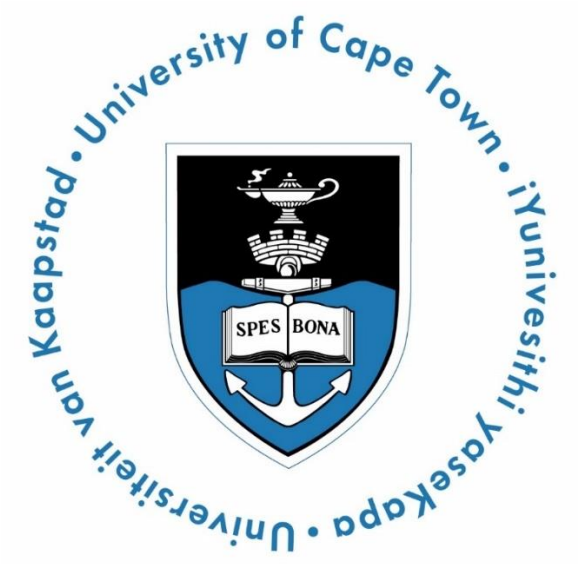
Deep Generative Models for High Energy Physics Detector Simulations & Convolutional Neural Networks for Particle Identification



Christiaan Gerhardus Viljoen

Department of Statistics || Department of Physics

Faculty of Science

University of Cape Town

This dissertation is submitted in partial fulfilment of the Degree of Master of Science

Dedicated to my mother, Elizabeth Suzanna Bloem Viljoen, who has always inspired me to follow my higher passions, despite the myriad difficulties that life makes us face; and to search fearlessly and incessantly for the deeper truths underlying our everyday world.

"A man may imagine things that are false,

But he can only understand things that are true;

For if the things be false,

The apprehension of them is not understanding"

– Sir Isaac Newton

Abstract

This Masters project was focused on the application of deep learning techniques towards specific aspects of particle physics. Its two main aims: *particle identification* and *high energy physics detector simulations* are pertinent to research avenues pursued by physicists working with the ALICE[1] TRD[2] detector, within the LHC[3] at CERN[4].

Aims

More formally, the aims of this project were as follows:

1. For particle identification: various neural networks were trained and assessed, to determine their ability to discriminate between electrons and pions, produced during proton-Lead (pPb) collisions conducted at the LHC in 2016, based on ADC[5] signal data produced as these particles were detected by the ALICE TRD. (Note that this work was done on uncalibrated raw TRD digits).
2. For high energy physics detector simulations: Geant4, a Monte Carlo toolkit used to simulate particle interactions with matter, was assessed in terms of how closely the simulated data it produces resembles true data taken by the TRD during collision events. In addition, as a step towards fast simulation, various deep generative modeling strategies were employed to produce simulated data samples which are likely under the observed (true) TRD data distribution. To this end, the following classes of latent variable models were prototyped: Generative Adversarial Networks, Variational Autoencoders and Adversarial Autoencoders. Data produced during these deep generative simulations were compared to real data in the same manner as that done for Geant4 data, in order to assess the feasibility of incorporating these types of models into future high energy physics event simulation software.

Summary of Results

Particle identification performance was defined by the ability of each neural network to minimize pion efficiency (ε_π, false positive rate), whilst maximizing electron efficiency (ε_e, true positive rate). A lower bound for the critical region (t_{cut}) in the distribution of $P(elec)$ predictions made by each neural network which results in $\varepsilon_e \approx 90\%$ was defined, in order to determine the ε_π for that neural network. The best set of results obtained, per

---

[1] A Large Ion Collider Experiment

[2] Transition Radiation Detector

[3] Large Hadron Collider

[4] European Organization for Nuclear Research

[5] Analog to Digital Converter

momentum bin, was as follows: $\varepsilon_\pi = 1.2\%$ in the $p \leq 2$ GeV/c range; $\varepsilon_\pi = 1.14\%$ in the 2 GeV/$c < p \leq 3$ GeV/c range; and $\varepsilon_\pi = 1.51\%$ in the 3 GeV/$c < p \leq 4$ GeV/c range.

In terms of results obtained for high energy physics detector simulations, distinguishing Geant4 data from real data was a trivial task when compared to the task of particle identification. Similarly, data produced by deep generative models were easily distinguishable from real data; but the obtained results (especially for adversarial autoencoders) appear to be promising enough to pursue in future research.

Keywords

Deep Learning, Convolutional Neural Networks, Particle Identification, High Energy Physics Detector Simulations, Generative Adversarial Networks, Variational Autoencoders, Adversarial Autoencoders, Geant4, ROOT, AliROOT, Tensorflow, Keras

TABLE OF CONTENTS

1 INTRODUCTION 8

1.1 BACKGROUND 8

1.2 AIMS 8

1.3 SUMMARY OF WORK DONE & MAJOR FINDINGS 9

1.3.1 Particle Identification 9

1.3.2 High Energy Physics Detector Simulations 10

1.3.3 The Structure and Organisation of this Thesis 10

2 HIGH ENERGY PHYSICS & CERN 12

2.1 THE STANDARD MODEL OF PARTICLE PHYSICS 12

2.1.1 Introduction 12

2.1.2 The Fundamental Particles 12

2.1.3 The Fundamental Forces 15

2.1.4 The Higgs Boson 15

2.1.5 Other Subatomic Particles: Baryons and Mesons 16

2.2 THE QUARK GLUON PLASMA (QGP) 17

2.2.1 Introduction to QGP 17

2.2.2 QGP, the Big Bang and the Micro Bang 20

2.3 CERN 20

2.3.1 The Large Hadron Collider 21

2.3.2 The CERN Experiments 23

2.3.3 The ALICE Detector & the Transition Radiation Detector 23

2.3.4 Particle Identification in the TRD 35

2.3.5 Methods used in Particle Identification 36

2.3.6 Particle Identification Accuracy 39

2.4 CALIBRATION 30

3 THEORY: STATISTICAL METHODS & MACHINE LEARNING 42

3.1 BACKGROUND: ARTIFICIAL INTELLIGENCE, MACHINE LEARNING & DEEP LEARNING 42

3.2 MATHEMATICAL BASIS: ARTIFICIAL NEURAL NETWORKS 48

3.2.1 Optimization 50

3.2.2 Regularization 56

3.3 CONVOLUTIONAL NEURAL NETWORKS 57

3.3.1 The Kernel Concept and Motivation for CNNs 57

3.3.2 Pooling 58

3.4 STATISTICAL TESTS 42

3.4.1 Hypotheses 42

3.4.2 Significance Level and Power ***Error! Bookmark not defined.***

3.4.3 Statistical Tests for Particle Selection 45

4 IMPLEMENTATION: MACHINE LEARNING FOR PARTICLE IDENTIFICATION 60

4.1.1 Data Extraction 60

4.1.2 Data Structure 60

4.1.3 Data Exploration 61

4.2 PARTICLE IDENTIFICATION: METHODS 65

4.3 PARTICLE IDENTIFICATION: RESULTS 67

4.3.1 Most successful approach 67

4.3.2 Summary of Other Results 70

4.4 CHAPTER CONCLUSIONS 72

5 THEORY: HIGH ENERGY PHYSICS DETECTOR SIMULATIONS 73

5.1 INTRODUCTION 73

5.2 MONTE CARLO SIMULATIONS: GEANT4 73

5.2.1 Background 73

5.3 DEEP GENERATIVE MODELS 74

5.3.1 Background: Latent Variable Models 75

5.3.2 Variational Autoencoders 76

5.3.3 Generative Adversarial Networks 81

5.4 ADVERSARIAL AUTOENCODERS 83

6 IMPLEMENTATION: HIGH ENERGY PHYSICS DETECTOR SIMULATIONS 84

6.1.1 Assessing Simulation Performance 84

6.2 IMPLEMENTATION:GEANT 85

6.2.1 Geant4 Configuration and Simulation 86

6.3 IMPLEMENTATION: VARIATIONAL AUTOENCODERS 87

6.4 IMPLEMENTATION: GENERATIVE ADVERSARIAL NETWORKS 92

6.4.1 Setup of the most successful GAN: 92

6.4.2 Distinguishing GAN-Simulated Data from Real Data 94

6.5 IMPLEMENTATION: ADVERSARIAL AUTOENCODERS 95

6.5.1 Set-up of most successful Adversarial Autoencoder: 95

6.5.2 Distinguishing AAE-Simulated Data from Real Data 97

6.6 CHAPTER CONCLUSIONS 98

7 CONCLUSIONS 100

7.1 MACHINE LEARNING FOR PARTICLE IDENTIFICATION 100

7.2 HIGH ENERGY PHYSICS DETECTOR SIMULATIONS 101

8 BIBLIOGRAPHY 102

ACKNOWLEDGEMENTS 107

ENDNOTES 108

1 INTRODUCTION

1.1 Background

Particle physics is a field of study which investigates the fundamental properties of our Universe at the subatomic scale. With modern advancements at the European Organization for Nuclear Research (CERN) resulting in an unprecedented amount of data being generated during particle collisions at the Large Hadron Collider (LHC), and with the field of machine learning finally entering an era where there is enough compute power cheaply available to deal with such data volumes, various tools and techniques from the discipline of machine learning can now be practically employed towards problems in the arena of particle physics.

1.2 Aims

This Masters project centres around two main aims:

Aim 1: Particle Identification

The first aim of this project focused on the application of machine learning techniques towards particle identification; in particular the classification of electrons (e) versus pions (π) produced during proton-Lead (pPb) collisions during various runs from LHC16q. Various neural network architectures, hyperparameter settings, etc. were assessed by optimising an electron acceptance cut-off point (t_{cut}) in the distribution of the classifying neural network's $P(electron)$ estimates, which minimises the amount of pion contamination (i.e. pion efficiency, ε_π), whilst maintaining high rate of electron acceptance (i.e. electron efficiency, ε_e), specifically $\varepsilon_e \approx 90\%$.

Aim2: High Energy Physics Detector Simulations

The second aim of this project centred around determining whether simulations obtained from Geant4 were as accurate as they are usually assumed to be and, additionally, to research the feasibility of making use of latent variable/ deep generative models for fast simulations in the future. To this end, a wide variety of generative models were prototyped, and the results of a few choice models will be presented in this thesis.

1.3 Summary of Work Done & Major Findings

1.3.1 Particle Identification

The input feature set for particle identification was, for each particle, a set of up to six 2D arrays ($X = (x_{ij}) \in \mathbb{N}^{17\times24}$) of ADC data produced by interactions of the particle within the six layers of the Transition Radiation Detector (TRD), which forms part of the ALICE detector at the LHC at CERN.

A large variety of deep learning classifiers were built towards achieving the goal of maximising electron efficiency (true positive rate, ε_e), while minimising pion efficiency (false positive rate, ε_π). Some of these models were simply fully-connected feedforward neural networks trained on flat, vectorised versions of the abovementioned input arrays; others were trained on the input arrays summarised in various ways; still other neural networks either made use of convolutional layers (both 2D and 1D convolutions) or recurrent layers of LSTM cells to varying extents.

As a sanity check, two non-deep learning methods, i.e. Gradient Boosting Machines and Random Forests were also tested for their usefulness as particle classifiers in this context.

Section 4.4 summarises the results obtained for each of the various models introduced above, with a slightly more comprehensive focus on results obtained by 2D Convolutional Neural Networks, which outperformed all other models at this task.

In order to compare results to previous work done on particle identification based on TRD data, pion efficiency performance was ultimately evaluated over specific ranges of particle momenta; in summary, the lowest pion efficiencies obtained per momentum bin, at an electron efficiency of $\varepsilon_e \approx 90\%$, were as follows:

- $\varepsilon_\pi = 1.2\%$ in the $p \leq 2$ GeV/c range
- $\varepsilon_\pi = 1.14\%$ in the 2 GeV/$c < p \leq 3$ GeV/c and
- $\varepsilon_\pi = 1.51\%$ in the 3 GeV/$c < p \leq 4$ GeV/c range

These specific results were obtained using an incrementally trained convolutional neural network, using Focal Loss as the objective function to be optimized, as described in Section 4.4.1.

1.3.2 High Energy Physics Detector Simulations

As a general purpose Monte-Carlo toolkit to simulate the passage of particles through matter, Geant4 is also widely used for particle physics detector simulations. Since Geant4 has a well-defined interface with the ROOT data analysis framework used by particle physicists at CERN, it is often used to simulate expected measurables relevant during high energy physics research.

In this project, Geant4 was configured to simulate pion tracklet TRD signals. The simulation was configured to load specific environmental- and other variable settings for a specific LHC run conducted in 2016. The accuracy of the obtained simulated data was subsequently assessed by using a convolutional neural network to discriminate between pion tracklets simulated by Geant4 and actual pion tracklets measured by the TRD.

The task of distinguishing Geant4 simulations from true data was trivial compared to the task of particle identification. These results are presented in section 6.2.1.1.

The practical use of deep generative algorithms for HEP detector simulations is currently an active field of research at CERN (see for example [1], [2], [3]). In keeping with the aims of this research avenue, three kinds of deep generative/ latent variable models were prototyped in this project, towards the task of simulating raw TRD data; namely Variational Autoencoders, Generative Adversarial Networks and Adversarial Autoencoders.

Each type of Latent Variable Model was assessed using the same classification strategy outlined above for Geant4 data. In summary, Adversarial Autoencoders performed particularly well, but the practical use of any of these techniques will be contingent on factors such as customisability of simulations and how well they can be made to integrate with existing simulation software and/ or the ROOT framework.

1.3.3 The Structure and Organisation of this Thesis

Chapter 2 introduces the field of High Energy Physics at the hand of the Standard Model of Particle Physics (SM). The fundamental particles and forces are discussed, along with a tabular delineation of major distinguishing features between electrons and pions.

A primordial state of deconfined matter, the Quark Gluon Plasma (QGP), which can be reproduced at the LHC on a miniscule scale, is introduced, along with a quick glance at the current understanding of the origins of our universe.

Then, a brief introduction to CERN, the ALICE collaboration, the TRD detector, as well as specific ways in which particles interact with matter (which allows for their detection) is presented.

Chapter 2 ends with a short overview of the various software platforms currently being utilised at CERN for HEP research.

Chapter **Error! Reference source not found.** [still working on this secdtion, not sure if it's needed?]

2 HIGH ENERGY PHYSICS & CERN

2.1 The Standard Model of Particle Physics

2.1.1 Introduction

The Standard Model of Particle Physics is a framework which allows us to understand the fundamental structure and dynamics of our universe in terms of elementary particles, where all interactions between elementary particles are similarly facilitated by an exchange of particles. In summary, based on our current understanding, our entire universe consists of a very sparse array of fundamental particles once we delve into the subatomic realm [4].

At an energy scale of 10^0 eV, the low energy manifestation of Quantum Electrodynamics (QED, the quantum field theory of the electromagnetic force) allows atoms to exist in bound states with negatively charged electrons (e^-) orbiting in quantized shells around a positively charged nucleus consisting of positively charged protons (p) and electrically neutral neutrons (n), constrained by the electrostatic attraction of these opposing electrical charges [4].

Quantum Chromodynamics (QCD) is the fundamental theory of the strong interaction, which binds protons and neutrons together within the nucleus of the atom. At this energy scale, the weak force causes nuclear β-decays of radioactive isotopes and is involved in the nuclear fusion processes that occur within stars; the nearly massless electron neutrino (ν_e) is produced during both of the abovementioned processes [4].

Therefore, almost all physical phenomena that occur under normal circumstances can be explained by the Electromagnetic-, Strong- and Weak Forces, Gravity (which is very weak, but explain the large-scale structure of the universe), and just four particles: the electron, proton, neutron and electron neutrino [4].

2.1.2 The Fundamental Particles

At higher energy scales, such as those obtained in experiments conducted using the LHC, protons and neutrons are understood to be bound states of truly fundamental particles called quarks, in the following manner: protons consist of two up-quarks and a down-quark p(uud), whereas neutrons consist of two down-quarks and an up-quark n(ddu) [4].

At the lowest energy level of the standard model, the first generation of particles are the electron, electron neutrino, the up-quark and the down-quark; these are currently considered to be truly elementary, in that they cannot be subdivided [4].

Higher energy scales result in the second and third generation of the four elementary particles; these are heavier versions of the first generation: for example, the muon (μ^-) is essentially a version of an electron which is 200 × heavier, i.e. $m_\mu \approx 200\, m_e$. The tau-lepton (τ^-) is the third generation of the electron, and is much heavier, i.e. $m_\tau \approx 3500\, m_e$. These mass differences do have physical consequences, but the fundamental properties and interactions of the various generations remain identical [4].

There hasn't been any evidence of further generations than these three, and so – according to current understanding – all matter in the universe seems to be circumscribed by the following twelve fundamental fermions, reproduced from [4]:

Table 1: The twelve fundamental fermions.

Leptons				**Quarks**		
	Particle		Mass/ GeV	Particle	Q	Mass/ GeV
First Generation	Electron (e^-)		0.005	Down (d)	-1/3	0.003
	Neutrino (v_e)		$< 10^{-9}$ †	Up (u)	+2/3	0.005

Second Generation	Muon (μ^-)		0.106	Strange (s)	-1/3	0.1
	Neutrino (v_μ)		$< 10^{-9}$ †	Charm (c)	+2/3	1.3
Third Generation	Tau (τ^-)		1.78	Bottom (b)	-1/3	4.5
	Neutrino (v_τ)		$< 10^{-9}$ †	Top (t)	+2/3	174

† While it is accepted that neutrinos are not massless, their masses are so small that they have not been precisely determined, however, the upper bounds for the estimated masses for neutrinos are around 9 orders of magnitude smaller than the other fermions [4].

The Dirac equation describes the state of each of the twelve fundamental fermions and indicates that for each fermion there is an antiparticle which has the same mass but opposite charge, which is indicated by a horizontal bar over the particle's symbol, or a charge symbol of the opposite sign, e.g. the anti-down quark is indicated by $\bar{d}$, whereas the antimuon is indicated by μ^+ [4].

Interactions between particles are facilitated by the four fundamental forces, but the effect of gravity at this scale is sufficiently negligible that it can be ignored without loss of accuracy. All particles take part in weak interactions and are therefore subject to the weak force. The neutrinos are all electrically neutral and therefore are not involved in electromagnetic interactions and are, so to speak, invisible to this force. Quarks carry what is termed as "colour charge" by QCD and are therefore the only particles that feel the strong force [4].

The strong force confines quarks to bound states within hadrons; quarks are therefore not freely observed under normal circumstances [4].

2.1.3 The Fundamental Forces

Historically, Newton stated that matter could interact with any other matter without the mediation of direct contact and, similarly, classical electromagnetism explained the electrostatic interaction between particles using fields [4].

Quantum Field Theory circumvents these non-material explanations and encompasses the description of each of the fundamental forces. Electromagnetism is explained by Quantum Electrodynamics (QED), the Strong Force by Quantum Chromodynamics (QCD), the weak force by the Electroweak Theory (EWT), Gravity has not been explained by the Standard Model yet; therefore, Einstein's General Theory of Relativity is still the best explanation of this force, but it falls within the bounds of Classical Physics. As such, the search to incorporate gravity into the Standard Model is an ongoing area of research and has resulted in exciting new theoretical research avenues such as string theory and loop quantum gravity arising [4].

Looking at electromagnetism, the interaction between charged particles occurs via the exchange of massless virtual photons, which explains momentum transfer via a particle exchange and circumvents the issue of a non-physical potential as the medium of interaction [4].

Similarly, there are virtual particles (gauge bosons) for both the Strong Force (i.e. the massless gluon) and Weak Force (i.e. W^+ and W^- bosons, which are around 80 times heavier than the proton; and the Z boson, which facilitates a weak neutral-current interaction). The gauge bosons all have spin 1, compared to the fermions whom all have spin ½ [4].

2.1.4 The Higgs Boson

The Higgs Boson, whose existence was confirmed by the CMS and ATLAS collaborations at CERN in 2012, but proposed in 1964 by three separate theoretical papers, breaks rank with the other particles outlined by the standard model in that it is a scalar particle which endows other standard model particles with mass, a property without which all particles would constantly move at the speed of light, c [4].

On their own, all particles are massless, but by interacting with the Higgs Field, which is always non-zero, the Higgs mechanism gives them their distinguishing masses [4].

2.1.5 Other Subatomic Particles: Baryons and Mesons

An in-depth explanation of all the possible subatomic particles that can be formed by combining the fundamental particles outlined in Section 2.1.2 lies outside the scope of this thesis, but it is warranted to mention them briefly, since one of the subatomic particles studied in this thesis: the pion, π, which manifests in two charged forms ($\pi^+ \; and \; \pi^-$) and one neutral form (π^0) falls in this class.

As mentioned, the nature of the QCD interaction is such that quarks cannot be observed as free particles. Instead they are found as bound states called hadrons. There are only three known hadronic states: baryons, consisting of 3 quarks (qqq), antibaryons, consisting of three antiquarks ($\bar{q}\bar{q}\bar{q}$) and mesons, consisting of an antiquark and a quark ($q\bar{q}$).

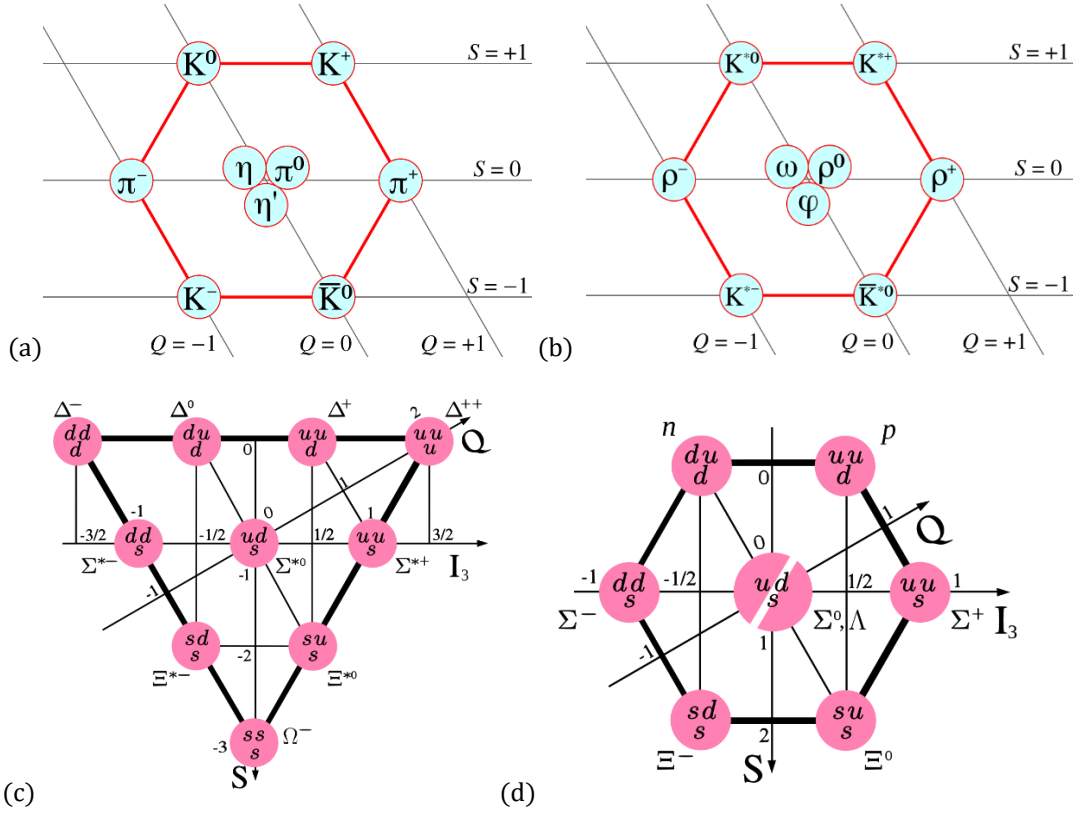


Figure 1: Mesons: (a) spin-1 nonet, (b) spin-0 nonet; and Baryons (c) spin-3/2 uds decuplet, (d) spin-1/2 uds octet

Figure 1 gives an overview of the complexity of the known Mesons, Baryons and Anti-baryons; the three charge varieties of the pion can be seen in the Mesons spin-1 nonet (a). This quick overview allows us to outline the distinguishing features of the two subatomic particles studied in this thesis, in Table 2.

Table 2: Physical Characteristics of electrons and pions

Particle	Electron (e)	Pion (π)

Symbol	e^-	π^0, π^+
Antiparticle	e^+	$\pi^+: \pi^-$ π^0: self
Mass	$0.51099 \text{MeV}/c^2$	$\pi^{\pm}$: 139.57018 MeV/c^2 π^0: 134.9766 MeV/c^2
Spin	$\frac{1}{2}$	0
Electric Charge	$-1e$	$\pi^+: +1e$ $\pi^-: -1e$ $\pi^0: 0e$
Substructure	Elementary particle No known substructure	$\pi^+: u\bar{d}$ $\pi^-: d\bar{u}$ $\boldsymbol{\pi^0: u\bar{u}\ or\ d\bar{d}}$

2.2 The Quark Gluon Plasma (QGP)

2.2.1 Introduction to QGP

The currently held view of the early universe, predicted by the standard model and supported by over three decades of High Energy Physics experiments and lattice QCD simulations, is that directly subsequent to the Big Bang, the universe was composed of a deconfined state of matter, known as the Quark-Gluon Plasma (QGP) [5].

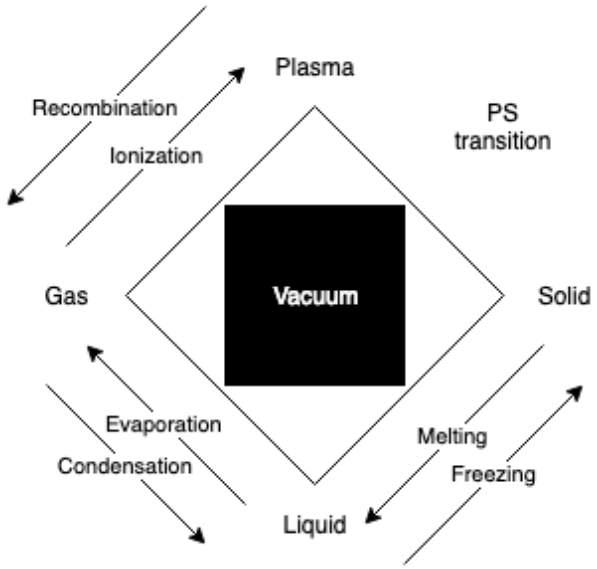


Figure 2: Simplified diagram of classical states of matter and transitions between them, with the Vacuum added as a fifth element, providing the space in which matter exists, reproduced and modified from [6].

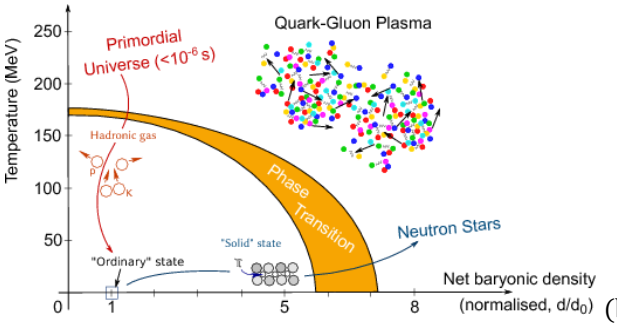
Statistical mechanics understands matter as a system in thermal equilibrium. Global observables, such as net charge, temperature and energy density define the average properties of such a system. As these global observables take on different values, radically different average properties can be held by the system, manifesting as different states of matter bounded by phase boundaries, which matter traverses via phase transitions [6], see Figure 2 for an illustration of this process.

If nucleons (protons and neutrons) were truly fundamental, i.e. if they were not bound states of smaller composite elements (quarks and gluons), a density limit of matter would be reached, when compressing it under ever higher pressure conditions. If, however, nucleons were truly composite states, increasing density would eventually cause their boundaries to overlap and nuclear matter would transition from a stable state of colour-neutral three-quark or quark-antiquark hadronic matter to a state of deconfinement, consisting mainly of unbound quarks [6].

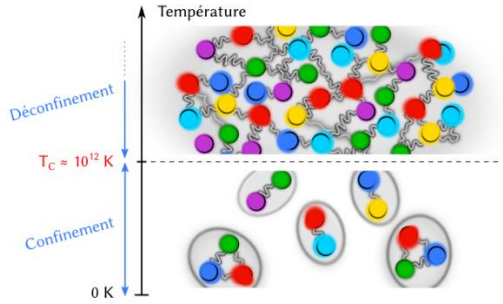
Hadrons all have the same characteristic radius of around 1 fm; it has been found experimentally that increasing density (through compression or heating), can result in the formation of clusters where there are more quarks within such a hadronic volume than logical partitioning into colour neutral hadrons allows for, thus leading to colour-deconfinement [6].

In Figure 3 (a), a simplified phase diagram of hadronic matter is depicted. Within the hadronic phase, there is a baryonic density/temperature boundary where transitions between mesons (colour-neutral quark-antiquark systems) and nucleons (colour-neutral three-quark systems) occur, (not shown in this diagram). The existence of diquarks as localised bound states within the QGP medium allows for yet another state of matter, the colour superconductor, discussion of which is outside of the scope of this dissertation. The phase boundary across which matter transitions from hadronic matter to the QGP is also shown.

(a)



(b)



(b)

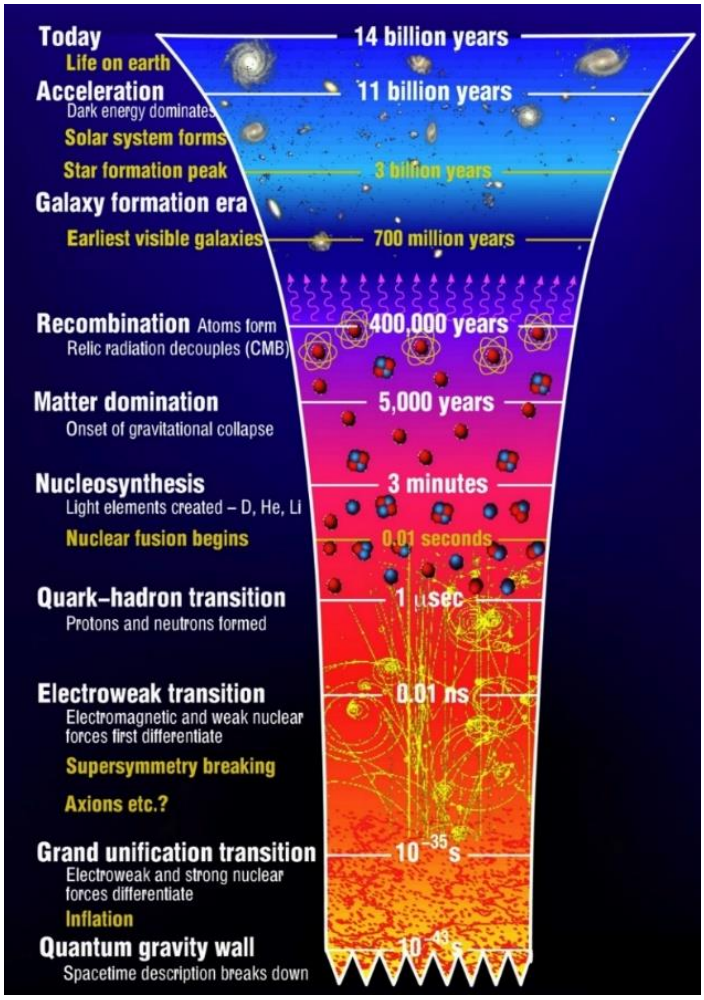


Figure 3: (a) Phase diagram of hadronic matter [7], (b) A simplified phase diagram of the nuclear phase transition along the temperature axis at low baryochemical potential (μB) [8],(c) The evolution of the Universe, from the Big Bang to Modern Day [9]

2.2.2 QGP, the Big Bang and the Micro Bang

It is estimated that, during the Plank epoch, which lasted until $t < 10^{-43}\ s$ after the Big Bang, the prevailing temperature was $T \simeq 10^{19}$ GeV, a temperature so high that the principles of general relativity do not apply, and which cannot be understood with present-day physical theory [10].

