2021-01

# DEEP LEARNING APPLIED IN THE CLASSIFICATION OF EVENTS GENERATED AT THE ATLAS EXPERIMENT

RODRIGUEZ MORA, JOHN IGNACIO

Universidad Técnica Federico Santa María
Departamento de informática
Valparaíso, Chile


# “Deep learning applied in the classification of events generated at the ATLAS experiment”

John Ignacio Rodríguez Mora

Thesis to apply for the professional title of
Ingeniero Civil en Informática

Guide: Raquel Pezoa, Ph.D.
Co-guide: Edson Carquín, Ph.D.
Member of dissertation committee: Claudio Torres, Ph.D.

13 January 2021

# Dedication

Dedicated to my little brothers Jeremy Rodríguez and Sebastian Rodríguez, and my mom Paola Mora, who always support me in any decision and project I made.

# Acknowledgment

I want to thank my friends and comrades Toni Alcayde, Maca Andrade, Javi Arce, Pedro Arriagada, Seba Bórquez, Pato Campaña, Claudia Fabbi, Carina *Juanita de Arco* Flores, Pablito Flores, Braulio *Pomposo* Fuentes, Dani Hebel, Cris Missana, Basti Quezada, Fabi Salas, Mati Valenzuela, Farid Zalaquett, Javierito Zavala and everyone that have being by my side in this journey through University. Love you all.

Thanks to my supervisor Raquel Pezoa for guiding me through this project and providing me knowledge and support every time I needed. Thanks to Edson Carquin and Claudio Torres for being part of this project too. Thanks to my teachers Cecilia Reyes and Hubert Hoffmann for their incredible work at the University, who always care about the wellbeing of their students.

Last but not least, special thanks to the project FONDECYT 3190740 for funding this thesis.

# Abstract

***Abstract —*** A Toroidal LHC Apparatus (ATLAS) is one of the two general purpose detectors at the Large Hadron Collider (LHC), at Conseil Européen pour la Recherche Nucléaire (CERN). Inside this, bunches of protons collide with a frequency of 40 MHz, and each collision, or event, can produce huge amounts of particles. The classification of LHC events is one of the most important analysis tasks in HEP, and a fundamental work for searching new phenomena. This work is focused in boosted di-Higgs decaying into $b\bar{b}\tau^{+}\tau^{-}$, handled as a classification task using deep learning techniques. Many models were trained, and the best 9 of them are tested, evaluated and compared. The best model resulted from taking an approach with Parameterized Neural Networks (PNN) and Cost-sensitive learning, specifically increasing the background class weight. Scores with these techniques reached above 0.9 F1-score on both background and signal classes. This work is a computer science study in collaboration with the Physics Department and CCTVal.

***Keywords —*** CERN, ATLAS, boosted di-Higgs, deep learning, event classification, LHC, class imbalance learning

# Acronyms

**A1** Autoencoder 1. x, 35, 42–44, 68

**A2** Autoencoder 2. x, 35, 42, 43, 45, 68

**ADA** ATLAS Data Analysis. 28

**ADASYN** Adaptive synthetic sampling. 21, 36, 45, 64, 68

**AI** Artificial Intelligence. 9, 10

**ALICE** A Large Ion Collider Experiment. 3

**ANN** Artificial Neural Networks. 5, 7, 9, 13, 14, 16, 22, 31

**ATLAS** A Toroidal LHC Apparatus. iv, ix, 1, 3–5, 7–9, 24, 64, 68

**AUROC** Area Under the ROC curve. 20

**BC1** Binary Classifier 1. ix, 32, 34, 35, 38–40, 54, 55, 57, 68

**BC1-MP** Mass Parameterized BC1. ix, 41, 68

**BC2** Binary Classifier 2. 33

**BC3** Binary Classifier 3. 33

**BC4** Binary Classifier 4. ix, 33–37, 40, 46, 49, 52, 54, 55, 57, 59, 60, 62, 64, 68

**BC4-ADASYN** BC4 with ADASYN. x, 46, 68

**BC4-CW** BC4 with Class Weights. x, 47, 68

**BC4-CW-MP** Mass Parameterized BC4 with Class Weights. x, xii, 47, 63, 65, 68

**BC4-MP** Mass Parameterized BC4. x, 41, 68

**BDT** Boosted Decision Trees. 1, 5–9, 64, 65, 68

**CCTVal** Centro Científico Tecnológico de Valparaíso. iv, 69

**CERN** Conseil Européen pour la Recherche Nucléaire. iv, 3, 7, 64

**CMS** Compact Muon Solenoid. 3, 5, 9

**DNN** Deep Neural Networks. ix, 1, 9, 22, 31–34, 36, 64, 68

**EWSB** Electro-Weak Symmetry Breaking. 1

**FFNNA** Feed-Forward Neural Network Architecture. 33

**FN** False Negative. 18, 19

**FP** False Positive. 18, 19

**HEP** High energy physics. iv, 4–7, 9

**LHC** Large Hadron Collider. iv, 1, 3–9, 17, 24

**LHCb** Large Hadron Collider beauty. 3

**LWTNN** Lightweight Trained Neural Network. 65–67

**ML** Machine Learning. 9–11, 20, 68

**MVA** Multivariate Data Analysis. 65

**PNN** Parameterized Neural Networks. iv, 16, 22, 35, 36, 51, 65, 67–69

**ROC** Receiver Operating Characteristic. 19, 20

**SMOTE** Synthetic Minority Over-sampling Technique. 21, 36, 64

**SR** Signal Region. 25, 38

**SR0** Signal Region 0. ix, xi, 6, 27, 38

**SR1** Signal Region 1. ix, xi, 6, 27, 28, 39

**SR2** Signal Region 2. ix–xii, 6, 21, 27, 28, 36, 38–63, 66

**SVM** Support vector machines. 7

**TMVA** Toolkit for Multivariate Data Analysis. 7

**TN** True Negative. 18

**TP** True Positive. 18, 19

# Contents

# List of Figures

# List of Tables

# Introduction

Which are the fundamental particles that form matter? What gives mass to these fundamental particles? These are two questions that physicists and scientists have tried to answer over the years. To answer the first question, a *Standard Model* of particle physics was developed, which defines the fundamental forces and elementary particles [Oerter and Holstein, 2006] that give shape to the universe. To answer the second question, the *Higgs mechanism* [Higgs, 1964] was developed, which allow bosons and fermions, elementary particles that form matter and transmit the forces by which they interact, to acquire a mass. From here, the concept of *Higgs boson* was born. The Higgs boson is the excitation of the quantum field whose coupling with standard model particles allow them to acquire a mass after spontaneous Electro-Weak Symmetry Breaking (EWSB).

One of the objectives of the LHC was to find evidence of the existence of the Higgs boson. To accomplish this goal, protons beams are collided against each other at high energies, in an effort to disintegrate these particles and turn them into different ones. Four big experiments are mounted around the interaction points of the LHC, each of them built with their own design and with different purposes. These experiments record the physics properties of the many particles produced by the collisions, generating terabytes of data per second.

Can the Higgs boson be identified directly by the detectors? The answer is no. If a Higgs boson is produced in a proton-proton collision, it instantly disappears by decaying into other particles due to its short lifetime. Moreover, the Higgs boson identification task could get even harder since there are other procedures that could decay to the same final state particles. Moreover, in this project case of interest not only one but two Higgs bosons could be produced by the decay of a new heavy particle. This happens when the mass of this particle is very large (> 1TeV) and, as a result of this, the decay products of the Higgs are collimated[1] along the Higgs direction and partially overlapped in the detector.

In the present body of work, data collected by the ATLAS experiment is going to be analyzed to identify boosted di-Higgs decaying into two *quarks bottom* $b\bar{b}$ and two *leptons tau* $\tau^+\tau^-$ events. The approach that has given the best results to identify these events in the past is the so called Boosted Decision Trees (BDT) [Rodriguez, 2019], having let Deep Neural Networks (DNN) in a second place, until now. Multiple DNN models are developed and trained in this project, having special focus on class imbalance issues and generalization, which have overcome past results with BDT.

In Chapter 1 context about the problem is presented, defining some physics concepts that form the base of this project, goals are specified and the contribution is stated. In Chapter 2 the state of the art is described, machine learning concepts are defined, DNNs are further explained and the class imbalance issue is approached. In Chapter 3 solutions are proposed, addressing ROOT files management, data analysis, data processing, model building and how the class imbalance issue was handled. In Chapter 4 models are trained, plotting training and validation loss over epochs. In Chapter 5 models are evaluated and compared between

[1] collimated: (of rays of light or particles) made accurately parallel.

each other, using different metrics that help to mitigate class imbalance. Finally, in Chapter 6 conclusions and highlights are presented, including future work.

# Chapter 1
# Problem definition

## 1.1 Context

The LHC at CERN, the European Organization for Nuclear Research, is the world's largest and most powerful particle accelerator [Evans and Bryant, 2008]. This huge collider consists of a 27-kilometer ring with superconducting magnets, located underground across the border between Switzerland and France, illustrated in Figure 1. Inside the LHC, beams of protons collide 40 million times per second, with a center of mass energy of $13$ TeV, of which only an enriched high energy sample of 1 kHz is saved for further analysis.

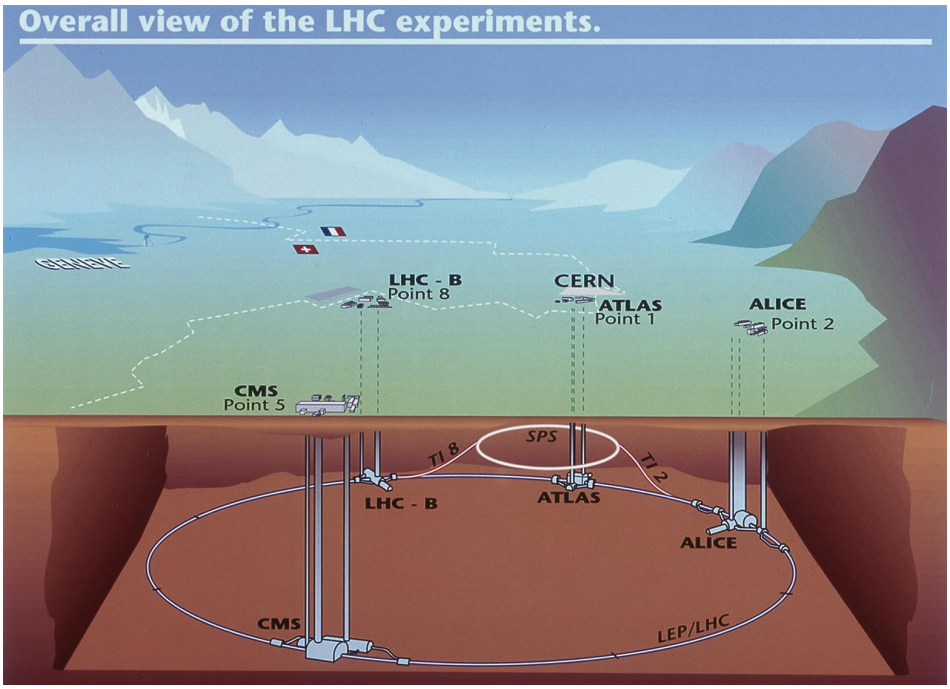



Figure 1: LHC and the location of its experiments. Image source `http://cdsweb.cern.ch/`.

ATLAS is one of the four main detectors of the LHC, alongside Compact Muon Solenoid (CMS), A Large Ion Collider Experiment (ALICE) and Large Hadron Collider beauty (LHCb). It is 40 m long, 25 m high and weights around 7000 t [ATLASCollaboration, 2008]. It has been measuring and registering the products from proton-proton collisions since 2010, and observed a particle that we now know to be the Higgs boson in 2012 [ATLASCollaboration, 2012]. The detector is formed of cylindrical layers at the centre of which lies the interaction point, which is the place where the two beam pipes intersect.

During operation both pipes contain beams of protons travelling at near the speed of light in opposite directions, the protons collide with a centre of mass energy of $13$ TeV. Figure 2 shows a cut-away representation of the detector revealing some of the internal structure.

Each layer is different in the way that it detects particles. Radiation trackers made of silicon or filled with Xenon, calorimeters and muon detectors are some of the particular components used in ATLAS. Data from these components can be used to measure physical properties of the particles decaying in the beam pipe. For example by observing the curvature of charged particles through the radiation trackers their momentum can be calculated. Measurements like this can be thought of as primitive data, another type of data, known as derived, is acquired by combining primitives.

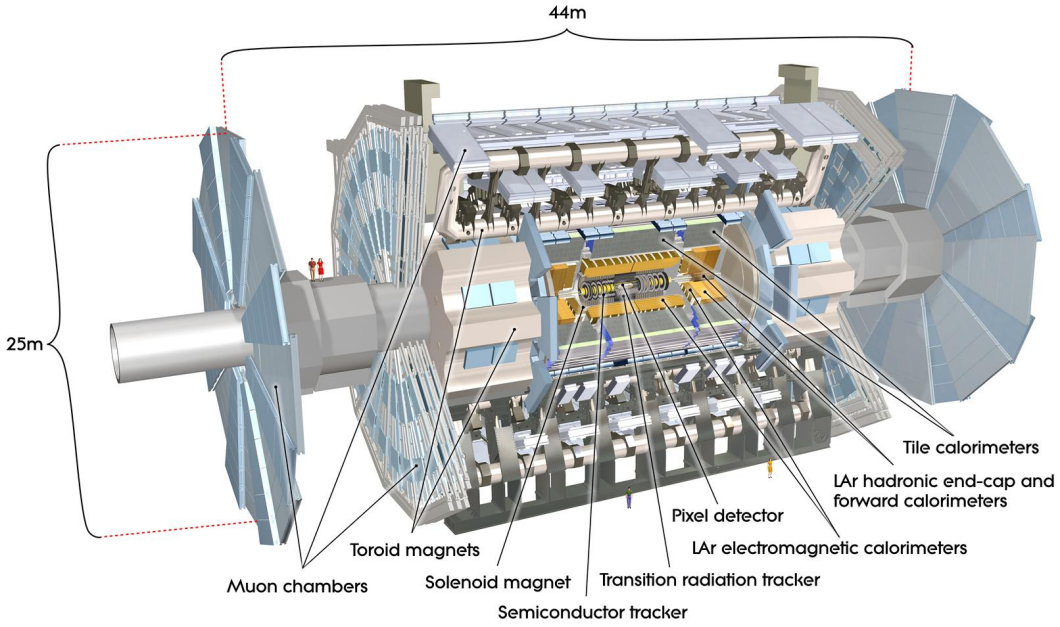



Figure 2: The ATLAS detector at LHC is a multipurpose particle physics experiment, with cylindrical geometry and a coverage in solid angle of nearly $4\pi$. The detector is 25 meters in height and 44 meters in length with an overall weight of approximately 7000 tonnes [ATLASCollaboration, 2008]. Image source: `http://cdsweb.cern.ch/`.

Each *event*, where a bunch of protons are crossing and many proton-proton collisions can occur, produces a huge amount of data, which is filtered using the ATLAS Trigger System [Aaboud et al., 2017], a critical component of any collider experiment that contains specialized algorithms to keep only the *interesting* events from the point of view of HEP. Despite this data reduction, the selected events still represent a very large amount of data. In fact, the ATLAS experiment keeps approximately $1$ out of $100,000$ events, which contributes to the generation of more than $20$ Pb per year [Radovic et al., 2018] during data taking.

The LHC and its experiments are designed to answer some of the fundamental questions of our universe such as what we are made of and how particles obtain their mass. This and all known fundamental particles —particles without internal structure— to date, as well as the electromagnetic, weak nuclear, and strong nuclear forces are very accurately described by the so-called *standard model* of particle physics [Sonneveld, 2019].

This model (see Figure 3) was developed in the early 1970s, and it has successfully explained

almost all experimental results and precisely predicted a wide variety of phenomena. Over time and through many experiments, the standard model has become established as a well-tested physics theory[2].

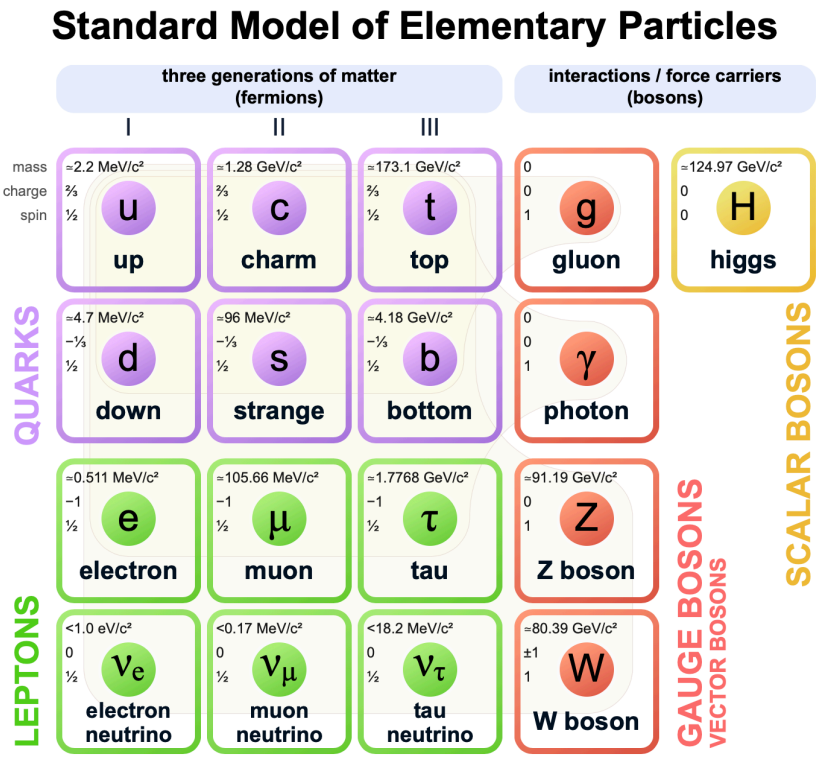



Figure 3: The standard model. Image source: `https://en.wikipedia.org/`.

It has been eight years since ATLAS [Aad et al., 2012] and CMS [Chatrchyan et al., 2012] announced the discovery of the Higgs boson. However, the Higgs boson remains the most mysterious of the known particles. A key interaction not yet observed by LHC experiments is the production of "a pair of Higgs" or di-Higgs. The Standard Model predicts that the Higgs field can interact with itself to create a Higgs boson pair. The rate with which this happens is critical, as it allows physicists to directly probe the potential energy of the Higgs field, which is responsible for giving mass to particles. Deviations from the expectation would be a strong hint of new physics.

## 1.2 Problem Description

Machine learning is already playing a role in the analysis of HEP data generated at the LHC. Traditionally, the most used techniques are shallow Artificial Neural Networks (ANN) and Boosted Decision Trees (BDT) [Albertsson et al., 2019]. The deep learning revolution has produced a significant impact in many fields, including HEP, bringing in higher levels of classification performance. However, deep learning algorithms still have substantial room for

[2]https://home.cern/science/physics/standard-model

improvement, especially in the context of HEP events classification, i.e. separating *signal* events from *background*. This classification is one of the most important analysis tasks in HEP, and a crucial stage in search of new phenomena in this field. Interesting events produced when protons collide inside the LHC are infrequent. In terms of machine learning classification, this means the number of samples belonging to the class of interest (or the positive class) is much smaller than the number of samples in the negative class. Hence, data naturally exhibit a vast imbalance in their class distribution. For instance, the Higgs boson is only produced in roughly one out of a billion LHC collisions and live only a small fraction of a second before their energy is converted into other particles.

This work aims to contribute —from the computer science point of view— in the analysis of the Higgs-boson pair (di-Higgs) production. The Higgs pair decays that are currently analysed in ATLAS are $b\bar{b}\tau^+\tau^-$, $b\bar{b}W^+W^-$, $b\bar{b}\gamma\gamma$, or $b\bar{b}, b\bar{b}$. Particularly, this work will be focused on the classification of events into two classes: (i) signal di-Higgs decaying into $b\bar{b}\tau^+\tau^-$, and (ii) background, using deep learning techniques.

The class imbalance problem will be analyzed in this project considering that, even if the class of interest (di-Higgs signal) is quite rare in reality, in simulated data the signal class can be majority or minority depending on the region being observed. There are three regions of interest in this body of work: Signal Region 0 (SR0), Signal Region 1 (SR1) and Signal Region 2 (SR2), all of them having different ranges for event variables. Special emphasis is taken on SR2, region where all the models proposed as solutions are trained and evaluated.

An important issue is that traditional machine learning approaches, including deep learning, usually assume class distributions which do not deviate too much from the balanced case. Thus, even deep learning techniques are affected by the highly imbalanced relevant data collected by the detectors at LHC.

It is worth to mention the concepts event, signal, and background will be understood as follows:

- *Event*: moment after the occurrence of a fundamental interaction between subatomic particles. For example, a collision between two particles.
- *Signal*: Event of interest from the physical point of view. In this work, the signal to be analyzed corresponds to the production of a pair of Higgs bosons (or di-Higgs) decaying to $b\bar{b}\tau^+\tau^-$.
- *Background*: Other events that could mimic the signal, either by the effect of the misidentification of particles or by another process that produce the same final state as the signal.

## 1.3 Current Solutions

This problem has being recently approached in [Rodriguez, 2019], where Boosted Decision Trees (BDT) were used to identify the event of interest. This is the only formal solution to

this problem to this day, apart from the one developed in this thesis. BDT is one of the most popular techniques to classify events in HEP, which is implemented in a computational framework vastly used by CERN annalists called ROOT [Brun and Rademakers, 1997].

ROOT is an open source framework developed in C++ implemented by CERN, which contains interactive tools for data analysis, data simulation and events generation. To analyze generated data in LHC in ROOT, Toolkit for Multivariate Data Analysis (TMVA) was created, which contains tools to perform multivariate analysis of HEP data in general, including support for Artificial Neural Networks (ANN), Support vector machines (SVM), decision tree learning, between others. For Python users, there is a module called PyROOT that allows to interact with ROOT and TMVA classes.

## 1.4 Goals

### 1.4.1 Main Goal

The main goal of this project is to design, implement and validate a deep learning-based algorithm to classify events generated by the ATLAS Experiment at CERN to identify the boosted di-Higgs production that decays into $b\bar{b}\tau^{+}\tau^{-}$ particles, taking into account the class imbalance problem, the validation metrics in the experimental HEP context, and the validation of the algorithm in the ATLAS analysis framework.

### 1.4.2 Specific goals

1. Form a particle physics knowledge base in the ATLAS experiment context.
2. Check the machine learning state of the art in the ATLAS experiment context.
3. Analyze the class imbalance problem.
4. Generate a proper dataset using the computational tools of the ATLAS experiment.
5. Validate using the standard metrics of machine learning and the performance metrics used in the particle physics context.
6. Validate the algorithm in the *ATLAS Analysis framework*.
7. Define appropriate deep learning techniques to develop the algorithm that helps in the identification of events corresponding to the production of a boosted di-Higgs bosons over the background.

## 1.5 Contribution

The main contribution of this work is the development of a new algorithm to analyze collisions generated by the LHC and and the products of these collisions which are detected and recorded by the ATLAS experiment. Particularly, this work will be focused on the classification of ATLAS events to identify the boosted di-Higgs signal decaying into the $b\bar{b}\tau^{+}\tau^{-}$ particles. The deep learning approach will be used to design, implement, and validate the proposed method. The validation is a crucial step in this work, because the proposed algorithm will be tested on the *ATLAS analysis framework*, which means that this work will be a direct contribution to the ATLAS Collaboration.

Every deep learning model built is stored in the following repository [Rodriguez, 2020a]. Additionally, tools to read ROOT files, process data, train models and plot evaluations can be found there. Tools to test this models is also being developed in the ATLAS analysis system, which are being added in this repository [Rodriguez, 2020b]. Support for deep learning and BDTs approaches can be found there.

