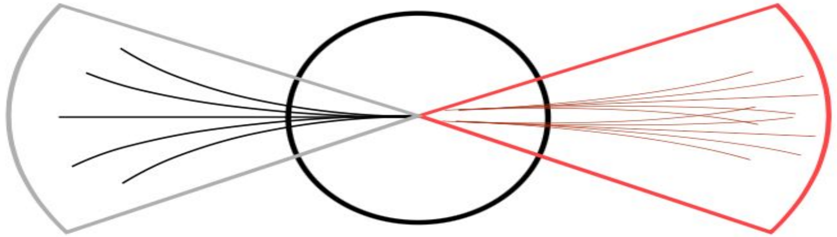
Applying Graph Neural Networks for Xbb tagging

Vishnu Sangli

Introduction

- Attempting to use Neural Networks for identifying boosted hbb jets; Xbb tagger performance used as reference
- Trying to identify h->bb decay jets where both b quarks are often merged into one jet
- Jet Types: [hbb (signal), QCD (bb), QCD (other)]



Architectures Used

- Feed-Forward Neural Networks (ANNs)
- Convolutional Neural Networks (CNNs)
- Graph Neural Networks (GNNs)

We will focus on GNNs, since they were most successful, some of them matching and beating Xbb performance.

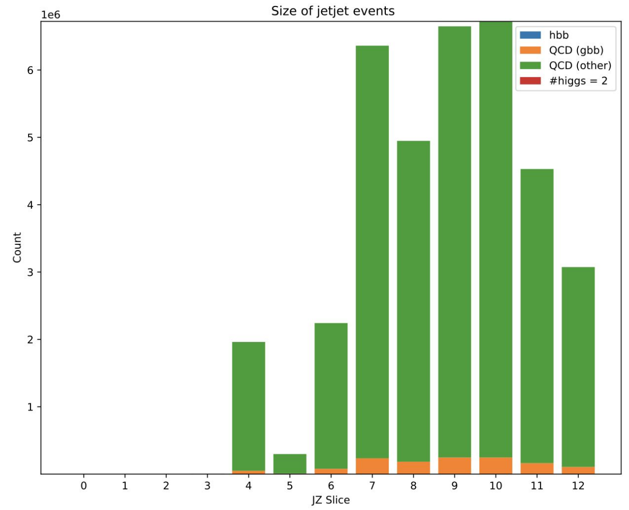
Sample Overview

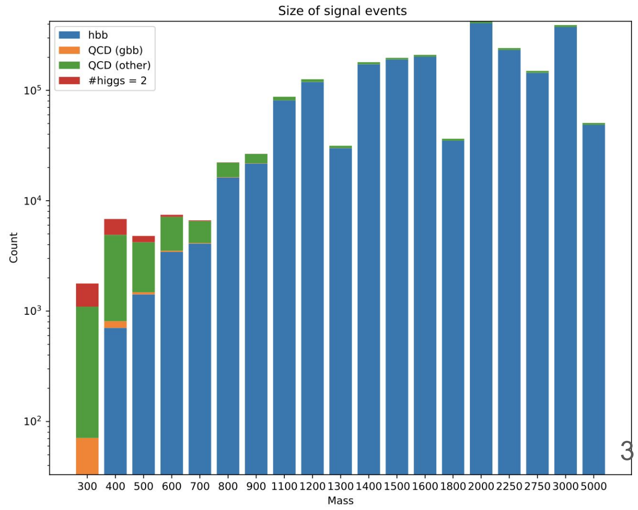
Samples used for each label

Hbb jets: hh_bbbb samples (Mass = [1000, 1100, 1400])

QCD (gbb): jetjet (Jz Slice = [5])

QCD (other): jetjet (Jz Slice = [5])





Thanks Zhicai for inputs!

Exploring the Data

Calorimeter, Track, and jet information were used as feature variables in separate models.

The following are the per-element features that were totally used for each type of data

Fatjet Data: ['C2','D2','e3','Tau32_wta', ‘Tau21’, ‘Split12','Split23']

Track: ['d0','z0SinTheta','qOverP', 'd0Uncertainty','z0SinThetaUncertainty']

Jet Selection: nConstituents > 2, pt>500e3

Truth Labels:

Hbb signal: GhostHBosonsCount==1

QCD (gbb): (df.GhostHBosonsCount==0)&(df.GhostBHadronsFinalCount == 2)

QCD (other): (df.GhostHBosonsCount==0)&(df.GhostBHadronsFinalCount != 2)

Other jets were excluded

Cont.

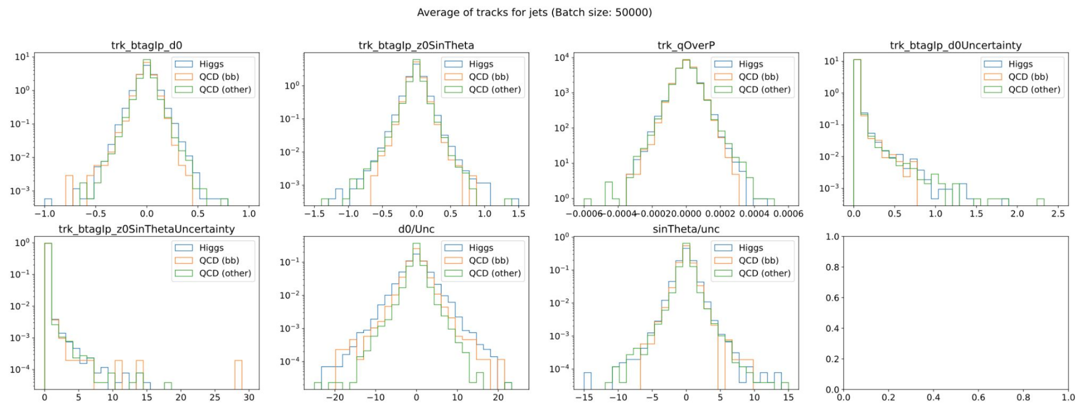
Data singularization: Engineering/Limiting # of tracks

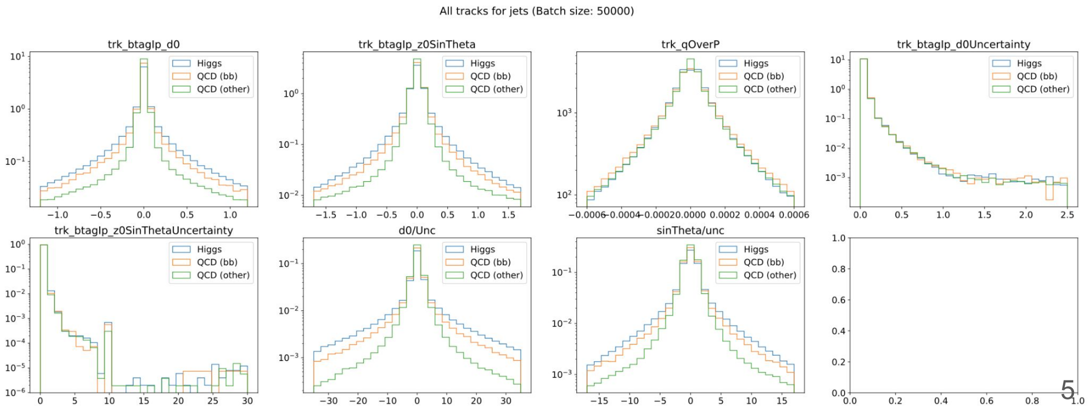
We investigated whether limiting/engineering track data is useful

Overall Track average: There is not much variance between labels

All tracks together: Discrimination in the tails of d0, z0

Certain features are distributed normally around 0 --> z-scoring, taking absolute, regular normalization

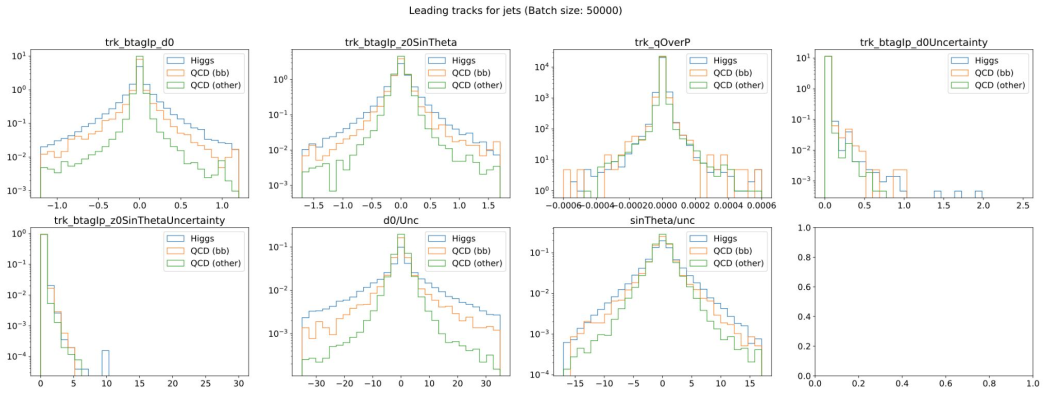


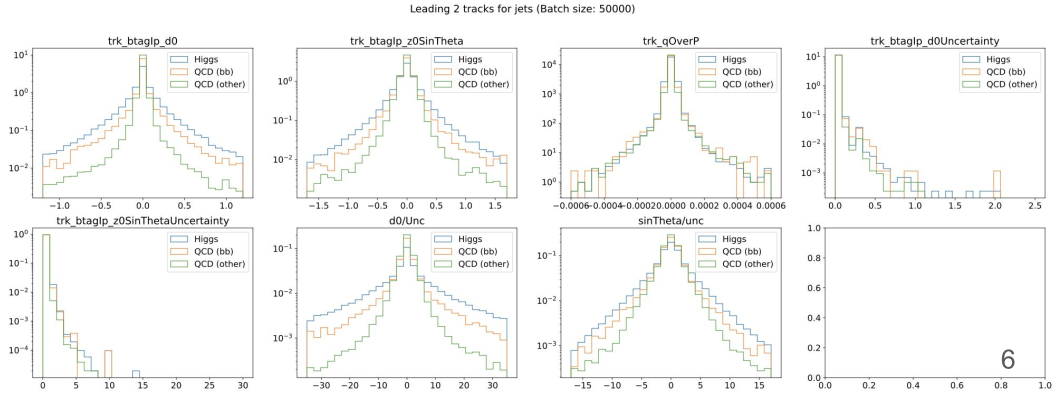


Observing Leading tracks

Variation between labels is a lot more distinct when considering the leading tracks of jets.

All tracks were kept in input data, since model performance increased with more tracks included.





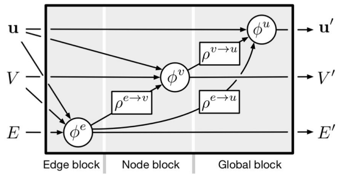
When taking leading 2 tracks: Jets with <2 tracks were excluded, since graph_nets could not recognize dummy axes

Graph Architectures Investigated

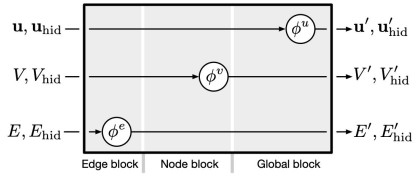
Dataset -> [0.47, 0.06, 0.47], batchsize = 10,000

Training size: 150,000 jets, Testing size: 50,000 jets

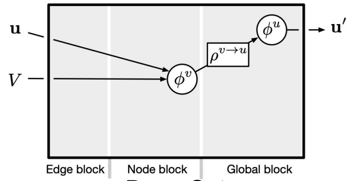
GNN Architectures Investigated:



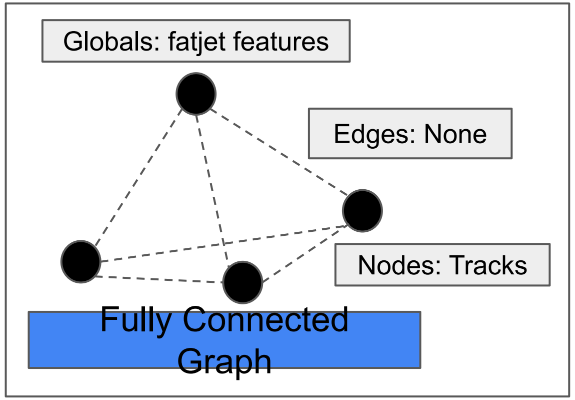
Full Graph Network



Graph Independent



Deep Set



Edges - None,

Nodes - tracks (d0, z0, etc),

Globals - fat jet info (c2, d2, t32, etc)

Graphs were fully connected with edges as zero vectors.

- Since the fatjet globals were not normalized, a graph independent normalization layer on globals was included in each model.
- Prediction logits: output graph's global variables

Image ref. Relational inductive biases, deep learning, and graph networks arXiv:1806.01261

GNN performance: High-Level

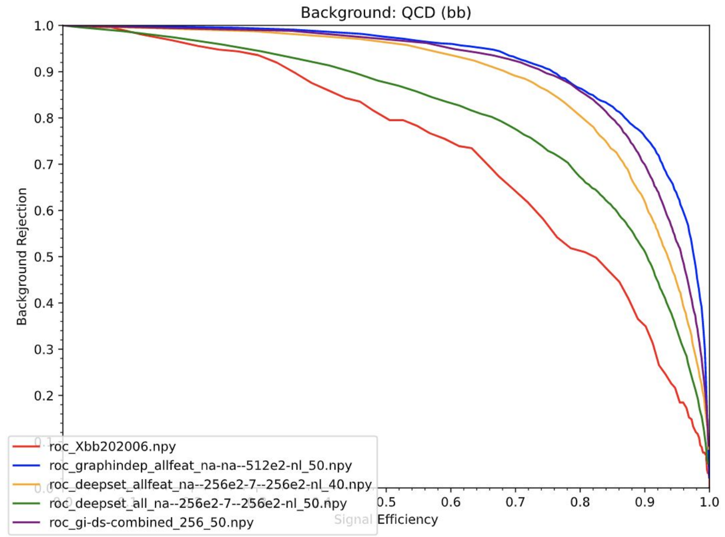
Red: Current ATLAS Xbb tagger

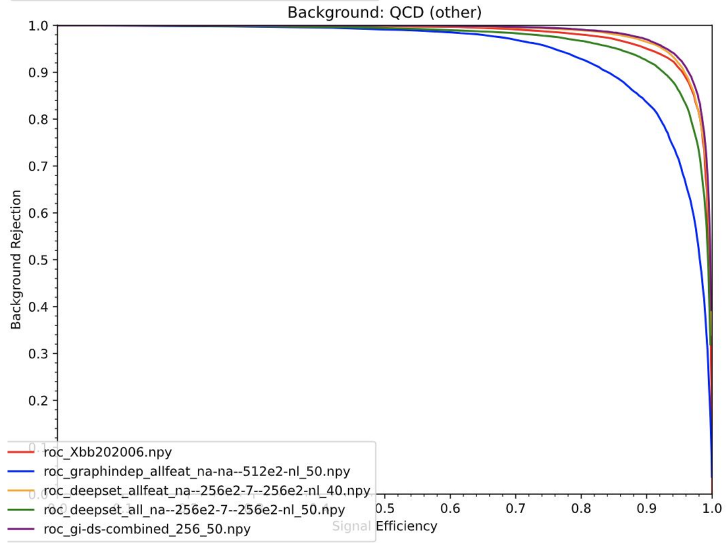
Blue: Graph Independent on tracks+features

Yellow: Deep Set on tracks+features

Green: Deep Set on tracks

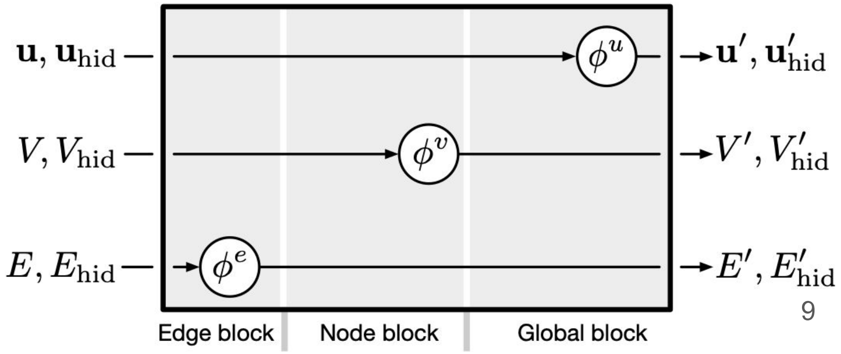
Purple: GI-DS model on tracks+features





Graph Independent Networks

- No interaction between graph attributes (node, global, edge) in graph independent networks.
- Our GraphIndep network **operates exclusively on fatjet global features (no tracks) with only a global function**
- Hence, we are emulating regular feed-forward NNs with this architecture.



