Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering
Department of Physics

Machine learning methods utilization for cosmic radiation particle identification in the SXRM detector

Bc. Matěj Vaculčiak

Supervisor: Ing. Michal Marčišovský, Ph.D.

Prague, 2019

Declaration

I hereby declare, that I have written this research project by myself and I've used only the materials stated in the references section.

I have no reason to object to use this work according to the section 60 of Act No. 121/2000 Coll., On Copyright, on Rights Related to Copyright, and on Change of Some Acts (Copyright Act).

Prague, 2019 .....

Bc. Matěj Vaculčiak

Acknowledgement

I would like to thank my supervisor Ing. Michal Marčišovský, Ph.D.and his always thoughtful spouse Ing. Mária Marčišovská, Ph.D. for their valuable advice and professional guidance and my closest ones, especially my beloved Adla, for their unceasing support.

Bc. Matěj Vaculčiak

Title:

Machine learning methods utilization for cosmic radiation particle identification in the SXRM detector

Author: Bc. Matěj Vaculčiak

Field of study: Experimental Nuclear and Particle Physics

Thesis type: Research task

Supervisor: Ing. Michal Marčišovský, Ph.D.

Abstract:

This work aims to examine the possibilities of utilization of the machine learning techniques as an output data processing method for the SXRM detector. The concept of the SXRM detector and the SpacePix detection chip, which is used as a sensitive component of the SXRM setup, are presented. The region of the primary SXRM deployment, a 400 km circular orbit, is also described, together with the sources accountable for the presence of the ionizing radiation. The simulations conducted in order to obtain the necessary data, and the employed toolkit Geant4, are described. The overview of the general machine learning problematics is given, and the deep neural networks are selected for a further examination. After the summary of the training procedure and its results, the possible future research directions are presented, such as the usage of more realistic data, or the utilization of the convolutional neural networks.

Keywords: machine learning, neural networks, SXRM, SpacePix

Název práce:

Aplikace metod strojového učení k identifikaci částic kosmického záření v detektoru SXRM

Autor: Bc. Matěj Vaculčiak

Obor: Experimentální jaderná a částicová fyzika

Druh práce: Výzkumný úkol

Vedoucí práce: Ing. Michal Marčišovský, Ph.D.

Abstrakt:

Předmětem práce je zkoumání využití možnosti aplikace metod strojového učení k analýze výstupních dat detektoru SXRM. Je zde představen koncept detektoru SXRM, který je vyvíjen v centru CAPADS na FJFI, i v něm použitého detekčního čipu SpacePix. Zároveň je popsána oblast primárního nasazení detektoru, tedy kruhová orbita ve výšce 400 km a zdroje ionizujícího záření, které je třeba brát v potaz. Jsou popsány simulace, jejichž prostřednictvím byla zajištěna data potřebná jako vstup pro příslušné metody strojového učení, i softwarové nástroje, které byli za tím účelem použity. Následně je poskytnut přehled problematiky strojového učení, zároveň s určením neuronových sítí, jakožto kandidáta pro aplikaci k analýze výstupu SXRM. Po popisu průběhu trénování a dosažených výsledků jsou nastíněny další možné směry výzkumu v oblasti aplikací nejen neuronových sítí, ale i dalších metod strojového učení.

Klíčová slova: strojové učení, neuronové sítě, SXRM, SpacePix

Contents

List of Figures

List of Tables

Introduction

In the Bachelor thesis [1], a development of the SXRM detector is presented, namely, the detector simulations were the main point of interest. The SXRM is a multilayer detector utilizing the silicon pixel detector SpacePix as a sensitive component. A further description of both devices together with an analysis of the environment, where the SXRM should operate, is presented in Chapter 1.

After a thorough analysis of the aforementioned simulation-obtained data, a suggestion to examine the possibilities of machine learning utilization as an output-data-processing method was made. This examination is the main topic of the submitted work.

Overall, the SXRM detector is designed to reconstruct the original direction, species and the initial energy of a detected particle. The SXRM itself should be able to reconstruct the particle trajectory and provide information about the energy deposition along the trajectory. The potential aim of the machine learning algorithm is to take this data as an input and provide the latter two mentioned information - the particle species and initial energy.

Not being sure, if the provided data is suitable for this task, only the task of particle-species reconstruction was examined. After a brief introduction to the concept of machine learning in Chapter 3, the deep neural network algorithm is chosen to be studied. A big advantage of the deep neural networks could be the possibility of their implementation directly on FPGAs[1], so that its usage could be very quick (just a basic linear algebra operations are needed) and immediate.

A detailed description of the neural network algorithms, in general, is given in Chapter 4. The particular deep neural network is then discussed in Chapter 5 together with results of the training procedure.

The presented analysis is conducted on two successive datasets. The first one consisted of the data obtained directly by the SXRM simulations, which were fundamentally more physically pure. Afterwards, a small non-linear distortion was presented in order to study the reconstruction accuracy on a more realistic dataset.

Further steps towards the realistic-data reconstruction, such as the conversion of the deposited energy to the final so-called ADC units, or the usage of convolutional neural networks, are suggested in conclusions.

[1] Field Programmable Gate Arrays, [2]

Chapter 1

The SXRM detector

The abbreviation SXRM denotes the SpacePix Radiation Monitor. It is is a concept of a multilayer detector designed to register and recognise the ionizing cosmic-radiation particles in the Earth orbit, namely to detect and measure the properties of Van Allen radiation belts.

The term SpacePix denotes a new generation of monolithic pixel detection chips, which are the sensitive component in the SXRM setup.

1.1 Primary objective

Since the near-Earth space has lately become an area of strong research and commercial interest, the need to monitor the radiation conditions there comes naturally. The so-called space weather conditions influence the possibilities of exploring the Solar System mostly by the impending radiation damage to both potential human spaceship crew and electronic equipment.

The space weather monitoring is one of the primary objectives of the SXRM detector, mainly as having a potential swarm of such satellites orbiting the Earth could help to prevent losses caused by radiation damage and bring the ability of the space mission planning improvement.

Having a potential to not only detect but also classify and reconstruct various properties of the detected radiation, the SXRM could also be used to study the particles surrounding us and bring a glimpse of further understanding the universe.

The near-Earth radiation conditions

The radiation in the primary deployment area of the SXRM consists of three main components:

- Van Allen radiation belts,
- galactic cosmic rays,
- solar particle events.

All of these being thoroughly studied during the SpacePix detector development [1], a brief summary follows.

Van Allen radiation belts

These toroid-shaped regions are formed by the Earth's magnetosphere and capture mostly high-energy protons and electrons. These originate mainly in the Solar particle events or in a decay of so-called albedo neutrons, which emerge from collisions of primary cosmic radiation with the Earth's atmosphere.

As shown in Fig. 1.1, generally there are two main belts, inner and outer. However, as a result of the Solar activity the shape and even their number can vary[1]. In the regions around Earth's poles the belts intersect the atmosphere in so-called horns.

The adjustment of Earth's magnetic and rotational axes is slightly disordered causing the lower Van Allen belt to intersect the atmosphere again and give rise to the so-called South Atlantic Anomaly (SAA).

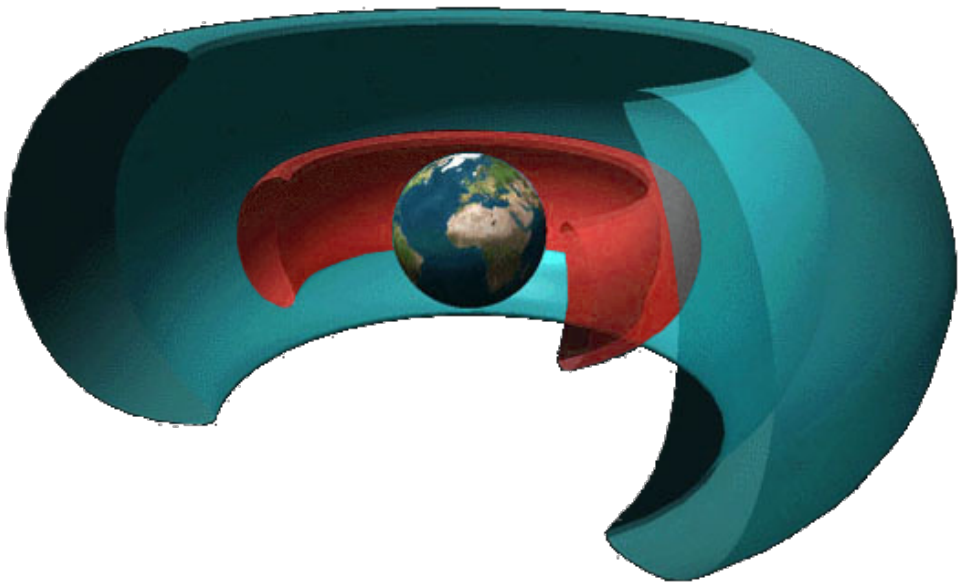

Fig. 1.1: A simplified visuzalization of the Van Allen belts [3]. The areas, of course, are not strictly bounded. The particle density rises inside the visualized areas and even the number of belts can vary.

[1] Under a specific circumstances a third radiation belt can be observed.

Galactic cosmic rays

Galactic cosmic rays originate outside the Milky Way and contain a wide spectrum of particles from protons up to heavy ions reaching relativistic energies. Fortunately, a major part of the radiation is shielded by the magnetosphere of Earth and the whole Solar System, so the particle flux is very low compared to the Van Allen belts.

However, the relativistic heavy particles can cause so-called single event upsets, damaging sensitive electronic circuits, which is why their monitoring is also kept in mind when designing the SXRM.

Solar particle events

Another source of high-energy heavy ions are unpredictable Solar particle events. There are two main causes of these events: solar flares and coronal mass ejections.

Sudden flashes on the Sun surface, solar flares, are in this sense not that interesting, as they are mostly a source of high-energy photons, not heavy ions. On the other hand, during the coronal mass ejections caused by a recombination of Sun coronal magnetic fields, a bulk of hot plasma is released into the interplanetary space, being a strong source of the heavy energetic particles.

Simulation point of interest

Considering the aforementioned cosmic radiation sources, several particle-species were chosen to be used in simulations during the SXRM development.

Firstly, electrons and protons with relevant energy spectra were chosen as their fluxes are strong (see Tab. 1.1) in the region of Van Allen belts.

From heavier ions originating in both Solar particle events and galactic cosmic rays, the lightest α-particle fluxes are considerable. The rest of the heavy-ion fluxes are more or less negligible with a slight peak at $^{56}_{26}$Fe. Helium and iron ions were therefore chosen to represent all the heavy charged particles.

To conclude, the simulations and following analyses were conducted for electrons, protons, $^{4}_{2}$He and $^{56}_{26}$Fe, with fluxes and energy spectra displayed in Tab. 1.1.

electrons		protons		$^{4}_{2}$He		$^{56}_{26}$Fe	
(0 - 10)	MeV	(0 - 250)	MeV	(0 - 1)	GeV	(0 - 10)	GeV
10^6	cm^2/s	10^5	cm^2/s	100	cm^2/s	0,1	cm^2/s

Tab. 1.1: Energy spectra and maximal integral[2] fluxes of investigated particles. Data are generated from [4] for a polar orbit at an altitude of 400 km. Detailed analysis can be viewed in [1].

1.2 The SXRM design

The SXRM detector is under development. Therefore, the final design has not been fully decided yet. Currently, there are two main geometric layouts: the conceptual and the proto-SXRM.

1.2.1 The proto-SXRM layout

The SXRM detector designed in the proto-SXRM consists of only 2 sensitive layers mediated by X-CHIP-03 chips [5] (red/yellow in Fig. 1.2), both placed on a simple PCB[3] (a blue object in Fig. 1.2) inside an aluminium case (purple in Fig. 1.2). The acceptance cone milled in the case restricts the direction of source radiation, making the coincidence measurement possible.

The proto-SXRM layout was together with other detectors developed in CAPADS[4] brought into space in July 2019 as a first testing model. It operates on the satellite SOCRAT-R on the 97,5° sun-synchronous orbit at the altitude 530 km, reaching both horns of Van Allen belts and the South Atlantic Anomaly. A picture of the whole SOCRAT-R load with proto-SXRM is in the Fig. 1.3.

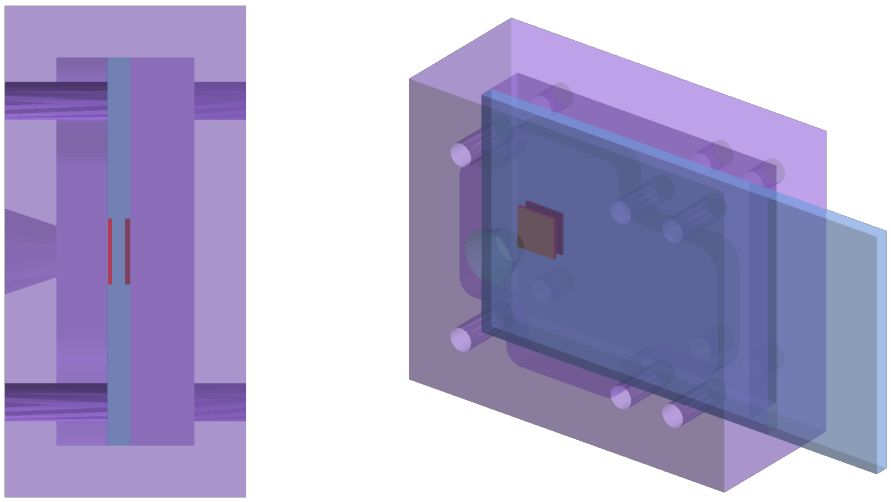

Fig. 1.2: A Geant4[5]model of the proto-SXRM layout. Sensitive X-CHIP-03 chips are coloured in red/yellow, carrier PCB blue and purple case with an acceptance cone and drilled-in holes for a mechanical connection.

