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

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. This distinction can be seen in the figure

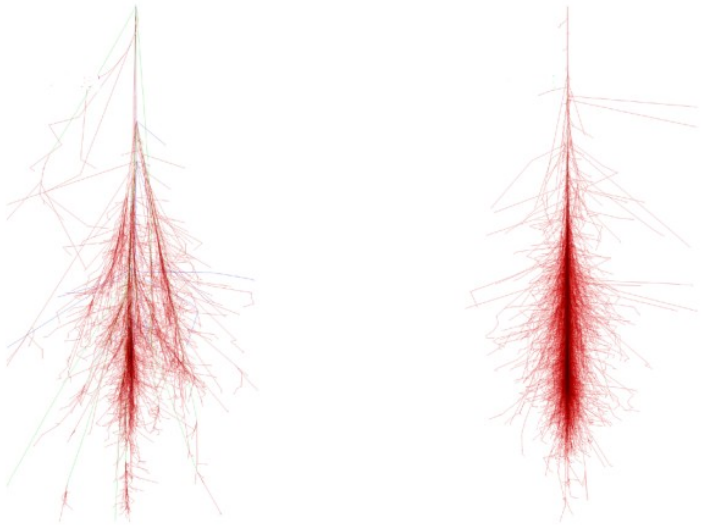

Figure 2.1: Two different type of air shower with a different particle at the origin of it. On the left, it's a shower originated from an hadron and on the right, a shower originated from a gamma.

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. Theses three tasks will be described below. 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.

Some fundamental concepts are described below to fully understand the thesis area of work.

2.1 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.

2.1.1 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.

2.1.2 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.

2.1.3 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.

2.2 Data

2.3 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.

State of the art

MLOps approach

Models

Observations/Results

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 State of Art

This section contains all the information to know before deep-diving into the project. It will include technologies expected to be seen in the project.

Multiple types of technologies will be referenced. All technologies seen in this chapter have a purpose in the project.

- .

A chapter is dedicated here to justify **technologies choices** in the project. The choice is made based on the previously described technology (3.1).

3.0.1 Angular Resolution

This metric is used to evaluate direction regression models. It works in a similar way as the energy resolution. It measures how precisely the direction is reconstructed for a particle. It also measures the spread between the real direction and the predicted one. It measures how well a model can localize gamma points in the sky and how much nearby sources influences the output with blurring. This blurring can become stronger depending on the angular resolution used to get the data.

It's the 68% containment radius of well reconstructed events in an energy bin. Another name for this is Point Spread Function (PSF) and is associated to non-gaussian distributions.

This can be visualized with the figure .

// ADD IMAGE

3.0.2 Differential Sensitivity

This metric is one of the most important when talking about model performance and classification. It measures the minimum gamma-ray flux the model can detect for a 5ÏČ point-like source. It can also be defined as a measure of the sensibility of the model.

The metric is represented in a graph as a curve. The lower is the curve, the better are the model results.

// ADD IMAGE

3.1 Technological choices

3.2 Ideas

3.2.1 Model Selection model

3.2.2 Extract params and provide to CNN as additional input

Extract top params like for the random forest and provide it as input for CNN (through layers or directly combined with image)

3.2.3 Model Selection model

3.3 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. Each component is described in this section combined to a graph for an overview of the architecture with interactions between components.

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.

The second architecture extends the first one by using bigger infrastructure and an increased amout of data.

//////////////TTTTTTTTOOOOOOOO CCCCCCCCCCOOOOOOOOMMMMMMMMPPPPPPPLLLLLLLLEEE

3.3.1 Baobab

Baobab is on of the three clusters provided by the University of Geneva to do high performance computing and calculation [1]. 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 [2].

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 (3.3.1.1).

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 (3.3.1.2).

Other than computational resources, the cluster also provide storage for good amount of data. The spaces used during the thesis are detailed below (3.3.1.3).

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 [3] with the SSH key.

3.3.1.1 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 [4], 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 [5].

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 3.1. 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 3.1: 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')"
```

3.3.1.2 SLURM

SLURM (Simple Linux Utility for Resource Management) [6] 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 [6]. 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 3.2 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 3.2: 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 3.2 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 7.3.1.

