MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so

Exploring Autoencoders for Anomaly Detection in Cherenkov Telescope Data

Master of Science HES SO in Engineering

Profil/Orientation Data science

Date of publication TODO

Produced by

Author Laetitia Guidetti

Under the supervision of Andres Upegui Posada

In collaboration with Matthieu Heller, University of Geneva

Declaration of Authenticity

I hereby affirm that this assignment is my own written work and that I have used no other sources or aids other than those indicated. All passages that have been quoted from publications or paraphrased from these sources are indicated as such. Additionally, any assistance from artificial intelligence tools has been appropriately acknowledged and documented.

Artificial intelligence tools were used for linguistic correction and the reformulation of certain passages to enhance clarity. All AI-assisted suggestions were carefully reviewed and, where necessary, modified to ensure the accuracy and integrity of the content.

Name: Laetitia Guidetti
Date: February 1, 2026

Acknowledgments

I would like to thank my supervisor, Andres Upegui Posada, for his guidance throughout this project. His feedback, careful review of my work, and helpful suggestions greatly supported the development of this thesis.

I also thank Matthieu Heller for his support and for taking the time to answer my questions throughout the project. His guidance and explanations helped me better understand the concepts behind the work and navigate its challenges.

I am grateful to Jakub Kvapil and Laurent Gantel for their support throughout this project. Their explanations, insights, and advice on both technical and conceptual aspects were very helpful and contributed to shaping different stages of the work.

Finally, I would like to thank everyone who contributed, directly or indirectly, to the completion of this thesis.

Abstract / Résumé

Abstract (English)

Gamma-ray astronomy studies the most energetic phenomena in the Universe by detecting the particle showers produced by gamma rays. However, the majority of detected events originate from hadronic cosmic rays, making it necessary to separate gamma-ray and hadronic events. This work explores the use of autoencoders for this task, leveraging their ability to learn compact data representations without explicit supervision.

The primary objective is to evaluate the performance of autoencoders for gamma/hadron separation in IACT data and to compare them to existing supervised methods. A secondary objective is to examine the ability of autoencoders to compress IACT data without significant loss of information relevant to analysis tasks.

To achieve these objectives, several autoencoder architectures were developed and trained on simulated images of atmospheric showers. Given the particular hexagonal structure of IACT images, the models were adapted to handle this specific geometry. Different image preprocessing techniques were also explored, such as logarithmic normalization, using masks to filter out irrelevant pixels, and applying quality cuts.

The results show that autoencoders can be used for gamma/hadron separation, although the performance is lower than that of existing supervised methods. However, autoencoders also demonstrate their ability to achieve strong compression (about a factor of 20) of images while preserving essential information for supervised tasks of proton/gamma classification and energy regression. This work highlights the potential of autoencoders as complementary tools for IACT data analysis, particularly in terms of image compression and data preparation for supervised methods.

Résumé (Français)

L'astronomie gamma étudie les phénomènes les plus énergétiques de l'Univers en détectant les gerbes de particules produites par les rayons gamma. Cependant, la majorité des événements détectés provient de rayons cosmiques hadroniques, ce qui rend nécessaire la séparation entre événements gamma et hadroniques. Ce travail explore l'utilisation des autoencodeurs pour cette tâche, en s'appuyant sur leur capacité à apprendre des représentations compactes des données sans supervision explicite.

L'objectif principal est d'évaluer la performance des autoencodeurs pour la séparation gamma/hadronique dans les données IACT, en les comparant aux méthodes supervisées existantes. Un objectif secondaire est d'examiner la capacité des autoencodeurs à compresser les données IACT sans perte significative d'informations pertinentes pour les tâches d'analyse.

Pour atteindre ces objectifs, plusieurs architectures d'autoencodeurs ont été développées et entraînées sur des images simulées de gerbes atmosphériques. Étant donné la structure hexagonale particulière des images IACT, les modèles ont été adaptés pour traiter cette géométrie spécifique. Différentes techniques de prétraitement des images ont également été explorées, telles que la normalisation logarithmique, l'utilisation de masques pour filtrer les pixels non pertinents et l'application de coupes de qualité.

Les résultats montrent que les autoencodeurs peuvent être utilisés pour la séparation gamma/hadronique, bien que les performances soient inférieures à celles des méthodes supervisées existantes. Cependant, les autoencodeurs démontrent également leur capacité à réaliser une forte compression (environ un facteur 20) des images tout en préservant les informations essentielles pour les tâches supervisées de classification proton/gamma et de régression de l'énergie. Ce travail met en lumière le potentiel des autoencodeurs comme outils complémentaires pour l'analyse des données IACT, en particulier en termes de compression d'images et de préparation des données pour les méthodes supervisées.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Gamma-ray astronomy plays a crucial role in studying the most energetic astrophysical phenomena in the universe, such as gamma-ray bursts, black holes, and pulsars. These sources emit gamma photons at extremely high energies that cannot be detected directly from the Earth's surface due to their absorption by the Earth's atmosphere. However, when these gamma photons interact with the atmosphere, they produce showers of secondary particles that travel faster than the speed of light in air, emitting Cherenkov light, a brief and intense luminous phenomenon. Imaging Atmospheric Cherenkov Techniques (IACTs) are designed to detect this light and reconstruct the properties of the primary particle, such as its energy and arrival direction.

However, it is not only gamma photons that interact with the Earth's atmosphere. In the vast majority of cases, it is hadronic cosmic rays (protons, nucleons, etc.) that collide with the atmosphere, producing hadronic showers. They also generate secondary particles and Cherenkov light detected by IACTs. Therefore, a crucial step in analyzing IACT data is the separation between gamma and hadronic events, in order to identify astrophysically interesting gamma signals among the hadronic background noise.

1.2 Objectives

In this context, the main objective of this thesis is to develop and evaluate an autoencoder-based method for gamma/hadron separation in IACT telescope data. Currently, the methods used for this task mainly rely on supervised classifiers, such as random forests and convolutional neural networks.

Autoencoders are capable of learning compact, low-dimensional representations of data without explicit supervision, making them particularly suitable for anomaly detection and efficient data compression. This work proposes to train an autoencoder solely on hadronic events, assuming that it will learn to effectively reconstruct these events but will struggle to reconstruct gamma events, which are considered anomalies in this context.

The goal is to evaluate the performance of this approach for gamma/hadron separation, comparing the results obtained with those of existing supervised methods. A secondary goal is to investigate the ability of autoencoders to compress IACT data without significant loss of information relevant to analysis tasks. This work aims to explore the potential of autoencoders to improve the detection of gamma signals in IACT data while enabling efficient data compression, thereby contributing to the advancement of analysis techniques in gamma-ray astronomy.

Chapter 2

Theoretical and Physical Framework

In order to address the objectives presented in the previous chapter, it is essential to have a solid understanding of the physical concepts underlying this work. The objective of this chapter is to provide the necessary background in gamma astronomy. It will cover the basics of gamma-ray astronomy, air showers, the IACT technique, and a description of the telescope used for this project.

2.1 Gamma-ray Astronomy

Gamma-ray astronomy is a branch of astrophysics that studies gamma rays emitted by highly energetic astrophysical sources. These sources are associated with some of the most energetic and extreme physical processes in the universe, such as supernovae, black holes, pulsars, and gamma-ray bursts. The study of gamma rays helps to understand these high-energy phenomena, as well as the nature of dark matter and the origin of cosmic rays [1].

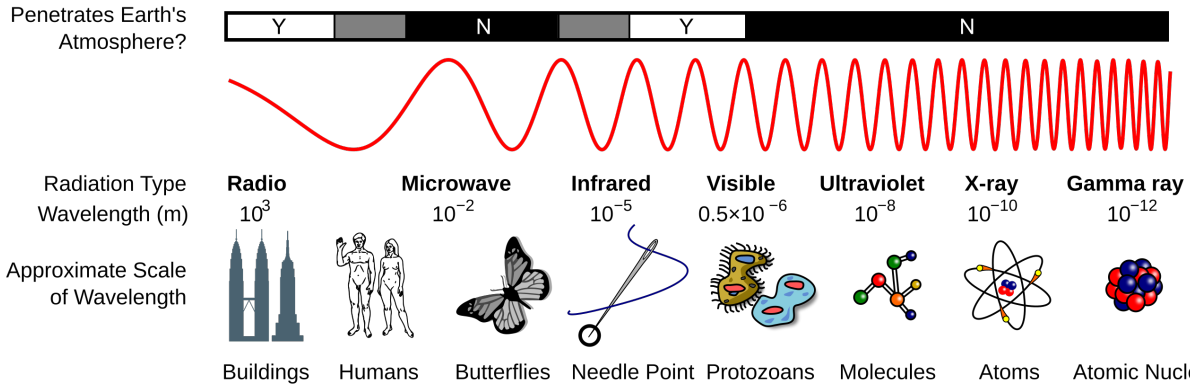


Figure 2.1: Diagram of the EMS system (source: Wikipedia [2]).

In the electromagnetic spectrum shown in Figure 2.1, gamma rays are located at the highest energy end, starting from approximately 1 MeV with no upper limit. They can be classified into different categories based on their energy [3]:

- Low-energy (LE): 1 - 100 MeV

- High-energy (HE): 0.1 - 100 GeV
- Very high-energy (VHE): 0.1 - 100 TeV
- Ultra high-energy (UHE): 0.1 - 100 PeV
- Extreme high-energy (EHE): 0.1 - 100 EeV

These values may vary slightly depending on the literature, but they give a general idea of the different energy ranges of gamma rays. Of course, depending on the energy range of interest, the observation instrument is different.

2.2 Cherenkov light

Gamma rays cannot be detected directly from the ground because they are absorbed by the Earth's atmosphere. From the ground, the study of these rays relies on the detection of the air showers they produce when they interact with the Earth's atmosphere. However, before addressing air showers, it is important to understand the phenomenon of Cherenkov light.

When a charged particle moves through a medium at a speed greater than the speed of light in that medium, an electromagnetic radiation called Cherenkov light is emitted. However, nothing travels faster than light in a vacuum, but in a medium like water or air, the speed of light is reduced [1].

This principle can be seen as the optical equivalent of a sonic boom. When the particle exceeds the speed of light in the medium, it creates a luminous shock wave, similar to the sound shock wave created by an object moving faster than sound in air.

This Cherenkov light constitutes one of the observable signatures of air showers detected by ground-based instruments.

2.3 Air Showers

An air shower is a cascade of secondary particles produced when a high-energy primary particle (such as a gamma ray or cosmic hadron) collides with molecules in the Earth's atmosphere. The primary particle produces multiple new lower-energy particles, which in turn interact and produce even more particles. These very high-energy particles move faster than the speed of light in air, thus emitting Cherenkov light. Depending on the nature of the primary particle, two types of air showers can be distinguished: electromagnetic showers and hadronic showers [4].

- **Electromagnetic showers**: produced by gamma rays, electrons, or positrons. They are mainly composed of electrons, positrons, and photons. The shape is that of a pointed cone directed towards the ground, with a particle distribution that decreases in density as one moves away from the central axis. These showers are characterized by a compact and regular structure. A diagram illustrating an air shower of gamma origin is shown in Figure 2.2.

- **Hadronic showers**: produced by hadrons, such as protons or nucleons. They are essentially composed of hadrons (protons, neutrons, pions, kaons), muons, and secondary photons and electrons. Among the secondary particles, there are also unstable particles that decay rapidly, such as pions and kaons. These showers have a more diffuse and irregular structure compared to electromagnetic showers.

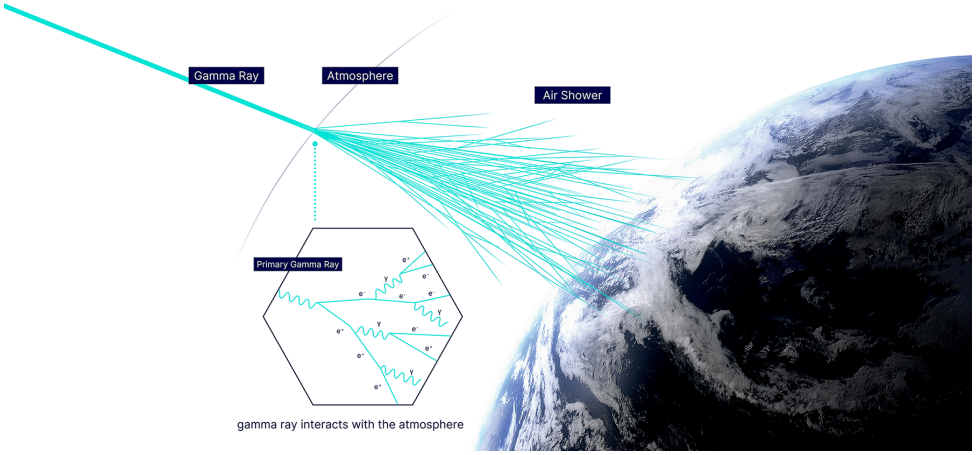


Figure 2.2: Diagram illustrating an air shower of gamma origin (source: CTAO [1]).

In our context, we are primarily interested in electromagnetic showers produced by gamma rays. Indeed, since gamma rays are electrically neutral, they are not deflected by galactic magnetic fields during their propagation, allowing us to trace back their direction of origin and identify their astrophysical sources. Hadronic showers, on the other hand, are initiated by charged cosmic rays whose trajectories are significantly affected by magnetic fields before entering the atmosphere. In addition, hadronic showers are more complex to analyze due to the diversity of secondary particles involved and the multiple interactions they generate.

Another important aspect to consider is that there is a much higher flux of hadronic cosmic rays compared to gamma rays reaching the Earth's atmosphere, making the discrimination between the two types of showers crucial for gamma-ray astronomy.

2.4 Imaging Atmospheric Cherenkov Technique (IACT)

To detect these air showers from the ground, the Imaging Atmospheric Cherenkov Technique (IACT) is used [5]. This technique relies on the detection of Cherenkov light emitted by air showers. Cherenkov telescopes are equipped with a large segmented mirror that reflects the Cherenkov light onto a sensitive detector, typically a camera composed of photo-detectors. The Cherenkov light is very faint and short-lived (a few nanoseconds), requiring highly sensitive and fast detectors to capture these signals. This allows for the recovery of a series of images of the air shower, which can then be analyzed to extract information about the primary particle.

This method aims to reconstruct three main properties of the primary particle from the image of the air shower:

- The type of the primary particle (gamma or hadron)

- The energy of the primary particle
- The arrival direction of the primary particle

These properties are essential for understanding the astrophysical sources of gamma rays and the physical processes involved in their production.

2.5 SST-1M Telescope

In our case, the telescope used is the Small-Sized Telescope - Single Mirror (SST-1M) prototype telescope [6]. The SST-1M is designed to detect very high-energy (VHE) gamma rays in the range of a few TeV to 300 TeV. It is equipped with a single mirror of four meters in diameter, composed of 18 hexagonal segments, and a camera based on Silicon-Photomultipliers (SiPM) that offers high sensitivity and fast temporal resolution.

In 2021, two SST-1M telescopes were installed at the Ondřejov Observatory site in the Czech Republic, 150 m apart from each other (Figure 2.3). They operate in stereoscopy, meaning they simultaneously observe the same air shower from slightly different angles. This configuration allows for a significant reduction in background using coincidence and a better reconstruction of the primary particle's properties by combining information from both telescopes. The images they provide use a hexagonal grid of 1,296 pixels, each pixel corresponding to a single SiPM. The hexagonal arrangement allows for a compact and uniform tiling of the focal plane, thus maximizing the collection of Cherenkov light.

Figure 2.3: The first (left) and second (right) SST-1M installed at Ondřejov Observatory (source: SST-1M project [6]).

Chapter 3

State of the Art

This chapter is divided into two sections. The first section presents the current methods used in gamma-ray astronomy for gamma/hadron separation. The second section introduces autoencoders, a neural network architecture used for anomaly detection. This architecture will be the basis for the method proposed in this thesis for gamma/hadron separation.

3.1 Gamma/Hadron Separation Methods

Cherenkov telescopes detect the light produced by air showers generated when high-energy particles interact with the Earth's atmosphere. Electromagnetic showers, initiated by gamma photons, exhibit different characteristics compared with hadronic showers. Exploiting these differences forms the basis of gamma/hadron separation methods. It is a crucial step in the data analysis pipeline, as it enables the identification of gamma-ray events admit a background dominated by hadronic cosmic rays.

Early methods relied on hand-crafted features extracted from images and rectangular cuts in the multidimensional feature space. Among the most commonly used features are the Hillas parameters, which describe the morphology of the air shower images, such as length, width, and orientation [7]. By applying cuts on these parameters, events can be classified as gamma-like or hadron-like. These cut-based methods are easy to implement but may lack performance in the face of data complexity.

More recently, supervised classifiers such as Random Forests (RFs) and Boosted Decision Trees (BDTs) have been trained on Hillas parameters [8]. These methods can capture non-linear relationships between features, thereby improving gamma/hadron separation performance. This approach has been widely adopted for data analysis in current Cherenkov telescope arrays.

Later, deep learning approaches were explored, using architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) [8]. These models can learn complex representations directly from raw images of the air showers, avoiding the information loss associated with feature extraction. These approaches are particularly promising, as they can potentially outperform traditional methods in terms of performance. However, they need larger

datasets and more computational resources for training. It is also more challenging to interpret the decisions made by deep learning models, which can make it difficult to validate the results, unlike more traditional methods.

More specifically for the SST-1M telescopes, a study [9] published in July 2025 uses RFs for gamma/hadron separation. RFs are trained in both mono-telescope and stereo configurations to classify simulated events. They are trained on Hillas parameters, with the width, log-intensity, and length of the Hillas ellipse identified as the most important features for the classifier. Figure 3.1 shows the ROC curve obtained with the mono and stereo classifiers. An AUC of 0.895 is observed in mono at 30° zenith angle, indicating good classifier performance.

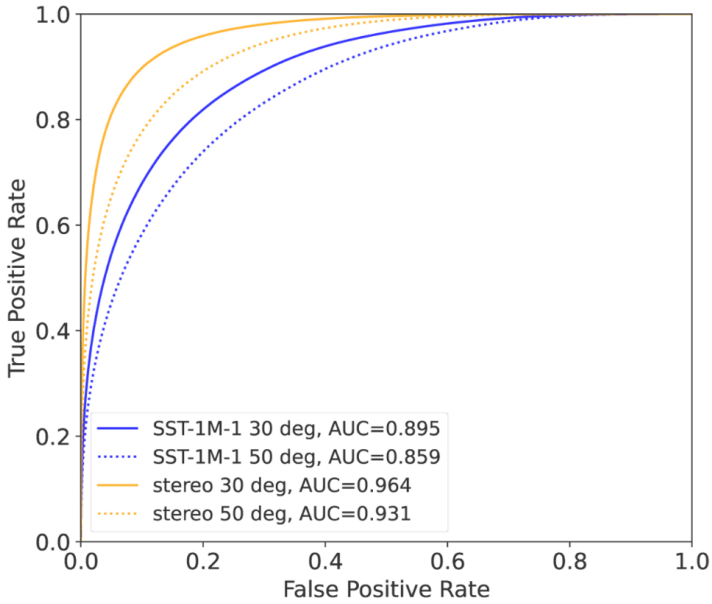


Figure 3.1: ROC curve of the RF gamma-hadron classifier for 30° (solid line) and 50° (dotted line) zenith angles. Line colors represent different regimes of observations (source: SST-1M study [9]).

Deep learning-based approaches have also been explored, as illustrated by the study introducing the CTLearn package for IACT data analysis [8]. This package allows building and training deep learning models using TensorFlow/Keras. Several architectures are presented in this article, such as CNN-RNN, a model composed of a CNN followed by a RNN that takes as input multiple images from different telescopes. Single-tel is a CNN model that takes as input the image from a single telescope. On simulated data for 20° zenith angle for the SST-1M telescope, this latter model achieves an Area Under the Curve (AUC) of 0.853.

These two studies are not directly comparable, as the first applied quality cuts before classification, while the second did not. Additionally, the training methodology and data used differ between the two studies and it's not at the same zenith angle. Nevertheless, they provide an overview of the current performance of different approaches for gamma/hadron separation.

The CTLearn package will not be used in this work, as the PyTorch library is required for certain specific functionalities implemented in this thesis. However, this package illustrates the growing interest in deep learning approaches in the field of gamma/hadron separation.

3.2 Autoencoders for Anomaly Detection

An autoencoder (AE) is a type of neural network composed of two main parts [10]: an encoder and a decoder. The encoder takes an input and compresses it into a lower-dimensional representation called latent space. The decoder then takes this latent representation and attempts to reconstruct the original input from it. The goal of the autoencoder is to learn to reproduce the input as faithfully as possible, by minimizing a loss function that measures the difference between the original input and the reconstructed output (typically Mean Squared Error (MSE)). The architecture of a basic autoencoder is illustrated in Figure 3.2.

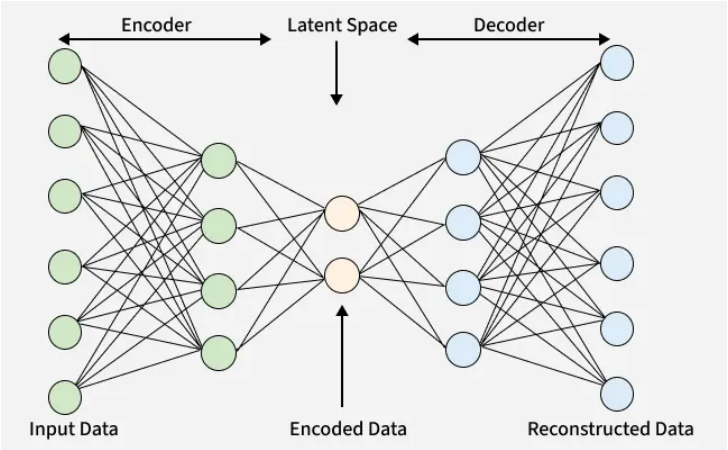


Figure 3.2: Architecture of a basic autoencoder, showing the encoder compressing the input into latent space and the decoder reconstructing the input from this representation (source: GeeksForGeeks [11]).

They are used for various tasks, including dimensionality reduction, anomaly detection, and data generation. In the context of anomaly detection, the idea is that the autoencoder will learn to effectively reconstruct normal data but will struggle to reconstruct abnormal data, resulting in a higher reconstruction error for the latter.

Historically, autoencoders were introduced in 2006 by Hinton and Salakhutdinov [10]. They demonstrated that autoencoders could learn compact representations of data, similar to Principal Component Analysis (PCA), but with the ability to capture non-linear relationships.

Autoencoders have since become a standard for anomaly detection, especially when the data is non-sequential such as images or tabular data [12]. This popularity can be explained by their ability to learn compact representations without explicit supervision (i.e., without labeled data), which is particularly useful when anomalous data is rare or difficult to obtain.

The recent review (2025) by Huang et al. [13] provides a comprehensive overview of the different architectures used for anomaly detection. Autoencoders are often a base for more complex architectures, such as Convolutional Autoencoders (CAE), which use convolutional layers to capture spatial features of the data, or Variational Autoencoders (VAE), which use probabilistic techniques to model the data distribution. Graph Autoencoders (GAE) are also mentioned, which are designed to work with graph-structured data. There are also hybrid models that combine autoencoders with other architectures, such as Generative Adversarial Networks (GANs).

To date, there is, to my knowledge, no study dedicated to the application of autoencoders to gamma/hadron separation. However, this approach has been used and has provided promising results in other related fields, such as anomaly detection in high-energy physics data [14].

In the context of gamma/hadron separation, autoencoders can be trained solely on hadronic events. The hypothesis is that the autoencoder will learn to effectively reconstruct hadronic events but will struggle to reconstruct gamma events, which are considered anomalies in this context. Thus, by evaluating the reconstruction error, one can classify events as hadronic or gamma based on whether the error is below or above a certain threshold.

Chapter 4

Data Description

This chapter describes the data used in this work, including their origin, structure, and a preliminary analysis of the dataset.

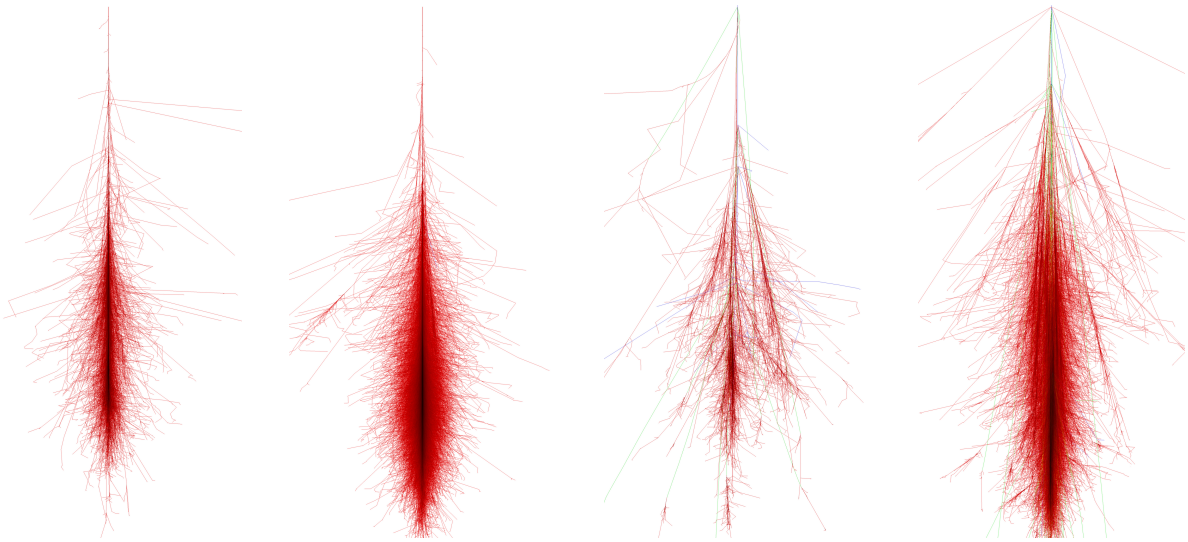
4.1 Data origin

The data used in this work are not real data from a telescope, but simulated data. Simulation allows generating a large number of events with known parameters, such as the type of primary particle (proton or gamma), its energy, direction, etc. This facilitates the evaluation of the performance of the developed models, as the results obtained can be compared with the actual parameters of the simulated events, which would be impossible with real data where this information is not known.

The simulation of the events was carried out prior to this work in two steps using the CORSIKA and sim_telarray tools:

- **CORSIKA [15]:** CORSIKA (COsmic Ray SImulations for KAscade) is a program that allows Monte Carlo simulations of air showers initiated by high-energy cosmic-ray particles. It models the development of the air shower in the Earth's atmosphere by taking into account the physical interactions of particles with the air. CORSIKA simulates the production of secondary particles, the generation of Cherenkov light, and other phenomena associated with air showers. An example of an air shower simulated with CORSIKA is shown in Figure 4.1.
- **sim_telarray [16]:** sim_telarray is a program used to simulate the response of imaging Cherenkov telescopes to air showers generated by CORSIKA. This includes all optical and electronic components of the telescope, such as mirrors, camera, detectors, etc., involved in measuring the Cherenkov signal. In this work, the simulations are specific to the SST-1M telescope design.

Results from the simulations are saved in the `simtel.gz` format. The dataset contains events simulated at a zenith angle of 20° and two types of particles: hadronic events initiated by



(a) Photon (100 GeV) (b) Photon (1 TeV) (c) Proton (100 GeV) (d) Proton (1 TeV)

Figure 4.1: Simulated air showers for photons and protons at different energies using CORSIKA, compiled by Fabian Schmidt, University of Leeds, UK [17].

protons and electromagnetic events initiated by gamma rays. For the rest of this work, they will be referred to as "proton events" and "gamma events", respectively. Also, since these are simulations, the data are already calibrated and thus at R1 level (see next section for data level definitions).

4.2 Data Structure

Data from IACT are generally organized into several levels, each corresponding to a different stage of data processing. The most common levels are R0, R1, DL1, DL2 and DL3. Definitions from the SST-1M pipeline documentation [18]:

- **R0 (Raw Data):** Raw waveforms in each pixel (uncalibrated)
- **R1 (Calibrated Data):** Calibrated waveforms (in photoelectrons, pedestal subtracted)
- **DL1 (Data Level 1):** Integrated charge and peak position of the waveform in each pixel + Hillas parameters.
- **DL2 (Data Level 2):** Reconstructed event parameters (energy, direction, primary type)
- **DL3 (Data Level 3):** Photon lists (after selection and gammaness cut) + Instrument Response Functions

Generally, Data Level 1 (DL1) data are used to create classification and regression models responsible for reconstructing event parameters (DL2) from telescope images. Therefore, this is naturally the data level used to train the autoencoders in this work.

4.2.1 Examples of DL1 data

Before delving into the details of DL1 data, it is important to see what they look like in order to better understand them. The structure of DL1 data will be explained in subsection 4.2.4,

notably the image, peak times, and image mask fields.

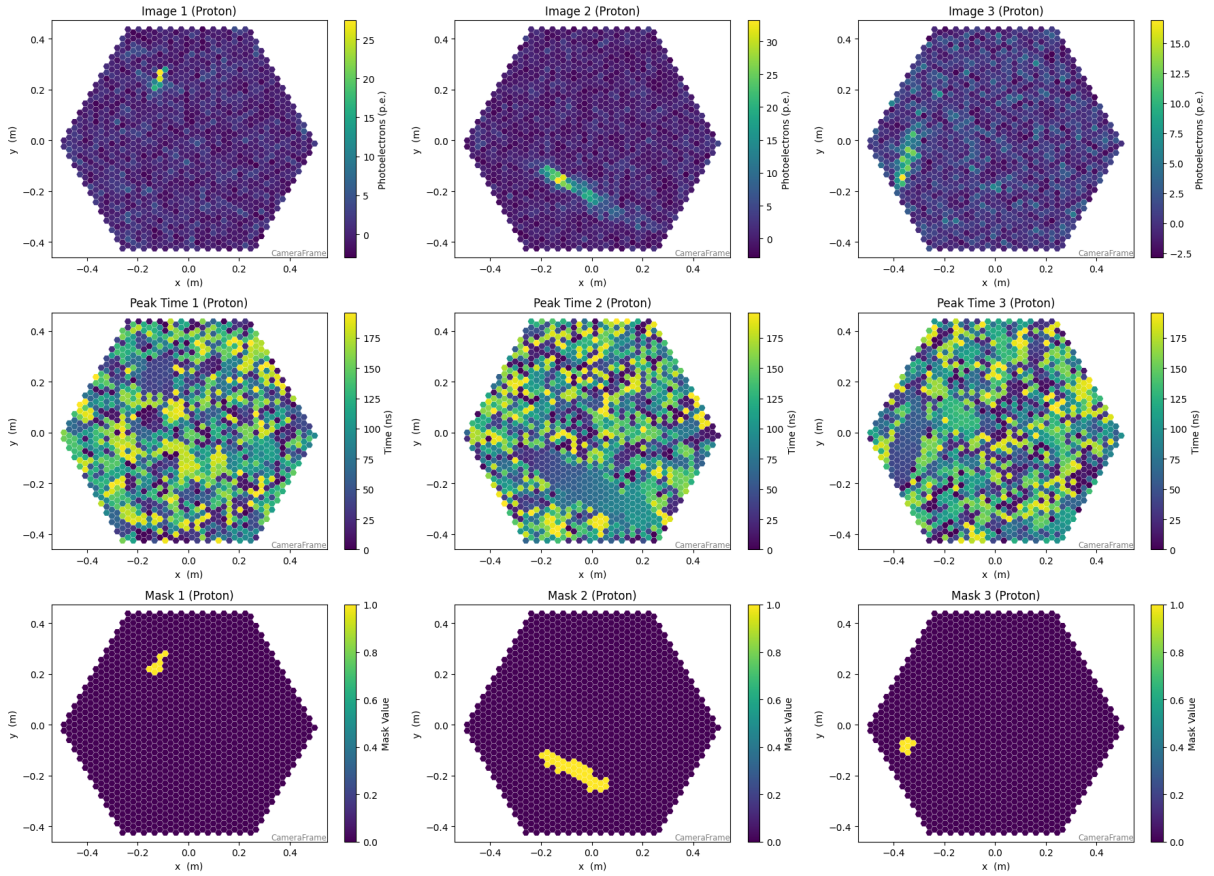


Figure 4.2: Examples of three proton events. For each event from top to bottom: DL1 image, peak times, and image mask.

Figures 4.2 and 4.3 provide examples of DL1 images, peak times, and image masks for proton and gamma events, allowing a first visual inspection of the data. From these examples, it is visually difficult to differentiate between the two types of events solely based on the DL1 images. It is possible to notice that gamma events are generally thinner and more elongated than proton events, which tend to be more diffuse as explained in Chapter 2. In the peak times, the presence of an event is clearly visible with very little variation among the pixels affected by the event, while the rest of the pixels have more random values. At first glance, there is no notable difference between the two types of events in the peak times. Finally, the image masks show which pixels were considered significant after image cleaning. It can be seen that not many pixels are retained and that the entire air shower is not always captured. If this information is used, modifications may need to be made to account for it.

This preliminary visual analysis suggests that the task of differentiating between proton and gamma events based on DL1 images is non-trivial, highlighting the importance of developing models capable of capturing complex features in the data.

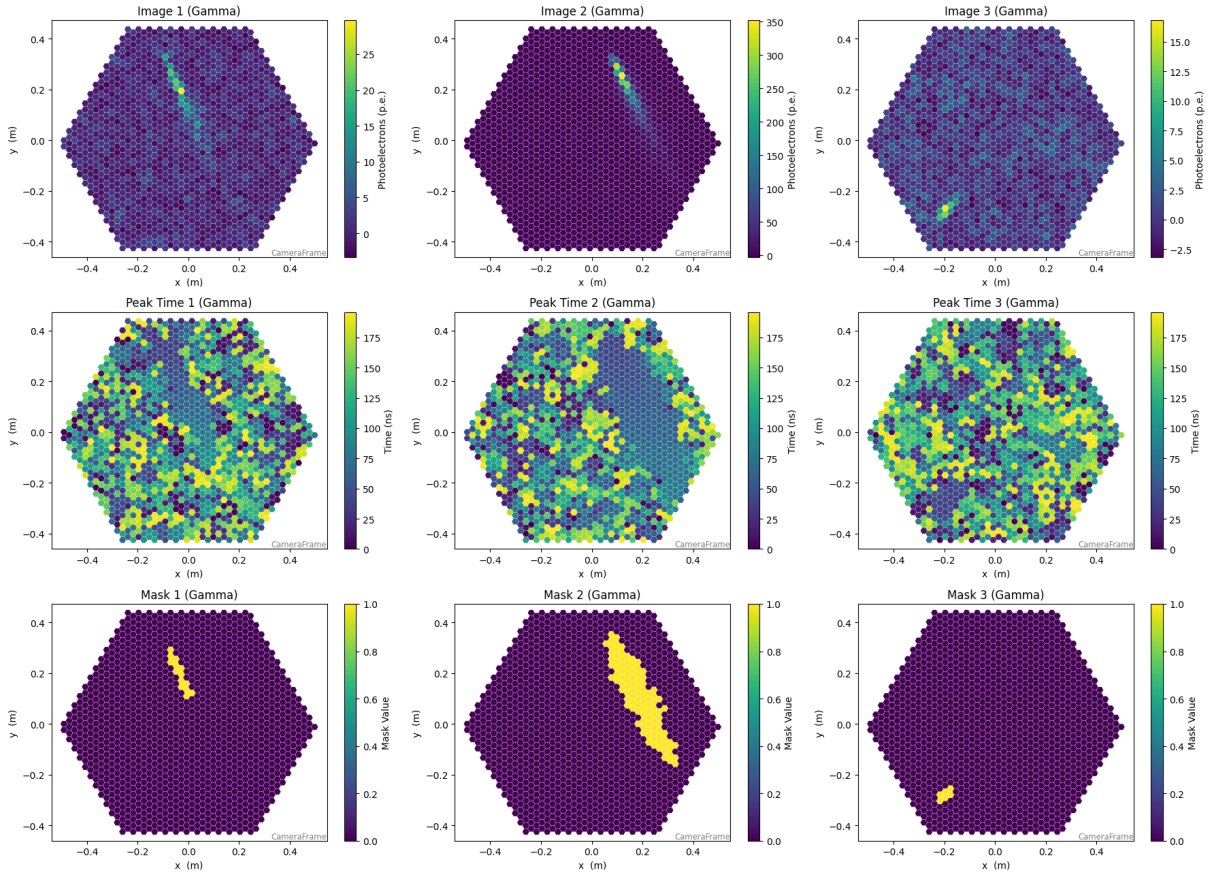


Figure 4.3: Examples of three gamma events. For each event from top to bottom: DL1 image, peak times, and image mask.

