Department of Physics and Astronomy

PHAS0056: Practical Machine Learning for Physicist

Machine Learning Report on Higgs Boson Detection

Abstract

B-quark tagging in the context of Higgs boson detection is investigated in this report. B-quarks are decay product of Higgs interaction. A cut-based analysis and a machine learning (ML) based analysis have been performed. The cut-based sensitivity is measured with the reconstructed b-quark mass (mBB) and reached a 1.863 sensitivity. The ML model sensitivity is measured with the neural network (NN) output score provided and reaches $2.767^{+0.026}_{-0.014}$. The ML algorithm is tuned using sensitivity as an indicator. Further investigation is performed on the importance of parameters and on the effect of variable dataset size in training.

Contents

April 3, 2024

1 Introduction

The detection of the Higgs boson, first achieved at the Large Hadron Collider (LHC) experiment in 2012 by the ATLAS and CMS detectors [1, 2], relies on sophisticated detection techniques due to the particle's extremely short lifetime of approximately 1.6×10^{-22} seconds and the large background associated to it's decay products. In high-energy proton-proton collisions, Higgs bosons are produced and rapidly decay into other particles, with one of the primary decay channels being $H \to b\bar{b}$ (bottom-anti-bottom quark pair).

To detect the Higgs boson, physicists focus on detecting the particles resulting from the decay of the bottom quarks produced in the $H \to b\bar{b}$ process. These bottom quarks subsequently undergo a process called hadronization, where they decay into lighter particles in a cascade of interactions until stable particles are produced. These stable particles are detected by the ATLAS and CMS detectors as collimated sprays of particles known as "jets". The presence and properties of these jets provide crucial information about the original bottom quarks and, ultimately, the Higgs boson.

The goal of this report is to develop and investigate a tagging algorithm for identifying jets originating from the decay of b-quarks, which themselves decay from the Higgs boson. The development of a good tagging algorithm for identifying jets originating from the decay of b-quarks faces significant challenges in distinguishing between background events and Higgs boson signal events. Some of this background is composed of light-quark and gluon jets, and some other non-Higgs processes. The key metric to assess the algorithm's efficacy is the signal sensitivity. In the cut-based analysis, that sensitivity is assessed with a discriminant based on the reconstructed b-jet mass. In the ML based analysis, the sensitivity is based on the NN output score provided. This ultimately determines, given the number of signal events compared to the number of background events, the likelihood that the process of interest will be observed.

This report will outline the methodology employed in developing the tagging algorithm, including data pre-processing, feature selection, model training, and performance evaluation. Additionally, it will discuss the significance of the algorithm in the context of particle physics research and the quest to further understand the fundamental building blocks of the universe.

1.1 Introduction to Dataset

The different variables in the dataset provided are shown in Table 1. There fifteen variables, all corresponding to either a signal event labeled one or a background event labeled zero. Although there are a number of different background processes present, the task can be reduced to a binary classification task.

Table 1: Kinematic and topological parameters used to identify events

Variable	**Description**	**Label**
n_J	Number of jets in the event (always 2)	nJ
n_{Tags}	Number of b-tagged jets in the event (always 2)	nTags
$\Delta R(b_1 b_2)$	Angular distance between the two b-tagged jets	dRBB
$p_T^B 1$	Reconstructed transverse momentum of the highest p_T b-tagged jet	pTB1
$p_T^B 2$	Reconstructed transverse momentum of the 2nd highest p_T b-tagged jet	pTB2
p_T^V	Reconstructed transverse momentum of the vector boson	pTV
m_{BB}	Reconstructed mass of the Higgs boson from the two b-tagged jets	mBB
m_{top}	Reconstructed top quark mass	Mtop
m_T^W	Reconstructed transverse mass of the W boson	mTW
E_T^{Miss}	Missing transverse energy	MET
$dY(W, H)$	Separation between the W boson and the Higgs candidate	dYWH
$d\phi(W, H)$	Angular separation in the transverse plane between the W boson and the Higgs candidate	dPhiVBB
$MV1_{\text{cont}}^{B1}$	Classification output of whether the leading jet is a b or not	MV1cB1_cont
$MV1_{\text{cont}}^{B2}$	Classification output of whether the sub-leading jet is a b or not	MV1cB2_cont
$n_{\text{cont}}^{\text{Jets}}$	Number of additional jets found in the event	nTrackJetsOR

2 Part 1: Cut-Based Analysis

The first part of this analysis aims to produce a cut-based analysis on the dataset to isolate the target signal as much as possible. This cut-based analysis will serve as a baseline threshold to assess against the performance of the Machine Learning based algorithm. Cuts are applied on relevant parameters where the b-quark-characteristics were mostly readily isolated. The list of parameters with their respective cuts are given in Table 2.

Table 2: Parameters included in the cut-based model

Label	Description	Cut
Mtop	Reconstructed Top quark Mass	$Mtop > 209000$
dRBB	Angular distance between two b-tagged jets	$dRBB < 1.39$
pTB2	Reconstructed transverse momentum of the b-tagged jet with the 2nd highest p_T	$pTB2 > 49000$
pTV	Reconstructed transverse momentum of the vector boson	$pTV > 140000$

The four parameters are chosen from the list of the list of 14 (Table 1) after varying the individual cut values and seeing where the sensitivity would be highest, and if it would increase at all. The final sensitivity achieved:

Sensitivity achieved before cuts: 1.499

Sensitivity achieved after cuts: 1.863

This is a increase of 0.364, or a 19.5% increase in sensitivity compared to the sensitivity before applying cuts. The cut-based analysis provides a baseline sensitivity to assess of the ML based model achieves significantly better than the cut-based analysis. Figure 1 shows a histogram of the mBB distribution where the target Higgs boson is the red outline (VH → Vbb x 20). Figure 2 shows the same plot after cuts are applied.

