




Deep Neural Network to Describe the Measurement of the Higgs Production in the Full Leptonic Channel via Vector Boson Fusion

Luis Sánchez¹ , Félix Díaz² , and Jhonny Rojas³ 

¹Universidad Tecnológica del Perú, Perú, jusa295@gmail.com

²Vicerrectorado de Investigación, Universidad Autónoma del Perú, Perú, felix.diaz@autonoma.pe

³Pontificia Universidad Católica del Perú, Perú, jrojash@pucp.edu.com

Abstract–

In this article, an analysis of the Higgs boson production via vector boson fusion in the $SM H \rightarrow WW \rightarrow 2l2\nu$ ($l = e, \mu$) is performed from an optimization technique in the event selection, called DNN analysis. This analysis compares the standard selection process that CERN performs to study the production of a particle from a cut-based analysis, where the study of statistical significance shows that DNN analysis can better separate signal and background events. To perform the DNN analysis, we optimized the neural network configuration to discriminate signal and background events effectively. Moreover, studies of activation functions such as RELU and Sigmoid, stochastic optimization methods such as ADAM, and regularization methods such as Dropout. All this leads to constructing an optimal neural network topology capable of learning events and signal and background discrimination. Finally, we found an important improvement of approximately 47 % and 27 % for Z_{VBF} and Z_{Higgs} , respectively.

I. INTRODUCTION

In recent years, deep artificial neural networks (including recurrent ones) have won numerous contests in pattern recognition and machine learning [1]. Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years [2]. Deep learning techniques are applied in some fields' Natural language processing, information retrieval, analysis of social networks, transportation prediction and sound processing [3]. A simple technique to extract the dark knowledge of a Deep Multi-Column Deep Learning Network and its compression into a shallow neural network (NN) causing not only the improvement of the train and test performance of the latter but a cheap way to approximate the former results but with fewer parameters [4]. Single units in a deep neural network (DNN) functionally correspond with neurons in the brain [5]. Multifaceted feature visualization can better understand deep neural networks by identifying which features each of their neurons have learned to detect [6]. Deep learning relies on multiplier layers of nodes and many edges linking the nodes forming input/output (I/O) layered grids representing a multiscale processing network [7].

The Higgs model is a keystone of the Standard Model and its supersymmetric extensions [8]. The Higgs is a massive and unstable boson with an extremely short half-life estimated to be about 1.56×10^{-22} seconds. Bosons are produced as part of the proton scattering process and are detected through their decay products. The collision of W or Z vector bosons can produce the Higgs boson, this process is called vector boson fusion (VBF) and it is represented as WWH and ZZH . Analogously there are other Higgs production channels such as gluon collisions, also by photon annihilation [9].

As the Higgs is created, it decays into a variety of particles, including quarks, photons, electrons, and muons. To determine if the Higgs boson exists, the energy and momentums of these particles are measured by the ATLAS and CMS detectors in the Large Hadron Collider (LHC) [10]. They use Monte Carlo event simulation techniques [11], and statistical analysis such as principal component and discriminant analysis to extract significant features from the data and classify Higgs boson decay events. In addition, artificial intelligence and machine learning have been used to identify patterns and trends in experimental data.

Under this context, machine learning optimization techniques are used to change the weights of a neural network during the time it's trained. Methods such as Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSprop), ADADELTA, and Adaptive Gradient (ADAGRAD) are the most used. The efficient training of neural networks in deep learning depends on optimization methods or optimizers. Many studies have compared different optimizers for training neural models, showing that certain optimizers work better for specific problems. The adaptive gradient descent optimization technique ADAM, modifies the learning step sizes for each parameter separately [12]. Stochastic Gradient Descent (SGD) is a traditional gradient descent optimization technique that modifies the network weights in accordance with the gradient of the loss function's opposite direction [13]. In terms of convergence and efficiency, ADAM can be more successful in DNN training than SGD and is typically quicker than SGD [14]. Moreover, ADAM is more resilient in terms of choosing the learning step size, necessitating less hyperparameter adjustment than SGD. This allows simula

ted or actual users to interact with the model in real-time ADAM is able to improve the performance of a wide and deep neural network [15]. Then, ADAM, the most widely used machine learning optimizer compare others, because it is faster and more efficient, is an optimization method based on stochastic functions using algorithms that properly combine the weights and biases to minimize the loss, this combination allows to calculate a learning rate (the network learns not to over train and minimizes the loss).

The aim of this work is to measure the production of Higgs in the full Leptonic Channel via Vector Boson Fusion taking data from 2016. H W^+W^- decay at a \sqrt{s} of 13 TeV utilizing a total L's of $35.9 fb^{-1}$ gathered from proton-proton collisions at the LHC base on the DNN.

II. THEORETICAL FRAMEWORK

The connections between the neurons are achieved by arranging the neurons in layers. Typically, each layer receives inputs from the preceding layer, with the first layer getting inputs from the input layer. The input layer nodes serve as the Neural Network (NN) variables. The final output of the NN is generated by the last layer and is depicted in *Figure 1*.

