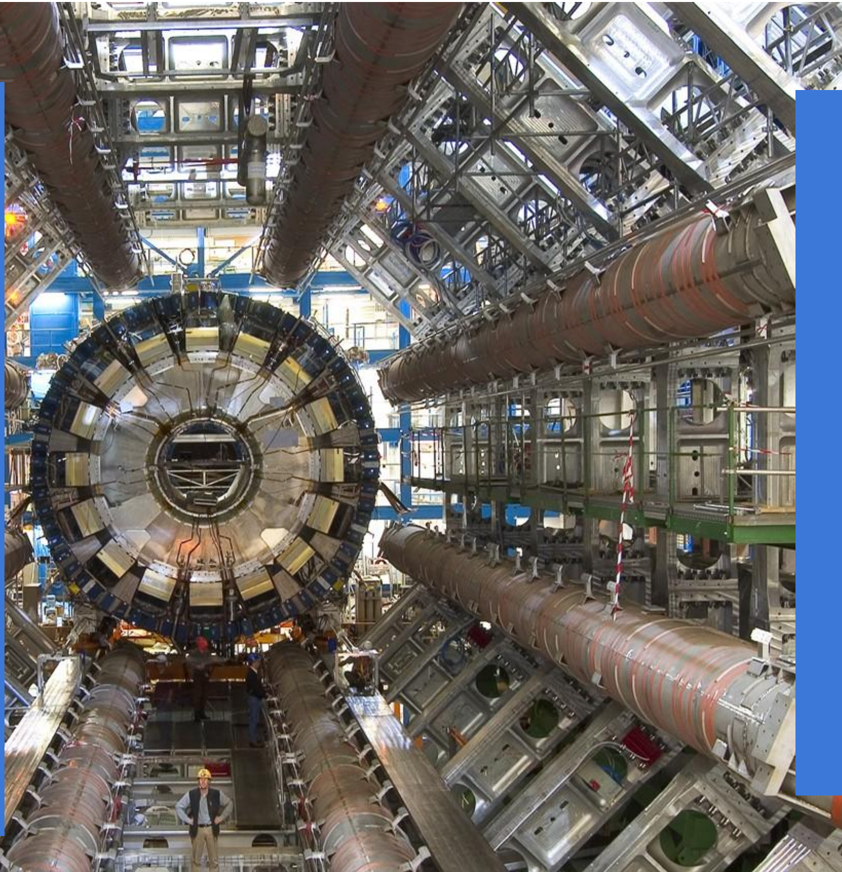
LABORATÓRIO DE INSTRUMENTAÇÃO
E FÍSICA EXPERIMENTAL DE PARTÍCULAS
partículas e tecnologia

