# *THE APPLICATION OF DEEP LEARNING TECHNIQUES TOWARDS PARTICLE IDENTIFICATION AND HIGH ENERGY PHYSICS EVENT SIMULATIONS*

*(RELATING TO THE ALICE TRD AT CERN)*


**Christiaan Gerhardus Viljoen**

Department of Statistics || Department of Physics

Faculty of Science

University of Cape Town

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

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

“

A man may imagine things that are false, but he can only understand things that are true, for if the things be false, the apprehension of them is not understanding.

”

—Sir Isaac Newton

# DECLARATION

This dissertation is the result of my own work and includes nothing, which is the outcome of work done in collaboration except where specifically indicated in the text. It has not been previously submitted, in part or whole, to any university of institution for any degree, diploma, or other qualification.

In accordance with the Department of Statistics guidelines, this thesis is does not exceed 20,000 words.

Signed:_____

Date:_____

Christiaan Gerhardus Viljoen, B.Sc., B.Sc. (Med.) (Hons.),
Cape Town

# ABSTRACT

This Masters Dissertation outlines the application of deep learning methods on raw data from the Transition Radiation Detector at CERN as well as simulated data from the Monte Carlo Event Generator Geant4, in order to achieve the following goals:

i. Classification Part I: Particle identification between electrons and pions

To this end, various feedforward neural networks, convolutional neural networks, as well as recurrent neural networks were built using Keras with a TensorFlow back-end, resulting in an ultimate pion efficiency of 2.2% at 90% electron efficiency. Raw data was extracted from the Worldwide LHC Computing grid using the ROOT data analysis framework, a C++ based platform maintained by physicists at CERN. R and Python were used interchangeably during various stages of data exploration, processing, analysis and model-building.

ii. Classification Part II: Distinguishing real data from data generated by Geant4

This stage of the project focused mainly on employing convolutional neural networks towards distinguishing real data from simulated data. Data was simulated using Geant4, a Monte Carlo toolkit which simulates the passage of particles through matter (here matter means highly accurate digital version of the ALICE detector geometry, including the TRD). Once simulated particles have been propagated through the detector, ROOT was used to reconstruct the simulated data to deliver it in a similar format to that given by raw data after processing.

iii. Deep Generative Modeling: Prototyping Variational Autoencoders and various Generative Adversarial Networks towards data generation

Various deep generative models were adapted to take as input raw TRD data and produce simulated observations which are likely under the training data distribution.

# ACKNOWLEDGEMENTS

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

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

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

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

---

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

---

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

---

TABLE OF CONTENTS

# LIST OF TABLES

# List of Figures

# List of Abbreviations and Acronyms

| | |
|---|---|
| ALICE | A Large Ion Collider Experiment |
| TRD | Transition Radiation Detector |
| CERN | European Organization for Nuclear Research |
| QGP | Quark Gluon Plasma |
| LHC | Large Hadron Collider |
| WLCG | Worldwide LHC Computing Grid |
| QCD | Quantum Chromodynamics |
| ML | Machine Learning |
| Pb-Pb | Lead-Lead Collisions |
| $\boldsymbol{e^-}$ | Electron |
| $\boldsymbol{\pi}$ | Pion |
| QED | Quantum Electrodynamics |
| p | Proton |
| n | Neutron |
| $v_e$ | Electron Neutrino |

# 1 INTRODUCTION

## 1.1 Background

This Masters Dissertation seeks to apply cutting edge techniques in Machine Learning (ML) towards:

- Particle identification of electrons and pions, from raw signal data produced by these particles as they traverse the Transition Radiation Detector (TRD), using deep feedforward neural networks, convolutional neural networks and recurrent neural networks, specifically Long-Short-Term Memory (LSTM) networks
- Similarly, various deep learning strategies were used to distinguish between real data from data from Monte Carlo simulations implemented in the Geant4 Monte Carlo simulation environment
- Prototyping of variational autoencoders and Generative Adversarial Networks towards the simulation of High Energy Physics (HEP) collision events

The motivation for each of these elements is as follows:

- Accurate particle identification (in particular, electron samples that are as pure as possible) allows physicists at the ALICE experiment to study the properties of the Quark Gluon Plasma. Since this deconfined state of matter rehadronizes quite soon after forming, it cannot be studied directly, but only via its decay products, of which the electron is one. To this end, having an electron sample which is as pure as possible is desirable and being able to accurately reject pions from the electron sample, whilst keeping as many as possible actual electrons in the sample under investigation is a major concern
- Being able to distinguish Monte Carlo simulations from real data, could be indicative that Monte Carlo simulations, used for calibration and calculations of detector response functions, etc. are not accurate enough and that they could potentially be tuned via various parameter settings in future studies to increase their accuracy
- Using deep generative models such as VAEs and GANs instead of Monte Carlo simulations could be a desirable future course of action, since these simulations are extremely fast compared to Geant4 simulations, but their use is contingent on whether they provide comparable accuracy to Geant4 simulations, as well as their customizability (e.g. is it possible to specify which particle, and at what momentum you want to simulate?).

A variety of different software packages were utilized during the course of this project, including ROOT for data extraction, Geant4 for event simulation, Python and R for statistical analysis and Keras with a Tensorflow back-end for deep learning implementations.

## 1.2 Summary of Work Done & Major Findings

The lowest pion efficiency at 90% electron efficiency obtained with chamber-gain corrected data was 2.2%. This result was obtained using a convolutional neural network as described in 8.1.

The highest balanced accuracy in distinguishing Geant4 simulated data from true raw data was 91.5%. This was also achieved using a convolutional network, discussed in 8.2.

In terms of Deep Generative Models, Variational Autoencoders gave results that look quite realistic to the human eye, but which could be easily distinguished from real data using a convolutional neural network.

A number of variations on the Generative Adversarial Network concept was tested out, each of which performed in vastly different ways, but none of these were found to generate samples that looked as realistic as those obtained from Variational Autoencoders or which could compete with Geant simulations.

## 1.3 The Structure & Organization of this Dissertation

Chapter 2: High Energy Physics & The CERN Experiment begins with a history of atomic theory from Democritus to the Standard Model of Particle Physics, outlines the fundamental particles and forces and the standard model vertices which explains their interactions, then touches upon two ways in which particles interact with matter relevant to this thesis and a quick overview of the Quark Gluon Plasma. Next, the CERN experiment is discussed, in terms of its establishment, particle acceleration hardware and the various experiments conducted at the LHC today, as well as a discussion of its currently used software for data analysis, ROOT and simulation, Geant4.

Chapter 3: The ALICE Detector & the Transition Radiation Detector goes into detail about the ALICE detector, focussing on the Transition Radiation Detector (TRD) and methods currently used for particle identification in the TRD, with a brief overview of their performance.

Chapter 4: Deep Learning introduces Deep Learning within the larger context of Machine Learning and Artificial Intelligence in general, then discusses the original theory upon which modern deep learning is based: Rosenblatt's perceptron. After this, the mathematical background for deep learning used in this dissertation is discussed, including feedforward neural networks, backpropagation, methods for regularization, convolutional- and recurrent neural networks and two types of generative models, namely variational autoencoders and generative adversarial networks.

Chapter 5: Statistical Tests is a short chapter explaining the statistical tests used for particle selection in this thesis at the hand of hypotheses, significance level and power. It also introduces the concepts of electron- and pion efficiency, important in the Results section of this dissertation.

Chapter 0:

Data outlines the format of the data used in this thesis, along with some example plots of TRD tracklet signals, electron and pion counts per run, Bethe Bloch curves for electrons and pions per run as well as nσ-electron plots for electrons and pions per run.

Chapter 7 is the Methods section of this dissertation.

Chapter 8 is the Results section of this dissertation, where only the most successful results are discussed.

Chapter 9 contains the Discussion and Conclusions, including a deeper analysis of results given by various architectures is covered in the Discussion section and a discussion of future work which can be done in this area.

# 2 HIGH ENERGY PHYSICS & THE CERN EXPERIMENT

## 2.1 A Brief History of Atomic Theory

The earliest correct model for the atom can be traced back to 400 BCE, when Democritus proposed that the entire universe consisted of fundamental particles, or "Atoms", which cannot be divided any further.

In 1803, Dalton refined this model to state that these indivisible atoms can have distinguishing chemical and physical traits and that they combine to form chemical compounds.

Then, in 1987, JJ Thompson discovered the electron and proposed an incorrect theory for subatomic structure in which negatively charged electrons were embedded within positive charges within the atom.

Rutherford, Marsder and Geiger disproved this model in 1911, with their seminal alpha-particle scattering experiment and put forth a more accurate model for the atom, in which most of the atom consists of empty space, with a dense core of positively charged protons.

In 1913, Bohr refined this model further, indicating that electrons orbit the positively charged atomic core at distinct energy levels. While this model did explain the emission spectrum of Hydrogen, it could not explain the emission spectra of any of the other elements.

Between 1924 – 1928, De Broglie, Heisenberg and Schrödinger each separately developed a similar quantum paradigm, where electrons have wave-like properties and appear in much more complex orbitals. This is still the accepted theory of atomic structure today.

There have been some refinements made to the quantum theory, as new information has come to light: a neutral subatomic particle, the neutron, was discovered in 1932, which solved the puzzle of why atoms were found to be nearly twice as heavy as expected based on proton number; this discovery also disproved Dalton's second law, which stated that all atoms of a specific element were identical, and resulted in the concept of isotopes (atoms with the same number of protons, but differing numbers of neutrons). In the same year, Cockroft and Walton split the atom for the first time, by bombarding Lithium atoms with electrons, splitting them into two Helium particles.

The 1950s brought about a new era in nuclear physics, in which particle accelerators with collision energies of a few hundreds of MeVs became affordable, along with cosmic ray and inelastic proton-scattering experiments; since this time, a whole host of subatomic elements have been discovered, many of which are unstable. The discovery of these new particles has led, over time, to the development and refinement of the modern Standard Model of Particle Physics.

## 2.2 The Standard Model of Particle Physics

## 2.2.1 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 (1).

At an energy scale of $10^0$ electron Volts (an electron Volt is a unit of energy, equivalent to the amount of work required to accelerate a single electron through a potential difference of 1 Volt), the low energy manifestation of Quantum Electrodynamics (QED) allows atoms to exist in bound states with negatively charged electrons ($e^-$) orbiting a positively charged nucleus consisting of positively charged protons ($p$) and electrically neutral neutrons ($n$), based on the electrostatic attraction of these opposing electrical charges (1).

Quantum mechanics explains the emergence of unique physical properties in different elements, which arise from their exact electronic structures. Quantum Chromodynamics (QCD) is the fundamental theory of the strong interaction, which binds protons and neutrons together within the nucleus of the atom. Similarly, at this energy scale, the weak force causes nuclear β-decays of radioactive isotopes and is involved in the nuclear fusion processes that occur within stars; the nearly massless electron neutrino $v_e$) is produced during both of the abovementioned processes (1).

Therefore, almost all physical phenomena that occur under normal circumstances can be explained by the Electromagnetic-, Strong- and Weak Forces, Gravity (which is very weak, but explain the large-scale structure of the universe), and just four fundamental particles: the electron, proton, neutron and electron neutrino (1).

## 2.2.2 The Fundamental Particles

At higher energy scales, of the order of $10^9$ electron Volt (or $10^0$ giga-electron Volt, 1 GeV), 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) (1).

At the lowest energy level of the standard model, the first generation of particles are then the electron, electron neutrino, the up-quark and the down-quark; these are currently considered to be truly elementary, in that they cannot be subdivided (1).

Higher energy scales, such as those achieved at modern particle accelerators, result in the second and third generation of the four elementary particles; these are heavier versions of the first generation: for example, the muon ($\mu^-$) is essentially a version of an electron which is 200 × heavier than a low energy electron, i.e. $m_\mu \approx 200\, m_e$. The tau-lepton ($\tau^-$) is the third generation of the electron, and is much heavier, i.e. $m_\tau \approx$

3500 $m_e$. These mass differences do have physical consequences, but the fundamental properties and interactions of the various generations remain identical (1).

Current experimental evidence indicates that there are no further generations than these three, and so all matter in the universe seems to be circumscribed by the following twelve fundamental fermions, reproduced from (1):

**Table 1: The twelve fundamental fermions.**

| **Leptons** | | | | **Quarks** | | |
|---|---|---|---|---|---|---|
| | Particle | Q | Mass/GeV | Particle | Q | Mass/GeV |
| First Generation | Electron ($e^-$) | -1 | 0.005 | Down (d) | -1/3 | 0.003 |
| | Neutrino ($v_e$) | 0 | $< 10^{-9}$ | Up (u) | +2/3 | 0.005 |
| Second Generation | Muon ($\mu^-$) | -1 | 0.106 | Strange (s) | -1/3 | 0.1 |
| | Neutrino ($v_\mu$) | 0 | $< 10^{-9}$ | Charm (c) | +2/3 | 1.3 |
| Third Generation | Tau ($\tau^-$) | -1 | 1.78 | Bottom (b) | -1/3 | 4.5 |
| | Neutrino ($v_\tau$) | 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 9 orders of magnitude smaller than the other fermions (1).

The Dirac equation describes the state of each of the twelve fundamental fermions and indicates that for each fermion there is an antiparticle which has the same mass but opposite charge, which is indicated by a horizontal bar over the particle's symbol, or a charge symbol of the opposite sign, e.g. the anti-down quark is indicated by $\bar{d}$, whereas the antimuon is indicated by $\mu^+$ (1).

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 (1).

The strong force confines quarks to confined states within hadrons and quarks are therefore not freely observed under normal circumstances (1).

## 2.2.3 The Fundamental Forces

Classical electromagnetism explained the electrostatic interaction between particles using a scalar potential, Newton stated that matter could interact with any other matter without the mediation of direct contact (1).

Quantum Field Theory circumvents this non-material explanation and encompasses the description of each of the fundamental forces. Electromagnetism is explained by Quantum Electrodynamics (QED), the Strong Force by Quantum Chromodynamics (QCD), the weak force by the Electroweak Theory (EWT), Gravity has not been explained by the Standard Model yet; therefore, Einstein's General Theory of Relativity is still the best explanation of this force, but it falls within the bounds of Classical Physics. As such, the search to incorporate gravity into the Standard Model is an ongoing area of research and has resulted in exciting new theoretical research avenues such as string theory and loop quantum gravity arising (1).

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 circumventing the issue of a non-physical potential as the medium of interaction (1).

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

### 2.2.4 The Higgs Boson

The Higgs Boson, whose existence was confirmed by the CMS and ATLAS collaborations at CERN in 2012, but proposed in 1964 by three separate theoretical papers, breaks rank with the other particles outlined by the standard model in that it is many orders of magnitude heavier and 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, $c$ (1).

The Higgs boson manifests as a disturbance of the Higgs field, which is non-zero in a vacuum, in contrast to the other fundamental particles which all have a vacuum expectation value of zero, i.e. $\langle 0 \rangle = 0$ (1).

In QFT, an expectation value is a real number calculated as the average over the expected values of an observable, weighted according to their respective likelihood.

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 (1).

## 2.3 Standard Model Vertices

The properties of the bosons in the associated quantum field theory for the various forces of the Standard Model (i.e. QCD for the strong force, QED for the electromagnetic force and EWT for the weak force), along with their coupling with the spin-half fermions, are illustrated by three-point interaction vertices of a gauge boson with an incoming and outgoing fermion. Each of these interactions also has an associated coupling strength $g$ (1).

A particle will only couple with the force-carrying boson if it carries the interaction's charge, for instance quarks are the only particles that carry colour charge and are therefore the only particles that can participate in the strong interaction with a gluon; similarly, only charged particles can interact with photons; but since all 12 of the fundamental fermions listed in Table 1 carry the weak isospin charge involved in the weak interaction, they all participate in this interaction (1).

The weak charged-current interaction differs from the other forces in that it is involved in the coupling of different flavour fermions. The $W^+$ and $W^-$ bosons carry charges +e and −e respectively, so in order for electric charge to be conserved, this interaction can only occur between pairs of fermions that differ by one unit of electric charge (1).

Figure 1 shows the main Standard model interaction vertices in the form of Feynman diagrams.

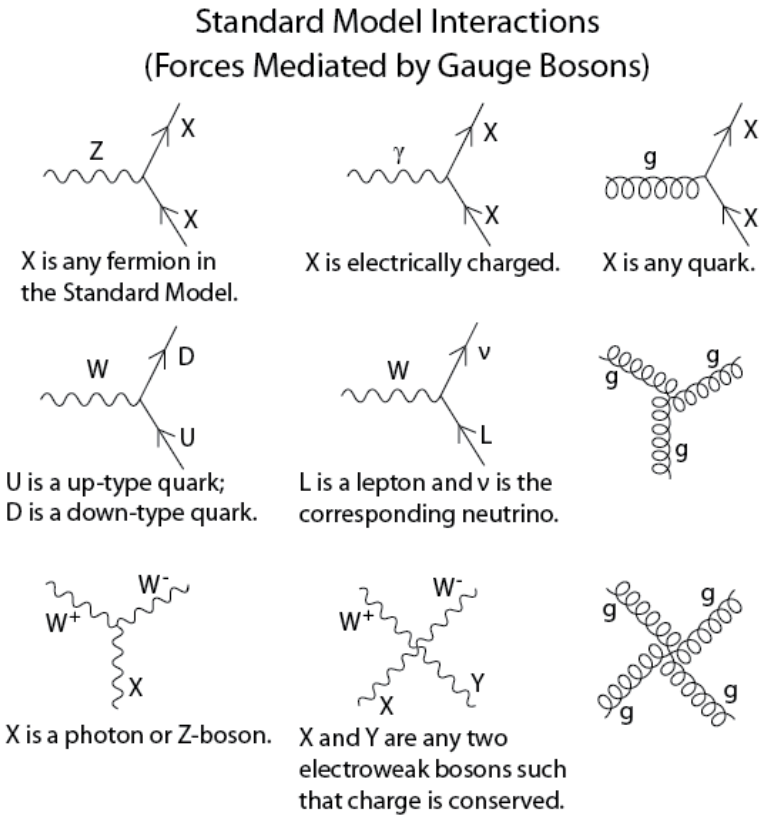



**Figure 1: Standard model interaction vertices (2)**

## 2.4 Interactions of Particles with Matter

In order to study subatomic particle, they need to be detected. Most particles produced during High Energy Physics Experiments are unstable and therefore decay within a specific characteristic mean lifetime $\tau$. Those particles with $\tau > 10^{-10}s$ will traverse several meters before decaying and are therefore directly detectable by particle detectors such as those installed at the Large Hadron Collider (LHC) at CERN. Particles with shorter lifespans are usually detected indirectly, by the interaction of their decay products with detector material (1).

### 2.4.1 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 (1):

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

Figure 2 and Figure 3, illustrate the characteristic energy loss curves for the two subatomic particles studied in this project, the pion $\pi$ and the electron $e^-$.

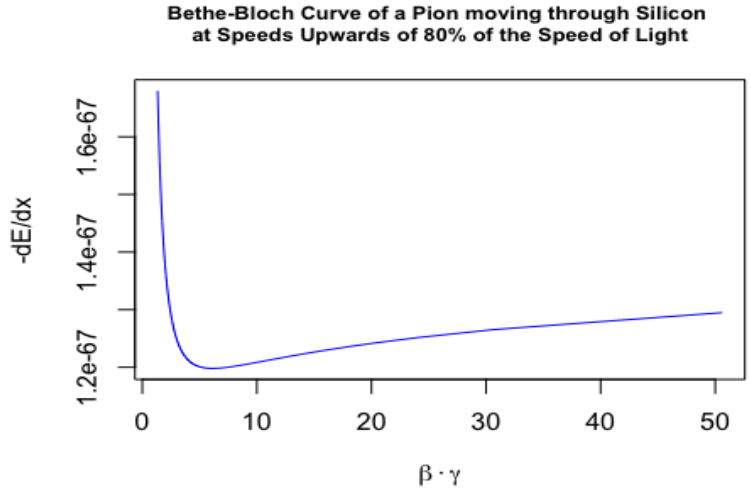



**Figure 2: Bethe-Bloch curve for a pion moving at relativistic speeds through silicon medium**

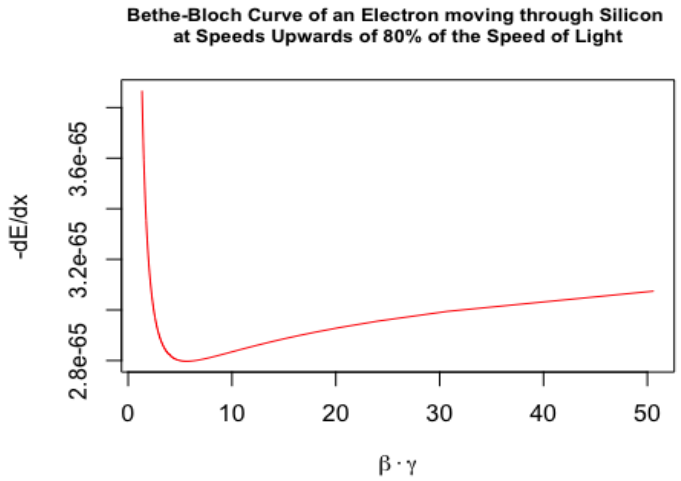



**Figure 3: Bethe-Bloch curve for an electron moving at relativistic speeds through a silicon medium**

### 2.4.2 Transition Radiation

Transition radiation is radiation 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 (3).

For relativistic particles, the photons emitted in this process extends into the X-ray domain and is 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 (4).

One of the main aims of this thesis is distinguishing electrons from pions. This is facilitated by the fact that electrons and pions have different characteristic energy loss curves, and 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.

## 2.5 The Quark Gluon Plasma (QGP)

### 2.5.1 Introduction to QGP

As mentioned above in 2.2.2, quarks and gluons are confined by the Strong Force to remain within the bound states of colour-neutral hadrons (e.g. protons and neutrons) and are therefore never found freely in nature. However, the currently held view of the early universe, predicted by the standard model and supported by over three decades of High Energy Physics experiments and lattice QCD simulations, is that directly subsequent to the Big Bang, the universe was composed of a deconfined state of matter, known as the Quark-Gluon Plasma (QGP) (5).

Statistical mechanics understands matter as a system in thermal equilibrium. Global observables, such as net charge, temperature and energy density define the average properties of such a system. As these global observables take on different values, radically different average properties can be held by the system, manifesting as different states of matter bounded by phase boundaries, which matter traverses via phase transitions (6), see Figure 4 for an illustration of this process.

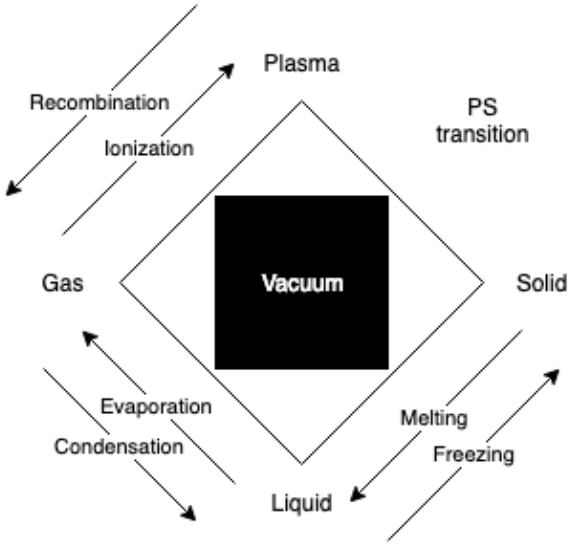



**Figure 4: Simplified diagram of classical states of matter and transitions between them, with the Vacuum added as a fifth element, providing the space in which matter exists (7), reproduced and modified from (6)**

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

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

In Figure 5, a simplified phase diagram of hadronic matter is depicted. Within the hadronic phase, there is a baryonic density/temperature boundary where transitions between mesons (colour-neutral quark-antiquark systems) and nucleons (colour-neutral three-quark systems) occur, (not shown in this diagram). The existence of diquarks as localised bound states within the QGP medium allows for yet another state of matter, the colour superconductor, discussion of which is outside of the scope of this dissertation.

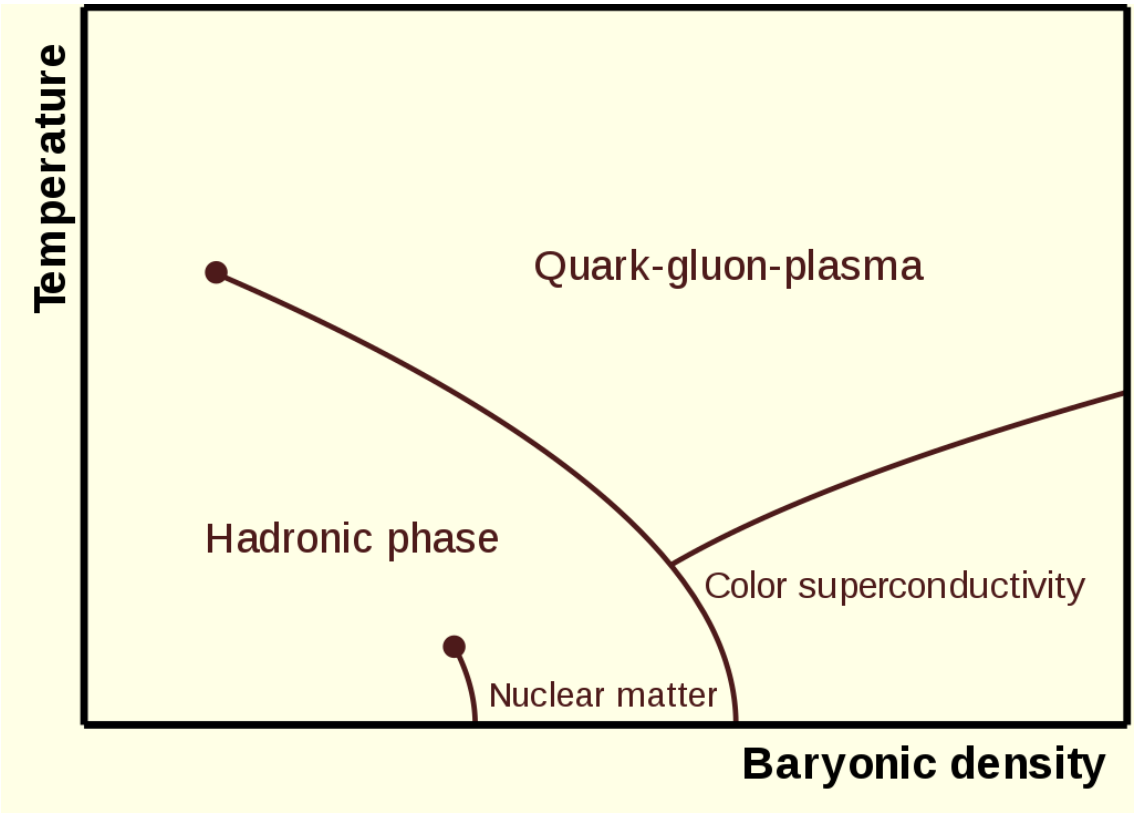



**Figure 5: Phase diagram of hadronic matter (8)**

## 2.5.2 QGP, the Big Bang and the Micro Bang

It is estimated that at t $10^{-43}s$ after the initial expansion of the Universe (affectionately termed the 'big bang', but which is more accurately described as a 'big inflation'), the prevailing temperature was T $\simeq 10^{19}$ GeV, a temperature so high that the principles of general relativity do not apply, and which cannot be understood with present-day physical theory (9).

Quarks and gluons propagated freely in this early deconfined space-time QGP expansion phase of the Universe, down to a temperature of T $\simeq 150$ MeV, a phenomenon thought to be caused by a change in the vacuum properties of this extremely hot early Universe (5).

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

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

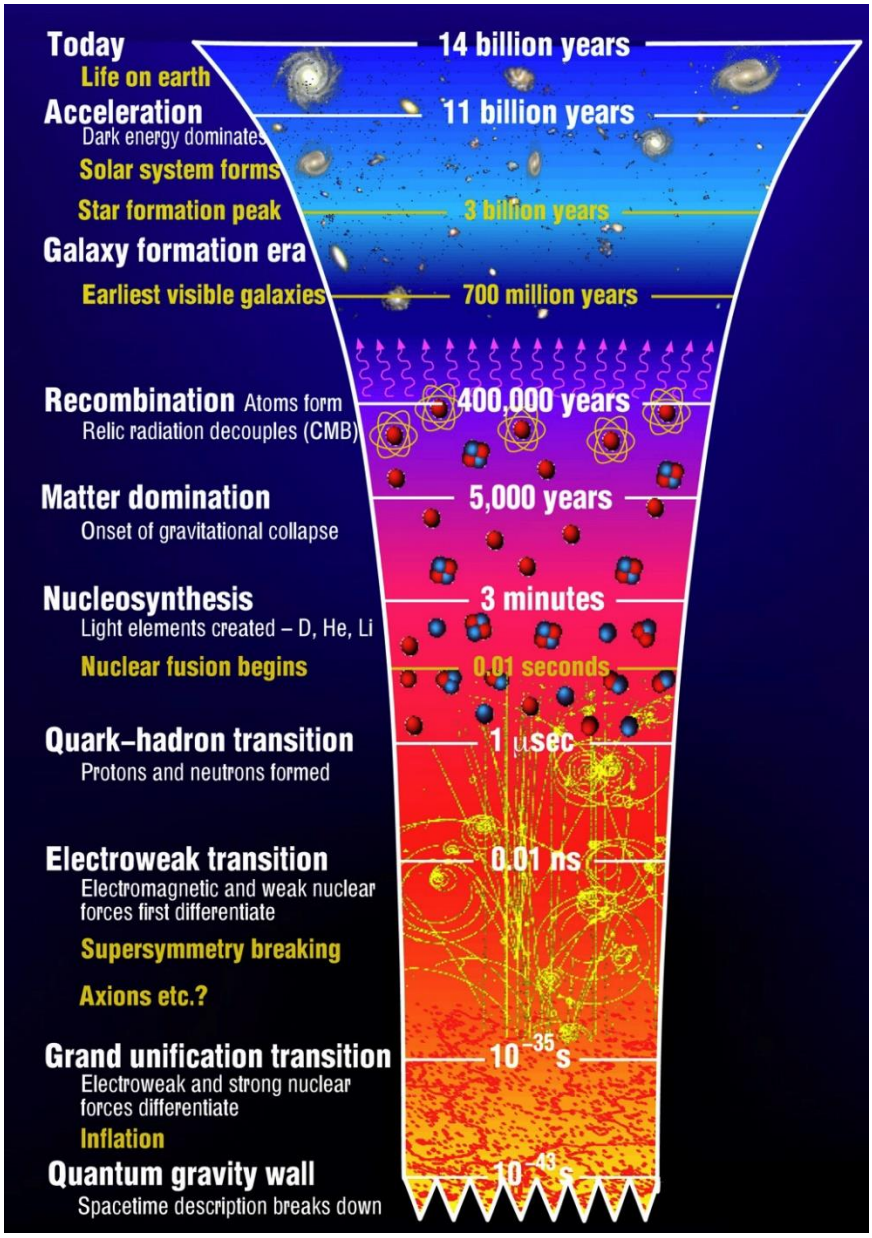



**Figure 6: The evolution of the Universe, from the Big Bang to Modern Day (10)**

## 2.6 The CERN Experiment

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 (11).