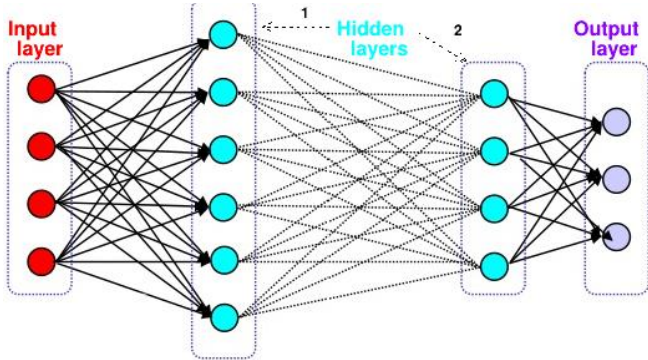


Figure 1. Schematic depiction of a layered ANN (densely connected feed forward network).

When designing a NN, one choice to be made is the connection between the output of one layer and the input of the next layer, as well as the internal structure of the layers. The most widely used structure is the fully connected layer, where each neuron receives the output of every node in the previous layer as input. The latter leads to a straightforward mathematical description of the NN: if x_0 is the vector of input values, the output of the following layer of neurons is represented by $h_1 = f(w_0 \cdot x_0 + b_0)$. Here, the layer is identified as h_1 . The layers between the input and output layers are referred to as hidden layers. The activation function of the neurons in this layer is represented by f . The weight matrix, w_0 , contains a vector of weights for each neuron, and the bias vector, b , holds a bias value for each neuron. The output of the subsequent layer would be designated as $h_2 = f(w_1 \cdot x_1 + b_1)$. Therefore, a recursive mathematical representation of the entire network is represented by,

$$h_{i+1} = f(w_i \cdot x_i + b_i) \quad (1)$$

where $x_i = h_i$. The equation (1) applies to all feed-forward networks, where there are no loops, and a neuron output does not impact its input. In addition, there are various other layer types, such as pooling layers, convolutional layers, and dropout layers. However, only dropout layers, which serve as a regularization technique, are utilized and addressed in this study context [16].

A. Activation Functions

The selection of the activation function for the neurons in a neural network is a crucial aspect of its design. This decision holds especially significant weight when considering the output layer, as the activation function of this layer sets the limits of the possible values the output can take. It defines the shape of the network output with respect to its inputs. In this section, we will examine and discuss some of the most frequently used activation functions.

The Rectified Linear Units (RELU) are the most commonly used activation functions, as seen in *Figure 2* (right). They are defined as $f(z) = (0, z)$, meaning that the output of a RELU neuron is equal to the weighted input of the neuron, $w_i[j] \cdot h_i$, if it is greater than a threshold of $-b_i[j]$, or zero otherwise.

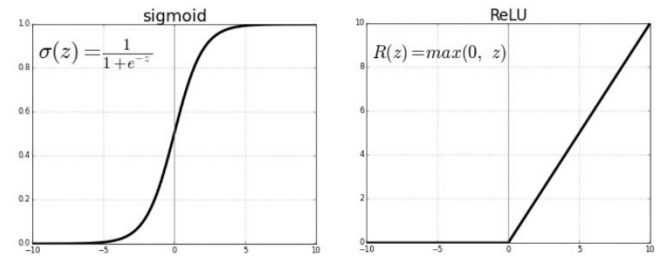


Figure 2. Sigmoid and ReLU activation functions.

One advantage of using RELU units is their ease of computation. However, there is a downside in that they can "die," meaning if their output becomes zero, it is possible that the training algorithm of the neural network will not find a weight update that brings the output back to a non-zero value. The latter is because most algorithms rely on gradient descent, and the gradient for negative values of $w_i[j] \cdot h_i + b_i[j]$ is simply zero.

Neurons with step function activation, known as perceptrons, played a crucial role in the evolution of neural networks. They were used to construct logical functions such as AND, OR, and NAND, which are the fundamental building blocks for all other functions in a computer.

Sigmoid units represent a more generalized form of perceptrons capable of producing a non-binary output. The sigmoid function is defined as $\sigma(z) = (1 + e^{-z})^{-1}$, as seen in *Figure 2* (left),

with z being represented in the same manner as the RELU units.

The sigmoid function can map its inputs to a range of values between 0 and 1, making it useful for binary classification tasks in the output layer of a neural network. Its continuous nature also allows for efficient gradient calculations during training compared to non-differentiable functions like the step function. However, as with the RELU function, the sigmoid can suffer from small gradients for extremely high or low input values, causing neurons to become unresponsive to weight changes in the network. RELU is used in the input to address this, and hidden layers of the neural network are RELU. In contrast, a sigmoid function is used in the output layer to ensure that the sum of all outputs is one and that individual outputs fall within the $[0,1]$ range.

B. Training of a neural network

When training a neural network, the fundamental approach is to compare its output on a given training input to a known target output. This comparison allows the modification of the weights, biases, and other trainable parameters of the neurons to enhance the alignment between the computed and target output. As such, the trainable parameters are typically initialized randomly at the start of the training process.

