Xudong 2021 Summer Research Placement Summary

Technologies

- Python 3 standard libraries: numpy, pandas, sklearn, matplotlib
- Machine learning libraries: tensorflow and xgboost
- Also used: shap (https://github.com/slundberg/shap)
  uproot (https://uproot.readthedocs.io/en/latest/)
  hh4b_utils (https://gitlab.cern.ch/hh4b/hh4b-python-utils)
  mplhep (https://github.com/scikit-hep/mplhep)

Files used

- mc files from /eos/user/l/lborgna/bbbb/NNT/cryptotuples-MAY21/mc/sm_hh/

  mm_hh_pythia_mc16a.root, mm_hh_pythia_mc16d.root,
  mm_hh_pythia_mc16e.root

  masks: pass_vbf_sel==False, X_wt_tag>=1.5, ntag>=4

- Data files from /eos/user/s/sgasioro/public/ggF_push/quad_yrOHE_bkt_Xhh/Xhh_45/

  data16_Xhh_45_NN_100_bootstraps.root,
  data17_Xhh_45_NN_100_bootstraps.root,
  data18_Xhh_45_NN_100_bootstraps.root

  masks: pass_vbf_sel==False, X_wt_tag>=1.5, ntag==2, rw_to_4b==True

Some initial results

At the beginning, the more features, the higher the AUC (and accuracy)

- 'X_hh','dEta_hh','njets','m_hh_cor' ,'X_wt_tag', 'eta_i','year’

  AUC ~ 0.70-0.78

- 'm_hh','X_hh','dEta_hh','njets','X_wt_tag','cos_theta_star','X_wt_notag','bkt_HT','pt_hh','pT_2','pT_4','eta_i','dRjj_1','dRjj_2','year’

  AUC ~ 0.85-0.90

- Basically all features (55)

  AUC ~ 0.95-0.96

Some initial results

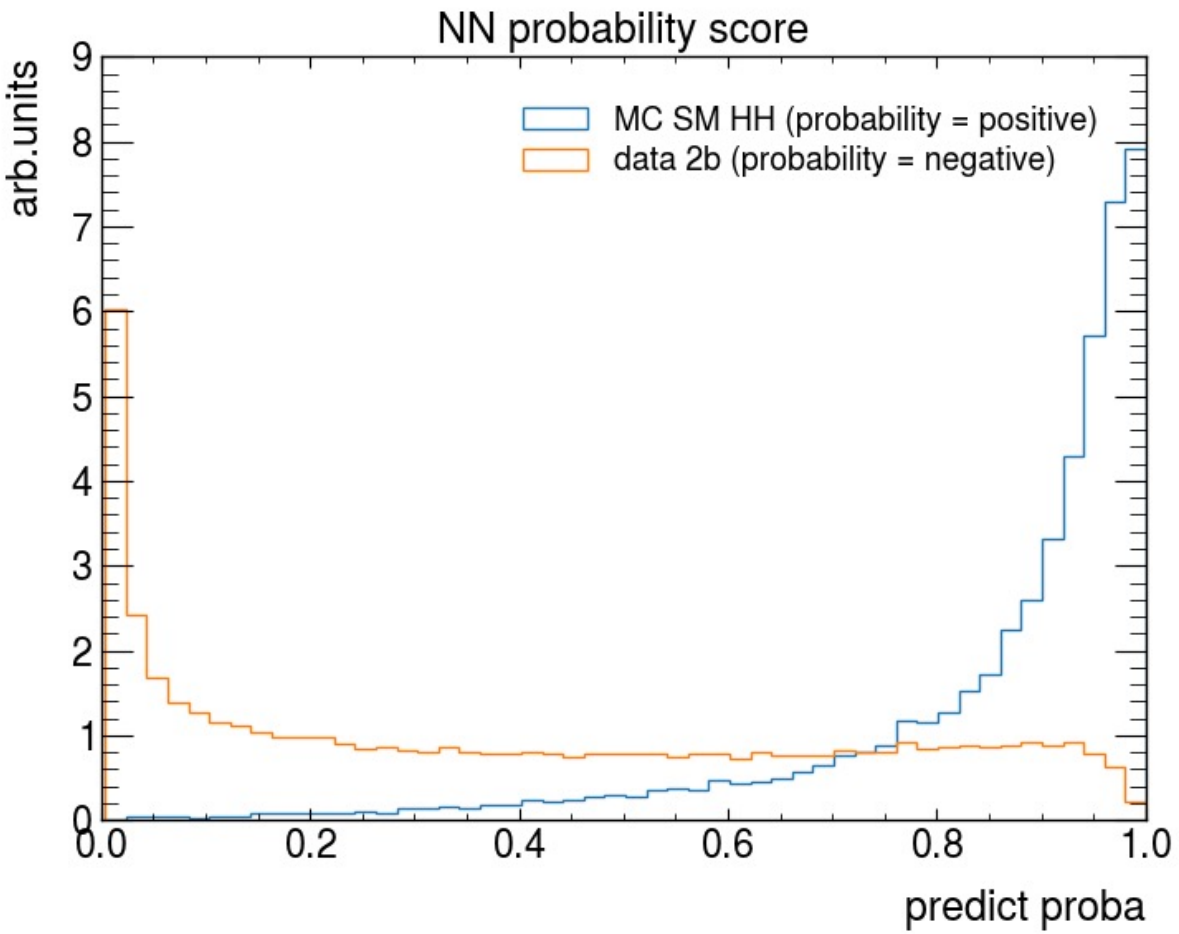
The 55 features:

'm_hh', 'X_hh', 'dEta_hh', 'njets', 'X_wt_tag', 'cos_theta_star', 'X_wt_notag', 'bkt_HT', 'pt_hh', 'pT_2', 'pT_4', 'eta_i', 'dRjj_1', 'dRjj_2', 'm_min_dj', 'm_max_dj', 'pairing_score', 'bkt_lead_jet_pt', 'bkt_third_lead_jet_pt', 'year', 'm_h1', 'E_h1', 'pT_h1', 'eta_h1', 'phi_h1', 'm_h1_j1', 'E_h1_j1', 'pT_h1_j1', 'eta_h1_j1', 'phi_h1_j1', 'angle_h1_j1'

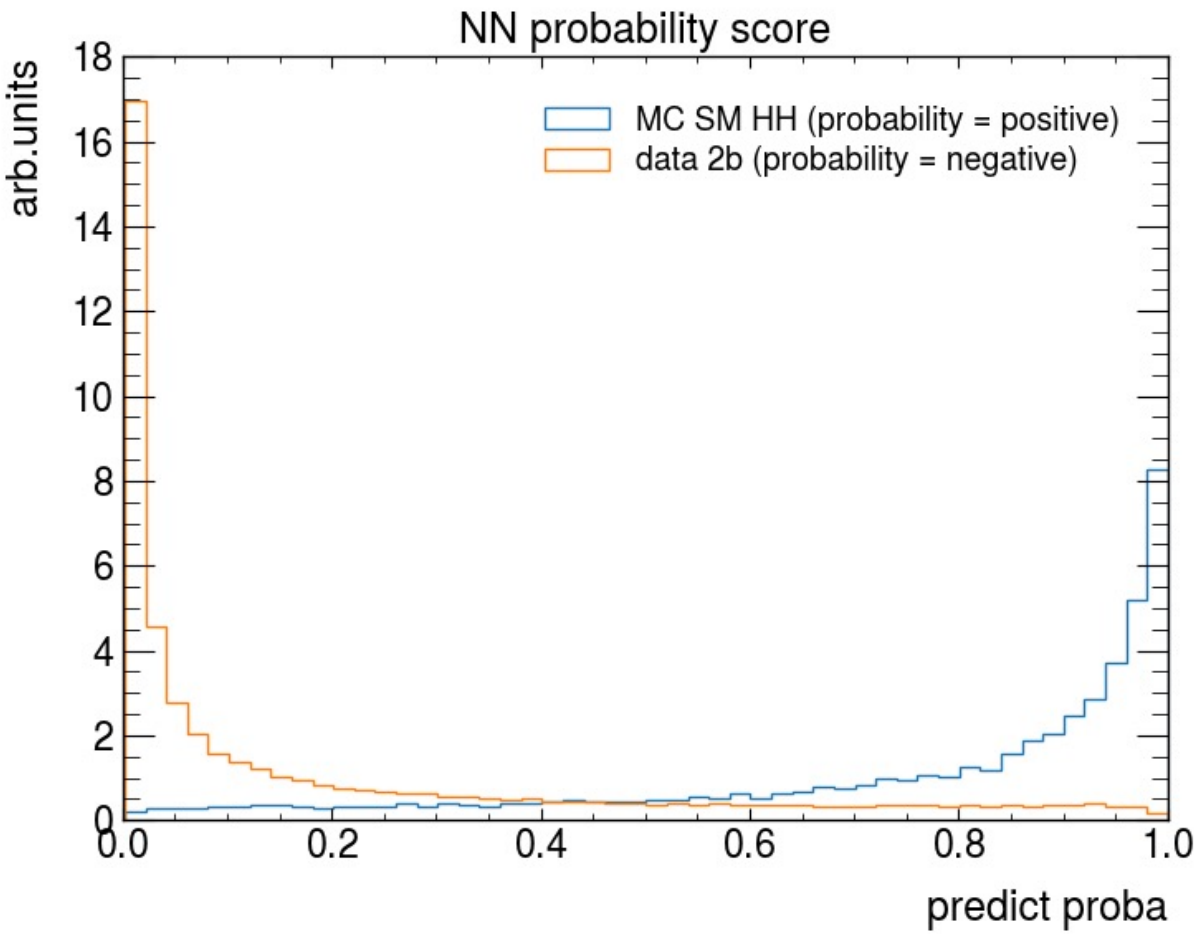
Note that for m_h1, there is also m_h2, and so on