[2]Integrated over the energy spectrum.
[3]Printed Board Circuit
[4]A research centre at FJFI, CTU in Prague.

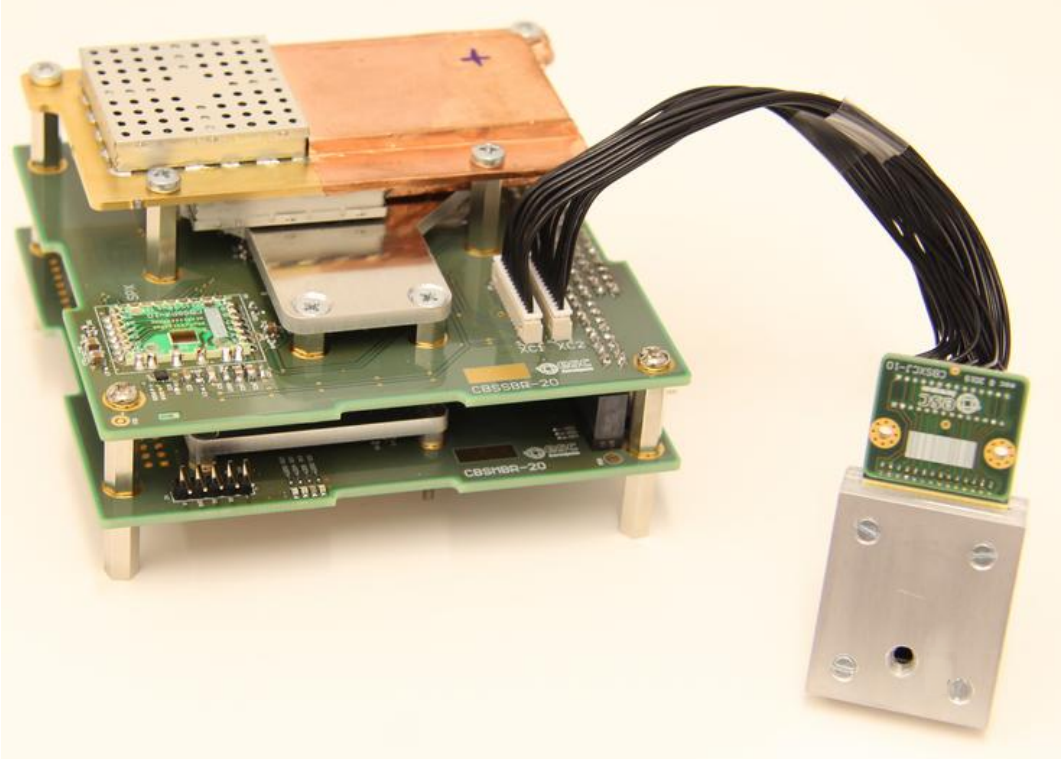

Fig. 1.3: The radiation-sensing payload of the SOCRAT-R satellite with proto-SXRM in the right bottom. The SXRM is connected to the rest of the load with black cables.

1.2.2 The conceptual layout

The conceptual layout is primary used for SXRM geometry optimization, so the materials and even the geometry itself vary[6] during the development. Latest 10-layer version is depicted in Fig. 1.4.

In general, the telescopic conceptual SXRM layout consists of 4 main components:

- the case (currently Inconel 600),
- the PCB (printed board circuits),
- the absorber (currently tungsten),
- the SpacePix chip.

[5] Geant4 [6] is a simulation toolkit further described in Chapter 2.

[6] For example, the number of layers is to be optimized based on its effect on the quality of data reconstruction with neural networks.

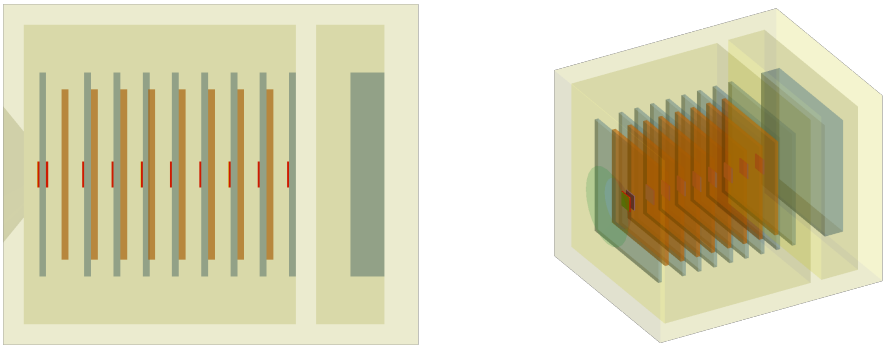

Fig. 1.4: A Geant4 model of the 10-layer conceptual SXRM layout. The yellow case has an acceptance cone covered with a turquoise titanium shielding layer and contains telescopic layout of red SpacePix chips, blue PCBs and orange tungsten absorbers.

Case

The case of the SXRM detector is made of Inconel 600 material as it provides better shielding of low energy particles then earlier used aluminium. An acceptance cone is milled in one of its sides in order to partially control the direction of the incoming radiation.

On the surface of the acceptance cone there is a thin (20 μm) layer of titanium to shield low energy photons and the active oxygen atoms in LEO[7] (turquoise plate in Fig. 1.4).

PCB

PCBs fulfil the simple purpose of a holder for the SpacePix chips and associated active electronics, providing their connection to the readout electronics[8]. The first PCB can be fabricated with a rectangular hole under the SpacePix chip, to minimize the interaction between the PCB material and the measured particles. The low energy particles that would otherwise be stopped can now reach the second SpacePix chip placed on the other side of the same PCB, so a coincidence measurement can still be conducted. Thin (100s μm), bendable flex PCBs are considered for future usage.

Absorber

The tungsten absorbers are one of the crucial components for particle reconstruction. As all of the particles of interest are ionizing, the principle of their species and

[7] Low-Earth Orbit

[8] Possibly placed in the back cavity of SXRM.

initial energy reconstruction lies in the reconstruction of the respective Bethe-Bloch curve, see [1].

As the ionizing particle traverses the material, the energy-loss rate ($\frac{\mathrm{d}E}{\mathrm{d}x}$) varies specifically for each particle. Whenever a particle deposits energy in the sensitive detector, a value[9] of the Bethe-Bloch curve is obtained. If there were no absorbers between the sensitive layers, all these points would be close to each other as the ionization losses in air are in this sense negligible and the curve reconstruction would be effectively impossible. The absorbers, on the other hand, sample the curve in the sense of expanding the distance between particular measured points. The absorber width can vary along the particle trajectory, so that the sampling can become more effective for a wide energy spectrum.

The problem of optimizing the number of layers is therefore equivalent to finding the number of points necessary to reconstruct and distinguish particular curves. The sampling effect is visible in Fig. 1.5, where the analysis of a 5-layer SXRM model performance was conducted. The example contains an analysis of a proton beam perpendicular to the sensitive layers and clearly shows the role of absorbers. The spectrum of deposited energies gets separated for various initial proton energies. Furthermore, particular visible patterns can be separated when adding the information about the hit layer, see [1].

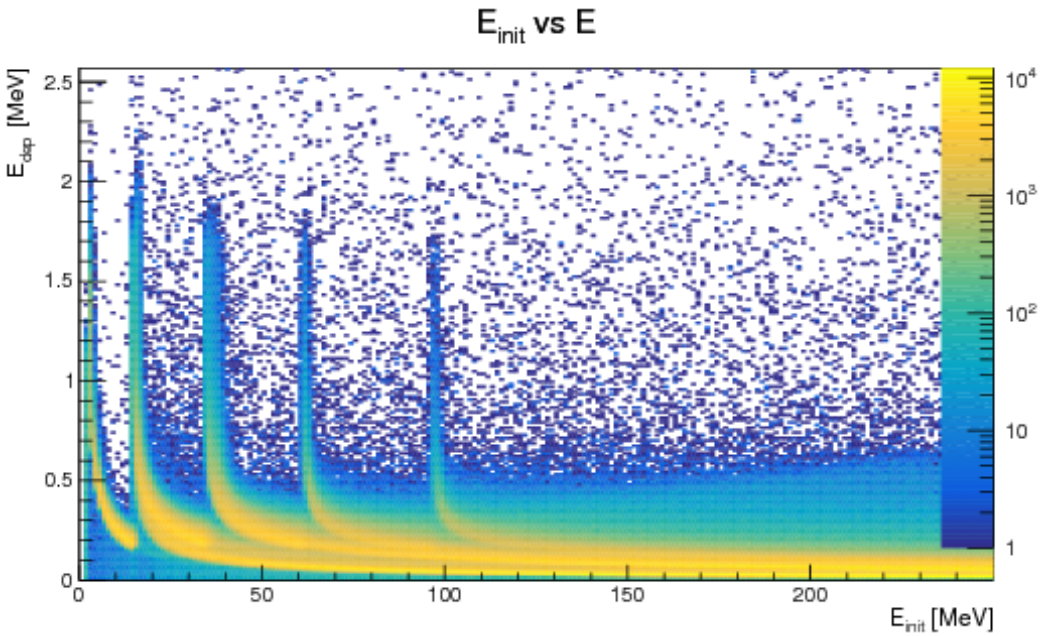

Fig. 1.5: A dependency of the energy deposited in sensitive areas of a 5-layer SXRM model on the initial particle energy, see [1].

[9] Actually an integral value over a small distance is obtained.

SpacePix

SpacePix [7] is a radiation-hard silicon monolithic pixel detector (see Fig. 1.6), part of whose development is described in [1]. Ideally, it should be capable of detecting all the particles described earlier as a point of simulation interest.

The matrix consists of 64×64 pixels with 60-μm pitch providing high spatial resolution and is covered with a 14-μm-thick layer of an insensitive material carrying essential electronics. The depleted region is created by a bias voltage in the top 30 μm of a pixel, depicted as a light-red region in the cross-section in Fig. 1.6.

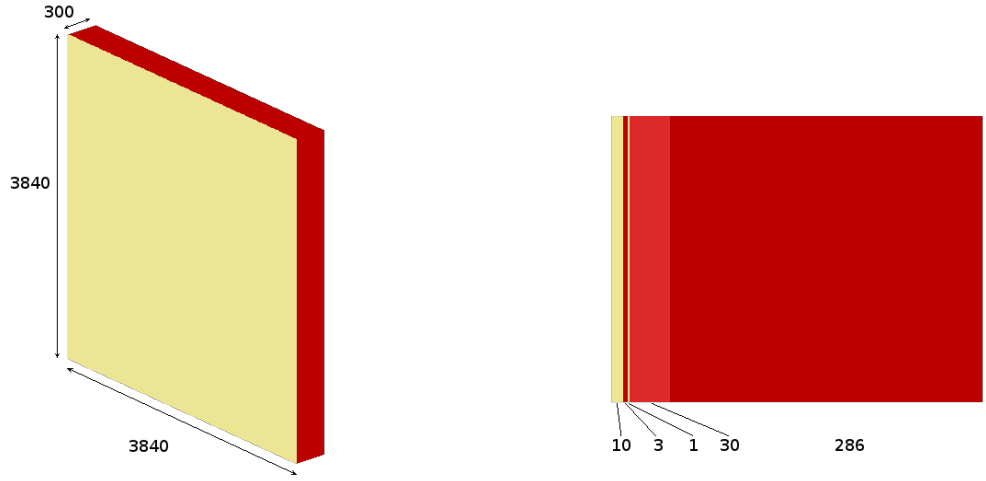

Fig. 1.6: A Geant4 model of the SpacePix chip, the dimensions are in μm. The insensitive regions with essential electronics are coloured in yellow, the silicon bulk is red. The light-red area visible in the cross-section (right) is the depleted region, where the detection occurs.

Chapter 2

Detector simulations

First practical part of this work is dedicated to detector simulations. Many improvement ideas and design concepts must be tested during the SXRM detector development. In an ideal case a full setup would be constructed and sent to the orbit every time a design change was made or an idea was to be verified. This is obviously impossible for financial and all sorts of practical reasons, mainly the time consumption.

Considering more realistic options, particular specialised irradiation facilities here on Earth, such as proton centres, could be used. This way the setup can be exposed to a specific kind of radiation, ideally the same as in the case of real usage. However, utilizing the irradiation facilities is still expensive and time-consuming. Furthermore, only a specific type of particles with limited properties (i.e. energies, spatial distribution, etc.) can be provided. Therefore, to cover the testing of the effects of all particles with energy spectra mentioned in Tab. 1.1, the utilization of specialized facilities is still not an optimal way of particular development-step validation.

The fastest and most flexible way to do so is to reproduce the real-life situation with computer simulations. Simulation tools, such as Geant4, which was used in this work and will be further described later, were developed for this purpose. The key feature making simulations a crucial part of detector development is the variability. Virtually any detector model can be programmed and exposed to any kind of radiation including various species, energies, and even spatial distributions.

