VBF H(cc) analysis

ANN and decorrelation strategy

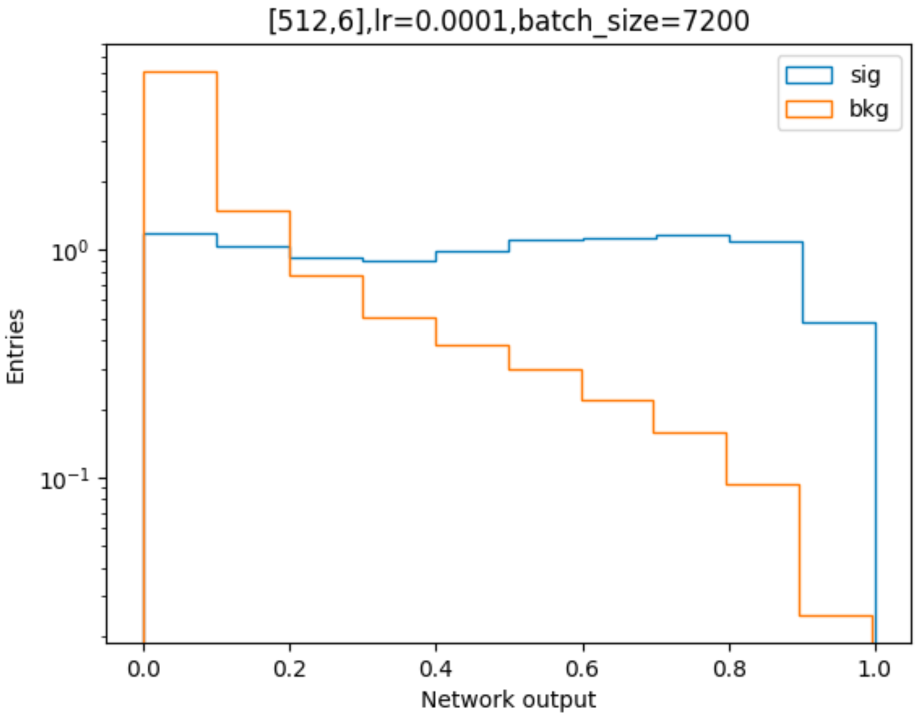
Greta Brianti, Roberto Iuppa, Marco Cristoforetti

?.01.2024

ATLAS EXPERIMENT
UNIVERSITÀ DI TRENTO
FONDAZIONE BRUNO KESSLER

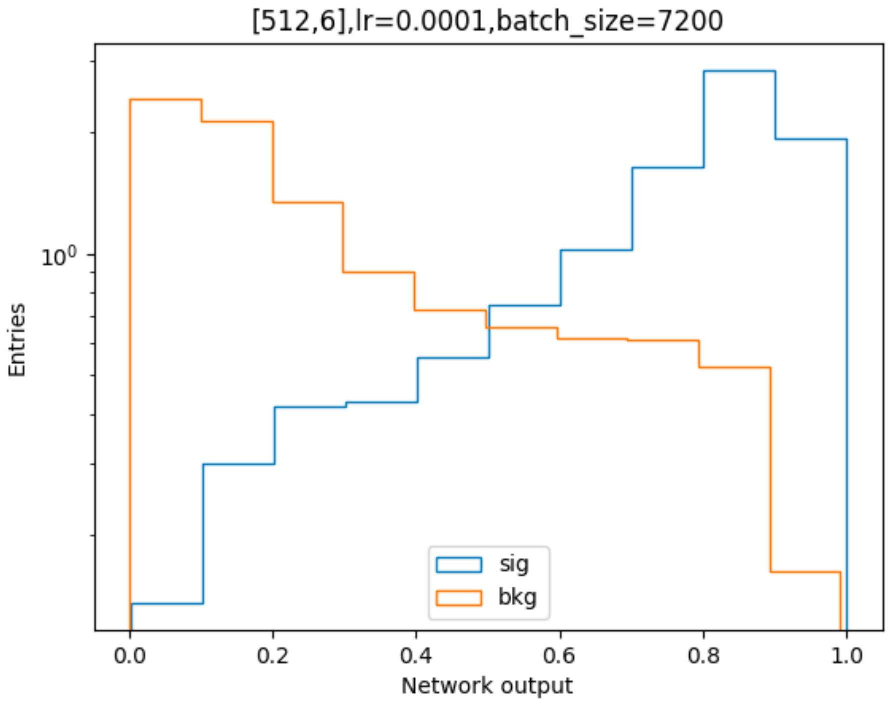
Huge unbalance between signal and background

Total number of train data:	102’618
Total number of validation data	102’938
Fraction of bkg data in training dataset:	0.791
Fraction of bkg data in validation dataset:	0.789
Fraction of sig data in training dataset:	0.209
Fraction of sig data in validation dataset:	0.211



Unbalanced data

Total number of train data:	42’878
Total number of validation data	102’938
Fraction of bkg data in training dataset:	0.500
Fraction of bkg data in validation dataset:	0.789
Fraction of sig data in training dataset:	0.500
Fraction of sig data in validation dataset:	0.211