One common approach for weight initialization is to randomly sample values from a Gaussian distribution centered at zero with a standard deviation of one. However, there exist various other methods for weight initialization. An alternative approach is to initialize weights from a Normal distribution, where most values are clustered around the mean (e.g., $\mu \approx 0$). Another option is to use a Uniform distribution, where each value has an equal probability of being selected. In this analysis, both normal and uniform initialization techniques were employed. These methods can enhance the convergence behavior of the network during training, ultimately reducing the time required to train the network model.

Once the weights are initialized, a loss function is computed to measure the discrepancy between the predicted output and the desired output. Subsequently, a gradient-based algorithm is typically employed to minimize this loss, improving the agreement between the neural network output and the training data. For classification tasks with an output node for each class, the cross-entropy function (S) is commonly utilized. This function is defined as follows:

$$S = -\frac{1}{h} \sum_x \sum_y [y_j \cdot \ln \ln(z_j) + (1 - y_j) \cdot \ln(1 - z_j)] \quad (2)$$

In this equation, the z_i represents the outputs of the distinct output neurons responsible for classifying the input, while y_i represents the desired output neuron values.

The minimization algorithm proceeds by iteratively adjusting the weights and biases in order to minimize the loss. One of the most straightforward algorithms used for this purpose is gradient descent, which involves calculating the derivative of the loss function with respect to the weight and bias parameters and adjusting them in the direction of the descending gradient with a variable step size. This step size is commonly referred to as the learning rate, and it is a tunable parameter in the network architecture. Typically, the learning rate is gradually reduced over time as the minimum is approached, a technique known as learning rate decay.

A technique that was attempted is called ADAM (Adaptive Moment Estimation), which is discussed in [17]. This algorithm is utilized for first-order gradient-based optimization of stochastic objective functions and is based on adaptive estimates of lower-order moments. Other commonly used algorithms include basic SGD, RMSprop, ADADELTA, and ADAGRAD.

C. Regularization

Overtraining is a common issue that may arise during the training of a neural network. The latter means that the network may learn patterns within the training data that are purely statistical and do not exist in the broader data set that the network is intended to describe. With the purpose of mitigating this effect, a portion of the training data is typically reserved for testing the network's performance on an independent data set. In contrast, the network is being trained on the remaining data.

When the loss function of the training data becomes smaller than that of the test data, the neural network begins to prioritize the statistical artifacts present in the training data. The latter is likely to occur over time, particularly when the network has learned most of the relevant features within the data or when there is inadequate training data, given the number of trainable parameters. Therefore, to treat the problem of overtraining, one possible approach is regularization, which includes commonly used techniques such as dropout layers.

The first technique involves extending the loss function by adding a penalty term of $-\frac{1}{2}\lambda\omega^2$, which is the squared sum of all weights and biases in the network multiplied by the strength of the regularization. This approach penalizes large individual weights within the network, which may indicate the over-reliance on a single input in certain network parts. Therefore, by doing so, it encourages the network to consider the broader patterns in the data rather than focusing solely on the distribution and fluctuations of a single input parameter.

The concept of dropout layers [18] represents a distinct approach to network averaging. In a dropout layer, nodes within the associated layer are randomly deactivated, temporarily removing them from the network as depicted in Fig. 3. In this

way, a different sub-network of the original network is trained in each iteration, effectively training multiple networks with shared neurons simultaneously. This approach has been demonstrated to prevent overtraining and enhance the network's performance.

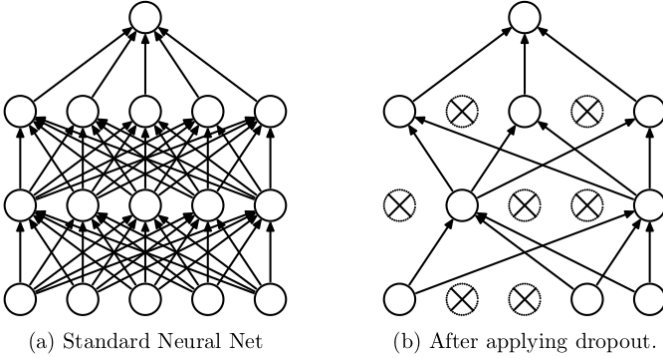


Figure 3. Dropout neural net model: A standard NN, from bottom to top, with 2 hidden layers as an example of a thinned net produced by applying dropout to the network on the left.

II. METHODOLOGY

This analysis primarily focuses on utilizing Deep Neural Networks (DNNs) to discriminate VBF signals from main backgrounds. The general functioning of DNNs has been explained earlier. This section provides an initial exploration of using DNNs in the VBF analysis. The knowledge and insights gained from these initial DNN tests are then used to construct an NN selector, which can effectively differentiate between signal and background. The applicability of this classifier for signal and background discrimination is then discussed. The DNNs were constructed using KERAS [19] with THEANO [20] and TensorFlow [21] backends. The data and MC samples employed here are the same as those used in the related VBF analysis, but the present work is limited to the full leptonic decay channel.

