## Particle Identification: Results

Three sets of results will be presented:

1. The most successful particle identification strategy on uncalibrated raw digits will be discussed in detail in Section 4.4.1.
2. A summary of other models that were built and trained for particle identification will be presented, at the hand of (for all 2D Convolutional Neural Networks built),
3. and as a text summary in Section 4.4.2 (for *all* models built).



Figure 53: Training accuracy and loss curves for discriminating GAN- from real data

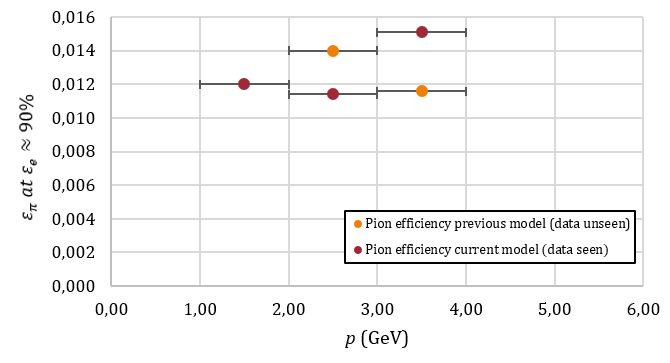


Figure 57: Training and loss curves for discriminating AAE- from real data



Figure 46: Accuracy and loss training curves for discriminating Geant4 from Real data

#### Setup of the most successful



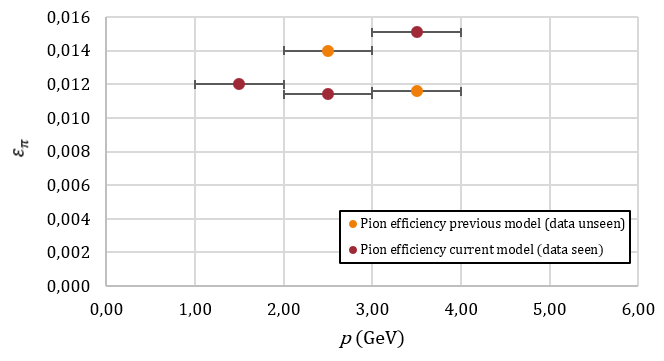




Figure 27: 2D Convolutional Networks are compared in terms of number of total layers (dense and convolutional combined, LHS) and number of convolutional layers (RHS). Learning rate is shown on a colour gradient as indicated by the legend. Number of epochs trained (mark size) and optimization algorithm (mark shape) are also indicated. Pion efficiency is plotted on the y-axis (logarithmic scale).

# Appendices

[Appendix A: Plotting the Bethe-Bloch Equation ii](#_Toc14519599)

[Appendix B: Plotting Binary Cross-Entropy 14](#_Toc14519600)

[Appendix C: The Anatomy Of An AliROOT Analysis Task 15](#_Toc14519601)

[Appendix D: Software Environment, Packages & Utilities 19](#_Toc14519602)

[Appendix E: Running and Monitoring Root Analysis Tasks 21](#_Toc14519603)

[Appendix F: Data Extraction, Data Quality and Data Pre-processing 25](#_Toc14519604)

[Appendix G: Old Methods Section 28](#_Toc14519605)

[Appendix H: Old Results Section 32](#_Toc14519606)

Various methods defined under the broader scope of machine learning were used to build classifiers to distinguish electron tracklet signals from pion tracklet signals produced during high energy physics (HEP) experiments. These tracklet signals, which were fed as input features to the abovementioned machine learning algorithms, manifested in this project as

As mentioned above in **Error! Reference source not found.**, 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





Figure 2: Six examples of simulated data created using a Generative Adversarial Network

Example images:





Figure 3: Six examples of simulated data created using an Adversarial Autoencoder

OLD ABSTRACT:

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

1. Particle identification; distinguishing 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 in the range, in the and in the range, all at electron efficiency of . The best results were obtained using an incrementally trained convolutional neural network, which was trained on data from particles in increasing momentum ranges sequentially.

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.

1. High Energy Physics Event Simulations, Part I: Distinguishing real data from data generated by Geant4

This stage of the project focused 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. ROOT was used to reconstruct the simulated data to deliver it in a similar format to that given by raw data after processing. A balanced accuracy score of 91.5% (with Sensitivity = 0.8575 and Specificity = 0.9725) was achieved, using a 2D Convolutional Neural Network.

The fact that Geant4 simulations were easily discriminated from real data motivated the third stage of this thesis.

1. High Energy Physics Event Simulations, Part II: Deep Generative Modeling

Various deep generative models were built to take as input raw TRD data and produce simulated observations which are likely under the training data distribution. Various strategies were employed towards deep generative modelling; with Adversarial Autoencoders giving the best results.

OLD INTRODUCTION:

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

* Particle identification (i.e. classification) of electrons and pions, from raw signal data produced by these particles as they traverse the Transition Radiation Detector (TRD), using various deep learning methods
* Optimizing High Energy Physics Event Simulations by:
  + Distinguishing between real data and data simulated by the Geant4 Monte Carlo simulation environment
  + Building deep generative models, namely Variational Autoencoders, Adversarial Autoencoders and Generative Adversarial Networks for the simulation of TRD data obtained during High Energy Physics (HEP) collision events and quantifying each model’s accuracy

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 (A Large Ion Collider Experiment) experiment to study the properties of the Quark Gluon Plasma (QGP), a primordial state of matter thought to have existed in the early universe. Since this deconfined state of matter rehadronizes quite soon after forming, it cannot be studied directly, but only via its decay products. To this end, accurate particle identification is extremely important
* 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 Variational Autoencoders (VAEs), Adversarial Autoencoders (AAEs) and Generative Adversarial Networks (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 practical 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-type, and at which momentum you want to be simulated?

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.

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

Quantum mechanics explains the emergence of unique physical properties in different elements, which arise from their exact electronic structures. Quantum field theories explain

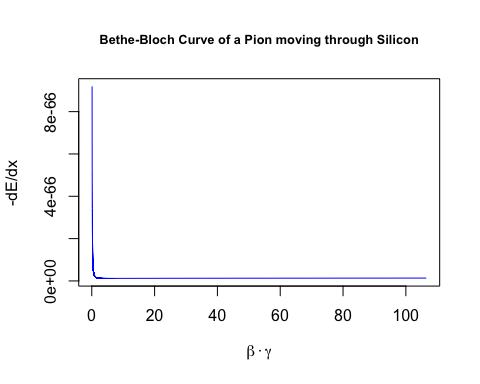
Appendix A: Plotting the Bethe-Bloch Equation

Create a Bethe-Bloch function:

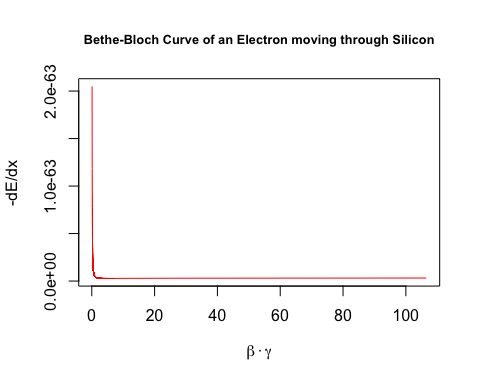
#Planck's constant:  
h <- 6.62607004e-34  
  
#Speed of light m/s  
c <- 299792458  
  
#Fine structure constant  
alpha <- 1/137  
  
#Mass of an electron Mass/GeV  
  
m.e <- 0.005  
  
#Density n, atomic number Z, the fraction of the speed of light the particle is moving at, beta, and the particle's velocity v are specified as parameters to the equation  
  
  
dE.dx <- function(n,Z,v,beta){  
 -4 \* pi \* h^2 \* c^2 \* alpha^2 \* ((n \* Z)/(m.e \* v^2)) \* log(((2 \* beta^2 \* gamma^2 \* c^2 \* m.e)/(I.e)) - beta^2,base=exp(1))  
}  
  
#For an electron traversing a silicon detector:  
  
v <- seq(0.1\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
  
  
Z <- 14  
  
#Effective ionization potential of the material  
  
I.e <- 10 \* Z  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
require(latex2exp)

## Loading required package: latex2exp

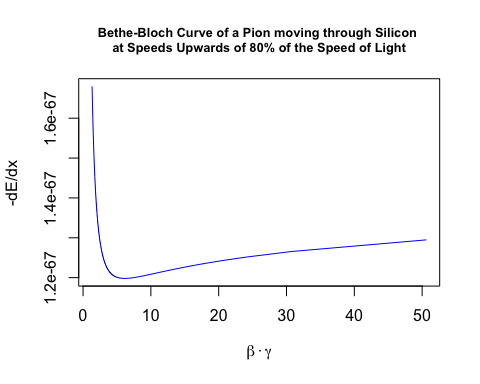
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



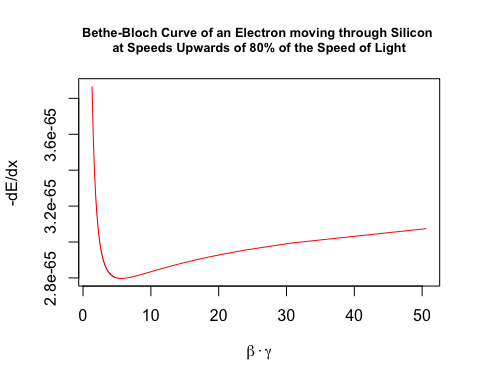
plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



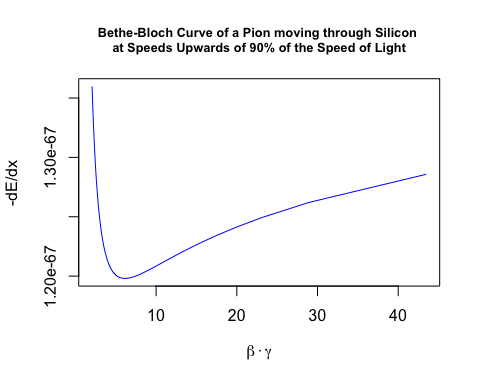
v <- seq(0.8\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
m.e <- 0.005  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon \nat Speeds Upwards of 80% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



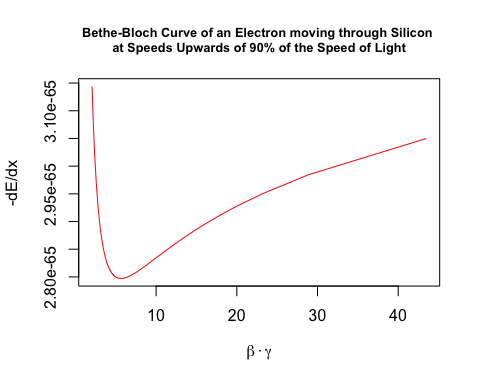
plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon \nat Speeds Upwards of 80% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



v <- seq(0.9\*c,c,100000)  
  
beta <- v/c  
  
#Lorentz factor  
  
gamma <- 1/(sqrt(1-(v^2/c^2)))  
  
n <- 1  
  
m.e <- 0.005  
  
electron.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
m.e <- 273.13\*m.e  
  
pion.y = dE.dx(n=n,Z=Z,v=v,beta=beta)  
  
plot(x=beta\*gamma, y=-pion.y,type="l",main="Bethe-Bloch Curve of a Pion moving through Silicon \nat Speeds Upwards of 90% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="blue",cex.main=0.8)



plot(x=beta\*gamma, y=-electron.y,type="l",main="Bethe-Bloch Curve of an Electron moving through Silicon \nat Speeds Upwards of 90% of the Speed of Light", xlab = TeX("$\\beta\\cdot\\gamma$"),ylab=TeX("$-dE/dx$"),col="red",cex.main=0.8)



Appendix B: Plotting Binary Cross-Entropy

Define a function to plot the binary cross-entropy loss function:

cross.entropy <- function(y,p){  
 -(y \* log(p,base = 10) + ((1-y)\*(1 - log(p,base=10))))  
}  
  
#if the predicted class is 1:  
  
y <- 1  
  
p <- seq(0,1,0.01)  
  
loss <- cross.entropy(y,p)  
  
require(latex2exp)

## Loading required package: latex2exp

plot(x=p,y=loss, type="b", col=rainbow(250),cex=0.5, main = TeX("J($\\theta$) = -(y log(p)-(1-log(p)))"),ylab = "Cross Entropy", xlab = TeX("$\\hat{y}$"))



Appendix C: The Anatomy Of An AliROOT Analysis Task

In AliROOT, all analysis tasks inherit from the base class **AliAnalysisTaskSE** (where SE stands for Single Event), which in turn is derived from the base class **AliAnalysisTask.**

All analysis tasks done in AliROOT inherit the following base methods from **AliAnalysisTaskSE**:

AliAnalysisTaskSE::AliAnalysisTaskSE();//constructor1

AliAnalysisTaskSE::AliAnalysisTaskSE(const char\*);//constructor2

AliAnalysisTaskSE::~AliAnalysisTaskSE();//destructor

AliAnalysisTaskSE::UserCreateOutputObjects();//user-defined output objects (results of physics analyses, which can be attached to output files)

AliAnalysisTaskSE::UserExec(Option\_t\*);//event loop, called for each event in the analysis: checks conditions for inclusion, accesses physics objects, fills histograms or other data containers with attributes from event

AliAnalysisTaskSE::Terminate(Option\_t\*); //deallocates memory after all steps in analysis have completed

The final element of an analysis task in AliROOT is the (.C) macro file, which creates and configures an instance of the particular C++ class.

##### The Class Header (.h)

Reproduced and modified from (35):

#ifndef AliAnalysisTaskMyTask\_H //include guard (aids in prevention of double inclusion, which may result from including parent and child classes, leading to multiple definitions for class members)

#define AliAnalysisTaskMyTask\_H //part of include guard

class AliAnalysisTaskMyTask : public AliAnalysisTaskSE //we define a class AliAnalysisTaskMyTask, which inherits from the base class AliAnalysisTaskSE

{

public:

// two class constructors, called when a new instance of the class is created

AliAnalysisTaskMyTask();

AliAnalysisTaskMyTask(const char \*name);

// class destructor, called when this instance of the class is deleted

virtual ~AliAnalysisTaskMyTask();

// called once at beginning of runtime

virtual void UserCreateOutputObjects();

// called for each event

virtual void UserExec(Option\_t\\* option);

// called at end of analysis

virtual void Terminate(Option\_t\\* option);

//class members

private:

AliAODEvent\* fAOD; //!<! pointer to a single input event

TList\* fOutputList; //!<! pointer to an output list, which holds all the output objects of the analysis

TH1F\* fHistPt; //!<! pointer to a histogram containing the transverse momentum (Pt) spectrum

//note that the !<! expression above is seen and evaluated by ROOT and is used in the generation of ROOT documentation

//ClassDef definition:

/// \cond CLASSDEF //surrounding comments for documentation generation

ClassDef(AliAnalysisTaskMyTask, 1); //this is a C pre-processor macro, used when class derives from TObject: it contains member declarations and inserts a few new members into the class, version number is incremented from 1 when definition of class changes

/// \endcond}; //surrounding comments for documentation generation

#endif //part of include guard

##### The Class Implementation (.cxx)

Reproduced and modified from (35):

//include statements for UserCreateOutputObjects:

**#include "TList.h"** *//TList class, an instance of which will contain a histogram in this example*

#include "TH1F.h" //ROOT 1-dimensional histogram class with one float per channel

//include statement for UserExec:

#include "AliAODEvent.h"

//implementation of class constructors:

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask() : AliAnalysisTaskSE(),

//members of the class are initialized in the constructors with their default values, if default values are not specified, these will be filled with random values, which could lead to unexpected behaviour

fAOD{0}, fOutputList{0}, fHistPt{0}

{

// This first constructor is the ROOT IO constructor, memory should not be allocated here

}

//in the second constructor, below, the input and output objects handled by the class are defined

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char\* name) : AliAnalysisTaskSE(name),

fAOD{0}, fOutputList{0}, fHistPt{0}

{

//input object is a TChain

DefineInput(0, TChain::Class());

//output object is a TList

DefineOutput(1, TList::Class());

}

//implementation of the UserCreateOutputObjects class:

AliAnalysisTaskMyTask::UserCreateOutputObjects()

{

// create a new TList that OWNS its objects

fOutputList = new TList();

fOutputList->SetOwner(true);

// create a histogram:

//from ROOT’s online documentation, this is the constructor for a TH1F:

//TH1F (const char \*name, const char \*title, Int\_t nbinsx, Double\_t xlow, Double\_t xup)

//seen below, we give the histogram the pointer name defined in the header file and give the histogram plot the same title, we define the histogram itself to have 100 bins on an x-axis bounded by [0,100]

fHistPt = new TH1F("fHistPt", "fHistPt", 100, 0, 100);

//add the histogram to the output list:

fOutputList->Add(fHistPt);

// add the list to our output file

PostData(1,fOutputList); //calling PostData() notifies client tasks of the fOutPutList data container that its contents have changed

}

//UserExec: the “event loop” (operations defined here are called for each event in the analysis):

AliAnalysisTaskMyTask::UserExec(Option\_t\*)

{

// get an input event from the analysis manager and cast it as an AliAODEvent

fAOD = dynamic\_cast<AliAODEvent\*>(InputEvent());

// check if there actually is an event, and throw a fatal exception with error message if not

if(!fAOD)

::Fatal("AliAnalysisTaskMyTask::UserExec", "No AOD event found, check the event handler.");

// Loop over all the tracks in the event and fill the histogram

// get the number of tracks in the input event

int iTracks{fAOD->GetNumberOfTracks()};

// iterate through all the tracks in the event:

for(int i{0}; i < iTracks; i++) {

//get the current track, cast it as an AliAODTrack

AliAODTrack\* track = static\_cast<AliAODTrack\*>(fAOD->GetTrack(i));

//if the track variable does not exist after the above operation, continue to the next iteration of the loop

if(!track) continue;

// here we do some track selection

if(!track->TestFilterbit(128) continue;

// get the transverse momentum of the track and fill the histogram with this data

fHistPt->Fill(track->Pt());

}

// save the output list

PostData(1, fOutputList);

}

##### The AddTask macro (.C)

Reproduced and modified from (35):

//this file instantiates our class, defines its input and output, and connects it to the analysis manager

AliAnalysisTaskMyTask\* AddMyTask(TString name = "name") {

//get a pointer to the analysis manager

AliAnalysisManager \*mgr = AliAnalysisManager::GetAnalysisManager();

// resolve the name of the output file

TString fileName = AliAnalysisManager::GetCommonFileName();

fileName += ":MyTask"; // create a subfolder in this file

// create an instance of the analysis task

AliAnalysisTaskMyTask\* task = new AliAnalysisTaskMyTask(name.Data());

// add this task to the analysis manager

mgr->AddTask(task);

// connect the manager to the task’s input container

mgr->ConnectInput(task,0,mgr->GetCommonInputContainer());

// connect the manager to the task’s output container (TList)

mgr->ConnectOutput(task,1,mgr->CreateContainer("MyOutputContainer", TList::Class(), AliAnalysisManager::kOutputContainer, fileName.Data()));

// important: return a pointer to this task

return task;

}

Appendix D: Software Environment, Packages & Utilities

## Software Environment

### AiROOT

AliROOT was built locally using alidock Docker container.

