UNIVERSITY COLLEGE LONDON

DEPARTMENT OF PHYSICS AND ASTRONOMY

PHAS0056

Student Number: 18076811

Identifying Collisions Containing Higgs Bosons at the Large Hadron Collider

Identifying Collisions Containing Higgs Bosons at the Large Hadron Collider

Student Number: 18076811

***Abstract*—Through use of a neural network trained to distinguish between signal events associated with $H \rightarrow \bar{b}b$ decay and background events, we were able to achieve a final mean sensitivity of $S_\mu = 2.59 \pm 0.06$, resulting in a 35% improvement upon a sensitivity of 1.91 associated with a traditional cut-based approach of signal and background event separation, and an 80% improvement compared to our raw data.**

I. Introduction

The Higgs boson was first discovered in 2012 through the decay channels [1]:

$$H \rightarrow \gamma\gamma \tag{1}$$

$$H \rightarrow ZZ \rightarrow 4l \tag{2}$$

This is despite the fact that the Higgs boson's primary decay channel is through a pair of *b*-quarks [2]:

$$H \rightarrow b\bar{b} \tag{3}$$

However, this decay channel was not observed at 5σ sensitivity until 2018 [3] due to the large amount of background events associated with similar decay paths. The dominant backgrounds arise from the production of multi-jet events, as well as hadronic decays of (i) the Z or W boson, (ii) top quark pairs, (iii) and singly produced top quarks [4]. The decay path of $Z \rightarrow \bar{b}b$ further complicates our task due to its similar mass to the Higgs[5].

Machine learning algorithms can be used to isolate events associated with the complex data produced in collisions at the LHC due to their ability to easily analyse high dimensional data.

In this paper, we shall optimise a neural network to distinguish between signal and background events. We shall compare our model's results to the traditional cut based approach for isolating signal events from background events, as well as giving an analysis of our final neural network.

II. Construction of Neural Network

A. Cut based approach

To train and test our models, we used events simulated by the ATLAS team [6]. This data file contains a series of variables associated with an event, as shown in Table I, as well if the event was associated with a Higgs decay (signal event) or not (background event).

Before constructing our neural network, a 'cut based approach' was used, where we applied cuts to our data using the functionality of Pandas data-frames [7]. The aim of this was to exclude as many background events as possible, whilst maximising the proportion of signal events. Cuts were found by defining an initial cut point (based on visual inspection of histogram plots [8] associated with each of our variables), before applying an iterative algorithm to find our optimized cut points. Each cut was then applied consecutively to our data frame, maximising our sensitivity. In Figure 1 we plot the variable m_{bb} for our uncut and optimized cut data frame. We chose the variable m_{bb} as our discriminant, as this variable provides the best separation between signal and background.

Through our cut based approach, we were able to achieve a final sensitivity of 1.91, an improvement of 34% compared to our initial data. This will act as our baseline, with the hopes that a neural network approach will be able to drastically increase upon this.

B. Neural Network - proof of concept

For our neural network approach, we need to train a model to predict if an event is a signal or background event, based on the event variables. Traditional methods for splitting our data into a training set and testing set would not be optimal for this task. Doing this would result in only a fraction of events being included in our test data, meaning our final sensitivity would not be representative of our whole data set.

To combat this, we instead split our data frame in two, defining an 'even' and 'odd' data set (based on the event ID, an arbitrary number). To calculate our final sensitivity, we created two identical models, training one of the even data set (even model) and one on the odd (odd model). We then used the models to predict the result of the opposite data set, before combining the sets of predictions, forming a final data frame consisting of all events along with a class prediction for each. This means our final sensitivity is based on all data points, resulting in a more representative result. However, the main disadvantage of this method is we limit each model to only be training on half of our data.

We created a simple pair of models [9][10], which were trained on a selection of the available variables. All variables were normalized before being inputted [11] to increase the rate of convergence for our models [12]. Our models had a final sensitivity of 2.22, with the plot shown in Figure 2. This represents a 16% increase in sensitivity when compared to our cut based approach, already a drastic increase.

C. Model optimization

Optimizing our neural network consisted of two main parts:

TABLE I: Variables available for model training, including their descriptions and their labels within our data set

Variable	Description	Label
n_j	Number of jets in the event	nJ
n_Tags	Number of b-tagged jets in the event	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 b-tagged jet with the highest p_T	pTB1
$p_T^B 2$	Reconstructed transverse momentum of the b-tagged jet with the second highest p_T	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
$MV1_{cont}^{B1}$	The classification output of whether the leading jet is a b or not	MV1cB1_cont
$MV1_{cont}^{B2}$	The classification output of whether the sub-leading jet is a b or not	MV1cB2_cont
n_{cont}^{Jets}	Number of additional jets found in the event	nTrackJetsOR

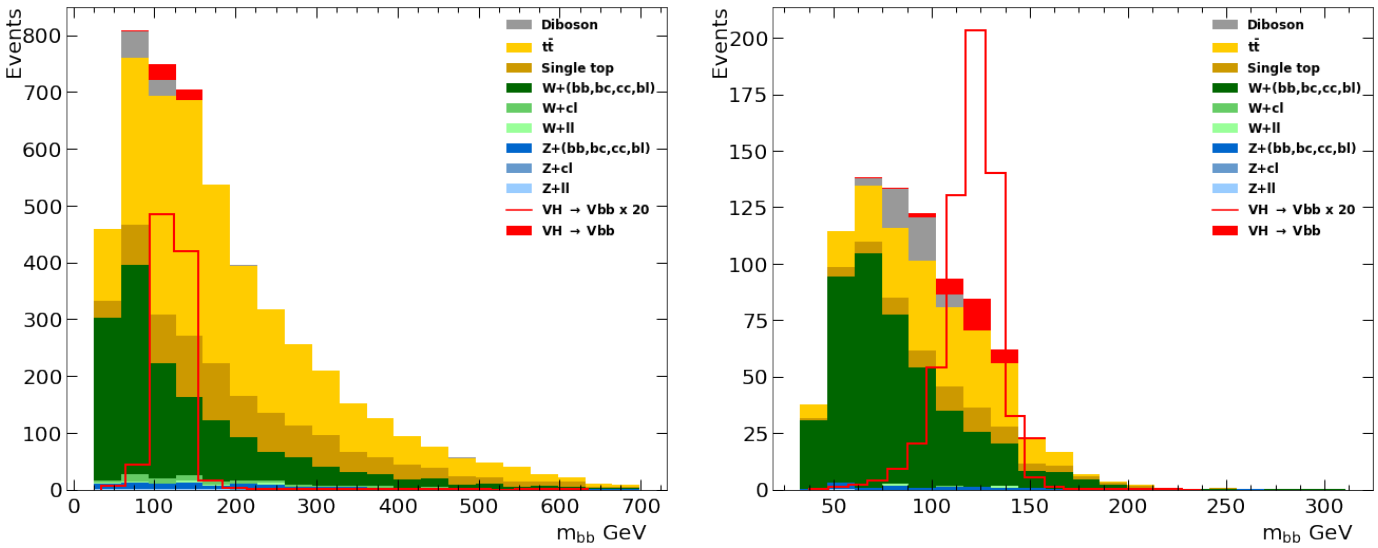

Fig. 1: Figure showing the variable plot of m_{bb} for our uncut data frame (left, sensitivity = 1.43) and our data frame after our optimized cuts have been applied (right, sensitivity = 1.91)

- Pre-processing of data
- Optimization of model hyper parameters

Our first and most obvious step for data pre-processing is the selection of variables to use to train our models. This was done by extracting groups of variables from our data frame into NumPy arrays [13] for ease of manipulation. Figure 3 shows our results, with Table II detailing which variables were included within each test. From this plot, we see clearly that the variables defined under '*useful_vars*' maximise our models' sensitivity.

Other attempts to increase sensitivity through pre-processing of data included attempting to run our neural network on a data frame after our optimized cuts had been applied, as well as through principle component analysis (PCA).

Unsurprisingly, training our models on our cut data frame did not improve the sensitivity. This is most likely due to the huge amount of data lost while applying cuts.
PCA lowers the dimensionality of our data by extracting dominant patterns within our data as a set of orthogonal variables, known as principle components [14]. For high dimensional data, this can increase training speed by the lowering of dimensionality. Principle component analysis can also help prevent the issue of over fitting [15]. The results for our PCA are shown in Figure 4. We see that for this case, PCA does not increase our models' sensitivity.

Various methods exist for optimizing the hyper-parameters for our models. Statistical methods such as Bayesian optimization aim to maximise model performance through statistical analysis of the model phase space, with multiple

TABLE II: Table containing the variables included within each 'named' selection of variables

Selection name	Variables included
all_var	All variables
useful_vars	$\Delta R(b_1 b_2)$, p_T^V, m_{BB}, m_{top}, m_T^W, E_T^{Miss}, $dY(W,H)$, $d\phi(W,H)$, $MV1_{cont}^{B1}$, $MV1_{cont}^{B2}$
useful_vars_plus	$\Delta R(b_1 b_2)$, p_T^B1,p_T^B2,p_T^V, m_{BB}, m_{top}, m_T^W, E_T^{Miss}, $dY(W,H)$, $d\phi(W,H)$, $MV1_{cont}^{B1}$, $MV1_{cont}^{B2}$
dem_variables	$\Delta R(b_1 b_2)$, m_{BB}, E_T^{Miss}, m_{top}, p_T^V

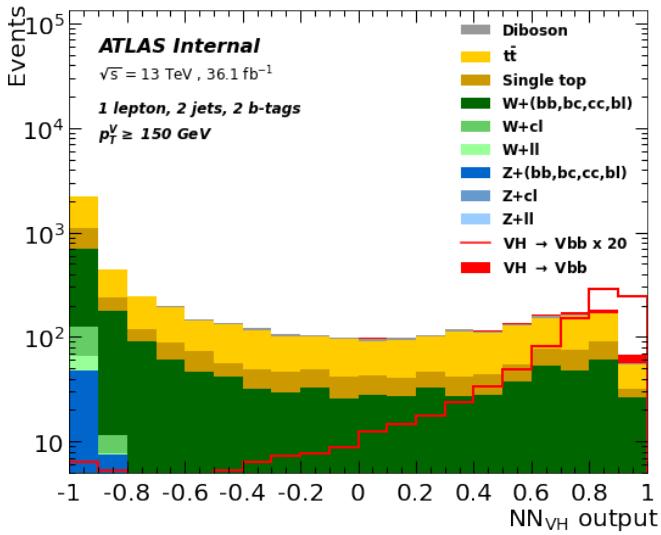

Fig. 2: Figure showing the sensitivity plot for our first neural network, with a final sensitivity of 2.22

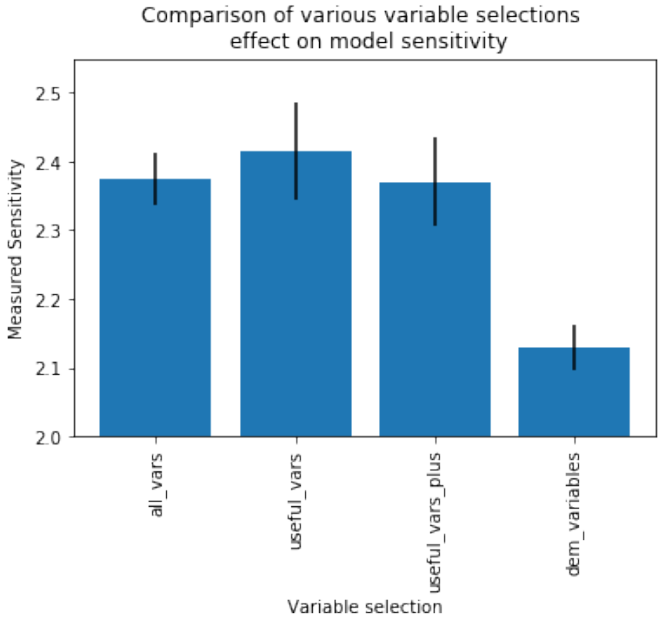

Fig. 3: Figure showing the average sensitivity for models with different input variables

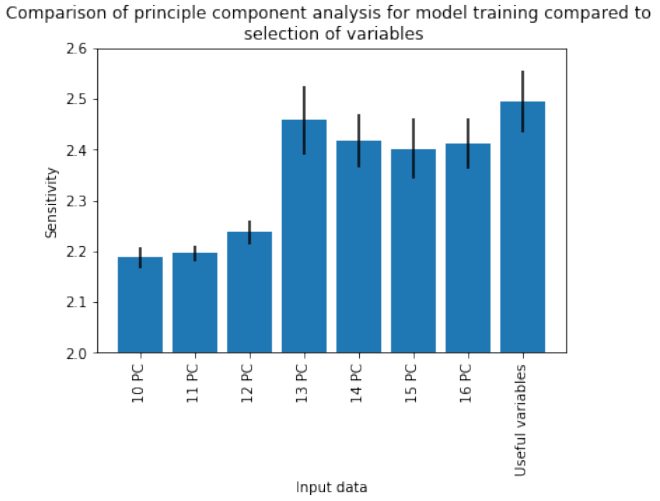

Fig. 4: Figure showing sensitivity after PCA for different number of principle components compared to our normal 'useful_vars' selection of variables

python [16] packages existing to help perform this algorithm [17][18]. However, by using a package to optimize our models we would get little insight them. We instead optimized each hyper-parameter in turn with an iterative search. This was chosen rather than a higher dimensional grid search due to computational limits. Although a gird search or Bayesian optimization may have resulted in a slightly higher final sensitivity, we noted that even our first simple neural network was able to achieve an improvement of 16% compared to our optimized cut based approach. With this level of improvement from a basic model pair, we believe that a simple iterative search can achieve near the same results as these more complex and computationally expensive methods.

