EPICAL-2 particle shower analysis with neural networks

Jan Scharf

Bachelor's Thesis

September 2023

Institut für Kernphysik
Fachbereich Physik
Goethe-Universität Frankfurt am Main

Supervisor: Prof. Dr. Henner Büsching
Second examiner: Prof. Dr. Harald Appelshäuser

Contents

Chapter 1

Introduction

Particle accelerators are used all over the world to study the physics of the smallest particles through particle collisions. Many different new particles can be created in particle collisions. A variety of particle detectors are used to identify and measure the created particles. Calorimeters form one group of particle detectors that are used to determine the energy of the created particles. When a particle enters a calorimeter, it creates a particle shower. Conventional calorimeters measure the energy that is deposited inside the calorimeter by these particle showers to determine the energy of the created particle. The digital calorimeter used in this work employs another principle to measure the energy of the created particle. It measures the spatial distribution of the particle showers, through highly-granular pixels. Therefore, new methods for the determination of the energy of the particles are needed. Classical approaches use deterministic functions based on theoretical reasoning to derive the energy of the particles. In this work, modern machine-learning approaches are explored for the prediction of the energy of the created particles and the identification of the type of the created particles. In contrast to classical approaches, machine-learning approaches determine the energy of the particle through statistical connections found in the data and do not require input from a theory. The use of machine learning in high-energy physics is becoming more and more common. Therefore, this work explores the possibilities of machine-learning approaches for new digital calorimeters.

1.1 Calorimeter physics

Calorimeters represent a group of particle detectors that are used to determine the energy of a particle entering the calorimeter. The particle entering the calorimeter is referred to as the *primary particle*, which initiates a cascade of inelastic collisions in the calorimeter. This cascade is called a *particle shower*. Particles produced

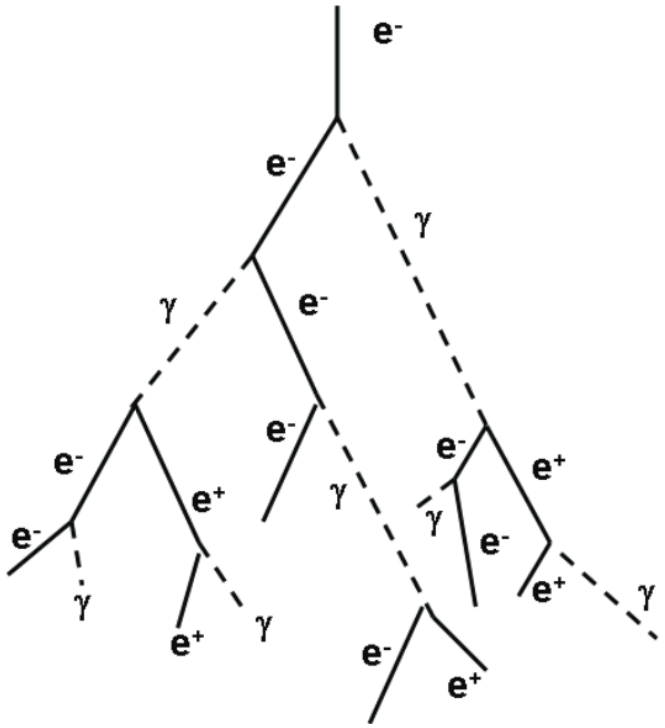

Figure 1.1: Illustration of an electromagnetic cascade [Fra11].

in a particle shower through inelastic collisions are called *secondary particles*. The physics processes underlying the inelastic collisions depend on the type of the primary particles. Typically, one can distinguish two types of particle showers: the electromagnetic and hadronic shower. In the following, a short description of both types of particle showers is given [KW20].

Electromagnetic showers can be initiated by electrons, positrons, or photons. Figure 1.1 shows a cascade of an electromagnetic particle shower. A particle shower is created through repeating processes of photon radiation by electrons and positrons and electron-positron pair creation by photons. The radiation of photons by electrons and positrons is caused by Bremsstrahlung. An important parameter is the *radiation length* X_0, which is defined as the length at which the primary particle's energy is reduced to e^{-1} of its initial energy through bremsstrahlung. The radiation length depends on the characteristics of the calorimeter and especially on its material. For example, the length needed for the particle to deposit all its energy in the calorimeter can be estimated through the radiation length [KW20].

Hadronic showers are produced by hadrons. They can be described through multiple inelastic processes of the strong and electromagnetic interaction. Figure 1.2 illustrates a cascade of a hadronic shower. Typically, a hadronic shower can contain

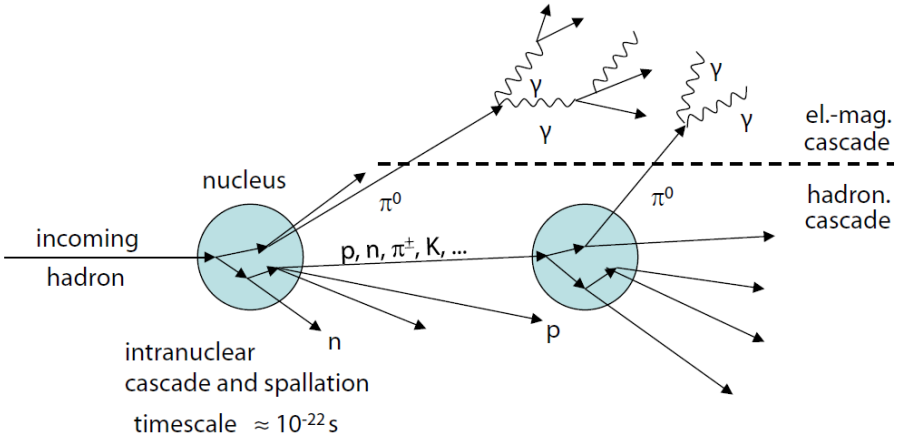

Figure 1.2: Illustration of a hadronic shower [KW20].

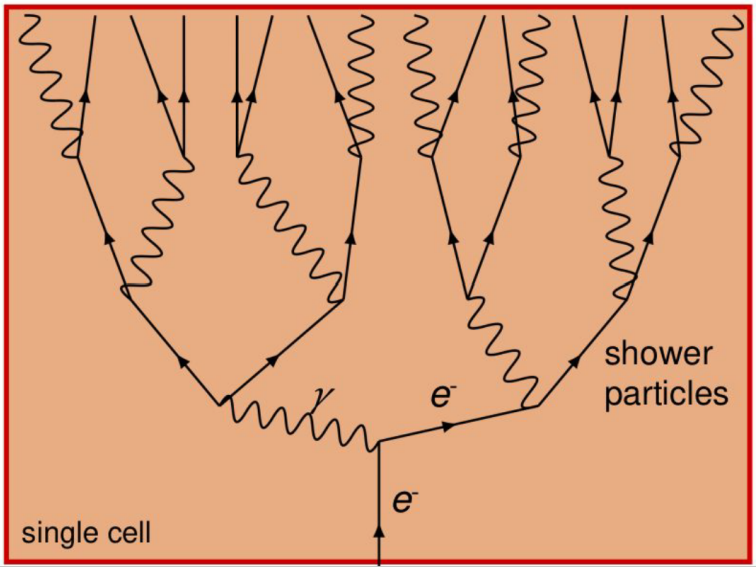

Figure 1.3: Illustration of an electromagnetic shower in a readout cell of a conventional electromagnetic calorimeter [Rog].

a hadronic and an electromagnetic cascade. Electromagnetic cascades mostly occur, when a π^0, either being the primary or secondary particle, decays into two photons. Similar to the radiation length defined for electromagnetic showers, the hadronic absorption length λ_a is defined as the length at which the number of hadrons is reduced to e^{-1} of the initial number of hadrons [KW20].

Different calorimeters are used to measure electromagnetic and hadronic showers. *Hadronic calorimeters* require a greater depth than *electromagnetic calorimeters*, as the hadronic absorption rate is generally larger than the radiation length for the same material. Therefore, it is not expected that a hadron will deposit all its energy in an electromagnetic calorimeter. The type of calorimeters covered in this work are

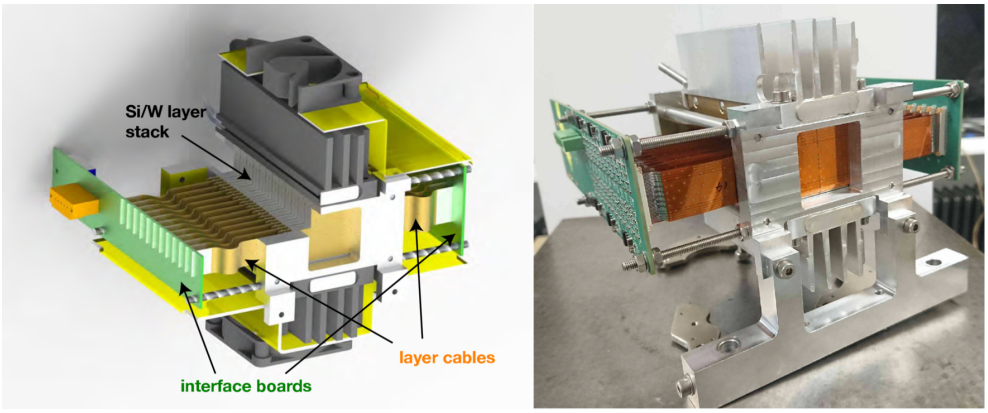

Figure 1.4: Design drawing (left) and picture of the EPICAL-2 (right) [Alm+23].

sampling calorimeters. They are built by two different alternating kinds of layers. The *passive* or *absorber layers* initiate the inelastic collisions, and the *active layers* are used to measure the energy and position of the secondary particles. In conventional calorimeters, it is assumed that a particle shower occurs within a single readout unit or cell. This is shown in Figure 1.3. A single readout unit usually contains many active and passive layers, with which it can measure the energy of the secondary particles. The energy deposited in this readout cell is assumed to be proportional to the energy of the primary particle so that the energy of the primary particle can be derived from the measurement of the cell. However, the digital pixel calorimeter used in this work does not follow the principle of conventional calorimeter to measure the energy of the primary particle. The next section introduces the electromagnetic calorimeter used in this work and its method to measure the energy of the primary particle.

1.2 EPICAL-2

The **e**lectromagnetic **pi**xel **cal**orimeter EPICAL-2 is an ultra-high granularity digital electromagnetic calorimeter prototype. EPICAL-2 was developed to explore the suitability of **AL**ICE **pi**xel **de**tector (ALPIDE) chips for electromagnetic calorimetry and its possible application in the **Fo**rward **Cal**orimeter FoCal of ALICE at CERN. The FoCal is an upgrade to the ALICE experiment, it combines the new high-granularity layers with conventional calorimeter layers [Col23]. In the following, the structure of EPICAL-2, a short description of the ALPIDE chips, and the classical process of measuring the energy of the primary particle are given. Figure 1.4 shows a picture of the EPICAL-2 [Alm+23].

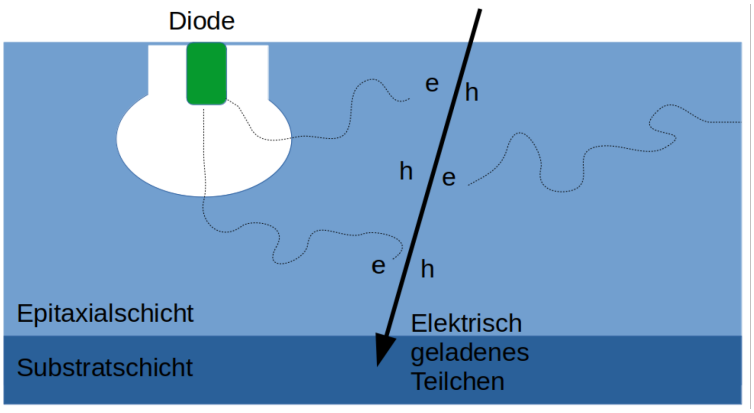

Figure 1.5: Illustration of an ALPIDE chip and a traversing charged particle [Sch22].

The EPICAL-2 consists of 24 layers. Each layer consists of a 3 mm thick tungsten absorber and an active layer using two ALPIDE chips side by side with a thickness of 50 µm. Each chip is made up of 512 × 1024 pixels. The two axes in a chip are named *columns* and *rows*. The EPICAL-2 surfaces cover an area of 30 × 30 mm^2, with a sensitive area of 27.6 × 29.9 mm^2 and 1024 × 1024 pixels. The detector has a depth of 85 mm and $\sim 25 \times 10^6$ pixels. The detector is oriented such that the first layer has no absorber in front of the ALPIDE chips. One of the two ALPIDE chips in layer 22 is broken and inactive [Alm+23].

Figure 1.5 shows the structure and measuring process of an ALPIDE chip. When a charged particle traverses the ALPIDE chip, it produces charges in the form of electron-hole pairs. The electrons are collected by diodes through electrical fields. If the collected charge exceeds a certain threshold, then the particle is measured. Each diode represents a pixel of the ALPIDE chip. If the collected charge in the diode exceeds a certain threshold, then the pixel is measured as a *hit*. The ALPIDE chip does not measure the deposited energy of the particle. A more detailed description of the ALPIDE chips can be found in [Alm+23].

Figure 1.6 illustrates the pixel response to an electromagnetic shower. From the pixel hits and their coordinates, a three-dimensional spatial hit distribution can be reconstructed that shows the shape of the particle shower. This spatial hit distribution can be used to derive the energy of the primary particle. The classical approach assumes a proportionality between the energy of the primary particle and the total number of hits $E \sim N_{hits}$. In the following, metrics for evaluating this proportionality based on the dataset used in this work and later introduced are presented. The first metric to evaluate this proportionality is the linearity between the energy of the primary particle and the mean of the total number of hits. Figure 1.7 (left)

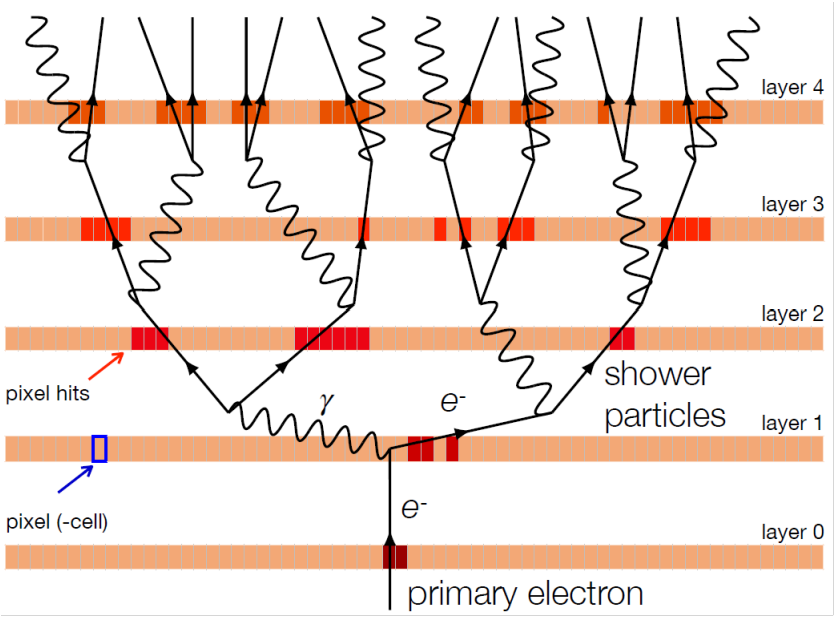

Figure 1.6: Illustration of an electromagnetic shower and pixel layers [Rog].

shows this linearity. The mean values of the total number of hits are fitted with the linear function $y = m \cdot x$ to determine the proportionality factor m. In the bottom part of the figure, the relative difference between the mean of the total number of hits and the linear fit function is shown. Figure 1.7 (right) illustrates the energy resolution of the total number of hits. The energy resolution describes the relative deviation of the total number of hits from the mean for the respective energy. Lower energy values show a higher relative deviation of the total number of hits than the higher values. The energy resolution of this classical approach has a proportionality of $\frac{\sigma}{\mu} \sim \frac{1}{\sqrt{E}}$. In Section 3.4 these metrics are compared to the results of the machine learning approach used in this work and [Alm+23].