Today, CERN is a truly international organization, with 22 member states, who contribute to operating costs and are involved in major decision making, many countries with observer status, and even more non-member countries with co-operation agreements, including South Africa (12).

CERN's research mandate revolves around finding answers to fundamental questions about the structure and evolution of our universe, as well as its origins; it aims to

achieve these goals by providing access to its particle accelerator facilities and compute resources to international researchers, who perform research that advances the forefront of human knowledge, for the benefit of humanity as a whole. As such, CERN is politically neutral and advocates for evidence-based reasoning, knowledge transfer from fundamental research to industry and grass-roots development of future generations of scientists and engineers (13).

### 2.6.1 Hardware

In order to fulfil its ambitious goals, CERN's facilities, located under the Franco-Swiss border (see Figure 7 for geographical context), boasts an intricate system of particle accelerators and -detectors and a data centre with over 174,000 processor cores, 150,000 Terabytes (TB) of Disk space and over 1,000 TB of random access memory (RAM) (14); this main datacentre is connected both to its extension in Budapest, Hungary and the multi-tier Worldwide LHC Computing Grid (WLCG), all of which operates at a data transfer rate of around 10 Gigabytes/second (GiB/s).

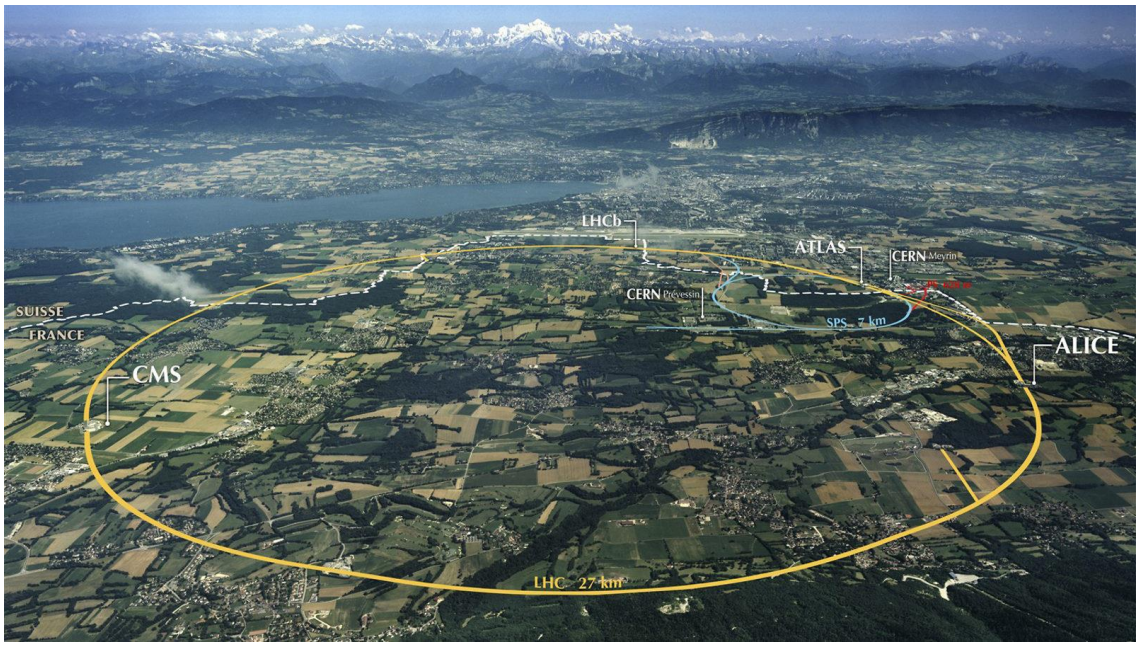



**Figure 7: CERN facilities in geographical context (15).**

High Energy Particle Accelerators

*The LHC Compared to Accelerators from Other Experiments World-wide*

At a circumference of 27 km, the Large Hadron Collider (LHC) is currently the largest particle accelerator in the world (16). To put this into perspective, the Relativistic Heavy Ion Collider (RHIC), located at the Brookhaven National Laboratory in New York, has a circumference of 3.8 km (17), Fermilab's Tevatron, which is no longer in operation, was 6.3 km in circumference (18) and the KEKB accelerator in Tsukuba, Japan also has a circumference of around 3 km (19).

It is also the most powerful particle accelerator in the world, with a centre of mass energy of 13 Tera-electron-Volts ($E_{CM} = 13\,TeV$ (16)), compared to RHIC, which operates at $E_{CM} \approx 200\,GeV$ (17), the Tevatron, which reached $E_{CM} \approx 1.8\,TeV$ (18) and KEKB at $E_{CM} \approx 10.58\,GeV$ (20).

*The LHC*

The LHC, located 50-175 m underground, is the final step in a chain of successive accelerators feeding beams of accelerated particles into each other at increasing energies, as can be seen in Figure 9.

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 it into its constituent protons and electrons (21). A simplified diagram depicting this process can be seen in Figure 8.


**Figure 8: The LHC Proton Source, connected to the Duoplasmatron device, which strips electrons off Hydrogen molecules, to produce the beams of protons which eventually collide within the LHC (21)**

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, which accelerates them up to 25 GeV, the Super Proton Synchrotron is the final intermediate step before proton beams enter the LHC and 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 energy of 6.5 TeV each (22).

To calculate the centre-of-mass energy at collision-time, we do:

$$E_{MC} = \sqrt{s} = \sqrt{(\sum_{i=1}^{2} E_i)^2 - (\sum_{i=1}^{2} \boldsymbol{p}_i)^2} = \sqrt{2^2 - 6.5^2} = 13 \text{ TeV} \quad (1)$$

This equation is derived from the relativistic relationship between energy and momentum, where the rest energy (invariant mass of a particle) is the familiar $E_0 =$

$mc^2$ and the kinetic energy from acceleration is $p_{tot} = p^2 + c^2$. To simplify the equations, the speed of light, $c$ is set at a constant $c = 1$ (23).

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 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 (22); because there are many protons in a single lead ion, the collision energies reached in PbPb collisions reach a maximum of 1150 TeV (22).

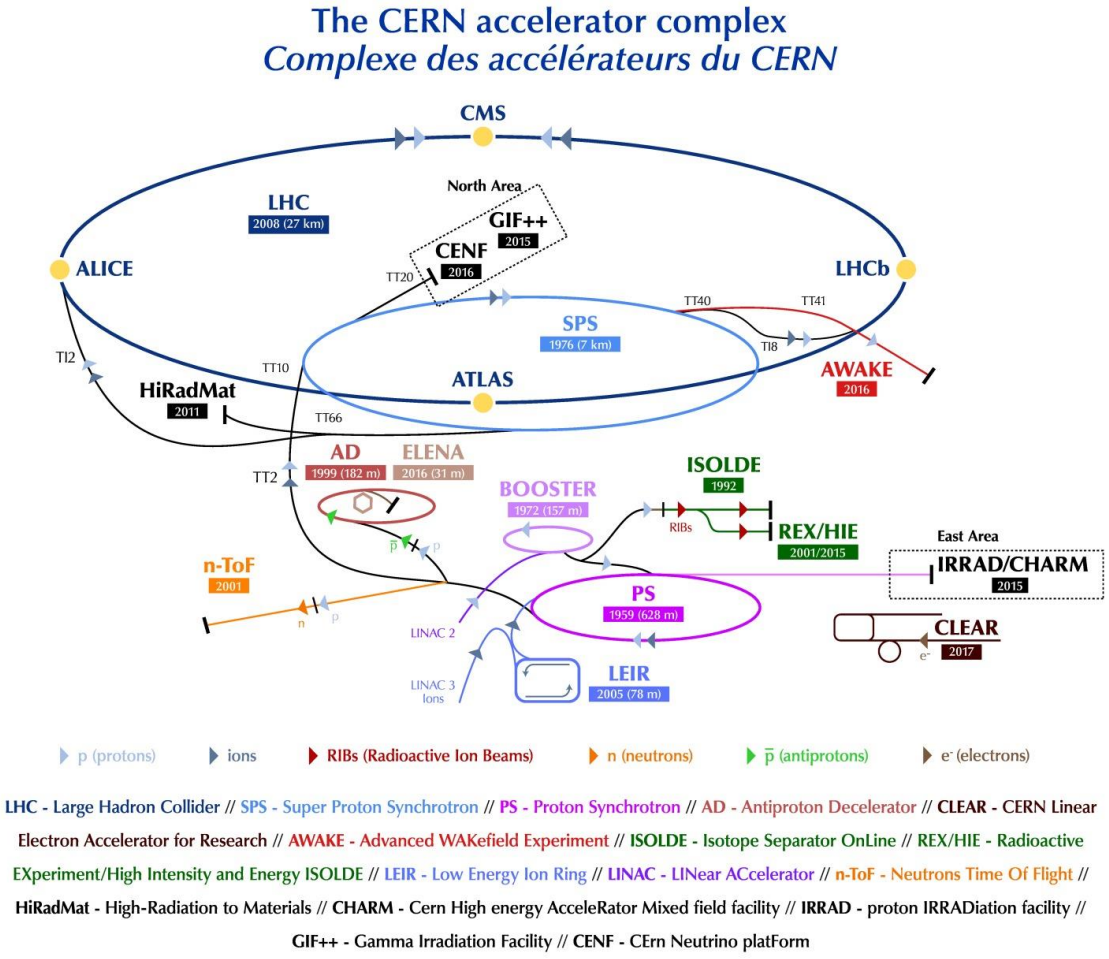



**Figure 9: The CERN accelerator complex (24).**

In order to achieve these high collision energies, a precise system of 1232 dipole magnets is required to keep particles in their circular orbits, with 392 quadrupole magnets employed to focus the two collision beams. The dipole magnets use niobium-titanium (NbTi) cables at a temperature of 1.9 K (-271.3°C). At these temperatures the cables conduct electricity with no resistance (i.e. they become superconducting) and

allow the magnetic field to reach 8.3 Tesla (8.3 T) required to bend the beams around the circular LHC ring (22).

The beams themselves are contained within a vacuum tube 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 around a billion collisions per second (22).

## The Seven 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 which use detectors with differing attributes to study specific areas within the broad area of fundamental subatomic Physics (25).

ATLAS and CMS investigate a very broad range of particle physics. Their independent design specifications allow any new discoveries at one of these detectors, such as the discovery of the Higgs' Boson in 2012, to be corroborated by the other (25). 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 (26).

ALICE and LHCb are the other two main experiments at CERN and are tasked with the discovery of specific physical phenomena (25). ALICE focuses on the extreme energy densities present during heavy ion collisions, which leads to the production of the Quark Gluon Plasma, a newly discovered phase of matter thought to have been dominant in the early universe, directly subsequent to the big bang (27). LHCb investigates subtle distinguishing nuances in the matter-antimatter dichotomy, as evidenced by attributes of the beauty quark (28).

TOTEM and LHCf are smaller experiments focused on forwardly thrown particles produced during non-central collisions, TOTEM investigates particles produced during non-central collisions on either side of the CMS experiment, while LHCf does the same for non-central collisions at the ATLAS experiment (25). LHCf uses some of these forwardly thrown particles produced at the LHC as a simulated source of cosmic rays to complement the calibration and interpretation of large-scale cosmic ray experiments (29).

MoEDAL is the most recent experiment at CERN and searches for a hypothetical magnetic monopole particle; theoretically envisioned, the magnetic monopole would be a subatomic particle with its own magnetic charge, whose evidence of existence would manifest as extensive damage to the MoEDAL detector (30).

## 2.6.2 HEP Software

### 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 (31).

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. The ROOT forums allow users of the platform to report bugs and suggest fixes and in this way contribute to the platform without being part of the official development team (31).

ROOT is freely available for download from (32) and can be installed using precompiled binaries or built from source using the GNU g++ complier on Unix platforms, such as Linux or MacOSX; Windows 10 64-bit users can make use of the Ubuntu subsystem or locally hosted Linux Virtual Machines to install and use ROOT, but native Microsoft Windows is not supported (31).

Upon installation, running the following line in a Unix terminal

```
> echo $ROOTSYS
```

will print the symbolic path to the top of the ROOT directory, e.g.

```
/Users/gerhard/root
```

Looking at the contents of this directory, `$ROOTSYS/bin` contains executables such as the main ROOT executable, daemons for remote ROOT file access and authentication of parallel processing capabilities, etc.

`$ROOTSYS/lib` contains the libraries for the C++ interpreter, image manipulation, ROOT base classes, as well as interfaces with event generators.

Additional directories exist, i.e. `$ROOTSYS/tutorials` which contains example .C macro files, `$ROOTSYS/test` which contains .cxx files and `$ROOTSYS/include` which contains the .h header files.

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 (33), the version of ROOT developed and used for LHC run 1 and run 2; and ROOT 6 (34), the version of ROOT developed for LHC run 3, scheduled to start in 2021 after the second long shut down period (LS2).

### AliROOT

AliROOT and AliPhysics are built on top of the base ROOT architecture to provide functionality specific to the ALICE collaboration.

C++ classes define all the code in ROOT, AliPhysics and AliROOT and enables the user to create variables (data) and functions (methods) specific to each class, as its members. A class's variables are usually accessed via the class's methods (35).

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.

### 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 (36).

Simulation software typically rests on four key components:

1. Event Generation
2. Detector Simulation
3. Reconstruction
4. Analysis

In a typical High energy Physics Simulation set-up, Geant4 is often used as the Detector Simulation component, tied to an event generator such at Pythia or HIJING, with ROOT used for Reconstruction and Analysis. As such, Geant4 has well-defined interfaces to the other components in the simulation set-up (37).

The following aspects of simulation are implemented in Geant4: materials and geometry of the detector system, fundamental particles and their transition through the detector and external electromagnetic fields, how the detector responds to these processes to generate data and storing said data for downstream analysis (37).

# 3 THE ALICE DETECTOR & THE TRANSITION RADIATION DETECTOR

# 3.1 The ALICE Detector System

Colliding heavy ions, such as the Pb-Pb collisions conducted at the LHC and studied at ALICE, offers the most ideal experimental conditions currently achievable for the reproduction of the primordial QGP matter (38). 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 around100,000 times hotter than the core temperature of our sun (38).

The QGP cannot be probed directly, but is studied via 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 (38).

The scale of the ALICE detector system is illustrated in Figure 10. The detector weighs 10,000 tonnes and has spatial dimensions 26m × 16m × 16m (27).

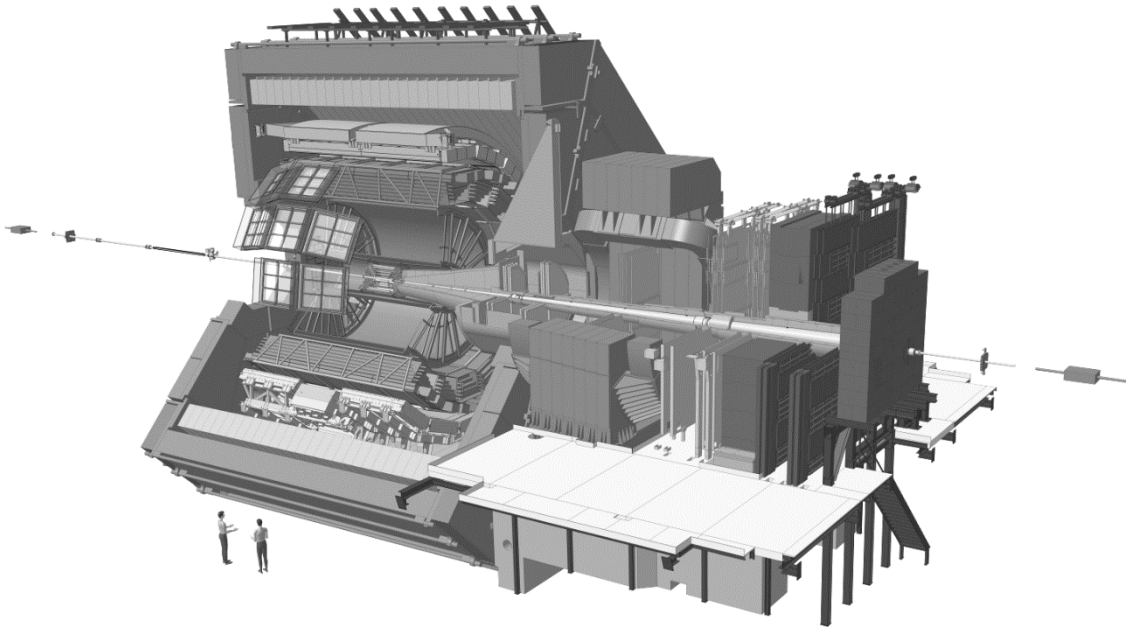


**Figure 10: The ALICE detector system (39)**

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 (38).

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

muons, which interact very weakly with matter and therefore generally travel much further through the detector system (38).

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).

Looking at the detector geometry in more detail, we first find, closest to the collision area, a central barrel part for measuring photons, electrons, hadrons, as well as a forward muon spectrometer, all of which is embedded in a large solenoid magnet and which covers polar angles between 45°-135°. Moving outward from the first layer of the central barrel, we find an inner tracking system (ITS, Figure 11), consisting of 6 planes of silicon pixel detectors (SPD), silicon drift detectors (SDD) and silicon strip detectors (SSD), which provide for high resolution particle detection (40).

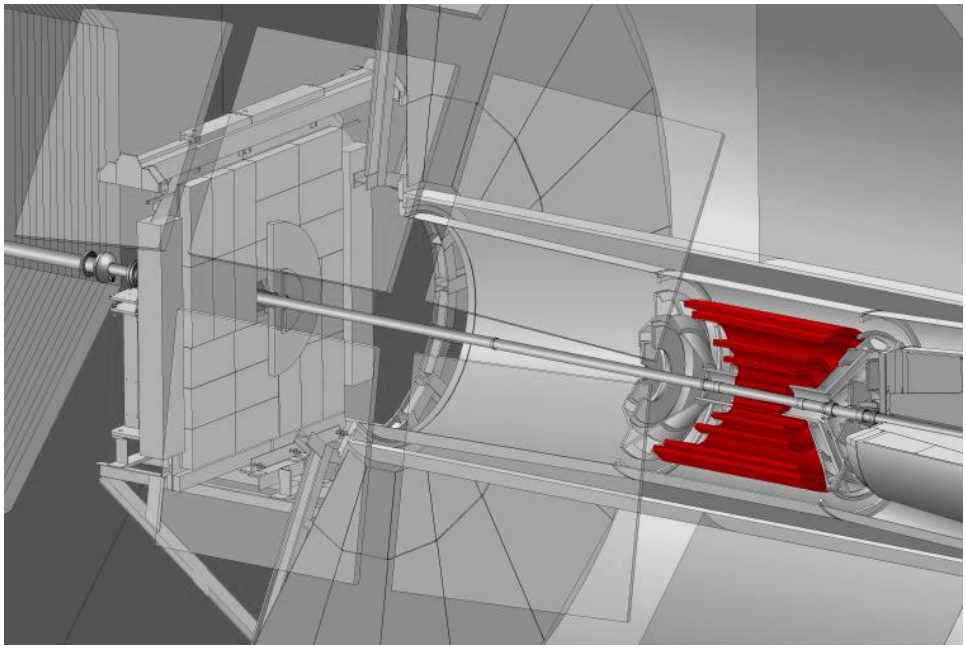


**Figure 11: ALICE Inner Tracking System (SPD, SDD, SSD) (39).**

The main functions of the ITS are: 1) the reconstruction of secondary vertices in the decay of strange- and heavy flavour particles, 2) particle identification and tracking of particles with low momentum, and 3) improving the resolution of impact parameters and momentum. The outer SSD detectors have analog readout for particle identification via d$E$/d$x$ (see section 2.4.1 The Bethe-Bloch Curve), in the non-relativistic (i.e. low $P_T$) region.

Next, as we move outwards, we find the Time-Projection Chamber (TPC, Figure 12).

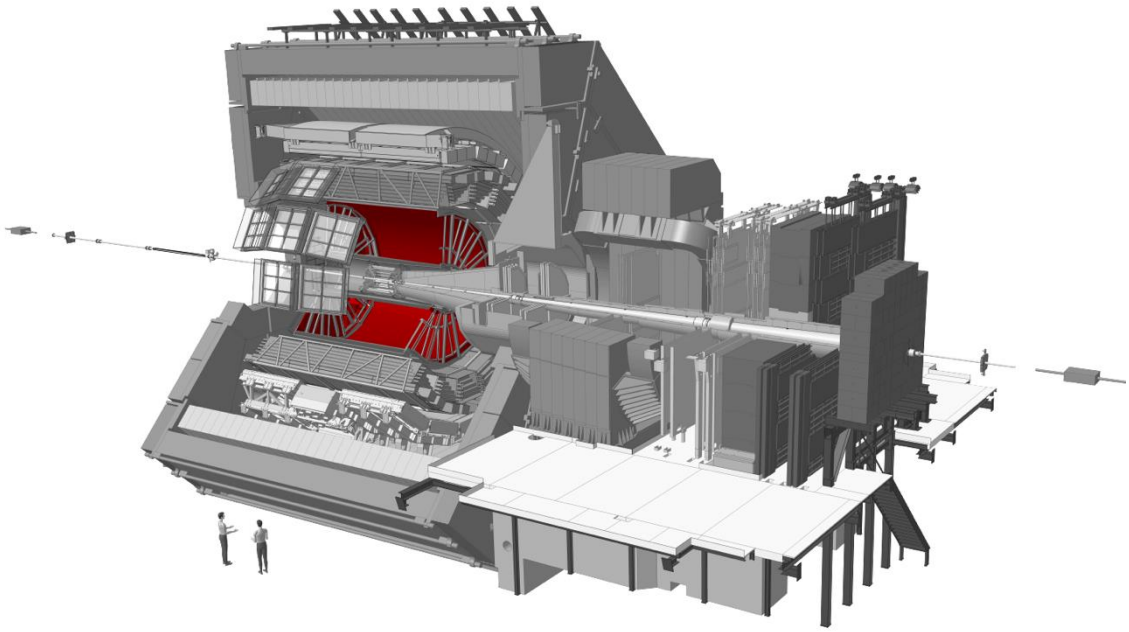


**Figure 12: ALICE TPC (39).**

As the main tracking detector, the TPC is a conservative system, sacrificing data volume and speed for redundant tracking mechanisms, which guarantee reliable performance, by ensuring good double-track resolution and by minimising space charge distortions (40).

After the TPC, we find three Time of Flight (TOF) particle identification arrays (Figure 13).

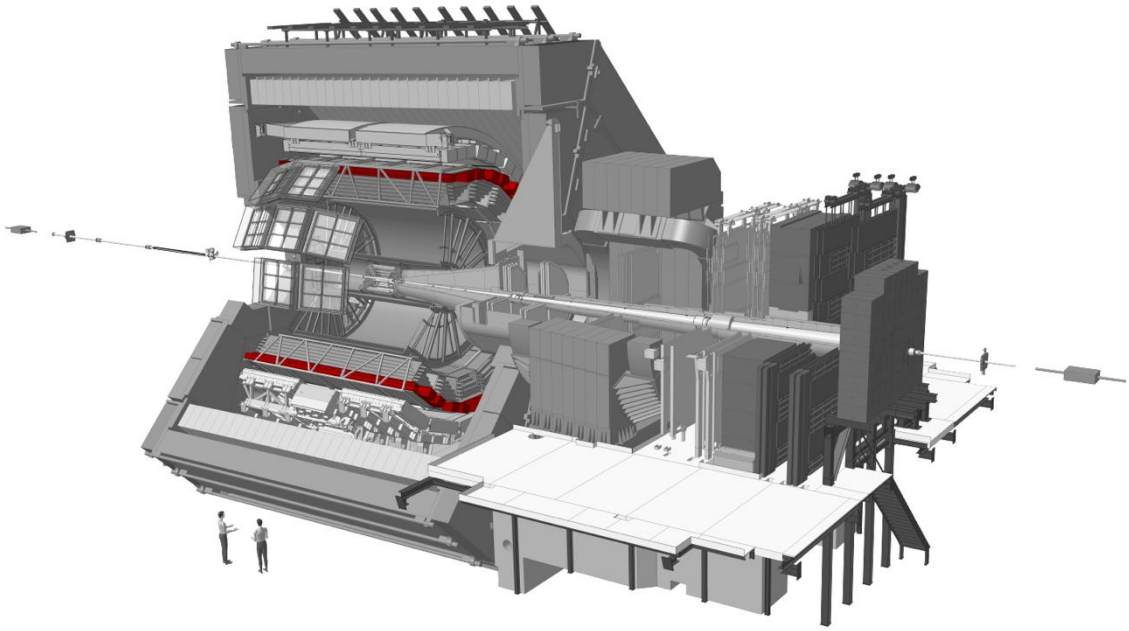


**Figure 13: ALICE TOF (39).**

Optimized for large acceptance and particle identification in the average momentum range, the TOF covers an area of 140 m² with 160 000 individual cells, the TOF offers time resolution of 100 ps.

Next, we find Ring Imaging Cherenkov Detectors (HMPID, Figure 14).

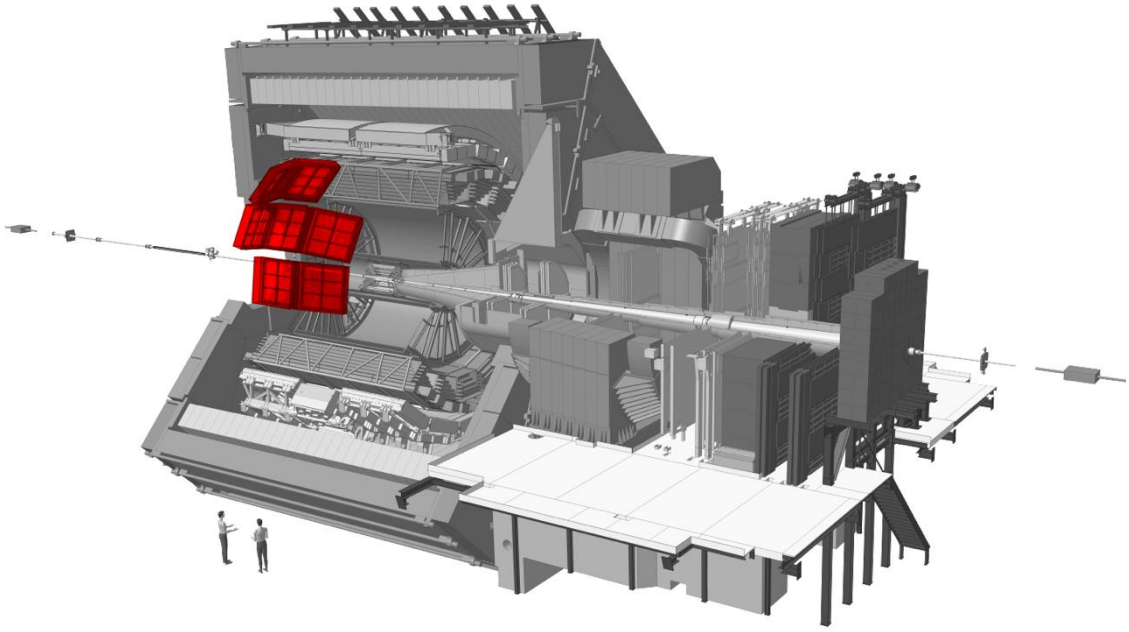


**Figure 14: ALICE HMPID (39).**

A single-arm detector consisting of an array of proximity focusing ring imaging Cherenkov counters, the HMPID extends particle identification (especially the identification of hadrons) towards a higher spectrum of momentum (40).

After the HMPID detectors, we get to the Transition Radiation Detector (TRD, Figure 15), part of the overarching topics of this dissertation.

