JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN

Pattern recognition using machine learning for the mRICH detector in the mCBM experiment

Mustererkennung durch machinelles Lernen für den mRICH Detektor im mCBM Experiment

Master Thesis
by

Martin Beyer

October 2022

Physics Institute II
FB07
Justus-Liebig-Universität Giessen

Supervisor: Prof. Dr. Claudia Höhne
Examiner: apl. Prof. Dr. Jens Sören Lange

Selbstständigkeitserklärung

Hiermit versichere ich, die vorgelegte Thesis selbstständig und ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt zu haben, die ich in der Thesis angegeben habe. Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, und alle Angaben die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht. Bei den von mir durchgeführten und in der Thesis erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis, wie sie in der ‚Satzung der Justus-Liebig-Universität zur Sicherung guter wissenschaftlicher Praxis' niedergelegt sind, eingehalten. Entsprechend § 22 Abs. 2 der Allgemeinen Bestimmungen für modularisierte Studiengänge dulde ich eine Überprüfung der Thesis mittels Anti-Plagiatssoftware.

Datum	Unterschrift

Abstract

The mRICH detector in the mCBM experiment is observing very characteristic hit patterns. Because of the mCBM detector setup, charged particles are passing directly through the mRICH detector, resulting in pixel clusters mostly inside ring structures. This causes the Hough Transform ring finder algorithm to reconstruct many false rings. For this work a supervised machine learning approach using a convolutional neural network (CNN) has been developed to recognize and remove those central pixel clusters, but also noise in general. This supervised approach requires labeled data, which is gathered from simulations. Since the simulation did not include a realistic detector response for charged particles, additional hit patterns have been added to simulations to get a comparable response to the observed real data. Applying the trained CNN to simulation and real data shows a large improvement for the ring finding process, which can be directly verified for simulations using Monte Carlo information.

Zusammenfassung

Der mRICH Detektor in dem mCBM Experiment zeigt sehr charakteristische Hit Muster. Aufgrund des mCBM Detektor Aufbaus, fliegen geladene Teilchen direkt durch den Detektor und hinterlassen Pixel Cluster, meisten in der Mitte von Ring Strukturen. Dies zusammen mit dem Hough Transform Ringfinder führt zu vielen falsch rekonstruierten Ringen. Für diese Arbeit wurde überwachtes maschinelles Lernen als Ansatz gewählt um dieses Problem anzugehen. Es wurde ein convolutional neural network (CNN) entwickelt, um zentrale Pixel Cluster und auch andere Hintergrundsignale zu entfernen. Für den überwachten Ansatz benötigt es kategorisierte Daten, welche aus Simulationen genommen werden. Da die Simulationen diese zentralen Hit Cluster nicht realistätsnah beschriebt, wurden zusätzliche Hit Muster hinzugefügt, um die Simulationsdaten vergleichbarer zu gemessenen echten Daten zu machen. Die Anwendung des trainierten CNN auf simulierte und echte Daten zeigt eine große Verbessung für die Ringfindung, was für simulierte Daten unter Verwendung von Monte Carlo Information direkt verifiziert werden kann.

Contents

Chapter 1

Machine learning

Nowadays artificial intelligence (AI) is getting increasingly more popularity in all kinds of areas, such as medicine, self-driving cars, speech and object recognition and more. A subgroup of AI is machine learning (ML), this focuses on training a statistical model based on data, preferably a lot of data. Since in recent year powerful GPUs became available, very large datasets and networks could be used for the learning process of a model. Even the usage of multiple billion parameters is possible. For this work, the supervised learning method is used exclusively. This requires the data to be labeled into different classes to be used in the learning process. Different approaches are semi-supervised learning methods, which are using labeled and unknown data for the training process or unsupervised methods, which are not using labeled data at all. This can be helpful, since labeling data is typically very time-consuming, but the overall performance is often not on par with supervised approaches.

1.1 Artificial neural networks

1.1.1 Neuron

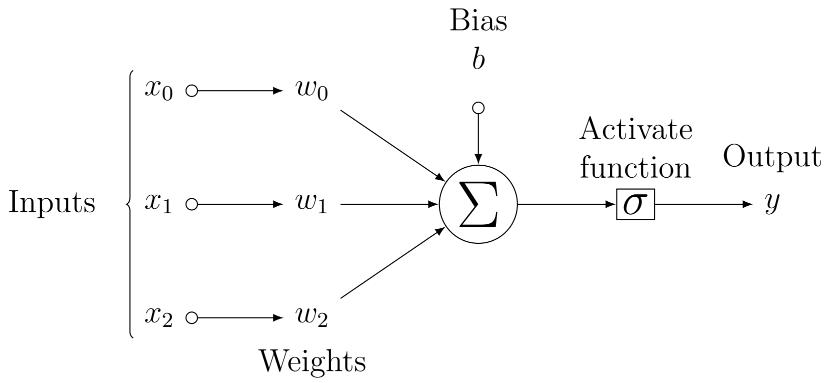


Figure 1.1: Illustration of an artificial neuron.

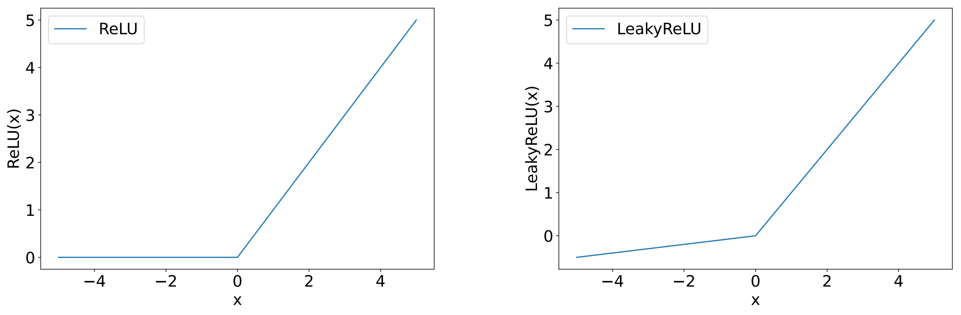
The artificial neuron or perceptron (see figure 1.1), is the most basic unit of a neural network, for given inputs x_i a single output value y is calculated using equation 1.1

$$y = \sigma(\sum_i w_i x_i + b) = \sigma(w^T x + b) \tag{1.1}$$

With w_i being the weight vector components, b the bias value and σ an activation function (see section 1.1.2). The argument of the activation function is a simple multidimensional linear function, the bias value is important in order to add a degree of freedom. As the name states, the inspiration comes from biological neurons. For those, the activation function would be most likely a threshold function which either fires or doesn't, depending on the neuron structure and the given inputs. For artificial neural networks different activation functions are used to allow more flexible networks.

1.1.2 Activation functions

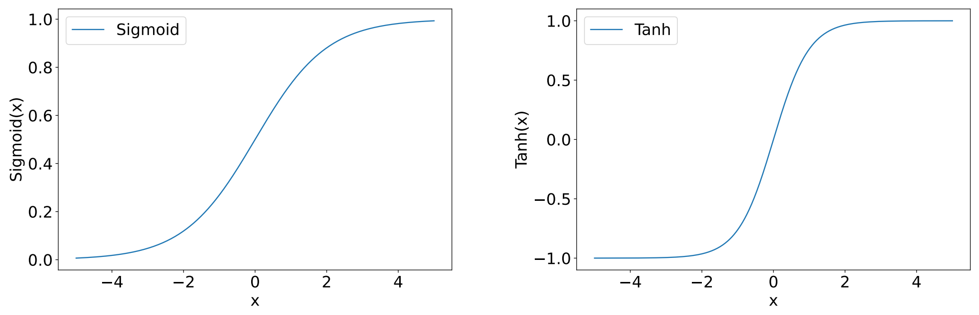
Activation functions are an essential part of neural network architectures. The task of the activation function is to transform the weighted input (+ bias) into a final output of a given node. The two main benefits of using activation functions are the introduction of non-linearity into the network and for the output nodes the mapping of output values to a desired interval (e.g. (0, 1)). In the case of a fully linear model, i.e. an activation function f(x) = x, all inputs together with weights and biases are just linearly transformed. No matter how many layers are added, the resulting network will be a simple linear function and the learning of complex tasks won't be possible. Effectively, a linear activation function would let the network layers collapse into a single layer. Therefore, it is really necessary to use non-linear activation functions, especially for hidden layers. Historically the sigmoid activation was used as the activation function for hidden layers.



(a) ReLU. (b) Leaky ReLU.

Figure 1.2: Common activation functions for hidden layers.

Because of common vanishing gradient problems [1] for the sigmoid activation and the fact that it's computational more expensive, nowadays different activation functions like the Rectified Linear Unit (ReLU) functions are used (fig. 1.2a). Those are far less susceptible to vanishing gradient problems. However, specifically for the ReLU activation it can happen that a neuron output always, i.e. for all neuron inputs, maps to 0 after passing through the ReLU activation, thus making this activation useless. This can for example be caused by a large negative bias value. This is called the dying ReLU problem and can be avoided using different activations yielding non-zero values for negative inputs, e.g. the leaky ReLU (fig. 1.2b).



(a) Sigmoid. (b) Tanh.

Figure 1.3: Common activation function for output layers.

Although the sigmoid function is not suitable for hidden layers, it is still an important activation function for the output layer to map values to the desired output interval. Common functions for binary classifications are the Tanh (fig. 1.3b) mapping from $\mathbb{R} \rightarrow (-1, 1)$ and the already mentioned sigmoid mapping from $\mathbb{R} \rightarrow (0, 1)$ (fig. 1.3a). For multiclass classification with N output variables, the Softmax function (eq. 1.2) is commonly used.

$$s_i(\vec{z}) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad \text{i = 0, ..., N-1} \tag{1.2}$$

1.1.3 Multilayer perceptron

The next step for building a neural network is to put together several neurons. This is done by passing an input into multiple neurons, while all of them typically have different weights and biases. The collection of these neurons is called a layer. Layers visible from the outside are the input layer and output layer, while everything in between is called a hidden layer. The number of neurons (also called nodes) can be varied for all layers including the output layer. Such networks (see fig. 1.4) are called multilayer perceptron (MLP) or fully connected networks (fc) or artificial neural networks (ANN). Such networks, including the non-linearity from the activation functions, allow the recognition of complex dependencies based on the inputs. However, they need to be trained correctly, need to have enough parameters, a suiting model architecture and the data has to have some sort of pattern.

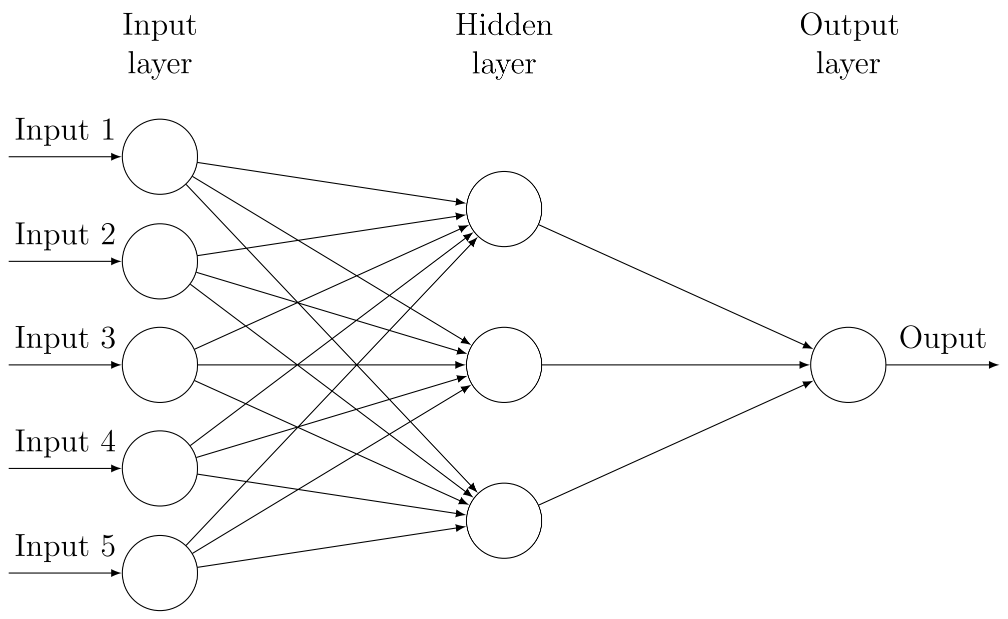


Figure 1.4: Basic MLP with each node being a neuron and a binary output layer.

Given the output $x^{(k-1)}$ of a previous layer with n_{k-1} neurons, the output of the current layer $x^{(k)}$ with n_k neurons can be calculated using equation 1.3, which is adapted from equation 1.1. To simplify later calculations the argument of the sigmoid function, is called $z^{(k)}$.

$$x_i^{(k)} = \sigma(\sum_{j=1}^{n^{(k-1)}} w_{ji}^{(k)} x_j^{(k-1)} + b_i^{(k)}) = \sigma(z_i^{(k)}) \qquad i = 1, ..., n^{(k)} \tag{1.3}$$

1.1.4 Backward propagation

The central algorithm of most machine learning tasks is the backward propagation (or backpropagation) process. The basic idea is to calculate an error based on a certain loss function (see section 1.1.5), this value then gets minimized by gradually adjusting the weights and biases of the model. To achieve this the gradient of the loss function with respect to weights and biases has to be calculated. Given a loss function, an error (equation 1.4) can be calculated using the output of the model y and the target value $\hat{y}$.

$$E = L(\hat{y}, y) \tag{1.4}$$

Calculating the partial derivative of the error with respect to a weight using equation 1.3, yields equation 1.5. The last 2 derivatives in equation 1.5 can easily be calculated using equation 1.3, simply being the input vector (eq. 1.6) for the layer and the derivative of the activation function (eq. 1.7).

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = \frac{\partial E}{\partial x_j^{(k)}} \frac{\partial x_j^{(k)}}{\partial w_{ij}^{(k)}} = \frac{\partial E}{\partial x_j^{(k)}} \frac{\partial x_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} \tag{1.5}$$

$$\frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} = x_i^{(k-1)} \tag{1.6}$$

$$\frac{\partial x_j^{(k)}}{\partial z_j^{(k)}} = \sigma'(z_j^{(k)}) \tag{1.7}$$

The first term $\frac{\partial E}{\partial x_j^{(k)}}$ is only known for the last layer N, where it can easily be calculated using the loss function together with the output y of the network and the target $\hat{y}$.

$$\frac{\partial E}{\partial x_j^{(N)}} = \frac{\partial E}{\partial y_j} \tag{1.8}$$

This leads to an iterative formula beginning from the last layer and being calculated towards the input layer. That is the reason why it is called back-propagation.

$$\frac{\partial E}{\partial x_j^{(k)}} = \sum_{l=1}^{n^{(k+1)}} \frac{\partial E}{\partial x_l^{(k+1)}} \frac{\partial x_l^{(k+1)}}{\partial z_l^{(k+1)}} \frac{\partial z_l^{(k+1)}}{\partial x_j^{(k)}} \tag{1.9}$$

Having calculated all necessary values in equation 1.5 for a given weight, an optimization step can be applied to update the weights using a learning rate α. The biases are calculated and updated similarly. Equation 1.10 demonstrates a weight update using a simple gradient descent (GD) optimizer.

$$w_{ij}^{(k)} \to w_{ij}^{(k)} - \alpha \cdot \frac{\partial E}{\partial w_{ij}^{(k)}} \tag{1.10}$$

Typically more advanced optimizers are used such as ADAM [2], for faster and better convergence. Those optimizers use techniques like adaptive learning rate and momentum.

1.1.5 Loss functions

The whole training process of a network is based on optimizing the loss value. The overall goal is to minimize the loss value, in order to fit the model to the data. Loss functions also have to be differentiable (e.g. equation 1.9), like activation functions (see equation 1.7). A common loss function used for regression tasks of all kind, is the mean squared error (MSE). It is also regularly used as a loss function for NN training and is calculated using equation 1.11

$$MSE = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 / N \tag{1.11}$$

with y_i being the predicted values from the model and $\hat{y}_i$ the true/target values. One disadvantage of using the MSE is that large deviations from the true value dominate strongly because of using the difference squared. For classification tasks the binary cross entropy (BCE) loss is a good choice, since parts of the equation dropping off when using target values of 0 and 1.

$$BCE_i = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{1.12}$$

For each output value the BCE loss can be calculated using equation 1.12. From the set of results, the total loss can be derived by calculating the mean or sum. Since the target values $\hat{y}_i$ are mostly 0 or 1, the logarithm is constantly diverging. For example in the PyTorch framework [3], this is avoided by restricting the log to have a minimum value of -100, which make the equation easily computable for all values $\hat{y}_i$. The BCE will therefore strongly penalize predictions which are close to the wrong target value, e.g. the target value is 1, but the prediction is close to 0.

It should be noted that some combinations of output activation functions and loss functions run into problems. For example the MSE and the sigmoid activation in the output layer (see Appendix A).

1.1.6 Model training

The overall training process starts with collecting/generating labeled data, this in itself can be very time-consuming. An available dataset typically gets separated into 3 different datasets

- Training data: Data directly used for training.
- Validation data: While training the model, the loss value and maybe other metrics are calculated after each training epoch, to see how well the model is generalizing and to avoid overfitting or underfitting.
- Test data: Another dataset which is not used for training and is more unbiased as the validation dataset[1]. The model is normally evaluated on this dataset, to see how the model performs on unknown data.

For the training process the full training data set is separated into batches of a certain size, this is computational more efficient, since nowadays GPU's are used for the training process. Therefore, the weights and biases are only updates after one batch, which also helps to generalize and train the model. If the training is performed on all batches (the full training dataset), a so called 'epoch' is over and the next epoch starts, most likely shuffling the training dataset beforehand. A method to choose the best epoch to stop the training and to save the model, it to always keep track of the validation loss value. If the validation loss at the end of an epoch has not been improved after a defined number of 'patience' steps, the model is saved. This so called early stopping helps to prevent overfitting. Overfitting means that the network is too well adapted to the training dataset and generally performs way worse on unknown datasets. Underfitting is the opposite, when the network has a poor performance on the training dataset. An illustration on how a common overfit looks like based on the loss curve, in figure 1.5 an overfit is happening after the early stop epoch.

[1]The validation dataset is usually used to decide when to stop the training process and therefore isn't an independent dataset anymore.

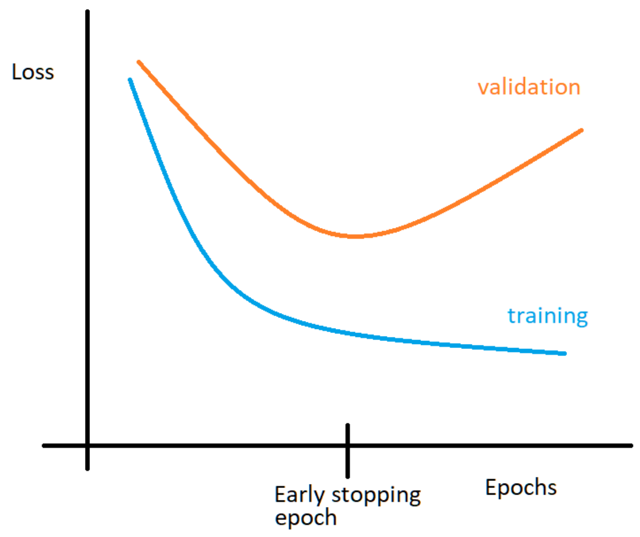


Figure 1.5: Loss curves for train and validation data, showing a clear overfitting for a higher epoch count. Early stopping would select the model at the epoch with the lowest validation loss.

1.2 Convolutional neural networks

For image data it is often of interest to recognize objects or patterns only in certain areas of the picture. This uses the assumption, that pixels close to each other are more related than pixels with a large distance in between them. Using fully connected networks (section 1.1.3) for such tasks would require an immense amount of parameters, in particular for large picture sizes. Thus, those networks become practically unusable and one has to choose a different architecture to achieve a good classification performance. For this task convolution is a more suitable approach to extract and process information, since it benefits from spacial invariance while processing data and is first focusing on local areas. Spacial invariance means that all parts of the picture are processed the same way, at least in the first layers. All calculations and illustrations in this section are done for 2d convolution, but can easily be generalized to more dimensions. In addition, all inputs, outputs, convolution kernels, padding and striding are kept with equal sizes for all dimensions.

1.2.1 Convolution

The convolution operation for neural networks is similar to the convolution in mathematics, shown in equation 1.13 with $g, h : \mathbb{R} \to \mathbb{R}^n$.

$$(g * h)(x) = \int g(y)h(x-y)dy \tag{1.13}$$

For a discrete space like 1d grid data, the integral turns into a sum (equation 1.14) with the index position a.

$$(g * h)(a) = \sum_i g(i)h(a-i) \tag{1.14}$$

For 2 dimensions this is expressed using 2 sums and grid positions a, b (equation 1.15).

$$(g * h)(a, b) = \sum_{i,j} g(i, j)h(a-i, b-j) \tag{1.15}$$

For convolution layers a set of filters (or kernels) with a given size of $f \times f$ are used. With this restriction only nearby pixels get convoluted together. This is called locality and is a main concept of convolution layers. Eventually information from a larger distance to the source pixel can be convoluted by using multiple convolution layers or downsampling methods. Networks using multiple convolution layers are called convolutional neural networks (CNNs). Almost always an odd filter size like 3×3 or 5×5 is used, since the convoluted information from neighboring pixels to the center pixel is most interesting. For even-sized filters there is no center pixel after the convolution process (see fig. 1.6). Given the corresponding weight matrix k, the convoluted output can be calculated using equation 1.16[2]

$$y_{a,b} = \sum_{i=-u}^{u} \sum_{j=-u}^{u} k_{i,j}\, x_{a+i,b+j} + B \tag{1.16}$$

with the source pixel indices a, b and the filter size $f \times f = (2u+1) \times (2u+1)$ being odd. For each filter, there is one bias value B added and the output $y_{a,b}$ is passed through an activation function, the same as for MLPs.

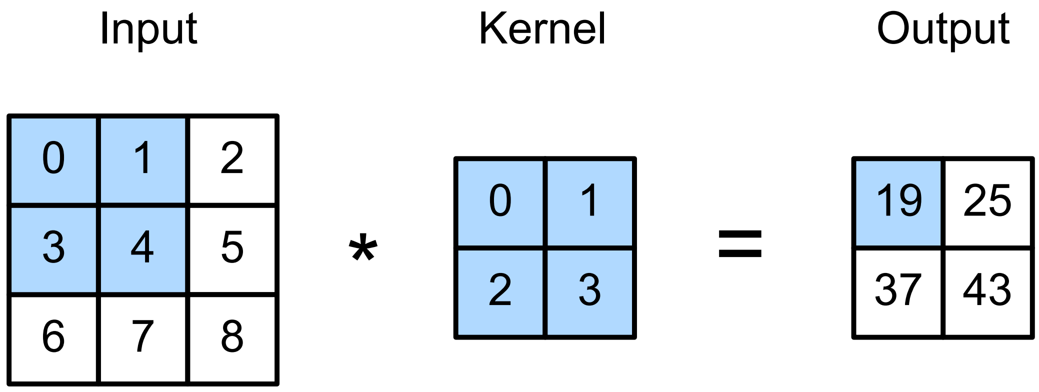


Figure 1.6: Convolution using a 2×2 kernel. [4]

A example calculation for the output of a convolution is shown in figure 1.6, the output will have a reduced dimension compared to the input, which can be avoided by using padded values (section 1.2.2). For the input layer is common that the data size is of $W \times H \times C$. For hidden layers this is always the case, because multiple filters with different weights and biases are used. With W and H being the width and height of a picture and C being the channel dimension. For example an RGB picture would have a size of

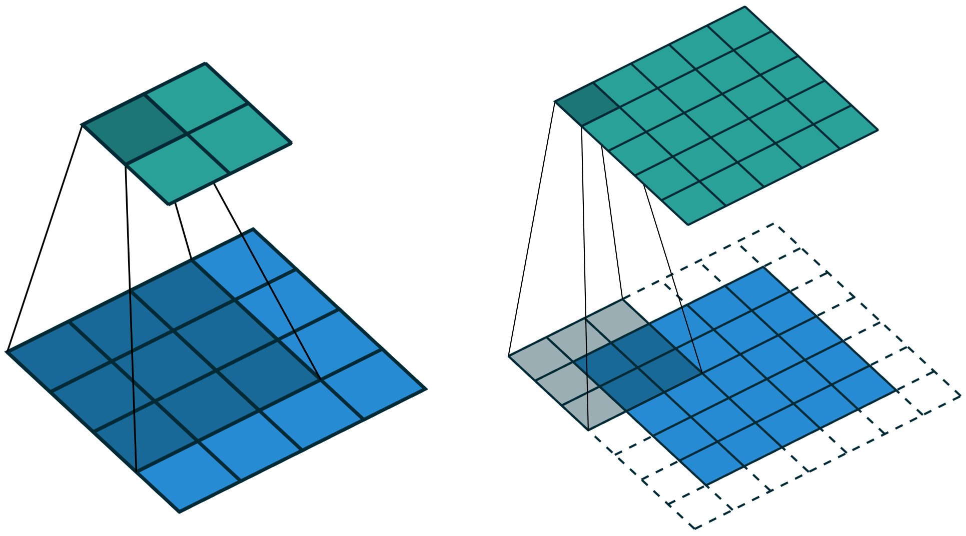
[2]Conventionally the sign is changed from a-i to a+i and b-j to b+j.

$W \times H \times 3$ with the 3 color channels. For input data with a channel size of $c_{in} > 1$, the channel size for the convolution output will be c_{out} = Number of filters. The filters will have a size $f \times f \times c_{in}$. Applying a filter on input data results in a so-called feature map. Those feature maps are then stacked along the channel dimension. The output for each output channel can be calculated using equation 1.17.

$$y_{a,b,c_{out}} = \sum_{i=-u}^{u} \sum_{j=-u}^{u} \sum_{c_{in}} k_{i,j,c_{in},c_{out}} \, x_{a+i,b+j,c_{in}} + B_{c_{in}} \tag{1.17}$$

1.2.2 Padding and striding

When applying the convolution operation seen in equation 1.16, the resulting output will be of different dimension depending on the filter size, illustrated in figure 1.7a. Adding multiple convolution layers will then result in a significant reduction of the data dimension compared to the input, this might not be desired.



(a) Padding 'valid'. (b) Padding 'same'.

Figure 1.7: Illustration of different padding settings. The input pixels are shown in blue, while the padded pixels are shown in white. [5]

Another problem is that edge pixels get processed less often compared to pixels in the center of the data, which might result in the loss of valuable information. To avoid this and the change of the data dimension, a technique called padding can be used. Depending on the filter size, additional pixels p get added to the boundaries of the input data. For the setting 'valid' the padding value p is 0, for 'same' the value depends on the filter size with $p = \frac{f-1}{2}$ and f being odd. The output from the convolution including padding 'same' will have the same dimension as the input (see 1.7b), while for padding 'valid' the data size reduces when using filter sizes larger than 1×1. There are different possibilities which values to use for the padded pixels, most commonly used is the so called zero-padding, i.e. setting all additional pixels to 0. Another option would be for example to do a reflection of the input pixels to the padded pixels, i.e. in figure 1.7b reflecting the values of the blue boundary pixels to the padded pixels.

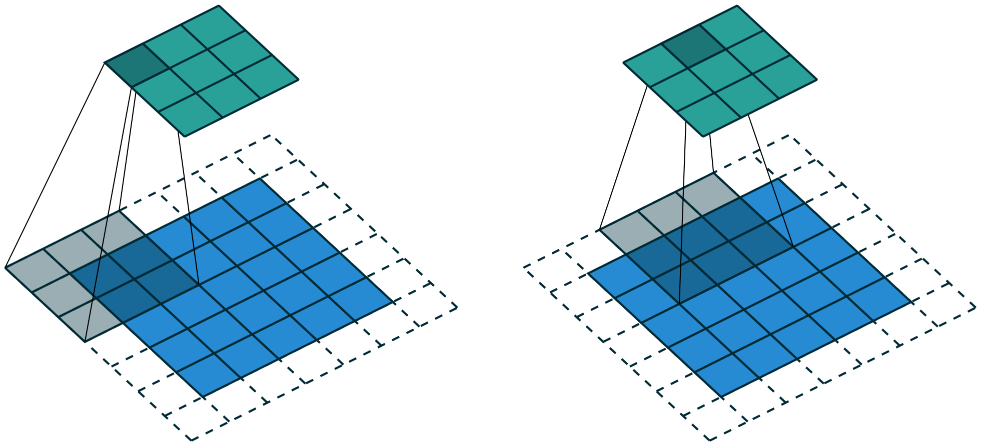


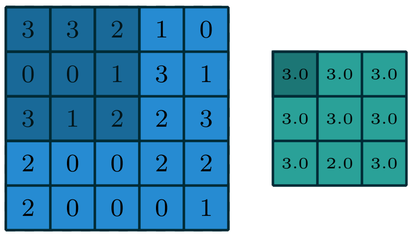
Figure 1.8: First 2 convolution operations including striding of size 2 in both directions and padding 'same'. [5]

A different option for convolution layers is striding. The stride parameter defines the step size the filter moves at a time, i.e. for example one pixel or two. A stride of 2 is illustrated in figure 1.8, the output size will then be smaller than the input and this can effectively be used as a down sampling method. Given the input dimension N_{in} for one direction and stride value s, the resulting output dimension N_{out} for this direction can be calculated using equation 1.18.

