Masterthesis

Ring reconstruction in the MAPMT photodetector plane of the mCBM-RICH detector with the help of Convolutional Neural Networks

Ringrekonstruktion in der MAPMT Photodetektorebene des mCBM-RICH Detektors unter Verwendung von Convolutional Neural Networks

Zur Erlangung des akademischen Grades
Master of Science - Physik

JUSTUS-LIEBIG-
UNIVERSITÄT
GIESSEN

vorgelegt am II. Physikalischen Institut
Fachbereich 07 - Mathematik und Informatik, Physik, Geographie

---

Robin Haas **August 2022**

Autor:	**Robin Haas**
Matrikelnummer:	**4015991**
Abgabe:	**30.08.2022**
1. Gutachter:	**Prof. Dr. Claudia Höhne**
2. Gutachter:	**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 gute 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

Zusammenfassung

Ziel dieser Arbeit ist die Erprobung verschiedener Methoden zur Rekonstruktion von Cherenkov-Ringen in der MAPMT-Photodetektorebene des RICH Detektors des mCBM-Experiments. Die hier getesteten Methoden können später auf den vollständigen CBM-RICH übertragen werden. Diese Methoden beruhen auf künstlichen neuronalen Netzwerken — genauer gesagt Convolutional Neural Networks (CNN).

Das CBM (Compressed Baryonic Matter) Experiment ist als eines der Hauptforschungsprogramme für FAIR (Facility for Antiproton and Ion Research) neben dem GSI Helmholtzzentrum für Schwerionenforschung in Darmstadt geplant. Es handelt sich um ein Fixed-Target-Experiment zur Untersuchung von Schwerionenkollisionen mit Strahlen, die durch den SIS100-Beschleuniger von FAIR produziert werden. Einer der wichtigsten Subdetektoren für Elektronenidentifikation von CBM ist der Ring Imaging Cherenkov (RICH) Detektor. Dieser RICH Detektor benutzt eine Photodetektorebene aus MAPMTs, auf denen die Projektion von Cherenkov Kegeln — produziert von Elektronen — als Ringe abgebildet werden können. Die korrekte Bestimmung der Ringparameter ist von wesentlicher Bedeutung, da diese helfen die Elektronen zu identifizieren. Das mCBM-Experiment wurde 2018 erstmals in Betrieb genommen. Hauptziel ist die Etablierung und Erprobung der Free-Streaming Auslese, des Datentransport und der Online-Eventrekonstruktion. Darüber hinaus dient mCBM zum Testen von Prototypen der CBM-Subdetektoren sowie zur Optimierung von Strukturdesigns, Software und Hardware.

Diese Arbeit konzentriert sich auf die Ringrekonstruktion in der Photodetektorebene des mRICH, dem RICH Detektor des mCBM-Experiments. Momentan werden Ringe von CBM/mCBM mithilfe eines Algorithmus basierend auf der Hough-Transformations Methode (HTM) rekonstruiert. Für simulierte Teilchenkollisionen gibt es auch eine "ideale" Version dieses Algorithmus, der später als Referenz für den Vergleich des CNN-Ansatzes mit der Standard-HTM verwendet wurde.

Für die Ringrekonstruktion wurden zwei CNN-Modelle erstellt und getestet. Der erste Ansatz bestand darin, die Extraktion der Ringparameter als ein Regressionsproblem mit mehreren Zielen zu behandeln. Um das Netzwerk zu trainieren, wurde ein Toy-Modell erstellt um Bilder zu generieren — ähnlich denen, die in der Photodetektorebene des mRICH erzeugt werden — gepaart mit den Ringparametern als Zielvorgaben. Optisch sieht die Leistung des Modells im Vergleich zur idealen HTM gut aus. Weiterhin wurden die mittleren quadratischen Fehler des CNN-Modells sowie der HTM in Bezug auf die ideale HTM berechnet, welche für das CNN-Modell etwas kleiner ausfielen.

Beim zweiten Ansatz wurde zuerst ein Objektdetektor zur Erkennung von

Cherenkov-Ringen trainiert, welcher Begrenzungsrahmen um diese Ringe erstellt. Auf den extrahierten Rahmen wurde dann Ringregression für einzelne Ringe angewendet. Dieses Modell schneidet ebenfalls gut ab mit einem noch kleineren Fehlerwert als beim ersten Ansatz.

Insgesamt schnitten beide Modelle etwas besser ab als die HTM im Vergleich zur idealen HTM. Dennoch lagen die Vorhersagezeiten bei etwa 1 und 1000 Größenordnung höher für das erste bzw. zweite Modell im Vergleich zur HTM. Diese hohen Vorhersagezeiten können sicherlich verbessert werden, da sie in erster Linie auf eine mangelnde Modelloptimierung zurückzuführen sind.

Abstract

This work aims at testing different methods for the reconstruction of Cherenkov rings in the MAPMT photodetector plane of the mCBM experiment's RICH detector. Methods tested here can later be exported to the full CBM RICH. These methods are based on artificial neural networks — specifically convolutional neural networks (CNN).

The Compressed Baryonic Matter (CBM) experiment is planned to be one of the main research programs for the Facility for Antiproton and Ion Research (FAIR) next to the GSI Helmholtzzentrum für Schwerionenforschung in Darmstadt. It will be a fixed target experiment studying heavy ion collisions with beams produced by the SIS100 accelerator at FAIR. One of CBMs main subdetectors for electron identification is the Ring Imaging Cherenkov (RICH) detector. This RICH detector uses a photodetector plane made of MAPMTs on which the projection of Cherenkov cones — produced by these electrons — can be depicted as rings. Correctly identifying the ring parameters is essential as they will allow to identify the electrons. The mCBM experiment started first operation in 2018 with the main goal of establishing and testing the free-streaming readout, data transport and online event reconstruction. In addition, mCBM serves for testing prototypes of the CBM subdetectors and also for optimization of structural designs, software and hardware.

This work focusses on the ring reconstruction in the photodetector plane of the mRICH, which is the RICH detector of the mCBM experiment. At the moment, rings from CBM/mCBM are being reconstructed with an algorithm based on the Hough transform method (HTM). For simulated particle collisions there is also an "ideal" version of that algorithm, which later was used as a reference for comparing the CNN approach to the standard HTM.

Two CNN models were created and tested for ring reconstruction. The first approach was to treat the extraction of ring parameters as a regression problem with multiple targets. To train the network, a toymodel was created to generate images — similar to the ones created in the photodetector plane of the mRICH — paired with the ring parameters as targets. Visually the model's performance looks good when compared to the ideal HTM. Additionally, the mean squared errors of the model as well as the HTM with respect to the ideal HTM were computed, resulting in slightly smaller values for the CNN model.

In the second approach, first an object detector was trained to detect Cherenkov rings, which generated bounding boxes around these rings. On the extracted boxes ring regression for single rings was applied. This method performs also well with an even smaller error value than the first model.

Overall, both models performed slightly better than the HTM when compared to the ideal HTM. Nonetheless, the prediction times were about 1 and 1000 orders

of magnitude higher for the first and second model respectively when compared to the HTM. These high prediction times can certainly be improved as they are primarily caused by lack of model optimization.

Contents

List of Figures

1 Introduction

1.1 Standard Model

The fundamental findings of particle physics are summarized in the Standard Model. It describes all currently known elementary particles as well as their interactions with each other. These include the electromagnetic, weak and strong interactions. So far, the standard model does not include gravity, as it could not be formulated as a quantum theory yet.

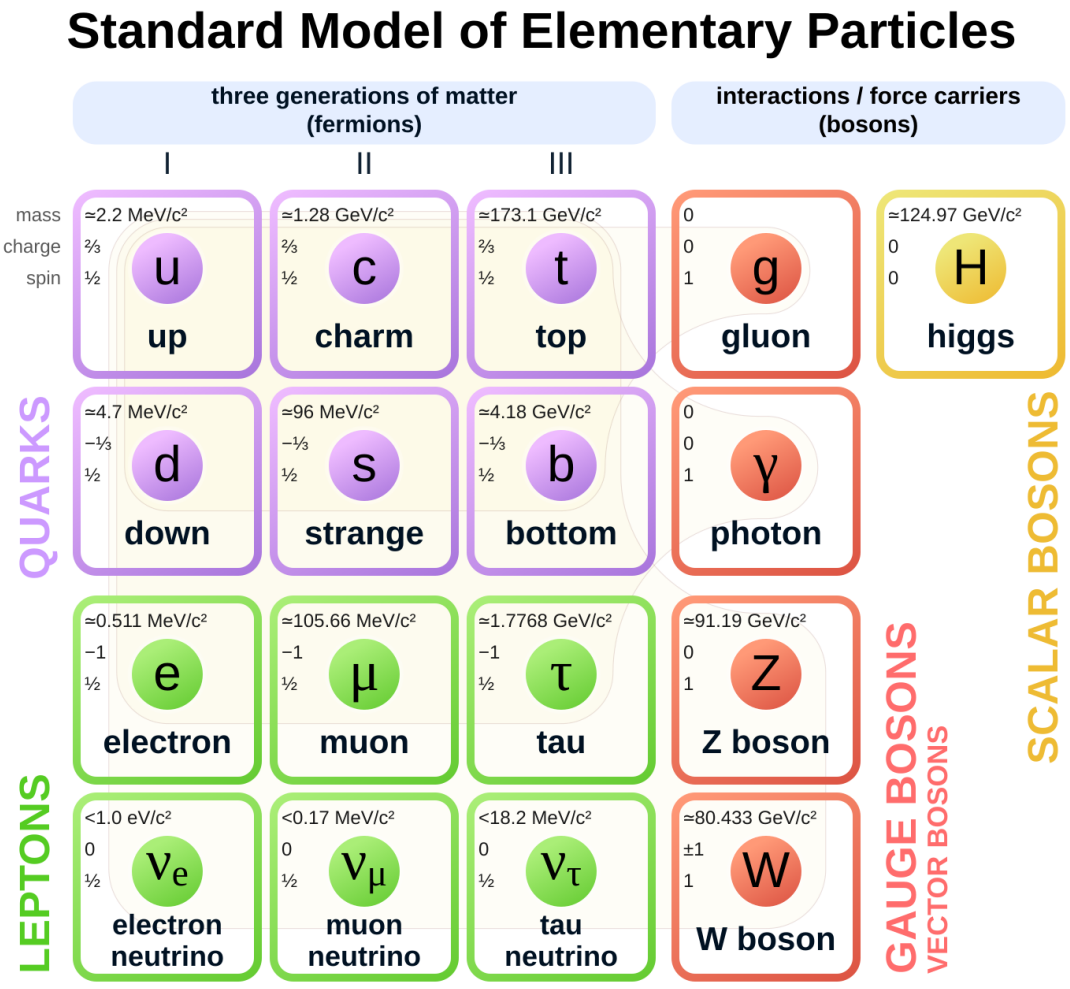

Figure 1.1: Elementary particles of the standard model [1].

Figure 1.1 shows all known elementary particles. These include quarks and leptons as well as the gauge bosons which mediate the aforementioned interactions. Of these particles merely up and down quarks and electrons are used to our surrounding form matter.

There are three quark generations with two quarks. Each quark can carry one of three color charges, which is the charge of the strong interaction. Taking into account the respective anti particles the number of quarks adds up to 36. The upper row of quarks (up, charm, top) has an electric charge of $\frac{2}{3}e$ while the lower row (down, strange, bottom) has values of $-\frac{1}{3}e$.

Leptons also come in three generations. The electron, muon and tau all have an electric charge of $-e$, whereas their corresponding Neutrinos ν_e, ν_μ and ν_τ carry no electric charge. Considering the anti particles the number of leptons sums up to 12 as leptons do not carry color charge.

Quarks can interact with other particles via all four fundamental forces, while e, μ and τ interactions exclude the strong interaction. Neutrinos on the other hand only participate in the weak interaction which makes them quite hard to detect.

With a spin value of $s = \frac{1}{2}$, which is a half-odd-integer, both quarks and lepton are classified as fermions.

The gauge bosons are considered as the force carriers and carry a spin of $s = 1$. The Photon (γ), which carries no electric charge itself and is massless, is the carrier of the electromagnetic interaction.

The Gluon (g) mediates the strong interaction. It is massless, too, with no electric charge but carries a color charge composed of a color (r, g, b) and an anti color ($\overline{r}$, $\overline{g}$, $\overline{b}$). With the help of group theory this results in a color octet and a color singlet giving a total of 9 color combinations. Considering, that the color singlet is colorless and consequently unable to change the color of quarks it cannot exist. Therefore, there are 8 different gluons in total.

The W^+, W^- and the Z^0 bosons are responsible for the weak interacion. Their electric charge is displayed in their exponent. Unlike the γ and the g these bosons are massive with about 80 $\frac{GeV}{c^2}$ for both W's and about 91 $\frac{GeV}{c^2}$ for the Z^0 boson.

Lastly, the Standard Model includes the Higgs boson. It is the quantum excitation of the Higgs field, which produces the masses of all elementary particles. The Higgs is electrically neutral and has a spin of $s = 0$.

Counting all mentioned (anti-)particles their total number in the Standard Model sums up to 61.

1.2 QCD phase diagram

The potential in a system of bound quarks can be seen in Figure 1.2. For small r the first part of the potential ($-\frac{4}{3}\frac{\alpha_S}{r}$) is predominant, resulting in a repulsive force between quarks similar to the electromagnetic interaction of same-charge particles. For large distances the potential is proportional to r. Thus, energy increases linearly with the distance between quarks. This is why free quarks cannot be observed. Even when increasing the energy of the system to move the quarks

further apart, at some point it is energetically more favorable to produce another quark-antiquark pair. However, this so called *confinement* can be overcome at

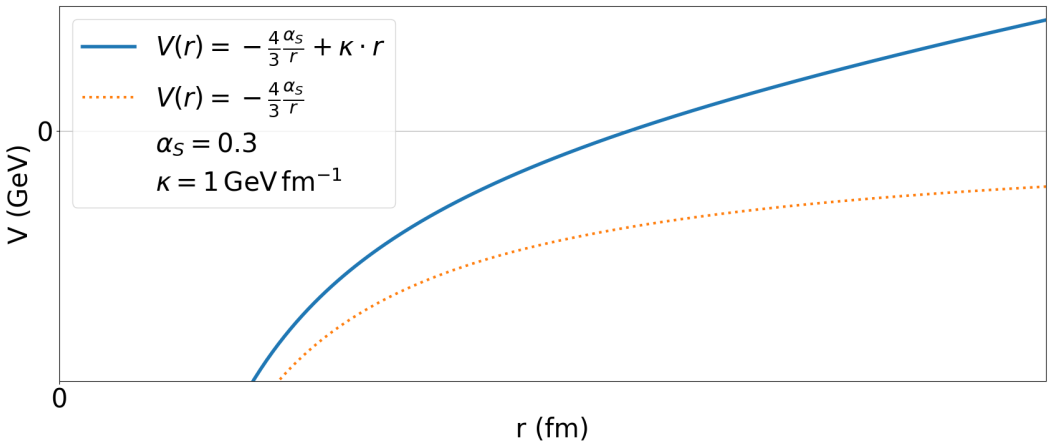

Figure 1.2: Potential of the strong interaction depending on the distance r between two bound quarks; α_S: strong interaction coupling constant, which is not constant like the name suggests; κ: field energy per length (string tension).

extreme temperature and pressure. This state is known as quark-gluon-plasma (QGP) in which quasi free quarks and gluons are ovservable. To verify this transition one can measure the degrees of freedom as free quarks have more of these than bound quarks. Following formular can be used to that end:

$$g = \frac{30 \cdot \epsilon}{T^4 \cdot \pi^2}$$

- g: degrees of freedom
- T: temperature
- ϵ: energy density

According to lattice QCD calculations the phase transition at $\mu_B = 0$ (baryon density $= 0$) occurs at a temperature of $T_C = 155\,MeV$ and an energy density of $\epsilon_c = 0.7\,\frac{GeV}{fm^3}$ [2]. This phase transition is a crossover transition and thus shown with fading out colours in Figure 1.3 at low net-baryon density. At large baryon densities a first order phase transition is conjectured, indicated with uncertainties by the yellow band in Figure 1.3. Hence, there has to be a critical point between the crossover and the first order phase transition, which has not been located experimentally nor theoretically yet. A common tool for studying the QCD phase diagram are heavy ion collisions as these can momentarily create extreme temperatures and energy densities. Different center-of-mass energies of

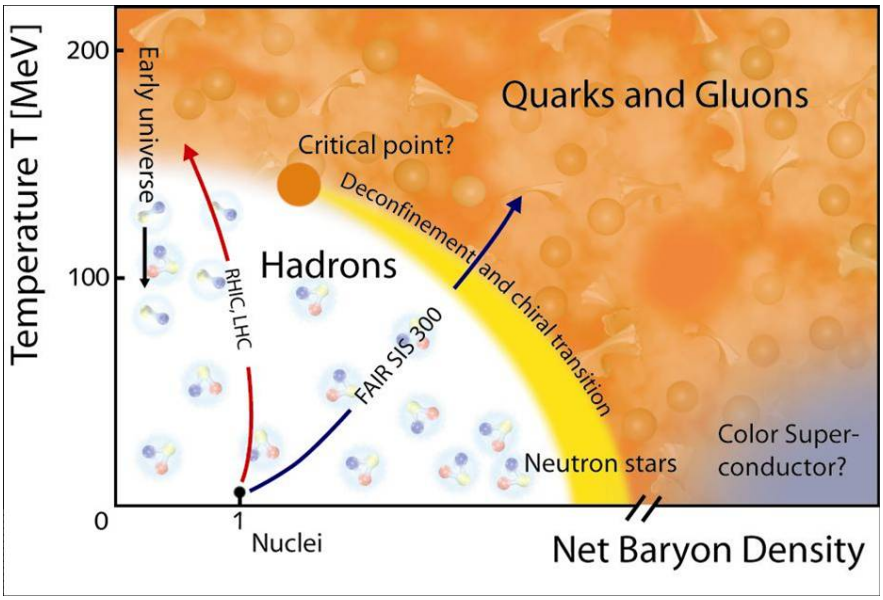

Figure 1.3: Conjectured QCD phase diagram[3].

the collisions allow the experiments to explore certain regions of the QCD phase diagram. For example, the Large Hadron Collider (LHC) at CERN explores very high temperatures with nearly zero net-baryon density. The created matter at these conditions is assumed to resemble the universe a few microseconds after the Big Bang. Studying the cooling process of this plasma might create insights in the development of the universe and formation of matter.

The currently constructed SIS100 accelerator at FAIR in Darmstadt on the other hand reaches up to 11 AGeV collision energy for Au+Au collisions [4]. Here, moderate temperatures but high baryon densities are created. Such matter is e.g. presumed to be created in the merger of neutron stars [5]. Furthermore, it has been shown by recent theoretical calculations, that the critical point — if it exists — might be included in the SIS100 range [6].

1.3 Dileptons as probes of the Quark-Gluon-Plasma

In heavy ion collisions a short-lived region of extreme density and temperature, the so called "fireball" is created. To analyse matter in such a state is quite challenging, since a lot of particles created in the fireball are also very short-lived. For example, the ρ-meson has a mean lifetime of $\tau \approx 4.5 \cdot 10^{-24}\,\mathrm{s}$, which is even shorter than the fireball's lifetime of about $\tau \approx 10^{-23}\,\mathrm{s}$.

In order to still be able to characterize the fireball, pairs of a lepton and the corresponding anti-lepton — e.g. $e^{\pm}$ and $\mu^{\pm}$ — are an excellent probe. As described in section 1.1, these dileptons do not take part in the strong interaction. Since the strong interaction takes the most part of the interactions inside the

fireball, dileptons can leave the fireball almost undisturbed. Thus, by analysing dileptons one can study the characteristics of the in-medium properties of hadrons and therefore of the fireball itself. The mass of the mother particle of a certain dilepton can be calculated if momentum and opening angle are known. These invariant masses from many collisions can then be combined to create an invariant mass spectrum as seen in Figure 1.4.