A. Analysis strategy

Many discrimination methods begin by defining kinematic observables and analyzing their distributions. A specialized algorithm is then employed to identify the most effective discriminatory variables. These variables serve as inputs for various discrimination techniques, whose efficacy is evaluated based on the values of the configuration parameters.

In the HWW VBF analysis, it is necessary to discriminate between several backgrounds and the signal. The primary backgrounds include the DY, WW, and TOP processes, with the ggH process also considered a background. The DNNs were evaluated individually for each background to consider their distinct differences from the signal. However, the optimal

performance was achieved when all backgrounds were considered together.

The DNN strategy is devised to attain high efficacy in background suppression. The methodology created can be categorized into the subsequent steps:

- 1) Specify a preliminary group of low-level variables to serve as inputs for the DNN.
- 2) Define the dataset for both signal and background events.
- 3) Train, validate, and test the DNN using previously defined data sets. It is important to mention that during DNN training, the efficacy and stability of the DNN rely on the values of the tuning parameters, primarily selected through a cross-validation technique to prevent overfitting.
- 4) The efficacy of the DNN is evaluated using classifier methods like the receiver operating characteristic (ROC) curves, which showcase the diagnostic ability of a binary classifier system while varying the discrimination threshold. The Area under the ROC curve (AUC) serves as a degree or metric of separability. It indicates the extent to which the model can differentiate between classes. A higher AUC signifies that the model is more proficient in accurately predicting 0s as 0s and 1s as 1s. It is important to mention that we use the AUC due to the small amount of data.

In this analysis, the approach does not involve constructing high-level variables (which are created with previous discrimination). Instead, variables corresponding to the final state objects p_T , η , and ϕ for leptons and jets-containing all pertinent information of the final state are utilized, as indicated in Table 1. The superscript represents the final state, l_1 is the first lepton, l_2 is the second lepton, j_1 is the jet due to the defragmentation of the first quark packet, and j_2 is the jet due to the defragmentation of the second quark packet

Table 1. Kinematic variables used as DNN inputs

Variables	Description
$p_T^{l_1}, \eta^{l_1}, \phi^{l_1}$	Leading lepton p_T , η , and ϕ
$p_T^{l_2}, \eta^{l_2}, \phi^{l_2}$	Trailing lepton p_T , η , and
$p_T^{j_1}, \eta^{j_1}, \phi^{j_1}$	Leading jet p_T , η , and ϕ
$p_T^{j_2}, \eta^{j_2}, \phi^{j_2}$	Trailing jet p_T , η , and

For the purpose of training deep neural networks, the dataset is selected by applying all cuts that define the signal region, except for the high-level cuts: $|\Delta\eta_{jj}| > 3.5$ and $|\eta_{li} - (\eta_{j_1} + \eta_{j_2})/2|/|\Delta\eta_{jj}| < 0.5$. Tables 2 and 3 display this dataset and its division for each m_{jj} region. Therefore, signal samples are utilized for other Higgs masses to enhance the statistics of signal events while maintaining an appropriate balance between signal and background events.

Process	2016 low m_{jj}	2016 high m_{jj}
qqH ($m_H = 125$ GeV)	1.2 K	2.7 K
qqH (other masses)	2.6 K	2.7 K
ggH ($m_H = 125$ GeV)	0.1 K	0.1 K
WW	0.4 K	0.2 K
DY	0.4 K	0.2 K
TOP	4.5 K	2.1 K
Σ Signal	3.7 K	5.4 K
Σ Background	5.4 K	2.6 K

Table 2. Number of events used in DNN for each sample and dataset split.

Training	Testing	Validation
90 %	10 %	20 % train
80 %	20 %	20 % train
75 %	25 %	20 % train
70 %	30 %	20 % train
60 %	40 %	20 % train
50 %	50 %	20 % train

Table 3. Dataset split used in DNN for each sample.

B. Neural network building

Keras, a Python library for developing and evaluating deep learning models, is known for its powerful yet user-friendly features. It leverages the efficient numerical computation libraries Theano and TensorFlow, enabling you to define and train neural network models with just a few lines of code. The following steps are involved in this process.

Load data

When working with machine learning algorithms that rely on stochastic processes, such as random numbers, it is advisable to set the random number seed. The latter ensures that the same code can be executed multiple times and produce identical results. Therefore, this is particularly helpful for demonstrating results or comparing algorithms using the same randomness source.

Moreover, to facilitate the use of the dataset described in Table 9, which includes a set of variables per event related to binary classification, signal events should be defined as 1 and background as 0. Consequently, this allows for direct use with neural networks, which require numeric input and output

values. In addition, data can be loaded directly using the NumPy tool, a Python extension that provides extensive support for working with vectors and arrays. The dataset consists of 12 input variables (selected from Table 1) and one output variable (the last column), the class variable.

After loading, the dataset can be divided into input variables (X) and output class variables (Y). The random number generator should be initialized to ensure the reproducibility of the results. With the data loaded and the random number generator initialized, the next step is to define the neural network model.