A series of SXRM models was created in Geant4 in order to conduct simulations of particle-detector interactions and validate individual design changes during the development process. Particular models vary in number of sensitive layers, used materials or for example thickness of PCBs. Two of the models (the conceptual SXRM and the proto-SXRM) were already described in Chapter 1. The 10-layer conceptual model is this chapter's main point of interest as it was used to simulate data used later in this work.

The data obtained by simulations were used for various general design analyses, such as material optimization, during the development process. In the context of

this work, the main data usage was as an input dataset for the neural network training - to see, if this machine learning method is suitable to be employed as an efficient SXRM output analysis technique.

2.1 Geant4

Geant4 [6] is a toolkit providing a set of C++ libraries to simulate the physical effects of a particle traversing through a material. It was developed at CERN as a follow-up of Geant3, which was used earlier to provide simulation data during major CERN experiments (ATLAS, ALICE) development. Currently, Geant4 accounts for one of the most widely used simulation tools in detector physics.

Geometry construction

The major advantage of Geant4 is the possibility to create complex geometrical objects with a wide range of possible materials. Initially, the *world* volume is created. It represents the coordinate space - the virtual world in which the whole simulation takes place.

Generally, the Geant4 objects (including the world mother volume) are created in three successive levels:

- Solid Volume,
- Logical Volume,
- Physical Volume.

Firstly, a so-called *Solid Volume* is created. This instance contains merely the information about the object geometry, i.e. dimensions and shape. Secondly, the material properties are assigned to the object to create a *Logical Volume*[1]. At last, a *Physical Volume* is created by placing the Logical Volume properly into the coordinate space with additional properties such as an index of the particular object, etc.

All of the objects used in the virtual environment must be created this way without any intersections as these could cause a subsequent errors in the program. Note, that as all of the geometry is defined directly in the C++ code, it is impossible to change it during a simulation.

Physics List utilization

Another important advantage of Geant4 is the use of so-called Physics Lists. These classes contain information about physical processes employed during the simulation process. As particle traverses through the material it loses energy via various

[1] The Logical Volume can be imagined as a particular LEGO building block.

mechanisms dependent on the particle species, energy, etc. (see [1]). Geant4 offers the possibility to pick only the appropriate interaction mechanisms in order to save computational time. Naturally, the more realistic the simulation is, the better. However, the possibility to omit negligible physical effects can lead to a significant speed-up of the simulation. It is up to the physicist conducting the simulation to be aware of the effects incurred by leaving out parts of physics and to find the optimal quality to efficiency ratio.

To ensure high simulation quality, there are several reference Physics Lists[2], that are regularly updated and validated by the real physical data provided by major facilities like CERN.

Particle-passage simulation

Being a set of C++ libraries, Geant4 does not provide a general Graphical User Interface (GUI). It can only be displayed after the whole geometry of the experimental virtual world is prepared. After the code is compiled, no further geometry adjustments can be made and the the simulation can be conducted. An optional GUI can be displayed, so that the simulation performance can be visually validated and particle trajectories can be observed, see example in Fig. 2.1. Geant4 also provides the possibility to control the simulation via special macros.

At the beginning of the simulation a particle source is defined. It contains information about the primary particles such as species, direction and their energy distribution. The very last information provided is the number of particles to be generated in one beam. All of the particles are then simulated one by one and after the whole beam is processed, the particle source properties can be redefined again.

A simulation of a single particle passage consists of a number of steps. After a step is done, a distance to another location, where the particle interacts, is calculated. Afterwards, the particle is propagated to this place, and the process repeats until the particle energy falls under a pre-set threshold value.

If needed, the data acquisition can be done after every single step (this is, for example, the case of a Bragg curve analysis) or at the end of a particle propagation, from a predefined volume. In the latter case, a sensitive detector[3] must be assigned to this volume, e.g. a pixel in a SpacePix chip model.

[2] Such as the FTFP_BERT used for the SXRM simulations. Other reference physics lists can be found on https://geant4.web.cern.ch/node/155.

[3] A class storing predefined physical information in a specific volume, e.g. an energy deposition in a pixel.

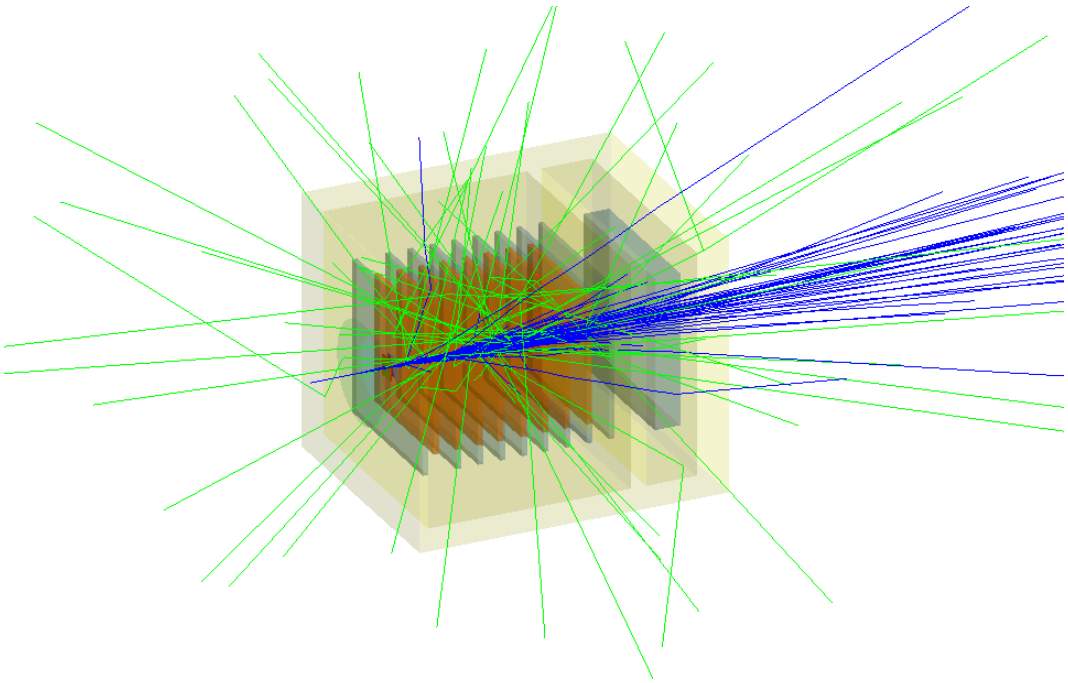

Fig. 2.1: A graphical output of a Geant4 simulation. A beam of 100 protons with energies from 0 MeV to 250 MeV traverses through the SXRM detector. The curves depicting the particle trajectories distinguish between electrically positively charged (blue), neutral (green) and negatively charged (red) particles.

2.2 SXRM simulations

For the purpose of the SXRM detector development a number of various simulations was conducted. However, as this work aims to verify the possibility of neural-network usage for the SXRM-output analysis, only some of the simplest simulations were used.

The sensitive element of the SXRM detector is the SpacePix - an ionizing radiation detector, therefore the main focus should be put on the electromagnetic physics. Considering this, the FTFP_BERT[4] reference Physics List was used during the simulations as it provides well-validated electromagnetic physics [8]. Because ions $^{4}_{2}$He and $^{56}_{26}$Fe were also a subject to simulation, slight changes were made to the original FTFP_BERT in order to include the ion elastic physics effects.

In Chapter 1, the role of absorbers in the SXRM setup was explained together with the method of Bragg-curve reconstruction. Therefore, the first thing to verify is that neural networks are able to utilize the idea discussed in Chapter 1 and give satisfactory results for particle classifications. Therefore, the spatial beam

[4]Used for example by the ATLAS experiment.

distribution could be neglected[5] and unlike in the case of other SXRM simulations, the particle source was stationary.

Simulations of particles defined in Tab. 1.1 with appropriate energy spectra were conducted with a beam statistics of 10^5 particles per species. Simulations with 10^7 particles per beam were also conducted but did not bring any major improvement while the training duration increased dramatically. The initial trajectory was always perpendicular to the telescopic SXRM setup as indicated in Fig. 2.2. Overall, the data used for a neural-network training was made of $4 \cdot 10^5$ entries, 10^5 for each particle.

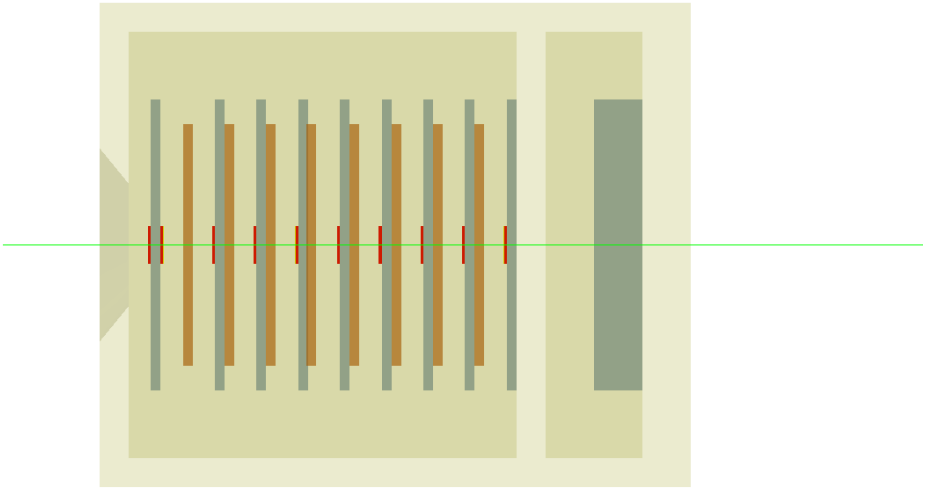

Fig. 2.2: A beam direction of the simulated particles (electrons, protons, ${}^4_2$He and ${}^{56}_{26}$Fe. The trajectory is depicted on an example of the non-interacting toy particle geantino.)

[5]Knowing the fundamental idea behind the particle-species reconstruction it is obvious, that a more complex spatial source distribution would not affect whether the neural network is fundamentally able to utilize the input data to classify particles.

Chapter 3

Machine learning

Machine learning methods provide an alternative approach to solving various types of complex problems[1] concerning data analysis and quickly became a widely used tool in various aspects of everyday life. The current success of machine learning is related to technological progress as the increase in the available computational power is significant. Furthermore, large amounts of various training data are available as a result of unceasing internet usage expansion. These two aspects can be considered the main reasons that the learning of the algorithms could have become fast and efficient.

According to [9], the term machine learning designates a field of computer science that studies algorithms and techniques for automating solutions to complex problems that are hard to program using conventional programming methods.

Conventional programming methods tackle every individual problem separately. A detailed program design is created for a particular problem and implemented as a program in a certain programming language. This means that all the steps conducted during the program run are predefined and stay changeless during the problem solution.

Machine learning methods, on the other hand, approach problems in a fundamentally different way. The main idea is to exploit the particular analysed data to adjust the program itself and so reach better performance. Unlike in the case of conventional programming methods, only a conceptual program design is created here, and the final state is decided (learned) by the program.

The aforementioned complex data analysis problems approached by machine learning can be generally categorised in 3 main domains: classification, clustering and prediction.

[1] Popular examples of complex problems solved by machine learning methods are hand-written-text recognition, language translation, or the progress in autonomous-vehicle development.

Classification

Solving a classification problem requests the algorithm to assign an input to one of the previously defined classes. An example of such a problem is distinguishing between a picture of a cat and a dog.

Clustering

Clustering means dividing a set of inputs into subsets based on a specific similarity between the data. However, unlike in the case of classification, the number of subsets does not have to be known beforehand. Therefore, the clustering can be particularly useful for revealing previously unknown or unnoticed relations between the input data.

Prediction

Prediction tasks require the algorithms to learn based on historical data and try to predict a model of the system's future behaviour. As an example of these algorithms' utilization a stock exchange, where the stock-value prediction is of interest, can be mentioned.

3.1 Method types

There are two main distinguishable types of input data: labelled and unlabelled. The labelled inputs consist of pairs containing the data itself and a specific label. This means that when a machine-learning algorithm is provided with a labelled input, it can also be verified whether the output is correct or not. Unlabelled data, on the other hand, do not contain such information, and therefore can only be used in specific machine-learning methods. Having access to labelled data is obviously more comfortable. However, obtaining the labels requires often a lot of human resources and, therefore, it is costly.

As discussed at the beginning of this chapter, machine learning comprehends a whole field of various algorithms. In order to get a clearer overview, the particular methods can be categorised based on the fundamental operation principles as follows [9]:

- supervised learning,
- unsupervised learning,
- semi-supervised learning,
- reinforcement learning.

3.1.1 Supervised methods

Algorithms classified as a supervised learning work with labelled datasets exclusively, so any time a data point is processed, the correctness of the output can be verified thanks to the provided data label. This way, the supervision over the learning of a machine is possible.

Intuitively, these methods are, in a way, some of the closest to the case of natural human learning. Similarly to a child learning to recognise various animal species, the supervised methods also utilize having somebody to tell them what animal they are looking at. While in the case of a child this somebody is a parent, for machines it is the label in the dataset, who gives it the right answer. Based on this supervision a successive algorithm improvement is possible.

There are two main general problems that can be approached by supervised methods: classification and regression.

Classification

Classification as described earlier is a problem defined as redistributing points of dataset into specific predefined subsets. Example of such a problem is depicted in Fig. 3.1, where data points (points in the scatter plot) are labelled (distinguished by colour). The machine's task is to learn to assign a correct colour to a previously unknown point.