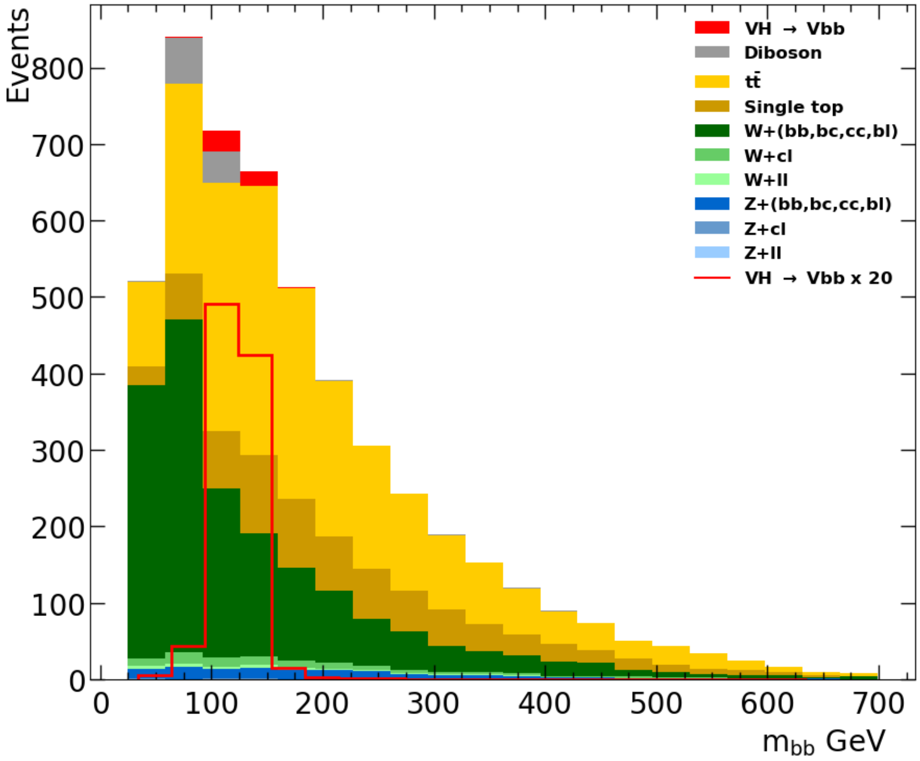


Figure 1: Sensitivity Plot Before Applying Cuts. The sensitivity value obtained from the model is 1.499

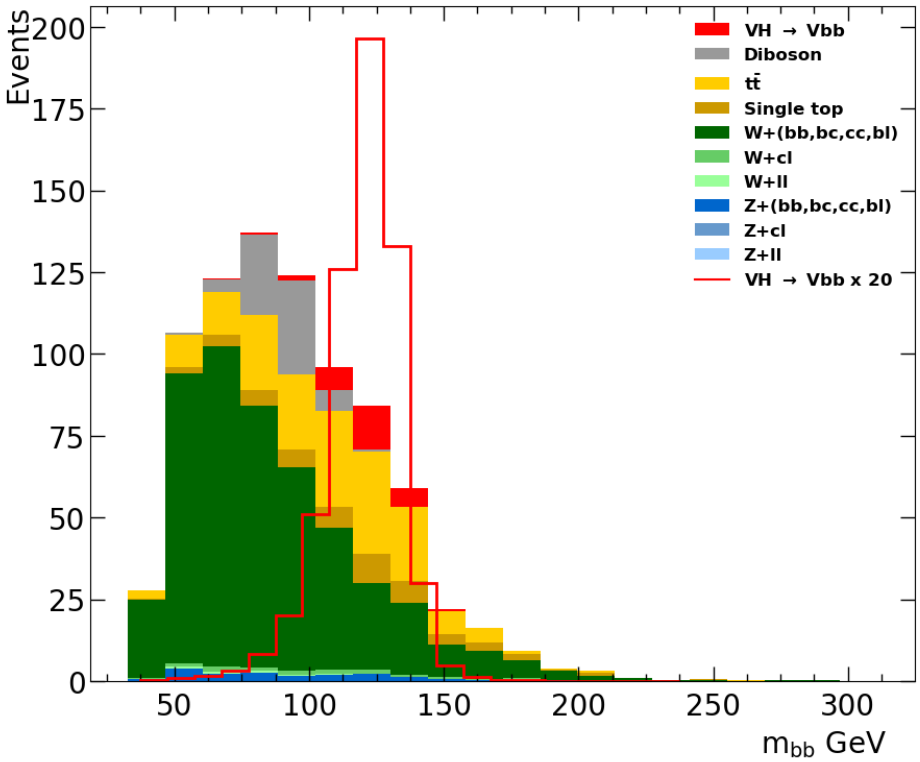


Figure 2: Sensitivity Plot After Applying Cuts. The sensitivity value obtained from the model is 1.863

3 Part 2: Machine Learning Algorithm

3.1 Pre-Processing of input-variables

Data pre-processing is essential for optimising neural network performance. Initially, data cleaning eliminates rows where missing values are detected. This maintains the integrity and prevents biased learning, as some variable in the data would then have more data points that others. The next step is normalisation. This ensures equitable influence on model training. Parameters can have very different scales, to prevent parameters with higher values to influence training more than parameters with smaller values, normalisation of data is performed. In this case, normalisation is done with mean of zero and standard deviation of one. The following steps is to remove unnecessary data, or data that is expected to have very little influence on the learning of the model. For this specific task, parameters nJ and nTags do not tell the model anything specific, as they are always equal to two. These parameters are removed, and training is performed with the thirteen remaining parameters shown in Table 1.

