PanTau BDT vars



1p0n 1p1n	1p1n 1pXn	3p0n 3pXn	
0	√	0	BDTVar0_Basic_NNeutralConsts_trans
0	0	√	BDTVar1_Charged_HLV_SumM_trans
√	0	0	BDTVar2_Charged_JetMoment_EtDRxTotalEt_trans
0	0	√	BDTVar3_Charged_StdDev_Et_WrtEtAllConsts_trans
√	0	0	BDTVar4_Combined_DeltaR1stNeutralTo1stCharged_trans
0	√	0	BDTVar5_Neutral_HLV_SumM_trans
√	0	√	BDTVar6_Neutral_PID_BDTValues_BDTSort_1_trans
0	√	0	BDTVar7_Neutral_PID_BDTValues_BDTSort_2_trans
√	0	0	BDTVar8_8Neutral_Ratio_1stBDTEtOverEtAllConsts_trans
0	√	√	BDTVar9_Neutral_Ratio_EtOverEtAllConsts_trans
√	√	√	BDTVar10_Neutral_Shots_NPhotonsInSeed_trans

0: not defined , fill with zeros. √: defined for the BDT.



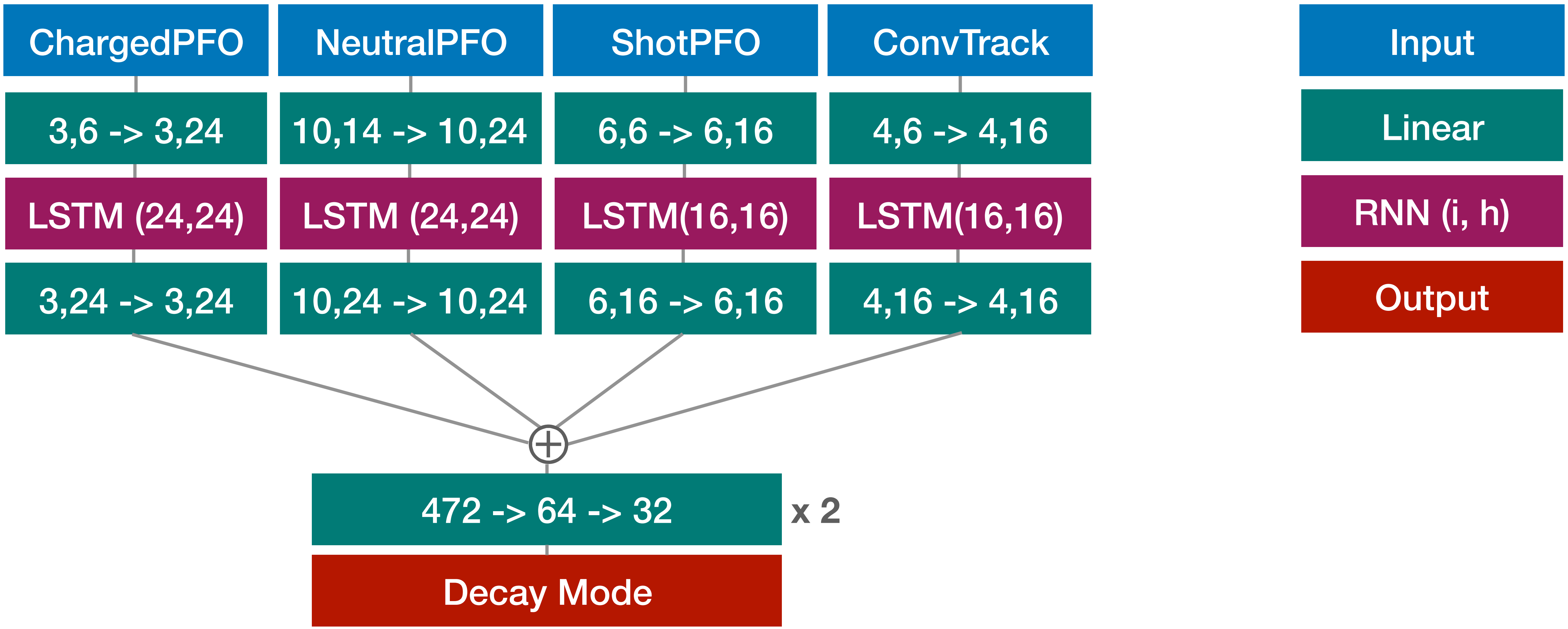
To be improved…

- Input variable:
    - Redundant variables?
    - New powerful discriminant variables?
    - Properly transformed?
    - The way to feed them into the NN?
- Neural Network:
    - Alternative architectures?
- Tool:
    - Better workflow? Memory problem? Paralell?
- Others:
    - Training / testing datasets
    - …

More-or-less involved today

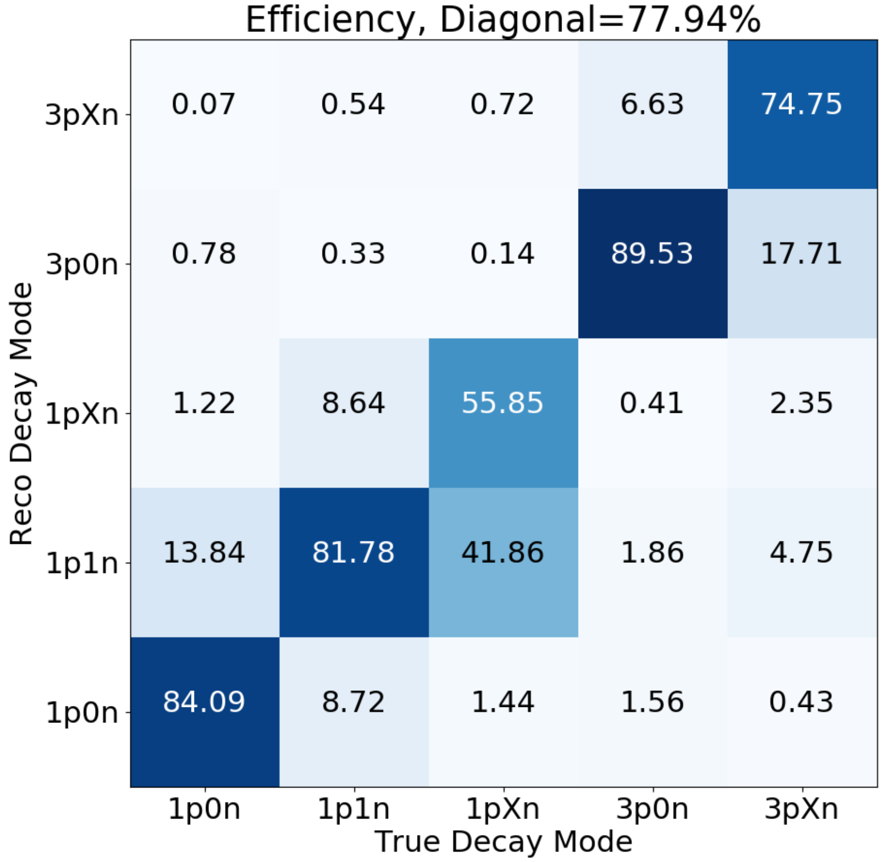
- Input variable:
  - **Redundant variables?**
  - New powerful discriminant variables?
  - **Properly transformed?**
  - **The way to feed them into the NN?**
- **Neural Network:**
  - **Alternative architectures?**
- **Tool:**
  - **Better workflow? Memory problem? Paralell?**
- Others:
  - Training / testing datasets
  - …