# Chapter 2
# Conceptual Framework

The present chapter presents state of the art related to studies made in ATLAS and LHC about identification tasks and machine learning. Also defines concepts of Machine Learning (ML), deep learning, Artificial Neural Networks (ANN) and class imbalance learning.

## 2.1 State of the art

Machine Learning (ML) has been fundamental to obtain successful results in classification of LHC generated events. Works as [P.Chiappettaa et al., 1994] show the search of the Higgs boson with Artificial Neural Networks (ANN), by classifying into background and Higgs boson production. In [Roe et al., 2005], Boosted Decision Trees (BDT) and ANN are used to identify neutrinos oscillations in Fermilab, improving the results using adaboost.

In past decades, ANNs weren't the most popular choice to perform classification tasks, due to the lack of power of computers in those years. But now, thanks to the advances on computation, ANNs are making a comeback. Multiple deep learning algorithms and techniques have been applied to LHC physics and the ATLAS experiment, as stated in [Guest et al., 2018], in tasks like event selection, jet classification, tracking and fast simulation. Moreover, when the Higgs boson was discovered, both ATLAS and CMS contributed using ML algorithms applied to data from collisions of particles [ATLASCollaboration, 2012].

Additionally, the use of machine learning its not only restricted to the study of HEP, but also it is used in data management in general, as shown in [Kuznetsov et al., 2016], where predictions about data access where made based on CMS meta-data, or also in [Bonacorsi et al., 2015], where data transfer latency is monitored and machine learning is used to predict congestion.

Lastly, a recent study on this same matter, identifying boosted di-Higgs decaying into $b\bar{b}\tau^+\tau^-$, was made [Rodriguez, 2019], with special emphasis on BDT, getting 85% of F-score. DNNs are also trained in this project, but their results weren't as good as the ones obtained with BDT, mainly because a deeper research on this matter was needed. The goal of the present body of work is to continue with this project and deepen into DNNs, to improve previous results on the identification of this rare event and contribute to deep learning research.

## 2.2 Machine Learning

Machine learning is a field inside Artificial Intelligence (AI) that arises from the questions: could a computer go beyond "what programmers know how to order it to perform" and learn on its own how to perform a specified task? Rather than programmers crafting data-processing rules by hand, could a computer automatically learn these rules by looking at data [Chollet, 2017].

Some definitions to Machine Learning have been proposed by experts through time. Alan Turing's paper [Turing, 1950] introduced a benchmark standard for demonstrating machine intelligence, such that a machine has to be intelligent and responsive in a manner that cannot be differentiated from a human being:

*Machine Learning is an application of artificial intelligence where a computer/machine learns from the past experiences (input data) and makes future predictions. The performance of such a system should be at least human level.*

A more technical definition given by Tom M. Mitchell's [Mitchell, 1997]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*. For instance, in a handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified, accuracy
- Training experience E: a dataset of handwritten words with given classifications

In order to perform the task T, the system learns from the dataset provided. A dataset is a collection of many examples. An example is a collection of features.

As shown in Figure 4, before machine learning, AI was about inserting a big amount of rules into a system, so it could respond given some data (Symbolic AI) [Chollet, 2017].

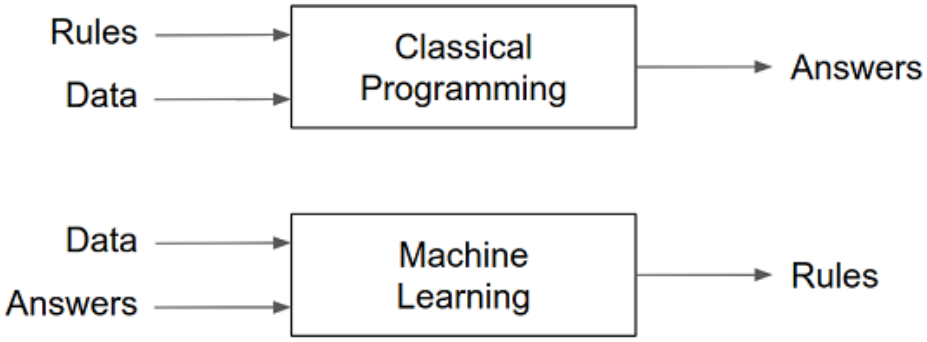



Figure 4: ML input versus Classical Programming input. Image source: [Chollet, 2017].

A ML model is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task. For instance, automating the task of tagging vacation pictures can be done with a ML system with many examples of pictures already tagged by humans, by learning statistical rules for associating specific pictures to specific tags.

### 2.2.1 Key elements

To develop algorithms based on ML, three things are needed:

- **Input data points**. For instance, if the task is speech recognition, these data points could be sound files of people speaking. If the task is image tagging, they could be pictures.
- **Examples of the expected output**. In a speech-recognition task, these could be human-generated transcripts of sound files. In an image task, expected outputs could be tags such as dog, cat, and so on.
- A way to **measure whether the algorithm is doing a good job**. This is necessary in order to determine the distance between the algorithm's current output and its expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call learning.

### 2.2.2 Types of Learning

- **Supervised learning**. The machine experiences the examples along with the labels or targets for each one. The labels in the data help the algorithm to correlate the features. Two of the most common supervised machine learning tasks are **classification** and **regression**.
- **Unsupervised learning**. With unclassified and unlabeled data, the system attempts to uncover patterns from the data. There is no label or target given for the examples. One common task is to group similar examples together called **clustering**.
- **Reinforcement learning**. Refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal. For example, maximize the points won in a game over many moves.

The central problem in machine learning is to learn a good **representation** of the input data. A representation is a way of looking at the data, for instance, an image can be represented as a 2D matrix of RGB points or in the HSV format[3]. Some machine learning tasks might be easier with some representations than with others. So, technically, machine learning can be described as a search for a useful representation of some input data.

[3]HSV (Hue, Saturation, Value) is a color model that describes colors (hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value.

### 2.2.3 Evaluating models

In machine learning, the goal is to achieve models that **generalize**, i.e. that perform well on never-before-seen data, and **overfitting** is the central obstacle. To do this, splitting the available data is crucial. Training, validation and testing are the partitions needed, so during the training phase, the model trains with the training data and test with the validation data (Figure 5), and when the model is ready, it is tested one last time with the test data.

The reason why three (and not two) datasets are used is because a *tuning* of the model configuration is needed. The parameters that can be tuned in a machine learning model are called *hyperparameters*. Hyperparameter values can be changed to control the learning process, meanwhile, the value of other parameters, like node weights, are derived by training and cannot be adjusted. Hyperparameter tuning is done with the feedback of the model performance in validation data. A consequence of this is a possible overfitting to the validation set, that can be avoided by having a testing set, which haven't been seen by the model.

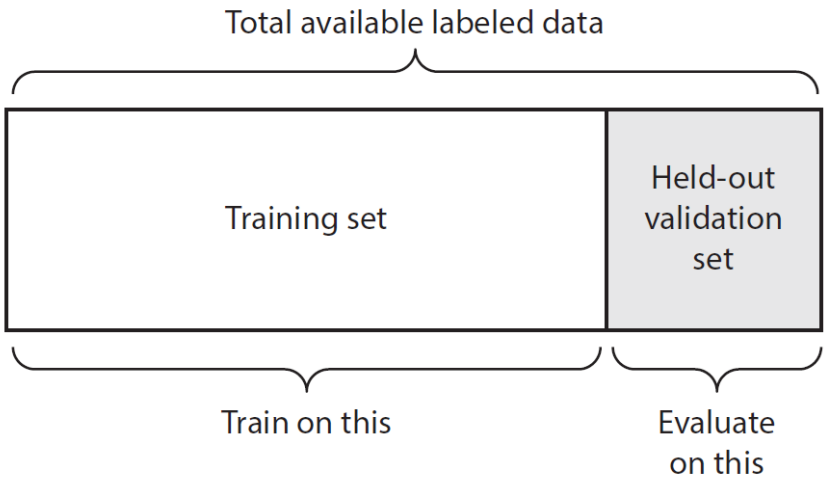



Figure 5: Splitting data to be used in training phase. Image source: [Chollet, 2017].

## 2.3 Deep Learning

In 1957, Frank Rosenblatt [Rosenblatt, 1957] defined the perceptron, an algorithm for supervised learning of binary classifiers. It receives some inputs, applies a weighted sum along with a transformation, and returns the result. This algorithm is consider the base of what we called **deep learning**.

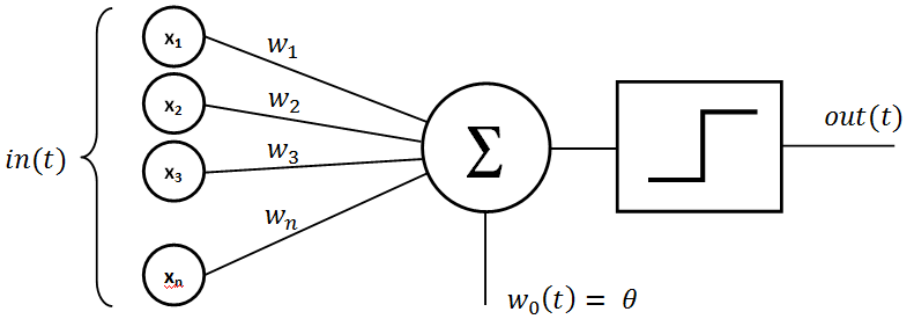



Figure 6: Perceptron by Frank Rosenblatt. Image source: `https://commons.wikimedia.org/wiki/File:Perceptron_moj.png`

Deep learning is a subfield of machine learning that presents a new take on learning representations from data, putting special emphasis on learning successive layers of increasingly meaningful representations. *Deep* stands for this idea of successive layers of representations, not a deeper understanding.

These layered representations are (almost always) learned via models called Artificial Neural Networks (ANN), structured in literal layers stacked on top of each other. These layers perform simple data transformations, learned by exposure to examples, one after another, until an output layer.

The specification of what a layer does to its input data is stored in the layer's **weights**. In technical terms, we'd say that the transformation implemented by a layer is parameterized by its weights. In this context, learning can be seen as finding a set of weights for each layer, such that the network will correctly map each example to its target.

But, how is it measured the correctness of the input? By observing this output and calculating how far it is from what we expect. This is the job of the **loss function**: it takes the predicted value and the expected value and computes a distance score, capturing how well the network has done in the specific example.

This score is used as a feedback signal to adjust the weights in a direction that will lower the loss score for that example. This adjustment is performed by the **optimizer** by implementing the backpropagation algorithm [Kostadinov, 2019], [Hecht-Nielsen, 1989]. The entire workflow is illustrated in Figure 7.

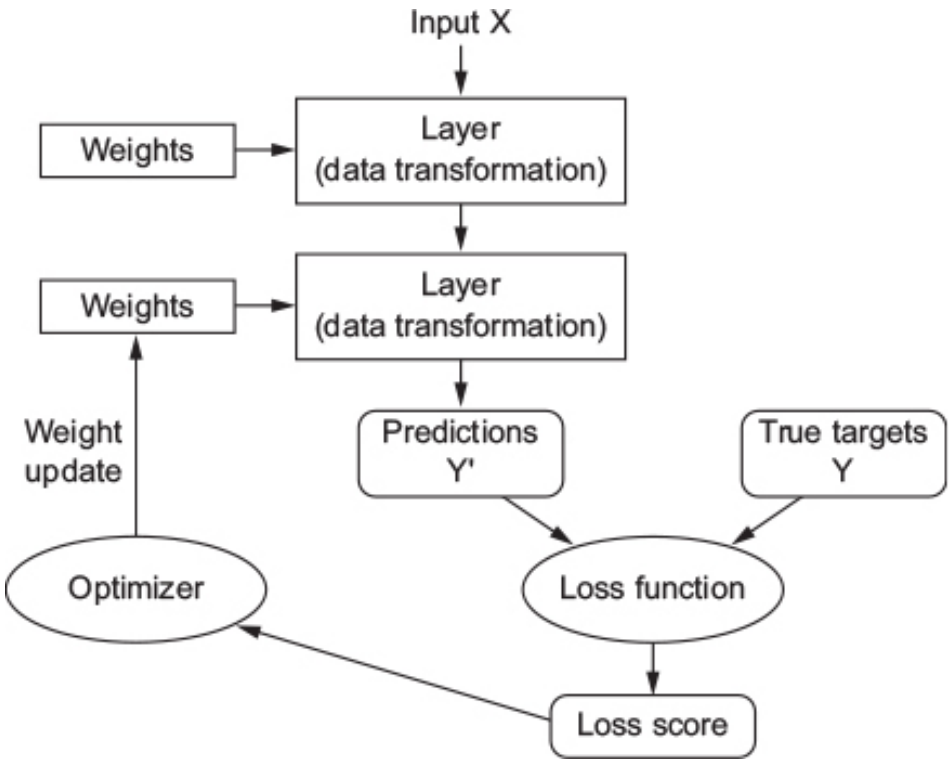



Figure 7: Neural network scheme. Image source: [Chollet, 2017].

The basic computation unit in an ANN is the **neuron**, which can be:

- **Input neuron**: Each original feature feeds this specialized unit.
- **Output neuron**: Unit that produces the final output.
- **Hidden neuron**: located between the input and output in charge of learning the underlying representation.

Layers can be composed by one or more neurons. Each neuron linearly combines the attributes generated in the previous layer (using its weights) and then transforms the total signal calculated using a non-linear function. This last function is called **activation function** [Goodfellow et al., 2016], and one between multiple alternatives can be selected. Given the activation function $\sigma$ that is applied to the linear combination of the neuron input, some examples activation functions are:

$$a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right) \tag{1}$$

- **Linear**. It leaves the liner combination intact.

$$\sigma(x) = x \tag{2}$$

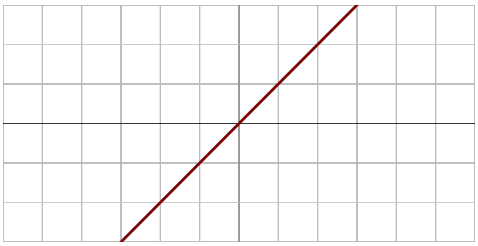


Figure 8: Linear activation function. Image source: `https://en.wikipedia.org/wiki/Activation_function`

- **Sigmoid**. It smooths the gradient preventing jumps in output value, and leaves the result between 0 and 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

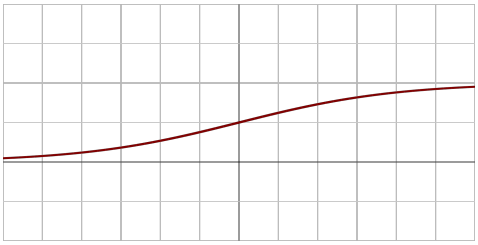


Figure 9: Logistic (also called Sigmoid or Soft step) activation function. Image source: `https://en.wikipedia.org/wiki/Activation_function`

- **TanH**: Zero centered function that goes from -1 to 1.

$$\sigma(x) = \tanh(x) \tag{4}$$

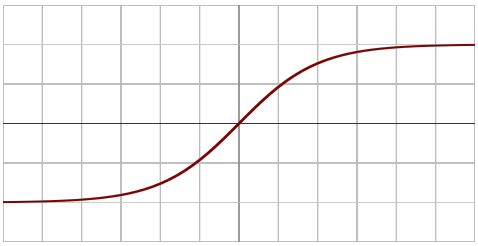


Figure 10: TanH activation function. Image source: `https://en.wikipedia.org/wiki/Activation_function`

- **ReLu** (Rectified Linear). It behaves like a linear activation function but turns negative values into zero.

$$\sigma(x) = \max(x, 0) \tag{5}$$

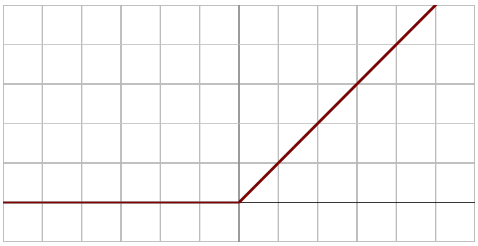


Figure 11: Rectified linear activation function. Image source: `https://en.wikipedia.org/wiki/Activation_function`

- **SoftPlus**: The idea of this function is to have a similar behaviour to ReLu, but being differentiable everywhere in the function.

$$\sigma(x) = \log(1 + e^x) \tag{6}$$

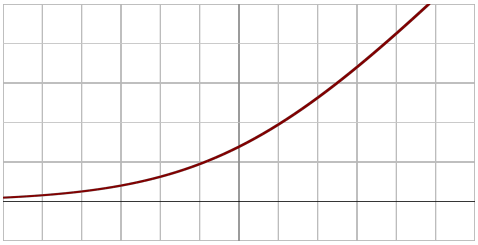


Figure 12: SoftPlus linear activation function. Image source: `https://en.wikipedia.org/wiki/Activation_function`

The last layer (output) must use activation functions appropriate for the learning task that wants to be solved. For example, in regression problems, where the output is continuous (position, speed, price, temperature, etc), an output to the range [0, 1] may be too restrictive. In regression settings the linear activation function is the most used.

## 2.4 Parameterized neural networks

Parameterized Neural Networks (PNN) are a special type of ANN defined in [Baldi et al., 2016], characterized by the way it generalize solutions. If multiple datasets need to be learned, then a solution would be to train an ANN model for each one of this datasets. This could bring great results in each dataset separately, but maintaining a lot of models can become tricky. PNN solves this issue by training just one model for all datasets, generalizing the solution. This is done by joining all datasets and adding a new column that indicates the property that defines each dataset.

PNNs had been tried originally in high-energy physics with great results [Baldi et al., 2016], and now has been implemented in the present body of work, by parameterizing the mass of the signal. Further details of this procedure is stated in Section 3.7.2.

## 2.5 Class imbalanced classification problem

As it was stated before, classification predictive modeling involves predicting a class label for a given observation. An **imbalanced classification problem** occurs when the distribution of examples across the known classes is biased [Kuhn and Johnson, 2013]. The distribution can vary from a slight bias to a severe imbalance, where there is one example in the minority class for hundreds, thousands, or millions of examples in the majority class.

Most of the classification algorithms were designed around the assumption of an equal or similar number of example for each class, that's why making an imbalanced classification can be a tough challenge. This results in models performing poorly when predicting, specifically for the minority class. Generally, in the real world the minority class is the most important one, and, therefore, the problem is more sensitive to classification errors for this class than the other ones.

Other, less general, names used to describe this problem are:

- Rare event prediction.
- Extreme event prediction.
- Severe class imbalance.

The class imbalance problem is defined by the distribution of classes, i.e. the number of examples that belong to each class, in a specific training dataset [He and Ma, 2013]. It is common to describe the imbalance of classes in a dataset in terms of ratios. For instance, an binary classification problem with an imbalance of 1 to 100 or 1:100. Another way to describe it would be using percentages of the training dataset. For example, 70 percent of the examples belong to the first class, 25 percent to the second class, and 5 percent to the third class.

As previously mentioned, the classification of the LHC events, or separating *signal* events from the *background* is one of the most important analysis tasks in HEP, and a crucial stage in search for new phenomena in this field. A difficulty in this classification task is the identification of the infrequent and relevant events, from the dominating background. For instance, a Higgs boson is produced only once every few billion proton-proton collisions at the LHC. This means the number of samples belonging to the class of interest (or the positive class) is much smaller than the number of samples in the negative class. Hence, the data naturally exhibit a vast imbalance in their class distribution, and this problem, will be analyzed in this thesis work.

First, evaluating the model performance is needed, and then, solutions need to be applied in order to improve this performance or get rid of the imbalance problem.

### 2.5.1 Performance metrics

Accuracy, while being an important and unavoidable metric for evaluating a trained classifier, can be misleading and therefore should be used cautiously and alongside other metrics [Fawcett, 2006]. A table that is widely used when dealing with classification problems is the **confusion matrix**.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It shows the ways in which the classification model is confused when it makes predictions. It gives insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

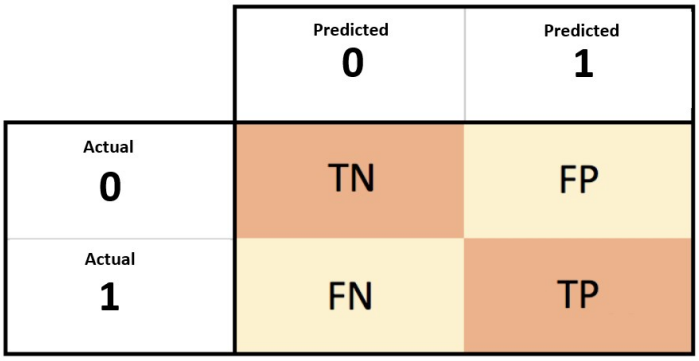



Figure 13: Confusion matrix for a binary classifier. Image source: `https://towardsdatascience.com/demystifying-confusion-matrix-29f3037b0cfa`

Figure 13 shows a confusion matrix for a binary classifier, which has some terms to be defined:

- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

From here, multiple metrics can be calculated:

- **Classification Rate or Accuracy**: As is stated before, accuracy has an issue: It assumes equal costs for both kinds of errors, which is not correct in imbalanced problems.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

- **Recall**: Ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (a small number of FN).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{8}$$

- **Precision**: Total number of correctly classified positive examples divided by the total number of predicted positive examples. High Precision indicates an example labelled as positive is indeed positive (a small number of FP).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{9}$$

  Having **high recall and low precision** means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives. On the other hand, having **low recall and high precision** shows that a lot of positive examples are missed (high FN) but those we predict as positive are indeed positive (low FP).

- **F1**: Represents both recall and precision. It is calculated using harmonic mean in place of arithmetic mean as it punishes the extreme values more. The F1 will always be nearer to the smaller value of Precision or Recall.

$$F_1 = 2 \cdot \frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \tag{10}$$

Another metric is the Receiver Operating Characteristic (ROC) curve, defined with respect to a given class $C$. Given a point $x$ and model that outputs a $P(C|x)$ probability that $x$ belongs to $C$. Given $T$, a threshold, $x$ belongs to $C$ if and only if $P(C|x) \geq T$. If $T = 1$, a point is labelled as belonging to $C$ only if the model is 100% sure. If $T = 0$, every point is labelled as belonging to $C$.

Each value of the threshold $T$ generates a point (FP, TP) and, then, the ROC curve is the curve formed by going through $T = 0$ to $T = 1$. A good model will have a curve that increases quickly from 0 to 1, meaning that only a little precision has to be sacrificed to get a high recall, as can be seen in Figure 14.

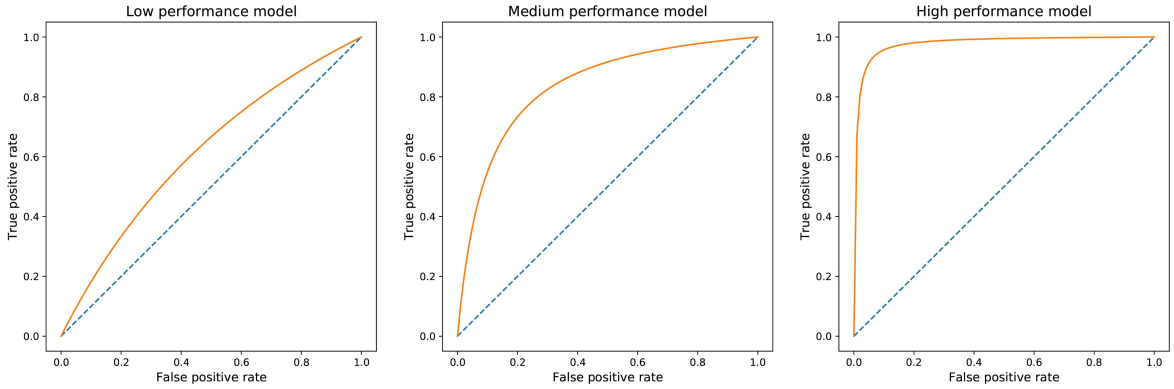



Figure 14: ROC curves depending on the effectiveness of the model. Image source: `https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28`

Based on the ROC curve, another metric can be built that is easier to use to evaluate the model: the Area Under the ROC curve (AUROC). AUROC acts a little bit as a scalar value that summarises the entire ROC curve. As it can be seen, the AUROC tend towards 1.0 for the best case and towards 0.5 for the worst case. Here again, a good AUROC score means that the model evaluated does not sacrifice a lot of precision to get a good recall on the observed class (often the minority class).