AliROOT was built from source on the hep01 server hosted at UCT

### R Statistical Software

#### Packages

### ROOTR

### Keras & Tensorflow

In this case, here is the suspect process ($processid = 28525):

gviljoen 28525 0.0 0.0 119612 2168 ? S 13:02 0:00 sshd: gviljoen@pts/0

### Python

Appendix E:

Appendix F: Data Extraction, Data Quality and Data Pre-processing

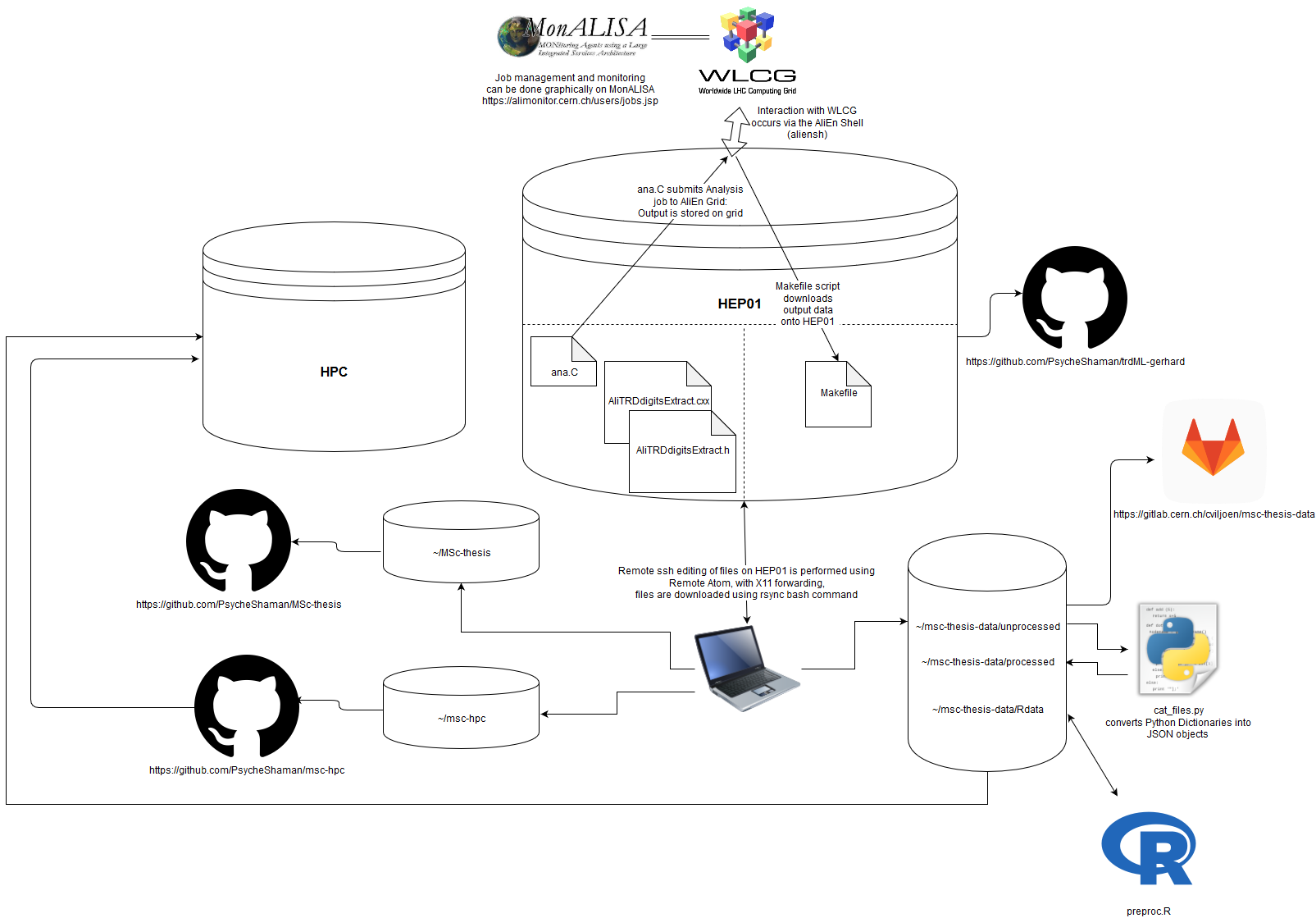


Figure 8: Data pre-processing environment and repository logic

### Running Digits Extract Task on AliEn Grid

* Stage 1
* Stage 2
* Stage 3

### Data Extraction from WLCG

#### From Alien to HEP01

Makefile

#### From HEP01 to Local Machine

Into data backup directory: <https://gitlab.cern.ch/cviljoen/msc-thesis-data>

Directory structure

scp -r gviljoen@hep01.phy.uct.ac.za:/alice/cern.ch/user/c/cviljoen/wd/outDir265377/000265377/ .

rsync -av --stats --progress gviljoen@hep01.phy.uct.ac.za://alice/cern.ch/user/c/cviljoen/wd/od/ .

#### From Local Machine to HPC

### Data Quality Assessment and Descriptive Statistics

### File Merging and Conversion of Python Dictionaries to JSON Objects

Cat\_files.py

### Loading JSON Files into R Environment and Data Wrangling for Deep Learning

Wrangle.R

Appendix G: Old Methods Section

#### Model 1

An initial benchmark feedforward model was built, compiled and trained, according to the following lines of Python code:

model1 = Sequential([

Dense(256, input\_shape=(24,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(64),

Activation('relu'),

Dense(2),

Activation('softmax')

])

model1.compile(loss='categorical\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

history = model1.fit(x\_train, y\_train,

epochs=epochs,

validation\_split=0.15,

shuffle=True,

verbose=2)

The input features to this model were as follows:

* Time-bin sums across all pads, divided by the mean of the entire x sample
* Missing data removed
* Electrons oversampled (the electron sample in the training data was taken thrice)

This model was trained on 5778261 samples and validated on 642029 samples.

As can be seen in 0, this model failed to train, so an approach was taken to account for class imbalances using a different method, and by starting with a very simple architecture and sequentially adding complexity to the model.

#### Sequential Model Building

##### Stage 1

The model was built, compiled and trained according to the following Python code:

sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)

model1\_dropout\_0\_5 = Sequential([

Dense(32, input\_shape=(32,)),

Activation('relu'),

Dense(2),

Activation('softmax')

])

batch\_size=32

model1\_dropout\_0\_5.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

history = model1\_dropout\_0\_5.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_split=0.1,

shuffle=True,

verbose=2,

class\_weight=class\_weights)

Model was trained on 883591 samples, and validated on 98177 samples, due to undersampling of pions.

As can be seen in 0, this model did train, in contrast to the first model, although it did not achieve validation accuracy above 75%.

##### Stage 2

Following the successful running of the above model, the same model was run, without compensating for class imbalances by undersampling pions, but by maintaining the proportionally greater contribution to the loss function by the underrepresented “electron” class.

This model was trained on 2720444 samples and validated on 302272 samples.

##### Stage 3

After successful training of the single layer, 32 node neural network above, a larger network was constructed as follows, using the same dataset, optimizer, etc.

model1\_dropout\_0\_5 = Sequential([

Dense(128, input\_shape=(32,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(2),

Activation('softmax')

])

As can be seen in 0, increasing the model capacity in this way does have its benefits in terms of accuracy, without seeming to overfit too much, therefore the next model was built with much higher capacity.

##### Stage 4

model1\_dropout\_0\_5 = Sequential([

Dense(128, input\_shape=(32,)),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(128),

Activation('relu'),

Dense(2),

Activation('softmax')

])

While this model was slightly more accurate than those discussed before, it is clear when looking at 0 that the model was not generalizable to the validation set, therefore, before increasing model complexity, regularization in the form of dropout was introduced as follows:

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

Equation

### 2D Convolutional Neural Networks

#### Stage 1

Trained on 770981 samples, validated on 85655 samples:

epochs = 100

model = Sequential()

model.add(Conv2D(16, (2, 2), padding='valid',input\_shape=(17,24,1),data\_format="channels\_last"))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(3, 3)))

model.add(Flatten())

model.add(Dense(256))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(2))

model.add(Activation('softmax'))

sgd = tensorflow.keras.optimizers.SGD(lr=0.01, clipvalue=0.5)

model.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

history=model.fit(x\_train, y\_train,

epochs=epochs,

validation\_split=0.1,

shuffle=True)

#### Stage 2

epochs = 100

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu', input\_shape=(x\_train.shape[1],x\_train.shape[2],x\_train.shape[3]),data\_format="channels\_last"))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

sgd = tensorflow.keras.optimizers.SGD(lr=0.01, clipvalue=0.5)

model.compile(loss='binary\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

batch\_size=32

history=model.fit(x\_train, y\_train,

epochs=epochs,

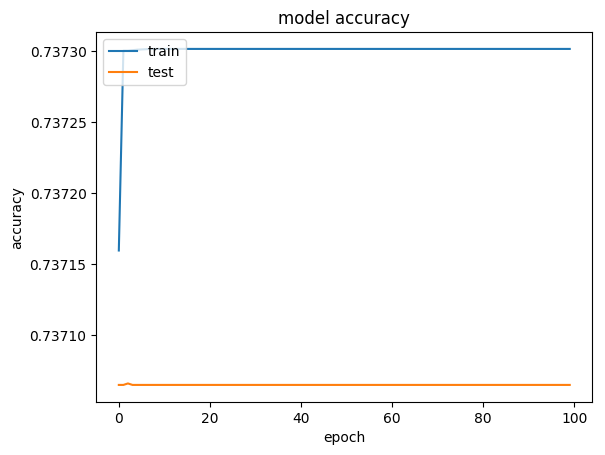
validation\_split=0.1,

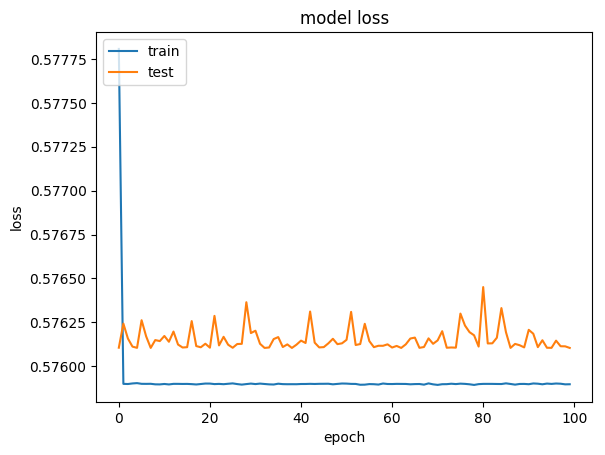
shuffle=True)