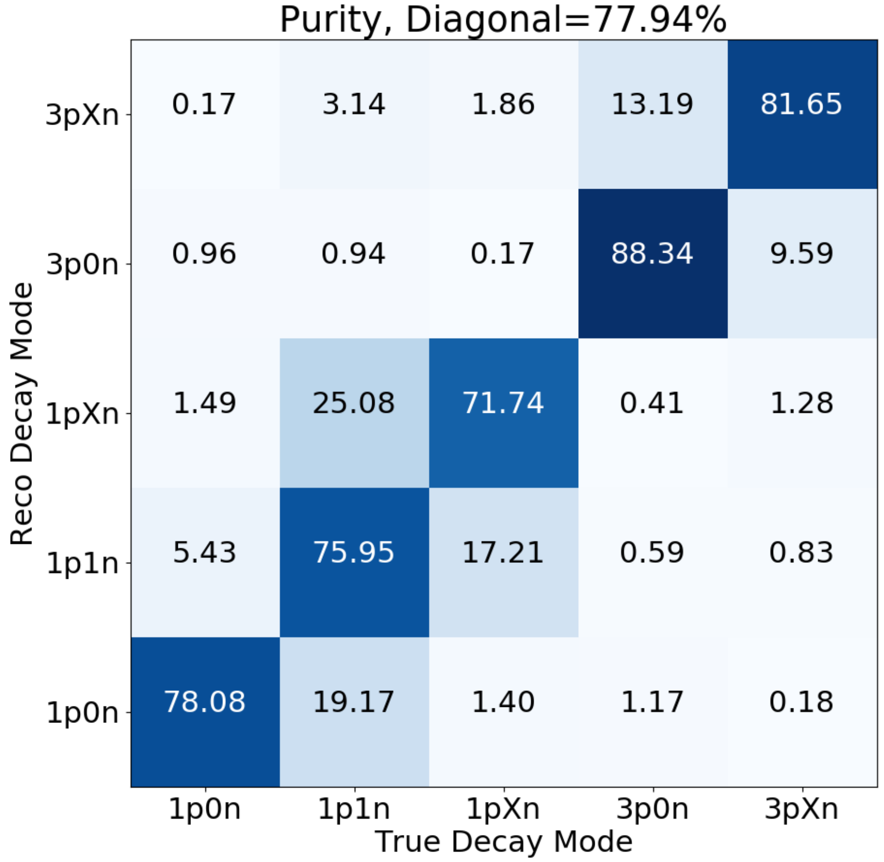
Baseline - the "multi-RNN"



- In the following:
  - the inputs are the same: from Charged PFOs, Neutral PFOs, Shot PFOs and Conversion tracks
  - Training: 4M (1/4 mc16d gammatautau), Validation: 1M, Testing: 0.1M

Baseline - the "multi-RNN"

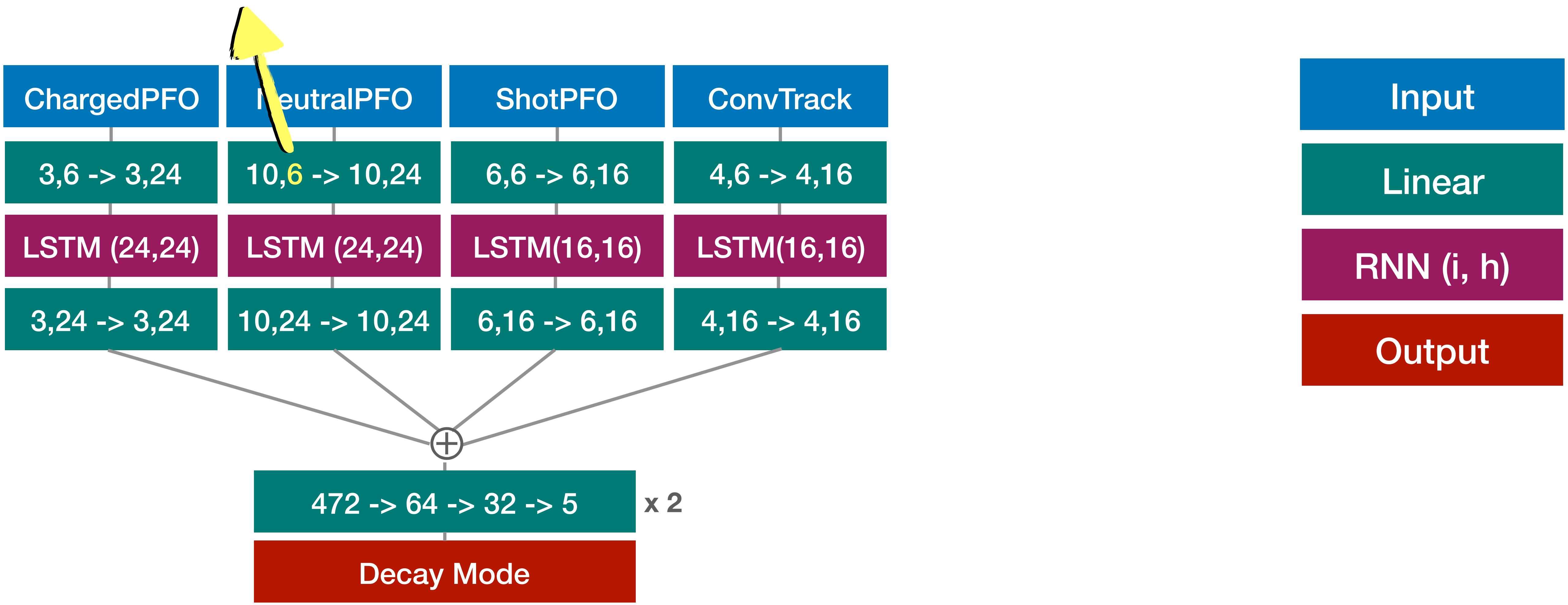




4-vector of individual objects (wrt. Tau jet)

