UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DATA ANALYSIS IN UNDERWATER NEUTRINO TELESCOPES USING ADVANCED MACHINE LEARNING TOOLS

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Antonio Alejandro Aslan Suarez

TUTORIZADO POR

Alicia Herrero Debón

Joan Salvador Ardid Ramírez

CURSO ACADÉMICO: 2020/2021

Acknowledgments

First, I would like to thank my family for all the support they have given me throughout these college years, for congratulating me in the good times and comforting me in the bad ones. Although in previous years I may have been a little reluctant on the discipline you have instilled in me, I thank you now, because without it I would not have been able to write this thesis, and I am sure it will help me in my future career and personal life. And even though we will be even further apart in the years to come, we will always be as close as ever.

Nor can I forget those people who in these years have become brothers for me, since the moments that we have lived sharing room and apartment will not be erased no matter how much time passes. To my old high school classmates and my new friends in Valencia, thank you for all the good times we have spent together.

Finally, I would also like to thank my professors, those who have taught me the topics of the degree with great enthusiasm that I now consider essential for my future career. Especially to my tutors of this thesis: Alicia, Salva and Miquel. Thank you for giving me the opportunity to be part of this incredible work, although it has been a challenge for me to complete it, it helped me to put into practice all the concepts and methods of machine learning that I have learned in the last couple of years.

Abstract

The electromagnetic spectrum is not the only way for humans to study the stars; in fact, the study of the trajectory of neutrinos arriving at Earth can allow the study of massive and energetic celestial objects at distances much greater than the electromagnetic spectrum would allow. However, these subatomic particles can only be studied indirectly using the Cherenkov light captured by the photoreceptor lines that form neutrino telescopes such as the ANTARES Telescope. Currently, the data obtained by the ANTARES telescope is analyzed by two algorithms that are able to estimate the neutrino trajectory components. The problem with these algorithms is that they require the event to be captured by multiple lines to obtain a good result, but most of the events captured by the telescope are recorded by a single line, so there are many events from which not all the useful information can be extracted. Taking into account the existing gap in the current capability of estimating trajectories of these events, the objective of this work is to create an algorithm capable of improving the prediction capabilities of existing algorithms for the estimation of the Zenith component of the trajectory of events captured at the ANTARES telescope by a single line. In addition, the resulting model will also be used to make a first estimate of the neutrino energy.

On the other hand, and unlike the algorithms currently used, in our case we will choose to use a neural network model, since, being a multipurpose machine learning algorithm, these models are capable of identifying and performing the regression of any type of nonlinear function. Specifically, the final model obtained is a convolutional neural network capable of obtaining a probabilistic distribution of the Zenit component of the neutrino trajectory based on images made with the information received in the photoreceptors. The capacity of the developed algorithm has managed to reduce the trajectory estimation error by 50% compared to current algorithms, thus demonstrating that using a machine learning model can be a better option for trajectory estimation.

Resumen

Las radiaciones electromagnéticas no son la única forma que el ser humano tiene para estudiar los astros, de hecho, el estudio de la trayectoria de los neutrinos que llegan a la Tierra puede permitir el estudio de objetos celestes masivos y energéticos a distancias mucho mayores de lo que el espectro electromagnético podría permitir. Sin embargo, estas partículas subatómicas solo pueden ser estudiadas indirectamente utilizando la luz de Cherenkov captada por las líneas de fotorreceptores que forman los telescopios de neutrinos como el Telescopio ANTARES. Actualmente, los datos obtenidos por el telescopio ANTARES son analizados mediante dos algoritmos que son capaces de estimar las componentes de la trayectoria de los neutrinos. El problema de estos algoritmos reside en que requieren que el evento sea captado por múltiples líneas para obtener un buen resultado, pero la mayoría de eventos captados por el telescopio son registrados por una única línea, por tanto, hay muchos eventos de los cuales no se puede extraer toda la información útil. Teniendo en cuenta el vacío existente en la capacidad actual de estimación de trayectorias de estos eventos, el objetivo de este trabajo es la creación de un algoritmo capaz de mejorar las capacidades de predicción de los algoritmos existentes para la estimación de la componente Zenit de la trayectoria de los eventos captados en el telescopio ANTARES por una única línea. Además, el modelo resultante también será utilizado para realizar una primera estimación de la energía del neutrino.

Por otro lado, y a diferencia con los algoritmos utilizados actualmente, en nuestro caso optaremos por utilizar un modelo de red neuronal, ya que, al ser un algoritmo de aprendizaje automático multipropósito, estos modelos son capaces de identificar y realizar la regresión de cualquier tipo de función no lineal. Concretamente, el modelo final obtenido es una red neuronal convolucional capaz de obtener una distribución probabilística de la componente Zenit de la trayectoria del neutrino en base a imágenes realizadas con la información recibida en los fotorreceptores. La capacidad del algoritmo dearrollado ha logrado disminuir el error de estimación de la trayectoria en un 50% comparado con los algoritmos actuales, demostrando así que utilizar un modelo de aprendizaje automático puede suponer una mejor opción para realizar la estimación de trayectorias.

Contents

List of Figures

List of Tables

Part I

Report

Chapter 1

Introduction

This chapter starts with section 1.1 with a brief explanation of the importance of particle physics and the motivation behind this whole thesis. Then an overview of the current state of neutrino detection and reconstruction algorithms is given in section 1.2. After studying the needs on the matter, the objectives are defined in section 1.3 and with those a possible solution to the problem will be proposed in 1.4.

1.1 Thesis Purpose

For centuries, the human being has being capable of observing celestial bodies that surround us with the use of electromagnetic radiation. In the beginning, the visible light spectrum was observed with the bare eye to determine the planets and the stars that make up the solar system. Later, with the invention of the telescope, we were capable of achieving a more in-depth study of those celestial bodies. Nowadays, the telescope has evolved into a complex machine that can receive the whole electromagnetic spectrum (infrared and ultraviolet light, gamma rays, radio waves, etc) for the observation of far-away galaxies. To this arsenal of observation techniques, a new method has been added for the observation of extremely far and massive celestial bodies using the detection and reconstruction of their neutrino's trajectory.

According to the standard model, neutrinos are subatomic particles with no charge and almost no mass, so they are not affected by neither electromagnetic or gravitational fields (neutrinos are only affected by the weak force presents in the atomic nucleus), which makes them travel through the space without being affected, unlike photons or cosmic rays. This property makes them extremely useful for the study of distant celestial bodies. If the trajectory components of the neutrino are known then the body position can be accurately known. This trajectory is described by the Azimuth and Zenith angles as it is shown in Figure 1.1. Moreover, knowing the energy of the neutrino can give a more complete picture of the object that produced it. Although the intrinsic properties of neutrinos make them an extremely efficient carrier of information, they also make them hard to be detected, as normally neutrinos do not interact with any matter.

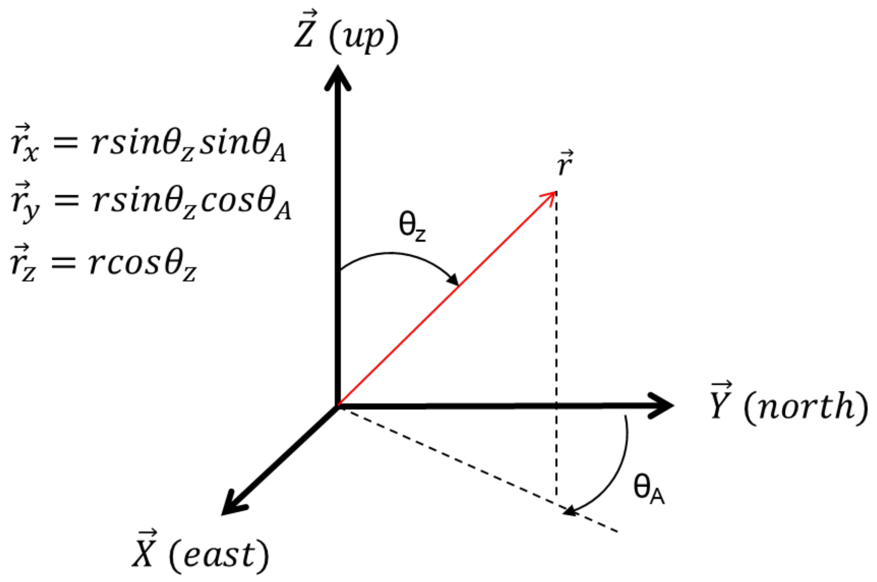

$\vec{Z}$ (up)
$\vec{r}_x = r sin\theta_z sin\theta_A$
$\vec{r}_y = r sin\theta_z cos\theta_A$
$\vec{r}_z = r cos\theta_z$
$\vec{r}$
θ_z
$\vec{Y}$ $(north)$
θ_A
$\vec{X}$ $(east)$

Figure 1.1: Azimuth (θ_A) and Zenith (θ_z) angles obtained from the neutrino's trajectory [1].

1.2 State of the Art

This section will briefly explain the operating principle behind the ANTARES underwater telescope and the algorithms used to estimate the trajectory from the events captured by the telescope. Finally, a full analysis will be carried out on the current state on the subject to determine which will be the objectives and needs of this thesis.

1.2.1 ANTARES Neutrino Telescope and working principle

The ANTARES telescope [4] was built on early 2008 near the coast of France with the purpose of indirectly detecting high-energy ascending neutrinos using the Cherenkov radiation. This electromagnetic radiation is emitted by any kind of charged particle moving faster than the speed of light in a specific dielectric medium where light speed is much lower than the light speed in vacuum, such as water [9]. When a charged particle moves through the medium, it will polarize the molecules around it and as the molecules returns to their ground state, they will re-emit the energy given from the particle as photons in the range of blue visible light, which will propagate in spherical wavefronts. If the moving particle is moving faster than the speed of light in the medium, the spherical wavefronts will get pushed together creating a constructive inference that will lead to a cone-shaped wavefront as seen on Figure 1.2. It should be stated that the neutrino does not create the Cherenkov radiation as it has no charge, however the muons which are created in the rare event that the neutrino interacts with the nucleus of an atom in the medium through the weak force are the ones responsible for this electromagnetic radiation.

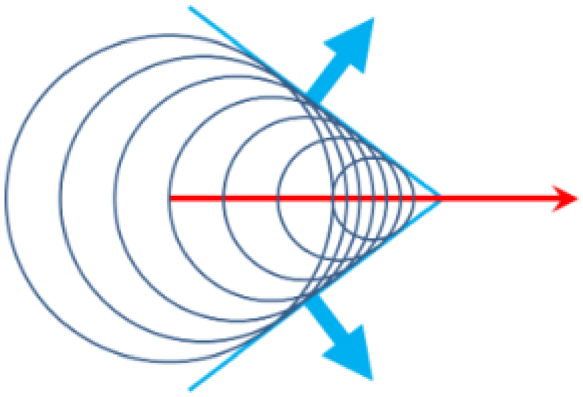


Figure 1.2: Constructive inference creating a cone-shaped wavefront, similar to the *sonic-boom* created by an airplane when it travels faster than the speed of sound [2].

Due to the low level of visible light emitted by the muon, it is extremely important for the detector to be located at 2.5 Km deep in the ocean, where no light is present and the usage of extremely sensitive photomultipliers is possible. These photomultipliers or optical modules are able to measure the amplitude and time of arrival of the photon cone created by the muon to a nanosecond scale.

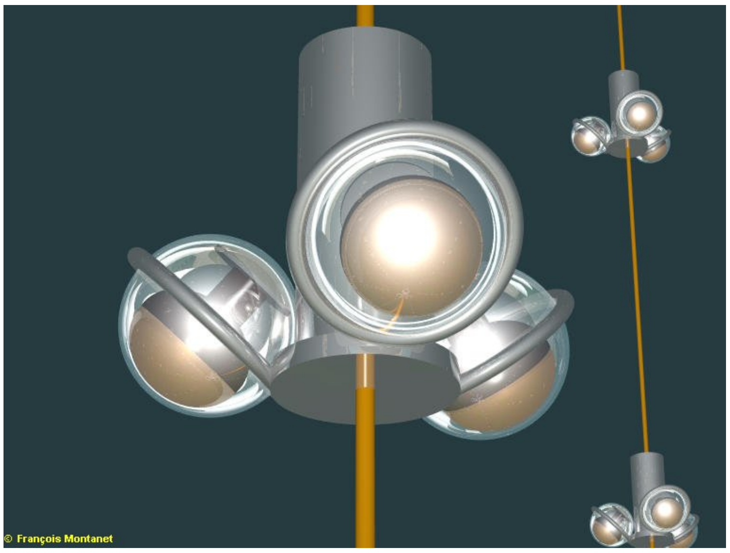


Figure 1.3: Close view representation of a line floor composed by three optical modules [3].

To be able to record events in a wide area and have enough data to estimate the trajectory, the telescope is composed of 12 vertical lines that rise 500 meters over the sea floor (Figure 1.4). Each one of those lines have 25 floors with three photomultipliers in each one of them pointing 120º one from another and positioned 45º below the horizontal axis to enhance the detection of ascending neutrinos (Figure 1.3).

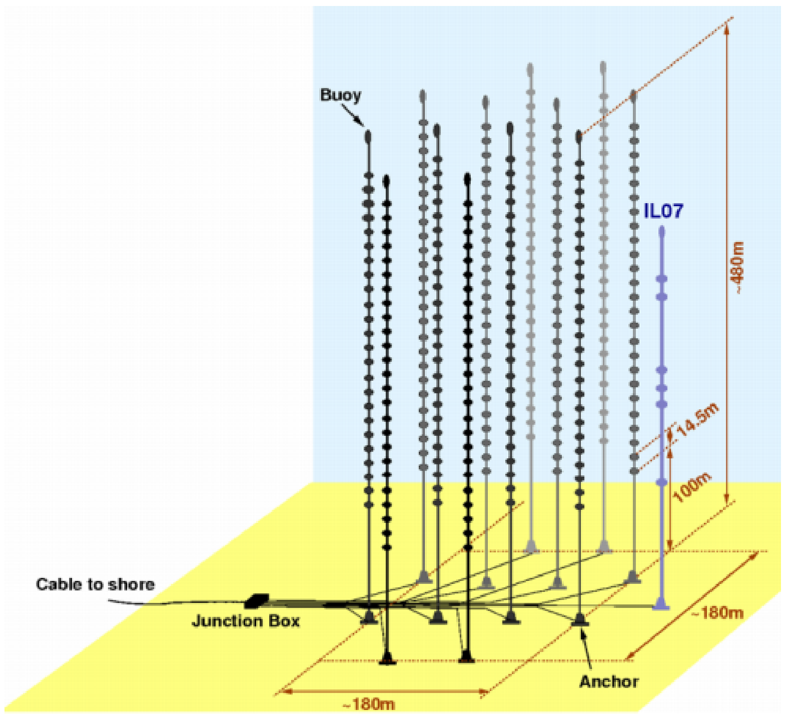


Figure 1.4: Schematic of the ANTARES telescope [4].