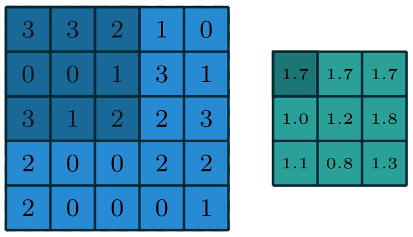
$$N_{out} = \frac{N_{in} + 2p - f}{s} + 1 \tag{1.18}$$

1.2.3 Pooling

Another way of reducing the data dimension can be done by using pooling layers. Those layers also have the option to set the filter size and striding amount, to regulate the overall downsampling factor. The most common option is a filter size of 2x2 with striding 1, to reduce both dimension sizes by a factor of two. An example for the calculation of the pooling operation is shown in figure 1.9, comparing two popular pooling layers Maxpool and Averagepool. Maxpool keeps the largest value within the kernel, Averagepool calculates the average.



(a) Maxpool.



(b) Averagepool.

Figure 1.9: Comparison of 2 pooling methods using a 3×3 kernel without padding and striding. [5]

The difference between those two different pooling methods are that Maxpooling rejects a lot of data, while Averagepooling retains a lot. Which pooling layer to use depends on the specific data and task, e.g. it might happen that Averagepooling can not extract the most important information, while Maxpooling can since it focuses only on the most dominant features.

1.2.4 Upscaling using convolution transposed

The opposite effect of pooling layers have upsampling layers, which are increasing the dimension of given data by resizing. Common ways are methods like nearest neighbor, bilinear interpolation or bicubic interpolation [6, 7]. A different approach of upsampling is the counterpart of a convolution layer, called convolution transposed layer. Those are using the same concepts as a normal convolution layer like the filters including weights and biases and activation functions and are therefore a trainable way of upsampling. The transposed convolution process is illustrated in figure 1.10. Since a stride value of 2 used, the output dimension doubles for both directions.

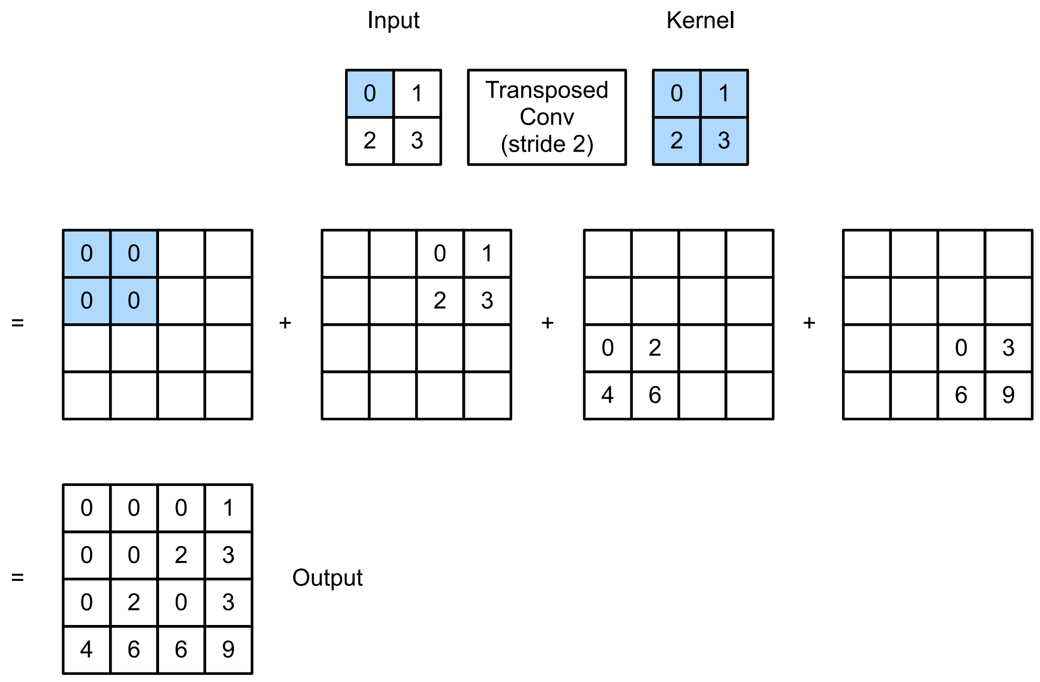


Figure 1.10: Transposed convolution using a 2×2 kernel and striding of 2. [4]

1.3 Evaluation

1.3.1 Confusion matrix

A confusion matrix is a very good representation of a classification performance. A common way for a binary classification task using one output value, is to set a threshold value to decide which values belong to which class. The output value could simply be in between (0, 1). As a threshold, for example a fixed value can be used. For all cases where the prediction is above a certain threshold value, the sample is attributed to class 1 and to class 0 if below this threshold.[3] For a given threshold value, four outcomes can occur for a binary classification task:

- True Positive (TP): Represents the number of data which is actually positive, and the predicted class is also positive.
- True Negative (TN): Same as TP, but with the actual and predicted class being negative.
- False Positive (FP): The Type I error, data belonging to the actual negative class getting classified as positive.
- False Negative (FN): The Type II error, data belonging to the actual positive class getting classified as negative.

Those results can be represented in a matrix format, illustrated in figure 1.11.

[3]For multiple outputs values, each belonging to one class, the softmax activation is mostly used together with taking the index of the largest value in the output vector as the predicted class.

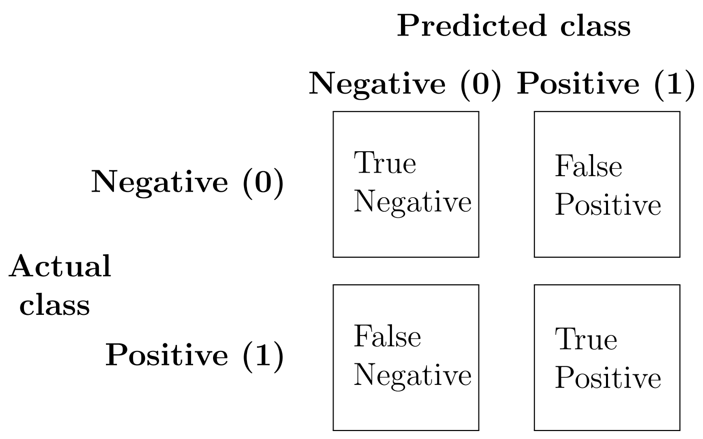


Figure 1.11: Confusion matrix for binary classification.

Based on this confusion matrix, several metrics can be calculated. The Accuracy (equation 1.19) of a classifier, representing how often any of the class predictions are correct. This metric is basically becoming meaningless for strongly unbalanced datasets, having a lot more data for one class, e.g. 1% belongs to class 0 and 99% to class 1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.19}$$

The Sensitivity (equation 1.20) is describing how 'sensitive' the classifier is. I.e., if the actual value is positive, how often the classifier predicts correctly. It is also called recall.

$$Sensitivity = \frac{TP}{TP + FN} \tag{1.20}$$

The Specificity (equation 1.21): if the actual value is negative, how often the classifier predicts correctly.

$$Specificity = \frac{TN}{TN + FP} \tag{1.21}$$

The Precision (equation 1.22): if the predicted value is positive, how often the classifier is correct.

$$Precision = \frac{TP}{TP + FP} \tag{1.22}$$

Which metric to 'trust' the most depends on the specific task[4] and dataset. Many more metrics can be calculated from the confusion matrix, like the F1 score (equation 1.23) which values unbalanced datasets better by using the harmonic mean of precision and sensitivity.

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{1.23}$$

The confusion matrix concept can be easily expanded to more than 2 classes.

[4]For example an e-mail spam filter, marking actual important e-mails a spam is very bad, while spam marked as a reasonable e-mail is still acceptable.

1.3.2 ROC curve and AUC

The Receiver Operating Characteristic curve (ROC curve) is a graphical representation of the classification performance for binary classifier models[5]. This is done by varying the threshold value for the binary classifier. For a certain threshold, the true-positive rate (TPR) and false-positive rate (FPR) are calculated using equations 1.24 and 1.25.

$$TPR = \frac{TP}{TP + FN} = Sensitivity \tag{1.24}$$

$$FPR = \frac{FP}{FP + TN} = 1 - Specificity \tag{1.25}$$

The ROC curve is then derived by calculating the TPR and FPR for the threshold spectrum (0, 1). Example curves are shown in figure 1.12, together with a random classifier which always has 50% TPR and 50% FPR.

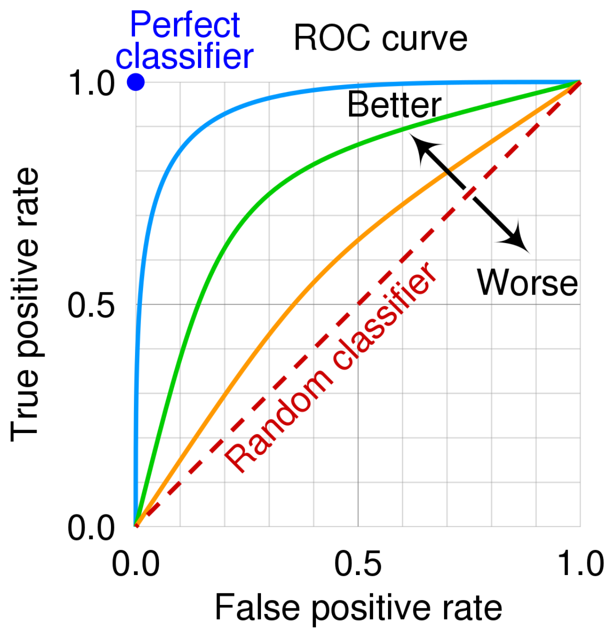


Figure 1.12: ROC curves for different classifiers and the random classifier as reference. [8]

[5]If expanding this to more than 2 classes, it is common to compute ROC curves for all class pair combinations or one class versus all others.

It is very helpful using the ROC curves to compare different classifiers. For example in figure 1.12, the classifier corresponding to the blue curve has an overall better classification performance compared to the green curve. A perfect classifier would always have a 100% TPR and 0% FPR. The ROC curve can also be used to determine a classification threshold, e.g. find the threshold which maximizes TPR - FPR. It is also common to just use the mean of the possible output range, e.g. 0.5 for a output in an interval of [0, 1].

The Area Under the ROC Curve (AUC) is a single value metric in the interval (0, 1). A larger number indicates a better classifier. It measures the performance independently of the specific classification threshold value. In addition, it is a very good metric for evaluating datasets having a class imbalance, compared to metrics like accuracy (equation 1.19).

Chapter 2

Introduction

2.1 The Standard Model

The Standard Model (SM) of particle physics is currently the best theory to describe the fundamental structures of matter and forces. Forces are connected to the electromagnetic, weak and strong interactions, while the fourth known fundamental force, the gravity is not unified in the theory. With precise measurements performed at several experiments studying collisions of particles, the SM has predicted many results with very high precision. The theories to describe the SM are the Quantum Electrodynamics (QED), electroweak theory and Quantum Chromodynamics (QCD), together forming the symmetry group of the Standard Model SU(3)×SU(2)×U(1). Those, together with the 12 fermions, 5 exchange particles and the Higgs boson [9] represent the full SM (see fig. 2.1). The fermions (spin = $\frac{1}{2}$) are separated into leptons and quarks, which are sorted into three generations. The mass of the particles is increasing for each generation, e.g. electrons are approximately 200 times lighter than muons. While the electrons, muons and taus have an electrical charge of -1, quarks have a fractional electrical charge of $\frac{2}{3}$ or $-\frac{1}{3}$ depending on the family they belong to.
The gauge bosons (spin = 1) are the exchange particles for the forces, mediating interactions between particles.

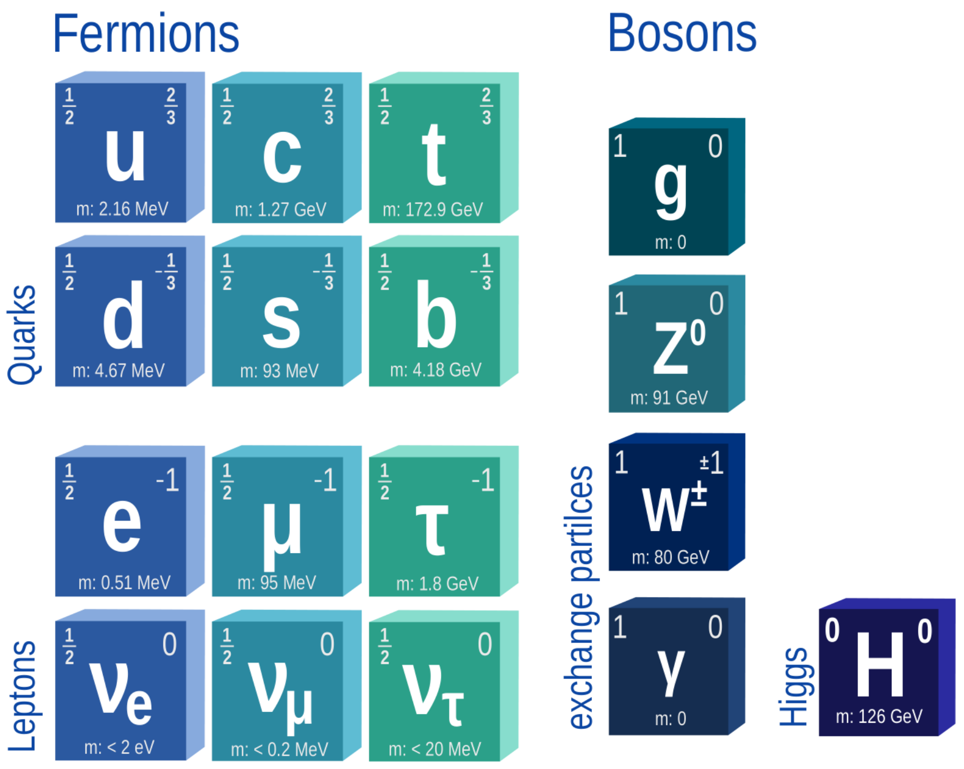


Figure 2.1: Sketch of the Standard Model. With the particle spin shown on top left, the electrical charge on top right and the mass in the bottom for each box. [10]

2.2 Electromagnetic Interaction

The QED is describing the interaction of charged particles by exchange of a mass-less photon, this is the gauge boson for the electromagnetic interaction with a symmetry of group of U(1). The photon has no charge and only interacts with other particles carrying electrical charge, there is no coupling between photons. The coupling strength of the force is defined by the fine-structure constant α [10].

$$\alpha = \frac{e^2}{4\pi\epsilon_0\hbar c} \approx \frac{1}{137} \tag{2.1}$$

This constant allows interactions described by the QED to be calculated very precisely using perturbation series. A series is calculated for a certain number of orders in α. Because the coupling is constant and < 1, the series is converging. Usually only a couple of order are calculated, because the amount of correction terms is growing rapidly with each consecutive order.

2.3 Electroweak theory

The electromagnetic and the weak interaction were unified into one theory by Glashow, Salam and Weinberg [11, 12]. This is called the electroweak interaction, and requires two additional quantities, the weak isospin I and the weak hypercharge Y. Both manifest themselves in the symmetry group for the theory of SU(2)×U(1). With U(1) being responsible for the weak hypercharge, which requires 1 massless gauge boson B and the symmetry SU(2) for the weak isospin requiring 3 massless gauge bosons W^1, W^2, W^3. The weak hypercharge can be connected to the electrical charge and the isospin with $Y = 2(Q - I_3)$, with I_3 being the third component or the isospin vector. With the weak mixing angle θ_W, also known as the Weinberg angle, the bosons W^3 and B can be connected to the known physical bosons, the Z^0 and the photon γ. The $W^{\pm}$ bosons can be acquired with a combination of W^1 and W^2 without the mixing angle. At this point in the theory, all four bosons were mass-less. By introducing the Higgs boson and the Higgs mechanism in the theory, the 3 bosons $W^{\pm}$, Z^0 are gaining their physical masses. The photon γ still remains mass-less, since there is no coupling to the Higgs field. The masses for the $W^{\pm}$ and Z^0 boson are also related via the mixing angle $cos\theta_W = \frac{m_{W^{\pm}}}{m_{Z^0}}$, with $m_{W^{\pm}}$ and m_{Z^0} being the masses of the bosons. Measuring the masses is a common way to determine the mixing angle experimentally.

2.4 Strong Interaction

The gauge group for the QCD is SU(3), with the nonabelian gauge bosons known as gluons [13, 10]. For the Quantum Chromodynamics, there are 6 different quarks called 'flavors', up (u), down (d), strange (s), charm (c), bottom (b) and top (t). Each of those carries a color charge as a quantum number, red (r), green (g) or blue (b). The corresponding anti quarks carry the anti-color charges anti-red ($\bar{r}$), anti-green ($\bar{g}$) or anti-blue ($\bar{b}$). Gluons on the other hand carry a color and a different anti-color and are self-interacting. Given the color combinations, there are 8 different gluons. A 9th combination would carry the same color and anti-color and therefore being color neural, causing no interaction. This also means that quarks and gluons are the only particles interacting with the strong force, since for example leptons do not carry a color charge and are therefore not interacting strongly. The main two hadron[1] groups of particles are baryons, which always have 3 valence quarks or anti-quarks, and mesons with two quarks, one being a quark and the other an anti-quark. For example a proton has a valence quark content of uud, with all quarks having a unique color, so the overall proton color is neutral. Combining the electrical charges of the quarks is resulting in an integer amount of the elementary charge (i.e. $q_{proton} = \frac{2}{3}e + \frac{2}{3}e - \frac{1}{3}e = +1e$). The fact that never there were any free quarks or colored hadrons observed, is correlated to the confinement. Unlike leptons which can be free particles, quarks inside hadronic matter are always bound to at least another quark, always leading to overall color neural bounds. There are other more exotic hadrons such as tetraquarks [14] or pentaquarks [15] possible.

[1]A particle made up from quarks and gluons, bound together by the strong interaction.

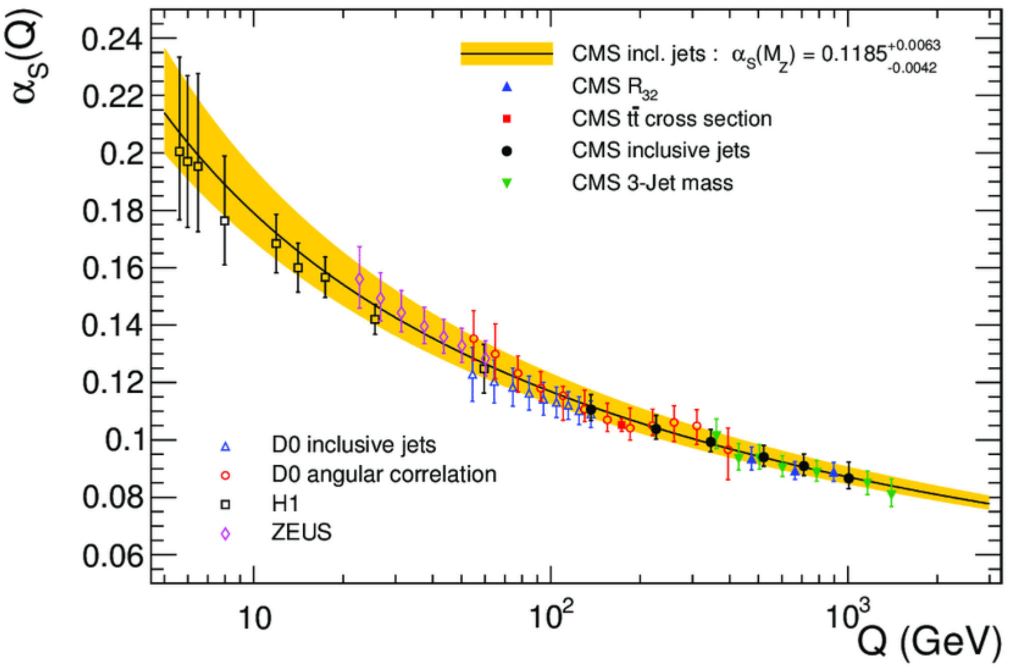


Figure 2.2: The coupling of the strong force as a function of the scale Q. Data from several detectors and experiments are shown, including the LHC, HERA and Tevatron colliders. [16]

The QCD coupling constant $\alpha_s(Q^2)$ for the interaction of quarks and gluons is dependent on the momentum transfer Q^2. The running coupling constant is shown in figure 2.2 measured by several experiments. For large momentum transfers Q^2 the coupling decreases logarithmically, the limit $Q^2 \to \infty$ is called asymptotic freedom and quarks are behaving as (quasi) free particles. On the other hand for low momentum transfers, the interaction becomes stronger, resulting in the confinement of quarks and gluons.

2.5 QCD phase diagram

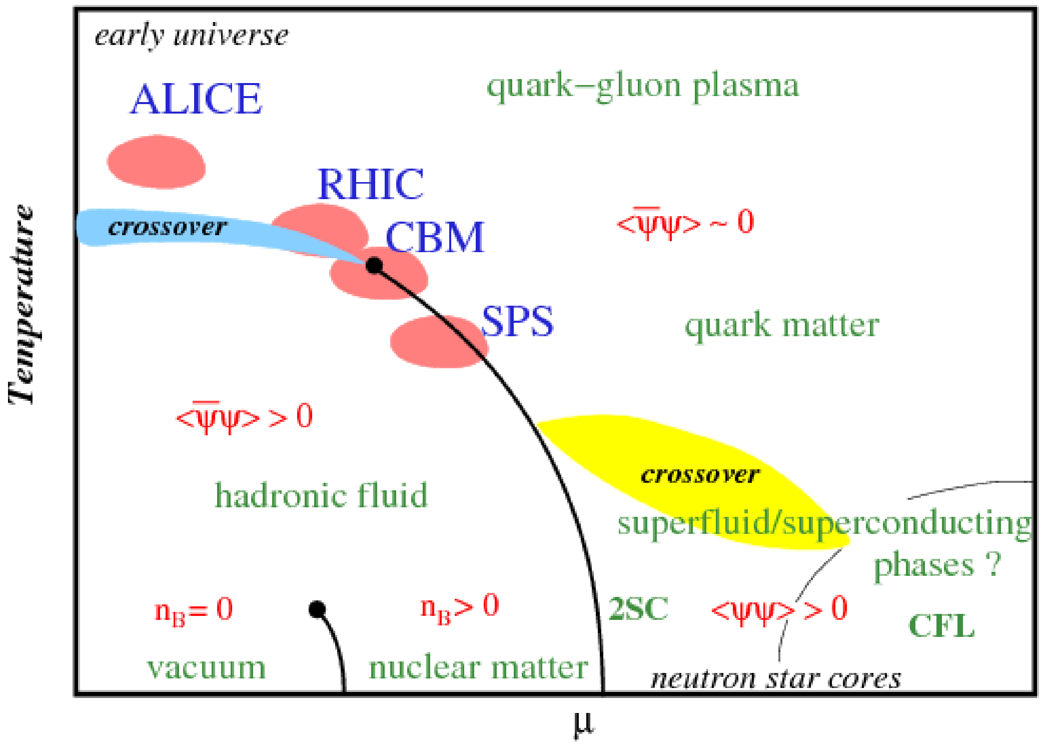


Figure 2.3: Schematic drawing of the QCD phase diagram. [17]

The QCD phase diagram for quarks and gluons is describing states of QCD matter, phase transitions and critical endpoints. QCD matter is described thermodynamically using the pressure P, temperature T and chemical potential(s) μ. Of special interest is the equation of state, describing the pressure with respect to the temperature and chemical potential. The current conception of the QCD phase diagram is schematically drawn in figure 2.3. For very high temperatures and optionally large baryon densities, a quark-gluon plasma (QGP) phase is expected. Similar to the electromagnetic plasma where electrical charged particles are separated, the QGP separates color charge, i.e. quarks and gluons. Because of the confinement, quarks can not just be separated from each other to gain color free particles. In the hadronic state, the separation of 2 quarks would lead to another quark pair being produced, since this is an energetically favored state. The result would still be color neutral hadrons. Although there is confinement for the hadronic states, for high temperatures and/or high compression of nuclear matter it is expected that quarks and gluons are changing to a deconfined QGP state. This would lead to (quasi) free particles carrying a color charge. Especially interesting are the phase transitions and the position of the critical endpoint. For certain areas of the phase diagram, non-perturbative calculations can be used. For low densities ($\mu \to 0$) QCD can be calculated using numerical lattice calculations [18]. This method has shown very good agreement with experimental measurements, but is not applicable to higher densities due to the numerical sign problem.

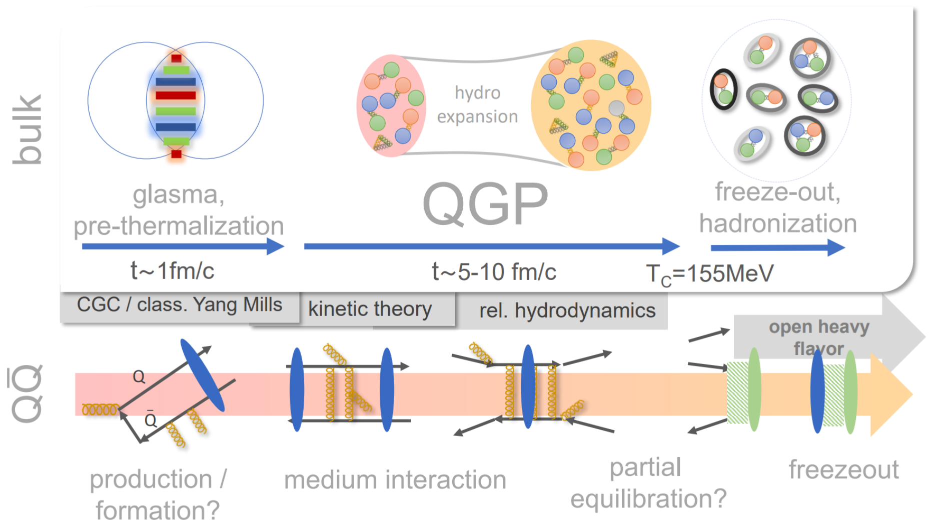


Figure 2.4: Stages of a heavy ion collision, including the different theories to describe them. [19]

The QCD phase diagram can be investigated experimentally using heavy ion collisions. For this, particle accelerators are used to accelerate heavy ions to relativistic velocities and collide them with other heavy ions. This is done using either collider or fixed target experiments, while colliders typically can achieve higher energies. On the other hand fixed target setups may have higher interaction rates.

In figure 2.4 different stages of a collision are shown. It starts of with two Lorentz contracted ions colliding with a certain overlap defined by the impact parameter b. While the two ions are colliding, the elliptical overlap region will have a very high energy density. Further interactions resulting in a so-called fireball. Depending on the energy and baryon density, this fireball may be composed of different states of matter, e.g. a quark-gluon plasma or baryon-dense matter. However, this fireball only lasts for a very short amount of time, approx. 10 fm/c. In case a quark-gluon plasma has been created, it directly expands and cools down, ultimately resulting in hadronization. The fireball matter which might have consisted of quasi deconfined quarks and gluons, is now recombining into color neutral hadrons. Given the chemical abundance of the hadrons, the correlated temperature is called the chemical freeze-out temperature, which can be measured in the experiment. This gives the possibility to determine the QCD phase diagram experimentally. The CBM experiment aims towards building a detector capable of measuring the QCD phase diagram in the area where the critical endpoint is expected (see fig. 2.3). Important signatures to characterize the created matter is electromagnetic radiation from the fireball, measurable e.g. by the reconstruction of di-leptons. Electromagnetic probes can penetrate the fireball as they do not interact strongly, thus carrying information from the dense matter to the experiment.

Chapter 3

mCBM

The miniCBM (mCBM) detector is the test setup for the SIS100 CBM experiment. The overall goal is to test and optimize each detector and especially the triggerless free-streaming data acquisition, the interplay between the detectors and online event reconstruction[20]. Data reconstruction algorithms can be tested and optimized on actual real data. The geometry for the mCBM including all subdetectors is show in figure 3.1, they are rotated by 25° relative to the beampipe.

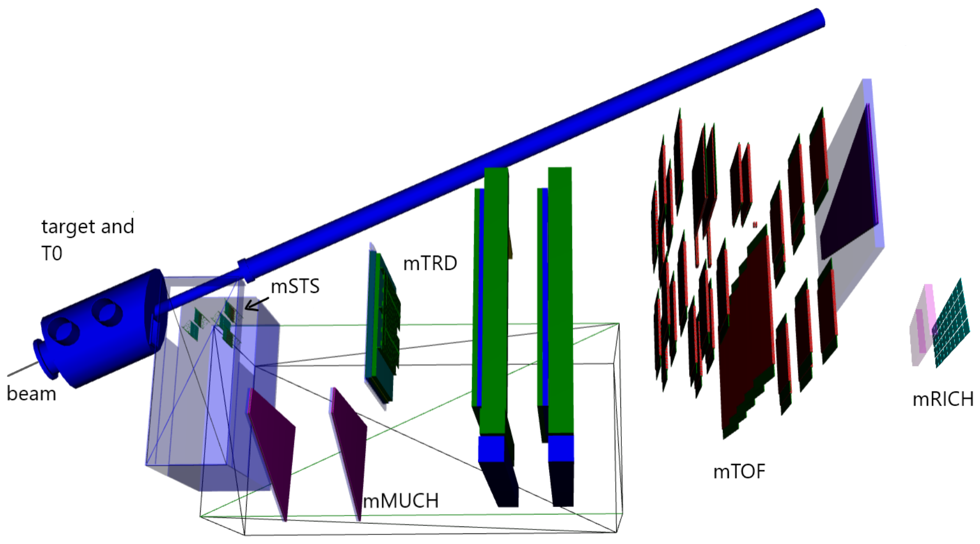


Figure 3.1: mCBM geometry setup (mcbm_beam_2022_06_16).

