

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství



Identifikace svazkových částic pomocí
detektorů CEDAR a strojového
učení na experimentu COMPASS v
laboratoři CERN

Identification of Beam Particles
Using CEDAR Detectors and
Machine Learning in the COMPASS
Experiment at CERN

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. František Voldřich
Vedoucí práce: Ing. Martin Zemko
Rok: 2022

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Akademický rok 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. František Voldřich
Studijní program:	Aplikace přírodních věd
Obor:	Aplikace softwarového inženýrství
Název práce česky:	Identifikace svazkových částic pomocí detektorů CEDAR a strojového učení na experimentu COMPASS v laboratoři CERN
Název práce anglicky:	Identification of Beam Particles Using CEDAR Detectors and Machine Learning in the COMPASS Experiment at CERN

Pokyny pro vypracování:

1. Seznamte se se softwarem a metodami používanými pro analýzu dat na experimentu COMPASS
2. Seznamte se s fyzikálními principy činnosti detektoru CEDAR
3. Prozkoumejte různé přístupy a konfigurace strojového učení pro identifikaci částic svazku
4. Implementujte vhodné metody strojového učení pro identifikaci částic a optimalizujte je na fyzikálních datech
5. Navržené řešení otestujte a integrujte do stávajícího systému pro analýzu fyzikálních dat

Doporučená literatura:

- [1] ABBON, P., et al. The COMPASS Setup for Physics with Hadron Beams. In: *Nuclear Instruments and Methods in Physics Research, A: Accelerators, Spectrometers, Detectors and Associated Equipment*. Elsevier, 2014, pp. 69–115.
- [2] AGGARWAL, C. C. *Neural Networks and Deep Learning: A Textbook*. Springer, 2019. ISBN 978-331-9944-630.
- [3] BICKER, K., et al. CEDAR PID using the Likelihood Approach for the Hadron-Beam. COMPASS Note, 2017.
- [4] GERASSIMOV, S., et al. PHAST - PHysics Analysis Software Tools (software package for physics analysis). 2020.
- [5] CHOLLET, F. *Deep Learning with Python*. New York: Manning Publications, 2017. ISBN 978-161-7294-433.

Jméno a pracoviště vedoucího práce:


Ing. Martin Zemko

Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

Jméno a pracoviště konzultanta:

Dr. Marcin Stolarski

Laboratory of Instrumentation and Experimental Particle Physics, Lisabon, Portugalsko


.....
vedoucí práce

Datum zadání diplomové práce: 15. 10. 2021

Termín odevzdání diplomové práce: 2. 5. 2022

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

.....
děkan

V Praze dne 15. 10. 2021

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v příloženém seznamu.

Declaration

I declare that I have carried out this diploma thesis myself and I have mentioned all used information sources in bibliography.

V Praze dne

.....
Bc. František Voldřich

Poděkování

Chtěl bych poděkovat Ing. Martinovi Zemkovi za vedení mé práce a podnětné připomínky při jejím zpracování. Současně děkuji Dr. Marcinu Stolarskimu za poskytování zpětné vazby a cenné rady.

Acknowledgment

I would like to thank my supervisor Ing. Martin Zemko for supervising my diploma thesis and his incentive notes during its processing. I would also like to thank Dr. Marcin Stolarski for providing me with feedback and valuable advices.

Bc. František Voldřich

Název práce:

Identifikace svazkových částic pomocí detektorů CEDAR a strojového učení na experimentu COMPASS v laboratoři CERN

Autor: Bc. František Voldřich

Studijní program: Aplikace přírodních věd

Obor: Aplikace softwarového inženýrství

Druh práce: Diplomová práce

Vedoucí práce: Ing. Martin Zemko

Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Konzultant: Dr. Marcin Stolarski

Laboratory of Instrumentation and Experimental Particle Physics, Lisabon, Portugalsko

Abstrakt: Cílem této práce je seznámit se se softwarem a metodami využívanými pro sběr a analýzu dat na experimentu COMPASS v laboratoři CERN, pochopení principu detektorů CEDAR a prozkoumání možností strojového učení. Tyto poznatky mají být dále využity k vybrání vhodné metody strojového učení pro identifikaci částic na základě detektorů CEDAR, její konfigurace a implementace. Řešení má být otestováno a integrováno do stávajícího softwaru pro analýzu fyzikálních dat, aby mohlo být využito při analýze dat z detektorů CEDAR naměřených v roce 2018.

Klíčová slova: CERN, COMPASS, CEDAR, PHAST, Neuronové sítě

Title:

Identification of Beam Particles Using CEDAR Detectors and Machine Learning in the COMPASS Experiment at CERN

Author: Bc. František Voldřich

Abstract: The goal of this thesis is to get familiar with software and methods used for data acquisition and analysis in COMPASS experiment at CERN laboratory, the principles of CEDAR detectors and to investigate methods of machine learning. This knowledge is to be further utilized to select eligible machine learning method for particle identification based on the CEDAR detectors, its configuration and implementation. The solution should be tested and integrated back into the current software for physics data analysis so that it can be used for CEDAR data analysis for the 2018 data.

Key words: CERN, COMPASS, CEDAR, PHAST, Neural Networks

Contents

Introduction	13
1 COMPASS Data Analysis Software	15
1.1 COMPASS experiment	15
1.1.1 AMBER	15
1.2 DAQ and CORAL	16
1.2.1 DAQ hardware structure	16
1.3 Other CERN services	17
1.3.1 CERN CentOS	17
1.3.2 Linux Public Login User Service	17
1.3.3 CERN Batch Service	18
1.4 ROOT	19
1.4.1 mDST	19
1.5 PHAST	20
1.5.1 PHAST Event Display	21
1.6 TensorFlow	21
1.6.1 Keras	21
1.6.2 Frugally-deep	22
2 CEDAR detectors	23
2.1 Detector principle	23
2.2 Problems using CEDAR information	25
2.2.1 Likelihood method	26
3 Machine Learning approach	29
3.1 Classification	29
3.2 Artificial neural networks	30
3.2.1 Activation functions	31
3.2.2 Cost functions	33
3.2.3 Optimizers	35
3.2.4 Types of neural networks	37
4 New methods for beam particle identification	41
4.1 Problems with the 2018 data taking	41
4.2 Using neural networks for classification	44
4.2.1 Method 1: NN as a direct classifier	44
4.2.2 Method 2: NN as a PMTs response predictor	44
4.2.3 Method 3: NN as PMTs pattern predictor	45
4.3 Training data	46
4.3.1 Dataset types	46

4.3.2	Measured data	47
4.3.3	Monte Carlo simulations	50
5	Implementation	53
5.1	Datasets preparation	53
5.2	New methods for particle identification	54
5.2.1	Neural networks implementation	55
5.2.2	Method 1: NN as a direct classifier	59
5.2.3	Method 2: NN as PMTs response predictor	60
5.2.4	Method 3: NN as PMTs pattern predictor	60
5.3	Analyses	60
5.4	Meta parameters optimization	64
5.4.1	Running the algorithm	65
5.4.2	Network type selection	67
5.5	Comprehension of gradual results	68
5.5.1	Estimating efficiency	68
5.5.2	Methods analysis	74
5.5.3	Hybrid method	78
5.5.4	Dataset size	79
5.5.5	MC files analyses	80
5.5.6	Adding second track	83
5.5.7	Adding PMT pads to MC	84
5.5.8	Adding PMT pads to measured data	85
5.6	Integration to PHAST	86
5.7	Future development	89
5.7.1	Improving kaon proxy	89
5.7.2	Application for AMBER	90
5.7.3	Using the prediction as probability	90
	Conclusion	91
	Bibliography	92
	Appendices	99
A	Additional plots	99
A.1	Using approximations of beam angles	99
A.2	Confusion matrices	100
A.3	Model output vs. angle	101
A.3.1	Method 1 - Measured data	101
A.3.2	Method 1 - Monte Carlo simulations	105
A.3.3	Method 2 - Measured data	107
A.3.4	Method 2 - Monte Carlo simulations	111
A.4	Method 3 angle histograms	112
B	CD contents	117

Introduction

Hadron beams used in the COMPASS experiment are a mixture of particles, e.g. a 190 GeV negative hadron beam contains about 97 % of π^- , 2.5% of K^- and $< 1\%$ of \bar{p} . CEDAR detectors, based on Cerenkov effect, in the COMPASS experiment beamline were designed to identify a particles in limited intensity beams with a divergence below $65 \mu\text{rad}$. For this purpose two alike CEDARs are positioned upstream the target. However, during the 2018 data taking, a beam with 15 times higher intensity was used. CEDARs were prepared to withstand such conditions by a major upgrade of the frontend electronics and photomultipliers as well as a redesign of the corresponding firmware. In addition, the beam divergence of those runs was up to $300 \mu\text{rad}$ with only 10-15 % of particles being within the designed divergence range. Hence, the previous method of data analysis using the likelihood approach cannot be used in this case. [1][36][42]

The goal of this thesis is to investigate different machine learning approaches and their usage. Upon selecting a suitable approach, a new machine learning based method for beam particle identification in COMPASS using data from CEDARs should be implemented and configured. The solution should be then tested and integrated back into the current software for the data analysis written in C++.

The first chapter briefly describes the COMPASS experiment, the software used for its data analysis and other utilized software tools.

Chapter 2 deals with the principles of CEDAR detectors and previous methods of data analysis.

The next chapter introduces classification problem and its possible solution using machine learning, or more precisely artificial neural networks. Different types and architectures of networks are presented.

In chapter 4, some alternatives of a new method for beam particle identification together with motivation to develop a new one are introduced. Subsequently, a procedure for obtaining training data is outlined.

The last chapter describes the implementation of aforementioned methods and tools for its analysis. A genetic algorithm based approach for meta parameters optimization is described, the new methods and types of networks are compared and the best performing model is selected. A procedure for integration of a trained network into the current data analysis software is developed. On top of that, multiple datasets are examined and the findings are presented.

Chapter 1

COMPASS Data Analysis Software

1.1 COMPASS experiment

COMPASS or Common Muon Proton Apparatus for Structure and Spectroscopy is a high-energy physics experiment with fixed target. The experimental hall is situated on a beamline of the Super Proton Synchrotron (SPS) particle accelerator, which is located at the CERN laboratory's North area in Geneva, Switzerland. The goal of the experiment is the study of hadron structure and spectroscopy using high intensity muon and hadron beams. In 2012, the original experiment was replaced by COMPASS II focusing on the Deeply Virtual Compton scattering, Hard Exclusive Meson Production, Semi-inclusive Deeply Inelastic Scattering, polarized Drell-Yan processes and Primakoff reactions. Around 200 physicists from 13 different countries and 25 institutions work on the experiment. COMPASS II ended in 2022 and will be replaced by AMBER or Apparatus for Meson and Baryon Experimental Research. [1][46][49][51]

1.1.1 AMBER

COMPASS evolution into the AMBER will consist of upgrades of the existing components as well as installation of new detectors, targets and updates in the read-out technology. AMBER will focus on three main goals, which are:

1. An independent precision determination of the electric mean-square charge radius of the proton,
2. Drell-Yan and J/ψ production experiments using the conventional M2 pion beam,
3. measurement of proton-induced antiproton production cross sections.

To achieve these goals, beams consisting of muons, pions, kaons and (anti-)protons impinged on liquid hydrogen, helium-4 or nuclear targets will be used. The beam

composition is important especially for the separation as discussed in chapter 5.7.2. The outcome of this work is expected to be applicable for the AMBER experiment as well. [22][46]

1.2 DAQ and CORAL

The COMPASS DAQ is a hybrid system consisting of hardware and software parts for reading out analog signals directly from experiments' detectors, converting them to digital values and saving them. The current system replaced the previous one in 2014 and has been in use since then. [1][4][26][38][40]

Data taking is separated into measurements of a single physics event. An event represents a single collision of particles (or a collision of a particle and a fixed target as is the case of the COMPASS experiment) and its trajectories. This data taking process is controlled by Trigger Control System (TCS) that discards uninteresting data based on predefined criteria in order to lower the data volume. This is important because the detectors produce around 400 MB/s of data, which yearly accounts for around 1 PB of data or equivalent of 10^{15} bytes. [1][4][26][26]

Since individual detectors record only subevents (i.e. partial information about the trajectory of a particle in the detector), the next step is to assemble those subevents into events. A C++ program called CORAL is used for this task. Resulting files are saved to CASTOR (CERN Advanced STORAge manager) in ROOT trees called mini Data Summary Trees (mDST) as elaborated in chapter 1.4.1. [1]

1.2.1 DAQ hardware structure

DAQ contains several layers of FPGAs¹ modules that process information recorded by detectors and build the physics events. The first layer consists of front-end cards that readout analog data from the detectors and convert them to digital form. There are around 300 000 data channels coming from the detectors. [4][26][40]

In the next step, data flow into three layers of multiplexers that combine several inputs into a single output data stream. Effectively, this means sorting the data into subevents. The first two layers consist of HGeSiCA, CATCH and GANDALF modules, Slink multiplexers and TIGER VXS data concentrators. There are 8 FPGA cards (referred to as Data Handling Cards or DHC) in the next layer. [4][26][40]

The next layer is a single FPGA module with switch firmware that handles event building. Data of the same events recorded by different detectors and transmitted through different channels are grouped together. The last layer consists of readout computers with DAQ software. Data are readout and temporarily saved to local hard drives before moving to CASTOR. The scheme of this structure can be seen in fig. 1.1. [4][26][40]

¹FPGA or field programmable gate array chips are special integrated circuits whose behavior can be programmed. [3]

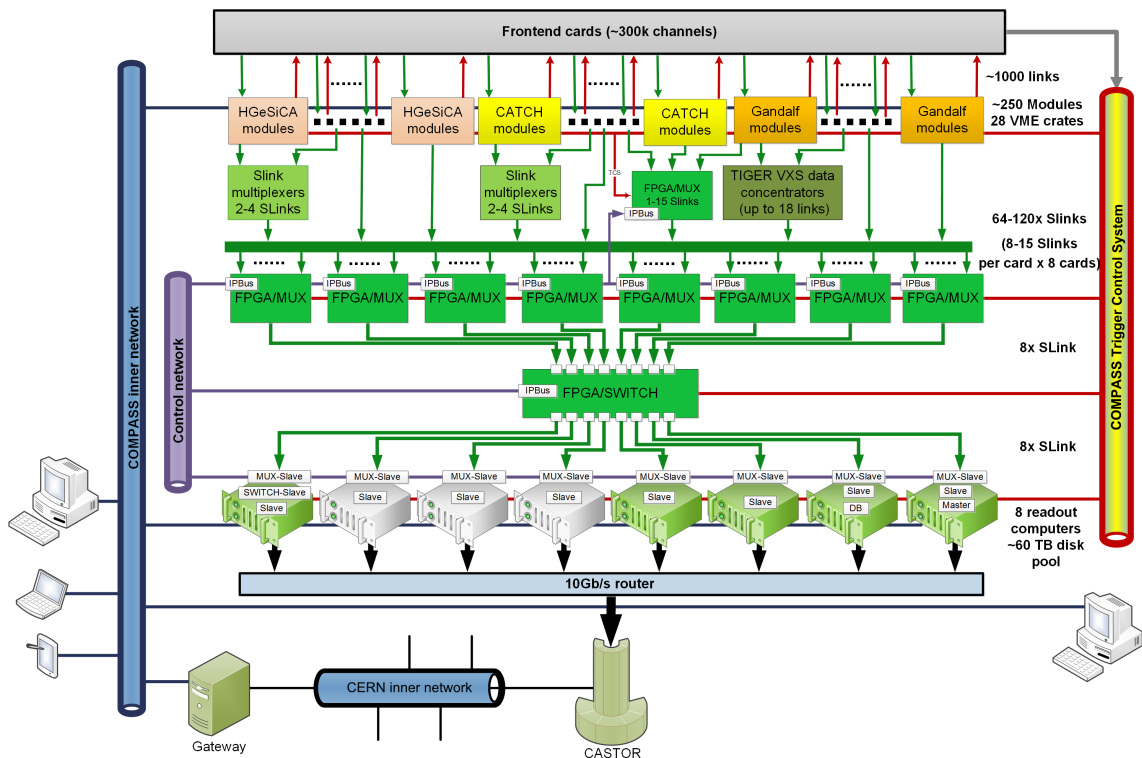


Figure 1.1: The structure of the DAQ hardware. [26]

1.3 Other CERN services

Some software and hardware provided by the IT Department of CERN will be utilized.

1.3.1 CERN CentOS

CERN CentOS is CERN customized distribution that is built on top of the CentOS Core and is tailored to integrate within the CERN computing environment. [50]

CERN CentOS is fully compatible with CentOS Core, therefore with the (Red Hat) Enterprise Linux: all software used or built on one of the versions should function properly on any other version. [50]

For this work, CERN CentOS 7.1 will be used for any local development.

1.3.2 Linux Public Login User Service

LXPLUS is the interactive logon service to Linux for all CERN users. The cluster consist of public machines running Linux (mainly CERN CentOS 7) in 64 bit mode for interactive work. One can access LXPLUS using SSH protocol. It grants access to public services, namely to:

- networked file storage for CERN users **AFS**
- various versions of numerous compilers and interpreters: GCC, G++, Python, Ruby, Perl etc.
- the CERN Mail Server
- CERN Print System
- CERN storage center **EOS**
- CASTOR
- the CERN batch system (**HTCondor**)

[62]

1.3.3 CERN Batch Service

CERN Batch Service is based on HTCondor, an open-source High-Throughput Computing (HTC) software framework for distributed parallelization of computationally intensive tasks.² It allows users to queue up jobs in the system and maximize the utilisation of the batch farm of around 100 000 cores with respect to a fair-shared system. [47][54]

The batch service also offers use of GPUs³, mainly NVidia Tesla T4 with CUDA Toolkit extension. This is especially beneficial for neural networks training as it comes down to matrix multiplication for gradient computations (see [41]), where GPUs perform very well thanks to parallel computing. [47]

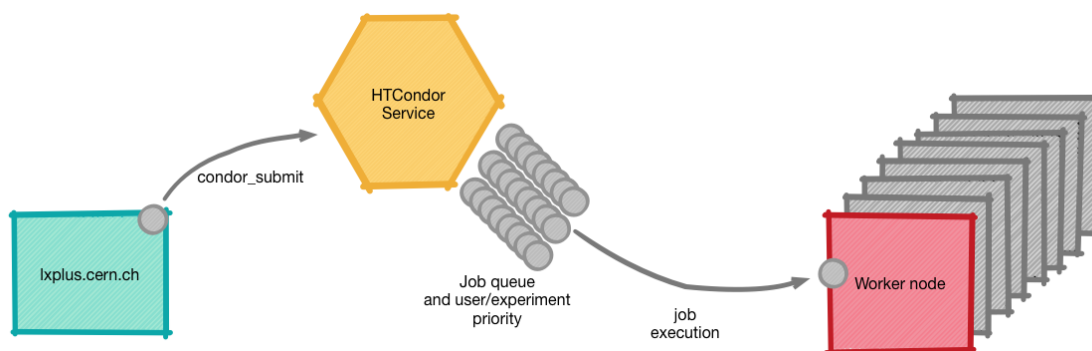


Figure 1.2: Schematic view of the CERN HTCondor service. [48]

²HTCondor uses computing power of machines connected over a network by effectively harnessing shared resources with distributed ownership. [54]

³Graphics processing units.

1.4 ROOT

ROOT is an object oriented framework for data processing and analysis developed at CERN. It was designed especially for data analysis in particle physics experiments, thus it is optimized to handle large amount of data in the most efficient way. [8][58]

ROOT allows its users to save and access data in a compressed binary form, to mine data, to create graphics and visualizations (histograms, scatter plots, curve fitting) and to build custom applications. ROOT code can run in an interactive mode thanks to Cling C++ interpreter or it can be compiled. A graphical user interface can be created in both cases. ROOT can also be integrated with other programming languages such as Python and R. [8][58]

When saving files, ROOT by default splits events into its pieces (the variables) and builds the file by putting together those variables. This allows for maximum efficiency of the internal compression and speeds up the process of looping over few variables of each event thanks to caching mechanisms present in disk controllers and in the operating system. [8][58]

Another key feature of ROOT is its data container - *tree*. It can contain *branches* or *leaves*. While a branch can be an arbitrarily complex object and even another tree, a leaf is always a simple variable and is therefore the end point of a branch. This design resembles the data structure of operating systems. Trees are usually split into branches when saving to a file as mentioned above. [8][58]

ROOT v6.24 is used for this work.

1.4.1 mDST

The mini data summary tree denoted by mDST is a ROOT file containing reconstructed events data from the experiment or Monte Carlo simulations. Specifically, it includes:

1. Event dependent information
 - Tracks
 - Vertices
 - Calorimeter clusters
 - Hits
 - Raw information (DAQ digits)
2. Event independent information
 - Tracking detectors geometry
 - Calorimeters geometry
 - RICH geometry

- Magnetic field maps
- Material maps

[16]

1.5 PHAST

PHAST (PHysics Analysis Software Tools) is another C++ framework developed at CERN specifically for COMPASS data analysis on the level of mDST. It offers an access to reconstructed events information and detector properties as well as tools for visualization (see chapter 1.5.1) and mDST processing and filtering. It can also output in mDST format at the stage of event reconstruction (called microDST or shortly μ DST). [15][16]

PHAST data analysis is done via a so called *user function*. It is a function to be called for every event in the input mDST file. This can be done recursively, i.e. using the output microDST and reading it again by PHAST for further processing. Multiple files processing can also be done at once as well as using multiple user functions one after another. [15]

The user function is to be specified only inside `./user` directory and defined as `UserEventN()` with `N` being a natural number that is not yet used. For each `N` there are really 3 user functions:

- `UserEventN()` - function is called for every event
- `UserRunEndN()` - function is called every time run number is changing in the input stream
- `UserJobEndN()` - function is called upon the job end

[15]

When a user function is developed and put into `./user` directory, it will be automatically compiled, put to shared library and linked to executable by running `make` command in top PHAST directory. This has to be done every time any source code file is added or modified⁴. [15]

Events are in PHAST represented by `PaEvent` class with many methods for data access. Event dependent information can be therefore described as a tree of objects of class `PaEvent`. Event independent information are represented by `PaSetup` class. Detailed documentation can be found on PHAST Class List website.⁵ [15]

PHAST v8.022 is used for this work.

⁴When existing files are modified, only the affected files are recompiled.

⁵<http://ges.web.cern.ch/ges/phast/doxygen-html/annotated.html>

1.5.1 PHAST Event Display

PHAST can be also compiled with activated graphics mode (`touch WITH_GRAPH` prior to compilation). Input mDST files can then be inspected using PHAST Event Display, which outlines the experiment schematic structure with an event. An event (a particle collision) contains at least one particle and its trajectory, but usually contains multiple. Phast Event Display shows a schematic of this event with multiple filtration options and context menus. [16]

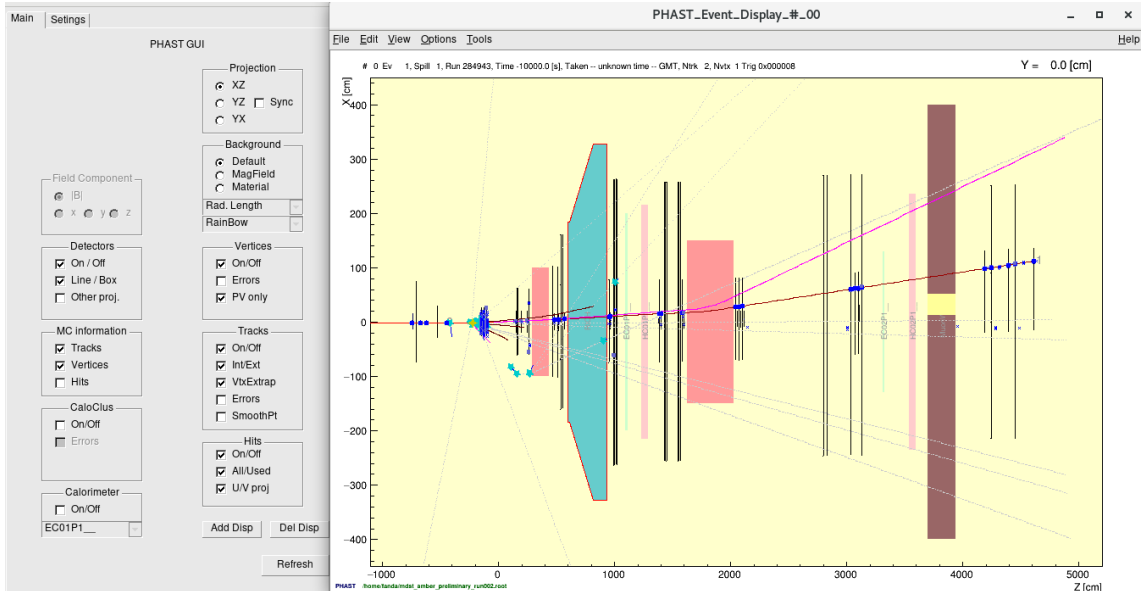


Figure 1.3: PHAST Event Display example. Rectangles represent detectors, dots represent particles and lines their trajectories.

1.6 TensorFlow

TensorFlow is a free open source library for all sorts of machine learning with focus on deep neural networks. It was originally developed for Google’s internal use and is still used in its production. TensorFlow can run on multiple CPUs as well as GPUs with possibility to use CUDA Toolkit extension. Version 2.8.0 is used. [55][61]

1.6.1 Keras

Keras is an open-source API specification written in Python that describes how a deep learning framework should implement certain parts. It was framework agnostic and supported different backends (Theano, TensorFlow), but over the time Tensorflow has fully adopted Keras API and integrated it as a sub-module. Keras is therefore the high-level API of TensorFlow 2, which is the only supported backend since version 2.4. Keras is focused on enabling fast experimentation and development. [55]

The installation and development setup of above mentioned software tools can be challenging and was therefore described in detail in preceding work [41].

1.6.2 Frugally-deep

Frugally-deep is a header-only library for exporting a TensorFlow model and using it for forward pass in C++ programs without linking it against TensorFlow, because its parts needed for prediction are re-implemented in C++. This reduces executable size significantly. [19]

It requires C++ 14 and three additional, also header-only, libraries: `json`, `fplus` and `Eigen`. Version 0.15.16 is used.

Chapter 2

CEDAR detectors

CEDAR is an abbreviation for ‘CErenkov Differential counters with Achromatic Ring focus’. It is used for beam particle identification of the particles that cross it.

To distinguish pions, kaons and antiprotons from which the negatively charged hadron beam is mainly composed, two alike CEDAR detectors are positioned approximately 30 meters upstream of the COMPASS target. [1][42][43]

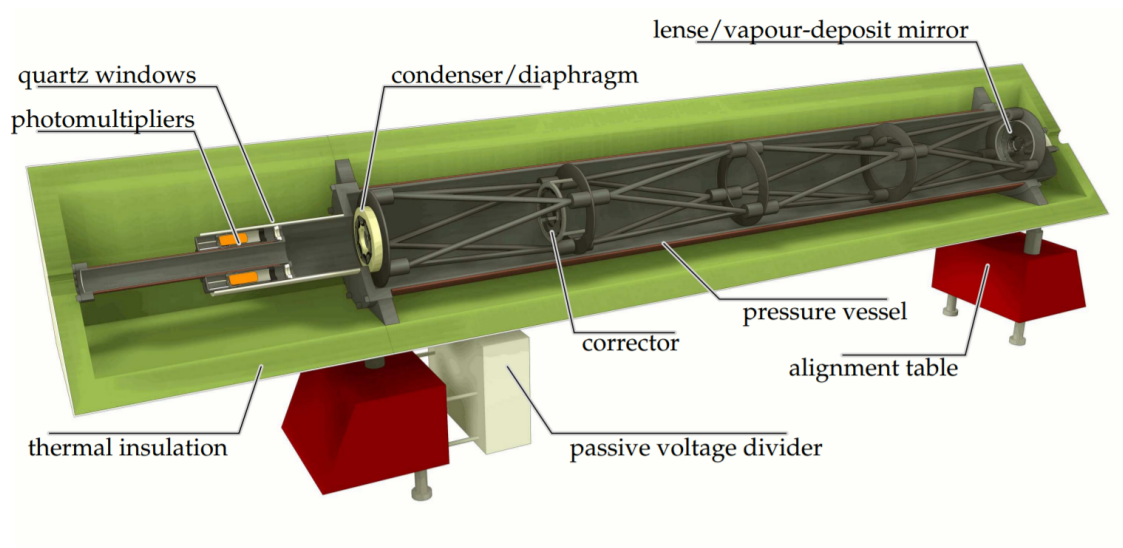


Figure 2.1: Schematic view of a CEDAR detector. [1]

2.1 Detector principle

The CEDARs make use of Cherenkov radiation, which is emitted by charged particles passing through a dielectric medium at a speed greater than the phase velocity of light in that medium. Beam particles of different types with the same momentum that traverse the CEDAR counter emit Cherenkov light in different angles due to unlike masses. [1][42]

This light is focused onto a ring shaped diaphragm by a concave mirror and a system of lenses (lens, corrector, condenser). The diaphragm is located in the focal plane perpendicular to the beam direction whereby compensating for the chromatic aberration¹ in the gas helium with which the vessel is filled. It is set to select photon rings with a fixed radius. The radius of the photon ring and diaphragm is matched by adjusting the pressure in the vessel. [1][42][43]

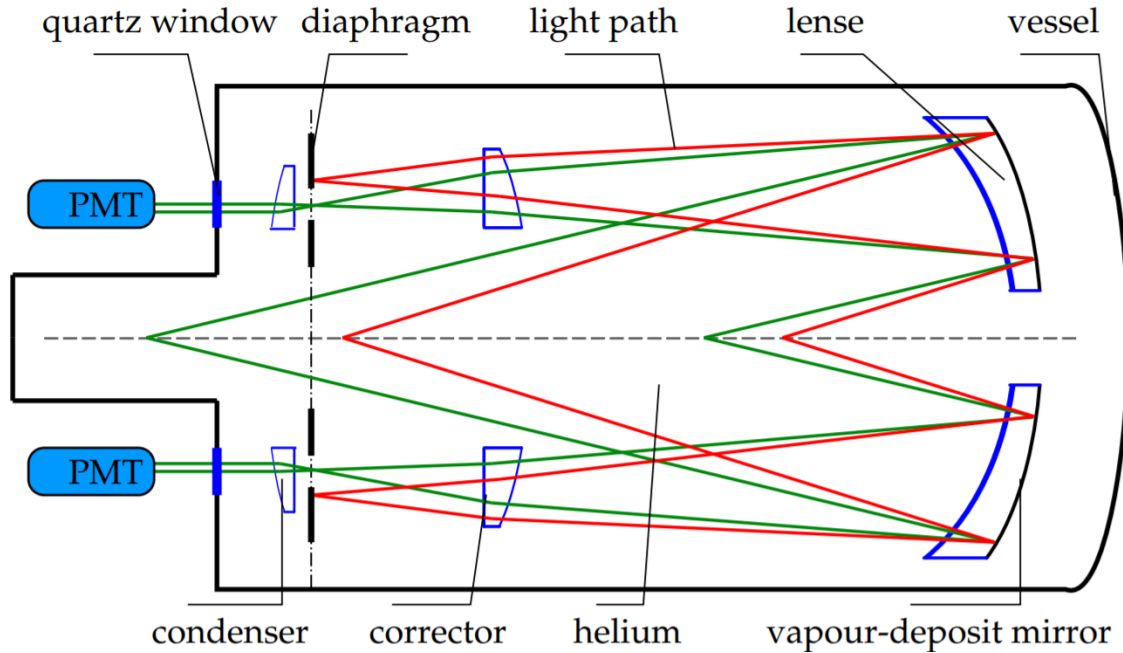


Figure 2.2: CEDAR schematic - two particles with the same momentum but with different masses, represented by red and green lines, radiate Cherenkov light at different angles. This results in rings with different radii. A diaphragm selects the rings from the required particle type (green line), while the radiation from the other particle does not go through the diaphragm to trigger signal from the PMTs. [1]

At last, the Cherenkov photons are detected by 8 photomultipliers (PMTs) arranged in a ring around the optical axis of the detector behind the diaphragm.

Because the momentum of beam particles is approximately the same, the emitted Cherenkov radiation angle is the same for particles of the same type. This means that CEDARs do not need to measure the actual emission angle value. To select certain particle species, pressure inside the vessel and the diaphragm opening is tuned so that the light is focused onto the PMTs, while Cherenkov light of the other species with different photon ring radius is shielded by the diaphragm. [1]

¹Chromatic aberration is a failure of a lens to focus all wavelengths to the same point, making resulting reflection look blurry.

2.2 Problems using CEDAR information

The photon rings are smeared by several effects, e.g. beam divergence, over time temperature changes and imperfect precision of alignment. In order to keep the refractive index constant along the whole 6 meters long vessel, good thermal insulation and conduction is obligatory. Beam divergence can be compensated only by broadening the diaphragm opening, which leads to lower purity of the particle identification. In addition, CEDARs parameters change drastically over time which requires a run-by-run calibration. [42]

The original paper [5] describes the maximum beam divergence for the detector to operate properly for beams of different energies up to 370 GeV. For 190 *GeV* beams it is around $65 \mu\text{rad}$.

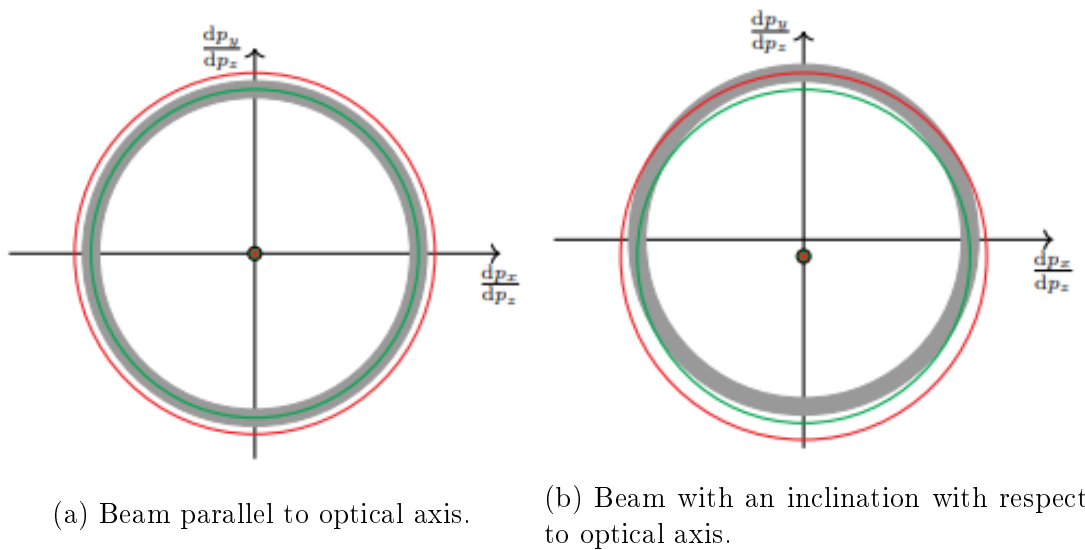


Figure 2.3: Illustration of the Cherenkov rings of a pion (red) and a kaon (green) without and with an inclination with respect to optical axis of CEDAR, when the detector is set to select kaons. The gray ring represents the acceptance area of the diaphragm. One can observe how an inclination of the beam influences PMTs receiving signals. Here a kaon would not give signal in the topmost and bottommost PMT while a pion would hit the topmost PMT.² [42]

Taking the above mentioned into account, a signal in at least a certain number of PMTs (usually 6 out of 8) is required in order to identify a particle. This is known as the *majority approach*, however it works only when the particles traverse the CEDAR parallel to its optical axis. Otherwise, the photon rings are tilted out of or into the acceptance of the diaphragm as illustrated in fig. 2.3. Because the spread in the inclination of particles is of the same order of magnitude as the distance between kaon and pion Cherenkov angles, particles traversing at larger angles of the optical axis of CEDAR have to be excluded from further analysis.³ The majority approach was shown to have a low efficiency of about 40 % to 50 %. [42][43]

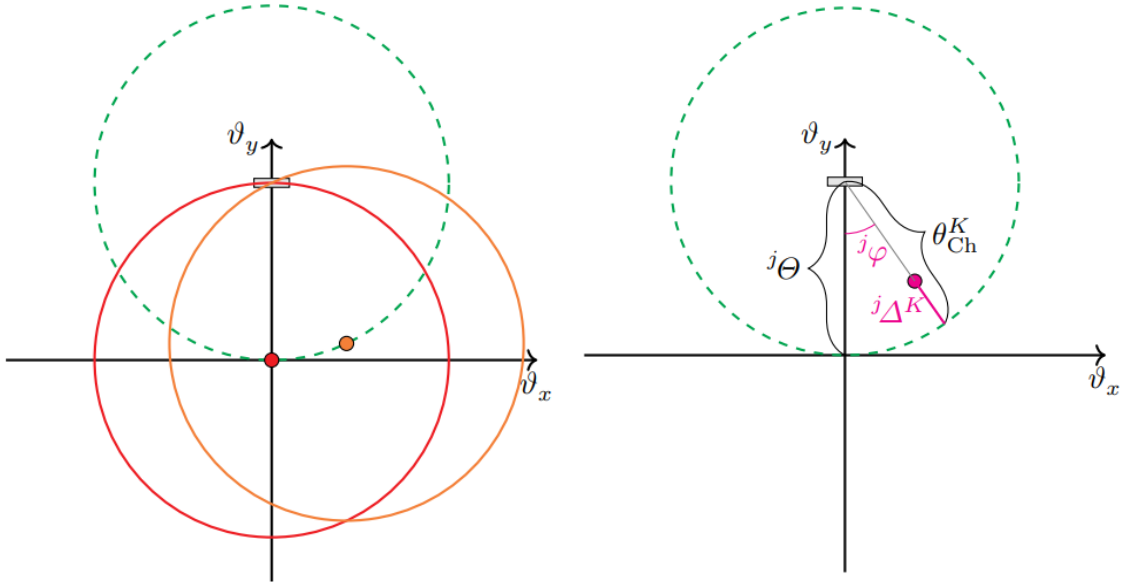
²In reality, the PMTs are shifted by 22.5° clockwise.

