Machine Learning for Particle Identification & Deep Generative Models Towards Fast Simulations for the ALICE Transition Radiation Detector at CERN

Christiaan Gerhardus Viljoen

Department of Statistics || Department of Physics

Faculty of Science

University of Cape Town

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

Supervised by: Dr Thomas Dietel

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 thesis outlines the application of machine learning techniques, predominantly deep learning techniques, towards certain 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 (A Large Ion Collider Experiment) Transition Radiation Detector (TRD), within the Large Hadron Collider (LHC) at CERN (The European Organization for Nuclear Research).

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 a high rate of electron acceptance (i.e. electron efficiency, ε_e), specifically $\varepsilon_e \approx 90\%$.

Summary of Results for Particle Identification

Particle identification in this thesis was performed on uncalibrated TRD digits data, in contrast to work that has been done in this area before. For this reason, the presented results are only comparable, in terms of accuracy, to some of the less useful methods that have been investigated by others. Nonetheless, a much more comprehensive search across the space of possible neural network types and architectures was done in this project and this exploratory work has resulted in some interesting findings coming to light, such as the usefulness of the Focal Loss function in mitigating for the extreme class imbalances present in this dataset. The main goals of the first aim of this thesis were: (1) to show that the obtained results were comparable to previous studies, with slightly inferior performance (as expected, due to the omission of the calibration phase of data pre-processing); and 2) getting to know the dataset at hand well enough to enable the second aim of this thesis, which is discussed below.

Aim2: High Energy Physics Detector Simulations

The second aim of this project centred around determining whether detector simulations obtained from Geant4 were as accurate as they are usually assumed to be. Additionally, for the first time, exploratory research was done into the feasibility of making use of latent variable/ deep generative models for fast detector simulations for the ALICE Transition Radiation Detector. To this end, a wide variety of generative models were prototyped, and the results of a few choice models will be presented.

Summary of Results for High Energy Physics Detector Simulations

Perhaps the most important result of this thesis is the fact that a convolutional neural network was able to distinguish between simulated data (obtained by making use of Geant4 as the detector simulation component) and true data obtained by the TRD during a specific LHC pPb run. An investigation is presented which delves into some of the major differences between the two types of data.

Additionally, various deep generative models were prototyped towards the simulation of the TRD detector response, some of these models produced results which could indicate that these models would be fruitful avenues to explore as part of the detector simulation component of the O^2 software currently being developed for LHC Run 3.

TABLE OF CONTENTS

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 (many of which are open source) can now be practically employed towards problems in the area of particle physics.

1.2 Aims

This Masters project centres around two main aims:

Aim1: Particle Identification (e vs π): an extensive amount of (fully connected-, 1D and 2D convolutional- and recurrent (LSTM-)) neural networks, as well as two tree-based methods (random forests and gradient boosting machines), were trained and assessed, to determine their ability to discriminate between electrons (e) and pions (π), produced during proton-lead (pPb) collisions conducted at the LHC in 2016, based on raw (uncalibrated) TRD digits data. Particle identification performance was defined by the ability of each model to minimize pion efficiency (ε_π, false positive rate), whilst maintaining an electron efficiency (ε_e, true positive rate) of $\varepsilon_e = 90\%$. A lower bound for the critical region (t_{cut}) in the likelihood-ratio distribution of $P(elec)$ predictions made by a Bayesian combination of probability estimates for up to six independent "tracklet" measurements of a single particle track, which results in $\varepsilon_e \approx 90\%$ was defined, in order to determine the ε_π for that model. The best set of results obtained, per 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. These results are compared against previous work done in this area. Using the focal loss function (a parameterised modification of the more traditionally used binary cross-entropy loss function, which down-weights the loss magnitude for well-classified examples), to inform gradient descent via backpropagation, was a crucial step which allowed for the use of the full dataset (which suffers from extreme class imbalances), without having to downsample the dataset to prevent the neural network from becoming biased towards predicting the dominant class. Momentum binning and incremental training per momentum bin perhaps also had some influence on the most

successful model achieving a much lower pion efficiency than other models built. An analysis of other distinguishing factors that could determine the relative success across models developed for this specific use case is presented. Note that the main motivation for performing particle identification in this project was to gain hands-on experience in optimising various types of deep learning models and to develop a better understanding of TRD data. This led into the second (more important) deep generative/ latent variable modelling phase of this project.

Aim 2: High Energy Physics Detector Simulations: Geant4, a Monte Carlo toolkit used to simulate particle interactions with matter (used in conjunction with the AliROOT physics analysis software, developed and used by the ALICE collaboration at CERN), was assessed in terms of how closely the simulated data it produces resembles true data taken by the TRD during collision events; by training a convolutional neural network to classify whether a specific image is from the "real"- or "Geant4 simulated"- data distribution and analysing the results. Distinguishing Geant4 data from real data was a trivial task when compared to the task of particle identification. This is one of the most important results reported in this thesis, given the wide use and general trust that is placed in the quality of Geant4 simulations. If nothing else, it indicates that various parameters used to set-up a Geant simulation can be fine-tuned in future work, in order to minimise the differences between simulations and true data. Furthermore, as a step towards fast simulation, various deep generative modelling 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 (GANs), Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs). Data produced during these deep generative simulations were then compared to real data, via the same methodology used to compare Geant4-simulated data to real data. In order to assess the feasibility of incorporating these types of models into future high energy physics event simulation software, an investigation was made into how realistic their simulated signals are; and a brief exploration of the latent space of a more successful VAE was conducted, in order to illustrate the potential interpretability of said latent space. While generative models are exceptionally hard to train, some very promising results (especially for VAEs and AAEs) indicate that they would indeed be worthwhile to pursue, as part of more formal and extended future research; particularly if the latent space for a specific model can be understood well enough to make the deep generative simulation process flexibly manipulable: hopefully to the extent that an exact particle type, travelling at a specific momentum, during a certain LHC run, which is subject to certain environmental parameters, etc. can be specified in advance of running a deep generative detector simulation. This level of control *is* currently possible with Geant4 and therefore the feasibility of using deep generative models for detector simulations will definitely depend on their customisability, interpretability and controllability (as well as, more obviously, their accuracy and speed).

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 ($X = (x_{ij}) \in \mathbb{N}^{17\times 24}$) 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 arriving at a test statistic $t(x)$ with sufficient statistical power to discriminate between: signals originating from e, and signals originating from π; and subsequently optimizing a cut-off point t_{cut} defining a critical region in the calculated $P(elec)$ distribution, which minimizes pion efficiency (false positive rate, ε_π), at a

significance level $\alpha = 0.1$ (i.e. where the true positive rate or electron efficiency is $\varepsilon_e = 90\%$). $P(elec)$ is calculated by combining the likelihoods of six tracklets according to a Bayesian approach and the Neyman-Pearson Lemma.

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 and had access to the raw data $X = (x_{ij}) \in \mathbb{N}^{17\times 24}$.

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 4.3 summarises the methodology followed, as well as 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 (perhaps, in part, because many more of these models were built, compared to other model types).

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 4.3.1.10.

1.3.2 High Energy Physics Detector Simulations

1.3.2.1 Geant4

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, such as detector response functions, etc.

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 proton-Lead (pPb) run (2016/LHC16q/000265343) conducted at the LHC 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 during 2016/LHC16q/000265343.

The task of distinguishing Geant4 simulations from true data was trivial compared to the task of particle identification. These results are presented in section 6.2.2.

1.3.2.2 Deep Generative Modelling

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, i.e. each type of simulation strategy, including Geant4 and Deep Generative Models were run through the same neural network, which was trained to distinguish said simulations from true TRD raw digits data.

In summary, Variational- and Adversarial Autoencoders performed particularly well, but the practical use of any of these deep generative techniques will be contingent on factors such as customisability of simulations and how well they can be made to integrate with existing simulation software such as Geant4 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, followed by a tabular delineation of major distinguishing features between electrons and pions.

A primordial state of deconfined matter understood to have been prevalent for a short period of time subsequent to the Big Bang, known as the Quark Gluon Plasma (QGP), which can be reproduced at the LHC on a minuscule 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. The way in which this data is recorded, processed and collected, as well as some mention of its calibration, is also discussed. This leads into current methods in use at ALICE for e vs. π discrimination as well as an assessment of their accuracy. Finally, Chapter 2 ends with a short overview of a subset of the various software platforms currently being utilised at CERN for HEP research, namely ROOT, AliROOT and Geant4.

Chapter 3 introduces various basic statistical concepts and theorems, which leads into an explanation of the specific statistical tests used for particle identification; this section, in turn, leads into an overview of the field of Deep Learning, starting with the concept of a single neuron and building up to the concepts of Fully Connected Feedforward-, Convolutional- and Recurrent- Neural Networks, which were all employed towards particle identification research in this project.

Chapter 4 is essentially a combined methods- and results- section which firstly outlines the procurement and structure of the dataset used for particle identification and (after giving a brief overview of results from previous theses that were submitted on this topic); delves into the various stages of modelling that were gone through: from extremely simple benchmark models, through an exploration of a large number of different neural network architectural varieties and hyperparameter settings, before arriving at the implementation of the convolutional neural network which resulted in the most successful particle identification results.

Chapter 5 discusses the theory needed to understand methods used for particle detector simulations, at the hand of the currently used Monte Carlo-based simulation framework (Geant4), as well as Deep Generative Modelling strategies which could potentially be employed in this area, by accurately modelling the underlying detector data distribution.

Chapter 6 is the Methods and Results section for the implementation of the concepts discussed in Chapter 5. Both Geant4 simulations and Deep Generative Model “simulations” are discussed in terms of how they were implemented and how closely they resembled true data, by running each simulation through the same convolutional neural network architecture, which was independently trained to discriminate each type of simulated data from true TRD raw digits data.

Chapter 7 summarises and pulls together the various threads of work done for this Masters project and gives some suggestions for future work.

Lastly, the Bibliography is followed by the Appendices, which contain additional material, which did not find its way into the main text and the Acknowledgements Section.

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].

The low energy manifestation of Quantum Electrodynamics (QED, the quantum field theory of the electromagnetic force) describes atoms as 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. 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 (v_e) is produced during both of the abovementioned processes [4].

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 explains 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 (where collision energies of ~13 TeV have been achieved), 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].

Elementary particles are currently considered to be truly elementary, in that they cannot be subdivided [4], the fundamental particles are also classified into different generations. Particles in each of the different generations have different flavour quantum numbers and masses, but identical interactions. The first generation of particles is the electron, electron neutrino, the up-quark and the down-quark. At higher levels of the mass hierarchy, one finds the second and third generations of elementary particles. The fundamental particles are summarised in Table 1.

There hasn't been any evidence of further generations than these three, and so – according to current understanding – all ordinary matter in the universe seems to be circumscribed by the following twelve fundamental fermions, reproduced from [4]. Note that ordinary matter only accounts for ~4.6% of the energy density in the Universe and that the rest consists of dark matter (~24%) and dark energy (~71.4%).

Table 1: The twelve fundamental fermions.

	Leptons			Quarks		
	Particle	**Charge (e)**	**Mass/GeV**	**Particle**	**Charge (e)**	**Mass/GeV**
First Generation	Electron (e^-)	-1	0.005	Down (d)	-1/3	0.003
	Eelctron-Neutrino (ν_e)	0	$< 10^{-9}$ †	Up (u)	+2/3	0.005
Second Generation	Muon (μ^-)	-1	0.106	Strange (s)	-1/3	0.1
	Muon-Neutrino (ν_μ)	0	$< 10^{-9}$ †	Charm (c)	+2/3	1.3
Third Generation	Tau (τ^-)	-1	1.78	Bottom (b)	-1/3	4.5
	Tau-Neutrino (ν_τ)	0	$< 10^{-9}$ †	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 6 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 $\bar{\mathrm{d}}$, 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 is not described by the Standard Model; therefore, Einstein's General Theory of Relativity is still the best explanation of this force.

The search to incorporate gravity, the Standard Model, as well as other unexplained physical phenomena such as the nature of dark matter and dark energy into a Grand Unified Theory is an ongoing area of research and has resulted in exciting new theoretical research avenues such as string theory and loop quantum gravity arising; these research streams are collectively known as Physics beyond the Standard Model (BSM) [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 (8 types of gluons, which are massless and confined to within bound states, along with quarks) 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 [5] and ATLAS [6] collaborations at CERN in 2012, but proposed in 1964 by three separate theoretical papers ([7], [8], [9]), 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, along with the strong force gauge bosons: gluons. 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 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).

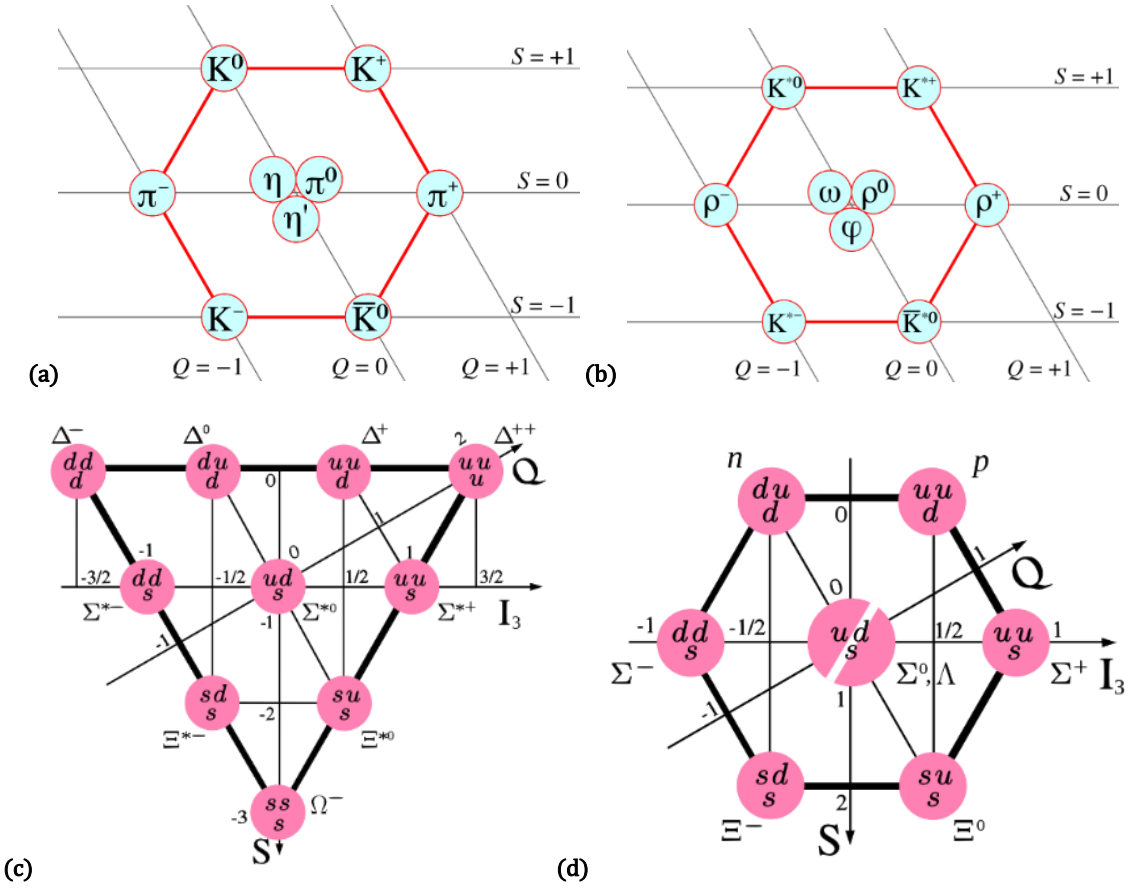


Figure 1: Mesons: (a) spin-1 nonet [10], (b) spin-0 nonet [11]; and Baryons: (c) spin-3/2 uds decuplet [12], (d) spin-1/2 uds octet [13]

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^+, π^0
Antiparticle	e^+	π^+: π^- π^0: *self*
Mass	0.5109989461 MeV [14]	139.57061 MeV [14] 134.9766 MeV [15]
Spin	$\frac{1}{2}$	0
Electric Charge	$-1e$	π^+: $+1e$ π^-: $-1e$ π^0: $0e$
Substructure	Elementary particle No known substructure	π^+: $u\bar{d}$ π^- $d\bar{u}$ π^0: $u\bar{u}$ and $d\bar{d}$

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, for up to a microsecond after the Big Bang, the universe was composed of a deconfined state of matter, known as the Quark Gluon Plasma (QGP) [16].

Statistical mechanics describes 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 phases of matter. Matter can change phase by traversing across various phase boundaries where certain phase transitions occur [17], see Figure 2 for an illustration of this process.

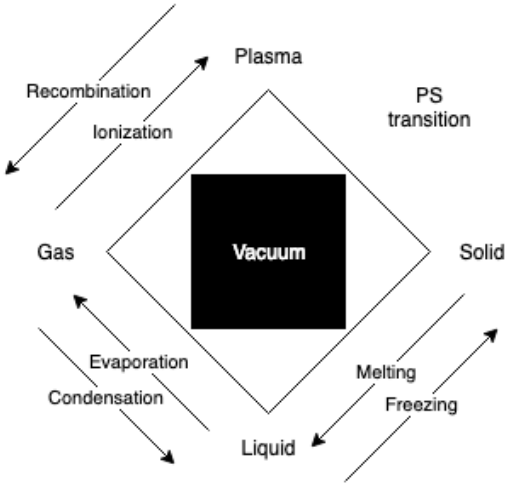


Figure 2: Simplified diagram of classical states of matter and transitions between them, with the Vacuum added as a fifth element, providing the space in which matter exists, reproduced and modified from [17].

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 [17].

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 [17].

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. 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 shown in orange. Figure 3 (b) gives a slightly more visual representation of

the nuclear phase transition and Figure 3 (c) is a simplified diagrammatic overview of the current understanding of the evolution of the Universe. The QGP is understood to have been abundant in the early universe up to the quark-hadron transition shown at $1\mu s$.

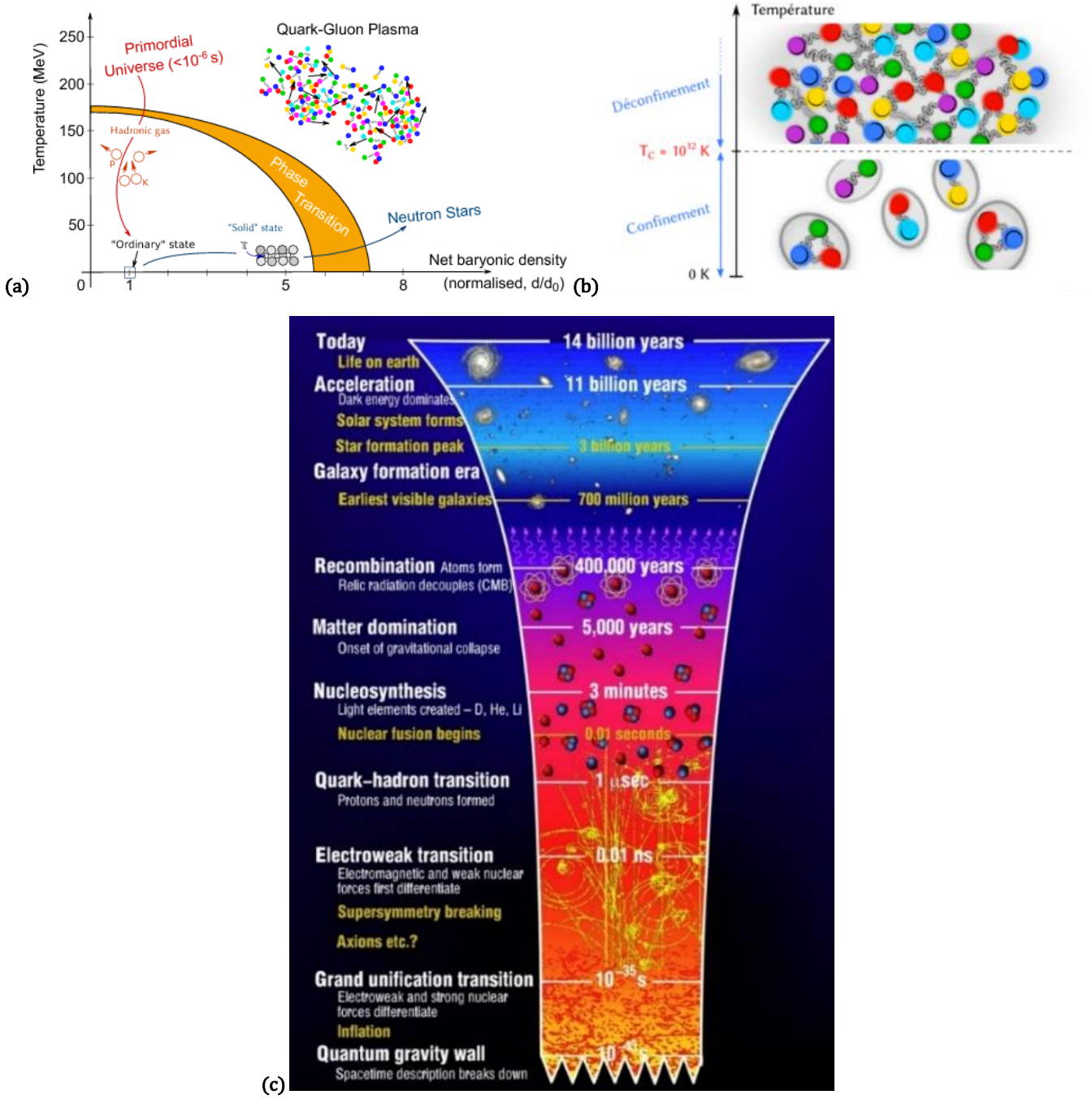


Figure 3: (a) Phase diagram of hadronic matter [18], (b) A simplified phase diagram of the nuclear phase transition along the temperature axis at low baryochemical potential (μ_B) [19], (c) The evolution of the Universe, from the Big Bang to Modern Day [20]

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 $T > 10^{19}$ GeV, a temperature so high that the principles of general relativity do not apply, and which cannot be understood with present-day physical theory [21].

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 ≃ 150 MeV, a phenomenon thought to be caused by a change in the vacuum properties of the extremely hot early Universe [16].

To understand how matter was formed in the early Universe, heavy-ion collisions, such as the Lead-Lead (PbPb) proton-lead (pPb) collisions performed at ALICE, result in a minuscule 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 various hadrons are formed and leave traces in the ALICE detector material, giving physicists an indication of how matter arose as the early Universe rapidly cooled down [16].

Since the QGP cannot be detected directly, it is studied via its decay products. Accurately distinguishing between electrons and pions produced in the abovementioned high-energy physics experiments conducted at ALICE 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 Council 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 [22].

Today, CERN (now the European *Organization* for Nuclear Research) 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 [23].

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 the development of future generations of scientists and engineers [23].

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 [23], with a circumference of ~27km and a centre of mass energy of $E_{CM} = 13\ TeV$ [24].

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 4 (d).

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 [25].

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 [26].

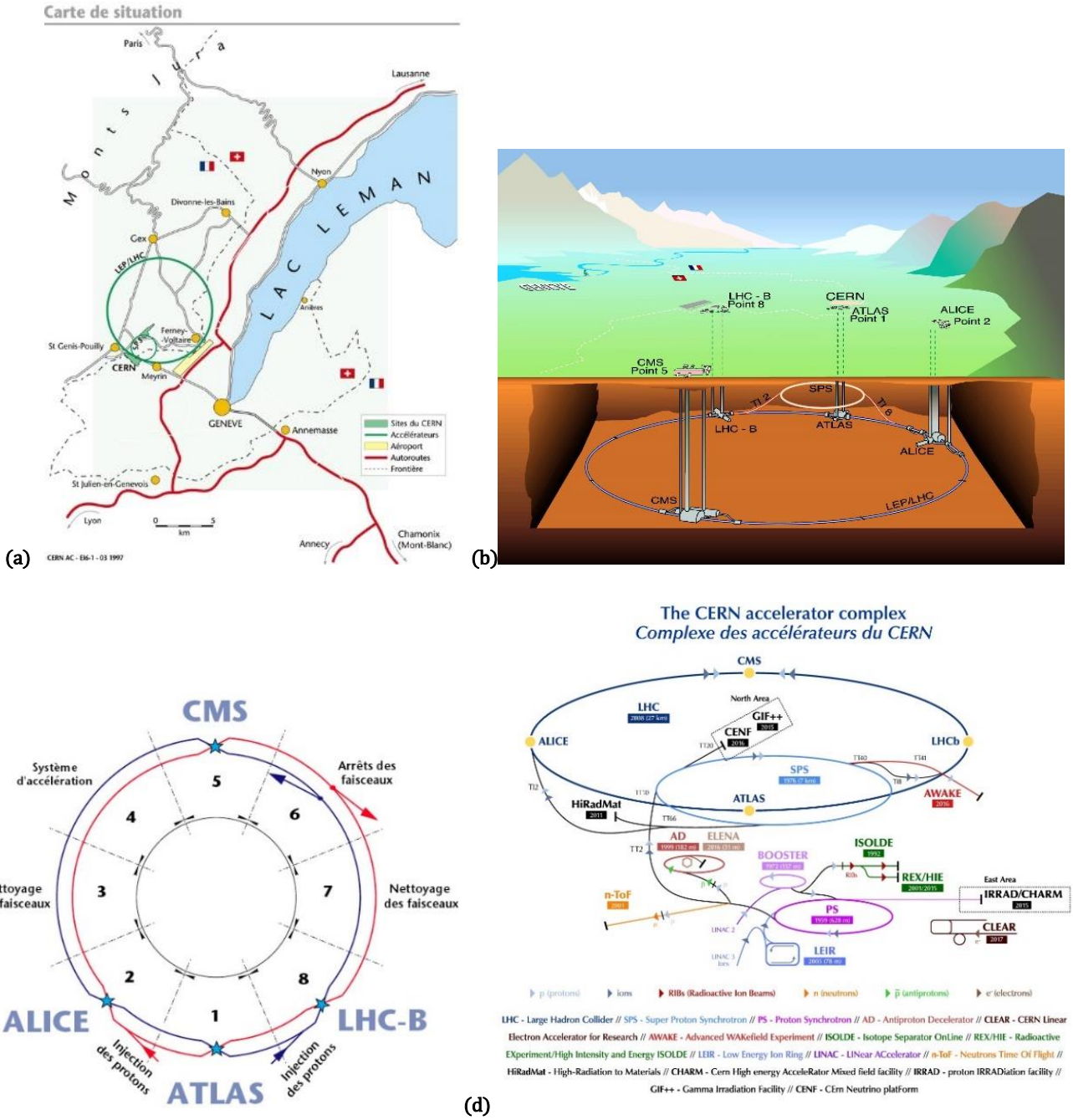


Figure 4: CERN's LHC facilities (a) in geographical context [27], (b) as a 3D diagram illustrating the depth of the underground system [28], (c) a diagram showing the LHC's proton injection points and the location of the main experiments' collision points [29], (d) The CERN accelerator complex [30].

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 [31].

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 superconducting cables have zero resistance; this allows the magnetic field to reach the 8.3 T required to bend the beams around the circular LHC ring [24].

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 [24].

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 (A Toroidal LHC ApparatuS) and CMS (Compact Muon Solenoid) investigate a very broad range of particle physics [32], [33]. 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 [32].

ALICE (A Large Ion Collider Experiment) and LHCb (LHC beauty) are the other two main experiments at CERN and are tasked with the discovery of specific physical phenomena [34]. ALICE focuses on the extreme energy densities present during heavy-ion collisions, which leads to the production of the QGP [35]. LHCb investigates subtle distinguishing nuances in the matter-antimatter dichotomy, as evidenced by attributes of the beauty quark [36]. 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 PbPb 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 [37]. 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 [37].

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 [37].

A simplified diagram of the ALICE detector system is illustrated in Figure 5. The detector weighs 10,000 tonnes and has spatial dimensions 26m × 16m × 16m [35].

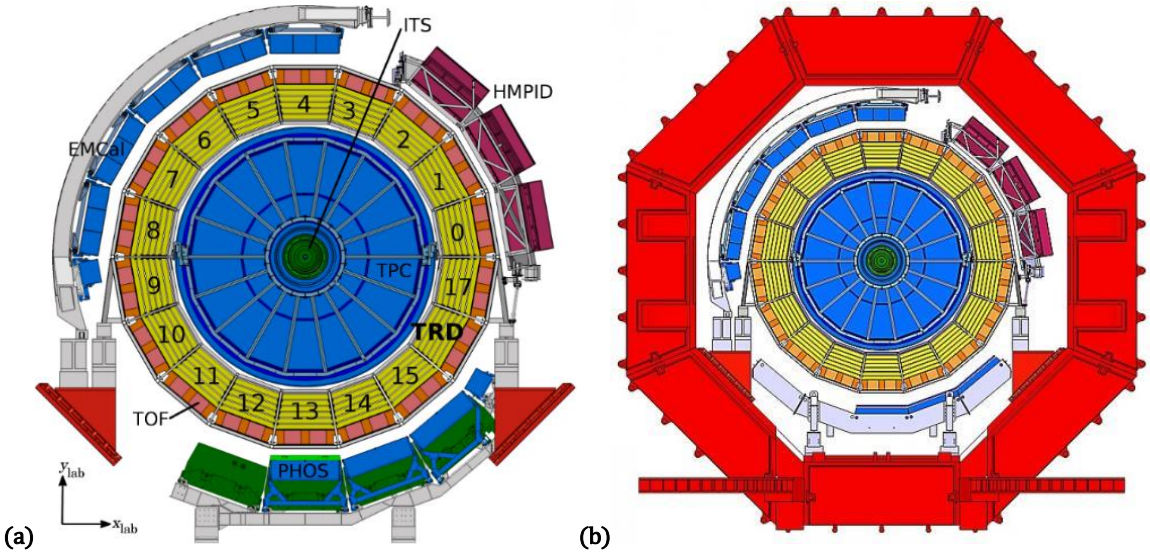


Figure 5: (a) A schematic cross-section of the ALICE detector, with the TRD shown in yellow and its 18 sectors in azimuthal angle numbered, as well as the location of other ALICE detector components: ITS, TPC, HMPID, EmCal, TOF and PHOS [38], (b) cross-section of the ALICE detector shown in context of the surrounding magnet structure [39].

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 [37].

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 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) [37].

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 [40]. 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 [40].

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 [41].

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 [42].

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). In sectors 13-15 of the TRD, the chambers in the middle stack were omitted from installation, in order to reduce the amount of material in front of the PHOS detector. So, in total, there are $(18 \times 5 \times 6) - (3 \times 6) = 522$ individual detector elements in the TRD [42], at a radial distance of 2.9 – 3.7 m from the beam axis [41]. The TRD is shown in the context of the full ALICE detector in Figure 5 (highlighted in yellow), illustrating its 18 subdivisions in ϕ-direction, as well as the six-layer architecture of each of these sectors. Figure 6 gives additional detail about the TRD's design: the hierarchical multi-component structure of the TRD detector is shown in Figure 6 (c); the 5 subdivisions of a TRD supermodule along the z-axis, as well as its six stacked layers, is shown in Figure 6 (a); and the slightly angled design of a supermodule cut in ϕ-direction resulting in slightly wider top layers is shown in Figure 6 (b).

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 [42].