For our iterative search method, we had to decide an order of our hyper-parameters to optimize. Our final hyper-parameter test order was formed by first deciding the form of all hidden layers and input layer, before moving onto compile time hyper-parameters associated with the models, followed by model fitting parameters. The final order of hyper-parameter optimization was as follows:

- **Layer Count**: The number of hidden layers in our models. A large number of other hyper-parameters are greatly dependent on this (neurons per layer, activation functions per layer, the inclusion of drop out and batch normalization layers) and so it is our most important hyper-parameter to decide. A test was run allowing for a range of different layer counts with neuron count per layer of $n = 16$ for various training epochs. Our final results, shown in Figure 5, show that peak sensitivity occurs at $L = 2$.
- **Neuron Count**: The number of neurons in each hidden

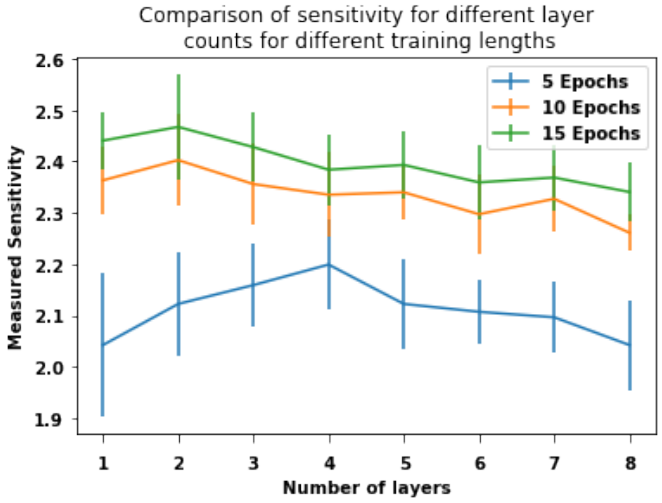

Fig. 5: Measured sensitivity for varying layer counts for a selection of different training epochs. Each layer contains neuron count $L_n = 16$

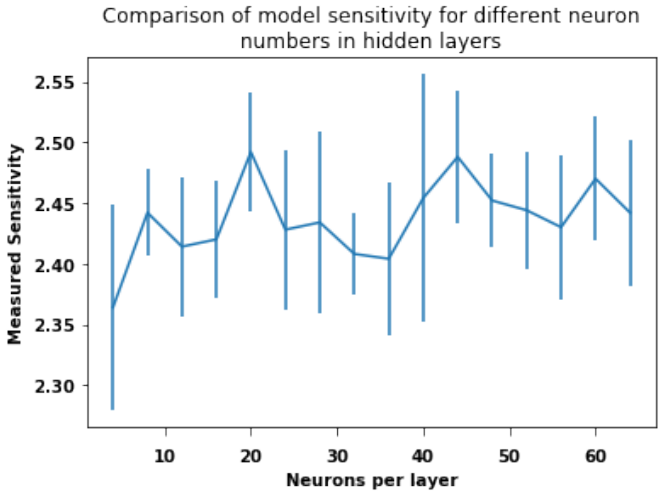

Fig. 6: Measured sensitivity for varying neuron counts in both layers

layer. This hyper-parameter is greatly linked with the complexity of a model. Finding an optimum value for this hyper-parameter is important as values that are too high can decrease model accuracy by adding in too much complexity, resulting in longer training times. Our final results, shown in Figure 6, show a slight increase in average sensitivity until the point $n = 20$, after this point, statistical noise makes it more difficult to see a clear result, however the plot does appear to be tending towards a flat sensitivity. Using too many neurons increases the models' complexity too much, resulting in a higher variance, and so we decided to use $n = 20$ neurons per layer.

- **Activation Function**: The activation function allows for non-linearity within our models. In our models we have a total of 4 activation functions: the input activation

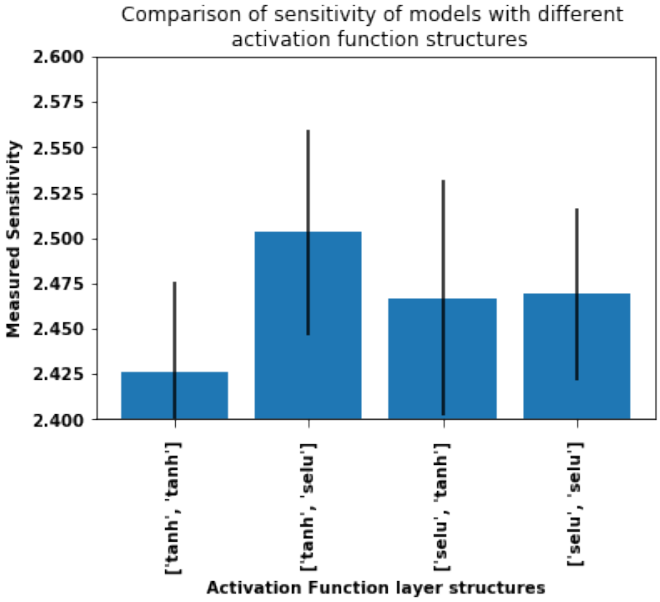

Fig. 7: Measured accuracy for the best structure of best two activation functions

function, the activation functions for each of our two hidden layers, and the activation function in our output layer. This final activation function shall be kept as 'sigmoid' for now, to allow for compatibility with the loss function we have been using so far, 'binary cross-entropy'.

We first tested all possible activation functions in our hidden layers, setting each hidden layer to use the same activation function, with results shown in Figure A.1. The best four activation functions were then tested for further iterations, shown in Figure A.2, with 'tanh' and 'selu' found to be those associated with highest sensitivity. We then took these activation functions, and ran the 4 possible combinations of them within our hidden layers, to find the final activation function structure for our hidden layers. As shown in Figure 7, we see that our optimum set up has 'tanh' and 'selu' as the activation functions for our first and second hidden layers respectively.

We next moved on to optimizing the activation function for our input layer. As before, a preliminary test was run on all activation functions, seen in Figure A.3, before the best selection was tested for further iterations. This test showed that the 'linear' activation function - equivalent to no activation function - achieved the highest sensitivity, as shown in Figure 8.

- **Dropout layers**: Dropout layers result in a fraction of neurons being dropped while training, preventing the neuron units from co-adapting too much [19], with the aim of preventing over fitting. For our models, we attempted a drop out layer after each dense hidden layer, as seen in Figure A.4 and Figure A.5, we clearly see that inclusion of drop out layers have a negative effect on our models' sensitivity. This is likely due to the relatively

small number of neurons present in these models. As a result, we shall not be utilising drop out layers.

- **Batch Normalization**: Batch normalization is useful for increasing the rate of convergence, with less ability to directly increase model accuracy [20]. Our results, shown in Figure A.6 show that our models' sensitivity is largely independent of the inclusion of batch normalization layers, with no increase in sensitivity observed. We shall not include any batch normalization layers within our final models.

- **Optimizer**: Optimization functions help speed up the rate of convergence for our models. We tested all optimizer functions available in TensorFlow with results shown in Figure A.7, before running further tests on the best of these. Our final results show that 'adamax' - a modified version of 'adam' - had the highest sensitivity after 15 epochs, as seen in Figure 9.

- **Loss Function**: Our loss function defines the loss value calculated after each batch while training, the value we wish to minimize. The task for our models is to distinguish between 2 possibilities: signal events, or background events. As a result, it was likely that the binary cross-entropy loss function, one designed for performing binary categorization, would be most suitable. When tested against all other default loss functions within TensorFlow, this result was clear, with results shown in Figure A.8 and Figure 10.
  Further tests were run comparing binary cross entropy to a set of custom loss functions (Equation A.1, A.2, A.3 plotted in Figure A.9). These functions all worked by calculating the Δ value ($|y_{True} - y_{Predicted}|$), and then returning high loss values for $\Delta > 0.5$ (An incorrect prediction), and low loss values for $\Delta < 0.5$ (correct model prediction). Preliminary results are shown in Figure A.10, with our best loss function found to be:

$$L_{batch} = \sum_{n} \left[\frac{1.01}{1.01 - \Delta_n} - 1 \right] \tag{4}$$

  As shown in Figure 11.

- **Batch Size**: The number of data points for each batch before new model weights are calculated. Smaller values can decrease overall training time by decreasing the number of training epochs required, while high values result in quicker training time per epoch. Un-optimized values can also result in the model getting stuck in local minima. Our results are shown in Figure 12, with a batch size of 32 achieving the best sensitivity.

- **Epochs**: With our final models' hyper-parameters chosen, the number of training epochs was chosen. This was tested last due to its high dependence on other model hyper-parameters. Results are shown in Figure 13, with full model convergence occurring at approximately 20 training epochs.