³Until 2018, the inclination was measured by a silicon beam telescope at the target position

2.2.1 Likelihood method

Although the beam inclination leads to a low efficiency of the majority approach, fig. 2.3 shows that a certain PMTs hit pattern is characteristic for different particle species at a given inclination. This phenomenon was utilized to develop a *likelihood ansatz* as described in detail in [42] to better compensate for the aforementioned issues. It led to a higher efficiency of about 87 % for kaon identification and 99 % for pion identification. It is integrated as a PHAST user event (fitting the free parameters of the parameterization) and it is commonly used in analyses of hadron induced processes. [42]

The main idea is to use the response of PMTs separately. The probability of a PMT response on kaons or pions is parameterized as a function of the direction of the emitted photons. The direction depends on the particle species and the beam inclination θ_x and θ_y with respect to nominal beam axis. [42][43]



(a) Possible beam kinematics of the same particle species (red and orange points). (b) A particle with arbitrary kinematics (purple point).

Figure 2.4: An inclination space of a beam particle with respect to the CEDAR optical axis. The gray box represents the sensitivity area of a PMT. The first plot shows Cherenkov rings hitting the observed PMT in centrally for points lying on top of the green dashed circle. [42]

The coordinate system is then designed in a way that the hit probability does not depend on the Cherenkov angle and thus on the particle type. The difference of angles of different particle species are included in the coordinate system and a common parameterization of the CEDARs response can be used for both pions and kaons. The probability of a PMT signal is parameterized as a function of the particle

and then calculated at the CEDAR position using known beam optics and a so called transport matrix. This is important especially for the Likelihood method. [42]

kinematics $(^j\Delta^K, ^j\varphi)$. This approach leads to 73 free parameters to be fitted from data for a single CEDAR. [42]

In fig. 2.4, the particles of a certain type that hit the PMT centrally, lie on a circle around the PMT with radius determined by the corresponding Cherenkov ring (green dashed circles). For a particle of a type K and arbitrary kinematics, the probability of the PMT being triggered mainly depends on the distance from the ideal kinematics (green dashed circle) $^j\Delta^K$, Additionally, it depends on the angle of the tilt of the Cherenkov ring $^j\varphi$. This transformation from (θ_x, θ_y) to $(^j\Delta^K, ^j\varphi)$ is different for each PMT and particle species the CEDAR is supposed to select. Hence, the parameterization is independent on the particle species as the different Cherenkov angles are included in the coordinate system used. [42]

However, the 2018 COMPASS data taking used a much higher beam intensity than ever before (15 times higher than in previous COMPASS runs in 2008 or 2009). In addition, the beam divergence RMS during those runs was $\approx 120\mu\text{rad}$ per plane⁴ with only about 10 - 15 % of events falling into the designed operating radius. A major upgrade of the fronted electronics and PMTs at CEDARs as well as a redesign of the corresponding firmware took place to prepare for this data taking. The previously used analysis method using likelihood approach is not applicable, though, as elaborated in 4.1.

⁴This beam divergency of $300\mu\text{rad}$ is not uncommon.

Chapter 3

Machine Learning approach

Machine learning (ML) is vaguely defined as computational methods that use experience and data to improve performance or make predictions and decisions. It involves creating computer programs that are able to carry out certain tasks without being explicitly programmed to do so, based on available data. [2][10][24]

Because data are used for the learning, their quality and volume are crucial to ML algorithms. As in other areas of computer science, analysis of time and space complexity of the algorithm is critical to measure its quality. In addition, a notion of *sample complexity* is needed to evaluate the amount of sample data required for the algorithm to learn. ML is therefore inherently related to statistics and data analysis. [24]

The problem at hand is an archetypal example of classification problem solvable by *supervised learning* approach.¹

3.1 Classification

Classification is the problem of determining a category or categories (called *classes*) of some *item(s)*. This is often performed by analyzing the observations (called *instances*) into some set of quantifiable properties (called *features*) and comparing those with corresponding properties in sample data. Other option is to compare observations to previous observations using some similarity or distance function. These algorithms are called **classifiers**. [10][24]

Classification can be performed on structured or unstructured data. Examples of classification could be reading hand written digits, dividing incoming e-mails into spam and non-spam, labeling documents based on their topic or image classification tasks such as recognizing shapes and objects in photos. [2][24]

¹The computer is presented with sample input data and desired output (labels), with its goal to find a relationship between the training data features and its labels in order to be able to predict the label of an unseen data. This is the most common approach in classification, regression and ranking problems. [2][24]

The number of classes is usually not large, but can actually be unbounded as is the case of text classification or speech recognition. As the goal of this thesis is to develop a machine learning algorithm that is able to predict a likelihood for a particle traversing CEDAR detector to be a pion or a kaon, it is a problem of binary classification.

There are plenty of methods for solving binary classification problems, such as **Decision tree learning**, **Bayesian networks**, **Logistic regression** or **Support-vector machines**. This thesis will not go into further details about any of these as it was already discussed in the preceding work [41] and instead solely focus on the selected technique called **Artificial neural networks**. The main reasons for selecting neural networks are:

1. **Outstanding results** - NNs generally achieve very good performance for classification tasks and act as an universal approximator. [31]
2. **Advanced software toolkit** - with TensorFlow and Keras, development and deployment of neural networks is achievable and developed code is easily transferable.
3. **Previous experience** - NNs were successfully used for different projects in the collaboration, thus some of the gained knowledge can be utilized.

3.2 Artificial neural networks

Neural networks, vaguely inspired by the biological constitution of a brain, consist of a system of nodes (*artificial neurons* or just *neurons*) and connections between those nodes with weights assigned to each connection. These nodes are usually divided into *layers*, where neurons from one layer are connected to neurons in the next layer, i.e. the output from one layer is the input for the next one.² One layer serves for initial input values and one for the output of the network (i.e. the classification result in our case), with optional so called *hidden layers* in between.³

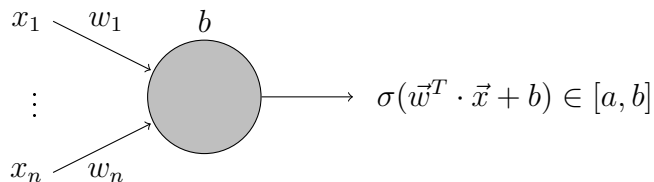


Figure 3.1: Schematic view of an artificial (sigmoid) neuron that converts multiple inputs into a single output in some interval based on the activation function.

²These are called *feedforward* neural networks, because data flows in one direction from the input layer to the output layer.

³There is no special meaning for them to be called ‘hidden’. It simply means they are not input nor output layers.

The output of a neuron is determined by the *weights* \vec{w} associated with each of its inputs x and a so called *bias* b which are transformed by an *activation function* σ : a function that maps the real numbers to some arbitrary interval.

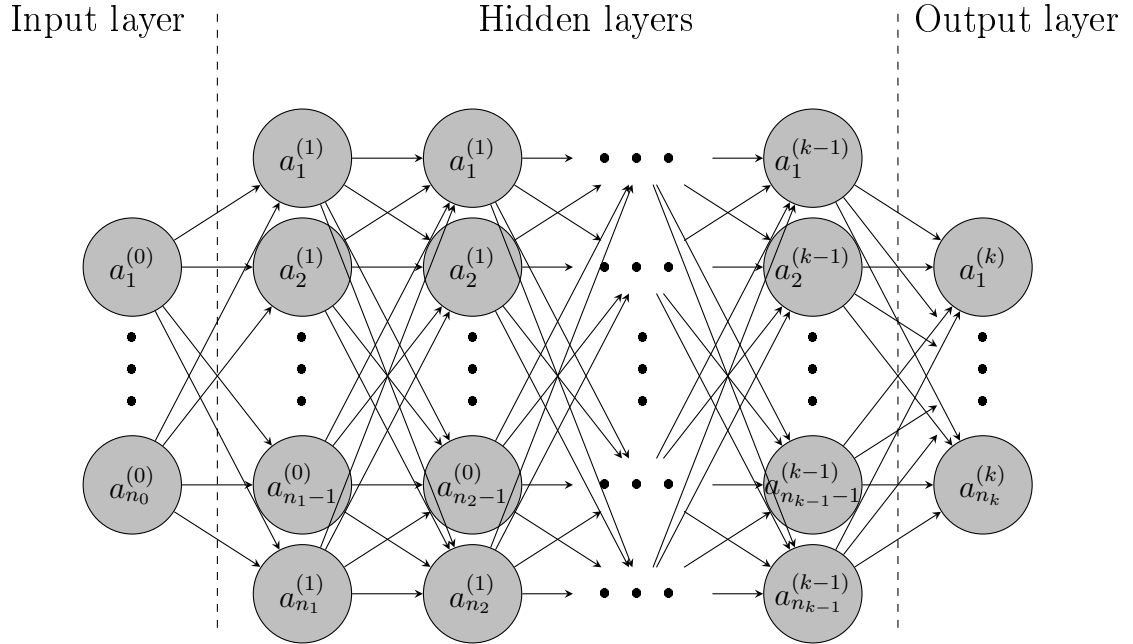


Figure 3.2: A *deep neural network*⁴ with arbitrary hidden layers and arbitrary number of neurons in each layer.

3.2.1 Activation functions

There is a number of activation functions to choose from. Particularly, one could consider the following:

1. Sigmoid function

Sigmoid function is used as an alias for the logistic function:

$$\text{sigmoid}(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)} = 1 - \text{sigmoid}(-z). \quad (3.1)$$

Writing it out for every weight, input and bias, we receive

$$\text{sigmoid}(\vec{w}, \vec{x}, b) = \frac{1}{1 + \exp(-\vec{w}^T \cdot \vec{x} - b)} = \frac{1}{1 + \exp(-\sum_{i=1}^n w_i x_i - b)}. \quad (3.2)$$

The main advantage of using the sigmoid function is the easy computation of corresponding derivatives in learning algorithms.

2. Swish function

The swish function is defined as

$$\text{swish}(z) = z \cdot \text{sigmoid}(\beta z) = \frac{z}{1 + \exp(-\beta z)}, \quad (3.3)$$

⁴Deep refers to having 2 or more hidden layers.

where β is either constant or a trainable parameter. In Keras, $\beta = 1$ and the function becomes equivalent to the Sigmoid-weighted Linear Unit (SiL).

3. ReLU function

The rectifier or ReLU (Rectified Linear Unit) activation function, also known as the ramp function, is defined as the positive part of its argument, i.e.:

$$ReLU(z) = z^+ = \max(0, z). \quad (3.4)$$

It was found that ReLU enables better training of deep networks, compared to some other activation functions such as the sigmoid function. This is mostly caused due to the sparse activation, better gradient propagation, fast computation and scale-invariance. [32]

Potential problems can arise due to its non-differentiability at zero or unbounded character. However, the biggest problem is that in certain conditions, a neuron can become inactive for all inputs, therefore no gradients flow backwards through that neuron and it essentially blocks learning. Some modifications aim to solve these problems, Keras allows to specify 3 parameters:

$$ReLU(z, \alpha, M, T) = \begin{cases} \min(z, T) & \dots \text{ if } z > T \\ \alpha z & \dots \text{ otherwise} \end{cases} \quad (3.5)$$

4. Softmax function

Softmax is usually used for the last layer of a classification network because the result can be interpreted as a probability distribution as it is in range (0, 1) and sum over all output neurons is equal to 1.

$$softmax(\vec{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)} \text{ for } i = 1, \dots, n \text{ and } \vec{z} = (z_1, \dots, z_n)^T \in \mathbb{R}^n \quad (3.6)$$

5. Tanh function

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = \frac{\exp(2z) - 1}{\exp(2z) + 1} \quad (3.7)$$

6. ELU function

Exponential Linear Unit diminishes the vanishing gradient effect and enables faster learning as the mean activations are closer to zero and thus the gradient is brought closer to the natural gradient.

$$elu(z) = \begin{cases} z & \dots \text{ if } z > 0 \\ \alpha(\exp(z) - 1) & \dots \text{ otherwise} \end{cases} \quad (3.8)$$

A slight modification called SELU (Scaled Exponential Linear Unit) can be obtained by simply scaling ELU by some factor to ensure a slope larger than one for positive inputs.

[29]

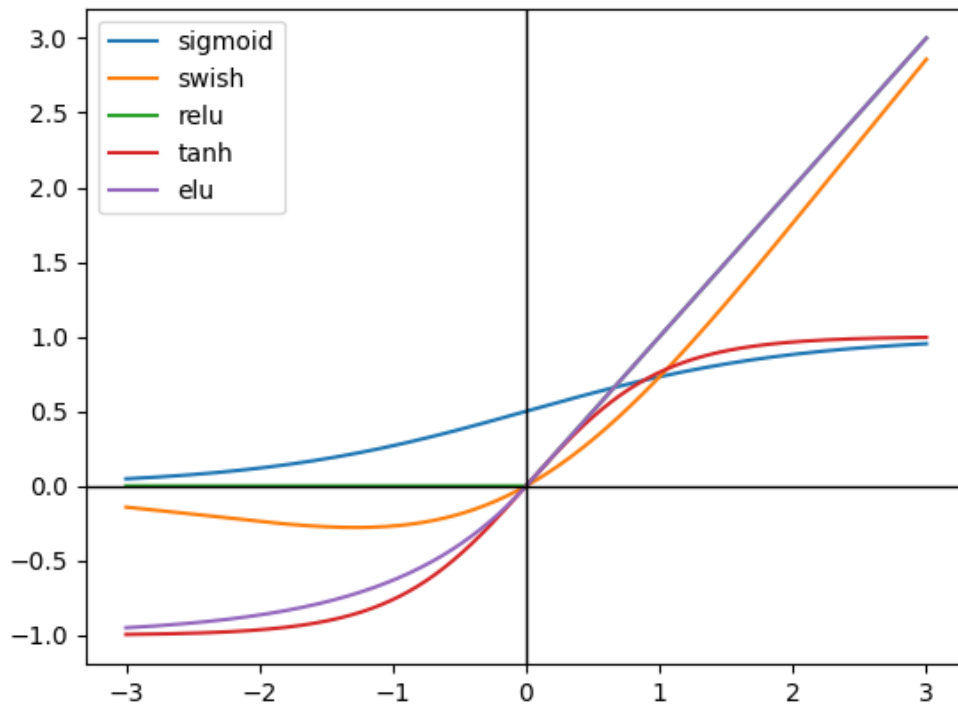


Figure 3.3: Graphs of some of the presented activation functions.

3.2.2 Cost functions

In order to quantify how well is the NN performing, we define a *cost function*.⁵ To put it plainly, the cost function simply compares how much on average the NN deviates from the correct output. The aim is going to be to minimize this non-negative function.

There is a reason why we define a cost function instead of directly trying to maximize the number of correct classifications. The problem with approaching the task directly is that the number of correct classifications is not a smooth⁶ function of the weights and biases. In general, making small changes to these weights and biases will not cause a change in the number of correctly classified training data points. This makes it difficult to assess expedient changes in order to improve the performance.

The cost function instead quantifies exactly how much the NN deviates from the correct result. This function is smooth, which allows us to use powerful algorithms such as the *gradient descent*.

There are multiple of loss functions to choose from, such as:

⁵Sometimes also called *objective* or *loss function*.

⁶A smooth function is infinitely differentiable over some domain.

1. Quadratic cost function

Quadratic cost function or mean squared error, denoted by MSE, is defined as:

$$MSE(\mathbf{W}^1, \dots, \mathbf{W}^k, \vec{b}^1, \dots, \vec{b}^k) = MSE(\mathbf{W}, \mathbf{b}) = \frac{1}{2n} \sum_{i=1}^n \|\vec{y}_{x_i} - \vec{a}_i\|^2, \quad (3.9)$$

where $\mathbf{W}^s, s \in \hat{k}$ is a matrix of all weights for s -th layer with elements w_{ji}^s representing a weight for connection of i -th neuron in $(s-1)$ -th layer and j -th neuron in s -th layer, $\vec{b}^s, s \in \hat{k}$ is a vector of all biases for neurons in s -th layer, \vec{y}_{x_i} is the correct label for i -th training data \vec{x}_i and \vec{a}_i is the output of the NN for this data. We sum over n available training data points and the NN has $(k+1)$ layers including the input and the output layers. The input layer will have index 0 and the output layer index k .

The ordering of the j and i indices in weights matrices may seem counter-intuitive. The advantage of ordering weights in such fashion is that the *activation* (i.e. the output) of the s -th layer is

$$a^s = \sigma(\mathbf{W}^s a^{s-1} + b^s). \quad (3.10)$$

Note that $\sigma(\vec{z}) := \begin{pmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_p) \end{pmatrix}$, that is applying the activation function element-wise with a vector as a result.⁷ [41]

2. Binary Crossentropy

Specifically for a binary classification problems, using Binary Crossentropy⁸ as a loss function can prove very beneficial, and the output can be interpreted as approximated probability.

$$BC(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^n \vec{y}_{x_i} \log(\vec{a}_i) + (1 - \vec{y}_{x_i}) \log(1 - \vec{a}_i), \quad (3.11)$$

Reading the formula, $\log(\vec{a}_i)$ is added to the loss for each input with label 1. Conversely, $\log(1 - \vec{a}_i)$ is added to the loss for each input with label 0. For inputs within a small phase space, we can assume the output of the network to be constant. We can then rewrite the sum as:

$$BC(\mathbf{W}, \mathbf{b}) = S \log(p) + B \log(1 - p), \quad (3.12)$$

where $S = \# \text{events labeled as 1}$ and $B = \# \text{events labeled as 0}$ and p is the network output. Taking a derivative of eq. 3.12 and setting it equal to 0, we conclude:

$$\begin{aligned} \frac{\partial}{\partial p} (S \log(p) + B \log(1 - p)) &= 0 \\ \frac{S}{p} - \frac{B}{1 - p} &= 0 \\ p &= \frac{S}{S + B}. \end{aligned} \quad (3.13)$$

⁷Without this notational trick the weights matrix would have to be transposed.

⁸The function is derived using entropy and information theory.

Therefore the minimum is realized by the probability and all of the statistical tools can be used. MSE leads to the same result, but binary crossentropy is usually preferred in classification problems while MSE in regression tasks. Binary crossentropy can be generalized for multi-class classification into a **Categorical Crossentropy**. [17][35]

3.2.3 Optimizers

So far, we presented a basic NN model, but we restrained from a crucial part: learning. When we talk about learning, what we want is an algorithm which lets us find weights and biases such that the NN approximates correct output $\vec{y} = y(\vec{x})$ for all training data \vec{x} .

Again, there is a number of learning algorithms to choose from. They all make use of computing (or approximating) gradients and modifying the weights and biases in a fashion to improve loss function. For the computations of gradients, *backward propagation of errors* or simply *backpropagation* algorithm is used.

1. Gradient Descent

The foundation for all algorithms is the gradient descent, which derivation was shown in the preceding work [41]. Let \vec{v} be vector of all weights and biases. Then we can iteratively update the weights and biases using this formula:

$$\vec{v} \rightarrow \vec{v} = \vec{v} - \xi \overrightarrow{\nabla C}, \quad (3.14)$$

Where ξ is a positive parameter known as the *learning rate* and $\overrightarrow{\nabla C}$ is the gradient⁹ of the cost function. Eq. 3.14 gives a prescription of how to iteratively adjust the variables so that the value of a loss function $C(\vec{v})$ keeps decreasing.

Notice that cost functions have the form $C = \frac{1}{n} \sum_{i=1}^n C_i$, i.e. an average over costs of every individual training data. This means we would need to compute the gradients $\overrightarrow{\nabla C}_i$ for all training inputs and then average them to get the resulting gradient $\overrightarrow{\nabla C} = \frac{1}{n} \sum_{i=1}^n \overrightarrow{\nabla C}_i$, which is extremely computationally laborious and learning will therefore occur slow. [41]

2. Stochastic Gradient Descent (SGD)

To accelerate learning we can use a method called *stochastic gradient descent*. Instead of computing the gradient for every training data and then modifying the weights and biases, we divide the training dataset into groups of random m items (referred to as *mini-batch*)¹⁰ and average their gradients. This gives us a good estimate of the true gradient $\overrightarrow{\nabla C}$ while accelerating the learning significantly:

$$\overrightarrow{\nabla C} = \frac{1}{n} \sum_{i=1}^n \overrightarrow{\nabla C}_i \approx \frac{1}{m} \sum_{j=1}^m \overrightarrow{\nabla C}_{i_j}, \quad (3.15)$$

⁹Note the arrow above gradient symbol: in this context we consider gradient to be a vector, somehow inconveniently written using two symbols.

¹⁰It is worth mentioning that this adds another meta-parameter besides the learning rate ξ .

where i_j is a random index from all indices of training data. To connect this with the NN, lets label vector of all of the weights as $\vec{w} = (w_1, \dots, w_p)^T$ and vector of all of the biases as $\vec{b} = (b_1, \dots, b_q)^T$,¹¹ $C = C(\vec{w}, \vec{b})$. We can now write the gradient descent in terms of components $k \in \hat{p}$ and $l \in \hat{q}$ for a given mini-batch as

$$w_k \rightarrow \tilde{w}_k = w_k - \frac{\xi}{m} \nabla C_{i_j}^{(k)} = w_k - \frac{\xi}{m} \sum_{j=1}^m \frac{\partial C_{i_j}}{\partial w_k} \quad (3.16)$$

$$b_l \rightarrow \tilde{b}_l = w_k - \frac{\xi}{m} \nabla C_{i_j}^{(l)} = b_l - \frac{\xi}{m} \sum_{j=1}^m \frac{\partial C_{i_j}}{\partial b_l}, \quad (3.17)$$

where $\nabla C_{i_j}^{(k)}$ is a k -th element of gradient $\overrightarrow{\nabla C_{i_j}}$. [41]

3. Stochastic Gradient Descent with momentum

SGD with momentum aims to speed up convergence through denoising the derivatives by exponentially weighting averages of the gradients. This function is called *momentum*. The momentum is computed using all previous gradients and giving higher weights to the most recent updates. [9]

4. Adaptive Gradient (AdaGrad)

The key idea of AdaGrad is to allow an adaptive learning rate for each of the weights. The learning rates are adapted relative to how frequently a parameter gets updated during training through dividing the learning rate by the square root of the cumulative sum of the current and past squared gradients. The more updates a parameter receives, the smaller the learning rate becomes. It can have problems with slow convergence for high number of iterations. [20][56]

5. AdaDelta

AdaDelta tries to solve the problem of slow convergence of AdaGrad by slowing down the learning rate diminution, or more precisely by updating the learning rates based on a moving window of gradient updates instead of using all past gradients. [21][20][56]

6. RMSprop

RMSprop is also an improvement of AdaGrad, but instead of using cumulative sum of squared past gradients, it uses exponential moving average of these gradients, similar to momentum. [20][56]

7. Adaptive moment estimation (Adam)

Adam is a combination of RMSprop and a momentum. It uses the exponential moving average of the past gradients for computing the new gradients (like SGD with momentum) and updates the learning rate in the same fashion as RMSprop. [21][20][33][56]

¹¹Notice the vectors are of different sizes: in a given layer there is a weight for every connection to every neuron, but only one bias for every neuron.

Other optimizers are for example **Adamax**, **Nadam** or **FTRL** algorithm. When training a network, data are shuffled and then presented to do network iteratively in batches of predefined size. This process is repeated with reshuffled data in so called *epochs* (a full cycle through a dataset).

3.2.4 Types of neural networks

Recurrent neural networks

So far we considered exclusively feedforward networks, i.e. networks that do not get any feedback from themselves. This means that information flows in one direction from the input layer to the output layer. However, different approach is possible: allowing loops in the network and hence creating a *recurrent neural network*. This is in fact much closer to the biological analogy of how neurons work. [24][27]

When reading this sentence, understanding of each word is based on understanding the previous words.¹² It is unclear how a feedforward NN could use its reasoning about previous words to make predictions. This is where recurrent NN offer a solution by allowing the information to persist because of the loops present in them. [27]

It can be difficult to understand what is going on in the loop, but really it is just multiple copies of the same network, each passing information to the next one as illustrated in fig. 3.4. This shows that recurrent NN are tied to sequences of data (e.g. audio, time series or natural language). In essence, they are able to connect previous information to the present task, such as using previous words in a sentence to make predictions about the next word. [27]

Because physical events being recorded by the CEDARs ought to be independent, usage of recurrent networks do not seem to provide any benefits.

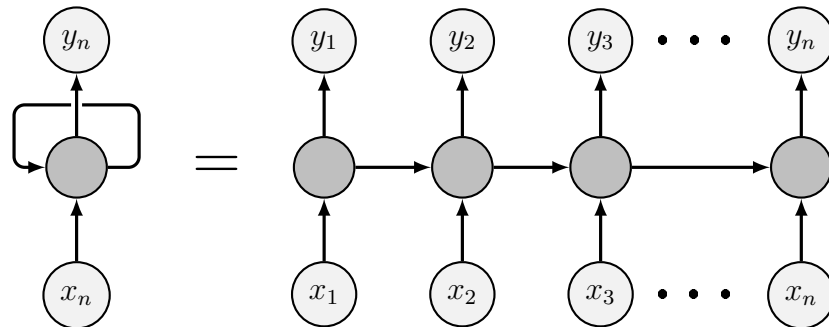


Figure 3.4: A recurrent neural network as multiple copies of a single feedforward network passing information to each other, where $x_i \in \hat{n}$ and $o_i, i \in \hat{n}$ is the input and the output, respectively.

¹²Another example would be listing the alphabet forwards and backwards. It is much easier to remember (or predict) the next letter when the sequence leads to it as opposed to starting from, say, the letter F.

Convolutional neural networks

Convolutional neural networks (CNNs), as the name indicates, use convolution¹³ in place of general matrix multiplication in at least one of its layers. These are called *convolutional layers*. The element involved in carrying out the convolution operation is called a *kernel* or a *filter*¹⁴, which is a matrix of predefined dimensions. This matrix is then multiplied by a portion of the input of the same size, gradually sliding from the beginning of the input to its end. This reduces the input into a form that is easier to process, without losing features critical for reliable prediction. [30]

The reduction is further achieved by using *pooling layers* that use either max pooling or average pooling. Pooling returns a maximum or an average from a portion of the input covered by the kernel.

CNNs are mostly used in image and video recognition because they are generally very good in capturing patterns (the spatial and temporal dependencies) and can process large inputs while reducing the number of learnable parameters compared to a simple feedforward network. [30]

Random Vector Functional Link neural networks

Optimization of the network over all of its parameters (weights and biases) can be very computationally intensive. Furthermore, backpropagation based algorithms often suffer from slow convergence, getting stuck in a local minimum and high sensitivity to the learning parameters. For these reasons, a different approach was proposed and later evaluated in [45]. A RVFL network has randomly generated weights and biases that stay constant during the learning phase between the input and hidden layers. These constants are generated such that the activation functions are not all saturated, i.e. $a_i^{(0)} - b_i^{(0)} < 0$ for at least one i . It was shown that such network is a universal approximator for continuous functions on compact sets. An enhancement to RVFL are direct links from the input layer to the output layer, which were proven in [45] to have a significant effect on the overall performance. [45]

In RVFL network, the input features (let x_i be i -th input feature) are transformed into the so called enhanced features (let \tilde{x}_i be i -th enhanced feature), by feeding it through the enhancement nodes which are present in the hidden layers (with randomly generated weights and biases). The enhanced and the original features are then fed-forward to the output neurons (with weights and biases to be optimized). [45]

In this structure, only the output weights (both for the connections from last hidden layer and input layer) $W^{(k-1)}$ and $W^{(*)}$ are to be optimized. This means solving the problem:

$$t_i = \begin{pmatrix} x_i \\ \tilde{x}_i \end{pmatrix}^T \cdot \begin{pmatrix} w_{\bullet i}^{(k-1)} \\ w_{\bullet i}^{(*)} \end{pmatrix} \quad (3.18)$$

¹³A function derived from two given functions by integration that expresses how the shape of one is modified by the other.

¹⁴A filter could for example detect edges, geometrical shapes etc.

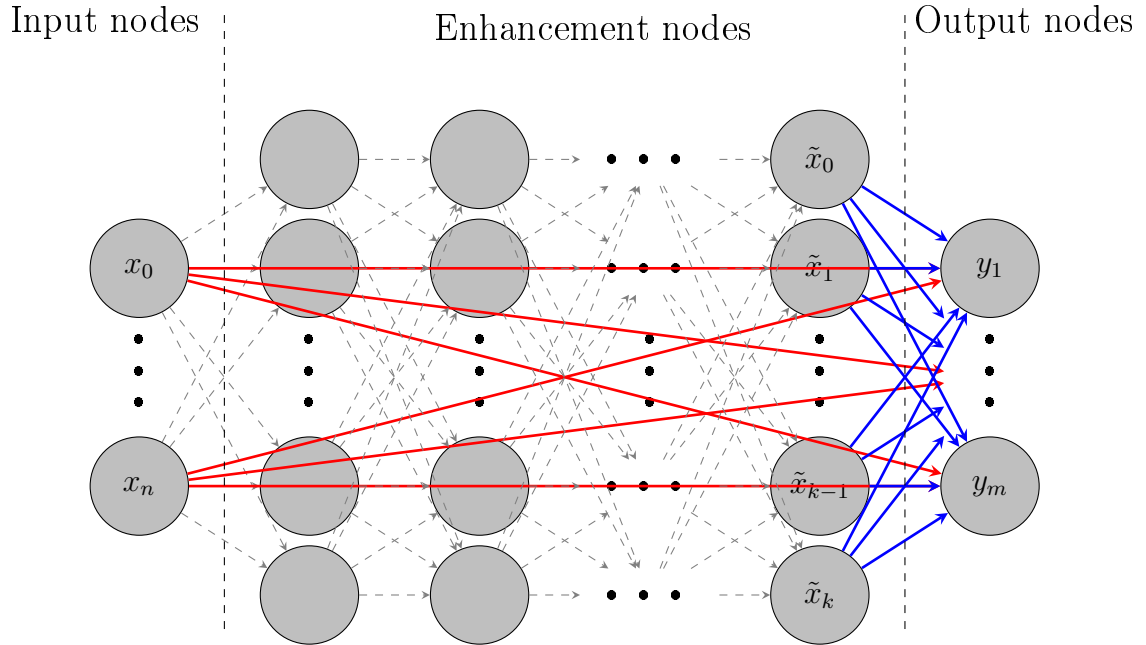


Figure 3.5: A RVFL network with arbitrary hidden layers and an arbitrary number of neurons in each layer. Red lines represent direct connections between input and output layers with weights matrix $W^{(*)}$ and blue lines connections between last hidden layer and output layer with weights matrix $W^{(k-1)}$. Only the colored connections have trainable parameters.

for $i \in \hat{N}$, where N is the number of data samples and t_i is the target i -th output neuron value.¹⁵ [45]

Directly solving eq. 3.18 can be prone to overfitting. A regularization on the solution by e.g. least square method or preference of the solution with a smaller norm value is usually performed to obtain the solution in practice. The optimization of the output weights can be divided into two classes based on the algorithm used. One is an iterative RVFL, which uses gradient of the error function and iteratively tries to minimize the error. The second is a closed-form based RVFL, which obtains the weights in a single step. This is done using pseudo-inverses, out of which Moore-Penrose pseudo-inverse is the most commonly used one. [45]

Developing a RVFL using Keras is possible, because Keras layers may receive a `trainable` parameter to indicate whether the layer should be trained. Adding direct connection from the input to the output layer can be done using multiple inputs and merging it. The functional API has to be used for this purpose.

Radial basis function network

RBF networks usually consist of three layers: input, output and one hidden layer. The input layer feeds the features to the hidden layer, which increases its dimen-

¹⁵For simplicity of notation, the formulation is the same even for cases with biases in the output neurons.

sion by applying non-linear radial basis function¹⁶ as the activation function. This transformation is performed because non-linearly separable classification problems are more linearly separable in higher-dimensional space, according to Cover's theorem. [11][28]

Therefore

$$\# \text{ features} \leq \# \text{ hidden neurons} \leq \# \text{ samples.} \quad (3.19)$$

The radial basis function compares each input to some *prototype*, which is a vector from the training set. Each neuron in the hidden layer has a prototype vector and hence computes the similarity between the input vector and its prototype vector. The most commonly used is Gaussian basis function:

$$a_i = \phi(\vec{x} - \vec{c}_i) = \exp\left(-\frac{\|\vec{x} - \vec{c}_i\|^2}{2\delta_i^2}\right), \quad (3.20)$$

where \vec{c}_i is the prototype of i-th neuron and $\delta_i \neq 0$ is the neurons bandwidth (a non-trainable parameter to be obtained by e.g. a clustering algorithm). The activations tend towards a typical bell curve, where \vec{c}_i describes its center and δ_i its width. The output is then computed in a standard fashion, i.e.:

$$y_j = \sigma(\sum_i w_{ji} a_i + b_j), \quad (3.21)$$

where y_j is the output of j-th neuron and w_{ji} is the weight associated with the connection between i-th hidden neuron and j-th output neuron. More often than not, the bias is left out and identity function is used for activation. Note that w_{ji} describes the height of the bell curve's peak. [28]

Besides classification, RBF networks can be used for function approximation or time series prediction.

RBF network can be implemented in Keras by reimplementing a subclass of `tensorflow.keras.layers.Layer`.

¹⁶RBF is a function $\phi : \cdot \rightarrow \mathbb{R}$ that measures distance of the input and some fixed point, either the origin or some other fixed point called the *center*.

Chapter 4

New methods for beam particle identification

This chapter will provide theoretical proposition of three different methods for beam particle identification using data from CEDARs and reasoning for a need to develop a new one.

4.1 Problems with the 2018 data taking

Because the 2018 data taking used a beam with much higher intensity, the silicon beam telescope had to be removed as it could not withstand the increased radiation. Instead, fiber detectors were installed. While measurements of beam position have

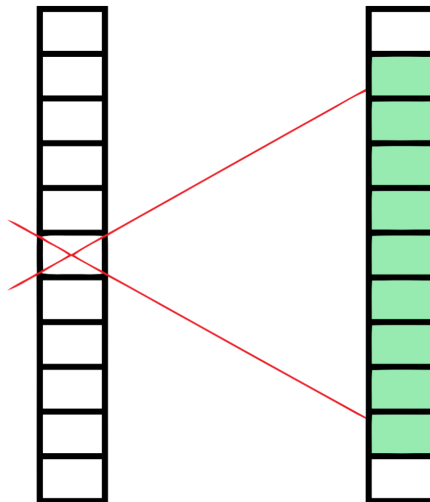


Figure 4.1: Two fiber detectors measuring beam angle.

alike precision when performed by the silicon telescope or fibre detectors, fibre detectors have around 5 – 10 times higher error margin for beam slope measurements. On the other hand, fibre detectors have better time resolution.

The spatial resolution of a beam telescope can be even further improved, while the same does not apply for fiber detectors. This is because of the nature of fiber detectors, where a signal gets picked up by a slot of certain width as shown on fig. 4.1. For a reasonable distance between fiber detectors, there is a small number of angle values that can be measured, which leads to correlated input data and inaccurate transport matrix (eq. 4.1) as there is a small discrete set of angles dX_{spect} and dY_{spect} . Another factor is that signal at ‘borders’ of the slots gets picked up by one of the neighboring slots, so a smearing must be taken into account.

This results in effect shown in fig. 4.2.