DNN UNCERTAINTIES IN VLQ SEARCH AT LHC

Gilberto Cunha
contact: gcacademic@outlook.pt



METHODOLOGY

01

DATA PRE-PROCESSING
Clean data and apply cuts

02

CLASSIFY EVENTS
Deep Neural Network: Signal vs Background

03

ANALIZE DNN PREDICTION UNCERTAINTIES
Monte Carlo Dropout

VLQ SIGNAL

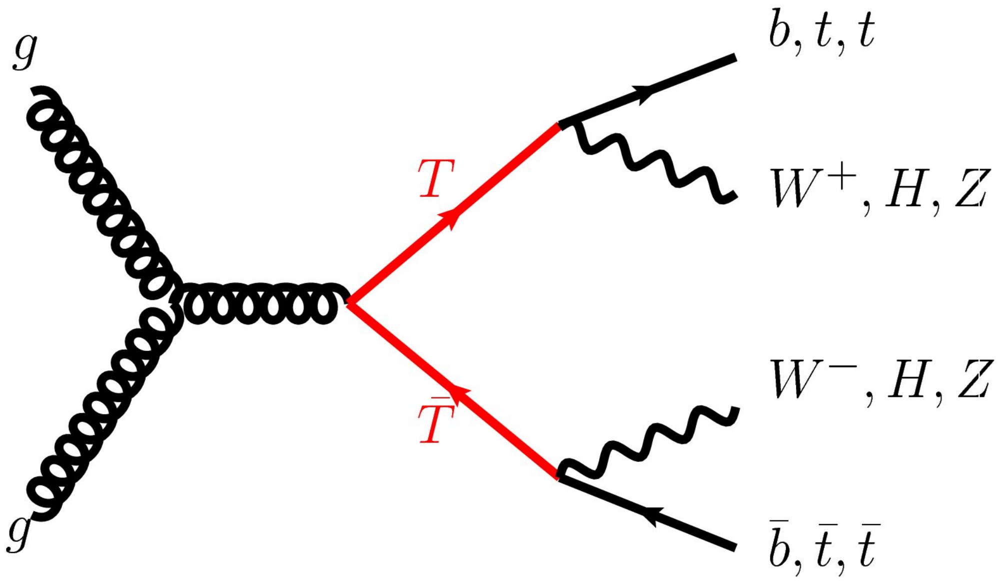


Fig. 2: VLQ general Feynman diagram

- Background data is dileptonic
- Focus on T to tZ decays to capture the dileptonic part of VLQ signal

DATA STRUCTURE

	Electron1_Eta	Electron1_PT	Electron1_Phi	Electron2_Eta	Electron2_PT	Electron2_Phi	Electron_Multi	FatJet1_Eta
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0	0.482720
1	-2.060421	30.932735	-1.365277	0.000000	0.000000	0.000000	1	0.000000
2	-1.025947	40.282574	-1.773086	0.288352	26.201660	-0.694144	2	0.000000
3	1.084838	82.556099	2.932473	0.000000	0.000000	0.000000	1	0.969367
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0	0.000000
...	...	...	...	...	...	...	...	...
49981	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0	0.856027
49987	0.803573	115.304886	-2.760615	0.394527	63.806351	2.506781	2	-1.067106
49988	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0	-0.674905
49992	0.311730	141.319260	2.593879	0.543723	120.261703	1.999698	3	0.436267
49999	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0	0.369915