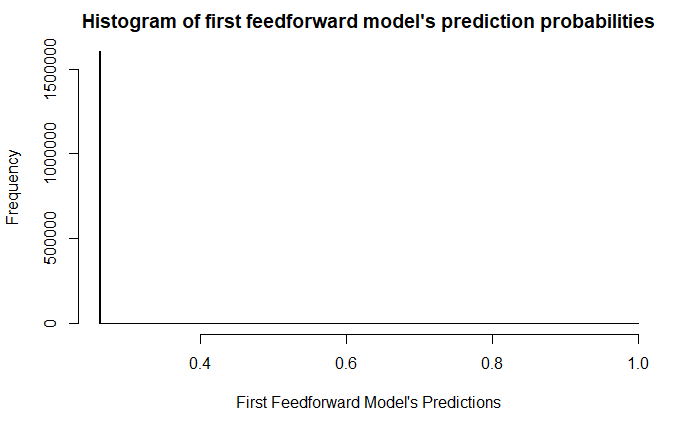
### Recurrent Neural Networks

Appendix H: Old Results Section

#### Model 1

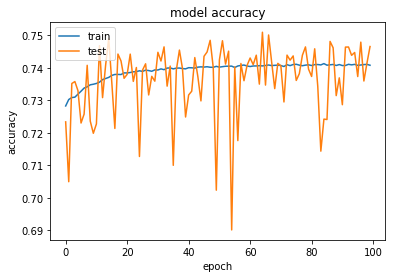


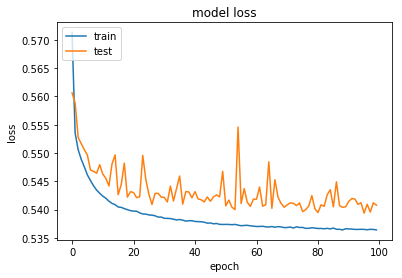


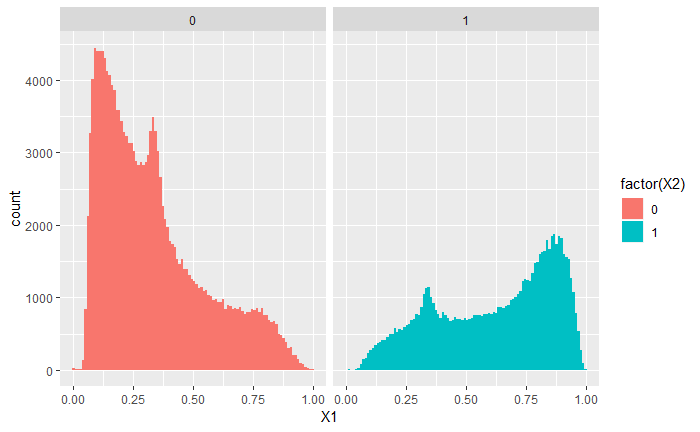


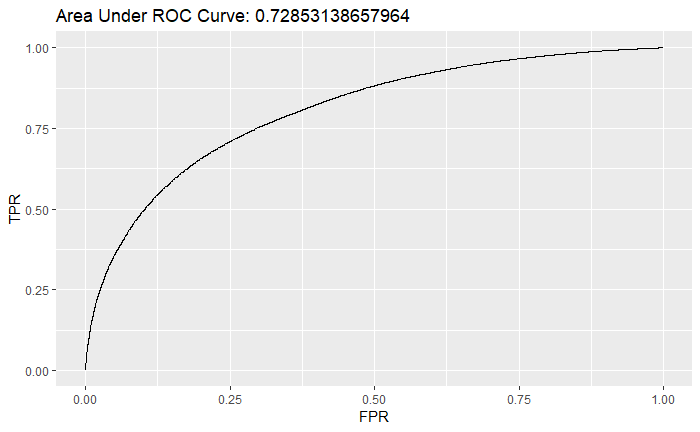
#### Sequential Model Building

##### Stage 1

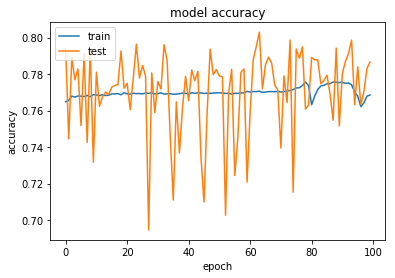


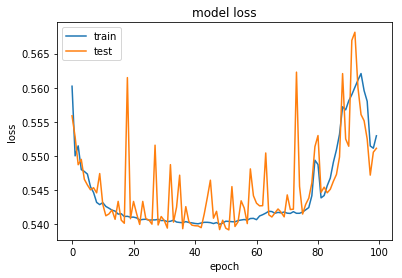


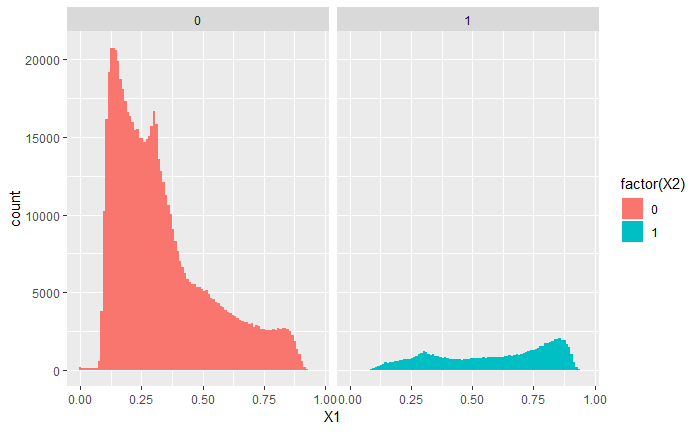


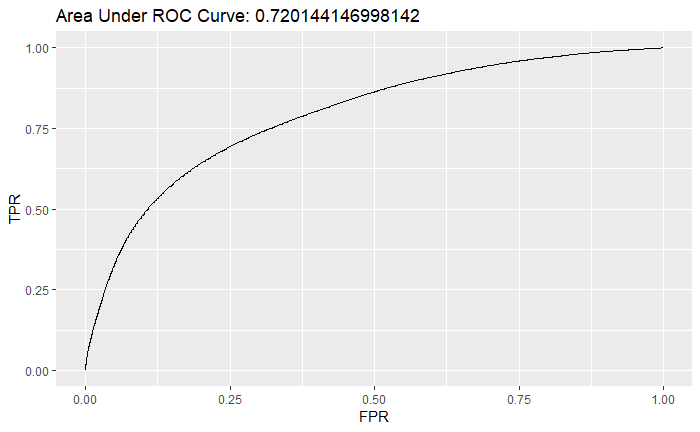


##### Stage 2

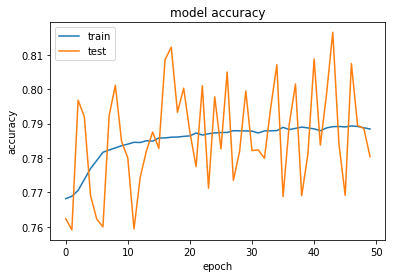


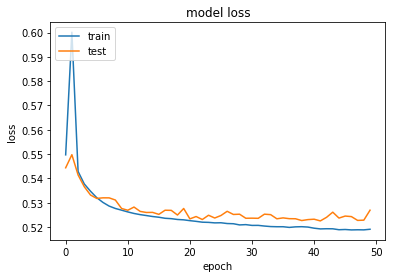


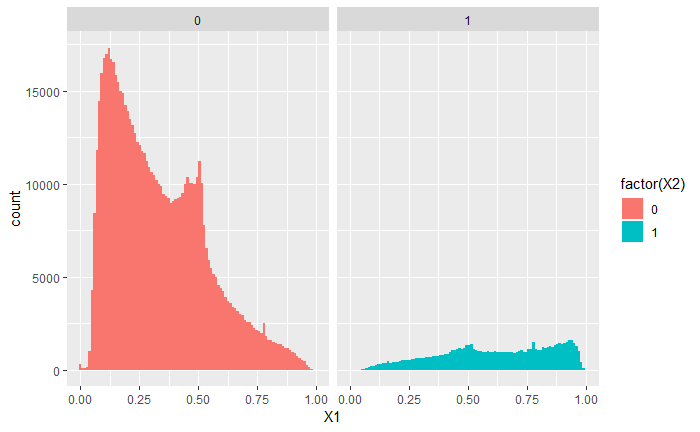


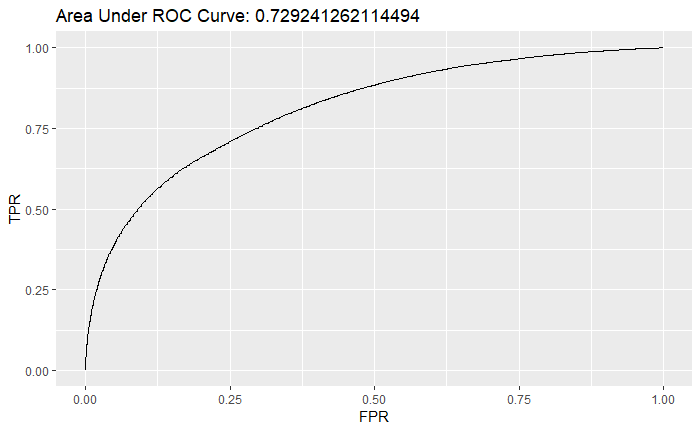


##### Stage 3

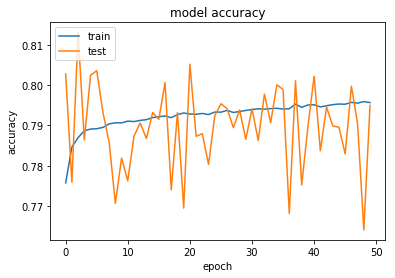


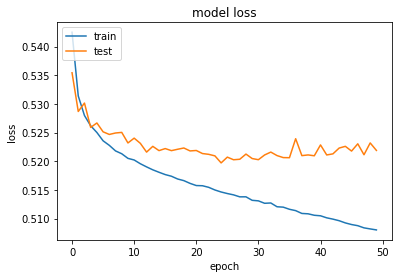






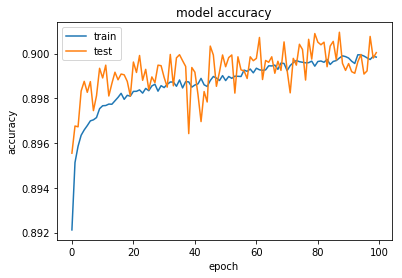
##### Stage 4

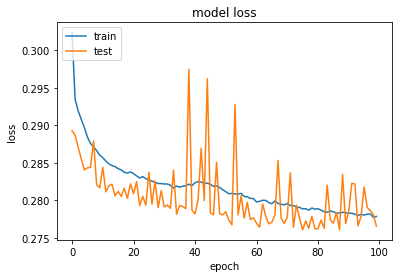




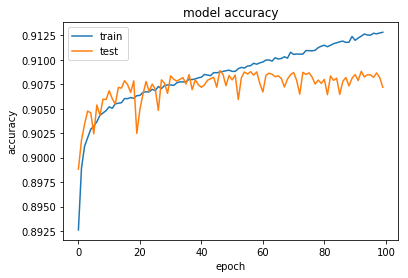
### Convolutional Neural Networks

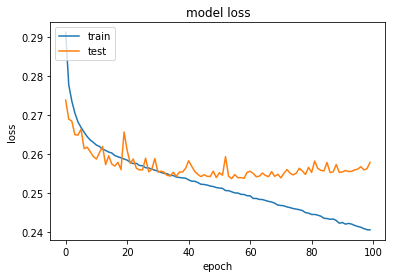
#### Stage 1



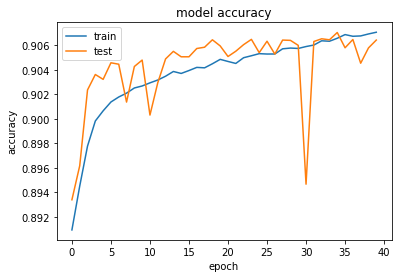


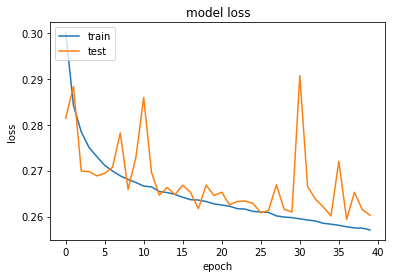
#### Stage 2





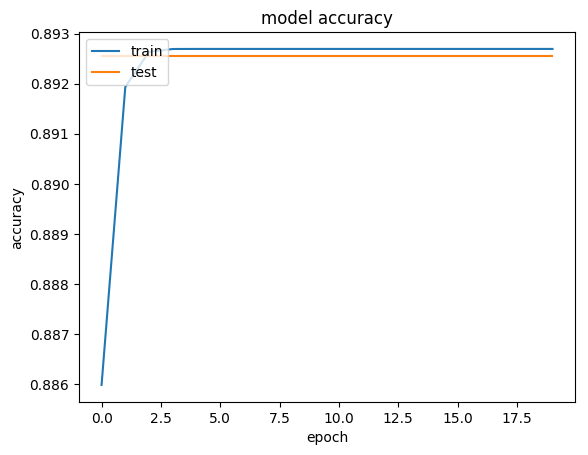
#### Stage 3

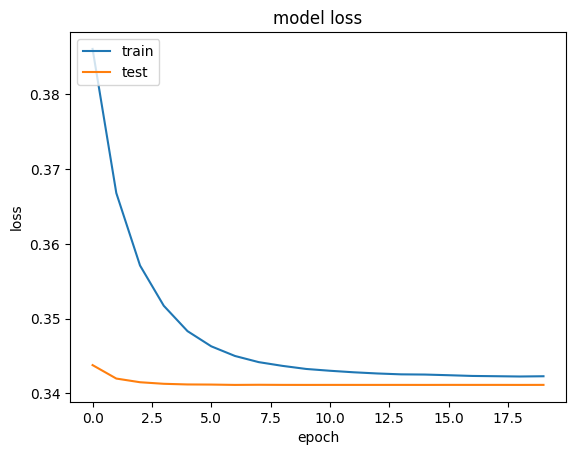




#### Stage 4

#### Stage 5





## Autoencoders



Appendix i: 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, EWT (QED for the electromagnetic force and 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 [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 **Error! Reference source not found.** 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 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 9 shows the main Standard model interaction vertices in the form of Feynman diagrams.

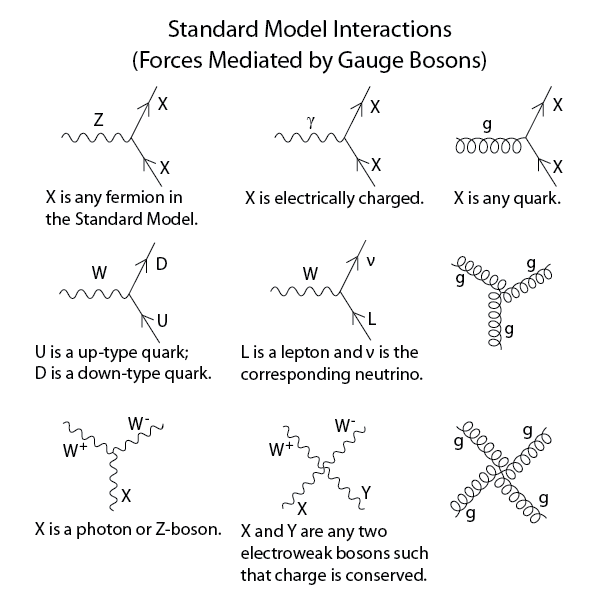


Figure 9: Standard model interaction vertices [2]

Appendix J: Data Center

CERN has 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).

This is wrong:

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

= 13 TeV [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 and the kinetic energy from acceleration is . To simplify the equations, the speed of light, is set at a constant [23].

TOTEM and LHCf are smaller experiments focused on particles emitted in the forward direction 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].

. 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

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.

### A Note on Geometry

Figure 18 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].

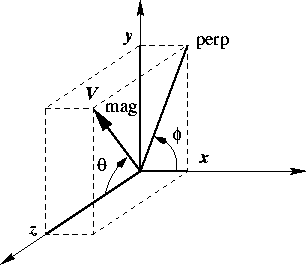


Figure 18: Cylindrical coordinates as used in geometric coordinate specifications for measurements made in experiments conducted at the LHC [42].

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, discussed in section **Error! Reference source not found.**) 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 (ρ, θ, ϕ) of a point P, one can firstly obtain ρ, 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 θ, as the angle between the positive x-axis and the line segment from the origin to point Q. Finally, one would calculate ϕ 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, , which is a specification of a particle’s angle relative to the beam (z-) axis.

## 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 [38].

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 [38].

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 [38].

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 [38].

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.) [38].

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. [38]

## Mathematical Background for Deep Learning

### Rosenblatt’s Perceptron

For k = hidden layers, , we compute the element-wise gradient on the layer’s output (before the non-linear activation function is applied):

And the gradients on the weights and the bias term:

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

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

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

Where α is a weighting hyperparameter, determining the extent of contribution of the parameter norm penalty to the magnitude of the regularized loss function J̃, i.e. setting α = 0 eliminates regularization and increasing its value results in more regularization [38].

Various norms Ω can be used in such a setup and can be applied to the entire set of network parameters θ 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 [38].

Ideally, each ANN layer should have its own α coefficient, but doing so increases the search space for the optimal value, so a global α is sometimes used in practice [38].

##### 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:

In general, the norm is specified by:

##### Weight Decay Regularization

The 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

to the objective function [38].

The norm is known as the Euclidean norm, because it gives the magnitude of the Euclidean distance from the origin to the point defined by . It is squared in this regularization technique for computational efficiency, because calculating the derivative with respect to each component of the unsquared norm involves all its elements, whereas the derivative for each component of the squared norm depends only on the corresponding element of [38].

Figure 19 (RHS) illustrates the manner in which introducing an norm penalty introduces an additional constraint on the objective function, i.e. having to minimize the magnitude of the 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 [38].

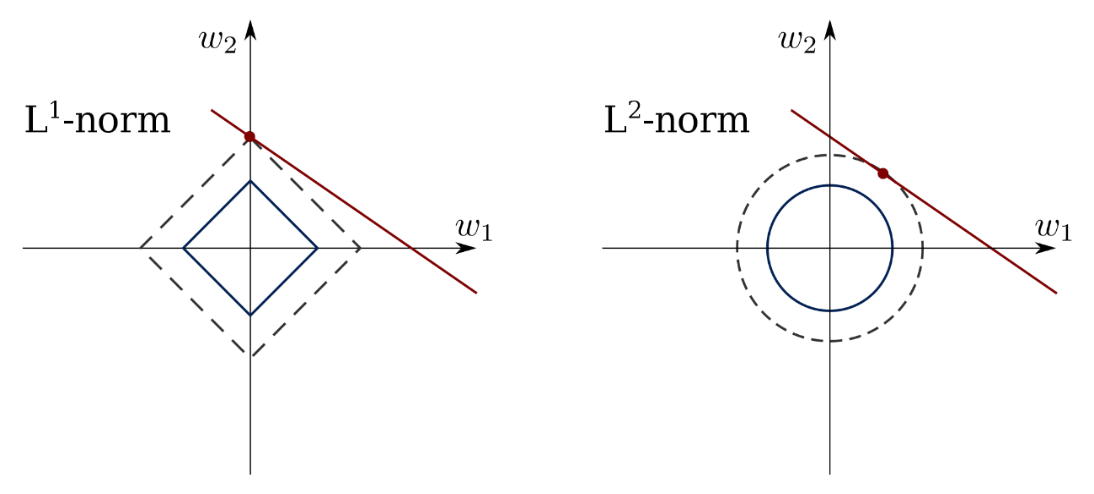


Figure 19: and norm penalties

##### Regularization

regularization adds a slightly different weighted parameter norm penalty

to the objective function.

When 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 regularization; but in contrast to regularization, allows weights to be shrunk down to zero, resulting in a more sparse neural network, depending on the magnitude of the weighting parameter . 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 [38].

When using either or regularization, care has to be taken to select the right level for , since a large 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 [38].

##### 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 [38].

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 [38].

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 [38].

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 [38].

##### 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) [38].

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 [38].

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 [38].

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 [38].

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 [38].

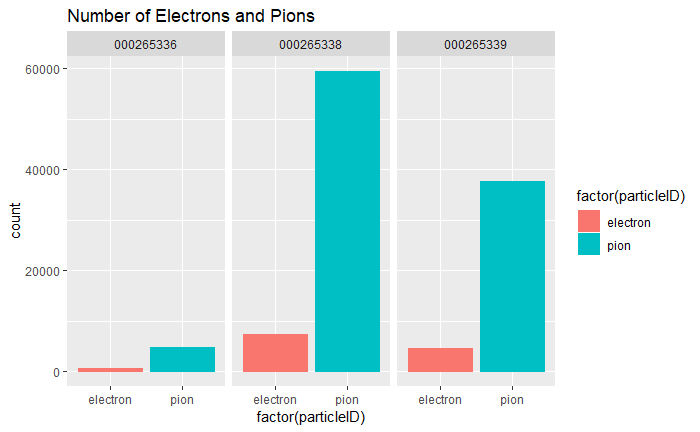
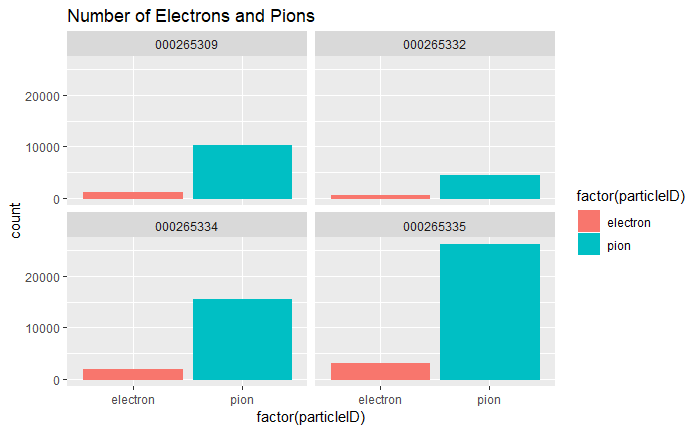
Early stopping is often used in conjunction with other regularization techniques, since it is unobtrusive towards the learning dynamics, i.e. it does not change how the neural network arrives at its optimal weights, it simply changes when to stop adjusting them to prevent overfitting [38].