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 7, 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

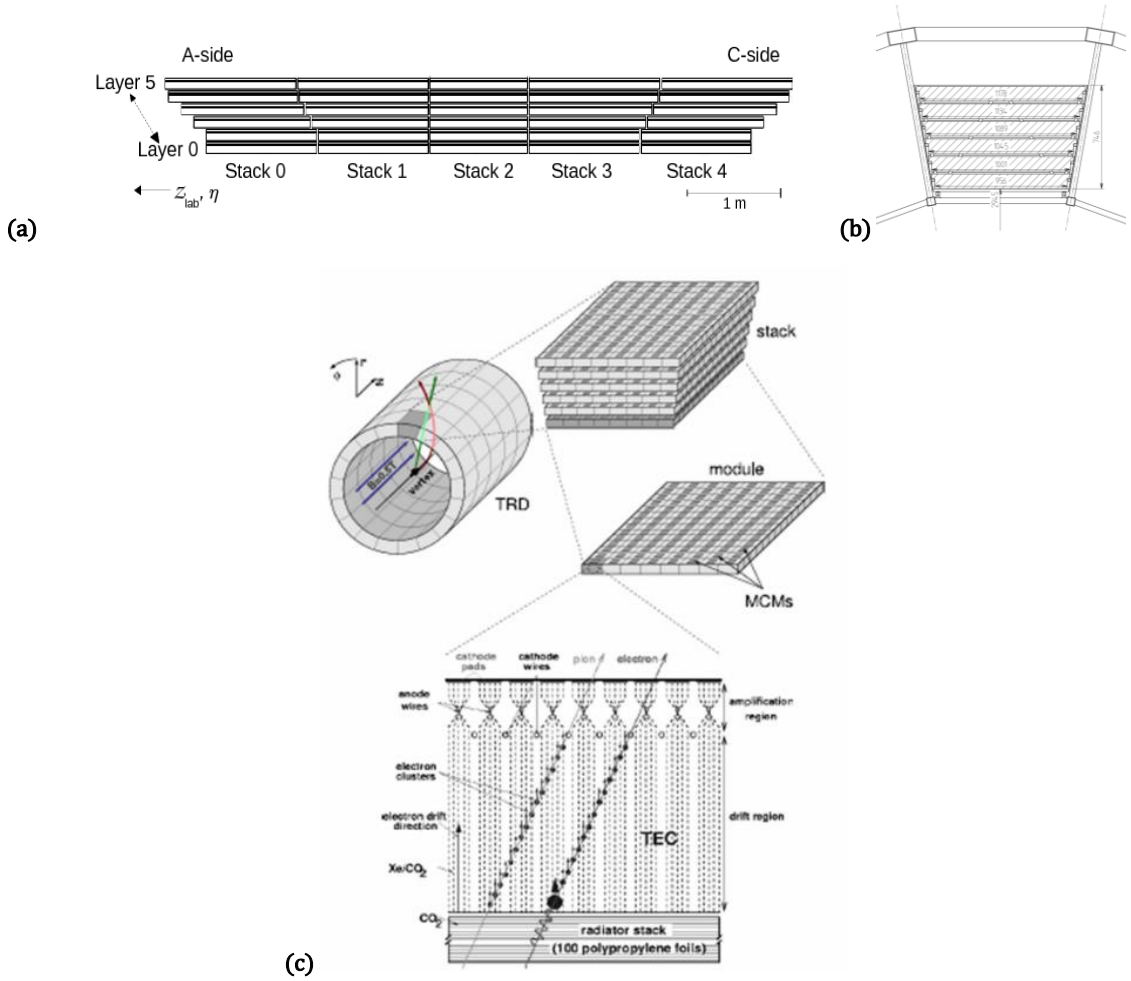


Figure 6: (a) Schematic cross-section (longitudinal view) of a TRD supermodule [38], (b) TRD supermodule cut in ϕ-direction [38] and (c) the hierarchical substructure of the TRD showing how each of the 18 TRD supermodules are composed of 5 stacks of 6 read-out chambers (ROCs) with Multi-chip modules (MCMs) that are involved in digitizing and collecting data from multi-wire proportional chambers (MWPCs) [43]

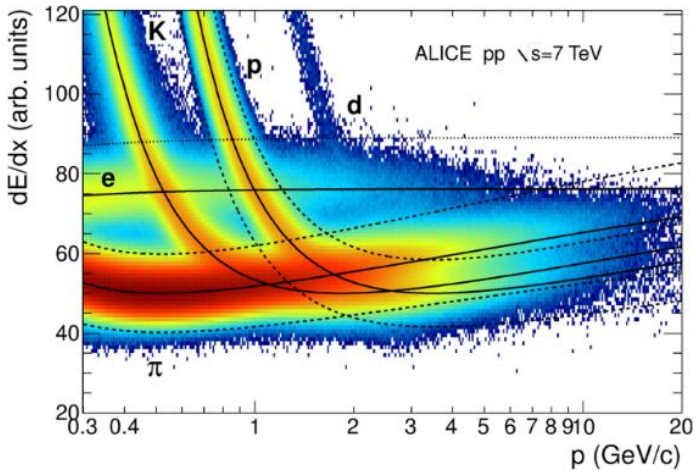


Figure 7: Bethe-Bloch curve for various subatomic particles as measured by the ALICE TPC at $\sqrt{s} = 7TeV$ [44]

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 [45].

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 [38].

The drift time of gas particles within the MWPC provides fine-grained positional information about where the particle passed through the radiator. The detected signal takes the form of charged gas molecules (ionized via interaction with charged 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 8. The positive ions produced during the avalanche process move toward the surrounding electrodes and induce a positive charge on the pad plane and it is this signal which is converted by Analog to Digital Converters in the Multi-chip modules to the TRD digits data, which was studied during this project.

Determining precisely the location of the avalanche in azimuthal angle requires that the induced charge be shared by several readout pads on the pad plane, as shown in Figure 8. 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 based on this data. 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) situated on Multi-chip-modules (MCMs, shown in context of the TRD geometry in Figure 6(c)) assists the integrated ALICE triggering system via tracklet search and electron candidate identification (the TRD trigger generates a level-1 accept (L1A) which has to occur on a timescale of 6μs) [42]; the FEE also identifies the TR signal, while providing tracking-, momentum- and mass reconstruction capability [42].

Figure 9 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. A total of 18 PASA circuits are located on a single analog chip.

2. A 10 MHz Analog to Digital Converter (ADC) converts the analog input it receives from the PASA to the tracklet preprocessor
3. 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 determine candidate tracklets.

Finally, the Gated Tracking Unit (GTU), which is not part of the FEE, but part of the readout electronics, receives data from the FEE for triggering and readout purposes.

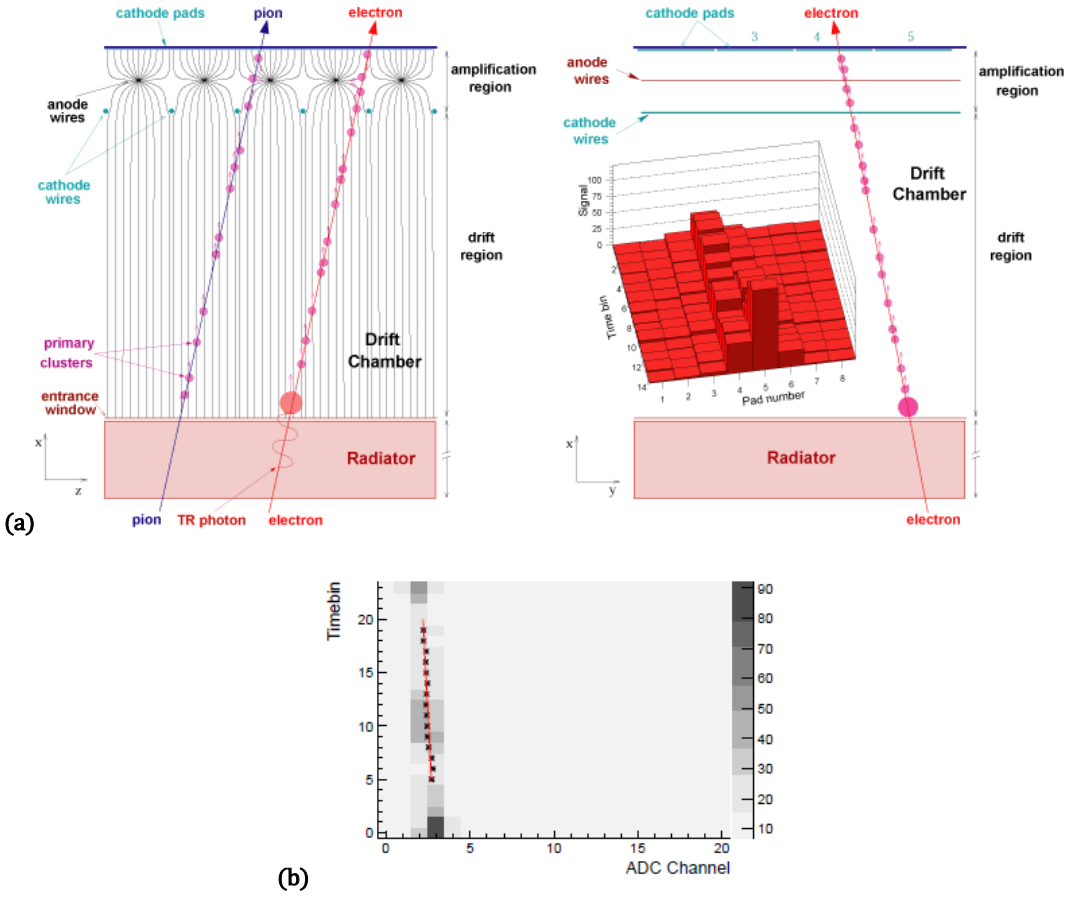


Figure 8: (a) A schematic representation of the components in an MWPC module [46], (b) Local tracking in one Multi-Chip Module (MCM, discussed in 2.3.3.1.1.3), which processes the ADC data from 21 channels and fits a straight line through the clusters that it finds (asterisks), overlaid on the raw ADC digits data generated [47].

It should also be noted that the following data filtering steps occur as part of a digital filter chain implemented on the TRD FEE: The pedestal of the signal is equilibrated, local gain variations are corrected for by a gain filter and ion tails are suppressed by a tail cancellation filter. Additional calibration of TRD data is discussed in the next section.

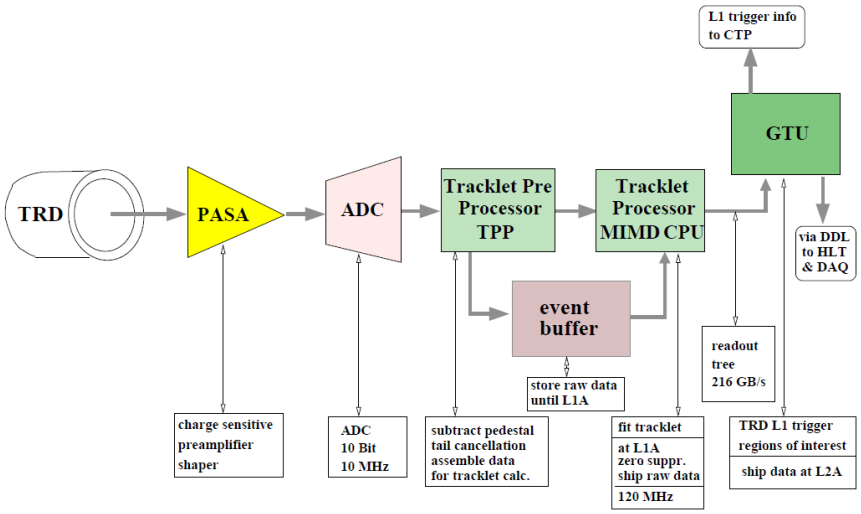


Figure 9: Diagrammatic representation of the logical components of the TRD front-end electronics [42].

2.3.3.1.1.4 TRD Data Calibration

There are four basic parameters involved in the calibration of TRD data (shown in Figure 10 at the hand of the chamber cross-section and the average pulse height plot for pions):

The basic calibration parameters are defined as follows:

Time offset: The peak in the average pulse height plot at $0.5\mu s$ (as shown in Figure 10) corresponds to charges from both sides of the anode wires. The position of this anode peak provides the time offset parameter, which is influenced by the distance from the anode wires and other delays, such as the time delay between when a collision occurs and the time the trigger signal reaches the TRD

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

Gain: Gain is proportional to the integral of the pulse-height over time (at higher drift velocity there would be shorter, but higher pulses, at the same gain)

Noise: Pad noise can be estimated from the standard deviation of fluctuations around 0, cf. the bottom left of the average pulse height diagram in Figure 10, i.e. before the initial peak

Lorentz angle: (Not seen in Figure 10). The presence of a magnetic 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.

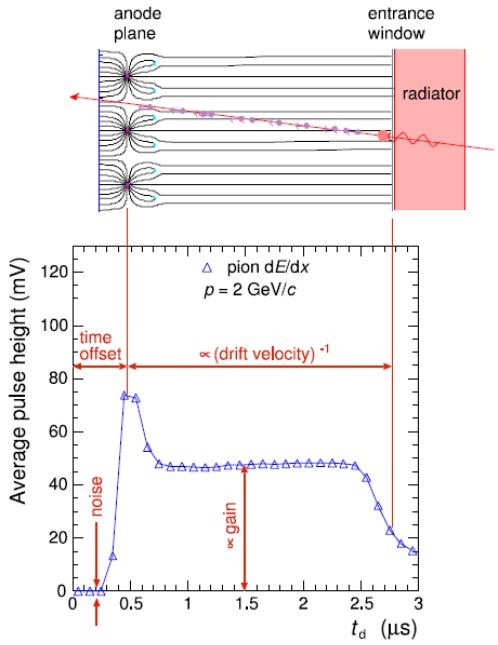


Figure 10: Main TRD signal calibration parameters, indicated as annotations on a plot of the average pulse height as a function of drift time for pions, in addition, the chamber cross-section is shown at the top for better understanding [38]

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

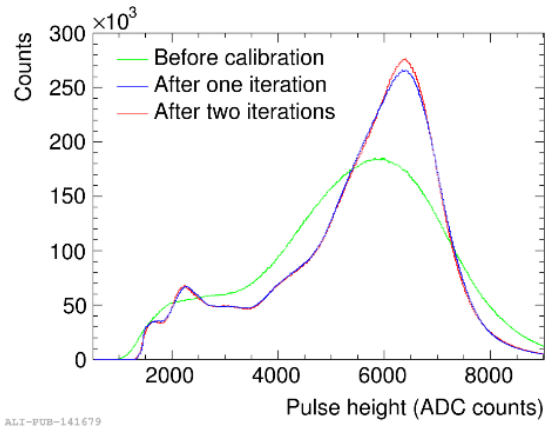


Figure 11: Pulse height spectrum before the Kr-based calibration, after one and after two iterations (calibrations performed in consecutive years) for one TRD read-out chamber [38]

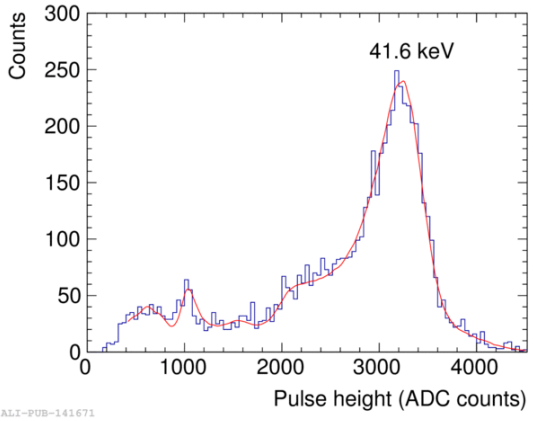


Figure 12: Pulse height spectrum accumulated for one pad during the Kr-calibration run. The smooth solid line represents the fit from which the gain is extracted [38]

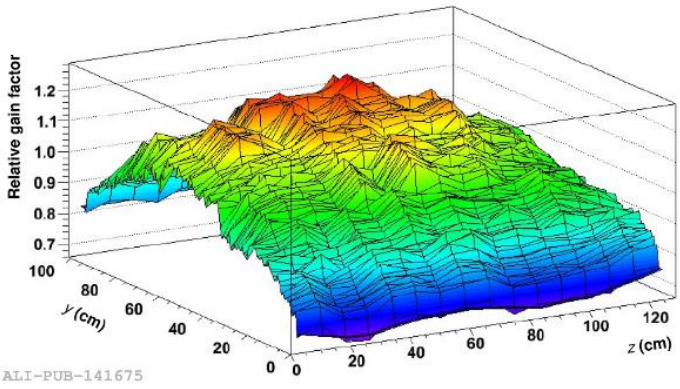


Figure 13: Relative pad gains for one chamber calibrated with electrons from Kr decays [38]

2.3.3.1.1.5 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 ($^{\mathrm{d}E}/_{\mathrm{d}x}$) to the signal, from the contribution made by Transition Radiation photons and is, therefore, an important factor in distinguishing between electrons and pions [38]. The electron identification capability is also used to trigger at level 1 [41].

Figure 14 shows the time evolution of the TRD 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.

Also evident from Figure 14 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 travelling 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 14, since it will be absorbed preferentially close to the radiator, adding its signal to the ionization energy of the track [41].

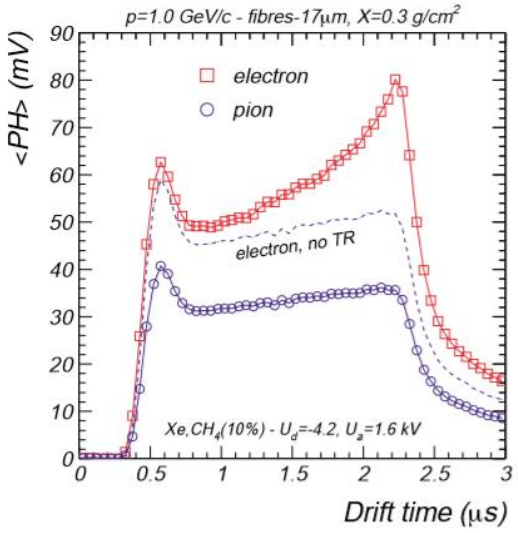


Figure 14: Average pulse height as a function of drift time for electrons and pions (both at $p = 1\mathrm{GeV}/c$) [42].

It should be emphasized that Figure 14 shows the *average* pulse height over time. In truth, there are large fluctuations around this average, as can be seen in Figure 35.

2.3.4 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. The 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)

2.3.4.1 Likelihood Methods

The concepts of Likelihood and Maximum Likelihood Estimation are discussed in Section 3.1.3.

2.3.4.1.1 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 15 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.

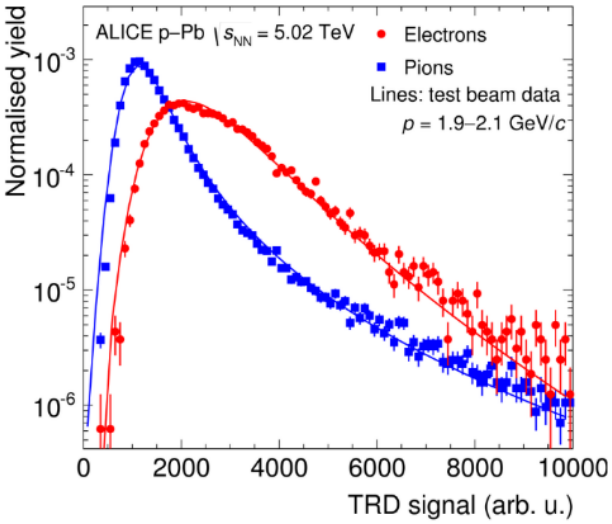


Figure 15: Total integrated charge, normalised to tracklet length, measured in a single read-out chamber for electrons and pions in pPb collisions at $\sqrt{s_{NN}}$ ≈5.02TeV. Test beam measurements were scaled by a common factor to compensate for gain differences [38].

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 $K_s^0 \rightarrow \pi^+\pi^-$ [41]. V_0 candidates are reconstructed using a secondary vertex finder algorithm [48]. More information about obtaining clean reference data for particle identification can be found in [49].

2.3.4.1.2 Two-, three- and seven-dimensional Likelihood (LQ2D, LQ3D, LQ7D)

Two-, three- and seven-dimensional likelihood methods each take the temporal evolution of the signal (Figure 14) 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. An assumption is made that the signals from different slices are statistically independent.

2.3.4.2 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 by the TRD working group at ALICE was trained using a similar approach as LQ7D, i.e. splitting the TRD signal 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 ([49], [50], [51]) are summarised in section 4.2

2.3.4.3 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 and is not generally used for distinguishing electrons from pions). 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.

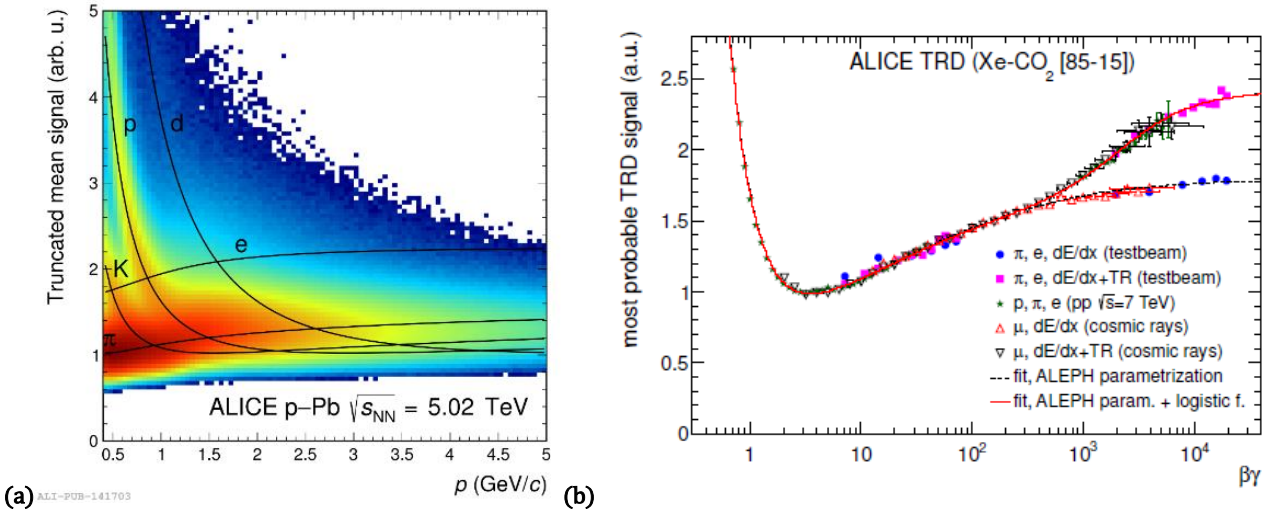


Figure 16: (a) Truncated mean signal as a function of momentum for p-Pb collisions at $\sqrt{s_{NN}} = 5.02$ TeV. The solid lines represent the expected signals for various particle species [38], (b) The most probable values of TRD signals as a function of $\beta\gamma$, clearly showing the distinguishing transition radiation signal from electrons at higher $\beta\gamma$, which is mostly lost during the truncation procedure for the Truncated Mean method [52].

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.

2.3.5 Particle Identification Accuracy

To calculate the accuracy of the abovementioned methods, clean reference samples were used. The separating power of these approaches is 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) [41].

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 (Figure 17 (c)), and that electron efficiency is sometimes sacrificed during analysis to obtain a more pure sample (Figure 17 (b)) [41].

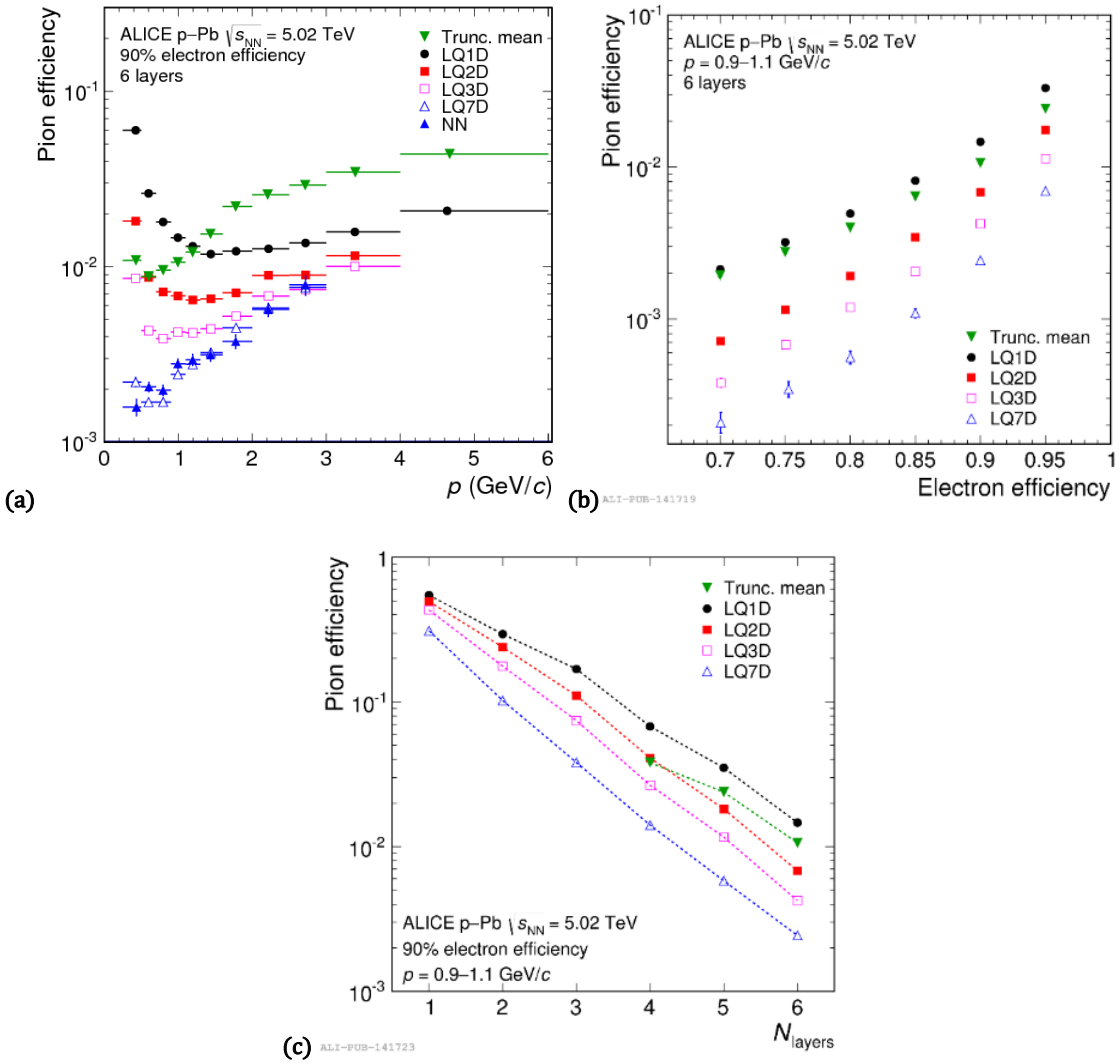


Figure 17: (a) Particle identification performance of the TRD, based on various methods discussed. (b) Pion efficiency as a function of electron efficiency. (c) Pion efficiency as a function of the number of tracklets obtained [38].

Figure 17 (a) shows how pion efficiency depends on momentum for the six 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, especially at high momenta, since transition radiation with its attendant high charge deposition is more likely to be removed during the truncation procedure [41].

It is also clear from this plot that the misidentification of pions as electrons (False Positive Rate) is reduced substantially by the LQ7D and Neural Network techniques, compared to truncated mean-, LQ1D-, LQ2D- and LQ3D- methods, and that the temporal evolution of the signal (exploited at higher granularity for the more accurate analysis methods) is, therefore, a highly informative feature for particle identification [41].

2.3.5.1 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 [53].

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 [53].

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 [54], the version of ROOT developed and used for LHC run 1 and run 2; and ROOT 6 [55], the version of ROOT developed for LHC run 3, scheduled to start in 2021 after the second long shut down period (LS2).

2.3.5.1.1 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 enable 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 [56].

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.

2.3.5.1.2 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.

2.3.5.2 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 [57].

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 usually 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:

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

Equation 3

[58].

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.

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

Equation 4

Here, $P(A)$ is referred to as the prior probability (or evidence) and $P(A|B)$ is referred to the posterior probability (or likelihood, explained in slightly more detail in the next section, 3.1.3), i.e. we can adjust our estimate for $P(A)$, based on other evidence at our disposal.

[58].

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. cf. Section 3.1.2, we are referring to the posterior probability of the parameters, given the data.

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

Equation 5

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:

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

Equation 6

[58].

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 [59]. 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 agree 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 (combined using a Bayesian approach for up to 6 tracklets pertaining to a single track, as explained in Section 3.1.7) 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 18 illustrates how setting a threshold value for the test statistic, i.e. t_{cut}, results in rejection of the null hypothesis when $t > t_{cut}$.

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.

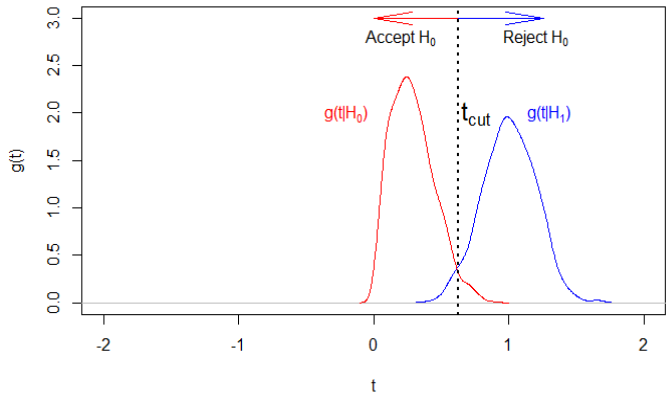


Figure 18: An illustration of rejection or acceptance of the null hypothesis, under the assumed distributions of H_0 and H_1, when t falls in the critical region $t > t_{cut}$

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:

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

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 of 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 18, 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: Summary of Statistical Errors

	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. Θ_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:

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

Equation 9

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 likelihood 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" (e) as signal and "pion" (π) 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.

In order to give more power to the test statistic (cf. Section 3.1.6), we can combine the probability of each of up to 6 tracklets (pertaining to a single track) of being an electron (obtained from each of up to 6 detector layers in the TRD), based on a Bayesian approach outlined in the formula below to calculate the probability for the full track:

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

Equation 10

Here, $P_j(elec)$ is the probability of a single tracklet being an electron, obtained from layer j of the TRD, 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 11

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

Equation 12

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-based 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 [60].

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, needs to be 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 raw pixel data from 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 [60], this could involve tasks 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 that input feature set; usually, this representation learning process will involve the algorithm identifying the major factors of variation which effectively explain the observed data and discarding those which are not useful to the algorithm [60].

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 19), 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 [60]. In other words, an initial 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 (Supervised ML tasks are usually framed

broadly as classification or regression, while Unsupervised ML tasks are concerned with finding hidden patterns in the data without explicitly predicting a label or a real-valued outcome regarding that data).

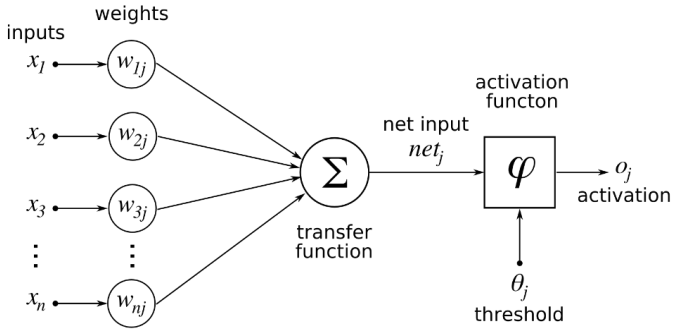


Figure 19 Schematic of a single neuron, with inputs multiplied by weights, a bias term is added to the summation in the transfer function (not shown) and this net input is then passed through a non-linear activation function, φ [61]

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 the desired response, y. Feedforward neural networks have one-way information flow from input features to output, whereas recurrent neural networks have feedback connections [60].

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

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

Equation 13

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 [62].

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 [60].

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 neuron in each f_a in the chain of functions, and b is a real-valued bias term, which is essentially an intercept term for each neuron in each f_a [60].

It is easy to see that chaining such a set of linear models without applying nonlinear transformations (denoted as $\varphi(f_a(x))$) to what is 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 [60]. 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 follows: $\varphi(w^T x + b) = \max(w^T x + b, 0)$ is often the first introduced and easiest to understand (see Figure 20). For a more advanced overview of various activations and their performance, please see [63]. 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.

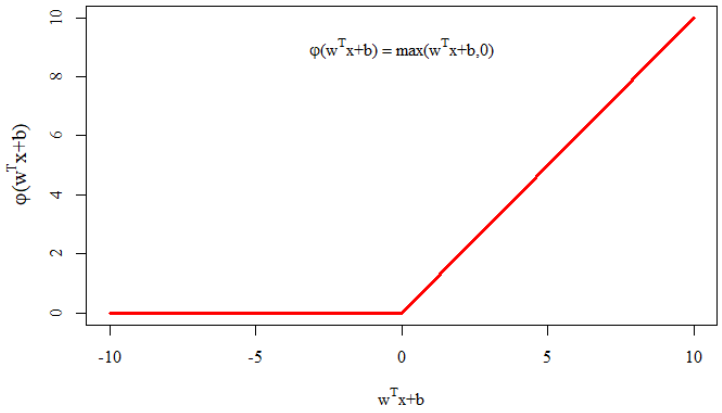


Figure 20: ReLU activation function

Combining the concepts explained above gives us a representation for a single hidden layer in an ANN as follows (Equation 14, where W is a matrix and $\boldsymbol{x}$ and $\boldsymbol{b}$ are vectors):

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

Equation 14

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 15

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, $\boldsymbol{h}_{2,\dots,n}$. 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 $f^*(x)$, i.e. where the neural network $f(x)$'s output is $\hat{y} \approx f^*(x)$; where $f^*(x) = y$ [60].

Figure 21 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.

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. [60]

3.3.1.1 Loss Functions

Neural networks are optimised by making use 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 designed for specific use-cases can be found in [60]. The two loss functions that were predominantly used for particle identification in this project are discussed below.

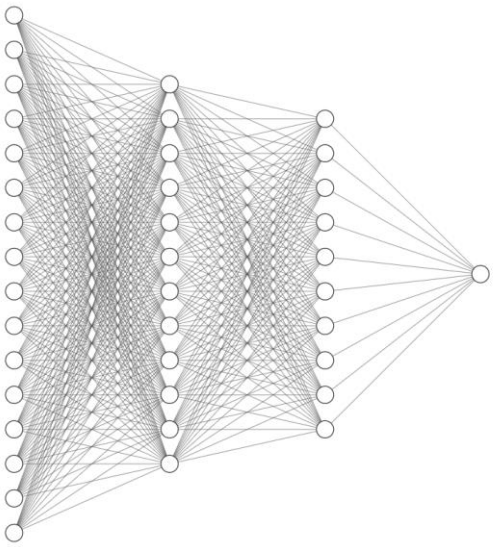


Figure 21: Schematic depiction of a fully-connected feedforward neural network (16:12:10:1), i.e. with an input layer with 16 neurons (nodes) 2 hidden layers, with 12 and 10 neurons, respectively and a single-node output layer, with the weights between two neurons depicted as connecting lines between them [64].

3.3.1.1.1 Binary Cross-entropy

Binary cross-entropy is defined as:

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

Equation 16

Here, $\hat{y}$ is the model's estimate for the probability of an observation being of a particular class y, i references a specific observation in the training dataset and n indicates the total number of observations [60]. Figure 22 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 16).

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 (one-hot encoding is a technique which allows one to assign a numeric value to a categorical datatype, by arbitrarily, but exclusively, assigning either a value of 0, or a value of 1 to each class: here we will use $e = 1, \pi = 0$), 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.