```
--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 ...
'
```

3.3.1.3 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 [7] 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.

3.3.2 Calculus

Calculus is a server where data concerning SST1M Cherenkov telescopes. A 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.

3.4 Tools

Building and analyzing models is an essential aspect of the thesis. Therefore, this task should be the most comfortable to do as managing models can be complex and time-consuming. To solve this issue, multiple tools will be used or put in place in order to reduce the amount of time or the computational resources required by models handling. The process is to identify the best model for each task described earlier. Tools created or used will help to have an automated and reproducible environment for the models. Evaluation is also an important aspect the tools will enhance.

3.4.1 Task scripts

As described in the architecture of the project, in local, Jupyter notebooks are used to test technologies and new features. Theses notebooks are messy due to the exploration purpose they have. Exploring ways to optimize the task and different tools to do it is good but when going to an

production environment, some order needs to be made. There is also a need to have the simplest and clearest documented code to do a specific task in case of takeover or deeper understanding needs of the project.

Therefore, the main tasks like the training, the testing and the evaluation of models were separated in Python script with their bash execution script handling the SLURM environment (chapter 3.3.1.2) interaction. These scripts handles only their corresponding task and execute it clearly. The bash scripts describes the resources required to execute the task, some configuration details like the type of module required or some additional environment settings that needs to be provided and finally the command to run the python script. Details can be found about some details to handle the data in the chapter 3.3.1.2.

Separating tasks in different scripts comes with advantages like the reproducibility of the experiences conducted. Due to the restrictions on SLURM, executing multiple tasks separately or re-running failed tasks doesn't require to re-run the whole process.