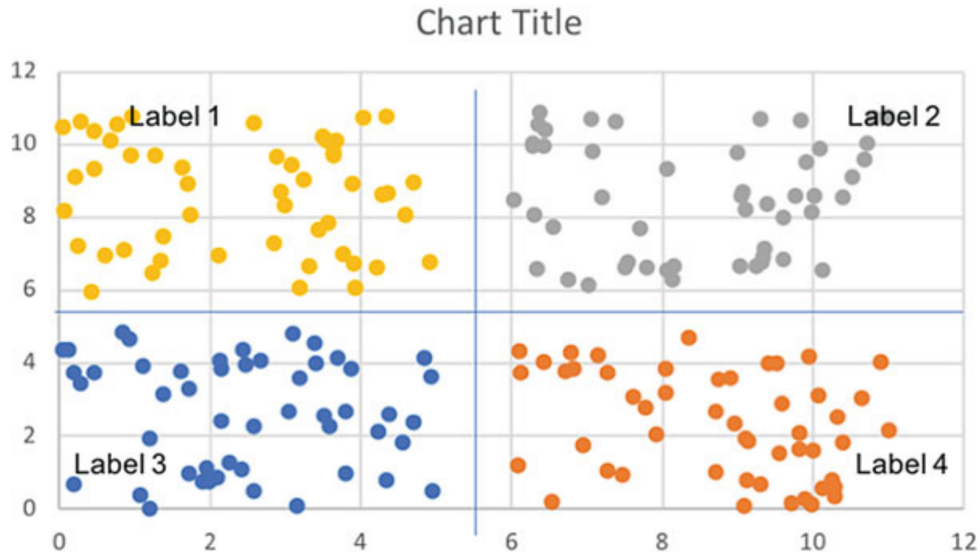

Fig. 3.1: A toy example of a labeled dataset for a classification algorithm. The classification result is depicted by the blue lines dividing the plot into four distinct areas. After a previously unknown data point is provided, the machine classifies it based on which of the four areas it belongs to. Figure taken from [9].

An important feature of the classification algorithms is that the set of classes is predefined and distinct[2]. If the output is considered as a variable it is a distinct variable.

Regression

Being a subject of supervised learning, regression algorithms also require labelled dataset as an input, just like the classification ones. However, unlike in the earlier case, the output comes as a continuous variable[3].

The general idea of regression is to create a model able to make forecasts, so in terms of the earlier problem classifications, regression helps to solve prediction problems. Therefore, as an example, the prediction of stock value can be given again.

3.1.2 Unsupervised methods

Knowing the meaning of the supervision in supervised learning methods, it is clear, that in the case of the unsupervised ones the unlabelled data is used. The machines learning without supervision do not use any feedback during the training and only focus on finding trends of similarities in the particular data points.

These methods are tightly connected to clustering problems and can be helpful in finding complex, previously unseen connections between data points.

3.1.3 Semi-supervised methods

Similarly to the situations with supervised learning with labelled data and the unsupervised learning with unlabelled data, the semi-supervised learning methods utilize partially labelled data. This amounts to a machine having a big amount of input data only a part of which is labelled.

As mentioned earlier, the main difficulty with labelled data is that it is hard to obtain. Semi-supervised learning methods omit this problem as it is able to use unsupervised training to cluster the data and follow with using the small number of labelled ones to assign labels to the whole set. The main difference to the unsupervised learning is that thanks to the labels, the data is not abstract here anymore.

[2]The term "distinct set" is used in [9]. Describing the set as finite would probably be more precise.

[3]Similarly to the classification case, for the set of all possible outputs the term infinite is probably more precise.

3.1.4 Reinforcement methods

The reinforcement learning methods bring some of the most popular results, such as computers defeating chess grandmasters.

These methods focus mainly on tasks with huge number of possible states, just like in the mentioned case of a chess game.

Another focus of reinforcement methods are situations that change during the program run. This, of course, concerns games again[4], but also on practical aspects of human life such as autonomous-vehicle development. The environment keeps changing and the machine must be able to react properly in real-time.

The main component of the reinforcement machine is an agent. It is the program itself, observing and learning from the environment. This environment changes due to external impulses or due to actions of the agent itself. Because of the agent-environment interaction, the concept of time also has to be presented, so the causality can be preserved. Therefore, particular actions happen in time-steps. Last important thing that has to be well-defined is the agent's objective. This way the agent can get feedback to its actions, for example, if it conducts a chess move leading to a loss, it notes.

The training then happens by the agent trying various decision sequences and evaluates if the objective was achieved and if the way of achieving it can be optimized.

3.2 SXRM-output-analysis method

Because of being generated in the Geant4 simulations, the whole input dataset, used for SXRM development, is labelled. Therefore, the supervised-learning methods can be utilized.

The main objective is to be able to distinguish detected particles based on their species (electrons, protons and ions $^{4}_{2}$He and $^{56}_{26}$Fe). Clearly, this is a classification problem with the result-space of four classes.

From the variety of supervised-learning algorithms, the well documented and widely used neural networks have been chosen. Therefore, the following chapter will be dedicated to their thorough description.

Another potentially interesting task could be to use machine-learning algorithms to reconstruct the initial particle energy (a regression problem). This is a subject to future research.

[4] One can, for example, find youtube videos of machines playing Snake or Mario.

Chapter 4

Neural networks

Neural networks are currently a very popular data-analysis programming paradigm. Opposing the conventional approach, where the general problem is split into smaller tasks easier to solve by computer, a neural network itself learns by observing the data, eventually being able to solve complex problems such as hand-written-text recognition or, as shown in this work, cosmic-radiation-particle classification and in a way energy reconstruction.

In the context of this work, neural networks are used as a supervised learning method, as the testing data set is separated from the whole data set at the beginning and provided to improve performance during the training process.

4.1 Architecture

Each neural network has a specific internal structure defining its learning properties. This structure, starting with the basic single neuron, is a subject to this section.

Neurons

The centrepiece of a neural network structure is a neuron. This object takes as an input a set of information (x_1, x_2, x_3 in Fig. 4.1) and returns a number as a result, so-called activation a. The neuron itself consists of three basic components:

- weight vector $\vec{w}$,
- threshold b,
- activation function f.

The *weight vector* $\vec{w}$ encodes the information about the significance of a particular element of the input vector. The larger effect the input has on the final decision, the larger the value of the respective weight is. Each element of the weight

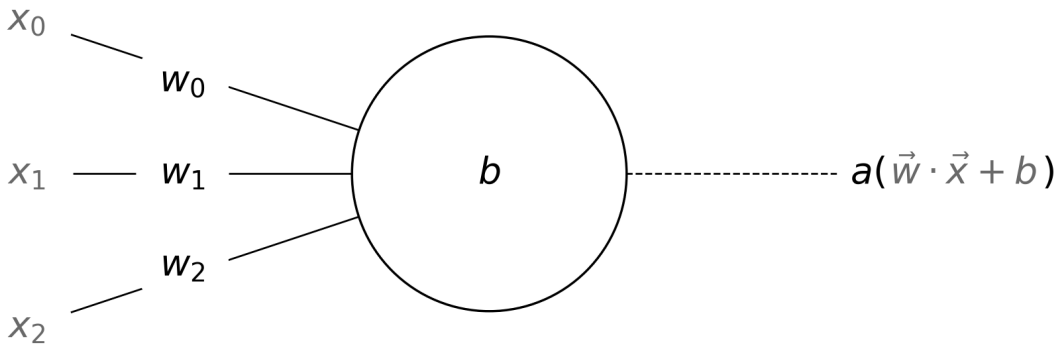

Fig. 4.1: A simplified scheme of a single neuron with three input connections $(w_0, w_1, w_2) \equiv \vec{w}$, threshold b and an activation function a.
Processing input data $(x_0, x_1, x_2) \equiv \vec{x}$ results in a single output $a(\vec{w} \cdot \vec{x} + b)$.

vector is represented with a single line connecting the input layer (x_1, x_2, x_3) and the neuron in Fig. 4.1.

The *threshold* b is depicted inside the neuron itself. Loosely speaking, it represents the general willingness of neuron to be activated (to output 1)[1].

The *activation function* $f(z)$ somehow characterises the way, how the neuron makes the final decision. It usually maps the input real number to a subset of [0, 1] interval, depending on the chosen function. However, there are exceptions such as currently widely used ReLU. Since various problems need various approaches, there are several particular activation functions used. From the most common, following activation functions (see Fig. 4.2) can be shown:

- the step function $f(x) = \begin{cases} 1, & x \leq 0 \\ 0, & x > 0 \end{cases}$, mapping $\mathbb{R} \to \{0, 1\}$ [2],
- the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$, mapping $\mathbb{R} \to (0, 1)$,
- ReLU[3] $f(x) = \max(0, x)$, mapping $\mathbb{R} \to \mathbb{R}^+$.

[1] A nice way to approach the functionality of neuron (for more detailed description see [10]) is to imagine a situation, when one is deciding whether or not to go to a concert, based on if a friend is to join them (x_1), the weather is good (x_2) or if the place can be reached by car (x_3), all of these represented by boolean 0 or 1. Multiplying the inputs (0 or 1) by a weight (personal importance of the factor), one gets a single number that can be compared to a previously stated threshold and the final decision can be reached

[2] Such neuron is denoted as a perceptron.

[3] Rectified Linear Unit

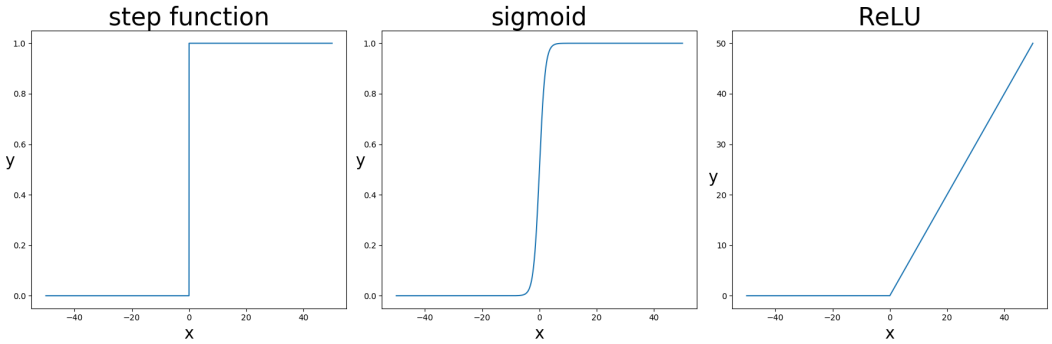

Fig. 4.2: Comparison of various activation functions: step function, sigmoid function and ReLU respectively.

In general, the procedure of information passing through a neuron can be described in three main steps:

1. As the input vector $\vec{x}$ enters a neuron, it is multiplied by the weight vector $\vec{w}$.
2. The threshold value is added to the result of the scalar product $\vec{w} \cdot \vec{x}$.
3. The resulting number is fed as an input to an activation function $f(z)$, whose result is the output of the whole neuron.

This whole operation can be interpreted as a decision made by neuron, based on the available information.

Mathematically speaking, a neuron is simply a multivariable function whose output (activation), is a single number and the whole procedure in a single neuron can be expressed by equation

$$a(\vec{x}) = f(\vec{w} \cdot \vec{x} + b). \tag{4.1}$$

Neural network structure

Having the neuron described, let us now construct a more complex structure capable of making more subtle decisions - a neural network.

It consists of neurons organized in layers stacked up one after another. Each neuron in a layer is connected to all neurons in neighbouring layers. Example of such a network is depicted in Fig. 4.3.

Three types of layers are distinguished in a neural network.

Input layer is the first one (4 neurons in Fig. 4.3). It contains the input data and its neurons do not have any bias or weights.

Output layer is obviously the last one (2 neurons in Fig. 4.3), carrying the net's final decision.

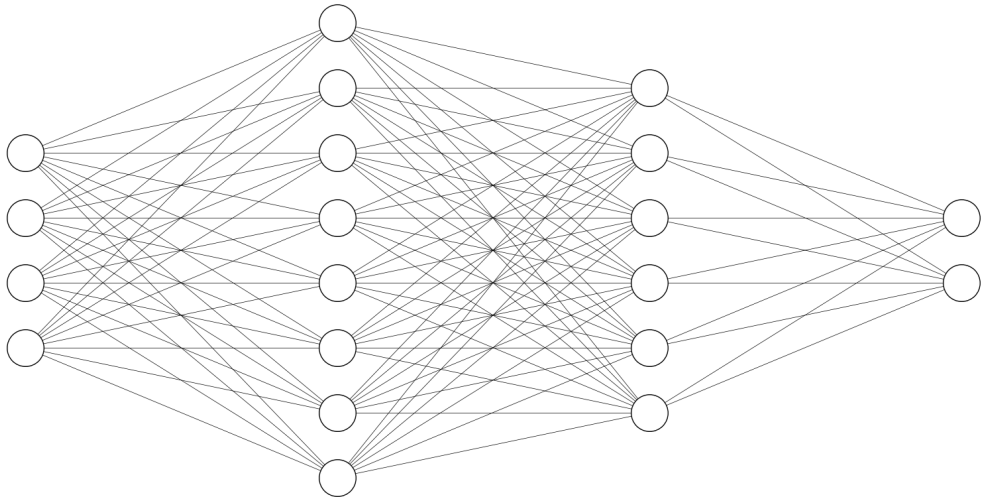

Fig. 4.3: Example of a deep neural network consisting of 4 layers: 4 neurons in input the layer, 8 and 6 neurons in hidden layers respectively and 2 neurons in the output layer.