Some initial results

TensorFlow

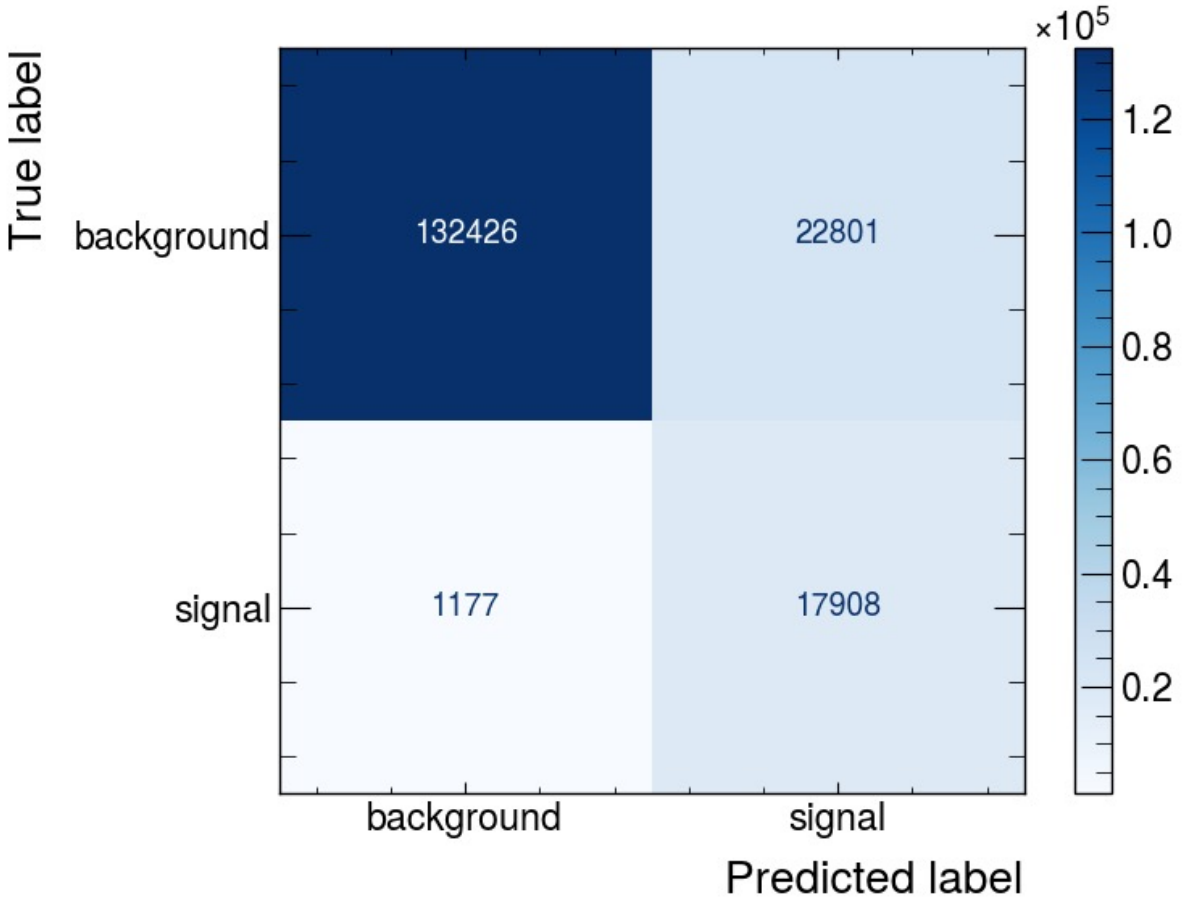


XGBoost

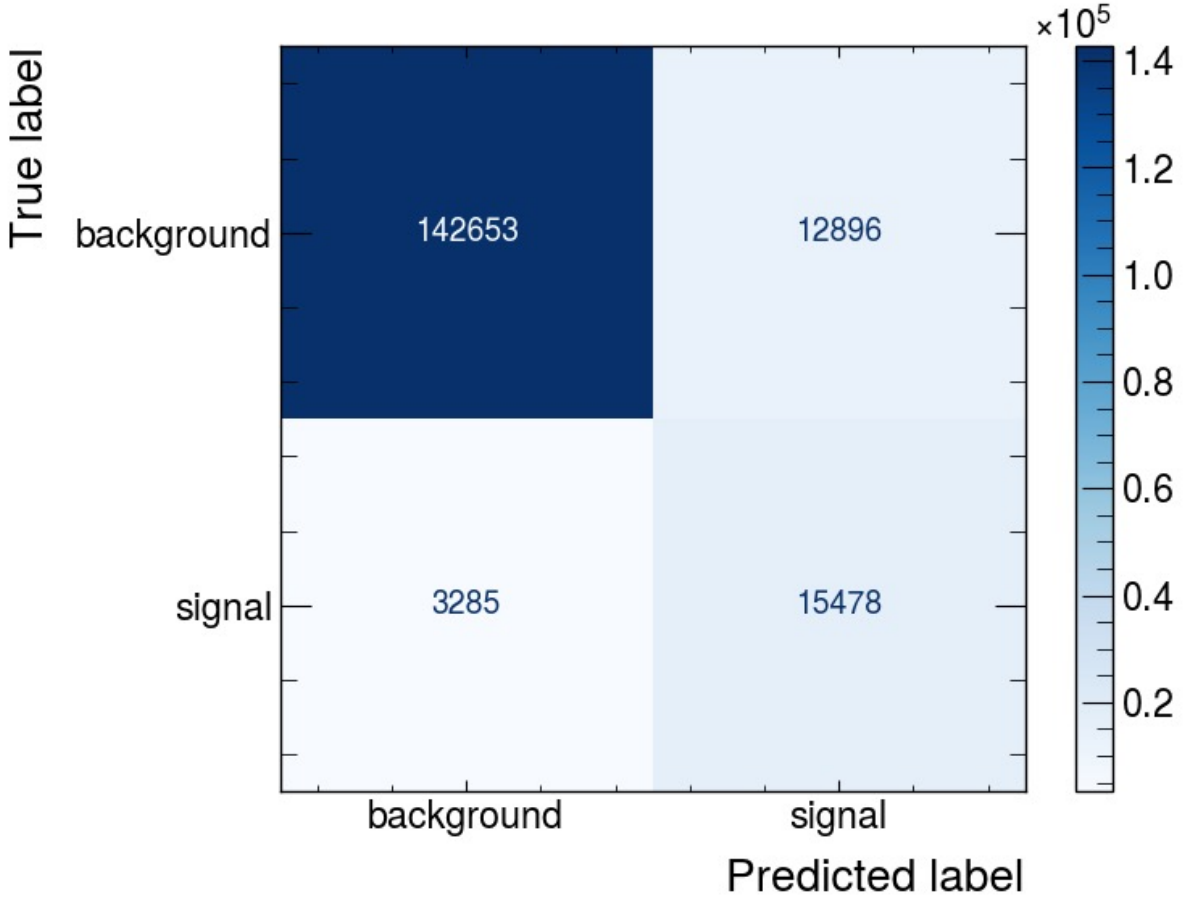


Some initial results

TensorFlow



XGBoost

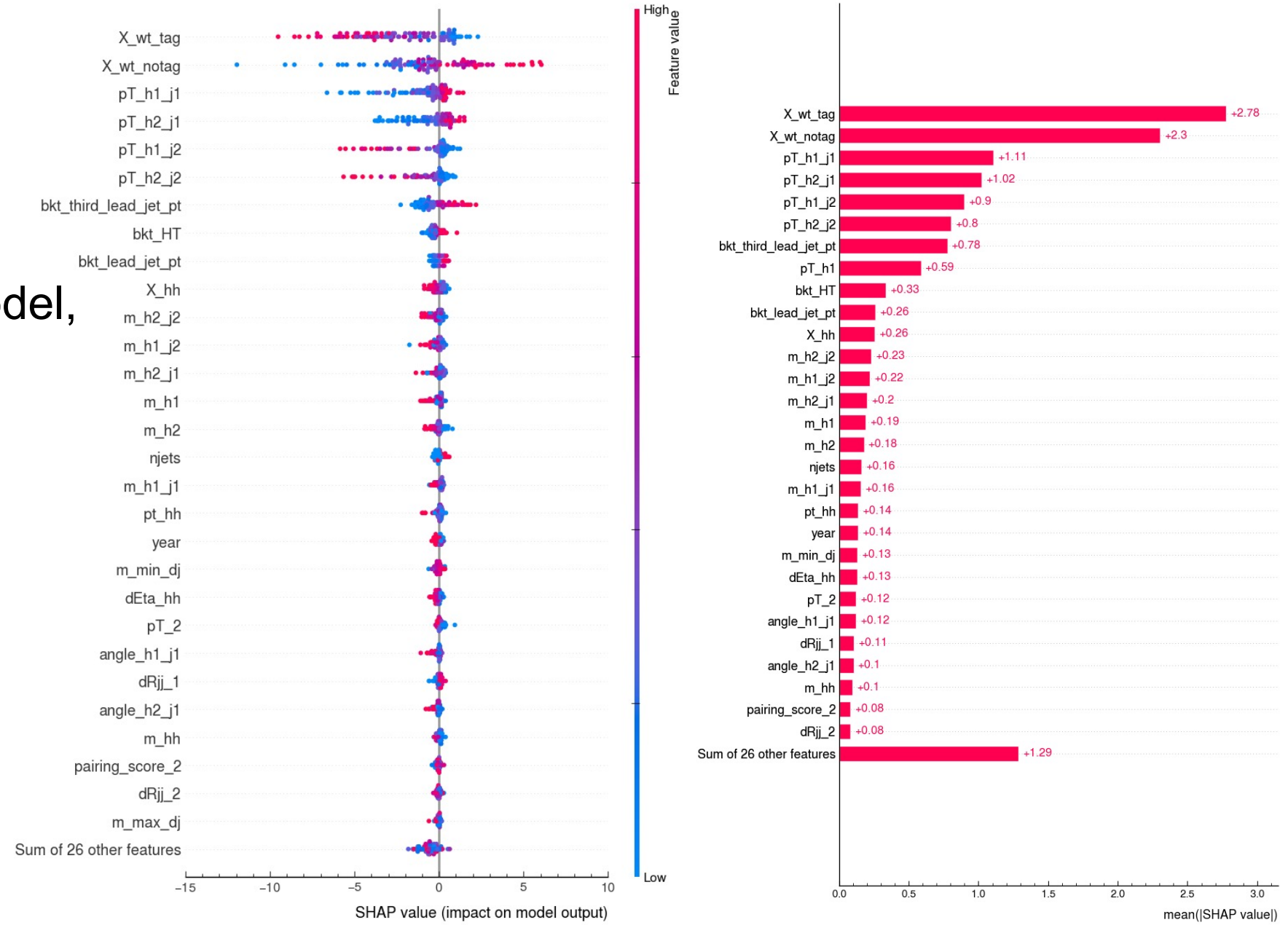


Some initial results

- Predict signal if only very confident → higher precision
- Avoid missing too many cases of signal → higher recall