1.2.2 Current estimation algorithms

As the ANTARES telescope has several different lines far from one another, the majority of the events are registered only by one line (single-line events), while other, more energetic events are registered by multiple lines (multi-line events). To estimate their trajectory, two algorithms where chosen as standards for the ANTARES telescope, the AAFit algorithm uses the information from multiple lines to estimate the Azimuth and Zenith components of high-energy neutrinos. On the other hand the BBFit algorithm is capable of accurately estimating the Zenith component for low-energy muons in both single and multi-line events. Although both of these algorithms have a high accuracy and reliability when fed with multilineal events, the BBFit algorithm performs very poorly for Zenith estimation on single-line events, and as these events represent a high percentage of the total amount registered by the ANTARES telescope, it is a **priority to enhance or develop a new algorithm that can accurately estimate the Zenith component of single-line events**.

Currently, the Astroparticle Physics research group at the UPV is working on developing a new algorithm that is capable of improving the prediction accuracy of these events using machine learning algorithms. Their current approach is a fully-connected neural network which is fed using preprocessed data and an *off-line* convolution, which is explained with greater detail in section 3.1. Using this approach, the group was able to beat the BBFit prediction's capability and is planning on applying the same principle for the estimation of neutrino's energy, a characteristic that none of the current algorithms have.

1.3 Thesis Objective

As explained in subsection 1.2.2, there is a need for improvement in the estimation of the Zenith component in single-line events registered in the ANTARES telescope. Thus, the main objective of this thesis will be the **development of a machine learning algorithm to estimate the Zenith component** in collaboration with the Astroparticle Physics research group. Using their prepossessing code as a foundation and **optimizing** it for future work [10].

Furthermore, to continue with the current line of work of the research group, the proposed architecture for the Zenith estimation algorithm will be applied to make an initial energy estimation algorithm.

1.4 Justified solution

In the realm of machine learning algorithms, there is a wide variety of algorithms solutions that could be used in this thesis, but as the output of the model should be a numerical value, only regression algorithms shall be considered. In this category, multivariate regression, random-forest an support vector machines should be taken as possible solutions.

1.4.1 Neural Networks

Although some of these lines of work may be interesting and even yield the best solution possible, the algorithm that has been chosen for this project thesis are **neural networks**. These algorithms work by the segmentation of the problem in smaller, simpler units called **perceptrons**. Each one of them having their own weights (ω_i) and biases (b), which are used to create a lineal combination of their inputs (x_i) to get an output ($\hat{y}$) by means of a non-linear activation function (σ) selected by the programmer (1.1).

$$\widehat{y} = \sigma[\sum_i(\omega_i \cdot x_i) + b] \tag{1.1}$$

At first glance, one could argue that this algorithm will yield the same results as multivariate regression, and if the analysis is done from a unitary standpoint it is, as the activation function of a perceptron in (1.1) is the same one used by multivariate regression (except for the non-linear activation function). But as the name implies, when using multiple layers consisting of several perceptrons (hidden layers), each one with its corresponding weights and biases, neural networks are capable of identifying and approximating complex non-linear functions [11, 12]. The regression capabilities of a network will be highly dependent on its architecture, which is defined by the number of hidden layers and perceptrons in each layer. This property of neural networks make them extremely useful **multi-purpose algorithms**, which is the reason why they have been chosen for this thesis.

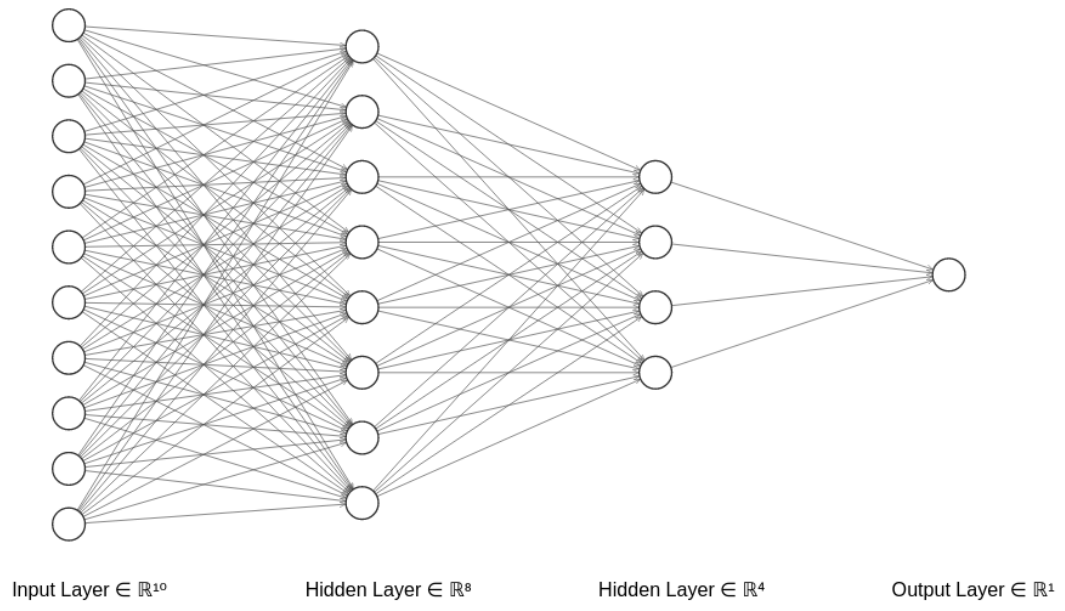


Figure 1.5: Sample architecture with ten inputs and one output. The network is made of two hidden layers, each one of them consisting of six and four perceptrons. Each layers feeds the next one until the final solution is given by the output layer.

To identify and approximate the functions and relationships between the input data (x_i) and the expected output (y), the main goal of the network shall be to **minimize a loss function** $J(\hat{y}, y)$ that tracks the error of the algorithm when finding the function between the input and output data. For regression problems, mean square error can be used as the loss function (1.2). As $\hat{y}$ will be a function of **all the features** of the network, finding the global minima of the loss function even in a simple network as Figure 1.5 using pure function optimization techniques would be impossible, as the net is made out of hundreds or even thousand of weights and biases, so other numerical methods should be used.

$$J(\hat{y}, y) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{1.2}$$

To fine-tune the parameters that minimize $J(\hat{y}, y)$, after computing a **forward-pass** to calculate $\hat{y}$ and $J(\hat{y}, y)$, the algorithm will compute the **backpropagation** [13] of the loss functions through all the features (weights and biases or θ) of the network. Meaning that it will compute the derivative of the loss function with respect to those features and they will get updated according to the results of an optimization algorithm, such as gradient descent in equation (1.3). Other, more complex optimization algorithms may be used, such as Adam [14] or Adadelta [15] optimizers, but all of them work with the same goal, **finding the values for the features that minimize the loss function as much as possible**.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{1.3}$$

1.4.2 Convolutional Neural Networks

As stated in subsection 1.2.2, the research group is currently working on feeding pre-processed images with an off-line convolution that feeds the fully-connected layers of the neural network. This convolution is applied to enhance and extract the details and features of the image by multiplying each one of the image's pixels and its neighbouring pixels by a matrix, also known as a kernel, as shown in Figure 1.6. Each kernel is defined by its size and the numerical values of the matrix, which are known as the kernel's weights, which will determine the extracted features of the convoluted images, for example in Figure 1.7 a predefined canny kernel is applied to extract the edges of objects in the image.

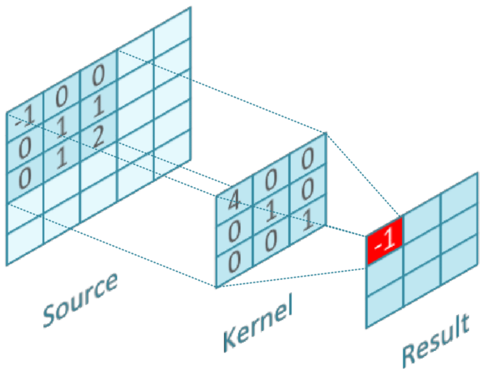


Figure 1.6: Visual representation of the convolution operation of an image [5].

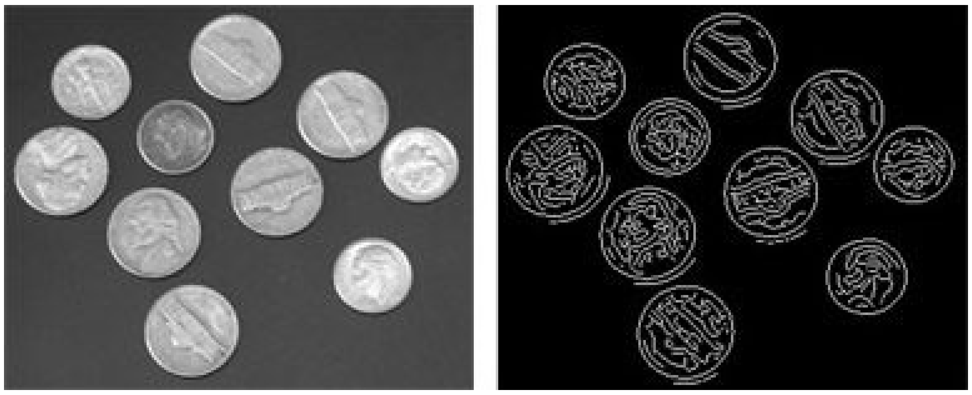


Figure 1.7: Canny Filter applied for edge detection [6].

Although this approach could lead to good results if the kernel's weights are fine-tuned, selecting those weights by hand can be almost impossible. Luckily, the convolution can be made *on-line* and the weights can be optimized by the network using Convolutional Neural Networks or CNNs. These networks are made out of two separate parts: convolutional layers and fully-connected or dense layers (which are the same ones explained in subsection 1.4.1). The convolutional layers apply a series of kernels through convolution operations to the input image, generating new image's channels and giving more *depth* to it. Once the features of the images has been extracted, the convolution operation is followed by a pooling operation to reduce the XY components of the image as the location of the features becomes irrelevant [7]. Pooling layers become extremely important when dealing with big input images, as they greatly reduce the number of trainable parameters, thus reducing the computational requirements of the network.

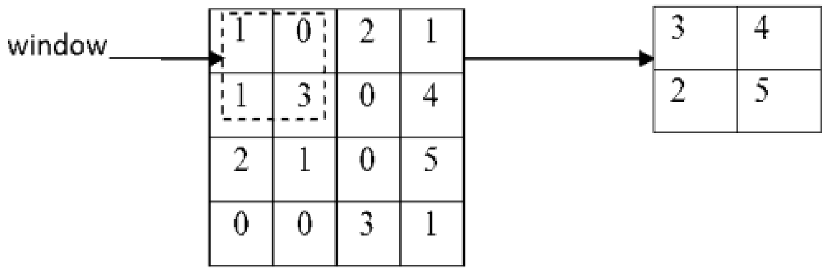


Figure 1.8: MaxPooling operation performed with a 2x2 window, the resulting pixel from each window corresponds to the highest one. The original 4x4 image has been reduced to a 2x2 image [7].

Convolutional layers can be applied in series until the final image is flattened (converted to a one-dimensional vector) and fed to the dense layers of the model, which are the ones responsible for obtaining the final result of the network. As the convolution operations are made *on-line*, the weights of the kernels are backpropagated and optimized the same way as perceptron's weights to minimize the loss function.

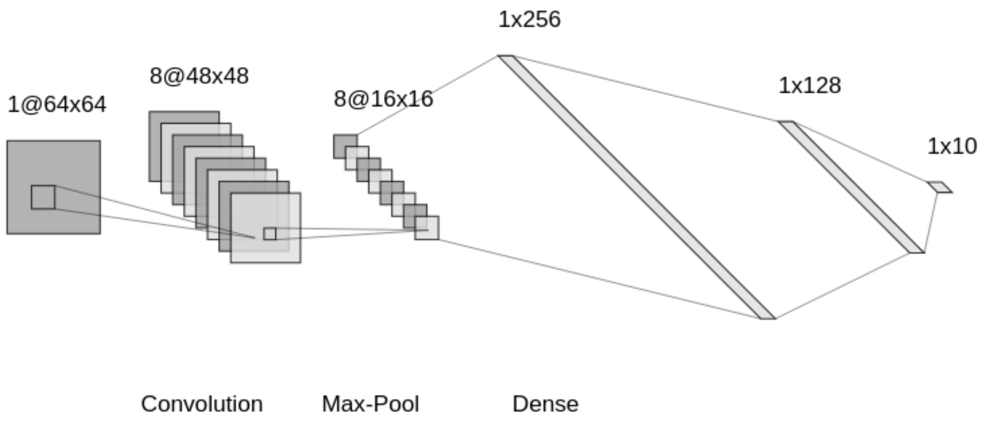


Figure 1.9: Sample CNN aquitecture comprised of one convolutional layer which applies eight 16x16 filters and a 2x2 Max-Pooling layer. The output image is flattened and fed to two hidden layers dense network.