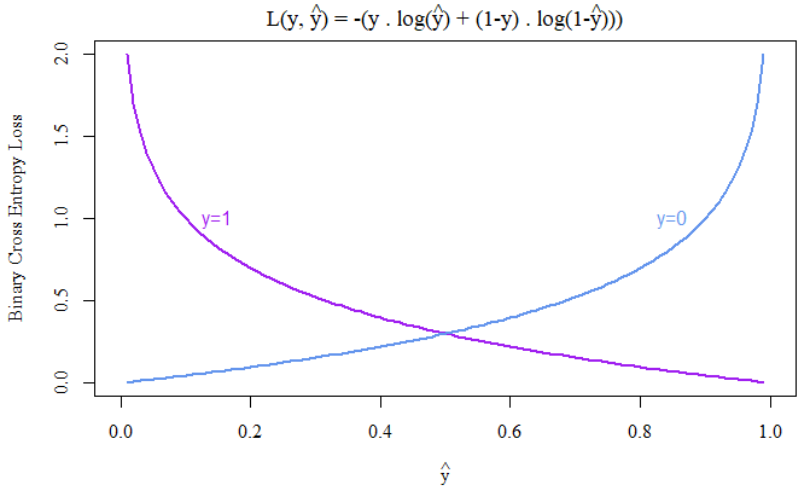


Figure 22: Illustration of the descent towards zero, of the Binary Cross Entropy Loss Function as ŷ, or $p_{model}(y|x)$, approaches the true y.

3.3.1.1.2 Focal Loss

Focal Loss was proposed by [65] 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:

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

Equation 17

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

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

Equation 18

Focal loss simply 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:

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

Equation 19

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

Equation 20

The authors suggest using $\gamma = 2$ [65], 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 23 shows how the Focal loss function decreases steeply as classification probability approaches the true class level (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.

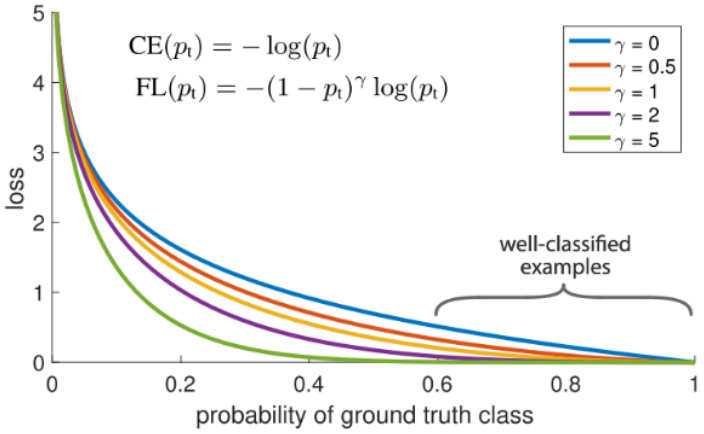


Figure 23: Focal Loss where the true class is 1 [65].

Focal loss is not a default loss function in Keras, but it has been implemented as a custom loss function here [66] for Python. The author of this thesis subsequently adapted this for use in Keras with R here [67], 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 [60]:

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

Equation 21

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 $x_{i,\ldots,n}$ through the MLP, assessing the output, $\hat{y}$, according to a loss function, J, and individually adjusting each of the mapping functions $f_a^{j,\ldots,m}$ 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 $\frac{\partial J_k}{\partial f_a^j}$. [60], until a (hopefully global) minimum is achieved [60]. Note that the gradient of the loss function will be $\nabla = 0$ when either a local or global minimum is reached (shown in Figure 24).

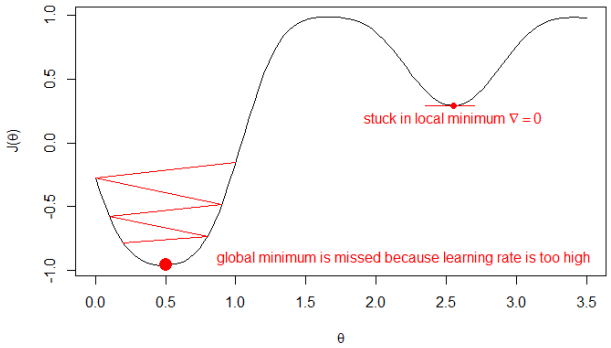


Figure 24: Pitfalls faced during optimization

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 24 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 any other trainable parameter θ_i at iteration (or epoch) , k, is as follows:

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

Equation 22

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 24 and others [60].

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 network with 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 $\boldsymbol{a}^{(k)}$ in each layer, $\boldsymbol{k}$are calculated as follows

Compute the gradient of the loss function on the output layer:

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

for: $\boldsymbol{k} = \boldsymbol{l}, \boldsymbol{l}-\mathbf{1}, \ldots, \boldsymbol{l}$, **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 [68].

Stochastic 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 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 additional hyperparameter $\alpha \in [0,1)$ determines the rate of exponential decay; updating θ using momentum is done as follows:

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

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

Equation 23

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 [68].

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 [60].

Regularization strategies can entail, 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 [60].

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 and manipulate successfully 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 [60].

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 [60].

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 [60].

3.4 Convolutional Neural Networks

3.4.1 The Kernel Concept and Motivation for CNNs

Convolutional Neural Networks (CNNs) is 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 [60].

3.4.2 The Convolution Function

In practice, data fed to a CNN usually consists of a grid of vectors, effectively adding a depth- (or channel-) dimension to the usual width- and height- dimensions of a grid. In deep learning packages such as Tensorflow, indices of values in such a tensor specify 4 locations, i.e. each value has a row-, column-, channel- and observation index [60].

3.4.2.1 The Convolution Operation

Given an element of a 3-D input tensor $\boldsymbol{V}_{i,j,k}$ i.e. a value in the ith channel, jth row and kth column of a single training observation $\boldsymbol{V}$, which is convolved by a 4-D kernel tensor $\mathbf{K}$ to generate an element in an output tensor specified by $\boldsymbol{Z}_{i,j,k}$, then $\boldsymbol{K}_{i,j,k,l}$ represents the strength of the connection between an element in channel i of the output tensor ($\boldsymbol{Z}_{i,:,:}$) and an element in channel j of the input tensor ($\boldsymbol{V}_{j,:,:}$), offset by k rows and l columns between the input and output element. Thus $\boldsymbol{Z}_{i,j,k}$ is calculated as follows:

$$\boldsymbol{Z}_{i,j,k} = \sum_{l,m,n} \boldsymbol{V}_{l,j+m,k+n-1} \boldsymbol{K}_{i,l,m,n}$$

Equation 24

where the summation over the indices is for all valid indices in the tensor [60].

An example of a simple 2D convolution (multiplying a 3×4 matrix by a 2×2 kernel) is shown below (adapted from [60]).

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

$$=$$

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

Equation 25

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 [60]. 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 [60].

Parameter sharing allows 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 [60].

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 $g(f(x)) = f(g(x))$. 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 [60].

3.4.3 Pooling

CNN layers are generally composed of three operations, shown in Figure 25:

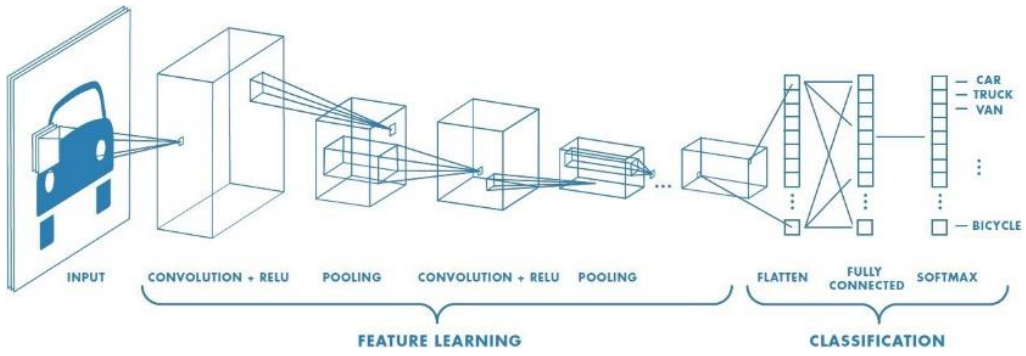


Figure 25: Simplified diagram of a convolutional neural network [69]

These three operations are executed as follows:

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

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 [60].

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

Pooling serves the purpose of ensuring 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 do 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 [60].

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 [60].

Pooling with down-sampling is achieved by reducing the number of pooling operations relative to the number of detector units, by introducing a stride greater than one. See Figure 26 for an illustration of the effect of using different stride widths with the same pool-width. It is straightforward to see that down-sampling applied in this manner will lead to fewer inputs to process, reduced memory requirements and improved statistical efficiency [60]

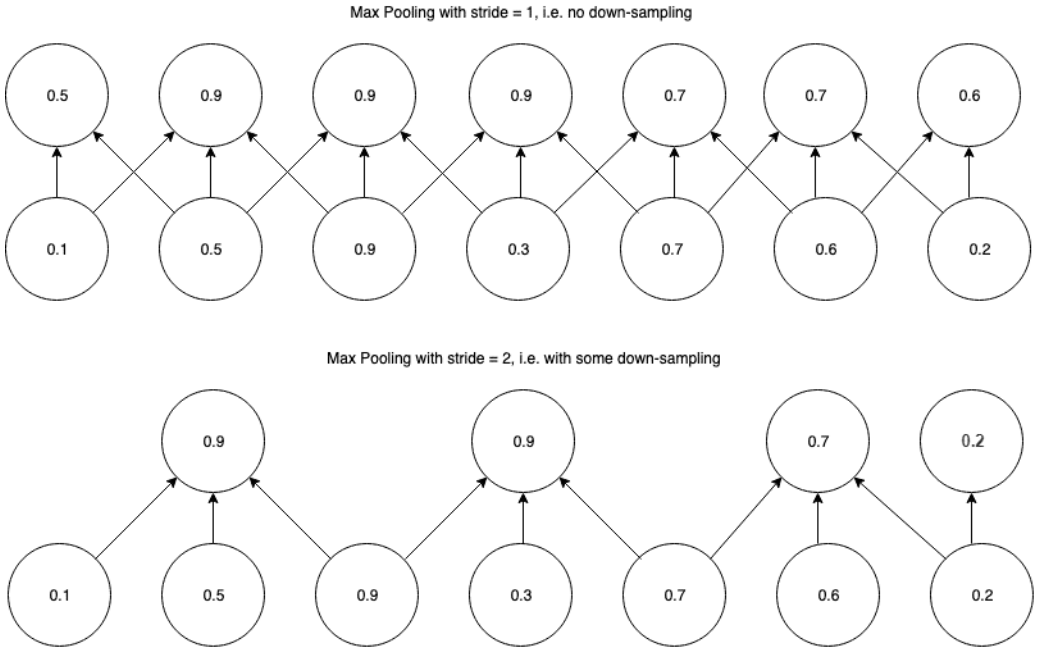


Figure 26: An illustration of the concept of max pooling, using pool-width of 3 with a stride of one (top panel) vs a stride of two (bottom panel), adapted from [60].

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 [60].

Convolutions applied with no padding of the input image are known as valid convolutions, where pixels in the output of a convolution operation 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 [60].

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 [60]. 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 fewer connections to the output image and therefore that their influence on the network as a whole will be reduced [60].

3.5 Recurrent Neural Networks

Recurrent neural networks (RNNs) are specifically designed to process sequential values. Parameter sharing is an essential aspect of RNNs and facilitate the detection of patterns that could potentially occur in more than one place in the sequence; this family of ANNs also accounts for sequences of differing length [60].

Parameter sharing in RNNs manifest in the form that each element of the output is a function of previous elements of the output within a specified range and is updated using the same rule used to update previous elements of the output [60].

The input vector to an RNN will be vectors $x^{(t)}$ with timestep t consisting of a range from 1, …, τ [60].

Recurrent neural networks extend the concept of a computational graph to include cyclical connections, where the present value of a variable is understood to have an influence on its future value [60].

3.5.1 Computational Graphs

Computational graphs are visual depictions which formalize a set of operations applied to an input vector, for example, the computational graph formalizing the ReLU activated output of a hidden unit, i.e. $H = max\{0, WX + b\}$, would look as follows:

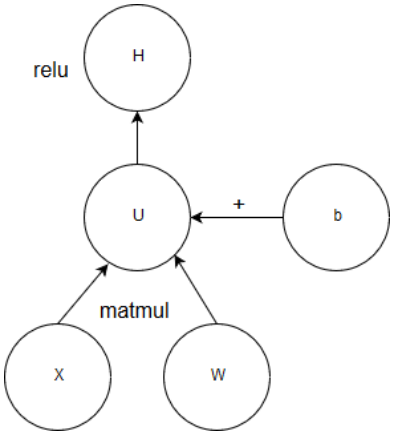


Figure 27: ReLU activated hidden unit in a Neural Network depicted as a computational graph, adapted from [60].

In RNNs, computational graphs manifest as repetitive chains of operations which result in parameter sharing across neural network architectures [60].

As an example, a dynamical system is classically expressed as:

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

Equation 26

Here, the state of the system at time t, i.e. $s^{(t)}$, explicitly depends on its state at the previous time step (t-1).

This graph can be unfolded for a finite number of timesteps, τ, by applying the above expression $\tau - 1$ times, e.g. if τ=3:

$$s^{(3)} = f\left(s^{(2)}; \theta\right)$$

$$= f\left(f\left(s^{(1)}; \theta\right); \theta\right)$$

Equation 27

The above equation can be represented as an acyclic graph, which does not make use of recurrence, as follows:

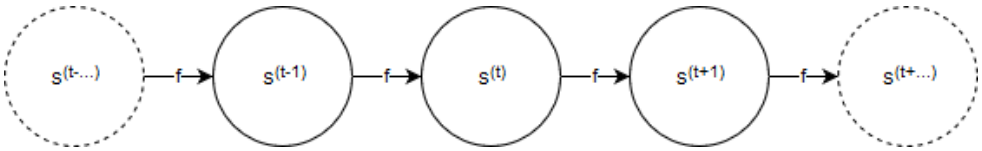


Figure 28: Acyclic computational graph of a dynamical system, adapted from [60].

If we extend this to express the dynamical system's state at any point being informed by all the previous states of the system, the equation becomes:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

Equation 28

This is the basic formula upon which RNNs are built, where the "states" of the system are the neural network's hidden units, i.e.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

Equation 29

[60].

3.5.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) recurrent neural networks are a highly successful class of RNN which deals with the problem of exploding or vanishing gradients introduced by other RNN implementations by enforcing constant error flow through the internal states of special units, called memory cells, shown in a computational graph in Figure 29.

Multiplicative input and output gates protect other units from irrelevant inputs and currently irrelevant stored memory states, respectively. Each memory cell as shown above consists of a central linear unit with a self-connection which is fixed. In this way, a memory cell can decide whether or not to save information about its current state, based on inputs from other memory cells.

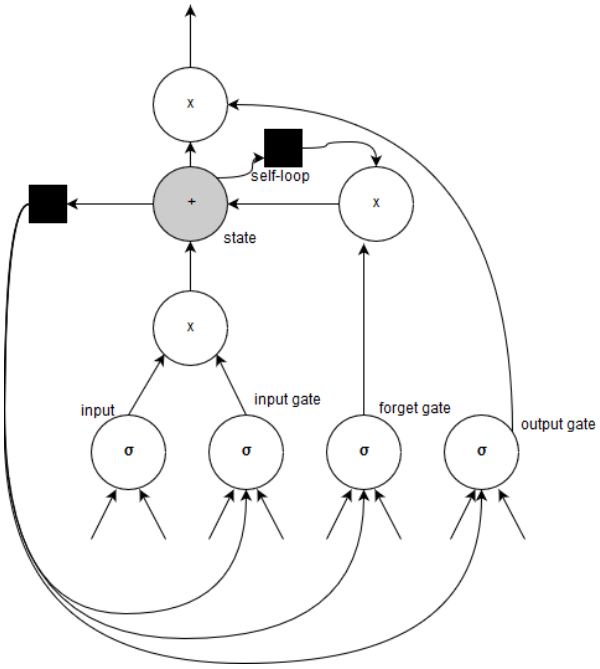


Figure 29: Graph-based representation of special LSTM units, adapted from [60].

4 IMPLEMENTATION: MACHINE LEARNING FOR PARTICLE IDENTIFICATION

4.1.1 Data Extraction

Please see [70] 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, this process is described in detail in Appendix V and summarised below. The specific runs from LHCq16 that were analysed are listed in Appendix I.

4.1.1.1 AliROOT implementation: Data Extraction

4.1.1.1.1 Files:

Please note that the files used for data extraction have been modified by various authors. The original implementation was coded by [71] (this was based on [72]), thereafter it was modified by [73] for his Honours Thesis and again by [74] for his Third Year Project.

For this project, some additional modifications were made to the files, in order to modify the output and extract additional variables. In summary, the uncalibrated TRD digits data, as well as some additional attributes (shown below) were extracted for electrons and pions that passed certain kinematic cuts.

The following information was extracted for each track and appended to the output file, `"pythonDict.txt"`:

- LHC16q Run number
- Event number in the run
- V_0 track ID
- Track number in the event
- PDG Code
- The TPC's n_{σ^e} and n_{σ^π} estimates, which indicate how many standard deviations the track was away from the expected $^{dE}/_{dx}$ value at its momentum
- Transverse momentum (p_t)
- TPC $^{dE}/_{dx}$
- Momentum (p)
- Angular coordinates for the track: η, θ, ϕ
- The TRD digits for 17 pads surrounding the expected coordinates in each of the six layers of the chamber it passed through (expected central pad ± 8 pads in either direction); as well as:
    - the detector chamber number
    - row number and
    - column number

where it was calculated to have passed through the TRD

4.1.2 Data Structure

An example of the raw text data obtained for a single track is shown in Appendix IV. 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, as explained above (regarding the meta-information) and in Section 2.3.3.1.1.2 (regarding the structure of TRD raw digits).

In essence, for each particle, a set of up to six 2D arrays ($X_{1,\dots,k\in[1,\dots,6]} = (x^{(k)}{}_{ij}) \in \mathbb{N}^{17\times 24}$) of ADC data is obtained.

Figure 30 depicts 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 6 example images in Figure 30, the signal for 17 pads in the TRD layer was 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 (each time bin spanning $0.1\mu s$) within the pad, giving an indication of the time-evolution of the signal.

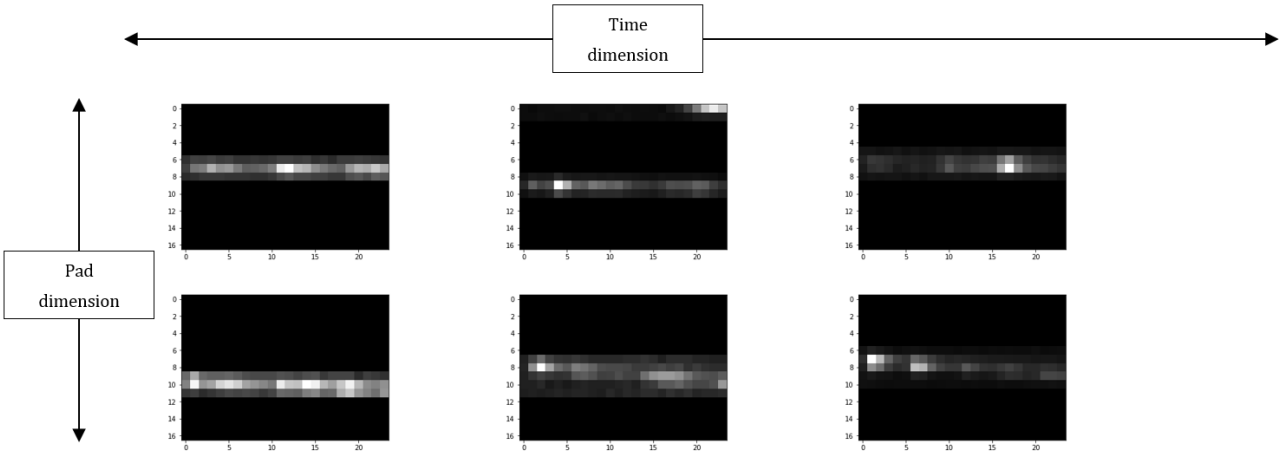


Figure 30: Six example TRD tracklets, with time-direction indicated along the x-axis and pad-direction indicated along the y-axis for each image shown

4.1.3 Data Exploration

When reading all data 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 some of the particle identification datasets used for training (i.e. subsets of the full dataset were used in some cases) and this resulted in up to an additional 14.5% of all pion tracklets and 12.6% of all electron tracklets being removed from these datasets.

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 31 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 [75].

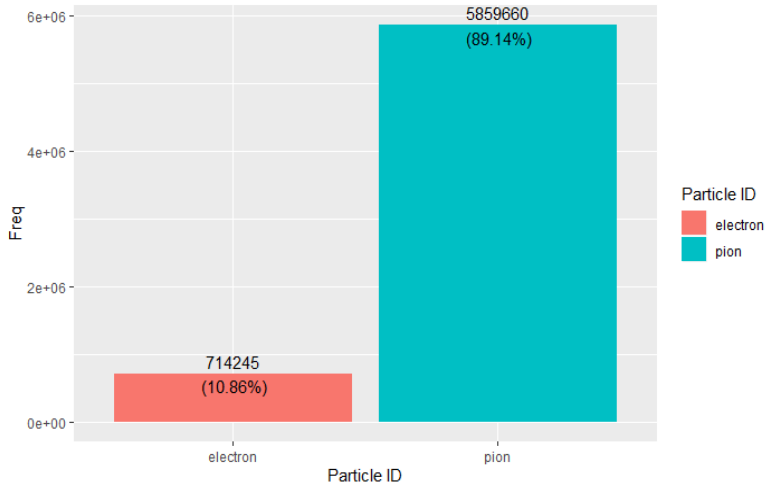


Figure 31: Number of Particles, per Particle ID, across all runs

4.1.3.2 Momentum bin counts: number of tracklets per Particle ID

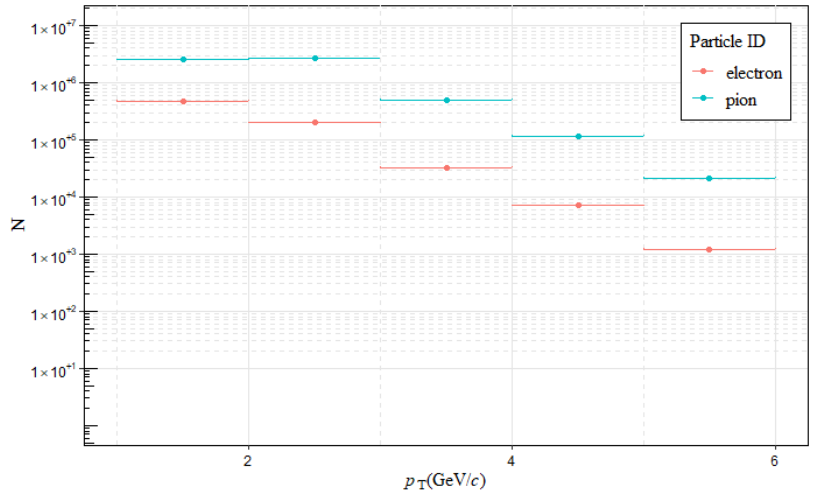
From Figure 32, one can see how this class distribution differs for particles in different momentum ranges. Particularly, there is a larger proportion of electrons in lower-momentum bins, i.e. $\boldsymbol{p \leq 2\ \mathrm{GeV}/c}$ and $\boldsymbol{2\ \mathrm{GeV}/c < p \leq 3\ \mathrm{GeV}/c}$. The proportionally higher fraction of electrons vs pions seen in the lower momentum bin is mostly because there was a physics cut made on momentum in `AliTRDdigitsExtract.cxx`, to keep only tracklets from pions in the $2GeV \leq P \leq 6GeV$ range and electrons in the 1.5 GeV/$c \leq p \leq$ 6 GeV/c range.

4.1.3.3 Characteristic Energy Loss Curves (Bethe-Bloch)

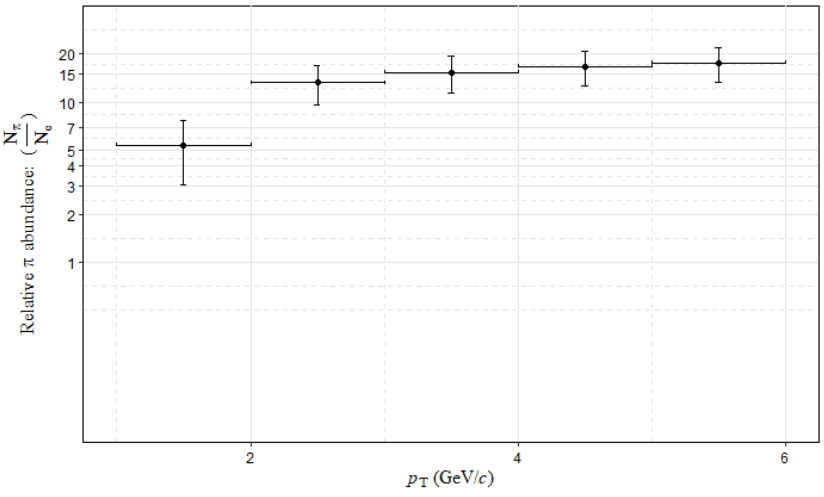
From Figure 33, the expected increased energy loss of electrons relative to pions, in the low GeV range is apparent. Note also that the "ground truth" particle IDs used in this thesis were estimated from V_0 decays, but the long tail towards low $^{dE}/_{dx}$ for the electron sample (shown in Figure 33) 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.

4.1.3.4 Average Pulse Height

Figure 34 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 Figure 35).



(a)



(b)

Figure 32 (a) Histogram of electron and pion counts per momentum bin (b) Histogram of the relative π-abundance in each momentum bin in (a).

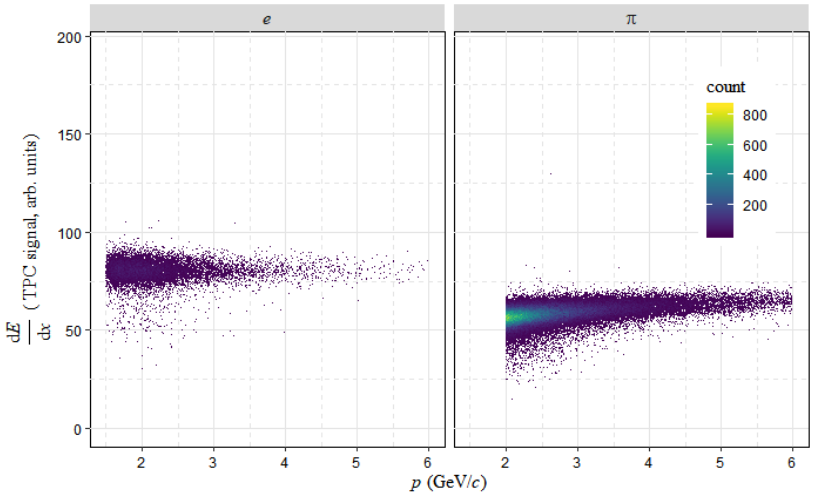


Figure 33: Energy Loss per Unit Path Length as a function of Momentum, for Electrons and Pions, as measured by the ALICE TPC

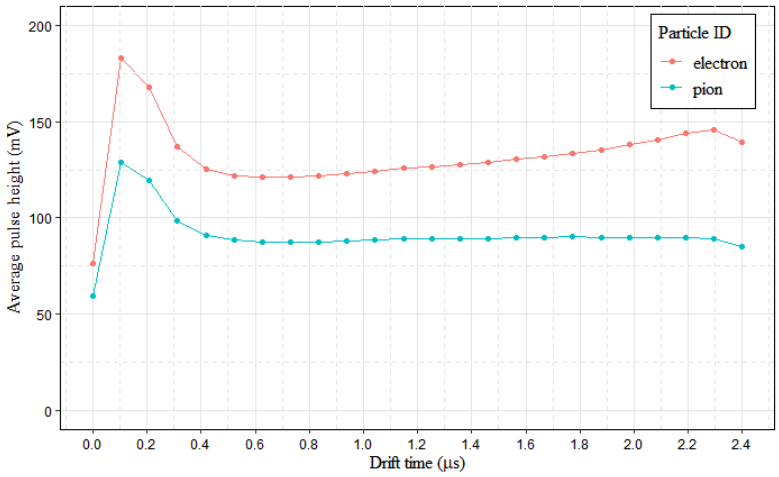


Figure 34: Time Evolution of the Average Pulse Height Signal, per Particle ID (for tracklets from the entire momentum range)

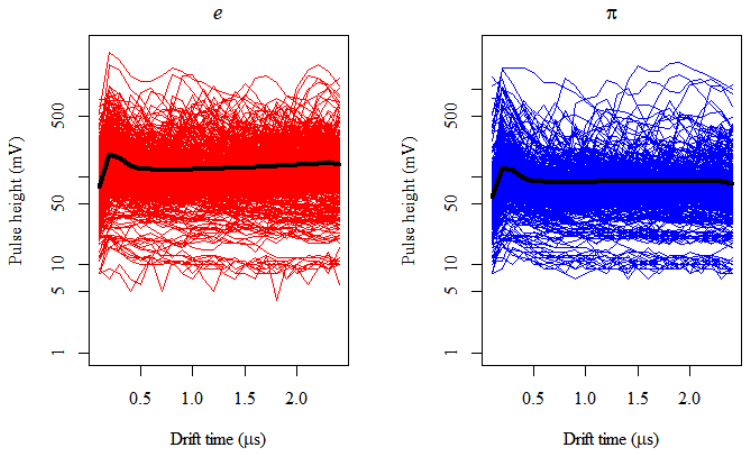


Figure 35: Pulse height as a function of time for 1000 randomly sampled electrons and 1000 randomly sampled pions, with the average pulse height, indicated as a black line for each particle species, illustrating the considerable fluctuations that occur around the average.

Figure 36 shows the declining number of tracks that reach the furthermost layers of the TRD.

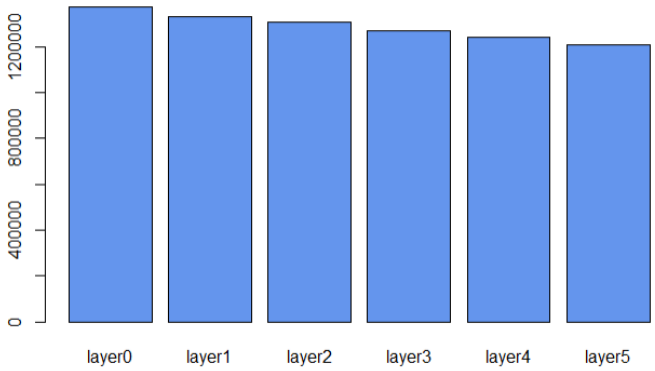


Figure 36: Number of Entries per TRD Layer Number, for all Runs

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 compared to the neural 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 and in Section 2.3.5, data used in this thesis was not properly calibrated.

4.2.1 [50] (Masters thesis, 2017):

Input data: 7 or 8 features, consisting of charge deposition, obtained by splitting into and summing over 7 time-bin 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 37.

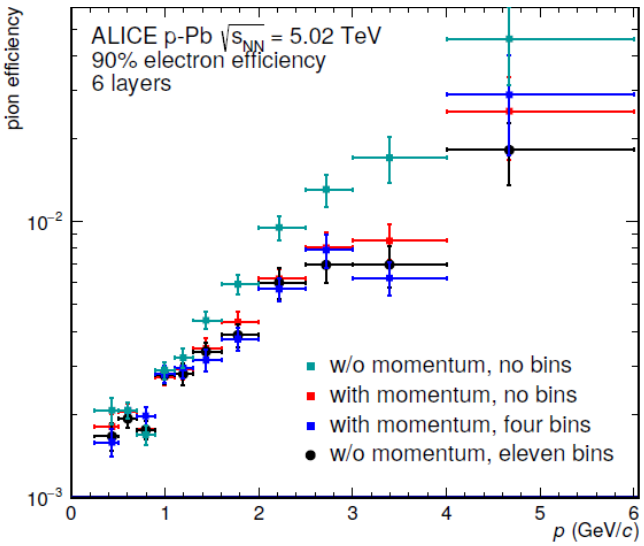


Figure 37: Pion efficiency results from [50]

4.2.2 [49] (PhD thesis, 2010):

Input data: charge deposition, obtained by splitting into and summing over 8 time-bin slices

Implementation: various neural networks with varying 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 the number of neurons in the input layer in Figure 38.

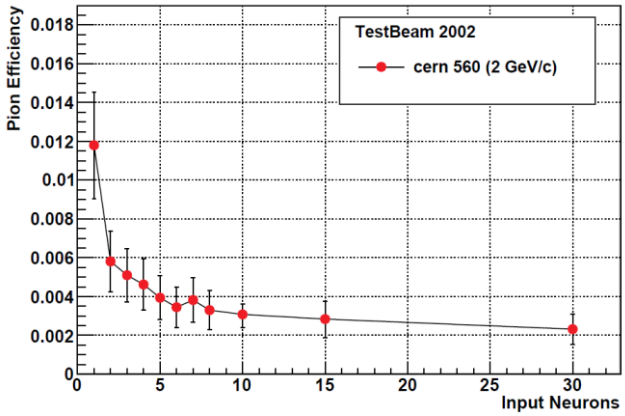


Figure 38: Pion efficiency results from [49]

4.2.3 [51] (PhD thesis, 2018):

Input data: charge deposition, obtained by splitting into and summing over 7 time-bin 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 using a neural network in this thesis are reported for only a single momentum bin around $p = 1\text{GeV}/c$. Particle identification was not the main aim of this thesis, but it is mentioned that the obtained results were generally on par with [49].