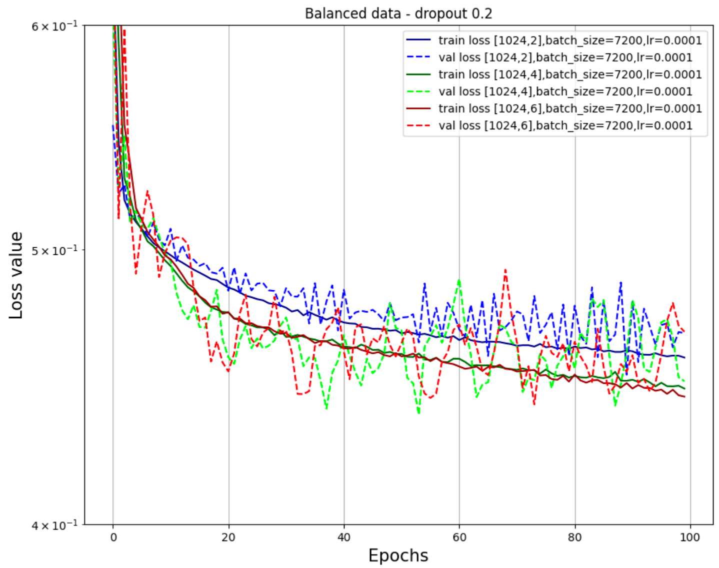
Balanced data on train dataset

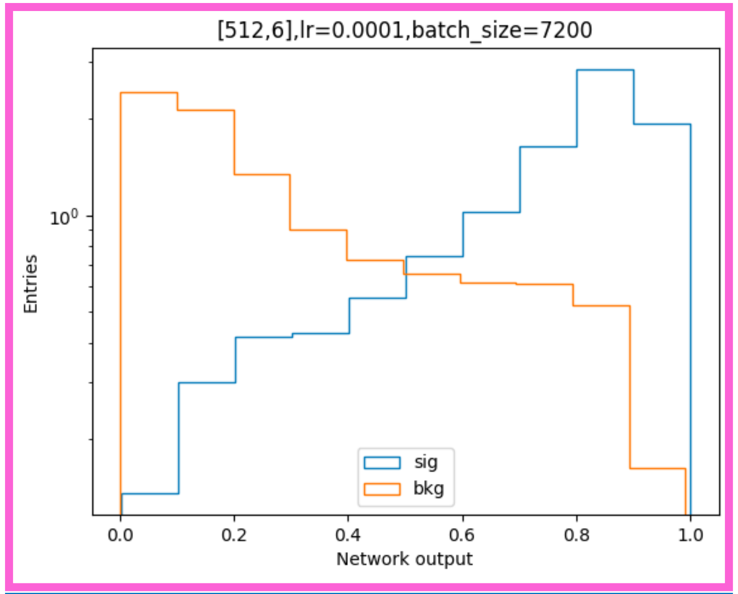
Pytorch network
- Sample: cc
- Epochs = 100
- Mod = 0
- Dropout = 0.2

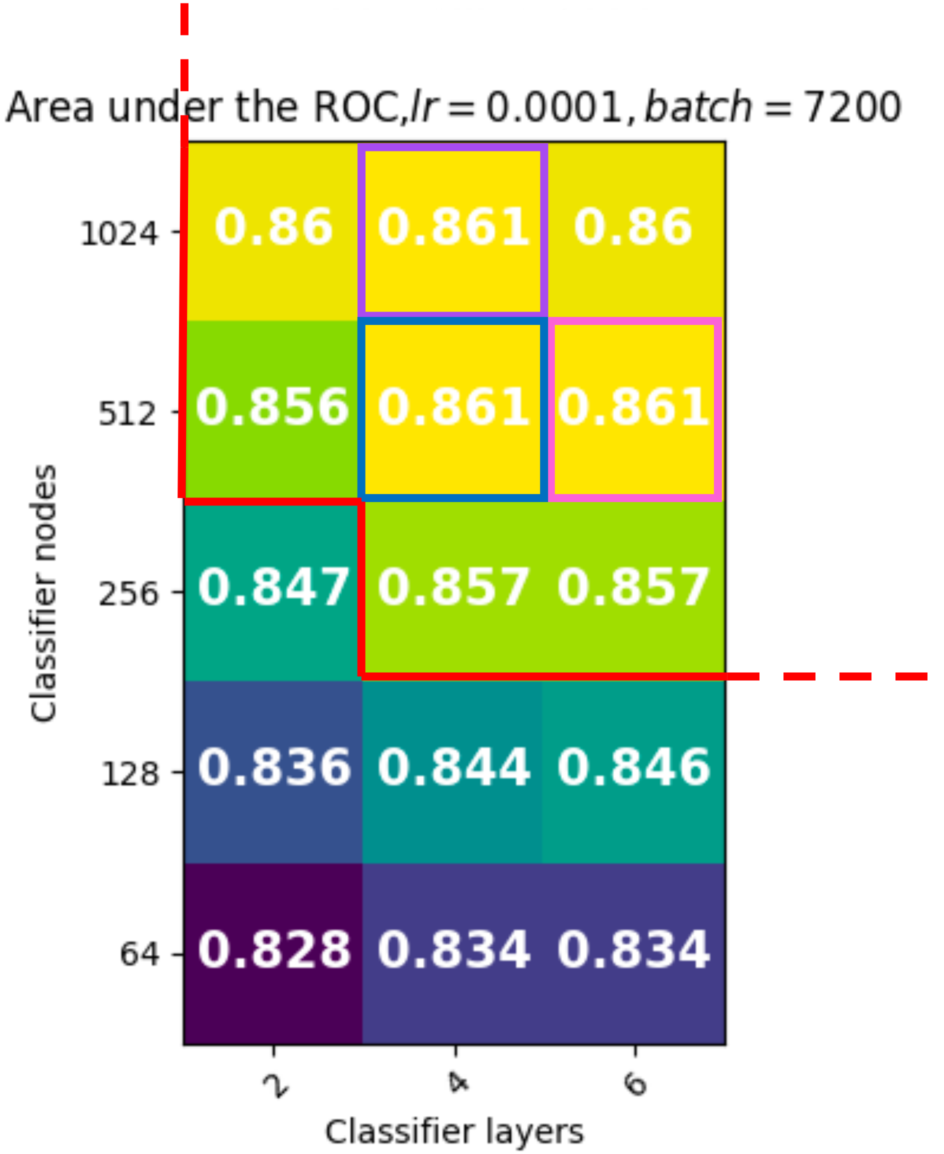
Downsampling of the training dataset.