3.1 mRICH detector

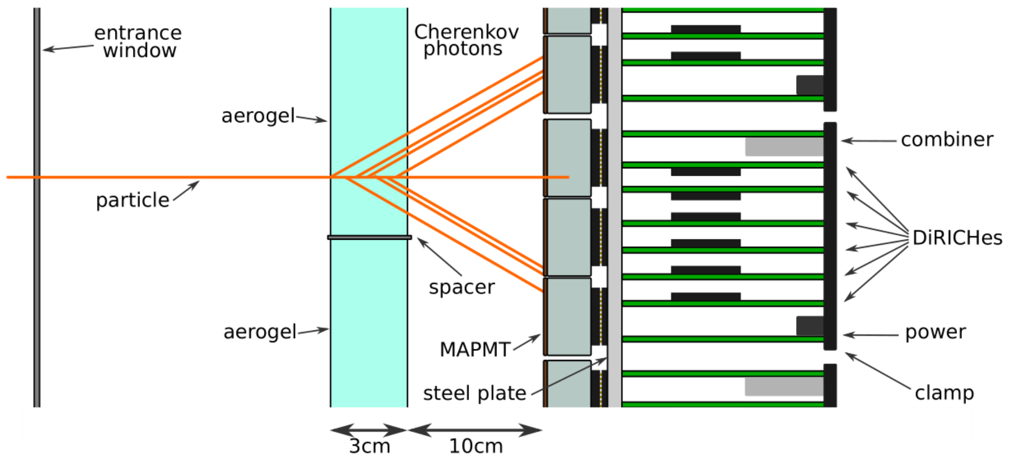


Figure 3.2: Schematic side view of the inner mRICH detector. This includes the aerogel, MAPMT's, readout electronics, support structures and a basic illustration of the produced Cherenkov photons. [21]

For the miniRICH (mRICH), a proximity focusing RICH (Ring-Imaging Cherenkov detector) with two aerogel blocks as radiator and 36 Hamamatsu H12700B-3 MAPMTs as photon detector are used (see Figure 3.2). It is placed directly behind the mTOF, this setup and the resulting information from both detectors should help to separate electrons from pions and fast pions from protons. The two 20x20x3 cm^3 aerogel blocks (see Figure 3.3) are stacked vertically connected via a 1 mm thick ABS spacer, together forming the full radiator. The aerogel is the same used in the CLAS12 RICH detector, measurement showed a refraction index of n=1.05 at 405 nm [22]. The aerogel blocks are hydrophilic, therefore the full detector is flushed with dry nitrogen. The aerogel and entrance window of the photodetector are separated with a gap of 10 cm. Each of the 36 MAPMTs consists of 8x8 pixels each, this results in a total amount of 2304 pixels for the full detector, with each pixel having the size of 6 mm x 6 mm. Pixels on the edges are a bit larger with 6.25 mm x 6.25 mm. Lastly, MAPMT's have a gap in between each other of size 1 mm. The overall acceptance of the photo detection plane is 47.6 cm x 21.3 cm.

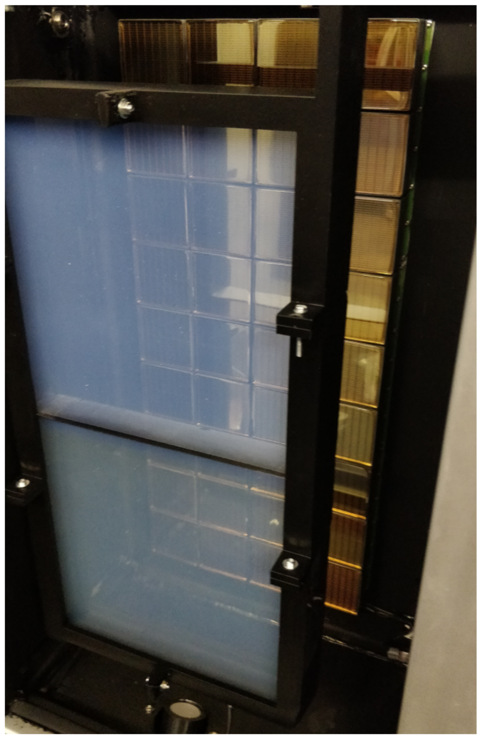


Figure 3.3: Inside of the mRICH, having two attached aerogel blocks and 36 MAPMTs behind it. [21]

3.1.1 Cherenkov radiation

The underlying effect all Cherenkov detectors are using, is the Cherenkov effect[23]. This effect occurs when a charged particle is traveling through a medium with a higher velocity, than photons inside this medium. The velocity of photons v_γ in a certain medium (equation 3.1) reduces, compared to the vacuum velocity c_0 based on the refraction index n.

$$v_\gamma = \frac{c_0}{n} \tag{3.1}$$

This is the reason why particles can be faster in the medium, compared to photons, whereas in the vacuum this can never happen since massive particles never reach the vacuum speed of light c_0. In general the refraction index n is energy dependent, based on the dispersion relation of the given material.

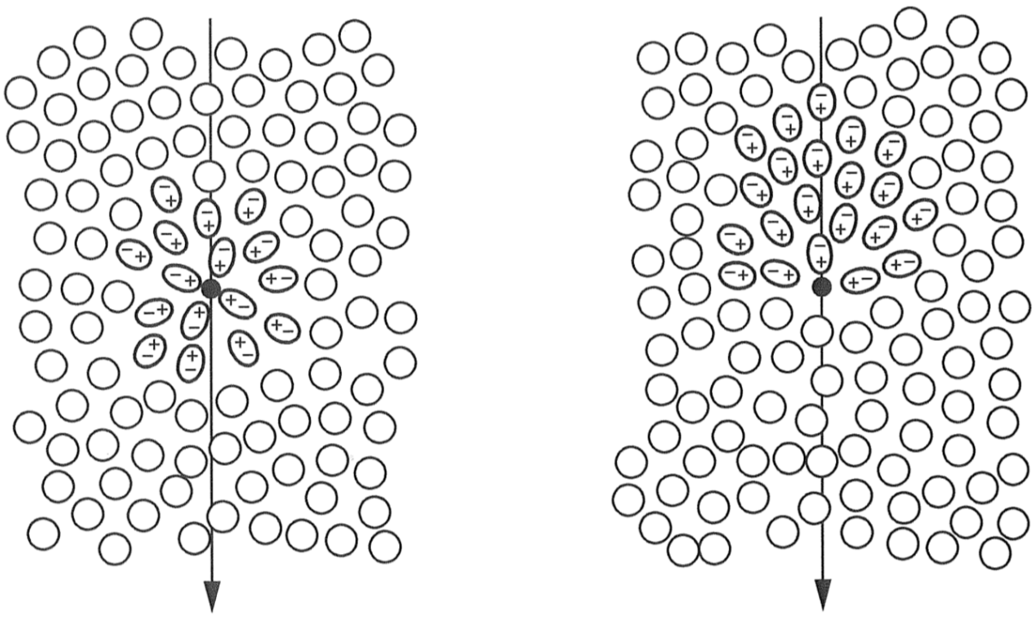


Figure 3.4: Left: a particle with $v_p < \frac{c_0}{n}$ causing a symmetrical polarization. Right: a particle with $v_p > \frac{c_0}{n}$ causing an asymmetrical polarization. [24]

When a charged particle is passing through a medium, it causes a local polarization of the surrounding medium (illustrated in fig. 3.4). This leads to excitation of the molecules in the medium, which results in energy emission in form of photons when those molecules return to their ground state. In the case of particle velocities $v_p < \frac{c_0}{n}$, this leads to a symmetrical polarization and on the outside view there is no constructive interference effects of the resulting wavefronts visible. However, if the particle has a velocity above this threshold $v_p > \frac{c_0}{n}$, the polarization is asymmetric. The result is a constructive interference of the emitted wavefronts, which causes Cherenkov radiation to be emitted with an angle θ_C.

$$cos(\theta_C) = \frac{1}{n\beta} \tag{3.2}$$

Given this and the fact that particles are limited by the speed of light with $\beta < 1$, a maximum angle can be calculated with the limit $\beta \to 1$.

$$cos(\theta_{max}) = \frac{1}{n} \tag{3.3}$$

Given the maximum angle in equation 3.3 and the threshold velocity β_{th} required to produce Cherenkov photons

$$\beta_{th} = \frac{1}{n} \tag{3.4}$$

the threshold energy for a charged particle with rest mass m_0 can be calculated.

$$\frac{E_{th}}{m_0 c^2} = \gamma_{th} = \frac{1}{\sqrt{1 - \frac{1}{n^2}}} = \frac{1}{sin(\theta_{max})} \tag{3.5}$$

Given the momentum of a particle and the opening angle of the Cherenkov cone (or ring radius in a detection plane), particles with different masses can be separated.

In CBM, the RICH detector will be used to identify electrons. A clean and efficient identification of those is necessary to access di-electrons as promising probe of the created dense QCD matter.

3.1.2 Time over Threshold

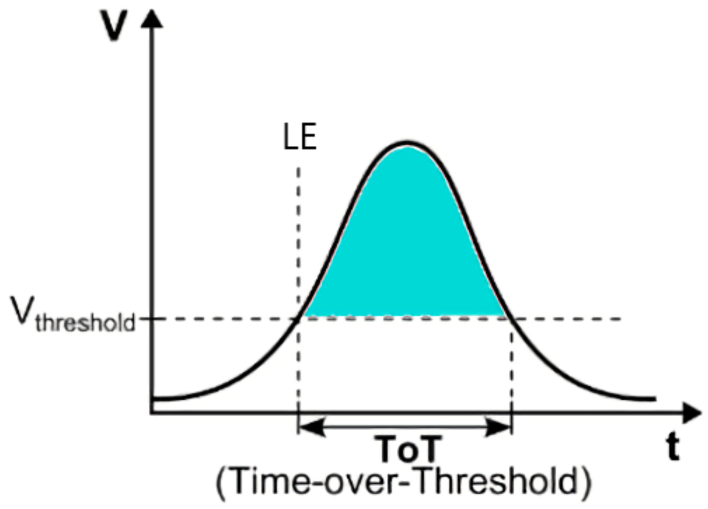


Figure 3.5: Illustration of the Time over Threshold calculation. With LE being the leading edge of the signal. (adapted from [25])

The data gathering of the mRICH detector starts with photons producing signals in the MAPMTs. The output signal of the MAPMT is a voltage and is processed by DiRICH boards, with each board processing signals from 32 MAPMT channels. Each DiRICH contains voltage threshold settings for all channels and the Time-to-Digital converter (TCD). All signals are amplified in the DiRICH, since the TCD requires larger voltages. The TCD creates timestamps, those contain the time information of the rising and falling edge of the signal, which are the times the signal rises above or falls below the threshold voltage respectivly. Setting a voltage threshold for the DiRICH, the Time over Threshold (ToT) can be calculated from the rising and falling edge timestamps of the MAPMT signals (illustrated in fig. 3.5). For more detailed information abount the full readout chain see [21]. The data gathered for each MAPMT channel is recorded in a set of time slices, those are saved to a Time Slice Archive (TSA-file) file. This data has to be unpacked to proceed with the analysis. The resulting information are so called digis, including the address, the leading edge (LE) time information and the ToT. The ToT is an important first information to reduce the amount of noise in the mRICH detector.

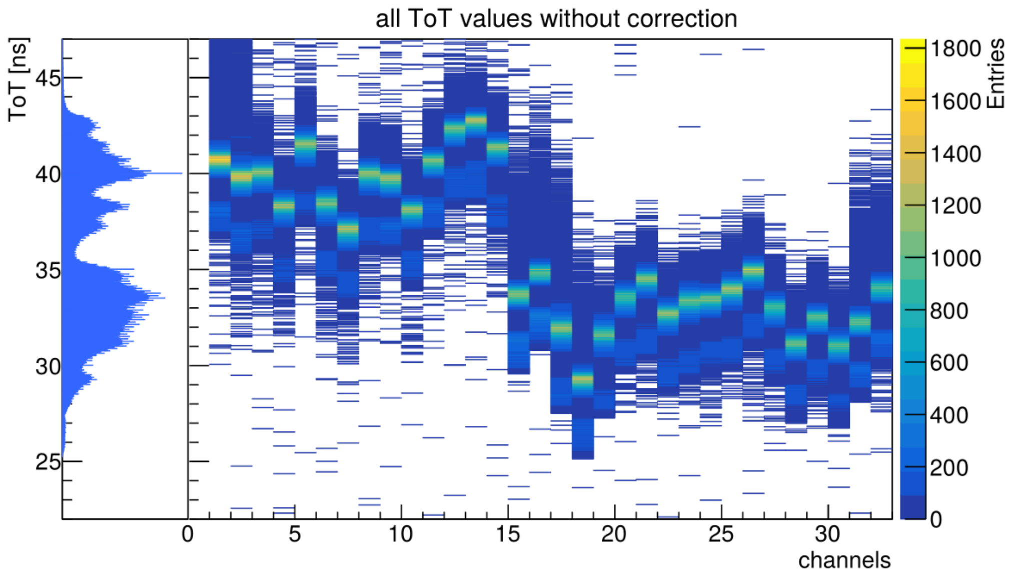


Figure 3.6: Uncorrected ToT spectra for DiRICH 0x7261, run 831 mCBM beamtime 2020 [21].

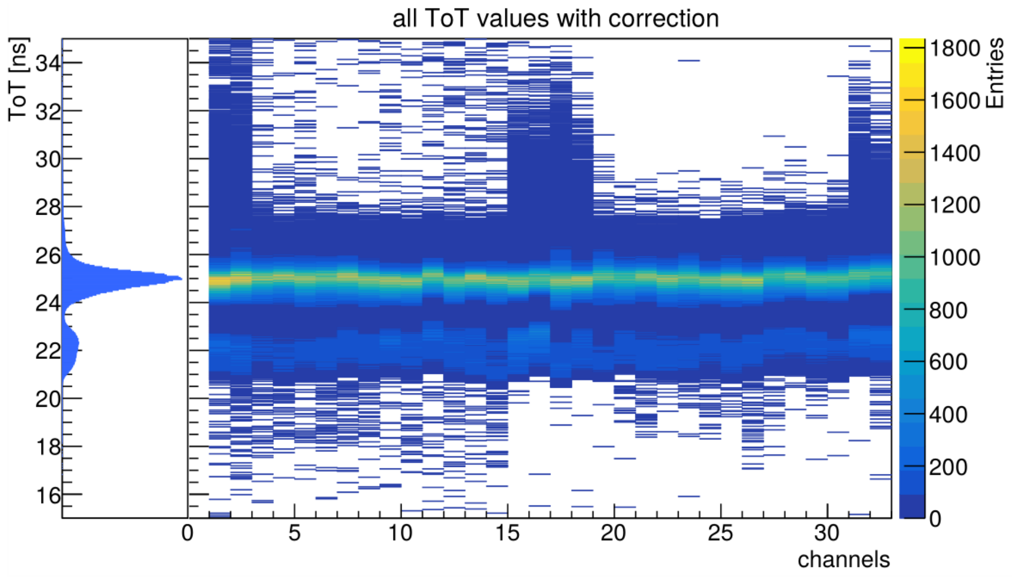


Figure 3.7: Corrected ToT spectra for the above DiRICH and run [21].

For the DiRICH 0x7261 board the ToT's for all 32 channels are shown in figure 3.6. For each individual channel, there is a two peak structure visible, but overall the ToT distributions for each channel are shifted. Applying an offset correction for each channel, shows this two peak structure also in the projection of all channels (see figure 3.7 left side). Given this corrected peak structure, a common threshold for all channels and boards can be applied to the ToT, removing the small peak. The small peak is attributed to result mainly from capacitive crosstalk in the last dynode of the MAPMTs [26]. This already reduces the amount of noise significantly. Given a found event using the EventBuilder[27], RICH digis are then converted to actual RICH hits using the ToT value and cut. Those are used for further analysis, i.e. the ring finding process. Typical event building criteria are 1 digi in the T0 and multiple hits in the mTOF and mRICH.

3.1.3 ICD correction

All individual channels have some delay, due to signal routing in the electronics, this is called the Inner Channel Delay (ICD). To correct the differences for each channel, an ICD correction is calculated. This is done using the fact that for an actual Cherenkov ring, all true ring hits have the same time. Rings are searched for using the Hough ring finder (described in section 3.1.4). The correction is calculated based on the ring time, i.e. the mean time of all ring hits, and the hits matched to it using the hit address and time. Repeating the process a couple of times, improves the timing for each channel (see figure 3.8). The accuracy of the method is tied to the ring finder performance, i.e. finding the rings correctly. This correction (see [21] for details) is based on run 836 and can be used in the following for all runs and beamtimes, because it is stable in time.

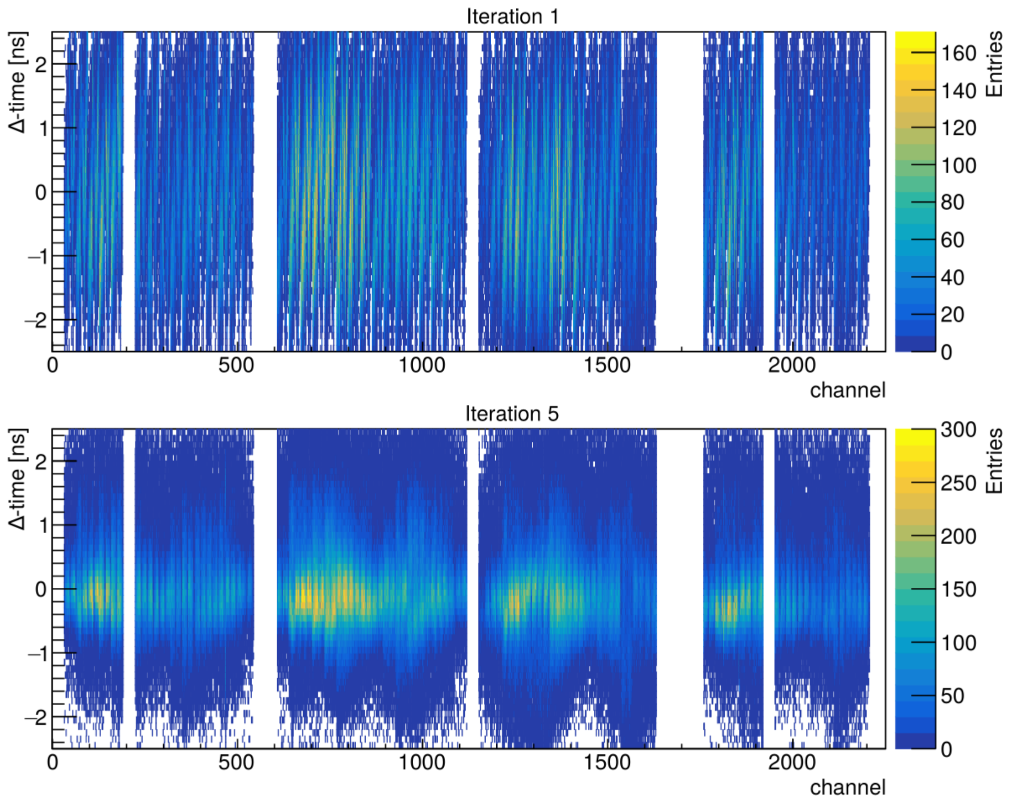


Figure 3.8: ICD correction after the 1st and 5th iteration. Shown are time differences of a hit and the meantime of the corresponding ring for all channels in the mRICH. [21].

3.1.4 Hough ring finder

Event reconstruction of data in the mRICH detector means finding ring structures and fitting rings to them, i.e. gaining information about the position and radius for each ring. This approach is used to contribute to the overall particle identification (PID) of the full detector. The implemented ring finder algorithm [28] is based on the Hough Transform (HT) [29]. The HT in general is very effective in the recognition of shapes such as straight lines, circles or ellipses in a picture. The HT transforms the information in the detector specific coordinate system into a system with new dimensions depending on the required shape, for rings the ring center and radius are e.g. stored. In general given three points, a circle can always be constructed from those, i.e.

the parameters $(x_{center}, y_{center}, R)$ can be extracted.

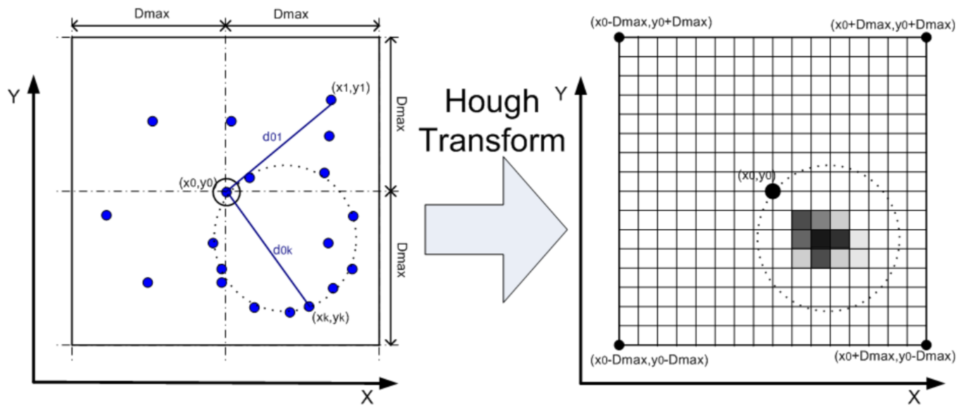


Figure 3.9: Left: Local search area with D_{max} being the maximum diameter of the ring and (x0, y0) being the initial hit coordinates. Right: The resulting ring center candidate distribution. [28].

For the three hits method implemented, the algorithm is iterating over all combinations[1], calculating the center and radius of the calculated ring and filling those values into two histograms. The first one contains the potential ring centers in a 2D histogram, while the second one stores the ring radius in a 1D histogram. In those histograms a peak structure will develop (fig. 3.9), corresponding to a ring candidate. If those peaks are higher than a certain threshold, the candidate ring is accepted.

[1]To save computation time, those combinations are restricted by factors such as distances between hits.

3.2 Simulated data

The Ultra relativistic Quantum Molecular Dynamics (UrQMD [30]) software is used to simulate events for a certain heavy ion collision system. Given a geometry (e.g. fig 3.1), all particles resulting from UrQMD simulated collisions are tracked through the geometry using GEANT3 [31]. GEANT3 is simulating the interactions between the particles and the (detector) meterial. This results in Monte Carlo (MC) information, namely particle track information and points for a given detector, where particles passed through. For example a RichPoint is a point in the photo detector plane of the mRICH, where a photon or a charged particle passed through. Using this RichPoint information, RichDigis are constructed using additional parameters to adjust for example the number of hits for a given set of points due to crosstalk, adding noise and adding time smearing of the hits. Adjusting those parameters will be discussed in chapter 4. For the simulation, there are two different event modes

- Timebased, emulating the behavior of the free streaming data and using the same EventBuilder as for real data analysis.
- Eventbased, using the individual events from the UrQMD process, having full knowledge over all events. Only available for simulated data.

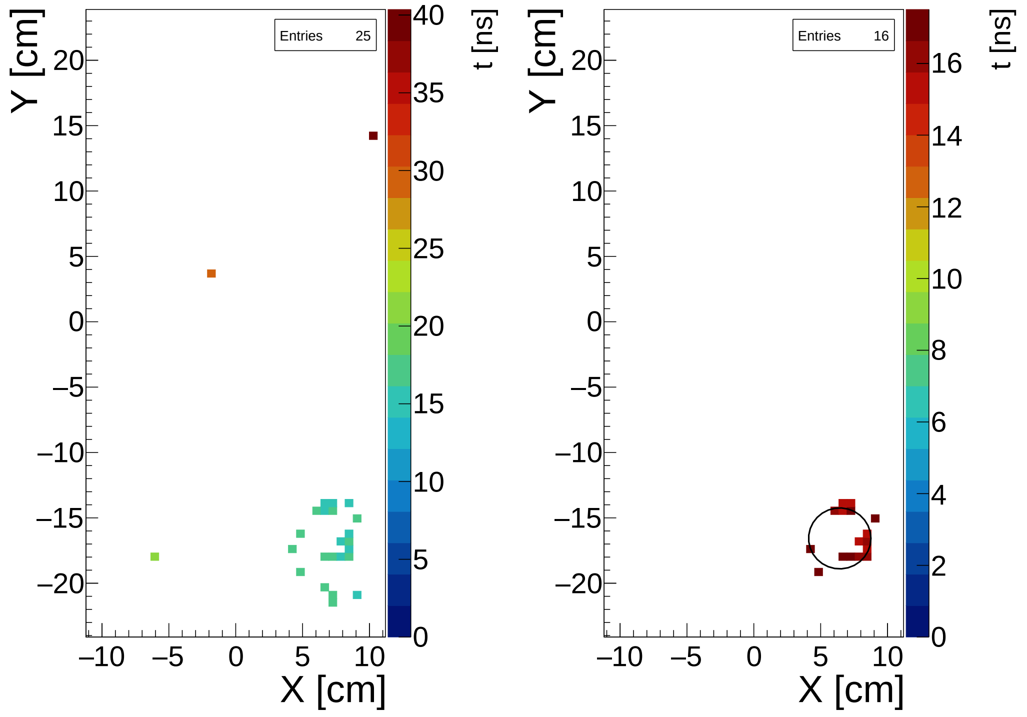
For the timebased simulations, the connection between MC data, found events and RichHits/rings is not fully implemented yet. Therefore, all analysis using MC data can only be done in the eventbased mode, such as calculating if a ring was properly found.

Chapter 4

Simulation adjustment & ML preprocess

In the mCBM detector setup, charged particles pass directly through the mRICH detector. These particles produce additional digis in the MAPMT pixels, which lead to typical digi clusters around the position in the detector where the charged particle passed through. Those patterns can also be seen in other detectors such as the Belle II RICH detector[32]. The main contributor for this is the MAPMT glass window, in which additional Cherenkov photons are produced. Another contributor are photons produced inside the nitrogen flushed gab between aerogel and MAPMT. The opening angle of Cherenkov photons for $\beta = 1$ particles in N_2 is $\theta = 2.4°$ only, this ring would have a radius smaller than the side length of a MAPMT pixel. Therefore, those photons typically only cause a single digi in the center of a ring structure. For heavy particles passing through, it happens quite often that nearly all pixels in one MAPMT light up. In general these clusters can be anywhere in the mRICH plane, but the clusters we are interested in, are those inside ring structures. This might cause wrongly reconstructed rings seen e.g. in figure 4.1. The Hough Transform ring finder is not suited well for such patterns. This results in the observation of more smaller rings (typical half the radius of the actual ring) constructed from centerhits and ringhits. In addition, for those cases the actual full ring is mostly not found.

Because the typical structures of rings differ from the charged particle clusters, a supervised machine learning approach is used in order to classify hits as either noise or actual ringhits, which we are interested in. The two core parts of supervised machine learning is the data and the model used to extract and process information. Considering the data part, this itself also splits into two parts. The first one are the so called labeled data, having precise information on "good" or "bad" hits. Here, a sufficient amount of data is needed. In the case of particle physics this is no problem at all, because there is access to simulations and therefore basically an unlimited amount of data available with detailed information. The second is the data quality, which in our case means how close the simulation is to the real data. This will be discussed first in this chapter. This is a crucial task, since for this work a fully supervised approach is used. Lastly, one needs to find an appropriate model to extract and gain information to classify hits. Different models and the selection are discussed in chapter 5.



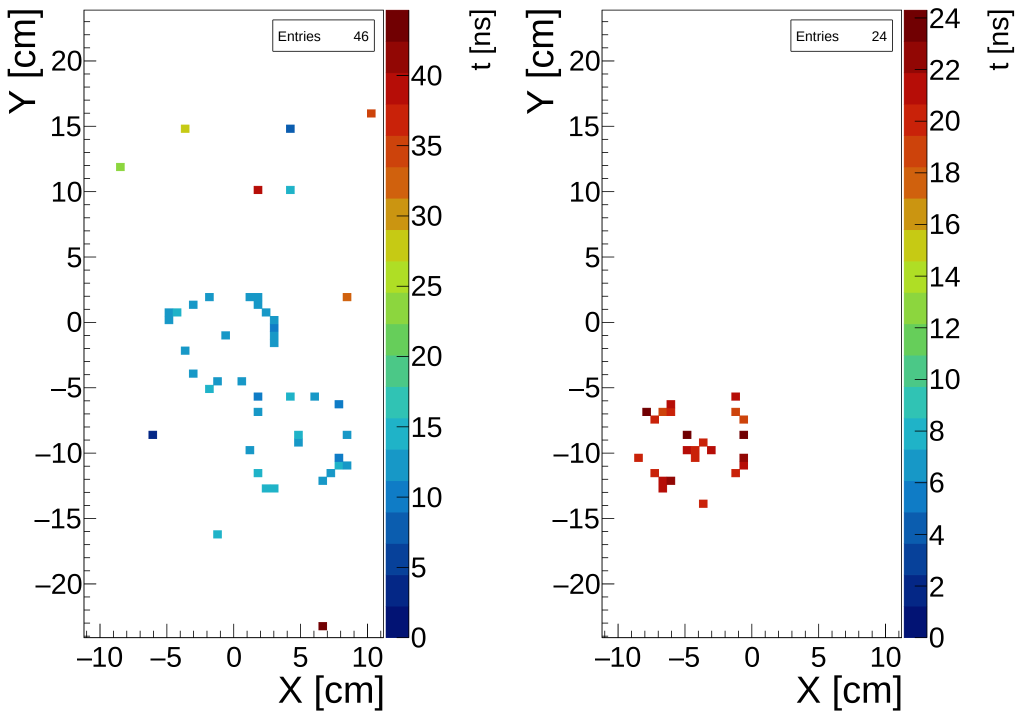
(a) Real data event with all hits inside this event.

(b) Real data event with the found ring and all hits matched to it.

Figure 4.1: Single event display for a real data event, showing a typical wrongly reconstructed ring caused by hits in the center.

4.1 Simulation parameter adjustment

Common ring structures are shown in figure 4.2 for current simulation and real data events. Simulation events have mostly 1-2 hit in the ring center, while real data events have very commonly 4-6 centerhits, caused by charged particles passing through the MAPMT's.



(a) Simulation data event.

(b) Real data event.