Here is the list of current tasks possible to execute on the baobab server (main ones) :

- **Training** : Process where data is loaded and used to train a model
- **Testing** : Test the model on the specified data
- **Model Report** : Generate a report with metrics and graphics to evaluate the model performances
- **Compare model** : Take all models from a specific task and compare the results between the model to get an overview of a potential better model

Others less important scripts are available and are used mostly to handle the data and adapt it to an adequate state to be used by model generation. There aren't directly related to the thesis main goals, but ensure tools for future deployment of data.

- **Convert Data** : Convert data from compressed format (.gz) to HDF5 file (.h5)
- **Count Events** : Count the number of events stored in a folder (reads all files)
- **Merge Files** : Tool to reduce amount of files by merging them together
- **Reduce Amount of Data** : Strip from the files unwanted sections like waveforms to gain space and save computational resources

The following scripts don't have an entire chapter dedicated, but there are anyway provided in the appendix (10).

3.4.2 Report Generation

One of the tools used to accelerate the process of comparing models is the model report. A script is prepared to provide, in a good visual looking, all information to determine the performance of the models.

3.4.3 Comparison of models

- Show examples of graphics handling - Show handling of metrics

3.4.4 Configuration file

- Enhance reproducibility without changing the code (have multiple models handled) - Describe elements in the configuration file (general description for each section)

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, objectives, etc. of the data.

There is also a part dedicated to how the data is handled and used by models to train and test them. Optimization of the data is required to comply to the production environment where the models are built.

4.1 What do we want ?

Doing Cherenkov Imaging of gamma-ray and hadron has multiple purposes. The data helps us to identify three main values in order to better understand the domain of shower imaging. Theses three values are the following :

- **Particle type** : Type of element at the origin of the shower
- **Energy** : Intensity of energy detected