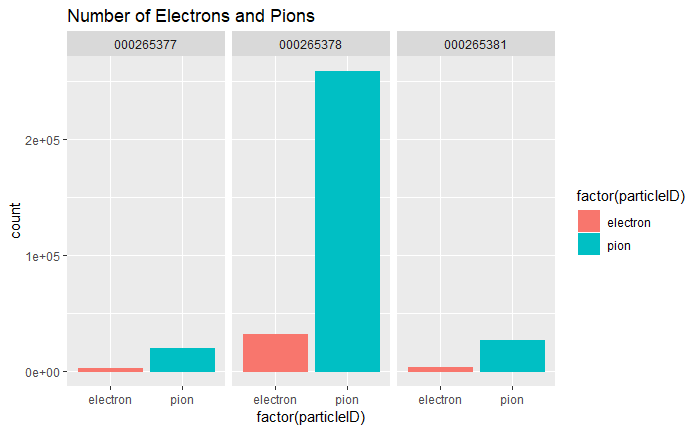
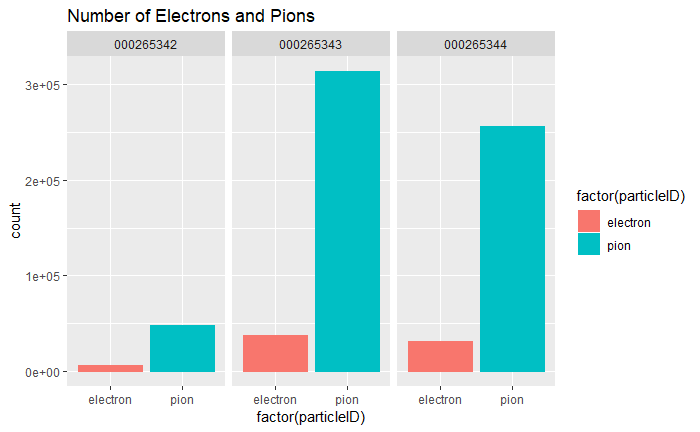
Full convolutions result from applying enough zero-padding to allow each pixel to be visited k times in each direction of the convolution operation, and therefore should result in an output with m+k-1 pixels. This results in output pixels near the border being influenced by fewer pixels than output pixels near the centre, making the kernel harder to train [38].

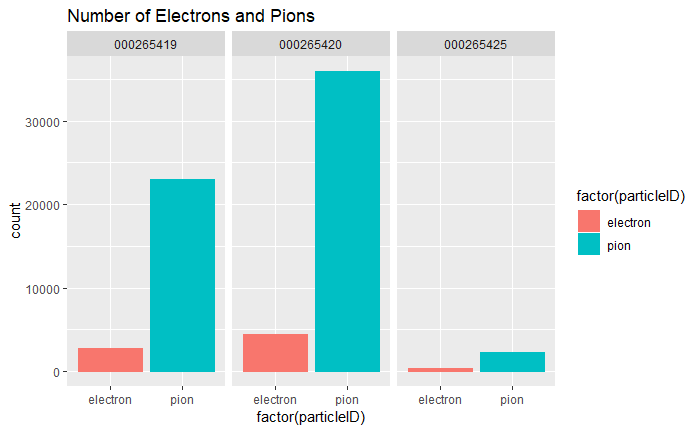
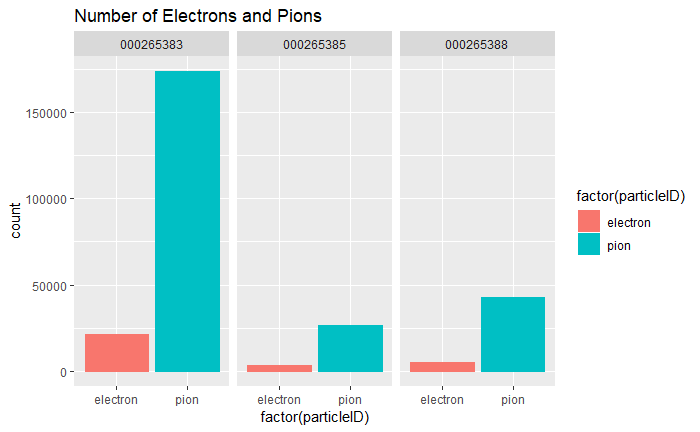
The ideal amount of padding generally lies between the amount of padding required to achieve valid- and same convolutions [38].

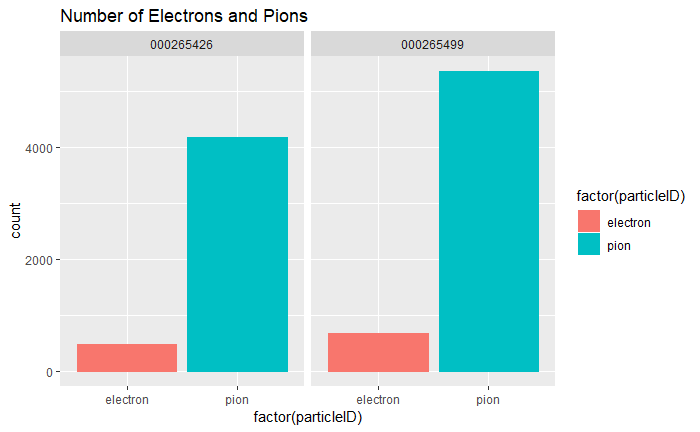
## Electron and Pion Counts per Run

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.



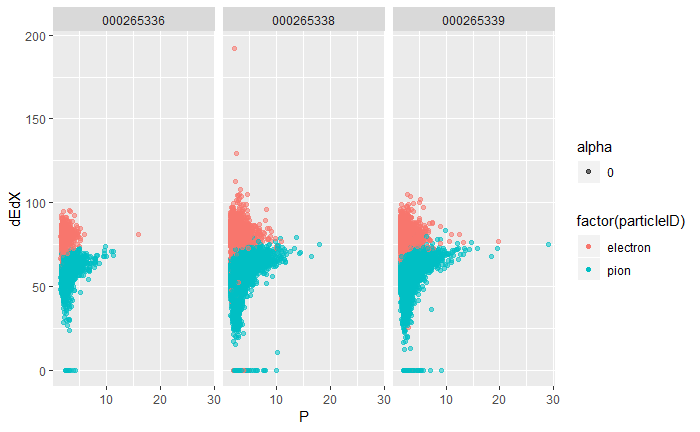
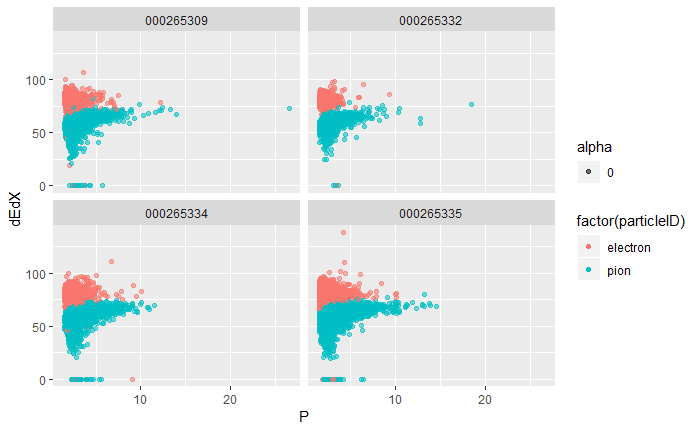


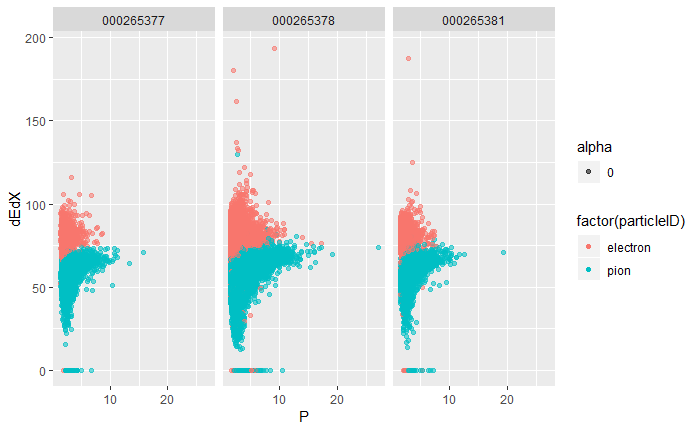


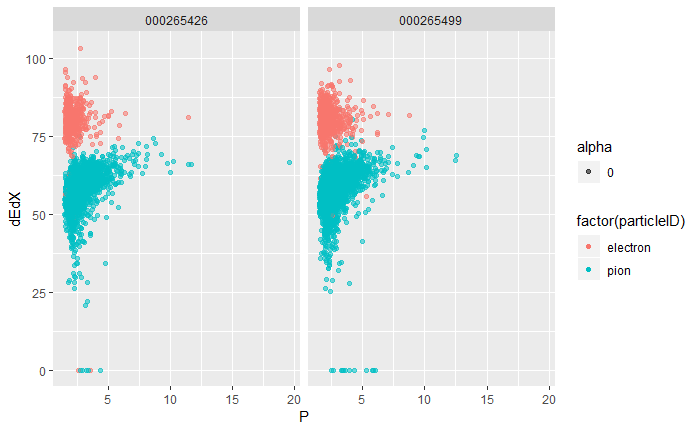
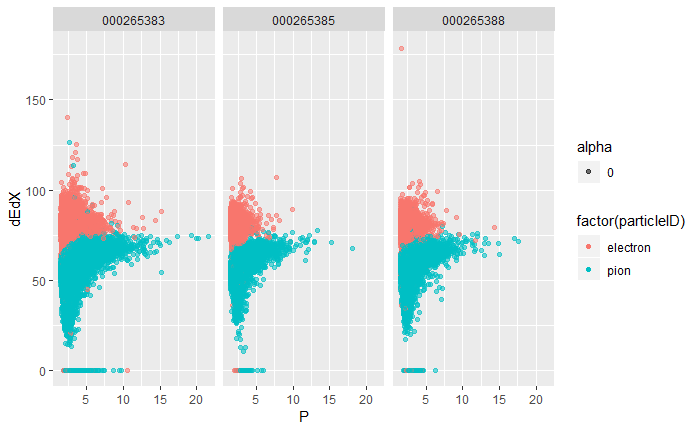


## Bethe Bloch Curve per Run for Electrons and Pions

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.

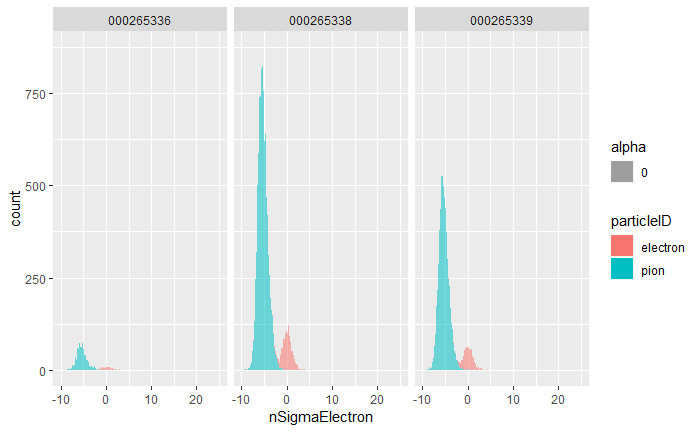
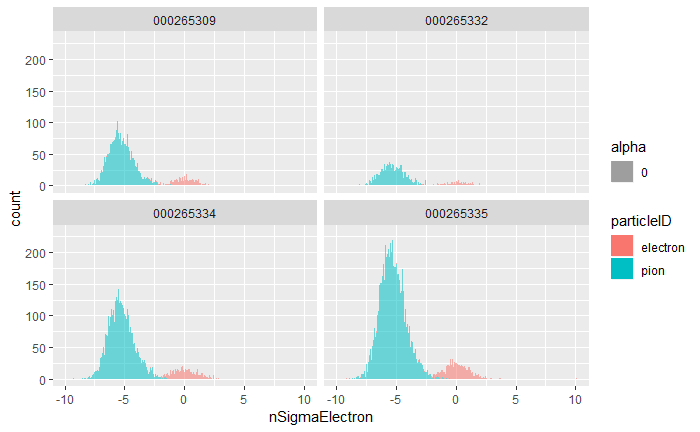


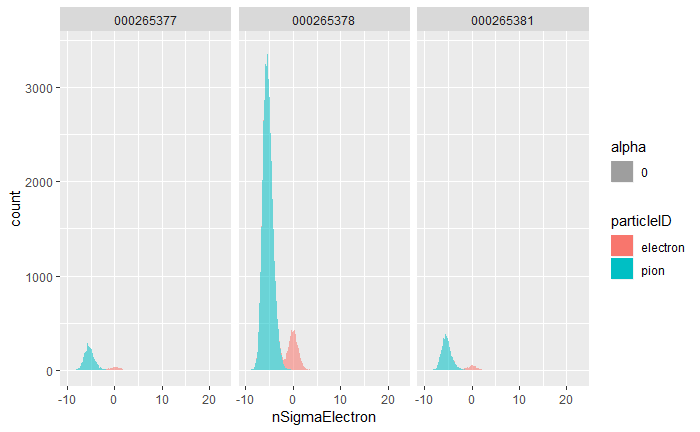
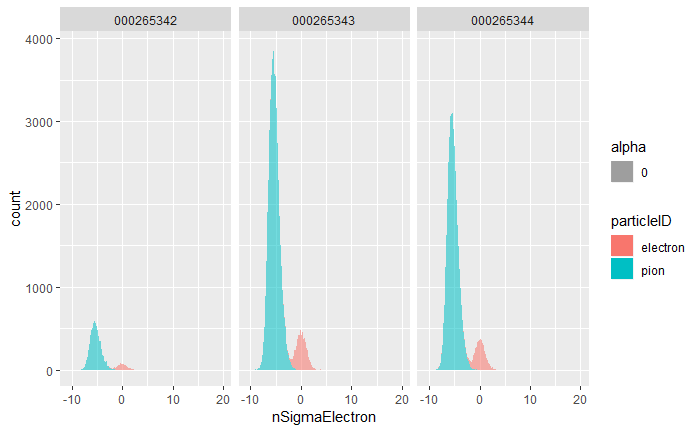


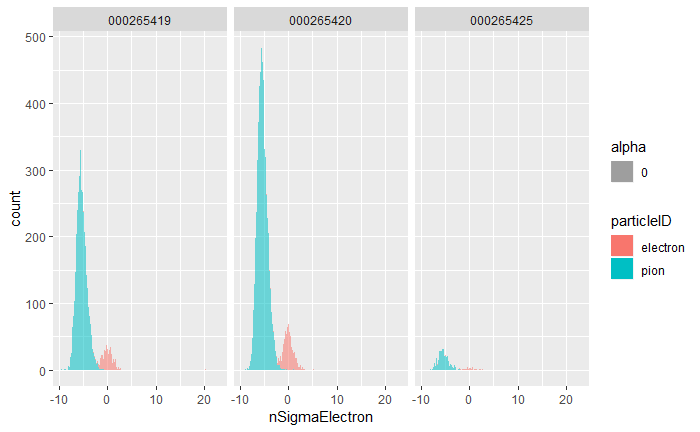
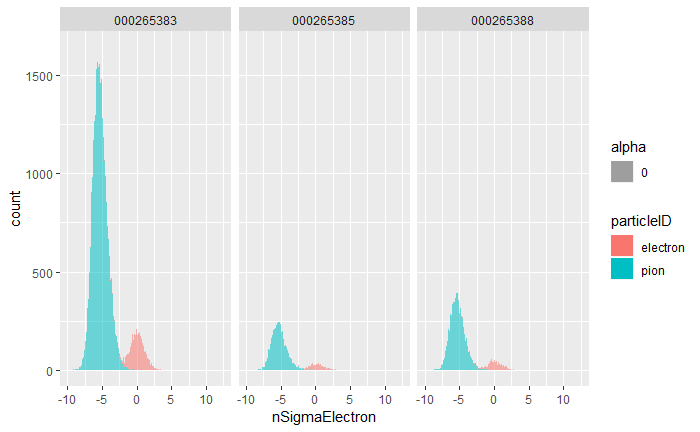


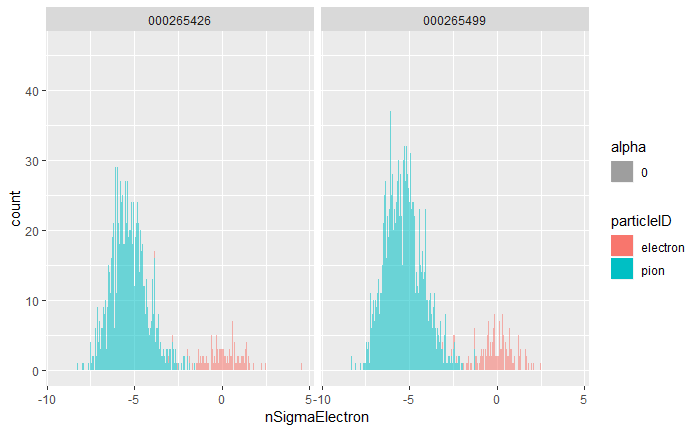
## nσ Electron per Run for Electrons and Pions

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 centred around an nσ Electron value of around -5.









ROOT is freely available for download from [25] 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 [24].

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 [12], Fermilab’s Tevatron, which is no longer in operation, was 6.3 km in circumference [13] and the KEKB accelerator in Tsukuba, Japan also has a circumference of around 3 km [14].

It is also the most powerful particle accelerator in the world, compared to RHIC, which operates at [12], the Tevatron, which reached [13] and KEKB at [16].

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

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

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

Applying column-wise scaling and centering, to normalise the dataset as follows:

A linear regression model was used as a very simple benchmark to compare future results against, using the abovementioned dataset, which contained around 99% pions.

Using a naïve of the 99th percentile of the output distribution of the linear model, a single tracklet pion efficiency of at electron efficiency was achieved.

Following this, was optimized by maximizing the area under the ROC curve when assessing the linear regression model’s results, which improved electron acceptance, but decreased pion rejection results to at electron efficiency .

A benchmark for deep learning was also established using this dataset, by upsampling electrons with replacement to arrive at a sample of equal size to the pion sample.