Any layers between these two are denoted as *hidden layers* (8 and 6 neurons in Fig. 4.3). Neural networks with two or more hidden layers are called *deep neural networks*.

Earlier in this chapter, an output of a single neuron, whose input came from the very first layer $\vec{x}$ was described. Looking at the neural network as a whole, the output of a full layer needs to be described instead. To find an expression for the output of l -th layer, whose input comes from a previous (l -1)-st layer we need to extend (4.1) to a vector form

$$\vec{a}^{\,l}(\vec{x}) = \vec{f}^{\,l}(\mathbb{W}^{\,l} \cdot \vec{a}^{\,l-1} + \vec{b}^{\,l}). \tag{4.2}$$

It is straight forward, that the output of the l -th layer will be described with activation vector $\vec{a}$ and it's thresholds $\vec{b}$. The individual weight vectors get organised into a weight matrix $\mathbb{W}$, for example the matrices $\mathbb{W}_1$, $\mathbb{W}_2$ and $\mathbb{W}_3$ in Fig. 4.3 have dimensions $(4, 8)$, $(8, 6)$ and $(6, 2)$, respectively[4].

[4]By a simple matrix multiplication it can be verified, that element $\mathbb{W}_{jk}$ really connects j-th neuron in the l -th layer and k-th neuron in the (l-1)-st layer.

4.2 Learning

The training procedure happens under a "supervision" of the data labels. In the case of neural networks, the feedback on how well the algorithm performed is brought by the so-called cost function. Based on its values, the network adapts its internal properties and learns. An example of the learning process, thoroughly described later in this chapter, is depicted in Fig. 4.4.

Cost functions

Initially, the weights and thresholds are defined randomly, or at most heuristically, so at first, as the network processes the input vector. The result may be very distant from the desired true answer. To quantify the the network quality, the so-called cost function $C(w, b)$ is introduced. It gives the measure of distance between the neural network's classification prediction and a correct result[5].

For a simple example of a cost function, a mean squared error

$$C(w,b) = \frac{1}{2n} \sum_{\vec{x}} ||\vec{y}(\vec{x}) - \vec{a}(\vec{x}, w, b)||^2 \tag{4.3}$$

can be taken into account. Here n is the total number of training inputs. Vectors $\vec{a}(\vec{x}, w, b)$ and $\vec{y}(\vec{x})$ are the final output from the network and the desired correct result, respectively, while $\vec{x}$ is given as a training input. The summation is done over all these inputs.

Another, slightly more complex example of a loss[6] function is the cross-entropy function[7]

$$C = 1\frac{1}{n} \sum_{\vec{x}} \left[y \ln a + (1-y) \ln(1-a) \right], \tag{4.4}$$

where the notation stays the same as in case of the mean square error. This, at the first sight peculiarly looking cost function[8] is widely used as it improves learning speed especially in case of parameters very far from optimal setting.

A variation of this cost function, so-called categorical cross-entropy is used for a multi-class classification later in this work's analysis.

---

[5] Say the neural net in Fig. 4.3 classifies fruit as apples $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ or pairs $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and results $\begin{pmatrix} 0.4 \\ 0.1 \end{pmatrix}$ after an apple is given to it. Cost function gives a numerical expression of the result's correctness.

[6] Loss function is sometimes used as a synonym to the cost function.

[7] For a notation simplicity the arguments w, b and parameters $\vec{x}$ are omitted.

[8] It meets the cost function properties such as being non-negative and being 0 in a limit $a(x) \to y$. Note, that the network output should be function with range $\mathcal{R} = [0, 1]$, so not e.g. widely used ReLU.

Learning procedure

As indicated by the form of the cost function, it's variables are w and b, representing all the weights and thresholds in the net, while other inputs such as vectors of results are considered to be parameters. An important consequence of this definition will be clear soon.

The essential problem of the machine learning is to adjust the architecture of a chosen algorithm to perform as good as possible. In the case of neural networks, this optimization is conducted by a successive modification of it's parameters w, b. It is thanks to the aforementioned cost-function definition that this essential problem can be simplified and restated as a mathematically thoroughly described multivariable function minimization.

As mentioned earlier, the initial network-parameter configuration results in random outputs, corresponding to high cost function values. However, the training data set is fed to the network in small batches and after processing each one of them a discrete step[9] in minimizing the cost function is conducted.

This step consists in the evaluation of the cost function[10] and adjustment of the network parameters, so the value of the current cost function decreases. This cost-function decrease naturally means a slight improvement of the network's performance - it slowly learns.

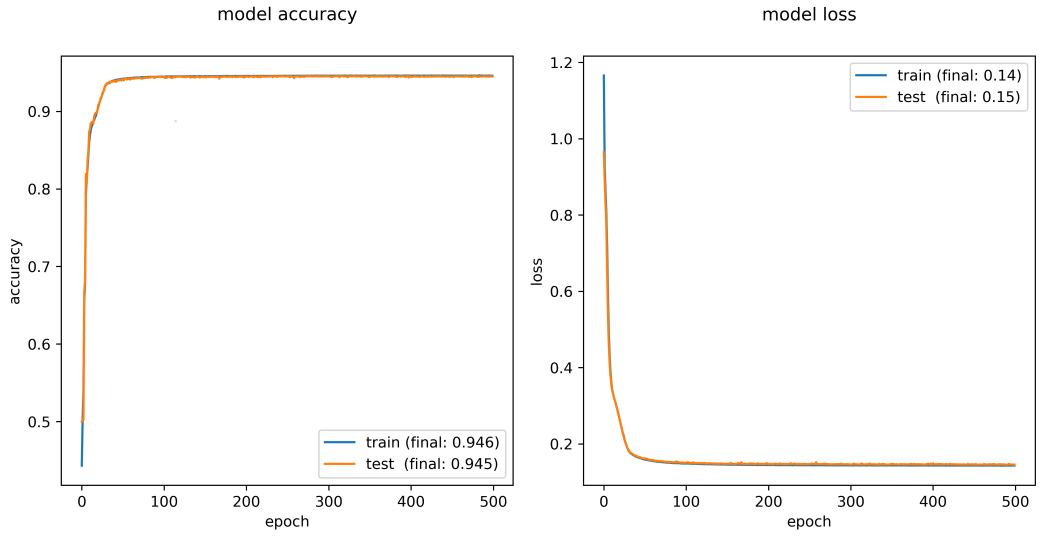

Fig. 4.4: An example of the neural network accuracy[11](left) and the loss function value (right) development through the training procedure. Both are evaluated after every epoch.

[9]Numerical methods such as a stochastic gradient descent are used.

[10]Note that because the cost function is parameterized by the input data, it's exact form varies after processing every batch.

After the whole training dataset is processed, bunch by bunch, a so-called training epoch is finished. The whole training process can contain an arbitrary number of epochs, where after the end of each of them the performance improvement is evaluated. As depicted in Fig. 4.4, the evaluation is carried out for two distinct parts of data.

The training dataset consists of data given to the neural network during training, while the testing dataset contains data that are excluded from the training process, simulating a real situation of unknown data classification. Plots in Fig. 4.4 show the training process of a particular neural network used in this work, concretely the development of the accuracy and the loss function values through training epochs.

Overtraining

The importance of using the testing dataset for a training monitoring is eminent in the case of a so-called overtraining.

It is a situation when the network performance on the training dataset rises, but the testing-data performance suddenly plummets or at least does not keep up anymore. In this case, the network is too adjusted to the training data and can not be trusted to perform well used outside the set. An example of overtraining is given in Fig. 4.5. The example shows a binary classifier, being focused on the training data too much and not being able to predict the data distribution properly.

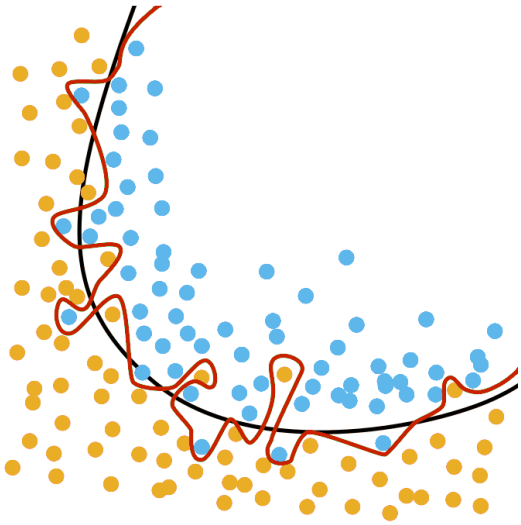

Fig. 4.5: An example of a well trained (black curve) and an overtrained (red curve) binary classifier.

[11] Fraction of correctly classified events over the whole set.

Chapter 5

Data analysis

The main objective of this work is to determine whether there is a machine-learning algorithm that can be utilized for the analysis of the SXRM-detector output.

In the first iteration, a problem of particle classification, that is reconstructing the detected-particle species, was examined. The question stands: is it fundamentally possible to approach this problem with machine learning? Therefore, the available, labelled and physically very clear simulation data could have been used. A simple distortion to the data is presented in a later part of this chapter. A deep neural network depicted in Fig. 5.1 was eventually chosen to be examined.

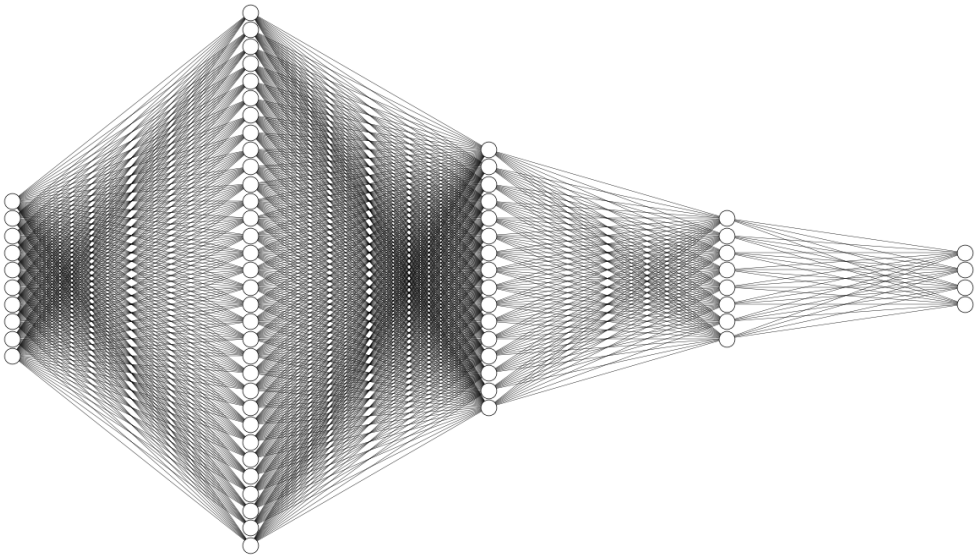

Fig. 5.1: A scheme of the examined deep neural network. It consists of the input layer with 10 neurons, hidden layers with 32, 16 and 8 neurons, respectively, and the output layer with 4 neurons representing classes: electron, proton, $^{4}_{2}$He and $^{56}_{26}$Fe.

5.1 The deep neural network

A deep[1] neural network, a popular algorithm utilizing the analogy with human brain was chosen as the main subject of examination. A structure of the one used for the purposes of this work is depicted in Fig. 5.1.

Structure

Though the input layer in Fig. 5.1 consists of 10 neurons, other neural networks were constructed during the analysis, with the number of neurons in the input layers between 2 and 10. The efficiency of the neural networks was then compared in order to find the optimal number of sensitive SpacePix detectors in the SXRM setup. As mentioned in Chapter 1, the general idea of the analysis is to sample a specific Bragg curve, so the variation in the number of input neurons corresponds to the number of points needed to conduct an efficient sampling.

The structure and number of the hidden layers were chosen more or less randomly, as tuning these so-called hyper variables is an advanced machine-learning topic not necessary to solve this work's problem. Furthermore, a major positive effect of the tuning is not guaranteed.

The task is to classify incoming particles as electrons, protons, $^{4}_{2}$He or $^{56}_{26}$Fe. Therefore, the network's output layer consists of four neurons.

Activations & cost function

In case of all layers except the output one, ReLU was chosen to be the activation function. Just like with the structure of the neural network, the choice of ReLU was based purely on its current popularity among the machine learning community.

The activation of the last layer is softmax[2] - a function that takes a vector of real numbers and provides a vector of probabilities on the output. The output vector meets the probability definition as all the elements are non-negative, less or equal to one, and their sum is equal to one. Especially the second property is important in this case because the categorical cross-entropy[3] cost function is used to provide feedback to the neural network during the training.

For the cost function minimization process the Adam optimizer [12] was used.

[1] As described in Chapter 4, this neural network is denoted as deep, as it has three hidden layers.

[2] The softmax function is defined as $S(a_j) = \frac{e^{a_k}}{\sum_k e^{a_k}}$. For more detailed examination of ReLU and softmax applications see [11].

[3] As mentioned in Chapter 4, the input of the cross-entropy function must be numbers in range $\mathcal{R} = [0, 1]$.

5.2 Used software

Several libraries, toolkits and programming languages were utilized during the whole work. The simulation C++ [13] toolkit Geant4 was presented in Chapter 1, since it was used to generate the input dataset. The popular analysis toolkit ROOT [14] was used to store the data and some of the analyses were also conducted. For the construction and training of the neural networks the Python Keras interface was used.