### 2.5.2 Machine learning for class imbalanced data

Different ML techniques have been tried over the years to handle class imbalanced problems, from modifying the training dataset to decrease imbalance, to modifying the model's learning algorithm to increase sensitivity towards the minority class. Such techniques are grouped in three big categories: data-level, algorithm-level and hybrid methods [Johnson and Khoshgoftaar, 2019].

#### Data-level methods

Data-level methods like **oversampling** and **undersampling** are techniques that modify the training distribution in order to partially or fully balance the dataset, decreasing class imbalance.

Undersampling discards data from the majority class, meanwhile oversampling will create synthetic data for the minority class. Both of these techniques could bring new issues to the table if they are not done properly. Discarding data decreases the information to learn and adding data can cause overfitting. Due to this problems, intelligent methods have been developed for both techniques: intelligent undersampling methods aim to preserve relevant information; and intelligent oversampling methods target overfitting reduction.

Since the event that has to be identified is a very rare phenomenon, simulated data has to

be used to train and test machine learning procedures. This data has different distribution in each simulated environment, differing from the real distribution of the signal. Because of this, undersampling methods are not further researched. SR2 is a region of interest where there are not many samples to learn, so decrease them would harm severely the learning process. On the other hand, oversampling methods are further research to increase the amount of background samples on SR2, since signal is the majority class. In reality, signal is the minority class with a severe imbalance, but since this environment has simulated data, more signal events have been created. To reflect reality, each event has a weight that indicates the probability of its occurrence. Therefore, signal events have a smaller weight than background events. In deep information about simulated data can be found in Section 3.1.

Intelligent oversampling methods include Synthetic Minority Over-sampling Technique (SMOTE) and Adaptive synthetic sampling (ADASYN), with ADASYN being part of the proposed solution in Section 3.8.1, trained and validated in Section 4.4.1, and evaluated in Section 5.1.1.

**Algorithm-level methods**

Algorithm-level methods don't alter the training data distribution, like data-level methods, instead, the learning or decision process is modified in a way that increases the relevancy of the minority class. Generally, algorithms are modified to take a class penalty or weight into consideration, or the decision threshold is changed in a way that helps the minority class and reduces bias towards the majority class.

Cost-sensitive learning is part of the algorithm-level methods, where penalties are assigned to each class through a cost matrix. Increasing the cost of the minority class means that its importance is increased, therefore, the likelihood that the learner will incorrectly classify events from this class will decrease.

In this research, cost-sensitive learning is applied by giving weights to both classes, background and signal. As stated before, background is the minority class, therefore, different weights greater than the signal weight are applied to the background to improve the performance on this group. Proposed solution with this method is stated in Section 3.8.2, training and validation in Section 4.4.2 and evaluation in Section 5.1.1.

**Hybrid methods**

Data-level and algorithm-level methods can be combined to improve performance on class imbalance learning. One way is to perform data sampling to reduce class noise and imbalance, and then apply cost-sensitive learning or thresholding to reduce the bias to the majority class.

# Chapter 3
# Proposed Solution

In the present chapter, all the work done in this project will be reviewed in deep, from the initial steps of reading data to the deploy of final models. The overall look to the proposed solution and workflow provided by this body of work is shown in Figure 15. The workflow to approach this problem can be divided into 5 big categories: ROOT files management, dataset analysis, dataset preprocessing, ANN model building, training and evaluation.

First steps of this project are related to ROOT files management, where data that comes from the ATLAS experiment is handled to turn it into a format that's easier to analyze in Python. Once the data is ready to be used in Python, studies regarding its features and characteristics are done, and signal distribution is calculated in each dataset. In deep discussion about this can be found in Section 3.1. Last data processing step is done just before being used as input for ANN model training, and it is detailed in Section 3.5.

Once the data was prepared, ANN models were started to build. Layers, neurons, activation functions, kernel initializers, dropout ratios, among other parameters had to be chosen and tuned, in a continuous cycle of training and evaluating. In deep model building and hyperparameters tuning process is shown in Section 3.6.

Finally, final versions of every model built are shown in detail. DNNs are shown in Section 3.7.1, PNNs are shown in Section 3.7.2, autoencoders are shown in Section 3.7.3 and models with applied imbalance learning are shown in Section 3.8.

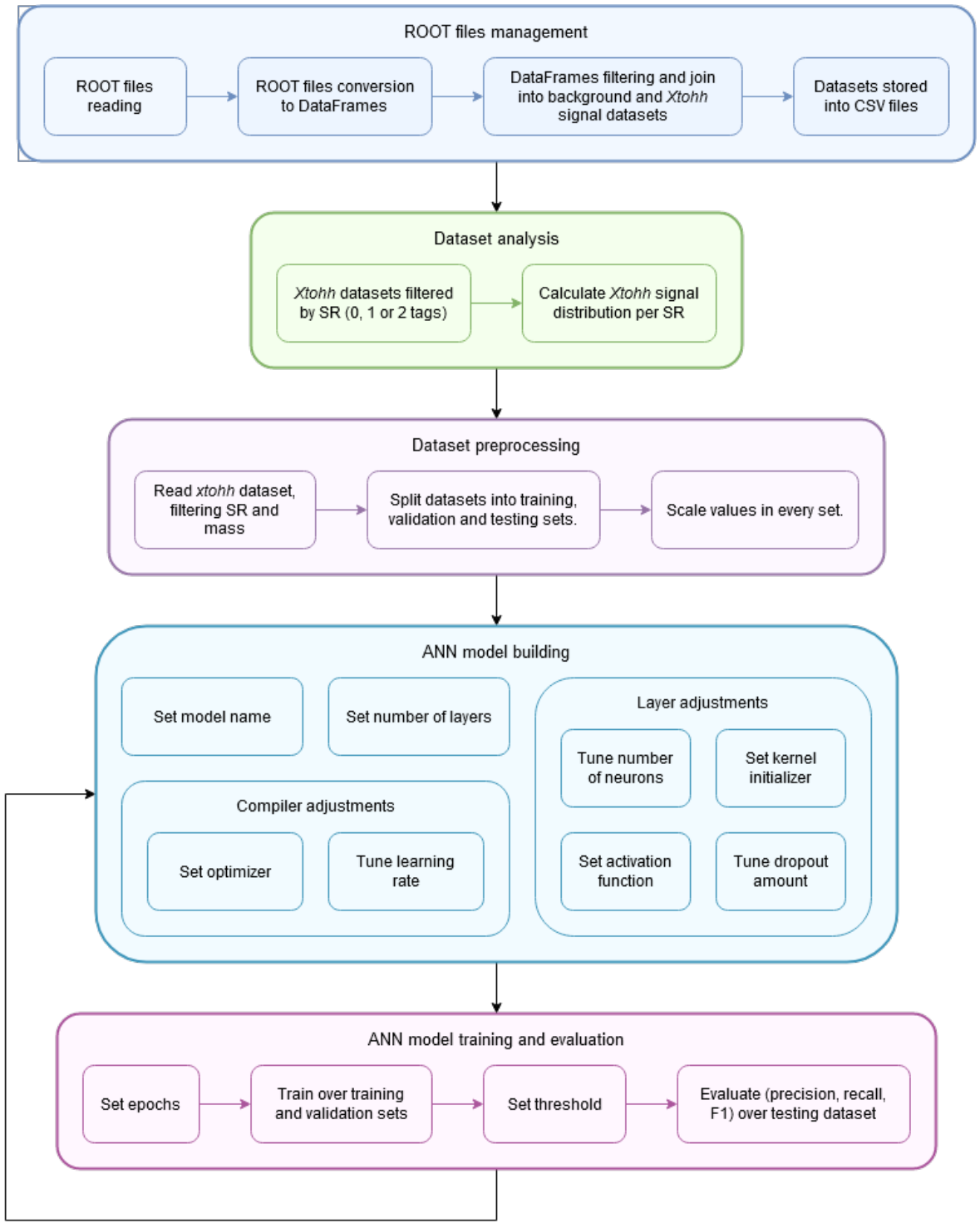



Figure 15: Overall project workflow and proposed solution.

## 3.1 Data and ROOT files management

This problem involves classifying a rare phenomenon, the generation of a di-Higgs, two boosted Higgs bosons, decaying into $b\bar{b}\tau^+\tau^-$ particles. Because of this, real data is not available for training classifiers, instead, simulated data has been used in this project. A simulator artificially generates events following the standard model and a model of the detector, taking into account noise and possible artifacts.

Simulated data is obtained by the di-Higgs analysis group, stored in the LHC distributed system, which has centers and servers all around the world [CERN, 2020a]. Three software programs in a row are used to produced this data: *MadGraph5* [Alwall et al., 2011], which generates the simulation of the main event with proton-proton collisions; *Pythia8* [Sjöstrand et al., 2015], which simulates particle fragmentation, hadronization and decay that occurs after the collision between two protons; and *Athena*, the ATLAS software, which simulates the final particle interactions with the ATLAS detector and the reconstruction and calibration of the objects used during the analysis. Meanwhile, real data is the measurements taken by the ATLAS experiment.

Data that comes from the ATLAS experiment is stored in `.root` files. These roots files contain TTree objects[4], which have an identifier called *Nominal* and branches that stores variables and its values. This representation can also be seen as a table, where each branch represents a column from it. Due to this, the conversion from TTree to DataFrame is pretty straight forward using PyROOT. These DataFrames are filtered to have only relevant information, and then joined to generate signal and background datasets that are saved into CSV files in a final step

These are some concepts and abbreviations used in the branches that help understanding their meaning:

- **FJ**: *fat jets*. A *fat jet* corresponds to a geometrically big beam that includes a decay of a heavy particle.
- **btag**: *btagged jets*. Identified as beams coming from bottom quarks.
- **pt**: *transverse momentum*. Component of the momentum perpendicular to the beam axis.
- **bbtt**: $b\bar{b}\tau\bar{\tau}$.
- **hh**: pair of Higgs bosons.
- **DT**: pair of $\tau$ particles (*di-*$\tau$)
- **eta**: pseudorapidity $\eta$ is a function of the particle's angle to the beam's line or axis.
- **phi**: azimuth $\phi$ is the angle measured from the axis x, around the beam. The ATLAS detector is symmetric in $\phi$

[4] TTree is a ROOT class that represents a columnar dataset where any C++ type can be stored in its columns. https://root.cern.ch/doc/master/classTTree.html

- **dPhi**: azimuth difference ($\Delta\phi$).
- **dR**: distance in $\eta$ $\phi$ ($\sqrt{\Delta\phi + \Delta\eta}$).

The branches on the tree are:

1. **sample**

   Tag of the event or type of process produced. This can take the following categories:

   - `Xtohh<mass>`: Production beyond the Standard Model Higgs of <mass>[GeV] decaying into two Higgs bosons. The sample masses are 1000, 1200, 1400, 1600, 1800, 2000, 2500 and 3000.
   - `W`: Production of a $W$ boson.
   - `WWPw`: Production of two $W$ bosons.
   - `WZPw`: Production of a $W$ boson and a $Z$ boson.
   - `ZZPw`: Production of two $Z$ bosons.
   - `ZeeSh221`: Production of a $Z$ boson decaying to a pair of electrons $e^+e^-$.
   - `ZtautauSh221`: Production of a $Z$ boson decaying to a pair of tau leptons $\tau^+\tau^-$.
   - `data`: Real data.
   - `fakes`: Events that produce the same final state as the di-Higgs, but don't correspond to it.
   - `stopWt`: Production of a *top quark* on the $Wt$ channel, where appears a $W$ boson and a *top quark*.
   - `stops`: Production of a single top quark in the $s$ channel.
   - `stopt`: Production of a single top quark in the $t$ channel.
   - `ttbar`: Production of a pair of *top quarks*.

2. **EventWeight**

   Weight of the event. It is calculated by the generators of simulated data. Its sum corresponds to the number of Events, and in consequence, these can be seen as the probability of an event of being generated. The non-simulated (real) events have weights of 1, meanwhile simulated events generally have weights between 0 an 1, but they can be negative or even values greater than one too.

3. **EventNumber**

   ID of the event.

4. **m_region**

   Region that determines the range of values that the event's variables can have. SR is one of these regions.

5. **m_FJNbtagJets**

   Amount of *fat jets* with *btags* (or *btagged fat jets*).

6. **m_FJpt**
   Transverse momentum of the *fat jets*.
7. **m_FJeta**
   Pseudorapidity $\eta$ of the **fat jets**.
8. **m_FJphi**
   Azimut $\phi$ of the **fat jets**.
9. **m_FJm**
   Mass of the *fat jets*.
10. **m_DTpt**
    Transverse momentum of the *di-*$\tau$.
11. **m_DTeta**
    Pseudorapidity $\eta$ of the *di-*$\tau$.
12. **m_DTphi**
    Azimuth $\phi$ of the *di-*$\tau$.
13. **m_DTm**
    Mass of the *di-*$\tau$.
14. **m_dPhiFTwDT**
    Azimuth difference ($\Delta\phi$) between the *fat jet* and the *di-*$\tau$.
15. **m_dRFJwDT**
    Distance between the centers of the *fat jet* and the *di-*$\tau$.
16. **m_dPhiDTwMET**
    Azimuth difference ($\Delta\Phi$) between *di-*$\tau$ an the missing pseudorapidity.
17. **m_MET**
    Missing transverse energy, i.e. energy that wasn't detected.
18. **m_hhm**
    Mass of the pair of Higgs bosons.
19. **m_bbttpt**
    Transverse momentum of $b\bar{b}\tau\bar{\tau}$.

The `sample` variable is the one used to define the signal and the background. Every `Xtohh sample` is consider signal and all the remaining ones are consider background. Since a binary classifier was developed, different backgrounds are not classified.

## 3.2 Dataset analysis

As indicated before, the occurrence of a di-Higgs is a rare event, therefore, the classes, signal and background, are very imbalanced, but in simulated data, the imbalance depends on the region (`m_region`). The region of interest for this work is the so called *signal region*, and this one is divided by number of tags (`m_FJNbtagJet`), which can be 0, 1 or 2: Signal Region 0 (SR0) has a small amount of signal (very imbalanced); Signal Region 1 (SR1) has a little bit more of signal; and Signal Region 2 (SR2) has a lot of signal and pretty low background (very imbalanced).

All events are weighted and its value indicates their probability of occurrence. This weights are needed to balance the amount of data each simulated sample is equivalent to. Weighted and non-weighted distribution is shown below in Table 1, Table 2 and Table 3. Signal distribution $d$ and weighted signal distribution $d_w$ are calculated with Equations 11, where $w_i \in \mathbb{R}$ is the weight of event $i$, $y_i \in \{0, 1\}$ is the class of the event $i$ and $N$ is the total number of events.

$$d_w = \frac{\sum_i^N w_i y_i}{\sum_i^N w_i} \qquad d = \frac{\sum_i^N y_i}{N} \tag{11}$$

| Xtohh | $d_w$ | $d$ | Signal events | Total events |
|---|---|---|---|---|
| 1000 | 0.002 | 0.017 | 902 | 52528 |
| 1200 | 0.007 | 0.034 | 1823 | 53449 |
| 1400 | 0.012 | 0.053 | 2892 | 54518 |
| 1600 | 0.016 | 0.069 | 3825 | 55451 |
| 1800 | 0.019 | 0.080 | 4505 | 56131 |
| 2000 | 0.020 | 0.083 | 4670 | 56296 |
| 2500 | 0.020 | 0.082 | 4641 | 56267 |
| 3000 | 0.015 | 0.059 | 3262 | 54888 |

Table 1: SR0 signal weighted and non weighted distribution, signal events number and total events number, per Xtohh mass.

| Xtohh | $d_w$ | $d$ | Signal events | Total events |
|---|---|---|---|---|
| 1000 | 0.037 | 0.342 | 2154 | 6293 |
| 1200 | 0.155 | 0.548 | 5014 | 9153 |
| 1400 | 0.235 | 0.663 | 8138 | 12277 |
| 1600 | 0.283 | 0.712 | 10219 | 14358 |
| 1800 | 0.309 | 0.735 | 11472 | 15611 |
| 2000 | 0.321 | 0.743 | 11948 | 16087 |
| 2500 | 0.296 | 0.716 | 10445 | 14584 |
| 3000 | 0.220 | 0.623 | 6842 | 10981 |

Table 2: SR1 signal weighted and non weighted distribution, signal events number and total events number, per Xtohh mass.

| Xtohh | $d_w$ | $d$ | Signal events | Total events |
|---|---|---|---|---|
| 1000 | 0.287 | 0.870 | 1647 | 1893 |
| 1200 | 0.687 | 0.948 | 4475 | 4721 |
| 1400 | 0.773 | 0.966 | 6897 | 7143 |
| 1600 | 0.802 | 0.971 | 8122 | 8368 |
| 1800 | 0.811 | 0.972 | 8555 | 8801 |
| 2000 | 0.811 | 0.972 | 8560 | 8806 |
| 2500 | 0.773 | 0.964 | 6683 | 6929 |
| 3000 | 0.669 | 0.941 | 3936 | 4182 |

Table 3: SR2 signal weighted and non weighted distribution, signal events number and total events number, per Xtohh mass.

## 3.3 ADA library

Every function and class created for the present work was stored in a custom library named ATLAS Data Analysis (ADA). This library is divided in different sub-libraries by functionality, to modularize the code.

- **data**: data processing an root/dataframe handlers.
- **info**: statistical information from dataframes.
- **model**: machine learning models

- **plot**: plotting of train-val losses, confusion matrix, evaluations and others.

Main methods and classes that ADA provides are illustrated in Figure 16.

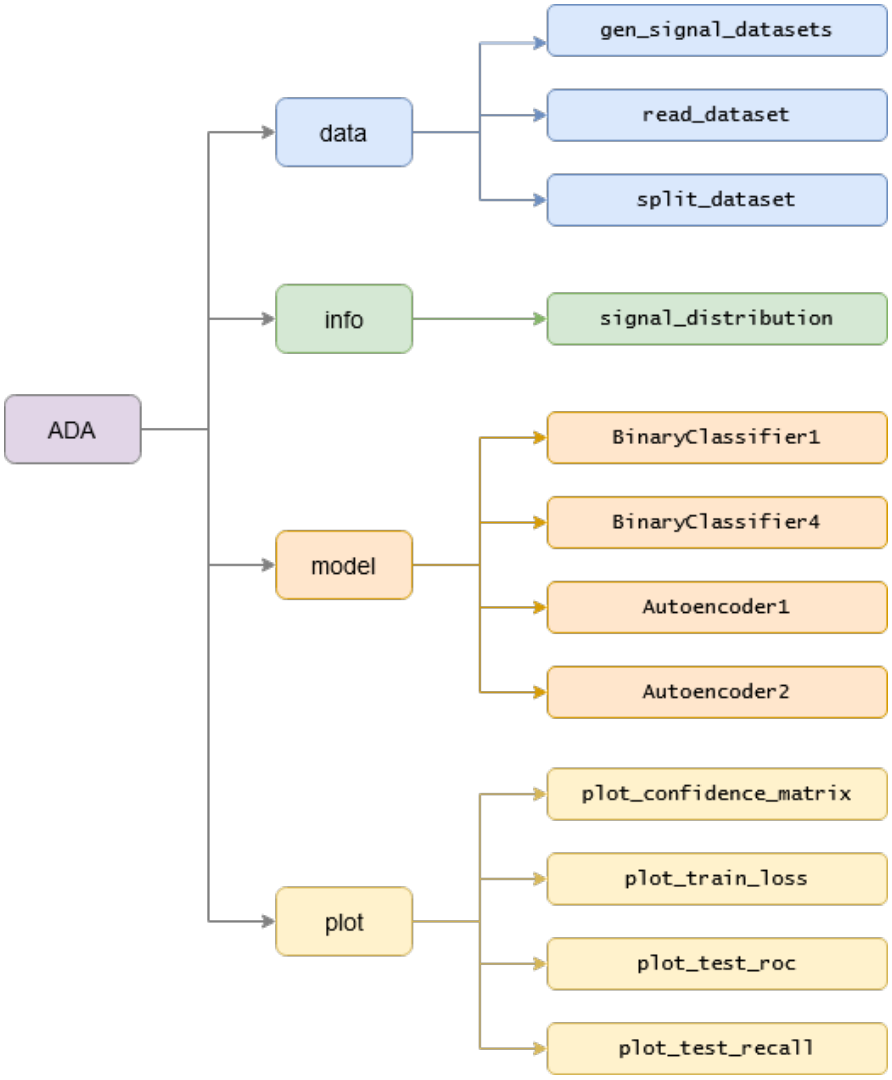



Figure 16: ADA library illustration of its main methods and classes.

ADA library can be found in [Rodriguez, 2020a], where all the tools for data analysis and deep learning models can be found.

## 3.4 Feature Selection

Every event is characterized by 19 properties, indicated above in Section 3.1, from which, `sample`, `EventWeight`, `EventNumber`, `m_region` and `m_FJNbtagJets` are used to filter the datasets, therefore, they are not used for training, leaving 14 features to work with. To mea-

sure relevancy of this properties, and then, select the ones that will be part of the machine learning process, multiple methods to rank features were used. From these, the five best features were selected to build features sets. All methods tested and its features sets are listed below.

- Chi-square ($\chi^2$) test: `{m_FJpt, m_hhm, m_DTpt, m_MET, m_bbttpt}`
- Extra trees: `{ m_hhm, m_FJm, m_DTm, m_FJpt, m_MET }`
- Random forest: `{ m_hhm, m_FJm, m_DTm, m_dRFJwDT, m_MET }`
- All features: `{ m_FJpt, m_FJeta, m_FJphi, m_FJm, m_DTpt, m_DTeta, m_DTphi, m_DTm, m_dPhiFTwDT, m_dRFJwDT, m_dPhiDTwMET, m_MET, m_hhm, m_bbttpt }`

These feature sets were tested with *Mass parameterized Neural Network with class weights* defined in Section 3.8.3, using different combinations of class weights and thresholds, resulting in 352 different models. Comparing the F1 from every model, both in signal and background, all the top 10 models are trained with all the features, no model trained with a feature selected set was presented in the first positions. Because of this, and the fact that training with the 14 features takes almost the same time then training with a selected feature set, from here on all the features are used to build models.

## 3.5 Data preprocessing

First, every root file is converted into CSV files, by separating the signals in different `csv`'s and leaving the background in its own. After this step the following files are created:

- `Xtohh_background.csv`
- `Xtohh1000.csv`
- `Xtohh1200.csv`
- `Xtohh1400.csv`
- `Xtohh1600.csv`
- `Xtohh1800.csv`
- `Xtohh2000.csv`
- `Xtohh2500.csv`
- `Xtohh3000.csv`

Then these files are read using `ada.data.read_data()`. This function reads the csv's into a pandas dataframe and applies the processing needed before splitting the data into train-val-test datasets. First, a column named `label` is added to the dataframe, where the signal takes the value 1 and the background the value 0. Then the dataframe is filtered, only keeping the events from the SR region with tag 2 (or any other region and tag chosen). In the final step, the columns selected as features and the *EventWeights* column are filtered from the dataframe and returned.