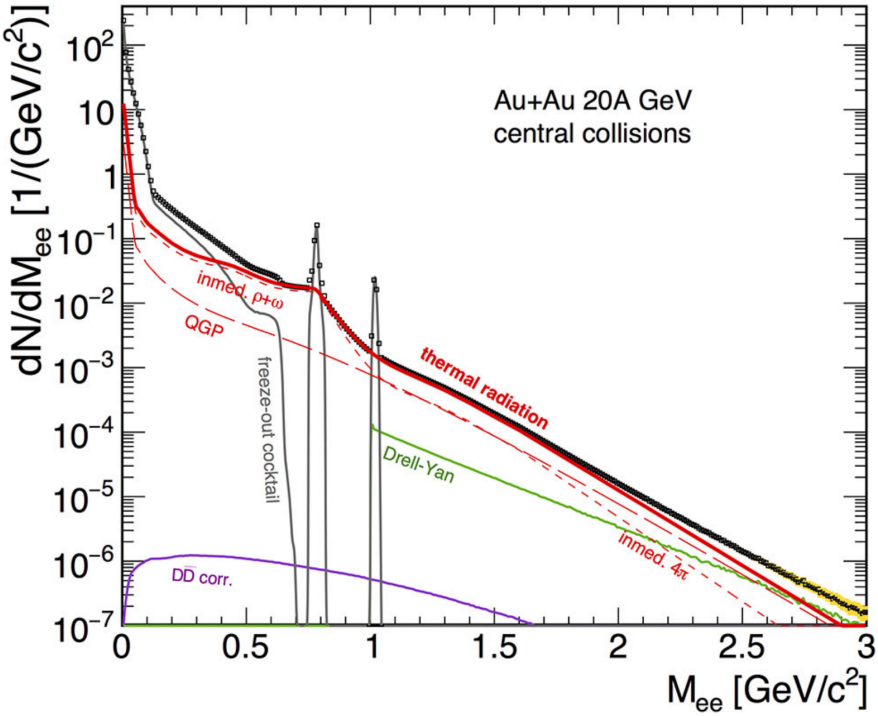

Figure 1.4: Expected invariant mass spectrum of $e^{\pm}$-pairs for central Au+Au collisions at 20 AGeV beam energy. Various $e^{\pm}$-sources are shown together with their impact on certain mass regions.[7].

One specific challenge of dilepton analysis is, that there are various sources of dileptons. In particular the numerous π^0-mesons decay in $\gamma\gamma$ with potential conversion of the photons into e^+e^-. This leads to a spectrum predominated by combinatorial background. This background mostly consists of uncorrelated dileptons, but there are also correlated ones originating from other sources — mostly the already mentioned π^0-decays and γ-conversions. γ may directly convert into a lepton pair, e.g. $\gamma \rightarrow e^+ + e^-$, while π^0 can decay via the π^0-Dalitz-decay $\pi^0 \rightarrow e^+ + e^- + \gamma$ (1%) or the much more common decay $\pi^0 \rightarrow \gamma\gamma$ (99%). In order to accurately describe the in-medium hadrons, it is essential to reduce this background as much as possible.

Most important is a high precision and efficiency in the electron identification. In CBM, this task will mainly be performed by the RICH detector, which will be explained in more detail in sections 2 and 3.

2 CBM Experiment

In the future the **C**ompressed **B**aryonic **M**atter experiment will be one of the main research pillars for the **F**acility for **A**ntiproton and **I**on **R**esearch (FAIR) which is located next to GSI Helmholtzzentrum für Schwerionenforschung in Darmstadt.

The aim of this fixed target experiment is to examine the high baryon density region of the QCD phase diagram with the help of high-energy nucleus-nucleus collisions. The CBM detector will be able to measure the behavior not only of hadrons (including rare diagnostic probes like multi-strange hyperons) but also charmed particles and lepton pairs with unprecedented precision and statistics. So far, the majority of these particles have not been studied in the FAIR energy range. Measurements will be carried out at reaction rates up to 10 MHz as means to reach the desired precision. Therefore, the experiment needs "very fast and radiation hard detectors, a novel data read-out and analysis concept including free streaming front-end electronics, and a high performance computing cluster for online event selection" [9]. Many of the prototypes, components and event reconstruction algorithms are already being used and tested in the miniCBM experiment with heavy-ion beams at the GSI-SIS18 accelerator during FAIR phase 0.

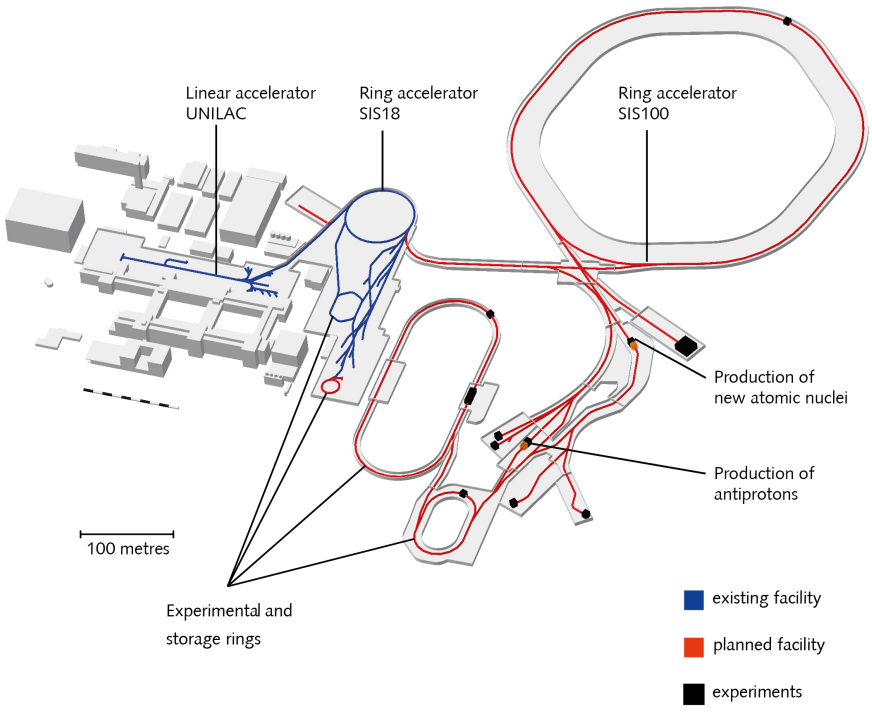

Figure 2.1: FAIR facility at GSI in Darmstadt [8].

2.1 CBM subdetectors

A schematic view of the full CBM detector together with the HADES detector is depicted in Fig. 2.2. The setup is shown with the RICH in place for electron measurements. For muon identification it will be replaced by the MUCH. In the following, each subdetector will be described shortly going throug the full setup from left to right.

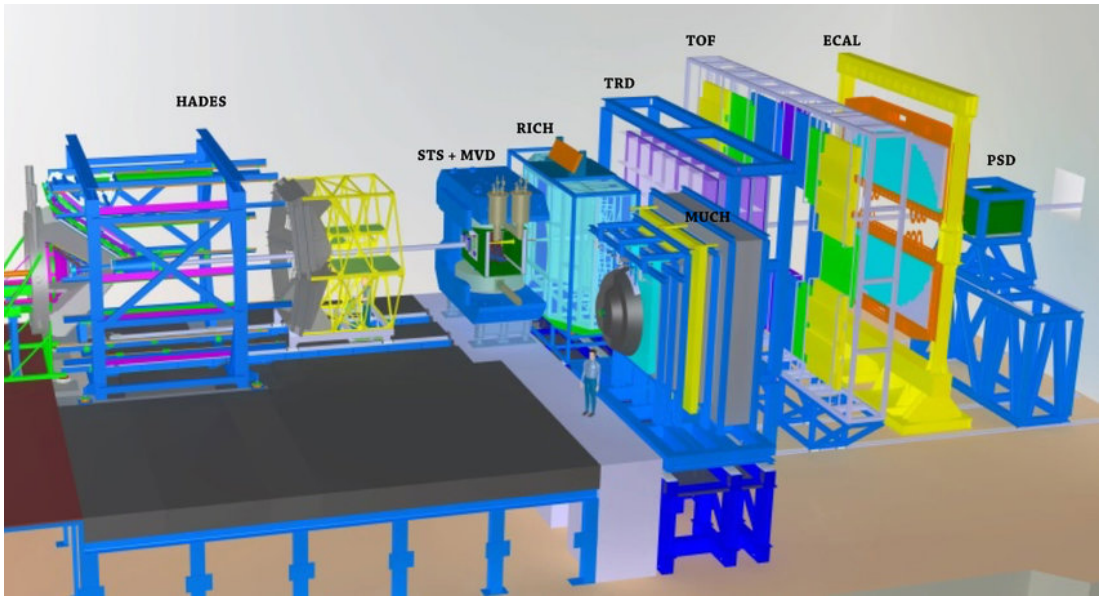

Figure 2.2: Sketch of the CBM detector shown in electron measurement arrangement [10].

2.1.1 Superconducting Dipole Magnet

In order to determine the momentum of charged particles a dipole magnet is used to bend the trajectory of these particles. Here a superconducting magnet is used, which has a magnetic field of 1 Tm in a radius of ≈ 0.5 m around the magnet's center with it's maximum at about 1 T.

2.1.2 MVD

Around 5-20 cm downstream in the magnet the **M**icro **V**ertex **D**etector is located. It is a small pixel detector consisting of four stations and is used to distinguish between primary and secondary vertices of short lived particles and delayed decays. This can significantly improve vertex and track reconstruction and is essential for good background reduction in the dilepton analysis. [11]

2.1.3 STS

Also located inside the dipole magnet is the **S**ilicon **T**racking **S**ystem which can be seen in Fig. 2.3. It is build from 8 stations which together can track particles inside the magnetic field in order to determine their momenta. The STS is mainly build from double sided silicon micro-strip sensors with a thickness of 300 µm which together sum up to about two million read-out channels [12]. Designed with a single hit resolution of 25 µm and a read-out strip pitch of 58 µm this results in a momentum resolution $\frac{\Delta p}{p}$ of about 1% [13].

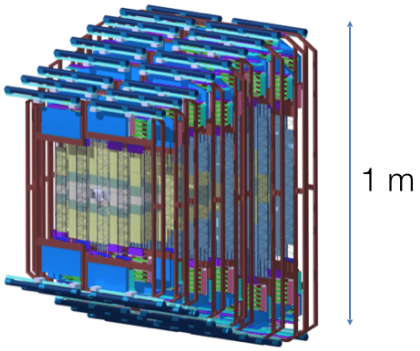

Figure 2.3: Schematic view of the STS used in the CBM detector [13].

2.1.4 RICH

Next detector downstream the beam pipe is the **R**ing **I**maging **CH**erenkov detector. It is used for electron identification and pion suppression for momenta up to 10 GeV. Carbondioxide (CO_2) will be used as radiator gas which has a refraction index of n = 1.00045 at 0°C and 1 atm [13].

Charged particles passing through the radiator gas at high enough velocity produce Cherenkov photons. These photons will be reflected by two sets of 6 mm thick mirror tiles and then be focused on detector planes. The mirror tiles are made from glass coated with Al+MgF_2 and have spherical shape with a radius of 3 m [15].

The detector planes are built from **M**ulti-**A**node **P**hoto **Multiplier T**ubes (MAPMT). They form an array of 14x7 MAPMT backplanes for each detector plane. Each backplane carries 2x3 MAPMTs. More details on Cherenkov radiation and RICH detectors follow in section 3.

2.1.5 TRD

The **T**ransition **R**adiation **D**etector is also used to identify electrons but starting at higher momentum than the RICH (1.5-2 GeV). The TRD contains a radiator material in which a Multi-Wire Proportional Chamber (MWPC) is embedded.

Transition radiation will be produced for charged particles propagating through the radiator, which will then be absorbed in a readout chamber. The TRD yields information on the specific energy loss and in combination with TOF can also deduce the charge of the particles. This not only helps in further suppressing pions but also makes it possible to identify light nuclei.

2.1.6 TOF

A tool for hadron identification is the **T**ime-**O**f-**F**light detector. It measures the time difference of a common start to the individual stop of each particle in the various TOF layers. With the help of previous detectors like the STS and TRD, particles can also be tracked in the CBM setup. The measured position and time information can be used to compute the particles' velocity β [15]. Together with the momentum provided by tracking in the STS, β can be used to calculate the mass of the particle with the following equation:

$$m^2 = p^2 \cdot (\frac{1}{\beta^2} - 1)$$

2.1.7 ECAL

For additional electron and photon identification, an **E**lectromagnetic **CAL**orimeter is planned for CBM. It would be a "Shashlik" type calorimeter, which contains alternating stacks of absorber (here: lead) and scintillator materials connected to wavelength shifting fibers. Electrons and photons traversing the ECAL lose energy by electromagnetic showers inside the absorbtion material which in turn is measured as visible light inside the scintillator material. The intensity of the light from multiple scintillator stacks can then be used to determine the energy of the particles.

2.1.8 PSD

The final detector in the CBM setup is the **P**rojectile **S**pectator **D**etector. Like the ECAL it is also designed with alternating lead and scintillator stacks attached to wavelength shifting fibers. Its purpose is to measure the collision centrality as well as the event plane orientation, which are dependent on the projectile nuclei fragment (spectator) positions and energy determined by the PSD.

2.2 mCBM experiment

The mCBM experiment started operation in 2018 with the goal of testing the detector chain with the final free-streaming/triggerless readout and online data reconstruction. Furthermore, mCBM not only enables testing the prototypes for the CBM subdetectors in the real CBM environment, but also helps with refining

the design of structures, software and hardware. Figure 2.4 shows the mCBM detector setup in March 2020 with six protype subdetectors. Even though this setup is lacking a dipole magnet, which usually is neede for reconstruction of the momentum of particles, the reconstruction of Λ baryons should still be possible by tracking the secondary decay vertex [13].

mCBM is located next to the HADES experiment and is being operated with the SIS18 Heavy Ion beam.

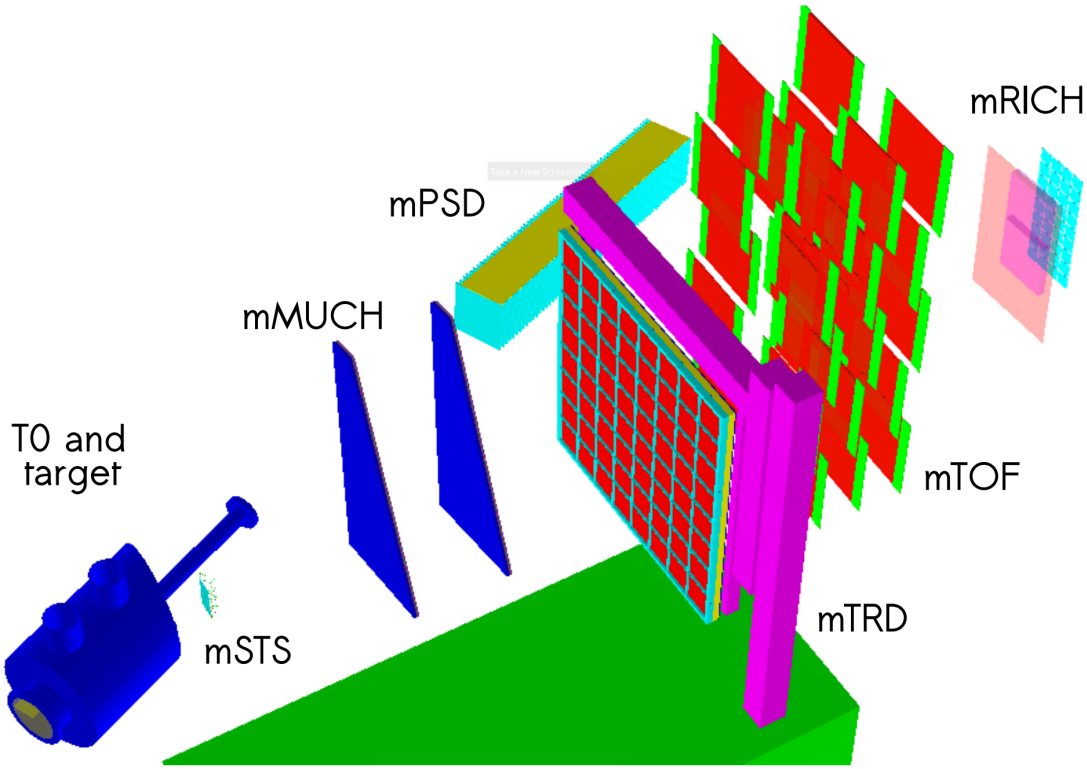

Figure 2.4: Geometry of the mCBM experiment in March 2020 [13]. All detectors are positioned at an angle of 25° with regard to the beampipe, except the mPSD which is rotated by only 5°.

2.2.1 mRICH

The mRICH is the core subdetector for ring reconstruction and for testing the free-streaming RICH DAQ [13]. Unlike the CBM RICH which uses mirrors for focussing Cherenkov Rings, the mRICH is a proximity focussing RICH detector. More details on these design types follow in section 3.2.

A schematic drawing as well as a view of the inside of the mRICH can be seen in Figure 2.5. The size of the mRICH is considerably smaller than that of the CBM RICH. The full photodetector plane only consists of 36 MAPMTs which form a

rectangle the size of 9 by 4 (Fig. 2.5 right). 10 cm in front of the photodetector plane two radiator blocks made of aerogel, which is a popular radiator material for Cherenkov detectors, are mounted. It is a low density material that can be manufactured with various refraction indices. The mRICH uses the same aerogel type as used by the CLAS12 collaboration RICH detector [13], which has a refraction index of 1.05 at 405 nm [14], resulting in $\gamma_{thr} = 3.27$ [13] and a pion threshold momentum of $p_{thr} = 436\,\text{MeV}$ [13].

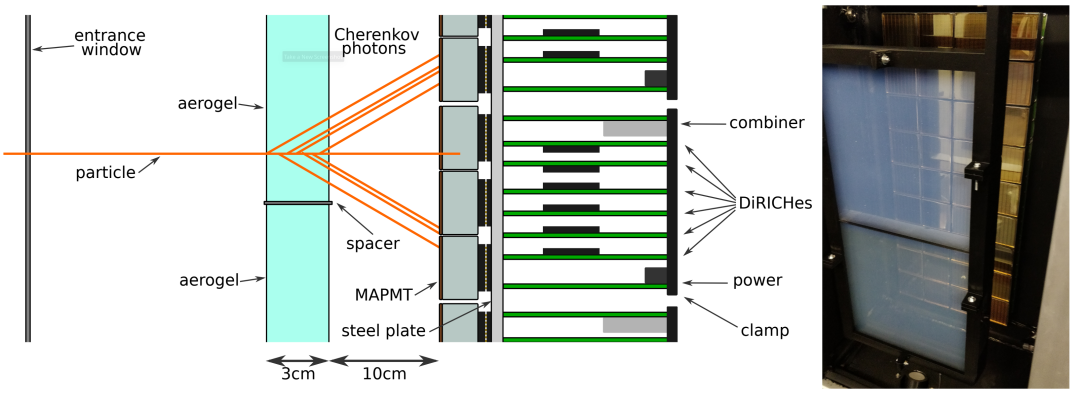

Figure 2.5: **left:** Schematic side view of the mRICH detector [13]. **right:** Inside view of the mRICH. The two radiator blocks are mounted 10 cm in front of the photodetector plane [13].

3 Ring Imaging Cherenkov detectors

This section will focus on RICH detectors in more detail, as this work deals with signals from a RICH detector. First the Cherenkov Effect will be explained, which is exploited by RICH detectors in order to identify charged particles. Then the two types of RICH detectors used by CBM and mCBM will be presented.

3.1 Cherenkov Radiation

The Cherenkov Effect was first experimentally observed in 1934. The discovery was made by the Soviet scientist Pavel Cherenkov under the supervision of Sergey Vavilov, of which a theory was later published in 1937.