1.4.3 Mixture Density Networks

Finally, to be able to qualify the results of the model and estimate the standard deviation (or error) of each output of the network, the model will include a Mixture Density Network [16] for its final stage. These kind of networks do not only output the numerical result of the regression, but instead the result is a normal distribution, with its corresponding mean (μ) and standard deviation (σ). To achieve a good estimation for both of these components, the algorithm will maximize the probability that its results are inside the normal distribution of the output. Or in other words, the model will try to maximize the probability density function (PDF) given by the equation (1.4). But as all the optimizers are built to minimize a function, the loss will be defined as (1.5), therefore minimizing the negative logarithm of the PDF will be the same as maximizing it.

$$\phi_i(\mathbf{y} \mid \mathbf{x}) = \frac{1}{(2\pi)^{1/2}\sigma_i(\mathbf{x})} \exp\left\{ -\frac{\left\|\mathbf{y}_i - \boldsymbol{\mu}_i(\mathbf{x})\right\|^2}{2\sigma_i(\mathbf{x})^2} \right\} \tag{1.4}$$

$$J(y,x) = -\ln\left\{ \sum_{i=0}^{n} \phi_i\left(\mathbf{y}_i \mid \mathbf{x}_i\right) \right\} \tag{1.5}$$

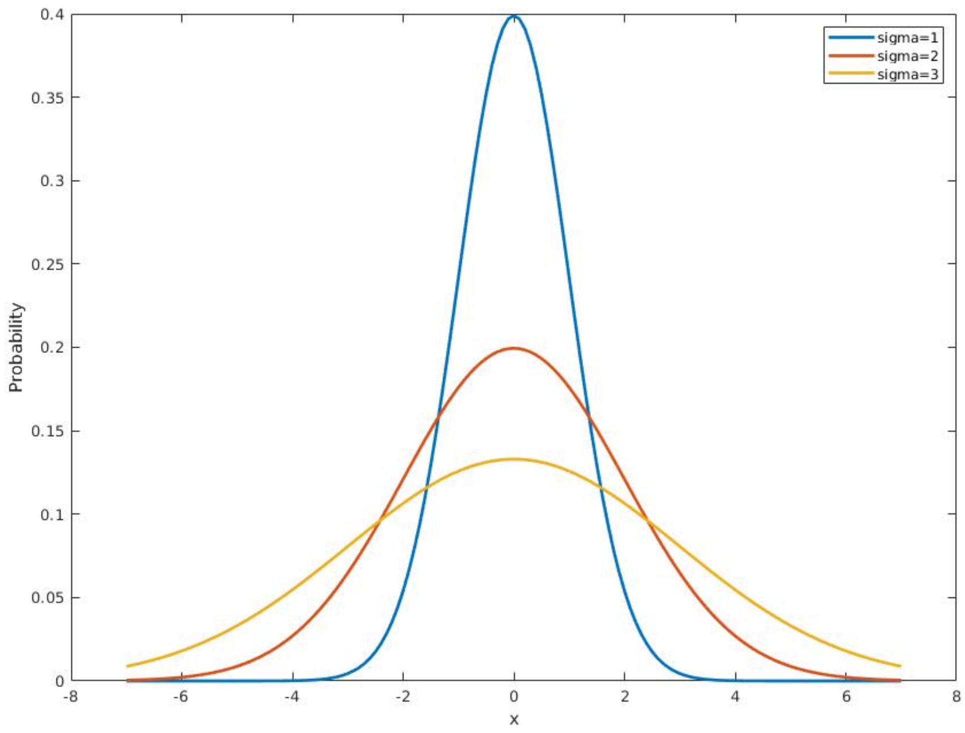


Figure 1.10: Probability density function plot for distributions with $\mu = 0$ and different σ. A lower σ results in a more centered distribution, hence it is more likely for the output value to be equal to μ.

1.4.4 Final Model Structure

As stated in subsections 1.4.2 and 1.4.3, the model that will be used for Zenith regression will be based on a Convolutional Neural Network with a Mixture Density Network, as it is shown in Figure 1.11. Using this configuration, the model will output a normal probability distribution where μ will represent the most likely Zenith value with its corresponding deviation or σ, which will provide an error or prediction quality estimation. The plot in Figure 1.10 demonstrates how the value of σ affects the likelihood of the real zenith component being equal to the μ output.

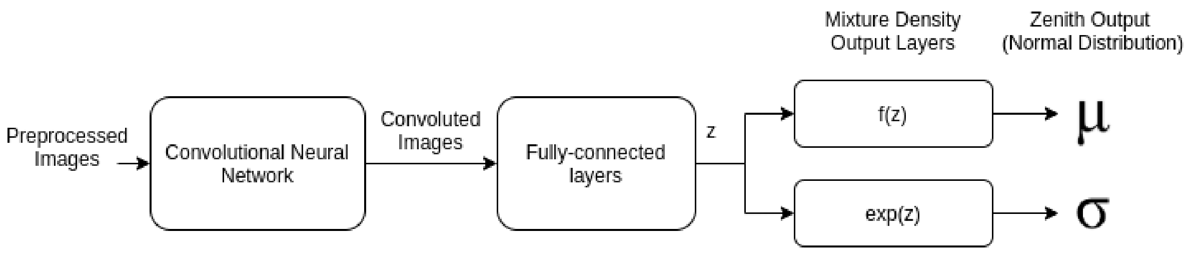

Mixture Density Output Layers
Zenith Output (Normal Distribution)
Preprocessed Images
Convolutional Neural Network
Convoluted Images
Fully-connected layers
z
f(z)
exp(z)
μ
σ

Figure 1.11: Block diagram of the Zenith regression structure

Once a proper model is chosen for the Zenith regression, it will be used for energy estimation model, however it will require a distance to event variable to work properly, as according to the inverse square law, the energy that arrives to the photoreceptors (information that is inside the input image) depends on the energy at the source and the distance between them. So, if the distance is unknown, the model will not tell the difference between an high-energy event far from the line, and a low-energy event close to it. Therefore, as an intermediate step for the energy estimation model, a new model will be defined that will output the distance from the line to the event (d_c) using the image and zenith component as inputs. A visual representation of this new defined d_c magnitude can be found of Figure 1.12.

Once all of these variables have been correctly estimated, they will be fed to another model whose objective is only to estimate the energy having both the Zenith and d_c components. The model will behave exactly as the Zenith one, but will output the normal distribution of the energy of the event.

It is important to state that the three models will be trained and evaluated separately, and the architecture used for all of them will be the same, the difference will be the inputs and outputs of each one.

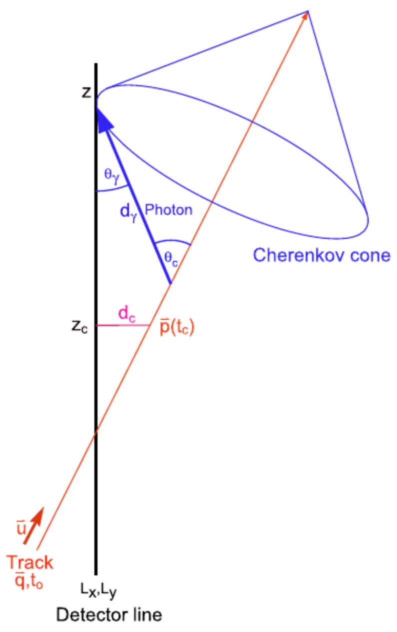


Figure 1.12: Illustration of d_c used to represent the distance from line to event which will be estimated by the distance regression model.

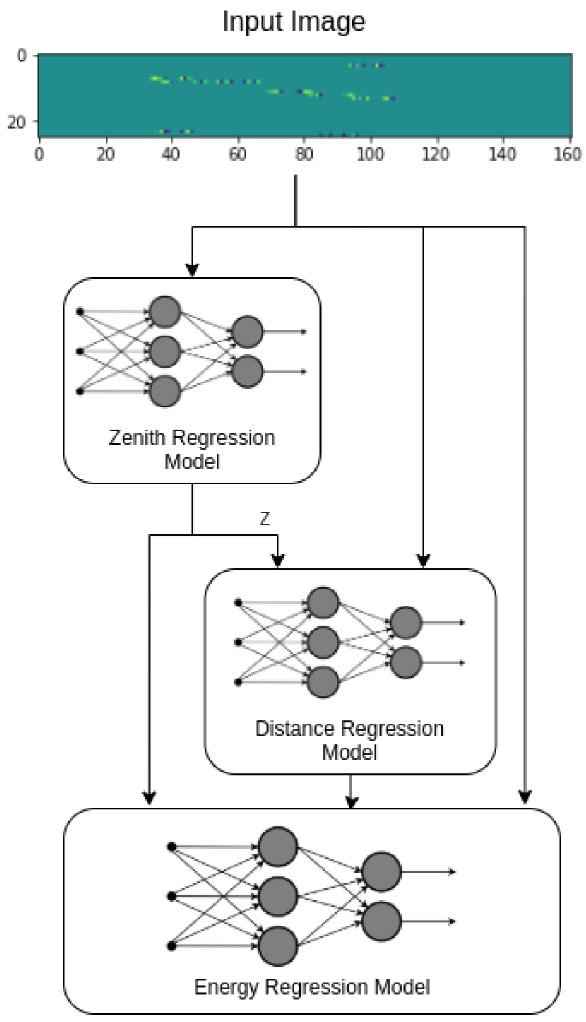


Figure 1.13: Block Diagram illustration of the final model for energy estimation.

Chapter 2

Materials

2.1 Dataset

As the adopted machine learning solution for this thesis falls under the supervised learning category, the dataset that will be used to train the networks, it is required to have both the input and output values for each event, making it impossible to train the models using the real events registered at the ANTARES telescope, as the real components of these events are unknown and can only be estimated using the algorithms described in 1.2.2. If this thesis objective is to beat the capabilities of those algorithms, the output data from them can not be used to train the model, as it will perform equally or worse than those algorithms, but will never outperform them if the output data is estimated using those.

To be able to properly train and compare the new model with the other algorithms, the dataset used through this thesis will be simulated using statistics from real events [17]. Although the simulated data will not include possible noise and uncertainties which are present in real events at the ANTARES telescope, it serves the purpose of comparing the thesis's model with current state of the art algorithms, as the dataset includes both the simulated data and the predictions from those algorithms.

2.2 Software

For the creation of the different preprocessing scripts and learning models it was decided to used python as the programming language, which has converted as an standard for the creation of projects of automatic learning, due to the fact that it delivers the best ratio between ease of programming and the speed at which the interpreter can execute the code. Furthermore, the language offers libraries optimized for the creation of this type of projects, such as *Numpy* for matrix calculation, *pandas* to read and store the simulated dataset and the *multiprocessing* library for the easy and convenient parallelization of the code.

Apart of this libraries, it was necessary to choose a framework to declare and train the different neural network models. At this point, it could be chosen between tensorflow, developed by Google, and pytorch, developed by Facebook. Despite being tensorflow a more mature framework that allows the declaration of models using high level APIs and also the declaration of models in a generic way, finally pytorch was chosen as it allows to make a programming of the models at a lower level in a clearer and more elegant way, which enables to declare the models having an absolute control and understanding over them. Moreover, pytorch allows and easy parallelization between different GPUs, so in future works this option could be easily implemented.

2.3 Hardware

Optimizing the previous preprocessing script from the research group is one of the main objectives of the thesis, and so it is important to state which are the characteristics of the PC used to develop and carry out the benchmarks presented in section 3.1, which are presented in the Table 2.1.

Operating System	Manjaro Linux 64-bits
Central Processing Unit (CPU)	AMD Ryzen 5 2600X @ 3.6GHz
Random-access Memory (RAM)	16 GB
Interal Storage	500 GB SATA SSD
Graphics Processing Unit (GPU)	NVIDIA Geforce 1060
GPU Memory (VRAM)	6 GB

Table 2.1: Characterics of the development PC

This PC was used for development and training until the Zenith Regression models grew to a point where a more powerful GPU was needed. So a server solution was used in order to train the final models, whose characteristics are found in Table 2.2.

Operating System	Ubuntu 20.04 LTS
Central Processing Unit (CPU)	AMD Ryzen 5 3600X @ 3.6GHz
Random-access Memory (RAM)	32 GB
Interal Storage	500 GB SATA HDD
Graphics Processing Unit (GPU)	NVIDIA Tesla K80
GPU Memory (VRAM)	16 GB

Table 2.2: Characteristics of the server PC to train the final models

Chapter 3

Methodology

3.1 Event Prepocessing

To feed the event data to the different neural network models, the first step is to compress all the information from the photomultipliers of the line into a simpler image that can be read by the model. This task is handled by the preprocessing scripts of the project, whose objective is to read the raw data from the simulation files and output three different datasets to train, validate and test the models.

3.1.1 Working Principle

The working principle behind the preprocessing code is to codify the events into RGB images, so that the resulting image contains the information of each photoreceptor in the line, where the x axis in the image represents time in nanoseconds, the y axis corresponds to each one of the floors of the line, and each channel of a RGB pixel is the amplitude received by each one of the three photoreceptors of the corresponding floor in that particular timeframe. Thus the resulting image will have dimensions $[25, N_t, 3]$, where N_t is the timeframe ratio, that can be selected by the user in the preprocessing script. The research group selected this ratio to be 161 nanoseconds which yields the best results possible without having excessive computational requirement, therefore the final dimensions of the processed images will be $[25, 161, 3]$. It should be noted that to smooth the resulting image, a gaussian regression filter is applied to each one of the 75 photomultiplier's channels of the line. Figure 3.1 serves as an example for the gaussian filter application and also aids to corroborate the selection of the N_t parameter, as no useful information is present passed 200 nanoseconds after the reception of the event.

3.1.2 Previous Script

The research group created a *python* script whose job is to preprocess all the events in the simulation dataset. Firstly, the script separates the 2550 initial files into three different

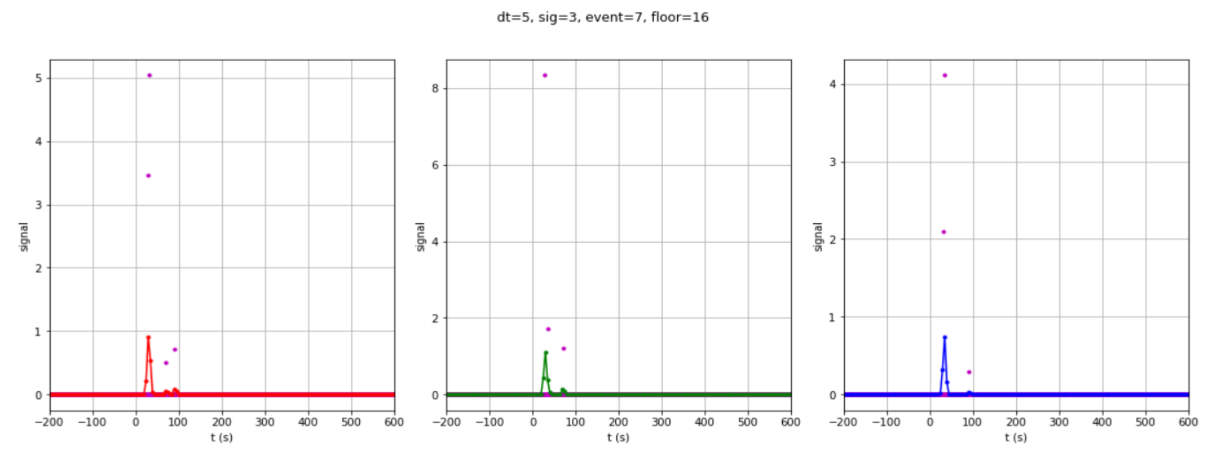


Figure 3.1: Comparison between the original signal (purple dots) and the result of the corresponding gaussian regression filter applied on the 16th floor [8].

different datasets with the same number of files, by doing so, each file contains less events than before. After separating the files in different folders, the methods described in subsection 3.1.1 are applied to each event, obtaining its corresponding RGB image.

