UNIVERSITY
OF OSLO

Master's thesis

The Application of Supervised Machine Learning in the Search For BSM-Physics

A Study of Ordinary Dense, Parametrized and Ensemble Networks and their Application to Physics

William Hirst

Computational Science: Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

4th May 2023

William Hirst

The Application of Supervised Machine Learning in the Search For BSM-Physics

A Study of Ordinary Dense, Parametrized and Ensemble Networks and their Application to Physics

Supervisors:
Professor Farid Ould-Saada
Dr. Eirik Gramstad

Abstract

This will be the abstract.

Acknowledgments

Contents

List of Figures

Introduction

The Standard Model (SM) is one of the most successful scientific theories ever created. It accurately explains the interactions of leptons and quarks as well as the force carrying particles which mediate said interactions. The model is a result of over a century of work demanding the contributions of great minds like Paul Dirac (*1902-1984*), Erwin Schrodinger (*1887-1961*) and Richard Feynman (*1918-1988*). In 2012 the SM achieved one of its crowning successes when we discovered the Higgs boson [1]. Much of the accolade was rightfully given to the theoretical work on the SM, but another aspect of the discovery was equally important. Data analysis was and is a crucial part of any new discovery in physics.

In spite of the success of the SM, there are still questions we are unable to answer. The SM is yet to include gravity or even explain the energy-matter density in the universe. Theoretical physicists are constantly at work trying to accommodate these aspects of the universe, introducing extensions of the SM like Supersymmetry (SUSY) [2] or String Theory [3]. To precisely test these theories we require larger and larger amounts of data. During the period of Run 3 the Large Hadron Collider (LHC) is projected to reach an integrated luminosity of approximately $500 fb^{-1}$ with Run 4 reaching over $1000 fb^{-1}$[1]. This is compared to the current $139 fb^{-1}$ accumulated from Run 2. With the data sets in particle physics progressively growing, so does the demand for numerical algorithms and frameworks to analyze them. One of the most exiting tools which is playing and will play an important role in upholding this demand, is Machine Learning (ML).

ML is rapidly becoming an indispensable tool in many scientific fields. In areas ranging from cancer research to predicting supernova events, machine learning is being applied to problems once thought as impossible to solve. Particle physics is no exception. Jet flavor classification [4], separating jets from gluons [5] or using ML to create efficient signal regions [6] are just some examples where ML is a vital tool. The traditional approach for ML in High Energy Physics (HEP) is the use of Boosted Decision Trees (BDT) and shallow Neural Network (NN). Especially with the release of `XGBoost` [7] in 2014, the BDT has shown to be both stable and able of reaching incredible precision. In later years supervised Deep Neural Networks (DNN) have become more and more popular, partly due to their versatility and diversity in architecture.

This thesis will study ML as it is applied to the search for new physics in proton-proton collisions produced by the LHC and collected by the ATLAS detector. Specifically, I have studied different ML models as they search for chargino-neutralino pair production in final states with three leptons and missing transverse momentum. I explored a range of ML models, such as ordinary dense Neural Network, Parameterized Neural Network [8], ensemble models including layers introduced in the paper by Wang et al. [9] and Boosted Decision Trees. Additionally, I introduced a new layer, Stochastic-Channel-Out, which resembles the channel-out layer described in the aforementioned paper. Each model was studied and tested in their ability to achieve sensitivity in a diverse data set, including events representing different variations of new physics, particularly variations in choice of mass for the chargino and neutralino.

[1] The projections were taken from the graph created by CERN https://lhc-commissioning.web.cern.ch/schedule/images/LHC-nominal-lumi-projection.png (Accessed: 21.04.2023).

Outline of the Thesis

This thesis is divided into 4 chapters: the first two introducing relevant theory and background for the analysis; the third presenting details on the implementation and preparation of the analysis; and the fourth presenting and discussing the results. At the end of the analysis I summarize the findings in my analysis in the *Conclusion & Outlook* section, as well as include some additional figures and tables in the appendix.
The *first* chapter will give an introduction of the SM as well as discuss the new physics I will be searching for. This chapter will introduce relevant phenomenology surrounding particle physics, give a brief description of proton-proton collisions at the LHC and discuss the physics behind the data set utilized in the analysis.
The *second* chapter covers the necessary background in regard to ML and data analysis in general. This chapter will introduce relevant topics surrounding data analysis such as optimization, regularization and hyperparameters. It will explain the algorithms underlying the NN and BDT's, as well as introduce methods to improve the aforementioned methods. In the final parts of the chapter, I will discuss how ML relates to a Beyond Standard Model (BSM) search and how one assess the results.
The *third* chapter describes the implementations of the analysis. This will include discussing the relevant frameworks, data formats and other tools, as well as diving a little further into the data set. Additionally, this chapter will present the preselection cuts and other preprocessing steps used to generate the final data sets, both simulated and measured collision data and present the comparison between the two. Finally, this chapter will present the models I studied in the analysis, displaying the architectures, and explaining the general strategy utilized for training and validating the models.
In the *fourth* and final chapter, I present and discuss the results from the analysis. In the first four sections, I study four different categorize of ML models in how they perform on a subset of the data, as well as different attributes of each model. The fifth section compares the performance of the four aforementioned models, then compare their results with and without a PCA in the sixth section. Finally, I compare the three best models I found when testing on a subset of the data on how well they perform on the complete signal grid, then compare all three to a previous analysis made by ATLAS.

Chapter 1

The Standard Model of Elementary Particles and Beyond

Although this paper will primarily emphasize the ML aspect of a particle physics analysis, there are still terms and phenomena that are worth defining. In this chapter, I will introduce the basics of particle physics, ranging from explaining the leptons and quarks to defining the layers inside a particle detector. Furthermore, I will introduce the relevant extension of the SM, i.e. Supersymmetry (SUSY), and present some Feynman diagrams to give further insight into the data set.

1.1 The Building Blocks

As early as Ancient Greece, humans pondered the nature of the most elementary building blocks of the universe. The Greeks imagined a rope of a given length and a pair of scissors with adjustable size. Then one could ask, how many times can you cut the rope in half? If the answer is less than infinite, what are you left with?
In 1897, Joseph John Thomson (*1856-1940*) discovered the first elementary particle using the Cathode Ray Tube [10]. The particle that Thomson discovered was named the *electron*. Prior to the time of discovery, we believed atoms to be the smallest building blocks. After the discovery of the electron, the discovery of the proton and neutron quickly followed. It was not until more than 50 years after the discovery of the proton (by Ernest Rutherford *1871-1937* [11]) that we discovered that also protons and neutrons could be further dissected to smaller particles. We call these particles quarks. The 'final-piece'[2] of the puzzle came in 1956 [12] when we discovered the (at that time thought of as massless) neutrino. In later years, we have discovered the tau, muon and their 'neutrino partners'. Together, the electron, muon, tau and the neutrinos form the leptons. We refer to both leptons and quarks as fermions.
Upon the evolution of quantum mechanics and physics as a whole, we started to divert our focus from 'what' and over to 'how'. How can we explain all the complex interactions that emerge between these simple particles? Through the creation of the SM and countless experiments, we discovered that forces are interactions between particles and fields. The SM describes forces as fields which are mediated through particles called gauge bosons.
The four fundamental forces responsible in the universe are electromagnetism (Quantum Electro Dynamics (QED)), the weak-force, the strong-force (Quantum Chromo Dynamics (QCD)) and gravity. The most familiar boson is the photon. The photon is responsible for the mediation of QED and is responsible for all electromagnetic effects, such as the ones allowing us to see objects using our eyes. Similarly, the $W^{\pm}$ and Z bosons are responsible for the weak-force which allows for radioactive decay. And the gluon is responsible for QCD which holds the quarks inside the protons and neutrons together. Gravity is the only force not included in the SM, but would (if one day included) presumably have its own force carrying particle, graviton.
The final building block in the universe introduced and described by the SM is the Higgs boson. The Higgs boson was proposed by Robert Brout, Francois Englert and Peter Higgs in 1964 and discovered at CERN in 2012. The Higgs boson, sometimes called the God particle, is responsible for giving mass to the particles in a process called spontaneous symmetry breaking of the electroweak theory [13]. Together the fermions and the bosons make up all the particles in the SM.

[2] Given the nature of this thesis, the existence of further pieces is implied.

Generation	*Flavour*	Mass [MeV]	Charge [Elementary charge]
$1st$	e	0.511	-1
$1st$	ν_e	< 0.001	0
$2nd$	μ	105.66	-1
$2nd$	ν_μ	< 0.17	0
$3rd$	τ	1776.8	-1
$3rd$	ν_τ	< 18.2	0

Table 1.1: A list of all leptons along with their generation, flavor, mass and charge.

1.1.1 The Leptons

In table 1.1, a summary of all leptons is found, along with the respective mass and electric charge. The leptons are all elementary particles with half-integer spin[3], $\pm 1/2$. For reasons that are yet to be known, the leptons come in three generations. Each generation contains a pair of charged and neutral lepton. The first generation contains the electron, e^- and the electron-neutrino, ν_e. The second contains the muon, μ and the muon-neutrino, ν_μ. And the third generation contain the tau, τ^- and ν_τ. The generations are numbered by the mass of the charged lepton, where the first generation is the lightest. As is often the case in particle physics, the heavier a particle, the lower the probability of it being produced in particle interactions. This, and the fact that they behave similarly to jets in a detector, explains why particle physicists often neglect the τ when speaking about leptons.
The charged leptons are all massive particles ranging from a fraction of 1eV to more than a 10^9eV. The neutrinos were up until the turn of the millennia presumed to be massless. This was not only backed by experiments but also by the SM which had not previously been experimentally challenged. In 1998 [14], it was discovered that neutrinos in fact do have mass, although a very small mass. Given the size of the masses we are yet to accurately measure the mass of the neutrinos, but we have found them all to be less than 20 MeV[4].

1.1.2 The Quarks

> 'Three quarks for Muster Mark!
> Sure he hasn't got much of a bark.
> And sure any he has it's all beside the mark.' [15]

The poem above was written by James Joyce (*1882-1941*) in 1939, and was the motivation for Gell-Mann (*1929-2019*) when naming the inner particles of hadrons, quarks. We can categorize quarks as being either a down- or up-type. All down-type quarks have a negative electrical charge equal to 1/3 that of the electron (e) and all positive quarks have a positive charge equal to 2/3 that of the electron (+e). Similarly to leptons, all quarks have a spin equal to 1/2 and like the leptons are divided in three generations. Each generation of quarks are made of a pair of one up- and one down-type quark. The first generation contains the up, u and the down, d quark, the second the charm, c and the strange, s quark and third the top, t and the bottom, b quark. Also similarly to leptons, the higher the generation and mass the more energy is needed to create them.

Generation	*Flavour*	Mass [MeV]	Charge [Elementary charge]
$1st$	u	2.2	$+2/3$
$1st$	d	4.7	$-1/3$
$2nd$	c	1280	$+2/3$
$2nd$	s	96	$-1/3$
$3rd$	t	173100	$+2/3$
$3rd$	b	4180	$-1/3$

Table 1.2: A list of all quarks along with their generation, flavor, mass and charge.

Similarly to how difference in spin allows leptons to stay in an otherwise similar quantum state, the quarks have 'color'. The colors of quarks are what connect them to the strong-force. QCD is what allows quarks to change color. It predicts asymptotic freedom when quarks are free at short distances, also known

[3]Spin is a quantum number which predicts the effect of an applied electromagnetic field.
[4]The lightest neutrino μ_e, is found to have an upper bound of 1eV.

as color confinement. Briefly explained, color confinement results in quarks never existing in isolation but always in a quark-antiquark pair (meson) or in three quark state (baryons) such as protons and neutrons. Given color confinement, quarks are never directly observed in experiments, instead we detect the signature of quarks forming hadrons in a process called hadronization. Quark hadronization leads to narrow, collimated jets of charged particles, which explains why we call these signatures 'jets of hadrons'.

1.2 The Forces

Why do the nuclei of atoms hold together? Why do the electrons revolve around said nuclei? And why can neutrons convert to protons and vice versa? These phenomena are all described through the interactions of leptons and quarks. But how do we describe the interactions? In the previous section I briefly mentioned that the interactions of the leptons and quarks (or fermions) are mediated by the gauge bosons. The explanation of force as a mediation of bosons is the cornerstone of the SM and is explained through the introduction of Quantum Field Theory (QFT). QFT is a precise mathematical framework which relies on the properties of local symmetries (Gauge symmetries) and field theory. Given the scope of this thesis an introduction to QFT will not be given, yet certain attributes of the forces themselves are of interest.

1.2.1 Electromagnetism

The Electromagnetic (EM) force is one of the two macroscopic forces[5]. This means that most people have directly experienced it and therefore have built an intuition for it. If you place two oppositely charged objects close enough, they attract. The closer they are, the more they attract. Electric charge is the property that causes particles to experience electromagnetic forces. This is due to the boson responsible for mediating it, the photon (γ). The photon is massless, and only couples to particles with an electric charge, which means only particles with a non-zero charge can interact through the electromagnetic force.

1.2.2 The Strong Force

The strong force is, alongside the weak force a microscopic force. The strong force holds nuclei together and is (as the name suggests) the strongest force. Similarly to how electric charge plays a role in electromagnetism, the strong force has *color*. Not to be mistaken with the spectrum of frequencies of light, color in the SM is known as the charges associated with the strong force. There are three conserved color charges, 'r', 'b' and 'g', or red, blue and green. Unlike photons which are neutral, the gluons carry color and anticolor and only couple to colorful[6] particles. Due to the three color charges, there exists a collection of independent color states, 8 which corresponds to the number of gluons. The only particles with color are the quarks, antiquarks and gluons, which explain why only they experience the strong force.

1.2.3 The Weak Force

The weak force is the weakest of all the forces of the SM. This is due to the bosons coupling weakly to fermions, and the fact that the $W^{\pm}$ and Z are massive, which makes them short-lived. The charged weak force is the only force which allows for flavor change[7]. They interact only with left-handed particles, meaning particles who's spin is counterclockwise relative to its direction of motion. The charge associated with the weak force is the weak isospin. All left-handed fermions and right handed-handed antifermions have non-zero isospin (1/2), meaning the W and Z bosons interact with both the leptons and the quarks.

1.3 Beyond the Standard Model

1.3.1 Why look beyond?

> 'There is nothing new to be discovered in physics now.
> All that remains is more and more precise measurement.' [16]

[5] The other being gravity.

[6] Meaning particles with a non-zero color charge.

[7] Flavor is a term used to differentiate the fermions, i.e. the six leptons (electron, muon, electron neutrino etc.) and six quarks (top, bottom, charm etc.). A change in flavor therefore means to transition from one of these flavor to another through the weak charge bosons. For example $\mu_e \rightarrow e^- W^+$

The quote above is rumored to have been spoken by William Thompson (*1824–1907*), better known as Lord Kelvin when addressing the British Association for the Advancement of Science in 1900. The statement was followed by a long period of advancements in the field of physics by the likes of Max Planck (*1858–1947*) and Albert Einstein (*1879–1955*). Less than half a decade after Lord Kelvin uttered the famous words, began the development of Quantum Mechanics. Just as Kelvin was wrong back then, would he be wrong today. For although SM explains a large range of phenomena, there are yet many mysteries to explain in the universe and even problems rooted in the SM.

- SM in its current form cannot incorporate *gravity*. The hope has been to integrate gravity into SM through the discovery of a gravity-carrying particle, the gravtion [17]. So far, no-such particle is found.
- *Dark matter* and *dark energy* make up more than 90% of the energy-density in the observable universe, but is found to lie beyond the SM [18].
- Inflation is today the leading explanation to what happened in the early-stages (the first fraction of a second) of the universe. It explains a universe in which all space undergoes a rapid increase in rate of expansion. None of the fields explained by the SM are capable of causing any such expansion.
- Finally, what is the origin of the neutrino mass and is Charge-Parity (CP) violated in the neutrino sector?

1.3.2 Supersymmetry, the Chargino and the Neutralino

Supersymmetry (SUSY) has for many years been an interesting candidate for BSM physics. SUSY aims to extend the SM as to introduce a symmetry between matter and force (i.e. fermions and bosons). SUSY suggests that each SM particle has an additional Superpartner (SP) which we call a sparticle. The sparticles all differ by half a spin from their original SM particle. The symmetry introduced by SUSY is what is known as a broken symmetry. This is because sparticles are predicted to be much heavier than their corresponding SM SP, often in the range of 100-1000GeV. The difference in spin means that the SP of a fermion is a scalar boson and the SP of a boson is a fermion. SUSY is a candidate to address many problems in physics, some of which are: the hierarchy problem; fixing the mass of the Higgs; and possibly the mystery of dark matter. There are many variants of SUSY, all of which introduce a set of new particles. In this thesis I will study a signal which stems from the simplest variant of SUSY, which is the most similar to the SM, namely the Minimal Supersymmetric Standard Model (MSSM).
In this thesis I will be studying ML models as they process data including particles introduced by MSSM, the *neutralino* ($\tilde{\chi}_{1,2,3,4}$[8]) and the *chargino*($\tilde{\chi}^{\pm}_{1,2}$). The neutralino is the lightest sparticle introduced by MSSM, and is therefore stable, as it can not decay into a lighter sparticle. It is the sparticle of a mixture of the neutral gauge bosons introduced by the weak force, and the Higgs gauge boson, making the neutralino a fermion. Similar to the neutrino, the neutralino has no EM charge, and only interacts through the weak force, making it an ideal candidate for dark matter. The chargino is similar to the neutralino, with the exception that it is electrically charged. For a more thorough explanation of MSSM and its application, the reader is referred to [2].

1.4 Proton-Proton collisions at the LHC

1.4.1 An Introduction to Particle Accelerators and Detectors

With a circumference of 27 km, the LHC *particle accelerator* is the largest piece of scientific equipment ever built. It consists of two separate large tubes aligned with powerful magnets. An electromagnetic field is applied to accelerate bunches[9] of hadrons (specifically protons) and lead, and the magnets are used to bend and focus the bunches. One set of bunches are accelerated in one of the tubes, and another set is accelerated in the other direction inside the other tube. The bunches are accelerated to relativistic speeds. (v=0.99999991c) before they collide at a rate of once every 25 nanosecond. The energy released from the collisions enable the creations of new particles.
To measure the collisions, we use *particle detectors*. In this thesis I will be using data collected by the ATLAS (A Toroidal LHC Apparatus) detector. The ATLAS-detector is the largest general-purpose detector at the LHC and took first data from particle collisions in 2009. The detector consists of several layered cylinders and end-caps around the point of collision. In figure 1.1, taken from the ATLAS collaboration [19]

[8] I will in this thesis use the notation of $\tilde{\chi}_{1,2,3,4}$ to refer to the neutralino, instead of $\tilde{\chi}^0_{1,2,3,4}$
[9] Packets of around 10^{11} protons.

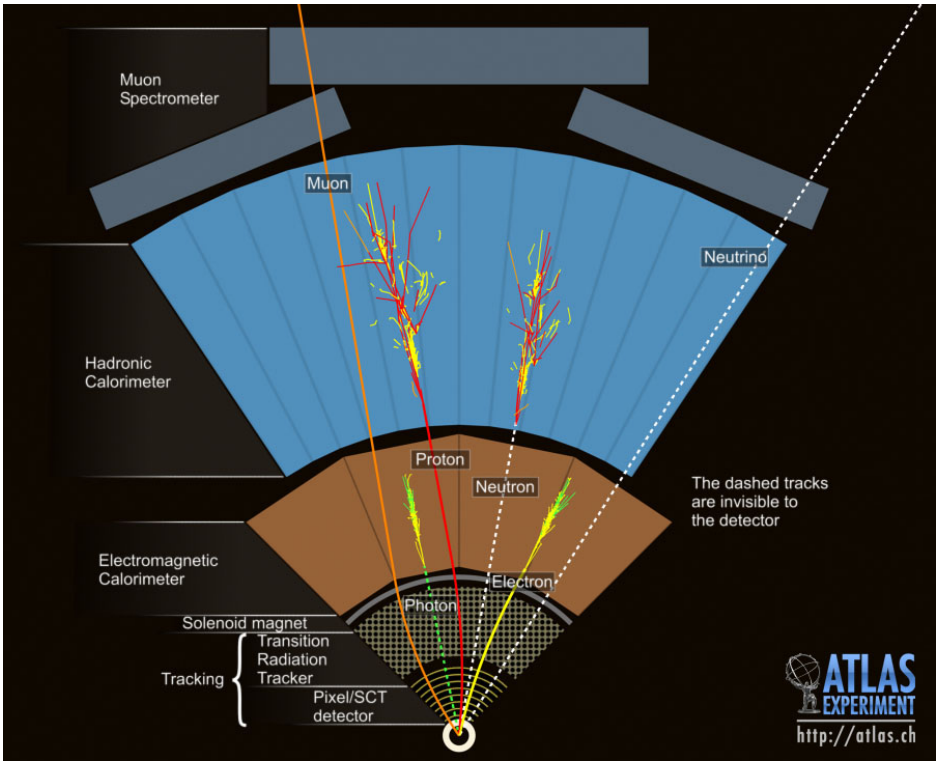


Figure 1.1: Event Cross-Section in a computer generated image of the ATLAS detector [19].

the cross-section of the detector along with the paths of different particles is visualized. The inside of a detector can be summarized in the following points, listed from innermost to outermost layer:

- *Inner Detector*: The inner detector consists of three layers, Pixel detector, Semi-Conductor Tracker and Transition Radiation Tracker. Its purpose is to measure EM interactions between the particles produced in the collision and the material in the layers. The measurements are made at discrete points and can be used to infer the trajectories of the particles. A magnetic field is applied to the inner detector to bend the paths of the particles, enabling the measurement of the momentum and charge.
- *Calorimeters*: The ATLAS detector has two calorimeters, the *Electromagnetic calorimeter* and the *Hadronic calorimeter*. The EM calorimeter is the first layer of the two and measures the energy of photons and electrons interacting through the EM field. The hadronic calorimeter is designed to measure the energy of hadrons (i.e. protons, neutrons, mesons etc.).
- *Muon Spectrometer*: Contrary to electrons, photons and hadrons, the muons do not stop in the calorimeters. The muon spectrometer measures their trajectories in the outer part of the detector allowing the extrapolation of the path from the inner detector to muon spectrometer. Similarly to the inner detector, the muon spectrometer is surrounded by a magnetic field to deflect the path of the particles which allows it to measure momentum and charge.

1.4.2 Kinematics

The kinematic variables of a particle collision are crucial in any HEP analysis and are explained using simple geometry. In figure 1.2 I have drawn a simple axis to illustrate the kinematics of a particle in both the longitudinal (left) and transverse (right) plane. In a three-dimensional axis where the particles colliding travel along the z-axis, the zy-axis defines the longitudinal plane. The angle between the z-axis and the direction of the momentum of one of the particles, defines the polar angle of said particle, θ. The xy-axis define the transverse plane and the angle between the x-axis and the direction of the transverse momentum define the azimuthal angle, ϕ. The transverse momentum and the azimuthal angle are both used in this analysis and are used to create further features. Instead of the polar angle, a preferred feature is the pseudorapidity, η. The pseudorapidity is defined as

$$\eta = -ln\left[tan\left(\frac{\theta}{2}\right)\right] \tag{1.1}$$

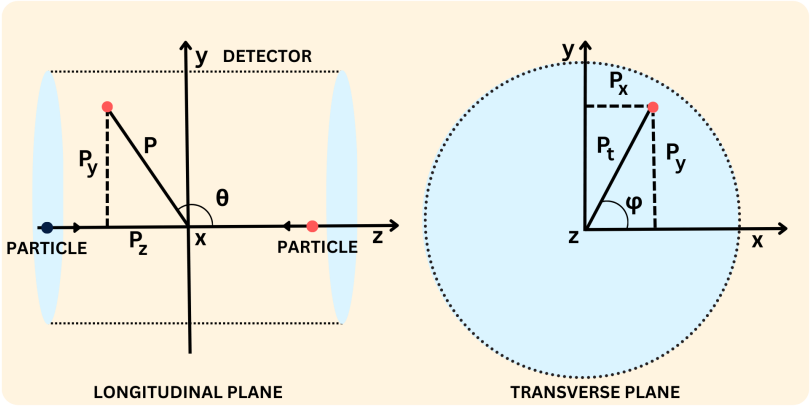


Figure 1.2: An illustration of the general kinematics in a particle-collision, inspired by the figure in the paper by Gramstad [20]. The illustration shows both the longitudial and transverse plane.

The pseudorapidity is preferred to the polar angle because differences in η are Lorentz invariant under boosts along the longitudinal axis. A large range of features are created using the variables described above. In the appendix I have added a summary of all the features used in this analysis (see table 2). One of these features is the distance between the particle in the $\eta\phi - axis$, ΔR. We define ΔR as

$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}. \tag{1.2}$$

1.5 The Signal

In this thesis, I will compare ML models in their ability to learn the trends of the data, which will contribute to our ability to separate the signal from the background. Specifically for this thesis, I will be studying an expansion of the SM which includes SUSY (see section 1.3.2). The signal I will aim to separate from the background, is one which produces a WZ pair, through a chargino-neutralino pair. Figure 1.3 shows a Feynman diagram for such a process. The Feynman diagram shows a chargino-neutralino pair, ($\tilde{\chi}_1^{\pm}$ and $\tilde{\chi}_2$) where both sparticles produce a boson (W and Z respectively) and a $\tilde{\chi}_1$, the lightest neutralino. Given that the neutralinos in the final state are neutral, there are three leptons and a large amount of *missing transverse energy* in the final state. In the section introducing the particle detector 1.4.1, I described how many of the layers rely on interactions in the EM field. Particles which only interact through the weak force are highly unlikely to interact with the detector and so in general cannot be seen. However, their presence can be inferred if they carry away sufficient transverse energy/momentum to measurably unbalance the event, leading to the concept of missing transverse energy.

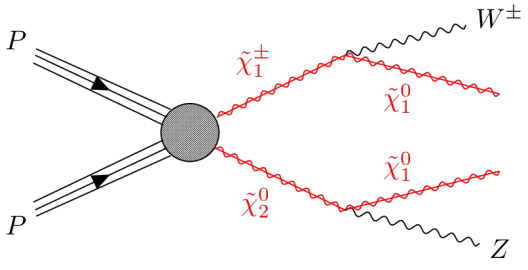


Figure 1.3: The Feynman diagram of the WZ-signal.

1.6 The Background Channels

We define background as anything that is not of interest, i.e. not signal. As will be further explained in later sections, we will explicitly demand a three lepton final state in all collisions considered in the analysis. This will remove a lot of background, but not all. Due to the imperfect nature of the reconstruction of events, a demand for three lepton final state will not be without errors. This leaves room for more variation in the

background than one might expect. In this section I will cover the channels[10] which will be of importance during the analysis. I will also discuss which background channels are the hardest to reduce in a potential new physics signal region, also called the irreducible background. These are channels whose features are largely indistinguishable from the signal. Note that the sections bellow are listed by the size of the contribution to the three lepton final state data (from greatest to least).
The processes are classified similarly to how they were classified in the original Monte Carlo (MC) simulated data produced by ATLAS and used in this study. For the sake of making several of the figures in the thesis more readable, I decided to merge several of the classifications. Diboson(llll), diboson(lll) and diboson(ll) are in later figures categorized as simply diboson. Similarly, the three smallest processes, triboson, W-jets and Higgs, are all merged to the category *Others*.

Z-Jets

The Z-jets channel contribute the most amount of events in the data. The channel consists of all events resulting in a Z-boson alongside jets. If the Z-boson decays into two leptons, the additional jet can act as a fake lepton and looks like a three lepton final state. In figure 1.4a I have written the Feynman diagram of an example of such a channel. The figure shows a quark-antiquark pair leading to a Z-boson and gluon. The Z boson decays into two leptons (normally e^-e^+ or $\mu^-\mu^+$) and the gluon hadronizes as a jet of hadrons which may obtain a b-hadron. From this point, the b-hadron can decay semileptonically, creating what is called non-prompt leptons, or be misidentified as a lepton. Both scenarios constitute what is normally called fake non-prompt leptons.

Diboson (lll)

Diboson channels are defined as channels resulting in two bosons. In the case of (lll), the dibosons decay into a total of three leptons. In figure 1.4b I have drawn the Feynman diagram of an example of such a channel. The figure shows a W- and Z-boson production through a quark-antiquark pair. The W-boson decays into a lepton with missing transverse energy and the Z-boson decays into a pair of leptons. The similarity to the signal, i.e. a three lepton final state with missing transverse energy, makes the diboson(lll) process hard to separate from the signal.

$t\bar{t}$

The $t\bar{t}$ channel is defined as a proton-proton collision resulting in a top quark-antiquark production through the strong interaction. In figure 1.4c I have drawn a Feynman diagram of an example of such a channel. The figure shows gluon-gluon fusion producing a pair of top quarks. The top quark-antiquark pair decay into a bottom-quark and a W boson. The channel constitutes a background when both W bosons decay into a charged and a neutral lepton and one of the b-quarks.

Diboson (llll)

In the case of diboson (llll), the channel refers to events resulting in two Z-bosons which decay into four leptons. In figure 1.4d I have drawn a Feynman diagram of an example of such a diagram. The figure shows a quark-antiquark pair annihilating into two Z-bosons. The two Z-bosons decay into two pairs of leptons. This process constitutes a background when one of the leptons is not reconstructed in the detector.

Top Others

The top other channel is similar to the $t\bar{t}$ channel, in that it results in top quarks. The main difference between the two, is that the top other process does not produce a top quark-antiquark pair through strong interaction of quarks. In figure 1.4e I have drawn an example of a top other process, where a top quark-antiquark pair is produced from a bottom quark-antiquark pair collision mediated through a W boson. From this point, the top quark-antiquark pair decays to a similar state as described in section regarding the $t\bar{t}$ processes. Both the top other and $t\bar{t}$ processes exhibit large amounts of missing transverse energy in the final state, making them both difficult to separate from the signal.

[10]By channels, we refer to the stages by which a particle decays to a specific final state. Often this refers to the different particles which mediates the process from initial- to final state.

Single Top

The single top channel, similarly to top other and the $t\bar{t}$ channel also produces a top quark, but in the case of single top only one top quark is produced. In figure 1.4e I have drawn the Feynman diagram of such a channel. The Feynman diagram displays a top quark produced through the strong interaction of a bottom quark and a gluon. The top quark is produced through the interaction with a W boson which decays into a charged-neutral lepton pair and the top quark decays to a W boson and bottom quark.

Diboson (ll)

The diboson(ll), similarly to diboson(llll) produces two bosons, but instead of ending in a four lepton final state, ends in a two lepton final state. Figure 1.4g, displays the Feynman diagram of a diboson(ll) process where a W-pair is produced through the annihilation of two fermions through a charged boson. To produce the two lepton final state, the two bosons must each decay into a charged-neutral lepton pair.

Triboson

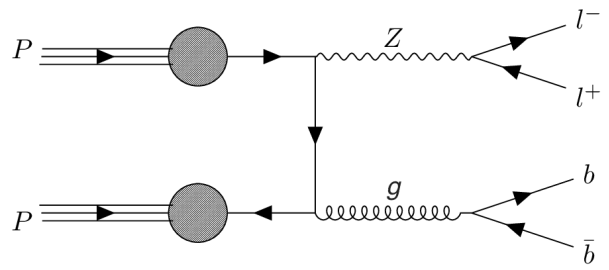
The triboson channel is defined as a proton-proton collision producing three bosons. In figure 1.4h, I have drawn a Feynman diagram displaying an example of a triboson process. The Feynman diagram shows a quark-antiquark pair annihilating to three bosons, two W and one Z. The Z-boson decays to a pair of leptons, and the two W bosons each decay into a charged-neutral lepton pair resulting in a 4-lepton final state with missing transverse energy.

W-jets

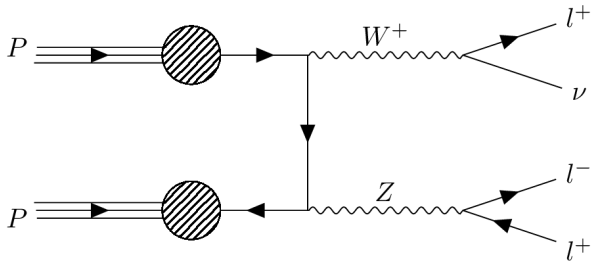
The W-jets is defined as a proton-proton collision producing a W boson alongside jets. In figure 1.4i I have drawn an example of a Feynman diagram for a W-jets processes. The Feynman diagram displays a pair of quarks colliding to form a W boson alongside a gluon. The gluon decays into a bottom quark-antiquark pair and the W boson decays into a charged-neutral lepton pair. Given that this only produces the one lepton, it relies heavily on poor reconstruction, and is hence quite rare.

Higgs

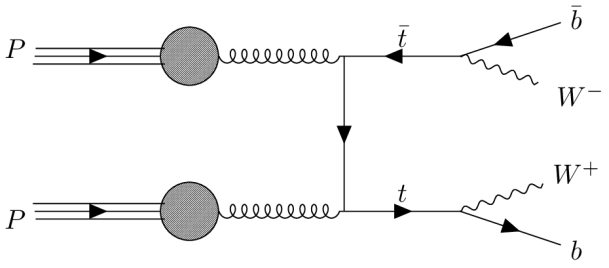
Finally, we have the Higgs process, which is defined as proton-proton collision producing a Higgs boson. The Higgs boson is relatively heavy (125 Gev) and couples to mass. Therefore, the Higgs process is the rarest process in the data set. In figure 1.4j, I have drawn a Feynman diagram of a collision producing a Higgs boson. The Feynman diagram displays a gluon pair annihilation through exchanging a virtual top quark, and producing a virtual top quark-antiquark pair. The latter pair, annihilate, producing a Higgs boson, which quickly (due to its mass) into a pair of Z bosons where one is virtual. The Z boson pair then further decay into a total of 4 leptons.



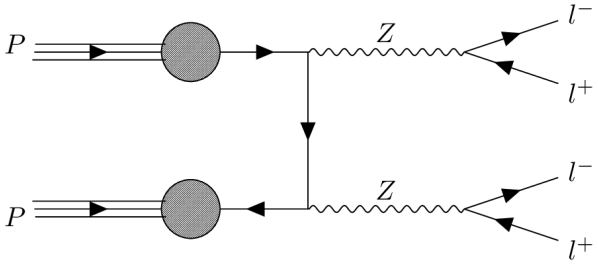
(a)



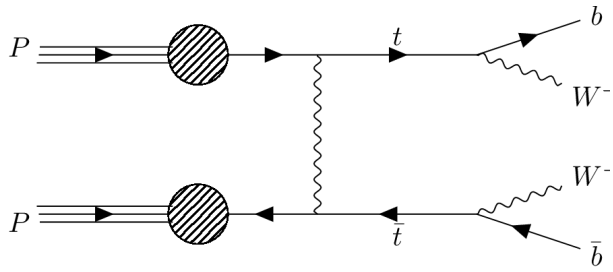
(b)



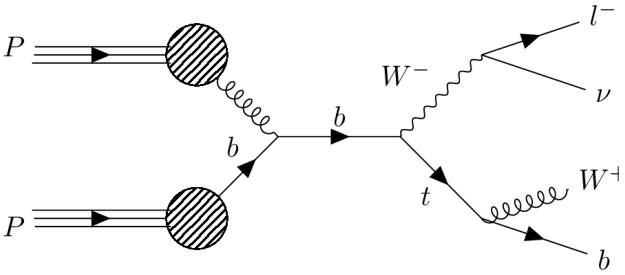
(c)



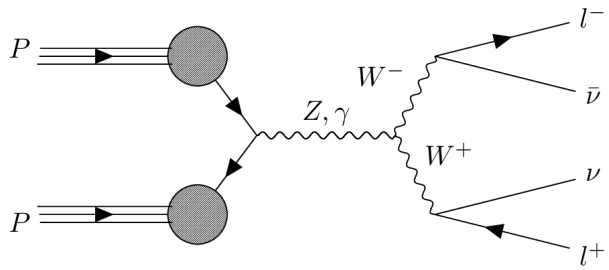
(d)



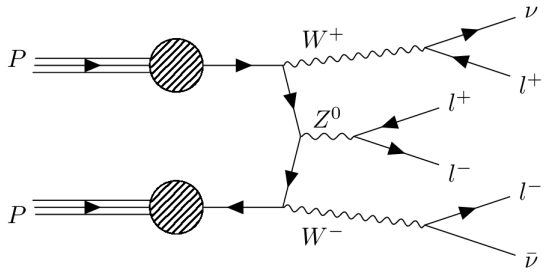
(e)



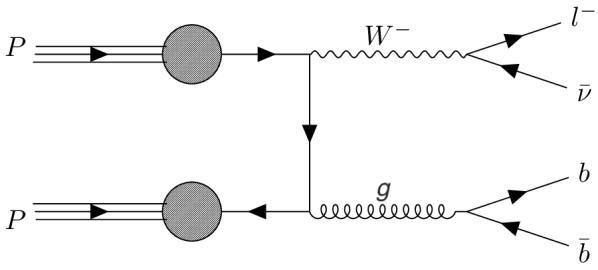
(f)



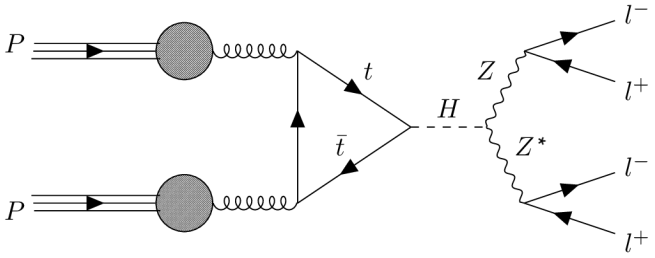
(g)



(h)



(i)



(j)

Figure 1.4: A collection of examples of Feynman diagrams for the SM background processes. The diagrams display an example of the processes $Z - jets$ (1.4a), $Diboson(lll)$ (1.4b), $t\bar{t}$ (1.4c), $Diboson(llll)$ (1.4d), $TopOthers$ (1.4e), $SingleTop$ (1.4f), $Diboson(ll)$ (1.4g), $Triboson$ 1.4h, $W - jets$ (1.4i) and $Higgs$ (1.4j).

