University of Milano - Bicocca

Physics Department

Bachelor's Degree Course

A Machine Learning Approach for Particle Tracking in RICH Detectors

Supervisor:
Martino Borsato

Co-Supervisor:
Maurizio Martinelli

Candidate:
Daniele Ghezzi

ACADEMIC YEAR 2024/2025

Abstract

The Ring Imaging Cherenkov (RICH) detectors of the LHCb experiment at CERN exploit Cherenkov radiation to perform particle identification (PID) in the second stage trigger (HLT2): by measuring the radius of the Cherenkov light projection on the photodetector plane, they enable the reconstruction of the particle's identity. This work explores a new possible application of RICH detectors in the first stage trigger (HLT1) as tracking devices. In fact, the center of the Cherenkov ring corresponds to the particle's position in the detector, providing an additional point for reconstructing the particle's trajectory.

However, the localization of the centers is extremely challenging, since a single pp event produces a large number of overlapping rings, especially in the central region of the photodetector plane. Due to the computational complexity of the task, supervised machine learning algorithms, in particular deep learning and computer vision methods, emerge as a natural choice. To obtain labeled data for training and validation of the neural networks, the events are simulated with a synthetic generator based on Monte Carlo techniques.

Two different models are studied, which incorporate distinct paradigms for approaching the problem: the YOLO (You Only Look Once) model treats it as a regression task and directly outputs the inferred coordinates of the centers, whereas the UNet model generates a probability heatmap whose peaks correspond to the centers' positions. These networks are compared within a consistent evaluation framework, also considering inference time. The results show that YOLO achieves the highest overall performance with the lowest inference time of about 13 ms on a NVIDIA A6000 GPU, while UNet provides higher precision on the detected peaks. Nevertheless, both models struggle to identify all ground truth centers: the best performing YOLO model correctly identifies only 52.7% of them on a generated dataset, indicating the need for further in-depth studies.

Contents

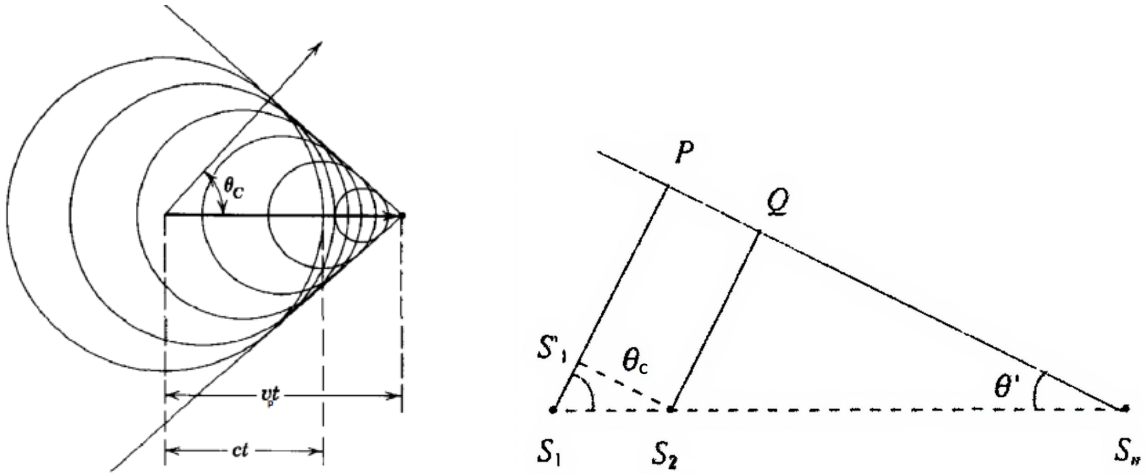
1. Introduction

The **LHCb detector** at CERN is one of the four experiments at the **Large Hadron Collider** (**LHC**), the world's most powerful particle accelerator ever built, alongside CMS, ATLAS and ALICE. LHCb is primarily designed to investigate the **matter-antimatter asymmetry** observed in the universe by studying decays that exhibit CP violation, typically in processes with charm (c) and beauty (b) quarks. The **Ring Imaging Cherenkov** (**RICH**) detectors are part of the LHCb apparatus and use **Cherenkov radiation** hits on their photodetectors to perform particle identification (PID). The radius of each ring depends on the momentum of the particle and on its mass, allowing to classify the particle identity using a combination of the information from the tracker and the RICH. Due to the large number of particles and Cherenkov photons, extracting this information is extremely complicated. Nonetheless, despite the background noise caused by the other overlapping rings, sophisticated reconstruction algorithms enable PID with high precision.

This thesis explores the potential use of RICH detectors as **tracking devices**, by locating the centres of the Cherenkov rings which correspond to the particles position in the detector. Because of the complexity of the ring patterns observed on the photodetectors, traditional computational algorithms are insufficient. Therefore this work investigates new promising **machine learning** approaches.

1.1 The Cherenkov Effect

A charged particle traversing a dielectric medium can emit electromagnetic radiation if its velocity exceeds the phase velocity of light in that medium. This phenomenon is known as **Cherenkov radiation** [15]. It arises when the charged particle excites the molecules of the medium, inducing in them a transient electric dipole moment. These dipoles act as impulsive sources of electromagnetic radiation, so that every point in the particle trajectory can be considered as a source of spherical waves. According to the Huygens' principle, the envelope of these wavefronts form a common conical wavefront.



(a) (b)

Figure 1.1: *On the left, the Cherenkov Effect and its envelope with spherical fronts. On the right, the geometrical construction of the effect. The image on the left is reprinted from Jackson's Classical Electrodynamics [10], while the one on the right from Fisica, Volume II (Mazzoldi, Nigro, Voci) [15].*

Consider the situation shown in Figure 1.1(b) and the two distinct points S_1 and S_2 along the particle trajectory: the corresponding points on the conical envelope P and Q must be in phase; therefore, also the point $S_1^{'}$ must be in phase with S_2 since they are equidistant from P and Q, respectively. This phase condition holds only if the time taken by the light to travel in the medium from S_1 to $S_1^{'}$ is equal to the time taken by the particle from S_1 to S_2. This can be expressed as

$$\frac{\overline{S_1S_2}}{v_p} = \frac{\overline{S_1S_2}\cos\theta_c}{v}$$

where v_p is the particle velocity, v the velocity of light in the medium and θ_c the Cherenkov emission angle. Rearranging this relation gives:

$$\cos\theta_c = \frac{v}{v_p}$$

Since $v = c/n$, where n is the refractive index of the medium, and $\beta = v_p/c$, the Cherenkov relation can be written as:

$$\cos\theta_c = \frac{c}{nv_p} = \frac{1}{n\beta} \tag{1.1}$$

It is clear that the effect occurs only if $v_p > v$. Expressing the particle velocity in terms of the momentum p and its mass m the relation becomes:

$$v_p = \frac{c^2|\mathbf{p}|}{\sqrt{m^2c^4 + \mathbf{p}^2}} > \frac{c}{n}$$

Solving this inequality leads to a threshold condition for the momentum:

$$|\mathbf{p}| > \frac{mc^2}{\sqrt{c^2n^2 - 1}}$$

Below this threshold, the particle cannot emit Cherenkov radiation. Moreover, as the particle velocity approaches the speed of light in vacuum ($v_p \to c$), the value of β tends to 1, and the Cherenkov angle asymptotically reaches its maximum value:

$$\theta_c^{max} = \arccos\left(\frac{1}{n}\right)$$

This maximum angle is the same for all particle types, as it is independent of the particle mass. The spectrum of Cherenkov radiation extends over a wide wavelength range, with shorter wavelengths contributing more significantly to the emission. In fact, the number of photons N emitted per unit path length dx and per unit wavelength interval $d\lambda$ is described by the Frank-Tamm formula [11]:

$$\frac{d^2N}{dx\,d\lambda} \propto \frac{1}{\lambda^2}\left(1 - \frac{1}{\beta^2[n(\lambda)]^2}\right) = \frac{\sin^2[\theta_c(\lambda)]}{\lambda^2} \tag{1.2}$$

As a result, the emission is stronger in the ultraviolet region, although in practice it is limited by the absorption properties of the medium and the dispersion of its refractive index. The number of emitted photons is relatively low, which can be estimated by integrating the Frank–Tamm formula over the relevant wavelength range. Consequently, the low intensity of the Cherenkov light necessitates the use of highly sensitive photodetectors for its detection.

1.2 LHCb experiment and the RICH detectors

The **LHCb** (**Large Hadron Collider beauty**) [5] detector features a unique **single-arm forward geometry**, setting it apart from the other LHC experiments. This geometry is optimized for studying the interactions of beauty and charm hadrons, which are primarly produced at small angles with respect to the beam axis. A key research focus at LHCb is **CP violation**,

one of the Sakharov Conditions [20] necessary to explain the **barion asymmetry** in the universe. The current values of CP violation predicted in the Standard Model are not sufficient to account the asymmetry, therefore LHCb searches new possible deviations from the Standard Model which can be found particularly in B and D mesons decays.

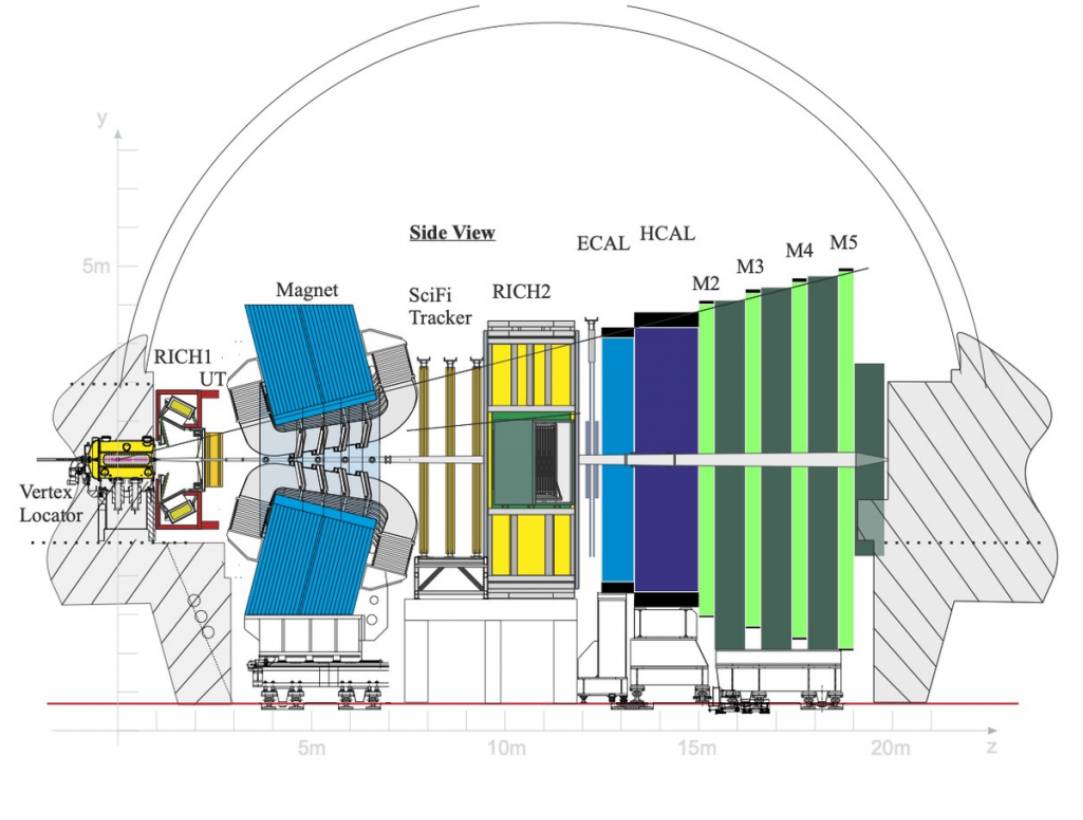


Figure 1.2: *The LHCb detector and its subdetectors in Run* 3. *Reprinted from [13].*

An illustration of the detector layout is shown in Figure 1.2. The overall LHCb structure covers a polar angular range from 10 to 250 mrad and is composed of several sub-detectors, each designed to perform a specific function within the experiment.

The proton beams collide inside the **Vertex Locator** (VELO). These collisions produce a large number of particles, some of which contain beauty and charm quarks that decay rapidly. The VELO is responsible for precisely locating the points where the proton–proton interactions occur, known as primary vertices, as well as the locations where unstable particles decay, known as secondary vertices. These points are fundamental for the tracking system, responsible for the measuring of charged particles tracks and their momentum. The tracking system includes

also the **Upstream Tracker** (UT) and the **Scintillating Fibre tracker** (SciFi tracker), positioned before and after the **Dipole Magnet**, respectively. The magnet introduces a deflection in the path of charged particles, allowing their momentum to be determined from the curvature of their tracks. The RICH system comprises two detectors, **RICH1** and **RICH2**, positioned upstream and downstream the magnet. These two detectors perform particle identification by exploiting Cherenkov radiation and are optimised to cover complementary momentum ranges. Particle energy is measured using two calorimeters: the **ECAL**, optimised for electromagnetic showers from electrons and photons, and the **HCAL**, which is sensitive to hadronic showers. Finally, the **Muon Stations** (M2–M5) play a key role in the identification of muons, which are frequently produced in the decays of B and D mesons and are crucial for their reconstruction.

1.2.1 The RICH detectors

The RICH system consists of two detectors, **RICH1** and **RICH2**, whose primary purpose is to distinguish between pions (π), kaons (K), and protons (p). Together, they provide efficient particle identification over a wide momentum range, from approximately 2.5 GeV/c (for pion–kaon separation) up to 100 GeV/c. In particular, the identification of pions is of great interest for LHCb physics, as they constitute the dominant decay products of beauty and charm hadrons.

RICH1 is placed closed to the VELO and is optimised for the identification of particles at low momentum in the range 2.5 GeV/c – 50 GeV/c. The detector uses fluorocarbon gas C_4F_{10} as a Cherenkov radiator, which has a refractive index of $n = 1.0014$ at a wavelength of $\lambda = 400$ nm. The detector covers an angular acceptance from 25 mrad to 300 mrad. Conversely, RICH2 is located downstream the SciFi tracker and contains CF_4 gas with a refractive index of $n = 1.0005$ at a wavelength of $\lambda = 400$ nm. Also, to suppress scintillation, 5% of CO_2 is added to the mixture. This detector covers the momentum range 15 GeV/c – 100 GeV/c and has an angular acceptance of 15 mrad – 120 mrad [23] [4].

RICH1 is divided into two vertical halves, each functioning as an independent detector. Each part has a configuration which includes a system of mirrors designed to direct Cherenkov photons onto the photodetectors, as illustrated in Figure 1.3. First, a spherical mirror collects the photons emitted in the radiator and reflects them onto a planar mirror. The light is then

redirected through a quartz window onto the focal plane, where an array of Multi-Anode Photomultiplier Tubes (MaPMTs) is positioned. These MaPMTs detect the Cherenkov photons and return the characteristic ring-shaped light patterns used for subsequent particle identification. In particular, the central region exhibits a higher density of overlapping Cherenkov rings, since it is closer to the beam line where most particles originate. The area covered by the planes is about 1200x1200 mm^2. The entire detector is enclosed in a magnetic shield to prevent spurious particles from reaching the detection area, which are originated from the interaction point in the VELO.

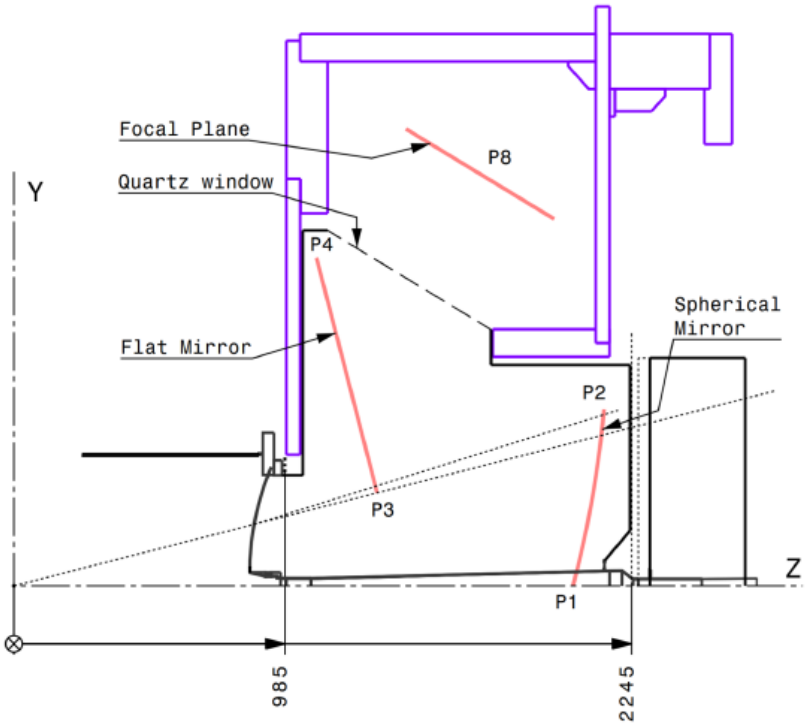


Figure 1.3: *RICH1 upper structure. Reprinted from [4]*

RICH2 has a similar optical geometry to RICH1 but is composed of two distinct horizontal halves, each protected by a magnetic shield that is less intense than the one used in RICH1. In each half, a spherical mirror collects and collimates the Cherenkov photons onto a flat mirror, which then redirects the light through a quartz window onto the MaPMTs.