3.2 Model Architecture

A fully connected feed forward neural network has been chosen for the binary classification task due to the non-sequential nature of the data and the relatively small number of parameters (thirteen). Another important aspect of this architecture is that with multiple hidden layers, the model can learn hierarchical relationships between inputs. In this data set, some parameters are more important than others. Tuning the model involves optimising various hyper parameters such as the width (number of nodes) and length (number of layers), training parameters (epochs, batch sizes and learning rate), activation functions, optimisation algorithms and loss functions. The universal approximation theorem assures us that a sufficiently large feed forward neural network can represent any function (relationship) [3]. However, ensuring that the algorithm learns the relationship without overfitting is crucial. Overfitting occurs when the model memorizes the training data but fails to generalise to new data. To prevent that, tuning decisions are guided by the model's performance on *test* data measured by the sensitivity to NN score output, rather than solely relying on training accuracy or loss.

Furthermore, it is essential to recognise that all parameters of the neural network are interconnected. So there is no practical method in finding the best possible parameters. The only possible way of finding the best combination of parameters would be to try every combination. This is impractical due to the vast number of possibilities.

3.2.1 Length and Width of model

A fully connected feed forward neural network comprises of an input layers, one ore more hidden layers, and an output layer. When tuning the model, an important assumption is made about model complexity: simple models tend to generalize the data better than very complex models, which tend to overfit the data. Therefore, the approach here is to start with a simple feed forward neural network with a decreasing number of nodes in each layer. Decreasing the number of nodes in each layer will help the model to only keep the most important parameters as they move through the layers.

The number of nodes in each layer is estimated with plots of the number of nodes against the model's sensitivity to NN output score. An assumption that can be made about feed forward neural network is that layers are independent in the forward direction [4]. For example, the number of nodes in the second layer will have no impact on the efficacy of the first layer. The strategy is to build the model one layer at a time, where the number of nodes in each layer is gradually determined by plotting the number of layers in the new layer against the model's sensitivity. The last layer is added when the sensitivity reaches a plateau.

The plots of sensitivity against number of nodes in each layer can be seen on Figure 3. When the sensitivity starts to somewhat plateau against nodes, an appropriate number is chosen. Taking the highest sensitivity is not sensible as there is variability in the models performance with equal training.

The final model summary can be seen in Table 3

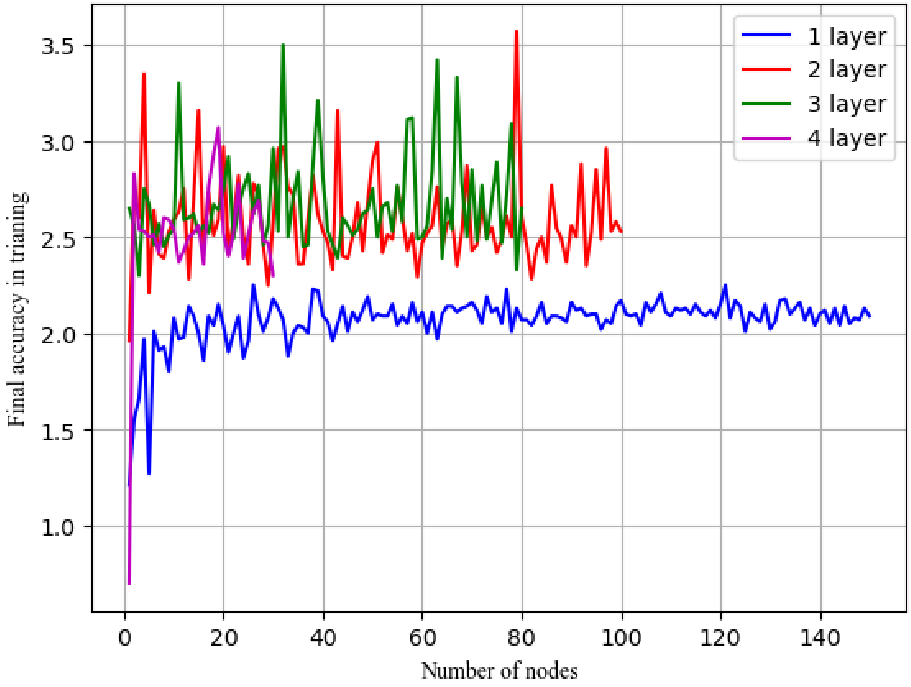


Figure 3: Sensitivity Variation Across Different Numbers of Nodes in Each Layer. Each line in the plot corresponds to a specific layer of the neural network architecture, showcasing how sensitivity varies with the number of nodes in that layer.

Table 3: Summary of the Model Architecture. This table provides an overview of the neural network model's architecture, including the type of layers, their output shapes, and the number of parameters.

Layer (type)	**Output Shape**	**Param #**
(Dense)	(None, 120)	1560
(Dense)	(None, 65)	7865
(Dense)	(None, 49)	3234
(Dense)	(None, 14)	700
(Dense)	(None, 1)	15
Total params:		7699 (30.07 KB)
Trainable params:		7699 (30.07 KB)
Non-trainable params:		0 (0.00 Byte)

3.2.2 Optimisation and Loss Functions

Considering the optimiser and loss functions together is crucial when selecting them. The loss function represents the quantity to the minimized during training, while the optimiser determines how the network will be updated based on the loss function. There is a range of possible combinations to choose from and all are given in Table 4.

Table 4: Supported Optimisers and Loss Functions for Binary Classification in Keras.

Optimisers	**Loss Functions**
SGD	BinaryCrossentropy
RMSprop	BinaryFocalCrossentropy
Adam	CategoricalFocalCrossentropy
Adadelta	Hinge
Adagrad	CategoricalHinge
Adamax	SquaredHinge
Nadam	KLDivergence
Ftrl	

The plot in Figure 4 gives the performance of each combination of loss function and optimiser algorithm, measured by the NN output score sensitivity. Upon repeated measurements of this plot, slight variations would be seen in the highest performing combination. Nonetheless, the top contenders remain consistent. For the final model, the `Nadam` optimiser paired with the `BinaryFocalCrossentropy` loss function are selected. This combination, in addition to reaching high sensitivity, also has a very low associated loss and very high accuracy in training.