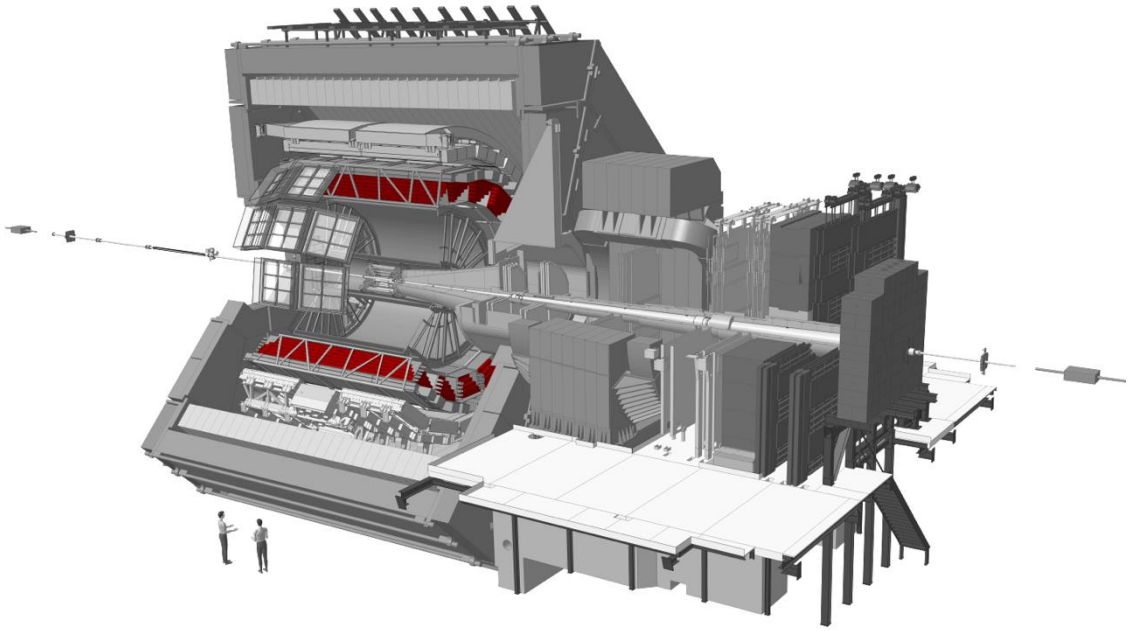


**Figure 15: ALICE TRD (39)**

The TRD identifies electrons of high momentum, above 1 GeV/c, to quantify production rates of quarkonia and heavy quarks in the mid rapidity (relativistic velocity) range (40). Six time expansion wire chambers filled with $Xe - CO_2$ are used in conjunction with attendant composite polystyrene radiators to distinguish electrons from other particles by comparing their actual energy deposition in the detector to their characteristic d*E*/d*x* curves (40).

The outer layers of the central barrel are occupied by two electromagnetic calorimeters (PHOS, Figure 16, and EMCal, Figure 17).

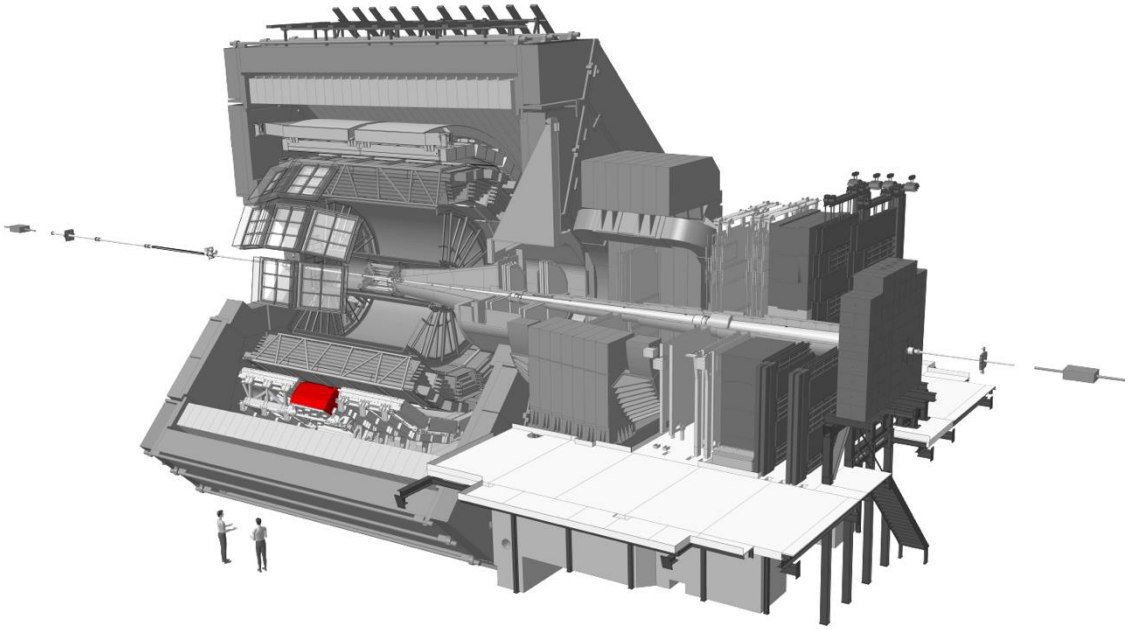


**Figure 16: ALICE PHOS (39).**

Another single-arm detector, PHOS is an electromagnetic calorimeter which gives a high-granularity and -resolution view of photons, to distinguish their production mechanisms (i.e. whether they arise from thermal emission or hard QCD processes). Scintillating $PbWO_4$ crystals amplify the signal to give good resolution of lower energy photons. Charged particles are vetoed by a set of multiwire chambers, inwardly adjacent to PHOS (40).

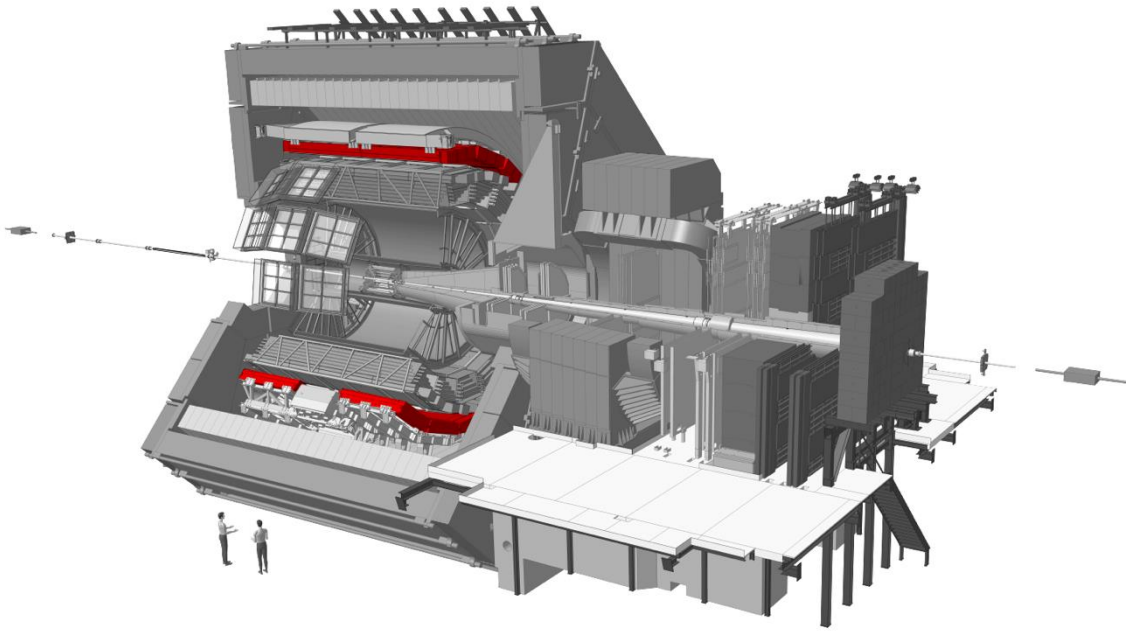


**Figure 17: ALICE EMCal (39).**

EMCal is a lead-scintillator sampling calorimeter, larger than PHOS, it is used in the measurement of jet production rates and fragmentation functions (functions used to calculate the probability that specific observed final states arise from a given quark or gluon) (40).

All of the detectors in the central barrel, except for HMPID, EMcal and PHOS, cover the full azimuth, i.e. they can detect particles at all angles around the central collision area (40).

Outside of the central barrel, a variety of smaller detector elements are found (V0, T0, PMD, FMD, ZDC) that are involved in the triggering of data collection for a specific event, as well as global event characterization (40).

The forward muon arm (covering angles between 2°-9° relative to the collision centre) completes the picture of the ALICE detector (Figure 18). It consists of 14 planes of triggering and tracking chambers, as well as various muon absorbers and its own dipole magnet (40).

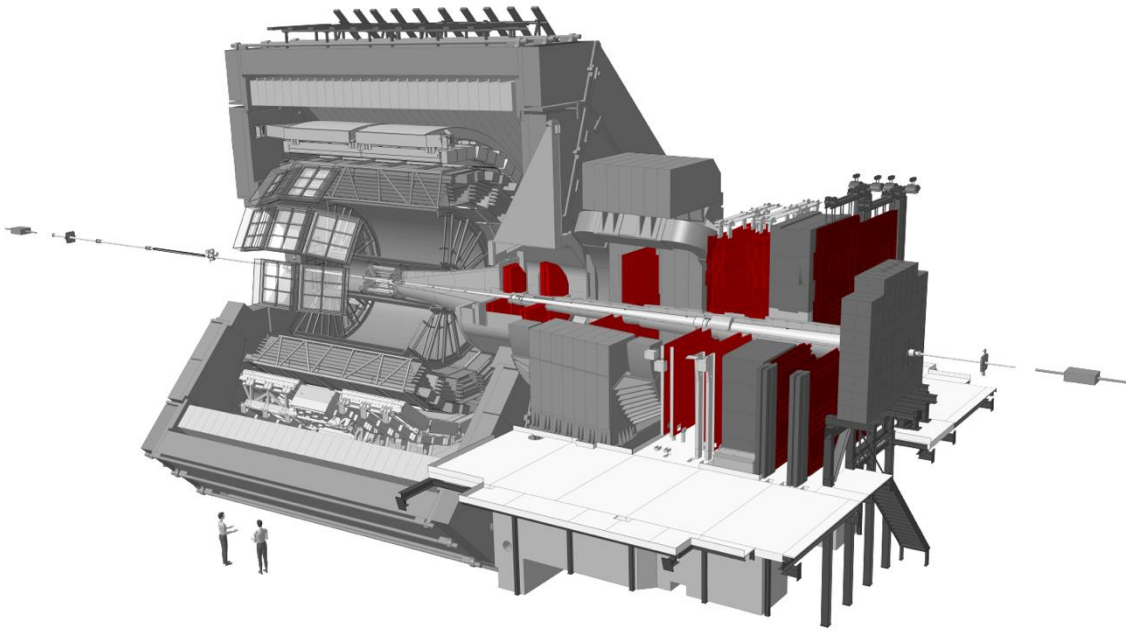


**Figure 18: ALICE forward Muon arm (39).**

The measurement of heavy-quark resonance production is fulfilled by the Muon spectrometer, its small angle relative to the beam-line allows acceptance down to zero transverse momentum. It is made up of a composite absorber and ten thin cathode strip planes acting as high granularity tracking stations. An additional muon filter and four Resistive Plate Chambers are employed in the processes of triggering and muon identification. The muon spectrometer is protected from secondary particles produced in the beam pipe, by a 60 cm-thick absorber tube (40).

## 3.2 The Transition Radiation Detector

At particle momenta above 1 GeV/c, the pion rejection strategy for electron identification employed in the 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 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 (41).

### 3.2.1 A Note on Geometry

Figure 19 serves as a guide to understanding the coordinate system used at the LHC and in this thesis.

The point of beam intersection (the collision centre) acts as the zero-point in geometric coordinate expressions (x = 0, y = 0, z = 0). Cylindrical coordinates are specified from

this origin, with the z-axis pointing along the beam line (with positive z coordinates indicated along this plane in the direction of the muon arm) (40).

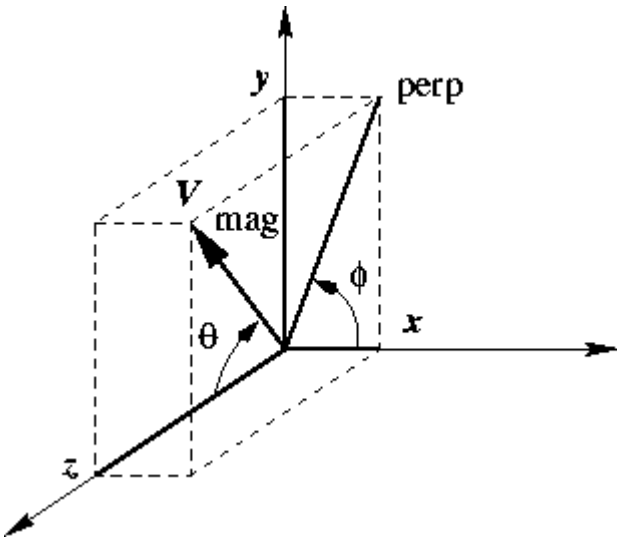



**Figure 19: Cylindrical coordinates as used in geometric coordinate specifications for measurements made in experiments conducted at the LHC (42).**

Where appropriate, traditional Cartesian coordinates are used, for instance when talking about the location of a detector element. In these cases, the y-axis proceeds from the origin in the direction of the wires in the Multi-Wire Proportional Chambers (MWPC, dicussed in section 3.2.3) and also indicates the direction of deflection in the magnetic field, the x axis proceeds from the origin in the direction of electron drift (40).

In order to specify the cylindrical coordinates ($\rho$, $\theta$, $\phi$) of a point P, one can firstly obtain $\rho$, by measuring the distance from the origin to point P. Next, one would project a line from P onto a point Q on the *xy*-plane, to obtain $\theta$, as the angle between the positive x-axis and the line segment from the origin to point Q. Finally, one would calculate $\phi$ as the angle between the positive z-axis and the line segment from the origin to point P (43).

An additional geometric term used in HEP literature is pseudorapidity, $\eta$, which is a specification of a particle's angle relative to the beam (z-) axis.

## 3.2.2 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 TOF and TPC detectors is filled by the six 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. So, in total, we have $18 \times 5 \times 6 = 540$ individual detector elements in the TRD (41) at a radial distance of 2.9 – 3.7 m from the beam axis (44).

Each individual detector element consists of the following broad components: 1) a radiator (4.8 cm thick), 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 (41).

### 3.2.3 TRD Measurement Mechanism

As the name suggests, transition radiation occurs when a particle transits across a dielectric boundary, this radiation is often measured in particle detectors to inform track reconstruction. Multiple boundaries are typically required to increase radiation yield, and since highly relativistic particles emit transition radiation that extends into the X-ray domain, the TRD utilizes gases with high proton-number (Z) to absorb this radiation, resulting in a high yield of energy deposition relative to the energy lost via ionization (41).

The drift time of gas particles within the MWPC provides fine-grained positional information about where the particle tracklet passed through the radiator. The detected signal takes the form of charged gas molecules (ionized via interaction with transition radiation photons and amplified through a chain of interactions between gas molecules), finally being absorbed by a negatively charged wire (anode), this process is shown in Figure 20.

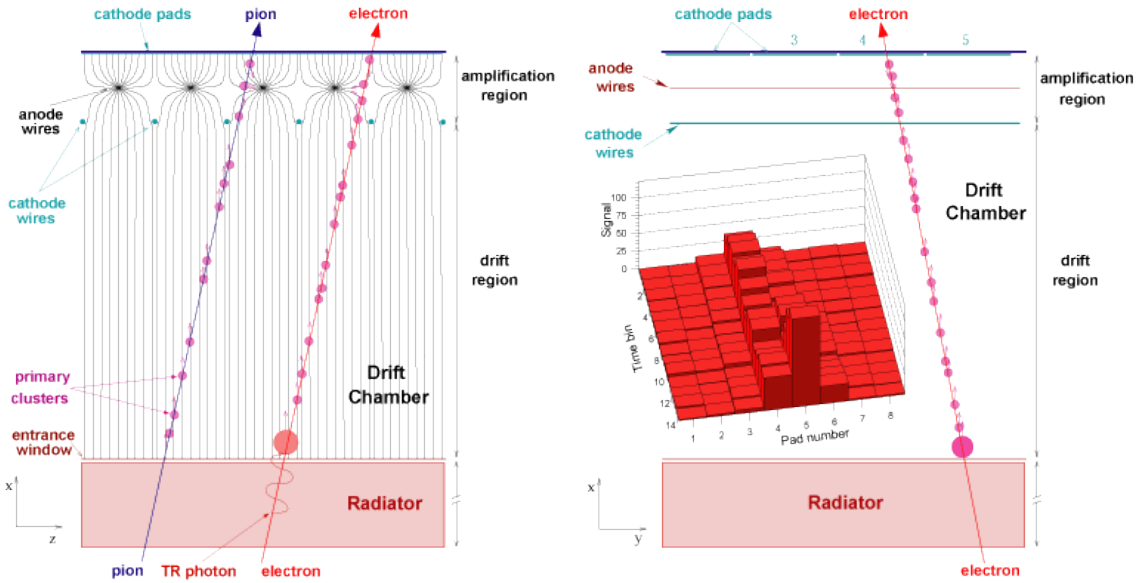



**Figure 20: A schematic representation of the components in an MWPC module**

### 3.2.4 Particle Identification in the TRD

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

vs. electrons) is no longer possible. The time evolution of signals generated in the TRD is an important factor in distinguishing between electrons and pions. The electron identification capability is also used to trigger at level 1 (44).

Electron identification and triggering as mentioned above 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 (44).

The TRD signal originally induced on the segmented cathode plane is captured and processed by a preamplifier-shaper circuit, this processed signal is then digitized by a 10 MHz ADC to take samples of the time-evolution of the signal at defined 100 ns intervals (44).

Figure 21 shows the time evolution of the abovementioned signal at p = 2 GeV, for both electrons and pions. 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 (44).

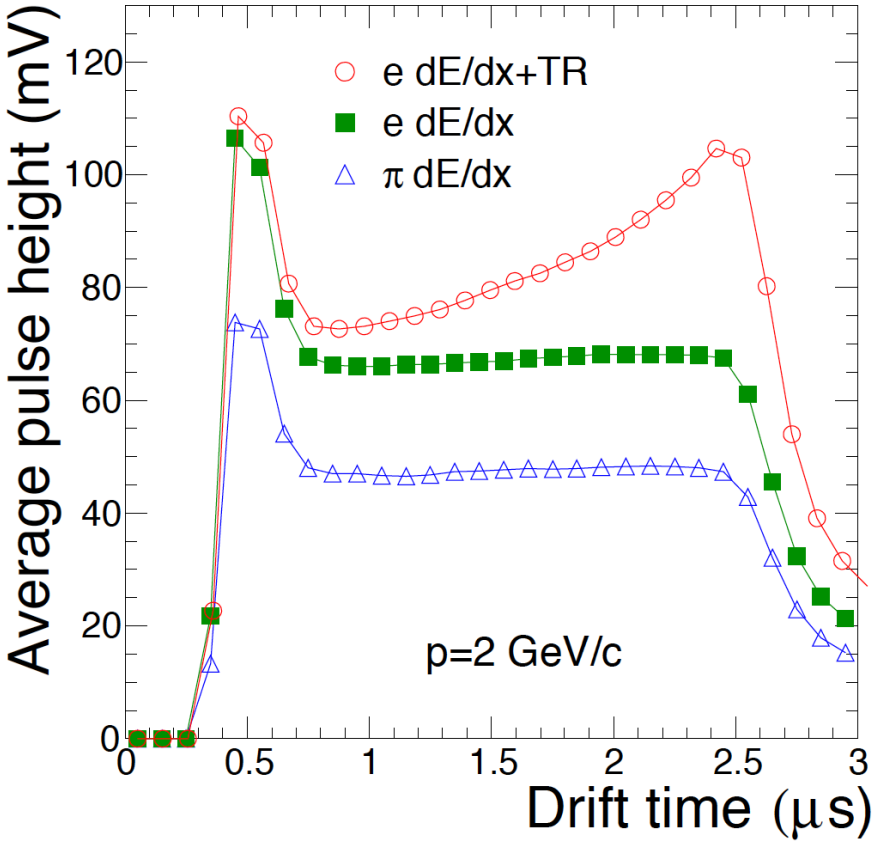



**Figure 21: Time evolution of the TRD signal, measured as pulse height vs drift time for electrons and pions (both at p = 2GeV) (44).**

Also evident from Figure 21 is that, in this momentum region, the pulse height of electrons is much higher than that for pions, because electrons have higher characteristic energy loss (dE/dx) in this region (44).

An average of one transition radiation photon in the X-ray domain will be emitted by an electron traveling at a highly relativistic speed (above $\gamma \sim 800$), since it will cross many dielectric boundaries in the radiator portion of a detector element, the absorption of this type of photon is evidenced by an additional peak at later times in Figure 21, since it will be absorbed preferentially close to the radiator, adding its signal to the ionization energy of the track (44).

In order to distinguish muons originating from particle-particle collisions from muons originating from cosmic rays, cosmic runs were performed at ALICE; this data, along with energy loss and transition radiation measurements from test-beams at the proton synchrotron (CERN PS) in 2004, and proton-proton (pp-) collisions performed at $\sqrt{s} = 7$ TeV at ALICE, provides reference distribution data used for particle identification in the ALICE TRD (see Figure 22 for a plot showing the most probable signal dependence on $\beta\gamma$ (44)).

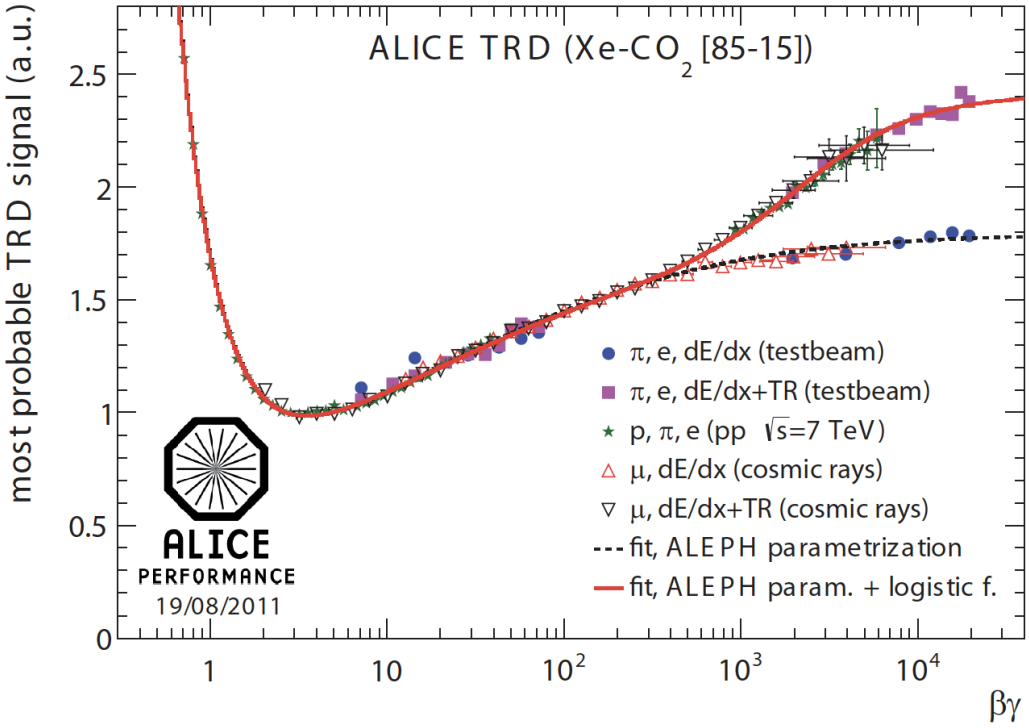



**Figure 22: Reference distributions for most probable signal dependence on βγ in the TRD, i.e. from measurements taken in pp-runs, test beams and measurements from cosmic rays (44).**

Methods used in Particle Identification

Currently, the following methods are employed for particle identification based on TRD data:

1. Truncated mean of the signal
2. One- and two-dimensional likelihood estimations

3. Neural Networks

*Truncated Mean*

The truncated mean signal is the combined signal of Transition Radiation + Specific Ionization Energy , this method focusses on classifying electrons vs pions based on their expected energy loss as per the Bethe Bloch curve shown in Figure 23.

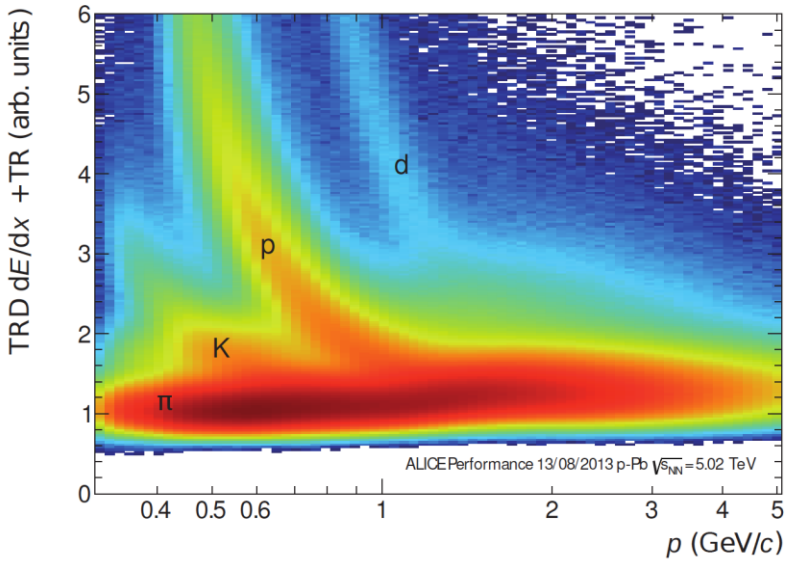



**Figure 23: Truncated mean signal (dE/dx + TR) for various charged particles as measured for p-Pb collisions at $\sqrt{5.02}$ TeV. This method allows for particle identification of light particles and hadrons (44).**

*One-dimensional Likelihood (LQ1D)*

One dimensional likelihood estimation is performed on the total integrated charge left by a particle in a single chamber in the TRD (i.e. a single tracklet). Figure 24 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 (44).

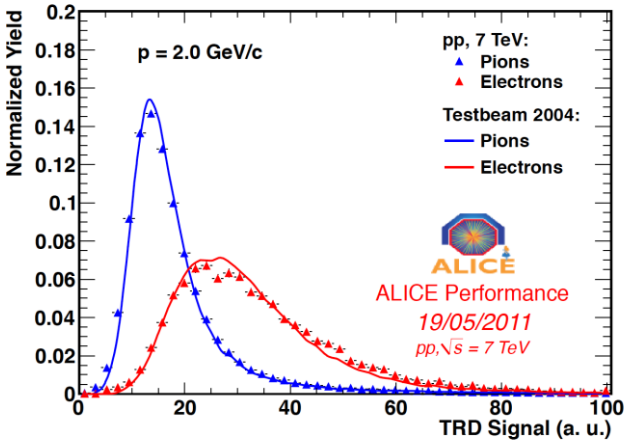



**Figure 24: Normalised distribution of charge deposition for electrons and pions in a single TRD chamber (44).**

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 a clean reference sample of electrons arising from photon conversion (44).

*Two-dimensional Likelihood (LQ2D)*

Two-dimensional likelihood takes the temporal evolution of the signal (Figure 21) into account by splitting the signal into two time-bins and summing the charge in each bin and calculating the likelihood based on pure pion- and electron samples from collision data (44).

*Neural Networks*

A neural network was trained using a similar approach as LQ2D, but instead of splitting and summing over two time-bins, the input feature-set to the neural network was obtained by splitting into seven time-bins and summing the charge over each bin, respectively (44).

Particle Identification Accuracy

To calculate the accuracy of the abovementioned methods, clean reference samples were used. The separating power of these approaches are often expressed as pion efficiency (the fraction of pions incorrectly classified as electrons, i.e. the false positive rate or fallout rate) at a specific electron efficiency (the fraction of electrons correctly identified, i.e. the true positive rate or sensitivity) (44).

Figure 25 shows the obtained pion efficiency for the methods discussed above, as a function of electron efficiency, it is clear from this plot that the misidentification of pions as electrons (False Positive Rate) is reduced substantially by the LQ2D and Neural Network techniques, compared to truncated mean- and LQ1D methods, and that the temporal evolution of the signal is therefore a highly informative feature for particle identification (44).

It is important to note that pion suppression (the inverse of pion efficiency) is hampered when a particle passes through fewer than the available six layers of the TRD, and that electron efficiency is sometimes sacrificed during analysis to obtain a more pure sample (44).

Figure 26 shows how pion efficiency depends on momentum for the four methods under discussion, data is plotted for samples where electron efficiency of 90% is obtained. LQ1D and LQ2D are quite accurate at low momenta where the emission of transition radiation commences, but their separating power decreases at higher momenta as transition radiation production saturates and pions deposit more energy, making it harder to tell them apart. The truncated mean method performs poorly at high momenta, since transition radiation with its attendant high charge deposition is more likely to be removed during the truncation procedure (44).

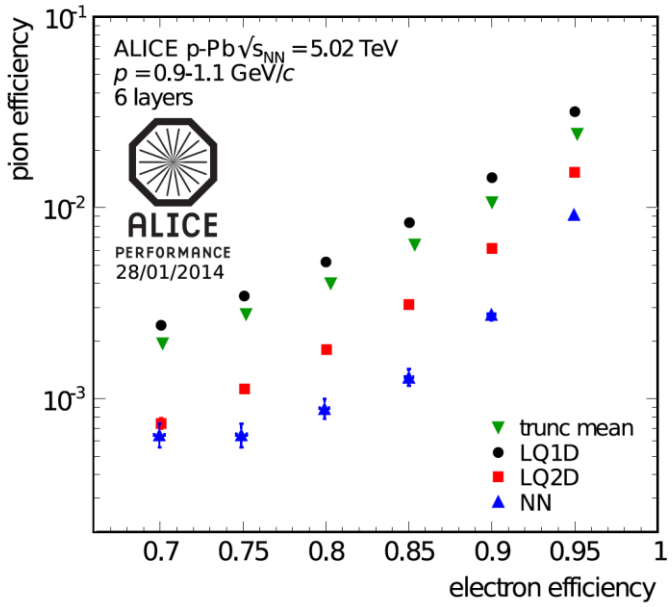



**Figure 25: Pion efficiency as a function of electron efficiency for the various particle identification methods discussed (44).**

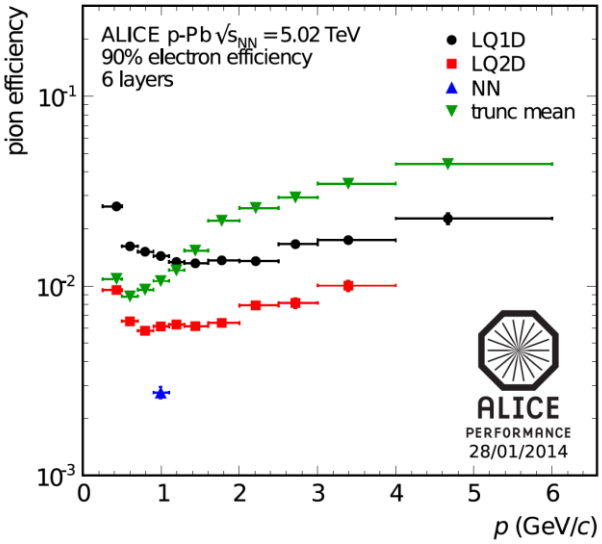



**Figure 26: Momentum dependence of pion efficiency for various methods (where electron efficiency is at 90%)**

# 4 DEEP LEARNING

# 4.1 Deep Learning within the Context of Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a branch of Computer Science concerned with getting computers to perform tasks that are characteristic of those performed by the human mind. The field of AI encompasses both hard-coded rule-based programs (known as the knowledge base approach to AI, which has largely remained ineffective), as well as Machine Learning, which is an approach to AI which aims to get computers to perform these tasks without explicitly coding the solutions for them (45).