4.2.2 R1 Data

Before describing DL1 data, it is important to understand R1 data, as DL1 data are derived from them.

R1 data are the calibrated data obtained after processing the raw R0 data or directly from simulations. They contain the calibrated waveform for each pixel of the camera, expressed in photoelectrons (p.e.) after pedestal subtraction. In the case of the SST-1M telescope, each pixel has 50 samples taken at four ns intervals, covering a window of 200 ns. For an event, there is therefore an array of shape (1296, 50). This includes a lot of information, much of it noise, as the air shower usually leaves a significant signal only on a small part of the camera and for a short period. Using R1 data directly to train autoencoder models would be very computationally expensive and may not be effective due to the large amount of noise present in the data.

4.2.3 R1 to DL1 Data

The difficulty mentioned earlier is resolved by using DL1 data, which are derived from R1 data through a process of calibration and signal extraction. In our case, the transformation from R1 data to DL1 data is performed using the `ProcessorTool` [19] from `ctapipe` library [20]. It also allows converting simtel format files into HDF5 files, which are much easier to manipulate with Python. The default parameters were used, specifying that the R1 waveform data, DL1 images,

and Hillas parameters should be written to the output file. In this process, two classes used are particularly important: `CameraCalibrator` and `ImageProcessor`.

- `CameraCalibrator` [21]: This class performs the complete calibration of the camera. It allows the transformation of raw R1 data into calibrated DL1 data. It performs this operation through two main steps: First, an optional data volume reduction, producing DL0 level data. Second, signal extraction and calibration, which consists of reducing of the waveforms into per-pixel signal-summarising parameters (DL1), namely the integrated charge and the arrival time of the pulse maximum.

- `ImageProcessor` [22]: This class performs image cleaning and extraction of parameters from the cleaned images. It comes after camera calibration. The image cleaning produces a binary mask indicating the pixels considered significant for analysis. This mask is created using the `tailcuts_clean` function [23], which consists of selecting pixels that pass a two-threshold tail-cuts procedure. Examples of masks are shown in Figures 4.2 and 4.3. Then, if the image quality is sufficient (i.e., if there are enough significant pixels), the image parameters are extracted. This includes calculating Hillas parameters, which describe the shape and orientation of the shower image in the camera, as well as other parameters such as leakage, morphology, and pixel intensity statistics.

There are several other steps in the data processing pipeline, but these are the most relevant for the DL1 data used in this work. The `ProcessorTool` orchestrates the use of these classes to convert R1 data into DL1 data, ensuring that the necessary calibration and image processing steps are performed correctly.

4.2.4 DL1 Data

Previously, we described how DL1 data are obtained from R1 data. Now, it is important to understand the structure of the DL1 data themselves and the information they contain. DL1 data include the integrated charge and the peak position of the waveform for each pixel, along with other parameters. The data are stored in a `ctapipe DL1CameraContainer` [24], which contains the following relevant fields for this work:

- **image:** Numpy array of camera image, after waveform extraction of size 1,296 in the case of the SST-1M telescope.
- **peak_time:** Numpy array containing the position of the peak of the pulse as determined by the extractor. The shape is the same as the image.
- **image_mask:** Boolean numpy array where `True` means the pixel has passed cleaning. The shape is also the same as the image.
- **parameters:** Container of image parameters, including Hillas parameters.

For the rest of this work, the terms "image", "peak_time" and "image_mask" will directly refer to the corresponding fields in the `DL1CameraContainer`. These are the main data used to train the autoencoder models.

The "parameters" field notably contains the Hillas parameters, which are currently those used

in most anomaly detection works in the context of imaging Cherenkov telescopes (see Chapter 3). These are a set of parameters that describe the orientation and shape of a shower image in the camera. The `hillas_parameters` [25] function from `ctapipe` allows to compute these parameters from the cleaned image. The main Hillas parameters are as follows:

- **intensity:** the total sum of the pixel charges.
- **skewness:** the measure of the asymmetry.
- **kurtosis:** the measure of the tailedness.
- **x:** the centroid x-coordinate.
- **y:** the centroid y-coordinate.
- **r:** the radial coordinate of centroid.
- **phi:** the polar coordinate of centroid.
- **length:** the standard deviation along the major-axis.
- **length_uncertainty:** the uncertainty of the length.
- **width:** the standard spread along the minor-axis.
- **width_uncertainty:** the uncertainty of the width.
- **psi:** the rotation angle of ellipse.

They have several advantages and disadvantages for the three tasks of gamma-ray detection. They allow to reduce the complexity of the data by condensing the information of the image into a few key parameters, which greatly facilitates the training of models. However, this dimensionality reduction inevitably leads to a loss of information, which can harm the model's ability to detect subtle differences in the data. A model trained directly on DL1 images could potentially capture more complex and nuanced features of the data, thereby improving anomaly detection.

This is why, in this work, the focus is on the use of raw DL1 images rather than solely on Hillas parameters.

4.3 Data visualisation

As previously mentioned, the data used in this work come from simulations of the SST-1M telescope, converted into DL1 data using `ctapipe` [20]. The dataset contains simulated gamma and proton events.

The images, peak times, and image mask are one-dimensional numpy arrays of size 1,296, corresponding to the number of pixels in the SST-1M telescope camera. However, this camera has a hexagonal geometry, and to exploit the spatial characteristics of the data, it is necessary to have information about the spatial arrangement of the pixels. For this, there is the `CameraGeometry` class from `ctapipe` [26], which provides camera-related information used and allows processing and displaying the data. It contains several attributes, notably the most important for this work:

- **pix_x:** Numpy array containing the x-coordinates of each pixel in the camera.
- **pix_y:** Numpy array containing the y-coordinates of each pixel in the camera.

- **neighbors:** Adjacency list indicating the neighboring pixels for each pixel in the camera.

Using this information, it is possible to reconstruct the hexagonal arrangement of the pixels in the camera. The method employed for the preparation of the data will depend on the autoencoder model used and will be described in the corresponding chapter. The camera information allows to create a `CameraGeometry` object that is directly present in the HDF5 files and is easy to use.

`CameraGeometry` can also be used to visualize the data with the `CameraDisplay` class [27]. This is a visualization class that allows displaying camera images while taking into account the hexagonal geometry of the pixels. It requires a `CameraGeometry` object and a data array of the same size as the number of pixels in the camera to display the data according to the spatial arrangement of the pixels. For this work, this class was used to visualize DL1 images, peak times, and image mask. As an example of its capabilities, Figures 4.2 and 4.3 show DL1 images, peak times, and image masks visualized with `CameraDisplay`.

Chapter 5

Methodology

This chapter describes the methodology used to train and evaluate the different autoencoders presented in this work. This includes the training procedure, chosen hyperparameters, evaluation metrics, and data preprocessing steps.

5.1 Training procedure

The autoencoders are trained to reconstruct their input data. The encoder compresses the input data into a lower-dimensional latent representation, while the decoder reconstructs the original data from this compressed representation. The training process involves minimizing a loss function that measures the difference between the input and the reconstructed output. The architectures of the different models are described in Chapter 6.

In this project, proton events are considered normal, while gamma events are considered anomalies. Consequently, the autoencoders are trained exclusively on proton data. The model is then evaluated on a separate test set containing both proton and gamma events to assess its ability to detect anomalies (gamma events). The evaluation focuses not on the quality of image reconstruction, but rather on the model's ability to distinguish between proton events (normal) and gamma events (anomaly) based on reconstruction error.

The models are implemented using the `PyTorch` [28] library, `PyTorch Geometric` [29] for graph-based operations, and `HexagDly` [30] for hexagonal convolutions.

5.1.1 Computational resources and training time

To accelerate computation, all models were trained on the High Performance Computing (HPC) cluster Baobab at the University of Geneva. Several GPUs were used, including NVIDIA RTX A6000 and NVIDIA RTX 4090. The choice of GPU depended on availability on the cluster at the time of training.

In this work, the most time-consuming part was loading the data from storage into GPU memory. This duration strongly depended on the activity on the cluster and the load of the file system.

For the same script, the duration for reading the data could vary from a few minutes to more than two hours depending on system load.

The model training part was generally much faster and took between 5 and 15 minutes for 10 epochs, depending on the model and the size of the data used. This duration is more stable and less affected by system load, as the data are already in GPU memory during training. The training duration for each model is not indicated in this report, as it is relatively short and not a limiting factor in this case.

5.1.2 Data Used

The data used are described in detail in Chapter 4. The training set contains 573,217 proton events. During model training, 20% of the training data are used as a validation set to monitor convergence and prevent overfitting. It is also used to tune hyperparameters if necessary.

The test set contains 70,000 proton events and 70,000 gamma events. Depending on the case, the data used to train the models may vary between the different experiments presented in this work. These variations will be specified in the corresponding sections.

To ensure comparability of experiments, the same training and test datasets were used for all models presented in this work. The test data contain a balanced number of proton and gamma events to facilitate performance evaluation. It's important to note that in real-world scenarios, proton events are much more numerous than gamma events. However, while the simulated data allow the construction of a balanced test set, this does not reflect the real event distribution, which is strongly dominated by proton events.

5.1.3 Normalization

To train the models, data normalization is necessary. The raw data have very different scales across different events depending on the energy of detected particle. For example, most pixels have charges close to 0, but some pixels can have very high charges (several thousands) as shown in Figure 5.1. Inputs with such varied scales can make classification task difficult, and can cause unstable gradients during minimization.

Therefore, min-max normalization was performed per image, removing values below zero. Thus, the value of each pixel will be between zero and one based on the minimum and maximum values of the image. This method preserves the relative distribution of pixels while ensuring that all images have the same scale. If the normalization is modified for certain experiments, this will be specified in the corresponding sections.

For the peak time, min-max normalization between 0 and 196 was used. 196 ns corresponds to the maximal value of the signal acquisition for a waveform of 50 samples with a step of four ns. This allows preserving the temporal information while scaling it between zero and one.

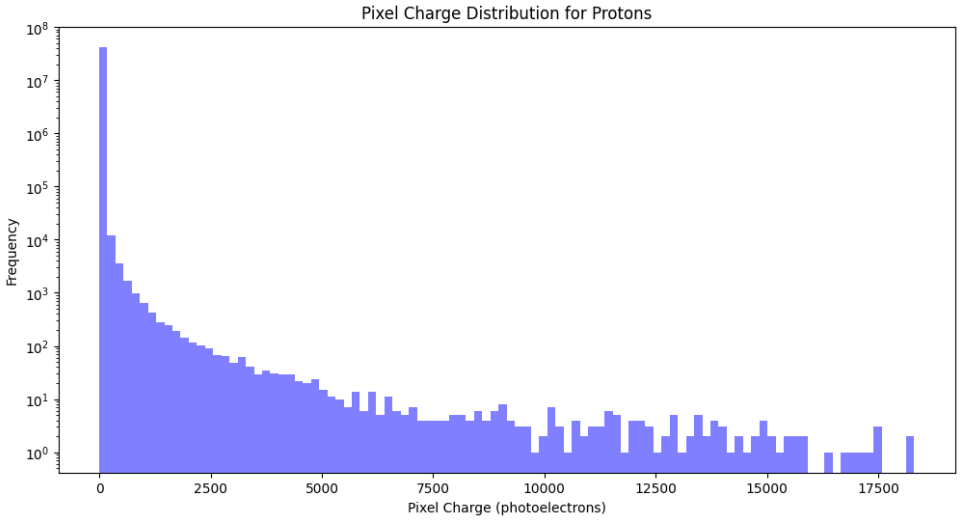


Figure 5.1: Pixel charge distribution before normalization for a sample of 32,627 images from the training set. Most pixels have low charge, while a few exhibit very high values, resulting in a wide distribution. The y-axis is shown on a logarithmic scale.

5.1.4 Hyperparameters

For training the models, several hyperparameters needs to be defined. These hyperparameters include the loss function, optimizer, number of epochs, and batch size.

The loss function used to train the models is the MSE [31] between the input image and the image reconstructed by the autoencoder. It is the standard loss function for autoencoders, as it directly measures the quality of data reconstruction. Its formula for an image is as follows:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{5.1}$$

where:

- N is the total number of pixels in the image
- x_i is the value of pixel i in the input image
- $\hat{x}_i$ is the value of pixel i in the image reconstructed by the autoencoder
- The summation is over all pixels in the image

It strongly penalizes large reconstruction errors, which in our case would correspond to an anomaly (gamma). Other loss functions were tested, such as MAE (Mean Absolute Error), but they did not show significant improvement over MSE.

The optimizer used is Adam (Adaptive Moment Estimation) [32], which is a stochastic optimization method that combines the advantages of Momentum and RMSprop. It adapts the learning rates for each parameter based on the moments of the gradient, allowing for faster and more stable convergence. As with MSE, Adam is a standard choice for training neural networks, including autoencoders. The default parameters of Adam are used (PyTorch implementation [33]):

- Learning rate: 0.001
- Beta 1 (β_1): 0.9
- Beta 2 (β_2): 0.999
- Epsilon (ϵ): 10^{-8}

The number of epochs used is generally 10, which is sufficient for the models to converge without overfitting. The loss function is calculated on a validation set at the end of each epoch to ensure that the model generalizes well on unseen data. The number of epochs can be adjusted based on the convergence observed on the validation set.

The batch size used during training is 64, which balances memory usage and training stability. A larger batch size provides a more stable estimate of the gradient but requires more memory, while a smaller batch size can lead to noisier gradients but allows training with fewer resources. A batch size of 64 is a reasonable choice for provided dataset.

The model hyperparameters, such as the optimizer, batch size, and number of epochs, were chosen after several exploratory trials aimed at ensuring stable convergence and satisfactory performance on the validation set. To confirm these choices, several tests were carried out on different models and data configurations throughout this master's work to ensure the relevance of the selected hyperparameters.

Although no exhaustive search of all hyperparameter combinations was performed for each model, the chosen values demonstrated consistent and robust performance across various tests. Therefore, the selected hyperparameters are considered suitable for the models presented in this work.

5.2 Model Evaluation

Model evaluation is crucial to measure the performance and ability of the autoencoders to generalize on unseen data. Tests are performed on a test dataset distinct from the training and validation sets.

5.2.1 Metrics

To compare the models, two main metrics were chosen: accuracy and AUC.

Accuracy is a straightforward metric that measures the proportion of correct classifications made by the model [34]. It ranges from zero to one, where one means all classifications are correct, and zero means all classifications are incorrect. It is a simple and intuitive metric. It can be problematic if the classes are imbalanced, but in our case, there is a similar number of gamma and proton events in the test data.

However, the models presented in this report provide a reconstruction of the input images rather than direct classification. Therefore, events are classified as gamma or proton based on the reconstruction error (MSE). The idea is that proton events should have a lower reconstruction error since the model is trained only on this type of event. In contrast, gamma events, considered

anomalies, should have a higher reconstruction error. Since the number of gamma and proton events is balanced in the test data, the median reconstruction error is used as the threshold for classification. Events with an error below the median are classified as protons, while those with an error above are classified as gammas. The median was chosen over the mean because it is less sensitive to extreme values and provides a more robust threshold for classification. Note that this classification method is specific to this use case and cannot be applied to real data where protons are much more numerous than gammas and classes are not known in advance. This classification method is used only for comparative purposes in this work. This method is used to get an idea of the model's ability to detect anomalies. An optimal threshold could be determined later based on the specific needs of the application.

The second metric used is the AUC and it is derived from the Receiver Operating Characteristic (ROC) curve [34]. The ROC curve is a graph that shows the performance of a binary classifier at different classification thresholds. For this, we plot the True Positive Rate (TPR) against the False Positive Rate (FPR) for different thresholds. The AUC is the area under this curve, it is an overall measure of model performance regardless of the chosen threshold. An AUC of one indicates perfect classification, while an AUC of 0.5 shows a model similar to random guessing. An example of a ROC curve is shown in Figure 5.2.

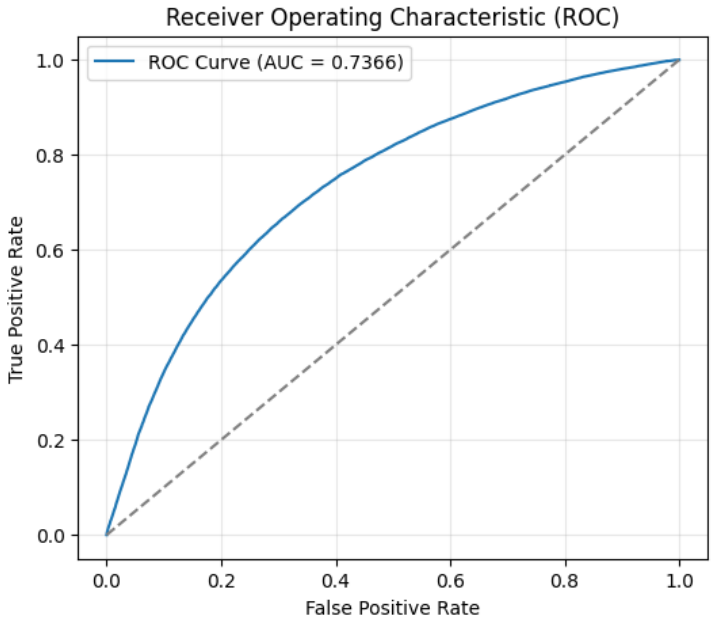


Figure 5.2: Example of a ROC curve illustrating the relationship between True Positive Rate (TPR) and False Positive Rate (FPR) at various classification thresholds. The Area Under the Curve (AUC) quantifies the overall performance of the classifier.

This metric is more relevant than accuracy in our case, as it evaluates the model's performance across all possible thresholds rather than at an arbitrary threshold like the median. This provides a more comprehensive view of the model's ability to detect anomalies. However, AUC is more difficult to interpret than accuracy, as it does not directly correspond to a proportion of correct classifications.

The two metrics are complementary, as the first provides an easily interpretable value, while the

second offers a more robust evaluation of model performance.

In addition to these metrics, precision, recall, and F1-score are also calculated at the threshold used for accuracy [34]. Precision measures the proportion of true positives among events classified as positive, while recall measures the proportion of true positives among actually positive events. The F1-score is the harmonic mean of precision and recall. These additional metrics mainly provide a better understanding of model performance, especially in the context of anomaly detection, where false positives and false negatives can have different implications. For understanding the results, example of image reconstruction for different models will also be presented to visualize how well the models reconstruct the input images.

The MSE will not be used to compare models, as it can vary significantly depending on normalization or the type of data used. Additionally, this work focuses on anomaly detection (gammas) rather than image reconstruction quality. A model may have a low reconstruction error but be poor at detecting anomalies, and vice versa. Therefore, it is not a relevant metric for model evaluation. However, it can be used in specific cases to better understand the behavior of some models.

The distribution of reconstruction error for different event types (gamma and proton) will be shown only when relevant, as it does not consistently provide a clear measure of model performance but can occasionally help interpret a specific model's behavior.

The results will mainly be presented in the form of summary tables of metrics for each tested model. For each modification made to the model (architecture, data type, cleaning method, etc.), a comparative table of models will be used to highlight performance improvements or regressions compared to the previously best-performing version.

Chapter 6

Model Architecture

The objective of this chapter is to present the different basic architectures used for autoencoders in this work. It aims to present their general functioning as well as the motivations behind their selection. Also, some architectures require specific pre-processing of the data, which will also be explained here.

6.1 Flat autoencoders

The first architecture is the simplest: a symmetric encoder and decoder composed of dense layers (fully connected layers). A dense layer is a layer where each neuron is connected to all the neurons of the previous layer, thus requiring a large number of parameters. Mathematically, a dense layer can be represented as follows:

$$y = \sigma(Wx + b)$$

where:

- x is the input vector ;
- W is the weight matrix ;
- b is the bias vector ;
- σ is the activation function (ReLU in this case) ;
- y is the output vector.

The encoder compresses the input image into a smaller vector, then the decoder reconstructs the image from this compressed vector. The encoder and decoder each consist of two dense layers, with a Rectified Linear Unit (ReLU) activation function between the layers. After the encoder, the size of the latent vector is 64. The input is the original image, a simple 1D vector of size 1,296 (number of pixels in the camera). The output is also a 1D vector of the same size, representing the reconstructed image. There is no specific pre-processing of the data for this model. The DL1 images are used as they are, in the form of 1D vectors.

The summary of the architecture is presented in Figure 6.1. Despite its simplicity, this model

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AE                                       [1, 1296]                 --
├─Sequential: 1-1                        [1, 64]                   --
│    └─Linear: 2-1                       [1, 128]                  166,016
│    └─ReLU: 2-2                         [1, 128]                  --
│    └─Linear: 2-3                       [1, 64]                   8,256
├─Sequential: 1-2                        [1, 1296]                 --
│    └─Linear: 2-4                       [1, 128]                  8,320
│    └─ReLU: 2-5                         [1, 128]                  --
│    └─Linear: 2-6                       [1, 1296]                 167,184
==========================================================================================
Total params: 349,776
Trainable params: 349,776
Non-trainable params: 0
Total mult-adds (M): 0.35
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.01
Params size (MB): 1.40
Estimated Total Size (MB): 1.42
==========================================================================================
```

Figure 6.1: Summary of the flat autoencoder architecture.

has a large number of parameters (over 300,000), due to the fact that each pixel is connected to every neuron in the next layer and the size of the input image is relatively large. This can lead to overfitting if the number of training examples is insufficient.

This model is the simplest possible and does not take into account the spatial structure of the image. Each pixel is treated independently of the others, whereas the spatial relationship between pixels is generally important for image reconstruction. However, this model serves as a reference to evaluate the performance of other more complex architectures. For the rest of the work, this architecture will be called "Flat AE".

6.2 CNN Autoencoders with square pixels

The second architecture aims to exploit the spatial structure of the images using convolutions. However, the pixels of the camera are hexagonal, and standard convolutions are implemented for square pixels. To use these convolutions, the simplest solution is to transform hexagonal pixels into square pixels. This allows the use of standard deep learning tools for images, but this transformation inevitably leads to a loss of information.

For this, the `DL1 Data Handler library` [35] was used, which is a Python library developed to read and manipulate DL1 data from imaging Cherenkov telescopes. It includes several methods to transform hexagonal images into square images. After studying and testing several of these methods, two of them were selected: ShiftingMapper and BilinearMapper.

- **ShiftingMapper:** This method applies a shifting transformation to map images from a hexagonal pixel grid to a square pixel grid. This method creates a square image of size 48x48 (2304 pixels), with areas outside the hexagonal image filled with zeros. This method makes it possible to keep a reasonably sized image that is simple to compute. However, it leads to a significant loss of information.

- **BilinearMapper:** The bilinear mapping method generates a bilinear interpolation table to perform the image transformation. It relies on Delaunay triangulation [36] to identify nearest neighbors during the interpolation process. The mapping table is normalized to preserve pixel intensity values. This approach is well suited for applications requiring smooth and continuous pixel data mapping and is recommended as the default method for hexagonal pixel cameras [35]. This method creates a square image of size 96x96 (9216 pixels), which allows to keep more information from the original hexagonal image compared to the ShiftingMapper. However, the resulting image is larger and still contains areas filled with zeros. This requires more computational resources for model training.

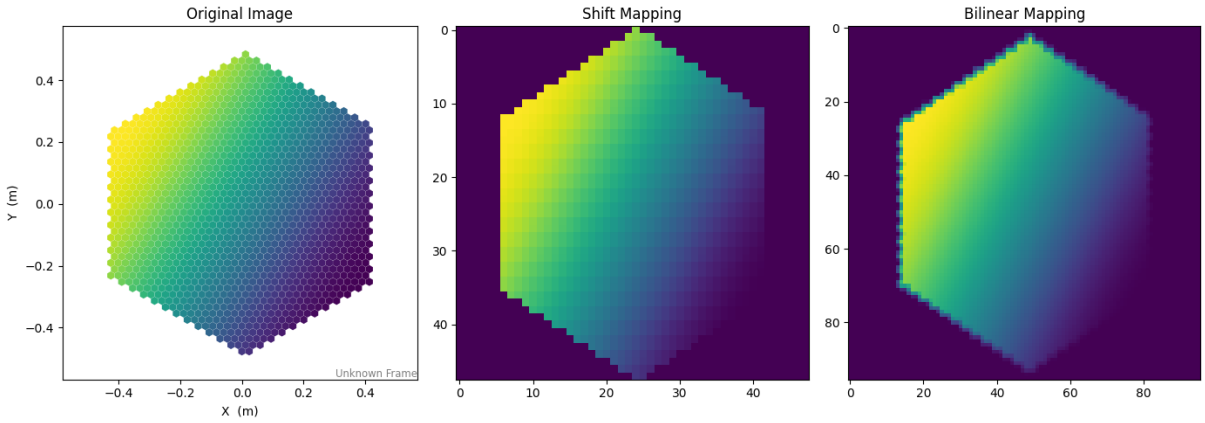


Figure 6.2: Comparison of hexagonal to square image transformation methods. On the left, the original hexagonal image, in the middle, the image transformed using ShiftingMapper (48x48), on the right, the image transformed using BilinearMapper (96x96).

To visualize the differences between these two methods, Figure 6.2 presents an example of a hexagonal image transformed into a square image with each of the methods. For both methods, we can see black areas (zero values) around the hexagonal image, corresponding to areas outside the camera and a stretching of the hexagon. In the ShiftingMapper transformation, we notice a square structure in the image. In the BilinearMapper transformation, the hexagon is more continuous and does not have a square structure, but the image is larger and the borders are blurrier. Thus, there is indeed a loss of information in both cases, but the images remain exploitable for reconstruction.

After transforming the hexagonal images into square images using one of these two methods, an autoencoder architecture was designed based on convolutional and transposed convolutional layers.

A convolutional layer is a type of neural network layer specifically designed to process grid-structured data, such as images. It applies a set of filters (or kernels) to the input image to extract local features. Each filter slides over the image, performing a convolution operation that involves multiplying the pixel values by the filter weights and summing the results to produce a single output value. This process is repeated for each position of the filter on the image. Several parameters need to be defined, including the number of filters, filter size, stride (the step size for moving the filter across the image), and padding (the addition of pixels around the image

borders). In Figure 6.3, we can see an example of a 3x3 filter applied to a 5x5 image, producing a 3x3 feature map. This type of layer is effective in capturing local patterns in images and reducing the number of parameters compared to a dense layer. Indeed, the filter weights are shared across the entire image, unlike in a dense layer where each connection has its own weight. Each filter has different weights, and multiple filters are used to extract different features from the image, such as edges, textures, etc.

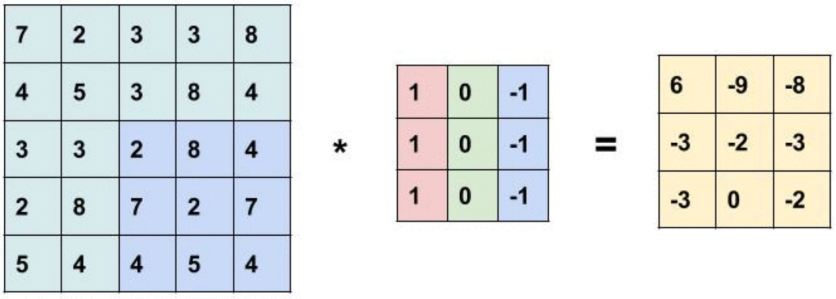


Figure 6.3: Example of a convolutional layer with a 3x3 filter applied to a 5x5 image, producing a 3x3 feature map, source : Medium [37].

A transposed convolutional layer allows for up-sampling (increasing the spatial size of an image). It is the opposite operation of a standard convolutional layer, which typically reduces spatial dimensions. It is often used in the decoders of autoencoders or in generative neural networks to reconstruct images from latent representations. The operating principle is similar to that of a standard convolutional layer, but instead of sliding a filter over the image to reduce its size, the transposed convolutional layer makes each input pixel contribute to a larger area in the output according to the filter weights, which increases the size of the output. An example is shown in Figure 6.4, where a 2x2 filter is applied to a 2x2 feature map, producing a 3x3 image. As with standard convolutional layers, several parameters need to be defined, such as the number of filters, filter size, stride, and padding.

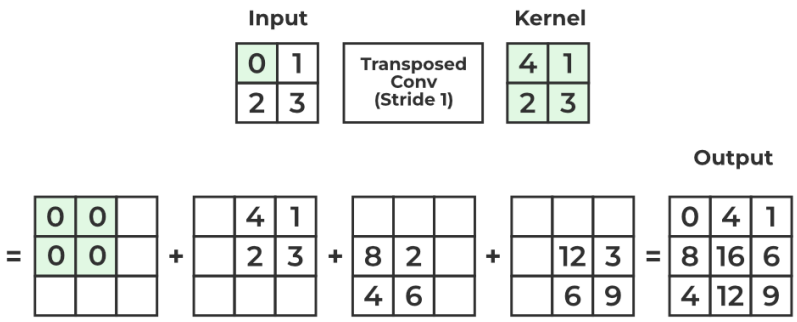


Figure 6.4: Example of a transposed convolutional layer with a 2x2 filter applied to a 2x2 feature map, producing a 3x3 image, source : GeeksForGeeks [38].

This architecture uses convolutional layers in the encoder and transposed convolutional layers in the decoder. The encoder consists of three convolutional layers, each followed by a ReLU activation function. It reduces the size of the image while increasing the number of channels,

then at the end, it flattens the extracted features and projects them into a latent space of size 64. The decoder starts with a dense layer that takes the latent vector and projects it into a larger feature space, then uses three transposed convolutional layers with ReLU activation functions to reconstruct the original image. The size of the input image depends on the transformation method used: 48x48 for ShiftingMapper and 96x96 for BilinearMapper. The summary of the architecture is presented in Figure 6.5, for the case of ShiftingMapper (BilinearMapper is similar, but with larger image sizes).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AE                                       [1, 1, 48, 48]            --
├─Sequential: 1-1                        [1, 64]                   --
│    └─Conv2d: 2-1                       [1, 4, 23, 23]            40
│    └─ReLU: 2-2                         [1, 4, 23, 23]            --
│    └─Conv2d: 2-3                       [1, 8, 11, 11]            296
│    └─ReLU: 2-4                         [1, 8, 11, 11]            --
│    └─Conv2d: 2-5                       [1, 16, 5, 5]             1,168
│    └─ReLU: 2-6                         [1, 16, 5, 5]             --
│    └─Flatten: 2-7                      [1, 400]                  --
│    └─Linear: 2-8                       [1, 64]                   25,664
├─Sequential: 1-2                        [1, 1, 48, 48]            --
│    └─Linear: 2-9                       [1, 400]                  26,000
│    └─ReLU: 2-10                        [1, 400]                  --
│    └─Unflatten: 2-11                   [1, 16, 5, 5]             --
│    └─ConvTranspose2d: 2-12             [1, 8, 11, 11]            1,160
│    └─ReLU: 2-13                        [1, 8, 11, 11]            --
│    └─ConvTranspose2d: 2-14             [1, 4, 23, 23]            292
│    └─ReLU: 2-15                        [1, 4, 23, 23]            --
│    └─ConvTranspose2d: 2-16             [1, 1, 48, 48]            65
==========================================================================================
Total params: 54,685
Trainable params: 54,685
Non-trainable params: 0
Total mult-adds (M): 0.58
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.07
Params size (MB): 0.22
Estimated Total Size (MB): 0.30
==========================================================================================
```

Figure 6.5: Summary of the autoencoder architecture with square pixels (ShiftingMapper).

This architecture requires significantly fewer parameters than the flat AE, thanks to weight sharing in the convolutional layers. This helps to limit the risk of overfitting and better capture the spatial features of the images. However, the model still requires a lot of parameters in the dense layer in the middle, as the image is still quite large even after the convolutions. However, the transformation of hexagonal images into square images leads to a loss of information, which can affect the quality of the reconstruction. For the rest of the work, this architecture will be called "Square AE", with the specification of the transformation method used (Shifting or Bilinear).

6.3 Graph Autoencoder

The third architecture uses graph structures to process hexagonal images without transforming them into square images. This approach allows preserving the original spatial structure of the

images and avoids the loss of information associated with the transformation into square images. Graphs are data structures composed of nodes (or vertices) and edges that connect these nodes together, as shown in Figure 6.6. In the case of hexagonal images, each pixel can be represented as a node, and the edges represent the neighborhood between pixels. Figure 6.7 illustrates how a hexagonal image can be represented as a graph.

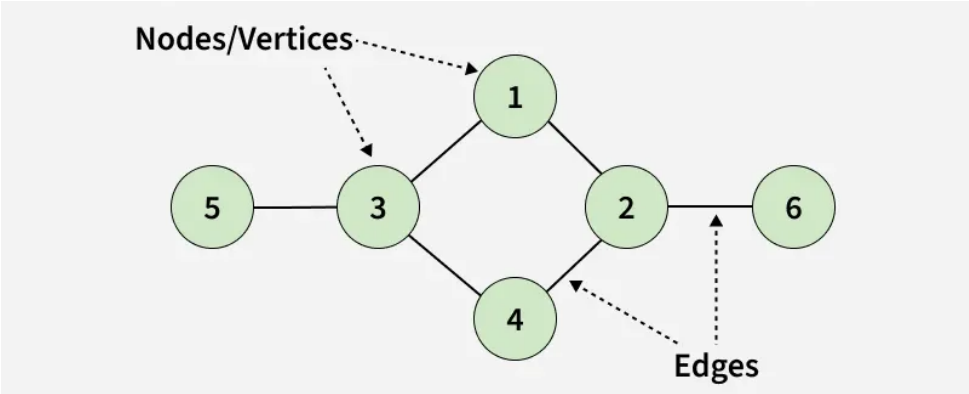


Figure 6.6: Structure of a graph, source: GeeksForGeeks [39].

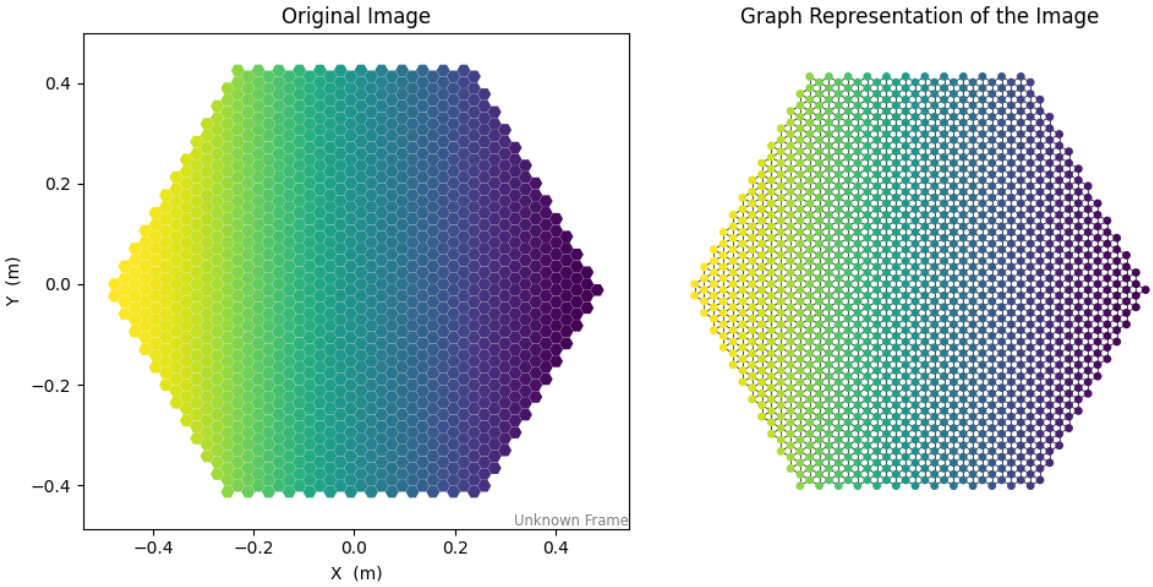


Figure 6.7: Representation of a hexagonal image as a graph. Each pixel is a node connected to its neighbors by edges.