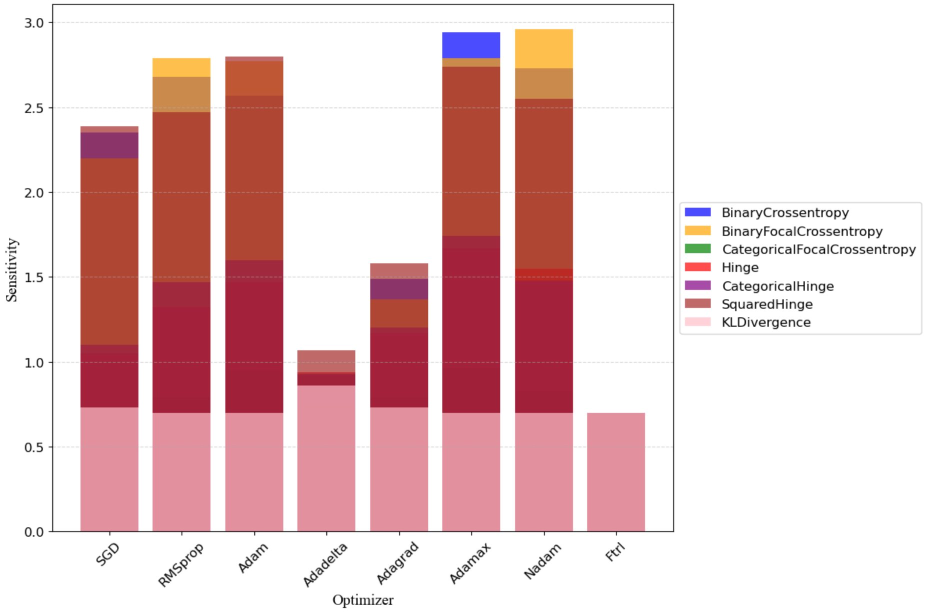


Figure 4: Sensitivity Achieved with Different Combinations of Loss Function and Optimiser, Grouped by Optimiser Functions.

3.2.3 Activation Function

The choice of activation function for each layer depends on the data provided and task at hand. In this binary classification task, where one corresponds to the target signal and zero corresponds to background, the `sigmoid` activation function is a suitable choice for the output activation. The `sigmoid` squishes the output values of the neural network into a probability score. This makes it intuitive to interpret the output of the neural network as the likelihood or confidence of a sample belonging to the positive class. For the hidden layers, using the same activation from the a list of all supported activation functions by `keras`, the different performances are shown on Figure 5. The highest performing activation functions are `relu`, `softsign`, `elu`, `silu` and `mish`. Out of five choices, `relu` is chosen because it's simplicity to compute and will be used in all hidden layers.

3.2.4 Training Parameters

There are three important training parameters to consider: learning rate, batch size and epochs. Learning rate determines the step size at which model parameters are updated during training. The batch size specifies the number of samples that are propagated through the network at each training iteration. The epoch refers to one complete pass through the entire training dataset. To optimise the training parameters, the parameters are assessed independently and in this order: learning rate, batch size and then epoch.

A high learning rate will result in larger steps, the model will converge quicker but might overshoot the optimal solution in gradient descent. Smaller learning rates require more epochs to reach convergence but can lead to more stable training [3]. Learning rates are assessed on training with three epochs. On Figure 6a, a plot of learning rate against sensitivity at a pace of three epochs is shown. Here it can be seen that smaller learning rates perform better than larger learning rates. A learning rate of 0.005 is optimal as it gives the same precision as smaller learning rates but will get there faster.

The number of epochs for a learning rate of 0.005 is the next to be assessed. Assessing the optimal number of epochs for such a learning rate is performed with a batch size of 58. Figure 6b shows sensitivities for epoch from one to ten.

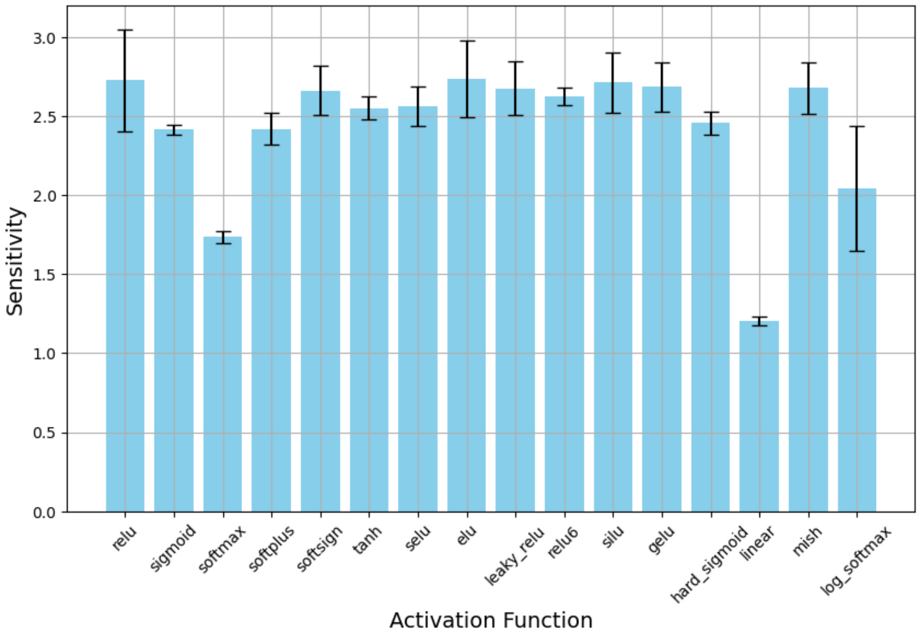
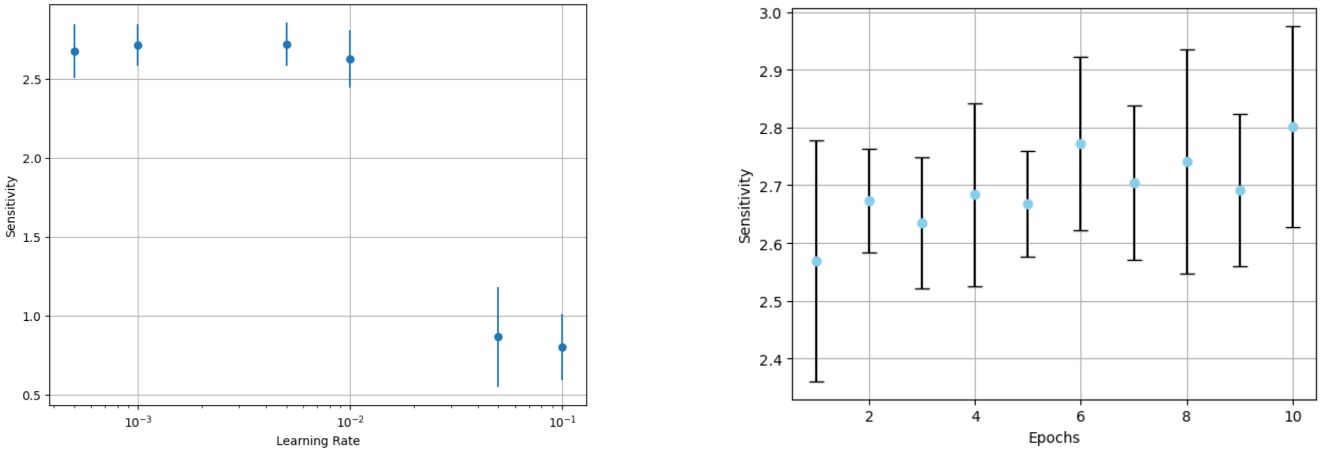