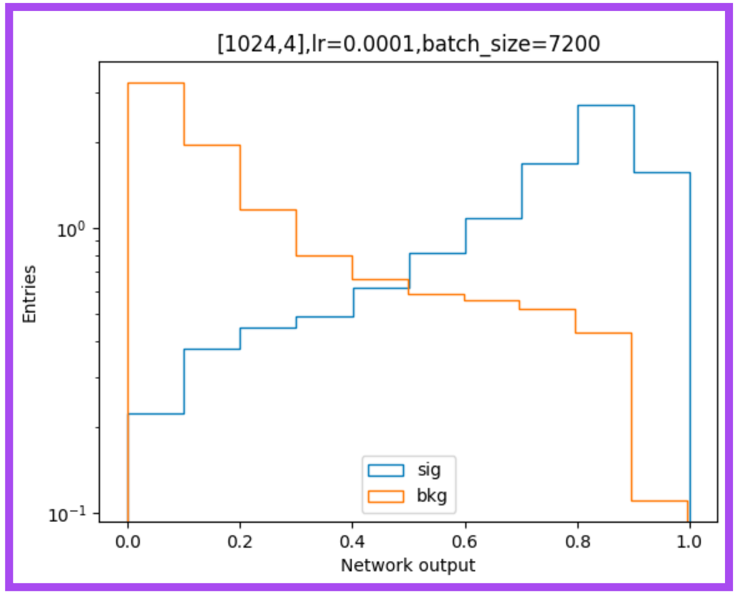
60% reduction of the training dataset.