1.3 Analysis goal and methods

In this work, the type of primary particle and the energy of electron events are derived from the spatial distribution of the particle shower using machine learning algorithms. An overview of the analysis goals, methods, and used data is illustrated in Figure 1.8. The analysis is split into three tasks. The *particle identification (PID)* classifies the type of primary particle of an event. The *binary particle classification (Binary Clasi)* serves as a simple version of the particle identification and only distinguishes two types of primary particles at one energy. The *electron energy regression (EER)* predicts the energy of electron events. These tasks are studied with two different machine-learning algorithms: the *feedforward neural network (FNN)*, and the *convolutional neural network (CNN)*. As the spatial hit distribution consists

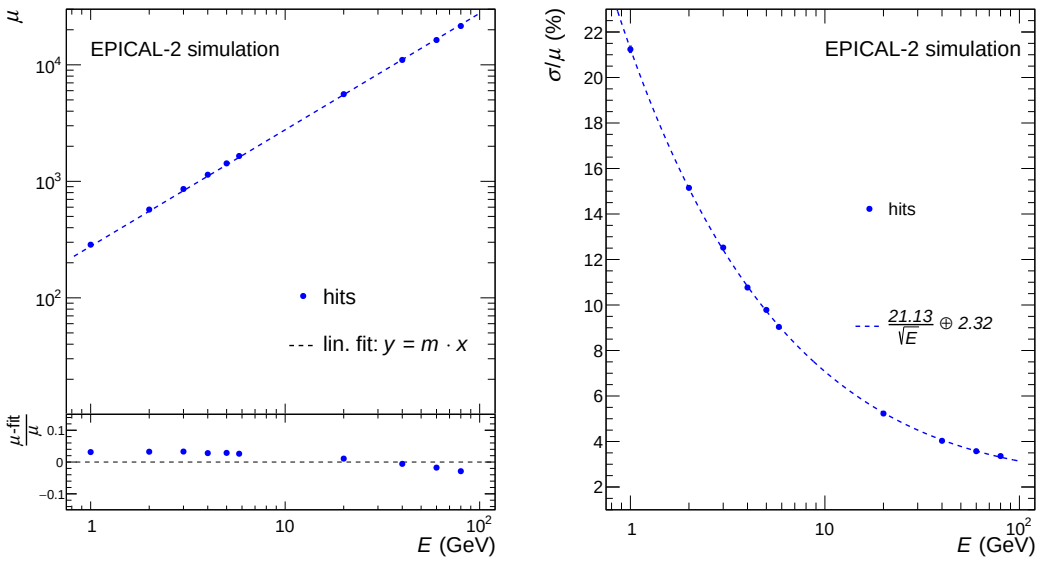

Figure 1.7: Energy linearity (left) and resolution (right) of the classical approach, using the total number of hits to estimate the energy of the primary particle [Sch].

of either $\sim 25 \times 10^6$ pixel responses (being either no hit or a hit) or usually $\sim 1 \times 10^4$ hits, it is not a suitable data format for the networks used in this work due to its high dimensionality. The CNN and FNN require a smaller number of parameters extracted from the spatial hit distribution to work efficiently. A detailed analysis of the extracted parameter is given in Chapter 3.1.

1.4 Data

To create the different machine learning models used in this work, data is needed. The model algorithms require a dataset in which the type of primary particle and its energy are known for every event. Therefore, the dataset used in this work contains Monte Carlo simulations of the EPICAL-2 detector response and thus its measurements of the spatial hit distribution. The EPICAL-2 simulation is performed using Allpix^2, which is a generic pixel detector simulation framework based on GEANT4 and ROOT [Alm+23]. A simulated event consists of registered hits and their coordinates. In the simulation, one chip in layer 22 is set inactive, as the chip is broken in the detector. The dataset consists of different types of primary particles at different energies. In the following, the energy of the primary particle is denoted with E_{prim}. The content of the dataset is described in Table 1.1. The dataset was provided by [Rog].

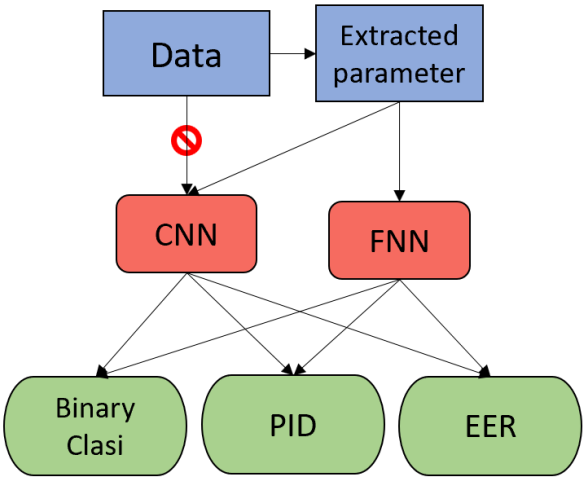

Figure 1.8: Illustration of the different analysis tasks (green), methods (red), and data format (blue) they work on.

Particle	E_{prim} in GeV	Number of events per energy
e^-	1, 2, 3, 4, 5	$\sim 100,000$
e^-	20, 30, 40, 60, 80	$\sim 40,000$
μ^-	20, 30, 40, 60, 80	$\sim 50,000$
π^+	20, 30, 40, 60, 80	$\sim 50,000$
κ^+	20, 30, 40, 60, 80	$\sim 50,000$
p^+	20, 30, 40, 60, 80	$\sim 50,000$

Table 1.1: Datasets used in this work. A more detailed description of the number of events per energy can be found in Appendix A.

Figure 1.9 shows examples of the spatial hit distribution of a simulated hadronic (left) and electromagnetic (right) shower. Figure 1.10 projects these spatial hit distributions onto the columns to emphasize the difference between the two types of showers. The two example events show a significantly different spatial hit distribution and therefore can be distinguished from each other. The pion event has a narrower particle shower and fewer hits than the electron event. This is important, as the particle identification and the calculation of the energy of the primary particle are based on the differing spatial hit distribution. The next chapter provides a detailed description of the two machine-learning approaches used in this work.

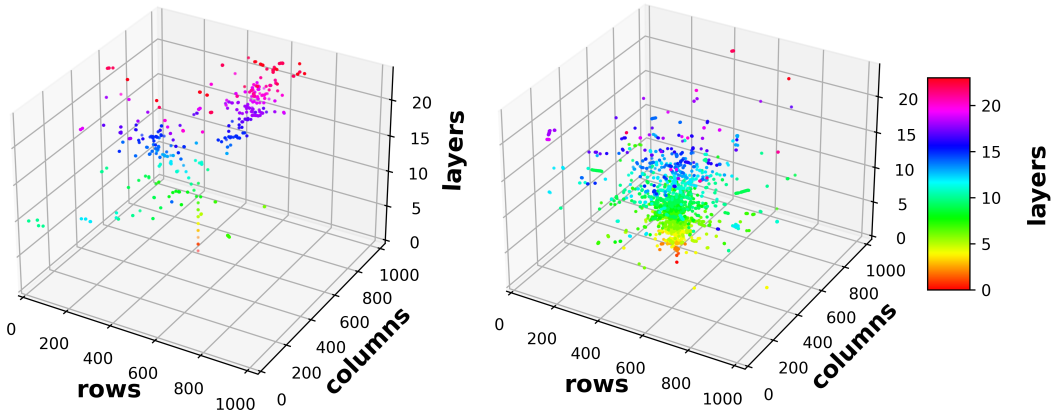

Figure 1.9: Spatial hit distribution of an example pion (left) and electron event (right).

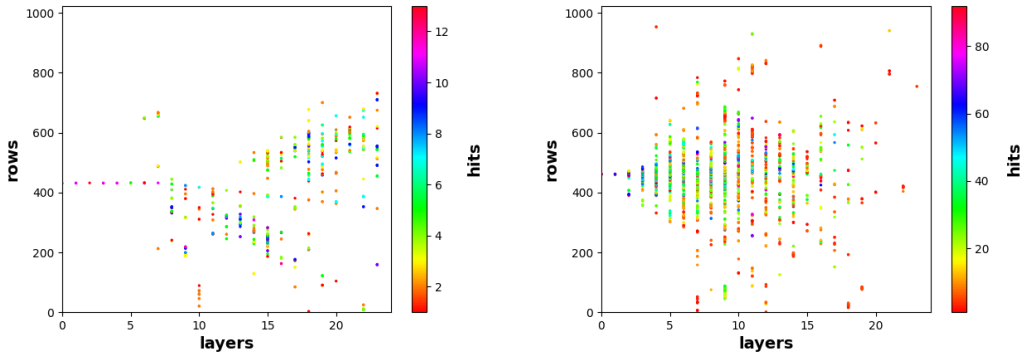

Figure 1.10: Projection on to the columns for a pion event (left) and electron event (right). These are the same events as in Figure 1.9.

Chapter 2

Concept of neural networks

This chapter introduces the feedforward and convolutional neural network. Before the detailed description of these neural networks, general terms of machine learning are introduced and explained. Then, metrics that are used to evaluate the performance of the models shown in this analysis are described. The introduction to neural networks follows the arguments in [Gér22].

In machine learning and other kinds of data analysis, *datasets* are collections of individual *data points*. A data point is represented by the *features* of this data point. However, other features of a data point that are not given or measured might be of interest to the analysis. These other features usually correlate with the given features and thus can be estimated or calculated. For example, a healthiness or nutritional content score, such as the Nutri-score, of a dataset containing different food items, is calculated. Every food item is defined by salt, fat, and other nutritional contents. Calculating the healthiness or nutritional content score represents a new derived feature. In machine learning, the given features are called *features of the data point*, and the features that have to be derived are called *labels*.

The class of machine learning algorithms used in this work aims to find a function that maps the features to the labels. This requires data where the labels are already known, also called *labeled data*. All machine learning algorithms requiring labeled data to find the optimal mapping function are called *supervised algorithms*. The process of determining a mapping function is called *training*. After training, the algorithm can be used on *unlabeled datasets* to *predict* their labels. There are two different kinds of labels, *continuous values* and *classes*. The group of machine learning algorithms that handle labels with continuous values is called *regression*, and the group of machine learning algorithms that work on class labels is named *classifications*. The *task* of a machine learning algorithm is defined by its objective,

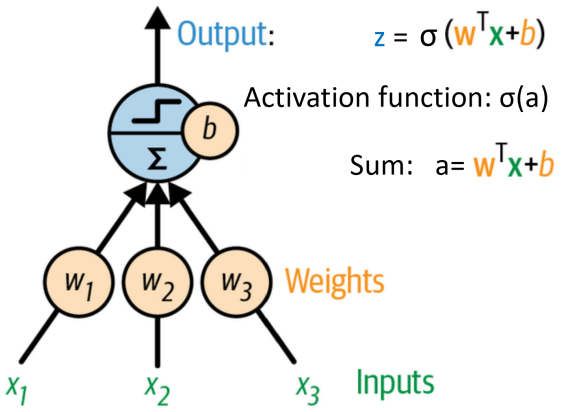

Figure 2.1: The structure of a single neuron, modified [Gér22].

containing the derived features and their respective data format. The success of a machine learning algorithm depends on the specific task and the model used to achieve it.

After this general introduction to machine learning algorithms, three kinds of supervised machine learning algorithms are presented. In this thesis, the analysis uses two types of algorithms: the *feedforward neural network (FNN)*, and *convolutional neural network (CNN)*. All the networks consist of *base units* that are also described in the following.

2.1 The single neuron

The *single neuron* is the base unit of the *feedforward neural network*. Figure 2.1 shows the structure of a neuron. In this example, three *input values* are used to calculate the *output* of the neuron. Generally, the number of input values is defined by the number of features of the data points. For M features, the input for the data point i is written as a *feature vector* or a so-called *input vector* $\vec{X}^i = (x_1^i, x_2^i, \cdots, x_M^i)$. Every input gets multiplied by an *adaptable weight* w_m, which is stored in the so-called *weight vector* $\vec{W} = (w_1, w_2, \cdots, w_M)$. The products are added up with a *bias* b. This sum is called *activation* a. The calculation of the activation a is equivalent to a linear regression. An *activation function* $\sigma(a)$ is applied to the activation. This function usually adds a *non-linearity* to the neuron. The neuron's activation function can be chosen from a set of several different linear and non-linear functions. The result of the activation represents the *output* of the neuron. Here, the output

of the neuron is described as z^i. The following equation describes this process:

$$z^i = \sigma(\sum_{m=1}^{M} w_m \cdot x_m^i + b) \qquad . \tag{2.1}$$

It is good practice to split the dataset into a *training*, *validation*, and *test dataset* before the training of a neuron. The weights and biases are adjusted using the training dataset to minimize the difference between predicted and actual labels. This optimization process is called training. Training is performed on every data point in the dataset and repeated multiple times. One complete iteration over the training dataset is called an *epoch*. The validation and test dataset are used for the *evaluation* of the model performance at different steps. A more detailed description of the training process and the different activation functions can be found in [Gér22].

2.2 Feedforward neural networks

A *feedforward neural network*, or *FNN* for short, is built by combining single neurons side by side and in series. This network only passes values in one direction, called forward, giving it the name forward neural network. It can be separated into *layers*, with each layer receiving the output values of the previous layer as their input values. A layer can contain one or more neurons that work side by side, they are not connected to each other but to all the neurons of the previous layer. A layer of the feedforward neural network is referred to as a *dense layer*. Figure 2.2 illustrates the structure of a feedforward neural network. The first layer is called the *input layer*. This layer is not made of neurons but symbolizes the input feature vector $\vec{X}^i$. The last layer is called the *output layer*. All layers between the input and output layers are named *hidden layers*. Commonly, the neurons used in the hidden layer have the same activation function. The output layer usually has a different activation function depending on the task of the network. Feedforward neural networks only work on data that can be represented in a vector and higher dimensional data can result in a high number of weights, which is not computationally efficient. For example, a feature vector with 1024 features is connected to a dense layer with 1024 neurons. This already uses $1024 \cdot 1024 \approx 1 \times 10^6$ weights and 1024 biases. Because of this, other networks targeting different data types were developed.

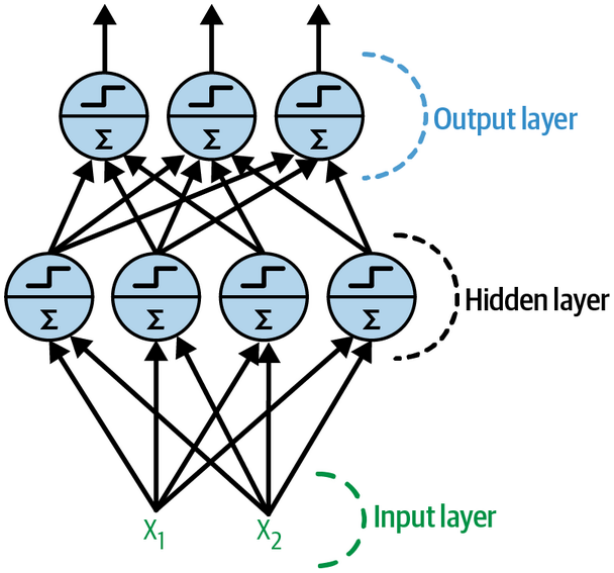

Figure 2.2: The basic structure of a neural network [Gér22].

2.3 Convolutional neural networks

Another kind of neural network used in this work is the *convolutional neural network*, or *CNN* for short. In contrast to the feedforward neural network, a convolutional neural network works on grid-structured data and not just vectors. It is also more efficient on high-dimensional datasets. Convolutional neural networks are usually used for images, as they can capture two-dimensional related features. A particle shower measurement of the EPICAL-2 can be described as a three-dimensional picture, introducing the use of convolutional neural networks to this work. However, this is not computationally efficient. The analysis chapter describes this problem in detail. In the following sections, the conceptual idea of a convolutional neural network and the kind of data it works on its base unit, and the architecture of a convolutional neural network are introduced.

The CNN is inspired by the visual cortex of a human. In the visual cortex, many neurons have a small *local receptive field*, which means that the neurons are limited to stimulation from a specific visual field region. In Figure 2.3 five neurons and their local receptive field of a house are shown. By combining the information from the features these five neurons have detected, such as lines and edges, more complex features can be constructed. CNNs use this approach and combine features from local receptive fields into more complex features. But instead of the local receptive field being limited to one part of the visual field, it is limited to one specific feature in the visual field. For example, the local receptive field of the CNN searches for the location of a car in a picture (note that a car is already a complex feature and is

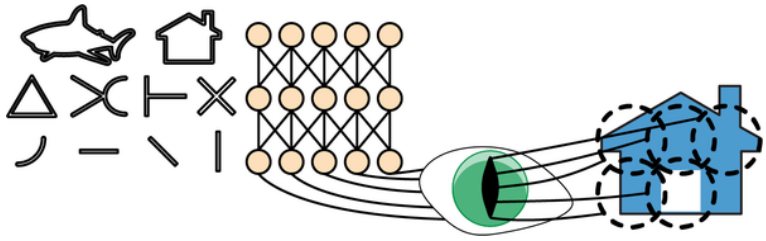

Figure 2.3: Illustration of a local receptive field and the combination of simple features to complex features [Gér22].

not the first feature the network sees). The CNN uses the mathematical operation of *convolution* to create a local receptive field, which gives the network its name [Gér22].

CNNs are mainly used in *computer vision*, which is a field of machine learning that is used for gird-like data from images and videos. It can only work on data that can be represented in a tensor. This means only *picturelike data*, with a grid symmetry, can be processed. Since CNNs are mostly used on pictures, the features a data point is represented by are usually called *pixels*.

2.3.1 The filter kernel and convolutional layer

The base unit of a convolutional neural network is called the *filter kernel* and represents the adaptation of a local receptive field. The *convolutional layer* combines multiple filter kernels into the core layer of the CNN. There are different kinds of convolutional layers depending on the dimensionality of the data. This introduction covers two-dimensional data that are represented in a matrix, such as pictures, but convolutional layers for one-dimensional and three-dimensional data follow the same principle. The two-dimensional convolutional layer is called *Conv2D* in the following and also in the programming library used in this work [Ten23].