1.2.2 The trigger system

The LHCb experiment features a cutting-edge trigger system [1] designed to reduce the enormous amount of data coming from the proton-proton collisions into a manageable volume. In particular, the pp collisions generate approximately 4 TB/s and the trigger system returns about 10 GB/s, which can be stored for further offline analysis. During Run 1 and Run 2, LHCb employed a hardware trigger, known as L0, in combination with two software-based High Level Triggers (HLT1 and HLT2). The hardware trigger used fast electronic signals from subdetectors, such as the calorimeters, to select interesting events within a few milliseconds, filtering out most background events. This approach had the disadvantage of potentially discarding rare or unexpected events of interest. In contrast, the software trigger performs a full event reconstruction using GPUs and CPUs, enabling a much more accurate and flexible selection. Starting from Run 3, the hardware trigger was completely removed, and the two software triggers now handle the entire data acquisition and event selection process in real-time, as shown in Figure 1.4.

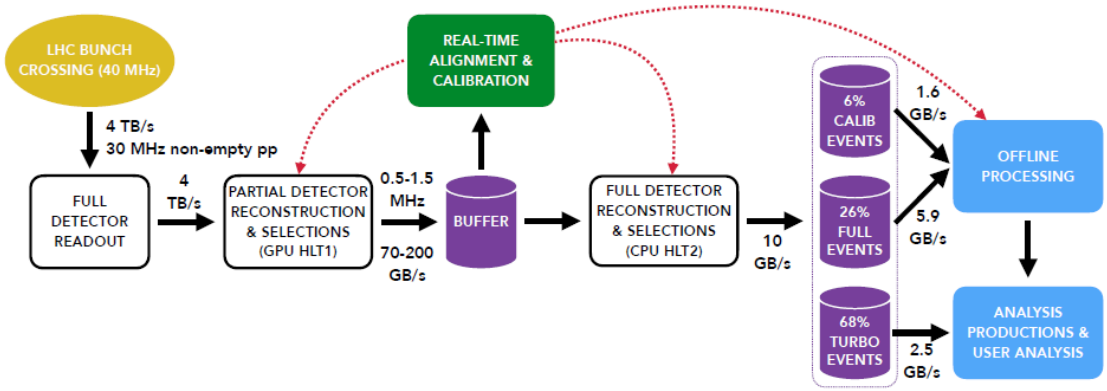


Figure 1.4: *LHCb trigger system: data coming from pp collisions are processed and selected by HLT1 and HLT2. Reprinted from [1].*

The **HLT1** is the first stage in the trigger system and is entirely handled by GPUs. It reconstructs the VELO primary and secondary vertices and the total tracks, thanks to UT and SciFi trackers, regardless of whether they are displaced from the primary vertices or not. The momentum of the particles is measured with a precision at the percent level. Before the reconstruction step, a Global Event Cut (GEC) is applied to discard a fraction of events containing

an excessive number of tracks, in order to reduce the computational load. HLT1 is also able to distinguish whether a particle is a muon or not. Then the data are selected based on these four categories: events relevant to the LHCb physics program; calibration information essential for further reconstruction[1]; events with physics signatures not covered by the first category; technical samples used for tasks such as luminosity determination and system monitoring. The HLT1 stage receives data from pp collisions at an input rate of 30 MHz and then the selected events are written in a buffer at an average rate of 1 MHz.

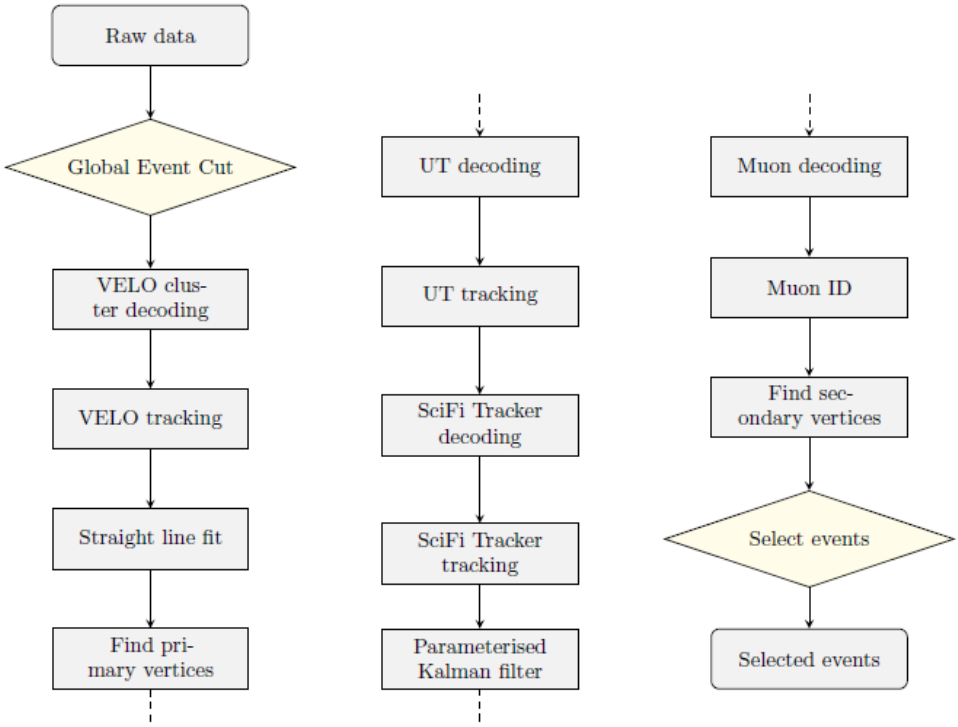


Figure 1.5: *HLT1 pipeline used in Run 3 for track reconstruction and muon identification. Reprinted from [1].*

The **HLT2** is the second stage and runs on CPUs. It performs a full, offline-quality reconstruction of the events with the precision required for detailed physics analysis. HLT2 uses calibration and alignment information from HTL1 to improve tracks reconstruction and particles identification, leveraging all subdetector measurements. Finally, it selects the most interesting

[1]For example, variations in pressure and temperature of the gas inside the RICH detectors affect its refractive index, and consequently alter the expected Cherenkov angle. Continuous monitoring of these parameters is essential to reduce systematic uncertainties in particle identification.

events using approximately 1000 dedicated selection algorithms.

The thesis proposes the use of RICH detectors as tracking devices by locating Cherenkov ring centers in the first stage of the trigger system, with the aim to improve the quality of the selected data. This task is particularly challenging due to the large number of overlapping Cherenkov rings, especially concentrated at the center of the RICH detectors. To address this, machine learning techniques are required, which are well suitable since HLT1 has a highly parallel architecture and runs entirely on GPUs. By providing more precise track information, this approach could also reduce the computational load on CPUs during HLT2, enhancing the efficiency and accuracy of the event reconstruction performed by downstream algorithms.

1.3 Machine Learning

Machine Learning is a class of computational tools that extract meaningful information from large datasets. Unlike traditional computer algorithms, machine learning can detect and extract patterns from complex data without the need for explicit, detailed instructions from a human programmer on how to perform the task [21]. This capability is particularly relevant for identifying Cherenkov ring centers, as it enables the recognition of subtle spatial features and correlations. As a result, the problem naturally lends itself to machine learning.
A first distinction in machine learning is between supervised and unsupervised learning.
Supervised learning is a machine learning paradigm in which an algorithm learns to associate unseen input data to specific outputs by relying on examples of input-output correlations. This process is made possible through prior training on labeled data, where each input example is paired with its corresponding correct output, allowing the model to generalize on unseen data. Example tasks are classification, regression, segmentation.
Unsupervised learning is a paradigm where, unlike supervised learning, the model learns patterns from unlabeled data and attempts to discover hidden patterns and structures within the dataset, by analyzing the inherent properties and relationships of the data.

1.3.1 Deep Learning and Artificial Neural Networks

Among machine learning models, **Artificial Neural Networks** (ANNs) [9] are a powerful class used in both supervised and unsupervised learning, inspired from the structure of the human brain. They form the foundation of **Deep Learning**, a subfield characterized by multilayered architectures of neurons capable of learning complex features. Each neuron holds real-valued inputs, processes them and communicates the output to other neurons via weighted connections. More specifically, when a neuron receives inputs x_i from the previous i-th neurons, it applies a summation function Σ as follows:

$$z_j = \Sigma(x_1, ...x_n) = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

where w_{ji} is the connection weight between the j-th neuron and i-th neuron, b_j is the bias term associated to the j-th neuron and z_j is the current value of the neuron, known as logit. The weights and the biases are the network's parameters, as they define the model behaviour and output. An activation function f, such as ReLU or sigmoid, is then applied to the logit to produce the neuron's output a_j:

$$a_j = f(z_j)$$

In general, a neural network consists of different layers of interconnected neurons:

1. Input layer: each neuron receives a part of the raw data and forwards it to the next layer;

2. Hidden layers: intermediate layers that communicate with each other and extract increasingly abstract features;

3. Output layer: produces the final prediction, often as a probability distribution over the possible outputs.

Artificial Neural Networks are mostly employed within the supervised learning paradigm. Training consists in tuning the network's parameters by minimizing a **loss function** $\mathscr{L}$, which quantifies the discrepancy between the network's predicted output and the corresponding labeled target. The choice of loss function is crucial and depends on the network's purpose. The parameters are updated through an optimization process known as **backpropagation**, in which

the loss gradient is computed with respect to each parameter. These gradients are then used to iteratively adjust the weights and biases, thereby reducing the loss and progressively improving the network's performance. The backpropagation is guided by an optimization function, such as the Adam (Adaptive Moment Estimation) optimizer.

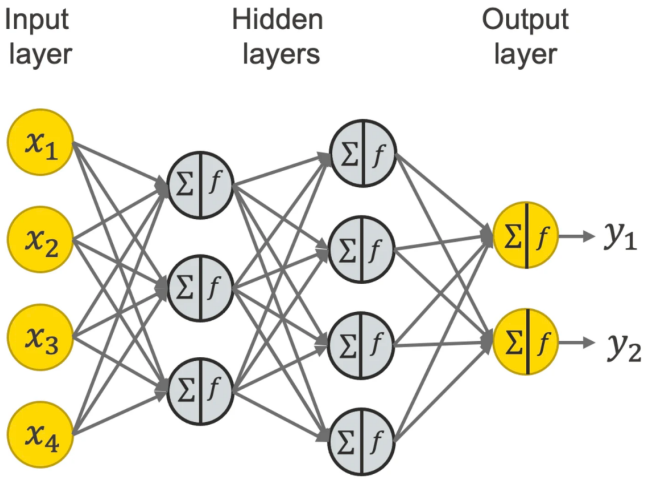


Figure 1.6: *Schematic view of the structure of an Artificial Neural Network: the input values x_i provided to the input layer are forwarded to the hidden layers, where each neuron applies the summation function Σ followed by the activation function f. The final results y_i are given in the output layer as probabilities.*

The loss function is computed both during training and during validation, in which the network is tested on unseen labeled data to quantify its performance. In general, the lower the validation loss, the better the network's performance. It is important to note that the loss function used for training can differ from the one used for validation.
The loss functions most commonly used in this work are L1, L2 and Cross Entropy:

- **L1 Loss** (Mean Absolute Error): for a general vector $\mathbf{y} \in \mathbb{R}^n$ of ground truth values and predictions $\hat{\mathbf{y}} \in \mathbb{R}^n$, the L1 loss is defined as

$$\mathscr{L}_1(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

  In the case of points in a 2D plane, where $P = (x_P, y_P)$ is the prediction and $T = (x_T, y_T)$ the ground truth, the L1 loss becomes

$$\mathscr{L}_1(P,T) = |x_P - x_T| + |y_P - y_T|$$

- **L2 Loss** (Mean Squared Error): for a given vector $\mathbf{y} \in \mathbb{R}^n$ and predictions $\hat{\mathbf{y}} \in \mathbb{R}^n$, the L2 loss is defined as

$$\mathscr{L}_2(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

  For points in a 2D plane $P = (x_P, y_P)$ and $T = (x_T, y_T)$, the L2 loss is

$$\mathscr{L}_2(P, T) = (x_P - x_T)^2 + (y_P - y_T)^2$$

- **Cross Entropy Loss**: given a discrete probability distribution over C classes with predicted probabilities $\mathbf{p} = (p_1, \ldots, p_C)$ and ground truth labels $\mathbf{y} = (y_1, \ldots, y_C)$, the cross entropy loss measures the dissimilarity between the predicted distribution and the true distribution as follows:

$$\mathscr{L}_{\text{CE}}(\mathbf{p}, \mathbf{y}) = -\sum_{c=1}^{C} y_c \log p_c$$

  The cross entropy penalizes predictions that assign low probability to the true class, encouraging the model to output high confidence for correct labels.

  In the special case of two classes, the cross entropy simplifies to the Binary Cross Entropy (BCE):

$$\mathscr{L}_{\text{BCE}}(p, y) = -[y \log p + (1 - y) \log(1 - p)]$$

  where $p \in [0, 1]$ is the predicted probability of the positive class and $y \in \{0, 1\}$ is the ground truth label, with $y = 1$ indicating the positive class and $y = 0$ the negative class.

The training process is controlled through the **hyperparameters**, which are values chosen before training and directly influence how the model learns. Among them, we note:

- **Learning rate**: the step size with which the optimizer updates the parameters at each iteration;
- **Batch size**: the number of samples taken from the dataset before performing a parameters update;
- **Number of Epochs**: the number of times the entire dataset is processed during training.

Efficient training depends also on an appropriate choice of the optimizer, its hyperparameters, and the activation functions used in the network.

2. Computer Vision for RICH tracking

RICH detectors perform particle identification with high efficiency and precision in the second trigger stage HLT2. This is possible thanks to the track reconstructions and momentum measurements carried out in HLT1, which enable the determination of particle masses through the analysis of the extracted Cherenkov angles. However, RICH detectors could also exploit the Cherenkov rings pattern to locate **ring centers** already in the first trigger stage, providing additional **tracking points** corresponding to the particle positions in the gas radiator. This additional tracking information could improve particle identification and event selection, while reducing the computational load in both HLT1 and HLT2.

Computer vision, a branch of deep learning focused on enabling machines to interpret and analyze visual data, is well suited for this task, since the electric signals from the MaPMTs can be rasterized into images with dimensions corresponding to the size and sensitivity of the detector. Computer vision techniques have broad applications across technology and IoT, addressing various tasks such as classification, segmentation, object detection and keypoint detection. In this work, we will focus on keypoint detection.

2.1 Keypoint Detection

Keypoint detection is a computer vision task which consist in the identification and localization of points of interest and has applications in human pose estimation, facial recognition, object tracking and medical imaging [8]. In the context of the LHCb experiment, keypoint detection could play a crucial role in identifying the centers of Cherenkov rings produced by charged particles in RICH detectors. Indeed, it can directly find the centers avoiding the segmentation of each ring, thus reducing inference time and computational load compared to segmentation and classification approaches. The field of keypoint detection was revolutionized by deep learning with the introduction of new algorithms primarily based on **Convolutional Neural Networks** (CNNs). CNNs feature a hierarchical and layered architecture, which makes them widely used since they perform well on heterogeneous and noisy data. They are able to extract features from

training data that enable them to recognise complex patterns and generalise to unseen samples. In terms of problem formulation, keypoint detection is often approached in two main ways:

1. The first approach involves direct **regression** of the keypoint coordinates, where the network predicts the exact coordinates of each point. Regression is computationally efficient and suitable for real time applications, but it can struggle with ambiguous or noisy inputs. This method has been employed in this thesis with the YOLO-Pose model [25].

2. The second approach generates **heatmaps** for each keypoint, producing a probability distribution over the image that indicates the likely positions of the points. The coordinates of the keypoints are then obtained as the peak locations of the heatmap. This approach can preserve spatial information, leading to more robust localization, especially when dealing with overlapping keypoints. In this work, heatmap based detection is implemented using the UNet architecture [19].