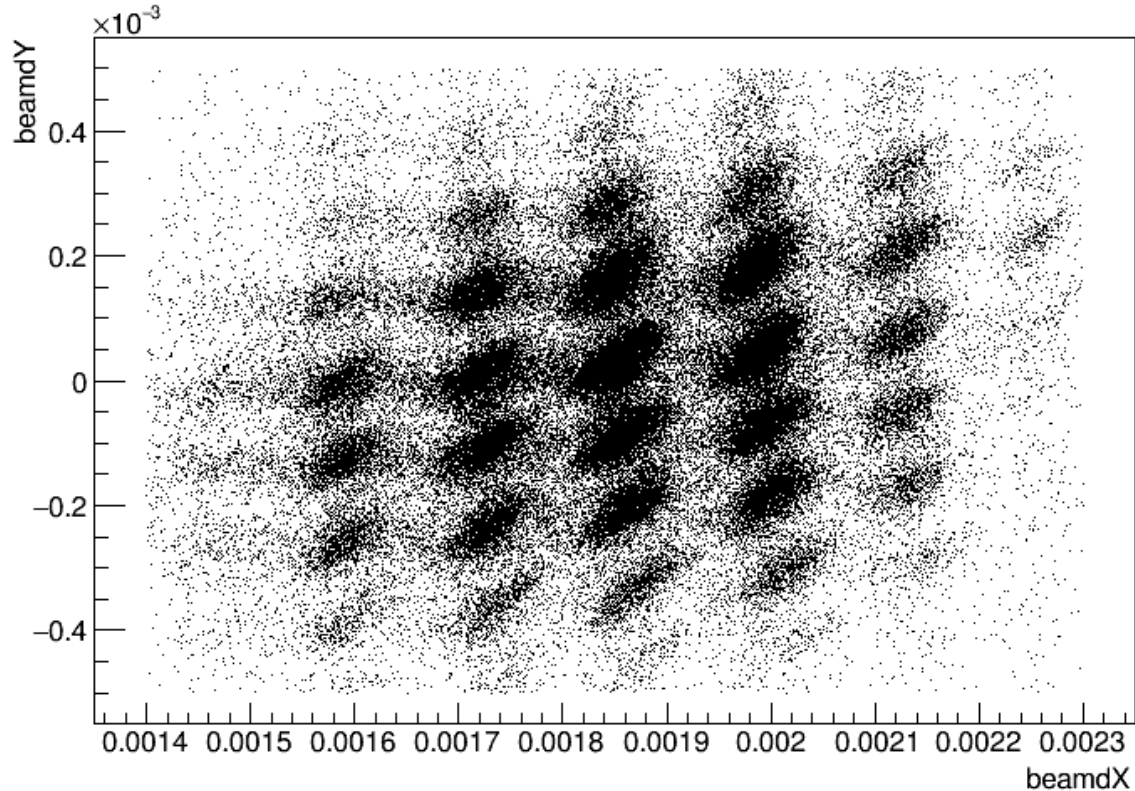


Figure 4.2: A correlation between beam angles caused by low angle measurement precision of fibre detectors.

From the X and Y position at the spectrometer and corresponding angles dX and dY , the angle at CEDAR location is calculated using transport matrix given by the beam group. Note that the beam position at CEDAR does not interest us as the detector was designed such that only the beam angle at CEDAR is of importance (see chapter 2).

The matrix is basically assuming the angle at CEDAR position to be a linear combination of aforementioned parameters, i.e.

$$dX_{cedar} = a \cdot X_{spect} + b \cdot dX_{spect}. \quad (4.1)$$

The same equation holds for Y and dY . In 2008, $b = 0$, so the angle at CEDAR position depended only on the position at spectrometer.

However, $b = 0$ is not the case for the 2018 data taking and the low angle precision of fibre detectors is thus a big factor. The fact that $b \neq 0$ is caused by having beam with higher intensity and colliding particles with polarized target, hence needing to keep it at low temperature. The target is enlarged in order to prevent it from overheating, which requires different settings of focusing magnets and by that means changes the beam optics for which CEDARs were built.

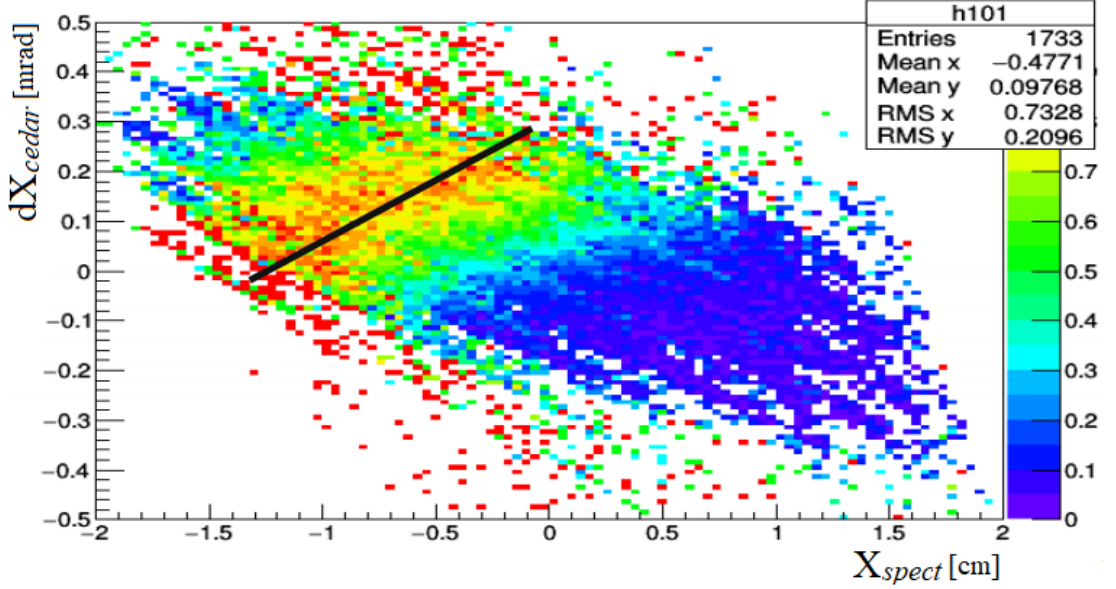


Figure 4.3: Histogram of PMTs response ratio in X_{spect} and dX_{cedar} coordinates. [36]

The transport matrix given by the beam group seems incorrect, as PMT response (color maps) in fig. 4.3 show clear correlation between dX_{cedar} and X_{spect} , while efficiency should be constant for the given dX_{cedar} . One can find better transformation matrix by ‘rotating’ the histogram in fig. 4.3 in such way that the PMT response indeed only depends on its angle at CEDAR: [36]

$$\begin{aligned}
 dX_{cedar} &\approx 0.18(\cos(1.15) \frac{1000dX_{spect} - 1.891}{0.172} + \sin(1.15) \frac{X_{spect} + 0.4323}{0.8852}) \\
 dY_{cedar} &\approx -0.12(\cos(0.47) \frac{1000dY_{spect} - 0.07835}{0.2071} + \sin(0.47) \frac{Y_{spect} - 0.1654}{0.6209}).
 \end{aligned} \quad (4.2)$$

Therefore all four position parameters could be aggregated into two. It is worth investigating the performance of the NN, when it is given the four original values versus the 2 computed angles. If all is well, there should not be much difference - although the NN could actually be able to find a better approximation for the angles at CEDAR position. A good exercise is to present the NN with both the original data and the approximations. If the learning phase is successful (depends on many factors including the NN structure, data labeling etc.), there should not be any improvement when inputting more parameters as the information is already present in the 4 original parameters.

Results of this exercise were as expected, although the network with 4 input variables describing angle and position seem to perform slightly better and using all 6

parameters also resulted in an insignificant improvement. However, this conclusion is not entirely clear due to the problems with kaon proxy as discussed in chapter 5.5.1. The results can be found in app. A.1.

4.2 Using neural networks for classification

There are three distinct methods proposals to examine. While the first one tries to directly predict the particle type, the other two formulate and compare two hypothesis to get the final likelihood.

4.2.1 Method 1: NN as a direct classifier

The most straightforward approach is to use the data about beam position and angle at spectrometer together with responses from PMTs of each CEDAR separated bitwise for the input and have a single output neuron. Note that its value cannot be directly interpreted as a likelihood, due to the imbalance between the kaon and the pion samples, referred to as the signal and the background sample. For each CEDAR, one model will be trained on a dataset with mixed background (pion) and signal (kaon) data.

The outputs of the two models should be combined so as to aggregate the information, e.g. by addition, multiplication, logical AND and so on¹. A threshold should be then selected in a way that the sample is proportionally divided into two sets (2.5 % of events in the background sample should be labeled as kaons and the rest as pions) or by selecting a desired efficiency working point.

4.2.2 Method 2: NN as a PMTs response predictor

The second approach is to use the network to predict a probability that certain PMT fires assuming the event is induced by kaon or by pion. Therefore there will be at least two models: one trained on the background dataset and the other on the signal dataset.

The input will consist of 4 nodes corresponding to beam position and angle at spectrometer or the 2 computed eff. angles. However, it is unclear how each of the models should be built in terms of the output layer. One could try predicting all 16 PMTs responses at once, separating it into two models based on CEDARs or building a network for each of the PMTs response individually. This is to be investigated in the implementation phase.²

The probabilities outputted by the NN would then be compared with the actual CEDARs responses and a likelihood (or log-likelihood) function could be formulated

¹Using a single network for both CEDARs is problematic and elaborated in chapter 5.5.1

²Obviously, one would start with one network for all PMTs responses and check its stability and performance and only after that divide it into smaller networks.

for the particle to trigger a certain PMTs response pattern when being a kaon or a pion (denoted by S and S').

$$\begin{aligned} & {}^S\mathcal{L}(S, X, Y, dX, dY) \\ & {}^{S'}\mathcal{L}(S', X, Y, dX, dY). \end{aligned} \tag{4.3}$$

Note that the likelihood function is dependent not only on the input variables, but on the trained NN as well. This is emphasized by the left superscript, which is formally redundant.

We then calculate both of these likelihoods for a given event. Lets consider a model trained on signal first, i.e. on kaons. Lets assume the NN predicted a certain PMT to fire for with probability $p_i, i \in \{1, 2, \dots, 16\}$, e.g. when $p_1 = 0.3$ we expect the first PMT to fire with 0.3 probability and not give signal with 0.7 probability. Lets denote the PMTs actual responses by $x_i \in \{0, 1\}$. Lets denote the probability of obtaining given response by ℓ_i , that is:

$$\ell_i = \begin{cases} p_i \dots & \text{if } x_i = 1 \\ 1 - p_i \dots & \text{if } x_i = 0. \end{cases} \tag{4.4}$$

Calculating ${}^S\mathcal{L}(S, X, Y, dX, dY) := \prod_{i=1}^{16} \ell_i$ we get a likelihood of the given response pattern occurring assuming the particle is a kaon.³ Doing the same for the model trained on pions, we obtain the value for ${}^{S'}\mathcal{L}(S', X, Y, dX, dY)$, i.e. the probability of this PMTs response pattern occurring for a pion.

The thing to keep in mind with this approach is that the likelihood is not really a probability as it does not take into account the correlation between PMTs patterns.

4.2.3 Method 3: NN as PMTs pattern predictor

The aim of this method is to take into account the above mentioned correlation between PMTs responses. There will be two networks for each CEDAR, one trained on the signal sample and the other trained on the background sample. Each of these networks will have 256 neurons in the output layer, each neuron accounting for a certain combination of PMTs.⁴ Note that the output layer should use an activation function such that a sum over all output neurons is one.

The hope is that the network would be able to predict the probability for a given particle to produce a particular CEDAR response based on its position and angle at spectrometer, hence the output of the model will be a likelihood of the observed PMTs pattern occurring assuming a species of the particle that induced it. The

³Because the numbers will be generally very small, it can be beneficial to define a log-likelihood function and calculating $\sum_{i=1}^{16} \log(\ell_i)$

⁴There are 8 PMTs in one CEDAR that can output 2 values, which results in $2^8 = 256$ combinations.

likelihoods of both hypothesis are divided and the result is compared to some pre-defined threshold, similarly to method 1.

Note that we need to create a network for each CEDAR due to the exponential growth of the possible combinations. Trying to predict both CEDARs responses at once would require $2^{16} = 65536$ neurons in the output layer. Given four input parameters the NN will definitely be very unstable, i.e. it will get perturbed a lot by minor changes to its inputs.

This concern holds even for the architecture with 4 (or 2) input neurons and 256 output neurons. This may lead to a need for further reduction of the output layer, which on the other hand results in overlooking correlation between PMTs response.

If the learning phase is successful, method 1 and 3 are expected to give rather similar results, while method 2 might perform worse due to the aforementioned correlation.

4.3 Training data

As discussed, in order to utilize neural networks one needs enough data of high quality for the learning phase. The aforementioned methods will be tested on the real data as well as different Monte Carlo simulations.

4.3.1 Dataset types

When a neural network is being trained, three datasets are usually used. The first one is called the *training dataset* and it is used to fit the parameters (compute the gradient), i.e. the weights and biases of the connections between neurons. Also during the training phase, a so called *validation dataset* is used to provide an unbiased evaluation of a model fit on the training dataset while adjusting the model's hyperparameters⁵. In Keras, it is easy to set the training to validation dataset ratio by defining a `validation_split` parameter in `tf.keras.models.Model.fit` function call used for training. The validation is then performed after every epoch. [6]

Another validation approach is using a *k-Fold Cross-Validation* algorithm. First, the training dataset is split into k equal sized groups. One of the groups is then used as the validation dataset, while the rest is used for training. The cross-validation process is repeated k times, with each of the subsamples being used exactly once. This way, each group is used once for validation and $k-1$ times for training. Using k-Fold Cross-Validation algorithms is especially useful in situations with limited data samples. [7]

A potential improvement for unbalanced datasets is offered by the *Stratified k-Fold Cross-Validation* algorithm. It extends the regular k-fold Cross Validation through maintaining the same class ratio for all k folds as the ratio in the original dataset.

⁵The hyperparameters do not get adjusted automatically, but the information allows the developer to tune it accordingly. It is also a good indicator of overfitting.

The third dataset is called the *test dataset* and is only used after the training is finished. It gives an unbiased evaluation of the final model. It is important that this data were not used during the training phase.

4.3.2 Measured data

Data from mDST reconstructed in CORAL were further processed using one of existing PHAST user events used for previous analysis. The resulting `.root` files contain a ROOT tree with number of variables. The position and angle at spectrometer are represented by `beamX`, `beamY`, `beamdX` and `beamdY` variables. The CEDARs response is encoded as a `byte` data type⁶ and represented by `CE1PMn` and `CE2PMn` variables. These will form the foundation for our NN to learn. Other variables contain information about pressure, run number, time etc.

We will be using several datasets. First of all, using the variable `nBeamTracks` we will separate events with only one detected track and events with multiple tracks. As one beam events are the least complicated, the most meaningful results are to be expected. This accounts for approx. 40 % of all events.

Unfortunately, not all tracks get detected and so even some events from the first dataset may be caused by multiple particles. This correlated noise is expected to be the main issue in the classification task.

As the CEDARs were set to select kaons, which account only for ≈ 2.5 % of the events, this data will be labeled as pions (note that this label is correct only for 97.5 % of the events, so one needs to be aware of that not only when using these data for validation and testing, but even for training). We will call this the *background sample* and there is no trouble obtaining more than sufficient number of those events. For the background sample, run 287510 will be used.

Kaon proxy

A worse situation is with obtaining a kaon sample, since in reality there is no such thing. However, as presented in chapter 2, the refractive index and thus the resulting Cherenkov ring of a particle depends also on the pressure inside the CEDAR vessel. This can be utilized to obtain some sort of kaon proxy. In certain pressure range photon rings of pions have the same diameter as the photon rings of kaons at the working pressure, hence the CEDARs treat it like a kaon sample. For CEDAR 2, which overall performs better, the pressure range that accounts for the working

⁶Because there are 8 PMTs for one CEDAR, when we index them, we can represent their response as a binary number with 8 digits (for example 10100001 means that the first, the third and the eighth PMTs gave signal). This gives us $2^8 = 256$ combinations, so it can be represented by an integer in this range. This is the equivalent of `byte` data type. Thus, the example binary number is equal to 161. When selecting events where a certain PMT fired, one can use bitwise AND with the corresponding decimal number. Lets say we want to select all events where the fourth PMT fired. This can be simply achieved by converting the binary number 00010000 to decimal, which results in 16, and then specifying condition `(CE1PMn&16)>0`. This is useful for CEDAR efficiency plots.

pressure of 10.31 bar is somewhere between 10.20 bar and 10.24 bar⁷. We will call this the *signal sample*.⁸

Usually, it is important to handle the cases with unbalanced training data with care as the NN could tend to overfit. There are number of techniques that can be used for such cases. From the easiest approaches, one could try undersampling, oversampling or synthetic sampling. While undersampling means removing some data from the class with more observations, the other two do the exact opposite. Oversampling is basically just duplicating existing data of the class with less observations and synthetic sampling aims to synthetically manufacture observations of unbalanced classes which are similar to the existing using nearest neighbors classification.

To get events in the working pressure range where pions can be used as a proxy kaons, we use run 287088, during which a so called *pressure scan* was performed. This means filling the vessel of CEDARs rapidly with *He* gas and then slowly letting it out, hence decreasing the pressure again. To select events only after the vessel was already filled all the way up, a cut based on time of the event has to be done as shown in fig. 4.4. For the selection, variables `timestamp` and `CE2Pressure` present in the ROOT file will be used.

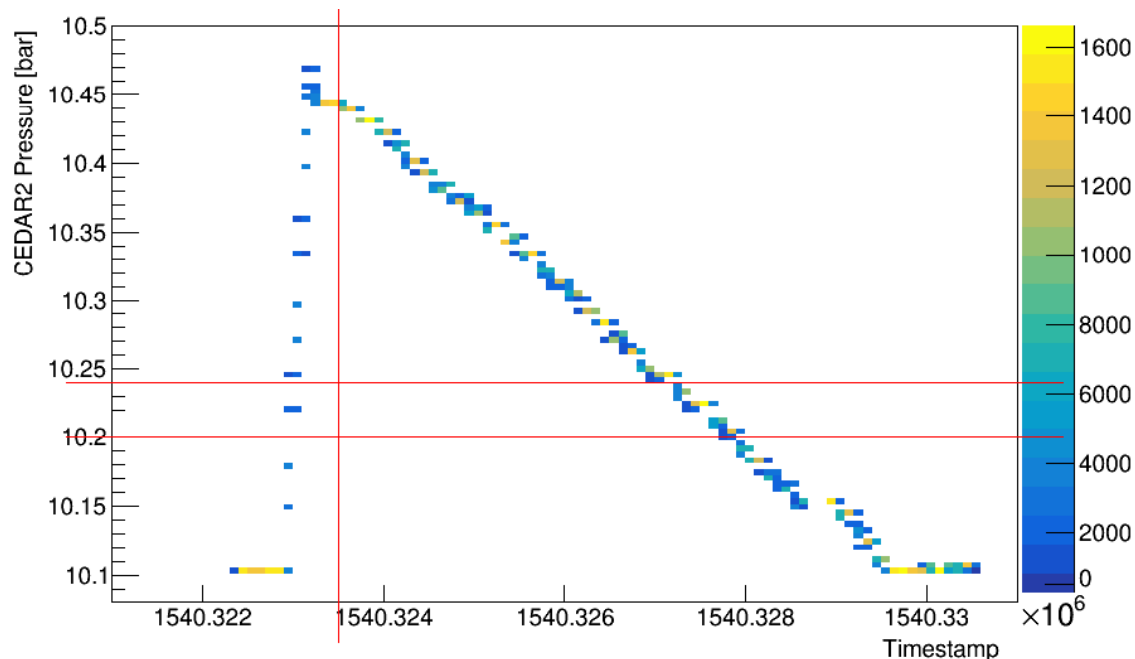


Figure 4.4: A pressure scan with cuts to select events where pion can be used as kaon proxy.

Because both CEDARs operate at different pressures and the filling of the vessel takes place simultaneously, data from only one of the detectors can be utilized at the given moment. For run 287088, CEDAR 2 gives good results and therefore will

⁷This range also accounts for smearing, i.e. selecting events with not fully clear signal. For this reason, also samples with smaller pressure range will be prepared in order to examine whether better purity compensates for smaller sample size.

⁸Like for the background sample, the label will be correct for only ≈ 97.5 % of events.

be used. CEDAR 1 was performing under special hardware conditions (different diaphragm opening size), so its data are generally not suitable for training.

The main problem with the kaon-proxy approach is that pressure scans are formally performed only once per year (although usually with higher frequency). Combined with the speed of the gas releasing, only around 43000 events fit the filtration criteria for this run and this number further decreases when the pressure range is reduced.

Both samples will be combined, creating a *mixed dataset* (for method 1); the background sample will be labeled as 0 representing pion while the signal sample as 1 for a kaon.

Electronic stability

Unfortunately, from time to time part of CEDAR 2 and/or part of CEDAR 2 and full CEDAR 1 were not read due to electronic instability. Those events should be eliminated from the sample used for training. To do so, one can create a spill histogram and fill it with number of events in the given timestamp range and a second histogram with events where e.g. PMT 1 fired.

When everything is fine, the ratio of these two plots should be more or less flat. However, one can see that for some spills PMT 1 fires much less often than for others. These are considered bad spills and they ought to be eliminated from the training samples. This should be done for all 16 PMTs. The expectation is that there are two groups: first group containing PMTs 5–8 from CEDAR 2 and the second one with full CEDAR 1 and PMTs 1–4 from CEDAR2. This can be seen in fig. 4.5.

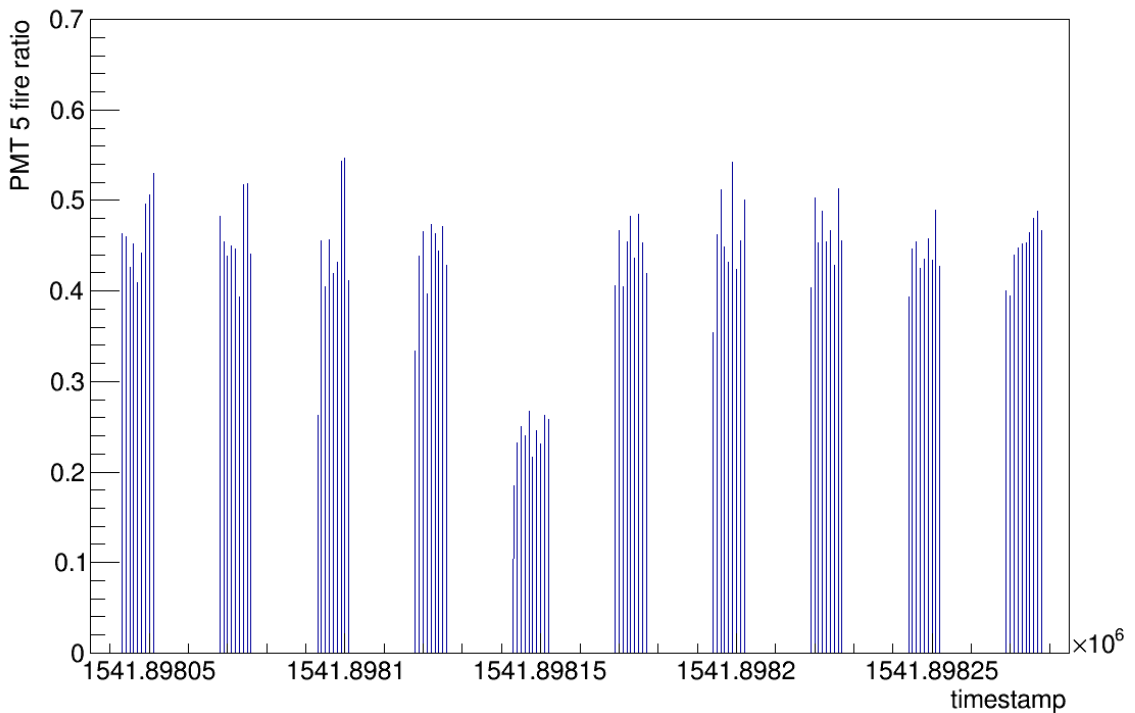


Figure 4.5: A single bad spill for PMT 5 in CEDAR 2.

4.3.3 Monte Carlo simulations

Besides data from detectors, some Monte Carlo (MC)⁹ simulations will be utilized in order to provide more training data¹⁰ as well as identify main issues causing troubles in separation. Only a single CEDAR MC data exist.

The original MC data were ‘too perfect’, because it did not include angle smearing, correlated and random noise and inefficiency. These effects had to be taken into account to better fit the data. When efficiency vs beam angle for different number of PMTs firing is plotted, one can see that the MC simulations do not copy the reality. For example, the bottom left plot of fig. 4.6 shows almost a perfect efficiency of selecting 0.025 kaons, slowly decreasing with the increase of the angle. Unfortunately, that is not the case in the measured data.

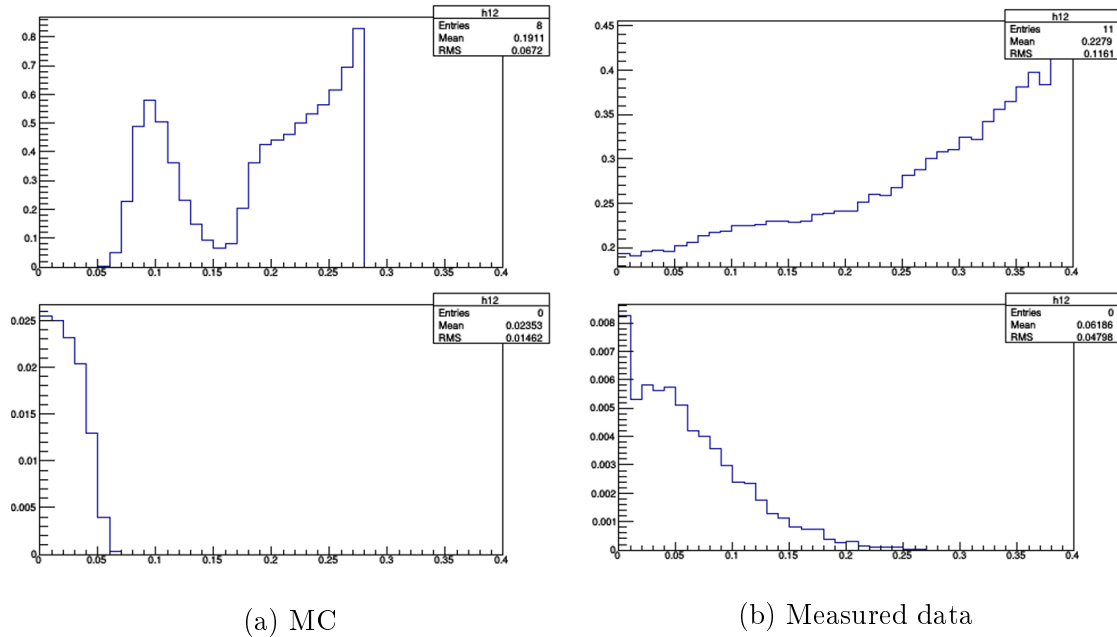


Figure 4.6: Efficiency vs beam angle for 2 (top) and 8 (bottom) PMTs firing.

There were four ‘damages’ done to the MC to better account for data. These abbreviations will be used when talking about these dataset:

1. MC-1xx: additional not detected track (correlated noise)
2. MC-x1x: additional random noise and inefficiency (two similar damages performed at once)
3. MC-xx1: bad knowledge of beam angle at the CEDAR

For example, a dataset with both additional undetected track and bad knowledge of the beam angle at CEDAR will be denoted by MC-101.

⁹MC datasets were obtained from COMPASS collaboration and were not prepared as a part of this thesis.

¹⁰This is especially important for method 3, for which there is not enough training data.

It turned out the most crucial for the distribution description is correlated noise. Without it, no reasonable match between data and MC could be achieved. Tails description remain problematic even after the modifications as shown in fig. 4.7. Because determining which of the aforementioned issues has the biggest impact on the separation can provide valuable information about potential for improvement, all combinations of problems were examined.

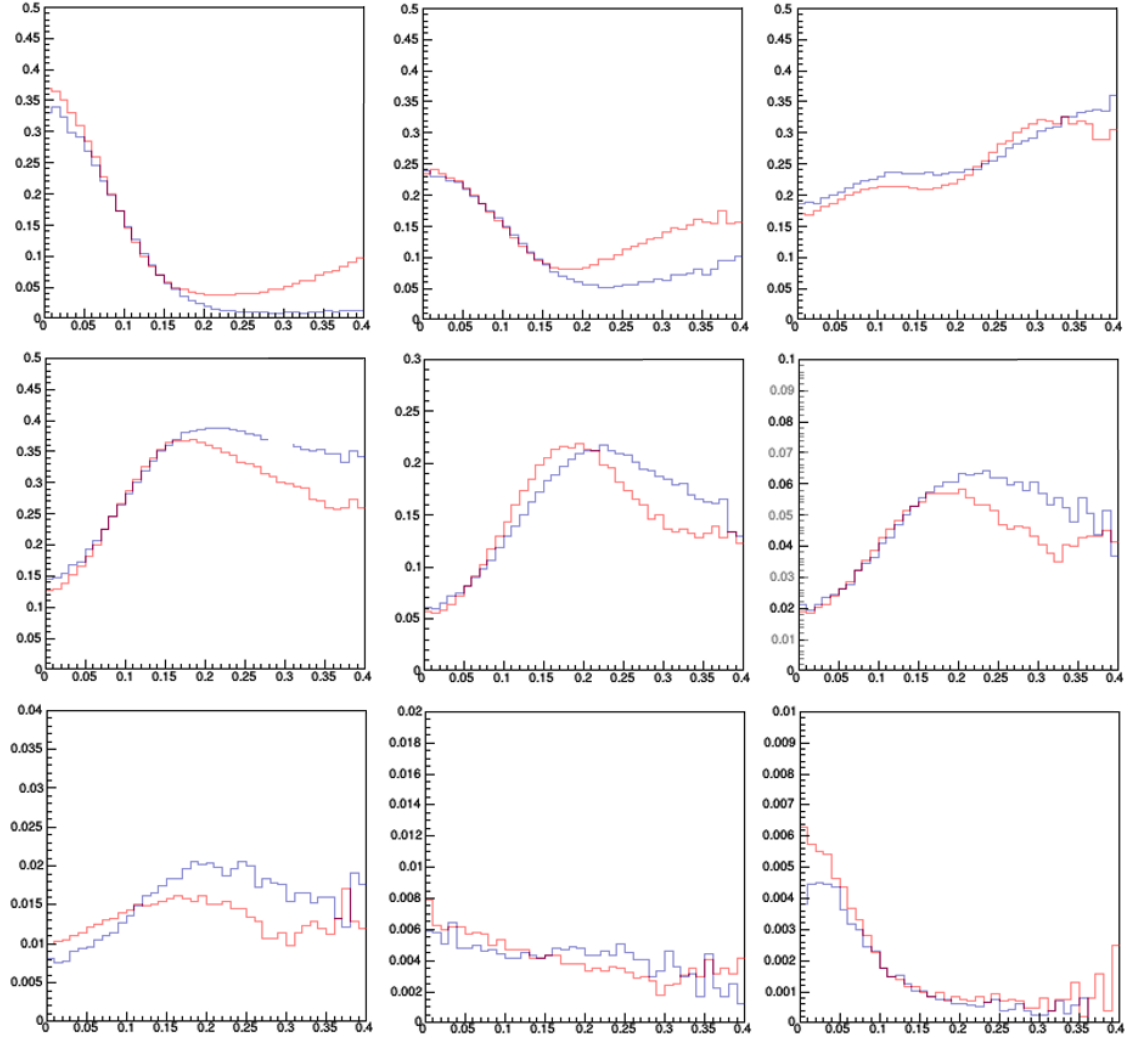


Figure 4.7: Efficiency vs beam angle for 0, 1, ..., 8 PMTs firing. The red curves represent data from detectors and the blue curves modified MC data.

Chapter 5

Implementation

This chapter describes the implementation of aforementioned methods, tools for its testing, analysis and optimization and at last the integration process of a trained classifier into the C++ physics analysis software PHAST with no dependence on a Python Interpreter.

Because of the nature of this classification problem, the development phase included examination of the preliminary results and adaption with respect to the findings. The most important discoveries that determined the direction of the next development will be presented in this chapter as well.

At last, some future suggestions especially concerning the AMBER experiment are presented .

5.1 Datasets preparation

The first step to this work was the preparation of data. In order to prepare data for training one must filter respective events from the ROOT file created by PHAST user event associated with CEDARs analysis.

Method `TTree::Draw()` available in ROOT can be used to select events passing a certain criteria. One can then loop over leafs of the tree (variables) and save the content of the ones that are of interest. This procedure is shown in list. 5.1 with a code excerpt of looping over ROOT tree and saving contents of the `beamX` variable from events that pass the condition upon `nBeamTracks` , `CE2Pressure` and `timestamp` to a file.

The background dataset as well as kaon proxy can be obtained in this fashion.

It is also advisable to convert the datasets into `.npy` format ties together with the Python Numpy package, which is optimized for reading and writing speed. For simplifying the conversion, a Python class called `FileConverter` was developed. It can convert a single text file or loop over a whole directory and convert all of its contents.

```

1 //access file and tree
2 TFile *f = new TFile("pscan2.root", "READ");
3 TTree* tree = (TTree*)f->Get("U32_CEDAR/USR32");
4 std::string condition = "nBeamTracks==1 && CE2Pressure>=10.20
   ↳ && CE2Pressure<=10.24 && timestamp>1540323600"
5 //save events that pass conditions to event_list
6 tree->Draw(">>event_list",condition.c_str(), "goff");
7 TEventList *list = (TEventList *)gDirectory ->
   ↳ Get("event_list");
8 tree->SetEventList(list);
9 Double_t beamX;
10 Byte_t CE1PMn,CE2PMn;
11 //set branch address of beamX to beamX variable address
12 tree->SetBranchAddress("beamX",&beamX);
13 std::ofstream outFile;
14 outFile.open (out);
15 //loop over events
16 for(int i=0; i<list->GetN(); i++){
17     //get entry from the event list
18     tree->GetEntry(list->GetEntry(i));
19     outFile << beamX<<std::endl;
20 }

```

Listing 5.1: Example code of preparing data from a ROOT tree.

5.2 New methods for particle identification

As presented in chapter 4.2, there are three methods to be implemented. The first one is a direct classifier, while the other two try to utilize a likelihood approach and hence require a model for both hypothesis.

All of these methods utilize the `CModel` class that implements a neural network together with methods for its training and predictions. Usually, each `CModel` instance will account for one CEDAR.

The class contains the following public methods:

- `__init__` - The constructor that initializes specified attributes and loads a model.
- `loadModel` - Loads a specified neural network model and assigns it to the instance that invoked it.
- `saveModel` - Saves the model currently linked to this instance.
- `trainModel` - Creates a new model and trains it according to passed parameters. Most of the work related to neural networks is performed by this method as elaborated in chapter 5.2.1,

- **evalModel** - Evaluates the model on a given test dataset.
- **clearTempDir** - Clears a directory used for saving partial progress of the training. It is called automatically upon training finish, but has to be called explicitly in case of an early termination.
- **predict** - Computes (and optionally saves) a prediction given by the current model for the passed dataset.
- **loadPrediction** - Loads a prediction from specified file.
- **setThreshold** - Sets the threshold above which particles are considered kaons.
- **computeThreshold** - Computes the threshold for which specified number of events are above in a given prediction.
- **getEffPoints** - Computes the threshold and purity for which the model has specified efficiency.
- **setEffPoint** - Sets the efficiency working point for this model.
- **setThresholdByEffPoint** - Sets the threshold to match the specified efficiency working point.
- **getAllEffPoints** - Computes the threshold and purity for which the model has all efficiencies in range of 1 to 100.
- **exportModel** - Exports the model into binary `.h5` file for further processing.
- **convertToJson** - Converts the specified model in `.h5` format to `.json` using the frugally-deep library.¹
- **appendToExportedModel** - Appends a list of the input variables and all working points to an exported model.
- **exportToJson** - Exports the model into the `.json` format readable by the C++ `CModel` class to be used in PHAST. Combination of the former 3 methods.

5.2.1 Neural networks implementation

As mentioned, TensorFlow or more precisely the included Keras module will be used for neural networks development.² In total, three network types were developed to be used in all methods. Number of input and output neurons as well as the input and output data are different for each method, but the same for each network type.

¹See chapter 5.6.

²Note that Keras exists as a standalone API as well and may differ from the one included in TensorFlow.