The filter kernel consists of a *weight matrix* $\mathbf{W}$ which has the shape $(f_h \times f_w)$ with f_h being the height and f_w the width of the filter kernel in pixels and a bias b. For a filter kernel with a height of $f_h = 3$ and width of $f_w = 3$, $\mathbf{W}$ has a shape of (3×3). Thus, this filter kernel consists of ten adaptable weights. The filter kernel shown in Figure 2.4 has the same size and illustrates the calculation process. The filter kernel is placed on a region of the *input matrix* $\mathbf{X}$ with the same size as the filter kernel. Every pixel $x_{i',j'} \, \epsilon \, \mathbf{X}$ from the specific part of the input matrix

is multiplied by its overlying weight and added up to form the activation $a_{i,j}$. An activation function $\sigma(a)$ is applied on the activation to calculate the output $z_{i,j}$. The output can be represented in an *output matrix* $\mathbf{Z}$. Then the filter kernel is moved to the next section of the input data. In Figure 2.4 this is visualized with the first region colored in red and then the filter kernel is moved to the blue-colored region. The next region of the input data is defined by the *stride* or step size. The stride $\vec{S} = (s_h, s_w)$ defines how many pixels the filter kernel moves to the next part of the input data. A visualization of stride is given in Figure 2.5, where a stride of $(2, 2)$ was defined. This means the filter kernel moves 2 pixels to the right to reach the blue region. The same weights are used for every region of the input data, which is referred to as *weight sharing*. This minimizes the number of weights and makes the number of weights independent of the size of the input matrix. The calculation described above can be expressed through the following equation, with $x_{i',j'}$ being a pixel from the input data and $z_{i,j}$ being a pixel from the output of the filter kernel:

$$z_{i,j} = \sigma(a_{i,j}) = \sigma(b + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} x_{i',j'} \cdot w_{u,v}) \text{ with } \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} . \tag{2.2}$$

The output matrix $\mathbf{Z}$ of a filter kernel is called *feature map*. Because the filter kernel is seen as a local receptive field searching for a specific feature, the output data is thought of as a map of the input data that is specifically weighted for this feature. This means that the feature map also saves where this feature is detected.

The size of the output data is not necessarily the same as the input data. Usually, the size of the output data is reduced, but it depends on the size of the filter kernel and the stride chosen. In Figure 2.4 a different stride is chosen as in Figure 2.5 resulting in a different size of the output in these figures. Other techniques such as *zero padding* can be used to change the size of the output feature map. Zero padding is not applied in this work, but an introduction can be found in [Gér22].

A *convolutional layer* combines multiple filter kernels side by side. The combination of multiple filter kernels allows the convolutional layer to search for multiple features at once. Using multiple filter kernels side by side produces multiple feature maps. To handle multiple feature maps, a new third dimension is introduced, stacking the feature maps on top of each other. For example, for n feature maps of the size $(d'_h \times d'_w)$, the output of this layer has the shape $(n \times d'_h \times d'_w)$. The following convolutional layer uses this output as its input data. This implies that a convolutional layer needs to handle input data that consists of stacked feature maps or multiple picture representations. The extra dimension n is added to the filter

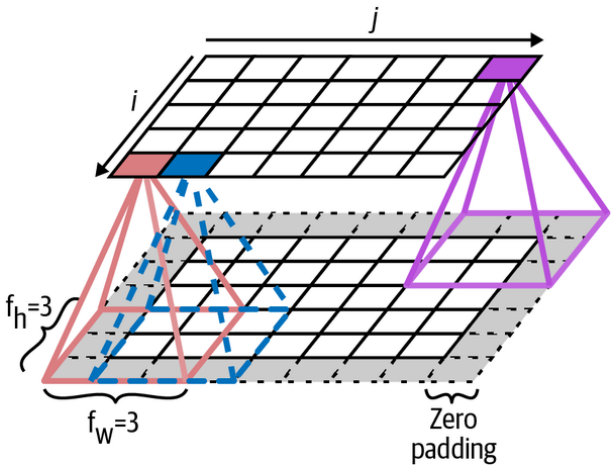

Figure 2.4: Visualization of a (3×3) filter kernel and zero padding [Gér22].

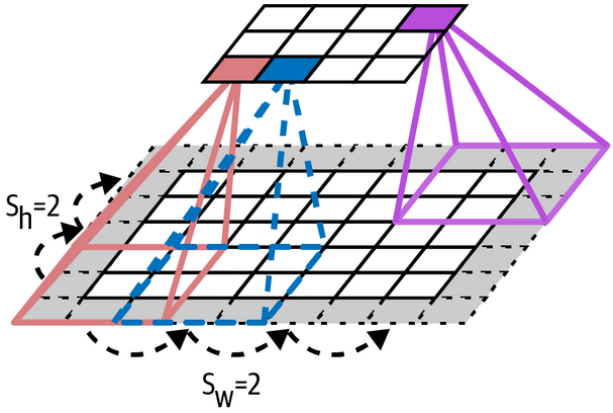

Figure 2.5: The same visualization as used in Figure 2.4 for a (3×3) filter kernel, now including a stride of $s_h = 2$, $s_w = 2$ [Gér22].

kernel as well, expanding its shape to $(n \times f_h \times f_w)$, which can now handle a stack of feature maps as input data. This means the filter kernel goes through the whole stack of feature maps or picture representations at every step. Figure 2.6 illustrates the use of multiple convolution layers on an RGB (red, green, blue) representation of a picture. By adding another dimension to the filter kernel, the amount of weights increases with the factor n: $|\mathbf{W}'| = n \cdot |\mathbf{W}|$. For a filter kernel with the shape (3×3) and 4 feature maps, $3 \cdot 3 \cdot 4 + 1 = 37$ weights are used. The weights are not shared for different feature maps, as illustrated in Figure 2.6. The input data can also be described as a stack of pictures, such as RGB pictures that have red, green, and blue pixel representations, as shown in Figure 2.6. This means that the first layer may already have to handle multiple feature maps. The initial Equation 2.2 can be expanded to handle multiple feature maps, so the filter kernel k from a convolution layer is represented by the following expression, with n being the number of feature

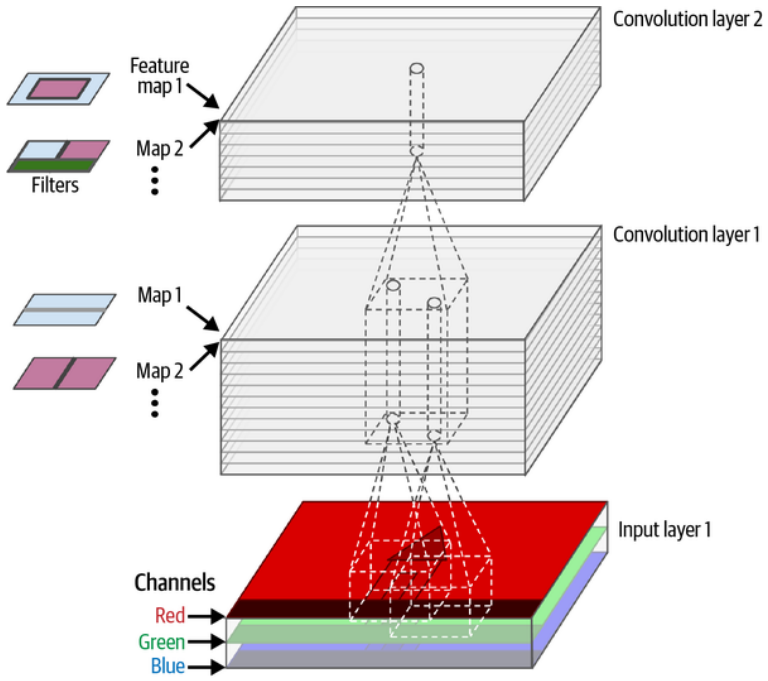

Figure 2.6: Visualization of two convolutional layers using an RGB picture as input data [Gér22].

maps in the input data and $x_{i',j',k'}$ a pixel from this stack of input data:

$$z_{i,j,k} = \sigma(b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{n-1} x_{i',j',k'} \cdot w_{u,v,k',k}) \text{ with } \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} . \quad (2.3)$$

2.3.2 Pooling

Only using convolutional layers in a network works well on small images, but for bigger images, such as (256×256) pixel pictures, many layers, and higher strides are needed to reduce the size of the output data. To solve this problem, *pooling* operations are introduced. Pooling layers depict a modified version of the filter kernel, that does not use any adaptable weights. They used *aggregation functions* to reduce the dimensionality and information of the input data. The *max*, *min*, and *average* functions are the most widely used aggregation functions. Two main types of pooling layers, *global* and *local pooling*, are distinguished.

The local pooling operation, illustrated in Figure 2.7, is very similar to a filter kernel. But instead of multiplying weights to the input pixels, an aggregation function is performed on the region of the pooling kernel. Also, pooling operations are always performed separately for each feature map (pooling through all feature

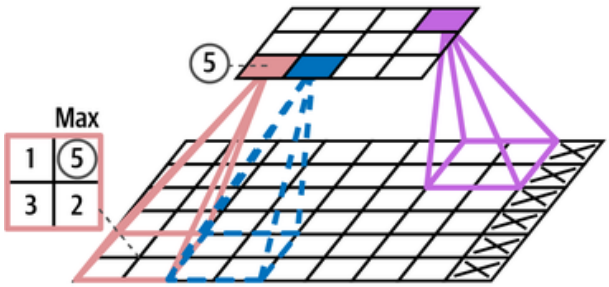

Figure 2.7: Visualization of *local max pooling* [Gér22].

maps is called *depthwise pooling*, but is not needed in this work). Usually a stride of the same size as the kernel size is used to achieve the most efficient *dimension reduction* while using all the pixels. The most commonly used local pooling layer called *MaxPooling2D* in this work and the programming library applied in this work [Ten23], uses a kernel size of (2×2), a stride of $(2, 2)$, and the max aggregation function. Equation 2.3 can be modified to describe a pooling layer, using an aggregation function $\oplus$ for the feature map k of the n feature maps:

$$z_{i,j,k} = \oplus\left(\sum_{u=0}^{f_h-1}\sum_{v=0}^{f_w-1} x_{i',j',k}\right) \text{ with } \begin{cases} i' = i \cdot s_h + u \\ j' = j \cdot s_w + v \end{cases} . \tag{2.4}$$

The difference between global pooling and local pooling is that the pooling operation is applied to the whole feature map at once. Therefore, global pooling is performed on all pixels of the feature map and not just on one region. This reduces the whole feature map to one value or a stack of feature maps into a vector. The output vector $\vec{Z} = (z_0, z_1, \cdots, z_n)$ has the same length n as the number of input feature maps. The global pooling operation for the feature map k can be summed up into an equation, with d_h and d_w being the height and width of the feature map.

$$z_k = \oplus\left(\sum_{u=0}^{d_h-1}\sum_{v=0}^{d_w-1} x_{u,v,k}\right) \quad . \tag{2.5}$$

2.3.3 Convolutional neural network architecture

A convolutional neural network does not only use convolutional and pooling layers. Convolutional layers and filter kernels operate based on the principle of identifying simple features and combining them into complex features. This process can also be described as feature extraction. However, a convolutional neural network should not only extract features but find a mapping function to perform a specific task. It was found, that using only convolutional and pooling layers is not able to sufficiently find this mapping function. To resolve this issue, dense layers are used after the convolutional layers to find a mapping function from the extracted features to the specific task. With this, a convolutional neural network can be divided into two parts. The first part uses convolutional and pooling layers to extract complex features that are the *best-dividing features* for the dataset. The second part of the network resembles a feedforward neural network, mapping the best-dividing features to the specific task. For the transition between the parts, global pooling or *flatten layers* are used. A flatten layer converts any tensor into a sequential vector.

2.4 Performance metrics

This section introduces performance metrics for classifications and regressions that are used in the next chapter. *Precision*, *recall* and *accuracy* are performance metrics used to evaluate classification models. The three metrics are introduced based on a binary classification with the labels positive and negative but are not limited to binary classifications Figure 2.8 illustrates the calculation of these metrics. The *true positive* (tp) is the number of positive data points predicted as positive data points, the same goes for the *true negative* (tn) with negative data points. *False positive* (fp) is the number of negative values classified as positive data points, the other way around is called *false negative* (fn). Accuracy is the relative number of data points that are predicted correctly. Precision measures the percentage of the number of data points that are predicted correctly into a specific class by the model to the total number of data points classified as this class, increasing the precision minimizes the number of false positive data points. Recall measures the relative number of data points that are predicted correctly by the model to the total number of data points in this class, this means it measures how many positive data points are found and identified by the model. By increasing the recall, the number of false negative data points decreases. These metrics are not independent of each other and, especially precision and recall, should be considered together. For multiple classes, these metrics are usually calculated for every class [JM23].

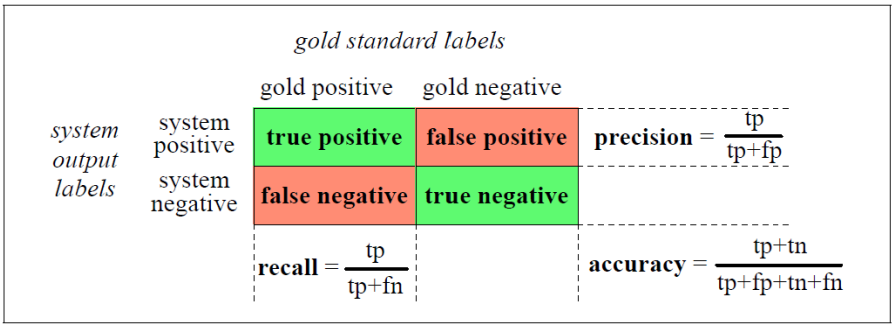

Figure 2.8: Confusion matrix for a binary classification system, with the golden labels being the actual labels of the data points [JM23].

The metric or loss function mostly used for regression tasks is the *mean-squared error* (MSE). The difference between a metric and a loss function is that loss functions are used to optimize the neural networks and metrics are used to further analyze the results and performance of the neural network. The MSE is the average squared difference between the predicted output value $\hat{y}$ and the true label y with the following equation:

$$\text{MSE}(\hat{y}) = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \quad . \tag{2.6}$$

The MSE is not equal to the variance of the predicted values, as the true label is not necessarily the same value as the mean of the predicted values. However, the MSE can be written as the variance of the predicted values with an added bias term. A more detailed explanation of this bias and its connection to the MSE can be found in [Jam17].

Chapter 3

Analysis

This chapter presents a detailed description of the analysis methods used in this work and their results. The two main goals of this analysis are the identification of the type of primary particle and the prediction of the energy of the primary particle for electron events using neural networks. To test the analysis methods, first, a simplified particle identification called binary particle classification is studied. FNNs and CNNs are used to explore all three tasks. However, the spatial hit distribution of an event is too large to be efficiently used in a FNN or CNN. For example, a three-dimensional CNN using the spatial hit distribution as a three-dimensional image takes ~ 1 minute to classify or predict 10 events on the server architecture used for this work. This means for a training dataset containing $100,000$ data points that one training epoch, one iteration of the training dataset, would take ~ 7 days. Usually, multiple epochs are required during training, making this approach impossible. To reduce the size of the data from the spatial hit distribution, the longitudinal hit distribution is introduced, which is a parameter extracted from the spatial hit distribution. In the following, the longitudinal hit distribution is studied for its usability for the three presented tasks.

3.1 Longitudinal hit distribution

In this section, the *longitudinal hit distribution (LHD)* is introduced, which is used as an input parameter for the neural network models presented in the next section. The LHD describes the number of hits per layer for all 24 layers of the EPICAL-2. With that, the number of features of the data is reduced from 25×10^6 zeros or ones to 24 values. In the following, the distinctiveness of the LHD originating from different particle types and energies is studied.

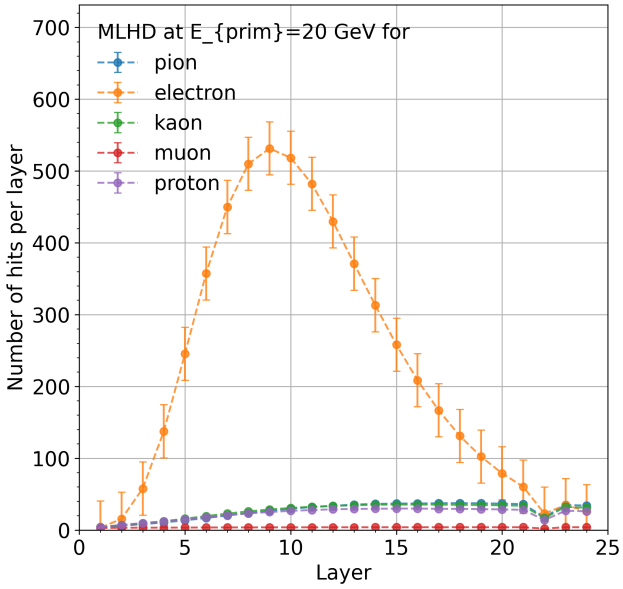

Figure 3.1: Mean LHD of pions (blue), electrons (orange), muons (green), protons (red), and kaons (purple) at $E_{prim} = 20$ GeV.