Chapter 2

Introduction to Machine Learning and Data Analysis

Within this chapter, I will introduce the basics of data analysis and ML, as well as describe the algorithms underlying the set of models I tested during my analysis. Additionally, I will in the final part of the chapter describe where ML fits into a particle physics analysis and how I intend to assess the performance of the analysis.

2.1 Phenomenology

ML differs from other analysis tools in their ability to learn. Where a purely analytical model is static in both method and performance, a ML model aims to be dynamic and self improving. ML utilizes data to leverage towards an optimal model. The extent of utilization of data defines a ML model as being either *supervised*, *semi-supervised* or *unsupervised*. In the case of supervised ML, a set of *targets* are provided along with the data which allows a ML model to learn how to map a set of inputs to a target. In this thesis I will use the notation of $X = [\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}]$ to refer to a data set, $T = [t_0, t_1, t_2, ..., t_N]$ to the corresponding target and $Y = [y_0, y_1, y_2, ..., y_N]$ to the output of the model where N is a number of data points. The data sets contain a set of vectors with length equal to the number of features. The target and output could similarly contain vectors, but will in this thesis be restricted to scalar values. Generally, the target values can be both continuous values, in which case the ML method aims to perform a *regression*, or discrete values, in which case the ML method aims to perform a *classification*.
The goal of supervised learning is to apply knowledge obtained from a set of inputs and targets to predict the target of a new data set. The success of any prediction is dependent on the quality of the data used during training, or *training-data*. The training-data is required to be both representative of the new data set (test-data) and be of sufficient size to adequately train the algorithm. The latter point stems from a phenomenon known as *over-fitting*, which is a problem where the training data becomes overly specialized to only predict the target of the training-data, and nothing else[11]. In this thesis the main focus will be on the application of supervised learning.
In the case of unsupervised ML, no target is provided. The motivation for unsupervised learning is to create a model which is independent of any target and instead provide some metric which the algorithm can minimize during training. Such models are often useful in the case where one is not certain what one is looking for. An independence of target means that the model is not overly sensitive to any specific trends or patterns, we call this being *unbiased*. Instead of learning to detect or predict specific phenomena, unsupervised learning is often used to detect anomalies in the data, or find clusters.
Semi-unsupervised learning is a loose term which finds itself in the middle of the previous two. There is no clear definition, but the term is often used in the case where one strides away from traditional supervised learning to minimize bias. The goal is to alleviate as much bias as possible, but at the same time converge towards the usually superior performance of supervised learning.

[11] More on this in later sections.

2.2 Optimization

For a general function $\mathcal{C}$ dependent on a set of parameters $\boldsymbol{\theta} = \{\theta_0, \theta_1, ..., \theta_{N_\theta}\}$, the goal of optimization is to find optimal parameters as defined by a predicated goal. In the case of ML, the parameters are what define the model and the function is some metric which defines the error of the prediction. This means that in our case, the optimization problem corresponds to finding the set of parameters corresponding to the minimum value of $\mathcal{C}$. Several methods can be applied to optimization problems, all with their own advantages and disadvantages. In most methods the use of the gradient of the function, $\boldsymbol{\nabla}_\theta(\mathcal{C})$ is involved in one way or another. Many of the methods used in this analysis are based on one of the simplest optimization methods, the *gradient descent*-method.

2.2.1 Cost functions

How we define the performance of a ML model is not only important when evaluating the model, but is crucial during training. In the case of classification, it is natural to assume an appropriate metric should involve a comparison between the predicted classification and the true classification. The variation of performance metrics stems from the diversity of how one quantifies the comparison between the two. During training, we define an objective function used to guide the model towards optimal tuning. We call this function the *cost function*.

Binary Crossentropy

The *binary crossentropy* function is a very popular cost function for classification problems, as it can be used to compare the predicted probability distributions to the true probability distributions for a set of classification. It is ideal, as it heavily punishes high probabilities (high output) for wrong classification and heavily rewards high probabilities for correct classifications. For a set of predictions Y and a set of targerts T, the binary crossentropy function is defined as

$$\mathcal{C}\left(Y, T\right) = -\sum_{i=1}^{N} \left[\mathbf{y}_i \log\left(\mathbf{t}_i\right) + \left(1 - \mathbf{y}_i\right) \log\left(1 - \mathbf{t}_i\right)\right]. \tag{2.1}$$

2.2.2 Stochastic Gradient Descent and Mini-Batches

The gradient descent method aims to obtain the optimal parameters of a model through the application of the derivative of the cost-function, $\mathcal{C}$ with respect to the parameters in the model, $\boldsymbol{\theta}$. When evaluated at a given point in the parameter space, the negative of the gradient, $-\boldsymbol{\nabla}_\theta(\mathcal{C})$ is used to move closer to the optimal set of parameters. The negative of the gradient corresponds to the direction for which a small change $d\boldsymbol{\theta}$ in the parameter space will result in the biggest decrease in the cost function. Finding the minimum value is an iterative process, meaning the steps in the direction of $-\boldsymbol{\nabla}_\theta(\mathcal{C})$ is finite. The size of the step is a hyperparameter decided by the user and is called the learning rate, η. The evolution from a step i to $i+1$ becomes

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \nabla_\theta \mathcal{C}\left(\boldsymbol{\theta}_i\right). \tag{2.2}$$

Choosing the η can drastically affect the performance of the gradient descent method. Having a too large η and one risks 'jumping' over the true minimum or simply never allowing for parameters to reach a high accuracy. Too small η and may result in spending computation time beyond reason as well as being sensitive to local minima.
For a given data set of size N, we find the gradient $\boldsymbol{\nabla}_\theta(\mathcal{C})$ through the sum of the gradient for each data point $\mathbf{x}_i$

$$\boldsymbol{\nabla}_\theta(\mathcal{C}) = \sum_{i=0}^{N} \boldsymbol{\nabla}_\theta(\mathcal{C}_i(\mathbf{x}_i)). \tag{2.3}$$

Note, that in the equation above, each data point, $\mathbf{x}_i$ is given an equal weighting in the sum. This does not necessarily have to be the case. In the scenario that the points in the data set should be prioritized differently during training, a simple weight w_i can be multiplied in the sum of equation 2.3. We call these weights, *sample-weights*.
The calculation of the gradient over numerous data points can be time-consuming. An alternative approach

which both reduces time and introduces randomness, is the application of *mini-batches*. Instead of summing the gradient over the full data set, one randomly samples a predetermined number of points, B, creating a subset of the data which is used to update θ. The points sampled to create the subset, can be sampled both with or without replacement[12], meaning allowing the same data points to be sampled several times or not. An epoch is completed after N/B iterations of updating θ. The added stochastic element reduces the risk of getting stuck in local minima. Gradient descent with randomly sampled mini-batches is called Stochastic Gradient Descent (SGD).

2.2.3 Memory, Adaptive Learning and ADAM

Although SGD can be highly effective, it tends to be prone to oscillations between to points in the parameter space. In other words jumping back and forth between the same to sets of parameters, without ever converging closer to the minima. To alleviate this issue, we introduce momentum. The momentum aims to act as a form of memory, conserving some momentum from the previous update. By defining the new step between θ_{i+1} and θ_i as

$$\mathbf{v}_i = \eta \cdot \nabla_\theta \mathcal{C}\left(\boldsymbol{\theta}_i\right), \tag{2.4}$$

we can implement momentum to the algorithm as

$$\mathbf{v}_i = \mathbf{v}_{i-1}\gamma + \eta \cdot \nabla_\theta \mathcal{C}\left(\boldsymbol{\theta}_i\right), \tag{2.5}$$

making our new parameter iteration equal to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mathbf{v}_i, \tag{2.6}$$

where $\gamma \in [0-1]$ is a parameter which defines the size of the momentum.
An additional flaw in the SGD optimization algorithm, is that it treats all regions in the feature space equally. Ideally, we would want the algorithm to focus on the most relevant areas (steep regions), while quickly moving past less relevant areas (flat regions). A solution to this is the introduction of an *adaptive learning rate*. By updating the learning rate based on the surface of the feature space, one can assign a prioritization in the feature space which optimizes the search for optimal parameters.
The most intuitive way to prioritize steep areas, is by introducing the second derivative of $\mathcal{C}$, but this can be quite time-consuming. The Adaptive Moment Estimation (ADAM) optimizer aims to create an approximation of the second derivative, through the use of momentum and an adaptive learning rate. The ADAM optimizer introduces a set of new parameters, and includes both the linear and squared term of the derivative of $\mathcal{C}$ (i.e. $\nabla_\theta \mathcal{C}$ and $(\nabla_\theta \mathcal{C})^2$) with memory parameters for both (β_1 and β_1 respectively). Additionally, the ADAM optimizer holds a memory of the average of the gradient and squared gradient, which is used to create faster convergence. For a thorough description of the ADAM, the reader is referred to the paper by Kingma et al. [21].

2.3 Hyperparameters

The tuning of parameters is a vital part of building an optimal ML model, though not all parameters are set during training. Parameters that are chosen prior to and fixed during are called *hyperparameters*. We differentiate between two types of hyperparameters; model hyperparameters and algorithm hyperparameters. Model hyperparameters refer to parameters used to define the architecture of the model. Examples of this could be the size and depth of a NN or the maximum depth of a Decision Trees (DT). These parameters are not tuned during training, but will nonetheless have a great impact on the performance of the model. Algorithm hyperparameters on the other hand, do not have a direct impact on the performance of the model. These are parameters that mainly affect the effectiveness and quality of the training process. Examples of this are the learning rate of a NN, or the batch-size used during training. Regardless of whether we are discussing model- or algorithm hyperparameters, it is in our interest to find the optimal choice of parameters. The choice of parameters is made prior to the final training, and will be discussed in the following section.

2.4 Data Handling

As previously mentioned, the data used in ML is as, if not more, important than the model itself. As a consequence, there are several steps taken to ensure that the data is in a good state before we apply the ML

[12] In this analysis the data is sampled with replacement.

models to it. In this section I will discuss some of these aforementioned steps, both what they do and what each step hopes to achive.

2.4.1 Feature Scaling

The range of values for the different data features can vary immensely, and for most cost functions, this is a problem. For some features, a deviation on the magnitude of 10^3 can be a good approximation, whereas for others it can be a vast overestimation. When a model is to define which direction it wants to tune, it is crucial that the errors across all features are weighted equally. Scaling aims to alleviate this issue by transforming all features to have a relatively equal range of values while simultaneously preserving all information regarding each feature. Exactly how one chooses to scale the data heavily affects the performance of the model and is regarded as a hyperparameter of the model.

Standard Scaler

In this analysis, I decided to use the highly popular scaling method, the *standard scaler*. The standard scaler implemented in this study uses Scikit-learns's `StandardScaler` [22]. The standard scaler function scales each feature individually by subtracting the mean and dividing by the standard deviation. In doing so the resulting scaled data has a mean of 0 and a standard deviation of 1. Mathematically the standard scaler, $\mathcal{S}$, transforms a data set, X, as

$$\mathcal{S}(X) = \frac{X - \boldsymbol{\mu}_x}{\boldsymbol{\sigma}_x}, \tag{2.7}$$

where $\boldsymbol{\mu}_x$ and $\boldsymbol{\sigma}_x$ are vectors with the elements being the mean and standard deviation for each feature, respectively.

2.4.2 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique used in many analyses. The goal of PCA is to reduce the dimensions of a high-dimensional data set while at the same time conserving as much of the variance (or value spread) in the data as possible. The motivation behind dimensionality reduction is (mainly) rooted in two points. The first is noise reduction. Some features could not only be non-contributing during training, but could even introduce noise. The second reason is lack of convergence. In a large data set with many features and different classifications (in our case channels), a ML could struggle to identify the most important trends. By reducing the dimensionality in the data, the hope is that this would be easier.
In short, a PCA finds the direction in the feature space along which the data has the largest variance (called principal components), using a linear combination of the original features, and projects the data upon it, creating a new data set. The principal components are ordered such that the first captures the most variation, the second captures the most variation orthogonal to the first and so on. Before applying the PCA, it is essential to scale the data, to ensure a realistic representation of the variance in each feature. We can summarize the algorithm of the PCA in the following 6 steps:

1. Center the data around 0 by subtracting the mean from each feature.
2. Calculate the covariance matrix to find the covariance of each feature pair.
3. Calculate the eigenvalue and eigenvectors of the covariance matrix.
4. Order the eigenvector by size of the eigenvalues to define the directions in the feature space with the largest variance.
5. Cast the data along these directions to form a new data set with the new features ranked from largest to lowest variance.
6. Remove the features with the least variance according to some threshold defined by the user.

The threshold mentioned in the final point, is decided by the user, and defines how much of the variance from the original data set should be included. For example if the threshold is set to 70%, X number of the last features will be removed such that at least 70% of the variance is preserved.
In figures 2.1 and 2.2 I have plotted the distribution of 10, 100 and 300 samples for the features with the most and second most variance (2.1), and least and second least variance (2.2) after applying PCA to our

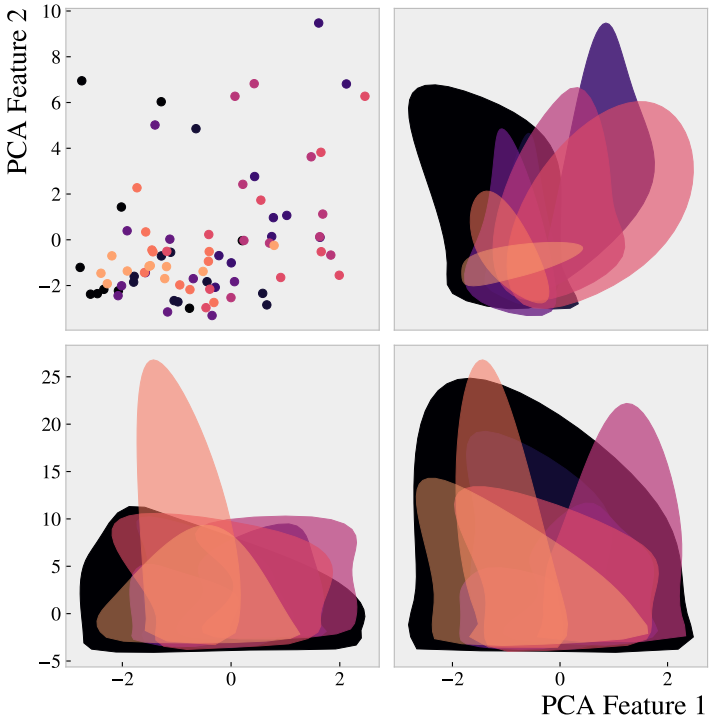


Figure 2.1: The value distribution of the two PCA-features containing most variation for (left to right, up to down) 10, 10, 100 and 300 samples from each channel (visualized with different colors). Each sample filling the requirement with being less than one standard deviation from the mean of both features, respectively.

data set. In this PCA I am yet to remove features, so all four features are taken from a data set where all the variance is still conserved. Each color in the figures represent different categorize of collisions, i.e. $t\bar{t}$ or diboson. With this in mind, the different scales on the y- and x-axis (10^0 for figure 2.1 and 10^{-8} for figure 2.2) exhibit how the PCA creates new features where there is a vast difference in variance (and therefore value to the analysis), and justifies how one can remove features while preserving most of the variance. In figure 2.2 we observe that all channels exhibit the same trends and are practically identical. In other words, these features would not contribute in training a model to distinguish the different channels. In comparison, figure 2.1 displays large variance and individual trends among the different processes.

2.5 Regularization

In ML, overfitting occurs when a model becomes overly tuned to random variation in the training data and as a consequence fails to extract trends which would allow it to predict previously unseen data. The architecture of a NN, the maximum depth of a BDT or even the size of the data set can all contribute to overfitting. In the case of deep learning especially, overfitting can be a large problem and is therefore a focus in this thesis. Apart from predicting on a new data set, there are no rigid methods to detect overfitting. Instead, there exist a variety of methods to minimize the risk of over-fitting which are collectively known as *regularization*. In ML, regularization is known as any attempt to reduce the error in a prediction by reducing overfitting. Generally one can categorize regularization as being either implicit or explicit. Explicit regularization means adding terms to the optimization problem. This is a very direct way of ensuring no part of the model becomes overly dominant. Examples of this could be adding a penalty in the cost function of a NN to ensuring no weights become too large. Implicit regularization is a less direct attempt of hindering overfitting. This could be changing the depth of the ML model, varying the cost function or altering the data itself.

2.5.1 Early Stopping

A simple implicit regularization method is to introduce *early stopping* in the training pipeline. Early stopping simply means to stop training before the parameters of the model are allowed to over tune. The usual approach is to introduce a goal for the training, which when reached, ends the training. Examples of these

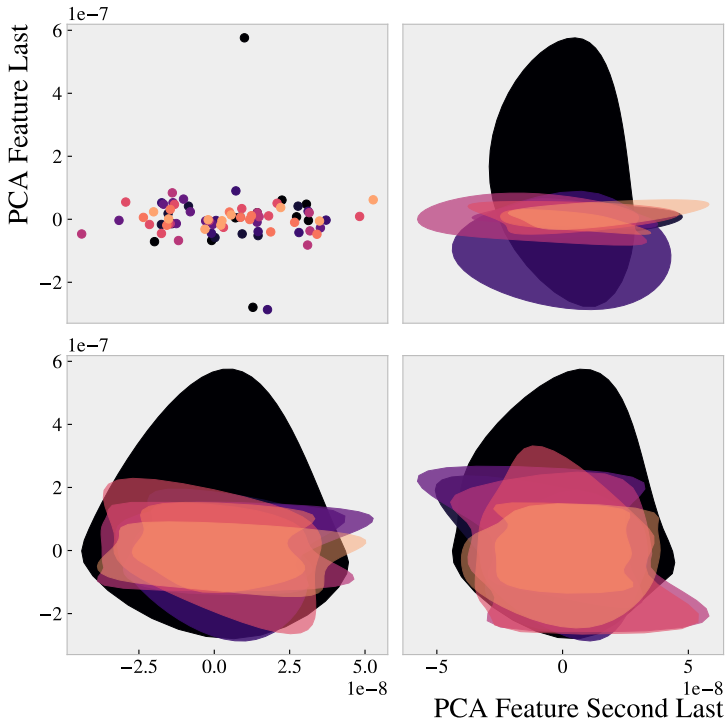


Figure 2.2: The distribution of the two PCA-features containing the least amount of variation for (left to right, up to down) 10, 10, 100 and 300 samples from each channel (visualized with different colors). Each sample filling the requirement with being less than one standard deviation from the mean of both features, respectively.

goals could be a predetermined loss value on the training set or training until lack of progress on a second unseen data set. The latter approach is the one I will use in this thesis.

2.5.2 Ensembles

When comparing different ML methods by performance, more often then not the top performing model includes ensembling in one way or another. Like the word suggests, ensembling in ML means using a collection of ML models to create one complex model. The key point to ensembling is that an ensemble of weak learners can together be a strong learner. There are many ways to create ensembles of models, most methods fall in to one of three categories; *bagging*, *boosting* and *stacking*. Creating an ensemble of models through bagging, means to use several models each trained on their own sample from the same data set. The overarching new model is created by averaging the predictions from the ensemble of models. The method seeks to create a unique set of models through exposing each individual model to different training sets. Boosting is different to bagging in that it uses the same training data on all the models. The diversity in the models when boosting, stems from intentionally choosing the architecture of the models such that it reduces the error made by the previous ensemble of models. Finally, stacking uses a predetermined model to decide how to combine the predictions made by the ensemble.

2.6 Neural Networks

The concept of a NN has been around for more than 80 years, and today they are one of the most popular and successful ML methods. The key to its popularity stems from its versatility, achieving high performance in a large range of both regression and classification problems. One of the defining qualities in a NN is the possibility of diverse architecture, meaning that there are many categories of NN, where each category has an even deeper selection of networks. Categories ranging from Convolutional Neural Network (CNN), Recursive Neural Network (RNN) to simple Feed-Forward Neural Network (FFNN), where each category is specified for their own set of problems. In this section I will introduce some fundamental definitions in regard to NN, and go through the underlying algorithm of the back- and forward propagation.

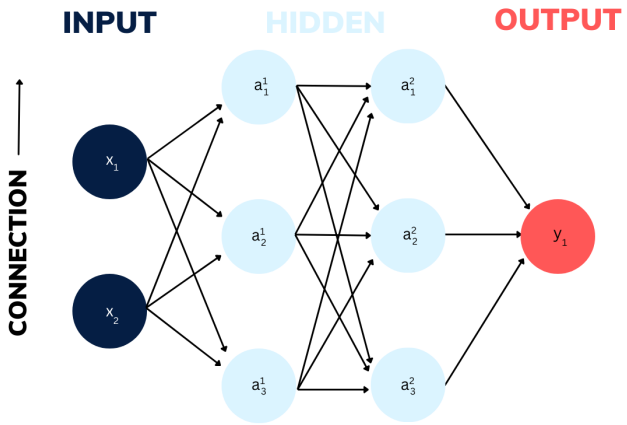


Figure 2.3: An illustration of the architecture of a NN with two hidden layers.

2.6.1 General Structure

There are often drawn comparisons between the structure of the neural network, and the way the human mind operates, hence neural. Similarly to the human mind, a NN is composed of different neurons, or nodes communicating information backwards and forwards in different regions. In the case of a neural network we call these regions layers. All layers are composed of a specified number of nodes. A NN has three types of layers; *input layer*, *hidden layer* and *output layer*. There is only one input layer, and it has the same number of nodes equal to the number of features for each data point. There can be an arbitrary number of hidden layers, with each hidden layer containing an arbitrary number of nodes. Finally, the NN has an output layer. The output layer contains a number of nodes equal to the number of features for the target.
The neural network functions by passing information in between the different layers through nodes. The nodes are simply pockets of information, each containing a value. All the nodes in the input layer are (in most cases) connected to all nodes in the nearest hidden layer, and likewise said hidden layer is connected to the next hidden layer. This structure continues until we reach the final layer, the output layer. The structure is illustrated in figure 2.3. The figure shows a simple NN with a two-dimensional data set (two nodes in input layer), two hidden layers with three nodes each and a one-dimensional target value. It also illustrates how a NN aims to map from data to a prediction. In figure 2.3 we can see how all the different nodes are connected, illustrated by the arrows. The passing of values between different nodes are controlled by a set of weights and bias parameters. These parameters are defined for each connection and are what will be tuned during training. The weights and biases for a given connection of two nodes, defines the effect one node has on the other.
In a traditional FFNN the information is passed linearly (in figure 2.3, from left to right) in a process we call *forward propagation*. Other variants can include the information taking a more complex route. It is often the route from input- to output layer that categorizes the type of NN. In this report I used a set of simple FFNN's.

2.6.2 Feeding Forward

With the structure described in the previous section, a trained model, $\mathcal{F}$, produces a prediction, Y, for a data set, X, by passing information from input layer, through all hidden layers, to the output layer, which we call forward-propagation. In this section I aim to explain the underlying algorithm and math used by the NN to map input to output.
We imagine the passing from hidden layer $l-1$ to l, where $l \in \{2, ..., L\}$[13] and L is equal to the number of hidden layers. The activated value of a node in layer l, is defined as a_j^l (as indicated by figure 2.3), where $j \in \{0, 1, ..., N_l\}$ and N_l is equal to the number of nodes in l. The value of a_j^l is defined as the *activated* sum of all nodes in the previous layer, a_j^{l-1} where the sum is weighted by a parameter $\mathbf{w}_\mathrm{j}^\mathrm{l}$ and shifted by the bias, b_j^l for $j \in \{0, 1, ..., N_{l-1}\}$. The activated value of a node j in layer l is defined as

$$z_j^l = \sum_{k=1}^{N_{l-1}} w_{kj}^l a_k^{l-1} + b_j^l, \tag{2.8}$$

[13]There is a special case for when $l = 1$ which will be addressed in the next paragraphs.

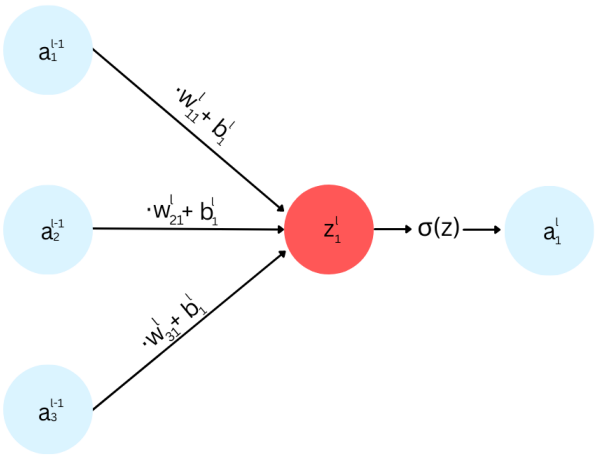


Figure 2.4: An illustration of a forward propagation from one layer to a node in the next.

where w^l_{kj} corresponds to the weight in $\mathbf{w}^l_k$ specific for the connection between node k and j.
To attain the full activated value, a^l_j we pass z^l_j through the *activation function*. The activation function, σ^l is a generally non-linear function used to control the limit or expand the value range for the node values. The activation function is general for all nodes in a given layer, but can vary in between layers. Therefore, we find a^l_j by the equation

$$a^l_j = \sigma\left(\sum_{k=1}^{N_{l-1}} w^l_{kj} a^{l-1}_k + b^l_j\right) = \sigma^l(z^l_j). \tag{2.9}$$

A more detailed illustration of the information passed from one layer to a node in the next layer is displayed in figure 2.4. In figure 2.4, we see all steps described in the process; *(1)* all nodes in l-1 are summed with a corresponding weight, *(2)* the sum is shifted by a constant term (bias), *(3)* the scaled and weighted sum defines the inactivated value z^l_j, *(4)* we define a^l_j by passing the sum through σ^l. This method is used to pass information between all layers, except between the first and the second. In this case we simply replace the activated term, a^{l-1}_k in equation 2.9, by the input data, x_i for $i \in \{0, 1, ..., N\}$, where N is equal to the number of features for the data. In the case $l = 1$, 2.9 becomes

$$a^1_j = \sigma\left(\sum_{k=1}^{N} w^1_{kj} x_k + b^1_j\right) = \sigma(z^1_j). \tag{2.10}$$

And in the case where $l = L$, a^L_j is equal to the final output.

2.6.3 Back Propagation

The backward propagation acts as the engine that drives the training of a neural network. It has the purpose of tuning the weights and biases of the network, which are originally given random values, to achieve maximum performance, as defined by some *cost function*, $\mathcal{C}$. The cost function defines to which metric we aim to optimize the network. When a neural network produces a prediction, Y, the error in the prediction is defined by the cost function as $\mathcal{C}(Y, T)$, where T is the target. To minimize $\mathcal{C}$, the backward propagation utilizes the gradient with respect to the weights and biases in the network, as described in the section regarding optimization 2.2. Instead of a direct calculation of the gradient, which is very computationally heavy, the backward propagation aims to calculate the gradient through a recursive algorithm which traces the error backwards through the network. It is this algorithm I will describe in this section.
When minimizing the error defined by $\mathcal{C}$, we can apply several optimization algorithms described in section 2.2. Common for the algorithms is the use of the gradient of the cost function with regard to the tunable parameters. In our case these parameters are the weights and biases, $w^l_{i,j}$ and b^l_j. The goal of the backward propagation is therefore to calculate the gradients $\partial\mathcal{C}/\partial w^l_{i,j}$ and $\partial\mathcal{C}/\partial b^l_j$.
To begin with we can derive an expression for $\partial\mathcal{C}/\partial w^l_{i,j}$. We use the chain-rule to define

$$\frac{\partial\mathcal{C}}{\partial w^l_{i,j}} = \frac{\partial\mathcal{C}}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{i,j}},$$

which we can simplify further by using equation 2.8 to calculate the second term, which becomes

$$\frac{\partial \mathcal{C}}{\partial w_{i,j}^l} = \frac{\partial \mathcal{C}}{\partial z_j^l} a_i^{l-1}.$$

We can redefine the first term in the equation above as δ_j^l and write

$$\delta_j^l \equiv \frac{\partial \mathcal{C}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{C}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l},$$

where we have again used the chain rule for all contributing nodes. To calculate the final partial derivative we write

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial z_k^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = w_{jk}^{l+1} (\sigma^l(z_j^l))',$$

where we have again used equation 2.8. This gives the expression for δ_j^l

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{jk}^{l+1} (\sigma^l(z_j^l))' \tag{2.11}$$

Finally this gives us the expression

$$\frac{\partial \mathcal{C}}{\partial w_{i,j}^l} = \delta_j^l a_j^{l-1}. \tag{2.12}$$

Next we want to derive $\partial \mathcal{C} / \partial b_k^l$. We simply use the chain rule and derive

$$\frac{\partial \mathcal{C}}{\partial b_j^l} = \frac{\partial \mathcal{C}}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l},$$

which from equations 2.11 and 2.12 is simply

$$\frac{\partial \mathcal{C}}{\partial b_j^l} = \delta_k^l \cdot 1 = \delta_j^l, \tag{2.13}$$

From all three derived expressions, equations 2.11, 2.12 and 2.13 we see that to calculate for all $l \in \{0, 1, .., N_l\}$ we must first calculate δ_j^L and apply a recursive propagation. To calculate $\delta_j^L = \partial \mathcal{C} / \partial z_j^L$ we again apply the chain rule with the assumption of the activated node in the cost-function, and we find

$$\delta_j^L = \frac{\partial \mathcal{C}}{\partial a_j^L} \left(\sigma^L(z_j^L)\right)'. \tag{2.14}$$

This expression, similarly to equation 2.11, is dependent on the choice of $\mathcal{C}$ and the activation functions. Now that equation 2.14 is defined, we can see that the gradient of the parameters in all other layers can be calculated. A full *epoch* is defined as a forward propagation which creates a prediction, followed by a backward propagation which tunes the parameters in an attempt to correct the errors in the prediction.

2.6.4 Activation Functions

As mentioned in previous sections, activation functions define how the nodes in the previous layer sum to define the value of the node in the current. There are many types of activation functions, where all have advantages and disadvantages. The choice of activation function for each layer is defined before training, making it a hyperparameter. The activation functions applied and tested in this thesis are the following:

- *Sigmoid*

$$\sigma(z) = \frac{1}{1 + e^{-z}} = a \in [0, 1]$$

- *LeakyReLU*

$$\sigma(z) = \left(\begin{cases} z, & \text{if } z \geq 0 \\ \alpha z, & \text{otherwise} \end{cases} \right) = a \in (-\infty, \infty),$$

where α is scalar.

,

where z is an inactivated node which is activated to define a, the activation.

2.6.5 Network Ensembles, Dropout and LWTA Networks

So far in the thesis, I have only covered dense layers, meaning that every node in a layer is connected to the nodes in the previous and following layer. This definition covers many, but not all hidden layers. Some layers do not function to pass values back and forth between nodes but instead functions by dynamically changing the architecture of the NN. In this analysis, I have implemented both the popular *dropout*-layer, and also a group of layers fitting the definition introduced in the paper [23], as Local-Winner-Takes-All (LWTA).

Dropout

The dropout layer functions by assigning a predetermined probability to each neuron in a given layer. Based on the probability, a number of neurons (dependent on the probability) is dropped in each forward pass. By doing this the dropout layer creates an architecture for the network for every round of training. Each unique architecture represents its own network in what becomes an ensemble of networks.
As mentioned in previous sections, creating ensembles of models is a form of regularization. In the case of dropout, it minimizes the risk of overfitting by hindering a phenomenon known as complex co-adaptation. Complex co-adaptation happens when a neuron becomes overly dominant such that neighboring neurons no longer contribute (relative to the dominant neuron) and therefore lack the motivation to tune. When this happens, networks become fragile and overly specialized to the training data. By randomly dropping neurons, the neighboring neurons are no longer allowed to be passive and are forced to tune to compensate. During evaluation, the dropout layers no longer take action. Instead, the dropout rate (frequency of drop) r is used to scale the weights by a factor $1 - r$. The prediction made by the resulting model can therefore be seen as the average of all the smaller networks. In other words, a neural network containing dropout layers can be viewed as a bagging ensemble (see subsection 2.5.2).

Channel-Out

Dropout layers are not the only layers to dynamically change the architecture of a network. In this thesis I additionally explored the lesser known, *Channel-Out*-layer as introduced in the paper by Wang et al. [9]. Channel-out, similarly to dropout creates an ensemble of networks by removing a set of nodes for each round of training. Contrary to dropout, channel-out does not choose the nodes to drop at random, but instead creates local *units* by grouping the nodes, and removes all nodes except the node with the largest activation in each respective unit. The 'removed nodes' are multiplied by 0, as to remove any contributions to the next layer. The goal of channel-out is to create an ensemble of networks through *trend specific paths*, where each network is specialized for a given trend in the data. This would create an ensemble of networks, which not only applies a form of regularization, but also improves a phenomenon known as *long-term memory* in the model. The phenomenon of long-term memory is discussed in the article by Srivastava et al. [23], and is simply a measure of how well a model is able to retain information on a data set, A, after training on an additional training set, B. This effect is also relevant for the two layers I will discuss in the next sections.
In this thesis I have implemented the layer such that upon prediction, the redimensionalization is still active. This means that for each data point, a neural network is chosen based on the specific activations in each of the nodes. In other words, the channel-out layer creates something resembling a stacking ensemble (see subsection 2.5.2).

Stochastic-Channel-Out

As I will describe in further detail in sections to come, I choose to implement channel-out layer myself. A consequence of this was that I was able to experiment with it. In section 2.6.5, I described the issue of complex co-adaptation. I also described a possible solution to the issue, the dropout layer. Although dropout and channel-out function rather similarly, they do not deal with this issue similarly. Where the dropout

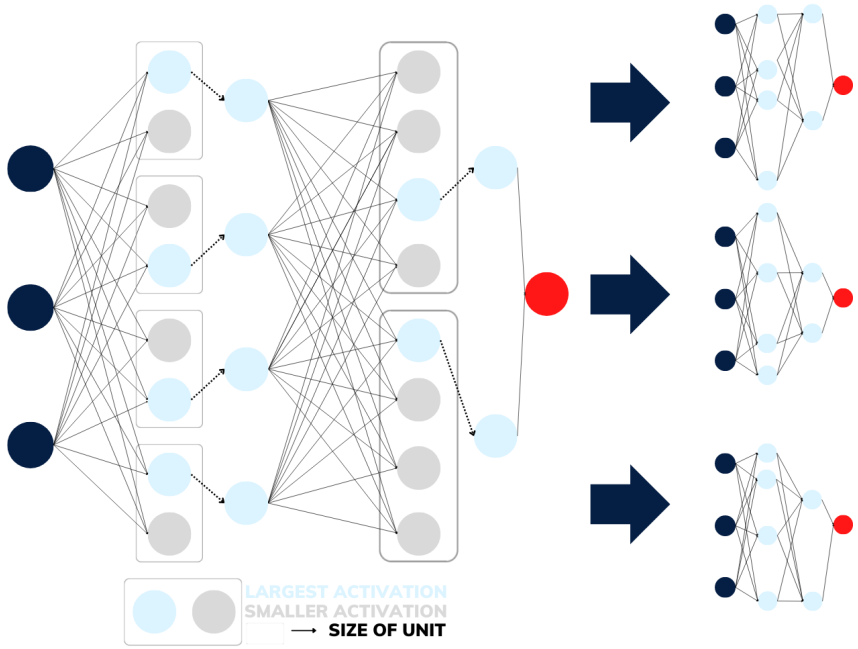


Figure 2.5: An illustration of a Neural network with two hidden layers, each with 8 neurons. The first and second hidden layers have a maxout activation layer with four and two units respectively. The figure also illustrates the resulting ensemble of smaller neural networks as a consequence of the maxout activation layers.

layer will force dormant nodes to contribute in the training by allowing for dominant nodes to be dropped, the channel-out would instead allow for dominant nodes to take further control inside their respective unit. To remedy the issue of complex co-adaptation inside the units, I proposed a new layer, the Stochastic-Channel-Out (SCO). SCO functions similarly to channel-out, with the exception of the units. Where the channel-out layer utilizes static local units[14] throughout training, the SCO instead utilizes dynamic units which change for each data point. In other words, for each data point, each node would potentially have a new group of nodes to compare with when finding the largest activation. The goal of the SCO is to capitalize on the randomness of the dropout layer and the trend specific paths' aspect from the channel-out layer.
In the same manner as for channel-out, the redimensionalization is active during prediction and training for the SCO. This means that also the SCO creates an ensemble resembling a bagging ensemble.