- **Direction** : The original direction of the particle before the start of the shower

Each of theses helps to reconstruct the original event and understand thoroughly details about it.

4.1.1 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 4.1).

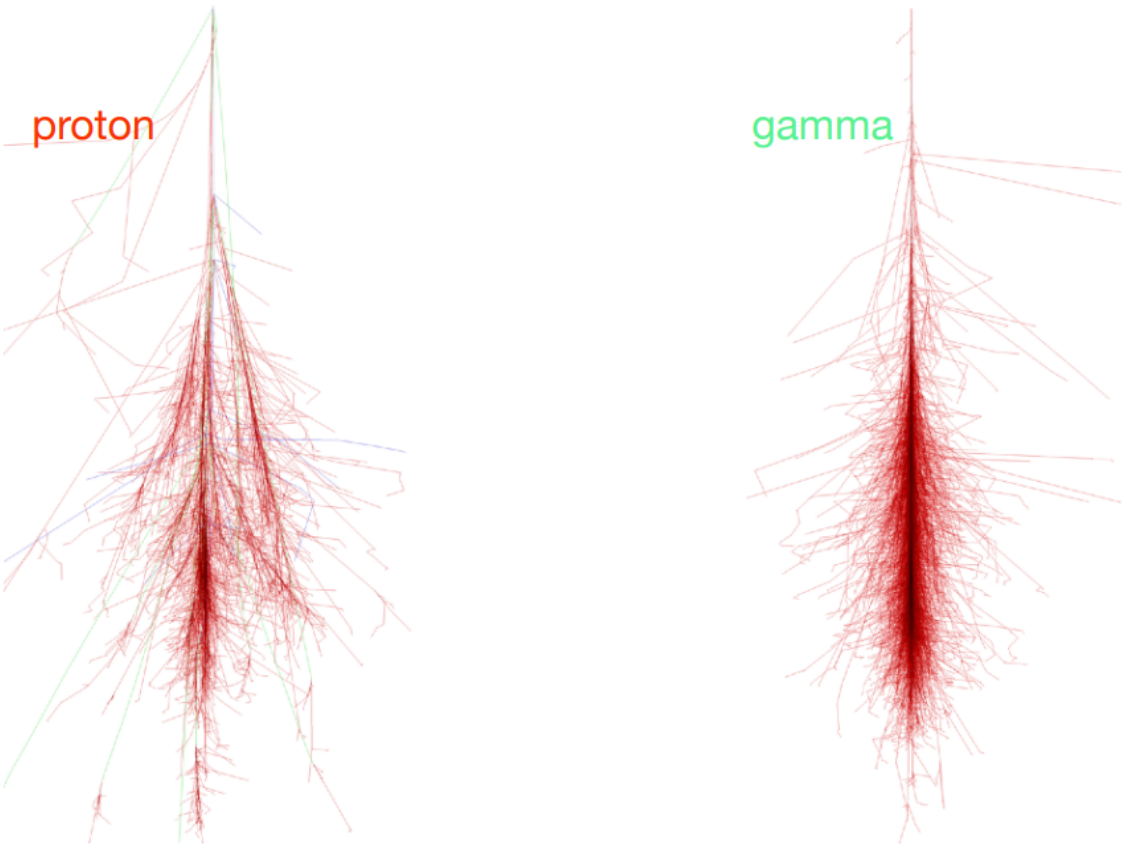

Figure 4.1: Shower distribution for two types of particle

4.1.2 Energy

4.1.3 Direction

4.2 Format

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$_t$elarray", $one of the simulation tools. As it's said, the thesis works with simulated data and not real data. T$

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 (4.3.2) 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.

4.2.1 Generation of data

4.2.2 Content of HDF5 file

4.3 Compliance with the server

The data available on calculus (3.3.2) cannot directly been used

4.3.1 Migration to the cluster

4.3.2 Conversion of data

4.3.3 New storing space

4.3.4 Merging files

4.3.5 Keep only wanted data

4.4 Type of data

The project is wide and needs to work on different level to trigger, analyze and generate new data and information to get insights. The data used for this project is also core elements for others tasks related to gamma-rays. Therefore, some projects needs to work with different levels of processing due to restrictions and purposes of the task they need to proceed.

4.4.1 Waveform

4.4.2 Frame

R0

R0 data corresponds to the raw data. This is the data

R1

DL1

DL2

DL3

Monte Carlo

Data Structure

. âŤIJâŤĂâŤĂ configuration/ âŤĆ âŤIJâŤĂâŤĂ instrument/ âŤĆ âŤĆ âŤIJâŤĂâŤĂ subarray/ âŤĆ âŤĆ âŤĆ âŤŤâŤĂâŤĂ layout âŤĆ âŤĆ âŤŤâŤĂâŤĂ

telescope/ âŤĆ âŤĆ âŤIJâŤĂâŤĂ camera/ âŤĆ âŤĆ âŤĆ âŤIJâŤĂâŤĂ geometry$_0$*geometry*$_1$*readout*$_0$*readout*$_1$*opticsobservation*/*observation*$_b$*lockscheduling*$_b$*locktelescope*/*p*

file to work with

predictions des types : dl2/event/subarray/classification/CTLearn prediction de l'Ãĺnergie : dl2/event/subarray/energy/CTLearn groundtruth : simulation/event/subarray/shower show distribution des events : simulation/service/shower$_d$*istribution*

5 MLOps approach

5.1 Task Breakdown

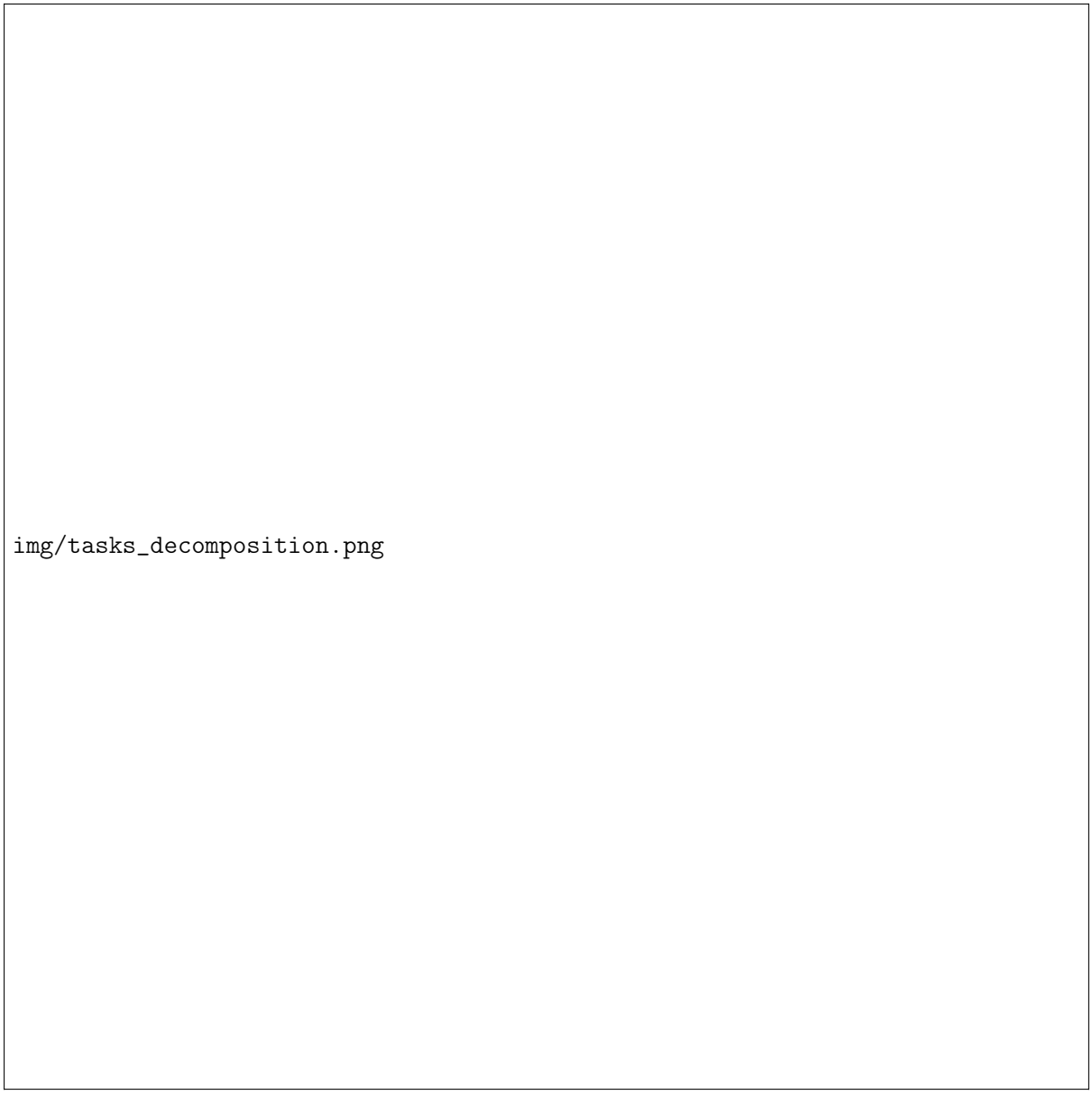

Figure 5.1: Task decomposition

5.2 Configuration files

5.3 Report Generation