Graph Independent Best

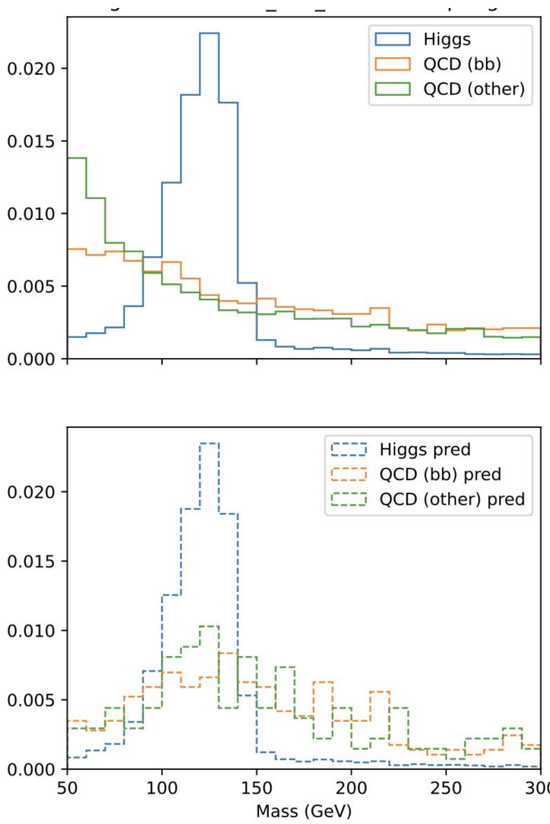
Model Blue

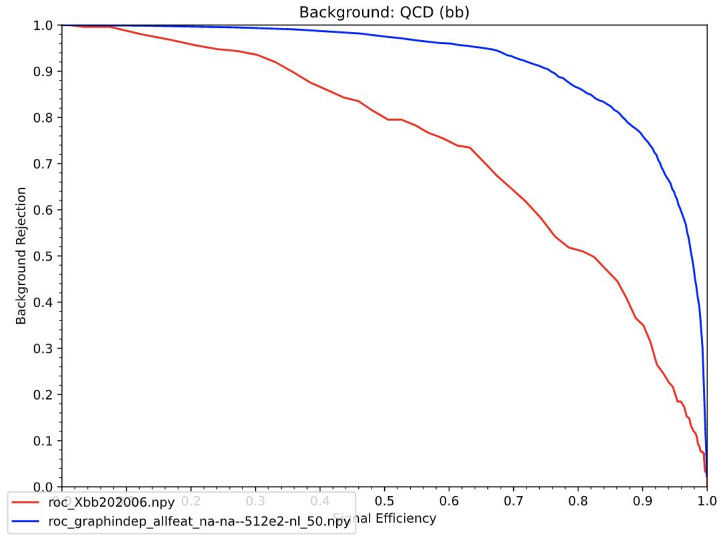
Model Arch: (global:[512, 512, 3])

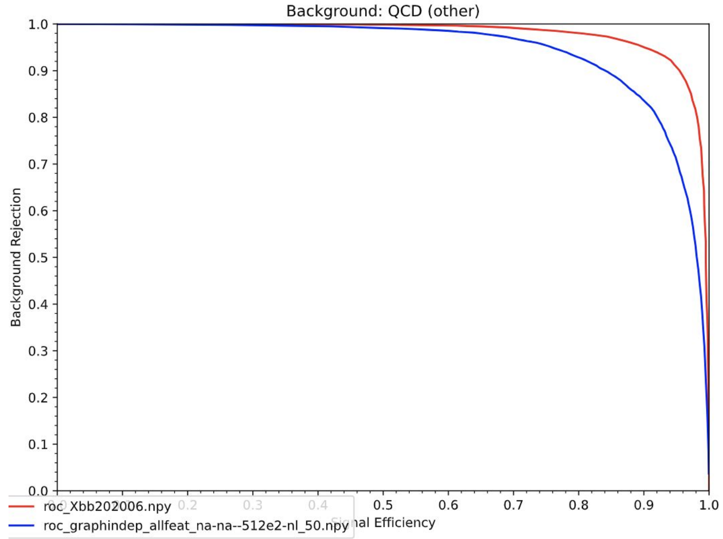
GI model crosses 1e-1 AOC for both labels.

AOC

QCD (bb) - 0.09
QCD (other) - 0.06

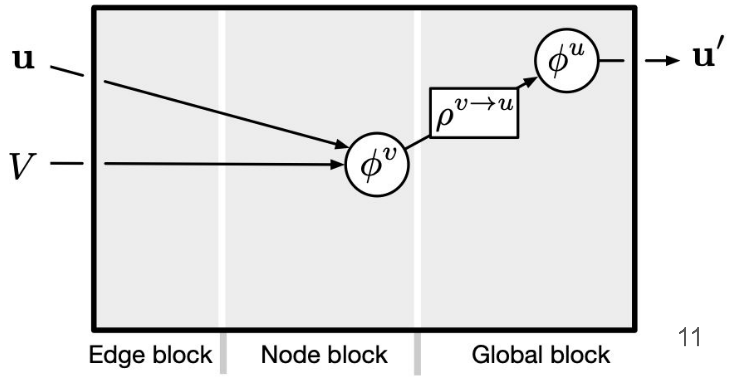






Deep Sets

- Most appropriate graph net architecture for our data, since we have node and global info w/o any emphasis on edges.
- The output graph's global variables are updated based on node and global information, hence becoming our output logits.
- This model architecture performed best for QCD (other), since it had track d0 parameters.
- 2 versions of iterations
  - Training over tracks (nodes) and fatjet feat (globals)
  - Training over only tracks (nodes)



DS tracks+feat best

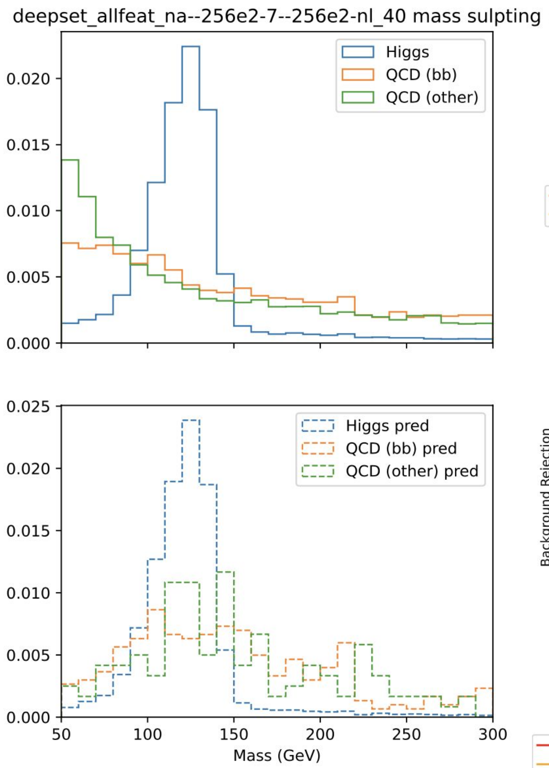
Arch: (node:[256, 256, 7],

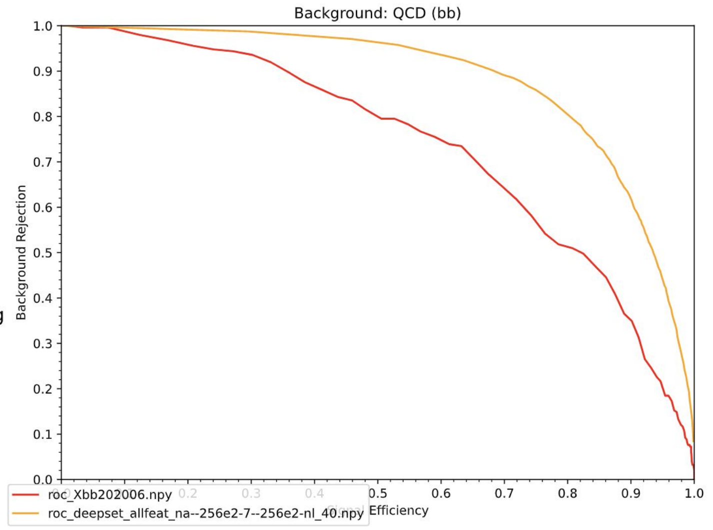
global:[256, 256, 3])

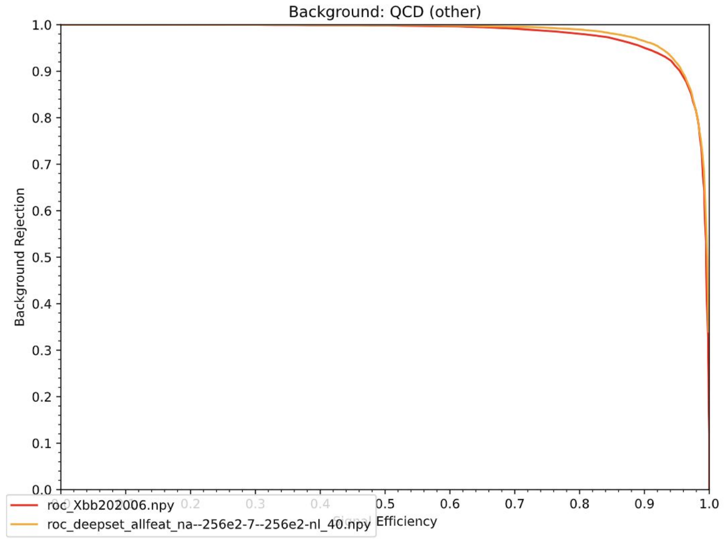
AOC

QCD (bb): 0.12

QCD (other): 0.017







DS tracks best

Model Green

Arch: (node:[256, 256, 7],

global:[256, 256, 3])

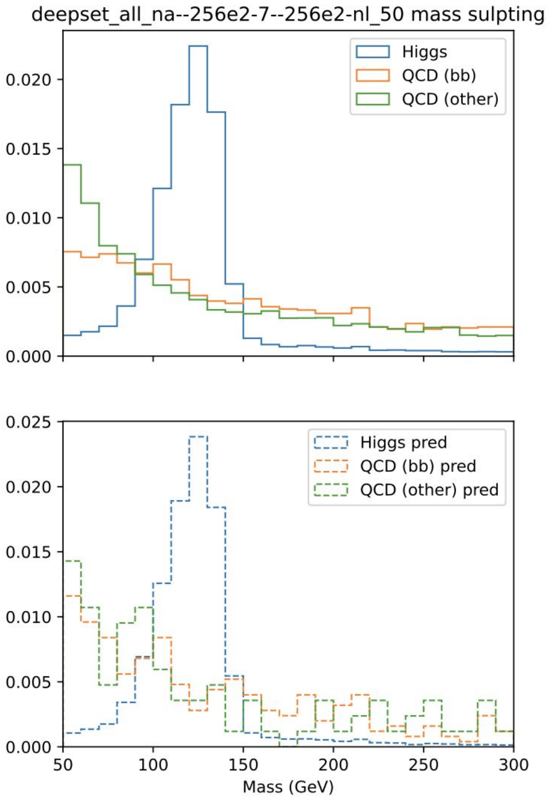
AOC

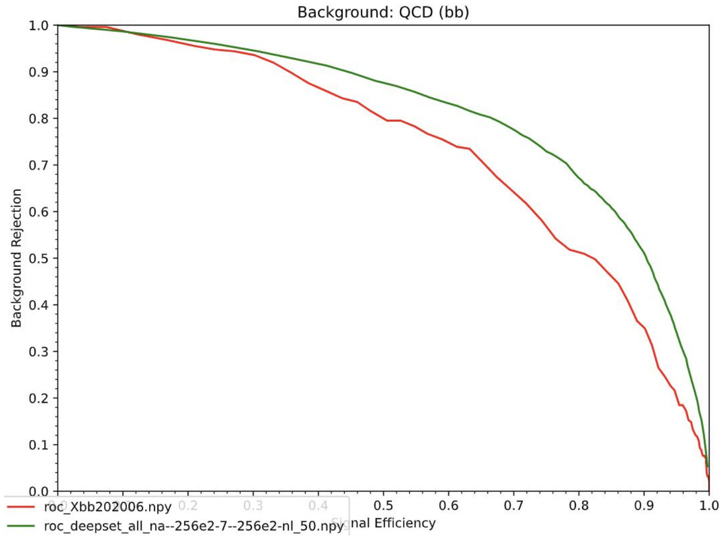
QCD (bb):

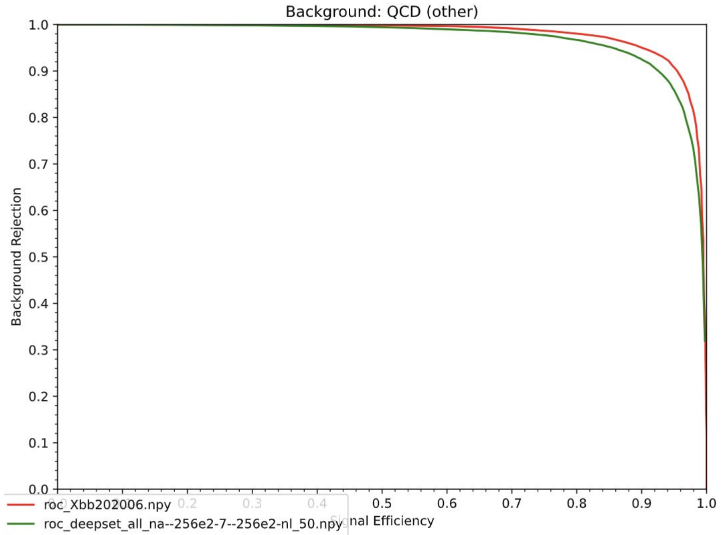
QCD (other):

- Lower-mass distributions of Labels 1 & 2 somewhat match original distribution.