Maxout

Additionally to the channel-out I will be applying a second layer discussed in the paper by Wang et al. [9], the *Maxout* layer. The maxout layer functions very similar to the channel-out with a small difference in the dropping of nodes. In the dropout layer, nodes are dropped by neglecting all contributions from said node, or in other words by multiplying the activation from the dropped nodes by zero. This is replicated in both the channel-out layer and the SCO layer. In doing so all nodes, activated and dropped alike, possesses their own set of weights and biases. This is not the case for the maxout layer. In the maxout layer, each unit possesses just one set of weights and biases. For each forward pass in the network, the largest activated node will contribute to the nodes in the following layer using the same weights and biases as the rest of the nodes in the units. However, the nodes only share parameters connected to the layer in front, not behind. This allows for the maxout layer to have fewer parameters to tune, while at the same time building trend specific pathways similar to channel-out and SCO.
In figure 2.5 I have illustrated a maxout network. The figure shows a NN with two hidden layers with eight nodes each. In the first hidden layer the maxout layer creates four units, each containing two nodes, and in the second layer the maxout creates two units with four nodes each. The resulting output from the first layer is the four nodes with the highest activation in their respected unit, likewise the second layer's output is two nodes. To the right of the network in figure 2.5, I have illustrated how the different configuration of paths through the network creates an ensemble of networks with their own architecture. Note in the figure how, as described in the previous paragraph, each node possesses its own connections to the previous layer,

[14] I.e. the nodes are placed in local units in the beginning of training and held constant throughout.

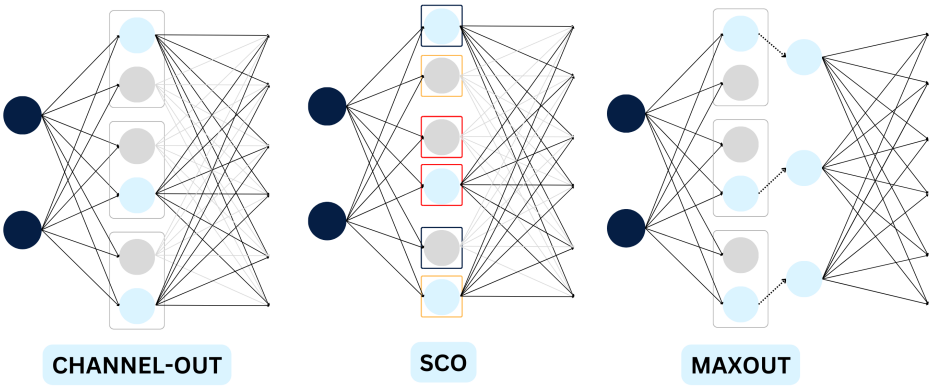


Figure 2.6: An illustration of three different layers, channelout, SCO and maxout. The figure shows how each layer redimensionalize the network and how channelout and SCO differ from maxout in regard to the relationship between units and parameters. The gray lines and nodes represent dropped nodes, whereas the blue nodes and dark lines represent the nodes which are allowed to contribute. In the case of SCO the different colors surrounding the nodes represents different units, which visualizes how any set of nodes could define a unit.

while sharing the connections with the rest of the unit, to the next. In figure 2.6 I have drawn a separate illustration to show how this differs from both channel-out and SCO in this regard. Figure 2.6 illustrates how all three layers redimensionalize the network, how channelout and SCO differ from maxout in regard to the relationship between units and parameters and finally how channel-out differs from SCO in choice of units[15]

2.6.6 Parametrized Neural Network

In this thesis I will be studying BSM-simulations with a diversity in choice of free parameters. This means that my signal is of itself diverse. The free parameters studied in this thesis are the mass of new particles. The mass of a hypothetical particle in a given BSM theory can greatly alter the feature space spanned by processes including said particle. This means that a ML-model could potentially struggle to tune according to all trends in the signal. The obvious solution to this problem is simply to implement one model per mass combination, or choice of mass, and simply reusing the background. Although this approach is very popular, I have decided not to use it for (mainly) three reasons:

- This approach has been carefully studied for many years, and leaves little room to further exploration.
- Some signals overlap in the feature space. This means that by neglecting signals with relative similar mass, you could be neglecting statistics which could help tune to your original signal.
- By including two signals with relatively similar masses in the training, some[16] suggest that the model would be able to interpolate between the masses and cover a larger search area.

When the data set, X, is dependent on a set of free parameters, $\theta = [\theta_0, \theta_1, \theta_2, ..., \theta_N]$, we can write $X(\theta)$. For a single neural network trained on a set of parameters, we define the model as $\mathcal{F}(X(\theta))$. In the case where we build one model for each choice of parameter, we get a set of models defined as $\{\mathcal{F}_a(X(\theta_a))$, $\mathcal{F}_b(X(\theta_b))$, ...,$\mathcal{F}_c(X(\theta_c))\}$, where a, b and c are indices which correspond to individual choices of parameters respectively. In the paper by Baldi et al. [8], they propose a separate solution to the problem than the one proposed above, the Parameterized Neural Network (PNN). By simply including the parameters as additional features, we can write $\mathcal{F}(X, \theta)$. In doing so, we can utilize all the statistics at the same time, while also aiding the NN in its effort to recognize all individual trends for all signals. The difference in approach between parameter specific networks and the PNN are highlighted in figure 2.7. In practice, it is aiding the NN by adding a shift to the total output from the initial layer according to the discrete distribution of the parameters. The hope is that each discrete shift will motivate a separate individualistic tuning for each choice of parameter. For the SM background, adding a parameter representing the mass of a BSM particle is meaningless. Therefore, the background is randomly assigned values according to the same distribution used in the signal data. In doing so, one is essentially assigning each set of signal its own portion of background data.

[15] In figure 2.6 this is highlighted by the different colors surrounding the nodes in SCO, where each color represents a unit.
[16] See the paper by Baldi et al. [8].

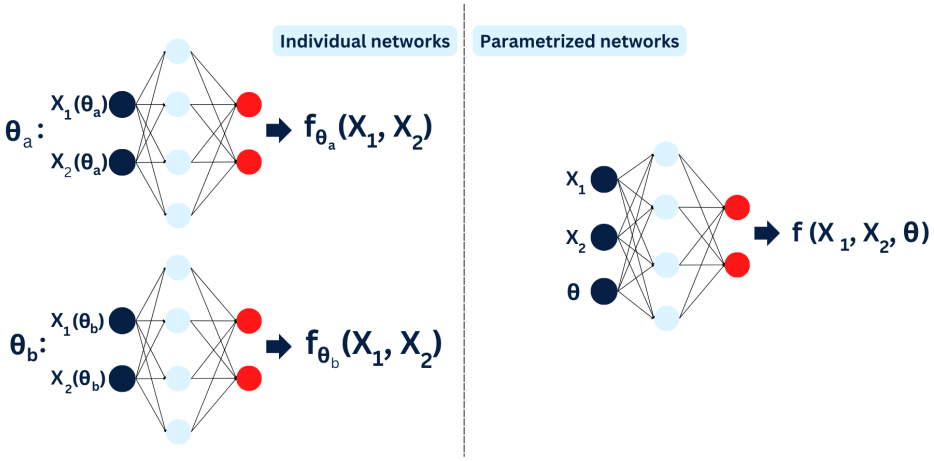


Figure 2.7: An illustration of a comparison between the parameter individualistic network approach and the PNN.

2.7 Decision Trees and Gradient Boosting

2.7.1 Decision Trees

Decision Trees (DT) are, similarly to NN some of the most popular ML methods used today. Contrary to NN, DT are famously easy in theory. Despite the simplicity of DT, they are capable of solving a large range of complex problems. In this section I will cover the use of DT as applied to a supervised classification problem.

The goal of decision trees are to create a flowchart-like tree structure from input to output. Similarly to the traditional cut-and-count (CC) method used in particle physics, a DT places *rectangular-cuts* on a data set. It does so to create a collection of thresholds $\{c_1, c_2, ..., c_{N_c}\}$, which when applied to a new data set and target, will sort each data point to the corresponding target.

In figure 2.8, I have illustrated a simple DT classifying a 4-dimensional input data to one of three classifications. As is visualized in figure 2.8, the DT applies a set of thresholds on the data to find the route applicable to a target. For each applied cut, the data split into what we call *branches*. Each branch represents a subset of the data. The final subset, after a sufficient amount of cuts ends in what we call *leafs*. Each leaf contains a label which is assigned to the subset of data of which it contains.

Just like most ML-methods there are many kinds of DT, each with their own architecture and benefits. In the case of DT the defining qualities can be summarized in its choice of *Depth* and *Optimization*. When provided with a data set, a DT can in theory create as many cuts needed to map each individual data point, $\mathbf{x}_i$, to the corresponding target, t_i. Doing so would not only be very computationally heavy, but would almost certainly lead to overfitting if applied to any new data. Instead, when building a DT one defines a maximum depth. This means that we must define a limit to the number of cuts, which subsequently leads to the need for a prioritization of cuts.

Building a DT and choosing which cuts to apply at what point, is the equivalent of training a network. How to decide by what standard one chooses to build the hierarchy of cuts is again the equivalent of choosing an optimizer. For more information on DT, the reader is referred to the book by Hastie et al. [24].

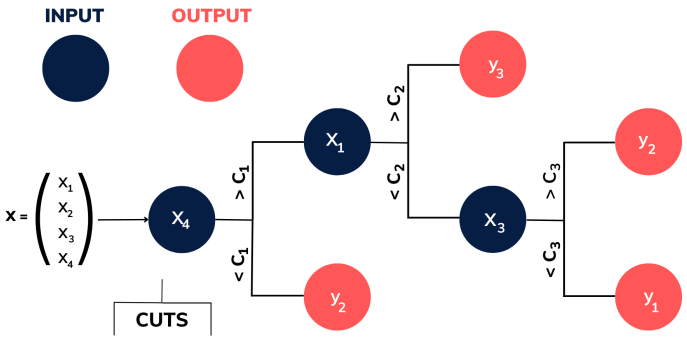


Figure 2.8: An illustration of a simple DT, mapping a four dimensional input (x_1, x_2, x_3, x_4) to one of three values in the target space (y_1, y_2, y_3), through a set of cuts $\{c_1, c_2, c_3\}$.

2.7.2 Gradient Boosting in Decision Trees

Gradient-boosting is an algorithm which uses a collective of 'weak' classifiers in order to create one strong classifier. In the case of gradient-boosted trees the weak classifiers are a collective of shallow trees, which combine to form a classifier that allows for deeper learning. As is the case for most gradient-boosting techniques, the collecting of weak classifiers is an iterative process.
We define an imperfect model $\mathcal{F}_m$, which is a collection of m number of weak classifiers, or estimators. A prediction for the model on a given data-points, $\mathbf{x}_i$, is defined as $\mathcal{F}_m(\mathbf{x}_i)$, and the target for the aforementioned data is defined as t_i. The goal of the iterative process is to minimize some cost-function $\mathcal{C}$ by introducing a new estimator h_m to compensate for any error, $\mathcal{C}(\mathcal{F}_m(\mathbf{x}_i), y_i)$. In other words we define the new estimator as:

$$\tilde{\mathcal{C}}(\mathcal{F}_m(\mathbf{x}_i), \mathbf{y}_i) = h_m(\mathbf{x}_i), \tag{2.15}$$

where we define $\tilde{\mathcal{C}}$ as some relation defined between the observed and predicted values such that when added to the initial prediction we minimize $\mathcal{C}$.
Using our new estimator h_m, we can now define a new model as

$$\mathcal{F}_{m+1}(\mathbf{x}_i) = \mathcal{F}_m + h_m(\mathbf{x}_i). \tag{2.16}$$

Similarly to how we define a depth of trees, we can define the degree of boosting. We define this as the amount of trees used in the iterative process, or M. This means that the final classifier becomes

$$\mathcal{F}_M(\mathbf{x}_i) = \sum_{i=0}^{M} h_i(\mathbf{x}_i) \tag{2.17}$$

The `XGBoost` [7] framework used in this analysis enables a (advanced) gradient-boosted algorithm, and was initially created for the Higgs ML challenge. Since the challenge, `XGBoost` has become a favorite for many in the ML community and has later won many other ML challenges. `XGBoost` often outperforms ordinary decision trees, but what it gains in results it looses in interpretability. A single tree can easily be analyzed and dissected, but when the number of trees increases this becomes harder. For a more detailed explanation of the `XGBoost` framework, the reader is referred to [7].

2.8 Machine Learning Applied to a BSM Search

So far I have presented the goal of my analysis (to discover new physics) as well as the tools I will be using to achieve it (ML). In this section I will discuss how ML is applied to the problem as well as how it compares to traditional methods.

2.8.1 The Traditional Approach

As discussed, I have been presented with two sets of data; the measured collision data from ATLAS and the simulated MC data. The origin of the latter data set will not be covered in great detail in this thesis, but it is worth mentioning that the simulations are based on SM theory. In other words, by comparing the

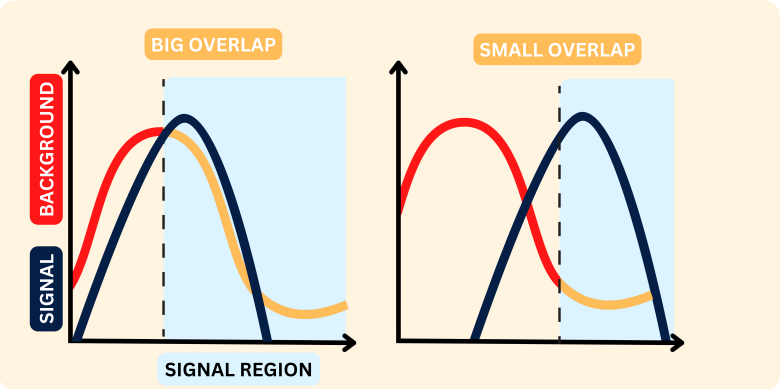


Figure 2.9: An illustration of a traditional cut-and-count (CC) approach and how non-overlapping features lead to effective signal regions. In the first figure (left) the distributions of signal and background are relatively similar, and the signal region contains large amounts of background. In the second (right), the distributions of signal and background are greatly different, and this variable would allow for a much more effective signal region.

measured collision data with the SM simulations, we are essentially comparing what is predicted by the SM to what is measured by experiment. If the two differ in ways not explained by simulation error or other non physics related errors, one could interpret the deviation as new physics.
In short, a search for new physics, is a search for deviations in the comparisons of simulations to experimental data. At first thought, this might seem like a simple task. Given that the deviations would be large, it would be easy. In reality, any new physics predicting large contributions in the data currently measured by ATLAS has been excluded a long time ago. Today, any promising extension of the SM predicts to contribute rarely in any collision. As will be presented in later sections some theories that will be searched for in this thesis only contribute to a total of 6 collisions in a data set consisting of more than 3800000 events ($< 0.002\%$). Not only would such a deviation be incredibly hard to detect, but it would be close to impossible to determine if such a deviation is rooted in new physics or simply noise.
The traditional approach to this problem is to study the data in physics motivated regions. For example, in section 1.5 I presented the Feynman diagrams of the signals I searched for in this thesis. As mentioned, this type of final state is expected to exhibit large amounts of missing transverse energy[17]. The traditional approach is to neglect all the data with small amounts of missing energy, and only consider events of interest. By applying these kinds of constraints on the data, you are creating a region where you expect to find as much of the signal and as little of the background as possible. After applying a sufficient amount of demands (or cuts), you would count the remaining data in your *search region* and check for deviation. This approach is called cut-and-count (CC).

2.8.2 The Machine Learning Approach

For a signal which greatly differs from the SM, the CC method could be sufficient. But, in cases where the signal is similar to background, it becomes harder to create effective[18] cuts. In figure 2.9, I have drawn an illustration of two different feature distributions both displaying the distribution for a hypothetical signal and background. Additionally, I have drawn a threshold in both distributions which represents the cut made to create a search region. In the first figure (left) the distributions of signal and background are relatively similar, and the signal region contains large amounts of background. In the second (right), the distributions of signal and background are greatly different, and this variable would allow for a much more effective signal region.
The goal of introducing ML, is to create a new feature in the data set where background and signal exhibit as little overlap as possible. This is very similar to how we introduce physics motivated high-level features like invariant mass, but instead of using an analytical function grounded in physics, we apply the output of a trained ML model. Then, once the ML-variable is created, we apply a cut (similar to CC) which will define an effective search region.

[17] Due to the neutralions being both heavy, neutral and interacting only weakly with ordinary matter.
[18] By effective cuts I mean cuts which removes large amounts of the background while at the same time preserves as much of the signal as possible.

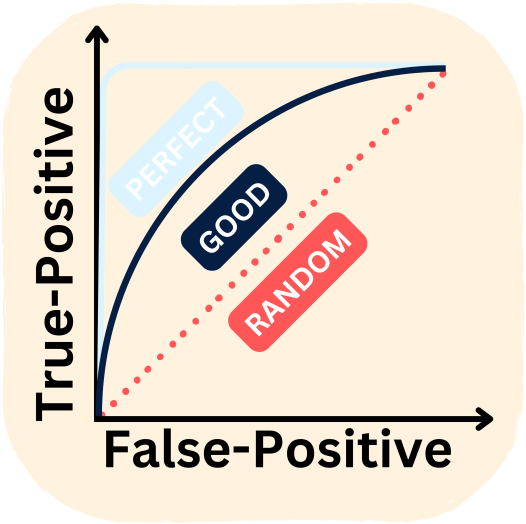


Figure 2.10: An illustration of a ROC curve and how a random, good and perfect classifier would differ.

How we create the ML-variable can vary depending on the type of ML. In the case of unsupervised ML, one aims to perform a type of anomaly detection. One unsupervised approach is to overfit on the background, then hope that said overfitting is reflected when a new process is introduced. An effective unsupervised approach would be incredibly beneficial, as it would be totally model independent. In other words, the same model could be applied to all signal. As mentioned in earlier sections (see section 2.1), the focus of this thesis will not be on unsupervised ML, but supervised. The largest difference between the two is the introduction of an additional data set, the simulated signal. By simulating and training on what we expect the new physics to look like, we are able to achieve a much more effective output which is tailored to what we expect the new physics to look like. Though, what supervised learning gains in performance it looses in generalizability, i.e. the ability to find new physics it has not trained on.

2.9 Model Assessment

2.9.1 The Rate of True-Positve - ROC Curve

A Receiver Operating Characteristic (ROC) curve is a tool used to measure and visualize a binary classifiers' ability to predict trends. In figure 2.10 I have plotted an illustration of a ROC curve. The curve is plotted on an xy-axis where the x-axis represents false-positive rates and the y-axis represents true-positive. The different values for the curve are the rate of true positives with different thresholds, i.e. the value deciding whether an event is 1 or 0, signal or background. If a classifier has learned nothing and is simply guessing, the ROC curve will be a linear curve going from 0 to 1. This line is often drawn in ROC curve. The better the classifier is, the higher the ROC curve will bend towards the upper-left corner of the graph. The closer the line is to the diagonal, the worse the classifier.
A metric often used to measure a classifiers' ability create an output which effectively separates two categories, is the Area Under the Curve (AUC). The larger the area, the better the separation. An ideal classifier which perfectly separates two categories will achieve a AUC of 1. A classifier which simply guesses, will achieve an AUC of 0.5. Both this cases assume an equal weighting of both signal and background.
This present section was taken from some of my previous work, and can be found in the following rapport [25].

2.9.2 Statistical Assessment - Discovery & Exclusion

The statistical aspect of the analysis will not be of focus in this analysis, nonetheless there are some expressions which would be helpful to define. Let us assume that we expect to see no new physics in the collision data, we can call this hypothesis the b(background)-hypothesis. To expect no new physics, is not the same as expecting no deviation. We still assume statistical uncertainties, or noise to effect the simulations which will affect the distribution of our simulations and lead to fluctuations. If these fluctuations are random, we can assume that the noise can be described by a Gaussian distribution (which is a good approximation for

large statistics), which has a mean of 0 and a standard deviation equal to $\sqrt{b}$, where b is the number of SM simulations. Using this assumption, we can state that the higher the deviation, i.e. the further away from the mean, the less likely such a deviation is explained by our b-hypothesis. Given a large enough deviation, we can even define the hypothesis as rejected with a certain confidence.
We measure the deviation between the observed collision data and the simulated SM data in units of significance, Z. For a given signal region with n_{obs} events of measured collision data and b simulated SM data, we define the significance of a deviation as

$$Z = \sqrt{2\left[n_{\text{obs}} \ln \frac{n_{\text{obs}}}{b} - n_{\text{obs}} + b\right]} \text{ or } Z = \sqrt{2\left[(s+b)\ln\left(1+\frac{s}{b}\right) - s\right]}, \tag{2.18}$$

where we have defined the signal as $s = n_{obs} - b$. In the case of large statistics ($b >> s$), we can write Z as

$$Z = \frac{n_{obs} - b}{\sqrt{b}} = \frac{s}{\sqrt{b}}. \tag{2.19}$$

By studying equation 2.19 we observe that the significance of a deviation is simply the signal measured in units of standard deviation of the Gaussian distribution. In physics, one often deems the results a discovery, or the b-hypothesis as rejected, if $Z > 5$.

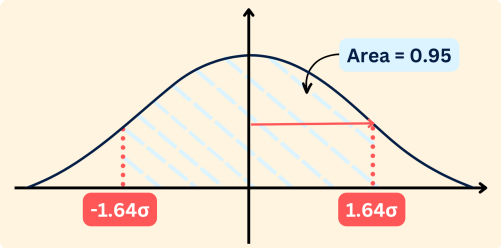


Figure 2.11: An illustration of a Gaussian distibution and the area under the curve defined by a significance equal to 1.64.

Before attempting to search for deviations in the real data, we are interested analysing the sensitivity of our method as well as calculate the expected number of signal in our signal region. To study the sensitivity of an analysis one looks only at the simulated background and signal and performs a study to see how well one can separate background from signal. The measure of the sensitivity of our models will be of great focus in this thesis.
In the case we are measuring the sensitivity, we do not have n_{obs}. Instead we simply define s as the number of events of the simulated signal inside the signal region. This number would be spesific for each mass combination and will be a measure for how sensitive the model is for that exact combination. We call the significance calulated using the simulated data for *expected significance*. In physics, we define a model as sufficiently sensitive, if it is able to achieve a significance of 1.64. This number is chosen based on its relation to the b-hypothesis. A significance of 1.64, equals the distance from the mean of the Gaussiwan distibution which sums to an area of 0.95 (see figure 2.11). There are many ways to inteprate this. A layman intepration (which is sufficient for this theisis), is that given the b-hypothesis is true, there is a 95% probability that we will measure a signal which produces a significance of less than 1.64. Therefore, there is only a 5% probability that the b-hypothesis can explain the deviation. For more information on significance and discovery in physics the reader is reffered to the slides [26].

Chapter 3

Implementation & Preparation of the Analysis

Before starting my analysis and applying the set of ML models, I first had to prepare the data set. This chapter will present the steps, tools and frameworks used in the preliminary work (and other aspects of the analysis), as well as present the resulting feature distributions. Towards the end of the chapter I will present the different architectures for each model and the general strategy applied to training.

3.1 The Simulated Data

3.1.1 Monte Carlo Simulated Data

Up to this point in the thesis I have discussed two data sets: the measured collision data and the simulated MC data. As far as my relation to the latter data set I have only used it directly in my analysis and have not been involved in production. The simulations applied in the creation of our simulated data set are all based on Monte Carlo (MC) simulations[19]. Through simulations, we are able to (among other things); test our understanding of the detector, model the expected SM background events in different regions or model new BSM physics. The pipeline for producing simulated events, closely mimic the pipeline from real collision to data set. The pipeline for simulating collision events can be summarized in the following steps

- *Generate events*: Simulate each event in the collision, marking each event with the corresponding process (see section 1.6)
- *Simulate detection of events*: Simulate the detection of each event in the different layers of the detector described in section 1.4.1
- *Digitization*: Translate the interaction between the particles in the events and the detector to real signals
- *Reconstruct events*: Go through the same steps applied to real collisions to reconstruct particles from the detector output.

3.1.2 The Simulated Signal

As I have mentioned in previous sections I will be searching for a large range of signals, all related to the same physics, but with different set of parameters, i.e. choices for the masses of the charginos and neutralino. In figure 3.1 I have drawn a grid displaying the different mass combinations searched for in this analysis. The masses range from $\tilde{\chi}_1 \in [0, 400]$GeV and $\tilde{\chi}_2 \in [200, 800]$GeV[20]. In the beginning of my analysis I only received a subset (which will be referred to as the *original* data set.) of the full signal set. In the figure I have marked the original data set with a white label in the top right corner of the box. The original signal set contains a total of 30 different mass combinations in the ranges $\tilde{\chi}_1 \in [0, 400]$ and $\tilde{\chi}_2 \in [400, 800]$. In the full signal data set there are a total of 90 different mass combinations.
Given that I received the original signal data set many months before receiving the full set, most of my

[19] For more information on MC simulation, the reader is referred to [27].

[20] The chargino, $\tilde{\chi}_1^{\pm}$, and second neutralino, $\tilde{\chi}_2$, are mass degenerate, meaning they share the same mass. As a consequence, I will refer to the mass of each particle interchangeably throughout the thesis.

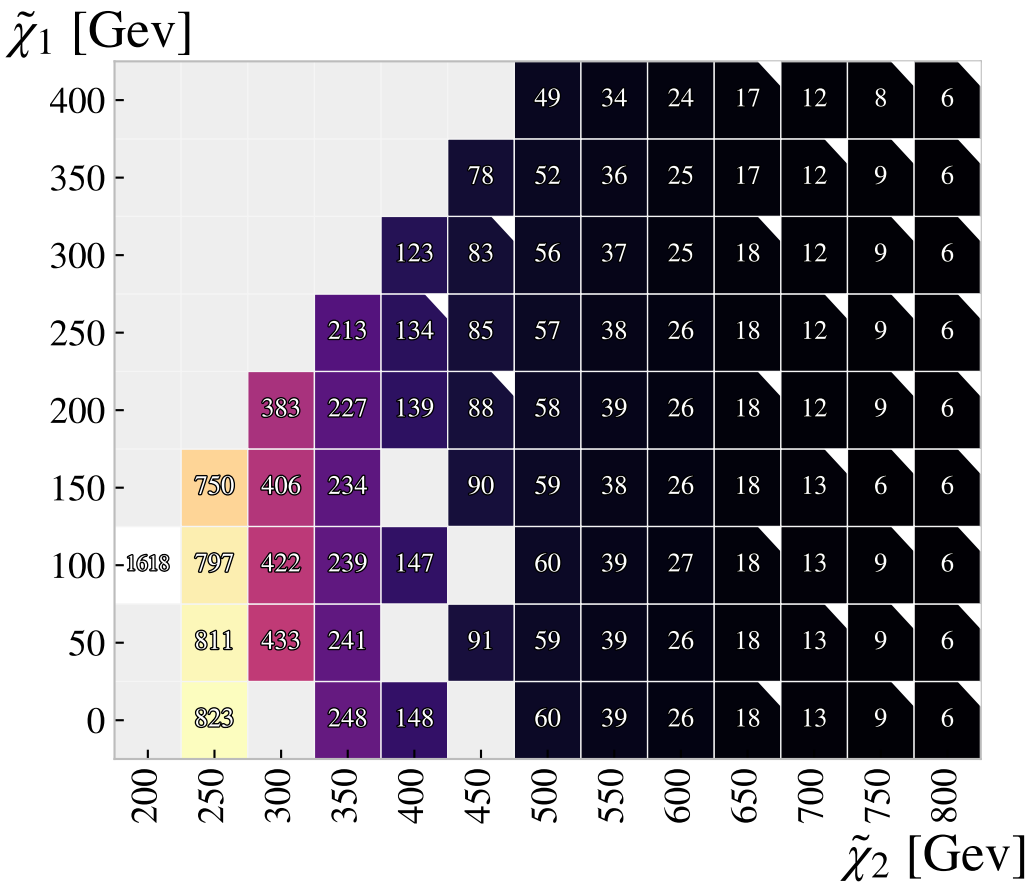


Figure 3.1: A depiction of all mass combinations and their respective event count in the full signal data set. Additionally, a white corner has been added to all combinations which define the original signal set.

results were carried out using the original set. Therefore, I choose to use the original data set to compare the performance of the models, then finally use the full set when comparing the results to previous analysis.

3.2 The Tools

Every year technology for generating and measuring particle collisions is improving. As a consequence, the amount of data increases drastically. The ATLAS experiment is one of the largest particle detector experiments currently operating at the CERN laboratory near Geneva. ATLAS alone generates approximately 1 petabyte of raw data every second from proton-proton collisions at the LHC. In this analysis I will be studying proton-proton collisions at a center of mass energy $\sqrt{s}$ = 13TeV recorded by ATLAS detector between 2015-2018, corresponding to an integrated luminosity of $139.0 fb^{-1}$. With amounts of data this large, data handling and storing is a big challenge. Therefore, taking advantage of sophisticated numerical tools and data frameworks is pivotal if scientific development is to keep up with technological development.

3.2.1 ROOT

In many aspects of my analysis, ranging from statistical analysis to plotting I utilize the `ROOT` framework. `ROOT` [28] is at its core a large `C++` library and data structure made specifically for big data analysis and computing, as well as data visualization. Today, all ATLAS data is stored as ROOT-files, summing up to more than 1 exabyte (10^{12}) distributed worldwide through the World LHC Computing Grid (WLCG)[29]. `ROOT` has many High Performance Computing (HPC) qualities which makes it ideal for particle physics analysis which demands heavy computations. Additionally, many particle physics-specific packages have been developed to make it an even better tool. Any function not already in library, can easily be added in a HPC-effective manner through `C++`.
Many of the plots presented in the present thesis were created using `ROOT`. `ROOT` has implemented a highly intuitive and effective Application Programming Interface (API) for data comparison and visualization.

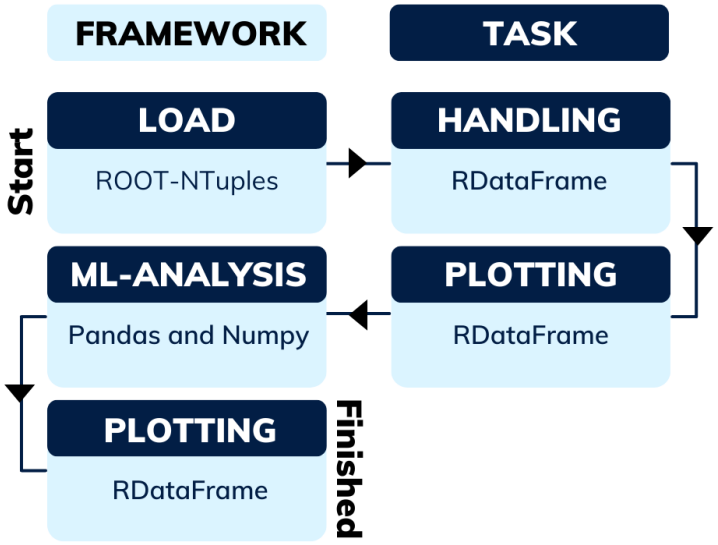


Figure 3.2: A visual summary of the workflow and frameworks used in the computational analysis.

`ROOT` allows for quick and direct comparison between data through an advanced graphical user interface. Furthermore, a lot of functionality for creating complex stacked histograms are implemented in the `ROOT` API, such as uncertainty calculations, ordering of histograms and statistical analysis tools. A final and important benefit of `ROOT` is its impressive handling of memory which makes it ideal to handle large amounts of data.

3.2.2 Data Structure and Frameworks

Both data sets used as input to the analysis in this thesis, the measured and simulated, were stored in `ROOT`, in the format of `NTuples`. `NTuples` are `ROOT` objects designed to store large amounts of data. They allow for non-symmetrical entries, meaning rows with different number of columns. The simulated and measured collision data are both stored in `NTuples`, with a matrix like structure where the columns represent the variables/features, and the rows represent each event. Hence, allowing non-symmetrical entries is essential for the purpose of storing data on particle collisions, given that not all variables are relevant for all events. For example the p_T of the third lepton in an event in a two lepton final state, does not make sense.
When starting preprocessing I load the `NTuples` using a `ROOT` interface called `RDataFrame`[21]. `RDataFrame` allows for easy addition of new columns as well as filtering of events through native functionality. As a consequence I used `RDataFrame` to calculate all features not already in the data set, such as the sum of transverse momentum, or the invariant mass of the three leptons.
After pre-processing was completed, I used `RDataFrame`'s *AsNumpy* function to translate the data frame into `Numpy` objects, which then allowed me to transform it to a `Pandas` data frame [30]. This is done because `Pandas`, like most ML-tools, work in a strict `Python` environment. `Pandas`, similarly to `RDataFrame`, includes a deep computational library, and is optimal for analysis of big data. When the full ML pipeline (data-handling, training, validation etc.) is completed the data is transformed back to `NTuples`, to take advantage of the plotting functionality in `ROOT`. The workflow of the data pre-processing is visualized in figure 3.2.

3.2.3 Computing Features in ROOT: Example

In this section I will cover a simple example to highlight the steps taken in preparing the data set used in the analysis. As mentioned earlier, the two main frameworks used were `RDataFrame` and `Pandas`. In this example I will cover the case of a feature not already in the `ROOT` file, namely the trilepton invariant mass. All loading of data is done using the `ROOT` framework and is easily read using the `RDataFrame` interface. To effectively generate new features we want to stay in `ROOT` environment. Therefore, we create a `C++` file, `helperfunction`. The `helperfunction` contains additional `ROOT` functions which are used in the analysis and are not already native to `ROOT`. In the case of computing the trilepton invariant mass, the `C++` function

[21] For full documentation on RDataFrame, see https://root.cern/doc/master/classROOT_1_1RDataFrame.html (Accessed 16.04.2023).

is created like shown in listing 3.1.
In listing 3.1, we see a couple of measures taken to uphold to the ROOT environment. The first measure is the typecasting to $VecF\_t$. $VecF\_t$ is created to wrap floats in the native ROOT vector object, $RVec$. The same is done in other cases such as float and integers, $VecI\_t$ and $VecB\_t$. The second measure was using TLorentzVector [22] to calculate the invariant mass. TLorentzVector is a class native to ROOT with many built-in functions. In this case we use the class to create four-vectors through the kinematic variables, P_t, η, ϕ and mass[23]. Then, through the TLorentzVector class we can simply add, through vector sums, together and extract the invariant mass of the three-lepton system.

```
// Compute the trilepton invariant mass
float ComputeInvariantMass(VecF_t& pt, VecF_t& eta, VecF_t& phi, VecF_t& m) {
  TLorentzVector p1;
  TLorentzVector p2;
  TLorentzVector p3;
  p1.SetPtEtaPhiM(pt[0], eta[0], phi[0], m[0]);
  p2.SetPtEtaPhiM(pt[1], eta[1], phi[1], m[1]);
  p3.SetPtEtaPhiM(pt[2], eta[2], phi[2], m[2]);
  return (p1 + p2 + p3).M();
}
```

Listing 3.1: C++-function which implementes the calculation of M_{lll}.

The functions implemented in the C++ library (helperfunction) off-loads, from Python, all the time, consuming work to C++, which significantly speeds up the computation. The C++ library is compiled prior to the execution of the main Python-code, which will be presented in the following. The main code, doing the selections, calculations and plotting, uses the RDataFrame interface in a Python environment. Whenever heavy calculations are needed the RDataFrame interface calls on function in the C++ library described above. In the code written in listing 3.2, I show a simple example of loading new C++ functions, filtering out events, calculating new features and adding said features to a histogram. The first three lines of code both compiles and include the helperfunction into the ROOT framework. Then I loop over all keys in the data frame, which in my case are the different channels (i.e diboson, $t\bar{t}$ etc.). For each channel I select 'good' events, based on the criteria I will present in section 3.4. Then I use RDataFrame's Define function to calculate and add a new feature using the ComputeInvariantMass-function. Finally, I save the feature as ROOT object called Histo1D, which I later plot using ROOT API. In this example I chose the trilepton invariant mass, but in the analysis all additional features were calculated using a similar method.

```
R.gROOT.ProcessLine(".L helperFunctions.cxx+");
R.gInterpreter.Declare('#include "helperFunctions.h"')
R.gSystem.Load("helperFunctions_cxx.so")

for k in df.keys():
    # Define good leptons
    isGoodLepton = "feature1 < cut1 && feature2 >= cut2"

    # Define good leptons in dataframe
    df[k] = df[k].Define("isGoodLepton",isGoodLepton)

    # Define number of good leptons
    df[k] = df[k].Define("nGoodLeptons","ROOT::VecOps::Sum(isGoodLepton)")

    # Demand 3 good leptons
    df[k] = df[k].Filter("nGoodLeptons == 3")

    # Define Invariant Mass (lll)
    df[k] = df[k].Define("mlll","ComputeInvariantMass(lepPt[isGoodLepton],
                                                      lepEta[isGoodLepton],
                                                      lepPhi[isGoodLepton],
                                                      lepM[isGoodLepton])")
    # Add to histogram
    histo["mlll_%s"%k] = df[k].Histo1D(("mlll_%s"%k,
                                        "mlll_%s"%k,40,50,500),
                                        "mlll",
                                        "wgt_SG")
```

Listing 3.2: Python-file for calling dataframe and calculating M_{lll}.

[22] For full documentation on TLorentzVector, see https://root.cern.ch/doc/master/classTLorentzVector.html (Accessed 16.04.2023).
[23] These features will be described in the next section 3.3.