Shortly after the Planck epoch, following a short exponential inflation phase, quarks and gluons propagated freely in an early deconfined space-time QGP expansion phase of the Universe, down to a temperature of $T \simeq 150$ MeV, a phenomenon thought to be caused by a change in the vacuum properties of the extremely hot early Universe [5].

To understand how matter was formed in the early Universe, heavy ion collisions, such as the Lead-Lead (Pb-Pb) collisions performed at ALICE, result in a miniscule space-time domain of QGP (which one can refer to as a 'micro bang'), in which local quark-gluon deconfinement occurs. The subsequent hadronization process, during which protons, neutrons and other subatomic particles are formed, leaves traces in the ALICE detector material, giving physicists an indication of how matter arose as the early Universe rapidly cooled down [5].

Since the QGP cannot be detected directly, it is studied via its decay products. Accurately distinguishing between electrons and pions is an important step in this process and as such is the motivation for the particle identification phase of this master's project.

2.3 CERN

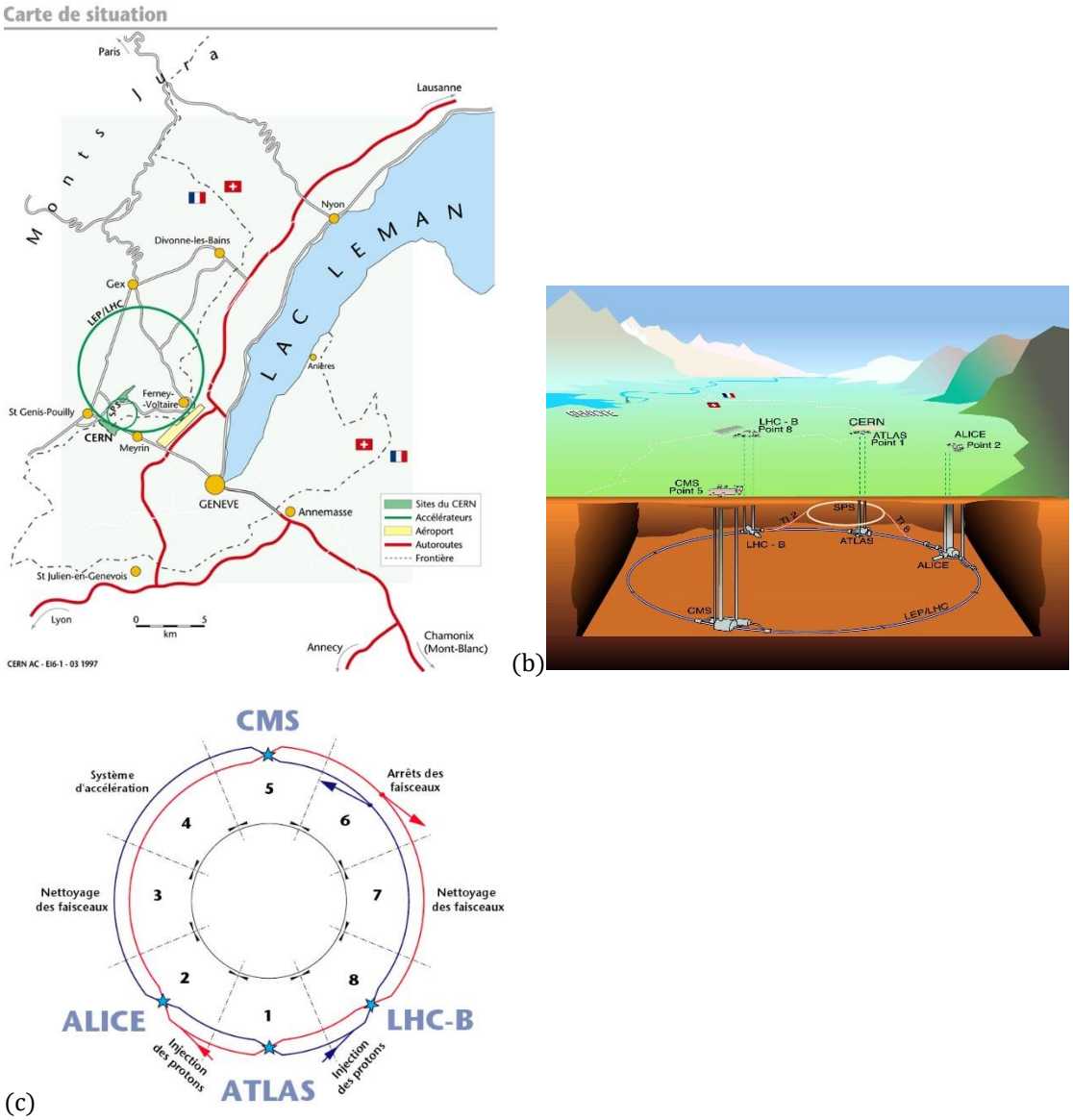
At the end of 1951, a resolution was agreed upon to establish a European Organisation for Nuclear Research (CERN: Conseil Européen pour la Recherche Nucléaire) at an intergovernmental UNESCO meeting in Paris. The final draft of the CERN commission was signed by twelve nations in 1953 [11].

Today, CERN is a truly international organization, with 23 member states (some of which are non-European), who contribute to operating costs and are involved in major decision making; a few countries with associate member status or observer status; and non-member countries with co-operation agreements, including South Africa [12].

CERN's research mandate revolves around finding answers to fundamental questions about the structure and evolution of our universe, as well as its origins; it aims to achieve these goals by providing access to its particle accelerator facilities and compute resources to international researchers, who perform research that advances the forefront of human knowledge, for the benefit of humanity as a whole. As such, CERN is politically neutral and advocates for evidence-based reasoning, knowledge transfer from fundamental research to industry and grass-roots development of future generations of scientists and engineers [12].

2.3.1 The Large Hadron Collider

The LHC, located under the Franco-Swiss border (see Figure 4), boasts an intricate system of particle accelerators and -detectors. The LHC is currently the largest and most powerful particle accelerator in the world [12], with a circumference of ~27km and a centre of mass energy of $E_{CM} = 13\ TeV$ [13].



(a) (b) (c)

Figure 4: CERN's LHC facilities (a) in geographical context [14], (b) as a 3D diagram [15], (c) a diagram showing the LHC's proton injection points and the location of the main experiments' colision points [16] .

Located 50-175 m underground, the LHC is the final step in a chain of successive accelerators feeding beams of accelerated particles from one accelerator into the other at increasing energies, as can be seen in Figure 5.

The LHC's proton source is a bottle of compressed Hydrogen, which releases its contents into a Duoplasmatron device, which subsequently surrounds the H_2 molecules with an electrical field and separates the gas into its constituent protons and electrons [17].

A linear accelerator (LinAc2) injects these protons into a booster ring (PS booster) at an energy of 50 MeV, where proton beams are accelerated up to 1.4 GeV, before being injected into the Proton Synchrotron (PS), which accelerates them up to 25 GeV, the Super Proton Synchrotron (SPS) is the final intermediate step before proton beams enter the LHC: proton beams reach an energy of 450 GeV around this accelerator beam before they begin their 20 minute acceleration around the LHC before reaching an ultimate energy of 6.5 TeV each [18].

An entirely different protocol is employed to generate the lead ions used in heavy-ion collisions (pPb, PbPb) studied at ALICE. A highly pure Lead (Pb) sample is heated up to a temperature of 800°C and the resulting Pb vapour is ionized by an electron current, which manages to strip a maximum of 29 electrons from a single Pb atom. Those atoms with higher resulting charge are preferentially selected and accelerated through a carbon foil, which strips most ions to Pb^{54+}. These ions are accelerated through the Low Energy Ion Ring (LEIR) and subsequently through the PS and SPS, where it is passed through a second foil, which strips off the remaining electrons and passes the fully ionized Pb^{82+} ions to the LHC, where beams of Pb-ions are accelerated up to 2.56 TeV per nucleon ; because there are many protons in a single lead ion, the collision energies reached in PbPb collisions reach a maximum of 1150 TeV [19].

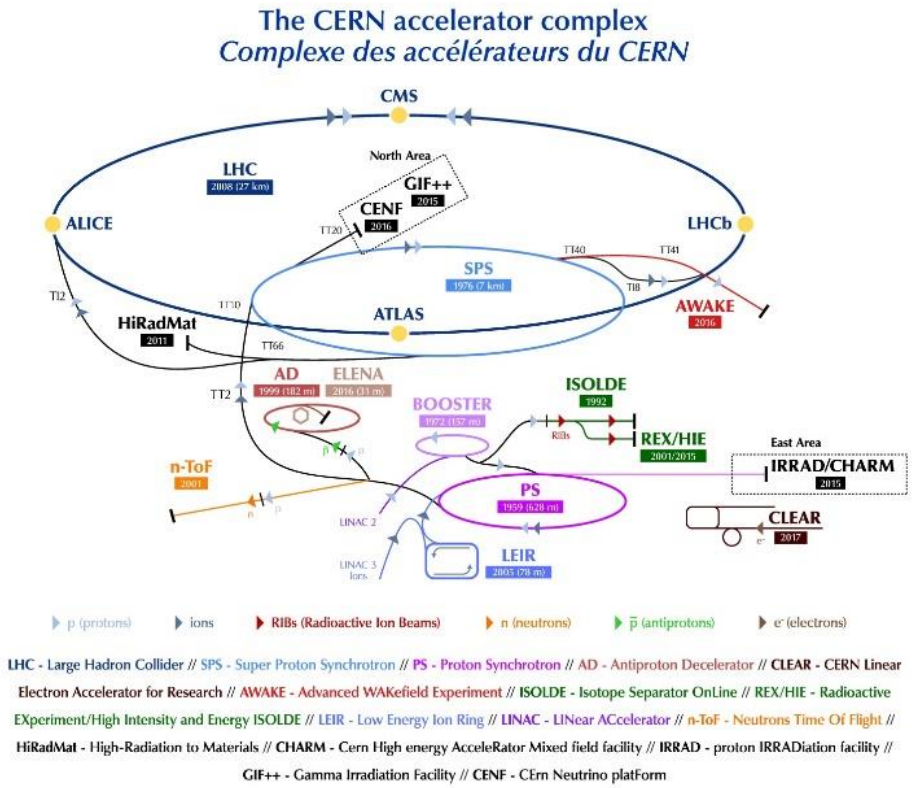


Figure 5: The CERN accelerator complex [20].

In order to achieve these high collision energies, a precise system of 1232 dipole magnets is required to keep particles in their circular orbits, with 392 quadrupole magnets employed to focus the two collision beams. The dipole magnets use niobium-titanium (NbTi) cables at a temperature of 1.9 K (-271.3°C). At these temperatures, the cables become superconducting and the reduced resistance allows the magnetic field to reach the 8.3 T required to bend the beams around the circular LHC ring [13].

The beams themselves are contained within a beam pipe emptier than outer space ($P_{vac} = 10^{-13} atm$) and are accelerated by electromagnetic resonators and accelerating cavities to 99.9999991% of the speed of light, which means that a beam goes around the 26.659 km LHC ring around 11,000 revolutions/second, resulting in an average bunch crossing frequency of 30 MHz and around a billion collisions per second [13].

2.3.2 The CERN Experiments

Collisions at the LHC result in a multitude of particles being produced. Observing the produced particles from different perspectives produces evidence relevant to different research streams; as such, there are several collaborations at CERN, each of which uses detectors with differing attributes to study specific areas within the broad area of fundamental subatomic Physics .

ATLAS and CMS investigate a very broad range of particle physics [21], [22]. Their independent design specifications allow any new discoveries at any one of these detectors, such as the discovery of the Higgs' Boson in 2012, to be corroborated by the other. Other research avenues pursued at these experiments include the search for additional dimensions as well as the constituent elements of dark matter. The ATLAS detector is the largest particle detector ever built, weighing 7000 tonnes with dimensions 46m × 25m × 25m [21].

ALICE and LHCb are the other two main experiments at CERN and are tasked with the discovery of specific physical phenomena [23]. ALICE focuses on the extreme energy densities present during heavy ion collisions, which leads to the production of the QGP [24]. LHCb investigates subtle distinguishing nuances in the matter-antimatter dichotomy, as evidenced by attributes of the beauty quark [25]. In addition, there are several other smaller experiments hosted at the LHC as well.

2.3.3 The ALICE Detector & the Transition Radiation Detector

2.3.3.1 The ALICE Detector System

Colliding heavy ions, such as the Pb-Pb collisions conducted at the LHC and studied at ALICE, offers the most ideal experimental conditions currently achievable for the reproduction of the primordial QGP matter [26]. A transition

from ordinary matter to a state of deconfinement occurs at a critical temperature $T_c \approx 2 \times 10^{12} K$, which is around 100,000 times hotter than the core temperature of our sun [26].

The QGP cannot be probed directly, but is studied via decay particles produced during the hadronization process that occurs as the QGP cools down and quarks and gluons recombine in various ways; the ordinary-matter particles produced in this process interact with various detector elements and leave traces in the detector material that are generally recorded via electronic signals [26].

A simplified diagram of the ALICE detector system is illustrated in Figure 6. The detector weighs 10,000 tonnes and has spatial dimensions 26m × 16m × 16m [24].

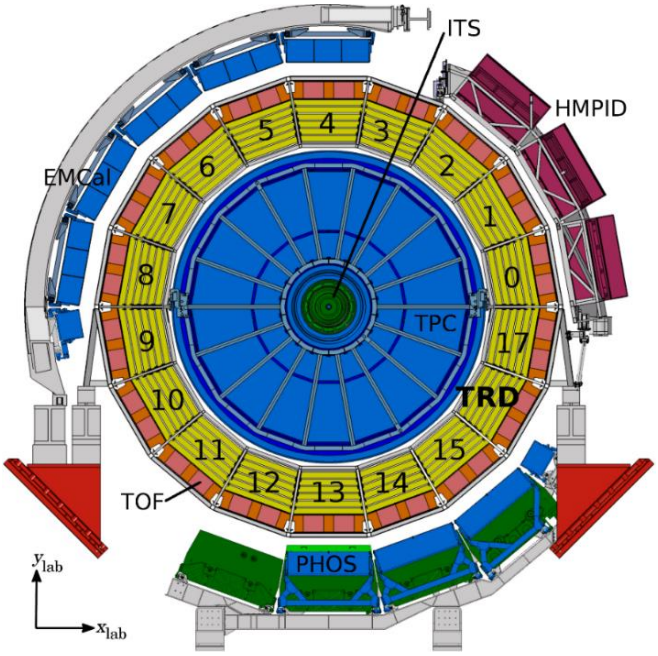


Figure 6: A schematic cross-section of the ALICE detector, with the TRD shown in yellow and its 18 sectors in azimuthal angle numbered.

A uniform magnetic field is applied over the detector, to allow particles to propagate in curved paths through the detector geometry, with the extent of curvature of the particle's track through the detector being inversely correlated to the particle's momentum; additionally, the sign of charge of a particle can also be deduced from its track curvature [26].

The ALICE detector has a total of 18 stacked subdetectors involved in specific particle tracking tasks, these are broadly divided into: Tracking Systems, situated closest to the collision area, which make use of digital track-reconstruction of particle-detector interaction traces to indicate the path of a particle; these are followed by Electromagnetic and Hadronic Calorimeters, through which particle cascades are generated as particles enter and are absorbed by the calorimetric material, with the magnitude of a particle's energy deposition acting as the signal

in these subdetectors; all of which is surrounded by the Muon System in the outermost layer (since muons interact very weakly with matter and therefore generally travel much further through the detector system) [26].

High momentum resolution is obtained in all the detector elements over the high multiplicity densities (number of particles produced per unit volume) present in heavy ion collisions [27]. In addition to heavy ion collisions, lighter ion- as well as proton-nucleus and proton-proton collisions are also performed at ALICE, and this entire momentum range can be accurately measured by the ALICE detector [27].

2.3.3.1.1 The Transition Radiation Detector

Electron identification and triggering capabilities provided by the ALICE TRD enables the in-depth study of physical phenomena such as jets, the semi-leptonic decay of heavy-flavour hadrons and the di-electron mass spectra of heavy quarkonia; in turn, these phenomena act as probes to study the Quark Gluon Plasma [28].

More specifically, at particle momenta above 1 GeV/c, the pion rejection strategy for electron identification employed by the Time Projection Chamber (TPC), is no longer sufficient. The TRD's main goal is to expand the range of the ALICE Collaboration's Physics objectives by providing accurate electron identification capabilities at these high momenta, by supplementing its own data with data obtained from the Inner Tracking System (ITS) and TPC; as well as the operation of event triggers that determine whether data from a specific collision should be kept, based on measurements such as collision centrality, amongst others. As an added benefit, the TRD informs the ALICE central barrel's calibration, and the data it produces is used extensively during track reconstruction and particle identification [29].

2.3.3.1.1.1 TRD Design Synopsis

Pseudorapidity coverage in the TRD is similar to the other detector elements in the central barrel, i.e. $|\eta| \leq 0.9$. The space between the Time of Flight (TOF) and TPC detectors is filled by the 6 layers of the TRD, which are subdivided in azimuthal angle into 18 sectors, with an additional segmentation into 5 sectors occurring along the z-axis (parallel to the beamline). So, in total, there are $18 \times 5 \times 6 = 540$ individual detector elements in the TRD [29] at a radial distance of 2.9 – 3.7 m from the beam axis [28]. The TRD is shown in the context of the full ALICE detector in Figure 6 (highlighted in yellow), illustrating its 18 subdivisions in ϕ-direction, as well as the six-layer architecture of each of these sectors. Figure 7 gives additional detail about the TRD's design: the hierarchical multi-component structure of the TRD detector is shown in Figure 7 (c); the 5 subdivisions of a TRD supermodule along the z-axis, as well as its six stacked layers is shown in Figure 7 (a); and the slightly angled design of a supermodule cut in ϕ-direction resulting in slightly wider top layers is shown in Figure 7 (b).

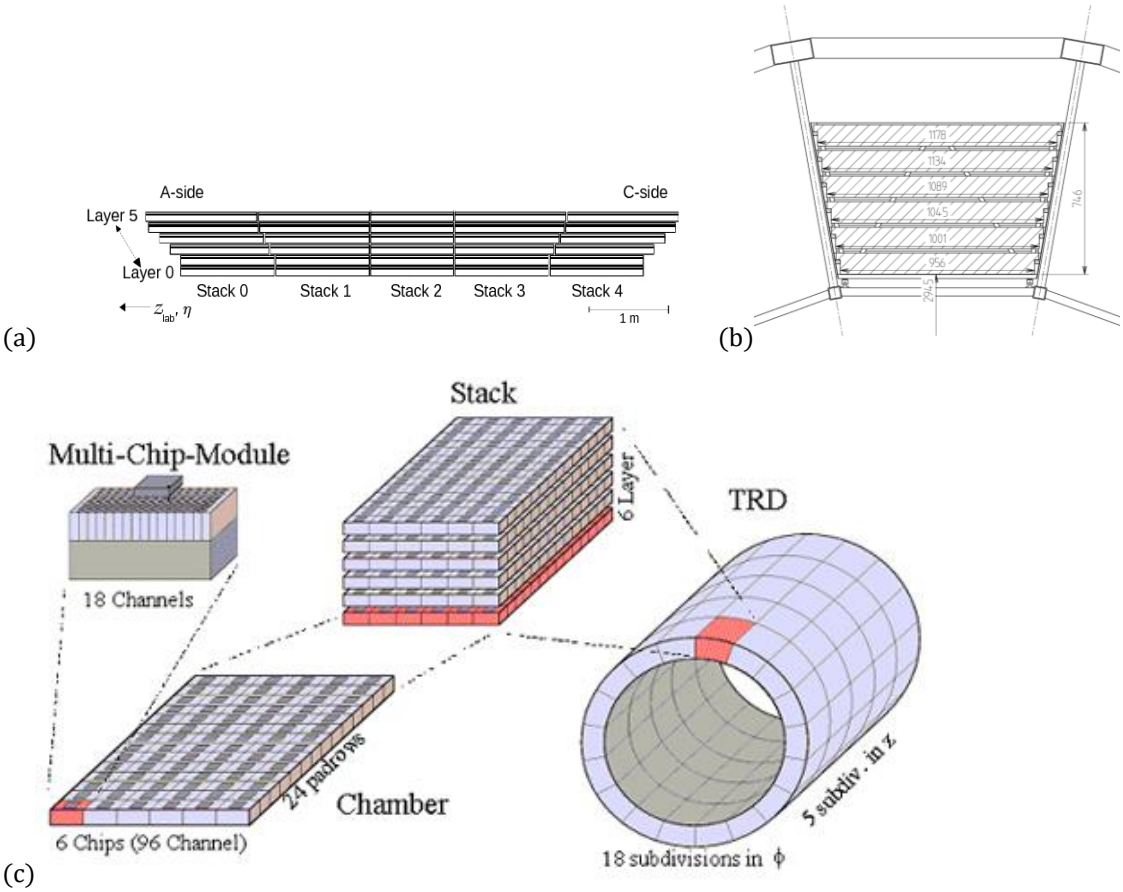


Figure 7: (a) Schematic cross section (longitudinal view) of a TRD supermodule, (b) TRD supermodule cut in ϕ-direction and (c) the hierarchical substructure of the TRD

Each individual detector element consists of the following broad components: 1) a radiator (4.8 cm thick) and 3 cm drift region, 2) a 0.7 cm multiwire proportional readout chamber and 3) front-end electronics to convert from an amplified particle energy-deposition signal to a digital signal, which is eventually stored if deemed interesting by the multi-tiered TRD trigger system [29].

2.3.3.1.1.2 TRD Measurement Mechanism

Interactions of Particles with Matter

In order to study subatomic particles, they need to be detected. Most particles produced during High Energy Physics Experiments are unstable and therefore decay within a specific characteristic mean lifetime τ. Those particles with $\tau > 10^{-10}s$ will traverse several meters before decaying and are therefore directly detectable by particle detectors. Particles with shorter lifespans are usually detected indirectly, by the interaction of their decay products with detector material [4].

The Bethe-Bloch Curve

The Bethe-Bloch equation describes the energy lost by a charged particle moving at relativistic speed through a medium, as a result of electromagnetic interactions with atomic electrons. A single charged particle with velocity $v = \beta c$, passing through a medium with atomic number Z and density n, will lose energy as a result of ionisation of the medium, as a function the distance travelled in the medium, according to the Bethe-Bloch formula (Equation 1) [4]:

$$\frac{dE}{dx} \approx -4\pi\hbar^2 c^2 \alpha^2 \frac{nZ}{m_e v^2}\left\{ln\left[\frac{2\beta^2\gamma^2 c^2 m_e}{I_e}\right] - \beta^2\right\}$$

Equation 1

In Equation 1, I_e is the effective ionisation potential of the medium. The $\frac{1}{v^2}$ term explains the high energy loss for low energy particles. For high energy particles, where $v \approx c$; $\frac{dE}{dX}$ depends logarithmically on $(\beta\gamma)^2$, which is defined by Equation 2. This explains the relativistic rise seen in Figure 8, which illustrates the characteristic energy loss curves for various subatomic particles as measured by the TPC at $\sqrt{s} = 7\ TeV$, including the two subatomic particles studied in this project, the pion π and the electron e .

$$\beta\gamma = \frac{v/c}{\sqrt{1-(\frac{v}{c})^2}} = \frac{p}{mc}$$

Equation 2

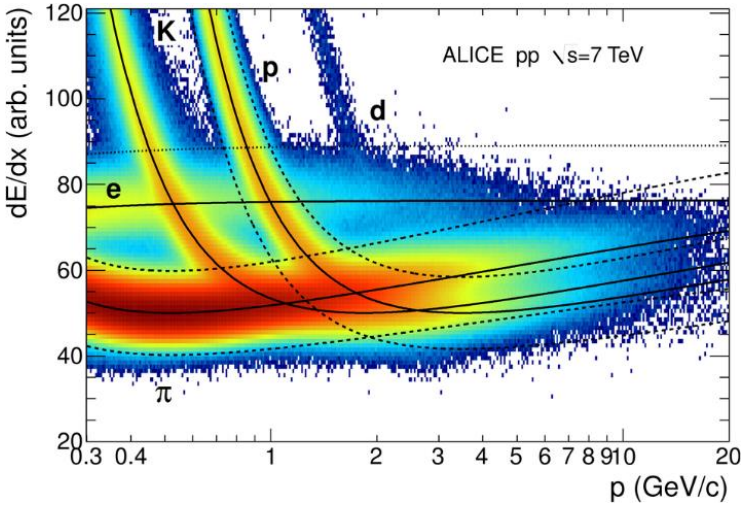


Figure 8: Bethe-Bloch curve for various subatomic particles as measured by the ALICE TPC at $\sqrt{s} = 7TeV$

Transition Radiation

Transition radiation is emitted by a charged particle as it traverses the boundary between two mediums with different optical properties; no significant energy loss occurs in this process, but the resultant radiation is an important aid in detecting charged particles in HEP experiments [30].

For relativistic particles, the photons emitted in this process extends into the X-ray domain and are highly forward-peaked compared to the direction the particle is moving in; transition radiation yield is increased by stacking multiple radiative boundaries in gas detectors, such as the Transition Radiation Detector (TRD) at ALICE, and placing high atomic number (high-Z) gases within subsequent chambers to absorb the emitted X-ray photons [31].

The drift time of gas particles within the MWPC provides fine-grained positional information about where the particle tracklet passed through the radiator. The detected signal takes the form of charged gas molecules (ionized via interaction with particles or transition radiation photons and amplified through a chain of interactions between gas molecules), finally being absorbed by electrodes on the pad plane before moving through an amplification region where their accelerated movement towards negatively charged wires (anode) cause an avalanche which provides gas amplification in the range of 10^4, this process is shown in Figure 9. The positive ions produced during the avalanche process move toward the surrounding electrodes and induce a positive charge on the pad plane.

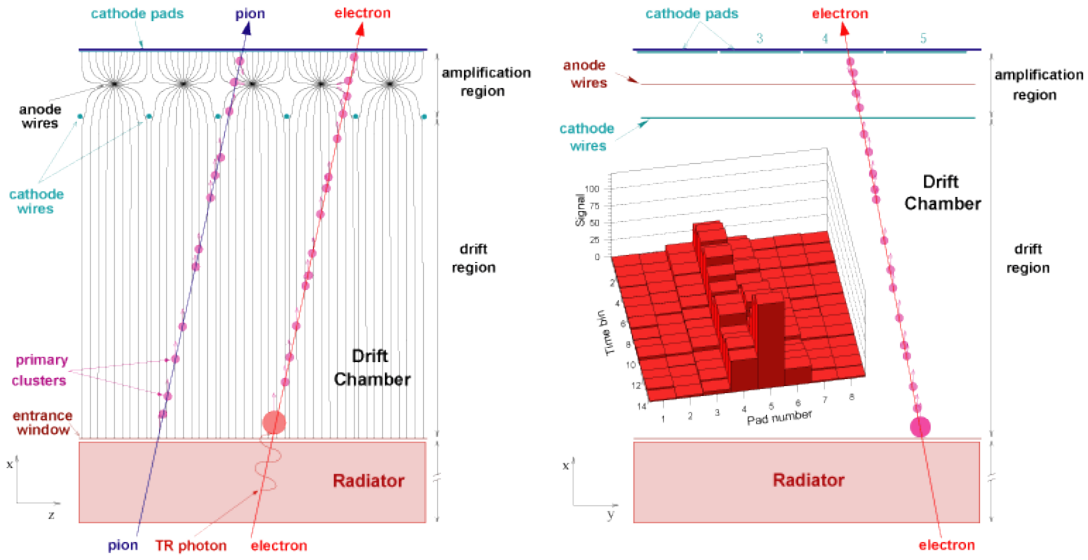


Figure 9: A schematic representation of the components in an MWPC module

Determining precisely the location of the avalanche in azimuthal angle requires that the induced charge be shared by several readut pads on the pad plane and determining the tracks'angle in the rφ-plane is achieved by measuring the azimuthal position in each of 15 time bins. Ideally charge should be shared by two or three adjacent pads, since a poorer signal-to-noise ratio results if more than three pads fire, this also increases the amount of data produced and limits the separation of individual tracks. A particle's momentum is determined by the particle track's deflection angle, because Xe ions'drift velocity is known very precisely, the r-coordinate can be determined by the arrival time.

One of the main aims of this thesis is distinguishing electrons from pions. This is facilitated by the fact that electrons and pions have different characteristic energy loss curves (particularly at low momenta, electrons have a higher relative energy loss); as well as the fact that electrons emit transition radiation and pions don't.

2.3.3.1.1.3 TRD Front-End Electronics

In order to convert the analog signal discussed in Section 2.3.3.1.1.2 to a digital signal which can be stored and analysed, a complex system of on-detector front-end electronics (FEE)

1. assists the integrated ALICE triggering system via tracklet search and electron candidate identification (the TRD trigger generates a level-1 accept (L1A) and has to occur on a timescale of $6\mu s$) [29]
2. identifies the TR signal, while also providing tracking-, momentum- and mass reconstruction capability [29].

Figure 10 gives an overview of the logical components of a single channel of the TRD's 1.156×10^6 channel FEE.

The major building blocks of the FEE are:

1. A charge sensitive PreAmplifier/ShAper (PASA)
   The signals from the detector pads are amplified by a charge-sensitive preamplifier, this is followed by a pole-zero cancellation circuit which ensures a more symmetrical output and two second order shaper-filters which ensure a shaped output pulse, finally, an output amplifier then delivers a 10 bit differential 1V range output signal to the ADC, depending on the ADC's driving capabilities and output levels
2. An analog chip
3. A 10 MHz Analog to Digital Converter (ADC) converts the analog input it receives from the PASA to the tracklet preprocessor
4. The digital circuitry required to process and store data for subsequent readout: The Tracklet Preprocessor (TPP) processes data during drift time at digitisation rate, this prepares data to be sent to the Tracklet Processor, a micro CPU operating at 120 MHz, which processed data from all time bins to determin candidate tracklets, which are shipped to the Global Tracking Unit (GTU) for storage in memory until readout.

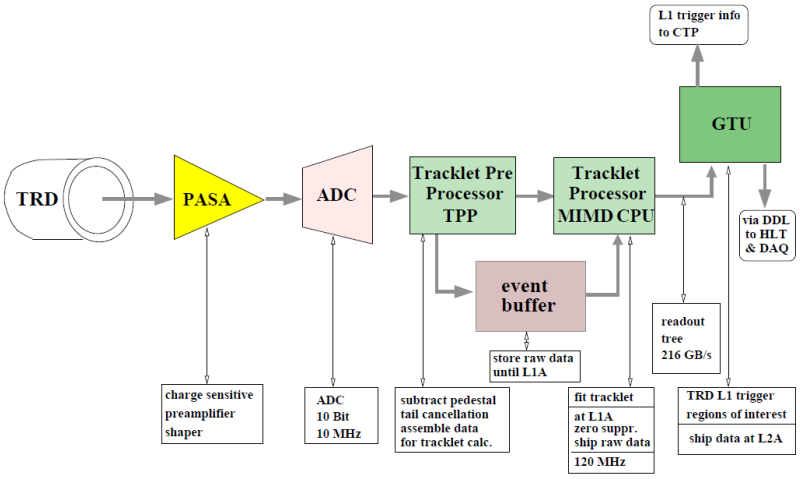


Figure 10: Diagrammatic representation of the logical components of the TRD front-end electronics.

It should also be noted that the following data filtering steps occur as part of a digital filter chain implemented on the TRD FEE:

1. The pedestal of the signal is equilibrated
2. Local gain variations are corrected for by a gain filter
3. Ion tails are suppressed by a tail cancellation filter

Additional calibration of TRD data is discussed in the next section.