YOLO and UNet backbone architectures are CNN based and exemplify how they can be adapted to different contexts.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a standard model in deep learning and computer vision, since they are highly effective in solving complex pattern recognition task on images. Their success relies on exploiting a hierarchical feature extraction enabled by three fundamental building blocks: convolutional layers, pooling layers and fully-connected layers [16].
The **convolutional layer** is the core component of a CNN, in which each neuron is connected not to the entire input, but only to a small spatially localized region, called the receptive field. This local connectivity allows the network to efficiently exploit spatial correlations in the data. Mathematically, the operation is defined as a discrete convolution between the input feature map and a set of filters (or kernels). Each filter slides across the spatial dimensions of the input and computes a dot product between the pixel values and its weights, followed by the addition of a bias term. The result is a new feature map, which highlights the presence of certain local patterns, like edges or textures, in different parts of the image. Both the weights and the biases are learnable parameters, optimized during training. An important property of this architecture

is weight sharing: the same filter is applied across all spatial locations and reduces the number of parameters. After the convolution, an activation function is applied to facilitate the learning. The **pooling layers** serve the purpose of downsampling the feature maps produced by the convolutional layers. This reduction in spatial resolution decreases the computational cost in subsequent layers and makes the network less sensitive to small shifts or distortions in the input. The most common strategy is max pooling, where the maximum value in a local region, like a 2×2 window, is selected. The pooling operation ensures that the most relevant features are retained while discarding redundant information.

The **fully-connected layers** have neurons which are highly connected to all units in the previous layer, enabling the network to integrate the spatially distributed features into a global representation of the input. This stage is typically used to produce the final output, such as classification probabilities.

CNN based models are effective because they focus on local patterns in the images while successive layers progressively capture more abstract patterns. Also weight sharing is important, as it minimizes the number of parameters and improves generalization on unseen data.

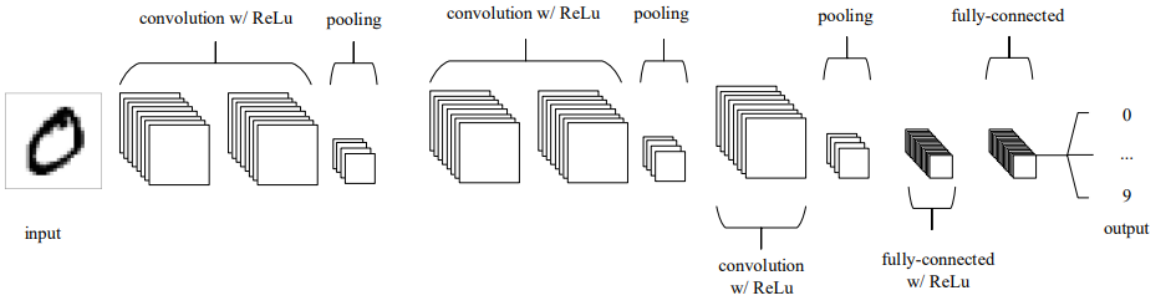


Figure 2.1: *A common CNN architecture, with sequences of convolutional and pooling layers, followed by fully-connected layers. Reprinted from [16].*

2.1.2 Metrics

Quantifying the efficiency and accuracy of a keypoint detection model requires the adoption of **validation metrics** tailored to the specific problem under consideration. The quality of a model can be evaluated by checking whether predicted keypoints fall within a defined distance threshold from the ground truth. The most straightforward choice is the **Euclidean Threshold**.

A predicted point P with coordinates (x_P, y_P) is considered correct if its distance from a ground truth keypoint T with coordinates (x_T, y_T) is smaller than d:

$$\overline{PT} = \sqrt{(x_P - x_T)^2 + (y_P - y_T)^2} < d \tag{2.1}$$

Alternative thresholds, such as Object Keypoint Similarity (OKS) or Intersection over Union (IoU), are often adopted in YOLO models to enhance human pose evaluation. However, despite its simplicity, the Euclidean Threshold proves to be the most suitable choice for our purposes. The thresholds are used to classify each predicted point as a true positive, false positive or false negative, which are then used to compute the evaluation metrics.

- **True Positive** (**TP**): the predicted point is a correct prediction under the chosen threshold. For example, setting the Euclidean Threshold at 4 px, a point whose distance from the ground truth is 2 px is a true positive.

- **False Positive** (**FP**): the predicted point is considered incorrect because it is farther than the chosen threshold from any ground truth point. For example, using the same 4 px threshold, if the closest ground truth point is 7 px away, the prediction is a false positive. The number of false positives can be computed as the total number of predicted points minus the true positives.

- **False Negative** (**FN**): a ground truth point that has not been matched by any predicted point within the chosen threshold. For example, if no prediction lies within 4 px from a ground truth point, that ground truth is counted as a false negative. The number of false negatives can be calculated as the difference between the total of ground truth points and the true positives.

In some cases, multiple predicted points may fall within the threshold of the same ground truth keypoint, which could lead to counting both predictions as true positives. To avoid this, the Hungarian algorithm is used to perform an optimal one by one assignment between predictions and ground truth. First, the distance matrix between every predicted and ground truth keypoint is computed. Distances larger than the threshold are penalized with a very high cost, preventing those pairs from being matched. The Hungarian algorithm is then applied to find the assignment

that minimizes the total cost while ensuring that each predicted point and each ground truth point are matched at most once. Pairs within the threshold are counted as true positives.
We can now introduce the metrics used to validate and compare the models presented in this thesis [17]:

- **Precision** (**P**): it measures the proportion of predicted point which are correct under the chosen threshold and can be computed as

$$P = \frac{TP}{TP + FP} \tag{2.2}$$

  A high precision means that the model makes few false detections, but it does not guarantee that all ground truth points are found.

- **Recall** (**R**): it is the ratio of ground truth points correctly detected within the threshold, specifically:

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

  High recall means that the model is able to detect most of the ground truth points.

- **F1 Score**: it is the harmonic mean between precision and recall, useful when both false positives and false negatives are equally important. In particular, F1 is used to make an overall model evaluation. It is defined as:

$$\mathrm{F1} = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.4}$$

  An F1 score of 1 indicates perfect precision and recall, while 0 indicates a completely incorrect model.

Another important evaluation metric is the **mean inference time** of the model. The model runs in the HLT1 trigger stage, which receives collision data from the detector at an input rate of 30 MHz. After the Global Event Cut, this rate is reduced and its processing time corresponds, as an order of magnitude, to about $0.1\,\mu$s per event. Achieving such a short inference time is extremely challenging, if not practically impossible, for a deep learning model. This constraint could be alleviated by designing the model to be as fast as possible and by deploying it on a dedicated set of GPUs for RICH tracking, operating in parallel. For this reason, inference time

has to be considered as an evaluation metric, trying to keep it as low as possible. Edge machine learning techniques, like quantization, should also be considered in further improvements to minimize the inference time and also the computational load on GPUs.

2.2 The dataset

Supervised machine learning models require large labeled dataset for effective training. In RICH detectors, labeled data can only be obtained through full **Monte Carlo simulations** of pp collisions and LHCb detector response. However, this approach is not considered in this thesis due to time constraint and computational costs. Instead, the dataset generation is based on a customized method developed by Giovanni Laganà [7], to whom I express my deepest gratitude for sharing the generation codes. This method replicates the key physical processes in pp collisions and RICH1 detector by exploiting probabilistic distributions derived from full Monte Carlo simulations via Kernel Density Estimation (KDE). The generator samples values from these distributions and applies the physical principles of the Cherenkov effect to reproduce the typical features of the RICH1 detector. For the purposes of this thesis, we focus exclusively on RICH1 due to time constraints, although a similar approach to dataset generation and keypoint detection models could also be applied to RICH2. Now, let us briefly review the method used to generate the dataset. For a more detailed explanation, please see [7].

First, a particle type is selected according to defined probabilities computed from a full Monte Carlo

$$P(p_i) = \frac{N_{p_i}}{N_{tot}}$$

where N_{p_i} is the number of the i-th particle in the full Monte Carlo sample and N_{tot} the total number of particles. The possible particle types are all charged particles with a lifetime long enough to reach the RICH detectors, namely electron, muon, pion, kaon and proton. For the selected particle, a momentum value is then sampled from a known distribution obtained via KDE on the full Monte Carlo dataset, as shown in Figure 2.2. Using equation 1.1, the Cherenkov angle θ_c is computed. Then, the ring radius R on the MaPMTs plane is obtained through the proportionality

$$R = \frac{\tan(\theta_c) \cdot R_{max}}{\tan(\theta_c^{max})}$$

where $R_{max} = 100\,\text{mm}$ is the maximum detected angle in RICH1 and θ_c^{max} is the maximum Cherenkov angle. From full Monte Carlo simulation, we also extract the parameter N, defined as the average number of photon hits for particles with $\beta \to 1$. This value is adjusted to account for the particles which have $\beta < 1$ according to the Frank-Tamm formula (equation 1.2) as

$$\tilde{N} = N \cdot \frac{\sin^2(\theta_c)}{\sin^2(\theta_c^{max})}$$

The number of photon hits on the MaPMTs plane is then sampled from a Poisson distribution with mean $\tilde{N}$, as Cherenkov photon emission is a stochastic process. After that, the ring center on the MaPMTs plane is sampled from a distribution obtained via KDE from full Monte Carlo data (see Figure 2.3). The ring is generated by sampling photon polar positions from a uniform distribution over $[0, 2\pi)$.

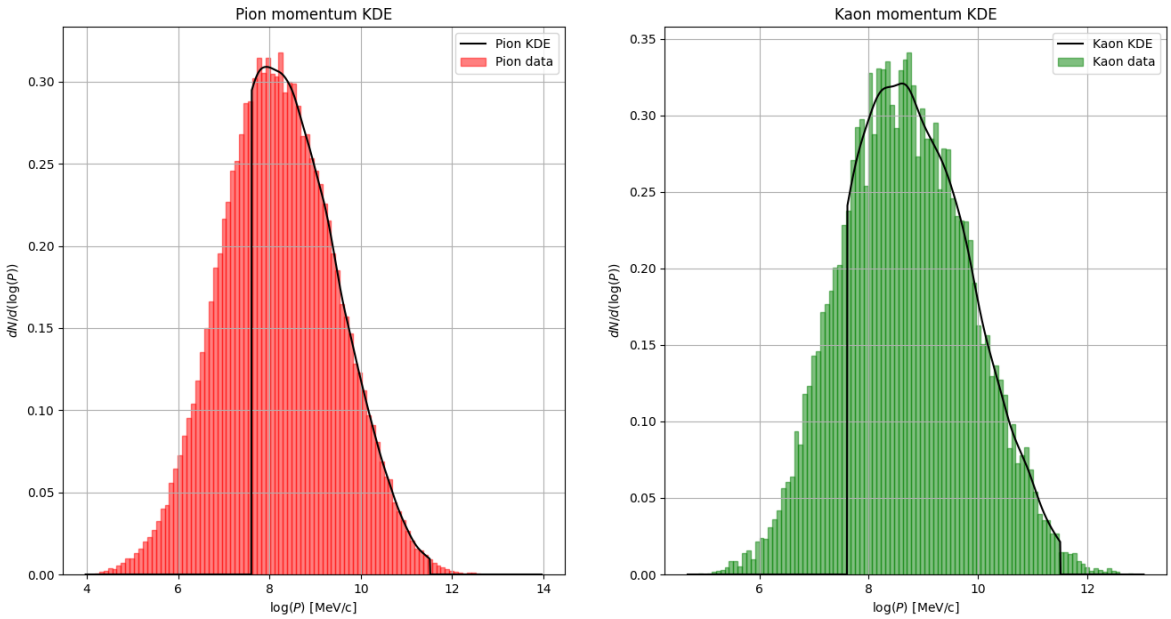


Figure 2.2: *Kernel Density Estimations on full Monte Carlo samples of pion and kaon momenta. The KDEs show a lower and upper cut, which correspond to RICH1 momentum acceptance.*

RICH1 rings are affected by radial noise originating from three sources [3]:

1. **Chromatic error**: due to the spectrum of Cherenkov radiation and the response of the photodetectors, photons have different wavelengths, leading to changes in the refractive index. This results in a chromatic error of 0.59 mrad.

2. **Optical error**: caused by variations in the photon emission point within the gas radiator and aberrations from the mirrors, leading to 0.38 mrad.

3. **Geometrical error**: because of the finite size of the MaPMTs sensors (≈ 2.88 mm [2]), corresponding to

$$\frac{2.88\,\text{mm}}{\sqrt{12}} \simeq 0.83\,\text{mm} \simeq 0.46\,\text{mrad}$$

The overall radial error, obtained from the quadrature sum of these contributions, is

$$\sigma_c = 0.82\,\text{mrad} \simeq 1.5\,\text{mm}$$

Radial noise from these apparatus effects is implemented by adding to each photon position a Gaussian random value with zero mean and standard deviation σ_c (in mm).

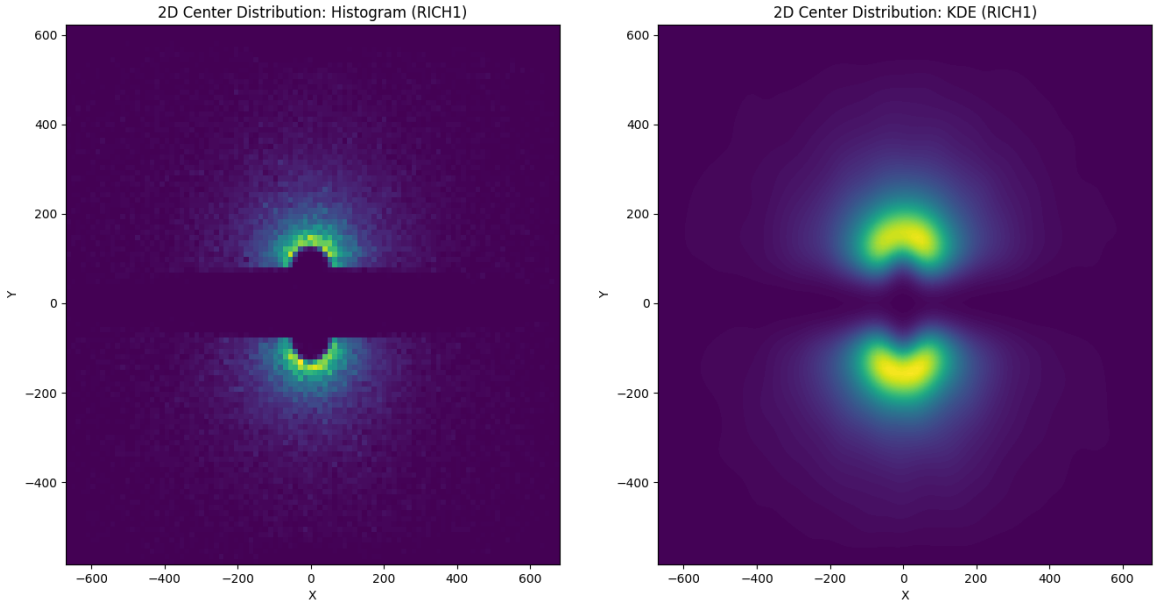


Figure 2.3: *Kernel Density Estimation on the full Monte Carlo sample of ring centres, combining the two MaPMT planes (upper and lower) of RICH1. As expected, a high density of centres is observed in the central region of the image.*

The dataset generator creates labeled and controllable events, as the generator parameters are known. An important setting is the choice of the distributions, which in our case are given by the KDEs from full Monte Carlo simulations, but could in principle be modified to allow for a more targeted study of the model. A key parameter is also the number of rings generated per

event: based again on full Monte Carlo simulations, the mean number is about 160 rings per event for LHC Run 3. The generator returns the centers of the rings along with other labels, such as the radii. A key difference from the generator developed by Giovanni Laganà is that, in his implementation, the centers are randomly selected to simulate the tracking system in HLT1. These centers are then used to isolate a region of the image containing the corresponding particle ring, from which the ring radius can be inferred and subsequently used for PID. In contrast, our generator returns all produced centers. However, centers without associated rings are not useful in our case, as they correspond to particles that do not produce Cherenkov radiation. Therefore, these centers are discarded and replaced by particles with a randomly sampled momentum that ensures the emission of Cherenkov light.

Another important aspect is the rasterization of the event into an image suitable for analysis by the keypoint detection model. The images are in gray format, with pixel values in the range $[0, 255]$. A pixel value of 0 corresponds to the absence of photons, while non-zero pixel values represent signals generated by one or more photons. Two different image sizes are considered, each motivated by a distinct requirement. In the hypothesis that detector pixels are evenly spaced, if each image pixel is intended to correspond one-to-one with the MaPMTs, whose physical size is 2.88 mm, the resulting image size is approximately:

$$\frac{1200\,\text{mm}}{2.88\,\text{mm}} \simeq 416\,\text{px}$$

Alternatively, if the MaPMT positions are not evenly spaced, the rasterized image is driven by the rings' radial error σ_c. Then, the number of pixels is determined by the ratio between the MaPMT plane length and the radial noise:

$$\frac{1200\,\text{mm}}{1.5\,\text{mm}} = 800\,\text{px}$$

Different image sizes lead to diverse inference speed as well accuracy and precision of the predicted keypoints. Therefore, both image formats are characterized using YOLO and UNet models. A rasterized event example is shown in Figure 2.4.