3.3 Selecting Features for the Analysis

The choice of which features to study and which to neglect is crucial in a search for new physics. This is particularly true in the case of applying machine learning. The general motivation for including a given feature can be based on several factors. One reason being its ability to provide a trend which we can exploit when creating our signal regions. By this I mean that it is a variable were there is a diversity in the distribution between the different channels. This is often physics motivated, for example including a feature on the transverse missing energy on the basis that we are searching for a signal with large amounts of it. Another reason is grounded in the MC-simulation's ability to represent the variable. If there seems to be a clear deviation between the measured and simulated data, typically in some defined control regions believed to not contain any new physics, the feature could be adding noise to the data set. In the following sections I will discuss the various features included in the data sets (both simulated and measured), which are also summarized in the appendix in table 2.

3.3.1 Lepton Variables

Information regarding the kinematics of the leptons were added into the data set: i.e. the transverse momentum, p_T, the pseudo rapidity η and the azimuth angle ϕ. All kinematic features were represented individually for each lepton. For example p_T was added as three columns, $p_T(l_1)$, $p_T(l_2)$ and $p_T(l_3)$, where the ordering of the leptons were based on the momentum from highest (l_1) to lowest (l_3). Similarly, I added information regarding the charge ($\pm$) and flavor (electron, muon) of each lepton. Based on the kinematic variables the transverse mass M_T of each lepton was calculated, ΔR and the transverse component, E_T^{miss}, and azimuth angle, ϕ^{miss}, of the missing momentum.
The variables described in the section above are often considered as low-level features. These are very useful in many (if not all) searches and contribute little to no bias to your analysis. High-level features can be added to guide a hypothetical model in its search. The features used in this analysis were inspired by several publications of ATLAS I read in preparation of the analysis [31, 32].
Firstly I added different mass variables, namely m_{lll} and $m_{ll}(OSSF)$ (Opposite Sign Same Flavour (OSSF)). The first being the trilepton invariant mass and the latter being the dilepton invariant mass of the pair with OSSF. In the case of more than one possible OSSF-pair, the pair with the highest invariant mass was chosen. Secondly I added variables composed of the sum of the transverse momenta. These variables include the sum of the transverse momenta of all three leptons, $H_T(lll)$, of the same sign lepton pair, $H_T(SS)$, and the sum of the momentum for all three leptons added with the missing transverse energy $H_T(lll) + E_T^{miss}$. Finally, I added the significance of the E_T^{miss}, $S(E_T^{miss})$. The $S(E_T^{miss})$ is based on the log-likelihood ratio which compares the reconstructed E_T^{miss} to the expected missing transverse energy in the case that there is no E_T^{miss}. For more information on $S(E_T^{miss})$ the reader is referred to [33].

3.3.2 Jet Variables

Now we can have a look at the jet-features. Given that the final state of interest should be independent of jets, there are not many features added with jet information. But, given the risk of miss identification and errors in reconstruction, some features were added. The first features added were the number of jets, both all signal[24] jets and number of b-jets. The latter information was divided in two columns based on the efficiency of a multivariate analysis used to separate jet-flavors. The efficiencies used for b-tagging are 77% and 85%. The last information added for the jets were the mass of the leading pair (based on p_t) di-jet mass, m_{jj}.

3.4 Data Preprocessing and Preselection Cuts

To allow for deep learning and a thorough analysis one must try to keep as much of the data as possible. At the same time, including large amounts of irrelevant data, can be both redundant and destructive[25]. Therefore, preprocessing in the form of preselection cuts are necessary. The cuts applied in the analysis were grouped in two definitions, baseline and signal. The baseline requirements are written in table 3.3a and the signal requirements are written in table 3.3b. Both sets of requirements were taken from the ATLAS article from 2022 [31]. Given the definitions we demand that each event contains exactly three signal (3.3b) and three baseline leptons (3.3a), thereby removing any event with more or less.
Leptons are identified in the detector by using a likelihood-based method combining information from

[24] See section 3.4.
[25] By including large amounts of irrelevant data, you risk the ML-model tuning unnecessarily to remove easily reducible background, which could compromise performance.

different parts of the detector. The criteria of Loose or Tight identification are simply different thresholds in the discriminant, where Loose is defined with a lower threshold than Tight [34]. The overlap removal is used to solve any cases where the same lepton has been reconstructed as both a muon and an electron, and leptons reconstructed as jets (or vice versa). The boolean of *lepPassOr* simply applies a set of requirements to avoid any double counting. The cut for the longitudinal track parameters, z_0 is applied to ensure that the leptons originate from the primary vertex.
As for the requirements for the signal leptons, we require all baseline requirements to have passed with the addition of a few more. We require Loose isolation for both electrons and muons. This means imposing criteria on the activity (i.e. additional tracks and deposits in the calorimeters) in a cone around the lepton. This is used to suppress QCD-background events and reduce fake leptons. Similarly to the z_0-cut, the transverse track parameter is also used to ensure origin from primary vertex.

Requirement	Baseline electrons	Baseline muons
Identification	–	Loose
Overlap Removal	lepPassOR	lepPassOR
$\eta-$ cut	$\lvert\eta\rvert < 2.47$	$\lvert\eta\rvert < 2.7$
$\lvert z_0 \sin(\theta)\rvert$ cut	$\lvert z_0 \sin(\theta)\rvert < 0.5$ mm	$\lvert z_0 \sin(\theta)\rvert < 0.5$ mm

(a)

Requirement	Signal electrons	Signal muons
Baseline	yes	yes
Identification	Tight	–
Isolation	LooseVarRad	LooseVarRad
$\lvert d_0\rvert/\sigma_{d_0}$ cut	$\lvert d_0\rvert/\sigma_{d_0} < 5.0$	$\lvert d_0\rvert/\sigma_{d_0} < 3.0$

(b)

Table 3.3: Two tables displaying the baseline 3.3a and signal 3.3b requirements applied to the data as part of the preprocessing.

In addition to the simple cuts, we must insure a good comparison between MC- and real data. Often one finds large deviation between the two in the case of either very large or very small transverse momentum, p_T. The latter case can often be caused by poor reconstruction or miss identification. These are issues we aim to solve by checking different triggers.
The triggers used in the data set were dielectron and dimuon triggers taken from previous ATLAS publications [35–37]. Unfortunately, in the earlier parts of the analysis, I discovered that the data set I was given, did not contain the correct information regarding the triggers. After spending some time trying to compensate for this, my supervisor (Eirik Gramstad) suggested an alternative. Instead of filtering using the triggers, an additional criterion of at least two leptons containing $p_T > 20$GeV was placed on the data. The criteria ensure that the momentum of the leptons are above the threshold of the triggers

Validation

As mentioned in previous sections, the comparison between MC simulated and measured collision data is a crucial part of the analysis, and we must therefore ensure an adequate reconstruction of the real data before we begin the analysis. This is not only true for the low-level features, but for all features used in the analysis. Therefore, we will in this section compare both sets of data for a subset of features included in the analysis.

Feature	l_1	l_2	l_3
p_T	–	33*b*	33*a*
η	–	33*c*	33*d*
ϕ	34*a*	34*b*	34*c*
M_T	34*d*	34*e*	34*f*
Charge	35*a*	35*b*	35*c*
Flavor	35*d*	35*e*	35*f*

(a)

Feature	*Reference*
$\phi(miss)$	36*b*
M_{lll}	36*c*
$M_{ll}(OSSF)$	36*d*
Sig E_T^{miss}	35*a*
$H_T(lll)$	36*f*
$H_T(SS)$	37*a*
$H_T(lll) + E_T^{miss}$	37*b*
ΔR	36*a*
Nr of signal Jets	37*c*
M_{jj}	37*d*
Nr of B-jets(77)	37*e*
Nr of B-jets(85)	37*f*

(b)

Table 3.4: References to figures for all lepton (3.4a) and event (3.4b) specific feature distribution.

In figure 3.5, I have drawn the event distribution for the p_T (3.5a) and η (3.5b) for the leading lepton, as well as the E_T^{miss} (3.5c) and flavor combination (3.5d) of the three leptons in the final state. The error bars in each bin are set by default to $\sqrt{\#events\ per\ bin}$. The distribution of the remaining features have been

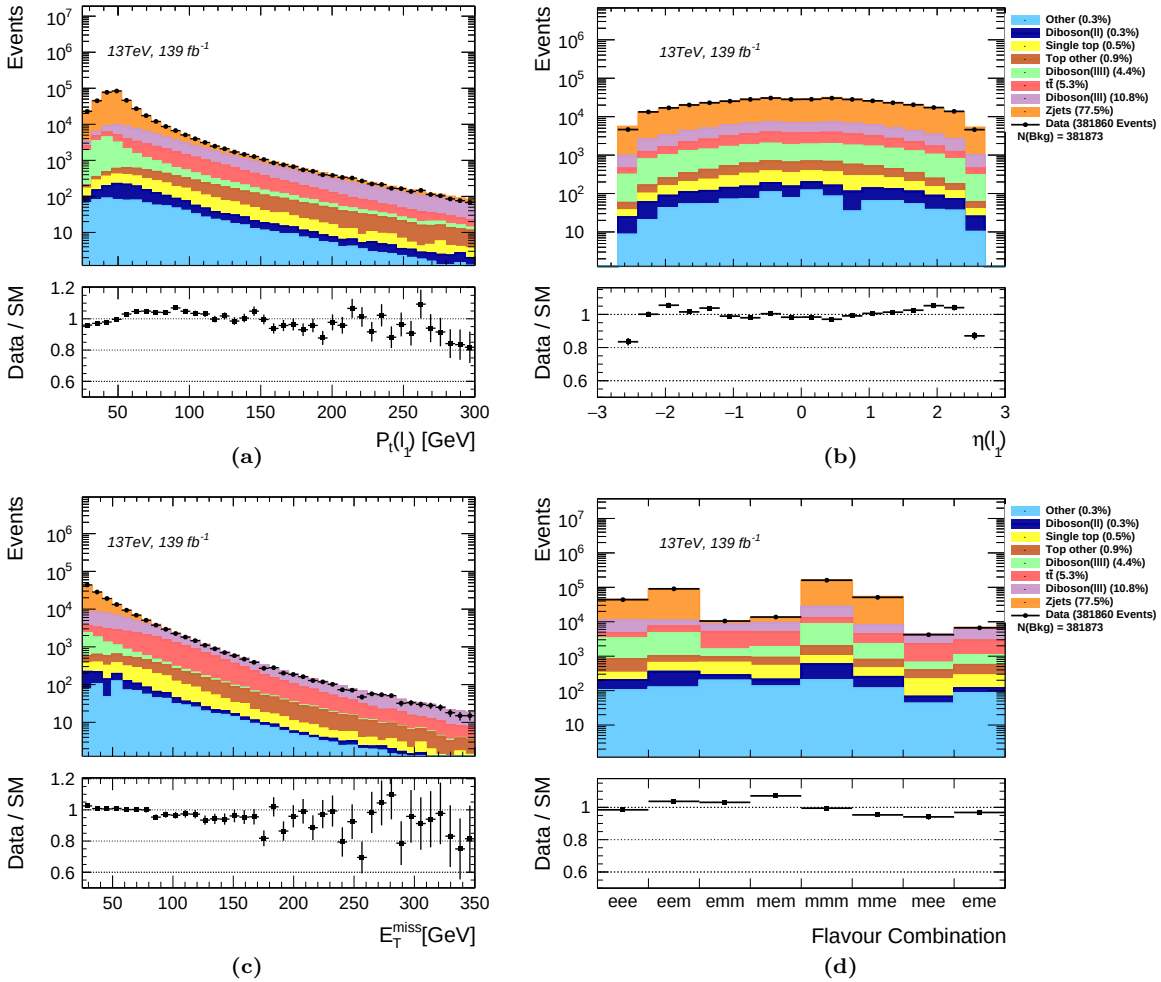


Figure 3.5: MC and real data comparison showing the p_T 3.5a and η 3.5b of the first lepton. The distribution of E_T^{miss} 3.5c and the flavor combination of the three final state leptons 3.5d.

added in the appendix B.1. I have added two tables with references to each feature not displayed in the main thesis, in the case the reader is interested in studying specific features. Table 3.4a contains all lepton specific features and table 3.4b contains all event specific features.

Under each figure I have drawn the ratio between the measured collision data and the MC for each bin. By studying the ratio-plots for each figure we observe that the ratio for all bins for all features are between 1.2 and 0.8. Most bins even lying closer to 1. The bins displaying the largest errors are in the higher p_T-range. This is exemplified in figures 3.5a and 3.5b[26].

In the figure 3.5a we can observe that for (relatively) small p_T (< 100GeV) all events lie well within a range of $[0.9 - 1.1]$ ratio. Whereas for higher p_T (> 200GeV), the errors move closer to a range of $[0.8 - 1.2]$ ratio. The difficulty of simulating SM processes in high p_T range is a known phenomenon, and not specific to this analysis. Thankfully, a smaller portion of the data lies in the high p_T-range and therefore most of the simulation exhibits a solid comparison to measured data. By studying the figures in the appendix B.1, we can deduce that this trend continues throughout the full feature set. Additionally to studying the ratio for each feature in different bins, we can read from the labels that there are a total of 381873 measured collisons in the data, compared to 381860 simulated events. Simply put we observe that the MC seems to adequately imitate the trends of the observed data for all features used in the analysis.

Apart from the excellent agreement between observed and simulated data, we can note a couple of other expected but interesting points. The first being the size of each channel. $Z - jets$ is by far the largest channel followed by the $Diboson(lll)$. Although $Z - jets$ is the largest channel, by comparing the Feynman-diagrams of each channel (section 1.6), $Diboson(lll)$ should be the hardest background to separate due to the similarities in the final states of the signal[27].

[26] Due to η's dependence on the polar angle (see equation 1.1), the larger the p_T the higher the absolute value of η

[27] I.e. Diboson(lll) has a 3 lepton final state with large amounts of missing transverse energy.

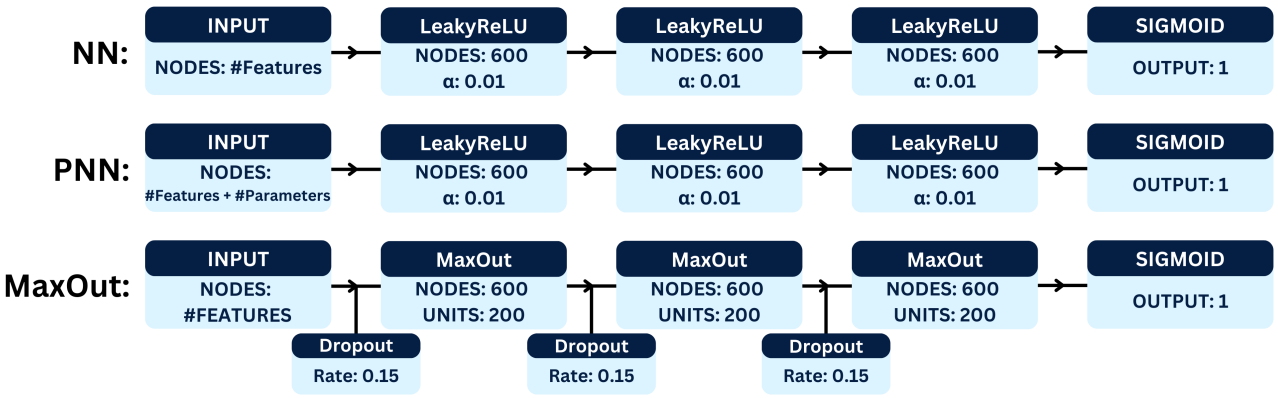


Figure 3.6: A visual summary of the network architectures used in this analysis, for the ordinary dense NN(top), the PNN(middle) and the maxout model(bottom).

3.5 The Machine Learning Models

3.5.1 Model Selection

In this analysis I have chosen to compare 4 different 'types' of ML-models, ordinary dense Neural Network (NN)[28], Parameterized Neural Network (PNN), ensembles (networks utilizing the channel-out, SCO or maxout layer) and Boosted Decision Trees (BDT). The first three methods are all variations of NN. I have deliberately chosen to focus on NN given that there is far more freedom in the design of the architecture of a NN compared to `XGBoost`. Additionally, this was motivated by the selfish reason being that I found the networks more interesting to study and dissect. Therefore, most of the analysis, comparisons and discussions are focused on the NNs, while `XGBoost` is included as a loose benchmark.
The choice of the three network architectures is motivated by a wish to compare three relatively different approaches. Especially in how they perform with a diverse data set. The LWTA models apply trend specific paths, the PNN includes parameters in the feature set, and the dense NN relies on a deep set of weights and biases to uphold long-term memory. I would assume that the optimal architecture would be a network which combines elements from each, but for the purpose of discussion and research I have chosen to keep them somewhat separate.
All NN variants implemented in this analysis were created, trained and deployed using the `TensorFlow` framework [38]. This choice was rooted in several factors, for example that `TensorFlow` has an intuitive API, it has very effective HPC attributes, and it includes a large diversity in functionality which allows for experimentation.

3.5.2 Model Architecture

When choosing a network architecture, there are several ways to proceed. One way is to apply a grid search. A grid search is simply defining a grid of parameters to test, then running through all combinations and choosing the highest performer. With a sufficient amount of tests, a grid search should converge towards an optimal architecture. Grid searches are very common and there exists a large range of very complex varieties [39]. For my analysis I chose not to perform a grid search, for several reasons. The first being interpretability. Understanding a NN is already hard, allowing for complex and unique architectures would only add another layer of mysticism. The second is the size of the data set. The larger the data set, the more data would be needed to adequately perform tests for each combination of parameters. Not only is this time-consuming, but trying to mediate this issue could lead to poor performance. The third and most important reason is that I wanted to experiment with the architectures. By manually tuning the parameters I was able to achieve a far better understanding of the final architecture. In the following sections I will describe the architecture of all the variations of networks and the BDT used in this analysis.

[28] Note that for the remaining part of this thesis, I will refer to this model as a 'dense NN', 'ordinary dense NN' and in some plots, just NN.

Dense Networks - Ordinary and PNN

The purpose of the dense NN is to compare the more complex networks to what is usually considered an ordinary dense NN. The general structure of the network is summarized in figure 3.6, with the network label NN. The figure shows a dense NN with three hidden layers, all with 600 nodes each. As is the default setting of the dense layer implemented in `TensorFlow`, the weights and biases are initialized using the so-called *Glorot Uniform Initializer* (see the article by Glorot [40] for more information). All hidden layers utilize the *LeakyReLU* activation (see section 2.6.4) with $\alpha = 0.01$, and the output layer utilizes the sigmoid function. The NN, similarly to all network variants in this analysis, utilizes the ADAM optimizer (see section 2.2.3) implemented in `TensorFlow`[29] with the default settings, where the cost function is chosen to be the binary crossentropy function described in section 2.9.
The PNN architecture is included to represent the model proposed in the article by Baldi et al. [8]. The architecture is illustrated in figure 3.6, with the label PNN. The figure shows a practically identical structure to the dense-NN, with the only difference being in the input layer. As was discussed in section 2.6.6, the PNN includes free parameters of the signal[30] alongside the features in the input layer.

MaxOut, Channel-Out and SCO

The ensemble models are slightly more complex than both the dense NN and PNN in terms of architecture. To limit the complexity for comparison reasons I choose to build an identical architecture for the maxout, channel-Out and SCO model, with the only difference being which of the three layers is used. In figure 3.6 I have illustrated the maxout model architecture, with the label MaxOut. The figure shows a network with six hidden layers, three maxout and three dropout. The network alternates between dropout and maxout, starting with dropout and finishing with MaxOut. The MaxOut layers have 600 nodes each which is reduced to the 200 nodes with the largest activation in their respective units. Each dropout layer has a dropout rate of 0.15. The Channel-out and SCO models have the same architecture as the maxout, but replacing the maxout layer with each of the two respectively.

BDT

The main motivation to apply a BDT is simply to benchmark my analysis, and therefore not a lot of effort has been put into the design of the architecture. As a consequence, the default parameters[31] of the `XGBoost` model have been used. The main parameters of the model are summarized as the following:

- η (learning-rate) = 0.3
- Max depth = 6
- Maximum number of trees = 100
- Objective = 'binary:logistic'

The objective parameter refers to the learning task of the model, where *'binary:logistic'* specifies logistic regression for binary classification.

3.5.3 Creating Custom Layers

The field of ML is one of the most dynamic and fastest growing fields of research today. This means that regardless, of the brave attempt made by voluntary contributors and many large tech companies, there will always be new and exciting ML tools and algorithm not yet implemented in their library. This was certainly the case in this thesis. For several of the non-dense layers I was forced to dive into the world of ML development and create my own implementation.
All LWTA methods described in section 2.6.5, channel-out, SCO and maxout, resemble a layer already implemented by `TensorFlow`. This layer is called MaxPooling1D[32]. But, to allow for experimentation I decided to not use `MaxPooling1D`, but instead implement my own custom layers. All custom layers were implemented by creating new functions inside `TensorFlow`'s dense layer, which are called when the network performs a forward pass. In the following sections I will describe the algorithms underlying the implementation of each layer. For the actual `python`/`TensorFlow` implementation, see the appendix C.

[29] See `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam` (Accessed:24.04.2023) for the full default parameters.
[30] In our case, the masses of the chargino and neutralino.
[31] See `https://xgboost.readthedocs.io/en/stable/parameter.html` for a complete overview of default parameters.
[32] For more information on MaxPoolind1D, the reader is referred to the documentation (`https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool1D`, accessed 24.03.2023).

Channel-Out

In algorithm 1, I summarized the implementation of channel-out used in this analysis. For each input which is to be passed through a forward-pass, the function first passes it through the weights and biases as described in section 2.6.2. This is true for all the other layers as well. In line 7 I reshape the input to add a dimension which creates the units. In line 10 I create a new variable, *Output*, by reducing each unit to the largest activated node. Further, I set each node which is not equal to the largest activation to 0. Finally, I reshape the output to the original shape and return the output.

Algorithm 1 The pseudocode for implementing the channel-out layer in TensorFlow

```
1:  def Channel-out(Input):
2:      % Pass input through weight kernel and adding bias terms
3:      Input ← Input × Weights
4:      Input ← Input + Bias
5:
6:      % Reshape input into units
7:      Input ← Reshape(Input, (Nr Units, Size of Units))
8:
9:      % Reduce input to the largest activation in each unit
10:     Output ← Max(Input)
11:
12:     % Set original activation where activation is largest and 0 where it's not
13:     Output ← Where(Input == Output, Input, 0)
14:
15:     % Reshape to original size
16:     Output ← Reshape(Output, (Input's Original Shape))
17:     return Output
```

SCO

In section 2.6.5 I described how SCO is an extension of channel-out. This is certainly also the case for the algorithms explaining the implementation. In algorithm 2 I have described the algorithm used in the implementation of SCO. Similar to channel-out, the algorithm begins by passing input through weights and biases. Then, contrary to channel-out, the nodes of the input are shuffled. This is to ensure that when the input is reshaped (in line 10), the units are made differently for each pass through the function. Again the maximum activation is found to create a new variable, this time called $OutputShuffle$. In line 16 $InputShuflle$ and $OutputShuffle$ are compared such that where $InputShuflle$ is equal to the largest activation, the value is set to 1 and otherwise set to 0. $OutputShuffle$ is then unshuffled, and reshaped to the original shape. Finally, the input which has been passed through the weights and biases, is multiplied with *Output* and returned.

Max-Out

In algorithm 3 I have summarized the logic behind the implementation of the maxout layer. Of the three layers (channel-out, SCO and maxout), maxout was the simplest to implement. After passing the input through the weights and biases and reshaping it to form the units, the input is reduced to only include the largest activation in each layer. Finally, the output is reshaped to the size equal to the number of units.

Algorithm 2 The pseudocode for implementing the SCO layer in TensorFlow

```
def SCO(Input):
    % Pass input through weight kernel and adding bias terms
    Input ← Input × Weights
    Input ← Input + Bias

    % Shuffle all the values
    InputShuflle ← Shuffle(Input)

    % Reshape input into units
    InputShuflle ← Reshape(InputShuflle, (Nr Units, Size of Units))

    % Reduce input to the largest activation in each unit
    OutputShuffle ← Max(InputShuflle)

    % Set 1 where activation is largest and 0 where not
    OutputShuffle ← Where(InputShuflle == OutputShuffle, 1, 0)

    % Un-shuffle all the values
    Output ← UnShuffle(OutputShuffle)

    % Reshape to original size
    Output ← Reshape(Output, (Input's Original Shape))

    % Multiply input with output to set all input that are not the largest, to zero
    NewOutput ← Input × Output

    return NewOutput
```

Algorithm 3 The pseudocode for implementing the maxout layer in TensorFlow

```
def MaxOut(Input):
    % Pass input through weight kernel and adding bias terms
    Input ← Input × Weights
    Input ← Input + Bias

    % Reshape input into units
    Input ← Reshape(Input, (Nr Units, Size of Units))

    % Reduce input to the largest activation in each unit
    Output ← Max(Input)

    % Reshape to size equal the number of units
    Output ← Reshape(Input, (Nr Units))
    return Input
```

3.6 Model Training and Validation

3.6.1 Training and Validating Data

When building an ML-model the usual approach is to divide your data into three sets; training, validation and testing. The training set is used to tune the internal parameters of the model, i.e. the weights and biases of a NN or the cuts of a DT. The validation set is used to tune the hyperparameters of the model, f.ex. the architecture of the NN or the maximum depth of the DT. The test set should only be used when the model is finished, and is used to benchmark the models' performance. In our case, the performance we are interested in is the performance on the full MC-bakground set and its comparison to the measured collision data. Therefore, in this analysis only two sets of data will be used, training and validation where both are sampled from the simulated MC data set. The validation set was added as a precaution to reduce overfitting when applied to the measured collision data.
The overarching strategy in creating the training and validation set is summarized in the following steps:

1. Shuffle the data set.
2. Split the data set in two, training (80%) and validation (20%)
3. Scale the two data set such that the sum of the weights of the background is equal to the sum of the weights of the signal in each data set.
4. Scale both data sets using the Standard Scalar approach (see section 2.4.1) using the parameters (the mean and standard deviation) of the training set on both sets.

The first step ensures an equal distribution of processes in both data sets. The $80-20\%$ is a popular practice in ML (see [41]) and was chosen here for convenience. The third step refers to the sample-weights introduced in section 2.2.2. The sample-weights are only included in the simulated data (and therefore training data), to scale each simulated event based on the (among other things) probability of the collision, otherwise known as the cross-section. The scaling of the weights is done to ensure an equal prioritization for background classification and signal classification. In other words, if there was a large imbalance between signal and background, the model would be motivated to tune more towards one trend than another. The final step is motivated in the section regarding data handling 2.4.1.

3.6.2 Training Strategy

To best compare the different ML-models, I decide to apply the same training strategy to all (including the BDT). The strategy is simply to train the model with the training set, then apply an early-stopping algorithm (see section 2.5.1) with a cut-off based on the performance on the validation set. For every epoch the model makes a prediction on the validation data set, and logs the results. If more than 10 epochs go by without improving on the epoch with the best result, training stops and the weights of the epoch corresponding to the best performance on the validation set are reset. This strategy was repeated for the BDT, but logging for each new additional tree instead of epoch.
The performance from each epoch was measured in AUC (see section 2.9.1), where the AUC was calculated with the same weighting as in training (50% signal and 50% background). The distribution of signal vs background is important when studying AUC, as it will greatly affect the value. For example, a classifier which predicts both signal and background to be background is a poor classifier. But the larger the amount of background relative to signal, the higher the AUC would be.

3.7 Handling Negative Weights in a BDT

In section 3.6.1 I mentioned the inclusion of sample weights in the training of the ML models. Most weights are in the range of $w_i \in [0, 1]$, but some sample weights are negative. The negative weight problem is a well known problem in the world of ML-analysis for HEP, and is a consequence of higher perturbative accuracy. For the purpose of visualizing distributions and training a NN, negative weights are not a problem. But, when using the weights in the training of the `XGBoost` classifier, the algorithm does not work[33].
Before deciding how to mediate the issue I decided to plot the distribution for a small subset of features for all the events with negative weights. In figure 3.7a and 3.7c I plotted the distribution of the leading lepton for

[33]Upon further research in the codebase of XGBoost, it seems as though negative weights are not allowed as it interfere with XGBoost's use of the hessian matrix. Due to time constraints, I decided not to pursue this further.

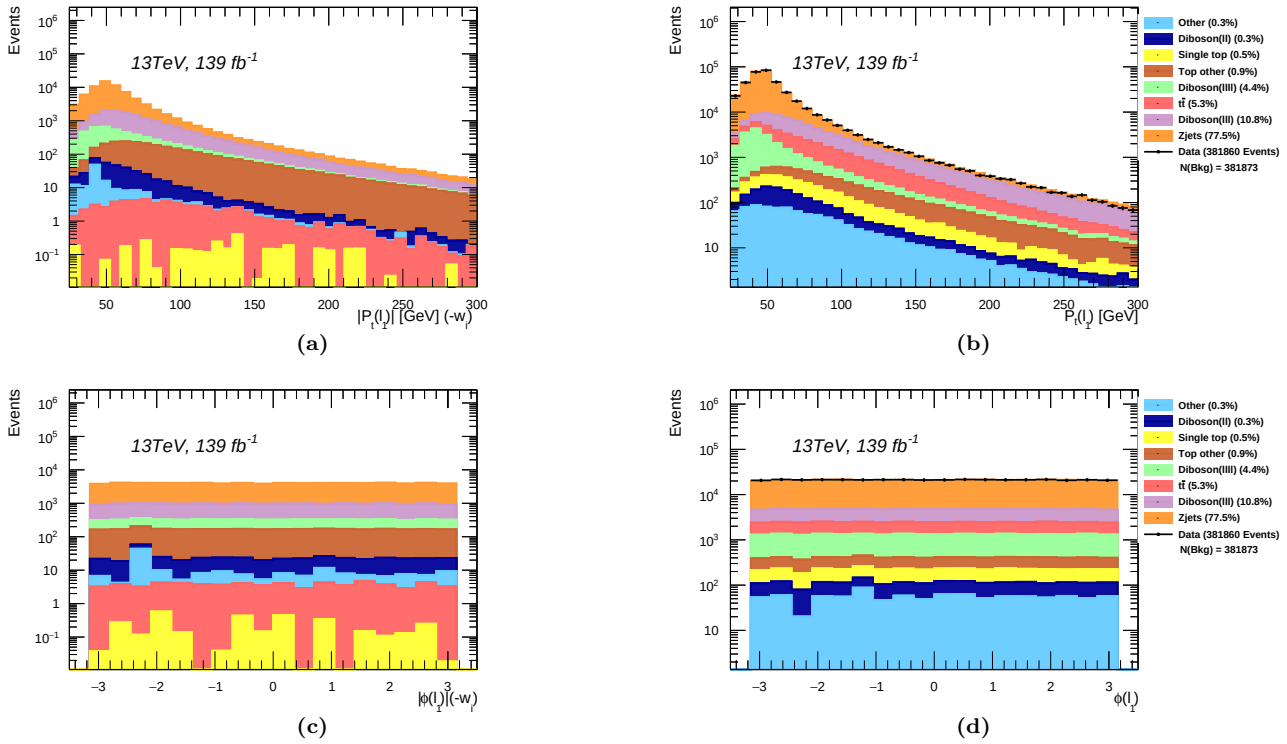


Figure 3.7: The event distributions of the leading lepton for the features P_t and ϕ. Figures 3.7a and 3.7c display only the events with negative weights (for P_t and ϕ respectively) whereas 3.7b and 3.7d show the full data set.

the features p_T and ϕ, but only including events with negative weights. Additionally, I drew the distribution for the full data set for the same features in figures 3.7b and 3.7d. By comparing the distributions, we can observe that the full data set and the negative weights' subset are very similar. There are differences in distributions when comparing the SM individually, but when studying the background as a whole, the two sets exhibit the same trends across all regions. This means that the negative weights essentially scale the background uniformly over the full feature space, which motivates treating the negative weight problem uniformly over the full feature space. How one chooses to deal with this problem can heavily affect the results and thus many solutions are suggested. Given the main focus of this report, the simplest solution was chosen. The solution was motivated by the internal notes of the ATLAS article [42], who introduced the solution when faced with the same problem. The solution is to take the absolute value and normalize all the weights, conserving the total sum of the weights and at the same time changing all negative signs. The simple procedure is shown bellow in equation 3.1,

$$w_i = P \mid w_i \mid \tag{3.1}$$

$$P = \sum_{j=0}^{N-1} \frac{w_j}{\mid w_j \mid}, \tag{3.2}$$

where w_i is the weight of an event i, $i \in [0, N-1]$ and N equals the total number of events.

3.8 Defining the Signal Region and Calculating the Significance

When producing an output I define the signal region through a brute force method which maximizes the significance. Based on trial and error and the study of the output from several models I found that, due to the unbalanced ratio of signal to background, a successful signal region would have to be very 'strict'. Therefore, for each model output I implemented a set of 200 cuts ranging from 0.9 to the highest output of the model, with most cuts falling in the range of > 0.98. For each cut the significance is calculated where all signal and background events larger than said cut are included in the signal region. The cut corresponding to the largest significance is used to define the final signal region.
In most calculations of the significance I use equations 2.18. This equation does not include any uncertainty,

which is why any results using said equation is only for comparison reasons between the different ML models. In the final sections of the thesis I am interested in including an uncertainty to the results. To do this I utilize a function in `ROOT`, the `RooStats.NumberCountingUtils.BinomialExpZ`, which takes the number of expected signal and background, as well as the systematic uncertainty of the background, and calculates a significance based on this.

Chapter 4

Results & Discussion

In this, the final chapter of my thesis I will present the results from my analysis. First, I will present the result from my study of each model individually, followed by sections comparing the performance from all models on a subset of the data. Finally, I will compare the models on their performance on the complete signal grid and compare them to a similar ATLAS analysis from 2021 [32].

4.1 Benchmarking the Analysis with Boosted Trees

Boosted trees have been an essential part of High Energy Physics (HEP) analysis for many years (see [43, 44]). The `XGBoost` framework has likewise been used as the benchmark for many ML analysis, often chosen for its performance in sensitivity and extreme High Performance Computing (HPC) capabilities. Another reason for its popularity is the incredibly simple Application Programming Interface (API), allowing to create and predict a model in two lines of code. Additionally, its incredible boosting capabilities means that it is little affected by variations in its structure. This leads many to the conclusion that the default parameters (see section 3.5.2) are often the best.
I choose to perform a sensitivity analysis for the original signal data set using an `XGBoost` model. The results will act as a benchmark when performing further testing with NN-variants. In figure 4.1 I have presented a grid displaying the expected significance (see section 2.9.2) for each mass combination in the original signal data set. The significance presented in the grid (and will be in the next results to come) was calculated with equation 2.18. In the figure we can observe that the `XGBoost` model performs better for smaller masses. This results can be somewhat counterintuitive, due to signal with smaller masses having a larger resemblance to background than signal with large masses. The explanation is simply that there are far more events with smaller masses than there are with larger masses. By studying figure 3.1 we know that there are a total of 134 events with ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 400$GeV) compared to 6 events with ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 800$GeV). Not only does this mean that the model will have had more small mass signals to train on than large mass, but also that a potential signal region would have to keep far more of the large mass signal to achieve a high significance.
To further investigate the performance of the `XGBoost` model, I have drawn the distribution of the output for the entire background data set as well as for signal with 4 different mass combinations. The result is found in figure 4.2. The figure shows the output of the `XGBoost` model with the full output range 4.2a and the output ranging from [0.975-1.00][34] (4.2b). From the figure we observe a clear separation between signal and background, where most of the signal is given a high output (> 0.9) and most of the background is given small output (< 0.1). To further support the effect of statistics in the signal, we can take note of the amount of signal in the higher range of the output (0.975-1). Although the model is able to achieve a much higher effectiveness (is able to preserve more of the signal) for higher mass signals, there are over 4 times as many events with mass ($\tilde{\chi}_1 = 250$, $\tilde{\chi}_2 = 400$GeV) (the lightest mass combination in the figure) then there are of any other.
A final observation from figure 4.2, is the study of which SM processes contribute the most for higher model output. In figure 4.2b, we can see that in the output range of $[0.975 - 1]$, the highest contributing processes are $Diboson(lll)$, $Top-Other$ and $t\bar{t}$. On the other hand $Z-jets$, which is originally the largest contributing processes, is almost removed in the higher output range. This aligns with my predictions made when studying the different processes in section 1.6.

[34] This interval was chosen to study the higher output region.