Define model

In Keras, models are constructed as a sequence of layers; each added one at a time until the desired network topology is achieved. A fully connected multi-layer network structure is utilized for the analysis at hand. It includes an input layer with 12 neurons corresponding to the 12 variables, one or more inner layers with a large number of neurons, and an output layer with 1 neuron to predict the signal or background class.

To initialize the network weights, a small random number is generated from either a uniform or Gaussian distribution, depending on the layer. A uniform distribution is used for the output layer to generate random weights between 0 and 0.05, a standard uniform weight initialization in Keras. A "normal" distribution is used for all other layers to generate small random numbers from a Gaussian distribution.

Therefore, a sequential model is created to implement the latter, and the layers are added using appropriate initialization methods and activation functions. It allows for greater control over the network topology and can improve the accuracy of the final model.

Compile model

After defining the model, the next step is to compile it. Then, Keras utilizes efficient numerical libraries (also known as the "backend"), such as Theano or TensorFlow, to represent the network for training and prediction. The backend automatically chooses the best way to run the network on hardware, including CPU or GPU, and can even distribute computation across multiple devices.

During compilation, we must specify additional properties necessary to train the network and find the best weights to predict this problem. It includes specifying the loss function used to evaluate a set of weights, the optimizer used to search for different weights, and any optional metrics we want to collect and report during training.

In this analysis, we will use binary cross-entropy, the logarithmic loss function defined in Keras, for binary classification problems. We will also use the efficient ADAM gradient descent algorithm as the optimizer. These choices can significantly impact the performance of the model, and

selecting appropriate loss functions and optimizers is an essential part of building an effective neural network.

Fit model

To train our loaded model, we can call the fit function. The training process involves iterating through the data set for a fixed number of epochs, each comprising several weight updates based on a batch of instances. The batch size refers to the cases evaluated before each weight update. In this analysis, we experimented with different values for the number of iterations and the batch size, ultimately choosing around 400-1000 iterations and batch sizes of 10 and 50 for low and high m_{jj} , respectively. These values were selected through a process of trial and error.

Evaluate the model

To accurately evaluate the model's performance, it is ideal to separate the dataset into training, testing, and evaluation sets. However, for simplicity, we trained the NN on the entire dataset and evaluated its performance on the same dataset. While this approach gives us an idea of the model's accuracy, we must determine how well the algorithm can perform on new data. To evaluate the model's performance on the training dataset, we used the evaluation function in the model, passing the same input and output used to train the model. Thus, it generated a prediction for each pair of input and output and collected scores, including the average loss and any metrics set, such as accuracy. We separated 80 % of the events for training and 20 % for testing, with 20 % of the training set used for validation, as selected from Table 4.

Parameter	Test values
Hidden layers	1, 2, 3, 4, 5, 6
Nodes per layer	4, 24, 48, 72, 96, 120, 240, 10 K
Batch size	5, 10, 20, 50
Epochs	200, 400, 1 K, 2 K, 5 K
Dropout per layer	10 %, 15 %, 20 %, 25 %

Table 4. Configurations for the DNN optimization.

III. RESULTS

Fine-tuning the parameters has maximized the network's performance, which was measured using the AUC metric on the validation dataset. However, we had to make many decisions when designing and configuring the DNN models. We empirically resolved most of these decisions by testing them on real data through trial and error. Therefore, it's crucial to have a robust evaluation method to measure the performance of DNN models.

We trained the NNs until the accuracy on the training and validation datasets stopped increasing, which prevented overfitting and maximized the achievable classification performance. Figure 4 and Figure 5 present a robust method for evaluating the performance of DNN models, considering a good analysis strategy and correct construction method. Figure 4 and Figure 5 compare the model accuracy and loss for each epoch in the training and validation sets of the best DNN configurations in low and high regions.

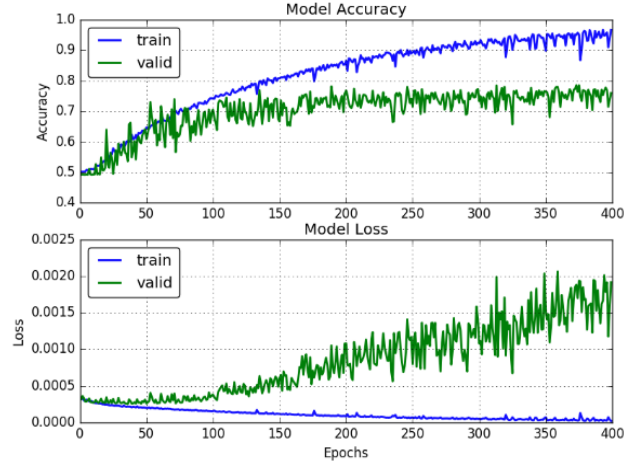


Figure 4. Model accuracy and loss for low m_{jj} regions. Training and validation along training epochs for best DNN configurations.