- F1 score: weighted average of precision and recall

	Precision	Recall	F1 Score
TensorFlow	0.440	0.938	0.599
XGBoost	0.545	0.825	0.657

From the XGboost model, using SHAP



Some initial results in CR

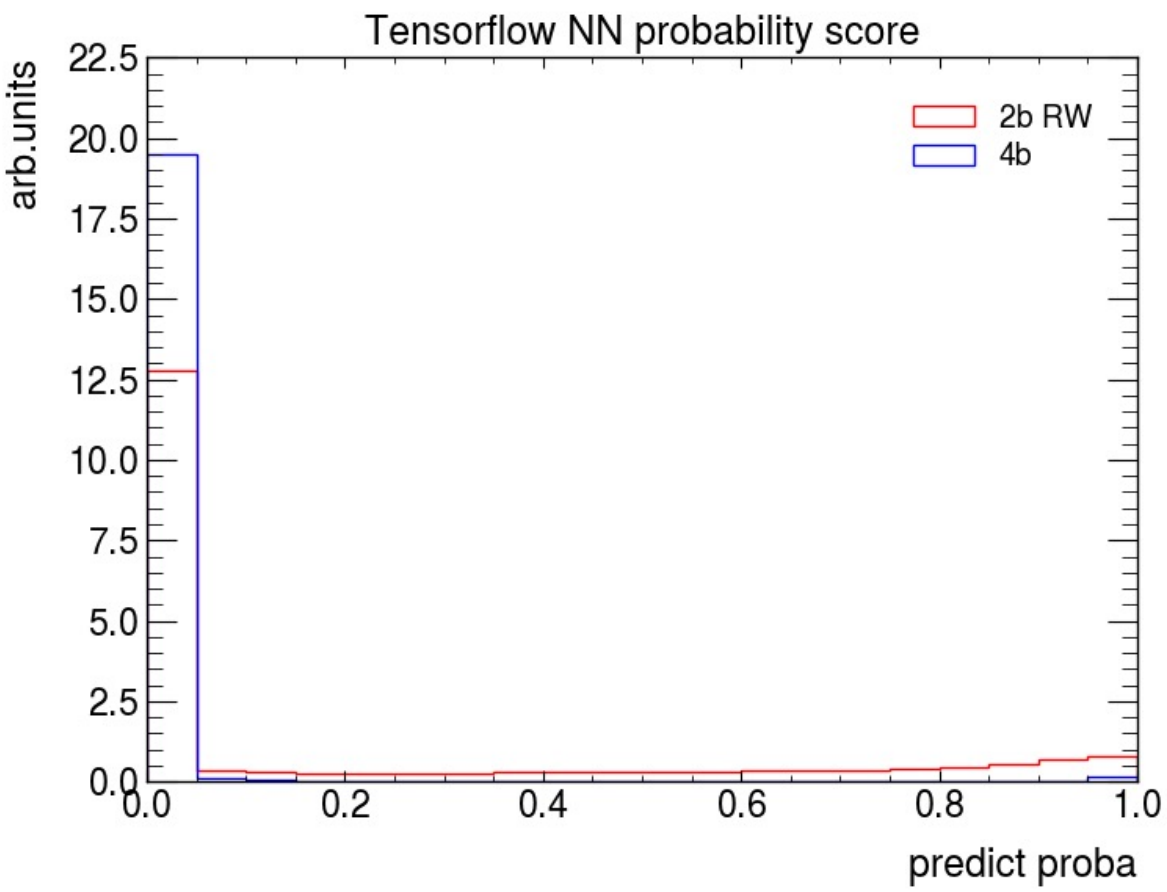
Masks:

- 2b RW: pass_vbf_sel==False, X_wt_tag>=1.5, ntag==2, rw_to_4b==True
- 4b: pass_vbf_sel==False, X_wt_tag>=1.5, ntag>=4