First, the LHD is studied for a fixed energy of $E_{prim} = 20$ GeV to examine the LHD of different primary particles. Figure 3.1 displays the mean LHD of electrons, pions, kaons, protons, and muons at an initial energy of $E_{prim} = 20$ GeV. The figure shows three distinctive shapes of the LHD for the different primary particles. The mean LHD of electrons displays a maximum in the lower layers and a decrease in the number of hits per layer in the higher layers. In contrast to this, the mean LHD of hadrons has no maximum but shows a steady increase in the number of hits per layer. Muons are not expected to shower in an electromagnetic calorimeter, and therefore the LHD of muons does not display any maximum or significant increase in the number of hits per layer. In the following, the LHD of hadrons, muons, and electrons are studied separately for all energies of the primary particle used in this work.

The specific types of hadrons can not be distinguished well with their LHD and therefore pions, kaons, and protons are referred to as hadrons in the following. As hadrons can also develop electromagnetic cascades, it is useful to divide the hadron events into showering hadrons (hadrons with shower, HwS) and not showering hadrons (hadrons without shower, HwoS) as their LHD varies strongly. To study this variation, Figure 3.2 shows the total number of hits per event to divide the hadrons in these two classes. In the figure, the total hit distribution of all ener-

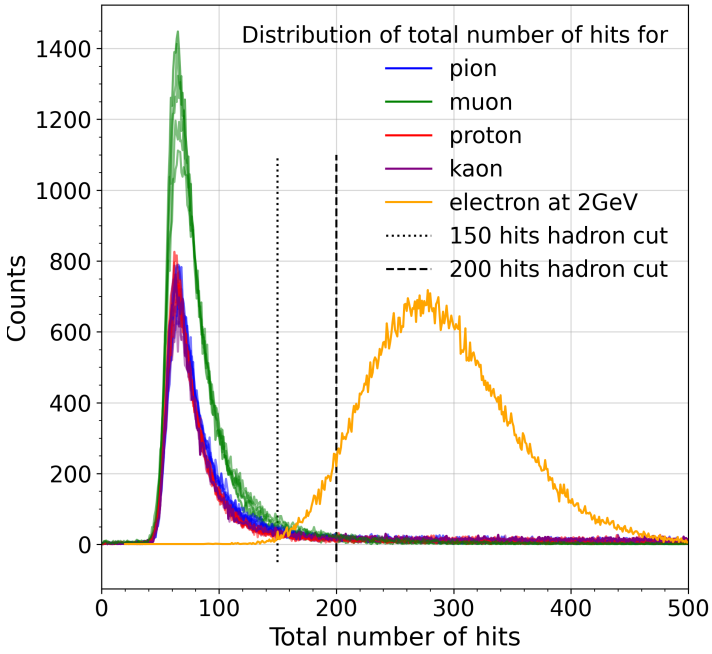

Figure 3.2: Distribution of total hits per event for pions (green), muons (yellow), protons (blue), kaons (red) with all energies and comparison to electrons (orange) at $E_{prim} = 1$ GeV. Multiple lines with the same colors are drawn for the different energies between $E_{prim} = 20$ to 80 GeV for all hadrons and muons. The black dashed and black dotted lines represent cuts made to define showering hadrons and not showering hadrons. The figure is limited to 500, but higher numbers of total hits than this limit exist.

gies of the primary particles that are studied in this work for hadrons and muons are shown in the same color and the total hit distribution of electrons at $E_{prim} = 1$ GeV is displayed for comparison. The total hit distribution of hadrons has a maximum between 50 and 100 hits, which does not show an energy dependence. Two cuts at 150 and 200 total hits are studied, which divide the hadrons into not showering hadrons with a lower number of total hits and showering hadrons with a higher number of total hits than the chosen cut. The 150 total hits cut is introduced since almost all total hits of electrons at $E_{prim} = 1$ GeV, which are considered as showers, have a number of total hits larger than the 150. On the other hand, the 200 total hits cut is chosen to include the majority of the total hit distribution of hadrons as not showering hadrons, because it is generally not expected that hadrons do shower or deposit their total in an electromagnetic calorimeter. With these two cuts, the hadrons can be separated into the two classes of showering and not showering hadrons.

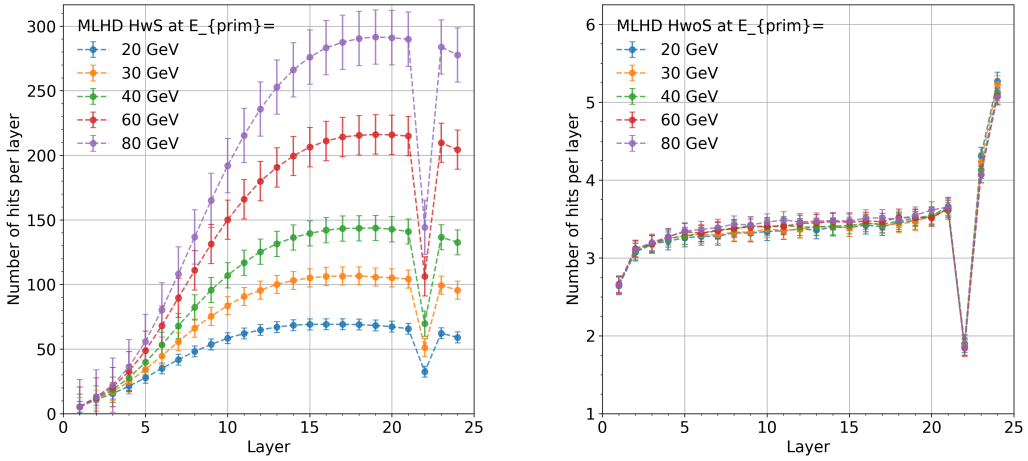

Figure 3.3: Mean LHD of showering hadrons (left) and not showering hadrons (right) with a 200 total hits cut averaged over all hadron types.

The mean LHD of hadrons after employing a total number of hits cut has different shapes for showering and not showering hadrons. Figure 3.3 shows the LHD of showering (left) and not showering hadrons (right) using the 200 total hits cut. The comparable figure using the 150 total hits cut can be found in Appendix B.2. The showering hadrons (right) show an increase in the number of hits per layer as a function of the layers, also a maximum is seen in the range of layers 15 to 21. A strong energy dependence is observed as well. The not showering hadrons (left) display only a small increase in the number of hits per layer as a function of the layers, and no energy dependence is found. The figure only illustrates the mean LHD averaged of all types of hadrons. The mean LHD for pions, kaons and protons can be found in Appendix B.1 using the 200 total hits cut and in Appendix B.3 using the 150 total hits cut as a small difference in the magnitude of the mean LHD of showering hadrons of the different hadron types for the same energy of the primary particle is seen. Nevertheless, the showering hadrons class is not divided into its specific hadron type. The mean LHD of showering hadrons shows an energy dependence, which makes the prediction of the energy of the primary particle on the LHD possible. However, as the hadrons do not deposit all their energy in the EPICAL-2, the usefulness and validation of this prediction are questionable.

Figure 3.4 (left) shows the mean LHD of muons, which has a similar shape as the mean LHD of the not showering hadrons. The muon LHD displays only a small energy dependence, with a maximal difference of ~ 1.5 hits per layer between 20 and 80 GeV. Comparing this to the energy dependence of the showering hadrons, the energy dependence of the muons is negligible. As the muons and not showering hadrons

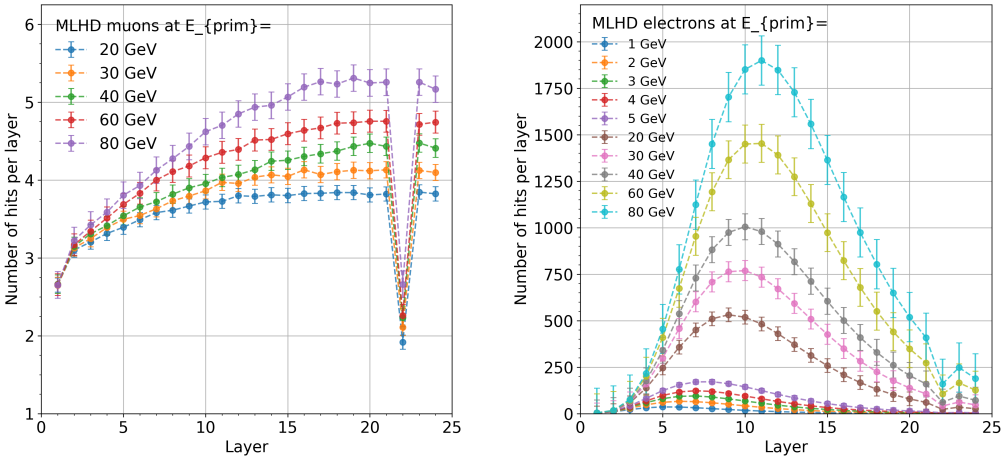

Figure 3.4: Mean LHD of muons (left) and electrons (right) of different energies.

have a similar shape, they are treated as one class in the particle identification task.

Figure 3.4 (right) shows the mean LHD of electrons. It displays a strong energy dependence, which allows the electron energy prediction task to work on the LHD. The mean LHD of electrons of all energies has a maximum in the lower layers and a decrease in the number of hits per layer in the higher layers. The maximum of the mean LHD moves to higher layers and higher magnitude or total hits with increasing energy. Figure 3.5 illustrates this with the relative distribution of the maximum of the LHD of electrons for the different energies of the primary particle. The figure also shows that the maximum is not equal in all events of one energy but follows the distribution in the figure. Therefore, it is expected that the LHD of an individual electron event does not necessarily have the same maximum position as the mean LHD of electrons.

In conclusion, the LHD is a useful parameter for neural networks to predict the particle type, or rather decide between electrons, showering hadrons, and non-showering hadrons/muons. Also, the prediction of the electron energy is possible through the strong energy dependence of the LHD. Another parameter that can be of interest for future studies is the lateral shower profile, as it is expected to differ for hadronic and electromagnetic showers.

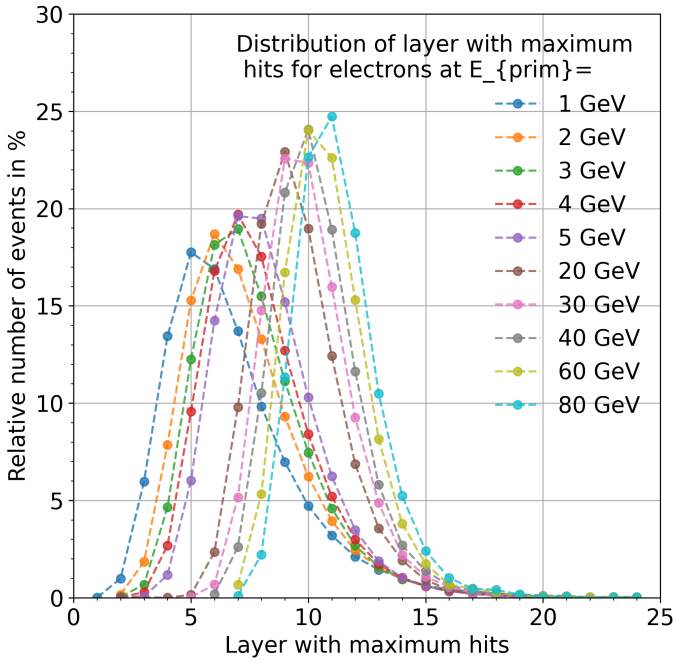

Figure 3.5: Probability distribution of the maximum layer of the LHD for electrons.

3.2 Binary particle classification

First, the simple binary particle classification is studied. The goal is to classify the primary particle of an event either as a pion or an electron with an initial energy of $E_{prim} = 20$ GeV. By reducing the types of primary particles and energies of the primary particle, the LHD as a parameter for the neural networks and the neural network architectures can be tested in a more explainable environment than in the full particle identification.

The dataset for this task consists of $40,000$ events, with $20,000$ events for pions and electrons each. It is split into the training dataset containing $24,000$ events, the validation dataset with $8,000$ events, and the test dataset holding $8,000$ events. The datasets contain an equal amount of events from both classes. The LHD is extracted for all events and used as input for the neural network, the mean longitudinal distribution of the two classes is shown in Figure 3.1.

Three neural networks are compared for the binary particle classification: two FNNs and one CNN. As the layer with the maximum of the LHD varies over individual events for the LHD of electrons and showering hadrons, as seen in Figure 3.5 for the electrons, a pure FNN approach that is directly connected to the LHD of every layer

may not be the most efficient. For this reason, a CNN is chosen as a comparison as it does not directly connect to the LHD of every layer, but searches for features in the LHD. One network uses a CNN before the FNN, and two FNNs are trained to study this hypothesis. The network using a CNN is named *CNN1*, and the model only using a FNN with the same number of parameters, meaning weights and biases, as CNN1 is named *FNN1*. The third model is a FNN named *FNN2* with fewer parameters than the other models to study if a smaller number of parameters is sufficient to solve the binary particle classification task. To transition from the CNN to the FNN in the CNN1, a one-dimensional global average pooling layer is applied. The input layer of all networks uses the 24 values of the LHD, and the output layer uses two neurons with a softmax function, representing the electron and pion. All other hidden layers use the GELU activation function. The model is compiled with the NAdam optimizer, sparse categorical cross-entropy as a loss function, and early stopping. The metrics calculated are the accuracy, precision, and recall. A detailed description of these functions can be found in the program libraries documentation, such as [Ten23]. The full network architecture can be found in Appendix C.1. No optimization was performed on the model architecture, but could be considered for further studies and optimization.

All models show an overall good performance on the binary particle classification task and have similar performance metric values. The evaluated metrics of precision, and recall introduced in Section 2.4 for the predictions of the test dataset are listed in Table 3.1. The class accuracy is calculated differently than the introduced accuracy by dividing the number of correctly classified events by the total number of events from this class. The CNN1 shows the best electron accuracy and the FNN1 has the best pion accuracy, but all networks only have small differences in these metrics. To further study the precision and recall the absolute values of true positive, false positive, etc. are illustrated in the confusion matrix for every model in Figure 3.6. The confusion matrix of CNN1 shows that two electrons were wrongly classified as pions and 22 pions were identified as electrons. The other two models both have 4 misclassified electrons and a lower number of falsely identified pions than the CNN1. Figure 3.7 illustrates the LHD of the misidentified events of the CNN1 in comparison to the mean LHD of electrons, and pions. Similar figures for the FNN1 and FNN2 can be found in Appendix C.3. The wrongly classified pions show a significantly different LHD than the mean pion distribution and rather follow the mean distribution of electrons. This indicates that the wrongly classified pion events shower in the EPICAL-2, which further motivates the division of hadrons into showering and not showering events to improve the model performance on the LHD. In contrast

metrics in %	CNN1	FNN1	FNN2
el accuracy	99.95	99.90	99.90
el precision	99.45	99.60	99.48
el recall	99.95	99.90	99.90
pi accuracy	99.45	99.60	99.48
pi precision	99.95	99.90	99.90
pi recall	99.45	99.60	99.48

Table 3.1: Comparison of test performance metrics in the binary particle classification task.

to the FNN models, the CNN1 was able to correctly classify electron events that shower in higher layers and that were wrongly identified by the FNN1 and FNN2, shown in Appendix C.3. Even with the wrongly classified electron event that starts showering in the layer 13, the CNN1 is more robust against varying shower starts than the FNN models.

The good performance of the models shows that the LHD is sufficient as an input parameter for the studied models for the binary particle classification task. Compared to the FNN models, the CNN1 is more robust to varying shower starts of the LHD of electrons. For future studies, the lateral hit distribution can be added to the LHD as an input parameter and an optimization of the network architecture can be performed to further increase the performance of the models. As these models show good performance on this simple task, they present a starting point for particle identification and electron energy regression.

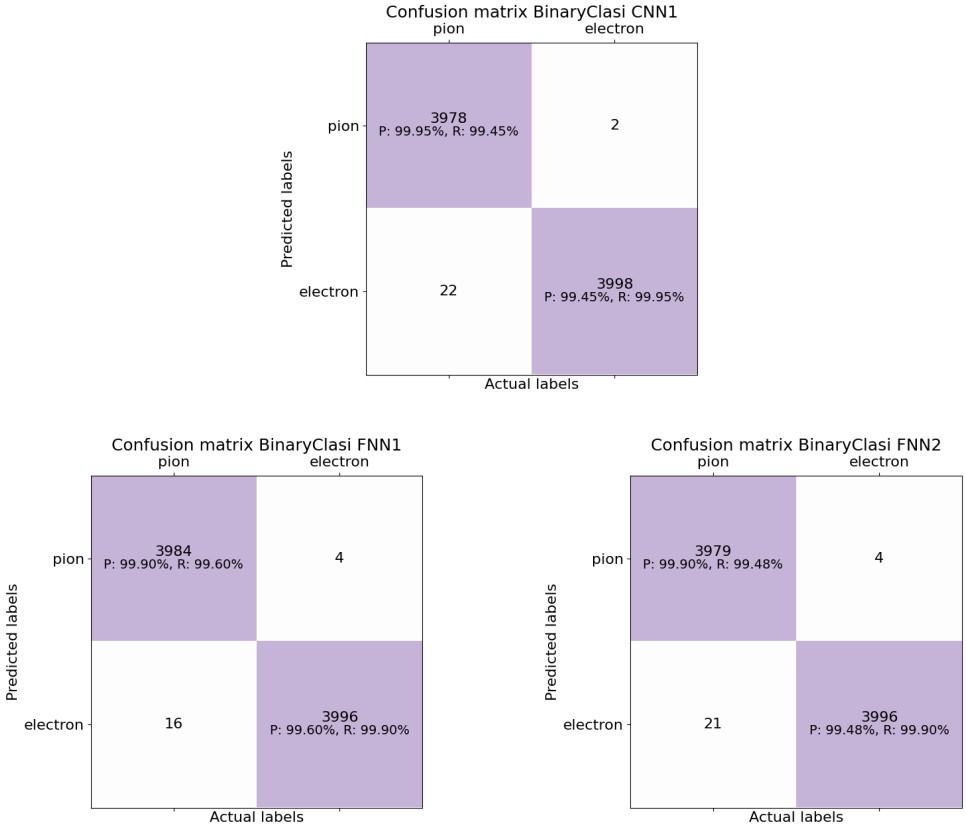

Figure 3.6: Confusion matrix with precision and recall of the CNN1 (top), FNN1 (bottom left) and FNN2 (bottom right) model.