For implementing this architecture, the `PyTorch Geometric` library [29] was used, which is an extension of `PyTorch` specifically designed for graph processing. It is possible to define a graph by specifying the nodes and edges. In the `CameraGeometry` structure presented in Chapter 4, the "neighbors" field is present, indicating the neighboring pixels for each pixel. This information was used to construct the list of edges of the graph. For each pixel, a node was created to which one or more features were assigned depending on the input data, and an edge is created between two nodes if the corresponding pixels are neighbors.

The architecture of this graph autoencoder uses Graph Convolutional Network (GCN) convolutional layers [40] in the encoder to extract features from hexagonal images and a Multi-Layer

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
GraphAutoencoder                         [1296, 1]                 --
├─GCNConv: 1-1                           [1296, 32]                32
│    └─Linear: 2-1                       [1296, 32]                32
│    └─SumAggregation: 2-2               [1296, 32]                --
├─GCNConv: 1-2                           [1296, 4]                 4
│    └─Linear: 2-3                       [1296, 4]                 128
│    └─SumAggregation: 2-4               [1296, 4]                 --
├─Sequential: 1-3                        [1296, 1]                 --
│    └─Linear: 2-5                       [1296, 32]                160
│    └─ReLU: 2-6                         [1296, 32]                --
│    └─Linear: 2-7                       [1296, 1]                 33
==========================================================================================
Total params: 389
Trainable params: 389
Non-trainable params: 0
Total mult-adds (M): 0.46
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.72
Params size (MB): 0.00
Estimated Total Size (MB): 0.72
==========================================================================================
```

Figure 6.8: Summary of the graph autoencoder architecture.

Perceptron (MLP) in the decoder for reconstruction. The encoder consists of two GCN layers, each followed by a ReLU activation function. The decoder is an MLP with two dense layers, also with a ReLU activation between the layers. In the middle, the size of the features for each node is four. The input is the graph constructed from the hexagonal image and the output is the reconstructed graph. The summary of the architecture is presented in Figure 6.8.

The principle of a GCN layer is to aggregate the features of a node with those of its neighbors to capture the local structure of the graph, then apply a linear transformation followed by an activation function. Each node updates its features based on those of its neighbors, allowing the model to learn representations that take into account the structure of the graph. Mathematically, a GCN layer is defined as follows (without its activation function) [41]:

$$X' = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}X\Theta$$

where:

- X is the node feature matrix ;
- $\hat{A}$ is the adjacency matrix of the graph with added self-loops ;
- $\hat{D}$ is the degree matrix associated with $\hat{A}$;
- Θ is the weight matrix learned by the layer.

The weights Θ are shared among all nodes, which means that the model has few parameters to learn, as seen in Figure 6.8 compared to the previous architectures. The risk of overfitting is therefore limited, but the model has some drawbacks. If multiple GCN layers are stacked, the model can capture increasingly global information about the graph, as each node aggregates the features of its neighbors at each layer. The result becomes increasingly smooth, as the features of the nodes eventually converge to similar values. Also, this model does not reduce the size of

the graph, the number of nodes remains the same throughout the model. The information is therefore not compressed if only one feature per node is provided. However, the model cannot simply learn an identity function, as the weights are the same for all nodes and each node is obligatorily influenced by its neighbors. This point should not be neglected, as the model must learn to reconstruct the input graph from the aggregated features. For the rest of the work, this architecture will be called "Graph AE".

6.4 Autoencoder with hexagonal convolutions

The last architecture directly uses convolutions adapted to hexagonal images, thus avoiding the transformation into square images and the associated loss of information. For this, the `HexagDly` library [30] was used, which provides convolution and pooling methods specifically designed for hexagonal images while integrating with `PyTorch`.

The difference between hexagonal and square pixels lies in their spatial arrangement. Hexagonal pixels are organized in a grid where each pixel has six direct neighbors, unlike square pixels which have four or eight depending on the definition of connectivity. Therefore, convolution operations need to be adapted to account for this particular structure.

For this, `HexagDly` uses an addressing scheme that maps hexagonal data onto a regular 2D grid. Then, it adapts the convolution and pooling operations to account for the hexagonal structure. To perform a correct hexagonal convolution, the kernel is divided into several sub-kernels which, when combined, cover the true neighbors of a point in the hexagonal grid. Due to the alternating offset between columns of the hexagonal grid, these sub-kernels themselves need to be offset differently depending on whether the kernel is centered on an even or odd column.

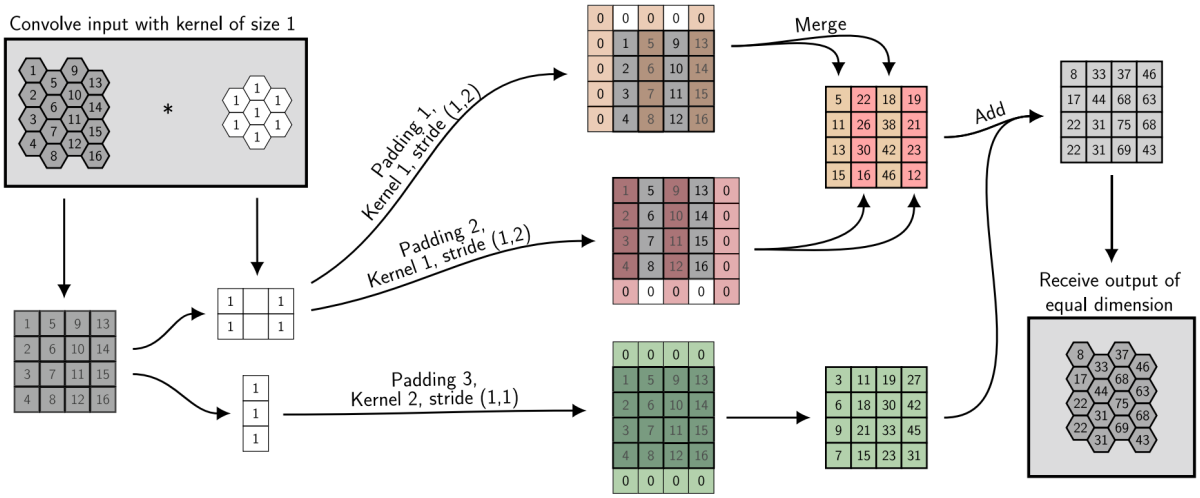


Figure 6.9: "Realisation of a hexagonal convolution with a kernel of size one. First, the input data is rearranged into a tensor and the kernel is divided into rectangular sub-kernels. For every sub-kernel, different paddings and strides are applied to the input to account for the shifted columns. The results of the sub-convolutions are then merged and added to receive the convolved hexagonal data in tensor format", source: HexagDly [42].

Figure 6.9 shows an example of how a hexagonal convolution kernel of size one is applied to a

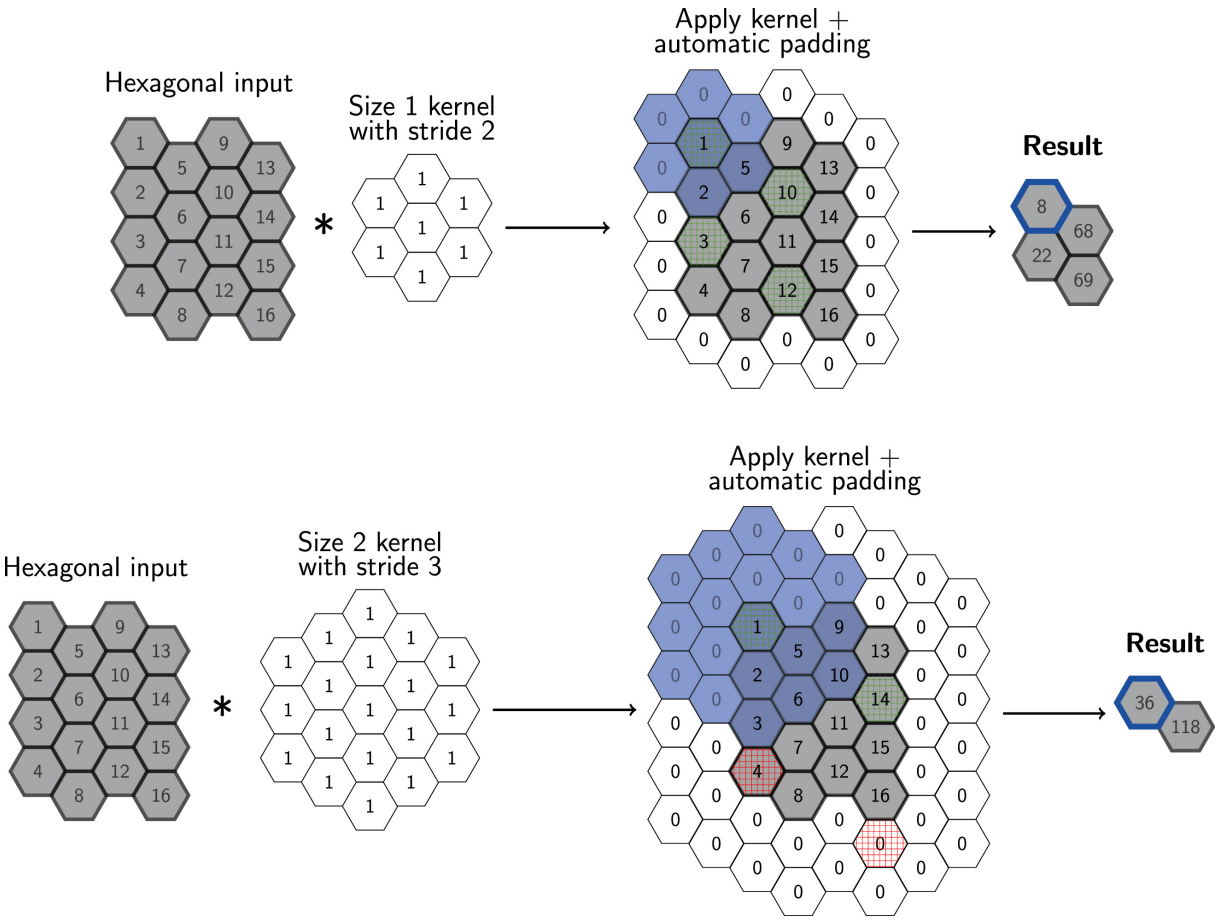


Figure 6.10: "Illustration of the padding and striding scheme for convolutions with different kernel sizes and strides. The green and red elements mark valid and omitted steps, respectively. The first position of a kernel is marked in blue as well as its corresponding output cell in the result.", source: HexagDly [42].

hexagonal image. Thus, this method allows performing convolutions that respect the hexagonal structure of the images, while remaining within the `PyTorch` framework. This allows using all the other features of this library.

Figure 6.10 illustrates the padding and striding scheme used for hexagonal convolutions with different kernel sizes and strides. It shows how the convolution operation works in practice, taking into account the unique structure of hexagonal grids.

To use this library, it is first necessary to transform the hexagonal images into a format compatible with `HexagDly`. This involves reorganizing the hexagonal pixels into a data structure that the library can process. The expected input uses an "odd-q" offset coordinate system, where rows are aligned and columns are offset (odd columns shifted down). Figure 6.11 illustrates this coordinate system.

In our case, the pixels of the different hexagonal images are contained in a 1D vector of size 1,296 (number of pixels in the camera). However, from `CameraGeometry` structure presented in Chapter 4, the x and y coordinates of each pixel ("pix_x" and "pix_y") are available. They were therefore used to create new coordinates adapted to the "odd-q" format. It was necessary to

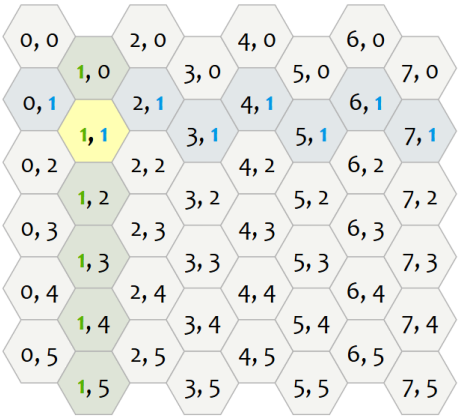


Figure 6.11: "odd-q" vertical layout shoves odd columns down, source: Red Blob Games [43].

shift the x coordinates of each pixel by one unit, because the rightmost pixel of the camera must be on an even (non-shifted) column, which is not the case with the original coordinates. After addressing, the image is of size 36x49 (1764 pixels), with areas filled with zeros corresponding to spaces outside the hexagonal camera. This conversion is illustrated in Figure 6.12.

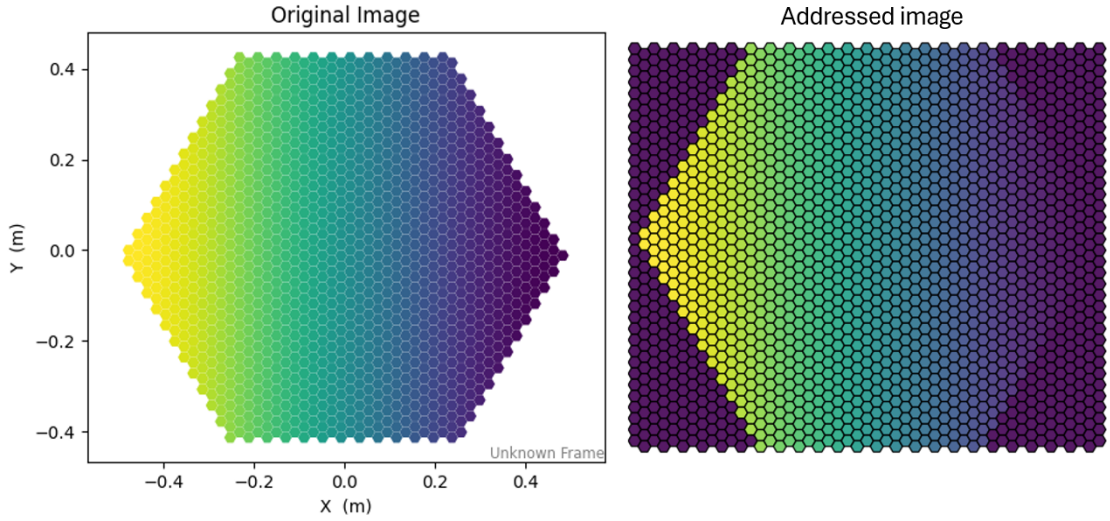


Figure 6.12: Conversion of a hexagonal image into an "odd-q" addressed image compatible with `HexagDly`. On the left, the original hexagonal image. On the right, the image after addressing, with areas filled with zeros outside the hexagonal camera.

A problem still remains: `HexagDly` implements hexagonal convolution and pooling operations, but not hexagonal transposed convolution operations which are necessary for the decoder. An implementation would be very complex and time-consuming to carry out, and would exceed the scope of this work. Therefore, to work around this problem, this operation was replaced with a combination of interpolation followed by a standard hexagonal convolution. The interpolation increases the size of the image, then the hexagonal convolution processes the enlarged image. The "nearest" method was used for the interpolation, which simply copies the value of the nearest pixel to fill in the new pixels created during the upsampling [44] as shown in Figure 6.13. This method is simple and fast, and it works well in practice. The result is not exactly the same as

a transposed convolution, but it remains similar and very fast.

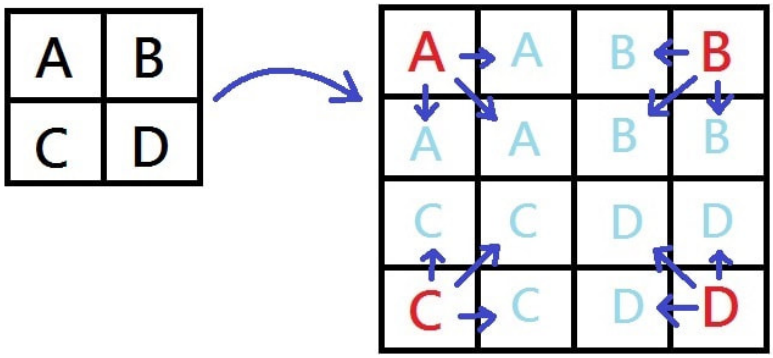


Figure 6.13: Example of "nearest" interpolation method for upsampling, source: Jason Chen's Blog [45].

The architecture of this autoencoder uses hexagonal convolutional layers in the encoder and interpolations followed by hexagonal convolutions in the decoder. The encoder consists of two hexagonal convolutional layers, each followed by a ReLU activation function. A stride of two is used to reduce the size of the image at each layer. After the encoder, the size of the image is 9x13 (117 pixels) on four channels. The resulting feature map is flattened and projected into a latent space of size 64 via a dense layer. Then, a dense layer projects the latent vector into a space of size 9x13x4, which is then reshaped into a hexagonal image. After, the decoder starts with an interpolation to increase the size of the image, followed by a hexagonal convolution with a ReLU activation function. This process is repeated a second time to reconstruct the original image. The size of the input image is 36x49 (1764 pixels) after "odd-q" addressing. The summary of the architecture is presented in Figure 6.14.

The number of parameters of this architecture is intermediate between that of the flat AE and that of the Graph AE, and therefore similar to that of the AE square, as can be seen in Figure 6.14. This result is logical, as the operation is similar to that of the AE square, but with hexagonal convolutions instead of standard convolutions. This architecture allows preserving the hexagonal structure of the images and avoiding the loss of information related to the transformation into square images, while exploiting the advantages of convolutions to capture the spatial features of the images. For the rest of the work, this architecture will be called "Hexa AE".

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
HexAE                                    [1, 1, 36, 49]            --
├─Sequential: 1-1                        [1, 8, 18, 25]            --
│    └─Conv2d: 2-1                       [1, 8, 18, 25]            160
│    └─ReLU: 2-2                         [1, 8, 18, 25]            --
├─Sequential: 1-2                        [1, 4, 9, 13]             --
│    └─Conv2d: 2-3                       [1, 4, 9, 13]             228
│    └─ReLU: 2-4                         [1, 4, 9, 13]             --
├─Flatten: 1-3                           [1, 468]                  --
├─Linear: 1-4                            [1, 64]                   30,016
├─Linear: 1-5                            [1, 468]                  30,420
├─Unflatten: 1-6                         [1, 4, 9, 13]             --
├─Sequential: 1-7                        [1, 8, 18, 25]            --
│    └─Conv2d: 2-5                       [1, 8, 18, 25]            232
│    └─ReLU: 2-6                         [1, 8, 18, 25]            --
├─Conv2d: 1-8                            [1, 1, 36, 49]            153
==========================================================================================
Total params: 61,209
Trainable params: 61,209
Non-trainable params: 0
Total mult-adds (M): 0.06
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.24
Estimated Total Size (MB): 0.33
==========================================================================================
```

Figure 6.14: Summary of the autoencoder architecture with hexagonal convolutions.

Chapter 7

Experimental Setup and Results

This chapter presents the various experiments conducted to find the best approach for anomaly detection using autoencoders on the given dataset. Each section describes a specific modification made to the model or data, along with the results obtained. It begins with the presentation of the baseline models used as a reference in Chapter 6, then explores various approaches to optimize anomaly detection. The entire training and evaluation procedure is detailed in Chapter 5. If a modification is made compared to the baseline model or training procedure, it is explained in the corresponding section. Otherwise, the different models, parameters, and normalization remain unchanged from the previous chapters.

7.1 Baseline Models

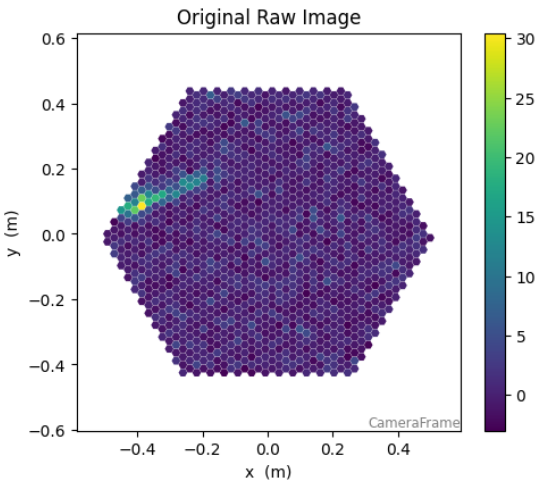
The first step of this experimental study is to establish a solid baseline using basic models. These models will serve as a point of comparison to evaluate the effectiveness of various modifications and improvements made later. The baseline architectures used are described in detail in Chapter 6, and the training and evaluation procedures are explained in Chapter 5. The data used are the images from DL1 dataset.

Table 7.1: Baseline model performance metrics.

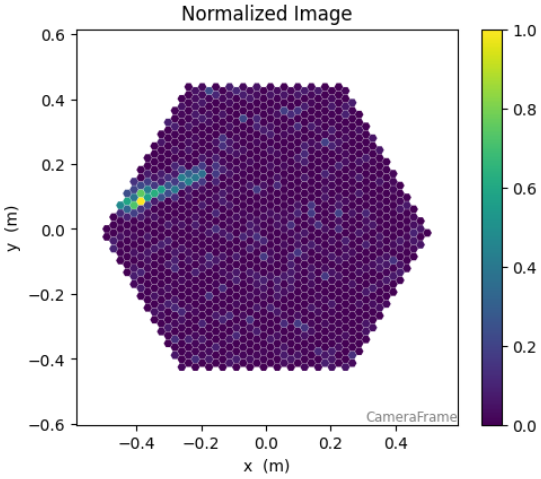
Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (base)	0.4133	0.44	0.44	0.44	0.44
Square AE Shifting (base)	0.4168	0.44	0.44	0.44	0.44
Square AE Bilinear (base)	0.4182	0.44	0.44	0.44	0.44
Graph AE (base)	0.4200	0.43	0.43	0.43	0.43
Hexa AE (base)	0.4204	0.44	0.44	0.44	0.44

The results obtained with the baseline models, presented in Table 7.1, show that all models obtain very poor results, to the point where a random model would be expected to perform better (Accuracy and AUC are below 0.5). This means that currently, gamma images are better reconstructed than proton images, even though the models were trained only on protons. This is not very surprising either, as protons have a more complex behavior than gammas, which have a more regular behavior. Therefore, a model capable of capturing this complexity is needed to

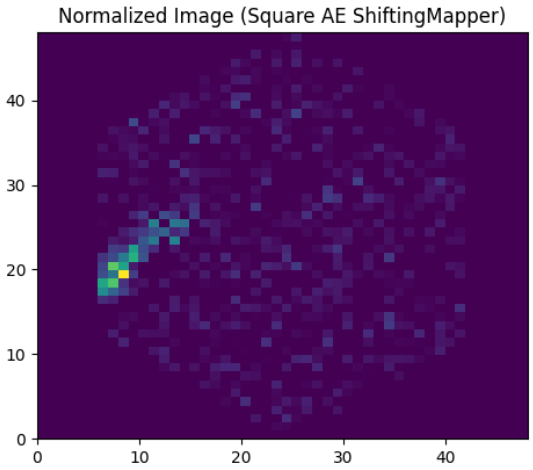
successfully detect gamma events as anomalies. To find possible improvements, we must first understand what the models have learned to do. For this, we need to look at how the models reconstruct the camera images.



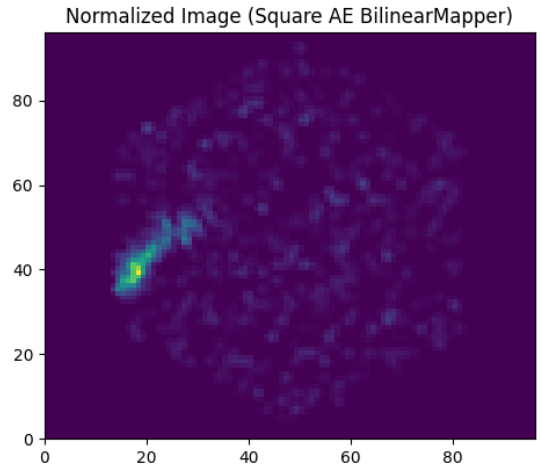
(a) Original Image



(b) Normalized Image



(c) Normalized and ShiftingMapper

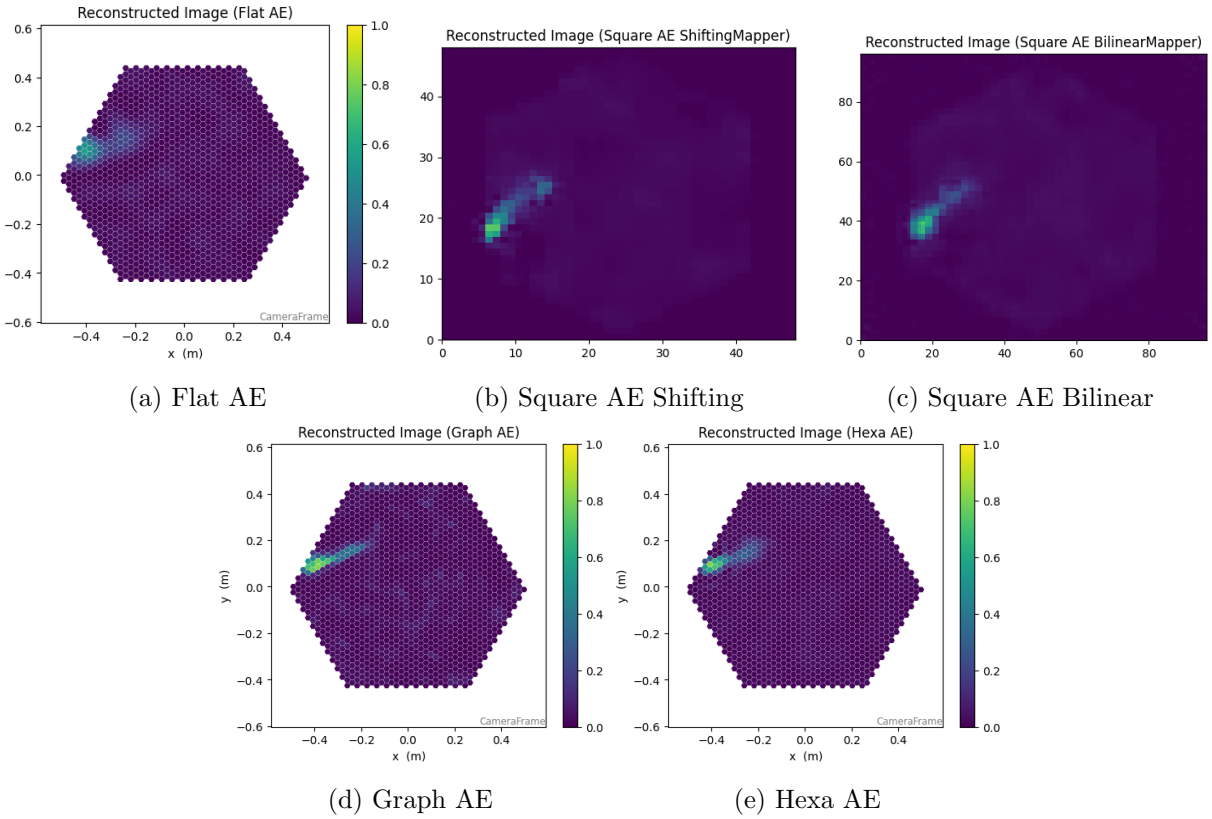


(d) Normalized and BilinearMapper

Figure 7.1: Pre-processing steps applied to the original images before being fed into the models. The images correspond to an event from a proton: (a) original image, (b) normalized image, (c) image mapped using the ShiftingMapper, and (d) image mapped using the BilinearMapper.

Figure 7.1 shows the different pre-processing applied to the images depending on the model used before being fed into the models. Since the normalization removes values below zero and then divides by the maximum value of the image, there is not much visible difference between the original image and the normalized image. However, the images mapped with the two transformation methods show a significant difference from the original image, as the hexagonal pixels are transformed into square pixels, which noticeably alters the image. These images serve as a starting point to understand how the models process the data and why they fail to effectively detect anomalies.

From the reconstructed images obtained with the different models presented in Figure 7.2, we can observe that all models try to reconstruct the background of the image and some of the air



(a) Flat AE (b) Square AE Shifting (c) Square AE Bilinear

(d) Graph AE (e) Hexa AE

Figure 7.2: Comparison of reconstructed images obtained with different autoencoder architectures. The scale is the same for all images for better comparison. The images are reconstructed from the normalized input images shown in Figure 7.1 of a proton event. Each subfigure shows the reconstruction from a different model: (a) Flat AE, (b) Square AE with ShiftingMapper, (c) Square AE with BilinearMapper, (d) Graph AE, and (e) Hexa AE.

shower, but not the fine details of the image such as the brightest part of the shower. The models also attempt to reconstruct the background noise, which is undesirable since noise is random and has no impact on the origin of the event. This suggests that the models have not learned to differentiate important features of proton images, which explains why they fail to effectively detect anomalies. Also, the MSE is currently calculated over all pixels in the image, while the majority of pixels are background noise. This means that the models are primarily optimizing to reconstruct the background noise, which is not useful for anomaly detection. Additionally, for the Square AE, the model must learn the hexagonal shape of the image, which complicates the reconstruction task. Overall, there is no significant difference between the different architectures, all seem to encounter the same issues.

This experiment clearly shows that several points need improvement, such as finding a way to clean the background noise or forcing the model to focus on the important parts of the image.

7.2 Event selection based on Hillas computability

After exploring the data, it turns out that for a significant number of images the Hillas parameters cannot be calculated. This occurs when the recorded air shower is too faint to allow the

definition of a valid image mask, which is a prerequisite for the calculation of Hillas parameters, as described in Chapter 4. In such cases, the air shower is not sufficiently bright, and it becomes impossible to reliably estimate its morphology and orientation. The ability to compute Hillas parameters is therefore closely linked to the quality of the image. Conversely, if Hillas parameters cannot be calculated, it indicates that the image is of low quality, with a faint or poorly defined air shower. This could potentially impact the model's ability to learn relevant features from the data.

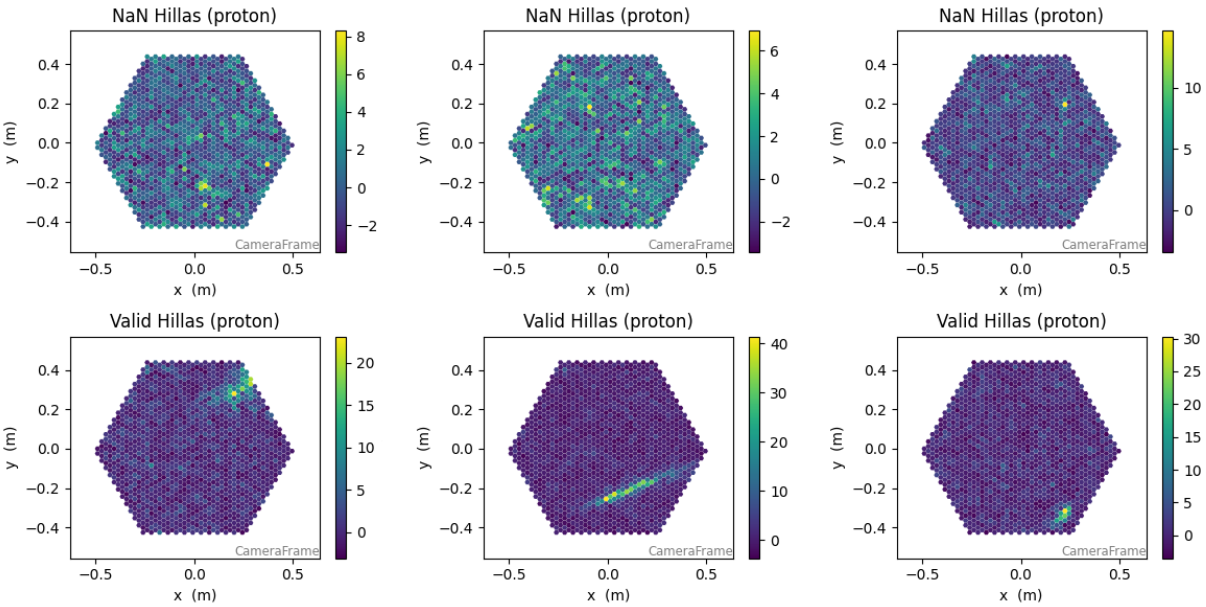


Figure 7.3: Comparison of images with and without Hillas parameters for proton events. The top images show events for which Hillas parameters cannot be calculated, while the bottom images show events with calculable Hillas parameters.

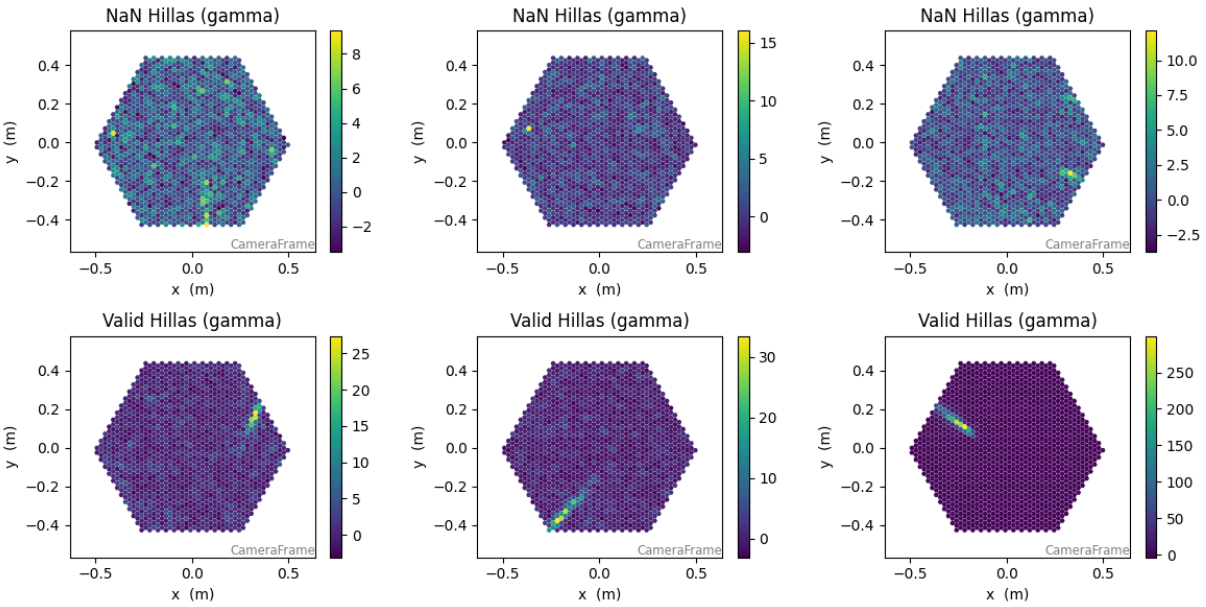


Figure 7.4: Comparison of images where Hillas parameters can and cannot be calculated for gamma events. The top images show events for which Hillas parameters cannot be calculated, while the bottom images show events with calculable Hillas parameters.

Figures 7.3 and 7.4 show examples of images where Hillas parameters can and cannot be calculated for proton and gamma events, respectively. We can observe that events without calculable Hillas parameters are dominated by background noise, and the air shower is either very faint or not visible at all with just one or two pixels showing some activity. In contrast, events with calculable Hillas parameters exhibit more defined structures, where the air shower is clearly visible. At first glance, it therefore seems logical that models would learn better from images with calculable Hillas parameters.

The RF presented in the state of the art use Hillas parameters to perform classification, determine the direction and energy of events (see Chapter 3). In cases where they cannot be calculated, these events are simply excluded from the analysis. Therefore, since images with calculable Hillas parameters are of higher quality, the experiment was redone using only these images to see if it improves the model's performance. This reduces the size of the dataset by approximately 49.3% (from 573,217 to 290,303 images), but the number of events remains sufficient to train the models. This modification is also applied to the test data. However, enough events with calculable Hillas parameters are present to maintain the same total number of events (70,000 for protons and 70,000 for gammas). The same number of events could be maintained because not all available events were used in the previous experiment. It is also important to note that these are not exactly the same events as in the previous experiment, and this should be taken into account when comparing the results.