This section describes the different elements that can be found in the report used to evaluate a specific model. The report could includes the different elements described below depending on the task and how the generation is handled. The generation of the report is done by using a python file structured to generate all the information below. The python file is available in the appendix.

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. 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 [8].

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 5.2 displays an example of training loss graphic.

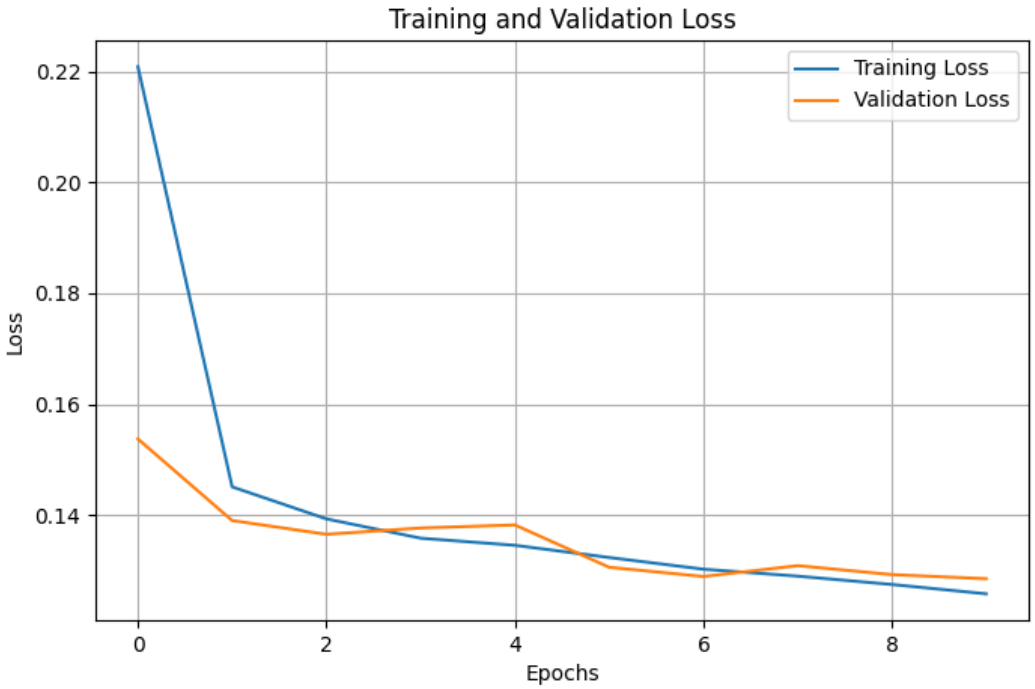

Figure 5.2: Graphic containing the training loss and the validation loss obtained through epochs. This helps to understand possible problems during training like underfitting or overfitting. This could also indicates that the current number of epochs isn't enough to converge.

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 5.3 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 5.3: 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 (2.1.1). 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 5.4 is a clear example of a ROC curve representation with a legend displaying the AUC value.

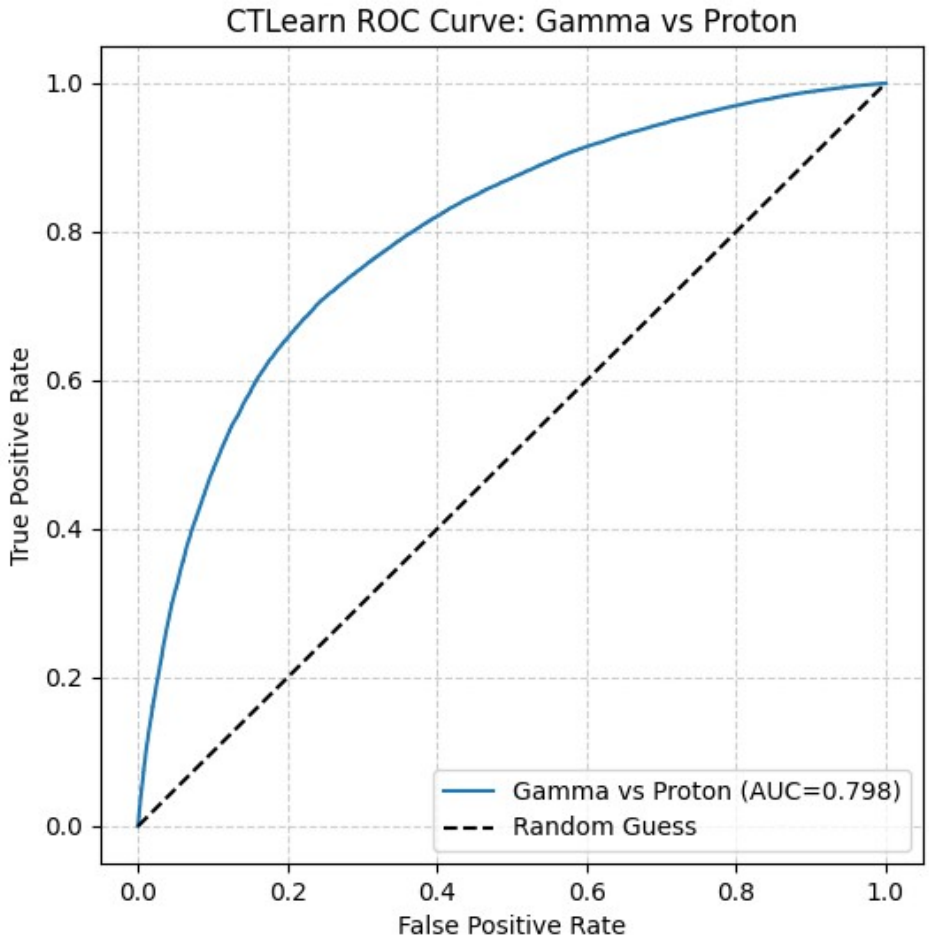

Figure 5.4: ROC Curve graphic displayed using TPR and FPR. There is also a diagonal displaying random guess values. The legend display the model label and the AUC to get a better overview of the model performance on the curve.

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

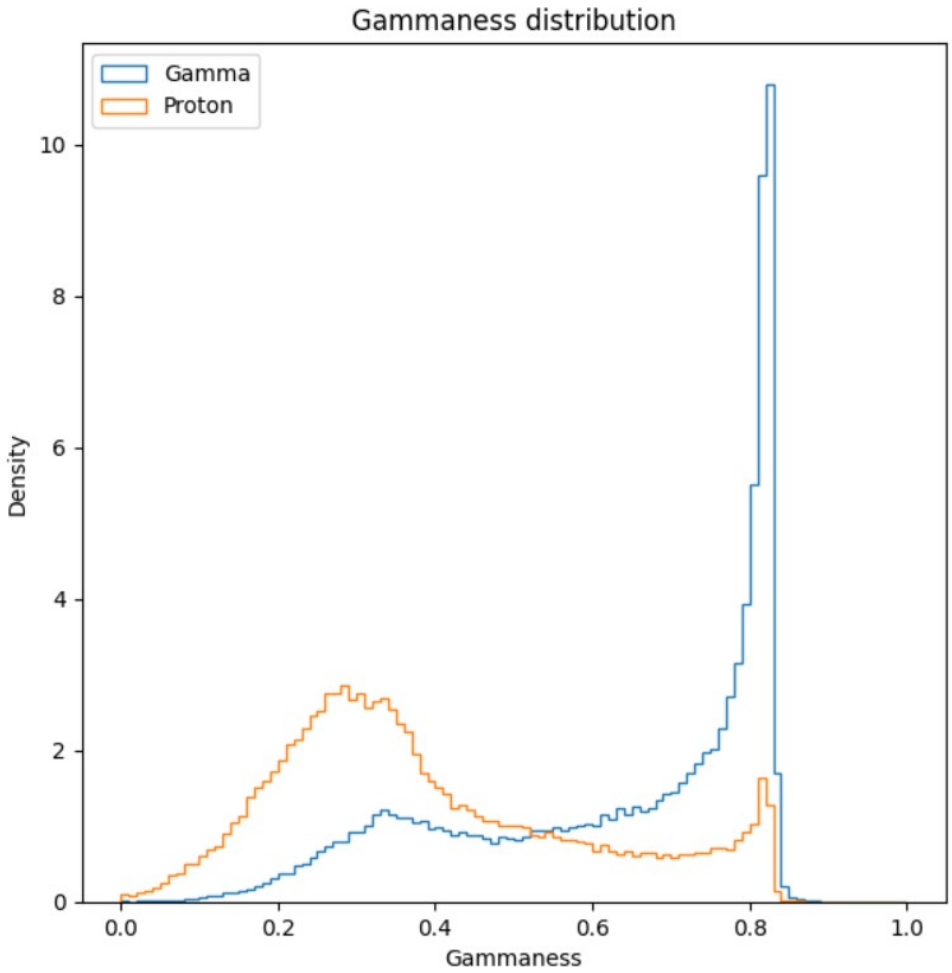

Figure 5.5: Gammaness distribution graphic where the distribution for each type of particle can be seen. The axis used are the gammaness and the density of events contained at each level of gammaness. In the example, the separation between gamma and hadron can be seen. It's not clear which means some 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).

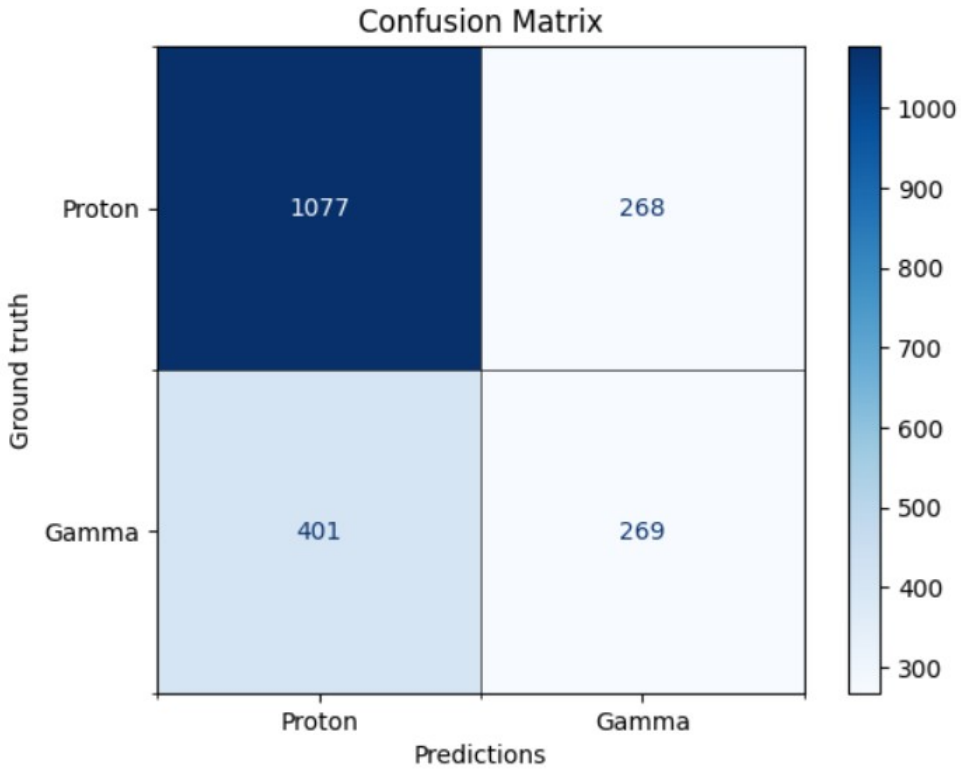

Figure 5.6: The confusion matrix displays a 2x2 matrix (in the task scenario but could be more) described gamma and hadrons events classification. It shows if values were correctly evaluated and the corresponding number of values associated to the scenario. Colors are also displayed to get a quick idea 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.

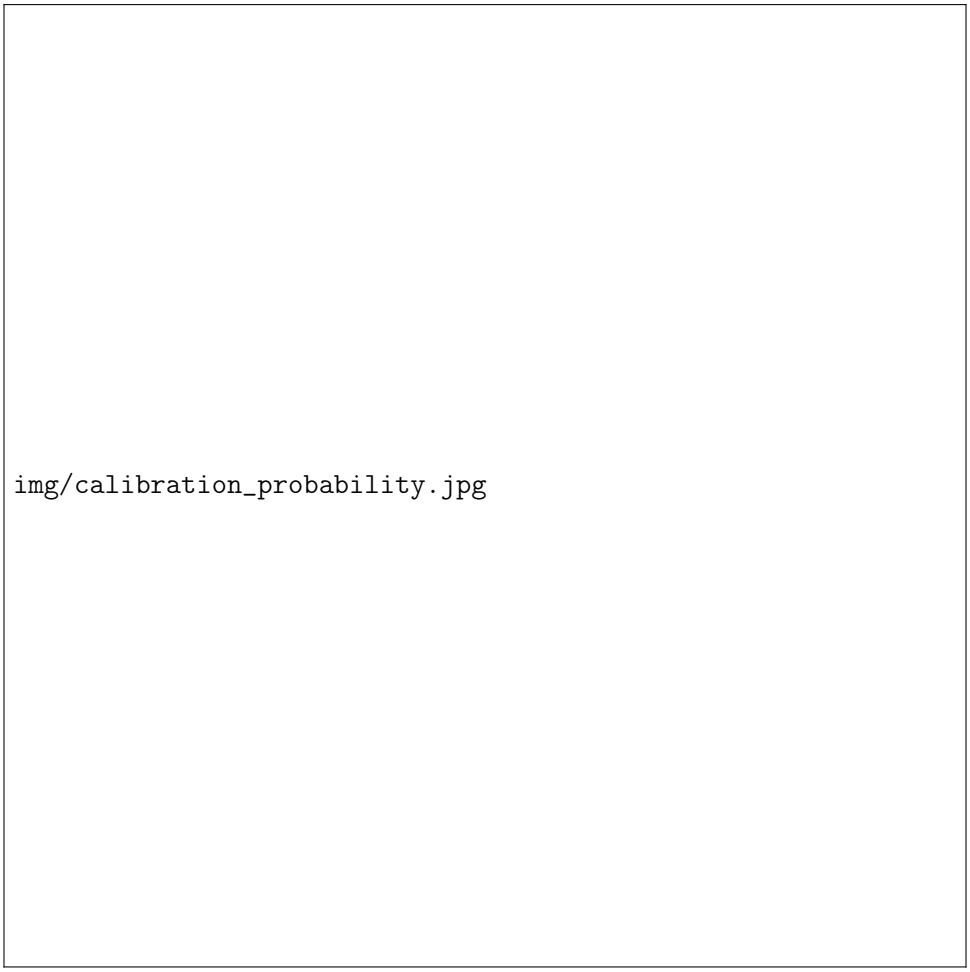