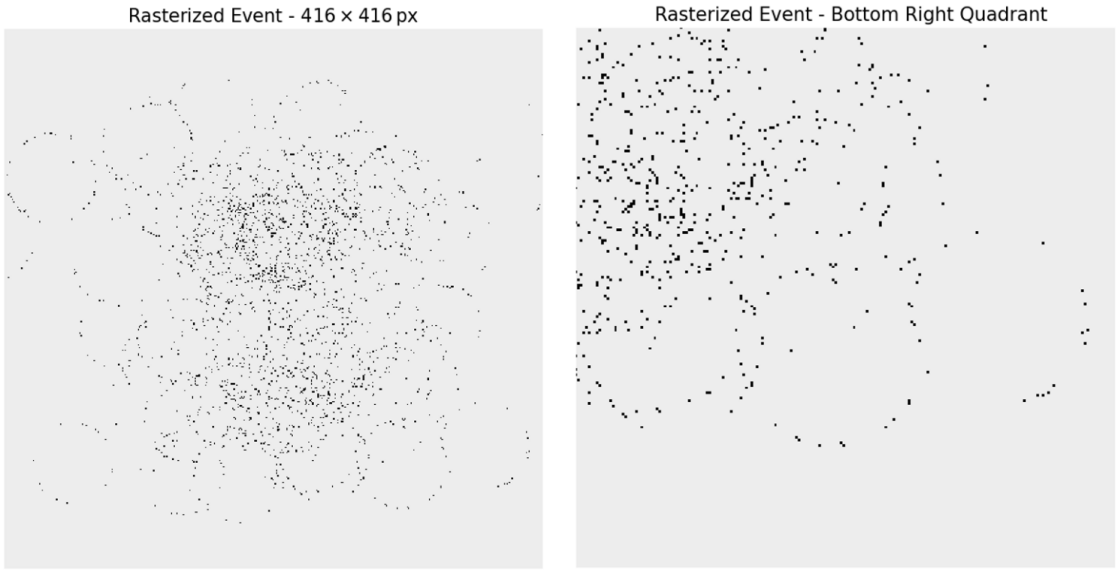


Figure 2.4: *Example of rasterized image (*416×416*px) with* 150 *rings. On the left the full image, while on the right a zoom on the bottom right quadrant. Colors are inverted for better visualization: actual background is black and photon hits are white.*

3. YOLO model

You Only Look Once (**YOLO**) is an object detection model first introduced by Redmon et al. in 2015 [14]. The YOLO algorithm has a unique approach to computer vision since the detection problem is reformulated as a single **regression** task over the entire image. In fact, unlike other CNN like UNet (see Chapter 4), YOLO's methodology diverges from common two-stage detection, employing a single neural network to predict bounding boxes, class probabilities and keypoint coordinates in the entire image once. For this reason, YOLO model achieves a remarkable inference speed, making it a suitable model for real-time applications.

YOLOv11 is the latest YOLO model released by Ultralytics [25] and extends its capabilities to multiple computer vision tasks, such as object detection, instance segmentation and **pose estimation**. Pose estimation is a keypoint detection model focused on detecting and classifying human body parts, but it can be adapted to different contexts through supervised training on custom datasets. Since inference speed is among our first concerns, YOLOv11 is suitable for our purposes and can be trained on RICH1 Cherenkov synthetic dataset.

3.1 YOLOv11-Pose

The YOLOv11 keypoint detection model is called **YOLOv11-Pose**. Its architecture adapts the YOLO framework for pose detection, which is primarily designed to detect points in images to track human poses, movements and also to classify body parts. The general YOLO framework, shared across all YOLOv11 models, consist of three parts: backbone, neck and head [14]. These parts are preceded by an **input pipeline**, which resizes images to a predefined dimension and normalizes pixel values to the $[0, 1]$ interval.

The **backbone** serves as a primary feature extractor, using convolutional layers and specialized blocks to generate feature maps at multiple resolutions. The convolutional layers, known as CBS (Conv-BN-SiLU), downsample the image with a 2D convolution, followed by batch normalization and an activation function. The CBS layers are followed by C3k2 modules, which split the feature maps into two parts: one part is processed through further convolutions to

extract more features, while the other is concatenated to subsequent layers without additional processing, reducing computational load and inference time. The C3k2 employs 2×2 convolutions as kernel size. The backbone ends with a Spatial Pyramid Pooling - Fast (SPPF) block, followed by a Cross Stage Partial with Spatial Attention (C2PSA) block. The SPPF block enriches the features with multi-scale spatial information by applying a 5×5 max pooling sequentially three times, then concatenating and fusing the pooled features via a 1×1 convolution. The C2PSA integrates a positional self-attention (PSA) mechanism, which allows the network to selectively emphasize relevant regions in the image, leading to more accurate detection for smaller or partially occluded objects. For a more detailed description of the attention mechanism, please see Chapter 4.

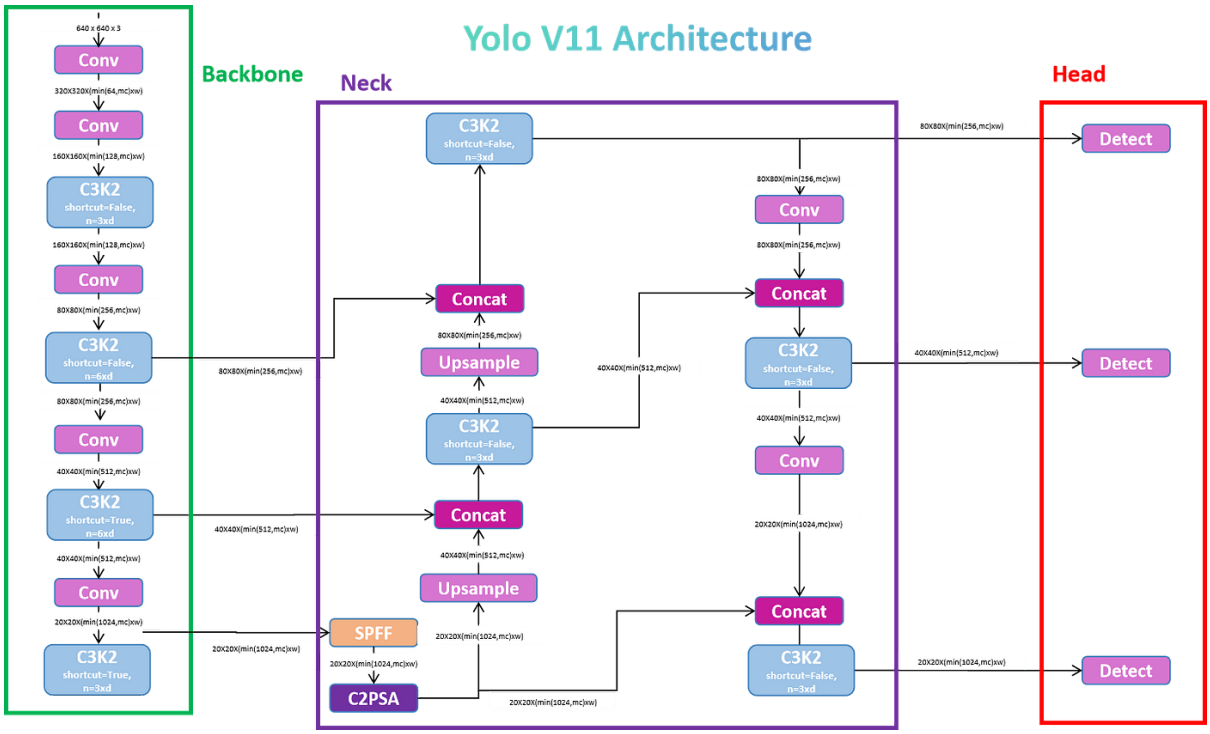


Figure 3.1: *YOLOv11 general architecture, showing the three main components: backbone, neck and head. The backbone extracts multi-scale features, the neck fuses them and the head produces the final predictions. Reprinted from [18].*

The **neck** combines features at different scales and passes them to the head. This involves upsampling, C3k2 modules and concatenation of feature maps from different levels. The output

consists of three sets of feature maps with strides[1] of 8, 16 and 32, allowing the network to capture both fine details and global context.

Finally, the **head** generates the final predictions. It processes the feature maps from the neck through C3k2 and CBS blocks, followed by convolutional layers that reduce the features to bounding box coordinates and class scores. In YOLOv11-Pose, the head also outputs keypoint positions. For the RICH application, however, only the keypoint coordinates are relevant, while bounding boxes and classification are not required.

YOLOv11-Pose is available in multiple versions: nano (n), small (s), medium (m), large (l) and extra-large (xl), which differ in model size, number of parameters and computational cost, allowing users to choose a trade off between speed and accuracy depending on the application. Due to time constraints, only the YOLOv11-Pose small model has been considered in this thesis, as it is regarded as the best compromise between sufficient accuracy and low inference time.

3.1.1 Loss Function

YOLOv11-pose model presents a complex loss function $\mathscr{L}$ given by the sum of five different losses [25] [24]:

- **Box Loss** $\mathscr{L}_{\text{box}}$: it is used to learn the localization of object bounding boxes and it is based on CIoU (Complete Intersection over Union). The IoU measures the overlap between a predicted and a ground truth bounding box, defined as

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{B_p \cap B_{gt}}{B_p \cup B_{gt}}$$

  where B_p is the predicted bounding box and B_{gt} is the ground truth bounding box. The CIoU extends IoU by adding terms that penalize the distance between the box centers and the mismatch in aspect ratio. Formally, the box loss is

$$\mathscr{L}_{\text{box}} = 1 - \text{IoU} + \frac{\rho^2(B_p, B_{gt})}{d^2} + \alpha v \tag{3.1}$$

[1] The stride of a layer in a convolutional neural network indicates the factor by which the spatial resolution of the input is reduced to produce the feature map. Smaller strides preserve higher spatial detail, while larger strides capture more global context but at lower resolution.

where $\rho(B_p, B_{gt})$ is the distance between the predicted and the ground-truth box centers, d is the diagonal that encloses both boxes and αv measures the difference between the box ratio [12]. Intuitively, if the boxes are far apart and have different shapes, the loss increases. Conversely, if the boxes coincide, the loss tends to zero.

- **Distribution Focal Loss** $\mathscr{L}_{\text{dfl}}$: this loss refines the prediction of bounding box coordinates by avoiding direct rounding to discrete pixels. Instead, each coordinate is represented as a discrete distribution over adjacent bins, so that a continuous value can be expressed as a weighted combination of its two nearest bins. The ground truth coordinate is therefore encoded into a distribution; then a Cross Entropy loss (see Section 1.3.1) is applied between this distribution and the predicted one.

- **Classification Loss** $\mathscr{L}_{\text{cls}}$: it is employed for multi-classes object detection, based on Cross Entropy. For RICH application, the class is unique since it is the ring center's one, thus the classification loss is meaningless.

- **Pose Loss** $\mathscr{L}_{\text{pose}}$: this loss teaches the model the correct positions of keypoints. It is based on the SmoothL1 function, which combines the properties of L1 and L2 losses (see Section 1.3.1).

  Given a predicted point P and a ground truth T, the **SmoothL1** function is:

$$\text{SmoothL1}(P,T) = \begin{cases} 0.5 \cdot \text{L2}(P,T) & \text{if L1} < \beta \\ \text{L1}(P,T) - 0.5 \cdot \beta & \text{otherwise} \end{cases}$$

  where β is a threshold, typically set to $\beta = 1$. SmoothL1 is useful because for small errors it behaves quadratically, allowing the model to improve with higher precision since minimal errors are penalized. For large errors, it behaves linearly, so a single large error does not dominate the total loss or destabilize the weight updates during training: L1 is more robust to outliers, whereas L2 can be unstable in their presence.

  As a result, the Pose loss is defined as:

$$\mathscr{L}_{\text{pose}} = \frac{1}{K} \sum_{k=1}^{K} v_k \, \text{SmoothL1}(P_k, T_k) \tag{3.2}$$

  where $v_k \in \{0,1\}$ is a keypoint visibility flag and K the total number of ground truth keypoints.

- **Objectness Loss** $\mathscr{L}_{\text{obj}}$: this loss measures the confidence of the model that an object is present in a given grid cell. The loss is computed over three different grids corresponding to the three strides used in the model head for prediction. For example, given an input image of size $416 \times 416\,\text{px}$ and a stride of 8, the resulting grid is 52×52 cells. Each predicted cell outputs an "objectness" score p_{obj}, which represents the probability that the cell contains an object. It is obtained from transforming the raw logit z_{obj} (an unbounded output from the network representing objectness) in a probability bounded between 0 and 1 using the sigmoid function σ:

  $$p_{obj} = \sigma(z) = \frac{1}{1+\exp(-z_{obj})}$$

  The objectness loss is then defined using the Binary Cross Entropy (BCE) between the predicted score p_{obj} and the ground truth label $y_{\text{obj}} \in \{0,1\}$:

  $$\mathscr{L}_{\text{obj}} = \frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \text{BCE}(p_{obj}^{(i)}, y_{obj}^{(i)}) \tag{3.3}$$

  where $y_{\text{obj}}^{(i)} = 1$ if the cell contains an object and 0 otherwise. This loss penalizes both false positives and false negatives, guiding the model to predict boxes only for cells that actually contain objects.

The total loss is a weighted sum of these five individual loss functions, where the weights are training hyperparameters and can be adjusted. The default weights, calibrated for human pose estimation, are:

$$\mathscr{L}_{\text{total}} = 7.5\,\mathscr{L}_{\text{box}} + 12.0\,\mathscr{L}_{\text{pose}} + 2.0\,\mathscr{L}_{\text{obj}} + 0.5\,\mathscr{L}_{\text{cls}} + 1.5\,\mathscr{L}_{\text{dfl}}$$

For RICH ring center identification, the box detection is not directly relevant since our main interest is in keypoints. However, the weights of the box-related losses cannot be set to zero, because YOLO relies on bounding boxes to help localize the keypoints accurately. In fact, some attempts with no box-related losses failed and the model did not converge. Therefore, to emphasize the importance of keypoints over boxes, the loss used to train the model has been adjusted as follows:

$$\mathscr{L}_{\text{total}} = 3.0\,\mathscr{L}_{\text{box}} + 25.0\,\mathscr{L}_{\text{pose}} + 6.0\,\mathscr{L}_{\text{obj}} + 0.5\,\mathscr{L}_{\text{cls}} + 0.6\,\mathscr{L}_{\text{dfl}} \tag{3.4}$$

Here, the weight ratio between $\mathscr{L}_{\text{box}}$ and $\mathscr{L}_{\text{dfl}}$ is kept the same as in the default configuration, while the pose and keypoint objectness weights have been increased to focus the model on accurate keypoint identification. Overall, the model trained with this loss function achieved better evaluation metrics compared to the model trained with the standard loss. This loss is used both for training and validation.

3.2 Training and Results

The training of the YOLOv11-Pose model and its evaluation on a test dataset were performed on a system equipped with an NVIDIA A6000 GPU with 40 GB of dedicated memory and an AMD EPYC 7402P CPU with 128 GB of RAM.

First, the training dataset was generated using the synthetic data generator. A total of 25000 events were generated, 90% used for training and 10% for validation. Each events was generated with a number of rings uniformly distributed in the range $[150, 175]$.

Subsequently, the generated events were rasterized into images in `jpg` format at resolutions of 416×416 px and 800×800 px, while the corresponding YOLO compatible labels were derived from the generator output. In particular, YOLO labels require the bounding box vertex coordinates and the keypoint coordinates, both normalized to the interval $[0, 1]$, and the object class. For simplicity, the bounding boxes were chosen to be square, each circumscribing a ring, while the object class is unique and corresponds to the ring center. The training of the YOLO model was done following the guidance of reference [25].

The training was run for both image sizes without early stopping, with the number of epochs determined by monitoring the validation loss. By default, YOLO selects the best model according to precision and recall evaluated with the Object Keypoint Similarity (OKS) threshold. However, this metric is not suitable for RICH applications, as it was originally designed for human pose estimation. In this context, OKS proved to be unreliable, whereas a Euclidean threshold yielded more accurate estimates of precision, recall and F1-score. Since such Euclidean metrics are not implemented in YOLO, the final models were selected when the validation losses appeared to have converged, which in each training session corresponds to the last epoch. The training and validation loss plots for both image resolutions are shown in Figure 3.2.