The success of Machine Learning algorithms is largely determined by the representation of the data fed through them. 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 (45).

In the case of machine learning for image classification, which loosely ties back to some of the aims in this project, it is not always immediately obvious as to which features will be informative to an ML algorithm. For example, feeding raw pixel values into a linear regression model should not be very effective, since images vary in terms of positional information, lighting, sharpness, rotation, etc. (45)

Representation learning is a solution to feature generation in which ML is applied, not only to map from a feature set to an output, but also towards automatically learning the most useful representation of the data; usually this representation will encompass identifying the major factors of variation which effectively explain the observed data and discarding those which are not useful to the algorithm (45).

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, but once compiled into a Multi-layer Perceptron, the rich texture of the input data distribution can be very accurately captured, since useful features discovered in the first layers of such a neural network can be combined in various ways to create additional useful features (45). Continuing with the image classification example, an early layer of a convolutional neural network may detect edges in an image, the next layer may detect corners and shadows, and layers further down will ideally detect actual visual elements (faces, car lights, arms, etc.) (45).

# 4.2 Mathematical Background for Deep Learning

## 4.2.1 Rosenblatt's Perceptron

The original Rosenblatt paper (46) outlining the concept of the "perceptron" aimed to develop a theory to explain: 1. How sensory information is detected by biological

organisms, 2. how that information is subsequently processed and stored and 3. how mental comprehension or organismal behaviour (which he termed "*preference for a particular response*") was driven by the first two processes.

He outlined a mathematical framework for these mechanisms, at the hand of the following constructs:
1. **S-points:** sensory units which can possess any of a number of response curves based on the signal strength of incoming information
2. **A-units:** association cells located in an "association area" $A_{II}$, which in some of his models was preceded by a "projection area" $A_I$
3. S-points are connected in specific ways to A-units and forward their stimulus response to them, in the form of an inhibitory or an excitatory impulse
4. $\boldsymbol{\theta}$: A threshold value assigned to each A-unit dictates whether it will fire, based on the algebraic sum of excitatory and inhibitory signals received, from either S-points or preceding A-units
5. The connections between S-points and A-units, and between A-units themselves is random, and not all elements of such a network are connected to each other
6. Response units, $R_1, R_2, \dots, R_n$, receive a large number of inputs from the $A_{II}$ set, called its source-set, and have feedback mechanisms to A-units in its source set. (46)

He put forth various models for response curve summation and how these networks would learn (46), but while the mathematical constructs he proposed were oversimplifications of the complexity of biological brains, they were found to be extremely useful in training computers to emulate their capabilities.

### 4.2.2 Deep Feedforward Neural Networks

At its most basic level, an artificial neural network (ANN) is an approximation of a mapping function $f_a$, which maps from a set of input features $x_i\ ;\ i = \{1,2, \dots, n\}$ to a response, $y$. Feedforward neural networks have one-way information flow from input features to output, whereas recurrent neural networks have feedback connections (45).

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

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

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 (46).

The process of training such a network, $f$, to give the closest approximation to the desired output, $y$, is an iterative process, involving passing many observations, each having the same feature set $x_{i,\dots,n}$ through the MLP, assessing the output, $\hat{y}$, according to an error metric, $E$, and individually adjusting each of the mapping functions $f_a^{j,\dots,m}$ according to their contribution to the differential of the magnitude of error at the conclusion of each training step $k$. In other words, a parameter set $\theta$, pertaining to each $f_a^j$ is iteratively adjusted according to $\frac{\partial E_k}{\partial f_a^j}$. (45).

The set of nested approximation functions outlined above are commonly referred to as hidden layers, the dimensionality of the outputs of each layer is known as its width, or as the number of neurons in that particular hidden layer (45).

In order to produce subtle derived features from the input feature set, nonlinear transformations are applied to the output of each layer in the network, which in itself is a simple linear function of the form $w^T x + b$ , where $w^T$ is a vector of weights of the same length as the set of input features, which are essentially a set of coefficients for each $f_a$ in the chain of functions, and $b$ is a real-valued bias term, which is essentially an intercept term for each $f_a$ (45).

It is easy to see that chaining such a set of linear models without applying nonlinear transformations (denoted as $\phi(f_a(x))$) to what are essentially an arbitrary number of linear regression functions ($y = \beta_1 x_1 + \beta_1 x_1 + \beta_1 x_1 + c$), one would simply arrive at another linear model (45).

Non-linear transformations applied over $w^T x + b$ allow deep learning models to more accurately model the multidimensional feature space of the data distribution.

A commonly used nonlinear transformation $\phi$, or activation function, in modern deep learning algorithms is the rectified linear unit (the ReLU function), which is simply an affine transformation, reminiscent of the response curves envisioned in Rosenblatt's paper, of the form $\phi(f_a(x)) = \max\{0, f_a(x)\}$ (45).

Various other nonlinear transformations (more commonly known as activation functions) can be viewed on the Keras website at (47).

Combining the concepts explained above, then gives us a representation for a single hidden layer in an ANN as follows:

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

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

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

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

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

We now have a vector of weights multiplied by a vector of input features, which can be the original features fed to $h_1$, or the weighted outputs of previous hidden units in $h_{2,\ldots,n}$. Since we essentially have a vector of hidden units, we also have a vector of bias terms, and all of these hyperparameters, collectively referred to as $\theta$, need to be optimized to arrive at a reasonable approximation of a theoretically optimal mapping function $f^*(x) = y$ (45).

To achieve the optimization of $\theta$, most deep learning models utilize the concept of maximum likelihood, to minimize a loss function $J(\theta)$, for example, binary cross entropy:

$$J(\theta) = -(y\log(p)) - (1 - \log(p)),$$

where $p$ is the model's estimate for the probability of an observation of being of a particular class $y$ (45).

Please see Appendix B for the code used to generate Figure 27, which shows how, as $p_{model}(y|x)$ approaches the true $y$ (in this binary classification example, $y = 1$), the binary cross entropy loss function approaches 0.

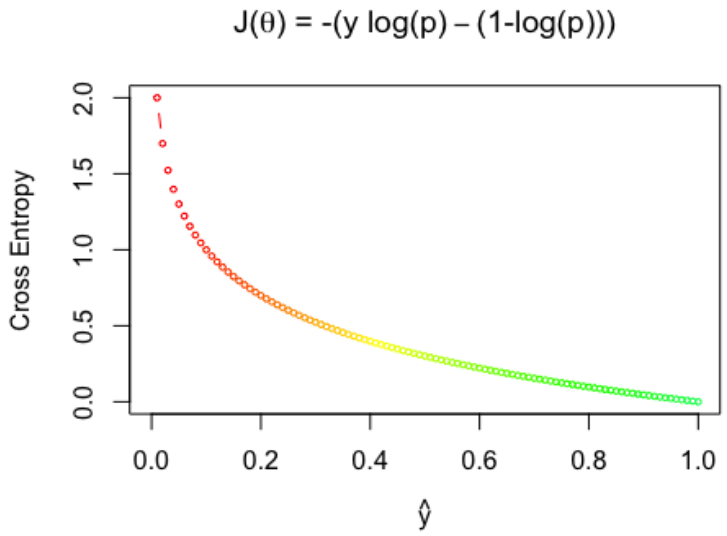



**Figure 27: Illustration of the descent towards zero, of the Binary Cross Entropy Loss Function as ŷ, or $p_{model}(y|x)$, approaches the true y.**

The chain rule of calculus is employed by backpropagation to enable the derivative of the loss function to be redistributed through the network, based on the partial derivative of each hyperparameter with respect to the derivative of the loss function (45):

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

For k = $l,\ l-1, \ldots, 1$ hidden layers, $h^{(l)}, h^{(l-1)}, \ldots, h^{1}$, we compute the element-wise gradient on the layer's output (before the non-linear activation function is applied):

$$g \leftarrow \nabla_{a^{(k)}} J = g \odot f'\left(a^{(k)}\right)$$

And the gradients on the weights and the bias term:

$$\nabla_{W^{(k)}} J = g\, h^{(k-1)T} + \lambda \nabla_{W^{(k)}} \Omega(\theta)$$

$$\nabla_{b^{(k)}} J = g + \lambda \nabla_{b^{(k)}} \Omega(\theta)$$

Here, $\lambda$ represents the weight decay penalty, where the size of the weights are constrained, in a manner inversely proportional to $\lambda$ . A regularizer $\Omega(\theta)$ is added to the loss, where $\theta$ contains all the weight and bias parameters.

This gradient is then propagated to the activations of the preceding layer:

$$g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)T} g$$

In this manner all weights and biases in the ANN are repeatedly adjusted, proportionately to their contribution to the loss function at that iteration, until a (hopefully global) minimum is achieved (45).

## 4.2.3 Regularization and Optimization for Deep Learning

### 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 (45).

Regularization strategies can be achieved by, for example, constraining parameter values by adding penalty terms to an objective function or by explicitly constraining parameters. Carefully designed regularization processes can improve performance on test data by encoding prior domain knowledge, making an undetermined problem determined, or by simplifying the model so that it generalizes better (45).

Regularization in deep learning models often involves limiting the capacity (the hypothesis space) of an ANN by introducing a parameter norm penalty $\Omega(\theta)$ to the loss function J. The loss function regularized in this fashion is denoted by $\tilde{J}$, as follows:

$$\tilde{J}(\theta, X, y) = J((\theta, X, y) + \alpha\Omega(\theta)$$

Where $\alpha$ is a weighting hyperparameter, determining the extent of contribution of the parameter norm penalty to the magnitude of the regularized loss function $\tilde{J}$, i.e. setting $\alpha = 0$ eliminates regularization and increasing its value results in more regularization (45).

Various norms $\Omega$ can be used in such a setup and can be applied to the entire set of network parameters $\theta$ or a specific subset, e.g. all the weights can be regularized, but all the bias terms can be set to escape regularization, because weights encode the interaction between two variables under a variety of circumstances, whereas bias terms only affect the output of one variable (45).

Ideally, each ANN layer should have its own $\alpha$ coefficient, but doing so increases the search space for the optimal value, so a global $\alpha$ is sometimes used in practice (45).

#### *A Note on Norms*

Norms are a means of measuring the size of a vector, by mapping them to non-negative values, by satisfying the following properties:

1. $f(x) = 0 \implies x = 0$
2. $f(x + y) \leq f(x) + f(y)$
3. $\forall \alpha \in \mathbb{R},\ f(\alpha x) = |\alpha| f(x)$

In general, the $L^p$ norm is specified by:

$$||x||_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}}$$

$L^2$: *Weight Decay Regularization*

The $L^2$ parameter norm is a simple regularization strategy which shrinks the weights of an ANN closer to the origin by adding the squared and weighted parameter norm penalty

$$\lambda \Omega(\theta) = \lambda ||w||_2^2 = \frac{\lambda}{2} w^T w$$

to the objective function (45).

The $L^2$ norm is known as the Euclidean norm, because it gives the magnitude of the Euclidean distance from the origin to the point defined by $x$. It is squared in this regularization technique for computational efficiency, because calculating the derivative with respect to each component of the unsquared $L^2$ norm involves all its elements, whereas the derivative for each component of the squared $L^2$ norm depends only on the corresponding element of $x$ (45).

Figure 28 (RHS) illustrates the manner in which introducing an $L^2$ norm penalty introduces an additional constraint on the objective function, i.e. having to minimize the magnitude of the $L^2$ norm in addition to minimizing the loss function causes the weights to be shrunk, since this larger regularized loss function is interpreted as having higher variance (45).

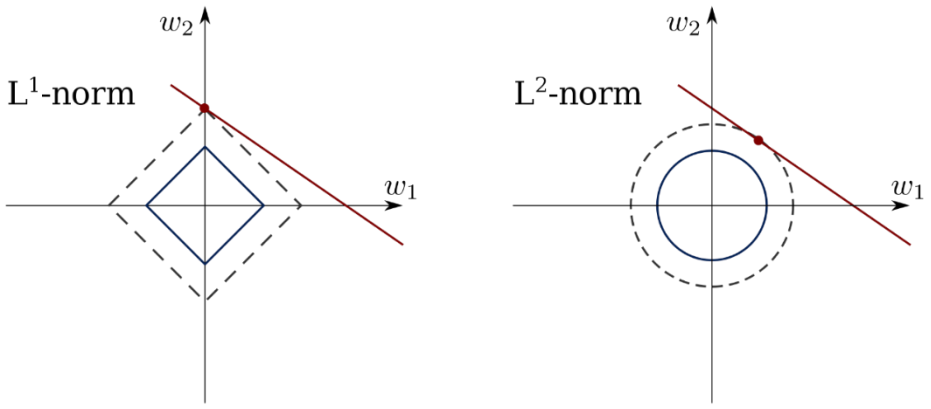



**Figure 28: $L^1$ and $L^2$ norm penalties**

### *$L^1$ Regularization*

$L^1$ regularization adds a slightly different weighted parameter norm penalty

$$\lambda\Omega(\theta) = \lambda||w||_1 = \lambda \sum_i |w_i|$$

to the objective function.

When $L^1$ regularization is used, as the sum of the absolute values of the weights of the ANN increases, the loss function will also increase, as it does for $L^2$ regularization; but in contrast to $L^2$ regularization, $L^1$ allows weights to be shrunk down to zero, resulting in a more sparse neural network, depending on the magnitude of the weighting parameter λ. This phenomenon allows for better feature selection, by reducing the amount of connections in the network and therefore removing the influence of some features on its output (45).

When using either $L^1$ or $L^2$ regularization, care has to be taken to select the right level for λ, since a large λ could result in the backpropagation algorithm getting trapped in a local minimum or where the weights are shrunk by so much that they can't impart any useful information to the next layer (45).

### *Early Stopping*

By saving the parameter setting at the conclusion of each epoch during training, one can return the network to the parameter setting where the validation error was at its lowest (the point at which the network started overfitting to the training set) (45).

One can also prevent a model from passing that point by specifying early stopping criteria, which will kick the neural network out of training when a defined minimum improvement on the validation error has not occurred for a defined number of epochs (45).

Computational efficiency is maintained by checking the abovementioned conditions at specified training intervals, i.e. not checking whether early stopping criteria have been met after each epoch. Storing parameter settings can be made more efficient by saving in a slower form of memory, such as hard disk space to keep available random-access memory or GPU memory space sufficient for model training (45).

Once early stopping has been reached, the checkpointed model can be trained further by adding the previously held out validation data to the training data and monitoring the objective function as a guide for when to interrupt training (45).

Alternatively, once early stopping criteria are reached, one can retrain a completely new neural network, with the same hyperparameters as the stopped network, for the number of epochs it ran, but this time using the full training + validation data for training (45).

Early stopping is often used in conjunction with other regularization techniques, since it is unobtrusive towards the learning dynamics, i.e. it does not change how the neural network arrives at its optimal weights, it simply changes when to stop adjusting them to prevent overfitting (45).

*Ensembled Models*

Bagging and other model averaging techniques involve training a multitude of models and allowing each of them to vote towards the outcome, making use of the principle that a number of Deep Learning models which have each been set up differently should not all make the same "cognitive errors" when learning useful representations to inform accurate predictions on the test set (45).

Bagging, in particular, requires construction of multiple training datasets by sampling with replacement from the full training dataset, resulting in around a third of the full training observations not being present in each of the resampled training sets, and different observations being missing in each (45).

Since random weight initialization and random minibatch selection can result in slightly different weight parameterisation, even when the same architecture is trained multiple times on the same dataset, model averaging is a highly reliable way to reduce overfitting (45).

Boosting is an alternative approach to ensembled methods, which actually increases the capacity of the ensemble by learning based on the variance of previous neural networks by adding additional neural networks sequentially, or even by incrementally introducing hidden units to a single ANN (45).

*Dropout*

The computational cost of training and evaluating an ensemble of more than 10 neural networks can become impractical in terms of memory and runtime constraints. Dropout is a computationally inexpensive alternative regularization method, which achieves a similar outcome (45).

Dropout regularization consists of 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 (45).

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 (45).

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 (45).

## Optimization

The essential optimization objective in deep learning is to find the optimal set of hyperparameters $\theta$ to minimize the objective function $J(\theta)$ (45).

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 (45).

### *Batch and Minibatch*

The process of minimizing the objective function ***J***, can be made more efficient by sampling a "minibatch" of training examples at each iteration, this process also compensates for redundancy in the training data, where many observations are effectively contributing the same information regarding the gradient of the loss function (45).

Using smaller batches can also prevent overfitting; this regularizing feature is optimal when using a batch size of one with a very small learning rate to maintain stability because the gradient will have high variance in this case, but the combination of a slow learning rate and high number of iterations for each epoch can result in very long compute time during training (45).

### *Stochastic Gradient Descent*

Stochastic gradient descent (SGD) is a commonly used optimization algorithm that makes use of the average gradient of the loss function over a minibatch of training examples as an unbiased estimate of the true gradient; along with a learning rate $\epsilon$, which decreases over time to compensate for noise in the gradient introduced by stochasticity of the process. The learning rate at iteration *i* is denoted as $\epsilon_i$. The learning rate is often set to decay linearly until iteration τ, i.e.

$$\epsilon_i = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

where $\alpha = \frac{i}{\tau}$. After iteration $\tau$, $\epsilon$ is commonly leaved constant (45).

### *Momentum*

Momentum, in the context of deep learning, is a method which results in accelerated learning compared to SGD, by taking an exponentially decaying moving average of past gradients into account when updating weights during backpropagation. A weighting hyperparameter $\alpha \in [0,1)$ , determines the rate of decay of previous gradients in determining this so-called momentum (45).

A simplified representation of the SGD algorithm with momentum looks as follows:

Given:

- Learning rate **ϵ**

- Momentum decay parameter $\boldsymbol{\alpha}$
- Initial velocity $\boldsymbol{v}$
- Initial parameter to be updated $\boldsymbol{\theta}$

While stopping criteria is unmet, DO:

1. Sample minibatch of size *m* from training data
2. Compute gradient:
$$g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right)$$
3. Update velocity:
$$v \leftarrow \alpha v - \epsilon g$$
4. Update parameter:
$$\theta \leftarrow \theta + v$$

Commonly used values of $\alpha$ are 0.5, 0.9 and 0.99, and similarly to the learning rate, $\alpha$ can also be adapted over time (45).

*Nesterov Momentum*

A momentum variant based on the accelerated gradient method proposed by Nesterov, Nesterov momentum updates parameters according to the following rules:

$$v \leftarrow \alpha v - \epsilon\nabla_\theta \left[\frac{1}{m} L\left(f\left(x^{(i)};\theta + \alpha v\right); y^{(i)}\right)\right]$$

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

Nesterov momentum differs from standard momentum in that the gradient is evaluated after velocity is applied, whereas in standard momentum, the gradient is evaluated first, before velocity is calculated and applied, as can be seen in the following algorithm for Nesterov momentum, compared to that for standard momentum shown above (45).

Given:

- Learning rate $\boldsymbol{\epsilon}$
- Momentum decay parameter $\boldsymbol{\alpha}$
- Initial velocity $\boldsymbol{v}$
- Initial parameter to be updated $\boldsymbol{\theta}$

While stopping criteria is unmet, DO:

1. Sample minibatch of size *m* from training data
2. Apply interim update:
$$\tilde{\theta} \leftarrow \theta + \alpha v$$
3. Compute interim gradient:
$$g \leftarrow \frac{1}{m}\nabla_{\tilde{\theta}} \sum_i L\left(f\left(x^{(i)};\tilde{\theta}\right), y^{(i)}\right)$$
4. Update velocity:
$$v \leftarrow \alpha v - \epsilon g$$
5. Update parameter:
$$\theta \leftarrow \theta + v$$

*Adaptive Learning Rates*

Since the learning rate is a very important hyperparameter and difficult to set; a variety of algorithms have been developed by the deep learning community that dynamically modify the learning rate as training progresses.

*AdaGrad*

The AdaGrad algorithm adapts each one of a model's parameters by scaling them inversely proportional to the square root of the summed historical square values of their individual gradients. In doing so, AdaGrad ensures that parameters that have a greater influence on the objective function (i.e. those that contribute a larger partial derivative to the objective function) have a rapidly shrinking learning rate $\alpha$, whereas those with smaller partial derivatives of the objective function decrease their learning rate much more slowly (45).

*RMSProp*

RMSProp is an adaptation of AdaGrad that uses an exponentially weighted moving average, therefore not taking into account all historical gradients but in essence forgetting gradients that fall outside of a specified length scale, $\rho$ (45).

*Adam*

Originating as an acronym for "adaptive moments", the Adam algorithm is generally touted as an optimization strategy robust to various settings of hyperparameters. Adam combines features of momentum and RMSProp, by using momentum to estimate the first moment of the gradient and by applying bias corrections to both the first and second order moments of the gradient (45).

## 4.3 Convolutional Neural Networks

### 4.3.1 The Kernel Concept and Motivation for CNNs

Convolutional Neural Networks (CNNs) are an extension of deep learning models, highly successful in processing data with a grid-like topology, e.g. images. At least one linear mathematical operation, called a convolution, is applied in CNNs, usually in addition to the general matrix multiplication performed in traditional feedforward neural networks (45).

An example of a simple 2D convolution (multiplying a 3×4 matrix by a 2×2 kernel) is shown below (adapted from (45)).

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

=

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

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 (45). 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, such as faces. 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 (45).

Parameter sharing allow certain parameters to be used by more than one function in a CNN, unlike traditional neural networks, which use each weight in a neural network in just one operation when the network's output is calculated. In a CNN, each element of the kernel is multiplied by every element of the input matrix (where dimension differences do not allow for this, edges may be padded with zero-valued matrix elements to enable it). The weights of the kernel function are learnt and applied uniformly, i.e. they are not relearned at each position of the input matrix, again this has benefits with regards to computational efficiency (45).

Equivariance to translation is a phenomenon which results from parameter sharing and means that the output of a convolutional layer changes in the same way that its input changes, i.e. $f(x)$ is said to be equivariant to a function $g$ if $g\big(f(x)\big) = f(g(x))$. In a convolution operation, the function $g$ translates (shifts) the input matrix in some way, but since the convolution operation is equivariant to the function $g$, it does not matter at which (x,y) coordinates a feature occurs in the input matrix, since it will still result in the same output after the convolution operation has been applied (45).

### 4.3.2 Pooling

CNN layers are generally composed of three operations:

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 (45).

Pooling serves the purpose of insuring invariance to local translation, where the presence of a feature matters more than its location. In some cases, the specific orientation and location of a feature does matter though. Pooling over separate convolutions that are independently parameterized can allow the ANN to learn which translations it should be invariant to which translations it shouldn't be invariant to (45).

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 29 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 (45).

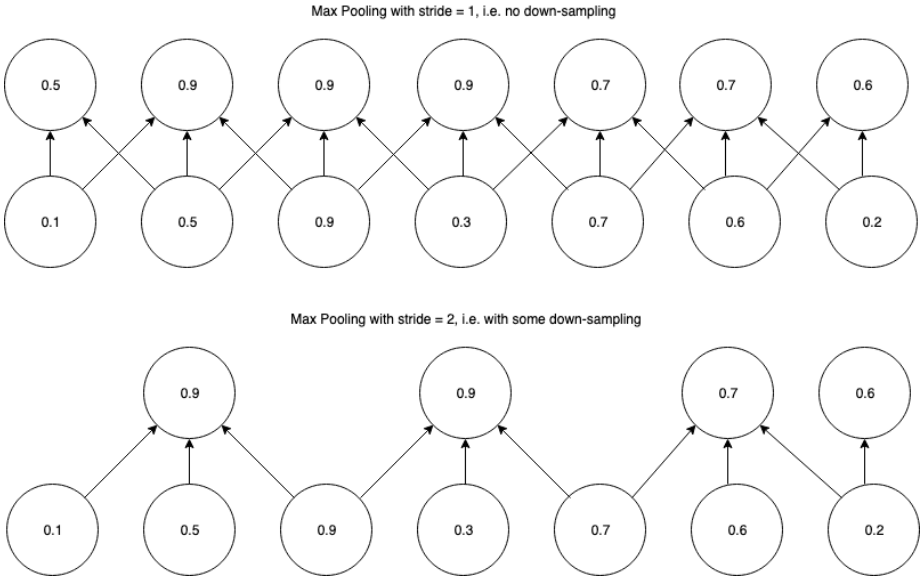



**Figure 29: An illustration of the concept of max pooling, using pool-width of 3 with a stride of one (top panel) vs a stride of two (bottom panel) (48).**

## 4.3.3 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 (45).

The Convolution Operation

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

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

where the summation over the indices is for all valid indices in the tensor (45).

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 (45).

Figure 30 illustrates how implementing a convolution with stride = 2, i.e. only sampling every second pixel for convolution, is mathematically equivalent to performing downampling after a convolution applied to all pixels (i.e. stride = 1), followed by downsampling (45).

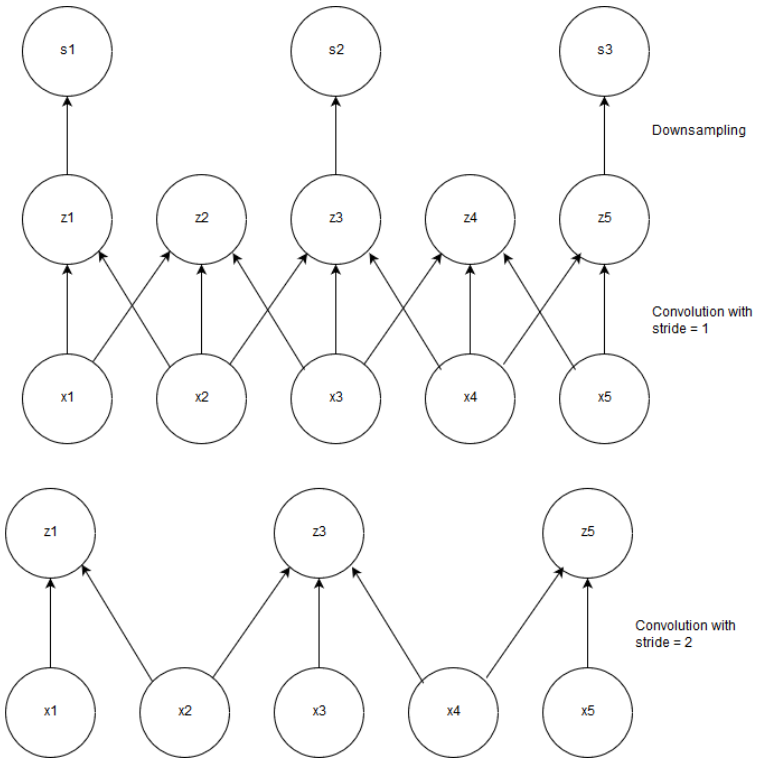



**Figure 30: Illustration of mathematical equivalence of implementing a convolution with unit stride followed by downsampling to implementing a convolution with stride = 2.**

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 (45).

Convolutions applied with no padding of the input image are known as valid convolutions, where pixels in the output of a convolution are a function of the same amount of pixels in the input, and the kernel can only be applied to positions on the image where the kernel is contained by the image (45).

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 (45). Although same convolutions do not limit the size of the network and allow one to build neural networks of arbitrary depth, they still result in pixels close to the edges of the image having less connections to the output image and therefore that their influence on the network as a whole will be reduced (45).

Full convolutions result from applying enough zero-padding to allow each pixel to be visited *k* times in each direction of the convolution operation, and therefore should result in an output with *m*+*k*-1 pixels. This results in output pixels near the border being influenced by fewer pixels than output pixels near the center, making the kernel harder to train (45).

The ideal amount of padding generally lies between the amount of padding required to achieve valid- and same convolutions (45).

## 4.4 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 (45).

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 (45).

The input vector to an RNN will be vectors $x^{(t)}$ with timestep *t* consisting of a range from 1, …, $\tau$ (45).

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 (45).

### 4.4.1 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. $H = \max\{0, WW + b\}$, would look as follows:

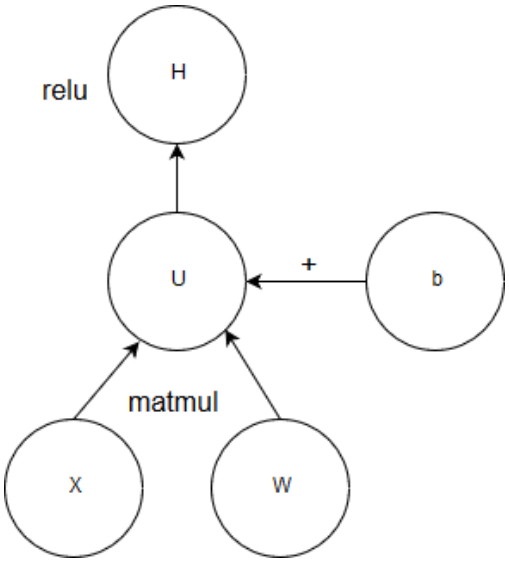



**Figure 31: ReLU activated hidden unit in a Neural Network depicted as a computational graph**

In RNNs, computational graphs manifest as repetitive chains of operations which result in parameter sharing across neural network architectures (45).

