Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master's Thesis
Academic Year 2025-2026

Orientation : Data science (DS)

Machine Learning for Cherenkov Telescope SST-M

Autor

Hugo Varenne

Supervisor

Upegui Posada Andres
MSE Teacher

Representative

Matthieu Heller
DPNC - Department of Nuclear and Particle Physics

Version : 1.0.0

Geneva, Switzerland // TM, 2025

Acknowledgements

Version history

The report got majors updates during the project to adapt its content to standards and best practices. Table below gives an overview of each version major changes

Version	Details
1.0.0	Initial project structure

Contents

1 Executive Summary **1**

2 Context **2**
2.1 Structure of the report . . . 3

3 Materials and methods **6**
3.1 Gamma ray . . . 6
3.1.1 Air shower . . . 8
3.1.2 Cherenkov light . . . 9
3.2 Triggering system . . . 10
3.3 Reconstructed data . . . 11
3.3.1 Type of primary particle . . . 11
3.3.2 Energy of primary particle . . . 11
3.3.3 Arrival direction of the primary particle . . . 11
3.4 CTLearn library . . . 12
3.5 IACT . . . 12
3.6 Architecture . . . 14
3.6.1 Baobab . . . 15
3.6.1.1 Production environment . . . 16
3.6.1.2 SLURM . . . 17
3.6.1.3 Dedicated space . . . 19
3.6.2 Calculus . . . 19

4 Data **21**
4.1 Prediction tasks . . . 21
4.1.1 Particle Classification . . . 21
4.1.2 Energy Regression . . . 22
4.1.3 Direction Regression . . . 22
4.1.4 Particle type . . . 23
4.2 Generation of data . . . 23
4.3 Format of files . . . 25
4.3.1 Content of HDF5 file . . . 26
4.4 Compliance with the server . . . 32
4.4.1 Transfer to the cluster . . . 33
4.4.2 Conversion of data . . . 33
4.4.3 Move Data across the cluster . . . 33
4.4.4 Merging files . . . 34
4.4.5 Keep only wanted data . . . 34

5 MLOps approach **35**

5.1 Task Breakdown . . . 36
5.2 Configuration files . . . 37
5.3 Report Generation . . . 38
5.3.1 Generic Section . . . 39
5.3.1.1 General information . . . 39
5.3.1.2 Model and Runtime Metrics . . . 40
5.3.1.3 Graphics . . . 41
5.3.2 Particle Classification . . . 43
5.3.2.1 Evaluation Metrics . . . 43
5.3.2.2 Graphics . . . 45
5.3.3 Energy Regression . . . 49
5.3.3.1 Evaluation Metrics . . . 49
5.3.3.2 Graphics . . . 51
5.3.4 Direction regression . . . 55
5.3.4.1 Evaluation Metrics . . . 55
5.3.4.2 Graphics . . . 57
5.4 Comparison of models . . . 60
5.4.1 Metrics modifications . . . 61
5.4.2 Graphics modifications . . . 62

6 Models **66**
6.1 Reference . . . 66
6.2 CTLearn Models . . . 67
6.2.1 ResNet . . . 67
6.2.2 CNN . . . 70
6.2.3 Custom model . . . 72
6.3 Tasks . . . 75
6.3.1 Prepare model . . . 75
6.3.2 Training . . . 76
6.3.3 Testing . . . 80
6.4 Model Optimization . . . 81
6.4.1 Pruning . . . 81
6.4.2 Quantization . . . 85
6.5 New Models . . . 86

7 Conclusion **87**
7.1 Conclusion . . . 87
7.2 Challenges . . . 87
7.2.1 Link GPU to Tensorflow . . . 87
7.2.2 Accelerate training speed . . . 88
7.2.3 CTLearn library . . . 88
7.2.4 Quantization of complex models . . . 88
7.3 Limitations . . . 89
7.4 Future improvements . . . 89
7.5 Personal reflection . . . 90

8 Declaration of honor **91**

9 References **92**

10 Appendix **103**

1 Executive Summary

2 Context

Gamma rays (or gamma radiations) are a form of electromagnetic radiation coming from the sky. They are coming from different kind of astronomical events or objects. This includes black holes, solar flares, supernova explosion or nebulae, normally not visible in optical. The particularity of this type of ray comes from the fact that it has the smallest wavelengths and the most energy of any type of ray in the electromagnetic spectrum. It's not directly at the origin of these events, the origin being a cosmic ray. It's when this cosmic ray interacts with matter, magnetic fields or light that the gamma ray is produced.

The issue with gamma rays is that it's complicated to detect from the Earth because of the atmosphere blocking them to prevent destructing life on Earth. This means there is two possibilities to detect them : Space telescopes (e.g. Fermi-LAT) or Ground-based Cherenkov telescopes (e.g. CTA, HESS). Space telescopes are currently too small to stop VHE (Very High Energy) gamma rays and fails to collect enough to have this possibility efficient.

CTLearn is the library providing Machine Learning and Deep Learning solutions to work with imaging atmospheric Cherenkov telescopes and is the central core of this thesis. The work will be conducted with Ground-based Cherenkov telescopes and the data they collected about events occurring in the sky.

How can they detect events even with the atmosphere blocking them ? This is where Cherenkov comes in. The idea is to use the atmosphere as part of the detector. The atmosphere is too thin like the telescope to avoid effects of the absorption of a VHE gamma ray to be detected from the ground. When a gamma ray interacts with the atmosphere, it creates the Cherenkov light, cascades of high energy subatomic particles with a blue color. This cascade is faster than the speed of light and powerful telescopes need to be used to analyze it. There are also known as air showers and comes with two distinct variations : the electromagnetic ones (gammas) and the hadronic ones (protons, electrons, etc.).

The difference comes from the way the shower develops. The shower with a gamma as origin will have a smooth lateral distribution when having hadrons as origin results in a clumpy lateral distribution.

A camera will capture what the telescope detects, storing and treating information to be treated as data for future usage (e.g. with CTLearn).

Three important information needs to be extracted from the data to understand the event correctly : the type of particle, the energy and the direction of the event. At the start, some machine learning models were used to predict theses values (e.g. Random Forest). The limited aspect of ML and the need of high precision when predicting values have encouraged specialists to explore

others solutions. This is where Deep Learning intervenes. Its ability to analyze 2D images (e.g. with CNN) and the complexity it provides made it a good candidate to enhance IA usage to work with the analysis of events. Currently, the actual best model is a ResNet model working for the three tasks described earlier. However, there is still place to better models and results.

The thesis intervenes with the idea of improving the current solution in place. Not only a better model could be found, but the current environment to generate and evaluate theses models could be improved. The library and the system in place was introduced by astrophysicists and not computer scientists. Computer science point of view could improve the global usage of the CTLearn library and model utilization by providing standardization, tools and automatization in the model handling process. It could also bring new interesting ideas to work on to further improve the analysis of air showers.

At the end of the thesis, it's expected to provide tools (in the form of scripts or others) and a potential new model to predict required information.

2.1 Structure of the report

In order to have a better overview of the content of the report, the structure and content of main chapters will be described here.

Context

Context aims to give an overview of why this thesis exists. It introduces many concept and mains elements treated during the thesis. It gives information about current situation and stakes of the thesis. This context is really close to an executive summary in the form.

Materials and Methods

Material and Methods is a section regrouping elements that needs to be understand in order to get deeper in the thesis. It's a prolongation of the context, developing mentioned concepts and tools that are sufficiently important for the thesis to have dedicated space. It's here to explain the physics part of the thesis.

It includes physics concept like the air showers (3.1.1) or the Cherenkov light (3.1.2). Theses elements are essential to understand the stakes of projects like this thesis. Additionally, others concepts such as the telescope and the trigger system are worth mentioning details about them thus not being directly part of the thesis.

Another element included here is the space dedicated the environment of the thesis (3.6). Multiples workspaces (local, cluster, etc...) and tools have been used to work in this thesis. This means some schemes and information needs to be provided to understand roles and interactions between theses environments.

Data

Data (4) from IACT have a lot of complexity and makes it difficult to work with. Therefore, a major part of the thesis needed to be consecrated to analyze the data and related elements. It includes description of metrics to extract and how the data is artificially generated to provide

enough information to train models. It also includes processing steps for the data to be used by CTLearn models in the most optimal way. Data format and content, like waveforms and frame, is an essential part to understand and needs to be detailed.

Reconstructing gamma rays (4.1) is the purpose of the CTLearn project. Models should be able to extract three different data : particle type, energy and direction. Their purposes needs to be thoroughly explained and understand.

Data comes in multiple format (4.3), zipped or unzipped. Some specifications needs to be provided to understand format purposes and their content. The content of the files needs to be carefully explained to understand their purpose. Related to that, data are simulated, meaning there is a need to understand how the generation works (4.2).

Data weren't directly in an adapted format to work with. Some processing steps (4.4) have been executed through the thesis in order to improve the efficiency and compliance of their usage with the CTLearn library. Each step is described with the objective in mind when introducing it.

MLOps approach

MLOps (5) approach and integration is one of the main axis of the thesis. It regroups details about this methodology and what contribution related to this have been integrated as part of the thesis. It includes tools like the report generation or even concepts like the tasks breakdown. This concept is fully detailed as an introduction. MLOps is an important aspect to take in account in order to optimize and to make easier the process of model generation.

The concept of task breakdown (5.1) is the first aspect of the methodology integrated. Adapting the structure (going from Jupyter Notebook to scripts) to a production environment is a related subject to this.

The concept of configuration files (5.2) is another aspect of MLOps. It has many benefits and enables a better tracking of experience. Details about how it operates can be found. This concept is provided with examples of configuration files for each task in the appendix.

The most substantial part of the MLOps integration in the thesis is the tool dedicated to generate a report (5.3). This tool is a first step to get an overview of models performances with specific metrics and graphics depending on the task. It also relies generic information useful like the inference time per event. The report helps to understand where the model is better and can be used to compare to other models.

Another interesting tool is the comparison of models (5.4). This tool generates a report with metrics and graphics, on the basis of the report generation tool, that goes further in the comparison of models by comparing multiple models together with results next to each other, easier to analyze.

Models

Models (6) is one of the main axis explored during the thesis. It regroups everything related to models creation and usage as well as related subjects such as optimization. This is an essential part of the thesis as models performance and improvements is the core of it.

To get an idea of performances of models created during the thesis, some reference metrics and graphics from the literature related to CTLearn has be summarized and centralized as a goal for

future models generated to beat (6.1).

Some models pre-built are provided by the CTLearn library and were mostly used during tests and implementation of tools (6.2). Theses models and their specificity have been listed and detailed as well as configurable options that can be set to modify them. Additionally, a custom model option is available for models with different architectures to be used. This part includes a description of a template dedicated to fit theses customs models with CTLearn library requirements (6.2.3).

Tasks (6.3) related to the model generation needs to be detailed as the features and configurable parameters may be technical. Three tasks composes the model generation : preparation, training and testing of the model. It explains which interactions with the CTLearn are required to make it work.

Model optimization (6.4), in the case of the thesis, was only explored in surface. Techniques like pruning or quantization are essential depending on resources restriction the environment of the model may have. Some details and examples of implementation can be found for more details on that.

Conclusion

The conclusion of the report (7) provides an overview of all the work accomplished in the project. This into two types of feedback: a more formal feedback on the results obtained, what was results, what was achieved and how it fitted in with the schedule, and a more personal feedback on how the of working on the project. This is also the place where any problems encountered are mentioned and explained, as well as any future improvements that might be envisaged.

Bibliography, Appendix

Sections are also provided to reconcile all the sources (8), figures and appendices that were used in this project. Theses different chapters are intended for this purpose. Mentioning all these elements supports the work that has been carried out, and can provide insights on potential future projects. Appendix (10) contains images, scripts and documents that explores further aspects of the thesis or elements that couldn't be included in it.

3 Materials and methods

Materials and methods section is dedicated to details element introduced in the context of the thesis It also reflects astrophysics elements that will be mentioned through the report. It regroups the physics concept used and explain them for readers from another domain.

It includes unorganized and unrelated notions such as tools, concept or even objects related in any way to the subject of the thesis.

3.1 Gamma ray

Cosmic-rays [1] can be seen as atomic particles accelerated to extraordinarily high energies, at almost the speed of light. They constantly bombard the Earth from space, coming from space object like galaxies, nebulae, pulsars or event black holes. During their travel time, they are accelerated and deflected by multiple objects, making it difficult to understand their origin and track them. The composition of cosmic ray is mostly protons and helium nuclei (99%) with a small percentage of heavier nuclei (1%) [2].

Gamma rays can be seen as the child of the cosmic-ray, generated when cosmic-ray interaction with matter, radiation fields or magnetic fields. This is the most energetic form of electromagnetic radiation. The interest in them relies in the fact that they preserves directional information from the the interaction point and can be detected by telescopes. The travel direction of this type of ray is in a straight line because it's insensitive to astrophysical objects and magnetic fields, It helps to identify objects (figure 3.1) that accelerated cosmic rays and generated gamma rays. There are multiple processes in existence that covers the interaction of the cosmic ray with objects to generate a gamma ray. Theses processes [3] are called the Leptonic processes (for electrons) and the hadronic process (for protons and nuclei).

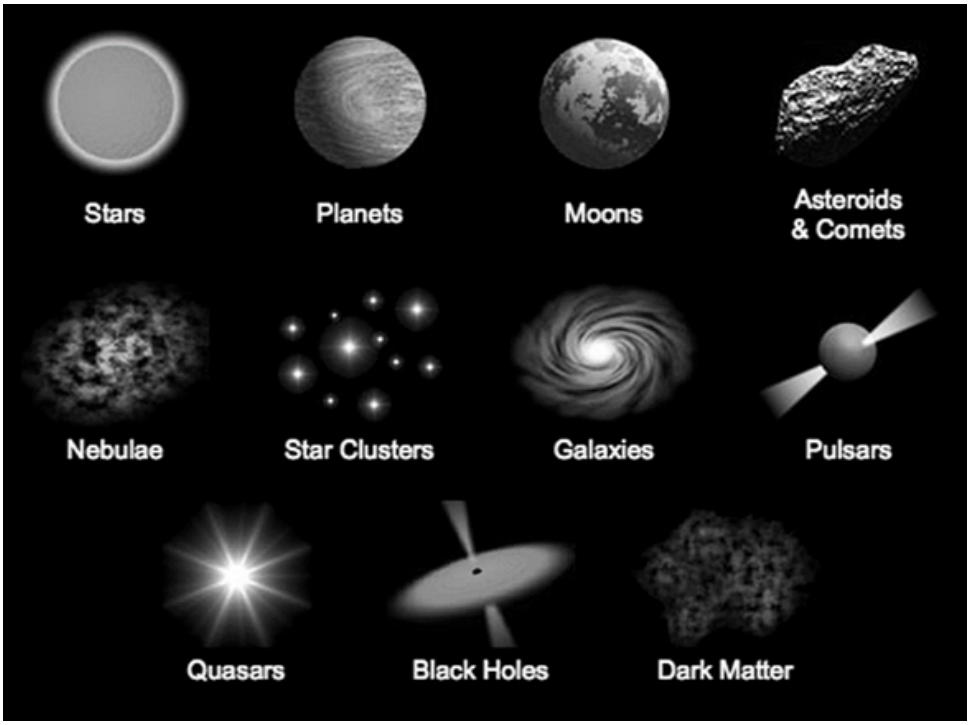


Figure 3.1: Non-exhaustive of astrophysical objects than can collides with cosmic-ray and generate gamma-ray. [4]

Information carried by a gamma ray are crucial to understand the spatial objects. Among the information, some of them are more interesting like the arrival direction or the energy. Theses types of rays enables to study the high-energy astrophysics and more precisely magnetic fields, particle interaction or dark matter. To get an idea of what a gamma ray looks like when captured, refer to the figure 3.2.

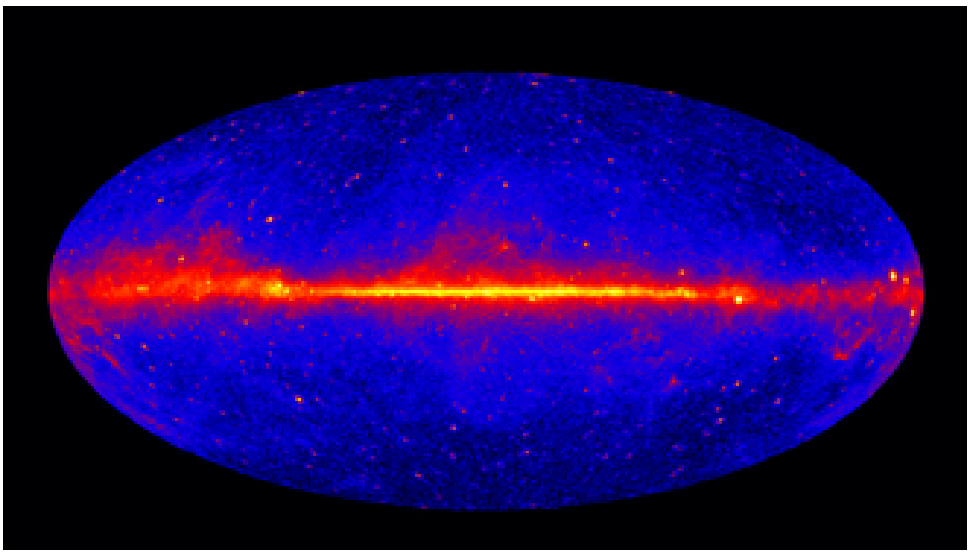


Figure 3.2: Gamma ray observation of the sky captured by the Fermi telescope for 5 years of observation. [5]

They cannot be directly detected from the ground of Earth because of its interaction with the

atmosphere. When the gamma-ray hits the atmosphere, it creates air showers as a reaction.

3.1.1 Air shower

Air showers [6] [1] results from the interaction of high-energy particles with the atmosphere like gamma rays. This interaction creates a cascade of subatomic particles called extensive air showers. Air showers are the way to observe high-energy particles from Earth as the primary particle cannot reach it without being absorbed. This cascade occurs a chain reaction with subparticles. Theses particles travels at a high speed, faster than the speed of light in the air, creating what is called the Cherenkov light. It can be compared to the blow-up produces after passing the speed of sound.

An air shower longitudinal development is almost of five kilometers and is produced around 10 km above the see level. They also can be extremely large depending on their stability. The figure 3.3 displays an example of shower development from the gamma ray until it reaches Earth.

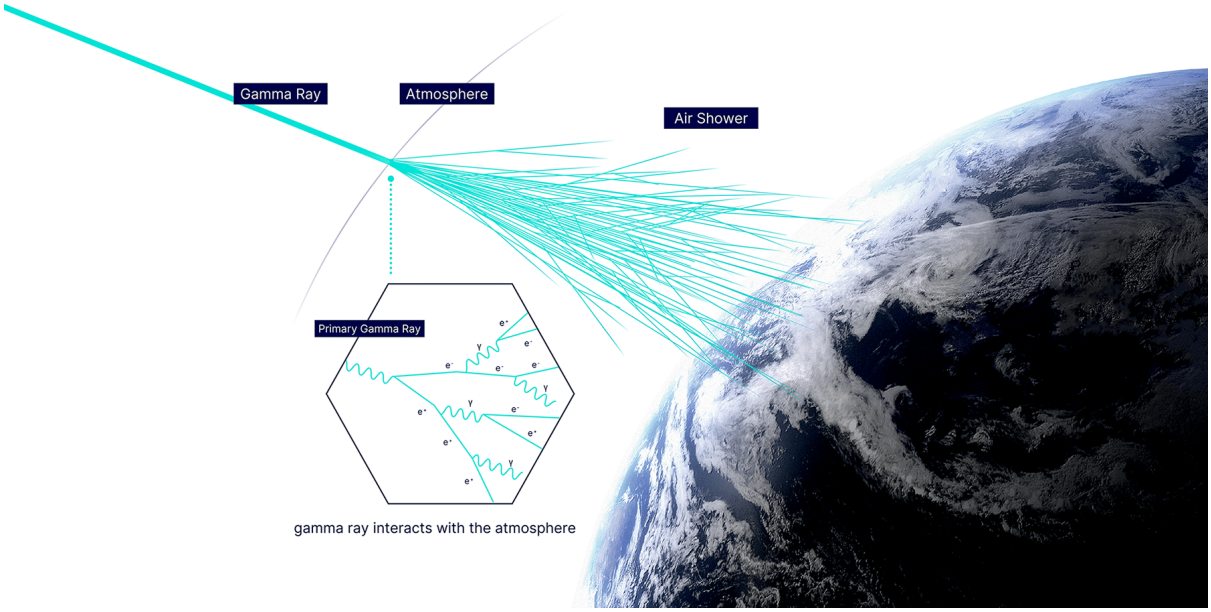


Figure 3.3: Gamma ray shower development from collision with the atmosphere to Earth's ground. [1]

Two types of air showers are generally produced : electromagnetic showers and hadronic showers. The type is determined by the particle that interacted with atmosphere. The figure 3.4 displays a visual distinction between both types of showers.

Electromagnetic showers [7] occurs when gamma rays collides with the atmosphere. It requires specific particles from the gamma ray to be considered. The particles are electrons, positrons and photons. It's a simple and more predictable shower than the hadronic. The global air showers is compact and have a smooth development. It develops by doing pair production. It means that gamma ray is converted in electron-positron pair. Theses pairs emits themselves gamma rays that will also collides again. This second process is called Bremsstrahlung.

Hadronic shower [8] occurs when a cosmic ray collides with the atmosphere. It's the dominant type of showers occuring. It's instable and irregular, diffucult to work with. The complexity comes from

the stronger nuclear interactions. When colliding with the atmosphere, it creates various secondary hadrons such as pions, kaon or others type of hadrons. Unstable secondary hadrons like this decays into muons, neutrinos and even gamma ray. This type of shower mixes various forms of particles, proving the instability of this type of shower.

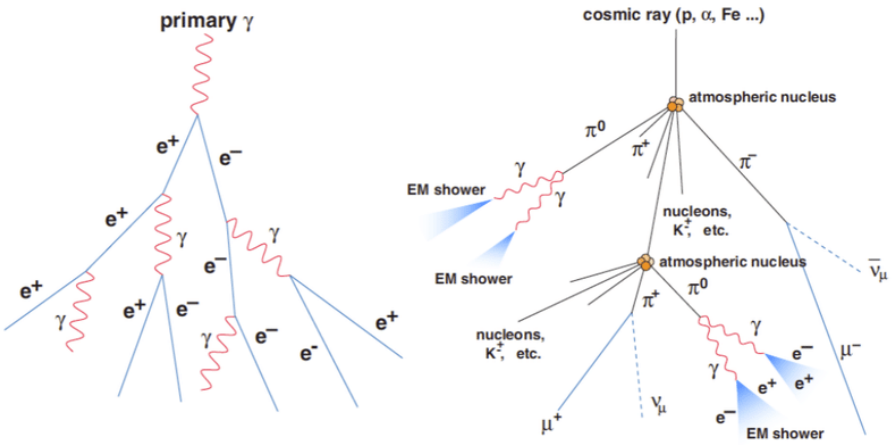


Figure 3.4: Representation of an electromagnetic shower (left) and an hadronic shower (right). [9]

The type of showers that is interesting is the electromagnetic ones. They are initiated by gamma rays and are lesser complex to analyze than hadronic ones. Electromagnetic showers follows a logical pattern, easier to understand. Unfortunately, this type is much less present compared to hadronic.

3.1.2 Cherenkov light

Cherenkov light [1] usually occurs in the development of an air shower in the atmosphere when secondary particles are faster than the speed of light in a medium. The effect is similar to the sonic boom when speed of sound is exceeded. Indeed, it's possible to get faster than the speed of light when the light goes through a medium like water or air. It emits electromagnetic radiation and it's characterized by a faint blue glow. The blue light is really faint, lasting only few nanoseconds. The color blue appears because its emitted at shorter wavelengths in the visible spectrum (around 400nm). The figure 3.5 displays a look at the Cherenkov light. Another place where this phenomenon can be found is in nuclear reactor where particles are particularly agitated.

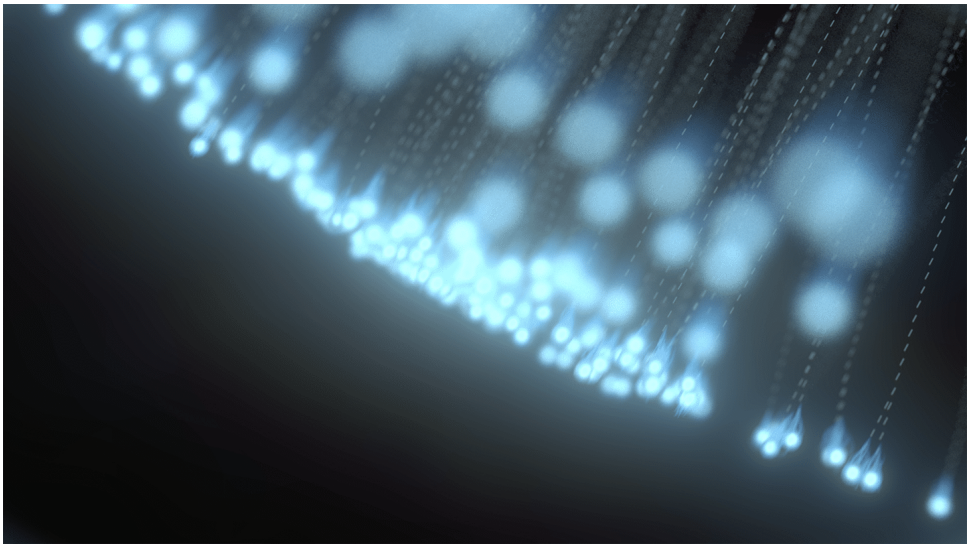


Figure 3.5: Visual representation of Cherenkov light emitted by secondary particles from an air shower. [1]

It's what's detected by IACT telescopes. Although the light lasts a very short amount of time, it has the time to spread on the ground over a large area, allowing telescopes to detect it. Based on the time, shape and brightness of the phenomenon, it can do some reverse engineering to reconstruct the particle at the origin of this event.

3.2 Triggering system

The triggering system [10] is a solution implemented to manage the overwhelming amount of data that is captured by telescope and their camera, making it impossible to store everything. Camera samples signal at high speed (hundreds of MHz) with lot of pixels, but only a tiny amount of the data captured contains gamma-ray events, the rest being pollution of Night Sky Background and unexpected events. Therefore, some filtering needs to be done in the form of a real-time trigger system that will save events only if they are relevant.

The trigger system [11] needs to be extremely fast and reliable to filter effectively data. The trigger system relies on multiple levels, increasingly selective the deeper it gets. The event is saved only after passing through the complete trigger system.

The triggering system used for the telescopes in the scope of this thesis (SST-1M) is called DigiCam. It's a fully digital camera readout and trigger system for SST-1M. The particularity of this system is that it digitizes the signal to perform triggering. It uses FPGA to work and is highly configurable with thresholds definition and the scalability of the tool.

Digicam triggering system is composed of three main levels [10]. The first is a pixel-level trigger where a threshold is applied to suppress low-level noise and fluctuations. It also determines potential pixel containing Cherenkov light. The second level is more at a camera-level where clusters of pixels simultaneously activated are identified. The signal is summed up to determine if it exceeds a global threshold. It helps to suppress NSB and selects events with patterns consistent to air showers. The last level is an array/stereo trigger where multiple telescopes are combined to analyze event that passes to this point. It observes if multiple telescopes triggered the same event in the same time

window. It prevents local noise or single-telescope fluctuations to be detected as an event.

3.3 Reconstructed data

Telescopes and cameras do not observe particles directly, but only consequences of its interaction of the primary particle with the atmosphere. As stated previously, it's the Cherenkov light produced by the shower that is observed by instruments.

It means the recorded data cannot be interpreted directly. It's an issue as an outrageous amount of cosmic-ray and gamma-ray goes through the atmosphere as well as noise and background light, complicating the extraction of valuable information as the gamma-ray. Refer to the gamma-ray chapter to see its importance (3.1).

Based on data collected from telescopes and camera, it's possible to reconstruct parameters of the particle at the origin point. The parameters that needs to be identified to understand the particle at the origin are the type, the energy and the arrival direction of the particle. Theses parameters can be used for scientific analysis of the gamma-ray origin.

3.3.1 Type of primary particle

The reconstructed parameter is the type of primary particle. The type is known from the morphology of the air showers. Like the two showers types previously seen, the parameter is set to determined to which category the air shower is. The two categories are electromagnetic and hadronic. The first one can be identified as the signal, smooth, elongated and compact (gamma rays). This is also from this type of air showers that others parameters like energy and direction are calculated. The other one is identified as the background. It's a more complex and irregular air shower due to decays and nuclear interactions (cosmic rays).

The repartition of the two types is highly disproportionate with the hadronic being massively more present.

3.3.2 Energy of primary particle

The second reconstructed parameter is the energy. Its value is estimated from the total amount of Cherenkov light produced by the air shower or the shape and impact distance of the shower from the telescope. This parameter is generally reconstructed from gamma-ray (electromagnetic) events in order to have accurate results. It's generally expressed in TeV.

3.3.3 Arrival direction of the primary particle

The last parameter is the arrival direction of the primary particle. The idea is to get from the air shower, the original trajectory of the primary particle. Because gamma rays travels in a straight line, the direction will help to get to the origin of the gamma ray, most likely an astronomical object. The direction is reconstructed from multiple elements such as the Cherenkov light pattern, the image orientation and the timing the event is detected. It can be even more precise when combining multiple telescopes observations of a same event, with multiple angles of visualization (stereo).

3.4 CTLearn library

CTLearn library [12] is a python package providing Deep Learning solutions for IACT. It handles multiple data level from CTAO like waveforms or images. Theses models helps to do reconstruction of events using information from DL1 data-level in HDF5 format. It produces neural networks for the three main reconstruction parameters : particle type, energy and arrival time.

This package is still under developement and maintained through regular releases with new features. It uses TensorFlow in order to generate models and works along generic package of the domain like ctapipe or DL1DataHandler.

3.5 IACT

To detect gamma-ray events, CTAO or the Cherenkov Telescope Array Observatory is using multiple types of instruments. It includes telescopes, satellite and others type of camera, located on different sites in the world. Among theses instruments, IACTs (Imaging Atmospheric Cherenkov Telescopes) are ground-based telescopes used to detect very-high-energy gamma-ray events through the Cherenkov light emitted when theses events collides with the atmosphere (air showers). IACT combines a telescope with a segmented mirror and a camera oriented to capture what is observed by the mirror (each mirror equals one pixel). It records the behavior of the Cherenkov light distribution by looking ant the shape, orientation and intensity of it.

The interest of the ground observation [13] [14] with theses telescope is because of its high angular resolution and sensitivity. It helps to have precised details and identify simply source of the gamma-ray event. This approach comes with constraints like the fact it only works at nigh for a small amount of time because of light pollution coming from stars or from Earth. This also includes the fact that the detection isn't directly on the particle but on the Cherenkov light resulting from the collision with the atmosphere. The fact that telescope are used makes the field of view very small, limiting the sky coverage.

The following figure 3.6 gives an idea of the representation of the IACTs.

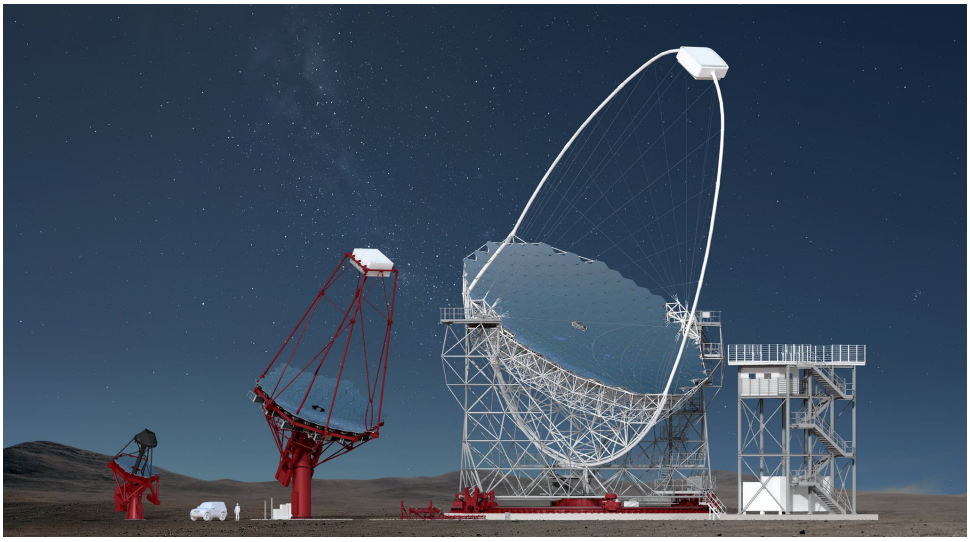


Figure 3.6: IACT telescopes. Three categories of telescopes are represented here. The difference relies in their size and number of mirrors. From the left to the right : SST, MST, LST. [1]

The telescopes available through IACT, three main categories of telescopes can be listed [1].