- Reduced mass sculpting effect might be due to absence of jet features

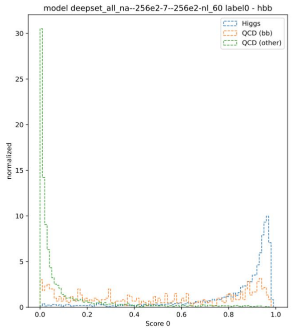


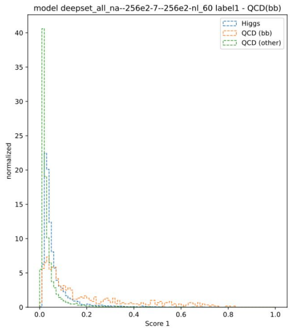


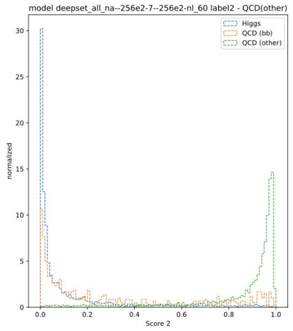


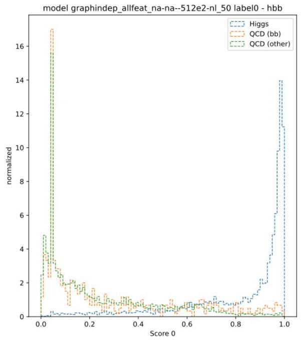
Comparing the best single models per label

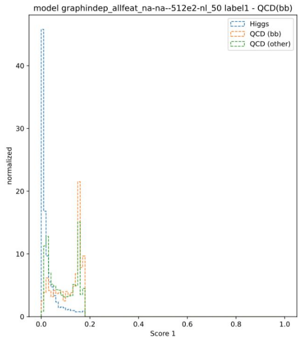
Graph independent networks seem to perform significantly better for QCD (bb) rejection, while Deep sets are marginally better for QCD (other) rejection.

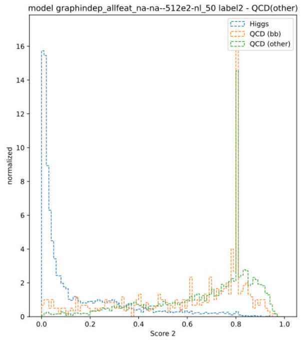












Model Yellow
Deep Sets (node - [256, 256, 7], global - [256, 256, 3])
50 epochs

QCD (bb) - 0.12 AOC
QCD (other) - 0.017 AOC

Model Blue
Graph Independent (global - [512, 512, 3])
50 epochs

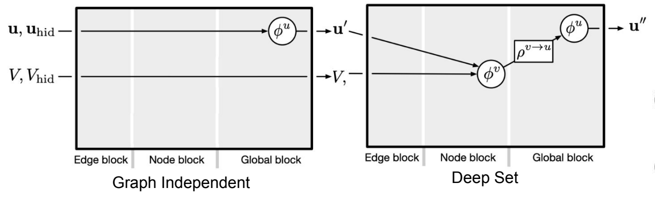
QCD (bb) - 0.09 AOC
QCD (other) - 0.06 AOC

Combining Graph Indep & Deep Set

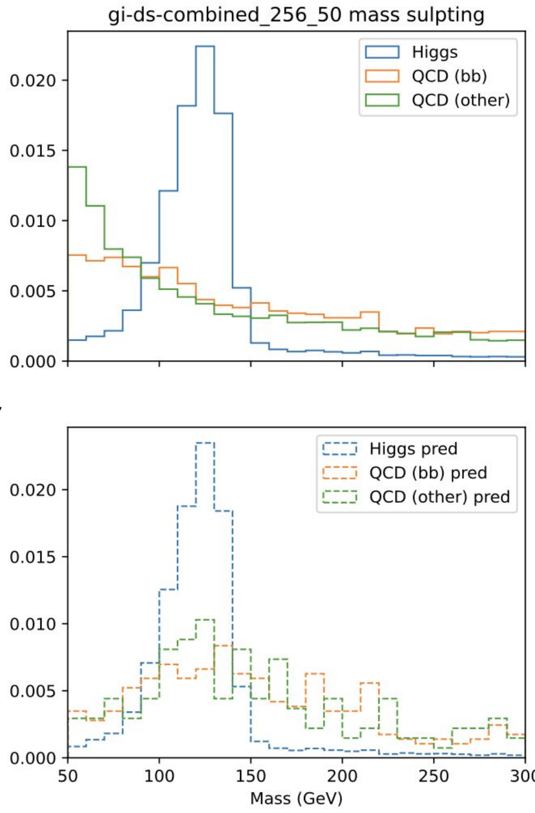
Combining GI & DS through 2 methods:

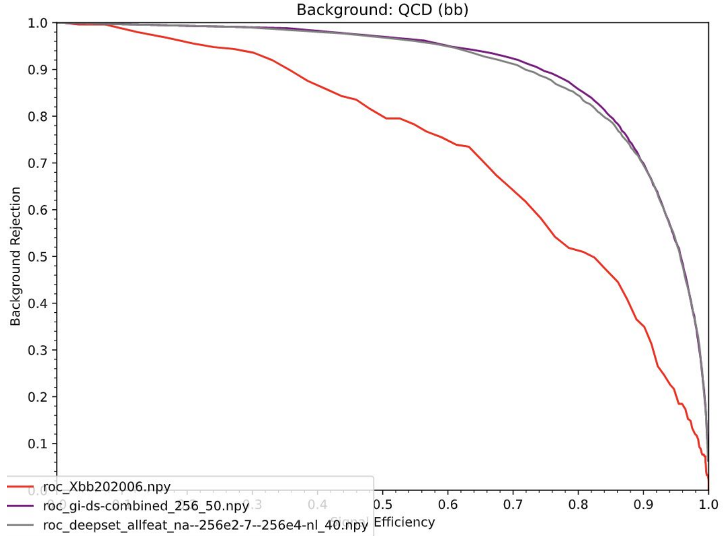
- Increasing DS node model complexity
- Training a GI-DS pipeline

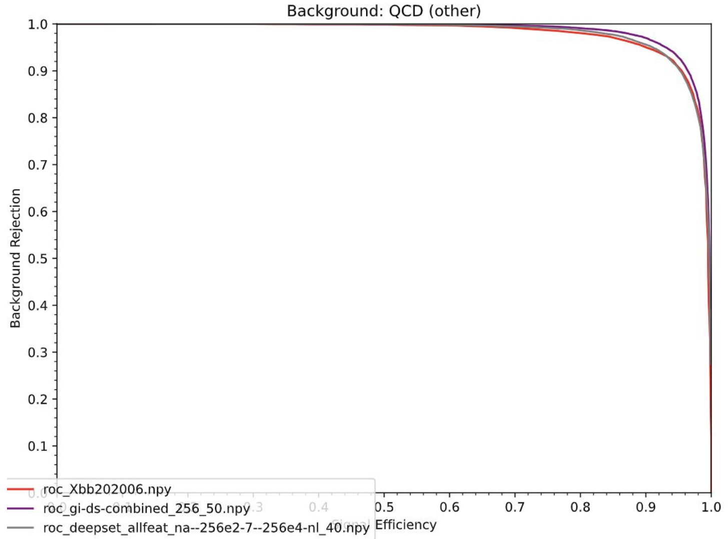
Purple: Best GI-DS model



Gray: Best 4-hidden layer DS model. (Orig. 2 hidden layers -> global func)

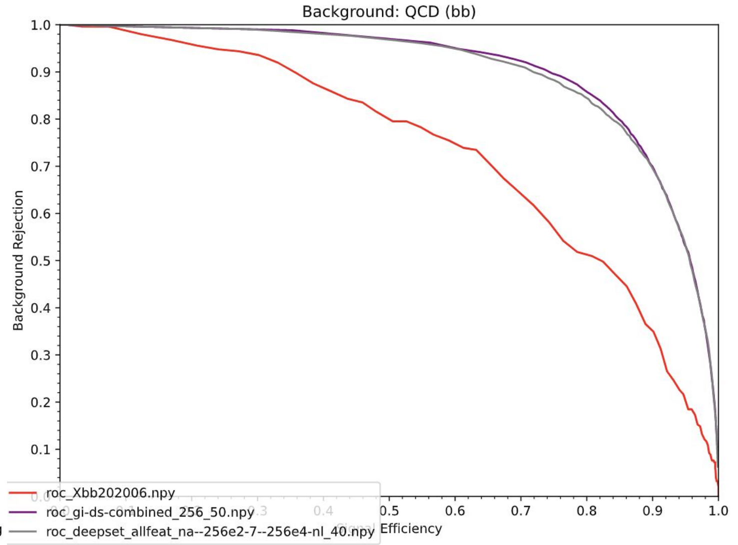


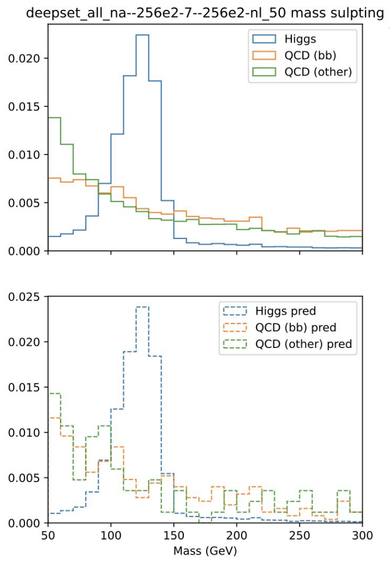


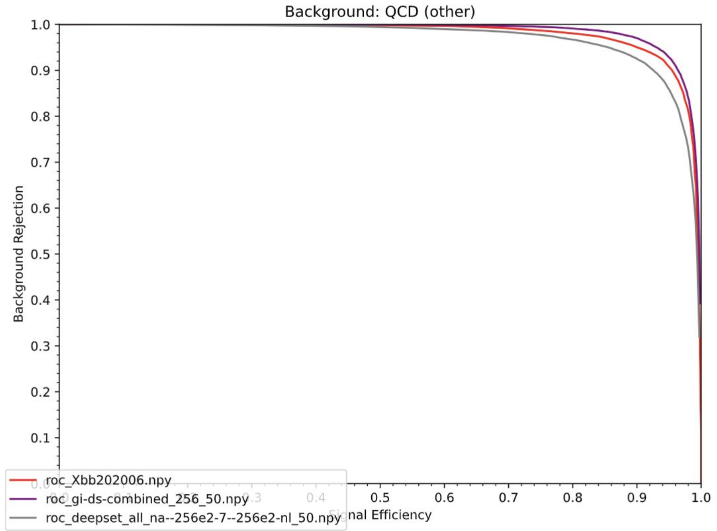


Conclusion

- Graph Neural Networks could match and beat the Xbb tagger
- Presence of Jet features results in sculpting issue for QCD labels
- Combined GI-DS model beats Xbb tagger, but suffers from mass sculpting
- Deepset on tracks has notable performance with reduced sculpting effect







Appendix

Data Stats Cont.

Initial Stage of GNN work

Using madgraph-simulated samples for [hbb, gbb, other] discrimintation that also forced mass threshold

Nature of Datasets and Model Architecture

Data -> Graph Conversion

Track data was used for the Graph Networks. Each jet was a fully connected graph with nodes representing the tracks and their information. No global weights were used and no edge weights were used.

Track data used : ['trk_btagIp_d0','trk_btagIp_z0SinTheta', 'trk_qOverP', 'trk_btagIp_d0Uncertainty', 'trk_btagIp_z0SinThetaUncertainty']

Model Architecture: 2 layers of Interaction networks, followed by a Relation Network

Each Interaction Network:

- Edge model: 2-layer MLP with ReLu activation
- Node model:2-layer MLP with ReLu activation

Utilizing graph global variables

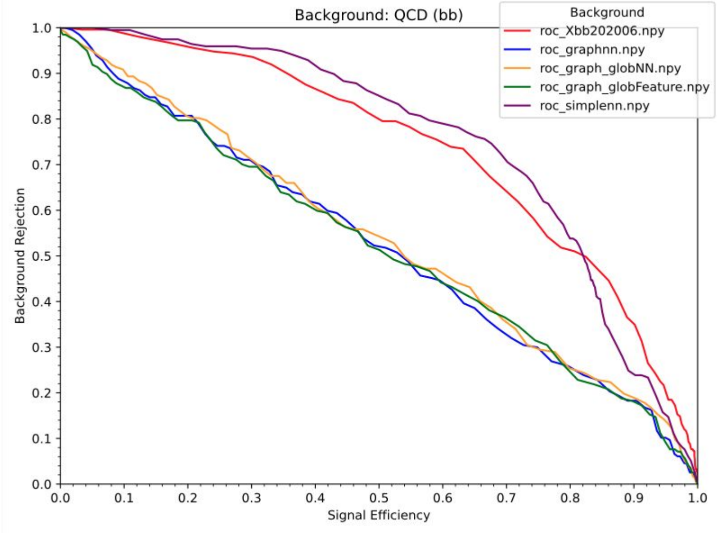
- graph_globFeature: jet mass
- graph_globNN: simplenn label 0 prediction
- Graphnn: no global variables

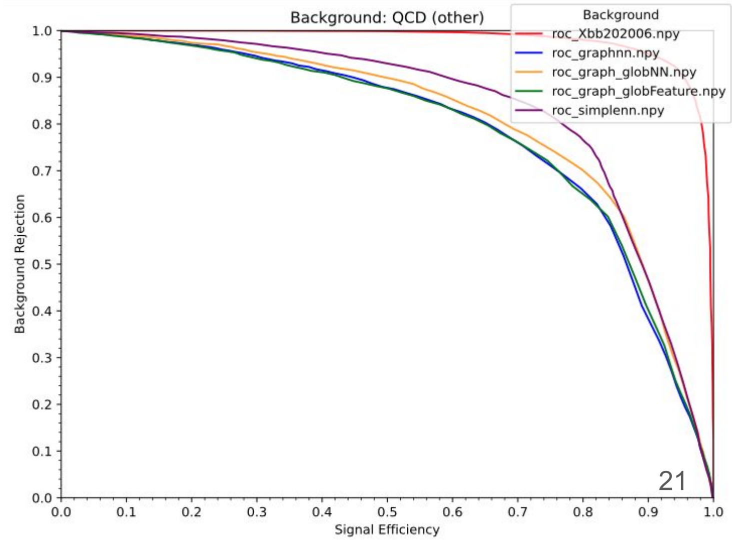
Graphs could only hold 1 global variable, not an array of multiple features

Obs: using mass shows virtually no improvement, while label 0 predictions display minor improvement towards the Feature NN model.