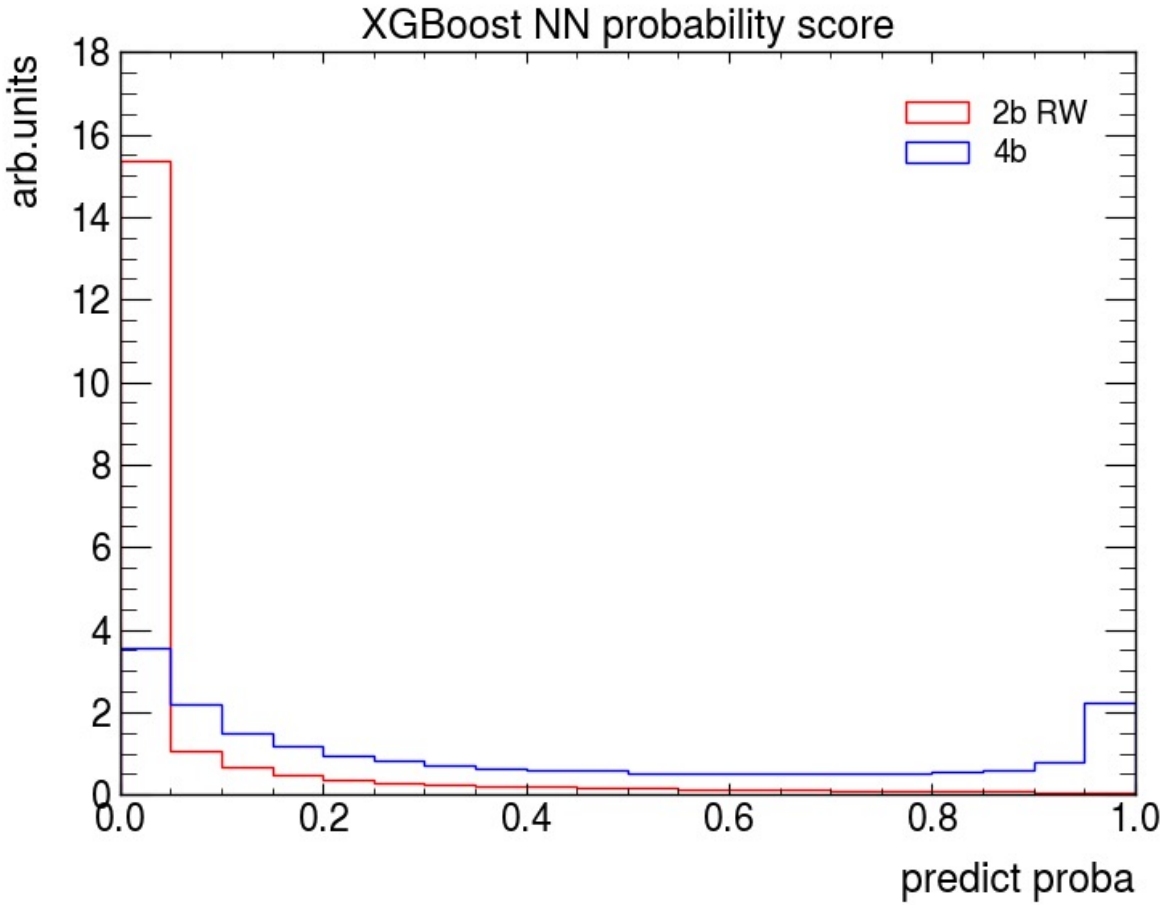
However, 2bRW and 4b did not agree well

Some initial results in CR

TensorFlow

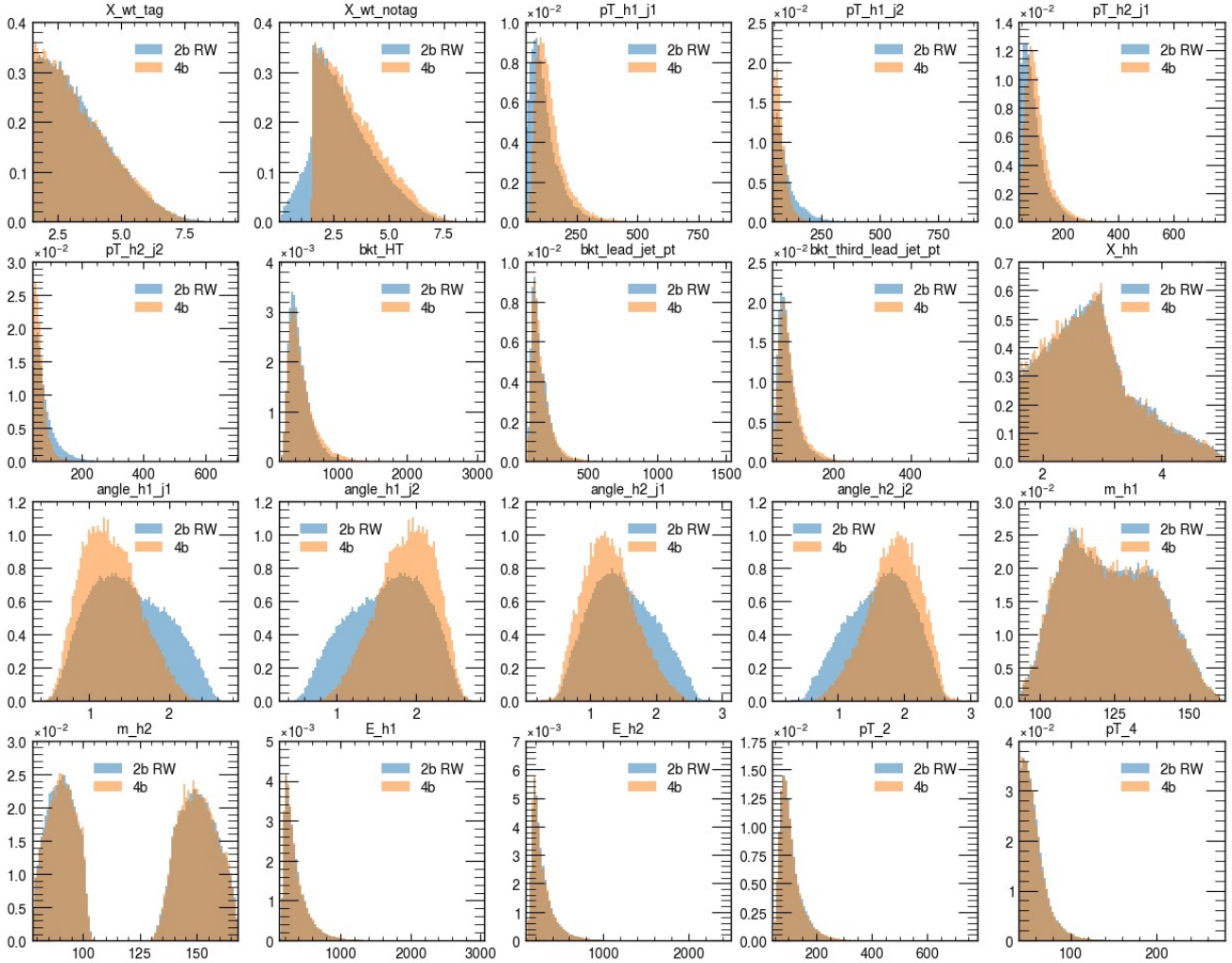


XGBoost



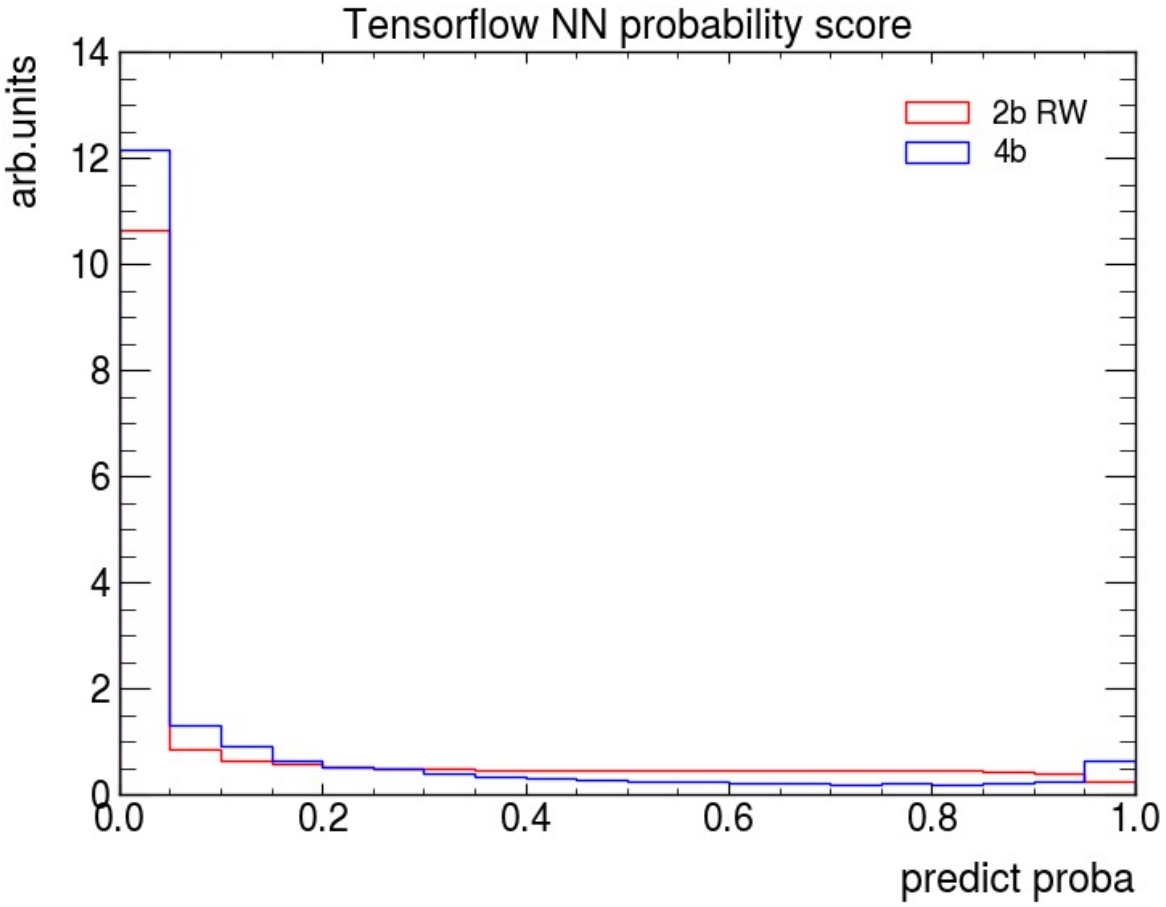
CR features: 2bRW vs 4b

- The most important features according to SHAP values
- Some features like X_wt_notag or the angles are **very different**, so these were removed