The names of the models in this section are suffixed with "hillas-based" to differentiate them from the baseline models. This name will also be used in future experiments to describe this quality cut based on Hillas computability.

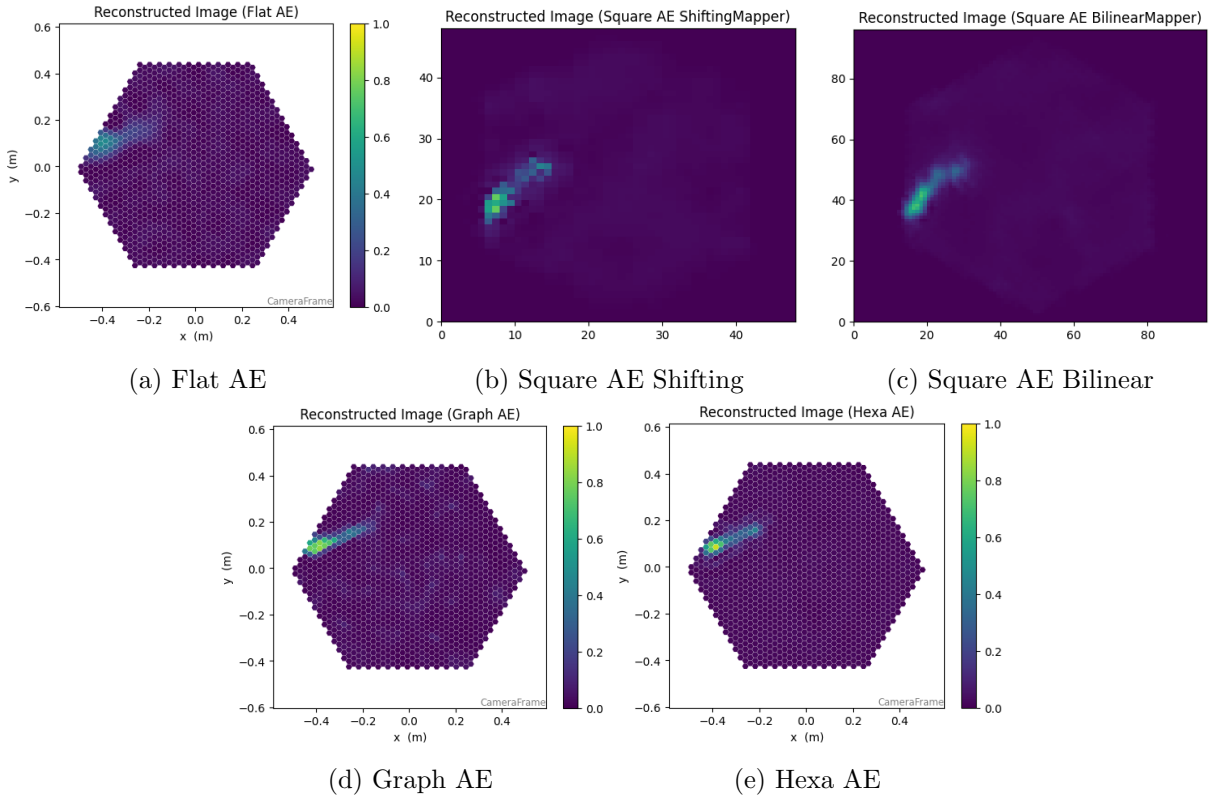
Table 7.2: Model performance metrics using only data with calculable Hillas parameters.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (base)	0.4133	0.44	0.44	0.44	0.44
Square AE Shifting (base)	0.4168	0.44	0.44	0.44	0.44
Square AE Bilinear (base)	0.4182	0.44	0.44	0.44	0.44
Graph AE (base)	0.4200	0.43	0.43	0.43	0.43
Hexa AE (base)	0.4204	0.44	0.44	0.44	0.44
Flat AE (hillas-based)	0.4542	0.46	0.46	0.46	0.46
Square AE Shifting (hillas-based)	0.4554	0.46	0.46	0.46	0.46
Square AE Bilinear (hillas-based)	0.4612	0.47	0.47	0.47	0.47
Graph AE (hillas-based)	0.4692	0.46	0.46	0.46	0.46
Hexa AE (hillas-based)	0.4635	0.47	0.47	0.47	0.47

The results obtained using only events where Hillas parameters are calculable, presented in Table 7.2, show a notable improvement in performance compared to the baseline models. All metrics have increased. However, the values remain very low, and a random model would still perform better. Nevertheless, this experiment confirms that data with a more visible shower impacts the model results.

It is possible that higher quality images allow the models to learn more relevant features of proton events, but it is also possible that the images present in the test dataset are easier to

analyze, which could also explain the improvement in performance. Therefore, it is important to verify whether there has been a real improvement in the model's learning or if the test data are simply easier.



(a) Flat AE (b) Square AE Shifting (c) Square AE Bilinear

(d) Graph AE (e) Hexa AE

Figure 7.5: Comparison of reconstructed images obtained with different autoencoder architectures using only data with calculable Hillas parameters. The scale is the same for all images for better comparison. The images are reconstructed from the normalized input images shown in Figure 7.1 of a proton event. Each subfigure shows the reconstruction from a different model: (a) Flat AE, (b) Square AE with ShiftingMapper, (c) Square AE with BilinearMapper, (d) Graph AE, and (e) Hexa AE.

For a more visual analysis, the reconstructed images obtained using only events with calculable Hillas parameters are presented in Figure 7.5. By comparing these images with those obtained previously (Figure 7.2), we observe that the models reconstruct the air shower better than in the previous case. However, they continue to reconstruct the background noise, which is undesirable. A visual analysis was conducted on other events, and the results are similar. Therefore, it is clear that the models have learned to better reconstruct the images and that the test dataset is not simply easier. It is also interesting to note that Hexa AE and Graph AE seem to reconstruct the images better than the other architectures, which could indicate that these architectures are more suited to this type of data.

It would have been theoretically simpler and more relevant to analyze the quality of the reconstructions using the mean MSE of the reconstructed images on the test dataset. However, since the test images are not the same between the two experiments, this comparison would not make sense.

This experiment shows that selecting higher quality images can improve the model's learning. However, the performance remains very low, indicating that other improvements are necessary to achieve better results.

7.3 Background pixel masking

In previous experiments, it was observed that background noise in the images influenced the models. To address this issue, one possible approach is to apply a mask to the images before training the models. This would set the background pixels to zero and retain only the pixels related to the air shower. In this way, the models would be forced to focus on reconstructing the air shower.

A mask is already available in the DL1 data. It is calculated during the transformation from DL0 to DL1 images (see Chapter 4). This binary mask indicates which pixels are considered part of the air shower (value 1) and which pixels are considered background noise (value 0). The mask is not available for all images, as the air shower must be sufficiently bright to be detected. This is the same issue as for calculating the Hillas parameters. Therefore, images for which a mask is available correspond exactly to the events passing the Hillas-based quality cut. Thus, this experiment is conducted with exactly the same images as in the previous experiment (Section 7.2).

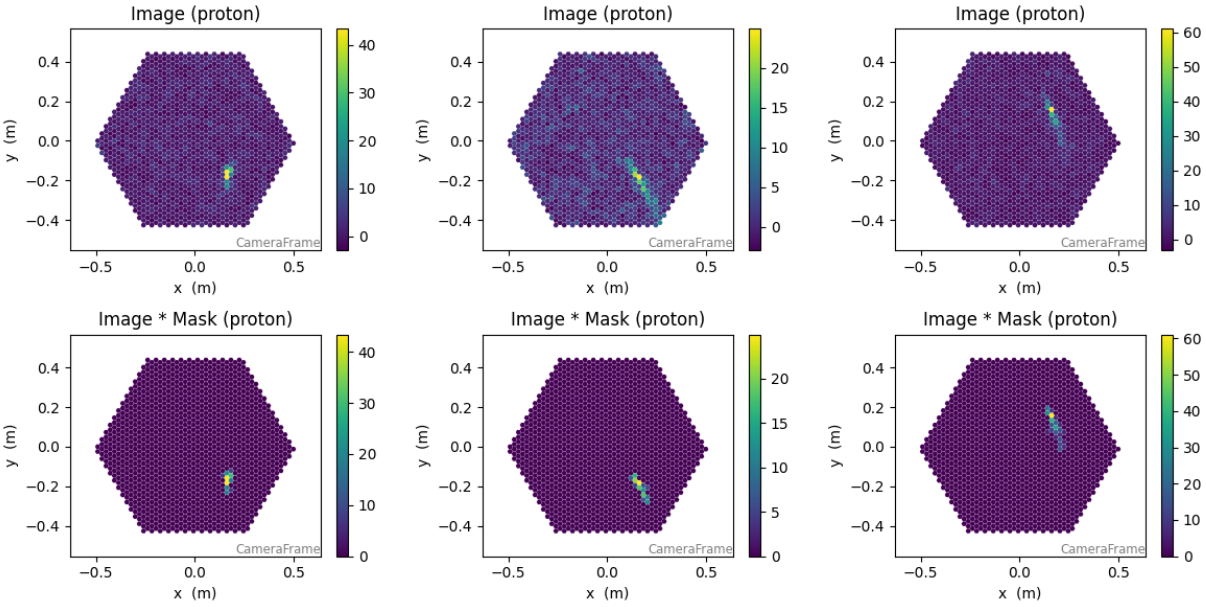


Figure 7.6: Example of applying the mask to proton event images. The top row shows the original images, while the bottom row shows the images after applying the mask, where background pixels are set to zero.

Figures 7.6 and 7.7 show examples of applying the mask to proton and gamma event images, respectively. We can clearly see that the background noise pixels are set to zero, while the pixels related to the air shower are retained. However, it is noticeable in some images (second image in Figure 7.6 and first image in Figure 7.7) that not the entire air shower is always preserved by the mask. Therefore, the mask is not perfect, but it still significantly reduces the amount of

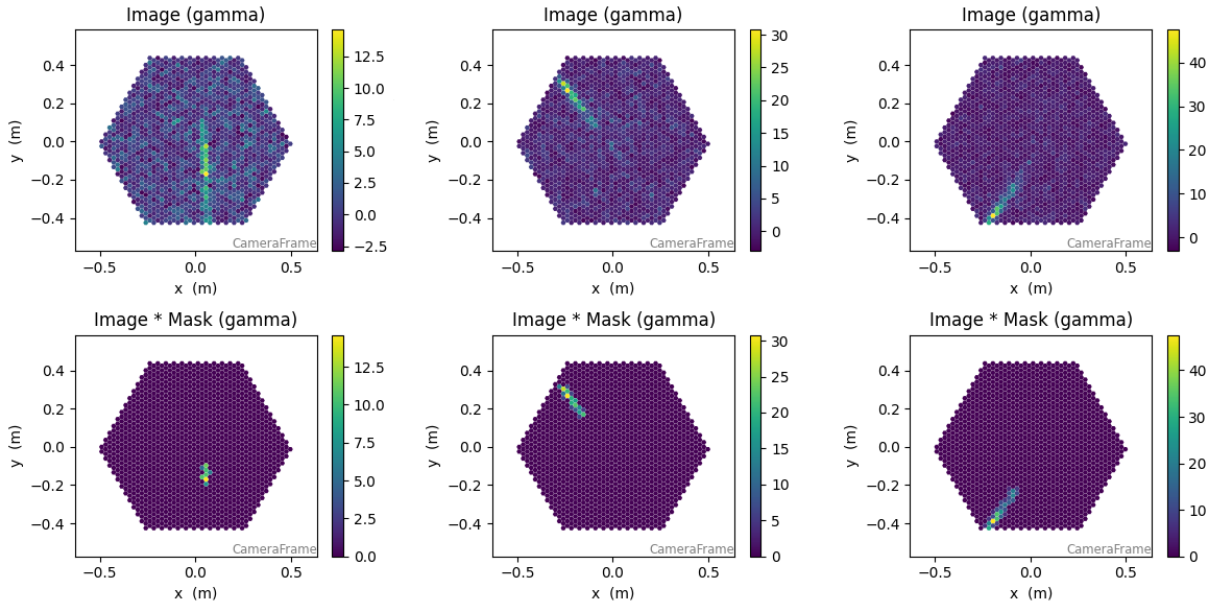


Figure 7.7: Example of applying the mask to gamma event images. The top row shows the original images, while the bottom row shows the images after applying the mask, where background pixels are set to zero.

background noise in the images.

Also, currently, the reconstruction error is calculated over all pixels in the image, while the majority of pixels are background noise. By applying a mask, it is possible to calculate the error only on the pixels belonging to the air shower, which removes the influence of background noise. The error calculation thus changes from (5.1) to:

$$\text{MSE}_{\text{masked}} = \frac{1}{M} \sum_{i=1}^{N} m_i (x_i - \hat{x}_i)^2 \tag{7.1}$$

where:

- N is the total number of pixels
- M is the number of important pixels (i.e., the pixels where the mask m_i equals 1)
- x_i is the value of the original pixel
- $\hat{x}_i$ is the value of the reconstructed pixel
- m_i is the binary mask that indicates whether pixel i is important (1) or background noise (0)

This change in the calculation of the reconstruction error allows us to assess only the model's ability to reconstruct the air shower.

The Square AE model is no used from this experiment onwards. To use a mask, it would be necessary to apply it before transforming the hexagonal image into a square image. This poses a problem when calculating the reconstruction error, as the binary mask is defined in the hexagonal space. It would then be necessary to retransform the reconstructed square image back into a hexagonal image to apply the mask. However, the inverse function of the mappers

used (ShiftingMapper and BilinearMapper) is not available, so this transformation cannot be performed. The other solution would be to transform the hexagonal mask into a square mask, but it would no longer be a binary mask, as the transformation to square space relies on interpolation operations. The values of the square mask would therefore be a continuous value between zero and one, which would introduce uncertainty about the definition of important pixels. Also, this model does not add much compared to the other architectures: Graph AE is simpler and Hexa AE natively handles hexagonal images. It is therefore simpler and more relevant to abandon it at this stage.

The results obtained are therefore only for the Flat AE, Graph AE, and Hexa AE models. They are trained on the masked images, and the reconstruction error is calculated only on the important pixels, following the same procedure as before. The data used are the same as in the previous experiment (Section 7.2), and this will also be the case for all subsequent experiments where a mask is applied. The names of the models in this section are suffixed with "mask" to differentiate them from the previously used models.

Table 7.3: Model performance metrics using masked images.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (hillas-based)	0.4542	0.46	0.46	0.46	0.46
Graph AE (hillas-based)	0.4692	0.46	0.46	0.46	0.46
Hexa AE (hillas-based)	0.4635	0.47	0.47	0.47	0.47
Flat AE (mask)	0.6254	0.59	0.59	0.59	0.59
Graph AE (mask)	0.5821	0.57	0.57	0.57	0.57
Hexa AE (mask)	0.5759	0.55	0.55	0.55	0.55

The results obtained using masked images and calculating the reconstruction error only on important pixels, presented in Table 7.3, show a significant improvement in performance compared to the previous models. The models now perform better than random guessing, which is a considerable progress. The Flat AE model achieves the best results, followed by the Graph AE and Hexa AE. The results are still not very high, but this experiment shows that masking the images and calculating the reconstruction error only on important pixels are promising avenues to improve anomaly detection.

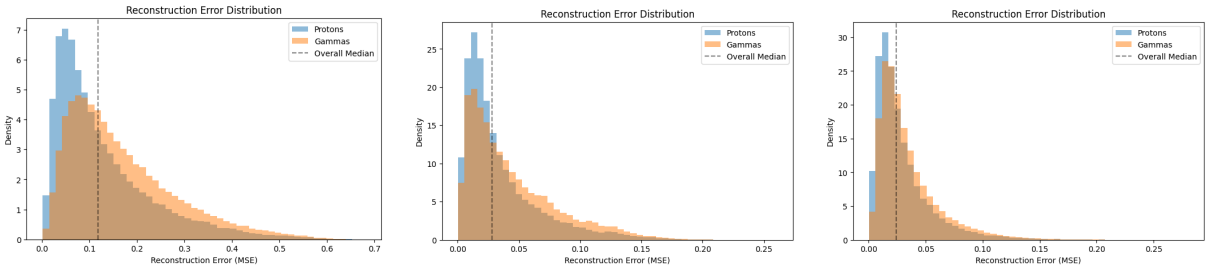
For the first time, the reconstruction error is calculated on exactly the same data for each model, so it is possible to compare the models with each other using this metric to see how well they reconstruct the images. Previously, this was not possible, as Square AE calculated the error on images that had undergone a hexagonal to square transformation, Hexa AE calculated the error on square images with hexagonal pixels (see Chapter 6), and Graph AE and Flat AE calculated the error on the original images. It is important to note that this comparison remains limited to models having exactly the same input and output data. Otherwise, differences in pre-processing can skew the comparison. Therefore, it is useless to compare future and previous experiments using this metric if the input or output data differ.

Table 7.4 shows that, as expected, the median reconstruction errors for protons are lower than those for gammas for each model. This confirms that the models have learned to better recon-

Table 7.4: Reconstruction error statistics on masked images. P stands for protons and G for gammas.

Model	Median P	Median G	Mean P	Mean G	Std P	Std G
Flat AE (mask)	0.0954	0.1405	0.1290	0.1691	0.1053	0.113o
Graph AE (mask)	0.0239	0.0335	0.0351	0.0444	0.0319	0.0361
Hexa AE (mask)	0.0217	0.0265	0.0293	0.0345	0.0250	0.0267

struct proton images. Additionally, we notice that the Graph AE and Hexa AE models achieve significantly lower reconstruction errors than the Flat AE model, suggesting that they are more effective at reconstructing the images. However, the Flat AE model achieves better results in terms of anomaly detection (AUC and Accuracy). This could indicate that Graph AE and Hexa AE are too proficient in reconstruction, making it more difficult to distinguish between protons and gammas via the reconstruction error.



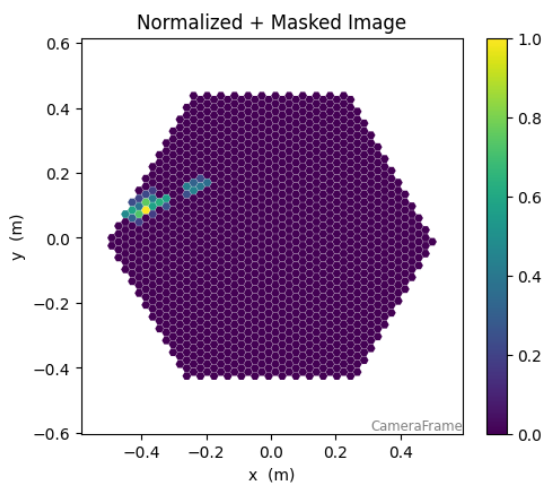
(a) Flat AE (b) Graph AE (c) Hexa AE

Figure 7.8: Reconstruction error distribution on masked images for proton and gamma events. Each subfigure shows the distribution for a different model: (a) Flat AE, (b) Graph AE, and (c) Hexa AE.

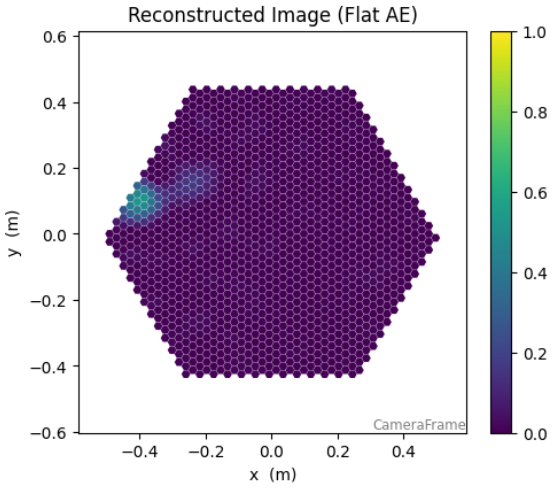
This is in line with Figure 7.8, which shows the distribution of reconstruction errors for protons and gammas. We can see that, for the Graph AE and Hexa AE models, the distributions of reconstruction errors for protons and gammas overlap more than for the Flat AE model. This means that it is more difficult to distinguish between the two types of events using the reconstruction error with these models. The overlap remains very high, which explains the still limited performance.

Figure 7.9 presents an example of an image of a proton event reconstructed by the different models using masked images. There is no longer any background noise in the input images, so the models focus solely on reconstructing the air shower. We can observe that the models manage to reconstruct the air shower more accurately than in previous experiments. Especially, the high intensity pixels are better reconstructed. Also, we notice that as previously assumed, the Graph AE and Hexa AE models reconstruct the images very well, while the Flat AE model struggles more. Reducing the reconstruction capacity of the Graph AE and Hexa AE models could potentially improve their performance in anomaly detection.

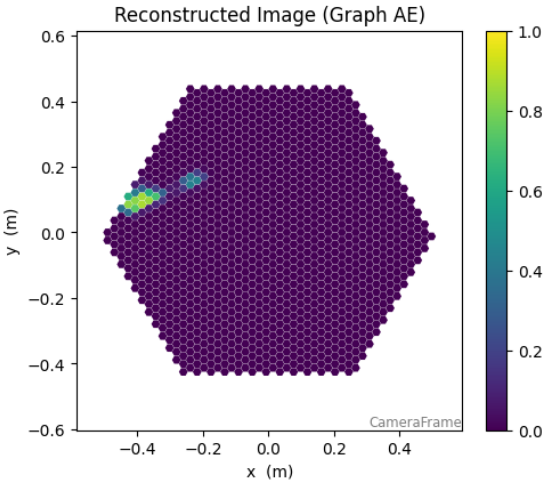
The application of the mask to the images and the calculation of the reconstruction error only on important pixels significantly improve the models' performance in anomaly detection. This experiment highlights the importance of focusing on relevant features in the data and removing



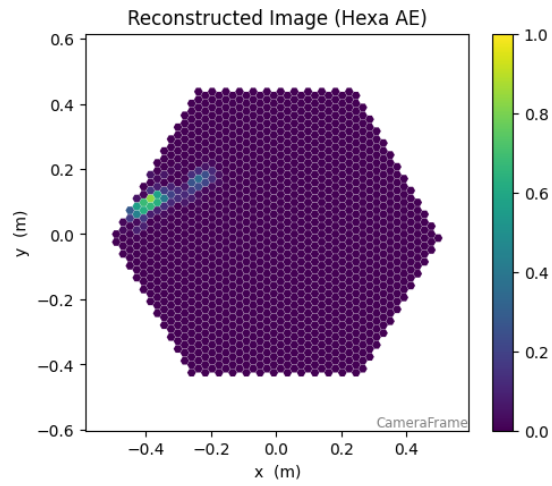
(a) Normalized and Masked Input



(b) Flat AE



(c) Graph AE



(d) Hexa AE

Figure 7.9: Comparison of reconstructed images obtained with different autoencoder architectures using a masked image of a proton event as input. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and masked input images shown in subfigure (a). Each subfigure shows the reconstruction from a different model: (b) Flat AE, (c) Graph AE, and (d) Hexa AE.

background noise to enhance model learning. For the future experiments, if the mask is used, the reconstruction error will always be calculated only on important pixels.

7.4 Temporal information from peak times

The use of the mask allows for the utilization of another type of information present in the DL1 data: the peak arrival time of each pixel (peak time). This information is a temporal indicator in nanoseconds that represents the moment when the pixel's signal reaches its maximum during the event. Its value can range from 0 to 196 ns, corresponding to the sampling window of 50 samples with a time resolution of four ns per sample. This information can be useful because the pixels corresponding to the air shower generally have more correlated peak times, while the

background noise pixels have random peak times.

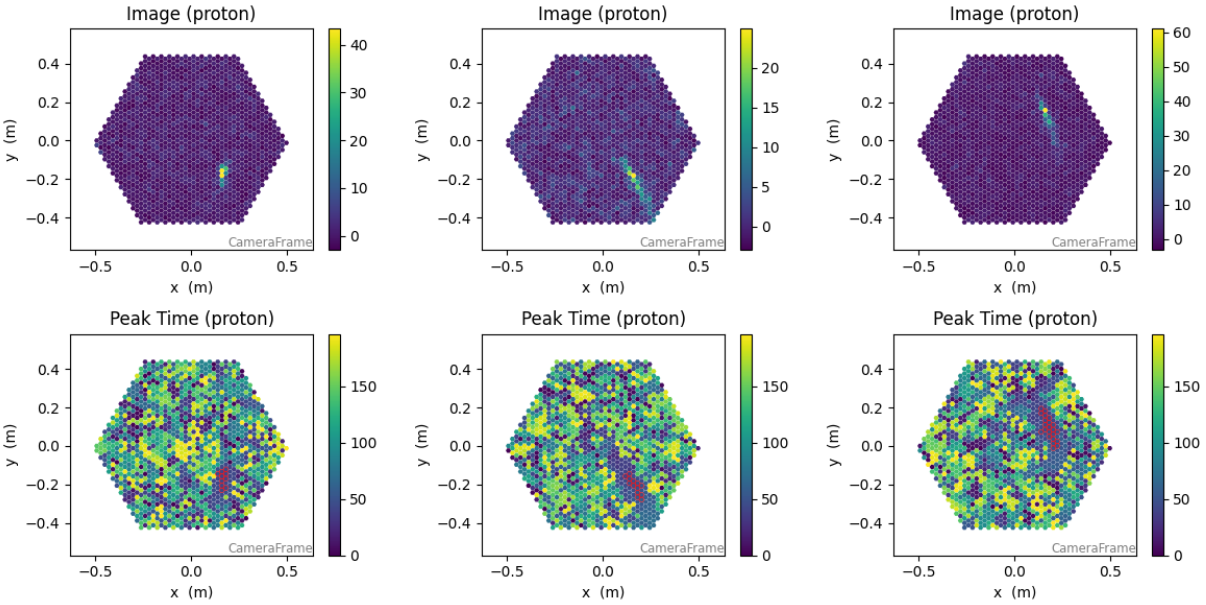


Figure 7.10: Example of peak time information for proton event images. The top row shows the original images, while the bottom row shows the corresponding peak time information for each pixel. The pixels outlined in red indicate the location of the mask.

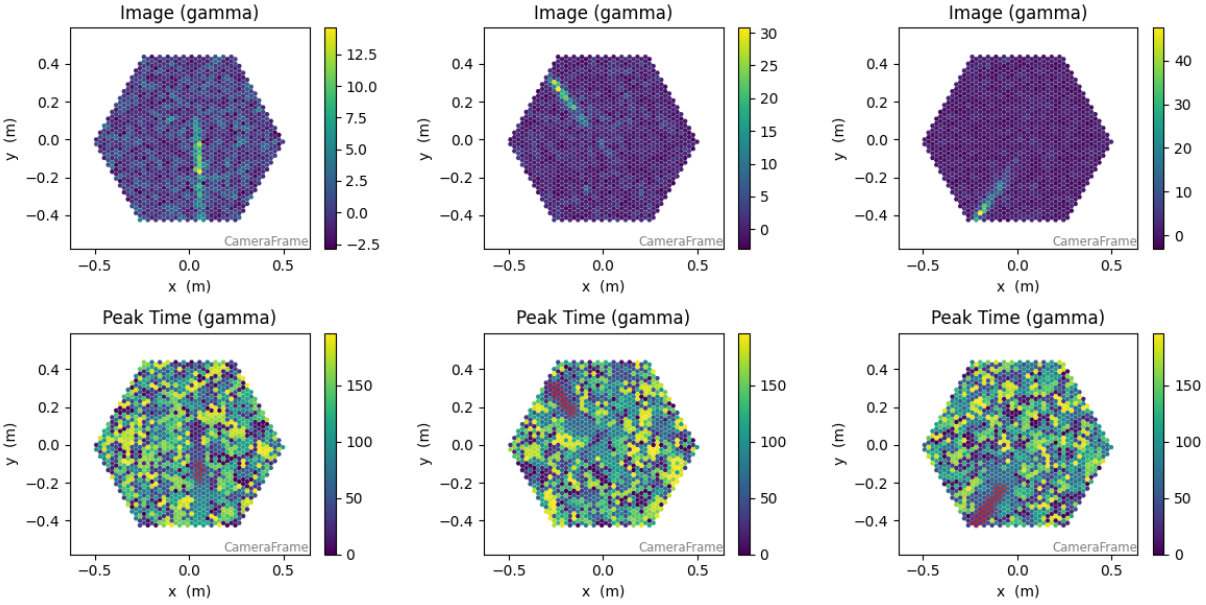


Figure 7.11: Example of peak time information for gamma event images. The top row shows the original images, while the bottom row shows the corresponding peak time information for each pixel. The pixels outlined in red indicate the location of the mask.

Figures 7.10 and 7.11 show examples of proton and gamma event images with the corresponding peak time information. We can observe that the area where the air shower is located exhibits more correlated peak times, while the background noise pixels have completely random peak times. Without the mask, this information would have been unusable, as the background noise would have dominated the peak time. With the mask, it is possible to retain only the important

pixels and use this information to help the models better reconstruct the peak time. However, in all peak time images, we can clearly see that the air shower area is not completely covered by the mask. This phenomenon is much more visible on the peak times than on the images, as some pixels of the air shower have a very weak signal but a well-defined peak time.

The normalization is different from that used for the images. Indeed, after masking, the retained pixels have values very close to each other (for example from 80 to 92), while the background noise pixels will all be at zero. If we simply normalized between zero and one, the majority of important pixels would end up in a very narrow range close to one. To avoid this, a new minimum value for the important pixels was calculated, it corresponds to the minimum peak time value among the important pixels minus four ns (one sample). Then, values below this new minimum are clipped to this minimum value. Finally, Finally, normalization is performed between zero and one with this new minimum and the maximum peak time value among the important pixels. This method allows for a better distribution of the values of important pixels between zero and one, while keeping the background noise pixels at zero.

For this experiment, the input is solely the normalized and masked peak time information, and the model is tasked with reconstructing this same information. The calculation of the reconstruction error is also performed only on the important pixels, as done previously. The names of the models in this section are suffixed with "peak" to differentiate them from the previously used models.

Table 7.5: Model performance metrics using peak time information.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (mask)	0.6254	0.59	0.59	0.59	0.59
Graph AE (mask)	0.5821	0.57	0.57	0.57	0.57
Hexa AE (mask)	0.5759	0.55	0.55	0.55	0.55
Flat AE (peak)	0.6254	0.60	0.60	0.60	0.60
Graph AE (peak)	0.6222	0.60	0.60	0.60	0.60
Hexa AE (peak)	0.5794	0.56	0.56	0.56	0.56

The results obtained using the peak time information, presented in Table 7.5, show a slight improvement in performance for the Flat AE and Graph AE models compared to the previous models using masked images. The Hexa AE model shows only a marginal improvement. This experiment indicates that the peak time information can be beneficial for anomaly detection, particularly for certain architectures.

The median reconstruction error and reconstructed images are not presented here, as the results are practically identical to those obtained previously with masked images and do not provide additional insights. The Flat AE has a higher reconstruction error than Graph AE and Hexa AE, and the reconstructed images show that Graph AE and Hexa AE reconstruct the images very well, while Flat AE struggles more.

This experiment mainly demonstrates that temporal information can be exploited. However, the analysis of the masking shows that the air shower is not completely covered by the mask, and there is room for improvement. Better coverage of the air shower would allow the full

exploitation of both temporal and image information.

7.5 Extended mask using temporal information

The previous experiments showed that the mask used did not always include the entire air shower, which prevents fully exploiting both temporal and image information. To address this issue, a new image cleaning algorithm was developed based on temporal neighborhood propagation of pixels. The idea is to start with a mask that has detected the presence of the air shower, then extend it over the entire shower, using the temporal information of neighboring pixels.

In the SST-1M pipeline [46], the mask used is not the same as the one provided in the DL1 data. The same image cleaning function is applied, but after another function is used to extend the mask based on the peak time information (see `time_delta_cleaning` [47]). This function adds pixels to the mask if they are neighbors of already masked pixels and if their peak time difference is below a certain threshold. The result is therefore a different mask from the one used previously. It was also tested, but it suffers from the same problems as the one provided in the DL1 data, namely that it does not always include the entire air shower, because the mask is only extended once. A mask extension algorithm was therefore developed.

Listing 7.1: Improved image cleaning algorithm based on temporal neighborhood propagation

```
def clean_image_improvement(peak, mask, adj_list, max_diff=5):
    new_mask = mask.copy().astype(np.uint8)
    queue = deque(np.where(new_mask == 1)[0])

    while queue:
        i = queue.popleft()
        for n in adj_list[i]:
            if new_mask[n] == 0 and abs(peak[i] - peak[n]) <= max_diff:
                new_mask[n] = 1
                queue.append(n)

    return new_mask
```

The algorithm presented in Listing 7.1 illustrates the improved image cleaning method based on temporal neighborhood propagation that was developed. It starts by using the initial mask to identify the pixels belonging to the air shower. Then, it iterates through all the pixels in the mask and examines their neighbors using a pre-calculated adjacency list. If a neighboring pixel is not yet included in the mask and the difference in peak time between the current pixel and the neighboring pixel is below a defined threshold (`max_diff`), then the neighboring pixel is added to the mask. This process is repeated until all relevant pixels are included in the mask. This method effectively extends the mask to cover the entire air shower by leveraging temporal information. There is no risk of an infinite loop, as each pixel can only be added to the mask once and is removed from the queue once processed. The difference with the standard image cleaning algorithm is that it does not stop after one pass, but continued to propagate until no new pixels could be added.

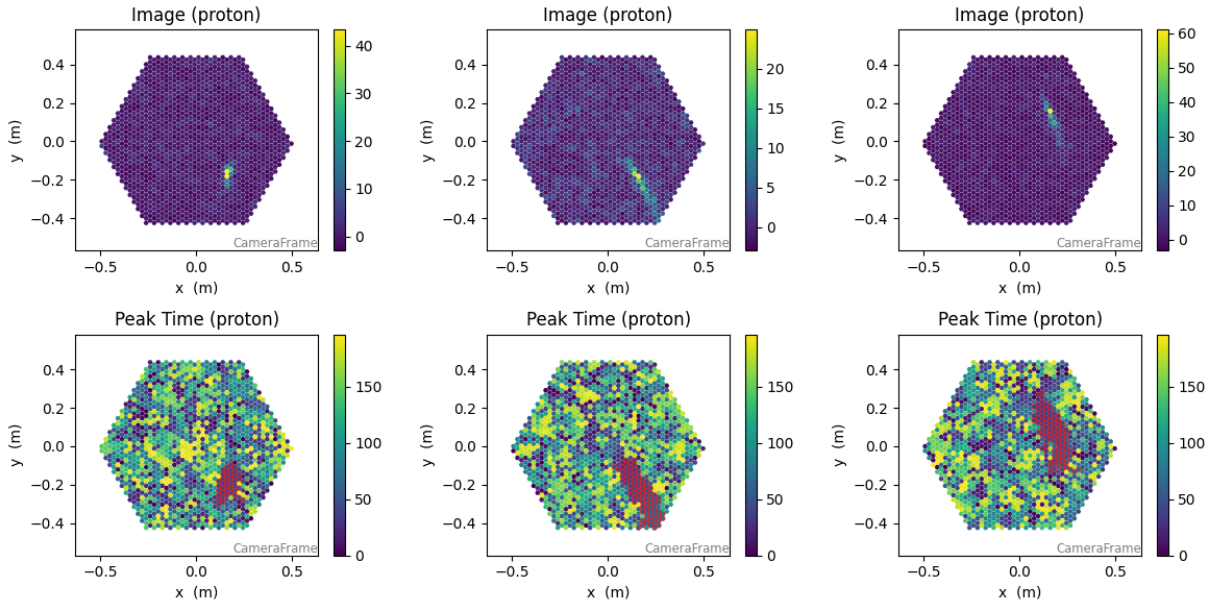


Figure 7.12: Example of applying the extended mask to proton event images. The top row shows the original images, while the bottom row shows the peak time with the corresponding extended mask outlined in red. Figures 7.6 and 7.7 used the standard mask for comparison.