- **LST** : Four Large-Sized Telescopes (45 meters) in the northern hemisphere with low-range of energy (20 to 150 GeV)
- **MST** : 23 Medium-Sized Telescopes (27 meters) across the world for an energy range of 150 GeV to 5 TeV
- **SST** : 37 Small-Sized Telescopes (9 meters) in the southern hemisphere for energy above 5 TeV

SST-1M telescopes

In the thesis, data from SST-1M telescopes are used, worth mentioning additional details about it. SST-1M telescopes [15] falls in the category of small sized telescope with a single mirror. As stated earlier, this type of telescopes covers an energy range of a few TeV to 300 TeV.

The telescopes uses a Davies-Cotton optical design to provide good imaging performances for a large FoV (Field-of-View) and limited time dispersion. The specifications of the telescope can be found in the figure 3.7. A key element of the SST-1M is its camera technology, using Silicon Photomultipliers (SiPMs). It makes it possible to work under bright moonlight conditions, extending the window of time to operate. The camera is composed of 1296 pixels and is equipped of a triggering system (DigiCam). Some others elements as the wavelength filters or light concentrators helps to reduce the Night Sky Background and maximizing the Cherenkov light.

Optical properties	Focal Length	5600 ± 5 mm
	f/D	1.4
	Dish diameter	4 m
	Mirror Area (*)	9.42 m^2
	Mirror Effective Area(*)	6.47 m^2
	Hexagonal Mirror facets	780 ± 3 mm
	Preliminary on-axis PSF real optical parameters	0.07°
	PSF (80% of FoV@ 4° off-axis)(**)	0.21°
Camera Characteristics	Camera (depth x width)	60 cm x 90 cm
	Total pixel number	1296
	Pixel linear size	23.2 mm
	Pixel angular size	0.24°
	FoV	9.1°
	Photosensors PDE	> 30%
	Sampling frequency	250 MHz
	Readout rate	0.6-1 kHz
	Time Spread RMS	< 0.25 ns

Figure 3.7: SST-1M telescope. This telescope is part of project that includes the University of Geneva. Specifications of the telescope of the telescope are also provided to get a better idea of how the telescope and the camera captures information.[16] [15]

Having one telescope only enables to work in mono mode. In order to go into stereoscopic mode, a important element to observe air showers from different angles, two telescopes are required. Two of theses telescopes are currently working at the OndÅŹejov Observatory, in Czech Republic.

3.6 Architecture

During the thesis, multiple environments and servers will be used. It's important to keep track of the environment used to understand different steps through the thesis. Essentiel components to understand will be described in subsections.

The architectures varies through the project. The first environment used concerns the discovery of data and created / testing models for different tasks using a small subset of the data. It operates in local and is very messy. Its sole purpose is to do minimalist tests on aspects treated in the thesis or for exploration of new technics and tools. The figure 3.8 displays the environment used.

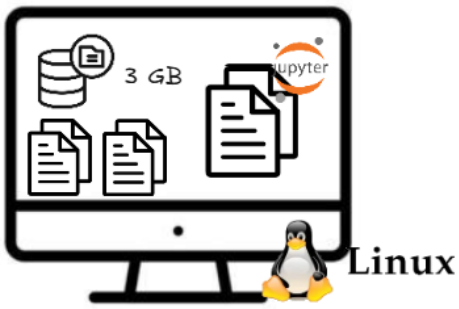


Figure 3.8: Usage of a Linux virtual machine (WSL) to test elements. The tests and exploration are done using Jupyter Notebook to execute steps independently. The amount of data used is very small because of space issues and a low complexity requirement. Configurations files, models, libraries are completely unstructured and mix together.

The second architecture extends the first one by using bigger infrastructure and an increased amount of data. It's available on a cluster and is built to be directly used for production. The structure is cleaned and there is no unnecessary files or elements. Some interactions exists between files and are explained with the figure 3.9.

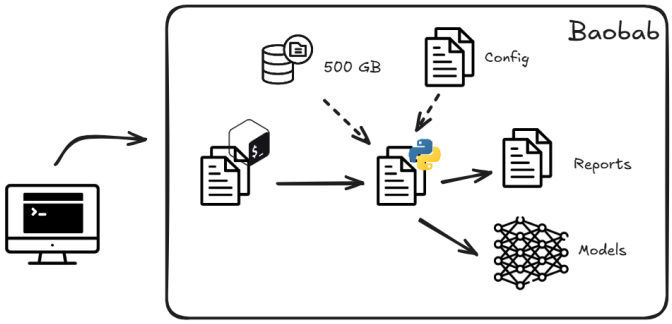


Figure 3.9: Environment on the cluster. It's close to a production environment. The access to it is done through a terminal of command with SSH. Files on the cluster are highly structured to make it clean and easy to understand. To execute jobs, a bash script must be associated to the corresponding Python script. The python scripts represented a single task for each of them. They allows to generate reports and models and store them on the cluster. To generate theses elements, Python scripts relies on a big amount of data and configurations files. The data is structured to be easy to use.

3.6.1 Baobab

Baobab is on of the three clusters provided by the University of Geneva to do high performance computing and calculation [17]. This cluster is adapted for parallel calculation and provides GPU and CPU units. This server is there for researchers requiring big infrastructure for their tasks. The

resources available on the cluster can be found in the documentation provided by the support [18].

The interest of using such infrastructure is multiple. A important amount of data is used in order to train models and theses models can be complex enough for a personal computer to struggle with computation. A personal computer have limited resources (space, memory, CPU capacity, ...) and working with large amount of data and big models required a lot of processing time. The cluster will reduce theses constraints and provide tools to run job on the different resources allocated.

In order to use the environment used in local, it's necessary to create on the cluster an environment. This environment is detailed in the section below (3.6.1.1).

It's also recommended to work with scripts to execute wanted tasks. Indeed, a lot of researchers access the resources every day, therefore requiring parallel execution and resources allocation. To solve theses issues, the server is using what can be called a resource manager : SLURM (3.6.1.2).

Other than computational resources, the cluster also provide storage for good amount of data. The spaces used during the thesis are detailed below (3.6.1.3).

The cluster is accessible remotely by using SSH and providing beforehand a SSH key to link the account. The account was requested at the start of the thesis to the support of the cluster. Files can also easily transferred from local to the cluster and vice versa by using SCP command [19] with the SSH key.

3.6.1.1 Production environment

To ensure reproducibility and better understanding of script behavior, Having a similar environment in local and on the production server is a must-have. Therefore, it's a requirement to provide the local configuration and packages on the cluster. A model system, Lmod [20], is in place on the cluster, allowing to load prebuilt configuration for different type of tasks. However, the current modules aren't enough to comply to the local environment with specific packages used. Therefore, another solution is possible : prepare its own container to run tasks called a Singularity [21].

A Singularity is used to run complex applications (with lots of packages and specific versions) on a cluster. It allow to have a simple and reproducible handling of applications. It takes the form of a container. The container is documented in one file with every library and corresponding versions documented.

To generate a Singularity container from a local environment, the first step is to export the current environment configuration as a YAML file.

```
conda env export > ctlearn.yml
```

Then, you should have a file containing every dependencies and corresponding versions like displayed on the figure 3.10. For the future steps, it's important to remove the section "prefix" from the yaml file.

```yaml
name: ctlearn
channels:
  - pytorch
  - anaconda
  - conda-forge
dependencies:
  - _libgcc_mutex=0.1=main
  - _openmp_mutex=4.5=4_kmp_llvm
  - aom=3.6.0=h6a678d5_0
  - astropy=6.1.3=py310h5eee18b_0
  - astropy-base=6.1.3=h0df7b8e_0
  - astropy-iers-data=0.2025.8.25.0.36.58=py310h06a4308_0
  - pip:
      - absl-py==1.4.0
      - annotated-types==0.7.0
      - anyio==4.10.0
      - argon2-cffi==25.1.0
      - argon2-cffi-bindings==25.1.0
      - arrow==1.3.0
      - asttokens==3.0.0
      - astunparse==1.6.3
      - async-lru==2.0.5
      - attrs==25.3.0
      - babel==2.17.0
```

Figure 3.10: Example of environment configuration saved in a YAML file. Generally, there are three big sections : the channels (packages repositories), the dependencies (installed via Conda) and the subsection for pip dependencies (installed via pip).

The file needs to be copied on the server to continue building the singularity. Once it's done, the last step can be executed. It implies a series of command to load a module prepared to help build the singularity and the corresponding command to store the container in a single file : ctlearnenv.sif here.

```
ml purge
module load GCCcore/13.3.0 cotainr
cotainr build ctlearnenv.sif --base-image=docker://ubuntu:22.04
    --accept-licenses --conda-env=ctlearnenv.yml -v
```

The Singularity is now ready to execute a task by calling it like the example below.

```
apptainer exec ctlearnenv.sif python3 -c "print('Hello World')"
```

3.6.1.2 SLURM

SLURM (Simple Linux Utility for Resource Management) [22] is a workload manager used on HPC instances from UNIGE. It's an open source and highly scalable tool to manage resources and schedule job on clusters. It's there to face the issue of allocation of resources simultaneously to lot of researchers working on the cluster. This tool is very popular because of its key features : allocation of access to resources for some duration of time, a framework to start, execute, monitor tasks of allocated nodes and finally an arbitrary contention of resources by using a queue system.

To put this in place, resources on the clusters are separated in nodes and included in partition for similar computational nodes. Theses nodes can either be CPU, GPU or others kind of computational resources. Having this system helps when a task (job in slurm language) needs to be executed by choosing an available space with all requirement met to work correctly. The usage of SLURM for

this thesis is really at a top level using only few of available features but more details can be found on their website [22]. At some point, it could be expected to run the different tasks of the thesis together by chaining them.

The details about the run configuration wanted is provided with a bash script explain in the chapter below. This script is then handled by SLURM by using the following command. This command will put the the job in queue and be started when resource are available.

```
sbatch script.sh
```

Some others commands not mentioned here can be used to monitor and analyze the execution of jobs on the tool.

Bash Script Structure

To execute tasks on SLURM, the most efficient way is to provide them by using bash scripts. Theses scripts will provide some details about the execution of the task. Each task has a dedicated script. The structure is still the same across all of them.

The figure 3.11 displays an example of script to interact with SLURM.

```
#!/bin/bash
#SBATCH --job-name=reduce-data-amount
#SBATCH --time=12:00:00
#SBATCH --partition=shared-gpu
#SBATCH --gpus=nvidia_geforce_rtx_3090:1
#SBATCH --mem=24GB

### Remove limit of files for training
ulimit -n 65535

### Execute the job
apptainer exec --nv \
  --bind /usr/local/cuda/targets/x86_64-linux:/usr/local/cuda/targets/x86_64-linux \
  --bind /usr/local/cuda/lib64:/usr/local/cuda/lib64 \
  --bind /srv/beegfs/scratch/shares/upeguipa/SST1M \
  ctlearnenv.sif bash -c '
export LD_LIBRARY_PATH="/usr/local/cuda/targets/x86_64-linux/lib:/usr/local/cuda/lib64:${LD_LIBRARY_PATH:-}"
python3 scripts/extract_only_images.py --input_dir /srv/beegfs/scratch/shares/upeguipa/SST1M/data/protons_diffuse/reduce_train/
'
```

Figure 3.11: Example of a bash script used to run a job on SLURM. It's separated in three sections : a part related to configuration of the job for SLURM, additional commands for the task and the main command to run the corresponding Python script.

The first section of the scripts is dedicated to the description of the job to run and the resources required to proceed. The information given are directly read by SLURM when a job is started. The list of attributes provided in the figure 3.11 shows the most important ones from a job configuration. Others exists but aren't use for this scenario.

- **job-name** : Name of the job (shown when monitoring tasks)
- **time** : Limit of time for the task to be completed
- **partition** : Group of computational node where task is executed
- **gpus** : Type and amount of GPUs required for this task
- **mem** : Memory required from the computational resource

The second part is rather small in this case. This contains all additional commands to execute before the Python script to make it work. In this case, a lot of files are used, therefore, the limit of files a script can work with needs to be extended. That's the purpose of the "ulimit" command.

The third and last part contains the command to execute the Python script, the core element of the task. It can be broken down as the following. The command start by providing the exection in a container runtime like previously seen.

```
apptainer exec
```

The first parameter of this function is there to allow NVIDIA GPU to work on the task. Related to that, there is also the binding of the CUDA libraries to make it work along with GPU. The GPU identification by the task has been a big challenge, more details can be found in the chapter 7.2.1.

```
--nv \
--bind /usr/local/cuda/targets/x86_64-linux:/usr/local/cuda/targets/x86_64-
                                            linux \
--bind /usr/local/cuda/lib64:/usr/local/cuda/lib64 \
```

Then there are additional bindings to do depending on where dependant resources can be found. Here, for example, data is stored on another file system, requiring binding for the Python script to see it.

```
--bind /path/to/wanted/data
```

The end of the container execution command is then provided with the singularity file and the type of task wanted. In this case, it needed to execute another command to link the CUDA library to the Python script by defining a variable. When all above is done, the script can be provided with the required input parameters.

```
ctlearnenv.sif bash -c '
export LD_LIBRARY_PATH="..."
python3 scripts/extract_only_images.py --input_dir ...
'
```

3.6.1.3 Dedicated space

As it was mentioned earlier, data is stored on the cluster in order to have a more efficient to generate models. The only issue with this is that depending on where data is stored, some constraints can appear in terms of space or reading speed. That's why two different spaces are used.

The first one is the personal space. This is the fastest way to read the data as the proximity between the scripts and data is ensured (locally close). The issue with the personal space is that it has a limited space of 1 TB, not convenient when using bigger amount of data.

The second is a shared space on a BeeGFS [23] file system. This file system is shared for each researcher working at the UNIGE on SST-1M satellite images. It has a bigger storage (10 TB) but the reading speed is a lot more important.

Depending on the advancement of the thesis and the needs for specific experiences, one or the other is used to generate models.

3.6.2 Calculus

Calculus is a server where data concerning SST1M Cherenkov telescopes. Lot of data is stored there for different applications (order of magnitude in TB). For the thesis, simulation data are the

main concern with the simulation of gamma and protons showers (gamma diffuse / proton diffuse) and gamma points. Theses different types of data will be useful depending on the task we want to generate model for.

4 Data

Data is an essential part of thesis. It's one of the most important subject to discuss. The complexity of the data used to train and test model is important. It's mostly composed of images obtained from a telescope, capturing events from the sky. A whole chapter details how the data is retrieved from the telescope (2). Here, a detailed analysis of used data is done by describing the format, content and processing of the data.

4.1 Prediction tasks

The gamma ray detection and identification implies to extract three different information from the image the event detected provided. It includes detecting the type of particle, the energy of the event and its direction. Theses three elements helps to understand the event and its related behavior for further analysis. Each of theses values are associated to a separated task that a model needs to be trained on. It helps the model to be more precise as it has only one focus. The different values depends on each over to be predicted but more details are coming below.

4.1.1 Particle Classification

The particle classification is the most essential task to execute on the events. The task itself is to classify between gammas and protons (hadrons) the events that are provided to it. For now, based on the data, there are only these two classes. So, right now, this a 2-class binary classification problem but it's possible that it can become a multiclass problem in the future.

This task is the central one of the thesis. It can be used as a filter to extract gammas events which are used by the others two others. Also when working with triggers, this task is employed to have an higher level of filters with a less accurate model and then, through the trigger levels having a precise model working only on filtered data. Background noise like the moonlight or light emitted from the ground will also falsify the results. Removing this noise or at least ignoring it should be considered.

The task works with gammas diffuse and protons diffuse events to work. Normally, the repartition of gammas and protons events is very unbalanced with protons having a 95% rate of presence.

The model will not generate the class it predicted for the event as most of models but rather computes a probability of the event to be a gamma event. Having something like this allows to understand better the behavior of the model and be more precise in the results. It also enable to use more ways to evaluate the model with specific metrics and graphics (brier score, etc.). The downside to this is that a threshold needs to be define in order to attribute a class to each event.

Sometimes, people tends to think that the threshold is always 0.5 and that below the threshold the class should be protons and above gammas. This isn't that simple. It can vary depending different aspects like the wanted precision, the purpose of the model (focus on recall or accuracy), ... This threshold can be trained and metrics and graphics can be used to determine the good one.

The importance of this task is to detect every gamma events and preventing to miss some of them. If a proton is misclassified as a gamma, it's not an issue, it's a loss of time. In the other case, it's critical as important information from the space could be missed.

4.1.2 Energy Regression

This second task is quite different from the first one. A regression model is built to predict the value of energy an gamma event is producing. This indicates that only gamma events (or at least expected) are provided to train the model. The model will try to approximate the energy of the provided event to the ground truth.

The regression means that different metrics and graphics will be used to evaluate the model. It also means that the performance of the model will never be exact and that it needs to be analyzed to see potential bottlenecks or misbehavior in the predictions.

The energy of an event can be found in a very high range of energy : 0.1 - 100 TeV. This means that the model needs to be able to work with different scales of energy and perform well on each of them. The evaluation process is therefore updated to see performance of the model on theses different scales of energy using bins and others separations between levels of energy.

As said previously, this task is conducted after doing the particle classification. It's expected to work only with gammas events, that's why the training is conducted on gamma diffuse event only and the testing on gamma point events.

4.1.3 Direction Regression

This task is the last one and is pretty similar to the energy task. The direction regression requires a model to predict the origin of an event by predicting the coordinates of that event. This model is also provided with only gamma events (or at least expected). The model will try to approximate the coordinates values and get the closer possible to the true direction the event is coming from.

The particularity of this model is the fact that two values needs to be predicted. The coordinates system used for this model is the Altitude-Azimuth system, specialized for locating objects in the sky. The performance of the model will never be exact and that it needs to be analyzed to see potential bottlenecks or misbehavior in the predictions.

A coordinate system as an output of the model means the evaluation is quite special. The coordinates needs to be evaluated individually and combined for the coordinate system in order to identify which coordinates is less accurate and the global accuracy of the model. The predictions are provided in degrees and are limited to a range of 0-90Âř for the altitude and 0-360Âř for the azimuth. The evaluation needs to be aware of the dimension aspect to choose correct metrics and graphics to visualize the details about the model.

As said previously, this task is conducted after doing the particle classification. It's expected to work only with gammas events, that's why the training is conducted on gamma diffuse event only

and the testing on gamma point events.

4.1.4 Particle type

The particle is rather simple. It's a binary classification problem with two classes identification : **Gamma-ray** and **Proton**. Gamma-ray is the important information that needs to be retrieved and protons are part of what is considered noise in the analysis of the data. Protons are part of a bigger category called hadron events resulting in a wider and more random spread of the shower. Each one have a distinct type of shower distribution, gamma-ray having a smooth and less random lateral distribution than protons (figure **??**).

4.2 Generation of data

Training models on complex task like the reconstruction of gamma-ray events requires large amounts of data. Data cannot be labeled by hands because of the complexity and time required to proceed. Therefore, another way must be introduced : Simulations. A simulation tries to artificially produce air showers for a gamma-ray events respecting certain restrictions and configuration. The advantage is that the ground truth (particle type, energy, direction) is already known and can be used to evaluate predictions tools and train models. The only issue with simulation is that it can't completely reproduce complex situations that occurs in reality such as the noise detail.

The simulation framework used for IACT data consists of many steps to provide a complete simulation [24]. The simulation works in two distinct steps using two different tools. It allows to generate realistic simulated data to analyze particle behavior and to train models efficiently. The figure **??** details which components does what on the simulation.

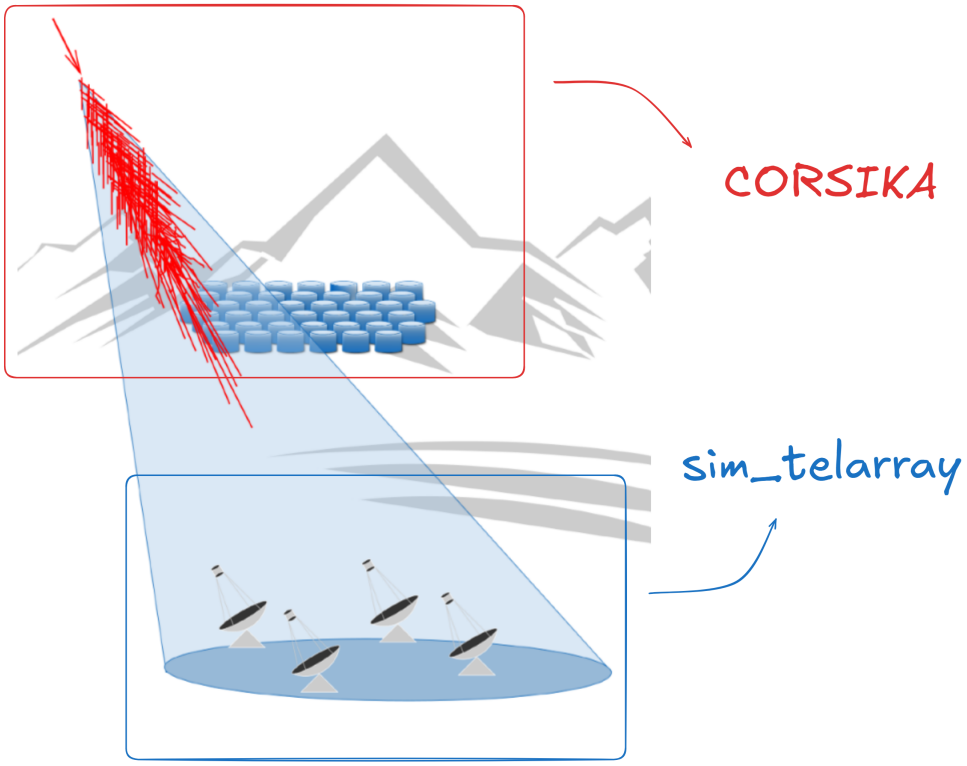


Figure 4.1: Visualization of parts of the simulation. The simulation can be separated in two elements : CORSIKA and sim_telarray. The first one focus on initializing a particle and build interaction with the atmosphere. It includes the generation of the air shower and related Cherenkov light. On the other hand, the second component focus on simulating the detector part : camera and telescope. It includes the triggering level system from it. [6]

The first component is the CORSIKA (COsmic Ray SImulations for KAscade) tool [25]. This tools simulates the interaction of primary particles with the atmosphere. It tracks the development of the air shower and related Cherenkov light emissions resulting from the interaction. The simulation is executed to have generated Cherenkov lights hit the detector layout. The simulation can be configured with many parameters like the particle type, energy range, altitude of the interaction or others. The treatment of hadronic and electromagnetic interactions needs to be separated. It provides also optimization processes to reuse generated shower but with different impact points.

The second component is the sim_telarray. This tool is oriented to simulate the telescope optics and camera response, in short, the hardware side. It simulates the detection of the events by the telescopes. It takes Cherenkov light generated by CORSIKA as an input to simulate the telescope and camera behavior. There are also lots of parameters that can be provided to vary the event detection such as the noise from the transmission and from Night Sky Background. It also provides a triggering system just like in reality to faithfully reproduce the detection. It finalizes by digitizing detected elements into waveforms samples adapted to future use cases.

The whole simulation process follows a Monte Carlo simulation approach [26] [27]. It tries to simulate randomness using probability distributions. It generates multiple samples according to theses probabilities on multiple parameters. With this, you can get an estimated behavior of a system without computing every possibilities, allowing to gain lots of resources and time.

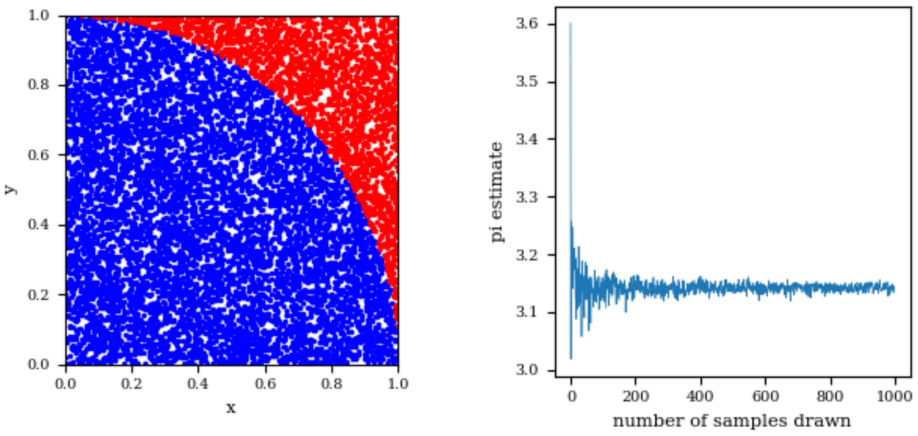


Figure 4.2: Monte Carlo simulation. Using the randomness of parameters, simulations have been produced in order to estimate the value of ÏĂ in this situation. It doesn't need to compute every possibilities to approximate ÏĂ as the right graphic displays. [28]

It starts by taking random samples for input to the system. As an example, for a IACT simulation, it refers to particle type, energy and direction but also context configuration such as the noise and others elements. Then the generation starts by doing many simulations with random parameters to cover statistically most of the scenarios. With all the simulation, it's, then, possible to check distributions, variance or uncertainties of the system simulated.

With IACT, this approach is essential because interactions of particles with the atmosphere are stochastic and the air shower development varies because of the total randomness of reality.

4.3 Format of files

The detected gamma-ray events are stored in various places across the working environment (Calculus, Baobab, etc.).

There is two types of files encountered in this thesis : .simtel.gz files and .h5 file.

The first one is a compressed CTA simulation file produced by "sim_telarray", one of the simulation tools. As it's said, the thesis works with simulated data and not real data. The reasons behind this choice are documented in the following chapter (**??**). It contains the raw simulated data with every details related to the generation of events. The files generated are too big to be kept in raw so Gzip is used to compress them. This allows to keep a massive amount of data on Calculus.

The second type of file is the .h5 extension. This second type of file is generally obtained after decompressing .gz files. The content of the file depends how the conversion from .gz files to .h5 files was conducted. Refer to the chapter dedicated to the conversion to see how it was done (4.4.2) and which elements are kept. To decompress the files, the library ctapipe is used as a conventional way to proceed and to keep an uniform structure of the data.

4.3.1 Content of HDF5 file

HDF5 or Hierarchical Data Format file [29] is, as its name says, a hierarchical container for managing and storing data. It has been format to easily manage various data, easily maintainable and understandable by others, have high performance with reading and writing of data, handles large datasets and various data structure using metadata system. This format is widely used by scientists and industries. Having everything stored at the same place isn't an issue as HDF5 is optimized to handle selective access by chunks.

This is exactly what's need for IACT Data because of variety and complexity of the data. Its design is built to enhance performance, scalability and reproducibility, essential with data used. It could be compared to an independent filesystem at this point. Some libraries help to handle this format in details with all the features related to it. This will not be explored in details as this is out of the scope of this thesis.

The structure of an HDF5 file is quite simple to understand. It's close to a directed graph in visual, with nodes linked together to form a hierarchy. Its includes three distinct elements: groups, datasets and attributes. Groups can be seen as directories or folders and content a category of data (e.g.: Configuration of simulations). It stores others groups and datasets. Datasets can be seen as files and stores information in multiple dimensions and datatype. It contains detailed values about a specific aspect of data. Attributes are the information in a format of key-value. It can be used to indicate provenance, versions or description of data. Additionally, attributes can be used at different levels and directly connected to groups for metadata. The figure **??** displays an example of HDF5 file architecture.

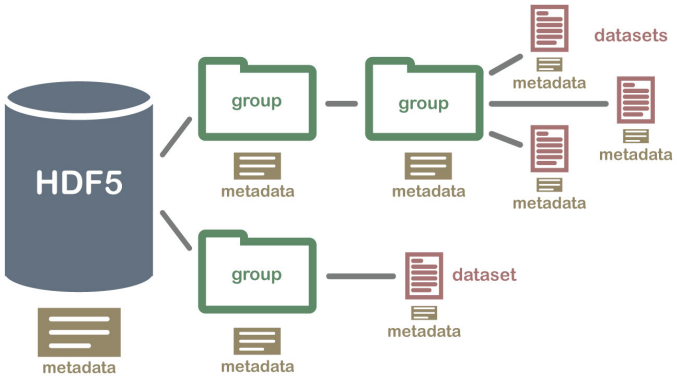


Figure 4.3: Example of HDF5 file structure with three elements (group, dataset, attributes) employed. By connecting groups and datasets, a structured dataset can be realized and separation between data can be optimal while having everything on the same file. Understanding and working with this format is easy because of the metadata distributed at each level of the hierarchy. [30]

Depending on the simulation and information kept, the content of the HDF5 file might change. Each chunk of data is dedicated to a specific state of gamma-ray events or simulation/configuration parameters. Here is a list of groups that can be found in theses files [31].

- **Configuration** : Processing parameters and software settings to produce data
- **R0** : Raw waveforms in each pixel (uncalibrated)
- **R1** : Calibrated waveforms (in photoelectrons, pedestal subtracted)

- **DL0** : Event-level pixel data before image parameterization.
- **DL1** : Integrated charge and peak position of the waveform in each pixel (Hillas parameters, frames).
- **DL2** : Reconstructed event parameters (energy, direction, primary type)
- **Simulation** : Simulation and Monte Carlo parameters

Data goes from waveforms (R1) up to reconstructed event parameters (DL2) by multiple processing and predictions steps to transform. Manipulating this data can be done at different levels to obtain various results depending on the use case. There is some ulterior steps like DL3 (IRFs) and DL4 but there aren't addressed during this thesis. The figure **??** give an overview of the data level workflow. It takes only in account data oriented event-wise starting from DL0.

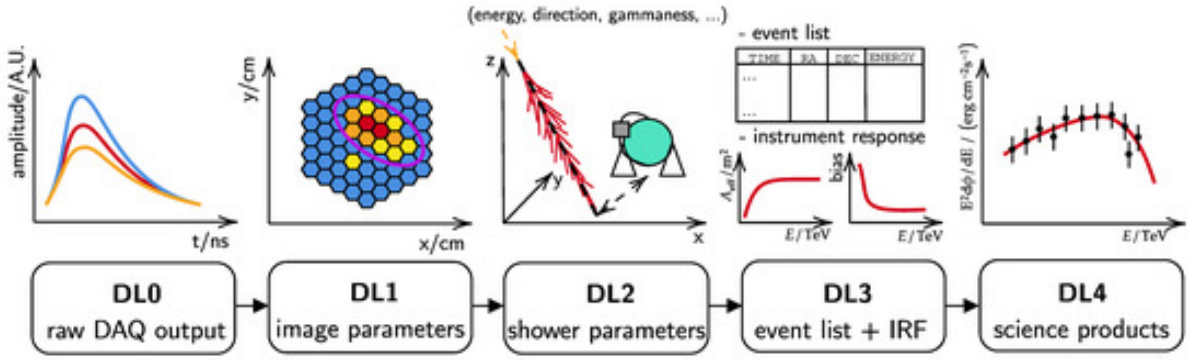


Figure 4.4: Workflow of data level (excluding R0 and R1). It shows that extraction of various information is done at every stage of data using the previous results. DL3 and DL4 are just for the show as they aren't part of the thesis. [32]

Configuration

Configuration group described how the simulation and data associated was generated. It contains metadata of the simulation, enhancing reproducibility of the simulation. The first thing it includes is related to the instrument used for the simulation. It includes optics for the telescope or camera geometry. There are also simulation details for the Monte Carlo such as the noise level the range of features from the air shower, schedules of simulation or even position of the observation. It also includes where the telescope is pointing and others aspects that helps to understand how the simulation is conducted. Metadata are also important in this part as they are used by multiple process like CTLearn library to verify quality and success of the simulation file.