The last preprocessing step is performed by `ada.data.split_data()`. This method pops the `label` column from the dataframe to turn it into the $y$ vector of true classes, pops the `EventsWeight` column to turn it into the $w$ vector of sample weights and leave all the remaining features into the $X$ dataframe of events. Then, it splits these datasets into train, val and test sets, with ratio indicated into the function parameters. This ratio depends on the approach that is taken. For neural networks, generally 0.6:0.2:0.2 or 0.5:0.3:0.2 ratio (train:val:test) are used. Before returning this sets, $X$ needs to be scaled, i.e. all its values need to be within a defined range. *Standard scaler* is a very popular scaler that comes with the *scikit learn*[5] library, which scales values ($x$) between 0 and 1 by subtracting the mean of all the data ($u$) and dividing by the standard deviation ($s$) (Equation 12). This one is used to scale training, validation and testing datasets (all three using the same scaler). Once this is done, the sets are returned.

$$z = \frac{x - u}{s} \tag{12}$$

## 3.6 Model building and hyperparameter tuning

First ANN properties that were thought and fixed were layers and neurons per layer. The best architecture so far for a DNN is defined in Section 3.7.1, and the best architecture for an autoencoder is defined in Section 3.7.3. These model architectures were fixed and used through the whole project.

The rest of the hyper-parameters from the model itself, activation functions, learning rate and optimizer, were tuned alongside training, validation and testing dataset size, and threshold. Each DNN and autoencoder was tuned separately, t(herefore these parameters differ from model to model. Tuning is made through the evaluation of different combinations of these hyperparameters, where each hyperparameter is given a fixed list of possible values, and a loop traverses all the values on each parameter list. To evaluate which combination is the best, F-score was calculated on both signal and background classes. Hyperparameter tuning of DNNs is further explained in Section 3.7.1.

## 3.7 Deep learning approaches

Different deep learning approaches have been tested in this project, which are defined in this Section. Architecture and models are shown in each approach, showing how they were built and which one is the best.

[5] Scikit learn is a python library for predictive data analysis. https://scikit-learn.org/stable/index.html

### 3.7.1 Deep neural networks

Different models and architectures have been built through this work, being the first approach to build a feed forward DNN. After some tweaking of layers and hyperparameters a final architecture was chosen, detailed below in Figure 17.

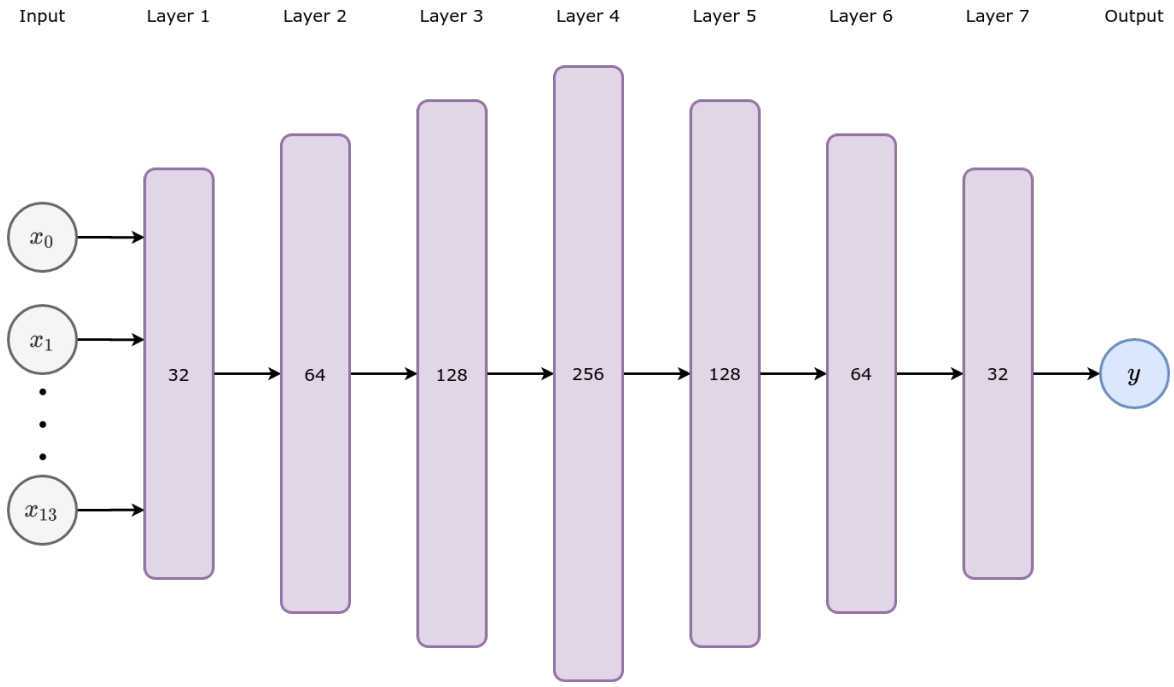



Figure 17: Feed forward DNN architecture.

This architecture has an input of 14 features, 7 hidden layers with 32, 64, 128, 256, 128, 64 and 32 neurons respectively, and one output neuron. Also, between all the layers there is a dropout rate of 0.2. The input layer has a *uniform* kernel initializer[6], meanwhile the hidden and output layers have a *he-uniform* kernel initializer. Every hidden layer has an activation depending on the model version, but the output layer always have a *sigmoid* activation. This ensures that the output is a value between 0 and 1. In consequence, a *threshold* is needed to determine which events are signal, which also depends on the model used.

*Learning rate* and *optimizer*, both hyperparameter related to the training step, are also specific from every model. On the other hand, the loss function is fixed as *binary crossentropy*.

Binary Classifier 1 (BC1) is the first model that have got good results training with the `Xtohh2000` dataset. As it was mentioned before, the hidden layers activation depends on the model, and in this specific model every hidden layer has a **softplus activation** (Equation 12). In the training step, this model uses a **learning rate of 0.05** and the **adagrad optimizer**. After trying different threshold values between 0 and 1, this model showed better performance with a threshold of 0.4 : events with outputs greater then 0.4 are signal.

[6]Initializers define the way to set the initial random weights of layers.

Feed-Forward Neural Network Architecture (FFNNA) is the model used to find an optimal combination of hyperparameters for this architecture. Learning rate, optimizers, dataset split ratio for training, validation and testing, and threshold are the hyperparameters tuned. Specific values of these parameters are listed below.

- Learning rates: 0.005, 0.01, 0.05, 0.1
- Optimizers: adam, adadelta, adagrad, adamax
- Activations: relu, softplus
- Dataset (train, val, test) split ratios: (0.6, 0.2, 0.2), (0.5, 0.3, 0.2), (0.5, 0.2, 0.3), (0.4, 0.3, 0.3), (0.4, 0.2, 0.4), (0.3, 0.3, 0.4)
- Thresholds: 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

Binary Classifier 2 (BC2) and Binary Classifier 3 (BC3) are early models for tuning hyperparameters. BC2 tuned every hyperparameter FFNNA did plus the dropout. BC3 tuned the same parameters as FFNNA, but they differ on implementation (BC3 is a sequential Keras[7] model and FFNNA is a functional keras model).

Binary Classifier 4 (BC4) is the model born by tuning hyperparameters with FFNNA on the `Xtohh` dataset, parameterizing the mass. It has **relu activation** (Equation 11), **adamax optimizer** and **0.05 learning rate**.

### 3.7.2 Parameterized neural networks

Building models for each mass of the Xtohh signal can become tricky and it's not optimal. In order to generalize and try to get better results, **parameterized** versions of the previously created DNNs were trained [Baldi et al., 2016], by incorporating the mass as a model input. The new resulting architecture is shown in Figure 18. By parameterizing the mass, only one model is trained for all the masses of the Xtohh.

[7] https://keras.io/

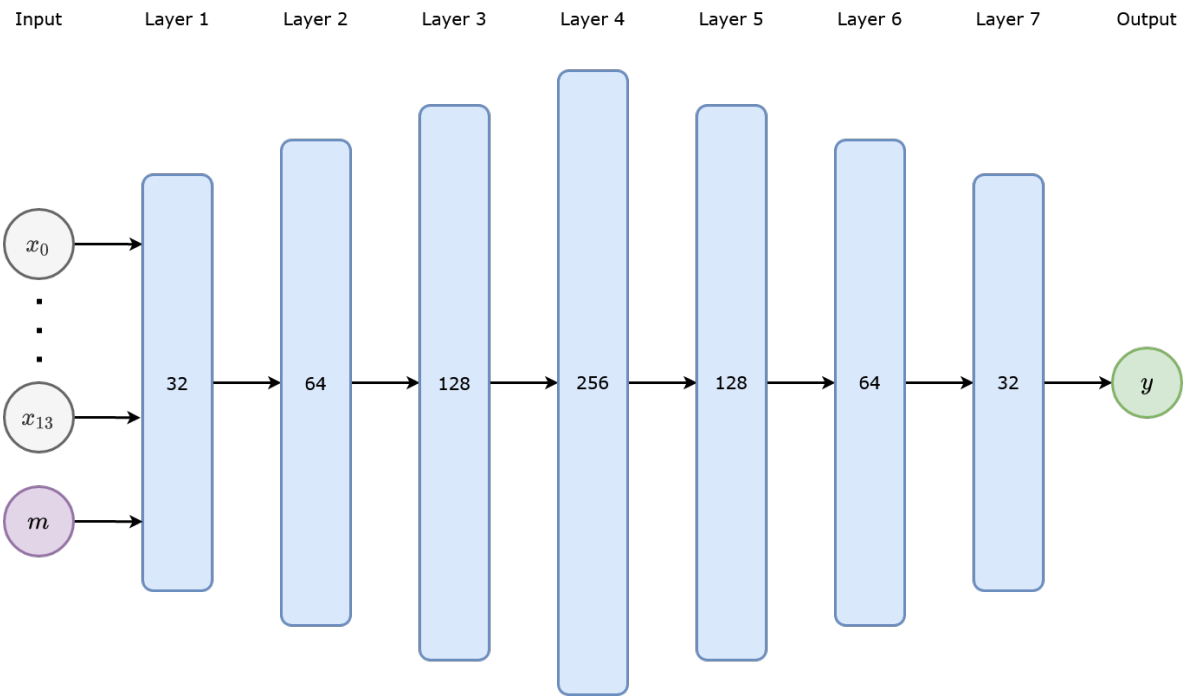



Figure 18: Mass parameterized feedforward DNN architecture.

BC1 and BC4 were trained with parameterized data, leaving the mass as one more feature to learn to.

### 3.7.3 Autoencoders

After successfully building feed forward neural networks, *autoencoders* has been started to build. Different architectures were tested and the following gives the best results, drawn in Figure 19. It consists of an input layer of 14 features, followed by an encoder layer of 8 neurons, a latent layer with 2 neurons, a decoder layer with 8 neurons and an output layer with the 14 reconstructed features.

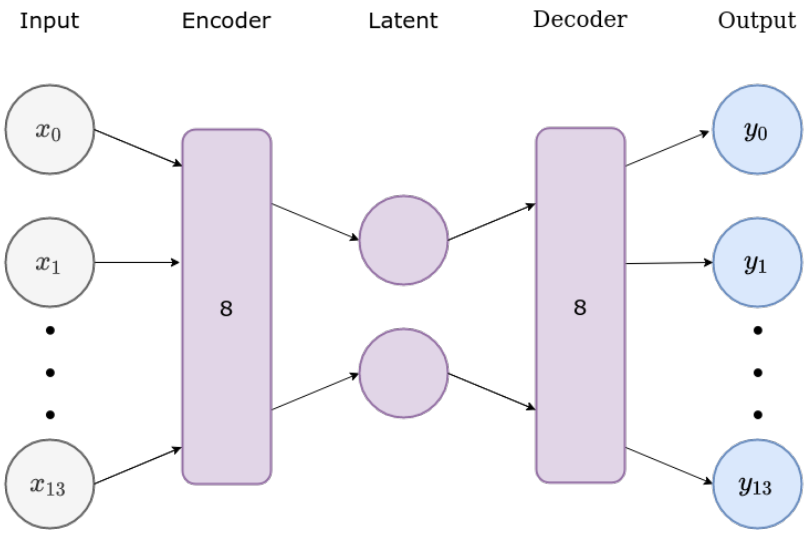



Figure 19: Autoencoder architecture.

Autoencoder 1 (A1) is the first autoencoder model developed with this architecture. Decoder, latent and encoder layer have *he_uniform* kernel initializer and *relu* activation, meanwhile the output layer has a *sigmoid* activation. Between every layer there is a dropout rate of 0.2. The model is compiled with a *mean squared error* loss and an *adam* optimizer with *learning rate* of 0.1.

To improve performance, a tuning of hyperparameters was performed with this architecture. Learning rate, optimizers and dataset split ratio for training, validation and testing were the hyperparameters tuned, taking the values listed below.

- Learning rates: 0.005, 0.01, 0.05, 0.1
- Optimizers: adam, adadelta, adagrad, adamax.
- Dataset split ratios (train, val, test): (0.6, 0.2, 0.2), (0.5, 0.3, 0.2), (0.5, 0.2, 0.3), (0.4, 0.3, 0.3), (0.4, 0.2, 0.4), (0.3, 0.3, 0.4).

Autoencoder 2 (A2) is the second and last autoencoder model built. It's similar to A1 in every aspect except for the optimizer: here **adamax** is used with a learning rate of 0.1.

Between all models with a deep learning approach, BC4 presented the best performance, improving results of its little brother BC1. Between all deep learning approaches, PNN was the best one, improving even more the performance of BC4. All the evaluations of this models are shown in Section 5.1.

## 3.8 Imbalanced learning approaches

To handle the imbalance issue in the SR2 dataset, which has a lot of signal but pretty low background (minority class), different techniques of imbalanced learning were used, which are detailed below.

### 3.8.1 Oversampling

Using the latest DNN model created, BC4, two techniques of oversampling were used: ADASYN and SMOTE. Both of them were trained and evaluated using different minority class ratios, from 0.1 to 0.5. The technique that showed better early results was ADASYN with 0.2 background ratio, so this one was used for further evaluation.

### 3.8.2 Class weights

In contrast of oversampling, which is a data-level method to deal with class imbalance issues, cost-sensitive learning handles this issue by modifying the inner algorithm. In this case, to apply cost-sensitive learning, weights are assigned to both signal and background classes. Every training process done before was made without class weights, i.e., both signal and background had weight 1 in the backpropagation. By changing this weights, more cost or *punishment* can be added when the model gets the results wrong for the minority class. In this case, more weight needs to be added to the background, since it's the minority class, so the model can improve its scores on this one.

It is worth mentioning that class weights are not the same as event weights. An event weight is the probability of occurrence of itself, and it's assigned to each sample to measure its individual relevancy. On the other hand, a class weight measures the importance of the entire class, punishing the classifier more when it mistakes the more weighted class.

For this approach, model BC4 was used, since it's the last model developed. To tune the weights per class, the weight for the signal class was fixed on 10, and the weight for the background class was varying between 10 and 20 (for values greater than 20 the model didn't show good results). The best ratio on early results was 10:11 (signal:background), being this one used for further testing.

### 3.8.3 Mass parameterized with class weights

After the improvement on performance seen with the PNN and the DNN with class weights, each one on their own, a model combining these two techniques was trained. BC4 is the base model for this approach, using the same values for hyper-parameters as the mass parameterized alone version. Following the same workflow used for tuning the deep neural network with class weights above, the background class weight is tuned taking values from

10 to 20, meanwhile the signal class weight is fixed on 10. The best ratio signal class weight to background class weight is 13:10, being this ones used for further research.

Between all the models using an imbalance learning approach, BC4 remains the best model, and the best imbalance learning approach was class weights with parameterized neural networks. Evaluation of this model and this approach can be found in Section 5.1.

# Chapter 4
# Training

In the beginning, the `Xtohh2000` dataset is used to train, validate and test the models, in SR. Although tag 2 is the priority of this work, work has been done in all three tags 0, 1 and 2. In latest steps of this work, `Xtohh` in all masses was used, giving priority to SR2.

## 4.1 Deep neural networks

BC1 is trained and tested on SR 0, 1 and 2, separately. Every dataset is partitioned into train, val and test sets, with ratios 0.6, 0.2 and 0.2 respectively. The training step is done using 50 epochs, using the $X$, $y$ and $w$ sets obtained in the preprocessing step. Down below plots of the training and validation loss can be found in Figures 20, 21, 22.

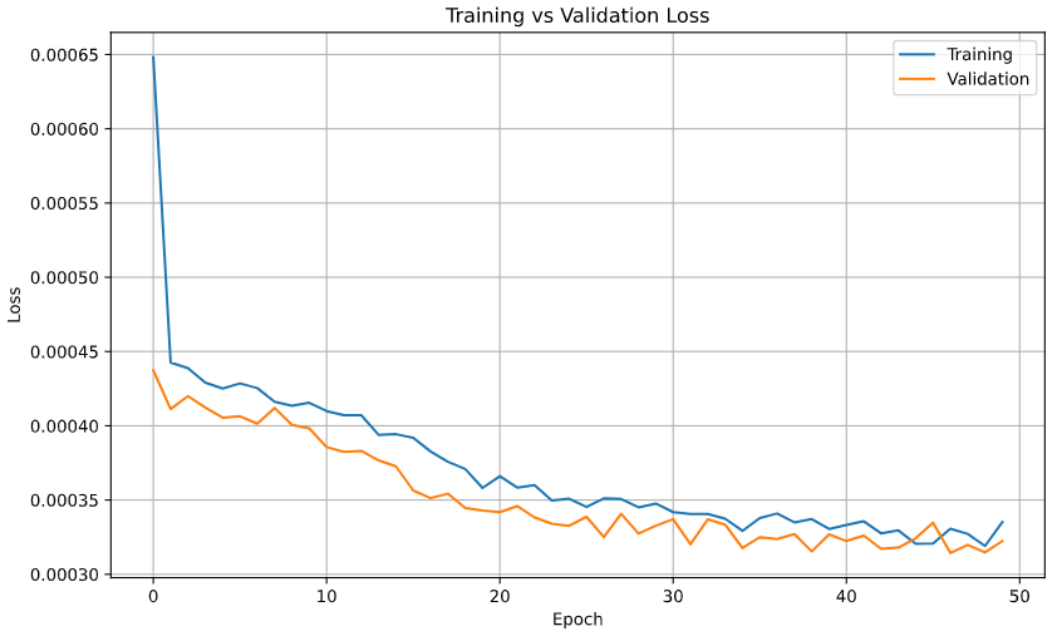



Figure 20: Train and validation loss of the BC1 model in SR0.

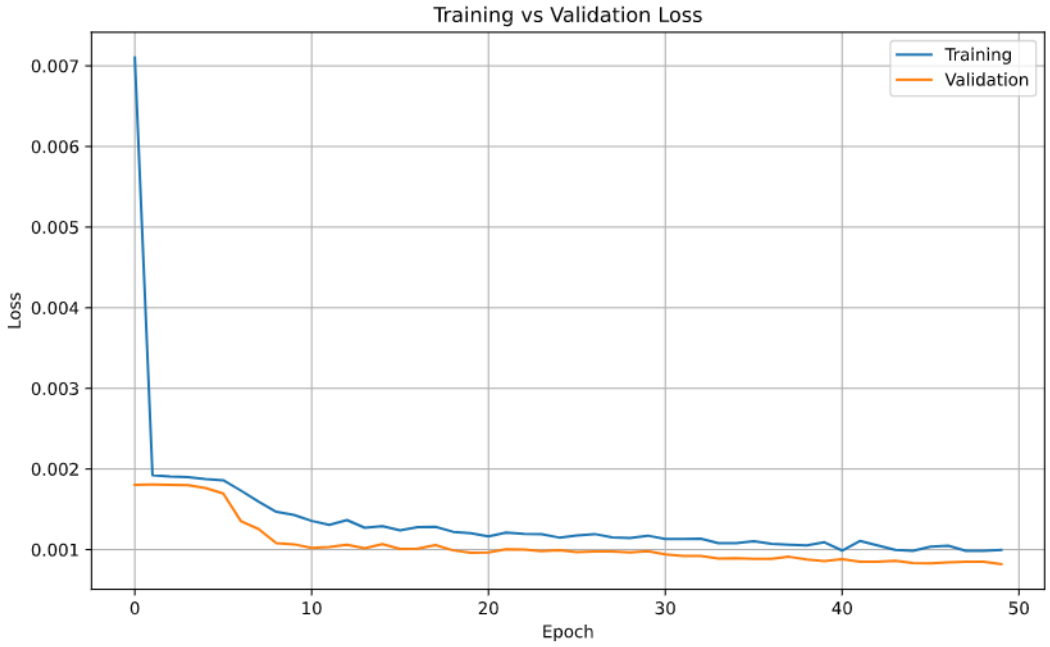



Figure 21: Train and validation loss of the BC1 model in SR1.

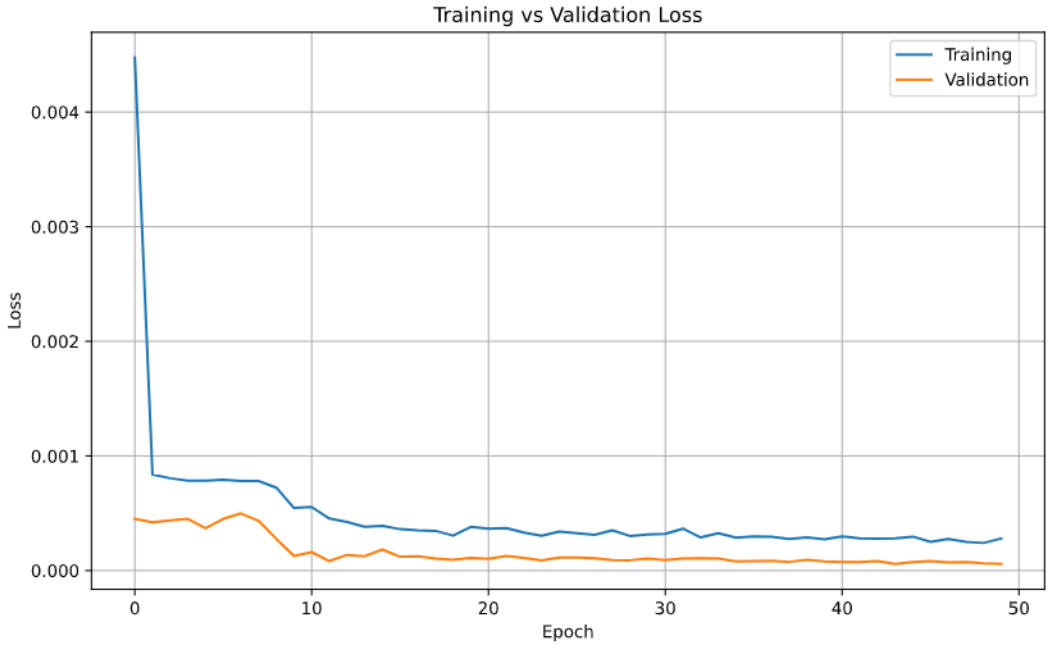



Figure 22: Train and validation loss of the BC1 model in SR2.

BC4 is trained and tested only in SR2. Every dataset is partitioned into train, val and test sets, with ratios 0.5, 0.3 and 0.2 respectively. The training step is done using 50 epochs, using the $X$, $y$ and $w$ sets obtained in the preprocessing step. Down below a plot of the training and

validation loss can be found in Figure 23.