Charged particles traversing a dielectric medium create a local polarisation of molecules (Fig. 3.1). If the velocity v of that particle is smaller than the speed of light in that medium, the polarisation is symmetric and not observable from the outside. With a velocity higher than the speed of light in that medium, the created polarisation no longer is symmetric and can be observed globally as the so called Cherenkov radiation. Analogous to the shock front of a supersonic aircraft, the Cherenkov effect also results in a wave front created by constructive interference of Cherenkov radiation. Photon emission then is perpendicular to

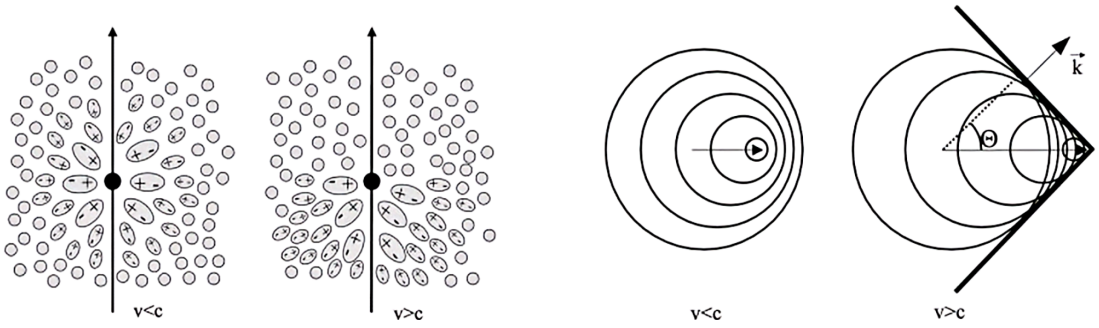

Figure 3.1: **left:** Polarisation of molecules in a dielectric medium by charged particles with $v < c$ and $v > c$ ($c = c_{vacuum}/n$); **right:** wave front created by charged particles with a velocity larger than the speed of light in the medium; the dotted line is of length $\frac{c}{n} \cdot \Delta t$ while the continuous line is of lenght $\beta \cdot c \cdot \Delta t$ [17].

the wave front as indicated by $\vec{k}$ in Figure 3.1 (right). The angle of the photon emission depends on the velocity β and the refractive index n of the medium according to the following formular [16] (see Fig. 3.1 for lenghts):

$$\cos\Theta = \frac{\frac{c}{n} \cdot \Delta t}{\beta \cdot c \cdot \Delta t} = \frac{1}{\beta \cdot n} \tag{3.1}$$

Here, the recoil of the photo emission is neglected, but quantum mechanical calculations show equation 3.1 to be valid for most cases [16].

Since the particle velocity cannot exceed the vacuum speed of light ($\beta = 1$), a maximum angle for Cherenkov radiation can be derived:

$$\cos\Theta_{max} = \frac{1}{n} \tag{3.2}$$

Furthermore, the relation $\beta \geq \frac{1}{n}$ can be deduced from equation 3.1 requiring $\cos\Theta \leq 1$, which defines the threshold velocity for Cherenkov photon creation:

$$\beta_t = \frac{1}{n} \tag{3.3}$$

3.2 RICH detectors

If particles above the Cherenkov threshold pass through a radiator with $n > 1$, they can constantly emit photons at an angle according to equation 3.1. This produces parallel Cherenkov cones along the particle trajectory. For a proper analysis of the Cherenkov radiation and the particle itself, there are several possible RICH design concepts.

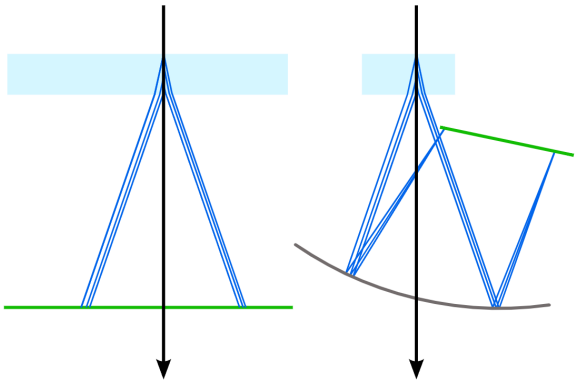

Figure 3.2: Proximity focussing and focussing RICH detector design types. **left:** Cherenkov radiation directly hits the photo detector plane. **right:** Cherenkov radiation will be focused on the photodetector plane by a spherical mirror. **cyan:** radiator, **green:** photodetector plane.

One possibility is to use a mirror focussing RICH detector, which is planned to be used for the main CBM experiment. Here, the Cherenkov photons are created in a large volume of radiator gas. They travel in particle direction towards a mirror which reflects and focusses them to its focal point (Fig. 3.2 right). A photodetector is used to detect the 2D projection of the Cherenkov cones, which become a single ring. With the mirror radius R_s and the focal length $f_s = \frac{R_s}{2}$ the Cherenkov ring radius can be approximated [16]:

$$R_c = f_s \cdot \Theta_c = \frac{R_s \cdot \Theta_c}{2} \Rightarrow \Theta_c = \frac{2R_c}{R_s} \tag{3.4}$$

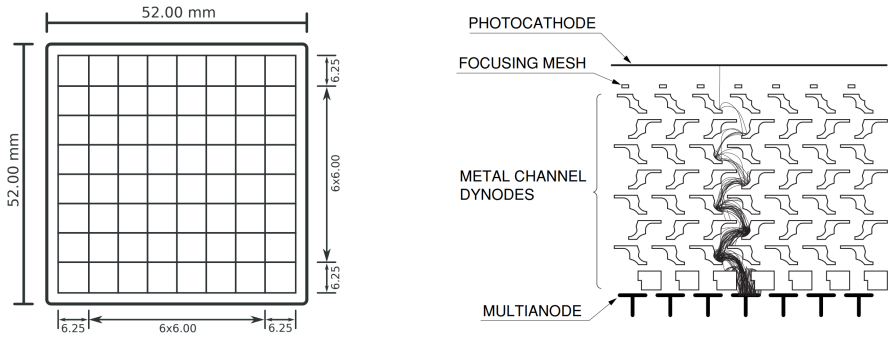

Figure 3.3: **left:** Hamamatsu type H12700B MAPMT with dimensions and pixel sizes [13], **right:** Electrode structure with an example of an electron cascade [19].

Together with equation 3.1 the particle velocity can be calculated. Additionally, using $p = \beta\gamma mc$ the mass of the particle can be determined. For that, its momentum has to be known, which can be obtained by a preceding tracking detector in a magnetic field. The STS covers this task for the CBM experiment (see 2.1.3).

$$m = \frac{p}{c}\sqrt{\frac{1}{\beta^2} - 1} = \frac{p}{c}\sqrt{(n \cdot \cos\Theta_c)^2 - 1} = \frac{p}{c}\sqrt{\left(n \cdot \cos\left(\frac{2R_c}{R_s}\right)\right)^2 - 1} \qquad (3.5)$$

Another possible design is the proximity focussing RICH detector, which is depicted left in Figure 3.2. Here, the radiator is typically made from liquid or solid material. Since this type does not use focussing elements like mirrors, the radiator has to be thin. This ensures that the Cherenkov cones approximately appear to be coming from the same origin. Like in the above mentioned focussing RICH type also rings can be observed on the photodetector plane, but their sharpness is highly dependent on the radiator thickness. Another difference to the focussing RICH is that particles traverse the photodetector itself. The downside of a thin radiator is that it cannot produce as much Cherenkov photons as a large volume of radiator gas. On the other hand, it makes a more compact design possible, which is why it was used to build the miniCBM detector.

As mentioned in section 2.1.4, the photodetector plane is build from arrays of **M**ulti-**A**node **P**hoto **M**ultiplier **T**ubes (MAPMT). The MAPMTs used in both, the CBM RICH as well as the mRICH, are of type Hamamatsu H12700B [20]. A schematic of the top view of that type can be seen in Figure 3.3 (left).

The functionality of these MAPMPTs is based on the external photoelectric effect [19]. Incident photons can emit several electrons, which travel to the first dynode due to the applied voltage. These photoelectrons then also emit multiple secondary electrons in that dynode, which travel to the next dynode, and so on until they finally hit the anode. The electron multiplication in such a cascade can

amplify the current by 10 to 10^8 times [19]. This process is depicted in Figure 3.3 (right). This amplification allows MAPMTs to measure single photons, which is essential for a RICH detector.

4 Artificial neural networks

Today, artificial neural networks (ANN) are very prominent in machine learning and were created to mimic the ability of learning in biological organisms. They use labeled training data, which means input data is correctly mapped to output data, in order to predict outcomes of unseen data. Therefore, artificial neural networks belong to supervised machine learning techniques. Examples of possible applications of ANNs are function approximation, pattern recognition and classification [21].

Figure 4.1a shows a single biological neuron, as well as the connection, also called *synapse* (Fig. 4.1c), of two neurons with the help of their *dendrites* and *axons*. Furthermore, a single artifical neuron can be seen (Fig. 4.1b) as well as the synapses of a simple ANN. Even though the comparison to biological neural nets is often criticized for being a poor representation of the brain's function, it has often helped design new ANN architectures [21].

This section will give an introduction to the basic architecture of neural networks and also explain Convolutional Neural Networks (CNN), which perform especially well in pattern recognition of images and was used as the main tool of this work for Cherenkov ring reconstruction in the mCBM RICH detector (see section 3.2).

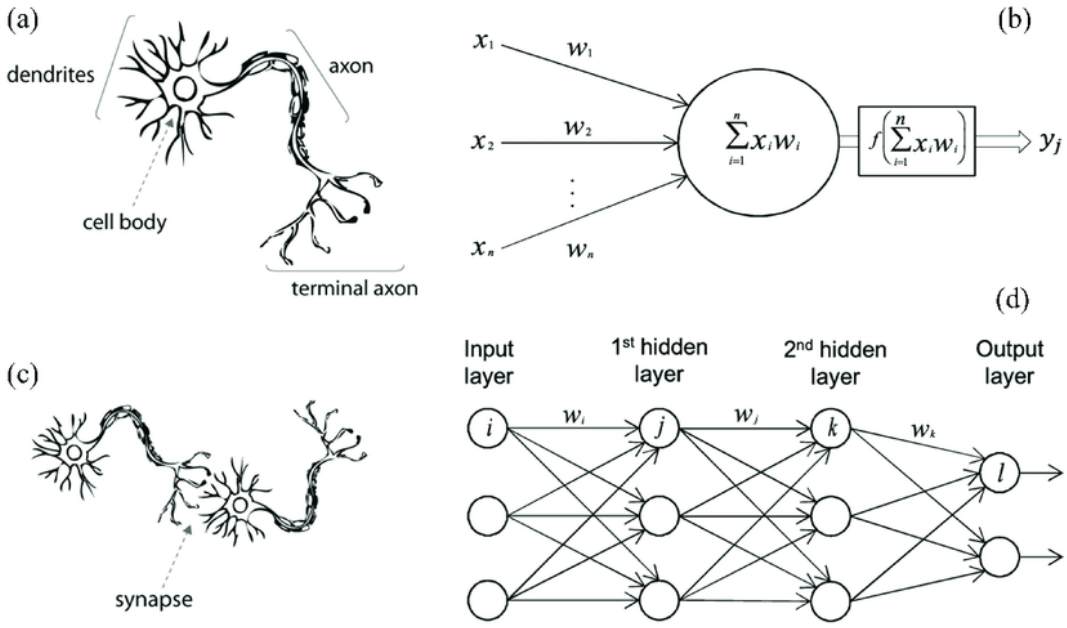

Figure 4.1: Biological neurons compared to artificial neurons/networks: **(a)** human neuron; **(b)** artificial neuron; **(c)** biological synapse; **(d)** ANN synapses [22].

4.1 Perceptron

The simplest possible neural network, called the *perceptron*, contains a single computational layer and an output node. Figure 4.2 depicts such a perceptron. In a given training dataset each set of m *feature* variables $\overline{X} = [x_1, ..., x_m]$ is assigned to an output y, which contains the *observed* value or *target* value. The inputs $\overline{X}$ also come with individual weights $\overline{W} = [w_1, ..., w_m]$ which can be changed during training. The goal now is to train the perceptron on this dataset, so that

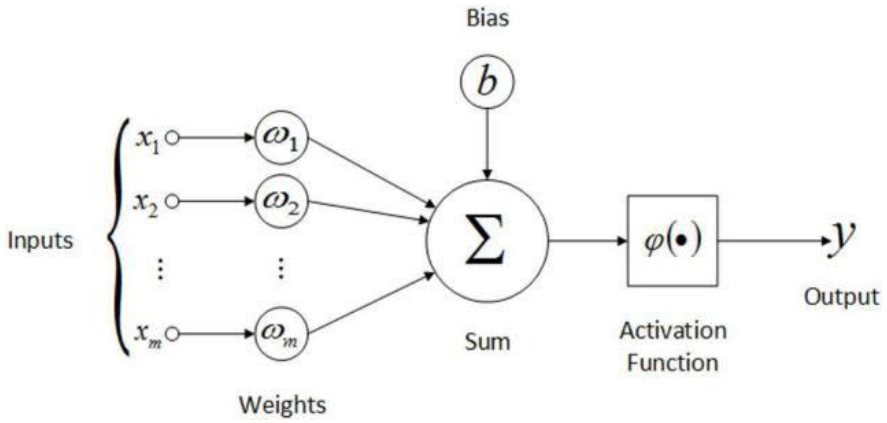

Figure 4.2: Architecture of the perceptron [23].

it can correctly predict the outcome of unseen data. Taking a binary classification task as an example (meaning $y \in \{-1, 1\}$) the prediction can be calculated by multiplying $\overline{X}$ and $\overline{W}$ and feeding the result to an activation function (here: sign function) [21]:

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X}\} = \text{sign}\{\sum_{j=1}^{m} w_j x_j\} \tag{4.1}$$

Here, $\hat{y}$ denotes the predicted value, while y is the true value. The sign function as well as some other popular activation functions can be viewed in Figure 4.3. Problems arise when training on data with imbalanced class distributions, e.g. if

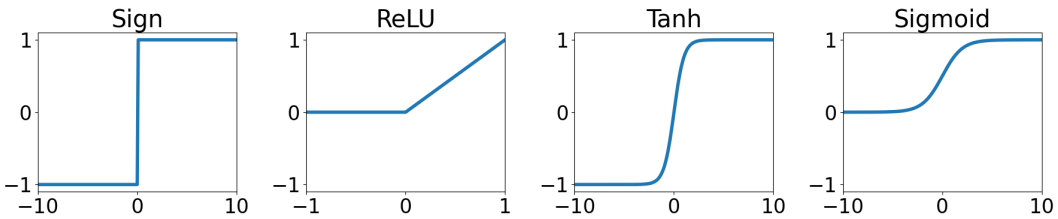

Figure 4.3: Commonly used activation functions for neural networks.

the majority of the data belongs to only one class. To prevent wrong predictions

in such cases, an additional invariant variable, called *bias*, will be introduced. This bias will also be fed into the sum, as can be seen in Fig. 4.2, which changes equation 4.1 to

$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\{\sum_{j=1}^{m} w_j x_j + b\} \tag{4.2}$$

The next step is to update the weights depending on the prediction error $E(\overline{X}) = y - \hat{y}$ and an additional parameter α, called the *learning rate*, which defines how fast the network learns.

$$\overline{W} \Leftarrow \overline{W} + \alpha E(\overline{X})\overline{X} \tag{4.3}$$

Typically, a loss function L_i is defined that the network tries to minimize during training. The weight updates can then be defined as *gradient-descent* of that loss function

$$\overline{W} \Leftarrow \overline{W} - \alpha \nabla_W L_i = \overline{W} - \alpha \frac{\partial L}{\partial \overline{W}} \tag{4.4}$$

When feeding training data in random order into the network this method is referred to as ***s**tochastic **g**radient-**d**escent* (SGD) or *mini-batch stochastic gradient-descent* when feeding data as subsets into the network instead of one at a time. Once the whole dataset was fed to the network it will be repeated several more times. One such cycle is denoted as an *epoch*.

4.2 Multilayer Perceptron

Like the name suggests, the **M**ulti**l**ayer **P**erceptron (MLP) cotains more than a single layer. It has additional layers in between the input and output layer, which are called *hidden layers*, since the computations in such layers are invisible to the user. An example MLP with 3 layers can be seen in Figure 4.1d. Note, that because no computations are performed inside the input layer it is typically not considered a layer itself. The depicted MLP does not have any bias nodes, but like in the perceptron these can be added to every computational layer in an MLP. Furthermore, because information is fed from input to output layer in forward direction, the MLP is also referred to as a *feed-forward* network.

4.3 Training neural networks

4.3.1 Backpropagation

Like it was demonstrated in section 4.1, updating weights for the perceptron was straightforward as the loss function directly depends on the weights and the bias. In the case of multilayer networks the computation of the loss function is more complex, because the outputs of hidden layers depend on the weights and biases of previous layers. Today, almost all kinds of artificial neural networks are trained

by utilizing the *backpropagation* algorithm [24], which basically consists of two phases.

In the *forward phase* input samples will be fed into the network and the prediction $\hat{y}$ will be calculated with the initial weight values. Together with the true output y the derivative of the loss function can be calculated with respect to the output.

Next, in the *backward phase* the derivative of the loss function will be computed with respect to every weight in the network, in order to update the weights via the gradient-descent algorithm. Since the output is a composition function of all the nodes in the network, these computations are not trivial. Figure 4.4 shows the computational graph of a simple network with two paths that will be used for an example calculation. Here, the derivative of $\hat{y}$ with respect to the weight w will be

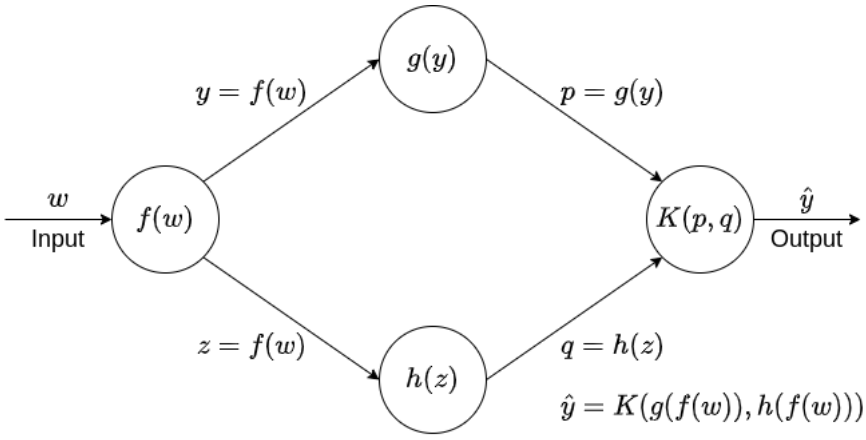

Figure 4.4: Demonstration of the chain rule in computational graphs; adapted from [21].

calculated. With the multivariate chain rule this derivative can be expressed as

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial \hat{y}}{\partial p} \cdot \frac{\partial p}{\partial w} + \frac{\partial \hat{y}}{\partial q} \cdot \frac{\partial q}{\partial w} \tag{4.5}$$

Applying the univariate chain rule to $\frac{\partial p}{\partial w}$ and $\frac{\partial q}{\partial w}$ yields

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial \hat{y}}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial w} + \frac{\partial \hat{y}}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial w} \tag{4.6}$$

This can also be written as

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial K(p,q)}{\partial p} \cdot g'(y) \cdot f'(w) + \frac{\partial K(p,q)}{\partial q} \cdot h'(z) \cdot f'(w) \tag{4.7}$$

These calculations could be done seperately for each weight in the network, but its easy to see that this would be very inefficient for larger networks with significantly

more paths. It is also evident, that the derivatives close to the output will be used multiple times by different weights in previous layers. The backpropagation leverages this fact. First the derivatives in layer N closest to the output will be computed, then the derivatives in layer $N-1$ and so on; hence the name backpropagation. Once all the derivatives are calculated, the weights will be updated with the gradient-descent algorithm.