- **Try1:** Only use p4 in RNN: $\phi(\tau), \Delta\phi(\text{pfo}, \tau), \eta(\tau), \Delta\eta(\text{pfo}, \tau), p_{\text{T}}, p_{\text{T}}(\tau)$

-> Drop the Neutral PFO cluster variables
(that defined PanTau BDT Variables)

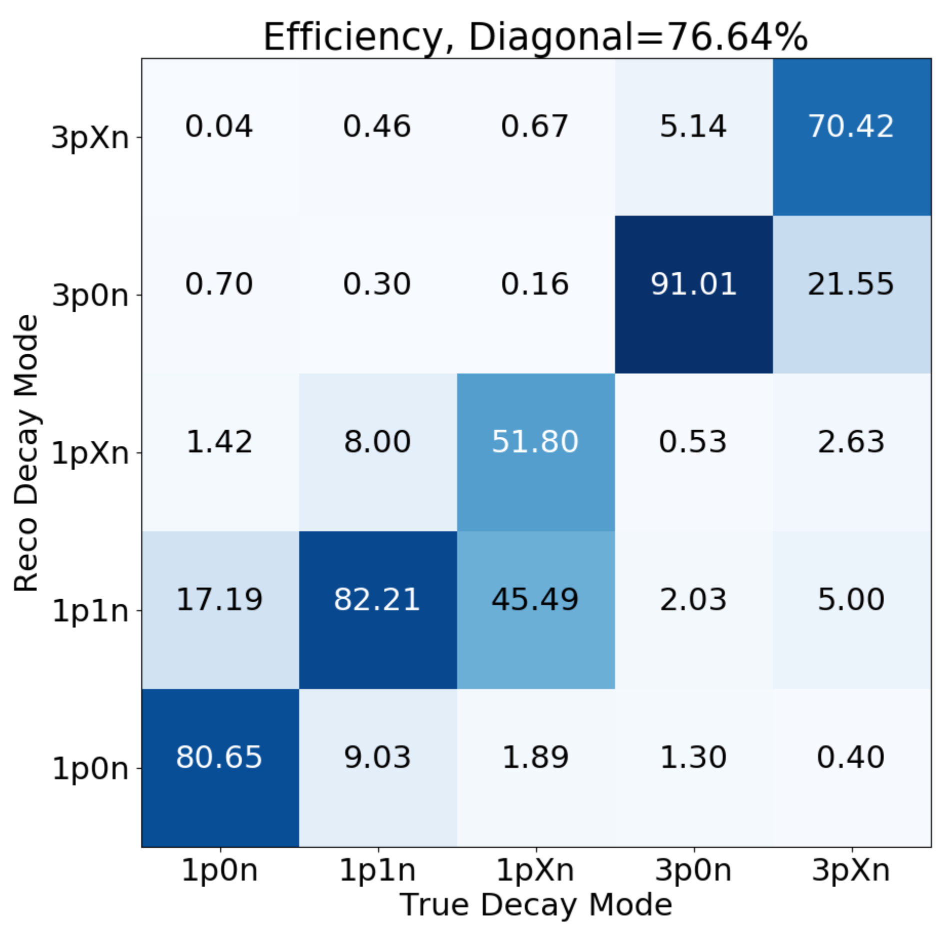


My assumption: With those infomation only, one would NOT get a good performance.

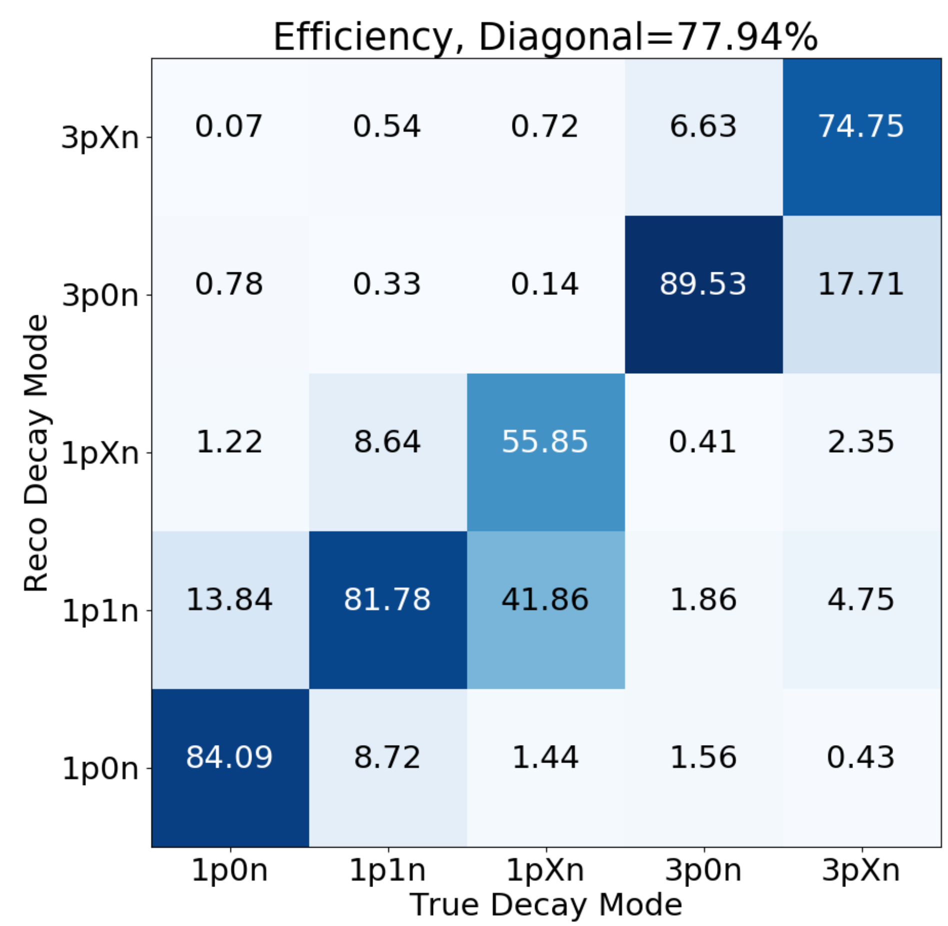
4-vector of individual objects (wrt. Tau jet)