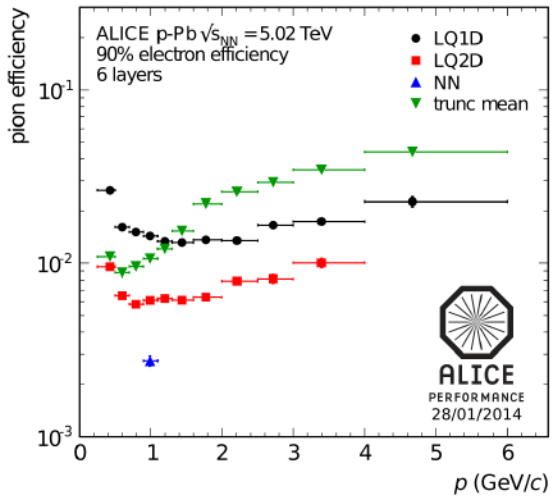


Figure 39: Pion efficiency results from [51]

4.3 Particle Identification Implementation: This thesis

Various Machine Learning strategies were employed in the task of particle identification. Please see [76] for the code used to build and train the various machine learning classifiers discussed. The implementation and results will be discussed from the very early stages, where models did not perform well, up to the most successful results.

4.3.1 A Journey of Discovery

4.3.1.1 Stage 1 of Model Building: Establishing Naïve Benchmarks & Generating Features by Hand

The initial approach towards particle identification entailed manual feature generation. Initially, a feature set was created, which took as input the following variables:

- Column sums of image pixels
- Five number summaries (Minimum, First Quartile, Median, Third Quartile and Maximum values, as well as Mean values) calculated across all the non-zero pixels in an image
- The number of non-zero rows in an image
- The column means of images in pixels
- The row means of images in pixels

Applying column-wise scaling and centring, to normalise the dataset was applied as follows:

$$x = \frac{x - mean(x)}{sd(x)}$$

Equation 30

A linear regression model was used as a very simple benchmark to compare future results against, using the abovementioned dataset, which contained around 99% pions. Using a naïve t_{cut} of the 99th percentile of the output distribution of the linear model, a single

tracklet pion efficiency of $\varepsilon_\pi = 0.91\%$ at electron efficiency $\varepsilon_e = 3.92\%$ was achieved. Following this, the value of t_{cut} was optimized by maximizing the area under the ROC curve when assessing the linear regression model's results, which improved results to $\varepsilon_\pi = 24.92\%$ at electron efficiency $\varepsilon_e = 66.85\%$.

It should be noted that class imbalances were not being accounted for at the very beginning as explained above. The next phase of the benchmarking process involved some experimentation with different methods to account for class imbalances, as well as hand-designing different feature-sets. Examples of things that were tried were:

Using a dataset with the same feature set described above, but which "upsampled" electrons with replacement to arrive at a sample of equal size to the pion sample. A fully connected neural network with architecture 256:128:64:32:16:2, with ReLU activations in the hidden layers and softmax activation in the output layer was trained for 500 epochs and achieved $\varepsilon_\pi = 13.15\%$ at $\varepsilon_e = 62\%$.

Next, a new dataset was created as follows: All electron tracks were included and a pion sample twice the size of the electron sample was added, before partitioning data into a training and test set. The following features were generated:

- The sum of all pixel values in each image
- Column sums of pixel values, scaled by dividing by the mean value of the entire matrix
- The lagged differences of the time evolution of pulse height (i.e. of the scaled column sums for each image), with both lag=1 and lag=2
- Further binning of the column sums, by adding the window sum of three columns at a time as an additional 8 features

A neural network with the following architecture 512:512:256:128:2, with ReLU activation functions in the hidden layers and softmax activation in the output layer managed to increase electron acceptance slightly, but with the attendant sacrifice of very high pion contamination, i.e. $\varepsilon_\pi = 63.1\%$ at $\varepsilon_e = 68\%$.

Then, the final experiment during this phase took inspiration from the current classification neural network in production at the TRD: time-bins were compressed by summing across three time-bins at a time, to create 8 new features, which were added to the 24 column sums.

Class imbalances were accounted for by allowing error in electron classification to contribute proportionately more to the loss function, by making use of a class weights dictionary (by specifying the `class_weight` parameter, calculated using the `compute_class_weight` function from the Scikit-Learn Python package [77] to the `keras::fit` function call). The neural networks built on this dataset overfit quite a bit to the training dataset and it was not possible to calculate pion efficiency scores for the final experiment. So, having set up an initial benchmark, we move into the next stage of model building.

4.3.1.2 Stage 2 of Model Building: High-Throughput Architecture- and Hyperparameter Search

During this stage of model building, class imbalances were accounted for by downsampling the pion sample to be equal in size to the electron sample (where only "clean" electron and pion tracks, ie. that consisted of 6 tracklets, were kept). An attempt at chamber gain calibration, using information from Kr-runs was made by [78], it is not certain whether this calibration procedure was successful, but nonetheless, it was this dataset that was used during the high-throughput stage of modelling. In addition, data was scaled as follows:

$$x = \frac{x - max(x)}{max(x)}$$

Equation 31

The SLURM-managed High-Performance Computing Cluster at UCT was utilized extensively to test the performance of various deep learning architectures, enabling one to train various deep learning models in parallel. During this stage, Keras was implemented in Python 3.6, with a Tensorflow CPU backend.

4.3.1.3 2D Convolutional Neural Networks

4.3.1.3.1 Summary

Dataset preparation: one image channel added to the $n \times 17 \times 24$ matrices, to produce an input tensor of dimensions $n \times 17 \times 24 \times 1$.

Best Result: $\varepsilon_\pi = 2.2\%$ at electron efficiency $\varepsilon_e = 90\%$.

4.3.1.3.2 Comprehensive Overview of Results and Discussion

Figure 40 compares the entire gamut of 2D Convolutional Neural Networks in terms of the number of layers (total and convolutional), the 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.) 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 since changes in the learning rate occurred at the same time that a switch was made to Adam; but common practice generally suggests that Adam is the more robust algorithm to use, due to its adaptive nature.

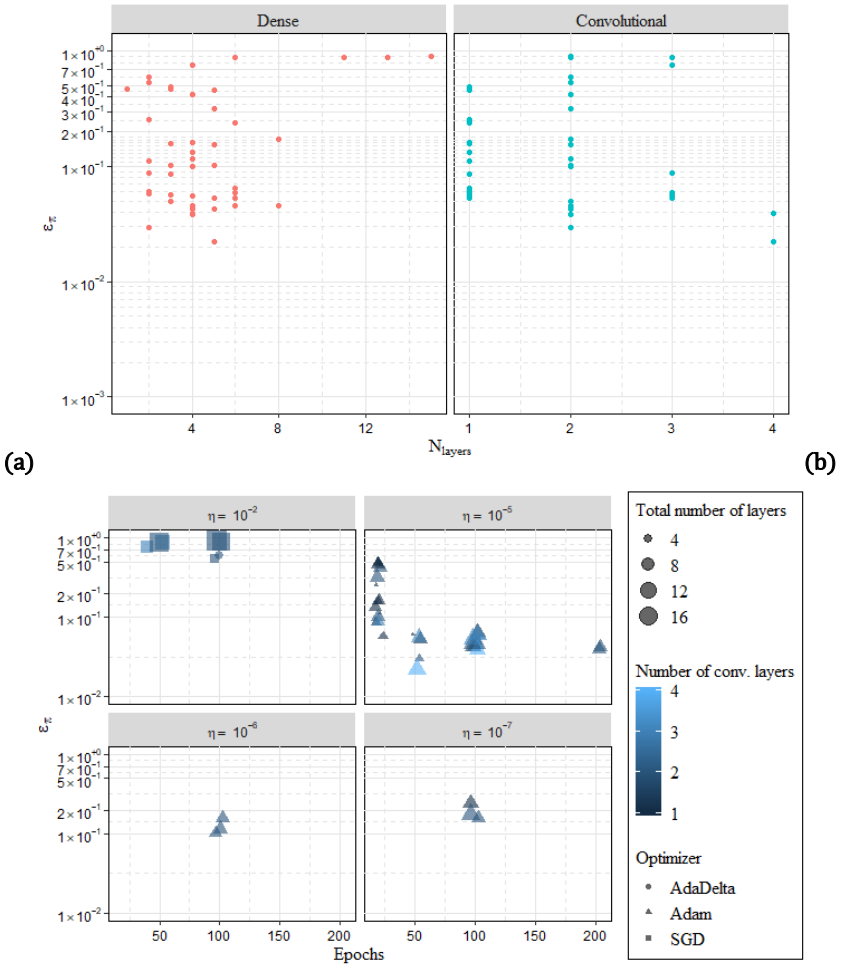


Figure 40 (a) Pion efficiency as a function of number of dense- (left) and convolutional- (right) layers (the same models are plotted on both sides of this figure) (b) Pion efficiency as a function of number of epochs trained, with additional information regarding model size and the optimizer used, indicated.

Although it is quite difficult to make any absolute statements about which model architecture is best suited to this problem, there are some important things that were learnt.

Using the correct learning rate is essential since using a very high learning rate will result in the gradient descent algorithm jumping over possible minima, whereas a learning rate which is too low will struggle to converge and might get stuck in local minima. 50-100 epochs seem to be enough iterations to train decently performing models, but the more important aspect to monitor is whether training loss and validation loss start to diverge, which indicates that the model is starting to overfit to the training set. It is hard to monitor this aspect when training on a server, but this can be achieved by piping the output of the Python script used for training to a text file, setting the Keras training verbosity to level 2 and monitoring the model's progress by calling the `tail -F out.txt` bash command to monitor the output to the text file once the model has begun running (this is also useful for debugging). In terms of computational cost, when running Convolutional Networks on the full dataset for 100 epochs, training time can take up to 2-3 days.

It is difficult to establish a clear guideline as to the number of layers (convolutional- or dense), which will result in a low pion efficiency at high electron efficiency. Models with a total of 8-9 layers seem to perform well in general, whereas models with 6 or fewer layers slightly worse, although the model which gave the second-best performance only had a total of 4 layers, of which two were convolutional. Using more than one convolutional layer is also a seemingly more successful strategy than using just one layer.

Using larger kernel sizes seems to be a good strategy, judging by the top two best-performing models, who both had 8x12 and 5x6 kernels in the first two layers of the network. The best-performing model was one of only two models that had 4 convolutional layers, the other model with this much convolutional depth being in the top 4 highest-performing models as well. There is no consistent indication of whether using Max Pooling improved performance in this task, but the top two models did not make use of them.

Most 2D Convolutional models were trained with a batch size of 32; while using a smaller batch size makes gradient updates slightly more volatile and therefore a little less accurate, using very large batch sizes makes this already computationally expensive procedure take even longer. Note that using the appropriate learning rate can account for the attendant volatility of the gradient which will inevitably result from a smaller batch size.

While a lot of time was spent on building different architectures, comparatively little time was spent on optimizing hyperparameters, such as learning rate, batch size, dropout rate, weight initialization strategies and testing different standard deviation settings with Gaussian Noise layers. In retrospect, spending some time on setting up Randomized Grid Searches could have been useful.

4.3.1.4 1D Convolutional Neural Networks

4.3.1.4.1 Summary

Dataset preparation: one channel added to the 24 column-sums, to produce an input matrix of $n \times 24 \times 1$

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

4.3.1.4.2 Comprehensive Overview and Discussion

Although much fewer models with 1D Convolutional architectures were built, a deeper convolutional architecture seems to be a good strategy in solving the problem of particle identification when using 1D convolutional layers. The best-performing model had four 1D Convolutional Layers, but it didn't perform that much better than the other models which had less convolutional layers. It was also the only model with such a deep convolutional architecture, so it's hard to say whether low pion efficiency truly depends that heavily on the depth of the convolutional architecture of a 1D CNN.

Interestingly, all 1D CNN models used Tanh activation functions in the hidden dense architecture, whilst 2D CNNs mostly relied on Sigmoid activations in their hidden dense layers. The range of outputs from a Tanh activation is $[-1,1]$, whereas the range of a Sigmoid activation is $[0,1]$ (see Figure 42). The Tanh range is, therefore, more symmetrical than that of the Sigmoid activation function and maps negative input values to negative output values, which provides for stronger gradients. Since the sigmoid function can only output positive values, it saturates faster (i.e. if the pre-activation input to a sigmoid function is close to zero, the gradient on that neuron might vanish, rendering it untrainable).

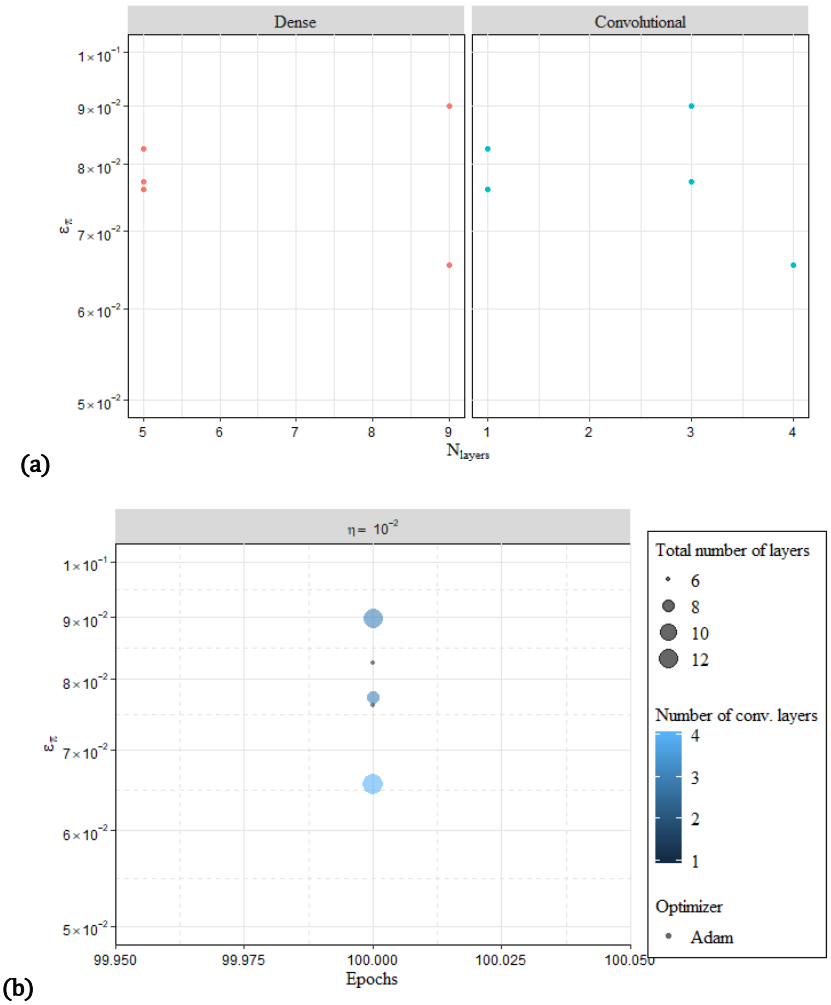


Figure 41 (a) Pion efficiency results for 1D Convolutional Neural Networks as a function of number of dense- (left) and number of convolutional layers (right), (b) Pion efficiency as a function of number of epochs trained, with additional information about learning rate, number of total and convolutional layers and the optimizer used

It is possible that the high performance of 1D CNNs, despite the fact that they were fed data with less information (column sums, instead of full images) can be partly explained by the choice of activation function. In addition, the fact that the input data to 2D CNNs was extremely sparse, with most pixels being close to zero, could have rendered a lot of the neurons in the early layers saturated at zero and therefore untrainable. Perhaps a better normalisation strategy in the case of 2D CNNs could have been: $x = \frac{x - mean(x)}{sd(x)}$, which would have

resulted in a zero centred and scaled pixel distribution, instead of an arbitrarily distributed pixel distribution in the range [0,1], especially considering the choice of activation function that was used.

Another thing to consider is that no activation functions were applied to the Convolutional Layers of any of the 1D CNNs that were built, i.e. $\phi(x) = x$, it is not possible to discern what the effect of this strategy was when comparing within the set of 1D CNNs, since all these models were built with no activation functions, but it's worth noting that the fourth-worst 2D CNN also employed this strategy, although this is entangled with the high learning rate that was used in this particular 2D CNN. While 2D Convolutional Neural Networks were more successful than 1D Convolutional Neural Networks, 1D CNNs nonetheless resulted in similar performance to 2D CNNs, even though they were fed data with lower dimensions, which could be an indication that using a dataset with fewer dimensions, could possibly cancel out some of the noise in the uncalibrated raw data. That being said, it should be said again that many more 2D CNNs were built during this project and that one would expect a 2D CNN model with sufficient capacity (size) and exposure to enough training examples to be able to distinguish noise from signal, up to a certain point.

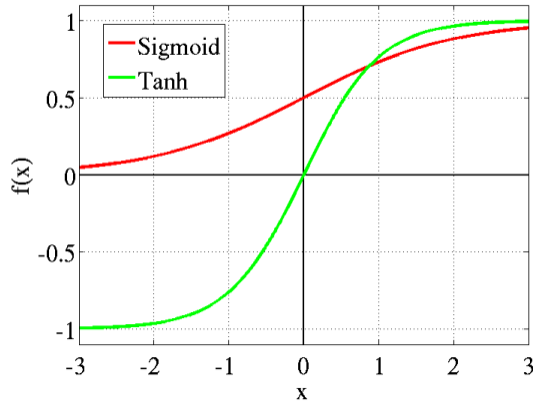


Figure 42: Sigmoid vs Tanh activation [79]

4.3.1.5 Fully Connected Feedforward Neural Networks

4.3.1.5.1 Summary

Dataset Preparation: the original $n \times 17 \times 24$ matrices were collapsed down to 17 row-sums and 24 column-sums, to arrive at a wide input matrix with dimensions $n \times 41$. Some networks were also trained with additional hand-designed features

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

4.3.1.5.2 Comprehensive Overview and Discussion

These networks resulted in the lowest performance of all models, but very few of them were built. Using a deeper architecture with a lower learning rate was slightly more successful than other strategies. Designing a custom set of features was also not particularly successful. Results for fully connected feedforward neural networks are shown in Figure 43.

4.3.1.6 LSTM Neural Networks

4.3.1.6.1 Summary

Dataset Preparation: the original $n \times 17 \times 24$ matrices were transposed to have dimensions n × 24 × 17, i.e. 24 time-bins with 17 features

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

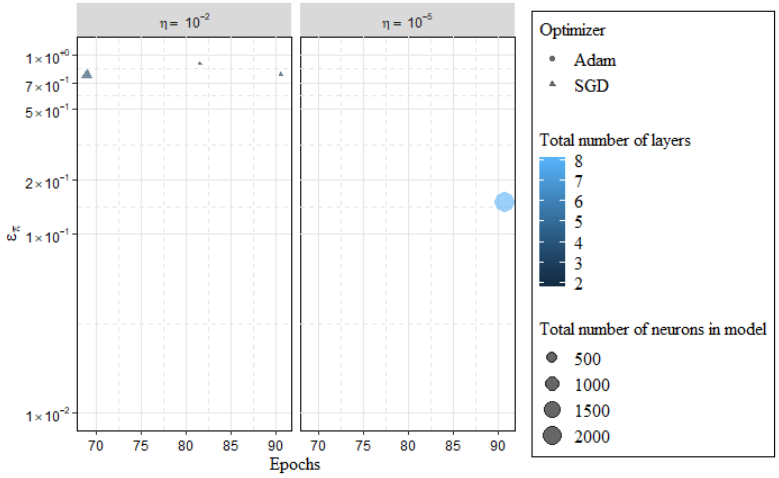


Figure 43: Pion efficiency as a function of the number of epochs trained for Fully Connected (Dense) Neural Networks, with additional information about learning rate, number of layers, the total number of neurons in the model and the optimizer used

4.3.1.6.2 Comprehensive Overview and Discussion

An interesting feature of this dataset is that it can be framed in multiple ways (as an image for 2D CNNs, as a flattened or summarised array for Dense Fully Connected Neural Networks and as a time-series for 1D CNNs and LSTM Networks). By transposing each image matrix, an input feature set with the following dimensions can be obtained: $n \times 24_{timesteps} \times 17_{features}$.

LSTM Networks did give the second-lowest pion efficiency of all models trained, but one might have expected them to perform better than 2D CNNs since a TRD signal is technically more of a time-series than an image. Perhaps their inferior performance can be explained by noise in the dataset (perhaps by summing 3 columns sequentially, but maintaining the number of rows, or by performing any other number of arbitrary data preprocessing steps, but generally one would hope that a deep learning model will do that kind of feature extraction by itself).

The lowest pion efficiency using LSTM networks was $\varepsilon_\pi = 8.5\%$, which is much higher than that achieved by the best 2D Convolutional Neural Networks. This was achieved by having 4 LSTM layers return sequences sequentially, followed by a final LSTM network which fed into a dense architecture of 5 fully connected layers of 128 nodes each; although similar performance ($\varepsilon_\pi = 8.9\%$) was obtained using only two LSTM layers, with the second layer going backwards, followed by four dense layers of 256 nodes each.

The results for LSTM networks are summarised in Figure 44.

4.3.1.7 Non-Deep Learning (Tree-Based) Models

Non-Deep Learning Methods (Gradient Boosting Machines and Random Forests) were implemented locally, using H2O.ai, using the default parameter settings for each model type.

This was mainly done as a sanity check but resulted in decent results with minimal effort.

4.3.1.7.1 Random Forests

Dataset preparation: 24 columns sums and 17 row sums for each tracklet (normalised and standardised).

Quick theoretical overview and implementation:

A random forest is a tree-based algorithm which grows k decision trees on k bootstrapped samples of the dataset, considering a random sample of $m < p$ features at each split, which is a decorrelating procedure which prevents strong predictors from having an overly large influence on the outcome of each tree and then averaging the output of all the trees grown.

The default `h2o.randomForest` parameter settings were used (50 trees, maximum tree-depth=20, number of variables sampled at each split=$\sqrt{p} = \sqrt{41} \approx 6$, etc.), using 10-fold cross-validation.

Results and Discussion:

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

These results were on par with LSTM and 1D Convolutional neural networks' results, without any fine-tuning or exploration of different hyper-parameter settings.

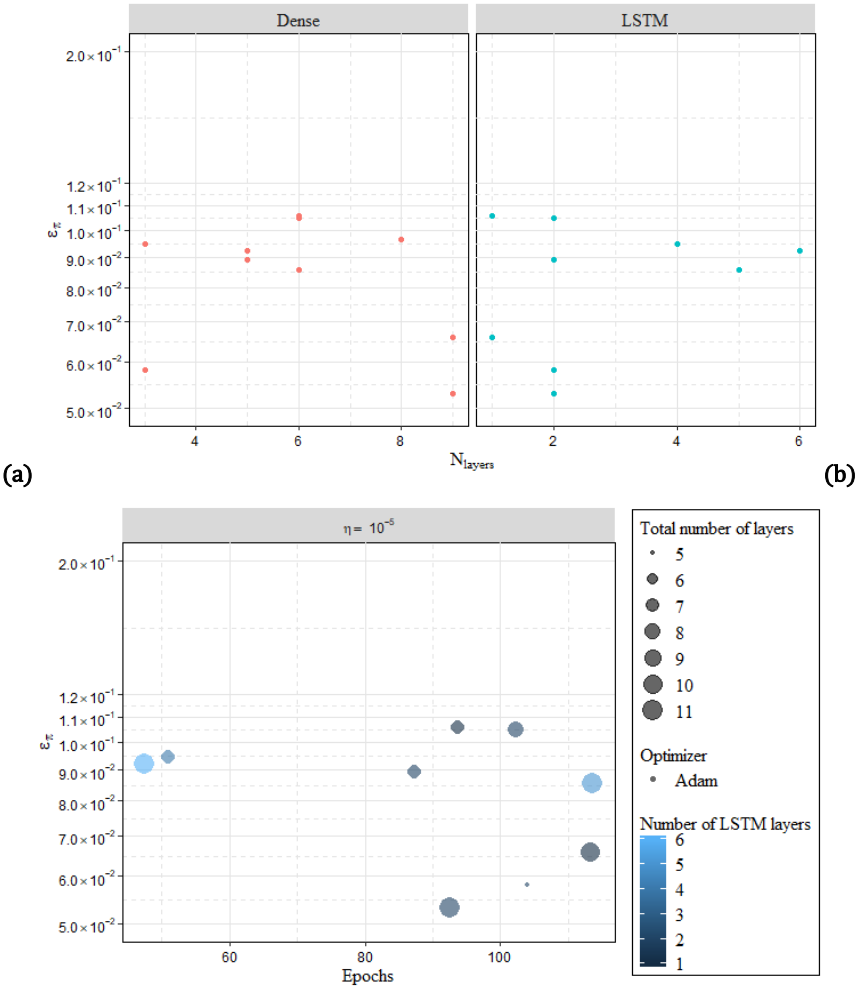


Figure 44 (a) Pion efficiency results for LSTM Neural Networks as a function of number of dense- (left) and number of LSTM layers (right), (b) Pion efficiency as a function of number of epochs trained, with additional information about learning rate, number of total- and LSTM layers and the optimizer used

4.3.1.7.2 Gradient Boosting Machines

Dataset preparation: 24 columns sums and 17 row sums for each tracklet (normalised and standardised).

Quick theoretical overview and implementation:

Gradient-boosting machines (GBMs) are another tree-based implementation involving the independent growth of multiple decision trees. Multiple weak learners (shallow trees) are combined, with each subsequent tree learning based on the residuals (error) of the previous tree. i.e. Each following tree accounts for additional variation which was not explained by the preceding trees in the chain.

The default `h2o.gbm` parameters were used (50 trees, maximum tree-depth=5, learning rate $\eta = 0.1$, etc.), using 10-fold cross-validation.

Results and Discussion:

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

Similar to the random forest experiment, GBMs resulted in results comparable to LSTM and 1D Convolutional Neural Networks, using the default parameters of the H2O package.

4.3.1.7.3 Discussion of Tree-based methods

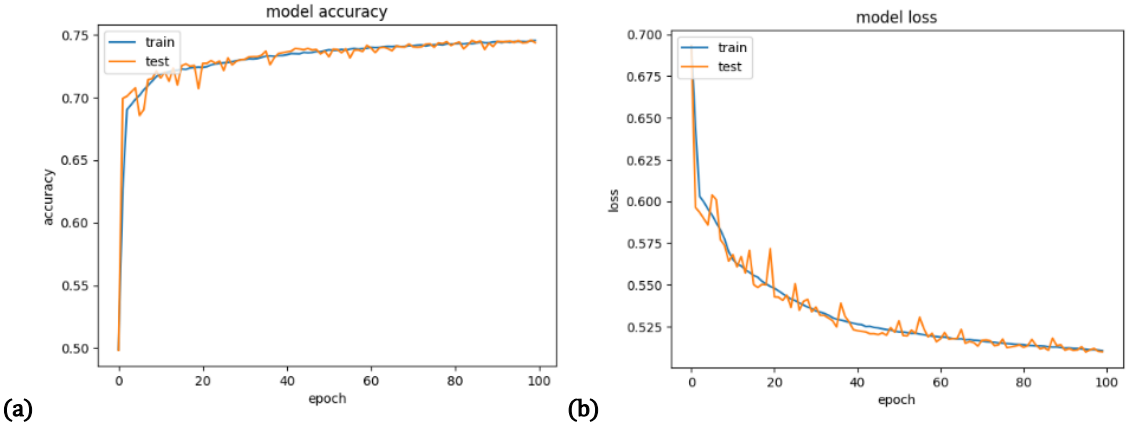
The fact that two algorithms, each with an entirely different theoretical underpinning than neural networks performed so well with minimal tuning, allows us to deduce a few things: Firstly, that there is a limit to the amount of information contained in the input dataset since almost all of these algorithms resulted in very similar ε_π. Secondly, that there might be something to be gained from exploring these tree-based methods further, but also that the ten-fold cross-validation implemented for these two algorithms was not done for any deep learning algorithms and that might be part of the reason they performed so well.

4.3.1.8 Discussion of Stage 2 of Model Building

At the conclusion of the second stage of model building, the following 2D convolutional neural network architecture (Appendix II: Figure 78) achieved the lowest pion efficiency $\varepsilon_\pi = 2.2\%$ at electron efficiency $\varepsilon_e = 90\%$:

Using the Adam optimizer with learning rate $\eta = 10^{-4}$, trained for 100 epochs with a batch size of 32, using binary cross-entropy as the loss function to be optimized.

The training and validation accuracy and loss graphs for this model are depicted below.



(a) (b)

Figure 45: (a) Training vs Validation Accuracy. (b) Training vs Validation Loss

Pion efficiency at 90% electron efficiency was calculated by writing a function which, when minimized, finds the cut-off point (t_{cut}, critical region) in the distribution of the test statistic found by combining all 6 (tracklet) estimates for a track, as outlined in 3.1.7, where this t_{cut} results in 90% of true electron tracks being classified as electrons.

The ultimate result of the minimization process is shown in Figure 46. In this set-up, any track which receives a combined probability above t_{cut} (shown as a vertical red line) was classified as an electron, otherwise, it was classified as a pion.

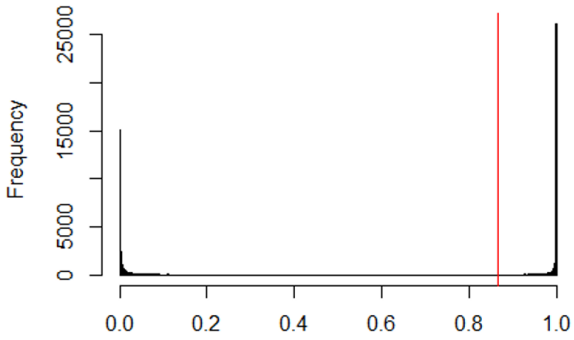


Figure 46: t_{cut} selection in *t(x)/P(elec)* allowing for 90% electron efficiency

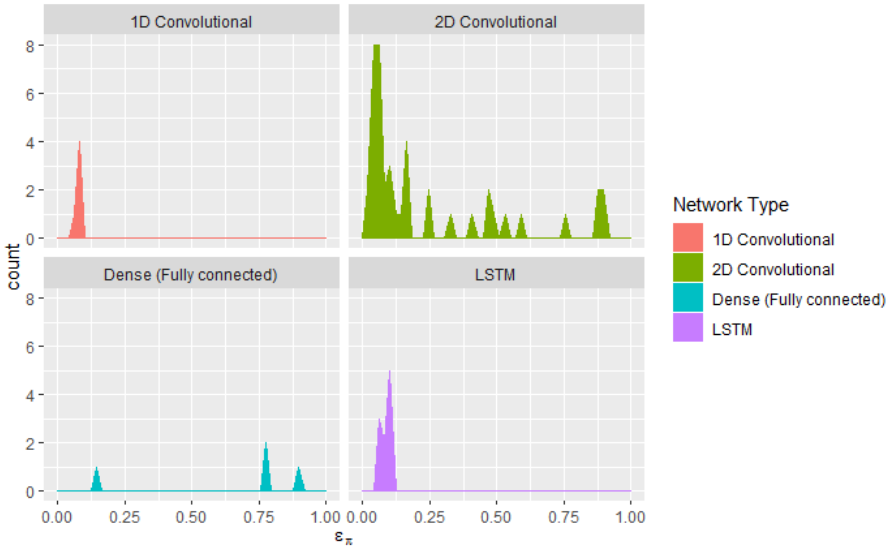


Figure 47: Distribution of ε_π for each model type

When looking at Figure 47, it is starkly apparent that the fact that 2D CNNs outperformed other models could be a function of the number of these models that were explored. There are also ways of combining LSTM and convolutional neural networks (see for example [80]), which might have been promising, but due to time constraints, this was not explored.

4.3.1.9 Some Remarks About Regularization

Figure 48 shows the effect of applying too much Dropout at training time. The left side of the figure shows a model which overfit during training and the right-hand side shows the effect of applying a Dropout rate of 0.5 to each layer of the same network.

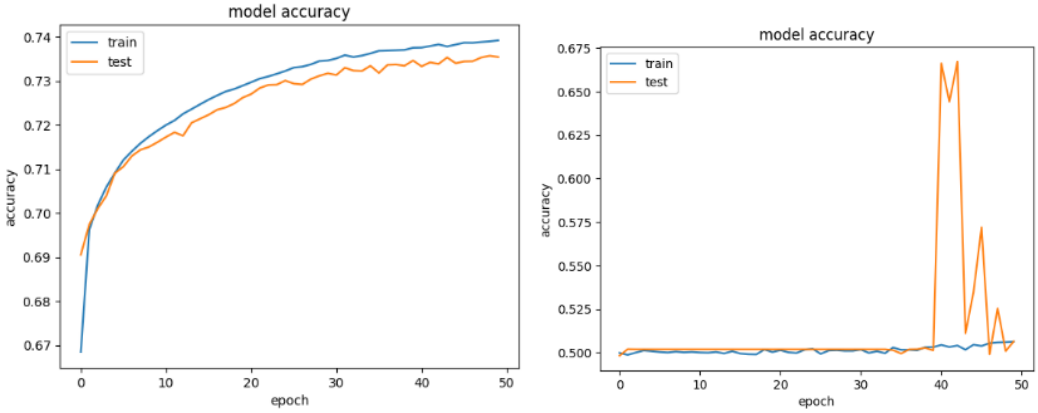


Figure 48: No Dropout (left-hand side) vs the Same Model with too much Dropout (right-hand side)

In general, using a Dropout rate of 0.2 on the fully connected layers of the network proved to be a decent strategy, bearing in mind that other hyperparameters also play a role. The use of a Gaussian Noise layer as the first layer of various convolutional architectures was employed, but was unsuccessful, as depicted in Figure 49. All models that incorporated Gaussian noise gave pion efficiencies of $\varepsilon_\pi >$ 45%. While there was not much experimentation with different standard deviations of the Gaussian noise layer, this method seems more applicable to image data which is less sparse, since it is only active during training time, and since most of the rows in our input data naturally has a value of 0, adding Gaussian noise will only serve to confuse the model when evaluated at test time.