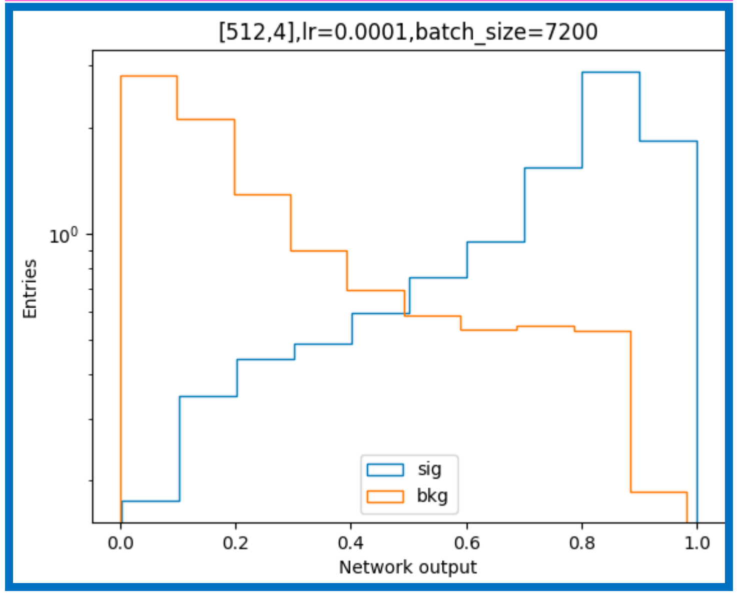
Balanced data results



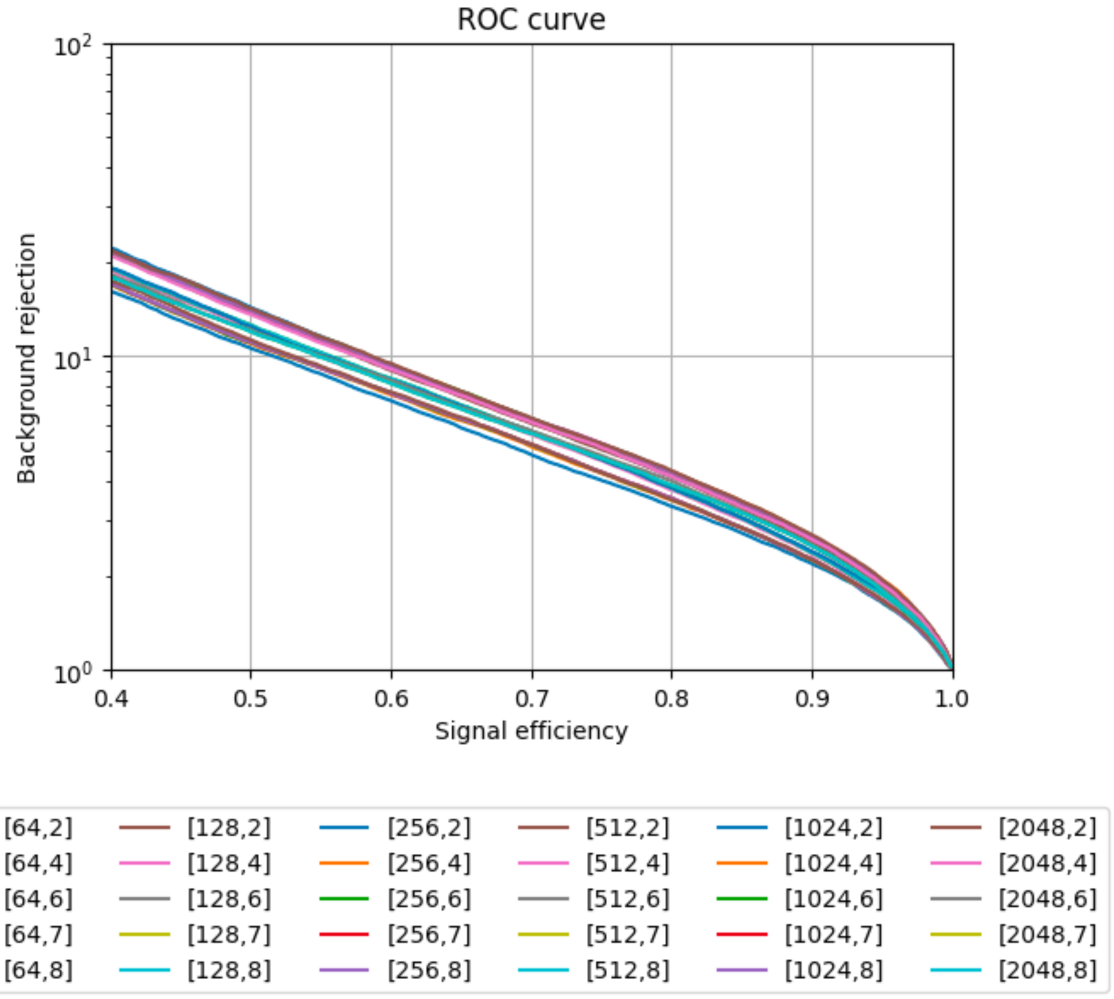


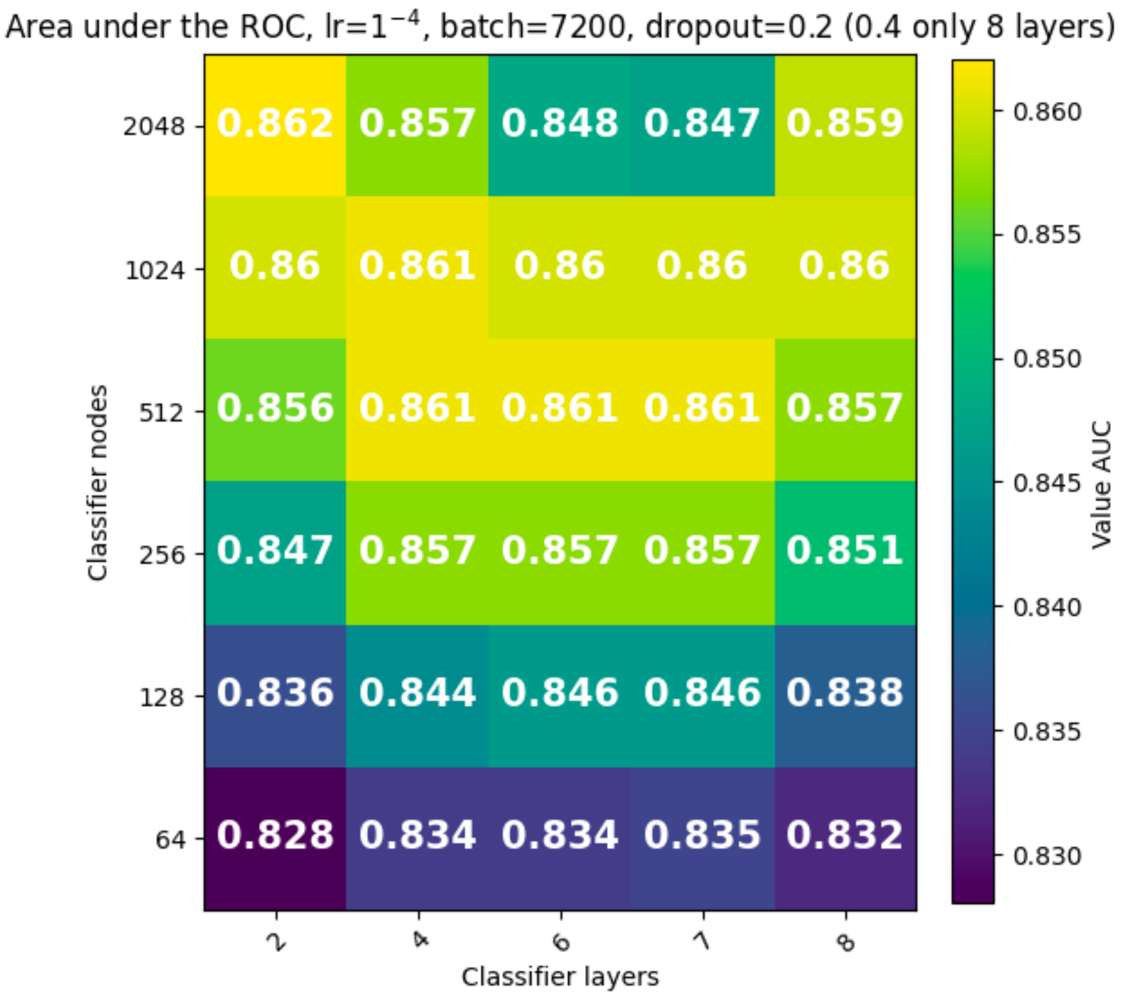






Extended results





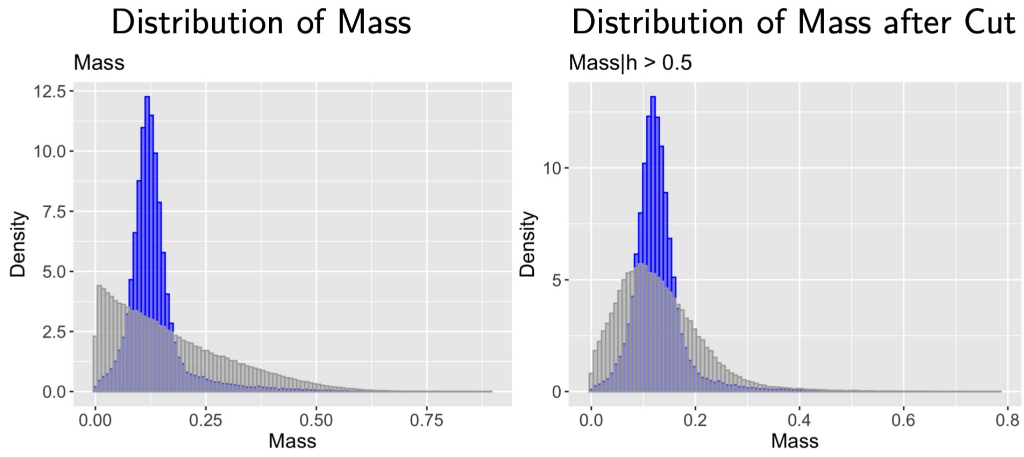
Conclusion

- Increase the number of events for training
- Overfitting under control with the dropout and large batch size
- Study on the batch size as Luke suggested