Keras

Keras [15] is a high-level neural networks application programming interface, written in Python [16] and capable of running on top of TensorFlow [17].

Unlike the rest of the used software, Keras is written in Python, which meant a slight inconvenience because both languages must have been used simultaneously sometimes. The Python code of the deep-neural-network model programmed in Keras reads:

```
model = Sequential()

model.add(Dense(32, input_shape = (d.getX().shape[1], )))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(8))
model.add(Activation('relu'))
model.add(Dense(4))
model.add(Activation('softmax'))

model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
        )
```

Here the *d.getX().shape[1]* term corresponds to the current number of SpacePix layers.

ROOT

Among the high-energy-physics community, the ROOT is a widely used data analysis framework. It was written in C++ programming language at CERN primary as a tool for large data processing.

The data obtained from Geant4 simulations was saved in the TTree structure and exported in the .root format. The main reason is that most of the previous analyses were conducted in the ROOT toolkit. Furthermore, the format is optimized for large data storage, so it is very efficient in space usage.

There is also a Python extension module PyROOT is available providing interaction directly with arbitrary ROOT class. However, as described later, a different data-loading method was utilized.

5.3 Data loading

As pointed out earlier, the input dataset was simulated in Geant4 and stored in a .root format. However, the input data type requested by the Keras sequential model is a NumPy array [18].

NumPy is a fundamental package for scientific computing with Python. The NumPy arrays provide N-dimensional objects standardly used for the representation of numerical data, enabling efficient implementation of numerical computation.

Unfortunately, loading NumPy arrays directly from rootfile[4] with PyROOT was extremely slow in comparison with a .csv file. The first idea was to convert the rootfile into .csv, keep the .csv file and only reconvert it again when changes were made to the simulated data. The .csv file would be then used as a standard source of the training dataset.

An attempt was made to carry this procedure with Python but in comparison with the same program written in C++, the time difference was enormous[5]. Eventually, the loading program was written in Python, checking, whether the .csv file is present. If it is not, the *subprocess* module is used to call a C++ program *root_to_csv* to efficiently create the .csv from a specific rootfile.

```
if path.isfile(file_csv) == 0:
        subprocess.run(
                ['../data/converter/build/root_to_csv',
                particle]
                )
```

As there is a specific rootfile for each particle, the variable *particle* is used to determine, which one to process.

Overall, the data provided as an input to the neural network was a combination of all the simulated particles - altogether $4 \cdot 10^5$ data points. A single data point consists of a 10-element vector and a label, specifying the particle species. Every element of the vector represents a size of energy deposition in a particular detector layer.

[4] Loading from rootfile means processing the TTree structure inside the .root file.

[5] Converting with Python took days, while converting with C++ took several minutes. This is probably not to blame Python but author's personal Python inexperience.

5.4 Clean data analysis

Having the training dataset prepared the deep-neural-network training could be performed. The training length was set to 500 epochs, so the training progress could be observed properly. The whole process was completed in the order of minutes[6].

In the Fig. 5.2, depicting the training progress, two plots are shown. As described in Chapter 4, after finishing an epoch (processing the whole training dataset), the value of the loss function and the accuracy of neural network are calculated. From these values, the graphs[7] are constructed. The two distinct curves appertain to the two sets of data on which the evaluation is conducted. Training data, which are presented during the learning process and training data, only presented for the purpose of progress evaluation.

In the first graph designated as *model accuracy*, the fraction of correctly classified points and the whole dataset is plotted. The second graph shows the evolution of the cost function minimization.

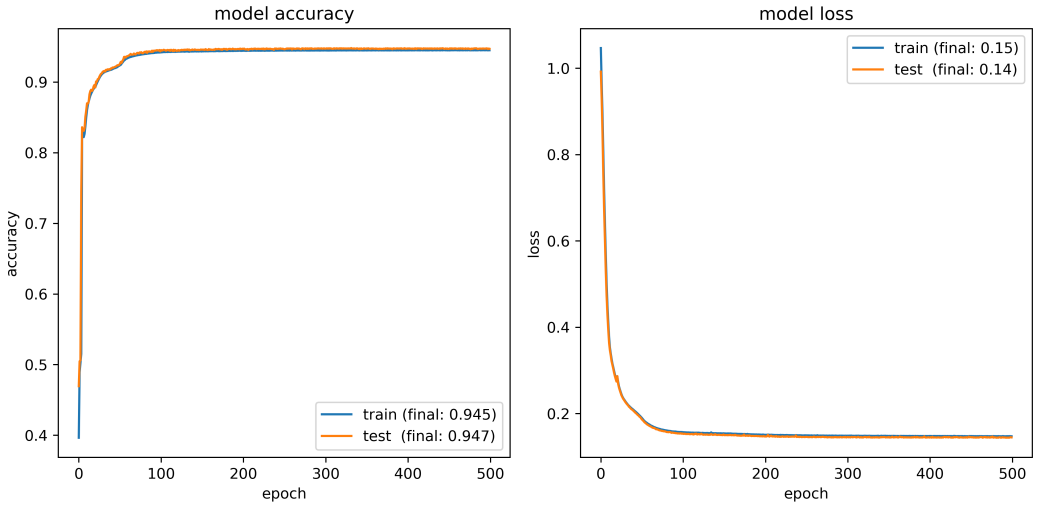

Fig. 5.2: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 2-layer input layer.

From the trend visible in both plots it is apparent, that the major part of the training is done in the first approximately 100 epochs. The testing-data accuracy after 100 epochs is 0.945, so the remaining 400 epochs correspond to an improvement of the performance by just two tenths of a percent.

Even better image about the deep-neural-network performance can be provided by so-called *confusion matrix* shown in the left part of Fig. 5.3.

[6] One training took approximately 5 minutes.

[7] There is no fitting conducted, the lines just connect neighbouring data point.

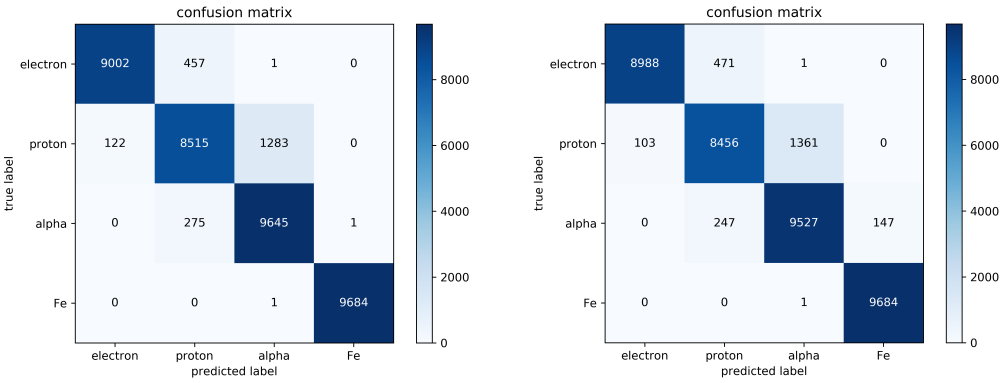

Fig. 5.3: Confusion matrices for a neural network with 2-neuron input layer. Left for clean data, right for distorted data.

The rows of the confusion matrix correspond classes as they are assigned to the data points by labels while the classes in columns are those assigned by the neural network. In an ideal case, where the network makes no mistake, the matrix is diagonal. The advantage of the confusion matrix format is that the non-diagonal matrix elements carry not only information about an error but also an exact error specification, e.g. here a proton was falsely classified as an electron 122 times.

The so far shown figures only describe the deep neural networks with 2-neuron input layer. The rest of plots and confusion matrices for deep neural networks with 4, 6, 8 and 10 neurons in the input layer is shown in attachments. Simply by observing the final model accuracy mentioned in the right bottom corners of the accuracy plots one can conclude, that 6 layers are the highest reasonable number. The networks with higher number of input neurons reached the same 99.2% accuracy as the one with 6 input neurons.

At this point, the question remains, whether the higher number of input neurons plays a role when processing distorted data or if higher accuracy can be reached with a different SXRM layout, e.g. if the absorber thickness was increasing for later sensitive layers.

To reach a final decision for the problem of SpacePix-layer optimization the information about possible reconstruction accuracy is only one of many. Eventually, the most important will restrictions on the physical detector parameters[8] such as total weight, dimensions, or energy consumption. However, these data are important to reach an optimal final decision.

Ultimately, based on the resulting accuracies it has been shown, that deep neural networks can be utilized for the analysis of the SXRM output data. Even at the simplest 2-layer setup, the 94% accuracy can be reached when reconstructing the clean simulation data.

[8]Since it is supposed to operate in space.

5.5 Distorted data analysis

Another step in the examination of the deep-neural-network suitability was an examination of a slightly distorted dataset. This distortion consists in the non-linear relation between the deposited energy (the eralier processed clean data) and the voltage induced in the SpacePix ASIC[9].

Data points to reconstruct this non-linear relation was provided by the designers of the SpacePix detector. A polynomial fit was carried out to obtain a MeV to V conversion function as depicted in Fig. 5.4.

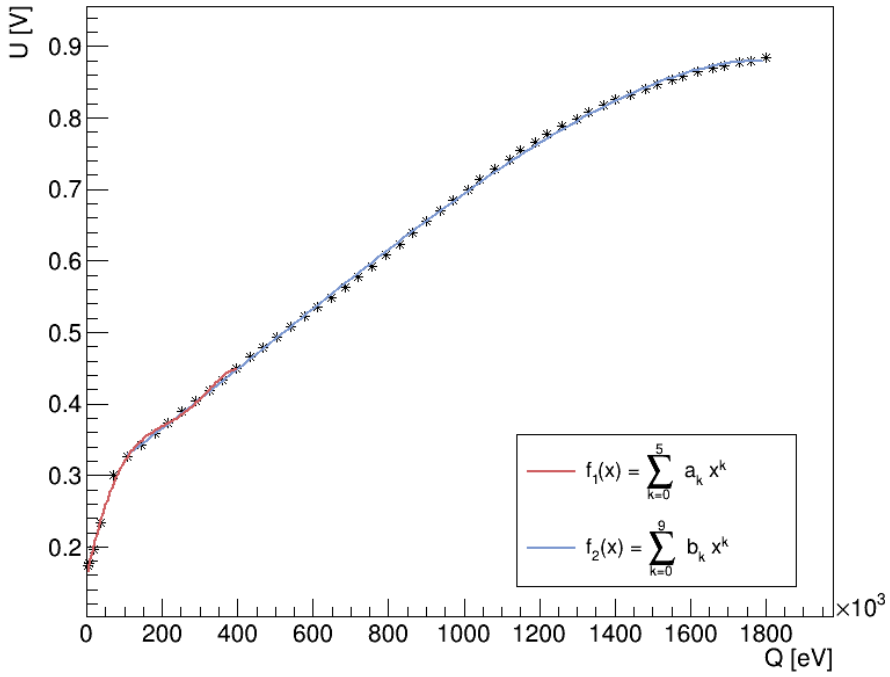

Fig. 5.4: A fit to obtain a conversion function between the deposited energy and the induced voltage.

The fit was calculated in two areas. In the interval $(0, 400)$ keV a 5-th order polynomial was used, in the interval $(100, 1800)$ keV it was a 9-th order polynomial. The coefficient values were obtained from a ROOT output as follows:

NAME	VALUE	ERROR	NAME	VALUE	ERROR
a0	1.61248e−01	5.19311e−01	b0	2.86981e−01	4.03533e−01
a1	2.35266e−06	8.25676e−06	b1	3.78436e−07	9.18117e−07
a2	−7.66155e−12	4.07539e−11	b2	8.99024e−14	7.90285e−13
a3	−8.69674e−18	1.25056e−16	b3	−5.99043e−20	5.31716e−19
a4	9.69941e−23	3.45899e−22	b4	−1.88494e−27	3.34167e−25
a5	−1.32334e−28	6.93149e−28	b5	1.17779e−33	2.05038e−31
			b6	3.34037e−40	1.22748e−37
			b7	−3.82041e−46	7.09607e−44
			b8	−1.10112e−52	3.90795e−50
			b9	−4.16866e−59	2.00007e−56

[9] Application-Specific Integrated Circuit

Using this conversion function[10] the initially clean dataset was distorted and fed as an input to the deep neural network.

The resulting training plots are shown in Fig. 5.5. Though the data was converted with a non-linear function, the accuracy decrease was only 0.5%. Considering, there will be only one more similar non-linear conversion to the final data obtained from SXRM, this result supports the usability of deep neural networks.

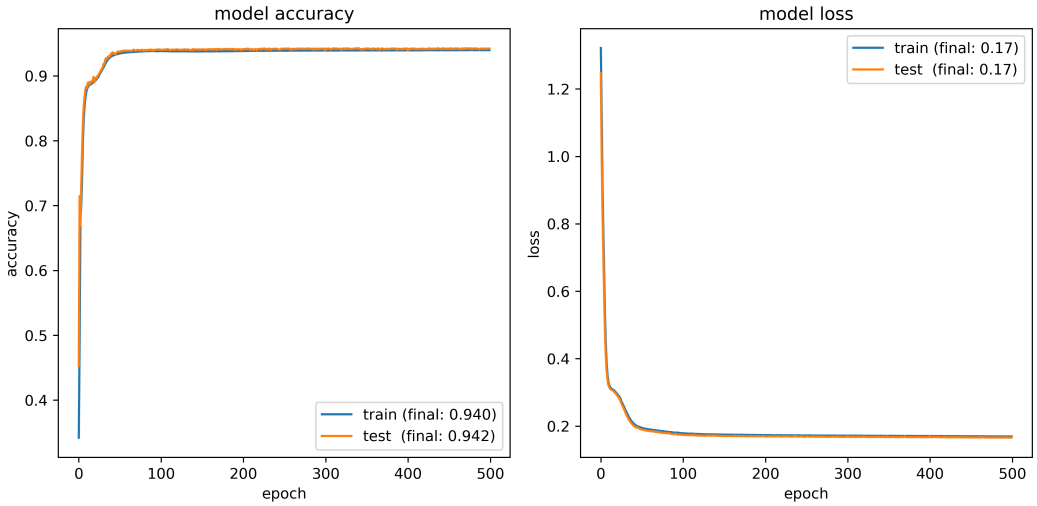

Fig. 5.5: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 2-layer input layer.