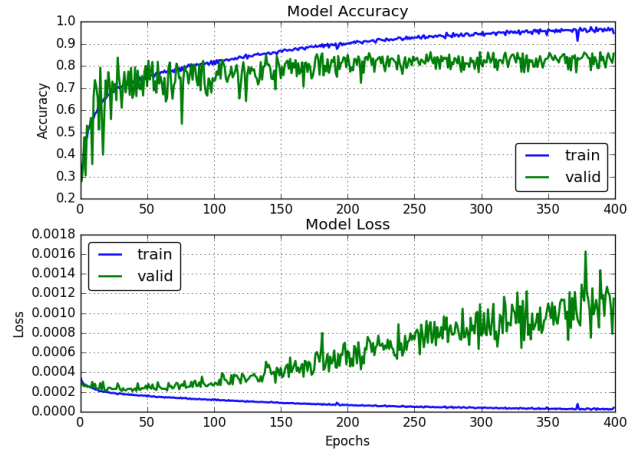


Figure 5. Model accuracy and loss for high m_{jj} regions. Training and validation along training epochs for best DNN configurations.

Usually, increasing the amount of training leads to higher accuracy. However, this is not always the case, and in some situations, it may be beneficial to terminate training early. The model may only fit the training and test datasets if sufficient training exists. Conversely, excessive training can overfit the training data and result in poor performance on the test set.

Therefore, to mitigate this issue, we utilized the following regularization techniques:

- Early stopping: Stop training when the performance on a validation dataset begins to deteriorate.
- Dropout: Remove inputs during training in a probabilistic manner.

To maximize the separation of background and signal events, a number between 0 and 1, called score, is the output of a binary classifier neural network. Figure 6 and Figure 7 shows the score distributions for the signal and background samples of both NNs, and the DNN discriminator demonstrates a good separation ability between the two classes.

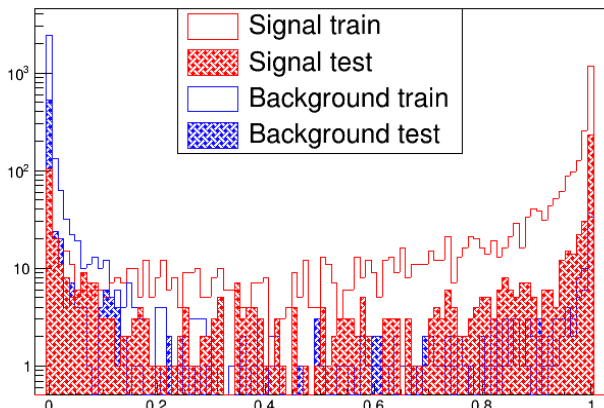


Figure 6. Score distribution for low m_{jj} regions. DNN discrimination for training and testing datasets for best DNN configurations. The horizontal and vertical axes represent the new discrimination variable ($DNNvar$) and the number of events, respectively.

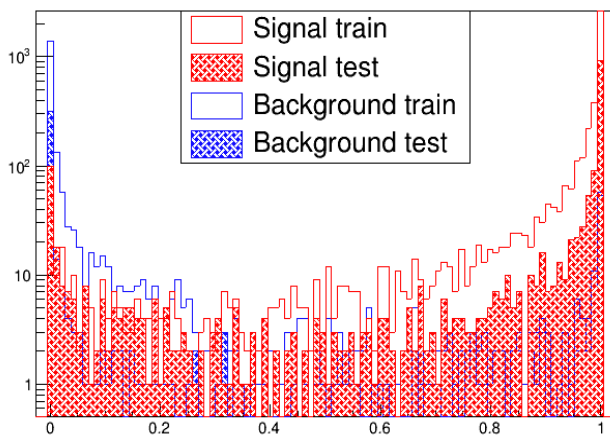


Figure 7. Score distribution for high m_{jj} regions. DNN discrimination for training and testing datasets for best DNN configurations. The horizontal and vertical axes represent the new discrimination variable ($DNNvar$) and the number of events, respectively.

Figure 8 and Figure 9 display the ROC curves, which illustrate the performance of the DNN discriminator for the signal and background samples. The AUC, a measure of classification accuracy, ranges from 82 % to 98 % for the low and high MJJ regions, indicating that our models efficiently discriminate between the signal and background by improving signal efficiency and background rejection.

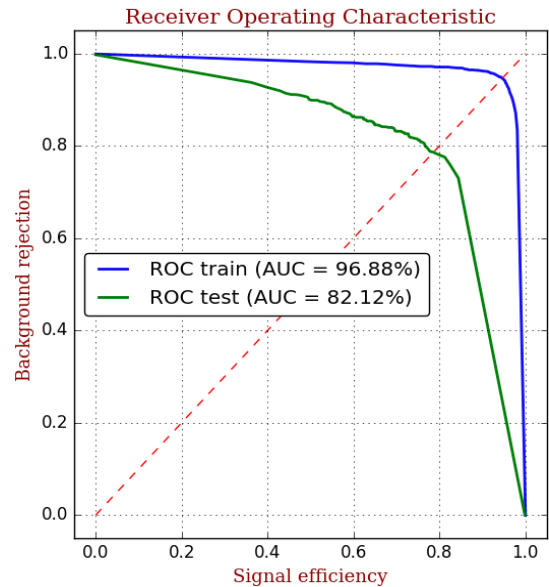


Figure 8. DNN performances for low m_{jj} regions. ROC curve for training and testing datasets for best DNN configurations.