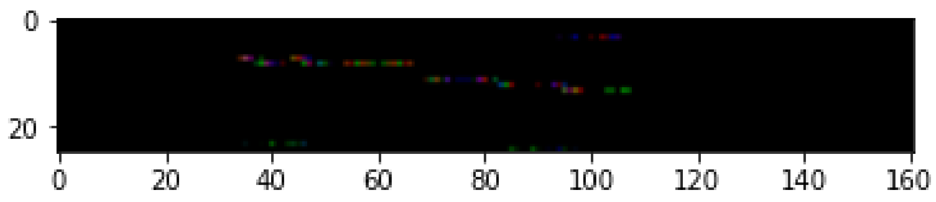


Figure 3.2: Sample event RGB image (after gaussian regression).

As seen on the sample event in Figure 3.2, it is difficult for a human to find any features or details in the image, and for the proposed fully-connected neural network it will be hard too. So after obtaining the RGB image, a new dimension is added to it to fit the off-line convolution applied to the grey scale of the image. This convolution operation uses a (2×10) kernel to highlight the features and details of the event by making the final image that is fed to the fully-connected layers to have dimensions $[25, 161, 4]$, where the first three channels will correspond to the value of the RGB image and the fourth is the result of the convolution operation.

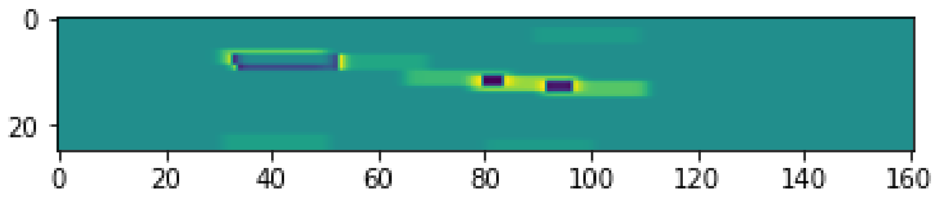


Figure 3.3: Result after applying the convolution to the sample event.

Once all the events have been processed, the images are normalized or "scaled" to achieve the same range of values on all the input pixels of the neural network. This step is extremely important as if no normalization is applied to the image, the algorithm will give more importance or "weight" to events that took place closer to the lines or are more energetic, therefore the amplitude picked up by the photomultiplier is much higher than other events [18]. The final output of the script are three different folders with Numpy's compressed array format or ".npz".

3.1.3 Optimized Script

Although the prepossessing script provided by the research group carried out the task successfully, it was extremely time consuming due to the high amount of data that it was supposed to process. The initial dataset contained 2550 compressed files, so using only one thread as in Figure 3.4 meant that it would take around 100 hours for the script to process all the data on a powerful server CPU.

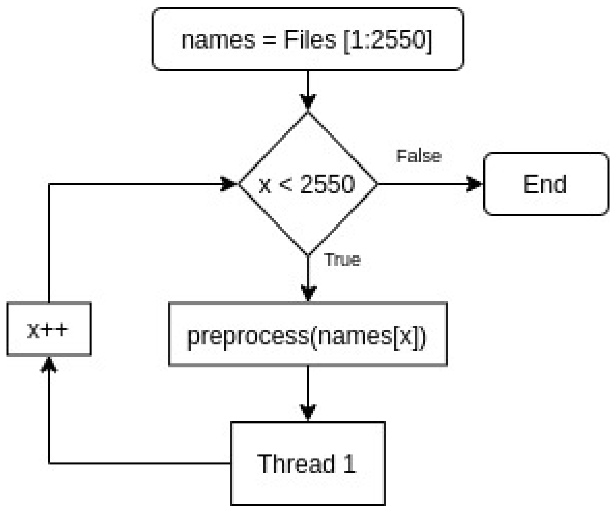


Figure 3.4: Preprocessing loop using a single-threaded process

The first step to reduce the running time of the script was to lower the number of files which were read, meaning that instead of separating the files into three different datasets at the beginning of the process (as stated in subsection 3.1.2), this step would take place at the end of the script, just before normalization. This simple change meant that the script would not have to be reading 7650 files in each step, but only the 2550 initial files, reducing running time by around 66%.

The second and most important step for the optimization of the script is **parallelization of the code**, so that the machine that is running the preprocessing script can utilize all the available threads in the CPU instead of just one of them. For this purpose, all the loops of the code whose objective is to read or write files where written again as functions that are called by **parallel loops** created by the *multiprocessing* library. This way, if the user decides to allocate n threads for the execution of the script (n is selected in the options of the script), the loops will be able to run their code n times faster when compared to the previous script. In Figure 3.5 it can be appreciated that the library comes with a high-level of abstraction, as synchronization and memory-management of all the threads is handled internally by the *Pool* object of the library. This example only shows the parallelization of the image processing step, but it was applied to almost all of the steps of the script.

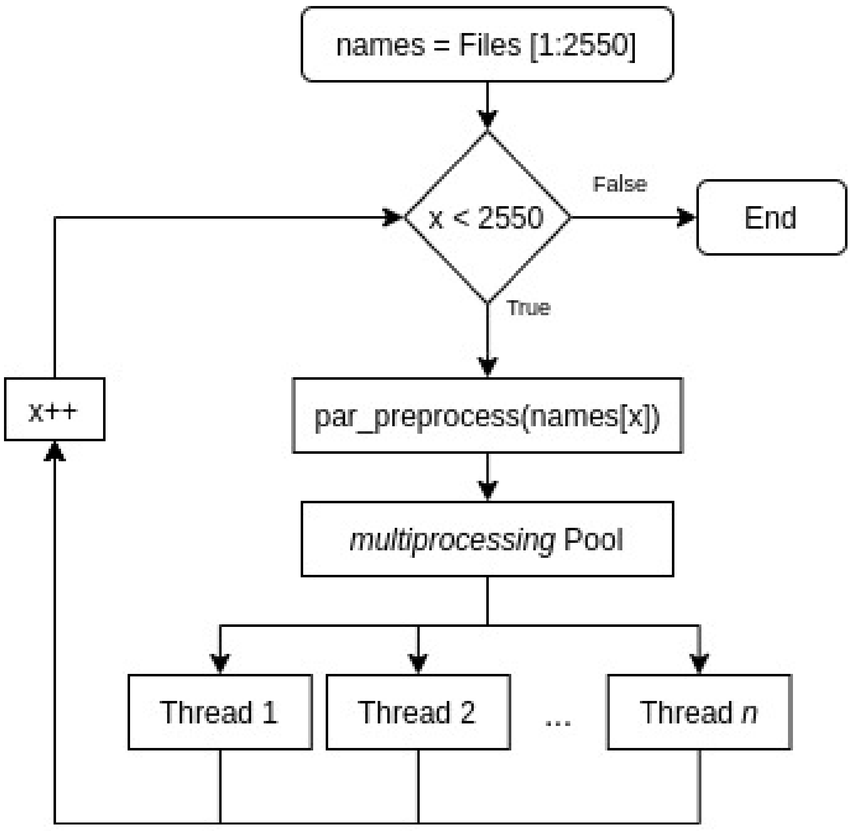


Figure 3.5: Preprocessing loop structure using a multi-threaded process.

After implementing both of these optimizations, the running time was cut down to only 10 hours when using 10 threads of a consumer-grade CPU, making the process of implementing and adding new features and changes to the prepossessing script easier than before, as there is less time wasted between implementing and testing the new features and changes.

As it will be explained in detail in further sections, due to the type of dataset that is being used to train the model (which contains a large number of small events), it becomes extremely important that the model can get access to the events as fast as possible for the GPU not to waste time. This can not be achieved using the output default format of the script (Numpy's compressed array or ".npz") , as its compressed state makes it too slow to be read constantly. To solve this problem, a benchmark was carried out between this file format a more suitable HDF5 format [19]. The purpose of the test was to find out the required time to load the file's contents and convert them into a *Pytorch* tensor, which emulates the workflow of the model's dataloader. The results of the benchmarks on Table 3.1 show that using this new file format can greatly reduce the data loading time, as each file takes close to 15 milliseconds less to be read with this format. It may not seem as much, but considering that for an epoch to take place the model should read more than a 1000 files, and 50 epochs are needed to train a solid model, the training time may be reduced by 2 hours just by changing the file format.

File Format	Number of files read	Mean time to read one file
.npz	120	18.00 ms
.hdf5	120	5.46 ms

Table 3.1: Results from file format benchmark (with forward-pass emulation)

Furthermore, some other minor changes and options were added to the code, one of the most significant is the option to generate a dataset for predicting both the Azimuth an Zenith component using RGB images as in Figure 3.2, where the target data include all the three dimensions of the neutrino's trajectory, or the dataset can be optimized for only Zenith and energy prediction, where the images have dimensions $[25, 161, 1]$ as the only channel contains the module of the RGB (Figure 3.6), the target vector only has the energy and the Z component of the trajectory to calculate the Zenith angle. But for both options the *off-line* convolution was discarded, as this dataset will be used for CNNs.

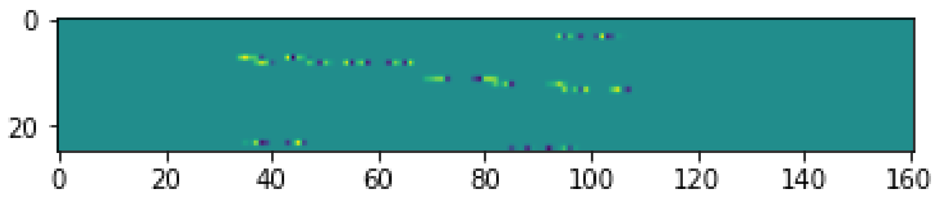


Figure 3.6: Sample event image for Zenith and energy estimation.

3.2 Model Structure

3.2.1 Dataloader

Dataloaders are a crucial part of every machine learning model, as its job is to **load and prepare batches or chunks of data from the dataset to be fed to it**. But in the bast majority of machine learning projects it is not considered at all when frameworks as *pytorch* or *tensorflow* are used, as those frameworks come with their own general-purpose dataloaders. These dataloaders are optimized to work with unitary files, meaning that each event has its own separate file. But as the dataset from the simulation has over one million events, the preprocessing script has to output files with n events each, as creating one file per event would imply a huge waste in hard disk space as each separate file has to allocate a minimum amount of space for its own structural information. If these kind of dataloaders would be considered for using in this project, those would need to open a file with n events just to load one of them, which will not only slow down the loading process, but also could generate a bottleneck if the hard disk bandwidth is surpassed, thus wasting much of the GPU computing power.

Due to this inconveniences, a new dataloader was created for this project to load multiple batches from the same file. For example, if a batch size of 50 and a file size of 1000 events were to be used to train the model, the new dataset would load a file in RAM and get 20 different batches from the same file, reducing the batch loading time and the hard disk's bandwidth usage. To assess the difference between both dataloaders, another benchmark was designed to measure the batch generation time, whose results are shown in Table 3.2 where it has been confirmed that the idle time between the request and arrival of a new batch has been lowered 33.66 milliseconds, reducing idle time in 98%.

Dataloader	**Number of batches generated**	**Mean time to generate one batch**
Pytorch's Default Dataloader	300	34.40 ms
Custom Dataloader	300	0.74 ms

Table 3.2: Results from batch generation bechmark (with foward-pass emulation)

Despite of the implementation of the new dataloader in the script of the model, when the first tests with simple architectures of neural networks were carried out, where the time to do the forward and backward pass was minimum, the GPU's usage was under the 60%. This was due to the fact of having a so small process time of the events, as the neural network was capable of processing the events faster than the hard disk was able to read them, making the hard disk capability of read a bottleneck, preventing the network from training at the maximum capacity of the GPU. It is for this reason that the benchmarks carried out previously have a delay between each read. This is exemplified in Table 3.3, where the benchmark carried out in Table 3.1 is repeated with no delay between each read. During this test, hard disk capacity reach its limit and the results show that both read times rise to levels where there is no considerable difference between them.

File Format	**Number of files read**	**Mean time to read one file (in ms)**
.npz	120	40.09
.hdf5	120	34.73

Table 3.3: Results from file format benchmark (no forward-pass emulation)

For this reason, an option was included in the dataloader's declaration to load the whole dataset into RAM when it is initialized so that the batch can be read directly from it and passed to the model from this high bandwidth memory. In Table 3.4, the dataloaders benchmark is repeated with no forward-pass emulation or delay, and as expected the batch generation time for both previous dataloaders rise while the batch generation time for the custom dataloader using RAM goes to insignificant values. Although theoretically this option is the best one to use, it shall only be restricted to machines with enough RAM capacity and when the GPU usage falls bellow 90%.

Dataloader	Number of batches generated	Mean time to generate one batch
Pytorch's Default Dataloader	300	87.09 ms
Custom Dataloader	300	2.98 ms
Custom Dataloader (RAM)	300	0.025 ms

Table 3.4: Results from batch generation benchmark (no forward-pass emulation)

3.2.2 Zenith Regression Structure

As stated in section 1.4, the model that will be used its a Convolutional Neural Network with a Mixture Density Network at its output for computing the normal distribution of the Zenith component of the event. To find out the best architecture for the model, some static hyperparameters will be set as a foundation to built the final architecture which will be set by other variable hyperparameters.

Static Hyperparameters

- **Input Dataset**: The input images for the network will be obtained through the optimized script described in subsection 3.1.3 with the options for Zenith and energy regression, as such the input images will be similar to the sample event shown in Figure 3.6. For the percentage of events dedicated to train, evaluate and test the network, a standard 70/20/10 percent split has been selected.