4.3.2 Under- and Overfitting

A common problem in training neural networks in general is *under-* and *overfitting*. This happens, when the architecture of the network and thus the complexity of the model described by the network is not chosen well for the problem at hand. Figure 4.5 shows an illustration of under- and overfitting and the complexitiy dependence of the loss.

Models with very low complexity have trouble converging to a low loss for both training and testing data. The model does not have enough parameters to accurately learn the features of the given data. This is referred to as underfitting.

Overfitting on the other hand may occur, when the model complexity is too high and the amount of training data is very limited. This leads to a network that does not generalize very well and rather memorizes data instead of learning its features. Therefore, the training loss is very low but the testing loss is high. To

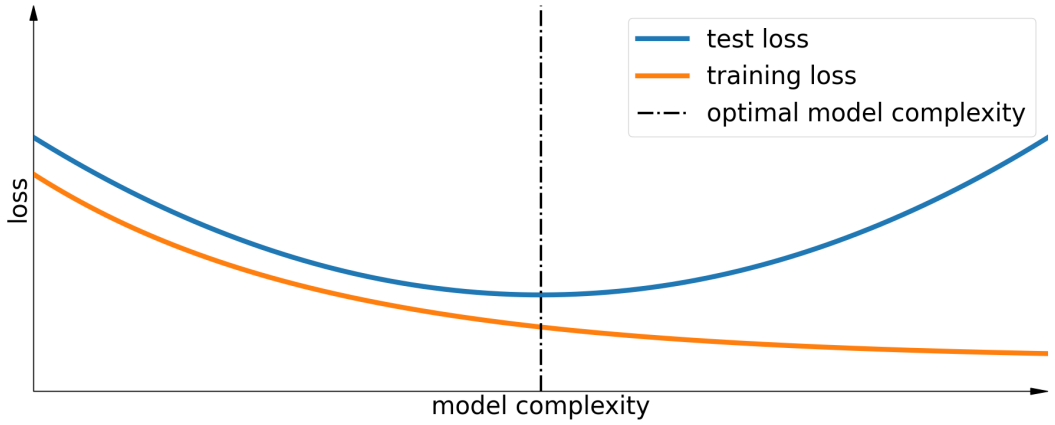

Figure 4.5: Illustration of underfitting (low model complexity) and overfitting (high model complexity).

avoid underfitting, the model complexity can simply be increased by increasing the number of layers and/or the number of units in the layers. To counter overfitting, it also seems reasonable to analogously reduce the complexity of the model. Nevertheless, it is often better to keep a relatively high complexity but instead use some kind of *regularization* [21]. Some of the most popular regularization techniques will be outlined briefly:

- **L2/L1 Parameter Regularization (also known as *weight decay*):** The L1 or L2 norm penalty will be added to the loss function. Thus, weight values will decrease towards zero in order to reduce the total number of non-zero parameters in the network. The difference between both methods is, that weights can actually reach zero with L1, while they only approach it with L2.

- **Dropout:** Some neurons will randomly be deactivated during each training step based on a pre-defined probability. This incites the networks to generalize better, since it does not have the full input information. Visualization of dropout applied to an examplary MLP can be seen in Figure 4.6.

- **Batch Normalization:** As the name suggests, the input mini-batches will be normalized before they will be fed into a layer. This not only can be used for regularization, but it also allows for higher learning rates and a more loose initialization [25].

- **Early stopping:** Another approach is to monitor the model performance on a validation dataset after each epoch. After each improvement a checkpoint of the model will be saved. When the loss stagnates or increases after several epochs, training will be stopped.

- **Augmentation:** This method applies different operations on the input in order to artificially create new data and therefore increase the training dataset. This is often used in computer vision, where for example input images will be distorted, cropped, expanded, flipped or rotated.

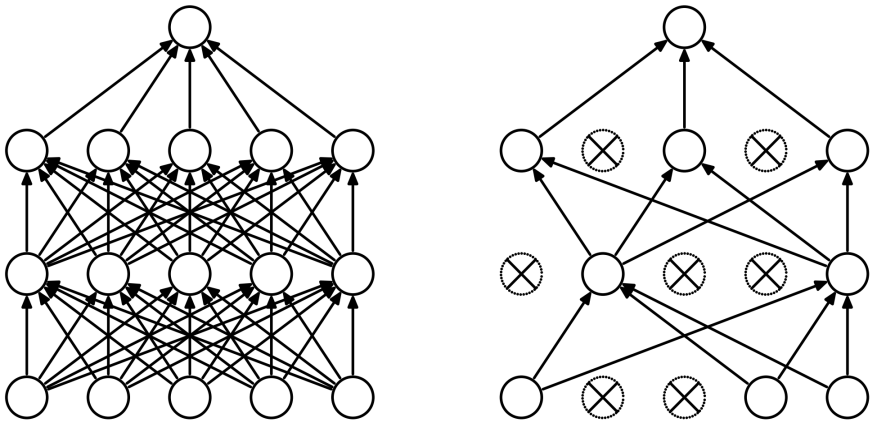

Figure 4.6: **Left:** A standard neural net with 2 hidden layers. **Right:** The same network with dropout applied. Crossed units have been dropped [26].

4.3.3 Learning rate

To outline the importance of the learning rate, consider a simple loss function $J(w)$. As explained, the goal of SGD (stochastic gradient-descent) is to update the weights according to the gradient in order to reach the global minimum of $J(w)$. With a too large learning rate (Figure 4.7 left) it is impossible for SGD to get deep enough into the minimum and it will only oscillate around it. On the other hand, a small learning may result in a lot of iterations and therefore a large training time until convergence. A solution to this problem could be to

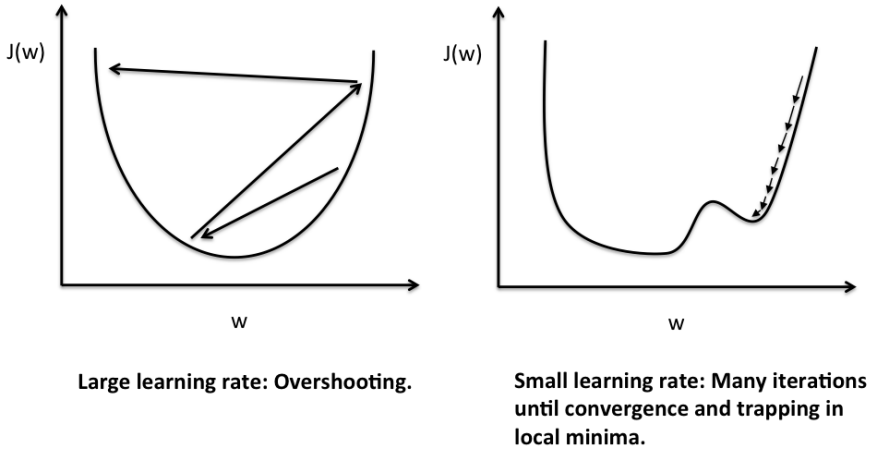

Figure 4.7: Impact of learning rate on a network with the loss function $J(w)$ [29].

decay the learning rate while training. For example, this can be a step-wise or exponential decay. It is also possible to monitor the loss of the model and decrease the learning rate only when this loss stagnates for a few epochs. Nevertheless, there is still the risk of a convergence to suboptimal local minima or saddle points, since the gradient also disappears in such regions (Figure 4.7 right).

Another way, which addresses both problems, is to introduce a smoothing parameter $\beta \in (0, 1)$ into the gradient-descent method. The weight updates described in section 4.1 can then be written as

$$\overline{W} \Leftarrow \overline{W} + \overline{V} \qquad \text{with } \overline{V} \Leftarrow \beta\overline{V} - \alpha\frac{\partial L}{\partial \overline{W}} \tag{4.8}$$

For $\beta = 0$ this update is identical to equation 4.4.

With this additional parameter earlier gradients will also be taken into account for the next update. This enables the updates to move in an "averaged" direction rather than in excessive "zigzags" [21]. This effect is illustrated in Figure 4.8. A ball rolling down a hill can be seen as an analogy. The ball will initially pick up momentum and can pass flat surfaces with ease. Similarly, β gives gradient-descent the ability to move quickly through flat regions of the loss surface and reduces

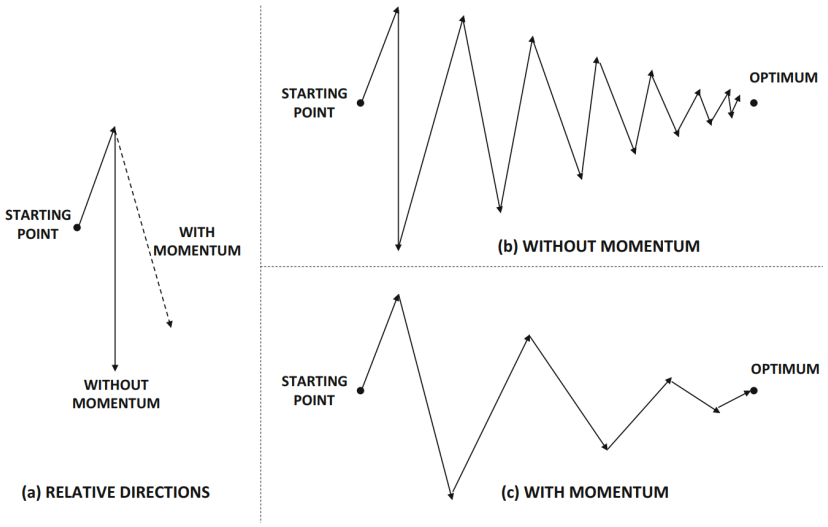

Figure 4.8: Smoothing zigzag weight updates with momentum [21].

the probabiltiy to get stuck in local minima. Therefore, β is also referred to as a *momentum*. Figure 4.9 shows the impact of momentum on the gradient-descent algorithm.

Further improvement can be made, when pairing the momentum-based SGD with a learning rate schedule. As mentioned above, this can be a simple step-wise or exponential decay but in recent years cyclical learning rate (CLR) schedules have shown much better results [27, 28]. The first part of such a cycle consists of increasing the learning rate to a maximum value, the second part decreases it again to the initial value. Using large learning rates seems counterintuitive, but it was shown that they both help in acting as regularizer as well as speed up training significantly [27, 28].

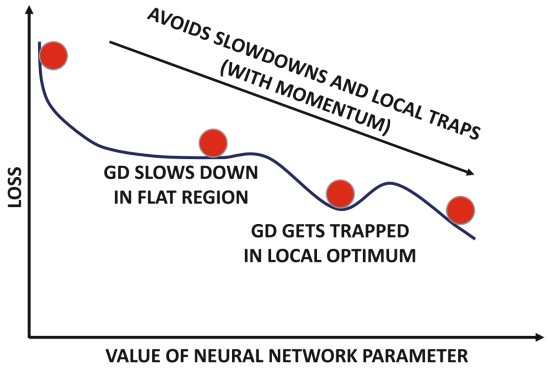

Figure 4.9: Navigation on complex loss surfaces with momentum. *GD* refers to standard gradient-descent without momentum. The optimization will be accelerated in flat regions of the loss surface and local minima will be avoided [21].

5 Convolutional neural networks

Since the multilayer perceptron generally shows great performance in pattern recognition, one would expect similar results when applied to images. Still, MLPs are typically not used in computer vision, which is mainly due to the computational inefficiency. For example, an image with the dimensions $200 \times 200 \times 3$ (3-channel rgb image) will result in an MLP with 120,000 weights in just one layer. Considering that image dimensions easily can be much higher and that there can be several layers in the MLP, the number of weights will be in the order of millions. This not only leads to high training time, but also can cause overfitting as the models complexity is very high [21].

Today, for this task mostly ***C**onvolutional **N**eural **N**etworks* (CNN or ConvNet) come into use. These kind of networks are build from a combination of *convolutional layers* and *pooling layers*, which are used for feature extraction and downsampling respectively. Before describing CNN architectures in general, first convolutions and pooling will be explained.

5.1 Convolutions

Convolutional layers use multiple *filters* or *kernels* on the input image in order to extract its features. These filters then "slide" over the input image and the dot product of the kernel with its underlying matrix will be calculated. Typical kernel sizes are 3×3 or 5×5, while the *stride* is 1. When referring to strides, the step size of the kernel to the right or bottom after each convolution is meant. These operations are illustrated in Figure 5.1. Note, that the image will be downsampled in this example from 4×4 pixels to 2×2 pixels. Often, this is undesirable, since some information may be lost at the edges of the image. To circumvent this problem, *padding* typically is introduced, which increases the size of the input array. For example, the dimensions of the source image in Figure 5.1 will increase from 4×4 to 6×6. The padded rows and columns will contain zeros. Now applying the filter to the padded image, the resulting image will have the same dimensions as the original input image. The kernel in this example is used for vertical edge detection. Table 1 shows some examples of different filter matrices, to get a better understanding of how filters work. These four kernels as well as the vertical edge detection kernel from Figure 5.1 were applied to a sample image, which can be seen in Figure 5.2.

5.2 Pooling

To downsample the input images in order to decrease the overall training time and resource usage, pooling layers are commonly used in CNNs. The two pooling types generally used are *max pooling* and *average pooling*. Analogous to convolutions, a

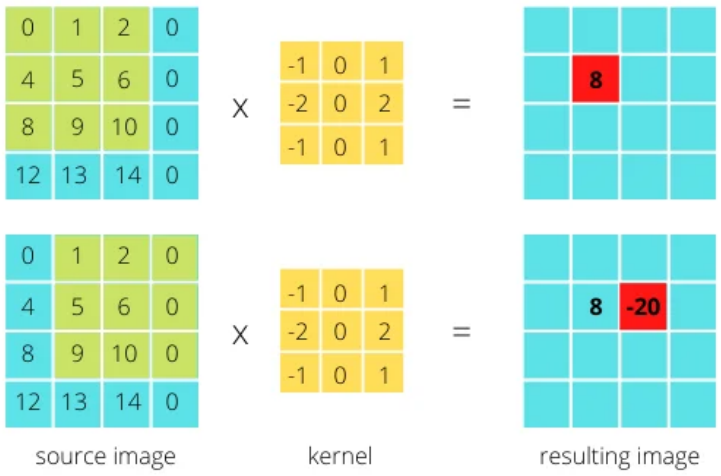

Figure 5.1: Example of applying a convolutional filter on an image array. The dot product of the source image with the kernel matrix will be computed to create the resulting image (2×2). Only the first (**top**) and second (**bottom**) dot products are illustrated. Here, the convolution is performed with a stride of 1 and no padding [31].

kernel also slides over the input image, but instead of computing the dot product, the maximum or average of the underlying matrix will be calculated.

An example of both the max pooling as well as the average pooling are depicted in Figure 5.3. Here, a pooling kernel of size 2×2 slides over the input image. Note, that different to convolutional kernels, which mostly use strides of size 1, pooling layers typically use strides equal to the kernel size. Hence, the example image will be downsampled from 4×4 pixels to 2×2 pixels, resulting in an output with a quarter of the number of input pixels. Depending on the input and kernel dimensions, padding may also be used in pooling layers.

5.3 CNN architecture

As described in section 5.1, filters will be applied to input images of dimension $L \times B$, resulting in an output image, which is also referred to as a *feature map* or *activation map*, of the same size. Multiple feature maps created by a number of d filters will make up one 3D convolutional layer. On each of the $L \times B \times d$ values, also an activation function is applied. Here, the ReLU function has shown to be superior to saturating functions like tanh or sigmoid with regard to accuracy and speed [41]. Max or average pooling may then be performed on the convolutional layer, thus downsampling the size of the feature maps but keeping the same

sharpening	blur
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 0.06 & 0.13 & 0.06 \\ 0.13 & 0.25 & 0.13 \\ 0.06 & 0.13 & 0.06 \end{bmatrix}$
sepia	**horizontal edge detection**
$\begin{bmatrix} 0.27 & 0.53 & 0.13 \\ 0.35 & 0.69 & 0.17 \\ 0.39 & 0.77 & 0.19 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Table 1: Examples of convolutional filters.

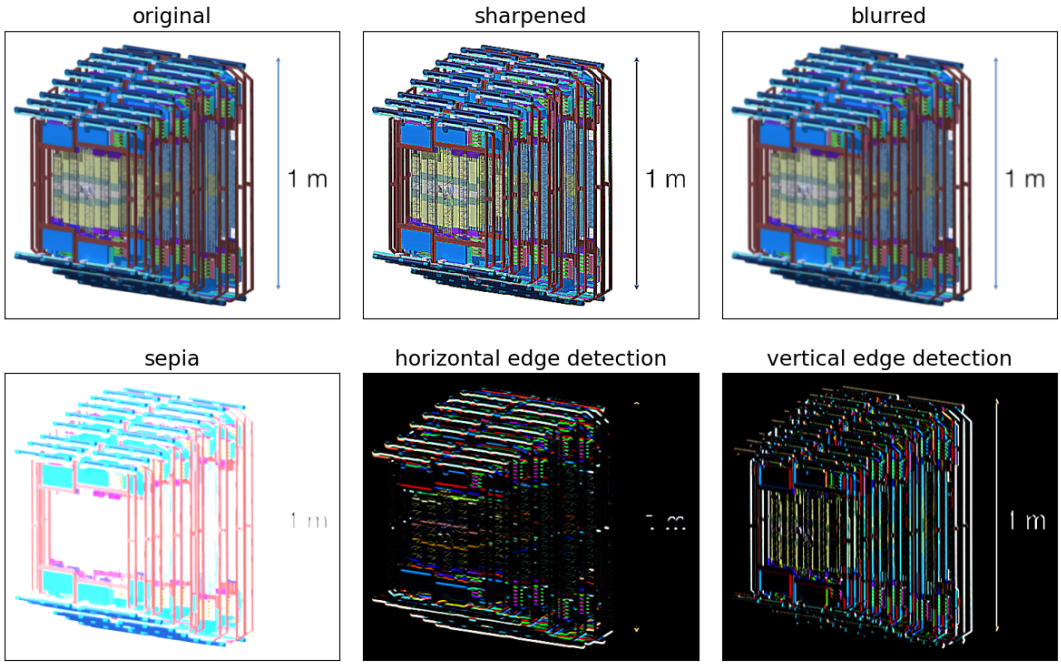

Figure 5.2: Different filters applied to the schematic of the CBM STS [13].

depth d. In deep CNNs there can be several convolutional layers before pooling is applied, since pooling can only be done a certain number of times before reducing the output size to $1 \times 1 \times d$.

The last pooling layer will often be flattened, meaning the 3D pooling layer will be reshaped to one dimension. Fully connected layers (also called *dense layers*) may follow along with the output layer. An example of a simple CNN architecture is depicted in Figure 5.4. It shows one convolutional layer and one pooling layer followed by several fully connected layers and finally the output.

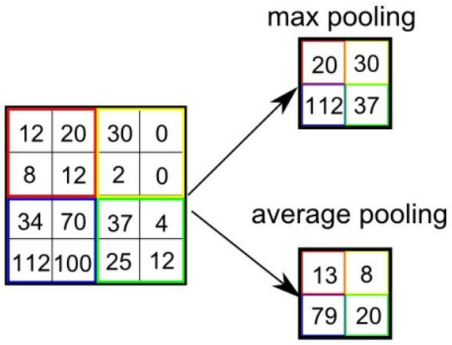

Figure 5.3: Max and average pooling operations with a kernel and stride of size 2×2 [32].

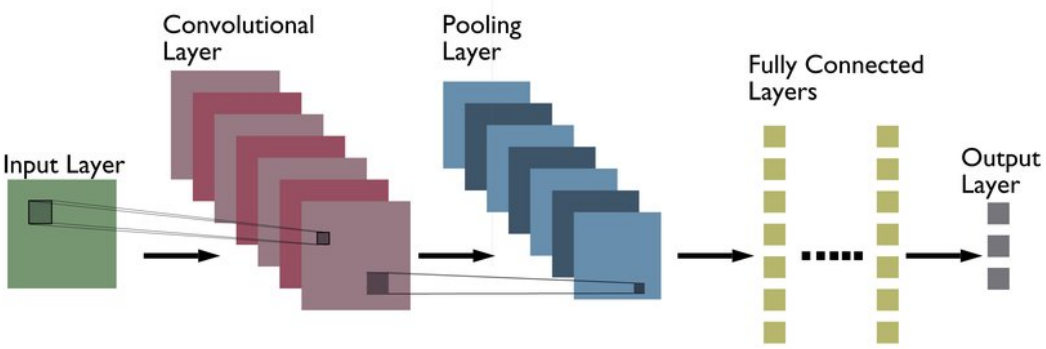

Figure 5.4: Example of a CNN architecture [33].