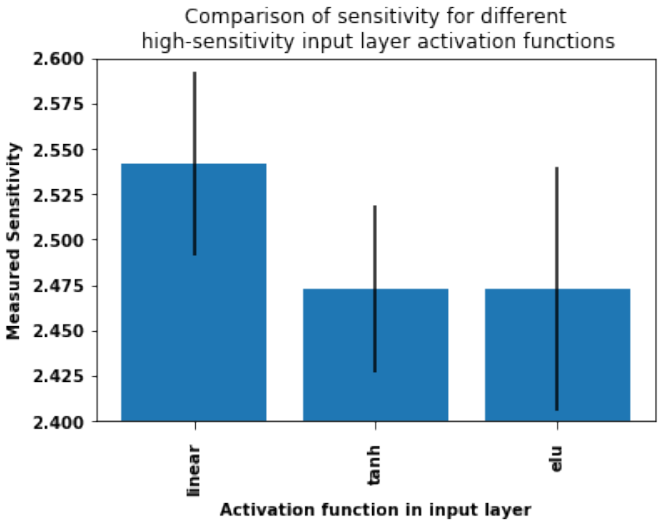

Fig. 8: Plot showing models' sensitivity for different activation functions in our input layer

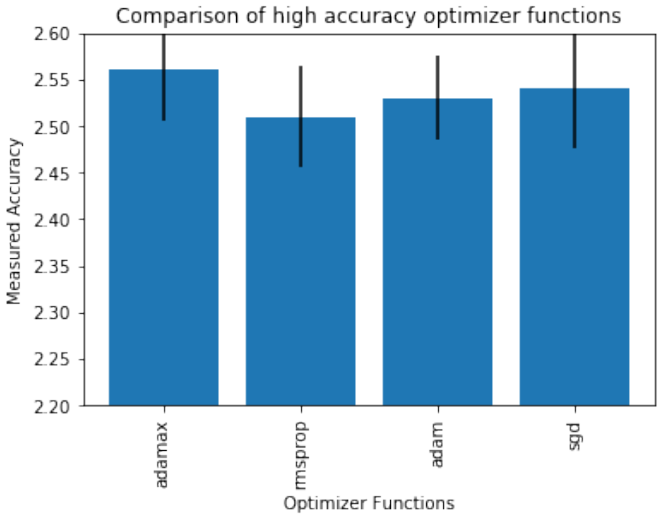

Fig. 9: Measured sensitivity for best set of optimizer functions available in TensorFlow

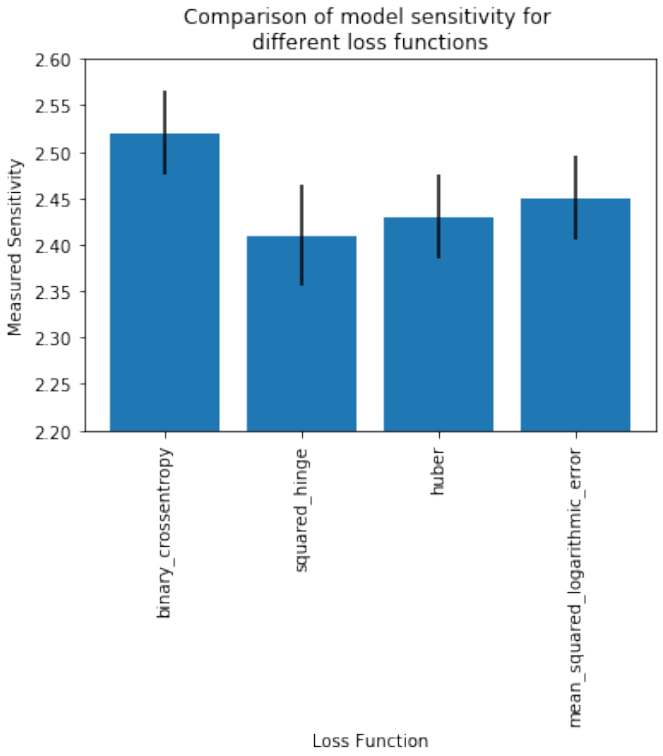

Fig. 10: Measured sensitivity for all possible loss functions available in TensorFlow

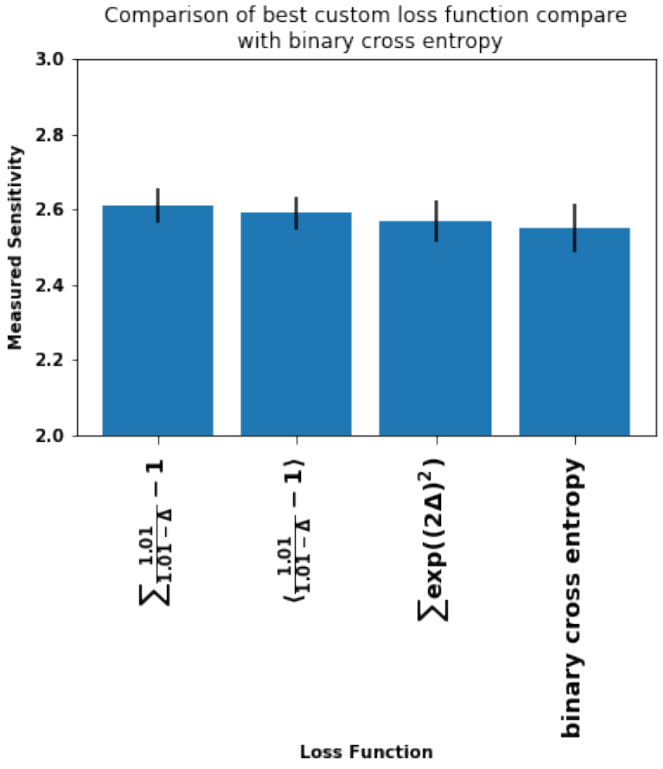

Fig. 11: Plot showing models' sensitivity for best selection of custom loss functions, compared to binary cross entropy

Layer Type	**Parameters**
Dense (Input)	10 Neurons, Linear Activation
Dense	20 Neurons, tanh Activation
Dense	20 Neurons, Hyperbolic Tangent Activation
Dense (Output)	1 Neuron, Sigmoid Activation
Optimizer Function: Adam Loss Function: $\sum \frac{1.01}{1.01-\Delta} - 1$ Training Epochs: 20 Batch Size: 32	
Total params: 771 Trainable params: 771 Non-trainable params: 0	

TABLE III: Table detailing all hyper-parameters for our final models

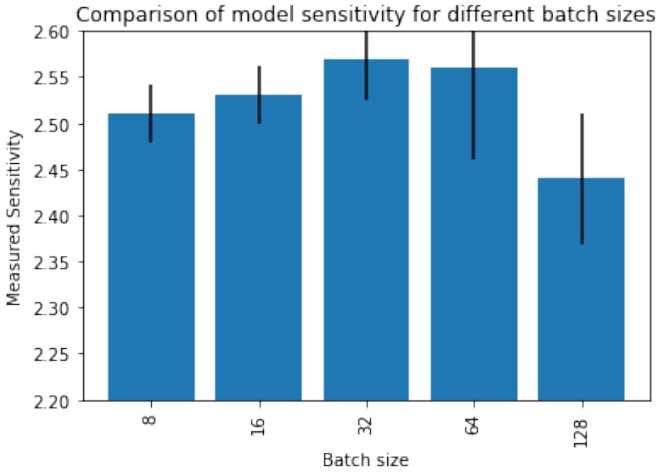

Fig. 12: Measured sensitivity as a function of batch size

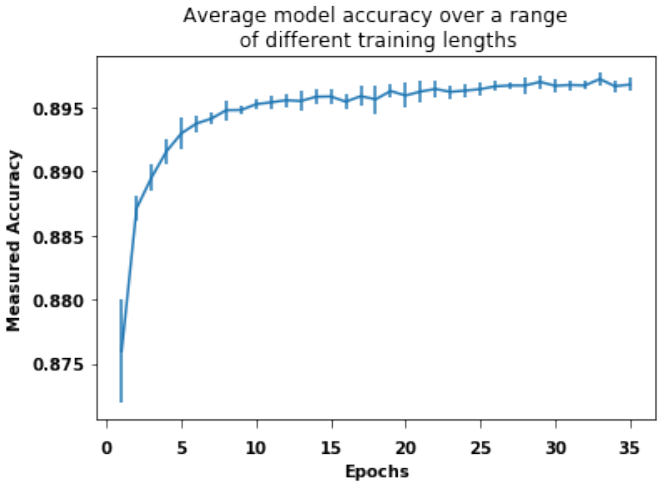

Fig. 13: Measured sensitivity as a function of training epochs

III. Analysis of Final Model

Our final model structure can be seen in Table III, with the final plotted result shown in Figure 14. To ensure an accurate

final sensitivity, we trained and tested our models 100 times, achieving a final average sensitivity of $S_\mu = 2.59$, with variance $\sigma^2 = 0.004$. This compares to a sensitivity of 1.43 for our initial data set, and 1.91 for our cut based approach, meaning a final improvement of roughly 35% was achieved with our neural network when compared to the cut based approach.

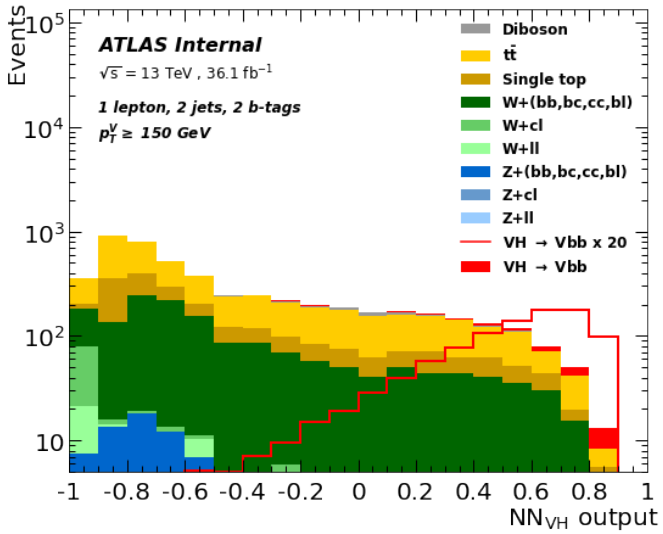

Fig. 14: Neural network output for our final models, achieving a mean sensitivity of $S_\mu = 2.59, \sigma^2 = 0.004$ after 100 iterations

A. Analysis of ratio of signal and background events within data set

The data sets used to train our models consists of roughly 60% signal events and 40% background events. After creating our final models, we analysed these data sets, running tests to see if this ratio of signal events to background events was optimal for model training. We took fractional random samples of signal and background events from our data sets to form new data sets with custom ratios of these events, with the results shown in Figure 15. Interestingly, we see that there appears to be a peak when around 70% of signal events are used. This fraction of signal events results in the number of signal events being closer to the number of background events. As a result, we hypothesised that our models would train best when given an equal number of signal and background events.

We decided to test this further by keeping the fraction of background events fixed at 100%, whilst varying the fraction of signal events. The result of this test can be seen in Figure 16. In this plot, we see that a slight peak occurs when using roughly 70% of our signal events. This fraction of signal events results in the data set containing approximately 53% signal events and 47% background events. This supports our hypothesis that our models are trained most optimally when there is an equal number of signal and background events present in our data set. However, we would expect this graph to peak for a 60% fraction of signal events, as this would result in a 50-50 split of signal and back ground events. This could be due to training noise or randomly selecting less useful, further tests were needed to support our hypothesis.

One issue of only taking a sample of our signal events is that we needlessly throw away useful data. As a solution to this, we instead attempted to up-sample our data set, by taking a sample of our background and events and adding it into our data set. The result of this is a larger data set, with some background events repeated. We tried different values for the percentage of up-sampling, with results shown in Figure 17. Similarly to when we took fractions of the signal events, the error in these plots makes it hard to draw a strong conclusion. However, there does still appear to be a peak for 55% up-sampling, the point at which the ratio of signal to background events is approximately equal.

A comparison of using our default data set compared to fractional signal events and up sampling of background events can be seen in Figure 18. We see that the difference between sensitivity for our three different data sets is small. Tests were run for 20 iterations, with all results falling within one standard deviation of each other. However, there does seem to be some improvement for our fraction signal event data set when compared to our default data set.

Our initial methods of up-sampling and taking fractions of our data sets were done by creating an initial modified data set and then running all training epochs on this data set. This method has some issues; for our fractional signal event data set, it means we completely throw out a fraction of our data. For our up sample background event data set, it means we repeatedly train on the same set of copied background events. To improve our tests, we wrote a custom training loop for our models, which created a new fractional or up sampled data set for each epoch. This allowed us to have our optimal ratio of signal and background events for each epoch, whilst reducing the aforementioned problem. Our results are shown in Figure 19.

We see that our fractional data set again has a slight advantage. This is likely because we do not over train our models by using repeated data points while training. Although the improvement compared to our default data set is small ($\Delta S_\mu = +0.06$), the consistency of this result is a strong indicator that our models train most optimally when there is an equal number of signal events and back ground events. As such, a simple way to improve the models' sensitivity would be for a larger number of background event data points to be generated and added to our data sets.

B. Importance of each variable

When choosing the best variables to use to train our models, we chose a selection of 10 of the 15 available variables. We attempted removing each of these 10 individual variables, in an attempt to see which variables are most important for our models. Figure 20 shows the results of this test. Clearly, we see that the variable m_{BB} is the most important, with a decrease in average sensitivity of 0.35, or approximately 15%. This is not surprising, we used this variable as our discriminant during our cut based approach as it provided the best separation between signal and background events.