R0 - Raw waveforms

R0 data [25] is the starting point of gamma-ray events. It corresponds as the lowest level of the data from IACT. It contains raw detector output, directly from the telescopes, values from the camera electronics without any modification or processing. This data is in a form of waveforms, one for each pixel, per each gain channel and per a determined time slice. It represents the on-line streamed raw data. A gain channel is used to read pixels of different amplitudes and find the most adequate way to read them. It includes parameters and information on events directly coming from the hardware and digitization related. Most of the time, it's not used for simulated data as the simulation is more stable like this and requires less parametrization. Theses waveforms are also

called ADC (Analog-to-Digital Conversion) waveforms to represent the digitized time evolution of the signal measured in a camera pixel [33]. The figure **??** represents R0 data in the most minimalist way.

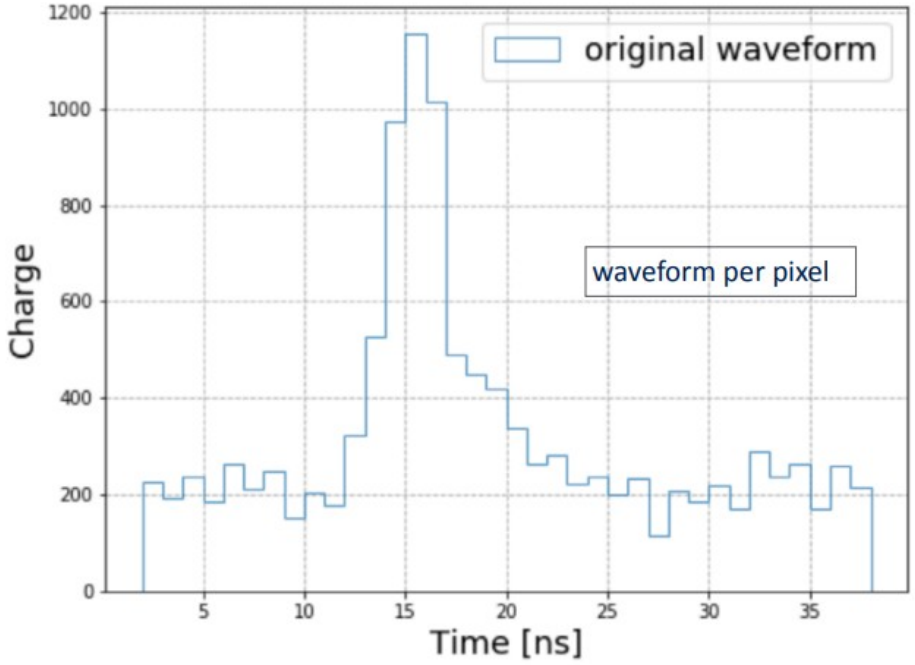


Figure 4.5: Example of R0 waveforms. It's a representation of the ADC counter, representation of the signal for a pixel, for a specific time slice with an integer value for each nanosecond in it. It's a representation of data detected by the camera of the telescope in the purest way. [6]

Data used during the thesis don't contains any dataset related to this part and therefore data content isn't detailed.

R1 - Calibrated waveforms

R1 data is also waveforms and it's the second level of data in CTAO environment. It's either directly obtained from the simulation (depending on settings) or processed from R0 data in two steps. An offset for each pixel is subtracted and the signal is converted from ADC to photoelectrons. Doing theses two steps are primordial if parameters and patterns needs to be extracted from the waveform. The noise is slightly reduced by proceeding like this.

The group related to HDF5 here includes the event id, the time of event, the gain channel selected, the waveform and small others interesting elements for the event. The figure 4.6 shows the processing performed on R0 data to obtain calibrated waveforms.

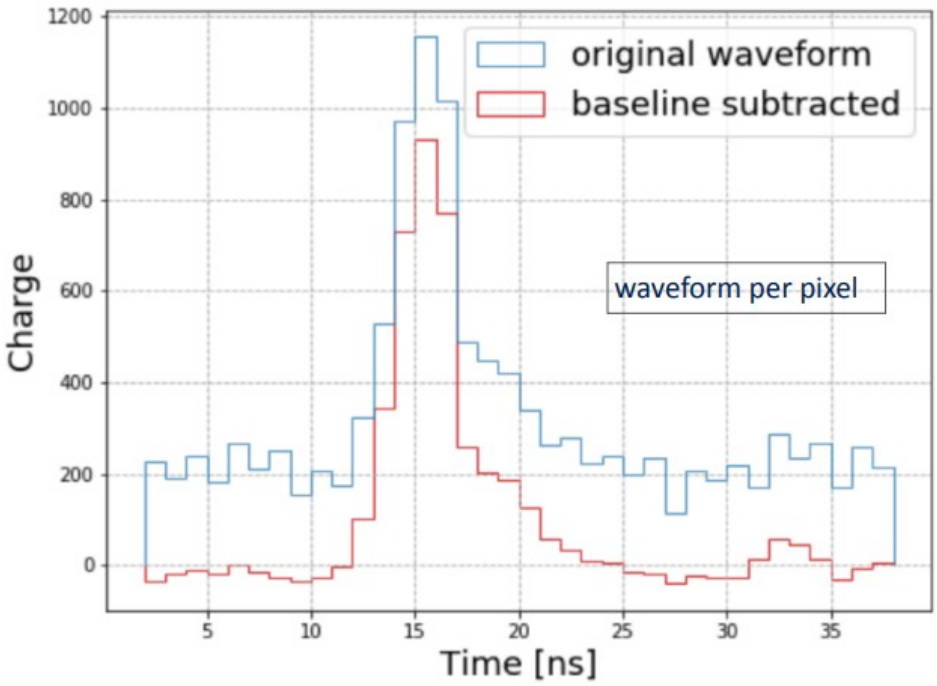


Figure 4.6: Example of R1 waveforms. This is calibrated waveforms with a clear distinction between the noise and the air shower visible. The photoelectrons are represented by the y-axis. [33]

DL0 - Archived data

This level of data isn't treated during the thesis but is worth mentioning. This level is there to keep only important data and reduce the amount of information by keeping only a small percentage of it. DL0 enables this reduction by using tools like the optimal gain selection. This small percentage is then used to reorganize the data into a structured format for features and patterns detection. It turns the information into events that are standardized across telescopes cameras. Globally, it can be said that this is a preparation step for analysis.

This step can optional, one of the reasons only few documentation mentions it. It serves as a bridge between the hardware data (camera and trigger system) and the physics interpretation and processing of data.

Data used during the thesis don't contains any dataset related to this part and therefore data content isn't detailed.

DL1 - Image parameters

DL1 data changes the current representation of images from pixel waveform to scalar quantities and per-event pixel arrays. The frame is a representation of the per-event pixel arrays with scalars representing an image from it. It extracts the charge integration, the pulse time, the gain channel, the Hillas parameters and some event details like the telescope used or the triggering type. It helps to give context to the event. The figure **??** illustrates the charge integration and the pulse time extraction where the information is contained.

The per-event pixel array is also cleaned in this data level to remove noise and keep only the Cherenkov light. It uses the pixel charges and arrival times to proceed. The cleaning of the array

isn't part of the thesis but still is worth mentioning. There are multiple configuration to proceed to clean arrays/images. Refer to this thesis for more details [34]. The cleaning allows to separate and highlight detected events.

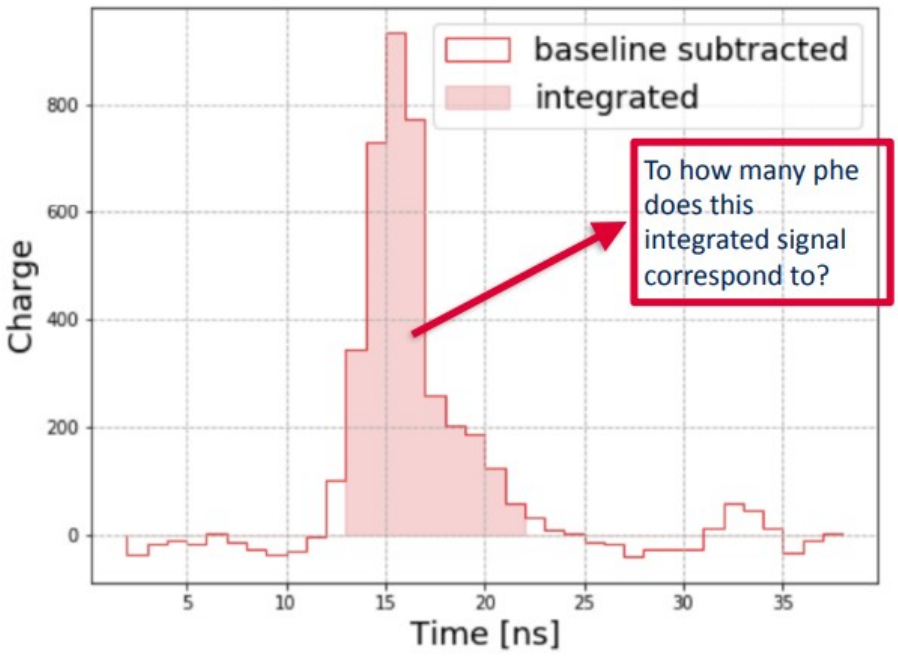


Figure 4.7: Extraction of the valuable information from the waveform. It takes the peak of the signal in account and keep a small window around it. Then the charge is calculated by adding up the charge in photoelectrons. This gives the value for the pixel. [6]

The figure **??** displays an example of Hillas parameters and how the parameters are determined.

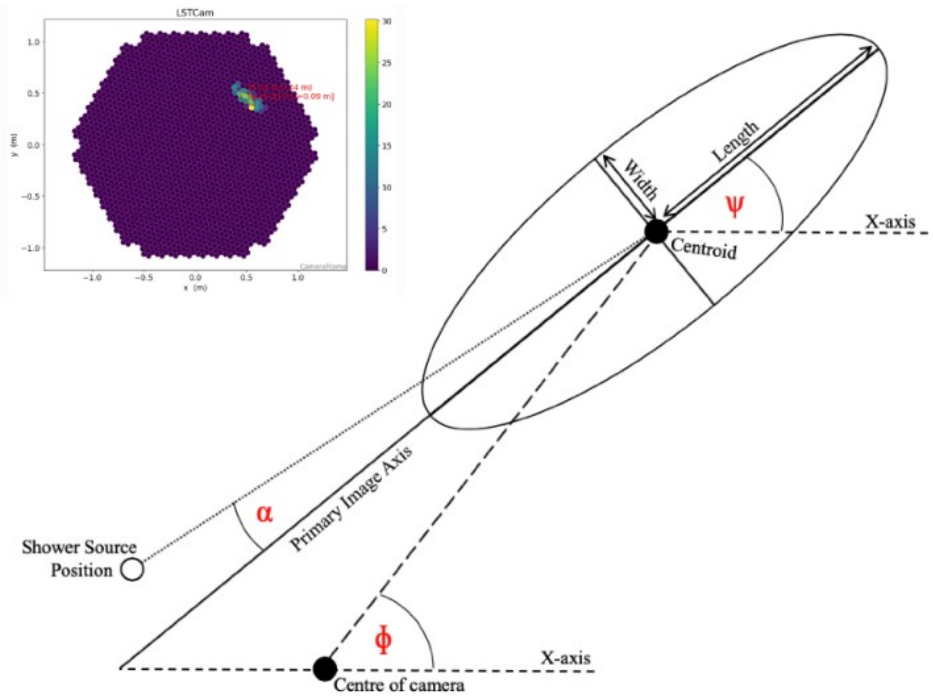


Figure 4.8: Hillas parameters extraction. This displays various parameters computed during an Hillas parametrization on a per-event pixel array. The parametrization is done on an event from a clean array as the picture includes. Some examples of parameters : Size, Width, Length, Center of Gravity, Leakage, etc. [6]

If a comparison is needed, DL0 waveforms (times series) grouped together represents a video and DL1 will gain the most important information from the video to build a single image. It does that by doing charge integration taking only peak of the signal from the waveform (event) and extracting it as a single value. It makes DL1 compact and uniform by reducing significantly information.

Theses elements are used to reconstruct metrics like the type of particle, the energy or the direction through algorithms and machine learning models by providing them theses parameters. On the other hand, Deep learning models tends to take the array/image to learn themselves the features.

This group contains multiple information like the triggers of events, the Hillas parametrization, the images and the image charge. It also has some statistics about images. It stores information per-telescope as images from a telescope to another differs from each others.

DL2 - Reconstructed events

DL2 is dedicated to the reconstruction of events. It summarizes the event by calculating information like the particle type, the energy or the direction of the air shower. This is obtained as a result of analyzing telescopes observations (one or multiple together) and apply them to algorithms or models.

In case of Deep Learning models, it directly works with pixels arrays to generate theses information. The figure 4.9 displays an example of pixel array representation. This representation needs to have some cleaning carried out and also to be converted to usable format for neural networks. The transformation step is required as current CNNs infrastructures works with a rectangle logic when the pixel array/frame is in hexagonal pixels.

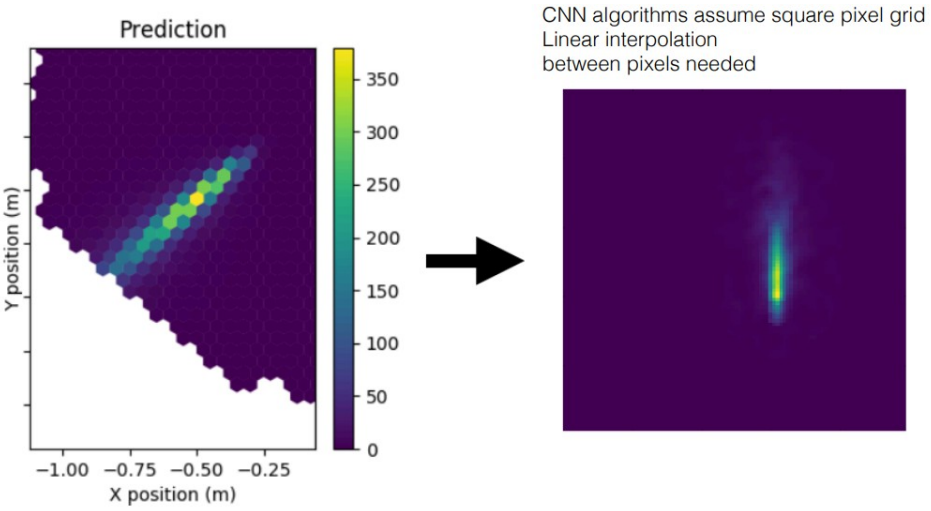


Figure 4.9: Example of an event in pixel array or frame. It represents on the left part of the pixel grid of the telescope image. Theses pixels are hexagonal and needs to be converted to rectangle ones to work with deep neural networks and classic libraries. [6]

It's used to evaluate performance of a model and see if it succeeded to identify important patterns and features. The data is compact and gives straightforward results and statistics for detailed analysis and for the conclusions to be drawn.

The group is composed of the result of the model. During the thesis, models were only trained on one task separately, so the group contains only result of reconstruction for a determined task.

Simulation

Simulation group isn't a data level as previously seen. This group is dedicated to the truth of the Monte Carlo simulation and stores ground truth elements such as parameters from the simulated air shower. It also includes additional information as the distribution histogram of an air shower, the simulated images from the camera or Hillas parameters derived from them.

It's useful to evaluate results of neural networks with the reality and get an idea of its precision. This data is only available on simulation data as for real data, there isn't enough data labeled to train a model. Events would need to be reconstructed by hands, taking too much time.

4.4 Compliance with the server

The data available on Calculus (3.6.2) cannot directly be used to train models and needs to go through some processing before being directly used on the server. This processing step is necessary for multiple purposes explained below.

The data used to generated model, and therefore transferred from Calculus are displayed in this chapter (3.6.2). The steps to obtain the correctly processed data were executed separately throughout the project, solving raised issues at the time to work with the data. The order in the subsections reflects the order of the data processing steps.

Normally, to use information, only the transfer of the data and the conversion of the data is

required. When going into details, issues with location and size of the data was raised, requiring modifications.

4.4.1 Transfer to the cluster

The first thing related to the data processing is a migration step. It is the act of getting the data stored on Calculus to the Baobab cluster to use it on daily basis. This is a one-time process as the data is used to explore models and isn't a production phase where data is transferred after they are generated. The data migration is handled with a single command used to transfer desired data from a folder.

```
rsync -av -e "ssh -i {SSH KEY}"
--files-from=<(ls | sed -n '{start_file},{end_file}')
. {cluster_address}:{destination_baobab}
```

This command, when executed in the folder, will take a determined amount of file by position (from starting point to ending point) and send it to the cluster. The cluster address needs to be provided before the folder where you want to store the data. To know which data is transferred, refer to this chapter (3.6.2).

4.4.2 Conversion of data

The data, after being transferred from Calculus, isn't ready to be used. Indeed, data is in a zip format, not usable by CTLearn. Therefore, the files needs to be unzipped.

A script using **ctapipe** library will convert the Simtel file (zip) in an HDF5 format. The script doing the transformation is available in the appendix. The script allows to precise what to keep from the zip file when unzipping. Depending on what is wanted, images, parameters or waveforms can be ignored during the decompression. The script applies procedurally on each file specified separately, not the most effective way to proceed but could eventually be parallelized on multiple CPUs in the future. The script is built so that only ".simtel.gz" files are unzipped.

The files aren't combined together during the transformation, ensuring the same number of compressed and decompressed files at the end of the process. The Simtel files aren't suppress at the end of the script because depending on the use case, others information, not kept for a scenario, could be necessary for another scenario.

In addition to that, another file is created when decompressing, the provenance log. This file is there to track the origin of an event. It includes processing steps applied to it, where does the data comes from and others similar details. The size of the file is big because of that. It's not required to have it to use the data. It can be suppressed to save space (done for the thesis) but additional information about the data provenance is lost like this.

4.4.3 Move Data across the cluster

Depending on the size and computation speed required for the models, Data location on the cluster changes. At a time, the space available where data was migrated wasn't enough. Therefore, a new location needed to be found. At another time, the speed to read files wasn't enough when generating models because of the location. Therefore, a new location needed to be found too. This

operation has been done a few times depending on needs in resources and time computation. Some details about the spaces used on the cluster can be found in the corresponding chapter (3.6.1.3).

Moving data across the cluster isn't complicate. It relies on using one command to move all data from a specified folder to another place (command **cp** or **mv**).

4.4.4 Merging files

Data on Calculus is stored in thousands of small files containing events. Each of theses files have different simulation configurations. When transferring data on the cluster, the number of files is kept. This is an issue when files needs to be read for multiple tasks because it slows down significantly processes and tasks depending on them. An issue related to that is well documented here (7.2.2). In order to reduce this I/O bottleneck, a tool from ctapipe library is available to safely merge files together : **ctapipe-merger**. The script is done to directly run a command with SLURM so the merge works correctly. There is a possibility to indicate in how many files the merging should result. By providing a directory, it will merge every HDF5 files wanted into a defined number of HDF5 files (merged respecting their order).

4.4.5 Keep only wanted data

Simulation data contains lots of different information about events like their format, parameters of simulation or even the waveforms related to them. Depending on the case, some information contained in the files might more useful than others. It could be useful to remove theses unwanted information to gain space on the cluster. With this in mind, a script has been made to extract only the frames and parameters of events. Waveforms aren't used during the training of the model making them unwanted data. The script cut the file to keep only wanted tables and information inside. This allows to cut almost 90% of the file as waveforms takes the majority of them. This process is executed for each file separately. It keeps only three sections : configuration (simulation environment like the telescope), dl1 (frames of events) and simulation (configuration of the simulation).

5 MLOps approach

MLOps or Machine Learning Operations [35] [36] is a methodology created to make machine learning models (deep learning and others too) more reliable, reproducible, deployable and maintainable in production. It's an extension from the DevOps [37] set of practices but oriented towards data and models. It focuses on the practical implementation and management of models.

Nowadays, large datasets are used to train various models in order to find the most efficient solution to predict a value, just like in this thesis. MLOps is the solution to handle this. Its aim is to optimize the usage of AI models and the data they used to save time and resources. Without using MLOps, the development of a AI solution required significant resources, lots of time and a really deep understanding of the process and handling of models and data. MlOps provides automation, reproducibility and efficiency in the development and production environment through deployment, monitoring and maintenance of models. The figure 5.1 shows the traditional MLOps life-cycle of a model.

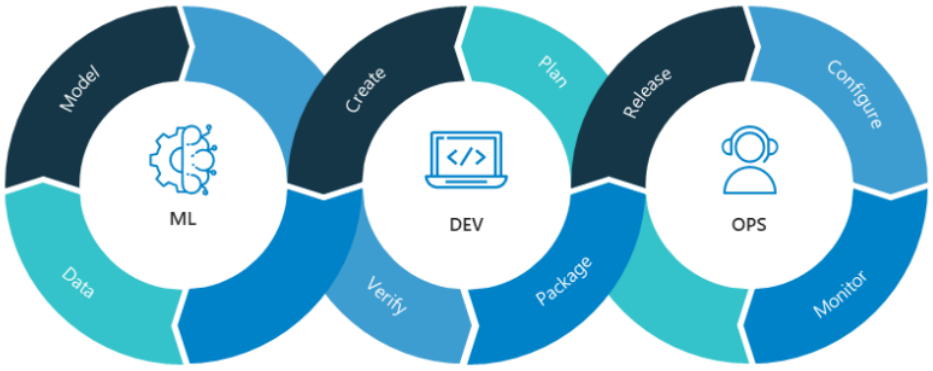


Figure 5.1: MLOps cycle representing the life-cycle for a model and the related data in order to deploy a model in a production environment. Deploying a model in a production environment isn't the end of the process as monitoring and maintenance are continuously provided to the deployed model, conducting eventually in a refinement of the model. This is mean the cycle can restart at some point. [38]

MLOps relies on a list of principle to build an adequate MLOps environment [39].

- **Collaboration** : Make communication between data scientists, engineers and relevant parties easier to understand the process in place
- **Continuous improvement** : Iterative approach where models are monitored and evaluated in order to stay accurate with the use case

- **Automation** : Automate repetitive tasks to avoid losing time with configuration
- **Reproducibility** : Ability to reproduce an experiment to analyze/debugging and compare results
- **Versioning** : Tracking of changes from a model to another
- **Monitoring and observability** : Analysis of model performance
- **Governance and security** : Secure access to model and data across the pipeline.
- **Scalability and security** : Scalable infrastructure for growing amount of data or incresing model complexity

As part of the thesis, some of theses aspects are prioritized like the automation, reproducibility and monitoring of the models results by analyzing and comparing them to each other. Another big part of the MLOps approach is related to data handling. In this scenario, data has been prepared for model usage in a separate state as CTLearn library process them differently. For more informations, refer to the dedicated chapter (4). Among elements of MLOps handled by the thesis, there are the experiment tracking with tools to generate report or compare models. For reproducibility and automation purposes, tasks has been separated and the code has been implemented to quickly understand parameters of the experiment and ways to reproduce it. The automation comes from the fact that the tasks can be chained and rerun in case of unexpected behavior without affecting the whole process. Handling models requires a defined structure, detailed, to facilitate takeover and usage of the tools.

MLOps is also there to close to gap between astrophysicists (in the thesis scenario) and data scientists, so each part belonging to one side can be easily understood and used by the other side.

This is a wild area with multiple ways and approach to incorporate this aspect. Tools and technics below are an overview of thesis elements that are related to an MLOps integration in the CTLearn environment. The environment doesn't include a total adequate MLOps environment as it could take forever but introduces some interesting elements that can be integrated easily to the CTLearn library.

5.1 Task Breakdown

One of the first aspect on the MLOps approach is to divide the pipeline in sub-tasks to be able to handle them separately. The general idea was to handle the whole pipeline to generate a model within a same task, therefore chaining everything after another. This is an interesting first way to generate a model as everything is at the same place and, when the code starts to run, it's fully completed in one run. This approach has some limitations when manipulations or rerunning needs to be proceeded. It's also not convenient because it's harder to identify where a problem occurs.

Going towards an MLOps methodology, this approach needs to be changed. The idea here is to separate this global job executed in a Jupyter Notebook in smaller components executing respectively a sub-task and are only dedicated to it. Theses tasks are separated in their respective Python script and can be executed separately in compliance with the requirements of the task. For example, the testing task requires a model already trained in order to work (so the train task should have be ran at least once). The decomposition in sub-tasks is represented in the figure 5.2.

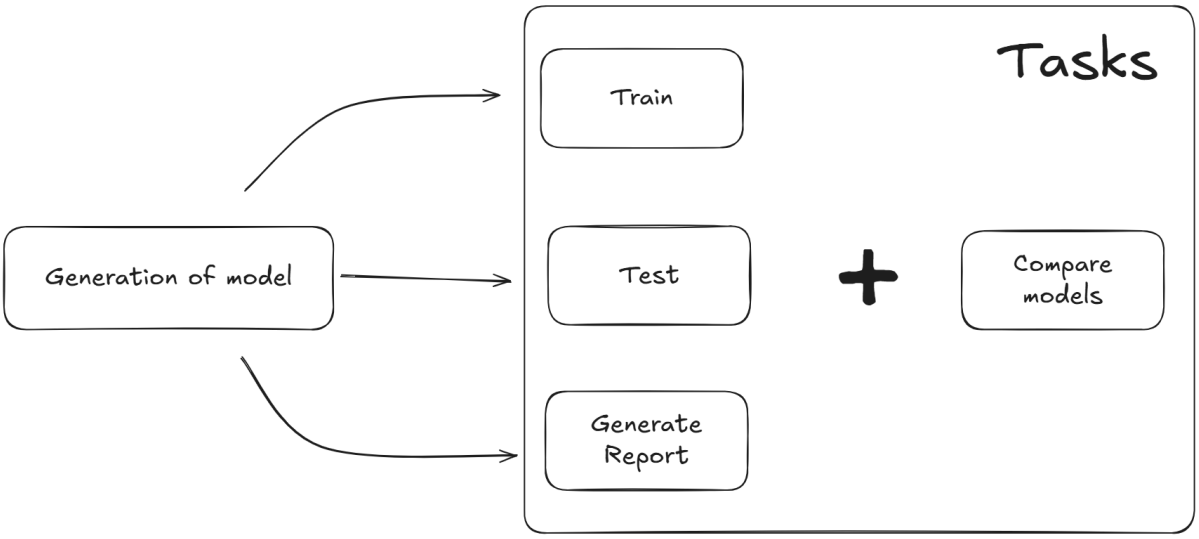


Figure 5.2: Task decomposition of the pipeline to generate a model and monitor it. The pipeline executed in a Jupyter Notebook is separated in three sub-tasks : training, testing and report generation of the model. This ensures more flexibility when executing tasks and more adapted to monitoring and maintenance purposes. Another task is included in this picture : comparison of the models. This tasks is outside the pipeline because it requires multiple models to work. It's considered in the task decomposition as its usage is the same as the others sub-tasks.

Breaking down tasks is useful for a pipeline as it makes the tasks independent, failure-prone and scalable. The tasks are then easier to monitor, maintain and update in an environment where new features are often implemented and multiple models are generated. It's also allows reproducibility and versioning of the tasks and models generated. The fault-tolerance is also important. Depending on the environment the script is run, restrictions could affect the success of the task, like limited time or resources. Having tasks separated avoids to have the whole pipeline failing and limits issue with the environment as tasks are not ran simultaneously. In case of failure, the failed task can be easily rerun without touching the whole pipeline. Information about the failure are easier to identify (place, what).

The task decomposition could be extended in future by including the data processing in the pipeline. In the thesis, data are processed in an earlier stage and never updated afterwards. For more details, refer the chapter dedicated to it (4.4).

5.2 Configuration files

Another element that can be included as part of the MLOps methodology are the configuration files. Configuration files are essential to avoid modification of code after each pipeline process. MLOps methodology states that the code and the model logic should not change from an experience to another. The only changes should be provided by the configuration file. An overview of a configuration file is displayed with the figure 5.3.

The configuration files already available are examples of configuration for various types of models with comments for possibilities of modification. The structure of the file is also a good way to understand needs and behavior of the pipeline with the CTLearn library. The separation has been made, so the tasks can look only at their allocated parameters to execute.

```yaml
prepare_model:
  model_type: 'ResNet' # ['SingleCNN', 'ResNet', 'LoadedModel']
  tasks: ['type'] # ['type', 'energy', 'cameradirection', 'skydirection']
  input_shape: [96, 96, 2] # Shape to determine
  num_classes: 2 # 2 for type classification, 1 for energy and direction regression
  temp_dir: '/home/users/v/varenneh/models/type/temp/' # Place to store temporary model (intermediate steps)
  CTLearnModel:
    attention_mechanism: null
    attention_reduction_ratio: null

#  CustomModel: # Only if Loaded Model
#    model_filename: "ComplexModel"
#    model_name: "ComplexName"

training_model:
  TrainCTLearnModel:
    output_dir: "/home/users/v/varenneh/models/type/resnet_batch64/" # Where model is saved
    input_dir_signal: "/home/users/v/varenneh/data/gammas_diffuse/train/" # Gamma values to use
    file_pattern_signal: ["gamma_*.h5"]
    input_dir_background: "/home/users/v/varenneh/data/protons_diffuse/train/" # Hadrons values to use (not for regression)
    file_pattern_background: ["proton_*.h5"]
    model_type: 'LoadedModel' # ['SingleCNN', 'ResNet', 'LoadedModel'] (same as in prepare_model)
    reco_tasks: 'type' # ['type', 'energy', 'direction'] (same as in prepare_model)
    n_epochs: 100
    batch_size: 64
    overwrite: true
    quiet: false
    percentage_per_epoch: 1.0
    log_level: "DEBUG"
```

Figure 5.3: Example of configuration file for a model generation. The task handled by the model is the particle classification of telescope images. The structure of the configuration file is important and must be kept as such to work correctly. New parameters can be added depending on the configuration wanted and available attributes in CTLearn library.

The configuration is also a good way to track the experiment, by combining it with others tools like Git. If the MLOps is extended to this point, it could be possible to see what changes improved the model with the versioning. Also, when it comes to comparing models, it could be easily done by comparing configurations. The configurations files are saved at the same place as their model, enabling a additional description of the model generated. The reproducibility is then ensured as all parameters are directly available. It's also useful for scenario where a error is detected in the pipeline, case where rerunning problematic tasks with the same configuration file makes the experiment consistent and still accurate. Theses files are also a safer and transparent way to work with an external library like CTLearn instead of directly working inside a Python script.

5.3 Report Generation

Report generation is the last step of the model generation pipeline. It can be seen as a way to control the model generated and its associated performance. It gives insights that can be analyzed in a centralized place by experts to determine if the model generated is accurate as expected.

Sections below describes the different elements that can be found in the report used to evaluate a specific model. There are multiple sections available in the report depending on the task. Some graphics and metrics are dedicated to a type of task, therefore not working nor displayed for others tasks than the one to which they are dedicated.

- **Generic Section** : Metrics and graphics in every report for details about the model and runtime performances

- **Particle classification** : Metrics and graphics dedicated to this task
- **Energy regression** : Metrics and graphics dedicated to this task
- **Direction regression** : Metrics and graphics dedicated to this task. It's the same for both Camera direction and Sky direction regression.

Like others tasks, the report is generated using a Python file with all metrics and graphics ready to be used. It requires to have a fully trained and tested model in order to work.

Reports helps to go towards the MLOps approach by providing an detailed validation with centralized information. This can be used either for decisions by users or by automation from specialized tools. Graphics provides a debugging view of the model to identify at which point problems are encountered. It's also a collaborative tool as different stakeholders can see what they are concerned about (astrophysicists with performances, engineers with learning curves, etc...). It records an overview of the output of the model enhancing the reproducibility of it.

The report generated is used to compare and get a quick overview of the model performance. There are also enough metrics and graphic in order to identify potentials lacks of the model. It reveals if the model is instable or overfitting. The visual aspect used come directly from a template generated by ChatGPT (for quick usage). The metrics, on the contrary, where chosen after analyzing possibilities to evaluate the different tasks on internet and on the CTLearn Manager library [40].