As seen in the plots (Fig. 14, Fig. 15, Fig. 16, Fig. 17) in attachments, the optimal number of input neurons is not 6 anymore for the distorted dataset. The accuracy still slowly rises with the number of SpacePix layers.

The confusion matrix for the distorted data is shown in Fig. 5.3.

[10]The conversion function f_1 was used for energy depositions from 0 to 115 keV, f_2 was used for the rest.

Conclusions

This work aims to examine the possibility of machine-learning methods utilization for the analysis of the SXRM detector output. As the main objective was to determine whether a particular method can be theoretically used, a real output data was not used during the work. The more accessible and physically-clean simulation data was used instead.

The primary aim of the analysis was to reconstruct the detected particle species. Based on the deployment environment analysis, four particle types were used: electrons, protons, and ions $^{4}_{2}$He and $^{56}_{26}$Fe, with energy spectra mentioned in Tab. 1.1. Based on the reconstruction quality, the optimization of the SXRM-detector geometry, namely the number of sensitive SpacePix layers, was conducted.

In Chapter 1 the concept of the multilayer SXRM detector was presented. Firstly, the deployment environment was described together with the radiation sources, namely Van Allen belts, galactic cosmic rays, and solar particle events. Subsequently, two main SXRM models were described, the first currently orbiting the Earth and the second being a subject to simulations for further development. The SpacePix chip used as an sensitive element in the SXRM setup was also presented.

The input training dataset for the particular machine-learning method was obtained by simulations of the SXRM detector. This procedure, together with a thorough characterization of the exploited Geant4 software, is described in Chapter 2.

A brief introduction to the field of machine learning is provided in Chapter 3. The fundamental paradigm is presented and the main branches of methods are described. A deep neural network algorithm is selected for an examination of the suitability for the SXRM-output-analysis.

A further examination of the neural networks was conducted in Chapter 4. This includes a thorough description of the algorithm architecture, with an explanation of the purpose of every particular component. The neural-network training is described and basic utilized mathematical concepts are presented. The concept of overtraining is also briefly explained.

In Chapter 5 the examination of the chosen deep neural network was described. Initially, the architecture and specifications of the particular network were pre-

sented. The software used during the work, such as Keras and ROOT, was also briefly described.

The results of the neural-network training for a clean dataset were presented, including the plots depicting the training progress, and confusion matrices, thoroughly describing the final network precision.

Afterwards, a simple distortion to the dataset was presented, based on the hardware specifications of the SpacePix detector. The physically pure data was converted to a voltage signal using a non-linear relation. A new training of the neural network followed. The results suggested, that this kind of a non-linear distortion does not have a significant effect on the particle-reconstruction quality.

Based on the results, the theoretical optimal number of SpacePix layers in the SXRM layout was designated to 6. However, other effects, mainly the physical geometry requirements, such as the total detector weight, will have to be taken into account.

The future aim of the research will consist of a more-realistic data analysis. This means mainly the currently unavailable conversion of the simulated data to the final obtained signal and including spurious effects like plasma effect, charge diffusion, or the role of noisy pixels. Another step to be made is to exploit the advantages of the pixel character of the SpacePix detector and use convolutional neural networks[11] for a more complex data analysis.

[11] The convolution neural networks are currently mostly used for the image recognition.

Attachments

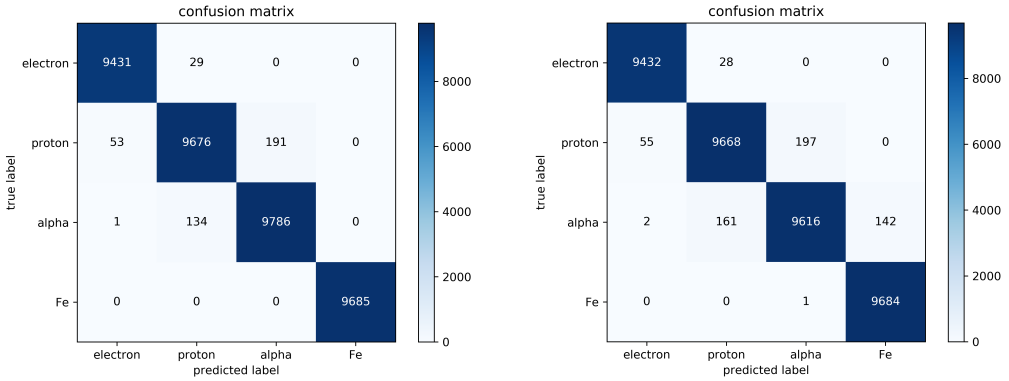

Fig. 6: Confusion matrices for a neural network with 4-neuron input layer. Left for clean data, right for distorted data.

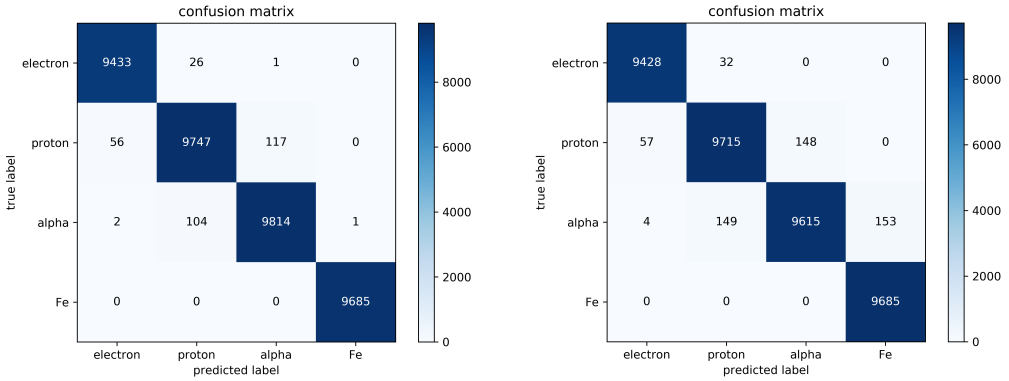

Fig. 7: Confusion matrices for a neural network with 6-neuron input layer. Left for clean data, right for distorted data.

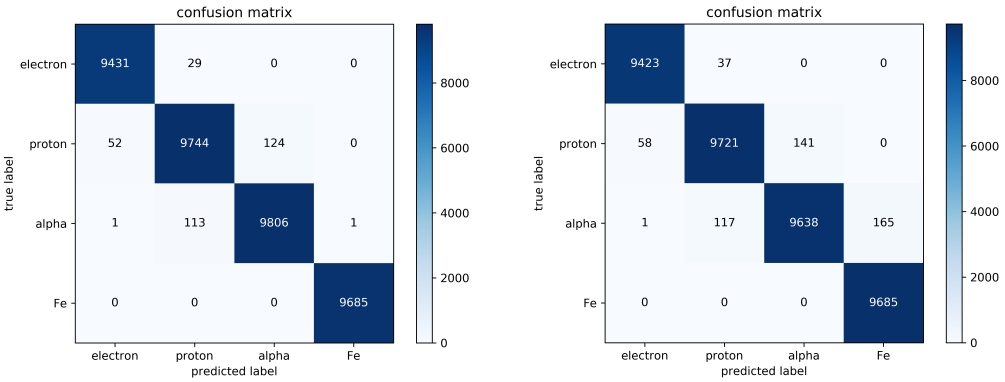

Fig. 8: Confusion matrices for a neural network with 8-neuron input layer. Left for clean data, right for distorted data.

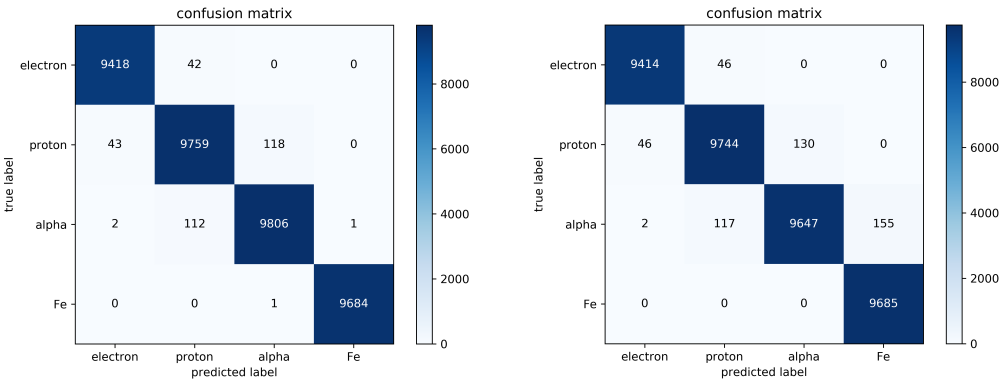

Fig. 9: Confusion matrices for a neural network with 10-neuron input layer. Left for clean data, right for distorted data.

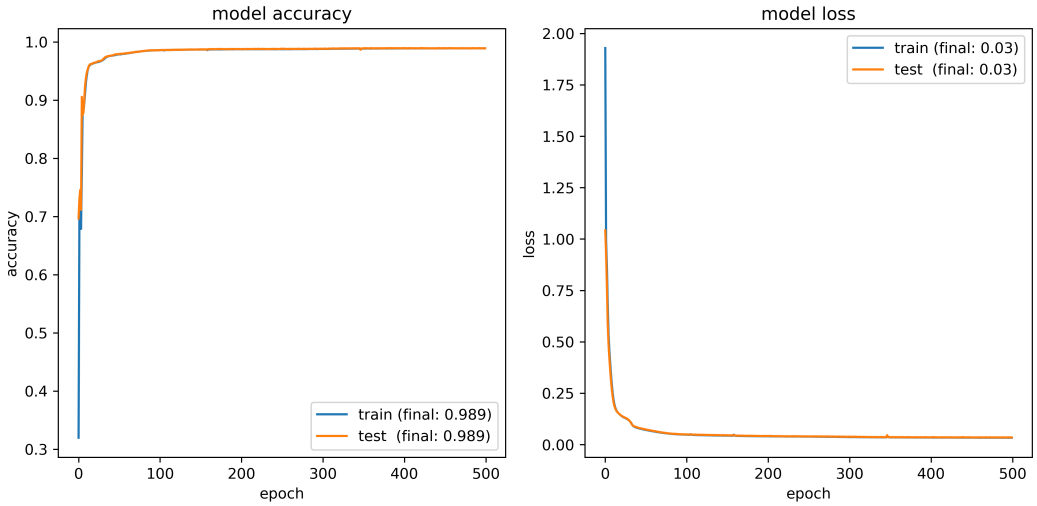

Fig. 10: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 4-neuron input layer processing clean data.

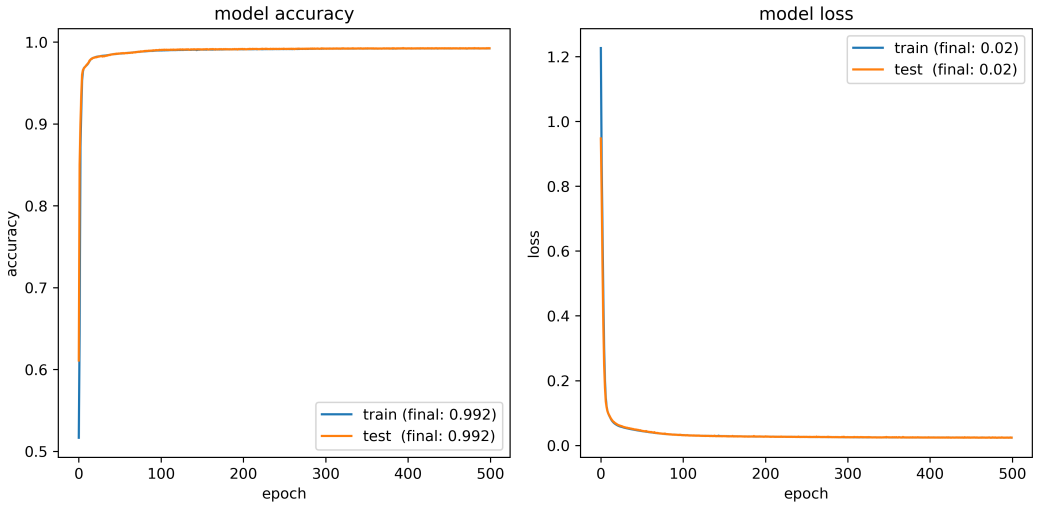

Fig. 11: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 6-neuron input layer processing clean data.