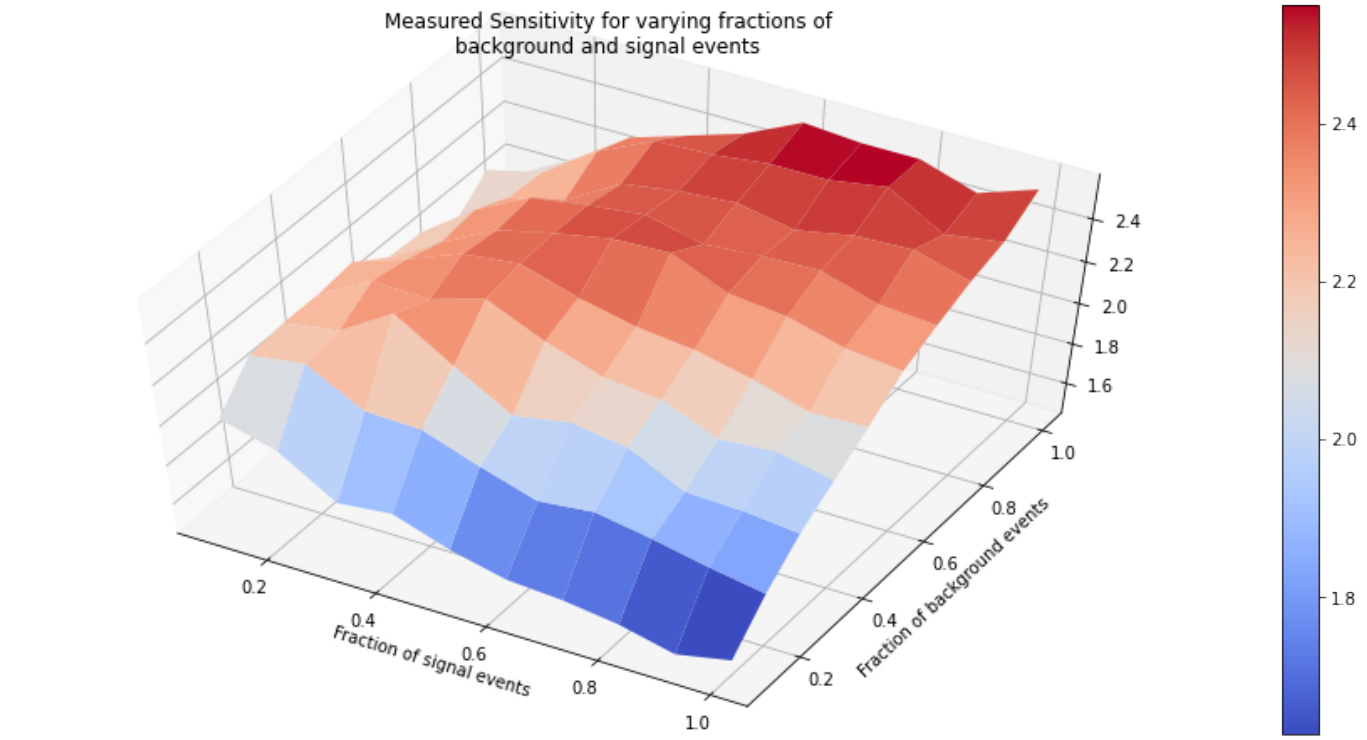

Fig. 15: Measured sensitivity for various fractions of signal and background event data points

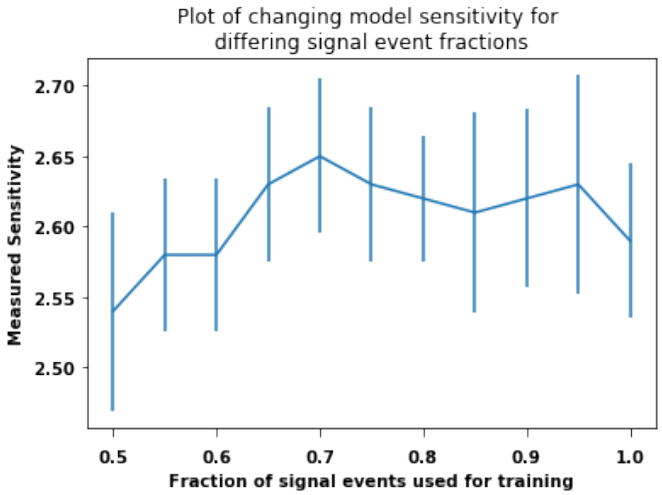

Fig. 16: Measured sensitivity for various fractions of signal events

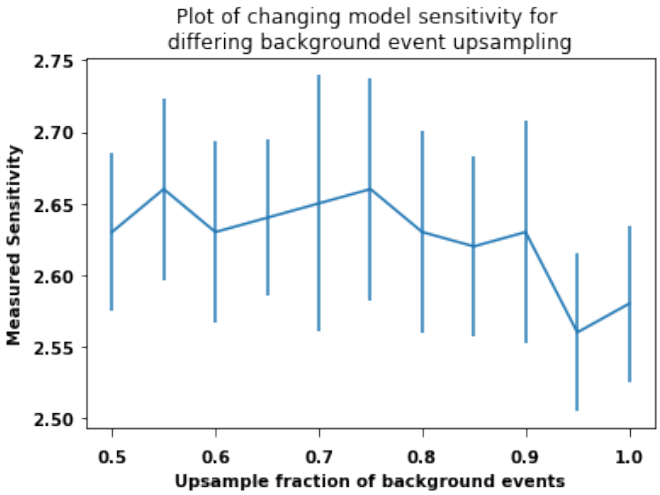

Fig. 17: Measured sensitivity for various fractions of background event up-sampling

Another result is the relatively large sensitivity drop when either of the variables $MV1^{B1}_{cont}$ or $MV1^{B2}_{cont}$. These variables represent the classification results as to whether the leading or sub-leading jets respectively are 'b' jets or not. As we are attempting to isolate the decomposition of the Higgs into a pair of b quarks, it is not surprising that information relating to whether the jets are associated with b-quarks are an important set of variables for our models.

We next decided to see how the remove of two variables effected our models' sensitivity. Our motivation for this came from an error when initially starting this project, in which we mistakenly excluded both $MV1^{B1}_{cont}$ and $MV1^{B2}_{cont}$. When we introduced these variables into our models, we saw a sudden increase in sensitivity of around 0.2, or 10%. The results of removing two variables at once can be seen in Figure 21.

Here we see the predicted result, with at least one of the two variables $MV1^{B1}_{cont}$ and $MV1^{B2}_{cont}$ being important to our models' sensitivity. Unsurprisingly, the variable m_{BB} is also highly important, with sensitivity greatly decreased for all two variable sets removed that included it. An interesting result is

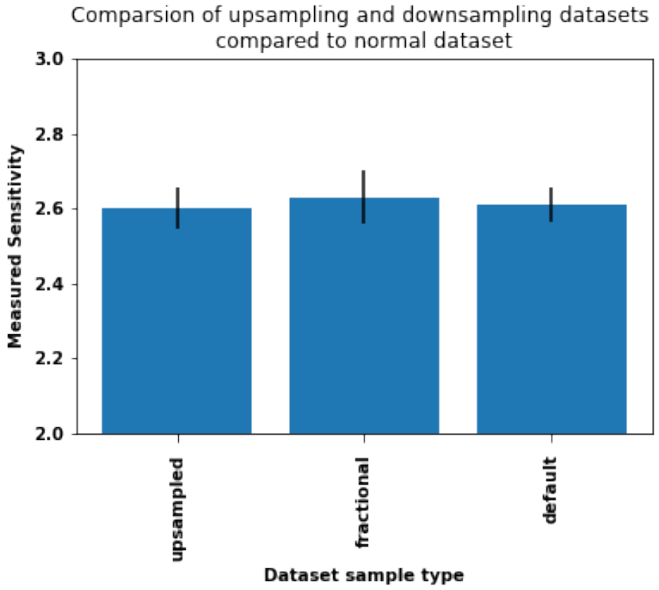

Fig. 18: Comparison of sensitivity for default data set, compared to a fractional signal event data set and an up sampled background event data set. Final results were up-sampled background: 2.60 ± 0.05, fractional signal: 2.63 ± 0.07, default: 2.61 ± 0.04

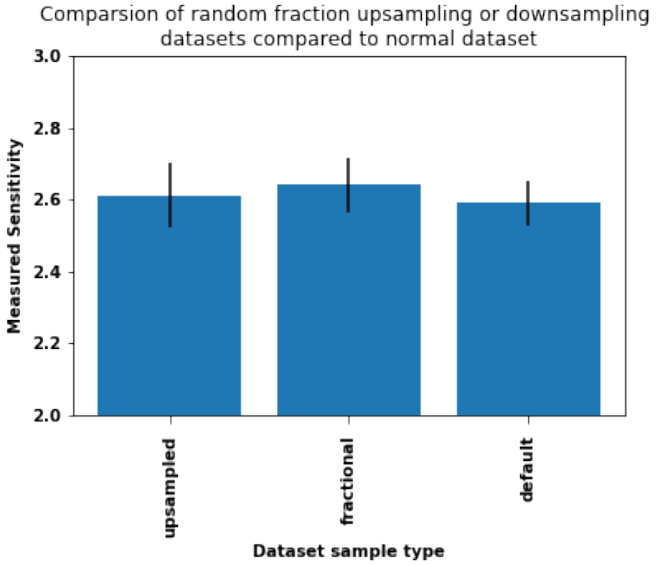

Fig. 19: Comparison of sensitivity for default data set, compared to a taking a new fractional signal event data set or up sampled background event data set for each training epoch. Final results were up-sampled background: 2.61 ± 0.09, fractional signal: 2.65 ± 0.09, default: 2.59 ± 0.06

the hugely significant decrease in sensitivity when both m_{BB} and $\Delta R(b_1, b_2)$ are removed. When doing this, our sensitivity drops to 1.3, lower than our initial data frame.

Further tests helped resolve this surprising issue. When we plot the matrix of correlation coefficients for our signal and background events, as in Figure A.11, we see that there is a

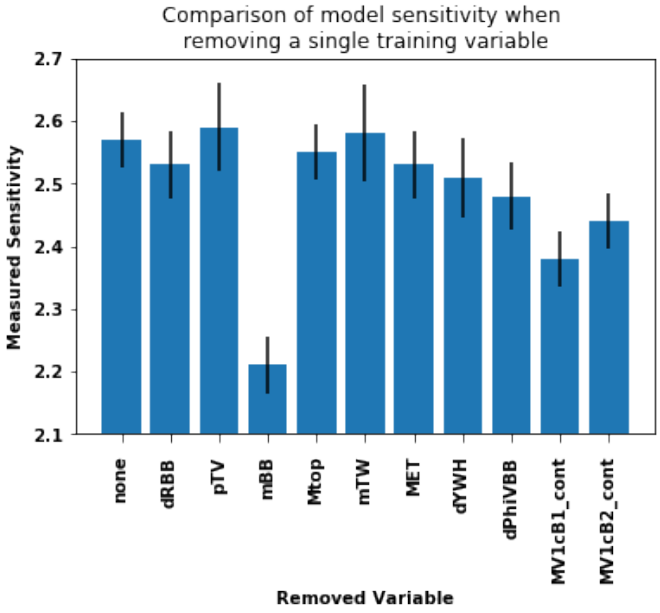

Fig. 20: Comparison of sensitivity after removing a single variable from the models' input

very strong correlation between $MV1c^{B1}_{cont}$ and $MV1c^{B2}_{cont}$ of $r \approx 0.9$, this is unsurprising. In any given signal event we expect the production of a pair of b-quarks in the form of jets. If the leading jet is tagged as a b-jet, it is logical that the sub-leading jet is also a b-jet.

When we look at the correlation between $\Delta R(b_1, b_2)$ and m_{BB}, we see that they have a correlation of $r \approx 0.83$. This high correlation explains why the removal of this variable pair decreases the sensitivity by so much.

This correlation arises due to the non-zero transverse boost Higgs, resulting in a decrease in the angle between our b-tagged jets [21], and hence we have a strong correlation between these variable sets.