Fig. 1: Pandas dataframe of the data

- Tabular
- Generated
- Experimental and generated features

PRE-PROCESSING

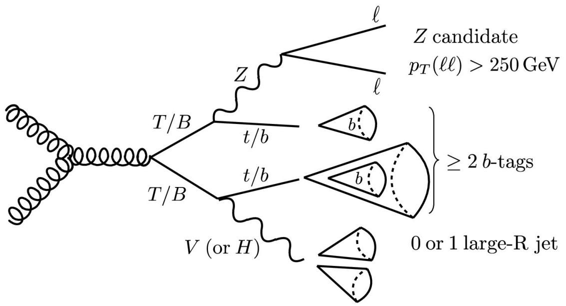


Fig. 3: VLQ Feynman Diagram for cuts

$$w_{i,s} = \frac{\sigma_i}{N_s}$$

Eq. 1: Gen weights computation

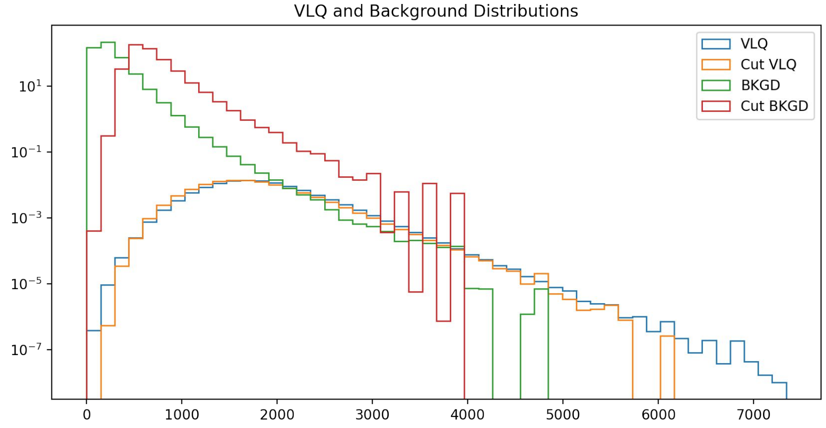


Fig. 4: VLQ and BKGD total transverse energy distributions

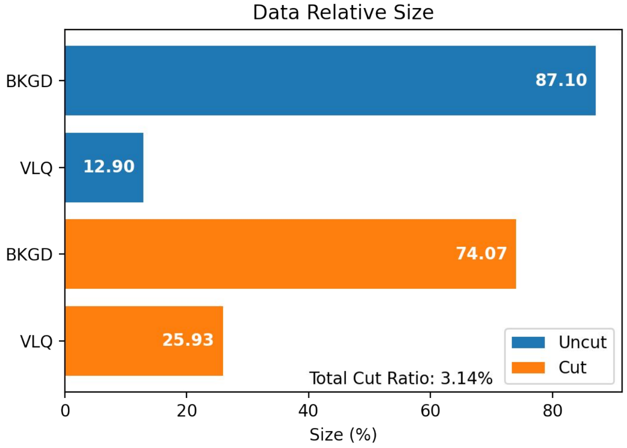


Fig. 5: Class size before and after cuts

1. Apply cuts: >= 2L and >= 1 Fat Jet
2. Calculate gen weights
3. Concatenate all samples

DATA DISTRIBUTIONS

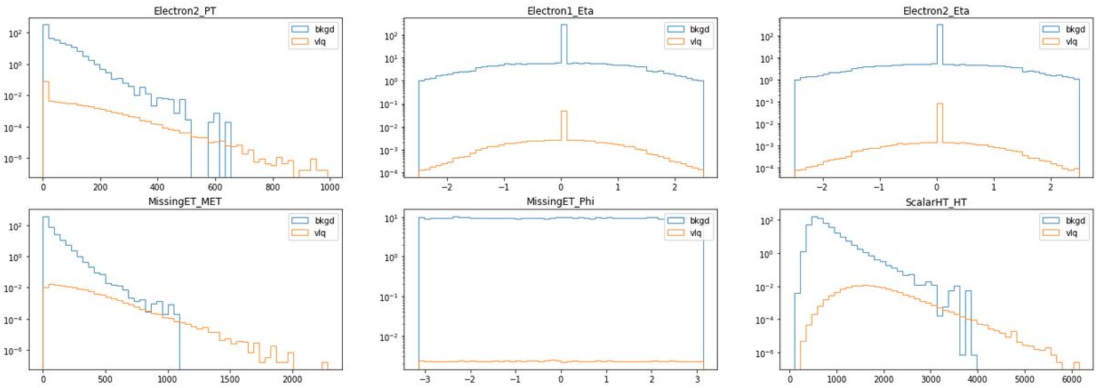


Fig. 6: Pre-processed data feature distributions

- Weighted distributions -> Physical distributions
- Capture the physical differences between signal and background in the data
- These differences will allow the model to separate the two