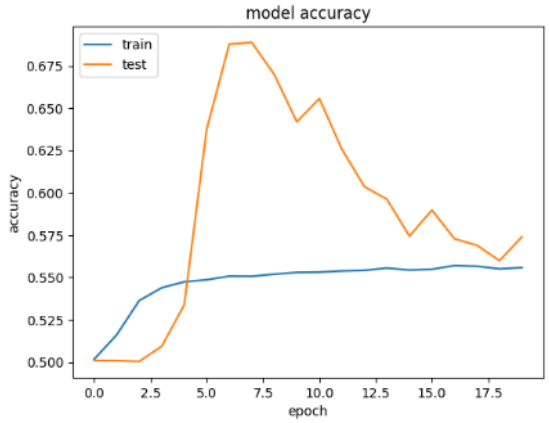


Figure 49: Training and validation accuracy for an example 2D CNN model making use of a Gaussian Noise layer with σ=0.2

It is important to apply some form of regularization, or to make use of early stopping, to prevent overfitting to the training dataset, which will severely influence performance on the test dataset.

4.3.1.10 Stage 3 of Model Building: Hitting the Sweet-Spot

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 $\eta = 1 \times 10^{-4}$.

Dataset preparation:

- The full (uncalibrated) dataset, split into the following momentum bins, was used during this stage:
  - $p \leq 2$ 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
- All tracklets with no signal, i.e. images where all the pixel values were zero, were removed.
- Data was not scaled, normalised or standardised.
- Data was not down-sampled or up-sampled to account for class imbalances.

Implementation and Results:

A Convolutional Neural Network (architecture shown in Appendix II: Figure 79) was trained incrementally, per momentum bin, by saving the weights-configuration of the model after training on the previous momentum bin.

A detailed description of the steps that were followed in this process are discussed below and the final results are summarised in Figure 51, in comparison the current methods used by the TRD working group of the ALICE collaboration, as discussed in 2.3.4.

Keras allows one to resume training of a model, by increasing the number of epochs in the `keras::fit()` function to the total number of desired epochs, i.e. number of epochs (previous training round) + additional epochs to be run, and specifying an `initial_epochs` argument, which should be the final epoch number from the previous training round.

Using this functionality, a 2D Convolutional Network was trained on particles in the $P \leq 2GeV$ range for 10 epochs (note that the class imbalance at this momentum range is less pronounced than for other momentum bins, cf. Figure 32). Figure 50 (a) shows the accuracy and loss curves obtained.

When it became clear that the model was not yet overfitting, but had some statistical power (validation accuracy ~80% compared to what a naïve majority class classifier would give, i.e. ~76%), the model was trained for an additional 20 epochs (Figure 50 (b)).

This model performed as follows on an independent test set of particles in the same momentum range that it was trained on: $\varepsilon_\pi = 1.2\%$ at $\varepsilon_e = 89.99\%$, which is much better than anything obtained during the second stage of model building. Surprisingly, this model performed almost equally well on particles in the following momentum bin, i.e. performance on particles with momenta $2\ GeV < P \leq 3GeV$ was $\varepsilon_\pi = 1.4\%$ at $\varepsilon_e = 90\%$, *before* being trained on particles in this momentum bin.

Another feature of Keras is that the state of the optimizer of a saved model can be discarded and only the weights of a previously trained model can be loaded into a new model with the same architecture. Using this functionality, training was continued, using the pre-trained model, on particles in the $2\ GeV < P \leq 3GeV$ range for 5 epochs.

When evaluated on a test set in *this* momentum range ($2\ GeV < P \leq 3GeV$), performance increased slightly to $\varepsilon_\pi = 1.14\%$ at $\varepsilon_e = 89.99\%$ And when tested on particles in the *next* momentum range ($3\ GeV < P \leq 4GeV$), performance remained surprisingly high: $\varepsilon_\pi = 1.16\%$ at $\varepsilon_e = 89.86\%$.

This model was then trained for a further 5 epochs on particles in the $3\ GeV < P \leq 4\ GeV$ range. Performance actually decreased when testing this new model on particles in the $3\ GeV < P \leq 4\ GeV$ range, to $\varepsilon_\pi = 1.51\%$ at electron efficiency $\varepsilon_e = 89.99\%$. This might be explained by the fact that there is a much smaller proportion of electrons in this momentum range than in the lower GeV ranges. And when tested on particles in the next momentum range ($4\ GeV < P \leq 5\ GeV$), the model's generalizability breaks down and predicts "pion" everywhere.

The incrementally trained model, which at this stage has already become extremely biased towards predicting "pion" with very high certainty, was then trained on particles in the $4\ GeV < P \leq 5\ GeV$ for a further 5 epochs. At this stage of training, it has seen a lot more pions than electrons, and the focal loss function seems no longer able to mitigate the extreme class imbalance. The model was not trained for particles in the $5\ GeV < P \leq 6\ GeV$ range as a separate training set since it had already started overfitting (beyond rescue) to the imbalanced class distribution.

An interesting fact about the results obtained during the final stage of model building is the fact that the data it was trained on was not scaled in any way. Perhaps this was also a contributing factor to the low pion efficiencies obtained, possibly because the input signal

arrays are very sparse, with most pixels being equal to zero; i.e. perhaps the gradients during backpropagation flow more strongly when such sparse input arrays remain unscaled. This is just guesswork; for the most part, neural networks remain proverbial black boxes.

Finally, these results are compared to previous results in Figure 51.

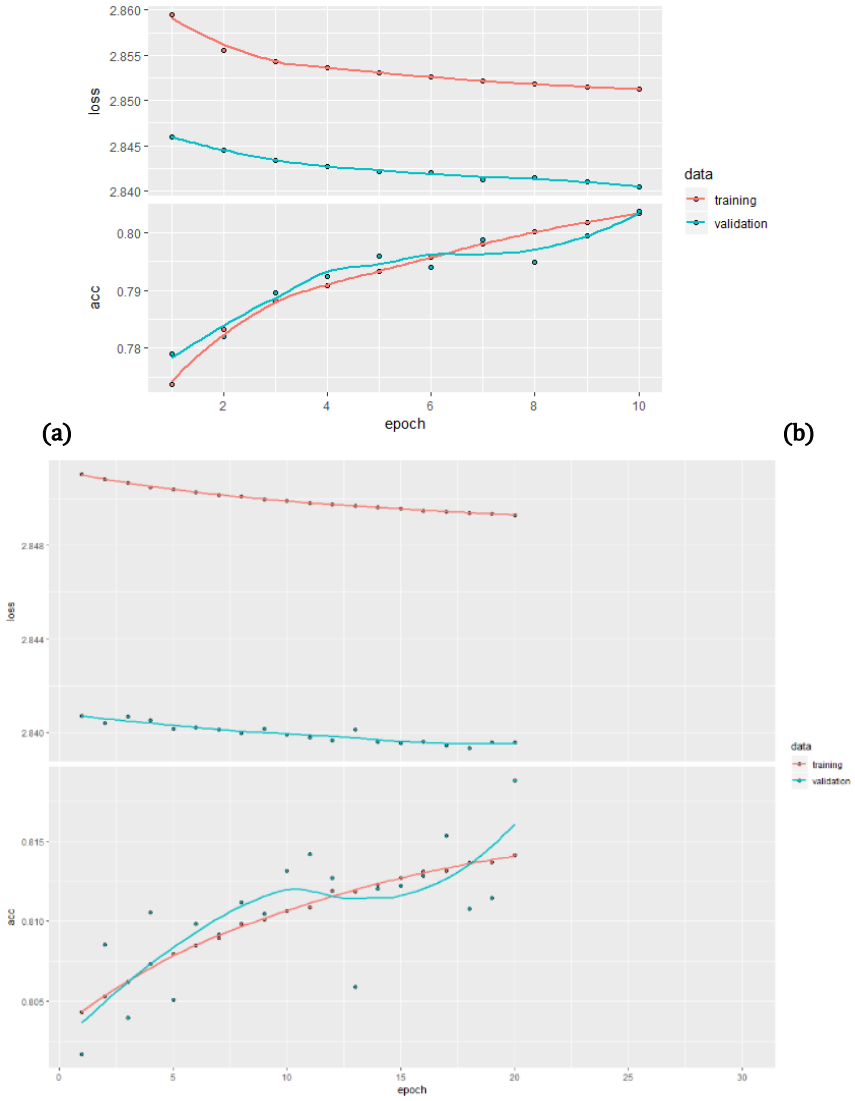


Figure 50 Training and loss curves for the most successful model, making use of Focal loss as the objective function to be minimised, trained for particles in the 2 GeV range (a) for 10 epochs, (b) for an additional 20 epochs

4.4 Chapter Discussion and 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 (or semi-calibrated) data.

A statement needs to be strongly made again is that, what is more important than arriving at the optimum neural network architecture-/hyperparameter combination, when aiming to achieve very low pion efficiencies, 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 the obtained 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 from separate pPB runs was used. In some cases, the signal was completely empty or could clearly be seen to be the result of, or influenced by, noise. However, 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.

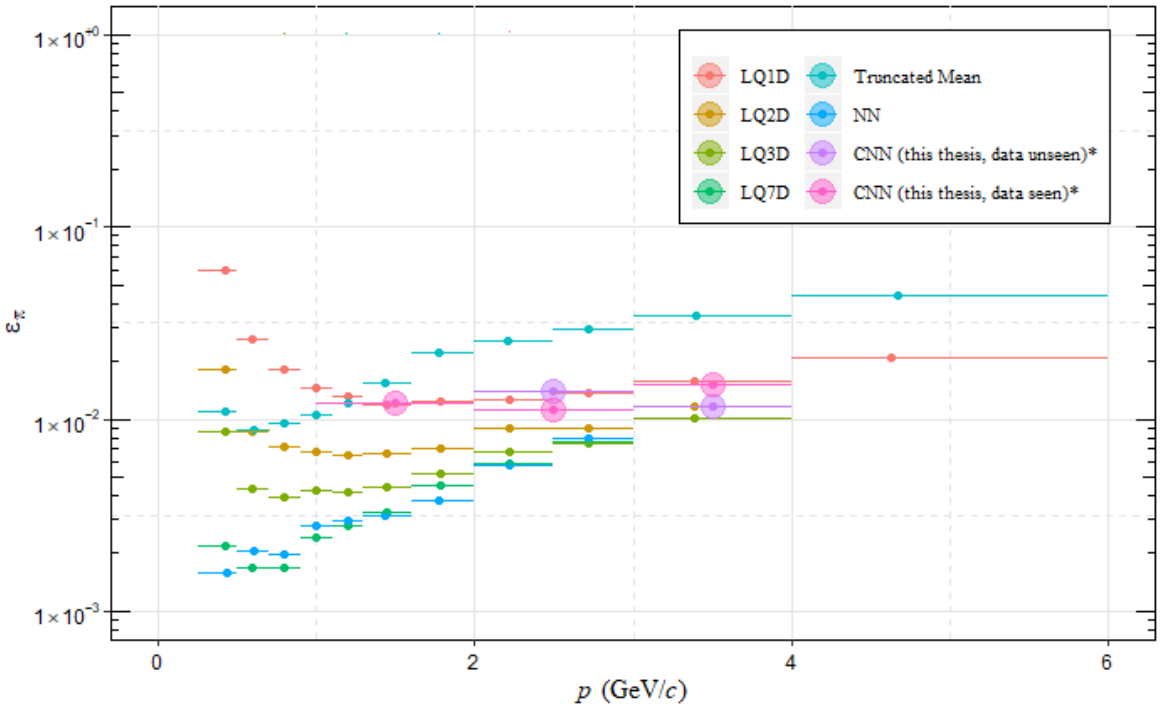


Figure 51: Summary of incrementally trained 2D Convolutional Neural Network: Pink marks indicate pion efficiency at 90% electron efficiency (when the incrementally trained model was evaluated on data from the momentum-bin that the model was *last* trained on). Purple marks indicate the results when testing the model on data from the *next* (higher) momentum bin (before training on data in that bin). Previous work done on π/e particle identification based on TRD data is shown for reference. Raw data points for the other methods were received from [81].

In addition, there is a potential limit to how much information is actually contained in these images and, quite possibly, there is a hard limit to the maximum achievable ε_π based on TRD data in isolation (especially when uncalibrated), regardless of how sophisticated and optimised a model one uses. However, during a full physics analysis, TRD data would not be the only available data for particle identification and, for instance, combining it with data from the TPC detector, one would arrive at a very low ε_π.

4.4.1 Future work

There are advanced hyperparameter tuning strategies, such as Bayesian-, Evolutionary- and Gradient-based Hyperparameter optimisation techniques that could be employed in future work in this area. But it is not recommended to expend any more effort on particle identification with an uncalibrated dataset since there seems to be an upper limit to the amount of distinguishing information that this type of data carries about the particle ID. The inferior results compared to previous work is definitely not due to an insufficient

amount of exploration of the space of possible models, or possible settings of hyperparameters, but because of inherent limitations pertaining to the input dataset used for training.

Future work on properly calibrated data could explore tree-based methods in more depth and could also make use of cross-validation for neural networks, or make use of ensembles of algorithms, where multiple algorithms that perform well "vote" towards an outcome, using either a weighted average based on their performance, using each of their outputs as input features to another algorithm (even something as simple as a linear regression model) or by making use of Bayesian and Likelihood ratio methods based on predictions from multiple algorithms.

The reason ensembling is suggested is that it is unlikely that models with different architectures will produce downstream derived features that are similar and therefore they might make different "cognitive errors" during prediction, which could be compensated for by averaging or combining their outputs in some way.

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.

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 [82].

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 as 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 [83]. Its physics implementation is transparent to investigation and validation and can be customised and extended [82].

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 [82]. Figure 52 shows how these categories are related, with lines indicating a “using” relationship (category with open circle uses the adjoining category). A full discussion of Geant4 lies outside the scope of this thesis, but it is worth recognising the complexity of its implementation.

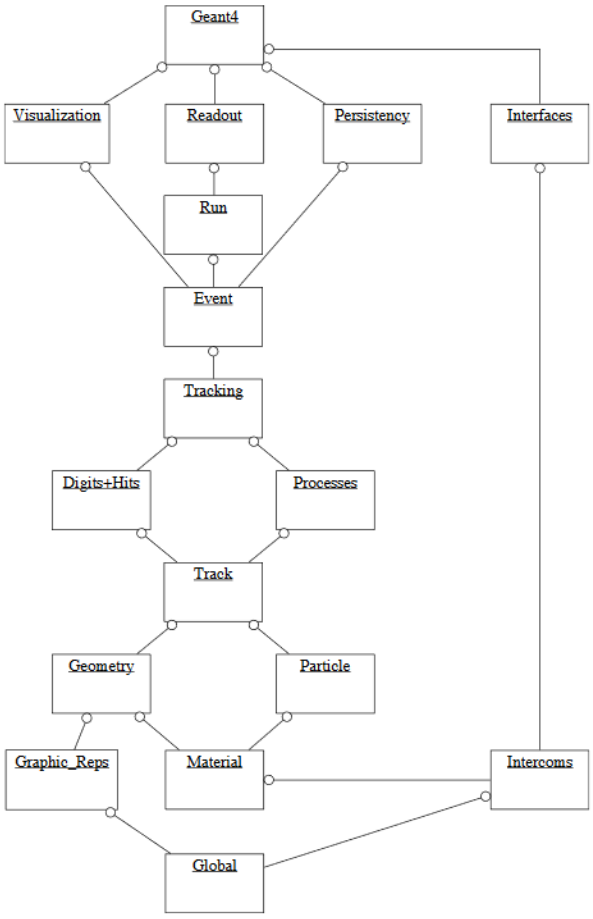


Figure 52: Top-level category diagram of the Geant4 toolkit [82]

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 [84].

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 [84].

Neural networks can be utilised as function approximators towards constructing a modelled distribution P as outlined above [84]. Deep Generative Modeling usually entails making use of the concept of a "latent space" representation of the input data; this will be explained in the following sections.

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 (from the latent representation space) before it starts simulating [84].

In order to deduce that a generative model is representative of the underlying distribution, 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 [84].

A vector of latent variables z, is 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 [84].

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

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

Equation 32

In Equation 32, $f(z;\theta)$ has been changed to a distribution $P(X|z;\theta\,)$, 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 [84].

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 [84].

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 [84].

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 [84]. Generally, a researcher would not explicitly specify what the dimensions of z specify, nor how the dimensions of z depend on one another [84].

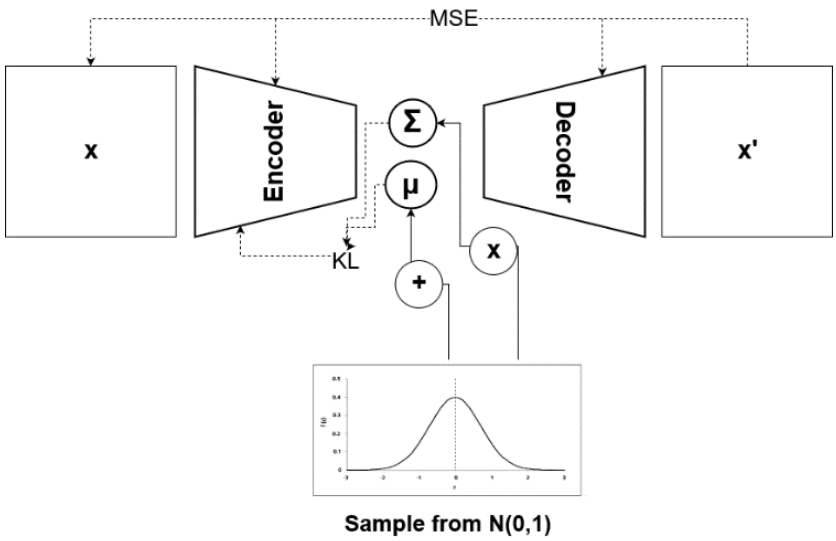


Figure 53: Simplified diagram of a Variational Autoencoder

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 [84].

$P(X)$ 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 [84].

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 $P(z)$ [84].

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 ($\mathcal{D}$, also referred to as relative entropy, which essentially measures how different two probability distributions are) 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 33

$P(X)$ 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 34

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 35

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 36

Equation 36 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 37

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 38

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 39

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 40

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.

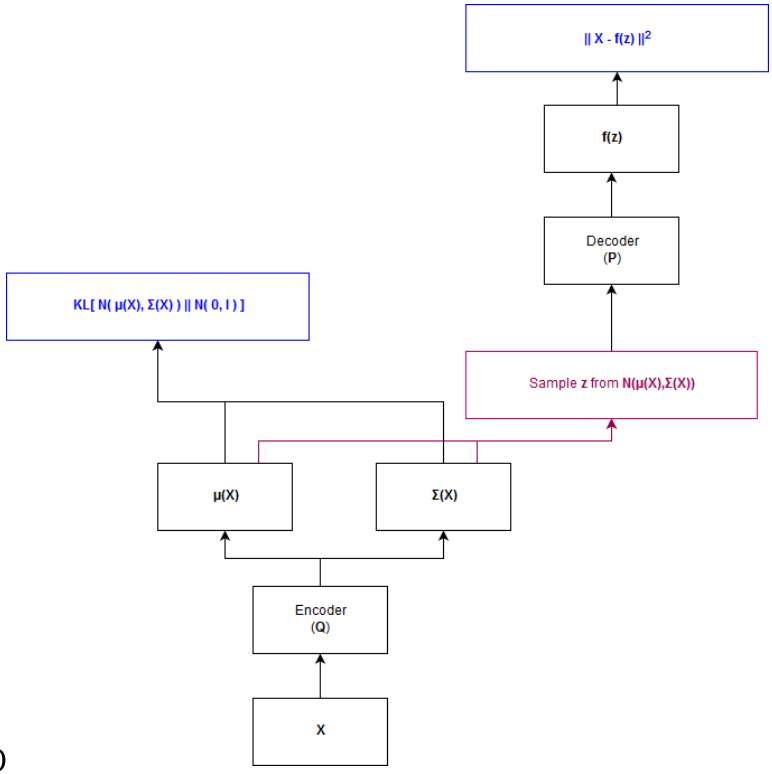
Figure 54 (a) illustrates how this proxy formula can be used by averaging over multiple samples to get to the expected outcome; but, since there is a sampling procedure embedded within the theoretical neural network, gradient descent cannot be performed on it.

Figure 54 (b), on the other hand, shows how a "reparameterization trick", which removes the sampling procedure from the neural network proper and treats it as an input layer, is implemented in practice. Since we have $\mu(X)$ and $\Sigma(X)$, we can sample ϵ from $N(0, I)$ and compute z from ϵ as follows: $z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon$.

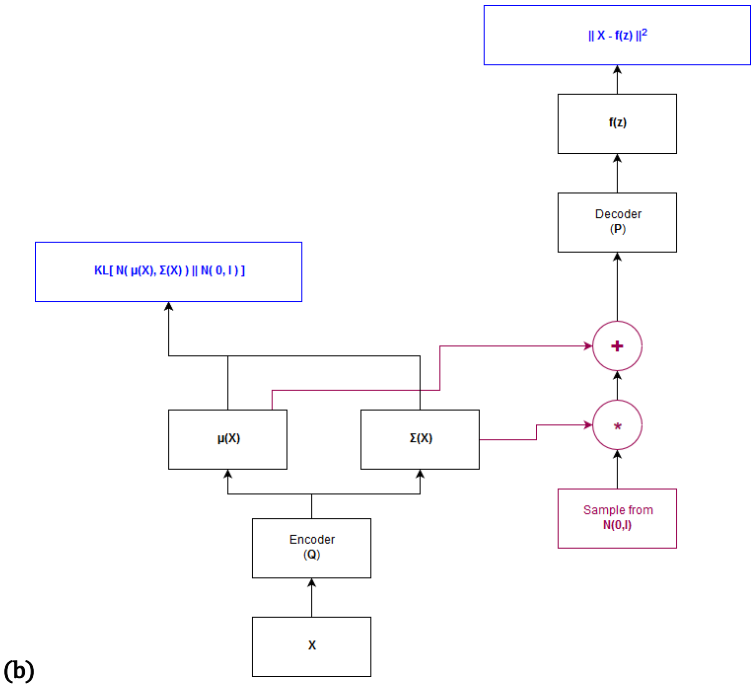
As a result, the gradient of the following equation will actually be taken:

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

Equation 41



(a)



(b)

Figure 54: Training-time VAE (a) Theoretical implementation (b) Reparameterization trick, which is implemented in practice

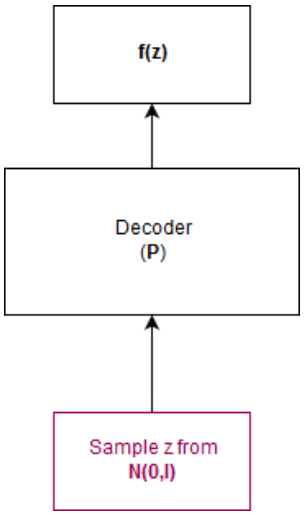


Figure 55: Testing time VAE

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

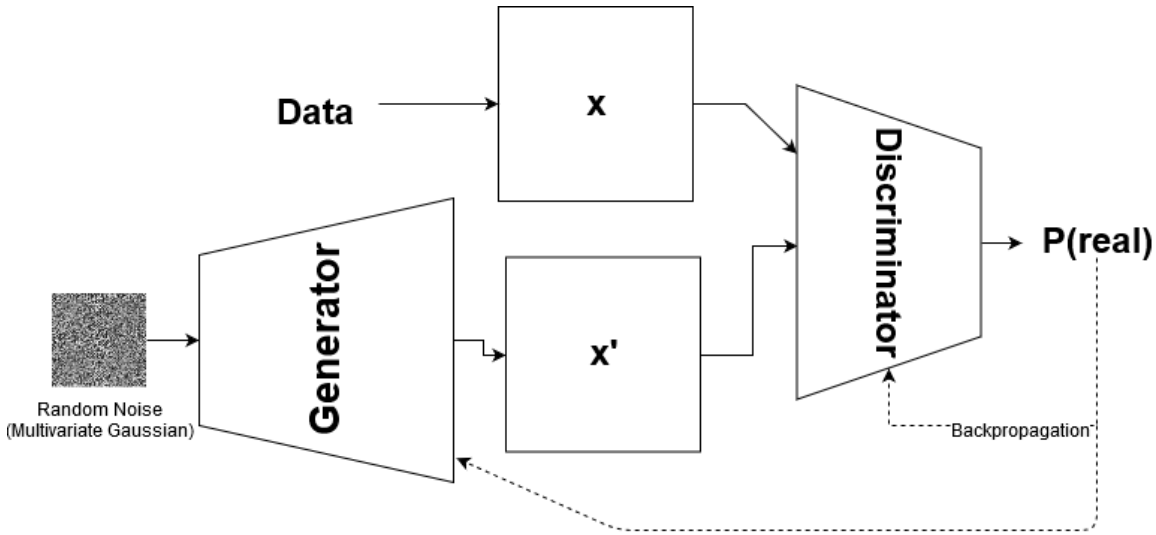


Figure 56: Simplified Diagram of a Generative Adversarial Network

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 [85].

The mini-max game can be expressed mathematically as:

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

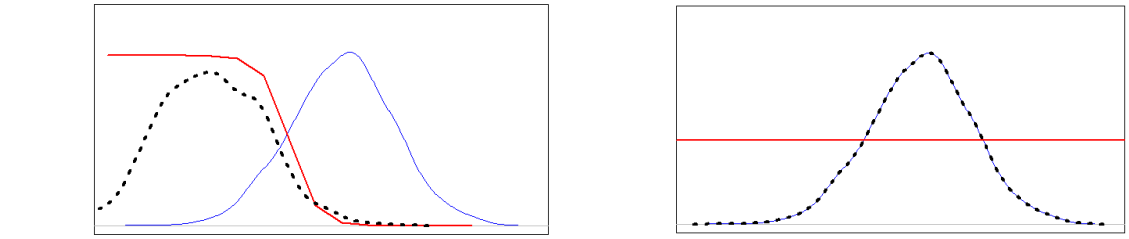
Equation 42

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 either the "data"- or the "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 $log\left(1 - D\big(G(z)\big)\right)$ will saturate, so an alternative objective function $log\ D\big(G(z)\big)$ 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 57 illustrates more intuitively what this training process converges to (if successful). Here, the generative model is able to capture the true data distribution perfectly and the discriminative network is unable to discriminate true from generated examples.



(a) (b)

Figure 57: (a) GAN Densities during training (simplified diagram showing distributions of only a single dimension), close to convergence, P(x) is shown in black, G(z) in blue and D(G(z)) in red, (b) those same GAN densities, once the Algorithm has converged: G(z) matches P(x) perfectly and D(G(z)) outputs 0.5 everywhere

5.4 Adversarial Autoencoders

Adversarial Autoencoders combine some of the concepts explained in detail above but were not explored to the same theoretical depth. Essentially, AAEs were designed to 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 (usually multivariate Gaussian) distribution $p(z)$, a process which results in meaningful samples being generated from any sample, from any part of the prior (latent) space [86].

Here, the encoder 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 [86]. In addition, a discriminating neural network assesses whether the encoded representation it receives originates from the prior distribution or from the encoded data distribution.

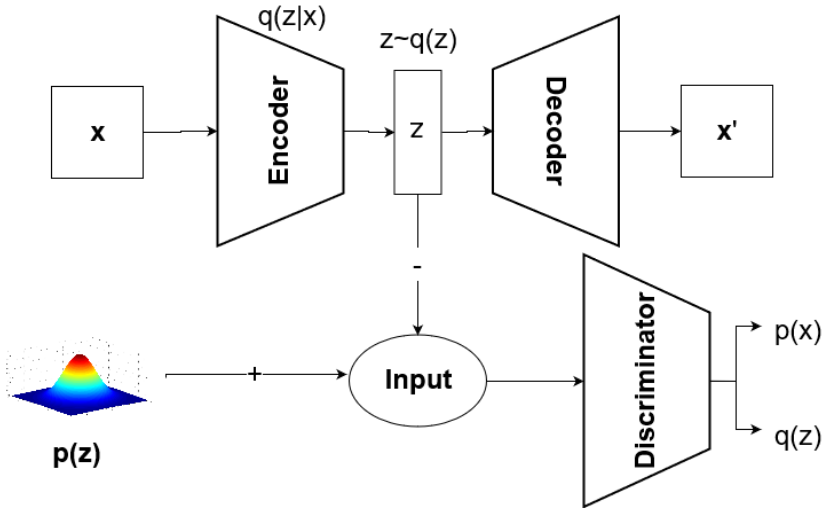


Figure 58: Simplified diagram of an Adversarial Autoencoder

6 Implementation: High Energy Physics Detector Simulations

6.1 Preamble: 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 (Appendix II: Figure 80), which was trained independently for each dataset.

Following training, predictions were run 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 59 for Geant4 data, Figure 66 for VAE data, Figure 70 for GAN data and Figure 76 for AAE data.

Lastly, in order to pertinently visualize images from each type of simulation, 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 (i.e. attained low $P(real)$ estimates) and which simulated samples were more realistic and therefore harder to distinguish from real data (i.e. attained low $P(real)$ estimates). These sampled images are shown in Figure 60 for Geant4 data, Figure 67 for VAE data, Figure 71 for GAN data and Figure 77 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 [87].

Figure 80 (Appendix II) 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 $\eta = 10^{-4}$, binary cross-entropy loss and a batch size of 32.

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.

6.2.1 Geant4 Configuration and Simulation

Please see [88] 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`).

6.2.2 Distinguishing Geant4-Simulated Data from Real Data

A convolutional neural network was able to distinguish Geant4-simulated pions (configured with environmental parameters from 2016/LHC16q/000265343) from *real* pions obtained during 2016/LHC16q/000265343 to a high degree of accuracy and therefore motivated the Deep Generative Modelling Section of this thesis.

Figure 59 shows the distribution of $P(real)$ estimates for Geant4 simulations and real data. Figure 60 shows example images from the Geant simulation that received increasing $P(real)$ predictions. Combining information from the two plots, one notices that the peak at $P(real)$~0.25 corresponds to empty images, where all pixels are equal to zero and thus makes this kind of image indistinguishable in terms of which distribution it is from. This is however perhaps an indication that Geant4 actually models the frequency of occurrence of such empty images in the true data distribution quite well.

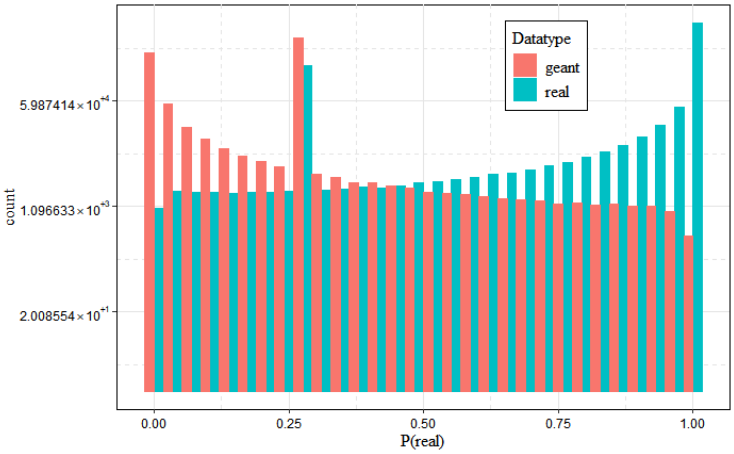


Figure 59: Distribution of $P(real)$ estimates for Geant4 vs Real data based on predictions from the discriminative neural network discussed above

6.2.3 An exploration of what might distinguish Geant4 from real data

What follows is an analysis of features between the Geant4-simulated and real datasets, to investigate what possibly differentiates them.

Firstly, the mean and standard deviation of ADC values for real and simulated datasets are not the same, i.e. $\mu_{sim} = 2.178$; $\mu_{real} = 4.527$ and $\sigma_{sim} = 11.989$; $\sigma_{real} = 17.995$. The distribution of simulated data's ADC values is slightly more skewed to the right than that of real data, i.e. $\gamma_{1_{sim}} = 19.170$; $\gamma_{1_{real}} = 17.391$. However, the maximum ADC value for both datasets is, of course, the same, i.e. $max_{sim} = max_{real} = 1023$.

Figure 61 shows the distribution of ADC values for simulated and real data, respectively.