A fully connected neural network with architecture 256:128:64:32:16:2, with ReLU activations in the hidden layers and softmax activation in the output layer was trained for 500 epochs and achieved at . This model’s accuracy and loss curves are shown in Figure 28.



Figure 27

Figure 28: Training and Loss Curves for Naïve Benchmark Deep Learning Model

and therefore the accuracy and loss curves shown in Figure 29 appear misleadingly successful. In addition was not optimised and was set to and the probabilities for single tracklets were at this stage not combined to reconstruct Bayesian probabilities for the full track case as explained in **Error! Reference source not found.**.

Figure 30 shows that there is not much separability in the probability distribution obtained from this model, with most estimates lying between 0.3 and 0.6.

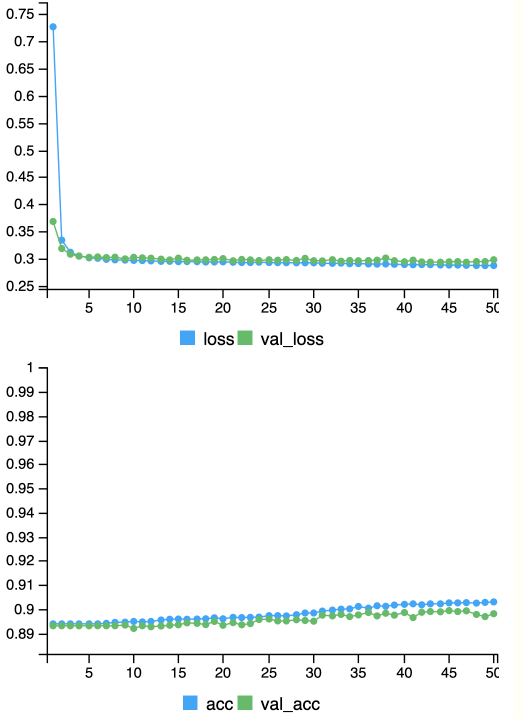


Figure 29

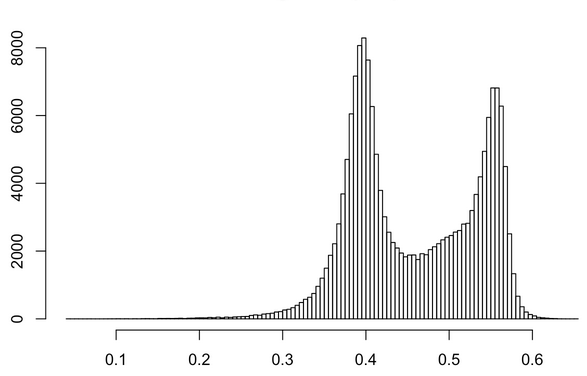


Figure 30

### 1D Convolutional Neural Networks

Figure 36, Figure 37, Figure 38, Figure 39 and Figure 40 summarise the results obtained for 1D Convolutional Neural Networks during the second stage of model building.

### LSTM Networks

### Dense (Fully Connected) Networks

Whilst they may not be very interpretable, the weights for the four convolutional layers of this model are plotted in Figure 32, Figure 33, Figure 34 and Figure 35 below.