Figure 5.7: Calibration probability graphic displays a curve corresponding to the calibration for a certain probability. It realizes on the probability (on x-axis) and the fractions of positives (percentages of gammas events for a determined sample). A default diagonal is there to display the best calibration 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 (2.1.2). 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 5.8 shows the content of this graphic.

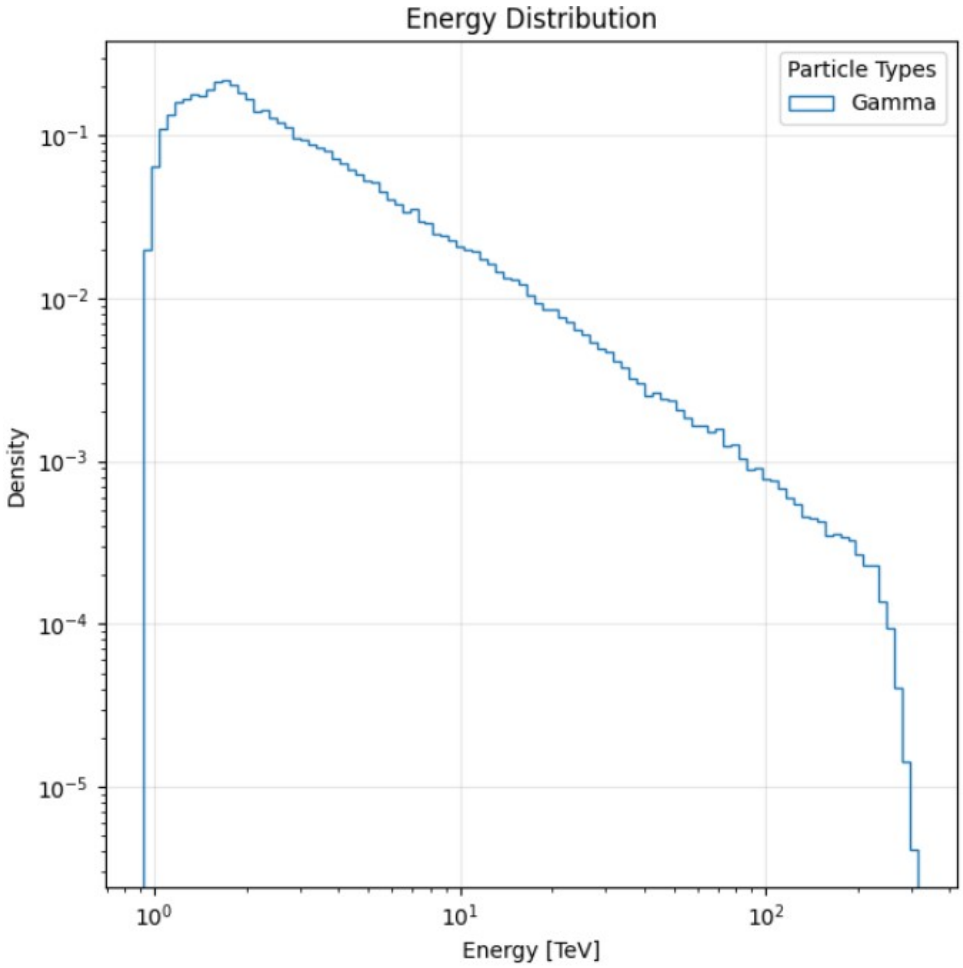

Figure 5.8: Distribution of events over energy ranges (bins). The histogram helps to get an idea of where events are concentrated. The x-axis shows the energy range for ground truth values with a log scaling and the y-axis displays the number of events corresponding to bins.

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 5.9. 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.

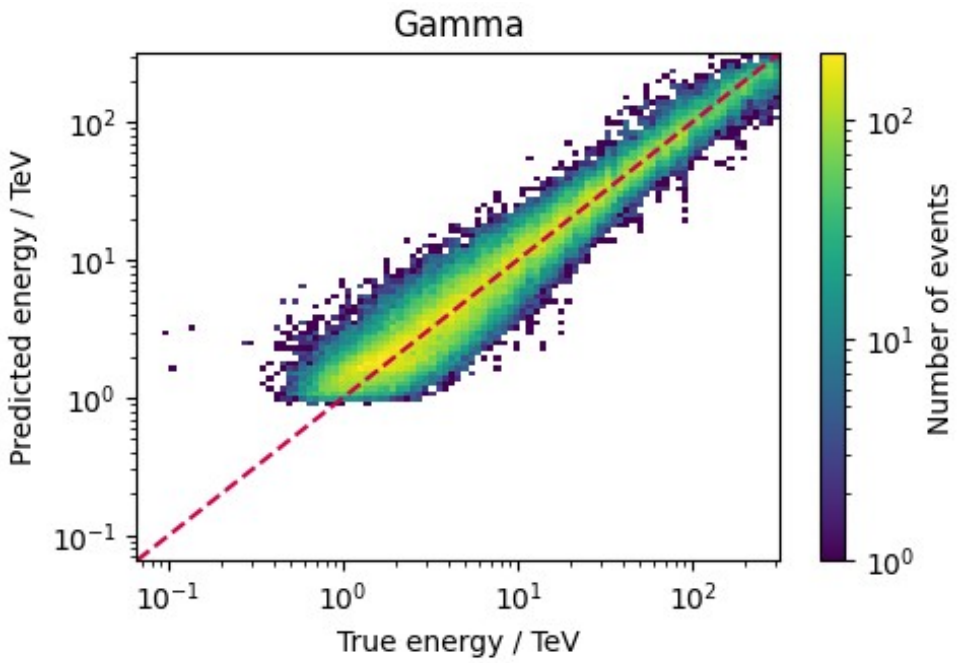

Figure 5.9: Migration matrix are useful to visualize an overview of events energy predictions. By grouping them in bins, the graphic becomes more readable and interpretable. The scales are in a logarithmic shape to comply to the range of prediction values possible. The axis, for ground truth and predictions, 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.

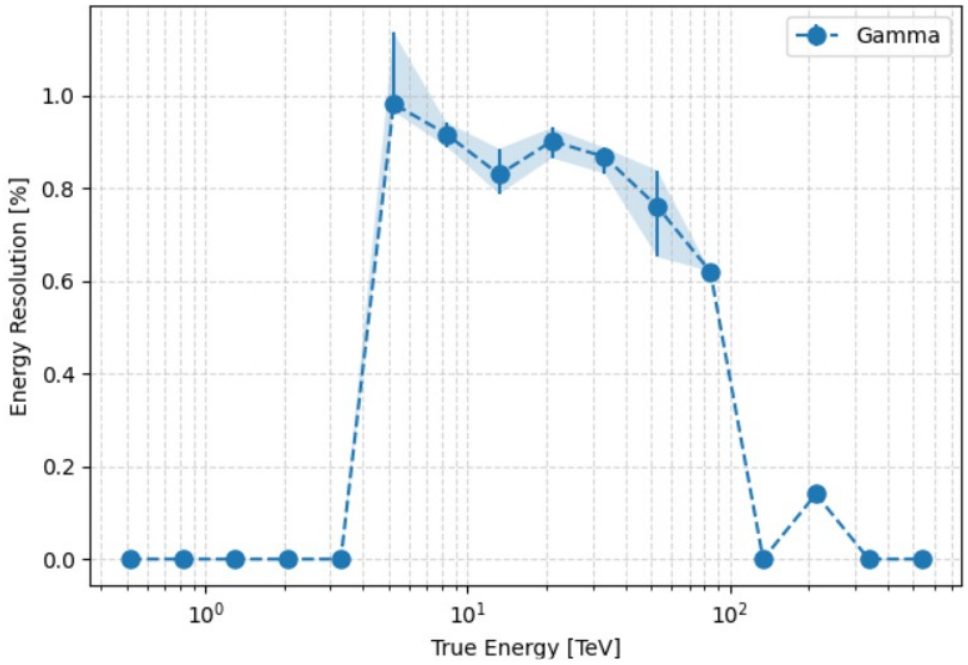

Figure 5.10: Energy resolution graphic computes how precise is the model for reconstructing the energy value depending on the energy scale (ground truth energy). The x-axis displays the real data in a logarithmic scale. For visualization purpose, the values are averaged in bins for defined ranges. The 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 5.11.

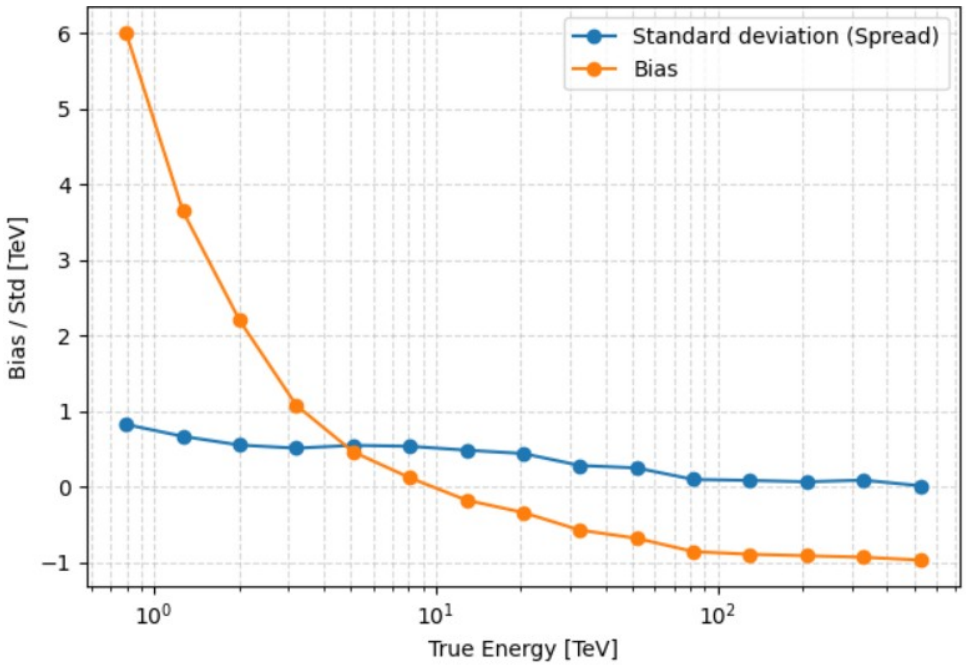

Figure 5.11: Bias and Standard deviation graphic. Two curves are displayed here : the bias and the standard deviation. There are displayed using bins average their ground truth values and reconstructed energy errors. The x-axis displays the bins (range) on a log scale and the y-axis displays the difference 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 (2.1.3). 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.

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.

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