Decorrelation strategy

Correlation metrics

GOAL: measure how the cut on the network output affects the background distribution of c/b- tagged jets invariant mass



[1]

The Jensen-Shannon

$$JSD(P||Q) = \frac{1}{2}(KL(P||M) + KL(Q||M))$$ [2]

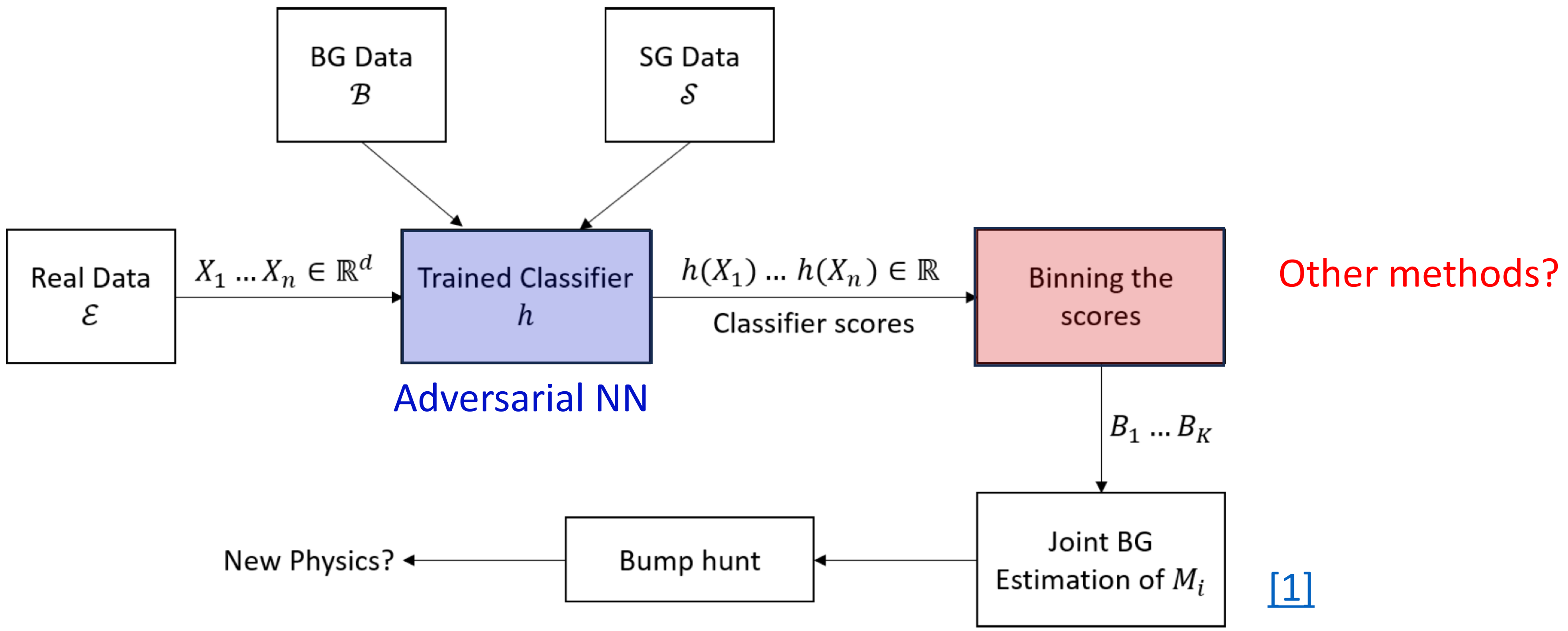
with $M = \frac{P+Q}{2}$ and KL the Kullback-Leibler divergence