Figure 5: Mean Sensitivity with Error Bars for Different Activation Functions. The bar plot illustrates the mean sensitivity values for various activation functions used in a neural network model, along with error bars indicating the standard deviation. Error bars represent the variability in sensitivity across multiple iterations of the model.



(a) Learning Rate vs. Sensitivity

(b) Sensitivity vs. Number of Epochs

Figure 6: Comparison of Sensitivity with learning rate (a) and epoch number (b).

With learning rate set at 0.005 and number of epochs set at 10. To investigate the batch size, two values are selected corresponding to either having a large (128) or a small (32) batch size. One Figure 7 shows that a large batch size performs better. The final training parameters are a learning rate of 0.005, epoch number of 10 and a batch size of 128.

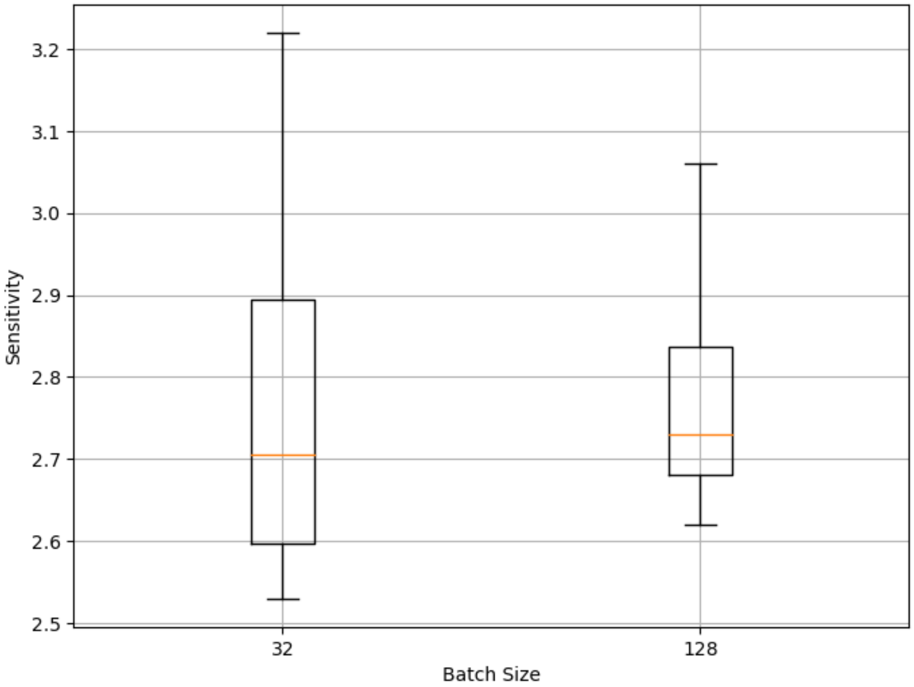


Figure 7: Effect of Batch Size on Sensitivity: Comparison of model sensitivity across different batch sizes (32 and 128) for a binary classification task.