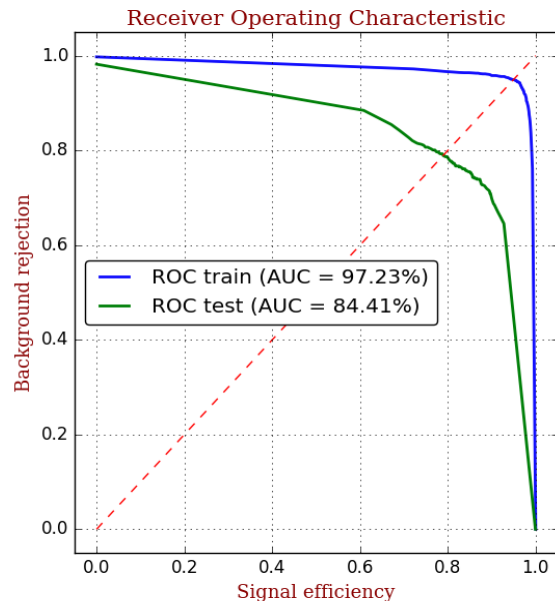


Figure 9. DNN performances for high m_{jj} regions. ROC curve for training and testing datasets for best DNN configurations.

We added a discriminant variable, $DNNvar$, to the trees of all the samples in this analysis. It is possible to utilize the shape of

this variable for constructing a likelihood fit to extract the expected signal significance within the statistical analysis. The latter is known as the shape-based analysis of this variable, consisting of 10 bins ranging from 0 to 1, as shown in Figure 10 and Figure 11. This variable is the most effective in discriminating the background from the signal.

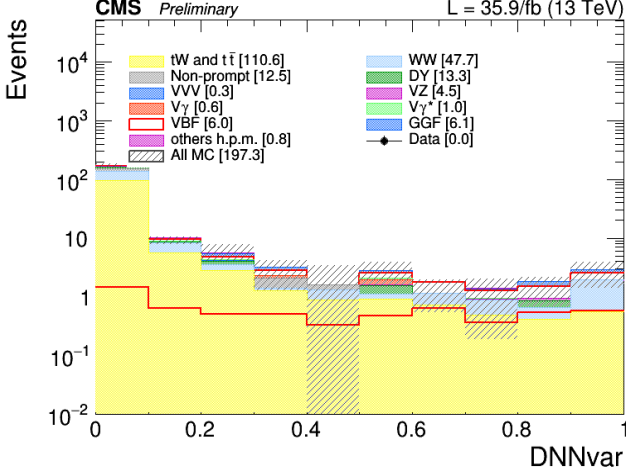


Figure 10. DNN variable for low m_{jj} regions after all the selections of the 2 jets VBF analysis with 2016 data.

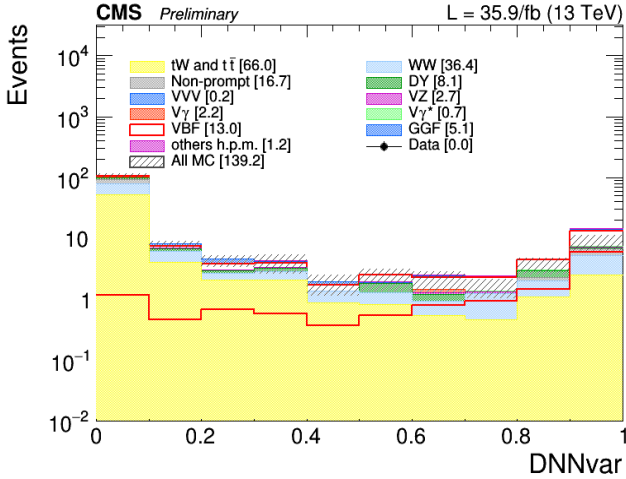


Figure 11. DNN variable for high m_{jj} regions after all the selections of the 2 jets VBF analysis with 2016 data.

In the VBF analysis using DNN, we combined the expected significance (Z) measurement for the low and high m_{jj} regions without the $\Delta\eta_{jj}$ cut: $Z_{VBF} = 2.2\sigma$ and $Z_{Higgs} = 2.8\sigma$.

The latter is a crucial result compared to the CMS experiment results [22] that study only the cuts analysis, a standard selection process, whose results are less significant, using $\Delta\eta_{jj}$ cut: VBF analysis without DNN: $Z_{VBF} = 1.5\sigma$ and $Z_{Higgs} = 2.2\sigma$.

The main reason for developing a deep neural network is to effectively suppress the background in measuring VBF Higgs production in the full leptonic channel. Developing an algorithm to construct a signal and background discriminant accomplished this task. We trained a set of simulated signal and background events to identify kinematic differences between the two processes. Therefore, we design each event to be more or less compatible with the signal or background topology. In this way, it is possible to improve the discrimination between signal and background with the purpose of background rejection and that the signal efficiency is both high and uniform for the signal process studied.