5.3.1 Generic Section

In a report for different types of models, there are some generic information displayed without looking at the type of task. Theses information gives some extra-information not related directly to the performance of the model. There are here to enhance reproducibility and get ideas of the environment used to generate and use the model.

5.3.1.1 General information

When building a report, some general information are useful to be displayed. Theses information are the same for the different type of models and helps to understand a bit better the environment used to produce the model and could help in case of reproducibility.

The timestamp of generation of the report and a clear definition of the task the model is built for are an example of information displayed.

The number of epochs are an important additional information to understand the model. It describes the number of time the dataset is passed through the model. It helps to learn patterns and features progressively. In the report, the information about the epochs includes the scheduled number of epochs to train the model and the number of epochs effectively used to train the model. This number can be lower than the first one if some early stopping has been defined in the configuration of the model.

In the same area, the batch size is also an important information to understand the behavior of the training. It describes the number of samples combined to update model weights when training. It helps to reduce the number of time the model is updated during training by averaging the loss of the batch samples.

The last information provided is the number of training and testing events used. The data environment is quite big and complex, conducting to potential problems while reading data. Theses information helps to know if the data was handled the same way by different models for the training and for the testing.

5.3.1.2 Model and Runtime Metrics

In order to compare and evaluate different models, it's required to look at how well they perform but also the resources required to make them work. With this in mind, some metrics has been defined to get an idea of the requirements for a model. Theses metrics can be related to the size of a model, the number of operation they need to compute a result or the time needed to complete a certain task.

Theses metrics are shared and used for each task as they display an overview of a model (without specifics).

Depending on the environment and the use case of the model, the interest of some metrics and their value can shift slightly. For example, the importance of metrics isn't the same for a model on an FPGA or a model with a lot of computation resources.

Number of layers and Number of parameters

Theses two metrics have a common purpose: determine the complexity of a model. The number of layers can be seen as the depth of the model, an essential part to explain the complexity. The number of parameters goes along with the number of layers and provides an idea of the complexity of a layer to provide information.

Having a bigger number of layers and parameters allows the model to identify more features and interesting patterns in the data used for training and, therefore, helps to obtain better results on a model. On the contrary, having less number of layers and parameters ensures a lower complexity, sometimes useful on infrastructures with limited resources.

For similar performances, the model with less parameters and layers should always be chosen to reduce the size in memory and the complexity of the model.

Estimated GFLOPs

This metric is related to the previous one (5.3.1.2). It evaluates also the complexity of a model but with a different angle. Instead of looking directly at complexity and memory usage, it calculates GFLOPs required to predict a value. This approach is more production-oriented and helps to understand computational costs of the model.

GFLOPs corresponds to Giga Floating Point Operations, Floating Point Operations being the unit of mathematical operations required for one forward pass (for a CNN model). The number of GFLOPs changes from a model to another depending the complexity, the architecture or the data used to predict it. This unit is directly related to the inference speed of a model and the energy consumption related to it. Theses two elements are very useful when environment and use case of a model are important.

A smaller value of GFLOPs means the model infer faster and consumes less energy. Minimizing this value is interesting when working with limited infrastructures.

Training Time, Inference Time & Inference per event

Theses three metrics are related but they have little difference between them. The objective with them is to calculate the time taken to have an operational model trained and tested, ready to use.

The training time includes the loading of the data, building of the model using the CTLearn library and the training of the model. The data loading is done in order to adapt the data provided to a format where a TensorFlow model can be trained. The data format is a bit special for telescope data in the scope of the thesis and adjustments needs to be made for it to work (Refer to chapter for more details). Each of this steps takes a certain amount of time depending on the size of the dataset but also depending of the configuration wanted for the model (number of epochs or complexity of the model).

The testing time includes only one task : testing of the model. It means predicting each event contained in the provided files and generate a corresponding output file containing the predictions.

The inference per event is just the mean time for the model to infer on one event. It's simply calculated by taking the total number of events and the total testing time.

It helps to get and idea of what is taking the most time between training and testing a model. The inference per event is an essential metric depending on the use case to know which model can do predictions faster. Minimizing this metric is the best thing for production purposes. It's also often used as a trade-off with precision of the model. It's difficult for both of them to be excellent, so a focus on one of them is needed.

5.3.1.3 Graphics

Graphics are pretty useful to display a lot of information and understand it quickly. Theses kind of elements helps to recognize patterns and also helps to compare different models. It helps to determine where problems could lie and also communicate more clearly information. Theses graphics will be used by every task and gives general information about a model.

Training Loss

This plot graphic is used to show how training loss and validation loss evolves over the epochs. It shows the behavior of the model over time and can show if the model is learning correctly.

The main idea is to minimize the loss over the epochs (both for training and validation) and have a smooth curve. Irregularities in the curve are due to an unstable training and require sometimes modifications on the structure or the batch size. A lot of disparity between the two curves could also indicates issues with the training like underfitting, overfitting or vanishing/exploding gradients.

This is a useful graph when talking about training purpose and give hints about problems on evaluation of the model with metrics and others graphics.

The figure 5.4 displays an example of training loss graphic.

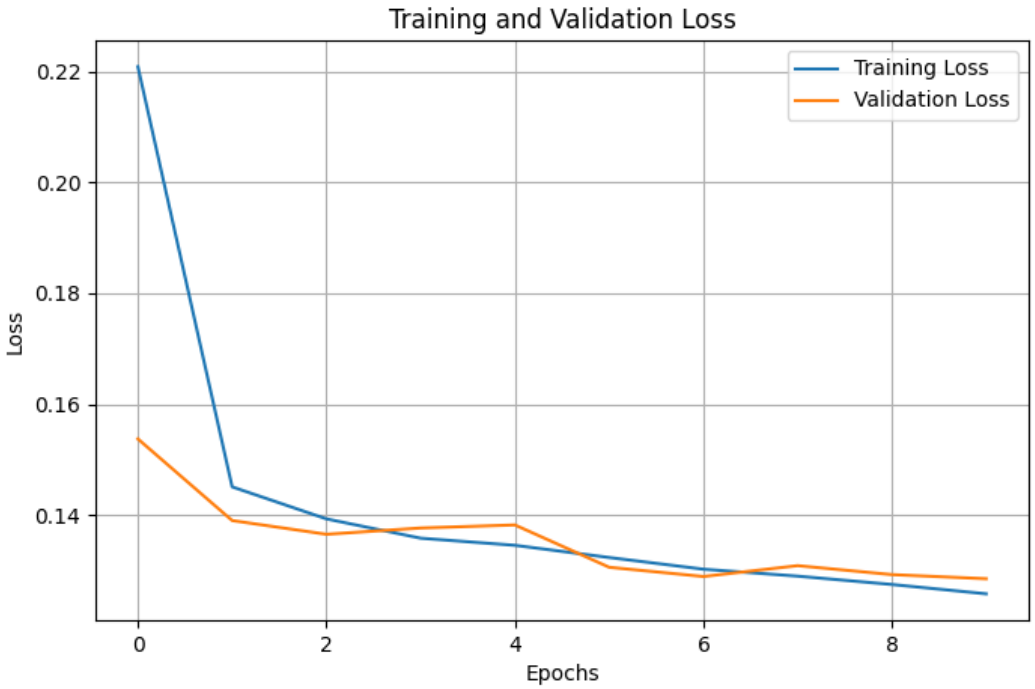


Figure 5.4: Graphic containing the training loss and the validation loss obtained through epochs. This helps to understand possible problems during training like underfitting or overfitting. This could also indicates that the current number of epochs isn't enough to converge.

Model Summary

This one is a bit tricky. It's not really a graphic but rather an architecture display directly from a function available in TensorFlow : **model.summary()**. This command displays as a text the structure of the model by showing connected layers with their name, type, output shape and the number of parameters they contain. Additionally, the total number of parameters and trainable parameters is displayed with the size of the model. This helps to understand the architecture of the model in depth and the input/output format of the model.