Figure 4.2: Single event displays showing typical ring structures for simulation and real data.

In order get a more realistic detector response, the simulation has to be adjusted. For this, the simulation is also running in timebased mode, to eliminate differences coming from the EventBuilder. All run information and settings are summarized in table 4.1.

data	simulation	real
beam	Au @ 1.24 AGeV	Au(69+) @ 1.23 AGeV
beamtime runId	-	2511
UrQMD files / Timeslices	urqmd.auau.1.24gev.mbias files 701 to 800	first 200 timeslices
mCBM target	2.5 mm thick Au target	
mCBM geometry setup	mcbm_beam_2022_06_16_gold	
Reference Derector	mTOF	
Detectors for event building	mTOF, mRICH	
Trigger Window mTOF	-60ns to 60ns	
Trigger Window mRICH	-60ns to 60ns	
Trigger minimum Digis mTOF	4	
Trigger minimum Digis mRICH	10	
mRICH ToT-cut	-	23.7ns to 30.0ns
PixelDeadTime	30ns	slightly more than 30ns

Table 4.1: All run information for timebased simulation and real data.

The main difference is the ToT-cut for the mRICH detector, which is not needed for simulations, and the dead time for each pixel. This is the time needed for each pixel to process a new digi, after creating one earlier (e.g. 30ns earlier). The dead time at the actual detector is related to the readout electronics. Furthermore, the simulation contains more parameters, which can be adjusted. In summary these are:

- CollectionEfficiency, a parameter adjusting the efficiency with which a photo electron is captured by the dynode system, thus adjusting the probability a RichPoint will result in a digi. This grants control of the amount of ring hits a given ring has.
- TimeResolution, a time smearing parameter to add a time uncertainty to each digi.
- NoiseDigiRate, to adjust the amount of random digis produced in the digitization process to simulate detector noise.
- Aerogel refraction index, since the properties of the used aerogel for the mCBM are not fully known, this parameter was also adjusted. The full dispersion relation is unknown and a fixed value for all energies is

used. This adjusts the ring radius, threshold energies and the amount of produced photons for a given charged particle passing through the radiator.

- CenterNoiseDigiRate, adjust the average amount of center noise hits produced in the digitization process.

For the mRICH simulation, the glass window in front of the MAPMT is not used, because there are several problems when running simulations including the glass window. First, photons often get 'trapped' inside the glass window and bounce back and forth, eventually reaching the iteration limit, which aborts the current simulated event. This may end up having events with its particles not fully transported/simulated, in addition the computation time is approximately 3 times higher compared to simulations without the glass window. Lastly, simulations including the glass window have shown that the additional amount of center hits is still far away from what is observed in real data. The glass window is therefore no solution in order to get the simulation more realistic, so a different approach is used. It is expected that the charged particles passing through the detector are responsible for the characteristic center hit pattern or are at least the main contributor. An attempt is tried to artificially produce noise digis in a region around pixels where charged particles passed through. The region for now is effectively a 5x5 kernel with the digi, generated from the charged particle, in the center. In addition, this artificial noise production is chosen to be limited to a single MAPMT for each charged particle. The indices X_c, Y_c and the time t_c are from the digis generated from a charged particle. With the given distance definition in equation 4.1, a noise digi with position indices X, Y is produced with the probability $P_c(X, Y)$ (eq. 4.2) and time $t_c(X, Y)$ (eq. 4.3) with $\mathcal{N}$ being the normal distribution. The parameter φ is the CenterNoiseDigiRate and needs to be adjusted according to what is observed in real data.

$$r_c(X,Y) = \sqrt{(X_c - X)^2 + (Y_c - Y)^2} \qquad X_c, X, Y_c, Y \in \mathbb{N}_0 \tag{4.1}$$

$$P_c(X,Y) = \begin{cases} \frac{\varphi}{r_c(X,Y)}, & \text{if } |X_c - X|, |Y_c - Y| \leq 2. \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

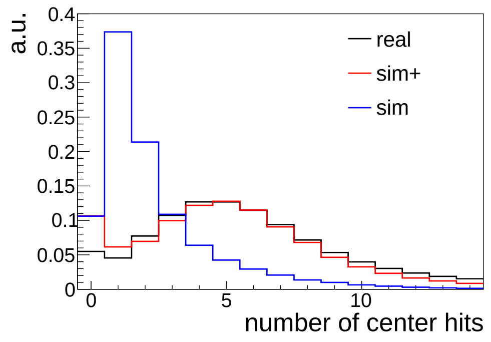
$$t_c(X,Y) = t_c + \mathcal{N}(0,1) \tag{4.3}$$

It should be noted, that this artificial noise production is for now completely independent of the particle properties (mass, momentum, ...). The adjusted parameters are summarized in table 4.2. The TimeResolution did not change, but might also need further adjustment in future, since the ICD correction was calculated based on the ring finder without neural network noise removal.

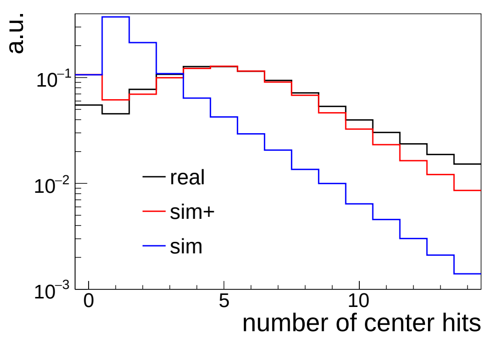
Setting	Simulation	Simulation+
CollectionEfficiency	0.45	0.45
TimeResolution	1.0	1.0
Aerogel refraction index	1.05	1.046
CenterNoiseDigiRate	-	0.40
NoiseDigiRate	150.0	150.0

Table 4.2: Simulation parameters, comparing the previous simulation to the adjusted simulation parameters ("Simulation+").

Comparing real data (real) to the previous simulation (sim) and the simulation with adjusted parameters and artificial noise (sim+), shows now strong similarities between real and sim+. The number of hits inside a ring (center hits), are now much more comparable to the real data as seen in figure 4.3. The real data is still slightly different for low and high amount of center hits. For the neural network training process, it is crucial that realistic patterns are included in the training data, not only the amount of hits. The previous simulation had also rings with a lot of center hits, but those patterns were certainly not the same as the ones artificially produced now or observed in real data. Most of the rings having a lot of center hits for the previous simulation, are rings overlapping with other ring structures.



(a) linear scale.



(b) log scale.

Figure 4.3: Comparing the number of centerhits for real, new simulation and old simulation data with adjusted parameters and additional noise added around hits

Comparing the amount of ring hits for each ring in figure 4.4, the adjusted simulation matches slightly more to real data, than the previous simulation.

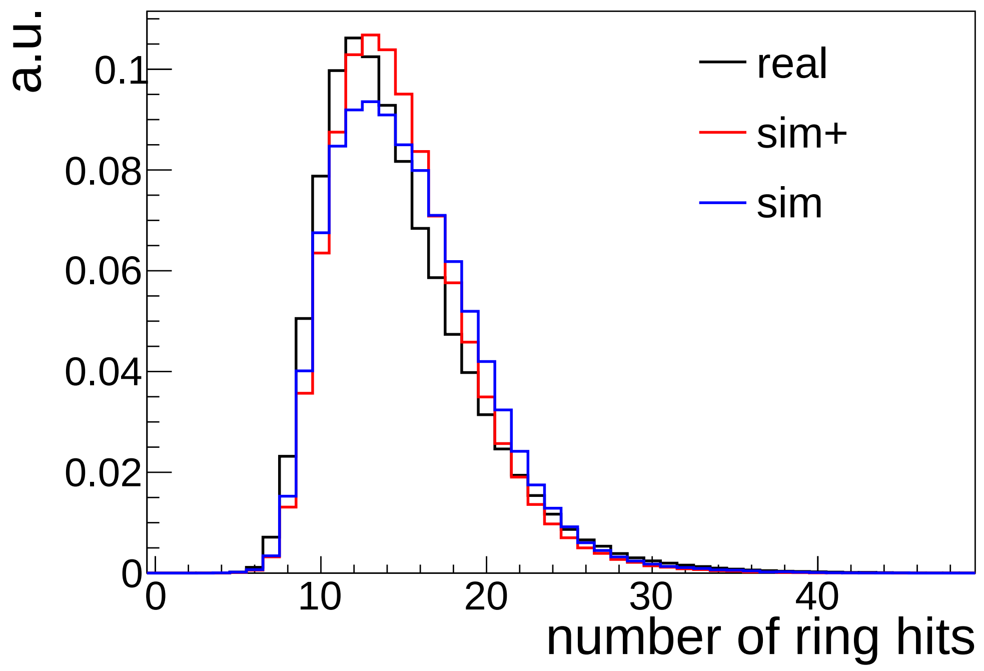
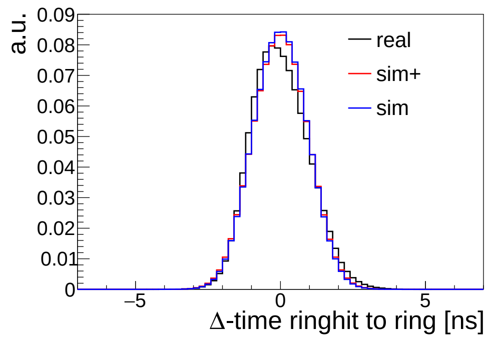
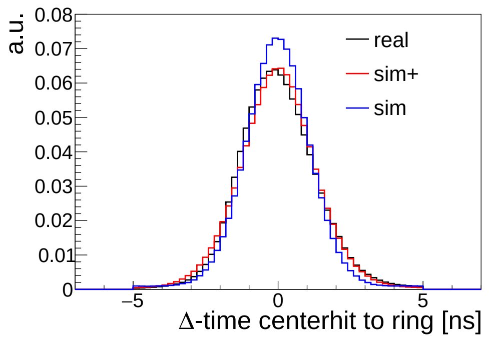


Figure 4.4: Comparing the number of ringhits.

The time difference for hits in found rings to the meantime of the ring (fig. 4.5a) shows a good accordance, except for the slight shift of the peak for the real data. The same is seen for the center hits in figure 4.5b; in general they are very comparable concerning the time values.



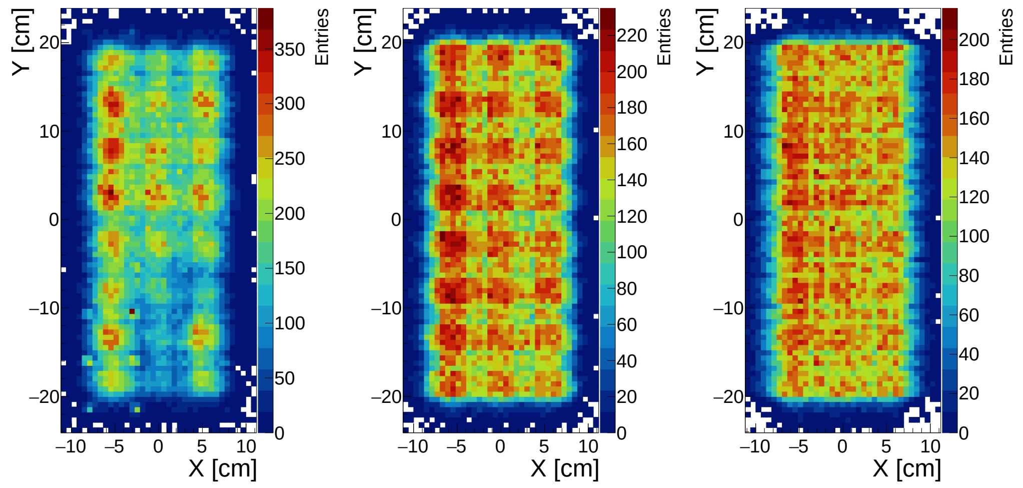
(a) Time difference of ring hits to the meantime of a found ring.



(b) Time difference of hits to the found ring (radius R) inside 0.8R and with a time difference of $|\Delta t| \leq 5$ns.

Figure 4.5: Comparison of ring (left) and center hits (right) with respect to the time difference to the mean time of a ring for real, new simulation and old simulation data.

The changes have a large effect on the distribution of found ring centers (fig. 4.6). For real data seen in figure 4.6a, a lot of rings and their ring centers are found in between MAPMT's, which results in this cluster structure. The same pattern is now also observed in the simulation (fig. 4.6b), when adding additional noise from charged particles. This is very different from the previous simulation (fig. 4.6c), where rings were found very homogeneously distributed across the whole mRICH plane. This comparison shows that this noise, produced by charged particles, is at least partially responsible for the pattern differences between old simulation and real data.



(a) real. (b) sim+. (c) sim.

Figure 4.6: Ring center positions in the mRICH plane for real, new simulation and old simulation data.

Comparing the ring radius distributions (fig. 4.7), the sim+ distribution is shifted to lower radii, which comes from the change of the aerogel refraction index and is now matching to the real data. Even more important is the appearance of the left 'shoulder' in the new simulation data, which also proves the connection of noise produced by charged particle to this anomaly.

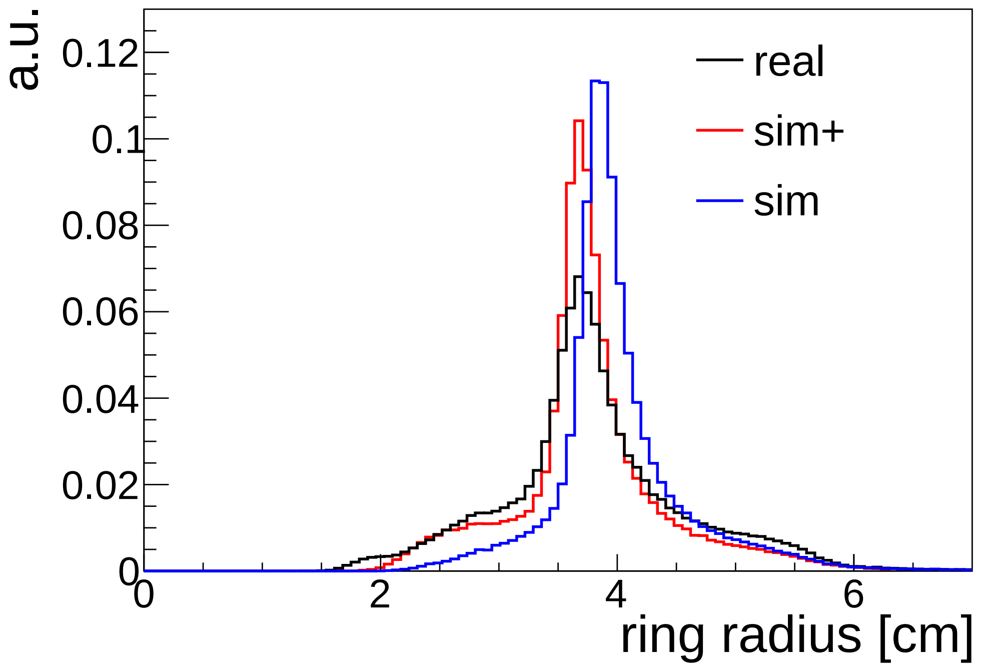
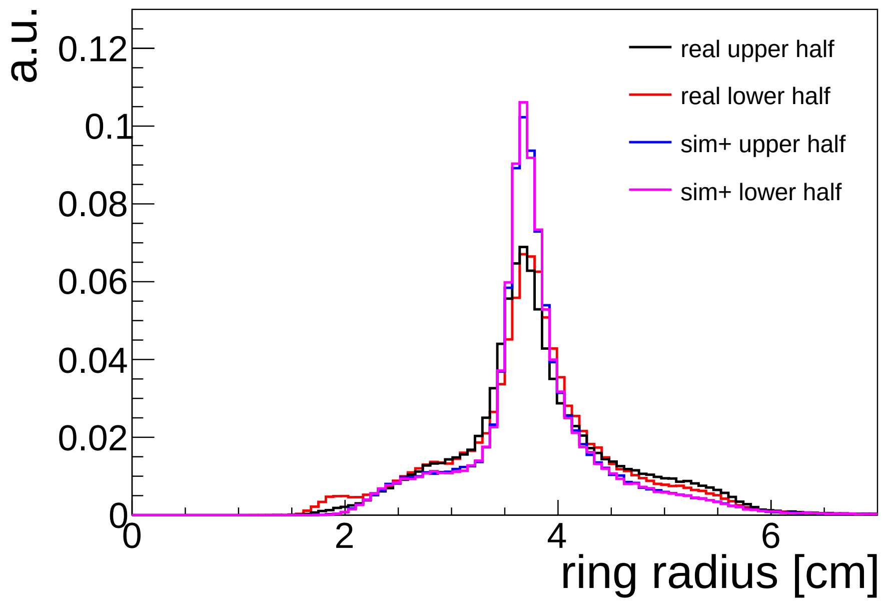
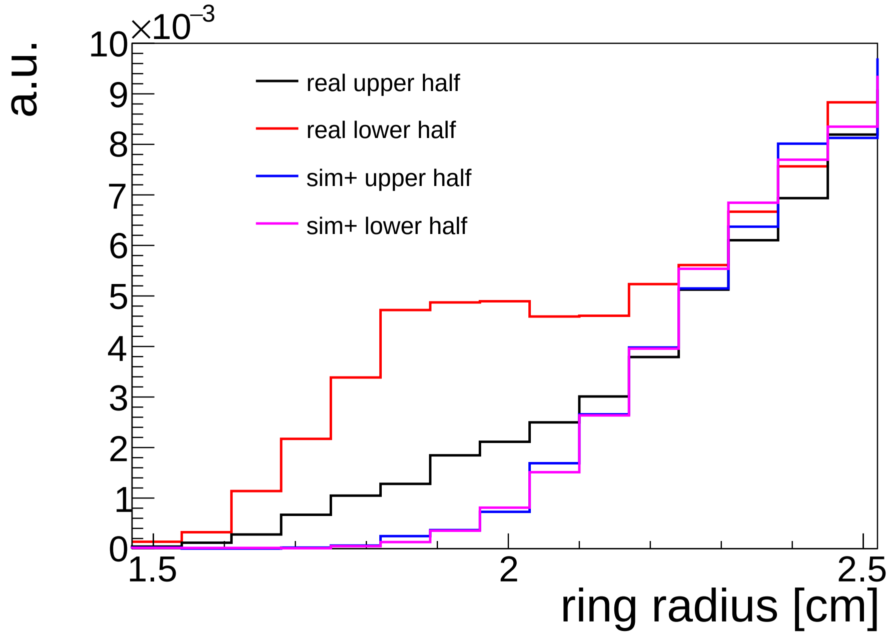


Figure 4.7: Comparing ring radius distributions for real, old simulation and new simulation data.

For the mRICH, two separate aerogel blocks are used as the radiator material. From those, the lower one is older and shows distinct differences for the amount of rings (fig. 4.6a), a slight shift in the ring radius distribution (fig. 4.8a) and an additional peak for very low radii (fig. 4.8b). The first point is most likely correlated to a reduced transmittance from the aged aerogel block. The shift of the ring radius peak might come from a change in the refraction index for the lower aerogel block, although this is only a very small difference. Lastly the amount of very small rings with radii in the region of 1.9 cm, are drastically higher compared to the upper half. This might come from MAPMT's fully lighting up, when a heavy charged particle passes through. Looking through a lot of single event displays, it seems that this occurs more often in the lower half, which would explain this additional peak. It can also be seen in the ring center distribution (fig. 4.6a), where in the lower left in contrast to the overall pattern, ring centers are enhanced in the center of all 6 MAPMTs in the lower left backplane.



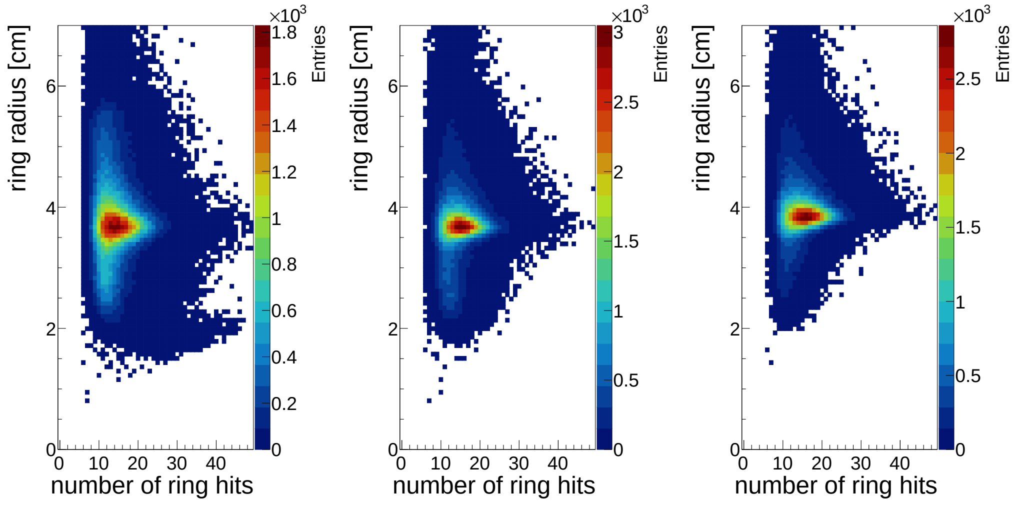
(a)



(b)

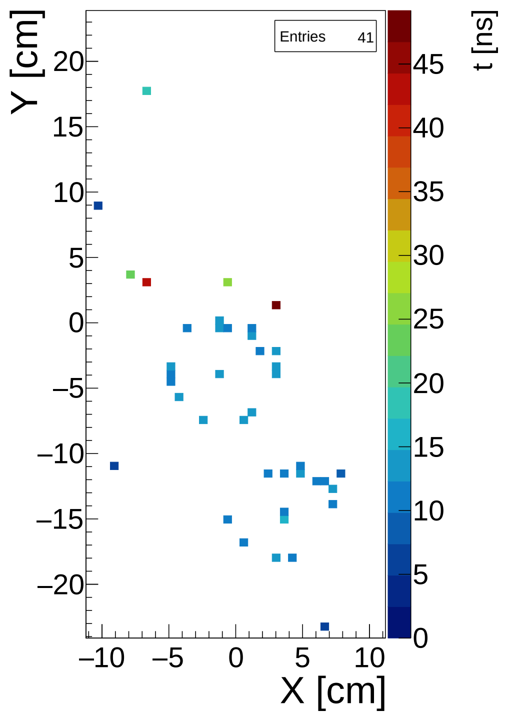
Figure 4.8: Comparing ring radius distributions for the upper and lower half of the mRICH plane for real and sim+ data.

The same effect can be seen when plotting the ring radius versus the number of hits per ring (see figure 4.9). Many rings with small radius and a high amount of attached hits are seen in real data. Since fully lighting up MAPMT's are not existing in simulations, those patterns and the resulting anomalies in the distributions can't be seen for any of the simulations.
A direct comparison of the simulations is shown in the single events displays in figure 4.10. The new simulation now includes rings with a lot more hits inside the ring structures compared to the previous simulation. The actual ring hits are different because of the randomness in the digitization process, which generates digis from RichPoints and also the random noise. Single event displays for the old and new simulation, together with the found rings are shown in Appendix B.1 and B.2.

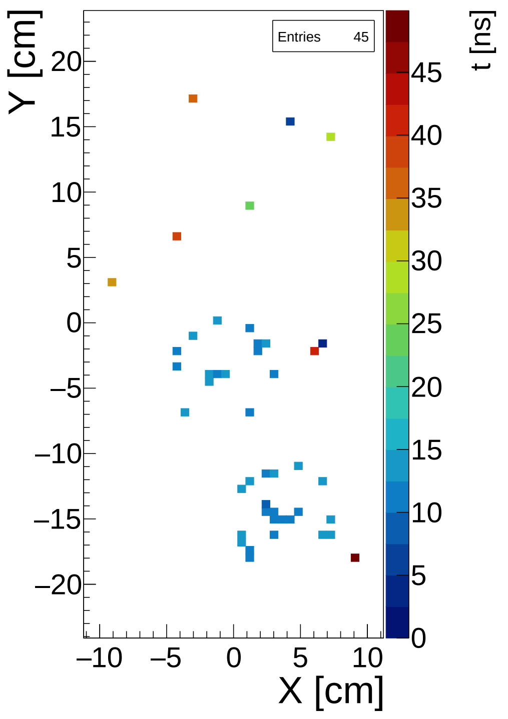


(a) real. (b) sim+. (c) sim.

Figure 4.9: Ring radius vs. number of hits per ring for real, new simulation and old simulation data.



(a) sim.



(b) sim+.

Figure 4.10: Single event displays showing typical ring structures for the old and new simulation with artificial noise added.

4.2 Data labeling

The labeling process requires Monte Carlo simulation information, in order to classify hits as either true hits or noise (hits). Those are defined as:

- True hit, a RICH hit containing at least 1 Cherenkov photon produced inside the aerogel volume.
- Noise hit, hits which are not a true hit, therefore containing no Cherenkov photon produced inside the aerogel volume.

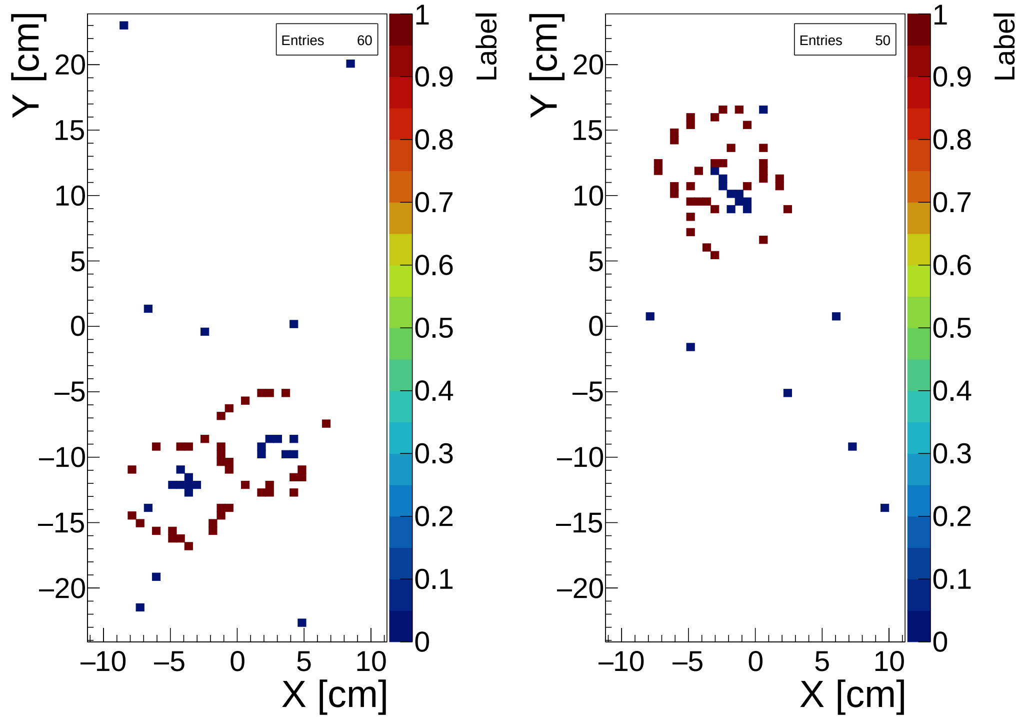


Figure 4.11: Single event displays showing hits labeled for the new simulation including artificial noise (true hit = 1, noise hit = 0).

Examples on how the labeling exactly looks like in single event displays, are shown in figure 4.11. Without centerhits and random noise very clean ring structures are left (red pixels). It is important to note that in general it is more desirable to preserve ring hits, than to remove more noise at the cost of ring hits. This is also the reason for asking only for 1 Cherenkov photon produced inside the aerogel volume. The task is now, to use this labeled data for the ML training process. Afterwards, the trained model is used to predict in data which hits are noise and can be removed, to simplify the task for the Hough Transform ring finder.

4.3 Time windows

The CNN (see section 1.2) requires that each pixel has one unique input for each inference cycle. A previous study [33] has shown that using explicit time values for each input considerably increases the classification performance. However, this is a problem in case of multiple inputs for the same pixel in one event. For example, when using full CbmEvents, it can happen that there are multiple hits at different times for the same pixel in this CbmEvent. This is where the PixelDeadTime from table 4.1 is used to our advantage. Since this dead time is for both real and simulation data $\geq$ 30ns, a sliding time window with a time length $<$ 30ns is used. This window processes through the already time sorted hit collection array and creates an input for the neural network if there are a certain number of hits inside a given time window. A basic pseudocode for the time window processing, including the neural network application is shown in Algorithm 1.

Algorithm 1 Sliding time windows algorithm including neural net inference.

```
Require: time window length, $t_L$
Require: time sorted hit collection, hit_collection
  $anchorIndex = 0$
  while $anchorIndex <$ number of hits in hit_collection do
      $t_{anchor}$ = hit_collection[$anchorIndex$] $\rightarrow$ Get Time
      $N$ = number of hits inside the time window $t_{anchor} \leq t \leq t_{anchor} + t_L$
      if $N \geq 7$ then
          pass all hits in this time window into the NN [1]
          label hit according to the NN output as a true hit or noise
          $anchorIndex \leftarrow anchorIndex + N$
      else
          label hit_collection[$anchorIndex$] as noise
          $anchorIndex \leftarrow anchorIndex + 1$
      end if
  end while
```

For now these windows are chosen to be non overlapping, in order so save computation time. This approach currently requires a fixed time window

[1]Having all time values incremented by 1ns, to differentiate the anchor hit from empty pixels.

length, which is chosen by minimizing the amount of rings at the end of a time window. The reason for this is to avoid having hits from ring structures in two separate time windows. For now a time length of 25 ns is chosen. When using this value, the majority of rings and ring hits are found at times < 15ns inside the time window (fig. 4.12), which avoids having too many ring structures split up into different time windows. For example, when using a time window length of 5 ns, a lot of ring structures will be split into different time windows. Since there are approximately 100 times more rings in those time region compared to > 15ns, more ring structure splits into different time windows will happen, which should be avoided.