3.3 Particle identification

As the simple binary particle classification task showed a good performance, the task can be broadened to include more primary particle types and energies. In this task, the type of the primary shower particle should be identified from the full provided dataset introduced in Section 1.2. The main target is to classify electron events correctly, to later derive their energy from the shower shape. This is not possible for hadrons and muons, as not all their energy is deposited in the EPICAL-2.

It is useful to divide the dataset into three classes for the particle identification: electrons, hadrons with shower, and hadrons without shower or muons. This separation is done to get better-dividing shapes of the LHD, which is studied in the previous Section 3.1. As it is also the main target to identify electrons, an extensive distinction between the different hadrons is not necessary. Here, models using the 150 and 200 total hits hadron cut are compared. The dataset includes $1,646,413$ events, that are unequally distributed over the pions, kaons, protons, electrons, muons, and their energies. The exact number of events per type of primary particle

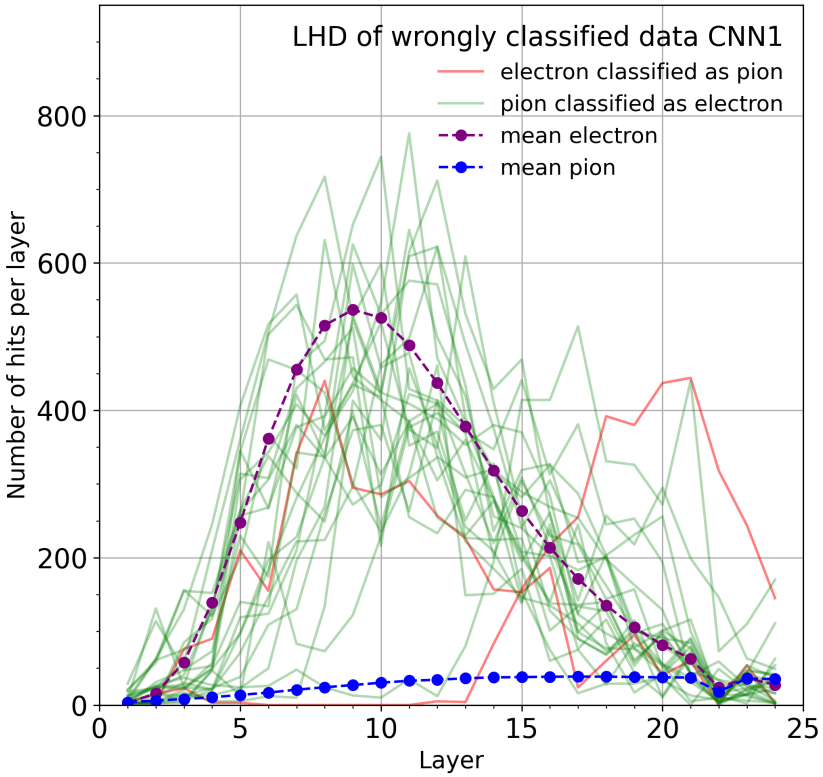

Figure 3.7: LHD of wrongly classified events by the CNN1 and for comparison the mean LHD for electrons and pions at $E_{prim} = 20$ GeV.

and energy of the primary particle can be found in Appendix A. The number of events per type of hadron and its energy also depends on the cut that divides the hadron events into showering hadrons and not showering hadrons. For example, the 200 hits hadron cut creates 640,000 events of electrons, 368,083 events of showering hadrons, and 638,330 events of non-showering hadrons and muons. To counteract this unequal distribution, class balance weights are given to each of the three classes and the different energies. For i being a combination of one of the three classes and one energy, the class balance weights for all events in i are calculated with the following formula:

$$\omega_i = \frac{0.25 \cdot N_{tot}}{N_i} \tag{3.1}$$

with N_i being the number of events in i and N_{tot} the total number of events. $0.25 \cdot N_{tot}$ was chosen as a normalization to get class balance weights of a similar magnitude and greater than 0.25 and to shift the range of the class balance weights to smaller values. These class balance weights are used during the training of the model to weigh the training loss and ensure all classes and energies are learned equally well.

Eight different models are trained for the 150 and 200 total hits cut for hadrons, either following the FNN1 or CNN1 structure and for each a model using the class balance weights and one without class balance weights. All models use the LHD as input and thus have a 24 neuron input layer. They use GELU as activation function, and they have a three-neuron output layer using the softmax activation function. The CNN models have a significantly higher number of parameters than the FNN models. The full model architecture is given in Appendix D.1.

All models show a similar performance on the task. In Figure 3.8 the categorical cross-entropy of the test dataset is plotted for the eight models and each class separately. For the electrons, a trend can be seen that the models with class balance weights have a lower test loss than the models without class balance weights, and the models using the 200 total hits cut outperform models with the 150 total hits cut. In the test loss of the non-showering hadrons and muons, the models using the 200 total hits cut also perform better. However, models with class balance weights show a different trend, the models using the class balance weights have a higher test loss than the models without class balance weights. No significant trend is found for the showering hadrons. In Table 3.2 more test metrics are compared. No model outperforms the other models in all metrics. However, the CNN models show a better performance for the electron class, and the models using the 200 total hits cut tend to have better accuracies than the models using the 150 total hits cut. The CNN4 has the highest total accuracy of 97.96%, nevertheless, all models have a total accuracy between 97% and 98%. The confusion matrix for every model can be found in Appendix D.2.

In general, the performance of all models is similar. Models using the 200 total hits cut and class balance weights mostly have higher performance metrics than models without. Nevertheless, a general trend could not be observed. To further improve this task the model architecture, the hadron cut, and the class balance weights could be optimized.

metrics in %	CNN2	CNN3	CNN4	CNN5	FNN3	FNN4	FNN5	FNN6
total hits cut	150	150	200	200	150	150	200	200
class bal. weights	yes	no	yes	no	yes	no	yes	no
accuracy	97.38	97.42	97.96	97.88	97.29	97.35	97.87	97.83
El accuracy	96.29	96.22	96.58	96.33	95.92	96.04	96.12	96.15
El precision	97.46	97.81	97.32	97.38	97.73	97.69	97.69	97.53
El recall	98.77	98.34	99.22	98.89	98.11	98.27	98.36	98.55
HwS accuracy	89.31	89.52	91.14	90.75	89.06	89.27	90.94	90.65
HwS precision	94.04	93.71	95.89	96.12	93.30	93.42	95.13	95.60
HwS recall	94.68	95.25	94.85	94.21	95.14	95.27	95.37	94.59
HwoS accuracy	97.00	97.16	98.23	98.30	97.14	97.14	98.41	98.34
HwoS precision	99.38	99.35	99.78	99.36	99.35	99.45	99.62	99.39
HwoS recall	97.59	97.79	98.44	98.93	97.76	97.66	98.78	98.94

Table 3.2: Comparison of test metrics for electrons (El), hadrons with shower (HwS), and hadrons without shower/muons (HwoS). The best (darker) and second-best (lighter) models of a metric are highlighted.

3.4 Electron energy regression

The previously discussed models can only predict the type of primary particle initiating the particle shower. However, also the energy of the primary particle can be determined from the spatial hit distribution. The classical approach of calculating the energy of the primary particle is introduced in Section 1.2. This section explores the machine learning approaches for the prediction of the energy of electron events. The energy prediction can only be performed for electrons, as they are the only particles included in this works' dataset that deposit all their energy in the EPICAL-2.

The dataset for this task consists of electron events with different discrete energies from $E_{prim} = 1$ to 80 GeV. As the dataset only includes discrete energy values, the particle identification task could be interpreted as a classification, where an electron event is classified as one of the given energies. However, real measurements usually include continuous energy values instead of discrete energy classes. Therefore, a regression task is more useful for further applications that do not use discrete energy values. However, using discrete energy classes as training data for a regression task can result in unique problems, which are not found in a regression task trained with continuous values or in a classification task with class values. This will be discussed later in this section in more detail. The number of events per energy class is unequally distributed in the dataset for the particle identification class. For this, the class balance weights of Equation 3.1 are used again. A modified version of the FNN and CNN used in the previous task are implemented here. They have a

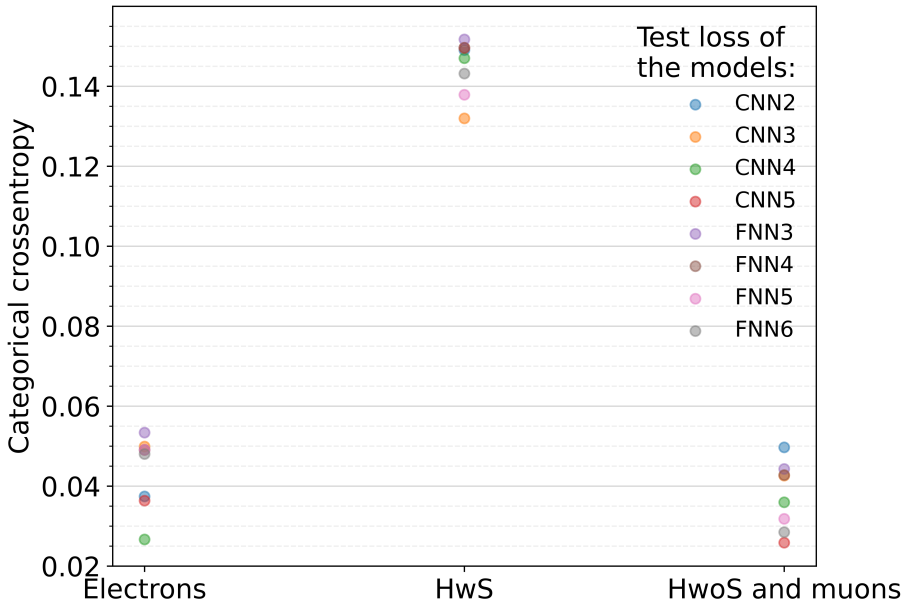

Figure 3.8: Comparison of the test loss of all models for the electrons, showering hadrons (HwS) and not showering hadrons/muons (HwoS and muons) class.

similar architecture as Appendix D.1, but their output layer consists of one neuron using the ReLU activation function. All models use the LHD of an event as input.

In the following, five metrics are studied to evaluate the model's performance on the test dataset. Figure 3.9 shows the mean squared error (MSE) calculated for each energy separately. The higher energies have a larger MSE than the lower energies. This is expected since the distances between the training energy classes are smaller at low energies. The model using the CNN with energy balance weights seems to perform better than the other models, which perform similarly. Figure 3.10 shows the difference between the predicted and true energy values. Also in this metric, the model using the CNN with energy balance weights has a smaller difference between predicted and true energies as compared to the other models. Another prominent feature illustrated in Figure 3.10 is the anti-symmetry of positive and negative differences. The model using the CNN with energy balance weights shows more frequent and larger positive than negative difference values, e.g. -2.5 GeV has ~ 10 counts compared to 2.5 GeV with ~ 50 counts. However, the other three models show a different trend with slightly more frequent and larger negative than positive values, e.g. -5 GeV has ~ 20 counts compared to 5 GeV with ~ 10 counts. To further investigate the antisymmetric behavior, Figure E.2 illustrates the distri-

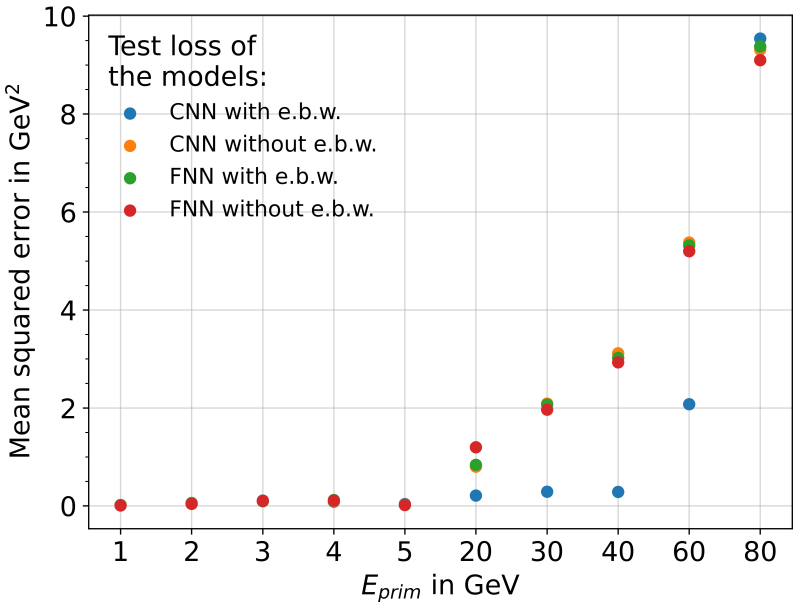

Figure 3.9: Comparison of the test loss (mean squared error) for all energies considered in this work and all models, either using the energy balance weights (e.b.w.) or not.

bution of the predicted energies for each energy class separately for the model using the FNN without energy balance weights. The comparable figures for the other models can be found in Appendix E. The distributions of the predicted energies of the higher energies $E_{prim} = 20, 30, 40, 60, 80$ GeV in Figure E.2 mostly follow a Gaussian trend. However, the distributions of the predicted energies of the lower energies $E_{prim} = 1, 2, 3, 4, 5$ GeV display unique shapes that differ from the expected Gaussian distribution. In all models, no predicted energy values below 1 GeV are observed, making 1 GeV the minimum of the range of the output values of all models. A possible explanation for this is the fact that no energies below this minimum are used during training. In contrast to this, no maximum energy is observed at $E_{prim} = 80$ GeV, which is the highest energy class included in the training dataset. Further research is needed to understand the cause of this seemingly contradicting behavior. The distributions of the predicted energies of $E_{prim} = 2, 3, 4, 5$ GeV could be explained by the different step sizes between the energy classes. For example, as 5 GeV is closer to 4 GeV than 20 GeV predictions tend to be more under than above $E_{prim} = 5$ GeV because this region is more explored than the region between 5 and 20 GeV. The distributions of the predicted energies of higher E_{prim} of other models can be found in Appendix E, which show antisymmetric differences from the expected Gaussian distribution could also be explained by the different step sizes. In further studies, models using the same energy class step size could be used to

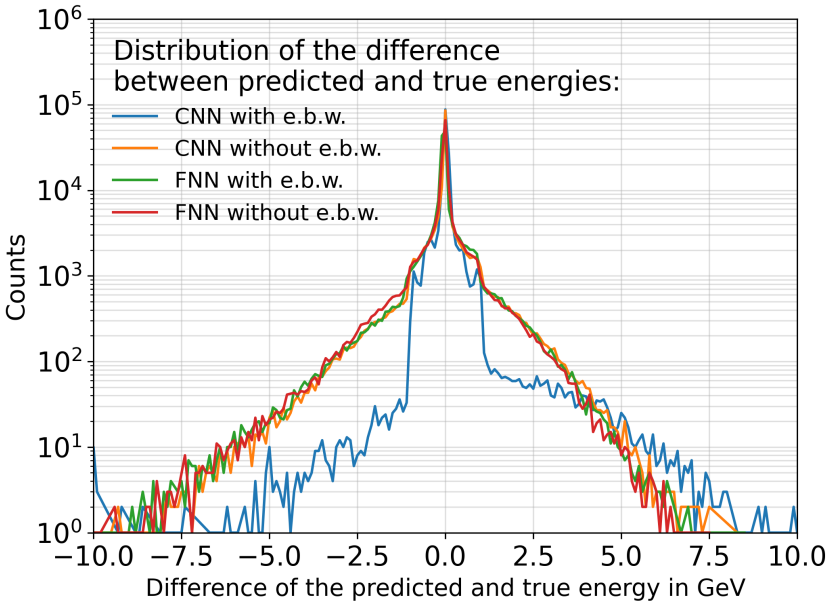

Figure 3.10: Comparison of the difference between the predicted and true ($pred - true$) energy for all models, either using the energy balance weights (e.b.w.) or not, including all energies considered in this thesis.

investigate the hypothesis of the importance of equal step sizes or a dataset containing continuous energy values could be used in training to study the impact of the discrete energies on the regression task.