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

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

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

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

$$s^{(3)} = f\left(s^{(2)}; \theta\right)$$
$$= f\left(f(s^{(1)}; \theta); \theta\right)$$

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

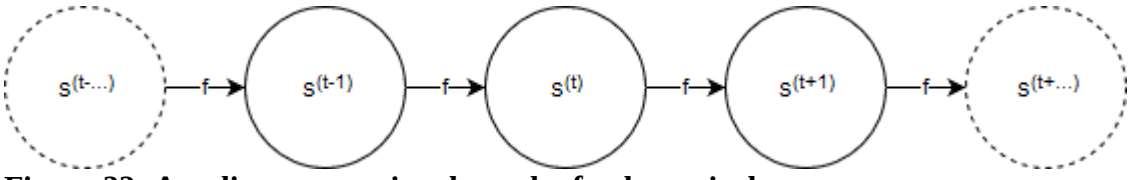



**Figure 32: Acyclic computational graph of a dynamical system**

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:

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

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.

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

(45).

### 4.4.2 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 33 (49).

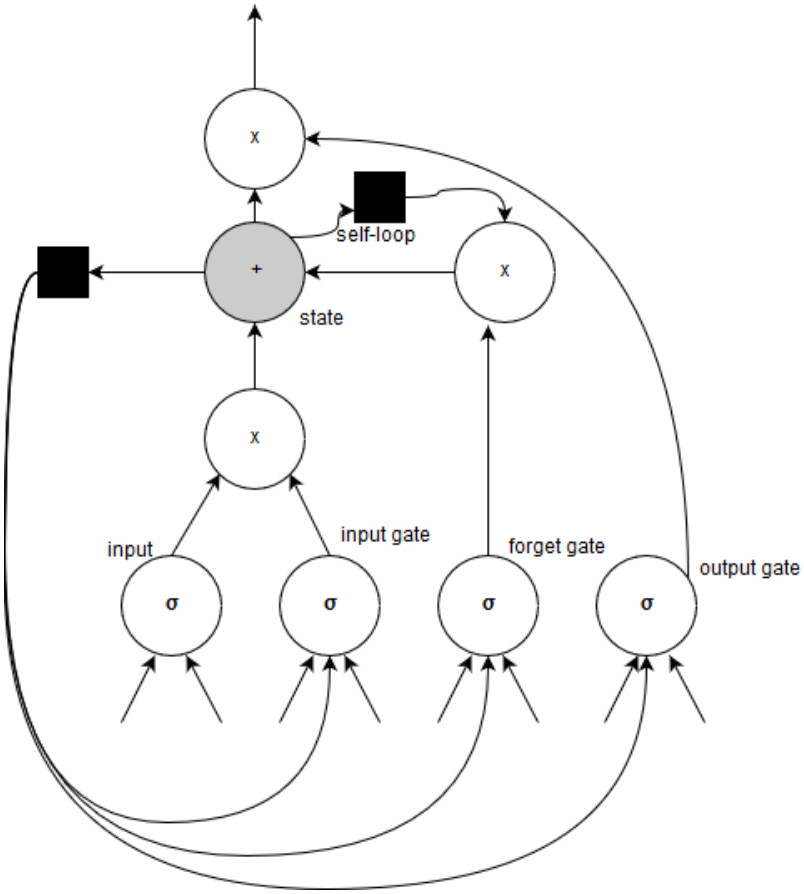



**Figure 33: Graph-based representation of special LSTM units**

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 (49). 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.

## 4.5 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 (50).

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 $P_{gt}(X)$ and we want to model a distribution $P$ which is as similar as possible to $P_{gt}$ and therefore allows us to generate new examples $X$ by sampling from $P$ (50).

Neural networks can be utilised as function approximators towards constructing a modelled distribution $P$ as outlined above (50).

### 4.5.1 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 labels seen during training, before the generative process commences, i.e. the model first chooses what it is going to simulate before it starts simulating (50).

In order to deduce that a generative model is representative, one needs to find that for each datapoint $X$ in $\chi$, there are one or more latent variable settings which result in the model generating something sufficiently similar to $X$ (50).

A vector of latent variables $z$, are sampled from a high dimensional latent space $Z$, according to a probability density function (p.d.f.): $P(z)$ defined over $Z$. A group of deterministic functions $f(z;\theta)$ parameterized by a vector $\theta$ in some space $\Theta$, with $f: Z \times \Theta \rightarrow \chi$. While $f$ is deterministic, $z$ is randomly sampled and $\theta$ is fixed, which makes $f(z;\theta)$ a random variable in the space $\chi$. $\theta$ 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 $X$ (50).

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

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

$f(z;\theta)$ has been changed to a distribution $P(X|z;\theta\,)$ in the expression above, in order to show explicitly that $X$ depends on $z$. Maximum Likelihood underpins the notion that if $X$ is likely to be reproduced, generated examples that are highly similar to $X$ are also likely to be produced, and dissimilar examples are unlikely (50).

VAEs 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 $\sigma$ multiplied by the identity matrix $I$, with $\sigma$ being a tuneable hyperparameter (50).

A VAE 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 $z$ (50).

### 4.5.2 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 generally not just defined by the label of the example that needs to be generated, but by other features specific to the example (50), in our case, $z$ would not just be *electron* or *pion*, but additional dimensions such as the particle's momentum, angle, etc. Generally, a researcher would not explicitly specify what the dimensions of $z$ specify, nor how the dimensions of $z$ depend on one another (50).

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 neural network then the initial layers will be involved in generating $z$ while the later layers 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 ascent (50).

$P(X)$ 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 (50).

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 $P(z)$ (50).

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

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

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

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

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 us:

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

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 in our case we are interested in accurately inferring $P(X)$ and therefore we want to find a $Q$ which does depend on $X$ and which also keeps $\mathcal{D}[Q(z)||P(z|X)]$ as small as possible:

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

The formula above 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 and the direct minimization of $\log P(X)$, in addition $P(z|X)$ is no longer intractable 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:

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

Where $\mu$ and $\Sigma$ are deterministic functions with learnt parameters $\vartheta$; in practice $\mu$ and $\Sigma$ are learnt via neural networks and $\Sigma$ 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:

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

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

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

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 we are doing stochastic gradient descent over different $X$ values from our dataset $D$, we want to perform gradient descent on the following formula:

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

By sampling a single value of $X$ and a single value of $z$, we can compute the gradient of $\log P(X|z) - \mathcal{D}[\mathrm{Q(z|X)||P(z)}]$, 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 34 illustrates how this proxy formula can be used by averaging over multiple samples to get to the expected outcome, but since there is a sampling procedure embedded within the neural network, gradient descent cannot be performed on it.

Figure 35, on the other hand, shows how a "reparameterization trick" removes he sampling procedure from the neural network proper and treats it as an input layer. Since we have $\mu(X)$ and $\Sigma(X)$, we can sample $\epsilon$ from $N(0, I)$ and compute $z$ from $\epsilon$ as follows: $z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon$.

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

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

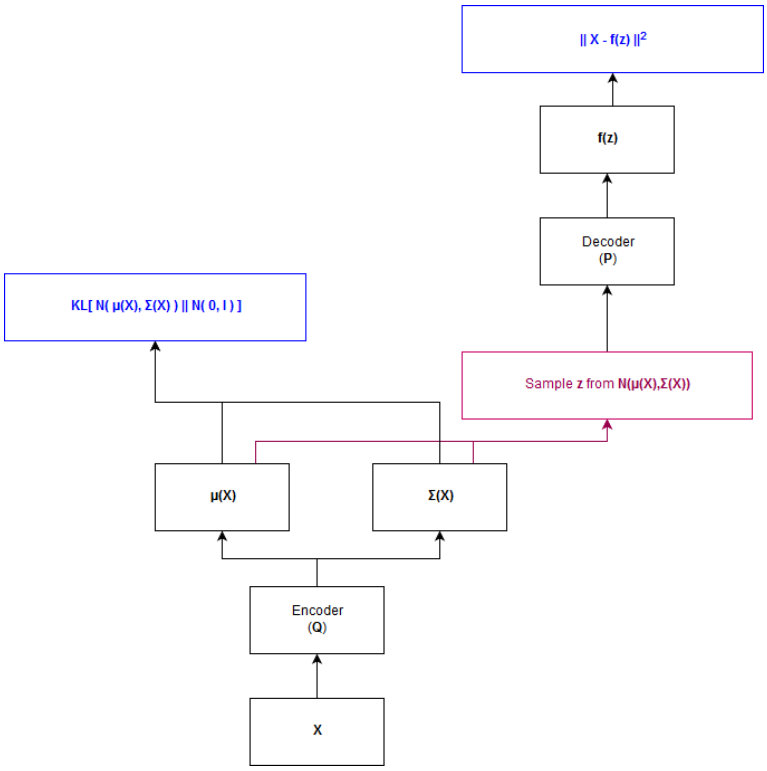



**Figure 34: Training-time VAE**

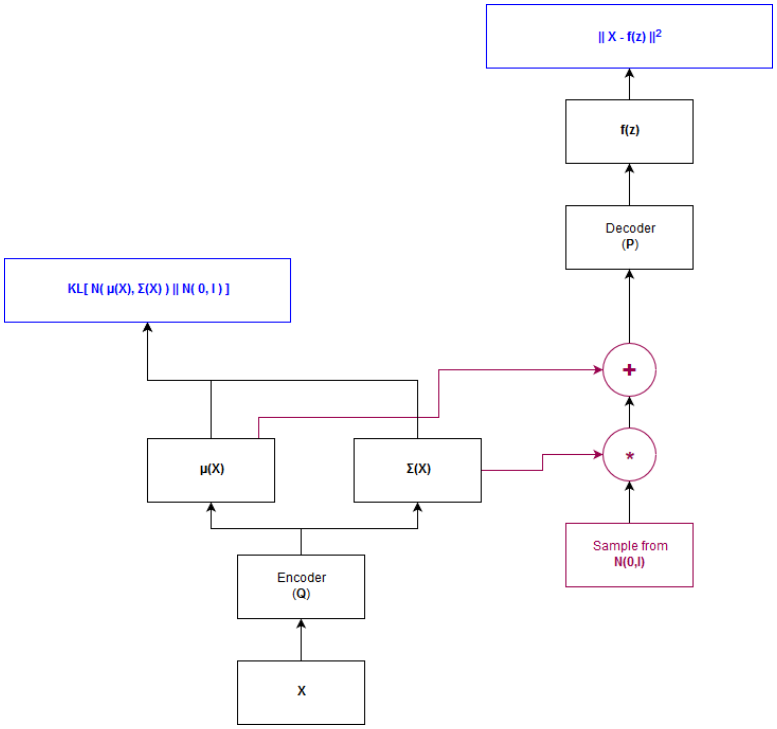



**Figure 35: Training-time VAE with reparameterization trick to enable backpropagation**

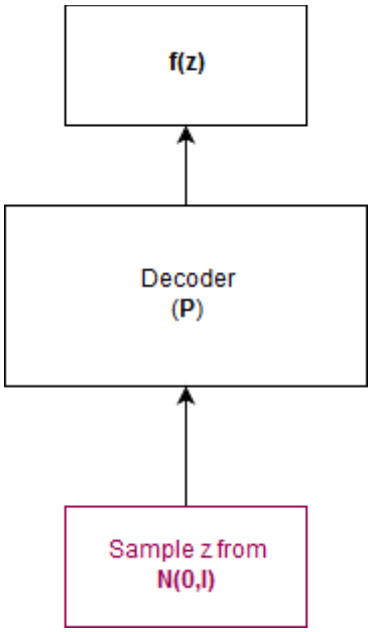



**Figure 36: Testing time VAE**

Once the model is ready to be tested, values from $z \sim N(0, I)$ are sampled and fed to the decoder; the encoder, along with the attendant reparameterization trick used during training are thrown away.

## 4.5.3 Generative Adversarial Networks

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 (51).

The mini-max game can be expressed mathematically as:

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

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

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

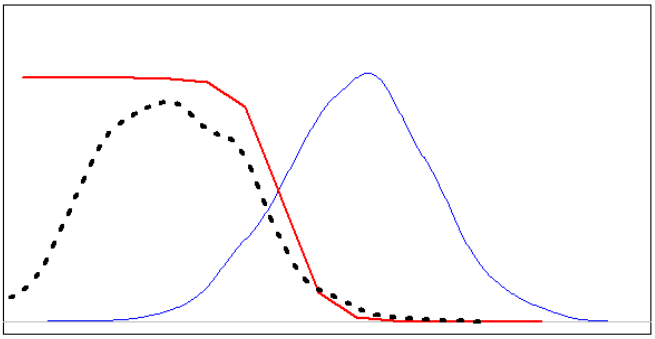


**Figure 37: Gan Densities during training, close to convergence, P(x) is shown in black, G(z) in blue and D(G(z)) in red**

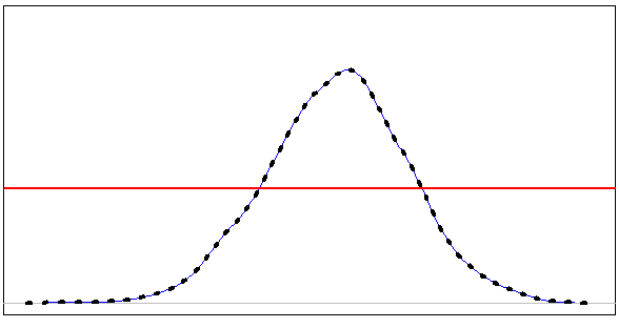


**Figure 38: Gan Densities during training, once the Algorithm has converged, G(z) matches P(x) perfectly and D(G(z)) outputs 0.5 everywhere**

### 4.5.4 Variations on the GAN concept used towards Event Simulation in this Dissertation

#### Auxiliary Classifier Generative Adversarial Network

Auxiliary Classifier Generative Adversarial Networks enforce class label conditioned synthesis models, i.e. by feeding a GAN the class label of an image $c \sim p_c$, $G$ is now a function of both a noise vector and the label of the image, i.e. $G(z, c)$, whereas $D$ is not only tasked with classifying whether data is real or simulated, but also the class label of the image. This process has been shown to stabilize training (52).

#### Adversarial Autoencoder

Adversarial Autoencoders match the aggregated posterior of the latent space vector from an autoencoder $q(z) = \int_x q(z|x)p_d(x)dx$ with an arbitrary prior distribution $p(z)$, a process which results in meaningful samples being generated from any sample from any part of the prior space. The decoder function learns a function to map from the imposed prior distribution to the data distribution. In this set-up, the generator of the GAN also acts as the encoder function of the autoencoder, a process which assists the generator in fooling the discriminator of the GAN into misclassifying simulated data as real data (53).

#### Bidirectional Generative Adversarial Network

Bidirectional Generative Adversarial Networks (BiGANs) make use of an encoder function which maps data $x$ into a latent feature space $z$ for the generative model, i.e.: $E: \Omega_x \rightarrow \Omega_z$. Here the discriminator has access to both the (simulated or real) $x$, as well as its latent encoding $z$ when classifying samples as real or simulated. In this way, BiGANs not only learn how to map from a latent space to data, but how to perform the inverse mapping, i.e. data to latent space (54).

Deep Convolutional Generative Adversarial Network

Deep Convolutional Generative Adversarial Networks make use of fully convolutional neural networks for both the generator and discriminator, using strided convolutions to allow these networks to learn their own spatial downsampling in the case of the discriminator and spatial upsampling in the case of the generator. These models also make use of Batch Normalization, which ensures that the input to any hidden unit has zero mean and a variance of 1, a process which compensates for poor initialization strategies, helps with the flow of gradients through complex models and preventing the generator from outputting similar simulated images from any sample of the noise space, a common issue which plagues GANs (55).

Least Squares Generative Adversarial Network

Whereas regular GANs use the sigmoid cross entropy loss function, which results in vanishing gradients when samples are generated that are classified as real data, but that are still far from looking like real data, Least Squares Generative Adversarial Networks (LSGANs) use an adaptation of the least squares loss function for the discriminator network, which has been shown to result in images of higher quality and result in networks that are more stable during training (56).

# 5 STATISTICAL TESTS

## 5.1 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 (57).

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. $f(x|H_0), f(x|H_1), \dots$

In order to assess how well the observed data agrees with any given hypothesis, a test statistic $t(x)$, which is a function of the observed data, is constructed.

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

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, without losing discriminative power.

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 39 illustrates how setting a threshold value for the test statistic, i.e. $t_{cut}$, results in rejection of the null hypothesis when $t > t_{cut}$.

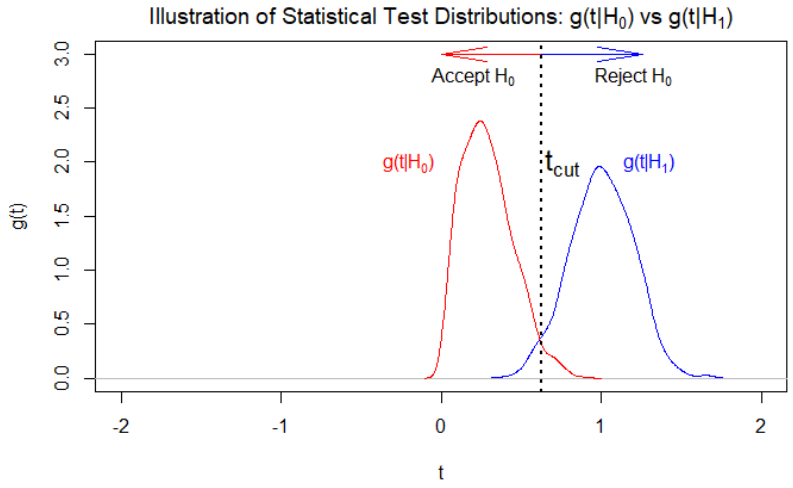



**Figure 39: An illustration of rejection or acceptance of the null hypothesis, under the assumed distributions of $H_0$ and $H_1$, when t falls in the critical region $t > t_{cut}$**

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

## 5.2 Significance Level and Power

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 value $\alpha$, called the significance level of the test.

In the example shown in Figure 39, a critical region is defined by a value: $t_{cut}$, which defines the lower decision boundary for rejecting the null hypothesis.

The significance level defined as such is given by

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

$H_0$ would not be rejected when $t < t_{cut}$, and there is a probability of $\alpha$ of rejecting $H_0$ when $H_0$ is in fact true (called an error of the first kind), as well as a probability of accepting $H_0$ when $H_1$ was actually true. The probability of making an error of the second kind is given by

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

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

## 5.3 Statistical Tests for Particle Selection

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

When looking at the probability of classifying a specific particle as a given type, we define the selection efficiencies, i.e. the electron efficiency $\varepsilon_{\text{e}}$ and pion efficiency $\varepsilon_{\pi}$ as follows:

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

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

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.

Based on the probability of a particle being an electron obtained from each of the 6 detector layers in the TRD, we use a Bayesian approach outlined in the formula below:

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

Here, $P_j(elec)$ is the probability of the track being an electron obtained from layer $j$.

# 6 DATA

## 6.1 LHC Runs Used

- 000265377
- 000265378
- 000265309
- 000265332
- 000265334
- 000265335
- 000265336
- 000265338
- 000265339
- 000265342
- 000265343
- 000265344
- 000265381
- 000265383
- 000265385
- 000265388
- 000265419
- 000265420
- 000265425
- 000265426
- 000265499

## 6.2 Data Structure

An example of the data obtained for a single track can be viewed at https://github.com/PsycheShaman/MSc-thesis/blob/master/NEW/example_pythonDict.txt. This data structure consists of a header section with meta-information about the track, i.e.

- The run-number and event the track was obtained from
- The V0 Track ID from which the track originated
- A track number, which is a unique identifier for the track in that event and run number
- A PDG code, which is 11 for electrons -11 for positrons, 211 for pions and -211 for *pion's antiparticle*
- n sigma electron and n sigma pion which are the number of standard deviations away from the expected electron- and pion signal, respectively
- The transverse momentum of the particle
- Information about the angle at which the particle is traveling, i.e. Eta, Theta and Phi
- The detector number, row and column the particle went through in layers 1-6 (indexed from 0-5 because of Python's indexing strategy)
- The raw data signal caused by the track as it traversed the six layers of the TRD

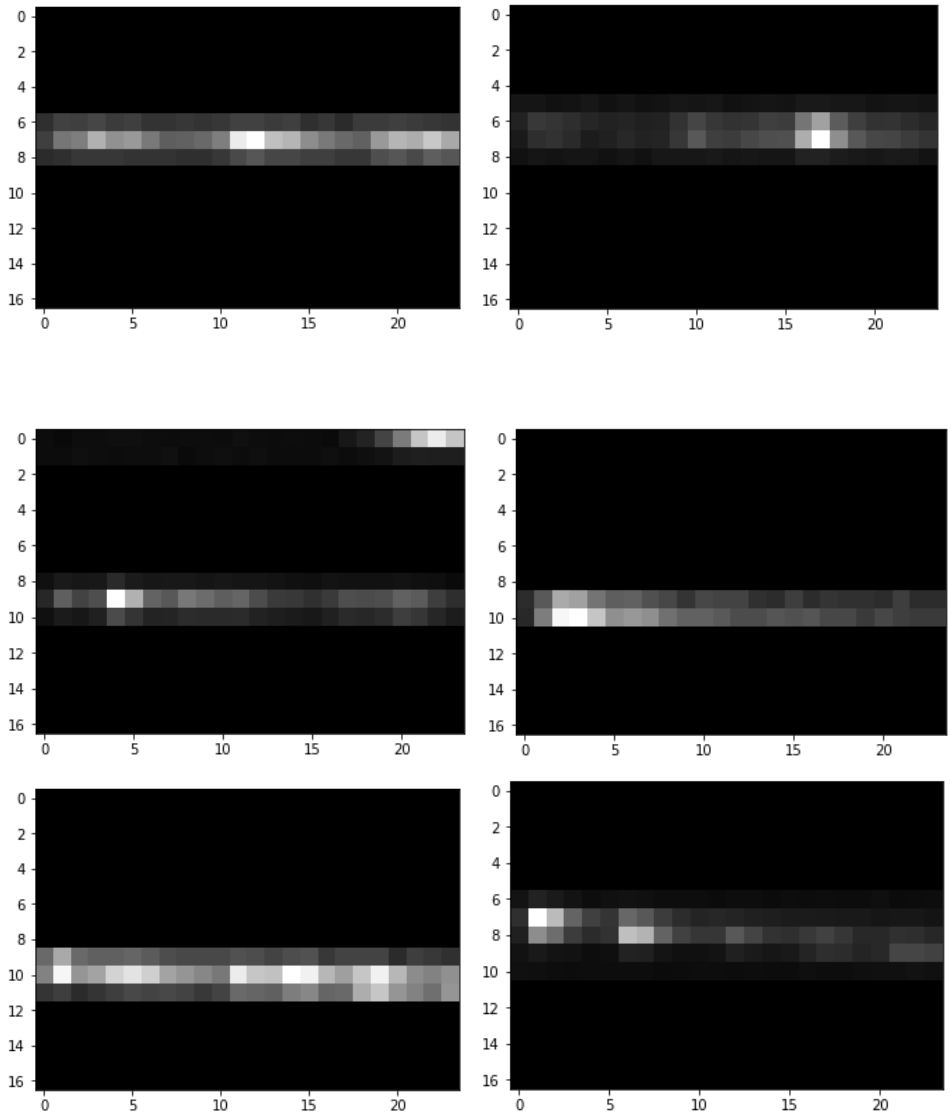
The raw data signal is in the form of a matrix of 17 rows and 24 columns, the columns of which represents the signal in each of the 24 time-bins for which data was taken, and the rows containing the specific pad in which most of the charge was deposited, along with the pads alongside it, in which charge also gets deposited.

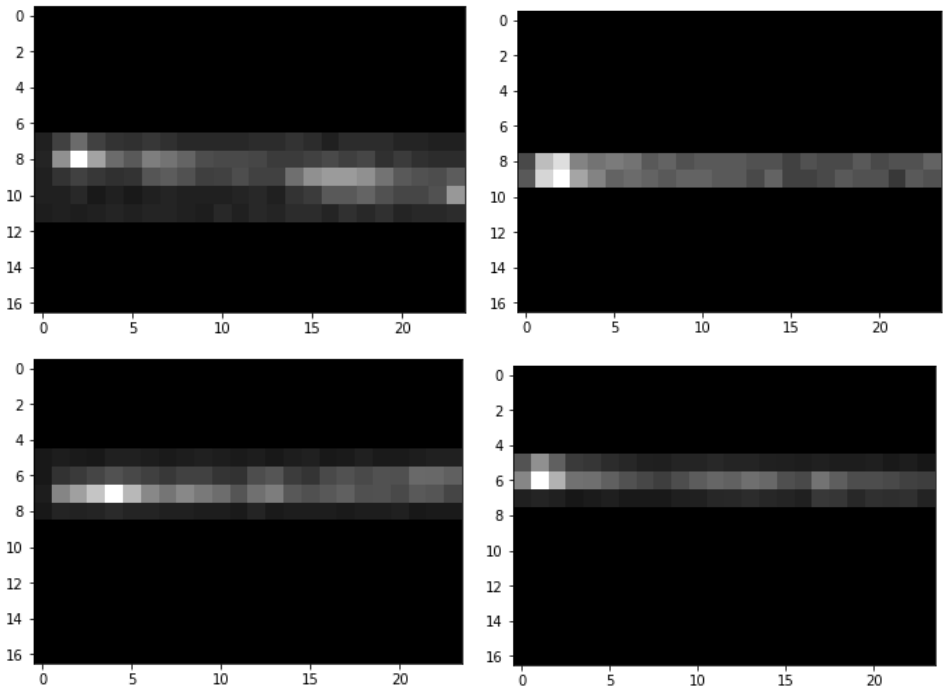
Most rows are filled with zeros, since a safety margin was incorporated, in case tracking was not accurate enough and this therefore makes sure that the signal for the track is in fact returned.

# 6.3 Graphical Overview of Data

## 6.3.1 Example Images of Tracklet Signals

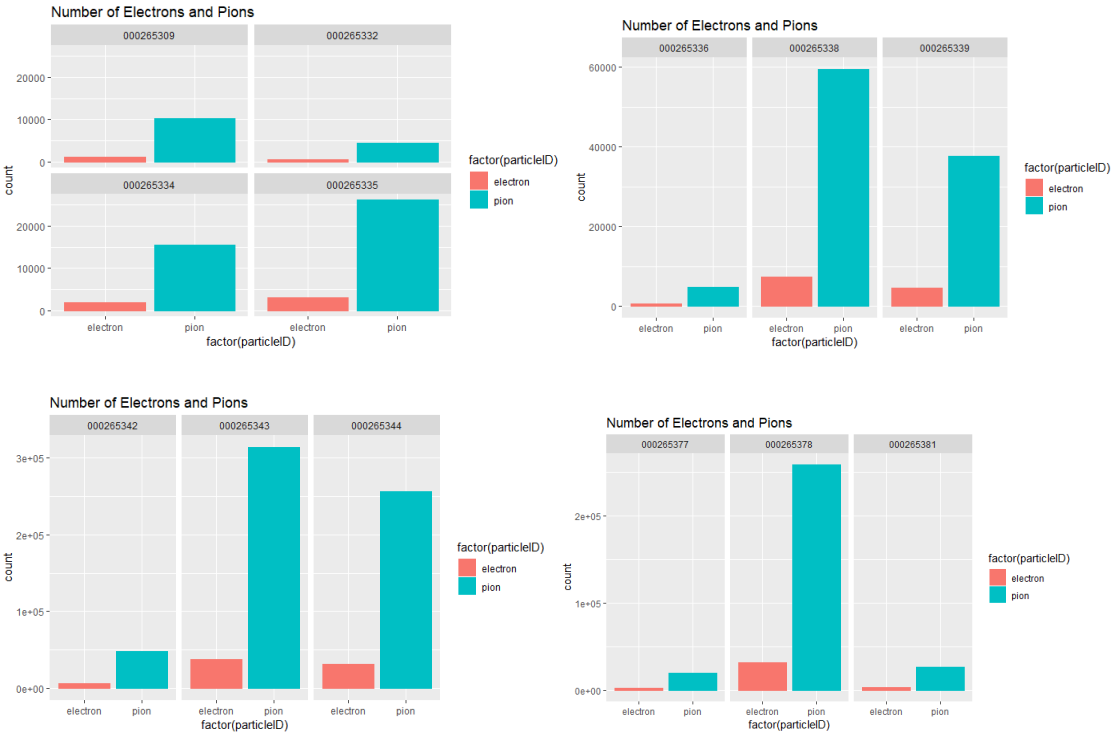
Below are 10 examples of single tracklet signals, each from a single layer of the TRD. In the images below, the signal for 17 pads in the TRD layer were added (along the rows of the image), in order to ensure that the signal for the tracklet will be captured in case tracking wasn't accurate enough. The columns in the below image represent the charge deposited at a specific time within the pad, i.e. moving across the pixels in a certain row of the image, gives an indication of the time-evolution of the signal as charge is deposited.