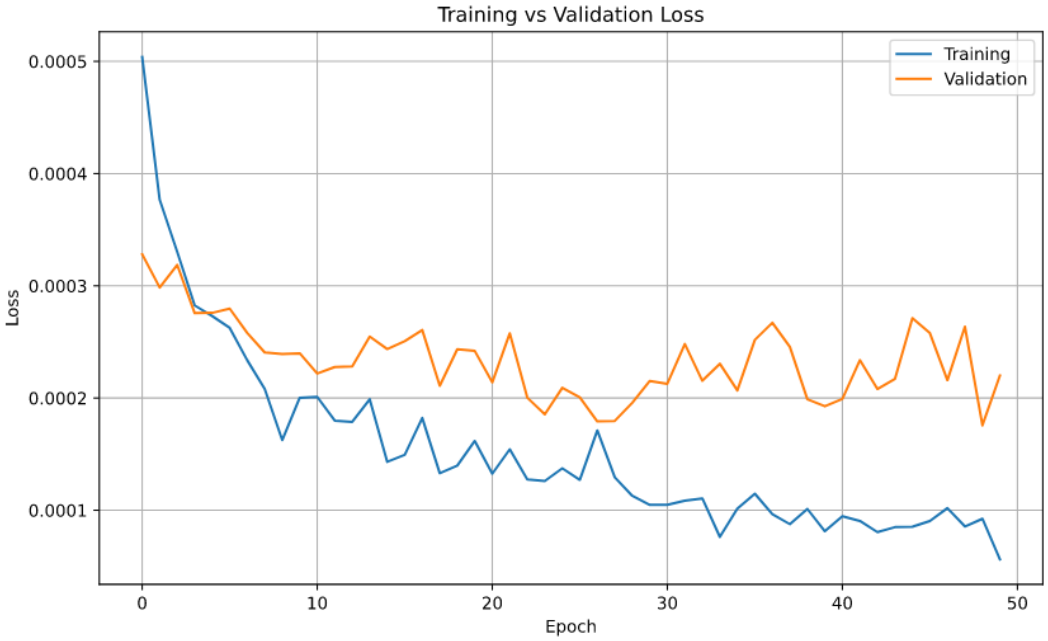



Figure 23: Train and validation loss of the BC4 model on SR2.

## 4.2 Parameterized neural networks

Both BC1 and BC4 are trained using a mass parameterized approach in SR2. All `Xtohh` events are joined in the same dataset, using a column to store their masses. This dataset is partitioned into train, val and test sets, with ratios 0.5, 0.3 and 0.2 respectively. The training step is done using 50 epochs, using the $X$, $y$ and $w$ matrices obtained in the preprocessing step. Down below training and validation loss plot can be found in Figure 24 for BC1 and Figure 25 for BC4.

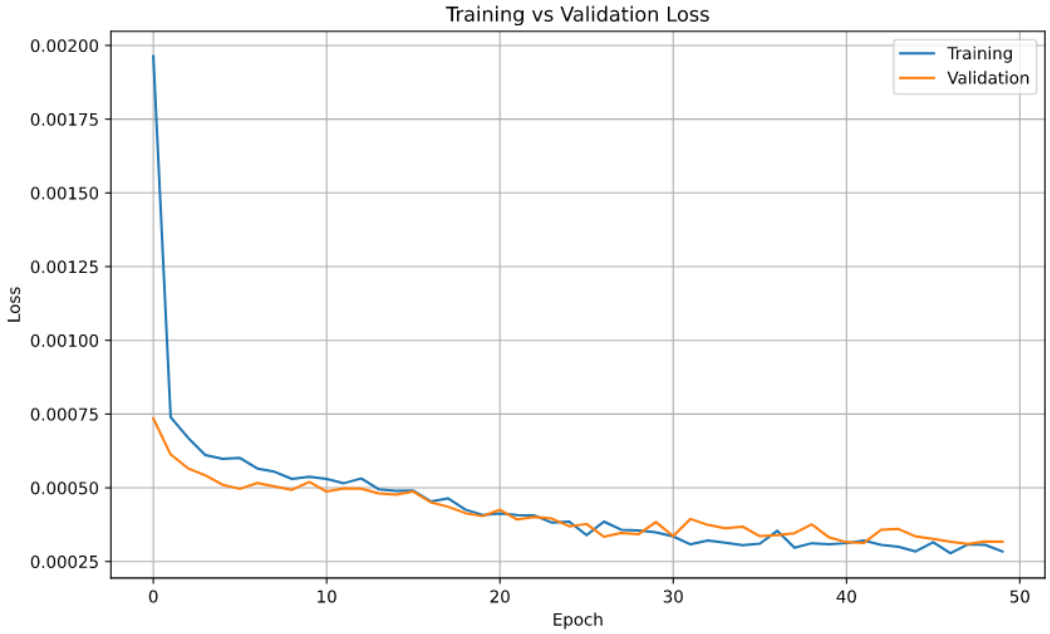



Figure 24: Train and validation loss of the Mass Parameterized BC1 (BC1-MP) in SR2.

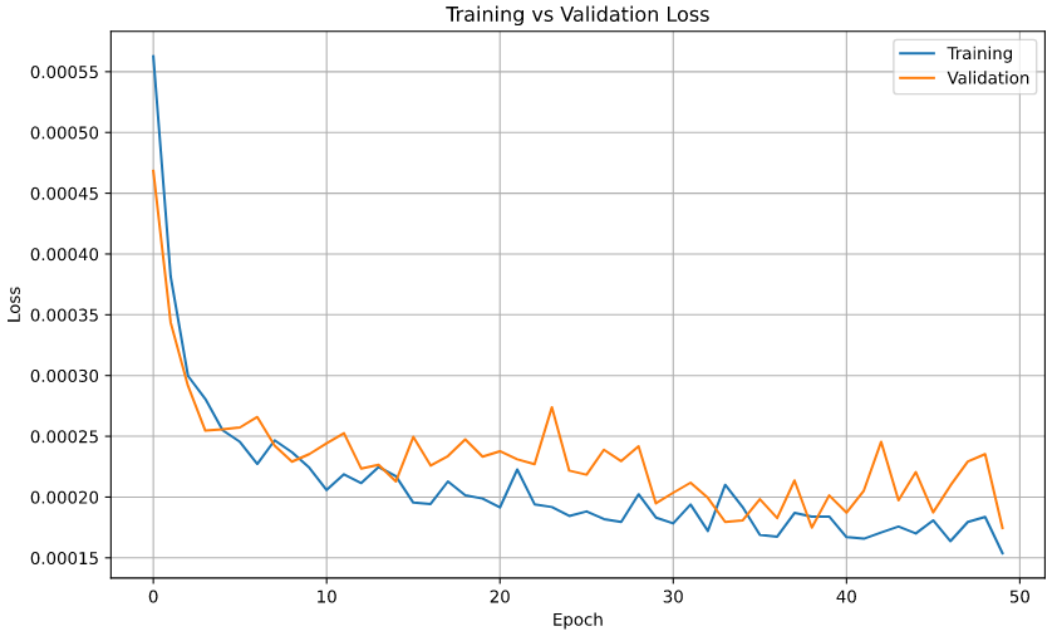



Figure 25: Train and validation loss of the Mass Parameterized BC4 (BC4-MP) in SR2.

## 4.3 Autoencoders

Autoencoders are generally used for *anomaly detection*, i.e. when the data is imbalanced and the vast majority of events are just from one class. In this case, in SR2, a big percent of the data is from class 1, the signal, leaving much less background. This is why an autoencoder is build to solve this problem specifically in SR2.

Before starting the training step, $X$ and $w$ sets are divided by class, background (0) and signal (1):

- `x_train` was divided into `x_train_0` and `x_train_1`.
- `x_val` was divided into `x_val_0` and `x_val_1`.
- `w_train` was divided into `w_train_0` and `w_train_1`.
- `w_val` was divided into `w_val_0` and `w_val_1`.

This division is done because the autoencoder needs to learn only the majority class, that in this case is the signal class. This way, the autoencoder is only fed with the class 1 dataset.

Both A1 and A2 Models are trained and validated with class 1 sets, so it can learn the structure of the signal. A plot of training and validation loss can be found down below in Figure 26 for A1 and Figure 27 for A2.

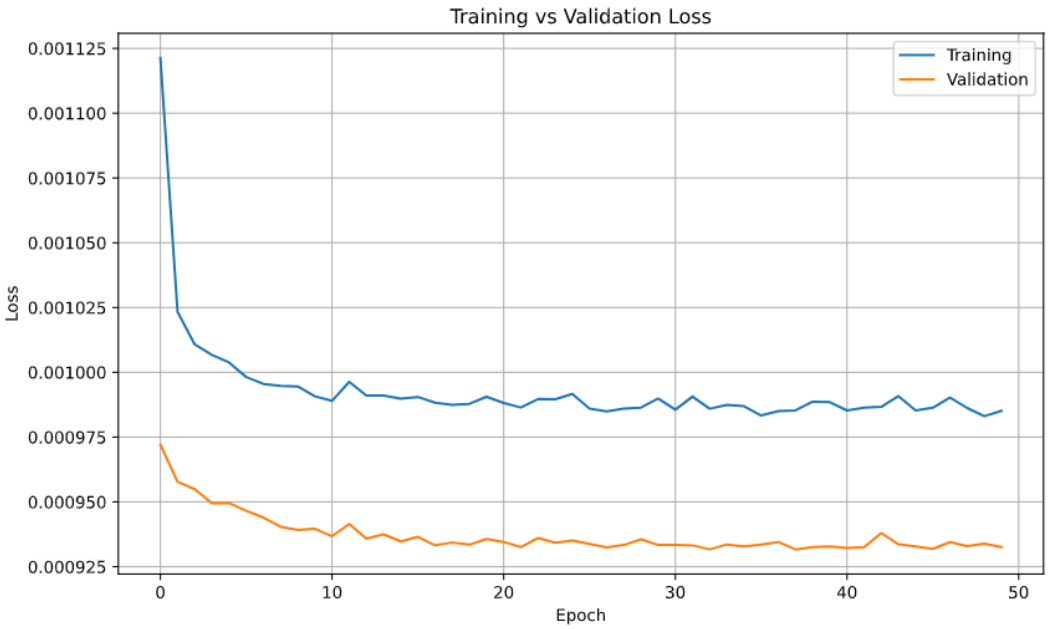



Figure 26: Train and validation loss of the A1 model in SR2.

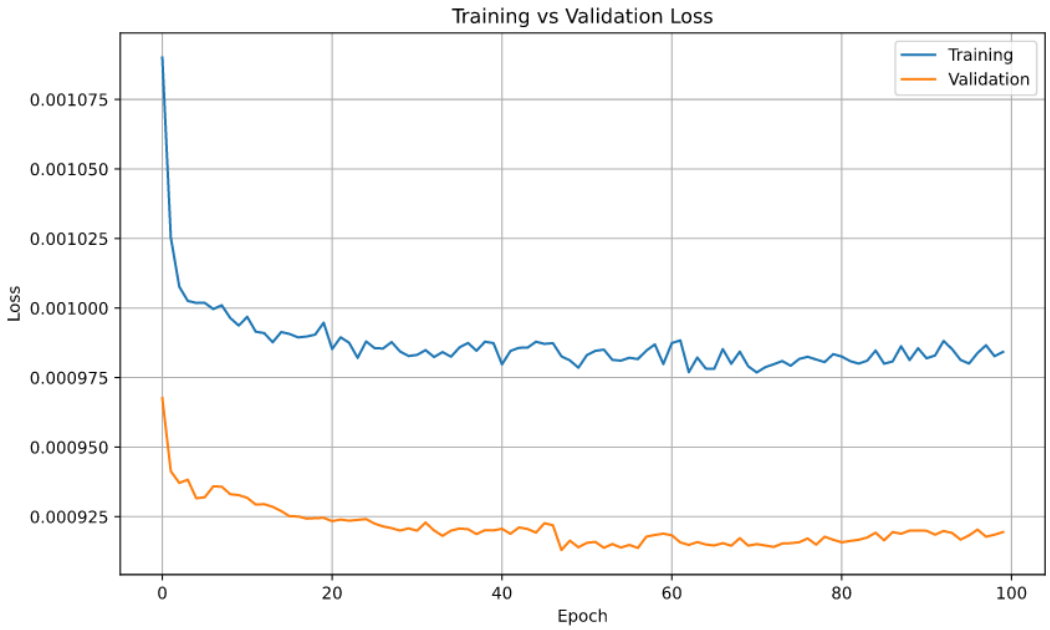



Figure 27: Train and validation loss of the A2 model in SR2.

After training, the models are tested with the whole test dataset (both classes) by generating the reconstruction error per event and plotting them, as shown in Figure 28 for A1 and Figure 29 for A2. With signal as orange dots and background as blue dots, a threshold can be estimated by looking at the plot. The threshold is fixed at **1.3** for A1 and **1.5** for A2.

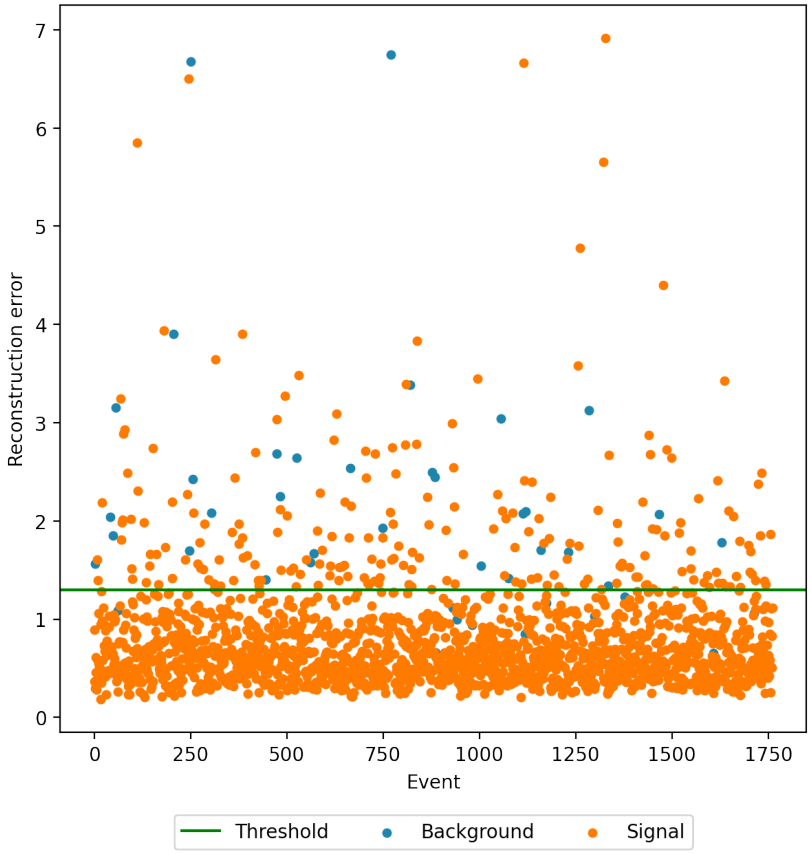



Figure 28: Reconstruction error from the A1 model on SR2.

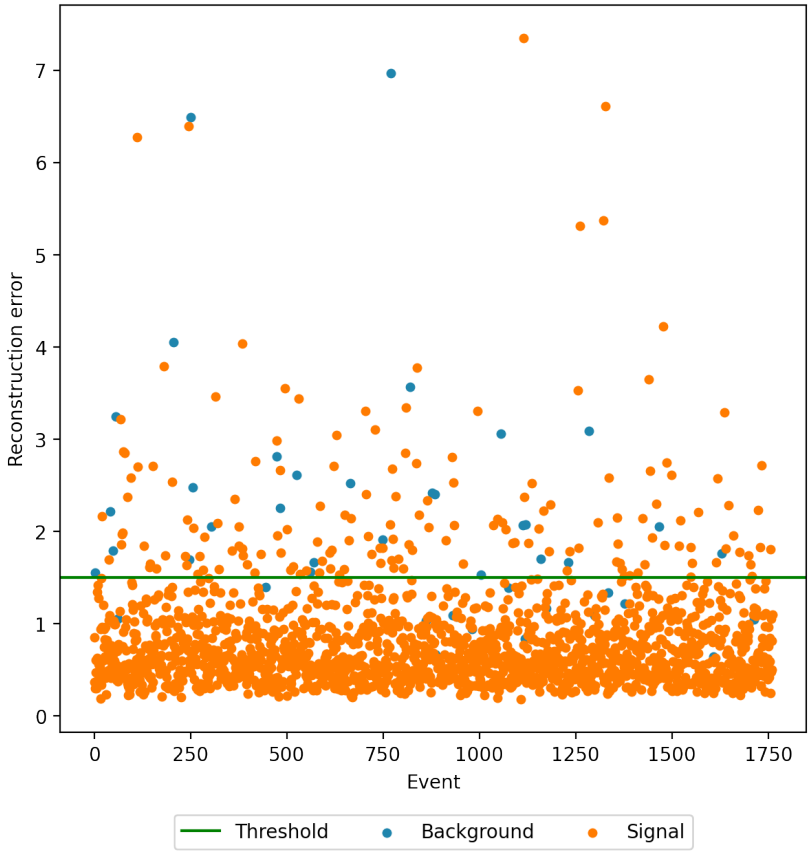



Figure 29: Reconstruction error from the A2 model in SR2.

The reconstruction error of an event indicates how well a model learns the event representation (encoding). Since the models are trained with signal datasets, it is expected that the reconstruction errors of the background are larger than then reconstruction errors of the signal. Graphically, this error can be plotted to estimate a threshold that separates best both classes, to use it later for prediction purposes. In Figure 28 and Figure 29 most of the signal events are grouped around the same error values, but there is a point where both signal and background starts to blend, which is not desirable. Having a lot of signal events with high reconstruction error makes harder to choose a threshold to separate both classes. This is why a second autoencoder (A2) was built, so the reconstruction error on signal decreases and the reconstruction error on background increases.

## 4.4 Class imbalanced learning models

### 4.4.1 Oversampling with ADASYN

To apply oversampling, first, `Xtohh2000` dataset is partitioned into training, validation and testing sets, with ratios 0.5, 0.3 and 0.2 respectively, and then ADASYN is applied to the

training set, increasing the amount of background events until it hits 20% of the dataset. Once the sets are ready, BC4 is trained with the oversampled training set, using 50 epochs. Training and validations loss are shown down below in Figure 30

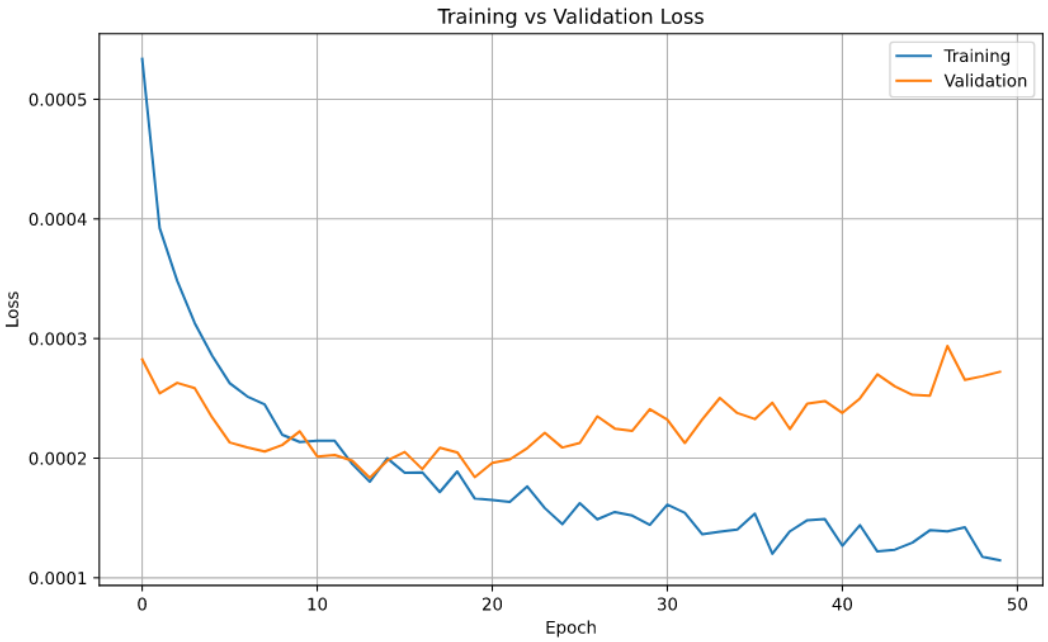



Figure 30: Train and validation loss of the BC4 with ADASYN (BC4-ADASYN) oversampling data with 0.2 ratio of minority class in SR2.

### 4.4.2 Class weights

Two approach were taken to train models with class weights: standard and mass parameterized. For the standard version, `Xtohh2000` was the signal to classify, meanwhile for the mass parameterized version all the **Xtohh** events were used, using a column to indicate their masses [8]. In both cases, the dataset was splitted into training, validation and testing sets, using ratios of 0.5, 0.3 and 0.2 respectively, using BC4 to train over the training dataset, using 50 epochs. Training and validation losses can be found on Figure 31 and Figure 32.

[8] *Xtohh* masses to work with: 1000, 1200, 1600, 1800, 2000, 2500, 3000

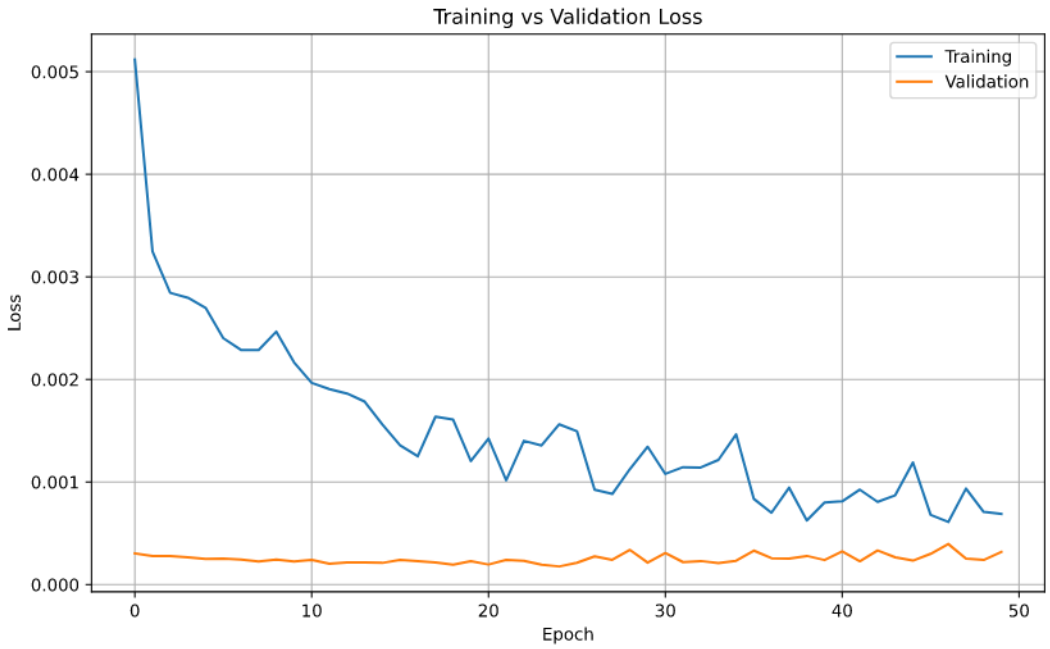



Figure 31: Train and validation loss of the BC4 with Class Weights (BC4-CW) with 11:10 background:signal ratio in SR2.