Plain network

The most straightforward approach is to build a simple deep feedforward neural network by linearly stacking hidden layers on top of the input layer. This is achieved by using `tf.keras.Sequential`. It also allows for adding Dropout layers, which randomly set input units to 0 with a specified `rate` at each step of the training process. This feature is used to prevent overfitting. Other inputs are then scaled by $1/(1 - \text{rate})$ in order to preserve the same sum over all inputs.

```
1 def __buildPlain(self, activations, dropRates):
2     #add normalization layer
3     mean, var=self.__compMeanVar(X_train)
4     layer = Normalization(mean=mean, variance=var,
5                             input_dim=self.structure[0])
6     #input and first hidden layer is different
7     model=Sequential([layer, Dense(self.structure[1],
8                                     activation=activations[0],
9                                     input_dim=self.structure[0])])
10    #loop from the second to the end
11    for idx, size in enumerate(self.structure[2:]):
12        if dropRates[idx]: model.add(Dropout(dropRates[idx]))
13        model.add(Dense(size, activation=activations[idx+1]))
14    return model
```

Listing 5.2: Example code for creating a plain neural network using TensorFlow.³

RBF network

```
1 def __buildRBF(self, X_train, betas, activation, dropRate, kmeans):
2     #first, select the initializer for the centers
3     if kmeans: initializer=InitCentersKMeans(X_train)
4     else: initializer=InitCentersRandom(X_train)
5     #create a RBF layer
6     rbflayer = RBFLayer(self.structure[1],
7                           initializer=initializer,
8                           betas=betas,
9                           input_shape=(self.structure[0],))
10    #build the model as: input layer-->RBF layer-->output Layer
11    model=Sequential([rbflayer,
12                      Dropout(dropRate),
13                      Dense(self.structure[2], activation=activation)
14                      ])
15    return model
```

Listing 5.3: Example code for creating a RBF neural network using TensorFlow and an external library [39].

³The normalization was originally performed manually as the `tf.keras.layers.Normalization` was still experimental in the version used at that time. It was implemented with it for easier use in C++ only after the other network types were discarded and hence no normalization layer is present in the list. 5.5 and list. 5.4.

In order to create a RBF network, one must reimplement `tf.keras.layer.Layer` with its constructor and three core methods: `__init__`, `build`, `call` and `compute_output_shape`. This layer was already developed in [39] and is used in this work⁴ together with its random centers and k-means clustering initializers. An example usage is shown in list. 5.5.

RVFL network

Unlike the previous cases, developing RVFL network cannot be done using `tf.keras.Sequential` because of the direct connections from the input layer to the output layer, known as skip or residual connections. However, building a model with non-linear topology can be achieved by using the functional API. A deep neural network can be interpreted as a directed acyclic graph and the functional API makes use of that: it offers tools for building a graph of layers.

```

1 def __buildRVFL(self, activations):
2     #define two identical input layers
3     input1 = Input(shape=(self.structure[0],))
4     input2 = Input(shape=(self.structure[0],))
5     #create the first hidden layer using input1
6     layers=[(Dense(self.structure[1],
7                   activation=activations[0],
8                   trainable=False)(input1))]
9     #add more hidden layers with input being the previous layer
10    for idx,size in enumerate(self.structure[2:-2]):
11        if dropRates[idx]:
12            layers.append(Dropout(dropRates[idx])(layers[idx]))
13            layers.append(Dense(size, activation=activations[idx+1],
14                                trainable=False)(layers[idx+1]))
15    #create the last hidden layer
16    if dropRates[-2]:
17        layers.append(Dropout(dropRates[-2])(layers[-1]))
18    x2=Dense(self.structure[-2],
19            activation=activations[-2],
20            trainable=False)(layers[-1])
21    #add the unused input neurons to the last hidden layer
22    conc = Concatenate()([input2, x2])
23    #create the output layer
24    if dropRates[-1]:
25        layers.append(Dropout(dropRates[-1])(conc))
26    out = Dense(self.structure[-1])(layers[-1])
27    #return model with the specified input and output nodes
28    return Model(inputs=[input1, input2], outputs=out)

```

Listing 5.4: Example code for creating a RVFL network using TensorFlow functional API.

⁴Some `import` statements in the code had to be updated for a newer TensorFlow version.

The process is then simple: we create two input layers and build an ordinary linear model using one of them (the second stays unconnected) while setting the `trainable` parameter to `False`. We append⁵ the unused input layer to the last hidden layer and connect both to the output layer, this time with `trainable=True`.

Training the network

Training of different networks is alike once the model is built. `CModel` class offers three validation approaches: simple validation split, k-fold cross-validation and stratified k-fold cross-validation. The latter two only differ in the instance `kf` passed as a parameter that can be either `sklearn.model_selection.KFold` or `sklearn.model_selection.StratifiedKFold`. A simplified method for training with (stratified) k-fold cross-validation could look like the following:

```

1  :
2  losses=[]
3  idx=0
4  #loop over folds
5  for train_index, val_index in
    ↪ kf.split(np.zeros(len(Y_train)),Y_train):
6  train_x = X_train[train_index]
7  train_y=Y_train[train_index]
8  validation_x = X_train[val_index]
9  validation_y= Y_train[val_index]
10 for x in range(iterations):
11     #checkpoint if better solution found for different folds
12     checkpoint = tf.keras.callbacks.ModelCheckpoint(
13         self.TEMPDIR+self.modelTitle+str(idx),
14         monitor='val_loss',
15         save_weights_only=True,
16         save_best_only=True,
17         mode='min')
18     callbacks = [checkpoint]
19     #fit the model, i.e. train it
20     history=model.fit(train_x,train_y,
21         validation_data=(validation_x,validation_y),
22         callbacks=callbacks,**kwargs)
23     #get best val loss value and save
24     losses.append(np.min(history.history['val_loss']))
25     idx+=1
26     #find best model
27     bestIdx=np.argmin(losses)
28 #load the best model weights
29 model.load_weights(self.TEMPDIR+self.modelTitle+str(bestIdx))
30 :
```

Listing 5.5: Example code for training a network.

⁵By appending we mean stacking neurons of both layers into a single one.

Setting a checkpoint and callback makes Keras save the best model after each iteration (each time the loss function improves between two epochs, the old file is overwritten). This way, the best model over all of the folds, iterations and epochs can be chosen and training phase can be interrupted without losing any of the progress. The temporary files containing the parameters are afterwards discarded upon normal termination or by calling the `clearTempDir` method.

5.2.2 Method 1: NN as a direct classifier

The first method takes the beam angle at CEDAR and the counter's response as the input. We usually create a model for each CEDAR and then combine both outputs using some transformation, i.e. addition, multiplication, logical AND etc. One could also create a single network with both CEDARs information as input, for which the `CModel` class can be used.

This might show problematic especially for the current data, though, because the signal proxy is obtained only from CEDAR 2 as the working pressures of both detectors differ. In the signal sample, CEDAR 1 response is replaced by CEDAR 2 response. The network might be able to pick up on this pattern and overfit. This is what was observed as shown in fig. 5.9 in chapter 5.5.1. Thus, one model per CEDAR is used.

```

1  :
2  #define datasets
3  trainingDataset="/path/to/datasets/trainingDataset.npy"
4  testDataset="/path/to/datasets/ttestDataset.npy"
5  #create a Classifier instance with No models
6  meth1Classifier=Classifier(None, inputCols1=np.r_[1:5,7:15],
7                               inputCols2=np.r_[1:5,15:23],
8                               labelCol=27, multiplicityCol1=25,
9                               multiplicityCol2=26, effPoint=0.6,
10                              DEFAULT=False)
11 #train and save models as "example-model1", "example-model2"
12 #it can be later loaded using Classifier.loadModels("example")
13 meth1Classifier.trainModels(trainDataset, "example")
14 #evaluate model and print results
15 print(meth1Classifier.evalModel(testDataset))
16 #generate predictions, average them and save
17 prediction="/path/to/predictions/example.npy"
18 meth1Classifier.predict(testDataset, prediction)
19 :
```

Listing 5.6: Example code for creating a model, training it, evaluating and generating predictions using the first method and the `Classifier` class.

Class `Classifier` ties together two models and contains among others the same methods as `CModel`. It allows for easy usage of both of the models at once by calling its methods. The prediction can be treated separately, or combined in a specified

way. This class also contains a method `analyze2models` that plots a histogram of the outputs of each model on one axis and computes number of events above threshold for each model and when combined. Example usage of this class is shown in list. 5.6.⁶

5.2.3 Method 2: NN as PMTs response predictor

Method 2 requires one model for the signal identification and one for the background identification. For working with these models, a child class of `CModel` called `PmtResponseModel` was created. It adds two more methods specific for methods 2 and 3. Method `compareProbs` computes the probabilities as shown in eq. 4.4 and method `getCondProb` returns a probability of a PMT response (method 2) or pattern (method 3) assuming it is induced by kaon or pion (depending on the training dataset used for the model).

Similarly to the previous case, a child class of `Classifier` called `M23Classifier` was created wrapping the two models in order to simplify the workflow. It adds several methods needed for handling the likelihoods, namely `meth2Predict` and `meth3predict`.

5.2.4 Method 3: NN as PMTs pattern predictor

The third method tries to resolve correlation present in the previous method by predicting the exact PMT response pattern. Therefore, the output layer consists of 256 neurons, each representing one combination of responses of the 8 photomultipliers. In order for method 3 to work, the PMTs response encoded as `byte` (see 4.3.2) is converted into a binary matrix with 256 columns filled with 0 and a single 1. This is achieved using `tf.keras.utils.to_categorical` after the dataset is loaded. Another option is to use `tf.keras.losses.SparseCategoricalCrossentropy` as the loss function. It accepts the labels as integer, each corresponding to a different class.

Because making predictions of 256 neurons is memory intensive, `meth3predict` performs these predictions in batches and converts them into a single value based on the PMT pattern that occurred for each of the models after each batch. Finally, it returns the ratio of these likelihoods, i.e. $\frac{L(\text{signal})}{L(\text{background})}$.

5.3 Analyses

As mentioned, the results are thoroughly analyzed in order to decide the direction of following development. For this reason, the most robust class eventually turned out to be `PredictionAnalyzer`.

⁶One can of course use two `CModel` instances instead, which in fact offers more flexibility as it allows specifying different function arguments for each model.

It works by tying it with an instance of the `CModel` (`PmtResponseModel`) or the `Classifier` (`M23Classifier`) class and the test dataset that was used to generate predictions (or that the analyzer should use for creating predictions). Afterwards, the prediction can be analyzed using number of methods or simply by running its method `analyzePrediction`. This method generates a report with numerous insightful information.

The class contains these public methods:

- `__init__` - The constructor that initializes specified attributes and generates prediction if none was passed.
- `getConfusionMatrix` - Returns confusion matrix of the prediction.
- `getConfusionStats` - Prints confusion matrix of the prediction together with some derived statistics such as sensitivity, sensibility, error rate and precision.
- `dividePredictionToBins` - Divides the prediction into specified number of bins based on its values and return the bins or its count together with the interval size.
- `plotHist` - Plots a histogram of the prediction with optional cut.
- `plotHistCurve` - Plots a histogram of the prediction divided into specified number of bins and fits a curve with optional cut.
- `cutAngleFilt` - Filters prediction and angle based on specified column and respective threshold value together with an operator ($>$, $<$ or $=$). This is especially helpful when distinguishing multiplicities in the analyses.
- `plotAngleScatter` - Plots a scatter plot of the prediction with respect to beam angle.⁷
- `getOutputWithPmts` - Get means of the prediction based on number of PMTs firing.
- `getOutputCumSum` - Calculate a vector of cumulative sum of the bin size (a number that describes the number of events that have bigger output than the bin maximum value), i.e. for bins=[1,3,6,15] the function iterates from the end and returns [25,23,21,15] .
- `plotAngleHist` - Plots a histogram of the prediction with respect beam angle.
- `plotAngleBins` - Plots means and RMS of binned prediction based on beam angle.
- `plotAngleHistBins` - Plots histograms of the prediction based on beam angle with means of bins and RMS (basically combination of `plotAngleHist` and `plotAngleBins` methods in one plot).

⁷Beam angle at CEDAR position is known for the MC simulations and estimated for the measured data using eq. 4.2.

- **computeForRoc** - Computes coordinates of points for a pseudo ROC curve (see chapter 5.5.1). For generating the points, `np.percentile` is used as it proved faster than a custom implementation which can still be accessed inside the `computeForRocUsingThreshold` method.
- **plotRoc** - Plots a ROC curve, possibly with logscale to measure the model performance.
- **plotMultRoc** - Plots multiple ROC curves into a single plot. This is used for models comparison.
- **plotMultRocDivided** - Plots multiple ROC curves into a single plot with the base being a specified curve. The base is used to divide all other curves by.
- **plotPredMultRoc** - Makes predictions with the same dataset using the passed models and plots the corresponding ROC curves into a single plot.
- **printSelectionVSMultiplicity** - Prints a table with number of total signal and background events before rejection, selected signal events, not selected signal events, selected background events and efficiency based on multiplicity and thresholds.
- **printStats** - Print some computed statistics about the prediction (see chapter 5.5.1)
- **computeForFalse** Computes points for a false positive curve (only for measured data).
- **plotFalse** Plots a false positive curve used to measure overfitting.
- **plotMultFalse** Plots multiple false positive curves into a single plot.
- **plotPredMultFalse** Makes predictions with the same dataset using the passed models and plots the corresponding false positive curves into a single plot.
- **computeForExpPur** - Computes the points of an expected purity curve (only for measured data).
- **plotExpPur** - Plots a expected purity curve used to measure overfitting.
- **plotMultExpPur** - Plots multiple expected purity curves into a single plot.
- **plotPredMultExpPur** - Makes predictions with the same dataset using the passed models and plots the corresponding expected purity curves into a single plot.
- **meth3plots** - Plots the prediction of the third method as a colormap in a eff. angle x and eff. angle y coordinates for a given set of patterns.
- **meth3plotsBySingleMultiplicity** Plots the prediction of the third method as a colormap in a eff. angle x and eff. angle y coordinates for some multiplicity. That is, group all patterns containing a specified number PMTs firing.

- **meth3plotsByMultiplicity** Plots the prediction of the third method as a colormap in a eff. angle x and eff. angle y coordinates for all PMT patterns for all multiplicities (256 figures).
- **meth3plotsBySymmetry** - Plots the prediction of the third method as a colormap in a eff. angle x and eff. angle y coordinates for grouped PMTs, such as 00000011, 00000110, ..., 1100000 and 11111000, 01111100, 00111110, 00011111.

For all of the plots, one can specify more parameters using `**kwargs`, which is directly passed to the corresponding plot function. A module `matplotlib` and especially `matplotlib.pyplot` is used for the plotting.

These classes were grouped into a package `cedarSeparation`. A scheme can be seen in fig. 5.1

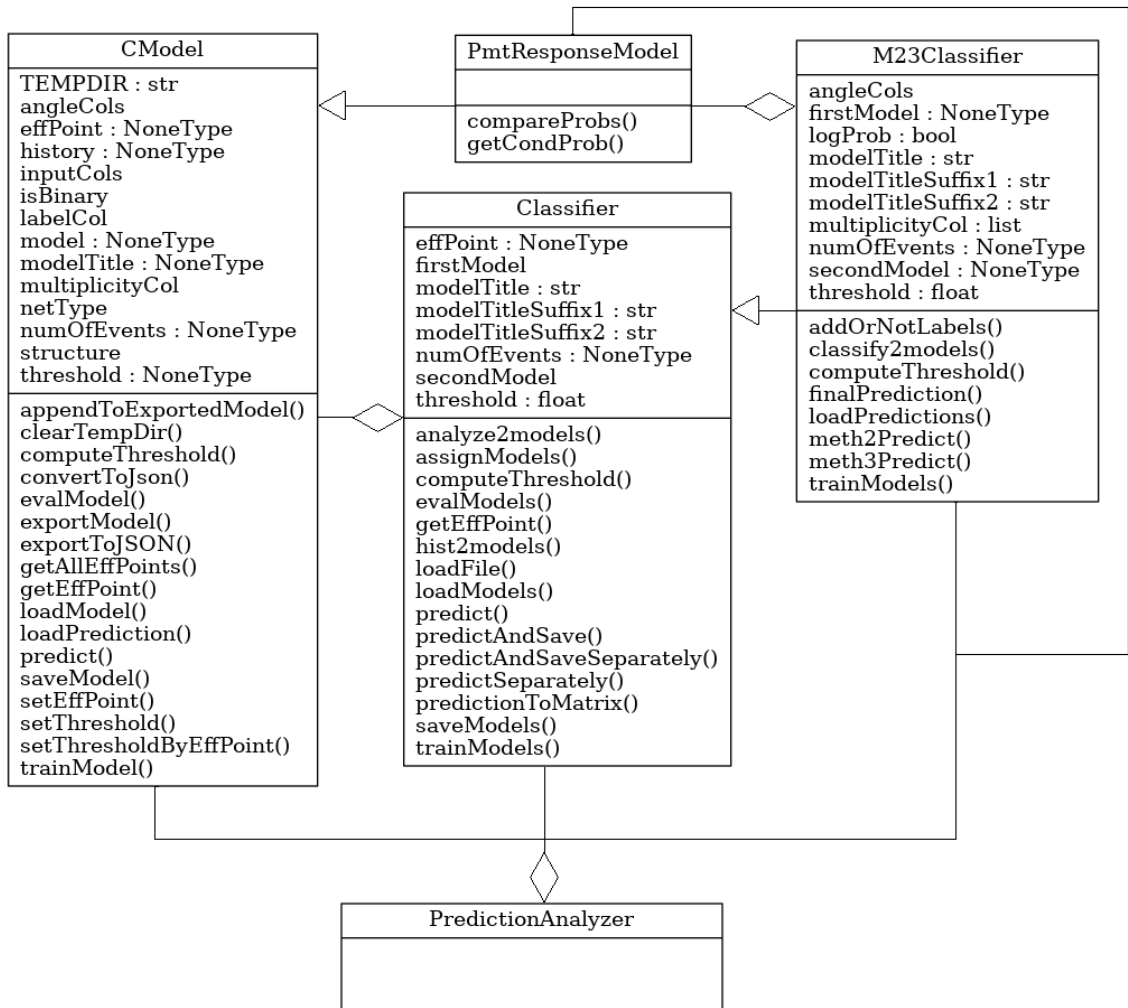


Figure 5.1: Class diagram of the `cedarSeparation` package.

5.4 Meta parameters optimization

Choosing the meta parameters of a neural network (i.e. the structure, activation functions, learning constants etc.) is no easy task. One can tune the parameters by trial and error or design a heuristic approach of some kind.

Both were tested in this work. A genetic heuristic inspired by biological evolution was developed. It mimics the laws of genetics with the goal of iteratively improving a solution.

We start by having a population and a fitness (or objective) function that measures the ‘quality’ of our individuals. Then, we create a new generation by combining the best individuals from the previous one.

The heuristic has the following steps: [23]

1. **Initialize:** Randomly generate a population of N candidate solutions. The properties (also called chromosomes or genotypes) are generated from predefined solution space that contains all permissible values (in this case all possible values of the meta parameters).
2. **Fitness:** Calculate the fitness (validation loss in our case) of all candidate solutions.
3. **Create a new generation:**
 - (a) **Selection:** Stochastically select a predefined number of chromosomes from the population. Individuals with better fitness are more likely to be selected.
 - (b) **Crossover:** Perform a crossover analogically to biological reproduction, i.e. combine each property of two individuals in order to create an offspring for the next generation that inherits traits of both parents.
 - (c) **Mutation:** Analogically to biological mutation, change the new offspring by randomly changing some properties with predefined probability in order to maintain genetic diversity from one generation to the next.
 - (d) **New generation:** Replace the current population with the new population.
 - (e) **Test:** Test whether the end condition (predefined number of iterations, fitness function value) is satisfied. If so, stop. If not, go back to Step 2.

A flowchart of this algorithm can be seen in fig. 5.2. Each iteration of this process is called a generation. The entire set of generations is called a run.

Parameters of the genetic algorithm must be tuned as well. For example, too small mutation rate may lead to genetic drift (change of frequency of some gene occurring) while too high mutation rate could cause a loss of good candidate solutions.

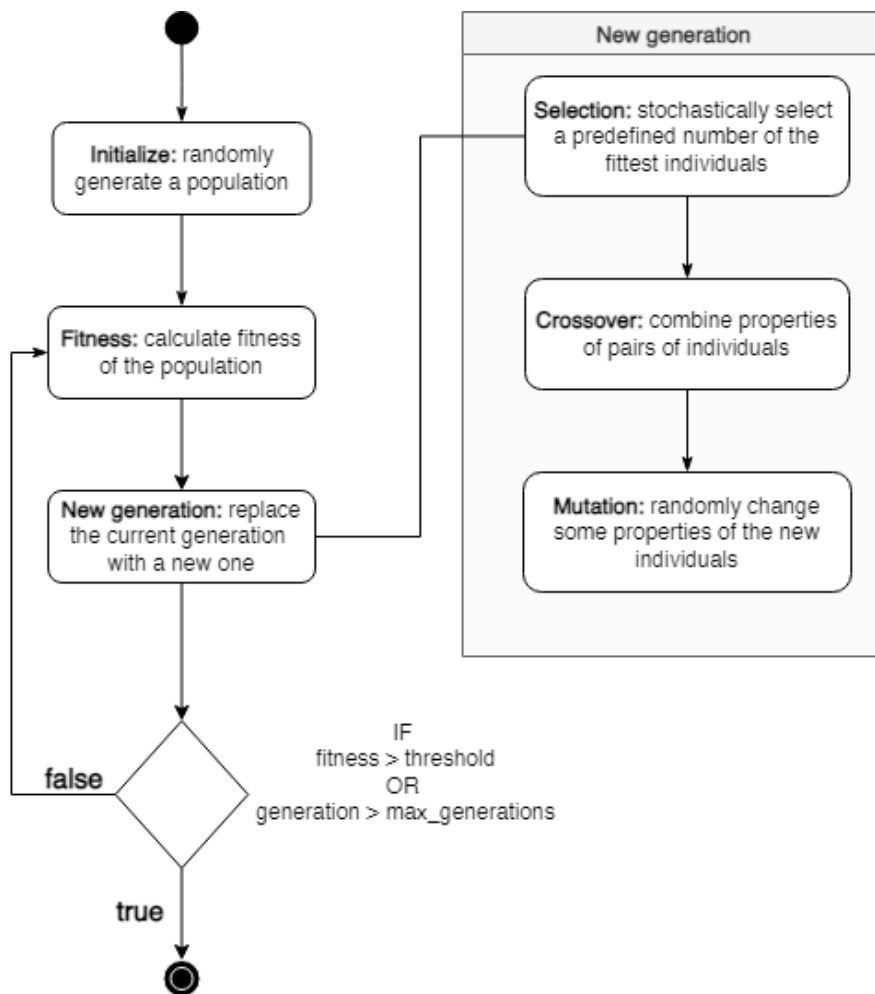


Figure 5.2: Flowchart of the genetic algorithm.

In this context, each individual is one neural network and therefore a training have to occur for each individual at every generation. For these reasons, HTCondor Service is used in order to run the algorithm.

Validation loss is used as a fitness function. In order to correctly compare individuals, a validation dataset is created and used for each network. In the end, a different test dataset is used to compare the result of the genetic algorithm and structures found by trial and error.

5.4.1 Running the algorithm

In order to run a script using HTCondor Service, the script is transferred onto the LXPLUS together with all the dependencies (plus training and validation datasets). The next step is to define a submit description file that specifies information needed by HTCondor to properly run the job. It contains path to the executable to run, input and output files, running duration etc. An example submit file can be seen in list. 5.7.

```

executable          = job.sh
arguments           = $(ProcId)
output              = out
error                = err
log                  = log
transfer_input_files = GADriver.py, CModel.py, genAlg.py,
    ↪ montecarloOrig.npy
when_to_transfer_output = ON_EXIT
request_GPUs        = 1
request_CPUs        = 1
+JobFlavour         = "testmatch"
queue

```

Listing 5.7: An example of the job description file to be submitted to HTCondor.

The next step is to write the executable `.sh` file. This file must begin with the line `#!/bin/bash` and contain installations of the used packages and the python scripts to run. For using TensorFlow with the current setup, one must also wrap the script in `scl enable devtoolset-9` to enable GCC 9. An example is shown in list. 5.8.

```

#!/bin/bash
scl enable devtoolset-9 - <<EOF
python3 -m venv myvenv3
source myvenv3/bin/activate
pip3 install tensorflow
pip3 install tensorflow-gpu
pip3 install matplotlib
pip3 install tabulate
python3 GADriver.py
EOF

```

Listing 5.8: An example of the executable file that installs required packages and runs `GADriver.py` script.

Finally, the script `GADriver.py` simply runs the genetic algorithm with population of 40 individuals for 500 generations as shown in list. 5.9.

```

1 from genAlg import genAlg
2 import numpy as np
3 import os
4
5 dataset=np.load(os.getcwd()+"/path/to/dataset.npy")
6 x=genAlg(dataset,40,500,verbose=0).run()

```

Listing 5.9: Example code of using genetic algorithm with the `genAlg` class. Note the appendix of the full path needed for HTCondor to correctly locate the dataset file.

After 500 generations with population of 40 individuals⁸, the best performing model was compared to a model with its parameters selected by hand. The results shown in fig. 5.3 appear very similar (the validation loss of the architecture selected by hand was circa 1 % lower), indicating that the problem seems to be insensitive to architecture of the network.

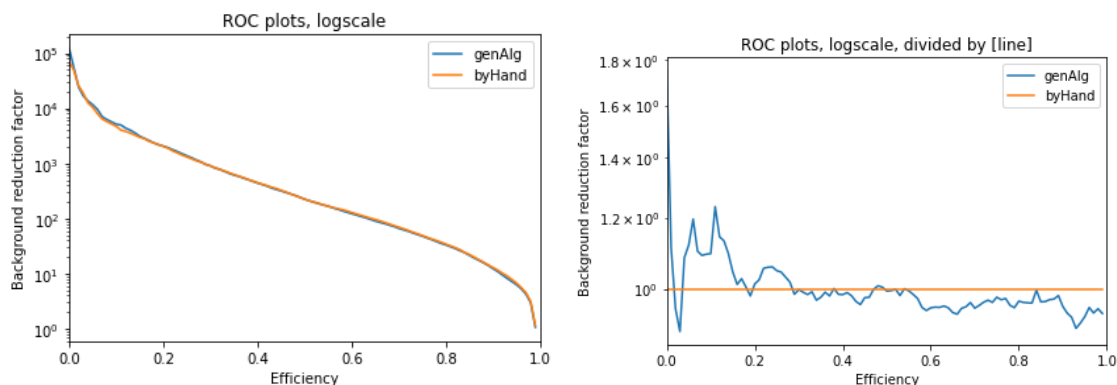


Figure 5.3: ROC curves of the best model created by genetic algorithm and by hand.

5.4.2 Network type selection

As mentioned, three network types were implemented. All were tested using `tf.keras.eval`. The results are shown in fig. 5.6 and 5.4. It shows that the plain network performs overall slightly better than RBF network in terms of validation loss, but significantly in terms of background reduction. The RVFL network performs the worst. The same conclusion was made by further inspecting the

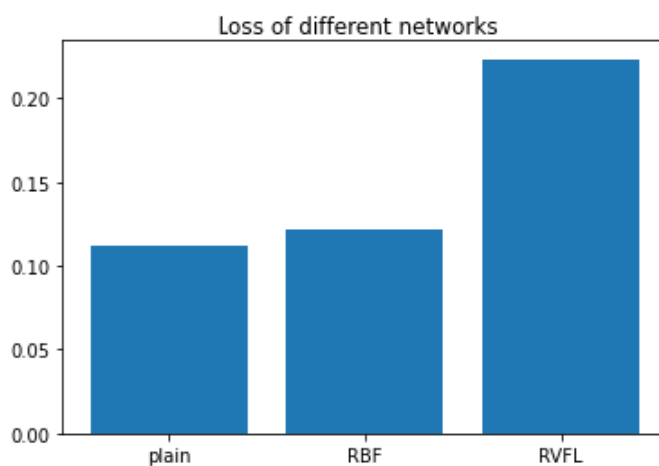


Figure 5.4: Values of loss function for a test dataset of different network types.

performance using the `PredictionAnalyzer` class and namely its `printStats`, `printSelectionVsMultiplicity` and `printConfusionStats` (fig. 5.5). At last,

⁸Multiple algorithm runs with different settings took place, all with similar results.

	plain	RBF	RVFL
True negative	483846	477976	471186
False negative	6088	6088	6088
False positive	3978	9848	16638
True positive	6088	6088	6088

Table 5.1: Confusion matrix of different network types at 50 % efficiency (sensitivity).

respective ROC curves (their meaning is further explained in chapter 5.5.1; generally the higher the curve, the better) in fig. 5.6 show this trend.

For analysis related to the first method, a plain network with 2 hidden layers containing 50 and 20 neurons was used. Furthermore, binary crossentropy, Nadam optimizer and swish activation function were selected. Learning rate was set to 0.01, betas to 0.99 and 0.999 and finally the drop rate to 0.2.

```

                predicted 0          predicted 1
-----
actual 0 true negative: 483846 false positive: 3978
actual 1 false negative: 6088  true positive: 6088
-----
statistic                                value
-----
sensitivity:                             0.5
background reduction                       122.63
accuracy:                                 0.979868
error rate:                               0.020132
specificity:                              0.991845
false positive rate                       0.00815458
precision                                  0.604808
Matthews correlation coefficient           0.539778
F score: F_0.5                            0.580473
F score: F_1                              0.547433
F score: F_2                              0.517951

```

Figure 5.5: Output of `PredictionAnalyzer.printConfusionStats` for the plain network. The other network types can be found in app. A.2.

5.5 Comprehension of gradual results

This section goes over some of the most important findings.

5.5.1 Estimating efficiency

The classification problem at hand has several problems. One being the incorrect labeling ought to be further examined. For the measured data, the sample is labeled as 0 even though around 2.5 % are signal events. When the proxy signal is added

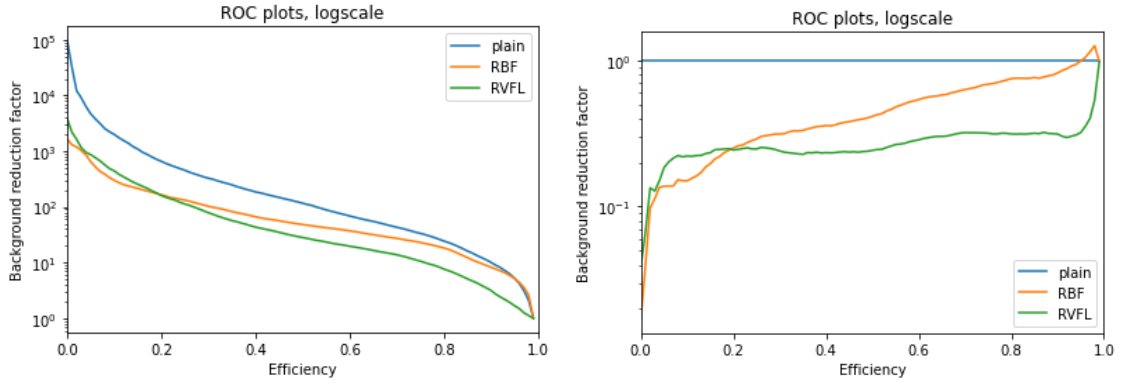


Figure 5.6: ROC curves of different network types. The plain network is used as the base for the second plot.

from the pressure cut, the whole training dataset still contains more incorrectly than correctly labeled kaons.

The problem does not exist in the available Monte Carlo simulations. In order to examine the effect of labels mismatch, the MC file was modified by duplicating the signal events and relabeling the duplicates to background. For a similar experiment, the kaons were not duplicated but half of them were relabeled. Both approaches led to a reasonably similar result (see fig. 5.5.1): the network’s ability to classify did not notably worsen, but the mean value dropped to half. This can be seen as a shift in the NN output peak as shown in fig. 5.8

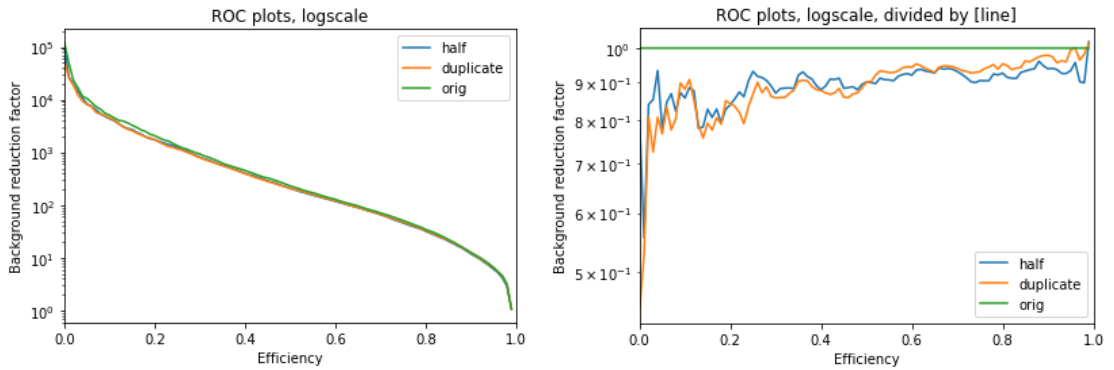


Figure 5.7: ROC curves of models trained on original dataset, dataset with doubled and relabeled kaons and a dataset with half of the kaons relabeled. The second plot uses the original curve as base.

This experiment represents a proof of work of using pressure cuts as the kaon proxy. More precisely this proves that the labels mismatch does not have a significant effect on the network training.

Another ancillary problem of labels mismatch concerns efficiency calculations. Because the labels in measured data are not fully correct, the efficiency can only be estimated. This is especially problematic for comparing models because the problem concerns validation and test datasets as well.

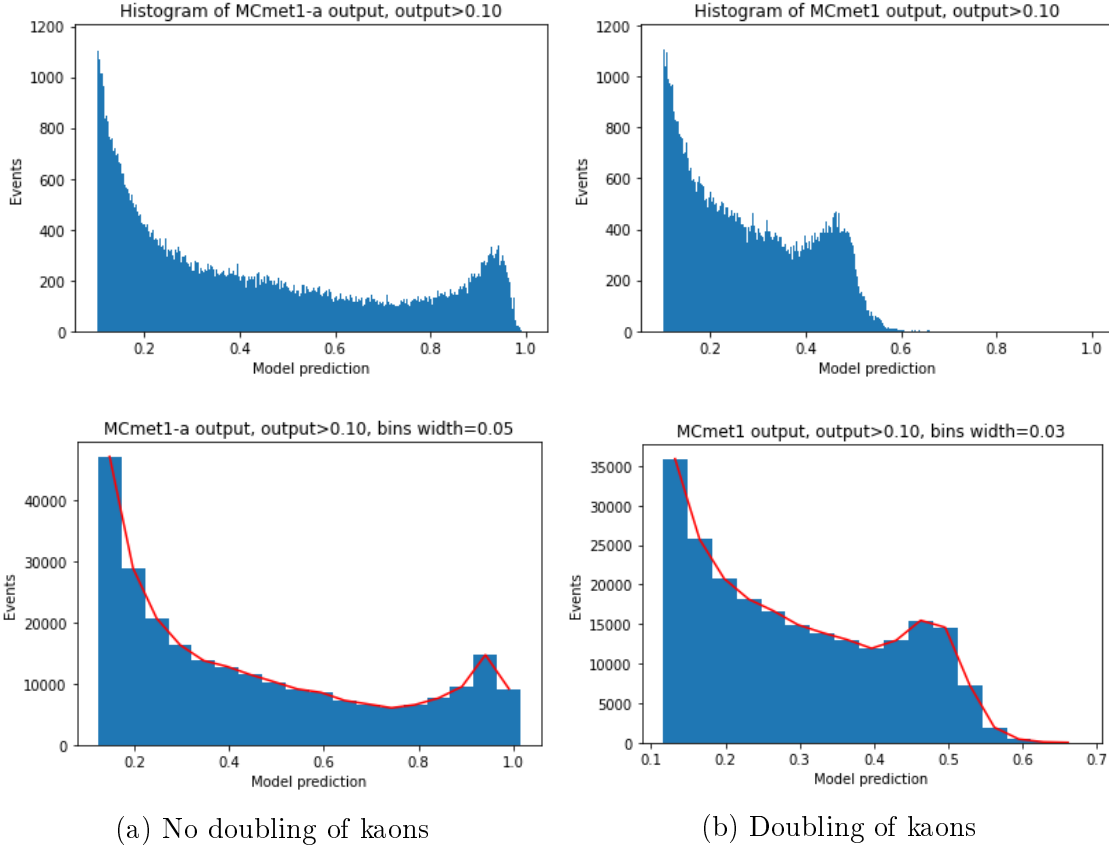


Figure 5.8: The shift in NN output when the incorrect relabeling is present in method 1. Note that only predictions higher than 0.1 are selected for these plots, which only accounts for $\approx 4.5\%$ of all of the tested events.