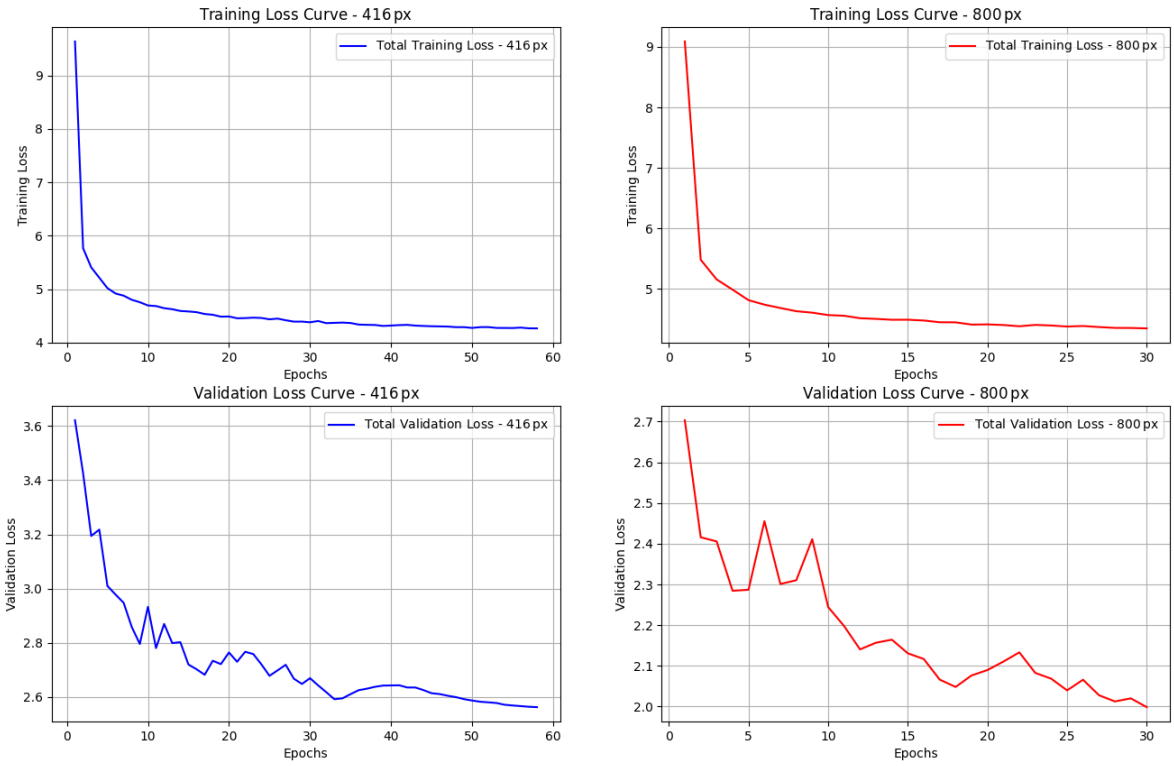


Figure 3.2: *Training and validation loss curves: on the left for images of size* 416 *px and on the right for images of size* 800 *px. The selected model corresponds to the last training epoch, when the validation curve has converged.*

The obtained models were tested on the same validation dataset with the computation of the evaluation metrics. In addition to the keypoint coordinates, YOLO models also return a **confidence score**, which represents the predicted probability that a true keypoint is present at the corresponding coordinates. Only keypoints with a confidence greater than a predefined threshold are included in the output, so the choice of an optimal confidence threshold must be made carefully. Moreover, to compute the evaluation metrics, a Euclidean pixel threshold is required. Assuming an acceptable relative error of $\sigma_{kp} = 1\%$ on the predicted keypoint coordinates on the MaPMTs plane, corresponding to an acceptable error of 12 mm, the Euclidean thresholds for the two image sizes are

$$d_{416\,\mathrm{px}} = 416\,\mathrm{px} \cdot 0.01 \simeq 4\,\mathrm{px} \qquad d_{800\,\mathrm{px}} = 800\,\mathrm{px} \cdot 0.01 = 8\,\mathrm{px}$$

To fully characterize the models, the validation dataset was analyzed using different combina-

tions of confidence values and Euclidean thresholds. For each pair of values, the F1 score was computed and the results are shown in Figure 3.3 and 3.4.

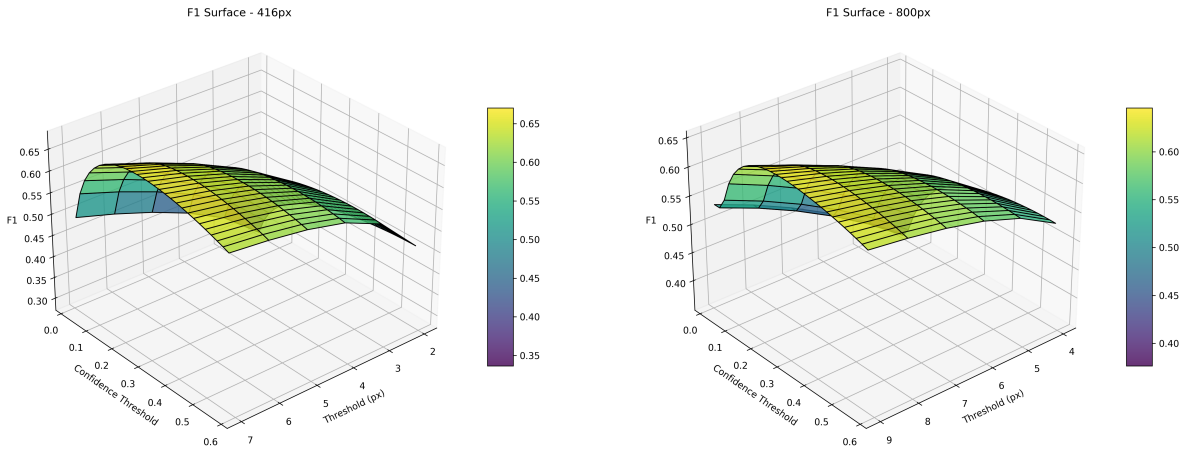


Figure 3.3: *F1 score as a function of the Euclidean and confidence thresholds. Increasing the Euclidean threshold leads to higher F1 values, since it tolerates larger relative errors.*

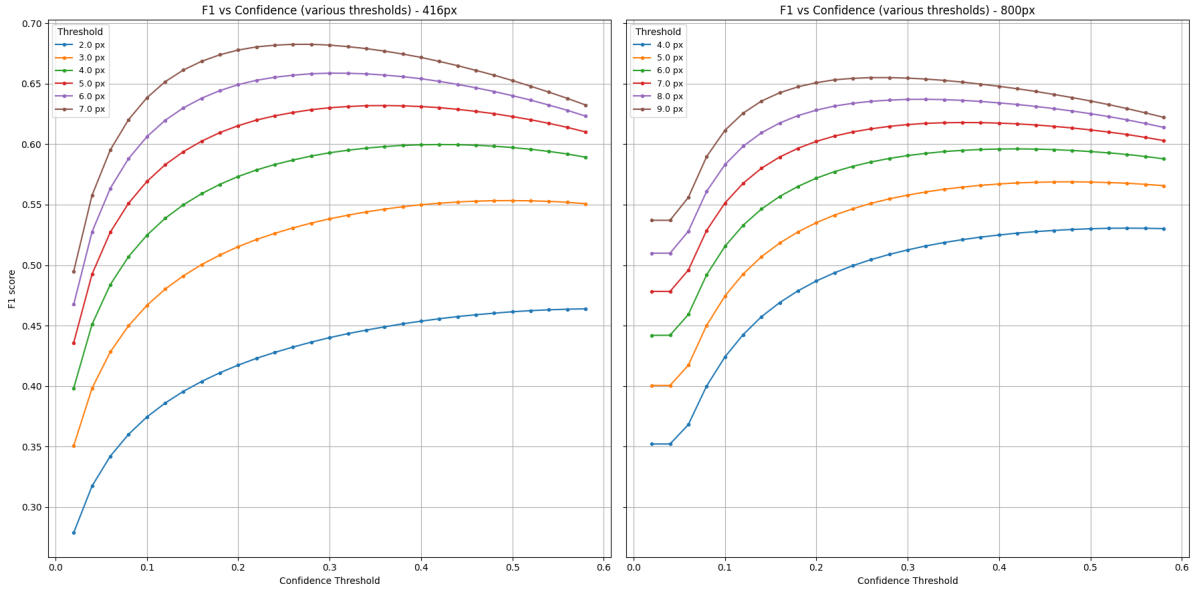


Figure 3.4: *F1 score at different Euclidean thresholds as a function of the confidence threshold. In all cases, the F1 reaches its maximum at intermediate confidence values, while it decreases for very low or very high confidence.*

As expected, the F1 score increases with higher Euclidean thresholds. On the other hand, the confidence score shows a different trend. In fact, the F1 score reaches its maximum when

the confidence is neither too low nor too high. A high confidence leads to a small number of very precise predicted keypoints, which results in high precision but low recall. Conversely, a low confidence produces many predicted keypoints, increasing recall but reducing precision. This behavior is also reflected in the trend of the total number of predicted keypoints, which is reported for both models in Figure 3.6.

At the required keypoint relative error of 1%, the 800 px model generally achieves a higher F1 score compared to the 416 px model. The performance metrics at the optimal confidence threshold are summarized in the tables below:

Img Size	Euclidean Threshold	Optimal Confidence Threshold
416 px	4 px	0.42
800 px	8 px	0.32

Img Size	Precision	Recall	F1 Score	# of Predicted KP	Inference Time (ms)
416 px	0.815	0.475	0.600	92.46 ± 0.06	12.693 ± 0.019
800 px	0.805	0.527	0.637	103.89 ± 0.06	13.541 ± 0.011

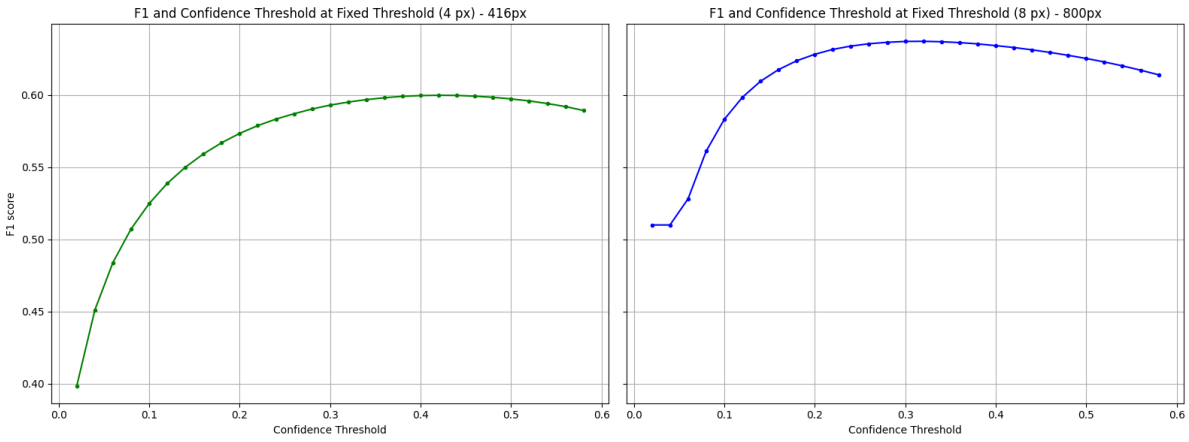


Figure 3.5: *F1 score as a function of the confidence threshold at fixed Euclidean threshold. The* 800 *px model outperforms the* 416 *px one, mainly due to its higher recall.*

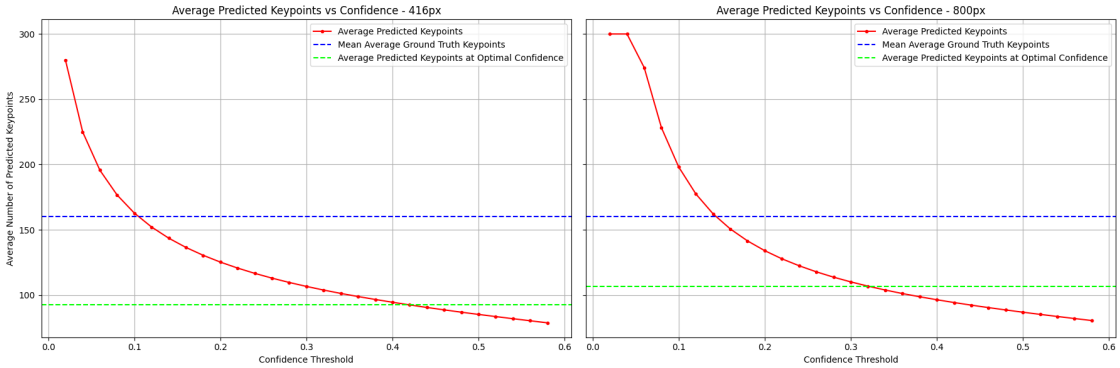


Figure 3.6: *Average number of predicted keypoints as a function of the confidence threshold. At the optimal confidence value, the* 800 *px model predicts more keypoints than the* 416 *px one, which explains its higher recall.*

Overall, the **800 px model slightly outperforms the 416 px one**, mainly due to its higher recall and consequently greater F1 scores. The larger input size preserves more spatial information, enabling more accurate ring centers detection despite a negligible increase in inference times. These results suggest that the 800 px configuration may be the optimal choice, although further investigations on YOLO model size and on the optimal loss function are required. Nevertheless, the recall shows that **the model identifies only 52.7% of the true ring centers**, highlighting a significant limitation in effectiveness.

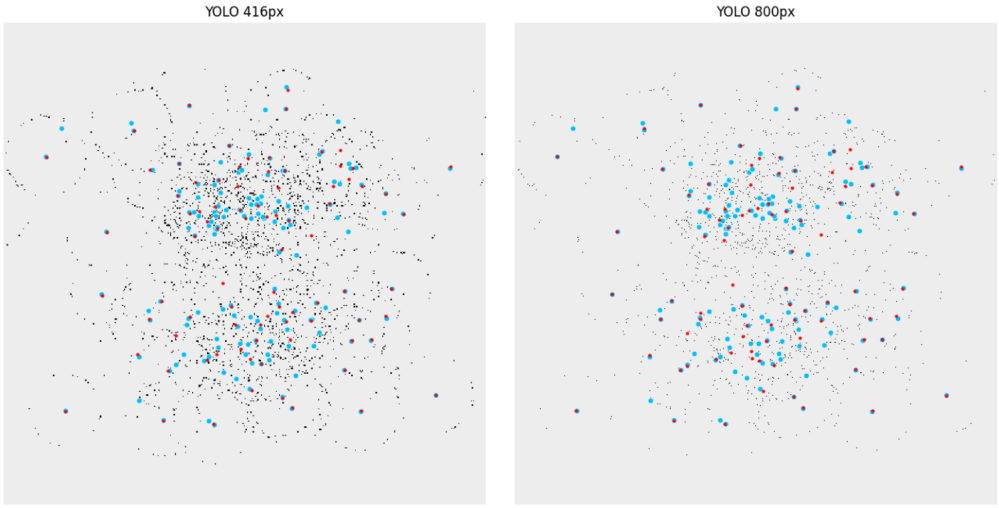


Figure 3.7: *Example of an event inferred by the two YOLO models: red points indicate the predicted ring centers, while cyan points represent the ground truth.*

It is evident from Figure 3.7 that the model struggles to localize keypoints in the central region of the image. Therefore, for a better characterization of the model, it is useful to compute the metrics separately for the central and peripheral zones, where the network shows different levels of efficiency. Based on the distribution of centers in Figure 2.3, each image is empirically divided into two regions: a central square with side length equal to half of the image size, and the surrounding peripheral area, as illustrated in Figure 3.8. The metrics are computed on the evaluation set at the corresponding σ_{kp} pixel thresholds and the results are reported in the table below.

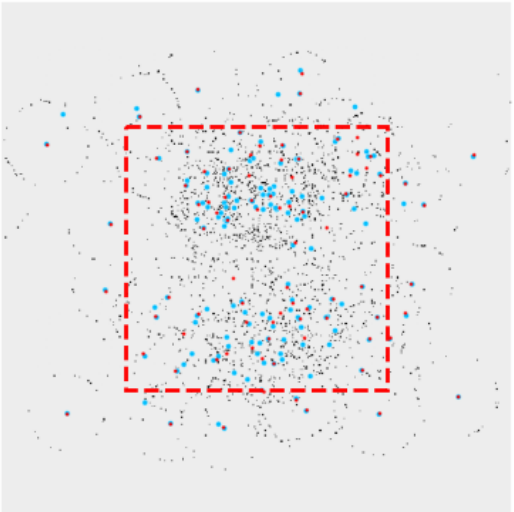


Figure 3.8: *Example of the division of each image into a central region (red dashed square) and a surrounding peripheral area, used for the computation of distinct performance metrics.*

Img Size and Region	Precision	Recall	F1 Score	# of Predicted KP	# of GT KP
416 px center	0.703	0.370	0.485	66.46 ± 0.03	126.27 ± 0.04
416 px peripheral	0.950	0.792	0.864	30.15 ± 0.01	36.17 ± 0.01
800 px center	0.765	0.430	0.550	70.98 ± 0.03	126.27 ± 0.04
800 px peripheral	0.885	0.767	0.821	31.35 ± 0.01	36.17 ± 0.01

As expected, both models achieve better metrics in the peripheral region. However, the 800 px model attains a higher F1 score in the central region compared to the 416 px model, which results in an overall better performance on the entire image.