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 5.12 shows an example of curve that can be obtained.

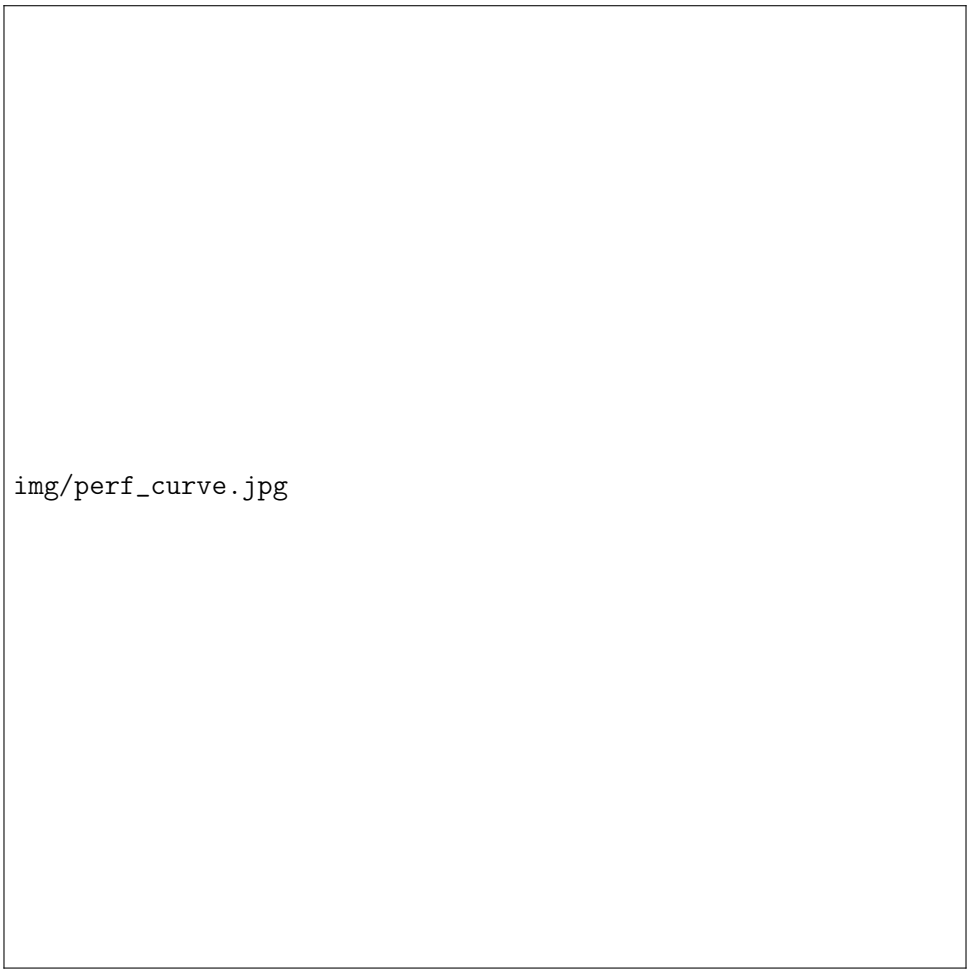

Figure 5.12: Curve showing the evolution of the accuracy with lower thresholds overtime. The x-axis displays the threshold defined arbitrary and the y-axis displays the accuracy at each threshold.

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 5.13) 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.

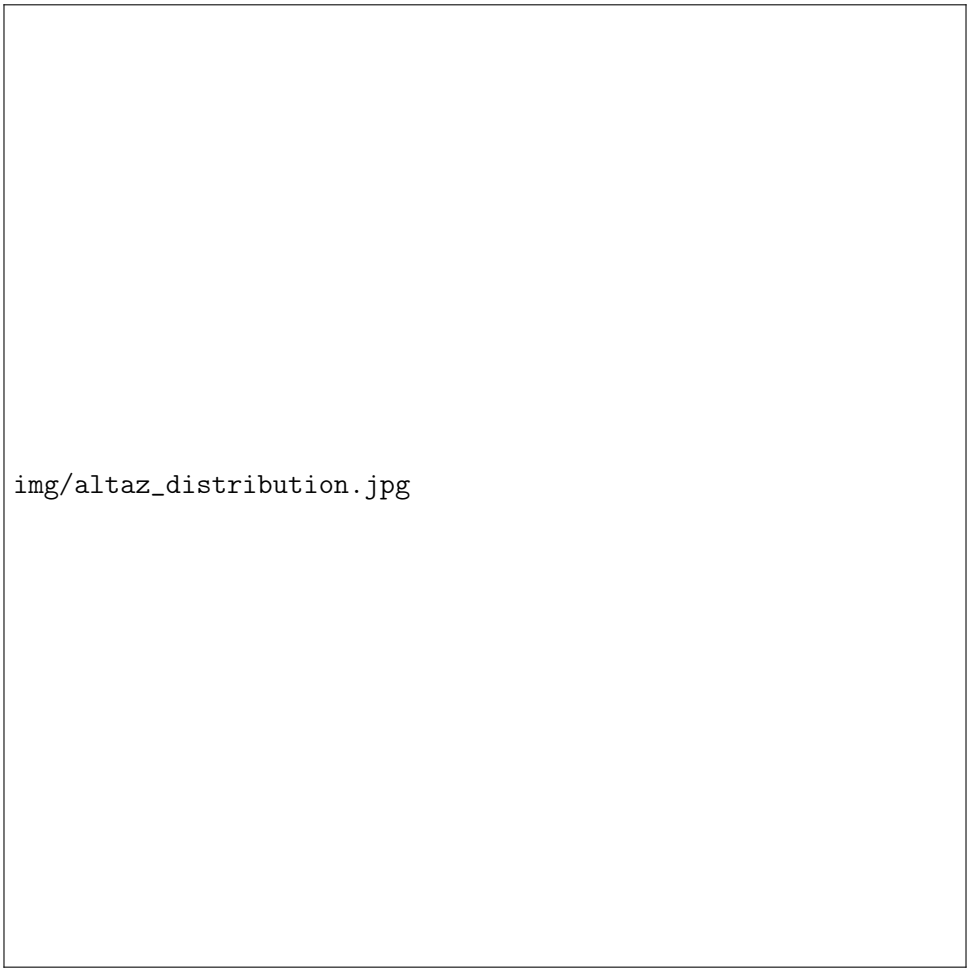

Figure 5.13: Distribution graphic showing repartition of the events in 2D coordinates system. The x-axis display the azimuth and the y-axis displays the altitude. The axis are in degrees and uses bins to compute the values. The density per bins is shown using a color bar. The cross represents the telescope 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 5.14.

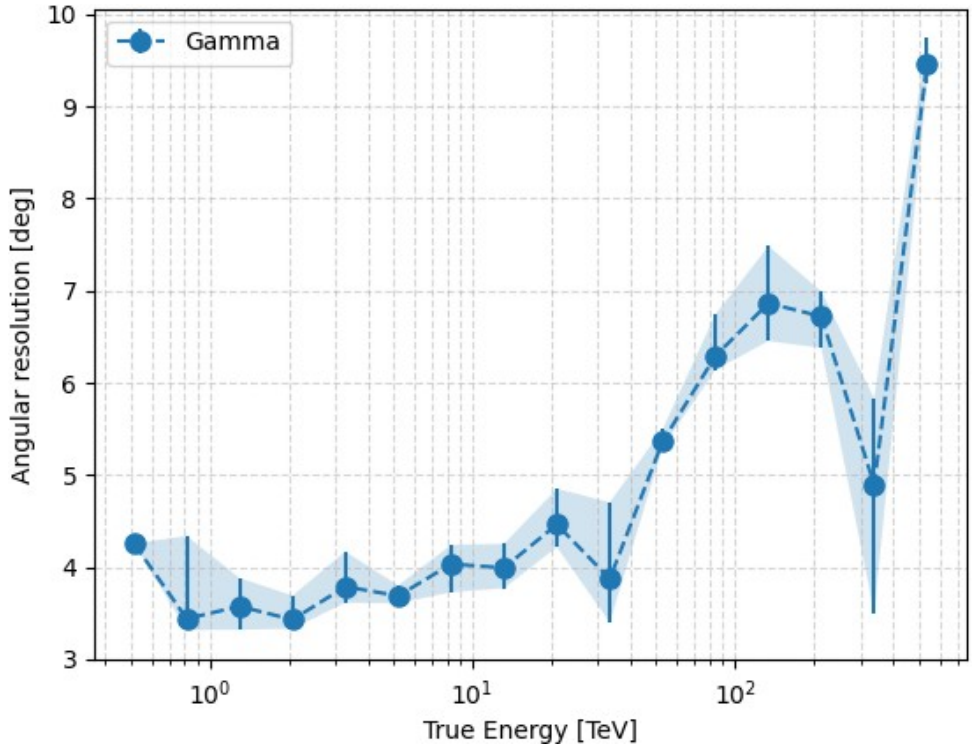

Figure 5.14: Angular resolution graphic computes how precise is the model for reconstructing the angular (degree) value depending on the energy scale (ground truth energy). The x-axis displays the ground truth energy of events in a logarithmic scale. For visualization purpose, the values are averaged 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 5.15 shows the different curves for this graphic.

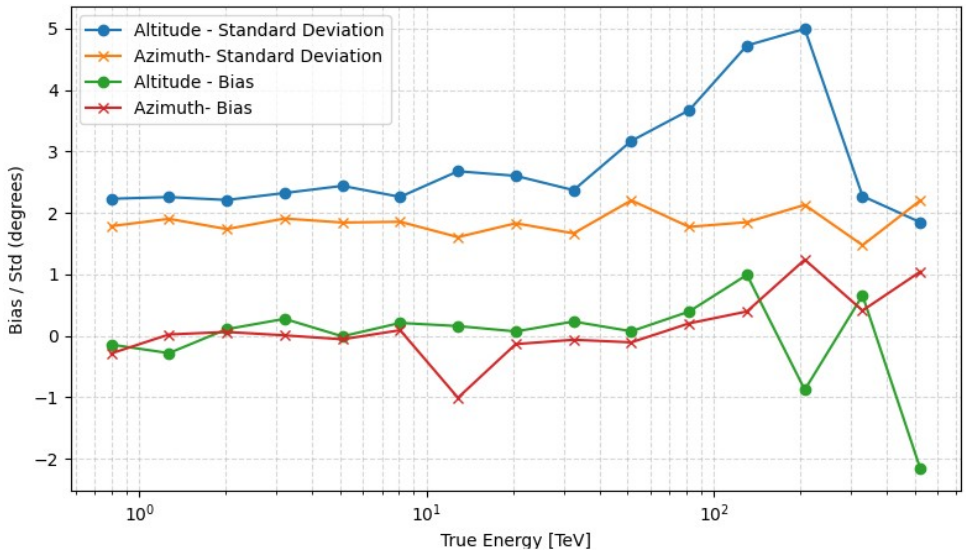

Figure 5.15: Bias and Standard deviation graphic for the two coordinates predictions. Each of them displays the bias and the standard deviation corresponding to them. The difference between the different curve are in the point type and the colors used. There are displayed using bins for the ground truth 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

6 Models

reference talk + work done with them + refer to issues (time deployement

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 [9], 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.

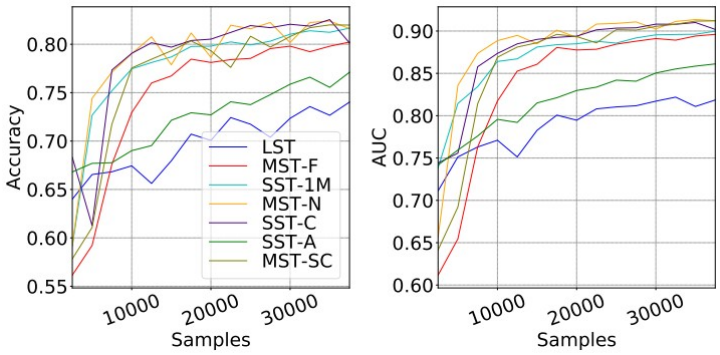

Figure 6.1: Accuracy and AUC metrics over the number of samples for each telescope. The main focus is on the SST-1M, the telescope from where the data used in this thesis are retrieved. The accuracy for this telescope is converging a bit upper than 0.8 and the AUC is converging almost at 0.9.

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.

6.2.2 CNN

The CNN model 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