The efficiency estimations and subsequent calculations are performed as following:

1. Make a cut by setting a threshold, number of events to be selected or an efficiency working point (sensitivity)
2. Estimate efficiency as the ratio of selected signal events over all signal events:

$$E = \frac{\#\text{selected signal events}}{\#\text{signal events}}$$
3. Compute:

- Expected number of kaons in BG sample: $K_{BG} = 0.025 \cdot \#\text{BG events}$
- Expected number of kaons to be found by NN: $K_{exp} = E \cdot K_{BG}$
- Number of BG events labeled by NN as signal: K_{NN}
- ‘False positive’: $F = K_{NN} - K_{exp}$ ⁹
- Expected purity: $P_{exp} = \frac{K_{exp}}{K_{NN}}$
- Selection purity (Signal/BG): $P = \frac{K_{exp}}{F}$
- BG reduction factor: $R = \frac{\#\text{BG events}}{K_{NN}}$

⁹Typically, number of false positives cannot be negative by definition. In this context, we redefine this metric to better measure performance of a network with incorrect labels.

Problems with false positive

Because we know how many kaons are in the data sample without any cuts (K_{BG}), based on efficiency of the network we also know how many should be there after an arbitrary cut. Difference between the number of observed kaons in the selection and expected number of kaons in the selection is called 'False positive'. These should in fact be background pions passing kaon selection criteria. $F = 0$ means the selection contained as much kaons as expected, but there is no room for background events. Thus, background rejection should go to infinity at the same efficiency if all is consistent. When $F < 0$, the network is basically 'too good' in selecting the kaon proxy.

Similarly, P_{exp} , i.e. the ratio of expected number of kaons to be selected K_{exp} divided by the number of selected BG events K_{NN} . When $P_{exp} > 1$, describes how much the efficiency estimation is in fact overrated. For example, if $P_{exp} = 2$ and we expect to see 10 thousand selected events from the measured data, then NN only selected 5 thousand. In addition, not all of those events are indeed kaons. This results in efficiency overestimation, which is exactly what was observed.

For more insights, we frequently plot these 3 curves:

1. **ROC curve** - the background rejection factor for different efficiencies.
2. **False positive curve** - the number of false positives for different efficiencies. If all is well, $F > 0$.
3. **Expected purity curve** - expected purity for different efficiencies. If all is well, $P_{exp} \leq 1$.

	Method 1	Method 2
Threshold	0.437273	8.96904
#Selected events	15107	25000
#Events	2394660	2394660
#Events labeled as BG	2364350	2364350
#Events labeled as Signal	43311	43311
Signal/Background ratio	0.0180888	0.0180888
#Selected signal events	12658	12658
Efficiency: E	0.292258	0.292258
#Selected BG events: K_{NN}	2449	12342
#Events labeled as BG	2364350	2364350
#Kaons in BG: K_{BG}	59859	59859
K_{exp}	17494	17494
#False positive: F	-15045	-5152
Expected purity: P_{exp}	7.14332	1.41744
Selection purity - Signal/BG: P	-1.16278	-3.39557

Table 5.2: Efficiency estimations (for alike efficiency) for measured data showing the problems of using a kaon proxy.

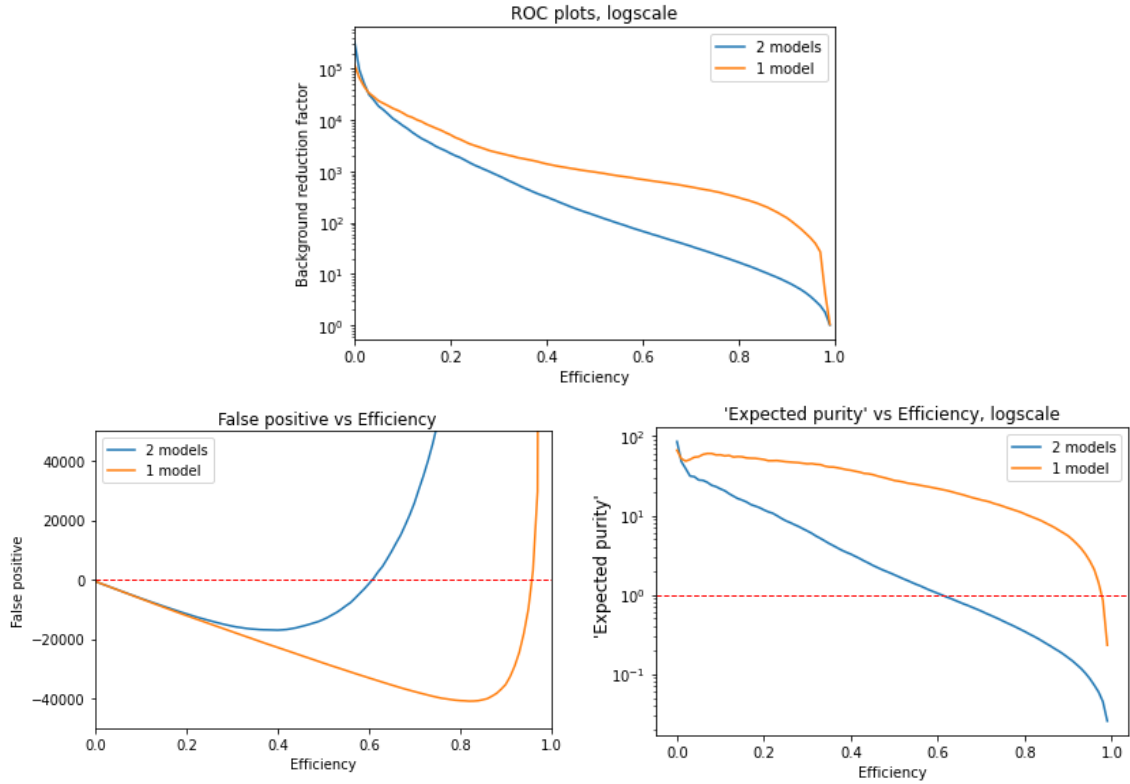


Figure 5.9: Using one model for each CEDAR vs one model for both detectors simultaneously.

These problems are apparent when using 1 model for both CEDARs versus using a model for each CEDAR as shown in fig. 5.9. While the first plot indicates better performance of the single model, the second and the third plot show that all the gain seems to be fake. Note that due to the low volume of kaon proxy, the dataset used for training is used for these predictions as well.¹⁰

Even with two models, the results are problematic as $F < 0$ for lower efficiencies. This is presented in tab. 5.2. For the second method, problems with overfitting seems to be smaller and only apparent when selecting fewer events (fig. 5.10). This can also be due to the method 2 being overall worse (the first plot in fig. 5.10) and having lower background rejection rate. Due to the lack of data, there is no way of telling which was the originator but as shown in fig. 5.18, method 2 performed worse on MC, so we can assume it was the latter.

Similar conclusion can be deduced from looking on network output for a mixed dataset (added kaon proxy) and a background dataset. One can see a ‘bump’ at model prediction value around 0.8 for the mixed dataset, but no such thing in the background sample. Histograms of both networks (each based on one CEDAR) show the same: a cluster at top right corner for mixed dataset is completely absent in the background data prediction.

¹⁰A validation dataset was extracted to use for the analysis, but the results did not differ from the ones obtained using all available data, so it was proceeded using the whole dataset. Formally, this could be an issue.

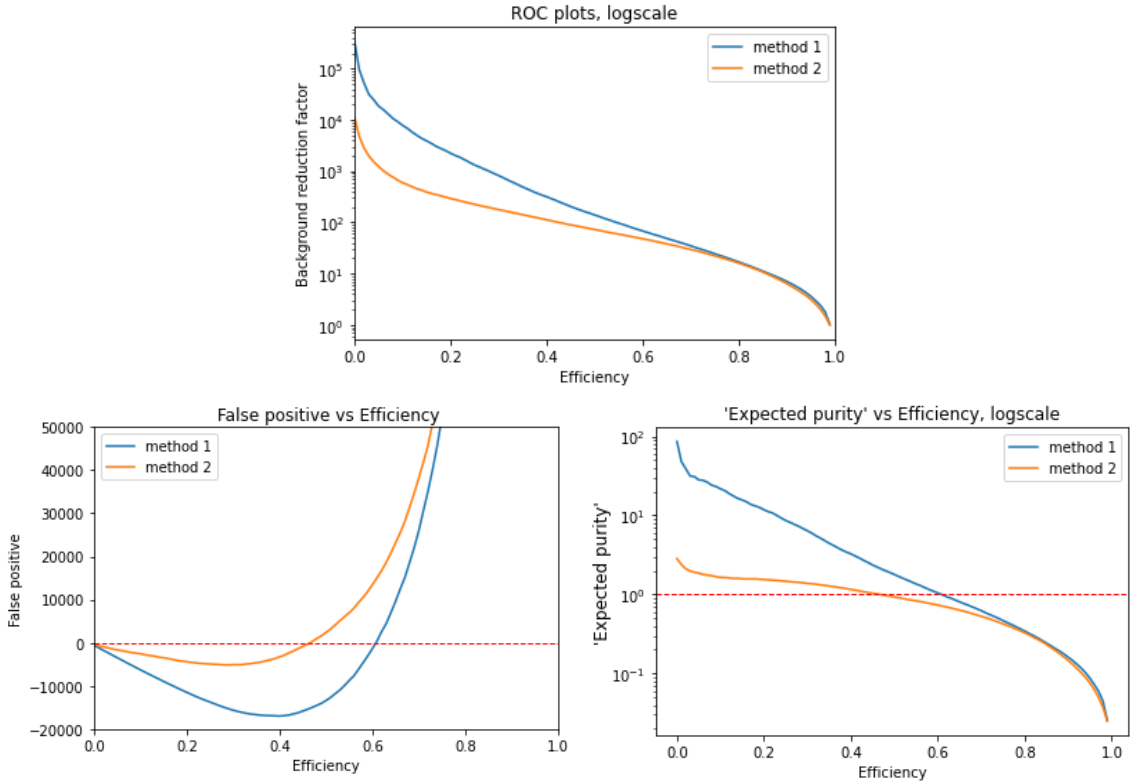
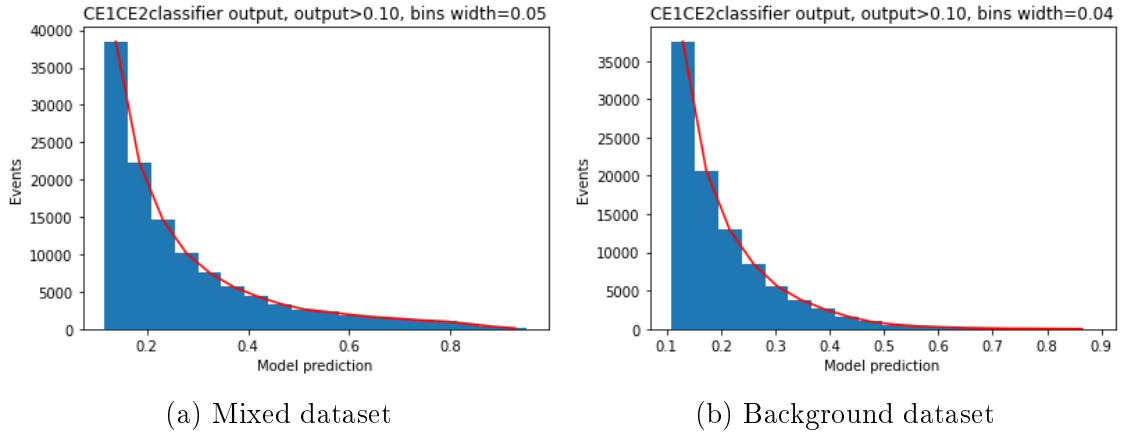


Figure 5.10: Methods 1 and 2 comparison on measured data.



(a) Mixed dataset

(b) Background dataset

Figure 5.11: Model prediction on different datasets. Outputs < 0.1 are discarded.

In order to improve the results, the pressure range from pressure cut was further reduced and examined. It was separated into four datasets within a pressure range of 0.01 bar to see if a signal sample of higher purity leads to an improvement despite the reduction of signal sample size for training.¹¹ The results shown in fig. 5.13 and in fig. 5.14 imply that except for the data in 10.20 - 10.21 bar, the ranges perform very similar. Narrowing the pressure range to 10.21-10.24 bar seems reasonable, but no significant improvement is to be achieved.

¹¹Non-intuitively, the best proxy would be the sample with the highest validation loss since it implies a closer resemblance to the real data.

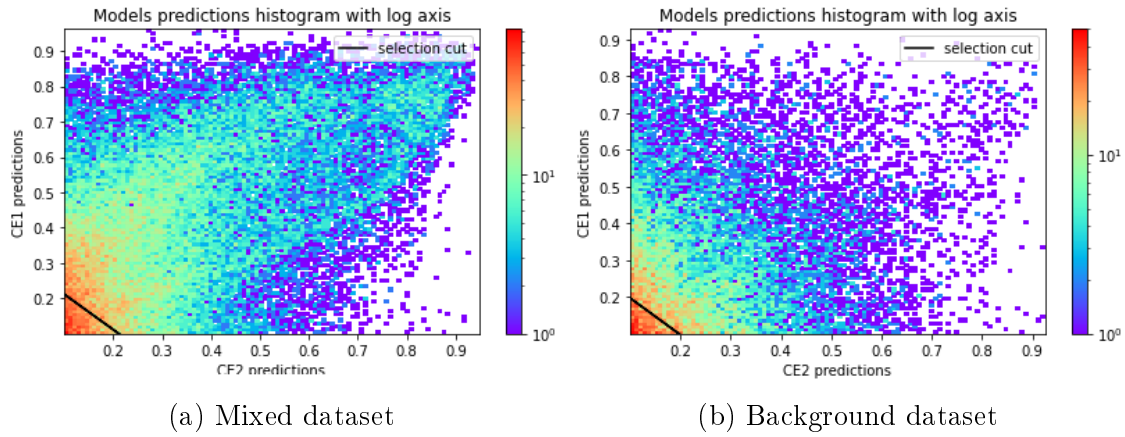


Figure 5.12: Prediction of a model based on CEDAR 1 (y-axis) and CEDAR 2 (x-axis) on different datasets. Outputs < 0.1 are discarded.

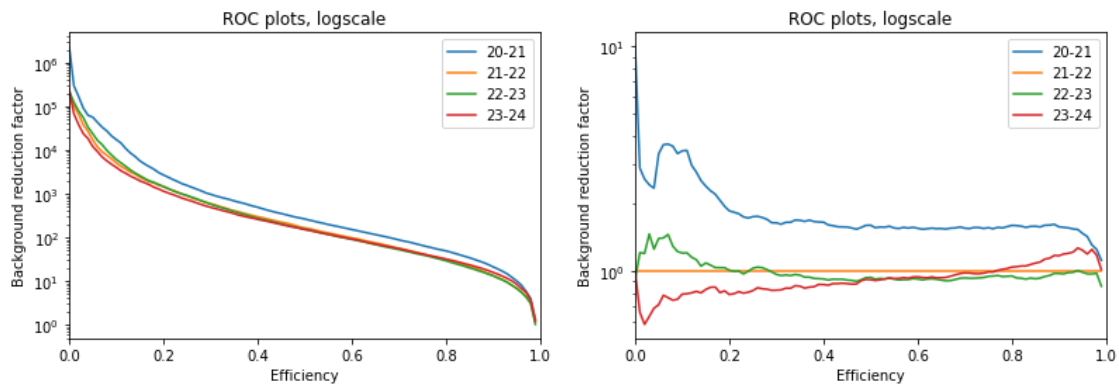


Figure 5.13: ROC curves for smaller pressure ranges used for kaon proxy. The second plot uses the orange curve as the base.

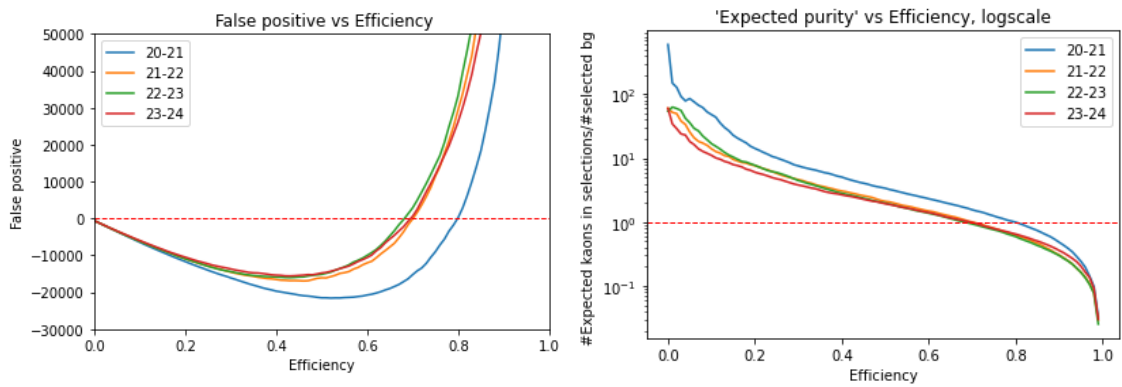


Figure 5.14: False positive and expected purity curves of the smaller pressure ranges used for kaon proxy.

5.5.2 Methods analysis

The methods outputs were further analyzed in order to inspect their behavior and find the best performing one. One of the interesting plots is the model output versus

the angle at CEDAR¹², filtered by number of firing PMTs. We should be able to see mean of the output to increase with higher multiplicities and reduce with bigger angles. This is shown in fig. 5.15. Especially at high multiplicities, outputs of the measured data are more scattered than MC.

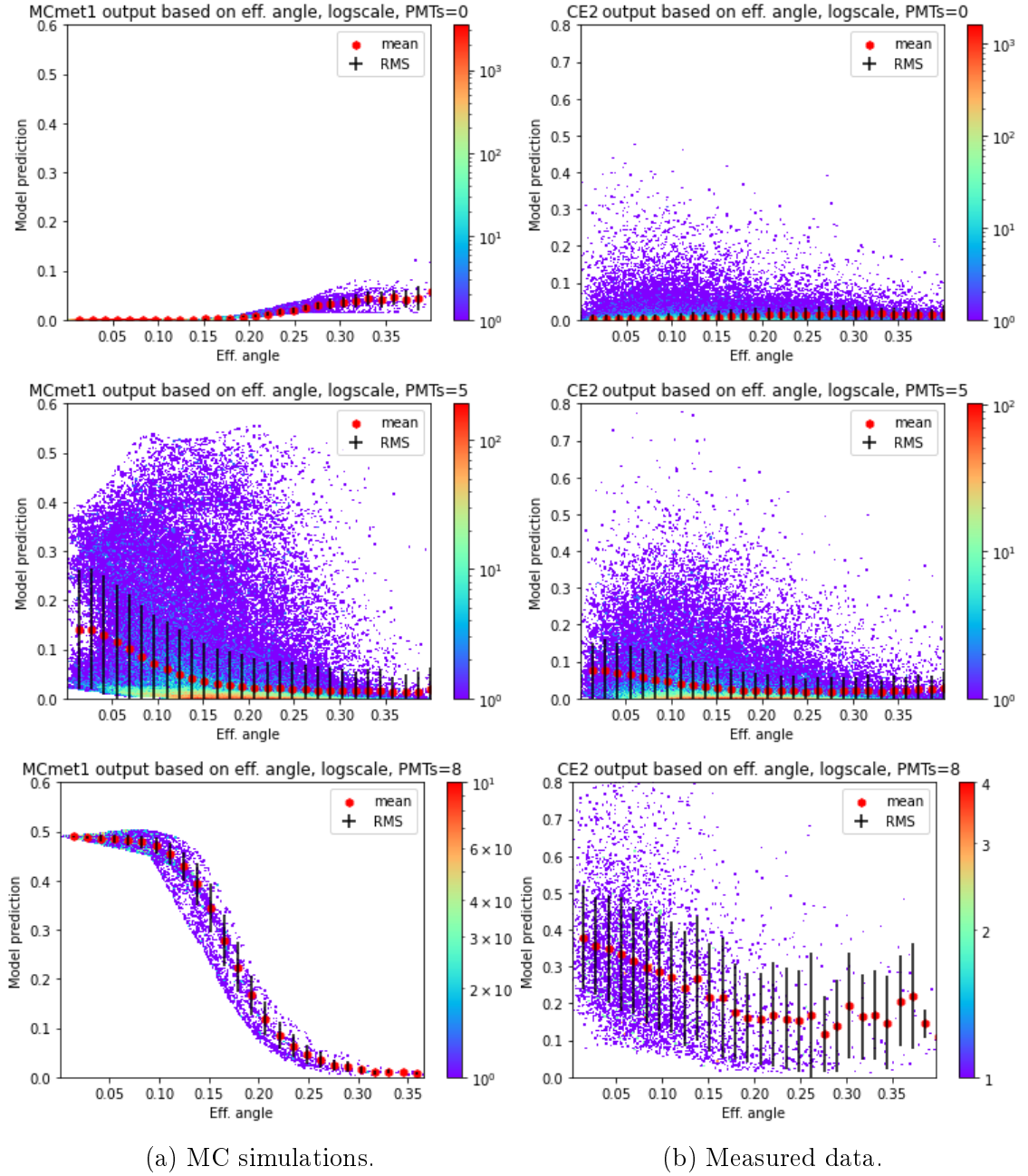


Figure 5.15: Output of method 1 based on eff. angle for MC and data of CEDAR 2 at multiplicities 0, 5 and 8.

Similar trends are observable for method 2. This is shown in fig. 5.16. The right side plots of both figures 5.15 and 5.16 were obtained using dataset without kaon proxy.

¹²Rather than the actual angle at CEDAR, the estimation as described in eq. 4.2 is used.

It uncovers that the methods work as expected and with higher multiplicities, more events are identified as signal. Plots of all multiplicities can be found in app. A.3.

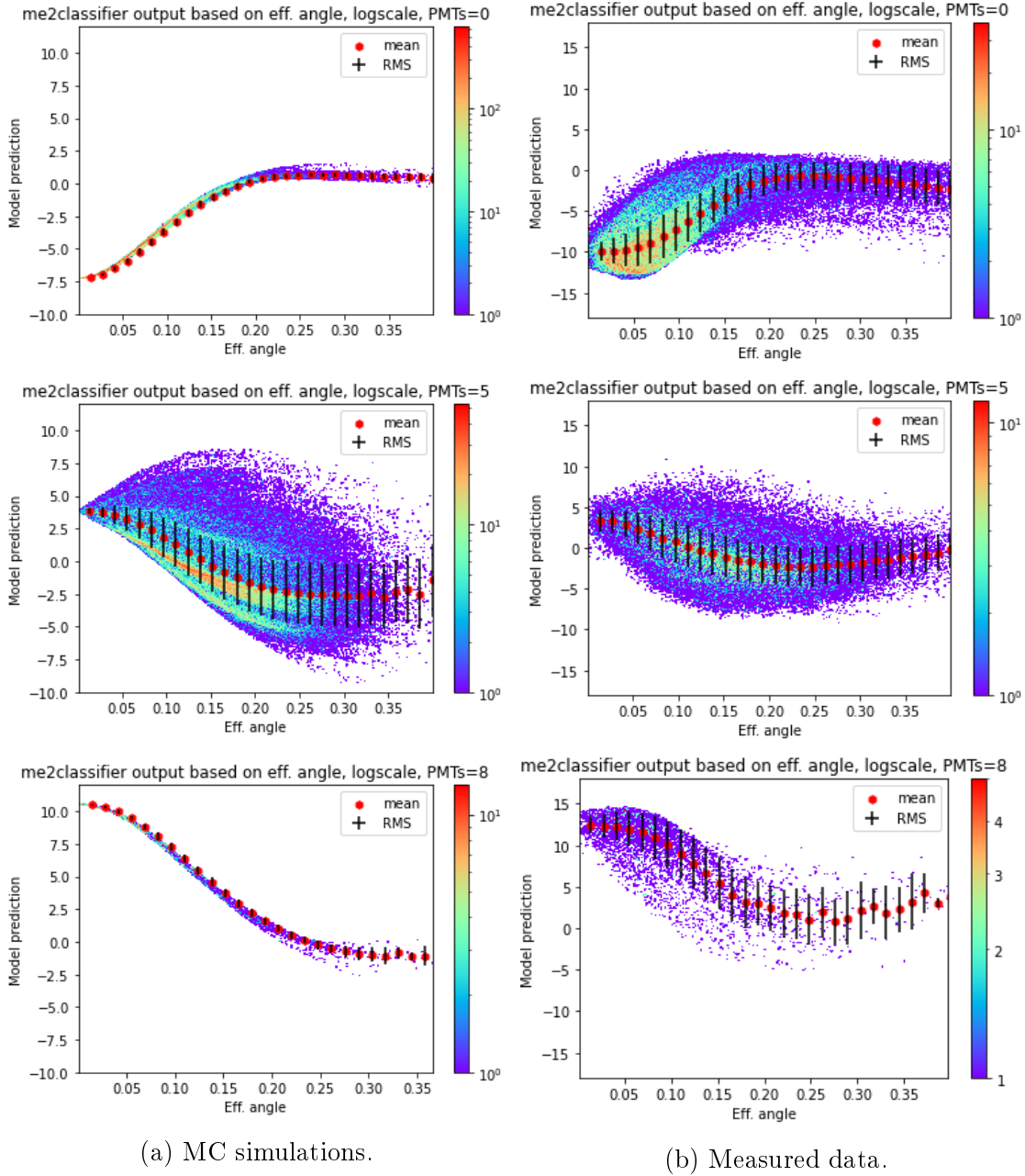


Figure 5.16: Output of method 2 based on eff. angle for MC and CEDAR 2 data at multiplicities 0, 5 and 8.

The third method could only be tested using Monte Carlo simulation, because it requires much more data for its training phase due to the output layer dimensions, but the trend was consistent. However, more insightful can be to plot the eff. x angle on x-axis, the eff. y angle on y-axis and finally the method output, i.e. $\log\left(\frac{p(K)}{p(\pi)}\right)$, on z-axis for combinations of active PMTs¹³. In fig. 5.17 one can observe the pattern

¹³This was done for all 256 combinations, but only the most interesting were included in app. A.4. The rest can be found on the attached CD.

‘rotating’ on the first three plots when different PMT fired. The last plot shows that for all 8 active PMTs, the output is higher with lower angles.

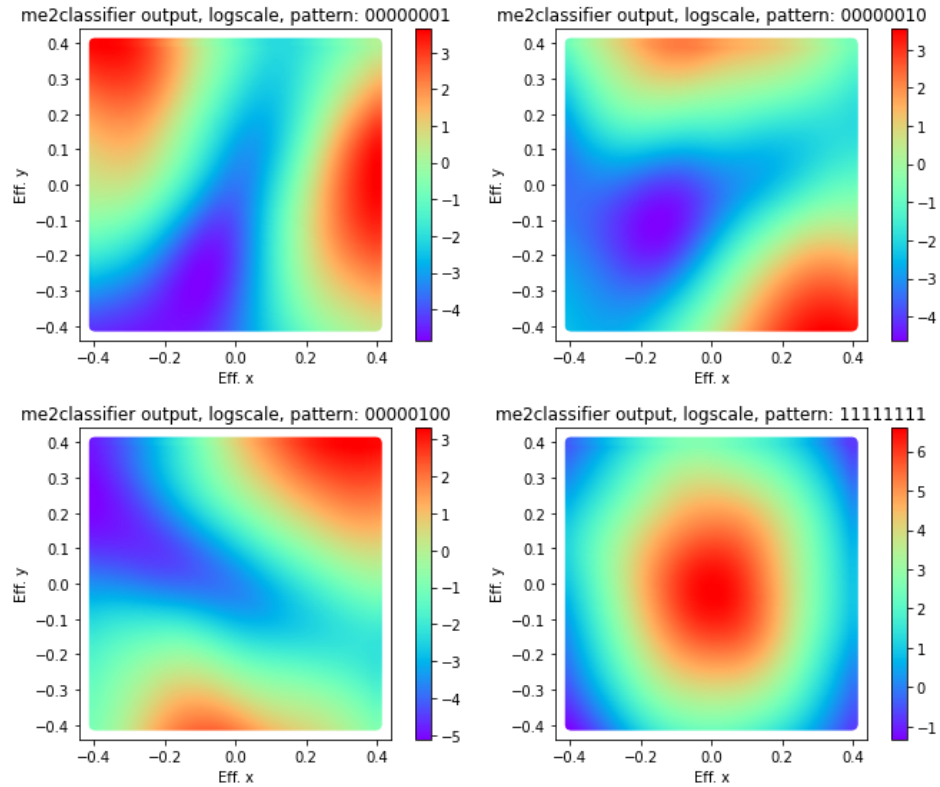


Figure 5.17: Output of method 3 as a colormap in eff. angle x and eff. angle y coordinates. First 3 plot contain events with one PMT firing, the last plot with all 8.

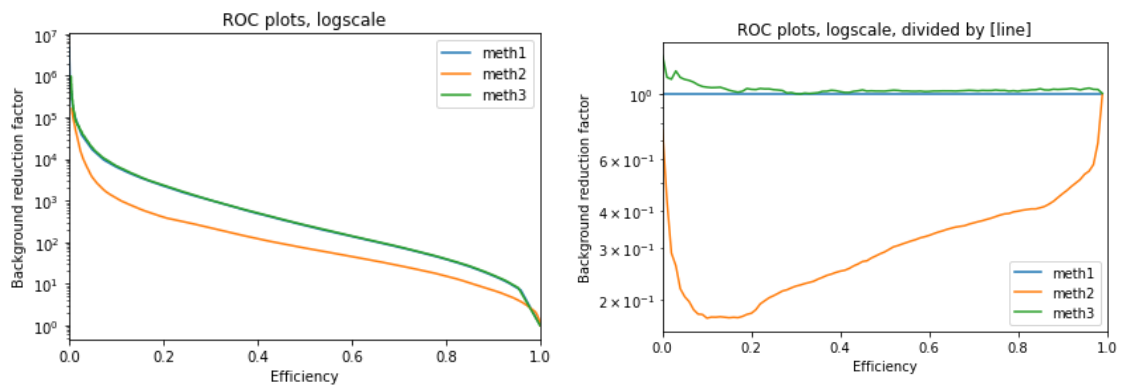


Figure 5.18: ROC curves of the three methods. For the second plot, curve of method 1 is used as the base.

The next step is to compare the methods. ROC curves mentioned earlier are a good indication of the method performance. Since as for the moment there are no good data to test and efficiency can only be estimated, MC simulations were used for the comparison shown in fig. 5.18. As expected, method 2 performed an order of magnitude worse in terms of background reduction due to the correlation not

taken into account. Surprisingly, method 3 gave basically the same results as the first method and yet no problems with stability were observed. That being said, the architecture contained far more neurons and the training, evaluation and prediction were more time and memory consuming. Because of that, the main focus is given to the first method.

5.5.3 Hybrid method

Because the kaon proxy from pressure cut seems problematic, a dataset containing measured background data and MC signal data was prepared. To estimate the angle at CEDAR for measured data, eq. 4.2 was used.

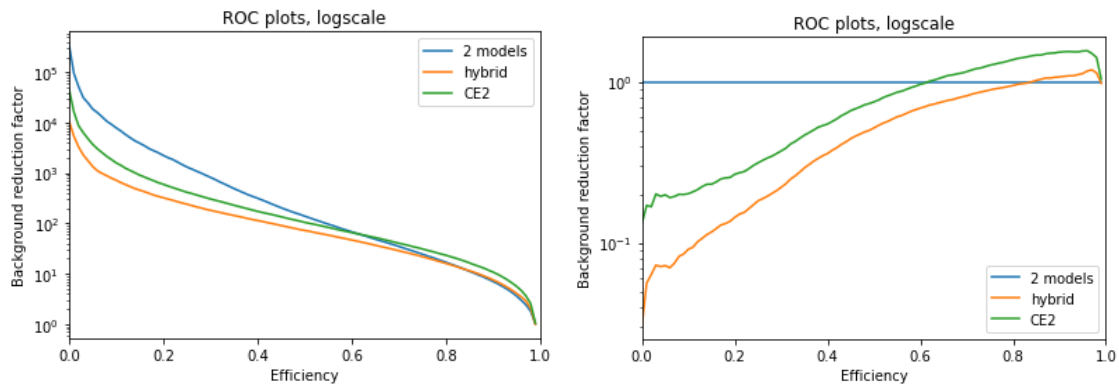


Figure 5.19: ROC curves of a model based on CEDAR 2, hybrid model and combination of models based on both dataset.

The efficiency estimations were compared with the ones of a single model based on CEDAR 2 and an average of models for each CEDAR as shown in fig. 5.19 and in fig. 5.20. It appears that the hybrid model performed the best in terms of overfitting and an improvement also offers using only CEDAR 2 at low efficiencies, but since the labels are incorrect, one does not have a certainty whether the used kaon proxy is better or the separation is simply worse.

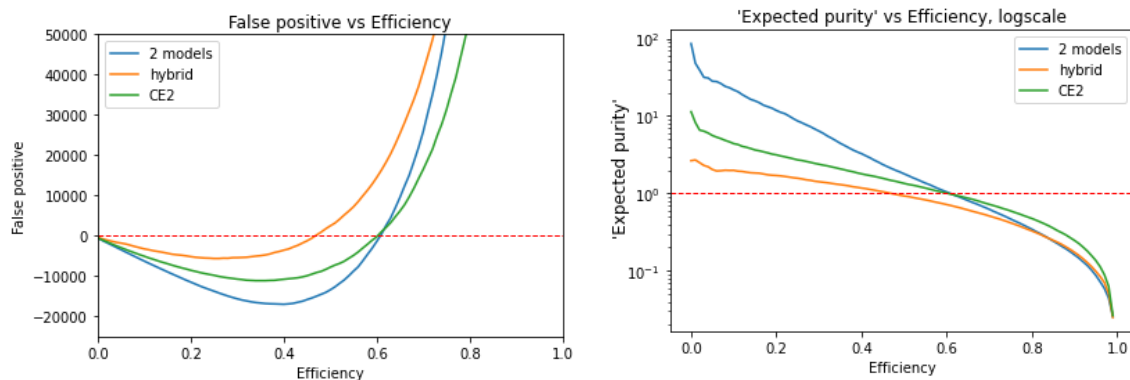


Figure 5.20: False positive and expected purity curves of a model based on CEDAR 2, hybrid model and combination of models based on both dataset.

5.5.4 Dataset size

Monte Carlo files that were being used contain ≈ 80 million events. Such datasets are very large and training using all available data takes a long time. The goal of this exercise was to determine whether there is a limit of improving the results with the use of more data. To do so, we start with using ≈ 5 million events for training and gradually halve it. The performance is then evaluated using a test dataset.

The results presented in fig. 5.21 can be biased, because other parameters were unchanged. Especially for batch size, this means that less weights and biases adjustments occurred. For this reason, in the next step, also the batch size was halved.

The results show that the improvements are only significant to around 300 thousand events. It also implies that the reduction of batch size for a smaller dataset can improve the overall performance to some extent. The next exercise was to also reduce the network itself with the dataset reduction to avoid overtraining on the small datasets. However, no difference was observed.

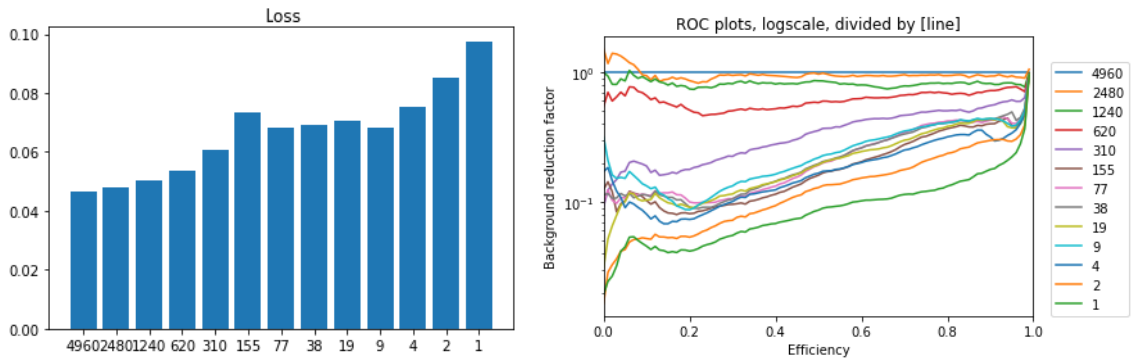


Figure 5.21: Values of loss functions and ROC curves of different dataset sizes (divided by curve of the largest dataset) used for training.