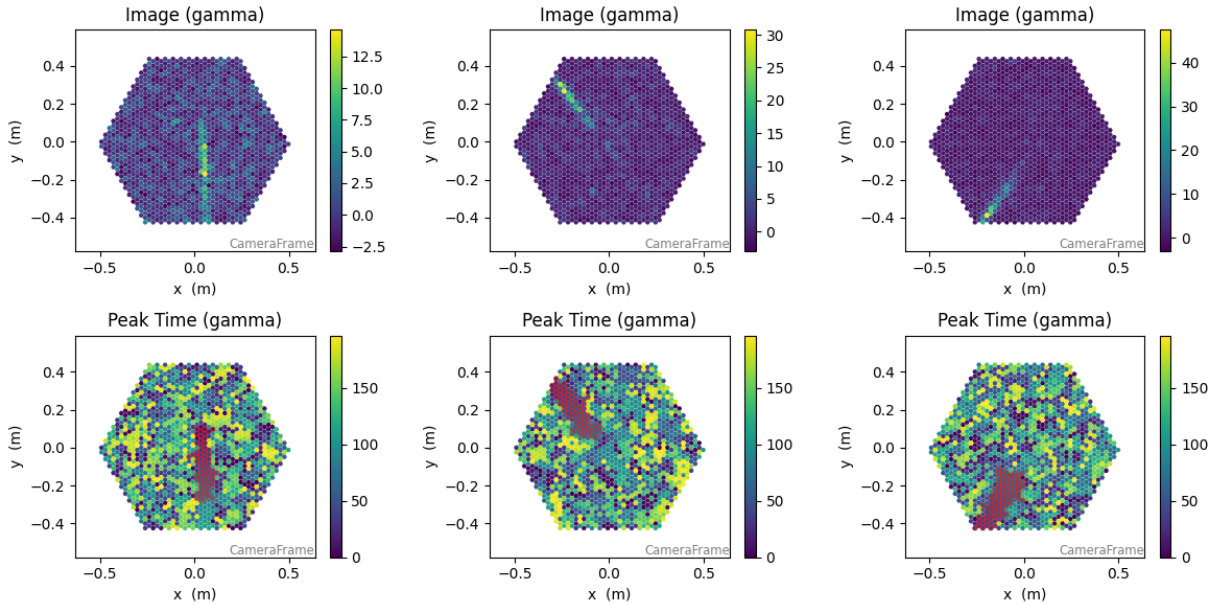


Figure 7.13: Example of applying the extended mask to gamma event images. The top row shows the original images, while the bottom row shows the peak time with the corresponding extended mask outlined in red. Figures 7.6 and 7.7 used the standard mask for comparison.

Figures 7.12 and 7.13 show examples of applying the extended mask to proton and gamma event images, respectively. By comparing these images with those obtained previously (Figures 7.6 and 7.7), we observe that the extended mask completely covers the air shower and no background noise pixels are retained. This improvement thus allows for fully exploiting the information provided by the air shower, both in terms of images and peak times.

The models used in this experiment are the same as before: Flat AE, Graph AE, and Hexa

AE. They are trained on the extended masked images, and the reconstruction error is calculated only on the important pixels, as before. The names of the models in this section are suffixed with "extended mask" to differentiate them from those used previously. For the input data, two configurations were tested: using the images or using the peak times, so the model names are suffixed with "image" or "peak" accordingly.

Table 7.6: Model performance metrics using images with extended masks and peak time information.

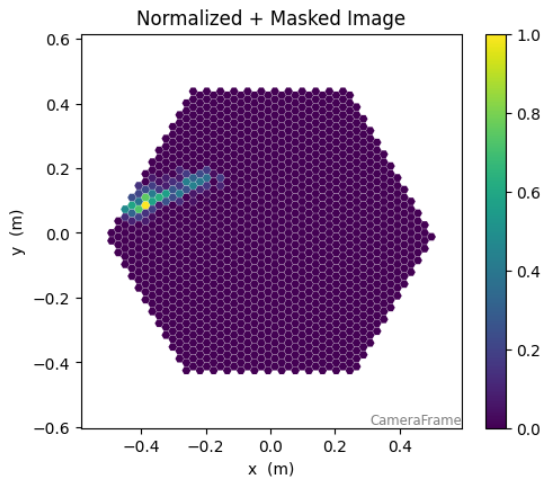
Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (mask)	0.6254	0.59	0.59	0.59	0.59
Graph AE (mask)	0.5821	0.57	0.57	0.57	0.57
Hexa AE (mask)	0.5759	0.55	0.55	0.55	0.55
Flat AE (peak)	0.6254	0.60	0.60	0.60	0.60
Graph AE (peak)	0.6222	0.60	0.60	0.60	0.60
Hexa AE (peak)	0.5794	0.56	0.56	0.56	0.56
Flat AE (extended mask image)	0.6416	0.61	0.61	0.61	0.61
Graph AE (extended mask image)	0.6362	0.60	0.60	0.60	0.60
Hexa AE (extended mask image)	0.6417	0.60	0.60	0.60	0.60
Flat AE (extended mask peak)	0.6398	0.60	0.60	0.60	0.60
Graph AE (extended mask peak)	0.6383	0.60	0.60	0.60	0.60
Hexa AE (extended mask peak)	0.6148	0.58	0.58	0.58	0.58

The results, presented in Table 7.6, show an improvement in performance for all models compared to those using the simple mask. The improvement is more pronounced for models using images as input data. The most notable improvement is achieved by the Hexa AE model using images with the extended mask, reaching an AUC of 0.6417 compared to 0.5759 previously. It thus achieves results similar to Flat AE and Graph AE. Models using peak time information yield lower results than their image-based counterparts this time, contrary to the previous experiment. The complete coverage of the air shower by the mask must allow for fully exploiting the information contained in the images, which seem more relevant than peak times for this task at first glance.

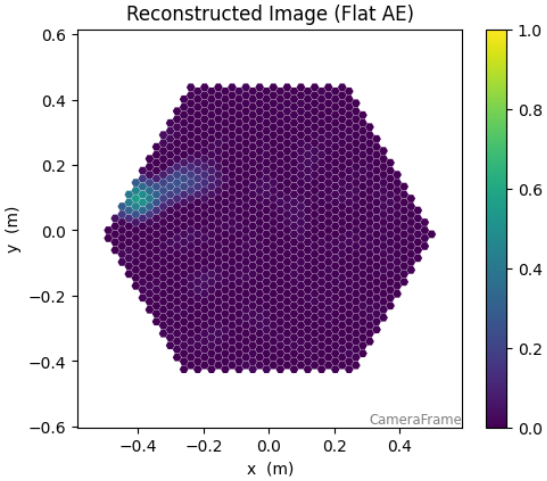
Figure 7.14 presents an example of an image reconstructed by the different models using extended masked images. Once again, we observe that the Hexa AE and Graph AE models reconstruct the images very well, while the Flat AE struggles more. However, the results are generally better than those obtained previously with the simple mask (Figure 7.9), which confirms that using the extended mask improves the quality of the reconstructions.

Figure 7.15 presents an example of peak time information reconstructed by the different models using extended masked peak time information. This time, all three models yield similar results in terms of reconstruction quality. The pattern is also easier to reconstruct than that of the images, which probably explains why the performance differences between the models are less pronounced than for the images.

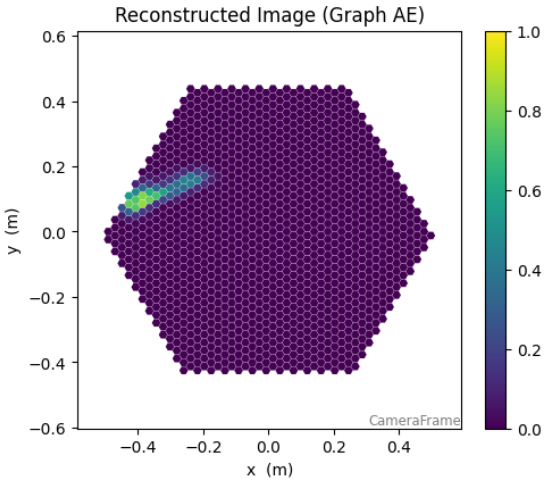
After this visual analysis, we see that Graph AE and Hexa AE have a very high reconstruction capacity, even with the extended mask. It would be interesting to reduce this capacity to see if it improves anomaly detection. This experiment demonstrates that using an extended mask allows for fully exploiting the information contained in the images, which helps improve anomaly



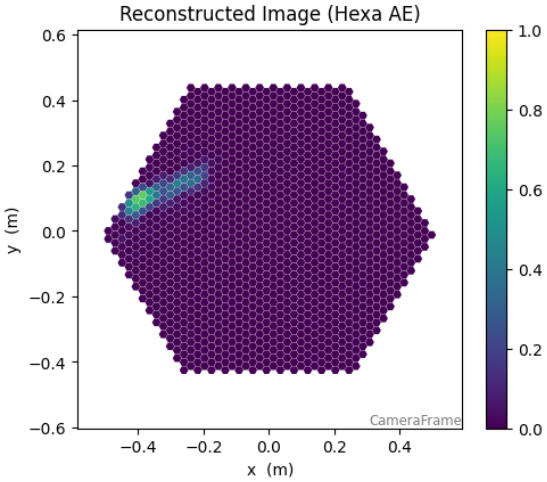
(a) Normalized and Masked Input (Image)



(b) Flat AE



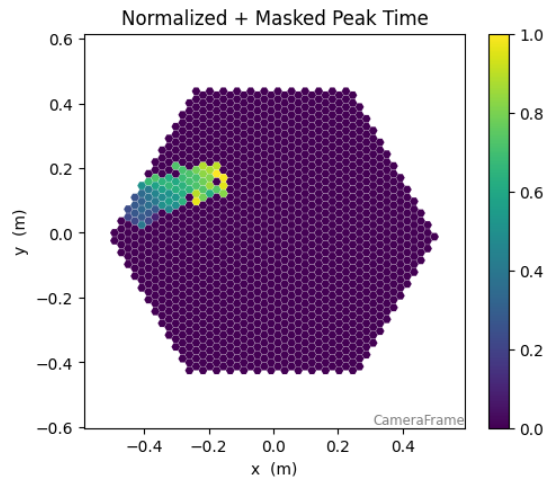
(c) Graph AE



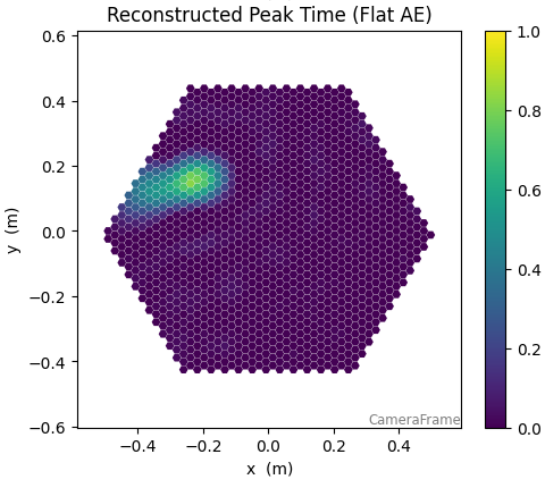
(d) Hexa AE

Figure 7.14: Comparison of reconstructed images obtained with different autoencoder architectures using a masked image of a proton event as input. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input images shown in subfigure (a). Each subfigure shows the reconstruction from a different model: (b) Flat AE, (c) Graph AE, and (d) Hexa AE.

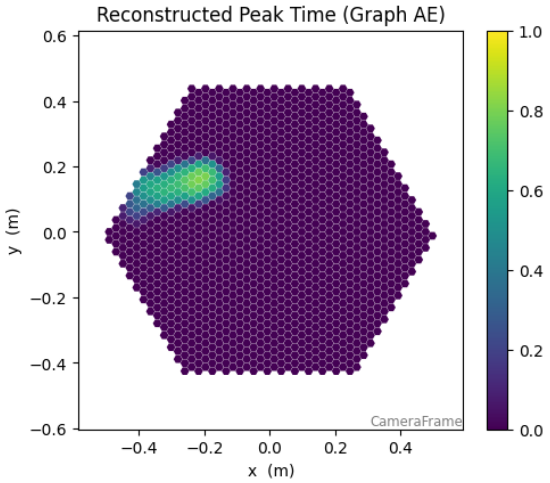
detection. The result on the peak time information is less conclusive, but it still shows that this information can be useful.



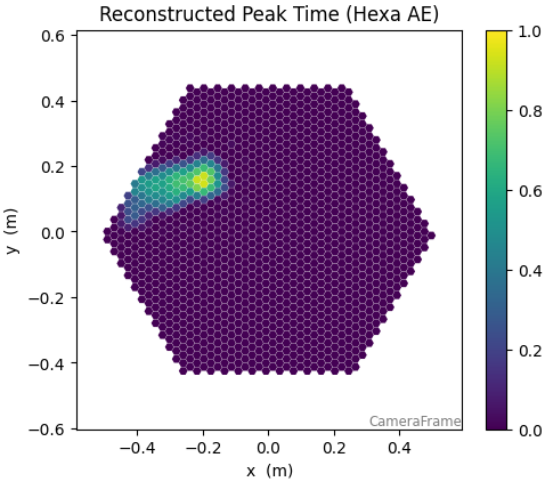
(a) Normalized and Masked Input (peak time)



(b) Flat AE



(c) Graph AE



(d) Hexa AE

Figure 7.15: Comparison of reconstructed peak time information obtained with different autoencoder architectures using a masked peak time of a proton event as input. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input peak time information shown in subfigure (a). Each subfigure shows the reconstruction from a different model: (b) Flat AE, (c) Graph AE, and (d) Hexa AE.

7.6 Latent space reduction

The previous experiments showed that the Graph AE and Hexa AE models had a very high reconstruction capacity, while Flat AE struggled much more. However, Flat AE achieved better results in anomaly detection. It is therefore possible that the high reconstruction capacity of Graph AE and Hexa AE makes it more difficult to distinguish between proton and gamma events. To test this hypothesis, the size of the latent space of the Graph AE and Hexa AE models was reduced to decrease their reconstruction capacity. The idea is that this could force the models to focus on the most important features of the images, which could improve anomaly detection.

For Graph AE, one and two features per node were tested, compared to four features previously.

For Hexa AE, latent space sizes of 32 and 16 were tested compared to 64 previously. No tests with Flat AE were performed, as its reconstruction capacity is already low. The architectures used are the same as before, described in Chapter 6, visible in Figures 6.8 and 6.14, but with the modified latent space size.

The models are trained on the extended masked images, and the reconstruction error is calculated only on the important pixels, as before. The names of the models in this section are suffixed with "X", where X is the size of the latent space, to differentiate them from those used previously.

Table 7.7: Model performance metrics using images with extended masks with reduced latent space size.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Graph AE (extended mask image)	0.6362	0.60	0.60	0.60	0.60
Graph AE (2)	0.6412	0.60	0.60	0.60	0.60
Graph AE (1)	0.6450	0.61	0.61	0.61	0.61
Hexa AE (extended mask image)	0.6417	0.60	0.60	0.60	0.60
Hexa AE (32)	0.6328	0.60	0.60	0.60	0.60
Hexa AE (16)	0.6362	0.60	0.60	0.60	0.60

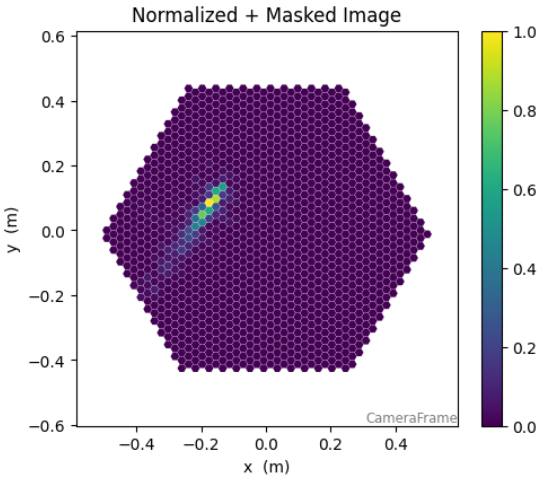
The results, presented in Table 7.7, show that reducing the latent space size slightly improves the AUC for the Graph AE model and slightly decreases the AUC for the Hexa AE model. However, the differences are very small and may be due to chance. Therefore, it is difficult to conclude that reducing the latent space size has a significant impact on anomaly detection. To further analyze this, we need to check the impact of this reduction on the reconstruction error.

Table 7.8: Reconstruction error statistics on extended masked images for protons and gammas with reduced latent space size. P stands for protons, G for gammas and Ext M for extended mask.

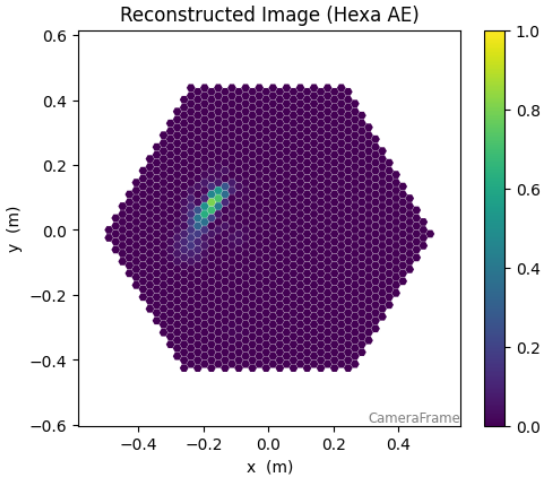
Model	**Median P**	**Median G**	**Mean P**	**Mean G**	**Std P**	**Std G**
Graph AE (Ext M image)	0.0088	0.0129	0.0112	0.0157	0.0098	0.0117
Graph AE (2)	0.0089	0.0133	0.0115	0.0159	0.0099	0.0114
Graph AE (1)	0.0088	0.0132	0.0112	0.0157	0.0095	0.0111
Hexa AE (Ext M image)	0.0149	0.0200	0.0171	0.0221	0.0113	0.0121
Hexa AE (32)	0.0156	0.0207	0.0181	0.0231	0.0124	0.0131
Hexa AE (16)	0.0181	0.0242	0.0210	0.0267	0.0139	0.0142

Table 7.8 shows that reducing the latent space size increases the median reconstruction error for Hexa AE, which is expected as the model's reconstruction capacity is reduced. However, for Graph AE, the median reconstruction error remains practically unchanged, suggesting that changing the number of features per node did not have a significant impact on the model's reconstruction capacity. This is not so surprising, as Graph AE does not reduce its number of nodes but rather the number of features per node, allowing the model to maintain a high reconstruction capacity. It does not perform direct compression like Hexa AE or Flat AE but modifies these nodes based on the characteristics of neighboring nodes. Another approach may be necessary to effectively reduce the reconstruction capacity of Graph AE.

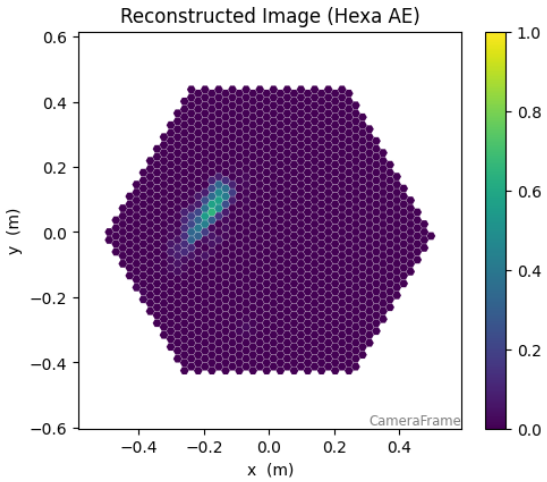
Figure 7.16 presents an example of an image reconstructed by the Hexa AE model with different



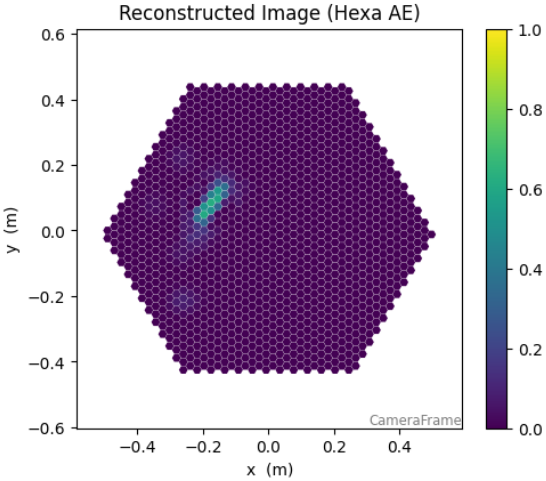
(a) Normalized and Masked Input (Image)



(b) Hexa AE (previous experiment)



(c) Hexa AE (32)



(d) Hexa AE (16)

Figure 7.16: Comparison of reconstructed images obtained with Hexa AE using different latent space sizes on a masked image of a proton event as input. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input images shown in subfigure (a). Each subfigure shows the reconstruction from a different Hexa AE: (b) Hexa AE (previous experiment with latent space size 64), (c) Hexa AE (32), and (d) Hexa AE (16).

latent space sizes. We observe that the reconstruction quality slightly decreases as the latent space size is reduced, which is expected. The differences are not very pronounced, but we can see that it is mainly the high-energy values that are less well reconstructed when the latent space size decreases. The areas with low values remain well reconstructed. This visual analysis mainly confirms the previous results and shows which part of the image is most affected by the reduction in latent space size.

No example of an image reconstructed by Graph AE is presented here, as the reconstructions are practically identical, regardless of the number of features per node. This result is consistent with the median reconstruction error, which does not change significantly.

These experiments indicate that reducing the latent space size has a limited impact on anomaly detection performance. While it slightly improves the AUC for Graph AE, it does not significantly affect Hexa AE. Further investigation is needed to find more effective ways to control the reconstruction capacity of these models to enhance their anomaly detection capabilities.

7.7 Logarithmic normalization for masked images

In previous experiments, the normalization used for the images was a min-max normalization per image, where each pixel is scaled between zero and one based on the minimum and maximum values of the image. However, this results in the vast majority of pixels having values very close to zero, with only a small number of pixels corresponding to the intensity peaks of the air shower having significantly higher values. This distribution of values can make learning more difficult for the models, as they have to focus on a small range of values.

During the initial experiments, a logarithmic normalization was considered to address this issue. However, the model would then focus too much on the low-value pixels, which represent background noise, and neglect the high-value pixels corresponding to the air shower. For this reason, this normalization was not adopted. However, with the use of the mask, it is now possible to ignore the background noise pixels and focus solely on the important pixels. Therefore, this logarithmic normalization was retried in this context.

To be more precise, the logarithmic normalization is performed by applying the following transformation to each important pixel (after applying the mask):

$$x' = \log(1 + x) \tag{7.2}$$

where:

- x is the original pixel value,
- x' is the normalized pixel value.
- log is the natural logarithm.

This is the `log1p` function available in `NumPy` [48]. The `+1` allows handling zero values, as the logarithm of zero is undefined. In this case, the logarithm of one is equal to zero, so pixels with a value of zero remain at zero after the transformation. This transformation compresses the range of pixel values, reducing the impact of high values while preserving differences between low and medium values.

Figures 7.17 and 7.18 show a comparison between min-max normalization and logarithmic normalization for proton and gamma event images, respectively. We can observe that logarithmic normalization allows for a better distribution of the important pixel values across a wider range, while min-max normalization compresses most of the important pixel values close to one. With this normalization, the models can focus on a wider range of important pixel values, which may help improve learning and anomaly detection performance.

The models used in this experiment are the same as before: Flat AE, Graph AE, and Hexa AE.

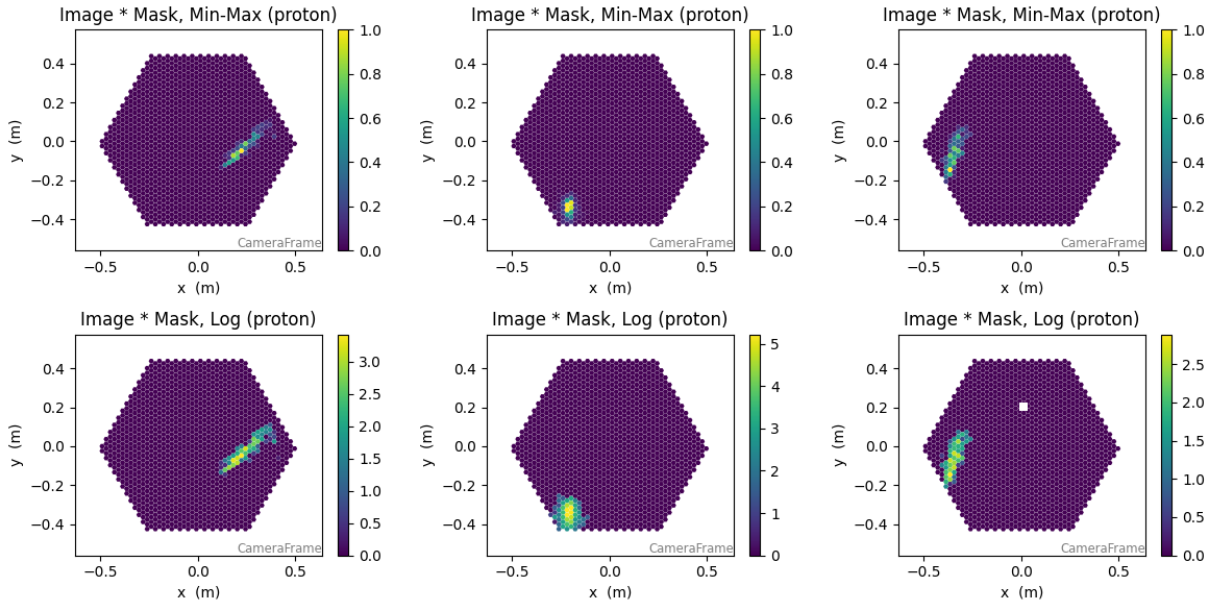


Figure 7.17: Comparison of min-max and logarithmic normalization for proton event images. The top row shows the image after min-max normalization, while the bottom row shows the same image after logarithmic normalization. Both normalizations are applied after masking, so only important pixels are considered.

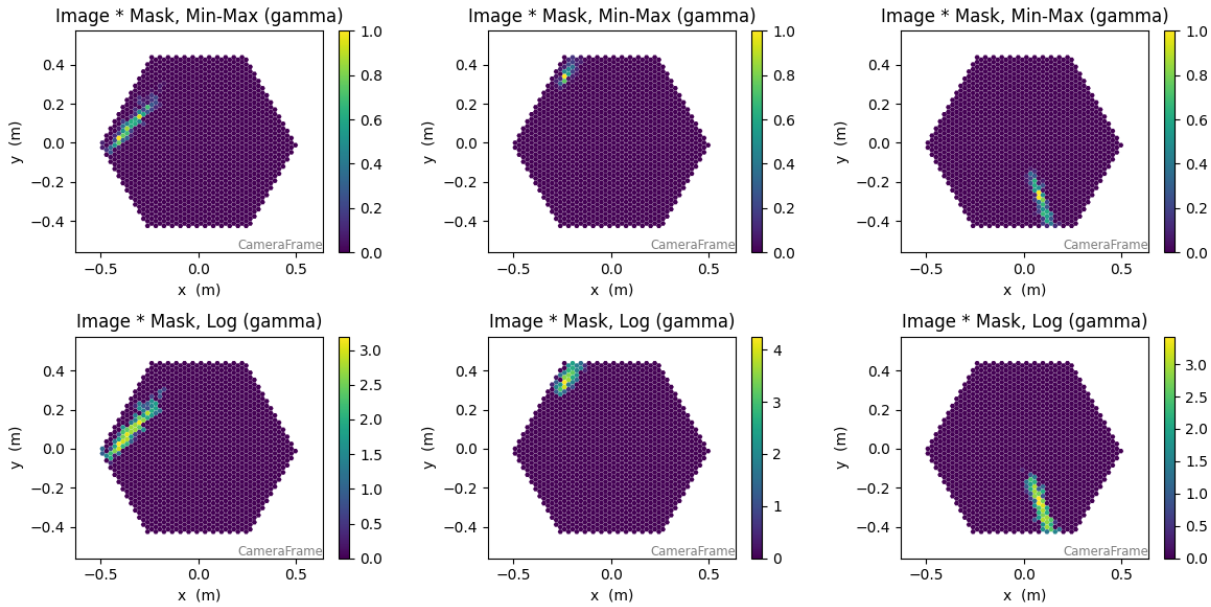


Figure 7.18: Comparison of min-max and logarithmic normalization for gamma event images. The top row shows the image after min-max normalization, while the bottom row shows the same image after logarithmic normalization. Both normalizations are applied after masking, so only important pixels are considered.

They are trained on the extended masked images, and the reconstruction error is calculated only on the important pixels, as before. The names of the models in this section are suffixed with "log" to differentiate them from those used previously.

The results, presented in Table 7.9, show a significant improvement in performance for the Flat

Table 7.9: Model performance metrics using images with extended masks with logarithmic normalization.

Model	AUC	Accuracy	F1-Score	Precision	Recall
Flat AE (extended mask image)	0.6416	0.61	0.61	0.61	0.61
Graph AE (extended mask image)	0.6362	0.60	0.60	0.60	0.60
Hexa AE (extended mask image)	0.6417	0.60	0.60	0.60	0.60
Flat AE (log)	0.7273	0.67	0.67	0.67	0.67
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Hexa AE (log)	0.5713	0.55	0.55	0.55	0.55

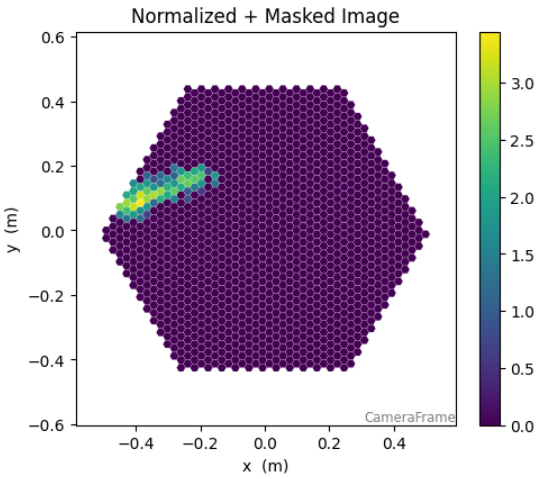
AE and Graph AE models using logarithmic normalization compared to those using min-max normalization. However, surprisingly, the Hexa AE model shows a decrease in performance with this normalization. This result is unexpected, as until now, Hexa AE behaved very similarly to Graph AE. Further analysis is needed to understand why Hexa AE does not benefit from logarithmic normalization like the other models. The improvement for the other two models is indeed very notable, with AUCs increasing from 0.64 to over 0.72.

Table 7.10: Reconstruction error statistics on logarithmically normalized extended masked images for protons and gammas. P stands for protons and G for gammas.

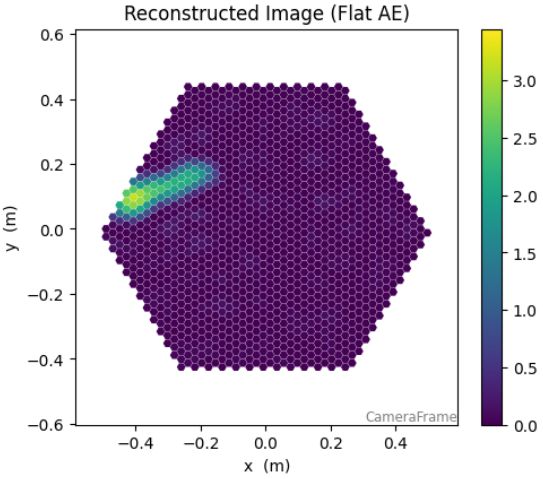
Model	Median P	Median G	Mean P	Mean G	Std P	Std G
Flat AE (log)	0.3647	0.4569	0.3842	0.4933	0.1141	0.1719
Graph AE (log)	0.3048	0.4031	0.3268	0.4406	0.1101	0.1744
Hexa AE (log)	0.3733	0.3975	0.3897	0.4247	0.1099	0.1437

The models use the same input data, so it is possible to compare the median reconstruction errors. Table 7.10 shows that Hexa AE is the model with the lowest median reconstruction error for gamma events, which is consistent with its poor performance in anomaly detection. However, all models have relatively close median reconstruction errors, but only Hexa AE does not benefit from logarithmic normalization. It is likely that the structure of the Hexa AE architecture does not benefit from this normalization. Therefore, modifications to the architecture may be necessary to fully exploit this normalization. This does not, however, explain why such a difference in anomaly detection performance appears.

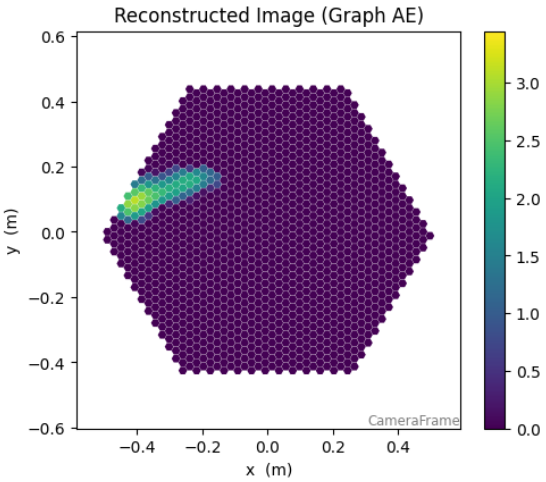
To try to understand this performance difference, a visual analysis of the reconstructed images was conducted. Figure 7.19 presents an example of an image reconstructed by the different models using logarithmic normalization. This time, all three models produce reconstructions of similar quality. The shower is well reconstructed, and the the high-energy pixels are much better represented than in previous experiments. This confirms that logarithmic normalization allows for better image reconstruction. However, a slight difference can be observed for the highest energy pixel. The Hexa AE model is the one that restores the highest value at this location, while the Flat AE and Graph AE models tend to underestimate this value. In terms of value, on the normalized image, the maximum value is 3.45, Hexa AE restores 3.53, while Flat AE and Graph AE restore values of 3.15 and 3.09, respectively. This phenomenon is not isolated and is found in all the images analyzed. Hexa AE tends to better restore high values than the other models with this normalization. It is also noticeable that the shower reconstructed by Hexa AE



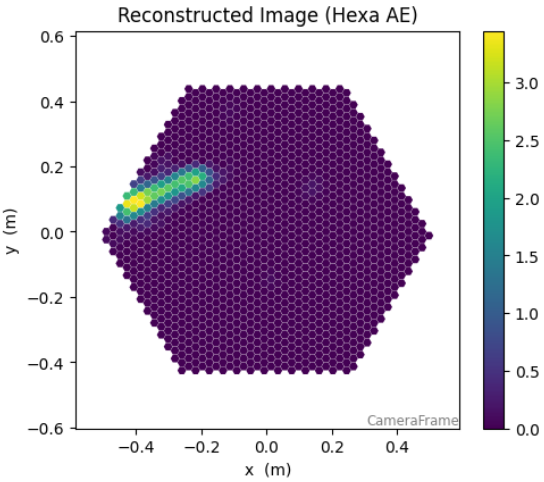
(a) Normalized and Masked Input (Image)



(b) Flat AE



(c) Graph AE



(d) Hexa AE

Figure 7.19: Comparison of reconstructed images obtained with different autoencoder architectures using a masked image of a proton event as input and logarithmic normalization. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input images shown in subfigure (a). Each subfigure shows the reconstruction from a different model: (b) Flat AE, (c) Graph AE, and (d) Hexa AE.

is narrower than those of the other models, a phenomenon also observed in several images.

These differences are likely the cause of Hexa AE's lower performance in anomaly detection. Indeed, high intensities and a narrower shower are typical characteristics of gamma events compared to protons. If the Hexa AE model better restores these characteristics, the difference in reconstruction error between proton and gamma events will be reduced, making the anomaly detection task more difficult. Therefore, the model needs to be modified if we want to make it more suitable for this normalization. It remains interesting to note that if the task to be performed were different, for example, reconstructing high-quality images, Hexa AE would be an excellent choice of model with this normalization.