- **Number of hidden layers**: In all of the proposed architectures there will be three hidden fully-connected layers, which is more than enough for a normal CNN as the images that will be fed to it will have all the features already extracted by the convolutional layers. The number of perceptrons per layer will depend on the size of the flattened image.

- **Dropout**: For preventing model overfitting, a 25% chance is added for every perceptron to "shut-down" during training, helping the model to generalize the training dataset into the validation and test datasets by making the training process more difficult.

- **Activation Function**: All of the hidden and convolutional layers will use the same activation function, which is set to be the ReLU activation function. This function is used as its simplicity reduces the backpropagation computational costs. The output layers of the MDN will use a different activation function, as the standard deviation of a distribution can not be negative or zero, the activation function of that layer shall be an exponential function [16].

- **Loss Function**: As stated in subsection 1.4.3, the loss function shall maximize the probability density function from (1.4), as such the loss function is defined in (1.5).

- **Optimizer**: The Adadelta [15] optimizer has been selected by its impressive adaptive learning rate capabilities.

- **Batch Size**: It defines the number of events that the optimizer takes to update the features of the model. A higher batch size implies a faster training but also less updates and optimizations are applied to the model per epoch. Considering that the training dataset is composed of 800.000 events, setting a batch size of 1000 events will yield the best results in the lowest training time possible.

- **Number of epochs**: Implies the number of times that the model is fed with the whole training dataset. It will be set to 100 epochs but an early-stopping mechanism will be implemented to end the training process if the model is not able to lower the cost function in 15 epochs.
- **Number of models**: For each proposed architecture, three models will be trained and the metrics of each architecture will be extracted from the best performing model.
- **Evaluation Metrics**: To easily compare the different models, three metrics will be used: the value of the loss function, the Mean Absolute Error (3.1) and the Root Mean Square Error (3.2). However, a more in depth analysis of the final models will be carried out in Chapter 4.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.1}$$

$$RMSE = \sqrt{\sum_{i=1}^{n}\frac{(\hat{y}_i - y_i)^2}{n}} \tag{3.2}$$

Variable Hyperparameters

To select the best possible variable hyperparameters, a number of convolutional layers and filters will be selected and then three different models will be trained, each one with a different kernel size. The two best performing kernels will then be trained with more convolutional layers to see if the model can benefit from the extra *image depth*.

Two Convolutional Layers Architecture

The first architecture to be tested will have two convolutional layers which will produce a total of 36 output channels. The selected kernel sizes are the standard (3×3) and (5×5) sizes, also a test will be carried out with the kernel size that the research group was using for its off-line convolution in the preprocessing script, which is a (2×10) kernel. The results after training the three models is presented in Table 3.5 where the (2×10) kernel yielded the best results followed by the (3×3) kernel, which is not surprising as the (2×10) size takes into consideration the non-symmetrical shape of the input image $([25, 161, 1])$.

Net Identifier	Kernel Size	Output Image Size	Loss Value	MAE	RMSE
Conv11	(2×10)	$[7, 35, 36]$	-0.00026	8.87º	13.17º
Conv12	(3×3)	$[6, 40, 36]$	-0.00021	9.24º	13.52º
Conv13	(5×5)	$[4, 38, 36]$	-0.00022	9.52º	13.65º

Table 3.5: Results from training the architectures with two convolutional layers

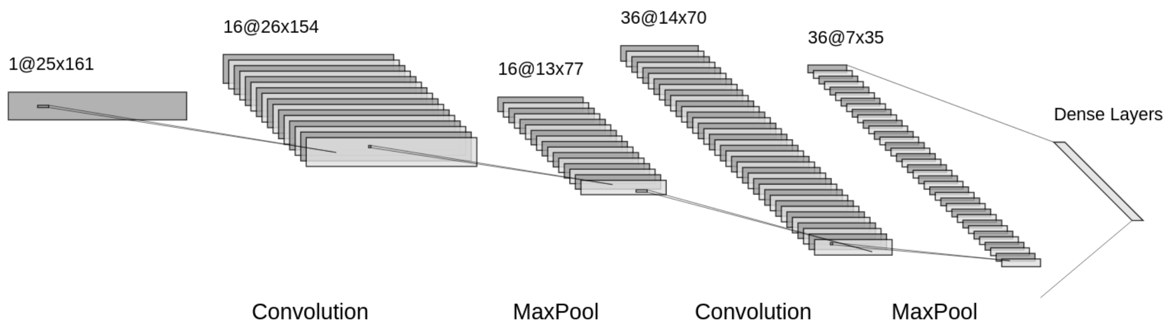


Figure 3.7: Architecture used for net Conv11 using a (2×10) kernel size. Dense layers have been excluded for diagram simplification.

Three Convolutional Layers Architecture

Now that two kernel sizes have been selected from the previous test, a new test will be carried out to study if the addition of more convolutional layer or more *image depth* is going to have a positive impact on the prediction capability of the model. For this new architecture, another convolutional layer will be added giving 64 channels to the output image. Results in Table 3.6 show that both kernel sizes benefit from the addition of the third layer when compared to the results from Table 3.5, but the (2×10) kernel showed such promising results that it will be selected as the kernel size to be used in the rest of the architectures.

Net Identifier	Kernel Size	Output Image Size	Loss Value	MAE	RMSE
Conv21	(2×10)	$[4, 14, 64]$	-0.00033	8.39º	12.47º
Conv22	(3×3)	$[3, 20, 64]$	-0.00027	8.96º	13.17º

Table 3.6: Results from training the architectures with three convolutional layers

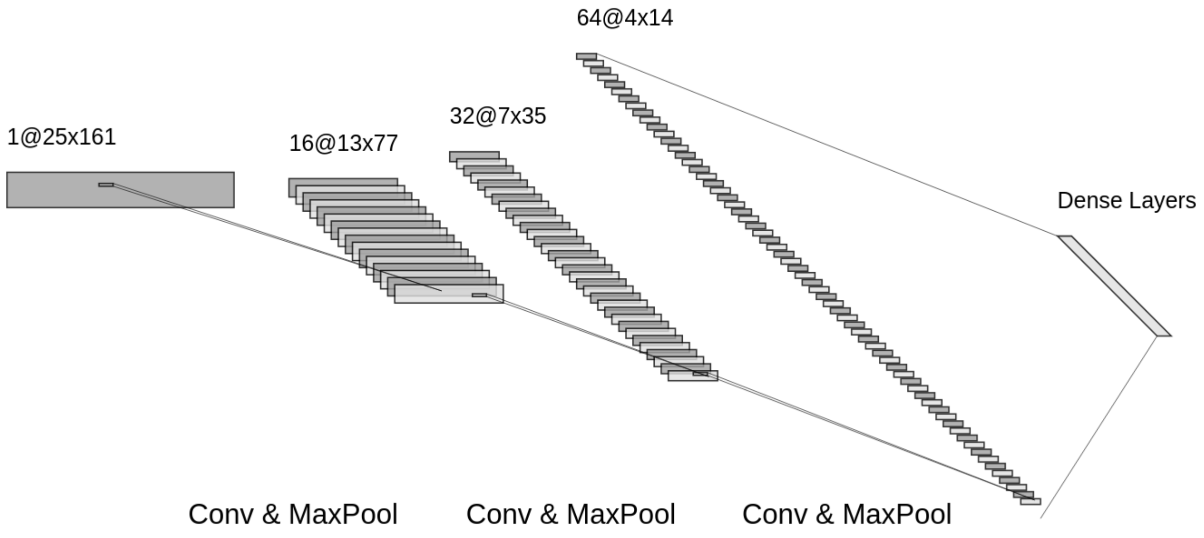


Figure 3.8: Architecture used for net Conv21. Dense and Maxpool layers have been excluded for diagram simplification.

Four Convolutional Layers Architecture

As adding more convolutional layers resulted in better prediction capabilities, a final test was done using the best performing (2×10) kernel and four convolutional layers capable of producing 128 output channels. As the addition of the third layer produced output images with small XY components, this fourth layer will be added but no MaxPooling operation will be applied after the convolution operation as in Figure 3.9, this way more output channels can be produced without reducing the image height and width. Table 3.7 shows the results from all the tests that were done using the previously defined architectures, showcasing that the addition of convolutional layers and image depth yielded the best results.

Net Identifier	Kernel Size	Output Image Size	Loss Value	MAE	RMSE
Conv11	(2×10)	$[7, 35, 36]$	-0.00026	8.87º	13.17º
Conv12	(3×3)	$[6, 40, 36]$	-0.00021	9.24º	13.52º
Conv13	(5×5)	$[4, 38, 36]$	-0.00022	9.52º	13.65º
Conv21	(2×10)	$[4, 14, 64]$	-0.00033	8.39º	12.47º
Conv22	(3×3)	$[3, 20, 64]$	-0.00027	8.96º	13.17º
Final	(2×10)	$[4, 14, 128]$	-0.00041	7.67º	11.92º

Table 3.7: Results from all the Zenith Regression architectures.

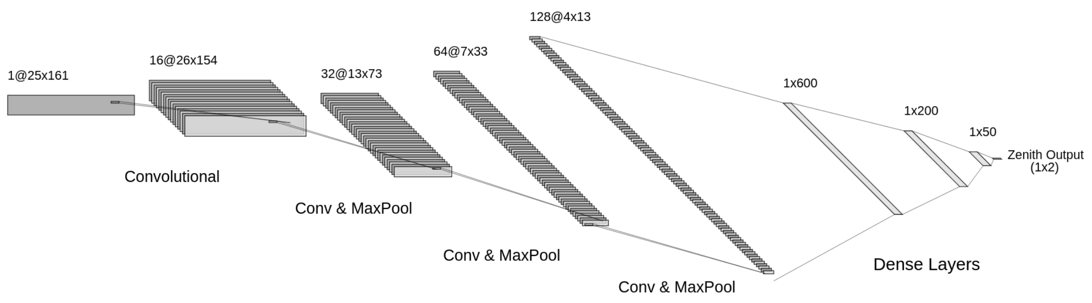


Figure 3.9: Final architecture for the Zenith Regression model.

3.2.3 Distance Regression Structure

At first, the architecture that was planned for using for the distance regression model was going to be the same one that yielded the best results for the Zenith regression, which is the one shown in Figure 3.9. However a alternative solution was proposed by the research group using the same convolutional layers but with fewer perceptrons at the fully-connected layers, as in Figure 3.10. Both architectures were trained and the results displayed in Table 3.8 confirms that for distance regression a more simple approach offers better results, as such, this alternative architecture was chosen for the distance regression model. It should be stated that in order to incorporate the value from the zenith regression model, both values μ_{zenith} and σ_{zenith} are concatenated to the flattened tensor from the images after it is convoluted by all the convolutional layers, so the zenith values are used only by the fully-connected layers.

Net Identifier	Dense Layers Layout	MAE	RMSE
Final	$600 \to 200 \to 50 \to 2$	6.27 m	9.81 m
Alternative	$128 \to 128 \to 32 \to 32 \to 2$	4.05 m	6.79 m

Table 3.8: Results from both distance regression architectures.

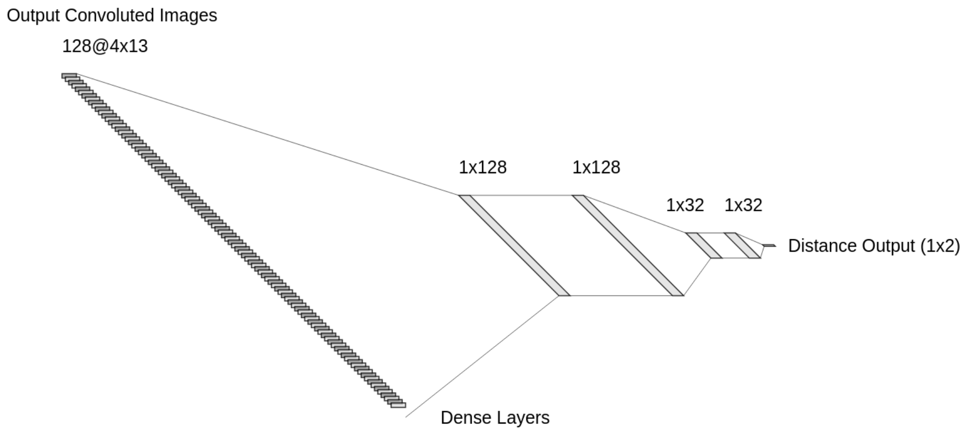


Figure 3.10: Proposed dense layers for the distance regression model. Convolutional layers were omitted for diagram simplification.

3.2.4 Energy Regression Structure

For the creation of the energy regression model, the first step is to closely inspect the distribution of the values that the model will estimate. The histogram in Figure 3.11 shows that there is a big difference between the minimum and maximum energy values (E_t), as the values can be between 5 and 10.000 GeV, and the distribution is skewed to the left. Using the energy values with this distribution resulted in poor training and estimation capabilities as the model would try to fit very different output values and even lead to exploding gradients (the derivatives of the features (1.3) went to infinity due to the big output values and cost function). To relieve these problems, the target values of the network (y and $\hat{y}$) were programmed to be the logarithm of the energy values, which results in a more "normal" distribution of the target values as in Figure 3.12. This will add a new step to the regression model, as the final predicted energy should be calculated as $E_p = 10^{\hat{y}}$. Finally, as the metrics of the model will be calculated using the energy values and not the logarithm of it, it is important to add a new metric which does not get affected by the skewness of the data, which will be defined as the mean relative error or MRE (3.3). Unlike the MAE, that is more affected with bigger values of energy (hence yielding to a greater deviation), the MRE will take into consideration the target value for the energy, weighting the deviation with the magnitude of the real value.

$$\text{MRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{E_p - E_t}{E_t} \tag{3.3}$$

Two different architectures were considered as candidates for the energy regression model, so both the zenith and distance regression architectures were trained using the outputs from the Zenith and distance models as inputs at the fully-connected layers. Surprisingly, results in Table 3.8 show that while the Zenith architecture achieved lower errors, the distance architecture resulted in a lower cost function. Meaning that the selected final architecture for the energy regression model will be the same one that was used for the distance regression one, as it achieved a lower cost value that takes into consideration both μ_{energy} and σ_{energy} outputs from the model, while MAE and MRE metrics only use the μ_{energy} for its calculations.