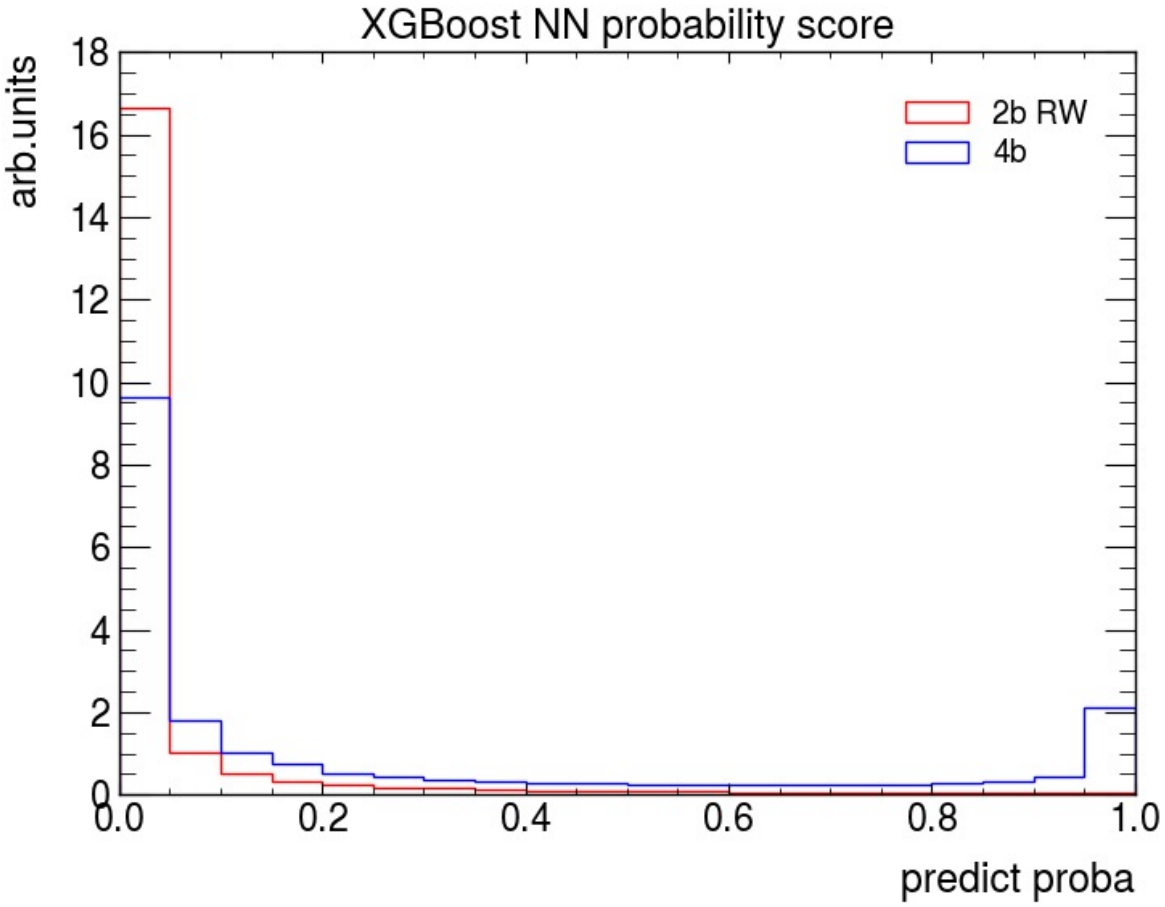


Removed very different features in CR

TensorFlow



XGBoost



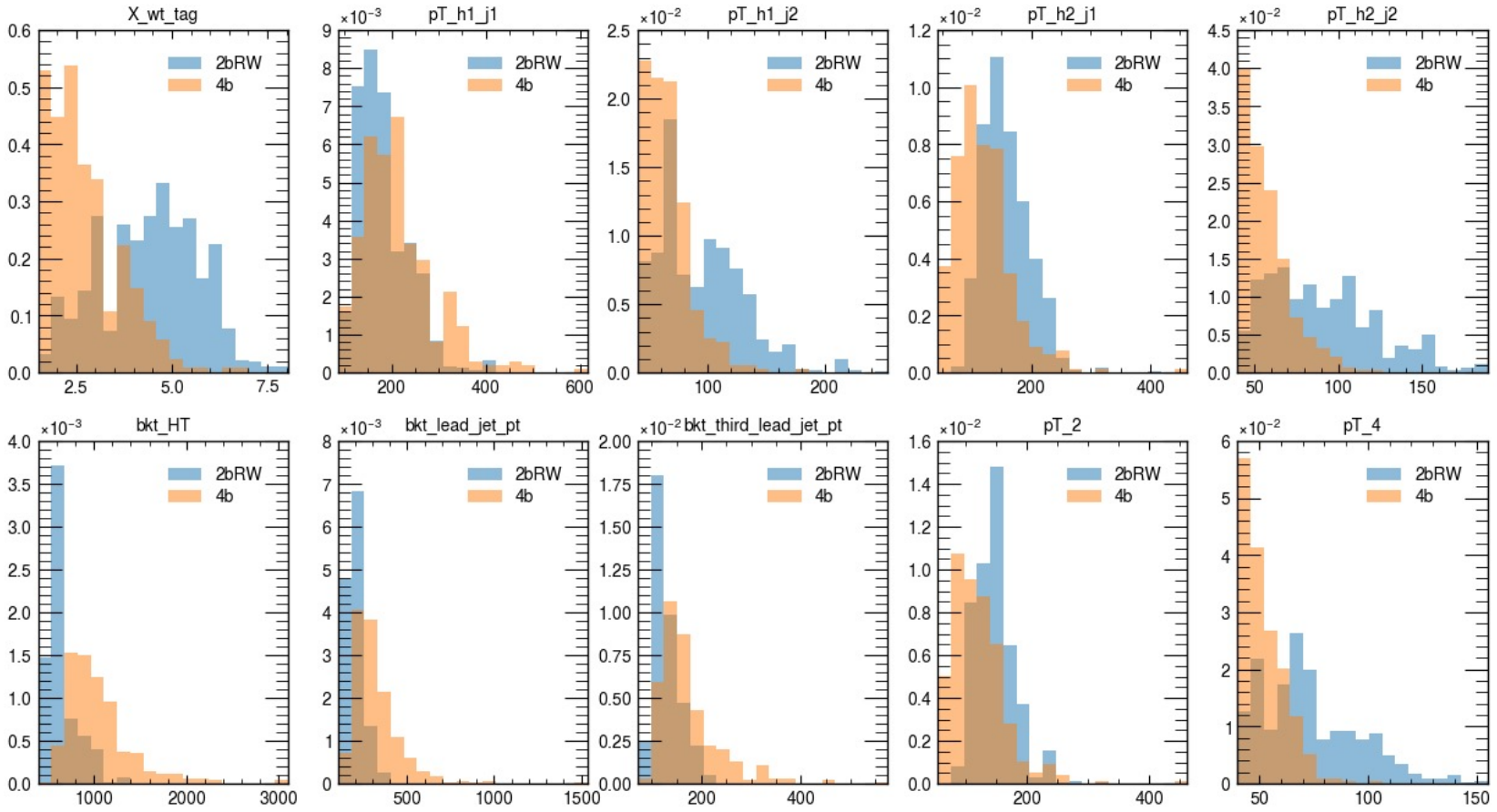
Removed very different features in CR

- After removing ['angle_h2_j2', 'angle_h1_j2', 'angle_h2_j1', 'angle_h1_j1', 'X_wt_notag’], 2bRW and 4b NN scores agree more
- However, XGboost models are not agreeing as well as TensorFlow models
- It is also more difficult to tune hyperparameters with XGBoost models, it is easier for TensorFlow models
- Decided to use **TensorFlow models only**

- Also found **small peak for 4b at signal** (1.0)

Comparison signal predictions in CR

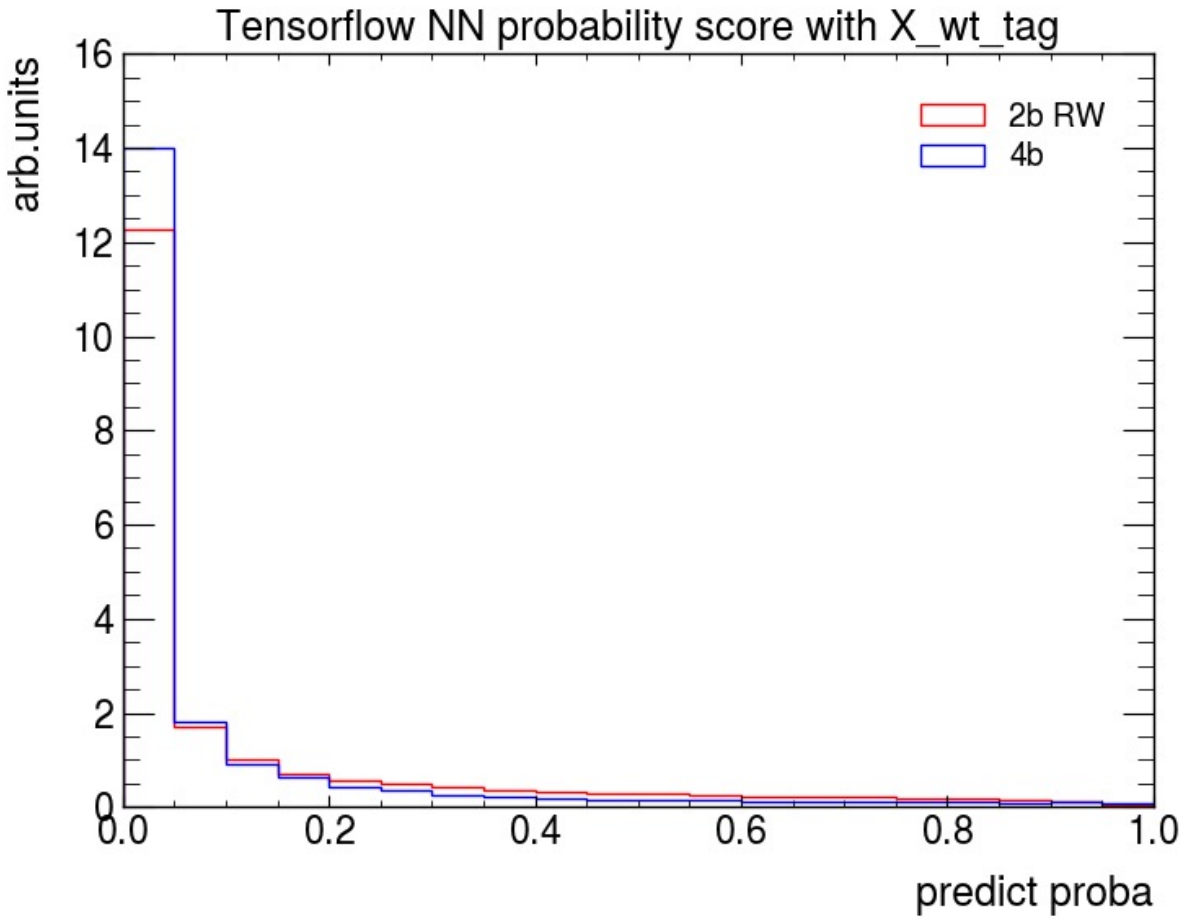
- Compared events from TensorFlow models where predictions were greater than 0.99
- Decided to **remove more variables** bkt_HT and pT_h1_j1 type features, and potentially X_wt_tag