This experiment shows that the normalization of input data has a significant impact on model

performance in anomaly detection. Logarithmic normalization significantly improves the performance of Flat AE and Graph AE models, while it requires adjustments for the Hexa AE model to fully benefit from it.

7.8 Modification of Hexa AE architecture

This section follows the previous experiment on logarithmic normalization, where the Hexa AE model did not benefit from this normalization unlike the other models. The goal is to modify the Hexa AE architecture to better exploit this normalization and improve its performance in anomaly detection. The idea is to try to reduce its reconstruction capacity on high values, which seems to be the main difference with the other models. To achieve this, several modifications to the architecture were tested. For reference, the original Hexa AE architecture is shown in Figure 7.20.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
HexAE                                    [1, 1, 36, 49]            --
├─Sequential: 1-1                        [1, 8, 18, 25]            --
│    └─Conv2d: 2-1                       [1, 8, 18, 25]            160
│    └─ReLU: 2-2                         [1, 8, 18, 25]            --
├─Sequential: 1-2                        [1, 4, 9, 13]             --
│    └─Conv2d: 2-3                       [1, 4, 9, 13]             228
│    └─ReLU: 2-4                         [1, 4, 9, 13]             --
├─Flatten: 1-3                           [1, 468]                  --
├─Linear: 1-4                            [1, 64]                   30,016
├─Linear: 1-5                            [1, 468]                  30,420
├─Unflatten: 1-6                         [1, 4, 9, 13]             --
├─Sequential: 1-7                        [1, 8, 18, 25]            --
│    └─Conv2d: 2-5                       [1, 8, 18, 25]            232
│    └─ReLU: 2-6                         [1, 8, 18, 25]            --
├─Conv2d: 1-8                            [1, 1, 36, 49]            153
==========================================================================================
Total params: 61,209
Trainable params: 61,209
Non-trainable params: 0
Total mult-adds (M): 0.06
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.24
Estimated Total Size (MB): 0.33
==========================================================================================
```

Figure 7.20: Summary of the autoencoder architecture with hexagonal convolutions.

The following modifications were tested:

1. Reduction of the number of filters in the convolutional layers
2. Reducing the size of the latent space
3. Adding batch normalization layers after each convolutional layer

The modification mainly to be tested is the reduction of the number of filters, as this should reduce the model's reconstruction capacity, particularly for high values. The base architecture has eight filters in the first convolutional layer, then four in the second. Reductions of these

values to four and two, then to two and one, were tested. Reducing the latent space size to 32, as in one of the previous experiments, was also tested. Finally, batch normalization layers were added after each convolutional layer to help regularize the model and potentially improve its performance.

A batch normalization layer normalizes the outputs of a previous layer by subtracting the mean and dividing by the standard deviation of the outputs over a mini-batch of data [49]. Once normalized, the outputs are scaled and shifted using two parameters learned during training. This helps stabilize and accelerate model training. Additionally, batch normalization acts as a form of regularization, helping to reduce overfitting.

The different combinations of these modifications were therefore tested to see their impact on the model's performance. The models are trained on the images with extended masks with logarithmic normalization and the reconstruction error is calculated only on the important pixels, as in the previous experiment. The names of the models in this section are suffixed by a description of the modified architecture to differentiate them from those used previously. The naming convention is as follows:

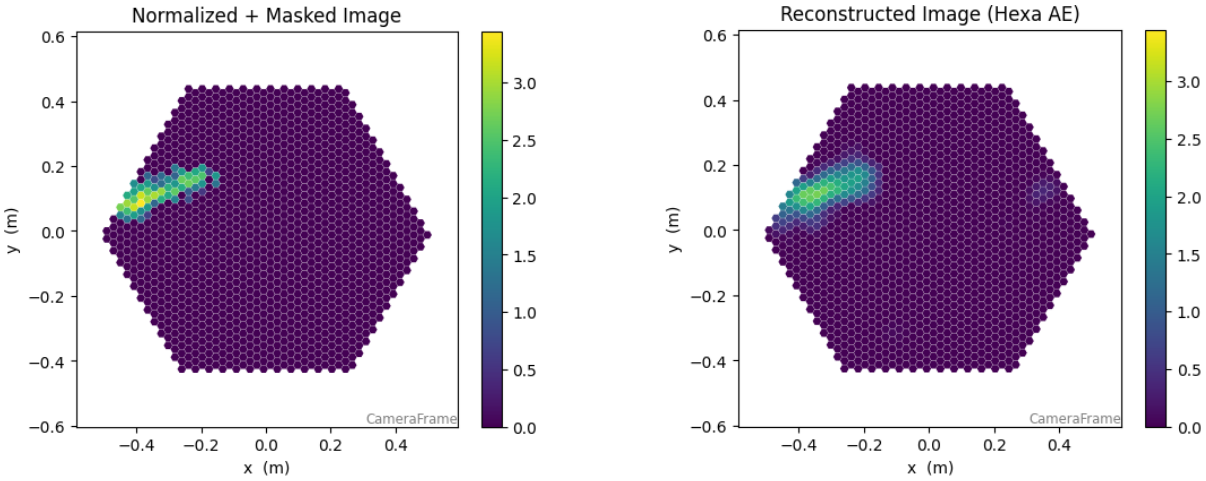
- "X_Y": X is the number of filters in the first convolutional layer, Y in the second.
- "Z": Z is the size of the latent space.
- "BN": indicates the addition of batch normalization layers.

Table 7.11: Model performance metrics for modified Hexa AE architectures with logarithmic normalization.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Hexa AE (extended mask image)	0.6417	0.60	0.60	0.60	0.60
Hexa AE (log)	0.5713	0.55	0.55	0.55	0.55
Hexa AE (4_2 64)	0.6639	0.62	0.62	0.62	0.62
Hexa AE (4_2 32)	0.6723	0.63	0.63	0.63	0.63
Hexa AE (4_2 64 BN)	0.6601	0.62	0.62	0.62	0.62
Hexa AE (4_2 32 BN)	0.6947	0.65	0.65	0.65	0.65
Hexa AE (2_1 64)	0.7178	0.67	0.67	0.67	0.67
Hexa AE (2_1 32)	0.7426	0.69	0.69	0.69	0.69
Hexa AE (2_1 64 BN)	0.7510	0.69	0.69	0.69	0.69
Hexa AE (2_1 32 BN)	0.7421	0.68	0.68	0.68	0.68

The results, presented in Table 7.11, show that reducing the number of filters in the convolutional layers significantly improves the performance of the Hexa AE model. Batch normalization also seems to help, particularly when combined with a reduced number of filters. The impact of reducing the latent space size is less clear, but it seems that this can also contribute to improving performance in some cases. The best performance is achieved with the architecture having two and one filters in the convolutional layers, a latent space size of 64, and with batch normalization, reaching an AUC of 0.7510. The improvement is therefore notable compared to the original Hexa AE architecture with logarithmic normalization, which had an AUC of only 0.5713.

Figure 7.21 presents an example of an image reconstructed by the modified Hexa AE model



(a) Normalized and Masked Input (Image)

(b) Hexa AE (2_1 64 BN)

Figure 7.21: Example of reconstructed image obtained with the modified Hexa AE architecture (2_1 64 BN) using a masked image of a proton event as input and logarithmic normalization. The scale is the same for all images for better comparison. The image is reconstructed from the normalized and extended masked input image shown in subfigure (a). Subfigure (b) shows the reconstruction from the modified Hexa AE model.

(2_1 64 BN) using logarithmic normalization. We can observe that the reconstruction seems to be of good quality, although inferior to that obtained with the original Hexa AE architecture (visible in Figure 7.19). In particular, high values are less well restored, and the shower profile appears slightly wider. The model is less accurate at reconstructing, but it is much better for anomaly detection, which is the desired objective. This visual analysis therefore confirms the quantitative results obtained previously.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AE                                       [1, 1, 36, 49]            --
├─Conv2d: 1-1                            [1, 2, 18, 25]            40
├─BatchNorm2d: 1-2                       [1, 2, 18, 25]            4
├─Conv2d: 1-3                            [1, 1, 9, 13]             15
├─BatchNorm2d: 1-4                       [1, 1, 9, 13]             2
├─Flatten: 1-5                           [1, 117]                  --
├─Linear: 1-6                            [1, 64]                   7,552
├─Linear: 1-7                            [1, 117]                  7,605
├─Unflatten: 1-8                         [1, 1, 9, 13]             --
├─Conv2d: 1-9                            [1, 2, 18, 25]            16
├─BatchNorm2d: 1-10                      [1, 2, 18, 25]            4
├─Conv2d: 1-11                           [1, 1, 36, 49]            39
==========================================================================================
Total params: 15,277
Trainable params: 15,277
Non-trainable params: 0
Total mult-adds (M): 0.02
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.05
Params size (MB): 0.06
Estimated Total Size (MB): 0.11
==========================================================================================
```

Figure 7.22: Summary of the autoencoder architecture with hexagonal convolutions (modified).

For future experiments, this modified Hexa AE architecture with logarithmic normalization will be used. To avoid confusion with the original architecture, it will therefore be named "Modified Hexa AE" in the rest of the document. Figure 7.22 summarizes this modified architecture. The number of parameters has been significantly reduced, from 61,209 to 15,277, which partly explains the reduction in the model's reconstruction capacity.

7.9 Event selection based on leakage

In the images, it sometimes happens that the shower is partially outside the field of view of the telescopes. As a result, part of the shower is not captured. In the data, an additional piece of information quantifies this point: the leakage, which indicates the fraction of shower pixels that are on the edge of the camera. This parameter is calculated with respect to the base mask (see Section 7.3) and ranges from zero to one. Events with high leakage are therefore those with a significant portion of the shower outside the field of view of the telescopes. This results in event images that are difficult to analyze correctly. Consequently, it is also possible that ignoring these events improves anomaly detection, as the images will be more complete and easier to analyze. This is similar to the Hillas-based quality cut presented in Section 7.2.

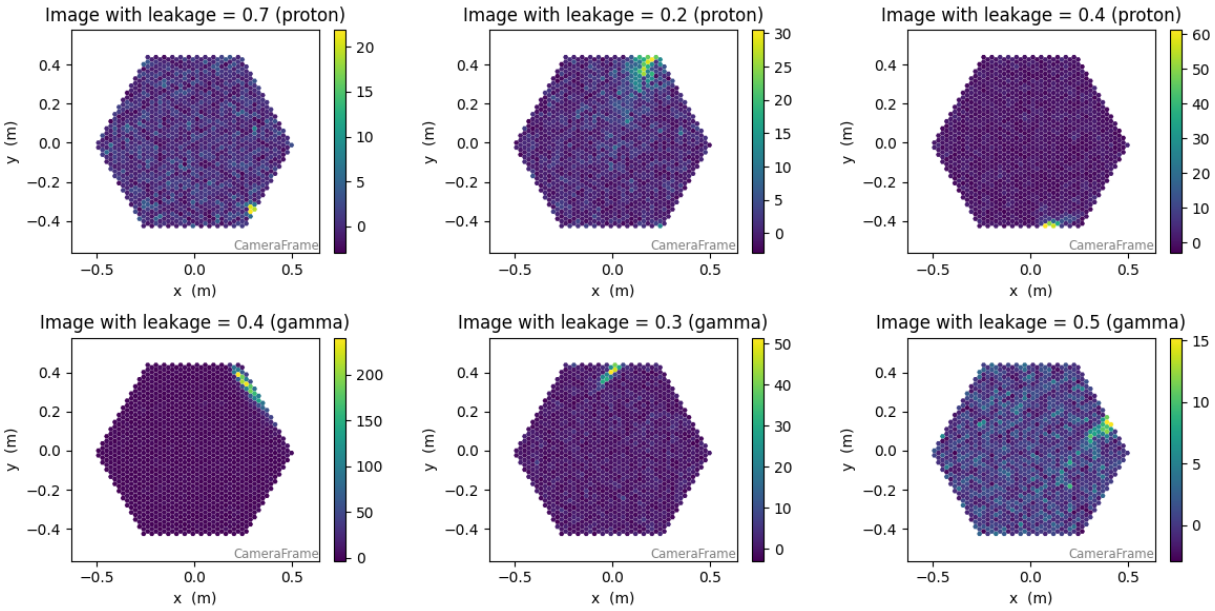


Figure 7.23: Example of event images with high leakage. The top row shows proton event images with leakage values of 0.7, 0.2, and 0.4 from left to right. The bottom row shows gamma event images with leakage values of 0.4, 0.3, and 0.5 from left to right.

Figure 7.23 shows examples of event images with leakage values greater than or equal to 0.2. We can observe that in these images, a significant portion of the shower is cut off, as expected. These images are therefore more difficult to analyze correctly, which could harm the performance of anomaly detection models.

To test this hypothesis, an experiment was conducted where events with leakage values greater than or equal to the threshold of 0.2 were ignored. This threshold was chosen after a visual analysis of several images with different leakage values and a discussion with an expert in the

field. This filtering was applied to the entire dataset before training the models, thus reducing the dataset size from 290,303 to 220,709 events; the reduction is relatively small. The models used are the same as in previous experiments: Flat AE, Graph AE, and Modified Hexa AE. They are trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The names of the models in this section are suffixed by "leakage-based" to differentiate them from those used previously.

Table 7.12: Model performance metrics using images with extended masks with logarithmic normalization and leakage filtering.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (log)	0.7273	0.67	0.67	0.67	0.67
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Modified Hexa AE (log)	0.7510	0.69	0.69	0.69	0.69
Flat AE (leakage-based)	0.7252	0.67	0.67	0.67	0.67
Graph AE (leakage-based)	0.7240	0.67	0.67	0.67	0.67
Modified Hexa AE (leakage-based)	0.7720	0.71	0.71	0.71	0.71

The results, presented in Table 7.12, show that removing events with high leakage does not have a significant impact on the performance of the Flat AE and Graph AE models. However, the Modified Hexa AE model shows a slight improvement, reaching an AUC of 0.7720 compared to 0.7510 previously. This improvement is modest, but it suggests that filtering out events with high leakage can be beneficial for certain models. However, overall, the impact of this filtering remains limited.

No visual analysis of the reconstructed images is presented here, as this experiment only removes a small portion of events from the dataset. Therefore, the reconstructed images are similar to those from previous experiments, and visual analysis would not provide significant additional information.

7.10 Autoencoder architecture variations

This section presents various experiments aimed at modifying the different architectures of the autoencoders used. The goal is to explore different configurations to improve anomaly detection performance. Each subsection describes a specific modification, the motivations behind this modification, as well as the results obtained.

7.10.1 Hexa AE: Change the interpolation method

Currently, the Hexa AE decoder uses the "nearest" method for interpolation during image upsampling (see Chapter 6). This interpolation method is the simplest and fastest, but it is not the most accurate. Indeed, it simply assigns to each pixel of the enlarged image the value of the nearest pixel in the original image. This can lead to visual artifacts and a loss of quality. Two other interpolation methods, "bilinear" and "bicubic," were therefore tested:

- **Bilinear**: This method performs linear interpolation in two dimensions, meaning it computes a weighted average of the surrounding pixels (4 pixels for a 2D image) to calculate

the new values during upsampling [44]. It provides smoother images than the "nearest" method but is also more computationally expensive.

- **Bicubic**: This method uses cubic interpolation, meaning it takes into account a larger number of surrounding pixels (16 pixels for a 2D image) to calculate the new values. The closest pixels have a higher weight in the calculation [44]. It generally produces even smoother and higher-quality images than the bilinear method but is also the most computationally expensive.

These methods are illustrated in Figure 7.24.

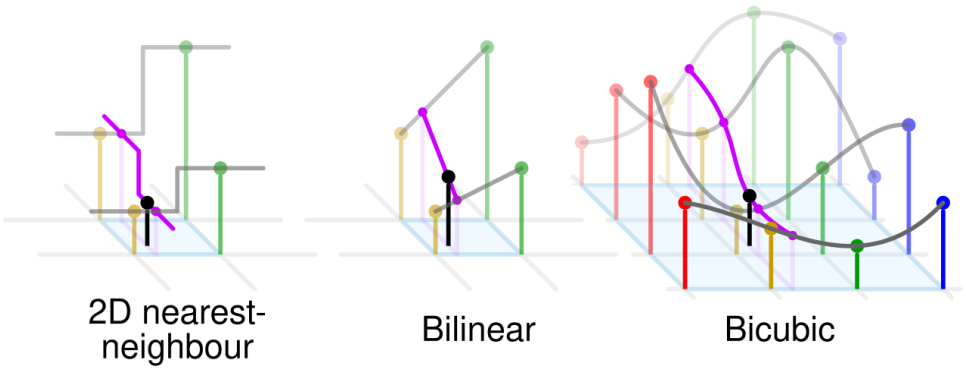


Figure 7.24: Comparison of interpolation methods used during upsampling. The left image shows the result of nearest neighbor interpolation, the middle image shows bilinear interpolation, and the right image shows bicubic interpolation. The black dots correspond to the point being interpolated, and the red, yellow, green and blue dots correspond to the neighbouring samples. Their heights above the ground correspond to their values, source: [50].

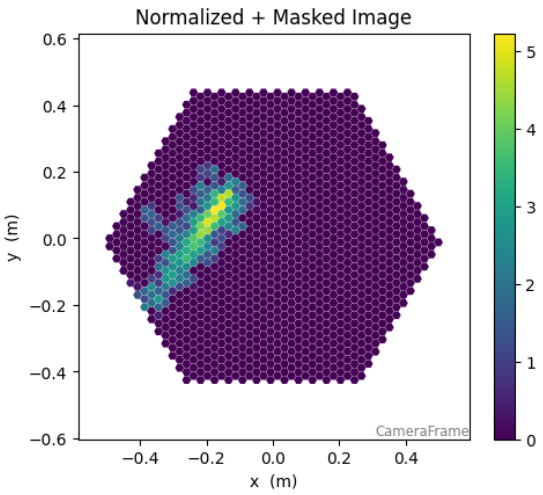
The Modified Hexa AE model is used as a base for this experiment. The modification only concerns the interpolation method used in the decoder. The models are trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The names of the models in this section are suffixed by the name of the interpolation method used to differentiate them from previously used models.

Table 7.13: Model performance metrics for Modified Hexa AE with different interpolation methods.

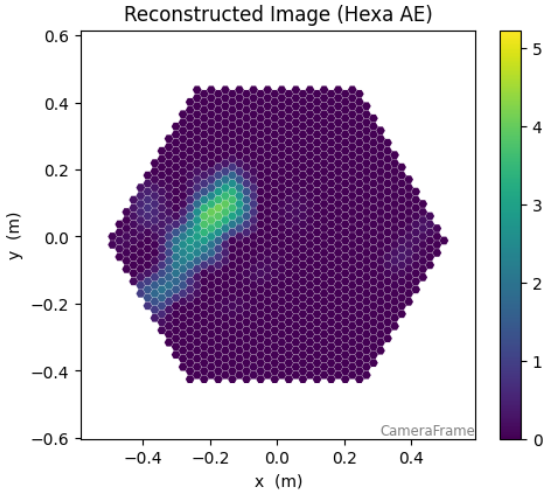
Model	AUC	Accuracy	F1-Score	Precision	Recall
Modified Hexa AE (log)	0.7510	0.69	0.69	0.69	0.69
Modified Hexa AE (bilinear)	0.7326	0.68	0.68	0.68	0.68
Modified Hexa AE (bicubic)	0.7142	0.66	0.66	0.66	0.66

The results, presented in Table 7.13, show that using more advanced interpolation methods does not improve the performance of the Modified Hexa AE model. In fact, the "nearest" method gives the best results in terms of AUC, while the "bilinear" and "bicubic" methods lead to a progressive decrease in performance. A visual analysis of the reconstructed images can help understand why these methods are not beneficial in this context.

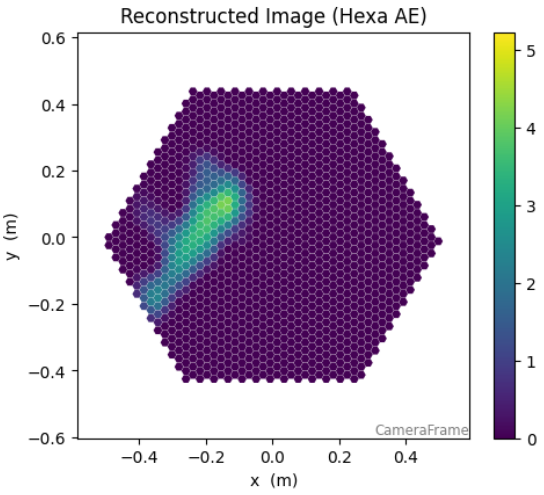
Figure 7.25 presents an example of an image reconstructed by the Modified Hexa AE model using different interpolation methods. We can directly observe that the "bicubic" method produces



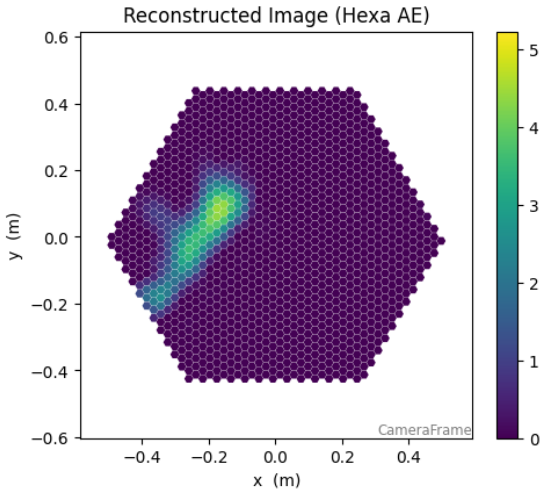
(a) Normalized and Masked Input (Image)



(b) Modified Hexa AE (nearest)



(c) Modified Hexa AE (bilinear)



(d) Modified Hexa AE (bicubic)

Figure 7.25: Comparison of reconstructed images obtained with Modified Hexa AE using a masked image of a proton event as input and logarithmic normalization. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input images shown in subfigure (a). Each subfigure shows the reconstruction from a different model: (b) Modified Hexa AE (nearest), (c) Modified Hexa AE (bilinear), and (d) Modified Hexa AE (bicubic).

the visually best reconstruction, with better representation of high intensities and a finer shower shape. However, despite this improved visual quality, the "bilinear" and "bicubic" methods do not lead to better anomaly detection. The problem is likely the same as mentioned in Section 7.7, namely that the models reconstruct too well and fail to sufficiently differentiate between proton and gamma events. Thus, even though the visual quality of the images is better, this does not translate into improved performance in anomaly detection.

An analysis of the median reconstruction error of the models (not shown here) indicates that the models with the "bilinear" and "bicubic" methods have significantly lower reconstruction errors than the model with the "nearest" method. This confirms that these models reconstruct

images better, but at the expense of the ability to detect anomalies. Therefore, in the context of anomaly detection, it is preferable to use the "nearest" interpolation method for the Modified Hexa AE model. However, it remains important to note that these more advanced interpolation methods can be beneficial in other contexts.

7.10.2 Graph AE: Prune the graph

Currently, the Graph AE model does not perform actual compression of images, since the number of nodes in the graph remains the same across all layers. However, a particular feature of graph-based models is that the graph structure can be adapted, with a variable number of nodes and edges. This flexibility allows for dynamically adapting the graph structure based on the input images.

In this experiment, the graph will be directly modified, particularly the edges. All edges connecting nodes where at least one node does not belong to the extended mask are removed. The resulting graph will thus correspond to a subgraph of the initial graph, limited to the relevant edges according to the mask. This approach leads to the removal of the vast majority of edges in the graph, as most pixels do not belong to the mask. On average, for proton events, only 75 pixels out of 1,296 belong to the mask, while for gamma events, this number is approximately 50 pixels, based on a sample of 50,000 events per class. Therefore, the number of edges in the graph is significantly reduced. The goal is to retain only the connections between important pixels. Consequently, convolutions are thus performed only between these pixels.

This approach is conceptually similar to removing nodes that do not belong to the mask. However, it allows for easier adaptation of the existing code and testing of this idea without modifying the handling of node numbers, by acting solely on the edges.

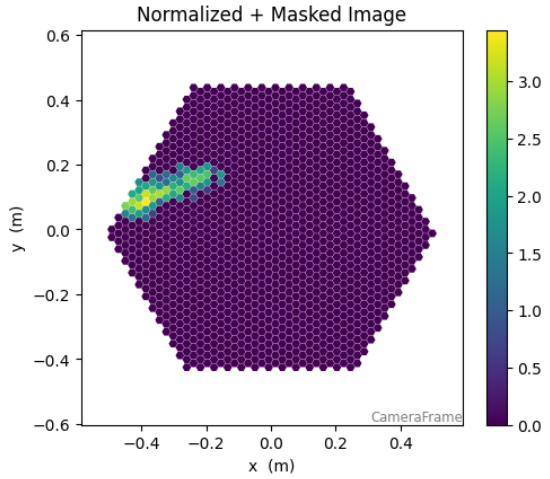
The Graph AE model is used as a base for this experiment. The modification concerns the removal of edges not belonging to the extended mask. The model is trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The name of the model in this section is suffixed by "pruned" to differentiate it from the model used previously.

Table 7.14: Model performance metrics for Graph AE with pruned graph structure.

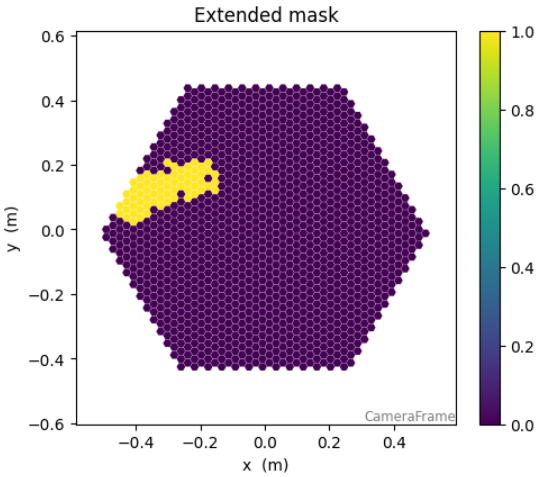
Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Graph AE (pruned)	0.6870	0.64	0.64	0.64	0.64

Results, presented in Table 7.14, show that removing edges from the graph does not improve the performance of the Graph AE model. In fact, this modification leads to a significant decrease in performance in terms of AUC. A visual analysis of the reconstructed images can help understand why this modification is not beneficial in this context.

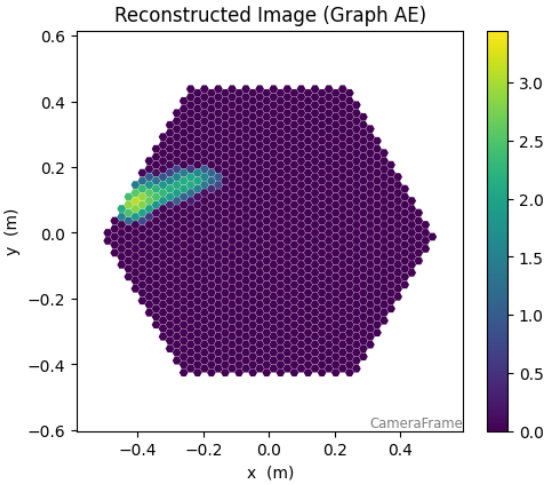
Figure 7.26 presents an example of an image reconstructed by the Graph AE model using the original graph structure and the pruned graph structure. We observe that both reconstructions are of similar quality. The main difference is that the reconstruction using the pruned graph perfectly respects the mask shape, which is expected since only connections between pixels in the



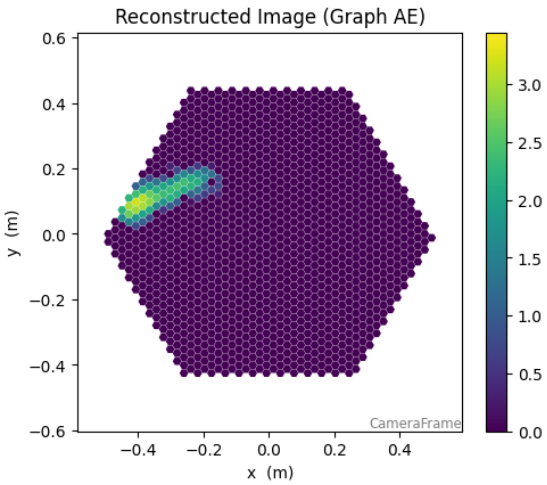
(a) Normalized and Masked Input (Image)



(b) Extended Mask



(c) Graph AE



(d) Graph AE (pruned)

Figure 7.26: Comparison of reconstructed images obtained with Graph AE using original and pruned graph structures with a masked image of a proton event as input and logarithmic normalization. The scale is the same for all images for better comparison. The images are reconstructed from the normalized and extended masked input images shown in subfigure (a). Subfigure (b) shows the extended mask used. Each subfigure shows the reconstruction from a different model: (c) Graph AE with original graph, and (d) Graph AE with pruned graph.

mask are retained. However, the overall quality of the reconstruction remains similar between the two models, and this model fails to better differentiate between proton and gamma events. This modification does show, however, that it is possible to reduce the number of edges in the graph without harming reconstruction quality.

An analysis of the median reconstruction errors (not shown here) indicates that the model with the pruned graph has a slightly lower reconstruction error than the model with the original graph. The conclusion is therefore similar to that of Section 7.10.1, the model reconstructs images better, but at the expense of the ability to detect anomalies. In the context of anomaly detection, it is therefore preferable to use the original graph structure for the Graph AE model. However, this experiment shows that it is possible to significantly reduce the number of edges

in the graph without harming reconstruction quality, which can be useful in other contexts.

7.11 Experiments without significant results

This section presents a list of experiments that were attempted but did not lead to significant progress in anomaly detection. Although these approaches did not yield significant improvements, they help identify avenues that are less promising for future research and provide insights for further exploration. The results here will therefore be much briefer than in the other sections. It primarily involves explaining the modifications made, the motivations behind them, and a summary of the results when available.

7.11.1 Graph AE: Testing alternative graph convolution layers

In the Graph AE model, the GCN convolutional layer from the `PyTorch Geometric` library [29] is used. This layer is a popular and efficient implementation, but there are others that could potentially offer better performance. Other types of convolution layers were therefore tested. The following layers were tested:

- **ChebConv [51]:** A graph convolution based on Chebyshev polynomials, which allows capturing information from neighbors up to order K. This allows for a large receptive field without increasing the number of model parameters [52].
- **GATConv [53]:** A graph convolution based on the attention mechanism, which allows weighting the importance of neighbors when aggregating information. This enables the model to focus on the most relevant neighbors for each node [54].
- **GINConv [55]:** A graph convolution that uses an MLP after aggregating information from neighbors. This layer is designed to be expressive and capable of capturing complex structures in graphs [56].

For this experiment, the GCN layers in the Graph AE encoder were replaced with the layers mentioned above. The models are trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The names of the models in this section are suffixed by the name of the convolution layer used to differentiate them from previously used models.

Table 7.15: Model performance metrics for Graph AE with different graph convolution layers.

Model	AUC	Accuracy	F1-Score	Precision	Recall
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Graph AE (ChebConv)	0.5152	0.51	0.51	0.51	0.51
Graph AE (GATConv)	0.4859	0.49	0.49	0.49	0.49
Graph AE (GINConv)	0.7079	0.65	0.65	0.65	0.65

Results, presented in Table 7.15, show that using these different convolution layers does not improve the performance of the Graph AE model. The models with ChebConv and GATConv show performance similar to a random model, while the model with GINConv layers is slightly

inferior to the original model with GCN layers. To understand these results, it is best to look at the median reconstruction errors of the models.

Table 7.16: Reconstruction errors statistics for Graph AE with different graph convolution layers. P stands for protons and G for gammas.

Model	Median P	Median G	Mean P	Mean G	Std P	Std G
Graph AE (log)	0.3048	0.4031	0.3268	0.4406	0.1101	0.1744
Graph AE (ChebConv)	0.0000	0.0000	0.0000	0.0000	0.0002	0.0001
Graph AE (GATConv)	0.0733	0.0715	0.0842	0.0899	0.0480	0.0734
Graph AE (GINConv)	0.2720	0.3536	0.2938	0.3536	0.1099	0.1873

The results presented in Table 7.16 show the median reconstruction errors of the different models. We can see that the models with ChebConv and GATConv layers have extremely low reconstruction errors, meaning they almost perfectly reconstruct all images, whether they are protons or gammas. Therefore, they fail to differentiate between the two types of events and are similar to a random model. Since Graph AE does not perform true compression of images, GCN layers could not learn to perfectly reconstruct the images, as the same weight is applied to a node and all its neighbors. This prevents the model from memorizing the images and forces it to learn more general representations. In contrast, ChebConv and GATConv do not have this constraint, allowing them to achieve near-perfect reconstruction. The model with GINConv behaves more similarly to that with GCN layers but fails to reach the same performance for anomaly detection.

This experiment shows that the choice of graph convolution layer is crucial for the performance of the Graph AE model. It is important to remember that the behavior observed here is specific to having only one feature per node. In other contexts, with more features per node, it is possible to achieve true compression and thus force the model to learn more general representations, even with ChebConv or GATConv. However, in the current context, it is preferable to use the GCN layer for the Graph AE model. The GINConv layer could be an alternative choice, but it does not bring improvement compared to GCN layer and is more computationally expensive.

7.11.2 Graph AE: Node reduction strategies

In the subsection 7.10.2, dges were removed from the graph to retain only those connecting important pixels. An alternative approach would be to directly remove unimportant nodes, which should yield similar results. However, it is also possible to use more sophisticated node reduction methods that retain the most relevant nodes based on criteria learned by the model. Two of these methods were therefore tested:

- **TopKPooling:** This method involves selecting the most important nodes by assigning them an importance score that is learned during training. This score depends on the node's features. Then, the K nodes with the highest scores are retained, while the others are removed [57].
- **SAGPooling:** This method is very similar to the previous one but uses a different approach to evaluate node importance. It employs self-attention to weight the nodes before

selecting them. The score thus depends not only on the node's features but also on its relationships with other nodes in the graph [58].

For this experiment, a node reduction layer was added between the two convolutional layers in the Graph AE encoder. The two methods mentioned above were tested, retaining 20% of the nodes during reduction. The model is trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The names of the models in this section are suffixed by the name of the reduction method used to differentiate them from previously used models.

Table 7.17: Model performance metrics for Graph AE with different node reduction methods.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Graph AE (TopKPooling)	0.7200	0.67	0.67	0.67	0.67
Graph AE (SAGPooling)	0.7362	0.68	0.68	0.68	0.68

Results, presented in Table 7.17, show that using these methods does not improve the performance of the Graph AE model. The results are very similar to those obtained with the original graph structure. The median reconstruction errors of the models are also very similar, as is the visual quality of the reconstructed images. These elements are not presented here, as they do not provide any additional relevant information.

Reducing the number of nodes in the graph therefore does not bring improvement in this context. This can be useful if it is necessary to store more compact versions of the images, but in this case, the TopKPooling and SAGPooling methods do not seem to be the most suitable. Directly using the mask to remove unimportant nodes, similarly to what was done with edges in subsection 7.10.2, would be a simpler and more effective approach.

Another problem encountered with these methods is that they have difficulty converging correctly during training. In some runs, the model failed to retain relevant nodes, and the final result was that the reconstructed features were all zero. This indicates that these methods can be unstable in this context, complicating their use. Therefore, it is not recommended to use these methods for this case without further adjustments.

7.11.3 Pixel value clipping

Another idea tested is to limit the maximum pixel values in the input images. The idea is that very high intensities are rare and can disrupt the training of the models. It is also very difficult to reconstruct them correctly, which can harm the anomaly detection. By limiting the maximum values, it is possible to reduce the impact of these extreme values and improve the reconstruction quality for the majority of pixels.

For this experiment, two different clipping thresholds were applied: 1,000 and 500. All values above these thresholds are thus set to the threshold value. The vast majority of images are not affected by this clipping. In a sample of 60,000 events, only 450 protons and 400 gammas have at least one pixel above 1,000. The models used are the same as in the previous experiments:

Flat AE, Graph AE, and Modified Hexa AE. They are trained on the images with extended masks with logarithmic normalization, as in the previous experiment. The names of the models in this section are suffixed by the clipping threshold used to differentiate them from previously used models.