Net Identifier	**Loss Value**	**MAE**	**MRE**
Zenith architecture	0.0005	957 GeV	2.179
Distance architecture	0.00026	977 GeV	2.41

Table 3.9: Results from both energy regression architectures.

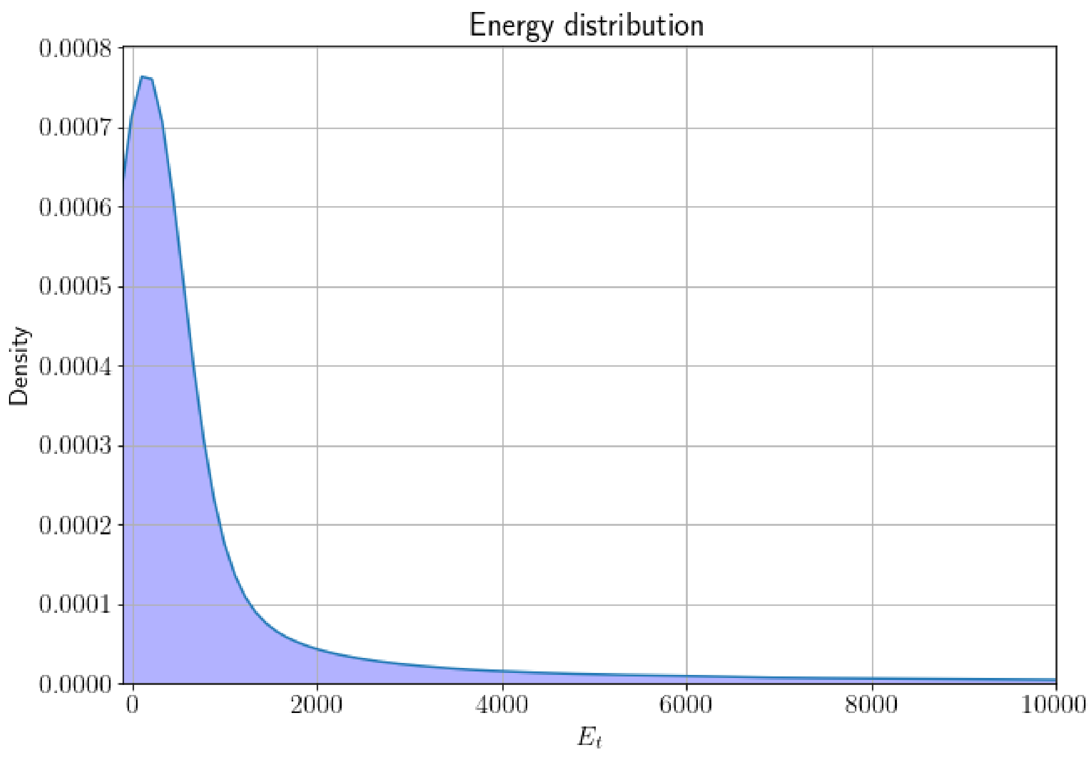


Figure 3.11: Distribution of energy values in the test dataset.

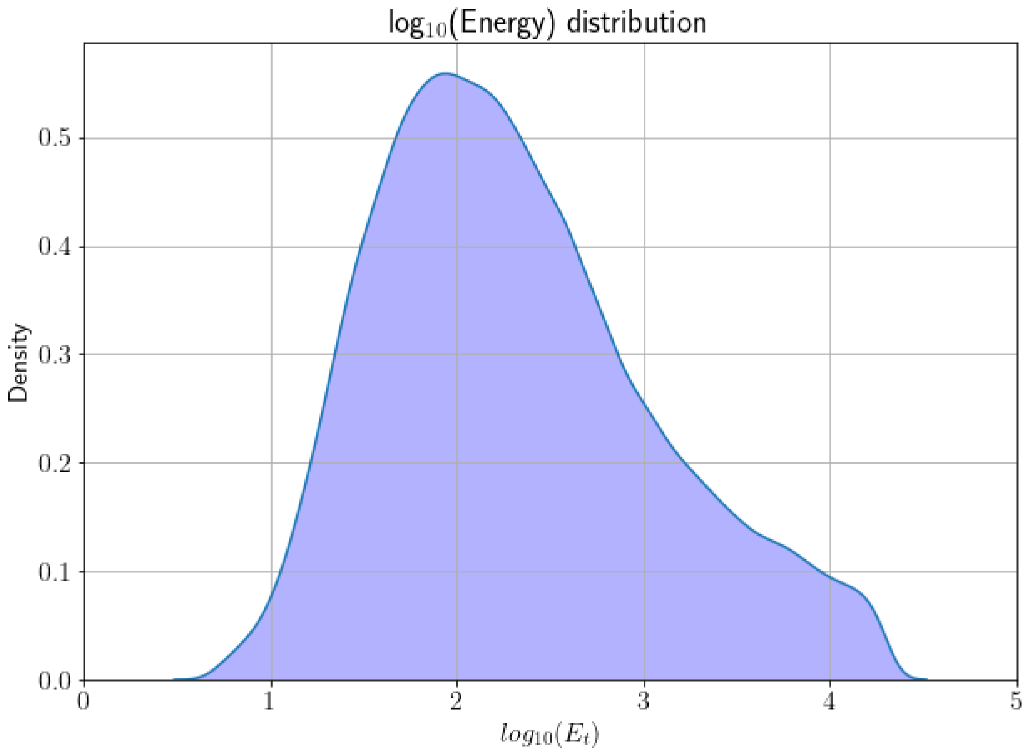


Figure 3.12: Distribution of the log energy values in the test dataset.

Chapter 4

Results

This chapter will show a more in-depth analysis of the results from the final models obtained in the previous chapter. Section 4.1 will cover the results from the Zenith regression model when compared to the BBFit algorithm, section 4.2 will briefly cover the results from the distance regression model as it is an intermediate step for computing the values for the energy regression model in section 4.3.

4.1 Zenith Regression Model

Firstly, an analysis of the overall numerical error comparison will be carried out between the Zenith regression model and the BBFit algorithm. Table 4.1 shows the resulting MAE (3.1) and RMSE (3.2) metrics for both algorithms on the Z component ($Z = \cos(Zenith)$) of the neutrino trajectory, where the new proposed regression algorithm was capable of improving the estimation error by 50%. On Table 4.2 the mean and standard deviation of the Zenith angle are computed for both models as the $\mu \pm \sigma$, confirming that not only the mean error has been improved, but also its dispersion. Finally, Table 4.3 represents the distribution of the Zenith error of the proposed model using 100% of the data and the 50% with the lowest σ at the output of the Mixture Density Network, both distributions can be graphically represented using the density plot in Figure 4.3. Furthermore, the data where the sigma is lower resulted in better prediction quality, verifying that the sigma output of the Mixture Density Network is working as expected providing an estimation of the prediction error as shown in Figures 4.1 and 4.2.

The BBFit algorithm comes with its own reconstruction quality indicator, which is done using the χ^2 test [20]. Figure 4.1 demonstrates that for the cases where the reconstruction quality for the BBFit algorithm is at its best, the new Zenith regression model still manages to achieve a lower error. The overall comparison between the results of both algorithms is presented in the density plot in Figure 4.4 where the Zenith regression model achieved a more linear relationship between the predicted and real values of the Zenith angles.

Error Component	**Zenith Regression Model**	**BBFit algorithm**	**Difference (%)**
MAE Z	0.099 m	0.198 m	50%
RMSE Z	0.1649 m	0.3353 m	50.82%

Table 4.1: Comparison between both Zenith regression model and BBFit using Z component metrics.

Ang. deviation	**Zenith Regression Model**	**BBFit**
Zenith	$7.66^{\underline{o}} \pm 9.12^{\underline{o}}$	$15.5^{\underline{o}} \pm 18.8^{\underline{o}}$

Table 4.2: Comparison mean and standard deviation for the Zenith angle between both Zenith regression model and BBFit.

σ **Percentage**	**Mean**	**Median**	**Q1**	**Q2**
100 %	7.668	9.124	1.889	9.925
50 %	5.535	3.52	1.51	7.183

Table 4.3: Data from the distribution of the error on the Zenith regression model.

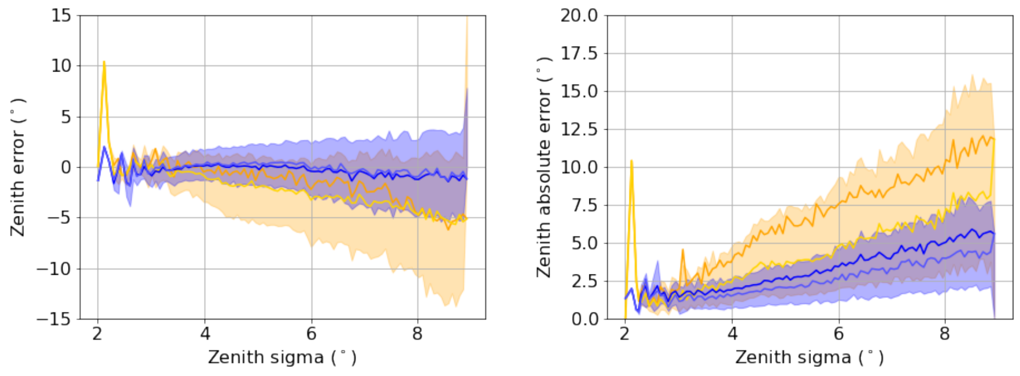


Figure 4.1: Zenith error vs. σ output from the model. Data was smooth using quartiles.

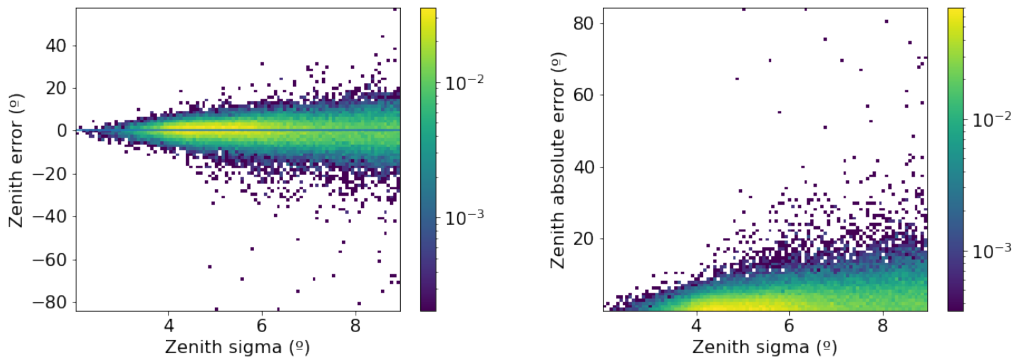


Figure 4.2: Histogram representing Zenith error vs. σ output from the model.

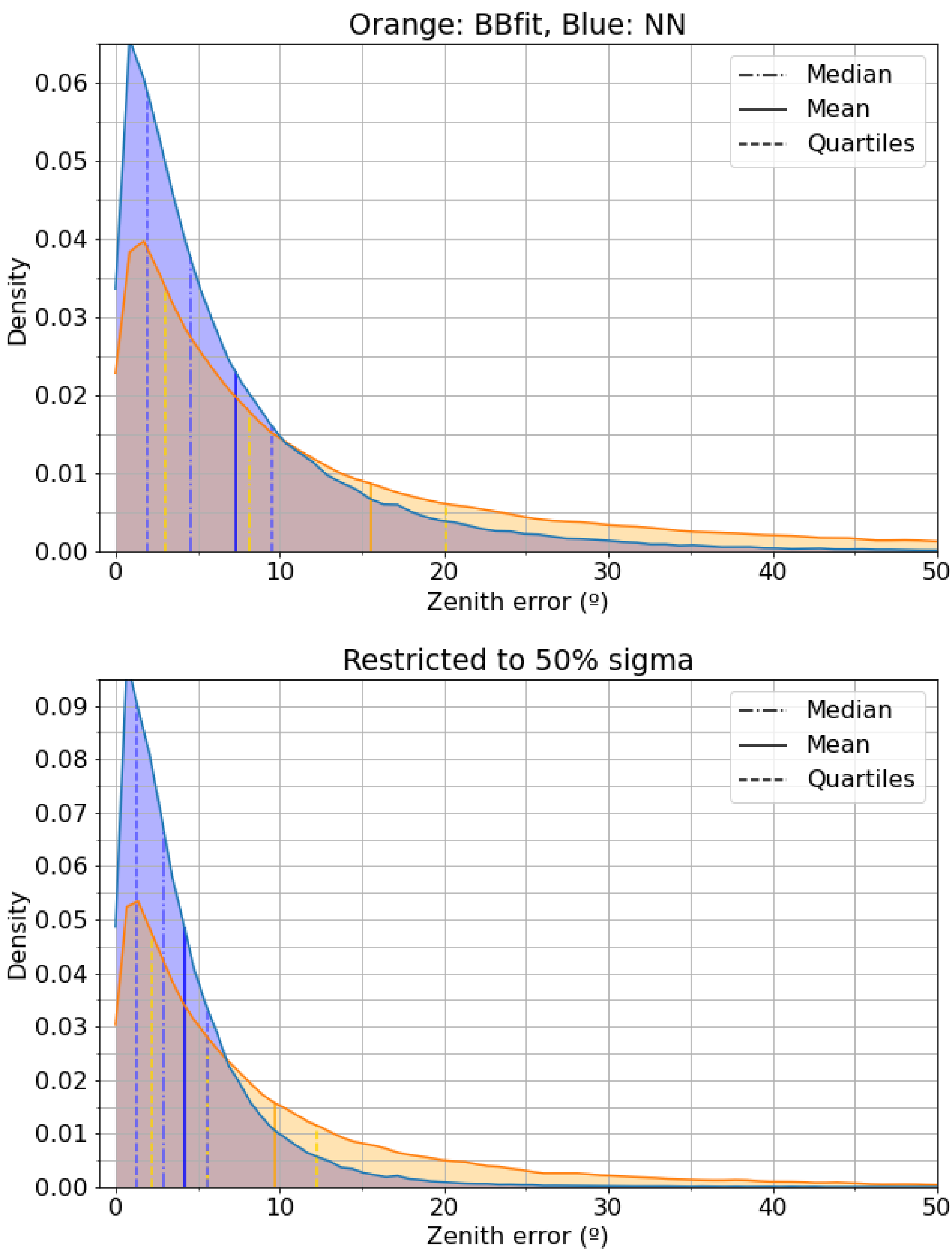


Figure 4.3: Density plots of the Zenith error from both the Zenith regression model (NN) and BBFit algorithm.