Commonly, CNNs contain more than one convolutional layer. In the first layers simple features, like edges and corners will be extracted, while in deeper layers more complex shapes, e.g. digits or even human faces, will be captured by composition of these primitive shapes. Therefore, low-level layers do not need as many filters as high-level layers. The big advantage of CNNs over MLPs is, that the spatial relationship will be preserved over multiple convolutional layers and therefore, the location of features on images is irrelevant. Also, because the activation of a layer only depends on a limited spatial region of the preceding layer, the connections in CNNs are considerably more sparse that in a fully connected MLP. This, in addition to the performed downsampling, can have a vast impact on memory usage and training time.

5.4 Object localization/detection

CNNs are commonly used for object localization or detection. Object localization consists of two steps. First, the object in an image needs to be classified and then a bounding box will be drawn around the object to locate it [21] (Fig. 5.5). The localization part can be handled as a regression problem with multiple targets [21],

Figure 5.5: Object detection performed on an image [34].

since the bounding box can be expressed with four values: x and y coordinate (e.g. of the top left corner), width and height [21]. The backbone of a localization network can be a classifier trained from scratch or one of the publicly available networks, which were pretrained on the *ImageNet* [35] dataset. As the convolutional layers of such a network are trained only for feature extraction they can be used for both classification and regression, while their weights stay fixed [21]. Only the *head* — formed by the last few fully connected layers and the output layer — is removed and replaced by the new classification and regression heads [21]. The full network can then be trained with the images as inputs and the classes and bounding boxes as outputs via backpropagation. The described architecture for a simple network that locates a single object in an image is shown in Figure 5.6.

Object detection is localization of a variable number of objects of different classes on an image [21]. An example of object detection performed on an image is shown in Figure 5.5. Since the number of objects on the image is unknown, the number of classification/regression heads is also unknown, which is why the same architecture from the localization network cannot be used here. A naive approach is to use the sliding window method, where the localization network for single objects will be applied for all possible bounding boxes in the image [21]. This is very resource consuming and far from being viable for real-time object detection.

A possible solution are *region proposal methods*, which generate bounding

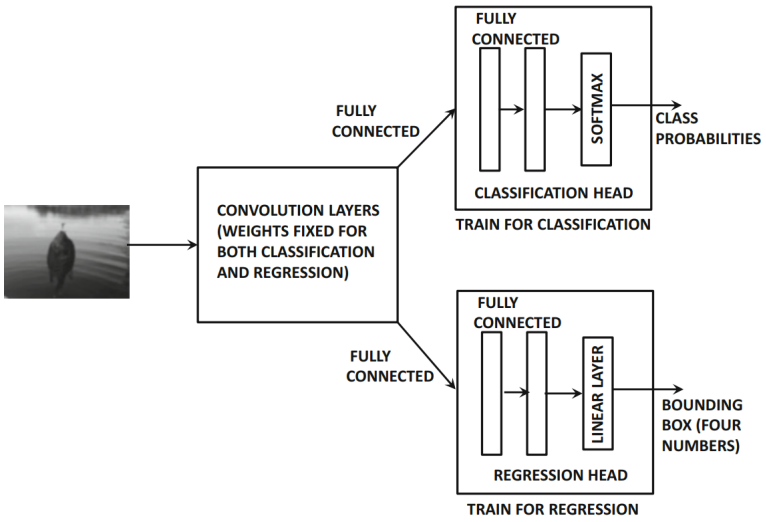

Figure 5.6: Basic concept of object localization on an image with a single object [21].

box candidates on which the object localization can be performed [21]. Some examples of these two-step object detectors are *Faster R-CNN* [37, 36] and the *Feature Pyramid Network* (FPN) [38, 36]. There are also one-step approaches like *YOLO* [39, 36] or *SSD* [40, 36], which are based on global regression and classification [36]. They map directly from image pixels to class probabilities and bounding boxes, which can bring down time expenditure [36]. Overall, there are quite a lot of different object detector designs, too many to be explained in detail.

6 Ring reconstruction in the mRICH detector plane

6.1 Introduction

The goal of this thesis was to use a machine learning approach in order to extract the parameters of Cherenkov rings in the mRICH photodetector plane (see section 2.2.1). An algorithm for event reconstruction in the (m)RICH detector is already implemented in *CbmRoot* [42], which is the framework for data analyis, simulation and reconstruction for (m)CBM. Below, this algorithm will be outlined briefly. After that, the toymodel used for training the networks will be introduced. Lastly, the two main networks that were created, will be presented and compared to the currently used ring reconstruction algorithm.

The software mainly used for creating the toymodel, neural networks, visualization etc. is the PYTHON [44] programming language together with the TENSORFLOW [45] and NUMPY [46] packages for Python.

6.1.1 Event reconstruction with CbmRoot

The event reconstruction in the (m)RICH detector with CbmRoot consists of the three parts depicted in Figure 6.1. First the particle tracks will be extrapolated

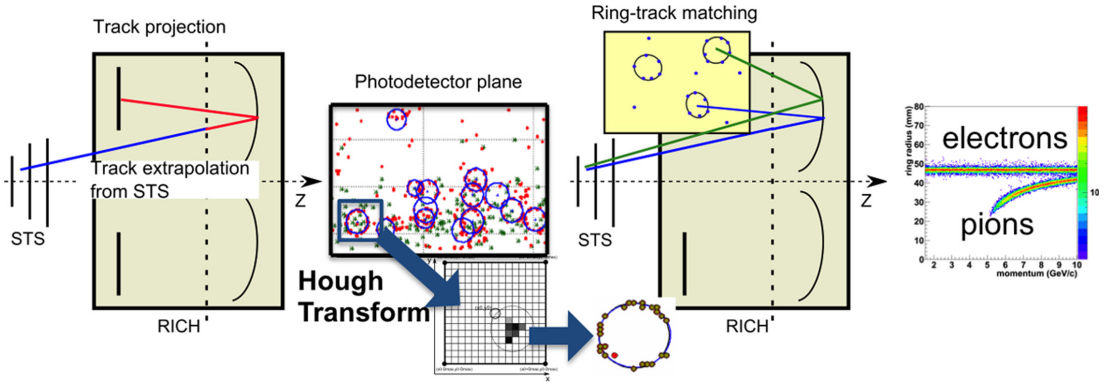

Figure 6.1: Event reconstruction in the RICH detector [48].

from STS and projected onto the photodetector plane. For the CBM RICH the extrapolation includes the mirrors, unlike for the mRICH where the tracks can be extrapolated directly onto the photodetector plane (see section 3.2 for the design differences).

The next step is to find rings in the detector plane with an algorithm based on the *Hough Transform* [49]. As three hits can define a ring, this method computes the ring parameters (center and radius) of the corresponding hit triplets. As this process can be very slow, only hit triplets with a radius below the maximum radius R_{max}, which depends on the RICH geometry, will be used. The found ring

parameters will be filled into a two dimensional histogram, in which the actual parameters of the rings are indicated by strong peaks. An additional ring or ellipse fitting algorithm is used to further refine these parameters.

The third and last step is to match the reconstructed rings to the extrapolated STS tracks. A last quality check based on an artificial neural network serves to distinguish primary electrons from secondary electrons and pions.

There is also an "ideal" version of this algorithm implemented in CbmRoot, which can use additional information from the Monte Carlo simulations like the parent of a particle or what particle created a specific hit in the photodetector plane and so on.

6.1.2 Simulations and denoising

Simulations of particle collisions and interactions of particles with matter are a very powerful tool in subatomic physics. They not only help with improving detector geometries but also serve to improve the data analysis. The simulations utilized by CBM/mCBM and in this work are based on *UrQMD* [51] and *Geant* [43].

UrQMD (**U**ltra **r**elativistic **Q**uantum **M**olecular **D**ynamics) is a framework, which is able to simulate proton-proton, proton-nucleus and nucleus-nucleus interactions with the Monte Carlo method [51]. Charge exchange, four-momentum conservation as well as strangeness are taken into account. The model incorporates 32 meson and 55 baryon types, ground state particles and resonances up to masses of 2.25 GeV [51]. Furthermore, particle-antiparticle symmetries and isospin symmetries are included.

Geant (**Ge**ometry **a**nd **t**racking) is a simulation toolkit for tracking particles through matter by utilizing Monte Carlo methods. The geometry of the experiment with all detectors and support structures is taken into account together with the effects of particles interacting with those materials, e.g. energy loss. A large palette of materials, elements and particles with energies ranging from a few eV up to the order of TeV, as well as electromagnetic and hadronic processes are included in Geant [43].

The particles created by an UrQMD-simulated collision can now be tracked by Geant through the geometry of the CBM/mCBM detector setup. The created Cherenkov photons generated by the passage of these particles through the radiator finally hit the photodetector plane to simulate an *event display* that represents a model of a real particle collision at CBM/mCBM.

As stated in sections 2.2.1 and 3.2 the photodetector plane of the mRICH is built by 9×4 MAPMTs while each MAPMT is made up by 8×8 pixels, resulting in a 72×32 pixel photodetector plane. This photodetector plane with exemplary, simulated events can be seen in the top row of Figure 6.2. Note, that when looking at the schematic view of the MAPMTs (Fig. 3.3), it can be observed that the pixel sizes slightly vary in corners. Additionally, the edges of the MAPMTs contain thin

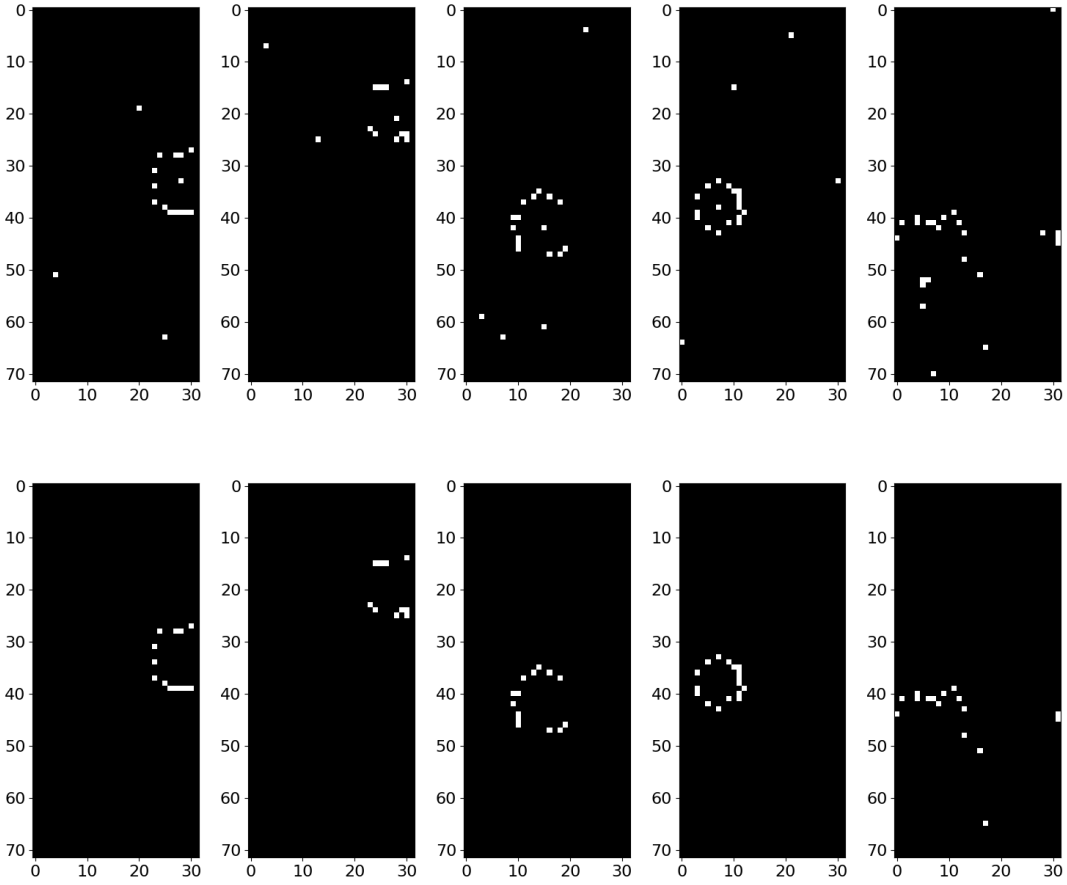

Figure 6.2: mRICH photodetector plane with hits from UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy; **top:** raw UrQMD events, **bottom:** same events with ideal denoising applied [50].

areas without pixels. Hence, the effective light-sensitive area of $48.5 \times 48.5\,\mathrm{mm}^2$ is smaller than the full area of $52 \times 52\,\mathrm{mm}^2$. Even though, the representation of the detector geometry with 72×32 pixels is therefore not exactly accurate, it will still be used throughout the rest of this thesis due to its simplicity.

The Cherenkov rings can be observed, but there is also some noise visible which cannot be assigned to any ring. The source of that noise could be charged particles crossing the MAPMTs or scattered Cherenkov photons. At the time of writing this thesis, a *denoiser* [50] also based on CNNs is being established in order to remove that noise.

For simulations an "ideal" denoiser can be implemented by removing all hits, i.e. "firing" pixels, originating from other sources than Cherenkov photons. Results from ideal denoising can be seen in the bottom row of Figure 6.2. This denoiser does not take the scattering into consideration, which is why poor rings can still occur (see right most event in Figure 6.2).

6.1.3 Toymodel

In order to train a convolutional neural network for ring reconstruction, a dataset which pairs images (event displays) to the target values (ring parameters) was needed. It was decided to create a toymodel that reflects real or simulated event displays. The advantage of using a toymodel is, that it can be easily modified and is able to produce a large enough dataset to avoid overfitting. The March 2020 mCBM beamtime campaign was used to help design this toymodel. Data analysis of run 831 of this beamtime shows that reconstructed rings have ring radii of roughly 1.8 cm-6 cm and about 8-30 number of hits [13]. When assuming a uniform pixel size of 6 mm (see Fig. 3.3) as a rough approximation, the radius range can be translated to 3-10 pixels. The algorithm for creating the images and

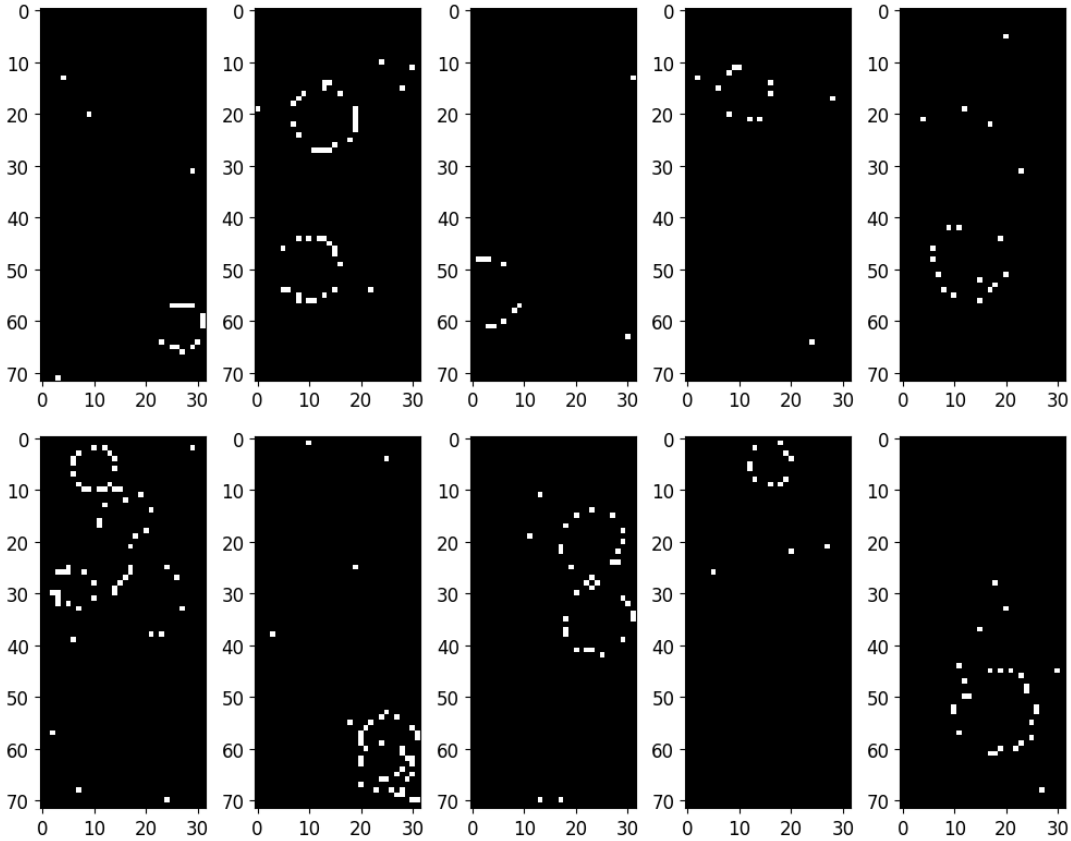

Figure 6.3: Sample event displays created by the toymodel.

data pairs will be outlined shortly.

1. **Initialize image:** Create an array of shape $72 \times 32 \times 1$ with zeros for all entries. The third value denotes the number of channels. Pixels of the photodetector plane either detect a photon or not. Generally, coloured

images have three channels, but here only one channel is needed to describe the state of the pixels (on or off).

2. **Initialize rings and parameter array:** Create multiple points on a circle with radius $r = 1$ pixel. The number of points will be randomly picked from the range (8,30). These points were created with some noise, i.e. they were moved in some direction and thus no longer form a perfect circle, based on a predefined noise value.

   Since, some rings in the RICH/mRICH have rather elliptical shape, the parameters will be stored in ellipse format:

   $$\begin{bmatrix} x & y & a & b & \theta \end{bmatrix}$$

   With

   - x: x-coordinate of ellipse center
   - y: y-coordinate of ellipse center
   - a: semi-major axis
   - b: semi-minor axis
   - θ: angle

   This allows to easily add ellipse creation in the future. Still, only perfect rings are used for now, i.e. $a = r$, $b = r$ and $\theta = 0$. Hence, a single ring is saved in the format

   $$\begin{bmatrix} x & y & r & r & 0 \end{bmatrix}$$

   A single event display can contain up to 4 rings [13]. Therefore, an array of the size 5×4 is initialized with zeros to be later filled with ring parameters.

3. **Scale and move rings:** Pick a number randomly from the range (3,10) to set the radius of the rings and move them to a random location in the range of the image shape, i.e. $x \in (0, 32)$ and $y \in (0, 72)$.

4. **Change pixel value:** Change the values of the array that correspond to point coordinates of the rings from 0 to 1.

5. **Save ring parameters:** Write the 5 ring parameters into the next row of the initialized parameter array.

6. **Repeat steps 2-5:** Depending on a randomly picked number of rings per event steps 2-5 will be repeated up to 3 more times.

7. **Add noise hits:** Even though, the neural network is designed to predict on denoised data, this data can still contain noise from scattered Cherenkov photons. Thus, up to 5 noise hits are added randomly on the image array.