Table 7.18: Model performance metrics using images with extended masks with logarithmic normalization and various clipping thresholds.

Model	**AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**
Flat AE (log)	0.7273	0.67	0.67	0.67	0.67
Graph AE (log)	0.7387	0.68	0.68	0.68	0.68
Modified Hexa AE (log)	0.7510	0.69	0.69	0.69	0.69
Flat AE (clip 1,000)	0.7283	0.67	0.67	0.67	0.67
Graph AE (clip 1,000)	0.7424	0.68	0.68	0.68	0.68
Modified Hexa AE (clip 1,000)	0.7517	0.69	0.69	0.69	0.69
Flat AE (clip 500)	0.7300	0.67	0.67	0.67	0.67
Graph AE (clip 500)	0.7212	0.67	0.67	0.67	0.67
Modified Hexa AE (clip 500)	0.7260	0.67	0.67	0.67	0.67

The results, presented in Table 7.18, show that clipping the maximum values yields results very similar to those obtained without clipping. There is no significant improvement in the performance in terms of AUC or other metrics, the differences being very minor and due to random variations. The median reconstruction error of the models is also very similar, as is the visual quality of the reconstructed images. These elements are not presented here, as they do not provide significant additional information.

Performing clipping of maximum values therefore does not seem useful. There is already a logarithmic normalization that reduces the impact of very high values. It is therefore likely that clipping does not provide any additional advantage in this context. However, this approach could be useful if logarithmic normalization is not used.

7.11.4 Explored input modalities without quantitative evaluation

This subsection briefly presents some input modalities that were explored, but for which a thorough quantitative evaluation was not performed. These approaches were tested on a reduced dataset, which shows a behavior similar to that of the complete dataset used for the other experiments presented in this chapter. However, the results obtained did not show sufficient potential to justify further exploration or evaluation on a larger scale. Therefore, only the ideas explored are presented here, without providing a detailed quantitative evaluation.

Combining image and peak time information

The first input modality explored is the use of both images and peak time as input data for the models. The idea was that the model could learn more complete representations by combining these two types of information, which could improve anomaly detection.

Concretely, for the Hexa AE architecture, an additional branch was added in the encoder to process the peak time. This branch was exactly the same as that used for the images, but

with different weights. The representations learned by the two branches are then concatenated before being passed to the latent space. The decoder follows the same logic, with two distinct branches to reconstruct the images and peak time from the latent space. It is then possible to independently calculate the reconstruction errors for the images and peak time and combine them to obtain an overall anomaly score, while weighting the relative importance of the two types of errors. However, this approach provided results identical to those obtained using only images as input data. The idea was therefore abandoned for Hexa AE, as it was much more complex to implement without bringing any additional performance.

The experiment was also carried out with the Graph AE architecture, by adding an additional feature to each node of the graph to represent the peak time. Thus, each node had two characteristics: the pixel intensity and the corresponding peak time. The rest of the model architecture remained unchanged. The reconstruction error is calculated in the same way as for Hexa AE, by combining the errors on the images and peak time. However, the finding was the same as for Hexa AE: the results obtained were identical to those obtained using only images as input data. The idea was therefore also abandoned for Graph AE.

The experiment was not tested on the other architectures, as the results obtained with Hexa AE and Graph AE were not promising.

Using waveforms as input data

The second input modality explored is the use of the event waveform (R1) as input data for the models. The idea was that DL1 images do not contain all the information available in the raw data, and that using waveforms could allow the models to learn much more complete representations.

The complete waveform includes 50 samples per pixel, which represents a much larger amount of data than the DL1 images. However, the event only lasts a few samples and the other samples contain only noise. Only 15 samples centered around the peak of the waveform were therefore kept, which allows capturing the relevant information while reducing the amount of data to be processed.

The models tested were Hexa AE and Graph AE. For Hexa AE, the number of input channels was increased to include the 15 samples of the waveform, while for Graph AE, each node of the graph was given 15 features. To remove noise, a mask similar to that used for DL1 images, but adapted to waveforms, was tested. The experiment was conducted with and without this mask.

However, the results obtained were significantly lower than those obtained using DL1 images as input data. The experiment with the mask gave better results than the one without. Without the mask, there was too much noise in the data, preventing the models from learning useful representations. With the mask, the models were able to focus on the relevant parts of the waveforms, but performance remained lower than that obtained with DL1 images. The computation time required to train the models was also much longer, complicating experimentation. Therefore, this approach was abandoned, as it did not provide any improvement over using DL1 images and was more complex to implement.

This method could nevertheless be interesting to explore further, as it would allow taking advantage of all the information available in the raw data. In any case, further adjustments would be necessary, such as optimizing the architecture of the models to better process waveforms and exploring different preprocessing techniques to reduce noise while retaining relevant information. However, these explorations are beyond the scope of this work and were not pursued.

7.11.5 Other experiments

Other experiments have been attempted but did not yield significant improvements. They are not presented in detail here, as they would require presenting a large number of results without providing additional relevant information.

In all the experiments presented in this chapter, the number of layers was not modified. However, numerous variations were tested in the different models. The number of layers currently present in each model is the one that gives the best performance. Generally, decreasing the number of layers degrades performance, as the model lacks sufficient capacity to learn relevant representations. Increasing the number of layers beyond three also tends to degrade performance, as the model becomes too complex and struggles to converge during training. Additionally, a higher number of layers significantly increases training time and memory requirements, complicating experimentation.

Another experiment attempted was to use peak time instead of images as input data for the models, similar to the experiment in Section 7.4. This was done on all the different models presented in this chapter. However, in all cases, using peak time did not improve performance over images. The results are generally significantly lower, confirming that images contain more relevant information for anomaly detection in this context. However, peak time remains very useful for calculating the extended mask, as explained in Section 7.5.

7.12 Summary of Results

This section presents a summary of the results obtained for the various experiments conducted in this chapter. The goal is to provide an overview of the performance of the different models and configurations tested, to facilitate comparison and analysis of the results.

A summary table of the results obtained is presented in Table 7.19. Each row corresponds to a specific experimental configuration with the following details:

- **Base model:** The base model used (Flat AE, Square AE, Graph AE, Hexa AE, Modified Hexa AE) explained in Chapter 6 and Section 7.8.
- **Normalization:** The normalization method applied to the input images (Min-Max, Log). Min-Max normalization is explained in Chapter 5, and logarithmic normalization in Section 7.7.
- **Quality cut:** The type of quality cut applied to the input images (None, Hillas-based, Leakage-based). The Hillas-based quality cut is explained in Section 7.2 and the Leakage-based quality cut in Section 7.9.

Table 7.19: Summary of experimental results for anomaly detection using autoencoders.

Base model	Normalization	Quality cut	Mask	Others	Section	AUC
Flat AE	Min-Max	None	None	None	7.1	0.4133
Square AE	Min-Max	None	None	Shifting	7.1	0.4168
Square AE	Min-Max	None	None	Bilinear	7.1	0.4182
Graph AE	Min-Max	None	None	None	7.1	0.4200
Hexa AE	Min-Max	None	None	None	7.1	0.4204
Flat AE	Min-Max	Hillas-based	None	None	7.2	0.4542
Square AE	Min-Max	Hillas-based	None	Shifting	7.2	0.4554
Square AE	Min-Max	Hillas-based	None	Bilinear	7.2	0.4612
Graph AE	Min-Max	Hillas-based	None	None	7.2	0.4692
Hexa AE	Min-Max	Hillas-based	None	None	7.2	0.4635
Flat AE	Min-Max	Hillas-based	Base	None	7.3	0.6254
Graph AE	Min-Max	Hillas-based	Base	None	7.3	0.5821
Hexa AE	Min-Max	Hillas-based	Base	None	7.3	0.5759
Flat AE	Min-Max	Hillas-based	Extended	None	7.5	0.6416
Graph AE	Min-Max	Hillas-based	Extended	None	7.5	0.6362
Hexa AE	Min-Max	Hillas-based	Extended	None	7.5	0.6417
Graph AE	Min-Max	Hillas-based	Extended	2 LS	7.6	0.6412
Graph AE	Min-Max	Hillas-based	Extended	1 LS	7.6	0.6450
Hexa AE	Min-Max	Hillas-based	Extended	32 LS	7.6	0.6328
Hexa AE	Min-Max	Hillas-based	Extended	16 LS	7.6	0.6362
Flat AE	Log	Hillas-based	Extended	None	7.7	0.7273
Graph AE	Log	Hillas-based	Extended	None	7.7	0.7387
Hexa AE	Log	Hillas-based	Extended	None	7.7	0.5713
Hexa AE	Log	Hillas-based	Extended	4_2 64	7.8	0.6639
Hexa AE	Log	Hillas-based	Extended	4_2 32	7.8	0.6723
Hexa AE	Log	Hillas-based	Extended	4_2 64 BN	7.8	0.6601
Hexa AE	Log	Hillas-based	Extended	4_2 32 BN	7.8	0.6947
Hexa AE	Log	Hillas-based	Extended	2_1 64	7.8	0.7178
Hexa AE	Log	Hillas-based	Extended	2_1 32	7.8	0.7426
Hexa AE	Log	Hillas-based	Extended	2_1 64 BN	7.8	**0.7510**
Hexa AE	Log	Hillas-based	Extended	2_1 32 BN	7.8	0.7421
Flat AE	Log	Leakage-based	Extended	None	7.9	0.7252
Graph AE	Log	Leakage-based	Extended	None	7.9	0.7240
Modified Hexa AE	Log	Leakage-based	Extended	None	7.9	**0.7720**
Modified Hexa AE	Log	Hillas-based	Extended	Bilinear	7.10.1	0.7326
Modified Hexa AE	Log	Hillas-based	Extended	Bicubic	7.10.1	0.7142
Graph AE	Log	Hillas-based	Extended	Pruned	7.10.2	0.6870

- **Mask:** The type of mask applied to the input images (None, Base, Extended). The Base mask is explained in Section 7.3, and the Extended mask in Section 7.5.
- **Others:** Other specific modifications made to the model or data that are not covered by the other columns. This includes the type of mapping used (Shifting, Bilinear) for the Square AE model, the number of dimensions in the latent space for latent space reduction, specific modifications made to the Hexa AE and Graph AE models.
- **Section:** The section of the chapter where the experiment is described in detail.
- **AUC:** The value of the AUC obtained for this experimental configuration, which is the

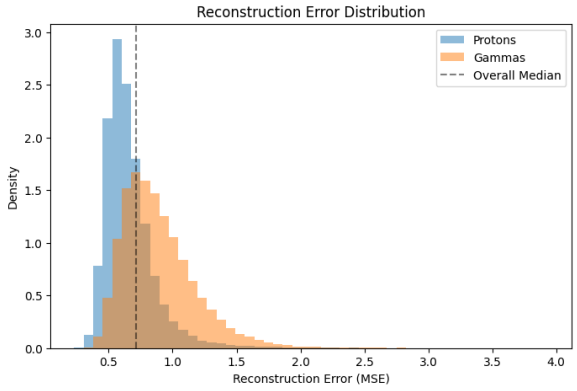
main metric used to evaluate anomaly detection performance.

Generally, this table allows us to see performance trends based on different experimental configurations. The factors that most contributed to performance improvement are:

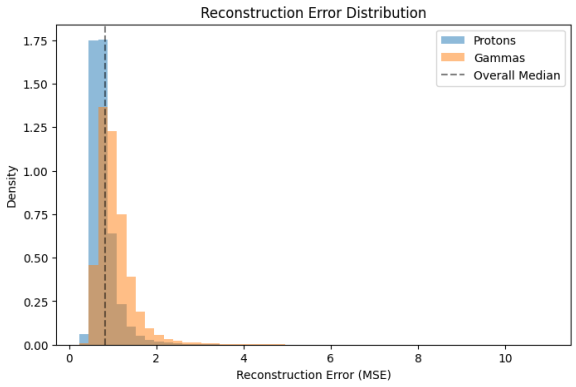
- The use of logarithmic normalization (Section 7.7).
- The application of the Leakage quality cut (Section 7.9).
- The use of the extended mask (Section 7.5).
- The modifications made to the Hexa AE architecture to create the Modified Hexa AE (Section 7.8).

We observe that these are mainly modifications related to the input data and image preparation. These observations suggest that the quality of the input data is the most crucial point for improving anomaly detection with autoencoders in this context. Modifications to model architectures also had an impact, as shown by the significant improvement obtained with the Modified Hexa AE compared to the base Hexa AE (Section 7.8), but they are less decisive than data preparation.

The two best results are highlighted in bold in the table. The best model is the Modified Hexa AE with logarithmic normalization, leakage-based quality cut, and extended mask, achieving an AUC value of 0.7720 (Section 7.9). The second-best model is the Modified Hexa AE with logarithmic normalization, hillas-based quality cut, and extended mask, achieving an AUC value of 0.7510 (Section 7.8). The difference between these two models is the quality cut applied to the input images.



(a) Modified Hexa AE with logarithmic normalization, leakage-based quality cut, and extended mask



(b) Modified Hexa AE with logarithmic normalization, hillas-based quality cut, and extended mask

Figure 7.27: Reconstruction error distributions for protons and gammas obtained with the two best models: (a) Modified Hexa AE with logarithmic normalization, leakage-based quality cut, and extended mask, and (b) Modified Hexa AE with logarithmic normalization, hillas-based quality cut, and extended mask.

Figure 7.27 shows the reconstruction error distributions for protons and gammas obtained with these two best models. We observe that the model with the leakage-based quality cut (subfigure (a)) achieves better separation between the two distributions, which explains its superior AUC performance. The model with the hillas-based quality cut (subfigure (b)) also shows good

separation but to a lesser extent.

Compared to the initial results, such as those presented in Figure 7.8 of Section 7.3, we observe a clear improvement in the separation between the two classes. This time, the two distributions are no longer completely overlapping, allowing for much higher AUC values. This illustrates the importance of the various modifications made to the models and input data to improve anomaly detection performance.

In the current state of the art presented in Chapter 3, the study presented in [9] achieves an AUC value of 0.895. This study also uses simulated data from the SST-1M telescope but with a different approach based on supervised RFs. The input data are not exactly the same, as this study uses different quality cuts, the data are taken at 30° altitude, and Hillas parameters are used as input features. However, this provides an order of magnitude of the performance achievable with supervised methods on this type of data. Compared to this result, the best unsupervised model presented in this chapter (Modified Hexa AE with leakage quality cut) achieves an AUC value of 0.7720, which is significantly lower.

It remains important to note that autoencoders are unsupervised models, meaning they do not use class labels (proton or gamma) during training. This characteristic partly explains the performance gap observed with supervised state-of-the-art methods. Nevertheless, the results obtained in this chapter show that it is possible to achieve reasonable performance in anomaly detection, even with unsupervised models.

Chapter 8

Autoencoder-Based Preprocessing for Supervised Learning

This chapter presents another application of autoencoders for the analysis of IACT data. Instead of using autoencoders for anomaly detection, as presented in the previous chapter, here they are employed to learn a compact, low-dimensional representation of images, which is then used to reconstruct and denoise them. The goal is to leverage these reconstructed images for supervised classification (proton/gamma) and regression (energy reconstruction) tasks.

8.1 Motivation and introduction

In the previous chapter, it was demonstrated that autoencoders can be used to separate proton and gamma events based on image reconstruction error. However, the results obtained remain significantly lower than those of supervised methods such as RFs and CNNs, as presented in Chapter 3.

The idea of this chapter is therefore to use autoencoders not for anomaly detection, but to leverage their ability to learn compact, low-dimensional representations of images that can be used to reconstruct high-quality images. The hypothesis is that autoencoders can learn a latent representation that preserves the relevant physical structures of air showers while mitigating noise and instrumental artifacts. Importantly, these compact representations also allow for data compression, which can be crucial when storage space or bandwidth is limited. The reconstructed images can then be used to extract relevant physical parameters, such as Hillas parameters, or even be used directly as inputs for supervised models such as CNNs.

To further investigate the impact of the latent space dimension on image reconstruction quality, three different latent space sizes are investigated: 32, 64, and 128. This will allow to evaluate how the size of the latent space affects the quality of the reconstructed images and, consequently, the performance of supervised tasks using these images.

The main objective of this chapter is to assess the extent to which reconstructed images produced by autoencoders preserve relevant information for supervised tasks, such as proton/gamma clas-

sification and energy regression. To obtain a reliable and meaningful comparison, for each task, a supervised model based on RFs will be trained using features extracted from the original images, and then three other RFs will be trained using features extracted from images reconstructed by autoencoders with different latent space sizes.

The performance of the four models will then be compared to evaluate the impact of using the reconstructed images. Thus, there will be four models per task (classification and regression): one using the original images and three using reconstructed images with different autoencoders. The general methodology followed in this chapter is as follows:

1. Design and train three autoencoders capable of effectively reconstructing images of air showers.
2. Extract physical features from both the original images and the reconstructed images.
3. Train four RFs for proton/gamma classification: one using features extracted from the original images, and three using those extracted from the reconstructed images.
4. Compare the performance of the four classification models to evaluate the impact of using the reconstructed images and different latent space sizes.
5. Train four RFs for energy regression: one using features extracted from the original images, and three using those extracted from the reconstructed images.
6. Compare the performance of the four regression models to evaluate the impact of using the reconstructed images and different latent space sizes.

It would also be interesting to test the direct use of reconstructed images as inputs to a classification or regression CNN, as this type of model is particularly effective for image analysis. However, this would require exploring a large number of possible architectures and configurations, which is not the purpose of this chapter. Therefore, the focus is on the use of RFs, which are supervised models that are simpler to configure and train. Additionally, RFs are commonly used in the field of gamma-ray astronomy, therefore, their optimal configurations are well known. It is thus quick and easy to test the impact of using reconstructed images on their performance.

8.2 Dataset and preprocessing

To ensure that the experiments on the different models are comparable, it is important to use the same dataset and quality cuts. This ensures that the observed performance differences are due to the modifications made by the autoencoder and not to variations in the data used.

For this, the same simulated dataset as in the previous chapter is used, comprising images of air showers generated by proton and gamma events. However, unlike the previous chapter, where autoencoders were trained only on proton images, here the autoencoders are trained on a mixed set of proton and gamma images. The goal is for the autoencoder to effectively learn to reconstruct both types of images.

The quality cuts used for this dataset are not the same as in the previous chapter. The goal of this chapter is also to compare, as closely as possible, the performance of supervised models with those present in the state of the art, as presented in Chapter 3. This comparison is not intended

to be perfectly equivalent but provides an order of magnitude relative to current methods.

Therefore, the same quality cuts as those used in the SST-1M pipeline [46] are used to be in conditions as close as possible to the state of the art. The quality cuts applied are as follows:

- **Intensity cut:** Only images with an intensity greater than 50 p.e. are retained. This also implies that Hillas parameters must be computable.
- **Leakage cut:** Only images with a leakage-2 less than 0.7, corresponding to a border range of two pixels, are retained.

As the RFs of the SST-1M pipeline are trained separately for the two telescopes, this chapter only uses the simulated images of the SST-1M-1 telescope, corresponding to the mono telescope configuration. The input data are also balanced between proton and gamma events by undersampling the gammas to have the same number of images for each type of event.

8.3 Autoencoder configuration for reconstruction

For image reconstruction, autoencoders based on the models from the previous chapter are used. However, the goal is not to choose the one with the best performance in anomaly detection, but the one that provides the best image reconstruction. For this, the observations made in the previous chapter regarding the reconstruction quality of the different models are relied upon.

First, the model will be based on the Hexa AE model, as this type of model has shown better reconstruction capability than Flat AE models. Additionally, image compression capacity is important in this context, and Graph AE models do not allow for true image compression, as explained in Chapter 6. Next, logarithmic normalization of the images is chosen, as it has shown better reconstruction quality than min-max normalization, particularly for high-intensity pixels, as explained in Section 7.7.

In all the models presented in the previous chapter, the Hexa AE model with logarithmic normalization that achieved the lowest median reconstruction error is Hexa AE (log), as presented in Section 7.7. It achieves a median reconstruction error of 0.3733 for protons and 0.3975 for gammas. This model will therefore be used as a base for the reconstruction experiments in this section. These error values were obtained by training the model only on proton images.

However, to improve reconstruction quality, it is necessary to train the model on both proton and gamma images. Additionally, minor modifications to the model architecture are made to try to improve reconstruction quality. In Section 7.10.1, it was noted that using bicubic interpolation in the decoder improved the visual quality of reconstructed images. Also, in Section 7.8, it was shown that adding batch normalization layers resulted in a more stable model and improved reconstruction quality. It therefore seems relevant to test these two modifications in this context.

Each modification is tested separately, as well as their combination. The term "P+G" is added to the model name to indicate that it is trained on a mixed set of proton and gamma images, to avoid confusion with the models from the previous chapter trained only on proton images. The base model used is therefore Hexa AE (log) and the variants tested are as follows:

- **Hexa AE P+G:** Hexa AE (log) presented in Section 7.7.
- **Hexa AE P+G Bicubic:** Hexa AE (log) with bicubic interpolation in the decoder.
- **Hexa AE P+G BN:** Hexa AE (log) with added batch normalization layers.
- **Hexa AE P+G Bicubic BN:** Hexa AE (log) with bicubic interpolation in the decoder and added batch normalization layers.

The models are trained on a dataset comprising images of proton and gamma events with the quality cuts explained in the previous section. The images are preprocessed with logarithmic normalization and an extended mask is used, identically to the Hexa AE (log) model presented in Section 7.7. The reconstruction error is measured on a separate test set. The models are evaluated based on the median reconstruction error.

Table 8.1: Reconstruction error statistics for different Hexa AE P+G variants trained on protons and gammas. P stands for protons and G for gammas.

Model	**Median P**	**Median G**	**Mean P**	**Mean G**	**Std P**	**Std G**
Hexa AE P+G	0.3397	0.3406	0.3476	0.3526	0.0843	0.0995
Hexa AE P+G Bicubic	0.3467	0.3538	0.3571	0.3694	0.0906	0.1070
Hexa AE P+G BN	0.3592	0.3736	0.3707	0.3919	0.0925	0.1149
Hexa AE P+G Bicubic BN	0.3732	0.3907	0.3868	0.4108	0.0981	0.1191

The results, presented in Table 8.1, show that the Hexa AE P+G model achieves the lowest median reconstruction error for both protons and gammas. Adding bicubic interpolation or batch normalization layers does not improve image reconstruction quality. The combination of both modifications also results in a higher reconstruction error. Therefore, the Hexa AE P+G model trained on both proton and gamma events is selected for the image reconstruction part of this chapter.

Figures 8.1 and 8.2 show examples of reconstructed images for proton and gamma events, respectively. It can be observed that the reconstructed images retain the main features of the air showers, although high intensities are slightly attenuated. The reconstructions remain very faithful to the input images and clearly visualize the shower structures. The Hexa AE P+G model has a latent space size of 64, allowing for a compression factor of about 20 compared to the original image size (1,296 pixels). This demonstrates that the model can effectively capture relevant information from the images while reducing their size.

To identify the impact of latent space size on the RFs, versions of this model with latent space sizes of 32 and 128 were also trained. For simplicity, these models are referred to as Hexa AE P+G (32) and Hexa AE P+G (128), respectively. The Hexa AE P+G model with a latent space size of 64 is referred to as Hexa AE P+G (64).

The statistics of the reconstruction errors obtained are presented in Table 8.2. As expected, increasing the latent space size improves the quality of image reconstruction, while reducing it degrades it. However, this does not necessarily mean that the model with the largest latent space size will yield the best results for supervised tasks.

Therefore, these models will be used to reconstruct images in the rest of this chapter, in order to

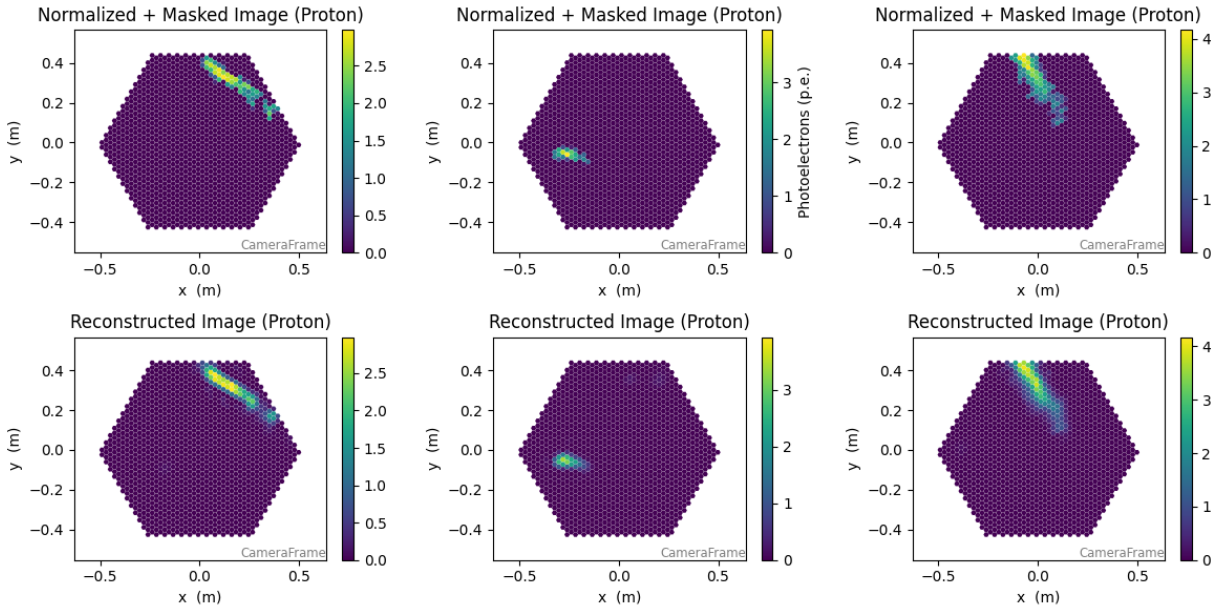


Figure 8.1: Example of reconstructed proton images using Hexa AE P+G trained on proton and gamma events. The first row shows the normalized and masked input images, and the second row shows the corresponding reconstructed images. The same scale is used for each original image and its reconstruction.

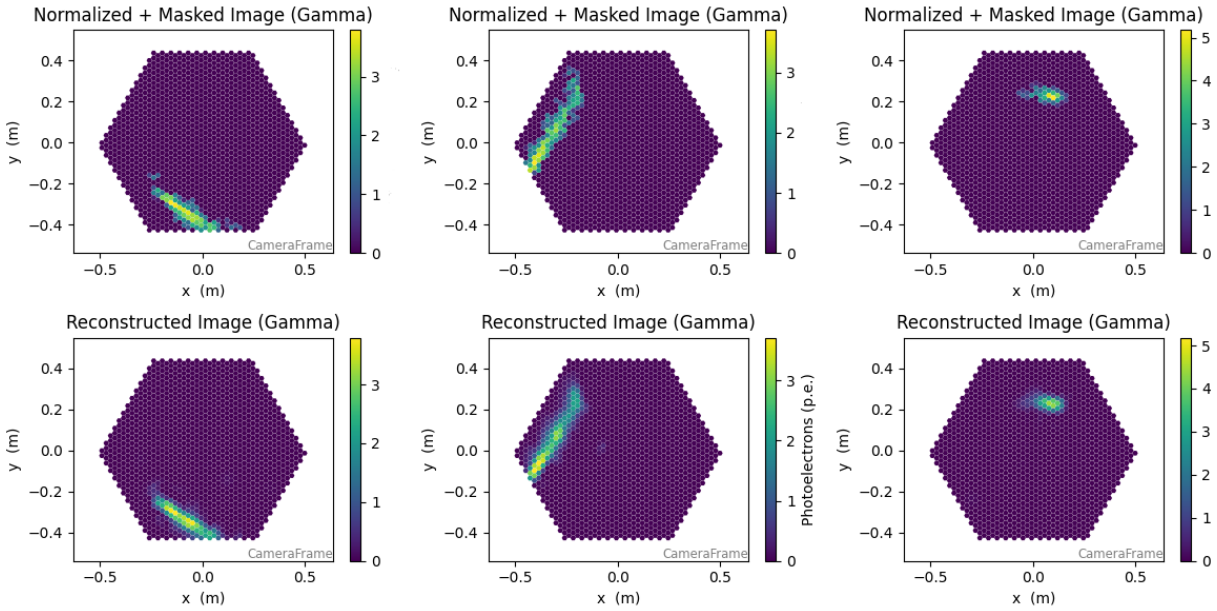


Figure 8.2: Example of reconstructed gamma images using Hexa AE P+G trained on proton and gamma events. The first row shows the normalized and masked input images, and the second row shows the corresponding reconstructed images. The same scale is used for each original image and its reconstruction.

evaluate the impact of using reconstructed images and different latent spaces on the performance of proton/gamma classification and energy regression.

Table 8.2: Reconstruction error statistics for Hexa AE P+G variants with different latent space sizes trained on protons and gammas. P stands for protons and G for gammas.

Model	Median P	Median G	Mean P	Mean G	Std P	Std G
Hexa AE P+G (32)	0.4108	0.4267	0.4259	0.4439	0.1155	0.1263
Hexa AE P+G (64)	0.3397	0.3406	0.3476	0.3526	0.0843	0.0995
Hexa AE P+G (128)	0.2949	0.2971	0.3022	0.3101	0.0746	0.0927

8.4 Hillas parameter and other feature reconstruction

Before extracting relevant physical features from the images for training supervised models, a preprocessing step must be applied to the images reconstructed by the autoencoders. Indeed, the autoencoders were trained on logarithmically normalized images. Therefore, to obtain the correct values, an inverse logarithmic denormalization must be applied. For this, the `expm1` function [59] from the `numpy` library is used; it is the inverse of the `log1p` function [48] used for logarithmic normalization of the images. It performs the following operation:

$$\text{expm1}(x) = e^x - 1 \tag{8.1}$$

To determine which features to use for the supervised tasks, those used in the SST-1M pipeline [60] for a single telescope (mono mode) are relied upon. The extracted features are as follows:

- **Hillas parameters:** The characteristics describing the shape and orientation of the shower image, including:
  - **Log intensity:** the logarithm of the total intensity of the image.
  - **Width:** the standard spread along the minor-axis.
  - **Length:** the standard deviation along the major-axis.
  - **Skewness:** the measure of the asymmetry.
  - **Kurtosis:** the measure of the tailedness.
  - **X:** the centroid x-coordinate.
  - **Y:** the centroid y-coordinate.
- **Leakage parameter:**
  - **Intensity width 2:** Intensity in photo-electrons after cleaning that are in the camera border of width=2 pixels.
- **Timing parameter:**
  - **Timing Slope:** Slope of arrival times along the main shower axis.
- **Other features:**
  - **WL:** A composite feature defined as the ratio between Width and Length: WL = Width / Length.

The Hillas parameters can be calculated using the `hillas_parameters` function [25], the leakage parameter with the `leakage_parameters` function [61], and the timing parameter with the `timing_parameters` function [62] from the `ctapipe` library [20]. The WL feature is a ratio calculated directly from the Width and Length Hillas parameters. These functions require cleaned images using a mask. To match the procedure used in the SST-1M pipeline [46], a

mask is calculated using the `tailcuts_clean` [23] and `time_delta_cleaning` [47] functions with the same parameters as in the latter. This mask is then applied to the original images before calculating the different features. For images reconstructed by all three autoencoder models, the same mask is used to calculate the extended mask described in Section 7.5. This extended mask is applied to the images before they are fed into the autoencoders, as it is necessary for proper functioning.

The only difference between the features extracted from the original images and those extracted from the reconstructed images is therefore related to the direct functioning of the autoencoders. All other steps from image preprocessing to feature extraction are identical to ensure a fair comparison between the two datasets.

8.5 Proton/gamma classification with reconstructed features

The features extracted from the original images and the reconstructed images are then used to train RFs for proton/gamma classification. Four separate RFs are trained: one using features extracted from the original images, and three others using features extracted from images reconstructed by the three autoencoder models with different latent space sizes (32, 64, and 128).

For this, the `RandomForestClassifier` class [63] from the `scikit-learn` library [64] is used. The model hyperparameters are the same as those used in the SST-1M pipeline [60], namely:

- **n_estimators:** 100
- **criterion:** "gini"
- **max_depth:** 30
- **min_samples_split:** 10
- **min_samples_leaf:** 10
- **min_weight_fraction_leaf:** 0.0
- **max_leaf_nodes:** null
- **min_impurity_decrease:** 0.0
- **bootstrap:** True
- **oob_score:** False
- **warm_start:** False
- **class_weight:** null

The values for the hyperparameters not listed here are the default ones. The models are trained on a training set and then evaluated on a separate test set. As in the previous chapter, the performance of the models is evaluated using the AUC metric.

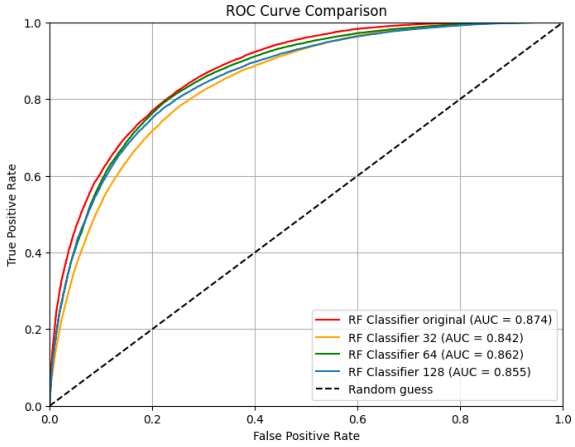
The results, presented in Table 8.3, show that both RFs achieve similar performance in terms of AUC. The model using features extracted from the original images achieves a slightly higher AUC (0.874) compared to the models using features extracted from reconstructed images. Among the latter, the model using images reconstructed by the Hexa AE P+G (64) model achieves the highest AUC (0.862), followed by the Hexa AE P+G (128) model (0.855) and finally the

Table 8.3: Performance of Random Forest classifiers for proton/gamma classification using features from original and reconstructed images.

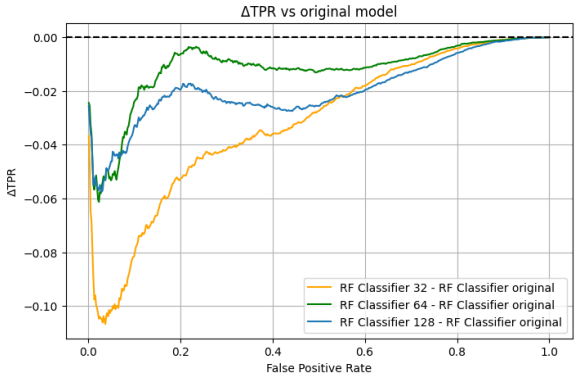
Model	**AUC**
RF with features from original images	0.874
RF with features from images reconstructed by Hexa AE P+G (32)	0.842
RF with features from images reconstructed by Hexa AE P+G (64)	0.862
RF with features from images reconstructed by Hexa AE P+G (128)	0.855

Hexa AE P+G (32) model (0.842). This indicates that increasing the latent space size does not necessarily improve classification performance, even though image reconstruction quality improves with a larger latent space size. This may be because a model with a larger latent space can capture additional details in the images that are not relevant for the proton/gamma classification task.

However, the difference between the four models is relatively small, suggesting that using images reconstructed by the autoencoders does not lead to a significant loss of relevant information for proton/gamma classification. This is particularly true for the Hexa AE P+G (64) model, which achieves an AUC very close to that of the model using original images.



(a) ROC curves for the four Random Forest classifiers.



(b) Difference in True Positive Rate (ΔTPR) between each RF with features from reconstructed images and the RF with features from original images.

Figure 8.3: Comparison of Random Forest classifiers trained on features from original and reconstructed images. (a) ROC curves for each model. (b) ΔTPR highlighting performance differences.