4.2 Distance Regression Model

In subsection 3.2.3, it was stated that the mean absolute error of the distance regression model is 4.03 meters, but for this model the distribution of the error shall be taken into consideration as the distribution of the distance in the test dataset is skewed to the left as shown in Figure 4.5, meaning that rare high-distance events will greatly impact the error metrics while other more common low-distance events won't. The metrics of the error are exposed in Table 4.4 and graphically represented in the Figure 4.6, where it is clear now that the error of the model is much lower than the MAE reflected. Finally, to assess the results from the reconstruction quality of this model (σ_d), Figure 4.7 shows that the model is achieving a good approximation of the absolute real error with it.

Magnitude	**Mean**	**Median**	**Q1**	**Q2**
$d_p - d_t$	0.4 m	-0.2 m	-2.05 m	2.59 m

Table 4.4: Data from the distribution of the error on the distance regression model.

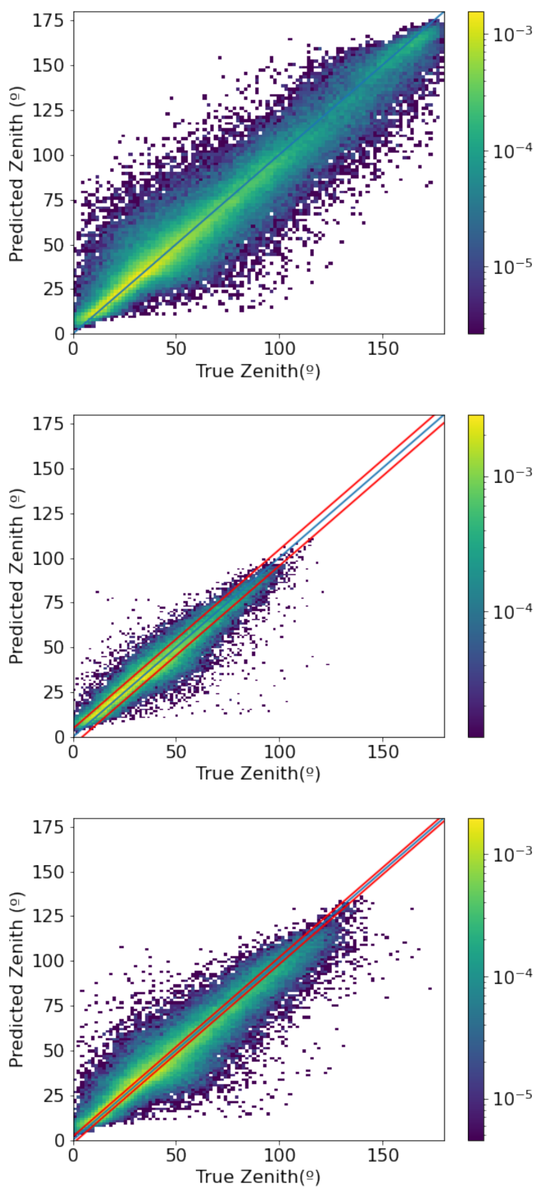
4.3 Energy Regression Model

Although the energy regression model was fed with good estimated input variables from the distance and zenith regression model, it performed very poorly, in some cases even missing by several orders of magnitude. Having a mean relative error higher than one (in the case of the model it was 2.41) means that the difference between the predicted and the real values is higher than the real one (4.1). Even if only the 50% lowest values for the σ_e are taken into consideration, the distribution in Table 4.5 demonstrates that although the majority of the events have a MRE close to ± 1, some other events have a MRE so high that take the mean out of the quartiles, the density plot of the distribution in Figure 4.8 even shows some events with MRE higher than 10, meaning $E_p - E_t > 10 \cdot E_t$. As for the σ_e output of the model, Figure 4.9 shows that it follows the same distribution of the Table 4.5, with some low values of σ having a high error component, so this output variable can not be used to savely state if the μ_e output is good or not.

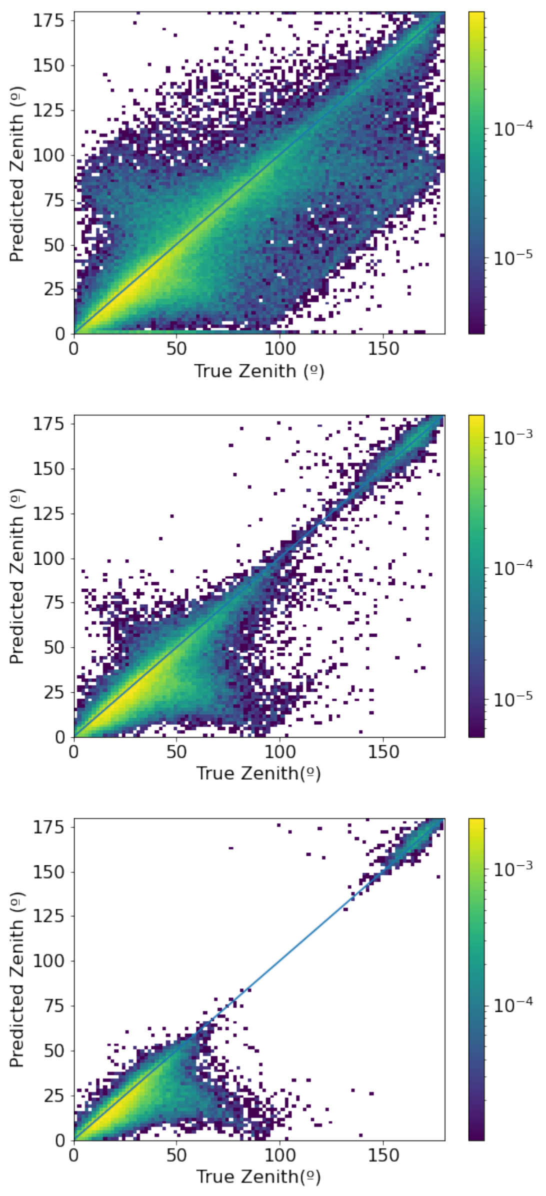
$$\text{MRE} = \frac{E_p - E_t}{E_t} > 1 \rightarrow E_p - E_t > E_t \tag{4.1}$$

Magnitude	**Mean**	**Median**	**Q1**	**Q2**
MRE	1.466	0.046	-0.58	1.33

Table 4.5: Data from the distribution of the relative error on the energy regression model.



(a) Zenith regression model



(b) BBfit

Figure 4.4: Histogram of the predicted vs. real Zenith components. Each cut takes 100/50/25 percent of the lower values of the reconstruction quality variables, which can be σ or χ^2.

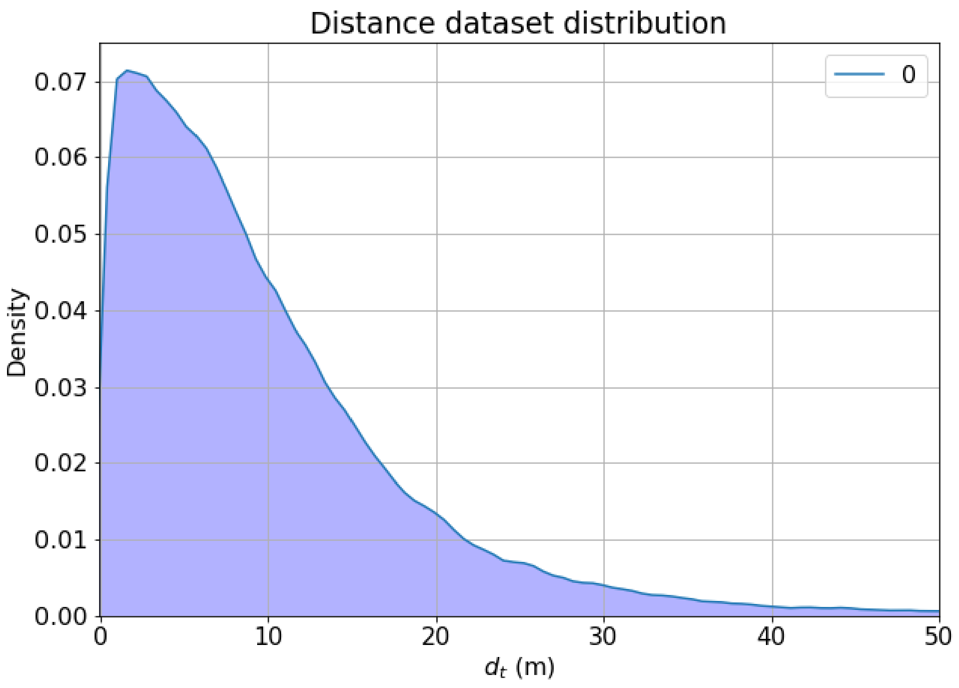


Figure 4.5: Density plot of the distance values in the test dataset.

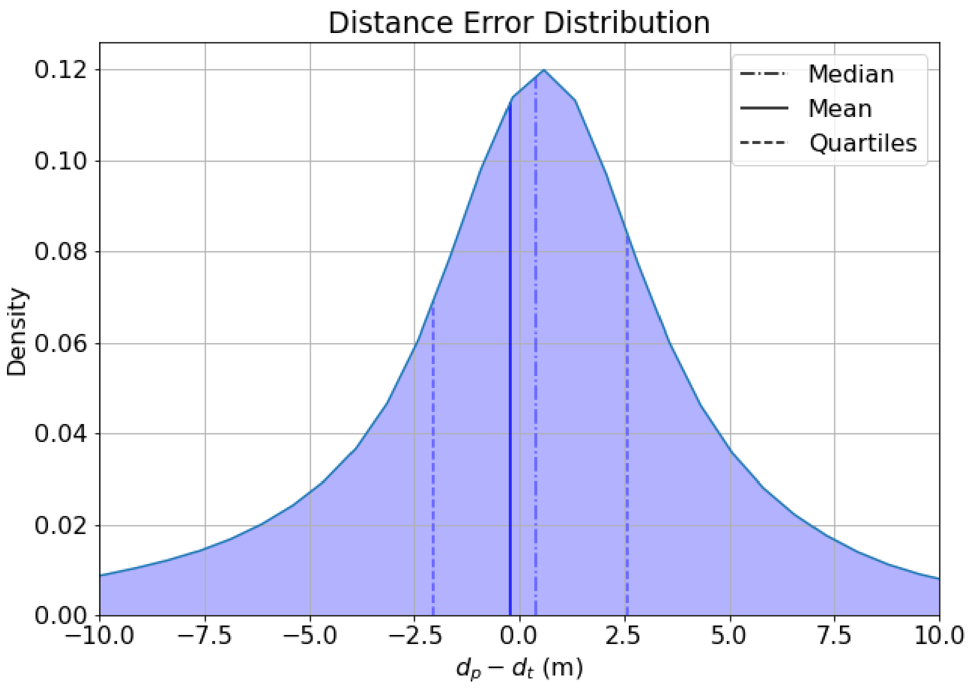


Figure 4.6: Density plot of the distance regression model error ($d_{predicted} - d_{true}$).

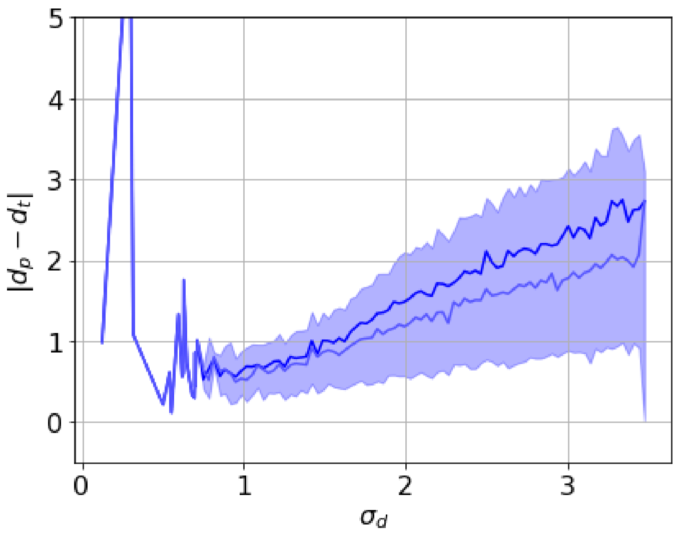


Figure 4.7: Absolute error vs. σ_d output of the distance regression model.

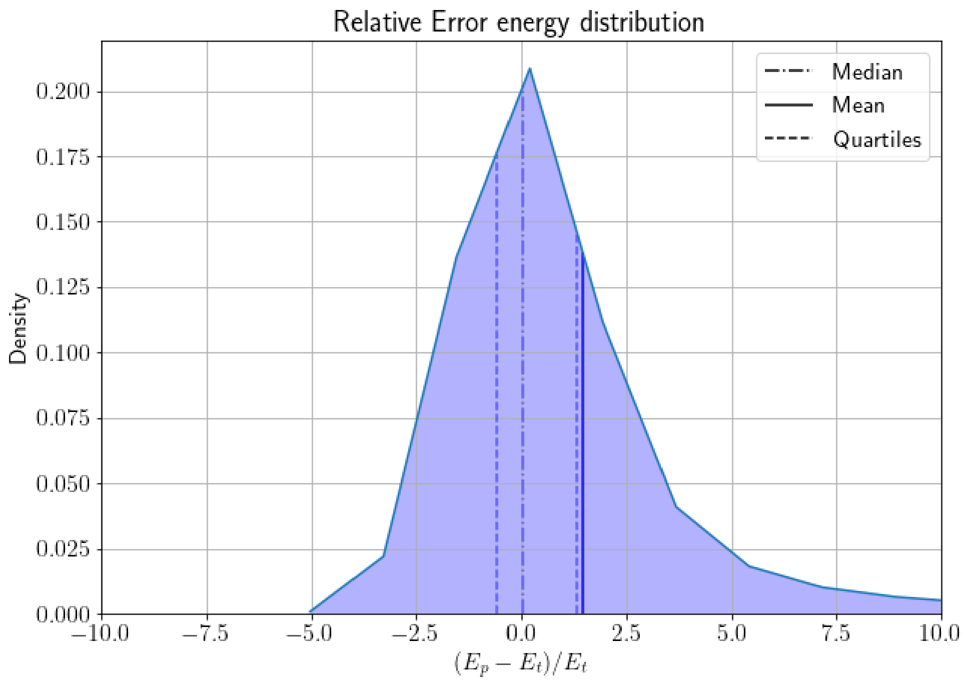


Figure 4.8: Density plot of the relative error distribution from the energy regression model.