2.3.3.1.1.4 TRD Data Calibration

There are four basic parameters involved in the calibration of TRD data (shown in Figure 11 at the hand of the chamber cross section and the average pulse height plot for pions):

1. Time offset
    - The peak in the average pulse height plot at $0.5\mu s$ (as shown in Figure 11) corresponds to charges from both sides of the anode wires. The position of this anode peak provides the time offset parameter, since it represents the distance from the anode wires
2. Drift velocity
    - At 2.8 μs there is an edge representing the entrance window. Drift velocity is inversely proportional to the difference in time between the entrance window edge and the anode peak
3. Gain
    - Gain is proportional to the mean pulse-height
4. Noise
    - The width of the pedestal is proportional to pad noise (seen at the bottom left of the pulse height plot, before the average pulse height starts to rise from zero)
5. Lorentz angle
    - (Not seen in Figure 11). The presence of a magentic field B perpendicular to the electric field E, which attracts ionization electrons to the anode wires $|E \times B| > 0$ leads to a Lorentz angle of around 9°, which needs to be determined in order to reconstruct tracklets.

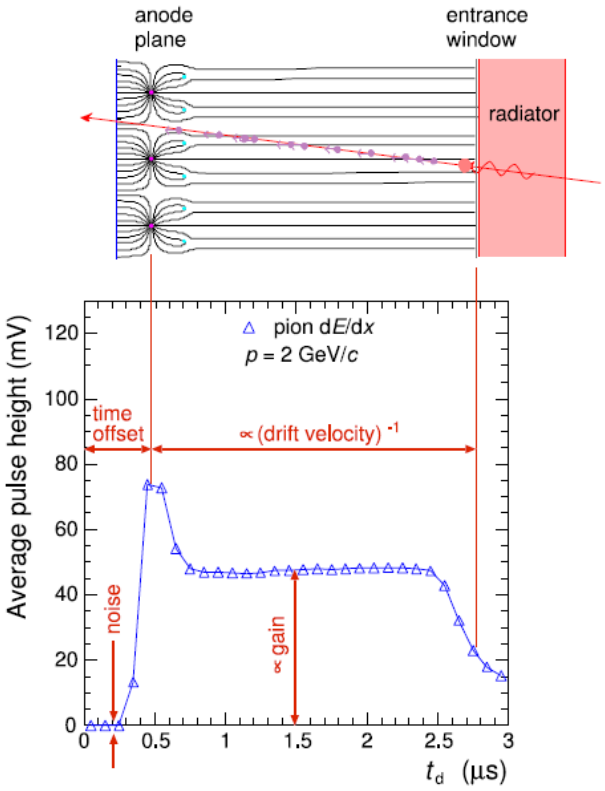


Figure 11: Main TRD signal calibration parameters, shown at the hand of average pulse height as a function of drift time plot for pions, in addition the chamber cross section is shown at the top for better understanding [31]

Routine calibration performed for the TRD is summarised in Table 3. To achieve the highest possible resolution, time offset, chamber status, gain, Lorentz angle and drift velocity are calibrated offline for each run

Table 3: Calibration parameters, along with their associated data sources and methods for implementation

Input data	Parameters	Implementation
Pedestal runs	Pad noise, pad status	Once a month, random events triggered, data collected without zero suppression. Baseline position of PASA and noise (μ_{noise} =

		1.2 ADC counts) determined Faulty pads (faulty FEE connection or faulty FEE, excessive noise, bridged with neighbor) recorded in OCDB and treated accordingly
Runs with ^{83m}Kr in the gas chamber	Relative pad gain	Pad-by-pad calibration: decay electrons from radioactive gas are measured, histogram binning of signal and horizontal stretching of reference distribution is performed, this stretching factor indicates pad gain (Figure 12 shows the effect of Kr-calibration in one TRD readout chamber, Figure 13 and Figure 14 show how pad gain factor is determined and relative gain factors across pads in a single TRD chamber)

Physics runs	Chamber status, time offset, drift velocity, Lorentz angle, gain	Inidividual chambers' anode and drift voltages adjusted once per annum, voltage also continuously adjusted based on atmospheric pressure

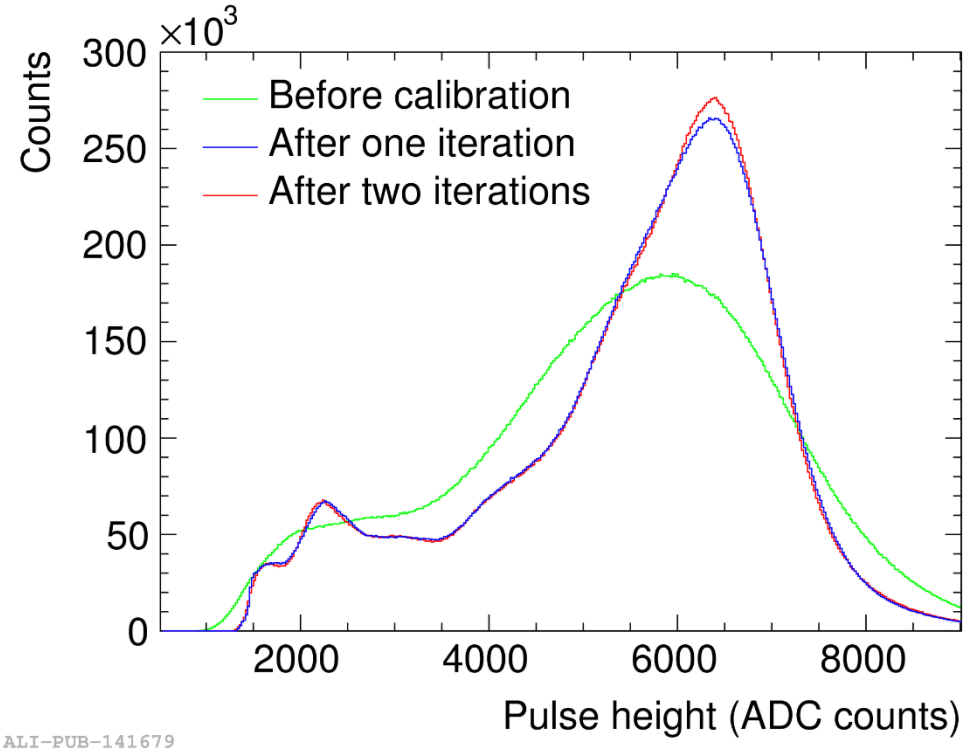


Figure 12: Pulse height spectrum before the Kr-based calibration, after one and after two iterations (calibrations performed in consecutive years) for one TRD read-out chamber

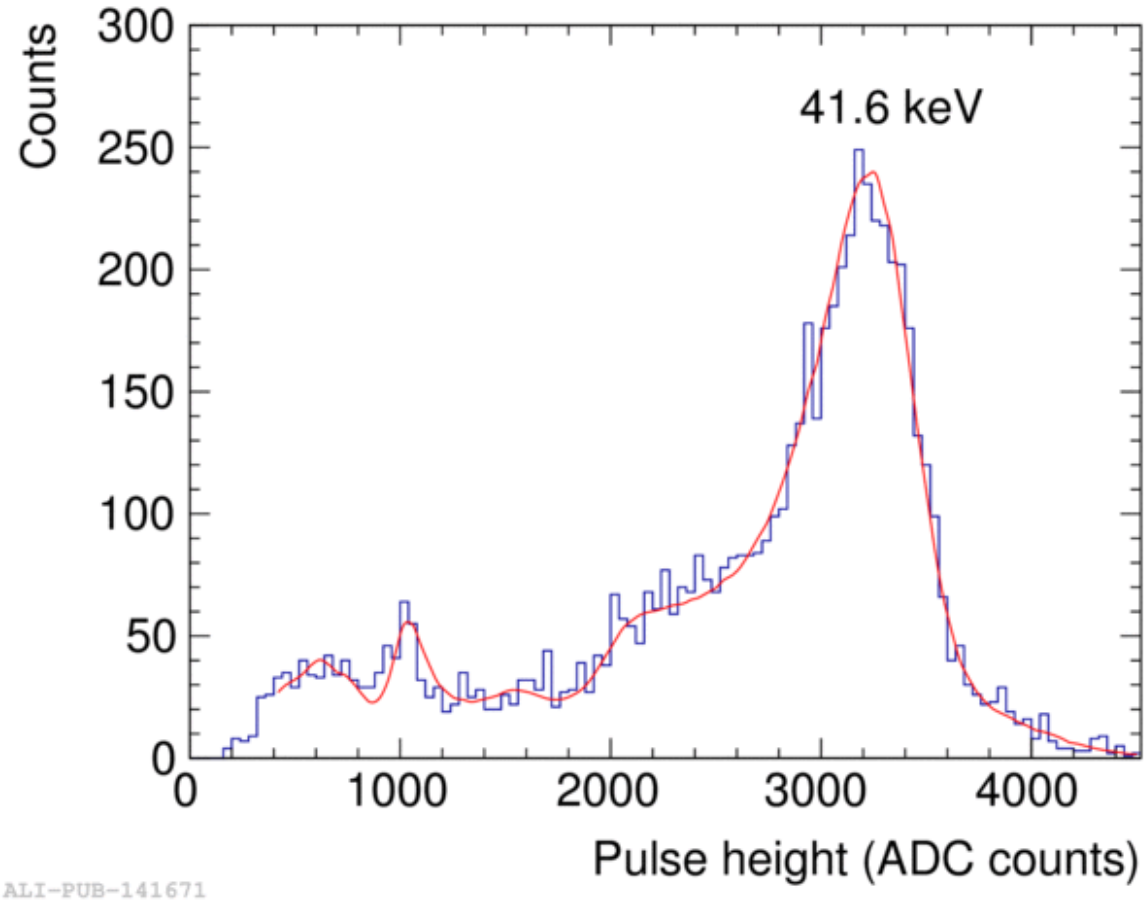


Figure 13: Pulse height spectrum accumulated for one pad during the Kr-calibration run. The smooth solid line represents the fit from which the gain is extracted

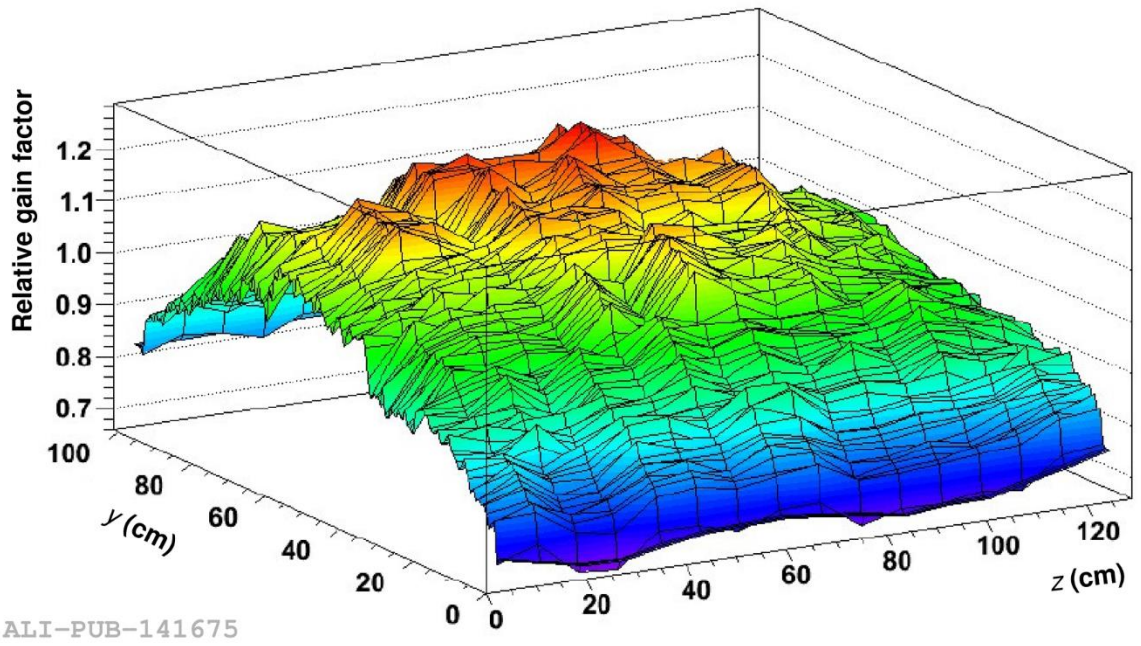


Figure 14: Relative pad gains for one chamber calibrated with electrons from Kr decays

2.3.4 Particle Identification in the TRD

At momenta $p > 1$ GeV/c, the TRD provides electron identification via the measurement of transition radiation. At these momenta, pion rejection achieved in the TPC via specific energy loss as per characteristic Bethe-Bloch dE/dx curves for pions vs. electrons becomes less accurate.

The temporal information contained in the drift time dimension of the TRD signal provides information about the depth in drift volume where ionization signals were produced; this allows for the separation of the contribution of the particle-specific ionization energy loss ($^{dE}/_{dx}$) to the signal, from the contribution made by Transition Radiation photons and is therefore an important factor in distinguishing between electrons and pions [31]. The electron identification capability is also used to trigger at level 1 [28].

As explained in section 2.3.3.1.1.3, the TRD signal originally induced on the segmented cathode plane is captured and processed by a PreAmplifier-ShAper (PASA) circuit, this processed signal is then digitized by a 10 MHz ADC (Analog-to-Digital Converter) to take samples of the time-evolution of the signal at defined 100 ns intervals [28].

Figure 15 shows the time evolution of the abovementioned signal at $P = 2\ GeV$, for both electrons and pions, by plotting the average pulse height for each particle type over time. The initial peak seen in earlier time-bins on the graph originates from the amplification region of the detector and the plateau that follows is caused by particles moving through the 3 cm drift region in the detector.

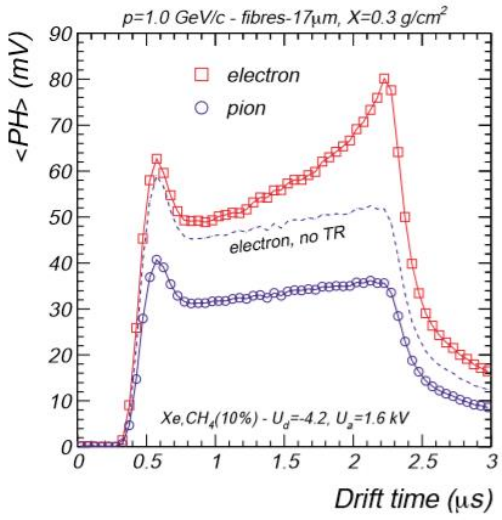


Figure 15: Average pulse height as a function of drift time for electrons and pions (both at $p = 1\text{GeV}/c$) [29].

Also evident from Figure 15 is that, in this momentum region, the average pulse height of electrons is much higher than that for pions, because electrons have higher characteristic energy loss ($^{dE}/_{dx}$) in this region.

An average of one transition radiation photon in the X-ray domain will be emitted by an electron traveling at a highly relativistic speed (above $\gamma \sim 800$), since it will cross many dielectric boundaries in the radiator portion of a detector element, the absorption of this type of photon is evidenced by an increasing average energy deposition at later times in Figure 15, since it will be absorbed preferentially close to the radiator, adding its signal to the ionization energy of the track [28].

It should be emphasized that Figure 15 shows the *average* pulse height over time. In truth, there are large fluctuations around this average, as can be seen in Figure 33.

2.3.5 Methods used in Particle Identification

Currently, the following methods are employed in production for particle identification (specifically, distinguishing between electrons, e and pions, π) based on TRD data:

1. One-, two-, three- and seven-dimensional likelihood estimations
2. Neural Networks
3. Truncated mean of the signal (this is a specialised method which is used to optimise the identification of particles other than e, i.e. Kaons (K), pions (π), protons (p) and muons(μ) and does not perform well at distinguishing e from π)

2.3.5.1 Likelihood Methods

The cencepts of Likelihood and Maximum Likelihood are discussed in Section ?.

2.3.5.1.1 One-dimensional Likelihood (LQ1D)

One dimensional likelihood estimation is performed based on the total integrated charge left by a particle in a single chamber in the TRD (i.e. a single tracklet). Figure 16 shows that electrons have on average a higher charge deposit, because they experience higher characteristic energy loss in this momentum range, as well as the fact that they emit Transition Radiation and pions don't.

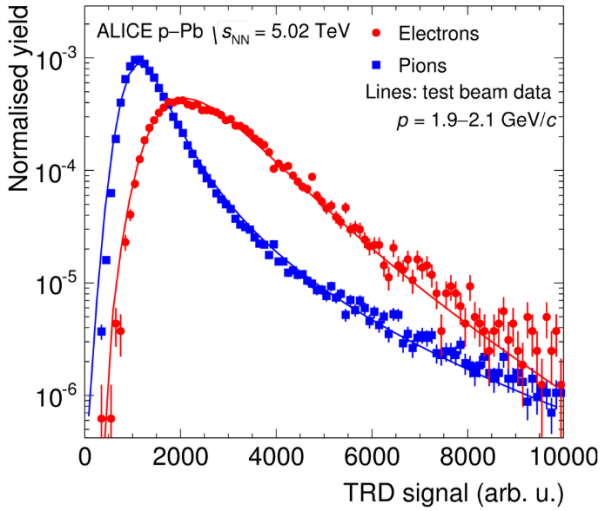


Figure 16: Total integrated charge, normalised to tracklet length, measured in a single read-out chamber for electrons and pions in pPb collisions at $\sqrt{s_{NN}}$ ≈5.02TeV. Test beam measurements were scaled by a common factor to compensate gor gain differences.

The reference distributions allow maximum likelihood estimations to be carried out on each particle traversing the TRD, i.e. the likelihood of it being a muon, pion, kaon or an electron. Pions are rejected based on momentum-dependent cuts based on the likelihood for electrons, taking into account an electron efficiency score calculated using clean pion and electron reference samples, which are obtained by keeping tracks originating from the following V_0 decays: $\gamma \rightarrow e^+e^-$ and $K_s^0 \rightarrow \pi^+\pi^-$ [28]. V_0 candidates are reconstructed using a secondary vertex finder algorithm [33]. More information about obtaining clean reference data for particle identification can be found in [34].

2.3.5.1.2 Two-, three- and seven-dimensional Likelihood (LQ2D)

Two-, three- and seven-dimensional likelihood methods each take the temporal evolution of the signal (Figure 15) into account by splitting the signal into two, three and seven time-bins respectively, summing the charge in each bin and calculating the likelihood based on pure pion- and electron samples from collision data.

2.3.5.2 Neural Networks

A comprehensive overview of the mathematics behind artificial neural networks is given in Section 3.3.

The currently used neural network used in production for particle identification was trained using a similar approach as LQ2D, but instead of splitting and summing over two time-bins, the input feature-set to the neural network was obtained by splitting into seven time-bins and summing the charge over each bin, respectively.

Since particle identification using neural networks is a major aim of this thesis, it is worth mentioning some previous theses which were focussed on similar aims. Pertinent results from these theses ([34], [35]) are summarised in section 4.2

2.3.5.3 Truncated Mean

The truncated mean method informs particle identification, based on the expected truncated dE/dx value per particle species (this technique is mainly used for the identification of hadrons, such as pions, kaons and protons). Calculating the truncated mean of the observed dE/dx distribution involves making a cut on a specified percentage off the higher end of the distribution of empirically observed dE/dx.

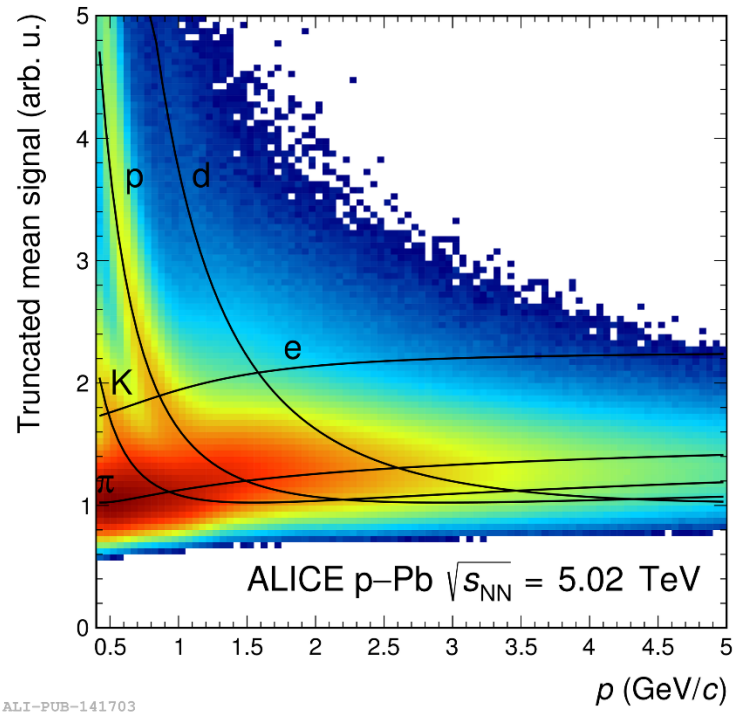


Figure 17: Truncated mean signal as a function of momentum for p-Pb collisions at $\sqrt{s_{NN}} = 5$TeV. The solid lines represent the expected signals for various particle species .

Observed dE/dx is influenced by Landau-distributed ionization fluctuations, Gaussian-distributed detector-resolution fluctuations, fluctuations in gas gain and other effects. Since the distinguishing transition radiation signal produced by electrons will generally be lost during the truncation procedure, this method is less accurate for distinguishing electrons from pions than other methods.

2.3.6 Particle Identification Accuracy

To calculate the accuracy of the abovementioned methods, clean reference samples were used. The separating power of these approaches are often expressed as pion efficiency (the fraction of pions incorrectly classified as electrons, i.e. the false positive rate or fallout rate) at a specific electron efficiency (the fraction of electrons correctly identified, i.e. the true positive rate or sensitivity) [28].

It is important to note that pion suppression (the inverse of pion efficiency) is hampered when a particle passes through fewer than the available six layers of the TRD, and that electron efficiency is sometimes sacrificed during analysis to obtain a more pure sample [28].

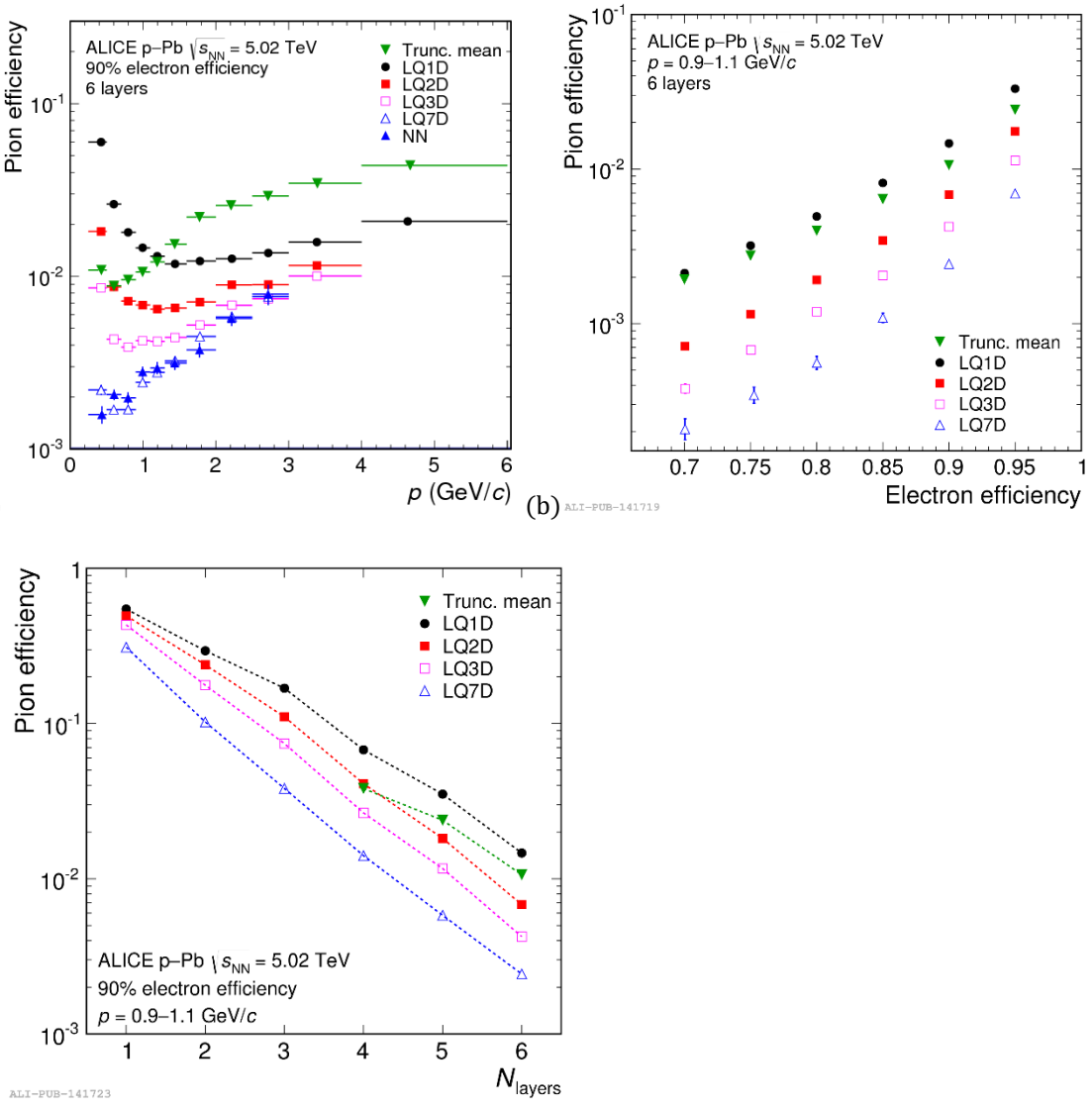


Figure 18: (a) Particle identification performance of the TRD, based on various methods discussed

Figure 18 shows how pion efficiency depends on momentum for the four methods under discussion, data is plotted for samples where an electron efficiency of 90% was obtained. LQ1D and LQ2D are not accurate at very low momenta, but their performance is quite good at slightly higher momenta where the emission of transition radiation commences, their separating power decreases again at higher momenta as transition radiation production saturates and pions deposit more energy, making it harder to tell them apart. The truncated mean method performs poorly at high momenta, since transition radiation with its attendant high charge deposition is more likely to be removed during the truncation procedure [28].

It is also clear from this plot that the misidentification of pions as electrons (False Positive Rate) is reduced substantially by the LQ2D and Neural Network techniques, compared to truncated mean- and LQ1D methods, and that the temporal evolution of the signal is therefore a highly informative feature for particle identification [28].

2.3.6.1 ROOT

ROOT is an object oriented data analysis platform developed in C++ for High Energy Physics implementations; in addition to its data analysis capabilities, ROOT is also used to transform the petabytes of raw data from collision events at the LHC into more compact and useful representations [35].

The basic ROOT framework provides default classes for most common use-cases and as the HEP community pushes research into new frontiers, they can use the object-oriented programming (OOP) approach followed by ROOT to make use of sub-classing and inheritance to extend existing classes. Similarly, the concept of encapsulation keeps the number of global variables to a minimum and increases the opportunity for structural reuse of code [35].

ROOT libraries are designed with minimal dependencies and as such are loaded as needed. At runtime, libCore.so (the core library) is always invoked; it is composed of the base-, container-, metadata-, OS specification- and ROOT file compression classes. Additionally, the interactive C++ interpreter library libCling.so is used by all ROOT 6 applications, it features a command line prompt with just-in-time interactive compilation to facilitate rapid application development and testing.

When building executables, libraries containing the needed classes are linked to. Extensive documentation is available online at the ROOT reference guides for ROOT 5 [36], the version of ROOT developed and used for LHC run 1 and run 2; and ROOT 6 [37], the version of ROOT developed for LHC run 3, scheduled to start in 2021 after the second long shut down period (LS2).

2.3.6.1.1 AliROOT

It is a common concept for each experiment at CERN to build software specific to their needs on top of the base ROOT architecture; as such, AliROOT and AliPhysics are built on top of ROOT to provide functionality specific to the ALICE collaboration.

C++ classes define all the code in ROOT, AliPhysics and AliROOT and enables the user to create variables (data) and functions (methods) specific to each class, as its members. A class's variables are usually accessed via the class's methods [38].

C++ code is split into header (.h) and implementation (.cxx) files, both having the same name as the class being defined. Header files list all the constants, functions and methods contained in a class. Implementation files use a class's methods to set and get variables' values in that class.

The concept of inheritance is frequently utilized to prevent unnecessary repetition of code. Child classes inherit common behaviours and attributes from base/ parent classes and define additional methods and variables that are not common to other classes deriving from the base class.

2.3.6.1.2 O^2 Software for Run 3

LHC run 3, scheduled to start in 2020, will require some upgrades to the ALICE detector to accommodate the much higher interaction rate that is being planned for, in order to more precisely measure attributes of heavy flavour hadrons, low mass di-leptons and low-momentum quarkonia. Since these physics probes have a very low signal-to-background ratio, a continuous readout process could result in upwards of 1TB/s of data being generated by the ALICE detector.

This will result in unique challenges, which will need to be met by an upgraded software framework for run 3 and run 4, known as O^2 (The Online-Offline Software Framework), which is currently being developed.

2.3.6.2 Geant4

Geant4 is a C++ toolkit for simulating how particles traverse through matter. Comprehensive and accurate simulations of particle detectors, using platforms like Geant4, is extremely important, since it provides a theoretical reference against which data can be compared. Should there be any statistically significant discrepancies between simulations and data, it could indicate that phenomena occurred which are not explicable by the Standard Model of Particle Physics and could in rare circumstances lead to the discovery of new fundamental principles of nature [39].

A slightly more in-depth discussion of Geant4 can be found in Section 5.2.1.

3 THEORY: STATISTICAL METHODS & MACHINE LEARNING

3.1 Statistical Methods

3.1.1 Marginal-, Joint- and Conditional Probabilities

Marginal probability denotes the probability of an event occurring, without taking the outcomes of other events into account; the probability that event A occurs is written as $P(A)$, and is simply calculated as the number of times A has occurred divided by the total number of possible events that have occurred.

Joint probability refers to the probability of two or more events occurring simultaneously; the joint probability of event A *and* B occurring is given as $P(A \cap B)$.

Conditional probabilities express the probability of an event occurring given that another event is known to have occurred; the probability of A occurring, *given* that B occurs, is expressed as $P(A|B)$. This is ususally done when we expect or know that the outcome of B will have some influence on the outcome of A.

The conditional probability $P(A|B)$ can be used to calculate the joint probability $P(A \cap B)$, as follows:

Equation 3

$$P(A \cap B) = P(A|B) \times P(B)$$

[32].

3.1.2 Bayes' Theorem

Bayes' theorem allows one to calculate the conditional probability $P(A|B)$ when the joint probability $P(A \cap B)$ is hard to calculate and the reverse conditional probability $P(B|A)$ is known or easier to calculate.

Equation 4

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Here, $P(A)$ is referred to as the prior probability (or evidence) and $P(A|B)$ is referred to the posterior probability (or likelihood), i.e. we can adjust our estimate for $P(A)$, based on other evidence at our disposal.

[32].

3.1.3 Likelihood and Maximum Likelihood Estimation

Given a set of observations $X = (X_1, \dots, X_n)$ of random variables sampled from one of a family of distributions P_θ: $f(\mathrm{x}|\theta)$, $\mathrm{x} = (x_1, \dots, x_n)$ denotes the density function of the data when θ is true.

The likelihood function (Equation 5) is a density function parameterised by a set of parameter values θ, formed from the joint probability of a sample of observed data and should be understood as a function of the parameters given the observed data distribution.

Equation 5

$$\mathcal{L}(\theta|x) = f(x|\theta), \theta \in \Theta$$

Maximum likelihood estimation (Equation 6) is a principle which allows one to estimate $\hat{\theta}$, as the most likely set of parameters given the observed dataset, for a probability distribution parameterised by θ. This is achieved by maximising the likelihood function; presuming that a unique global maximum exists:

Equation 6

$$\hat{\theta} = \arg\max_{\theta} \mathcal{L}(\theta|\mathrm{x})$$

[32].