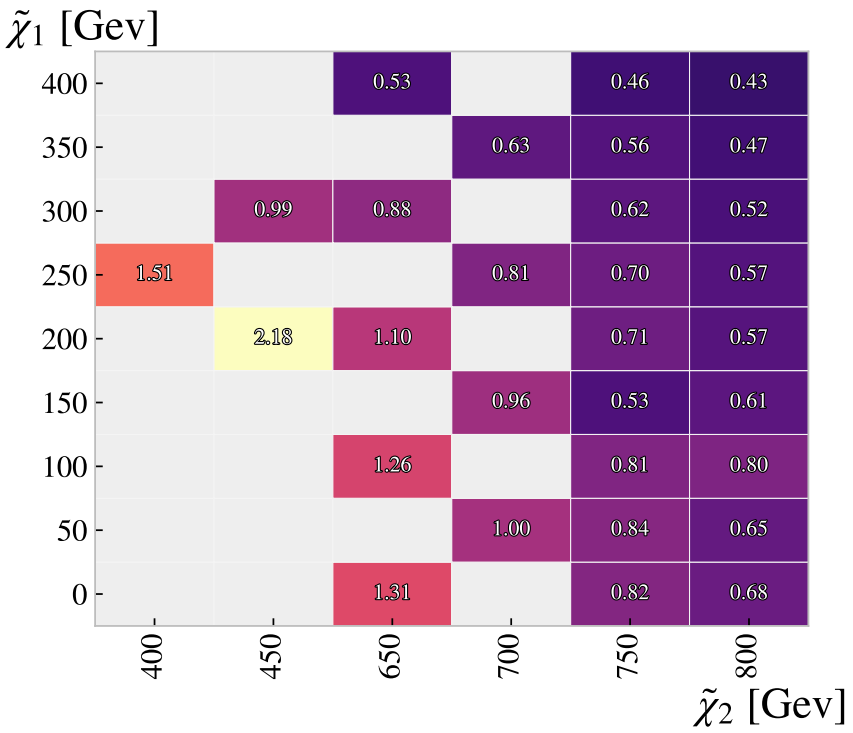
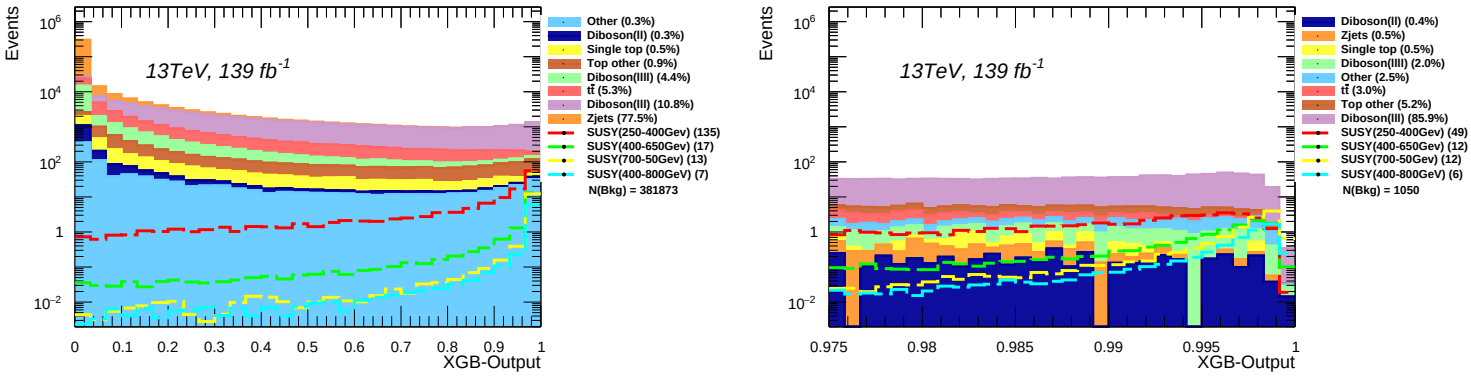


Figure 4.1: A grid displaying the expected significance on the original signal set, using the signal region created by the *XGBoost* network.



(a) **(b)**

Figure 4.2: The output distribution from a trained XGBoost model for the background and signals with 4 different mass combinations: ($\tilde{\chi}_1 = 250$, $\tilde{\chi}_2 = 400$GeV), ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 650$GeV), ($\tilde{\chi}_1 = 700$, $\tilde{\chi}_2 = 50$GeV) and ($\tilde{\chi}_1 = 400$, $\tilde{\chi}_2 = 800$GeV). The figure includes the full output range (4.2a) and the output ranging from 0.975-1.00 (4.2b). The number in parentheses indicate the fraction of each background and the absolute number of events for each signal point. The total number of background events is also shown.

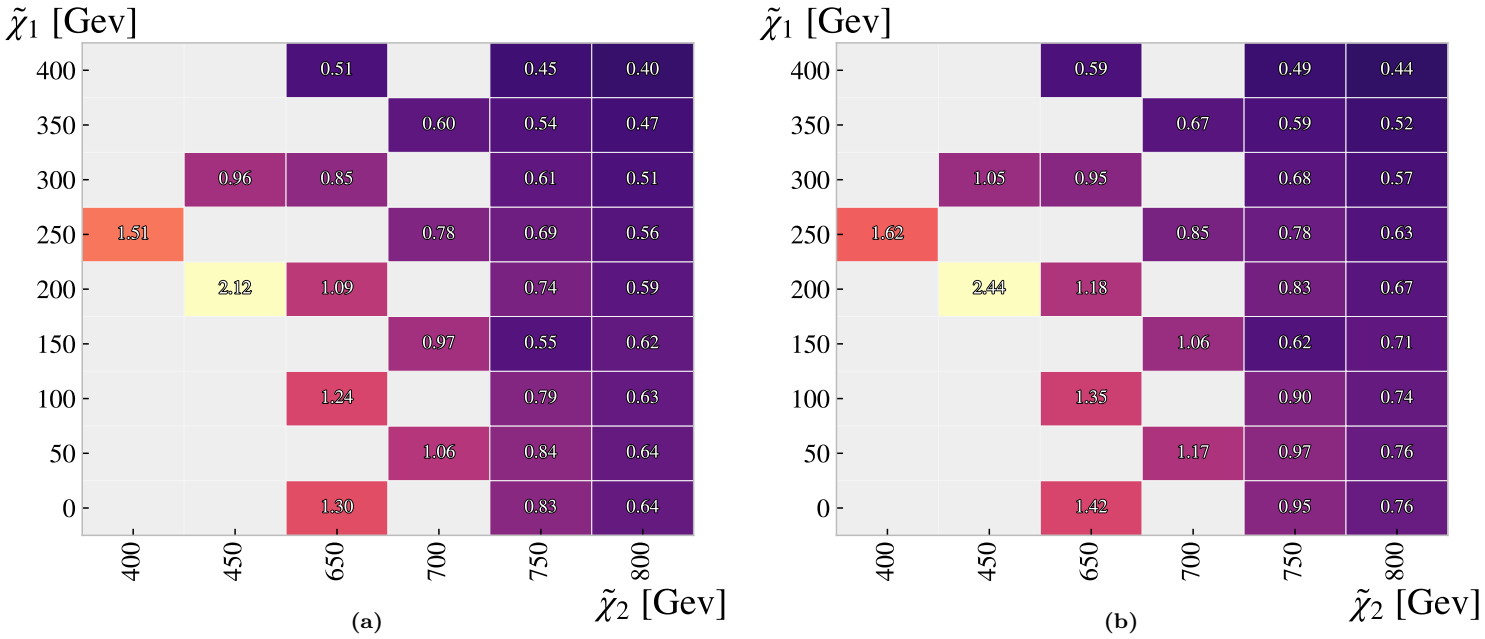


Figure 4.3: Two grids displaying the expected significance on the original signal set, using the signal region created by twp dense NN, one with 20 nodes per hidden layer 4.3a and one with 600 4.3b.

4.2 Dense Ordinary Neural Networks

4.2.1 A Study of the Number of Parameters in a Network

In previous sections I have discussed the reasoning for not wanting an automated process for building network architectures (otherwise known as hyperparameter searches). Although, I found manually choosing the architecture for each model to be preferable in this analysis, there are some downsides to that decision. When building each model I wanted to ensure that the performances of said models were representative of the category of models I wanted to build. In other words, I wanted to ensure that any drawbacks were related to the type of model and not a result of poorly chosen number of layers or nodes. Therefore, I wanted to relieve as much randomness in the choice of architecture as possible.

In this section I present a comparison of the results from two ordinary dense NN with different amount of nodes. The first network uses the dense NN architecture described in 3.5.2 and illustrated in figure 3.6, while the second network uses the same activation functions and amount of hidden layers, but with only 20 nodes in each layer. The comparison presented in this thesis represent a sample of networks I choose to compare when deciding the number of nodes and hidden layers to be used in the analysis. Both networks apply the training strategy described in section 3.6.2.

Figure 4.3a displays the expected sensitivity for the smaller network (20 nodes). By comparing with the results achieved by the `XGBoost` model in figure 4.1, we can observe that the shallow network performs very similarly to the `XGBoost` model. In fact, the `XGBoost` model seems to outperform (achieve a higher significance for most combinations) the network ever so slightly.

In figure 4.3b, I present the same grid as described above, but using the larger network. By comparing the results from the smaller network in figure 4.3a, we can discern that the larger network outperforms the smaller network for every single mass combination in the original signal set. The higher the number of nodes and layers, the more parameters need tuning during training, and consequently the more data is needed during training. Generally, the more complex the issue, the more parameters are needed in a model. Therefore, based on the comparison between the two networks, we can draw the conclusion that the data used in this analysis is enough to tune a relatively deep network with large amount of parameters. This result motivated the choice of layers and nodes for all networks used in the sections to come.

4.2.2 Parameter Specific Networks and Interpolation

As I have touched upon in earlier sections, one possible solution to a diverse signal set (in my case a signal set with many potential mass combinations) is to implement one model for each individual signal set. Initially one might assume that although this approach demands more work, it also produces the strongest

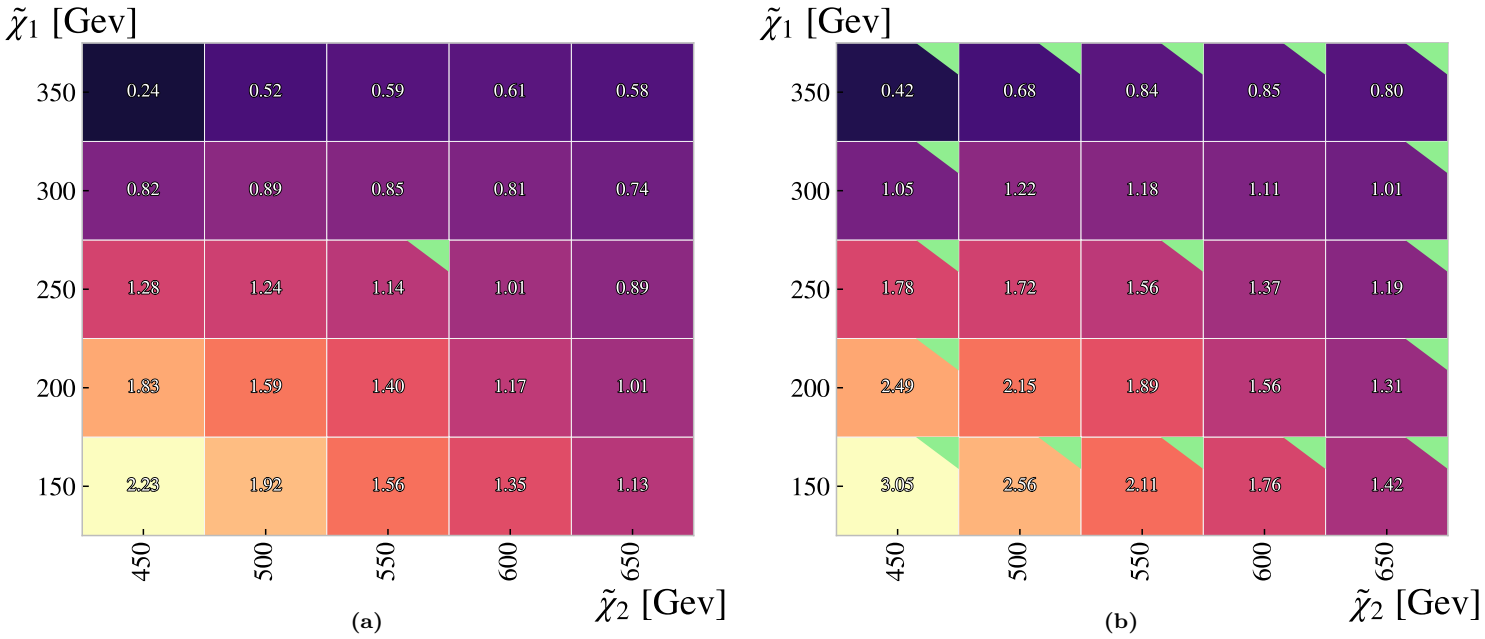


Figure 4.4: Two grids displaying the expected significance on a subset of the full signal set, using the signal region created by two dense NN's. Figure 4.4a presents the results from a model which has only seen one mass combination during training, $\tilde{\chi}_1 = 250$ and $\tilde{\chi}_2 = 550$GeV. Figure 4.4b presents the results from a model which has seen all mass combinations in the grid, but for the inner square of masses around the 250, 555 point. Each figure displays which combinations were used during training with the addition of a green corner.

performances for all mass combinations individually. I have already discussed how, by including different mass combinations in the same data set, we hope to reduce overfitting and potentially allow the model to interpolate between the different mass combinations. However, I have not yet discussed how by moving from a one mass combination signal set to a diverse set would affect the performance on the original signal. In this section I will present the results from an analysis which aims to study this.

In figure 4.4 I present the results from two models for which each is trained on two different sets of signals. Each figure displays which combinations were used during training with the addition of a green corner. The training used to produce the results aligns with the strategy described in section 3.6.2 and both apply an ordinary dense NN architecture described in section 3.5.2. Figure 4.4a presents the achieved sensitivity from a dense NN which has only trained on one mass combination (which will hence force be referred to as One-Mass-Model (OMM)), $\tilde{\chi}_1 = 250$ and $\tilde{\chi}_2 = 550$GeV, while figure 4.4b presents the achieved sensitivity after training on a large set of mass combinations (which will hence force be referred to as Several-Mass-Model (SMM)). Specifically, the latter figure shows the results after training on all signals on the outer square[35] in the figure 4.4b as well as the point in the middle, $\tilde{\chi}_1 = 250$ and $\tilde{\chi}_2 = 550$GeV.

My initial prediction for this comparison was that the OMM would outperform the SMM for the signal with $\tilde{\chi}_1 = 250$ and $\tilde{\chi}_2 = 550$GeV, while underperforming on all other data points. The expectation was that OMM would learn the trends of the only signal it had seen, while the SMM would do the same but for a larger set of mass combinations and at the same time interpolate the results for the masses in between. By comparing figures 4.4a and 4.4b we see that my prediction was not completely true. We can observe that the model which has trained on several-masses outperformed the OMM on every single combination. At first, I believed this to be caused by the OMM overfitting a lot sooner than the other model, therefore being stopped earlier in training by the early stopping criteria described in section 3.6.2. To test this theory, I drew the AUC score made after each epoch on both the training and validation set for the OMM and the SMM, shown in Figure 4.5a and 4.5b, respectively. By comparing the two figures, we observe that the OMM's performance on the validation set peaks in the first epoch, therefore stopping training after 10 epochs, while the SMM peaks after 6, allowing the SMM model to train longer. This is a clear indication that training on one mass combination leads to overfitting.

[35] I.e. all mass combinations which lie on the edges of the grid.

(a) (b)

Figure 4.5: A plot displaying the AUC score made after each epoch on both the training and validation set. Figure 4.5a shows the results from the OMM and figure 4.5b shows the results from the SMM.

Given that the OMM's performance is reduced by the early stopping criteria, I wanted to explore if it would be able to outperform the SMM given it was allowed to train deeper/longer. In figure 4.6, I display the achieved significance of the OMM in the case where early stopping has been removed, and the model was allowed to train for 15 epochs. By comparing figures 4.4b and 4.6, we can observe that even when early stopping is removed from the OMM, it is still outperformed by the SMM. This indicates that overfitting is not the only reason for the OMM's disappointing performance.

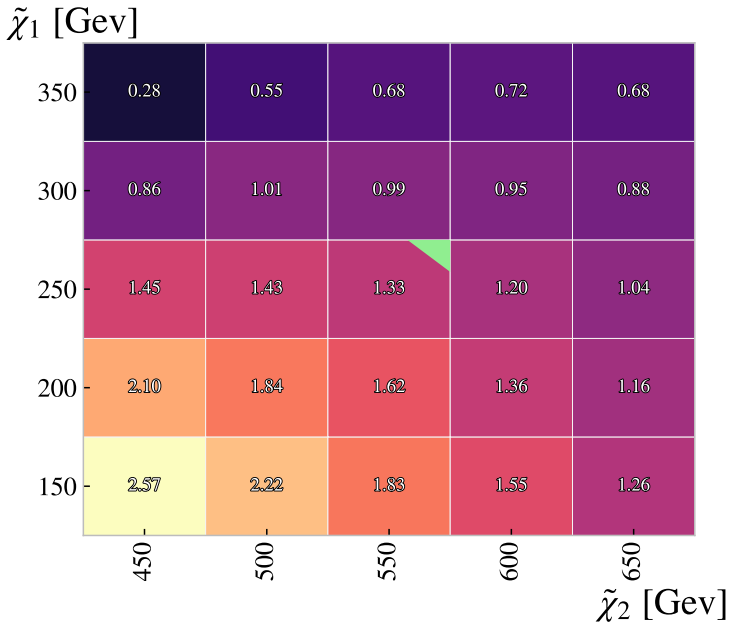
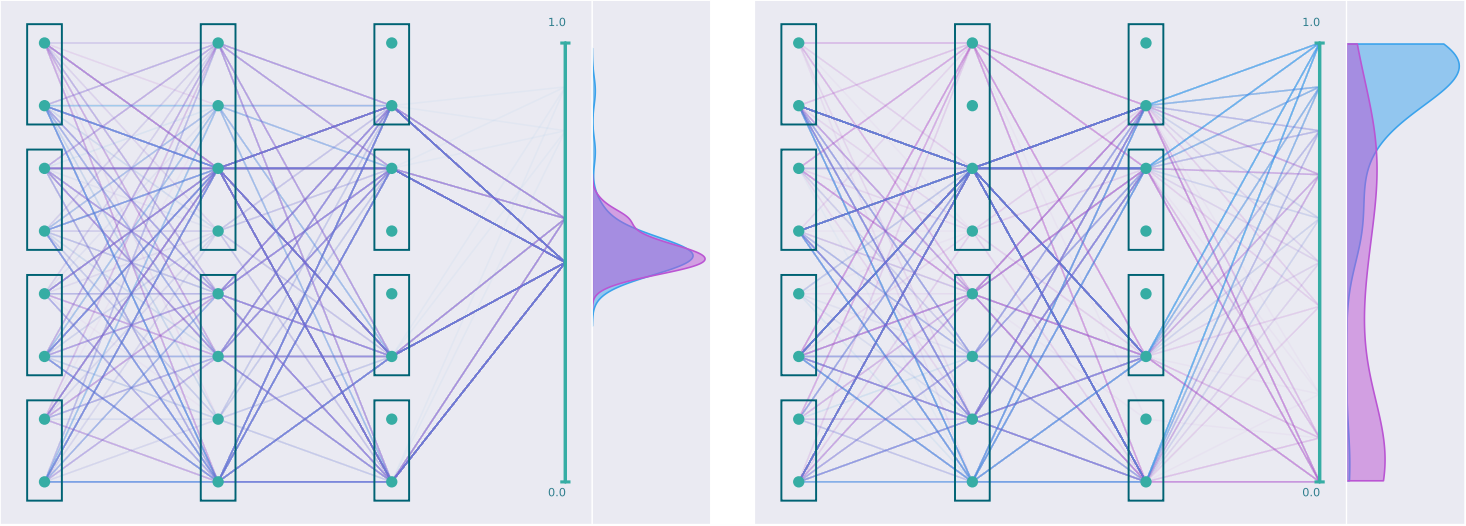


Figure 4.6: A grid displaying the expected significance on a subset of the full signal set, using the signal region created by the NN. The figure presents the results from a model who has only seen one mass combination during training, $\tilde{\chi}_1 = 250$ and $\tilde{\chi}_2 = 550$GeV and was allowed to train for 15 epochs without early stopping.

The most probable explanation for the underwhelming performance of the OMM is that nearby mass combinations (i.e. mass combinations with relatively similar masses) exhibit a lot of overlap in terms of feature trends. In other words, nearby mass combinations (in this example, mass combinations which differ by 100GeV) often contribute to the same type of tuning. This means that by including a larger range of signals, which are similar in mass, we are essentially increasing the amount of data for each individual signal. This gives further motivation to include diversity in the signal set.



(a) (b)

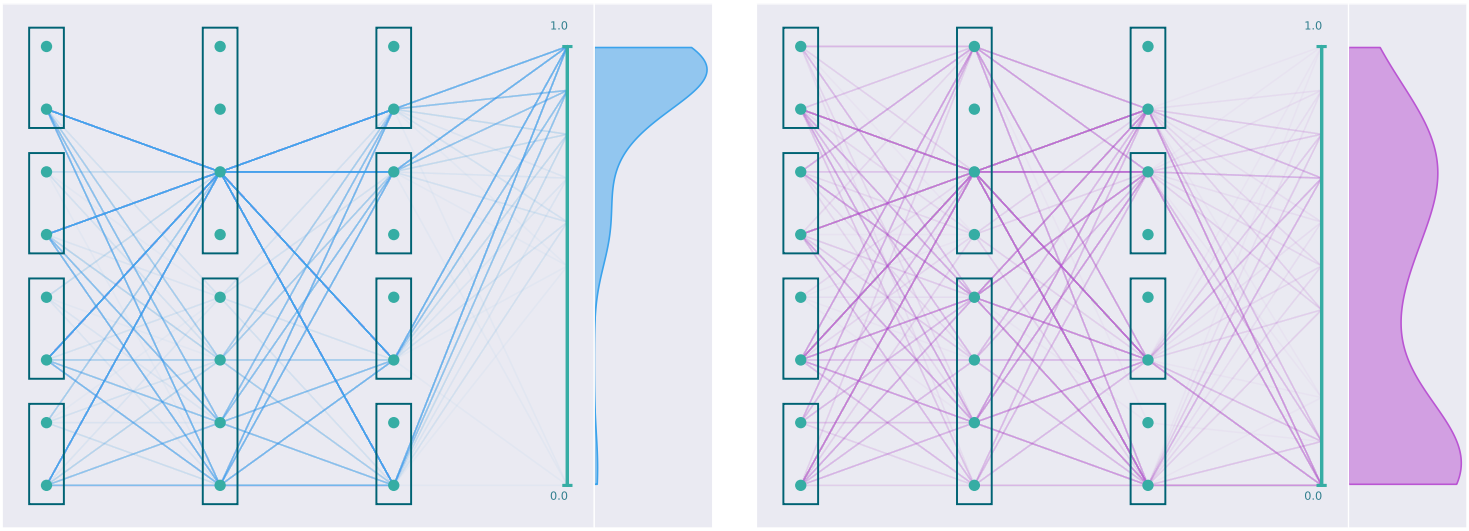
Figure 4.7: A calculated visualization of a three layer maxout network. Each path represents a data point where all connected nodes were the largest activation in their respective unit. The distribution on the far right represent the output distribution. The figure to the left (4.8a) is the result before training and the figure to right (4.8b) is after.

4.3 Ensemble methods

4.3.1 Visualizing Sparse Pathways

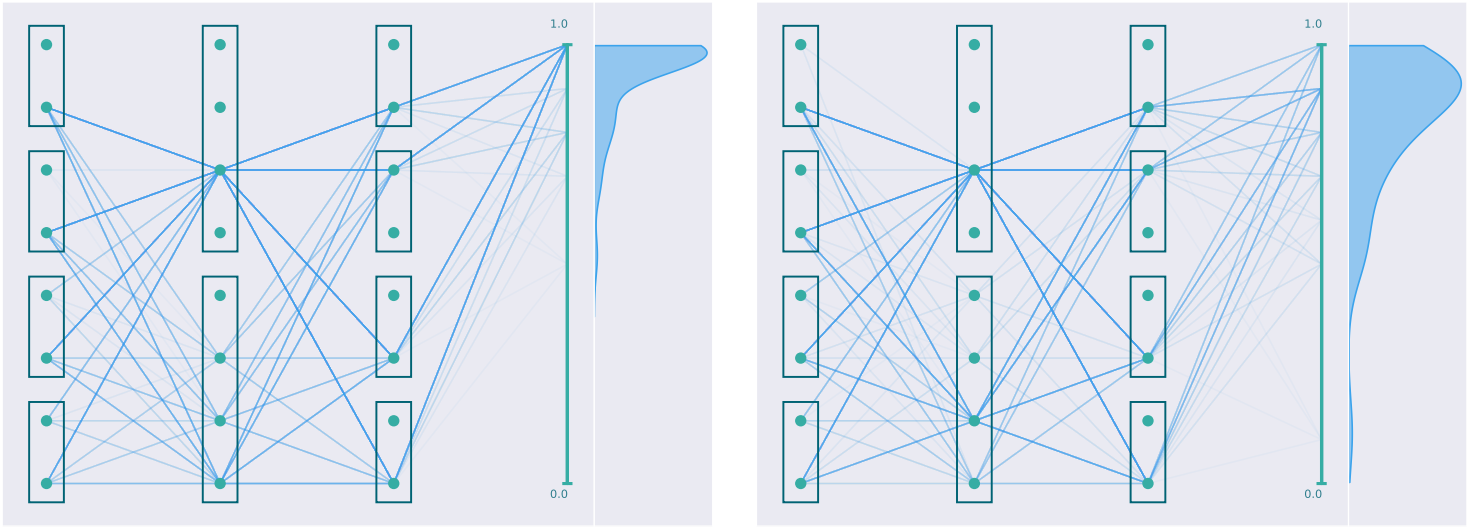
As mentioned in section 3.5.3, the channel-out, SCO and maxout layers applied in this analysis were created by me using the `TensorFlow` API. As such, I found it imperative to make sure that the layers worked as intended. To do this I created a small network with three layers, with eight nodes each, all applying maxout layers with four, two and four units respectively. In this section I will dissect the activations of said network before and after training.
In figure 4.7a, I have plotted the activation of 100 randomly sampled events, 50 background and 50 signal for an untrained model. Adjacent, I plotted the resulting distribution of the output. From the figure we observe little to no deviation between the activation from the signal and the background. This is mirrored in the distribution of the output which is centered around the middle of the range. This result is as expected, given that an untrained model holds no knowledge of the data and simply applies a random set of weights. A small exception from this result can be found for larger output values, where we observe a small distribution of signal values. This is due to the signal having some inherit differences to the background (for example high E_t^{miss}).
In figure 4.7b, I plotted a similar plot as described above, but using a trained model. In this figure the output is far more separated, and we see noticeable differences in the activation of nodes. To highlight the difference in activation, I drew two new figures where the signal (4.8a) and background (4.8b) were drawn individually. In figure 4.8 we notice that there is a noticeable variation in the activation for both signal and background. The differences are highlighted in the different paths through network, displaying that the model is able to differentiate between signal and background. Most noticeably, the two units in the middle hidden layer highlight this fact. In the case of the signal, the upper unit clearly favors the second bottom node. For the background this is also partly true, but with far more spread in the other nodes. Similarly, in the bottom unit (in the same layer), the background shows large activation in the uppermost node, while the signal data does not. To conclude, the maxout layer does indeed find trend specific paths through the network which will aid in separating the output for the signal and the background.

Ideally, we would want the model to not only be able to separate the signal from the background, but do so in a way which allows the network to store the different trends within the signal. In section 2.6.5, I described this ability as long-term memory. One way to implement long-term memory is through a high number of parameters. However through the LWTA layers we can achieve the same through specific paths. To study this I have created similar figures as discussed in the paragraphs above, but this time only including



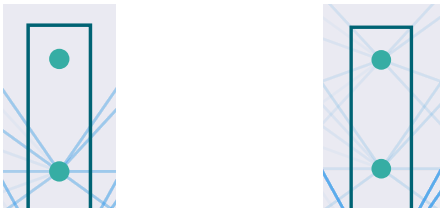
(a) **(b)**

Figure 4.8: A calculated visualization of a three layer maxout network. The lines represent the path through the nodes with the largest activation in their respective unit. The bolder the line the more frequently the path is used. The distribution on the far right represent the output distribution and the figure with blue paths (left) 4.8a is the result of signal, and the pink paths (right) is a result from background 4.8b.



(a) **(b)**

Figure 4.9: A calculated visualization of a trained maxout network with three hidden layers. The lines represent the path through the nodes with the largest activation in their respective unit. The bolder the line the more frequently the path is used. The distribution on the far right represent the output distribution. The figure to the left (4.9a) is a result of signal with ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) and the right (4.9b) ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV).

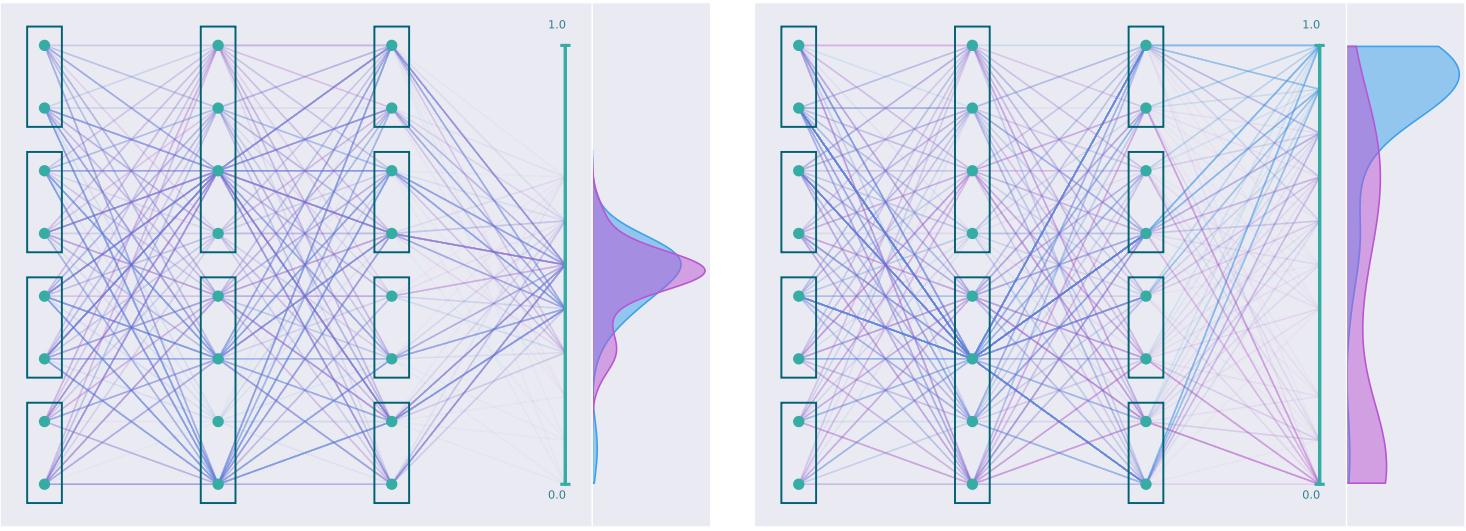


(a) (b)

Figure 4.10: A cut-out of the fifth and sixth node (counting from the top) in the second hidden layer, activated by the signal with ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) 4.10a and ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) 4.10b.

results from applying the network on one mass combination. Figures 4.9a and 4.9b present the results for the mass combinations $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV, and $\tilde{\chi}_1 = 200$ and $\tilde{\chi}_2 = 300$GeV respectively. By comparing the figures, we see a small but noticeable difference in paths. Again the middle hidden layer seems to be the differentiating factor. To highlight the differences, I have included a cutout, comparing the first and second node in the bottom unit for the middle layer of both figures, 4.10a and 4.10b. By studying the cutouts, we can discern that the maxout layer allows the model to discriminate both background from signal, and different variations of the signal through an increase in long-term memory. We can conclude that the maxout layer not only applies a form of regularization, but also increases the long-term memory of the model through trend specific paths through the network.

Finally, I wanted to compare the activations of the maxout model, to the activation of the SCO model. In section 2.6.5, I stated that the inspiration behind the SCO was to elevate the channel-out method in a way which reduced complex co-adaptation among neighboring nodes. In figure 4.11, I included a visualization of the activation of a three layer SCO network before (4.11a) and after (4.11b) training. By comparing the activations from the SCO layers to maxout, we can discern that the SCO behaves exactly as intended. In the figures visualizing the maxout layer (4.8), we can observe that several of the nodes which are not activated before training, are left dormant even after training. In comparison, the SCO layers display a far more balanced activation, and exhibits no signs of complex co-adaptation.



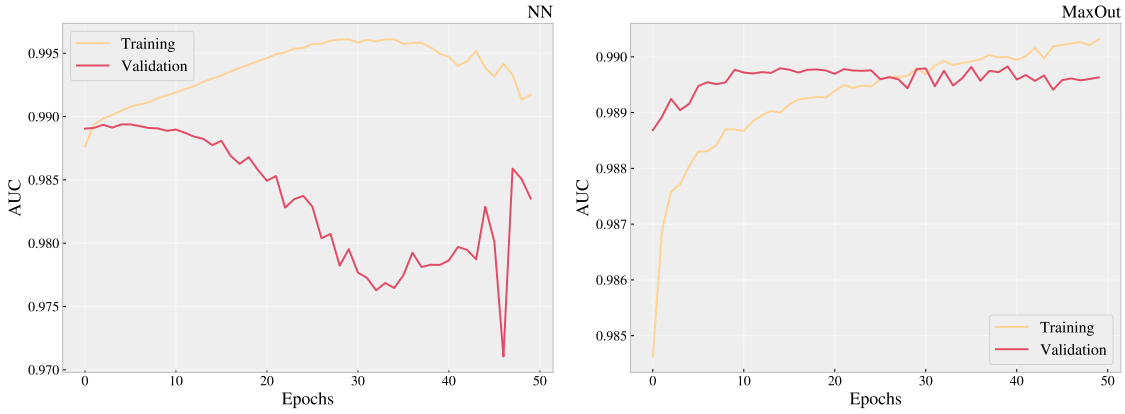
(a) (b)

Figure 4.11: A calculated visualization of a three layer SCO network. The lines represent the path through the nodes with the largest activation in their respective unit. The bolder the line the more frequently the path is used. The distribution on the far right represent the output distribution. The figure to the left (4.11a) is the result before training and the figure to right (4.11b) is after.

4.3.2 Training History and Overfitting

In section 2.5 I described how creating ensembles of networks is a form of regularization. Therefore, it is of interest to study the relationship between performance on the training set and the validation set for our ensemble methods. In figure 4.12 I drew plots displaying the performance on the training and validation

scores after each epoch (50 in total), as measured in AUC for both a dense NN (4.12a) and maxout (4.12b). In figure 4.12a we can observe that a deep, dense NN reaches a maximum in performance for the validation set after only a couple of epochs. This peak is then followed by a quick drop in performance, while the training set increases in performance. The drop in performance for the validation set and increase in performance for the training set is a sure sign of overfitting.
In figure 4.12b we observe that an ensemble method (in this case the maxout) displays a different trend. For the first 10 epochs the model increases in performance for both data sets. After this, the validation set does not decrease in performance, but simply holds stable while the training set increases. This is a sign that the model is not experiencing overfitting. Furthermore, the training history of the maxout model raises an interesting point. From experience I know that the maxout model reaches a peak in performance on the validation set after approximately 15-20 epochs. Given that the performance after this is stable, it is plausible that continuing training will not worsen the performance or lead to overfitting, but rather improve. This is definitely an interesting possibility for the LWTA layers, but will not be studied in this thesis due to time constraints.
In section 3.6.2, I discussed the training strategy used when training all models in this analysis. I mentioned the implementation of an early stopping criterion which makes sure that the model continues to train only as long as the performance on the validation data set increases. By studying the subfigures in figure 4.12, we can deduce that the ensemble methods will not only be able to avoid overfitting, but will as a consequence be allowed to train much longer than the other models.



(a) (b)

Figure 4.12: A plot displaying the AUC score made after each epoch on both the training and validation set. Figure 4.12a shows the results from the dense NN and figure 4.12b shows the results from a maxout network.

4.3.3 Comparing Achieved Sensitivity between Ensemble Methods

In this section I will present and discuss the performance of the three different networks discussed in section 2.5.2, channel-out, SCO and maxout. The results presented in this section were made using the original signal set (see section 3.1.2), the training strategy described in section 3.6.2 and the architectures described in section 3.5.2.
In figure 4.13 I present the achieved sensitivity of the maxout model using the original signal set. The grid shows the same trends as the previous models, i.e. the preference in the higher statistics mass combinations. By comparing the results from the maxout model to the deep, dense network presented in figure 4.3b, we discern that the dense network outperforms the maxout model (ever so slightly) for the higher statistic mass combinations. It is plausible that this is due to the difference in depth. The dense network utilizes 3 hidden layers of 600 nodes each, while the maxout, although built with the same architecture, only utilizes 200 nodes per layer when propagating input through the network. This could result in the dense network being able to more deeply tune for the trends in the higher statistics combinations.
However, the most interesting result for the maxout model, is its ability to tune for all mass combinations. Although the dense network outperformed the maxout model for the high statistics combinations, the maxout model outperformed the dense for most other combination (26 of the 30 possible mass points). This result can be credited to two factors. The first being maxout's effect as a form of regularization. In the previous section (see section 4.3.2), I presented how, maxout, through its regularization abilities, is able to uphold the early stopping criteria for a larger number of epochs. The second factor is maxout's innate long-term memory which was studied in section 4.3.1. From figure 4.13, we are able to deduce that although the

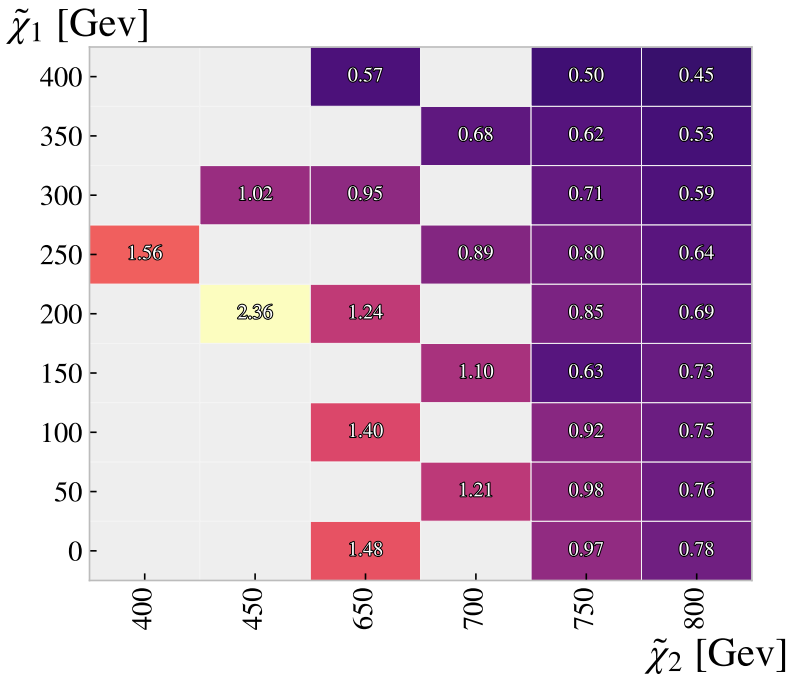


Figure 4.13: A grid displaying the expected significance on the original signal set, using the signal region created by the maxout network.

maxout model is not able to tune to the same depth for the lower masses, it is able to achieve a large level of generalizability through reducing overfitting and increasing long-term memory.