Looking at the number of pads per image that didn't produce any data (i.e. the number of row-sums of the image that are equal to 0), Figure 62 shows the distributions for simulated and real data:

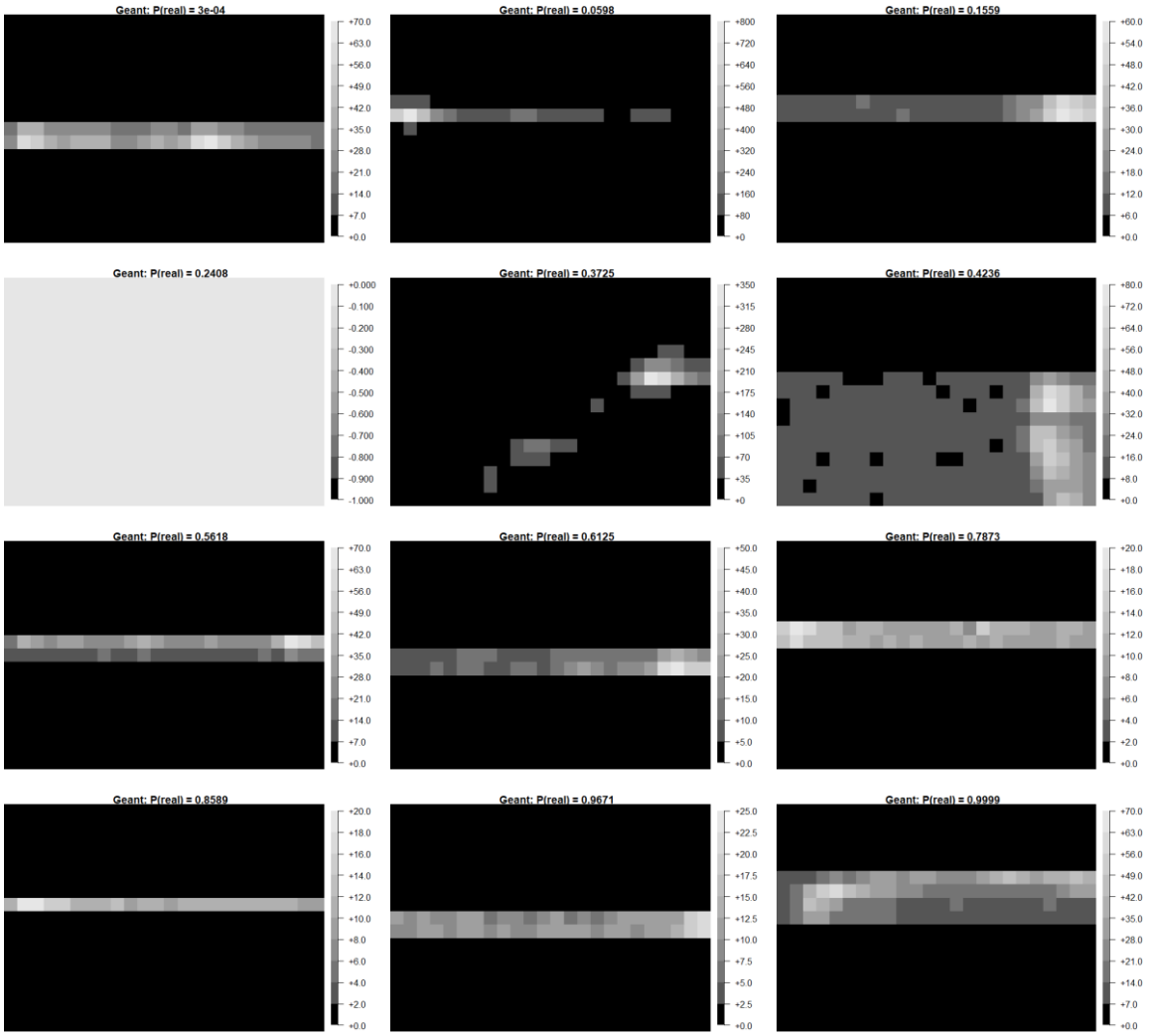


Figure 60: Twelve sampled Geant4-simulated pions arranged in order of increasing P(real) estimates

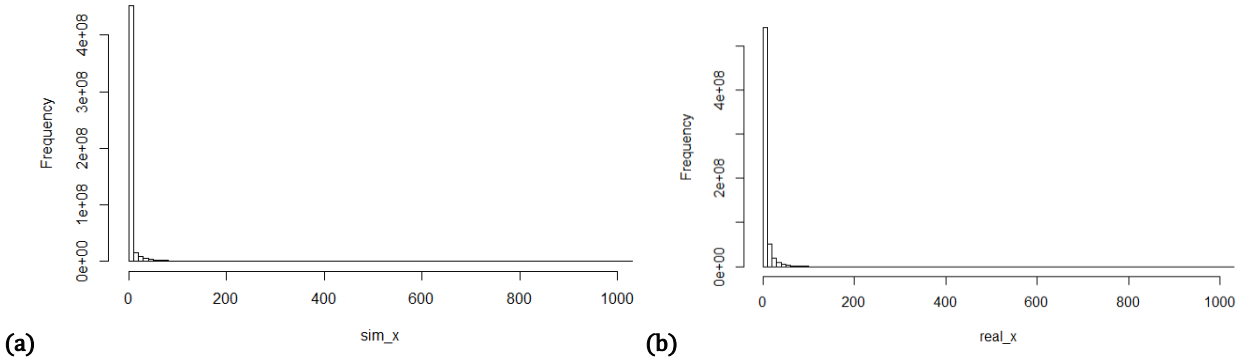


Figure 61: Distribution of ADC values for (a) Geant-simulated data and (b) real data

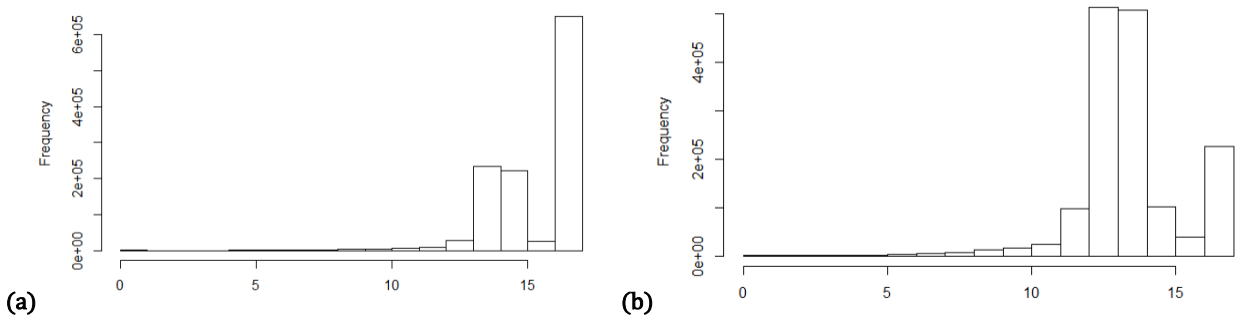


Figure 62: (a) Distribution of the number of pads with no data per image for simulated data (b) for real data

While these distributions are quite similar, one can see that real data is slightly more skewed towards the left and has far fewer instances of images which are completely devoid of signal, i.e. where all pixels in the image are equal to zero and therefore all 17 pads have no signal.

There are also noticeable differences in the distribution of mean ADC values per pad for real and simulated data (see Figure 63.

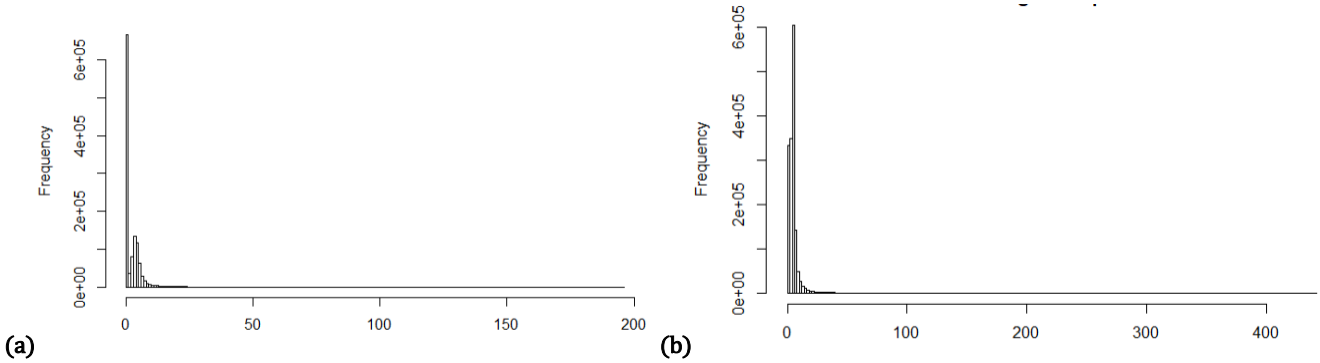


Figure 63: Distribution of mean ADC value per image for (a) simulated data, (b) real data

Finally, the mean pulse height as a function of time for Geant4 and real data (Figure 64) indicates that Geant4 data could perhaps be scaled by a determinable factor to make simulations match real data a bit better.

6.3 Implementation: Variational Autoencoders

6.3.1 Setup of the most successful Variational Autoencoder:

Most of the Variational Autoencoders in this project that were trained classically with Mean Squared Error (MSE) as the reconstruction loss, were found to result in images that were quite blurry (cf. Appendix III and note that most other VAEs prototyped resulted in similar "smeared-out" images).

The VAE that will be discussed below, was actually pre-trained, with images scaled to have pixel values in the range [-1,1] using MSE as the reconstruction loss at a very high learning rate ($\eta = 0.01$) for just 2 epochs, in order to get the configuration of the model's weights "in the right ballpark".

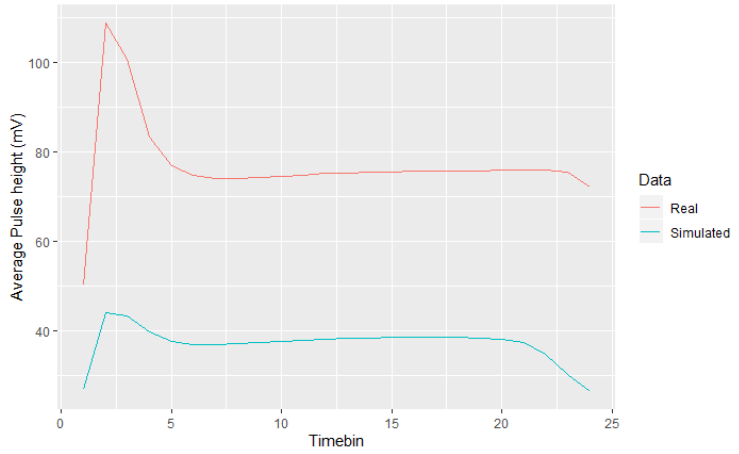


Figure 64: Average Pulse-height as a function of time-bin for Real and Geant4-simulated pion tracklets

After this pre-training stage, the output images looked as shown in Figure 65. Subsequent to this pre-training process, the same input images were then rescaled to be in the range [0,1] and the model's weight configuration was saved, while the reconstruction loss function was changed from MSE to Focal Loss and the learning rate was reduced drastically to $\eta = 10^{-8}$.

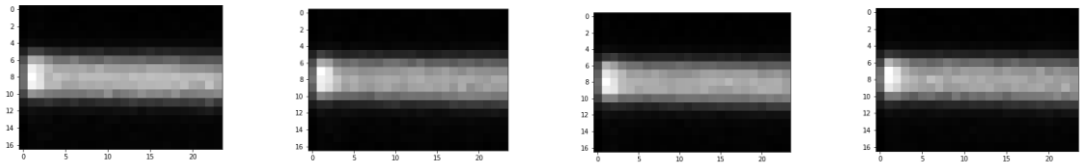


Figure 65: 3 example images from the pre-training stage, using MSE reconstruction loss and a high learning rate.

Model summary:

- N_{latent}=5
- Batch size=128
- Epochs = 164
- Sample size for each epoch = 500000

This model's architecture is shown in Appendix II: Figure 81.

6.3.2 Distinguishing Variational Autoencoder Data from Real Data:

Distribution of $P(real)$ estimates and examples of simulated images shown in Figure 66 and Figure 67.

6.3.3 Exploration of the VAE latent space

A visualization of points plotted along latent dimensions z_1, z_2 is shown in Figure 68. Areas of the latent space that result in more realistic simulated images result in clear clustering in sub-plot (c). Similar grouped clusters were seen at P(real)>=0.99 for all other combinations of axes. Latent space projections for other combination of $\boldsymbol{z_{i,j}}$ are not shown due to the inherent lack of interpretability of the latent space.

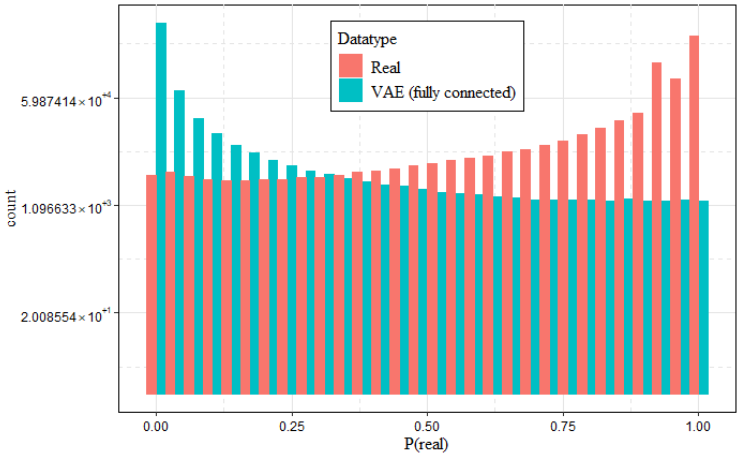


Figure 66: Distribution of $P(real)$ estimates for VAE-simulated and real data

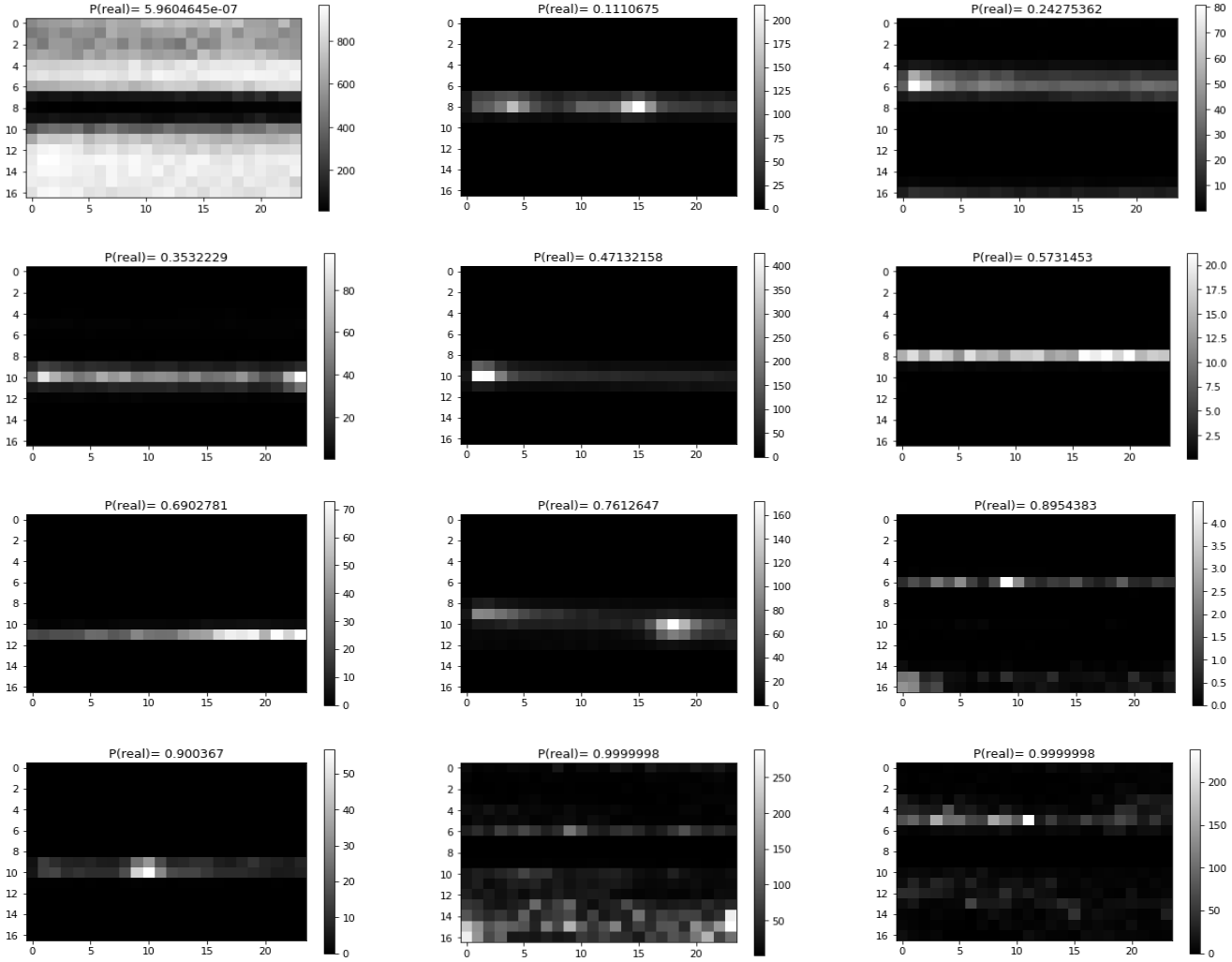


Figure 67: Twelve examples of VAE simulated tracklets, arranged in order of ascending $P(real)$ estimates

Similarly, Figure 69, shows a linear interpolation of the latent space dimensions z_1, z_2; where individual tracklet images are located at specific coordinates, with the other 3 latent dimensions set to zero. I.e. those z_1, z_2 coordinates at which an image is shown, along with 3

z-variables, each set to 0, are the initial input (x) values to Decoder function of the Variational Autoencoder, which maps to the specific images (y) shown there. Again, various other projections by combining other pairs of latent dimensions are possible (not shown here).

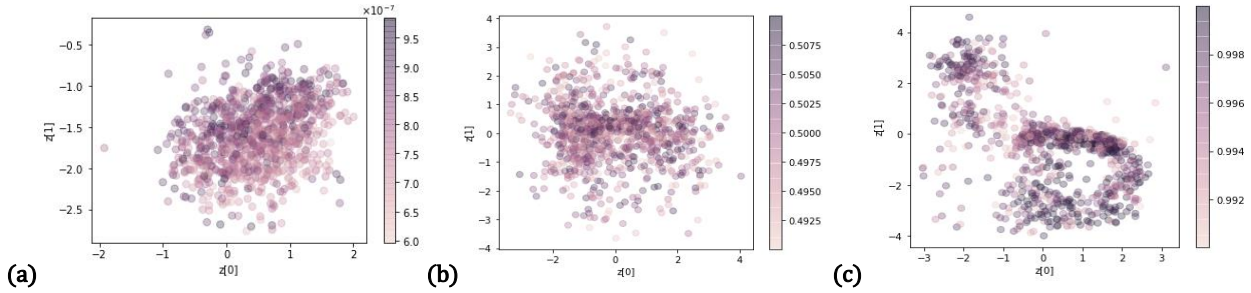


Figure 68: Scatterplot projections of simulated data points along latent space dimensions (z_0, z_1). Associated P(real) estimates for simulated tracklets produced from these latent points are represented on an independent colour scale for each plot. (a) $P(real) < 10^{-5}$, (b) $0.49 \leq P(real) \leq 0.51$, (b) $P(real) > 0.99$.

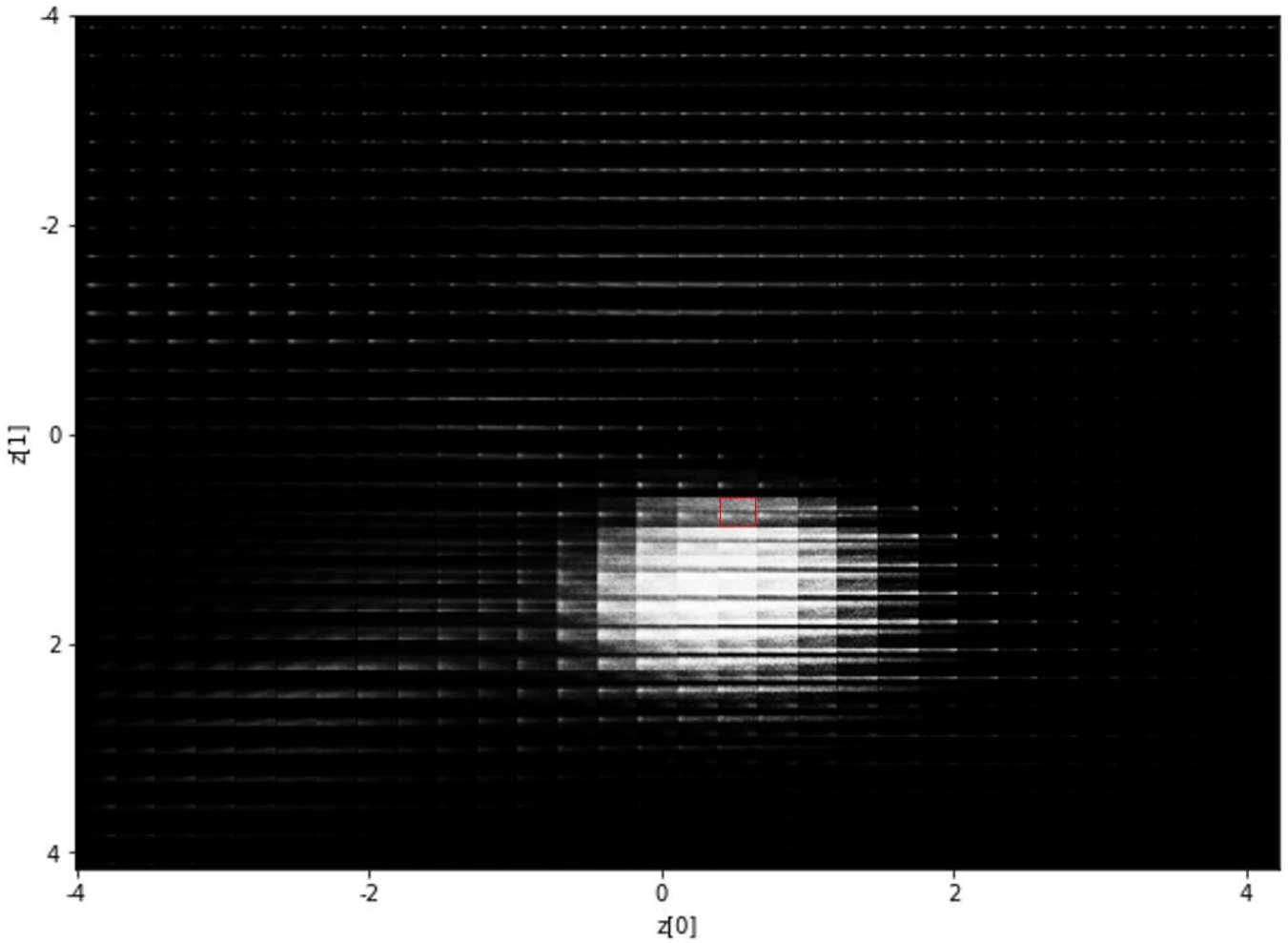


Figure 69: A linear interpolation of the latent space along latent dimensions (z_0, z_1), where the other three latent dimensions ($z_2, \dots, z_4$) are all set at z_i =0. The approximate boundaries around a single such tracklet image (near the centre of the plot) is highlighted in red to illustrate the concept better.

These visualizations of the latent space are promising investigations into how the latent space of a variational autoencoder can be understood (to some extent). Interestingly, something which was not done was to partition the input (true) dataset according to physical measurables like particle ID, momentum, the detector component it originates from, etc. Should the partitioned dataset then be passed through the encoder function of the VAE, one would be able to determine where, for example, a 2 GeV/c electron which results in a specific energy deposit, which was detected by a specific pad in a specific chamber, maps to in the latent space. Provided one has sufficient data for each combination of possible features of the particles one wants to simulate, one should be able to find specific areas in the latent space that those combinations of features map to, which would, in turn, allow one to give those latent encodings to the decoder function of a VAE to exactly specify the characteristics of the particle one wishes to simulate. Since the latent space is continuous, the inherent statistical fluctuations characteristic of physics measurements should also be captured (if the latent space is exploited properly), by generating unseen examples, which could possibly have resulted from two particles with the same characteristics. This could practically be enforced by sampling and adding to each element of the encoded vector handed to the decoder of the VAE, a corresponding element from a multidimensional random "noise" vector (ϵ).

6.3.4 Discussion: Variational Autoencoders

The motivation for the abovementioned pre-training procedure was mainly driven by the fact that MSE as the reconstruction loss resulted in blurry images, which appeared very similar, and also, by the fact that using MSE at a high learning rate was found to enable the model to output images that are in the "right ballpark" very quickly. In contrast, using Focal loss as the loss function results in an immensely high loss at the outset and also tends to produce almost "inverted" images, such as those found in the range $z_0 = 0.5, z_1 = 2$ of the latent space projection (Figure 69).

But, even though the optimizer, learning rate and the (scaled) image pixel range differed between the pre-training stage [-1,1] and the second stage [0,1]; when using Focal Loss as the reconstruction error after the pre-training stage, the model managed to recover from these differences rather quickly and eventually resulted in simulated images that were a lot more realistic, clear and variable than any other VAE built during this project.

Generative modelling is clearly not an exact science and quite often an unbelievable amount of seemingly strange strategies had to be employed in order to result in realistic simulations.

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 $\eta = 4 \times 10^{-4}$
    - Generator: Adam at Learning Rate $\eta = 2 \times 10^{-4}$
- 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: $b \sim TN(\mu = -2, \sigma^2 = 0.4, a = -2.2, b = -1.8)$

6.4.2 Distinguishing GAN-Simulated Data from Real Data

Distribution of $P(real)$ estimates and examples of simulated images shown in Figure 70 and Figure 71.

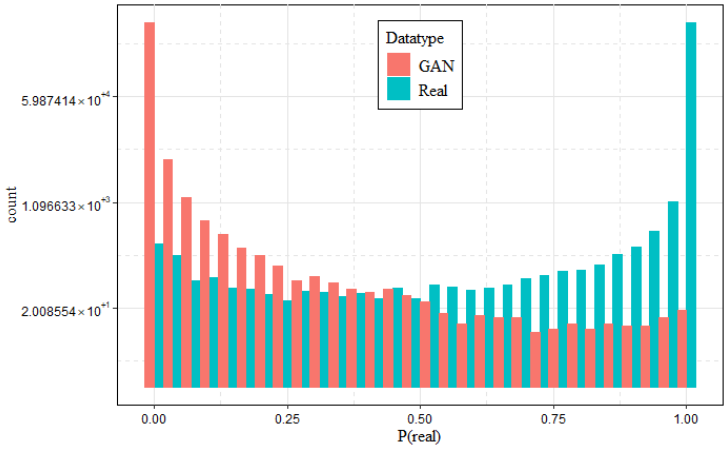


Figure 70: Distribution of $P(real)$ estimates for GAN and real data

6.4.3 Discussion: Generative Adversarial Networks

While various GAN architectures and variations on the GAN concept gave results that appeared vastly different in terms of "style", they all failed to capture the underlying distribution of the training data sufficiently. A possible reason for this is that the images used for training are quite small and have relatively little information compared to, for instance, images of human faces, where GANs have proven to be quite successful, but this is just speculation.

In summary, all of the GAN architectures developed in this project provided disappointing results, possibly due to the fact that GANs are quite hard to train since there are two neural networks pitted against each other in such a setup and if either of them become dominant during the training process, they can force the other into a parameter space which is not conducive to training either network properly.

An example of this can be seen in Figure 72, where after 21400 epochs, the Generator still outputs images that are seemingly just random noise, but upon closer inspection have similar features at similar coordinates across the different images (for some reason these nonsensical features resulted in a low error and therefore became prevalent). This phenomenon is known as Mode Collapse, which is said to occur when only a certain number of modes of the true distribution are captured by a GAN [89].

Batch Normalization is a strategy that ensures that a layer's outputs are normally distributed and therefore prevents the Generative component of the GAN to output images that look very similar, in Figure 73, one can see that the output images of a Bidirectional GAN (which employed this strategy) are highly dissimilar, but there is still a lot of noise around the main signal.

It is interesting to note that while one might expect GANs that employ convolutional layers to give better results than those making use of fully connected dense layers, Deep Convolutional GANs gave some of the most unrealistic simulated images (Figure 74). This could be due to the fact that convolutional layers introduce a prior that pertinent detected features are translationally invariant, i.e. they can appear anywhere in the image. While this might be a useful assumption during particle identification, it does not seem to hold for when simulating the data used in this project.

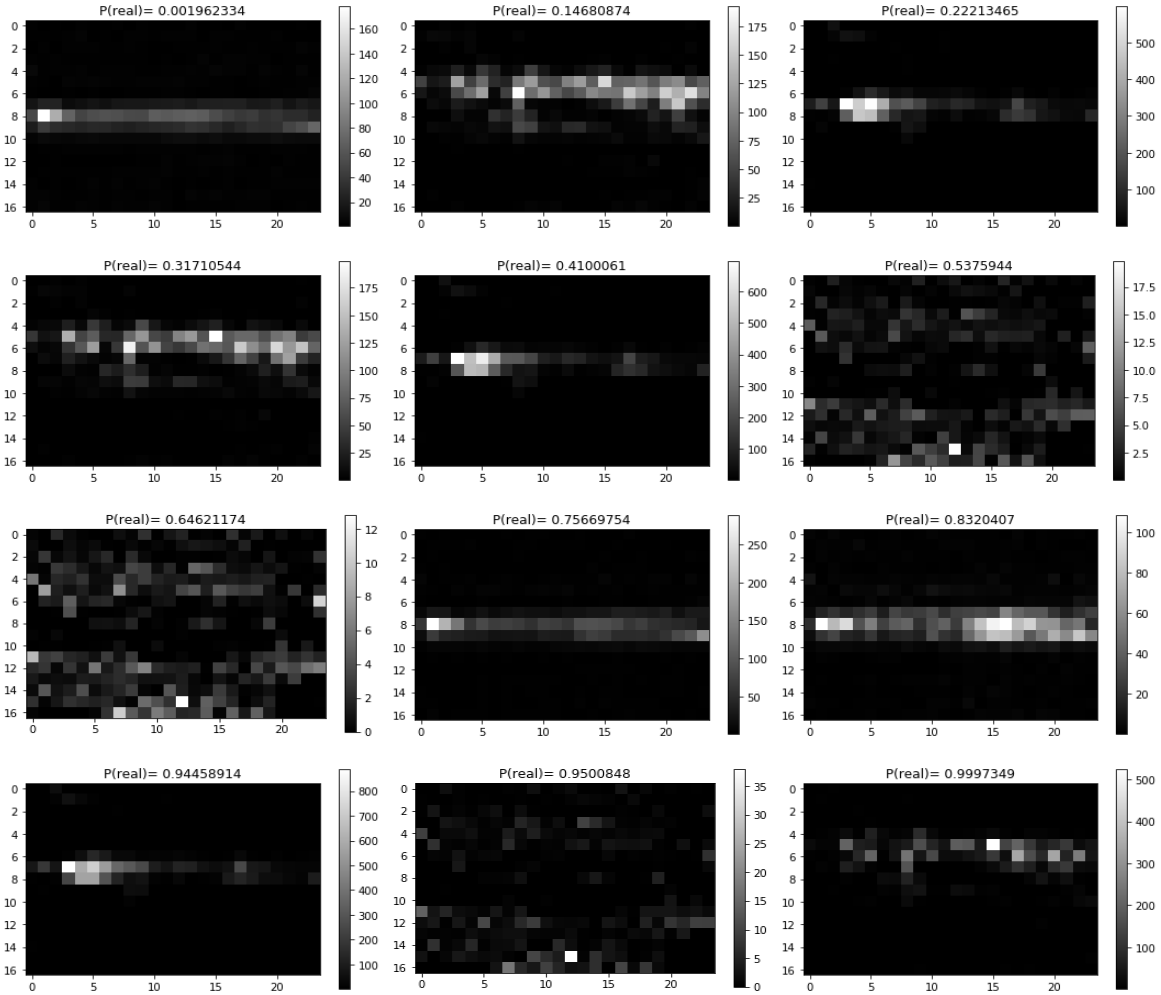


Figure 71: Twelve example GAN-simulated images, arranged in order of increasing $P(real)$ estimates

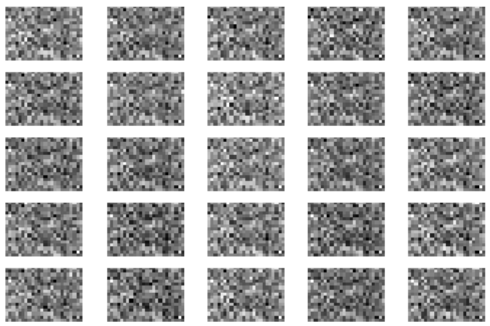


Figure 72: An illustration of Mode Collapse

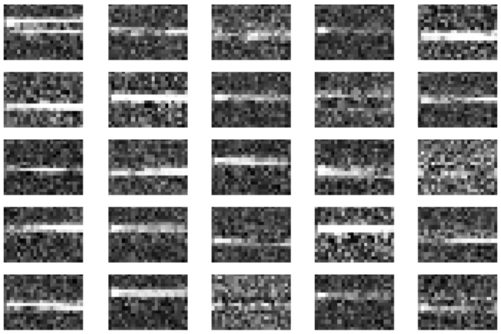


Figure 73: Illustrating the effect of Batch normalization to prevent output images from looking highly similar

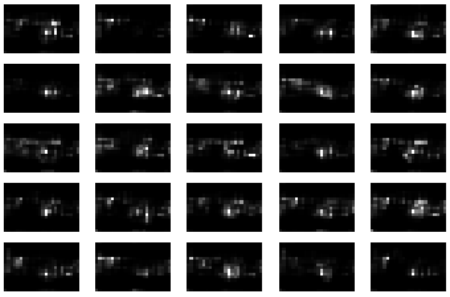


Figure 74: Illustrating how convolutional architectures in a GAN setup results in features that might exist in the training distribution, but do not appear in the right place

Training after a certain number of epochs also does not necessarily result in additional gains in performance, as can be seen in Figure 75, there is not much change in the output of a Least Squares GAN when training for an additional 293 000 epochs after 94 600 epochs, and training for an additional 61 400 epochs after that eventually results in images that actually look less realistic than those produced during earlier epochs.