3.1.4 Hypotheses

Statistical tests are mathematical constructs designed to enable a researcher to make a measurable statement concerning to what extent observed data agrees with probabilistic predictions made about it in the form of a hypothesis [48].

When performing a statistical test, a null hypothesis, denoted as H_0, is put forth, as well as one or more alternative hypotheses, (H_1, H_2, …).

Given a dataset of n measurements of a random variable $x = x_1, \dots, x_n$, a set of hypotheses H_0, H_1 are proposed, each specifying a joint probability density function (p.d.f.), i.e. $f(x|H_0), f(x|H_1)$, …

In order to assess how well the observed data agrees with any given hypothesis, a test statistic $t(x)$, which is a function of the observed data, is constructed. In this thesis we treat the output of a classifying convolutional neural network as the test statistic $t(x)$.

A specific p.d.f. for the test statistic, t, is implied by each of the hypotheses, i.e. $g(t|H_0), g(t|H_1)$, …

While the test statistic can be a multidimensional vector $t = t_1, t_2, \dots, t_m$ (in principle, even the original vector of observed data points $x = x_1, x_2, \dots, x_n$ can be used), constructing a test statistic of lower dimension (where $m < n$) reduces the amount of data being assessed, and should not lose discriminative power if $t(x)$ is well-

constructed. Finding such a test statistic is the major motivation behind Machine Learning (discussed in Section 3.2).

If a scalar function $t(x)$ is used as the test statistic, a p.d.f. $g(t|H_0)$ is given which t will conform to when H_0 is true, similarly t will conform to a different p.d.f. $g(t|H_1)$ when H_1 is true. Figure 19 illustrates how setting a threshold value for the test statistic, i.e. t_{cut}, results in rejection of the null hypothesis when $t > t_{cut}$.

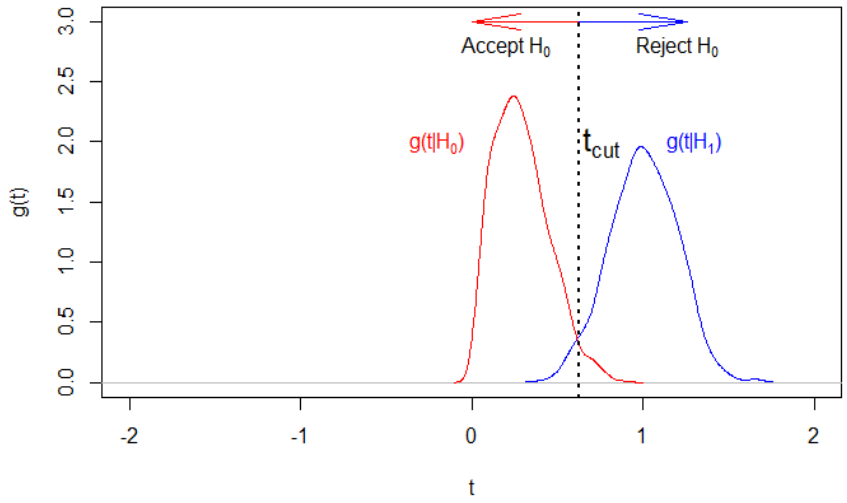


Figure 19: An illustration of rejection or acceptance of the null hypothesis, under the assumed distributions of H_0 and H_1, when t falls in the critical region $t > t_{cut}$

The support for various hypotheses under the observed data distribution is framed in terms of acceptance or rejection of the null hypothesis by defining a critical region for the test statistic, beyond which the null hypothesis is rejected; i.e. when the observed value of t lies within the critical region, we reject H_0. Conversely, when t lies within the complement of the critical region, it is said to be within the acceptance region, which will result in the researcher accepting H_0.

3.1.5 Errors of the First and Second Kind

In general, there is a chance that one of two errors can be made when performing statistical tests:

Type I Error: Rejecting a True H_0

In practice, H_0 would not be rejected when $t < t_{cut}$, but there is a probability of α of rejecting H_0 when H_0 is in fact true (called a false positive).

The significance level (α) defined as such is given by:

$$\alpha = \int_{t_{cut}}^{\infty} g(t|H_0)dt$$

Equation 7

In other words, the critical region for rejection of the null hypothesis is defined by a cut-off point, such that the probability of t being observed there is defined by a α.

In contrast, $1-\alpha$ gives the probability for accepting H_0 when H_0 is actually true (called a true negative, which is also known as the specificity of the test).

In the example shown in Figure 19, a critical region is defined by a value: t_{cut}, which defines the lower decision boundary for rejecting the null hypothesis.

Type II Error: Accepting a false H_0

There is also a probability β of accepting H_0 when H_1 was actually true (a false negative). β is given by:

$$\beta = \int_{-\infty}^{t_{cut}} g(t|H_1)dt$$

Equation 8

$1-\beta$ is called the power of the statistical test to discriminate against H_1.

These concepts are summarised in Table 4:

Table 4

	H_0 is true	**H_1 is true**
H_0 is rejected	**False Positive** $P(FP) = \alpha$	**True Positive** $P(TP) = 1-\beta$
H_0 is not rejected	**True Negative** $P(TN) = 1-\alpha$	**False Negative** $P(FN) = \beta$

3.1.6 Likelihood Ratio Tests & The Neyman-Pearson Lemma

The acceptance of a null hypothesis can be framed slightly differently as: a parameter θ lying within a specified subset Θ_0 of a parameter space Θ of a given statistical model. The alternative hypothesis will then be that θ lies in the complement of Θ_0, i.e. $\Theta\,{}_0^C$.

We can then define the likelihood ratio test (a method to assess how two statistical tests compare in terms of their respective goodness of fit to a set of observations), as follows:

$$\Lambda(x) := \frac{\mathcal{L}(\theta_0|x)}{\mathcal{L}(\theta_1|x)}$$

Where $\mathcal{L}(\theta|x)$ is the likelihood function. In this case H_0 is rejected at a significance level of $\alpha = P(\Lambda(x) \leq t_{cut}|H_0)$.

Given these concepts, the Neyman-Pearson Lemma states that the ikelihood ratio $\Lambda(x)$ as defined above is the most powerful statistical test at significance level α.

3.1.7 Statistical Tests for Particle Selection

In the case of electron-pion particle identification dealt with in this dissertation, we consider the class "electron" as signal and "pion" as background. As such, we define $H_0 = e,\ \ H_1 = \pi$, and by extension, we treat the output of the final hidden unit in the neural network as a test statistic in its own right, lying either within a p.d.f. $g(t|H_0)$ when it is an electron or $g(t|H_1)$ when it is a pion. In order to accept or reject H_0, we define a critical region t_{cut}. When $t \geq t_{cut}$, we classify the particle as an electron.

Based on the probability of each of 6 tracklets being an electron (obtained from each of the 6 detector layers in the TRD), we use a Bayesian approach outlined in the formula below to calculate the probability for the full track (all 6 tracklets combined):

$$P(elec) = \frac{\prod_{j=1}^{6} P_j(elec)}{\sum_{k \in e,\pi} \prod_{j=1}^{6} P_j(k)}$$

Equation 9

Here, $P_j(elec)$ is the probability of a tracklet being an electron obtained from layer j and $P(elec)$ is the combined probability of the full tracklet being an electron. t_{cut} is found in the distribution of $P(elec)$ in the test dataset. This cut-off point can be chosen so as to accept as many electrons as possible, but the price paid for high electron efficiency is a large amount of pion contamination in the electron sample.

When looking at the probability of classifying a specific particle as a given type, we define the selection efficiencies, i.e. the electron efficiency ε_e and pion efficiency ε_π as follows:

$$\varepsilon_e = \int_{-\infty}^{t_{cut}} g(t|e)dt = 1 - \alpha$$

Equation 10

$$\varepsilon_\pi = \int_{-\infty}^{t_{cut}} g(t|\pi)dt = \beta$$

Equation 11

The goal of training neural networks for particle selection is to find a $t(x)$ which is able to maximise ε_e, while minimising ε_π.

Then, in order to compare our results to previous results obtained, we will sacrifice some ε_e. Specifically, we will adjust t_{cut} to the point that it allows us to calculate the obtained ε_π at $\varepsilon_e \approx 90\%$.

3.2 Background: Artificial Intelligence, Machine Learning & Deep Learning

Artificial Intelligence (AI) is a branch of Computer Science concerned with getting computers to perform tasks that mimic those performed by the human mind (such as recognising faces from images, solving complex problems and learning from experience). The field of AI encompasses both hard-coded rule-based programs (known as the knowledge base approach to AI, which has largely remained ineffective), as well as Machine Learning, which is an approach to AI which aims to get computers to perform these tasks without explicitly coding the solutions for them [40].

The success of Machine Learning algorithms is largely determined by the representation of the data fed through them, i.e. a set of pertinent features (x) which can potentially be useful factors of variation that an algorithm can use to determine the desired outcome (y) for each observation, represented in a way that confers useful information to the algorithm (e.g. when determining the time from a clock, giving the angles of the watch hands could potentially be an easier representation to hand to an algorithm than presenting the data in the form of photographs of the clock; this process might involve manual feature engineering cf. representation learning described below).

Often, a large amount of an AI practitioner's time is dedicated to engineering the right feature-set to hand to a simple machine learning algorithm [40], this could involve taks such as feature selection (not all variables are necessary or useful) and data preprocessing (scaling and normalising data, dealing with missing values by exclusion or imputation, sensible handling of outliers, etc.).

Representation learning is a solution to feature generation in which ML is applied, not only to map from a feature set to an output, but also towards automatically learning the most useful representation of the data; usually this representation will encompass identifying the major factors of variation which effectively explain the observed data and discarding those which are not useful to the algorithm [40].

Deep Learning is an approach to representation learning which constructs useful representations based on a combination of simpler representations. In fact, the basic unit of a neural network is the perceptron, which in itself is a very simple function, i.e. $\varphi(\sum_{i=1}^{n} w_i x + b)$ (see Figure 20), but once compiled into a Multi-layer Perceptron, the rich texture of the input data distribution can be very accurately captured, because useful features discovered in the first layers of such a neural network can subsequently be combined in various ways to create additional useful features downstream [40]. In other words, an intitial set of features with a specific representation is transformed through various layers of the neural network in order to generate a new representation of the input data which is useful to hidden layers deeper in the network, and finally to the network as a whole to achieve its task (usually framed broadly as classification or regression).

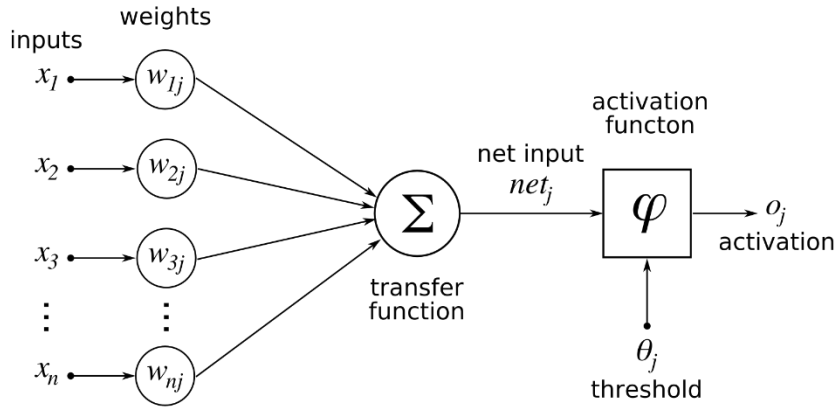


Figure 20 Schematic of a single neuron, with inputs multiplied by weights, a bias term is added to the summation in the transfer function (not shown) and this net input is then passed through a non-linear activation function [41]

3.3 Mathematical Basis: Artificial Neural Networks

At its most basic level, an artificial neural network (ANN) approximates a mapping function f_a, which maps from a set of input features x_i ; $i = \{1,2, \dots, n\}$ to a response, y. Feedforward neural networks have one-way information flow from input features to output, whereas recurrent neural networks have feedback connections [40].

Also called multilayer perceptrons (MLPs), deep feedforward networks are composed of an arbitrary number of nested approximating mapping functions, of the form:

$$f(x_{i,\dots,n}) = f_a^m(f_a^{\cdots}(f_a^2(f_a^1(x_{i,\dots,n}))))$$

Equation 12

The superscript of these functions, $f^{\cdot}$, indicates the layer index of the function in an ANN, with m indicating the depth of such a neural network. It is this concept of chained functions of arbitrary depth from which the term Deep Learning is derived [42].

The set of nested approximation functions outlined above are composed of an input layer f_a^1, an arbitrary number of hidden layers $f_a^{2,\dots,m-1}$ and an output layer f_a^m; with the dimensionality of the outputs of each layer is known as its width, or as the number of neurons in that particular hidden layer [40].

In order to produce subtle derived features from the input feature set, nonlinear transformations typically are applied to the output of each neuron in each layer in the network; each neuron itself is a simple linear function of the form $w^T x + b$, where w^T is a vector of weights of the same length as the set of input features, which are essentially a set of coefficients for each f_a in the chain of functions, and b is a real-valued bias term, which is essentially an intercept term for each f_a [40].

It is easy to see that chaining such a set of linear models without applying nonlinear transformations (denoted as $\phi(f_a(x))$) to what are essentially an arbitrary number of linear regression functions ($y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + c$), one would simply arrive at another linear model [40]. Non-linear transformations applied over $w^T x + b$ allow deep- (and even shallow-) learning models to more accurately model the multidimensional feature space of the data distribution. Various nonlinear transformations (more commonly known as activation functions) exist, of which the Rectified Linear Unit (ReLU), which activates its input as: $\varphi(w^T x + b) = \max(w^T x + b, 0)$ is often the first introduced and easiest to understand (see Figure 21). For a more advanced overview of various activations and their performance, please see [44]. Also note that non-linear activation functions at every layer of a network are not an absolute requirement for implementing one, but their use is recommended, unless there is a specific reason not to.

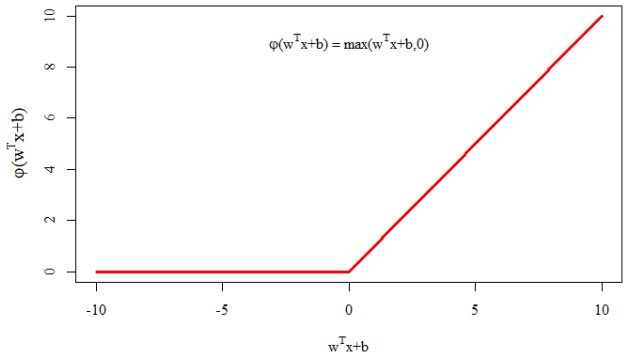


Figure 21: ReLU activation function

Combining the concepts explained above, gives us a representation for a single hidden layer in an ANN as follows(Equation 13, where W is a matrix and $\boldsymbol{x}$ and $\boldsymbol{b}$ are vectors):

$$\boldsymbol{h} = \phi(W^T \boldsymbol{x} + \boldsymbol{b})$$

Equation 13

And, by extension, for a neural network with three hidden layers:

$$\boldsymbol{h}^{(1)} = \phi^{(1)}(W^{(1)T} \boldsymbol{x} + \boldsymbol{b}^{(1)})$$

$$\boldsymbol{h}^{(2)} = \phi^{(2)}(W^{(2)T} \boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)})$$

$$\boldsymbol{h}^{(3)} = \phi^{(3)}(W^{(3)T} \boldsymbol{h}^{(2)} + \boldsymbol{b}^{(3)})$$

Equation 14

For each hidden layer, a matrix of trainable weights is multiplied by a vector of input features, which is either the original features fed to $\boldsymbol{h}_1$, or the weighted outputs of hidden units in the preceding layer, $\boldsymbol{h}_{2,\ldots,n}$. In addition, each hidden layer has an attendant vector of bias terms, and all of these parameters, collectively referred to as θ, need

to be optimized to arrive at a reasonable approximation of a theoretically optimal mapping function $f^*(x) = y$ [40].

Figure 22 shows more graphically how many neurons are combined into hidden layers which are in turn combined to form a fully connected feedforward artificial neural network, as discussed above.

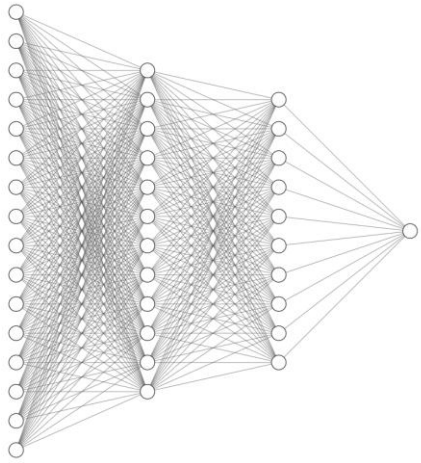


Figure 22: Schematic depiction of a fully-connected feedforward neural network.

3.3.1 Optimization

The essential optimization objective in deep learning is to find the optimal set of parameters θ to minimize an objective (loss) function $J(\theta)$, by utilising the concept of maximum likelihood. [40]

3.3.1.1 Loss Functions

Neural networks are not directly optimized, but they are optimised at the hand of surrogate loss functions, which when minimised, will result in a neural network which is optimised for the task it is set up to accomplish.

A comprehensive overview of loss functions designd for specific use-cases can be found in [41]. The two loss functions that were predominantly used for particle identification in this project are discussed below.

3.3.1.1.1 Binary Cross-entropy

Binary cross-entropy is defined as:

$$\mathrm{BCE}(y, \hat{y}) = -\frac{1}{n}\sum_{i}\left(y^{(i)}.\, log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right).\log\left(1 - \hat{y}^{(i)}\right)\right)$$

Equation 15

Here, $\hat{y}$ is the model's estimate for the probability of an observation of being of a particular class y and i references a specific observation in the training dataset [40]. Figure 23 shows how, as $\hat{y} = p_{model}(y|x)$ approaches the true y, the binary cross entropy loss function approaches 0 (this is only shown for one training

example, i; to compute the overall loss for the entire training sample, individual losses are summed over and averaged as shown in Equation 15).

Note that for multi-class classification the binary cross-entropy loss function can be extended to N classes and is then called categorical cross-entropy. In the case of binary classification, such as particle identification done in this thesis, one can simply one-hot encode the particle ID, i.e. $e = 1, \pi = 0$ and use a sigmoid activation function ($\varphi(x) = \frac{1}{1+e^{-x}}$, which is bounded in the range [0,1]) in a single-neuron output layer in order to use this loss function.

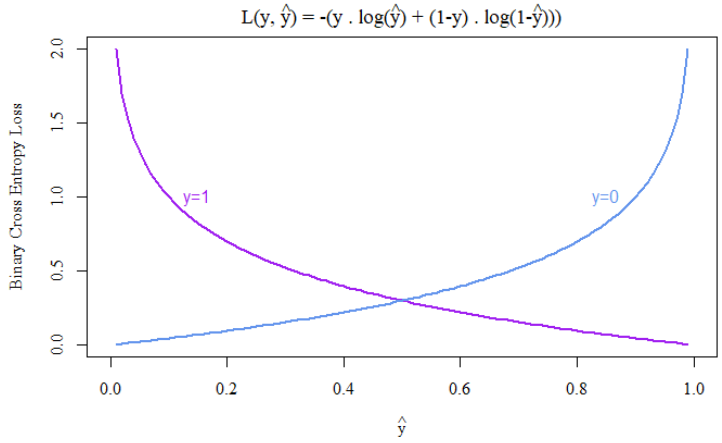


Figure 23: Illustration of the descent towards zero, of the Binary Cross Entropy Loss Function as ŷ, or $p_{model}(y|x)$, approaches the true y.

3.3.1.1.2 Focal Loss

Focal Loss was proposed by [47] as a modification of the binary cross-entropy loss function. Focal loss down-weights the importance of well-classified examples, effectively making training examples that are more difficult to classify contribute more to the overall loss.

This is important, since when there are extreme imbalances in the number of examples of foreground- (in the case of this thesis e) and background- classes (in the case of this thesis π) of the order of ~ 1:1000, a neural network will naturally obtain a lower overall loss when it favours the majority class: i.e. even though some examples might be very badly classified (and will have individually high binary cross-entropy loss), their contribution to the overall loss function will be overwhelmed due to the averaging process that occurs when calculating the loss function.

The way in which focal loss modifies binary cross-entropy is explained below:

First, we define the term p_t for convenience:

$$p_t = \begin{cases} \hat{y} & \text{if } y = 1 \\ 1 - \hat{y} & \text{if } y = 0 \end{cases}$$

This allows us to express binary cross-entropy as follows:

$$BCE(y, \hat{y}) = BCE(p_t) = -\log(p_t)$$

Focal loss imply adds a modulating factor $(1 - p_t)^{\gamma}$, which is parameterised by a focusing parameter γ to the binary cross-entropy loss function, as well as a weighting factor α_t, defined similarly to p_t as:

$$\alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{if } y = 0 \end{cases}$$

$$FL(y, p_t) = \alpha_t (1 - p_t)^{\gamma} . \log(p_t)$$

The authors suggest using $\gamma = 2$, in which case $p_t = 0.9$ would contribute a focal loss result which is 100 × lower than that obtained by binary cross-entropy and a $p_t \approx 0.968$ would contribute a loss which is 1000× lower than binary cross-entropy.

Figure 24 shows how the Focal loss function decreases steeply as classification probability approaches the true class level (again, where y=1). The use of this loss function allows for the training of models without having to resort to down-sampling or up-sampling to account for class imbalances and therefore makes it possible to use a much larger training dataset in these cases.

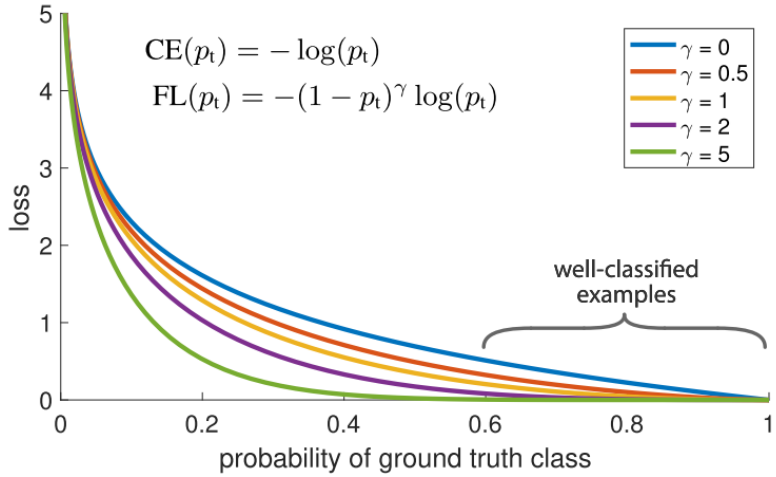


Figure 24: Focal Loss where true class is 1.

Focal loss is not a default loss function in Keras, but it has been implemented as a custom loss function here [i] for Python. The author of this thesis subsequently adapted this for use in Keras with R here[ii], since no equivalent custom focal loss implementation could be found online for R.

3.3.1.2 Minimization of Loss via Backpropagation

3.3.1.2.1 Introduction

The chain rule of calculus is employed via a process called backpropagation (see Section 3.3.1.2.2), to enable the derivative of the loss function J (as described in 3.3.1.1) to be redistributed through the network, based on the partial derivative of each parameter in the set θ with respect to the derivative (∇) of the loss function, where $\hat{y}$ is the output of the neural network at iteration k and y is the desired output [40]:

$$\nabla = \nabla_{\hat{y}} J(\theta) = \nabla_{\hat{y}} L(\hat{y}, y)$$

Equation 16

In practice, the process of training a neural network, f, to give the closest approximation to the desired output, y, is an iterative process, involving passing many observations, each having the same feature set $x_{i,\dots,n}$ through the MLP, assessing the output, $\hat{y}$, according to a loss function, J, and individually adjusting each of the mapping functions $f_a^{j,\dots,m}$ according to the contribution of each of their parameters to the differential of the magnitude of error at the conclusion of each training step k. In other words, a parameter set θ, pertaining to each f_a^j is iteratively adjusted according to $\frac{\partial J_k}{\partial f_a^j}$ [40], until a (hopefully global) minimum is achieved [40]. Note that the gradient of the loss function will be $\nabla = 0$ when either a local or global minimum is reached (shown in Figure 25).

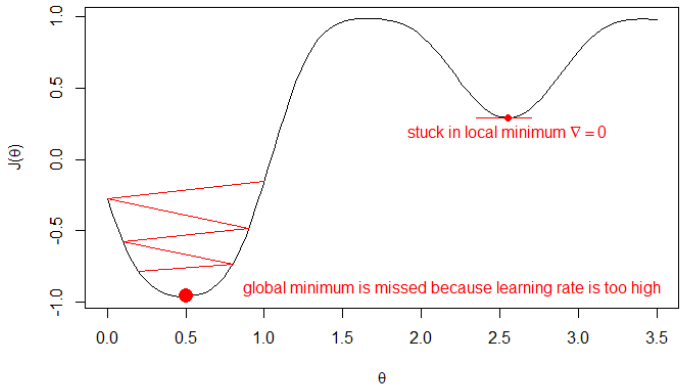


Figure 25: Pitfalls faced during optimization

The optimization process is constrained by a learning rate hyperparameter η, which is usually a small value $\eta \ll 1$ governing the step size of the weight update (see Figure 25 for an illustration of what can happen if the learning rate is too high). Therefore, the exact formula for updating an individual weight-, bias- or other trainable parameter θ_i at iteration (or epoch) , k, is as follows:

$$\theta_i = \theta_i - \eta \times \frac{\partial}{\partial \theta_j} J(\theta)$$

Adaptive learning rates, utilization of the second derivative of the loss function during training and various parameter initialization- and other advanced strategies can be employed to make the training/ optimization process more effective and to prevent the pitfalls shown in Figure 25 and others [40].

3.3.1.2.2 Backpropagation

Algorithm 1: Backpropagation at the conclusion of a single epoch

Given: input $\boldsymbol{x}$, target $\boldsymbol{y}$, a neural networ**k with $\boldsymbol{l}$ layers**′ estimate $\hat{\boldsymbol{y}}$, and the value of the loss function $\boldsymbol{J}(\boldsymbol{\theta}) = \boldsymbol{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$**;** the gradients of the activations $\boldsymbol{a}^{(\boldsymbol{k})}$ in each layer $\boldsymbol{k}$are calculated as follows

Compute gradient of the loss function on the output layer:

$$\nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$$

for: $\boldsymbol{k} = \boldsymbol{l}, \boldsymbol{l} - \mathbf{1}$,…,1 do:

Convert the gradient on the output of each layer to the gradient of the previous layer before the nonlinear activation is applied:

$$\nabla_{a^{(k)}} J = \nabla_{a^{(k)}} J \odot f'(a^{(k)})$$

Compute the gradients on the weight $W^{(k)}$ and bias ($b^{(k)}$) terms:

$$\nabla_{b^{(k)}} J = \nabla_{\hat{y}} J + \nabla_{b^{(k)}}$$

$$\nabla_{W^{(k)}} J = \nabla_{\hat{y}} J . \varphi(a^{(k-1)})^T$$

Propagate the gradients with regards to the previous layer's activation functions:

$$\nabla_{\varphi(a^{(k-1)})} J = {W^{(k)}}^T \nabla_{a^{(k)}} J$$

end for

3.3.1.2.3 Optimization Algorithms

3.3.1.2.3.1 Stochastic Gradient Descent

Many scientific fields make use of stochastic gradient-based optimization: as long as a parameterised scalar objective function is differentiable with regards to its parameters, gradient descent can be used to optimise said parameters to either minimise or maximise the objective function [44].

Stochatic gradient descent (SGD) is an optimization algorithm which approximates the true gradient of the dataset by calculating the gradient for a single training example at a time and using it that as an unbiased estimate of the true gradient. It is from this sampling procedure that the "stochastic" term is added to SGD's name.

Since passing single observations through a neural network can be computationally expensive (and volatile if the learning rate isn't small enough); in practice, subsamples (mini-batches) of data are usually evaluated sequentially (technically this is just an approximation of SGD, which is more accurately referred to as Mini-Batch Gradient Descent (MB-GD), but is still called SGD in most software package implementations). Once the entire dataset has been passed through the neural network once in batches of size m, an epoch of training is said to be concluded. It is good practice to shuffle mini-batches at each epoch to prevent update cycles from occurring. SGD (MB-GD) can be a highly efficient way to optimise parameters for a neural network or other differentiable function. Algorithm 2 shows how MB-GD is used to update a single parameter at each training step, using the mean-squared error as an example loss function.

Algorithm 2: SGD (MB-GD) update of the mean squared error (MSE) loss function

Given: learning rate η ; initial parameter θ

While stopping criteria unmet, do:

1. Sample minibatch $\{x^{(1)}, \dots, x^{(m)}\}$ and corresponding targets $y^{(i)}$
2. Compute gradient estimate according to Backpropagation (Algorithm 1):

$$\widehat{\nabla_\theta} = +\frac{1}{m}\nabla_\theta \sum_i \text{MSE}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$= +\frac{1}{m}.\sum_i \frac{\partial\left(f_\theta(x)^{(i)} - y^{(i)}\right)^2}{\partial\theta}$$

$$= +\frac{1}{m}\sum_i 2\left(y^{(i)} - \hat{y}^{(i)}\right).\frac{\partial\left(f_\theta(x)^{(i)} - y^{(i)}\right)}{\partial\theta}$$

$$= +\frac{2}{m}\sum_i \left(y^{(i)} - f_\theta(x)^{(i)}\right)$$

3. Apply update:

$$\theta \leftarrow \theta - \eta\widehat{\nabla_\theta}$$

End while

3.3.1.2.3.2 Variants of the SGD concept and other Optimization Algorithms

Various optimization algorithms exist that modify the basic concept of SGD, for example by making use of the concept of "momentum", which takes an exponentially decaying moving average of past gradients into account when updating weights during backpropagation to result in accelerated learning.

Momentum

The concept of Momentum in a deep learning context, is inspired by Newtonian laws of motion and represents the negative gradient of the loss function as a force moving parameters in the parameter space. In practice, momentum introduces an additional variable v, which represents the speed at which parameters move through the parameter space and is set to an exponential moving average (EMA) of the negative gradient (here we assume unit mass and therefore v is also the momentum of the parameter, according to $p = mv$, to complete the Physics analogy). An aditional hyperparameter $\alpha \in [0,1)$ determines the rate of exponential decay; updating θ using momentum is done as follows:

$$v \leftarrow \alpha v - \eta \nabla_\theta \left(\frac{1}{m} L\left(f\left(x^{(i)}\right), y^{(i)}\right)\right)$$

$$\theta \leftarrow \theta + v$$

Adam

The Adam optimizer was predominantly used during this project.

Originating as an acronym for “adaptive moments”, the Adam algorithm is generally touted as an optimization strategy robust to various settings of hyperparameters. Adam uses the concept of momentum to estimate the first moment of the gradient and also applies bias corrections to both the first and second order moments of the gradient [48].

3.3.2 Regularization

Regularization strategies are often employed in Deep Learning to reduce test error; by potentially sacrificing accuracy on training set predictions. Effective regularization reduces overfitting of the model to features only present in the training data, and therefore increases accuracy on unseen data [40].

Regularization strategies can be achieved by, for example, constraining parameter values by adding penalty terms to an objective function or by explicitly constraining parameters. Carefully designed regularization processes can improve performance on test data by encoding prior domain knowledge, making an undetermined problem determined, or by simplifying the model so that it generalizes better [40].

While other regularisation strategies such as Gaussian Noise Layers, L1- and L2-regularisation were used for some models in this project, dropout was the predominantly used regularization strategy; since its introduction to a model managed to maintain stability during training and it was found to be easier to control than other regularisation strategies.