In appendix A.1 I have included grids displaying the achieved sensitivity for both channel-out and SCO. Both models demonstrate similar performance to the maxout layer. To compare the three methods I created a 'pie-plot'. A 'pie-plot' compares the achieved sensitivity between several models and displays the results for each individual mass combination. In figure 4.14 I present the pie-plot comparing maxout, channel-out and SCO. Each mass combination includes a pie, where the size of each 'slice' represents the relative size of the significance compared to the other methods. For example, if a slice occupies half of the pie, then the method corresponding to that slice achieved a significance equal to the sum of the significance of the other methods. The color surrounding each pie marks which method achieved the highest sensitivity for the respective combination.

By studying the pie-plot in figure 4.14, we can deduce that maxout model outperforms the other two in most of the mass combination (24 out of the 30 mass points). From the sizes of each slice, we can deduce that all three models seem relatively equal in performance, maxout only outperforming the others by a small fraction. The most interesting observation from the 'pie-plot' is the performance from the SCO, as this model was in-part created by me. Except the mass combinations where maxout was the most sensitive, SCO was the best performing model. Most interestingly, it outperformed the channel-out model, which is the model most similar to SCO, in 9 out of the 30 mass point (see figure 27). In hindsight, I believe that by removing the SCO during prediction (similar to what is done for dropout), the SCO layer would greatly approve in performance, on a count of the fact that the performance on each event is dependent on the random choice of unit for that prediction. This means it is possible that upon prediction, a data point is sent through a path which has never been chosen during training. Nonetheless, this analysis is an indicator that although the maxout model was the highest performing model, the SCO layer shows great promise and should be further explored in further analysis.

4.4 Parametrized Neural Network

4.4.1 Discriminating Masses

When introducing the PNN in section 2.6.6, I mentioned that by including the masses of the introduced particles in the feature set, we motivate and individualistic tuning for each mass combination in the signal set. To test if this is indeed what happens, I will in this section present results where I manually assign different mass combinations the same parameter. The hope is that if indeed the PNN has been able to tune

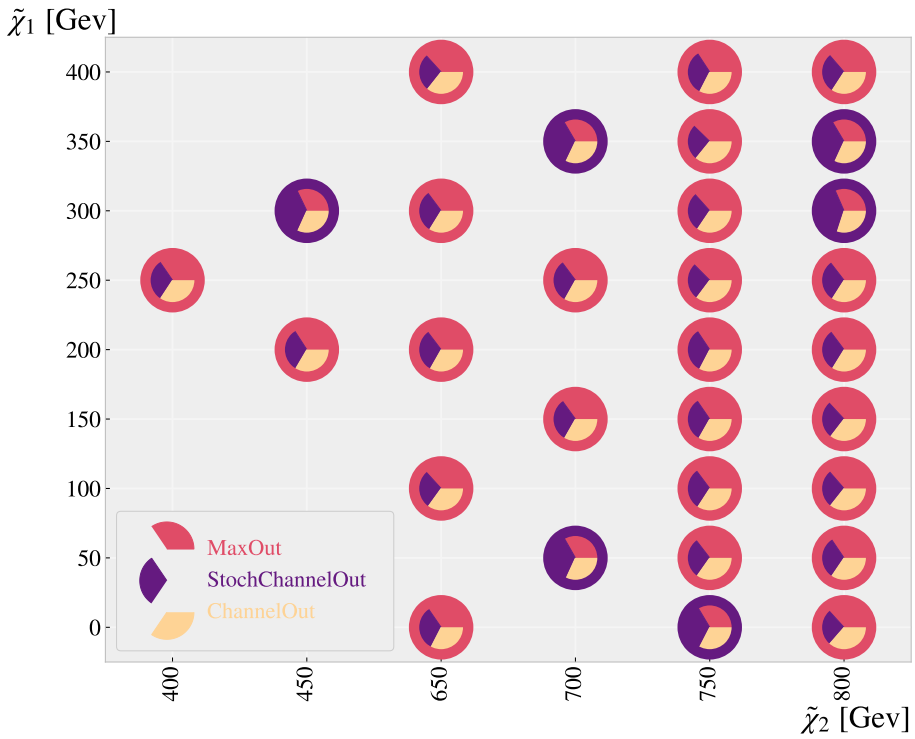


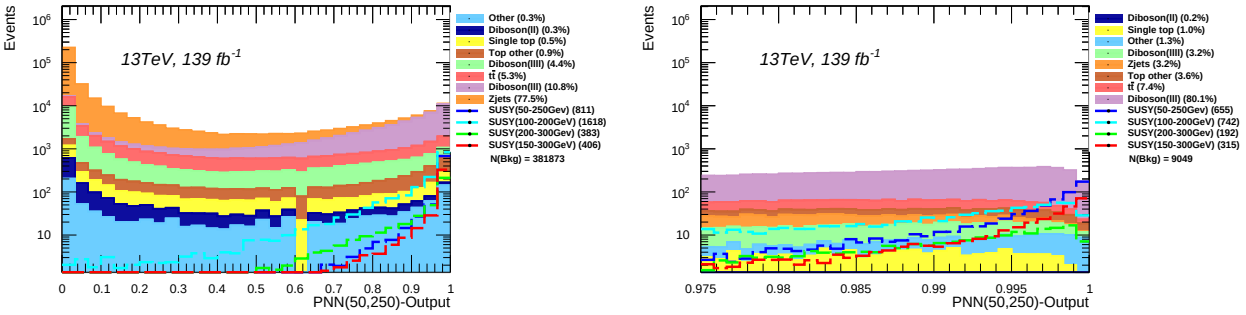
Figure 4.14: A sensitivity comparison between the ensemble networks (maxout, SCO, channel-Out) on the original signal data. The size of each slice represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

individually for each mass combination, then the PNN should perform best when the events are correctly labeled compared to when they are not.
In figure 4.15a I have drawn the distribution of the output from the trained PNN architecture (see section 3.5.2). The model was trained using the complete signal grid. In figure 4.15a, 4 signals have been included; $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV, $\tilde{\chi}_1 = 100$ and $\tilde{\chi}_2 = 200$GeV, $\tilde{\chi}_1 = 200$ and $\tilde{\chi}_2 = 300$GeV, and $\tilde{\chi}_1 = 150$ and $\tilde{\chi}_2 = 250$GeV. All data, including both signal and background were given the labels of 50 and 250 (corresponding to the masses $\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV). In figure 4.15a the full output range is included, whereas in figure 4.15b only the output in the range $[0.975 - 1]$ has been included. In both figures, it is evident that all mass combinations have been effectively separated from the background, even for the combinations which are given the wrong parameter. Nevertheless, upon close study of the figures and the corresponding legends, we can deduce that the signal which is given the correct label is also the signal which has the highest percentage of conserved events when applying a simple cut off 0.975[36].

In section 4.1 I presented results indicating that some signals were easier to separate from background than others. This leads me to the question; did the PNN perform better on the signal with $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV because of the labeling, or simply because this particular signal was easier to separate. To answer this question I conducted a second test. In figure 4.16 I repeated the analysis described in the paragraphs above, but where all the signals were given the label $\tilde{\chi}_1 = 200$ and $\tilde{\chi}_2 = 300$GeV. Similarly to the previous results, I drew the output for the full output range (4.16a) and after a cutoff off 0.975 (4.15b). Via the examination of the two figures, we can indeed see that my assumption was right. Despite the fact that events with masses equal to $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV are given the wrong parameter, the PNN achieves the highest efficiency when predicting on the aforementioned events.

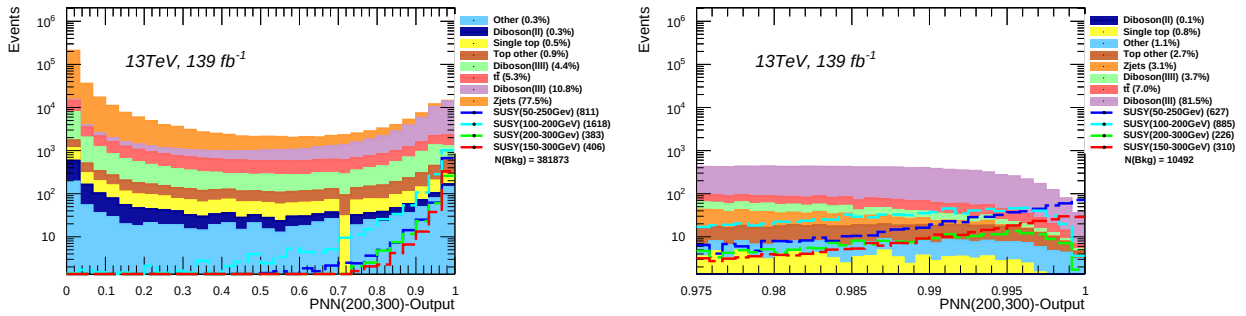
To further study this result, I created a table of the efficiencies of each mass combination for both results, after applying a simple cut of 0.975. The results are presented in table 4.1. It is evident from table 4.1 that the PNN achieves the highest performance when events are given the correct label, although not by much. In the events with masses $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV, the efficiency improved by a little over 3%, while the event with masses $\tilde{\chi}_1 = 200$ and $\tilde{\chi}_2 = 300$GeV, improved efficiency by almost 10%. Another interesting observation is that events with ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV) seem to prefer the label of ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) over that of ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV), despite the latter mass combination being closer in mass. A possible explanation is that the difference in mass ($\Delta m = |m_{\tilde{\chi}_1} - m_{\tilde{\chi}_2}|$), influences the trends in the data, similarly

[36]This is further evident when studying the efficiency rates in table 4.1.



(a) **(b)**

Figure 4.15: The output distribution from a trained PNN model for the background and signals with four different mass combinations: ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV), ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV), ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) and ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV), and where all the data were given the parameter features of ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV). The figure includes the full output range (4.15a) and the output ranging from 0.975-1.00 (4.15b). The number in parentheses indicate the fraction of each background and the absolute number of events for each signal point. The total number of background events is also shown.



(a) **(b)**

Figure 4.16: The output distribution from a trained PNN model for the background and signals with four different mass combinations: ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV), ($\tilde{\chi}_1 = 100$, $\tilde{\chi}_2 = 200$GeV), ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) and ($\tilde{\chi}_1 = 150$, $\tilde{\chi}_2 = 250$GeV), and where all the data were given the parameter features of ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV). The figure includes the full output range (4.15a) and the output ranging from 0.975-1.00 (4.15b). The number in parentheses indicate the fraction of each background and the absolute number of events for each signal point. The total number of background events is also shown.

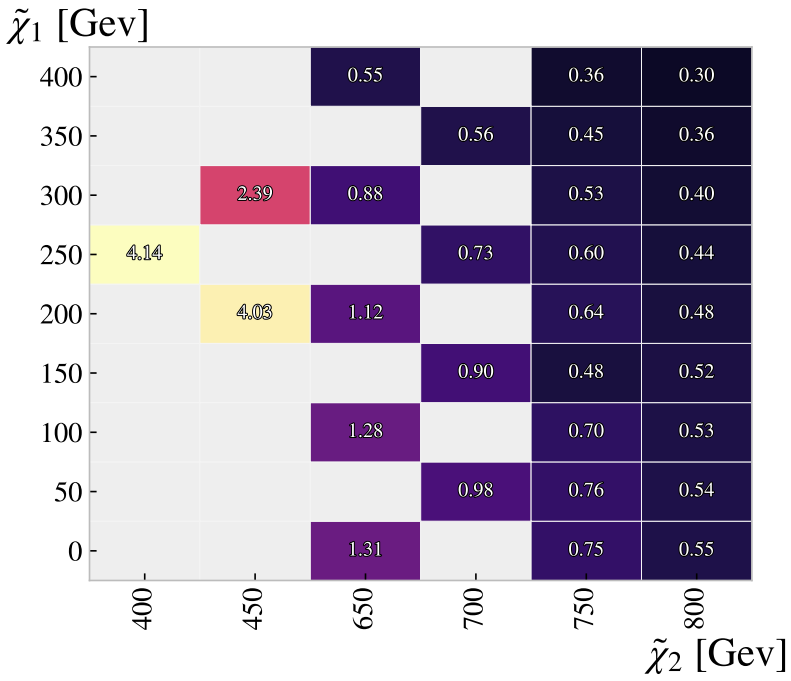


Figure 4.17: A grid displaying the expected significance on the original signal set, using the signal region created by the PNN network.

to what we saw that size of mass does. Regardless, from 4.1, we can deduce that the PNN does in fact discriminate between mass combinations and tune for them independently.

Label \ Channel	$(50, 250)$	$(100, 200)$	$(150, 300)$	$(200, 300)$
$(50, 250)$	**80.8**%	45.8%	**77.5**%	50.1%
$(200, 300)$	77.3%	**54.6**%	76.3%	**59.0**%

Table 4.1: A listing of the remaining procentage of each mass combination in the output range 0.975-1.00 using the labels ($\tilde{\chi}_1 = 50$, $\tilde{\chi}_2 = 250$GeV) and ($\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 300$GeV) respectively.

4.4.2 Sensitivity Result

In this section I will present the achieved sensitivity by the PNN on the original signal set. In figure 4.17, I present a grid displaying the sensitivity of the PNN on the original signal set. Similarly to the previous models', the figure indicates that the PNN is able to achieve a much higher sensitivity on the lower masses. The most notable difference to the previous models, is the span of values for the significance. The PNN almost doubles the highest significance achieved by any previous model (from 2.44 to 4.14), and simultaneously achieves the lowest significance (now 0.30). The most probable explanation to this result is the distribution of parameters for the background. In section 2.6.6, I described how the background is randomly assigned parameters such that the percentage of background with a given parameter is equal to the percentage of the signal with the same parameter. In other words, the higher the statistics for a given mass combination, the higher the amount of background will be 'linked' to it. Inn the same section I also described the hope that the parameters would shift the output from the initial layer in a way that motivates individualistic training. If this is true[37] it means that masses with larger statistics, are given larger amounts of background to train on. This could explain the uneven performance by the PNN.

In this analysis I choose to follow the methodology described in the article by Baldi et al. [8], but other variants of the PNN could be of interest in future studies. An alternative to the current setup for distributing parameters to the background, is to distribute the parameters evenly. In other words, for N different mass combinations, each set of parameters will be distributed to 1/N parts of the background. This would most likely produce a more balanced result. I considered using an even approach, but found that the current setup was more aligned with the rest of the analysis where a one-one, signal- background ratio approach has

[37] For a network of this depth, it is very hard to explain what happens during training, which explains the passive statement.

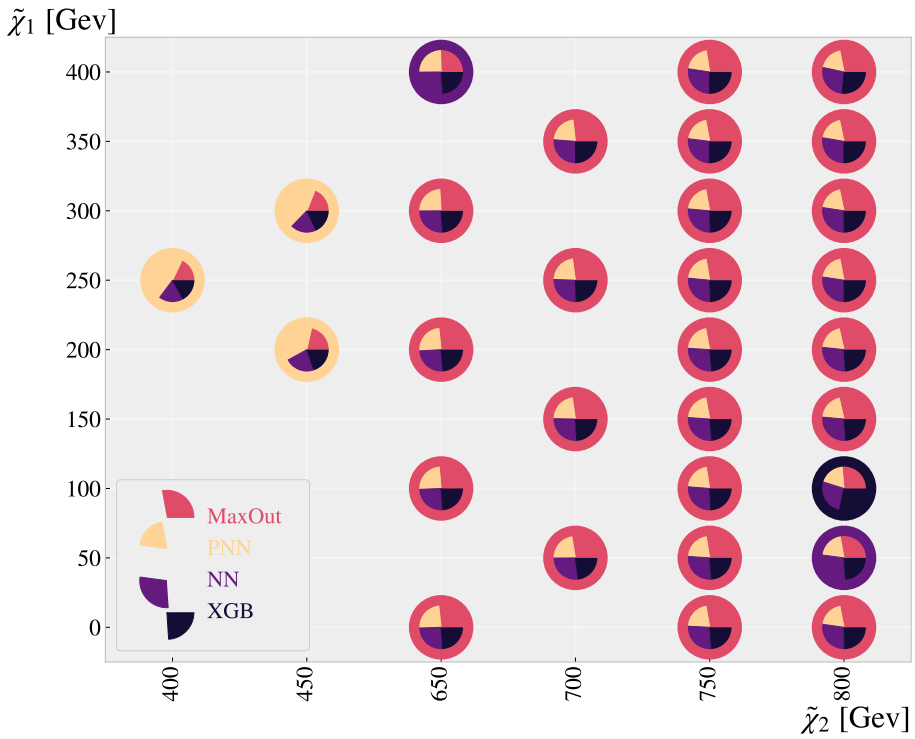


Figure 4.18: A sensitivity comparison between a dense NN, PNN, maxout and XGBoost on the original signal data. The size of each slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

been made[38]. The more interesting alternative would be to assign the background most similar to a given mass combination[39], the same parameters. This would focus the training of the PNN on the set of events which should be hardest to separate. One approach to determine which events are deemed 'most similar' to a combination, is to apply a prior ML analysis with a simple BDT or dense NN. Due to time constraints of this thesis, I decided not to test this approach, but would be an interesting area to explore.

4.5 Remarks on Comparison between Models on Original Signal Set

So far in the thesis I have presented results from two branches of ML, Boosted Decision Trees (BDT) and Neural Network (NN). In the case of the latter, I tested three further variations, a dense NN, ensemble networks and PNN. In figure 4.18 I have drawn a pie-plot comparing the sensitivity on the original signal set for the four different models. By assessing figure 4.18, we can deduce that the maxout network outperforms all other models in most mass combinations (24 out of 30 mass points), although not by much. Additionally, we can discern that the PNN is far more sensitive than the other models for higher statistics mass combinations, and underperforms for low statistic mass combinations. This indicates that the PNN is able to train much deeper than the maxout model, but struggles to uphold performance for a diverse signal set. The long-term memory of the maxout layer, allows the model to train relatively evenly for all trends in the signal set, but at the cost of never training to the same depth as the PNN
On the other hand, the `XGBoost` model is outperformed in all but one combination, implying that the networks are more ideal for this analysis. It is worth noting however, that not at a lot of time was spent improving the `XGBoost` model. While for the networks I experimented with different layers and tested for different architectures, no effort was put in deciding the architecture of the `XGBoost`. Likewise, the solution to the negative weights (see section 3.7) could by all means be optimized if given more attention. Therefore, it is worth saying that the results in this analysis do not imply that deep networks are better analysis tool for BSM searches than BDT, but rather exemplify the advanced capabilities of deep networks.

[38] In section 3.6.1 I discuss how I have chosen to scale the weights such that sum of the weights of the background is equal to the sum of the weights of the signal.
[39] By most similar I refer to the background with the largest overlap in the feature space to the signal.

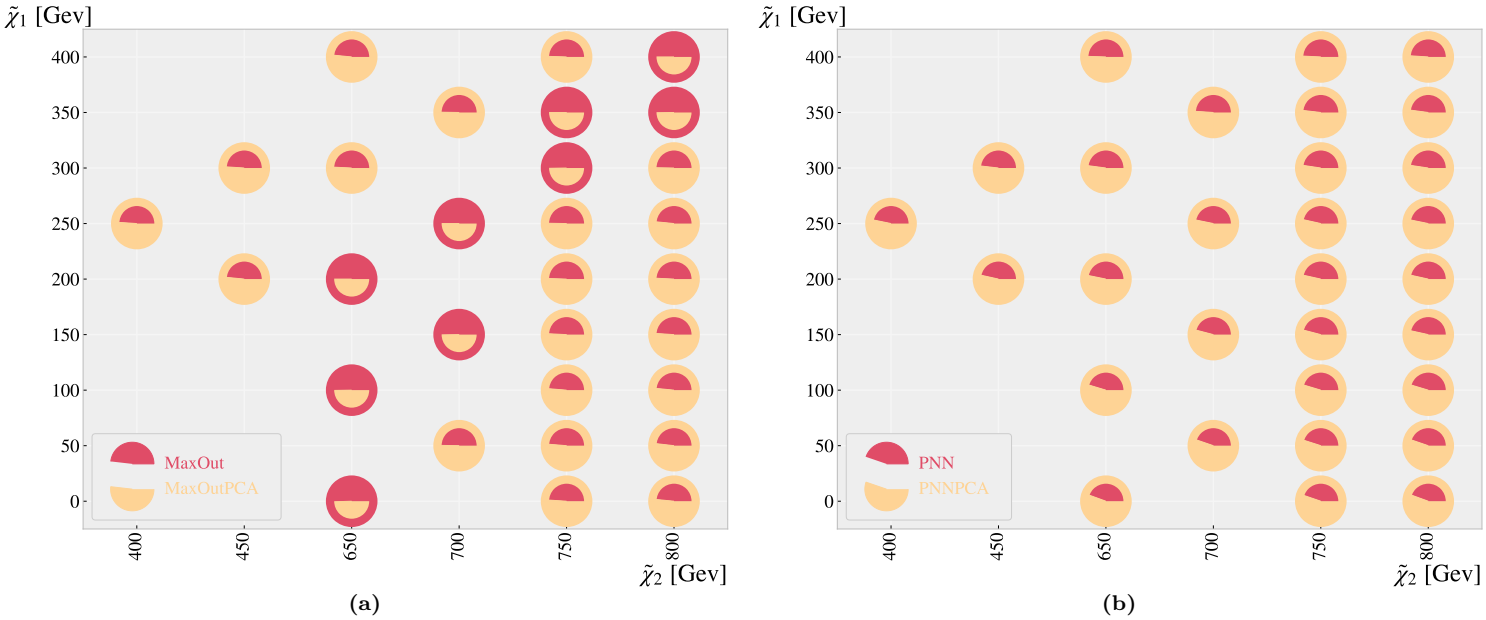


Figure 4.19: Two 'pie-plots' comparing sensitivity on the original signal set, where each figure shows the comparison between a model trained on data with and without a PCA. Figure 4.19a displays the comparison for the maxout model, and likewise figure 4.19b for the PNN. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

4.6 Increasing Sensitivity through PCA

So far in the analysis, all features have been given an equal weighting in the beginning of training, But, as mentioned in section 2.8, not all features contribute to an effective signal region. In section 2.4.2, I explained how through the use of PCA, we are able to create a new set of features which can be ordered by amount of variance. In this way, we can reduce the dimensionality of the data set, while at the same time preserving most of the variance. In this section I will present the results of training on a data set which has gone through such a PCA.

In this analysis I performed a PCA on the data set, and demanded that 99.99% of the variance should be preserved. With a threshold of 99.99%, five features were removed. In appendix A.2, I have included the grids displaying the new sensitivity for the original signal set, using data which has gone through a PCA. In figure 4.19, I have included two 'pie-plots' comparing the sensitivity of the maxout (4.19a) and the PNN (4.19b) with and without a PCA.

In figure 4.19a, we can observe a relatively similar behavior with and without the PCA. There seems to be no obvious trends in regard to improvement or decrease in performance for any of the mass combinations. But, to simplify the comparison between the two, we can treat all mass combinations equally. With this in mind, maxout achieves a higher sensitivity *with* a PCA in 21/30 combinations, indicating that PCA indeed increased performance. Similarly, the PCA increased performance for the PNN, but contrary to maxout it did so for every combination in the original signal set. Additionally, through closer study of the ratios for each 'pie', the PCA not only increased a larger number of combinations for the PNN, but also increased the performance by a larger degree. Through a similar analysis, I deemed the PCA to decrease performance for the NN, only improving 11/30[40].

To compare all three models discussed above with the PCA included, I created a 'pie-plot' which is displayed in figure 4.20. By comparing the 'pie-plots' made with and without PCA (see figure 4.18), we notice that maxout now outperforms the dense NN in combinations previously preferred by the dense NN. Similarly, the PNN now outperforms the maxout in three new combinations. By chance, all three of said combinations were combinations where the maxout model preferred no PCA. This is yet another indication that building models to be sensitive to a large range of signals can sometime lead to changes which improve performance for some data, and are destructive for others. Like I mentioned in the paragraph above, I decided to treat all combinations equally, but with the previous point in mind, weighting the importance of certain combinations based on scientific promise/interest could be a better approach.

[40]The 'pie-plot' comparing the dense NN with and without a PCA is included in the appendix A.2.

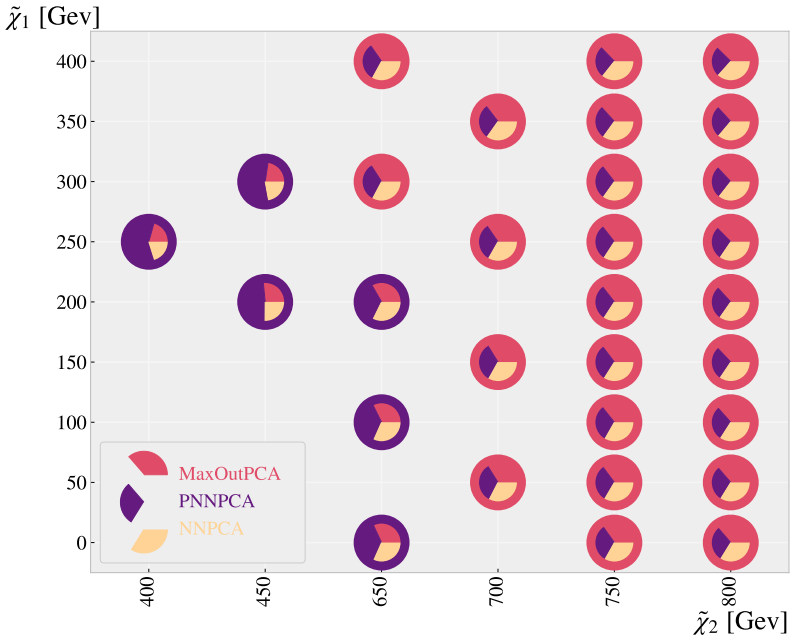


Figure 4.20: A sensitivity comparison between a dense NN, PNN and maxout on the original signal grid. A PCA analysis has been applied to the data being utilized in this result. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

4.7 Comparing Models on the Complete Signal Grid

So far in the analysis I have only tested on a subset of the signal grid, which I have called the original signal grid. In this section I will extend my search to include the complete signal grid displayed in figure 3.1. The full grid consists of 89 mass combinations compared to the original 30, and extends the mass ranges to $\tilde{\chi}_1 \in [0-400]$GeV and $\tilde{\chi}_2 \in [200-800]$GeV. In the figures to come, I have included a turquoise band around all points for which a significance of more than 1.64 is expected (see section 2.9.2). When comparing the results, we are not only interested in how sensitive the models are for each combination individually, but also for how many points we are able to achieve a sensitivity over 1.64. However, similarly to previous results, the significance does not include any uncertainty.

I will not apply all previously tested models to the complete signal grid. Instead, I will only apply the models I found most ideal for this analysis, based on the tests performed in the previous sections. I decided to choose one model from each 'type'[41] of network. Based on the results in section 4.3, where I compared the different ensemble methods, I found the maxout model to be the top performer. Likewise, in section 4.6, I found that both maxout and the PNN preferred to utilize data with PCA. Therefore, I will utilize the maxout model and PNN model defined in section 3.5.2 with the use of PCA. Finally, I will include the ordinary dense NN for the sake of diversity, but without a PCA, as this was found to be preferred.

In figure 4.21 I have drawn a grid displaying the sensitivity of an ordinary dense NN, on the complete signal grid. Again, we observe that higher statistics mass combinations, result in a higher significance. Additionally, the dense NN was able to achieve a sufficient significance for over 38 mass combinations, all between the ranges of $\tilde{\chi}_1 \in [0-250]$GeV and $\tilde{\chi}_2 \in [200-600]$GeV. What is even more interesting, is that by comparing the results on the full set with the results on the original set (see figure 4.3b), we notice that the network was able to improve its sensitivity on every single mass combination from the original signal set. This is yet another indication that the deep networks are able to exploit overlapping regions in the feature space between nearby combinations.

Figure 4.22 displays a grid presenting the sensitivity of the maxout model using the complete signal grid. The data used to train this model, has been through a PCA, and the architecture is the same as described in section 3.5.2. Similarly to the dense NN, by including the complete grid, the maxout model improved performance on all combinations in the original set. Also, the maxout model upholds the sensitivity criteria (> 1.64) for the same mass combinations. However, contrary to the tests performed on the original signal set, the dense network outperforms the maxout model for most of the signals points (74 out of 89 mass

[41] By network type, I am referring to either the ordinary dense network, the PNN or a LWTA model.

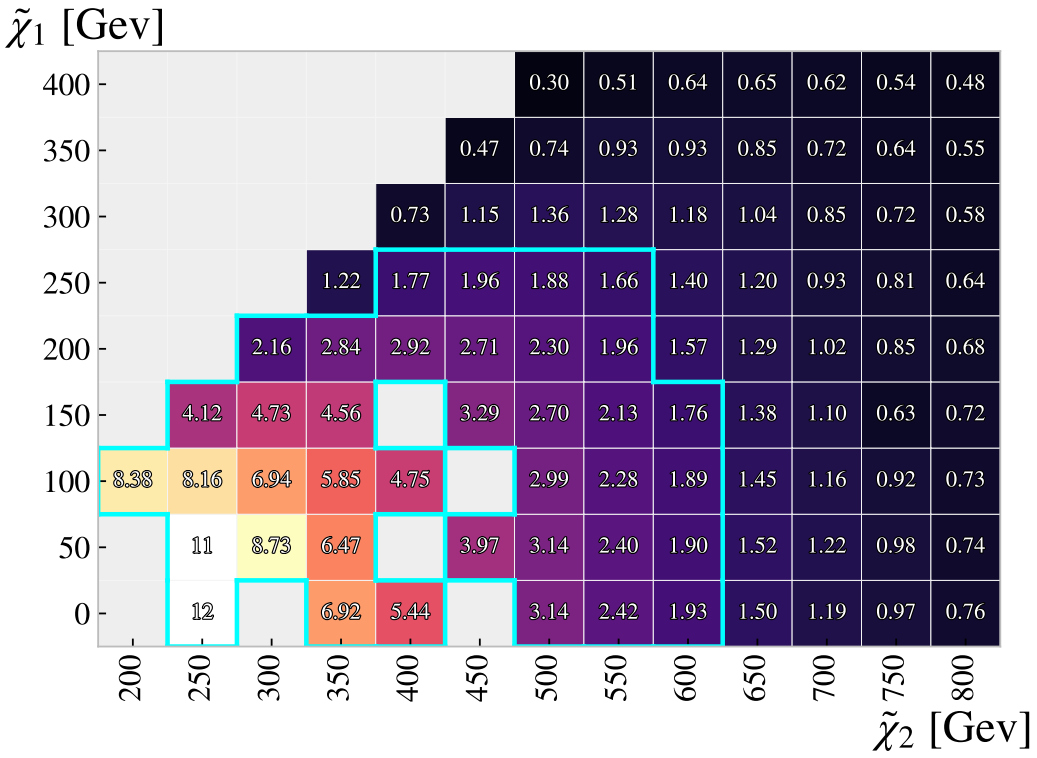


Figure 4.21: A grid displaying the expected significance on the complete signal grid, using the signal region created by the NN network. A band around each cell with a significance over 1.64 has been included.

combinations) when using the full signal grid. A possible explanation to the shift in performance could be that the dense NN lacked the statistics when training with the original signal set. When the complete signal grid is included, the dense NN (which is deeper than the maxout model) is able to train much deeper, which in tern, increases sensitivity. However, the maxout model outperforms the dense network for the low statistics combinations, again indicating the maxout layers ability to increase long-term memory.

Finally, I applied the PNN to the complete signal grid. The results are found in the grid presented in figure 4.23. Similarly to the tests performed with the original signal set, the PNN is able to achieve an incredible sensitivity for the high statistics mass combinations ($\tilde{\chi}_2 < 400$GeV). For the combinations with the highest statistics ($\tilde{\chi}_2 < 300$GeV), the PNN more than doubles the expected significance of the previous two methods. For the lower statistics combinations, the PNN drops in performance. Moreover, the performance of the PNN on the combinations with the highest statistics in the original signal set ($\tilde{\chi}_1 = 250$, $\tilde{\chi}_2 = 400$GeV and $\tilde{\chi}_1 = 200$, $\tilde{\chi}_2 = 450$GeV), has decreased by more than half. This is an indication that including further trends in the data was destructive for training. In other words, although the PNN achieves impressive sensitivity towards high statistics mass combinations, it suffers from the lack of long-term memory which allows the maxout model to uphold performance for lower statics combinations. In the appendix (A.3) I have included two pie-plots comparing the performance of the maxout model and PNN (figures 31 and 32 respectively) on the original signal set using models trained on the original and the complete signal grid, respectively. The 'pie-plots' further exemplify how the maxout model is able to exploit the full statistics to improve on most mass combinations (17/30), compared to the PNN where the performance drops for most combinations (28/30). Finally, in comparison to the ordinary dense network and the maxout model which both achieved sufficient sensitivity for 38/89 combinations, the PNN only did so for 33/89.

To summarize the comparisons on the complete signal grid, I created a 'pie-plot' in figure 4.24. As shown in the figure, the ordinary dense NN achieved the highest sensitivity in most of the combinations (49/89), followed by the PNN (25/89) and the maxout model (15/89). From the tests we can conclude that the PNN network achieves the highest sensitivity, but struggles for unbalanced data sets. On the contrary the ordinary dense neural network performs best with larger amounts of training data, achieving the highest sensitivity on the largest number of combinations, but never attains the same degree of sensitivity as the PNN. The maxout layer underperforms on most combinations, but exhibits impressive training memory, attaining the most balanced performance on all combinations.

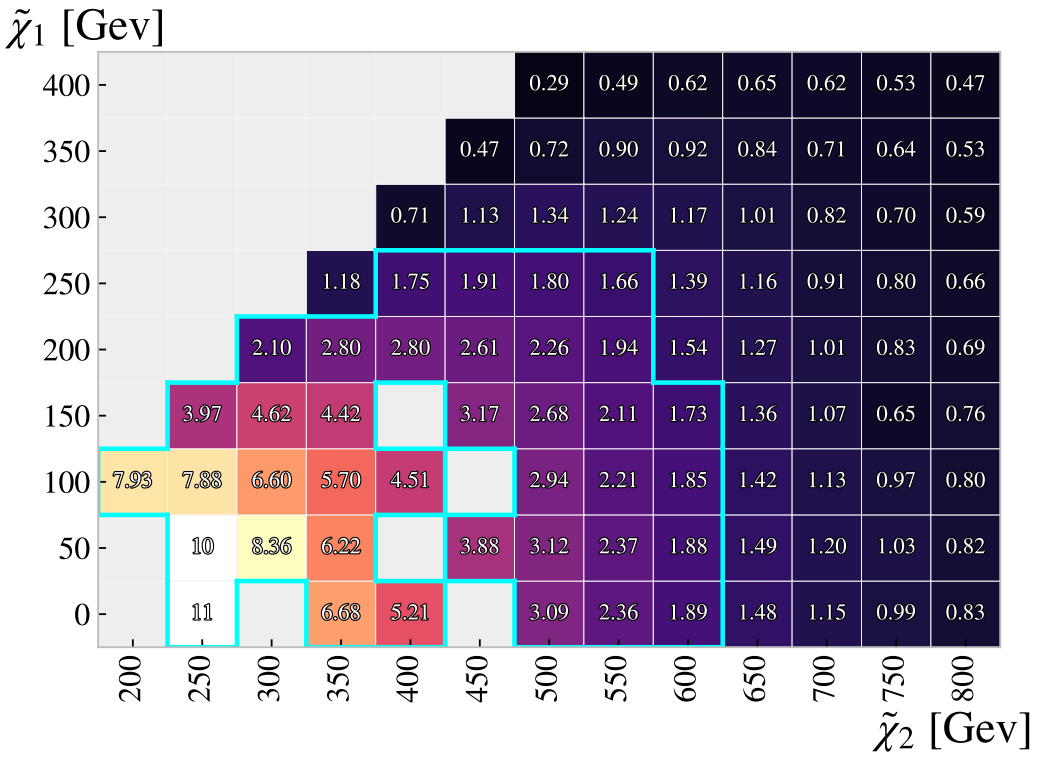


Figure 4.22: A grid displaying the expected significance on the full statistics signal set, using the signal region created by the maxout network. A band around each cell with a significance over 1.64 has been included.