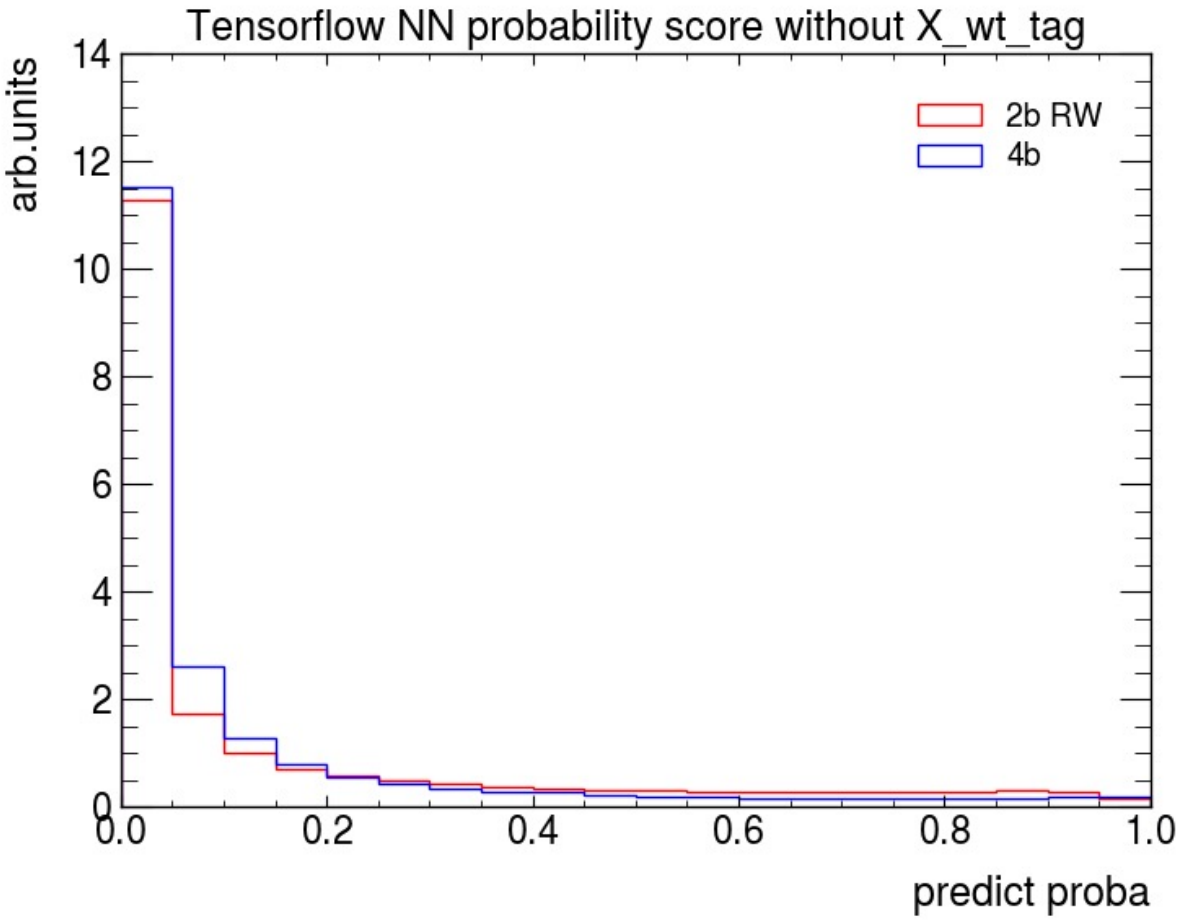


Hyperparameter tuning

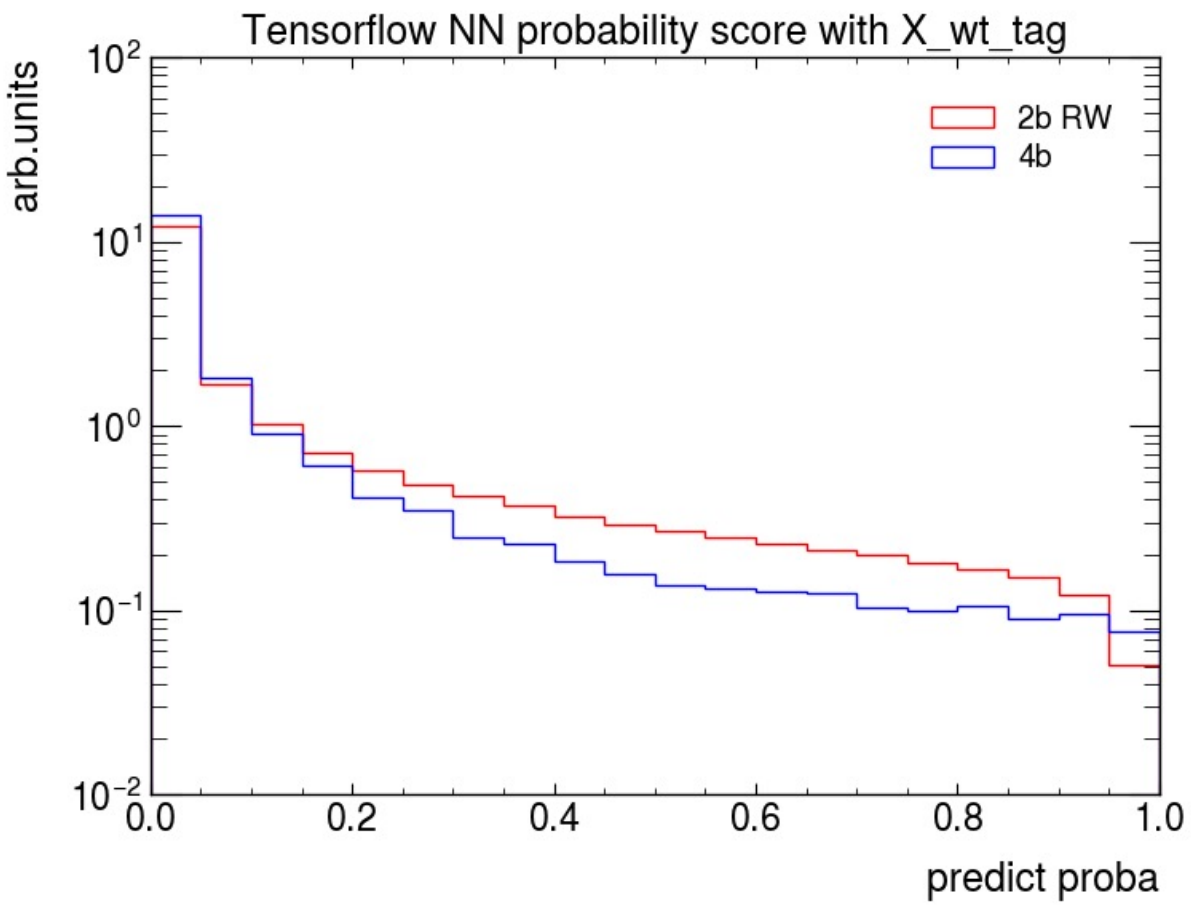
- TensorFlow models' hyperparameters can be tuned quite easily using **KerasTuner**. I decided to use **Bayesian Optimisation**.
- The model of the network is: Dense_0 + Dropout_0 + Dense_1 + Dropout_1 + … + Dense_output
- The number of hidden layers (Dense+Dropout) is a tunable parameter
- The number of units, the activation functions, the rate of dropouts and the learning rate for the optimiser (adam) are tunable
- Look for 100 trials (100 different combinations) and 3 executions per trial to reduce variance in the results.