In our case, the divergence is used to measure the difference between the normalised mass distributions of the background jets passing and failing, respectively, a given jet tagger cut:

$$JSD(P||Q) = JSD\left(\frac{N_{bkg}^{pass}(m)}{\sum_i N_{bkg}^{pass}} || \frac{N_{bkg}^{fail}(m)}{\sum_i N_{bkg}^{fail}}\right)$$

Decorrelation methods

GOAL: use a decorrelation method that doesn't affect the network classification performance



Make cuts on transformed classifier output $T(h(X))$, where $T(h(X))$ is independent of the protected variable $m_{bb/cc}$ for background data.

Optimal Transport (OT) [3]

GOAL: learn a monotonic transformation $\mathrm{T}(\cdot\,|c)$ between the input feature space $Q(\cdot\,|c)$ and a target feature space $\mathrm{P}(\cdot)$

A solution to this problem is the optimal transport map that gives us:

$$\nabla_x g(Q;\theta) = P \quad \& \quad \nabla_x f(P;\theta) = Q$$

Where θ represents the trainable parameters of the network and f, g are two convex functions.

The problem is solved by finding f, g → Input Convex Neural Networks [4]

The ICNN gives the transport function $\nabla_x f(x, c; \theta)$ with $f(x, c; \theta)$ a convex function in x but not in c.

Benefits:

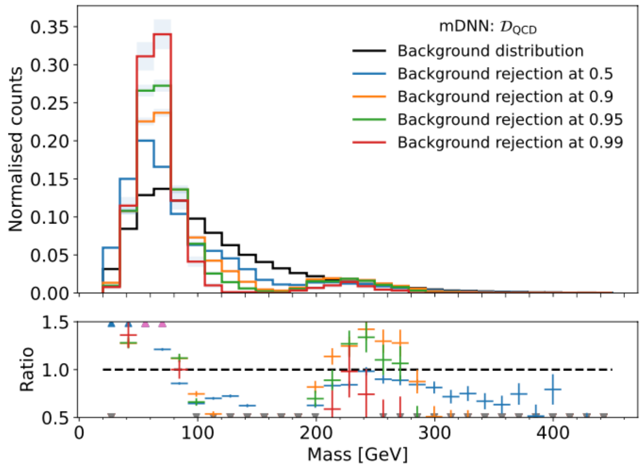
1. We make the output of the classifier INDEPENDENT from the invariant mass.
2. We do not affect the classifier's performance.

Results [3]

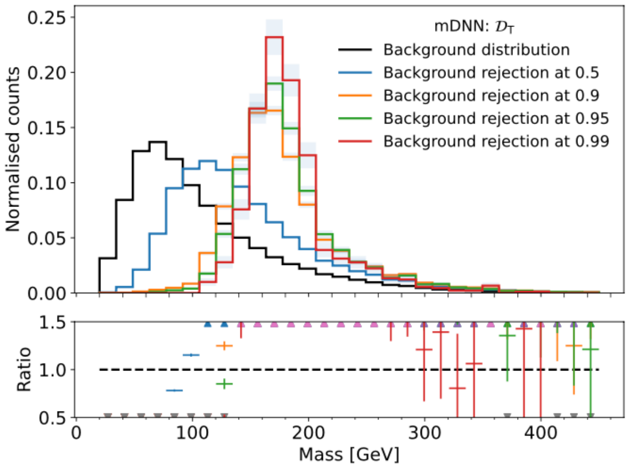
$$D_{QCD} = \frac{p_{QCD}}{p_T + p_{VB}}$$