- **Try1:** Only use p4 in RNN
- Efficiency matrix

4-vector

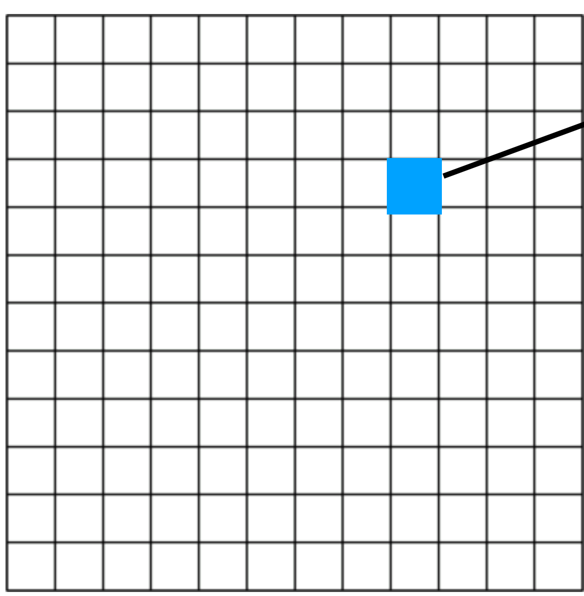


Baseline



4-vector of individual objects (wrt. Tau jet)

- **To prove this in an intuitive way**
- **Try2:**
  - Put the 4-vector infomation into an “image”
  - Train a CNN classifier.
- **Put the 4-vector infomation into an 12x12 “image” (four layers)**



X: $f(\Delta\phi)$

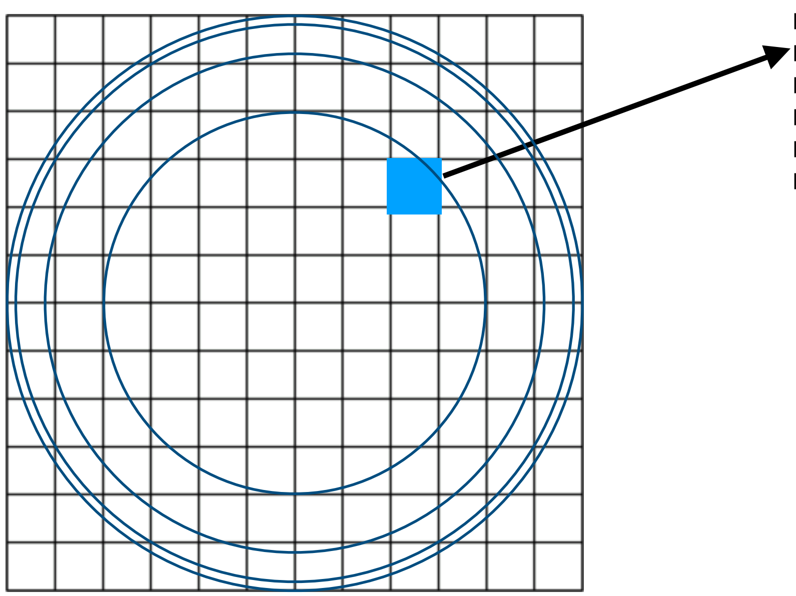
Y: $f(\Delta\eta)$

Z: $log_{10}(p_T)/log_{10}(p_T^{\tau})$

$$f(x) = \pm\sqrt{0.4^2 - (0.4 - x)^2}$$

4-vector of individual objects (wrt. Tau jet)

- **To prove this in an intuitive way**
- **Try2:**
  - Put the 4-vector infomation into an "image"
  - Train a CNN classifier.
- **Put the 4-vector infomation into an 12x12 "image" (four layers)**