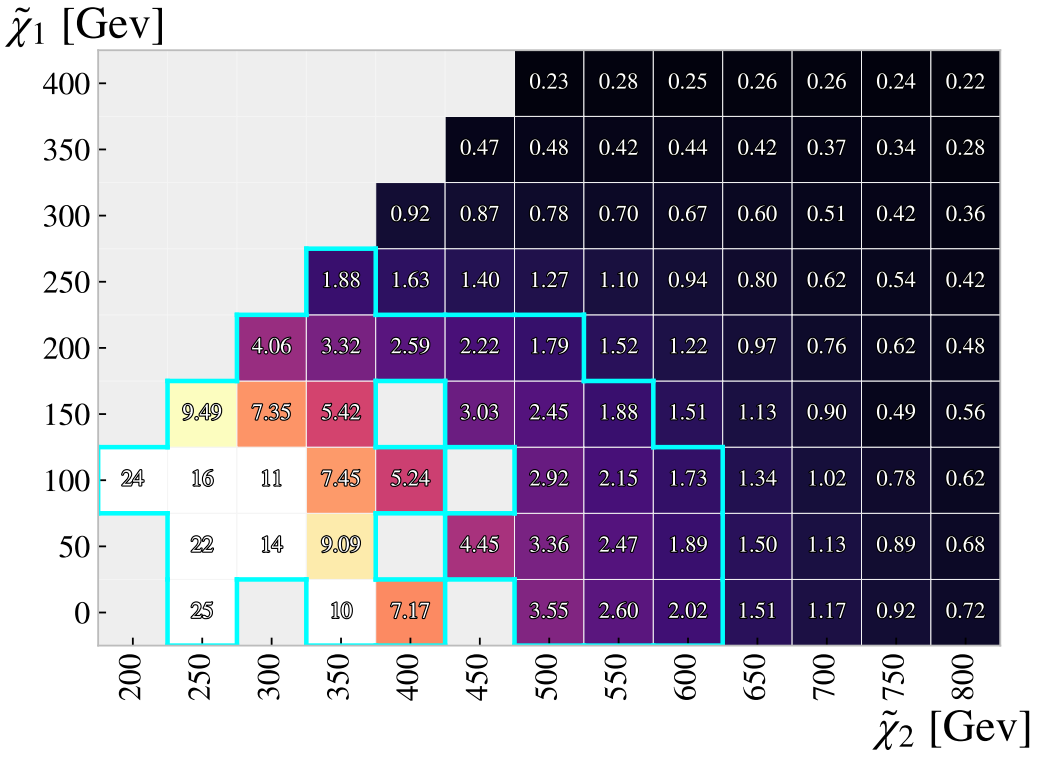


Figure 4.23: A grid displaying the expected significance on the complete signal grid, using the signal region created by the PNN network. A band around each cell with a significance over 1.64 has been included.

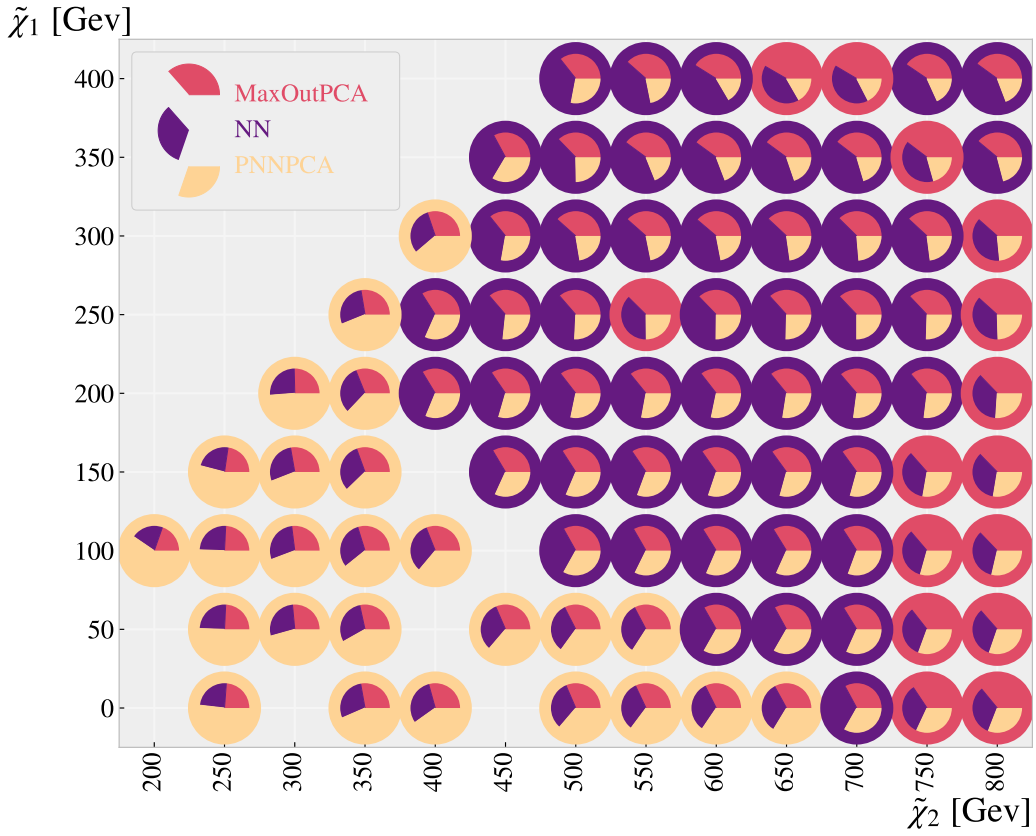


Figure 4.24: A sensitivity comparison between a dense NN, PNN and maxout on the complete signal grid. A PCA analysis has been applied to the data being utilized by the latter two models. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

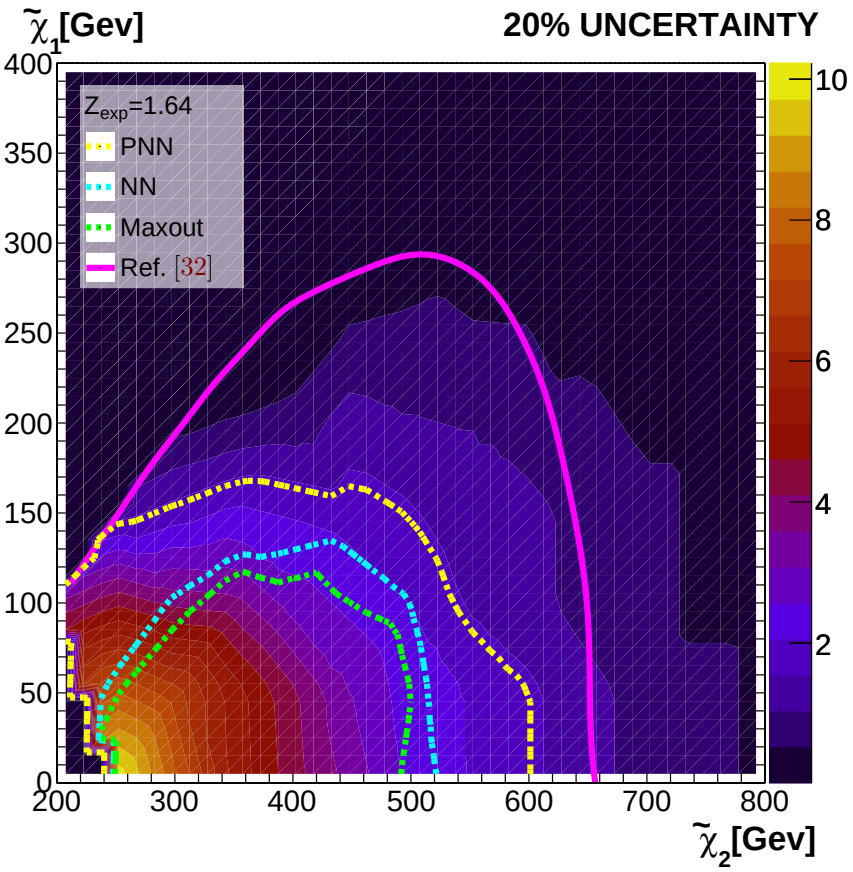


Figure 4.25: A surface plot of the significance achieved by the PNN on the complete signal grid. Contours are drawn around the band equal to a significance of 1.64 for the PNN (yellow), dense NN (cyan), maxout model (green) and the ATLAS analysis [32] (pink). The significance achieved by the ML models were calculated with a flat uncertainty equal to 20% of the background.

4.8 Comparing Exclusion Limits between Models and Previous Analysis

In the previous section I compared the expected significance between the three optimal models for the complete signal grid. In this section I will do the same, but with the introduction of a flat uncertainty on the background. Additionally, I will compare the achieved exclusion limits (the mass combinations with an expected significance of over 1.64) by the models created in this thesis, with the limit set in the paper by ATLAS from 2021 [32], *Search for chargino–neutralino pair production in final states with three leptons and missing transverse momentum in* $\sqrt{s} = 13$ *TeV pp collisions with the ATLAS detector.*
In the analysis by ATLAS, they have not assumed a flat uncertainty. Instead, an analysis has been made on the simulated data over individual regions and collision processes. Due to time constraints, a flat uncertainty has been made for comparison reasons. Aligning with what I have found when reading similar analysis (as well as based on recommendation from my supervisors), the uncertainty will be in the region of 20% of the SM background.

In figure 4.25, I have drawn the contours of the significance using the PNN, while outlining all points corresponding to a significance of 1.64. The same outline has been made on top of the contours of the PNN for the dense NN, maxout model and the statistical analysis made by ATLAS. The calculations of the significance for the output of the ML models were made with a flat uncertainty equal to 20% of the background. The figure shows that none of the models are able to extend the sensitivity achieved by ATLAS, although the PNN was able to equal the ATLAS analysis for smaller masses. Furthermore, it is evident from the figure that the NN and maxout model are not able to extend the limit as far as the PNN. In section 4.7 I presented the significance, calculated without uncertainty. In these calculations the dense NN and maxout model were able to extend the limit past that achieved by the PNN (in certain areas). This contradiction can be explained by the fact that PNN achieved a much higher sensitivity for smaller masses, in some cases

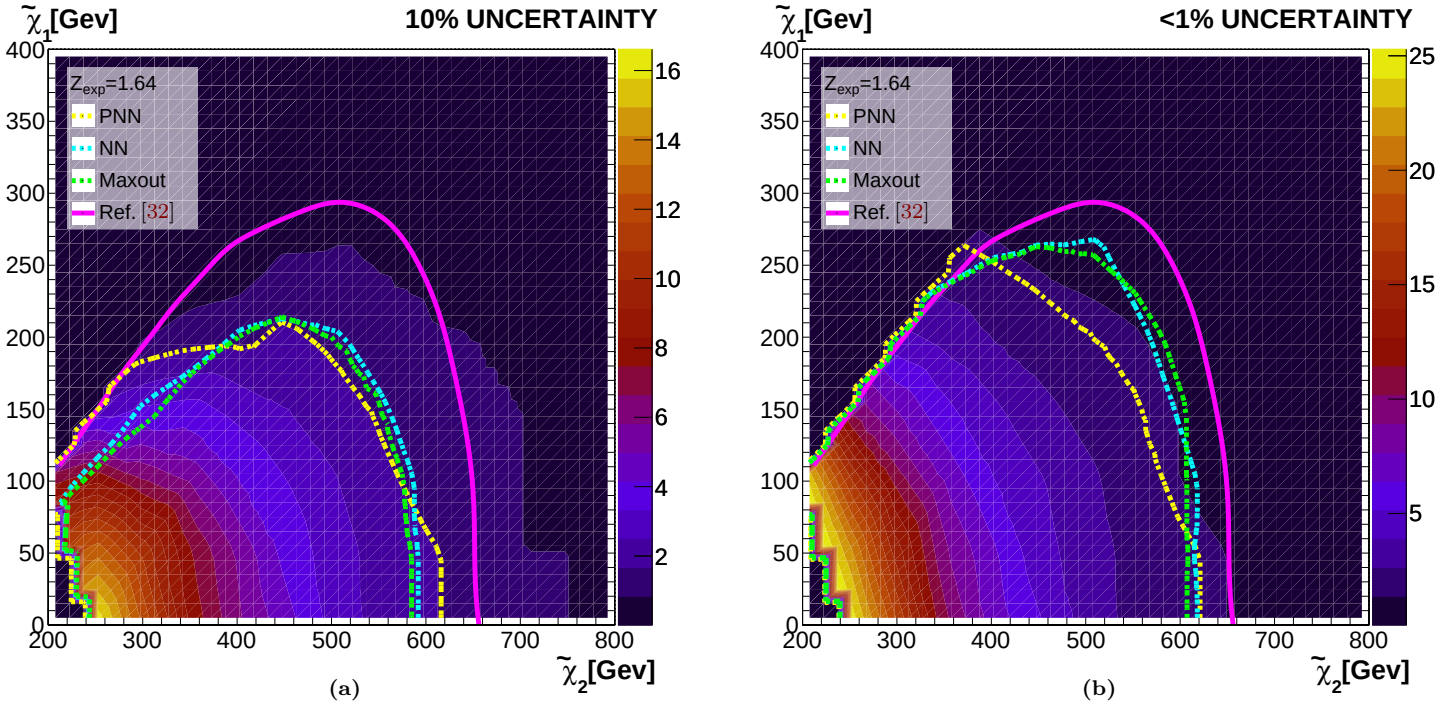


Figure 4.26: Two surface plots of the significance achieved by the PNN on the complete signal grid. Contours are drawn around the band equal to a significance of 1.64 for the PNN (yellow), dense NN (cyan), maxout model (green) and the ATLAS analysis [32] (pink). The significance achieved by the ML models were calculated with a flat uncertainty equal to 10% of the background in the left figure 4.26a and less than 1% in the right 4.26b.

even doubling the sensitivity to that of the dense NN and maxout model. When introducing uncertainty, the NN and maxout model are penalized for attempting to tune according to all signals, thereby not attaining enough sensitivity to 'survive' the reduced significance. In contrast, the PNN, although attempting to tune for all masses, includes the bias towards large statistics signals, which explains why there is less of a difference when comparing the limit with and without uncertainty.

To further study the sensitivity of the models, I produced two additional figures similar to the one above. In these figure I applied an uncertainty of 10% (4.26a) and less than 1% (4.26b) of the background. As expected, the limits converge close towards the limits observed in section 4.7 for smaller uncertainties. In figure 4.26a, we even discern that the limits set by the dense NN and maxout model extend past the PNN, for high masses ($\tilde{\chi}_1 > 175$ and $\tilde{\chi}_2 > 400$GeV), although they are outperformed for most other combinations. In figure 4.26b, I used an uncertainty close to 0%. As depicted in this figure, we discern that the dense NN and maxout model very closely resemble the limits from ATLAS, only being beaten by 50GeV for high mass $\tilde{\chi}_2$. For events in the mass region of $\tilde{\chi}_1 \approx 250$GeV and $\tilde{\chi}_2 \approx 400$GeV, the PNN seems to ever so slightly extend the limit produced by ATLAS. Although this is promising for the method, it is worth mentioning that the uncertainty is far from realistic. Finally, the size by which the limits are extended are so small that it could be explained by the method of interpolation between the grid of mass combinations.

A final observation is that the dense NN and maxout model, although not performing equally to the PNN for realistic uncertainties (20%), did achieve the highest performance in the areas which are not yet excluded ($\tilde{\chi}_1 \gtrsim 300$ and $\tilde{\chi}_2 \gtrsim 700$GeV). This is further exemplified in the contour plots for the dense NN and maxout model, which can be found in the appendix D. Especially the maxout model, which achieved the highest sensitivity for the largest masses of $\tilde{\chi}_2$, utilized an impressive long-term memory to draw benefits from the large statistics of the lower mass combinations, while preserving a relatively balanced performance on lower statistics performance. Compared to the PNN, which achieved difference in sensitivity of $max(Z) - min(Z) = 24.79$ in the case of zero uncertainty, the maxout model achieved a difference of only $max(Z) - min(Z) = 10.91$. When attempting to explore further regions in the parameter space of the chargino-neutralino pair, balanced performance and the ability to exploit the full statistics in the signal data, are critical attributes. Therefore, I believe, the generalizability and memory of the maxout model, makes it an interesting candidate for further studies.

Final Remarks and Possible Improvements

To summarize the discoveries in the figures above, none of the methods were able to extend the limits beyond those by ATLAS, at least not to a satisfactory degree. This is by no means to say that the methods discussed and presented in this thesis are not of interest. For starters, the matter of trying to extend the limit of sensitivity beyond that produced by ATLAS, was given little attention and consideration during the timespan of this thesis. Most of the time was instead spent comparing and exploring the different models. In contrast, the limit set by ATLAS is the result of a large team of focused scientist all attempting to push the limit as far as they can. This considered, I believe the models show great promise. If I were to prioritize expansion of exclusion limits, there are several aspects of the analysis in this thesis I would improve.

- *A more advanced analysis of the signal region*: In my analysis I chose to define the signal regions from each model's output using a brute force approach. A more advanced approach, applying optimization technics to find the maximum possible significance from each output, would most likely produce a higher sensitivity.
- *Implement multiple orthogonal signal regions*: In the analysis made by ATLAS [32], several signal regions were implemented and studied, with the final sensitivity equalling the combination of all the regions. This approach is far more complex, and can lead to a higher sensitivity. I believe that with the same treatment on the ML output presented in this thesis, a much higher significance would be achieved.
- *Devote additional time to studying the relationship between overfitting and significance*: The early stoppage criteria of ten epochs applied in this analysis was chosen for convenience and simplicity. This rigid criteria could be destructive for optimal performance, and by devoting additional time to study the relationship between the threshold for the criteria and optimal performance could lead to better performing models.
- *Perform a thorough hyperparameter search*: For comparison reasons I decided not to apply a hyperparameter search for the architecture of the networks. By attempting a grid search for the different hyperparameters, one might be able to find more sensitive models. Additionally, it would be interesting to attempt to combine the methods presented in this thesis, for example a PNN with LWTA layers.
- *Create region specific models*: In the final analysis I applied one model (per method) to study the full signal set. It would be interesting to explore the possibility of creating different models for different regions in the parameter (mass) space. In the analysis we have seen how models can both improve and worsen in performance with regard to specific masses by increasing the number of mass combinations in the training set. With individual models covering specific, possibly physics motivated regions, I believe some methods would benefit.
- *Focus on relevant regions*: In this analysis I have treated the performance of a model on all mass combinations equally. This was a choice I made to easier compare methods, but given the goal of extending the limits, it would most likely be beneficial to prioritize the regions closest to the existing limits, in hopes of achieving a higher sensitivity in these regions[42].
- *Further study of features*: In the beginning of the analysis, the signal was yet undecided. As a consequence, the variables used in the analysis are not tailored to the SUSY signal. In retrospect, other variables could be relevant for my analysis. For example the Z-veto ($|m_{lll} - m_z|$) or the stransverse mass (M_{T2}) [45], which were used in the analysis of ATLAS in 2021 [32].

[42] See the final paragraph in section 4.6

Conclusion & Outlook

In this thesis I have applied a range of ML models to the search for chargino-neutralino production resulting in a three lepton final state with missing transverse momentum. Two data sets were utilized during the analysis; simulated MC data, including both the SM background and the BSM signal, and measured proton-proton collisions at $\sqrt{s} = 13TeV$ produced at the LHC and detected at ATLAS. The models applied and studied during my analysis were a set of NN variants, in addition to a BDT which was used to create a benchmark for my analysis. The network variants encompassed a diverse array of approaches including an ordinary dense NN, ensemble networks employing LWTA layers and a PNN.

The simulated signal set included and studied in the analysis consisted of a set of orthogonal BSM variants, specifically different masses for the chargino ($\tilde{\chi}_1^{\pm}$) and neutralino ($\tilde{\chi}_1$). In my analysis I tested two approaches for dealing with a diverse signal set; training one model per variant, and training one model on a larger set of variants. Comparing the two, I found the latter to achieve a higher sensitivity as a consequence of a couple of factors. By including an assortment of variations of new physics, the ML models were able to avoid overfitting longer, which allowed for deeper learning. Furthermore, I found that the models were able to exploit overlapping feature trends between the variations, which resulted in more training data for all mass combinations. As a consequence, all remaining models were trained on a diverse signal set, and large parts of the analysis was focused on how each individual method handled this decision.

I studied three variants of LWTA layers; channel-out, maxout, and SCO. The first two were taken from the paper by Wang et al. [9], and the third layer was introduced in this present thesis. Each layer, reduces the number of nodes during a forward propagation, similarly to the dropout layer, but does so by comparing activation with other nodes in the layer, and dropping all but the largest node. To study the implementation and effect of these layers, I constructed a set of figures to visualize the activation and dropping of nodes, before and after training. When dissecting the figures, I observed that the LWTA layers (specifically the maxout layer) were able to build an ensemble of networks by means of trend specific paths. In other words, after training the model, the data chose different paths through the network dependent on if it was background or signal. Moreover, by comparing two sets of signal with different mass combinations, I found that the model was also able to differentiate between different variations of signal. This indicates a strong long-term memory, which allows the model to target a larger set of signal and is an important attribute in the LWTA layers. Additionally to studying the maxout layer, I repeated the analysis for SCO. By comparing the activation between the maxout and SCO layer, I found that the SCO layer was able to reduce complex co-adaptation by motivating all nodes to contribute to the output.

The PNN applied in the thesis was inspired by the network introduced in the paper by Baldi et al. [8]. The purpose of the PNN was to motivate the model to differentiate between the mass combination in the signal set, through including the parameters (choice of chargino and neutralino mass) as a feature. To study the effect of the PNN, I drew the distribution of a subset of mass combinations, where all the events were given the same parameters. The idea was that the PNN would perform better when events were given the correct parameters, compared to when they were not. By repeating this test, but assigning the data another set of parameters I found that PNN did perform better when predicting on data which were assigned correctly, although not by much. For example, compared to when the events were given the wrong parameters, signal events with masses equal to $\tilde{\chi}_1 = 50$ and $\tilde{\chi}_2 = 250$GeV, improved effectiveness by 3% when applying a rigid cut on the output of 0.975. This indicates that the PNN was able to discriminate between different variations of signal, even if only by a small amount.

When studying and comparing the performance of the ML models, two different sets of the signal were used, original ($\tilde{\chi}_1 \in [0-400]$GeV and $\tilde{\chi}_2 \in [400-800]$GeV) and the complete signal grid ($\tilde{\chi}_1 \in [0-400]$GeV and $\tilde{\chi}_2 \in [200-800]$GeV), where the first was a subset of the second. When comparing the achieved sensitivity, or

significance of the models on each mass combination in the original signal set, I found that the maxout model outperformed all other LWTA models, achieving a higher significance in 24/30 combinations. Although, the introduced SCO layer did not outperform maxout model, it did achieve a higher significance in 6 combinations. Most likely, by modifying the SCO layer during prediction, the performance of the layer would improve.

Four models (not including channel-out and SCO) were chosen when comparing performance on the original signal set; ordinary dense NN, maxout model, PNN and a BDT implemented using the default settings of `XGBoost` [7]. In the comparison I found that all three network variants were able to outperform the BDT, although I believe this to be explained by little attention given towards the architecture of the BDT. Out of the three network variants, the maxout model was able to achieve the highest significance for most mass combinations (24/30), but mostly in the higher mass range ($\tilde{\chi}_2 > 600$GeV). In the events with lower masses and higher statistics ($\tilde{\chi}_2 < 600$GeV), the PNN outperformed all others, almost doubling significance achieved by the maxout model. Shared among all models was the fact that they all found processes with high amounts of missing transverse energy difficult to separate from the signal, i.e. $Diboson(lll)$, $t\bar{t}$ and *top other*.

Before applying the models to the complete signal grid, I applied a PCA to study if it could improve performance. When requiring the conservation of 99.99% of the variation from the original feature set, 5 features were removed. Training the dense NN, maxout model and PNN with this new data set, I found it to improve the sensitivity of the two latter models. The comparison was based on the choice to weight the importance of all mass combinations equally.

Finally, in my analysis I compared the performance of my three best performing models (maxout model and PNN with a PCA, and the dense NN) on the complete signal grid. Additionally, I compared the results to the expected exclusion limits ($Z > 1.64$) set by ATLAS in 2021 [32]. The calculated significance for the models were done using a flat uncertainty of 20%, 10% and $< 1\%$. Based on the comparison I found that none of the ML models were able to extend the limits set by ATLAS, with the exception of the PNN when utilizing $< 1\%$ uncertainty. For an uncertainty of 20%, the PNN was able to achieve a limit which mirrors ATLAS for smaller masses ($\tilde{\chi}_2 < 250$GeV) and set a limit past that achieved by the other networks, for both the chargino and neutralino mass. When decreasing the uncertainty, the maxout model and dense NN were able to extend the limit past that achieved by the PNN for higher masses, but never surpassing the limit by ATLAS.

From my analysis I found that where the PNN exhibits bias towards higher statistic signal, the ordinary dense NN and maxout model are able to achieve a more balanced sensitivity. Especially the maxout model, with an impressive long-term memory was able to uphold a strong performance for lower statistics signal and smaller differences in significance ($\Delta Z \approx 10$ when uncertainty set to $< 1\%$). The effect of the long-term memory is further demonstrated in the comparison between the model after training on the original, and the complete signal grid. In said comparison the maxout model improved its performance on the original set in 17/30 mass combinations, compared to the PNN which only improved 2/30. Due to the fact that future analysis will need sensitivity in low statistics regions (high mass), I believe the LWTA layers to be interesting candidates for future models, in regard to their ability to exploit high statistics combinations while maintaining a strong performance on lower statistics signal.

In conclusion, my results indicate that although none of the ML models extended the expected exclusion limit made by the traditional CC approach from ATLAS in 2021 [32], the network variants showed great promise. I believe that through a more complex analysis of the signal region, both in choice of region and combining the statistics from multiple signal regions, the models would drastically increase in sensitivity. Furthermore, any future ML models would greatly benefit from including multiple overlapping new physics variants, especially through the application of models displaying long-term memory, similarly to the models utilizing the LWTA layers in this thesis.

Appendices

A Calculated Significance

A.1 The LWTA Models Applied to the Original Signal Set

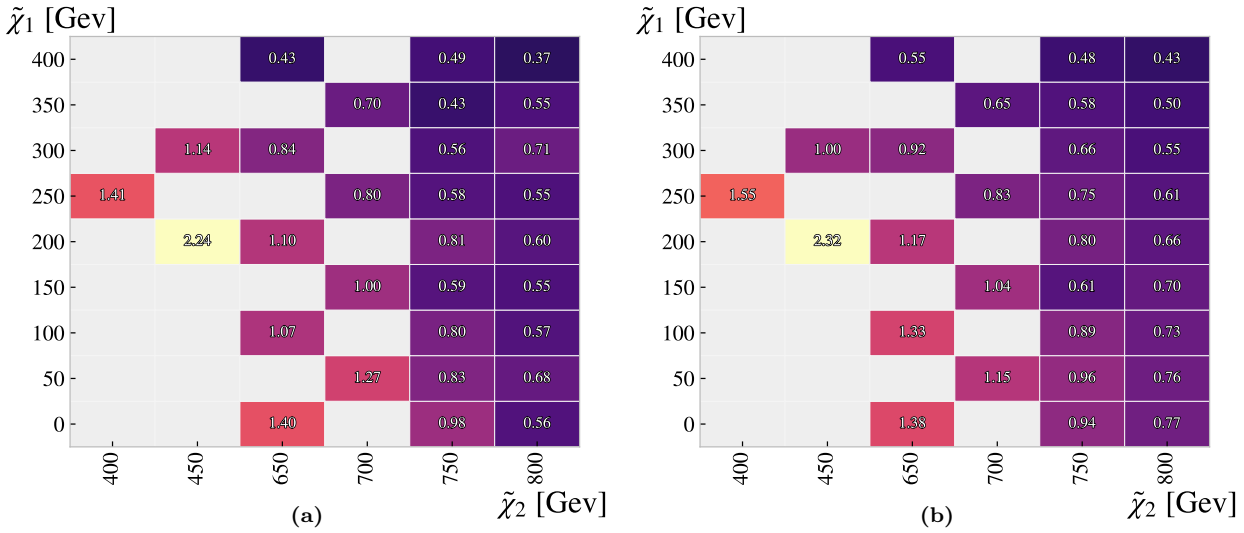


Figure 27: A grid displaying the expected significance on the original signal set, using the signal region created by the SCO 27a and a channel-out network 27b.

A.2 Results from the PCA

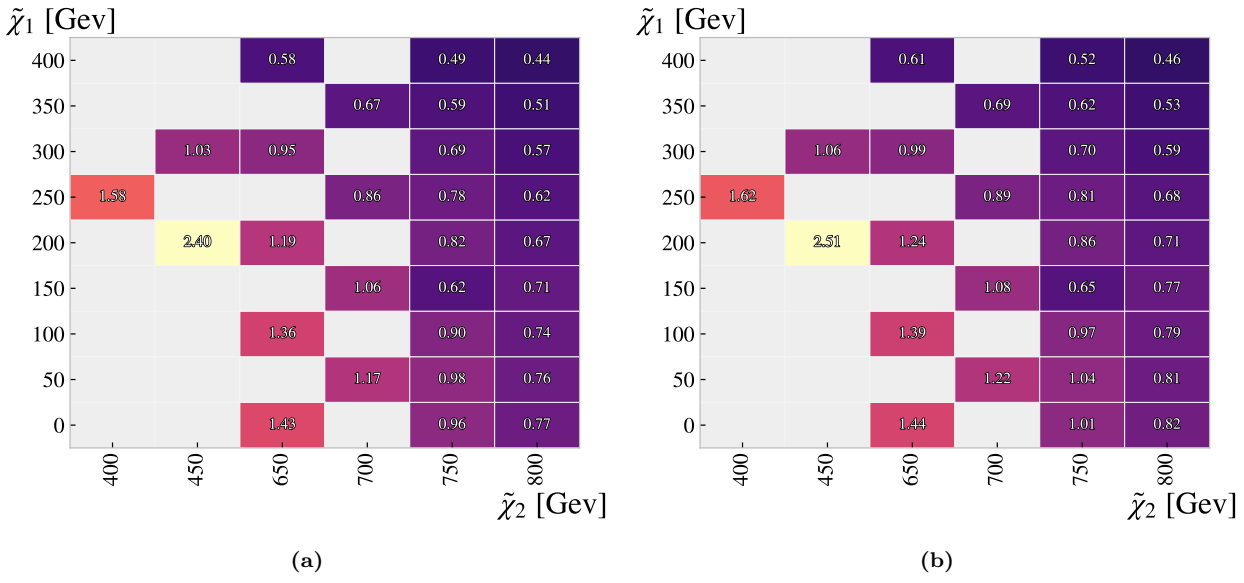


Figure 28: A grid displaying the expected significance on the original signal set, using the signal region created by the NN 28a and a maxout network 28b. A PCA analysis has been applied to the data being utilized in this result.

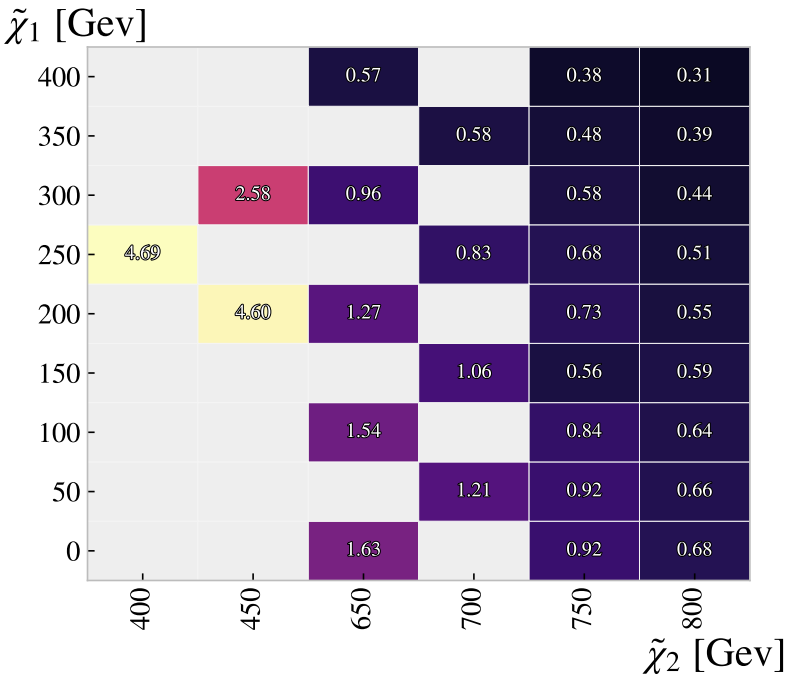


Figure 29: A grid displaying the expected significance on the original signal set, using the signal region created by the PNN network. A PCA analysis has been applied to the data being utilized in this result.