$$D_T = \frac{p_T}{p_{QCD} + p_{VB}}$$

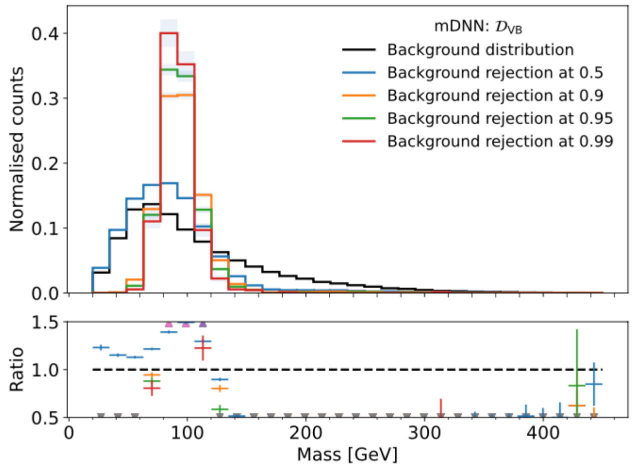
$$D_{VB} = \frac{p_{VB}}{p_{QCD} + p_T}$$



(a) $\mathcal{D}_{\mathrm{QCD}}$ projection



(b) $\mathcal{D}_{\mathrm{Top}}$ projection



(c) $\mathcal{D}_{\mathrm{VB}}$ projection

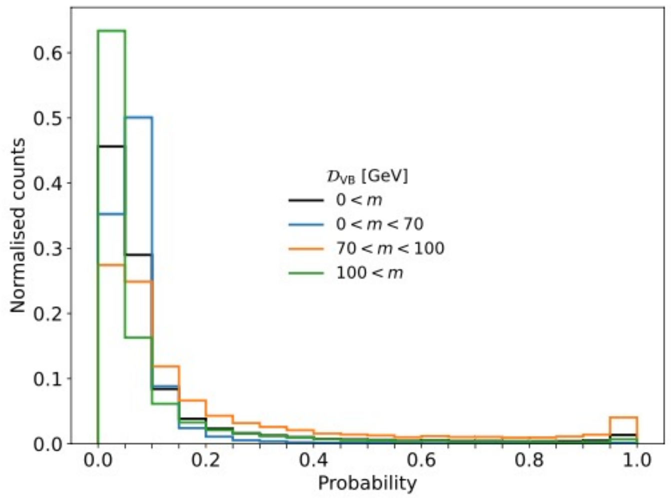
Jet generated from quark/gluon of QCD background