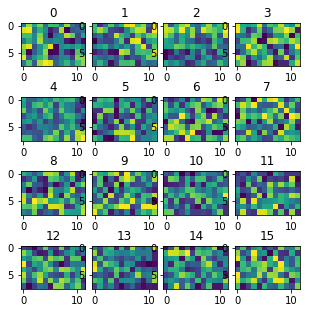


Figure 32: Weights of first convolutional layer

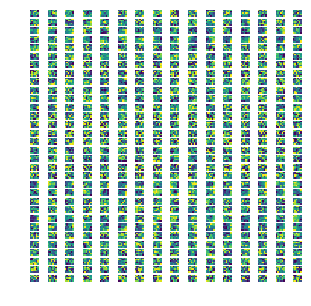


Figure 33: Weights of second convolutional layer



Figure 34: Weights of third convolutional layer



Figure 35: Weights of fourth convolutional layer

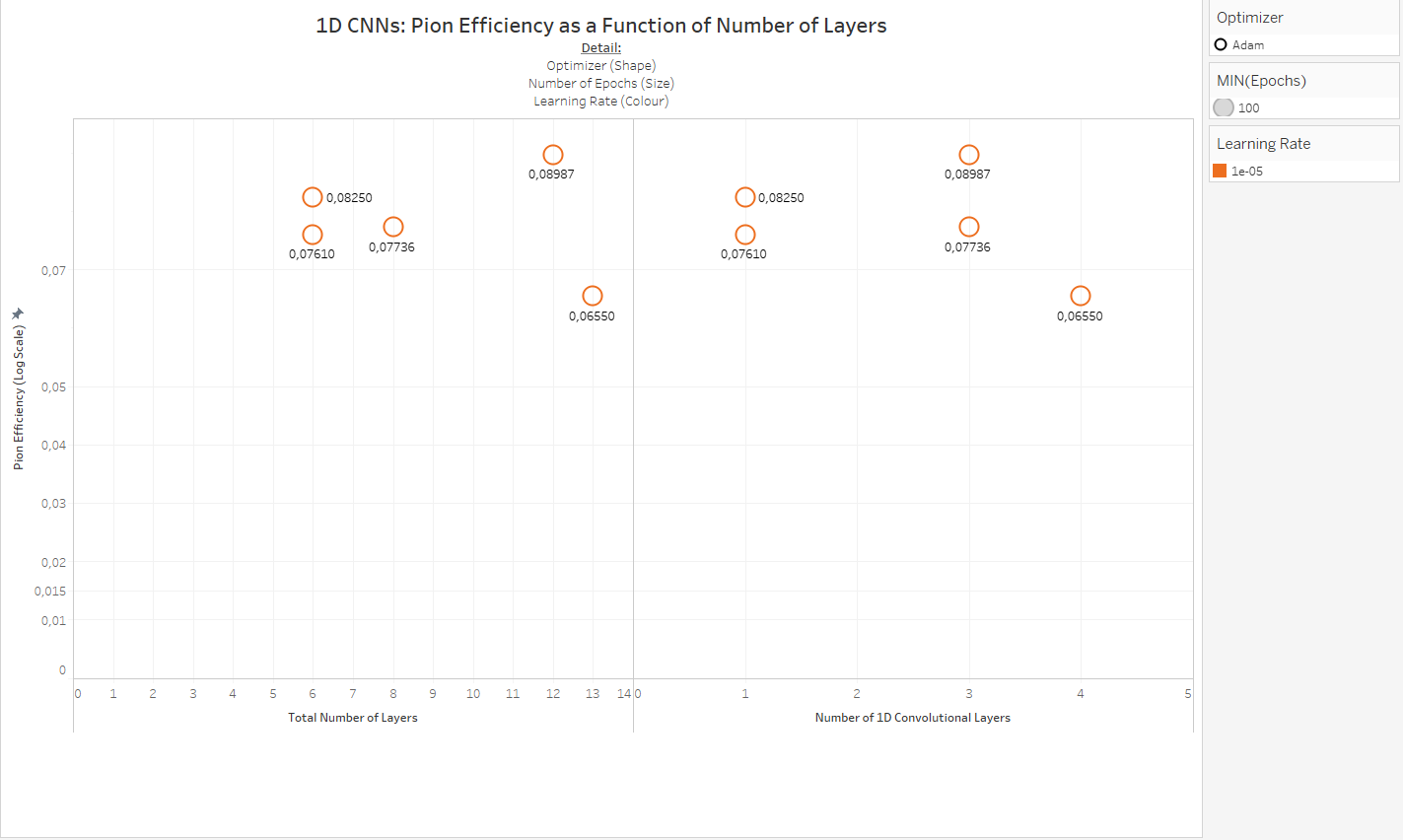


Figure 36

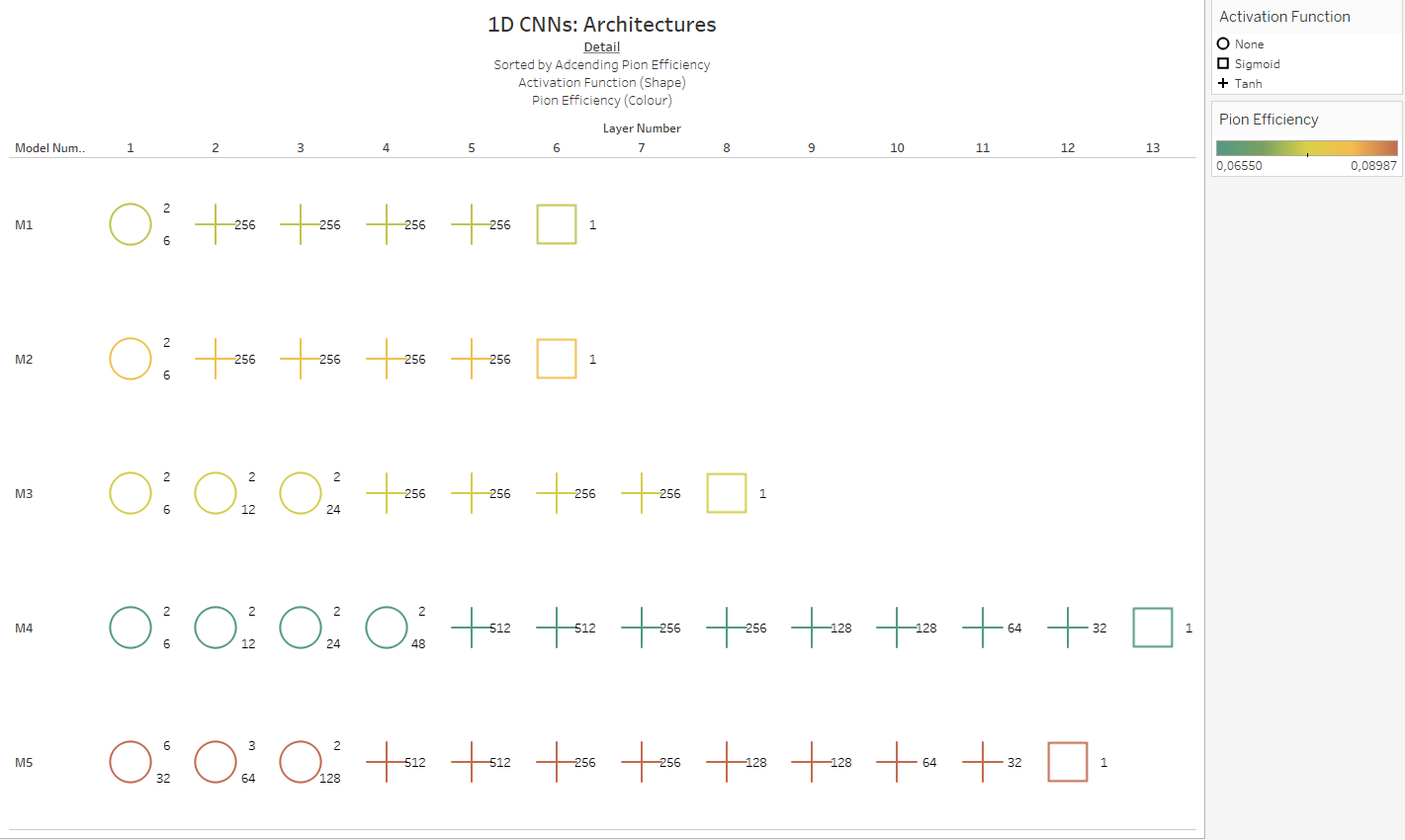


Figure 37

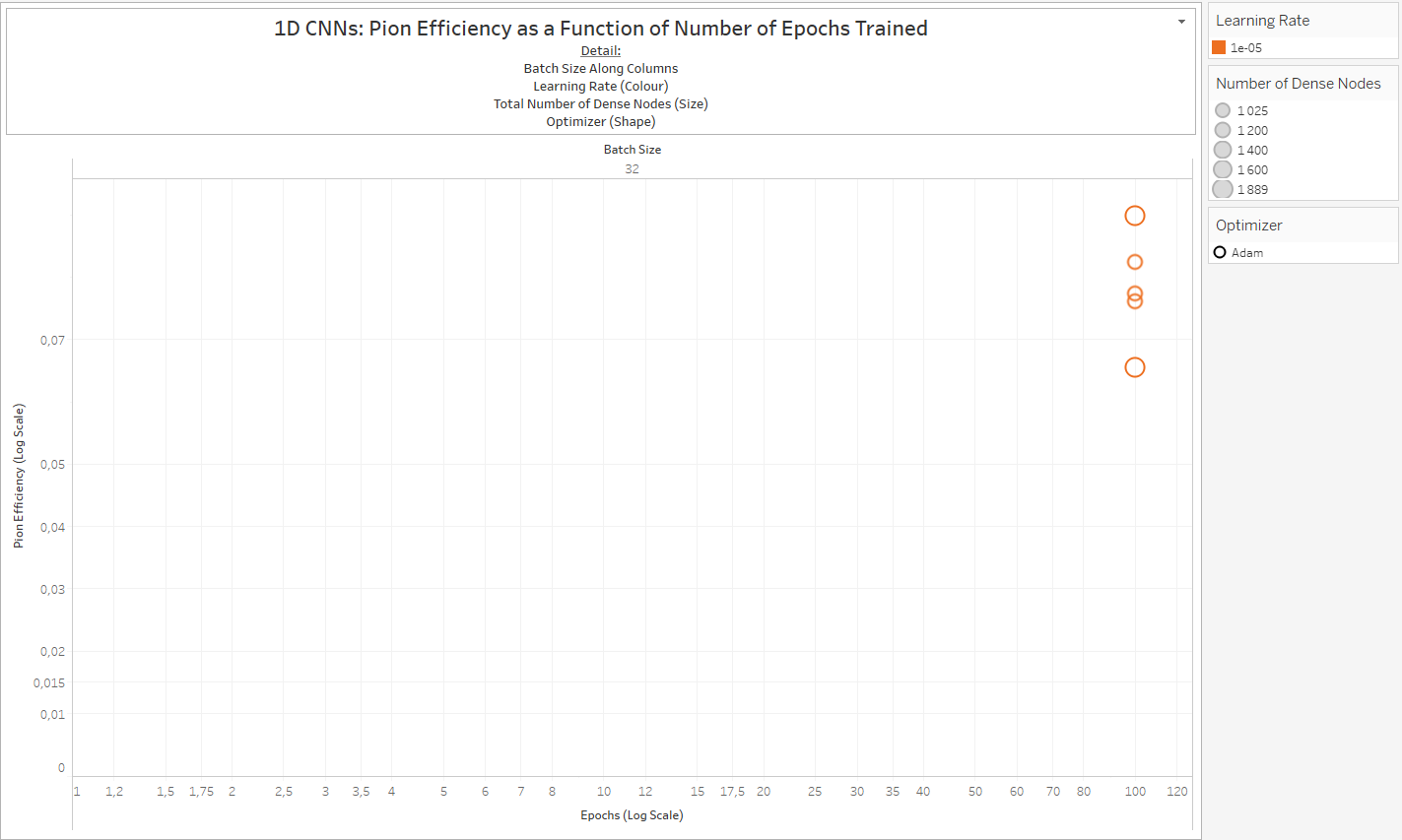


Figure 38

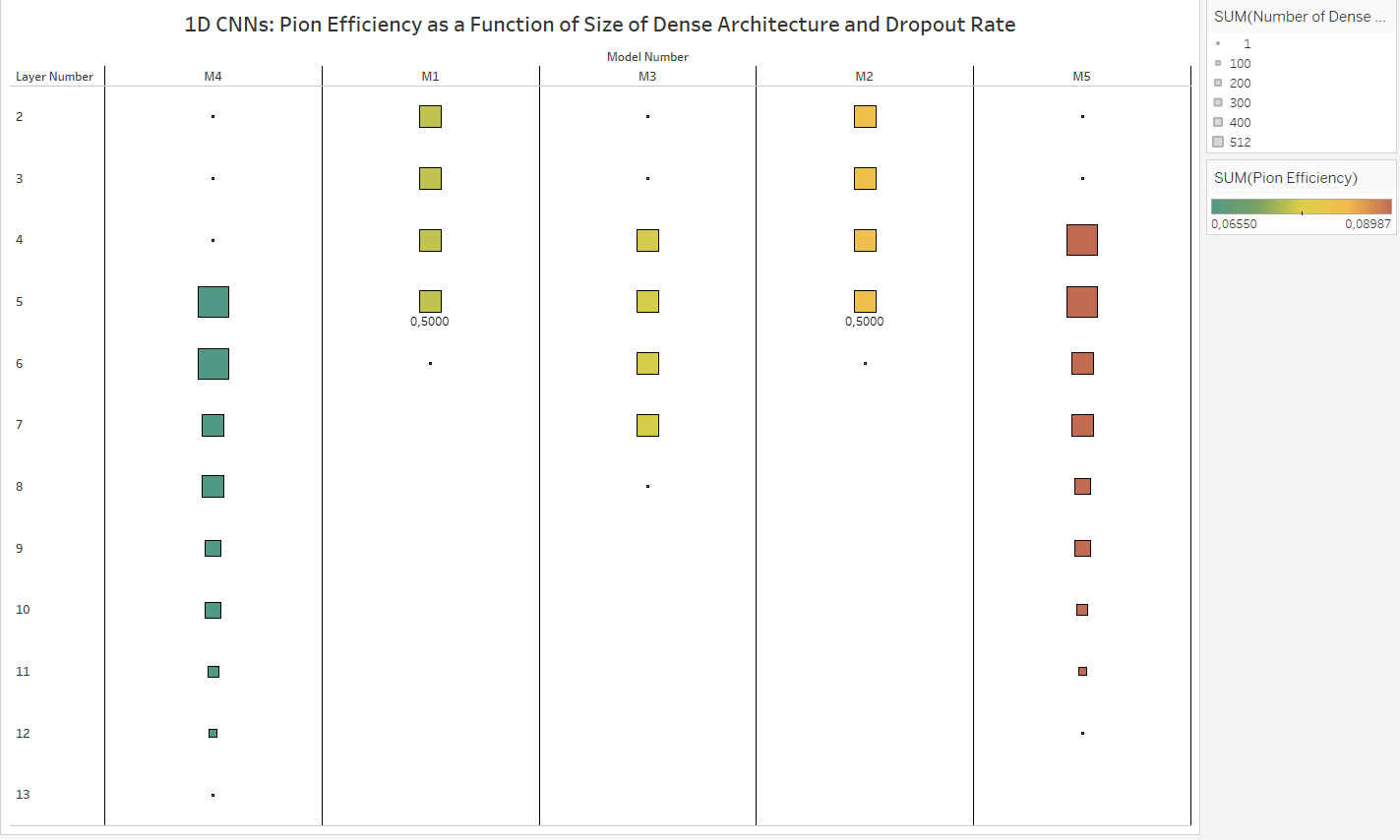


Figure 39

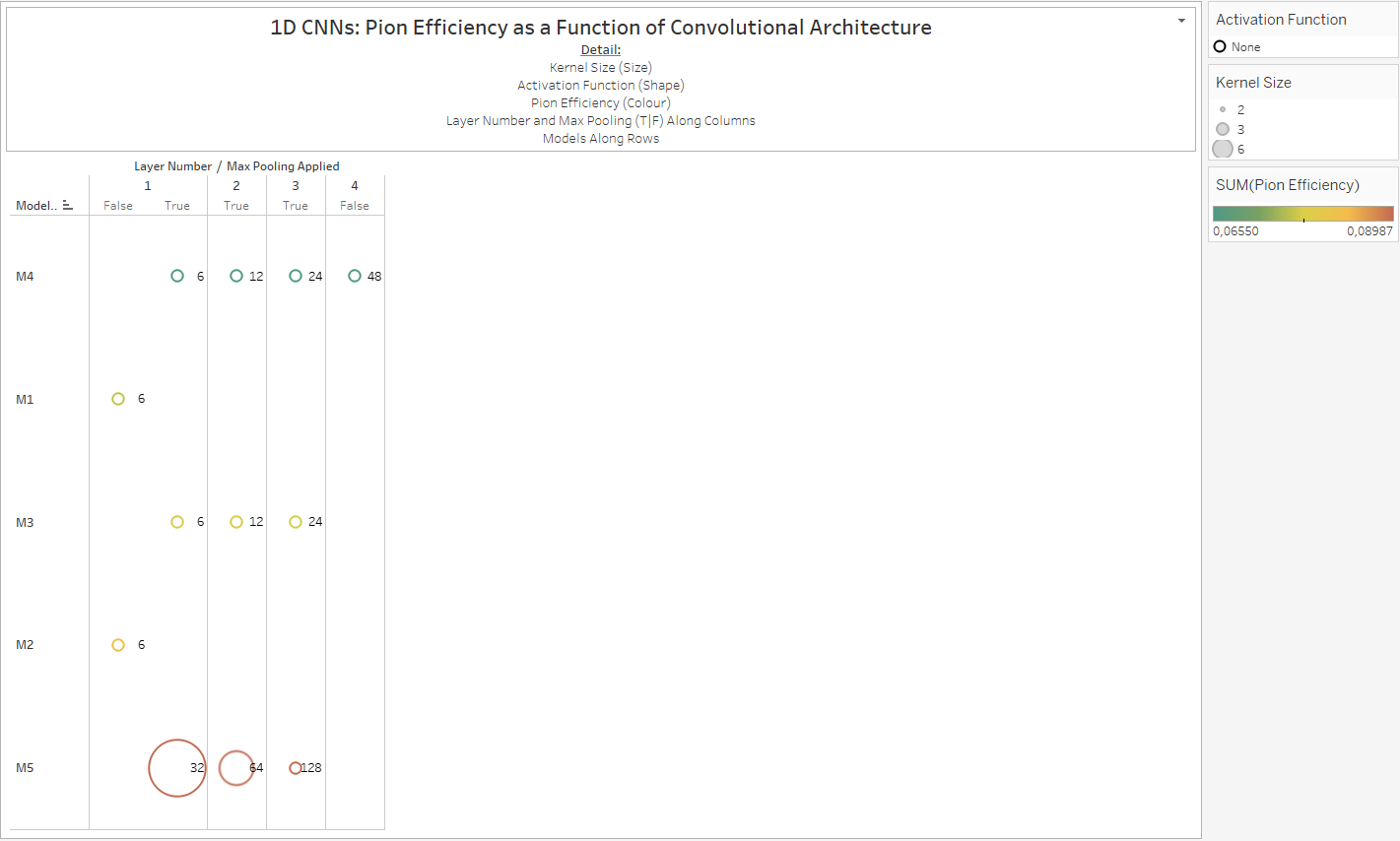


Figure 40

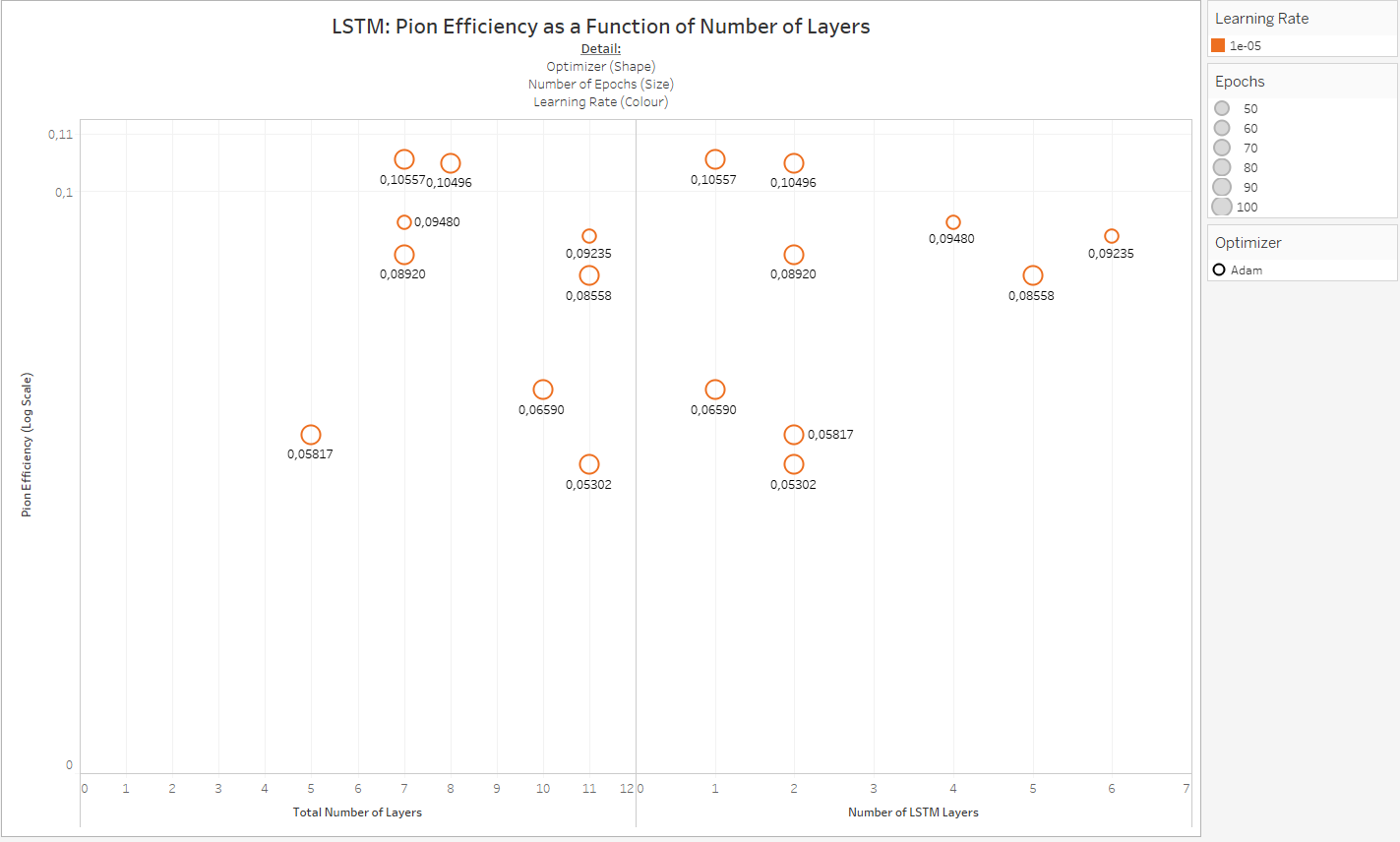


Figure 41



Figure 42

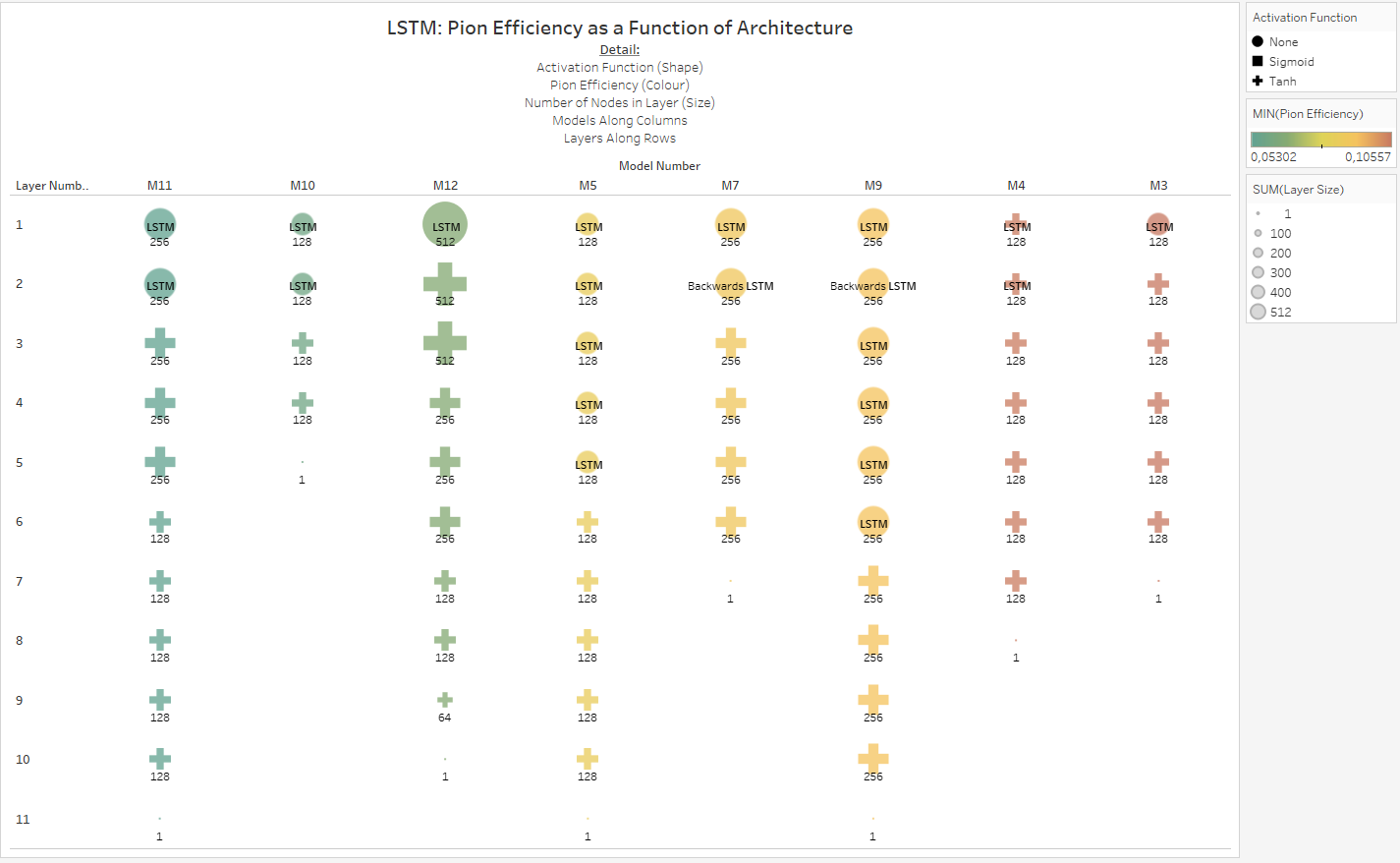


Figure 43

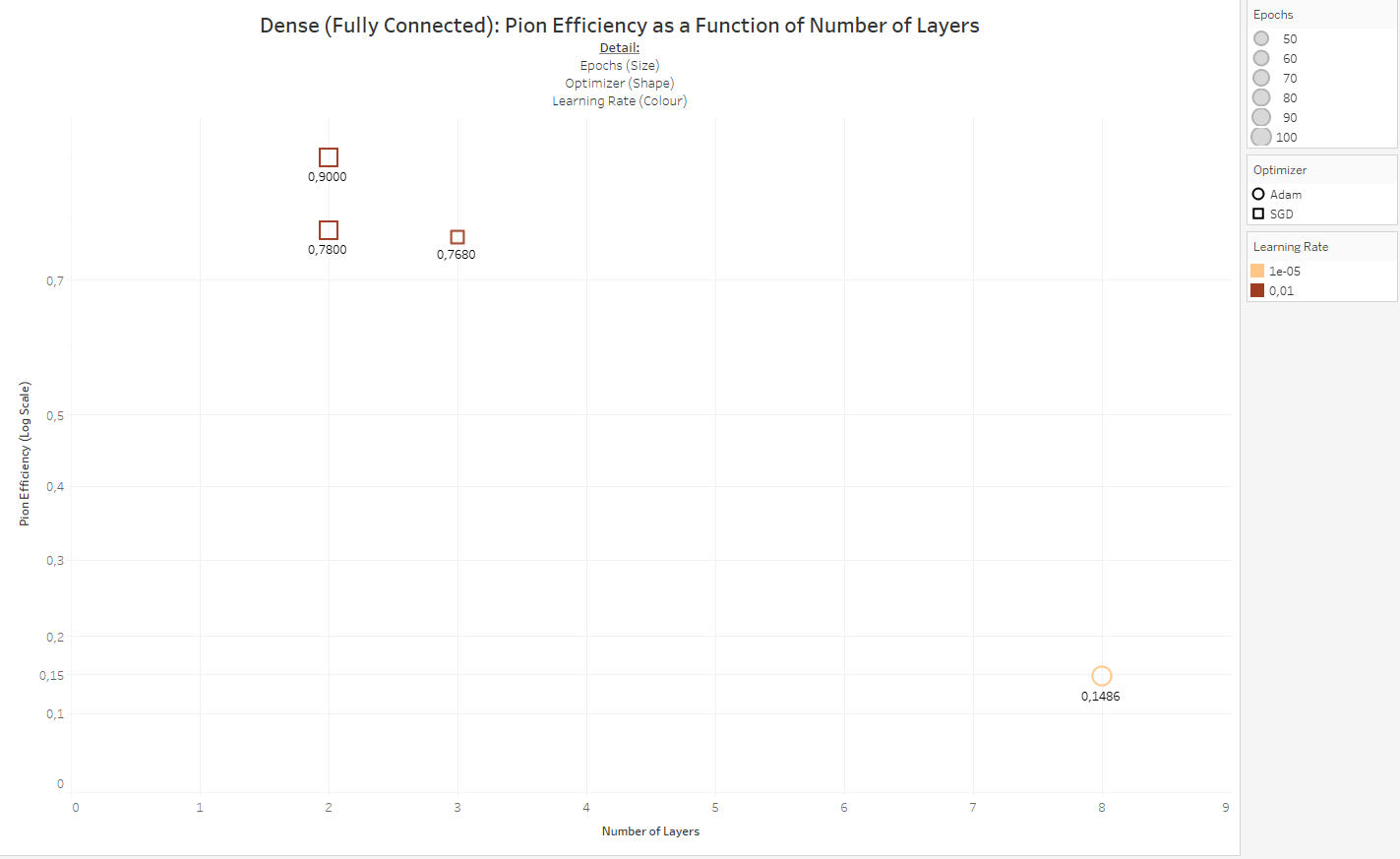


Figure 44

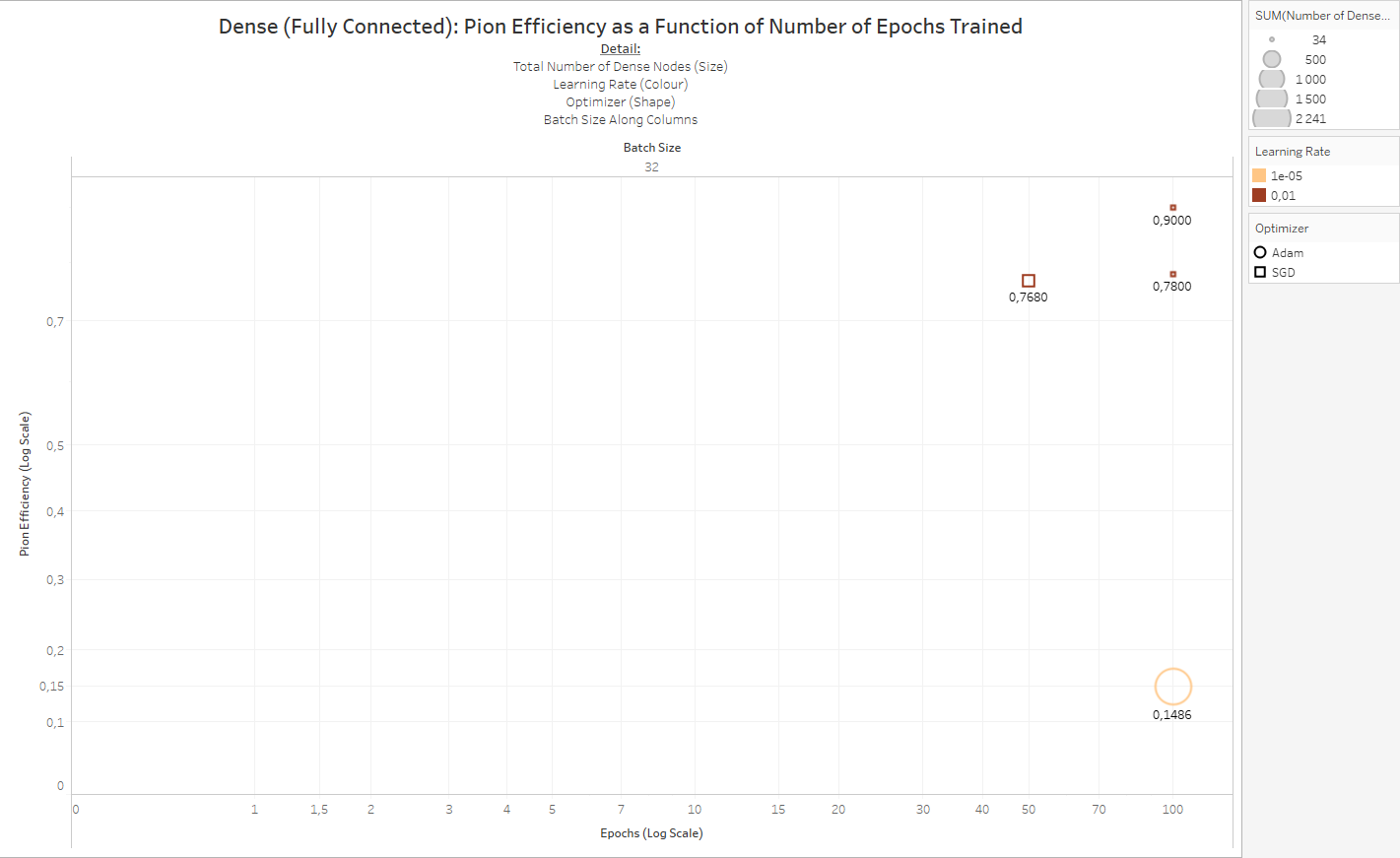


Figure 45



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 [29].