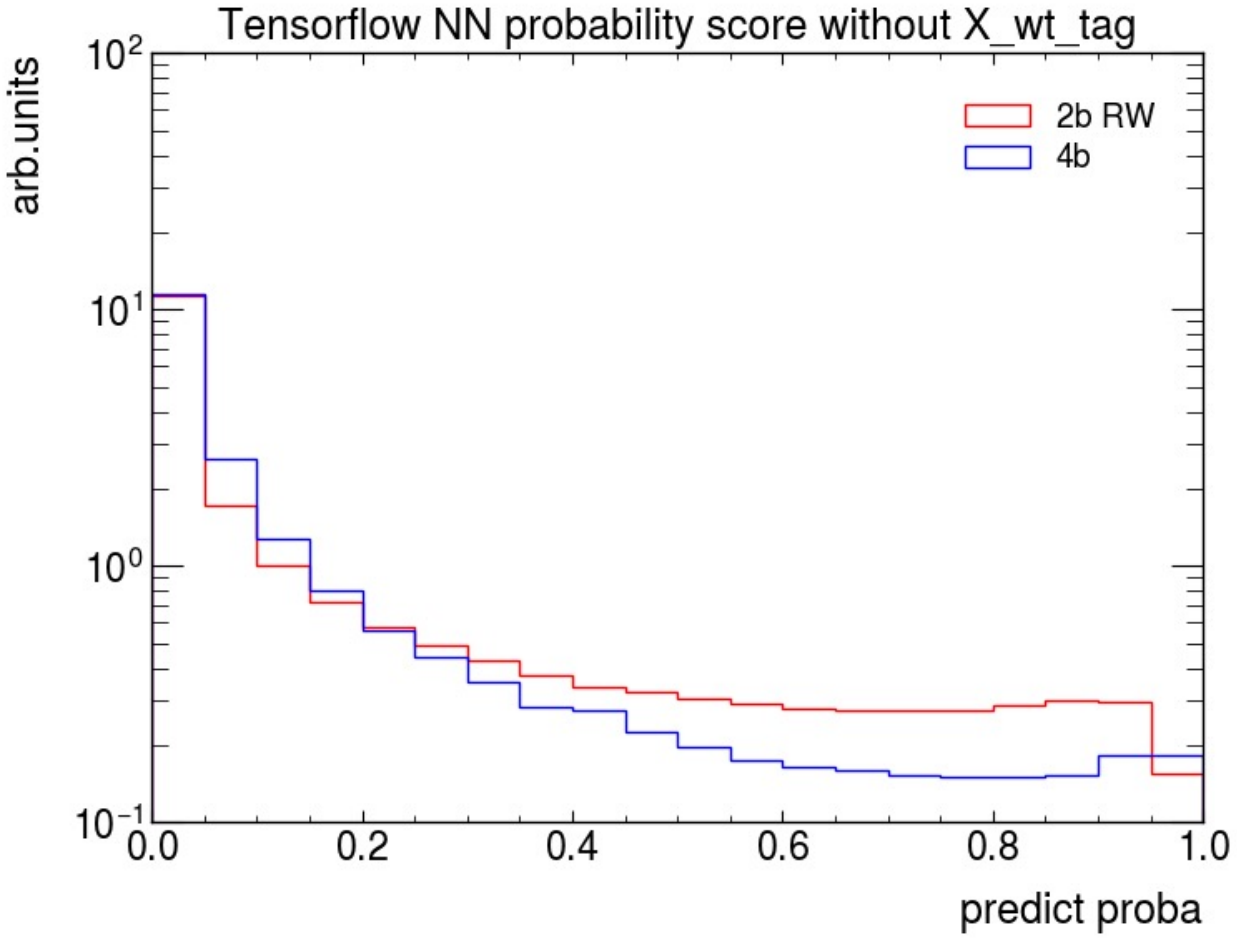
In the control region updated



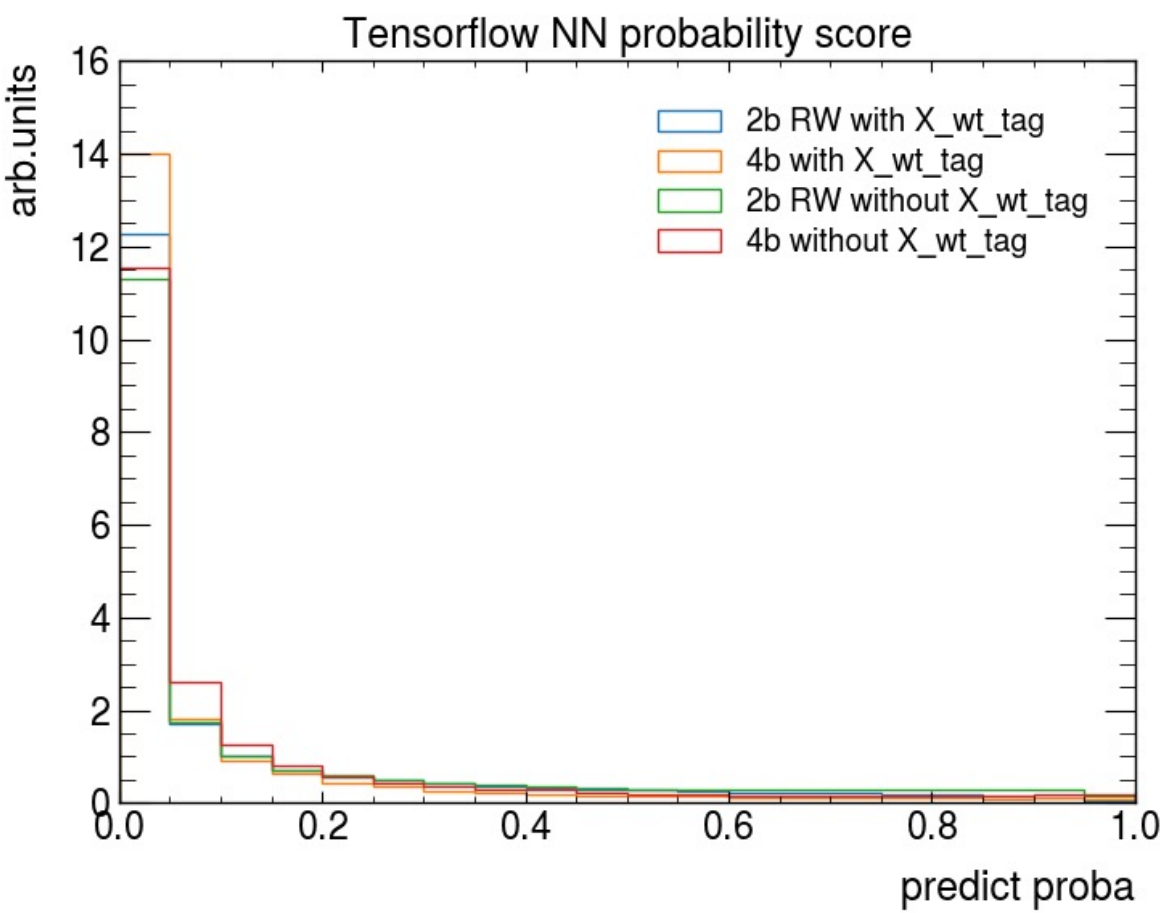


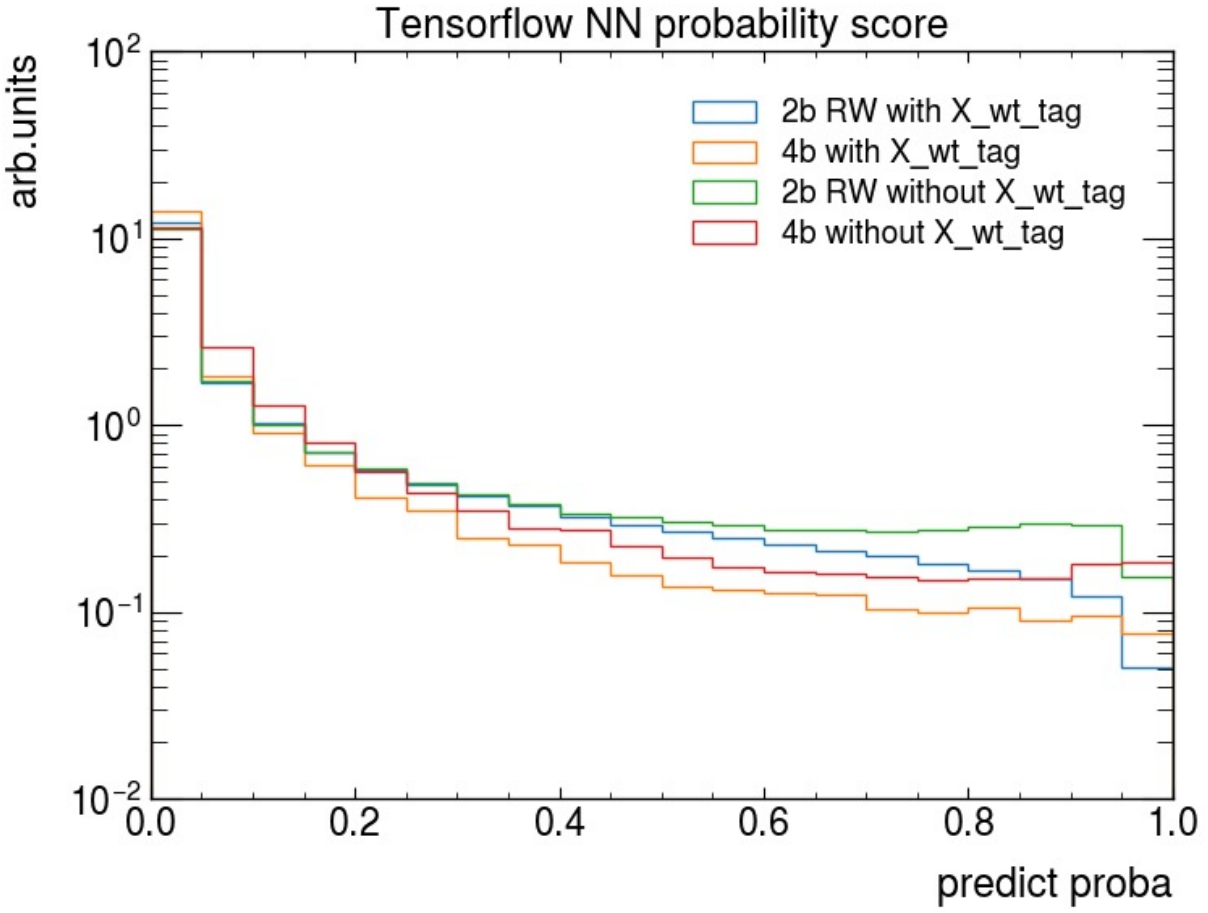
In the control region updated





In the control region updated

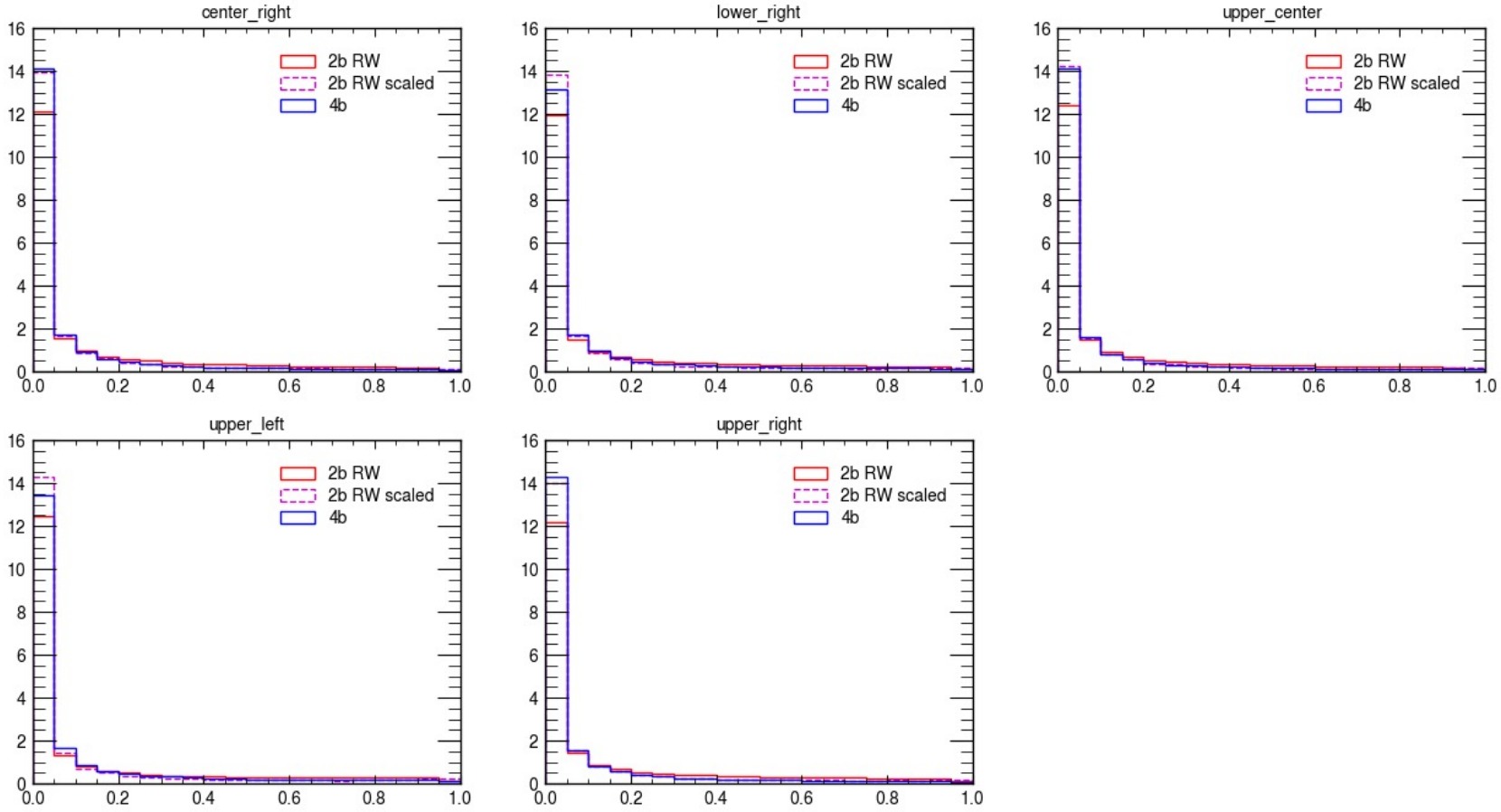




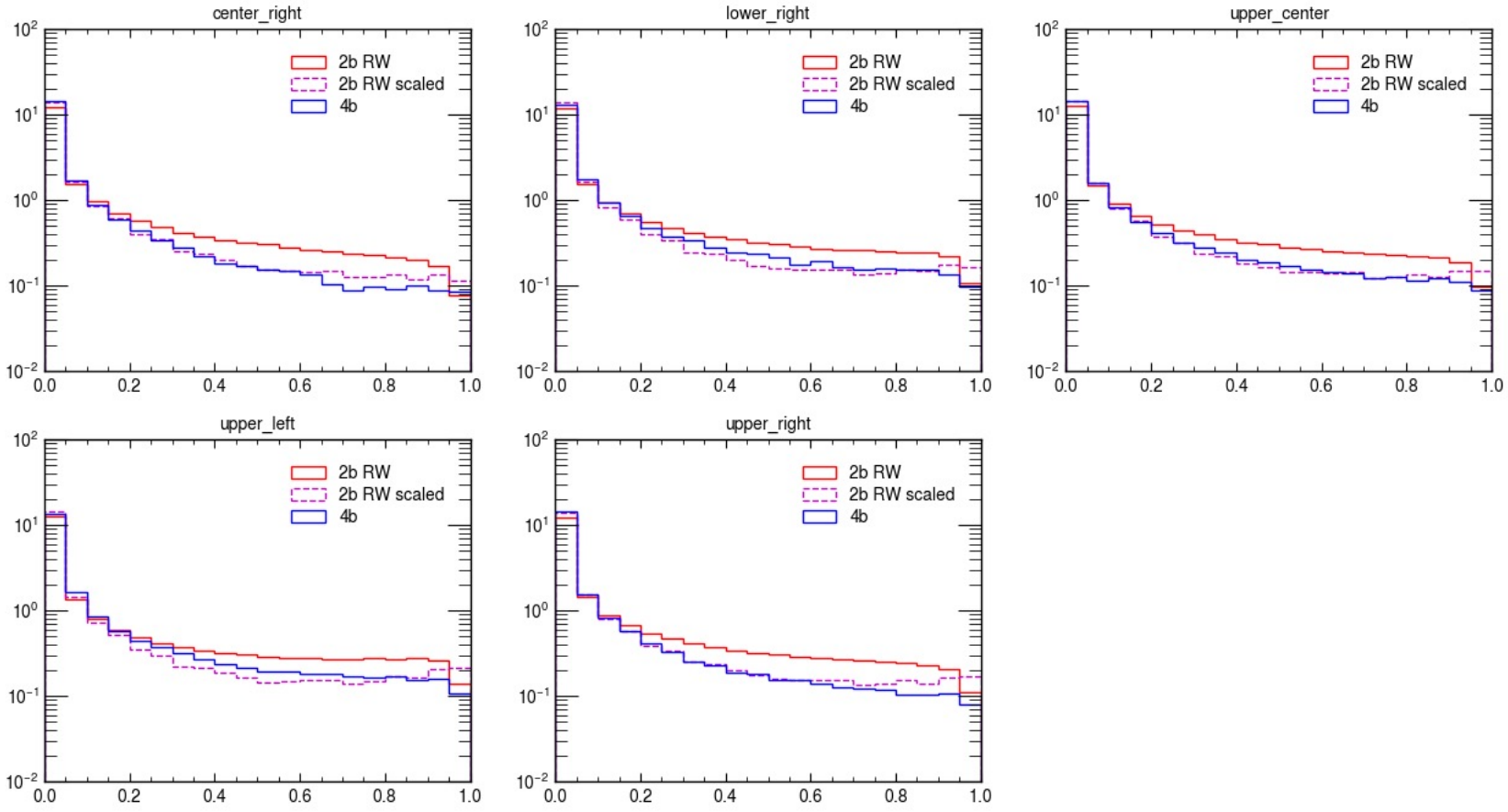
In the control region updated

- The best model using X_wt_tag outperformed and the 4b signal peak was also solved
- Found corrections for 2bRW by dividing the histogram weights in the CR and tested these in shifted regions from /eos/user/s/shhayash/HH4b/HH4b-ShiftReg-Validation/shifted/Data/

In the shifted regions



In the shifted regions



Other tests

- I also tried including pT_h1_j1 type variables, it sometimes worked well but not always, it depends how the model was actually trained. One could explore this further. It is difficult to say which features caused that 4b peak at 1.0.
- One thing that I tried but didn’t work well was Adversarial trained Neural Networks. I followed some works from other people and tried to de-correlate the masses m_h1 and m_h2. However, considering that normal Neural Networks already perform well in shifted regions, one could leave this for the future.

Summary

- Developed TensorFlow and XGboost models using many (up to 55) features and obtained good results in SR, but 2bRW vs 4b in CR was not agreeing
- Removed X_wt_notag and angles features; after validation in CR, TensorFlow models worked better but there was a small signal peak for 4b
- Removed bkt_HT and pT_h1_j1 type features; after validation in CR, results were looking better
- Found weights to scale 2bRW to 4b in CR and validated in shifted regions

Future steps

- Select the best models from KerasTuner and validate them in the control region. The second or third best model could work better than the best model. The metric used is not trivial.
- Include pT_h1_j1 type features (with X_wt_tag). More analysis could be done and again, the metric is not trivial.
- Explore more on XGboost models, they worked well at the beginning, but it is more difficult to tune the parameters.
- Use Adversarial Neural Networks to see if further improvement can be made in other regions.

Resources

Code:

- https://github.com/Xudong-Ke-Lin/HEP-HH4b-machine-learning

Useful Adversarial Neural Networks code:

- CERN jet tagger trained adversarially https://github.com/asogaard/adversarial/wiki