3.3 Assessment of complete model

The complete model architecture is listed below, the model reaches a final sensitivity of $2.767^{+0.026}_{-0.014}$ in NN output score.

```
def model_Final(num):
    """
    Create and compile the final neural network model.

    Parameters:
    - num: Number of features
    - learning_rate: Learning rate for the optimizer

    Returns:
    - Compiled Keras model
    - Sensitivity achieved by the model
    """
    variables = [’mBB’, ’dRBB’, ’pTB1’, ’pTB2’, ’pTV’, ’Mtop’, ’mTW’, ’MET’, ’dYWH’, ’
        dPhiVBB’, ’MV1cB1_cont’, ’MV1cB2_cont’, ’nTrackJetsOR’]

    # Prepare the data
    (x_train, y_train, w_train, dset_val, dset_test) = scale_prepare_data(df_train,
        df_val, df_test, variables, scaler=’standard’)
    selected_variables = best_pca(x_train, variables, num)

    # Prepare data with selected variables
    (x_train, y_train, w_train, dset_val, dset_test) = scale_prepare_data(df_train,
        df_val, df_test, selected_variables, scaler='standard')

    # Initialize a Sequential model
    model = Sequential()

    # Add the input layer with 120 units and ReLU activation function
    model.add(Dense(units=120, input_dim=num, activation='relu'))

    # Add hidden layers
    model.add(Dense(65, activation='relu'))
    model.add(Dense(49, activation='relu'))
    model.add(Dense(14, activation='relu'))

    # Add the output layer with 1 unit and sigmoid activation function for binary
        classification
    model.add(Dense(units=1, activation='sigmoid'))

    # Compile the model with BinaryFocalCrossentropy loss function and Nadam optimizer
        with the specified learning rate
    optimizer = Nadam(learning_rate=0.005)
    model.compile(loss='BinaryFocalCrossentropy', optimizer=optimizer, metrics=['
        accuracy'])

    # Fit the model on the training data
    history = model.fit(x_train, y_train, sample_weight=w_train, epochs=10, batch_size
        =128, verbose=1)

    # Calculate sensitivity on the test set
    df_test['decision_value'] = model.predict(dset_test[0])
    sensitivity = round(sensitivity_NN(df_test)[0], 2)

    return model, sensitivity
```

Listing 1: Final Neural Network Model

An example of the final model performance can be seen in Figure 8. The sensitivity achieved with the final model is $2.767^{+0.026}_{-0.014}$. This is 48.5% increase in sensitivity compared to the cut-based model.

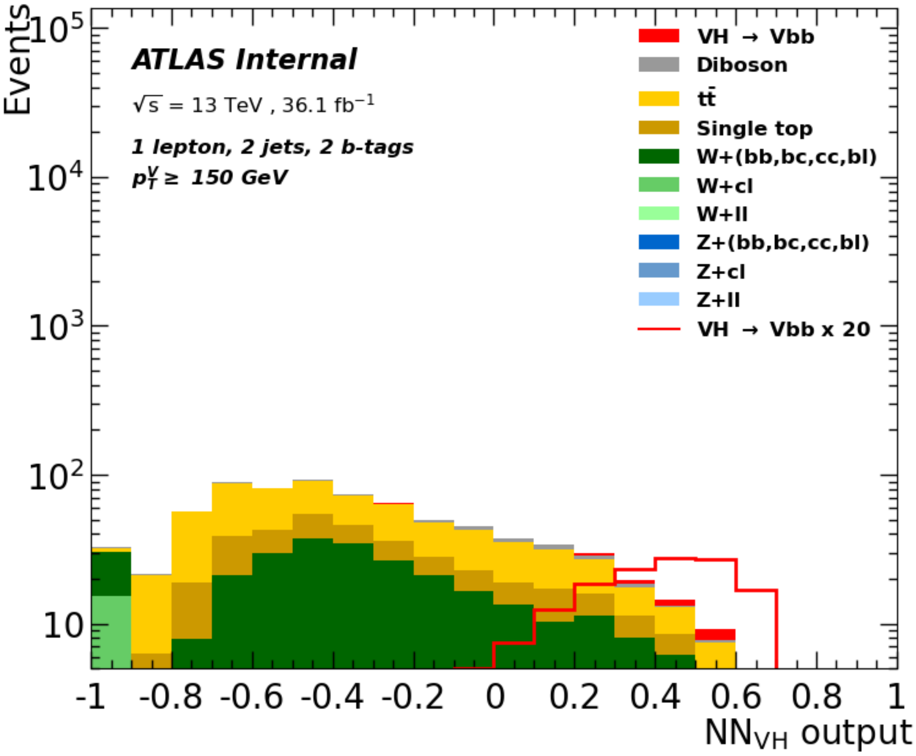


Figure 8: Histogram of NN output score after selecting signal events.

It is important to note that there is some training uncertainty in this model. Training uncertainty is the concept that the model, when being retrained under the exact same conditions, can have a varied performance. This happens due to the random elements present in network training such as random initialisation of the weights, and the random order the data samples are fed into the network during training. This can create variable final loss and accuracy in training leading to a variable sensitivity in each iteration. The plot on Figure 9 is a histogram of the sensitivities achieved after the model was reinitialised and retrained two hundred times. It achieved a $2.767^{+0.026}_{-0.014}$ in sensitivity with 95% confidence interval. The 95% confidence interval is calculated from bootstrapping the results into size of ten, and repeatedly resampling the original dataset with replacement. From these bootstrap samples, the mean is calculated for each iteration, resulting in a distribution of sample means. The 95% confidence interval is then determined by finding the range that encompasses the central 95% of these sample means, providing an estimate of the variability of the mean with respect to the original dataset.