C. *Performance of models under distortion of data*

The data set used to train our models was created using computational models, and so represents theoretically 'perfect' data. However, when true measurements are taken experimentally in the real world, there is always the chance for error. The most likely source of error is some systematic error, where all values of a given variable are distorted by some constant value.

Due to this fact, it would be prudent of us to test how our models' sensitivity changes under a constant distortion of this type. We defined our distortion as a fraction of the variables mean value, adding this fractional mean to all data points. We tried this for all variables in our models, with results shown in Figure 22.

Our results show that distorting our data by $\pm 2\mu$ has an overall negligible effect on our final sensitivity ($\Delta_{max} S_\mu \approx \pm 0.08$). One of the main reasons for this independence of our sensitivity with respect to systematic variable distortion is the fact that we used 10 dimensions of data for our models, meaning

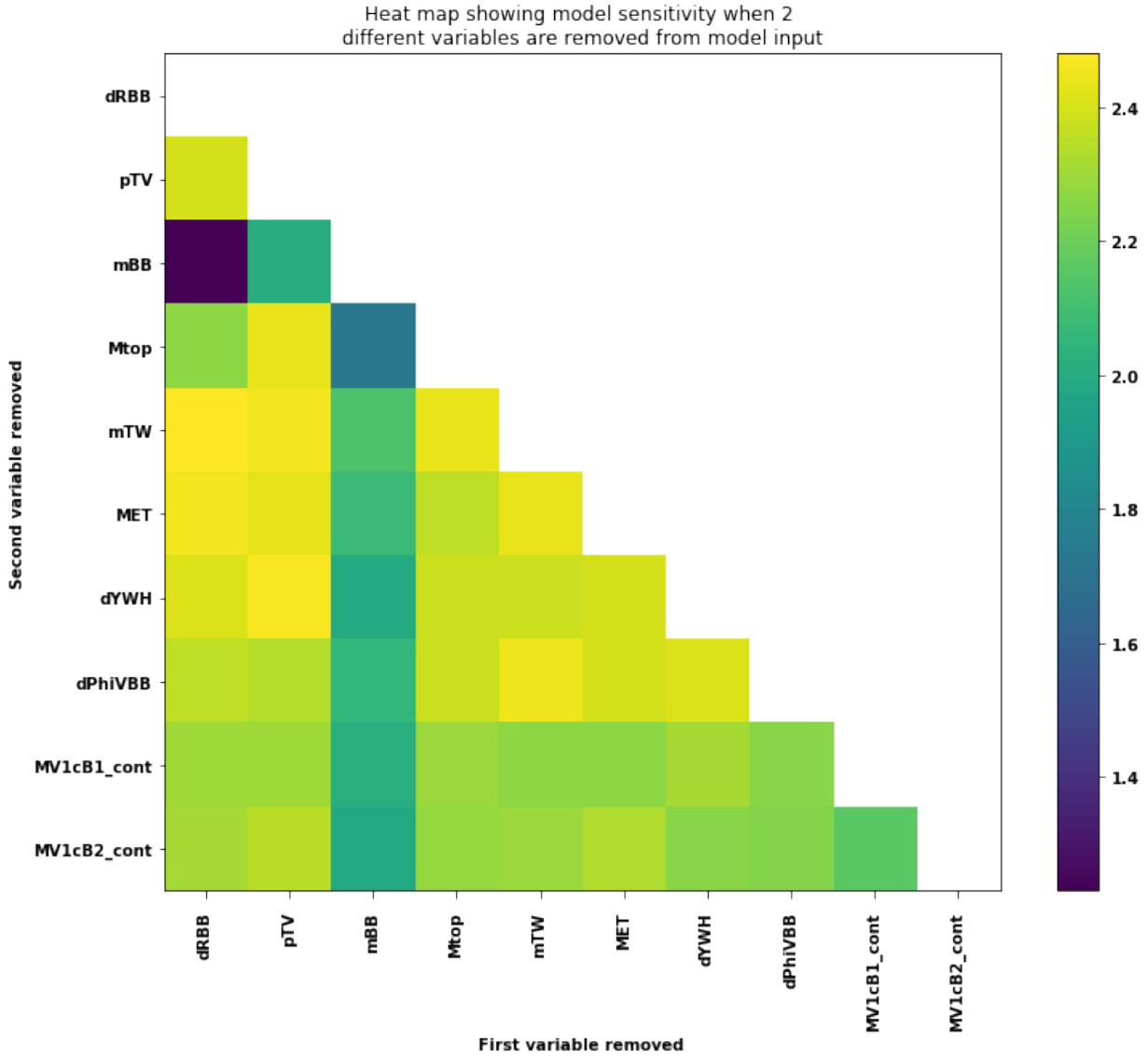

Fig. 21: Comparison of sensitivity after removing two variables from the models' input

that when any single variable is distorted, the overall effect on our data is small, and so the overall change in sensitivity is also small.

However, we see that for some variables we do see a somewhat consistent change in sensitivity under the variable distortion; mainly for large negative distortion of $\Delta R(b_1, b_2)$, and large positive distortion of $MV1^{B1}_{cont}$.

For $\Delta R(b_1, b_2)$, it is likely that a large negative distortion of this kind results in all data sitting outside of the expected range of our models. We see in Figure A.12 that the majority of our data has values of $0 \leq \Delta R(b_1, b_2) \leq 3.5$. A distortion of 2μ would result in almost all of our data to exist in the region $-2.6 \leq \Delta R(b_1, b_2) \leq 1$ as shown in Figure A.13. Our models works by assessing the relative probability that an event is likely to be a Higgs, with a high likelihood of a Higgs occurring when $0.6 \leq \Delta R(b_1, b_2) \leq 1.6$, as in Figure A.14. With the distortion of -2μ, almost all data falls outside this region, forcing our models to decrease their assumed probability that any given event is associated with a Higgs event. This results in more incorrect predictions, and so a lower sensitivity. However, we do not see this same decrease in sensitivity for a large positive distortion of this variable. The exact reason behind this is yet unknown. We hypothesise that it is due to the asymmetry of the values within this variable group, with Higgs events mostly being associated with low values of $\Delta R(b_1, b_2)$; however, this has not been tested, and so further study into this phenomena is required.

The other variable with the largest consistent change under variable distortion is $MV1^{B1}_{cont}$. We see that for large positive distortion, the sensitivity of our models increases. The majority of the values of this variable lie in the range $[0, 1]$ with a mean value of 0.87. This means that when we distort by a factor of $+2\mu$, we push all values into a region of values above 1.7. Surprisingly, this didn't have a significant detrimental effect on our sensitivity. Further research is needed into this.

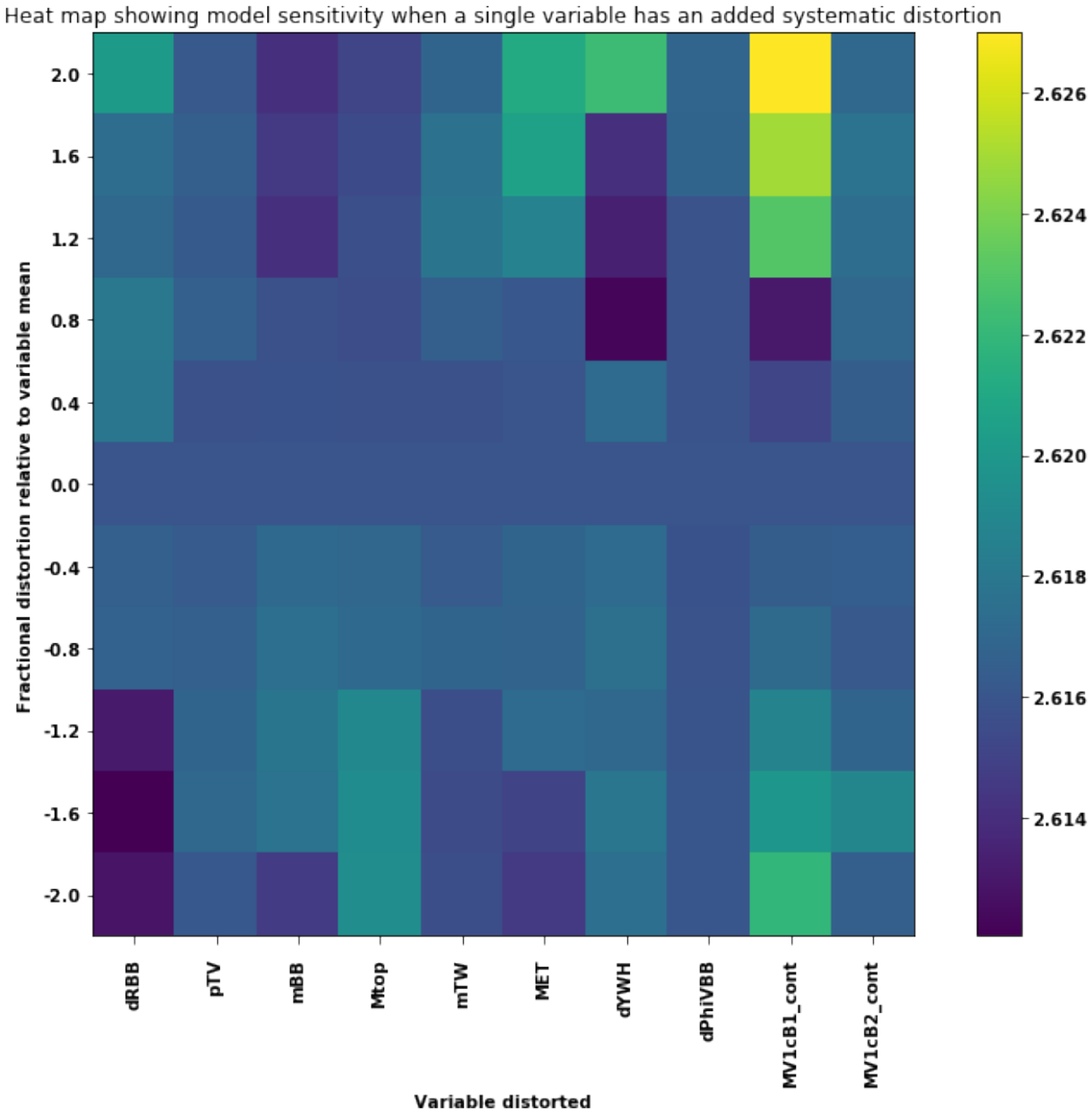

Fig. 22: Comparison of mean sensitivity after adding fractions of variable means to variable sets. Tested for 10 iterations

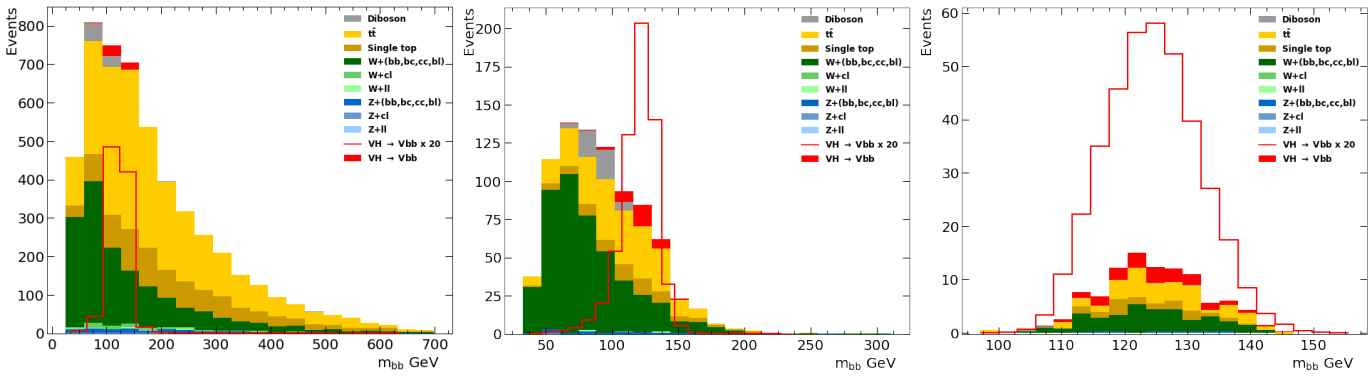

Fig. 23: Comparison of variable plot m_{BB} for our uncut data frame (left), our iterative-cut data frame (centre), and the high decision value cut data frame (right)