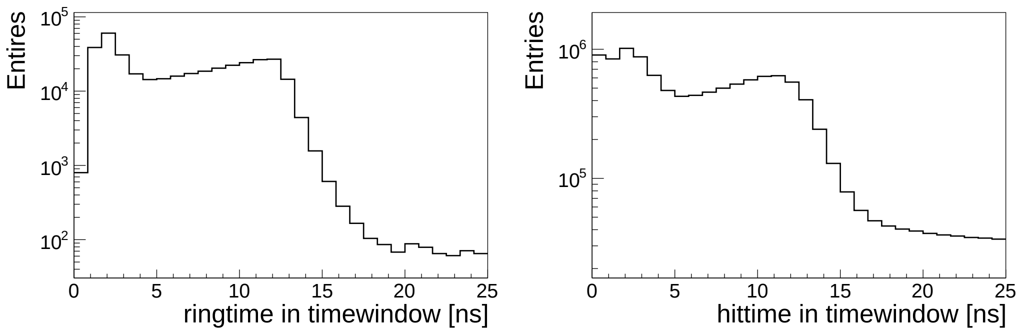


Figure 4.12: Ring time and hit time in a time window with 25 ns length.

Chapter 5

Neural network architecture

After having available labeled data and a processing workflow of hits via sliding time windows, the last missing part is the neural network itself. The TMVA [34] machine learning framework is currently the only option[1]to get a machine learning model properly running in the CbmRoot framework [35].

This comes with restrictions to the model architecture when using convolution layers. The main restrictions are: there is no up sampling layer or transpose convolution layer, only sequential models and a restriction concerning the use of a certain loss function for the regression task (only mean squared error available). So the only option currently within CbmRoot is, to simply stack convolution layers without any down/up sampling. This is definitely far away from current state-of-the-art CNN models, which are using for example residual connections [38] or inception blocks [39]. Nevertheless, this simple model approach has already shown good results on simulated data [33]. It gives insight on how well it actually works by analyzing simulation and real data directly in CbmRoot. In order to find a suitable model, the TensorFlow/Keras [40, 41] framework is used. The training is more convenient having tools like TensorBoard [42] and Weights & Biases [43] available, to keep track of all the different architectures and evaluation metrics. After finding an architecture, all different convolution layer settings are used in the TMVA CNN model and the training process is done only one more time in the TMVA framework.

[1]Current discussions are ongoing to integrate machine learning models from other frameworks (TensorFlow, Keras, PyTorch, ...), using ONNX [36] and ONNX Runtime [37].

5.1 Stacking convolution layers

Having these restrictions on the CNN architecture, the available options left are to vary the number of convolution layers and the settings for each layer. Namely, the amount of filters, filter sizes and activation functions for each individual layer. Specifically for the hidden layers, the ReLU activation function is always used. It is commonly used for hidden layers in most machine learning architectures, since it is very effective and computationally efficient. In order to always keep the input/output the same size of 72×32 (total number of MAPMT pixels), the padding for each convolution layer is set to 'same'. The training and test data sets consist of approximately 445×10^3 time windows each. The training data is the same used in the previous chapter, summarized in table 4.1 using the sim+ parameters from table 4.2. All models are tested on a separate simulation dataset, using the UrQMD files 801-900 instead of the files from table 4.1. From the training set, 20% is used for the validation data. These are used to keep track of the loss function in the train process and for an early stop. When the early stop patience value is reached, the parameters from the epoch having the best validation loss are saved and used for the final evaluation on the test data set. Very important to note is, that inputs having the value zero, also have this as the target value (pixels with no hit). This is a major downside of using CNN's for such sparse input data[2], since these zero targets also have to be trained. Therefore, quite some parameters are specifically trained to map from a zero input to a zero output. This results in a lot of 'useless' parameters, since the only output pixels for interest, are those who had a non-zero input (a hit). It also means, that the network will optimize in the first epochs towards this empty background, since this gains the most loss minimization. Only after a few epochs it optimizes towards getting the true hits reconstructed. When calculating the ROC (receiver operating characteristic) curve and confusion matrix values, only true and noise hits are considered for the calculation, empty pixels are ignored since they are of no interest. There are networks which operate differently and can handle sparse data a lot better, such as submanifold sparse convolutional networks [44] or graph neural networks [45]. Those will be further investigated in future. In the next sections, different model settings with respect to filter size and layer number are investigated.

[2]The number of non-zero inputs for each time window, is mostly < 50. Therefore, only < 3 % of the 2304 inputs are non-zero.

3x3 filter size

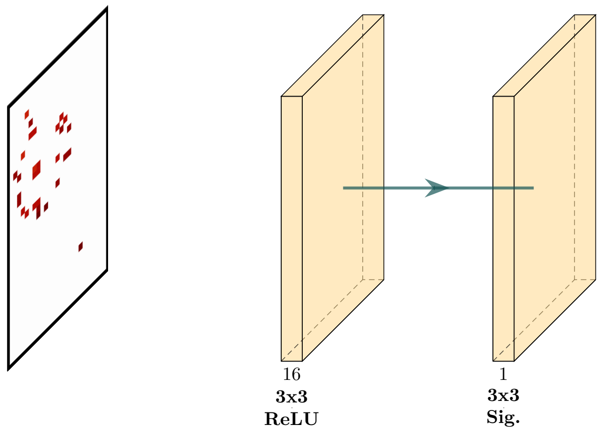


Figure 5.1: The 1x16 3x3 model architecture.

To get an idea which convolution layer settings are of importance, models including conv2d layers with 16 filters and a filter size of 3x3 are stacked. The number of hidden layers is increased from 1 to 6. The output layer for all stacked models is a conv2d layer with a 3x3 kernel size, 1 filter and a sigmoid (Sig.) activation function. The bias value is always used and trainable for all layers, also for the following architectures in this chapter. The architecture for the smallest network is shown in figure 5.1. This results in having the right output size and mapping $(0,\ 1) \rightarrow (0,\ 1)$. All information about the data amount, optimizer and the layers are summarized in table 5.1.

Setting	value
Training data size (time windows)	444643
Validation split size	20% of training data
Test data size (time windows)	446195
Loss function	binary cross entropy
Optimizer	ADAM
Learning rate	10^{-3}
Early stop patience	5
Batch size	128
Layer type	Conv2d
Hidden layer activation function	ReLU
Output layer activation function	Sigmoid
Padding	same[3]

Table 5.1: Data, optimizer and convolution layer information.

All metrics are calculated using a classification threshold of 0.5. With each added layer, the classification performance also increases, seen at the ROC curves (fig. 5.2). Table 5.2 summarizes different metrics for all tested models with 3x3 filters. The confusion matrix (fig. 5.3) for the 6 hidden layer model already shows a good performance. A good custom metric is the hit average, calculated by the average NN response value for all true hits. Especially the sensitivity is important to keep track off, in order to preserve the actual ringhits. The performance increase from using 1 to 6 hidden layers is at first glance obvious, because having more layers also means having more parameters. Taking the 1 hidden layer model as example, only information from pixels with a distance of 1 are used to update the initial pixel value. This means that stacking more layers results in a further message passing distance. Another way to pass information further, is to use a larger filter size.

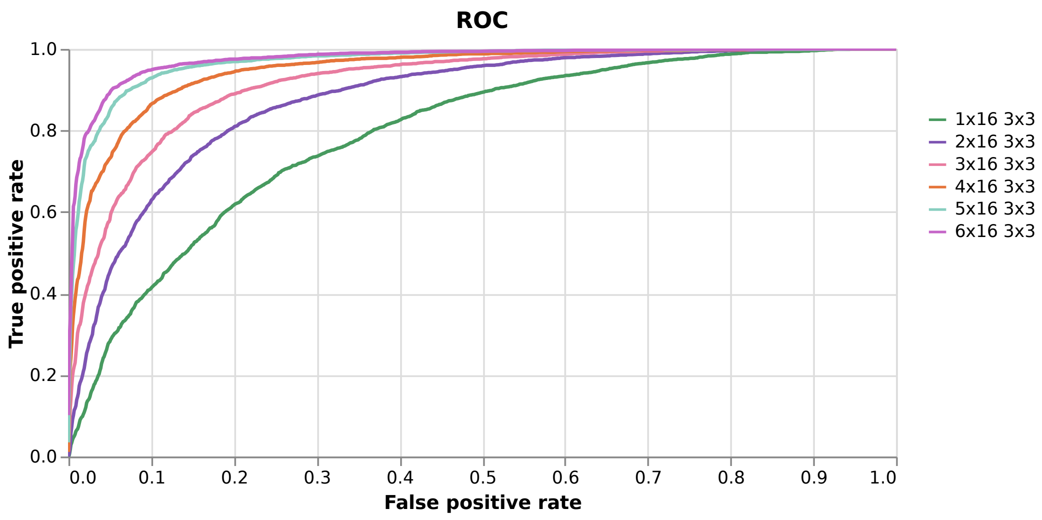
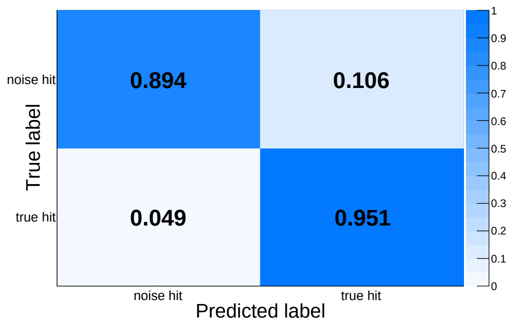


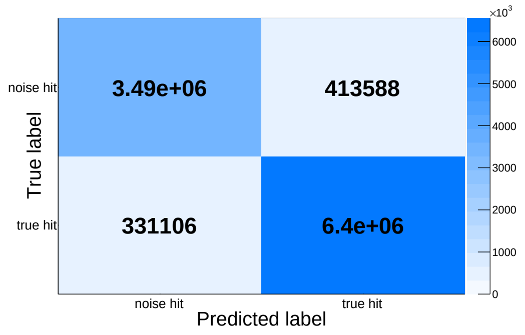
Figure 5.2: ROC curves for all 3x3 models.

Model	AUC	hit average	accuracy	sensitivity
1x16 3x3	0.7919	0.7263	0.7454	0.9021
2x16 3x3	0.8747	0.7880	0.8114	0.8733
3x16 3x3	0.9111	0.8253	0.8461	0.8950
4x16 3x3	0.9458	0.8691	0.8841	0.9284
5x16 3x3	0.9690	0.9027	0.9174	0.9337
6x16 3x3	0.9769	0.9171	0.9300	0.9508

Table 5.2: Metrics for stacking convolution layers with a 3x3 filter. For definitions see 1.3.1. AUC (area under the curve) with respect to the ROC curves in figure 5.2.



(a) Normalized across the true label.



(b) Total numbers.

Figure 5.3: Confusion matrices for the 6x16 3x3 model.

5x5 filter size

In this section the model is tested with a larger filter size, i.e. 6 layers with 9 filters and a filter size of 5x5 each are stacked. The number of filters is chosen to match the overall amount of parameter of the 6x16 3x3 model. This allows a fair comparison. For both classifiers, the ROC curve (fig. 5.4) and the metrics in table 5.3, there is an overall performance increase seen using 5x5 filters, although the model with 3x3 filters has a larger number of parameters. The underling reason for this is the larger message passing distance, when using 5x5 filters. Comparing the confusion matrices (fig. 5.5) to the previous one in figure 5.3, an improvement is seen across all categories.

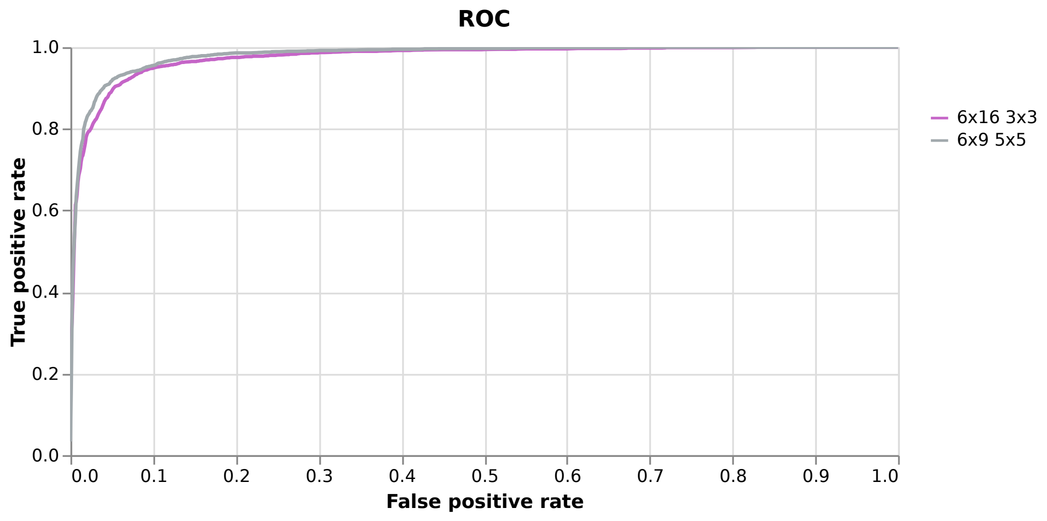
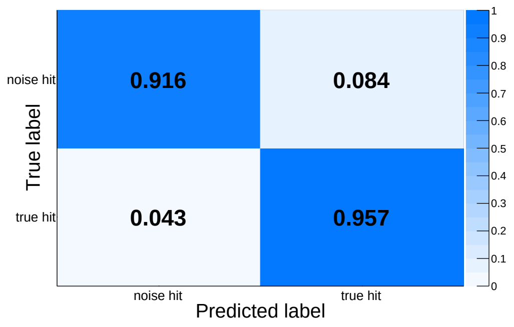


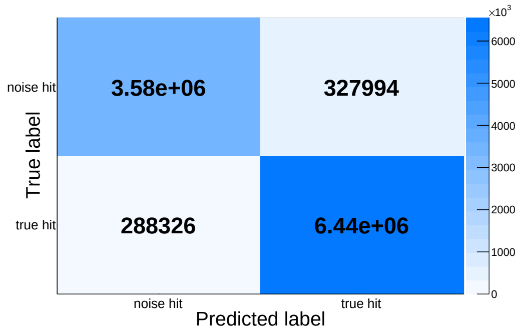
Figure 5.4: ROC curve for the 6x9 5x5 model, with the 6x16 3x3 curve as reference.

Model	AUC	hit average	accuracy	sensitivity	parameters
6x16 3x3	0.9769	0.9171	0.9300	0.9508	11905
6x9 5x5	0.9831	0.9313	0.9420	0.9572	10486

Table 5.3: Metrics for stacking convolution layers comparing 3x3 and 5x5 filters.



(a) Normalized across the true label.



(b) Total numbers.

Figure 5.5: Confusion matrices for the 6x9 5x5 model.

Mixed model using a different filter amount and size for each layer

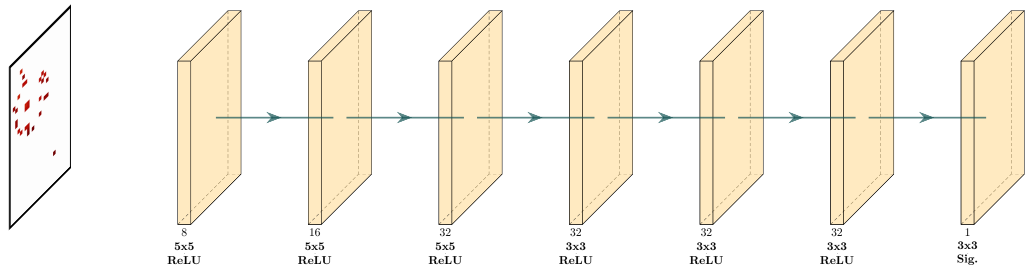


Figure 5.6: The mixed model using convolution layers with different filter sizes.

To further optimize, a model is constructed using 3 hidden layers with a 5x5 filter size and a small amount of filters, at the beginning of the stack. This architecture is chosen in order to pass information at larger distances early on in the neural network and keeping the number of computations low with a small amount of filters. Afterwards 4 layers with a filter size of 3x3 and a larger amount of filters are used, to reconstruct and classify the hits. The full model architecture is shown in figure 5.6. When looking at the ROC curves in figure 5.7, there is another performance increase visible. The same holds for the metrics in table 5.4 and the confusion matrices in figure 5.8. Even tough the number of parameters is larger, in this specific case the mixed model performs slightly faster with 9 ms vs. 12 ms for the 6x9 5x5 model per time window. These times are calculated from the average runtime evaluated from 1000 time windows, doing inference on the CPU[4] with a batch size of 1. This is the batch size later used for doing inference in the TMVA/CbmRoot framework.

[4]CPU used: Intel i5-4670K

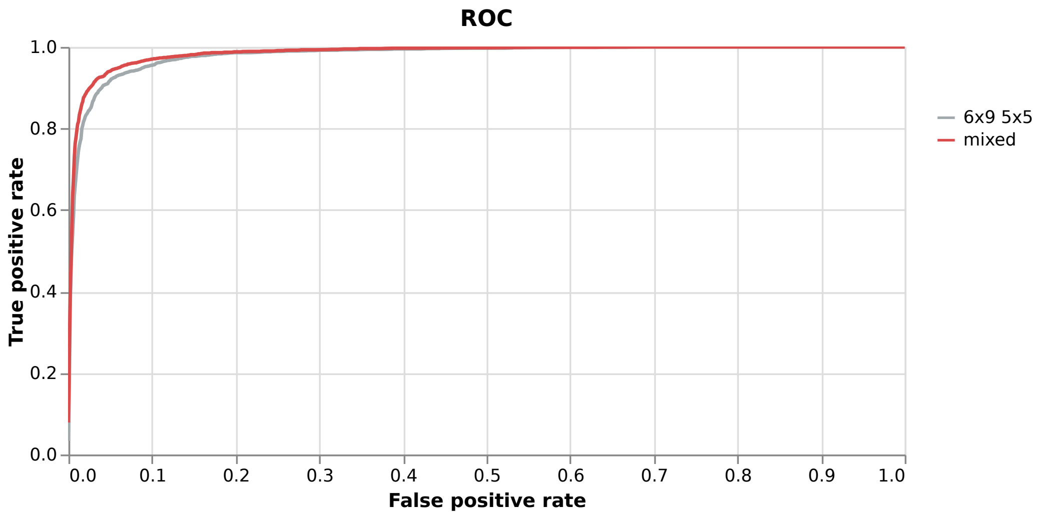
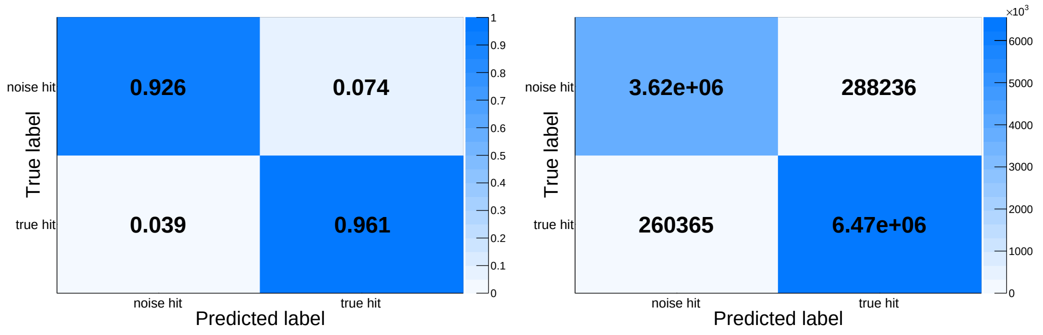


Figure 5.7: ROC curve for the mixed model, with the 6x9 5x5 curve as reference.

Model	AUC	hit average	accuracy	sensitivity	parameters
6x9 5x5	0.9831	0.9313	0.9420	0.9572	10486
Mixed	0.9863	0.9407	0.9484	0.9613	44289

Table 5.4: Metrics for stacking convolution layers comparing the 5x5 filter model with the mixed model having 3x3 and 5x5 filters.



(a) Normalized across the true label. (b) Total numbers.

Figure 5.8: Confusion matrices for the mixed model.

5.2 UNet model

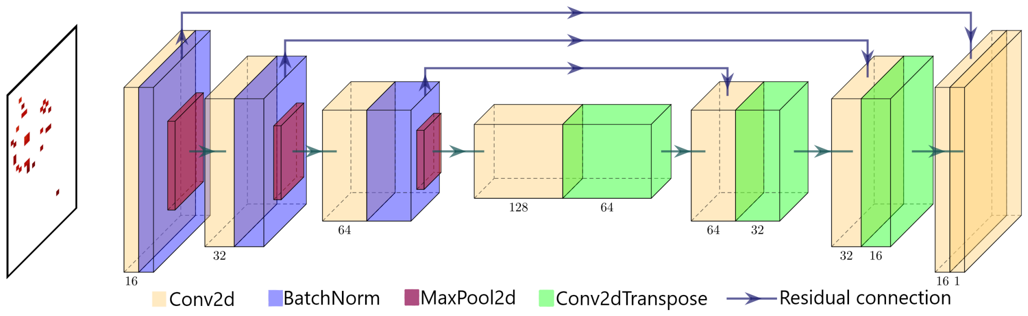


Figure 5.9: UNet model, all convolution layers are using a 3x3 filter size and the ReLU activation function is used for all hidden layers. The output layer uses the sigmoid activation. For up/downsampling in the MaxPool and Conv2dTranspose a factor of 2 is used.

In this section, a UNet [46] inspired network is tested in order to compare the performance of a state of the art neural network to the restricted model architectures available in TMVA. Such networks show very good performances on tasks like denoising, segmentation and instance segmentation. The full architecture is shown in figure 5.9. There are up/down sampling layers used to enhance the information passing distance. A similar network architecture would be CNN autoencoders which consist of an encoder and decoder part for the reconstruction. For sparse data those networks have the disadvantage, that the initial position information gets lost quickly when using down sampling. This poses a difficulty for the encoder part to properly reconstruct the right output at the right pixel position. An improvement upon this and the most important part of UNet's are thus the residual connections (or skip connections) [38]. They pass information past a layer or even multiple layers and afterwards concatenate it with data from later parts of the network in the channel dimension. This means that the initial information is still available in the decoder part, together with the information gained from previous layers. In addition, models including residual connections are mostly easier to train by having simpler loss surfaces [47] and overall perform better. The data, optimizer and batch size settings used are the same as given in table 5.1. The last convolution layer is also the same as used before. Both, ROC curve (fig. 5.10) and the metrics in table 5.5, show a classification performance increase compared to the mixed model. The UNet performs very good especially for the noise hit recognition seen in the confusion matrix (fig. 5.11). However, overall the performance increase compared to the mixed model is rather moderate on the cost of being more complicated than the simple convolution layer stacking in the mixed model. The UNet has approximately 6.6 times more parameters and the average inference time is 12 ms, compared to 9 ms for the mixed model. The reason for a fast computation time even though the larger number of parameters, is the up/down sampling which significantly reduces the number of operations. Comparing the number of floating point operations (FLOPS), the UNet has a lower value with 0.04857 GFLOPS compared to the mixed model with 0.1019 GFLOPS. The reason for the slower inference time of the UNet model most likely comes from the cumulative overhead from data transfer from RAM to the CPU cache because of the larger amount of parameters.

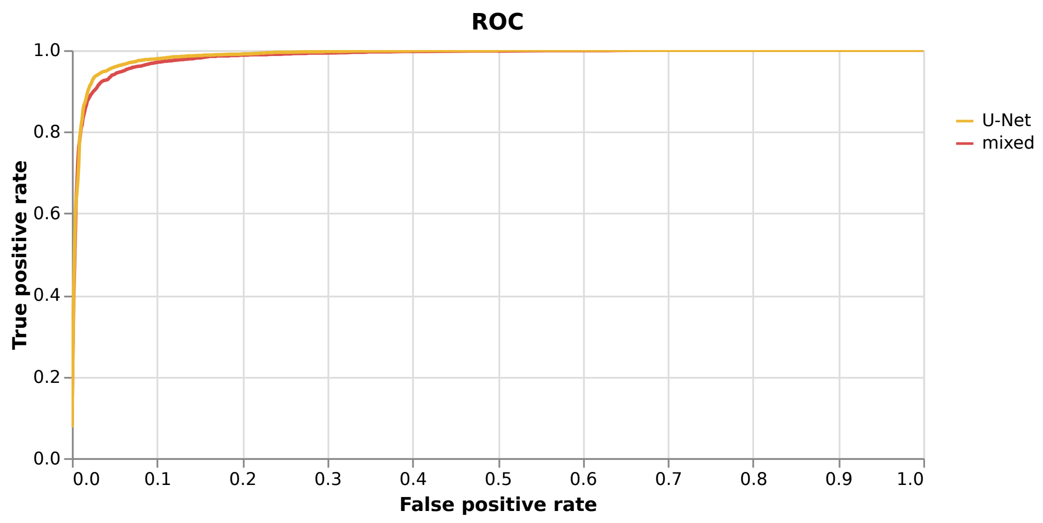
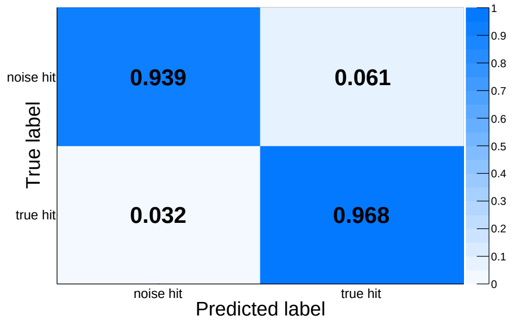


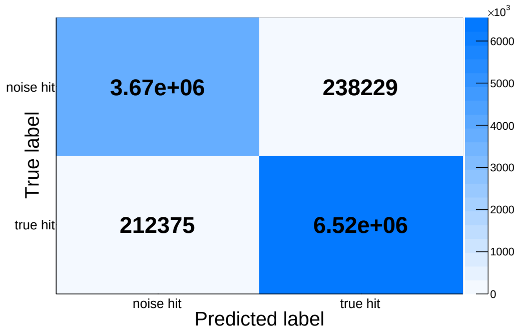
Figure 5.10: ROC curve for the UNet model, with the mixed curve as reference.

Model	AUC	hit average	accuracy	sensitivity	parameters
Mixed	0.9863	0.9407	0.9484	0.9613	44289
UNet	0.9899	0.9569	0.9576	0.9684	291505

Table 5.5: Metrics for stacking convolution layers comparing the 5x5 filters model with the mixed model having 3x3 and 5x5 filters.



(a) Normalized across the true label.



(b) Total numbers.

Figure 5.11: Confusion matrices for the UNet model.

Chapter 6

Analysis

For the following analysis of simulation and real data, the mixed model (see section 5.1) is trained and applied using the ROOT internal TMVA machine learning framework. A few adjustments have to be made, to get the model (see figure 5.6) properly running in TMVA. First, when working with CNNs the output in this framework is always expected to be a matrix. Since one of the dimensions is always needed for the batch size B (also when evaluating with a batch size of 1), only one dimension is left and because the output of the CNN model has to be $B{\times}72{\times}32$. Thus, a flatten layer has to be applied as the last layer. The resulting data format is then of size $B \times 2304$. In addition, the regression task from TMVA is used, the only available loss function for this task is the mean squared error. This leads to no convergence if an activation function mapping from (-∞, ∞) $\rightarrow$ (0, 1) is used, e.g. Sigmoid. Therefore, the Tanh activation function is used for the last convolution layer, instead of the Sigmoid function. To work around the fact that the mapping is from (-∞, ∞) $\rightarrow$ (-1, 1), all negative values are set to zero (effectively applying a ReLU function) when running prediction or evaluations on the test set. For the training process, noise hits are still labeled with value 0 and true hits with 1. Real data and all settings used, is the same as used before for the parameter adjustment chapter, summarized in table 4.1. In order to evaluate metrics like efficiencies for the ring finding, eventbased simulations have to be used, since the connection between Monte Carlo information and rich hits is not yet fully implemented for timebased simulations.

6.1 Analysis of simulated data

The model is tested on a separate simulation dataset with the settings from table 4.1 and the sim+ parameters (table 4.2), but using the UrQMD files 801-900 instead of the files from table 4.1. Training and evaluating the TMVA model on the test dataset shows a good separation between noise and true hits, which can be seen in figure 6.1. Comparing the results to the Keras framework shows no differences in classification performance. For both ends of the spectra, the difference of true and noise hits is around 1.5-2.0 orders of magnitudes. This already indicates a good classification performance. It should be noted, that the TMVA inference time for batch sizes of 1 take about 200 ms per time window, compared to the 9 ms using the Keras framework. When running on the FAIR/GSI HPC cluster nodes [48] the inference time reduces to approximately 70 ms per time window, because of an around 4 times larger CPU L3 cache size.