## A Brief History of Atomic Theory

The earliest 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 by stating 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 a – subsequently proven to be incorrect – theory for subatomic structure, in which negatively charged electrons were embedded amongst positive charges within an 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 surrounded by an electron shell.

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 additions 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; which resulted in the concept of isotopes (atoms with the same number of protons, but differing numbers of neutrons) being proposed. In the same year, Cockroft and Walton split the atom for the first time, by bombarding Lithium atoms with protons, 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.

(an electron Volt, eV, is a unit of energy, equivalent to the amount of work required to accelerate a single electron through a potential difference of 1 Volt),

## LHC Runs Used

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

ana.C (<https://github.com/PsycheShaman/trdML-gerhard/blob/master/ana.C>), 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.

, i.e.

* The run-number and event number which the track was obtained from
* The Track ID, identifying the primary vertex 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 and -211 for positively and negatively charged pions, respectively
* n-σ electron and n-σ 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
* dEdX estimate from the TPC
* 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

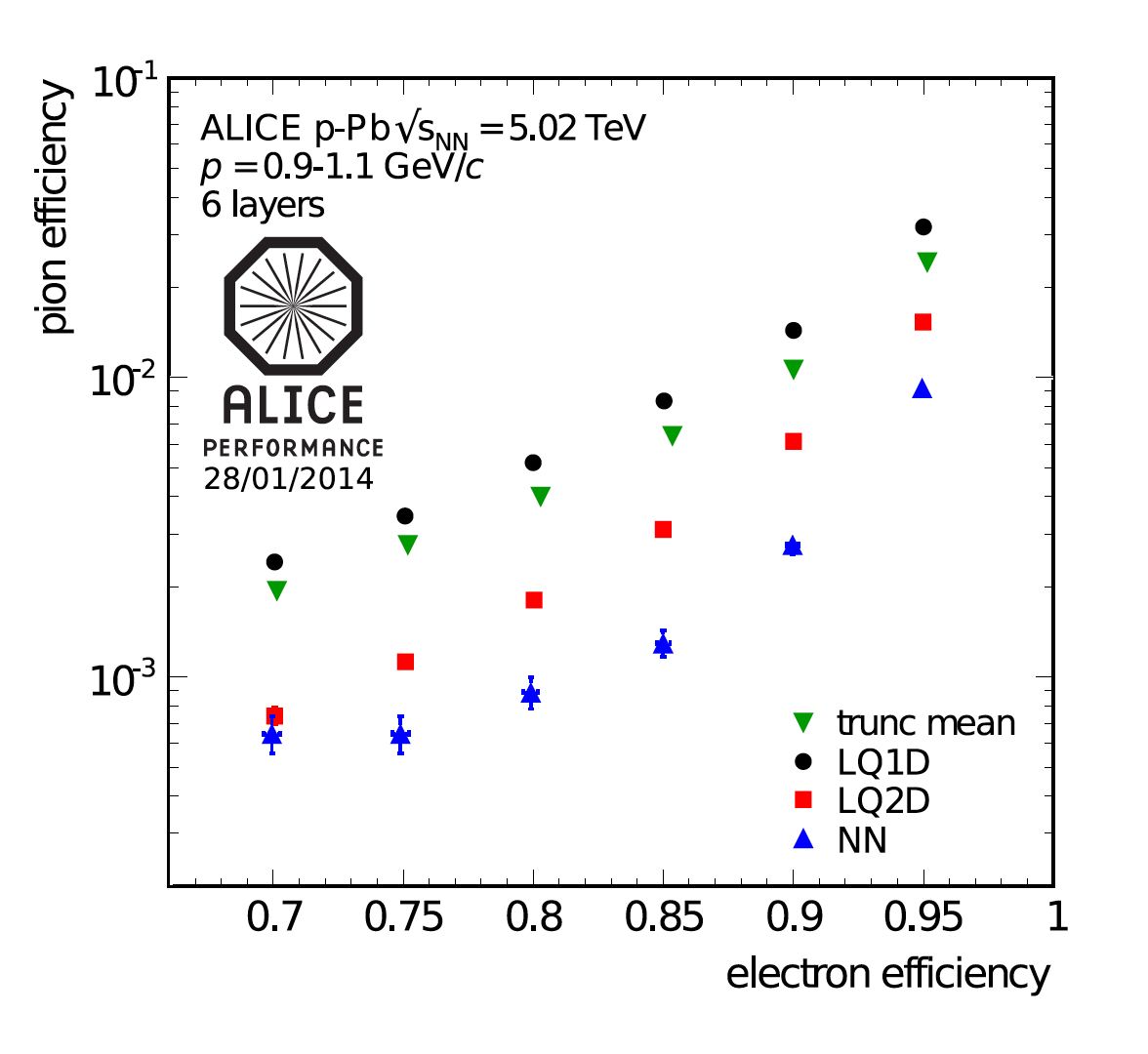


Figure 49: Pion efficiency as a function of electron efficiency for the various particle identification methods discussed [30].

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 = 7 TeV at ALICE, provides reference distribution data used for particle identification in the ALICE TRD (see Figure 50 for a plot showing the most probable signal dependence on βγ [30]).

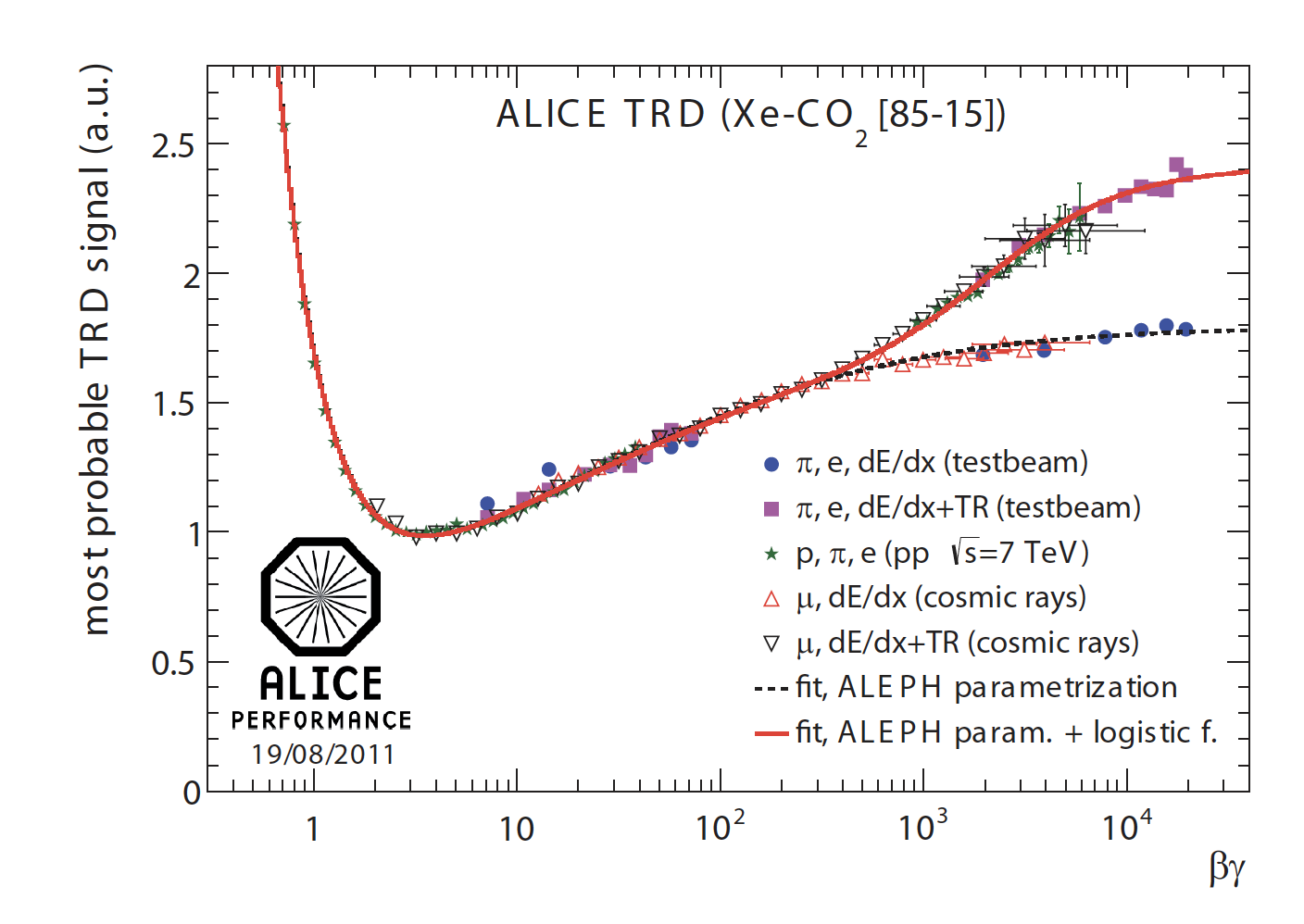


Figure 50: Reference distributions for most probable signal dependence on βγ in the TRD, i.e. from measurements taken in pp-runs, test beams and measurements from cosmic rays [30].

A commonly used nonlinear transformation , or activation function, in modern deep learning algorithms is the rectified linear unit (the ReLU function), which is simply an affine transformation, of the form [33].

##### 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 [33].

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 [33].

##### 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 , 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 . The learning rate is often set to decay linearly until iteration τ, i.e.

where . After iteration , is commonly leaved constant [33].

##### 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 , determines the rate of decay of previous gradients in determining this so-called momentum [33].

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

Given:

* Learning rate
* Momentum decay parameter
* Initial velocity **v**
* Initial parameter to be updated

While stopping criteria is unmet, DO:

1. Sample minibatch of size m from training data
2. Compute gradient:
3. Update velocity:
4. Update parameter:

Commonly used values of are 0.5, 0.9 and 0.99, and similarly to the learning rate, can also be adapted over time [33].

##### Nesterov Momentum

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

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 [33].

Given:

* Learning rate
* Momentum decay parameter
* Initial velocity **v**
* Initial parameter to be updated

While stopping criteria is unmet, DO:

1. Sample minibatch of size m from training data
2. Apply interim update:
3. Compute interim gradient:
4. Update velocity:
5. Update parameter:

##### 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 α, whereas those with smaller partial derivatives of the objective function decrease their learning rate much more slowly [33].

###### 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, [33].

#### Optimization

The essential optimization objective in deep learning is to find the optimal set of hyperparameters to minimize the objective function [33].

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 [33].

###### 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 [33].

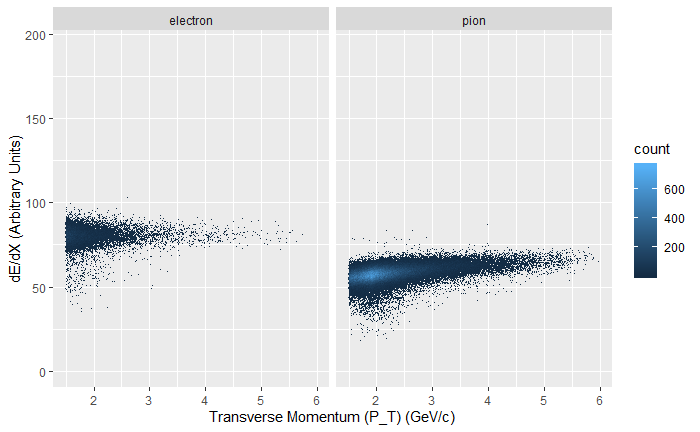


Figure 51: Don't know if I should rather keep this graph, it looks cleaner, there's a weird bump in the 2GeV range when looking at P instead of P\_T

### Exploratory Data Analysis

Some additional EDA follows, to look for interesting features in the dataset.

##### Mean pixel Value per TRD Layer

Figure 52 shows how the mean pixel value decreases from the innermost to outermost layers, a first indication that calibration would have increased the pion efficiencies obtained in this project. Much more granular calibration tactics were employed in earlier work done in this area, including chamber gain calibration on a run-by-run basis, as well as pad-by-pad calibration. Since the measurement mechanism employed during data-taking is an analog-to-digital transformation using an immense array of extremely sensitive sensors, the signal obtained from different detector layers and different pads, during different environmental conditions across a long time period, should ideally not be treated as coming from the same measuring device, since these devices will each have their own underlying signal distribution, which was not fed to the models trained in this project.

A form of calibration which feeds some of the above factors of variation as one-hot encoded variables to neural networks for both particle identification and deep generative modelling was investigated, but this meta-information alone explodes to upwards of 42GiB when represented in this fashion. The possibility of encoding this meta-information to a lower dimension using Autoencoders was explored, but the computational cost was deemed to be too much effort for potential pay-off, since there are not equal amounts of signals that originated from each combination of detector, layer and pad, for the neural network to learn useful representations from this information without overfitting, especially to spurious patterns from pads which have produced very little data.

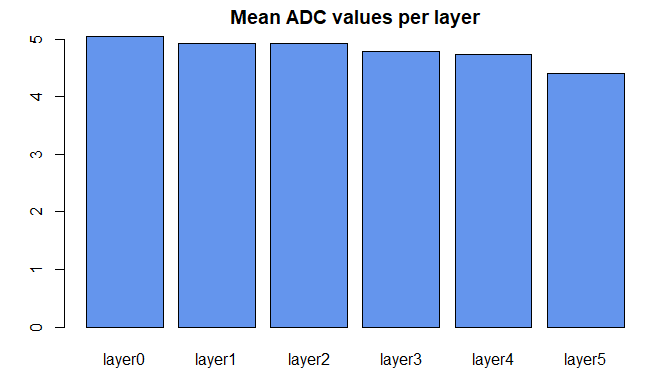


Figure 52: Mean pixel Values per TRD layer, for all runs

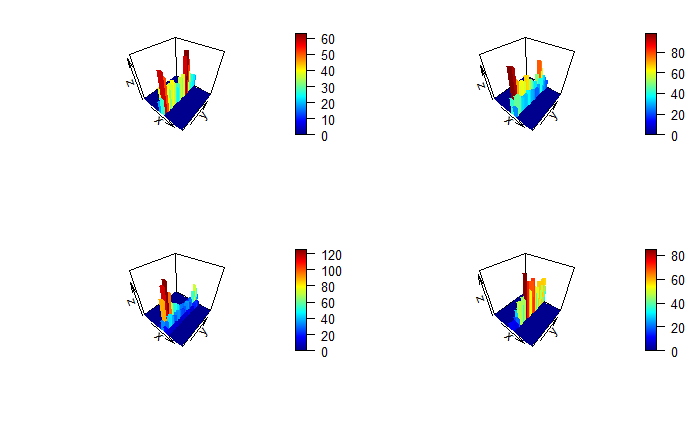


Figure 54: 3D Histograms of Four Randomly Sampled Geant4 Pion Tracklets

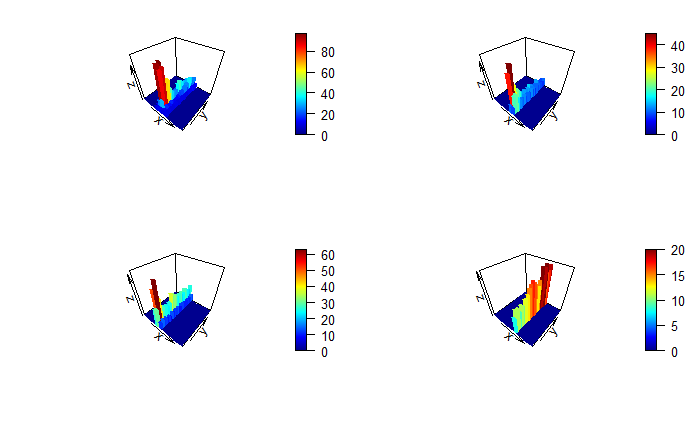


Figure 55: 3D Histograms of Four Randomly Sampled Real Pion Tracklets

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).