3.3.2.1 Dropout

Dropout is a computationally inexpensive regularization method, which results in training the entire ensemble of subnetworks which can be achieved by setting the output of a subset of hidden units to zero, thus approximating model averaging methods, such as explicit ensembles of multiple models [40].

Practically, dropout is achieved by a combination of mini-batch training and binary mask generation during each minibatch training round. The binary mask is of the same dimensions as the input- and hidden- units and each element in the mask is multiplied by its corresponding neuron, effectively pruning the neural network by setting the output of a random subset of neurons to zero [40].

The probability of sampling a 1 at each unit of the mask is a hyperparameter set before training. Each unit in the mask is sampled independently [40].

3.4 Convolutional Neural Networks

3.4.1 The Kernel Concept and Motivation for CNNs

Convolutional Neural Networks (CNNs) are an extension of deep learning models, highly successful in processing data with a grid-like topology, e.g. images. At least one linear mathematical operation, called a convolution, is applied in CNNs, usually in addition to the general matrix multiplication performed in traditional feedforward neural networks [40].

An example of a simple 2D convolution (multiplying a 3×4 matrix by a 2×2 kernel) is shown below (adapted from [40]).

$$\begin{matrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{matrix} * \begin{matrix} w & x \\ y & z \end{matrix}$$

$$=$$

$$\begin{matrix} aw + bx + ey + fz & bw + cx + fy + gz & cw + dx + gy + hz \\ ew + fx + iy + jz & fw + gx + jy + kz & gw + hx + ky + lz \end{matrix}$$

Equation 17

There are three major mechanisms that improve the accuracy of ML algorithms that motivate the implementation of convolutions in a deep learning architecture, namely parameter sharing, equivariant transformations and sparse interactions [40]. These will be discussed below.

Sparse interactions occur in CNNs because of kernels that are smaller than the input matrix, which means that every input unit does not have a connection to every output unit (as is the case in fully connected traditional

ANNs), this sparsity of weights allows for the detection of meaningful small-scale features, such as edges, which are combined downstream (via indirect interactions of neurons in preceding layers) into progressively larger features, such as textures, shapes and actual visual elements. Reducing the number of weights in this manner also leads to an increase in the efficiency of the neural network, since fewer operations are required per layer and fewer weights need to be stored and adjusted [40].

Parameter sharing allow certain parameters to be used by more than one function in a CNN, unlike traditional neural networks, which use each weight in a neural network in just one operation when the network's output is calculated. In a CNN, each element of the kernel is multiplied by every element of the input matrix (where dimension differences do not allow for this, edges may be padded with zero-valued matrix elements to enable it). The weights of the kernel function are learnt and applied uniformly, i.e. they are not relearned at each position of the input matrix, again this has benefits with regards to computational efficiency [40].

Equivariance to translation is a phenomenon which results from parameter sharing and means that the output of a convolutional layer changes in the same way that its input changes, i.e. $f(x)$ is said to be equivariant to a function g if $g\big(f(x)\big) = f(g(x))$. In a convolution operation, the function g translates (shifts) the input matrix in some way, but since the convolution operation is equivariant to the function g, it does not matter at which (x,y) coordinates a feature occurs in the input matrix, since it will still result in the same output after the convolution operation has been applied [40].

3.4.2 Pooling

CNN layers are generally composed of three operations, shown in Figure 26:

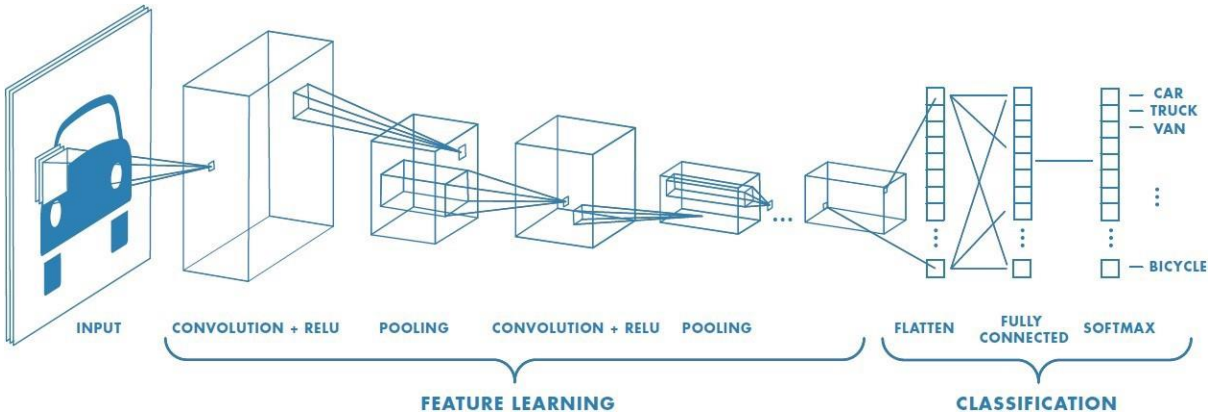


Figure 26: Simplified diagram of a convolutional neural network [48]

1. The appropriate amount of convolution operations, as introduced above, are applied in parallel over the input matrix
2. A non-linear activation function is applied to the output of each convolution operation performed in step one

3. A pooling operation introduces an additional final modification to the layer output

Note that "classical" images (such as photographs) generally consist of 3 channels (Red, Green and Blue); data used in this thesis are not images in the classical sense and therefore only have one channel. Nonetheless, since multiple convolutional kernels are typically applied at each convolutional layer, the datatype for convolutional and max pooling layers is usually a tensor (an array with more than 2 axes). Pooling and convolution operations shouldtherefore be seen as being applied at coordinates (*i, j, k*) of a layer $\mathbb{L}$ which is in fact a three-dimensional tensor, i.e. at $\mathbb{L}_{i,j,k}$, representing row, column and depth of the tensor.

The pooling function in step 3 above, performs a statistical summary over a window of outputs within a defined range, which could be, for example, the L_2-norm, mean or maximum over the series of rectangular ranges thus defined [40].

In this thesis, Max Pooling with a window size of 2×2 was generally employed.

Pooling serves the purpose of insuring invariance to local translation, where the presence of a feature matters more than its location. In some cases, the specific orientation and location of a feature does matter though. Pooling over separate convolutions that are independently parameterized can allow the ANN to learn which translations it should be invariant to which translations it shouldn't be invariant to [40].

For computational efficiency, downsampling of the convolution function can be implemented by skipping over some positions in the kernel, specified by a parameter called stride [40].

Zero-padding is often applied to the input vector in order to prevent it from shrinking by one pixel less than the applied kernel width, i.e. for an input image of width m and kernel width k, the output of the convolution with no zero-padding will be m-k+1, a situation which would enforce smaller networks and smaller subsequent kernels if not accounted for, which in turn would limit the capacity of the network to find useful representations of the data [40].

Convolutions applied with no padding of the input image are known as valid convolutions, where pixels in the output of a convolution are a function of the same amount of pixels in the input, and the kernel can only be applied to positions on the image where the kernel is contained by the image [40].

When just enough zero-padding is applied to the input image to ensure that the output will be of the same dimensions, the convolution is known as a same convolution [40]. Although same convolutions do not limit the size of the network and allow one to build neural networks of arbitrary depth, they still result in pixels close to the edges of the image having less connections to the output image and therefore that their influence on the network as a whole will be reduced [40].

4 Implementation: Machine Learning for Particle Identification

4.1.1 Data Extraction

Please see the following repository[iii] for code used to extract TRD digits (using the AliROOT/AliPhysics installation on the Hep01 cluster in the Physics Department at UCT), from the Worldwide LHC Computing Grid (WLCG).

TRD Analog to Digital (ADC) digits were extracted and filtered for p-Pb runs during 2016, by redirecting the C++ standard out to a text file.

Jobs were submitted onto the WLCG and monitored using the ALICE grid Monitoring site, Monalisa[iv]. Upon completion of each sub-job, the data produced was extracted back onto Hep01 using the aliensh environment.

Data was backed up in a semi-private GitLab repository[v], internally accessible by CERN members.

4.1.2 Data Structure

An example of the raw text data obtained for a single track can be viewed at[vi]. This data structure consists of a header section with meta-information about the track, as well as the raw TRD digits for up to 6 tracklets.

Below are some examples of single tracklets (a tracklet refers to the signal a particle produced in a single layer of the TRD, whereas a full track refers to up to 6 tracklets produced when a particle crosses all 6 layers of the TRD).

In each of the 8 example images in Figure 27, the signal for 17 pads in the TRD layer were added (along the rows of the image), centred around the expected position of the tracklet. The 24 columns in each of the example images represent the charge deposited during a specific time bin within the pad, giving an indication of the time-evolution of the signal.

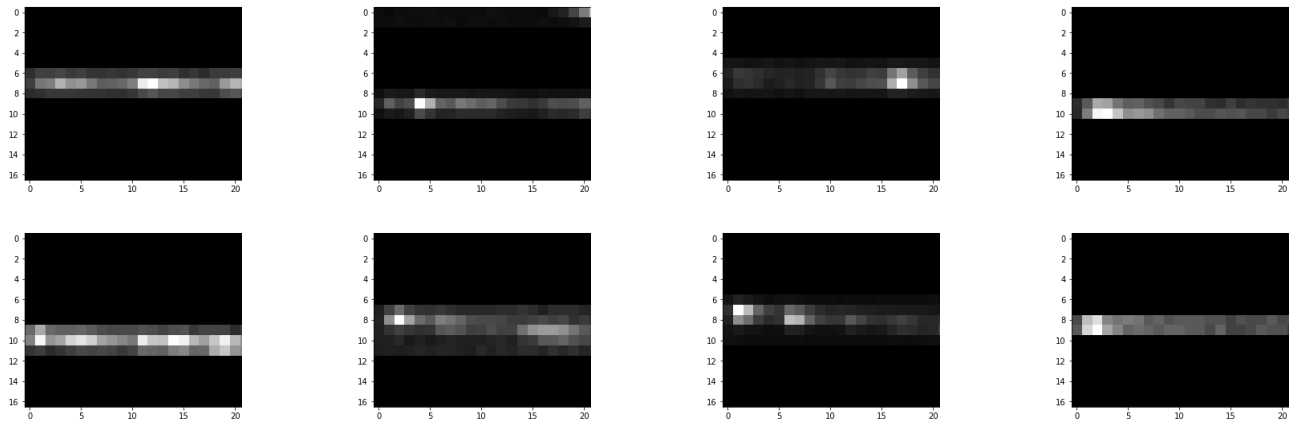


Figure 27: Eight example TRD tracklets, with time-direction indicated along the x-axis and pad-direction indicated along the y-axis for each image shown

4.1.3 Data Exploration

When read into a single list data structure, the full dataset amounts to ~19.7GiB.

While data for 1 565 438 tracks were extracted, only 7 735 493 tracklets of the expected 6 layers × 1 565 438 tracks = 9 392 628 tracklets were obtained. This is mainly the result of detector elements in the TRD being switched off or not working. Missing data of *this* type manifests as either an empty list for that layer in the python dictionary, or as a NULL value.

There is also a second type of missing data: 1 098 636 tracklets returned images, but these images carried no information to assist in particle identification. Every pixel in this type of image was equal to 0.

These images were removed from the particle identification dataset and this resulted in an additional 14.5% of all pion tracklets and 12.6% of all electron tracklets being removed from the dataset used for training and testing.

Technically, excluding this data also affects the true electron- and pion efficiencies reported in this thesis, but this data does not add any additional information, other than the insight that pions result in a slightly higher proportion of empty images compared to electrons.

4.1.3.1 Total Number of Tracklets per Particle ID

Figure 28 illustrates the extreme class imbalance in this dataset; if not appropriately accounted for, such a distorted class distribution can result in unwanted results when training Machine Learning models, such as the Accuracy Paradox, where a model seems to be achieving high accuracy during training but is simply echoing the unbalanced class distribution in its predictions and favouring the dominant class.

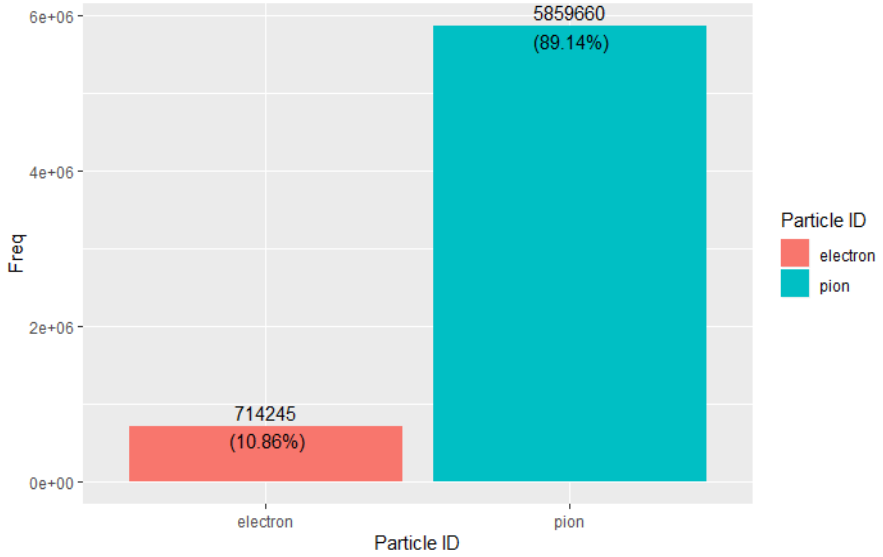


Figure 28: Number of Particles, per Particle ID, across all runs

4.1.3.2 Momentum bin counts: number of tracklets per Particle ID

From **Error! Reference source not found.**, one can see how this class distribution differs for particles in different momentum ranges. Particularly, there is a larger proportion of electrons in lower-momentum bins, i.e. $\boldsymbol{p \leq 2\ \mathrm{GeV}/c}$ and $\boldsymbol{2\ \mathrm{GeV}/c < p \leq 3\ \mathrm{GeV}/c}$. This only partly explains the increased performance in this momentum range (which will be discussed), since electrons are easier to distinguish in this momentum range, according to its characteristic energy loss (Bethe-Bloch) and Transition Radiation.

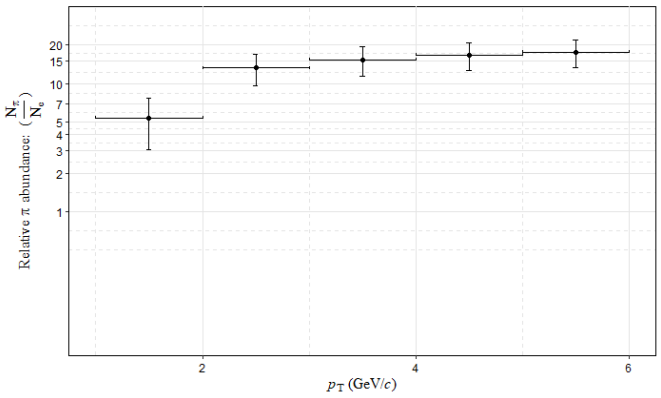


Figure 29

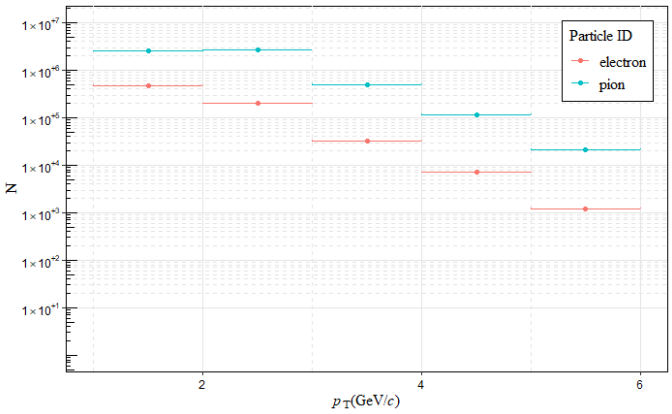


Figure 30

4.1.3.3 Characteristic Energy Loss Curves (Bethe-Bloch)

From Figure 31, the expected increased energy loss of electrons relative to pions, in the low GeV range is apparent. It should be noted that a cut was made on momentum, to keep only tracklets pions in the $2GeV \leq P \leq 6GeV$ range and electrons in the 1.5 GeV/$c \leq p \leq 6$ GeV/c range. Note also that the "ground truth" particle IDs used in this thesis were estimated from V_0 decays, but the long tail towards low $^{dE}/_{dx}$ for the electron sample (shown in Figure 31) indicates that there might be some pions that have been incorrectly identified as electrons using this method. This contamination places an additional limit on obtainable ε_π at $\varepsilon_e \approx 90\%$; and on the reliability of the ε_π and ε_e values reported in this thesis.

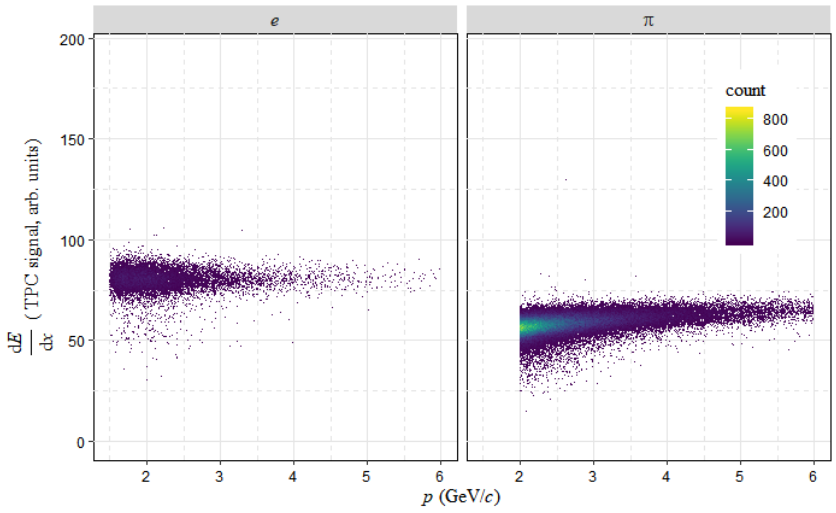


Figure 31: Energy Loss per Unit Path Length as a function of Momentum, for Electrons and Pions, as measured by the ALICE TPC

4.1.3.4 Average Pulse Height

Figure 32 shows the average pulse height as a function of time, for electrons vs pions, across the entire momentum range; the characteristic Transition Radiation (TR) signal can be seen for electrons in the later time bins of the plot. The average pulse height for electrons is also higher than that for pions, across all time bins, but there are significant fluctuations around this average (as can be seen in **Error! Reference source not found.** and **Error! Reference source not found.**).

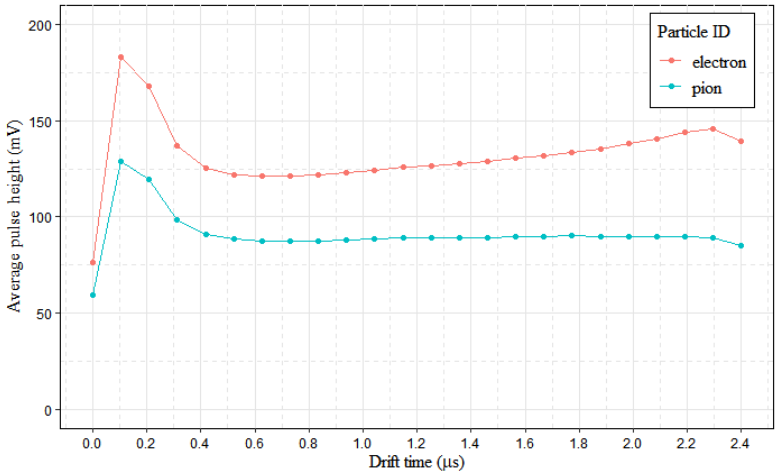


Figure 32: Time Evolution of the Average Pulse Height Signal, per Particle ID (for tracklets from the entire momentum range)

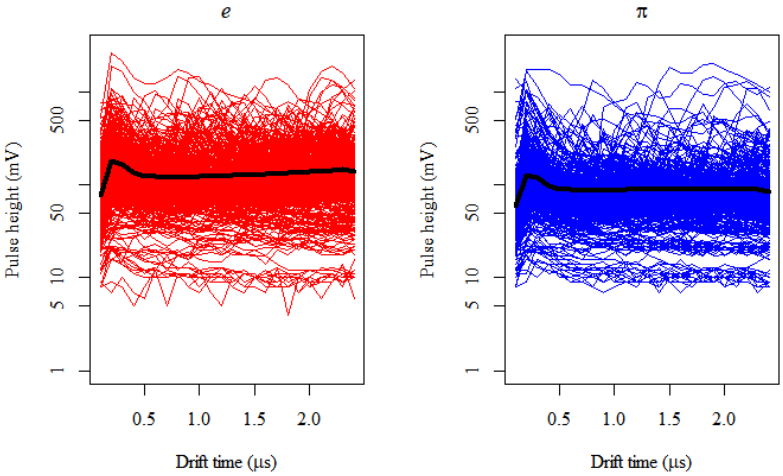


Figure 33

4.2 Particle Identification Results: Previous Theses

The following theses also applied Artificial Neural Networks towards e vs π discrimination, based on TRD data. These results are included here to give a point of reference to the results obtained in this thesis. It should be noted that the capacity of the neural networks developed in these previous theses are extremely limited compard to the nerual networks developed in this thesis, but the obtained results are much better. This is mostly explicable at the hand of the fact that, unlike the data used in previous theses reported here, data used in this thesis was not properly calibrated (as explained in section 0).

4.2.1 Martin Kroesen (Masters thesis, 2017) [35]:

Input data: 7 or 8 features, consisting of: charge depostion, obtained by splitting into and summing over 7 timebin slices (in all cases), as well as the particle's momentum (in some cases)

Sample size: 20 000 electron and 20 000 pion tracks

Implementation: Various single hidden-layer neural networks, all with 35 nodes in the hidden layer; some neural networks were trained across the entire momentum spectrum, while others were trained by splitting particles into either four or eleven momentum bins. A learning rate of $\eta = 0.1$ was used and networks were trained for 500 epochs.

Results: Pion efficiency results as a function of momentum are summarised in Figure 34.

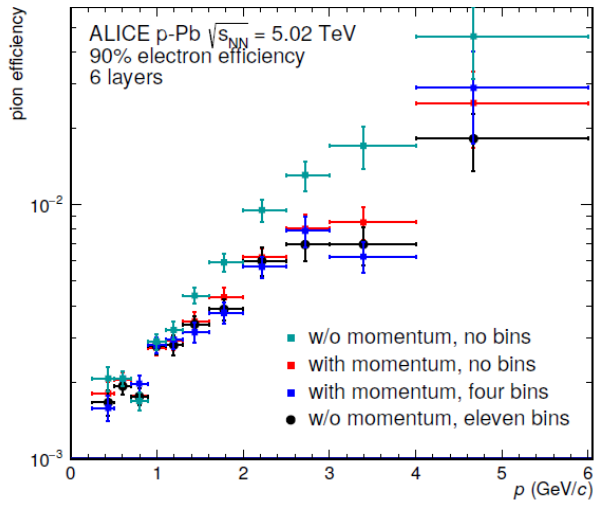


Figure 34: Pion efficiency results from [35]

4.2.2 Alexander Wilk (PhD thesis, 2010) [34]:

Input data: charge depostion, obtained by splitting into and summing over 8 timebin slices

Implementation: various neural networks with carying numbers of nodes in the input layers and two hidden layers with 15 and 7 nodes, respectively. A learning rate of $\eta = 0.001$ was used and networks were trained for 10 000 epochs.

Results: Pion efficiency results on ALICE Testbeam data from 2002 (at $p = 2\text{GeV}/c$) are shown as a function of number of neurons in the input layer in Figure 35.

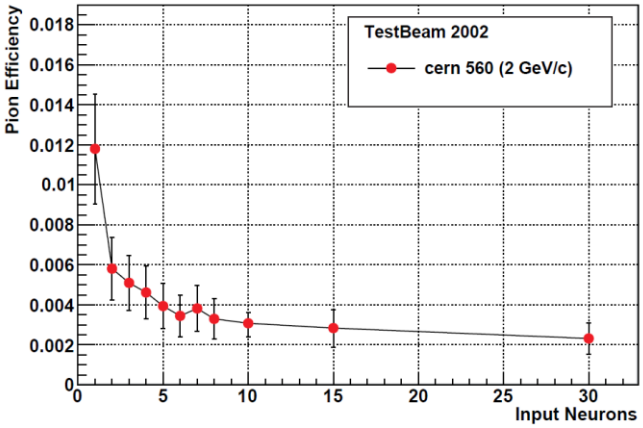


Figure 35: Pion efficiency results from [34]

4.2.3 Linus Feldkamp (PhD thesis, 2018) [50]:

Input data: charge depostion, obtained by splitting into and summing over 7 timebin slices

Implementation: Neural network with a 7 node input layer and 3 hidden layers with 10, 8 and 6 neurons respectively; various convolutional neural networks inspired by the "Inception-v4" network

Results: Results are not explicitly reported (particle identification was not the main aim of this thesis), but it is mentioned that obtained results are on par with [34].

4.3 Particle Identification Implementation: This thesis

Various Machine Learning strategies were employed in the task of particle identification. Deep learning models were built in Keras, with a Tensorflow backend, utilising the SLURM-managed UCT HPC Cluster extensively to train multiple models simultaneously.

Non-Deep Learning Methods (Gradient Boosting Machines and Random Forests) were implemented locally, using H2O.ai.

Please see the following repository[vii] for code used to build and train the various machine learning classifiers discussed.

4.4 Particle Identification: Results

Three sets of results will be presented:

1. The most successful particle identification strategy on uncalibrated raw digits will be discussed in detail in Section 4.4.1.
2. A summary of other models that were built and trained for particle identification will be presented, at the hand of **Error! Reference source not found.** (for all 2D Convolutional Neural Networks built),
3. and as a text summary in Section 4.4.2 (for *all* models built).

4.4.1 Most successful approach

The most successful pion rejection and electron acceptance results were obtained by incrementally training a 2D Convolutional Neural Network, using Focal Loss as the loss function to be optimised and Adam as the optimizer at a learning rate of $\eta = 0.00001$.

- The full dataset was used during this stage.
- All tracklets with no signal, i.e. images where all the pixel values were zero, were removed.
- Data was not normalised or standardised.
- Data was not down-sampled or up-sampled to account for class imbalances.
- Data was split into the following momentum bins:
  - $p \leq 2$ GeV/c, 2 GeV/$c < p \leq 3$ GeV/c and 3 GeV/$c < p \leq 4$ GeV/c
  - Results in the 4 GeV/$c < p \leq 5$ GeV/c and 5 GeV/$c < p \leq 6$ GeV/c were much worse and are not included here
- A Convolutional Neural Network (architecture shown in) was trained incrementally, per momentum bin, by saving the weights-configuration of the model after training on the previous momentum bin
- These results are summarised in Figure 36.

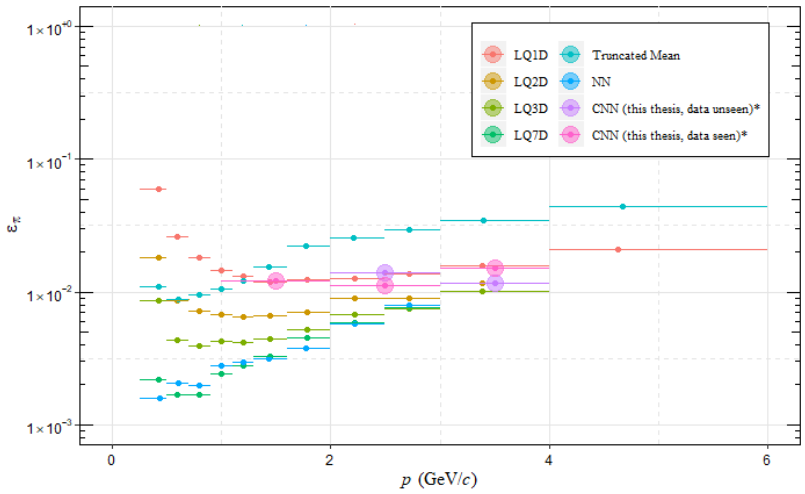


Figure 36: Summary of incrementally trained 2D Convolutional Neural Network: Red marks indicate pion efficiency at 90% electron efficiency, when the incrementally trained model was evaluated on data from the momentum-bin the model was last trained on. Orange marks indicate the results when testing the model on data from the next momentum bin (before training on data in that bin). i.e. after training the model on tracklets in the $p \leq 2\mathrm{GeV}/c$ range, the model was first evaluated on this range (first red mark), then tested on the $2\mathrm{GeV}/c < p \leq 3\mathrm{GeV}/c$ range (first orange mark), then trained and evaluated further.

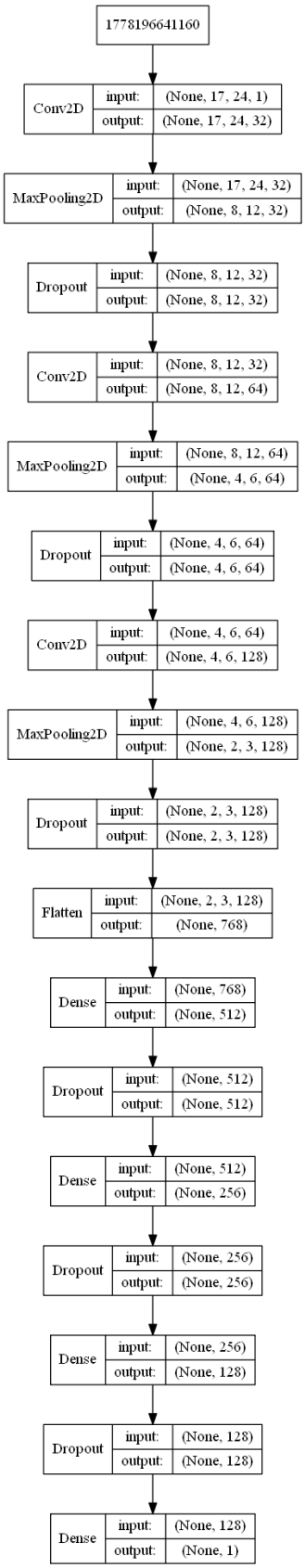

1778196641160
Conv2D
input: (None, 17, 24, 1)
output: (None, 17, 24, 32)
MaxPooling2D
input: (None, 17, 24, 32)
output: (None, 8, 12, 32)
Dropout
input: (None, 8, 12, 32)
output: (None, 8, 12, 32)
Conv2D
input: (None, 8, 12, 32)
output: (None, 8, 12, 64)
MaxPooling2D
input: (None, 8, 12, 64)
output: (None, 4, 6, 64)
Dropout
input: (None, 4, 6, 64)
output: (None, 4, 6, 64)
Conv2D
input: (None, 4, 6, 64)
output: (None, 4, 6, 128)
MaxPooling2D
input: (None, 4, 6, 128)
output: (None, 2, 3, 128)
Dropout
input: (None, 2, 3, 128)
output: (None, 2, 3, 128)
Flatten
input: (None, 2, 3, 128)
output: (None, 768)
Dense
input: (None, 768)
output: (None, 512)
Dropout
input: (None, 512)
output: (None, 512)
Dense
input: (None, 512)
output: (None, 256)
Dropout
input: (None, 256)
output: (None, 256)
Dense
input: (None, 256)
output: (None, 128)
Dropout
input: (None, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 1)

Figure 37: Most successful particle identification neural network, incrementally trained on increasing momentum ranges, using Focal Loss as the objective function to be optimized, data used to train this model was not down-sampled or up-sampled.