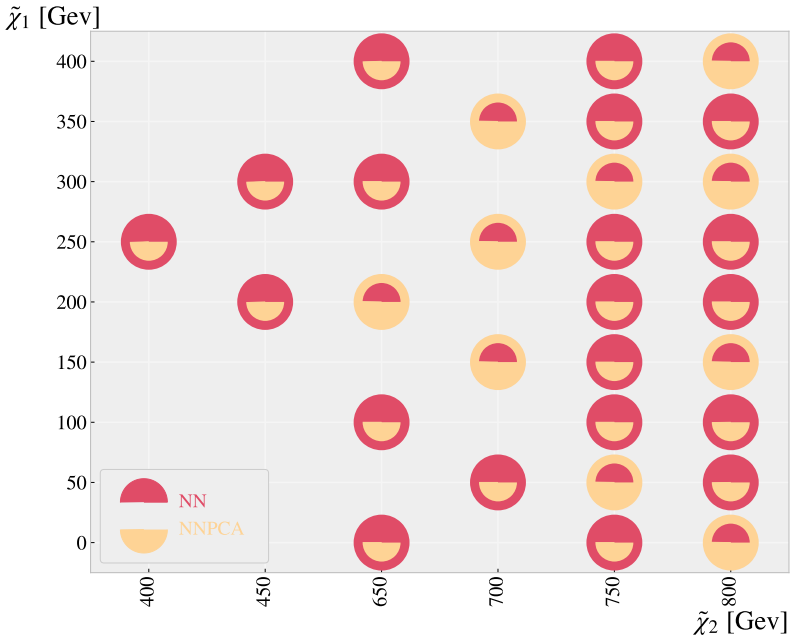


Figure 30: Pie-plot comparing sensitivity on the original signal set, where the figure shows the comparison between a model trained on data with and without a PCA. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

A.3 Comparing Models Trained on Original and the Complete Signal Grid

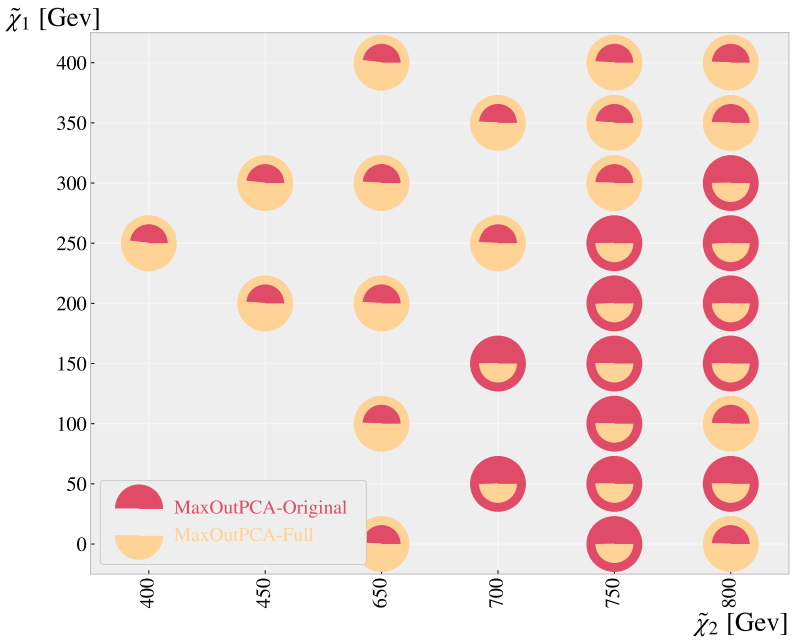


Figure 31: Pie-plot comparing sensitivity achieved by the maxout model on the original signal set, where the figure shows the comparison between a model trained on the original signal grid, and the complete signal grid.. A PCA analysis has been applied to the data being utilized in this result. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

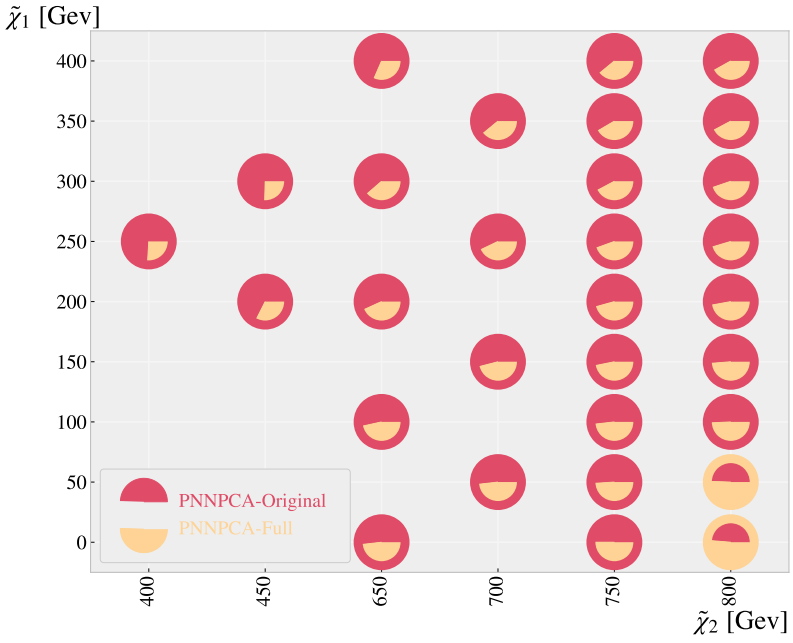
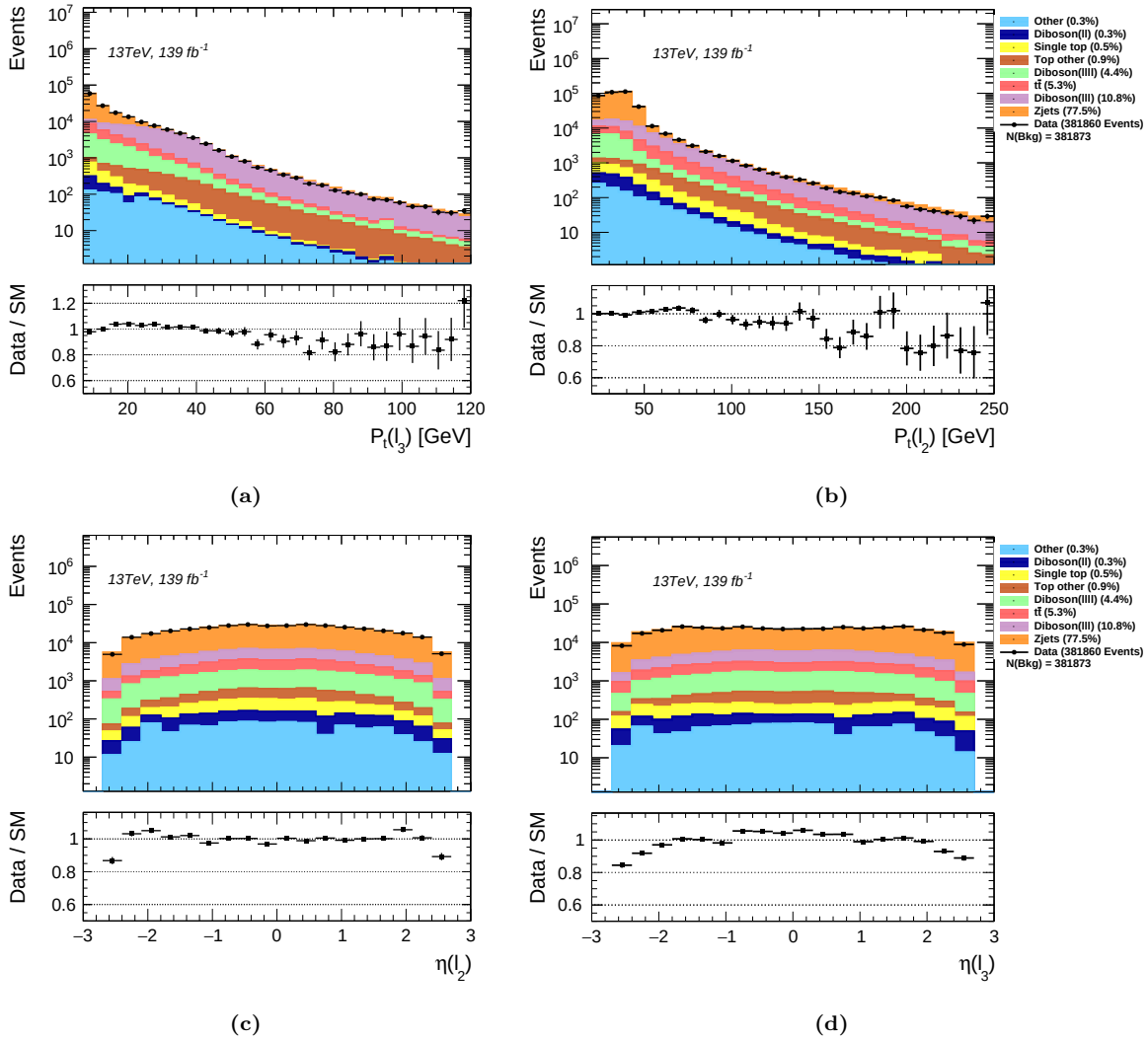


Figure 32: Pie-plot comparing sensitivity achieved by the PNN model on the original signal set, where the figure shows the comparison between a model trained on the original signal grid, and the complete signal grid. A PCA analysis has been applied to the data being utilized in this result. The size of each 'slice' represents the relative size of the significance and the color around each point displays the method with the largest sensitivity for the respective combination.

B The Features

B.1 The Feature Distribution



(a) **(b)**

(c) **(d)**

Figure 33: MC simulated and measured data comparison and event distribution for each channel over P_t for the third 33a and second 33b lepton. Similarly, the distribution over η for the second 33c and third 33d lepton.

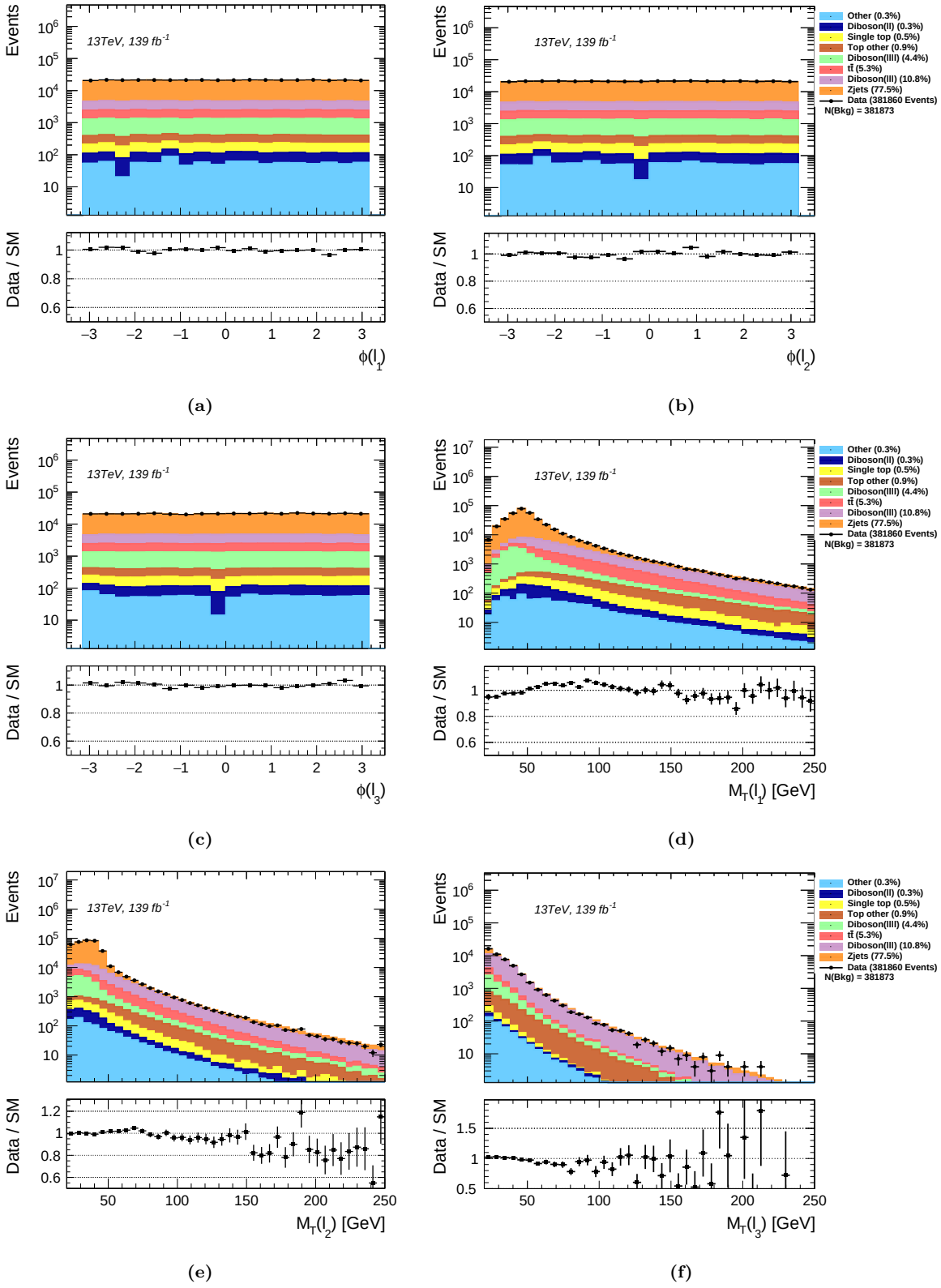


Figure 34: MC simulated and measured data comparison and event distribution for each channel over ϕ for the first 34a, second 34b and third 34c lepton. Similarly, the distribution over m_t for the first 34d, second 34e and third 34f lepton.

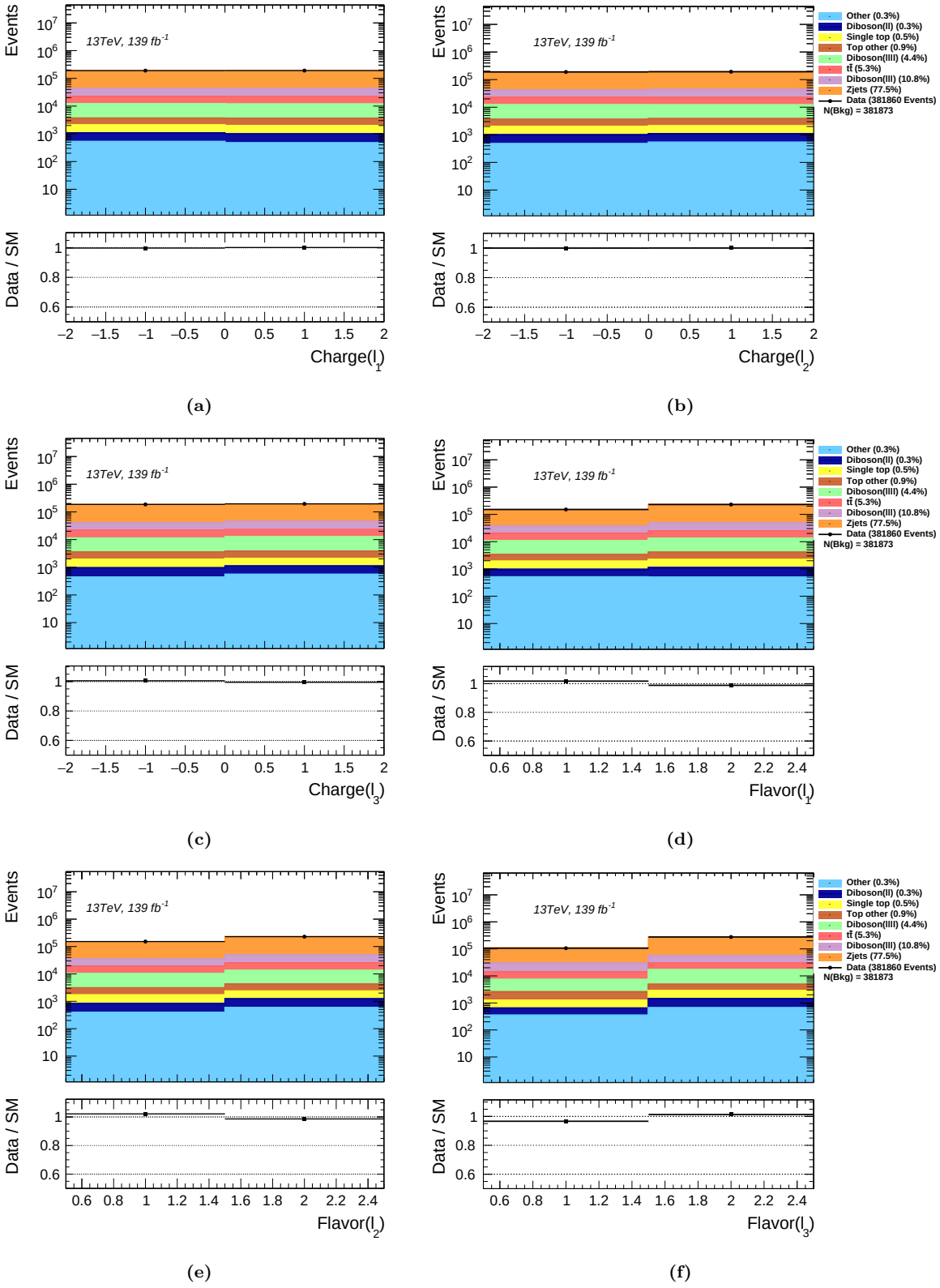


Figure 35: MC simulated and measured data comparison and event distribution for each channel over the charge for the first 35a, second 35b and third 35c lepton. Similarly, the distribution over the flavor for the first 35d, second 35e and third 35f lepton.

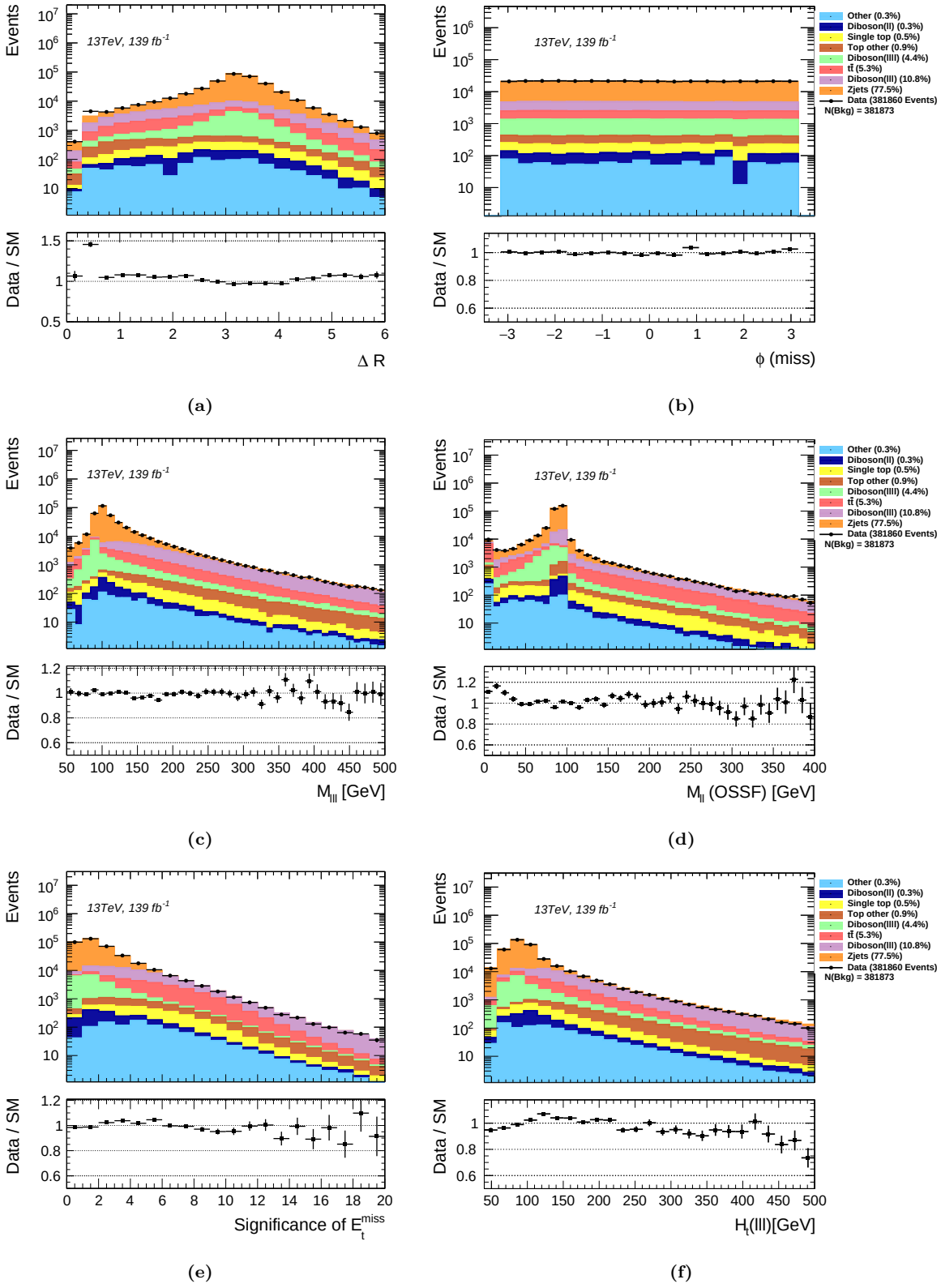


Figure 36: MC simulated and measured data comparison and event distribution for each channel over ΔR 36a and the azimuthal angel 36b of the missing transverse energy. The distribution of the invariant mass of the three leptons 36c and the OSSF pair 36d. The distribution over the significance of the missing transverse energy 36e and the sum of P_t 36f.

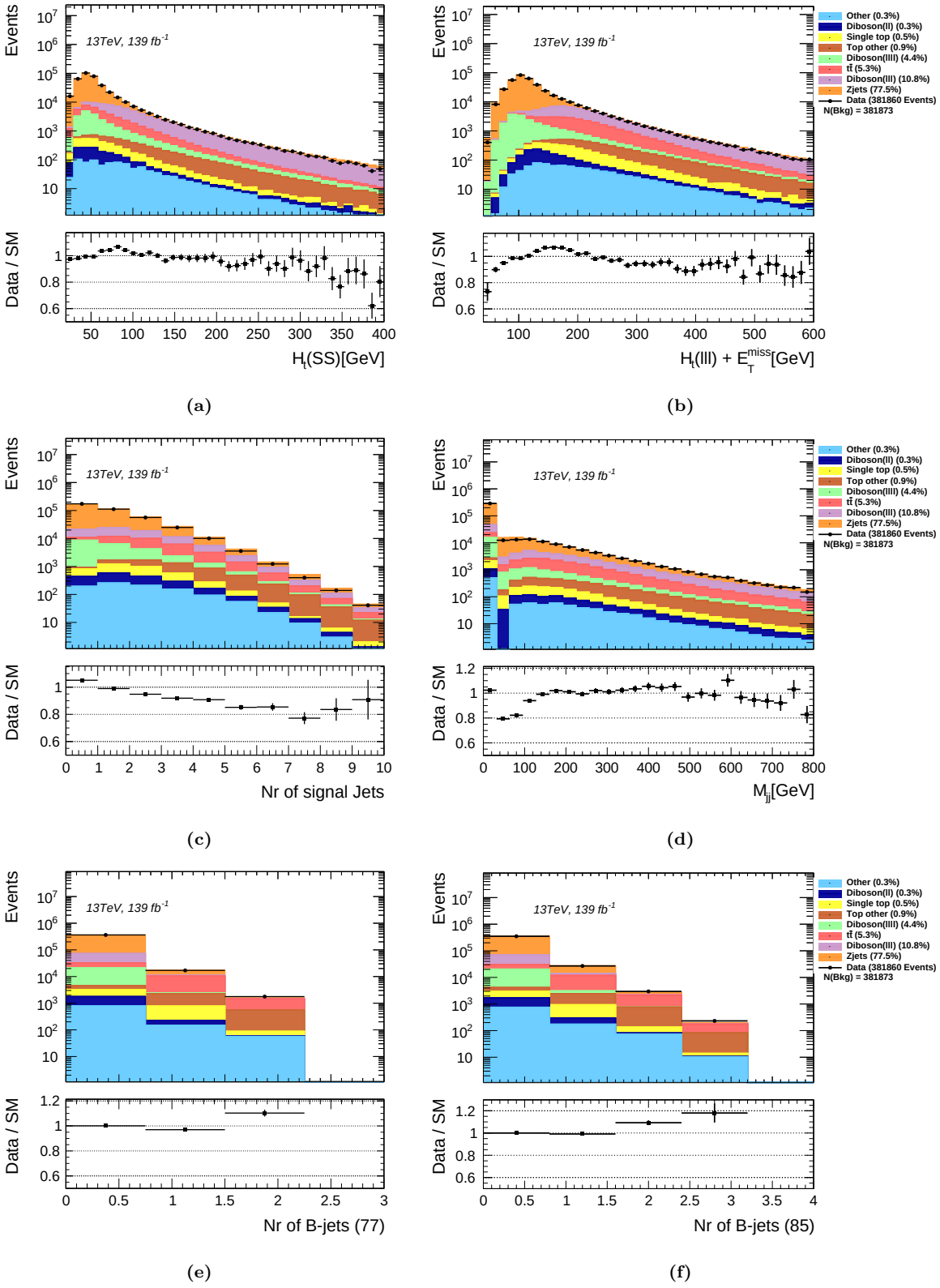


Figure 37: MC simulated and measured data comparison and event distribution for each channel over the sum of P_t for the SS pair 37a and the sum over all three leptons added with E_t^{miss} 37b. The distribution of number of signal jets 37c and the mass of the leading di-jet pair 37d. Finally, the number of B-jets with 77% 37e and 85% 37f certainty.

B.2 The Selection of Feature

Feature Name	Description
P_t	Transverse momentum
η_t	Pseudo rapidity
ϕ_t	Azimuthal angle
M_T	Transverse mass
$Charge$	EM charge
$Flavour$	Particle type
E_T^{miss}	Missing transverse energy
$\phi(miss)$	Azimuthal angle of the missing transverse energy
M_{lll}	Trilepton mass
$M_{ll}(OSSF)$	Mass of the OSSF pair
Sig E_T^{miss}	Significance of E_T^{miss}
$H_t(lll)$	Sum of P_t for all three leptons
$H_t(SS)$	Sum of P_t for the Same Sign pair
$H_t(lll) + E_T^{miss}$	-
ΔR	Distance defined in the $\eta - \phi$ space
Flavor combo	Combination of flavors for all three leptons
Nr of signal Jets	Nr of jets passing the signal criteria
M_{jj}	Mass of the leading jet pair
Nr of B-jets(77)	Number of B-jets with 77% efficiency
Nr of B-jets(85)	Number of B-jets with 85% efficiency

Table 2: A summary and description of all features used in this analysis.

C The implementation of Channel-Out, SCO and Maxout

Channel-out

```
def call(self, inputs: tf.Tensor,mask: tf.Tensor = None) -> tf.Tensor:
    # Pass input through weight kernel and adding bias terms.
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)

    num_inputs = inputs.shape[0]
    if num_inputs is None:
    num_inputs = -1

    # Reshaping inputs such that they are grouped correctly
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]
    inputs = tf.reshape(inputs, new_shape)

    # Finding maximum activations and setting losers to 0.
    outputs = tf.math.reduce_max(inputs, axis=-1, keepdims=True)
    outputs = tf.where(tf.equal(inputs, outputs), outputs, 0.)
    # Reshaping outputs to original input shape
    outputs = tf.reshape(outputs, [num_inputs, self.units])

    #Count the activate nodes. This variable is used when plotting the activations
        .
    self.counter = outputs

    return outputs
```

Listing 1: Python implementation for the custom activation function used to define the channel-out layer.

```
def call(self, inputs: tf.Tensor, mask: tf.Tensor = None) -> tf.Tensor:
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)
    num_inputs = inputs.shape[0]

    if num_inputs is None:
    num_inputs = -1
    # Create the indices to shuffle and unshuffle nodes
    shuffle_index = tf.random.shuffle(self.index)
    unshuffle_index = tf.tensor_scatter_nd_update(tensor = self.zeros ,
                                                  indices = tf.reshape(
                                                      shuffle_index,
                                                      [inputs.shape[1],1]
                                                      ),
                                                  updates = self.index)
    inputs_s = tf.gather(inputs, shuffle_index, axis = 1)


    # Reshaping inputs such that they are grouped correctly
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]
    inputs_s = tf.reshape(inputs_s, new_shape)

    # Finding maximum activations and setting losers to 0.
    outputs = tf.math.reduce_max(inputs_s, axis=-1, keepdims=True)

    outputs = tf.where(tf.equal(inputs_s, outputs), 1.0, 0.)
    # Reshaping outputs to original input shape
    outputs = tf.reshape(outputs, [num_inputs, self.units])

    outputs = tf.gather(outputs, unshuffle_index, axis = 1)
    outputs = tf.multiply(inputs, outputs)
    #Count the activate nodes. This variable is used when plotting the activations
    self.counter = outputs
    return outputs
```

Listing 2: Python implementation for the custom activation function used to define the SCO layer.

```
def call(self, inputs: tf.Tensor) -> tf.Tensor:
    # Passing input through weight kernel and adding bias terms
    inputs = gen_math_ops.MatMul(a=inputs, b=self.kernel)
    inputs = nn_ops.bias_add(inputs, self.bias)

    num_inputs = inputs.shape[0]
    if num_inputs is None:
        num_inputs = -1
    num_competitors = self.units // self.num_groups
    new_shape = [num_inputs, self.num_groups, num_competitors]

    # Reshaping outputs such that they are grouped correctly
    inputs = tf.reshape(inputs, new_shape)
    # Finding maximum activation in each group
    outputs = tf.math.reduce_max(inputs, axis=-1,keepdims=True)

    counter = tf.where(tf.equal(inputs, outputs), outputs, 0.)

    # Reshaping outputs to original input shape
    outputs = tf.reshape(outputs,[num_inputs, self.num_groups])

    #Count the activate nodes. This variable is used when plotting the activations
        .
    self.counter = tf.reshape(counter, [num_inputs, self.units])

    return outputs
```

Listing 3: Python implementation for the custom activation function used to define the maxout layer.

D Contour Plots for the Calculated Significance with a Flat Uncertainty

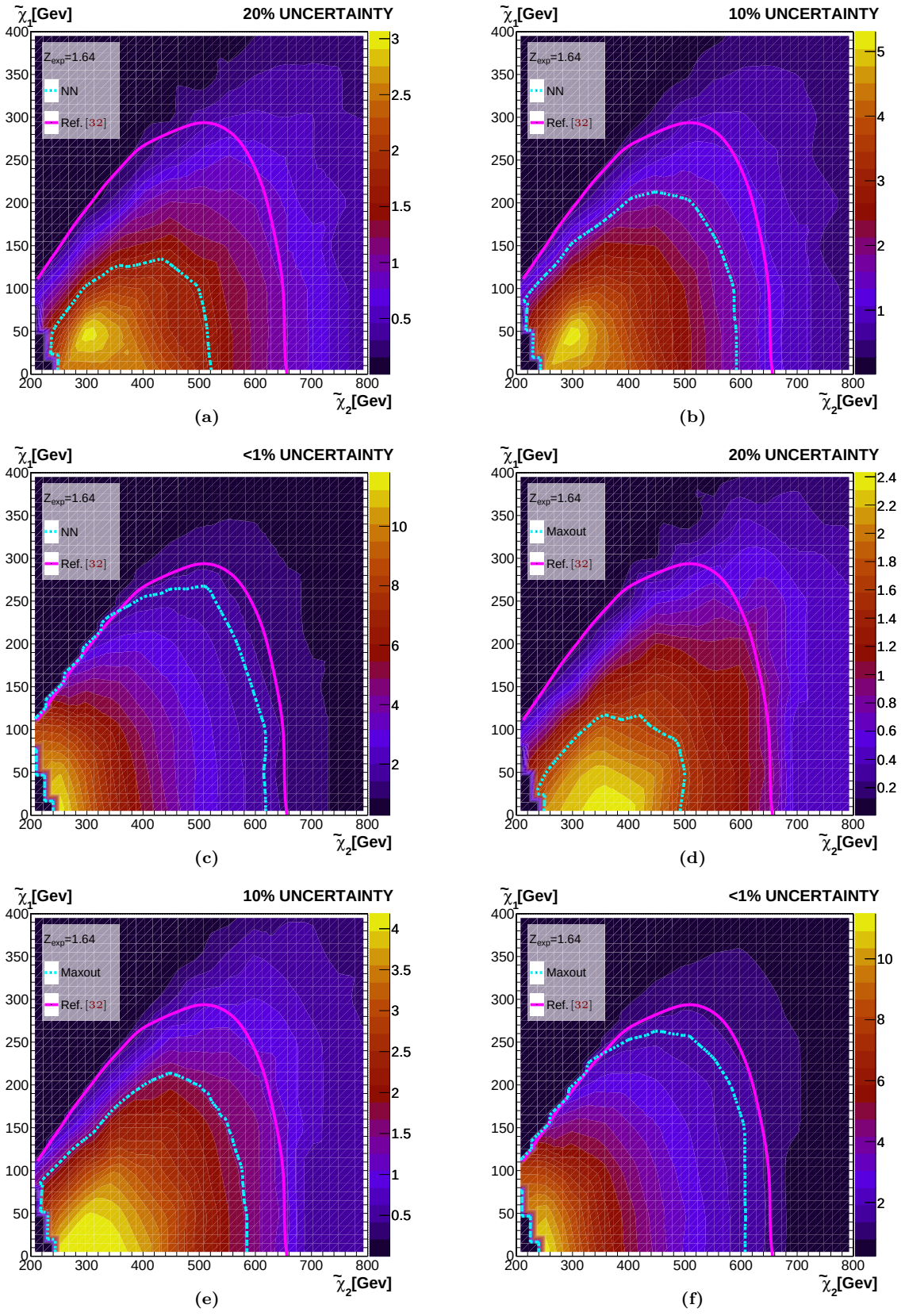


Figure 38: Contour plots of the significance achieved by the ordinary dense NN and maxout model on the complete signal grid. Contours are drawn around the band equal to a significance of 1.64 for each model respectively (cyan) and for the ATLAS analysis [32] (pink). The significance achieved by the ML models were calculated with a flat uncertainty equal to 20% (38a and 38d), 10% (38b and 38e) and $< 1\%$ (38c and 38f) for the dense NN and maxout model, respectively.

Acronyms

ADAM Adaptive Moment Estimation

API Application Programming Interface

ATLAS A Toroidal LHC Apparatus

AUC Area Under the Curve

BDT Boosted Decision Trees

BSM Beyond Standard Model

CC cut-and-count

CNN Convolutional Neural Network

CP Charge-Parity

DNN Deep Neural Networks

DT Decision Trees

EM Electromagnetic

FFNN Feed-Forward Neural Network

HEP High Energy Physics

HPC High Performance Computing

LHC Large Hadron Collider

LWTA Local-Winner-Takes-All

MC Monte Carlo

ML Machine Learning

MSSM Minimal Supersymmetric Standard Model

NN Neural Network

OMM One-Mass-Model

OSSF Opposite Sign Same Flavour

PCA Principal Component Analysis

PNN Parameterized Neural Network

QCD Quantum Chromo Dynamics

QED Quantum Electro Dynamics

QFT Quantum Field Theory

RNN Recursive Neural Network

ROC Receiver Operating Characteristic

SCO Stochastic-Channel-Out

SGD Stochastic Gradient Descent

SM Standard Model

SMM Several-Mass-Model

SP Superpartner

SUSY Supersymmetry

Bibliography

[1] ATLAS collaboration, G. Aad et al., *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, *Phys. Lett. B* **716** (2012) 1–29, [1207.7214].

[2] D. Chung, L. Everett, G. Kane, S. King, J. Lykken and L.-T. Wang, *The soft supersymmetry-breaking lagrangian: theory and applications*, *Physics Reports* **407** (2005) 1–203.

[3] A. Cole, S. Krippendorf, A. Schachner and G. Shiu, "Probing the structure of string theory vacua with genetic algorithms and reinforcement learning." 10.48550/arXiv.2111.11466.

[4] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban and D. Whiteson, *Jet flavor classification in high-energy physics with deep neural networks*, *Physical Review D* **94** (dec, 2016) .

[5] J. Pumplin, *How to tell quark jets from gluon jets*, *Phys. Rev. D* **44** (Oct, 1991) 2025–2032.

[6] P. Baldi, P. Sadowski and D. Whiteson, *Searching for exotic particles in high-energy physics with deep learning*, 1402.4735.

[7] T. Chen and C. Guestrin, *XGBoost: A scalable tree boosting system*, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016. DOI.

[8] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, *Parameterized neural networks for high-energy physics*, *The European Physical Journal C* **76** (2016) 235.

[9] Q. Wang and J. JaJa, *From maxout to channel-out: Encoding information on sparse pathways*, 1312.1909.

[10] J. J. T. M. F.R.S., *Xlviii. the relation between the atom and the charge of electricity carried by it*, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **40** (1895) 511–544.

[11] P. E. R. F.R.S., *Lxxix. the scattering of α and β particles by matter and the structure of the atom*, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **21** (1911) 669–688.

[12] F. Reines and C. L. Cowan, *The neutrino*, *Nature* **178** (1956) 446–449.

[13] J. Bernstein, *Spontaneous symmetry breaking, gauge theories, the higgs mechanism and all that*, *Rev. Mod. Phys.* **46** (Jan, 1974) 7–GS.

[14] Super-Kamiokande Collaboration collaboration, Y. Fukuda, T. Hayakawa, E. Ichihara, K. Inoue, K. Ishihara, H. Ishino et al., *Evidence for oscillation of atmospheric neutrinos*, *Phys. Rev. Lett.* **81** (Aug, 1998) 1562–1567.

[15] J. Joyce, *Finnegans Wake*. Penguin Books, New York, 1999.

[16] W. Thomson, *Lord kelvin addresed the british association for the advancement of science*, 1900.

[17] O. Antipin, D. Atwood and A. Soni, *Search for rs gravitons via wlwl decays*, *Physics Letters B* **666** (2008) 155–161.

[18] E. Oks, *Brief review of recent advances in understanding dark matter and dark energy*, .

[19] J. Pequenao, "Event Cross Section in a computer generated image of the ATLAS detector." 2008.

[20] E. Gramstad, *Searches for supersymmetry in di-lepton final states with the ATLAS detectorat $\sqrt{s}$ = 7 TeV*, .

[21] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 1412.6980.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[23] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez and J. Schmidhuber, *Compete to compute*, in *Advances in Neural Information Processing Systems*, vol. 26, Curran Associates, Inc.

[24] G. James and D. W. Trevor Hastie, Robert Tibshirani, *An introduction to statistical learning: With applications in R*, .

[25] S. Frette and W. O. Hirst, "Searching for the higgs through supervised and unsupervised machine learning algorithms." https://github.com/WilliamHirst/Advanced-Machine-Learning/blob/main/article/Project_1.pdf.

[26] M. Bugge, *Discovery and exclusion in high energy physics*, 2022.

[27] S. Raychaudhuri, *Introduction to monte carlo simulation*, in *2008 Winter Simulation Conference*, pp. 91–100. DOI.

[28] R. Brun and F. Rademakers, *Root - an object oriented data analysis framework*, in *AIHENP'96 Workshop, Lausane*, vol. 389, pp. 81–86, 1996.

[29] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, I. Fisk et al., *Update of the Computing Models of the WLCG and the LHC Experiments*. No. 2 in Technical design report. LCG.

[30] W. McKinney, *Data structures for statistical computing in python*, in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[31] A. Collaboration, *Search for type-III seesaw heavy leptons in dilepton final states in pp collisions at* $\sqrt{s}$ *= 13 TeV with the ATLAS detector*, 2008.07949.

[32] A. Collaboration, *Search for chargino–neutralino pair production in final states with three leptons and missing transverse momentum in* $\sqrt{s}$ *= 13 TeV pp collisions with the ATLAS detector*, 2106.01676.

[33] "Object-based missing transverse momentum significance in the ATLAS detector."

[34] M. Aaboud, , G. Aad, B. Abbott, D. C. Abbott, O. Abdinov et al., *Electron reconstruction and identification in the ATLAS experiment using the 2015 and 2016 LHC proton–proton collision data at* $\sqrt{s}$*= 13 TeV*, *The European Physical Journal C* **79** (aug, 2019) .

[35] A. Collaboration, *Performance of the ATLAS trigger system in 2015*, 1611.09661.

[36] A. Collaboration, *Performance of the ATLAS muon triggers in run 2*, 2004.13447.

[37] A. Collaboration, *Performance of electron and photon triggers in ATLAS during LHC run 2*, 1909.00761.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[39] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for NAS*, 1912.06059.

[40] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings.

[41] A. Gholamy, V. Kreinovich and O. Kosheleva, *Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation*, .

[42] ATLAS collaboration, G. Aad, B. Abbott, D. Abbott, A. Abed Abud, K. Abeling, D. Abhayasinghe et al., *Observation of WWW production in pp collisions at* $\sqrt{s}$ *= 13 TeV with the ATLAS detector. Observation of WWW Production in pp Collisions at* $\sqrt{s}$ *= 13eV with the ATLAS Detector*, 2201.13045.

[43] ATLAS collaboration, *Performance of the Reconstruction and Identification of Hadronic Tau Decays with ATLAS*, tech. rep., CERN, Geneva, 2011.

[44] ATLAS collaboration, *Performance of Top Quark and W Boson Tagging in Run 2 with ATLAS*, tech. rep., CERN, Geneva, 2017.

[45] C. G. Lester and D. J. Summers, *Measuring masses of semi-invisibly decaying particles pair produced at hadron colliders*, hep-ph/9906349.