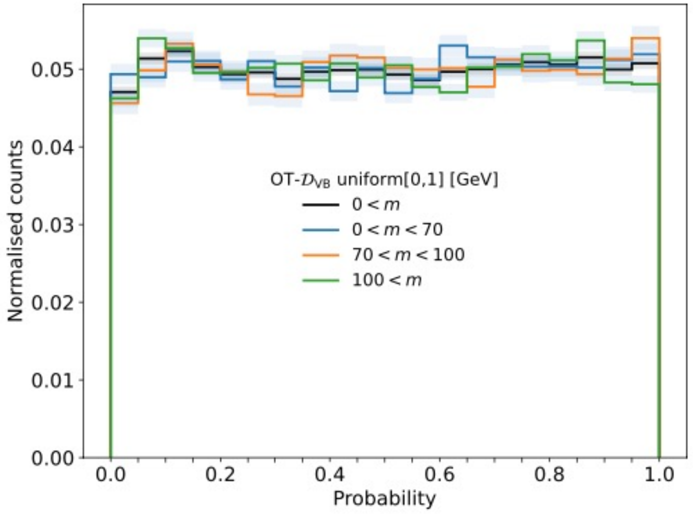
Jet generated from top quarks

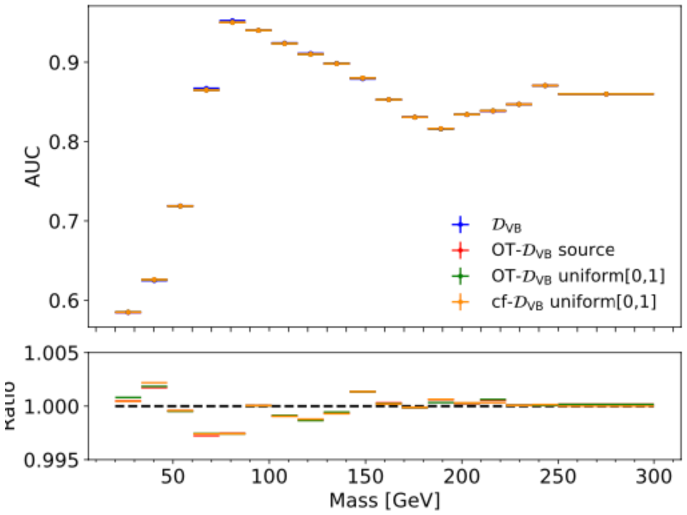
Jet generated from vector bosons

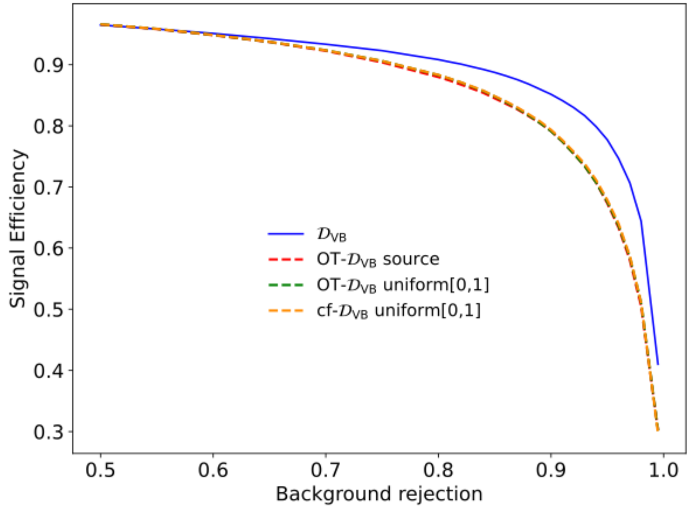
Classifier network

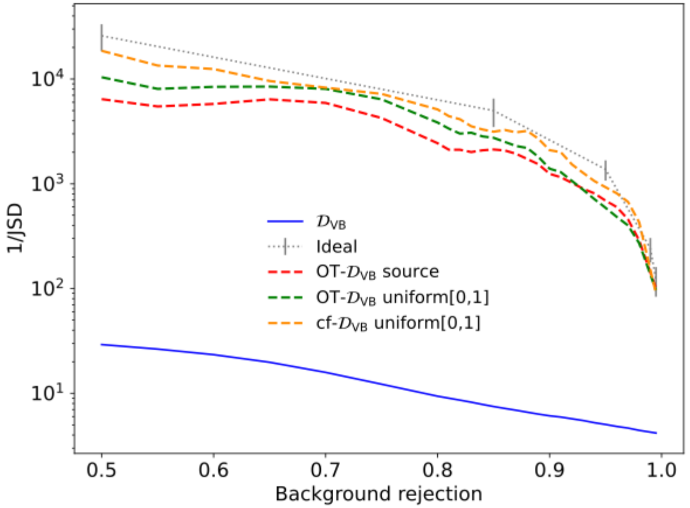
Decorrelation [3]











Conclusion

Backup

Features description

Input features (12)	Description
m_{jj}	Invariant mass of the VBF jet pair
$p_{T,jj}$	Transverse momentum of the VBF jet pair
$p_T^{balance}$	Ratio of the vectorial and scalar sums of the transverse momenta of c_1 , c_2, j_1 and j_2.
$(p_T^{j_1} - p_T^{j_2})\ (p_T^{j_1} + p_T^{j_2})$	Asymmetry in the VBF jet transverse momenta
$\Delta\eta(cc, jj)$	Separation in η between the c-tagged jet pair and the VBF jet pair
$\Delta\phi(cc, jj)$	the separation in ϕ between the c-tagged jet pair and the VBF jet pair
$\tan^{-1}\left(\tan\left(\frac{\Delta\phi(cc)}{2}\right) / \tanh\left(\frac{\Delta\eta(cc)}{2}\right)\right)$	the measure of the relative angle of η and ϕ between the two c-tagged jets.
n_{jets}	the number of jets with p_T > 20 GeV and $\|\eta\|$ < 4.5
$\min \Delta R(j_{1(2)})$	the minimum separation in R between the (sub)leading VBF jet and any jet in the event that is not a part of the b-tagged jet pair or VBF jet pair
$N_{trk}^{j_{1(2)}}$	the number of tracks matched to the (sub)leading VBF jet.

Pytorch Network for Classification task

```python
## Dataset preprocessing
batch_size = int(batch)
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


traindataset_cl = myDataset(X_train,y_train)
testdataset_cl = myDataset(X_test,y_test)


train_loader_cl = torch.utils.data.DataLoader(dataset=traindataset_cl,
                                              batch_size=batch_size,
                                              shuffle=True)
test_loader_cl= torch.utils.data.DataLoader(dataset=testdataset_cl,
                                            batch_size=batch_size,
                                            shuffle=True)
```

Scaling on the whole dataset *(TensorFlow: scaling on the batch with BatchNormalization layer)*

$$X' = \frac{X - mean}{\sigma}$$

$mean$: mean of the training sample → center the data before scaling
σ: unit standard deviation
[sklearn.preprocessing.StandarScaler]()

```python
# Neural Network with customizable number of layers and nodes

class NeuralNet(nn.Module):
    '''
    A DL model with customizable layers and nodes.
    '''
    def __init__(self, input_size, layers):
        super(NeuralNet, self).__init__()
        self.layers = nn.ModuleList()
        last_size = input_size
        for layer_size in layers:
            self.layers.append(nn.Linear(last_size, layer_size))
            last_size = layer_size
        self.layers.append(nn.Linear(last_size, 1))
        self.relu = nn.ReLU()

    def forward(self, x):
        for layer in self.layers[:-1]:
            x = layer(x)
            x = self.relu(x)
        x = self.layers[-1](x)
        return x
```

`Input: Number of features. nodes per layer`
`Output: Single output for binary classification`

```python
model = NeuralNet(len(X_train[0]), [256,256,256])
model
```
✓ 0.0s

```
NeuralNet(
  (layers): ModuleList(
    (0): Linear(in_features=12, out_features=256, bias=True)
    (1-2): 2 x Linear(in_features=256, out_features=256, bias=True)
    (3): Linear(in_features=256, out_features=1, bias=True)
  )
  (relu): ReLU()
)
```

Loss function
Extended results