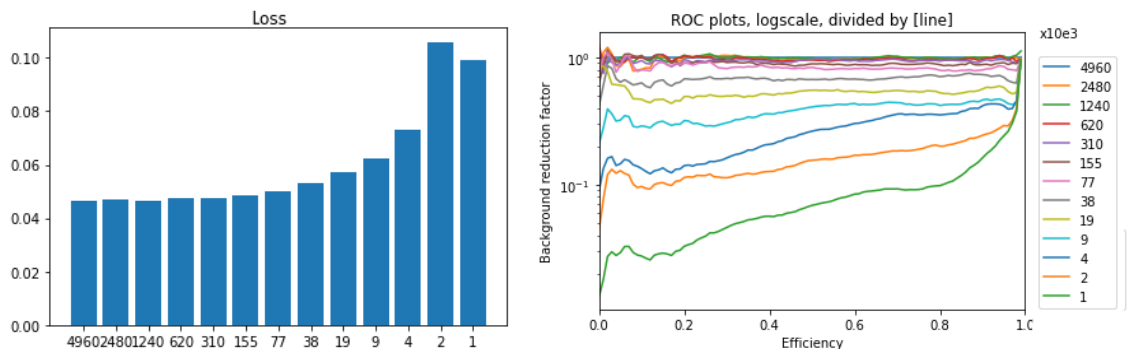


Figure 5.22: Values of loss functions and ROC curves of different dataset sizes (divided by curve of the largest dataset) used for training.

5.5.5 MC files analyses

In order to find what has the biggest impact on the network's ability to separate kaons and pions, 8 MC files containing all combinations of problems introduced in chapter 4.3.3 were examined individually.

The most impactful problem seems to be random noise and inefficiency (denoted by MC-010, see chapter 4.3.3). With added bad beam angle knowledge (MC-011), worse rejection rate than for MC-101 and MC-110 is obtained. This means that globally, additional track is less important from the point of view of the performance losses, despite the assumption.

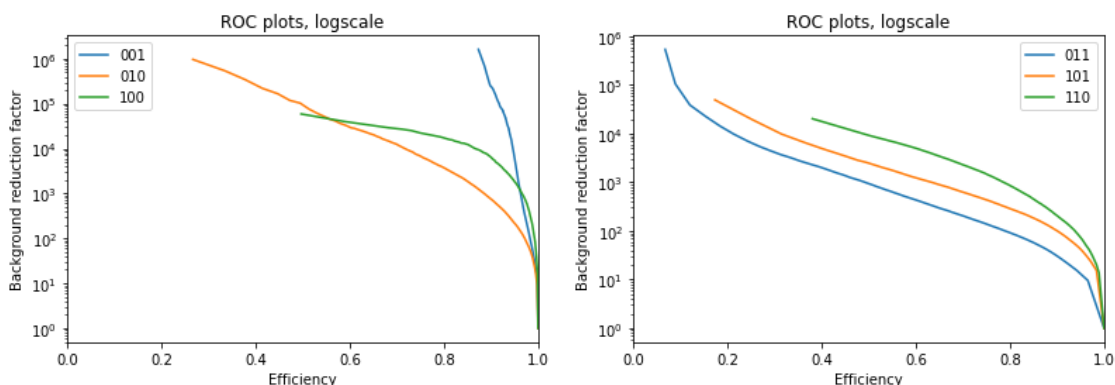


Figure 5.23: ROC curves of single problems and combinations of two problems. Curves ending prematurely imply that fineven very small change to efficiency leads to 100 % BG rejection and thus the factor goes to infinity.

However, for high multiplicities, the additional track leads to bad efficiency and purity. That being said, multiplicities 5, 6 and 7 seem problematic for MC-101 only. One can see the drop of efficiency and selection purity compared to MC-011 and MC-110. Note that the BG rejection rate does not worsen, in fact quite the opposite. Another observation is that almost half of the signal events at multiplicity 8 is lost due to inefficiency.¹⁴ This is shown in fig. 5.24.

On average, the sample contains around 35 times more pions than kaons, but this ratio differs for different multiplicities of fired PMTs (the 4th column in the table in fig. 5.24). This purity is very different between MC-110 and MC-101 in contrast to MC-011. Inefficiency seems to be a bigger problem than random noise, but these two modifications were done simultaneously. At the moment, CEDARs are assumed to have high efficiency of 93 %. The most surprising part is the high efficiency and selection purity at multiplicity 2 for MC-101, where the separation was expected to be more difficult compared to higher multiplicities.

The next step was to separate inefficiency and random noise to investigate their individual effects. For that, all files have to be examined on at a time again. Further, the notation is updated:

¹⁴Inefficiency of photomultipliers simulated by relabeling PMT response from 1 to 0 with some probability.

threshold=0.5									
#PMTs	#signal	#BG	purity	#selected signal	#selected BG	selection purity	#not selected sig	BG reduction	Efficiency
0	459	549097	0.001	0	0	nan	459	inf	0.000
1	7322	611523	0.012	54	38	1.421	7268	16092.7	0.007
2	32425	1287186	0.025	20453	4082	5.011	11972	315.3	0.631
3	31976	1646837	0.019	15660	5159	3.035	16316	319.2	0.490
4	25083	648228	0.039	19000	2193	8.664	6083	295.6	0.757
5	9079	90365	0.100	7074	1922	3.681	2005	47.0	0.779
6	5148	5665	0.909	4478	610	7.341	670	9.3	0.870
7	5144	176	29.227	5105	104	49.087	39	1.7	0.992
8	4697	2	2348.500	4697	1	4697.000	0	2.0	1.000

(a) Statistics for MC-011 dataset.

threshold=0.5									
#PMTs	#signal	#BG	purity	#selected signal	#selected BG	selection purity	#not selected sig	BG reduction	Efficiency
0	140	665240	0.000	0	0	nan	140	inf	0.000
1	3642	329306	0.011	497	254	1.957	3145	1296.5	0.136
2	34318	1155694	0.030	31728	1515	20.943	2590	762.8	0.925
3	25839	1827604	0.014	23526	4749	4.954	2313	384.8	0.910
4	28558	620699	0.046	23677	3020	7.840	4881	205.5	0.829
5	10031	147578	0.068	3121	1536	2.032	6910	96.1	0.311
6	6341	73570	0.086	1898	950	1.998	4443	77.4	0.299
7	3818	16995	0.225	1891	961	1.968	1927	17.7	0.495
8	8646	2393	3.613	8121	1014	8.009	525	2.4	0.939

(b) Statistics for MC-101 dataset.

threshold=0.5									
#PMTs	#signal	#BG	purity	#selected signal	#selected BG	selection purity	#not selected sig	BG reduction	Efficiency
0	395	476141	0.001	0	25	0.000	395	19045.6	0.000
1	6397	542522	0.012	4895	2088	2.344	1502	259.8	0.765
2	28504	1175559	0.024	25336	2510	10.094	3168	468.4	0.889
3	29977	1579152	0.019	26869	4449	6.039	3108	354.9	0.896
4	26258	758165	0.035	22973	3634	6.322	3285	208.6	0.875
5	12351	220253	0.056	10253	2659	3.856	2098	82.8	0.830
6	7030	69776	0.101	6219	933	6.666	811	74.8	0.885
7	5612	15650	0.359	5481	303	18.089	131	51.7	0.977
8	4809	1861	2.584	4783	176	27.176	26	10.6	0.995

(c) Statistics for MC-110 dataset.

threshold=0.5									
#PMTs	#signal	#BG	purity	#selected signal	#selected BG	selection purity	#not selected sig	BG reduction	Efficiency
0	395	476141	0.001	0	0	nan	395	inf	0.000
1	6397	542522	0.012	53	49	1.082	6344	11071.9	0.008
2	28504	1175559	0.024	18483	4732	3.906	10021	248.4	0.648
3	29977	1579152	0.019	13231	5806	2.279	16746	272.0	0.441
4	26258	758165	0.035	15868	3837	4.136	10390	197.6	0.604
5	12351	220253	0.056	3539	2143	1.651	8812	102.8	0.287
6	7030	69776	0.101	2095	1609	1.302	4935	43.4	0.298
7	5612	15650	0.359	3525	1630	2.163	2087	9.6	0.628
8	4809	1861	2.584	4373	695	6.292	436	2.7	0.909

(d) Statistics for MC-111 dataset.

Figure 5.24: Statistics of separation grouped by multiplicity (output of `PredictionAnalyzer.printSelectionVsMultiplicity`).

1. MC-1xxx: additional not detected track (correlated noise)
2. MC-x1xx: additional random noise
3. MC-xx1x: inefficiency
4. MC-xxx1: bad knowledge of beam angle at the CEDAR

Interpreting these results is problematic. It appears that when comparing a single effect, the most significant is the additional not detected track. As expected, combining the two biggest problems, i.e. adding not detected track and beam angle smearing, we receive the worst combination of two problems.

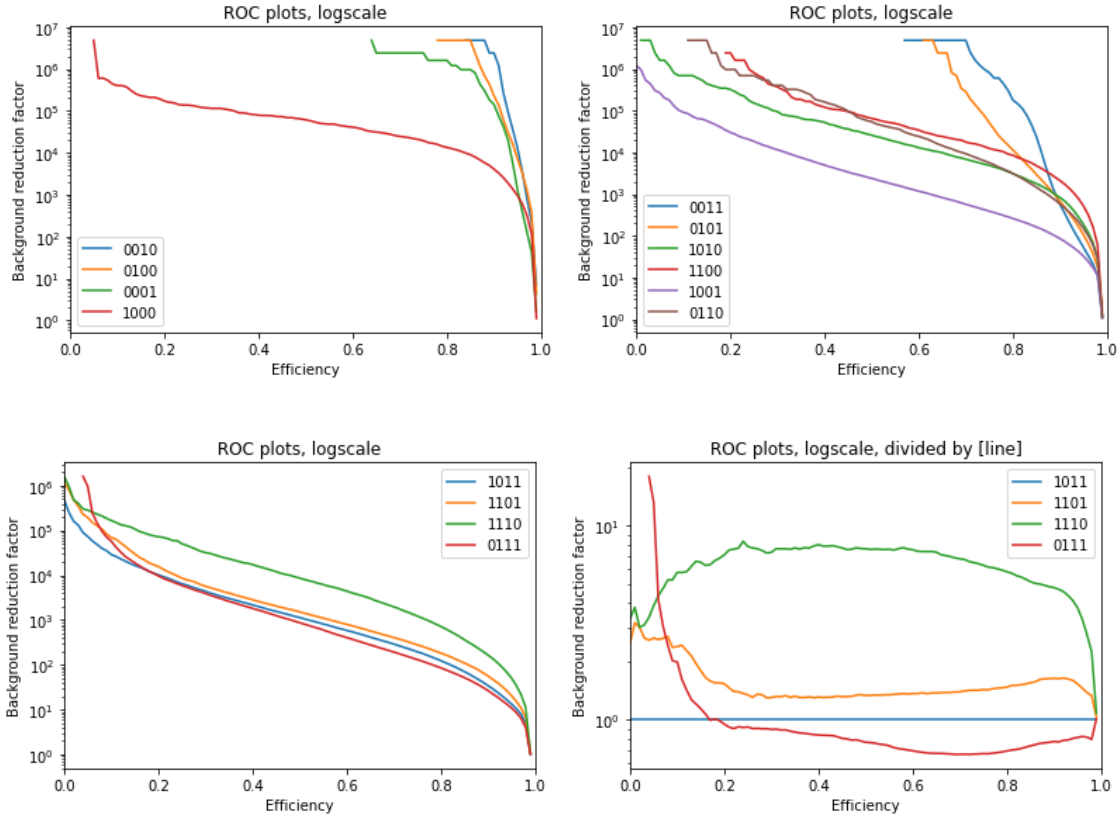


Figure 5.25: ROC curves of all combinations of a single, double and triple problems. The last plot is the same as the previous one, but divided by MC-1011 curve.

However, for combination of three problems, it is a completely different result. The worst performing is MC-0111, i.e. the dataset without the biggest single problem, additional undetected track.

Another issue that MC simulations show is the wide beam spread. According to CEDAR specification, the beam angle $dR = \sqrt{dX^2 + dY^2}$ should be below $65 \mu\text{rad}$ for a 190 GeV beam, while the current beam has RMS of dX and $dY \approx 120 \mu\text{rad}$ each. Thus only $\approx 10\text{-}15\%$ of events are within the designed radius.

In the original MC file before any damages, one can observe the efficiency peak moving in fig. 5.27. Events that trigger all 8 PMTs are mostly within the intended working radius. The network is then able to separate with ease. Even when the damages are performed, the separation depends on the angle greatly (see fig. 5.15).

These observations encourage the beam group to reduce angular spread of the beam. In fig. 5.26, classification improvements achieved by removing the most significant issue (combination of issues) is shown.

It appears that the best improvement can be achieved by removing the beam angle smearing (MC-1110). This shows the need for better beam angle measurements, hence usage of radiation resistant silicon beam telescopes.

However, if two problems were to be removed, better performance can be achieved by eliminating correlated and random noise.

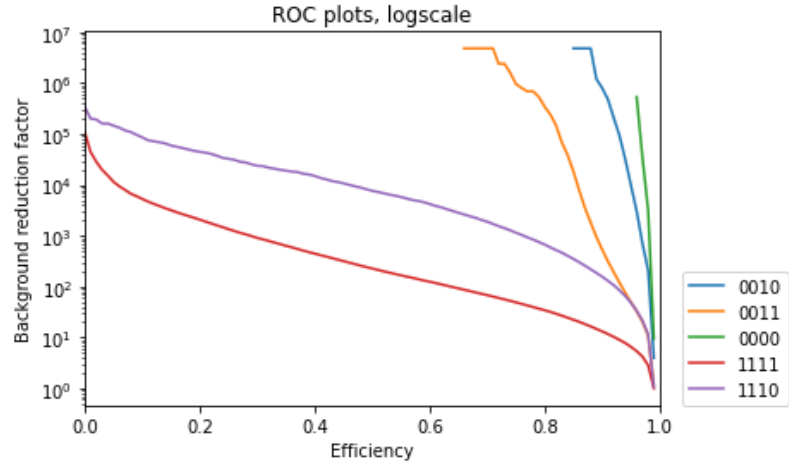


Figure 5.26: The biggest improvements achievable by removing 1, 2 and 3 problems.

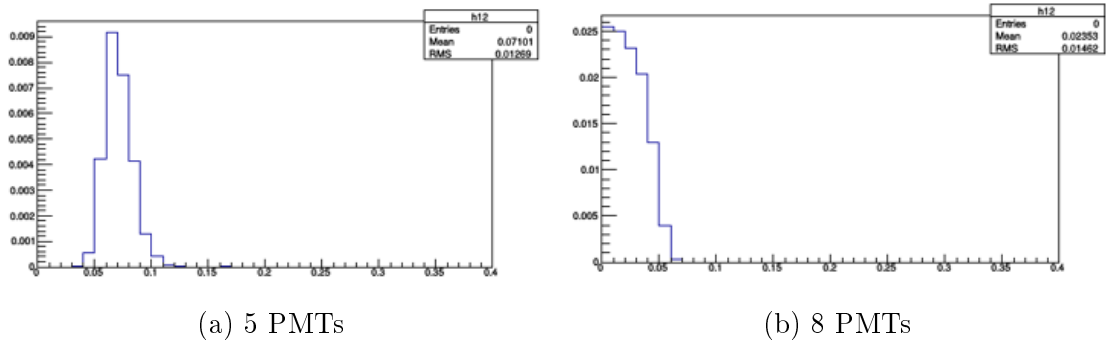


Figure 5.27: Number of events for multiplicity 5 and 8 divided by all events based on eff. angle in original MC with no ‘damages’.

5.5.6 Adding second track

So far, the easiest case was discussed, i.e. having only one particle crossing the CEDAR. This unfortunately accounts only for around 40 % of events. The next exercise is to test the model performance for cases with two tracks.

In order to do so, the MC file without additional undetected track (MC-0111) was modified by combining pairs of original events. For CEDAR response, logical OR was used and angles of both tracks were saved. Two situations were examined: one with the detected second track with its angle available to the network, and one without it.

The results in fig. 5.29 show that there was some improvement by knowing the angle, but overall the performance dropped significantly compared to the MC-0111 dataset.

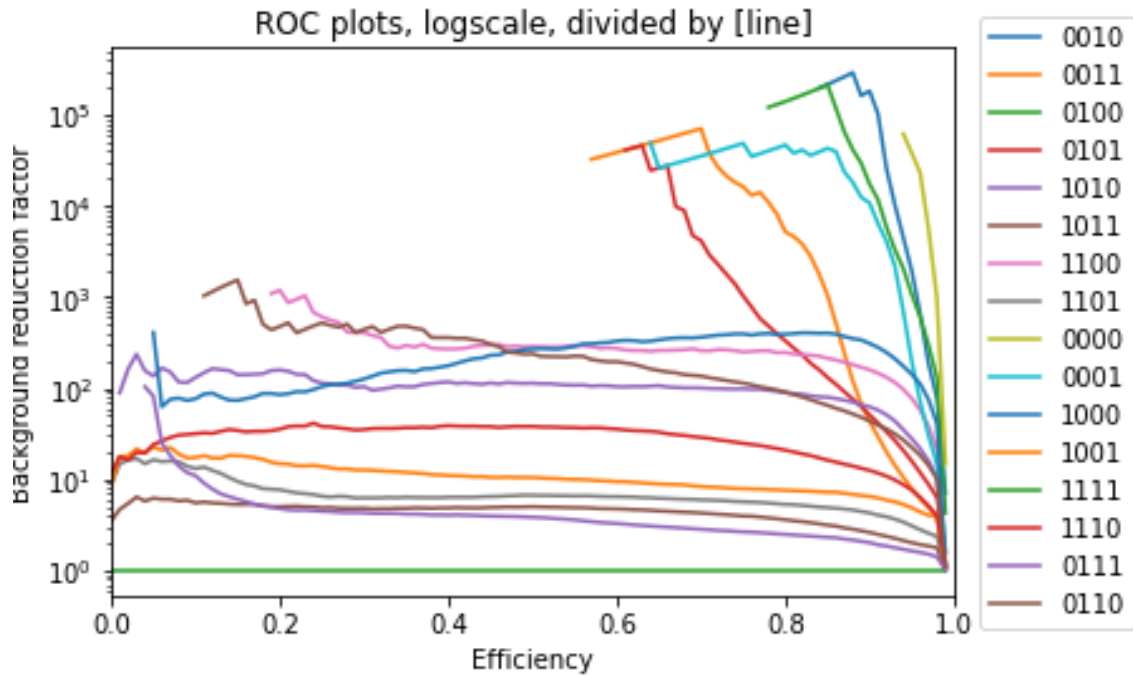


Figure 5.28: ROC curves of all problems combinations with MC-1111 as the base.

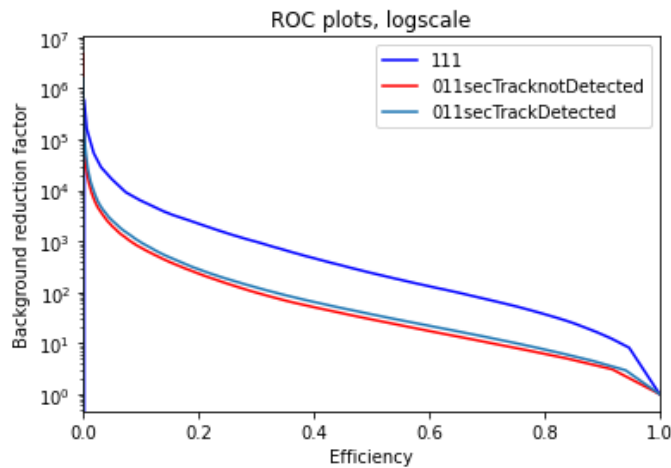


Figure 5.29: ROC curves of all MC-0111 dataset with added track and MC-1111 dataset.

5.5.7 Adding PMT pads to MC

So far, the PMT response was treated as a binary. In reality, each PMT consists of 4 pads that respond individually. Unfortunately, only the number of pads that gave signal was present in the original MC simulation.

The performance of a network that received this information was tested. It turned out that the results seem to improve significantly as shown in fig. 5.30, almost mimicking the ROC curve of dataset with the second track eliminated.

Even better results are expected when not only the number of pads were known,

but the pads were indexed and instead of 8 PMTs responses for each CEDAR we would have 32 responses of the pads.

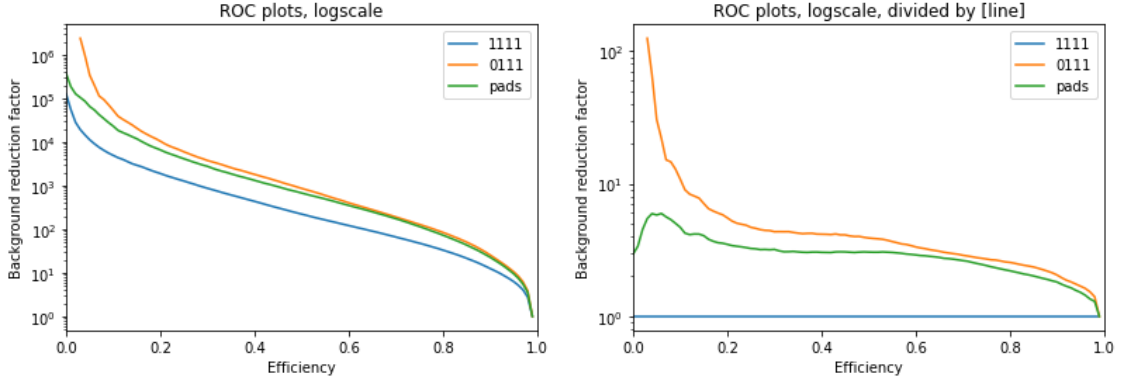


Figure 5.30: Comparison of ROC curves with and without using pads. The second plot uses curve of MC-1111 as the base.

5.5.8 Adding PMT pads to measured data

The responses of individual pads can be retrieved from data (unfortunately this information is not present in the MC simulations yet). In addition, the time when a pad gave signal is measured. For CEDAR 2, three models were tested. One with the individual pads responses, the second one included the time of the hits RMS, denoted by T_{RMS} , and in the third, events with $T_{RMS} > 0.8$ were discarded.

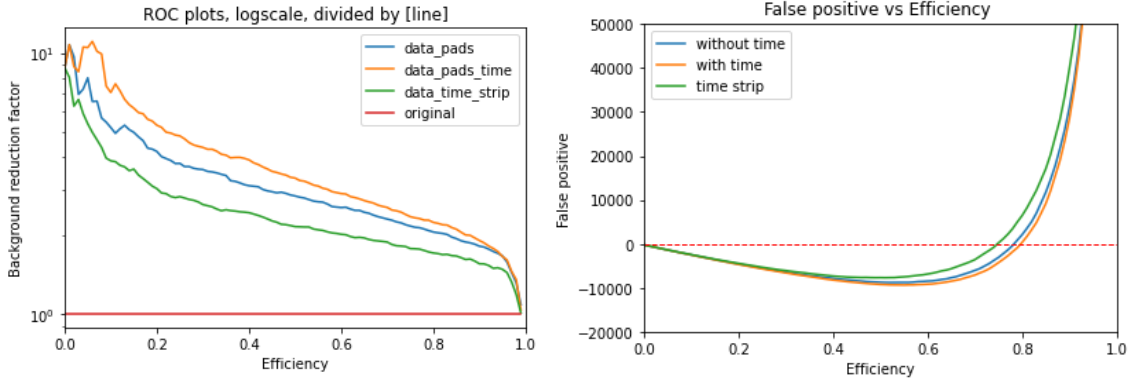


Figure 5.31: ROC curves of models with and without pads based on measured data, with the original model using binary PMTs response as base, and their false curves.

As fig. 5.31 shows, it appears that the more information is fed to the network, the more it is successful in selecting the kaon proxy. It is worth noting that when the network was presented with the hit time RMS, the mean of this variable in the selection for some high multiplicity (e.g. 5, which seemed the most problematic) lowered compared to the original dataset. This implies that the model was able to distinguish events caused by an undetected track to some extent.

Subsequently, the hit time was used to remove some off-time hits that were likely caused by a second undetected track, while keeping the original track. Once again, three datasets were inspected after the preprocessing: with individual pads, with number of pads firing and without any pad information. For each model, an alternative that included RMS of the hit time was examined as well.

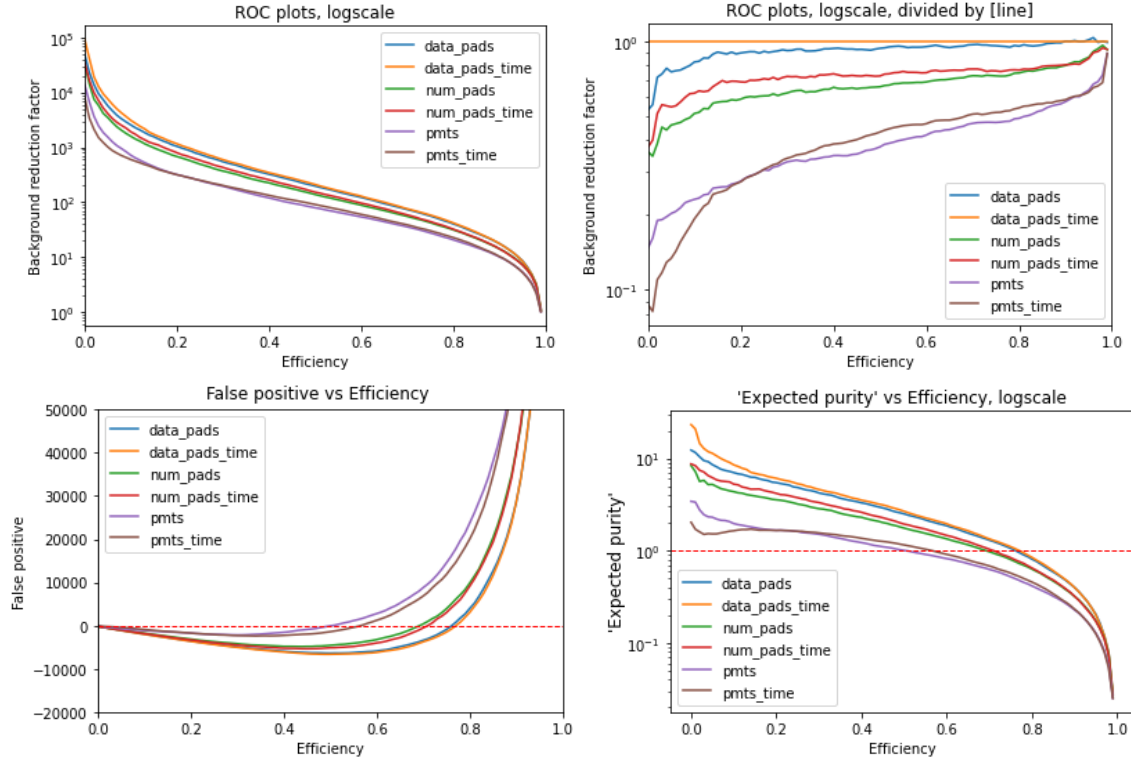


Figure 5.32: Adding individual responses of pad to the model.

The results show in fig. 5.32 show the expected trend, that is that the more information the model receives, the more it tends to select the kaon proxy. All of these exercise call for a better kaon proxy, further discussed in chapter 5.7.1.

5.6 Integration to PHAST

In the preceding work [41], a handful of different integration suggestions was presented both with and without introducing Python dependencies into PHAST. Since only inference is required in the C++ PHAST code, training the model in Python and then converting in for use in C++ with no further Python dependence was the preferred alternative.

For this purpose, frugally-deep library was used. After the model is trained, it can be saved into a binary `.h5` file. Frugally-deep then converts this file into `JSON` format readable for it in C++. Furthermore, all working points including efficiency with its corresponding threshold and purity together with the names of the input variables are appended to the `.json` file outside the main structure so that they are ignored by frugally-deep. These points are calculated using a specified dataset.

Upon training, a normalization of the dataset takes place. This is performed using a `tensorflow.keras.layers.Normalization` layer available outside the experimental package since TensorFlow 2.6 and supported by Frugally Deep since version 0.15.13. Using older versions requires performing the normalization by hand¹⁵ as was performed in the early stages of this work.

For usage in PHAST, a C++ class called `CModel`, which handles model loading and inference, was created. The constructor loads the model and runs automatic inference tests to ensure the same behavior as inside Python environment and prints expected input variables.

Overloaded method `isSignal` returns boolean value based on the specified efficiency and input in the form of `std::vector<FDEEP_FLOAT_TYPE>` or the model output. `FDEEP_FLOAT_TYPE` is set to `float` by default and can be changed to `double` by setting the macro. For debugging, a `VERBOSE` can be set to `true` to print additional information into the console. In the verbose mode, list. 5.10 would lead to an output similar to list. 5.11.

As illustrated, `CModel` can be used for inference in a standalone application. It can also be imported into a ROOT script processing a `TTree` and therefore into PHAST user event. For that, one must move the source codes of `CModel` class and frugally-deep library together with its required libraries (json, fplus and Eigen) into `lib` or `user` directory, where it is automatically compiled and linked upon building PHAST. Note that PHAST must be built using at least C++ 14 and so does ROOT.

```
1 #include "cmodel.h"
2
3 int main(int argc, char *argv[])
4 {
5     //create a CModel instance by loading json file
6     CModel model("/path/to/model.json");
7     //specify input vector
8     std::vector<FDEEP_FLOAT_TYPE> input{...};
9     //predict and compare to threshold of 50 % efficiency
10    bool isSignal=model.isSignal(0.5,input);
11
12    /******Alternative way******/
13
14    //predict and get the model output
15    auto result=model.predict(input);
16    //compare the output to threshold of 80 % efficiency
17    isSignal=model.isSignal(80, result);
18    return 0;
19 }
```

Listing 5.10: Example code of using a trained model in C++ standalone application.

¹⁵The normalization must be performed for every prediction using the same values as the ones used during training.

```

Loading json ... done. elapsed time: 0.001462 s
Building model ... done. elapsed time: 0.002796 s
Running test 1 of 1 ... done. elapsed time: 0.000340 s
Loading, constructing, testing of /path/to/model.json took
  ↪ 0.004999 s overall.
Loading working points...done. elapsed time: 0.010254 s
Input variables: (12 inputs in total)
beamX, beamY, beamdX, beamdY, PMT1, PMT2, PMT3, PMT4, PMT5,
  ↪ PMT6, PMT7, PMT8
Model output 0.1987298 < threshold 0.273611
Model output 0.1987298
Model output 0.1987298 > threshold 0.115984 --> signal event
  ↪ identified!

```

Listing 5.11: An example output of a code that uses the `CModel` class.

In the user event where one wishes to use the network only the `CModel` class must be included, in the standard way.

The execution time of the `CModel.predict` method was tested for different sizes of the network. For a network with 100 neurons the prediction took ≈ 0.5 ms on LXPLUS. With 10 times larger network (hence with 100 times more parameters), the prediction took ≈ 2 ms.

```

1  //open file and access tree
2  :
3  //declare variables
4  Double_t beamX, beamY, beamdX, beamdY;
5  Byte_t CE1PMn, CE2PMn;
6  //link branch addresses to variables
7  tree->SetBranchAddresses("beamX", &beamX);
8  :
9  //loop over events
10 CModel NNpredictor("/path/to/model.json");
11 for(int i=0; i<list->GetN(); i++){
12     tree->GetEntry(list->GetEntry(i));
13     //get input for the NN model
14     std::vector<FDEEP_FLOAT_TYPE>
15         ↪ input={FDEEP_FLOAT_TYPE(beamX), ...};
16     //get individual PMTs response
17     for(auto pmt: std::bitset<8>(CE2PMn).to_string())
18         input.push_back(FDEEP_FLOAT_TYPE(pmt - '0'));
19     //compute model inference
20     auto modelOutput=NNpredictor.predict(input);
21     :
22 }

```

Listing 5.12: Example code of using a trained model in a ROOT script.

Frugally-deep library together with its dependencies and `CModel` class were put into a publicly accessible directory together with a model trained using kaon proxy¹⁶. Working examples of usage in a standalone application, inside ROOT and finally in PHAST are included. In addition, one can always train and export a different model once e.g. a better kaon proxy is available using the `cedarSeparation` Python package, which is also publicly available.

5.7 Future development

In this section, some ideas for an additional development behind the scope of this thesis are presented, particularly concerning the AMBER experiment, where a proper kaon sample could potentially be obtained.

5.7.1 Improving kaon proxy

One of the problems the kaon proxy could suffer from is related to the undetected second track. While in reality particle that is undetected is the most often a pion (i.e. pion + pion or kaon + pion), the pressure scan gives a proxy of two kaons.

As chapter 5.5.5 explained, removing the angle smearing by using silicon telescopes would lead to the most significant improvement. It would also make possible to obtain a proper kaon sample.

The idea is to search for kaons that decay into a neutrino and a muon in between the two silicon telescope stations (each including two telescopes). The first station will therefore detect the parameters of a kaon at CEDAR, while the second should be able to detect a change in particle inclination, i.e a hint that the decay took place.¹⁷ In later part of the spectrometer this 'new' particle should be identified as a muon with energy below 100 GeV.¹⁸

Another improvement could be achieved by lowering the time for which a given pad gives signal (currently it lasts for 10 ns with the time measurement precision of 0.5 ns). This would allow for better preprocessing in order to discard hits caused by an undetected track.

¹⁶Note that working points are calculated by estimating efficiency as explained in chapter 5.5.1.

¹⁷If the decay takes place before CEDAR, the muon crosses it at very large angle. If it happens after CEDAR, but before the first beam telescope, only the muon parameters are known, but not parameters of the kaon at CEDAR.

¹⁸Pions can decay into muons as well, but the minimum energy of muon from such decay is about 107 GeV for the 190 GeV pion while for the kaon it can be between 8-190 GeV. Because pions produced by the original beam can further decay into muons with low energy, kaons could be identified by selecting e.g. 50-100 GeV muons.

5.7.2 Application for AMBER

The separation of pions and kaons is not an easy task because both of these particles have relatively similar emission angle for Cherenkov radiation. This is due to the fact that kaon is only ≈ 3.5 times heavier than pion. Another thing negatively affecting the separation is the small kaon representation in the beam of only ≈ 0.025 .

However, as mentioned in chapter 1.1.1, COMPASS is finishing its data taking phase and being replaced by the AMBER experiment, which will use a similar detector setup. Besides the current beam, a different beam consisting of 75 % protons and 25 % of pions with positive charge will be used for the AMBER data taking as well. As proton has ≈ 7 times larger mass than charged pion and the ratio of the particle types in the beam is 3:1, the separation based on CEDAR detectors will be much easier in that case.

For these reasons, this newly developed procedure is expected to perform substantially better when used for the AMBER data analysis with this beam.

5.7.3 Using the prediction as probability

As mentioned in chapter 3.2.2, the output of the network can be interpreted as a probability when using certain cost functions such as binary crossentropy. Instead of setting a threshold and selecting events that surpass it, all events can be used with the predicted probability. In this way, the events are weighted in accordance to the probability of being induced by kaons. From statistical point of view, this approach gives more relevant results and stronger statistical tools can be applied.

Conclusion

The focus of this thesis was on finding and analyzing a new method for CEDAR data analysis for the 2018 data taking and the upcoming AMBER experiment. Three different methods using artificial neural networks together with tools for its analysis were implemented and tested. Three types of networks were compared and a genetic algorithm for the meta parameters optimization was created. A procedure for exporting a model trained in Python and importing it into a C++ program using ROOT or PHAST with no Python dependence was developed.

Due to the problems with measured data mentioned throughout the thesis, more than 30 different datasets were examined, both from measured data and Monte Carlo simulations. The most significant problems aggravating particles separation were identified and some possible solutions suggested.

The findings presented in this thesis lies foundations and outlines direction of future advancements. The thoroughly documented code base can be utilized and expanded particularly for the AMBER experiment either by the author or another member of the collaboration.

In conclusion, all of the goals of this thesis were fulfilled.

Bibliography

- [1] ABBON, P. The COMPASS experiment at CERN. In: *Nuclear Instruments and Methods in Physics Research A577*. 2007, s. 455-518.
- [2] AGGARWAL, C. C. *Neural Networks and Deep Learning: A Textbook*. Springer, 2019. ISBN 978-331-9944-630.
- [3] BETZ, Vaughn. FPGA Architecture for the Challenge. In: *The Computer Engineering Research Group* [online]. University of Toronto [cit. 2020-06-25]. Available at: https://www.eecg.utoronto.ca/~vaughn/challenge/fpga_arch.html
- [4] BODLÁK, Martin, Vladimír FROLOV, Vladimír JARÝ, et al. FPGA based data acquisition system for COMPASS experiment. *Journal of Physics: Conference Series*. 2013, (513). Available at: doi:10.1088/1742-6596/513/1/012029
- [5] BOVET, Claude, René MALEYRAN, L. PIEMONTESE, Alfredo PLACCI a Massimo PLACIDI. *The CEDAR counters for particle identification in the SPS secondary beams: a description and an operation manual*. Geneve, Switzerland, 1982, 43 s. CERN Yellow Reports: Monographs. Also available at: <https://cds.cern.ch/record/142935/files/CERN-82-13.pdf>
- [6] BROWNIEE, Jason. What is the Difference Between Test and Validation Datasets?. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 14.7.2017 [cit. 2022-03-01]. Available at: <https://machinelearningmastery.com/difference-test-validation-datasets/>
- [7] BROWNIEE, Jason. A Gentle Introduction to k-fold Cross-Validation. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 23.5.2018 [cit. 2022-03-01]. Available at: <https://machinelearningmastery.com/k-fold-cross-validation/#:~:text=Cross%2Dvalidation%20is%20a%20resampling,is%20to%20be%20split%20into.>
- [8] BRUN, René, Fons RADEMAKERS. *ROOT - An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86.
- [9] BUSHAEV, Vitaly. Stochastic Gradient Descent with momentum. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online].