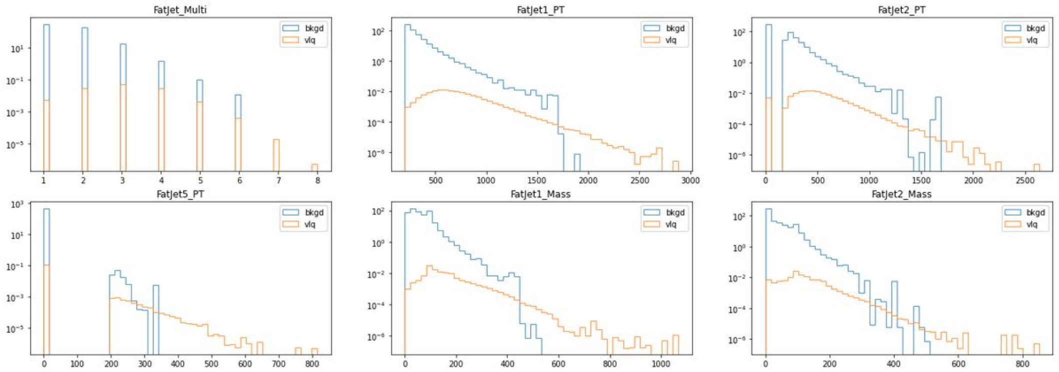


Fig. 7: More data feature distributions

THE MODEL

```
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 69)]              0
_________________________________________________________________
batch_normalization_15 (Batc (None, 69)                276
_________________________________________________________________
dense_36 (Dense)             (None, 84)                5880
_________________________________________________________________
dropout_16 (Dropout)         (None, 84)                0
_________________________________________________________________
dense_37 (Dense)             (None, 49)                4165
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 50
=================================================================
```

Fig. 6: Model summary

- 69 input neurons
- Batch Normalization after input layer
- Hidden layers w/ relu activation
- Dropout layer on top of hidden layers
- 1 output neuron w/ sigmoid activation

```python
def get_model(hidden_layers=[100, 100, 100], dropout=0.1, batch_norm=True, optimizer="Nadam", summary=True):
    """
    This function creates a keras model, given the desired hidden_layers, dropout rate
    and optimizer of choice

    hidden_layers -> [int]: size of each desired hidden layer
    dropout -> float: desired dropout rate
    optimizer -> string: optimizer you choose to utilize

    returns a keras model
    """

    # Generate model structure
    inputs = keras.Input(shape=(69,))
    bn = keras.layers.BatchNormalization()(inputs)
    drop = bn
    for i in range(len(hidden_layers)-1):
        fc = keras.layers.Dense(hidden_layers[i], activation='relu')(drop)
        if batch_norm:
            bn = keras.layers.BatchNormalization()(fc)
        else:
            bn = fc
        drop = keras.layers.Dropout(dropout)(bn, training=True)
    fc = keras.layers.Dense(hidden_layers[-1], activation='relu')(drop)
    outputs = keras.layers.Dense(1, activation='sigmoid')(fc)

    # Instanciate and compile model
    model = keras.Model(inputs, outputs)
    model.compile(optimizer=optimizer, loss="binary_crossentropy",
                  metrics=["accuracy", keras.metrics.AUC()])
    if summary: model.summary()

    return model
```

TRAINING

- Unbalanced classes -> class weights
- Weighted data -> gen weights
- Train, Validation and Test equal split

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{n=0}^{1} y_{i,n}\log(\hat{y_{i,n}})$$

Eq. 2: Binary Cross-Entropy Loss

$$c_n = \frac{N_{bkgd}}{N_n}$$

Eq. 3: Class weights computation

$$\tilde{\omega_i} = \frac{\omega_i}{\sum_{i=1}^{N}\omega_i}$$

Eq. 4: Normalized gen weights

$$L = -\sum_{i=1}^{N}\sum_{n=0}^{1} c_n\,\tilde{\omega_i}\,y_{i,n}\log(\hat{y_{i,n}})$$

Eq. 5: Weighted Binary Cross-Entropy Loss

HYPERPARAMETER TUNING

- Optimize model by tuning variable parameters
- Next parameters chosen by Bayesian Inference

```python
# Defining parameters
num_layers = trial.suggest_int("num_hidden_layers", 1, 4)
hidden_layers = []
for i in range(num_layers):
    num_features = trial.suggest_int(f"num_features_layer_{i}", 20, 150)
    hidden_layers.append(num_features)
dropout = trial.suggest_discrete_uniform("dropout", 0.05, 0.4, 0.01)
batch_size = trial.suggest_categorical("batch_size", [128, 256, 512])
batch_norm = trial.suggest_categorical("batch_norm", [True, False])
optimizer = "Adam"
es_patience = 10
```