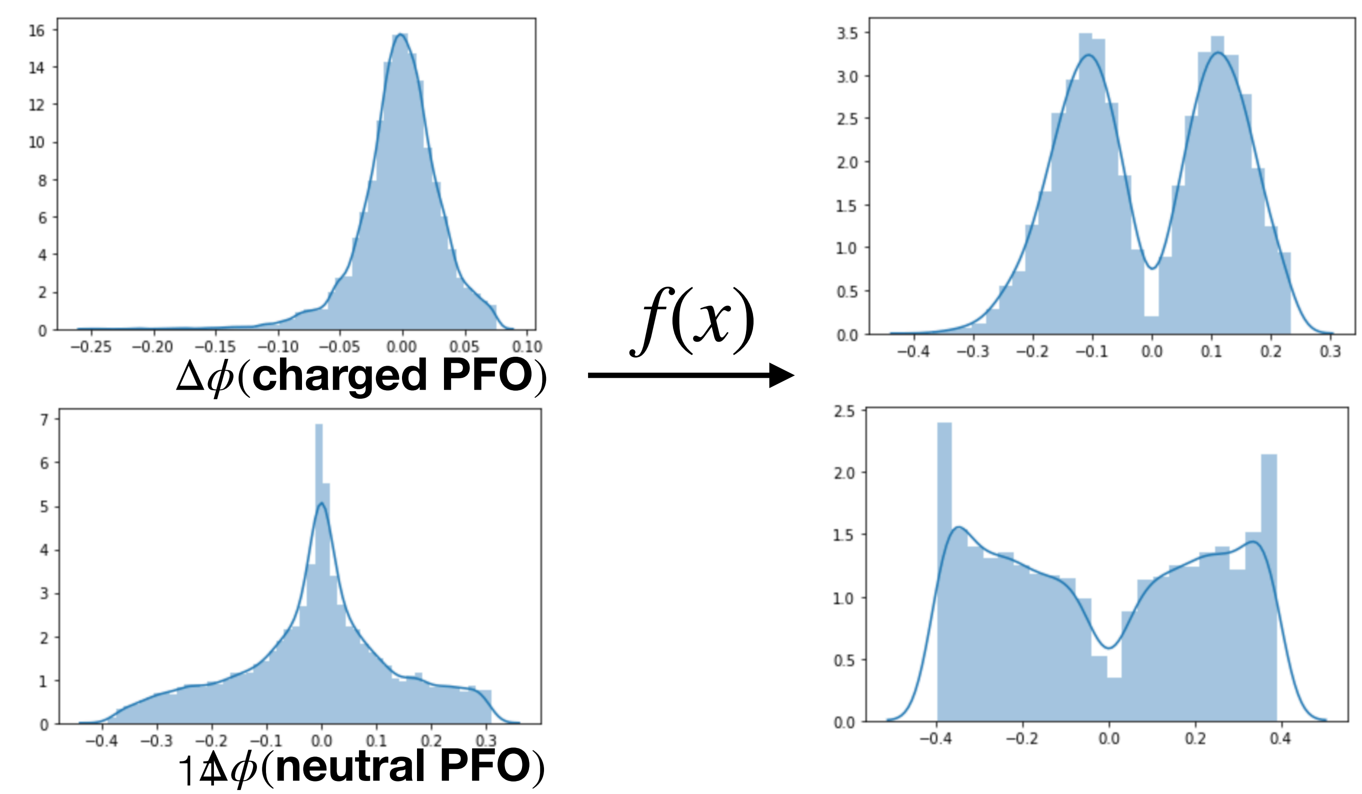


X: $f(\Delta\phi)$

Y: $f(\Delta\eta)$

Z: $log_{10}(p_T)/log_{10}(p_T^{\tau})$

$$f(x) = \pm\sqrt{0.4^2 - (0.4 - x)^2}$$



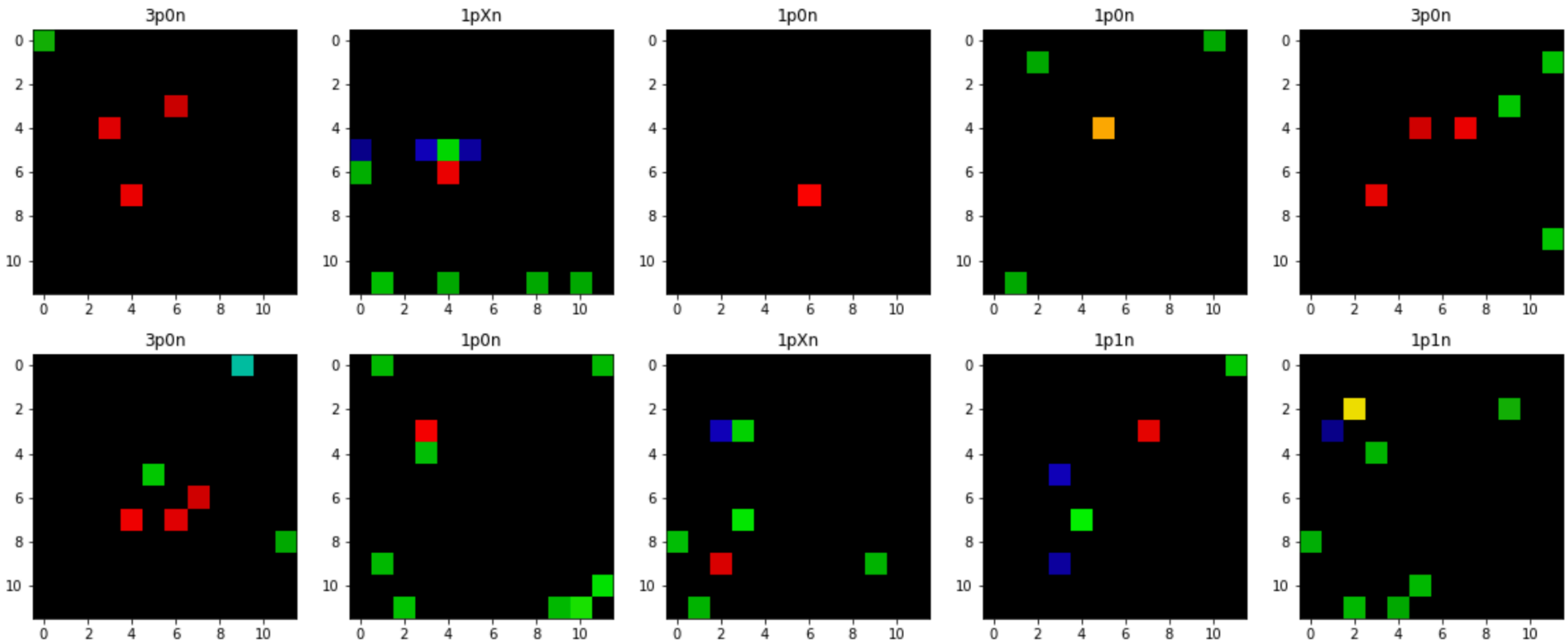
Inside to outside circles:

$\Delta R = 0.1,\ 0.2,\ 0.3,\ 0.4$

4-vector of individual objects (wrt. Tau jet)