Even though the distributions of the predicted energies are not all Gaussian, the mean and standard deviation of the distributions of the predicted energies are used to calculate the energy resolution and linearity introduced in Section 2.4. The linearity and energy resolution of the classical approach using the total number of hits to estimate the energy of the primary particle is shown in Figure 1.7. The energy resolution and linearity of the models can not be considered fully comparable to the energy resolution and linearity of the classical approach, because not all distributions of the predicted energies are Gaussian distributed. Figure 3.12 shows the linearity of the mean predicted energy of the models and the true energy (left). The mean energy prediction is fitted with the linear function $y = m \cdot x$ to the true energy for every model separately. A perfect model would deliver a gradient of $m = 1$. All models besides the model using the CNN with energy balance weights have a gradient lower than 1. This is consistent with the antisymmetric distributions shown in Figure 3.10. The relative difference between the mean of the predicted energies and the linear fit illustrated in Figure 3.12 (right) has larger differences for lower energies. This is caused by the antisymmetric and non-Gaussian distributions of

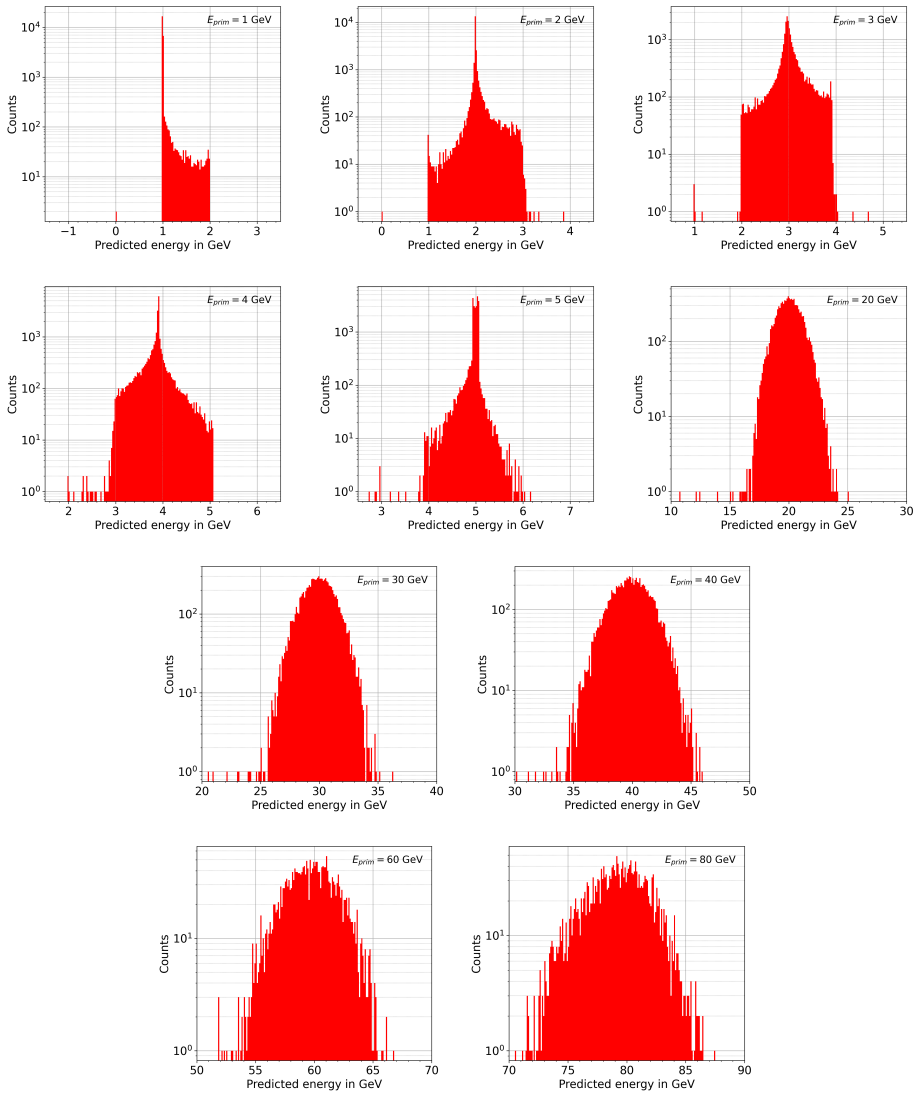

Figure 3.11: Distributions of the predicted energies of the energies E_{prim} =1, 2, 3, 4, 5, 20, 30, 40, 60, and 80 GeV for the model using FNN without weights. The x-axis shows the predicted energy and the y-axis is the counted amount.

the predicted energies of the lower E_{prim}, which make it impossible that the mean is equal to the true energy. In comparison to the ratio of the linearity of the total number of hits in Figure 1.7, the mean predicted ratios of the models perform worse.

Figure 3.13 shows the models energy resolution. The energy resolution describes the spread and accuracy of the energy prediction of the models. The energy resolution of every model shows better or similar performance than the energy resolution using the classical approach, but this should be considered with care as not all model energy prediction distributions follow a Gaussian distribution, as opposed to the distributions used to calculate the energy resolution of the classical approach. However, as the higher energy values mostly show a Gaussian distribution, the model resolu-

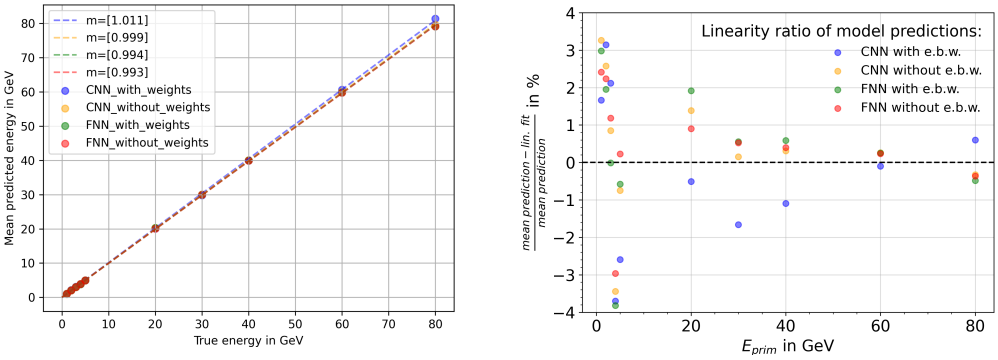

Figure 3.12: Model linearity of the mean energy predictions and the true energy with linear fits $y = m \cdot x$ and their gradient (left), the ratio of the relative difference of the mean energy prediction and the value of the linear fit (right).

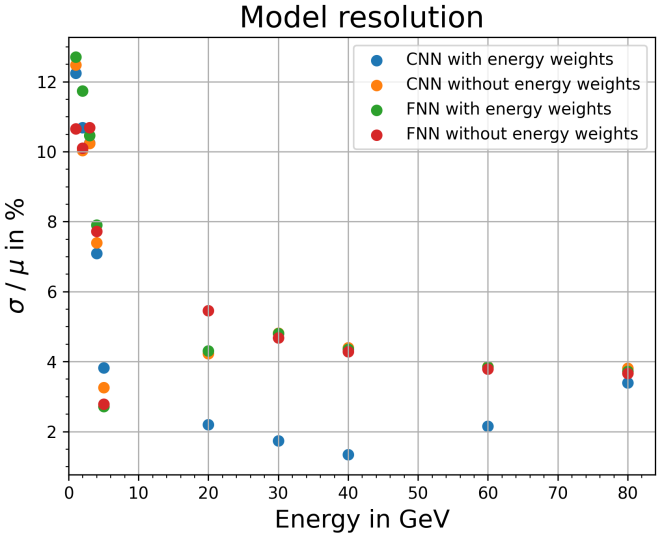

Figure 3.13: Energy resolution of all models.

tion is comparable for these energies. For the higher energies, the models show equal values of the energy resolution to the energy resolution of the classical approach. Only the model using the CNN with energy balance weights has lower values of the energy resolution than the classical approach. However, the validity of this result requires further studies because the distributions of the predicted energies of the higher E_{prim} for this model are generally not Gaussian distributions. A better model energy resolution implies that the models can reduce the error that is found for the estimation of the energy of the primary particle that only uses the total number of hits. Also instead of the $\frac{1}{\sqrt{E}}$ dependence found in the detector hits energy resolution, the model resolution is expected to follow a $\frac{1}{E}$ distribution, as the mean energy prediction should be $\mu \sim E$ resulting in $\frac{\sigma}{\mu} \sim \frac{1}{E}$.

In conclusion, it is found that the energy prediction of electron events with neural networks is possible, and its energy resolution is equal to or better than the energy resolution of the classical approach based on the total number of hits. However, the usage of energy classes with unequal step sizes for the regression task may induce the non-Gaussian distributions of the energy prediction seen in this work. Because of this, the energy resolution of the models and the classical approach are not comparable for all energies and all models. In further studies, this issue can be studied, by using equal steps for energy classes, e.g. only E_{prim} = 20, 40, 60, and 80 GeV, or with a new dataset using continuous energy values.

Chapter 4

Summary and outlook

In this thesis, particle identification and the prediction of the energy of electron events with neural networks based on the simulated EPICAL-2 detector response are explored. The detector response of the EPICAL-2 to a particle shower can be represented as a three-dimensional spatial hit distribution. However, this spatial hit distribution is a large data format as it encompasses the information of $\sim 25 \times 10^6$ pixels. To efficiently explore the use of neural networks, the longitudinal hit distribution is extracted from the spatial hit distribution to use as input to the neural networks. The longitudinal hit distribution is observed to divide three classes for the particle identification and separate the different energies of the electron events well. This makes the longitudinal hit distribution a good parameter for the explored tasks.

The particle identification is done on the basis of the longitudinal hit distribution dividing the three classes: electrons, showering hadrons, and not showering hadrons/muons. Different feedforward and convolutional neural networks are studied and compared for this task. The best model has a 97.96% accuracy, which describes the relative number of correctly classified events.

The electron energy regression is also based on the longitudinal hit distribution and uses different feedforward and convolutional neural networks to predict the energy of an electron event. The distributions of the predicted electron energy of the models show unexpected non-Gaussian distributions for some energies. This could be caused by the training dataset, which contains events from discrete energy values with varying step sizes. Also using discrete energy values for a regression task that predicts continuous values could play a role. The distributions of the predicted electron energy that show a Gaussian distribution perform equally well as the classical energy estimator using the total number of hits to predict the energy of electron events. Further research is needed to understand the cause of the distributions of

the predicted energies.

In this thesis, all models use the longitudinal hit distribution as input. However, there are other parameters that can hold additional information about the particle shower that could be included in the input of the models to increase their performance. An additional parameter could be the lateral hit distribution or lateral shower profile of every layer, studied in [Alm+23].

In this analysis, only feedforward and convolutional neural networks are used. However, there are more types of neural networks that can be explored. In further research, emphasis should be placed on neural networks based on graphs. The spatial hit distribution measured by the EPICAL-2 can be represented as a graph, which greatly reduces the size of the data format. The graph representation of the spatial hit distribution has an expected size between 3×10^2 to 3×10^4 hit features depending on the event, in contrast to the 25×10^6 pixel features of the spatial hit distribution for every event. Neural networks working on graphs for the explored tasks are graph neural networks and graph transformers. In detail, the dynamic graph convolutional neural networks (DGCNN) [Wan+19], the GarNet, and the GravNet [Qas+19] are candidates for the graph data of EPICAL-2.

Exploring graph neural networks and graph transformers opens up other possible tasks. In this work, it is assumed that all events contain only one primary particle, but this is not always the case. Graph-based neural networks should be able to handle tasks with multiple primary particles, such as energy prediction of each of the multiple primary particles. Another potential task is the simulation of particle showers with neural networks. Possible candidates for generative models that are able to simulate the spatial hit distributions of particle showers are generative adversarial networks (GAN) [Goo+14] in combination with graph transformers or the graph neural network, introduced above. More possible candidates of generative models can be found in [HEP], which are explored for other tasks and detectors but could potentially be applied to the EPICAL-2 as well.

Acknowledgements

I would like to express my deepest gratitude to Prof. Dr. Henner Büsching for giving me the opportunity to write this thesis and for his kind guidance during my academic journey. I am furthermore very grateful to Prof. Dr. Harald Appelshäuser for agreeing to examine this thesis. I want to express my gratitude to Mario Krüger and Tim Rogoschinski for guiding me in writing my thesis and developing the analysis. Furthermore, I would like to thank the EPICAL-2 group and especially Thomas Peitzmann and Nigel Watson for their advice and expertise. I want to express my gratitude to the Frankfurt working group for welcoming me, and for their advice and expertise. Finally, I want to thank my family and friends for their support throughout my studies.

Bibliography

[Alm+23] J. Alme et al. "Performance of the electromagnetic pixel calorimeter prototype Epical-2". In: *Journal of Instrumentation* 18.01 (Jan. 2023), P01038. DOI: 10.1088/1748-0221/18/01/p01038. URL: https://doi.org/10.1088%2F1748-0221%2F18%2F01%2Fp01038.

[Col23] ALICE Collaboration. *ALICE: FOCAL*. 2023. URL: https://alice-collaboration.web.cern.ch/menu_proj_items/FOCAL. (accessed: 07.09.2023).

[Fra11] Gian Franco Dalla Betta. *Advances in Photodiodes*. IntechOpen, 2011.

[Gér22] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and Tensor Flow*. 3rd ed. O'Reilly Media, Inc., 2022.

[Goo+14] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[HEP] HEP ML Community. *A Living Review of Machine Learning for Particle Physics*. URL: https://iml-wg.github.io/HEPML-LivingReview/. (accessed: 22.09.2023).

[Jam17] Gareth James. *An Introduction to Statistical Learning with Applications in R*. 8th ed. Springer, 2017.

[JM23] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd ed. unpublished draft, Aug. 2023. URL: https://web.stanford.edu/~jurafsky/slp3/.

[KW20] Hermann Kolanski and Norbert Wermes. *Particle Detectors*. Oxford university press, 2020.

[Qas+19] Shah Rukh Qasim et al. "Learning representations of irregular particle-detector geometry with distance-weighted graph networks". In: *The European Physical Journal C* 79.7 (July 2019). DOI: 10.1140/epjc/s10052-019-7113-9. URL: https://doi.org/10.1140%2Fepjc%2Fs10052-019-7113-9.

[Rog] Tim Rogoschinski. personal communication.

[Sch] Jan Schöngarth. personal communication.

[Sch22] Jan Schöngarth. "Simulation der Detektorantwort von MAPS auf einzelne Schauerteilchen". Bachelor Thesis. Goethe Universität Frankfurt, 2022.

[Ten23] Tensorflow. *Keras Documentation*. 2023. URL: `https://keras.io/`. (accessed: 31.07.2023).

[Wan+19] Yue Wang et al. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: `1801.07829 [cs.CV]`.

Appendix A

Dataset parameters

kind of particle	energy in GeV	N
electron	1	100,000
	2	100,000
	3	100,000
	4	100,000
	5	100,000
	20	40,000
	30	40,000
	40	40,000
	60	10,000
	80	10,000
muon	20	49,990
	30	49,992
	40	49,986
	60	49,985
	80	49,979
pion	20	49,999
	30	49,997
	40	49,993
	60	49,989
	80	49,988

kind of particle	energy in GeV	N
kaon	20	49,994
	30	49,995
	40	49,986
	60	49,990
	80	49,984
proton	20	49,996
	30	49,995
	40	49,992
	60	49,991
	80	49,987

Table A.1: Detailed description of the dataset used in this work. N represents the number of events for this class.

Appendix B

Longitudinal hit distribution of hadrons

B.1 Pions, kaons, and protons with 200 hits cut

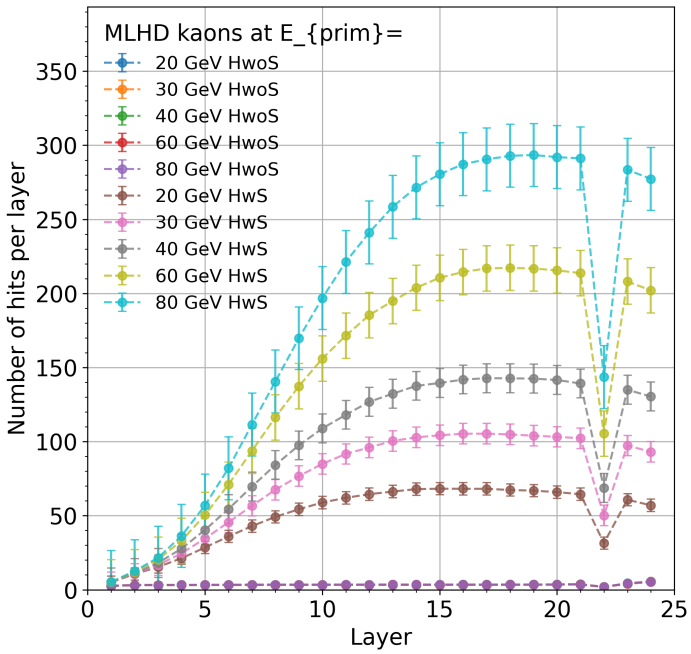

Figure B.1: Mean LHD of kaons using the 200 total hits cut.