As an example, this process will yield the following parameter array for an event with two rings.

$$\begin{bmatrix} x_0 & y_0 & r_0 & r_0 & 0 \\ x_1 & y_1 & r_1 & r_1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

To create a complete dataset, this algorithm can be performed multiple times. Some sample images created by the toymodel can be seen in Figure 6.3 and the same images with the ring fits of the corresponding parameter arrays in Figure 6.4

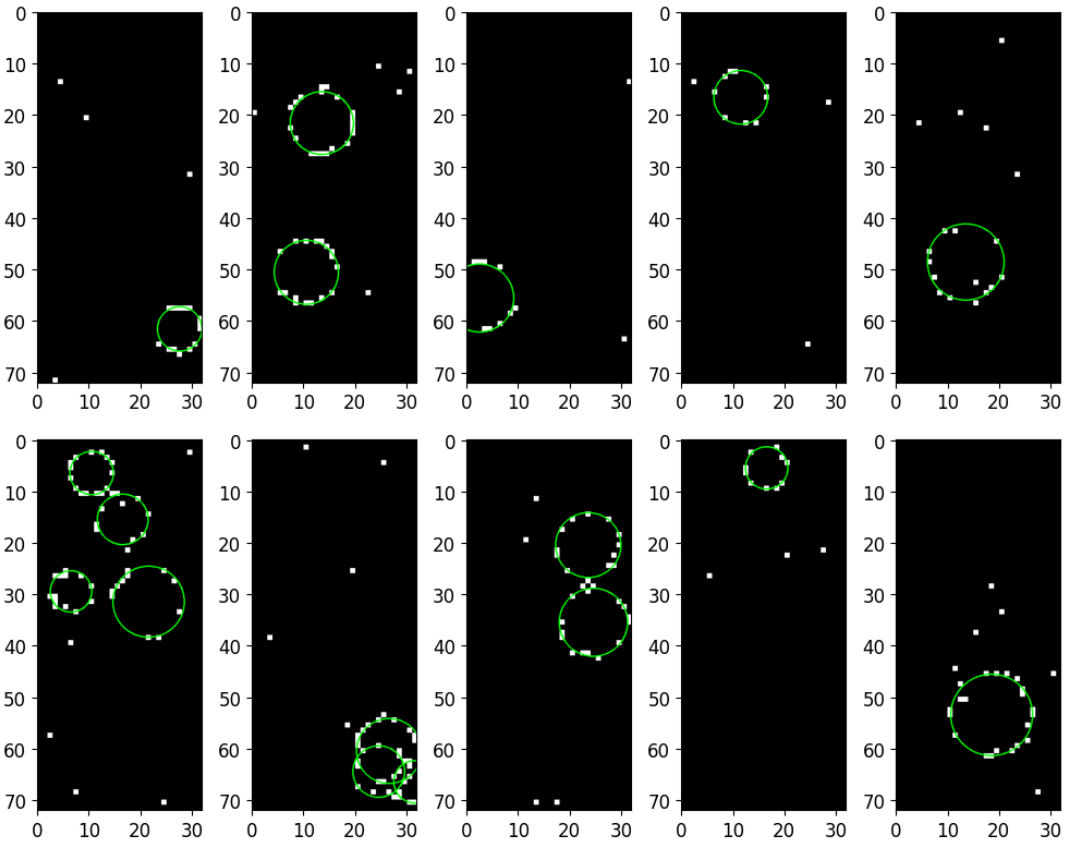

Figure 6.4: Trainings data pairs created by the toymodel for training the neural network. Input data is the images and target data s the ring parameters, which are already fitted here.

6.2 Model 1

6.2.1 Architecture and training parameters

The first approach was to treat the ring reconstruction as a regression problem with multiple targets. Therefore, a CNN was created, which maps the input images to the $5 \times 4 = 20$ ring parameters as outputs. This models architecture is shown in Figure 6.5. First, two consecutive convolutional layers were applied on the input with 64 filters of size 3×3. Max pooling was applied on the second layer, reducing the dimensions from $72 \times 32 \times 64$ to $36 \times 18 \times 64$. Zero or "same" padding was used in both convolutional and pooling layers. These three layer operation were applied three more times, which results in a layer of size $5 \times 2 \times 512$. This layer was then flattened to form a dense layer of size 3072. Another dense layer of size 64 was added followed by the last layer of size 20, which was reshaped to 5×4 to have the same dimensions as the parameter arrays. The ReLU function was used as activation for every computational layer. Additionally, batch normalization was applied on every layer input to prevent overfitting (see section 4.3.2).

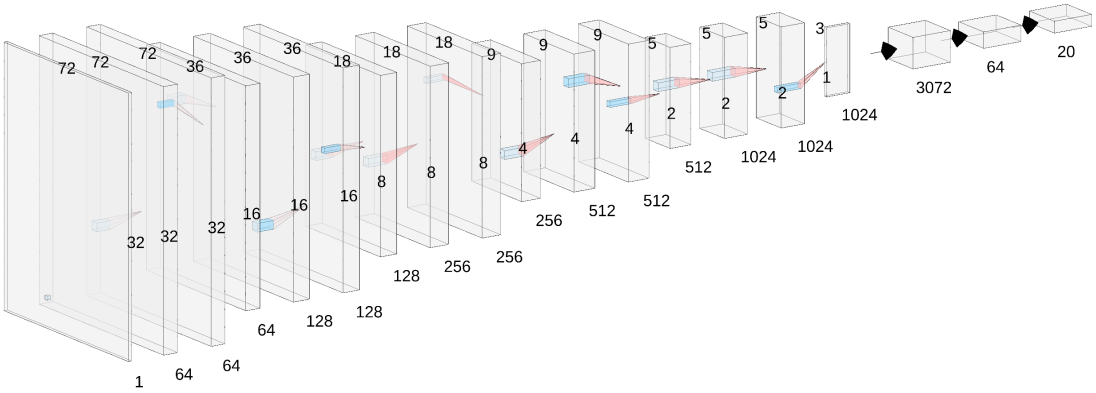

Figure 6.5: Architecture of the CNN model. The depth indicates the number of filters/feature maps in that layer.

SGD was used to optimize the loss function, which was chosen to be the mean squared error (MSE). Data was fed to the network in batches of the size 200. In order to speed up training, the "1cycle" policy introduced by Leslie N. Smith [28] was used to modify the learning rate as well as the momentum during training.

This policy increases the learning rate from the initial to the maximum value, which is 10 or 20 times [28, 52] the initial value, in the first part of the cycle. In the second part the learning rate decreases to a value that is a few orders of magnitude lower than the initial learning rate. The momentum is changed in similar fashion, with the differences, that it decreases in the first part and increases in the second part and the final momentum is equal to the initial momentum. To find the optimal learning rate, the "LR range test" proposed by Smith [27] was performed to determine the maximum learning rate of 0.1. Both schedules used

for training the CNN can be seen in Figure 6.6. The depicted schedules are based on FASTAIs [53] implementation of the "1cycle" policy.

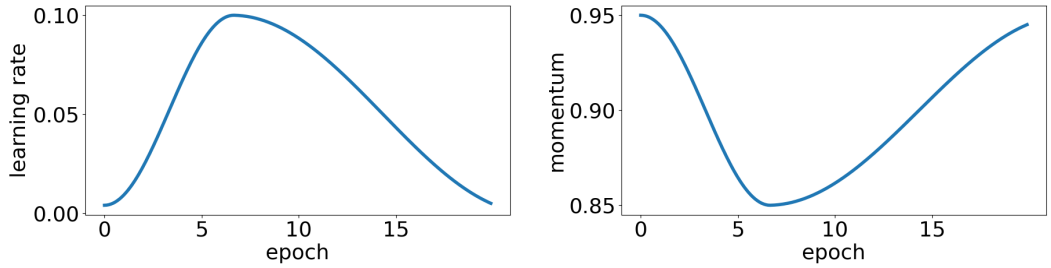

Figure 6.6: "1cycle" policy used for training the CNN. **left:** learning rate schedule with initial, maximum and final values of 0.004, 0.1 and 10^{-5} respectively; **right:** momentum schedule with initial value of 0.95 and minimum value of 0.85; the first part of each cycle is shorter than the second part as it only takes up 30% of the number of total steps.

6.2.2 Evaluation

Training was done on a dataset of the size 200,000 for a total of 20 epochs. Loss and accuracy for both training and validation set were recorded and drawn in Figure 6.7.

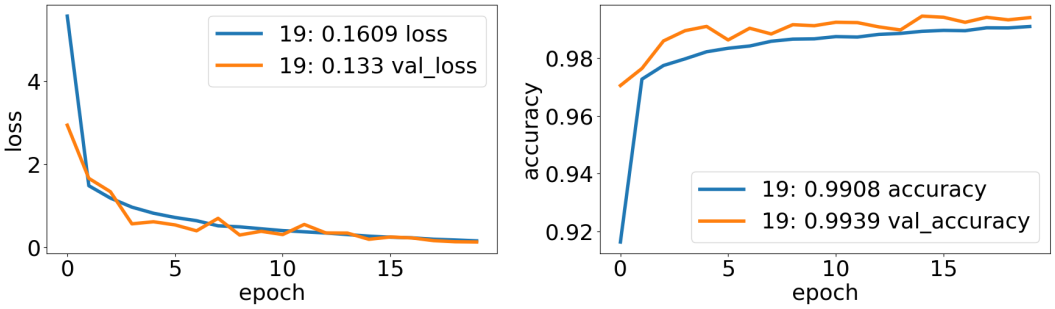

Figure 6.7: Loss and accuracy for training (**blue**) and validation (**orange**).

For further evaluation, the model was tested on a UrQMD-simulated dataset for Ag+Ag-collisions at 1.24 AGeV beam energy, which was denoised by the ideal denoiser described in section 6.1.2. Some cuts were applied on this simulation data to filter out some events where the ideal Hough Transform Method (HTM) extracted poor ring parameters. From 10,000 events only 5806 events were kept. A few examples of those events with ring fits with parameters extracted from the CNN model can be seen in Figure 6.8. Visual comparisons of the CNN model

with both, the ideal HTM and the standard HTM can be seen in Figures 6.10 and 6.11. The average prediction time was about 435 µs per event. The errors of

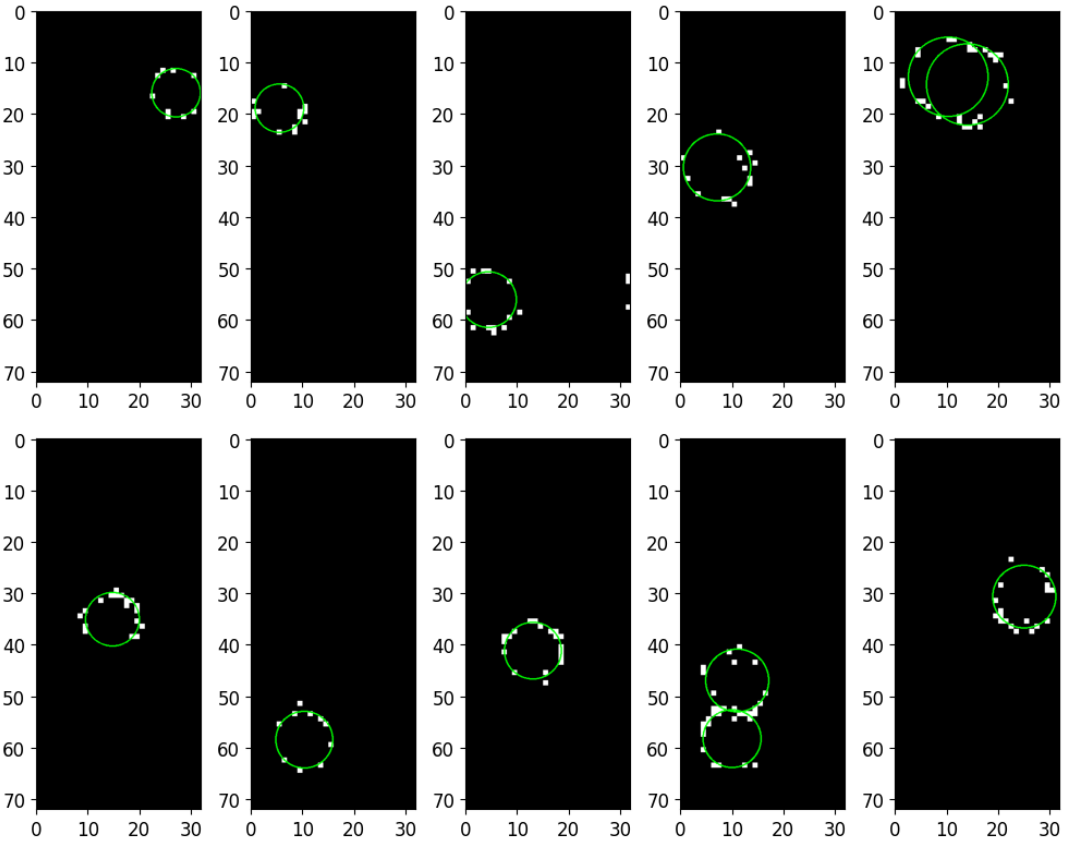

Figure 6.8: Ring fitting with parameters extracted from the CNN model. A dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy was used.

ring parameters of both CNN and standard HTM were calculated with respect to the ideal HTM. For this, the mean squared error (MSE) was used.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i}^{N}(p - \hat{p})^2$$

Where N equals 5806, p is the reference parameter yielded from the ideal HTM and $\hat{p}$ is the predicted parameter from either the CNN or the standard HTM. These errors for each parameter x, y and r can be seen in Table 2.

Additionally, the number of rings per event were compared for all three methods, which is depicted in Figure 6.16. It can be seen that the standard HTM primarily fits a single ring per event, while the number of two and three rings have more weight in the CNN and the ideal HTM.

	x (pixel)	y (pixel)	r (pixel)
$\sqrt{MSE_{CNN-idealHTM}}$	3.11	7.31	1.13
$\sqrt{MSE_{HTM-idealHTM}}$	3.29	7.36	1.13

Table 2: Errors of the regression CNN and the standard HTM with respect to the ideal HTM.

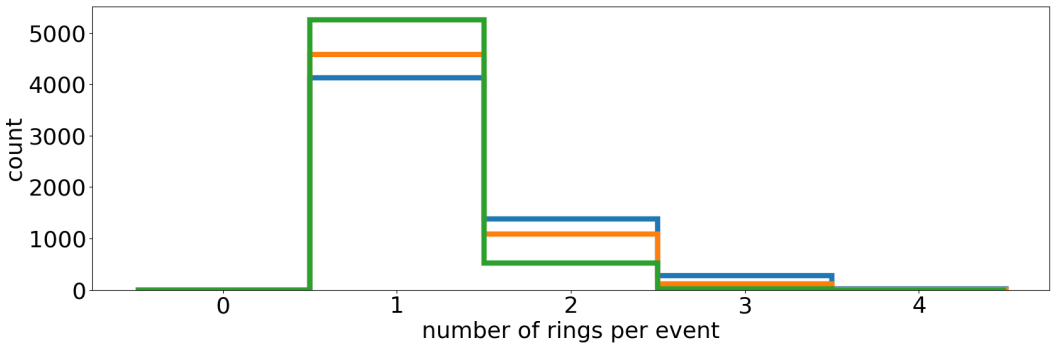

Figure 6.9: Number of rings per event for the regression CNN model (**blue**), standard HTM (**green**) and ideal HTM (**orange**).

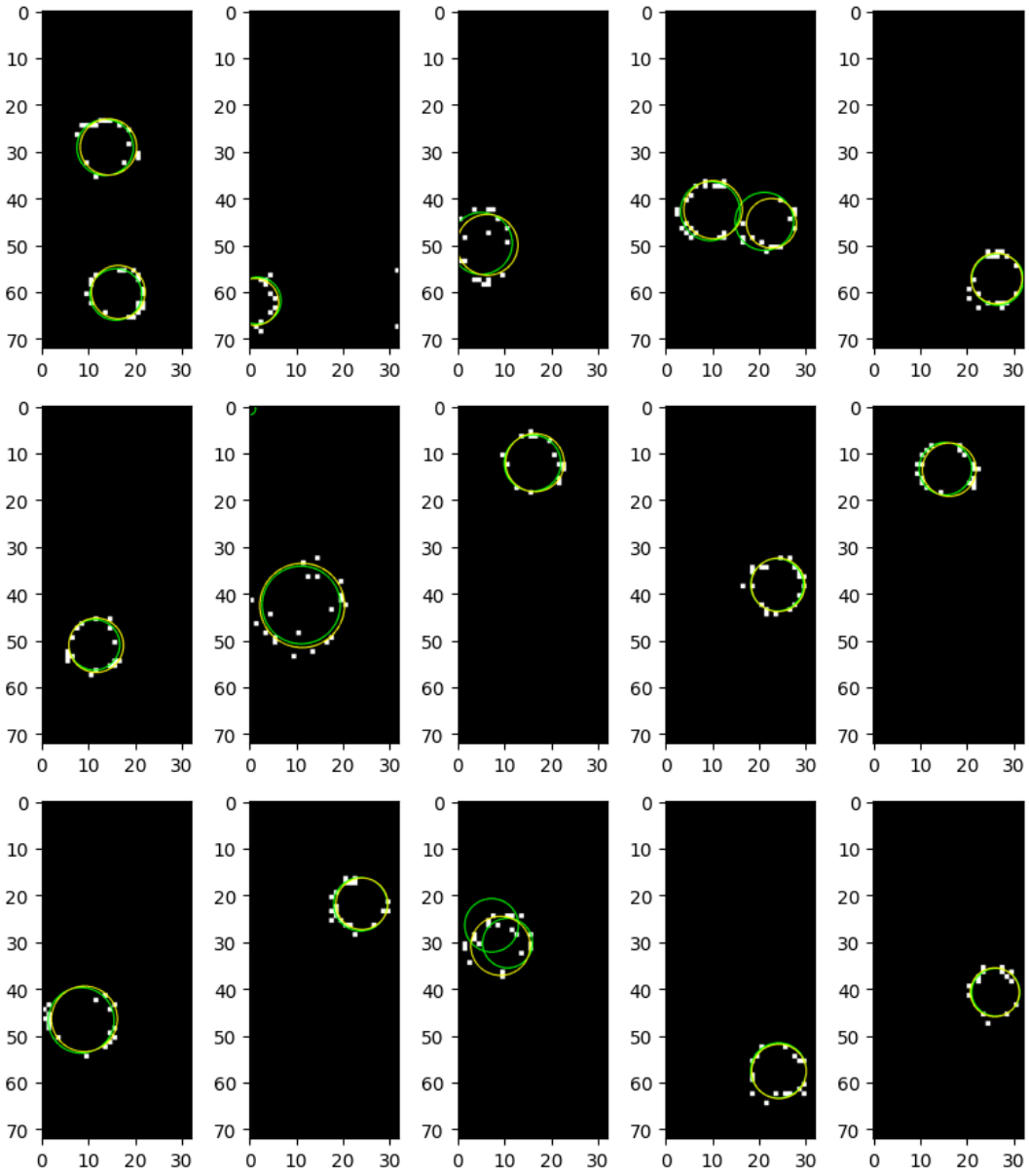

Figure 6.10: Ring regression CNN (**green**) vs. ideal HTM (**yellow**) on a dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy.