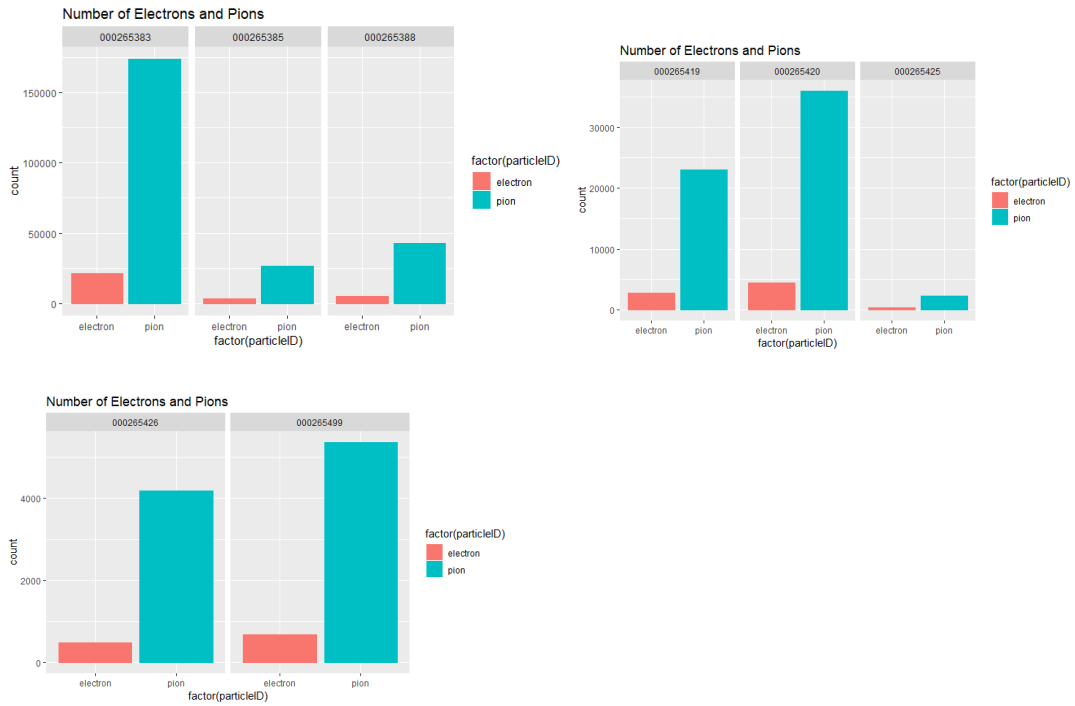




## 6.3.2 Electron and Pion Counts per Run

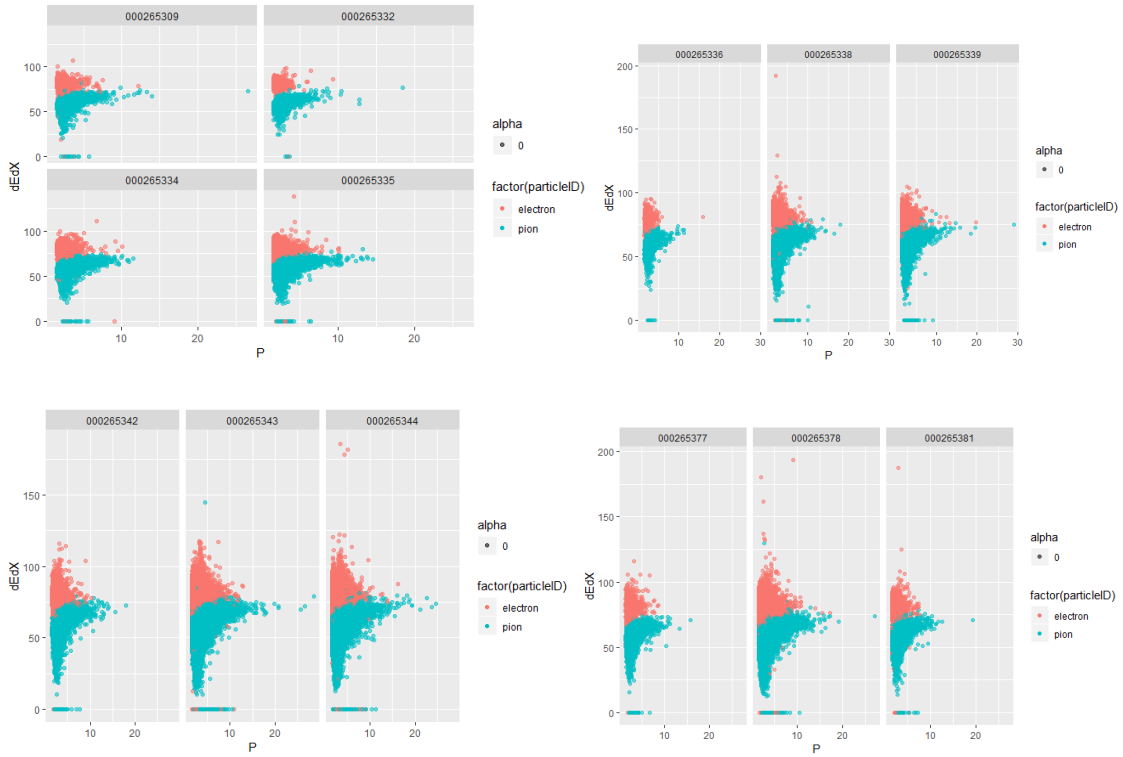
What follows below is a graphical overview of the number of electrons and pions which were measured in each run, from which it is immediately obvious that there is an overwhelming majority of pions detected.





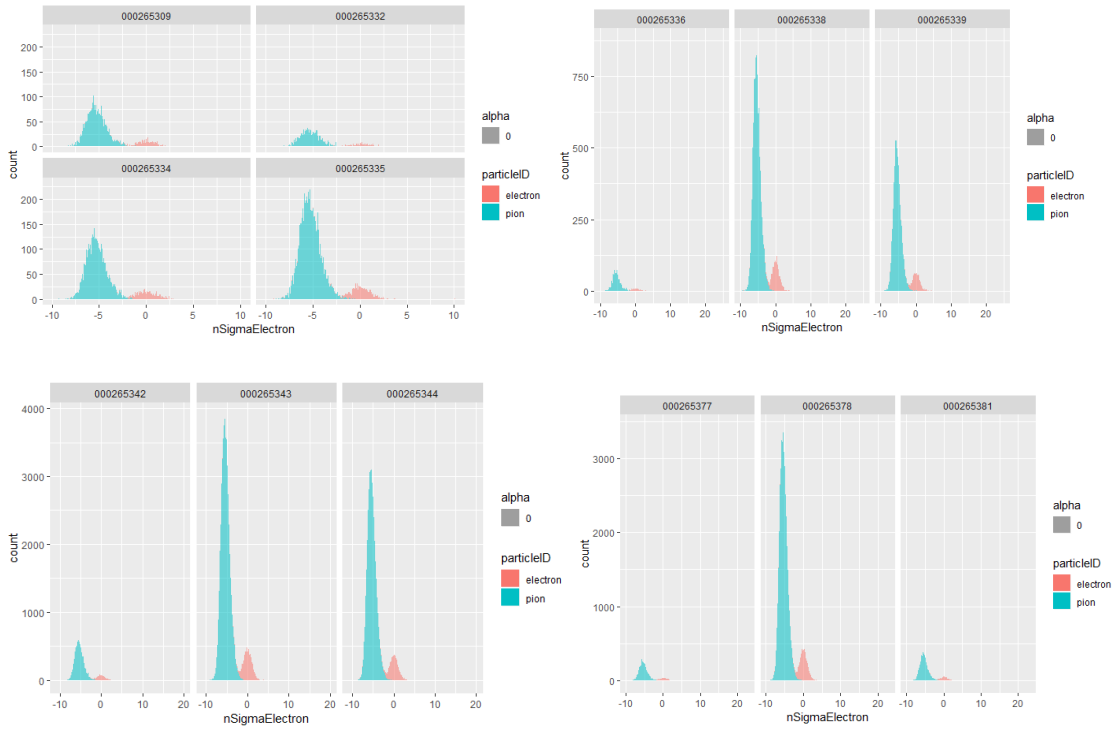

## 6.3.3 Bethe Bloch Curve per Run for Electrons and Pions

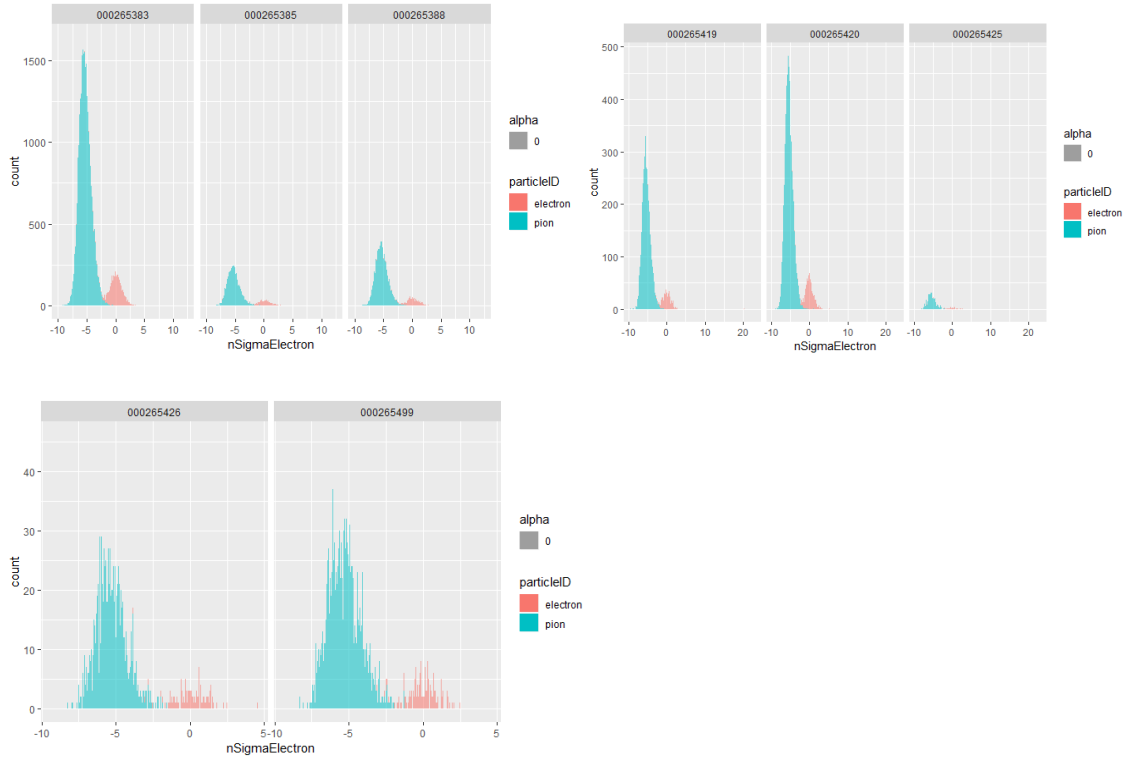
The plots below depict the energy loss as a function of detector material traversed, i.e. the Bethe-Bloch curves for electrons and pions, respectively. It is clear from these plots that electrons deposit proportionately more energy than pions in the lower GeV range.




## 6.3.4 nσ Electron per Run for Electrons and Pions

In the plots below, the statistical estimate for electron and pion identification is depicted, it is clear from the plots below that particles with a low nσ Electron value have been classified as electrons. Pions are centered around an nσ Electron value of around -5.




000265383
000265385
000265388
000265419
000265420
000265425
000265426
000265499
count
nSigmaElectron
alpha
0
particleID
electron
pion

# 7 METHODS

## 7.1 Data Extraction

Using the AliPhysics installation on the Hep01 cluster in the Physics Department at UCT, an AliRoot macro ana.C (https://github.com/PsycheShaman/trdML-gerhard/blob/master/ana.C) was modified from a version developed by other collaborators in the SA-ALICE group.

This script interfaces with https://github.com/PsycheShaman/trdML-gerhard/blob/master/AliTRDdigitsExtract.cxx, also modified from a previously developed C++ file, TRD digits were extracted and filtered for the runs specified in Chapter 5. By redirecting the C++ standard out to a text file, the relevant data was saved into Python dictionaries for each run.

Jobs were submitted onto the WLCG and monitored using http://alimonitor.cern.ch/. Upon completion, data was extracted back onto Hep01 using the aliensh environment and from there was transferred to a local machine using rsync.

Data was backed up in a semi-private GitLab repository, accessible by CERN members, at https://gitlab.cern.ch/cviljoen/msc-thesis-data.

For work done in Python, both raw digits and particle IDs were extracted into numpy arrays, using https://github.com/PsycheShaman/trdpid/blob/master/py_datatools/extract/dataset_generator.py a script developed by an SA-ALICE collaborator, Jeremy Wilkinson.

For work done in R, python dictionaries were transformed into JSON files using https://github.com/PsycheShaman/MSc-thesis/blob/master/NEW/data_preprocessing/cat_files.py, which can be read in by R.

## 7.2 Deep Learning for Particle Identification

Various deep neural network architectures were developed towards particle identification, using Keras with a Tensorflow back-end.

The following repositories host the code used to build and train feedforward, convolutional and LSTM neural networks towards particle identification:

https://github.com/PsycheShaman/msc-hpc

https://github.com/PsycheShaman/hpc-mini

https://github.com/PsycheShaman/MSc-thesis

A summary of the input tensors used for the various types of deep learning architectures mentioned above is as follows:

- Feed-forward neural networks were fed with time-bin summed data, i.e. the 17*24 matrices were collapsed down to 1*24.
- 2D convolutional neural networks were fed data with one image channel added to the 17*24 matrices, i.e. 17*24*1

- 1D convolutional neural networks were fed data with one channel added to the 1*24 timebin collapsed matrices, i.e. 1*24*1
- LSTM neural networks were fed data with the 17*24 matrices transposed to 24*17, i.e. 24 time-bins with 17 features

Class imbalances were accounted for by downsampling the pion sample to be equal in size to the electron sample.

Data was normalized as follows:

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

The SLURM-managed High Performance Computing Cluster at UCT was utilized extensively to test the performance of various deep learning architectures, enabling one to run various deep learning models in parallel.

## 7.3 Deep Learning for Distinguishing Geant4 data from real data

Geant4 based simulations were configured using https://github.com/PsycheShaman/trdpid/blob/master/sim/Config.C , simulations were run as per the following shell script https://github.com/PsycheShaman/trdpid/blob/master/sim/runtest.sh which calls upon the simulation script https://github.com/PsycheShaman/trdpid/blob/master/sim/sim.C the reconstruction script https://github.com/PsycheShaman/trdpid/blob/master/sim/rec.C and the analysis script https://github.com/PsycheShaman/trdpid/blob/master/sim/ana.C in sequence in order to create Monte Carlo simulations in a similar format to raw data analysed during particle identification.

Before commencing deep learning, the dataset was sanitized to ensure that similar ranges for key variables were covered by both real and simulated data, Eta distributions for real and simulated data were similar.

The following cut on Eta was applied to both datasets:

$$|\eta| \leq 0.9$$

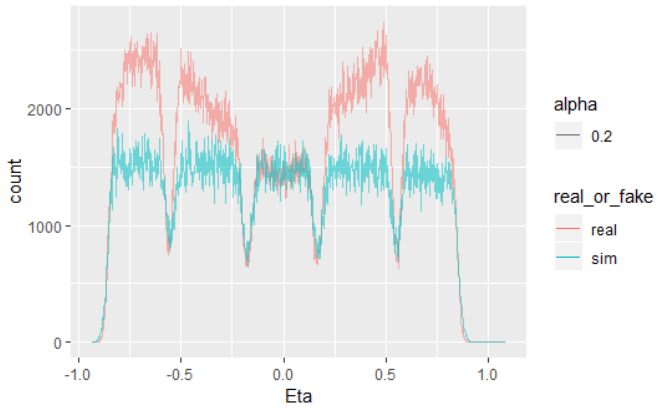



**Figure 40: Eta distributions for real and Geant simulated data**

nσ-pion estimates from the Time Projection Chamber (TPC) were dissimilar for real and simulated data (nσ = -999 is an error flag).

The following cut on nσ was applied to real and simulated data:

$$|n\sigma_{\pi}| \leq 3$$

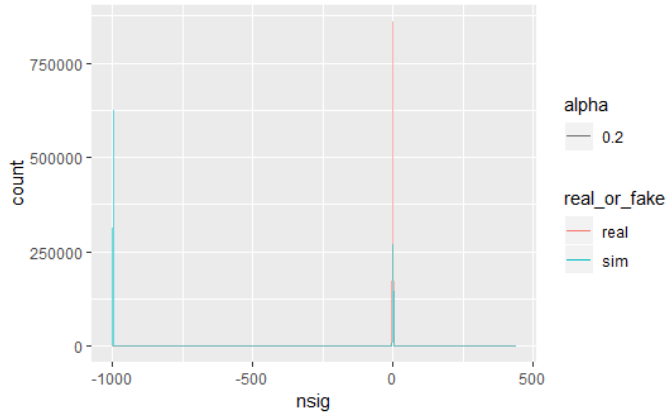



**Figure 41: nsigma pion estimate (TPC) distributions for real and Geant simulated data, before cut**

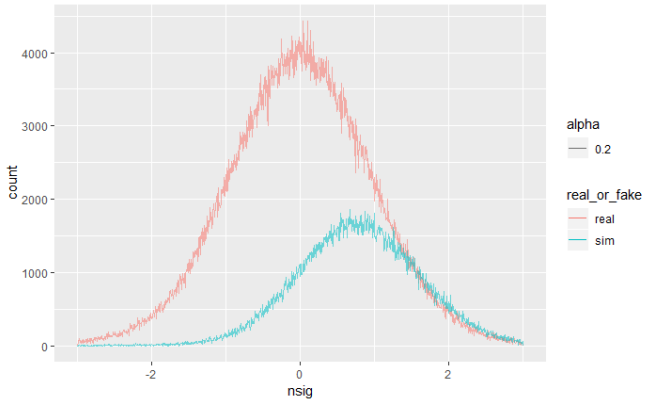



**Figure 42: nsigma pion distributions after applying cut**

Momentum ranges for real and simulated data were also quite dissimilar, the following cut on momentum was applied to both real and simulated data:

$$1.5 \leq |P| \leq 20$$

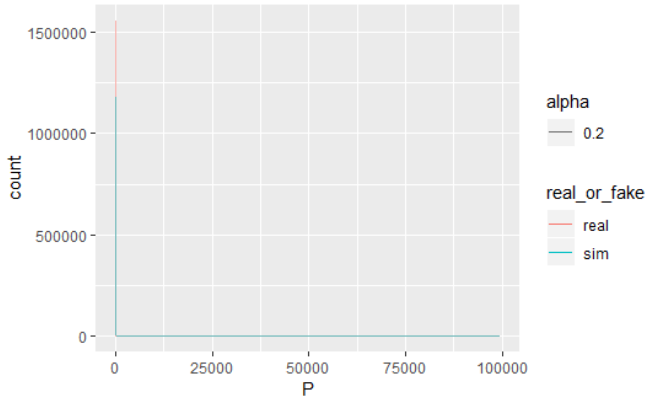



**Figure 43: Momentum distributions for both real and Geant simulated data, before cut**

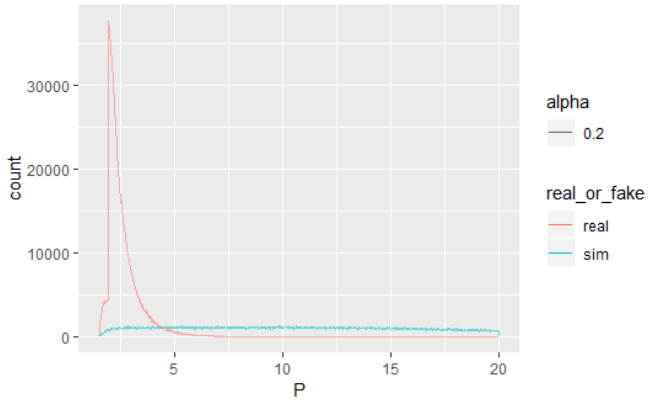



**Figure 44: Momentum distributions after cut**

The task of distinguishing simulated from real data was performed using convolutional neural networks with similar model architectures which resulted in low pion efficiencies at 90% electron efficiency during the first stage of the project.

## 7.4 Deep Generative Models Towards Event Simulation

Towards developing a prototype for event simulation, the following models were built:

- Autoencoders
- Variational Autoencoders
- Various types of Generative Adversarial Networks

The following repositories contain code used to build and run Deep Generative Models for this project:

https://github.com/PsycheShaman/deep-gen

https://github.com/PsycheShaman/Keras-GAN (forked from https://github.com/eriklindernoren/Keras-GAN and adapted to be able to work with data from this project).

# 8 RESULTS

# 8.1 Deep Learning for Particle Identification

## 8.1.1 Most useful model

After testing all the architecture possibilities, with various levels of hyperparameter settings, i.e. learning rate, activation functions, optimizers, batch sizes, epochs, number of convolutional/ recurrent/ dense layers, the following model achieved the lowest pion efficiency $\varepsilon_{\pi} = 2.2\%$ at electron efficiency $\varepsilon_{e^-} = 90\%$:

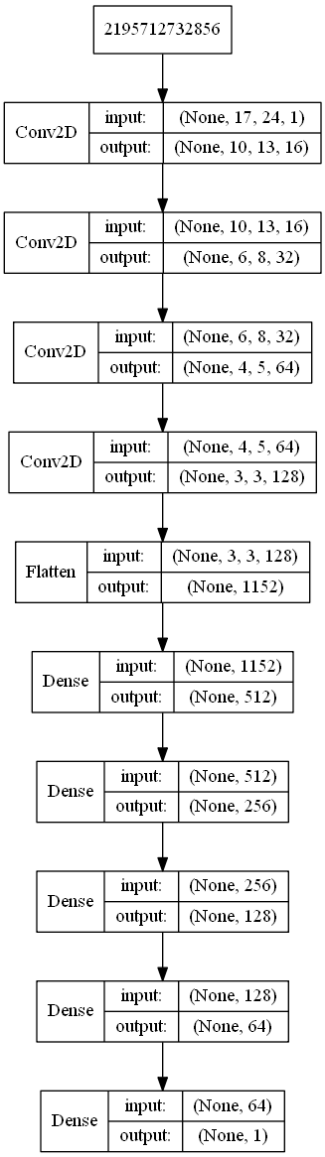



**Figure 45: Particle Identification Model Architecture**

Using the Adam optimizer with learning rate = 0.00001, 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.

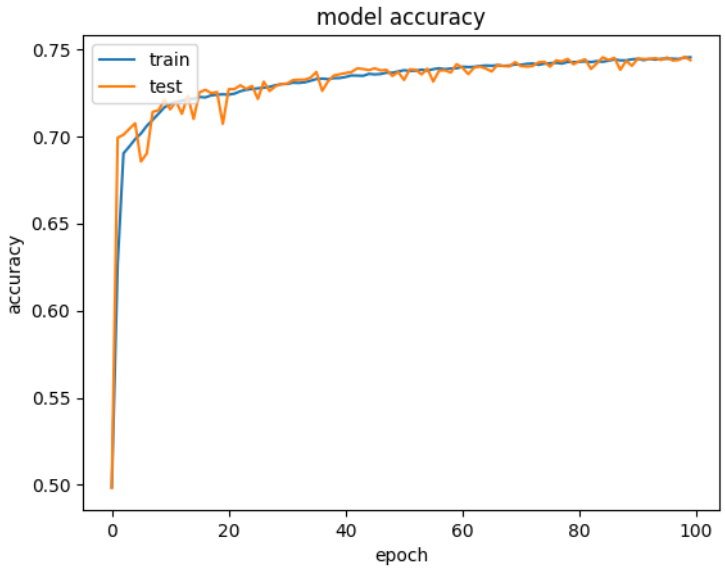



**Figure 46: Training vs Validation Accuracy**

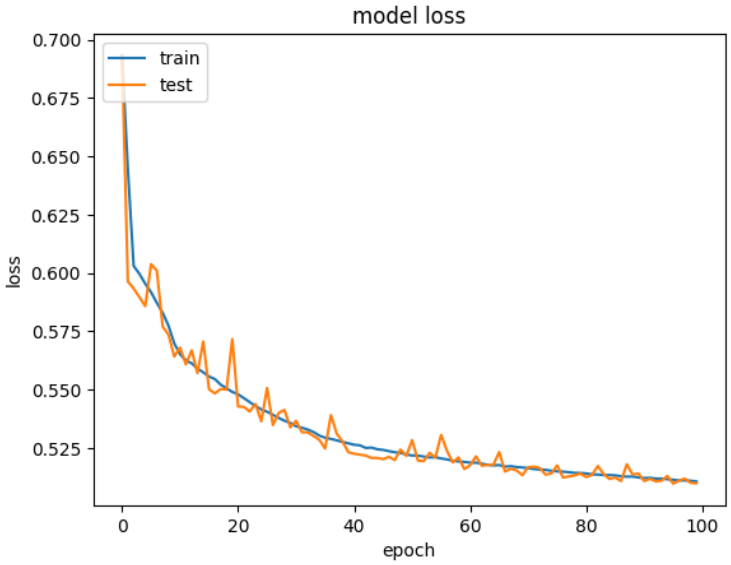



**Figure 47: Training vs Validation Loss**

Pion efficiency at 90% electron efficiency was calculated by writing a function which, when minimized, finds the cut-off point (critical region) in the distribution of the single node output layer which results in 90% of true electrons being classified as electrons, on a per-tracklet basis. This was done for tracks which left signal in all 6 layers of the TRD, allowing one to construct a test statistic taking all 6 estimates for the track into account, as outlined in 5.3 Statistical Tests for Particle Selection.

The result of the minimization process is shown in Figure 48: 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.

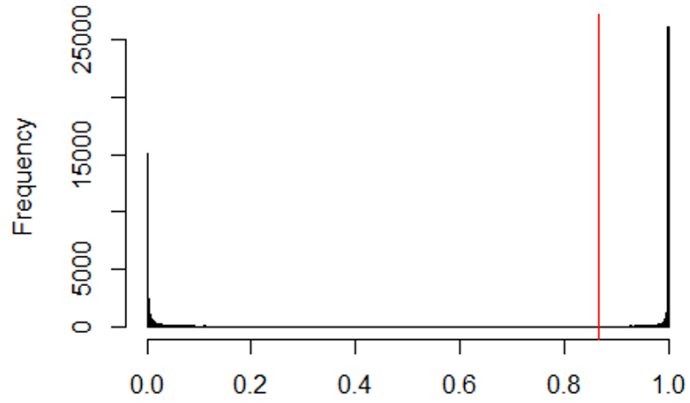



**Figure 48: t-statistic selection allowing for 90% electron efficiency**

After applying the classification based on $t_{cut}$, the histograms for the single node output layer's output for both electrons and pions is shown below:

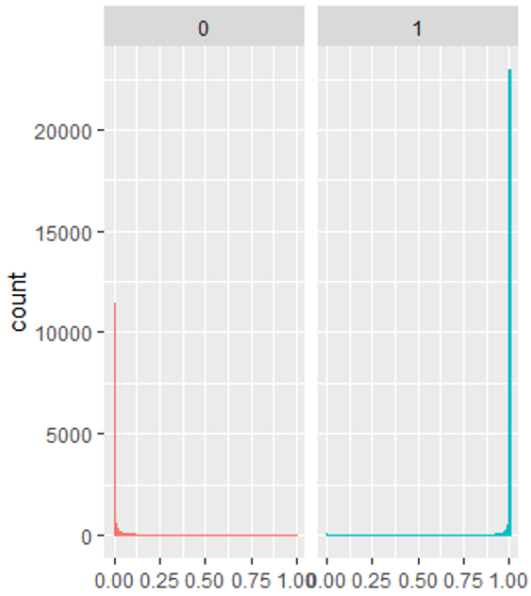



**Figure 49: Combined output probabilities for true electrons (1, blue) and true pions (0, red)**

Table 2 shows the obtained confusion matrix, with predicted labels along the rows and actual labels along the columns, here the diagonal of the confusion matrix are particles that were classified correctly:

**Table 2: Confusion Matrix for Particle Identification**

| Prediction/Actual | 0 | 1 |
|---|---|---|
| **0** | 47 833 | 4 893 |
| **1** | 1 088 | 44 028 |

Whilst they may not be very interpretable, the weights for the four convolutional layers of this model are plotted in Figure 50, Figure 51, Figure 52 and Figure 53 below.

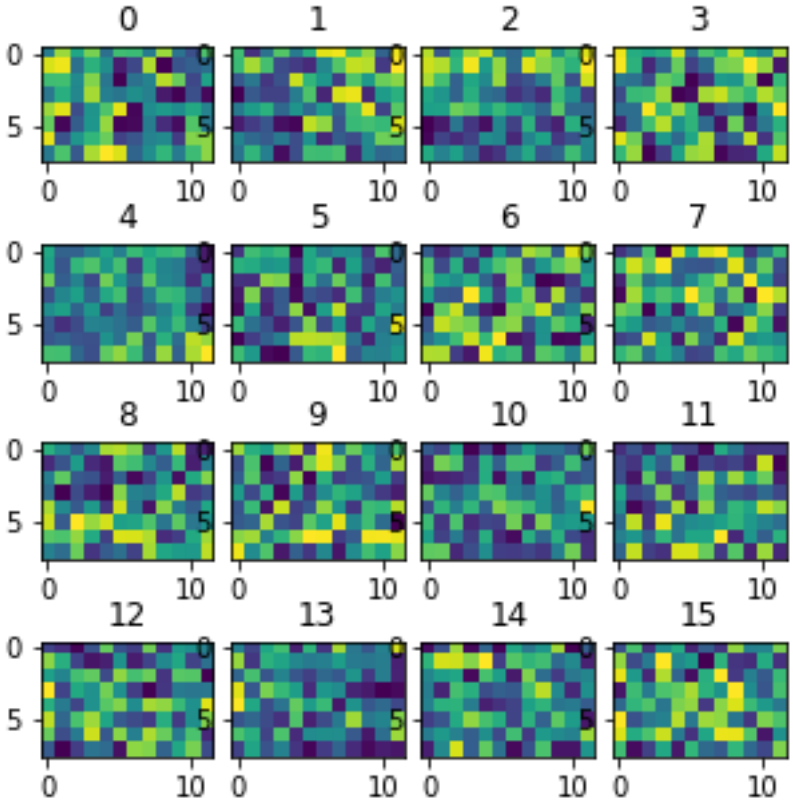



**Figure 50: Weights of first convolutional layer**

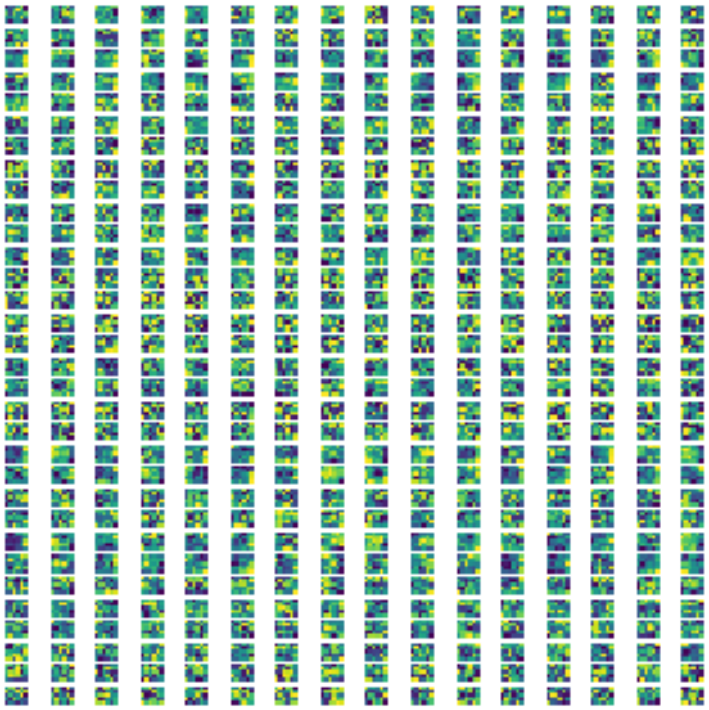


**Figure 51: Weights of second convolutional layer**

**Figure 52: Weights of third convolutional layer**

**Figure 53: Weights of fourth convolutional layer**

## 8.2 Distinguishing Geant Simulations from Real Data

Distinguishing Geant simulations from real data proved to be a much easier task than distinguishing real electrons from real pions, as depicted in the training graphs in Figure 59.