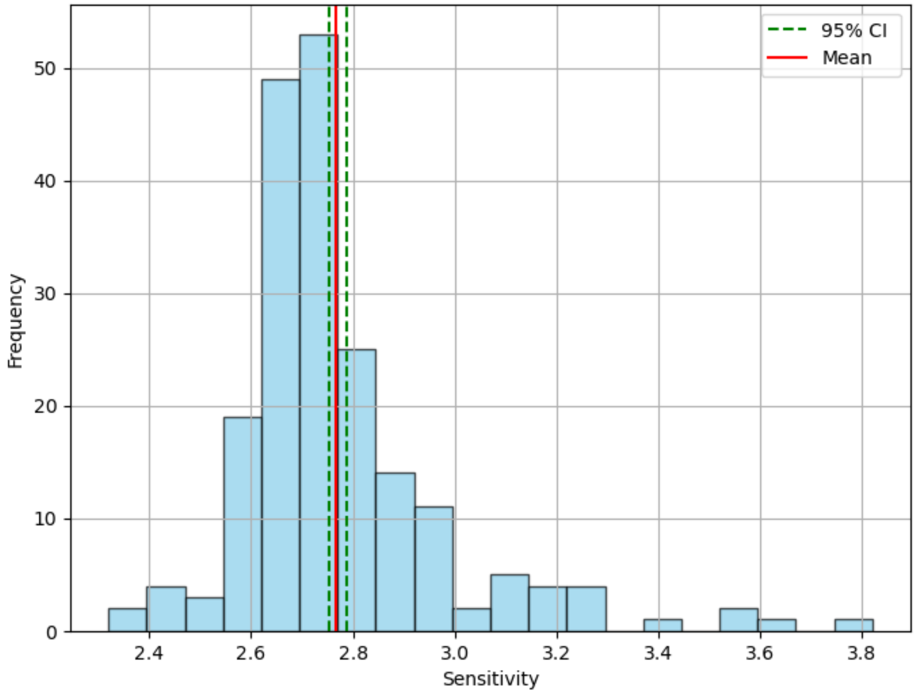


Figure 9: Histogram showing the distribution of sensitivity values with a 95% confidence interval and the mean sensitivity. The confidence interval CI is obtained from bootstrap with replacement with 10 samples.

3.3.1 Parameter Importance

To explore the significance of removing parameters, Figure 10 illustrates their average sensitivities. Parameters are removed in order of their least contribution to the variance, with those contributing less in the covariance matrix removed first. The order of contribution to the covariance matrix can be seen in Figure 11. This process continues until only the parameter contributing the most to the variance, mBB, remains.

The average sensitivity for each number of parameters is computed due to fluctuations in observed in sensitivities across iterations, as indicated by the error bars. The sensitivity keeps above 2.2 until keeping only two parameters. This is still a significant improvement over the cut-based analysis of about 18%. The model is robust to decreasing input parameters.

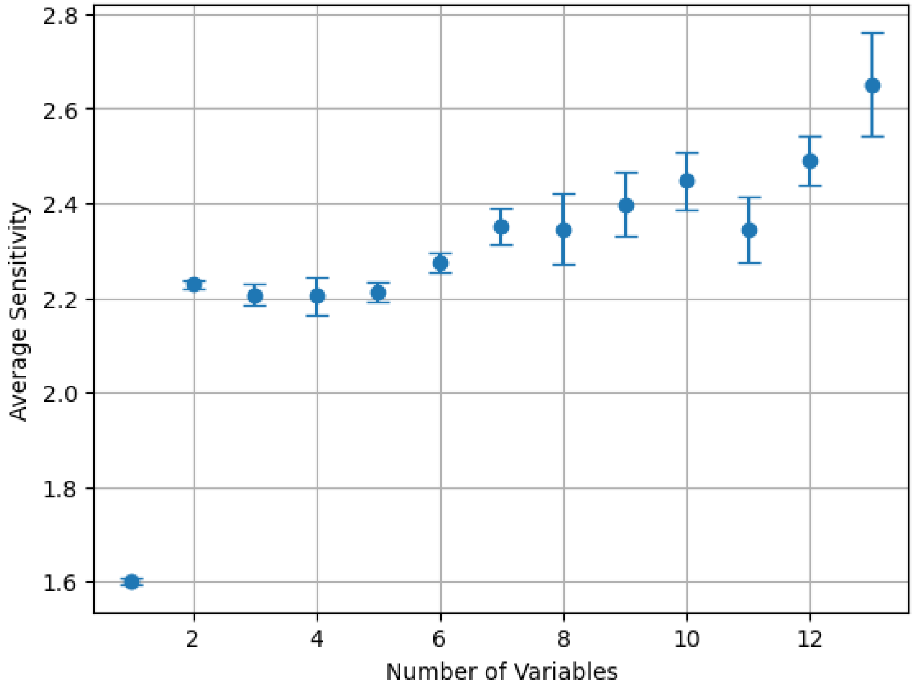


Figure 10: Plot of the average sensitivity (after 5 iterations) achieved after removing parameters iteratively.

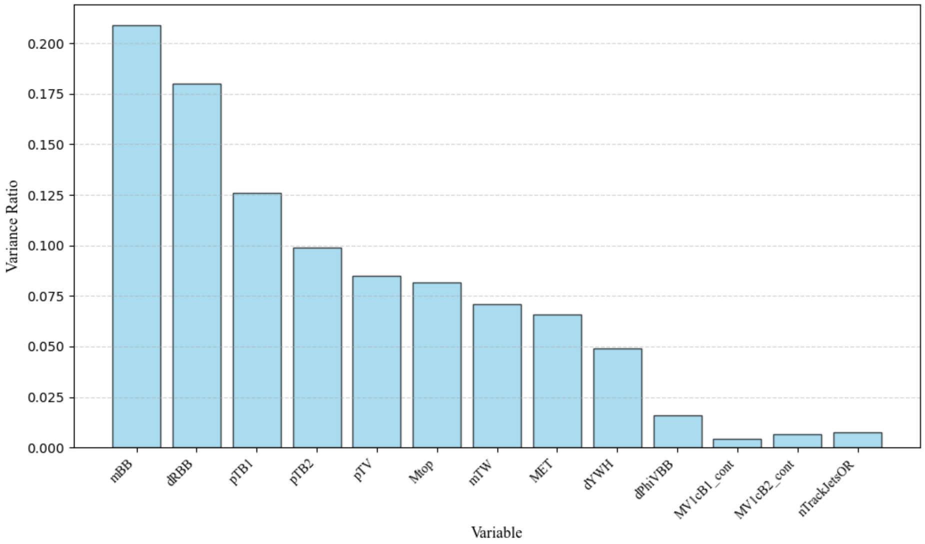


Figure 11: Variance ratio for every variable to assess which variable contributes the most to the variance.