- **To visualise:**

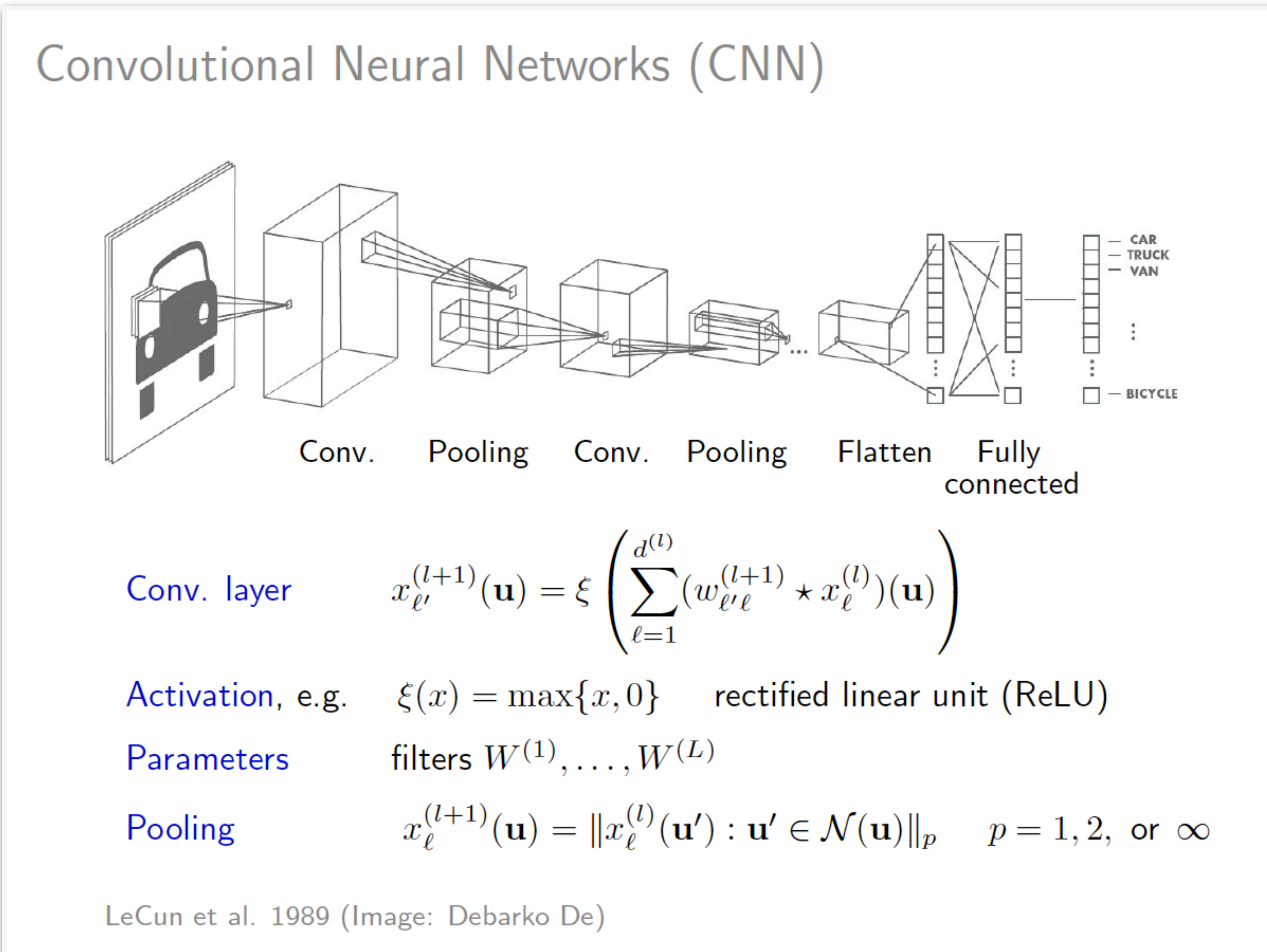
```
visualise_decaymode_cnn(lmdb_dir, shape, dtype)
```

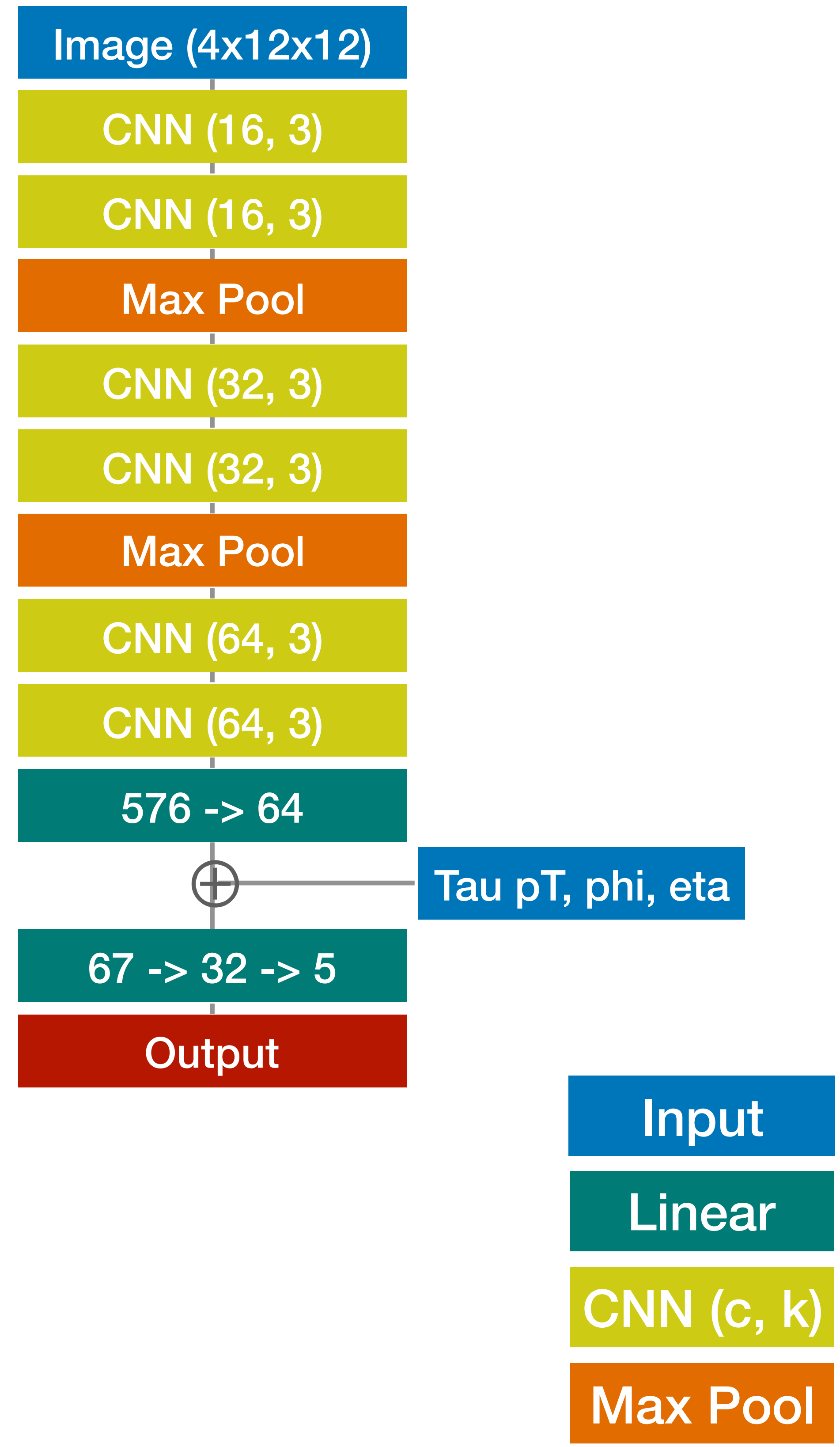


ChargedPFO NeutralPFO ShotPFO

4-vector of individual objects (wrt. Tau jet)