Providing all 3 simplenn prediction values could have set a baseline for the graph nn's performance, which was not possible.

At the time, I was unaware that graph_nets could only take singular global variables, and had actually tried to run the model on all features and all prediction outputs. If a single variable proved to be statistically significant from either sets, I could have specifically investigated by training with it.





Graph Networks on Leading 2 tracks

Batch size 10,000

Train: 21 (r10201), Test 10 (r9364)

graph_leading2: testing from batches of same data

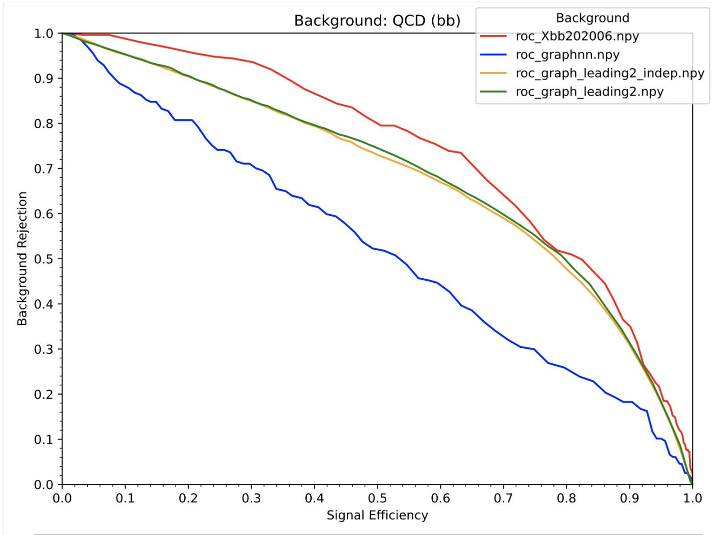
graph_leading2_indep: testing on different data

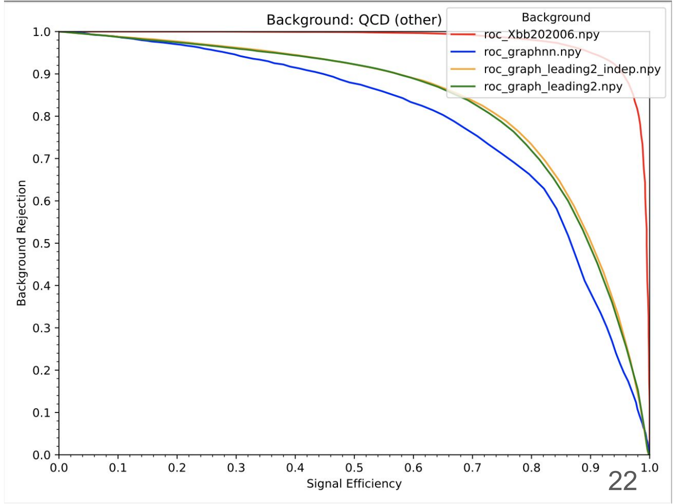
graphnn: baseline graph network from previous data (outdated)

Raw track data was used, which was not engineered.

Leading2_indep essentially shows that a model trained on a larger, structured, and engineered dataset outdid the previous graph network (which was trained on <100k jets). Better comparison will be made in following slides.

Model stats on next slide





Graph Network on all Tracks

To test whether the current data and model arch could grant any better performance, we trained a graph network on all tracks.

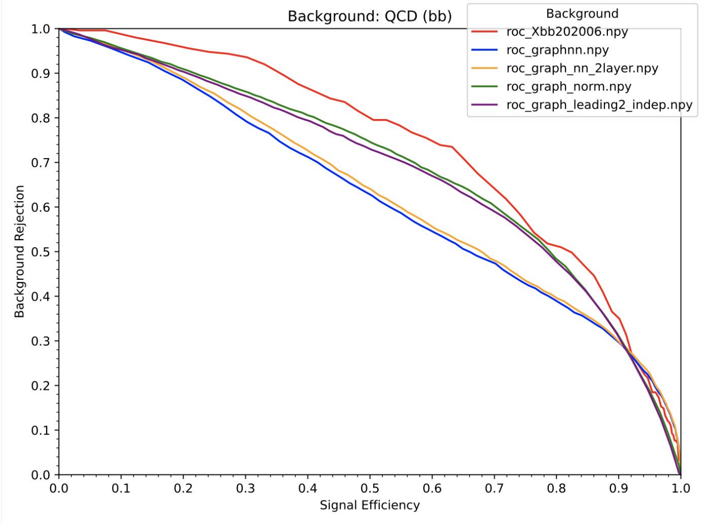
graph_nn_2layer: 2-layer network on all tracks, 20 epochs

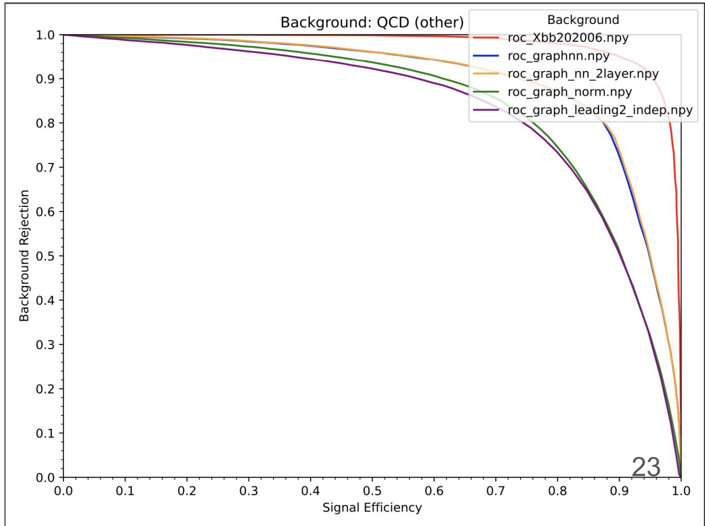
graph_nn: 1-layer on all tracks, 10 epochs

graph_norm: 2-layer network on leading 2 tracks, 20 epoch

Results:

- Improvement in QCD (other) rejection w.r.t graph_norm
- Decrease in performance for QCD (bb) rejection
    - better performance with only the two leading tracks -> need to figure out how to teach GNN to select only important tracks

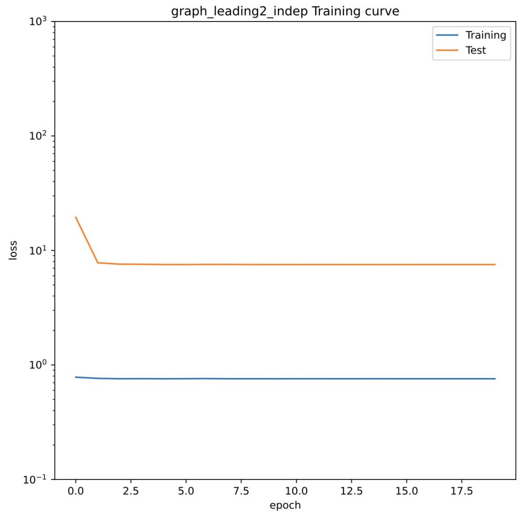


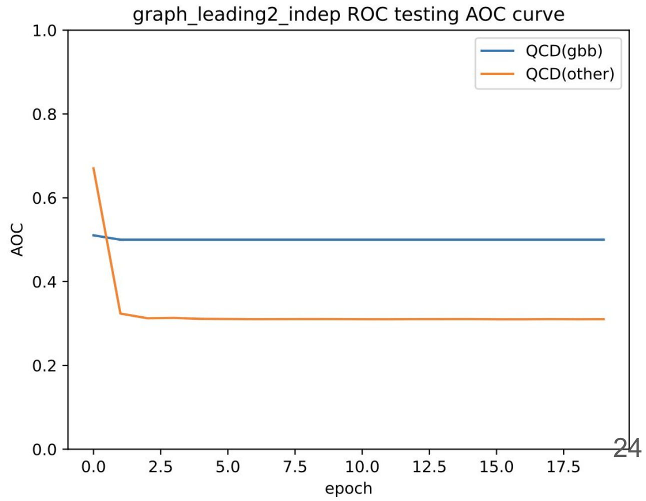


Leading_2: Model stats

There did not seem to be learning beyond the first epoch.

From this, I felt that there was no intrinsic characteristic of the leading tracks themselves other than the distribution of their features. The most a model could have learnt was to discriminate along the feature distribution boundaries.





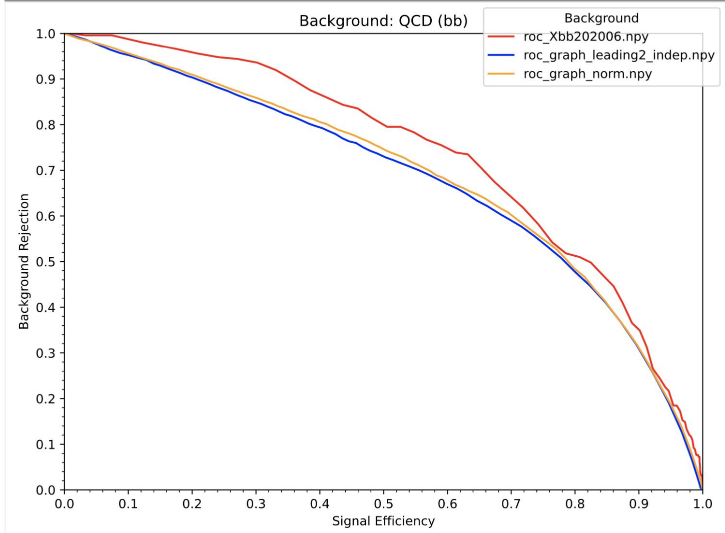
Training on abs, normalized graphs

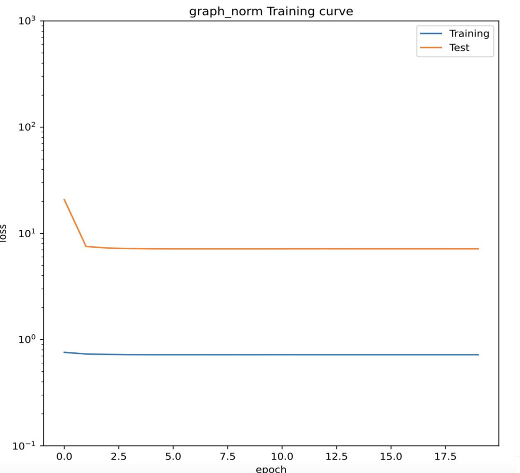
A graph network was trained on the absolute, normalized track features of the leading 2 tracks.

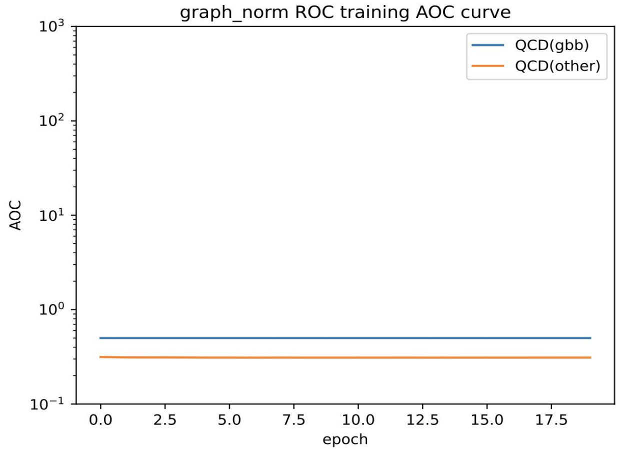
graph_norm: current graph network

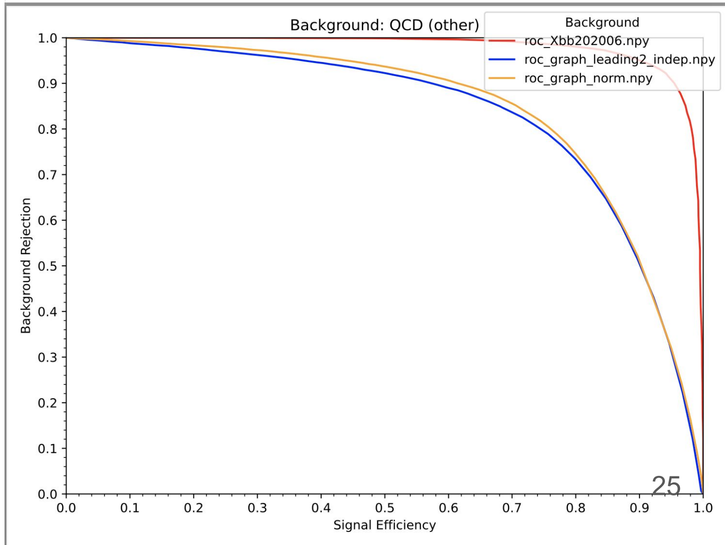
graph_leading2_indep: network trained on raw leading 2 tracks