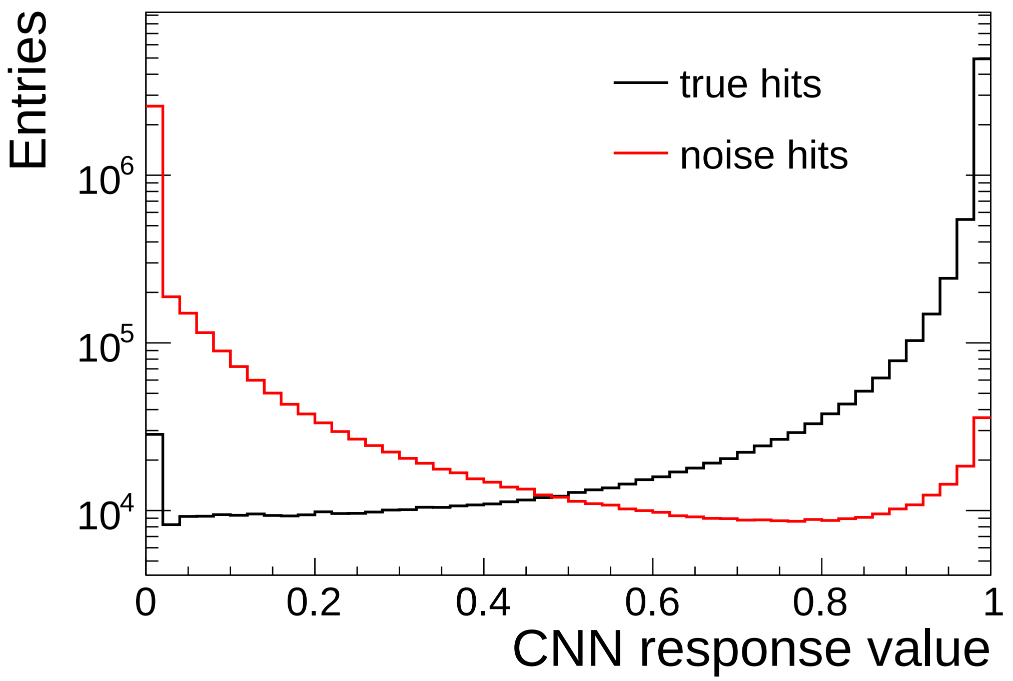
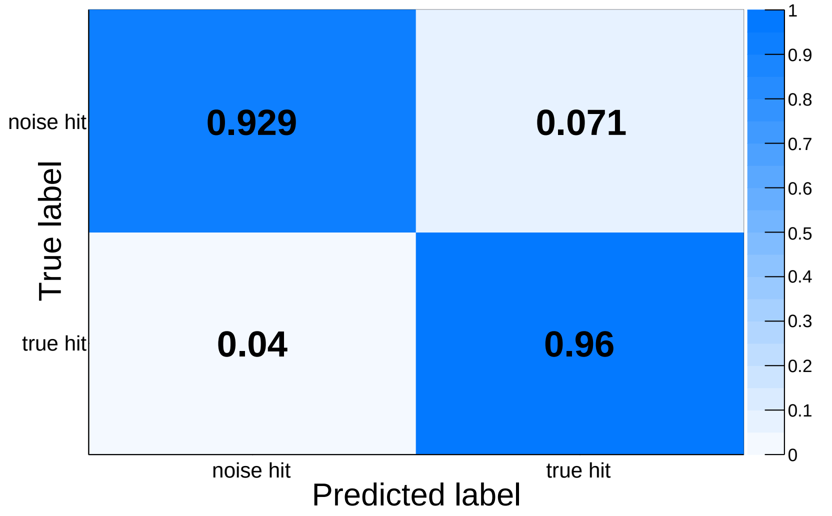
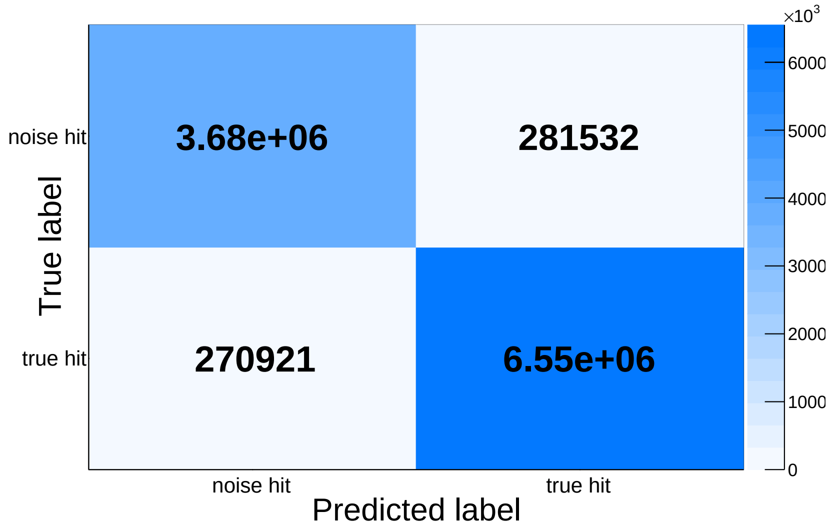


Figure 6.1: NN response values for true and noise hits in a logarithmic scale. The evaluation was done on the test dataset, which are the simulations from the UrQMD files 'urqmd.auau.1.24gev.mbias' 801 to 900.

For this analysis a fixed threshold value of 0.5 is applied to the NN response value, which means all hits with values ≥ 0.5 get classified as true and all others as noise hits. When applying the threshold, confusion matrices can be evaluated, seen in fig. 6.2. With a true hit recognition of 96%, there is on average a loss of approximately 1 from 20 true hits, while having a 93% noise recognition rate. It would also be possible to change this threshold value, for example setting it to a higher value to remove more noise, but this comes with a trade-off of also removing more actual ringhits in which we are interested in.



(a) Normalized across the true label.



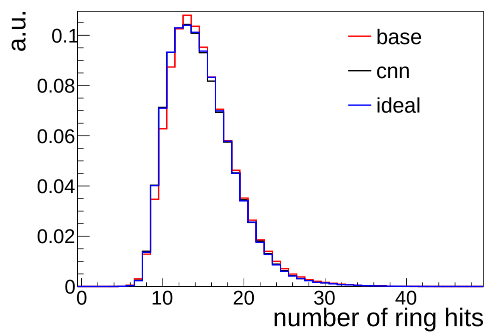
(b) Total numbers.

Figure 6.2: Confusion matrices for the TMVA stacked CNN model using a threshold value of 0.5. Evaluating hits in time windows from the test dataset.

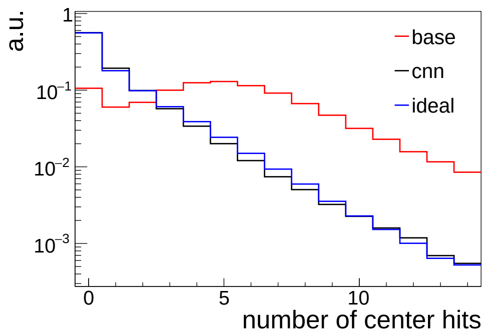
For the following analysis, three different setting for noise removal, before applying the ring finder, are compared for various distributions:

- base : no noise removal, only running the ring finder based on the Hough Transformation
- cnn : using the trained stacking CNN TMVA model before ring finding
- ideal : MC truth, removing noise according to the labeling process described in section 4.2 before ring finding

In order to have the same events for all three settings, the events are selected by the amount of non digitizer noise hits (≥ 5), since this amount is equal for all settings. Digitizer noise hits are those randomly generated all over the mRICH and the new artificial noise from charged particles. With the introduction of additional center hits duo to charged particles in simulations (section 4.1), a clear difference can be observed between the ring finding without any noise removal and the application of the CNN beforehand. Looking at the amount of center hits (fig. 6.3b) it is immediately visible that the number of center hits are strongly reduced applying the CNN. This is a first indication, that the machine learning algorithm does what it is supposed to do. For the number of ring hits, there is only a slight difference visible in the different distributions (fig. 6.3a), which also indicates that the overall ring structures are mostly conserved, considering the amount of ring hits.

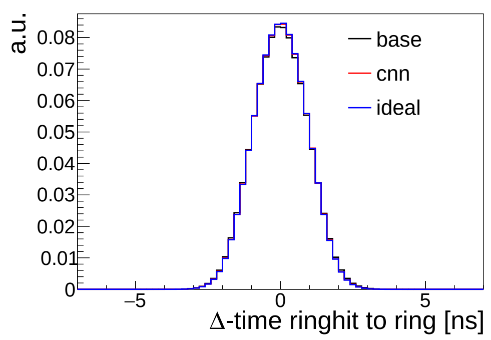


(a) Number of hits matched to the found ring.

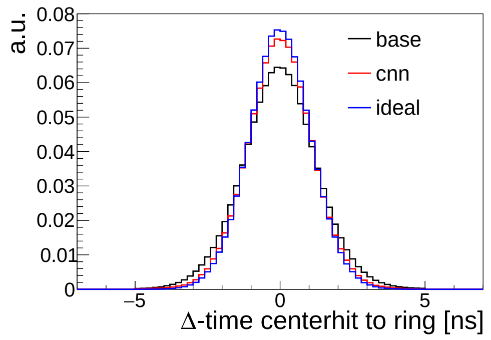


(b) Number of hits inside 0.8R and with a time difference of $|\Delta t| \leq$5ns.

Figure 6.3: Comparing the number of ring/center hits for different settings.



(a) Time difference of ring hits to the found ring.



(b) Time difference of hits inside 0.8R and a time difference of $|\Delta t| \leq 5$ns.

Figure 6.4: Comparison of ring/center hits vs. ring time difference for different settings.

Figure 6.4 shows the timing of hits matched to a ring (a) and hits within the ring (b). Only for the ring center hits some difference is seen indicating a wider spread of the center hits time. The most important outcome is that the noise removal also reduces small rings with radii $\leq$ 3.7 cm (see fig. 6.5). Those low radius rings indeed disappear for both noise removal settings. The CNN result also matches quite perfectly with the ideal ring radius distribution.

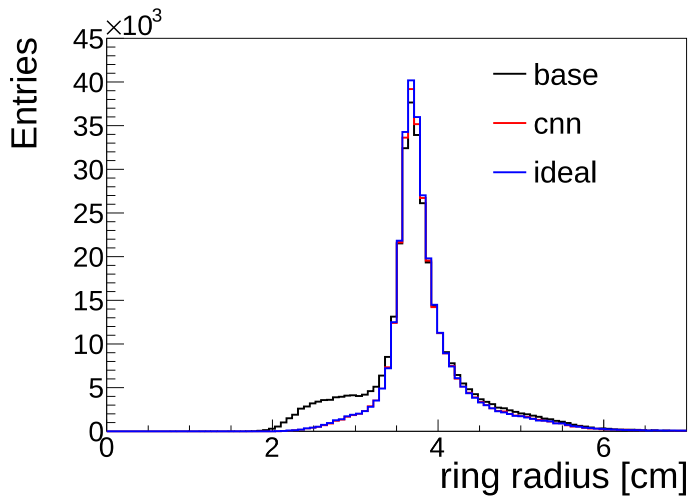
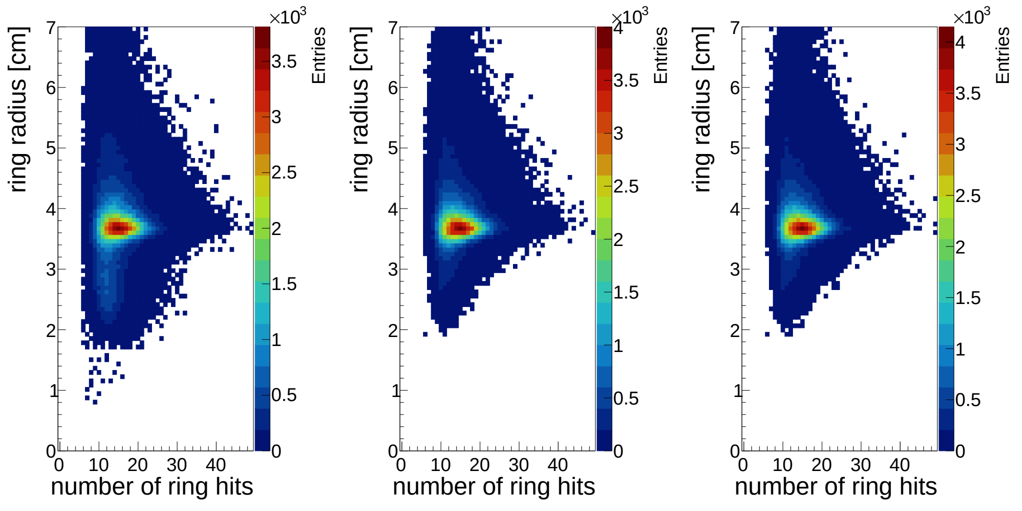


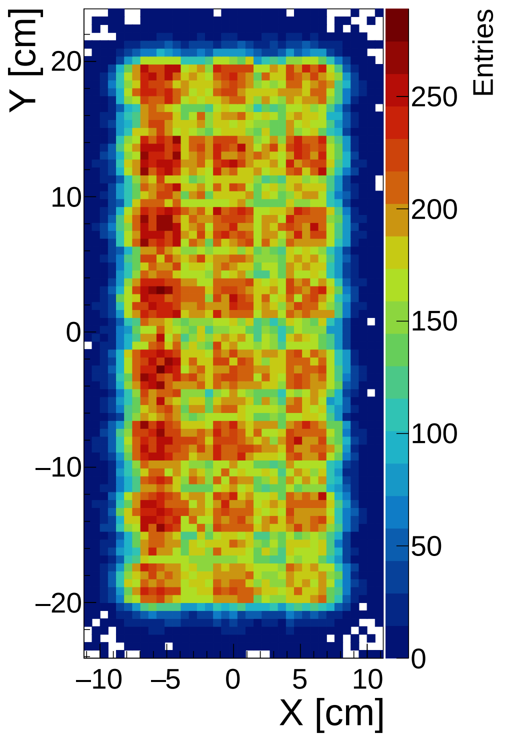
Figure 6.5: Ring radius distribution for different settings, the low radius entries mostly vanish for both noise removal settings.



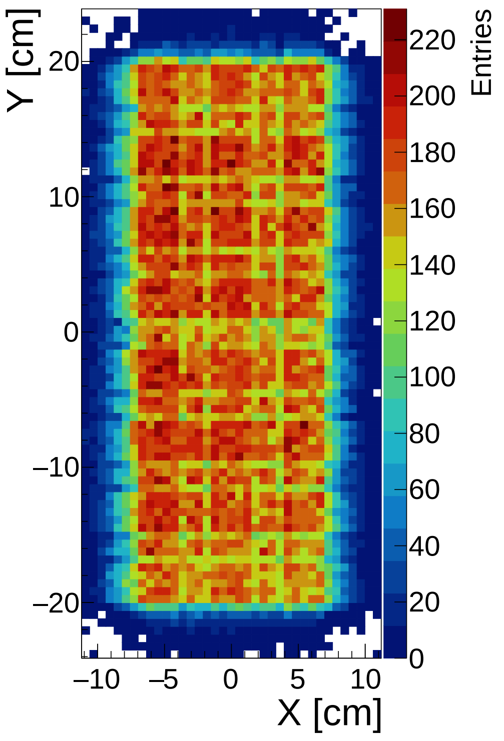
(a) base. (b) cnn. (c) ideal.

Figure 6.6: Number of ring hits vs. ring radius for different analysis settings.

The same behavior in shown in figure 6.6: Without noise removal, many small rings with radius ≤ 3.0 cm with quite some ring hits can be seen. Those rings are a result of the additional noise produced by charged particles. Both noise removal algorithms reduced those by a great amount, resulting in a sharper distribution. Here as well, the cnn application matches the ideal case very well.



(a) base.



(b) cnn.

Figure 6.7: Ring center positions in the mRICH plane for different settings.

Figure 6.7 shows centers of reconstructed rings, here a clear pattern is visible: without noise removal rings are found more often in between MAPMT's. With the application of the CNN this structures disappear completely, and the distribution of ring centers becomes more homogeneous.

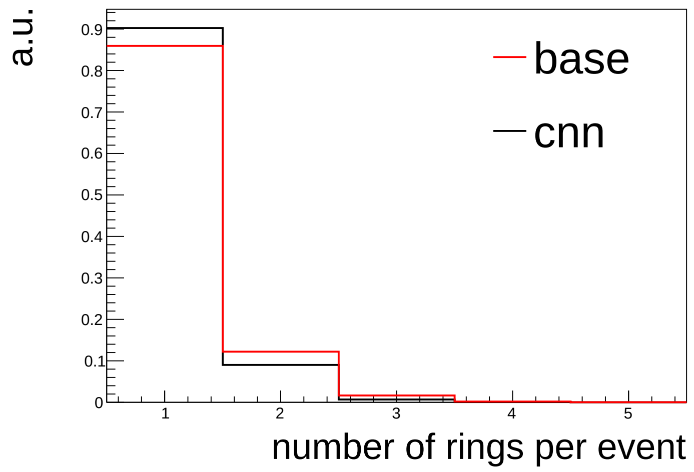


Figure 6.8: Rings per event for the base and cnn setting.

Comparing the number of rings per event in figure 6.8, with the application of the CNN, compared to the base setting more event with only 1 ring are found and fewer events with multiple ring.

This analysis of UrQMD simulations was explicitly made within the event-based mode, in order to have access to all Monte Carlo information. The amount of accepted and true reconstructed rings (or good rings) can be calculated with this information, which leads to single values indicating how good the overall ring finding performance really is.
The following definitions will be used:

Accepted rings: Count the number of hits generated by Cherenkov photons produced inside the aerogel for all charged particles. If this amount is larger than a certain threshold ($\geq$ 7 hits) for each mother particle, this is defined as an accepted ring. For example, 9 hits contain Cherenkov photons produced inside the aerogel by the charged particle X. Since this is larger than the threshold, those 9 hits are considered as an accepted ring together with the particle X information. This obviously is a very simple estimation of the expected number of rings, because for example the hit positions and ring structures are completely ignored. Nevertheless, some reasonable estimate on the number of expected rings that can potentially be found is necessary.

Good rings (based on ring hits): After ring finding, find the main contributor to the found ring, i.e. the charged particle which has the largest proportion of ring hits containing photons produced by this charged particle inside the aerogel. In case the ring has no accepted ring counterpart, it is directly declared as a false ring. If it is connected to an accepted ring, and the number of main contributor hits of the ring divided by the total number of ring hits is larger than the quota threshold of 0.7, this ring is considered as a true reconstructed ring or good ring.

Good rings (based on all hits): Another metric is to look at all hits from Cherenkov photons that the charged particle generated, instead of only the ring hits. This has the advantage, that a ring constructed from ring hits and center hits will be more likely considered as a false ring. The same threshold of 0.7 is set here for the number of main contributor hits of the ring divided by the total number of the main contributor hits. For the test dataset 10^6 events have been simulated, the number of accepted rings in this dataset is $N_{acc.}= 4.638 \times 10^5$. In table 6.1 and 6.2 the found rings and good rings for both metrics are listed. Comparing the three different settings, the base setting always finds overall more ring than the CNN or ideal algorithm. However, the number of good rings is smaller. The ratios of good rings to found or accepted rings clearly reveal, that for the base setting indeed many rings are found, however with an overall lower purity. The noise removal on the other hand yields a higher ratio of good rings to found or accepted rings. Obviously, the noise removal reduces the reconstruction of false rings. Hereby the number of good rings/found rings describes the overall ring quality, while the number of good rings/accepted rings gives an indication about the ring finder accuracy of finding expected rings. This can also be very nicely seen in the good rings/found rings calculation, which describes the overall ring quality, while the good rings/acc. rings describes the ring finding accuracy. To summarize, across all values there is a significant improvement using algorithms with noise removal, especially for the ring quality.

	found rings	good rings (ring hits)	good rings /found rings	good rings /acc. rings
base	3.708×10^5	2.767×10^5	0.746	0.597
cnn	3.191×10^5	2.978×10^5	0.933	0.642
ideal	3.227×10^5	3.077×10^5	0.954	0.663

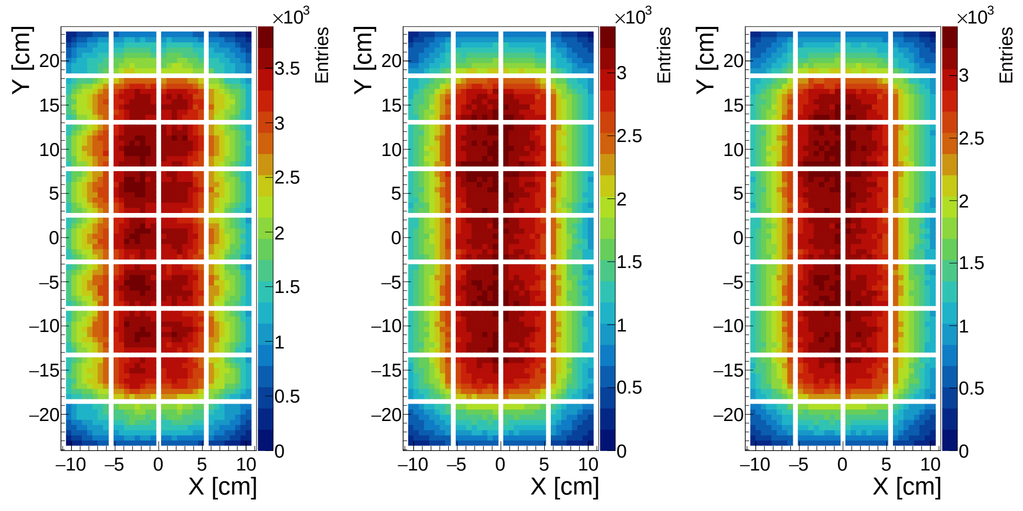
Table 6.1: Efficiency calculations for rings using the ring hit metric.

	found rings	good rings (all hits)	good rings /found rings	good rings /acc. rings
base	3.708×10^5	2.742×10^5	0.740	0.591
cnn	3.191×10^5	2.775×10^5	0.870	0.598
ideal	3.227×10^5	2.900×10^5	0.899	0.625

Table 6.2: Efficiency calculations for rings using the all hit metric.

For the found ring hits, there is a tendency towards finding more ring hits in the center of MAPMT's (fig. 6.9a). In case of the CNN and ideal noise removal setting in figure 6.9b and 6.9c, it seems that ring hits are slightly more often found at MAPMT edges. This is expected, since the edge pixels are slightly larger.

Single event displays applying the cnn setting and running the ring finder afterwards are shown in Appendix B.3.

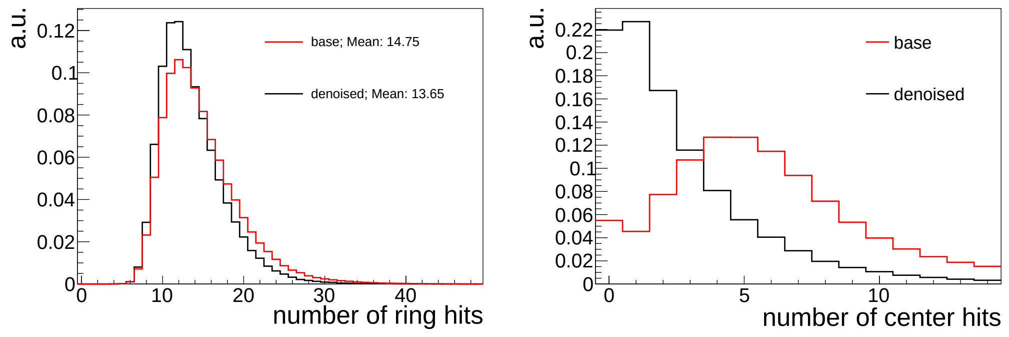


(a) base. (b) cnn. (c) ideal.

Figure 6.9: Distribution of ring hit positions for different settings.

6.2 Analysis of real data

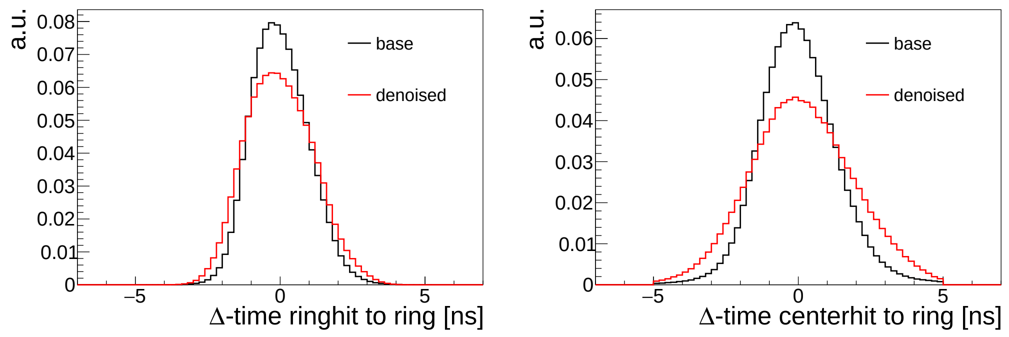
The same TMVA model is now applied to real data. The run information, all settings and the event selection using the trigger windows and values are listed in table 4.1. Similarly, as for the UrQMD simulations, a setting without noise removal (”base”) and with noise reduction using the CNN (”denoised”) are compared. Figure 6.10b depicts the number of center hits, a strong reduction of center noise is observed comparing the two settings. This was expected and is a first indicating that the noise reduction actually works. For the number of ring hits, the distribution from denoised events slightly moves towards lower values and has a smaller width, compared to the base setting.



(a) Number of hits matched to the found ring. (b) Number of hits inside 0.8R and with a time difference of $|\Delta t| \leq 5$ns.

Figure 6.10: Comparing the number of ring/center hits for different settings.

For the time difference distributions (fig. 6.11) within a ring and for the center hits, there is quite a difference between the two settings visible. Both distributions show a wider distribution for the denoised setting. This is most likely a combination of the better ring recognition using machine learning and the ICD correction. Since the ICD correction is strongly dependent on the ring finder performance and was calculated using no denoising, it might be that the worse ring purity has led to inaccuracies in the ICD correction values. In order to improve, the simulation parameter adjustment and training process of the machine learning model would have to be included in the iterative process of the ICD correction calculation. A possible iterative process chain: apply denoising NN, do ring finding, calculate ICD correction, adjust simulation parameters, train model on new sim data & repeat.



(a) Time difference of ring hits to the found ring.

(b) Time difference of hits inside 0.8R and a time difference of $|\Delta t| \leq 5$ns.

Figure 6.11: Comparison of ring/center hits vs. ring time difference for different settings.

In figure 6.12 the two ring radius distributions are compared, the denoised results are very similar to the simulation (fig. 6.5). The overall number of rings decreases, especially small rings and large rings in the tail on the right side of the peak are reduced. This change in the distribution is comparable to the simulation analysis, which is another indication, that the noise reduction also works properly for real data. Comparing the ring radius distribution directly to the simulation data using the cnn setting shows are very good agreement, especially for the left side of the peak.

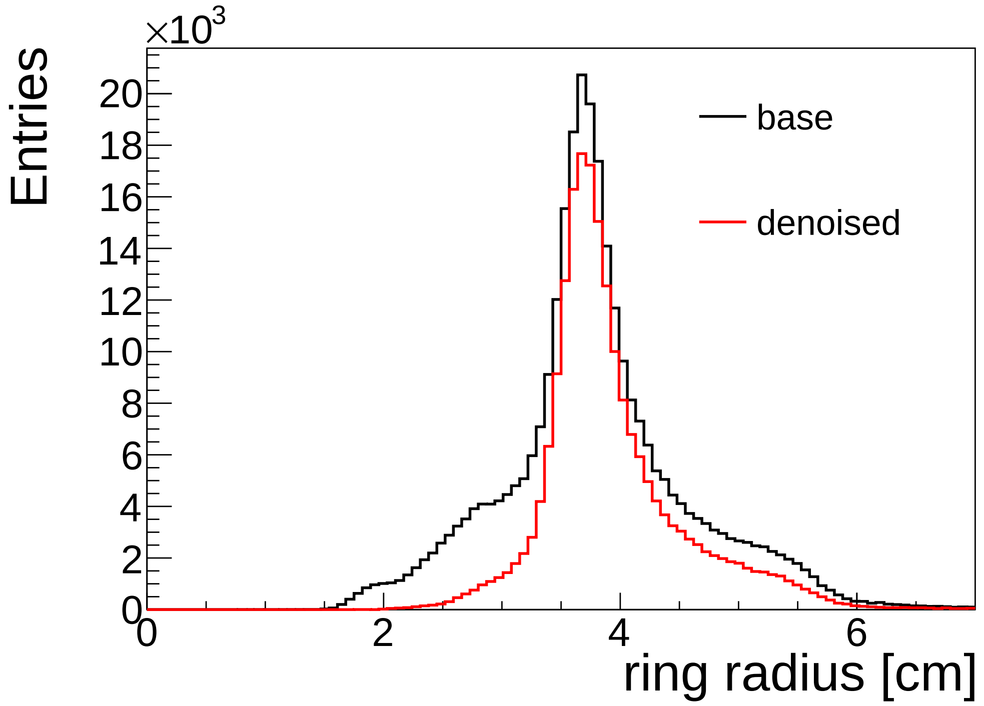


Figure 6.12: Ring radius distribution comparing the two setting, the low radius entries mostly vanish similar to the simulation.

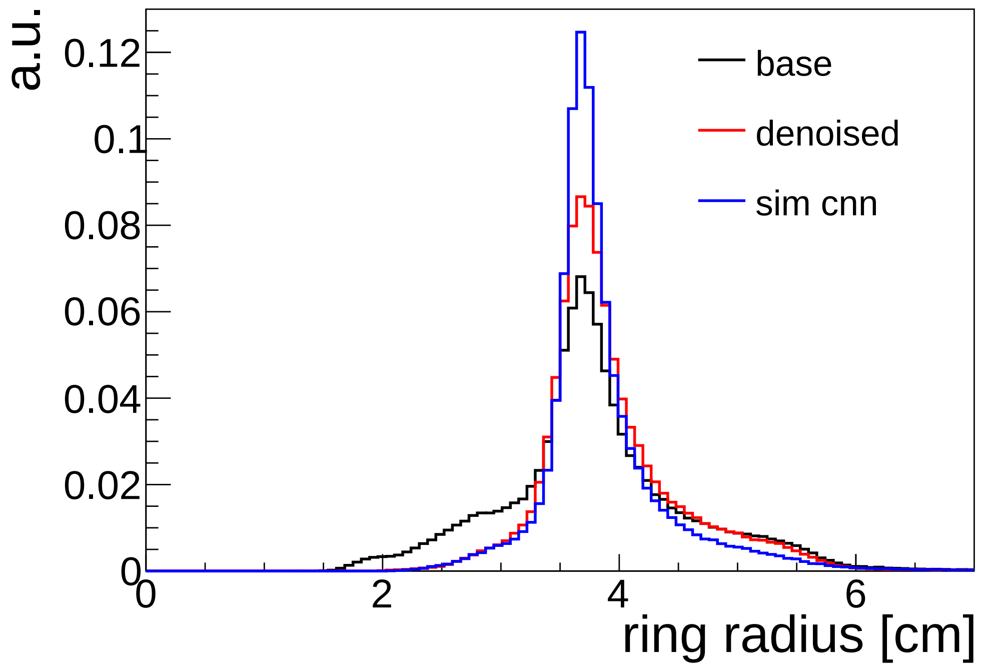


Figure 6.13: Ring radius distribution comparing the two settings for real data and the simulation using the cnn setting.