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

Label Smoothing 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

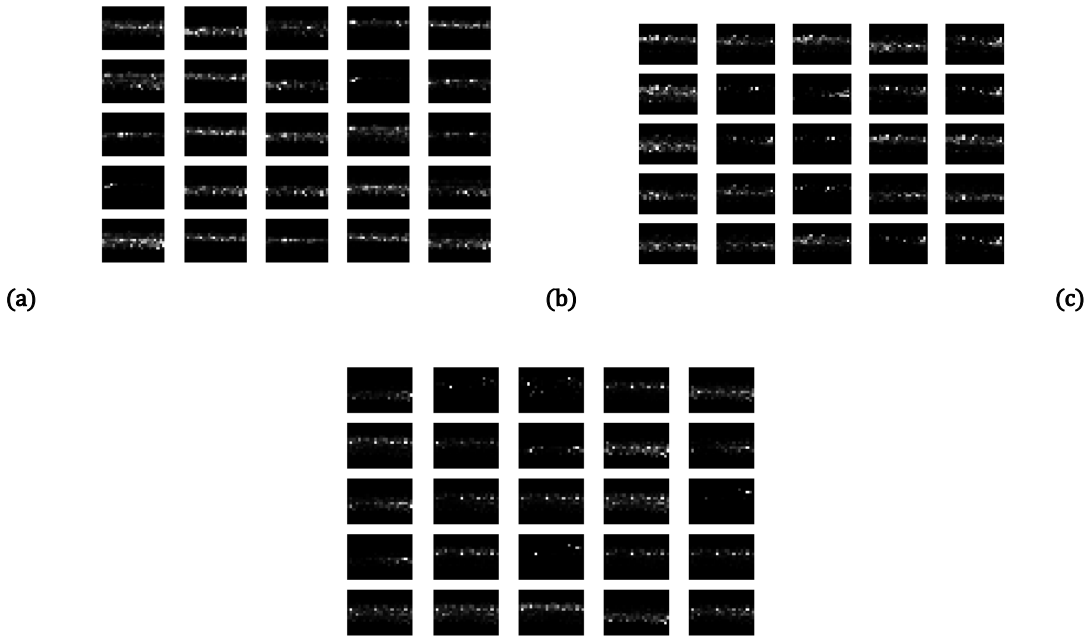


Figure 75: Least squares GAN (a) after 94 600 epochs, (b) after 387 600 epochs (c) after 449 000 epochs

6.5.2 Distinguishing AAE-Simulated Data from Real Data

Distribution of $P(real)$ estimates and examples of simulated images shown in Figure 76 and Figure 77.

6.5.3 Discussion: Adversarial Autoencoders

The relative success of AAEs probably has a lot to do with their proper enforcement of the latent space, allowing the generation of meaningful samples from anywhere within said latent space. These models were implemented quite late during this project and were experimented with a lot less than VAEs and GANs, but they do seem a lot more promising for this use-case, especially when compared to GANs.

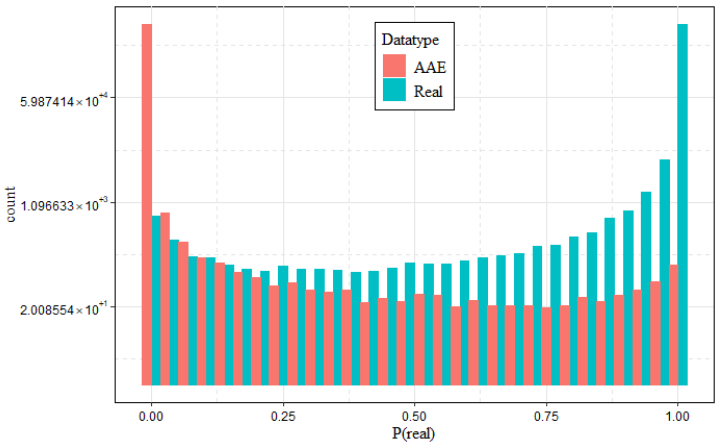


Figure 76: Distribution of $P(real)$ estimates for AAE and real data

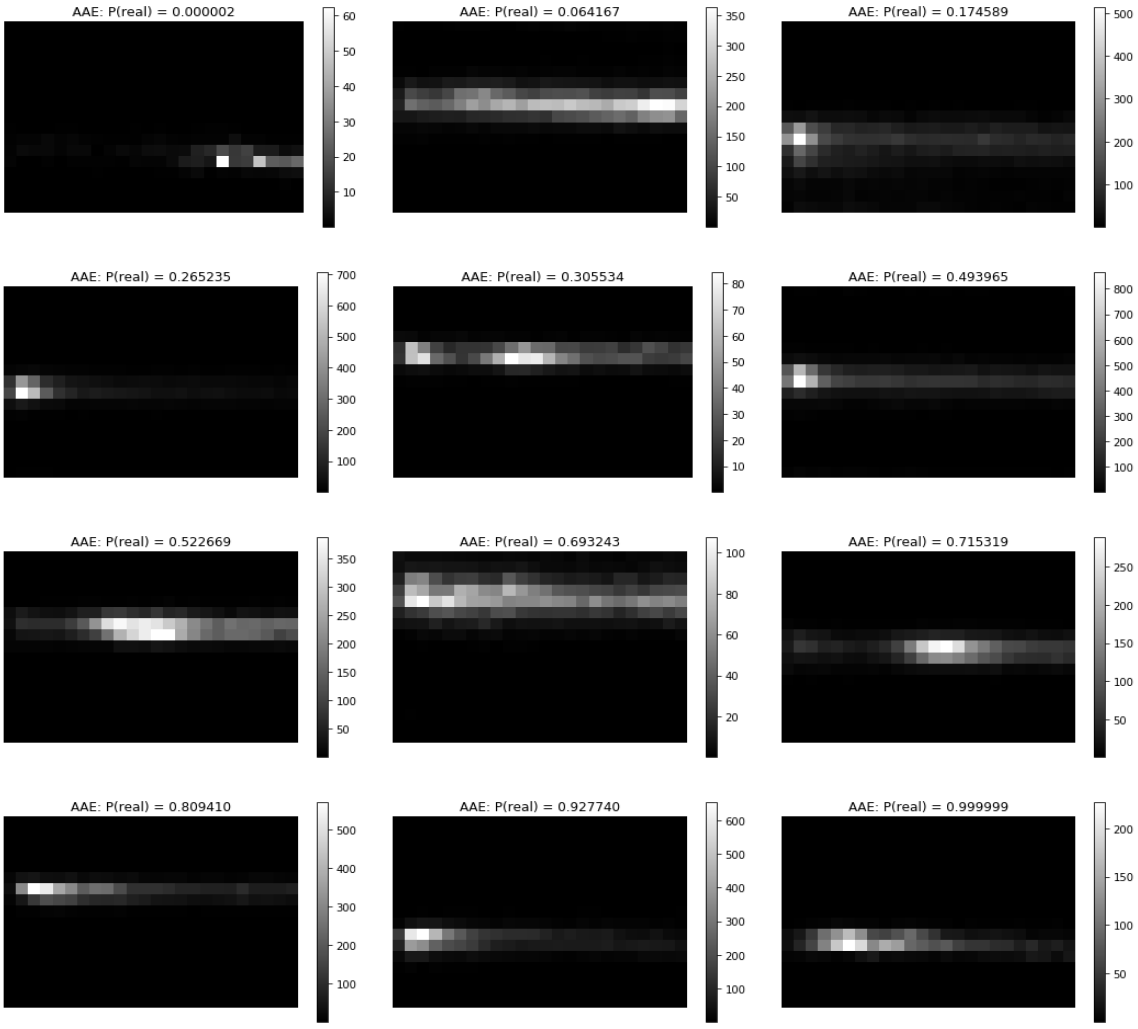


Figure 77: Twelve example AAE-simulated images, arranged in order of increasing $P(real)$ estimates

6.5.4 Chapter Discussion

6.5.4.1 Challenges faced during deep generative modelling

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 be able 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 than they are when training a classification model since realistic images can sometimes be generated, even when loss appears to be quite high. Similarly, classification accuracy and loss cannot truly be calculated when techniques such as label smoothing are used. It was also found that accuracy and loss metrics can seem to "remain in balance" while improving, i.e. even though these metrics are no longer changing, the images are becoming more realistic nonetheless.

There are some pitfalls to training locally, as opposed to training on a server, 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 the 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. For most of the models developed in this project, the full dataset was not used, due to local hardware limitations.

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.

Tweaking a single hyperparameter to see its effect would be ideal, but time constraints force one to often change multiple hyperparameters and architectures simultaneously. Using one's intuition as to what the effect of each of a combination of changes could possibly give result to, and being able to debug a Generative set-up at the hand of simulated images and knowledge of how adjusting a certain parameter (or set of parameters) in a certain way could potentially fix issues in the simulation process, becomes more important.

6.6 Chapter Conclusions

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 slightly 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 [90] (such as label smoothing and using different optimizers for the generator and the 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

The CNNs developed in this thesis (trained on uncalibrated data) produced pion efficiency results comparable to previous LQ1D and LQ2D methods (which were performed on properly calibrated input data), but their performance was much worse than LQ7D and fully connected neural networks currently in use by ALICE collaborators in the TRD working group.

While convolutional neural networks are generally accepted as being more conducive towards the accurate classification of array-like data than fully connected neural networks [91], 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. Or if data is simply calibrated properly before being used for neural network training.

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. As mentioned, this could be partly explained by the relative number of 2D CNNs developed, compared to other model types. A general observation is that larger neural networks (models with more trainable parameters, which therefore have a higher capacity to "learn"/"memorise") were on the whole more successful than smaller models, but that there is a limit to how much the size of the network can improve performance. In addition, larger networks are prone to overfitting and therefore need to be properly regularized.

The use of the focal loss function and incremental training per momentum bin resulted in immense gains over other methods explored. This loss function allowed for use of the full set of data (excluding empty images, which possibly also resulted in some gains), without having to downsample or upsample in any way, because of the much lower loss which results from well-classified examples and therefore prevented the neural network from becoming biased to the dominant class. The number of examples that this enabled the neural network to be trained on most probably also had a big influence on the success of the final and most successful model discussed for particle identification. It should also be noted that only one model was built using this approach and that building many models with the same architecture and hyperparameters on different subsets of the data could give an indication as to the potential positive influence of random statistical fluctuations due random weight initialisation, but looking at the other factors of variation between this model and other models, it seems more likely that the focal loss function and exposure to many more training examples as well as, possibly, the incremental training procedure was what set this model apart from the rest.

Ten-fold cross-validation was performed on tree-based methods, which was not done for other models, and which could potentially have resulted in additional gains if it was. The two tree-based methods' results were surprisingly good when taking into account that they were not fine-tuned or explored further than using the default parameters.

There is a potential limit to the amount of information that can possibly be extracted from the TRD in isolation, regarding a specific particle ID, regardless of the sophistication of the classifier used.

7.1.1 Suggestions for Future Work

At the hand of these results, one could posit that CNNs or LSTMs (or a combination) could potentially 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 now also supported on GPU (since ROOT 6.14) [92].

However, 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*.

Cross-validation and an exploration of tree-based methods, as well as more sophisticated hyperparameter searches and ensembling strategies, could potentially result in slight improvements in pion efficiency, but this statement is mainly speculative.

7.2 High Energy Physics Detector Simulations

7.2.1 Major findings

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. Results from VAEs were also quite promising and the lack of success with the GAN approach to simulation could be attributed to the limited time available during a one-year project. In fact, all of these models could be fruitful avenues to explore in the future.

A particularly exciting finding from the visualizations conducted on the latent space of a VAE is the potential guidance such explorations can provide when moving towards practically implementing them as part of an actual physics detector simulation. This would definitely have to involve understanding where specific particles, with specific measurable differences, tend to map to (on average) in the latent space representation for each specifiable combination of variables. Once that is determined and there is some confidence that the latent space is properly utilised, i.e. that meaningful samples (samples which could realistically arise in reality) are generated from anywhere in the latent space; that would enable one to specify exactly what one wishes to simulate and additionally, one would be able to model statistical fluctuations around the expected signal for a specific combination of features by adding a small random noise vector (ϵ) to the latent vector z, before handing it to the decoder function of the generative model. These concepts should hold for any generative model which samples from a latent space, and various combinations of latent variable models could potentially be combined towards this purpose. Once these conditions are met, interfaces to other components of the detector chain (e.g. to AliROOT) would probably centre around the methodology behind the currently used interfaces to the simulation component. These interfaces would need to be satisfied in a standard way and this would most likely be one of the guiding principles as to how these deep generative models would need to be designed from the outset. Should all of this be successful, significant speed-up over Monte Carlo simulations is expected.

7.2.2 Additional Observations

In general, generative models were found to be much harder to work with than classifying neural networks. A lot of experiments generally failed before something worked reasonably well, for each type of model prototyped. That being said, the experience gained during this project points towards some simple guidelines: it's best to start with something very simple that produces images that look

even slightly better than random noise and *then* adding additional layers or nodes or strict regularization measures. Starting with a very complex generative modelling strategy from the outset will also make debugging much harder.

There are almost no limits to the ways in which generative models can be modified and made to work differently. The Keras functional API made this process a lot easier than it would have been if one had to implement everything in native Tensorflow.

Intuition becomes an important aid in this process and can only be obtained via experience, but being involved in generative modelling was a far more valuable learning experience for truly understanding concepts in deep learning, than building classifiers was (although without having built many classifiers first, it wouldn't have been possible to attain much success with the more advanced VAEs, GANs and AAEs built in this project).

The fact that visible output can be qualitatively compared to what real TRD digits data could look like, as well as being able to assess how changing a set of parameters influences said visual output are the main features that allow generative models to be a good teaching tool.

It should also be said that frustration with these models drove further experimentation with increasingly creative modifications of the models and their training procedures, cf. using a truncated normal distribution with a negative mean in the bias term of the output layer to force that output layer to be closer to zero, or pretraining a VAE with different dataset-scaling, learning rate and reconstruction loss settings during two different stages.

A somewhat personal observation is that the idea underpinning VAEs is one of the most beautiful concepts in Statistics, but a somewhat less subjective statement is that VAEs can be used for a lot more than just generating images, e.g. for outlier detection or for the optimization of factory processes (by finding regions in the latent space that result in fewer errors or in higher efficiency during production of goods). There are also potential implementations for drug discovery and development, where VAEs can be used as part of a process to find artificial medicinal ligands that fit into active sites of receptors.

7.2.3 Suggestions for Future Work

Future work should perhaps 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, as was done in [1]). While this technique was attempted to some extent 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).

7.3 General Concluding Remarks

Making use of Tensorflow on GPU, via NVIDIA's CUDA libraries (assuming one has access to a GPU with sufficient specifications), would be a useful way to speed up the training of most of the models discussed, specifically those making use of convolution operations.

Besides particle identification and deep generative modelling for detector simulations, 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).

To this end, there is an active community of researchers applying Machine Learning to various Aspects of CERN [93] and as part of this Masters research, the author was lucky enough to attend the most recent Interexperimental Machine Learning Workshop hosted at CERN [94], which is a yearly event, including talks and workshops by CERN physicists involved in machine learning research and various high-profile industry experts.

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] The CMS Collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC," *Physics Letters B,* vol. 716, p. 30, 2012.

[6] The ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," *Physics Letters B,* vol. 716, pp. 1-29, 2012.

[7] F. Englert and R. Brout, "Broken Symmetry and the Mass of Gauge Vector Mesons," *Physical Review Letters,* vol. 13, no. 9, pp. 321-323, 1964.

[8] P. Higgs, "Broken Symmetries and the Masses of Guage Bosons," *Physical Review Letters,* vol. 13, no. 16, pp. 508-509`, 1964.

[9] G. Guralnik, C. Hagen and T. Kibble, "Global Conservation Laws and Massless Particles," *Physical Review Letters,* vol. 13, no. 20, pp. 585-587, 1964.

[10] Wikimedia Commons, "File:Meson nonet - spin 1.svg --- Wikimedia Commons, the free media repository," 2014. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Meson_nonet_-_spin_1.svg&oldid=141026885. [Accessed 2019 October 21].

[11] Wikimedia Commons, "File:Meson nonet - spin 0.svg --- Wikimedia Commons, the free media repository," 2015. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Meson_nonet_-_spin_0.svg&oldid=182154336. [Accessed 21 October 2019].

[12] Wikimedia Commons, "File:Baryon-decuplet-small.svg --- Wikimedia Commons, the free media repository," 2019. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Baryon-decuplet-small.svg&oldid=378029923. [Accessed 21 October 2019].

[13] Wikimedia Commons, "File:Baryon-octet-small.svg --- Wikimedia Commons, the free media repository," 2018. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Baryon-octet-small.svg&oldid=324287830. [Accessed 21 October 2019].

[14] Tanabashi, M; et al., "Particle Data Group," *Phys. Rev. D,* vol. 98, p. 030001, 2018.

[15] Beringer, J; et al., "Particle Data Group," *Phys. Rev. D,* vol. 86, p. 010001, 2012.

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

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

[18] A. Maire, "Multi-strange baryon production at the LHC in proton-proton collisions with the ALICE experiment (PhD thesis)," *University of Strasbourg,* 2011.

[19] A. Maire and J. Charles, "Phase transition to QGP matter : confined vs deconfined matter," *For the benefit of the ALICE Collaboration,* 2015.

[20] The Stephen Hawking Center for Theoretical Cosmology, "Universe Chronology," [Online]. Available: http://www.ctc.cam.ac.uk/images/contentpics/outreach/cp_universe_chronology_large.jpg. [Accessed 5 June 2019].

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

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

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

[24] 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.

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

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

[27] S. Dailler, "Map of the Geneva region and of the LHC.," 1997. [Online]. Available: https://cds.cern.ch/record/842399?ln=en. [Accessed 20 November 2019].

[28] V. Frigo, "LHC map in 3D," 1997. [Online]. Available: https://cds.cern.ch/record/842700. [Accessed 20 November 2019].

[29] V. Frigo, "LHC Structure," 1997. [Online]. Available: https://cds.cern.ch/record/842611. [Accessed 20 November 2019].

[30] E. Mobs, "The CERN accelerator complex - August 2018," 2018. [Online]. Available: https://cds.cern.ch/record/2636343/files/CCC-v2018-print-v2.jpg?subformat=icon-1440. [Accessed 26 January 2019].

[31] 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.

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

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

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

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

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

[37] ALICE, "ALICE Homepage," [Online]. Available: http://alice.web.cern.ch/. [Accessed 21 February 2019].

[38] 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.

[39] J. Thäder, "ALICE Figure Repository: General ALICE Cross-Section with L3 Magnet," 2012. [Online]. Available: https://alice-figure.web.cern.ch/node/3394. [Accessed 5 December 2019].

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

[41] 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.

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

[43] University of Heidelberg: Physics Department, "TRD Hardware Components and Glossary," [Online]. Available: https://alice.physi.uni-heidelberg.de/trd-db/info/trd.html. [Accessed 5 December 2019].

[44] The ALICE Collaboration, "Rapidity and transverse momentum dependence of inclusive J/ψ production in pp colisions at $\sqrt{s}$= 7 TeV," *Physics Letters B,* vol. 704, pp. 442-455, 2011.

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

[46] ALICE, "More details on the ALICE TRD," [Online]. Available: http://alice.web.cern.ch/detectors/more-details-alice-trd. [Accessed 5 December 2019].

[47] J. Klein, "Jet Physics with A Large Ion Collider Experiment at the Large Hadron Collider (PhD thesis)," 2014.

[48] 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.

[49] A. Wilk, *Particle Identification Using Artificial Neural Networks with the ALICE Transition Radiation Detector (PhD Thesis),* Münster, Germany: Westfälische Wilhelms-Universität Münster (WWU) - Institut für Kernphysik (KP), 2010.

[50] M. Kroesen, *Electron Identification with the ALICE TRD using Multivariate Analysis Techniques (Masters Thesis),* Heidelberg, Germany: Physikalisches Institut, Universitaet Heidelberg, 2017.

[51] L. Feldkamp, *Reconstruction of Beauty Jets in Proton-Proton Collisions at sqrt(s) = 7 TeV with ALICE (PhD Thesis),* Münster, Germany: Westfälische Wilhelms-Universität Münster (WWU) - Institut für Kernphysik (KP), 2018.

[52] X. Lu, "Exploring the performance limits of the ALICE Time Projection Chamber and Transition Radiation Detector for measuring identified hadron production at the LHC (PhD thesis)," *Ruprecht-Karls-Universität Heidelberg - Physikalisches Institut,* 2013.

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

[54] "ROOT 5 Reference Guide," [Online]. Available: https://root.cern/root/html534/ClassIndex.html.

[55] "ROOT 6 Reference Guide," [Online]. Available: https://root.cern/doc/v616/.

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

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

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

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

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

[61] Wikimedia Commons, "File:ArtificialNeuronModel english.png --- Wikimedia Commons, the free media repository," 2017. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:ArtificialNeuronModel_english.png&oldid=233886112. [Accessed 06 09 2019].

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

[63] P. Sibi, S. A. Jones and P. Siddarth, "Analysis of Different Activation Functions using Back Propagation Neural Networks," *Journal of Theoretical and Applied Information Technology,* vol. 47, no. 3, pp. 1264-1268, 2013.

[64] A. LeNail, "NN-SVG," [Online]. Available: https://alexlenail.me/NN-SVG/.

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

[66] U. Griffo, "Github," [Online]. Available: https://github.com/umbertogriffo/focal-loss-keras. [Accessed 21 September 2019].

[67] C. G. Viljoen, "Github Gist," [Online]. Available: https://gist.github.com/PsycheShaman/ea39081d9f549ac410a3a8ea942a072b. [Accessed 30 September 2019].

[68] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Published as a conference paper at the 3rd International Conference for Learning Representations*, San Diego, 2015.

[69] S. Saha, "Towards Data Science," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 26 November 2019].

[70] C. G. Viljoen, "Github," [Online]. Available: https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/ROOT/trdML-gerhard. [Accessed 4 December 2019].

[71] T. Dietel, 2016. [Online]. Available: https://github.com/tdietel/trdpid.

[72] P. Christakoglou, J. F. Grosse-Oetringhaus and C. Klein-Boesing, 2010. [Online]. Available: https://alice-offline.web.cern.ch/sites/alice-offline.web.cern.ch/files/uploads/AnalysisTrain/AliAnalysisTaskPt.cxx.

[73] C. Finlay, " (Honours Thesis)," 2017. [Online]. Available: https://github.com/chrisfinlay/trdML [this repository has been archived].

[74] A. Barreiros, "How accurately can a neural network be trained to use data from the TRD detector in ALICE to identify electrons from pions (Third Year Project)," 2018. [Online]. Available: https://github.com/ForeverANoob/trdML [this repository has been archived].

[75] F. J. Valverde-Albacete, "100% Classification Accuracy Considered Harmful: The Normalized Information Transfer Factor Explains the Accuracy Paradox," *PLoS ONE,* vol. 9, no. 1, p. e84217, 2014.

[76] C. G. Viljoen, "Github," [Online]. Available: https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/Particle_Identification. [Accessed 4 December 2019].

[77] "Scikit Learn," [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/utils/class_weight.py. [Accessed 2 November 2019].

[78] J. J. H. Wilkison, "On the Application of Machine Learning Techniques to Particle Identification in the Transition Radiation Detector of the ALICE Experiment (Honours Thesis)," 2019.

[79] S. Sharma, "Towards Data Science," [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed 4 December 2019].

[80] Q. Liu, F. Zhou, R. Hang and X. Yuan, "Bidirectional-Convolutional LSTM Based Spectral-Spatial Feature Learning for Hyperspectral Image Classification," *Remote Sensing,* vol. 9, no. 12, p. 1330, 2017.

[81] Y. Pachmayer, "[Personal communication via email]".

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

[83] 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.

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

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

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

[87] C. G. Viljoen, "Github," [Online]. Available: https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/Latent_Variable_Models. [Accessed 4 December 2019].

[88] C. G. Viljoen, "Github," [Online]. Available: https://github.com/PsycheShaman/MSc-thesis/tree/master/Code/ROOT/trdpid/sim. [Accessed 4 December 2019].

[89] A. Srivastasa, L. Valkov, C. Russel and M. U. Gutmann, "VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning," *31st Conference on Neural Information Processing Systems,* 2017.

[90] S. Chintala, "How to Train a GAN? Tips and tricks to make GANs work," [Online]. Available: https://github.com/soumith/ganhacks. [Accessed 4 December 2019].

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

[92] 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.

[93] "Interexperimental LHC Machine Learning Working Group," [Online]. Available: https://iml.web.cern.ch/. [Accessed 4 December 2019].

[94] "3rd IML Machine Learning Workshop," [Online]. Available: https://indico.cern.ch/event/766872/. [Accessed 4 December 2019].

[95] "MonALISA Repository for ALICE," [Online]. Available: https://alimonitor.cern.ch. [Accessed `4 December 2019].

[96] C. G. Viljoen, "CERN Gitlab," 2019. [Online]. Available: https://gitlab.cern.ch/cviljoen/msc-thesis-data. [Accessed 4 December 2019].

9 APPENDICES

APPENDIX I: RUN NUMBERS ANALYZED (FROM LHCQ16)

- 000265 309
- 000265 332
- 000265 334
- 000265 335
- 000265 336
- 000265 338
- 000265 339
- 000265 342
- 000265 343
- 000265 344
- 000265 377
- 000265 378
- 000265 381
- 000265 383
- 000265 385
- 000265 388
- 000265 419
- 000265 420
- 000265 425
- 000265 426
- 000265 499

APPENDIX II: DEEP LEARNING ARCHITECTURE DIAGRAMS

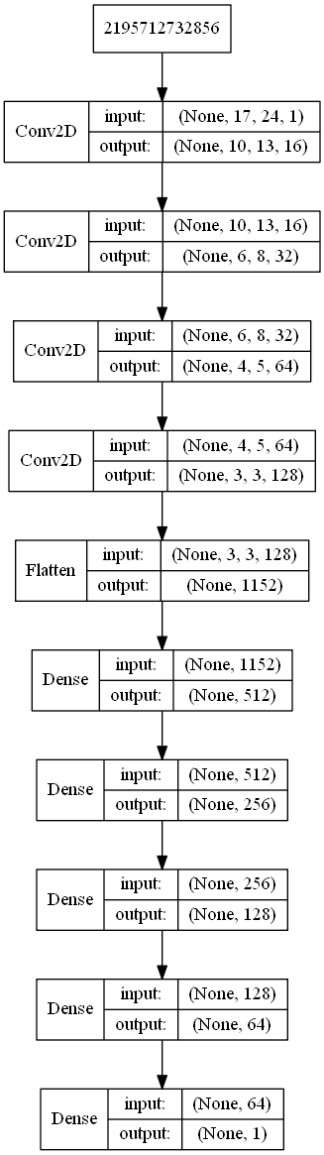


Figure 78: Most successful particle Identification Model Architecture (Stage 2 of Model Building)

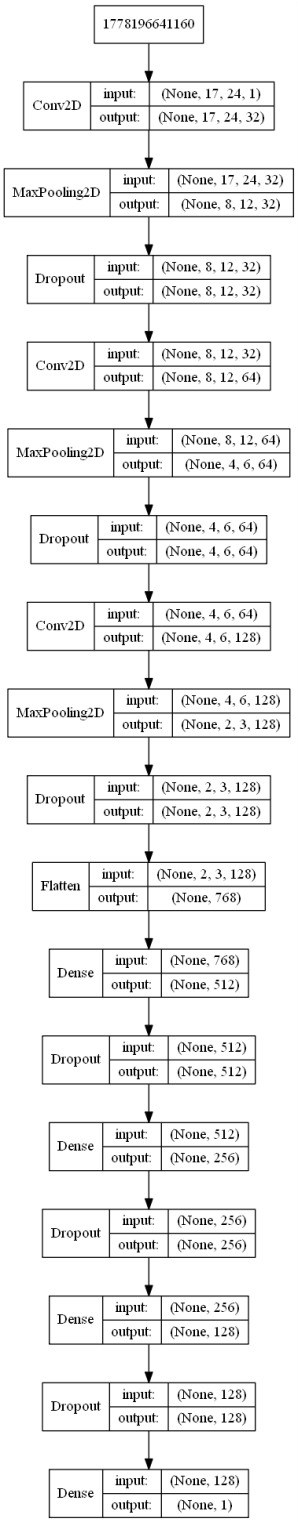


Figure 79: Most successful particle identification neural network (Stage 3 of Model Building), incrementally trained on increasing momentum ranges, using Focal Loss as the objective function to be optimized, data used to train this model was not down-sampled or up-sampled.

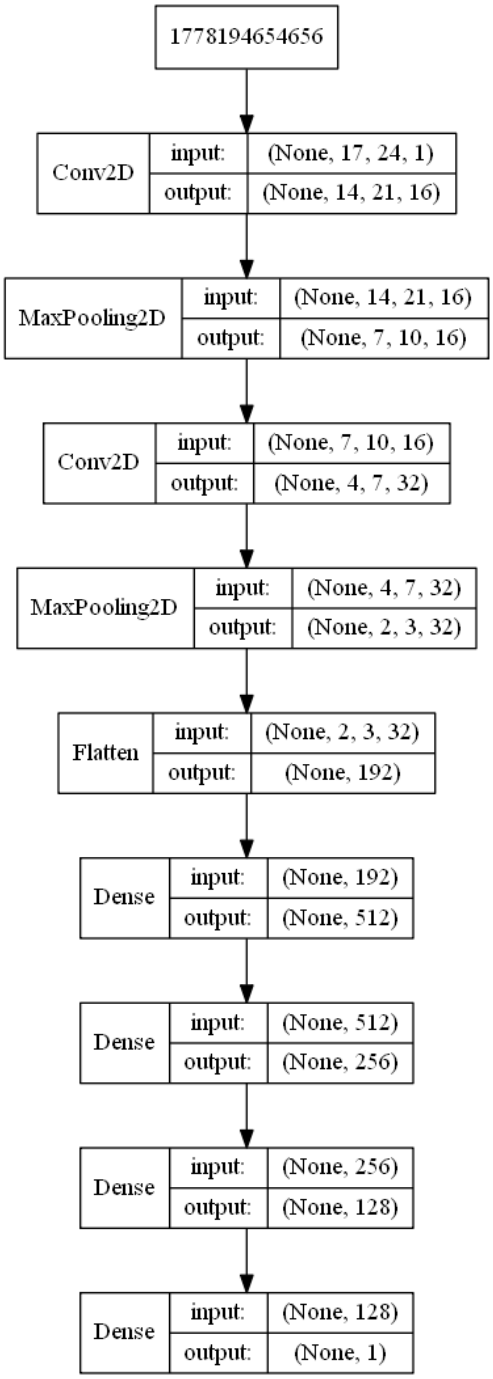


Figure 80: 2D CNN used to individually discriminate each type of simulated data from real data

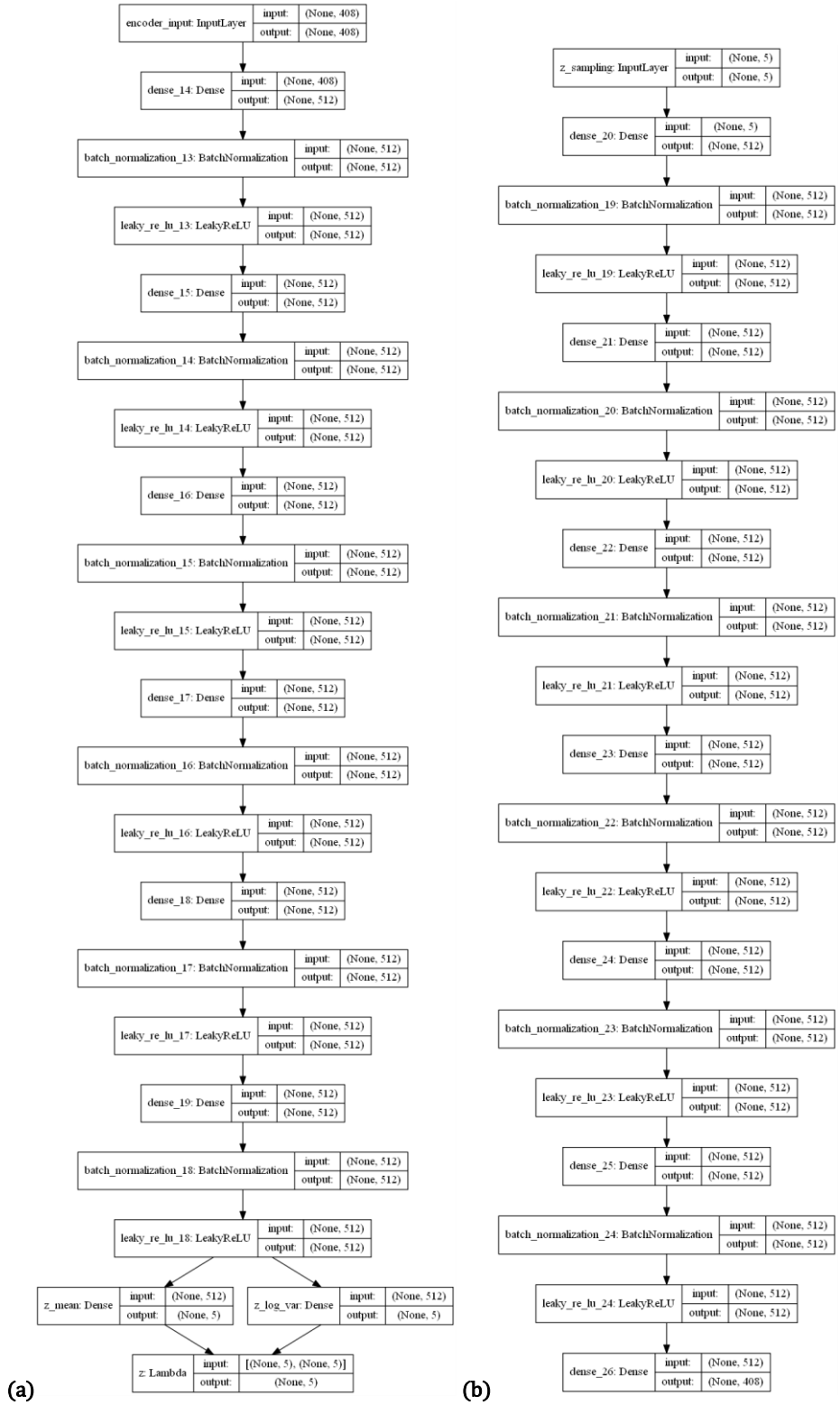


Figure 81: Most successful VAE architecture (a) Encoder, (b) Decoder

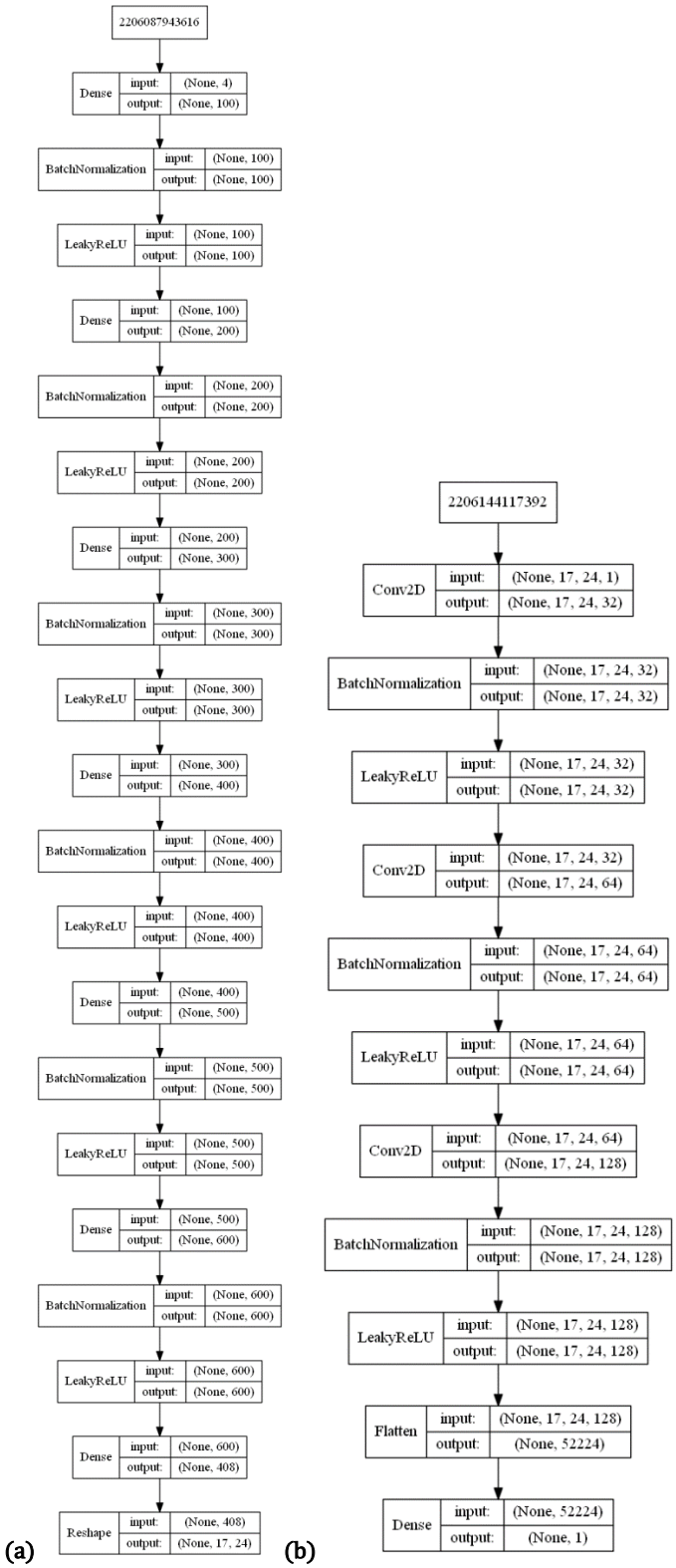


Figure 82: Most successful GAN architecture: (a) Generator, and (b) Discriminator