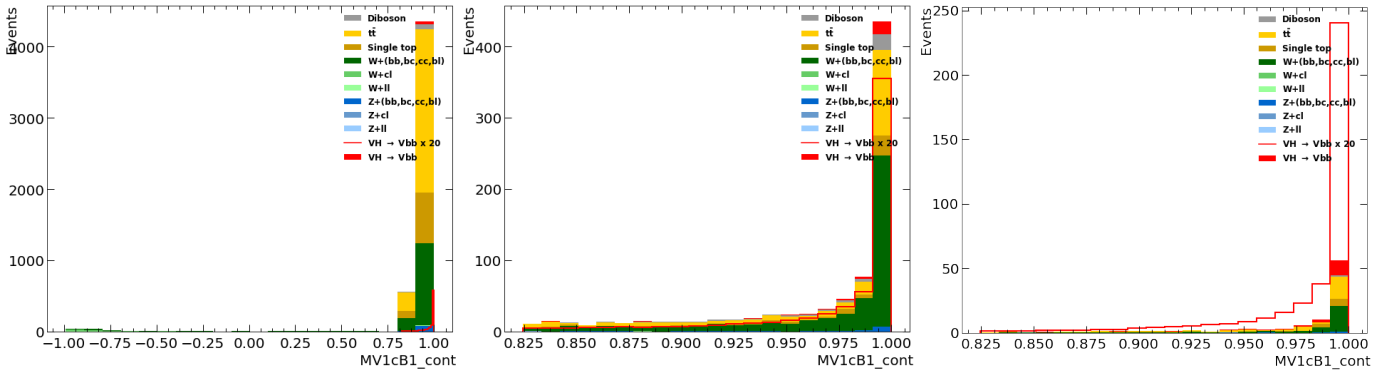

Fig. 24: Comparison of variable plot $MV1^{B1}_{cont}$ for our uncut data frame (left), our iterative-cut data frame (centre), and the high decision value cut data frame (right)

D. Events selected in the high sensitivity region

To calculate the sensitivity of our neural networks, a new variable is created within our data frame called 'decision_value'. This value represents the models' predictions on whether an event is a background or signal event. However, we can also use this variable to get some insight into the inner workings of our models. We first plot our decision value variable, as seen in Figure A.15. In this plot, we can see the result of our models, with the vast majority of background events occurring in the region of low decision value, below 0.5. If we cut our data frame to include only the high decision value events, we can get further insight into the decisions made by our models. We shall cut the decision value variable at $decision\_value > 0.65$. When we do this, we exclude a large portion of our background events.

We can now look at individual variables and compare the plots from our original uncut data frame, our iterative cut based approach data frame, and the high decision value data frame. Rather than look at all variables, we shall look at the two which have the overall most significant effect on our models' sensitivity: m_{BB} (Figure 23) and $MV1^{B1}_{cont}$ (Figure 24).

When looking at our variable plots for m_{BB}, we clearly see the improvement the neural network causes when compared to our cut based approach. The neural network approach excludes a large range of values for m_{BB}, leaving only data points with masses in the region $100GeV \rightarrow 155GeV$, compared to our range of $40GeV \rightarrow 310GeV$. This cut is logical, as we only expect masses close to the true mass of the Higgs, $125GeV$ [1]. In both our iterative cut result and our neural network result, we see a clear peak occurring at this value as one would expect.

One of the clearest differences between these plots, other than the number of remaining events, is the apparent complete exclusion of the Diboson particle. Although these particles are associated with Higgs decays, they are not the target for our neural network, and so it is logical for them to be completely excluded from our results.

When we look at our plots for $MV1^{B1}_{cont}$, we see similarities between our cut based method and the neural network output. We see that both exclude all values $MV1^{B1}_{cont} < 0.825$. However, our neural network has excluded a far larger quantity of data. It has correctly identified that the highest chance for the presence of a Higgs boson via the $H \rightarrow \bar{b}b$ decay route occurs when this value is high. This value represents the classification of our jet, with values close to 1 representing a high likelihood that this jet is a b-tagged jet. As b-tagged jets are the result of a pair of $\bar{b}b$, it is a strong indicator of a Higgs event occurring.

IV. Conclusion

As expected, our neural network approach to this task greatly improved upon the traditional method of using a cut based approach. Our neural network was also largely independent of small changes to input data, such as the removal of a single variable (excluding some important variables) as well as any systematic distortions within any one variable. Our final model structure was also relatively simple, requiring only 700 trainable parameters to greatly improve upon traditional methods of excluding background events from our data set.

Further experiments included the use of 'sample weights' when training our models. These are values associated with the relative likelihood of any single event occurring in an actual experiment, resulting in the inputted events being more representative of what would occur experimentally. The results are shown in Figure 25, where we see there is little improvement when compared to our previous final models, except for a reduction in variance by a factor of a half. This is unexpected, with a larger improvement in sensitivity expected due to the introduction of weightings. The reason for this lack of improvement is unknown, and is an area of further study.

As already stated in previous analysis of our models, another area for possible further study is the independence of these models with respect to systematic distortion of variable values. As stated in our analysis, we would expect our models' sensitivity to be largely independent of small distortions, but would expect a larger change in sensitivity for the large distortions tested. An analysis into the values of various neuron weights and biases associated with variables whose distortion had an effect on our models' sensitivity, compared to variables that were entirely independent of distortion, would likely shed more light on this effect. However, the analysis of individual neuron weights is a more complex topic, and beyond the scope of this paper.

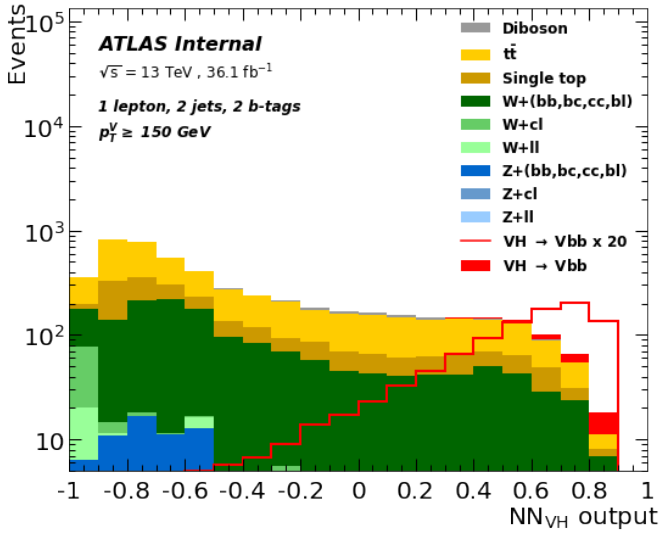

Fig. 25: Neural network output plot when sample weights were used for training in our final models, achieving mean sensitivity $S_\mu = 2.60$ with variance $\sigma^2 = 0.002$

APPENDIX

$$L = \frac{1.01}{1.01 - \Delta} - 1 \tag{A.1}$$

$$L = e^{4\Delta^2} \tag{A.2}$$

$$L = (2\Delta)^4 \tag{A.3}$$

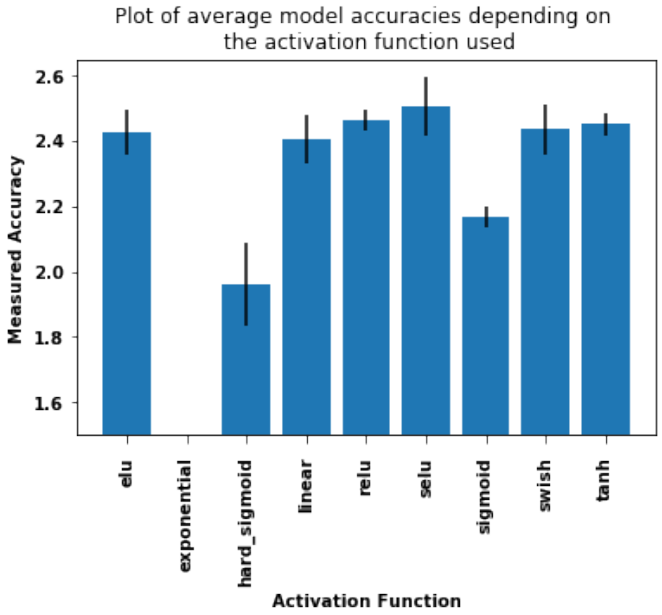

Fig. A.1: Measured sensitivity for different activation functions, where each hidden layer has the same activation function

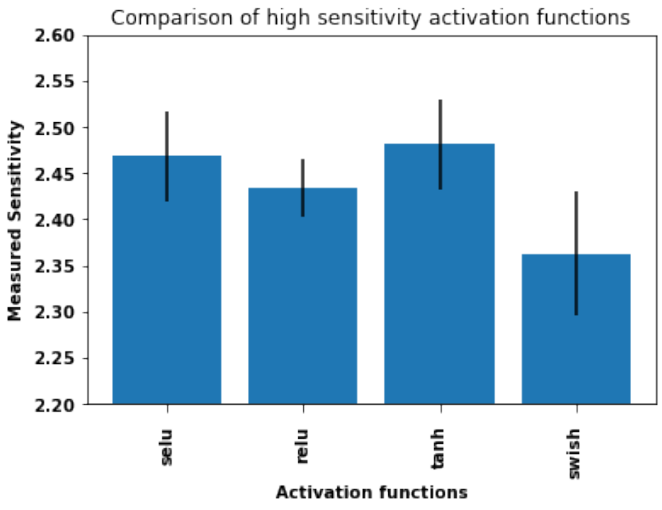

Fig. A.2: Measured sensitivity for the best selection of activation functions, where each hidden layer has the same activation function

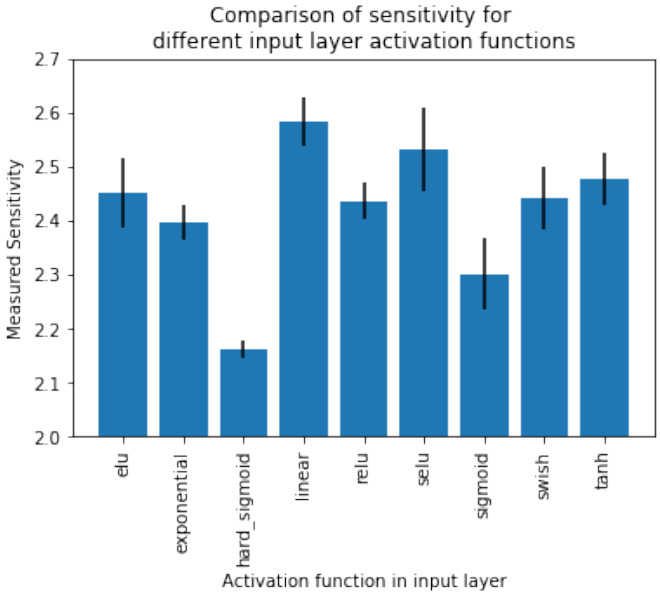

Fig. A.3: Measured sensitivity for different activation functions in our input layer