4. UNet model

The **UNet model** is a convolutional neural network proposed in 2015 by Ronneberger et al. [19] for biomedical image segmentation. It features a symmetrical encoder–decoder architecture, in which the input image is progressively downsampled to extract local features and then upsampled to produce the final output. A distinctive characteristic of UNet is the presence of skip connections between encoder and decoder layers, which allow the network to combine global contextual information with finer spatial details. This architecture makes UNet particularly effective for localization tasks, where it is commonly employed for pixel-wise segmentation and has proven especially successful in binary segmentation problems, such as tumor localization. In this thesis, the network has been adapted in finding the position of Cherenkov ring centers. Instead of producing binary segmentation masks, UNet generates probability heatmaps, whose peaks correspond to the most likely positions of the centers. This methodological choice differs substantially from the YOLO approach described in Chapter 3. In fact, the UNet does not directly output a finite set of keypoint coordinates, but rather a continuous map that must be further processed with a peak detection algorithm to extract the final keypoint positions. The inference process therefore consists of two stages:

1. **Prediction of the heatmap** by the network;

2. **Extraction of local maxima** from the heatmap to obtain the **coordinates** of the centers.

This two step pipeline introduces a computational overhead: inference times are longer compared to YOLO, which instead outputs keypoints directly. Nevertheless, the main advantage of this approach lies in its greater robustness to noise and to overlapping rings compared to regression models. Moreover, unlike YOLO, UNet can be built from scratch, making it easily customizable; in this work it was implemented using the PyTorch library.

4.1 Architecture

The UNet architecture follows an **encoder–decoder structure**. Conceptually, the encoder acts as the contracting part of the network, extracting the most relevant features of the input image,

while the decoder serves as the expansive part, restoring the original spatial resolution and producing the output heatmap. For this reason, the encoder progressively downsamples the image, whereas the decoder performs upsampling. However, the defining feature of UNet, which distinguishes it from classical autoencoders, is the presence of **skip connections**. These connections directly link feature maps from the encoder to those in the decoder, ensuring that important spatial information is not lost during the downsampling process. The inclusion of skip connections between the encoder and decoder gives the UNet its characteristic U-shape, as illustrated in Figure 4.1.

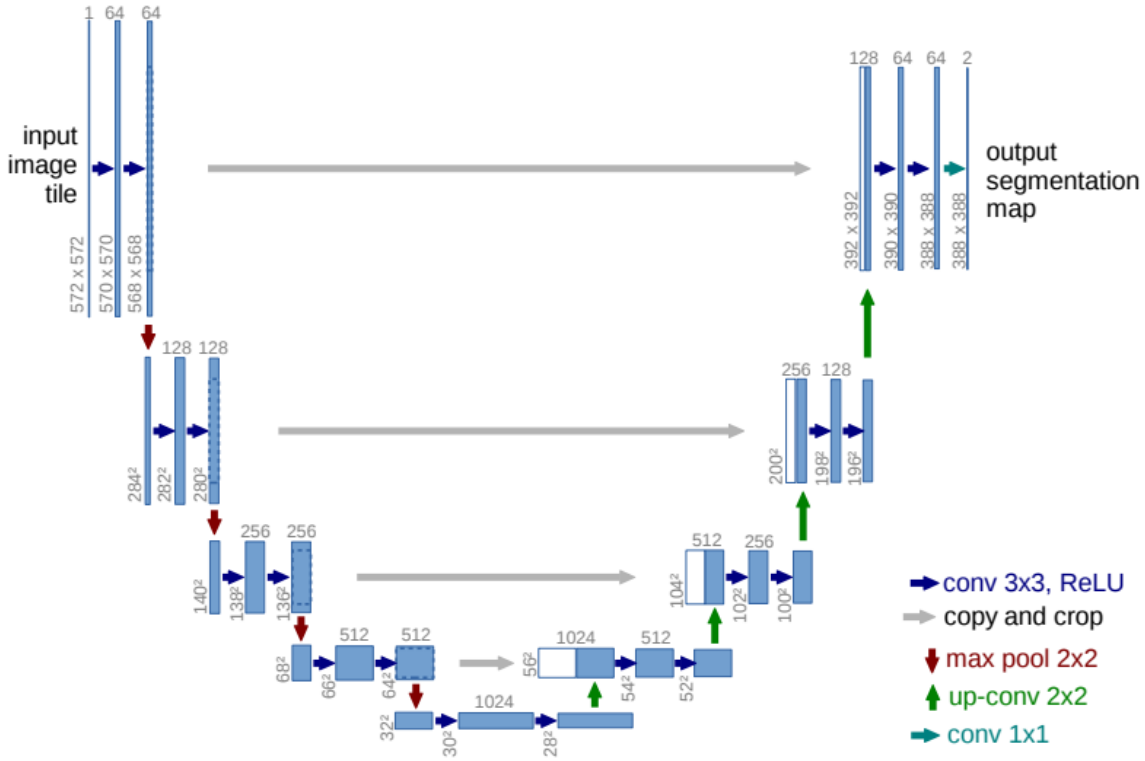


Figure 4.1: *Schematic representation of the UNet architecture. The input image is downsampled by the encoder, which extracts increasingly abstract features. The feature maps are then processed in the bottleneck before reaching the decoder, where they are upsampled to reconstruct the final heatmap. Skip connections concatenate the encoder feature maps with the corresponding decoder features, allowing a more precise reconstruction of the output. Reprinted from [19].*

The UNet models implemented in this work are based on the original structure proposed by

Ronneberger et al. shown in Figure 4.1:

- **Encoder**: it decreases the spatial resolution of the input image while increasing the number of feature channels, allowing the extraction of increasing abstract details. It consists of four main layers, each composed of two distinct convolutional layers. A single convolutional layer employs a discrete convolution with 3×3 kernels, batch normalization and applies ReLU as activation function. Each pair of convolutions is followed by a 2×2 max pooling layer, which reduces the spatial resolution while retaining the strongest activations. In general, the first stage takes the input image and produces 64 different feature channels, whose spatial dimensions are half the input size. In each subsequent stage, the number of channels is doubled, while the spatial dimensions are halved.

- **Bottleneck**: located after the encoder layers, the bottleneck further processes the feature maps using a double convolution block, reaching a total number of 1024 of feature channels. No pooling layer is applied in this stage.

- **Decoder**: the decoder gradually restores the spatial resolution of the feature maps from the bottleneck while reducing the number of channels. Each stage begins with a transposed convolution (known also as deconvolution) that doubles the spatial size and halves the number of feature channels. The resulting feature maps are then concatenated with the corresponding feature maps from the encoder through skip connections. Finally, each concatenated map is processed by a double convolution block to refine the features. This process is repeated for all four stages of the decoder, producing feature maps that progressively approach the original input resolution.

- **Output layer**: after the last decoding stage, a 1×1 convolution maps the features to one output channels, producing a heatmap where peaks correspond to the most likely positions of the Cherenkov ring centers. A sigmoid activation function is applied to obtain a probability heatmap normalized in $[0, 1]$.

The presented architecture has been revised with attention mechanisms. In particular, the final model used for further analysis incorporates **soft attention gates**, which improve UNet's effectiveness. Another attention mechanism, known as **hard attention**, was also explored, but

it showed poor performance and is therefore not considered in subsequent analysis. However, both mechanisms are described in the following sections.

4.1.1 Soft Attention Gates

The attention mechanism enhances neural networks by allowing them to selectively focus on the most informative parts of the input features while suppressing less relevant regions. In the case of UNet with **soft attention gates** [6], this translates into a more effective use of skip connections: instead of forwarding all encoder features to the decoder, the network learns spatial weights that highlight the regions most useful for accurate reconstruction. Since the mechanism is differentiable, these weights can be optimized through backpropagation.
Formally, the gate computes an attention coefficient $\alpha_{ij} \in [0,1]$ for each pixel x_{ij} in every encoder feature map. The gated output is given by

$$\hat{x}_{ij} = x_{ij} \cdot \alpha_{ij} \tag{4.1}$$

which is then passed to the subsequent decoder layers. To compute the coefficients, the attention gate receives two inputs:

1. The upsampled feature map from the decoder $g \in \mathbb{R}^{H \times W \times F_g}$, where $H \times W$ is the feature spatial size and F_g the number of channels;

2. The corresponding feature map from the encoder via skip connection $x \in \mathbb{R}^{H \times W \times F_l}$, where F_l is the number of encoding channels.

Both inputs are first linearly mapped through separate 1×1 convolutions, followed by batch normalization BN, to obtain the same number of channels F_{int}:

$$g_1 = \mathrm{BN}(W_g * g) \quad \text{with} \quad g_1 \in \mathbb{R}^{H \times W \times F_{int}}$$

$$x_1 = \mathrm{BN}(W_x * x) \quad \text{with} \quad x_1 \in \mathbb{R}^{H \times W \times F_{int}}$$

These transformed features are summed element-wise and passed through a ReLU activation:

$$s = \mathrm{ReLU}(g_1 + x_1) \quad \text{where} \quad \mathrm{ReLU}(z) = \max(0, z)$$

ReLU emphasizes positive activations while setting negative activations to zero, which can help highlight potentially relevant regions. Next, another 1×1 convolution W_ψ, followed by a batch normalization BN and a sigmoid function σ, produces a single attention map:

$$\psi = \sigma(\mathrm{BN}(W_\psi * s)) \quad \text{with} \quad \psi \in [0,1]^{H \times W \times 1}$$

Each coefficient ψ_{ij} represents the attention weight α_{ij} for the pixel with coordinates (i, j). The final output is obtained by the Hadamard tensor product:

$$\hat{x} = x \odot \psi \quad \text{with} \quad \hat{x} \in \mathbb{R}^{H \times W \times F_l}$$

where $\odot$ denotes element-wise multiplication. It is important to note that each skip connection is associated with a unique attention weight map. However, within a single layer, the same spatial attention map is applied across all feature channels of the encoder feature map. A schematic and more general representation of the soft attention gate is shown in Figure 4.2.

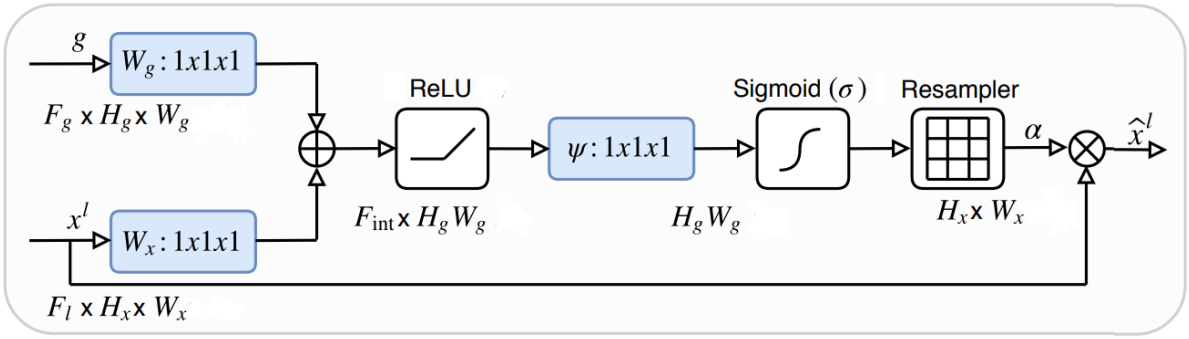


Figure 4.2: *Soft Attention Gate: the inputs g and x^l from the l-th encoding layer are processed through independent convolutional and activation blocks. The diagram illustrates the general case, where the spatial dimensions of the inputs differ and a resampling operation is required. In our implementation, g and x^l share the same spatial size and therefore no resampling is applied. Adapted from [6].*

4.1.2 Hard Attention

In this work, we explore a patch-based variant similar in spirit to hard attention, where the image is divided into fixed subregions processed independently. During training, separate losses are computed for each patch and combined with predefined weights, allowing the network to

learn distinct representations for each region.

In the context of Cherenkov ring center detection with UNet, the hard attention mechanism arises naturally, as the density of ring centers is extremely high in the central region of the image. This high concentration complicates the accurate detection of individual centers due to overlapping rings. By dividing the image into patches, the network can handle the central dense area separately from the peripheral regions, which can improve localization in principle. For this reason, two distinct image subdivisions were considered. The first consists of two patches: one covering the central region, allowing the network to focus on the high density area, and the other covering the complementary peripheral region of the image. The second subdivision consists of three patches, where the peripheral frame remains as a single patch, while the central region is further split into two symmetrical halves. These patches are illustrated in Figure 4.3. The exact pixel subdivisions were determined empirically, but they were guided by the kernel density estimation of ring centers, already shown in Figure 2.3.

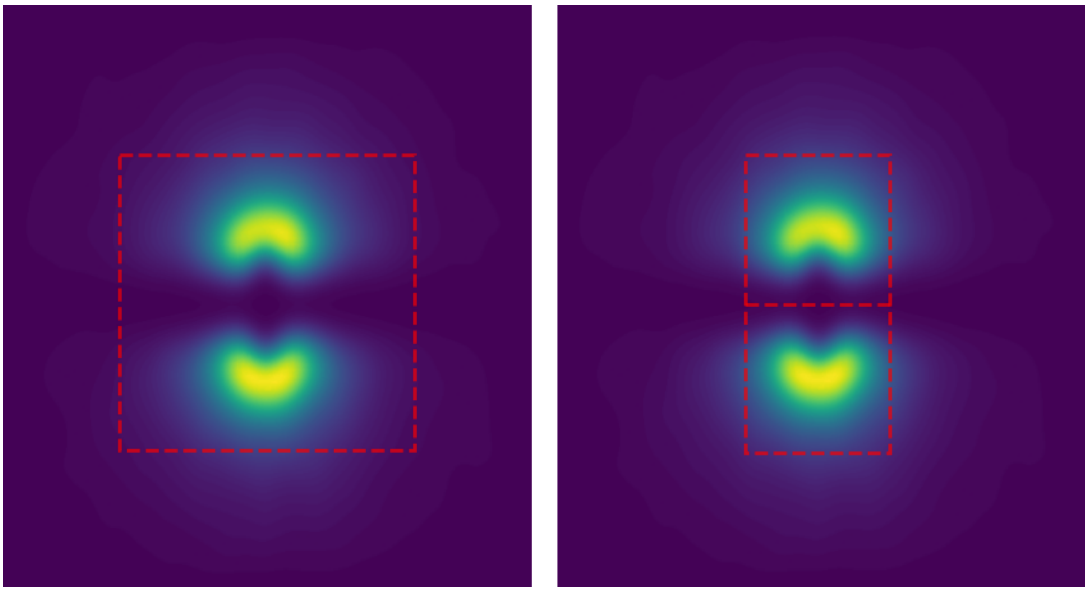


Figure 4.3: *Two distinct configurations of patches for hard attention. On the left, two patches are used: one covering the high density region and the other covering the peripheral zone. On the right, three patches: the central region is narrower and divided into two parts, since there are fewer ring centers in the middle.*

The training was performed using the same dataset employed for the YOLO-Pose model, using different combinations of loss functions described in Section 4.2.2. Additionally, the losses for each patch were weighted differently, since the central patch contains a much higher density of ring centers compared to the peripheral frame patch. In practice, higher loss weights were assigned to the central patches, in order to focus the network on this region. Despite these different strategies, the overall results were poor, as shown in Figure 4.4. Probably, cropping the images caused the network to lose the global context, making it harder to correctly identify ring centers. In particular, rings that were split across two patches became difficult to detect and the network could not effectively integrate information across the full image. This highlights a key limitation of hard attention in this setting: although it forces the network to focus on subregions, it also fragments the spatial relationships which are essential for accurate ring center detection. Consequently, hard attention was not used in subsequent analyses. However, further studies could explore improvements to this approach. Due to time constraints, we focus on the UNet with soft attention gates.