No improvement, due to my simple normalization



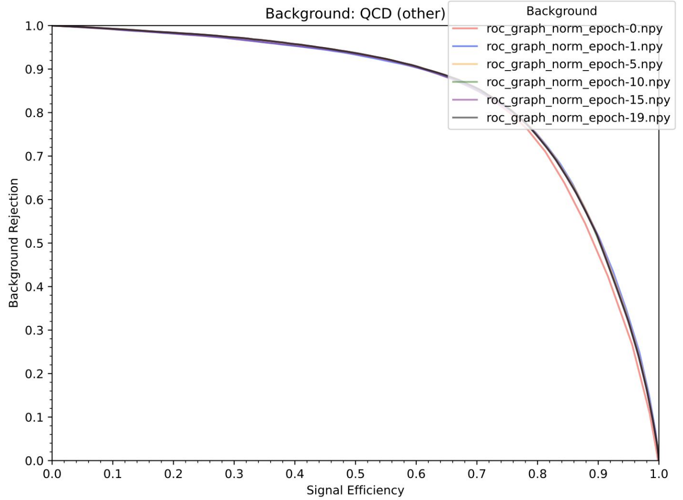


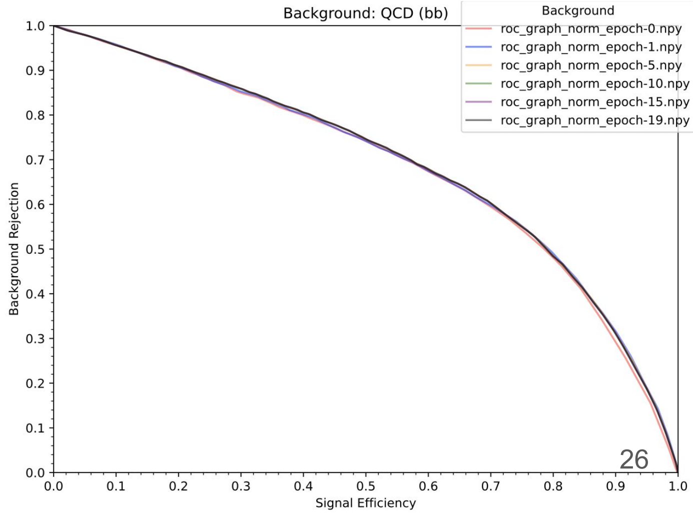


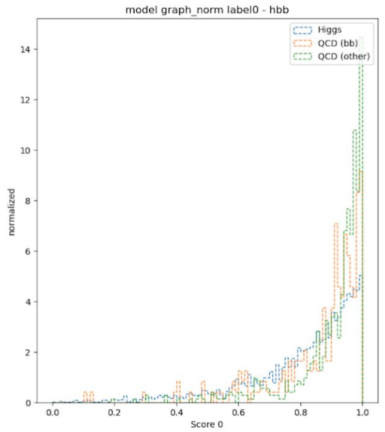


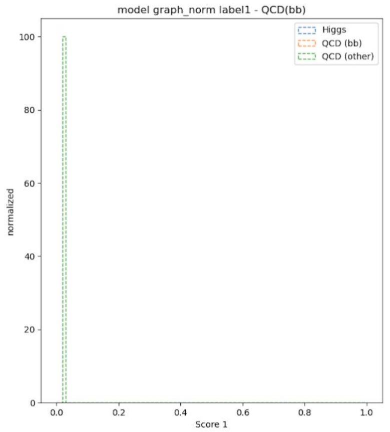
Graph_norm stats

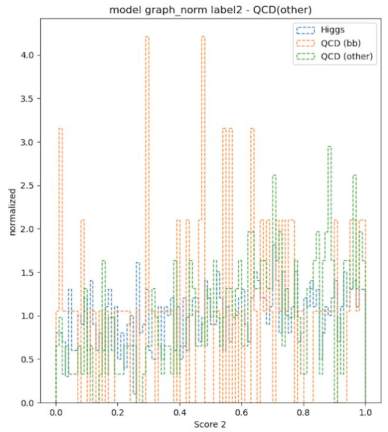
Right: ROC curves per-epoch, Bottom: Final score distribution in a batch









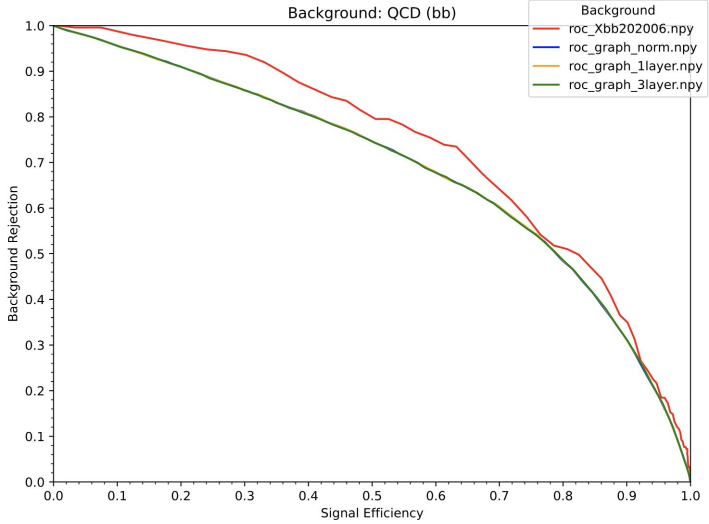


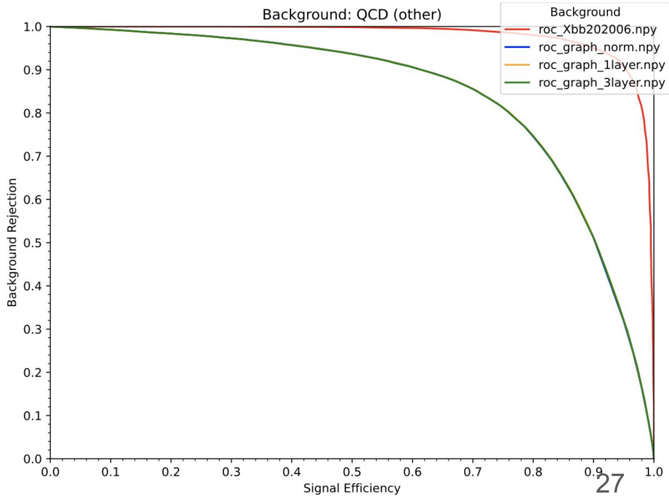
Model Complexity

Testing different number of Interaction network layers on the leading 2 normalized data

I felt the smooth nature of the roc curves indicated some missing dimensionality/complexity, which could be speculated to be shown through the absence of learning in loss curve.

However, all models have the exact same performance, indicating that limitations lie elsewhere, in model architecture or data



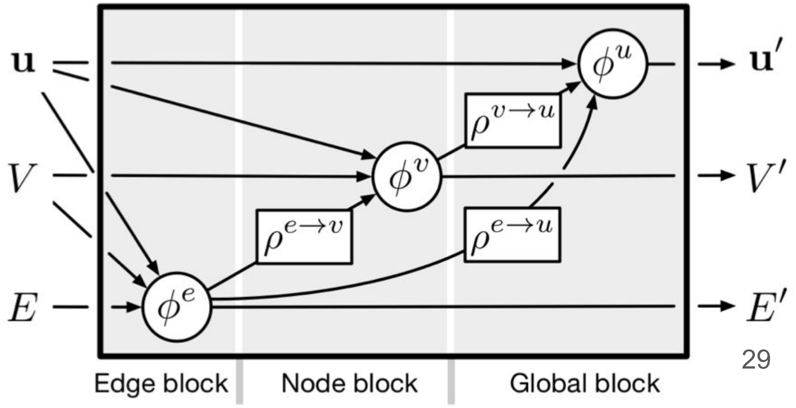


GNN Architectures Cont.

Graph Networks

- Standard architecture for operating on graphs.
- Since our graphs lack edge info, this architecture becomes slightly inappropriate.
- Since Node and global updates came after edge updates, we assumed that the lack of edge data would not affect performance.

I have lost most of the plots for these networks, but it was clear that this architecture was inappropriate.

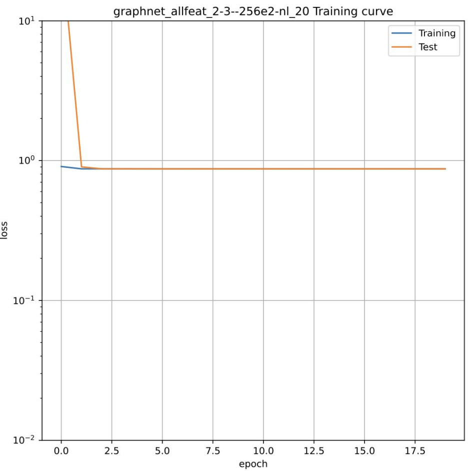


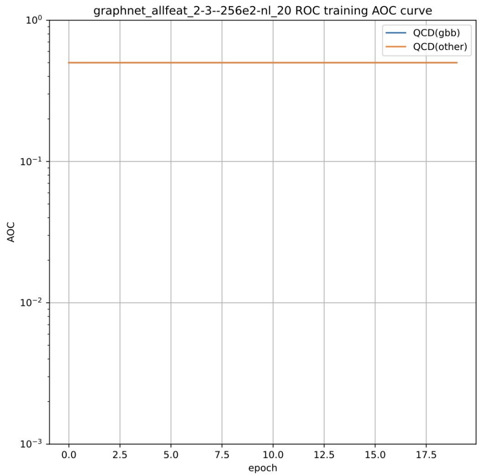
Graphnets, an example

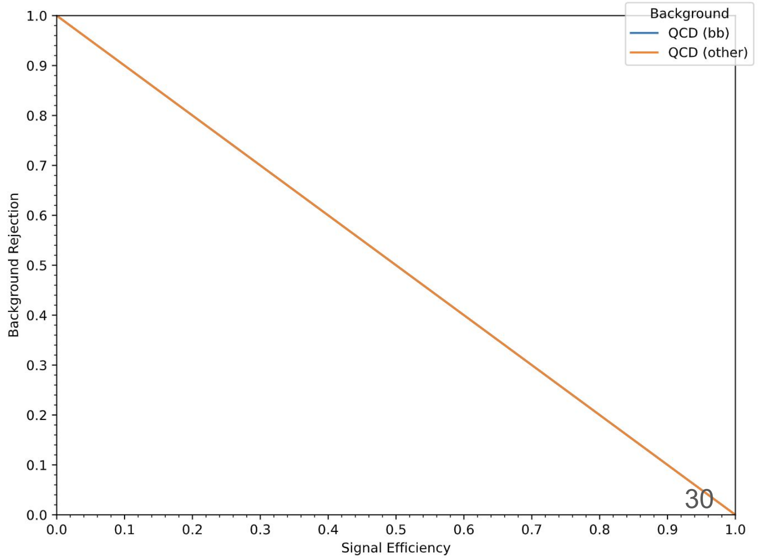
Data: all tracks with fatjet parameters as global variables

Graph network models: (Edge: [2], Node: [3], Globals: [256, 256, 3])

This network showed no learning, and hence was ineffective







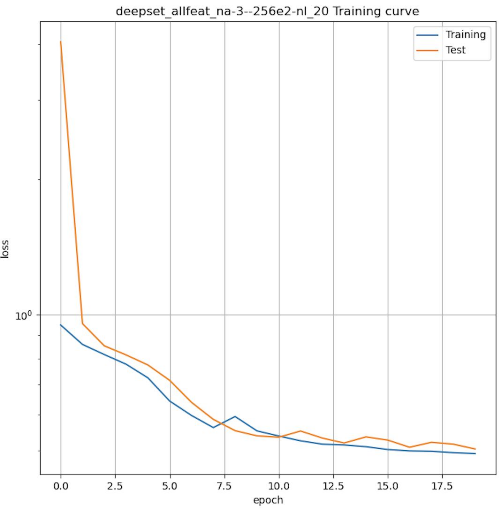
Reducing Learning Rate

Reducing the Learning rate was only observed to bring the model to the stable optimum faster, preventing over-correction. Hence, it was only useful for reducing training time, since all learning fluctuations stabilized at ~5e-1 loss, regardless of learning rate.

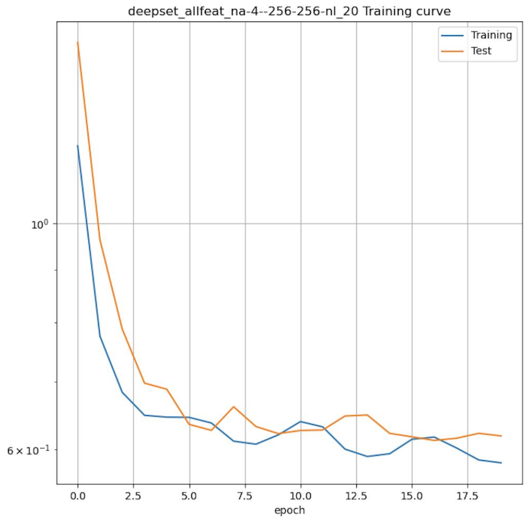
Learning rates below 0.01 were observed to train much slower than normal, requiring immense training time.

Learning rate: 0.1 → 0.01

Learning Rate: 0.01



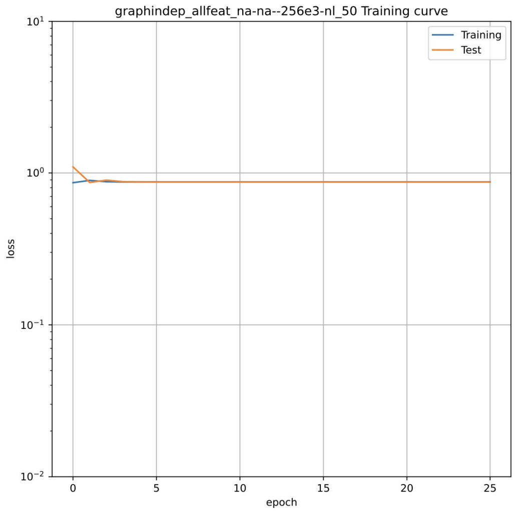
Learning Rate: 0.1

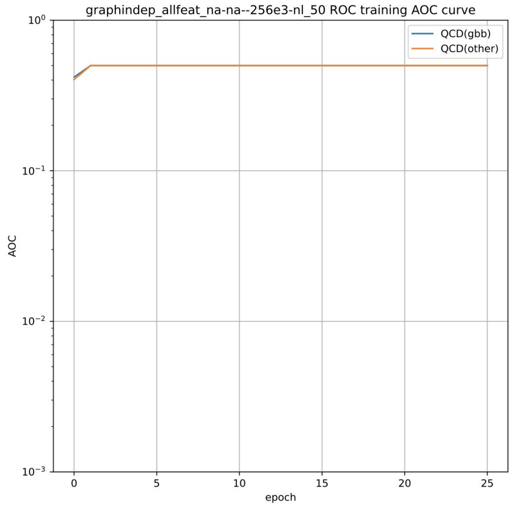


GI Increasing Complexity - 3 hidden layers

Graphindep_allfeat_na-na–256e3-nl_50: graph independent (global: [256, 256, 256, 3]) for 30 epochs
(Error in naming convention)

The vanishing gradient issue is greatly significant with an increase in complexity.
Possible approach-> reducing the learning rate, or individually training network layers