3.3.2 Sufficient Training Data?

The trade-off between identifying general trends and overfitting is crucial in model training. Over training on the same data can lead a model to strictly learn that data, potentially hindering its ability to generalize to new data. This phenomenon is evident in Figure 12, where the sensitivity does not increase linearly as expected. The sensitivity starts to plateau at a value of about 50,000 data points. It can be seen that beyond that point, further training does not significantly improve sensitivity. This is particularly important in the context of LHC b-tagging where millions of data points are used to train models. At such scales, computing power becomes a real consideration in building algorithms. Having a model that performs as well on less training data can save a lot a time. This might behave differently if there were more training data.

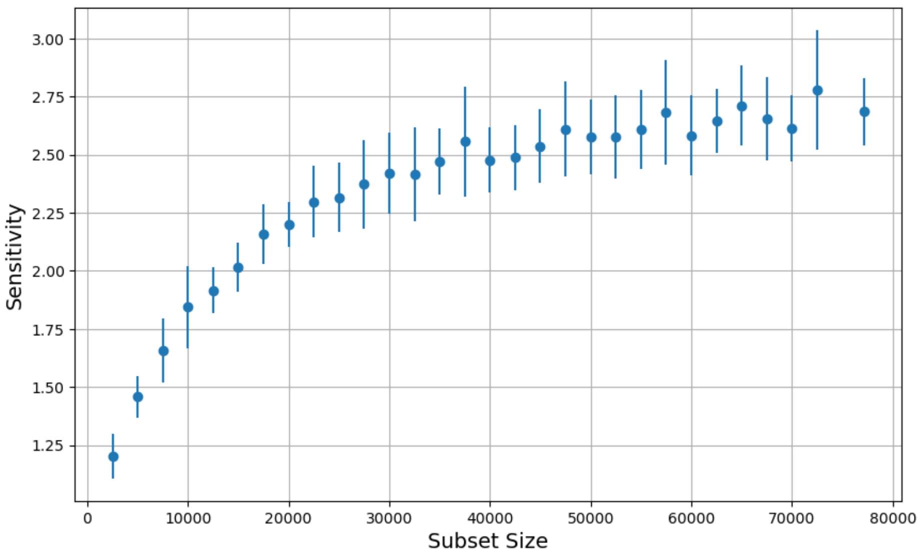


Figure 12: Sensitivity achieved for different sizes in the training dataset. The error bars come from sensitivity being measured 20 times.

3.3.3 NN sensitivity

Out of the 110,252 data points used in both the cut-based analysis and to test the sensitivity of the neural network, the cut-based analysis selected 52,591 signal events. The NN selects a an amount of signal events depending on the threshold probability associated with the event. When taking all the events regardless of probability, the NN has 52,591 signal events. These are progressively filtered out when the threshold probability increases. Figure 13 shows a plot of the ratio of the number of matching events in both analysis divided by the number of event selected in the cut-based analysis. This number is progressively decreasing due to the NN selection decreasing with increase threshold probability. The cut-based analysis and the NN select the same number of events (52,591) but with only just below 50% overlap in the lowest output region. In the high output region ($p \geq 0.7$) there is about 35% overlap between both datasets. The data is averaged out from ten iteration of the neural network. The error bars are given by the standard deviation. There is a large variability in overlap in the high NN output region.

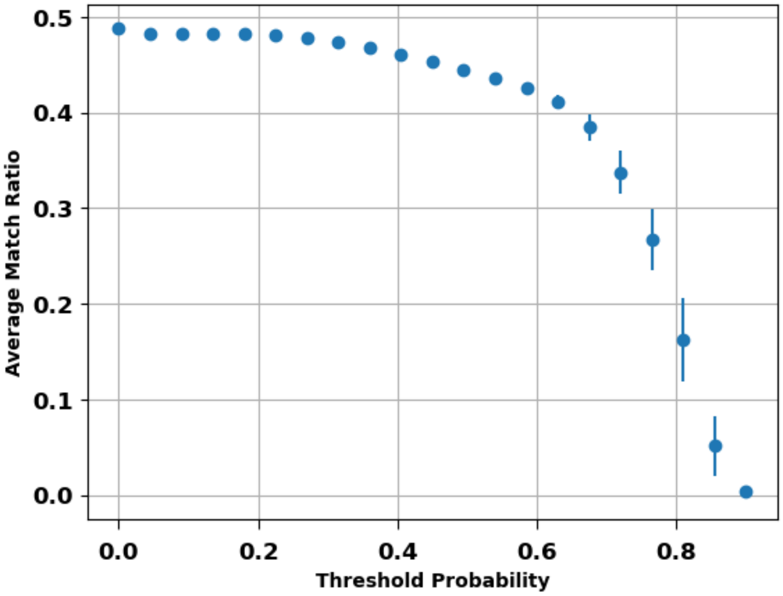


Figure 13: Average match ratio plotted against threshold probability, demonstrating the relationship between the probability threshold used for event selection in a neural network model and the resulting match ratio when compared to a cut-based approach.

4 Conclusion

This report presents a comprehensive analysis of b-quark tagging for Higgs boson detection, employing both cut-based and ML approaches. The cut-based analysis provided a baseline sensitivity of 1.863, achieved through selection of relevant parameter and application of appropriate cuts. This analysis served as a benchmark for evaluating the performance of the ML model.

The ML algorithm, a fully connected feed forward NN, shows superior performance with a sensitivity of $2.767^{+0.026}_{-0.014}$ in NN output score, representing a significant improvement of 48.5% over the cut-based analysis. Additional analysis revealed that the model remained robust even after reducing the number of input parameters. Training data sufficiency was also investigated demonstrating the beyond a certain point, increasing the size of the training dataset did not lead to significant improvement.

Overall, the results presented in this report underscore the effectiveness of ML techniques in b-quark tagging for Higgs boson approaches.

References

[1] Georges Aad et al. “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC”. In: *Phys. Lett. B* 716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[2] Serguei Chatrchyan et al. “Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC”. In: *Phys. Lett. B* 716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.