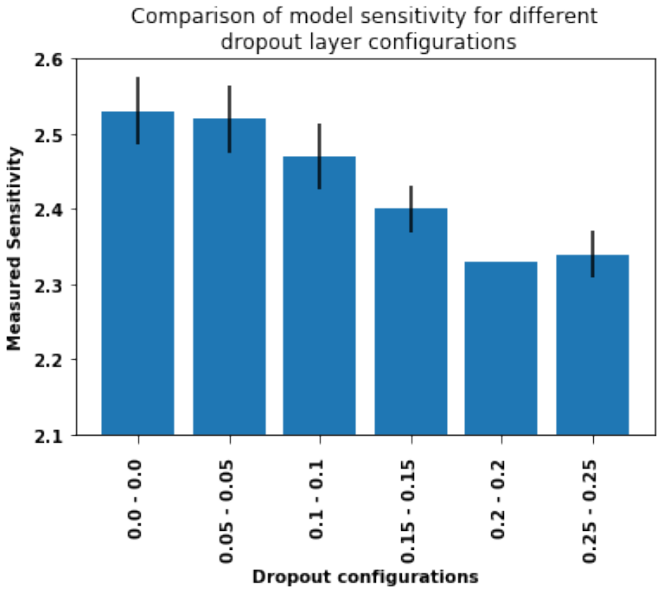

Fig. A.4: Measured sensitivity for different drop out layer structures

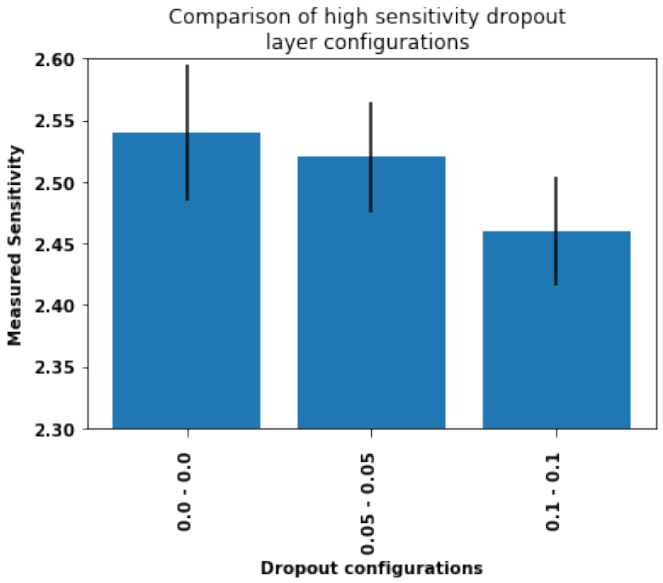

Fig. A.5: Measured sensitivity for best drop out layer structures

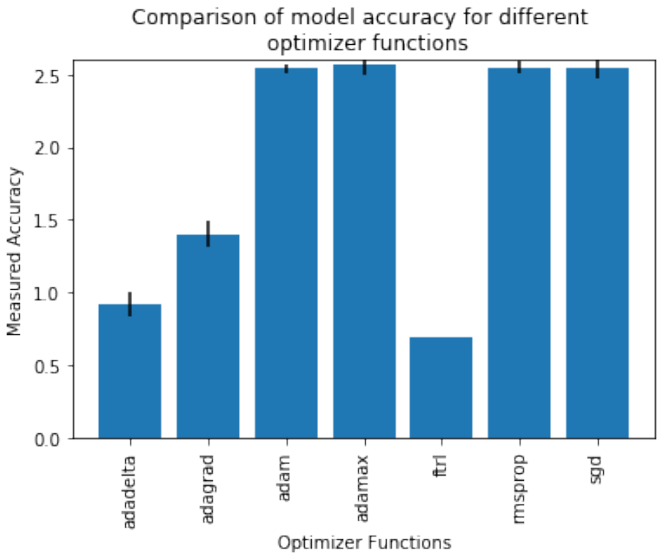

Fig. A.7: Measured sensitivity for all possible optimizer functions available in TensorFlow

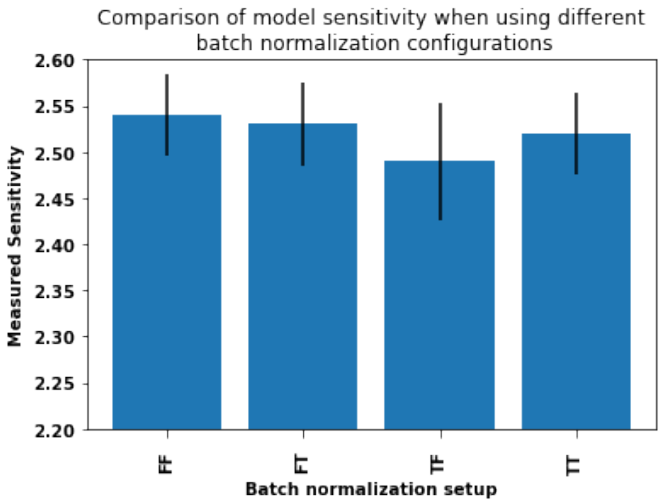

Fig. A.6: Measured sensitivity for different batch normalization structures. Batch normalization layers were placed after dense layers and drop out layers, with 'T' representing True (a batch normalization layer is present) and 'F' representing False.

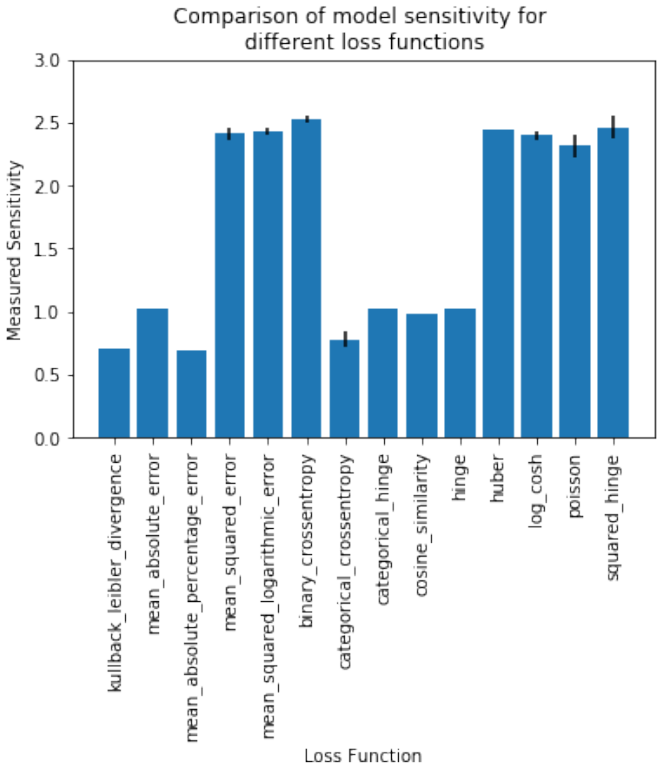

Fig. A.8: Measured sensitivity for all possible loss functions available in TensorFlow

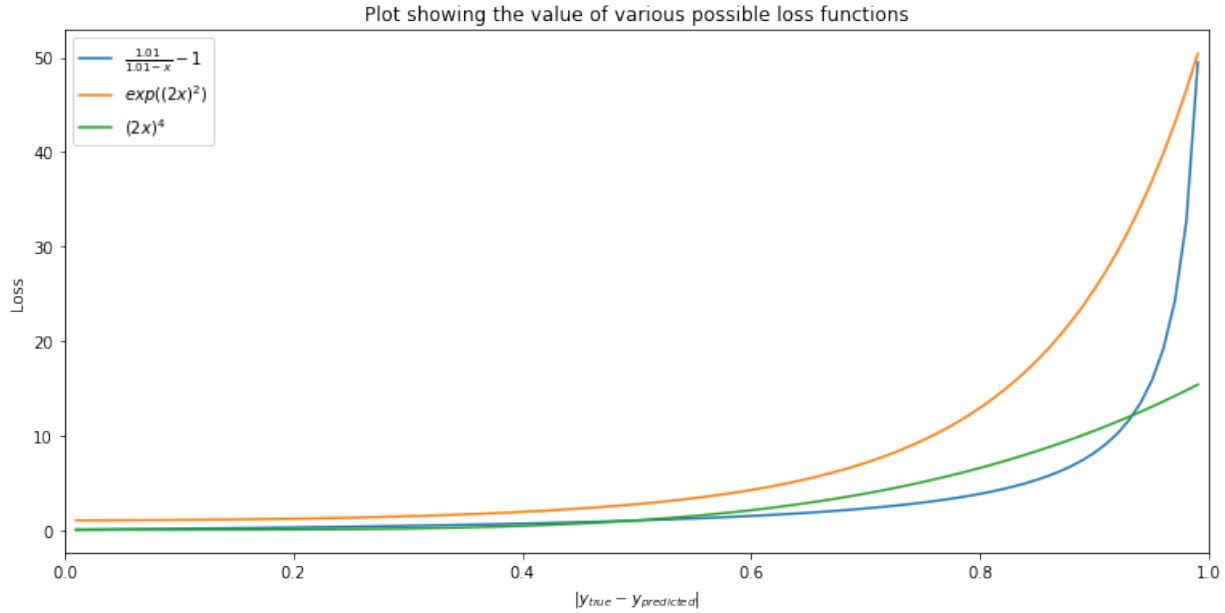

Fig. A.9: Plot of *f(x)* against x for a set of possible custom loss functions.

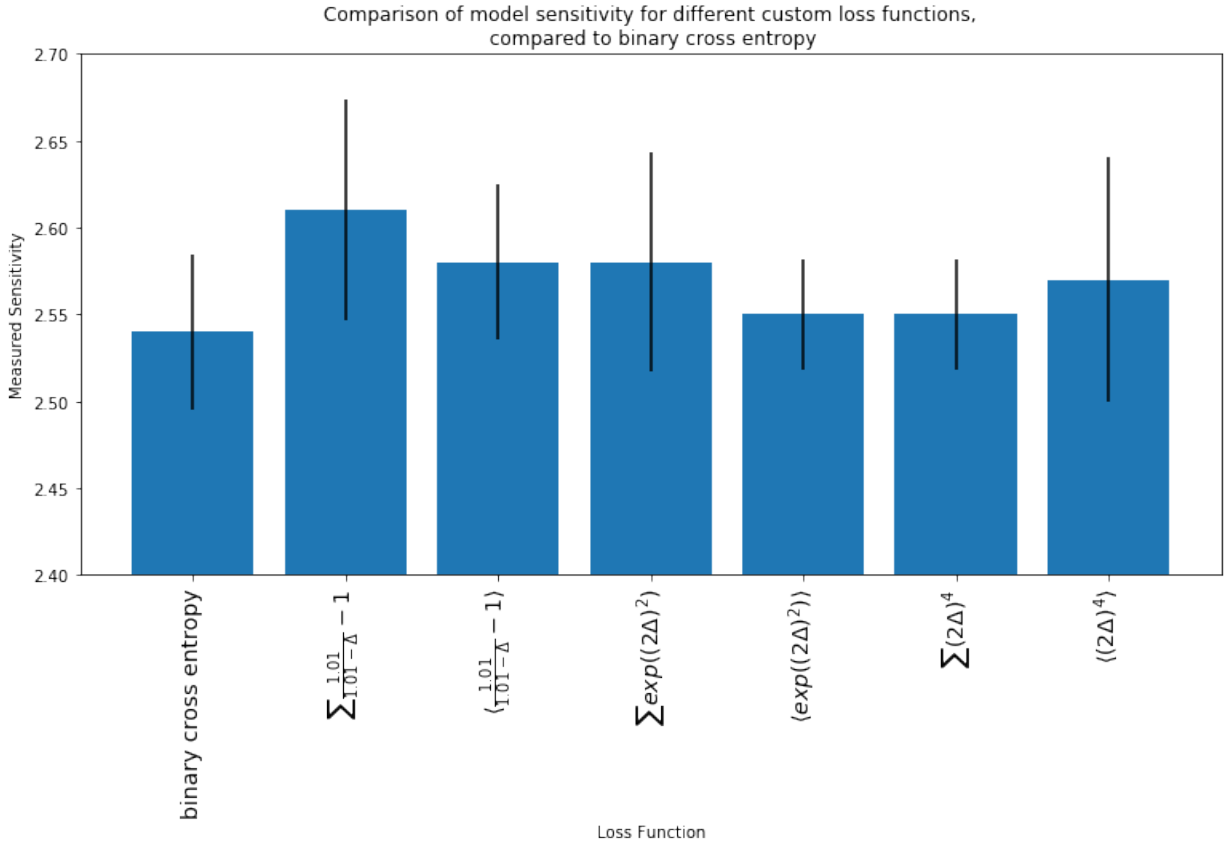

Fig. A.10: Measured sensitivity for custom loss functions compared to binary cross entropy