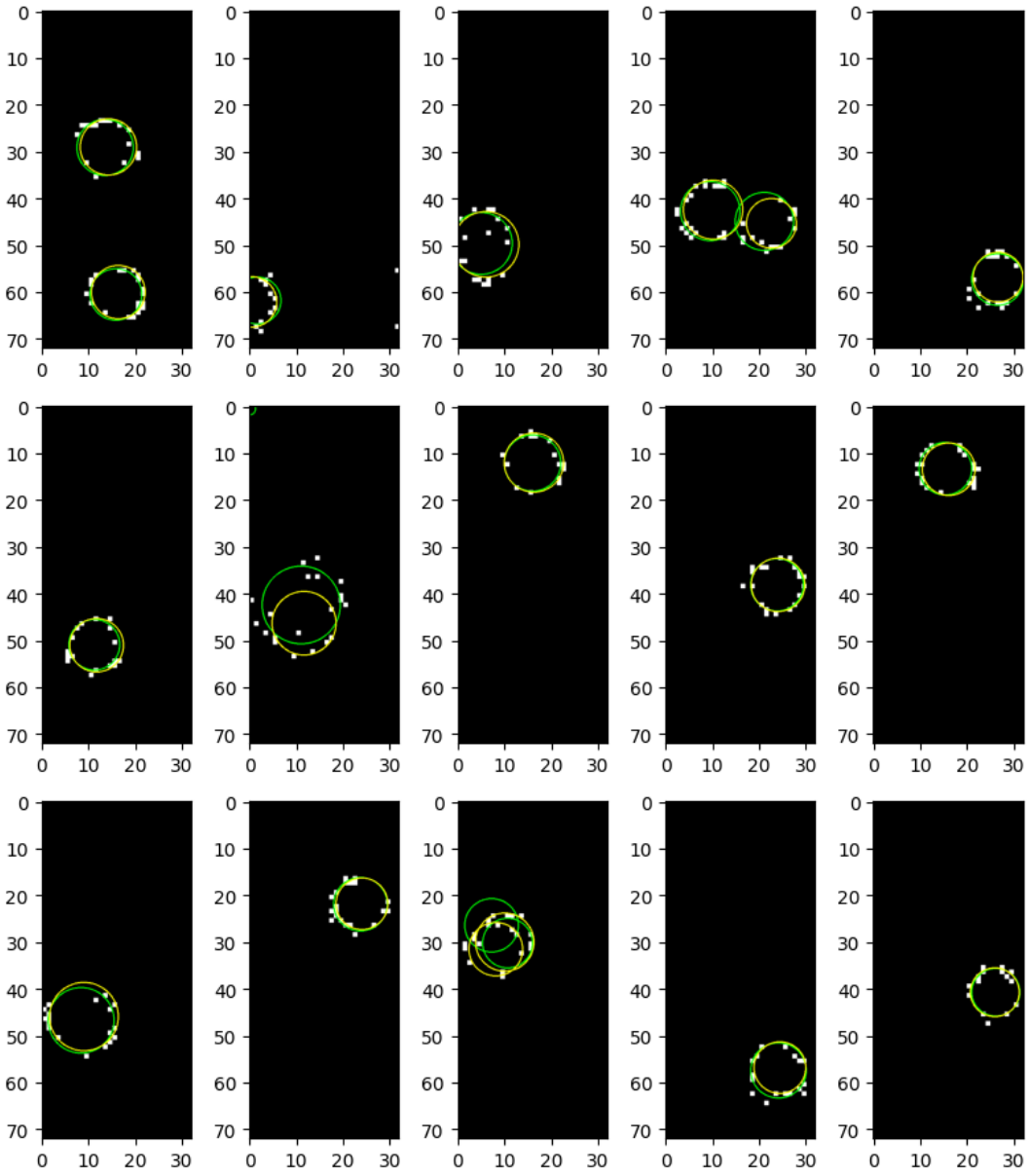

Figure 6.11: Ring regression CNN (**green**) vs. HTM (**yellow**) on a dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy.

6.3 Model 2

Another approach for the ring reconstruction was to use an object detector as explained in section 5.4 and then run a ring regressor, similar to the one used in section 6.2, on the extracted bounding boxes. This procedure is depicted for an example event display in Figure 6.12. This has the advantage that a maximum number of rings does not need to be defined when constructing the model, which is the case for the first CNN model (see section 6.2).

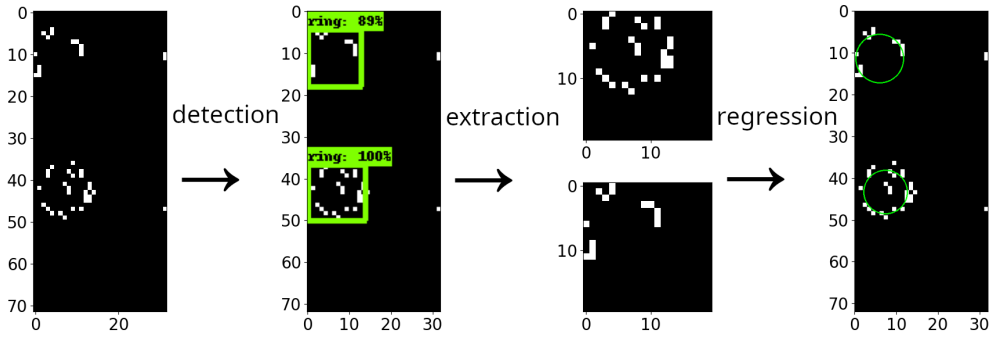

Figure 6.12: Example of the procedure for getting ring parameters from events. First, bounding boxes will be drawn around rings by an object detector. Then, these boxes will be extracted. Finally, a ring regression CNN will yield the ring parameters.

6.3.1 Object detection

Here, the open-source *TensorFlow Object detection API* [34] was used, which is built on top of TensorFlow and allows an easy to use construction, training and deployment of object detection models [34]. To create a custom object detector with this framework, it is necessary to get a decently sized dataset. It should at least contain a hundred images per class [54]. In general, these images have to be annotated manually, i.e. bounding boxes have to be drawn around the objects in the images. In this case, this process was automated by extending the toymodel (see section 6.1.3) to additionally return a dataset which contains these bounding boxes.

The pretrained *EfficientDet D0* [56] model was chosen from the *TensorFlow 2 Detection Model Zoo* [55] to be trained on the custom dataset. Most of the configuration was kept as set as the default. Training was done on a dataset of the size 5000 and a batch size of 4. SGD was used as the optimizer with learning rate/momentum schedules similar to that used in section 6.2. Here, in the warmup phase, which denotes the first part of the cycle, the learning rate was linearly increased from 0.001 to 0.01, and then decreased again as described in section 6.2. Figure 6.14 shows this learning rate schedule. Results of the model predictions

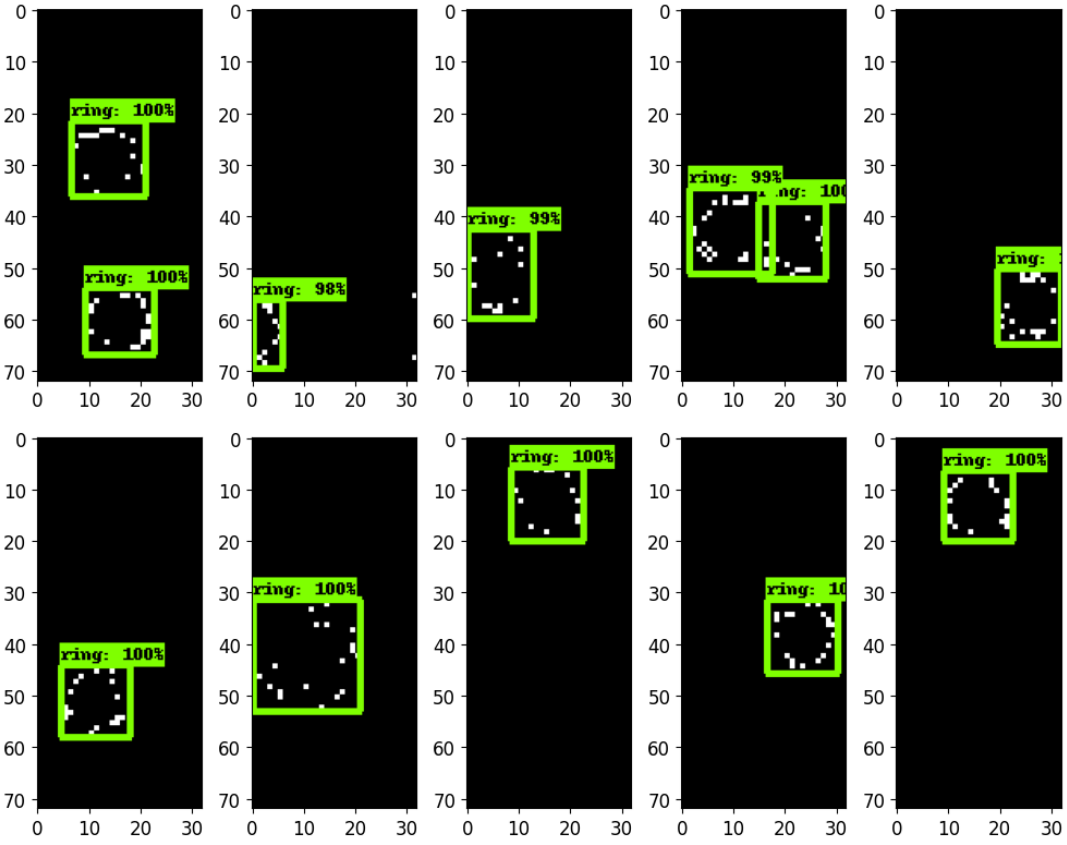

Figure 6.13: Object detection with the custom EfficientDet D0 on a dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy. Bounding boxes are drawn around rings together with the networks confidence that a ring was detected.

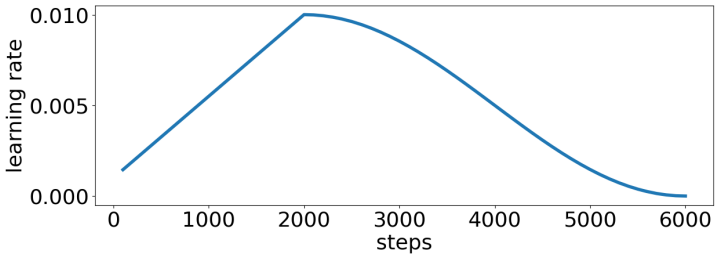

Figure 6.14: Learning rate schedule used for training the object detector.

on simulation data can be seen in Figure 6.13. All detections below a chosen confidence threshold of the network of 88% were cut.

	x (pixel)	y (pixel)	r (pixel)
$\sqrt{MSE_{CNN-idealHTM}}$	3.13	6.97	1.21
$\sqrt{MSE_{HTM-idealHTM}}$	3.29	7.36	1.13

Table 3: Errors of the detection + regression CNN and the standard HTM with respect to the ideal HTM.

6.3.2 Ring regression

Next, a ring regressor was trained for determining ring parameters for single rings. Therefore, the toymodel was modified to create a dataset with images the size of 20×20 containing only single rings. The model was trained for 3 epochs on a dataset of the size 200,000 with a batch size of 32. SGD was used with the same learning rate/momentum schedules as in section 6.2. Only the initial and maximum learning rates were changed to 0.0003 and 0.003 respectively. The model architecture can be seen in Figure 6.15. Batch normalization activation functions were applied in the same way as in section 6.2.

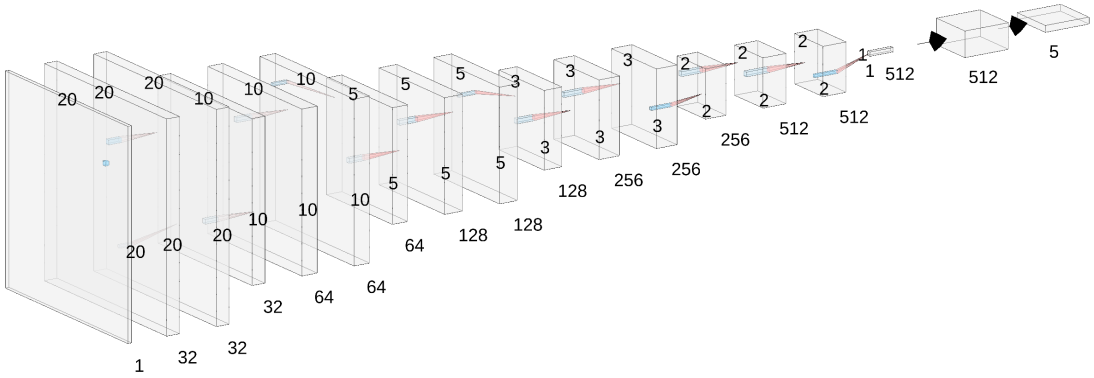

Figure 6.15: Architecture of the single ring regression CNN. The depth indicates the number of filters/feature maps in that layer.

6.3.3 Evaluation

During training of the detection model different losses were monitored, which can be seen in Figure 6.17. For the regression model loss and accuracy were monitored for both training and validation sets after each epoch (Figure 6.18). When combining both, the object detector and the ring regressor, ring parameters can be extracted from event displays. Some examples can be seen in Figure 6.19. The model was again compared to the ideal HTM and the standard HTM in Figures 6.20 and 6.21. Prediction time was about 87.14 ms per event. Additionally, the errors were calculated the same way as in section 6.2 (Table 3). Again, the number of rings per event were compared for each method, which can be seen

in Figure 6.16. Here, the histogram of the CNN contains a few events where no rings were fitted. This might be caused by a too strict probability threshold for the detection part. Otherwise, the histogram looks very similar to the one of the standard HTM.

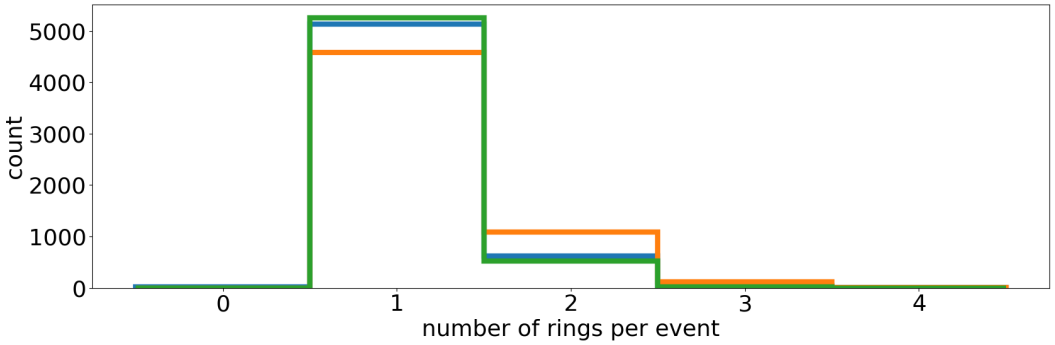

Figure 6.16: Number of rings per event for the detection + regression CNN model (**blue**), standard HTM (**green**) and ideal HTM (**orange**).

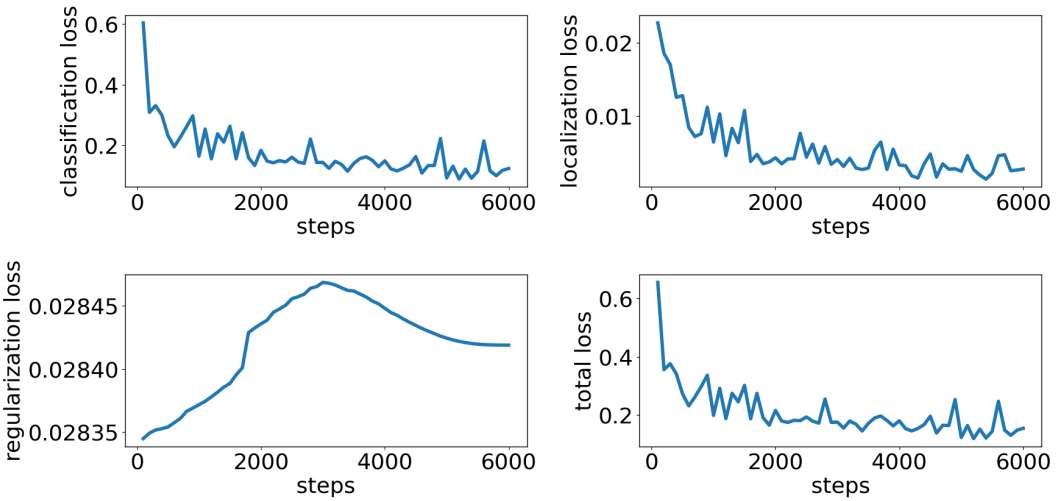

Figure 6.17: Losses recorded for training of the object detector.

6.4 Comparison and outlook

When comparing both CNN models with the (ideal) HTM the performance looks decent. For most events the fits from the CNN methods do not differ significantly from the HTM methods. However, even after filtering many events of the simulation dataset, there are still some events with substantial differences,

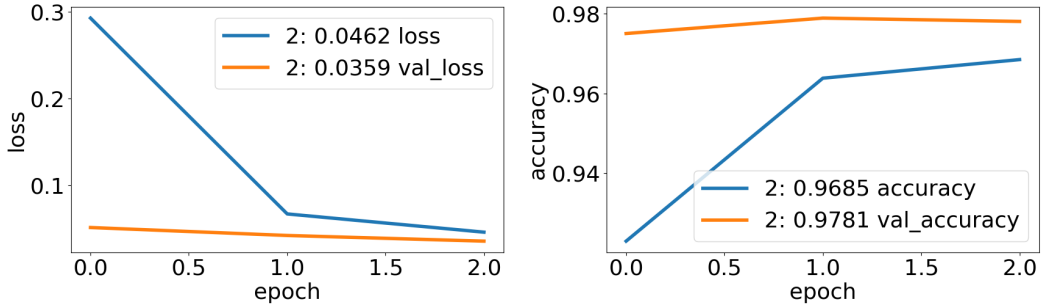

Figure 6.18: Loss and accuracy for training (**blue**) and validation (**orange**) of the single ring regressor.

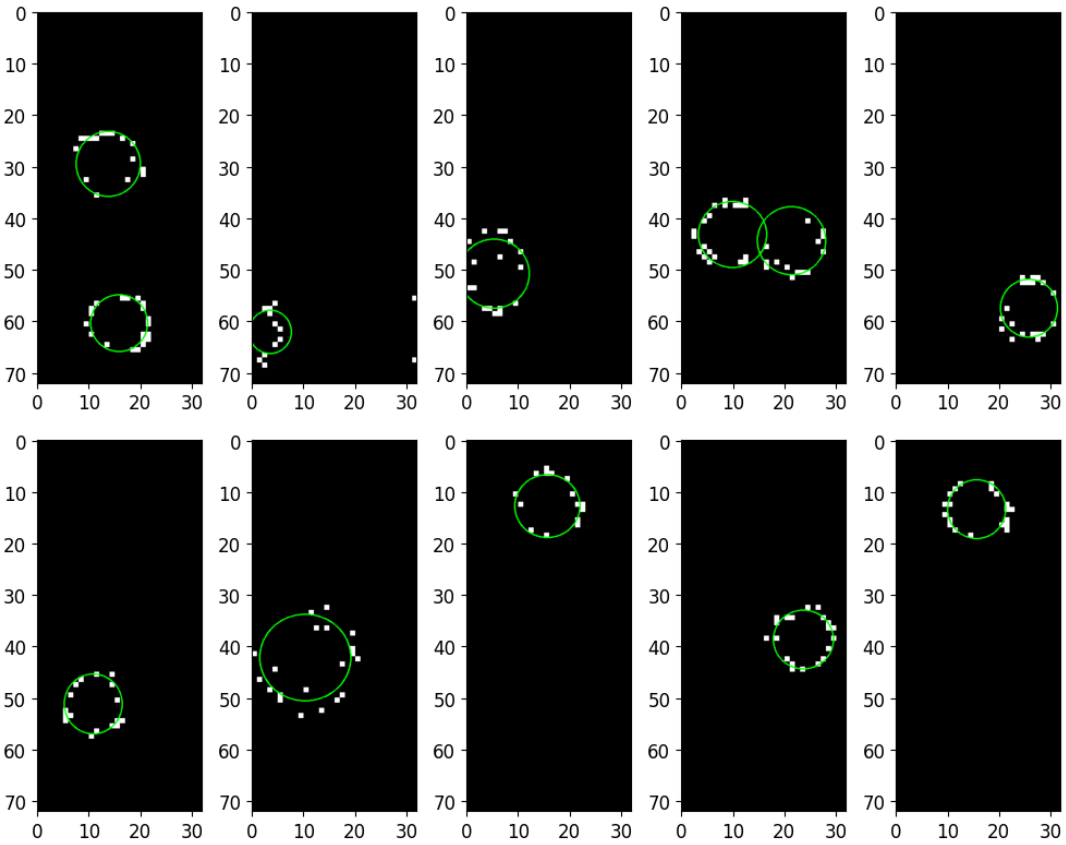

Figure 6.19: Ring reconstruction by object detection with the custom EfficientDet D0 and subsequent ring regression for single rings with a CNN. A dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy was used.