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 6.2.

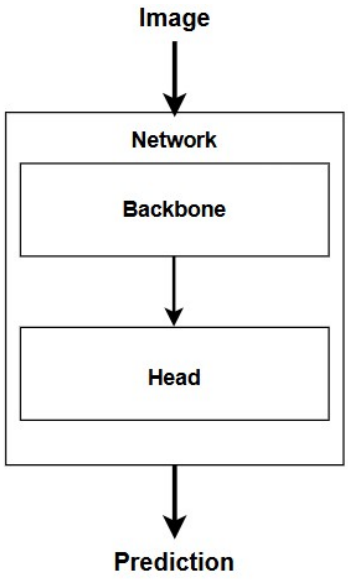

Figure 6.2: Network structure of the template for custom model. The network is separated in two parts : the head and the backbone. The Backbone handels high-level features and start processing the input when the head handles low-level features and computes the predictions of the model. Each of theses 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 5.1.

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 [10]. 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 6.3.

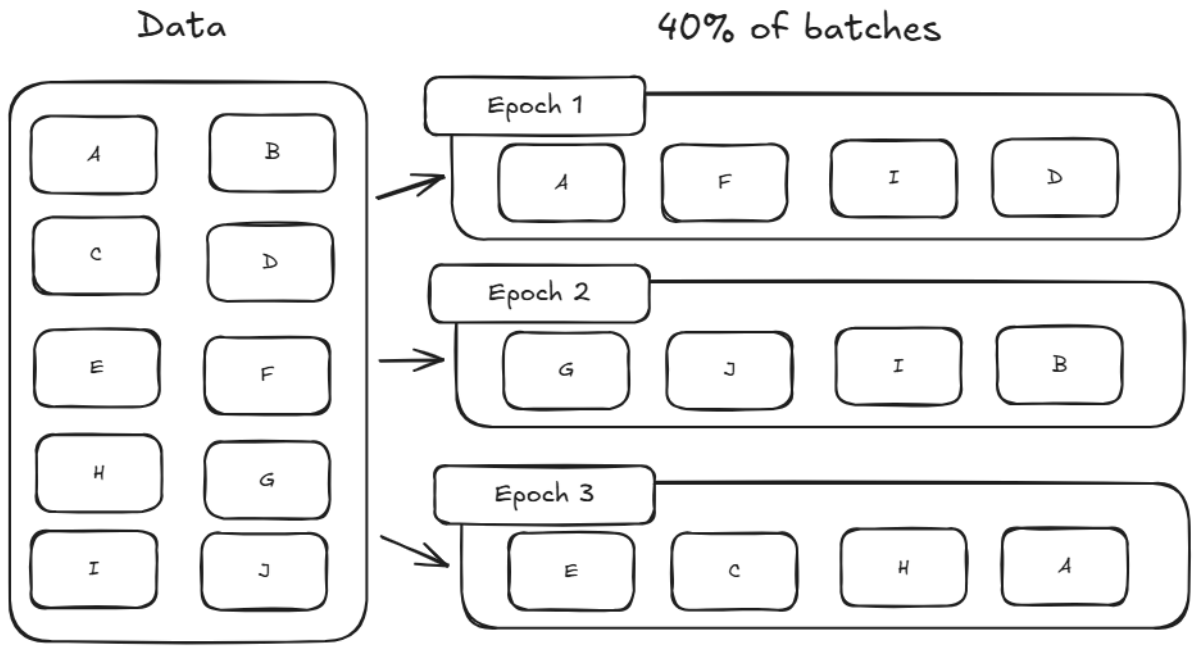

Figure 6.3: Principle of percentage of data used per epochs. The example displays the data separated into 10 batches. The model is then, given the provided percentage, taking a portion of the data by taking 40% of the batches randomly for a specific epoch. For each epoch, the batches selected are randomly 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 (7.3.2).

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 [11] 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 represents the tasks the tool is handling

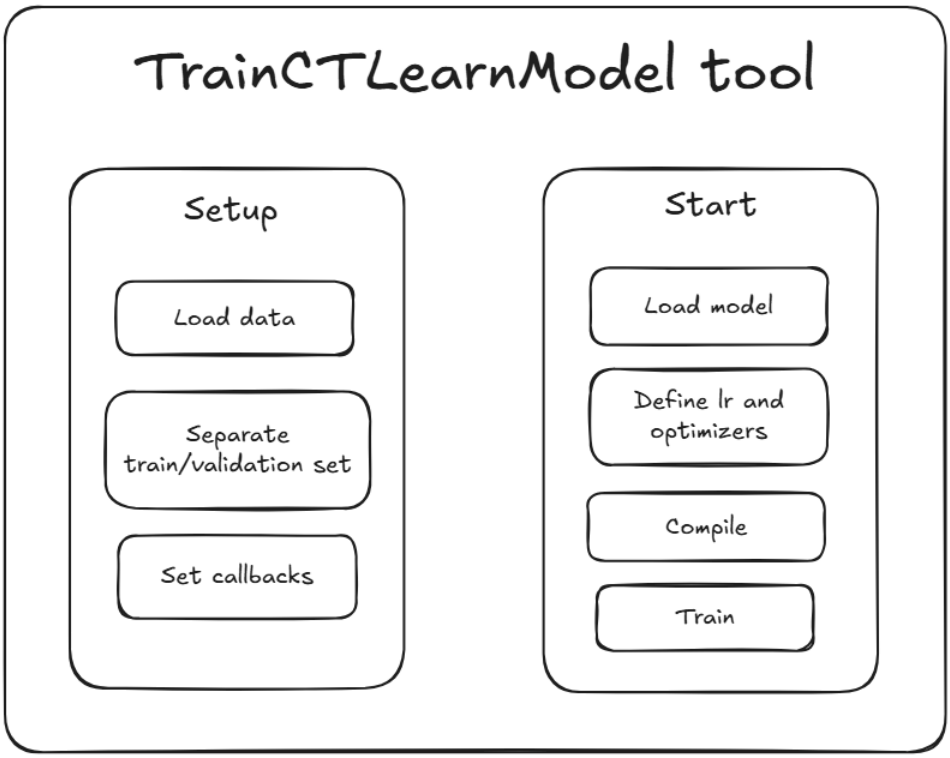

Figure 6.4

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,

6.3.3 Testing

parameters for testing

not same approach for mono and stereo mode

metrics

6.4 Model Optimization

talk about fpga and trigger objective

6.4.1 Pruning

6.4.2 Quantization

6.5 New Models

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 Personal conclusion

7.3 Challenges

7.3.1 Link GPU to Tensorflow

7.3.2 Accelerate training speed

7.4 Future improvements

8 Declaration of honor

Bibliography

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

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

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

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

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

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

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

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

[9] 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].

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

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

List of Figures

10 Appendix