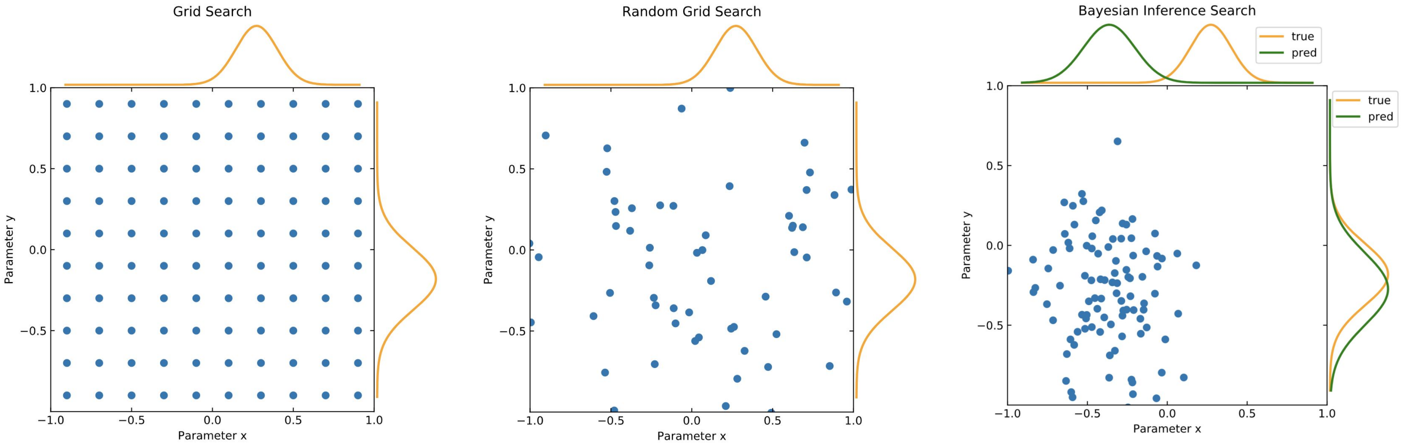


Fig. 7: Hyperparameter search method comparison

MONTE CARLO DROPOUT

```
num_models = 100
mcpreds = []

for _ in tqdm(range(num_models), total=num_models, desc="MCDropout"):
    mcpreds.append(model.predict(X_val))

mcpreds = np.array(mcpreds)
```

```
MCDropout: 100%|██████████| 100/100 [03:55<00:00,  2.35s/it]
```

```
mc_means = mcpreds.mean(axis=0)
mc_stds = mcpreds.std(axis=0)
```

- Dropout randomly zeros weights during forward pass
- Use dropout during predictions
- Make many predictions using the same model
- Take the mean as your final prediction
- Analyze predictions' standard deviation as a proxy for model prediction uncertainty