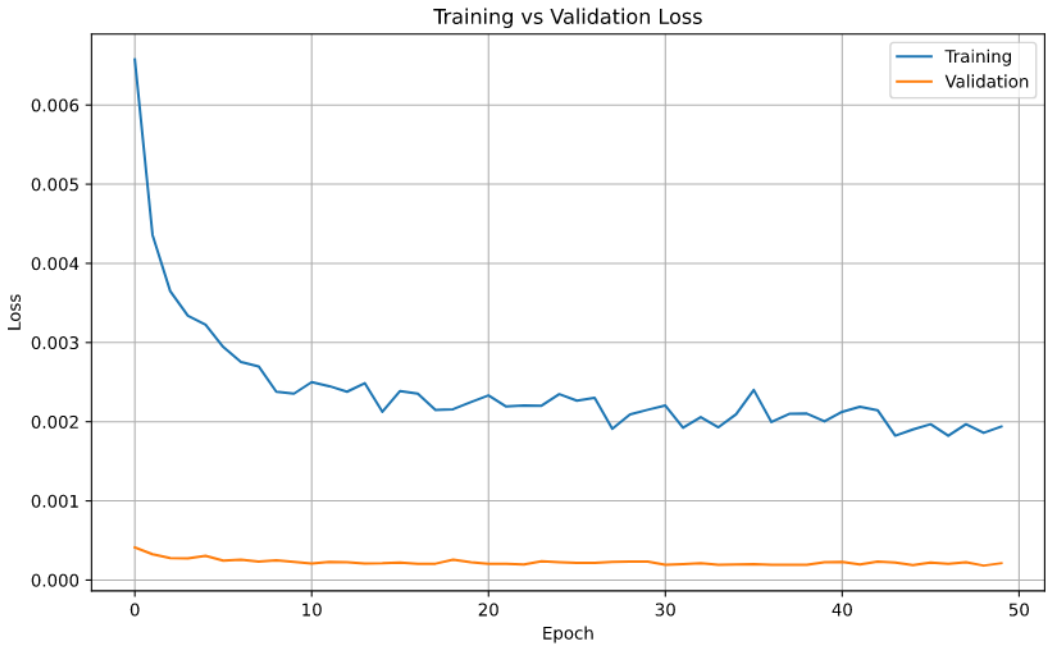



Figure 32: Train and validation loss of the Mass Parameterized BC4 with Class Weights (BC4-CW-MP) with 13:10 background:signal ratio in SR2.

# Chapter 5
# Evaluation

## 5.1 Model performance scores

All trained models were evaluated in their respective testing datasets, using their optimal thresholds chosen from tuning, to decide if an event is signal or background. Scores used to measure model performance are precision, recall and F1, which are compared through all the results per each `Xtohh` dataset.

### 5.1.1 Xtohh2000

Xtohh2000 is the dataset where more models were tested, since earlier models were trained with it. Model performance comparison plots can be found in Figure 33, with detailed results in Table 4. Weighted and non-weighted confusion matrices can be found in Figure 34 and Figure 35.

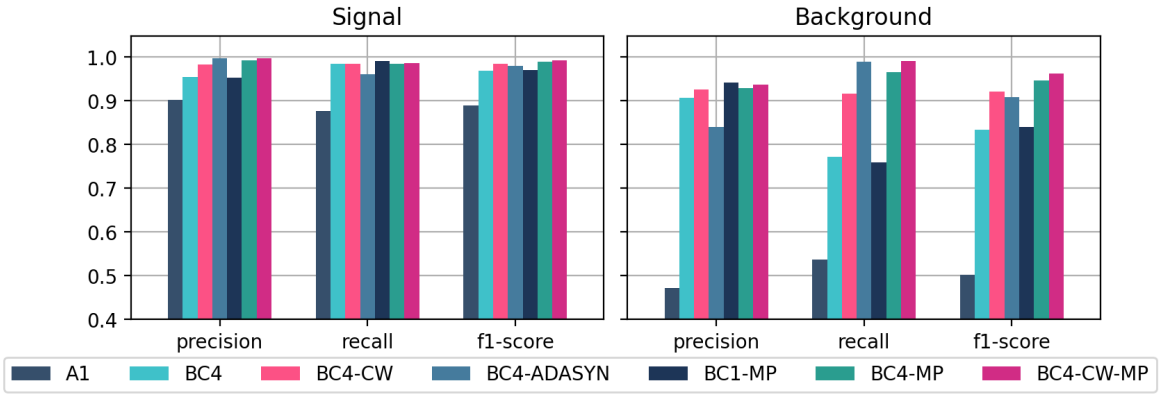



Figure 33: Model scores comparison in Xtohh2000 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| A1 | 0.902 | 0.877 | 0.889 | 0.471 | 0.536 | 0.501 |
| BC4 | 0.955 | 0.984 | 0.969 | 0.908 | 0.772 | 0.834 |
| BC4-CW | 0.983 | 0.985 | 0.984 | 0.925 | 0.917 | 0.921 |
| BC4-ADASYN | 0.998 | 0.961 | 0.979 | 0.840 | 0.990 | 0.909 |
| BC1-MP | 0.952 | 0.990 | 0.971 | 0.942 | 0.759 | 0.841 |
| BC4-MP | 0.993 | 0.985 | 0.989 | 0.929 | 0.966 | 0.947 |
| BC4-CW-MP | 0.998 | 0.986 | 0.992 | 0.937 | 0.991 | 0.963 |

Table 4: Model scores comparison in Xtohh2000 SR2 dataset.

According to these results, every model performed great on signal events, having precision, recall and F1 above 0.85. In background events, the performance metrics were more varied, for instance, the auto encoder A1 performed poorly with 0.501 F1-score, and the neural network BC1 had a better performance with F1 0.841. Therefore, for the case of models applied on the xtohh2000 dataset, the best models were the ones that use class weights to give more importance to the background class, begin the neural network BC4 and its parameterized version the ones with outstanding results.

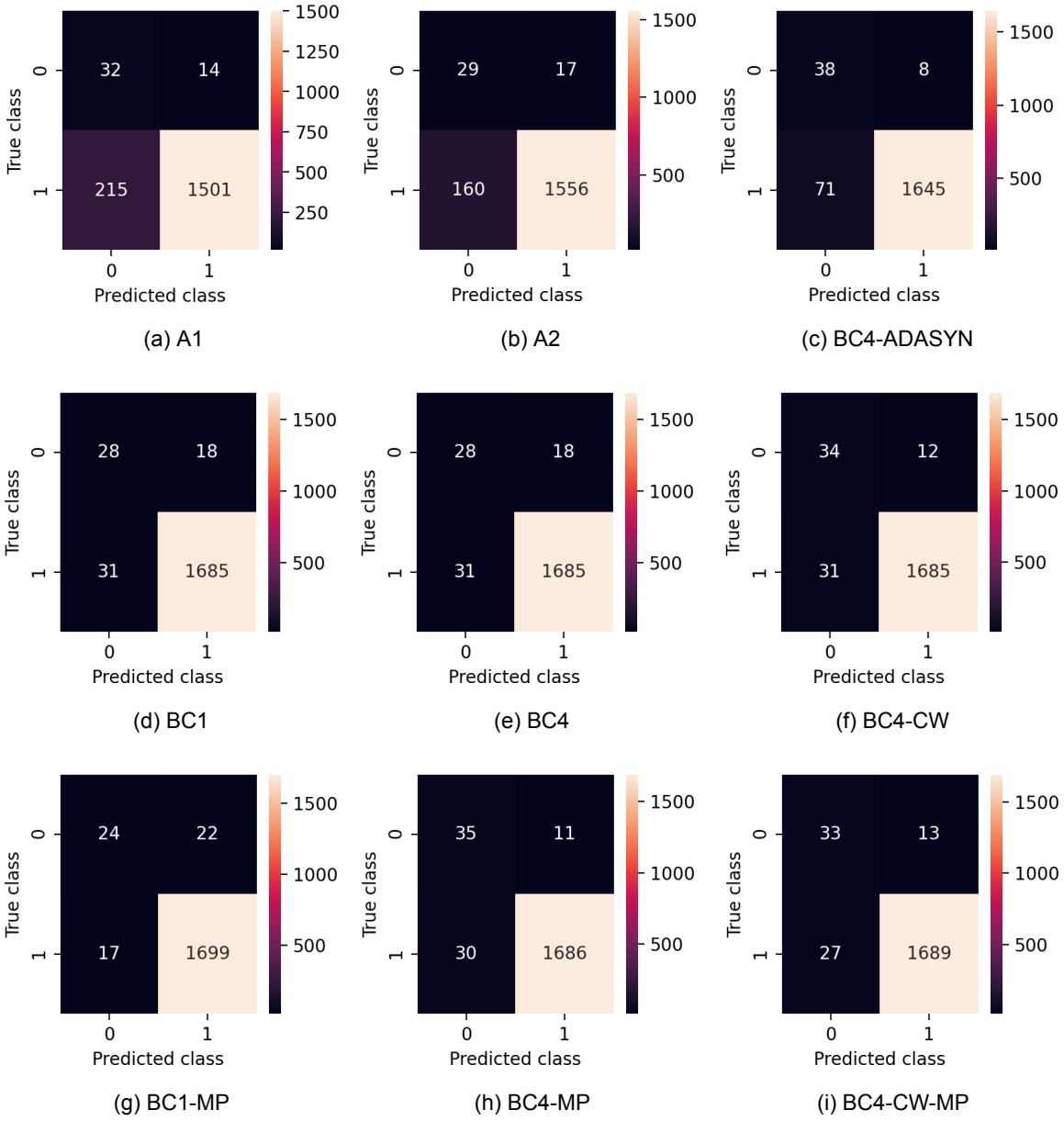
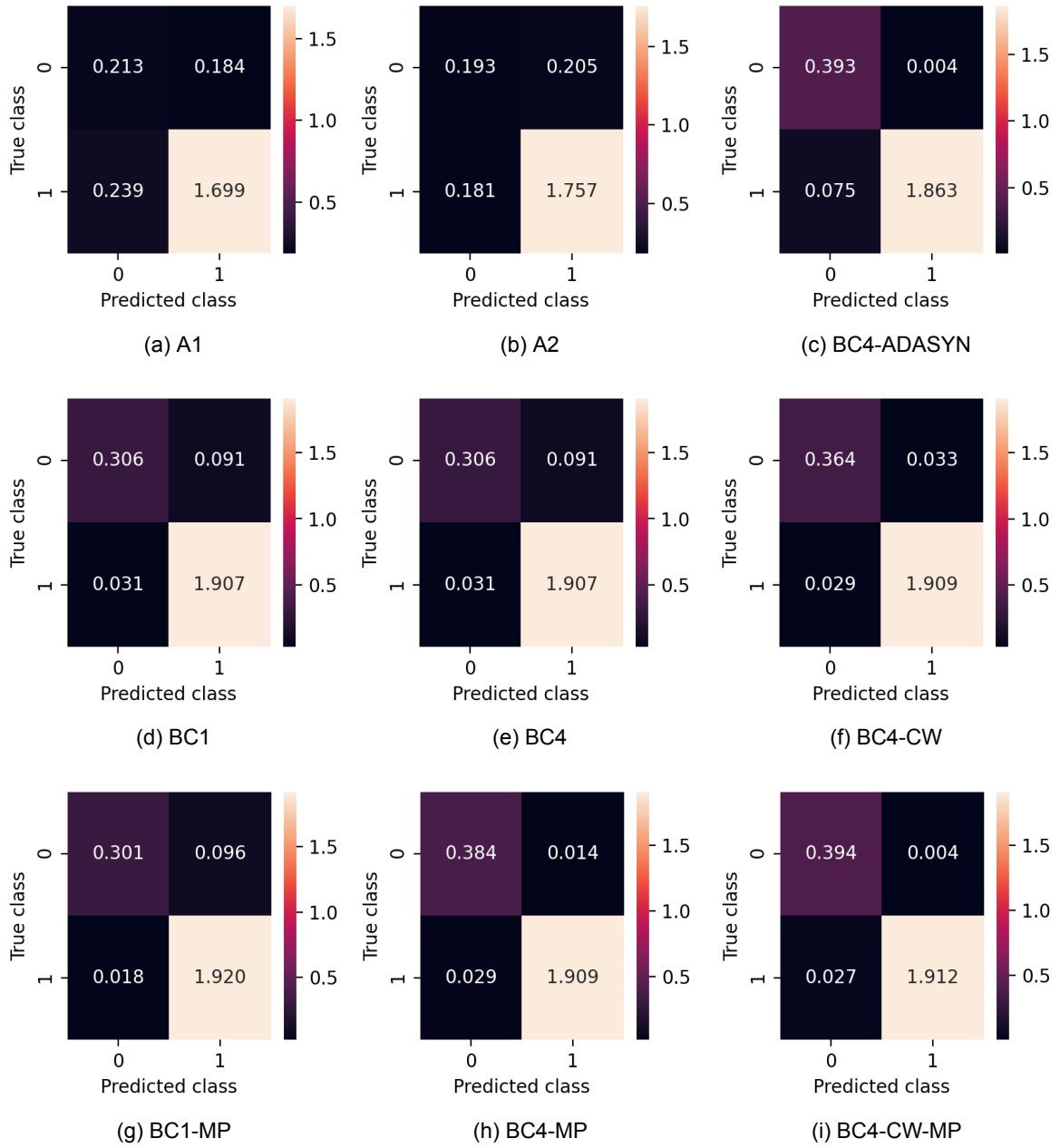



Figure 34: Model confusion matrices in Xtohh2000 SR2 dataset (class 0: background, class 1: Xtohh2000 signal).




(a) A1 (b) A2 (c) BC4-ADASYN

(d) BC1 (e) BC4 (f) BC4-CW

(g) BC1-MP (h) BC4-MP (i) BC4-CW-MP

Figure 35: Model weighted confusion matrices in Xtohh2000 SR2 dataset (class 0: background, class 1: Xtohh2000 signal).

### 5.1.2 Xtohh1000

Any other dataset apart from xtohh2000 was trained and tested using only PNNs, since they were not focused in early stages of this body of work. For xtohh1000, model performance comparison plots can be found in Figure 36, with details in Table 5. Weighted and non-

weighted confusion matrices can be found in Figure 37 and Figure 38.

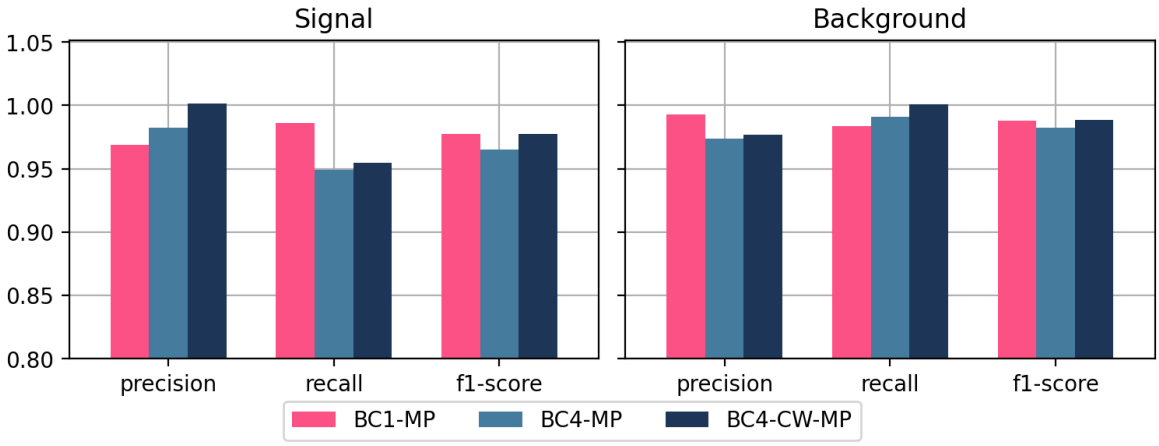
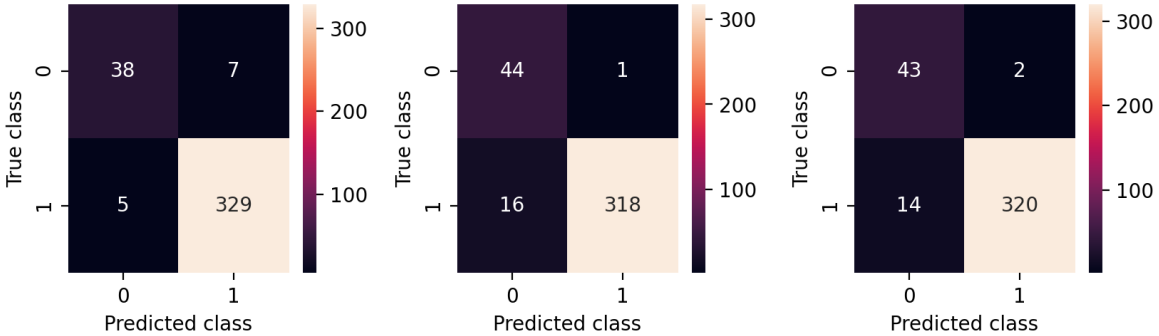



Figure 36: Model scores comparison in Xtohh1000 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.969 | 0.986 | 0.977 | 0.993 | 0.983 | 0.988 |
| BC4-MP | 0.982 | 0.949 | 0.965 | 0.974 | 0.991 | 0.982 |
| BC4-CW-MP | 1.000 | 0.955 | 0.977 | 0.977 | 1.000 | 0.989 |

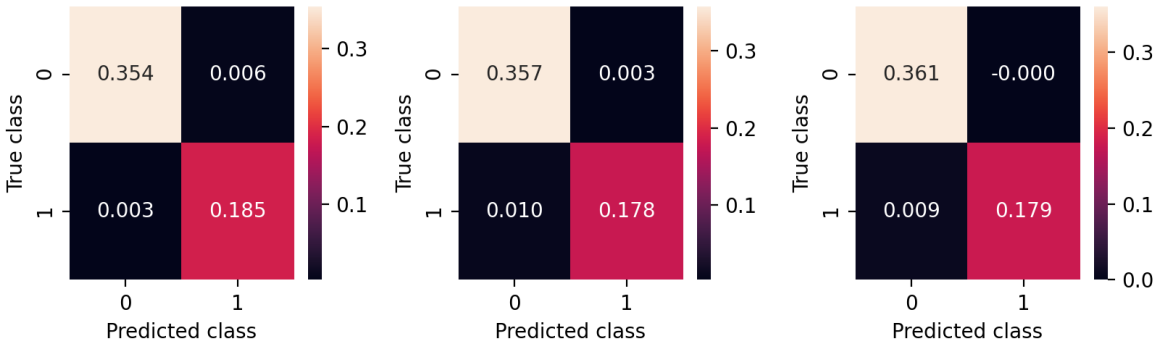
Table 5: Model scores comparison in Xtohh1000 SR2 dataset.

The three models obtained scores above 0.9 in all measurements, being the parameterized version of BC4 with class weights the one that has the best results in the background class, trading off some signal score.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 37: Model confusion matrices in Xtohh1000 SR2 dataset (class 0: background, class 1: Xtohh1000 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 38: Model weighted confusion matrices in Xtohh1000 SR2 dataset (class 0: background, class 1: Xtohh1000 signal).

### 5.1.3 Xtohh1200

For Xtohh1200, model performance comparison plots can be found in Figure 39, with details in Table 6. Weighted and non-weighted confusion matrices can be found in Figure 40 and Figure 41.

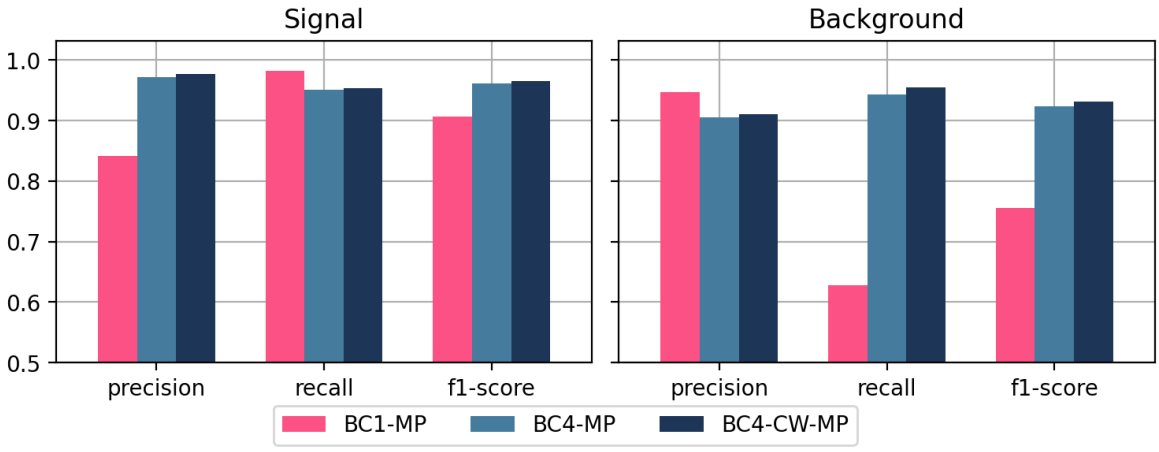
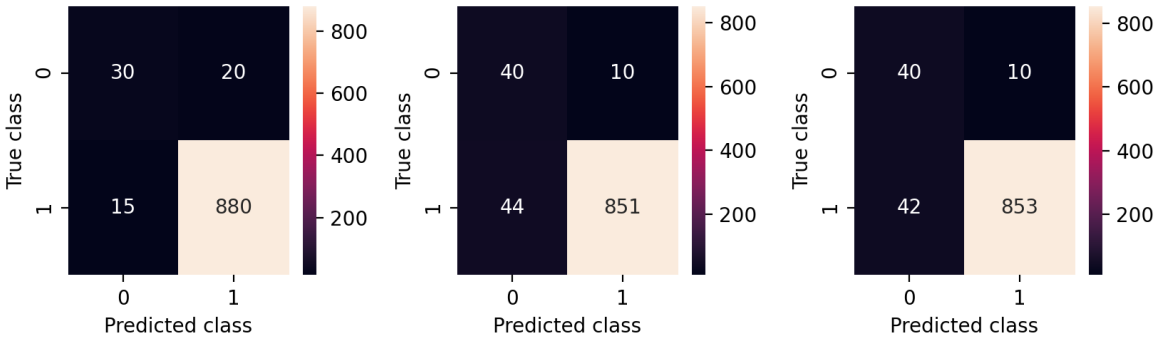



Figure 39: Model scores comparison in Xtohh1200 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.842 | 0.983 | 0.907 | 0.947 | 0.628 | 0.755 |
| BC4-MP | 0.971 | 0.951 | 0.961 | 0.905 | 0.944 | 0.924 |
| BC4-CW-MP | 0.977 | 0.954 | 0.965 | 0.911 | 0.954 | 0.932 |

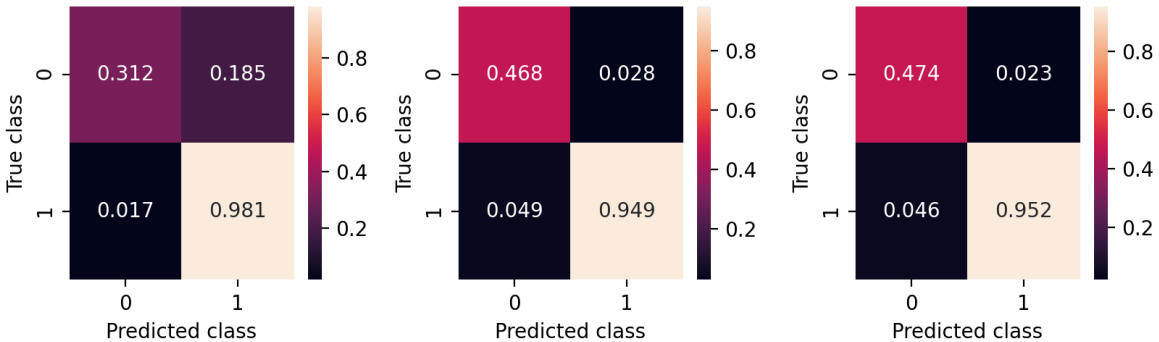
Table 6: Model scores comparison in Xtohh1200 SR2 dataset.