Figure 8.3 shows the ROC curves and the difference in True Positive Rate (ΔTPR) between each model using reconstructed images and the model using original images. It can be observed that the ROC curves of the four models are very similar, especially between the model using original images and the models using images reconstructed by Hexa AE P+G (64) and Hexa AE P+G (128). The images reconstructed by Hexa AE P+G (32) lead to a slightly lower performance, as indicated by the larger negative ΔTPR values across most of the False Positive Rate range. Hexa AE P+G (64) shows the smallest performance difference compared to the original model, with ΔTPR values close to zero across most of the FPR range. This confirms that using images reconstructed by Hexa AE P+G (64) allows for proton/gamma classification performance very

close to that obtained with the original images.

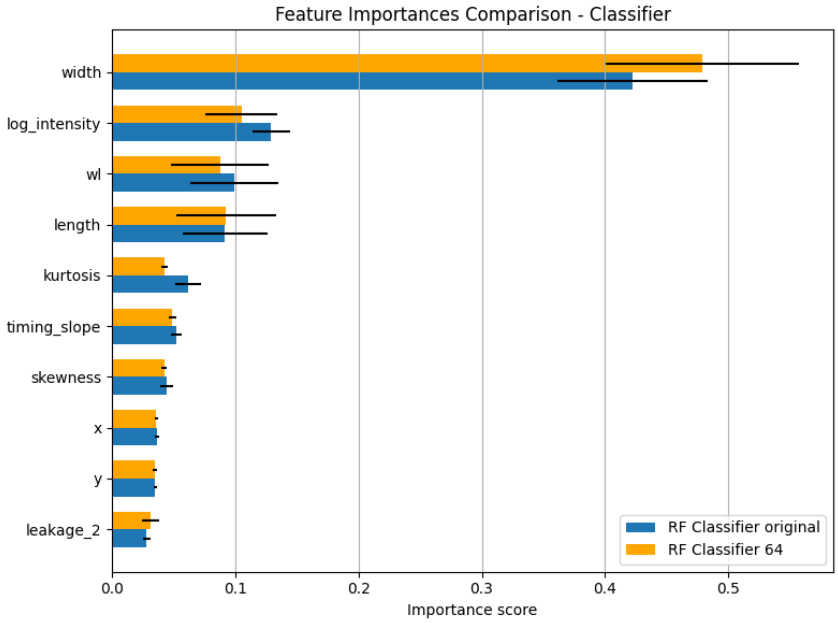


Figure 8.4: Feature importance for Random Forest classifiers trained on features from original images and images reconstructed by Hexa AE P+G (64).

Figure 8.4 shows the feature importance for the RF trained on features from the original images and the RF trained on features from images reconstructed by Hexa AE P+G (64). We can see that the feature importances are very similar for both models, with the same features standing out as the most important. These are width, log-intensity, the WL ratio and length. This suggests that the RF trained on features calculated from images reconstructed by the autoencoder manages to identify the same relevant physical features as the RF trained on the original features, and that the autoencoder has succeeded in preserving this information content.

In comparison to the study presented in [9], the most important features are also the same with similar importance. The best AUC obtained with features derived from the reconstructed images (0.862) is slightly lower than that reported in the state of the art (0.895). The experimental conditions are not exactly the same; the data used are different (a zenith angle of 20° here versus 30° in the state-of-the-art study), and the model in the state of the art is optimized for maximum performance, compared to the standard hyperparameters used here. Therefore, a performance difference is expected. It is important to note that the RF trained on features from the original images achieves an AUC of 0.874, which is also slightly lower than the state-of-the-art result. This suggests that the difference in performance is not primarily due to the use of reconstructed images but may be related to other factors.

To conclude, the use of images reconstructed by the Hexa AE P+G models allows for proton/gamma classification performance comparable to that obtained with the original images. These results suggest that using autoencoders as a method for compressing the images is effective, as it preserves the essential information for classification while reducing the data size. However, there is no improvement in performance compared to using the original images.

8.6 Energy Reconstruction with Random Forest

The features extracted from the original images and the reconstructed images are then used to train RFs for energy regression. Similar to the classification task, four separate RFs are trained: one using features extracted from the original images, and three others using features extracted from images reconstructed by the three autoencoder models with different latent space sizes (32, 64, and 128).

For this, the `RandomForestRegressor` class [65] from the `scikit-learn` library [64] is used. The model hyperparameters are the same as those used in the SST-1M pipeline [60], namely:

- **n_estimators:** 150
- **criterion:** "squared_error"
- **max_depth:** 30
- **min_samples_split:** 10
- **min_samples_leaf:** 10
- **min_weight_fraction_leaf:** 0.0
- **max_leaf_nodes:** null
- **min_impurity_decrease:** 0.0
- **bootstrap:** True
- **oob_score:** False
- **warm_start:** False

The values for the hyperparameters not listed here are the default ones. The models are trained on a training set composed only of gamma events and then evaluated on a separate test set also composed only of gamma events.

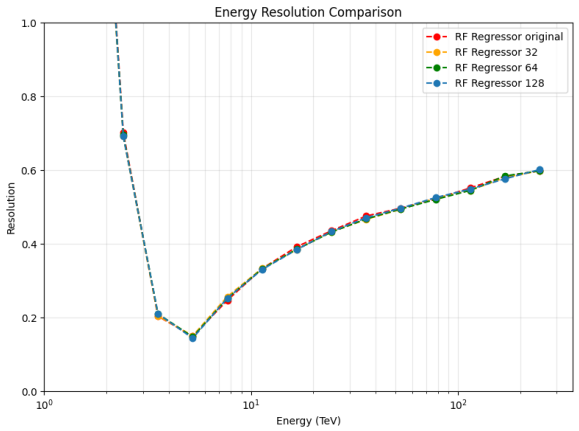
To compare the regression models, performance is evaluated using energy resolution [1]. This metric is calculated from the distribution of the relative error between the reconstructed energy E_{reco} and the true energy E_{true}, defined as follows:

$$\Delta E_{\text{rel}} = \frac{E_{\text{reco}} - E_{\text{true}}}{E_{\text{true}}} \tag{8.2}$$

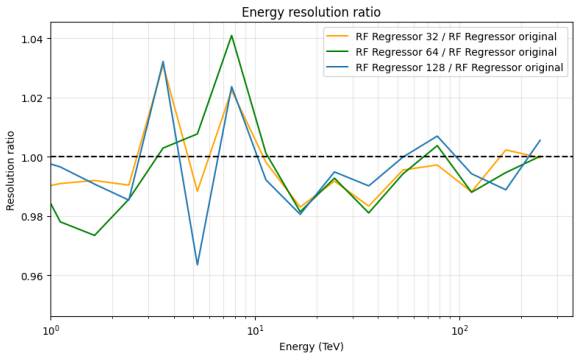
The energy resolution is then defined as the half-width of the interval centered on zero that contains 68.27 % of the ΔE_{rel} distribution. This indicates that 68.27 % of events have a relative error within this interval. The choice of 68.27 % corresponds to the fraction of events contained within a $\pm 1\sigma$ interval for a Gaussian distribution.

Energy resolution can vary significantly depending on the energy of the events. Therefore, to obtain a more detailed evaluation of the models' performance, energy resolution is calculated in bins of true energy. This allows for a clear observation of the models' behavior at different energies. The implementation used to calculate energy resolution in energy bins is the one present in the `ctaplot` library [66].

Figure 8.5 shows the energy resolution as a function of true energy for the four regression



(a) Energy resolution as a function of true energy for the four Random Forest regressors.



(b) Ratio of the resolution of each RF trained on features from reconstructed images to that of the RF trained on original features. Values < 1 indicate better performance, while values > 1 indicate worse performance.

Figure 8.5: Comparison of Random Forest regressors trained on features from original and reconstructed images. (a) Energy resolution as a function of true energy for each model. (b) Energy resolution ratio highlighting performance differences.

models, as well as the ratio of the energy resolution between each model using reconstructed images and the model using original images. For energy resolution, we observe that all models have significant difficulty at low energies (around 1 TeV). This is due to the fact that images of low-energy events have low intensity and are therefore more difficult to analyze. A resolution of 15 % is observed at around 4.5 TeV for all models, corresponding to the best performance achieved. Then, the energy resolution gradually increases again with true energy. This behavior is quite similar to that observed in the state of the art [9], although the values obtained here are significantly higher. This difference is not surprising, as energy regression is a more challenging task and requires learning a continuous target. Also, the amount of data available for training is lower, especially at very high energies, making the task more difficult for the models to learn.

The behavior of the four models is very similar across the entire energy range. By comparing the ratio of energy resolution between the models, we observe that the differences are relatively small. A maximum difference of around ± 4 % is observed. At high energies, the models using reconstructed images even show a slight improvement, on the order of 1 to 2 %, compared to the model using original images. This suggests that the features extracted from images reconstructed by the autoencoders retain sufficient relevant information for the energy regression task, allowing the models to perform comparably to those using features from the original images.

Figure 8.6 shows the feature importance for the RF trained on features from the original images and the RF trained on features from images reconstructed by Hexa AE P+G (64). We observe that the feature importances are very similar for both models, with the same features standing out as the most important. These include length, timing-slope, and log-intensity. Both models thus identify the same relevant physical features for energy regression, and the autoencoder has succeeded in preserving this information content. In the state of the art [9], the order of feature importance is also in this order, although length has a significantly higher importance than the other features.

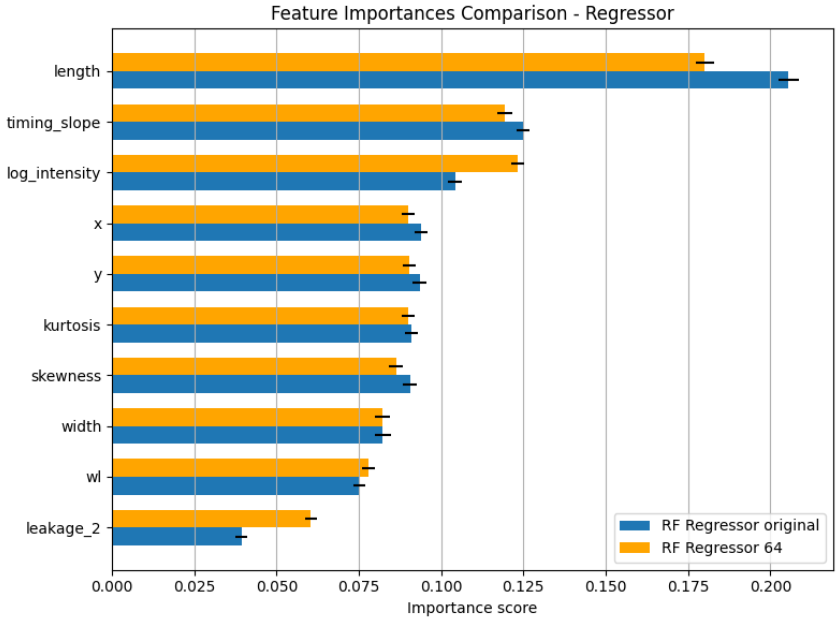


Figure 8.6: Feature importance for Random Forest regressors trained on features from original images and images reconstructed by Hexa AE P+G (64).

The conclusion is therefore the same as for proton/gamma classification: using images reconstructed by the Hexa AE P+G models allows for energy regression performance comparable to that obtained with the original images. This suggests that autoencoders can be used as an intermediate step, without significant loss of performance in the energy regression task, but without significant improvement either. At the same time, these results indicate that autoencoders provide an effective means of compressing the data while preserving the essential information.

Chapter 9

Discussion

This chapter discusses the results obtained in the previous chapters, highlighting the main conclusions, comparing them with the state of the art, pointing out the limitations of the proposed approach, and discussing potential future work to improve and extend it.

9.1 Summary of the main findings

In this work, the focus was on using autoencoders for anomaly detection (proton/gamma separation) in IACT data, while also exploring their application to image reconstruction for supervised tasks such as classification and regression.

For anomaly detection, several autoencoder architectures (Flat AE, Square AE, Graph AE, Hexa AE) and different image preprocessing techniques (min-max normalization, logarithmic normalization, masks, quality cuts) were tested. These experiments allowed me to identify the most effective configurations and draw the following main conclusions:

- **Impact of log normalization:** Logarithmic normalization of images significantly improves performance compared to min-max normalization. It allows for a better distribution of input values, facilitating learning for the autoencoder and the reconstruction of high intensities.
- **Use of masks:** The use of a mask helps to remove irrelevant pixels and retain only important information from the images. Extending the mask further improves image quality by preserving more useful information while eliminating noise. This masking strategy allows the autoencoder to focus on significant pixels and not be disturbed by unnecessary noise.
- **Quality cuts:** Applying quality cuts helps filter out difficult and noisy data, improving overall model performance by focusing on higher-quality events. However, this also reduces the amount of data available for training and evaluation, which may limit the model's generalization capability.
- **Model architecture:** The Hexa AE architecture, adapted to the hexagonal structure of the images, offers better performance than other tested architectures. Flat AE and Square

AE architectures do not effectively capture the structure of the data, while Graph AE does not achieve true image compression. Additional modifications to this architecture, such as adding batch normalization layers, further improved proton/gamma separation performance.

- **Reconstruction capability vs anomaly detection:** A model that reconstructs images well is not necessarily a good model for anomaly detection. If the model is too effective in reconstruction, it may learn to reconstruct both protons and gammas equally well, making separation difficult. A model with moderate reconstruction capability seems more suitable for this task.
- **Performance comparison:** The results obtained with autoencoders remain lower than those of supervised state-of-the-art methods, which is expected given the unsupervised nature of autoencoders. However, the achieved performances are reasonable and demonstrate the potential of autoencoders for this application.

Results show that modifications to data preparation have a significant impact on anomaly detection performance. The modifications made to the Hexa AE architecture also contribute to improved performance. Overall, these findings highlight the potential of autoencoders for anomaly detection in IACT data, while also indicating areas for further improvement.

Furthermore, the use of autoencoders for image reconstruction and data compression, applied to supervised tasks of proton/gamma classification and energy regression, was also explored, and the main conclusions are as follows:

- **Image compression:** The Hexa AE P+G (64) model effectively reconstructs images of air showers while achieving significant compression (factor of about 20, latent space size of 64, compared to 1,296 original pixels). The reconstructed images yield very similar results to the original images in supervised tasks, indicating that essential information is preserved despite the compression.
- **Feature importance:** The most important features for classification and regression remain the same, whether using original images or reconstructed images, indicating that the autoencoder manages to preserve the same relevant physical information.
- **Classification performance:** Using images reconstructed by the autoencoder to extract features allows for proton/gamma classification performance comparable to that obtained with the original images. The performance difference is relatively small, suggesting that the autoencoder preserves relevant information for classification.
- **Regression performance:** Similarly, for energy regression, using features from reconstructed images yields performance close to that obtained with original images.
- **Comparison with the state of the art:** For classification, the results obtained with RF trained on features from reconstructed images are slightly lower than those reported in the state of the art, but the data used are not exactly the same, which may explain this difference. For regression, the performance is worse than the state of the art, but this is

also observed when using RF trained on features from original images, suggesting that the difference is due to data differences rather than the use of autoencoders.

- **Overall benefit:** The use of the autoencoder allows for image dimension reduction, noise filtering, and preservation of essential physical information, thus providing an effective method for preparing data for supervised models.

Overall, these results demonstrate that using autoencoder-reconstructed images is a promising approach for supervised tasks, as it preserves essential physical information while effectively reducing noise and compressing the data.

9.2 Comparison with State-of-the-Art Methods

In Chapter 3, current supervised methods for analyzing IACT data were presented, notably the use of RFs for proton/gamma classification. This method relies on extracting relevant physical features from images, such as Hillas parameters.

The study presented in [9] achieved a proton/gamma classification performance with an AUC of 0.895 using RFs trained on features extracted from the original images. The model was trained on simulated data for the SST-1M-1 telescope (mono telescope), with specific quality cuts and a zenith angle of 30°. Other results are presented in the study for stereo configurations and other zenith angles, but here the focus is on the configuration closest to the experiments.

Table 9.1: Comparison of classification performance between autoencoder-based methods and state-of-the-art supervised methods.

Method	**AUC**
Autoencoder for anomaly detection Modified Hexa AE (leakage-based)	0.772
RF with features from images reconstructed by Hexa AE P+G (64)	0.862
RF with features from original images	0.874
State-of-the-art RF (sst-1m)	0.895

Table 9.1 summarizes the proton/gamma classification performance obtained with the different methods. The performance achieved by the state of the art is significantly higher than that obtained with the Modified Hexa AE autoencoder for anomaly detection and slightly higher than that obtained with reconstructed images for classification. However, the data used are not exactly the same: in this project, the data are at a zenith of 20°, and less data are used. The dataset remains very similar, and there is no doubt that using only an autoencoder for anomaly detection performs significantly below the supervised state of the art. However, using images reconstructed by an autoencoder allows for performance close to the state of the art, and the difference may be solely due to data differences. This is also supported by the fact that the RF trained on features from the original images achieves an AUC slightly lower than the state-of-the-art result. Also, the most important features remain the same, suggesting that using autoencoders preserves relevant physical information.

For energy regression, it is not appropriate to directly compare the results to the state of the art. Indeed, the energy resolution is calculated per bin, and the energy intervals are not identical

between the study [9] and this work. Comparing raw values for each bin could therefore be misleading. It is more appropriate to assess performance in terms of overall trends between the two studies.

The trends observed in the energy regression results are consistent with those reported in the state of the art. The energy resolution is very high at low energy, then decreases sharply towards 4 TeV before slightly increasing again at higher energies. This trend is similar to that observed in the study [9], although the absolute values of the energy resolution are higher in this work. However, this difference is also observed when using the original images, suggesting that the difference comes from the data used rather than the use of autoencoders.

Therefore, although autoencoders do not outperform current methods, they show interesting potential for analyzing this type of data particularly in terms of image compression and as a preprocessing step for supervised tasks.

9.3 Limitations of the proposed approach

Despite the promising results obtained in this work, several limitations must be taken into account:

- **Limited data:** The dataset used in this work is limited in terms of diversity and quantity. All data are simulated at a zenith angle of 20°.
- **Simulated data only:** The models were trained and evaluated only on simulated data. Applying the models to real data could yield different results due to differences between simulated and real datasets, such as background noise, observation conditions, and instrumental imperfections. However, it remains very difficult to evaluate performance on real data, as the labels (proton/gamma) are not known.
- **Limited hyperparameter tuning:** The hyperparameter tuning of the autoencoder models and RFs was limited. For the RFs, the same hyperparameters as in the SST-1M pipeline were used to ensure the fairest possible comparison with the state of the art. However, these hyperparameters are not necessarily optimal for features extracted from images reconstructed by autoencoders. A more thorough exploration could improve performance.
- **Limited exploration of architectures:** Although several autoencoder architectures were tested, the exploration of possible architectures remains limited. Different architectural modifications, such as adding layers, modifying activation functions, or using other techniques, could potentially improve performance.
- **Exclusive use of images:** This work focused primarily on the use of DL1 images, with limited use of peak time information. Other types of data could contain additional useful information for anomaly detection and image reconstruction.

These limitations highlight the need to continue the research in this area to improve the performance of autoencoders and their applicability to real data. However, these limitations also

suggest several avenues for future work aimed at improving and extending the results obtained in this work.

9.4 Future Work

Several directions for future research can be considered to improve and extend the results obtained in this work:

- **Exploration of different autoencoder architectures:** Testing other architectures, such as Variational Autoencoders (VAE), deep convolutional autoencoders, or recurrent autoencoders, could improve reconstruction quality and anomaly detection performance.
- **Use of full waveforms:** Further exploration of full waveforms (R1), which contain even more information than DL1 images, could improve current performance. In this work, only preliminary experiments were conducted, leaving many design choices and processing options still unexplored. Using autoencoders on R1 data could also enable substantial compression, as full waveforms represent the largest data volume in IACT observations.
- **Use of other data types:** Integrating other types of data, such as multi-telescope data, could enrich the inputs of the autoencoders and improve performance.
- **Thorough hyperparameter tuning:** Optimizing the parameters used for the RFs to improve the performance of supervised models using features extracted from reconstructed images. For autoencoders as well, a more thorough exploration of hyperparameters could yield better models.
- **Autoencoders for noise/event separation:** Leveraging autoencoders to distinguish events of interest from background noise. This case would be different from proton/gamma separation, as background noise can include various types of events unrelated to gamma rays or protons. The goal would be to detect and isolate relevant events for astrophysical analysis.
- **Autoencoders as feature extractors:** Exploiting the latent representations learned by autoencoders as input features for supervised models could capture more complex information than manually extracted features. This would reduce dependence on traditional physical features while maintaining a low number of features.
- **Pretraining with autoencoders:** Using autoencoders to pretrain supervised models, such as CNNs, could improve convergence and performance by providing better weight initialization.
- **Reconstructed images for supervised models:** Testing the direct use of images reconstructed by autoencoders as inputs for supervised models, such as CNNs. This would provide cleaner and less noisy data to supervised models and potentially improve performance compared to using original images.

These future research directions offer numerous possibilities to improve the use of autoencoders.

They could enable achieving performance comparable to, or even surpassing, current supervised methods while leveraging the advantages of unsupervised approaches.

Overall, this work demonstrates the potential of autoencoders for analyzing IACT data, both for image compression and reconstruction and for preparing features for supervised tasks. The identified avenues offer promising prospects for bringing the performance of unsupervised methods closer to that of current approaches.

Chapter 10

Conclusion

10.1 General Conclusion

Gamma-ray astronomy plays a crucial role in studying the most energetic astrophysical phenomena in the Universe, which emit gamma photons at extremely high energies. The analysis of these gamma rays relies on detecting the particle showers they produce when interacting with the Earth's atmosphere, through IACT. However, most events detected by these telescopes originate from hadronic cosmic rays. Therefore, a major challenge in this field is separating events of interest (gamma rays) from background events (hadronic cosmic rays), as well as accurately reconstructing their energy.

Current methods largely rely on supervised approaches, such as RFs, which require labeled data for training. The objective of this work was to explore the use of autoencoders, an unsupervised learning technique, for anomaly detection (proton/gamma separation) and image reconstruction of IACT data. The idea was to leverage the ability of autoencoders to learn compact and informative representations of the data while reducing noise and compressing images.

Results show that autoencoders, particularly the Modified Hexa AE architecture adapted to the hexagonal structure of the images, can be effective for anomaly detection in IACT data. The use of logarithmic normalization, masks to filter out irrelevant pixels, and quality cuts improved model performance. However, performance remains lower than that of current supervised methods.

For image reconstruction and data compression, autoencoders demonstrated their ability to preserve essential information needed for supervised tasks of proton/gamma classification and energy regression. The performance obtained with features extracted from reconstructed images is comparable to that obtained with original images, suggesting that autoencoders can be used as an effective intermediate step. Although no significant improvement in performance is observed, this allows strong data compression (about a factor of 20) while retaining relevant information.

Several avenues for future research can be explored to improve and extend these results. The images reconstructed by the autoencoders could be used directly as inputs for supervised models,

such as CNNs, to provide cleaner and less noisy data. Exploring other autoencoder architectures, such as Variational Autoencoders (VAE), as well as using waveforms (R1) could also enhance performance. Additionally, integrating other types of data, optimizing hyperparameters, and using autoencoders for signal/noise separation are promising directions.

Overall, this work highlights the potential of autoencoders as complementary tools for analyzing IACT data, especially in terms of image compression and data preparation for supervised methods.

10.2 Personal Conclusion

This final project allowed me to acquire new skills in technical, methodological, and scientific aspects. The use of autoencoders applied to IACT data enabled me to deepen my knowledge in machine learning, especially in unsupervised learning techniques. The unique hexagonal structure of the images pushed me to adapt and rethink classical image processing methods to effectively handle this specific data format.

An important part of this project was also focused on physics aspects, which are not my area of expertise. Therefore, it allowed me to broaden my knowledge in this field and better understand the scientific challenges related to gamma astronomy.

Overall, this project has been an enriching experience that allowed me to develop my technical and scientific skills while exploring an interesting and rapidly evolving research field.

Acronyms

AUC Area Under the Curve. 8, 21, 22, 36, 45, 51, 54, 56, 58, 61, 64, 65, 67, 72, 75–77, 84–86, 92

BDT Boosted Decision Tree. 7

CNN Convolutional Neural Network. 7, 8, 78, 79, 94

DL1 Data Level 1. 12–17, 24, 25, 36, 42, 46, 49, 73, 93, 94

FPR False Positive Rate. 22, 85

GAN Generative Adversarial Network. 9

GCN Graph Convolutional Network. 29, 30, 69, 70

HPC High Performance Computing. 18

IACT Imaging Atmospheric Cherenkov Technique. 1–3, 5, 8, 12, 78, 90–92, 94–97

MLP Multi-Layer Perceptron. 29, 30, 69

MSE Mean Squared Error. 9, 20, 21, 23, 38, 41

p.e. photoelectrons. 14, 80

PCA Principal Component Analysis. 9

ReLU Rectified Linear Unit. 24, 27, 28, 30, 34

RF Random Forest. 7, 8, 40, 77–81, 84, 86–88, 92–94, 96

RNN Recurrent Neural Network. 7, 8

ROC Receiver Operating Characteristic. 22

SiPM Silicon-Photomultipliers. 6

SST-1M Small-Sized Telescope - Single Mirror. 6, 8, 11, 12, 14–16, 49, 77, 80, 83, 84, 87, 92, 93

TPR True Positive Rate. 22, 85

Bibliography

[1] CTAO Consortium. "CTAO," Accessed: Dec. 19, 2025. [Online]. Available: https://www.ctao.org/

[2] Inductiveload, NASA. "A diagram of the electromagnetic spectrum (EMS)," Accessed: Dec. 19, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Electromagnetic_spectrum#/media/File:EM_Spectrum_Properties_edit.svg

[3] D. R. López-Coto, "Introduction to iact data analysis," in *First CTLearn Workshop*, 2024, pp. 1–8. [Online]. Available: https://indico.fis.ucm.es/event/29/contributions/558/

[4] C. de la Vaissière et al. "Gerbes Cosmiques," Accessed: Jan. 29, 2026. [Online]. Available: https://laradioactivite.com/articles/laboratoire/gerbes_cosmiques

[5] VERITAS Collaboration. "ACT Techniques and VERITAS Technology," Accessed: Jan. 29, 2026. [Online]. Available: https://veritas.sao.arizona.edu/about-veritas/atmospheric-cherenkov-technique-and-veritas-technologies/

[6] C. Alispach et al., "The sst-1m imaging atmospheric cherenkov telescope for gamma-ray astrophysics," *Journal of Cosmology and Astroparticle Physics*, vol. 2025, no. 02, p. 047, Feb. 2025, ISSN: 1475-7516. DOI: 10.1088/1475-7516/2025/02/047 [Online]. Available: http://dx.doi.org/10.1088/1475-7516/2025/02/047

[7] A. M. Hillas, "Cerenkov light images of eas produced by primary gamma," in *19th Intern. Cosmic Ray Conf-Vol. 3*, 1985.

[8] D. Nieto et al., *Ctlearn: Deep learning for gamma-ray astronomy*, 2019. arXiv: 1912.09877 [astro-ph.IM]. [Online]. Available: https://arxiv.org/abs/1912.09877

[9] Alispach, C. et al., *Observation of the crab nebula with the single-mirror small-size telescope stereoscopic system at low altitude*, 2025. DOI: 10.1051/0004-6361/202555292 [Online]. Available: https://doi.org/10.1051/0004-6361/202555292

[10] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[11] GeeksforGeeks. "Types of Autoencoders in Deep Learning," Accessed: Jan. 6, 2026. [Online]. Available: https://www.geeksforgeeks.org/numpy/types-of-autoencoders/

[12] R. Chalapathy and S. Chawla, *Deep learning for anomaly detection: A survey*, 2019. arXiv: 1901.03407 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1901.03407

[13] H. Huang, P. Wang, J. Pei, J. Wang, S. Alexanian, and D. Niyato, "Deep learning advancements in anomaly detection: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 12, no. 21, pp. 44 318–44 342, 2025. DOI: 10.1109/JIOT.2025.3585884

[14] T. Finke, M. Krämer, A. Morandini, A. Mück, and I. Oleksiyuk, "Autoencoders for unsupervised anomaly detection in high energy physics," *Journal of High Energy Physics*, vol. 2021, no. 6, Jun. 2021, ISSN: 1029-8479. DOI: 10.1007/jhep06(2021)161 [Online]. Available: http://dx.doi.org/10.1007/JHEP06(2021)161

[15] D. Heck, J. Knapp, J. Capdevielle, G. Schatz, T. Thouw, et al., "Corsika: A monte carlo code to simulate extensive air showers," *Report fzka*, vol. 6019, no. 11, 1998.

[16] K. Bernlöhr, "Simulation of imaging atmospheric cherenkov telescopes with corsika and sim_telarray," *Astroparticle Physics*, vol. 30, no. 3, pp. 149–158, 2008.

[17] CORSIKA developers. "CORSIKA - COsmic Ray SImulations for KAscade," Accessed: Jan. 8, 2026. [Online]. Available: https://www.iap.kit.edu/corsika/

[18] SST-1M Collaboration. "SST-1M Data Levels," Accessed: Jan. 3, 2026. [Online]. Available: https://sst1mpipe.readthedocs.io/en/latest/introduction.html#analysis-basics

[19] ctapipe developers. "ctapipe.tools.ProcessorTool Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.tools.process.ProcessorTool.html

[20] ctapipe developers. "Prototype CTA Pipeline Framework (ctapipe)," Accessed: Oct. 14, 2025. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/

[21] ctapipe developers. "ctapipe.calib.camera.calibrator.CameraCalibrator Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.calib.camera.calibrator.CameraCalibrator.html

[22] ctapipe developers. "ctapipe.image.ImageProcessor Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.image.ImageProcessor.html

[23] ctapipe developers. "ctapipe.image.tailcuts_clean Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.image.tailcuts_clean.html

[24] ctapipe developers. "Dl1cameracontainer — ctapipe documentation," Accessed: Oct. 14, 2025. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.containers.DL1CameraContainer.html

[25] ctapipe developers. "ctapipe.image.hillas.hillas_parameters Documentation," Accessed: Jan. 11, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.image.hillas.hillas_parameters.html

[26] ctapipe developers. "ctapipe.instrument.CameraGeometry Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/stable/api/ctapipe.instrument.CameraGeometry.html

[27] ctapipe developers. "ctapipe.visualization.bokeh.CameraDisplay Documentation," Accessed: Jan. 3, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.visualization.bokeh.CameraDisplay.html

[28] PyTorch contributors. "PyTorch Documentation," Accessed: Dec. 16, 2025. [Online]. Available: https://pytorch.org/docs/stable/

[29] PyG Team. "Pytorch geometric: A library built upon pytorch to easily write and train graph neural networks (gnns)," Accessed: Dec. 16, 2025. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/

[30] Constantin Steppa and Tim L. Holch. "Hexagdly: Processing hexagonal data with pytorch," Accessed: Dec. 24, 2025. [Online]. Available: https://github.com/ai4iacts/hexagdly

[31] PyTorch contributors. "PyTorch Documentation: torch.nn.modules.loss.MSELoss," Accessed: Dec. 16, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.MSELoss.html

[32] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980

[33] PyTorch contributors. "PyTorch Documentation: torch.optim.Adam," Accessed: Dec. 16, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html

[34] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," *Scientific Reports*, vol. 14, no. 1, p. 6086, 2024.

[35] cta-observatory. "DL1 Data Handler," Accessed: Dec. 21, 2025. [Online]. Available: https://github.com/cta-observatory/dl1-data-handler

[36] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[37] Rachit Tayal. "Deep Learning for Computer Vision," Accessed: Dec. 23, 2025. [Online]. Available: https://medium.com/data-science/deep-learning-for-computer-vision-c4e5f191c522

[38] GeeksforGeeks. "Apply a 2D Transposed Convolution Operation in PyTorch," Accessed: Dec. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/apply-a-2d-transposed-convolution-operation-in-pytorch/

[39] GeeksforGeeks. "Introduction to Graph Data Structure," Accessed: Dec. 23, 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/introduction-to-graphs-data-structure-and-algorithm-tutorials/

[40] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.02907

[41] PyG Team. "PyTorch Geometric Documentation: GCNConv," Accessed: Dec. 23, 2025. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GCNConv.html

[42] C. Steppa and T. L. Holch, "Hexagdly—processing hexagonally sampled data with cnns in pytorch," *SoftwareX*, vol. 9, pp. 193–198, 2019, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2019.02.010 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711018302723

[43] Red Blob Games. "Hexagonal Grids," Accessed: Dec. 24, 2025. [Online]. Available: https://www.redblobgames.com/grids/hexagons/

[44] Jorgecardete. "The Art and Science of Interpolation," Accessed: Jan. 14, 2026. [Online]. Available: https://medium.com/thedeephub/the-art-and-science-of-interpolation-b12b99f2e053

[45] Jason Chen. "Nearest Neighbor and Bilinear Interpolation," Accessed: Jan. 14, 2026. [Online]. Available: https://jason-chen-1992.weebly.com/home/nearest-neighbor-and-bilinear-interpolation

[46] SST-1M Collaboration. "SST-1M Data Processing Pipeline," Accessed: Dec. 30, 2025. [Online]. Available: https://github.com/SST-1M-collaboration/sst1mpipe/tree/main

[47] ctapipe developers. "ctapipe.image.cleaning.apply_time_delta_cleaning Documentation," Accessed: Jan. 4, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/v0.20.0/api/ctapipe.image.cleaning.apply_time_delta_cleaning.html

[48] NumPy Developers. "NumPy Documentation: numpy.log1p," Accessed: Dec. 31, 2025. [Online]. Available: https://numpy.org/devdocs/reference/generated/numpy.log1p.html

[49] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.03167

[50] Cmglee. "Comparison of 1D and 2D interpolation," Accessed: Jan. 14, 2026. [Online]. Available: https://commons.wikimedia.org/wiki/File:Comparison_of_1D_and_2D_interpolation.svg

[51] PyG Team. "conv.ChebConv Documentation," Accessed: Jan. 15, 2026. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.ChebConv.html

[52] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, 2017. arXiv: 1606.09375 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1606.09375

[53] PyG Team. "conv.GATConv Documentation," Accessed: Jan. 15, 2026. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATConv.html

[54] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1710.10903

[55] PyG Team. "conv.GINConv Documentation," Accessed: Jan. 15, 2026. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GINConv.html

[56] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, *How powerful are graph neural networks?* 2019. arXiv: 1810.00826 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.00826

[57] PyG Team. "pool.TopKPooling Documentation," Accessed: Jan. 15, 2026. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.TopKPooling.html

[58] PyG Team. "pool.SAGPooling Documentation," Accessed: Jan. 15, 2026. [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.SAGPooling.html

[59] NumPy Developers. "NumPy Documentation: numpy.expm1," Accessed: Jan. 30, 2026. [Online]. Available: https://numpy.org/devdocs/reference/generated/numpy.expm1.html

[60] SST-1M Collaboration. "SST-1M Data Configuration File," Accessed: Jan. 22, 2026. [Online]. Available: https://github.com/SST-1M-collaboration/sst1mpipe/blob/main/sst1mpipe/data/sst1mpipe_data_config.json

[61] ctapipe developers. "ctapipe.image.leakage_parameters Documentation," Accessed: Jan. 22, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.image.leakage_parameters.html

[62] ctapipe developers. "ctapipe.image.timing_parameters Documentation," Accessed: Jan. 22, 2026. [Online]. Available: https://ctapipe.readthedocs.io/en/latest/api/ctapipe.image.timing_parameters.html

[63] scikit-learn developers. "sklearn.ensemble.RandomForestClassifier Documentation," Accessed: Jan. 22, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[64] scikit-learn developers. "scikit-learn: Machine Learning in Python," Accessed: Jan. 22, 2026. [Online]. Available: https://scikit-learn.org/stable/

[65] scikit-learn developers. "sklearn.ensemble.RandomForestRegressor Documentation," Accessed: Jan. 22, 2026. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[66] T. Vuillaume, M. Linhoff, M. Jacquemont, and S. Bórquez, *cta-observatory/ctaplot: v0.6.5*, version v0.6.5, 2025. DOI: 10.5281/zenodo.17288481 [Online]. Available: https://doi.org/10.5281/zenodo.17288481

Appendix A

Source Code and Model Configurations

The appendices containing the source code and model configurations are available on the GitHub repository associated with this work: https://github.com/Chywou/TM_AE.