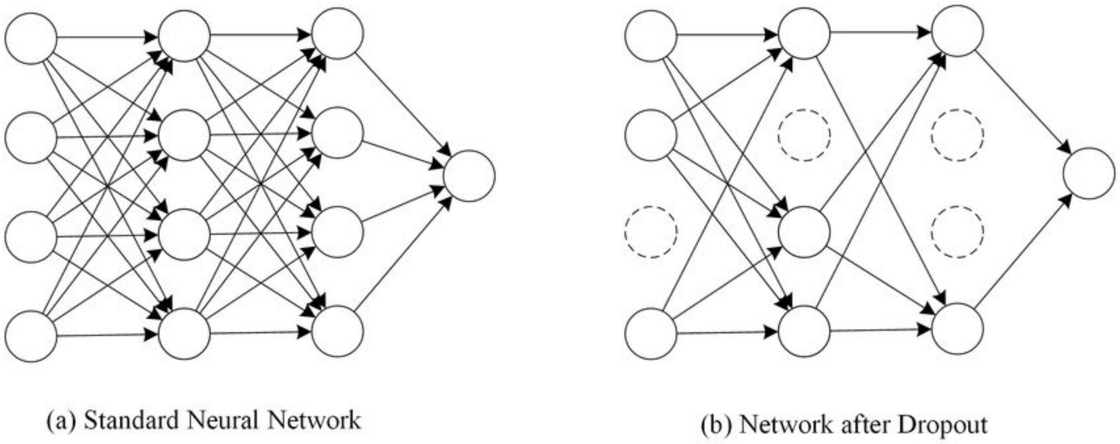


Fig. 8: Dropout Representation

RESULTS

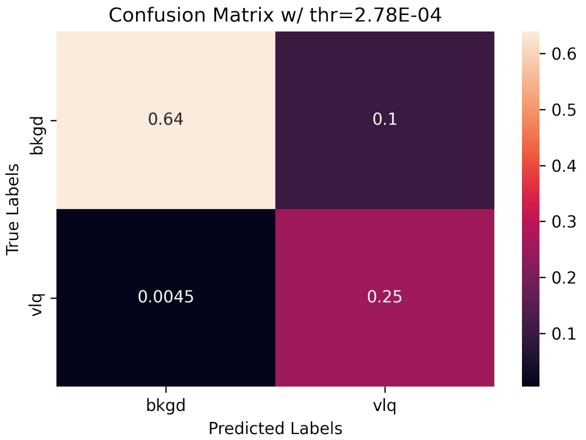


Fig. 9: MC Dropout Confusion Matrix

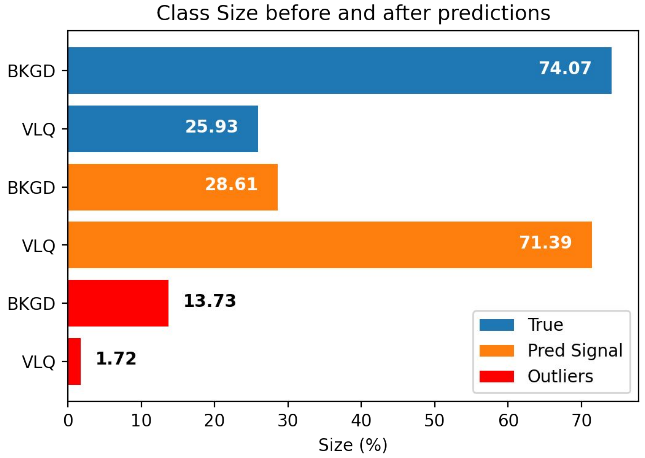


Fig. 10: Background/Signal ratio reduction after predictions

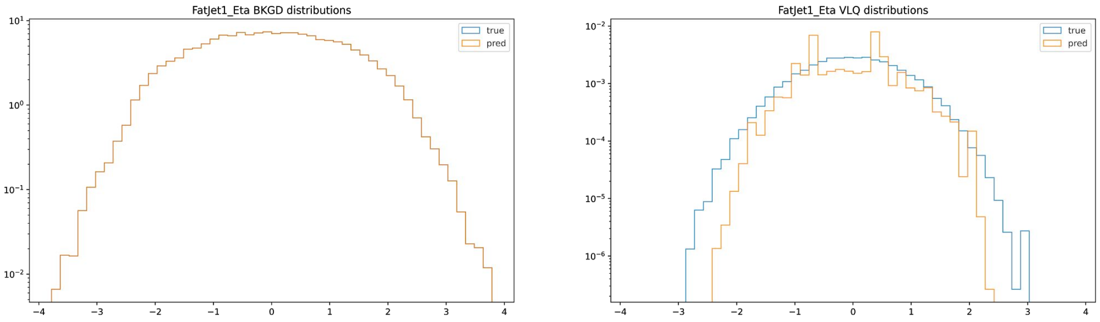


Fig. 11: MCDropout predictions w/ thr=0.5

Model	ROC AUC
Regular	0.99673
MC Dropout	0.99692

Table 1: Model ROC AUCs

RESULTS

- VLQ is dominant in high prediction uncertainties
- All VLQ samples have similar uncertainty distributions
- Only some BKGD samples have high uncertainty
- High uncertainty -> Mixing similar VLQ and BKGD