From the three models, parameterized BC1 is the one with the lowest performance, especially in background, having an F-score of 0.76, which is not bad on its own. Both class weighted and non-class weighted parameterized BC4 have great results above 0.9 F1 in both signal and background class.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 40: Model confusion matrices in Xtohh1200 SR2 dataset (class 0: background, class 1: Xtohh1200 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 41: Model weighted confusion matrices in Xtohh1200 SR2 dataset (class 0: background, class 1: Xtohh1200 signal).

### 5.1.4 Xtohh1400

For Xtohh1400, model performance comparison plots can be found in Figure 42, with details in Table 7. Weighted and non-weighted confusion matrices can be found in Figure 43 and Figure 44.

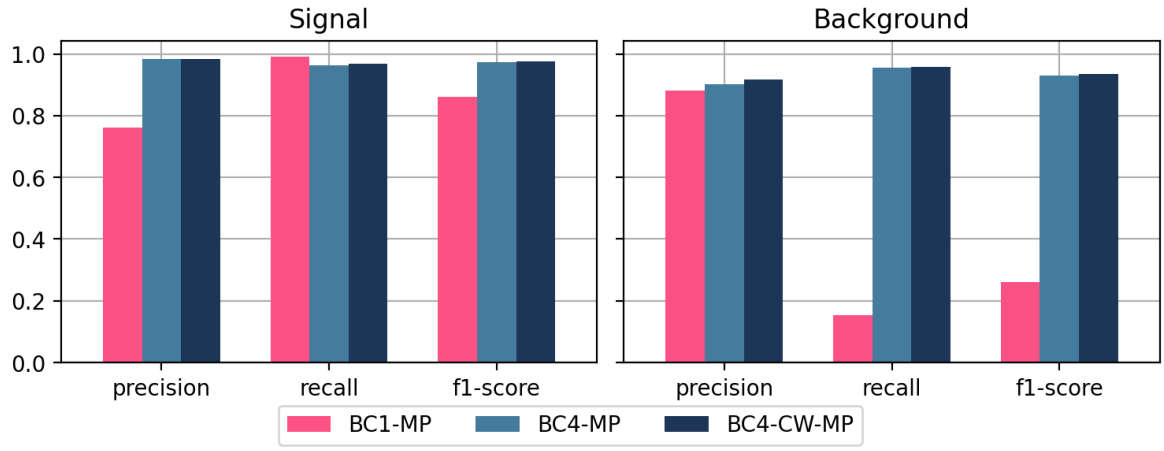
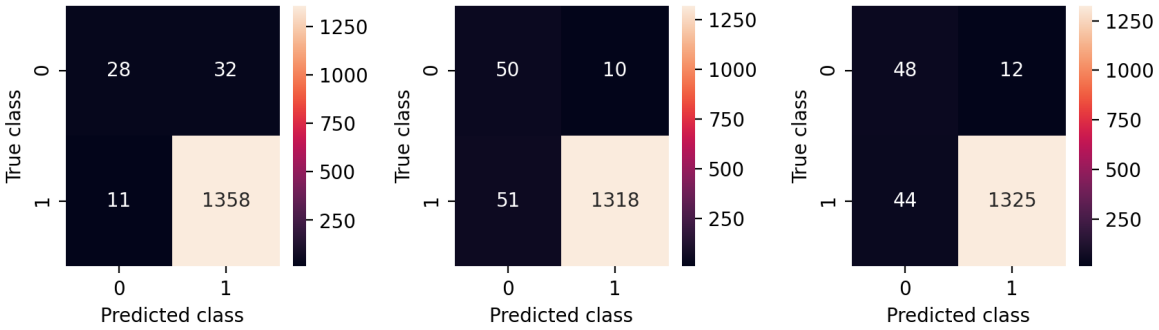



Figure 42: Model scores comparison in Xtohh1400 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.761 | 0.992 | 0.861 | 0.881 | 0.153 | 0.261 |
| BC4-MP | 0.984 | 0.962 | 0.973 | 0.902 | 0.957 | 0.929 |
| BC4-CW-MP | 0.984 | 0.968 | 0.976 | 0.916 | 0.957 | 0.936 |

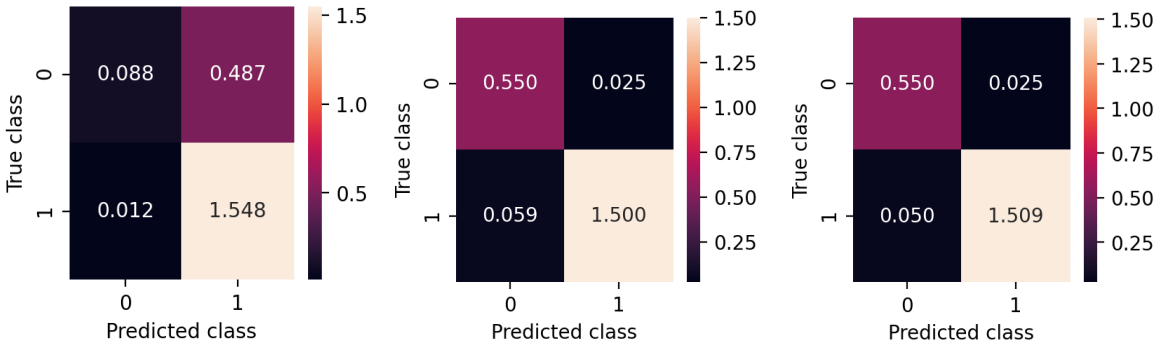
Table 7: Model scores comparison in Xtohh1400 SR2 dataset.

Following the behavior in Xtohh1200, BC1 lags behind respect to BC4 models, having 0.26 F-score, one of the worst scores yet. On the other hand, BC4 performs great in its two versions, in both signal and background class.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 43: Model confusion matrices in Xtohh1400 SR2 dataset (class 0: background, class 1: Xtohh1400 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 44: Model weighted confusion matrices in Xtohh1400 SR2 dataset (class 0: background, class 1: Xtohh1400 signal).

### 5.1.5 Xtohh1600

For Xtohh1600, model performance comparisons plots can be found in Figure 45, with details in Table 8. Weighted and non-weighted confusion matrices can be found in Figure 46 and Figure 47.

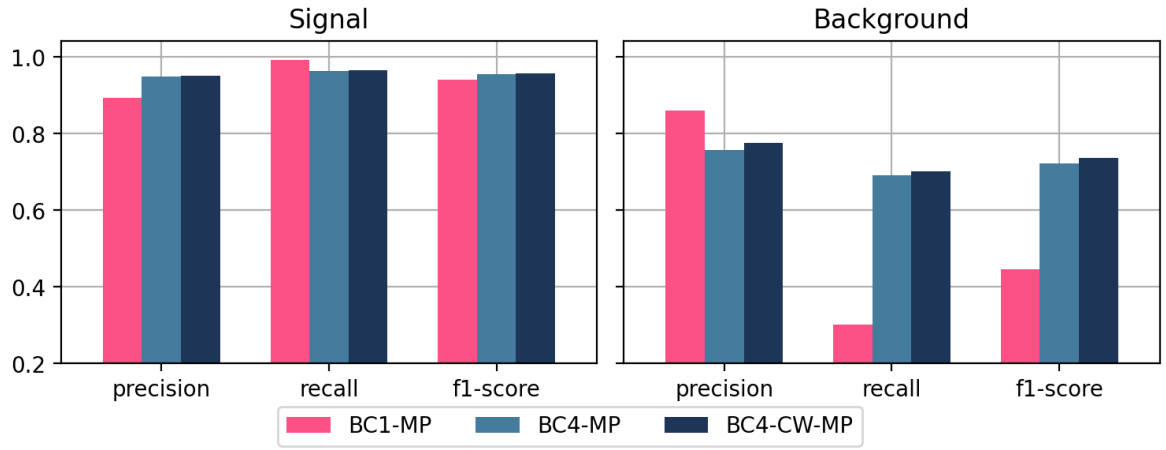
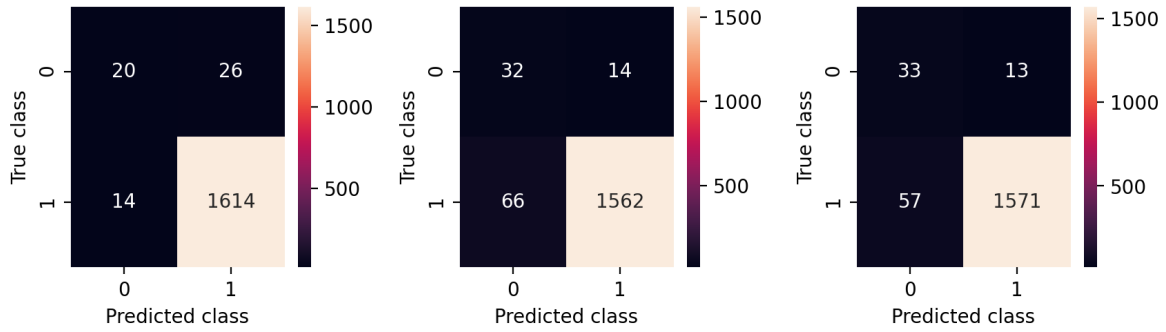



Figure 45: Model scores comparison in Xtohh1600 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.893 | 0.992 | 0.940 | 0.861 | 0.300 | 0.445 |
| BC4-MP | 0.949 | 0.962 | 0.955 | 0.757 | 0.691 | 0.722 |
| BC4-CW-MP | 0.950 | 0.966 | 0.958 | 0.775 | 0.702 | 0.737 |

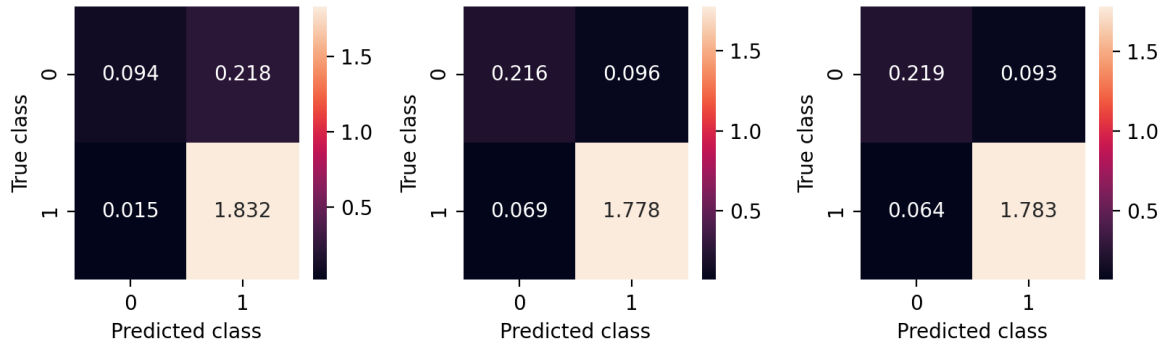
Table 8: Model scores comparison in Xtohh1600 SR2 dataset.

Similarly in previous datasets, BC4 outstands BC1, but not with the same great results. This time, BC4 reaches only 0.7 F1 in background.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 46: Model confusion matrices in Xtohh1600 SR2 dataset (class 0: background, class 1: Xtohh1600 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 47: Model weighted confusion matrices in Xtohh1600 SR2 dataset (class 0: background, class 1: Xtohh1600 signal).

### 5.1.6 Xtohh1800

For Xtohh1800, model performance comparisons plots can be found in Figure 48, with details in Table 9. Weighted and non-weighted confusion matrices can be found in Figure 49 and Figure 50.

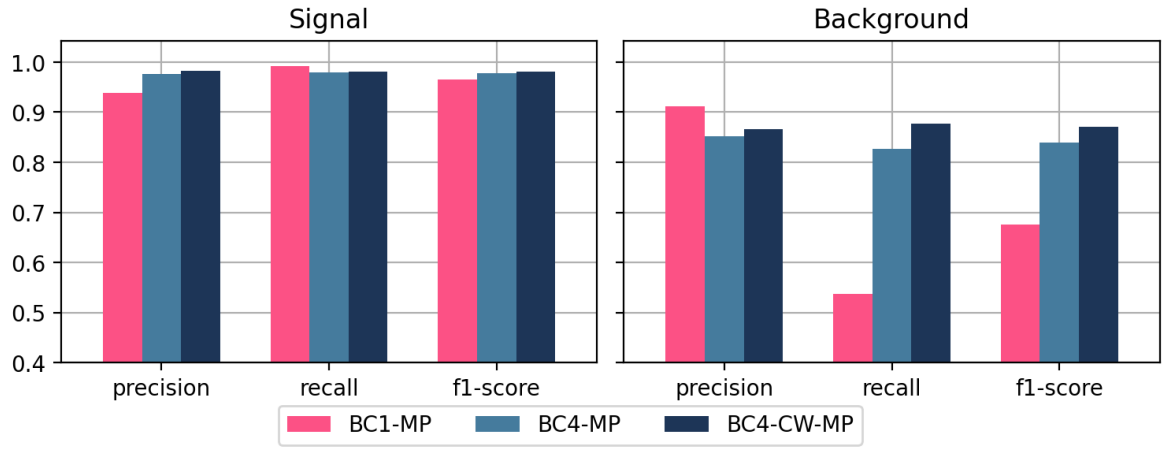
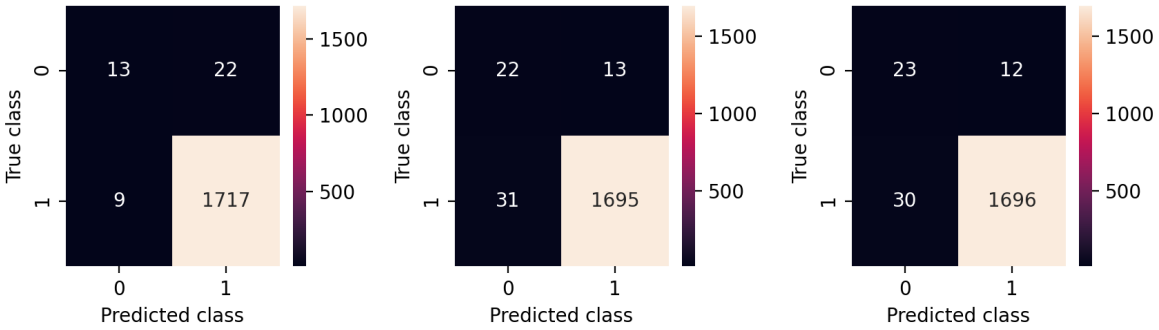



Figure 48: Model scores comparison in Xtohh1800 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.938 | 0.993 | 0.965 | 0.911 | 0.537 | 0.676 |
| BC4-MP | 0.976 | 0.980 | 0.978 | 0.852 | 0.827 | 0.839 |
| BC4-CW-MP | 0.983 | 0.981 | 0.982 | 0.866 | 0.877 | 0.872 |

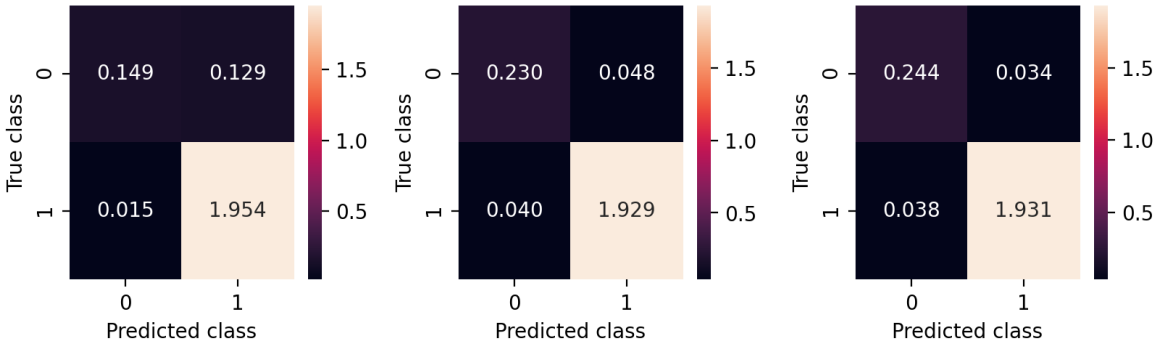
Table 9: Model scores comparison in Xtohh1800 SR2 dataset.

Similarly to the Xtohh1600 case, F1 in background lag behind scores in signal, reaching 0.87 F1 with parameterized BC4 with class weighted. Scores in signal keep reaching above 0.9 F1.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 49: Model confusion matrices in Xtohh1800 SR2 dataset (class 0: background, class 1: Xtohh1800 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 50: Model weighted confusion matrices in Xtohh1800 SR2 dataset (class 0: background, class 1: Xtohh1800 signal).

### 5.1.7 Xtohh2500

For Xtohh2500, model performance comparisons plots can be found in Figure 51, with details in Table 10. Weighted and non-weighted confusion matrices can be found in Figure 52 and Figure 53.

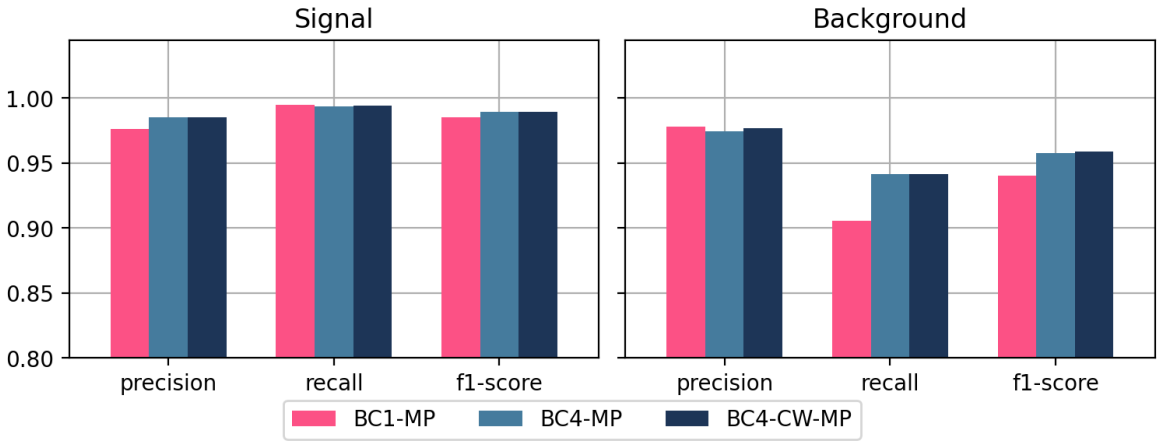
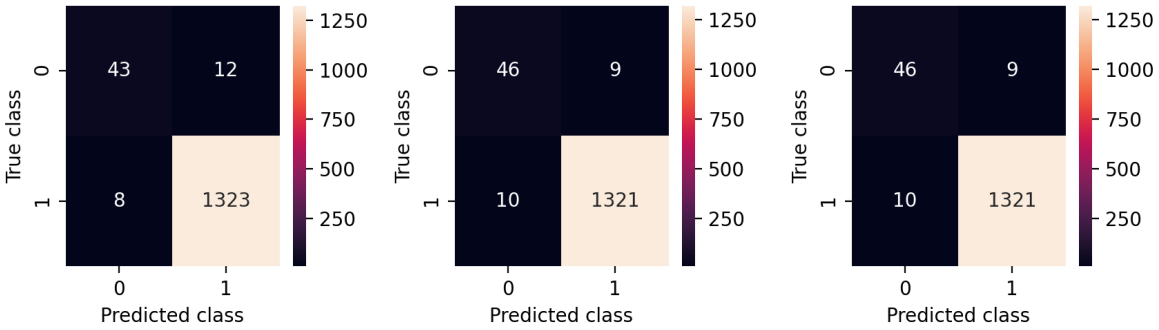



Figure 51: Model scores comparison in Xtohh2500 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.976 | 0.995 | 0.985 | 0.978 | 0.906 | 0.940 |
| BC4-MP | 0.985 | 0.994 | 0.989 | 0.975 | 0.941 | 0.958 |
| BC4-CW-MP | 0.985 | 0.994 | 0.990 | 0.977 | 0.941 | 0.959 |

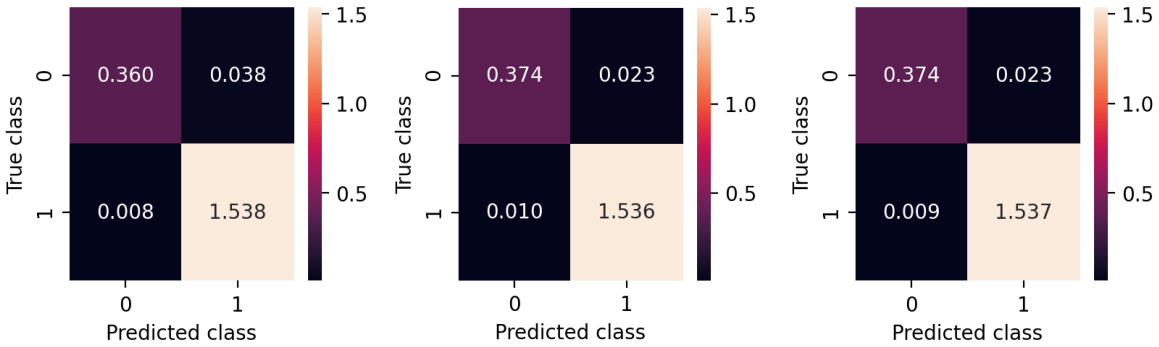
Table 10: Model scores comparison in Xtohh2500 SR2 dataset.

In contrast to the last datasets, scores in background increase again, reaching above 0.9 F-score in all models, being parameterized BC4 with class weights the outstanding one.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 52: Model weighted confusion matrices in Xtohh2500 SR2 dataset (class 0: background, class 1: Xtohh2500 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 53: Model weighted confusion matrices in Xtohh2500 SR2 dataset (class 0: background, class 1: Xtohh2500 signal).

### 5.1.8 Xtohh3000

For Xtohh3000, model performance comparisons plots can be found in Figure 54, with details in Table 11. Weighted and non-weighted confusion matrices can be found in Figure 55 and Figure 56.

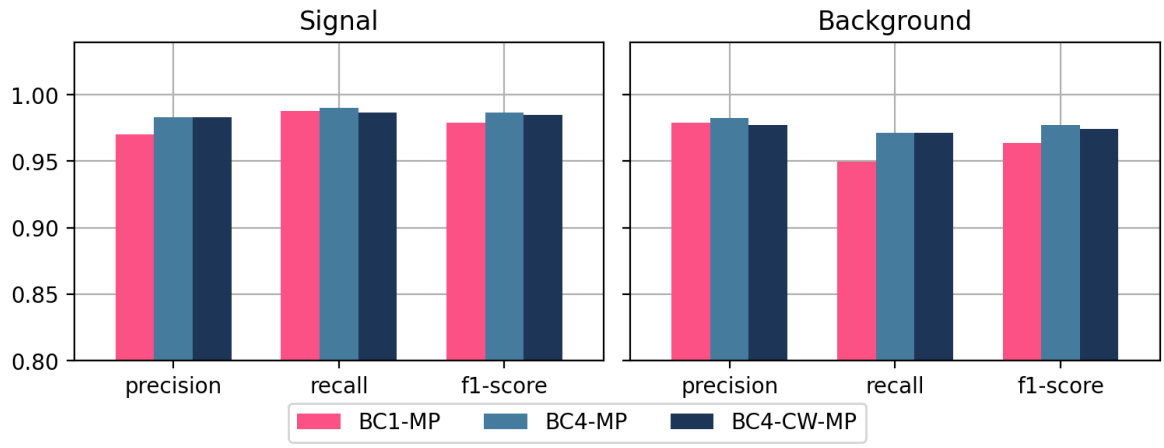
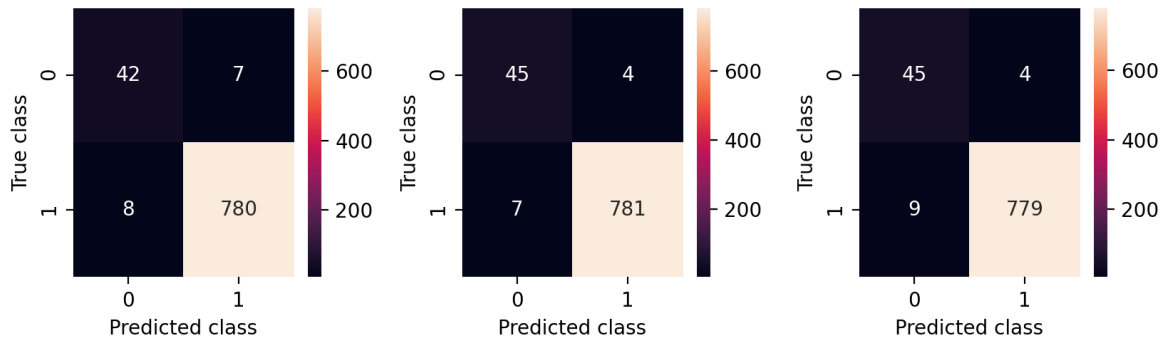



Figure 54: Model scores comparison in Xtohh3000 SR2 dataset.