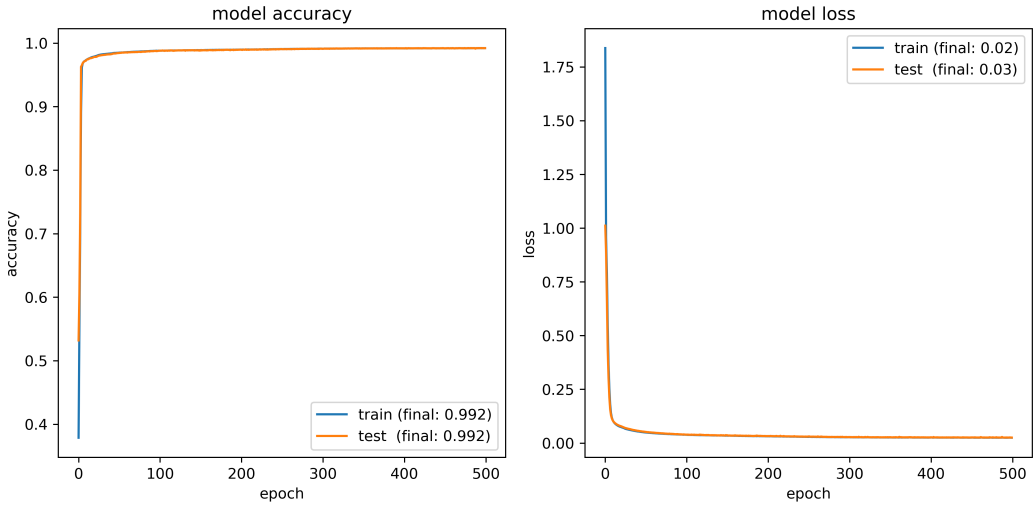

Fig. 12: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 8-neuron input layer processing clean data.

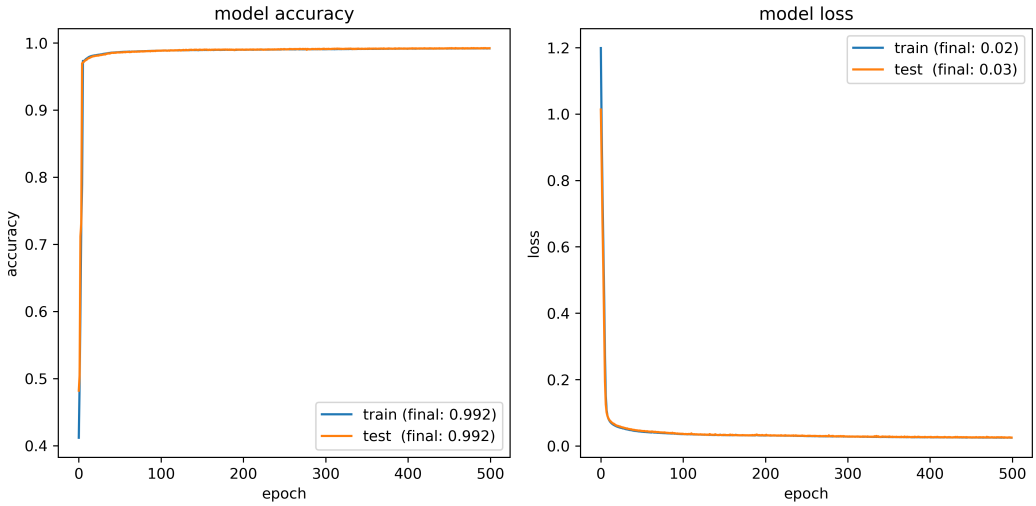

Fig. 13: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 10-neuron input layer processing clean data.

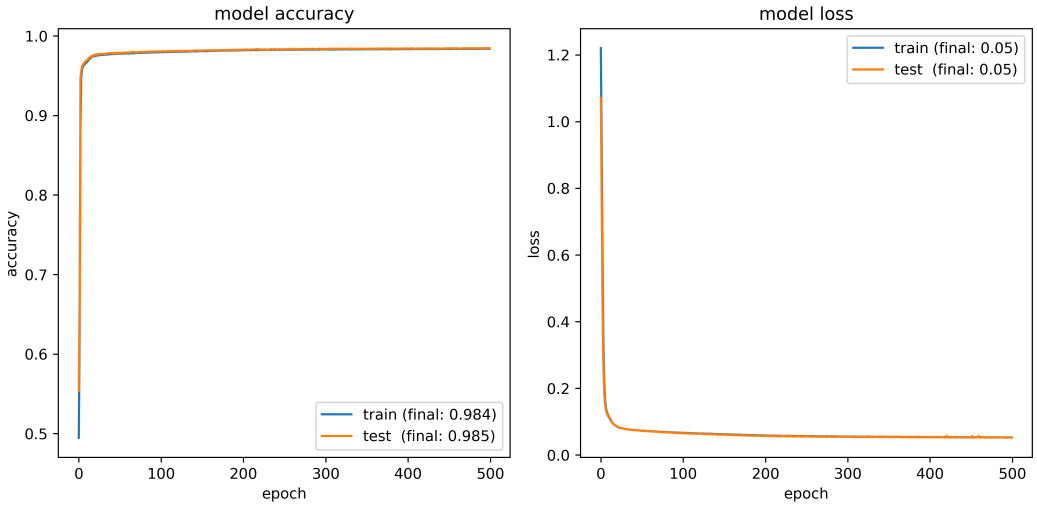

Fig. 14: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 4-neuron input layer processing distorted data.

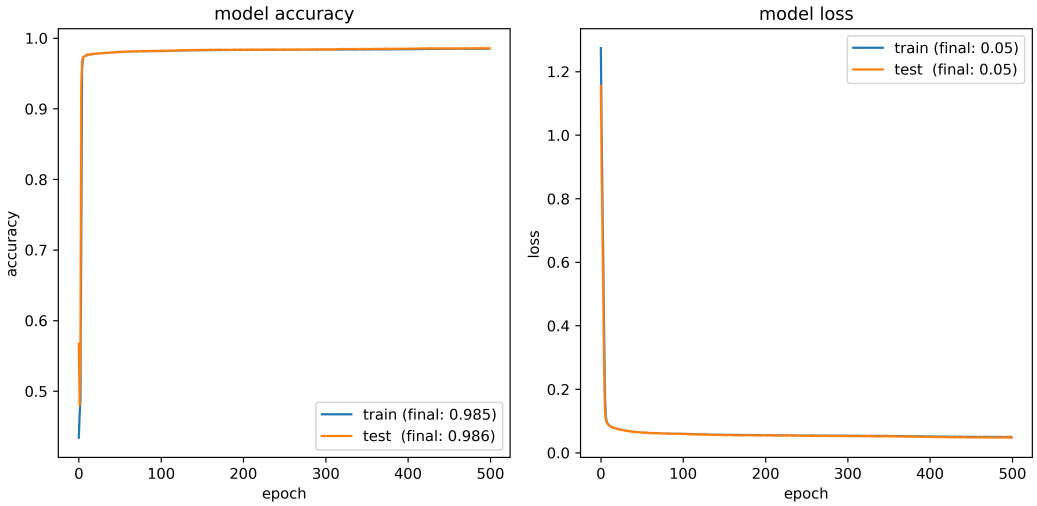

Fig. 15: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 6-neuron input layer processing distorted data.

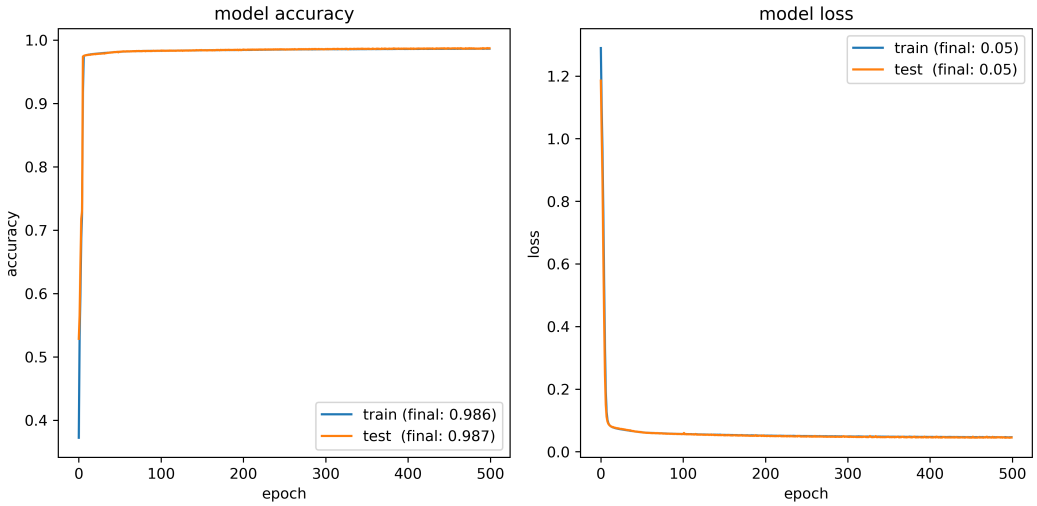

Fig. 16: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 8-neuron input layer processing distorted data.

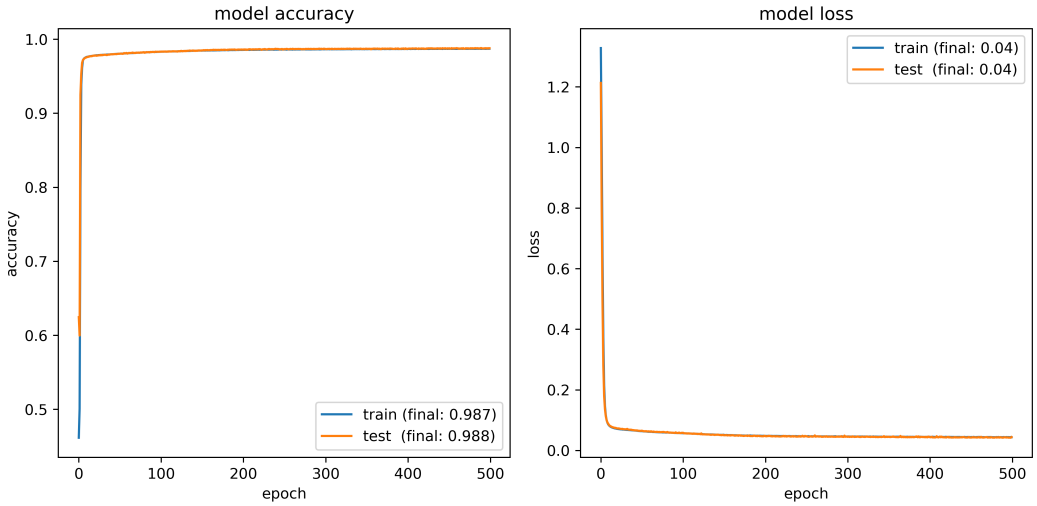

Fig. 17: The evolution of the neural-network accuracy and the loss function during training. Plots for a neural network with 10-neuron input layer processing distorted data.

Bibliography

[1] Matěj Vaculčiak. "Simulace odezvy polovodičových pixelových detektorů". Bakalářská Práce. České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, 2018.

[2] Amos R. Omondi and Jagath C. Rajapakse. *FPGA Implementations of Neural Networks*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 1441939423, 9781441939425.

[3] Johnson Space Center Space Radiation Analysis Group. *What is space radiation*. 2016. URL: https://srag.jsc.nasa.gov/spaceradiation/What/What.cfm.

[4] Heynderickx D. et al. "New radiation environment and effects models in the European Space Agency's Space Environment Information System (SPENVIS)". In: *Space Weather* 2.10 (). DOI: 10.1029/2004SW000073. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2004SW000073. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2004SW000073.

[5] M. Havranek et al. "X-CHIP-03 – SoI MAPS sensor with hit counting and ADC mode". In: *manuscript in preparation* ().

[6] Sea Agostinelli et al. "GEANT4—a simulation toolkit". In: *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303.

[7] Marčišovská M. et al. "Detection microelectronic IC for orbital measurement of cosmic radiation [Functional Sample]". In: (2018).

[8] Sunanda Banerjee and. "Validation of Physics Models of Geant4 using Data from CMS Experiment". In: *Journal of Physics: Conference Series* 898 (2017), p. 042005. DOI: 10.1088/1742-6596/898/4/042005. URL: https://doi.org/10.1088%2F1742-6596%2F898%2F4%2F042005.

[9] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to Machine Learning*. Springer, 2019. ISBN: 978-3-030-15729-6. URL: https://www.amazon.com/Introduction-Machine-Learning-Gopinath-Rebala-ebook/dp/B07RK2267F?SubscriptionId=

AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B07RK2267F.

[10] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com.

[11] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. cite arxiv:1803.08375Comment: 7 pages, 11 figures, 9 tables. 2018. URL: http://arxiv.org/abs/1803.08375.

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[13] Nicolai M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. 2nd. Addison-Wesley Professional, 2012. ISBN: 0321623215, 9780321623218.

[14] I. Antcheva et al. "ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization". In: *Computer Physics Communications* 180.12 (2009). 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures, pp. 2499 –2512. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2009.08.005. URL: http://www.sciencedirect.com/science/article/pii/S0010465509002550.

[15] François Chollet et al. *Keras*. https://keras.io. 2015.

[16] Guido Rossum. *Python Reference Manual*. Tech. rep. Amsterdam, The Netherlands, The Netherlands, 1995.

[17] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[18] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science and Engg*. 13.2 (Mar. 2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37. URL: https://doi.org/10.1109/MCSE.2011.37.