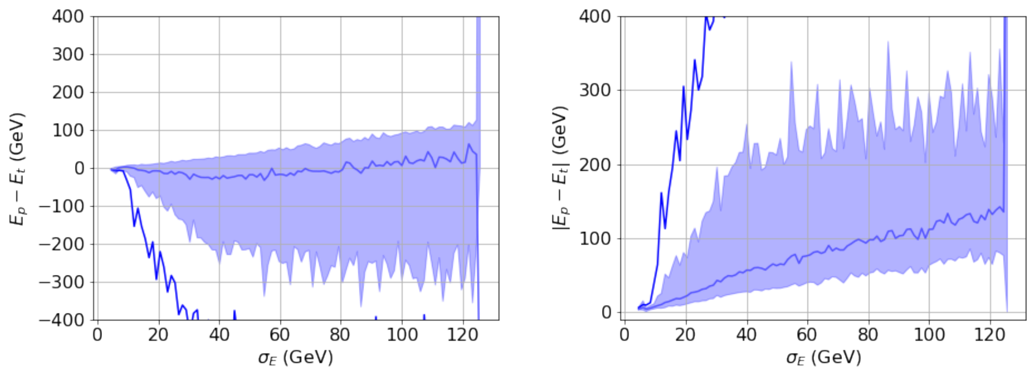


Figure 4.9: Absolute error vs. σ_e output of the energy regression model.

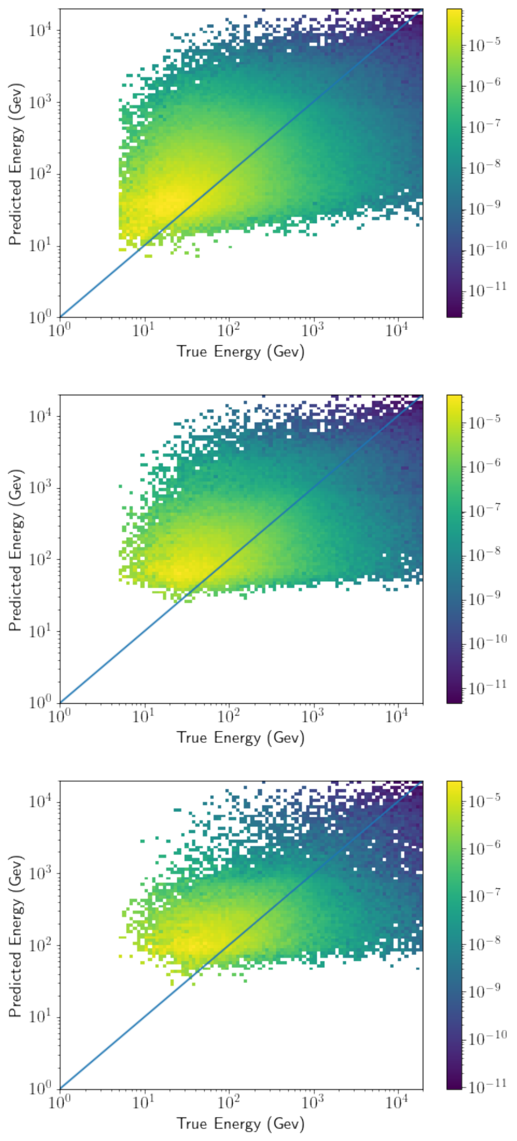


Figure 4.10: Density plot of the predicted vs. real energy values. Each cut takes 100/50/25 percent of the lower values of the reconstruction quality variable, σ_e.

Chapter 5

Conclusions and future work

5.1 Conclusions

As the results in section 4.1 show, the application of machine learning algorithms lead to better results in the estimation of the Zenith component in single-line events when compared with BBFit algorithm which was designed from a particle physics standpoint, even lowering the error metrics by more than 50%. This should serve as a demonstration that the usage of multi-purpose machine learning techniques can be as useful as the development of specifically designed algorithms. But an inherent problem with this type of algorithms should be taken into consideration, which is the fact that they work like a black box, as such their results should not be taken for granted in every case.

Although both the Zenith and distance regression models where capable of estimating their variables with a reasonable error, the energy regression model produced results which are not acceptable even when only selecting the values with the lowest σ_e, as most of the estimated values have a deviation in several orders of magnitude as Figure 4.10 shows. This is probably due to the lack of information that the model receives about the distance between the event and the line, as the model can not differentiate in the majority of the cases between low and high-energy events.

5.2 Future work

As it has been proven that the use of convolutional neural networks could estimate the Zenith component of the neutrino trajectory, a new architecture could be defined with two pairs of outputs for estimating both the Azimuth and Zenith components in single-line events, which none the AAFit and BBFit algorithms can do. In this case, it would be highly recommended to use more convolutional layers in the models as Table 3.7 implies that the model can benefit from more convoluted channels. This new model would allow for the accurate estimation of both components in the majority of events registered at the ANTARES telescope, which is currently being developed by the research group using the optimized preprocessing script described in this thesis. But the scope of the investigation should not be confined to single-line events, new models shall be trained for multi-line events and be compared with both the AAFit and BBFit algorithms to really know the potential of machine learning algorithms in comparison with their particle physics counterparts in their full potential.

With respect to the energy regression, a more in-depth study should be carried out on the impact that the lack of distance information is generating on the model's accuracy. Adding more variables containing the event position (such as the z_c in Figure 1.12) may help the model to estimate the energy of the event at the source with the information received at the photomultipliers.

Part II

Requirements

Chapter 6

Requirements

6.1 Motivation

The objective behind this part is the description of the minimum requirements and methods to run the scripts and implement the models for the correct estimation of the components of the neutrino's trajectory described in the report of this thesis.

6.2 Material Requirements

To run the preprocessing scripts and train the described models, a computer or a server with the following requirements shall be used:

Hardware

- The graphical processing unit (GPU) shall be a NVIDIA Tesla K80 or a superior model. It is important that the GPU comes with the proprietary CUDA cores from NVIDIA as the framework that is being used to train the models only works with these type of cores. The GPU RAM or VRAM shall not be less than 12Gb.
- The central processing unit (CPU) shall be an AMD Ryzen 3600X or a superior model. The CPU shall run at its base clock frequency and it shall contain at least 6 cores and 12 threads to take advantage of the multithreadding optimizations described in subsection 3.1.3. A cooling fan or any other refrigeration solution shall be correctly installed over the CPU to maintain the temperature inside the recommended range given by the manufacturer.
- The random access memory (RAM) shall not be less than 64Gb if the training process is expected to run in the minimum time possible. However the models can still be trained with a minimum of 16Gb, but the training process will be slowed down significantly.

- The minimum required space for installing the software is 250Gb, although the storage device shall be connected through an PCI Express connection for the models to access the information as fast as possible.
- To power the whole PC, the power supply selected for it shall be capable of constantly outputting 650W or more.
- The PC used for development shall be equipped with the minimum peripherals that enable an efficient interface, such as a mouse, keyboard and screen.

Software

The operating system (OS) installed on the PC shall be Manjaro Linux, although any other Linux distribution will provide the same results and the installation procedure will be similar. Once the operating system is properly installed, the following software shall be installed using the package manager of the distribution.

- `python-numpy`: Numerical and vectorial calculus python library.
- `python-pandas`: Data management library.
- `python-matplotlib`: Graph and plotting python library.
- `python-pytorch-cuda`: Framework for declaring and training the neural network models. This package comes with the required optimizations to train the models using the GPU's CUDA cores.
- `jupyter-notebook`: Server-client IDE application to run python code interactively.

The package manager shall install all the software and the dependencies required for its usage, including the NVIDIA's drivers. The specific code for the thesis shall be downloaded from Github [10]. Once all software has been correctly installed, the developer shall run the provided `software-check.py` script to check the correct installation of the whole environment. The script should not return any errors and the CUDA device shall be found by it.

6.3 Running Instructions

Processing the dataset

Once all the required software is downloaded and verified, the dataset shall be extracted and placed inside the `preprocessing` folder. Both the `read-and-process.py` and `separe-norm.py` shall be run sequentially, the first script will output one folder with all the processed images while the second script will create three folders containing fixed files to train, validate and test the models. Each script comes with header variables that will determine the properties of the final dataset and whose purpose is commented on the script. The resulting folder shall contain the files in the original ".npz" format, so the developer shall run the `to-hdf5.py` script to change the file type of the dataset to the more convenient HDF5 format. Move or make a symbolic link of the final dataset folder to both the `Models` and `Evaluation` folders.

Model definition and training

To train a model, the developer shall open one of the *notebooks* provided in the `Models` folder using the Jupyter IDE. After selecting the desired header options and training the model defined in the `Net` class, the script would have created an output folder containing several files. As an example, if the net identifier is set to be `Conv1`:

- `Conv1_architecture.txt` shall contain the information about the network's architecture.
- `Conv1_params.txt` contains the information about the options given to train the model, such as the optimizer, batch size, etc.
- `Conv1.pt` saves the state of the network when the loss function is at its lowest value, this file will be used for the evaluation of the network.
- `Conv1_loss.jpg` shows the evolution of both the training and validation loss throughout the training process.

6.4 Model Evaluation

After training a model, the user shall copy the resulting folder to the `Evaluation` folder. Open the desired evaluation notebook and specify the route of the model that shall be evaluated. Finally, the developer shall copy the network's architecture from the `Models` folder notebook into the evaluation notebook and run all the cells. Once finished, a figure folder shall be present in the `Evaluation` folder.

Part III

Budget

Chapter 7

Budget

The objective of this chapter is to give an overview of the required budget to carry out the development of this thesis.

7.1 Partial Budget

Hardware Cost

For the development of these thesis, a consumer-grade PC was used in conjunction with a server solution. For the consumer-grade PC, an amortization period of 6 years will be considered. The total hardware cost are shown in Table 7.1.

Component	Uds.	Quantity	Unit Price (€)	Amortization Period (months)	Amortized Interval (months)	Total (€)
AMD Ryzen 2600X	u	1	120,00	72	6	10,00
MSI Mortar Platinum	u	1	80,00	72	6	6,67
8 Gb RAM Stick	u	2	42,00	72	6	7,00
500 Gb SATA SSD	u	2	45,00	72	6	7,50
NOX Urano Power Supply	u	1	47,00	72	6	3,91
NVIDIA Geforce 1060	u	1	280,00	72	6	23,33
Server Solution	month	6	20,00	——	——	120,00
					Total	178,41

Table 7.1: Hardware Cost.

Software Cost

As all the required software to carry out this thesis is open-source, no software source will be considered. A detailed description of the required software can be found in section 6.2.

Labour Cost

Labour costs are considered to be the human resources necessary to carry out this project. In the case of this thesis, the labour force is composed of Alicia Herrero Debón and Joan Salvador Ardid Ramírez as tutors and Antonio Alejandro Aslan Suarez as student of the Degree in Automatic and Industrial Electronic Engineering and author of the thesis. These are shown in Table 7.2.

Name	**Ud.**	**Quantity**	**Unit Price (€)**	**Total (€)**
Ms. Herrero	h	15	25	375,00
Mr. Ardid	h	25	25	625,00
Mr. Aslan	h	450	12	5400,00
			Total	6400,00

Table 7.2: Labour Cost.

7.2 Total Budget

For the calculation of the total cost of the project, the partial budgets set out above will be taken into account. In addition, 13 % will be added for overheads costs and 6 % for industrial profit. Finally, 21% VAT will be added to the gross price. The total budget is shown in Table 7.3.

Partial Budget	***Total (€)***
1. Labour Cost	6400,00
2. Hardware Cost	178,41
Execution Cost	**6578,40**
13% Overhead Cost	855,19
6% Industrial Benefit	394,70
Cost before taxes	**7828,30**
21% VAT	1643,90
Total Cost	**9472,20**

Table 7.3: Total Cost.

Bibliography

[1] USGS. Zenith and azimuth angles associated with solar angle bands. `https://www.usgs.gov/media/images/zenith-and-azimuth-angles-associated-solar-angle-bands`.

[2] ESSnuSB. Water cherenkov detector. `https://essnusb.eu/glossary/water-cherenkov-detector/`.

[3] ANTARES. Virtual reality views and representations. `https://antares.in2p3.fr/Gallery/3D/index.html`, 2011.

[4] M. Ageron et al. Antares: the first undersea neutrino telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 656(1):11–38, 2011.

[5] Baptiste Wicht. *Deep Learning feature Extraction for Image Processing*. PhD thesis, 01 2018.

[6] MathWorks. Edge detection methods for finding object boundaries in images. `https://es.mathworks.com/discovery/edge-detection.html`.

[7] Sakshi Indolia and Anil Kumar Goswami. Conceptual understanding of convolutional neural network- a deep learning approach. *Procedia Computer Science*, 132(nil):679–688, 2018.

[8] José Luís Rocabado Rocha. Reconstrucción de trayectorias en telescopios submarinos de neutrinos mediante técnicas de machine learning. 2020.

[9] Boris M Bolotovskii. Vavilov - cherenkov radiation: Its discovery and application. *Physics-Uspekhi*, 52(11):1099–1110, 2009.

[10] Antonio Aslan. single-line-nn source code. `https://github.com/AntonioJojoto/single-line-NN-TFG`.

[11] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10):992, 2019.

[12] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[13] Hecht-Nielsen. Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks*, page nil, - 1989.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[15] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. 2012.

[16] Christopher M. Bishop. Mixture density networks. Workingpaper, Aston University, 1994.

[17] A. Albert and M. et al. André. Monte carlo simulations for the antares underwater neutrino telescope. *Journal of Cosmology and Astroparticle Physics*, 2021(01):064–064, Jan 2021.

[18] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, 1997.

[19] Mike Folk and Gerd Heber et al. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, page nil, - 2011.

[20] J.A. Aguilar et al. A fast algorithm for muon track reconstruction and its application to the antares neutrino telescope. *Astroparticle Physics*, 34(9):652–662, 2011.