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Model** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| BC1-MP | 0.970 | 0.988 | 0.979 | 0.979 | 0.949 | 0.964 |
| BC4-MP | 0.983 | 0.990 | 0.986 | 0.983 | 0.971 | 0.977 |
| BC4-CW-MP | 0.983 | 0.986 | 0.985 | 0.977 | 0.971 | 0.974 |

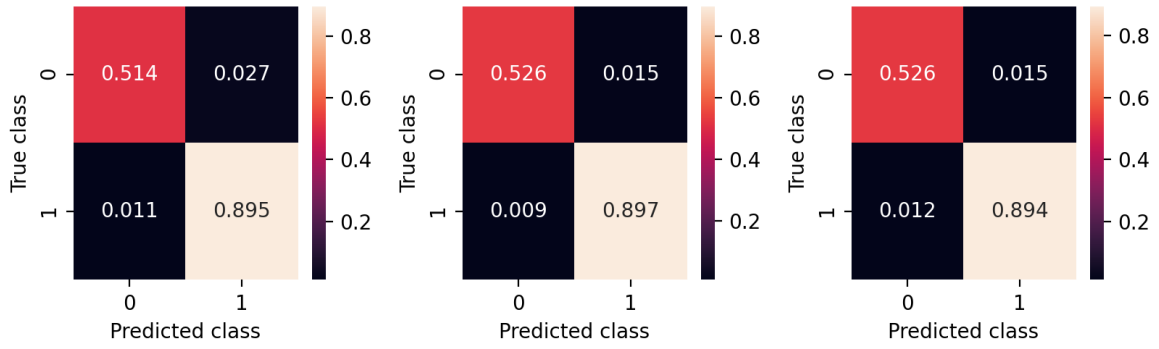
Table 11: Model scores comparison in Xtohh3000 SR2 dataset.

With similar results to Xtohh2500, both F1 in signal and background surpasses 0.95 for all the models. This time, parameterized BC4 without class weights takes the lead, by a small difference with the version with class weights.




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 55: Model confusion matrices in Xtohh3000 SR2 dataset (class 0: background, class 1: Xtohh3000 signal).




(a) BC1-MP (b) BC4-MP (c) BC4-CW-MP

Figure 56: Model weighted confusion matrices in Xtohh3000 SR2 dataset (class 0: background, class 1: Xtohh3000 signal).

## 5.2 Model performance comparison

Comparing results of every model in each Xtohh dataset, BC4-CW-MP was the one with the best performance, outstanding all the other models almost in every dataset, as is shown in Figure 33, Figure 36, Figure 39, Figure 42, Figure 45, Figure 48, Figure 51 and Figure 54. The improved performance of this model is consequence of its approach on the class imbalance issue and the parameterization of the Xtohh mass. Scores of this model in all datasets can be found in Table 12

| | Signal | | | Background | | |
|---|---|---|---|---|---|---|
| **Xtohh** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| 1000 | **1.000** | 0.955 | 0.977 | **0.977** | **1.000** | **0.989** |
| 1200 | 0.977 | 0.954 | 0.965 | 0.911 | 0.954 | 0.932 |
| 1400 | 0.984 | 0.968 | 0.976 | 0.916 | 0.957 | 0.936 |
| 1600 | 0.950 | 0.966 | 0.958 | 0.775 | 0.702 | 0.737 |
| 1800 | 0.983 | 0.981 | 0.982 | 0.866 | 0.877 | 0.872 |
| 2000 | 0.998 | 0.986 | **0.992** | 0.937 | 0.991 | 0.963 |
| 2500 | 0.985 | **0.994** | 0.990 | **0.977** | 0.941 | 0.959 |
| 3000 | 0.983 | 0.986 | 0.985 | **0.977** | 0.971 | 0.974 |

Table 12: Mass Parameterized BC4 with Class Weights (BC4-CW-MP) performance.

Since more weight is given to the background class, mistakes classifying background are punished more then mistakes classifying signal, improving scores on background. This also leads to a small decrease on signal scores, because of the trade-off made by giving less weight to the signal class. But this trade off works on favor of the background, a small sacrifice

on signal score makes a big improvement on background scores, as can be seen in the figures mentioned before.

Parameterization of the mass also plays an important role here. By joining all Xtohh datasets together, models can be trained with much more data then before and enrich its learning, since now have access to information about all the masses of Xtohh. In consequence, now only one model is needed for all the masses of the Xtohh signal.

Other methods to handle class imbalance were not as effective as the class weights approach. ADASYN with 0.2 ratio of minority class was the best oversampling method, but could not improve the results already obtained. This behaviour might be caused by the low amount of background events that the dataset contains. By having a small amount of events, it is difficult to reproduce more of these ones, because the algorithm doesn't have enough relevant information to simulate good background events. ADASYN and SMOTE produced poor simulated data due to this, and BC4 decrease its performance.

Autoencoders were another approach that didn't improve performance on background. The problem of these models is that some signal events were very similar to the background events, getting the same error reconstruction in the autoencoder. This made hard to choose a threshold to separate signal from background, as can be seen in Figure 28 and Figure 29: if the threshold is high, then background events are confused; if the threshold is low, then lots of signal events are confused.

## 5.3 CxAOD analysis framework

To evaluate identification tasks performance and other data analysis tasks, there is an a special framework maintained by the ATLAS collaboration called *CxAODFramework* [CERN, 2020b]. This project groups different data analysis tasks, including the classification task stated in this project in the submodule called *CxAODReader_HH_bbtautau*. To be able to include the evaluation of BDT and DNN models, changes had to be made to this module.

First contact with the framework involved installing it into a CERN account on the LXPLUS Service[9]. Installation is made by cloning the framework repository into a source folder and building it into a build folder, using *CMake*[10] and *Make*[11]. Once is built, executables defined in all *CMakeLists.txt*[12] can be used, including the one that reads models and build histograms with its test scores, called *HHframeworkReadCxAOD*. This executable receives a configuration file and the name of the output folder where the histograms will be built.

In order to evaluate the models trained in this thesis, new functionality was developed into

[9] LXPLUS Service is the interactive logon service to Linux for all CERN users.

[10] CMake is an open-source, cross-platform family of tools designed to build, test and package software.

[11] The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.

[12] CMakeLists.txt file contains a set of directives and instructions describing the project's source files and targets (executable, library, or both).

the library *AnalysisReader_hhbbtt*, which is used by HHframeworkReadCxAOD to evaluate and fill histograms. The first approach taken was to include BDT support to the library, since the framework already has tools to read this type of models. To accomplish this, two new functions were added: `init_MVA()` and `fill_MVA_hists()`. `init_MVA()` initializes the TMVA::Reader[13] with all the variables used to train BDT models, and then reads the BDT model to evaluate. This function is called in the initialization step of the script. Meanwhile, `fill_MVA_hists()` evaluates the model with a specific event and fill the histogram with the resulting score. This functions is called inside the event loop, when all the events are being evaluated. After this additions the framework was able to read and fill histograms with BDT scores.

After successfully integrating reading and evaluation of BDT models in the framework, PNN support was started to be developed. Unlike BDT, there is no tool to read deep learning models in the framework, making this task more challenging. So in order to read and evaluate deep learning models, specifically PNNs, since the best model evaluated here is BC4-CW-MP, a library to read and evaluate these models was developed, called *ParametricNet*, based on the work done in [Deutsch, 2020]. Once this library is ready to go, the same procedures added to the framework for BDTs can be added for PNNs, with a small amount of tweaks.

But before evaluating the model, it needs to be in a specific format for the ParametricNet to read it. This special format is provided by Lightweight Trained Neural Network (LWTNN) [Guest, 2020], which turns saved models to a JSON[14] file that can be read in C++. The script `kerasfunc2json.py` receives the architecture (JSON), the weights (H5[15]) and the inputs (JSON) of the model, and outputs the final JSON that will be read in C++. Using this script, the BC4-CW-MP model was converted to a JSON format. Also, LWTNN provides tools for C++ to read the JSON model. The `parse_json_graph()` method reads the JSON file and the `LightweightGraph` creates a graph from this input. These tools were used to read the BC4-CW-MP model into the CxAODFramework.

To test all the functionality added to the framework and the LWTNN converter, a new model was trained, called *MinimalDNN*. This model only has one hidden layer with 32 neurons, making it faster to train. Since it is desirable to evaluate the model in all the dataset, two versions of the model were trained: an *even model* that trains in even events (events with even identifier) and is tested in odd events (events with odd identifier), and an *odd model* that trains in odd events and is tested in even events. Both models were converted with LWTNN and passed to the framework to evaluate. Results obtained with the framework evaluation are illustrated in Figure 57 and Figure 58, showing that signal score distribution, colored in pink, tends to be closer to 1, and background score distribution, colored in blue, tends to spread in all the range between 0 and 1.

[13] The TMVA::Reader class serves to use the MVAs (Multivariate Data Analysis) in a specific analysis context

[14] JSON is a lightweight data-interchange format, easy for humans to read and write

[15] An H5 file is a data file saved in the Hierarchical Data Format that contains multidimensional arrays of scientific data

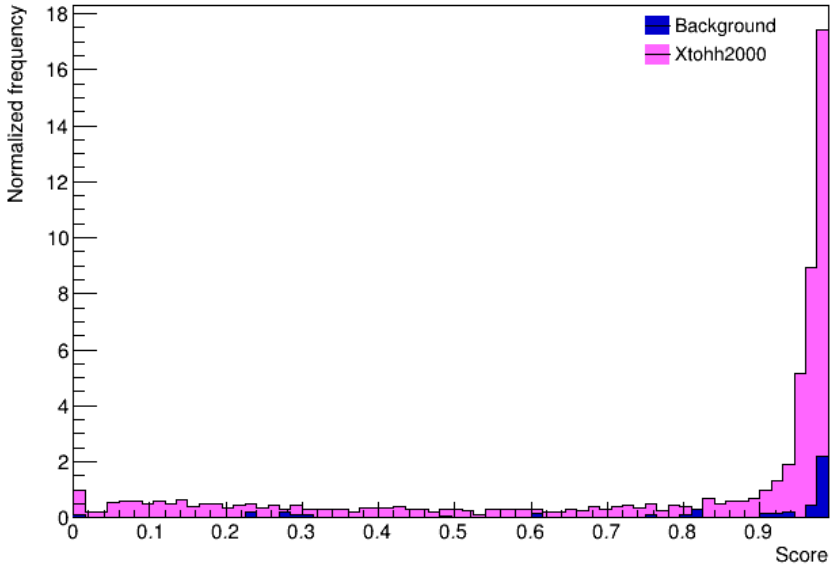



Figure 57: MinimalDNN score distribution in SR2 generated with CxAODFramework.

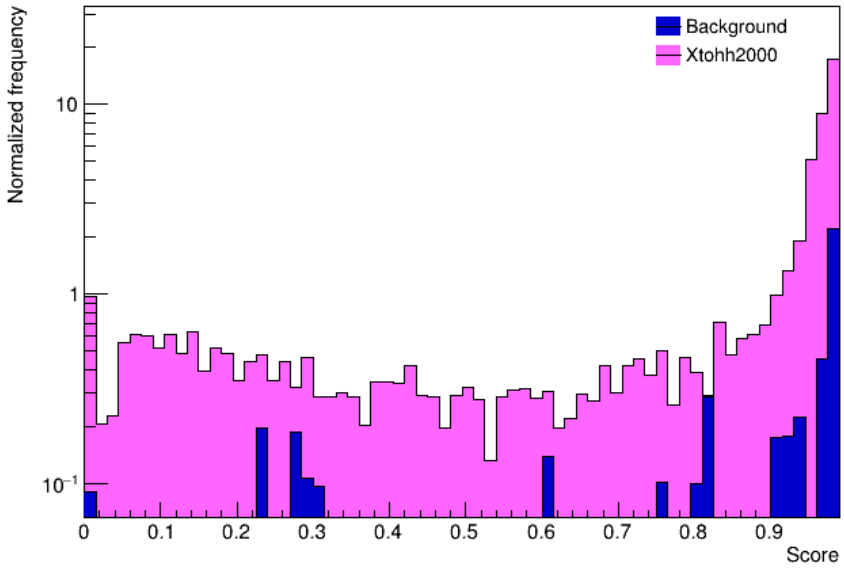



Figure 58: MinimalDNN logarithmic score distribution in SR2 generated with CxAODFramework.

These results show that the framework adaptation and the LWTNN converter are fully working, generating the desirable histograms with the evaluation scores. On the other hand, MinimalDNN has a decent performance on signal events, showing accumulation of events

in scores close to 1, but also has poor performance on background, where the scores don't show a clear tendency. It is worth to remember that MinimalDNN is not the best model developed, but is the model created to test the framework new functionality. Future work involves training and converting the best model to be evaluated in the framework.

For a full view of the code developed check the framework repository in [Rodriguez, 2020b]. Full framework support for PNNs with LWTNN format can be found in [Rodriguez, 2020d] with implementation on CxAODFramework in [Rodriguez, 2020c].

# Chapter 6
# Conclusions

## 6.1 Highlights and discussion

In the present body of work the classification of events coming from the ATLAS experiment to identify boosted di-Higgs signals has been approached. Specifically, this project focused on detecting di-Higgs decaying into $b\bar{b}\tau^+\tau^-$ particles, taking into account the class imbalance problem, the validation metrics, and the validation of the algorithm in the ATLAS analysis system. In conclusion, the main and specific goals stated in Section 1.4 were all accomplished for the data used in this work, meaning that the identification of di-Higgs decaying into $b\bar{b}\tau^+\tau^-$ particles was successful.

In early steps of this research, data coming from ATLAS was processed, turning root files containing events into individual csv datasets, each one containing a Xtohh signal with specific mass and the rest of events that are not Xtohh, i.e. background. This datasets are then preprocessed before entering the ML training process, droping irrelevant columns, leaving only the selected features, and filtering by region, in this case, signal region 2 is the one that this work focuses on.

After the data was ready, base model architecture for neural networks and autoencoders were built. In total, 9 models were developed and trained: autoencoders A1 and A2, DNNs BC1 and BC4, PNNs BC1-MP and BC4-MP, neural network trained with ADASYN oversampling BC4-ADASYN, neural networks trained with class weights BC4-CW and BC4-CW-MP. Parameterized neural networks were trained in all the Xtohh datasets, and standard deep neural networks were trained in Xtohh2000. In deep performance comparison of this models can be found in Section 5.2.

The model with best performance in all Xtohh datasets was BC4-CW-MP, having high F1 scores both in signal and background classes, as is the case of the Xtohh2000 dataset, were this model reached **0.992 F1** on signal and **0.963 F1** on background. This is due to the small trade-off in performance between signal and background, by giving more weight to the background class than the signal class. While performance in signal decreased a small amount, performance in background increased vastly, reaching scores as high as signal scores. On the other hand, parameterizing this model also increased performance thanks to the increased events amount in the dataset, making the model learn even more and generalizing the solution. All results of BC4-CW-MP can be found in Table 12.

After the evaluation of all the models, the best one (BC4-CW-MP) and a BDT model from last year project on this matter [Rodriguez, 2019] were chosen to be implemented in the ATLAS analysis system called CxAODFramework. Fully support for BDT and future support for (BC4-CW-MP) can be found in [Rodriguez, 2020c] repository. Additionally, support for PNN can be found in [Rodriguez, 2020d] repository, where class and methods to be used by the framework are developed. All this done thanks to the collaboration with the Physics

Department and CCTVal.

In terms of reproducibility, all the code developed, for both main work and CxAOD analysis framework can be re-used for other purposes. All the models developed here can be used to identify other types of signals, and all the functions that support this work can also be used. Access to models and data analysis methods can be found in the ADA library repository [Rodriguez, 2020a]. On the other hand, the CxAOD analysis framework version built in this project can be used and modified to evaluate other models by cloning its repository [Rodriguez, 2020c].

## 6.2 Future work

To keep improving in class imbalanced datasets, more approaches need to be tested, especially data-level methods and hybrid methods. Including an improved oversampling method to the class weights method worked in this project can be very beneficial to the model, improving more the results obtained.

Autoencoders are also worth a new revision. Changes on the model that learns the majority class can be made to increase the reconstruction error of the minority class and decrease the reconstruction error of the majority class. This way a threshold that separates both classes can be easily found.

Changes to loss function might be beneficial too, specially trying out the focal loss defined in [Lin et al., 2018]. Direct changes to the loss function can change the way the algorithm behaves and punishes mistakes. This way the loss function can be modified to benefit the minority class.

Last but not least, full evaluation of PNNs will be done with the *CxAODFramework*. In order to accomplish this, the best models developed in this thesis will be trained and converted accordingly so the framework is able to evaluate them. Also, a tuning of this model inside the framework can be done to improve results.

# References

[Aaboud et al., 2017] Aaboud, M., Aad, G., Abbott, B., Abdallah, J., Abdinov, O., Abeloos, B., Aben, R., AbouZeid, O. S., Abraham, N. L., and et al. (2017). Performance of the atlas trigger system in 2015. *The European Physical Journal C*, 77(5).

[Aad et al., 2012] Aad, G. et al. (2012). Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett.*, B716:1–29.

[Albertsson et al., 2019] Albertsson, K. et al. (2019). Machine learning in high energy physics community white paper.

[Alwall et al., 2011] Alwall, J., Herquet, M., Maltoni, F., Mattelaer, O., and Stelzer, T. (2011). Madgraph 5: going beyond. *Journal of High Energy Physics*, 2011(6).

[ATLASCollaboration, 2008] ATLASCollaboration (2008). The atlas experiment at the cern large hadron collider. *JINST*, 3:S08003.

[ATLASCollaboration, 2012] ATLASCollaboration (2012). Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*.

[Baldi et al., 2016] Baldi, P., Cranmer, K., Faucett, T., Sadowski, P., and Whiteson, D. (2016). Parameterized neural networks for high-energy physics. *The European Physical Journal C*, 76(5).

[Bonacorsi et al., 2015] Bonacorsi, D., Diotalevi, T., Magini, N., Sartirana, A., Taze, M., and Wildish, T. (2015). Monitoring data transfer latency in CMS computing operations. *Journal of Physics: Conference Series*, 664(3):032033.

[Brun and Rademakers, 1997] Brun, R. and Rademakers, F. (1997). Root — an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1):81 – 86. New Computing Techniques in Physics Research V.

[CERN, 2020a] CERN (2020a). Cern computing. `https://home.cern/science/computing`.

[CERN, 2020b] CERN (2020b). Cxaod framework. `https://gitlab.cern.ch/CxAODFramework`.

[Chatrchyan et al., 2012] Chatrchyan, S. et al. (2012). Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett.*, B716:30–61.

[Chollet, 2017] Chollet, F. (2017). *Deep Learning with Python*. Manning.

[Deutsch, 2020] Deutsch, C. (2020). Parametric net. `https://gitlab.cern.ch/cdeutsch/parametricnet/-/tree/master`.

[Evans and Bryant, 2008] Evans, L. and Bryant, P. (2008). Lhc machine. *Journal of Instrumentation*, 3(08):S08001.

[Fawcett, 2006] Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874. ROC Analysis in Pattern Recognition.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

[Guest et al., 2018] Guest, D., Cranmer, K., and Whiteson, D. (2018). Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68(1):161–181.

[Guest, 2020] Guest, D. H. (2020). Lightweight trained neural network. https://github.com/lwtnn/lwtnn.

[He and Ma, 2013] He, H. and Ma, Y. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 1st edition.

[Hecht-Nielsen, 1989] Hecht-Nielsen, R. (1989). Theory of the backpropagation neural network. *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1.

[Higgs, 1964] Higgs, P. W. (1964). Broken symmetries and the masses of gauge bosons. *Phys. Rev. Lett.*, 13:508–509.

[Johnson and Khoshgoftaar, 2019] Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27.

[Kostadinov, 2019] Kostadinov, S. (2019). Understanding backpropagation algorithm. `https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd`.

[Kuhn and Johnson, 2013] Kuhn, M. and Johnson, K. (2013). *Applied Predictive Modeling*. Springer.

[Kuznetsov et al., 2016] Kuznetsov, V., Li, T., Giommi, L., Bonacorsi, D., and Wildish, T. (2016). Predicting dataset popularity for the CMS experiment. *J. Phys. Conf. Ser.*, 762(1):012048.

[Lin et al., 2018] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal loss for dense object detection.

[Mitchell, 1997] Mitchell, T. (1997). Machine learning. *McGraw Hill*.

[Oerter and Holstein, 2006] Oerter, R. and Holstein, B. (2006). The theory of almost everything: The standard model, the unsung triumph of modern physics. *Physics Today - PHYS TODAY*, 59.

[P.Chiappettaa et al., 1994] P.Chiappettaa, P.Colangelob, Felicebc, P., G.Nardullibc, and G.Pasquariellod (1994). Higgs search by neural networks at lhc. *Physics Letters B*, 322:219–233.

[Radovic et al., 2018] Radovic, A., Williams, M., Rousseau, D., Kagan, M., Bonacorsi, D., Himmel, A., Aurisano, A., Terao, K., and Wongjirad, T. (2018). Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560.

[Rodriguez, 2019] Rodriguez, A. (2019). Algoritmo de machine learning para la identificaciÓn del boosted di-higgs en el experimento atlas.

[Rodriguez, 2020a] Rodriguez, J. (2020a). Atlas data analysis. `https://gitlab.cern.ch/joirodri/ada`.

[Rodriguez, 2020b] Rodriguez, J. (2020b). Cxaodframework. `https://gitlab.cern.ch/joirodri/cxaodframework/-/tree/joirodri/deep_learning`.

[Rodriguez, 2020c] Rodriguez, J. (2020c). Cxaodreader_hh_bbtautau. `https://gitlab.cern.ch/CxAODFramework/CxAODReader_HH_bbtautau/-/tree/joirodri/deep_learning`.

[Rodriguez, 2020d] Rodriguez, J. (2020d). Hhbbtautauneuralnetwor. `https://gitlab.cern.ch/joirodri/hhbbtautauneuralnetwork`.

[Roe et al., 2005] Roe, B. P., Yang, H.-J., Zhu, J., Liu, Y., Stancu, I., and McGregor, G. (2005). Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nucl.Instrum.Meth*, 543:577–584.

[Rosenblatt, 1957] Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory.

[Sjöstrand et al., 2015] Sjöstrand, T., Ask, S., Christiansen, J. R., Corke, R., Desai, N., Ilten, P., Mrenna, S., Prestel, S., Rasmussen, C. O., and Skands, P. Z. (2015). An introduction to pythia 8.2. *Computer Physics Communications*, 191:159–177.

[Sonneveld, 2019] Sonneveld, J. (2019). Searches for physics beyond the standard model at the lhc.

[Turing, 1950] Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460.