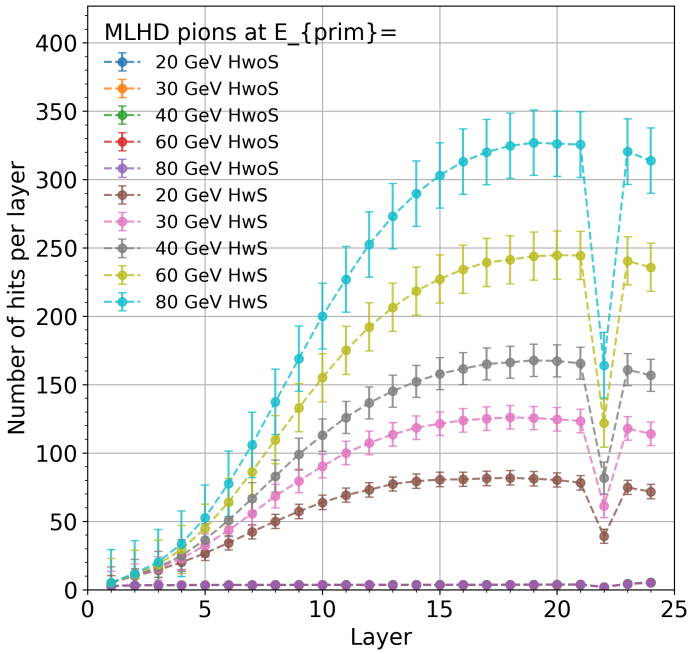

Figure B.2: Mean LHD of pions using the 200 total hits cut.

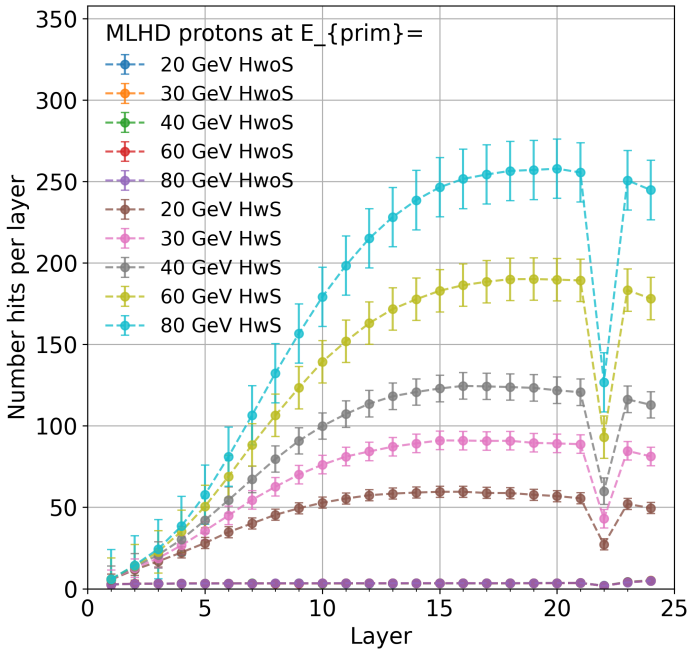

Figure B.3: Mean LHD of protons using the 200 total hits cut.

B.2 Hadrons with 150 hits cut

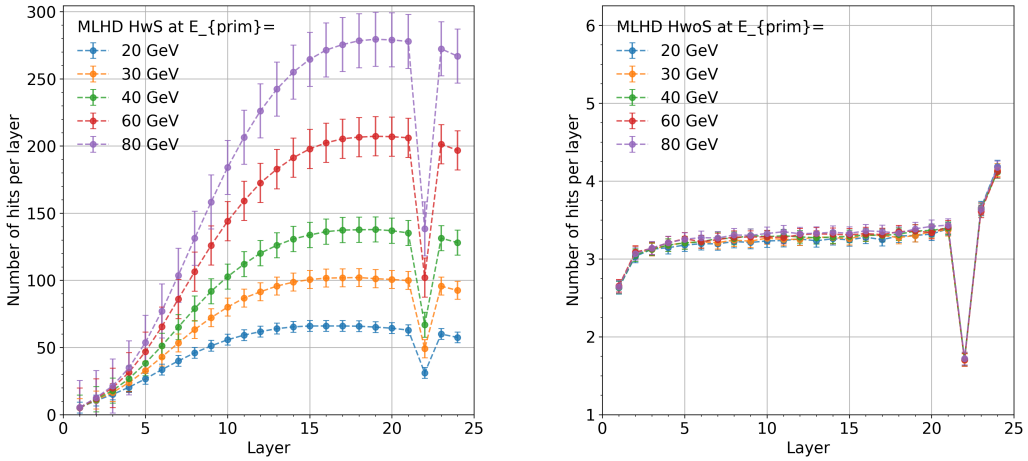

Figure B.4: Mean LHD of showering hadrons (left) and not showering hadrons (right) with a 150 total hits cut average over all hadron types.

B.3 Pions, kaons, and protons with 150 hits cut

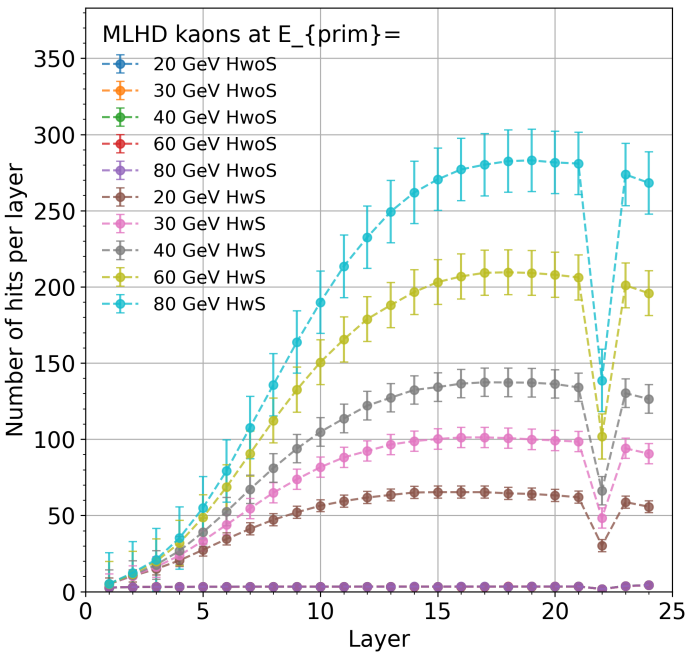

Figure B.5: Mean LHD of kaons using the 150 total hits cut.

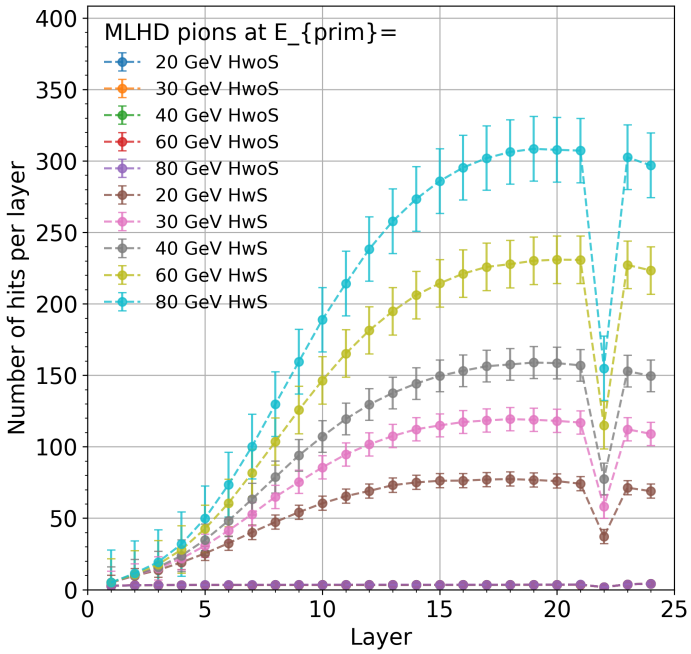

Figure B.6: Mean LHD of pions using the 150 total hits cut.

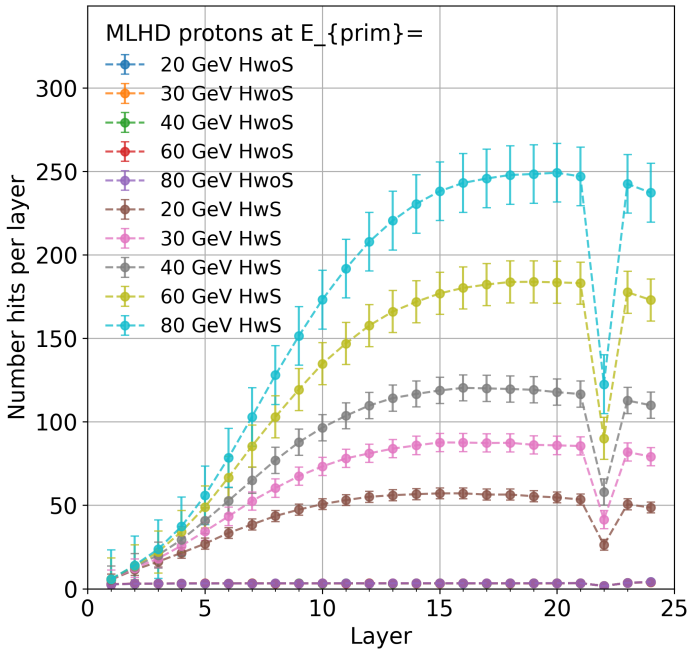

Figure B.7: Mean LHD of protons using the 150 total hits cut.

Appendix C

Binary particle classification

C.1 Model architectures

```
Model: "Binary particle classification CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 24, 1)]           0
_________________________________________________________________
conv1d (Conv1D)              (None, 22, 32)            128
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 20, 32)            3104
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 10, 32)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 8, 64)             6208
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 6, 64)             12352
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 30,242
Trainable params: 30,242
Non-trainable params: 0
_________________________________________________________________
```

Figure C.1: Architecture of the CNN1 model of the binary particle classification.

```
Model: "Binary particle classification FNN1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 24, 1)]           0
_________________________________________________________________
flatten (Flatten)            (None, 24)                0
_________________________________________________________________
dense (Dense)                (None, 128)               3200
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 66
=================================================================
Total params: 30,242
Trainable params: 30,242
Non-trainable params: 0
_________________________________________________________________
```

Figure C.2: Architecture of the FNN1 model of the binary particle classification.

```
Model: "Binary particle classification FNN2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 24, 1)]           0
_________________________________________________________________
flatten (Flatten)            (None, 24)                0
_________________________________________________________________
dense (Dense)                (None, 64)                1600
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 5,890
Trainable params: 5,890
Non-trainable params: 0
_________________________________________________________________
```

Figure C.3: Architecture of the FNN2 model of the binary particle classification.

C.2 Models loss of the training process

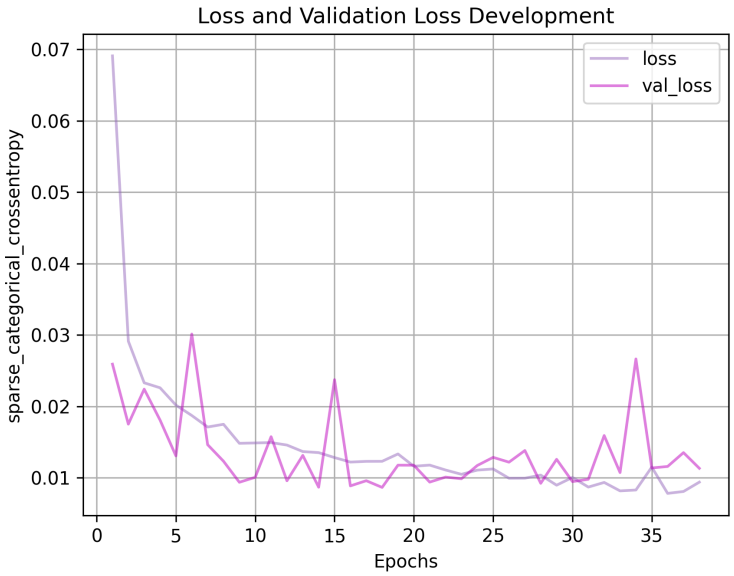

Figure C.4: Loss of the training process of the CNN1 model.

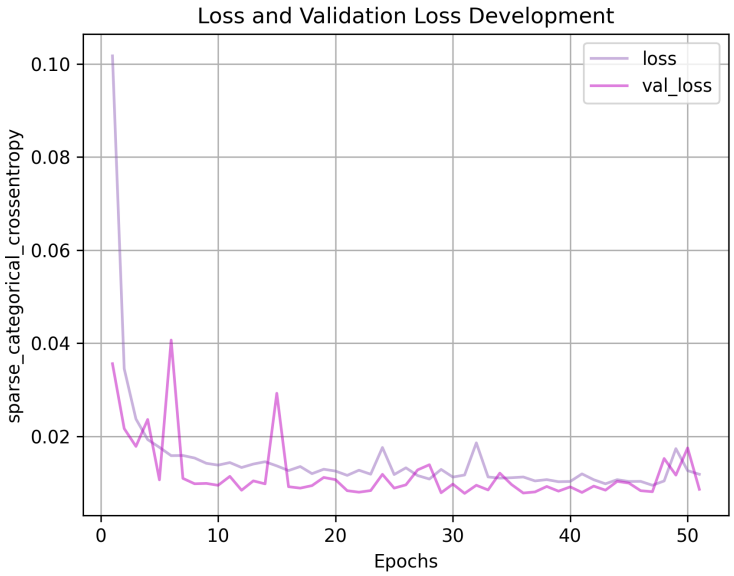

Figure C.5: Loss of the training process of the FNN1 model.

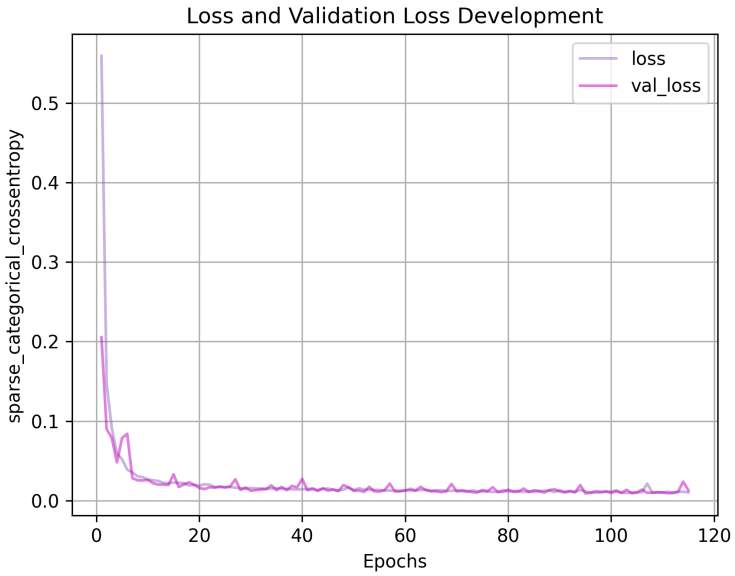

Figure C.6: Loss of the training process of the FNN2 model.

C.3 LHD of wrongly classified data of the FNN models

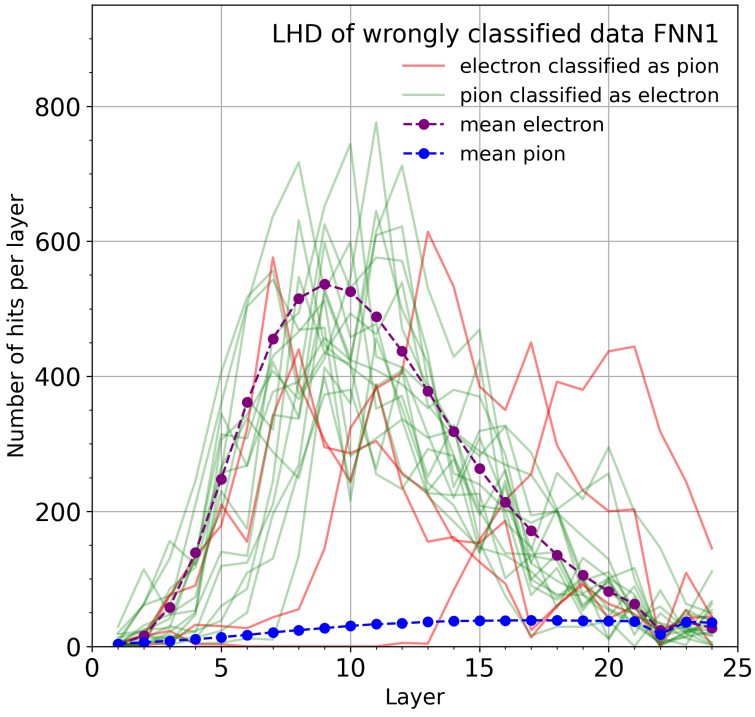

Figure C.7: LHD of wrongly classified events of the FNN1.