- **Train a simple deep CNN classifier.**

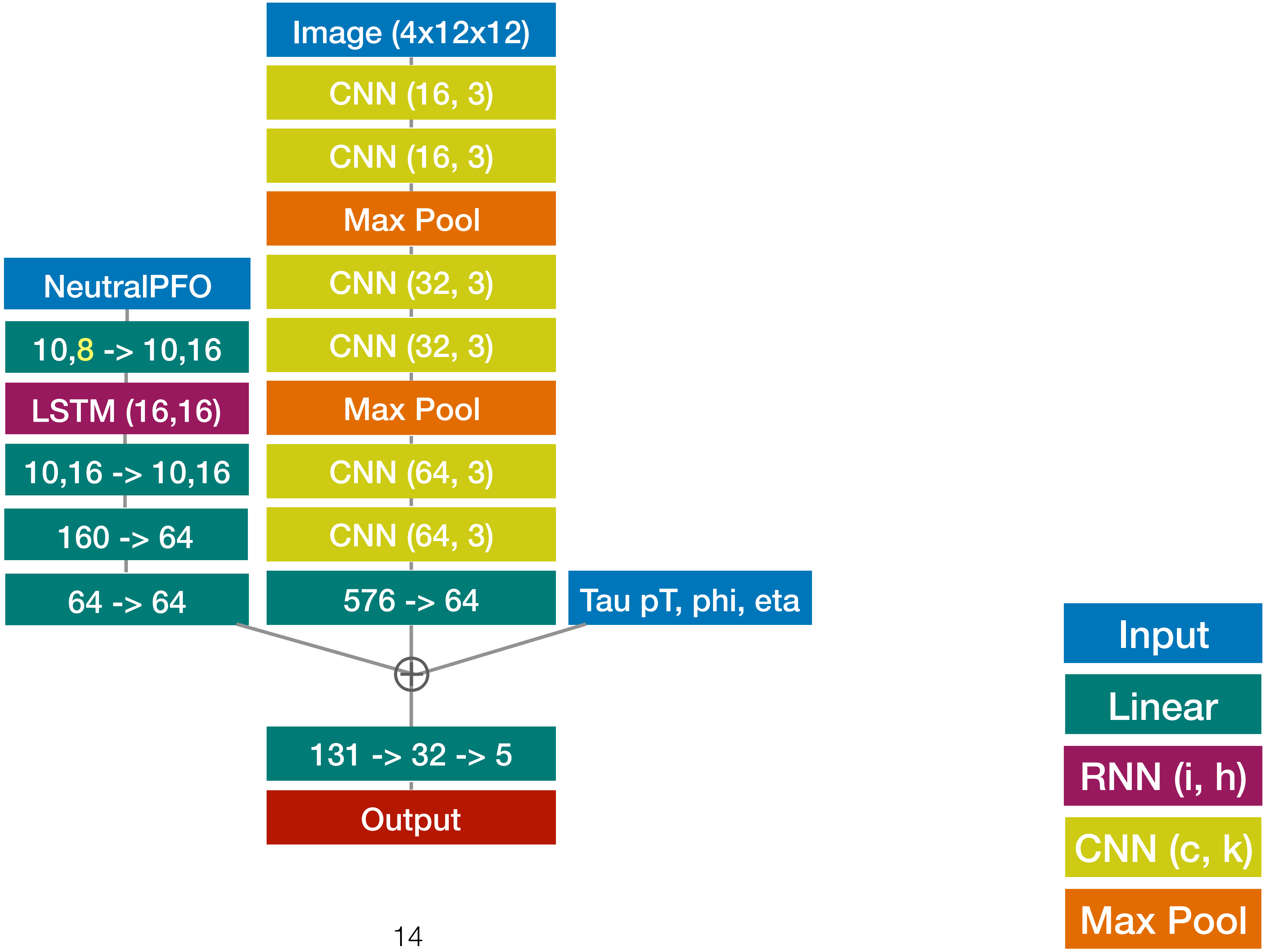




- **Result …**
  - Unfortunately the model was not found…
  - The validation diagonal efficiency was 76%

Try3: CNN + RNN

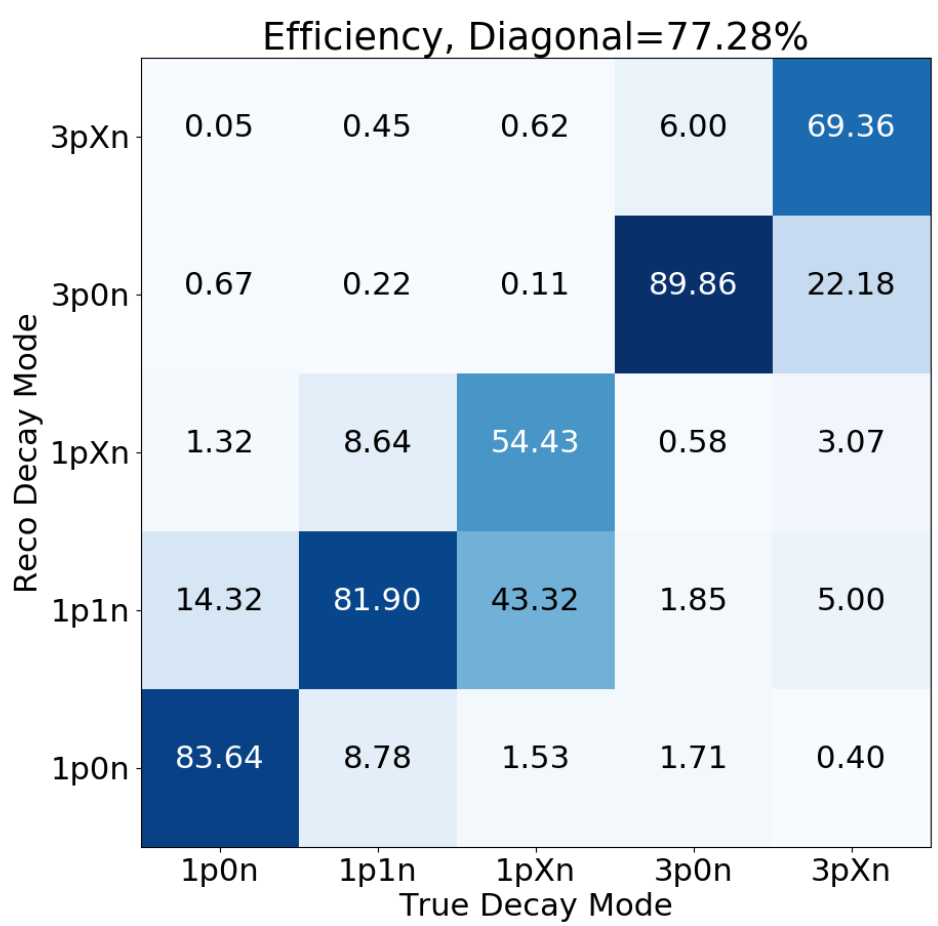
- **Now we can simplify the “multi-RNN” and combine it with the CNN**