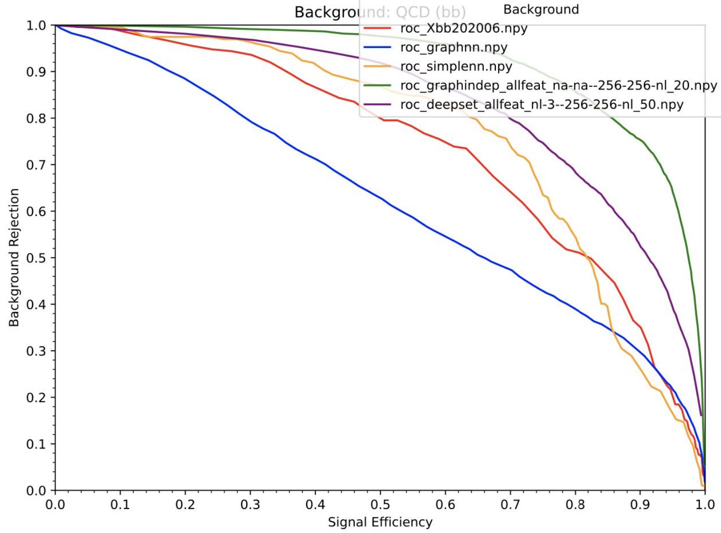
Simpler GI Model

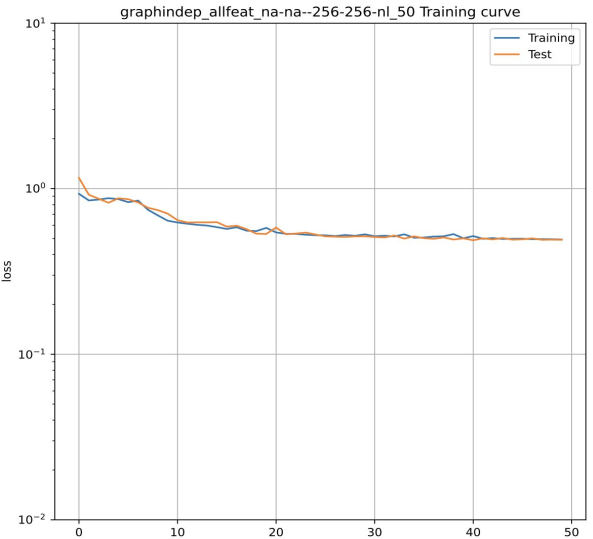
Graph Indep on all tracks w/ fatjet global info, (Globals: [256, 256, 3])

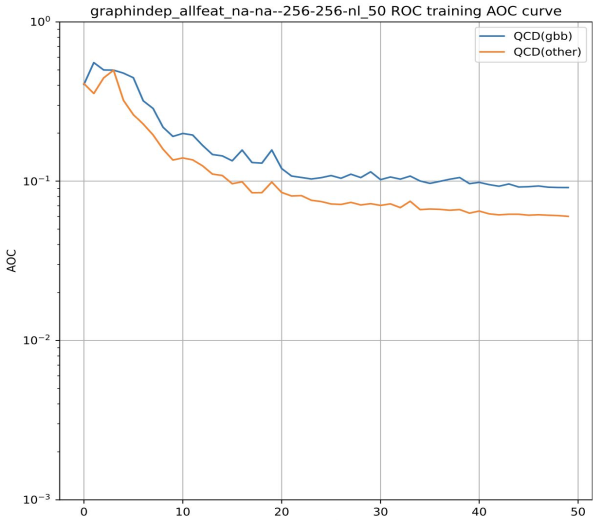
Simplenn: baseline feature nn

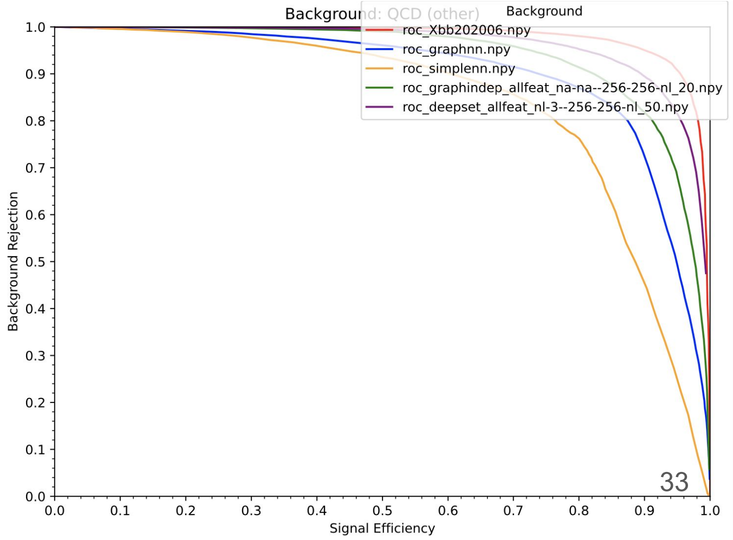
Graphnn: baseline previous graphnn

Graphindep_allfeat_na-na-256-256-nl_20: given model w/ 20 epochs training





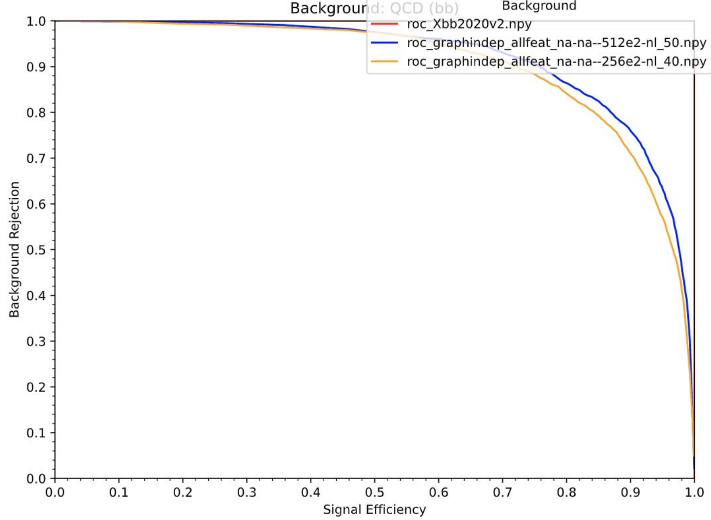


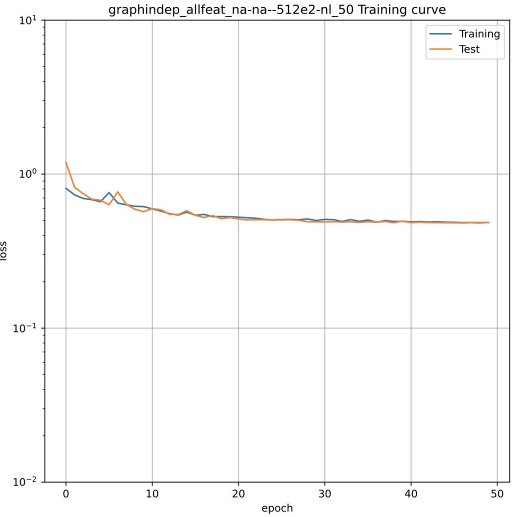


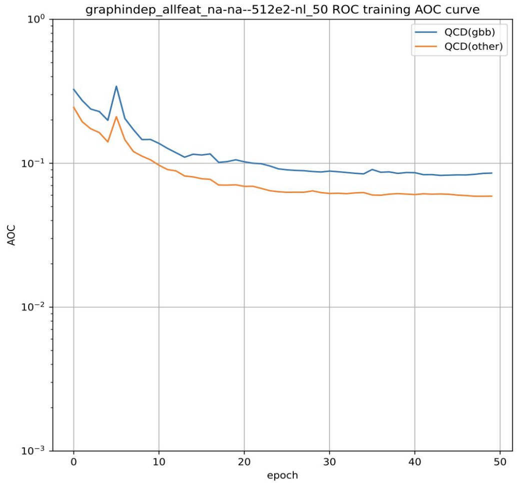
GI increasing Complexity - 512-node layers

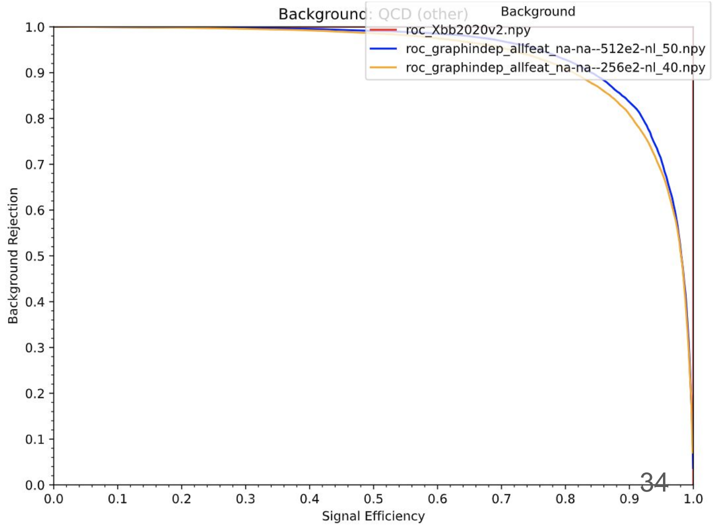
There a marginal improvement for both labels

(Error in 256-node layer model over 50 epochs, so its 40th epoch is shown)







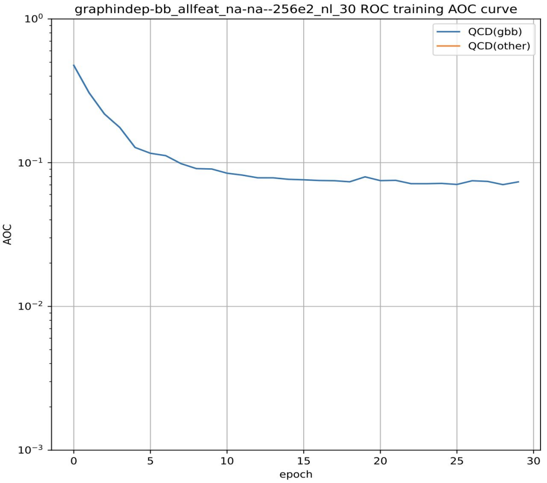


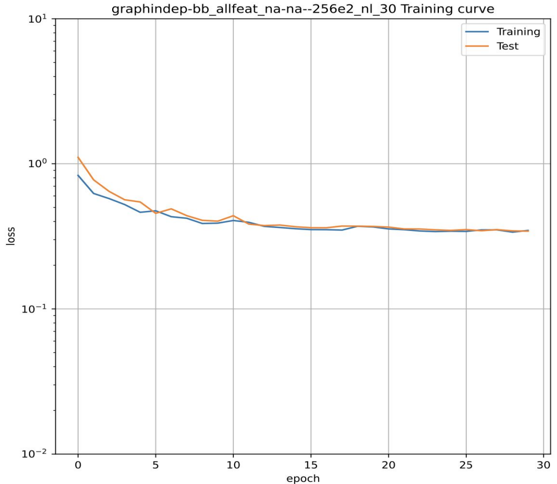
A 2-label Graph Independent QCD (bb) network

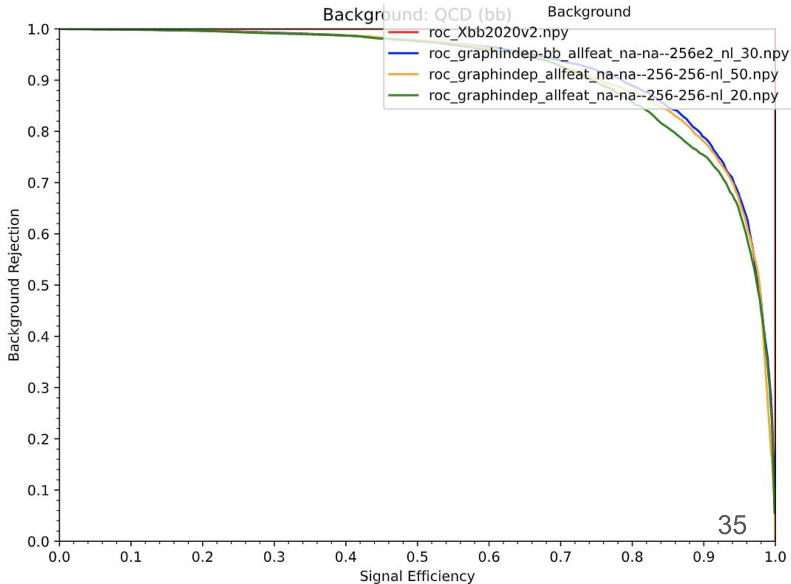
Blue (current): GI 2-label model (global:[256, 256, 3]) over 30 epochs

Yellow: GI model (global:[256, 256, 3]) over 50 epochs

Only a marginal improvement was observed. Given the reducing learning gradient, a significant increase is unlikely given longer time.







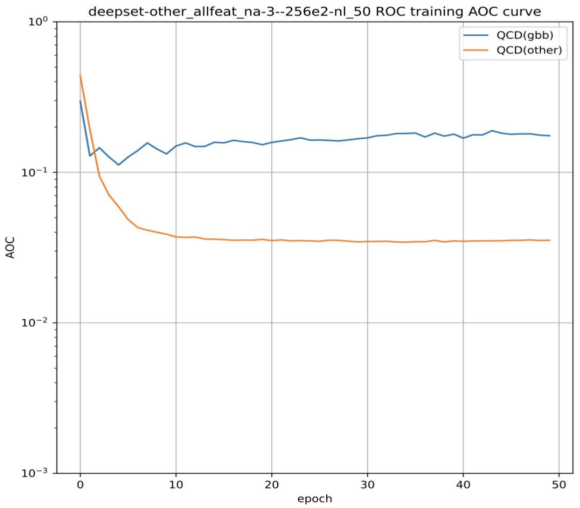
A 2-label Deep Set QCD (other) network

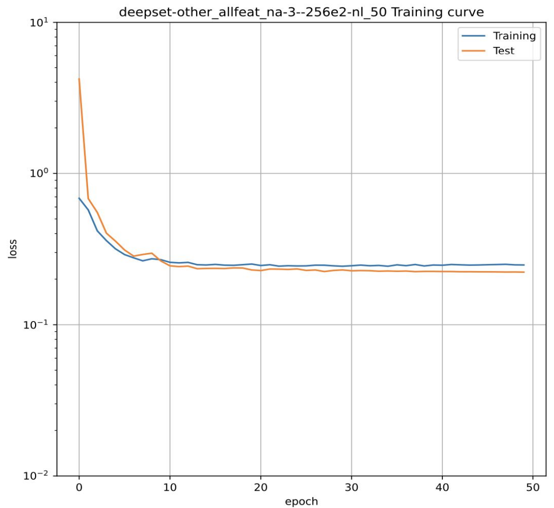
Red (Current): Deep Set 2-label model (node: [3], global: [256, 256, 3]) over 50 epochs

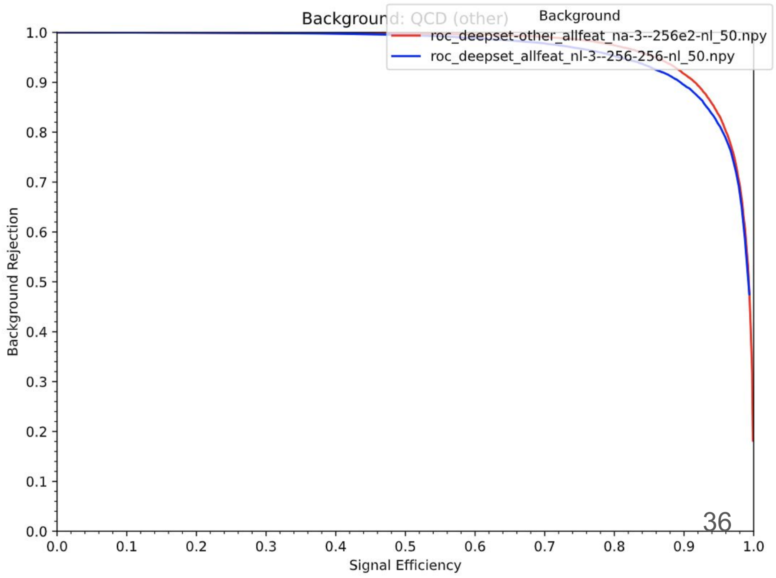
Blue: Regular Deep Set model (node: [3], global: [256, 256, 3]) over 50 epochs

There is a marginal improvement in rejection.