The mRICH uses two separate aerogel blocks, the lower one is older and shows some aging effects, most probably due to too high humidity in the first beamtime. We observe a smaller number of found rings in the lower half of the mRICH plane (see fig. 6.16a), which is most likely correlated to the different transmittance due to the altering process of the material. Comparing the ring radius distribution for the upper and lower half of the mRICH plane (fig. 6.14), it is immediately noticeable that even more smaller rings relative to the total ring amount are measured in the lower half of the mRICH plane. However, when applying the noise removal, the ring radius distributions for the upper and lower half match very well, in both cases removing the small radius peak. In comparison, ring radii are a bit larger in the lower half, which could be correlated to a smaller refraction index n due to aging [49]. A similar behavior can be observed for the number of ring hits in figure 6.15, rings with more hits are observed in the upper half of the mRICH plane, compared to the lower half. This should also be correlated to the aging effector of the aerogel, reducing the transmittance and therefore the number of hits.

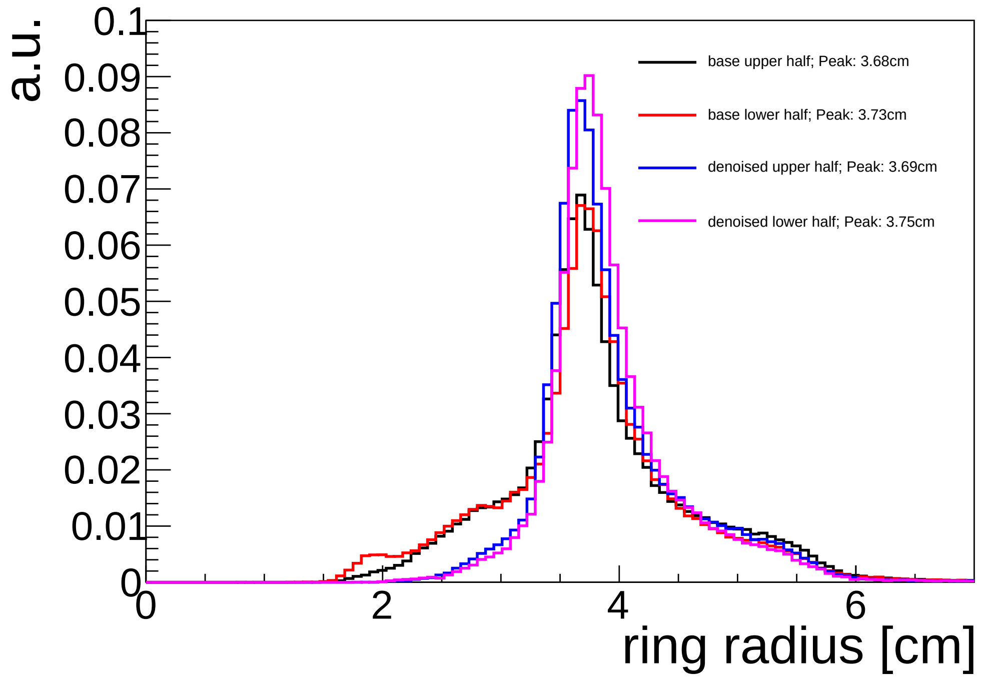


Figure 6.14: Ring radius distribution comparing rings in the upper and lower half of the mRICH for two settings.

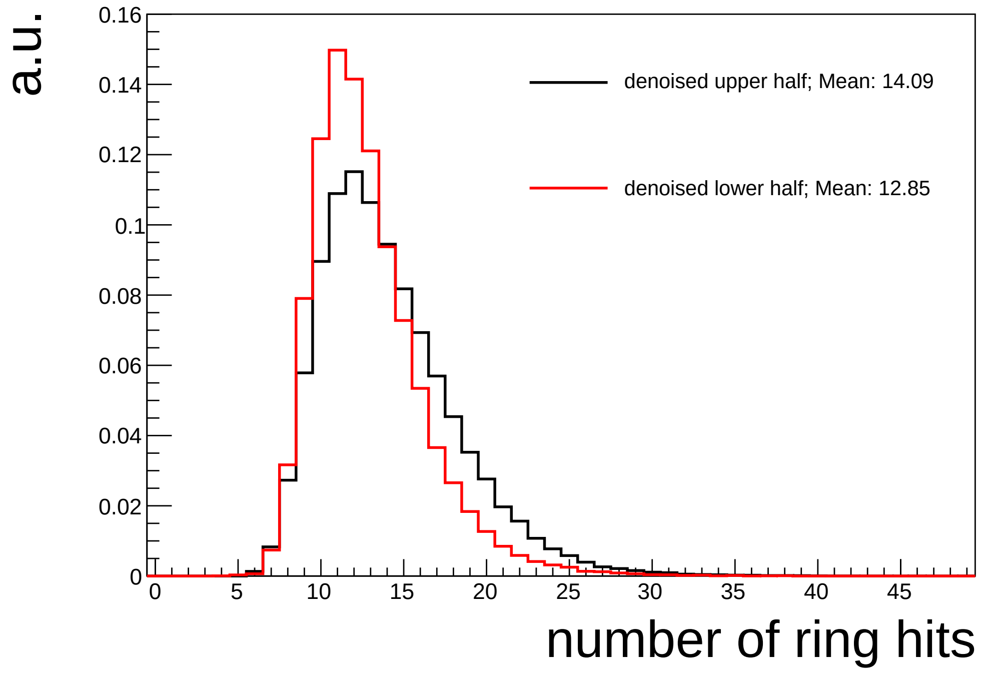
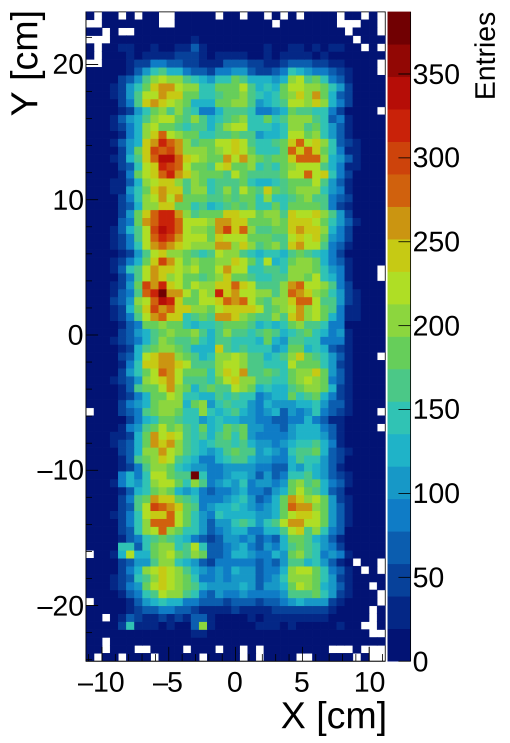
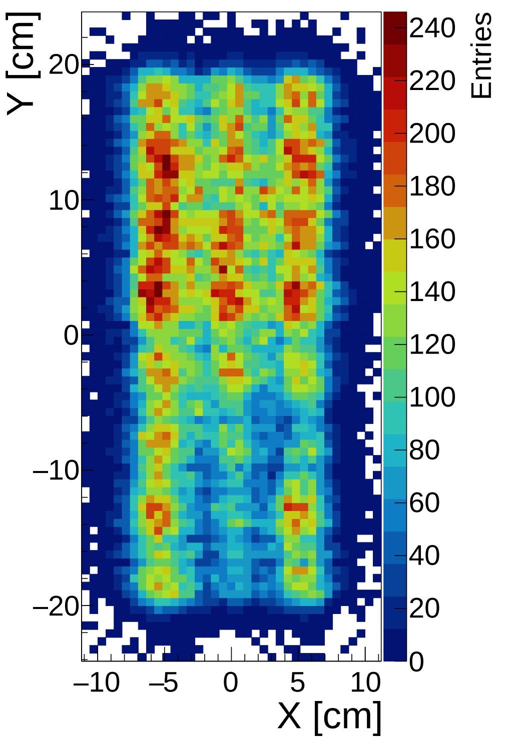


Figure 6.15: Number of hits matched to the found ring in the upper and lower half of the mRICH plane.

An explanation for the enhanced ring contribution at a radius of 2 cm, i.e. approximately half the MAPMT side length, might be that the lower half of the mRICH plane is more susceptible to having the full MAPMT lighting up (e.g. see Appendix figure B.20 upper figures). Often, a full ring is then reconstructed within the MAPMT area. This can also be seen in figure 6.16a, where there are bins in the center of MAPMT's in the lower half of the mRICH containing more rings than the surrounding bins. Concerning the overall positions of ring centers in figure 6.16b compared to 6.16a, this distribution just smoothes out a bit, but is still far from the homogeneous distribution in simulations (fig. 6.7).



(a) base.



(b) denoised.

Figure 6.16: Position of ring centers in the mRICH plane.

Subtracting both those histograms (base minus denoised) and dividing each bin content by the number of ring center in the base calculation (fig. 6.17), shows the percent change for each bin. Overall, (20-30)% of rings are removed using the denoising. This reduction for the amount of found rings is no problem, if the behavior is the same as in simulations, where the found rings are reduced, but the actual amount of good rings are larger than without denoising. Especially in the lower half we see that rings with a center in the MAPMT get removed more often. For others areas the removal seems to be homogeneous and therefore still not solving the fact, that rings are found more often in between MAPMT's.

This leads either to the conclusion, that we have another contributor to the phenomena of finding more rings in between MAPMT's, other than the large amount of center noise. Or simply that the combination of noise

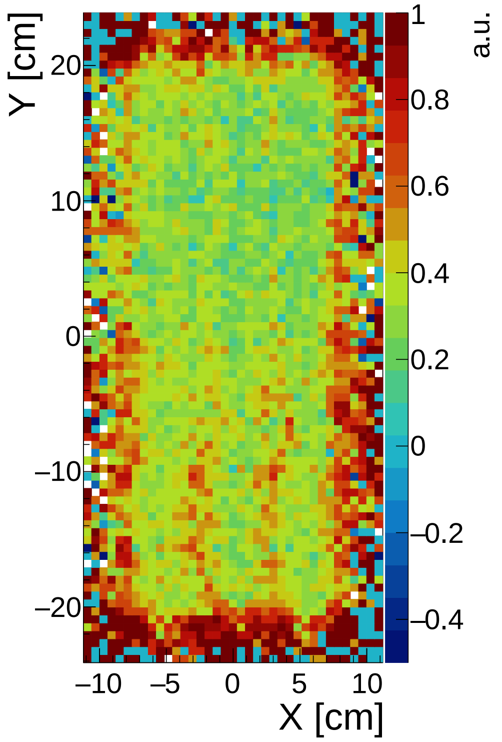


Figure 6.17: Number of ring centers subtracting the cnn from the base amount and normalizing each bin by the base amount.

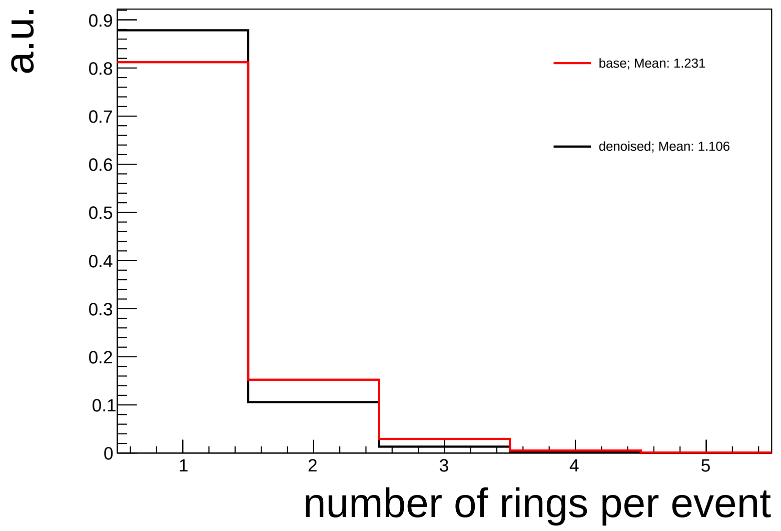


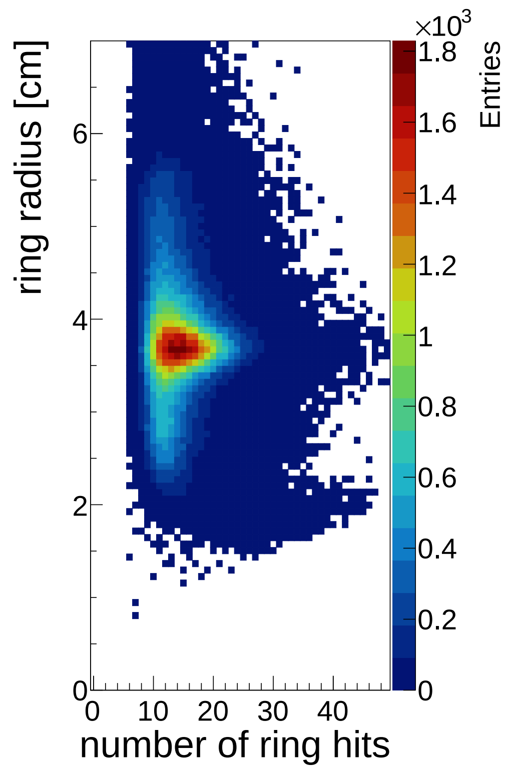
Figure 6.18: Rings per event for the base and denoised setting.

removal and ring finding is not good enough, probably correlated to the still not realistically enough simulation. Exactly this question is one of the main difficulties in all the noise removal tasks, using a fully supervised learning method. But since the addition of center noise, due to charged particles, in the simulation shows very similar pattern compared to real data, other training methods are now also thinkable of. Those could explicitly use real data for the training process (e.g. in a semi-supervised learning method) and might yield better results.

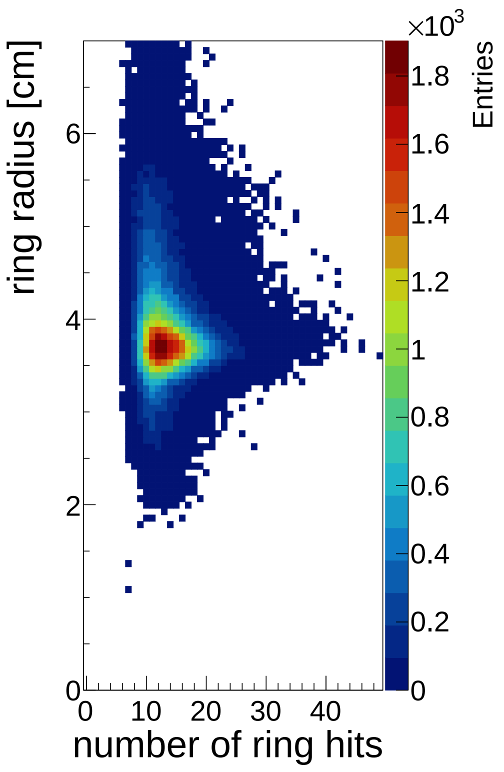
For the rings per event in figure 6.18, the number of event with a single ring increases, while events with multiple rings decrease. This is the same behavior as observed for the simulation in figure 6.8.

Comparing the ring hits vs. radius distributions, the low radius and high number of ring hits region completely disappears (fig. 6.19), very similar to the simulation results in figure 6.6.

Single event displays applying the denoising and running the ring finder afterwards are shown in Appendix B.4.



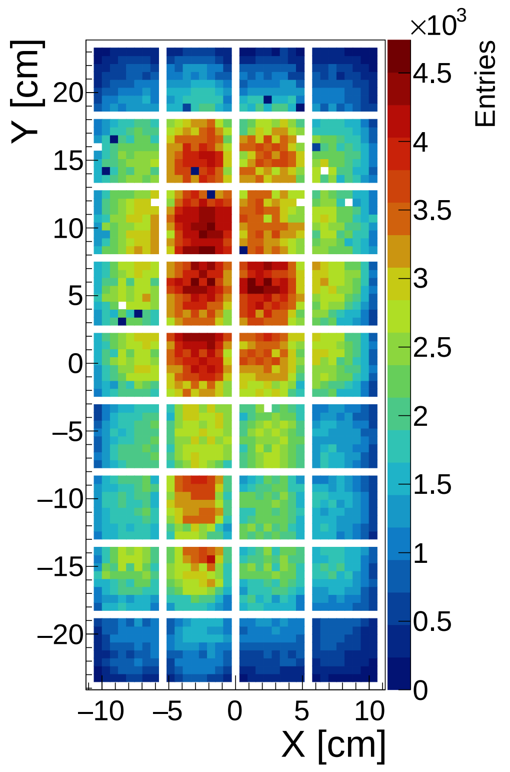
(a) base.



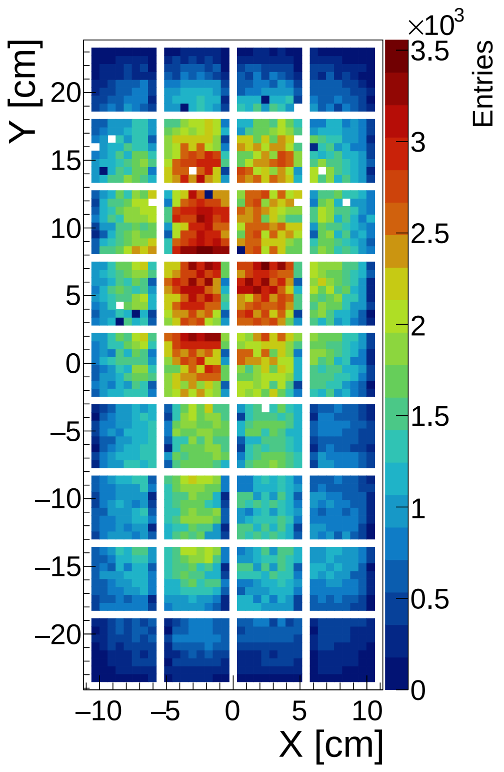
(b) denoised.

Figure 6.19: Comparing the number of ring hits vs. ring radius distributions.

Figure 6.20 shows the distribution of ring hits. Overall the number is lower in the lower half due to the aged aerogel radiator. For the denoised distribution (fig. 6.20b) a new pattern shows up, having a significantly decreased amount of ring hits on the vertical side pixels of the MAPMT. In dependence on the specific orientation of this MAPMT, this discrepancy is on the left or right side. Although, when looking closely at the base distribution 6.20a, the pattern is already there very slightly visible, but not to the extent seen in the denoised setting. The normalized difference for each bin (base minus denoised), calculated in figure 6.21, shows that this behavior is definitely systematic and should be further discussed. Due to the seen systematic, it is more probably to be correlated to the detector and not the ring reconstruction[1]. Also, given that this pattern is not occurring in simulations at all (see fig. 6.9). Indeed, the increased left/right removal of edge pixel hits is correlated to the hardware, i.e. a 180° rotation for every third MAPMT vertically. It remains to be checks whether those edge rows indeed show higher noise. Recent studies [50] have measured different dark rates and gains at the edge pixels, but having this effect for the denoising to this extent and only vertically on one side for each MAPMT, depending on the rotation, has to be further investigated.



(a) base.



(b) denoised.

Figure 6.20: Ring hits distribution for different settings, more noise is removed on one side for each MAPMT.

[1]Especially since the CNN has no positional awareness of MAPMT edge pixels

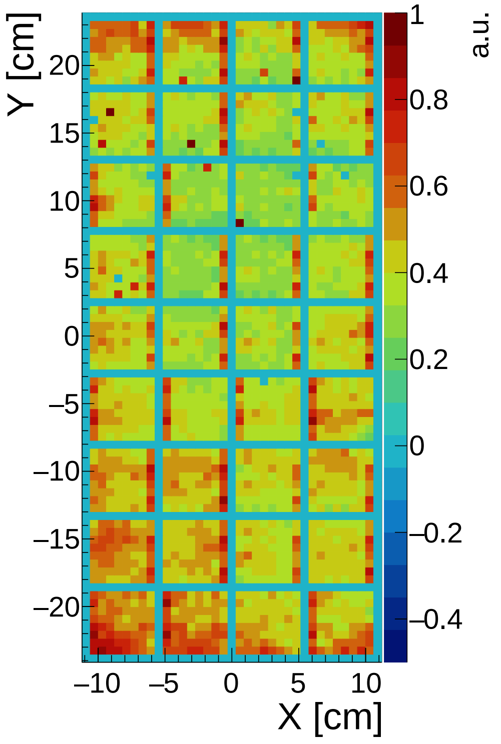


Figure 6.21: Number of ring hits subtracting the cnn from the base amount and normalizing each bin by the base amount. A systematic behavior occurs in having more noise on the MAPMT vertical edge pixels removed. Every third MAPMT (vertical) having more removed noise on the left edge pixels, because they are rotated by 180°.

Chapter 7

Summary and outlook

This thesis presents an approach to improve the ring finding performance for the mRICH using a CNN to reduce specific hit patterns confusing the ring finder and noise. Because of the mRICH detector setup, specific hit patterns are observed decreasing the ring finding accuracy. Most of those patterns are caused by charged particles passing directly through the photo detector plane. Since this is not seen in simulations, artificial hit patterns have been added to the simulation, in order to get a more comparable detector response to the observed real data. This artificial hit pattern is currently independent of all particle information like mass or momentum. Nevertheless, anomalies similar to what is seen in real data is now implemented also in simulation data. With this improved simulation, the actual Hough ring finder performance could be tested, when having a lot of noise inside ring structures. In addition, it was a crucial task to get more realistic data for supervised machine learning tasks, in order to perform more reliable predictions on real data. The actual CNN model has shown a very good performance, which was directly verified in simulations using Monte Carlo information. For the real data analysis, it definitely improved upon removing certain anomalies, such as a large contribution of rings with low ring radii in the ring radius distribution. To what extent it improves the overall particle identification, including other detector information i.e. mTOF track projections, needs to be investigated.

The main disadvantages of the current implementation using the TMVA framework, are the NN model restrictions and the inference time, which is very important for fast data analysis for the online monitoring. Importing more state to the art model using ONNX Runtime will result in models itself having better classification accuracy and runtime performance. Additionally, ONNX Runtime tries to optimize for fast inference time, which could be even further enhanced using lower floating point precision or even converting all calculations to integers (Quantization). Given the sparse input data of the mRICH, different neural network types such as graph neural networks and sparse convolution networks should be tried, since they operate on such data more efficiently. Now having a more realistic detector response for simulations, semi supervised learning methods might work as well, training on simulation and real data combined. Since this additional hit pattern removal machine learning application has shown a very good pattern recognition, other tasks should also be feasible. For example, direct PID based on ring structures or hit instance segmentation, which is basically the same task as the current Hough ring finder does.

Appendix A

Mean squared error and Sigmoid

For the sigmoid activation combined with the MSE loss function, there is no convergence at all. This is shown for a very simple calculation using one input x, target $\hat{y}$ and an arbitrary weight w, neglecting the bias value. The predicted value y for a single node is then calculated using equation A.1.

$$y = Sigmoid(wx) = \frac{1}{1 + e^{-wx}} \tag{A.1}$$

With the squared loss in equation A.2, both derivatives in equation A.3 and A.4, the update for the weight w can be calculated using equation A.5 with gradient descent as the optimizer.

$$SE = (\hat{y} - y)^2 \tag{A.2}$$

$$\frac{\partial}{\partial y} SE = -2(\hat{y} - y) \tag{A.3}$$

$$\frac{\partial}{\partial w} Sigmoid(wx) = y(1 - y)x \tag{A.4}$$

$$w' = w - \alpha \frac{d}{dw} SE = w - \alpha \frac{\partial SE}{\partial y} \frac{\partial y}{\partial w} = w + 2L(\hat{y} - y)y(1 - y)x \tag{A.5}$$

The last equation in A.5 includes the factor $y(1-y)$, which becomes 0 if $y \to 0$ or $y \to 1$, resulting in no updates of the weight for the desired prediction values.

Appendix B

Single event displays

B.1 Old simulation events without noise removal

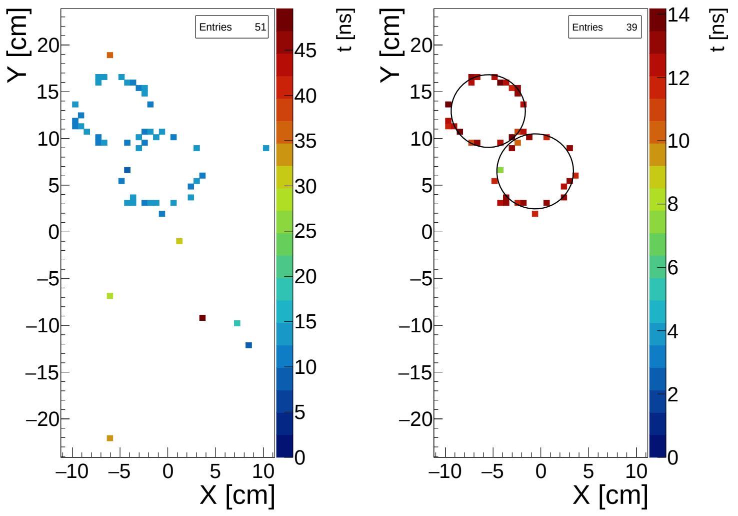
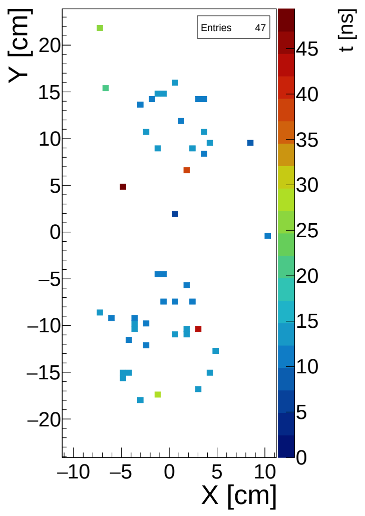
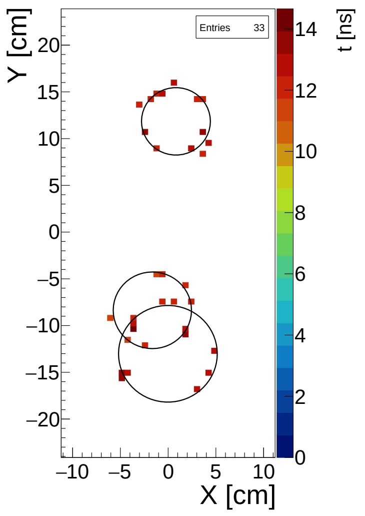
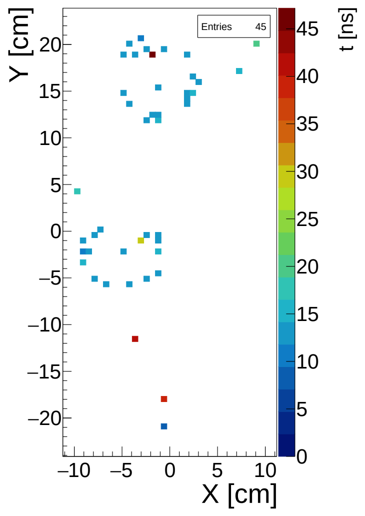


Figure B.1: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.