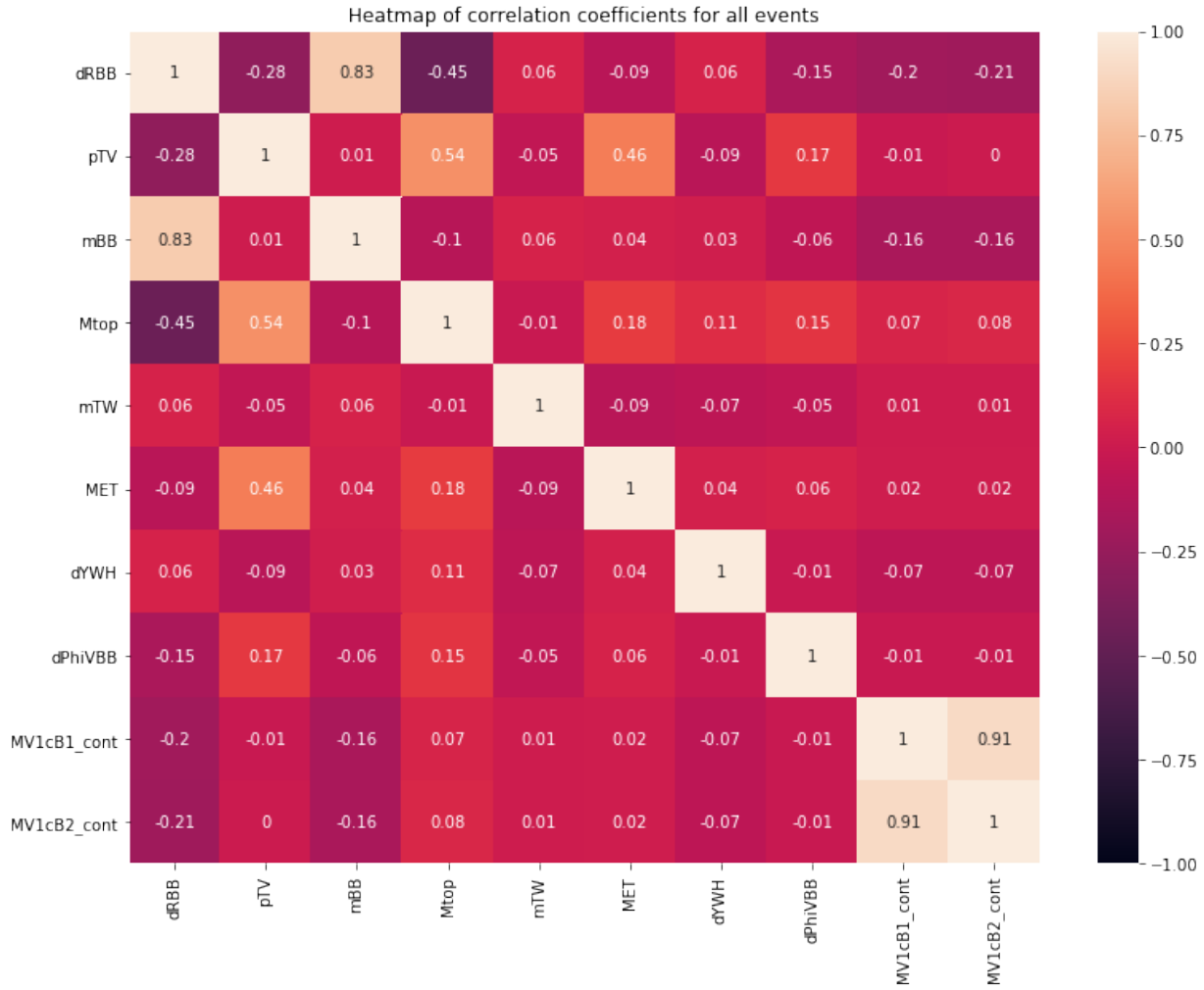

Fig. A.11: Heat-map[22] showing measured correlation coefficients of all input variables for all events

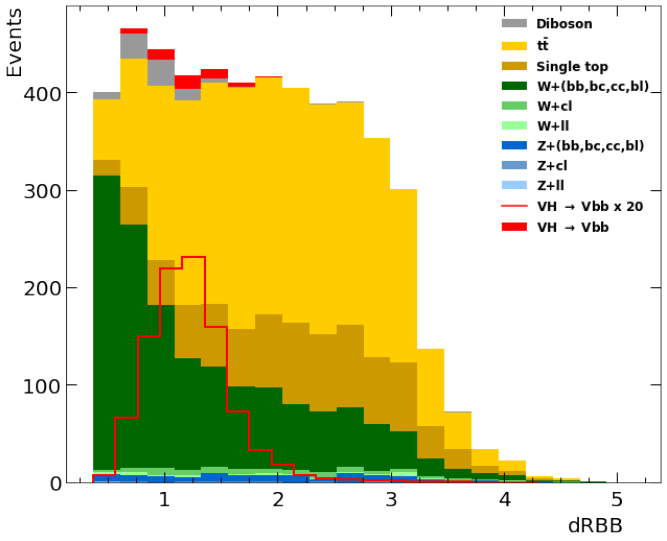

Fig. A.12: Variable plot for $\Delta R(b_1, b_2)$ from initial full data set

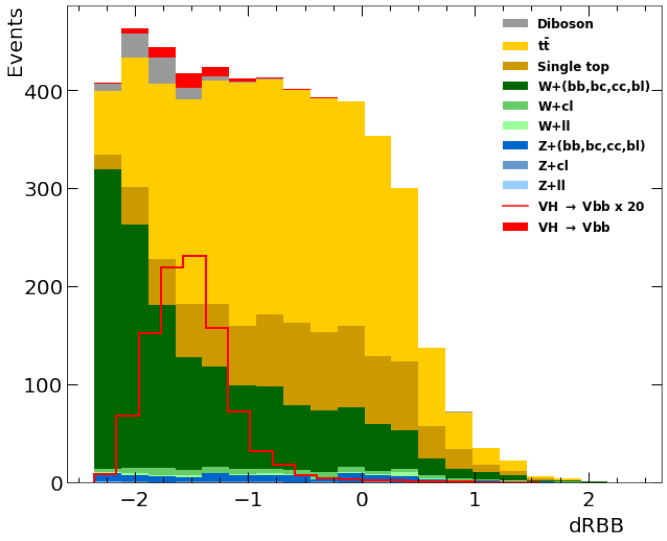

Fig. A.13: Variable plot for $\Delta R(b_1, b_2)$ for high decision value of neural network output, shifted by -2μ

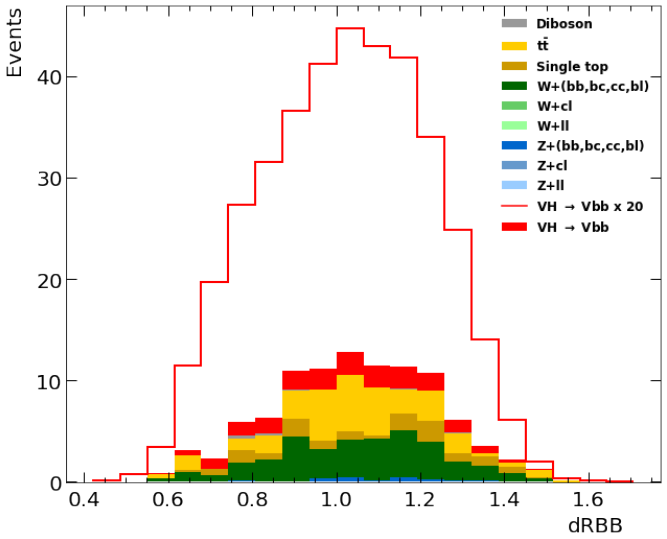

Fig. A.14: Variable plot for $\Delta R(b_1, b_2)$ for high decision value of neural network output

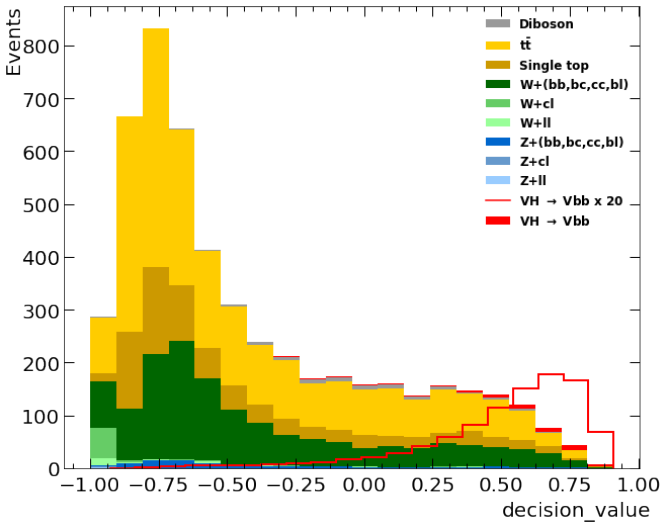

Fig. A.15: Plot of the variable decision value

REFERENCES

[1] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins, and et al., "Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc," *Physics Letters B*, vol. 716, p. 1–29, Sep 2012.

[2] G. Aad, B. Abbott, J. Abdallah, O. Abdinov, R. Aben, M. Abolins, O. S. AbouZeid, H. Abramowicz, H. Abreu, and et al., "Measurements of the higgs boson production and decay rates and coupling strengths using pp collision data at

$$\sqrt{s} = 7$$

s = 7 and 8 tev in the atlas experiment," *The European Physical Journal C*, vol. 76, Jan 2016.

[3] M. Aaboud, G. Aad, B. Abbott, O. Abdinov, B. Abeloos, D. Abhayasinghe, S. Abidi, O. AbouZeid, N. Abraham, H. Abramowicz, and et al., "Observation of h→bb¯ decays and vh production with the atlas detector," *Physics Letters B*, vol. 786, p. 59–86, Nov 2018.

[4] V. Khachatryan *et al.*, "Search for the standard model Higgs boson produced through vector boson fusion and decaying to $b\bar{b}$," *Phys. Rev. D*, vol. 92, no. 3, p. 032008, 2015.

[5] G. Aad, B. Abbott, D. C. Abbott, O. Abdinov, A. A. Abud, K. Abeling, D. K. Abhayasinghe, S. H. Abidi, O. S. AbouZeid, and et al., "Identification of boosted higgs bosons decaying into b-quark pairs with the atlas detector at 13

TeV

," *The European Physical Journal C*, vol. 79, Oct 2019.

[6] M. Aaboud, G. Aad, B. Abbott, O. Abdinov, B. Abeloos, S. H. Abidi, O. S. AbouZeid, N. L. Abraham, H. Abramowicz, and et al., "Evidence for the

$$h \to b\bar{b}$$

decay with the atlas detector," *Journal of High Energy Physics*, vol. 2017, Dec 2017.

[7] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[8] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[9] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on Nuclear Science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[13] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[14] H. Abdi and L. J. Williams, "Principal component analysis," *WIREs Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[15] M. P. Robertson, N. Caithness, and M. H. Villet, "A pca-based modelling technique for predicting environmental suitability for organisms from presence records," *Diversity and Distributions*, vol. 7, no. 1-2, pp. 15–27, 2001.

[16] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[17] A. Klein, S. Falkner, N. Mansur, and F. Hutter, "Robo: A flexible and robust bayesian optimization framework in python," in *NIPS 2017 Bayesian Optimization Workshop*, 2017.

[18] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, vol. 13, p. 20, Citeseer, 2013.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014.

[20] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, pp. 2483–2493, Curran Associates, Inc., 2018.

[21] A. M. Sirunyan, R. Erbacher, W. Carvalho, M. Górski, D. Kotlinski, B. Ujvari, S. Ozturk, A. Polatoz, R. Lander, N. Smith, *et al.*, "arxiv: Search for higgsino pair production in pp collisions at sqrt (s)= 13 tev in final states with large missing transverse momentum and two higgs bosons decaying via h to bb-bar," *Phys. Rev. D*, vol. 97, no. CMS-SUS-16-044, p. 032007, 2017.

[22] M. Waskom and the seaborn development team, "mwaskom/seaborn," Sept. 2020.