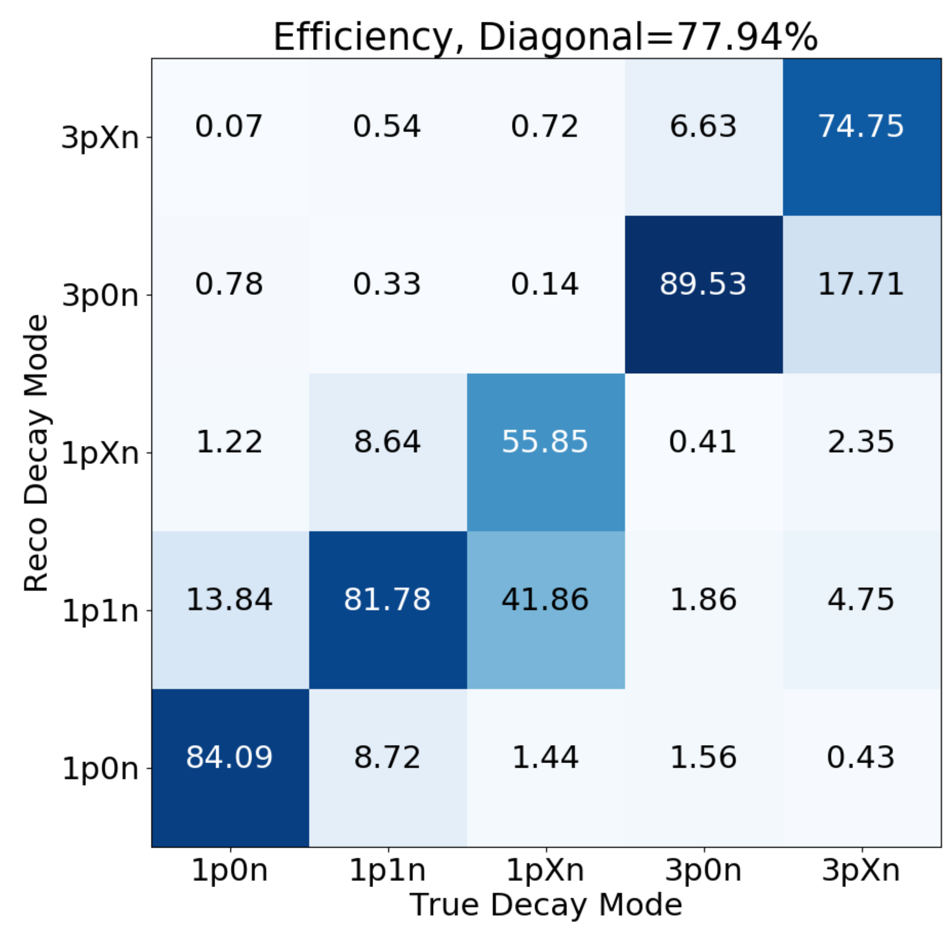
Try3: CNN + RNN

- **Efficiency matrix**

CNN+RNN



Baseline



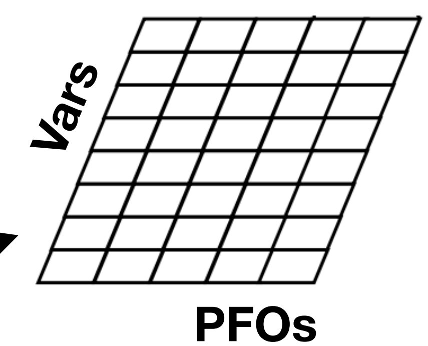
Package Status

ROOT I/O in pure Python and Numpy. (Replace root_numpy, directly use MxAODs)	Efficient loading of large dataset (working on this…)
Better control of computing graph (RNN hidden state, easier to combine multi algs)	Visualizing data, model and training

- **Core: Uproot + Lmdb + PyTorch**
  - Still converting MxAOD into flat ntuples now.
  - Almost no memory consumption using Lmdb
  - Pytorch is very flexible to use for testing and debugging the computing graghs
- **Basic code structure is in good shape**
- **ToDo:**
  - Visualising data and testing. (could be Jupyter-notebook-based)
  - Performance plots. (efficiency, ROC, …)
  - Deploy models in C++ framework

Summary

- **Investigate the variables that are curently being used**
  - With 4-vector information only, the classifier outperform BDT
  - As a next step, I'd like to understand the neutral PFO variables
- **Test alternative NN architectures**
  - Feed 4-vector information into images
    - CNN: compatible performance with multi-RNN
    - Image creation and CNN architecture can be improved
    - No limit on the number of objects (N), could be interesting if N is large -> larger image
  - CNN + RNN: compatible performance with multi-RNN
- **Interesting to try:**
  - Embedding the neutral branch in an "image": fully-CNN



  - Attention-based RNN: learn better the relationship between ojects and between branches
  - Define a customised loss function. (i.e. weighted CrossEntropyLoss)

Backup

Selection

```
df[(df["TauJets.truthDecayMode"] > 4) | (df["TauJets.IsTruthMatched"] != 1) |
   (df["TauJets.pt"] < 20000) | (df["TauJets.truthPtVis"] < 20000) |
   (df["TauJets.pt"] > 100000) | (df["TauJets.truthPtVis"] > 100000) |
   (df["TauJets.eta"] > 2.5) | (df["TauJets.truthEtaVis"] > 2.5) |
   ((df["TauJets.eta"] > 1.37) & (df["TauJets.eta"] < 1.52)) |
   ((df["TauJets.truthEtaVis"] > 1.37) & (df["TauJets.truthEtaVis"] < 1.52)) |
   ((df["TauJets.nTracks"] != 1) & (df["TauJets.nTracks"] != 3)) |
   ((df[“TauJets.truthProng"] != 1) & (df["TauJets.truthProng"] != 3))]
```