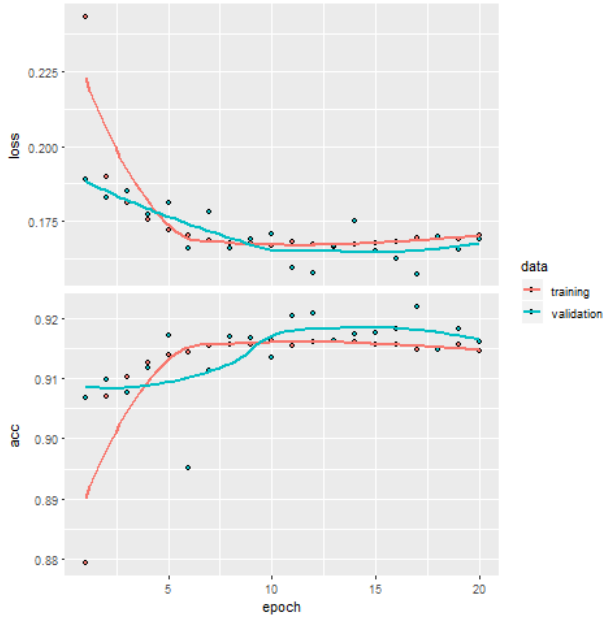



**Figure 54: Training loss and accuracy curves for training and validation data**

Table 3 shows the obtained confusion matrix for the following model architecture:

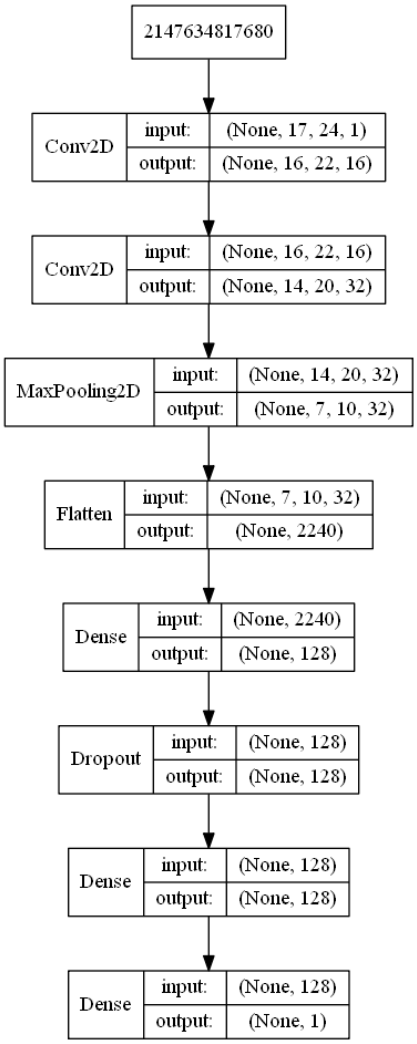



**Figure 55: Model architecture for distinguishing real from Geant simulated data**

**Table 3: Confusion Matrix for distinguishing between Geant vs Real Data**

| Prediction/Actual | 0 | 1 |
|---|---|---|
| **0** | 42 553 | 681 |
| **1** | 7 069 | 24 058 |

## 8.3 Deep Generative Models Towards Event Simulation

### 8.3.1 Variational Autoencoders

Although various VAEs were prototyped, the following architecture produced the best results:

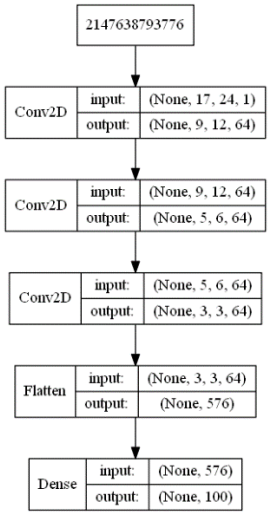



**Figure 56: Encoder**

Encoder returns the $\mu$ 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 $\varepsilon$ with $e^{\Sigma}$ and adding $\mu$, i.e.

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

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

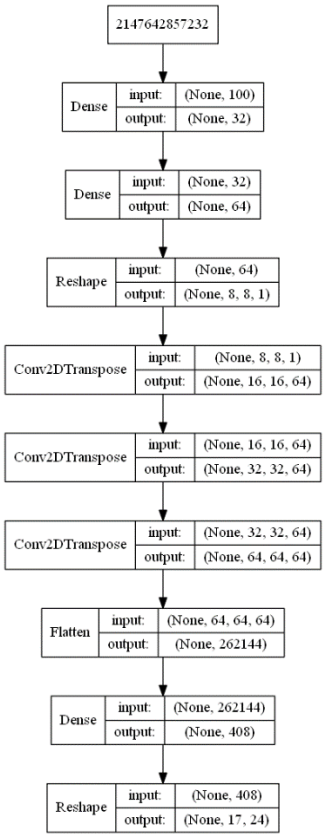



**Figure 57: Decoder**

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

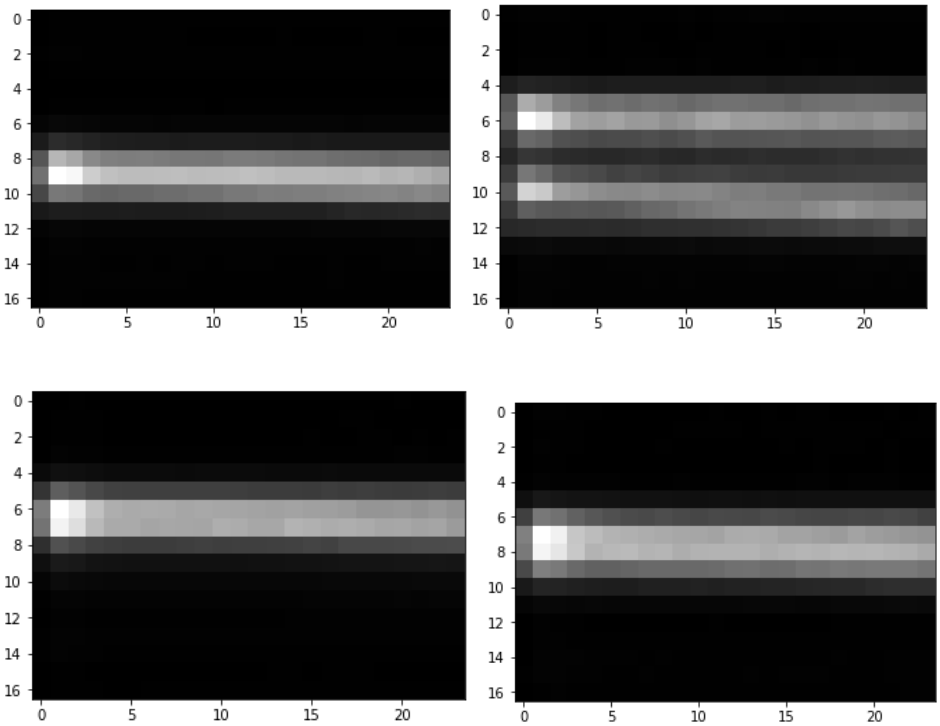


**Figure 58: Four examples of simulated data created using a Variational Autoencoder**

*Deep Learning Towards Distinguishing Variational Autoencoded Data from Real Data*

While these results look quite believable at first glance, it was quite easy to distinguish 100 000 real data samples vs 100 000 samples simulated with VAE using a CNN to 100% accuracy, as can be seen in Figure 59.

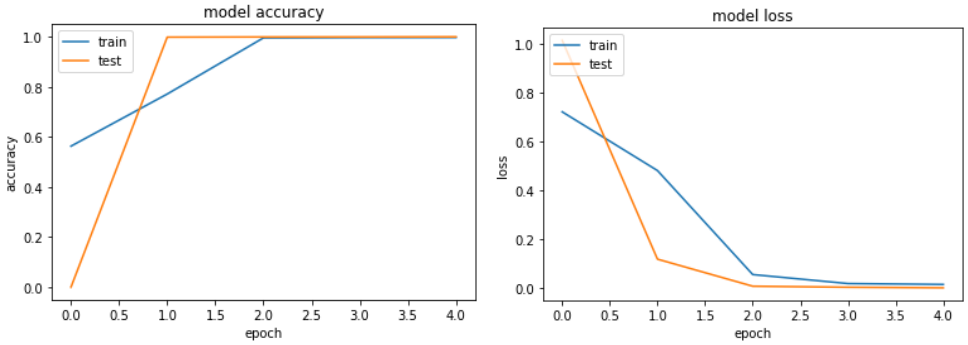



**Figure 59: Training accuracy and loss curves for training vs validation data**

## 8.3.2 Generative Adversarial Networks

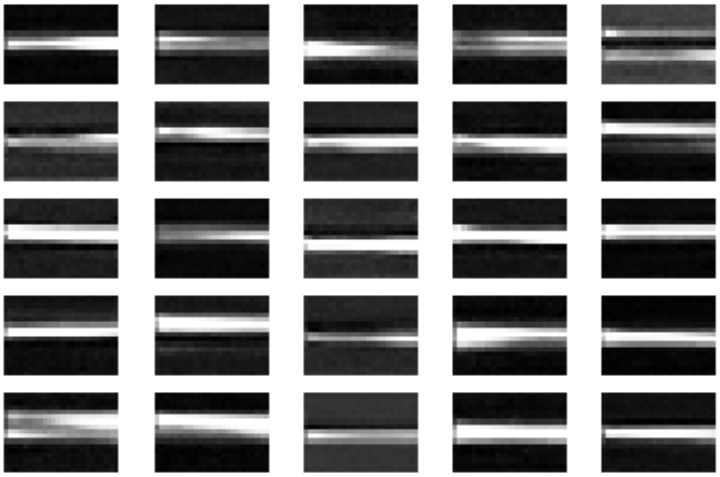
Below is a quick summary of the various Deep Generative Models used under the broader GAN category, along with how adjusting certain parameters lead to different results, illustrated by the accompanying images.

*Adversarial Autoencoder*

*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

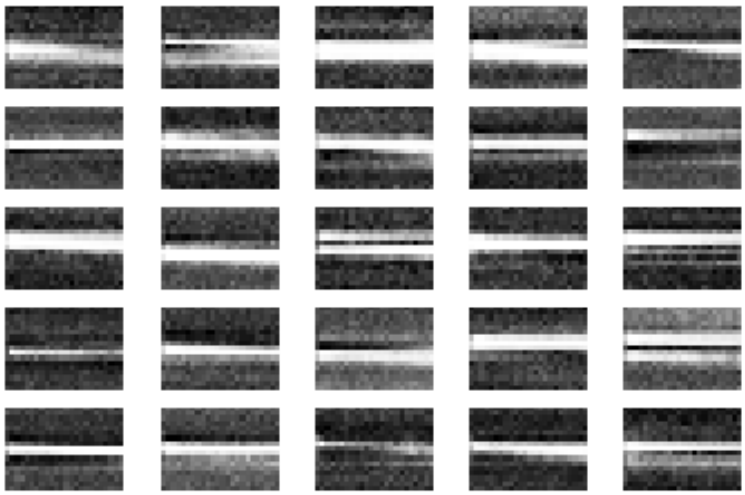
Sample result after 19800 epochs:



*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

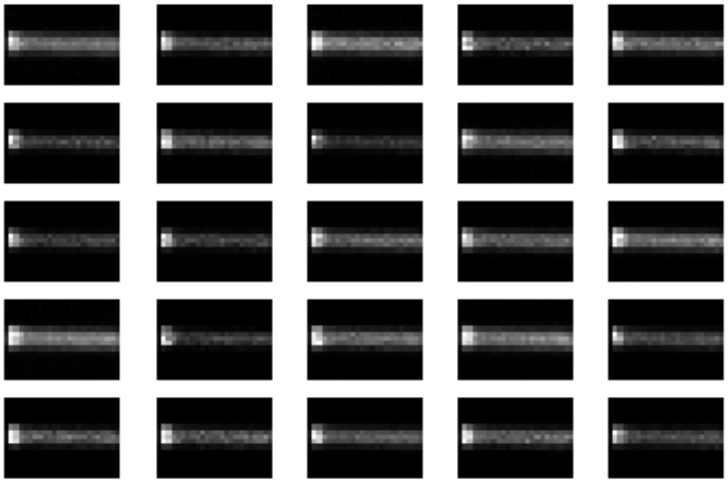
Sample result after 30800 epochs:



*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

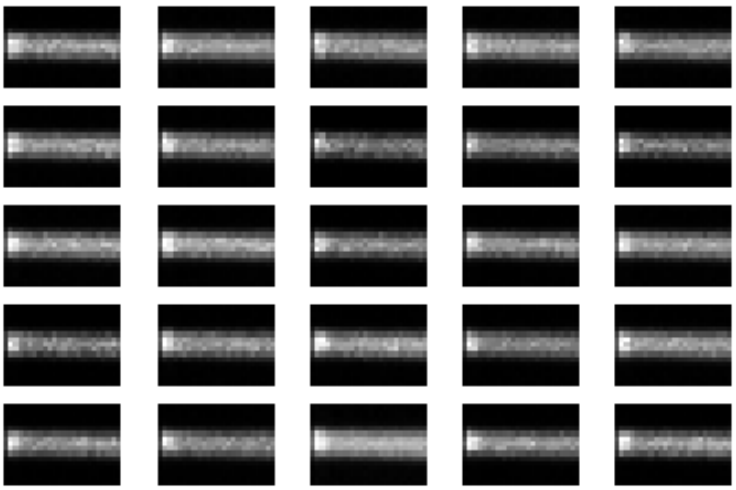
Sample result after 23600 epochs:



*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

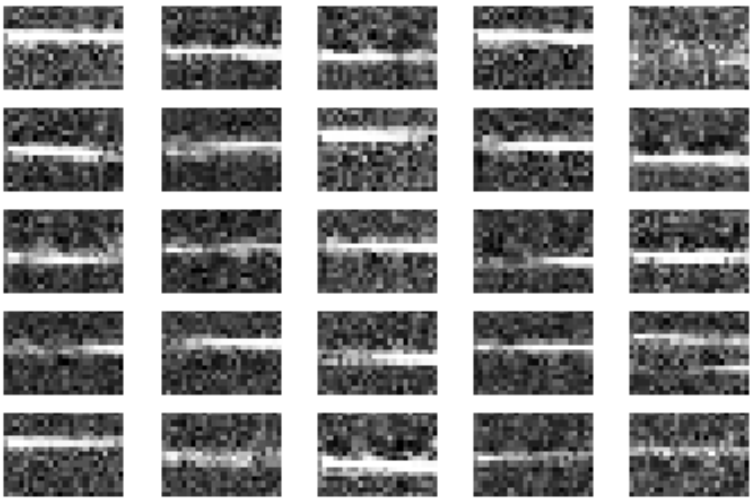
Sample result after 73200 epochs:



*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

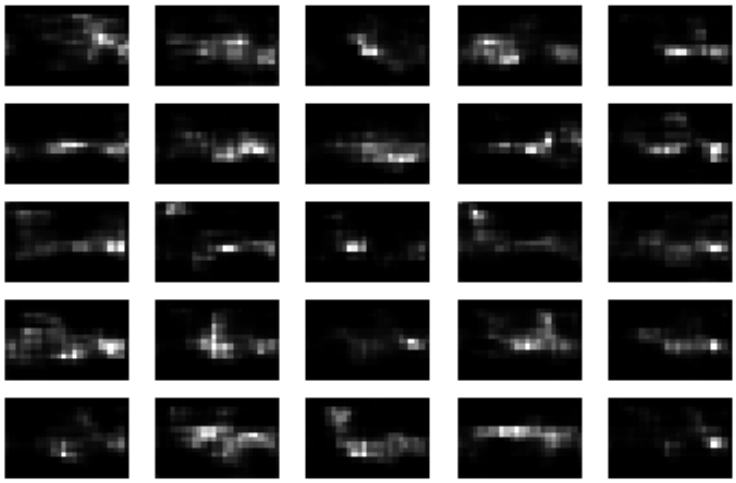
Example output after 39600 epochs:



*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.

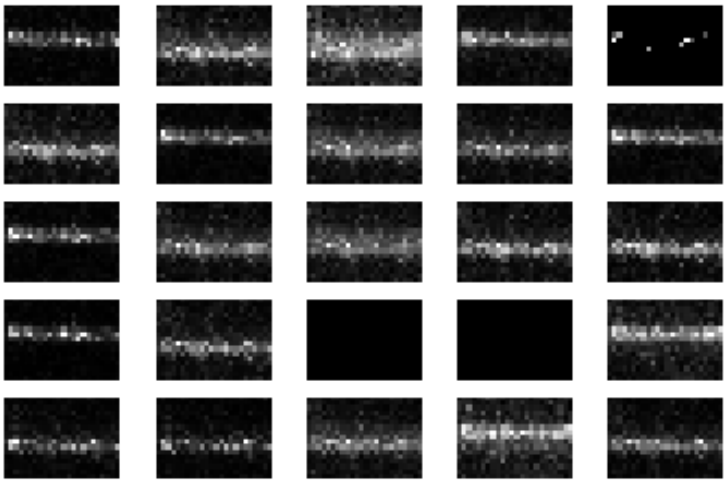
Example output after 57500 epochs:



*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.

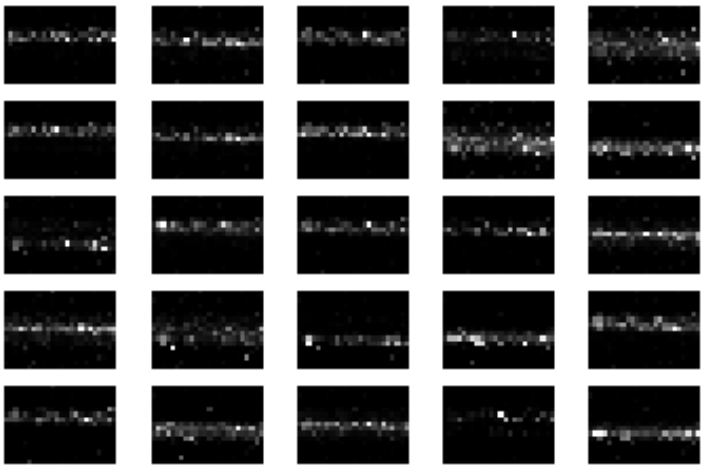
Example Output after 66400 epochs:



*Least Squares Generative Adversarial Network*

100 Latent dimensions
Adam Optimizer with learning rate = 0.00002 and beta1 = 0.5 using batch size = 32
Generator with 6 hidden dense layers with 256, 256, 512, 512, 1024 and 1024 hidden layers using leaky ReLU activation and an output layer with tanh activation, using Batch Normalization with momentum = 0.8 after each hidden layer
Discriminator with 8 hidden layers with 1024, 1024, 512, 512, 256, 256, 128 and 128 hidden layer and a single node output layer with no activation function

Example output after 64600 epochs:



# 9 DISCUSSION AND CONCLUSIONS

# 9.1 Discussion

## 9.1.1 Particle Identification Using Deep Learning

Inherent Limit on the Amount of Information Contained about the Class Label

One of the most salient features of the data used in this project is the inherent limit of the amount of information that the input features $x$ contain about the target feature $y$.

There seems to be an absolute limit at around 75% training accuracy, regardless of:

- which architecture was used (Figure 60, Figure 61 and Figure 62 show how this problem is potentially solvable by 2D Convolutional, LSTM and 1D Convolutional Neural Networks)

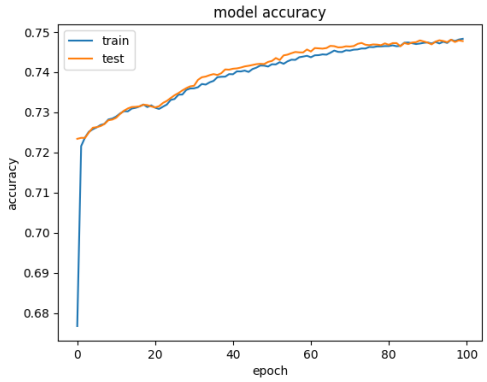



**Figure 60: Example of a 2D-Convolutional Network training to high validation accuracy**

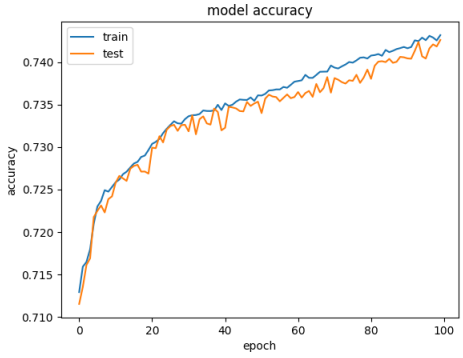



**Figure 61: Example of an LSTM Network training to high validation accuracy**

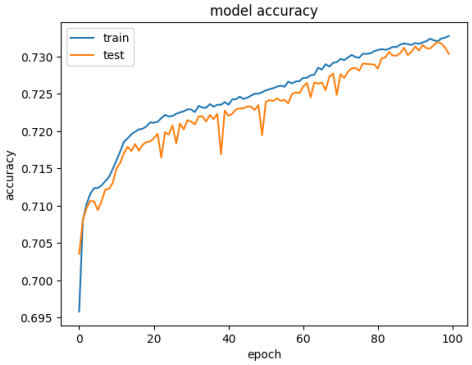



**Figure 62: Example of a 1D-Convolutional Neural Network training to high validation accuracy**

- the amount of epochs used for training (Figure 63 shows how training a highly successful model for twice the number of epochs only results in eventual overfitting)

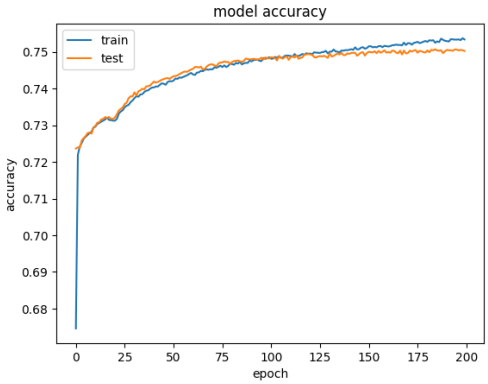



**Figure 63: Running a successful model for twice the number of epochs results in minimal gains and eventually, results in overfitting**

Convolutional Neural Networks

Regardless of the limitations outlined above, 2D convolutional neural networks resulted in the highest performance in general, but interestingly, even a very simplistic convolutional neural network (one convolutional layer, using 16 convolutional filters with kernel size equal to image dimensions, i.e. $17 \times 24$ and a single dense layer with 12 nodes) still gave comparable results, i.e. $\varepsilon_\pi = 6.2\%$.

Adding capacity to this simplistic architecture, by increasing from 1 to 4 dense hidden layers increased performance slightly, i.e. to $\varepsilon_\pi = 5.3\%$.

Using a single convolutional layer with smaller kernel size i.e. $3 \times 3$, and one dense layer did not give comparable results however and resulted in a much higher pion efficiency of $\varepsilon_\pi = 11.2\%$. Again, adding an additional dense layer improved results up to $\varepsilon_\pi = 8.5\%$, but when 3 hidden layers were used with Max Pooling after the second convolutional layer, performance dropped back down to $\varepsilon_\pi = 10\%$.

Conversely, using 3 convolutional layers with smaller kernel sizes, i.e. $3 \times 3$, $3 \times 3$ $and$ $2 \times 2$, and doubling the number of convolutional filters in each layer, i.e. 16, 32 and 64 filters in layer 1, 2 and 3 respectively, with Max Pooling layers after the first and second layers resulted in a pion efficiency of $\varepsilon_\pi = 5.5\%$.

Using a 2D Transpose Convolutional Layer with 16 filters having kernel size = $3 \times 5$ as the first layer, instead of a normal 2D Convolutional layer with the same configuration increased the pion efficiency of one model from $\varepsilon_\pi = 4.5\%$ to $\varepsilon_\pi = 3.8\%$, but no further gains were found when adding capacity by increasing the number of hidden layers, or when changing the kernel size of the 2D Transpose Convolutional Layer.

### 1D Convolutional Neural Networks

While 2D Convolutional Neural Networks were more successful than 1D Convolutional Neural Networks, 1D CNNs resulted in similar performance, even though they were fed data with lower dimensions, i.e. a compressed version of the data 2D CNNs were trained on, in the form of column sums of the original 2D image data. Using 4 1D Convolutional Layers (the first 3 of which were followed by Max Pooling operations), and 8 dense layers resulted in a pion efficiency of $\varepsilon_\pi = 6.5\%$. It should be noted that many more 2D CNNs were built during this project, but the fact that 1D CNNs gave comparable accuracy provides additional support to the hypothesis that there is a limit to the amount of information contained in the 2D images from the TRD about the Particle ID.

### LSTM Networks

The nature of this dataset is such that it can be framed as an image for Convolutional Neural Networks, but it is essentially a timeseries as well, with columns going across indicating the ADC signal at sequential time intervals and rows indicating where the charge was deposited (in which pad in a specific TRD chamber, row and column).

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. Using 6 LSTM layers, alternating between going backwards and forwards, with four dense layers of 256 nodes each made pion efficiency drop back down to $\varepsilon_\pi = 9.2\%$.

### The Effect of Regularization

Figure 64 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 network.

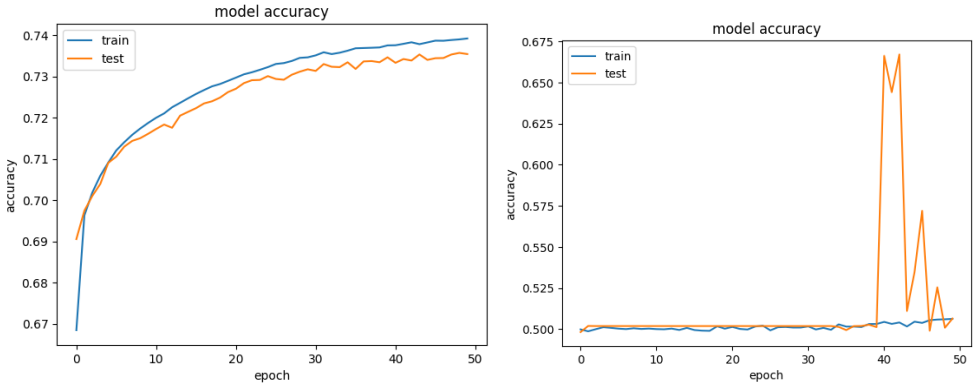



**Figure 64: No Dropout vs Same Model with too much Dropout**

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, e.g. a model employing the abovementioned Dropout rate which resulted in a pion efficiency of $\varepsilon_\pi = 6.5\%$ dropped down to a pion efficiency of $\varepsilon_\pi = 24.1\%$ when the learning rate (using the Adam optimizer) was reduced from $\lambda = 10^{-4}$ to $\lambda = 10^{-6}$.

The use of a Gaussian Noise layer as the first layer of various convolutional architectures was employed, but was unsuccessful, as depicted in Figure 65. 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.

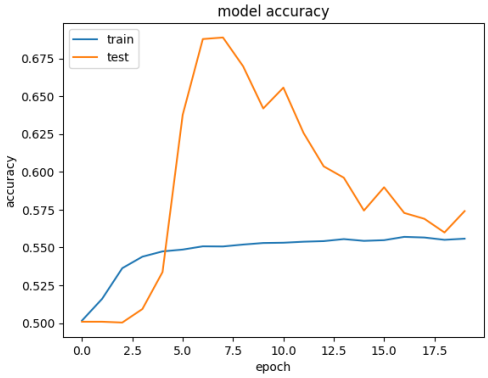



**Figure 65: Gaussian Noise with σ=0.2**

### 9.1.2 Distinguishing Simulated from Real Data

Distinguishing Geant simulated data from real data proved to be trivial compared to distinguishing real pions from real electrons. What follows is an analysis of features between the two datasets to investigate what 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 the same, i.e. $max_{sim} = max_{real} = 1023$.

Figure 66 and Figure 67 show the distribution of ADC values for simulated and real data, respectively.