The figure 5.5 demonstrate an example of model summary display.

```
Model: "CTLearn_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 96, 96, 2)]       0

 ThinResNet_block (Function  (None, 1024)              5357600
 al)

 fc_energy_1 (Dense)         (None, 512)               524800

 fc_energy_2 (Dense)         (None, 256)               131328

 energy (Dense)              (None, 1)                 257

=================================================================
Total params: 6013985 (22.94 MB)
Trainable params: 6013985 (22.94 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 5.5: Model summary containing information about layers of the model. Additional information like the total number of parameters and the size of the model are provided.

5.3.2 Particle Classification

Th particle classification is one of the tasks handled during this thesis. For more details, refer to the chapter (4.1.1). This sections includes metrics and graphics especially oriented towards evaluating this specific task and get insights from the results obtained.

5.3.2.1 Evaluation Metrics

In order to compare and evaluate different models, it's required to look at how well they perform in their task. With this in mind, some metrics are defined to give a global overview of the results obtained based on multiple criteria. Theses metrics can show different problems encounters with the data and depending on the use case, have a different importance and priority to comply to. Metrics for classification can be rather simple to obtain or complex depending on the case.

In this particular case of classification, it's required to be aware of one thing. The classification is provided as a probabilistic prediction. It means the prediction isn't directly a class label but rather a probability of the event belonging to a class (in the task case, it's gamma probability). However, commons metrics for classification works with class labels, not probabilities. For this, there is a need for a decision threshold in order to classify the values. In this classification scenario, with a binary-classification scenario (gammas or hadrons), it's really simple to classify the values. Under the threshold, values are considered as hadrons and values above are considered as gammas.

Accuracy

The accuracy is one of the most common metrics used to evaluate classification. It calculates the percentage of values correctly classified for the whole provided dataset. This metric is really simple but can sometimes lead to misinterpretation depending on results and the use case.

For the use case, the positives are the gammas and the negatives are the hadrons.

$$accuracy = \frac{\text{true positive} + \text{true negative}}{\text{true positive} + \text{true negative} + \text{false positive} + \text{false negative}}$$

Recall

The recall (also called True Positive Rate) is another common metric used to evaluate classification. It has a slightly different behavior. It calculates the ability of the model to correctly classify all values for a specific class. This metric can be very valuable for scenario where it's critical to not miss any positive even if false positive are increased. It focus on false negative.

For the use case, the positives are the gammas and the negatives are the hadrons. We, therefore, focus on classifying correctly all the gamma events.

$$recall = \frac{\text{true positive}}{\text{true positives} + \text{false negative}}$$

Precision

This metric is not directly used in the report but rather as a subpart for another metric : F1-Score (5.3.2.1). This metric works in a similar way than the recall. The metric, this time, calculates the capacity of the model to recognize values for a specific class. It focus on false positive.

For the use case, the positives are the gammas and the negatives are the hadrons. We, therefore, focus on the capacity to correctly classify gamma events.

$$precision = \frac{\text{true positive}}{\text{true positives} + \text{false positive}}$$

F1-Score

This metric is used as a trade-off metric for two others metrics seen earlier : precision and recall. Normally, only one of theses two values can be maximized while penalizing the other. Therefore, an optimal solution needs to be found to balance both of them in an optimal way. That is the F1-Score.

$$\text{F1-Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Brier Score

The Brier Score is different than previous metrics. Instead of having a binary classification focus, like the accuracy or the recall, it takes directly the probabilistic predictions to evaluate the quality of the model. It's used to evaluate the calibration of the model. This metric also penalizes confidence in a wrong prediction

With others metrics, there is no distinction between a probability of 0.51 and 0.87 (both gammas in this case) even if the second value is better. Brier Score solves this. It measures how close a

model predicts probabilities compared to the ground truth. It can be seen as the mean squared error between probabilities and ground truth, and it can be referred as a cost function.

$$\text{Brier Score} = \frac{1}{N} * \sum_{i=1}^{N} (p_i - y_i)^2$$

5.3.2.2 Graphics

Graphics are pretty useful to display a lot of information and understand it quickly. Theses kind of elements helps to recognize patterns and also helps to compare different models. It helps to determine where problems could lie and also communicate more clearly information. Theses graphics will help to understand the classification better.

ROC Curve

ROC Curve (Receiver Operating Characteristic curve) is graphical plot used for particle classification. It represents the ability of a model to identify differences for a binary-class problem (in the thesis case, gammas versus protons/hadrons). It uses the trade-off between the True Positive Rate (5.3.2.1) and the False Positive Rate (proportion of negative instances incorrectly classified as positive) to build the curve. It's a useful graphic as it doesn't take in account class imbalance unlike others metrics, useful for real-time application where gamma events are rare.

It can be used, for example, to determine a good threshold for probabilistic classification model (**??**) or to compare models.

The ROC Curve also comes along the AUC (Area-Under-the-Curve) metric to get a summary of the curve performance. It corresponds to the probability of the model to classify a random gamma event higher than a random hadron.

The figure 5.6 is a clear example of a ROC curve representation with a legend displaying the AUC value.

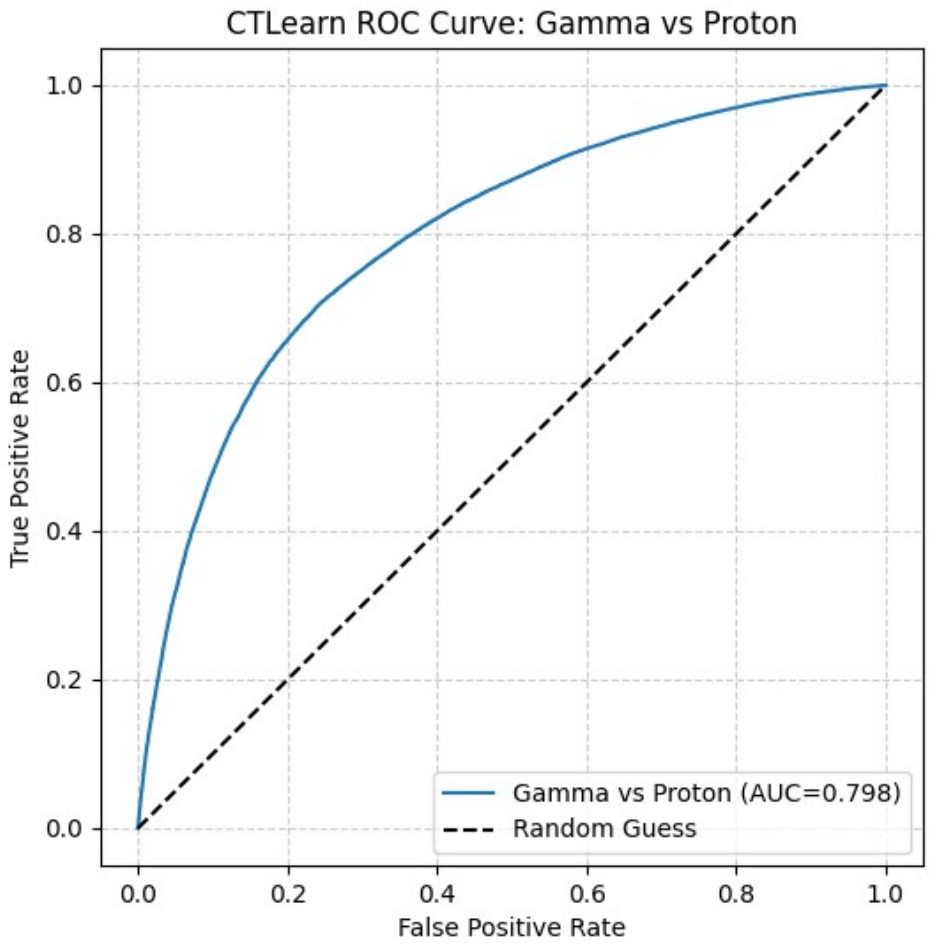


Figure 5.6: ROC Curve graphic displayed using TPR and FPR. There is also a diagonal displaying random guess values. The legend display the model label and the AUC to get a better overview of the model performance on the curve.

If the curve is close to the random guess diagonal, it means that the model guess randomly. Going closer the top-left corner indicates better performance.

Gammaness Distribution

This graphic will displays repartition of data (gamma and hadrons separately) over the probabilistic predictions obtained. The probability score obtained after classifying events is called the gammaness and it's what is displayed here. It's helpful to see if the model can separate the gammas and protons clearly. It's also helpful to choose a threshold for the classification as it displays more directly at which level of gammaness the gamma are the most correctly classified.

The figure 5.7 displays with two distinct colors the distribution of gamma/hadron events at each level of gammaness.

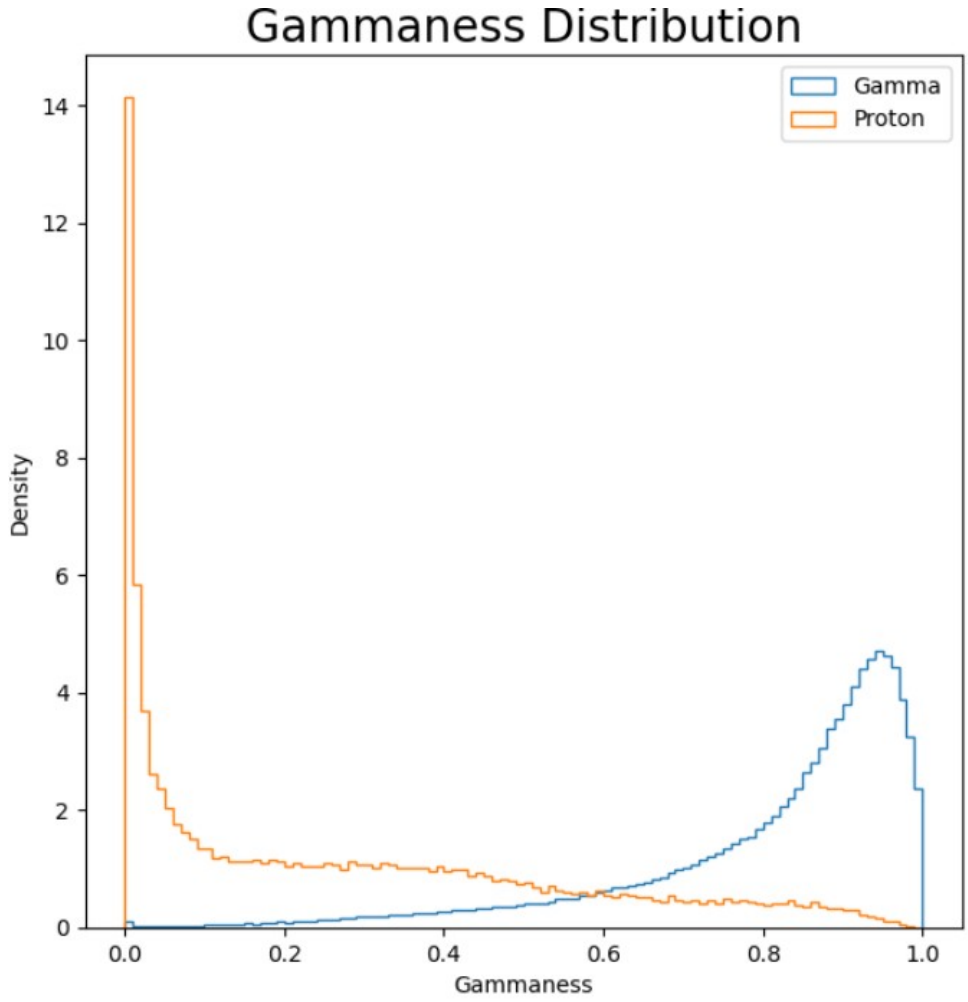


Figure 5.7: Gammaness distribution graphic where the distribution for each type of particle can be seen. The axis used are the gammaness and the density of events contained at each level of gammaness. In the example, the separation between gamma and hadron can be seen. It's not clear which means some improvements could be made.

Confusion Matrix

The confusion matrix is a table used to display classification results. Using this table, it's possible to calculate easily the accuracy, recall or precision of a model as it compares predicted classes with ground truth classes. It also gives hints about potential unbalanced dataset but providing the number of events classified correctly and incorrectly for a given class. It requires the predictions to be label of classes and not probabilities.

The figure displays a confusion matrix for the classification problem of the thesis (gamma versus proton/hadron).

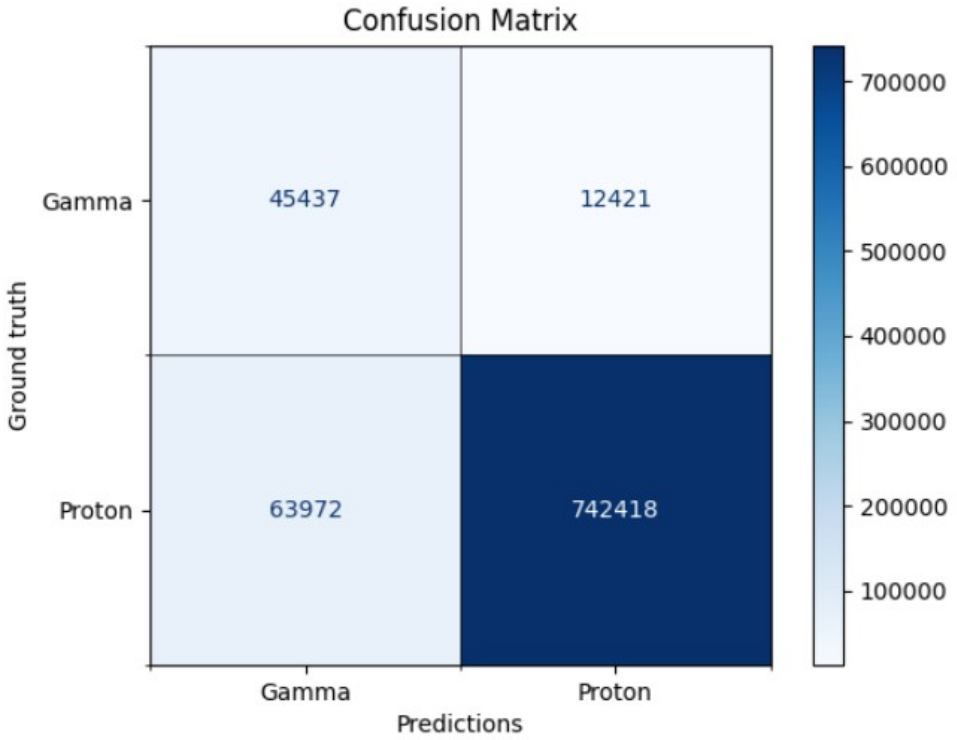


Figure 5.8: The confusion matrix displays a 2x2 matrix (in the task scenario but could be more) described gamma and hadrons events classification. It shows if values were correctly evaluated and the corresponding number of values associated to the scenario. Colors are also displayed to get a quick idea of where most of the data fall. The x-axis displays columns corresponding to predicted gamma and hadron events and the same goes on the y-axis for the ground truth.

Calibration probability

This graphic is related to the brier score (5.3.2.1). The brier score gives an overview of the calibration of a model where this graphic gets more details about probability mismatches. It measures the reliability of a model by evaluating probabilistic predictions based on the likelihood of an event belonging to a class.

A model can be accurate but not well calibrated. For example, if the prediction probability is 0.8, it's expected that the model is correct 80% of the time. This graph helps to know if probabilities can be trusted.

The graphic works by grouping predictions from a same probability range. A mean probability is, then, computed along side the true fraction of gamma events contained in this bin. With this, observations can be made to determine if model confidence and calibration at each level of probability are well calibrated.

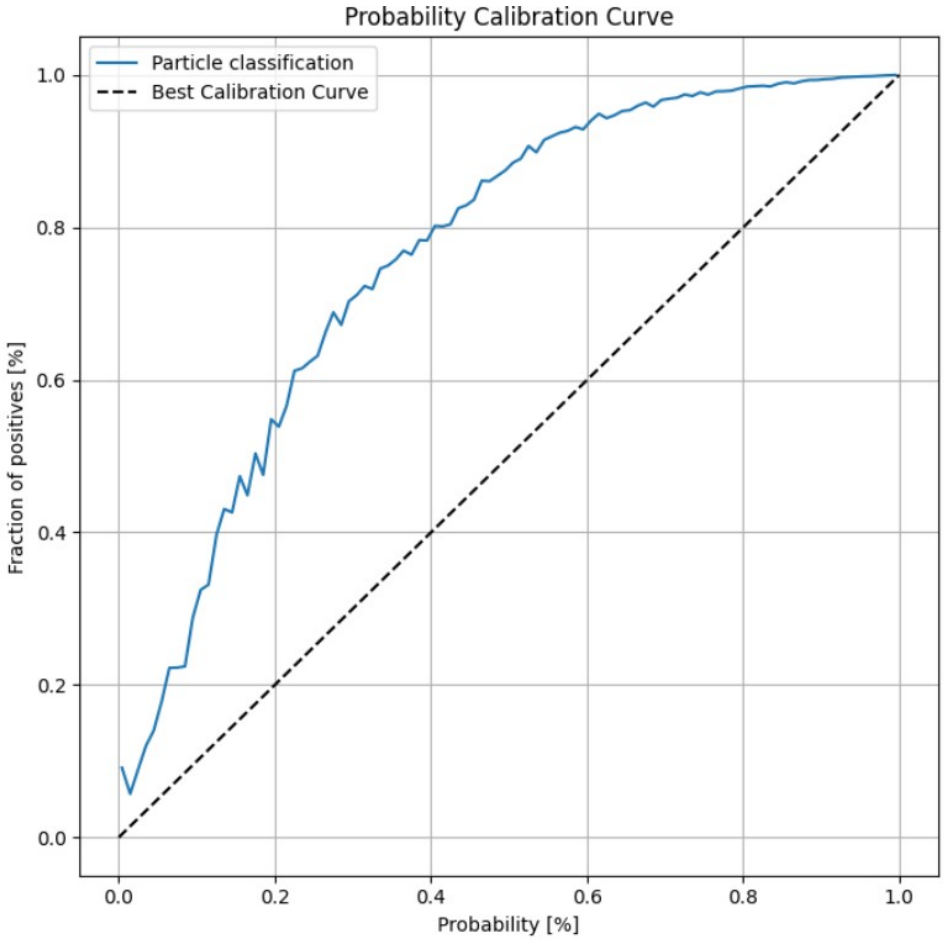


Figure 5.9: Calibration probability graphic displays a curve corresponding to the calibration for a certain probability. It realizes on the probability (on x-axis) and the fractions of positives (percentages of gammas events for a determined sample). A default diagonal is there to display the best calibration possible. The curve can be above or below the diagonal depending on the calibration obtained.

The black diagonal indicates the objective and also the best possible value for the calibration. It's expected to get closer as possible to this. Going under the diagonal means the model isn't confident and going above it means the model is overconfident.

5.3.3 Energy Regression

The energy regression is one of the tasks handled during this thesis. For more details, refer to the chapter (4.1.2). This sections includes metrics and graphics especially oriented towards evaluating this specific task and get insights from the results obtained. This task is normally conducted after doing the classification task. Therefore, the evaluation is mostly conducted only on gammas values.

5.3.3.1 Evaluation Metrics

In order to compare and evaluate different models, it's required to look at how well they perform in their task. With this in mind, some metrics are defined to give a global overview of the results obtained based on multiple criteria. Theses metrics can show different problems encounters with

the data and depending on the use case, have a different importance and priority to comply to. Metrics for regression can be various and complex due to the type of task.

MSE

MSE (Mean Squared Error) is one of the most common metrics used to evaluate the accuracy of regression models. It measures the how far the model predictions are from the ground truth. To do so, it takes the average squared difference between predicted values and the ground truth. The difference is called the residual.

It penalizes large differences between the prediction and the actual value. It means it's sensitive to outliers. The lower the value of this metric, the more accurate is the model.

$$\text{MSE} = \frac{1}{N} * \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

RÂš

RÂš (Coefficient of determination) is a metric used to analyze the variation in the data. It tells how much the model can explain the variability in the data. It evaluates the quality of a regression model.

It measures the percentage of variation the model can capture. The higher is the value, the better is the model in general (sensitive to overfitting). This metric can't be used alone because of its limitations (sensitive to noise or not working on non-linear relationships). This limitations can get results lower than 0 for the coefficient of determination, meaning that the model is worse than a basic prediction of the mean.

$$\text{RÂš} = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

RMSLE

RMSLE (Root Mean Squared Logarithmic Error) is a similar metric to MSE (5.3.3.1) but measure the difference between the logarithms of predicted and ground truth values. It's useful because it cares more about the relative error rather than the absolute error like MSE.

It isn't sensible to outliers like MSE (5.3.3.1) because of the scaling brought by the logarithms. This metric is well adapted for energy regression because of the multiple orders of magnitude (0.1-100TeV) that is predicted. By using relative error, the focus is more on the percentage of error rather than the absolute value of error.

$$\text{RMSLE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\log(1 + \hat{y}_i) - \log(1 + y_i))^2}$$

MAE

MAE (Mean Absolute Error) is a metric dedicated to measure the average difference between predictions and ground truth. It doesn't consider underestimation or overestimation as others metrics could do. This metrics is easy to interpret and robust to outliers. The only concerns with this metric is the small amount of insight it gives. This metric is similar to MSE.

Easy to interpret because if the value is 0.5 TeV, it means in average the predictions are off 0.5 TeV to the ground truth.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

5.3.3.2 Graphics

Graphics are pretty useful to display a lot of information and understand it quickly. Theses kind of elements helps to recognize patterns and also helps to compare different models. It helps to determine where problems could lie and also communicate more clearly information. Theses graphics will help to understand the energy regression better.

Energy Distribution

The energy distribution graph is a rather simple graphic. It's an histogram displaying the repartition off the ground truth events over an energy range. This graphic can display separately the different type of particles in order to get the difference between both. No predictions is implicated in this one.

The figure 5.10 shows the content of this graphic.

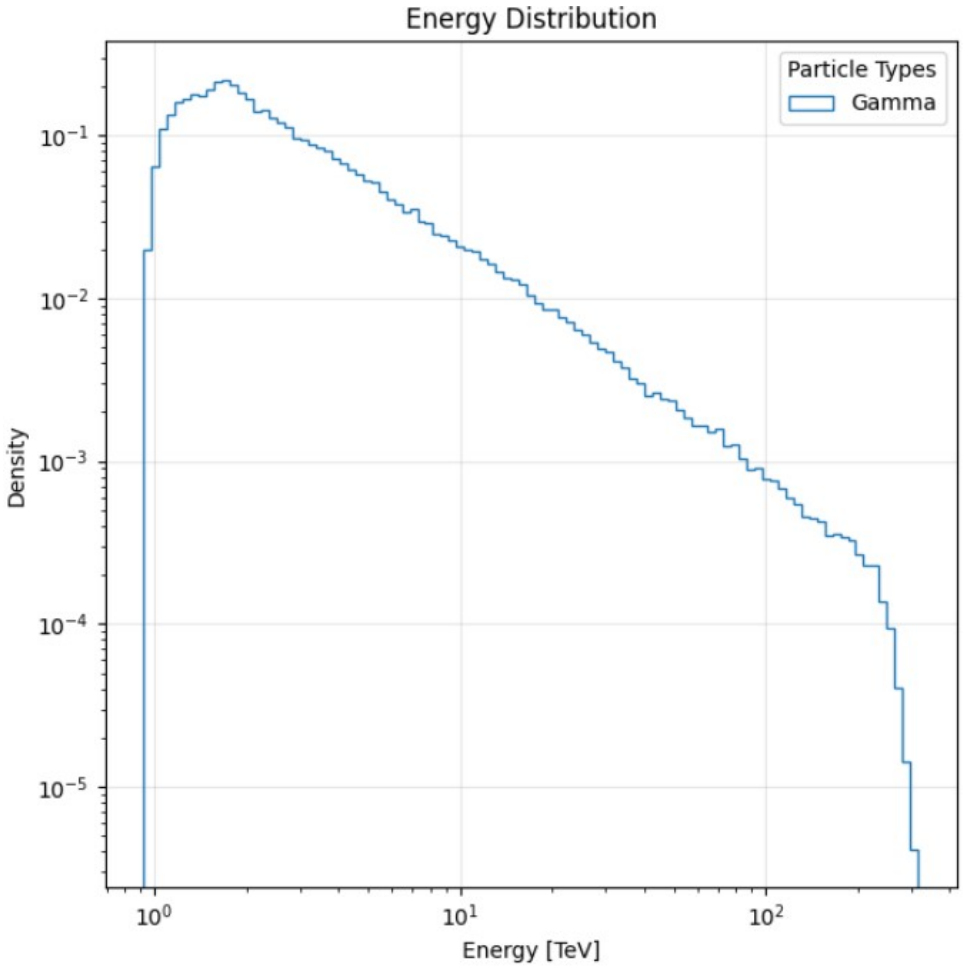


Figure 5.10: Distribution of events over energy ranges (bins). The histogram helps to get an idea of where events are concentrated. The x-axis shows the energy range for ground truth values with a log scaling and the y-axis displays the number of events corresponding to bins.

Migration Matrix

The migration matrix is also a graphic to evaluate energy regression models. It shows how the true energy is converted into reconstructed energy (with bins). It displays how ground truth values is reconstructed. It shows the distortion (divergence) the model does to the data.

This is used to show the reconstruction of the energy spectrum between real and predicted data. This is used generally used on gamma points data but can be extended to hadrons/protons. The events are distributed through bins (based on ground truth and predictions scale) in order to make it more readable.

The matrix generated should have its data concentrated around the diagonal describing the perfect resolution of the events. This can be visualized with the figure 5.11. This graphic is useful to see where the model is misinterpreting events at a wrong value. The warmer the matrix bin color is, the bigger is the amount of events in the corresponding bin.

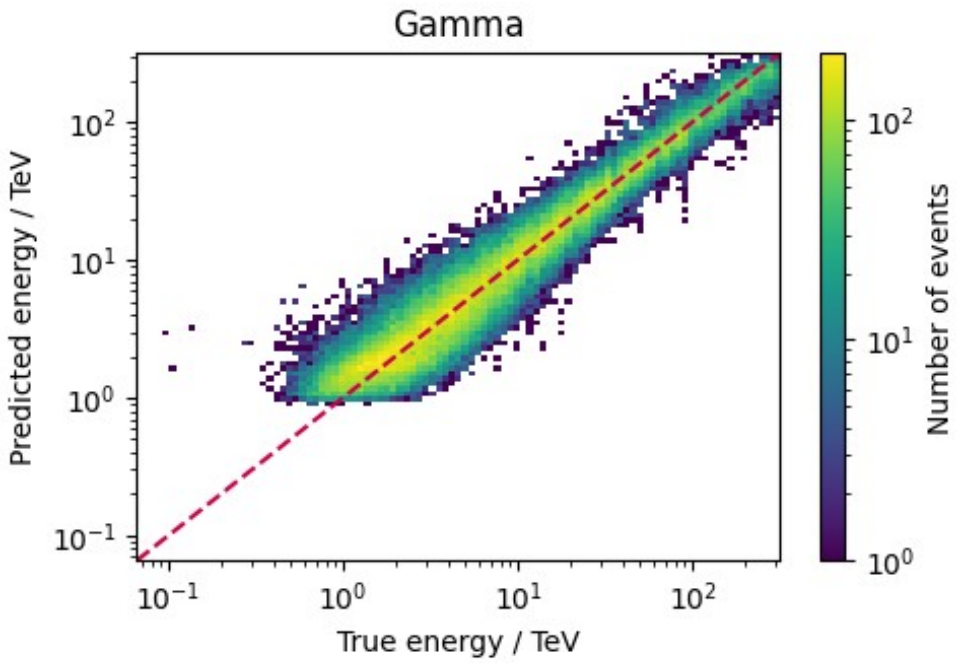


Figure 5.11: Migration matrix are useful to visualize an overview of events energy predictions. By grouping them in bins, the graphic becomes more readable and interpretable. The scales are in a logarithmic shape to comply to the range of prediction values possible. The axis, for ground truth and predictions, are displayed using a TeV unit (energy measure). A color scale has been added to get more insight about the place where most of the events falls. The ideal result should be to have events concentrated around the diagonal describing the perfect predictions.

Multiple information can be gain from this graphic. A large diagonal means an uncertainty in reconstruction of events when a narrow one means a good reconstruction. The bias can also be interpreted by looking at the position of the reconstructed diagonal compared to the perfect one. Being above means the model is overestimating and being below means it underestimates. Others small information can be gained from analyzing the graphic (at which level of energy it works better, patterns, ...).

Energy Resolution

The energy resolution is a graphic used to evaluate energy regression models. This graphic will display a curve evaluating how precisely a model can reconstruct the energy of gamma events. It shows the distribution of the relative error over the energy range in percentage. Like this, it's possible to know if a model is working better with high or low energy ranges. It helps to understand the model misbehavior on the energy range scope.

The calculation is done using the standard deviation of the gaussian of relative error between the reconstructed/predicted energy and the true energy. The lower the energy resolution is, the better is the spectral reconstruction.

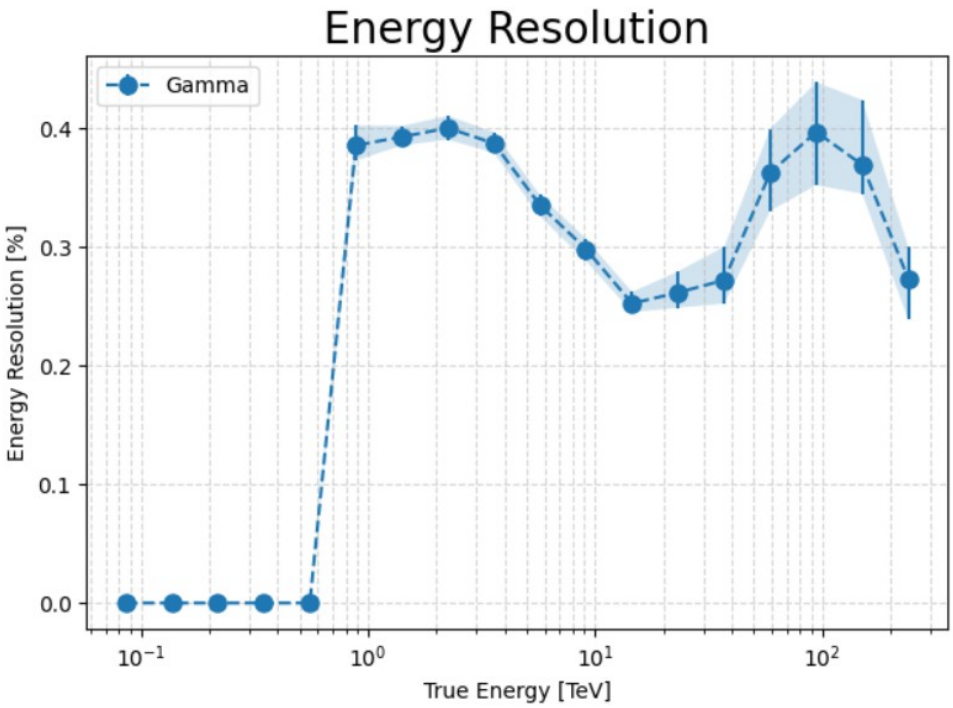


Figure 5.12: Energy resolution graphic computes how precise is the model for reconstructing the energy value depending on the energy scale (ground truth energy). The x-axis displays the real data in a logarithmic scale. For visualization purpose, the values are averaged in bins for defined ranges. The y-axis display the spread (energy resolution) in percentage of the predicted data. For each point on the graph, there is filling to show the upper and lower bound for bin (68% containment interval) of the events contained in the bins.

It helps to see where the model performs better. For lower energies, it might be difficult to have a good resolution as it can confuse with noise. It's the same for high energies where there are only few events.

Bias/Standard Deviation

This graphic displays two different curves : the bias and the standard deviation. It's normally computed as a unique metric value for the whole dataset but working with wide range of energy makes it interesting to split the metrics in a curve for defined range of energy (bins).

The bias is the metric calculating how far on average are the predictions of a model to the ground truth. It measures if the model overestimates/underestimates the energy. With bins, the value computed on the graph is the mean relative error as bias.

The standard deviation (or spread) measures how much residuals vary around the average error. It measures the precision of the model. It gives the statistical uncertainty of the model. It can be seen as a precision metric per energy bin.

The trade-off between the two are important to know if a model is accurate and precise. Normally by having a better bias, the standard deviation is worst and vice versa. Using bins, the observations can tell which range of energy is causing issues in the model. It can be seen in the figure 5.13.

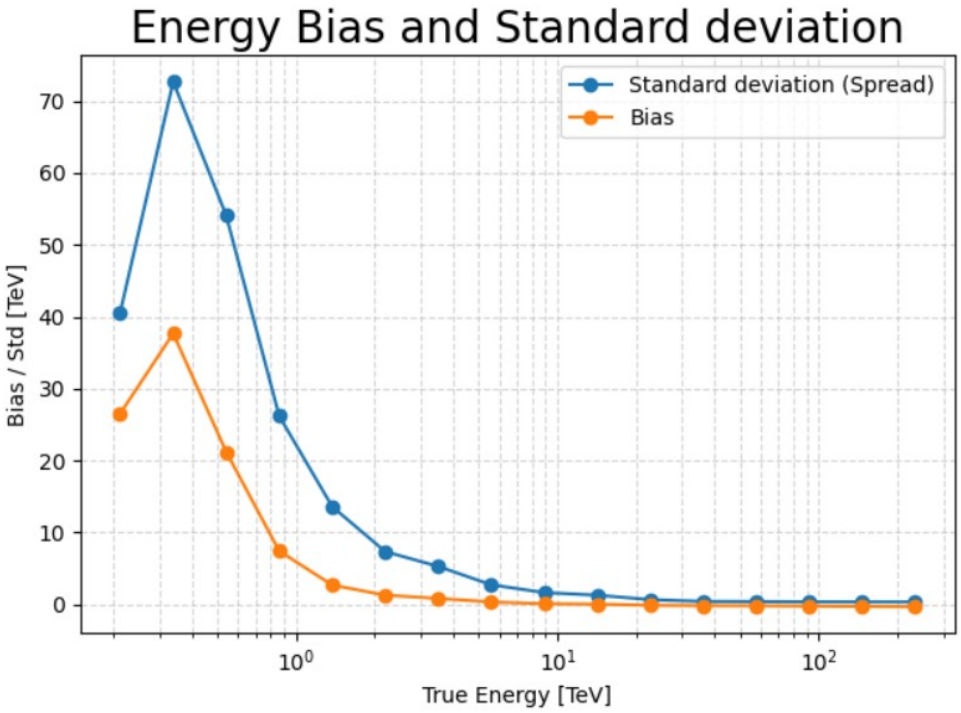


Figure 5.13: Bias and Standard deviation graphic. Two curves are displayed here : the bias and the standard deviation. There are displayed using bins average their ground truth values and reconstructed energy errors. The x-axis displays the bins (range) on a log scale and the y-axis displays the difference in TeV for the standard deviation and the bias.

5.3.4 Direction regression

The direction regression is one of the tasks handled during this thesis. For more details, refer to the chapter (4.1.3). This sections includes metrics and graphics especially oriented towards evaluating this specific task and get insights from the results obtained. This task is normally conducted after doing the classification task. Therefore, the evaluation is mostly conducted only on gammas values.

As a reminder, it's important to be aware that there isn't one but two values predicted as an output here : azimuth and altitude. Theses two predictions will be evaluated together or separately depending on the metric and graphic used.

The direction regression has two variants : Sky direction and Camera direction. It doesn't change the type of graphics used. Refer to chapter 4.1.3 for more details.

5.3.4.1 Evaluation Metrics

In order to compare and evaluate different models, it's required to look at how well they perform in their task. With this in mind, some metrics are defined to give a global overview of the results obtained based on multiple criteria. Theses metrics can show different problems encounters with the data and depending on the use case, have a different importance and priority to comply to. Metrics for regression can be various and complex due to the type of task. The measure used for every metrics is degree. Radians was also considered but it's less precise.

MDAE

MDAE (Mean Directional Absolute Error) is a metric measuring the average angular error between predicted events and the ground truth. In order to compute the directional error, azimuth and altitude needs to be combined and converted to 3D vectors. This helps to have a generic implementation of the metrics afterwards, working for different types of coordinates and preventing issues with angles, common when working with coordinates.

$$y_i^t \text{ and } \hat{y}_i^t = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\text{alt}) * \cos(\text{az}) \\ \cos(\text{alt}) * \sin(\text{az}) \\ \sin(\text{alt}) \end{bmatrix}$$

With the converted values, it's easier to calculate the metric as the interaction between the prediction vector and the ground truth vector is a simple dot product.

This metrics is easy to interpret and robust to outliers. The only concerns with this metric is the small amount of insight it gives. This metric is similar to MSE (5.3.3.1) and MAE 5.3.3.1). I's also rotation-invariant, meaning it doesn't depend on a coordinate system to work (using 3D vectors).

Easy to interpret because if the value is 0.5Âř, it means in average the predictions are off 0.5 degrees to the ground truth (in angular measurement).

$$\text{MDAE} = \frac{1}{N} \sum_{i=1}^{N} \arccos(y_i^t \cdot \hat{y}_i^t)$$

RMSDE

RMSDE (Root Mean Squared Directional Error) is a metric also relying on the conversion to 3D coords (5.3.4.1) as it implies working with both azimuth and altitude. This metric measure the average angular error but penalizes more large errors. It's similar to MSE (5.3.3.1) but for directional evaluation.

This metric is sensitive to outliers and aware of the distortion in dimensions with the conversion of 3D coords. It's used in combination with others metrics to see if large errors are made by the model.

$$\text{RMSDE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \arccos(y_i^t \cdot \hat{y}_i^t)^2}$$

MAE - Circular

This metric focus to evaluate one of the value predicted : the azimuth. The name includes circular because the range for the azimuth angle is 0-360 degrees. This is a Mean Absolute Error for this particular feature. The metric is similar in all point with MAE (5.3.3.1) with a twist to handle the circular aspect of the data. Distance between 1 degree and 359 degree should be 2 and not 358 as it should be computed by using only MAE. That's why a minimum selection is introduced to handle this aspect.

The metric will provide how off is the prediction of azimuth to the ground truth in degree.

$$\text{MAE Azimuth} = \frac{1}{N} \sum_{i=1}^{N} \min\left(|y_i - \hat{y}_i|,\ 360^\circ - |y_i - \hat{y}_i|\right)$$

MAE - Altitude

This metric is the same as the azimuth but for the altitude this time. Here, the handle of the circular aspect isn't necessary, which means that the metric is just like the classic MAE (5.3.3.1).

The metric will provide how off is the prediction of altitude to the ground truth in degree.

$$\text{MAE - Altitude} = \frac{1}{N}\sum_{i=1}^{N}(|y_i - \hat{y}_i|)$$

5.3.4.2 Graphics

Graphics are pretty useful to display a lot of information and understand it quickly. Theses kind of elements helps to recognize patterns and also helps to compare different models. It helps to determine where problems could lie and also communicate more clearly information. Theses graphics will help to understand the energy regression better.

Degree Performance Curve

This graphic is really simple but can give interesting insights about the range of precision of a model. The idea of the graphic was to see how imprecise the model can be to have an interesting accuracy. A curve is displayed to get a precision at different level of thresholds in degree. The accuracy is then display for each threshold. It can help to notice to which degree the model is precise.

The degree of error is defined by using the MDAE (5.3.4.1) metric where the difference in 3D coordinates between the predictions and the ground truth is used.

The figure 5.14 shows an example of curve that can be obtained.

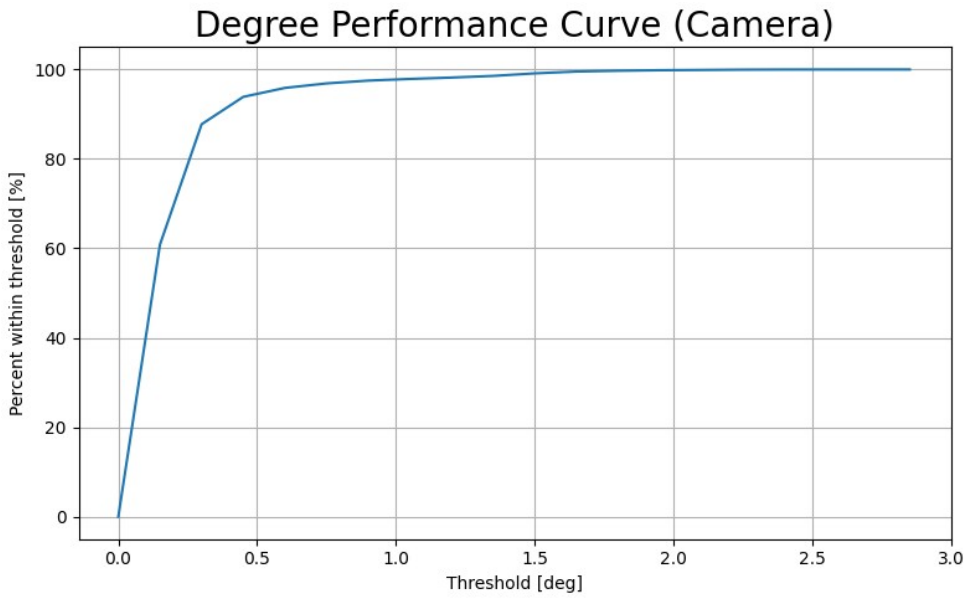


Figure 5.14: Curve showing the evolution of the accuracy with lower thresholds overtime. The x-axis displays the threshold defined arbitrary and the y-axis displays the accuracy at each threshold.

Altitude-Azimuth Distribution

This graphic uses visual code from the migration matrix to display the predictions for a direction regression model. It helps to visualize the density of the events in a heatmap using bins. The axis

corresponds to the two types of predicted coordinates : azimuth and altitude.

This graphic is helpful to identify patterns in the predictions of events. As the telescope is normally pointing in the same direction, a pattern should appear showing density of events around the pointing area of the telescope. Where the telescope is pointing is visualized using a cross. This indicates a reference of the area around which events should be detected. If it diverges from it, it could indicate a bias in the model.

The choice of graphic (figure 5.15) is done to concentrate around the area where the telescope is pointing. If no pointing reference was given, using a sky plot was also considered.

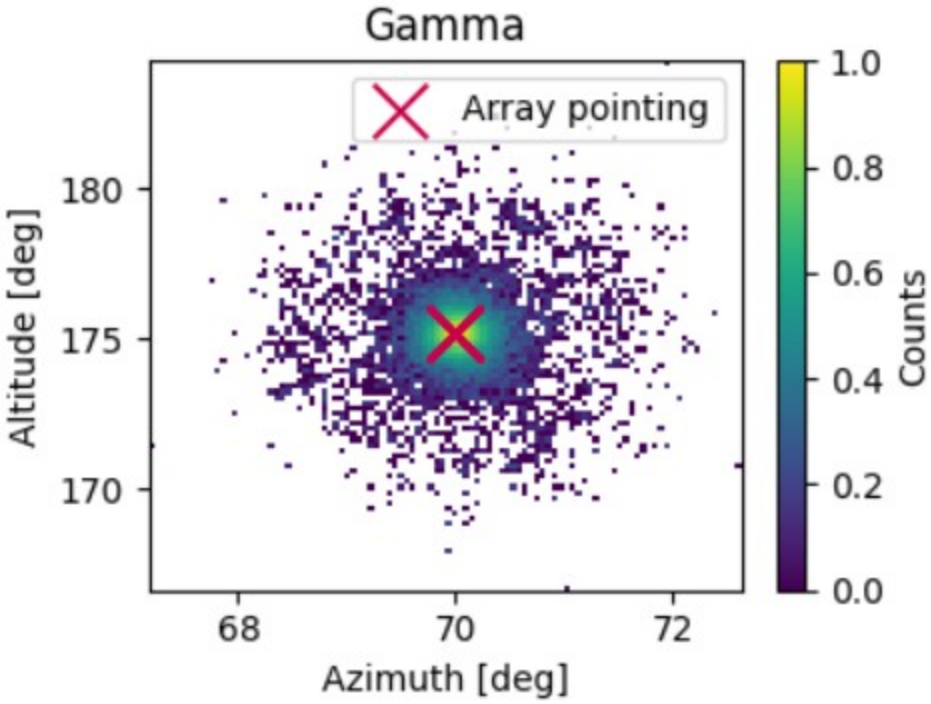


Figure 5.15: Distribution graphic showing repartition of the events in 2D coordinates system. The x-axis display the azimuth and the y-axis displays the altitude. The axis are in degrees and uses bins to compute the values. The density per bins is shown using a color bar. The cross represents the telescope pointing area and events should normally be detected around them. Depending on the use case, the events can be displayed in different graphic depending on their type.

Angular Resolution

This graphic is used to evaluate direction regression models. It works in a similar way as the energy resolution (5.3.3.2). It measures how precisely the direction is reconstructed for an event. The direction like others metrics is constructed by combining the two predicted coordinates : azimuth and altitude. The events are regrouped in bins corresponding to the ground truth energy. With this in mind, the angular resolution is calculated in degrees and displayed on the energy range scope. Like this, it's possible to know if a model is working better with high or low energy ranges, therefore understanding model misbehavior.

This graphic can be visualized with the figure 5.16.

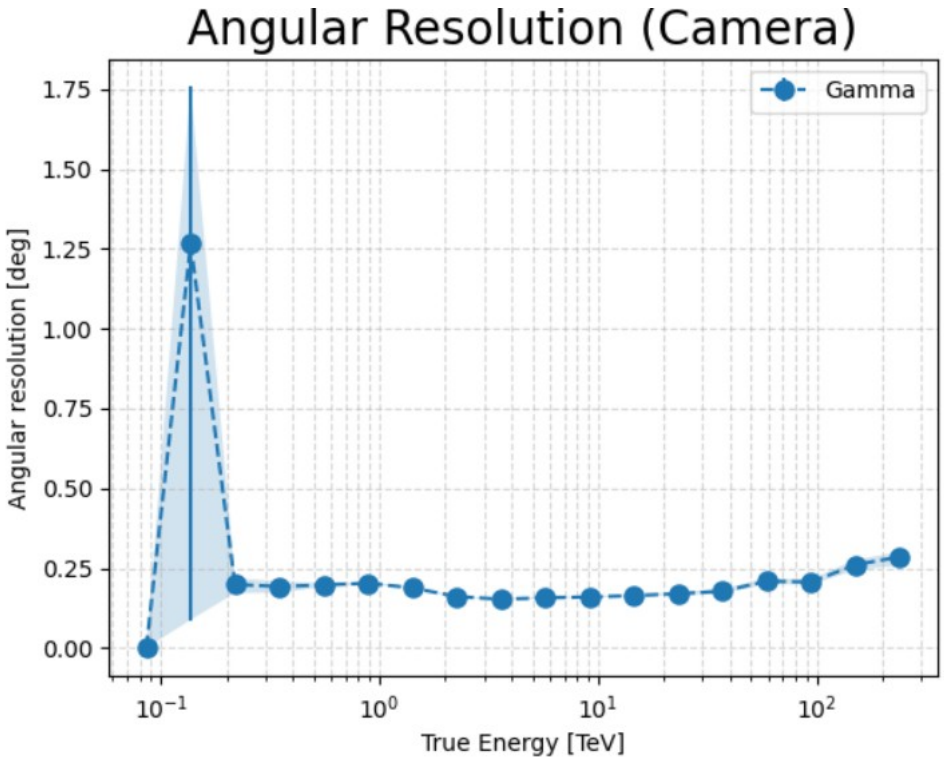


Figure 5.16: Angular resolution graphic computes how precise is the model for reconstructing the angular (degree) value depending on the energy scale (ground truth energy). The x-axis displays the ground truth energy of events in a logarithmic scale. For visualization purpose, the values are averaged in bins for defined ranges. The y-axis display the spread (angular resolution) in degree of the predicted data. For each point on the graph, there is filling to show the upper and lower bound for bin (68% containment interval) of the events contained in the bins.

Angular Bias/Standard Deviation

This graphic is working in the exact same way as the one for the energy (5.3.3.2). Refer to it for more details. The graphic displays the standard deviation and the bias per bins for both predicted coordinates : azimuth and altitude. It helps to identify if one of the coordinates is causing more problem than the other one. The values are displayed in degrees.

The figure 5.17 shows the different curves for this graphic.

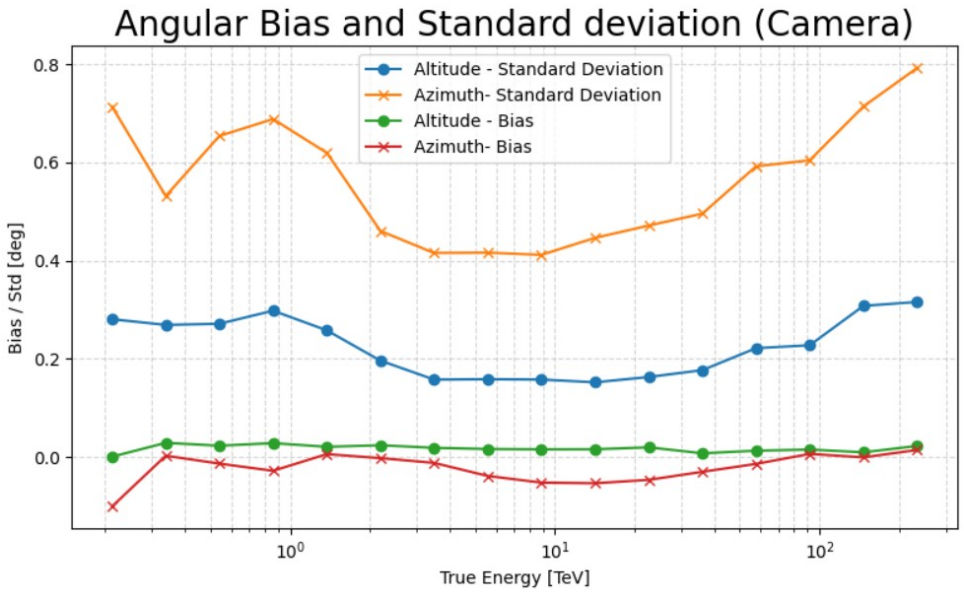


Figure 5.17: Bias and Standard deviation graphic for the two coordinates predictions. Each of them displays the bias and the standard deviation corresponding to them. The difference between the different curve are in the point type and the colors used. There are displayed using bins for the ground truth energy of events. The x-axis displays the bins (range) on a log scale and the y-axis displays the difference in degrees for the standard deviation and the bias.

5.4 Comparison of models

The other tool implement as part of the thesis is the comparison of models report. It's similar to the report generation as it's also a report and it's uses almost all the graphics and metrics detailed in the chapter dedicated to it (**??**). There are differences for graphics and metrics displays as the comparison of the models provided multiple values for same graphics and metrics. Some rearrangements needs to made in order to have a proper comparison of models. Theses changes are explained below.

Comparison of the models is an extended control system for the report generation. Instead on evaluating performance of one model, it's possible to compare multiple models at the same time in this scenario. It enables to make decisions and identify the best candidates for a determined task. It's also enabling objective evaluation as a report on only one model is subjective to determine if the model is better or not. It's also useful as the model could be used in different scenarios/environment and depending on that, a model could be better than another and vice versa (e.g. requirements for an triggering system isn't the same as model for a server usage). This tool is rather useful for a decision-making system.

The comparison of models occurs only on models for a same task. It doesn't display graphics for models of different tasks. The report generation is dedicated to a model and the comparison of models is the same but for a specific task and related models.

Some general information are available without comparing the model on the comparison report. It includes the task for which model are compared, the number of training and testing events and the number of models involved in the comparison. A dedicated chapter isn't necessary as theses values explain themselves quite simply.

5.4.1 Metrics modifications

Concerning the way to handle metrics, there are a lot of changes from the report implementation. In the model report, there were two sections dedicated to metrics : Model and Runtime metrics and Performance metrics. The same two sections are present in the comparison fo the model with the associated metrics. To compare metrics, two ways have been implemented : A brute comparison by metrics and a ranking system for the metrics. The combination of the two approaches allows to combine the precision and quick understanding of the metrics and which models is better depending on the wanted use case.

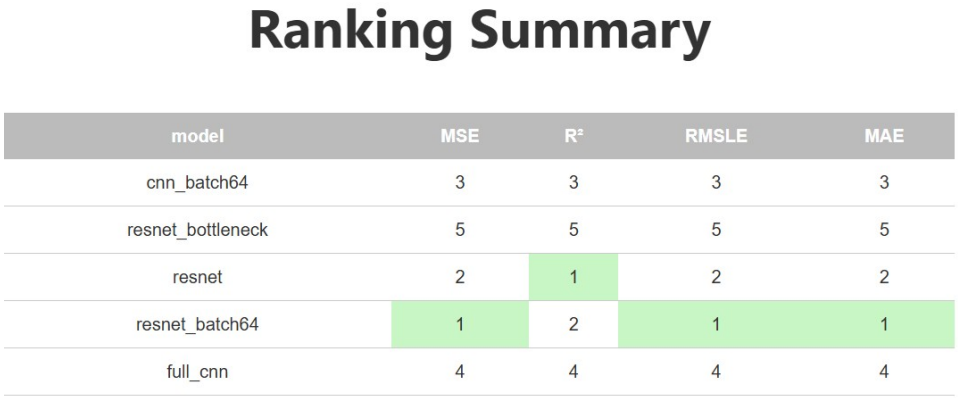
The brute comparison delivers detailed metrics values. It helps to compare the level of differences in metric results between models. The figure **??** shows a table with metrics comparison for each model. A row is associated to a model as a column to a metric. This display have a quick overview and easier comparison of models metrics.

Performance Summary

model	MSE	R^2	RMSLE	MAE
cnn_batch64	685.245	0.311	26.177	8.571
resnet_bottleneck	991.712	-0.689	31.491	10.678
resnet	635.286	0.447	25.205	7.705
resnet_batch64	613.277	0.430	24.764	7.508
full_cnn	751.043	0.069	27.405	9.350

Figure 5.18: Example of table for Performance metrics. It contains a first column stating the model name as a reference. Then the metrics are provided with their values. This table is for deeper comparison of a model to see if model outperforms in a specific metric.

The ranking comparison, on the other hand, gives an idea of performances against others models depending on metrics. It takes the same table system as the brute comparison and is very helpful to quickly identify the model with the best performances when lots of models are compared. To further accelerate understanding of the table, a green color background is applied to the model with the best value for a determined metric. This can be seen through figure **??**.



Ranking Summary

model	MSE	R^2	RMSLE	MAE
cnn_batch64	3	3	3	3
resnet_bottleneck	5	5	5	5
resnet	2	1	2	2
resnet_batch64	1	2	1	1
full_cnn	4	4	4	4

Figure 5.19: Example of table for Performance metrics. This one is dedicated to a ranking system. It rather shows ranking compared to others models for each metric instead of their value. It helps to quickly understand which model has better performance.

5.4.2 Graphics modifications

As said earlier, the comparison of the models reuses graphics from the report generation. Some of the graphics, depending on the task aren't really relevant for a comparison and are, therefore, not used for this tool. Displaying graphics for this tool is modifying the ones from the report generation to provide a clean comparison of the models. It combines the graphics from each model into one for comparison purposes.

There are two different ways to combine theses graphics. The first one, displayed by the figure 5.20, combines values inside a unique graphic values from the models. This is normally dedicated to line plot or similar plots when values from different models don't overlap on each other and are still legible while being on the same graph. The scale used for each model output is then the same, which isn't always the case for the second approach. The second one, displayed by the figure 5.21, provides sub-graphics for each model inside a same model. The graphics are directly comparable and the same as generated in their respective report. Matrix is a good example of graphics requiring this approach.

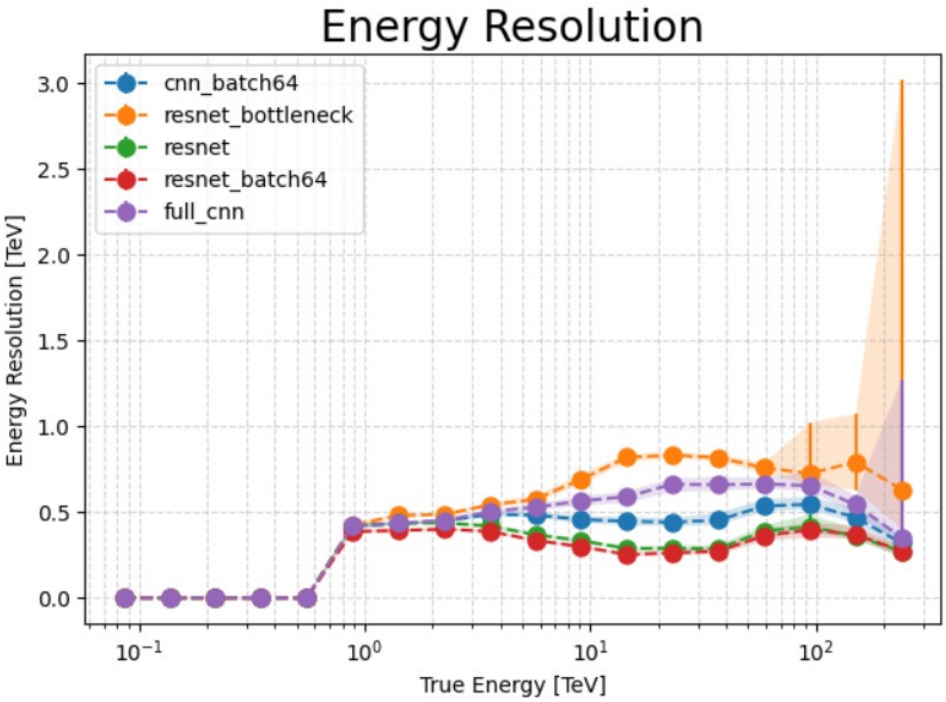


Figure 5.20: Example of a combined graphic in the comparison of models (Energy Resolution). A single graphic is displayed with, in this case, lines for each model determining the energy resolution over bins of energy. The same scale is used for every line and comparison can be easily done as elements are next to each other. This approach is used for simple displays of graphics.

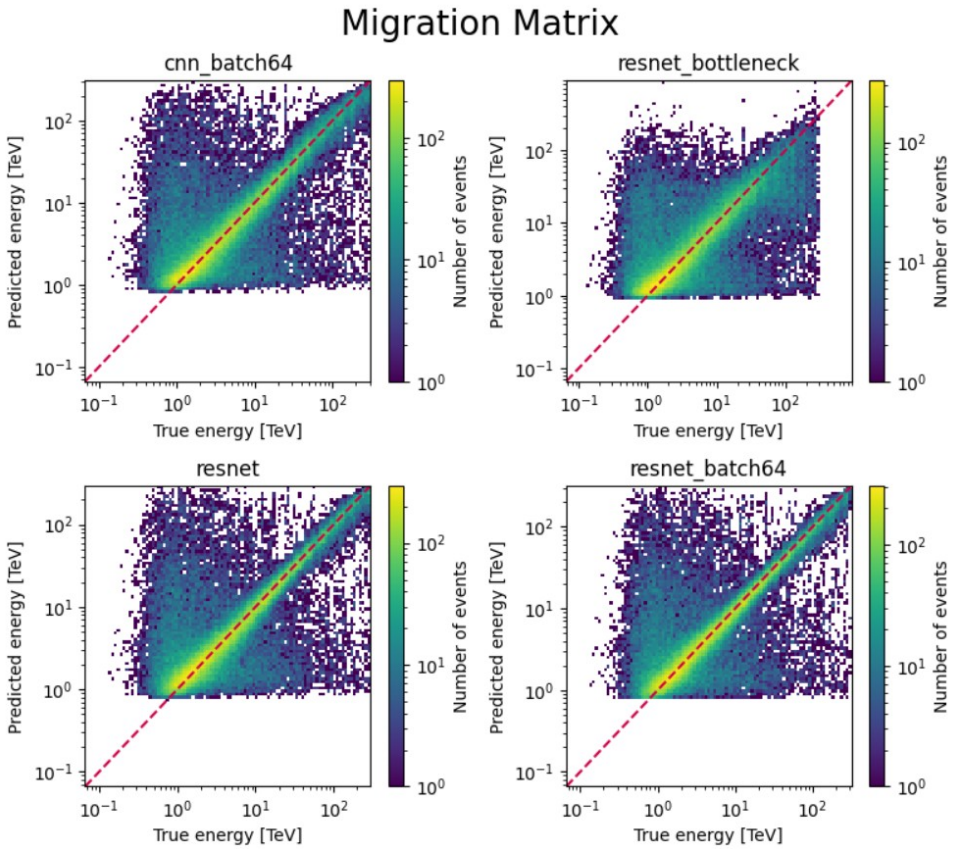


Figure 5.21: Example of a subgraph comparison (Migration Matrix). Graphics with complex features aren't combined but rather put next to each other to compare as easily as possible. The plot is exactly the same as in the model report for each of the models. The scale might vary depending on the subgraph displayed.

To give an idea about the graphics used for each task, the list below details and give a link about all the graphics present in the generated file. This list precises also if the graphic are merged together like the figure 5.20 or separated using subplots as the figure 5.21. An example of each comparison of task is also available in the appendix.

- **Particle classification**
  - **Confusion Matrix** (5.3.2.2) - Separated graphs.
  - **ROC Curve** (5.3.2.2) - Combined graph.
  - **Calibration Curve** (5.3.2.2) - Combined graph.
  - **Gammaness Distribution** (5.3.2.2) - Separated graphs
- **Energy regression**
  - **Energy Distribution** (5.3.3.2) - Separated graphs.
  - **Migration Matrix** (5.3.3.2) - Separated graph.
  - **Energy Resolution** (5.3.3.2) - Combined graph.
  - **Energy Bias and Standard Deviation** (5.3.3.2) - Separated graphs

- **Direction regression**
  - **Precision in degrees** (5.3.4.2) - Combined graph.
  - **Angular Distribution** (5.3.4.2) - Separated graph.
  - **Angular Resolution** (5.3.4.2) - Combined graph.
  - **Angular Bias and Standard Deviation** (5.3.4.2) - Separated graphs

6 Models

CTLearn needs to analyze telescope images and find key features to understand events and objects from the space. The library thought Deep Learning models were a interesting solution to explore. The library offers multiple implementation of models to train and predict the key features wanted depending on the task.

From this, it implies that models are a big fragment of the CTLearn library and therefore of the thesis. Models built here aim to be the best for their respective tasks. Looking at research papers, some references results can be used as a starting point for models created during the thesis to have as a beatable objective.

Models and related subjects such as optimization of them with deployment of FPGA or the code related to the generation of models are examples of what this section contains. The thesis is tied to the CTLearn library and some modifications are expected on the library. Some sections describes modifications and interactions with the library.

Models used in the thesis are also described with precision, including the parameters than can be modified, the default architecture or the main concept related to the model.

6.1 Reference

All models trained and tested during this thesis has one goal : Beat the reference model displayed in research papers, or at least get as close as possible to its performances on every task. This isn't the main goal obviously but producing a similar result could be a good indicator that the approach of the thesis is a good one to produce results.

The reference model varies from a paper to another, therefore choosing the model to beat and the performance metrics is important. Usually, models used as references for the papers uses less data than the models of this thesis implying some interesting results could come out of this. Some parameters needs also to be taken in account for the reference such as the batch size or the number of epochs to get a better idea of the reference.

The first reference is the CNN-RNN model from [41], combining CNN layers and RNN layers together. This model takes multiple images from a same imaged shower as input to determine the particle type of the event. This model is conducted only for a **particle classification** task. Experiment were conducted on multiple telescopes but the interest is for SST-1M telescope data here (same as the thesis data). Its works with energy from 20 GeV to more than 300 TeV and has a balanced dataset between types of particles. The model obtained a global accuracy of **0.809 %** and an AUC of **0.893 %** on the test set. More details about this metrics are displayed in the figure

6.1. The model was trained using 40000 batches of 16 events, less than models from the thesis.

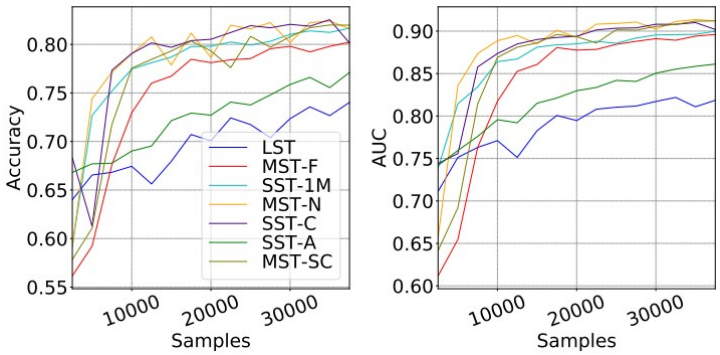


Figure 6.1: Accuracy and AUC metrics over the number of samples for each telescope. The main focus is on the SST-1M, the telescope from where the data used in this thesis are retrieved. The accuracy for this telescope is converging a bit upper than 0.8 and the AUC is converging almost at 0.9.

6.2 CTLearn Models

CTLearn library contains lots of features. Among them, there are pre-built models directly ready to be used by the library to train and test a model. Theses models are useful because the library adapted the models to be used with the library requirements and constraints. Theses pre-built models are modifiable by providing a configuration file with architecture features wanted like the number of layers or the usage of attention layers for example.

An additional type of model is available, called the LoadedModel, allowing to load custom models while respecting library restrictions like the name of layers, the separation between the head and the backbone of the model or others constraints like this. A custom model template has been created to avoid incompatible models and is documented in a dedicated chapter.

6.2.1 ResNet

The ResNet model available on the library can be seen as the reference model with the best results for every task of the thesis. It's also a complex model with a consequent number of parameters and layers. The computational resources required by this model is higher than others.

The ResNet (Residual Network) [42] [43] model is an architecture solving challenges very deep neural networks face when working with very deep networks. Normally, the deeper the model is, the harder it is to train the model, vanishing and exploding gradients are more frequent and performance can decrease. The idea is that the model learn residual mappings to train more effectively by using a skipping connection process, allowing information to flow directly across layers. This means that the residual mapping is combined to the convolutional layers output to have better results and keeping initial information. It allows better results and less issues when working with deeper models.

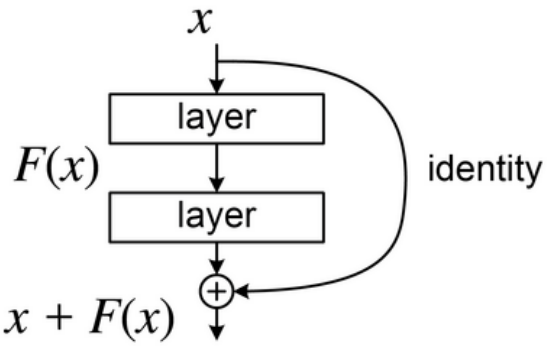


Figure 6.2: Main concept of the residual block. The idea is that the input is propagated with an identity mapping to skip some layers. The input is then combined to the output of the convolution layers through an Add element-wise operation. It helps to prevent vanishing/exploding gradients and limit the degradation of performance in very deep networks. This type of block is generally combined to other blocks to form a full network. [42]

The default configuration of the ResNet model is used a complex architecture with multiple blocks for a complex architecture.

The residuals blocks defines in the CTLearn library relies in two format : a bottleneck one and a basic one. A parameter is defined to switch from one to another. The difference between them relies on the number of filters and layers used but the fundamental concept is the same for both of them. The residual block makes models non-sequentials and complex. The figure 6.3 displays the bottleneck residual block and gives details about it.

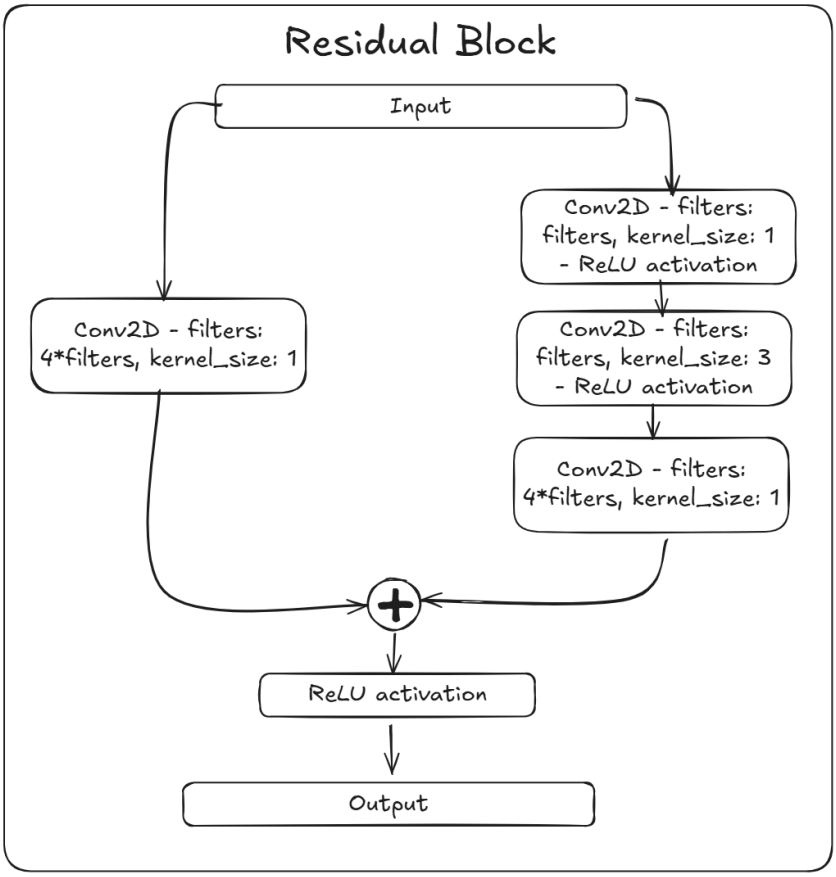


Figure 6.3: Default Residual block in CTLearn library. The residual block helps to keep computation efficiency and stability for deeper network. This is the default residual block used to build a ResNet model with the CTLearn library. It's a configurable block that takes advantage of the residual features by passing directly the input through a identity mapping/projection shortcut. The shortcut is represented with this identity map is combined through an addition with the normal convolutional pathway. A ReLU activation is then applied to stabilize the network. The output of the activation function is then set to the next block/layer, finishing the residual block step. Some elements are configurable for the residual block. First, there is the number of filters that can be changed depending on the block. A attention layer can also be added at the end of the convolutional pathway and before the add operation.

The central element of the ResNet network from CTLearn has been described. To have a full deep network, the model is stacking multiple residuals blocks. The figure 6.4 represents the architecture of the default model of the CTLearn library. It includes layers, representing stacked blocks with a defined number of filter. The residual blocks are only in the backbone part as the head is still some Dense layer to converge to the output. The number of stacked blocks and filters can be changed in the configuration file. This is represented by the same parameter as the CNN model, **architecture**.