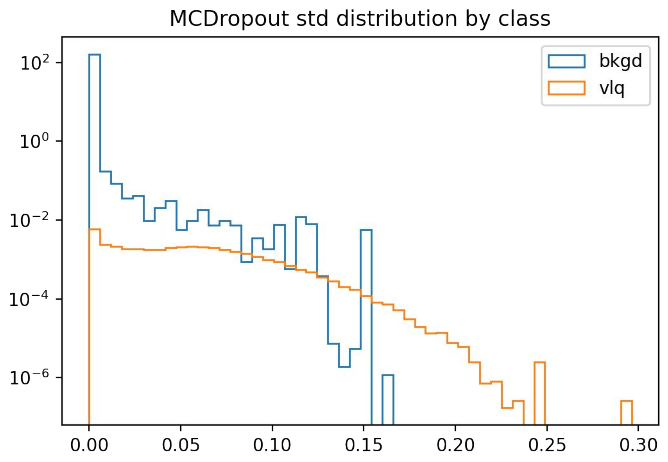


Fig. 12: Std deviation distributions per class

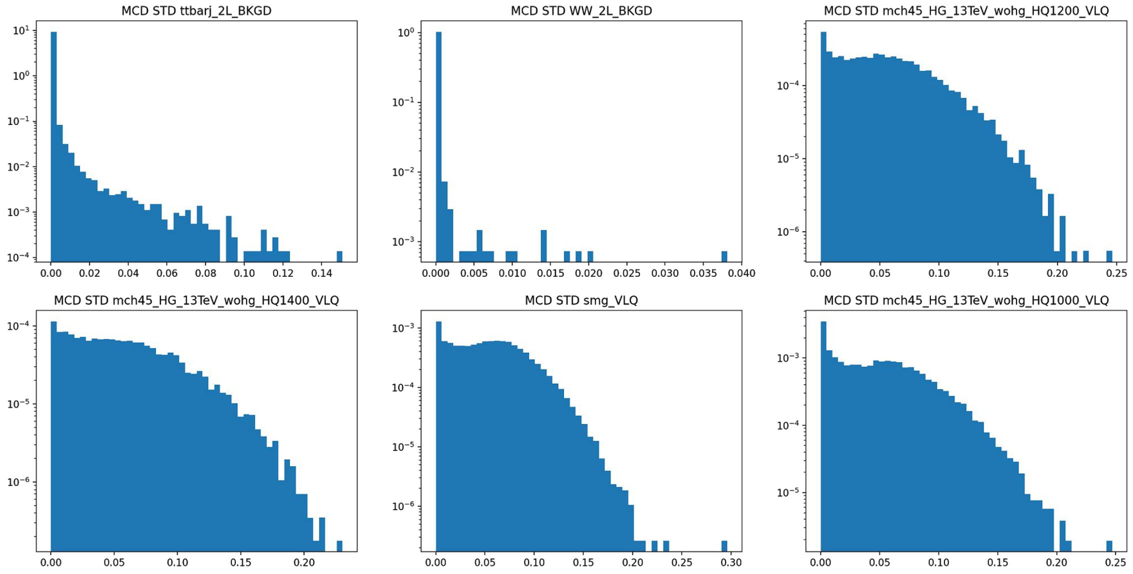


Fig. 13: Std deviation distributions per data sample

CONCLUSIONS

01

DNNs showed good results in reducing background to signal ratio

02

MC Dropout didn’t significantly improve the model

03

High prediction uncertainties arise from similarities in class distributions