(Dataset had 0.002% QCD (bb) jets so a curve is shown, must be ignored)







GI-DS Combined Arch Cont.

Individually training layers

Since network architectures are different, I decided to investigate a combination of Graph independent networks and deep sets.

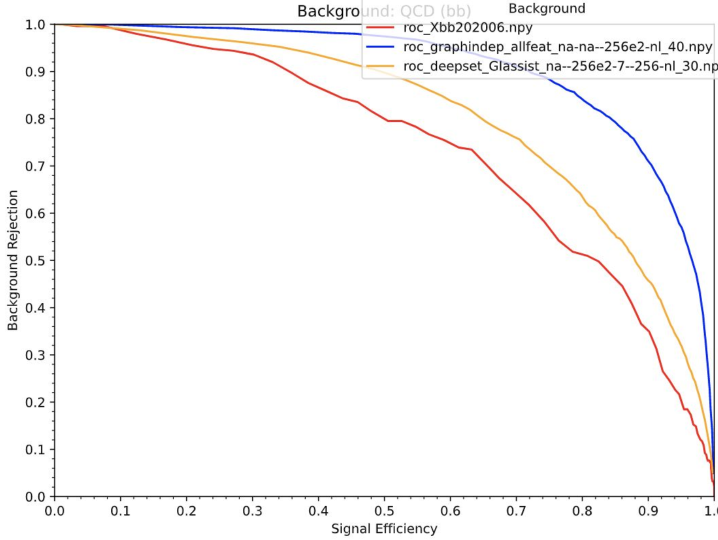
The initial graph independent network had only a global model, which passed through all nodes while outputting predictions as global variables.

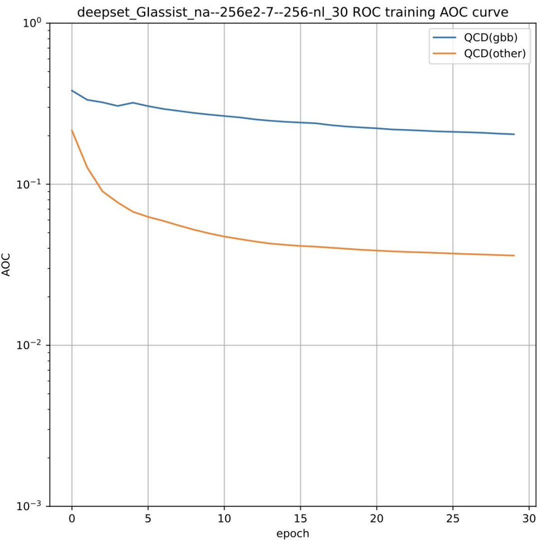
The outputted graph was then used as a transformed dataset for the deep sets’s training, which would aggregate the global variables with node track information.

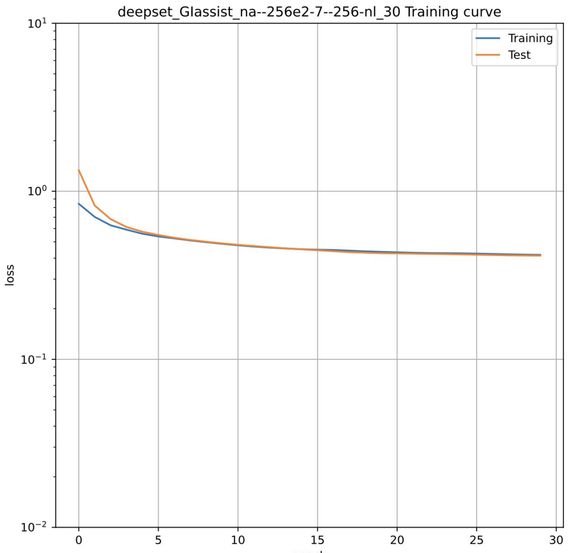
Potential area for exploration: Graph independent node model modifies and aggregates track nodes, with a secondary deep set operating over separate predictions by the global and node models

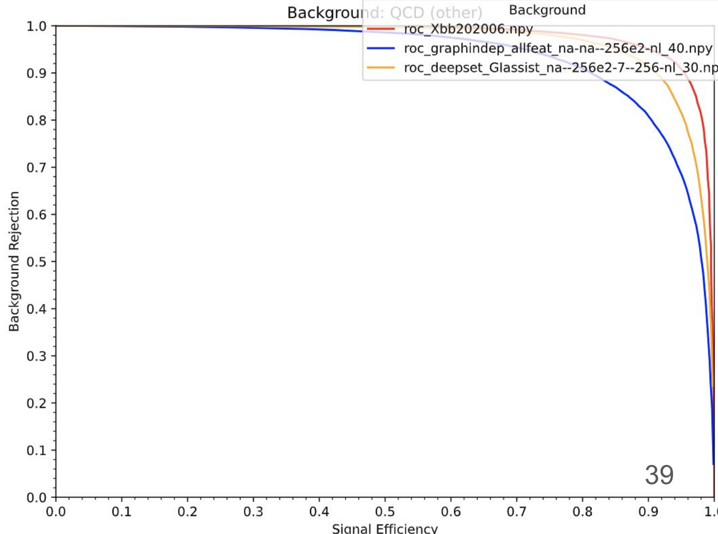
Primary: Graphindep(global: [256, 256, 3]) over 40 epochs

Secondary: Deepset(node: [256, 256, 7], global: [256, 256, 3]) over 30 epochs









Applying other Neural Network Architectures

ANNs, CNNs

Feature Variables Neural Net

Training an Artificial Neural Net on jet features

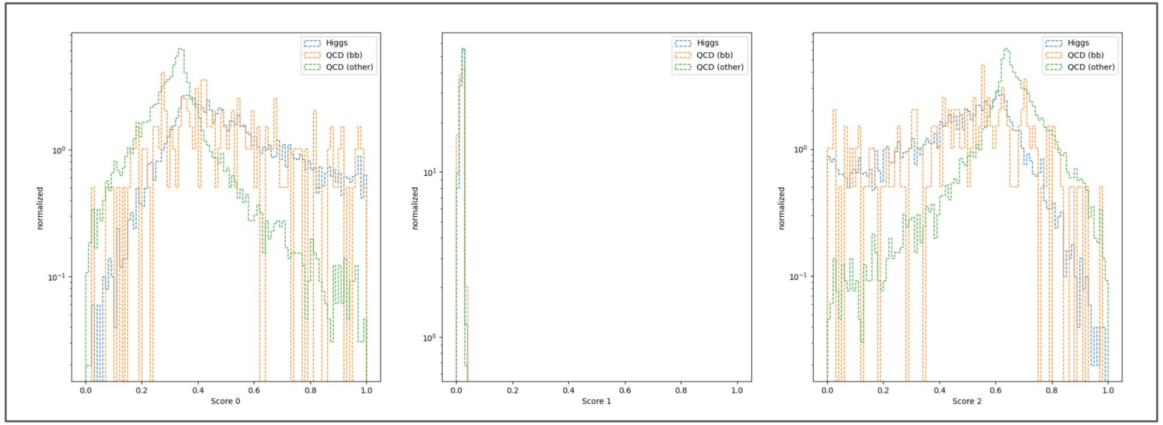
Simple Neural Network on feature variables

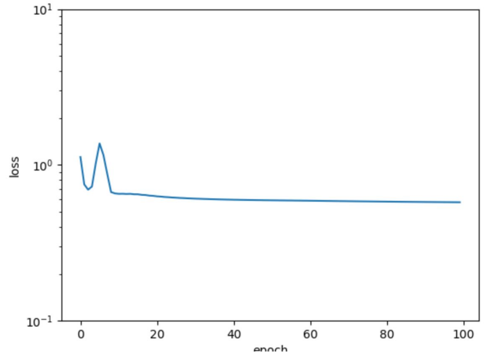
Feature variables – ['mass', 'C2','D2','e3', ‘Tau21’, 'Tau32_wta','Split12','Split23']

2 hidden linear relu-activated layers with 1024 nodes each

100 epochs

(roc curve on next slide)





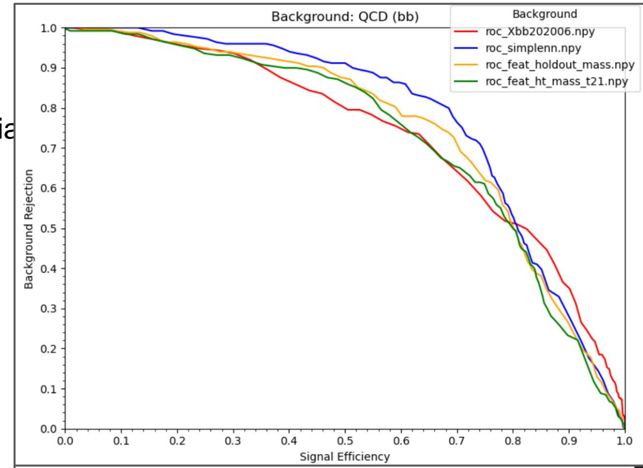
Feature Ranking: Mass

In order to observe the feature ranking, mass and Tau21 were excluded from the feature varia

Holdout_mass – mass excluded

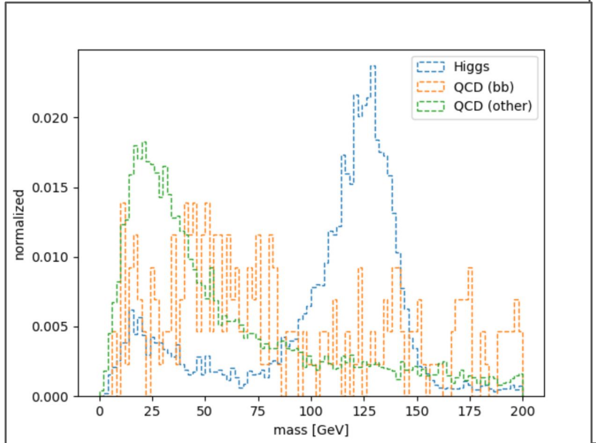
Ht_mass_t21 – mass and tau21 excluded

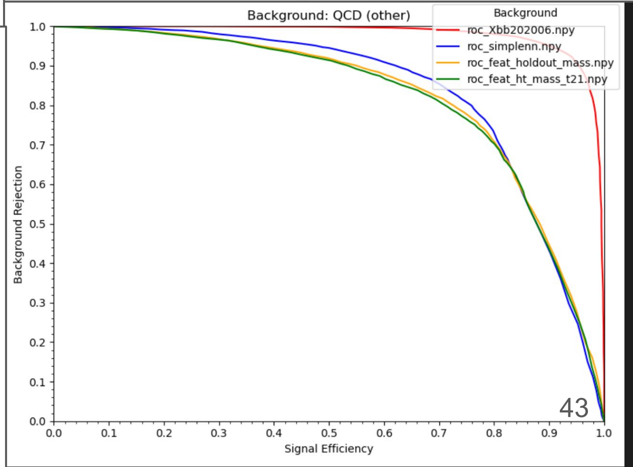
Simplenn - regular model (mass and tau21 included)



We excluded mass from the trainable jet features to gauge the networks dependence on mass for discrimination.

Results: We observed that excluding mass did not prove to have a major difference in model performance.





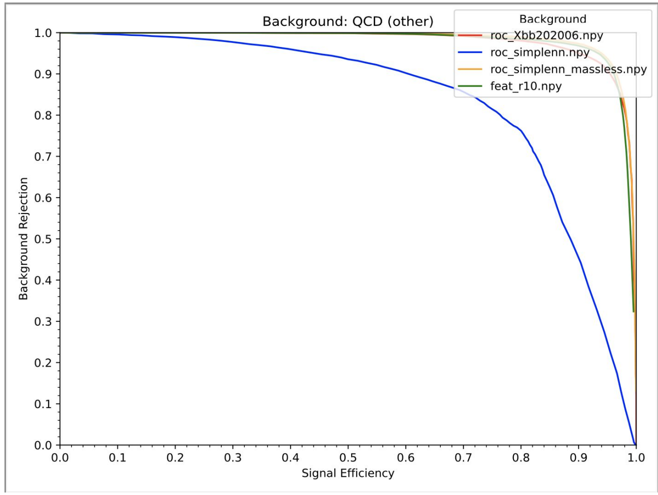
Using Xbb predictions as input features

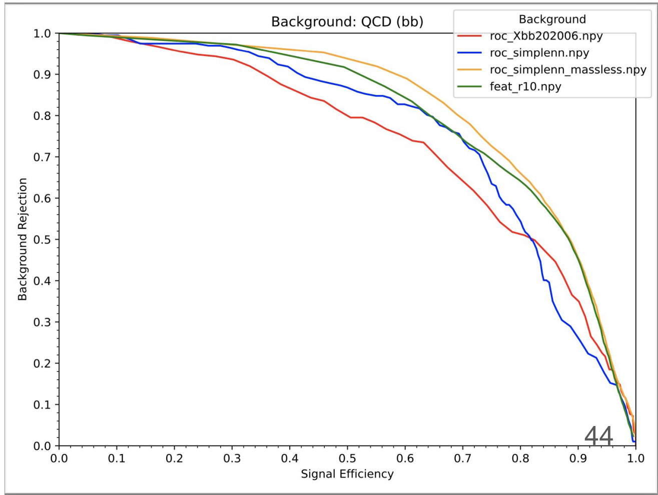
The 3 Xbb output predictions per jet were included as model features. Two iterations were conducted: with and without mass.