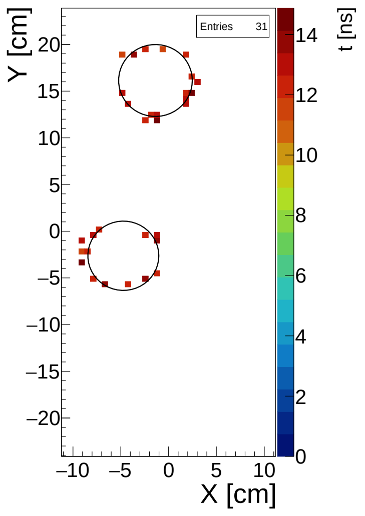


Figure B.3: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

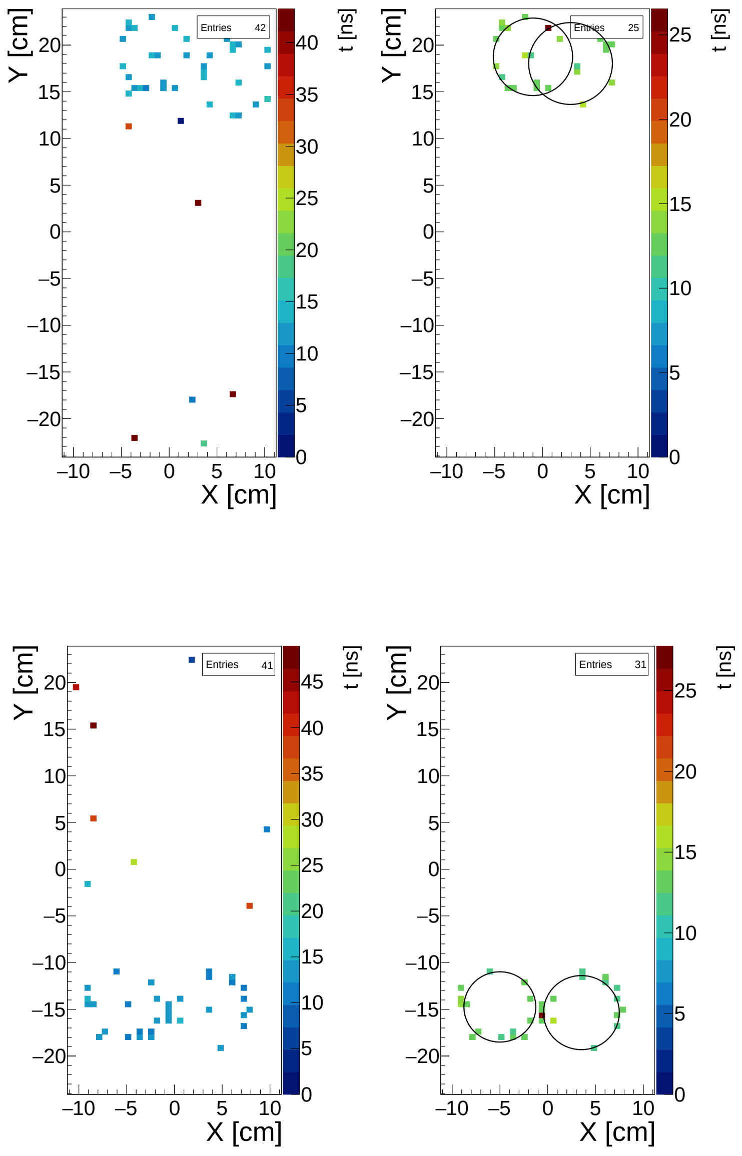


Figure B.5: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

B.2 New simulation events without noise removal

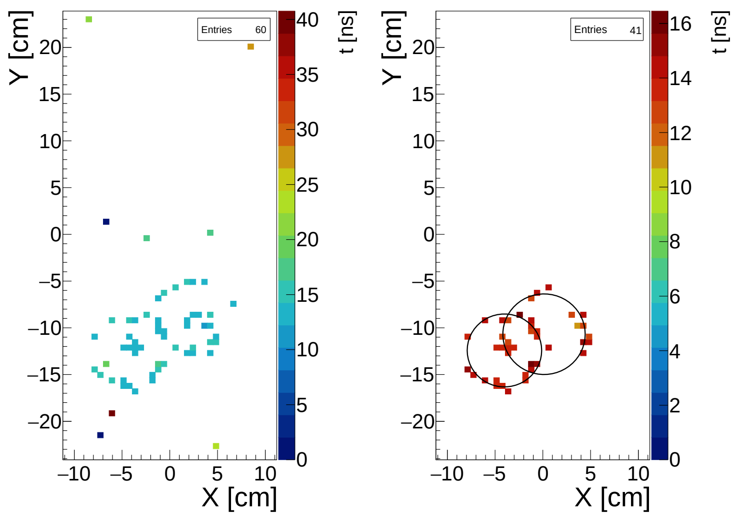


Figure B.6: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

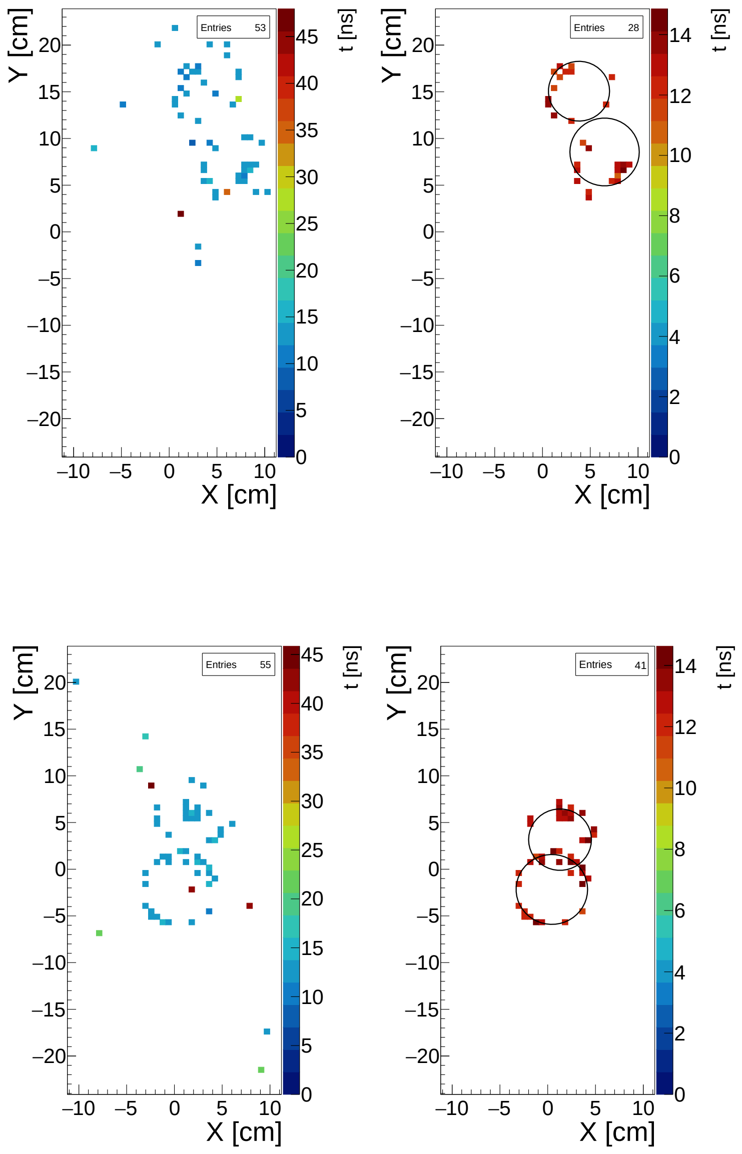


Figure B.8: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

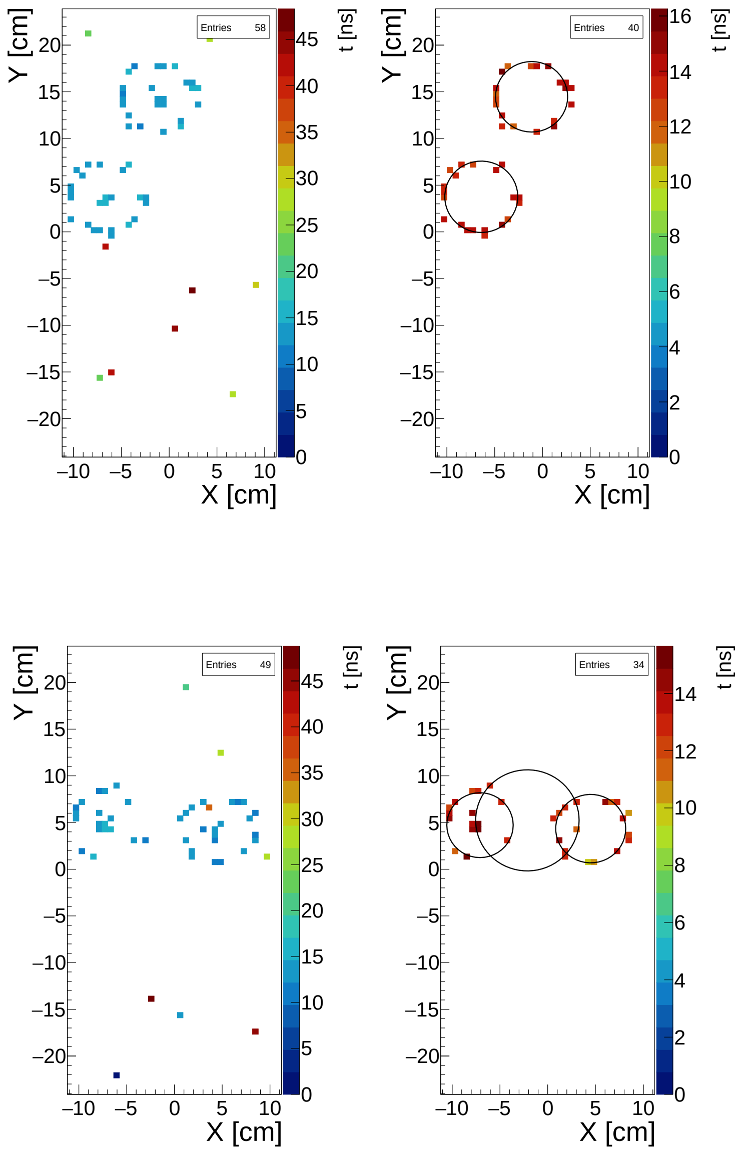


Figure B.10: Single event display. (Left) Hits in the mRICH plane with their time in the event color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

B.3 New simulation events using noise removal

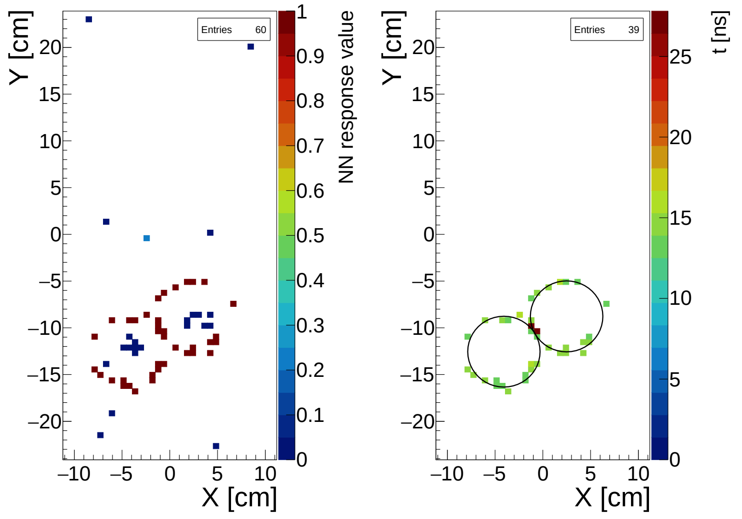
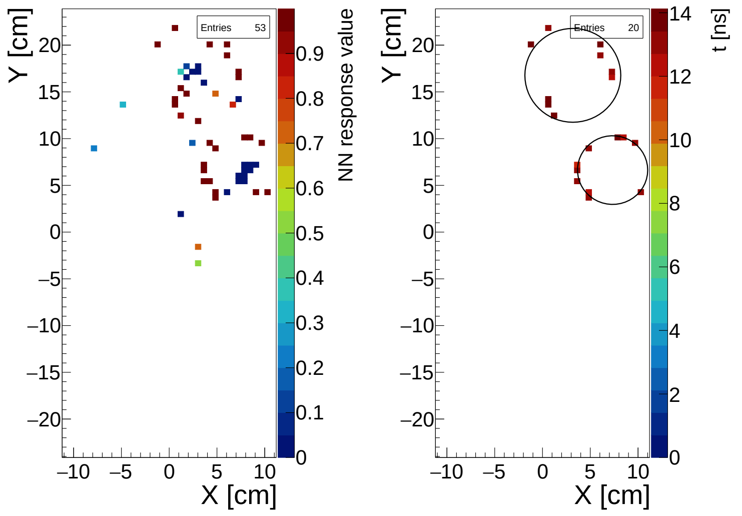


Figure B.11: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.



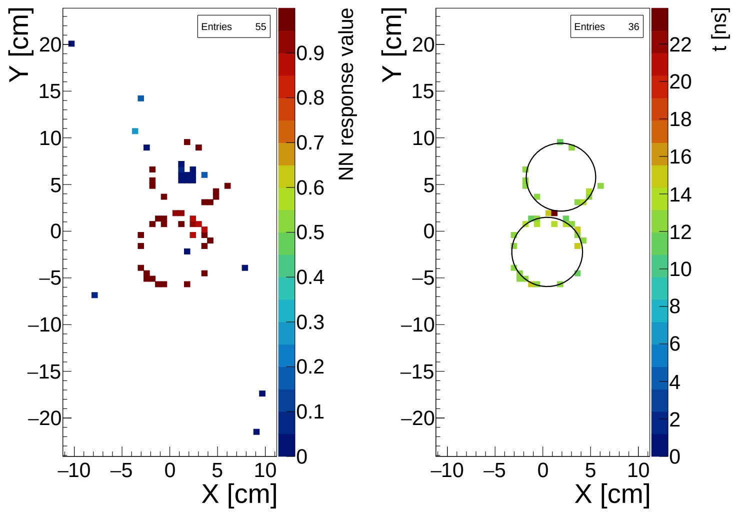


Figure B.13: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

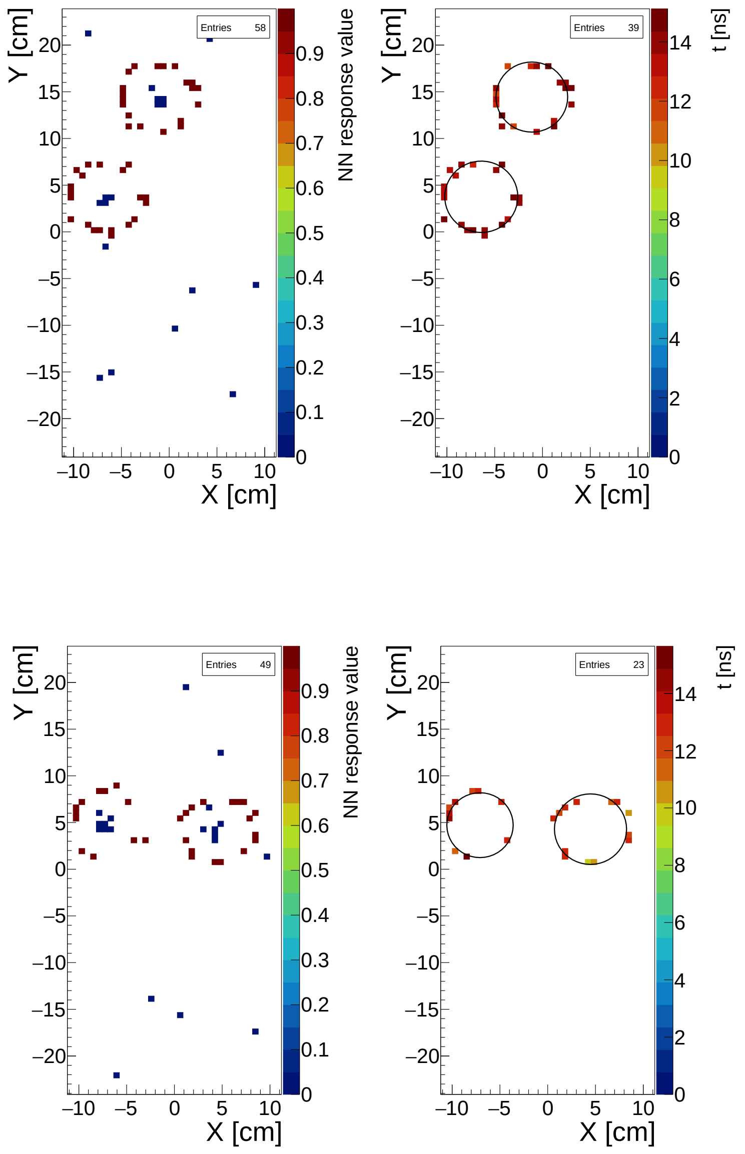


Figure B.15: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

B.4 Real data events using noise removal

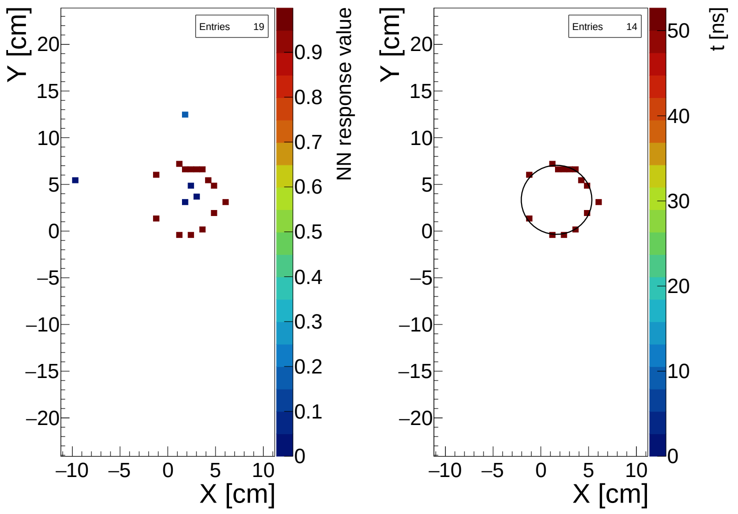
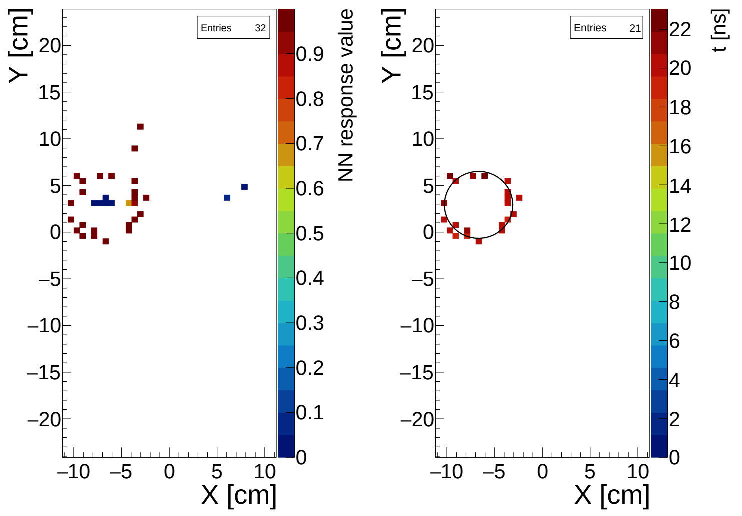


Figure B.16: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.



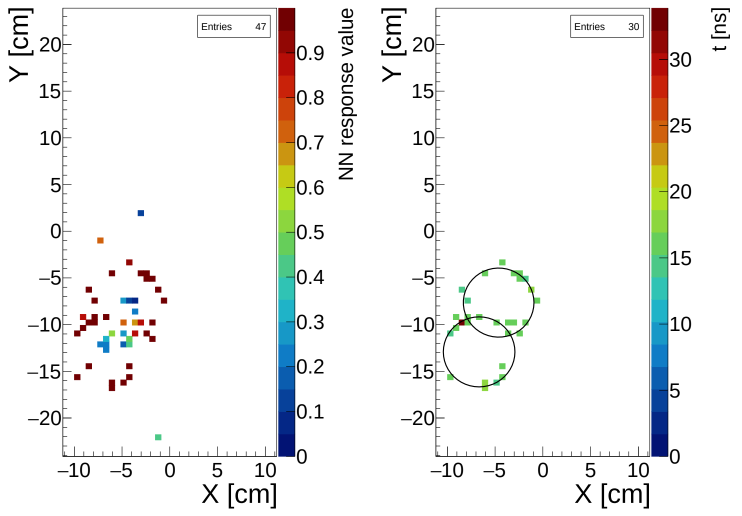


Figure B.18: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

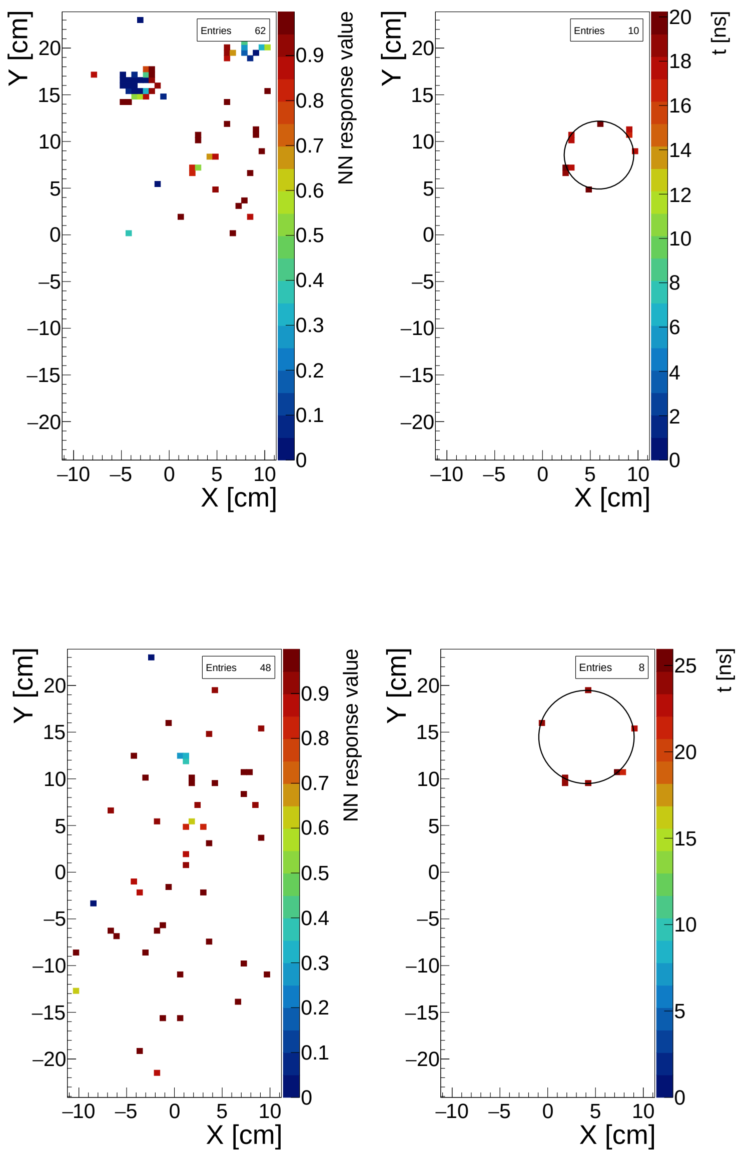


Figure B.20: Single event display. (Left) Hits in the mRICH plane with their NN response value color coded. (Right) Found rings and hits matched to the ring, with their time color coded.

Bibliography

[1] Hong Hui Tan and King Hann Lim. "Vanishing gradient mitigation with deep learning neural network optimization". In: *2019 7th international conference on smart computing & communications (ICSCC)*. IEEE. 2019, pp. 1–4.

[2] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[3] "PyTorch". In: URL: `https://pytorch.org`.

[4] Aston Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[5] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (Mar. 2016). eprint: `1603.07285`.

[6] Philippe Thévenaz, Thierry Blu, and Michael Unser. "Image interpolation and resampling". In: *Handbook of medical imaging, processing and analysis* 1.1 (2000), pp. 393–420.

[7] J Anthony Parker, Robert V Kenyon, and Donald E Troxel. "Comparison of interpolating methods for image resampling". In: *IEEE Transactions on medical imaging* 2.1 (1983), pp. 31–39.

[8] "Wikipedia, Receiver operating characteristic". In: URL: `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`.

[9] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29.

[10] Masaharu Tanabashi et al. "Review of particle physics". In: *Physical Review D* 98.3 (2018), p. 030001.

[11] Sheldon L Glashow. “The renormalizability of vector meson interactions”. In: *Nuclear Physics* 10 (1959), pp. 107–117.

[12] Steven Weinberg. “A model of leptons”. In: *Physical review letters* 19.21 (1967), p. 1264.

[13] Anthony Zee. *Quantum field theory in a nutshell.* Vol. 7. Princeton university press, 2010.

[14] Elena Santopinto and Giuseppe Galata. “Spectroscopy of tetraquark states”. In: *Physical Review C* 75.4 (2007), p. 045206.

[15] Hua-Xing Chen et al. “The hidden-charm pentaquark and tetraquark states”. In: *Physics Reports* 639 (2016), pp. 1–121.

[16] David d’Enterria et al. “High-precision *alpha_s* measurements from LHC to FCC-ee”. In: *arXiv preprint arXiv:1512.05194* (2015).

[17] Najmul Haque. “Some Applications of Hard Thermal Loop Perturbation Theory in Quark Gluon Plasma”. In: *arXiv preprint arXiv:1407.2473* (2014).

[18] Heng-Tong Ding, Frithjof Karsch, and Swagato Mukherjee. “Thermodynamics of strong-interaction matter from Lattice QCD”. In: *International Journal of Modern Physics E* 24.10 (2015), p. 1530007.

[19] Alexander Rothkopf. “Heavy quarkonium in extreme conditions”. In: *Physics Reports* 858 (2020), pp. 1–117.

[20] The CBM Collaboration. “mCBM SIS18”. In: 2020. URL: `https://indico.gsi.de/event/10698/contributions/44803/attachments/31410/39875/20200610-mCBM-proposal-21-22-fullVersion.pdf`.

[21] Adrian Amatus Weber. *Development of readout electronics for the RICH detector in the HADES and CBM experiments-HADES RICH upgrade, mRICH detector construction and analysis.* Tech. rep. 2021. URL: `http://dx.doi.org/10.22029/jlupub-288`.

[22] M Contalbrigo et al. “Aerogel mass production for the CLAS12 RICH: Novel characterization methods and optical performance”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 876 (2017), pp. 168–172.

[23] Pavel Alekseevič Čerenkov. "Visible radiation produced by electrons moving in a medium with velocities exceeding that of light". In: *Physical Review* 52.4 (1937), p. 378.

[24] N. Kolanoski H. & Wermes. "Teilchendetektoren: Grundlagen und Anwendungen". In: (January 2016). DOI: `doi:10.1007/978-3-662-45350-6.`.

[25] I. Manthos. "Signal processing techniques for precise timing with novel gaseous detectors". In: URL: `https://indico.cern.ch/event/1047066/contributions/4399272/attachments/2265836/3847074/Manthos_timing_hep21.pdf`.

[26] Jörg Förtsch. "Upgrade of the HADES RICH photon detector and first performance analyses". PhD thesis. Universität Wuppertal, Fakultät für Mathematik und Naturwissenschaften ..., 2022.

[27] D. Smith et al. "Event building in CBM". In: *CBM Progress Report 2021*. GSI. Darmstadt, 2022, pp. 176–177. URL: `http://repository.gsi.de/record/246663/`.

[28] Semeon Lebedev et al. "Ring recognition and electron identification in the RICH detector of the CBM experiment at FAIR". In: *Journal of Physics: Conference Series*. Vol. 219. 3. IOP Publishing. 2010, p. 032015.

[29] John Illingworth and Josef Kittler. "A survey of the Hough transform". In: *Computer vision, graphics, and image processing* 44.1 (1988), pp. 87–116.

[30] "The UrQMD Model". In: URL: `https://urqmd.org/`.

[31] Rene Brun et al. *GEANT 3: user's guide Geant 3.10, Geant 3.11*. Tech. rep. CERN, 1987.

[32] R Pestotnik et al. "The aerogel Ring Imaging Cherenkov system at the Belle II spectrometer". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 876 (2017), pp. 265–268.

[33] Martin Beyer, Semen Lebedev, and Claudia Höhne. "mRICH pattern recognition using convolutional neural networks". In: *CBM Progress Report 2021*. GSI. Darmstadt, 2022, pp. 92–93. URL: `http://repository.gsi.de/record/246663/`.

[34] Helge Voss et al. "TMVA, the toolkit for multivariate data analysis with ROOT". In: *XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research*. Vol. 50. SISSA Medialab. 2009, p. 040.

[35] "CbmRoot". In: URL: `https://git.cbm.gsi.de/computing/cbmroot`.

[36] "ONNX". In: URL: `https://onnx.ai`.

[37] "ONNX Runtime". In: URL: `https://onnxruntime.ai`.

[38] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. "Wider or deeper: Revisiting the resnet model for visual recognition". In: *Pattern Recognition* 90 (2019), pp. 119–133.

[39] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[40] "TensorFlow". In: URL: `https://tensorflow.org`.

[41] "Keras". In: URL: `https://keras.io`.

[42] "TensorBoard". In: URL: `https://www.tensorflow.org/tensorboard`.

[43] "Weights and Biases". In: URL: `https://wandb.ai/site`.

[44] "Submanifold Sparse Convolutional Networks". In: URL: `https://github.com/facebookresearch/SparseConvNet`.

[45] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics". In: *Machine Learning: Science and Technology* 2.2 (2020), p. 021001.

[46] Huimin Huang et al. "Unet 3+: A full-scale connected unet for medical image segmentation". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 1055–1059.

[47] Hao Li et al. "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems* 31 (2018).

[48] "High Performance Computing (HPC) cluster virgo". In: URL: `https://hpc.gsi.de/virgo/access/submit_nodes.html`.

[49] T Bellunato et al. “Study of ageing effects in aerogel”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 527.3 (2004), pp. 319–328.

[50] J. Förtsch et al. “Not published yet: Series Testing and Characterization of 1100 Hamamatsu H12700 Multianode Photomultiplier Tubes”. In: ().

Danksagungen

Zunächst möchte ich meiner Familie danken, für unschätzbare Unterstützung wärend des Studiums in finanzieller und moralischer Hinsicht. Besonderen dank gilt meiner Betreuerin Prof. Dr. Claudia Höhne, welche mich herzlich in dem Fachbereich aufgenommen hat und mit dem Vorschlag aufkam eine machine learning Anwendung zu entwickeln, was definitiv eine Leidenschaft in diesem sehr interessanten Themenberech bei mir erweckt hat. Speziellen dank auch an Adrian, der mich in den mRICH Detektor und das CbmRoot framework eingearbeitet hat und immer offen für eine sehr aufschlussreiche Diskussion war. Ebenso danke ich Semen für zahlreiche implementierungs Ratschläge und Hilfe bei Software Problemen. Auch danke ich Jan, der in Meetings stets sehr konstruktive Kritik an meiner Arbeit geäußert hat und den restlichen Leuten des Fachbereichs, Cornelius, Robin, Karina und Marten für interresante Diskusionen und eine sehr angenehme Arbeitsatmosphäre.