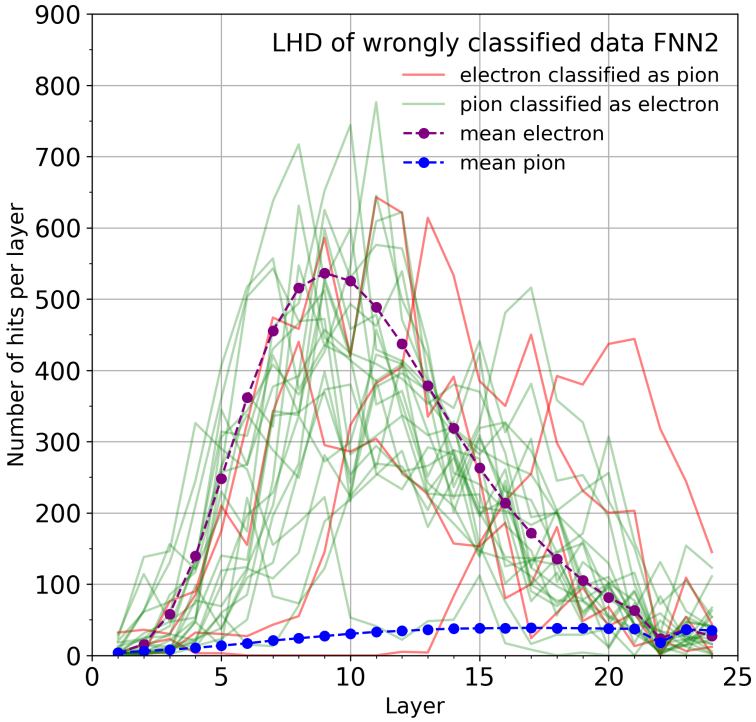

Figure C.8: LHD of wrongly classified events of the FNN2.

Appendix D

Particle identification

D.1 Model architectures

```
Model: "Particle identification CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 24, 1)]           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 22, 90)            360
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 20, 90)            24390
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 10, 90)            0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 8, 120)            32520
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 6, 120)            43320
_________________________________________________________________
global_average_pooling1d_1 ( (None, 120)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               12100
_________________________________________________________________
dense_5 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_6 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_7 (Dense)              (None, 20)                1020
_________________________________________________________________
clasout (Dense)              (None, 3)                 63
=================================================================
Total params: 128,923
Trainable params: 128,923
Non-trainable params: 0
```

Figure D.1: Architecture of the CNN models of the particle identification.

```
Model: "Particle identification FNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 24, 1)]           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 24)                0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               3200
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_11 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_12 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_13 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_15 (Dense)             (None, 3)                 99
=================================================================
Total params: 30,275
Trainable params: 30,275
Non-trainable params: 0
_________________________________________________________________
```

Figure D.2: Architecture of the FNN models of the particle identification.

D.2 Confusion matrices

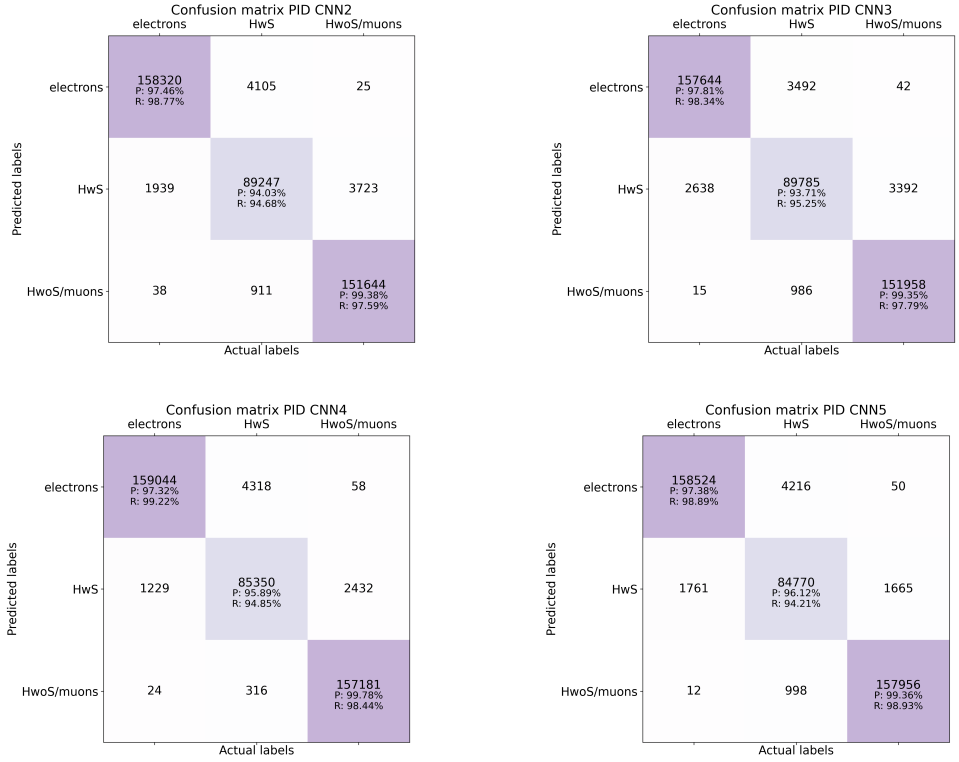

Figure D.3: Confusion matrix for the CNN models with (left) and without (right) class balance weights, and for the 150 (top) and 200 (bottom) hits hadron cut. For each class, the precision P and recall R are calculated.

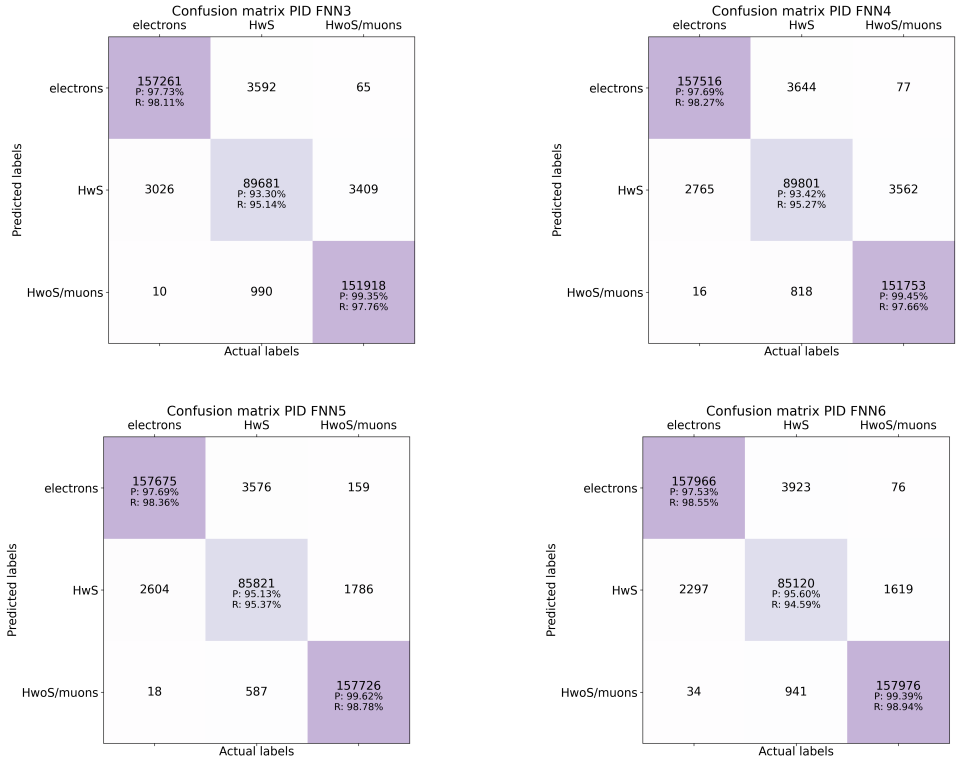

Figure D.4: Confusion matrix for the FNN models with (left) and without (right) class balance weights, and for the 150 (top) and 200 (bottom) hits hadron cut. For each class, the precision P and recall R are calculated.

Appendix E

Electron energy regression

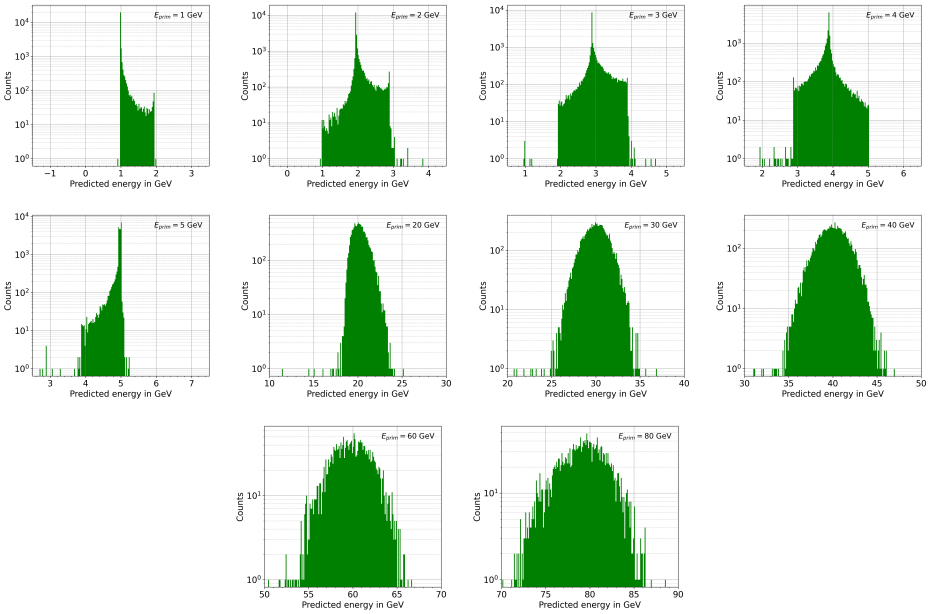

Figure E.1: Distributions of the predicted energies of the energies E_{prim} =1, 2, 3, 4, 5, 20, 30, 40, 60, and 80 GeV for the model using FNN with energy balance weights. The x-axis shows the predicted energy and the y-axis is the counted amount.

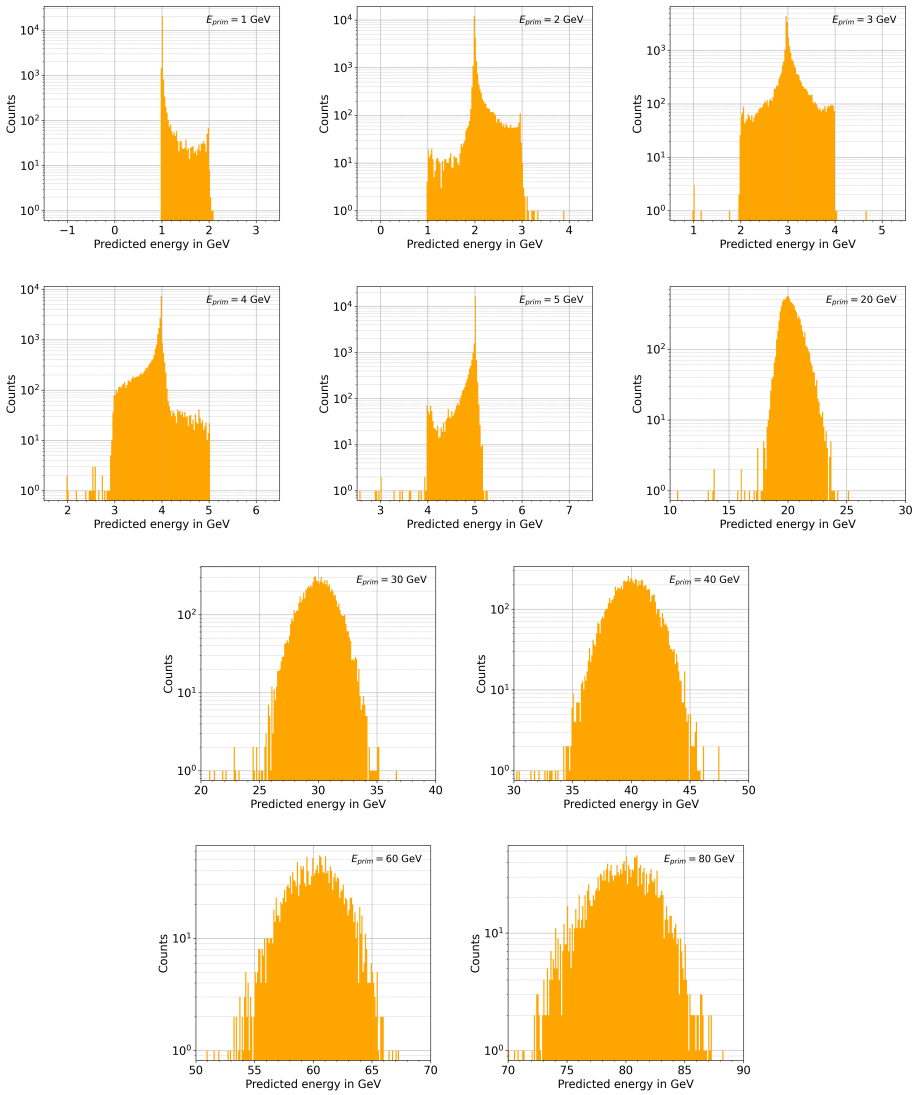

Figure E.2: Distributions of the predicted energies of the energies E_{prim} =1, 2, 3, 4, 5, 20, 30, 40, 60, and 80 GeV for the model using CNN without weights. The x-axis shows the predicted energy and the y-axis is the counted amount.

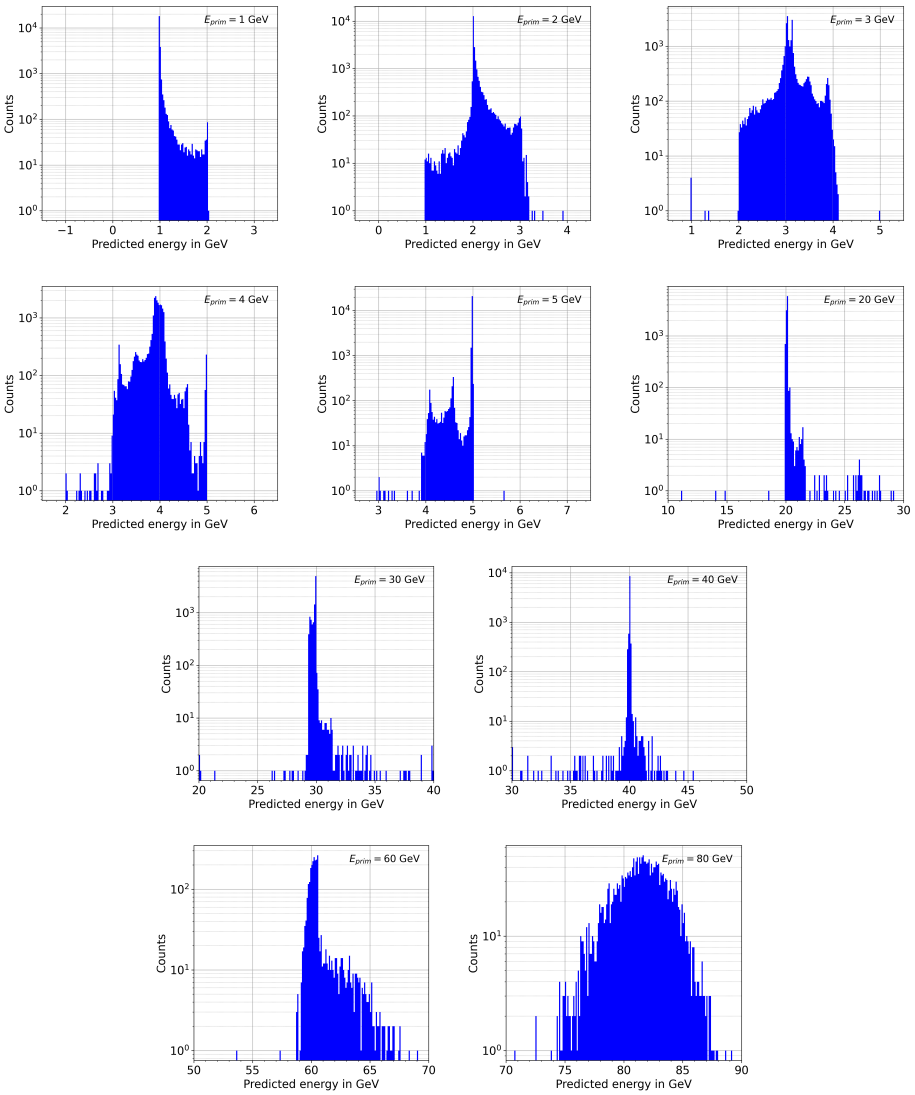

Figure E.3: Distributions of the predicted energies of the energies E_{prim} =1, 2, 3, 4, 5, 20, 30, 40, 60, and 80 GeV for the model using CNN with weights. The x-axis shows the predicted energy and the y-axis is the counted amount.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, sind von mir als solche kenntlich gemacht worden. Ferner erkläre ich, dass die Arbeit nicht - auch nicht auszugsweise - für eine andere Prüfung verwendet wurde.

Frankfurt a. M., den 28.09.2023

_____

Jan Scharf