e.g. a ring fit is missing or of poor quality. Examples of such events are shown

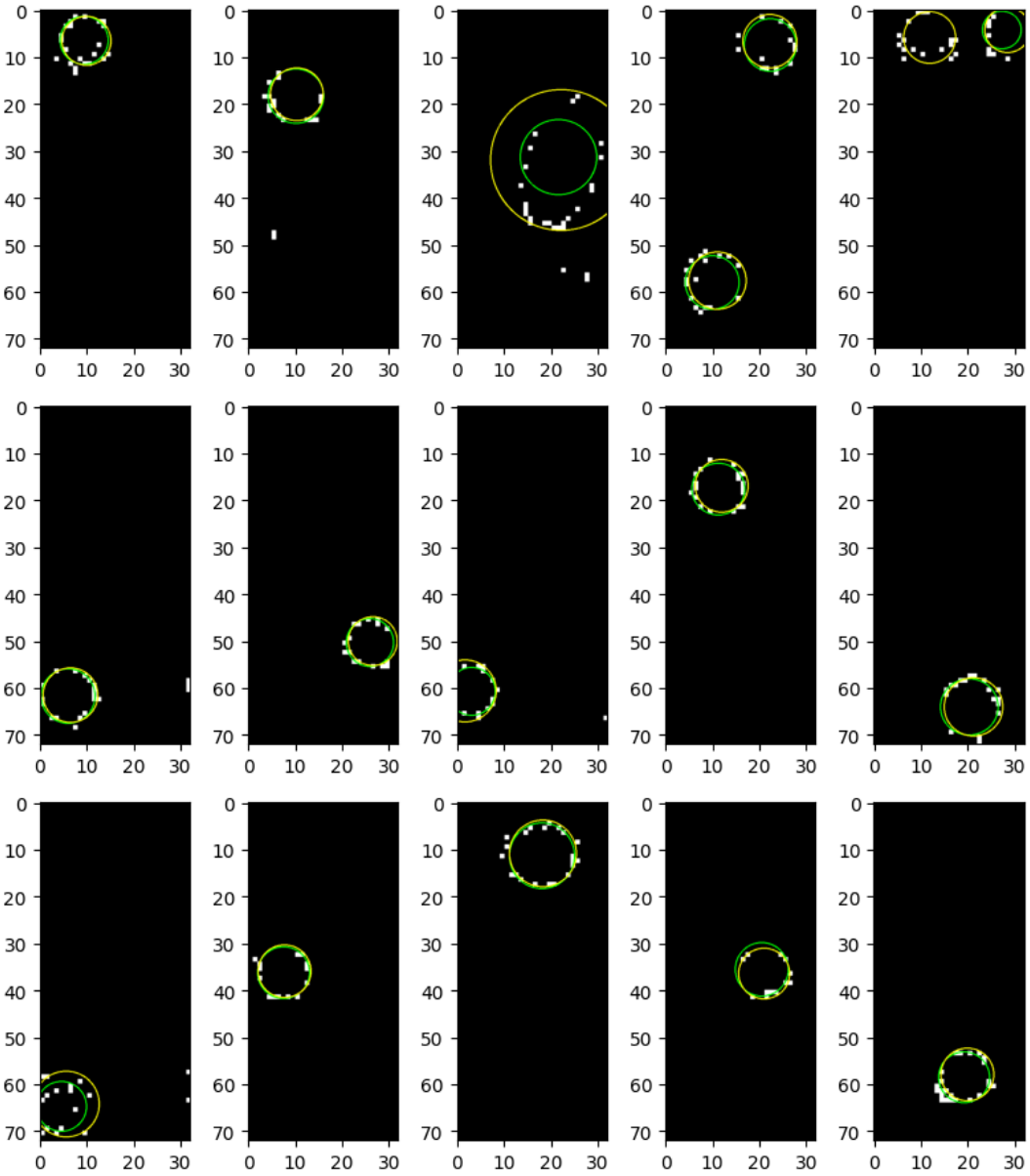

Figure 6.20: Ring reconstruction with detection + regression (**green**) vs. ideal HTM (**yellow**) on a dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy.

in Figure 6.22. These "bad" or missing fits might explain the high error values. Nonetheless, the error values of the CNN methods with respect to the ideal HTM were slightly lower than for the HTM with respect to the ideal HTM. The approach with the detection model shows better accuracy for finding the center position, while being slightly worse for radius determination. Still, the high prediction

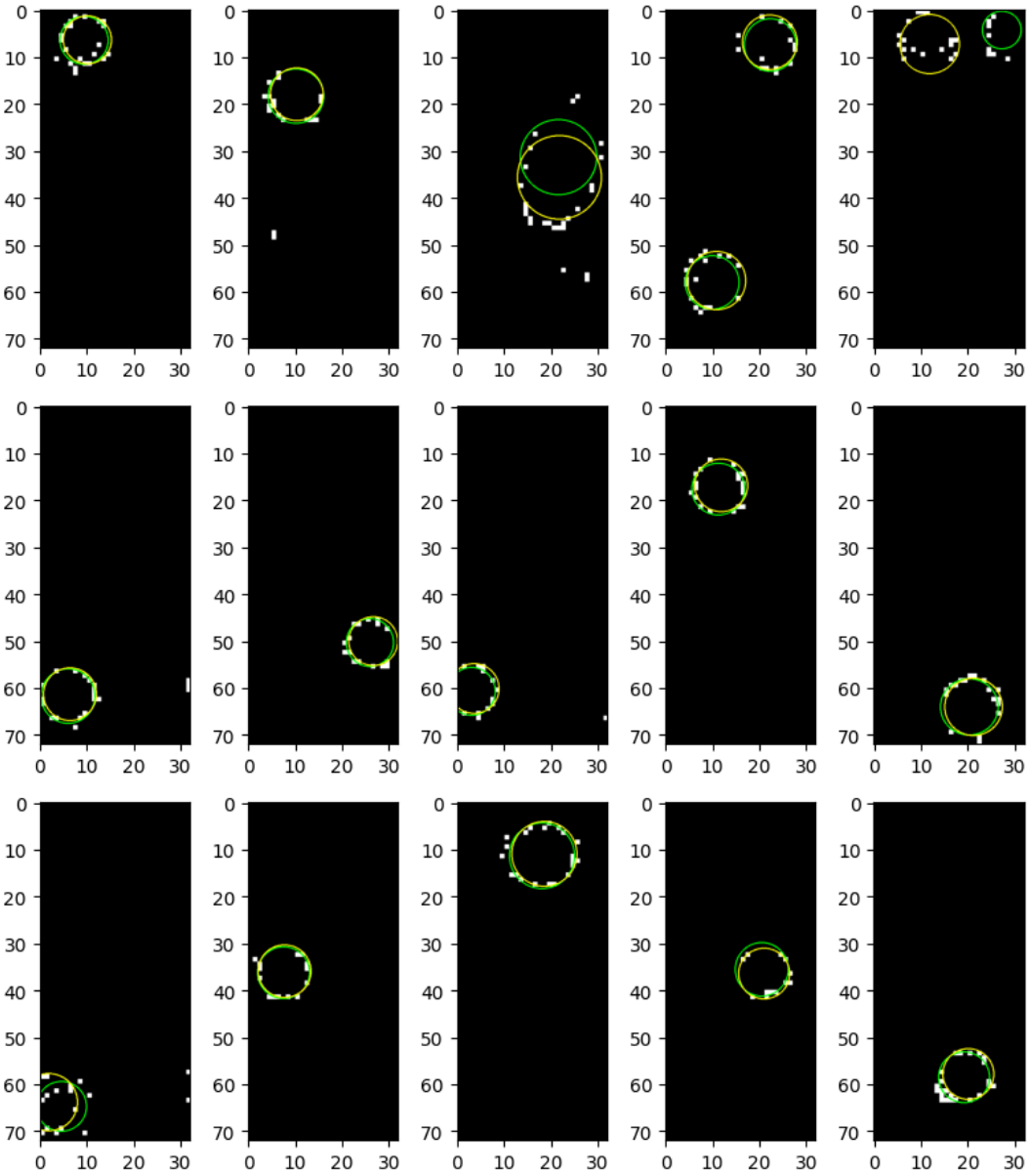

Figure 6.21: Ring reconstruction with detection + regression (**green**) vs. HTM (**yellow**) on a dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy.

time of about 87 ms per event currently makes it not suitable for CBM as fast prediction times are essential. The high prediction time is mainly due to the lack of optimization for the detection model at this time. As a reference the HTM takes about 45 µs per event [48].

Overall, the created CNN models have the potential to challenge the HTM in

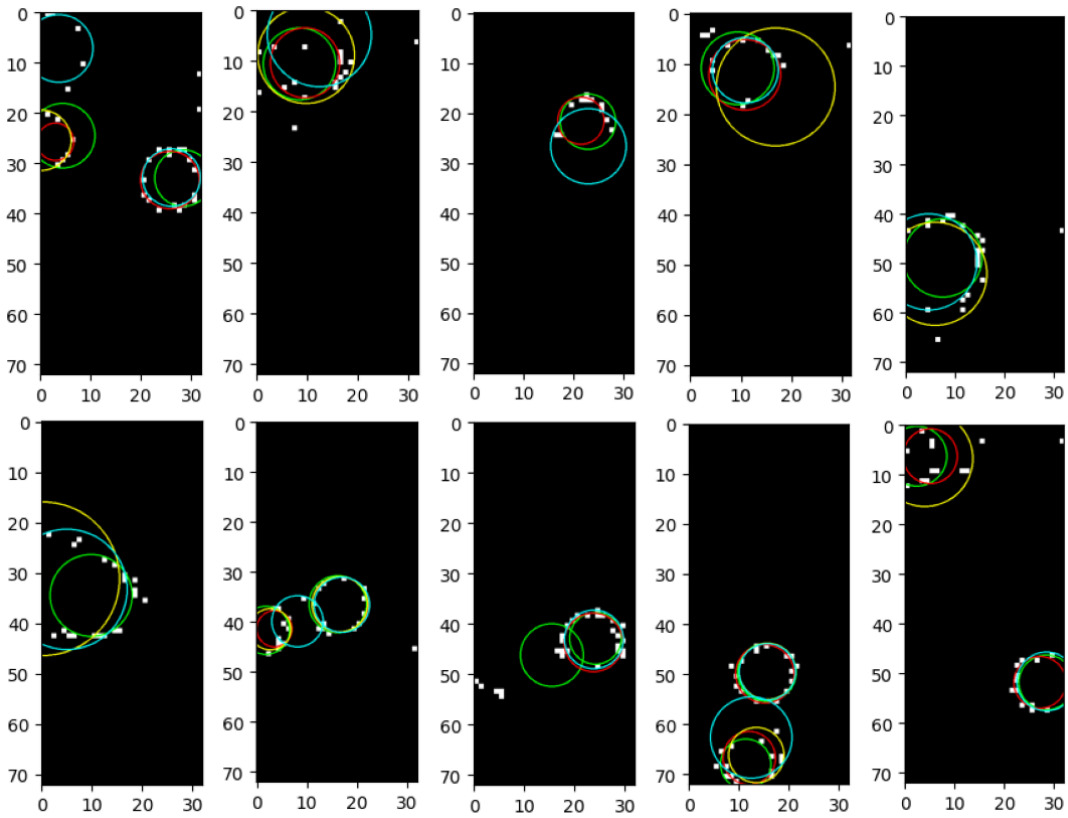

Figure 6.22: Examples of "bad" or missing ring fits for all four methods. **green:** regression model, **red:** detection + regression model, **yellow:** ideal HTM, **cyan:** standard HTM. A dataset of denoised UrQMD-simulated Ag+Ag-collisions at 1.24 AGeV beam energy was used.

terms of accuracy. Predictions times may also improve with further optimization. In the future, these models might be implemented in CbmRoot as alternative ring reconstruction methods. Currently, this is problematic as there is no sufficiently powerful tool for creating/deploying neural networks (e.g. TensorFlow) available in CbmRoot.

Danksagung

Zunächst bedanke ich mich bei Prof. Dr. Claudia Höhne, die mir ermöglicht hat an diesem interessanten Thema zu arbeiten. Ich bedanke mich auch bei Dr. Semen Lebedev, der mir bei jeglichen Problemen und Fragen zur Hilfe kam. Weiterhin danke ich der gesamten AG Höhne für die angenehme Zeit und das tolle Arbeitsklima. Besonderer Dank gebührt meiner Familie und meinen Freunden, die mich während des gesamten Studiums unterstützten.

References

[1] https://en.wikipedia.org/wiki/Standard_Model

[2] Borsányi, S., Fodor, Z., Hoelbling, C. et al. J. High Energ. Phys. (2010) 2010: 73. https://doi.org/10.1007/JHEP09(2010)073

[3] http://at-web.physik.uni-wuppertal.de/~kampert/Phasendiagram.jpg

[4] Johann M. Heuser: The Compressed Baryonic Matter Experiment at FAIR; GSI Helmholtz Center for Heavy Ion Research GmbH, Darmstadt, Germany

[5] M. Hanauske et al., J. Phys.: Conf. Ser. 878 012031 (2017)

[6] F. Gao and J. Pawlowski, Phys. Rev. D 102, 034027 (2020)

[7] T. Ablyazimov et al. "Challenges in QCD matter physics –The scientic programme of the Compressed Baryonic Matter experiment at FAIR". In: The European Physical Journal A 53.3 (Mar. 2017). https://link.springer.com/article/10.1140/epja/i2017-12248-y

[8] https://www.gsi.de/en/researchaccelerators/fair/the_machine

[9] www.cbm.gsi.de

[10] Challenges in QCD matter physics –The scientific programme of the Compressed Baryonic Matter experiment at FAIR - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-CBM-experimental-setup-together-with-the-HADES-detector-left-Each-setup-has-its_fig1_315506720 [accessed 12 Jul, 2022]

[11] Bengt Friman et al. "The CBM Physics Book - Compressed Baryonic Matter in Laboratory Experiments". Berlin Heidelberg: Springer Science & Business Media, 2011. ISBN: 978-3-642-13292-6

[12] Maksym Teklishyn. "The Silicon Tracking System of the CBM experiment at FAIR". In: EPJ Web of Conferences 171 (Jan. 2018), p. 21003.

[13] Weber, Adrian Amatus. Development of readout electronics for the RICH detector in the HADES and CBM experiments - HADES RICH upgrade, mRICH detector construction and analysis; http://dx.doi.org/10.22029/jlupub-288

[14] Contalbrigo, M. et al., Aerogel mass production for the CLAS12 RICH: Novel characterization methods and optical performance. *Nuclear Instruments And Methods In Physics Research Section A: Accelerators, Spectrometers, Detectors*

And Associated Equipment, 2017, https://www.sciencedirect.com/science/article/pii/S0168900217302644

[15] "Technical Design Report for the CBM Ring Imaging Cherenkov Detector". Tech. rep. 2013, 215 p. URL: https://repository.gsi.de/record/65526

[16] Hermann Kolanoski and Norbert Wermes. "Teilchendetektoren - Grundlagen und Anwendungen". Berlin Heidelberg New York: Springer-Verlag, 2016. ISBN: 978-3-662-45350-6.

[17] Michael F. L'Annunziata, Željko Grahek, Nataša Todorović, Handbook of Radioactivity Analysis: Volume 2 (Fourth Edition), Academic Press, 2020, ISBN 9780128143957, https://doi.org/10.1016/B978-0-12-814395-7.00006-4

[18] https://en.wikipedia.org/wiki/Ring-imaging_Cherenkov_detector#/media/File:RICH_two_types02_2013-03-15.svg

[19] Hamamatsu Photonics K. K. "PHOTOMULTIPLIER TUBES - Basics and Applications (3rd Edition)". URL: https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/etd/PMT_handbook_v3aE.pdf

[20] https://www.hamamatsu.com/eu/en/product/optical-sensors/pmt/pmt-assembly/metal-package-type/H12700B.html

[21] Charu C. Aggarwal: Neural Networks and Deep Learning - A Textbook, Springer International Publishing AG, part of Springer Nature 2018, DOI: https://doi-org.ezproxy.uni-giessen.de/10.1007/978-3-319-94463-0

[22] Meng, Zhenzhu & Hu, Yating Ancey, Christophe. (2020). Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows.

[23] https://dev.to/codeperfectplus/single-layer-neural-networks-in-machine-learning-perceptrons-18n8

[24] https://deepai.org/machine-learning-glossary-and-terms/backpropagation

[25] Ioffe, Sergey and Szegedy, Christian, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015, https://arxiv.org/abs/1502.03167

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (2014); DOI: https://dl.acm.org/doi/abs/10.5555/2627435.2670313

[27] Leslie N. Smith, Cyclical Learning Rates for Training Neural Networks, 2017, https://arxiv.org/abs/1506.01186v6

[28] Leslie N. Smith and Nicholay Topin: Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates, 2017, https://arxiv.org/abs/1708.07120

[29] https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

[30] https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5

[31] https://www.askpython.com/python-modules/opencv-filter2d

[32] Lung Cancer Detection using Deep Learning - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/Max-and-Average-Pooling-Operation-2_fig13_324728060 [accessed 7 Apr, 2022]

[33] Deep Neural Networks Meet CSI-Based Authentication - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/Typical-CNN-architecture_fig2_329608053 [accessed 7 Apr, 2022]

[34] Huang, J. et al., Speed/accuracy trade-offs for modern convolutional object detectors, 2016, http://arxiv.org/abs/1611.10012, Software available at https://github.com/tensorflow/models/tree/master/research/object_detection

[35] https://www.image-net.org

[36] Zhao, Z., Zheng, P., Xu, S. & Wu, X. Object Detection with Deep Learning: A Review, 2018, http://arxiv.org/abs/1807.05511

[37] Ren, S., He, K., Girshick, R. & Sun, J, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2015, http://arxiv.org/abs/1506.01497

[38] Lin, T., Dollár, P., Girshick, R., He, K., Hariharan, B. & Belongie, S., Feature Pyramid Networks for Object Detection, 2016, http://arxiv.org/abs/1612.03144

[39] Redmon, J., Divvala, S., Girshick, R. & Farhadi, A., You Only Look Once: Unified, Real-Time Object Detection, 2015, http://arxiv.org/abs/1506.02640

[40] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. & Berg, A., SSD: Single Shot MultiBox Detector, 2015, http://arxiv.org/abs/1512.02325

[41] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. NIPS Conference, pp. 1097–1105. 2012

[42] https://redmine.cbm.gsi.de/projects/cbmroot/wiki

[43] https://infoscience.epfl.ch/record/49909

[44] Van Rossum, G. & Drake, F.L., 2009. Python 3 Reference Manual, Scotts Valley, CA: CreateSpace. https://www.python.org

[45] Abadi, M. et al., 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from https://www.tensorflow.org

[46] Harris, C.R. et al., 2020. Array programming with NumPy. *Nature*, 585, pp.357–362. Available at https://numpy.org

[47] Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), pp.2825–2830, Available at https://scikit-learn.org/stable/index.html

[48] S. Lebedev et al., Event reconstruction in the RICH detector of the CBM experiment at FAIR, 2014, https://www.sciencedirect.com/science/article/pii/S0168900214004951

[49] P. V. C. Hough, Method and Means for Recognizing Complex Patterns, US220 Patent: 3,069,654 (1962)

[50] M. Beyer et al., mRICH pattern recognition using convolutional neural networks, CBM Progress Report 2021, https://repository.gsi.de/record/246663/files/cbm_pr2021_final.pdf

[51] Hannah Petersen et al. "A Fully Integrated Transport Approach to Heavy Ion Reac- tions with an Intermediate Hydrodynamic Stage". Phys. Rev., C78:044901, 2008, arXiv:0806.1695 [nucl-th].

[52] Leslie N. Smith, A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay, 2018, http://arxiv.org/abs/1803.09820

[53] Howard, J. & Others fastai. (GitHub,2018), github.com/fastai/fastai

[54] https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/index.html

[55] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[56] Tan, M., Pang, R. & Le, Q. EfficientDet: Scalable and Efficient Object Detection, 2019, http://arxiv.org/abs/1911.09070