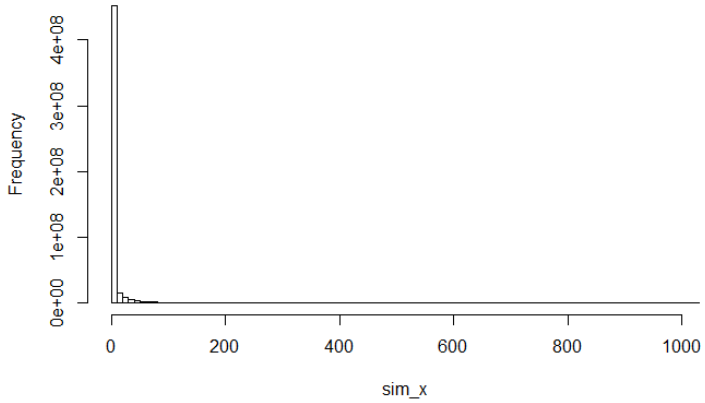



**Figure 66: Distribution of ADC values for simulated data**

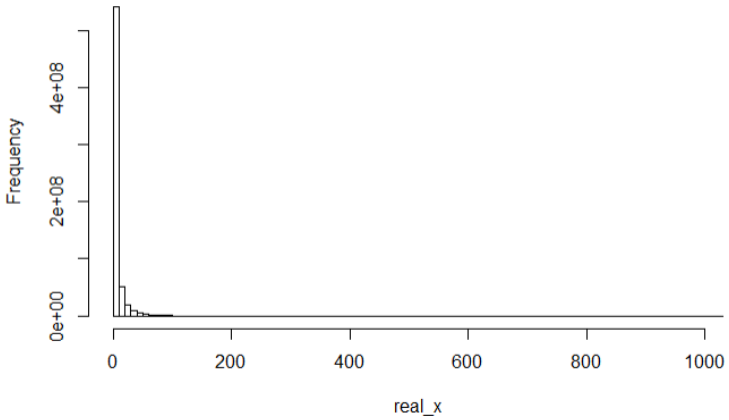



**Figure 67: Distribution of ADC values for real data**

Looking at the number of pads that don't contain any data (i.e. the number of rowsums of the image that are equal to 0), one sees the following distributions for simulated and real data:

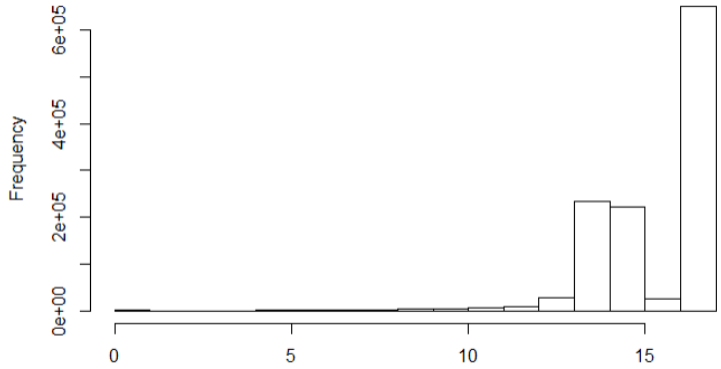



**Figure 68: Distribution of the number of pads with no data per image for simulated data**

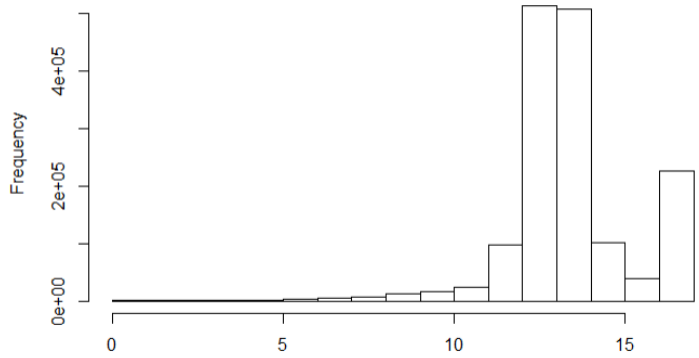



**Figure 69: Distribution of the number of pads with no data per image for real data**

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. 17 pads with no signal.

Finally, there are also differences in the distribution of mean ADC values per pad for real and simulated data (see Figure 70 and Figure 71).

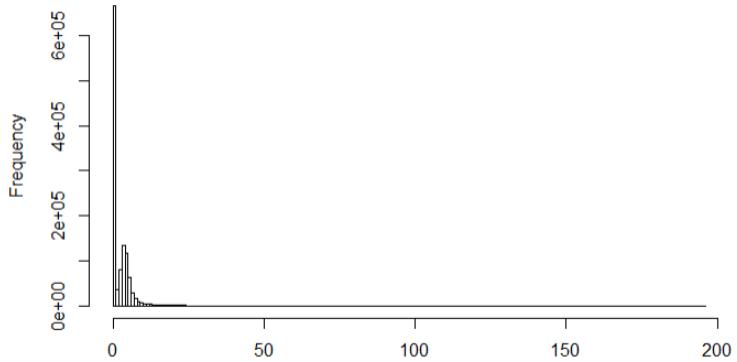



**Figure 70: Distribution of mean ADC value per image for simulated data**

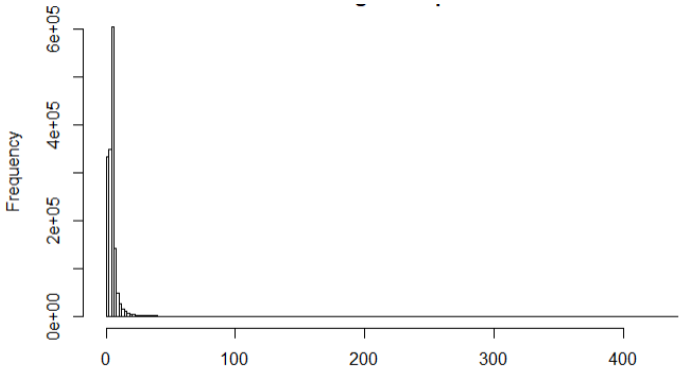



**Figure 71: Distribution of mean ADC value per image for real data**

### 9.1.3 Deep Generative Models towards High Energy Physics Event Simulations

While various deep generative architectures can give results that appear vastly different in terms of "style", they all fail 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.

While Variational Autoencoders give images that appear less spread-out along the pad dimension than any of the GAN architectures that were used, with less overall noise, they still fail to capture some of the volatility along the time dimension and appear somewhat "smeared out" compared to real data.

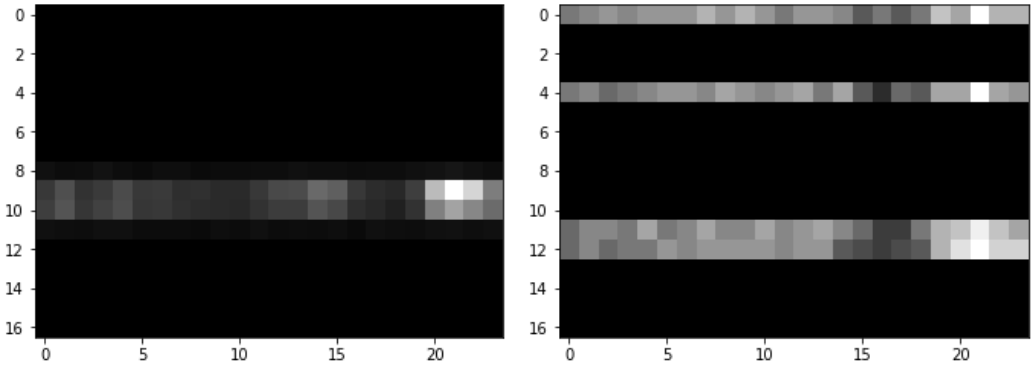



**Figure 72: Real images**

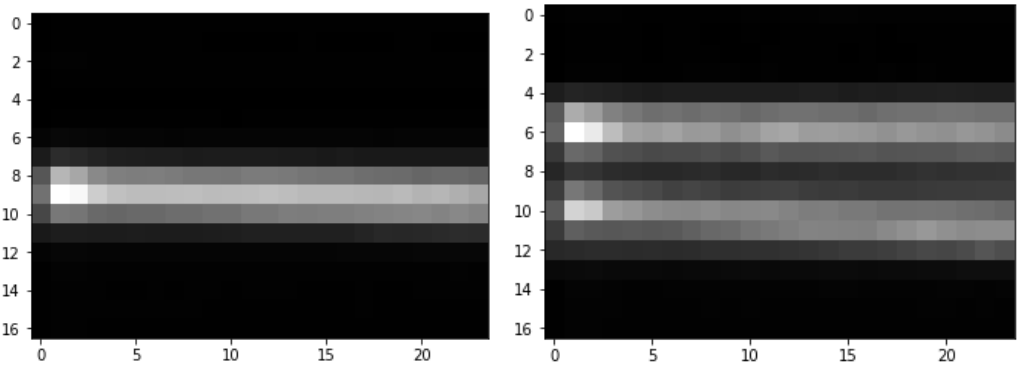



**Figure 73: Autoencoder outputs**

While Adversarial Autoencoders gave results that were slightly better than the other GAN architectures experimented with, all of the GAN architectures 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.

Examples of this can be seen in an Adversarial Autoencoder setup with very deep architecture in both the Generator and Discriminator (Figure 74), where after 21400 epochs, the Generator still outputs images that are seemingly random, but nonetheless have similar features across images (for some reason this resulted in low error and therefore became prevalent).

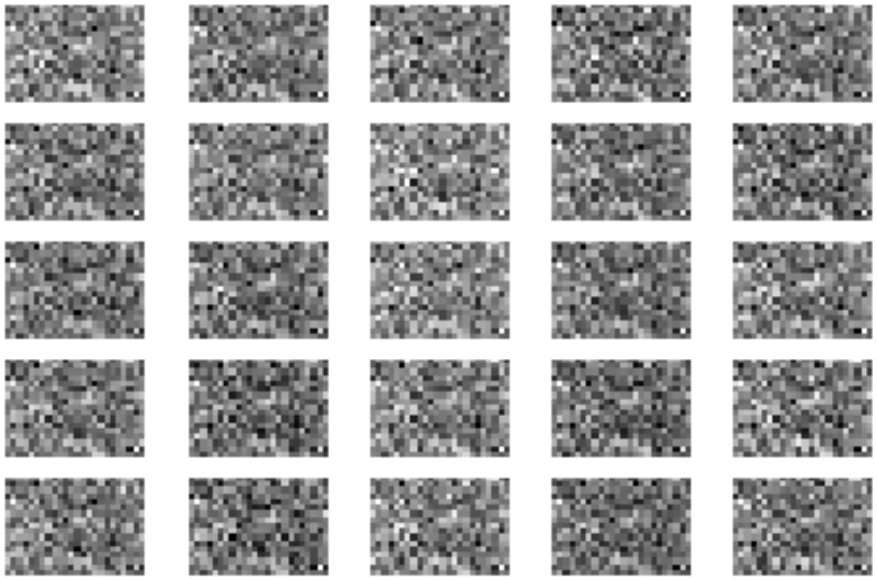


**Figure 74: An illustration of how the Generative network of a GAN can begin producing images with similar features that, even though they are not correct, result in low loss, because of a Discriminator which is not conducive to Generator training.**

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 75, one can see that the output images of the Bidirectional GAN (which employs this strategy) are highly dissimilar, but there is still a lot of noise around the main signal.

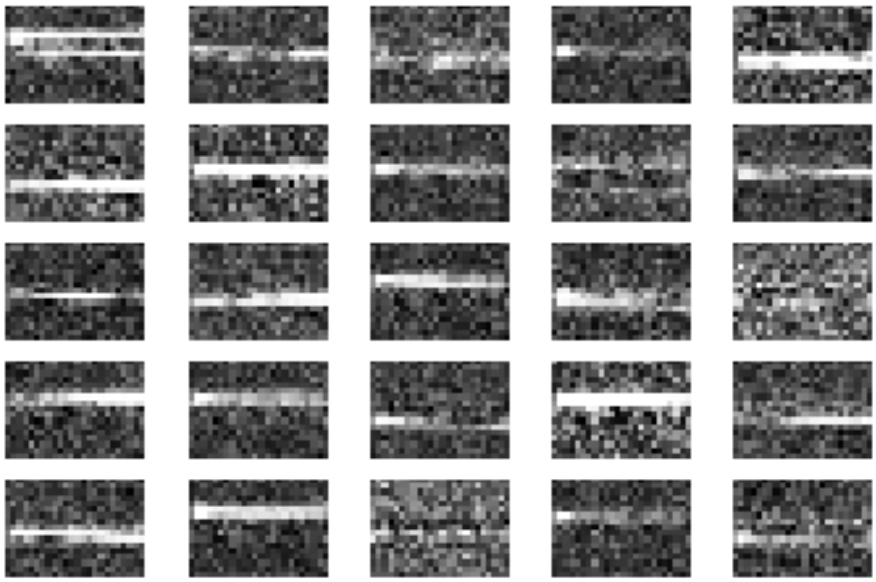


**Figure 75: Illustrating the effect of Batch normalization to prevent output images from looking highly similar**

It is interesting to note that while one might expect GANs that employ convolutional layers to give better results than fully connected dense layers, the Deep Convolutional GANs gave some of the most unrealistic simulated images (Figure 76). This could be due to the fact that convolutional layers introduce a prior that 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.

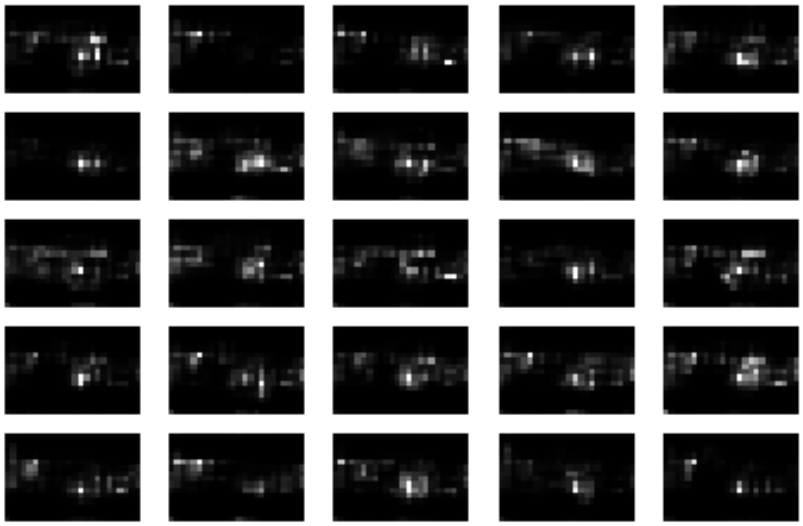


**Figure 76: Illustrating how convolutional architectures in a GAN setup results in features that might exist in the training distribution, but do not appear in the right place**

Training after a certain number of epochs does not necessarily result in additional gains in performance, as can be seen in Figure 77, Figure 78 and Figure 79, 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 look less realistic than those produced during earlier epochs.

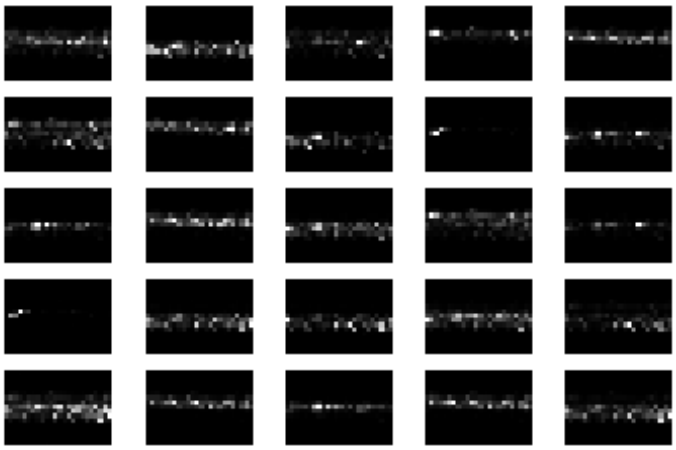


**Figure 77: Least squares GAN after 94600 epochs**

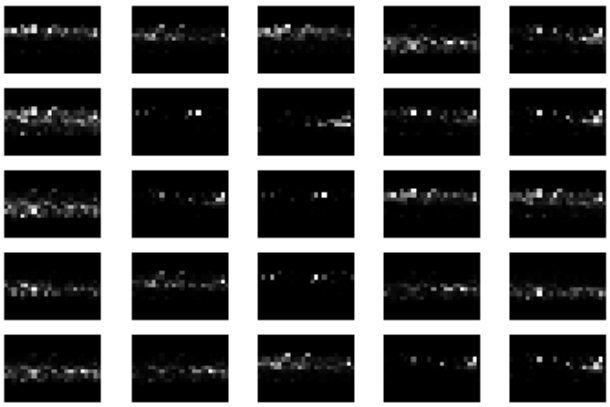


**Figure 78: Least squares GAN after 387600 epochs**

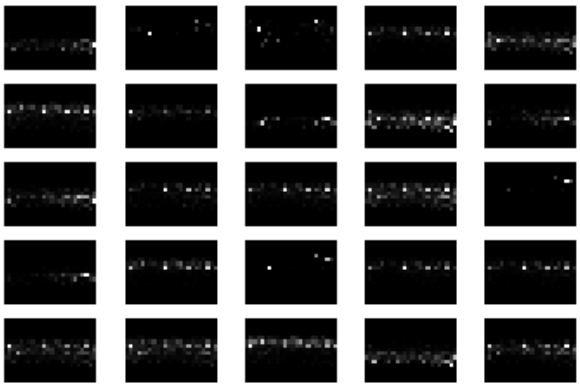


**Figure 79: Least squares GAN after 449000 epochs**

## 9.2 Conclusions

### 9.2.1 Particle Identification

While neural networks are very good at coming up with their own feature sets to find nested functions that can solve classification problems, what's more important in a deep learning project is making sure that the input data contains sufficient information about the class label.

While the work done in this project did not achieve comparable results to work done before, pad-per-pad calibration was performed on data in the work done before, which proved to be invaluable in normalizing the data for environmental and electronic variations which occur during data taking and affect how the signal manifests, a process which cannot be solvable by deep learning techniques.

### 9.2.2 Simulations

Probably the most important result of this dissertation is the fact that Geant simulations are easily distinguishable from real data. Whilst deep generative techniques do not appear to be able to give similar performance compared to Geant for this particular problem, there could be interesting ways to combine the two methods, e.g. by transforming the output given by Geant with deep generative techniques to make it appear more like real data.

## 9.3 Outlook and Future Work

### 9.3.1 Particle Identification

Future work should focus on applying additional data pre-processing steps before particle identification is applied. The data used for particle identification in this thesis was raw data from the TRD, whereas previous work used data which was calibrated pad-by-pad per run; however there does not seem to be much promise for arriving at increased accuracy using advanced deep learning methods compared to work that was done before

### 9.3.2 Simulations

As mentioned in the Conclusions section above, there should be scope to use the output of Geant simulations as a latent space to produce more realistic simulations, alternatively, a more realistic approach would entail tuning parameters used during simulation in the following script https://github.com/alisw/AliRoot/blob/master/TRD/TRDbase/AliTRDSimParam.cxx, a lot of hard-coded parameters could be adjusted recursively by using the loss function of a neural network which is used to distinguish between the two data sets as a parameter to be maximized, i.e. to produce simulated data that becomes harder to distinguish from real data as the multidimensional input parameters maximize the loss function of the discriminating neural network more, a setup similar to what occurs in GANs.

### 9.3.3 Outlook

Tensorflow and the Keras library are immensely useful for rapid prototyping of various deep learning architectures, whether they be for classification or regression problems. Whilst there exists a toolkit for machine learning within the AliRoot infrastructure, there should be benefits to exploring how using more modern deep learning libraries within the environment of AliRoot could extend its capabilities as Machine Learning becomes a more mature and effective field.

# 10 BIBLIOGRAPHY

1. *Modern Particle Physics.* **Thomson, Mark.** Cambridge, UK : Cambridge University Press, 2013. ISBN 978-1-107-03426-6.
2. ***Wikimedia Commons.* [Online] [Cited: 2 March 2019.] https://commons.wikimedia.org/wiki/File:Standard_Model_Feynman_Diagram_Vertices.png.**
**3. Particle Data Group. *The Review of Particle Physics.* 2018.**
**4. ALICE Collaboration. *The ALICE Transition Radiation Detector: construction, operation, and performance.* s.l. : CERN, 2017. arXiv:1709.02743v2.**
**5. *Connecting QGP-Heavy Ion Physics to the Early Universe.* Rafelski, Johann. 2013. Nuclear Physics B Proceedings Supplement.**
**6. *The Quark-Gluon Plasma∗ A Short Introduction.* Satz, Helmut. 2011. 6th International Conference on Physics and Astrophysics of Quark Gluon Plasma. https://arxiv.org/abs/1101.3937v1.**
**7. Viljoen, Christiaan Gerhardus. *Draw.io.* [Online] https://www.draw.io/?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1#G1X-ZGzxO_b4zo_74rY9Z9zaikz-Il6R8o.**
**8. QCD Phase Diagram SVG. *Wikimedia Commons.* [Online] [Cited: 18 2 2019.] https://commons.wikimedia.org/wiki/File:QCDphasediagram.svg.**
**9. Week 3: Thermal History of the Universe. *Caltech University.* [Online] [Cited: 20 February 2019.] www.astro.caltech.edu/~george/ay127/kamionkowski-earlyuniverse-notes.pdf.**
**10. The Steven Hawking Center for Theoretical Cosmology. [Online] http://www.ctc.cam.ac.uk/images/contentpics/outreach/cp_universe_chronology_large.jpg.**
**11. CERN. About CERN: Who we are: Our History. *CERN.* [Online] CERN. [Cited: 26 January 2019.] https://home.cern/about/who-we-are/our-history.**
**12. —. CERN: Who We Are: Our Governance: Member States. *CERN.* [Online] [Cited: 26 January 2019.] https://home.cern/about/who-we-are/our-governance/member-states.**
**13. —. CERN: About: Who We Are: Our Mission. *CERN.* [Online] [Cited: 26 January 2019.] https://home.cern/about/who-we-are/our-mission.**
**14. —. Grafana IT Overview. *CERN.* [Online] [Cited: 26 January 2019.] http://monit-grafana-open.cern.ch/d/000000884/it-overview?orgId=16.**
**15. Chardley, Sarah. LHC Does a Dry-Run. *Symmetry Magazine.* [Online] 20 March 2015. [Cited: 26 January 2019.] https://www.symmetrymagazine.org/article/march-2015/the-lhc-does-a-dry-run.**
**16. CERN. CERN Resources: FAQs: Facts and Figures About the LHC. *CERN.* [Online] [Cited: 06 01 2019.] https://home.cern/resources/faqs/facts-and-figures-about-lhc.**
**17. University of California Davis. RHIC. *UC Davis Nuclear.* [Online] [Cited: 27 01 2019.] http://nuclear.ucdavis.edu/~rpicha/rhic.html.**

**18. Encyclopedia Britannica. Tevatron Particle Accelerator.** ***Encyclopedia Britannica.*** **[Online] [Cited: 27 January 2019.] https://www.britannica.com/technology/Tevatron.**
**19.** ***CIRCUMFERENCE VARIATIONS OBSERVED AT KEKB.*** **Masuzawa, M, et al. 2002. Proceedings of the 7th International Workshop on Accelerator Alignment. Vols. Spring-8.**
**20. Kurokawa, Shin-Ichi and Olsen, Stephen L. The KEK B-Factory Experiment.** ***Stanford University.*** **[Online] [Cited: 27 January 2019.] www.slac.stanford.edu/pubs/beamline/29/2/29-2-kurokawa.pdf.**
**21. Taking a Closer Look at the LHC. The LHC Proton Source.** ***LHC Closer.*** **[Online] [Cited: 27 January 2019.] https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.proton_source.**
**22. CERN. LHC: The Guide.** ***CERN.*** **[Online] [Cited: 2017 January 2019.] https://home.cern/resources/brochure/cern/lhc-guide.**
**23. Field, Rick. PHY2061 - Enriched Physics 2 - Relativity 4.** ***University of Florida Physics.*** **[Online] [Cited: 27 January 2019.] http://www.phys.ufl.edu/~acosta/phy2061/lectures/Relativity4.pdf.**
**24. CERN. The CERN Accelerator Complex.** ***CERN Document Server.*** **[Online] [Cited: 26 January 2019.] https://cds.cern.ch/record/2636343/files/CCC-v2018-print-v2.jpg?subformat=icon-1440.**
**25. —. LHC Experiments.** ***CERN.*** **[Online] [Cited: 21 February 2019.] https://home.cern/science/experiments.**
**26. —. ATLAS Experiment.** ***CERN.*** **[Online] [Cited: 21 February 2019.] https://home.cern/science/experiments/atlas.**
**27. —. ALICE Experiment.** ***CERN.*** **[Online] [Cited: 21 Fenruary 2019.] https://home.cern/science/experiments/alice.**
**28. —. LHCb Experiment.** ***CERN.*** **[Online] [Cited: 21 February 2019.] https://home.cern/science/experiments/lhcb.**
**29. —. LHCf Experiment.** ***CERN.*** **[Online] [Cited: 21 Febraury 2019.] https://home.cern/science/experiments/lhcf.**
**30. —. MoEDAL Experiment.** ***CERN.*** **[Online] [Cited: 21 Febrary 2019.] https://home.cern/science/experiments/moedal.**
**31. —. ROOT Data Analysis Framework: User's Guide.** ***CERN.*** **[Online] May 2018. https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideA4.pdf .**
**32. [Online] https://root.cern.ch/.**
**33. ROOT 5 Reference Guide. [Online] https://root.cern/root/html534/ClassIndex.html.**
**34. ROOT 6 Reference Guide. [Online] https://root.cern/doc/v616/.**
**35. ALICE Collaboration (CERN).** ***ALICE Analysis Tutorial.*** **[Online] [Cited: 18 2 2019.] https://alice-doc.github.io/alice-analysis-tutorial.**
**36. CERN. High Energy Physics Simulations.** ***LHC @ home.*** **[Online] [Cited: 26 July 2019.] http://lhcathome.web.cern.ch/projects/test4theory/high-energy-physics-simulations.**
**37.** ***Geant4 - a simulation toolkit.*** **Agostinelli, Stefano, et al. 2003, Nuclear Instruments and Methods in Physics Research, Vol. A 506, pp. 250-303.**
**38. ALICE. ALICE Homepage.** ***ALICE.*** **[Online] [Cited: 21 February 2019.] http://alice.web.cern.ch/.**
**39. CERN.** ***CERN Document Server.*** **[Online] [Cited: 21 February 2019.] https://cds.cern.ch/record/2302924.**

**40. The ALICE Collaboration.** ***The ALICE Experiment at the CERN LHC.*** **s.l. : INSTITUTE OF PHYSICS PUBLISHING AND SISSA, 2008. 2008 JINST 3 S08002.**
**41. —.** ***The Technical Design Report of the Transition Radiation Detector.*** **Geneva : CERN, 2001. CERN/LHCC 2001-021 ALICE TDR 9.**
**42. CERN. Physics Vectors.** ***ROOT User's Guide.*** **[Online] [Cited: 23 February 2019.] https://root.cern.ch/root/htmldoc/guides/users-guide/PhysicsVectors.html.**
**43. Spherical Coordinates.** ***Maths Insight.*** **[Online] [Cited: 23 February 2019.] https://mathinsight.org/spherical_coordinates.**
**44.** ***Particle Identification with the ALICE Transition Radiation Detector.*** **Pachmayer, Yvonne. 2014. arXiv:1402.3508v1 [physics.ins-det].**
**45. Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron.** ***Deep Learning.*** **Cambridge, Masachusetts : The MIT Press, 2016. ISBN 9780262035613.**
**46.** ***The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.*** **Rosenblatt, F. 6, 1958, Psychological Review, Vol. 65.**
**47. keras.io. Available Activations.** ***Keras.*** **[Online] [Cited: 19 July 2019.] https://keras.io/activations/#available-activations.**
**48. Viljoen, CG.** ***Draw.io.*** **[Online] https://www.draw.io/?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1#G1Zbad00aGA6OXlYpDzl8ggBK2kRM08xzO.**
**49.** ***Long Short-Term Memory.*** **Hochreiter, Sepp and Schmidhuber, Jurgen. 8, 1997, Neural Computation, Vol. 9, pp. 1735-1780.**
**50. Doersch, Carl.** ***Tutorial on Variational Autoencoders.*** **Carnegie Mellon University. s.l. : ResearchGate, 2016.**
**51.** ***Generative Adversarial Nets.*** **Goodfellow, Ian, et al. s.l. : Curran Associates, Inc., 2014.**
**52.** ***Conditional Image Synthesis with Auxiliary Classifier GANs.*** **Odena, Augustus, Olah, Christopher and Shlens, Jonathon. 2017. Proceedings of the 34th International Conference on Machine Learning.**
**53.** ***Adversarial Autoencoders.*** **Makhzani, Alireza, et al. 2016. arXiv:1511.05644v2.**
**54.** ***Adversarial Feature Learning.*** **Donahue, Jeff, Darrell, Trevor and Krahenbuhl, Phillip. 2017. arXiv:1605.09782v7.**
**55.** ***Unsupervised Representation Learning with Deep Convolutional Genarative Adversarial Networks.*** **Metz, Luke, Radford, Alec and Chintala, Soumith. 2016. ICLR 2016. arXiv:1511.06434v2.**
**56.** ***Least Squares Generative Adversarial Networks.*** **Mao, Xudong, et al. 2017. arXiv:1611.04076v3.**
**57. Cowan, Glen.** ***Statistical Data Analysis.*** **Oxford : Oxford University Press, 1998. ISBN0198501560.**
**58. Weinberg, Steven.** ***The First Three Minutes.*** **Cambridge, Masachusettes : Fontana Paperbacks, 1976.**
**59. Robinson, Donald. [Online] [Cited: 23 February 2019.] http://donrmath.net/CalcIII/unit1/lesson7/u1l7.html.**
**60. NN SVG.** ***alexlenail.*** **[Online] [Cited: 21 April 2019.] http://alexlenail.me/NN-SVG/AlexNet.html.**