```
CTLearnModel:
    architecture: [
        {"filters": 48, "blocks": 2},
        {"filters": 96, "blocks": 3},
        {"filters": 128, "blocks": 3},
        {"filters": 256, "blocks": 3},
    ]
```

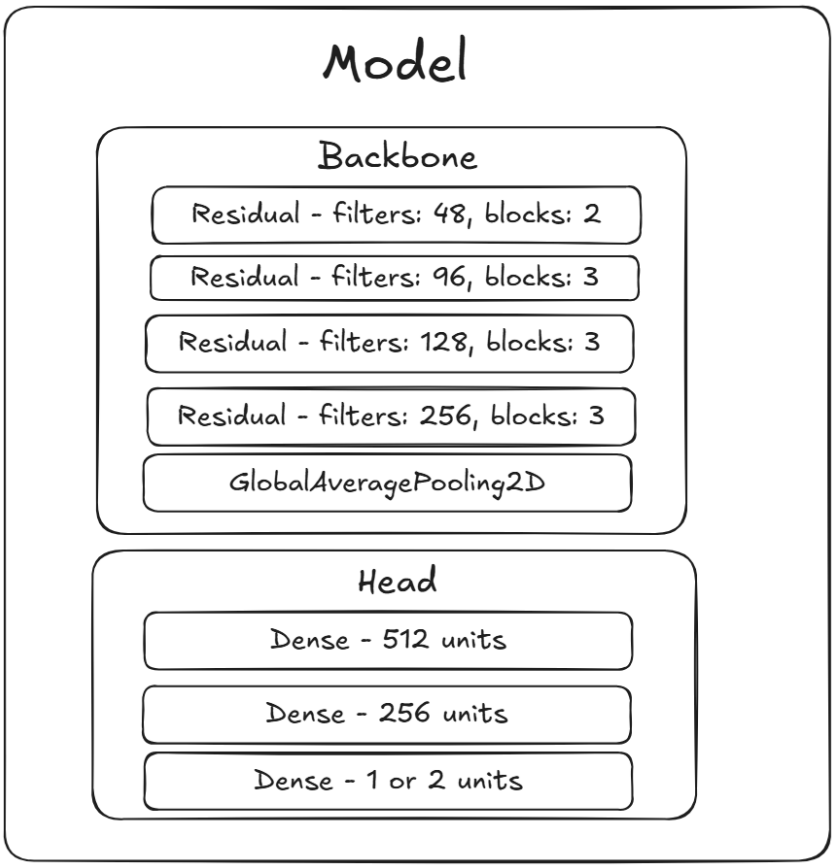


Figure 6.4: Default ResNet Architecture on CTLearn library. The architecture combined residual blocks from the figure 6.3. The idea is to build a very deep model by chaining series of residuals blocks. It allows to train a very precise model without too many parameters and computation time that a model without residuals connections would have. The head part is a classic combination of Dense layers that can be defined separately from the blocks.

With this, understanding the ResNet model and its behavior should be clear. Having a high complexity model is interesting to check if all the pipeline and calculations of metrics works correctly. It avoids facing theses issues later when having complex customs models. This model has been the reference when building tools or testing metrics along the CTLearn library. In some scenarios, the complexity of the model revealed some errors that wasn't showing up with simple models.

6.2.2 CNN

The CNN model [44] [45] available on the library can be seen as the simplest model possible. It contains a few layers, really simple just to build a model and test the whole pipeline of training and testing a model.

A CNN (Convolution neural network) model is a type of Deep Learning architecture used for computer vision and image processing. The layers of the model are 2 dimensional grids to match matrix-like data (e.g. spatial or temporal data). This allows to extract complex features and patterns. This architecture relies generally on multiple type of layers to work. Convolutional layers are the main CNN feature with its 2-dimensional filters able to extract patterns. MaxPooling and others along classic Dense layers can be used for this type of architecture.

The figure 6.5 displays an example of architecture for a CNN.

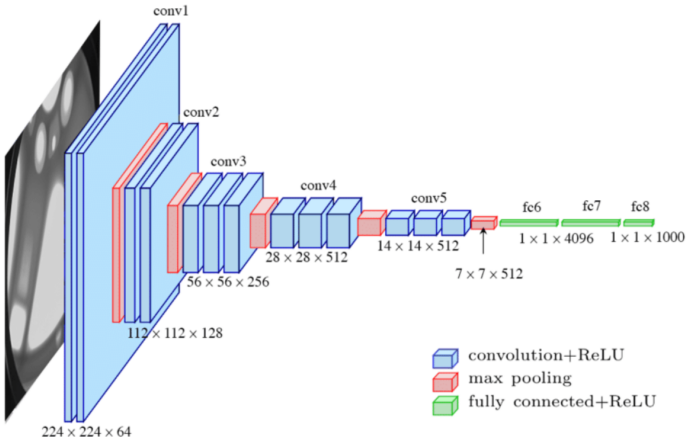


Figure 6.5: Example of VGG16-CNN Network with multiple convolutional layers and max pooling layers. Along the network, the size of the matrix is reduced to have a single output at the end with the prediction. This type of architecture manages to extract a lot of information from an image, from high-level features/patterns to detailed low-level features. [45]

Concerning the implementation on CTLearn library, the CNN model can have its backbone (and head) modified with the configuration file to adapt to the use case. The only thing is that the type of layer is still the same. It's always Conv2D layers for the backbone and Dense layers for the head.

To change the default backbone layers, the parameter **architecture** of CTLearnModel instance need to be define like the example below. Some others parameters like the pooling type or the normalisation can be changed. Refer to the CTLearn library for more details.

```
CTLearnModel:
    architecture: [
        {"filters": 32, "kernel_size": 3, "number": 1},
        {"filters": 32, "kernel_size": 3, "number": 1},
        {"filters": 64, "kernel_size": 3, "number": 1},
        {"filters": 128, "kernel_size": 3, "number": 1},
    ]
```

As said earlier, the head can also be modified. This time the parameter to use is **head_layers** (be aware to respect output dimensions). This parameters defines the number and size of head Dense layers. The task needs to be precised.

```
CTLearnModel:
    head_layers: {"type": [512, 256, 2]}
```

The figure 6.6 displays the default model used during the thesis with details about the implementation. The figure doesn't show all the details about activation functions or MaxPooling layers but the caption of the figure explains clearly theses details.

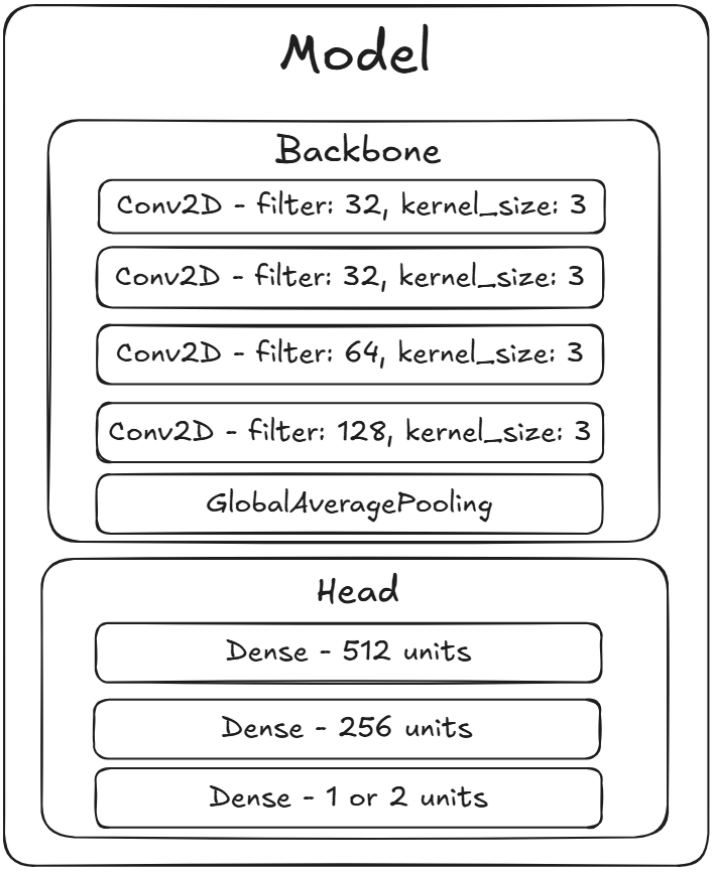


Figure 6.6: Default CNN implementation for CTLearn library. The model is separated in two parts : the backbone and the head. The backbone contains 4 convolutional layers with important parameters mentioned. Each of theses layers are followed by a MaxPool layer to reduce the dimensions of the matrix. The activation function for each layer is the ReLU activation function [46] and the padding is defined so that the Conv2D layer don't reduce the dimension of the matrix (only MaxPool). The backbone concludes with a GlobalAveragePooling layer, avering the spatial dimensions into one (height, width, channel into one dimension) just like flatten would do. The head has 3 layers that takes data in one dimension with reducing amount of units over the layers to converge to 1 or 2 outputs, depending on the task (e.g. 1 for energy regression and two for particle classification). For the classification, a SoftMax layer is added to finalize the model. For the head, the same activation function as the backbone is used.

6.2.3 Custom model

The custom model is the third way to use a model with the CTLearn library. On the contrary with the other two type of models, this isn't a prebuilt model but rather a way to provide a custom model to train. This approach has a lot of benefit like the liberty of choosing a type of models or even introducing the aspect of finetuning by providing pretrained models through this type of model.

For uniformity purpose when doing the training task, the training task is always provided model through the LoadedModel implementation. The model depending on the fact it's a custom one or not is built in another task to prepare it once for all. Refer to this chapter for more details (6.3.1).

Template

The template to build a custom model is rather simple. The model is built in way that the only elements that needs to be modified to make it work are the layers. The template is attached in the appendix (10).

There is a global class containing the whole model and storing the initial parameters like the input shape or the type of task. With the attributes initialized in the class, the model is built by separating in two distinct section : the backbone and the head. Theses elements are built separately in distinct classes. The network structure for the template is displayed in the figure 6.7.

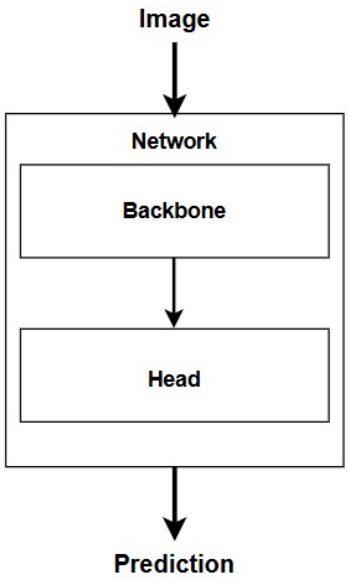


Figure 6.7: Network structure of the template for custom model. The network is separated in two parts : the head and the backbone. The Backbone handels high-level features and start processing the input when the head handles low-level features and computes the predictions of the model. Each of theses elements have a distinct class to define the model layers.

The choice of this kind of structure for the network is due to parameters in the CTLearn library giving the possibility to remove the head of the model and provide the default head provided by CTLearn on the data. The idea is then to keep this separation of head and backbone between the layers to be more flexible possible to work with the package. It's also a useful way to separate fine-tuning/transfer learning parts from unmodified weights. The reusability of this is also very high with the possibility to attach different head part to the backbone to see potential differences in the results.

```
class ComplexModel(Model):
    def __init__(self, input_shape=(96, 96, 2), reco_task="type", num_classes=
                                            10, dropout_rate=0.2, name="
                                            complex_test_block"):
        super().__init__(name=name)

        # Initialize model parameters
        self.input_dim = input_shape
        self.task = reco_task
        self.num_classes = num_classes
        self.model_name = name
```

```
        self.dropout_rate=dropout_rate

        # Build backbone and head
        self.backbone = self.build_backbone()
        self.head = self.build_head(name=self.task)

        # Prepare and build global model for graphic usage
        inp = Input(shape=self.input_dim, name="input")
        x = self.backbone(inp)
        out = self.head(x)
        self._graph_model = Model(inp, out, name=f"{name}_functional")

    # Build by chaining different layers
    def call(self, inputs, training=False):
        x = self.backbone(inputs, training=training)
        out = self.head(x, training=training)
        return out
```

As a reminder, the backbone is the core network of the model containing the high-level features. It takes most of the resources and can be heavy. Most of the parameters are included in this part and this is one of the reasons why when fine-tuning, usually the backbone is frozen. It's the starting point of the network where the data is provided.

```
# Backbone model (always used) and can be only element called if overwrite of
                                        head is wanted
# Parameters can be changed or added (name MUST end with "_block")
class ModelBackbone(Model):
    def __init__(self, dropout_rate=0.2, name="backbone_block"):
        super().__init__(name=name)

        # Define layers


        # Need to flatten before head
    def call(self, inputs, training=False):

        # Build the model by chaining different layers
        return ...
```

The head, on the contrary, can be seen as the end of the network where the predictions is computed with lighter and simpler layers. This is generally the part fine-tuned of the model.

Both of them are constructed by the network by wrapping their respective parts in functional model (or nested model) so it can be treated as a single block.