- 2022, 4.12.2017 [cit. 2022-02-16]. Available at: <https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d>
- [10] CHOLLET, F. Deep Learning with Python. New York: Manning Publications, 2017. ISBN 978-161-7294-433.
- [11] COVER, Thomas. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers*. 1965, **EC-14**(3), 326-334. Available at: doi:10.1109/PGEC.1965.264137
- [12] DANIELS, Jed. Developing Python applications in Qt Creator. *Stack Overflow* [online]. [cit. 2021-8-31]. Available at: <https://stackoverflow.com/questions/24100602/developing-python-applications-in-qt-creator>
- [13] EDGAR, Thomas a David MANZ. Chapter 4 - Exploratory Study. *Research Methods for Cyber Security*. Syngress, 2017, s. 95-130. ISBN 9780128053492.
- [14] GANDHI, Rohith. Support Vector Machine - Introduction to Machine Learning Algorithms: SVM model from scratch. *Towards Data Science* [online]. 7.6.2018 [cit. 2021-8-5]. Available at: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [15] GERASIMOV, Sergei. *PHysics Analysis Software Tools* [online]. 2021 [cit. 2021-7-17]. Available at: <http://ges.web.cern.ch/ges/phast/>
- [16] GERASIMOV, Sergei. *PHysics Analyses Software Tools (PHAST): Installation and use* [online]. In: . 11.5.2021 [cit. 2021-7-17]. Available at: https://indico.cern.ch/event/1036716/attachments/2243219/3805053/Amber_Tutorial_PHAST.pdf
- [17] GÓMEZ, Raúl. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. In: *Raúl Gómez blog* [online]. 2022, 23.5.2018 [cit. 2022-04-28]. Available at: https://gombru.github.io/2018/05/23/cross_entropy_loss/
- [18] GUPTA, Prashant. Decision Trees in Machine Learning. *Towards Data Science* [online]. 17.5.2017 [cit. 2021-8-4]. Available at: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
- [19] HERMANN, Tobias. *Frugally-deep* [online]. In: . 2022 [cit. 2022-04-15]. Available at: <https://github.com/Dobiasd/frugally-deep>
- [20] KARIM, Raimi. 10 Stochastic Gradient Descent Optimisation Algorithms + Cheatsheet. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 22.11.2018 [cit. 2022-02-16]. Available at: <https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9#:~:text=The%20difference%20between%20Adelta%20and,moving%20average%20of%20squared%20deltas.>

- [21] KUMAR, Satyam. Overview of various Optimizers in Neural Networks. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 9.1.2020 [cit. 2022-02-16]. Available at: <https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5>
- [22] LOPES, Ana. Meet AMBER: The next-generation successor of the COMPASS experiment will measure fundamental properties of the proton and its relatives. In: *CERN: Accelerating science* [online]. Geneva, Switzerland, 8.3.2021 [cit. 2022-02-14]. Available at: <https://home.cern/news/news/physics/meet-amber>
- [23] MALLAWAARACHCHI, Vijini. Introduction to Genetic Algorithms. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022 [cit. 2022-04-01]. Available at: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3#:~:text=A%20genetic%20algorithm%20is%20a,offspring%20of%20the%20next%20generation.>
- [24] MOHRI, Mehryar, Afshin ROSTAMIZADEH a Ameet TALWALKAR. *Foundations of machine learning*. Second edition. Cambridge, Massachusetts: The MIT Press, 2018. ISBN 978-026-2039-406.
- [25] NIELSEN, Michael. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [cit. 2021-8-10]. Available at: <http://neuralnetworksanddeeplearning.com/>
- [26] NOVÝ, Josef. Event Processing in the COMPASS DAQ. In: AMBROŽ, Petr a Zuzana MASÁKOVÁ. *Doktorandské dny 2015: sborník workshopu doktorandů FJFI oboru Matematické inženýrství*. 1. Praha: Nakladatelství ČVUT, 2015, s. 123-130. Also available at: <http://kmwww.fjfi.cvut.cz/ddny/historie/15-sbornik.pdf>
- [27] OLAH, Christopher. Understanding LSTM Networks. *Colah's blog* [online]. 27.8.2015 [cit. 2021-8-7]. Available at: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [28] ORR, Mark. *Introduction to Radial Basis Function Networks*. Edinburgh, Scotland: Centre for Cognitive Science, University of Edinburgh, 1996. Also available at: <https://faculty.cc.gatech.edu/~isbell/tutorials/rbf-intro.pdf>
- [29] RAMACHANDRAN, Prajit, Barret ZOPH a Quoc V. LE. *Searching for Activation Functions* [online]. 16.10.2017 [cit. 2022-02-15]. Available at: doi:1710.05941
- [30] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. *Towards Data Science* [online]. 2022, 15.12.2018 [cit. 2022-02-17]. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

- [31] SAHAY, Milind. Neural Networks and the Universal Approximation Theorem: And the boom of deep neural networks in recent times. In: *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 6.6.2020 [cit. 2022-04-25]. Available at: <https://towardsdatascience.com/neural-networks-and-the-universal-approximation-theorem-8a389a33d30a>
- [32] SANDERSON, Grant. *Neural networks* [online]. 5.10.2017 [cit. 2021-8-10]. Available at: https://www.youtube.com/playlist?list=PLZHQB0b0WTQDNU6R1_67000Dx_ZCJB-3pi
- [33] SANGHVI, Rajit. A Complete Guide to Adam and RMSprop Optimizer. *Towards Data Science: A Medium publication sharing concepts, ideas and codes* [online]. 2022, 20.2.2021 [cit. 2022-02-16]. Available at: <https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be>
- [34] SAOTOME, Taiji. *How to Install and Setup TensorFlow for NVIDIA GPUs on CentOS 7 Linux* [online]. 3.4.2019 [cit. 2021-7-28]. Available at: https://community.spiceworks.com/how_to/161323-how-to-install-and-setup-tensorflow-for-nvidia-gpus-on-centos-7-linux
- [35] SAXENA, Shipra. Binary Cross Entropy/Log Loss for Binary Classification. In: *Analytucs Vidhya: Analytics Community | Analytics Discussion | Big Data Discussion* [online]. 2022, 3.3.2021 [cit. 2022-04-28]. Available at: <https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/#:~:text=What%20is%20Binary%20Cross%20Entropy,from%20the%20actual%20value.>
- [36] STOLARSKI, Marcin. CEDAR analysis update. Talk at: *COMPASS Analysis Meeting* [online]. 17.12.2020 [cit. 2021-8-24]. Available at: https://wwwcompass.cern.ch/compass/software/analysis/transparencies/2020/am_201217/talks/Stolarski_201217.pdf
- [37] SWAMINATHAN, Saishruthi. Logistic Regression: Detailed Overview. *Towards Data Science* [online]. 15.3.2018 [cit. 2021-8-31]. Available at: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>
- [38] ŠERÝCH, Jakub. CERN: jak to celé funguje. *Jakub Šerých: Osobní stránky* [online]. Praha [cit. 2020-5-2]. Available at: <https://jakub.serych.cz/book/export/html/40>
- [39] VIDNEROVÁ, Petra. RBF-Keras: an RBF Layer for Keras Library. 2019. Available at https://github.com/PetraVidnerova/rbf_keras
- [40] VIRIUS, Miroslav. CERN, experiment COMPASS a ČVUT. *TECNICALL*. 2013, (1), 18.

- [41] VOLDŘICH, František. *Beam particle identification using CEDARs at the COMPASS experiment in CERN*. Prague, 2021. Research project. Czech Technical University. Supervisor: Martin Zemko.
- [42] Wallner, S., Bicker K., Friedrich J.M., Grube B., Huber S. a Krämer M. *CEDAR PID using the Likelihood Approach for the Hadron-Beam: 2008 Data Set*, COMPASS Note 2017-1, [online]. In: . 2017 [cit. 2021-7-17]. Available at: https://wwwcompass.cern.ch/compass/notes_public/2017-1.pdf
- [43] WEISROCK, Tobias. *Likelihood Methods for Beam Particle Identification at the COMPASS Experiment*. Prague, 2012. Available at: https://wwwcompass.cern.ch/compass/publications/conf_proc/proc/t2012/weisrock_praha2012_proc.pdf
- [44] YANG, Xin-She. 2 - Mathematical foundations. *Introduction to Algorithms for Data Mining and Machine Learning*. Academic Press, 2019, s. 19-43. ISBN 9780128172162. Also available at: <https://doi.org/10.1016/B978-0-12-817216-2.00009-0>
- [45] ZHANG, L. a P.N. SUGANTHAN. *A Comprehensive Evaluation of Random Vector Functional Link Networks* [online]. 2016 [cit. 2021-12-02]. Available at: [doi:https://doi.org/10.1016/j.ins.2015.09.025](https://doi.org/10.1016/j.ins.2015.09.025)
- [46] *AMBER: A new QCD facility at the M2 beam line of the CERN SPS* [online]. CERN, Geneve, 2022 [cit. 2022-02-14]. Available at: <https://amber.web.cern.ch/>
- [47] *Batch Docs: CERN Batch Service User Guide* [online]. 2022 [cit. 2022-02-27]. Available at: <https://batchdocs.web.cern.ch/index.html>
- [48] Batch Service Concepts: Basic Batch Pattern. *Batch Docs: CERN Batch Service User Guide* [online]. 2022 [cit. 2022-02-27]. Available at: <https://batchdocs.web.cern.ch/concepts/index.html>
- [49] *COMPASS: COMmon Muon Proton Apparatus for Structure and Spectroscopy* [online]. Geneve, Switzerland, 2021 [cit. 2022-02-14]. Available at: <https://wwwcompass.cern.ch/>
- [50] CentOS - Linux @ CERN. *Linux @ CERN* [online]. [cit. 2021-7-20]. Available at: <https://linux.web.cern.ch/centos/>
- [51] *CERN: Accelerating science* [online]. Geneve, Switzerland, 2022 [cit. 2022-02-14]. Available at: <https://home.cern/>
- [52] Extending and Embedding the Python Interpreter. *Python 3.9.6 documentation* [online]. [cit. 2021-7-28]. Available at: <https://docs.python.org/3/extending/embedding.html>
- [53] How to install nvidia driver with secure boot enabled? *Ask Ubuntu* [online]. [cit. 2021-8-2]. Available at: <https://askubuntu.com/questions/1023036/how-to-install-nvidia-driver-with-secure-boot-enabled>

- [54] *HTCondor Software Suite* [online]. 2022 [cit. 2022-02-27]. Available at: <https://research.cs.wisc.edu/htcondor/>
- [55] *Keras* [online]. [cit. 2021-7-28]. Available at: <https://keras.io>
- [56] Optimizers. *Keras: Simple. Flexible. Powerful.* [online]. 2022 [cit. 2022-02-16]. Available at: <https://keras.io/api/optimizers/>
- [57] *PyTorch: Official website* [online]. [cit. 2021-8-31]. Available at: <https://pytorch.org/>
- [58] *ROOT: analyzing petabytes of data, scientifically.* [online]. CERN, Geneve, 2021 [cit. 2021-7-16]. Available at: <https://root.cern/>
- [59] *Scikit-learn: machine learning in Python* [online]. [cit. 2021-8-31]. Available at: <https://scikit-learn.org/stable/>
- [60] Support-vector machine. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-8-9]. Available at: https://en.wikipedia.org/wiki/Support-vector_machine
- [61] *TensorFlow: An end-to-end open source machine learning platform* [online]. [cit. 2021-7-28]. Available at: <https://www.tensorflow.org/>
- [62] *The LXPLUS Service* [online]. 2022 [cit. 2022-02-26]. Available at: <https://lxplusdoc.web.cern.ch/>

Appendix A

Additional plots

In this appendix, some extra plots are included.

A.1 Using approximations of beam angles

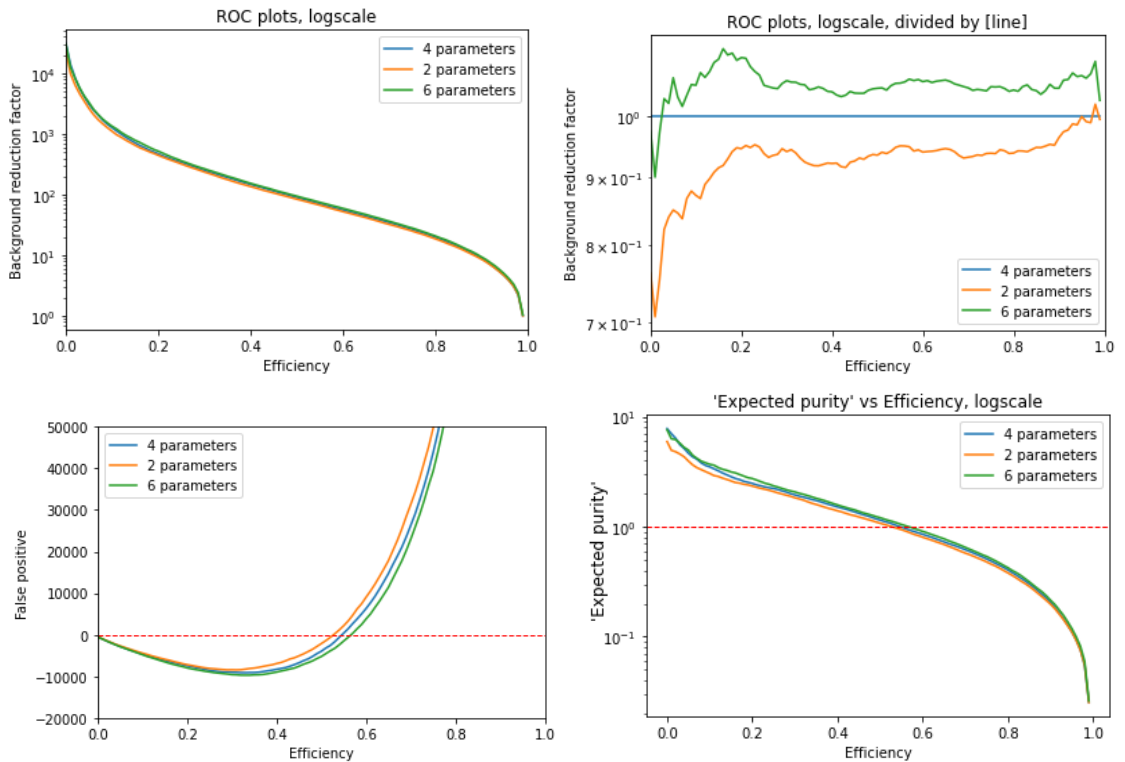


Figure A.1: Curves of networks trained using the 4 original variables, using their aggregation into 2 variables and using all six of these parameters.

A.2 Confusion matrices

This section contains the confusion matrices together with some derived statistics for different network types and fixed efficiency 0.5.

	predicted 0	predicted 1
actual 0	true negative: 483846	false positive: 3978
actual 1	false negative: 6088	true positive: 6088

statistic	value	

sensitivity:	0.5	
background reduction	122.63	
accuracy:	0.979868	
error rate:	0.020132	
specificity:	0.991845	
false positive rate	0.00815458	
precision	0.604808	
Matthews correlation coefficient	0.539778	
F score: F_0.5	0.580473	
F score: F_1	0.547433	
F score: F_2	0.517951	

(a) Plain network

	predicted 0	predicted 1
actual 0	true negative: 477976	false positive: 9848
actual 1	false negative: 6088	true positive: 6088

statistic	value	

sensitivity:	0.5	
background reduction	49.5353	
accuracy:	0.968128	
error rate:	0.031872	
specificity:	0.979812	
false positive rate	0.0201876	
precision	0.382028	
Matthews correlation coefficient	0.421031	
F score: F_0.5	0.400948	
F score: F_1	0.433125	
F score: F_2	0.470916	

(b) RBF network

	predicted 0	predicted 1
actual 0	true negative: 471186	false positive: 16638
actual 1	false negative: 6088	true positive: 6088
statistic	value	
sensitivity:	0.5	
background reduction	29.3199	
accuracy:	0.954548	
error rate:	0.045452	
specificity:	0.965893	
false positive rate	0.0341066	
precision	0.267887	
Matthews correlation coefficient	0.344767	
F score: F_0.5	0.295305	
F score: F_1	0.348863	
F score: F_2	0.426151	

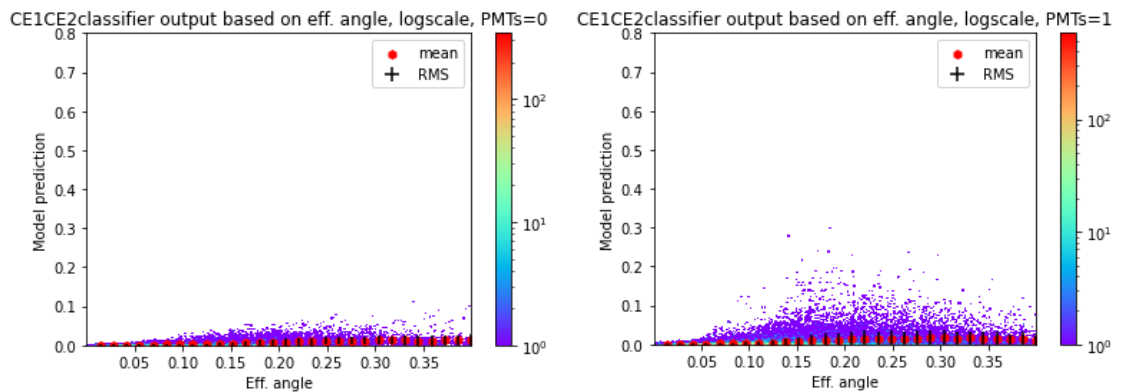
(c) RVFL network

Figure A.2: Output of `PredictionAnalyzer.printConfusionStats` for different network types.

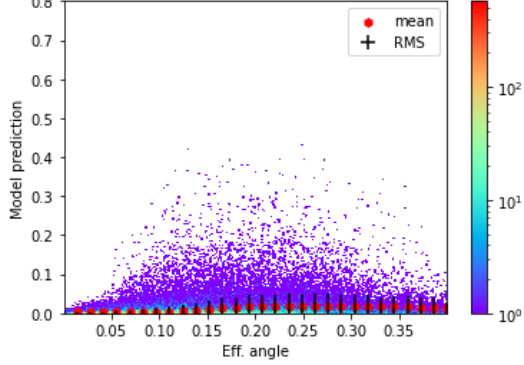
A.3 Model output vs. angle

This section contains plots for all multiplicities as shown in fig 5.15, i.e. the model output versus angle at CEDAR and different multiplicities. All of the used datasets are taken from measured data and are therefore without added kaon proxy. Similarly, no kaons are duplicated in MC files. This is output of the method `PredictionAnalyzer.plotAngleHistBins`.

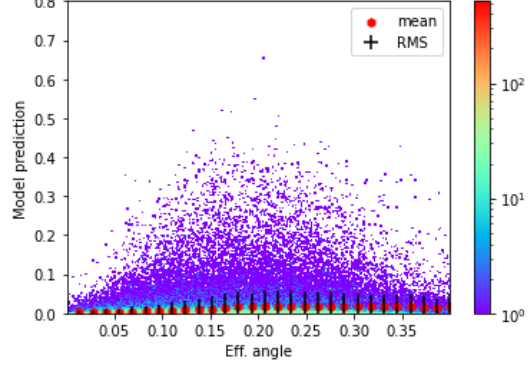
A.3.1 Method 1 - Measured data



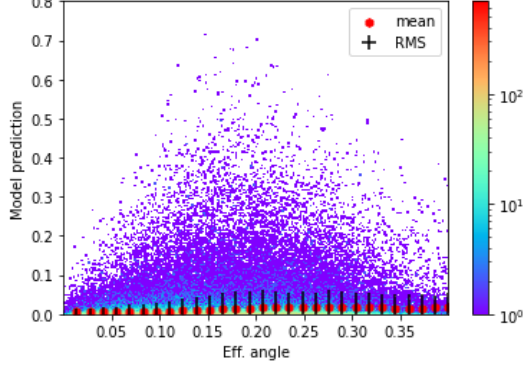
CE1CE2classifier output based on eff. angle, logscale, PMTs=2



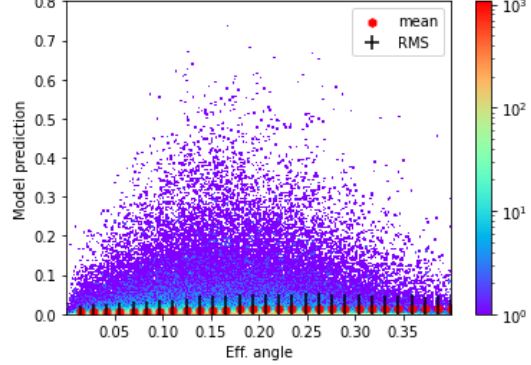
CE1CE2classifier output based on eff. angle, logscale, PMTs=3



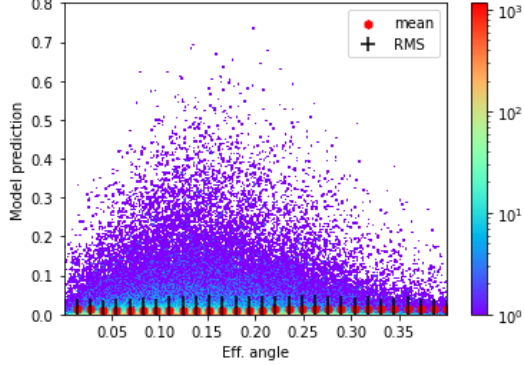
CE1CE2classifier output based on eff. angle, logscale, PMTs=4



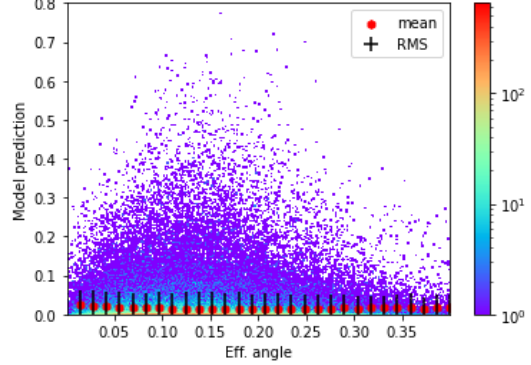
CE1CE2classifier output based on eff. angle, logscale, PMTs=5



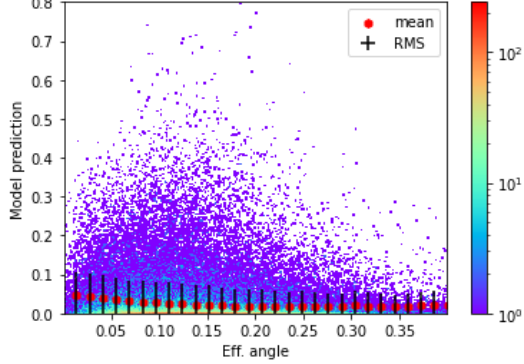
CE1CE2classifier output based on eff. angle, logscale, PMTs=6



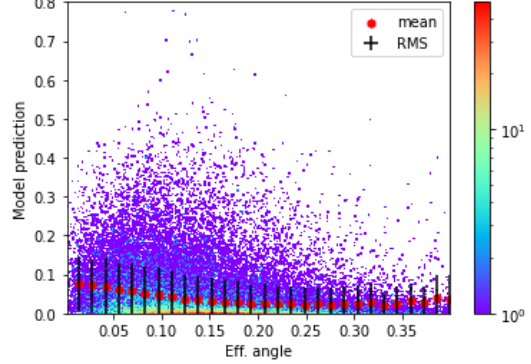
CE1CE2classifier output based on eff. angle, logscale, PMTs=7



CE1CE2classifier output based on eff. angle, logscale, PMTs=8



CE1CE2classifier output based on eff. angle, logscale, PMTs=9



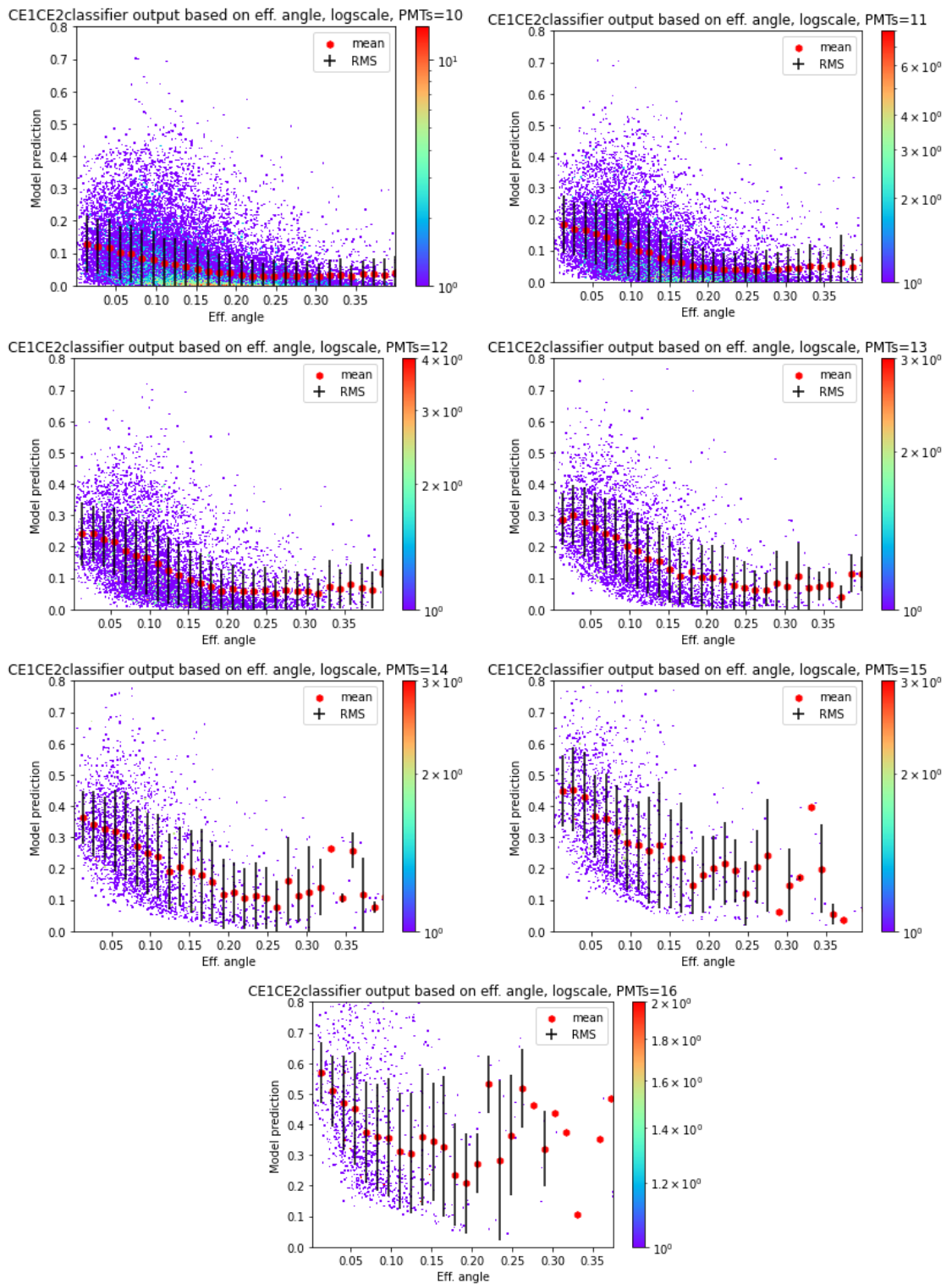
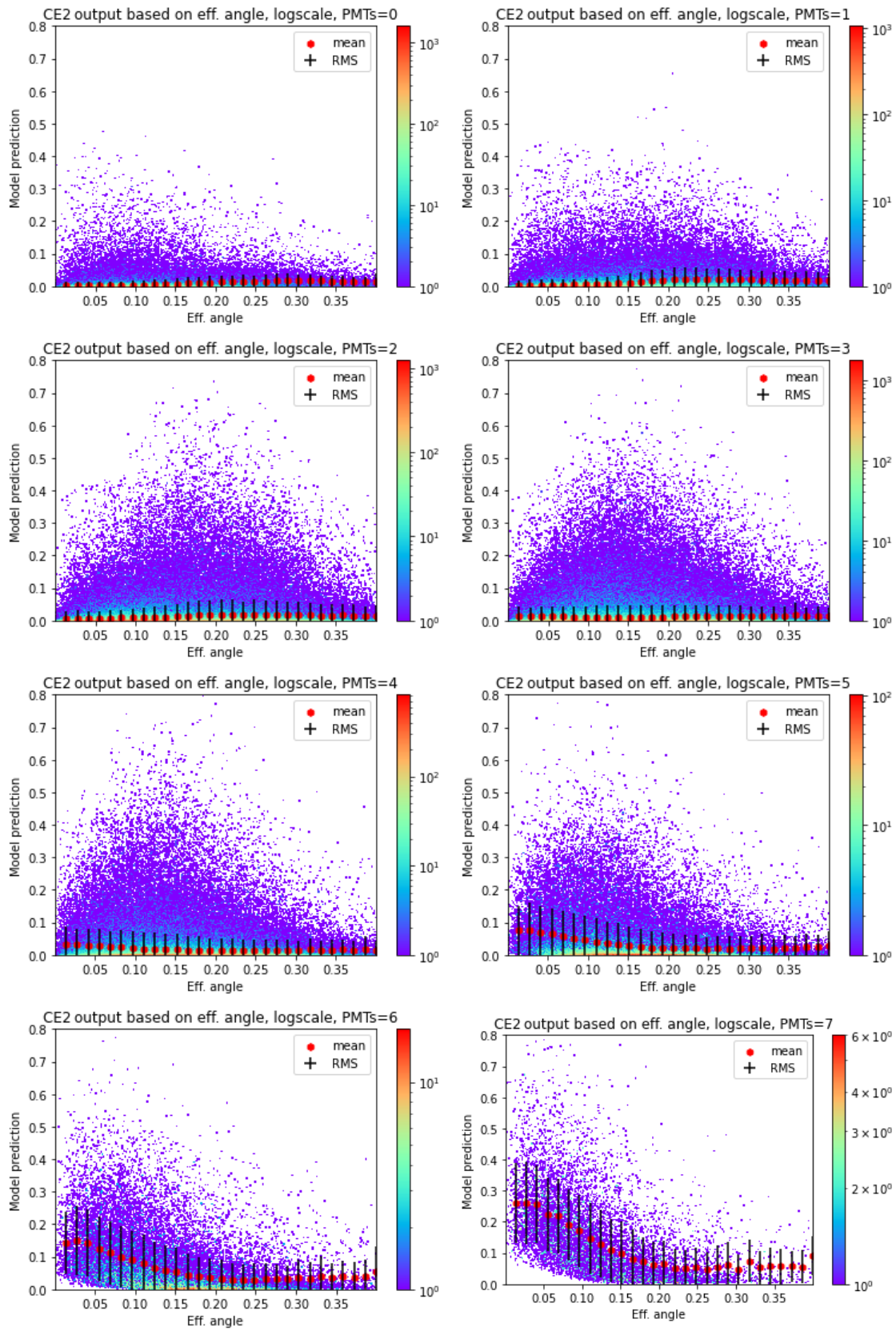


Figure A.3: Network output based on eff. angle for different multiplicities with both CEDARs combined.



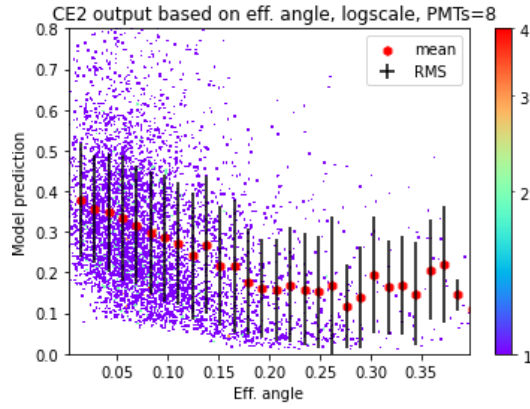
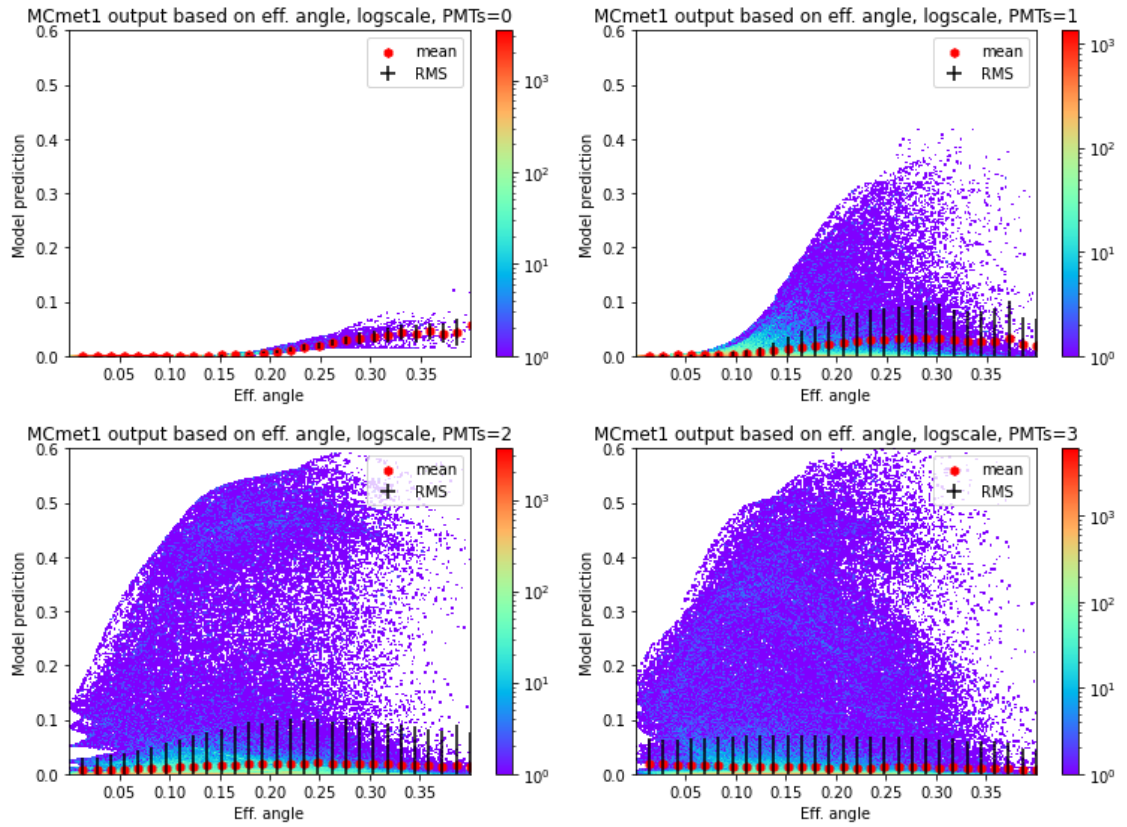


Figure A.4: Network output based on eff. angle for different multiplicities for CEDAR 2 only.