Using the tool, we combined the two studied subcategories (low and high m_{jj}) and extracted the results while considering all the correlations among the phase spaces. We turn the integrated data card into a RooFit workspace that implements a physics model defining the parameters of interest. We used the same combined data card to create different workspaces based on various physics interpretations of the results. Both signals in the two categories are scaled and used to quote the overall significance of the analysis. Using the Profile-Likelihood algorithm of the combined tool, they obtained the signal significance of both subcategories by fitting each subcategory at a time.

IV. CONCLUSIONS

We successfully trained a neural network to separate the main backgrounds from the VBF signal. We also extracted the expected signal significance by conducting a statistical analysis of the yields.

Moreover, the expected significance obtained with a cut-based analysis has been 1.4σ and 2.0σ for the VBF and Higgs signal, respectively.

Finally, we performed a more robust statistical analysis with a likelihood fit on the shape of the DNN classifier score, obtaining a result of 2.2σ and 2.8σ for the VBF and Higgs signal, respectively.

REFERENCES

- [1] Schmidhuber, J. (2015). Deep Learning In Neural Networks: An Overview. Neural Networks, 61, 85-117.
- [2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
- [3] N. Praveena and K. Vivekanandan, "A Review on Deep Neural Network Design and Their Applications," 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2021, pp. 1495-1501, doi: 10.1109/ICACCS51430.2021.9441826.

- [4] M. Carvalho and M. Pratama, "Improving shallow neural network by compressing deep neural network," 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 2018, pp. 1382-1387, doi: 10.1109/SSCI.2018.8628686.
- [5] Arend, L., Han, Y., Schrimpf, M., Bashivan, P., Kar, K., Poggio, T.A., DiCarlo, J.J., & Boix, X. (2018). Single units in a deep neural network functionally correspond with neurons in the brain: preliminary results.
- [6] Nguyen, A.M., Yosinski, J., & Clune, J. (2016). Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. ArXiv, abs/1602.03616.
- [7] Dinov, I.D. (2018). Deep Learning, Neural Networks. In: Data Science and Predictive Analytics. Springer, Cham. https://doi.org/10.1007/978-3-319-72347-1_23
- [8] Duca, V.D. (2003). Higgs Production at LHC.
- [9] ATLAS collaboration. (2018). Measurements of gluon-gluon fusion and vector-boson fusion Higgs boson production cross-sections in the $H \rightarrow WW^{*} \rightarrow e\nu/\mu\nu$ decay channel in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector. arXiv preprint arXiv:1808.09054.
- [10] Spira, M. (2017). Higgs boson production and decay at hadron colliders. Progress in Particle and Nuclear Physics, 95, 98-159.
- [11] Aad, Georges et al. "Search for a Charged Higgs Boson Produced in the Vector-Boson Fusion Mode with Decay $H(\pm) \rightarrow W(\pm)Z$ using pp Collisions at $\sqrt{s} = 8$ TeV with the ATLAS Experiment." Physical review letters 114 23 (2015): 231801 .
- [12] Wang, Y., Kang, Y., Qin, C., Wang, H., Xu, Y., Zhang, Y., & Fu, Y. (2021). Adapting stepsizes by momentumized gradients improves optimization and generalization.
- [13] Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., ... & Zecchina, R. (2019). Entropy-sgd: Biasing gradient descent into wide valleys. Journal of Statistical Mechanics: Theory and Experiment, 2019(12), 124018.
- [14] Keskar, N. S., & Socher, R. (2017). Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628.
- [15] Nanni, L., Maguolo, G., & Lumini, A. (2021). Exploiting Adam-like Optimization Algorithms to Improve the Performance of Convolutional Neural Networks. ArXiv, abs/2103.14689.
- [16] N. Srivastava, ET AL., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research 15 (2014) 1929-1958. <http://jmlr.org/papers/v15/srivastava14a.html>.
- [17] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", 2014. <http://adsabs.harvard.edu/abs/2014arXiv1412.6980K>, arXiv:1412.6980.
- [18] N. Srivastava, ET AL., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research 15 (2014) 1929-1958. <http://jmlr.org/papers/v15/srivastava14a.html>.
- [19] C. Francois, "keras", GitHub repository, 2015. <https://github.com/fchollet/keras>.
- [20] The Theano Development Team, ET AL., "Theano: A Python framework for fast computation of mathematical expressions", ArXiv e-prints (2016). <http://adsabs.harvard.edu/abs/2016arXiv160502688T>, <https://arxiv.org/abs/1605.02688>.
- [21] A. Abadi, ET AL., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems", Software available from tensorflow.org, 2015. <https://www.tensorflow.org/>.
- [22] CMS Collaboration., CMS Physics Analysis Summary, "Higgs to WWmeasurements with 15.2 fb⁻¹ of 13 TeV proton-proton collisions" <https://cds.cern.ch/record/2273908/files/HIG-16-021-pas.pdf>