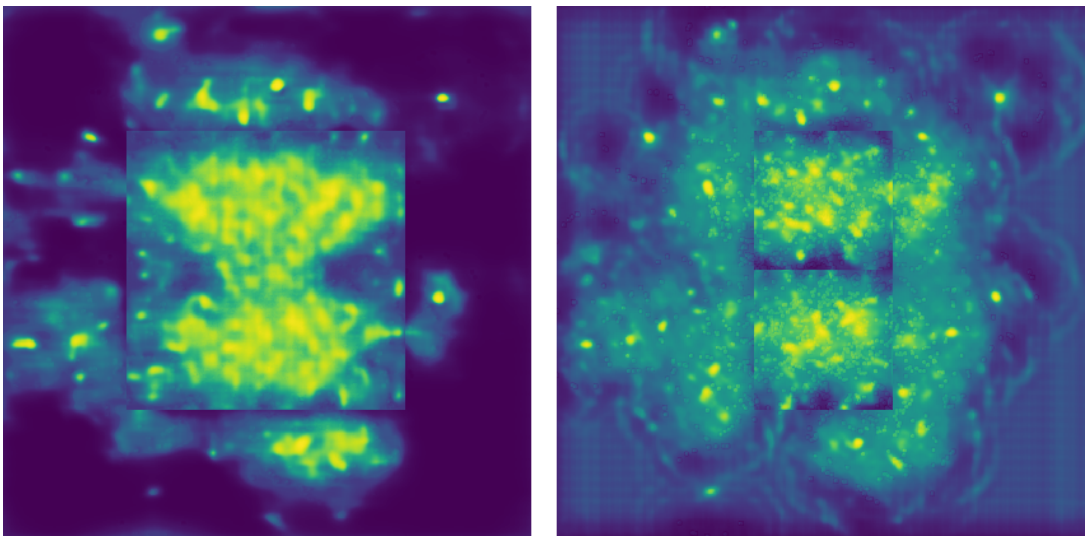


Figure 4.4: *UNet with hard attention heatmaps* $800 \times 800\,px$. *It is clear that, with both patch configurations, the results are highly imprecise, as the peaks corresponding to keypoints are not well defined. Moreover, along the patch borders, no centers are detected because the network does not focus on these areas. Consequently, fewer keypoints are identified here, reducing the recall metric.*

4.2 Training

Training the UNet is a crucial step in the construction and optimization of the network. In the following, only the UNet equipped with soft attention gates is considered, as this configuration has demonstrated better performance and more reliable localization of ring centers compared to alternative approaches. Moreover, the present work explored several approaches, and given the time available, the following configurations proved to be the most effective.

4.2.1 Labeled dataset

Supervised learning relies on labeled datasets. Since the UNet outputs a heatmap as prediction, the corresponding labeled data consist of a heatmap with peaks corresponding to ring centers. To create these label heatmaps, the following pipeline is adopted:

1. Generation of the dataset through the synthetic dataset generator (Section 2.2);

2. Rasterization of the dataset into images in `npy` format for memory efficiency, with grayscale values normalized in $[0,1]$;

3. Storing the ring center coordinate into `CSV` files;

4. For every event, the corresponding label is constructed from an empty image of the same size as the training data. For each center i, a 2D **gaussian function** is generated with mean $\mu_{\mathbf{i}} = (\mu_{x_i}, \mu_{y_i})$ corresponding to the center position and standard deviation $\sigma_{\mathbf{i}} = (\sigma_{x_i}, \sigma_{y_i})$:

$$G_i(\mathbf{x}, \mu_{\mathbf{i}}, \sigma_{\mathbf{i}}) = \frac{1}{2\pi\sigma_{x_i}\sigma_{y_i}} \exp\left[-\frac{1}{2}\left(\left(\frac{x-\mu_{x_i}}{\sigma_{x_i}}\right)^2 + \left(\frac{y-\mu_{y_i}}{\sigma_{y_i}}\right)^2\right)\right] \tag{4.2}$$

   The chosen standard deviations are $\sigma_{x_i} = \sigma_{y_i} = 2$, which occasionally lead to overlapping gaussians in the center of the image. However, these values were selected to facilitate model convergence. Further studies could explore alternative standard deviations, but due to time constraints, only this setting was considered. Finally the label is normalized to $[0,1]$, therefore representing a probability heatmap.

5. Storing the labeled data in npy format.

A label example is shown in Figure 4.5.

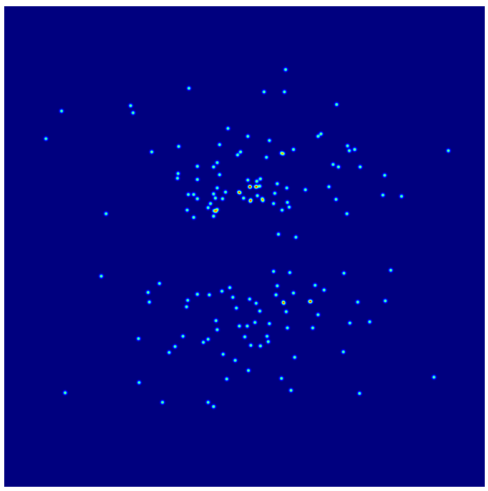


Figure 4.5: *Example of UNet label data: few gaussian blobs overlap in the center of the image.*

4.2.2 Loss Function

In training the UNet model, a combination of three different loss functions was used:

- **Binary Cross Entropy with logits** $\mathscr{L}_{\text{BCE}}$: as already discussed in Section 1.3.1, the BCE computes the loss value over predicted probabilities p_i and two classes $y_i \in \{0,1\}$, where $y_i = 1$ corresponds to the positive class and $y_i = 0$ to the negative one. BCE with logits is a numerically stable formulation of BCE: instead of applying the sigmoid function to the logits z_i before computing BCE, it directly incorporates the logits into the loss function

$$\mathscr{L}_{\text{BCE}} = \frac{1}{N}\sum_{i=1}^{N}\Big[\big(1+(w_{\text{pos}}-1)y_i\big)\max(z_i,0) - y_i w_{\text{pos}} z_i + \big(1+(w_{\text{pos}}-1)y_i\big)\log\big(1+e^{-|z_i|}\big)\Big]$$

  where $\max(z_i,0)$ selects only positive logits, N is the total number of prediction (pixels) and w_{pos} the **positive weight**, designed to adjust the loss function in the presence of class imbalance. This is particularly useful when the positive class is much less represented than the negative one, as it ensures that the loss calculation properly accounts for the unequal distribution of classes. In the RICH application, the positive weight is crucial

because the positive class represents pixels with non-zero values, which are far less frequent than the pixels with zero value. This weighting ensures that the contribution of positive pixels is not overwhelmed by the dominant background. The positive weight is computed before training as follows: first, the sum of positive pixel values N_{pos} is calculated. Then, the positive weight is defined as

$$w_{pos} = \frac{N_{neg}}{N_{pos}} \tag{4.3}$$

where N_{neg} denotes the weighted number of zero-valued pixels, obtained as the image size minus N_{pos}. This computation is performed for all labels in the training dataset and their mean value $\bar{w}_{pos}$ is used during training.

- **SmoothL1 Loss** $\mathscr{L}_{SL1}$: is the same loss based on SmoothL1 used in YOLO-Pose (see 3.1.1 - Pose Loss $\mathscr{L}_{\text{pose}}$).

- **Dice Loss** $\mathscr{L}_{\text{Dice}}$ [22]: it is a loss function commonly used in image segmentation tasks to measure the overlap between predicted and ground truth masks, for example images and their labels. It is derived from the Dice coefficient, which quantifies the similarity between two sets. Given a predicted mask $P^{(j)}$ and a ground truth mask $G^{(j)}$ of a j-th image, the Dice coefficient is defined as

$$\text{Dice}(P^{(j)}, G^{(j)}) = \frac{2\sum_i P_i^{(j)} G_i^{(j)}}{\sum_i P_i^{(j)} + \sum_i G_i^{(j)}}$$

where the summation is over all pixels i. The Dice Loss is averaged across the batch with N images:

$$\mathscr{L}_{\text{Dice}} = \frac{1}{N} \sum_{j=1}^{N} \left(1 - \text{Dice}(P^{(j)}, G^{(j)})\right) \tag{4.4}$$

This loss function is particularly effective in handling class imbalance, since it directly maximizes the overlap rather than treating each pixel independently. Dice Loss encourages the network to correctly predict small but important regions while reducing the influence of dominant background pixels.

The overall loss function is a weighted combination of these three losses, where the weights are hyperparameters of the training. After several trials, the best weights were found to be:

$$\mathscr{L} = 4.0\,\mathscr{L}_{\text{BCE}} + 3.0\,\mathscr{L}_{\text{SL1}} + 3.0\,\mathscr{L}_{\text{Dice}} \tag{4.5}$$

The BCE term is dominant, as it allows the network to generally locate relevant regions thanks to the positive weight parameter, while $\mathscr{L}_{\text{SL1}}$ and $\mathscr{L}_{\text{Dice}}$ are mostly used for fine tuning the predicted keypoints and suppressing background noise.

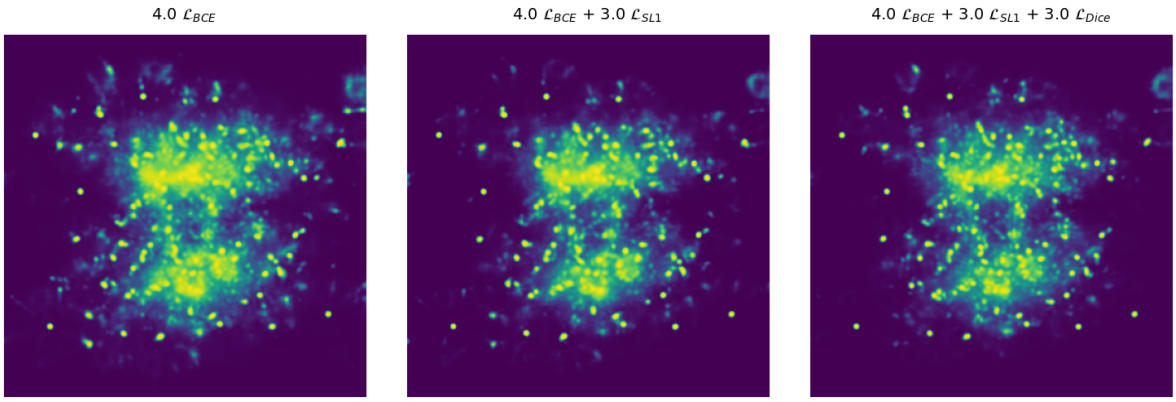


Figure 4.6: *Example of the same inferred image obtained with UNet models trained using different loss functions on the same dataset. The $\mathscr{L}_{BCE}$ loss captures the overall heatmap, while $\mathscr{L}_{SL1}$ and $\mathscr{L}_{Dice}$ refine the predictions by improving precision and suppressing background noise. The corresponding label is shown in Figure 4.5. In particular, note that in the first image some false positive peaks are present, which are then removed in the third image. It is also evident that all three models struggle to identify centers in high density regions.*

4.3 Keypoint Extraction

The UNet model outputs heatmaps, which are continuous probability maps whose peaks correspond to the positions of the ring centers. However, we are interested in discrete keypoint coordinates, both to reconstruct the particle tracks in the RICH gas radiator and to compute validation metrics for performance comparison with the YOLO-Pose model. The **extraction of keypoint coordinates** from the heatmap is a crucial step in the pipeline, as the metrics and the overall effectiveness of the network are strongly influenced by this final stage. Therefore, it requires careful attention and meticulous tuning in order not to compromise the network's performance, both in terms of accuracy and inference time.

The method applied here is the most straightforward and intuitive one and it could serve as a

basis for the development of more advanced keypoint extraction algorithms. It is based on the construction of a binary map from the predicted probability heatmap. First, a binary map is generated from the heatmap, where pixels with values above a predefined **binary threshold** are set to 1 and all others to 0. Then, the binary mask is segmented into connected components, which are groups of white pixels adjacent to each other. For each connected component, the centroid coordinates are computed and taken as the keypoint coordinates:

$$c_x = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad c_y = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{4.6}$$

where (x_i, y_i) are the pixel coordinates belonging to the component i and N is the number of pixels in the component. This step is executed on the CPU, as performing it on the GPU proved to be significantly slower. Finally, the keypoint coordinates are rounded to integer values. In addition to the coordinates, the covariance matrix of the connected component is also calculated to describe the component's shape. It could serve as a systematic uncertainty associated with the keypoint prediction, which may be useful for inferring the track inside the detector.

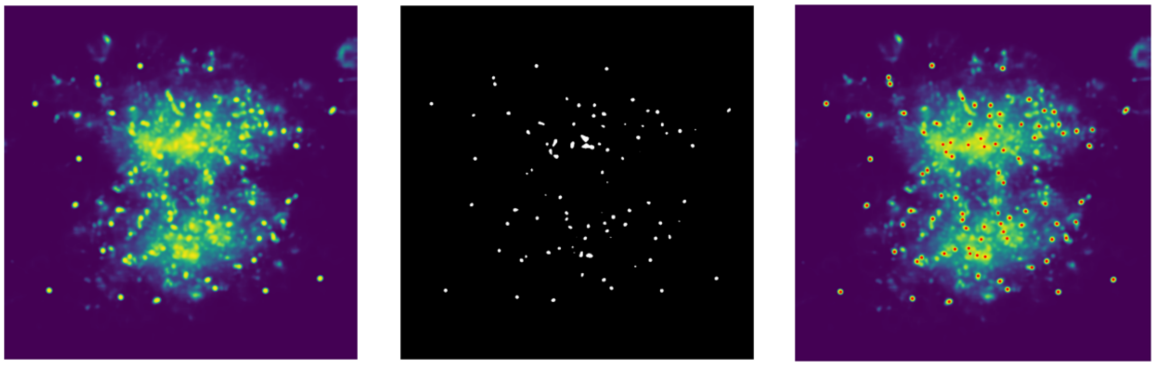


Figure 4.7: *Example of keypoint extraction at the optimal binary threshold. The second image shows the binary map, where each white spot represents a connected component. The red keypoints in the right image correspond to the centroids of these connected components.*

This method requires careful tuning of the binary threshold, as the keypoints predicted by the UNet depend on its value. The chosen threshold corresponds to the one that achieves the highest F1 score at a keypoint accuracy of 1%.

4.4 Results

UNet with soft attention gates models were trained and evaluated on the same hardware as YOLO-Pose (NVIDIA A6000 GPU, AMD EPYC 7402P CPU) using the same training and validation datasets. The events and their corresponding labels were rasterized at resolutions of 416×416 px and 800×800 px in `npy` format. The best models were selected based on the validation loss when it reached its minimum. The validation loss is the same as the training loss defined in 4.5. In particular, unlike YOLO training, early stopping was employed with a patience hyperparameter: if the validation loss does not improve for 7 consecutive epochs, the training session is stopped and the model with the lowest validation loss is considered the best model. Early stopping was triggered quite quickly: the best epoch for the 416 px model was the fifth, while for the 800 px model it was the fourth. This indicates that **the model struggles to improve its localization performance in the center of the image**, where the density of ring centers is the highest. This issue persists even when experimenting with different loss combinations, suggesting that the challenge is inherent to the high density regions rather than the specific choice of loss function.

Both models require their optimal binary threshold to maximize performance, specifically the F1 score. Therefore, F1 scores on the validation dataset were computed as a function of both the binary threshold and the Euclidean threshold. The results are shown in Figures 4.8 and 4.9.

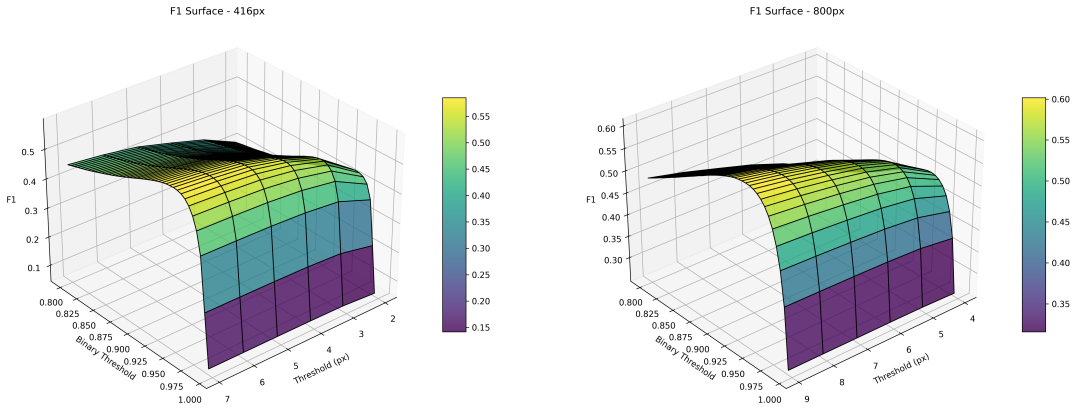


Figure 4.8: *F1 score surface of the UNet models as a function of the binary threshold and the Euclidean threshold on the validation dataset. The optimal binary thresholds are close to* 1*, as expected, since higher binary thresholds select only the most prominent peaks.*

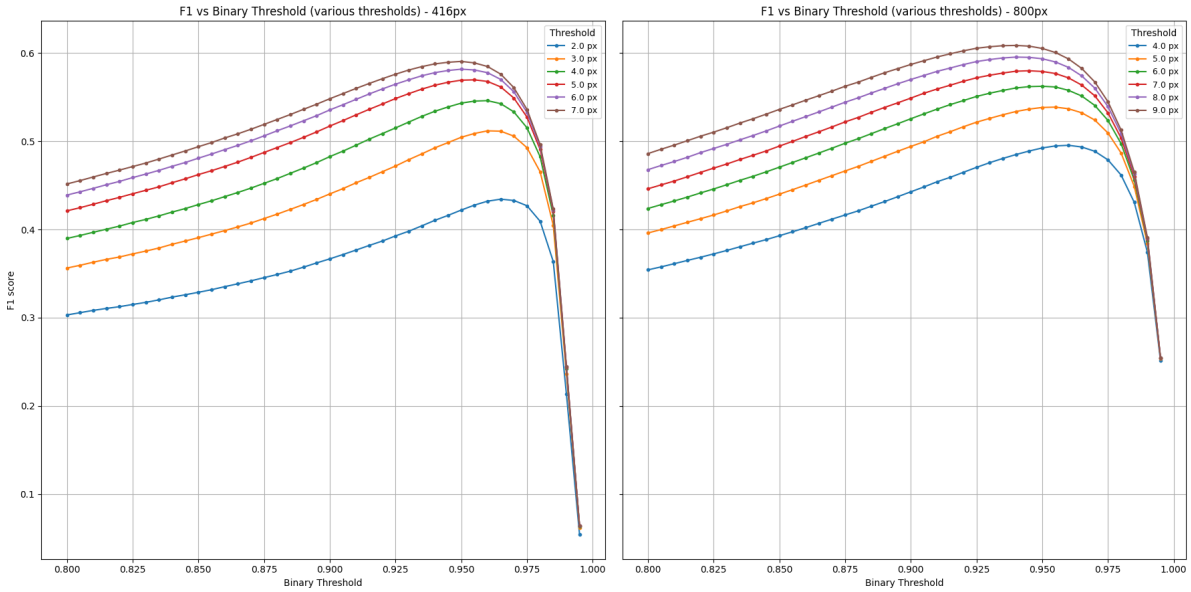


Figure 4.9: *F1 score as a function of the binary threshold. As the threshold increases, only the most sharpened peaks are selected, which affects the overall F1 performance.*

Following the approach used for the YOLO model, we assume an acceptable keypoint error of $\sigma_{kp} = 1\%$, corresponding to a 12 mm error on the MaPMT plane. This translates, for each model, to a Euclidean threshold of $d_{416\,\text{px}} \simeq 4\,\text{px}$ and $d_{800\,\text{px}} = 8\,\text{px}$. For these values, the optimal binary threshold is reported in the table below, along with the mean metrics evaluated on the validation dataset.