4.4.2 Summary of Other Results

Please note that models shown in Section 4.4.2 were trained on down-sampled data, incorporating all clean tracks (i.e. 6 tracklets obtained) for electrons and an equal number of pions. Data used for training at this stage was normalised as follows:

$$x = {}^{x}/_{\max(x)}$$

Equation 18

4.4.2.1 2D Convolutional Neural Networks

$\varepsilon_\pi = 2.2\%$ at electron efficiency $\varepsilon_e = 90\%$

Error! Reference source not found. compares the entire gamut of 2D Convolutional Neural Networks in terms of number of layers (total and convolutional), number of epochs trained, learning rate and optimiser used. Learning rate seems to be the most important distinguishing element in achieving low pion efficiency (some of the best-performing models used a learning rate of $\eta = 10^{-5}/\eta = 10^{-6}$, a very high learning rate ($\eta = 0.01$) results in poorly performing models, whereas a very low learning rate ($\eta = 10^{-7}$) does not converge within a feasible number of epochs.) Using more than one convolutional layer is also a seemingly more successful strategy than using just one layer, but no outright statements about architecture can be made. Models that were trained with low learning rates were all optimised using Adam and therefore no outright conclusions can be made about the best optimization algorithm, but common practice generally suggests that Adam is the more robust algorithm to use.

4.4.2.2 1D Convolutional Neural Networks

$\varepsilon_\pi = 6.55\%$ at electron efficiency $\varepsilon_e = 90\%$

4.4.2.3 Fully Connected Feedforward Neural Networks

$\varepsilon_\pi = 14.86\%$ at electron efficiency $\varepsilon_e = 89.99\%$

4.4.2.4 LSTM Neural Networks

$\varepsilon_\pi = 5.3\%$ at electron efficiency $\varepsilon_e = 90\%$

4.4.2.5 Non-Deep Learning (Tree Based) Models

4.4.2.5.1 Random Forests

$\varepsilon_\pi = 5.8\%$ at electron efficiency $\varepsilon_e = 90\%$

4.4.2.5.2 Gradient Boosting Machines

$\varepsilon_\pi = 6.59\%$ at electron efficiency $\varepsilon_e = 89.99\%$

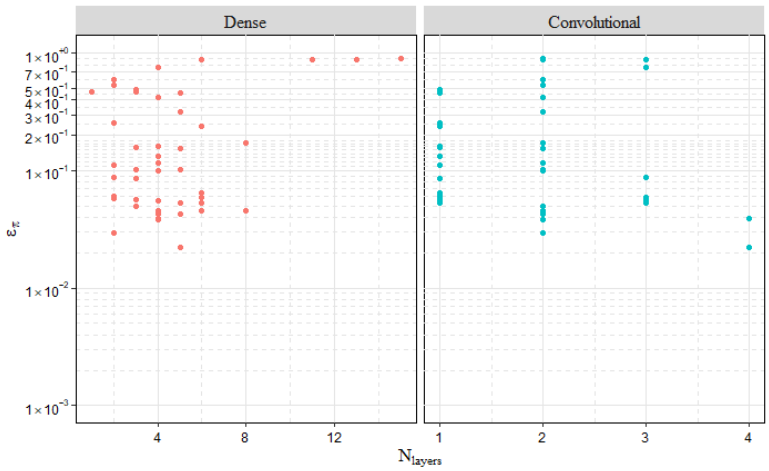


Figure 38

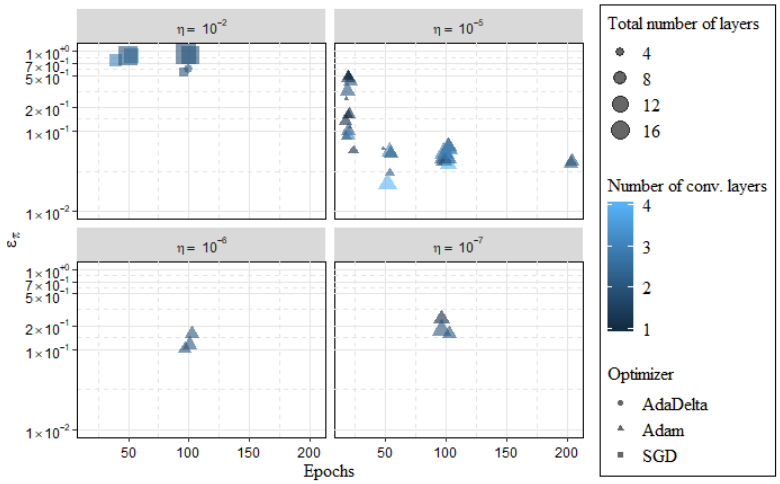


Figure 39

4.5 Chapter Conclusions

While a considerable amount of time was spent on finding an optimal architecture to solve the problem of particle identification, it is interesting to note that a wide variety of neural network architectures (1D CNNs, 2D CNNs, neural networks making use of LSTM cells) and other algorithms (Gradient Boosting Machines and Random Forests) all arrived at less than 7% pion efficiency at 90% electron efficiency on uncalibrated data.

Additionally, what is more important in achieving high pion efficiency is using properly calibrated input data. Since the ALICE TRD is an extremely sensitive detector with a variety of factors of variation that can influence how signal manifests. Chamber gain, pad-by-pad variations, environmental- and other factors all influence how the recorded signal manifests at a specific point in time. In this project, the uncalibrated raw signal data was used. Each signal was effectively treated as if it originated from the same measurement mechanism: the decreased performance compared to previous work done in this area is thus explained.

5 THEORY: HIGH ENERGY PHYSICS DETECTOR SIMULATIONS

5.1 Introduction

This chapter will cover various methods used for the simulation of TRD digits data in this project. The traditional Monte Carlo-based simulation software used for High Energy Physics simulations (Geant4), as well as three types of latent variable models will be introduced theoretically, following which an assessment of the performance of each method will be shown, according to the methodology described in Section 6.1.1.

5.2 Monte Carlo Simulations: Geant4

5.2.1 Background

As a general toolkit to simulate the passage of particles through matter, Geant4 is used in a wide array of applications and fields, from space engineering, to medical-, particle- and nuclear physics. Geant4 provides functionality for geometry, tracking, hits and physics models, over a wide range of energies, particles, materials and elements [49].

Geant4 was designed as the detector simulation component of a typical physics simulation setup, which generally contains the following components: an event generator, detector simulation, reconstruction and analysis. Geant4 is usually tied to an event generator such at Pythia or HIJING, with ROOT used for Reconstruction and Analysis. As such, Geant4 has well-defined interfaces to the other components in the simulation set-up [50]. Its physics implementation is transparent to investigation and validation and can be customised and extended [49].

Simulating the passage of particles through matter involves the following key elements:

- Geometry and materials
- Particle interactions in matter
- Management of tracking
- Digitisation and hit management
- Management of tracks and events

- Visualisation and a user interface

  Each of these elements is implemented in a class category, with a well-defined interface [49]. Figure 40 shows how these categories are related, with lines indicating a "using" relationship (category with open circle uses the adjoining category). Full discussion of Geant4 lies outside the scope of this thesis, but it is worth recognising the complexity of its implementation.

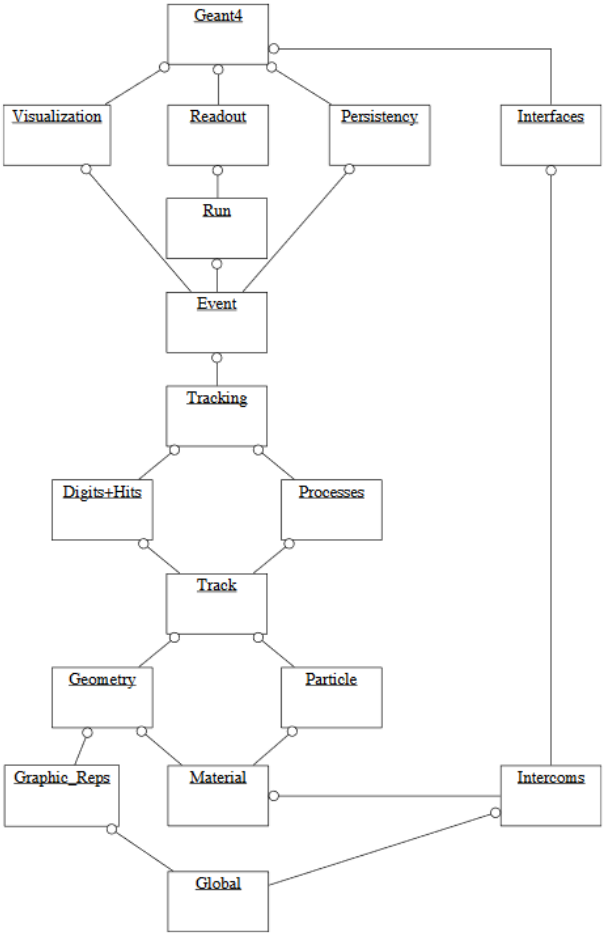


Figure 40: Top level category diagram of the Geant4 toolkit [49]

5.3 Deep Generative Models

Generative models are concerned with modelling potentially high-dimensional distributions. Dependencies between various random variables in the multidimensional distribution can also be captured during this modelling process [51].

Generative models are concerned with generating data that is similar to seen data, but not exactly the same, i.e. our training examples X are distributed according to some unknown distribution $P(\chi)$ and we want to model a

distribution P which is as similar as possible to $P(\chi)$ and therefore allows us to generate new examples X by sampling from P [51].

Neural networks can be utilised as function approximators towards constructing a modelled distribution P as outlined above [51].

5.3.1 Background: Latent Variable Models

When there are complex dependencies between the dimensions of the data, generative models become very hard to train. Latent variables are samples drawn from specific latent distributions constructed during training, before the generative process commences, i.e. the model first chooses what it is going to simulate before it starts simulating [51].

In order to deduce that a generative model is representative, one needs to find that for each datapoint X in χ, there are one or more latent variable settings which result in the model generating something sufficiently similar to X [51].

A vector of latent variables z, are sampled from a high dimensional latent space Z, according to a probability density function (p.d.f.): $P(z)$ defined over Z. A group of deterministic functions $f(z;\theta)$ are parameterized by a vector θ in some space Θ, with $f: Z \times \Theta \rightarrow \chi$. While f is deterministic, z is randomly sampled and θ is fixed, which makes $f(z;\theta)$ a random variable in the space χ. θ needs to be optimized so that sampling z from $P(z)$ will result in a high probability of $f(z;\theta)$ outputting data similar to the training data X [51].

More formally, we want to maximize the probability of each X, according to:

$$P(X) = \int P(X|z;\theta\,)P(z)dz$$

Equation 19

$f(z;\theta)$ has been changed to a distribution $P(X|z;\theta\,)$ in the expression above, in order to show explicitly that X depends on z. Maximum Likelihood underpins the notion that if X is likely to be reproduced, generated examples that are highly similar to X are also likely to be produced, and dissimilar examples are unlikely [51].

Generative models often model the output distribution as a Gaussian, $P(X|z;\theta\,) = N(X|f(z;\theta), \sigma^2 * I)$, i.e. the distribution has mean $f(z;\theta)$ and covariance equal to some scalar σ multiplied by the identity matrix I, with σ being a tuneable hyperparameter [51].

A generative model will in general not produce examples identical to any X, especially not during early training, but under the Gaussian assumption, $P(X)$ can be increased via gradient descent by making $f(z;\theta)$ approach X given some z [51].

5.3.2 Variational Autoencoders

Variational Autoencoders (VAEs) aim to maximize $P(X) = \int P(X|z;\theta)P(z)dz$ by defining latent variables z and integrating over z. Choosing the latent variables z are not trivial, since z is not defined by labelled attributes of the example that needs to be generated, but by other latent features specific to the example [51]. Generally, a researcher would not explicitly specify what the dimensions of z specify, nor how the dimensions of z depend on one another [51].

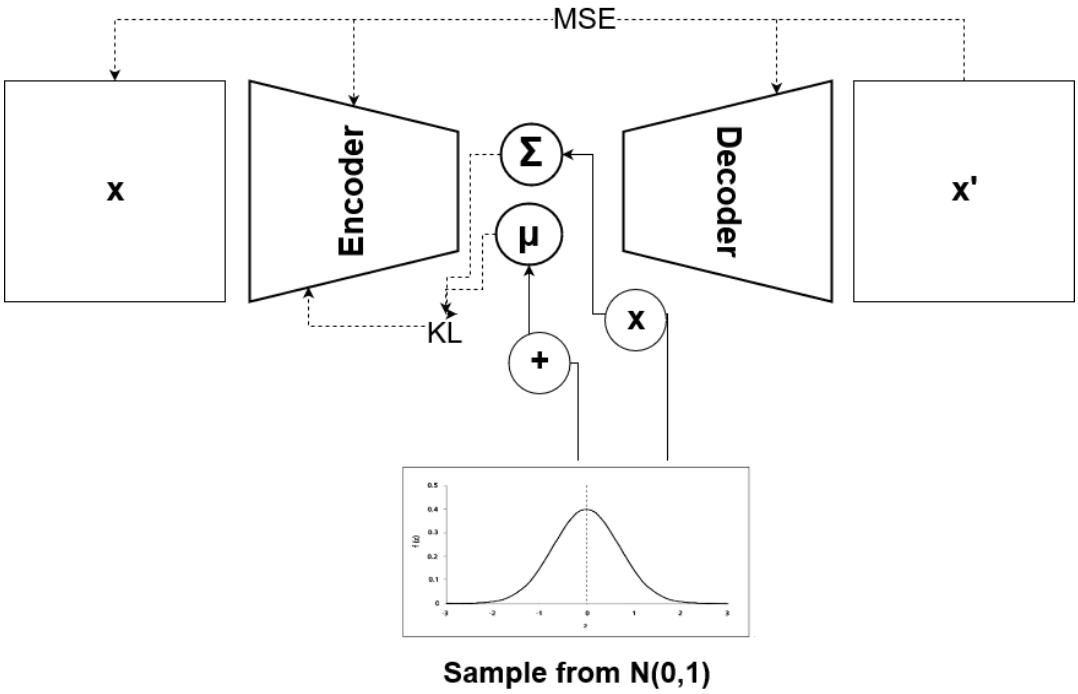


Figure 41: Simplified diagram of a Variational Autoencoder

In VAEs, z is drawn from a distribution $N(0, I)$, where I is the identity matrix; since any distribution in d dimensions can be generated by sampling from d normally distributed variables and mapping them through a function with high enough capacity to generate X. When $f(z;\theta)$ is a set of neural networks then the initial (encoding) network will be involved in generating z while the later (decoding) network will be concerned with mapping z to X. $P(X)$ will be maximized by finding a computable formula for it, taking its gradient at each epoch and optimizing it using stochastic gradient descent [51].

$P(X)$ can be computed approximately by sampling z values repeatedly $z = \{z_1, z_2, \dots, z_n\}$ and computing $P(X) \approx \frac{1}{n}\sum_i P(X|z_i)$. In high dimensional spaces, n might have to be very large before $P(X)$ can be accurately approximated [51].

For most z, $P(X|z)$ will be close to zero, but in order for the VAE to be useful, we need to sample z values that are likely to have resulted in X and sample only from that subset, a new function $Q(z|X)$ is needed to take an existing

X value and calculate a distribution of z values that could have realistically resulted in X being generated; this narrows the universe of z values down from the larger universe of all $z's$ likely under the prior $P(z)$ [51].

How $E_{Z\sim Q}P(X|z)$ and $P(X)$ are related is one of the basic tenets upon which variational Bayesian methods are built. The Kullback-Leibler divergence ($\mathcal{D}$) between $P(z|X)$ and $Q(z)$ for an arbitrary Q which does not necessarily have to depend on X, is given by:

$$\mathcal{D}[Q(z)||P(z|X)] = E_{Z\sim Q}[log\ Q(z) - log\ P(z|X)]$$

Equation 20

$P(X)$ and $P(X|z)$ can be added to this equation by applying Bayes rule:

$$\mathcal{D}[Q(z)||P(z|X)] = E_{Z\sim Q}[log\ Q(z) - log\ P(z|X) - log\ P(z)] + log\ P(X)$$

Equation 21

Since $log\ P(X)$ does not depend on z, it appears outside the expectation. Rearrangement of this formula, negation and contraction of part of $E_{z\sim Q}$ into a KL-divergence term gives:

$$log\ P(X) - \mathcal{D}[Q(z)||P(z|X)] = E_{Z\sim Q}[log\ P(X|z)] - \mathcal{D}[Q(z)||P(z)]$$

Equation 22

In the above equation, X is fixed and Q can be any distribution, regardless of whether it accurately maps X to $z's$ that could have produced X, but since the goal is to accurately infer $P(X)$, a Q needs to be found which *does* depend on X and which also keeps $\mathcal{D}[Q(z)||P(z|X)]$ as small as possible:

$$log\ P(X) - \mathcal{D}[Q(z|X)||P(z|X)] = E_{Z\sim Q}[log\ P(X|z)] - \mathcal{D}[Q(z|X)||P(z)]$$

Equation 23

Equation 23 is the central formula of the VAE, the left hand side is what needs to be maximized: $P(X)$, penalized by $-D[Q(z|X)||P(z|X)]$ (which will be minimized if Q is a high capacity distribution which produces z values that are likely to reproduce X), the right hand side is differentiable and can therefore be optimized using gradient descent.

When looking at the above equation, the right hand side takes the form of an autoencoder, where Q encodes X into latent variables z and P decodes these latent variables to reconstruct X.

On the left side of the equation, $log\ P(X)$ is being maximized while $\mathcal{D}[Q(z|X)||P(z|X)]$ is being minimized. While $P(z|X)$ is not analytically solvable and simply describes z values likely to reproduce X, the second term in the KL-divergence on the left is forcing $Q(z|X)$ to be as similar as possible to $P(z|X)$, and under a model with sufficient capacity $Q(z|X)$ should be able to be exactly the same as $P(z|X)$, which will result in $\mathcal{D}$ being zero. This will allow for the direct minimization of $log\ P(X)$. In addition, $P(z|X)$ is no longer intractable in this case, since $Q(z|X)$ can be used to solve for it.

In order to minimize the right hand side of the above equation via gradient descent, $Q(z|X)$ will usually take the form:

$$Q(z|X) = N(z|\mu(X;\vartheta), \Sigma(X;\vartheta))$$

Equation 24

Where μ and Σ are deterministic functions with learnt parameters ϑ; in practice μ and Σ are learnt via neural networks and Σ is constrained to a diagonal matrix format. $\mathcal{D}[Q(z|X)||P(z)]$ therefore becomes a KL-divergence between two multivariate Gaussians, computed in closed form as:

$$\mathcal{D}[N(\mu_0, \Sigma_0)||N(\mu_1, \Sigma_1)] = \frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^{\intercal}\Sigma^{-1}(\mu_1 - \mu_0) - k + log(\frac{det\Sigma_1}{det\Sigma_0})$$

Equation 25

With k indicating the number of dimensions of the distribution; this can be simplified to become:

$$\mathcal{D}[N(\mu(X), \Sigma(X))||N(0, I)] = \frac{1}{2}(tr\big(\Sigma(X)\big) + \big(\mu(X)\big)^{\intercal}\big(\mu(X)\big) - k - log\ det\ (\Sigma(X)))$$

Equation 26

The other term on the right hand side of the equation, $E_{Z\sim Q}[log\ P(X|z)]$, can be estimated by taking a sample from z and calculating $P(X|z)$ for that single sample to approximate $E_{Z\sim Q}[log\ P(X|z)]$.

Since stochastic gradient descent is performed in practice over different X values from the dataset D, we want to perform gradient descent on the following formula:

$$E_{X\sim D}[log\ P(X) - \mathcal{D}\big[Q(z|X)\big||P(z|X)\big]] = E_{X\sim D}[E_{Z\sim Q}[log\ P(X|z)] - \mathcal{D}[Q(z|X)||P(z)]]$$

Equation 27

By sampling a single value of X and a single value of z, we can compute the gradient of $log\ P(X|z) - \mathcal{D}[Q(z|X)||P(z)]$, which when averaged over multiple samples, converges to the full equation to be optimized.

The issue here is that $E_{Z\sim Q}[log\ P(X|z)]$ does not only depend on the parameters of P, but also those of Q, but this is not accounted for in the above equation. For VAEs to work properly, Q needs to be driven to produce $z's$ from X that are likely to be reliably decoded by P.

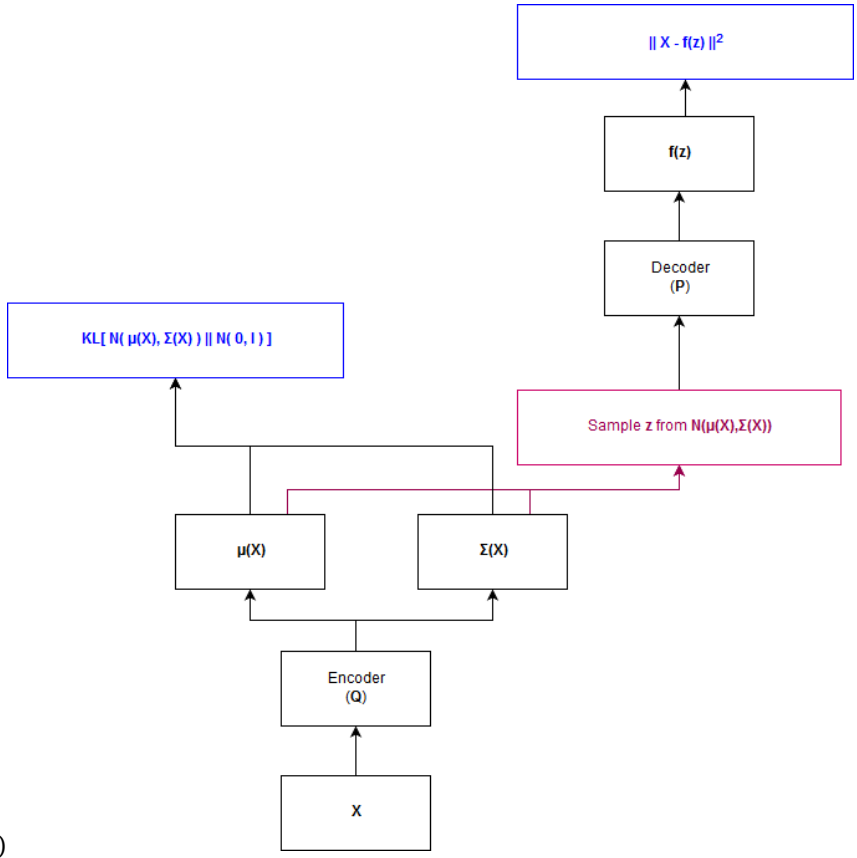
Figure 42 (a) illustrates how this proxy formula can be used by averaging over multiple samples to get to the expected outcome, but since there is a sampling procedure embedded within the neural network, gradient descent cannot be performed on it.

Figure 42 (b), on the other hand, shows how a "reparameterization trick" removes the sampling procedure from the neural network proper and treats it as an input layer. Since we have $\mu(X)$ and $\Sigma(X)$, we can sample ϵ from $N(0, I)$ and compute z from ϵ as follows: $z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon$.

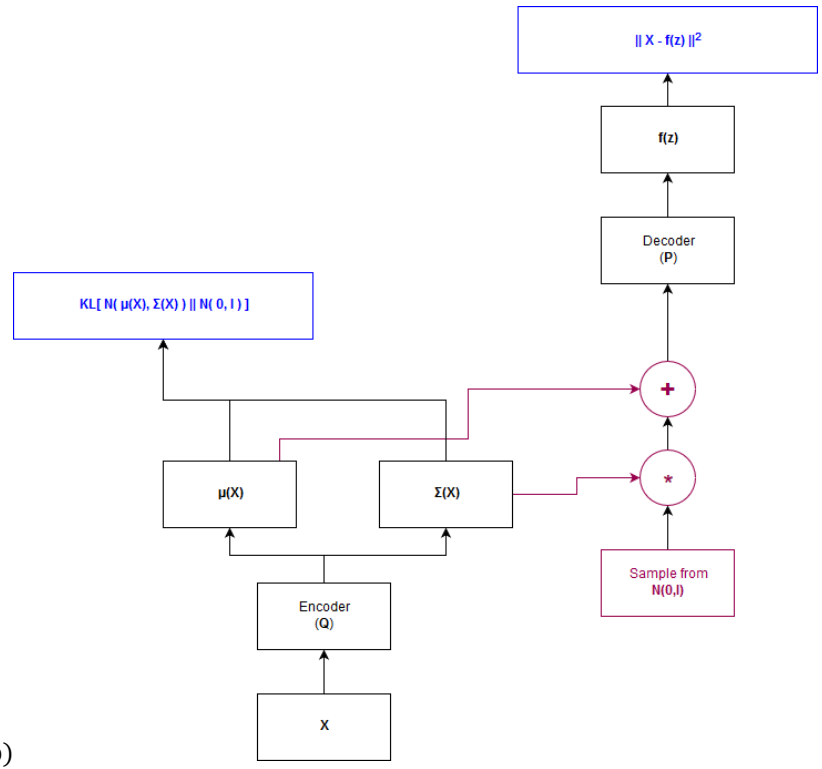
As a result, the gradient of the following equation will actually be taken:

$$E_{X\sim D}\left[E_{\epsilon\sim N(0,I)}\left[log\,P\left(X\middle|z=\mu(X)+\Sigma^{\frac{1}{2}}(X)*\epsilon\right)\right]-\mathcal{D}[Q(z|X)||P(z)]\right]$$

Equation 28



(a)



(b)

Figure 42: Training-time VAE

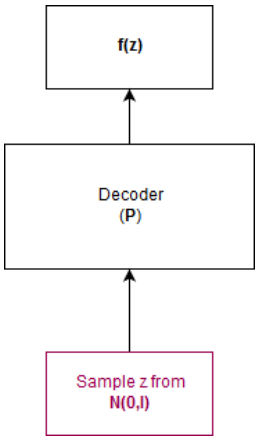


Figure 43: Testing time VAE

Once the model is ready to be tested, values from $z \sim N(0, I)$ are sampled and given as input to the decoder; the encoder, along with the attendant reparameterization trick used during training are no longer needed.

5.3.3 Generative Adversarial Networks

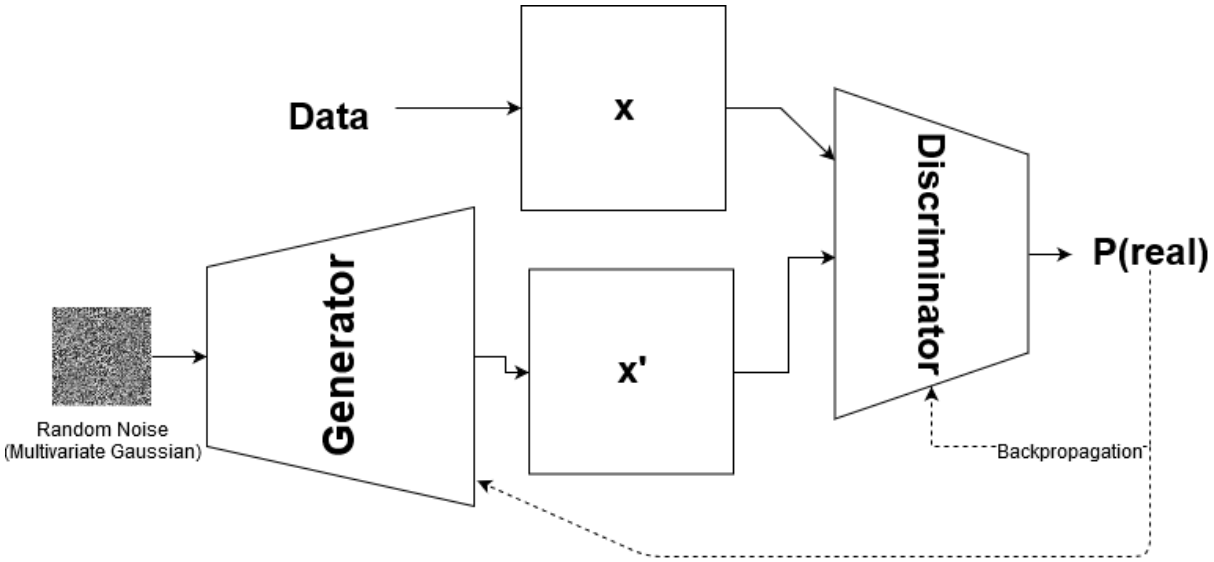


Figure 44: Simplified Diagram of a Generative Adversarial Network

Generative Adversarial Networks (GANs) are a deep learning framework which pits two neural networks against each other in an adversarial mini-max game: the generative model G is trained to the point where it accurately captures the distribution of the training data, and the discriminative network D takes the output of G and estimates the probability of whether G's output originated from the actual data distribution or from a model distribution [52].

The mini-max game can be expressed mathematically as:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[log\, D(x)] + E_{z\sim p_z(z)}[log\left(1 - D\big(G(z)\big)\right)]$$

Equation 29

Essentially, the objective is to maximize the probability of D assigning the correct label to samples from G, i.e. is a given observation from the "data"- or "model" distribution, while training G to minimize $log\left(1 - D\big(G(z)\big)\right)$, i.e. we want G to produce samples that are hard to discriminate from samples from the true data distribution.

This is done by sampling from a random noise vector z, with a defined prior $p_z(z)$ and learning a transformation from the noise vector to a distribution which is highly similar (preferably identical) to the true data distribution; in practice, this transforming function is the generative network $G(z, \theta_g)$, with θ_g being the parameters of a deep neural network which maps z to data space.

In practice, the training algorithm will alternately optimize D for k steps and G for a single step, which allows D to remain close to its optimum if G does not change too rapidly, this also allows for the algorithm to run computationally more efficiently and prevents overfitting. During the early stages of training, it will be quite easy

for D to discriminate between data and model samples, since G will still be learning to output more realistic samples, therefore G's objective function $log\left(1 - D\left(G(z)\right)\right)$ will saturate, so an alternative objective function $log\, D\left(G(z)\right)$ is maximized in practice by G, which does not change the dynamics of D and G much but allows for gradients that are sufficiently large to perform useful stochastic gradient descent.

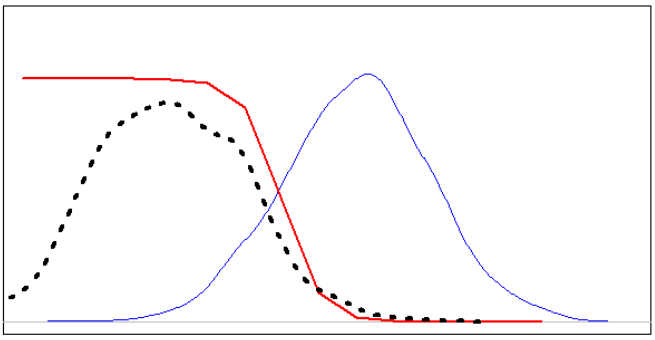


Figure 45: Gan Densities during training, close to convergence, P(x) is shown in black, G(z) in blue and D(G(z)) in red

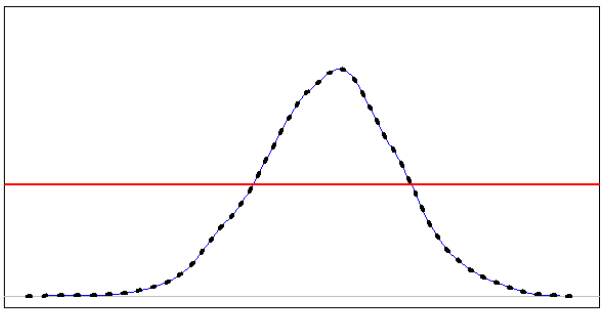


Figure 46: Gan Densities during training, once the Algorithm has converged, G(z) matches P(x) perfectly and D(G(z)) outputs 0.5 everywhere

5.4 Adversarial Autoencoders

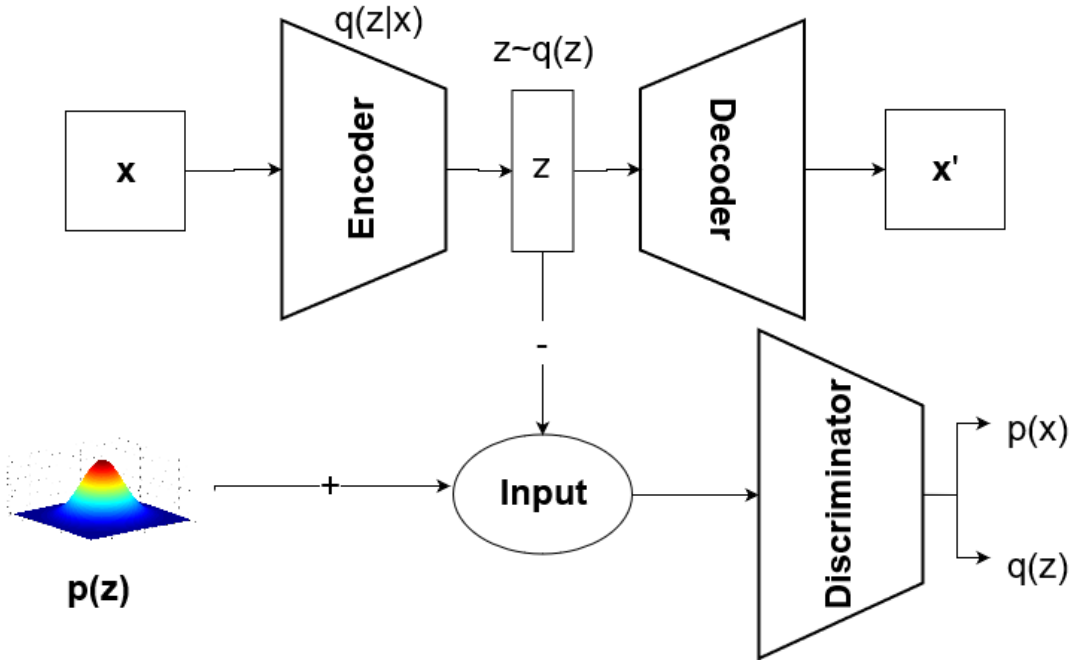


Figure 47: Simplified diagram of an Adversarial Autoencoder

Adversarial Autoencoders match the aggregated posterior of the latent space vector from an autoencoder $q(z) = \int_x q(z|x)p_d(x)dx$ with an arbitrary prior distribution $p(z)$, a process which results in meaningful samples being generated from any sample from any part of the prior space [53].