A.3.2 Method 1 - Monte Carlo simulations



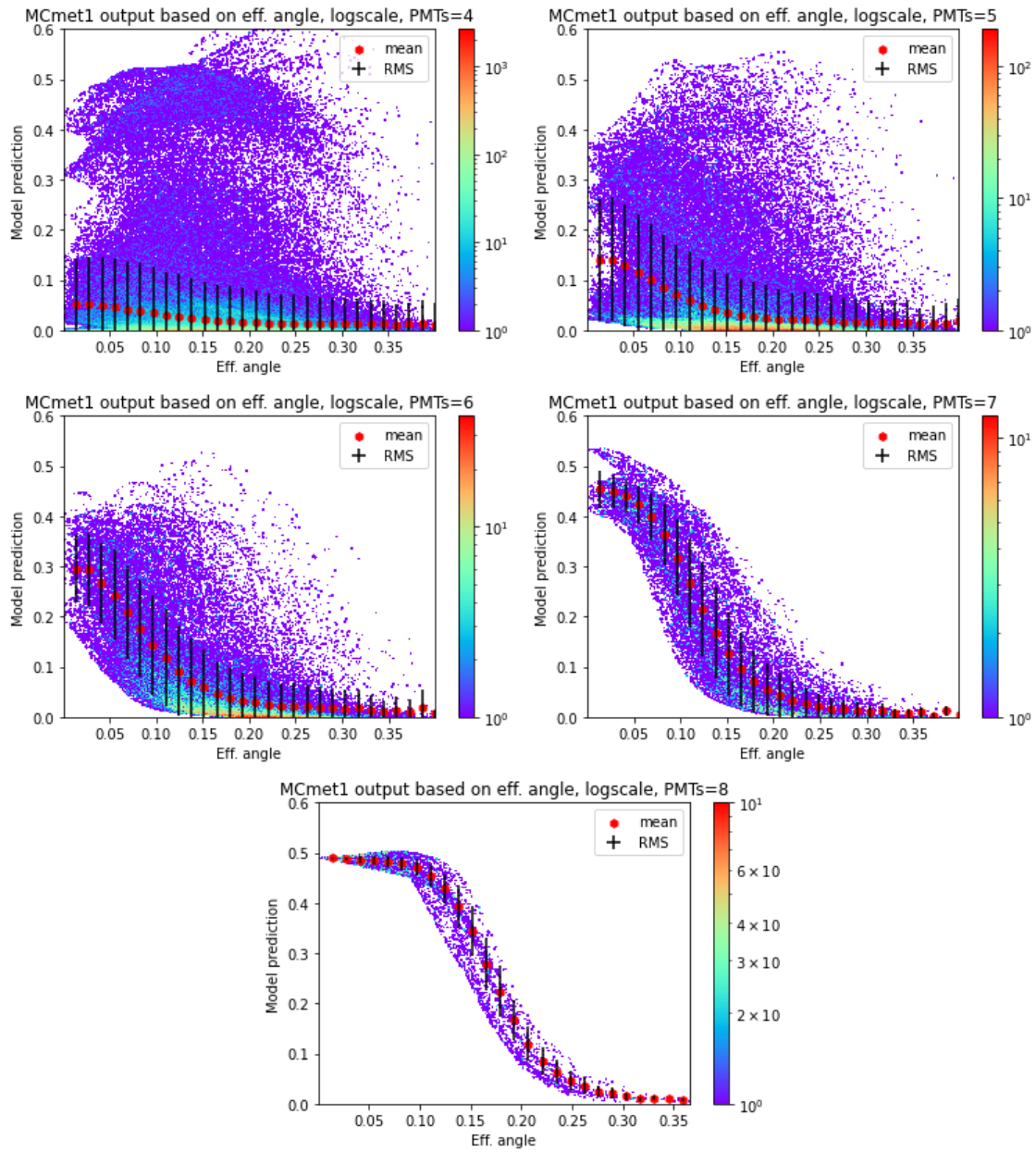
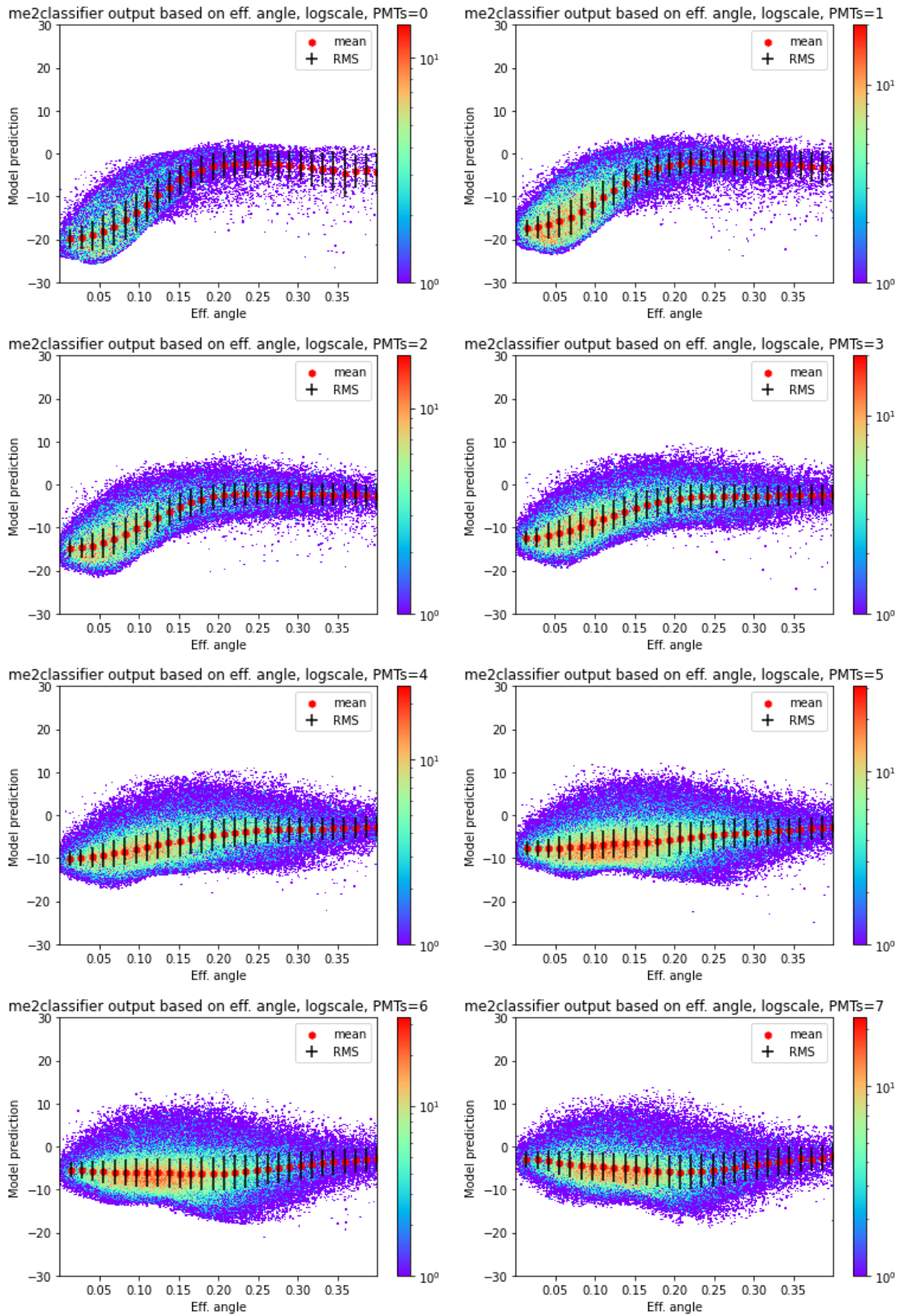
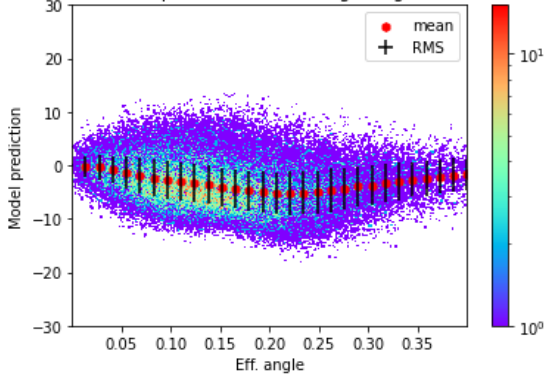


Figure A.5: Network output based on eff. angle for different multiplicities for simulation of one CEDAR only.

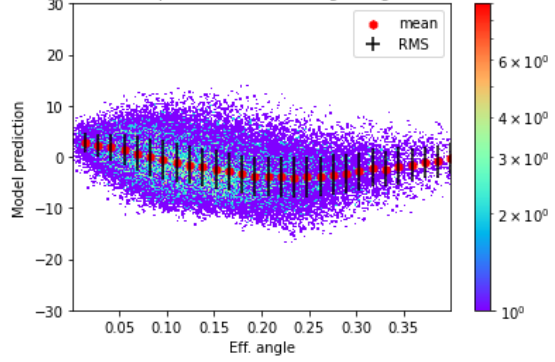
A.3.3 Method 2 - Measured data



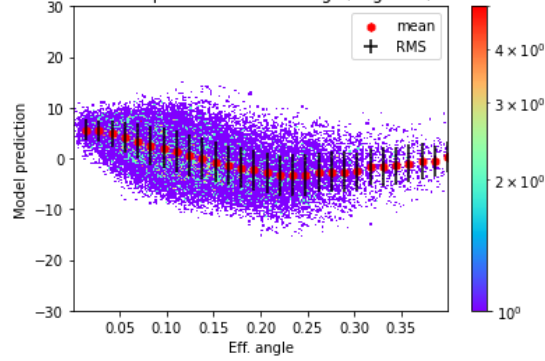
me2classifier output based on eff. angle, logscale, PMTs=8



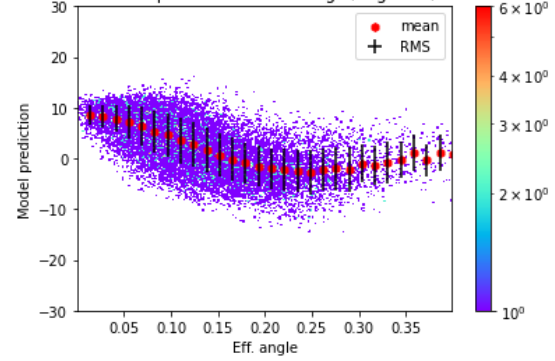
me2classifier output based on eff. angle, logscale, PMTs=9



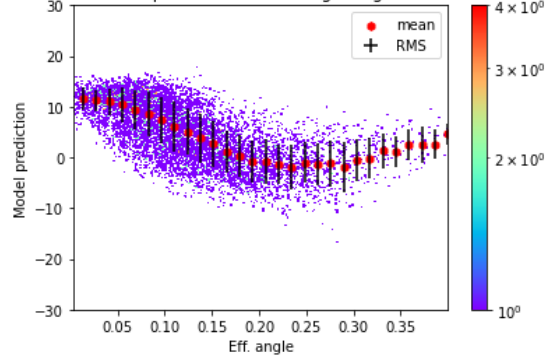
me2classifier output based on eff. angle, logscale, PMTs=10



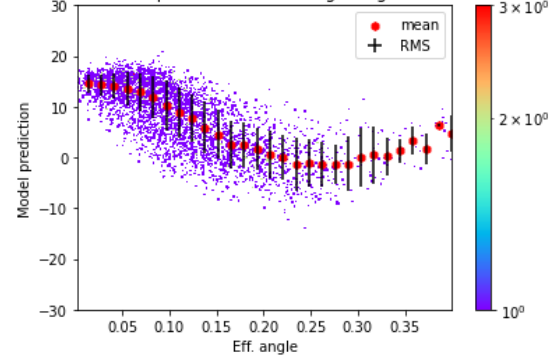
me2classifier output based on eff. angle, logscale, PMTs=11



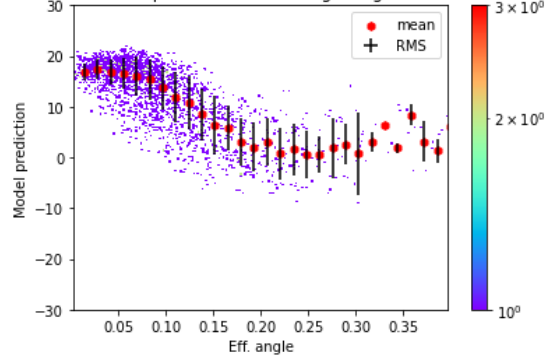
me2classifier output based on eff. angle, logscale, PMTs=12



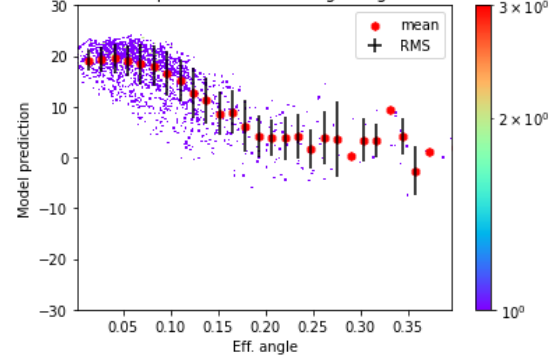
me2classifier output based on eff. angle, logscale, PMTs=13



me2classifier output based on eff. angle, logscale, PMTs=14



me2classifier output based on eff. angle, logscale, PMTs=15



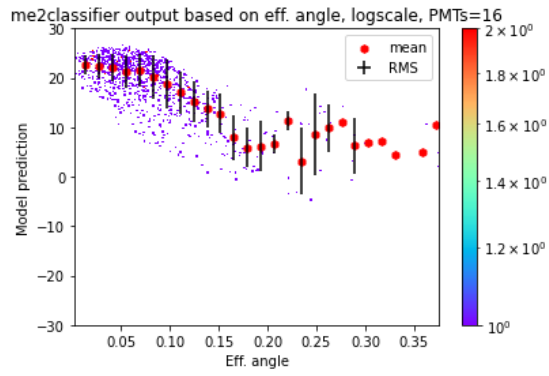
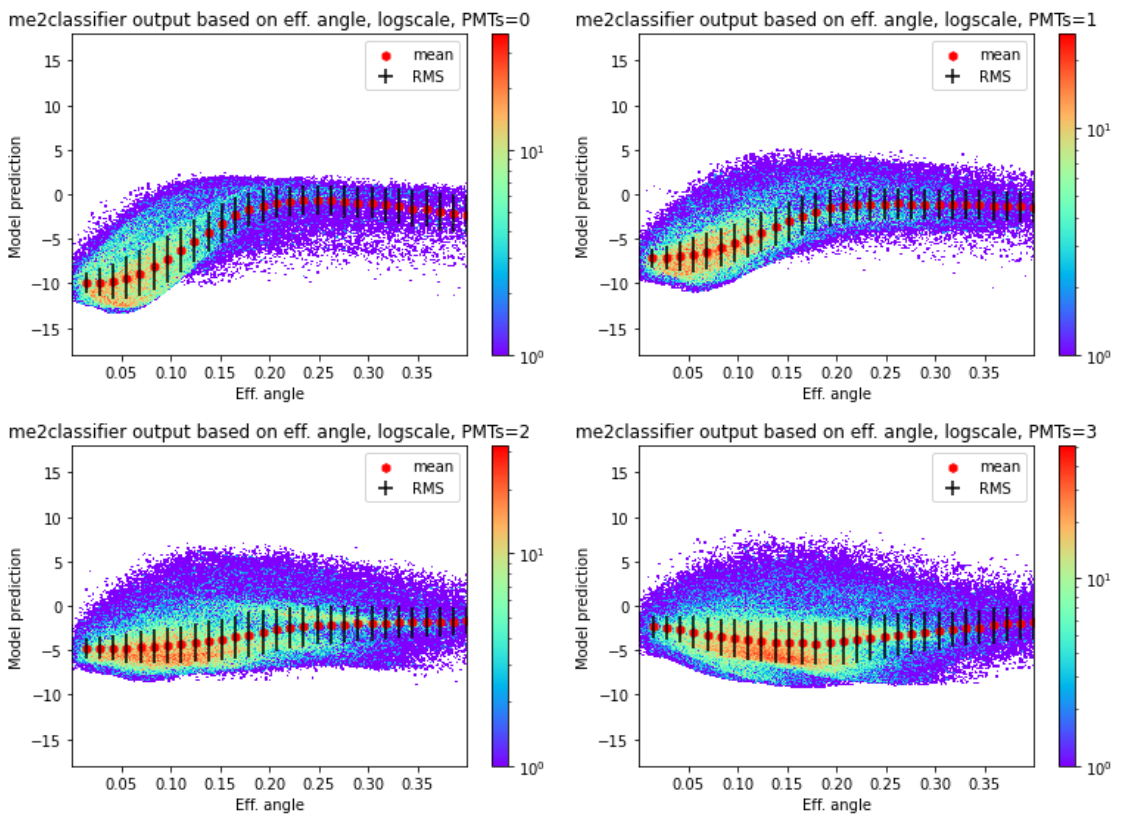


Figure A.6: Network output based on eff. angle for different multiplicities with both CEDARs combined.



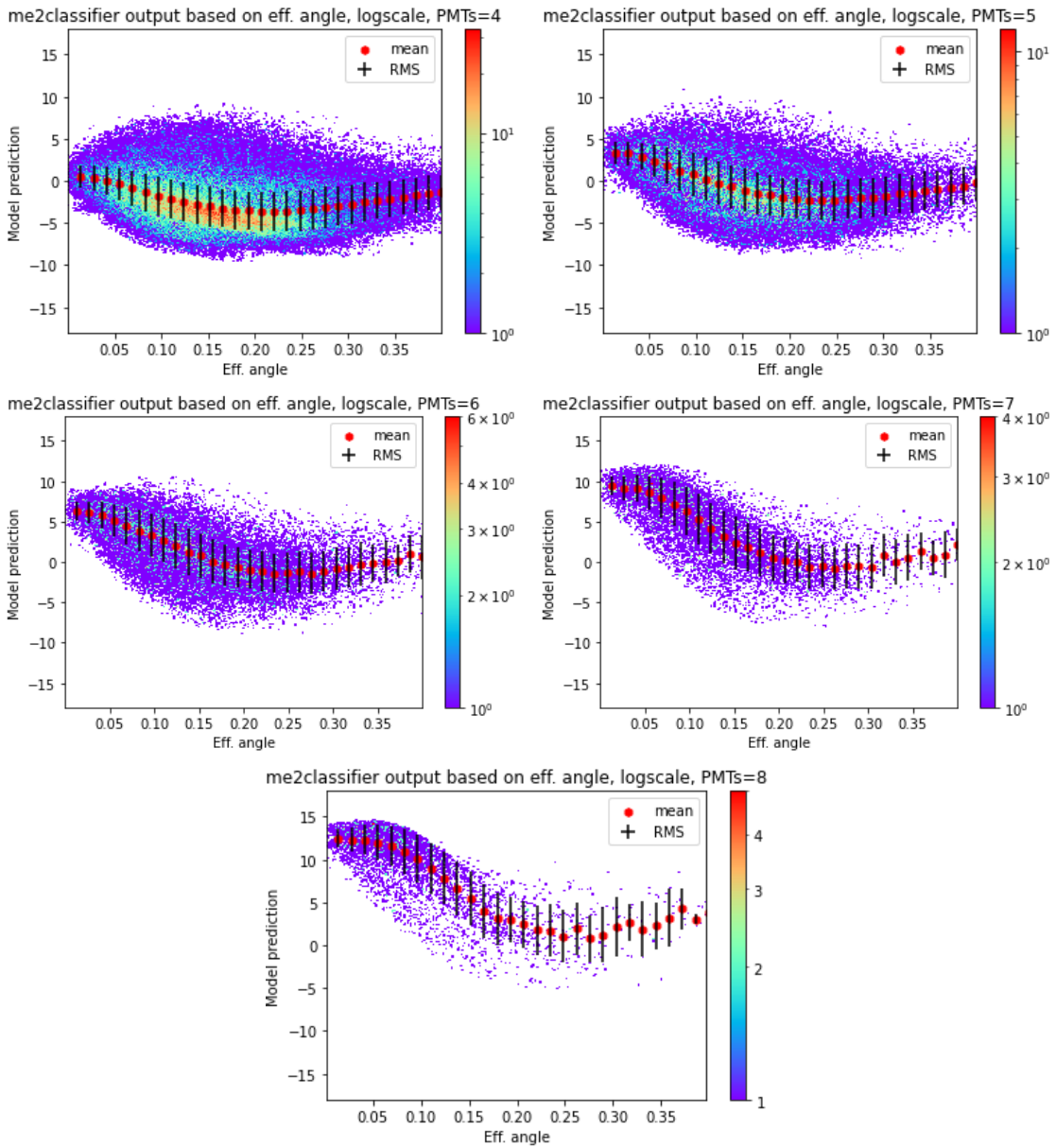
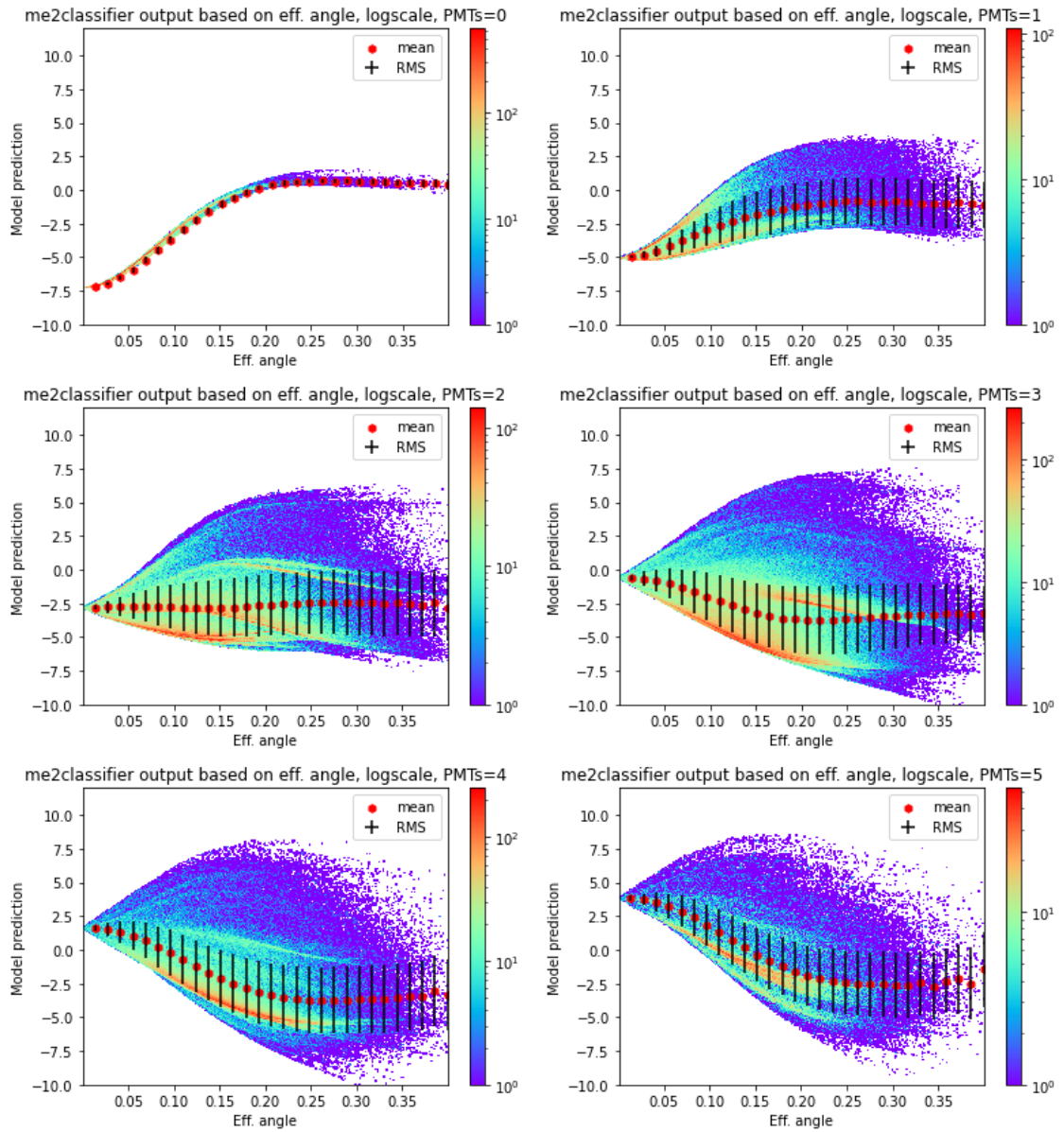


Figure A.7: Network output based on eff. angle for different multiplicities for CEDAR 2 only.

A.3.4 Method 2 - Monte Carlo simulations



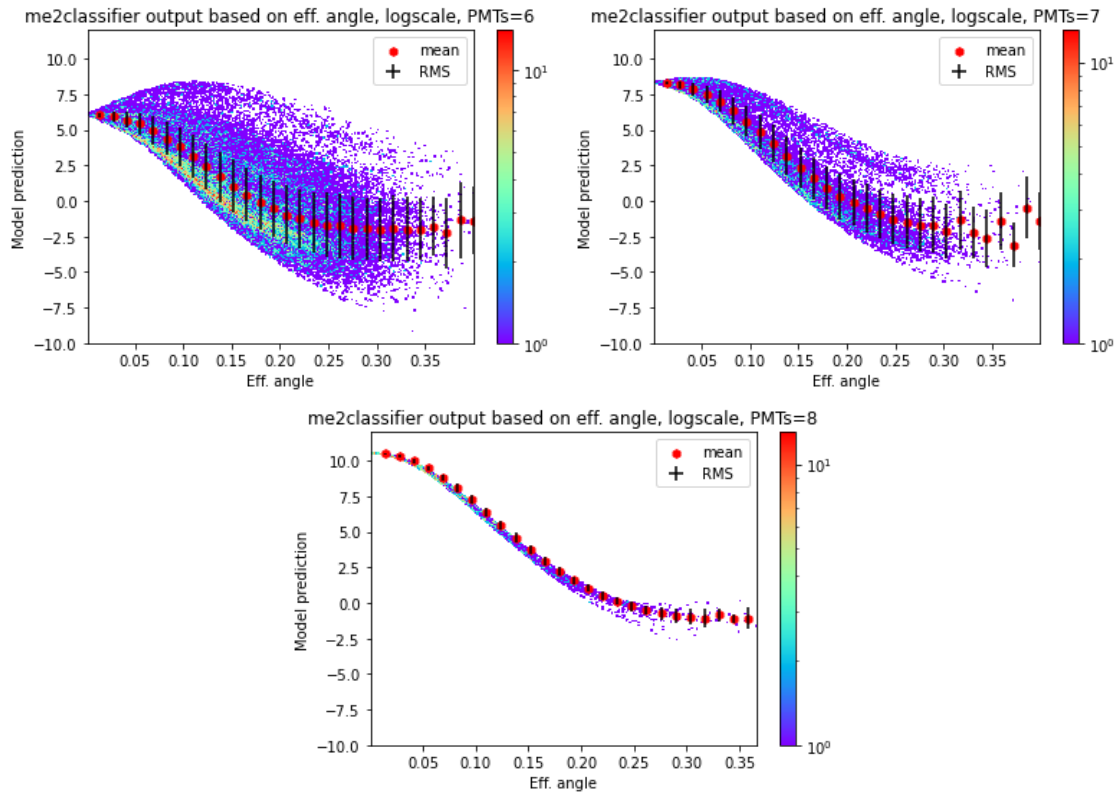
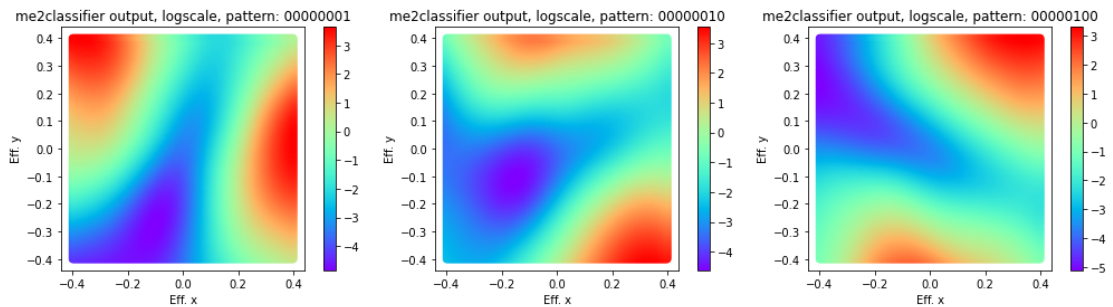
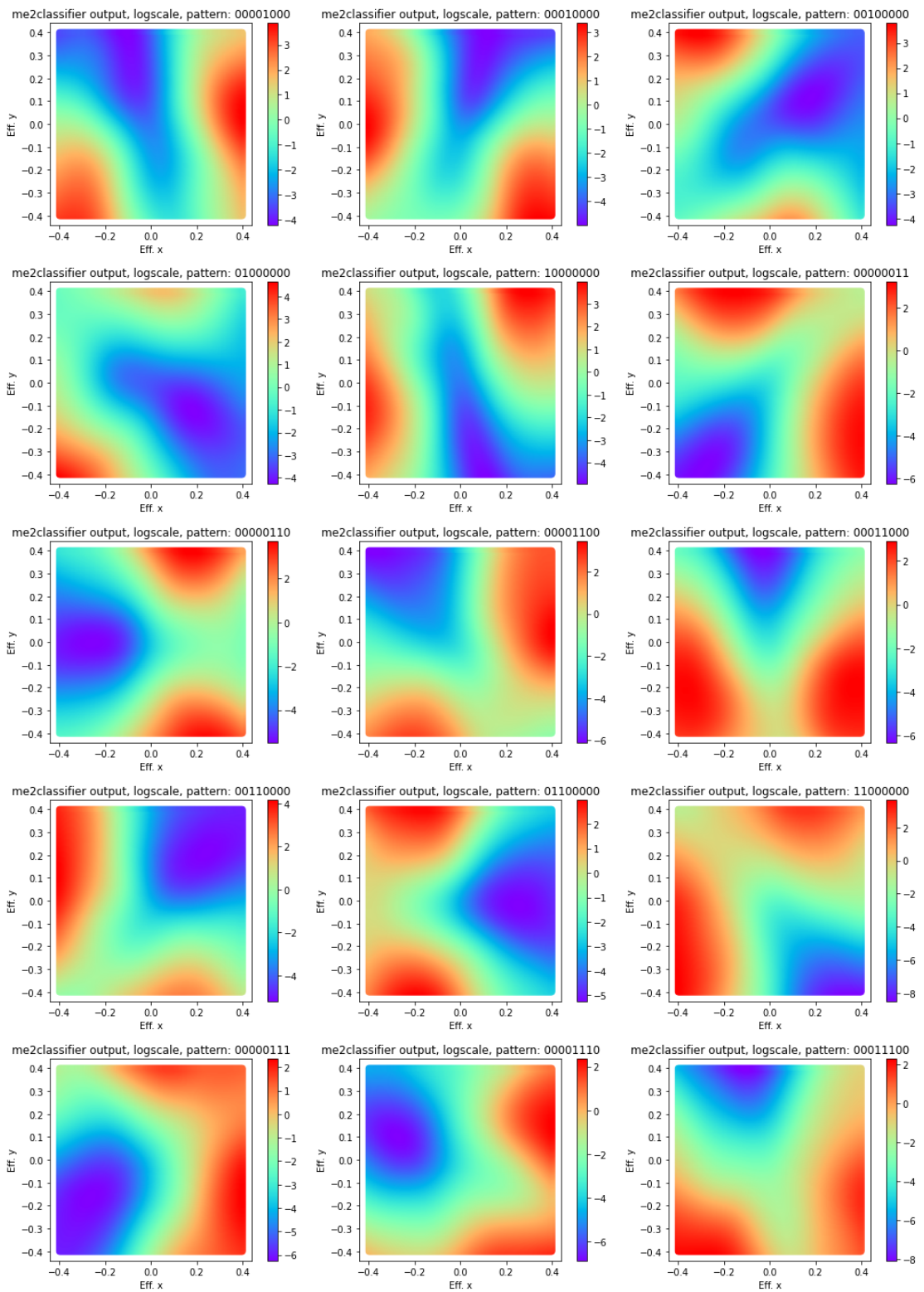


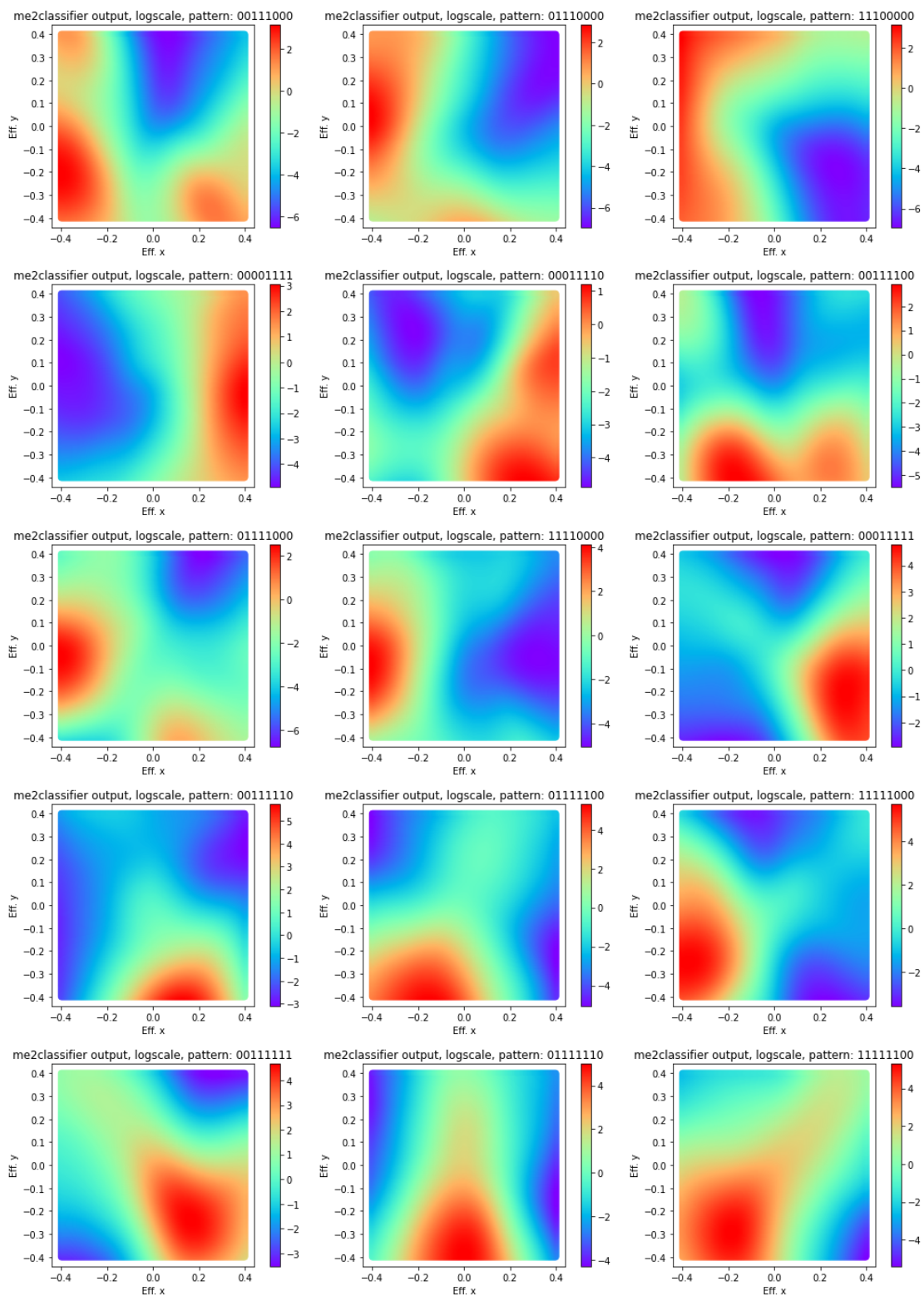
Figure A.8: Network output based on eff. angle for different multiplicities for simulation of one CEDAR only.

A.4 Method 3 angle histograms

This appendix contains colormaps of the output of method 3 (i.e. $\log(\frac{p(K)}{p(\pi)})$) as a colormap in eff. angle x and eff. angle y coordinates for symmetrical responses of PMTs.







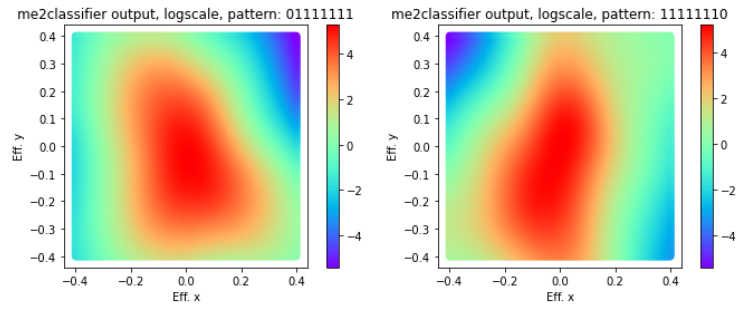


Figure A.9: Predictions as a colormap in eff. angle x and eff. angle y coordinates of symmetric responses, i.e. a group of PMTs responding together for method 3. Output of the method `PredictionAnalyzer.meth3plotsBySymmetry`.

Appendix B

CD contents

In the attached CD, several directories can be found beside the text of this thesis in `.pdf` format. Namely, it includes:

- **cedarSeparation** - full Python package including source codes of `CModel`, `PmtResponseModel`, `Classifier`, `M23Classifier` and `PredictionAnalyzer` classes.
- **inference** - source code of `CModel` C++ class together with frugally-deep and its dependencies inside the `includes` subdirectory. In addition, a model `model.json` trained using kaon proxy is present.
- **inference examples** - contains working example codes of using trained a trained model in a standalone application, inside ROOT and inside PHAST. A text file `README.txt` describes the process of training a network and exporting it for usage in PHAST.
- **models** - includes several trained models for loading in TensorFlow both for measured data and Monte Carlo simulations.
- **plots** - contains plots present in this thesis and some additional ones.
- **additional code** - holds other Python scripts developed as part of this thesis.