Img Size	Euclidean Threshold	Optimal Binary Threshold
416 px	4 px	0.960
800 px	8 px	0.940

Img Size	Precision	Recall	F1 Score	# of Predicted KP	Inference Time (ms)
416 px	0.905	0.390	0.544	69.95 ± 0.11	20.041 ± 0.369
800 px	0.784	0.483	0.597	100.10 ± 0.16	50.504 ± 0.266

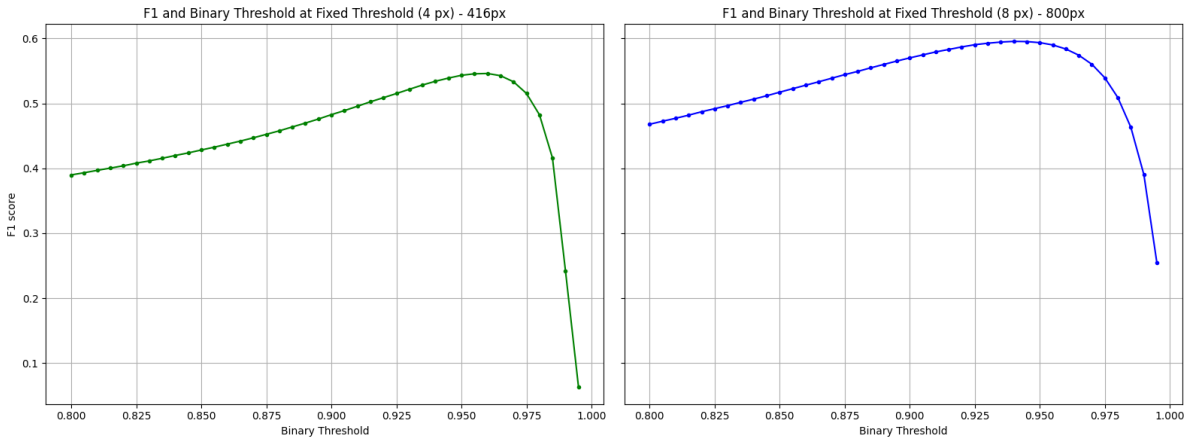


Figure 4.10: *F1 score as a function of the binary threshold at fixed Euclidean threshold. The 800 px model is slightly better than the 416 px one.*

In light of these results, we can draw the following conclusions:

- The 800 px model achieves a higher recall and a lower precision compared to the 416 px model, which overall results in a better F1 score. The higher recall is due to its ability to detect more keypoints than the 416 px model, as shown in Figure 4.11.

- The **F1 score is lower respect to YOLO model**, whose 800 px configuration achieved an F1 of 0.637. This difference is mainly due to the higher recall of YOLO. However, the UNet models tend to exhibit higher precision, as highlighted by the 416 px configuration.

- The 800 px model shows an inference time larger than the 416 px one. This increase is not caused by the heatmap inference itself, which takes on average only ~ 4.51 ms, but rather by the keypoint extraction stage. Since the 800 px images have more resolution than the 416 px ones, the masking and connected components processes must handle more pixels, leading to the observed increase in inference time. As a result, **the overall inference of the heatmap based approach is slower than YOLO**, mainly **because of the post-processing** needed to extract keypoints from the heatmap. The heatmap inference time alone is instead negligible compared to the overall inference time.

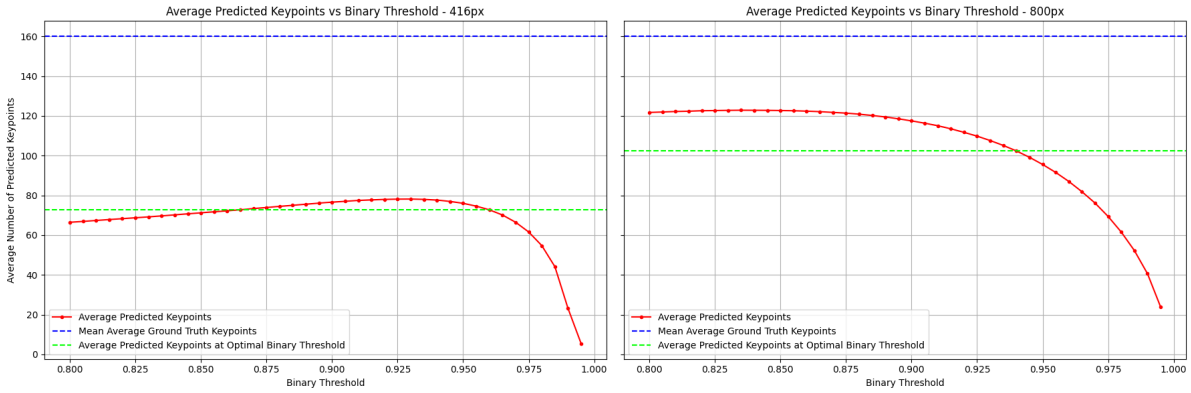


Figure 4.11: *Average number of predicted keypoints as a function of the binary threshold. At the optimal confidence value, the 800 px model predicts more keypoints than the 416 px one, which explains its higher recall.*

To better characterize the UNet models, the metrics are also computed separately for two distinct regions of each image: the central and the peripheral areas, following the same approach used for the YOLO model (see Figure 3.8). The results are reported in the table below:

Img Size and Region	Precision	Recall	F1 Score	# of Predicted KP	# of GT KP
416 px center	0.869	0.303	0.449	43.95 ± 0.03	126.27 ± 0.04
416 px peripheral	0.952	0.687	0.796	26.00 ± 0.01	36.17 ± 0.01
800 px center	0.743	0.411	0.529	69.72 ± 0.03	126.27 ± 0.04
800 px peripheral	0.867	0.723	0.787	30.39 ± 0.01	36.17 ± 0.01

As reported in the table, the UNet 800 px model achieves better performance in the image center, while the 416 px model is more precise in the periphery. Compared to YOLO, UNet reaches higher precision with the 416 px model and similar precision with the 800 px model, but both the models present lower recall. As a result, YOLO attains superior F1 scores, especially in the peripheral region.

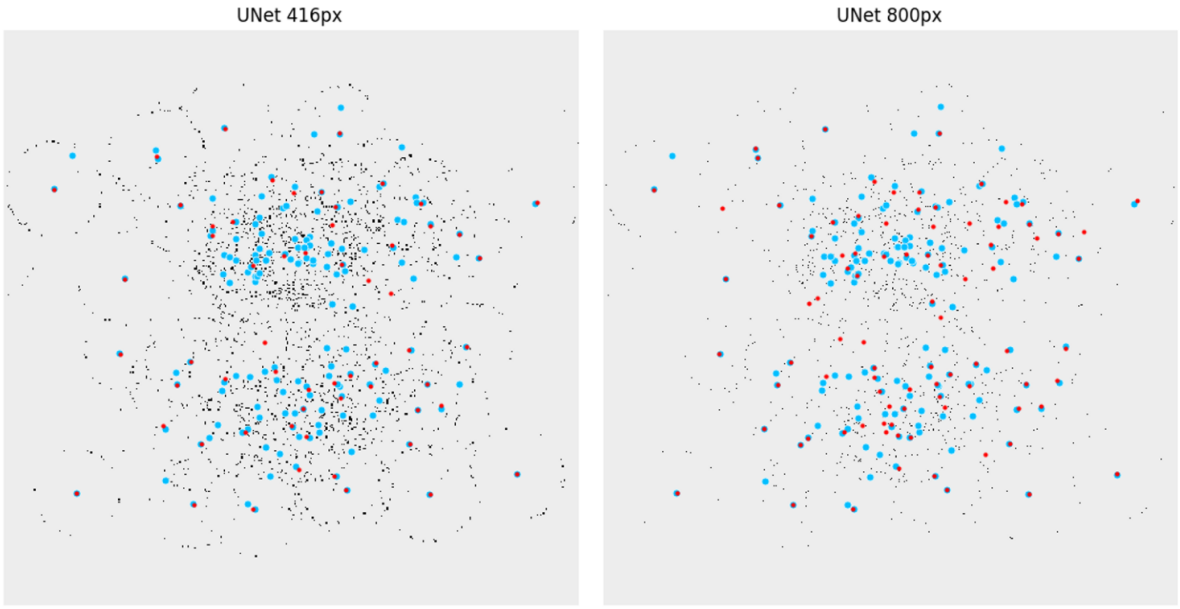


Figure 4.12: *Example of an event inferred by the two UNet models: red points indicate the predicted ring centers, while cyan points represent the ground truth.*

5. Outlook

In this thesis, machine learning models have been investigated as a novel and promising approach to exploit the RICH detectors as tracking devices through the localization of Cherenkov ring centers. The results demonstrate that this strategy is feasible and merits further exploration, particularly in light of the fast growing capabilities of machine learning methods.

Among the two models studied, YOLO outperforms UNet in terms of recall and F1 score, whereas UNet achieves higher precision but detects fewer centers. This outcome suggests that the regression based formulation of the problem is generally more efficient than the heatmap based approach, particularly with respect to inference time. In future real world applications, however, YOLO should not necessarily be regarded as the default choice over UNet: depending on the specific requirements of the experiment, precision may be more valuable than the total number of detected particles. Furthermore, the rasterization of events at higher resolution yields improved evaluation metrics, even if at the expense of increased inference time.

Both networks also exhibit limitations in reconstructing Cherenkov centers in regions characterized by high ring density, leading to an overall maximum recall of 52.7% obtained with the best performing YOLO model. This result indicates that only about half of the true centers are correctly identified within the required precision, highlighting the intrinsic complexity of the task.

In future works, several steps could be pursued to extend and improve the present study:

- **New model architectures**: to obtain better metrics, especially in high ring density regions, new models could be found and adapted to the problem.
- **Optimized training strategy**: data augmentation and loss functions explicitly designed for keypoint detection in high density scenarios could lead to better performances.
- **Parallelization and Edge ML**: parallel computing strategies and the deployment of models and strategies on edge hardware, like quantization, could substantially reduce inference time.
- **Validation on LHCb simulations and real data**: extending the study beyond synthetic

data and evaluating the models on full LHCb simulations and eventually real experimental data will be essential to assess ML applicability in realistic conditions.

- **Particle position reconstruction in the RICH gas radiator**: as a natural continuation of this work, the reconstruction of the particle position in the radiator medium should be investigated, employing optical techniques and dedicated simulations.

- **RICH for time-stamping**: In the second upgrade of the LHCb experiment, RICH detectors will get excellent timing capabilities at the level of tens of picoseconds per photon hit. Exploring the possibility of exploiting RICH detectors in a four dimensional reconstruction framework of tracks (4D), where spatial information is combined with timing, could open promising perspectives for event reconstruction in the harsh conditions of the high-luminosity phase of the experiment.

List of Figures

Bibliography

[1] R. Aaij and Abdelmotteleb et al. The lhcb upgrade i. *Journal of Instrumentation*, 19(05), 2024. Section 11: Trigger and real-time analysis. URL: http://dx.doi.org/10.1088/1748-0221/19/05/P05065, doi:10.1088/1748-0221/19/05/p05065.

[2] R. Aaij and Abdelmotteleb et al. The lhcb upgrade i. *Journal of Instrumentation*, 19(05), 2024. Section 7: RICH. URL: http://dx.doi.org/10.1088/1748-0221/19/05/P05065, doi:10.1088/1748-0221/19/05/p05065.

[3] Martino Borsato. Rich performance studies for u2 scoping document. URL: https://indico.cern.ch/event/1377881/contributions/5861120.

[4] LHCb Collaboration. LHCb PID Upgrade Technical Design Report. Technical report, 2013. URL: https://cds.cern.ch/record/1624074, doi:10.17181/CERN.BD4G.I3VP.

[5] The LHCb Collaboration. The lhcb detector at the lhc. *Journal of Instrumentation*, 3(08):S08005, aug 2008. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08005, doi:10.1088/1748-0221/3/08/S08005.

[6] Ozan Oktay et al. Attention u-net: Learning where to look for the pancreas, 2018. URL: https://arxiv.org/abs/1804.03999.

[7] Maurizio Martinelli Giovanni Laganà, Martino Borsato. Rings of light, speed of ai: Yolo for cherenkov reconstruction. URL: https://agenda.infn.it/event/43565/contributions/260127/.

[8] Pierre Gleize, Weiyao Wang, and Matt Feiszli. Silk – simple learned keypoints, 2023. URL: https://arxiv.org/abs/2304.06194, arXiv:2304.06194.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] John David Jackson. *Classical Electrodynamics (Third Edition)*. Wiley, 1998. Section 13.4, figure 13.5.

[11] John David Jackson. *Classical Electrodynamics (Third Edition)*. Wiley, 1998. pages 638-640.

[12] Tengjiao Jiang, Gunnstein Frøseth, and Anders Rønnquist. A robust bridge rivet identification method using deep learning and computer vision. *Engineering Structures*, 283:115809, 05 2023. doi:10.1016/j.engstruct.2023.115809.

[13] Christian Joram, Ulrich Uwer, Blake Dean Leverington, Thomas Kirn, Sebastian Bachmann, Robert Jan Ekelhof, and Janine Müller. LHCb Scintillating Fibre Tracker Engineering Design Review Report: Fibres, Mats and Modules. Technical report, CERN, Geneva, 2015. URL: https://cds.cern.ch/record/2004811.

[14] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. URL: https://arxiv.org/abs/2410.17725.

[15] Voci Mazzoldi, Nigro. *Fisica, Volume II*. Edises, 2001. pages 489, 530-531.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL: https://arxiv.org/abs/1511.08458, arXiv:1511.08458.

[17] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020. URL: https://arxiv.org/abs/2010.16061, arXiv:2010.16061.

[18] S Nikhileswara Rao. Yolov11 architecture explained: Next-level object detection with enhanced speed and accuracy. URL: https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL: https://arxiv.org/abs/1505.04597, arXiv:1505.04597.

[20] Andrei D Sakharov. Violation of cp invariance, c asymmetry, and baryon asymmetry of the universe. *Soviet Physics Uspekhi*, 34(5):392, may 1991. URL: https://dx.doi.org/10.1070/PU1991v034n05ABEH002497, doi:10.1070/PU1991v034n05ABEH002497.

[21] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[22] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. *Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations*, page 240–248. Springer International Publishing, 2017. doi:10.1007/978-3-319-67558-9_28.

[23] M. Aglieri Rinella G. et al The LHCb RICH Collaboration., Adinolfi. Performance of the lhcb rich detector at the lhc. *Eur. Phys. J. C 73, 2431*, 2013. doi:10.1140/epjc/s10052-013-2431-9.

[24] Ultralytics. Loss function used in yolo model. URL: https://github.com/ultralytics/ultralytics/blob/main/ultralytics/utils/loss.py.

[25] Ultralytics. Ultralytics yolo11. URL: https://docs.ultralytics.com/it/models/yolo11/.