The decoder function learns to convert the data distribution to the prior distribution and the decoder function learns a function to map from the imposed prior distribution to the data distribution. In this set-up, the generator of the GAN also acts as the encoder function of the autoencoder, a process which assists the generator in fooling the discriminator of the GAN into misclassifying simulated data as real data [53].

6 Implementation: High Energy Physics Detector Simulations

6.1.1 Assessing Simulation Performance

In order to assess how well Geant4 and each type of Deep Generative Latent Variable algorithm modelled real data obtained from the ALICE TRD during production, each type of simulated data was independently assessed using the same neural network architecture.

Specifically, in order to numerically determine the accuracy of each type of simulated data, a 2D Convolutional classifier was trained to discriminate between real and simulated data, training and loss curves are shown for Geant4 in **Error! Reference source not found.**, for VAE in **Error! Reference source not found.**, for GAN in **Error! Reference source not found.** and for AAE in **Error! Reference source not found.**. Some types of simulated data was easier to discriminate from real data and therefore early stopping was applied and not all networks were trained for the same number of epochs.

Following training, predictions were ran on real and simulated data, in order to view the distribution of $P(real)$ estimates for both real and each type of simulated dataset. These results are depicted in histogram form in Figure 49 for Geant4 data, **Error! Reference source not found.** for VAE data, Figure 55 for GAN data and Figure 58 for AAE data.

Lastly, in order to pertinently visualize images from each type of simulated across the distribution of $P(real)$ estimates for that data type, images were sampled in order to show which simulated samples were easy to discriminate from real data and which simulated samples were more realistic and therefore harder to distinguish from real data. In practice, twelve samples are shown for each simulated datatype, in order of increasing $P(real)$ estimates, with the first and the last image for each simulated datatype representing the minimum and maximum $P(real)$ estimate for that datatype. These sampled images are shown in Figure 50 for Geant4 data, **Error! Reference source not found.** for VAE data, Figure 56 for GAN data and Figure 59 for AAE data.

Code used to discriminate Geant4 data from real data, as well as code used to build and similarly assess various Deep Generative/ Latent variable models can be found here [viii].

Figure 48 shows the 2D CNN classifier used to discriminate each of the simulated datasets from real data. This architecture's weights were reinitialised after each time it was trained, i.e. the model was recompiled from scratch and trained independently for each individual simulated dataset. Convolutional layers had 16 and 32 filters,

respectively, both convolutional layers were implemented with a kernel size of 4×4, using "valid" padding and ReLU activation functions. Independent max pooling layers were applied with a pool-width of 2×2. All dense layers, including the output layer used sigmoid activation functions. This model was trained using the Adam Optimiser at a learning rate of $\eta = 0.00001$, binary cross-entropy loss and a batch size of 32. Models were trained for different number of epochs, as can be seen in the training accuracy and loss curves for each model type.

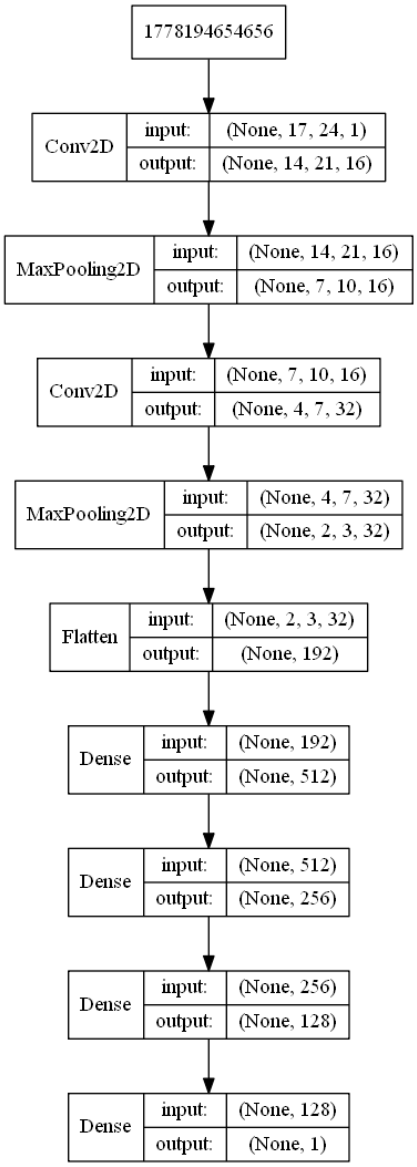


Figure 48: 2D CNN used to individually discriminate each type of simulated data from real data

6.2 Implementation: Geant4

In order to prove that Geant4 simulations might not be as accurate as assumed to be, a simulation was run, set to generate pions from the following LHC run: 2016/LHC16q/000265343. A convolutional neural network was able to distinguish simulated pions from real pions obtained during that run to a high degree of accuracy and therefore motivated the Deep Generative Modelling Section of this thesis.

6.2.1 Geant4 Configuration and Simulation

Please see the following repository[ix] for code used to Configure simulations (Config.C), code to simulate pions as per this specification (sim.C), code used to reconstruct hits for the TRD, TPC and other detectors whose reconstruction is depended upon for the reconstruction of TRD digits (rec.C) and code used to filter TRD digits and deliver data in the same format produced for real data (ana.C).

6.2.1.1 Distinguishing Geant4-Simulated Data from Real Data

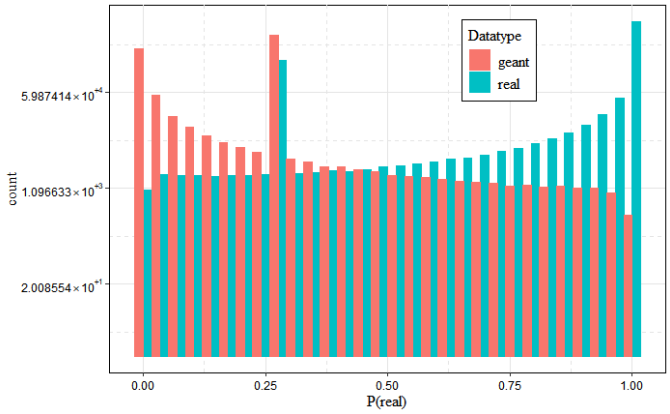
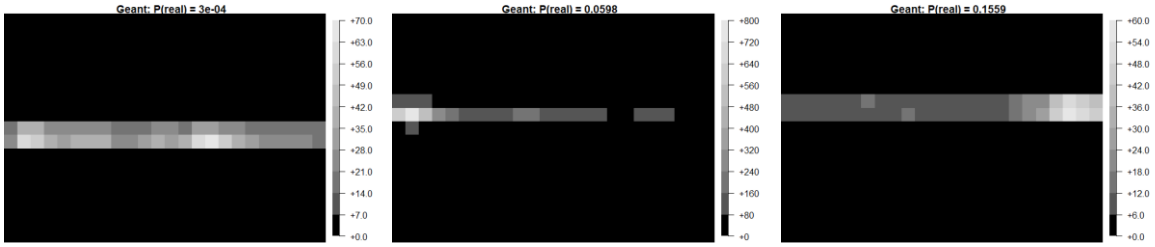


Figure 49: Distribution of $P(real)$ estimates for Geant4 vs Real data based on predictions from the discriminative neural network discussed above



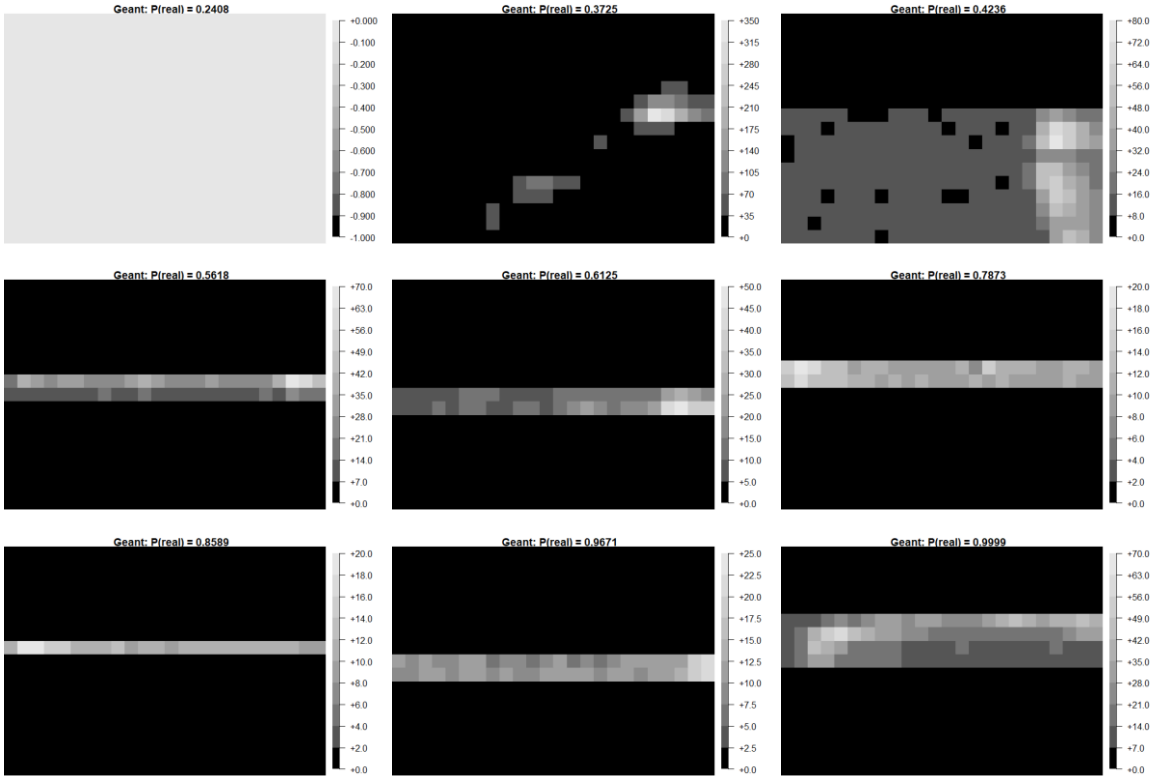


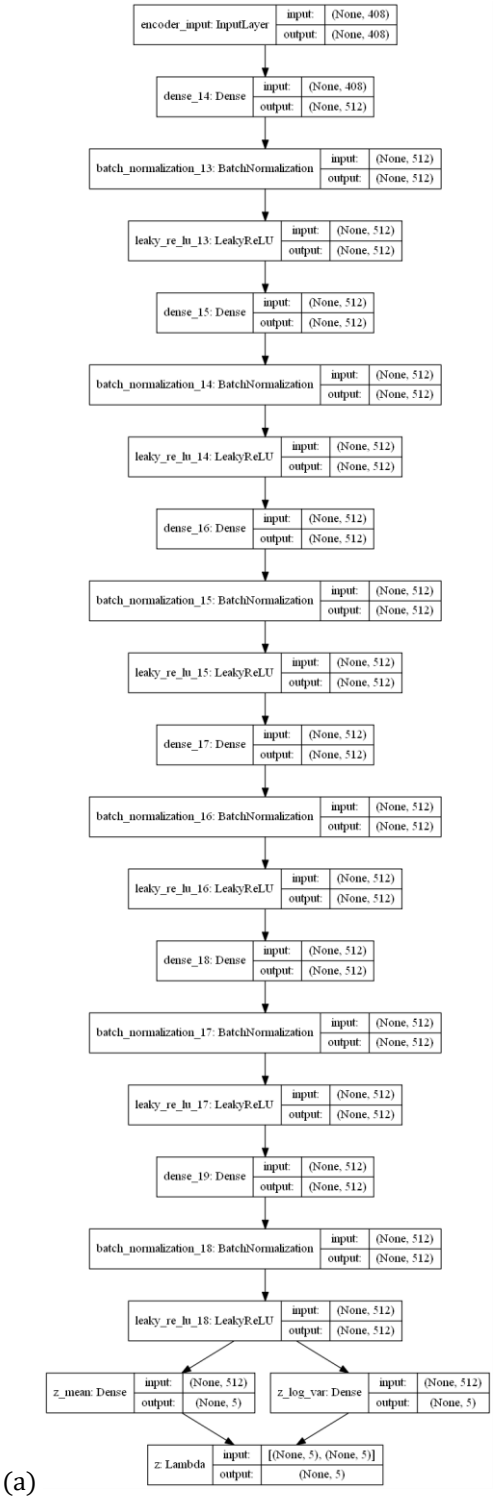
Figure 50: Twelve sampled Geant4-simulated pions arranged in order of increasing P(real) estimates

6.2.2 Discussion: Geant4

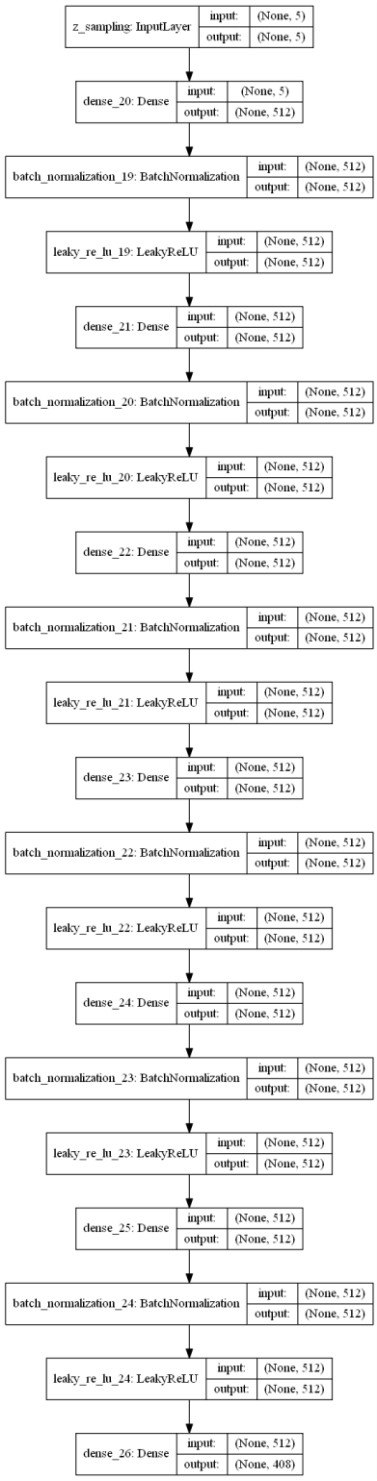
6.3 Implementation: Variational Autoencoders

6.3.1.1 Setup of the most successful Variational Autoencoder:

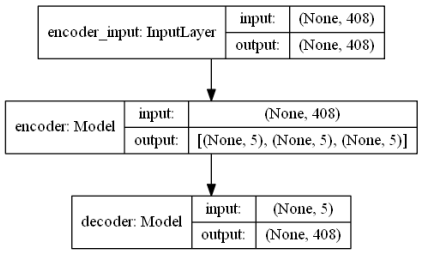
- N_{latent}=5
- Batch size=128
- Epochs = 164
- Sample size for each epoch = 500000
- Optimizer = Adam with learning rate $\eta = 10^{-8}$


encoder_input: InputLayer | input: (None, 408) | output: (None, 408)
dense_14: Dense | input: (None, 408) | output: (None, 512)
batch_normalization_13: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_13: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_15: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_14: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_14: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_16: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_15: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_15: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_17: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_16: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_16: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_18: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_17: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_17: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_19: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_18: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_18: LeakyReLU | input: (None, 512) | output: (None, 512)
z_mean: Dense | input: (None, 512) | output: (None, 5)
z_log_var: Dense | input: (None, 512) | output: (None, 5)
z: Lambda | input: [(None, 5), (None, 5)] | output: (None, 5)

(a)


z_sampling: InputLayer | input: (None, 5) | output: (None, 5)
dense_20: Dense | input: (None, 5) | output: (None, 512)
batch_normalization_19: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_19: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_21: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_20: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_20: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_22: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_21: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_21: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_23: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_22: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_22: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_24: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_23: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_23: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_25: Dense | input: (None, 512) | output: (None, 512)
batch_normalization_24: BatchNormalization | input: (None, 512) | output: (None, 512)
leaky_re_lu_24: LeakyReLU | input: (None, 512) | output: (None, 512)
dense_26: Dense | input: (None, 512) | output: (None, 408)

(b)



(c)

Figure 51

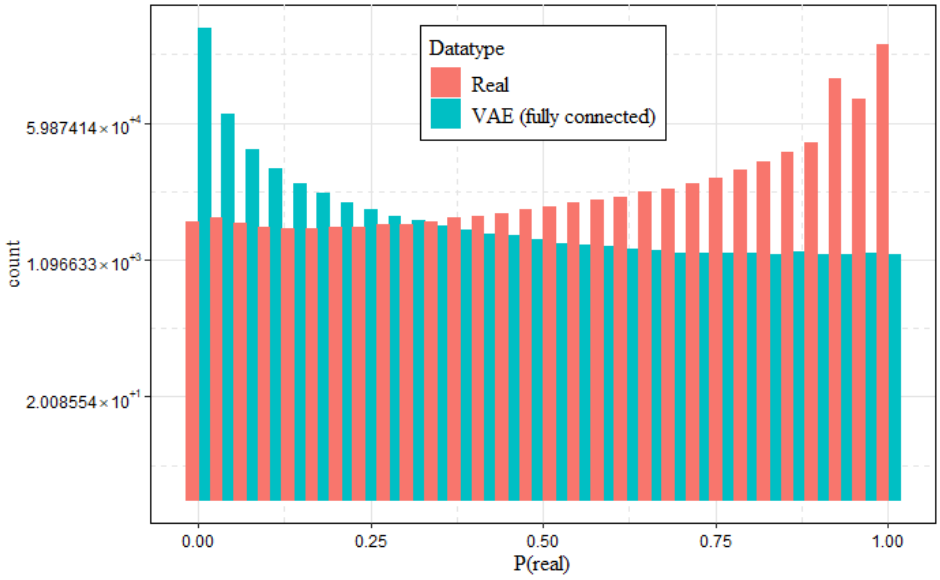
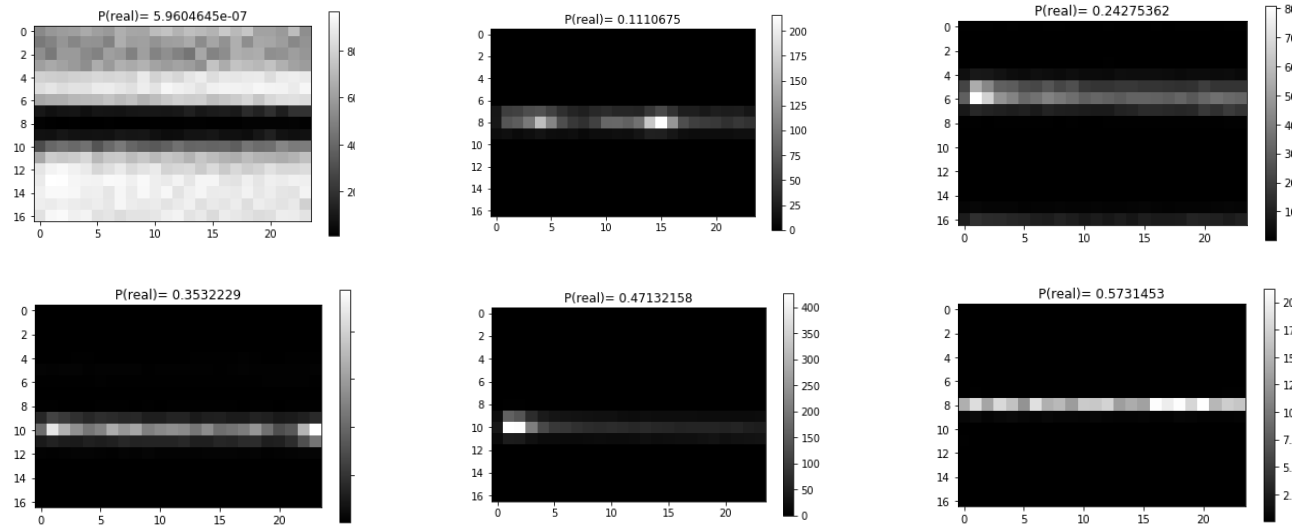


Figure 52



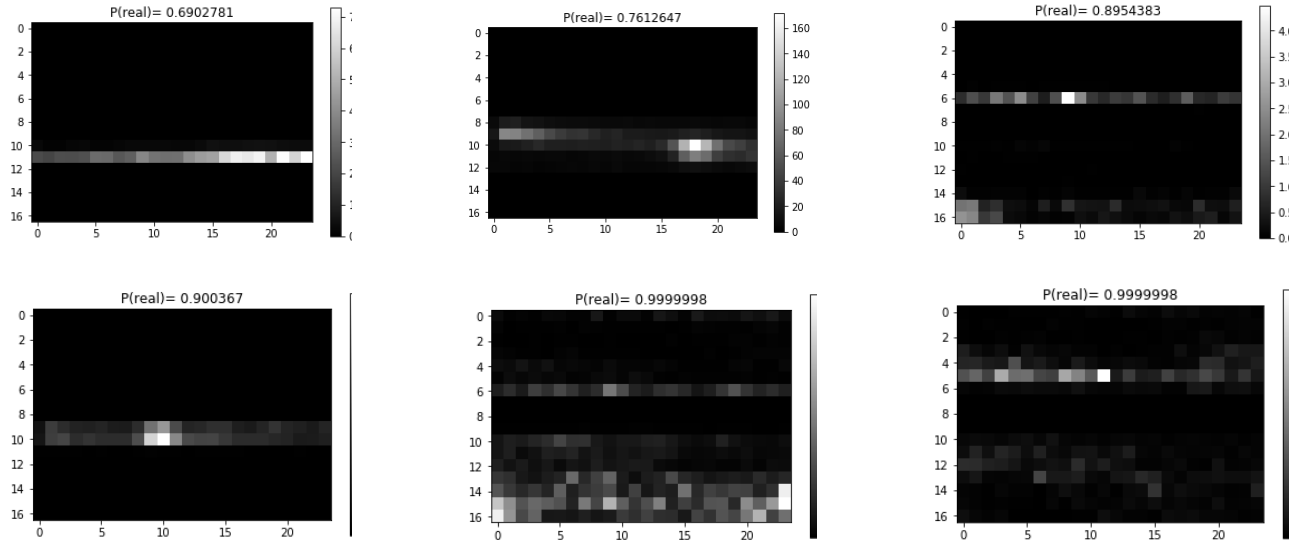


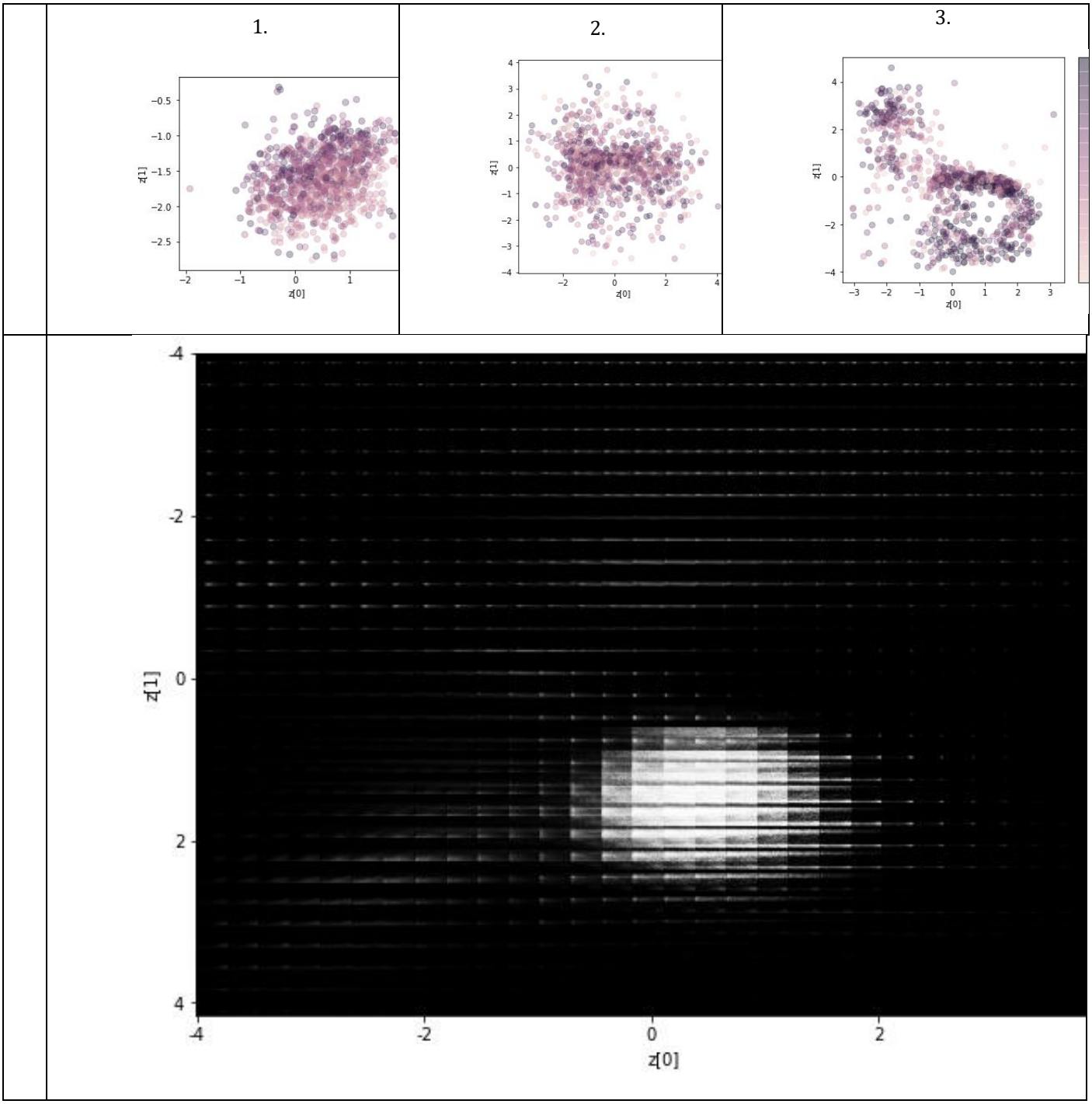
Figure 53

6.3.2 Exploration of the VAE latent space

Table 5: (a) Scatterplot projections of simulated data points along latent space dimensions (z_0, z_1). Associated P(real) estimates for simulated tracklets produced from these latent points are represented on an independent colour scale for each plot. Areas of the latent space that result in more realistic simulated images result in clear clustering along a manifold in plot (a.3). Similar grouped clusters were seen at P(real)>=0.99 for all other combinations of axes, not shown due to lack of interpretability

(b) A linear interpolation of the latent space along latent dimensions (z_0, z_1), where the other three latent dimensions $(z_1, \dots, z_4)$ are all set at $z_i = 0$, this allows one to visualize one aspect of how the latent space vector of a VAE encodes specific features. Various other projections by combining other pairs of latent dimensions are possible (not shown here).