```
    # Build backbone as functional layer (combine backbone layers as one for
                                            tensorflow recognition)
    def build_backbone(self, name="backbone_block"):
        inp = Input(shape=self.input_dim)
        backbone = ModelBackbone(dropout_rate=self.dropout_rate, name=name)
        out = backbone(inp)
        return Model(inputs=inp, outputs=out, name=name)
```

This way to proceed gives a clear abstraction and enhances the reusability of global and nested models. By proceeding this way, there is a need to override some functions from Keras Model class

for compatibility purposes with the architecture in place.

To comply with the CTLearn library, some restrictions must be respected. For example, there can only be one input passed to the model to work. There are also naming constraints. CTLearn library searchs for specific layers by looking at there names like for the backbone as the last layer must finish with "_block" to work. Same thing for the head where a correct task needs to be provided.

6.3 Tasks

As discussed in the chapter dedicated to the MLOps approach (5), the pipeline to produce a model is decomposed in multiple tasks. Some of them are completely related to the MLOps approach, however the generic task related to the generation of the model are closely tied to the modeling part. Theses tasks are the ones directly interacting with the CTLearn library and are dependent of it. They have a direct influence on the generation of a model and related results. Shorts explanations about theses tasks are therefore required. To visualize all the tasks related to the model generation for this thesis, refer to figure 5.2.

6.3.1 Prepare model

The preparation of the model is an essential task to have a uniform training of the model. It's included in the training of the model. The idea is to build a model that complies with the requirements of the CTLearn library. This is the first step of the creation of the model. The model can be loading a default configuration from CTLearn (ResNet or CNN) or load a custom model created using the template provided earlier (6.2.3). For now, pretrained models cannot be passed in the preparation step but with small changes it could be possible.

The preparation is separated in two cases : the usage of prebuilt CTLearn model or the usage of Custom models. In case of custom models, the file and the class name containing the model class need to be provided as an additional element. If prebuilt CTLearn models are used, a CTLearnModel instance is created with the model contained in it. Both of the scenarios saves the prepared model in a temp folder in order to be retrieve by the training task.

Some essentials parameters needs to be provided when preparing the models. Parameters displayed below are mandatory for this step. Additional elements like the attention mechanism for ResNet models aren't shown below. Theses additional elements are used to modify and adapt prebuilt models to certain situations.

- **Image shape** : Shape of images provided to the model and used as input layer
- **Type of model** : LoadedModel for custom models or ResNet/CNN for prebuilt models
- **Task** : Task for which the model is built. Depending on the task, head layers and compiler can change.
- **Number of classes** : Related to the task. Corresponds to the number of classes as output (1 when not classification)
- **Temporary directory** : Place to store the prepared model.

Like this, the model is ready to use for training with the CTLearn library. The model is also usable without the library but some modifications shall be done on the data processing to be usable by the

model.

6.3.2 Training

The core task of the model generation is the training of the model. This task comes right after the preparation of the model. This previous step can be skipped if a model is already prepared and respects constraints from CTLearn library. This step has been made with the idea to have the fewer changes possible when training different models. The differences are managed by the preparation step and the configuration file. It enhances reproducibility and automation of model generation.

The training step retrieves the temporary model stored previously and provides it along the configuration file to an internal CTLearn process, **TrainCTLearnModel**, that creates the training process with configuration and corresponding model. To train the model, the only step is to run this process.

The training, just like others tasks requires parameters to work correctly. Theses parameters are editable in the configuration file dedicated to the experiment. Refer to the corresponding chapter (5.2) for more details. All the parameters are passed through a Config instance from traitlets library [47]. The training tool from CTLearn has a lot of different configurations possible, so the focus has been done on some specifications while others are kept as default. Some parameters in the configuration file will not be described as they should not be touched.

- **Input data (signal)** : Data provided to the model containing gamma-ray events (Comes with a pattern parameter to filter the files).
- **Input data (background)** : Data provided to the model containing hadrons events (Comes with a pattern parameter to filter the files).
- **Task** : Task for which the model is built. Depending on the task, CTLearn library behaves differently.
- **Number of epochs** : Number of epochs to train the model
- **Batch Size** : Size of batches for training.
- **Output directory** : Directory where the model will be stored.
- **Percentage per epoch** : Percentage of the data used for each epoch (if not 100%, selected randomly)
- **Early Stopping** : Configuration to stop training earlier if changes from one epoch to another aren't enough.
- **Pruning of the model** : Parameters to prune the model
- **Mono/Stereo mode** : Whether to handle one telescope image or several telescope image at once.

For the training task (also testing task), modifications on the library are required as the process of pull requests and new package releases are not compatible with the limited time available for the thesis. Therefore, modifications of library has been done on specific files and redirection of packages has been done. Theses modified files are contained in a dedicated space for tools.

Concerning the training task, the train_model.py file has been modified to comply with the thesis usage of models. Among the modifications, two of the listed parameters were added. The percentage per epoch and the pruning option have been added. The pruning option is documented in a dedicated chapter for model optimization (6.4.1).

The percentage per epoch on the other hand is a feature added to solve at first the issue with the time to generate a model. The idea is that, after separating the data into batches, a random portion of the batches, according to a given percentage, is selected for the current epoch training. The next epoch will take a different random portion of batches. This allows to reduce the amount of data read during an epoch. This approach comes with another feature, the generalization of the model. That can be either an advantage or disadvantage depending on the percentage defined. By training with different data each epochs, the model won't be overtrained as it doesn't see the same data every time. However, if the percentage is too low, the model could possibly not see some data during training. This approach has a downside. Batches selected in the earliest epochs will define the model shape and have greater importance than latter epochs data as the learning rate is decreasing over the epochs. This concept is visualized using figure 6.8.

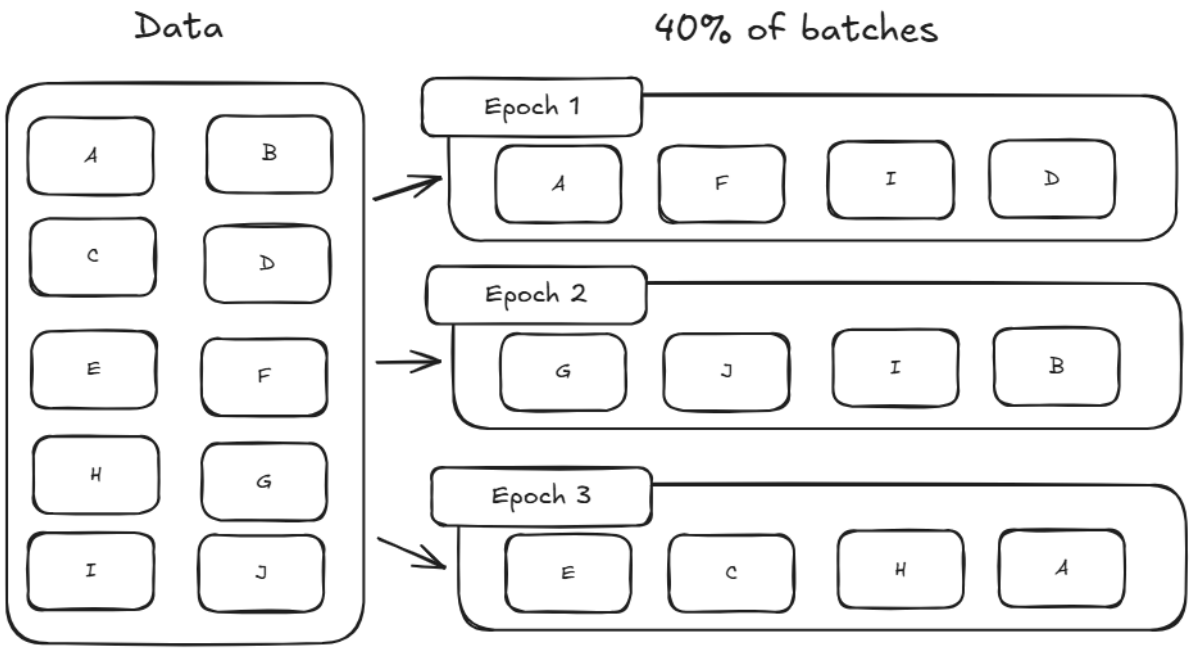


Figure 6.8: Principle of percentage of data used per epochs. The example displays the data separated into 10 batches. The model is then, given the provided percentage, taking a portion of the data by taking 40% of the batches randomly for a specific epoch. For each epoch, the batches selected are randomly chosen again. The randomness related to one epoch isn't influenced by others epochs results. It means it can have multiple times the same batch accross the epochs.

Another modification not related directly to the training has been done in the dl1_data_handler library on DLDataReader class to parallelize access to the files. Explanation can be found in the corresponding chapter (7.2.2).

Here is an overview of the training steps with the CTLearn tool. The first thing is to process the data by using a data reader that will process and load the data. Then, the data is separated in two sets : train and validation. After that, the callbacks are set up. The callbacks [48] are a sort of event listener that will do a task after each epoch like updating the learning rate or verifying

if early stopping requirements are met. Like this, the initialization of the tool is done. The next step is to load the model, define the basic elements of the training like the learning rate or the optimizers. After few minor set ups, like the pruning of the model, the model is compiled and the training started. The figure 6.9 represents the tasks the tool is showing the steps within the tool.

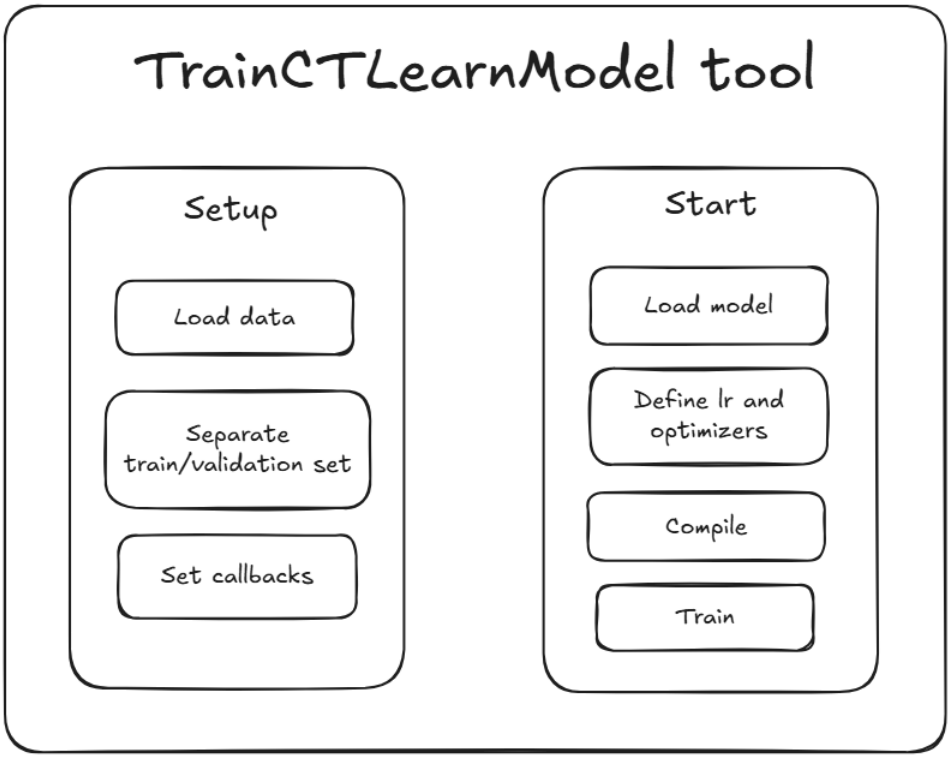


Figure 6.9: Repartition of steps within the TrainCTLearnModel tool. This tool has multiple features like the fact it processes the data to be in an adapted format. It also prepares the environment to train the model with callbacks. The model is then loaded inside the tool, compile and trained.

This concludes the internal training process for CTLearn library. There is still one element to talk about in this training task which is the metrics. Both for the training and testing task, some metrics are saved in a JSON file for additional information in the report and the comparison of models.

The metrics stored are the following ones :

- **Training time** : Time in millisecond to train a full model.
- **Training events** : Number of events used to train the model.
- **Number of parameters** : Number of parameters in the model.
- **Number of layers** : Number of layers in the model.
- **Estimated flops** : Number of FLOPs (Floating point operations per second) for one inference.

For each of theses metrics, a complete description and their interest have been mentioned in a dedicated chapter (5.3.1.2). The idea, here is to describe how to obtain theses information.

The number of events and the time taken to train the model is rather simple to obtain. To calculate the time needed to train the model, a timer is started just before the run function from

the TrainCTLearnModel instance and closed just after ending the training. For the number of events, there is an internal function in CTLearn that retrieves the number of events.

```
self.training_events = model.dl1dh_reader._get_n_events()
```

Concerning the number of parameters and layers of the model, the number of parameters is easy to retrieve. Keras provides an internal function for that. The number of layers is more complicated to retrieve and required to define a entire function for that. Normally, to display the number of layers, there are some internal parameters that can be used. The issue with this is that it doesn't take in account sub-layers from Functional layer or nested model. To fix this, a recursive function is in place to get all the layers recursively by going inside the whole model architecture (also inside nested model) to get the real number of layers. The figure 6.10 represents the function.

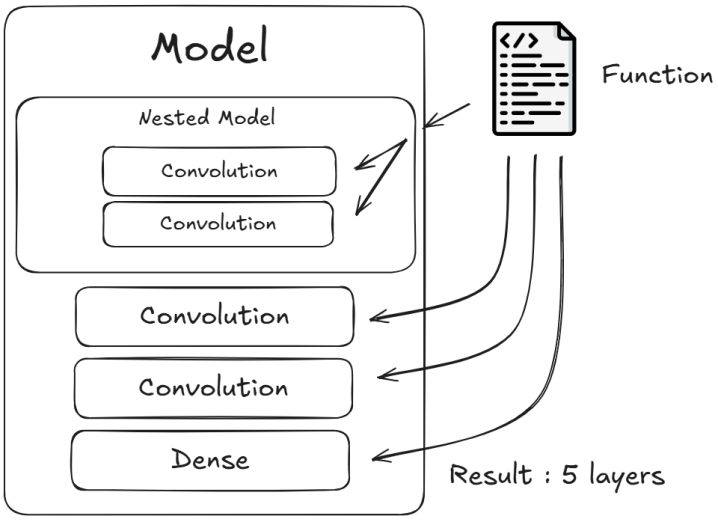


Figure 6.10: Representation of how the model retrieves the number of layers recursively. It goes inside the nested model / Functional layers to get the layers. It's important to know that some layers aren't taken in account like the input layer or the nested model itself. It has no interest to count an artificial layer.

For the number of FLOPs, it's even more complicated. Multiple solutions have been tried but because of the complex architecture of models, most of the time it didn't work. The solution for this scenario was, using TensorFlow, to convert the model to a graph function and use TensorFlow Profiler [49] to extract the number of FLOPs. It's important to know that this method only estimates the number of FLOPs per inference. The steps can be visualized with the figure 6.11.

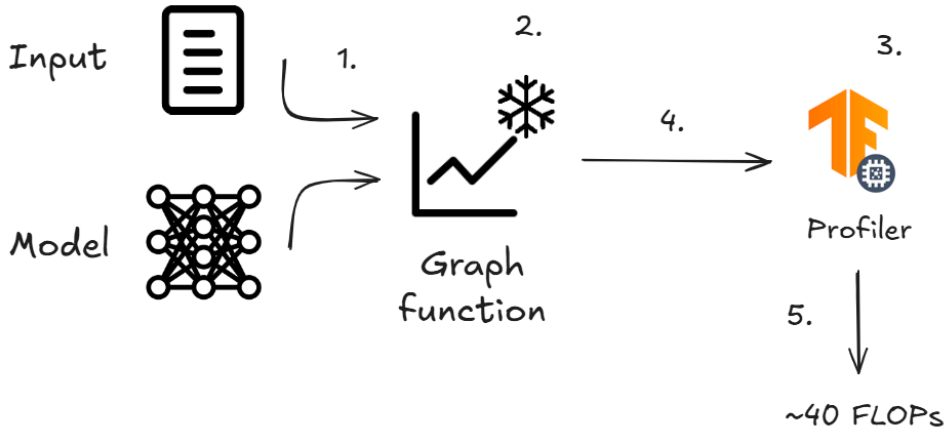


Figure 6.11: Process to obtain FLOPs from a model. 1. A dummy input is created and used together with the model to convert it in a graph function. 2. The graph is freezed so it can't be modified with manipulation. 3. TensorFlow Profiler is set up. 4. The graph is passed to the Profiler to retrieve performance metrics. 5. The estimated number of FLOPs is extracted

This concludes the training task.

6.3.3 Testing

The testing task is the logical step to do after training a model. Once a model is available, it can be tested on the dedicated data. Normally, if the configuration file is correctly setup, there should only be a few things to change. This is again to enhance reproducibility and automation of the whole process of model generation. It relies on two similar tools from the CTLearn library just like with the training step : MonoPredictCTLearnModel and StereoPredictCTLearnModel. Theses two tools inherits from the same class and the difference between them relies in the mode they handle.

Some parameters needs to be set in order for this task to execute correctly. There is only a few of them to set.

- **Data path** : Path to the data folder (global data folder, not a specific one).
- **Mono/Stereo mode** : Whether to handle one telescope image or several telescope image at once.

Depending on the mode and the task defined in the configuration file, one of the tools is chosen and executed. This step will produce HDF5 files just like the data provided with an additional section in it dedicated to the predictions of the model (DL2). They are stored in a dedicated folder in the directory allocated for the model.

Concerning the testing task, the predict_model.py file has been modified to comply with the thesis usage of models. It includes the two tools for the predictions. The modification applied is a fix for handling of the particle classification model. The fix is described there (7.2.3).

Like for the training task, there are some metrics calculated and stored in JSON file for future usage when generating a report or comparing models. The number of testing events and the total testing time (all inference time) are retrieved easily by defining a timer and using the internal function from CTLearn to count events.

6.4 Model Optimization

The model optimization is a large subject around model improvements. The idea is to modify/configure the model to be the most optimized for a designed task. This can be categorized in multiple sub-parts for a specific aspect of the model optimization. This includes optimization of training/performance by tuning hyperparameters or selecting a adequate optimizer/loss function, optimizing the deployment of the model by using compression tools, and others topics like this.

The model optimization of models, as part of thesis, concentrated essentially on optimization for compression of the model and computational cost efficiency. This aspect is explored for multiple reasons. The models generated with the CTLearn library haven't any resources issues because of their usage on the cluster (3.6.1). However, in the production environment, this might change.

The production environment uses a filtering system using triggers to reduce the massive amount of data captured by the telescope (3.2). Theses triggers uses algorithm and small models to do the filtering on a FPGA device. With this in mind, it has been considered to use the models generated with the CTLearn library on the FPGA device as a new triggering system, expecting to upgrade the current filtering system. The only issue with this is the fact that the models are currently too big and resource-intensive to be printed on the FPGA.

That's when the model optimization intervenes. By compressing and optimizing resources required, it could be possible to use theses models on low-end devices like an FPGA.

There are multiple possibilities to optimize models for an FPGA usage. Theses technics should comply with the structure of the CTLearn library (used to generated models). Therefore, only few technics have been tried as some of them like the distillation would require a lot of modification on the library.

6.4.1 Pruning

Pruning [50] [51] is an optimization technic used to optimize a model size. The core idea is to remove unimportant parameters from a neural network to reduce the size of the model and the computational complexity. The parameters are represented as the weights applied on the node and the biases. The removal of theses parameters can take various forms such as a random removal of weights or the removal of a percentage of less important parameters. The figure 6.12 displays an example of what pruning looks like.

This model optimization strategy works very well on Deep Learning networks, as they generally are over-parameterized with lots of layers and complex structure. The idea behind it is to shrink models by removing weights, neurons and even more so layers to reduce the size of the model and also the computational calculation when an inference is conducted. The removal is conducted by setting the weights to 0, therefore removing their value and cost in the calculation.

The downside of pruning is the accuracy loss. By removing parameters, the accuracy of the model is dropping. Depending on the pruning configuration, it can even become a real issue that the model is no longer of interest due to a lack of precision. The pruning needs, therefore, to be tested and balanced between the compression/inference wanted and the accuracy accepted. Another interesting characteristic from pruning is that, depending on the type of model (e.g. well trained models), it generalized the model, therefore making it better for production.

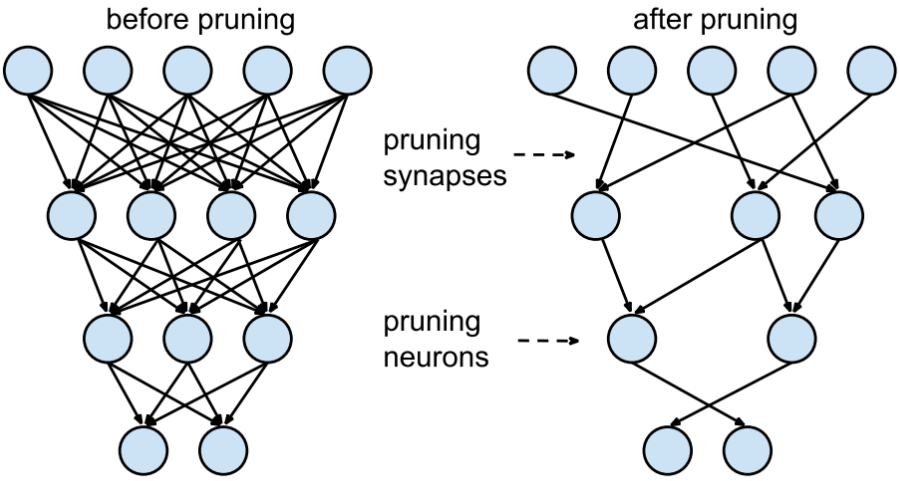


Figure 6.12: Pruning process where the model is displayed before and after the pruning optimization. Two levels of pruning are displayed here: the weights pruning represented by the synapses where the connections are removed and the neurons pruning when an entire matrix of weight is removed. [52]

There are two approaches [53] when talking about pruning : train-time pruning and post-training pruning. As the name suggests, the first one is conducted simultaneously with the training process and the second one is done after the model is fully trained. The train-time pruning does the pruning decisions simultaneously with the weight updates, to ensure a better sparsity in the network and to know which weights are less important. It's done through the iterations. The post-training pruning is done with the fully trained model. It's simpler to implement but it could affect the accuracy a lot more as the pruning isn't optimal. On the contrary, the train-time pruning produces more efficient models and the pruning process is optimized. However, the training step is a lot more complex to execute.

For the post-training model, there are multiple sub-types of pruning. The unstructured pruning is a naive approach where a threshold is used to define which parameters are removed. That means it helps to denoise/generalize a model but it doesn't improve that much the inference time. The structured pruning, on the other hand, relies on removing structure of weights like neurons or complete layers. This approach improves a lot the model inference but can drastically reduce the accuracy depending on the configuration.

The pruning technic is an interesting approach to optimize the model. It's particularly adapted to an FPGA use case as pruning helps to optimize resource usage.

Implementation in CTLearn

In order to operate in accordance with CTLearn, modifications have been made on some library's files. The idea behind it was to allow users to configure the pruning and add if wanted to the model. The configuration of the pruning can be provided by the configuration file where parameters are mentionned.

The type of pruning used is the train-time pruning with a integrated TensorFlow tool called **tfmot**. It's a TensorFlow module [54] that provides model optimization solutions for Keras models. This tool provides a configurable solution to implement pruning.

It's using the prune_low_magnitude function with a pruning schedule called PolynomialDecay. The particularity of this schedule process is that the model is pruned less at early stage than later stage to preserve early learning and avoid accuracy drops in the training of the model.

Three parameters are available to modifications from the configuration file.

- **Initial sparsity** : Indicates the sparsity (percentage of weights at 0) of the model when pruning begins. Generally, you want it to be at 0 for a first training.
- **Final sparsity** : Indicates the sparsity wanted at the end of the pruning
- **Begin step** : When to start pruning in the current epoch (after which number of batches). Generally, you don't want it to be zero to avoid unstable training

```
training_model:
  TrainCTLearnModel:
    pruning_model: # Pruning settings to use
      initial_sparsity: 0.00
      final_sparsity: 0.90
      begin_step: 1000
```

Some changes needs to be applied on the TrainCTLearnModel tool from CTLearn library to add this pruning feature. First of all, a callback is defined to counter the current number of steps (batches) the training is at. It's useful to know when to start and stop pruning.

```
class TrainCTLearnModel(Tool):
    def setup(self):

        ...

        # Pruning callback
        if self.pruning_model is not None:
            pruning_callback = tfmot.sparsity.keras.UpdatePruningStep()
            self.callbacks.append(pruning_callback)
```

Then, the pruning schedule is created and applied to the CTLearn model before starting to train the model using the prune_low_magnitude function.

The pruning isn't applied directly but each layer of the model is wrapped and add a mask for each weight. This way, the weights aren't directly affected and the global mask for the pruning can be updated through the epochs. It also means that, afterwards, you need to strip the pruning process or the model will keep the mask and therefore ends up being bigger than the original.

```
class TrainCTLearnModel(Tool):
    def start(self):

        ...

        # Set pruning of model
        if self.pruning_model is not None:
            self.log.info("Pruning CTLearn model.")
            validate_trait_dict(self.pruning_model, ["initial_sparsity", "
                                                     final_sparsity", "
                                                     begin_step"])
            # Get number of batches (corresponding to step per epochs)
            # Get last step for model (global number of steps)
```

```
            end_step = self.steps_per_epoch * self.n_epochs
            self.log.info("Pruning: steps_per_epoch=%s, end_step=%s (n_epochs=%
                                                s)",
                          steps_per_epoch, end_step, self.n_epochs)
            # Set the pruning procedure
            self.log.info("Parameters for pruning: initial_sparsity=%s,
                                                final_sparsity=%s,
                                                begin_step=%s",
                          self.pruning_model["initial_sparsity"], self.
                                                              pruning_model[
                                                              "
                                                              final_sparsity
                                                              "], self.
                                                              pruning_model[
                                                              "begin_step"])
            pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(
                                                initial_sparsity=self.
                                                pruning_model["
                                                initial_sparsity"],
                                               final_sparsity=self.
                                                                              prun
                                                                              [
                                                                              "
                                                                              fina
                                                                              "
                                                                              ]
                                                                              ,
                                               begin_step=self.pruning_model[
                                                                              "
                                                                              begi
                                                                              "
                                                                              ]
                                                                              ,
                                               end_step=end_step)
            # Apply pruning to the model
            self.model = prune_low_magnitude(self.model, pruning_schedule=
                                                pruning_schedule)

            ...

...

# Restore model state if pruning
if "pruning_model" in config_training.TrainCTLearnModel:
    pruned_model_path = os.path.join(config_training.TrainCTLearnModel.
                                        output_dir, "ctlearn_model.cpk")
    # Load pruned model
    with tfmot.sparsity.keras.prune_scope():
        pruned_model = tf.keras.models.load_model(pruned_model_path)
    # Strip model of pruning weights
    strip_model = tfmot.sparsity.keras.strip_pruning(pruned_model)
    # Save stripped model
    strip_model.save(pruned_model_path)
```

This concludes the pruning step.

6.4.2 Quantization

The quantization [55] is another optimization technic. It's rather simple to understand. The idea is to reduce the memory needs of the model by representing the weights with less precision (e.g. going from 8bit precision to 4bit precision). It also improves the computing efficiency, consumption and inference of the generated model. This alteration affects the model size because weights, neurons or layers are encoded in smaller size, therefore affecting the global model size. Less precision comes with a downside, which is the accuracy drop. A last interesting aspect is that, by doing quantization, the model could become able to run on more machines than couldn't handle high precision for example.

The quantization process can be represented by the figure 6.13.

Figure 6.13: Quantization process where the left is encoded in FP32 (Floating Point 32), with 32 bits for each value, and converted to INT8, encoded in 8 bits. It means the value is going from 4 billions possibilities to only 256, considerably improving matrix multiplication. Two things here can be observed : the reduction of the number of bits and the conversion into integers of the values. The parameters of the quantization aren't provided here, but depending on the settings, the function mapping for the integers values could change and gives a totally different output. [56]

The quantization can be applied at multiple parts of the model. It could be to weights, activations function or biases. It means the quantization process needs to be correctly defined and in order to operate correctly, the same process needs to be applied for each weight of a layer, model or neuron to keep the logic and the information from the model.

Like the pruning, there are different ways [51] to incorporate quantization to a model. There is the Post-Training Quantization (PTQ) and the Quantization-Aware Training (QAT).

PTQ is done after the training and is rather simple to put in place. The quantization is applied quickly through the entire model. It also means the quantization might not be optimal, resulting in considerable accuracy drops. Generally, this approach uses calibration data (subset of the dataset) to reduces this accuracy issue and the potential approximation errors that can impact the model.

The calibration data will help to determine the quantization parameters for the model and mitigate approximation errors by evaluating the model's performance with the calibration data. Depending on the results, the parameters can be updated to preserve as much as possible the accuracy.

QAT, on the other hand, is done in parallel, during the training. It's done so the model can adapt itself to work in lower precision levels. It helps the model to compensate itself approximation errors it could do. It can even be used to fine-tune the quantization process at some point. Because of the fact that it's conducted during the training, the training process is slightly longer but it produces better results than a PTQ approach.

The implementation with the CTLearn library wasn't conducted due to issues with quantization of complex models. A dedicated chapter explains this challenge (7.2.4). The main idea was to conduct PTQ approach at first and then explore a QAT approach when a first quantization process was stabilized.

6.5 New Models

Models presented earlier are classic models/configuration available on the CTLearn library. The custom model implementation (6.2.3) with the template provided allows to integrate new models with interesting features. This section is dedicated to theses new models that could bring interesting aspects of Deep Learning or completely new architectures that might improve performances on the classification and regression. It can be also seen as the first step to explore new possibilities using thesis work.

7 Conclusion

This chapter contains elements necessary to evaluate the work done on the thesis. This is to evaluate the global progress of the project. It will include sections dedicated to difficulties encountered or future improvements.

7.1 Conclusion

7.2 Challenges

During the thesis, at multiple occasions, some challenges appeared and must be resolved for the thesis to proceed smoothly. The idea is to find a solution or at least identify possibilities to address the issue. Some of them couldn't be resolved because of limitations, technicalities or even time available.

7.2.1 Link GPU to Tensorflow

The first challenge that occurred during the thesis is the usage of GPU on the baobab cluster (3.6.1). In order to train neurals networks on a decent amount of data, it requires to use GPUs to accelerate the whole process. However, GPUs with TensorFlow requires to have a precise configuration of the environment and compatibility.

The first issue comes from the fact that GPUs aren't automatically visible for the bash script. It's SLURM that handles this aspect and you must provide in details GPU configuration through the script to make it work. The second issue related to the first one is the fact that version of TensorFlow, CUDA and cuDNN must match to work.

This results in the task running on CPU because it couldn't detect GPUs. Multiples solutions have been tried to solve this. Assigning modules with specific CUDA version or directly referencing a PATH inside the Python script are examples of things tried.

It's only after combining multiple solutions that the GPUs could finally be detected by the Python Script.

The first element introduced is the flag **–nv** when running the singularity. It exposes the host GPU devices and drivers to the container and enables NVIDIA GPU Support. The second element introduced is the binding of the CUDA libraries. It mount CUDA libraries from the host inside the container to match driver version.

```
--bind /usr/local/cuda/targets/x86_64-linux:/usr/local/cuda/targets/x86_64-
                                            linux
--bind /usr/local/cuda/lib64:/usr/local/cuda/lib64
```

The last step is to define a variable to force the dynamic linker to prioritize the host CUDA libraries. Because the host CUDA libraries are exposed in the container, TensorFlow might choose the wrong one. It prevents that scenario to happen.