Simplenn - original NN
Feat_r10 - 10 epochs with mass feature included (Xbb included)
Simplenn_massless - 100 epochs without mass feature (Xbb included)

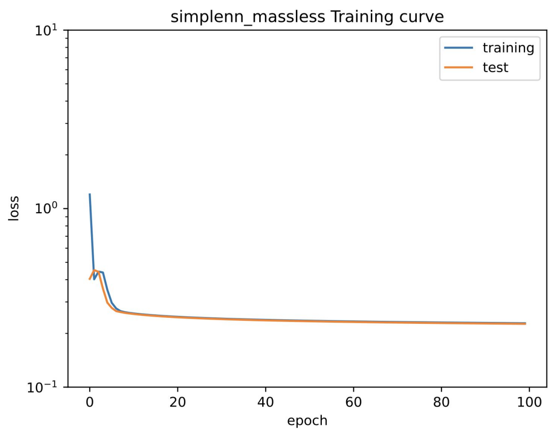
Results: Massless xbb-dependent model seems to perform better than mass-included xbb-dependent model,not to mention scope for improvement with a decreasing loss curve. Also, gbb rejection is significantly improved in the case of the massless model.

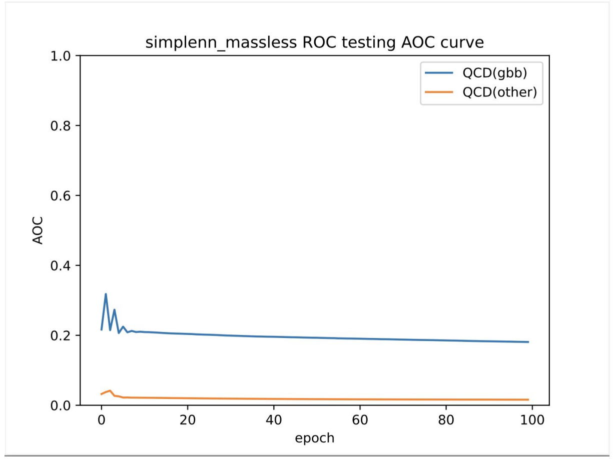
We observe an improvement in performance, revealing the potential for improvement for both gbb and misc background.

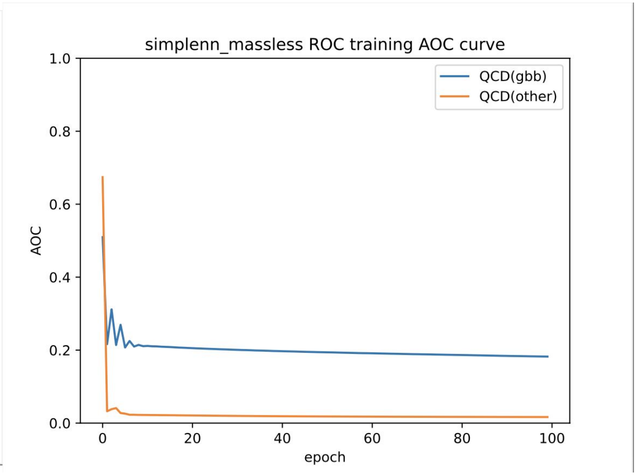




Training Information for Xbb-dependent, massless nn







Even after 100 epochs, the loss curve still continued to gradually decrease.

Convolutional Neural Networks (CNNs)

Simple CNN on Calorimeter images

Calorimeter heat maps were generated in terms of dphi and deta.

The below image shows the average heatmap post-engineering for each label in a total training dataset.

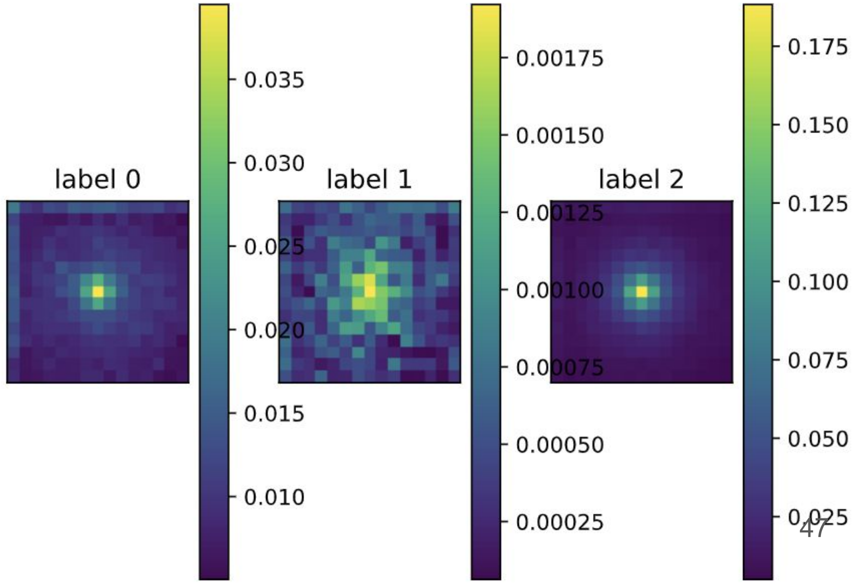
Label 0: gbb signal

Label 1: QCD (gbb)

Label 2: QCD (other)

Data Engineering was conducted via a log log method.

-> log(const. + nonnegative-rectified-log(x))

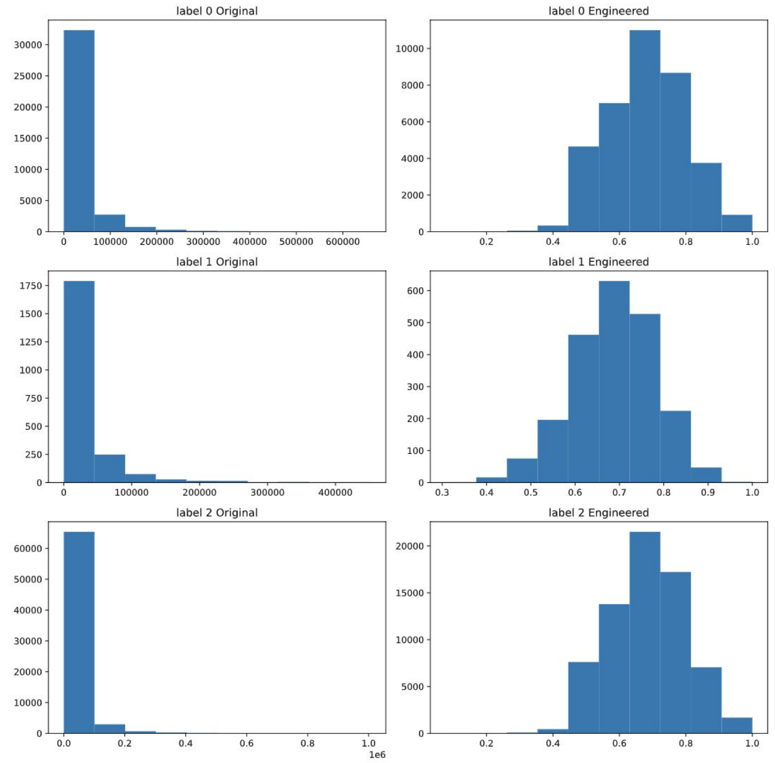


Calorimeter Image Feature Engineering

Various feature engineering methods were tested and applied exclusively on calorimeter images.

Current best displayed - rectified double log with <0 → 1 shift

The left graphs indicate the original distribution of all nonzero cell values in a batch/dataset, while the right is its engineered counterpart



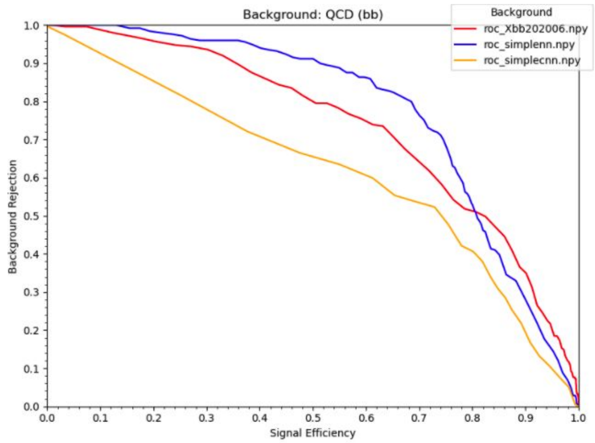
Training a CNN with 2 fully connected Layers

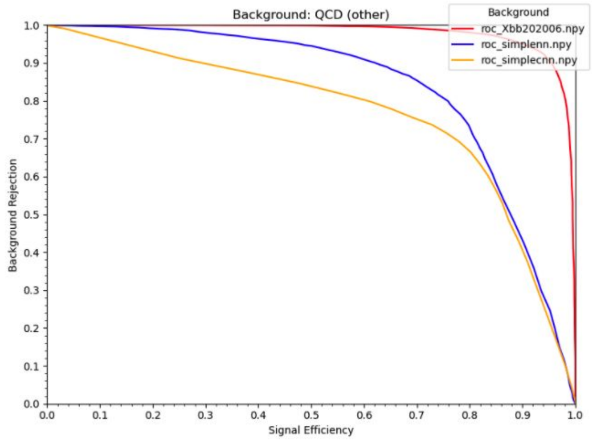
Layers -

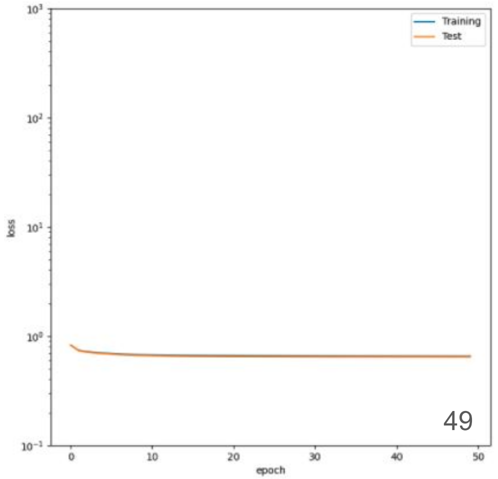
- Hidden1 - 10 output channels w/ kernel (3, 3), relu activation
- Hidden1 - 5 output channels w/ kernel (3, 3), relu activation

Run over 50 epochs

Best performance of a simple tensorflow cnn on heatmap images. Performance is worse that the feature nn maybe due to the sparsity of data in images







GNN on purely calorimeter data

Graph neural network applied on calorimeter data – acts as some form of upper bound for the cnn performance. Since the sparsity of data through image cells did not persist for graphs, I made each graph node be particular calorimeter constituent (individual particle constituent, not image cell).

– Input data is raw, pt ranges from [400, 120000]

– deta, dphi, m are unfiltered

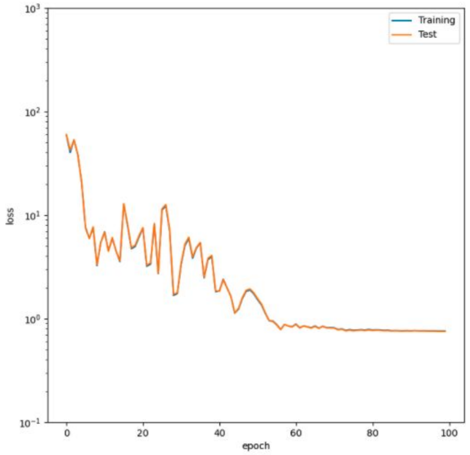
– ['pt', 'deta', 'dphi', 'm'] are the each node's (jet's) features

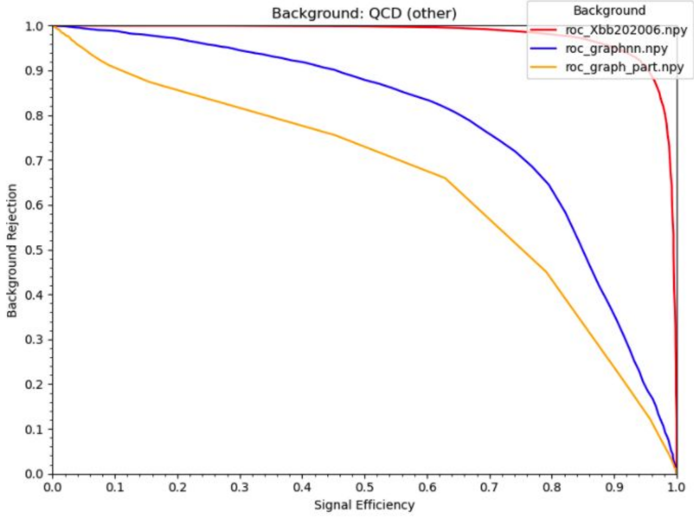
Graph_part: calo gnn in question

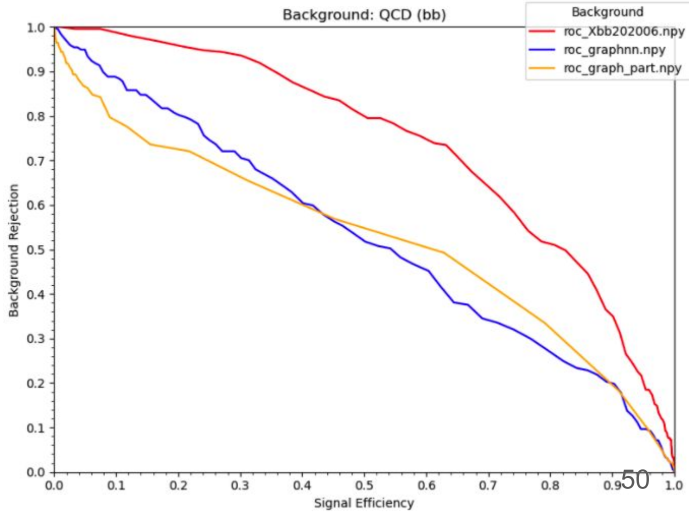
Graphnn: baseline graph nn on tracks

Results: While there was noticeable improvement in performance as compared to the cnn on calorimeter data (next slide), the loss curve indicates a limit to the performance that is significantly lower than Xbb as well as the baseline graphnn.

train/test overlap due to data/model infeasibility?


Training
Test
loss
epoch


Background: QCD (other)
Background
roc_Xbb202006.npy
roc_graphnn.npy
roc_graph_part.npy
Background Rejection
Signal Efficiency


Background: QCD (bb)
Background
roc_Xbb202006.npy
roc_graphnn.npy
roc_graph_part.npy
Background Rejection
Signal Efficiency