	P(real)< 10^{-5}	**0.49<=P(real)<= 0.51**	**P(real)>=0.99**



6.3.3 Discussion: Variational Autoencoders

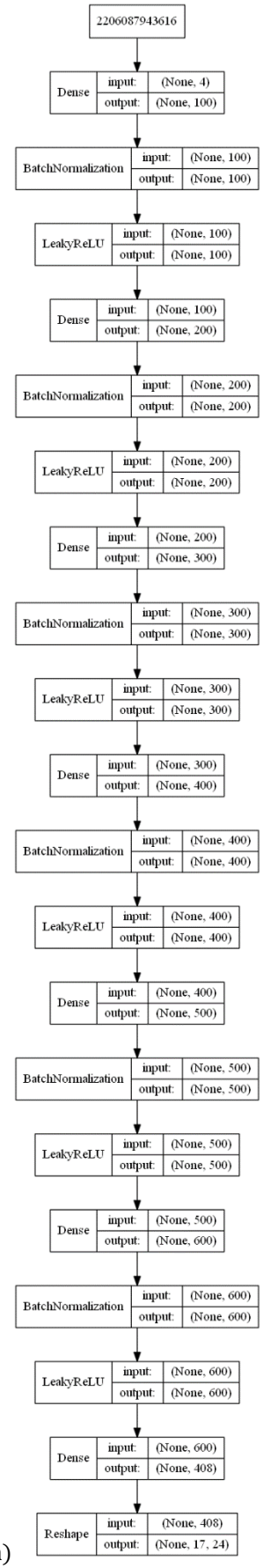
Mode collapse

MSE loss is shit

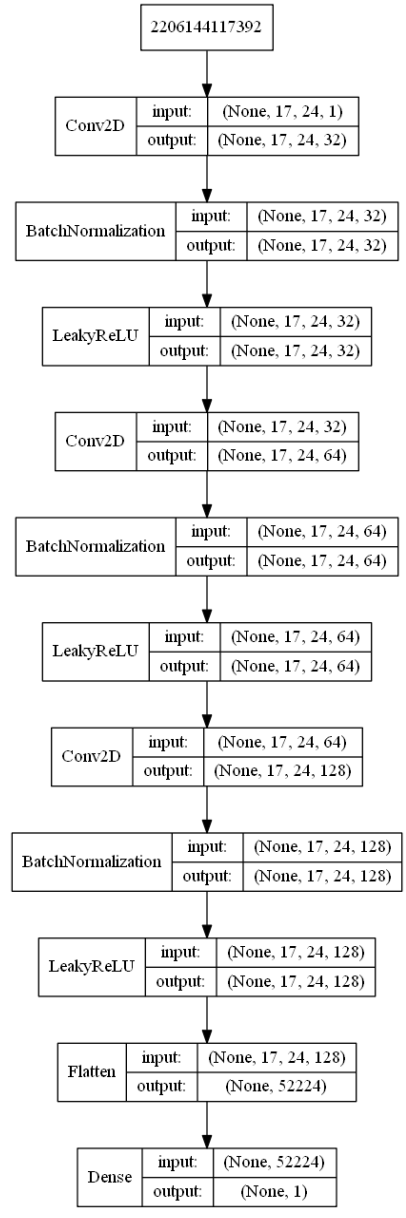
6.4 Implementation: Generative Adversarial Networks

6.4.1 Setup of the most successful GAN:

- $n_{latent} = 4$
- Batch size = 32
- Optimizers:
    - Discriminator: SGD at Learning Rate $\eta = 0.00004$
    - Generator: Adam at Learning Rate $\eta = 0.00002$
- Epochs = 200 000
- Label smoothing:
    - "True" labels were smoothed as follows: $y_{real} \sim U(0.71, 1.21)$
    - "False" labels were smoothed as follows: $y_{GAN} \sim U(0, 0.29)$
- Input image pixels were scaled to be in the range [-1,1]
- Generator's output layer bias term was initialised using a truncated normal distribution, as follows: $b \sim TN(\mu = -2, \sigma^2 = 0.4, a = -2.2, b = -1.8)$


2206087943616
Dense input: (None, 4) output: (None, 100)
BatchNormalization input: (None, 100) output: (None, 100)
LeakyReLU input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 200)
BatchNormalization input: (None, 200) output: (None, 200)
LeakyReLU input: (None, 200) output: (None, 200)
Dense input: (None, 200) output: (None, 300)
BatchNormalization input: (None, 300) output: (None, 300)
LeakyReLU input: (None, 300) output: (None, 300)
Dense input: (None, 300) output: (None, 400)
BatchNormalization input: (None, 400) output: (None, 400)
LeakyReLU input: (None, 400) output: (None, 400)
Dense input: (None, 400) output: (None, 500)
BatchNormalization input: (None, 500) output: (None, 500)
LeakyReLU input: (None, 500) output: (None, 500)
Dense input: (None, 500) output: (None, 600)
BatchNormalization input: (None, 600) output: (None, 600)
LeakyReLU input: (None, 600) output: (None, 600)
Dense input: (None, 600) output: (None, 408)
Reshape input: (None, 408) output: (None, 17, 24)

(a)


2206144117392
Conv2D input: (None, 17, 24, 1) output: (None, 17, 24, 32)
BatchNormalization input: (None, 17, 24, 32) output: (None, 17, 24, 32)
LeakyReLU input: (None, 17, 24, 32) output: (None, 17, 24, 32)
Conv2D input: (None, 17, 24, 32) output: (None, 17, 24, 64)
BatchNormalization input: (None, 17, 24, 64) output: (None, 17, 24, 64)
LeakyReLU input: (None, 17, 24, 64) output: (None, 17, 24, 64)
Conv2D input: (None, 17, 24, 64) output: (None, 17, 24, 128)
BatchNormalization input: (None, 17, 24, 128) output: (None, 17, 24, 128)
LeakyReLU input: (None, 17, 24, 128) output: (None, 17, 24, 128)
Flatten input: (None, 17, 24, 128) output: (None, 52224)
Dense input: (None, 52224) output: (None, 1)

(b)

Figure 54: GAN (a) Generator, and (b) Discriminator

6.4.2 Distinguishing GAN-Simulated Data from Real Data

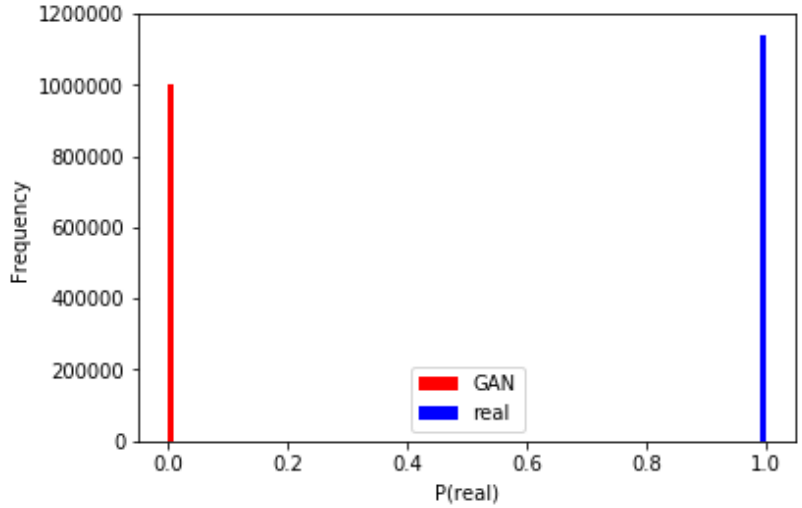
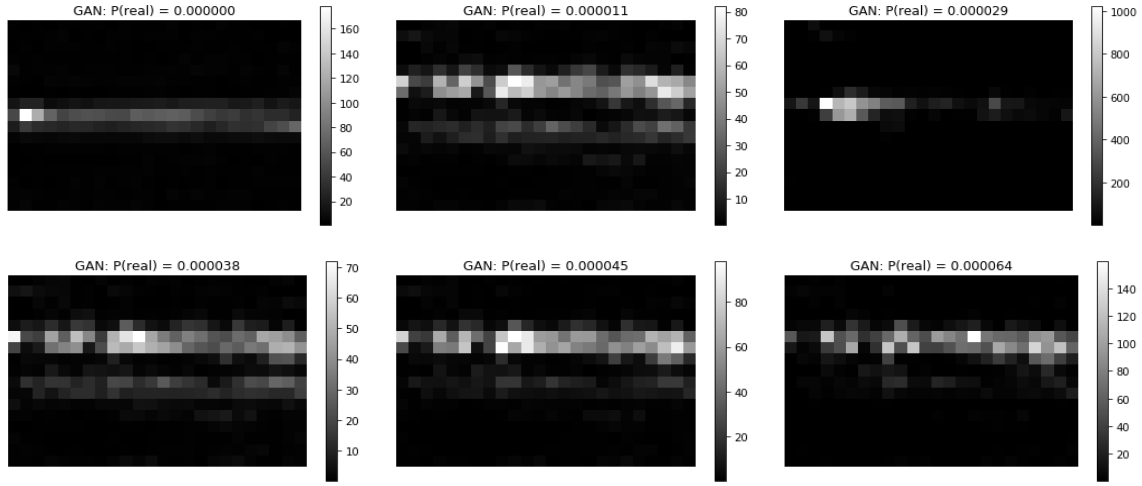


Figure 55: Distribution of $P(real)$ estimates for GAN and real data



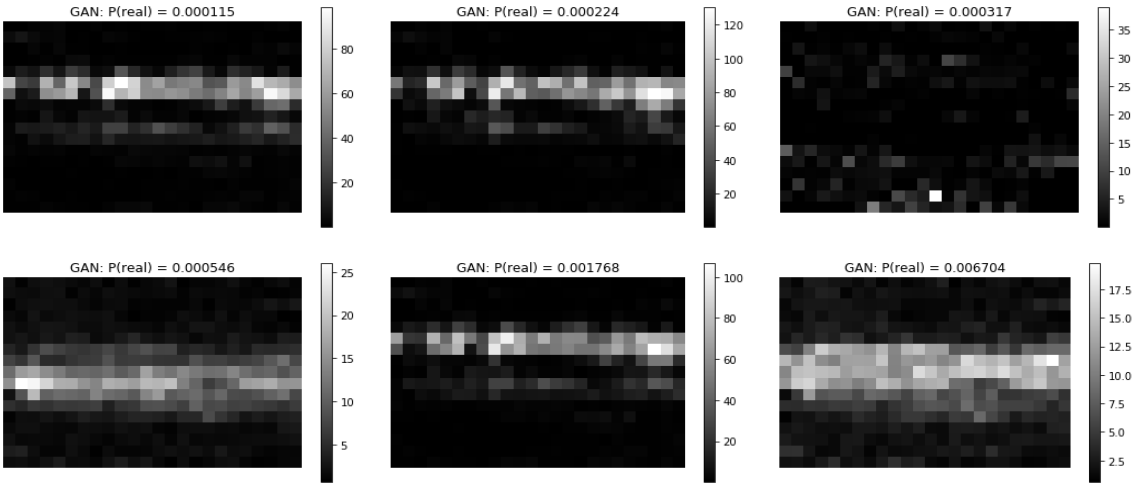


Figure 56: Twelve example GAN-simulated images, arranged in order of increasing $P(real)$ estimates

6.4.3 Discussion: Generative Adversarial Networks

6.5 Implementation: Adversarial Autoencoders

6.5.1 Set-up of most successful Adversarial Autoencoder:

- $n_{latent} = 4$
- Discriminator optimizer: SGD with learning rate $\eta = 0.00003$
- Generator optimizer: Adam with learning rate $\eta = 0.00001$ and parameter $\beta_1 = 0.5$ (Using different learning rates for the Discriminator and Generator is a method commonly suggested in practice to increase the stability of GAN training. It worked quite well for AAEs as well).
- Label smoothing:
    - Positive labels: smoothed to be in the range 0.9-1.4
    - Negative labels: smoothed to be in the range 0-0.1

  Label smoothing is implemented as follows:

$$y_{real} = 1 - 0.1 + (u \times 0.5)$$

$$y_{simulated} = 0 + (u \times 0.1)$$

  where $u \sim U(0,1)$.

This is another suggested method that improves GAN training stability, which also worked quite well for AAEs. This technique ensures that the Discriminator is never really sure about its prediction, giving the generator an advantage.

- Epochs = 400 000
- Batch size=32

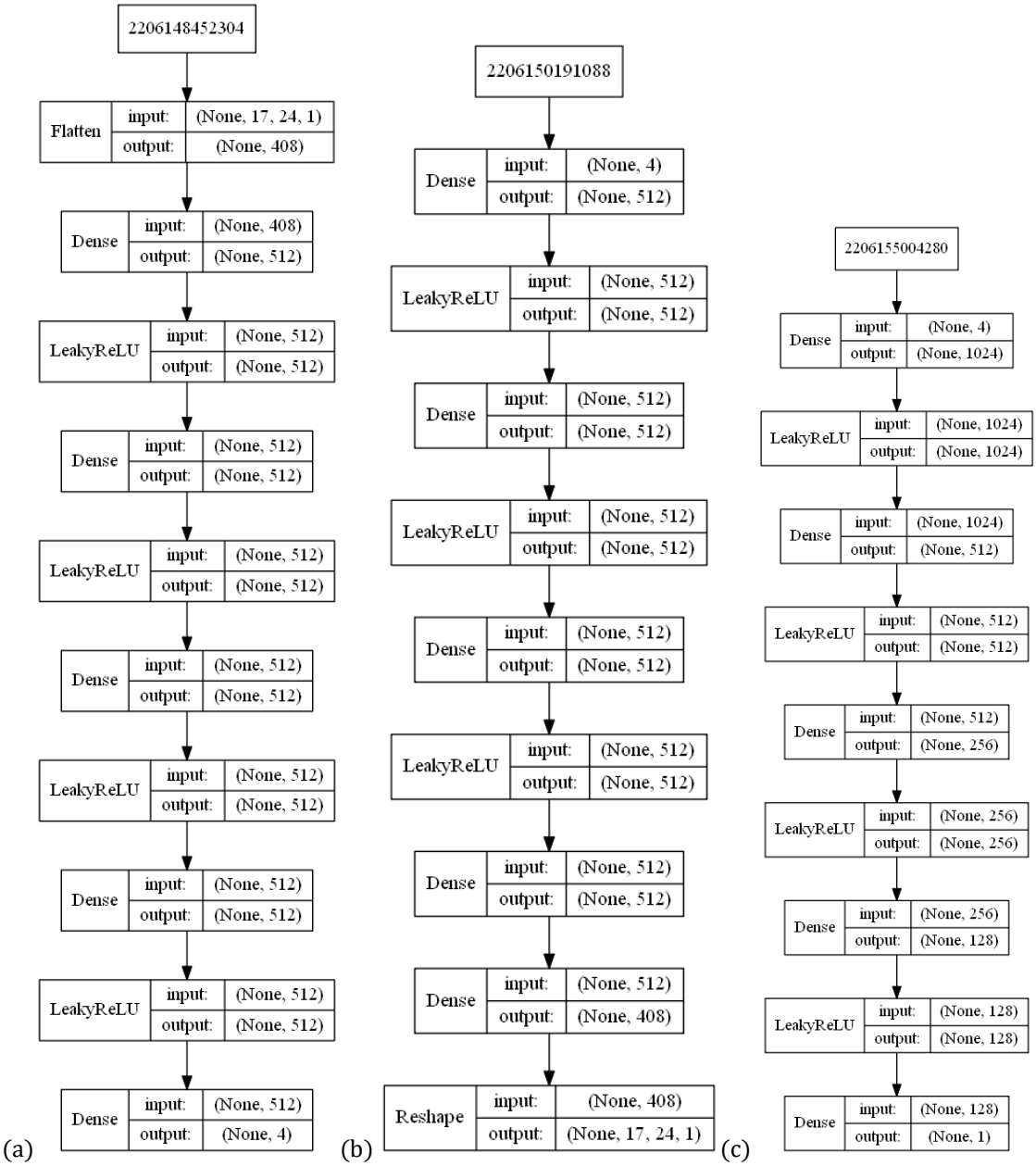


Figure 57: Adversarial Autoencoder (a) Encoder (b) Decoder (c) Discriminator

6.5.2 Distinguishing AAE-Simulated Data from Real Data

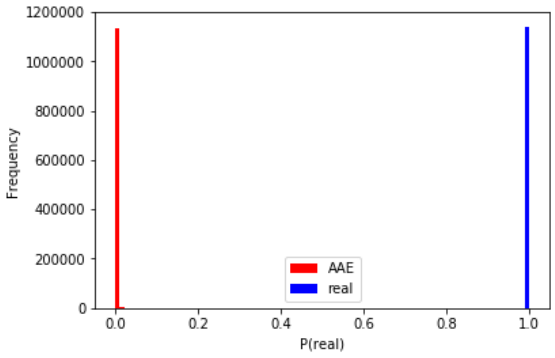


Figure 58: Distribution of $P(real)$ estimates for AAE and real data

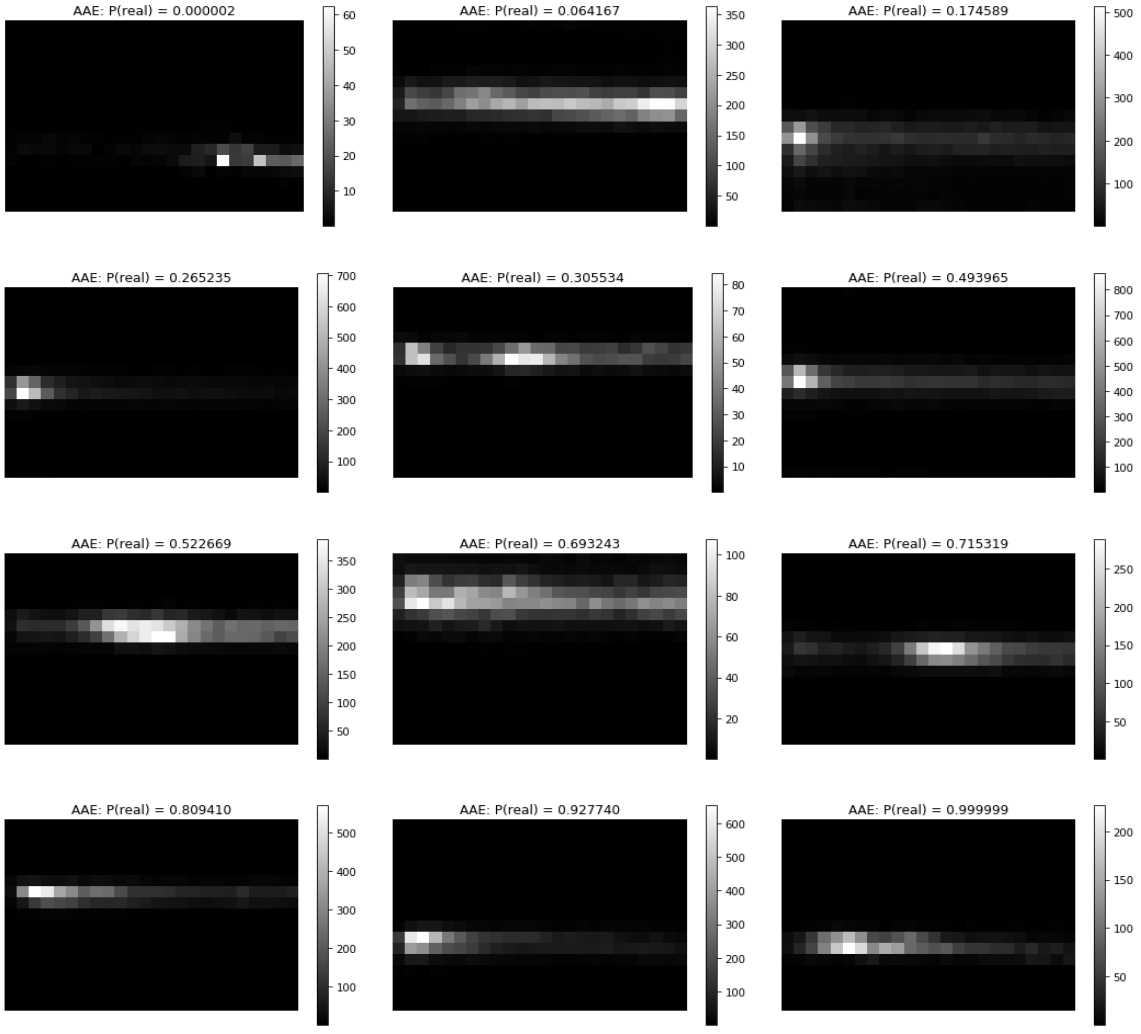


Figure 59: Twelve example AAE-simulated images, arranged in order of increasing $P(real)$ estimates

6.5.3 Discussion: Adversarial Autoencoders

6.6 Chapter Conclusions

It has been found during this project that it is arguably much more practical to train a generative model locally than on a server. The advantage of training locally lies in being able to qualitatively assess how realistic images appear (by plotting examples at specified stages during training) and to force-stop training of the model when it is clear that it is not improving.

Model loss and accuracy metrics are less informative when training a Generative model, since realistic images can sometimes be generated, even when loss appears to be quite high. Similarly, accuracy cannot be calculated when techniques such as label smoothing is used with binary cross-entropy as the loss function; in this case, discriminator accuracy remains at 0.

There are some pitfalls to this approach as well. Models with differing architectures, learning rates and other hyperparameter settings also take different numbers of training rounds/ epochs before they start generating realistic images. Sometimes, models that seem promising during early epochs, seem to get worse during later epochs. Since only one model can be trained at a time when running locally, and since loading all 9.3 million tracklets into memory and training generative models (sometimes using computationally expensive convolutional operations) is resource intensive and costly timewise, quite often a trade-off needs to be made and a model might have to be stopped before it reaches its full generative potential.

Additional constraints imposed by local training include having to be very careful about changing hyperparameters or architectures, because of the computational- and time cost, but this also means that fewer experiments can be conducted compared to training many neural network classifiers on a server and being able to quite reliably assess their performance based solely on training metrics and decreasing loss functions.

It was found that building generative models is extremely valuable as a tool to teach oneself to develop an intuitive understanding of how neural networks work in practice. Changing a specific hyperparameter or trying out a different architecture and seeing how it performs allows one to reason about the results (again: *qualitatively* assessing the images that a generative model produces as training progresses comes in handy here).

Using suggested techniques that have been proven to improve the stability of generative models in practice[x] (such as label smoothing and using different optimizers for the generator and the discriminator, using leaky ReLU activation functions for both the generator and discriminator at $\eta = 0.2$, and using Batch Normalisation with a momentum setting of 0.8 after each layer of both the generator and discriminator), which might be hard to motivate mathematically, but work quite well in practice, shows that practical experience is often equally as valuable as theoretical knowledge when working with advanced deep learning algorithms.

7 CONCLUSIONS

7.1 Machine Learning for Particle Identification

While convolutional neural networks are generally accepted as being more conducive towards accurate classification of array-like data than fully connected neural networks [54], their design cannot correct for the lack of proper calibration of input data, unless more information (e.g. chamber gain and pad-by-pad calibration factors) are provided to them.

The CNNs developed in this thesis (trained on uncalibrated data) produced pion efficiency results comparable to LQ1D and LQ2D methods (performed on properly calibrated input data), but their performance was much worse than LQ7D and fully connected neural networks (performed on properly calibrated input data).

When comparing results from this thesis across methods (on an uncalibrated dataset), one does see significant improvements when using 2D CNNs, compared to fully connected neural networks, tree-based methods, networks making use of LSTM cells and 1D CNNs. At the hand of these results, one could posit that CNNs could provide additional decreases in pion efficiency on a properly calibrated input dataset, over and above what is achievable with feedforward neural networks. This might be an avenue worth exploring, especially since, from the advent of ROOT 6.12, the implementation of convolutional- as well as recurrent layers have been supported in ROOT's Toolkit for Multivariate Analysis (TMVA) on CPU, and more recently CNNs are also supported on GPU (ROOT 6.14) [55].

The required speed at which predictions need to be made during an LHC run is such that, even though convolutional neural networks might result in slight increases in performance on pion rejection and electron acceptance rates, they are unlikely to be practically implementable on the available detector computing hardware (i.e. during a run), but this data could always be analysed *ex post facto*.

Besides particle identification, the era of open source machine learning and more affordable high performance computing opens up various exciting avenues in particle physics research, including the detection of outliers that could be indicative of Physics Beyond the Standard Model (BSM).

7.2 High Energy Physics Detector Simulations

This thesis has shown that deep generative models could indeed be an avenue to pursue in more formal future research at CERN. In particular, it has shown that Adversarial Autoencoders are able, due to their training procedure, to produce meaningful samples from anywhere in the latent space it samples from. Future work should definitely look into using Conditional GAN techniques, which enable the deep learning practitioner to have more control over the samples produced (for example, specifying total energy deposit by scaling and multiplying that value with the latent vector for each image). While this technique was attempted during this project, it was largely unsuccessful and would require a bit more time to perfect.

In addition, by using Auxiliary Classifier GANs (ACGANs) one might be able to specify the type of particle one wishes to produce. An ACGAN is a GAN variant in which the Generative model is provided with a class label in addition to a randomly sampled latent vector, from which it attempts to produce an image of the specified class; and the Discriminative model is – as usual – tasked with discriminating real or fake images, while also receiving the class label and an image as input).

8 BIBLIOGRAPHY

[1] M. Paganini, L. de Oliveira and B. Nachman, "CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks," *Physical Review D,* vol. 97, 2017.

[2] F. Carminati, A. Gheata, P. Mendez Lorenzo, S. Sharan and S. Vallecorsa, "Three dimensional Generative Adversarial Networks for fast simulation," *Journal of Physics: Conference Series,* vol. 1085, p. 032016, 2018.

[3] S. Vallecorsa, "Generative Models for Fast Simulation," *Journal of Physics: Conference Series,* vol. 1085, p. 022005, 2018.

[4] M. Thomson, Modern Particle Physics, Cambridge, UK: Cambridge University Press, 2013.

[5] J. Rafelski, "Connecting QGP-Heavy Ion Physics to the Early Universe," *Nuclear Physics B - Proceedings Supplements,* vol. 243, 2013.

[6] H. Satz, "The Quark-Gluon Plasma - A Short Introduction," *Nuclear Physics A - NUCL PHYS A,* vol. 862, pp. 4-12, 2011.

[7] "Cern Document Server," [Online]. Available: https://cds.cern.ch/record/2025215?ln=en. [Accessed 21 November 2019].

[8] "Cern Document Server," [Online]. Available: https://cds.cern.ch/record/2026889?ln=en. [Accessed 21 November 2019].

[9] The Steven Hawking Center for Theoretical Cosmology, [Online]. Available: http://www.ctc.cam.ac.uk/images/contentpics/outreach/cp_universe_chronology_large.jpg.

[10] "Week 3: Thermal History of the Universe," [Online]. Available: www.astro.caltech.edu/~george/ay127/kamionkowski-earlyuniverse-notes.pdf. [Accessed 20 February 2019].

[11] F. de Rose, "The birth of CERN," *Nature,* vol. 455, pp. 174-175, 2008.

[12] CERN, "CERN Annual Report 2018," *CERN Annual Reports,* 2019.

[13] O. Bruning, P. Collier, P. Lebrun, S. Myers, R. Ostojic, J. Poole and P. Proudlock, LHC Design Report, vol. Volume 1: The LHC Main Ring, Geneva: CERN Scientific Information Service, 2004.

[14] "Cern Document Server," [Online]. Available: https://cds.cern.ch/record/842399?ln=en. [Accessed 20 November 2019].

[15] "Cern Document Server," [Online]. Available: https://cds.cern.ch/record/842700. [Accessed 20 November 2019].

[16] "Cern Document Server," [Online]. Available: https://cds.cern.ch/record/842611. [Accessed 20 November 2019].

[17] M. A. Hone, "The Duoplasmatron Ion Source for the new CERN LinAc preinjector," *CERN/PS/LR,* vol. 79, no. 37, 1979.

[18] CERN, "The PS complex as proton pre-injector for the LHC - Design and implementation report," 2000.

[19] A. Beuret, J. Borburgh, H. Burkgardt, C. C. Carli, A. Fowler, M. Gourber-Pace, S. Hancock and M. Hourican, "The LHC Lead Injector Chain," *9th European Particle Accelerator Conference,* p. 1153, 2004.

[20] CERN, "The CERN Accelerator Complex," [Online]. Available: https://cds.cern.ch/record/2636343/files/CCC-v2018-print-v2.jpg?subformat=icon-1440. [Accessed 26 January 2019].

[21] ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *Journal of Instrumentation,* vol. 3, no. S08003, 2008.

[22] CMS Collaboration, "The CMS Experiment at the CERN LHC," *Journal of Instrumentation,* vol. 3, no. S08004, 2008.

[23] CERN, "LHC Experiments," [Online]. Available: https://home.cern/science/experiments. [Accessed 21 February 2019].

[24] CERN, “ALICE Experiment,” [Online]. Available: https://home.cern/science/experiments/alice. [Accessed 21 Fenruary 2019].

[25] CERN, “LHCb Experiment,” [Online]. Available: https://home.cern/science/experiments/lhcb. [Accessed 21 February 2019].

[26] ALICE, “ALICE Homepage,” [Online]. Available: http://alice.web.cern.ch/. [Accessed 21 February 2019].

[27] The ALICE Collaboration, The ALICE Experiment at the CERN LHC, INSTITUTE OF PHYSICS PUBLISHING AND SISSA, 2008.

[28] Y. Pachmayer, “Particle Identification with the ALICE Transition Radiation Detector,” Nuclear Instruments and Methods in Physics - Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, pp. 292-295, 2014.

[29] The ALICE Collaboration, The Technical Design Report of the Transition Radiation Detector, Geneva: CERN, 2001.

[30] Particle Data Group, The Review of Particle Physics, 2018.

[31] ALICE Collaboration, “The ALICE Transition Radiation Detector: construction, operation, and performance,” *Nuclear Instruments and Methods in Physics Research, A,* vol. 881, pp. 88-127, 2018.

[32] J. C. Watkins, An Introduction to the Science of Statistics: From Theory to Implementation, Preliminary Edition, University of Arizona, 2016.

[33] A. Aamodt, F. Bock, P. Braun-Munzinger, T. Dietel, P. Gonzalex, M. Heide, M. Ivanov, K. Koch, d. G. P. Ladron, A. Marin, M. Rammler, K. Reygers, D. Rohrich, E. Serradilla and J. Wessels, “Photon reconstruction with conversions in ALICE,” in *GSI Scientific Report*, 2009.

[34] A. Wilk, Particle Identification Using Artificial Neural Networks with the ALICE Transition Radiation Detector, 2010.

[35] CERN, “ROOT Data Analysis Framework: User's Guide,” May 2018. [Online]. Available: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideA4.pdf .

[36] “ROOT 5 Reference Guide,” [Online]. Available: https://root.cern/root/html534/ClassIndex.html.

[37] “ROOT 6 Reference Guide,” [Online]. Available: https://root.cern/doc/v616/.

[38] ALICE Collaboration (CERN), [Online]. Available: https://alice-doc.github.io/alice-analysis-tutorial. [Accessed 18 2 2019].

[39] CERN, "High Energy Physics Simulations," [Online]. Available: http://lhcathome.web.cern.ch/projects/test4theory/high-energy-physics-simulations. [Accessed 26 July 2019].

[40] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, Masachusetts: The MIT Press, 2016.

[41] "Wikimedia Commons," [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=224555. [Accessed 06 09 2019].

[42] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* vol. 65, no. 6, 1958.

[43] keras.io, "Available Activations," [Online]. Available: https://keras.io/activations/#available-activations. [Accessed 19 July 2019].

[44] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR 2015*, 2015.

[45] "Keras," [Online]. Available: https://keras.io/optimizers/. [Accessed 23 09 2019].

[46] Keras, "Keras Documentation: Losses," [Online]. Available: https://keras.io/losses/. [Accessed 26 09 2019].

[47] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollar, "Focal Loss for Dense Object Detection," *Computer Vision and Pattern Recognition,* 2018.

[48] G. Cowan, Statistical Data Analysis, Oxford: Oxford University Press, 1998.

[49] Geant4 Collaboration, "Geant4--a simulation toolkit," *Nuclear Instruments and Methods in Physics Research,* vol. 506, pp. 250-303, 2003.

[50] S. Agostinelli, J. Allison, J. Apostolakis and P. Arce, "Geant4 - a simulation toolkit," *Nuclear Instruments and Methods in Physics Research,* vol. A 506, pp. 250-303, 2003.

[51] C. Doersch, "Tutorial on Variational Autoencoders," ResearchGate, 2016.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," 2014.

[53] A. Makhzani, I. Goodfellow, B. Frey, J. Shlens and N. Jaitly, "Adversarial Autoencoders," 2016.

[54] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *arXiv Preprint,* 2019.

[55] K. Albertsson, S. Gleyzer, M. Huwiler, V. Ilievski, L. Moneta, S. Shekar, A. Vashista, S. Wunsch and O. A. Zapate Mesa, "New Machine Learning Developments in ROOT/TMVA," *EPJ Web of Conferences,* vol. 214, p. 06014, 2019.

[56] CERN, "ATLAS Experiment," [Online]. Available: https://home.cern/science/experiments/atlas. [Accessed 21 February 2019].

ACKNOWLEDGEMENTS

Firstly, I would like to thank my father, Christiaan Gerhardus Viljoen, for all the support – material, emotional and financial – he has selflessly provided to me throughout my life, and particularly towards my higher education journey. You have no idea how much appreciation I have for all the sacrifices you have made for me, and all the advice you have given me.

Secondly, I want to thank my aunt, Professor Emma Ruttkamp-Bloem, for all the mentoring she has provided to me in navigating the world of academia, and for the inspiration that her own academic career instils in me.

Thirdly, I want to thank Dr Thomas Dietel for providing me with this immense opportunity to be part of the largest scientific experiment in human history, and for the rigorous scientific guidance that he has, and continues to provide to me.

Lastly, I would like to thank my larger family, on both my father's and mother's side, for providing the loving and stable environment that makes any place we assemble Home.

Computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team: hpc.uct.ac.za

Travel to CERN was paid for by iThemba Labs via the SA-CERN agreement

ENDNOTES

[i] https://github.com/umbertogriffo/focal-loss-keras

[ii] https://gist.github.com/PsycheShaman/ea39081d9f549ac410a3a8ea942a072b

[iii] https://github.com/PsycheShaman/trdML-gerhard

[iv] http://alimonitor.cern.ch/

[v] https://gitlab.cern.ch/cviljoen/msc-thesis-data

[vi] https://github.com/PsycheShaman/MSc-thesis/tree/master/misc/example_pythonDict.txt

[vii] https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/Particle%20Identification

[viii] https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/Latent%20Variable%20Models

[ix] https://github.com/PsycheShaman/trdpid/sim

[x] https://github.com/soumith/ganhacks