```
export LD_LIBRARY_PATH="/usr/local/cuda/targets/x86_64linux/lib:/usr/local/cuda
                                            /lib64:${LD_LIBRARY_PATH:-}"
```

With theses three element combines, it's possible to run tasks on GPU.

7.2.2 Accelerate training speed

–> Parallelize –> Percentage of epochs –> Batch Size –> Change path –> Merge files

7.2.3 CTLearn library

Throughout the thesis, manipulations were required on the CTLearn library to adapt and add new elements corresponding to the MLOps approach or to solve challenges with the production environment. Waiting on merge requests and new releases weren't ideal. Therefore, an idea was put in place.

It consists on keeping modified files locally and redirect requests on it. The code directly points to theses files that themselves falls back to CTLearn library files. Theses files were implemented as tools to help Python scripts to use them. It avoids relying on system-installed version of the library. This enables quick maintenance and reusability of the scripts.

The files implicated were the tools **train_model.py** and **predict_model.py** and more especially their tools included in it. It allows to directly have a usable library on the cluster without doing complicated procedures to use a modified library.

The files with their modification are available in the appendix.

7.2.4 Quantization of complex models

Among the model optimization technics that can be applied to models, Quantization has been an issue. Normally, when working with simple and small models, the quantization is applied correctly to each layer without an issue. The problem is that with complex nested models, the quantization step isn't applied correctly to sub-layers. The quantization process fails to reach deeper layers. It's wanted to have a similar quantization process applied at each layer to keep uniformity and knowledge of the quantization applied.

This issue couldn't be resolved in time and the quantization was abandoned to progress in the thesis. A Jupyter Notebook is available in the appendix, describing all methods used to try and quantize a system.

7.3 Limitations

The thesis has faced multiple limitations that slowed or stopped the advancement of the model generation and related tools in place.

- **CTLearn Library** : CTLearn library is a well-developed library about deep learning on IACT. However, the library isn't easy to understand and use. Manipulating the library is also difficult because of numerous interdependencies and complex structure that breaks if changes aren't meticulously introduced.
- **Cluster limitations** : The cluster limited the training of models in multiple ways. First of all, the training isn't directly executed because Baobab (3.6.1) uses a queue system. The cluster also establishes time limits for a specific task, blocking long training multiple times. It can also be limited in terms of resources usable.

7.4 Future improvements

The thesis introduced the MLOps methodology and tools to manage the generation of models. This is only the introduction of it and multiple things could be done to further improve the CTLearn library and the generation of models, going towards better solutions to reconstruct gamma rays. With this limited time, It was only possible to scratch the surface of lots of aspects introduced during the thesis. Some ideas to explore in the future are listed below.

- **Get deeper in the MLOps approach** : Adding new MLOps features to further automate, optimize, etc. the generation of the model and its monitoring/maintenance. As an example, automation through external tools like GitHub with CI/CD or versioning of data and models with Weights&Biases could be an interesting start.
- **Find better models to reconstructed gamma ray** : The possibility has been enable to use custom models with the template (6.2.3) to test new models. It could be explored and benchmarked to find the most suitable model and hopefully, a better one.
- **Implementing models on FPGA** : Going further in model optimization could be interesting. Triggering system could benefit to use neural networks with high performance and low resource requirements and computation time.
- **Combine tasks** : Something mentioned at the start of the project but never again is the combination of tasks. To centralize the prediction, it could be useful to have a unique model predicting three outputs : type, energy and direction.
- **Test on real data** : Up until now, the models were trained, tested and compared on the basis of simulated data. It's important to verify if performance may varies when working with real data. Real data are a lot more instable and random than simulated data.
- **Optimize model generation** : The model generation did caused problems because of the time it took to train. Also related to that, the cluster has some time limitations that blocks a classic training of a model. Looking at ways to optimize model could be interesting.

7.5 Personal reflection

This type of project is an unique opportunity. I like astronomy and working on something related to this field was exiting for me. At first, I didn't thought about the difficulty of the subject. It started becoming clearer as the thesis progresses. I learned a lot of things about gamma rays and telescopes when doing my researches. I'm frustrated because I'm starting to really understand the physics side of the thesis, only when it's about to finish.

This project helped me to improve my skills, especially the understanding of Deep Learning models and MLOps methodology. Related to that, I could improve my capacity to adapt to unforeseen events like the limitations on the cluster or to complex library like CTLearn. I learned to understand a library in details and take advantage of it when building tools. I also learned new stuff like working with large datasets and challenges related to it or how to work with a cluster (SLURM and queue system).

The results of the thesis is also a bit frustrating as I haven't add enough time to explore deeper in the implementation of MLOps approach. I feel like I've only touched on various topics without going into detail.

I hope the solutions provided will help the CTLearn community and will be a basis in the future to generate new models. I introduced the subject of MLOps in CTAO and I hope it will continue going deeper in the methodology as time passes. I was happy to participate in global meetings with others collaborators to explore further the gamma ray domain.

8 Declaration of honor

I, the undersigned Hugo Varenne, hereby declare on my honor that the submitted work is the result of my own personal effort. I certify that I have not engaged in plagiarism or any other form of fraud. All sources of information used and any author quotations have been clearly referenced.

Bibliography

[1] Homepage, . URL https://www.ctao.org/.

[2] Heavy Nuclei - an overview | ScienceDirect Topics. URL https://www.sciencedirect.com/topics/earth-and-planetary-sciences/heavy-nuclei.

[3] Matteo Cerruti. Leptonic and Hadronic Radiative Processes in Supermassive-Black-Hole Jets. *Galaxies*, 8(4):72, October 2020. ISSN 2075-4434. doi: $10.3390/\mathrm{galaxies8040072}$. URL http://arxiv.org/abs/2012.13302. arXiv:2012.13302 [astro-ph].

[4] Celestial Bodies. URL https://unacademy.com/content/cbse-class-12/study-material/physics/celestial-bodies/.

[5] Gamma-ray astronomy, November 2025. URL https://en.wikipedia.org/w/index.php?title=Gamma-ray_astronomy&oldid=1320843996. Page Version ID: 1320843996.

[6] Provided documentation by unige on ctao general understanding, . URL None.

[7] E. A. Kuraev, Yu M. Bystritskiy, M. Shatnev, and E. Tomasi-Gustafsson. Bremsstrahlung and pair production processes at low energies, multi-differential cross section and polarization phenomena. *Physical Review C*, 81(5):055208, May 2010. ISSN 0556-2813, 1089-490X. doi: $10.1103/\mathrm{PhysRevC.81.055208}$. URL http://arxiv.org/abs/0907.5271. arXiv:0907.5271 [hep-ph].

[8] Thorsten Buss, Frank Gaede, Gregor Kasieczka, Anatolii Korol, Katja KrÃijger, Peter McKeown, and Martina Mozzanica. CaloHadronic: a diffusion model for the generation of hadronic showers, June 2025. URL http://arxiv.org/abs/2506.21720. arXiv:2506.21720 [physics].

[9] Mario Pecimotika. *Transmittance Simulations for the Atmosphere with Clouds*. PhD thesis, November 2018.

[10] P. Rajda, K. ZiÄŹtara, W. Bilnik, J. BÅĆocki, L. Bogacz, T. Bulik, F. Cadoux, A. Christov, M. CuryÅĆo, D. della Volpe, M. Dyrda, Y. Favre, A. Frankowski, ÅĄ Grudnik, M. GrudziÅĎska, M. Heller, B. IdÅžkowski, M. Jamrozy, M. Janiak, J. Kasperek, K. Lalik, E. Lyard, E. Mach, D. Mandat, A. MarszaÅĆek, J. MichaÅĆowski, R. Moderski, M. Rameez, T. Montaruli, A. Neronov, J. Niemiec, M. Ostrowski, P. PaÅŻko, M. Pech, A. Porcelli, E. Prandini, E. jr Schioppa, P. Schovanek, K. Seweryn, K. Skowron, V. Sliusar, M. SowiÅĎski, ÅĄ Stawarz, M. Stodulska, M. Stodulski, S. Toscano, I. Troyano Pujadas, R. Walter, M. WiÄŹcek, A. ZagdaÅĎski, and P. Åžychowski. DigiCam - Fully Digital Compact Read-out and Trigger Electronics for the SST-1M Telescope proposed for the Cherenkov Telescope Array, August 2015. URL http://arxiv.org/abs/1508.06082. arXiv:1508.06082 [astro-ph].

[11] M. Heller, E. jr Schioppa, A. Porcelli, I. Troyano Pujadas, K. ZiÈľtara, D. della Volpe, T. Montaruli, F. Cadoux, Y. Favre, J. A. Aguilar, A. Christov, E. Prandini, P. Rajda, M. Rameez, W. Bilnik, J. BÅĆocki, L. Bogacz, J. Borkowski, T. Bulik, A. Frankowski,

M. GrudziÅĎska, B. IdÅžkowski, M. Jamrozy, M. Janiak, J. Kasperek, K. Lalik, E. Lyard, E. Mach, D. Mandat, A. MarszaÅĆek, L. D. Medina Miranda, J. MichaÅĆowski, R. Moderski, A. Neronov, J. Niemiec, M. Ostrowski, P. PaÅŻko, M. Pech, P. Schovanek, K. Seweryn, V. Sliusar, K. Skowron, ÅĄ. Stawarz, M. Stodulska, M. Stodulski, R. Walter, M. WiĚľcek, and A. ZagdaÅĎski. An innovative silicon photomultiplier digitizing camera for gamma-ray astronomy. *The European Physical Journal C*, 77(1):47, January 2017. ISSN 1434-6052. doi: 10.1140/epjc/s10052-017-4609-z. URL https://doi.org/10.1140/epjc/s10052-017-4609-z.

[12] Tjark Miener, Daniel Nieto, Ari Brill, Bryan Kim, and Qi Feng. CTLearn: Deep learning for imaging atmospheric Cherenkov telescopes event reconstruction, March 2025. URL https://zenodo.org/doi/10.5281/zenodo.3342952. Language: en.

[13] J. A. Barrio. Status of the large size telescopes and medium size telescopes for the Cherenkov Telescope Array observatory. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 952:161588, February 2020. ISSN 0168-9002. doi: 10.1016/j.nima.2018.11.047. URL https://www.sciencedirect.com/science/article/pii/S016890021831622X.

[14] Alessio Berti. Observation of the gamma-ray sky from the ground: the IACT technique.

[15] https://www.unige.ch/sciences/astroparticle/projects/sst, March 2020. URL https://www.unige.ch/sciences/astroparticle/projects/sst. Last Modified: 2024-05-27T10:12:23Z.

[16] The first Cherenkov telescopes in the Czech Republic to observe the Crab Nebula - Akademie vÄŽd ÄŇeskÃĺ republiky. URL https://www.avcr.cz/en/news-archive/The-first-Cherenkov-telescopes-in-the-Czech-Republic-to-observe-the-Crab-Nebula/.

[17] High Performance Computing - e-Research - UNIGE, July 2019. URL https://www.unige.ch/eresearch/en/services/hpc. Last Modified: 2025-02-20T12:44:23Z.

[18] High Performance Computing - e-Research - UNIGE - Baobab, July 2019. URL https://doc.eresearch.unige.ch/hpc/hpc_clusters#for_advanced_users.

[19] Commande SCP de Linux, September 2021. URL https://www.ionos.fr/digitalguide/serveur/configuration/commande-scp-de-linux/.

[20] Lmod: A New Environment Module System âĂŤ Lmod 9.0.5 documentation. URL https://lmod.readthedocs.io/en/latest/index.html.

[21] Introduction to Singularity âĂŤ Singularity container 3.5 documentation. URL https://docs.sylabs.io/guides/3.5/user-guide/introduction.html.

[22] Slurm Workload Manager - Quick Start User Guide. URL https://slurm.schedmd.com/quickstart.html.

[23] marcel. Beegfs. URL https://www.beegfs.io/c/.

[24] Konrad Bernlohr. Simulation of Imaging Atmospheric Cherenkov Telescopes with CORSIKA and sim_telarray. *Astroparticle Physics*, 30(3):149–158, October 2008. ISSN 09276505. doi: 10.1016/j.astropartphys.2008.07.009. URL http://arxiv.org/abs/0808.2253. arXiv:0808.2253 [astro-ph].

[25] PhD thesis Gasparetto. Cta hierarchical data model. Technical report, UniversitÃă degli Studi di Trieste, 2025. URL https://arts.units.it/retrieve/e2913fdd-4852-f688-e053-3705fe0a67e0/thesis_PHD_Gasparetto_Reviewed_resized.pdf. Defines R0, R1, DL0, etc.

[26] What is Monte Carlo Simulation? URL https://www.geeksforgeeks.org/artificial-intelligence/what-is-monte-carlo-simulation/. Section: Artificial Intelligence.

[27] MÃľthode de Monte-Carlo, September 2025. URL https://fr.wikipedia.org/w/index.php?title=M%C3%A9thode_de_Monte-Carlo&oldid=228963734. Page Version ID: 228963734.

[28] Jordan Pannell. Monte Carlo estimation of pi, August 2020. URL https://jordanpannell.co.uk/blog/Monte-Carlo-estimation-of-pi/.

[29] HDF5: HDF5 Data Model and File Structure, . URL https://support.hdfgroup.org/documentation/hdf5/latest/_h5_d_m__u_g.html.

[30] Hierarchical Data Formats - What is HDF5? | NSF NEON | Open Data to Understand our Ecosystems, . URL https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5.

[31] SST-1M-collaboration/sst1mpipe, January 2026. URL https://github.com/SST-1M-collaboration/sst1mpipe. original-date: 2024-03-18T10:20:20Z.

[32] Cosimo Nigro, Tarek Hassan, and Laura Olivera-Nieto. Evolution of Data Formats in Very-High-Energy Gamma-Ray Astronomy. *Universe*, 7(10):374, October 2021. ISSN 2218-1997. doi: 10.3390/universe7100374. URL https://www.mdpi.com/2218-1997/7/10/374.

[33] K. Abe, S. Abe, A. Abhishek, F. Acero, A. Aguasca-Cabot, I. Agudo, C. Alispach, N. Alvarez Crespo, D. Ambrosino, L. A. Antonelli, C. Aramo, A. Arbet-Engels, C. Arcaro, K. Asano, P. Aubert, A. Baktash, M. Balbo, A. Bamba, A. Baquero Larriva, U. Barres de Almeida, J. A. Barrio, L. Barrios JimÃľnez, I. Batkovic, J. Baxter, J. Becerra GonzÃąlez, E. Bernardini, J. Bernete Medrano, A. Berti, I. Bezshyiko, P. Bhattacharjee, C. Bigongiari, E. Bissaldi, O. Blanch, G. Bonnoli, P. Bordas, G. Borkowski, G. Brunelli, A. Bulgarelli, I. Burelli, L. Burmistrov, M. Buscemi, M. Cardillo, S. Caroff, A. Carosi, M. S. Carrasco, F. Cassol, N. CastrejÃșn, D. Cauz, D. Cerasole, G. Ceribella, Y. Chai, K. Cheng, A. Chiavassa, M. Chikawa, G. Chon, L. Chytka, G. M. Cicciari, A. Cifuentes, J. L. Contreras, J. Cortina, H. Costantini, P. Da Vela, M. Dalchenko, F. Dazzi, A. De Angelis, M. de Bony de Lavergne, B. De Lotto, R. de Menezes, R. Del Burgo, L. Del Peral, C. Delgado, J. Delgado Mengual, D. della Volpe, M. Dellaiera, A. Di Piano, F. Di Pierro, R. Di Tria, L. Di Venere, C. DÃŋaz, R. M. Dominik, D. Dominis Prester, A. Donini, D. Dorner, M. Doro, L. Eisenberger, D. ElsÃď sser, G. Emery, J. Escudero, V. Fallah Ramazani, F. Ferrarotto, A. Fiasson, L. Foffano, L. Freixas Coromina, S. FrÃűse, Y. Fukazawa, R. Garcia LÃșpez, C. Gasbarra, D. Gasparrini, D. Geyer, J. Giesbrecht Paiva, N. Giglietto, F. Giordano, P. Gliwny, N. Godinovic, R. Grau, D. Green, J. Green, S. Gunji, P. GÃijnther, J. Hackfeld, D. Hadasch, A. Hahn, T. Hassan, K. Hayashi, L. Heckmann, M. Heller, J. Herrera Llorente, K. Hirotani, D. Hoffmann, D. Horns, J. Houles, M. Hrabovsky, D. Hrupec, D. Hui, M. Iarlori, R. Imazawa, T. Inada, Y. Inome, S. Inoue, K. Ioka, M. Iori, A. Iuliano, I. Jimenez Martinez, J. Jimenez Quiles, J. Jurysek, M. Kagaya, O. Kalashev, V. Karas, H. Katagiri, J. Kataoka, D. Kerszberg, Y. Kobayashi, K. Kohri, A. Kong, H. Kubo, J. Kushida, M. Lainez, G. Lamanna, A. Lamastra, L. Lemoigne, M. Linhoff, F. Longo, R. LÃșpez-Coto, A. LÃșpez-Oramas, S. Loporchio, A. Lorini, J. Lozano Bahilo, H. Luciani, P. L. Luque-Escamilla, P. Majumdar, M. Makariev, M. Mallamaci, D. Mandat, M. Manganaro, G. ManicÃš, K. Mannheim, S. Marchesi, M. Mariotti, P. Marquez, G. Marsella, J. MartÃŋ, O. Martinez, G. MartÃŋnez, M. MartÃŋnez, A. Mas-Aguilar, G. Maurin, D. Mazin, J. MÃľndez-Gallego, E. Mestre Guillen, S. Micanovic, D. Miceli, T. Miener, J. M. Miranda, R. Mirzoyan, T. Mizuno, M. Molero Gonzalez, E. Molina, T. Montaruli, A. Moralejo, D. Morcuende, A. Morselli, V. Moya, H. Muraishi, S. Nagataki, T. Nakamori, A. Neronov, L. Nickel, M. Nievas Rosillo, L. Nikolic, K. Nishijima, K. Noda, D. Nosek, V. Novotny, S. Nozaki, M. Ohishi, Y. Ohtani, T. Oka, A. Okumura, R. Orito, J. Otero-Santos, P. Ottanelli, E. Owen, M. Palatiello, D. Paneque, F. R. Pantaleo, R. Paoletti, J. M. Paredes, M. Pech, M. Pecimotika, M. Peresano, F. Pfeifle, E. Pietropaolo, M. Pihet, G. Pirola, C. Plard, F. Podobnik, E. Pons, E. Prandini, C. Priyadarshi, M. Prouza, S. RainÃš, R. Rando, W. Rhode, M. RibÃș, C. Righi, V. Rizi,

G. Rodriguez Fernandez, M. D. RodrÃŋguez FrÃŋas, A. Ruina, E. Ruiz-Velasco, T. Saito, S. Sakurai, D. A. Sanchez, H. Sano, T. ÅăariÄĞ, Y. Sato, F. G. Saturni, V. Savchenko, F. Schiavone, B. Schleicher, F. Schmuckermaier, J. L. Schubert, F. Schussler, T. Schweizer, M. Seglar Arroyo, T. Siegert, J. Sitarek, V. Sliusar, J. StriÅąkoviÄĞ, M. Strzys, Y. Suda, H. Tajima, H. Takahashi, M. Takahashi, J. Takata, R. Takeishi, P. H. T. Tam, S. J. Tanaka, D. Tateishi, T. Tavernier, P. Temnikov, Y. Terada, K. Terauchi, T. Terzic, M. Teshima, M. Tluczykont, F. Tokanai, D. F. Torres, P. Travnicek, A. Tutone, M. Vacula, P. Vallania, J. van Scherpenberg, M. VÃązquez Acosta, S. Ventura, G. Verna, I. Viale, A. Vigliano, C. F. Vigorito, E. Visentin, V. Vitale, V. Voitsekhovskyi, G. Voutsinas, I. Vovk, T. Vuillaume, R. Walter, L. Wan, M. Will, J. WÃşjtowicz, T. Yamamoto, R. Yamazaki, P. K. H. Yeung, T. Yoshida, T. Yoshikoshi, W. Zhang, and N. Zywucka. A new method of reconstructing images of gamma-ray telescopes applied to the LST-1 of CTAO. *Astronomy & Astrophysics*, 691:A328, November 2024. ISSN 0004-6361, 1432-0746. doi: 10.1051/0004-6361/202450889. URL https://www.aanda.org/articles/aa/abs/2024/11/aa50889-24/aa50889-24.html.

[34] Laetitia Guidetti. Provided documentation by unige on ctao general understanding, 2025.

[35] What is MLOps?, December 2021. URL https://www.databricks.com/glossary/mlops.

[36] Prince Canuma. MLOps: What It Is, Why It Matters, and How to Implement It, July 2022. URL https://neptune.ai/blog/mlops.

[37] What Is DevOps? | IBM, May 2025. URL https://www.ibm.com/think/topics/devops.

[38] Rick Merritt. What is MLOps?, September 2020. URL https://blogs.nvidia.com/blog/what-is-mlops/.

[39] What is MLOps? | IBM, April 2024. URL https://www.ibm.com/think/topics/mlops.

[40] Bastien Lacave. BastienLacave/CTLearn-Manager, December 2025. URL https://github.com/BastienLacave/CTLearn-Manager. original-date: 2024-12-18T18:01:02Z.

[41] D. Nieto, A. Brill, Q. Feng, T. B. Humensky, B. Kim, T. Miener, R. Mukherjee, and J. Sevilla. CTLearn: Deep Learning for Gamma-ray Astronomy, December 2019. URL http://arxiv.org/abs/1912.09877. arXiv:1912.09877 [astro-ph].

[42] Residual neural network, October 2025. URL https://en.wikipedia.org/w/index.php?title=Residual_neural_network&oldid=1314589246. Page Version ID: 1314589246.

[43] Residual Networks (ResNet) - Deep Learning. URL https://www.geeksforgeeks.org/deep-learning/residual-networks-resnet-deep-learning/. Section: Deep Learning.

[44] Introduction to Convolution Neural Network. URL https://www.geeksforgeeks.org/machine-learning/introduction-convolution-neural-network/. Section: Machine Learning.

[45] Ajitesh Kumar. Different Types of CNN Architectures Explained: Examples, December 2023. URL https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

[46] ReLU Activation Function in Deep Learning. URL https://www.geeksforgeeks.org/deep-learning/relu-activation-function-in-deep-learning/. Section: Deep Learning.

[47] Traitlets âĂŤ traitlets 5.14.3 documentation. URL https://traitlets.readthedocs.io/en/stable/.

[48] Keras Team. Keras documentation: Callbacks API. URL https://keras.io/api/callbacks/.

[49] Optimiser les performances de TensorFlow Ãă l'aide du profileur | TensorFlow Core. URL https://www.tensorflow.org/guide/profiler?hl=fr.

[50] Souvik Paul. Pruning in Deep Learning Models, June 2020. URL https://medium.com/@souvik.paul01/pruning-in-deep-learning-models-1067a19acd89.

[51] Alessandro Lamberti. Deep Learning Model Optimization Methods, March 2024. URL https://neptune.ai/blog/deep-learning-model-optimization-methods.

[52] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks, October 2015. URL http://arxiv.org/abs/1506.02626. arXiv:1506.02626 [cs].

[53] A Comprehensive Guide to Neural Network Model Pruning | Datature Blog. URL https://datature.com/blog/a-comprehensive-guide-to-neural-network-model-pruning.

[54] Module: tfmot | TensorFlow Model Optimization. URL https://www.tensorflow.org/model_optimization/api_docs/python/tfmot.

[55] What is Quantization? | IBM, July 2024. URL https://www.ibm.com/think/topics/quantization.

[56] Florian June. Model Quantization 1: Basic Concepts, May 2024. URL https://medium.com/@florian_algo/model-quantization-1-basic-concepts-860547ec6aa9.

List of Figures

3.1 Non-exhaustive of astrophysical objects than can collides with cosmic-ray and generate gamma-ray. [4] . . . 7
3.2 Gamma ray observation of the sky captured by the Fermi telescope for 5 years of observation. [5] . . . 7
3.3 Gamma ray shower development from collision with the atmosphere to Earth's ground. [1] . . . 8
3.4 Representation of an electromagnetic shower (left) and an hadronic shower (right). [9] . . . 9
3.5 Visual representation of Cherenkov light emitted by secondary particles from an air shower. [1] . . . 10
3.6 IACT telescopes. Three categories of telescopes are represented here. The difference relies in their size and number of mirrors. From the left to the right : SST, MST, LST. [1] . . . 13
3.7 SST-1M telescope. This telescope is part of project that includes the University of Geneva. Specifications of the telescope of the telescope are also provided to get a better idea of how the telescope and the camera captures information.[16] [15] . . . 14
3.8 Usage of a Linux virtual machine (WSL) to test elements. The tests and exploration are done using Jupyter Notebook to execute steps independently. The amount of data used is very small because of space issues and a low complexity requirement. Configurations files, models, libraries are completely unstructured and mix together. 15
3.9 Environment on the cluster. It's close to a production environment. The access to it is done through a terminal of command with SSH. Files on the cluster are highly structured to make it clean and easy to understand. To execute jobs, a bash script must be associated to the corresponding Python script. The python scripts represented a single task for each of them. They allows to generate reports and models and store them on the cluster. To generate theses elements, Python scripts relies on a big amount of data and configurations files. The data is structured to be easy to use. . . . 15
3.10 Example of environment configuration saved in a YAML file. Generally, there are three big sections : the channels (packages repositories), the dependencies (installed via Conda) and the subsection for pip dependencies (installed via pip). . . . 17
3.11 Example of a bash script used to run a job on SLURM. It's separated in three sections : a part related to configuration of the job for SLURM, additional commands for the task and the main command to run the corresponding Python script. . . . 18

4.1 Visualization of parts of the simulation. The simulation can be separated in two elements : CORSIKA and sim_telarray. The first one focus on initializing a particle and build interaction with the atmosphere. It includes the generation of the air shower and related Cherenkov light. On the other hand, the second component focus on simulating the detector part : camera and telescope. It includes the triggering level system from it. [6] 24
4.2 Monte Carlo simulation. Using the randomness of parameters, simulations have been produced in order to estimate the value of ÏĂ in this situation. It doesn't need to compute every possibilities to approximate ÏĂ as the right graphic displays. [28] 25
4.3 Example of HDF5 file structure with three elements (group, dataset, attributes) employed. By connecting groups and datasets, a structured dataset can be realized and separation between data can be optimal while having everything on the same file. Understanding and working with this format is easy because of the metadata distributed at each level of the hierarchy. [30] 26
4.4 Workflow of data level (excluding R0 and R1). It shows that extraction of various information is done at every stage of data using the previous results. DL3 and DL4 are just for the show as they aren't part of the thesis. [32] 27
4.5 Example of R0 waveforms. It's a representation of the ADC counter, representation of the signal for a pixel, for a specific time slice with an integer value for each nanosecond in it. It's a representation of data detected by the camera of the telescope in the purest way. [6] 28
4.6 Example of R1 waveforms. This is calibrated waveforms with a clear distinction between the noise and the air shower visible. The photoelectrons are represented by the y-axis. [33] 29
4.7 Extraction of the valuable information from the waveform. It takes the peak of the signal in account and keep a small window around it. Then the charge is calculated by adding up the charge in photoelectrons. This gives the value for the pixel. [6] . 30
4.8 Hillas parameters extraction. This displays various parameters computed during an Hillas parametrization on a per-event pixel array. The parametrization is done on an event from a clean array as the picture includes. Some examples of parameters : Size, Width, Length, Center of Gravity, Leakage, etc. [6] 31
4.9 Example of an event in pixel array or frame. It represents on the left part of the pixel grid of the telescope image. Theses pixels are hexagonal and needs to be converted to rectangle ones to work with deep neural networks and classic libraries. [6] 32

5.1 MLOps cycle representing the life-cycle for a model and the related data in order to deploy a model in a production environment. Deploying a model in a production environment isn't the end of the process as monitoring and maintenance are continuously provided to the deployed model, conducting eventually in a refinement of the model. This is mean the cycle can restart at some point. [38] 35
5.2 Task decomposition of the pipeline to generate a model and monitor it. The pipeline executed in a Jupyter Notebook is separated in three sub-tasks : training, testing and report generation of the model. This ensures more flexibility when executing tasks and more adapted to monitoring and maintenance purposes. Another task is included in this picture : comparison of the models. This tasks is outside the pipeline because it requires multiple models to work. It's considered in the task decomposition as its usage is the same as the others sub-tasks. 37

5.3 Example of configuration file for a model generation. The task handled by the model is the particle classification of telescope images. The structure of the configuration file is important and must be kept as such to work correctly. New parameters can be added depending on the configuration wanted and available attributes in CTLearn library. 38
5.4 Graphic containing the training loss and the validation loss obtained through epochs. This helps to understand possible problems during training like underfitting or overfitting. This could also indicates that the current number of epochs isn't enough to converge. 42
5.5 Model summary containing information about layers of the model. Additional information like the total number of parameters and the size of the model are provided. 43
5.6 ROC Curve graphic displayed using TPR and FPR. There is also a diagonal displaying random guess values. The legend display the model label and the AUC to get a better overview of the model performance on the curve. 46
5.7 Gammaness distribution graphic where the distribution for each type of particle can be seen. The axis used are the gammaness and the density of events contained at each level of gammaness. In the example, the separation between gamma and hadron can be seen. It's not clear which means some improvements could be made. 47
5.8 The confusion matrix displays a 2x2 matrix (in the task scenario but could be more) described gamma and hadrons events classification. It shows if values were correctly evaluated and the corresponding number of values associated to the scenario. Colors are also displayed to get a quick idea of where most of the data fall. The x-axis displays columns corresponding to predicted gamma and hadron events and the same goes on the y-axis for the ground truth. 48
5.9 Calibration probability graphic displays a curve corresponding to the calibration for a certain probability. It realizes on the probability (on x-axis) and the fractions of positives (percentages of gammas events for a determined sample). A default diagonal is there to display the best calibration possible. The curve can be above or below the diagonal depending on the calibration obtained. 49
5.10 Distribution of events over energy ranges (bins). The histogram helps to get an idea of where events are concentrated. The x-axis shows the energy range for ground truth values with a log scaling and the y-axis displays the number of events corresponding to bins. 52
5.11 Migration matrix are useful to visualize an overview of events energy predictions. By grouping them in bins, the graphic becomes more readable and interpretable. The scales are in a logarithmic shape to comply to the range of prediction values possible. The axis, for ground truth and predictions, are displayed using a TeV unit (energy measure). A color scale has been added to get more insight about the place where most of the events falls. The ideal result should be to have events concentrated around the diagonal describing the perfect predictions. 53
5.12 Energy resolution graphic computes how precise is the model for reconstructing the energy value depending on the energy scale (ground truth energy). The x-axis displays the real data in a logarithmic scale. For visualization purpose, the values are averaged in bins for defined ranges. The y-axis display the spread (energy resolution) in percentage of the predicted data. For each point on the graph, there is filling to show the upper and lower bound for bin (68% containment interval) of the events contained in the bins. 54

5.13 Bias and Standard deviation graphic. Two curves are displayed here : the bias and the standard deviation. There are displayed using bins average their ground truth values and reconstructed energy errors. The x-axis displays the bins (range) on a log scale and the y-axis displays the difference in TeV for the standard deviation and the bias. 55
5.14 Curve showing the evolution of the accuracy with lower thresholds overtime. The x-axis displays the threshold defined arbitrary and the y-axis displays the accuracy at each threshold. 57
5.15 Distribution graphic showing repartition of the events in 2D coordinates system. The x-axis display the azimuth and the y-axis displays the altitude. The axis are in degrees and uses bins to compute the values. The density per bins is shown using a color bar. The cross represents the telescope pointing area and events should normally be detected around them. Depending on the use case, the events can be displayed in different graphic depending on their type. 58
5.16 Angular resolution graphic computes how precise is the model for reconstructing the angular (degree) value depending on the energy scale (ground truth energy). The x-axis displays the ground truth energy of events in a logarithmic scale. For visualization purpose, the values are averaged in bins for defined ranges. The y-axis display the spread (angular resolution) in degree of the predicted data. For each point on the graph, there is filling to show the upper and lower bound for bin (68% containment interval) of the events contained in the bins. 59
5.17 Bias and Standard deviation graphic for the two coordinates predictions. Each of them displays the bias and the standard deviation corresponding to them. The difference between the different curve are in the point type and the colors used. There are displayed using bins for the ground truth energy of events. The x-axis displays the bins (range) on a log scale and the y-axis displays the difference in degrees for the standard deviation and the bias. 60
5.18 Example of table for Performance metrics. It contains a first column stating the model name as a reference. Then the metrics are provided with their values. This table is for deeper comparison of a model to see if model outperforms in a specific metric. 61
5.19 Example of table for Performance metrics. This one is dedicated to a ranking system. It rather shows ranking compared to others models for each metric instead of their value. It helps to quickly understand which model has better performance. 62
5.20 Example of a combined graphic in the comparison of models (Energy Resolution). A single graphic is displayed with, in this case, lines for each model determining the energy resolution over bins of energy. The same scale is used for every line and comparison can be easily done as elements are next to each other. This approach is used for simple displays of graphics. 63
5.21 Example of a subgraph comparison (Migration Matrix). Graphics with complex features aren't combined but rather put next to each other to compare as easily as possible. The plot is exactly the same as in the model report for each of the models. The scale might vary depending on the subgraph displayed. 64

6.1 Accuracy and AUC metrics over the number of samples for each telescope. The main focus is on the SST-1M, the telescope from where the data used in this thesis are retrieved. The accuracy for this telescope is converging a bit upper than 0.8 and the AUC is converging almost at 0.9. 67

6.2 Main concept of the residual block. The idea is that the input is propagated with an identity mapping to skip some layers. The input is then combined to the output of the convolution layers through an Add element-wise operation. It helps to prevent vanishing/exploding gradients and limit the degradation of performance in very deep networks. This type of block is generally combined to other blocks to form a full network. [42] 68
6.3 Default Residual block in CTLearn library. The residual block helps to keep computation efficiency and stability for deeper network. This is the default residual block used to build a ResNet model with the CTLearn library. It's a configurable block that takes advantage of the residual features by passing directly the input through a identity mapping/projection shortcut. The shortcut is represented with this identity map is combined through an addition with the normal convolutional pathway. A ReLU activation is then applied to stabilize the network. The output of the activation function is then set to the next block/layer, finishing the residual block step. Some elements are configurable for the residual block. First, there is the number of filters that can be changed depending on the block. A attention layer can also be added at the end of the convolutional pathway and before the add operation. 69
6.4 Default ResNet Architecture on CTLearn library. The architecture combined residual blocks from the figure 6.3. The idea is to build a very deep model by chaining series of residuals blocks. It allows to train a very precise model without too many parameters and computation time that a model without residuals connections would have. The head part is a classic combination of Dense layers that can be defined separately from the blocks. 70
6.5 Example of VGG16-CNN Network with multiple convolutional layers and max pooling layers. Along the network, the size of the matrix is reduced to have a single output at the end with the prediction. This type of architecture manages to extract a lot of information from an image, from high-level features/patterns to detailed low-level features. [45] 71
6.6 Default CNN implementation for CTLearn library. The model is separated in two parts : the backbone and the head. The backbone contains 4 convolutional layers with important parameters mentioned. Each of theses layers are followed by a MaxPool layer to reduce the dimensions of the matrix. The activation function for each layer is the ReLU activation function [46] and the padding is defined so that the Conv2D layer don't reduce the dimension of the matrix (only MaxPool). The backbone concludes with a GlobalAveragePooling layer, avering the spatial dimensions into one (height, width, channel into one dimension) just like flatten would do. The head has 3 layers that takes data in one dimension with reducing amount of units over the layers to converge to 1 or 2 outputs, depending on the task (e.g. 1 for energy regression and two for particle classification). For the classification, a SoftMax layer is added to finalize the model. For the head, the same activation function as the backbone is used. 72
6.7 Network structure of the template for custom model. The network is separated in two parts : the head and the backbone. The Backbone handels high-level features and start processing the input when the head handles low-level features and computes the predictions of the model. Each of theses elements have a distinct class to define the model layers. 73

6.8 Principle of percentage of data used per epochs. The example displays the data separated into 10 batches. The model is then, given the provided percentage, taking a portion of the data by taking 40% of the batches randomly for a specific epoch. For each epoch, the batches selected are randomly chosen again. The randomness related to one epoch isn't influenced by others epochs results. It means it can have multiple times the same batch accross the epochs. 77
6.9 Repartition of steps within the TrainCTLearnModel tool. This tool has multiple features like the fact it processes the data to be in an adapted format. It also prepares the environment to train the model with callbacks. The model is then loaded inside the tool, compile and trained. 78
6.10 Representation of how the model retrieves the number of layers recursively. It goes inside the nested model / Functional layers to get the layers. It's important to know that some layers aren't taken in account like the input layer or the nested model itself. It has no interest to count an artificial layer. 79
6.11 Process to obtain FLOPs from a model. 1. A dummy input is created and used together with the model to convert it in a graph function. 2. The graph is freezed so it can't be modified with manipulation. 3. TensorFlow Profiler is set up. 4. The graph is passed to the Profiler to retrieve performance metrics. 5. The estimated number of FLOPs is extracted 80
6.12 Pruning process where the model is displayed before and after the pruning optimization. Two levels of pruning are displayed here: the weights pruning represented by the synapses where the connections are removed and the neurons pruning when an entire matrix of weight is removed. [52] 82
6.13 Quantization process where the left is encoded in FP32 (Floating Point 32), with 32 bits for each value, and converted to INT8, encoded in 8 bits. It means the value is going from 4 billions possibilities to only 256, considerably improving matrix multiplication. Two things here can be observed : the reduction of the number of bits and the conversion into integers of the values. The parameters of the quantization aren't provided here, but depending on the settings, the function mapping for the integers values could change and gives a totally different output. [56] 85

10 Appendix