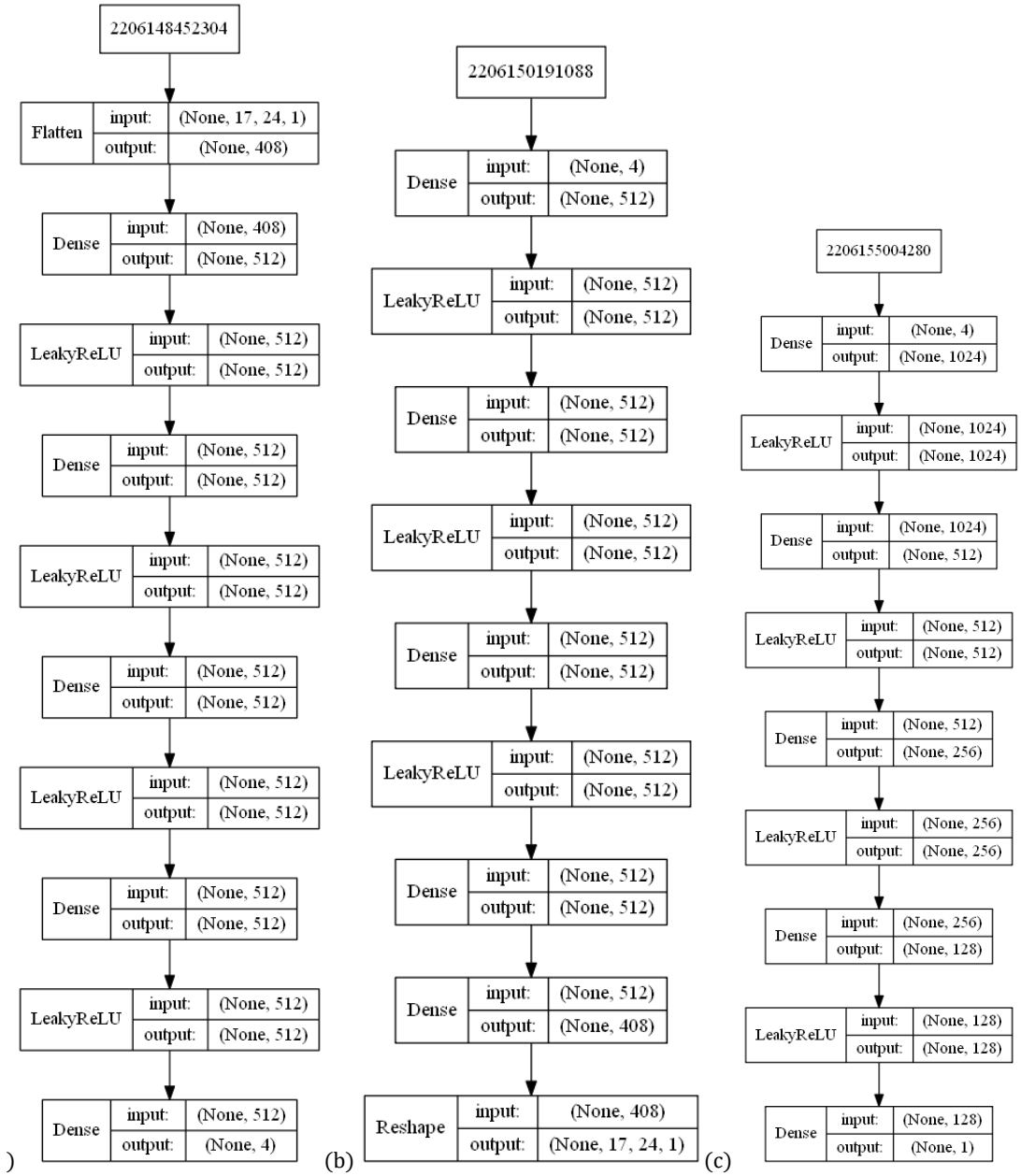


Figure 83: Most successful Adversarial Autoencoder architecture. (a) Encoder (b) Decoder (c) Discriminator

APPENDIX III: LESS SUCCESSFUL GENERATIVE MODELS

Autoencoders

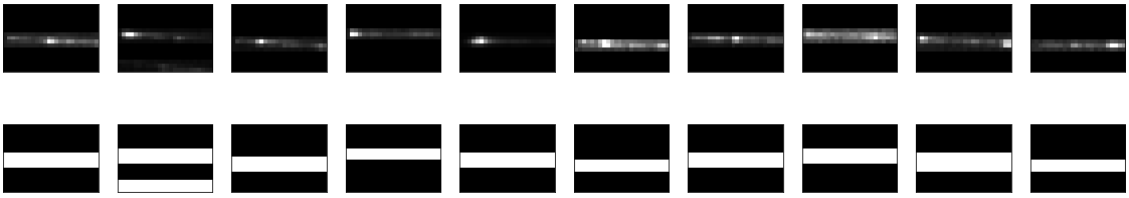


Figure 84: Original images shown in the top row, with their reconstructed auto-encoder versions beneath them.

A Variational Autoencoder with Convolutional Architecture:

- $n_{latent} = 100$
- Batch size = 64
- Optimizer: Adam, Learning rate $\eta = 10^{-4}$
- Epochs = 1 000 000
- Sample size at each epoch = 64

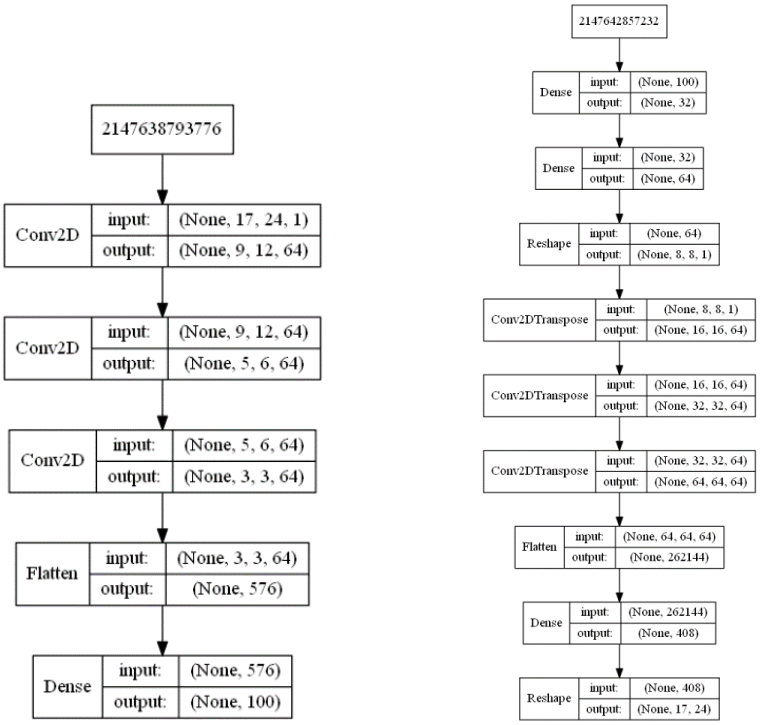


Figure 85: VAE Encoder (left) and Decoder (right)

Here, the encoder returns the μ for each of the 100 latent dimensions, $\sum$, which is calculated as $\mu \times 0.5$; and z, which is the result of multiplying a random normal vector ε with e^{Σ} and adding μ, i.e.

$$z = \left(\varepsilon \times e^{\Sigma}\right) + \mu$$

Equation 43

The input to the decoder is a sampled z vector as defined above.

Below are six examples of simulated tracklet image data, produced by the VAE as explained above.

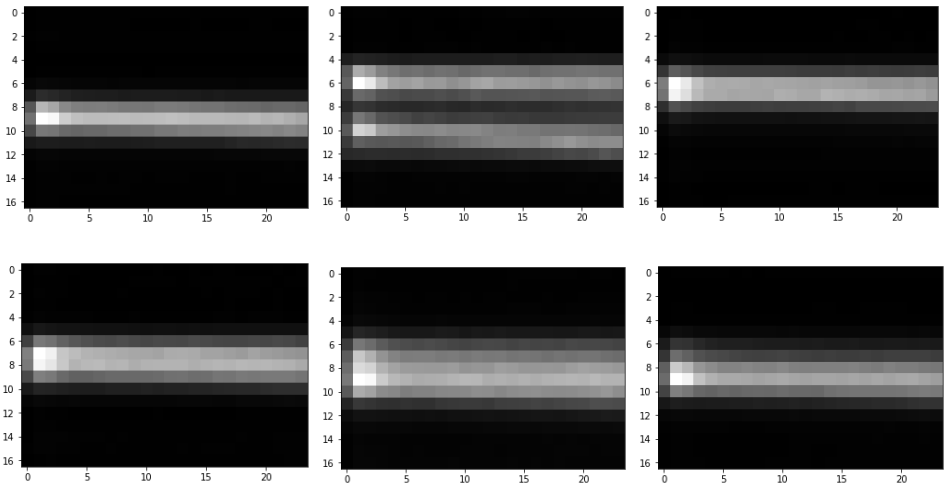


Figure 86: Six examples of simulated data created using a Variational Autoencoder with Convolutional Architecture

Boundary-Seeking Generative Adversarial Network

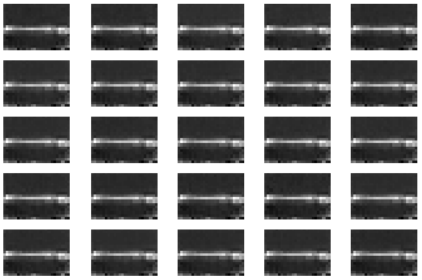
100 Latent dimensions

Adam optimizer with learning rate $= 0.000001$ and a batch size of 32

Generator with 8 hidden layers with 128, 256, 256, 256, 512, 512, 512 and 1024 nodes, using leaky ReLU activation and an output layer using a tanh activation

Convolutional discriminator using two convolutional layers, max-pooling and 5 hidden layers with 1024, 512, 256, 128 and 64 nodes and a single node output layer with sigmoid activation.

Example output after 29000 epochs:



Other Adversarial Autoencoders

AAE Version 1

10 Latent dimensions

Adam optimizer with learning rate = 0.002 and beta1 = 0.5

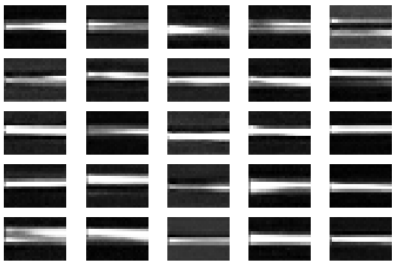
Batch size = 32

Encoder with 2 hidden layers with 512 nodes each, using leaky ReLU activation

Decoder with 2 hidden layers with 512 nodes each, using leaky ReLU activation and an output layer with tanh activation

Discriminator with two hidden layers, with 512 and 256 nodes respectively and sigmoid activation in the single-node output layer

Sample result after 19800 epochs:



AAE Version 2

100 Latent dimensions

Adam optimizer with learning rate = 0.00002 and beta1 = 0.5

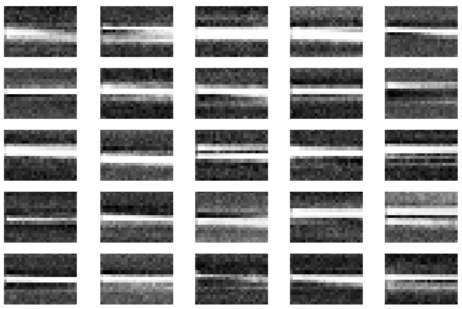
Batch size = 32

Encoder with 3 hidden layers with 512 nodes each, using leaky ReLU activation

Decoder with 3 hidden layers with 512 nodes each, using leaky ReLU activation and an output layer with tanh activation

Discriminator with two hidden layers, with 512 and 256 nodes respectively and sigmoid activation in the single-node output layer

Sample result after 30800 epochs:



AAE Version 3

8 Latent dimensions

Adam optimizer with learning rate = 0.000002 and beta1 = 0.5

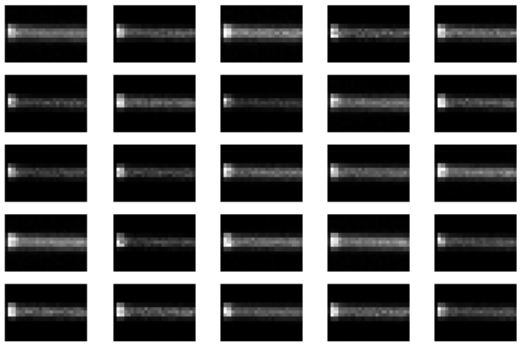
Batch size = 32

Encoder with 7 hidden layers with 1024, 512, 256, 128, 64, 32 and 16 nodes respectively, using leaky ReLU activation

Decoder with 4 hidden layers with 128, 256, 512 and 1024 nodes respectively, using leaky ReLU activation and an output layer with tanh activation

Discriminator with 4 hidden layers, with 1024, 512, 256 and 128 nodes respectively and sigmoid activation in the single-node output layer

Sample result after 23600 epochs:



AAE Version 4

12 Latent dimensions

Adam optimizer with learning rate = 0.000002 and beta1 = 0.5

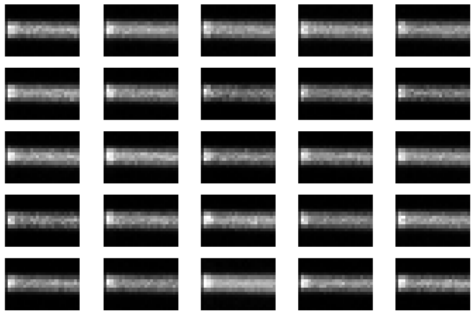
Batch size = 32

Encoder with 5 hidden layers with 1024, 512, 512, 128 and 128 nodes respectively, using leaky ReLU activation

Decoder with 4 hidden layers with 256, 256, 512 and 1024 nodes respectively, using leaky ReLU activation and an output layer with tanh activation

Discriminator with 8 hidden layers, with 512 nodes each and sigmoid activation in the single-node output layer

Sample result after 73200 epochs:



Bidirectional Generative Adversarial Network

100 Latent dimensions

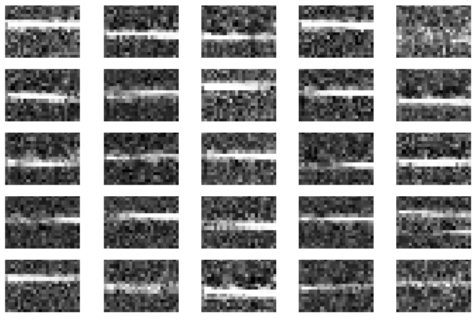
Encoder with 2 hidden layers with 512 units each, using Leaky ReLU activation and employing Batch Normalization with momentum = 0.8 after each

Generator with the same architecture as the encoder, with an additional dense layer of the same dimensions as the number of pixels in an image, with tanh activation

Discriminator with 3 hidden layers with 1024 nodes each using leaky relu activation and a single node output layer with sigmoid activation function

Trained using Adam Optimizer with Learning Rate = 0.0002 and Beta1=0.9 and a batch size of 32

Example output after 39600 epochs:



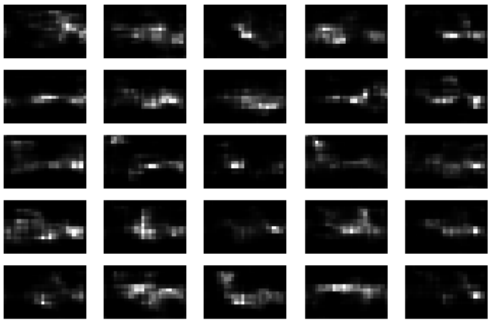
Deep Convolutional Generative Adversarial Network

100 Latent dimensions

Generator with Dense layer with 3072 nodes, reshaped to $4 \times 6 \times 128$ with 2D Upsampling, followed by 3 2D convolutional layers, with the first two followed by Batch normalization and 2D Upsampling, using ReLU activation in the first 4 layers and tanh in the final output layer

Discriminator with 4 2D convolutional layers, with Batch Normalization applied after the second, third and fourth layer and zero-padding after the second layer, using leaky ReLU activation in the hidden layer and sigmoid in the output node.

Example output after 57500 epochs:



Generative Adversarial Network

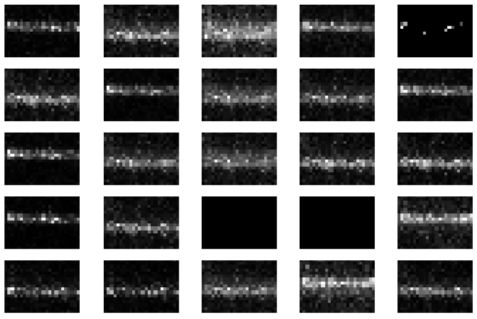
100 Latent dimensions

Two separate Adam Optimizers used for Generator and Discriminator, i.e. with learning rate = 0.00002 and beta1=0.5 for Discriminator and learning rate = 0.00001 and beta1=0.5 for Generator

Generator with 12 hidden layer with 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100,1200 hidden units, using leaky ReLU activation and a dense output layer of the desired output image dimensions with tanh activation, including Batch Normalization with momentum = 0.8 after each hidden layer

Discriminator with 7 hidden layers with 1024, 1024, 512, 512, 256, 256 and 128 units, respectively, using leaky ReLU activation and a single node output layer using sigmoid activation.

Example Output after 66400 epochs:



Note that this is by far not all of the Generative Models built that were deemed unsuccessful without having to numerically assess their results to be sure of that fact. Many more models were built which were even more unsuccessful.

APPENDIX IV: EXAMPLE PYTHON DICTIONARY FOR A SINGLE TRACK

```
1: {'RunNumber': 265309,
        'Event': 197,
        'V0TrackID': 13,
        'track': 14,
        'pdgCode': 211,
        'nSigmaElectron': -3.72201,
        'nSigmaPion': 0.0709847,
        'PT': 3.10114,
        'dEdX': 60.5312,
        'P': 3.10138,
        'Eta': -0.012371,
        'Theta': 1.58317,
        'Phi': 1.05959,
        'det0': 72,
        'row0': 6,
        'col0': 125,
        'layer 0': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [14, 60, 63, 43, 30, 24, 21, 19, 17, 16, 17, 18, 19, 16, 19, 20, 19, 18, 17, 22, 30, 51,
58, 40, ],
                              [14, 66, 81, 57, 37, 30, 27, 26, 22, 26, 24, 23, 26, 25, 35, 38, 37, 37, 36, 70, 91,
193, 286, 186, ],
                              [10, 15, 16, 13, 10, 10, 10, 9, 8, 10, 10, 12, 11, 11, 13, 14, 15, 13, 20, 39, 54, 70,
86, 72, ],
                              [10, 10, 11, 9, 11, 9, 8, 6, 8, 8, 8, 11, 10, 9, 9, 11, 8, 11, 12, 12, 13, 17, 16, 16,
],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
```

```
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      ],
     'det1': 73,
     'row1': 6,
     'col1': 123,
     'layer 1': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [13, 37, 28, 21, 22, 18, 20, 17, 17, 18, 16, 16, 15, 12, 12, 12, 16, 16, 16, 12, 17, 12,
16, 12, ],
                      [17, 64, 50, 38, 52, 39, 33, 36, 33, 35, 32, 30, 29, 26, 25, 28, 46, 65, 45, 36, 42, 35,
30, 22, ],
                      [13, 15, 10, 11, 15, 13, 10, 10, 14, 15, 10, 10, 14, 14, 11, 11, 18, 22, 20, 17, 21, 20,
15, 18, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      ],
     'det2': 74,
     'row2': 6,
     'col2': 122,
     'layer 2': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
```

```
                              [12, 22, 19, 13, 14, 14, 12, 10, 15, 14, 13, 15, 13, 12, 11, 12, 12, 13, 14, 14, 13, 14,
11, 12, ],
                              [14, 68, 48, 31, 25, 31, 32, 29, 50, 41, 39, 49, 45, 37, 32, 28, 30, 32, 38, 48, 35, 32,
32, 30, ],
                              [11, 16, 13, 10, 11, 14, 12, 11, 18, 15, 16, 19, 19, 17, 13, 14, 17, 16, 21, 26, 23, 20,
20, 20, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              ],
       'det3': 75,
       'row3': 6,
       'col3': 120,
       'layer 3': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [10, 13, 14, 13, 11, 12, 11, 11, 11, 12, 10, 10, 11, 11, 9, 10, 10, 12, 11, 7, 9, 9, 10,
10, ],
                              [13, 13, 58, 47, 32, 29, 24, 24, 30, 40, 33, 26, 28, 33, 25, 25, 22, 23, 25, 24, 23, 20,
19, 20, ],
                              [10, 12, 29, 23, 18, 18, 15, 17, 25, 29, 26, 20, 22, 32, 26, 23, 22, 24, 29, 32, 29, 26,
24, 23, ],
                              [13, 10, 11, 11, 11, 10, 10, 11, 9, 10, 10, 9, 11, 9, 10, 11, 11, 8, 12, 8, 8, 10, 10,
8, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
```

```
                        ],
        'det4': 76,
        'row4': 6,
        'col4': 118,
        'layer 4': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [12, 12, 11, 10, 12, 10, 13, 11, 9, 12, 11, 13, 9, 11, 11, 11, 11, 8, 12, 9, 11, 7, 9,
7, ],
                        [10, 38, 46, 31, 29, 23, 40, 29, 25, 20, 18, 42, 52, 38, 46, 45, 28, 25, 24, 24, 20, 17,
18, 19, ],
                        [10, 30, 40, 31, 28, 25, 46, 36, 26, 21, 21, 72, 115, 72, 107, 117, 69, 51, 50, 58, 51,
39, 36, 36, ],
                        [11, 12, 11, 13, 14, 10, 11, 12, 11, 9, 10, 15, 17, 14, 21, 16, 14, 12, 15, 14, 13, 11,
11, 11, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        ],
        'det5': 77,
        'row5': 6,
        'col5': 117,
        'layer 5': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ],
                        [12, 28, 22, 16, 12, 13, 12, 12, 13, 13, 12, 16, 13, 17, 14, 13, 15, 11, 13, 13, 14, 13,
13, 12, ],
```

[17, 60, 51, 33, 23, 23, 24, 28, 28, 26, 23, 38, 44, 67, 52, 35, 51, 53, 37, 42, 39, 34, 31, 29,],
[10, 15, 11, 10, 11, 11, 11, 10, 11, 12, 13, 13, 15, 22, 15, 16, 17, 20, 13, 19, 18, 16, 18, 18,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,],
],},

APPENDIX V: DATA EXTRACTION FROM THE WORLDWIDE LHC COMPUTING GRID (WLCG)

Here, the AliROOT files used to extract the TRD digits data used in this thesis is discussed in detail, as well as the workflow used to run the files and extract the files from the Worldwide LHC computing grid.

Header File (trdML-gerhard/AliTRDdigitsExtract.h)

This header file defines the `AliTRDdigitsExtract` class, which inherits from `AliTRDdigitsTask`.

The following public members are defined:

Methods: `SetV0KineCuts()`

This allows for the exclusion of V_0 candidates based on kinematic parameters, such as momentum.

Variables: `DigitsDictionary`

This is a data structure containing track- and other information.

As well as the following protected members:

Methods: `FillV0PIDlist(); AnalyseEvent()`

The names of these methods are relatively self-explanatory.

Variables: `AliPIDResponse* fPIDResponse; Int_t fEventNoInFile; Int_t universalTracki`

The first variable is an object holding object arrays of electron and pion tracks arising from V_0-decays.

The following two variables are iterators which are incremented during looping.

The Implementation File (trdML-gerhard/AliTRDdigitsExtract.cxx)

An output file stream redirects the C++ `stdout` (which would normally just be printed to the console), to a file: `"pythonDict.txt"`

There are histograms created by this implementation file, which we have ignored for the purposes of this project.

Input events are looped through: for non-empty events, the digits for each V_0 track are read; then V_0 if either track originating from the V_0 vertex is an electron ($e^{+/-}$, PDG code 11) or a pion ($\pi^{+/-}$, PDG code 211), it was added to the `DigitsDictionary` and to either the `fV0pions` or `fV0electrons` object arrays held by `fPIDResponse`.

Tracks in the `DigitsDictionary` were skipped if they did not pass the following kinematic cut: $p \geq 1.5\text{GeV}/c$ or if they did not contain any tracklet information.

The Analysis Macro (trdML-gerhard/ana.C)

The analysis macro serves to compile the Analysis Tasks defined in the previously discussed files. It loads and interprets the required libraries, header files and environmental variables from `$ROOT`, `$AliROOT` and `$AliPhysics`, as well as the macro to be executed.

It defines the version of AliPhysics to use and, when set to run on the grid instead of locally, it sets the appropriate directory that contains the `ESDs.root` files that should be analysed.

It is recommended to run `ana.C` for a single run number at a time, since there are user quotas that will result in your jobs being assigned a lower priority if you use too much virtual memory, etc.

`ana.C` also defines the output directory on the AliEn grid where the results of your analysis will be stored and then starts the analysis on the distributed architecture of the Worldwide LHC computing grid (WLCG).

Running Jobs on the ALICE Grid

Finally, once the AliPhysics environment is entered into and a token is initialised to grant access to the AliEn grid, the analysis macro **`ana.C`** can be run to submit jobs, terminate jobs and merge sub-jobs, at various stages, as explained by [74].

Jobs were submitted using ana.C onto the WLCG and monitored using the ALICE grid Monitoring site, Monalisa [95]. This monitoring website is useful, since sub-jobs will quite often fail, and can then be resubmitted while the job has not timed out. Upon completion of each job, the data produced was extracted back onto Hep01 using the aliensh environment, using a variant of the bash command **`scp`** called **`alien_cp`**.

The data produced during this process was backed up in a semi-private GitLab repository [96], internally accessible by CERN members.

The data extraction process that was employed for this project, along with the attendant Git repositories, is summarised in Figure 87.

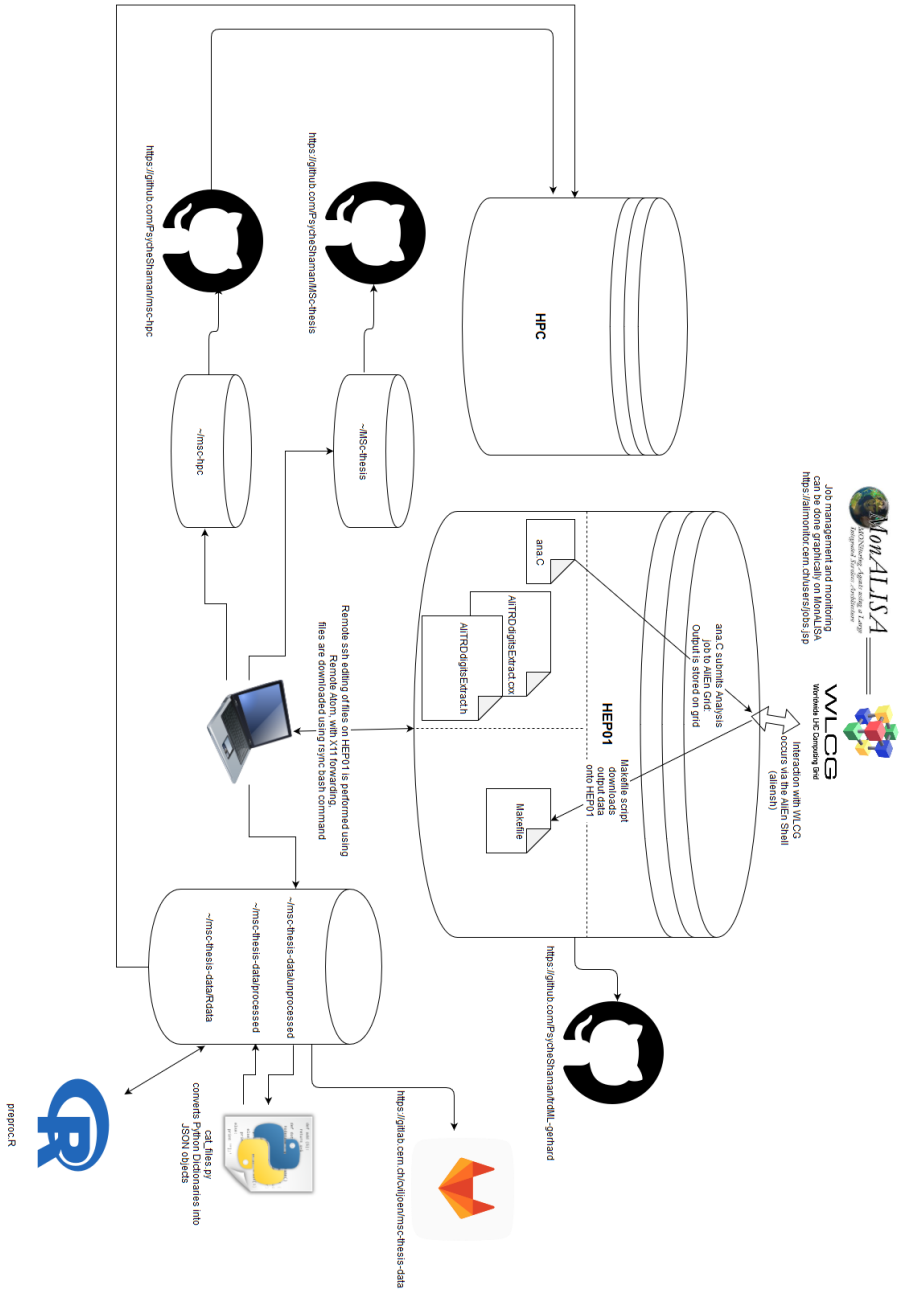


Figure 87: Data extraction and data pre-processing environment (the specifics have changed somewhat during the progress of this thesis)

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 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. I think I potentially learned a lot more in the final stretch of this project from your comments than I did during most of the rest of the project. Your comments helped to tie all the different concepts in this thesis together into a cohesive whole and I am truly excited about a potential PhD in the not-too-distant future. You have certainly prepared me for it!

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. I want to especially mention my grandmothers, Rynet Bloem and Maatjè Catherina Mare Dinkelmann, whom we are lucky to still have in our lives.

***

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

The AliROOT/AliPhysics installation on the Hep01 server at UCT, maintained by Dr Thomas Dietel, was used to run AliROOT and Geant4 implementations.

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

***