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

### GANs Stage 2

#### GAN “Hacks”

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:

Over the next few pages, an assessment of the usefulness of various model architectures and hyperparameter settings developed during this stage is conducted at the hand of plots.

### 2D Convolutional Neural Networks

All 2D Convolutional Neural Networks developed for Particle Identification are summarised in **Error! Reference source not found.**, Figure 83, Figure 84, Figure 85 and Figure 86.

It is immediately apparent from **Error! Reference source not found.** that

From **Error! Reference source not found.**, one can see that

, but this brings us to the point of why pion rejection results in this thesis are not comparable to previous work done on Particle Identification.

The data used in this project was not corrected for chamber gain, nor was pad-by-pad calibration performed on the raw dataset. These two essential steps explain the hard limit on accuracy/ pion efficiency, which seems to be at around 75% accuracy an 1-2% pion efficiency; therefore spending even more time on this phase of the project would be futile, since results on raw data will never compare with results from models trained on properly calibrated data.

A

### Particle Identification Using Deep Learning

#### Accuracy Paradox

As mentioned, when extremely large class imbalances are not accounted for, misleadingly good results may appear to occur during training (see Figure 56), but this level of accuracy just reflects the ratio of the classes in the dataset we’re working with (the model learns that it gets the lowest loss when it favours the prediction of pion, since there are so many more pions in our dataset, compared to electrons). This phenomenon is known as the “Accuracy Paradox”.

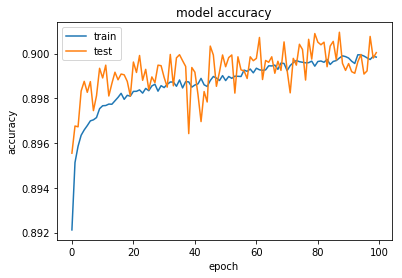


Figure 56: Accuracy Paradox

#### 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 contain about the target feature .

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

* which architecture was used (Figure 57, Figure 58 and Figure 59 show how this problem is potentially solvable by 2D Convolutional, LSTM and 1D Convolutional Neural Networks)

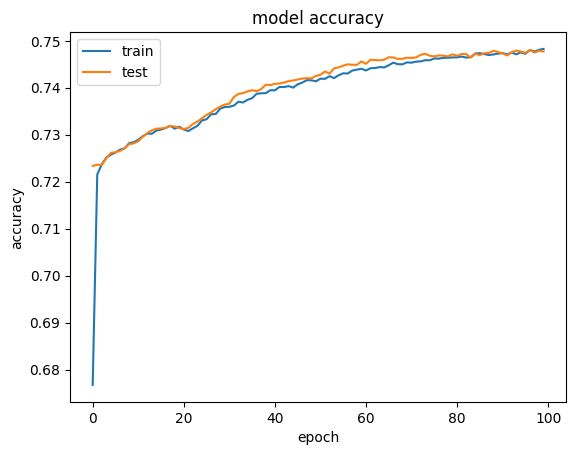


Figure 57: Example of a 2D-Convolutional Network training to high validation accuracy

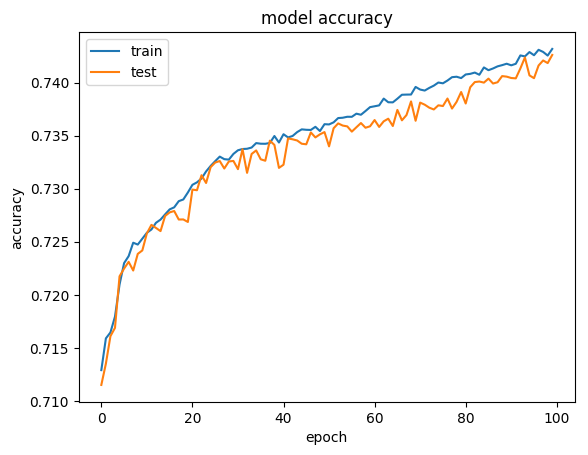


Figure 58: Example of an LSTM Network training to high validation accuracy

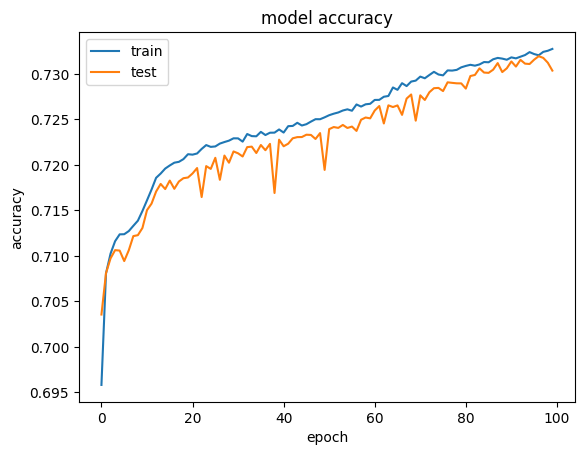


Figure 59: Example of a 1D-Convolutional Neural Network training to high validation accuracy

* the amount of epochs used for training (Figure 60 shows how training a highly successful model for twice the number of epochs only results in eventual overfitting)

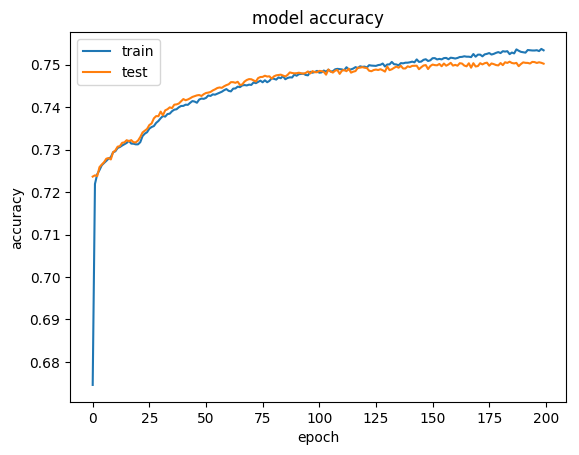


Figure 60: Running a successful model for twice the number of epochs results in minimal gains and eventually, results in overfitting

#### 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. and a single dense layer with 12 nodes) still gave comparable results, i.e. .

#### 1D Convolutional Neural Networks

, but the fact that 1D CNNs gave comparable accuracy provides additional suspicion 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).

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 .

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

## Conclusions

### 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.

### Simulations

Probably the most important result of this dissertation is the fact that Geant4 simulations are easily distinguishable from real data. Whilst deep generative techniques do not appear to be able to give similar performance compared to Geant4 for this particular problem, there could be interesting ways to combine the two methods, e.g. by transforming the output given by Geant4 with deep generative techniques to make it appear more like real data.

## Outlook and Future Work

### 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

### Simulations

As mentioned in the Conclusions section above, there should be scope to use the output of Geant4 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.

### 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.

After applying the classification based on , the histograms for the 6-tracklet estimate, for both electrons and pions is shown below:

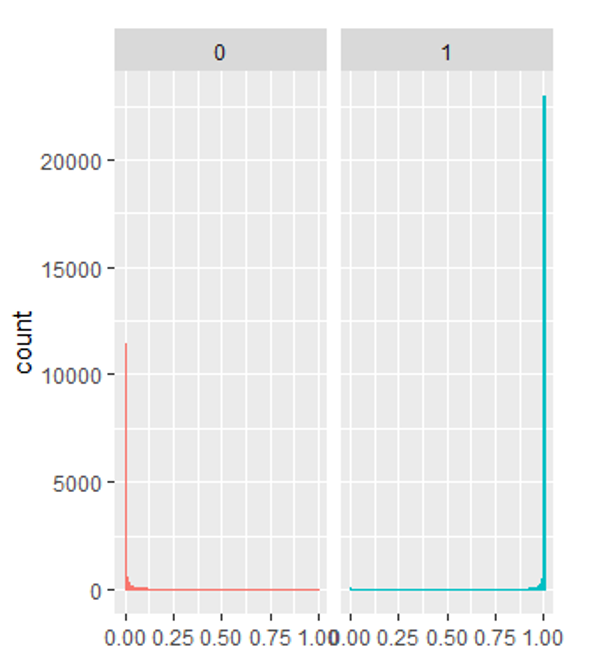


Figure 71: Combined output probabilities for true electrons (1, blue) and true pions (0, red)

for example, binary cross entropy:

, where is the model’s estimate for the probability of an observation of being of a particular class [33].

Figure 72 shows how, as approaches the true (in this binary classification example, ), the binary cross entropy loss function approaches 0.



Figure 72: Illustration of the descent towards zero, of the Binary Cross Entropy Loss Function as ŷ, or , approaches the true y.

As can be seen in Figure 73, there is a decreasing number of signals returned per layer, moving outwards from the innermost layers

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 |
| PbPb | Lead-Lead Collisions |
|  | Electron |
|  | Pion |
| QED | Quantum Electrodynamics |
| p | Proton |
| n | Neutron |
|  | Electron Neutrino |
|  | Muon Neutrino |
|  | Tau Neutrino |
| LSTM | Long Short-Term Memory |
| VAEs | Variational Autoencoders |
| GANs | Generative Adversarial Networks |
| nσ-electron | The TPC’s estimate for how many standard deviations away from the expected signal for an electron a particle is |
| eV | Electron Volt |
| u | Up quark |
| d | Down quark |
| s | Strange quark |
| c | Charm quark |
| b | Bottom quark |
| t | Top Quark |
|  | Muon |
|  | Tau lepton |
| EWT | Electroweak Theory |
| c | The speed of light |
|  | Vacuum expectation value |
| QFT | Quantum Field Theory |
| g | Coupling strength of standard model interaction vertices |
|  | Characteristic mean lifetime of subatomic particles |
| HEP | High Energy Physics |
| fm | Femtometer |
| s | Seconds |
| T | Temperature |
| UNESCO | United Nations Educational, Scientific and Cultural Organization |
| TB | Terabytes |
| RAM | Random Access Memory |
| GiB/s | Gigabytes per second |
| RHIC | Relativistic Heavy Ion Collider |
| Z | Atomic number |
|  | Hydrogen |
|  | Centre of Mass Energy |
| p | Momentum |
| pPb | Proton-Lead Collisions |
| Pb | Lead |
| PS | Proton Synchrotron |
| SPS | Super Proton Synchrotron |
| NbTi | Niobium-titanium |
| K | Degrees Kelvin |
| °C | Degrees Celcius |
| T | Tesla |
|  | Vacuum Pressure |
|  | Atmosphere |
| ATLAS | A Toroidal LHC Apparatus |
| CMS | Compact Muon Solenoid |
| LHCb | Large Hadron Collider beauty |
| TOTEM | The TOTal cross section, Elastic scattering and diffraction dissociation Measurement at the Large Hadron Collider |
| MoEDAL | The Monopole & Exotics Detector at the LHC |
| OOP | Object Oriented Programming |
| GNU | Gnu's Not Unix |
| OS | Operating System |
|  | Critical Temperature |
| m | Meter |
| ITS | Inner Tracking System |
| SPD | Silicon Pixel Detectors |
| SDD | Silicon Drift Detectors |
| SSD | Silicon Strip Detectors |
| dE/dx | Energy loss per unit pathlength |
|  | Transverse Momentum |
| TPC | Time Projection Chamber |
| TOF | Time of Flight |
| m² | Square meters |
| ps | Picoseconds |
| HMPID | Ring Imaging Cherenkov Detectors |
|  | Xenon - Carbon Dioxide Gas |
| PHOS | Photon Spectrometer |
| EmCal | Electromagnetic Calorimeter |
|  | Lead Tungstate |
| V0 | ALICE V0 Detector |
| T0 | ALICE Fast timing and trigger detector |
| PMD | Photomultiplicity detector |
| FMD | Forward multiplicity detector |
| ZDC | Zero Degree Calorimeters |
| cm | Centimetre |
|  | Pseudorapidity |
| MWPC | Multi-wire Proportional Chamber |
| ns | Nanoseconds |
| ADC | Analog to digital converter |
| γ | Relativistic Factor |
| LQ1D | One-dimensional likelihood |
| LQ2D | Two-dimensional likelihood |
| AI | Artificial Intelligence |
| MLPs | Multilayer Perceptrons |
| ReLU | Rectified Linear Unit |
| J | Objective function |
| SGD | Stochastic Gradient Descent |
|  | Learning rate at iteration i |
|  | Momentum decay parameter |
| v | Velocity |
|  | Parameter set |
|  | gradient |
|  | Interim parameter update |
|  | Length scale |
| Adam | Adaptive Moments |
| CNN | Convolutional Neural Network |
| RNN | Recurrent Neural Network |
| ANN | Artificial Neural Network |
|  | State of a dynamical system at timestep t |
| Z | Latent space |
| I | Identity matrix |
|  | Kullback-Leibler divergence |
|  | Mean |
|  | Multidimensional standard deviation |
| σ | Standard deviation |
| D | Discriminative Neural Network |
| G | Generative Neural Network |
| BiGANs | Bidirectional Generative Adversarial Networks |
| LSGANs | Least Squares Generative Adversarial Networks |
|  | Null Hypothesis |
| t | Test statistic |
|  | Threshold value for test statistic |
|  | Significance level |
|  | Power |
|  | Electron Efficiency |
|  | Pion Efficiency |
| PDG | Particle Data Group |
|  | Primary vertex |
|  | Skewness |
|  |  |

Again, it needs to be stressed that the poorer performance of particle identification obtained during this project is most probably fully accounted for by the quality of data was used here, in comparison to pad-by-pad calibrated data fed to much simpler neural fully connected neural networks in the previous work done.

List of Tables

[Table 1: The twelve fundamental fermions. **Error! Bookmark not defined.**](#_Toc19377333)

[Table 2: Confusion Matrix for Particle Identification 142](#_Toc19377334)

[Table 3 136](#_Toc19377335)

[Table 4: Confusion Matrix for distinguishing between Geant4 vs Real Data 155](#_Toc19377336)

List of Figures

[Figure 1: 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 [3], reproduced and modified from [4]. **Error! Bookmark not defined.**](#_Toc19377337)

[Figure 2: Phase diagram of hadronic matter [5] **Error! Bookmark not defined.**](#_Toc19377338)

[Figure 3: The evolution of the Universe, from the Big Bang to Modern Day [7] **Error! Bookmark not defined.**](#_Toc19377339)

[Figure 4: CERN facilities in geographical context [15]. **Error! Bookmark not defined.**](#_Toc19377340)

[Figure 5: 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 [17] 154](#_Toc19377341)

[Figure 6: The CERN accelerator complex [19]. **Error! Bookmark not defined.**](#_Toc19377342)

[Figure 8: The ALICE detector system [34] **Error! Bookmark not defined.**](#_Toc19377343)

[Figure 7: Bethe-Bloch curve for various subatomic particles as measured by the ALICE TPC at **Error! Bookmark not defined.**](#_Toc19377344)

[Figure 9: A schematic representation of the components in an MWPC module **Error! Bookmark not defined.**](#_Toc19377345)

[Figure 19: Filled contour map of a single pion tracklet’s signal, with pads along the *x* axis (columns) and timebins along the *y* axis (rows) 96](#_Toc19377346)

[Figure 20: 3D surface plot of the signal of a single pion tracklet's signal, with timebins along *x*, pads along *y* and pulse height along *a*. 96](#_Toc19377347)

[Figure 21: 3D histogram of a single electron tracklet's signal, with pads along *x*, timebins along *y* and pulse height along *z*. 96](#_Toc19377348)

[Figure 22: Number of Entries per TRD Layer Number, for all Runs 125](#_Toc19377349)

[Figure 23: Number of Particles, per Particle ID, across all runs **Error! Bookmark not defined.**](#_Toc19377350)

[Figure 24: Number of Particles (electrons and pions) in each of a set of defined momentum bins **Error! Bookmark not defined.**](#_Toc19377351)

[Figure 25: Energy Loss per Unit Path Length as a function of Momentum, for Electrons and Pions **Error! Bookmark not defined.**](#_Toc19377352)

[Figure 26: Don't know if I should rather keep this graph, it looks cleaner, there's a weird bump in the 2GeV range when looking at P instead of P\_T 101](#_Toc19377353)

[Figure 27: Time Evolution of the Average Pulse Height Signal, per Particle ID (for tracklets from the entire momentum range) **Error! Bookmark not defined.**](#_Toc19377354)

[Figure 28: Pulse height as a function of time for 4 randomly sampled electrons **Error! Bookmark not defined.**](#_Toc19377355)

[Figure 29: Pulse height as a function of time for 4 randomly sampled pions **Error! Bookmark not defined.**](#_Toc19377356)

[Figure 30: Mean pixel Values per TRD layer, for all runs 102](#_Toc19377357)

[Figure 31: Time evolution of the TRD signal, measured as pulse height vs drift time for electrons and pions (both at p = 2GeV) [37]. **Error! Bookmark not defined.**](#_Toc19377358)

[Figure 32: 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 [37]. 97](#_Toc19377359)

[Figure 33: Truncated mean signal (dE/dx + TR) for various charged particles as measured for p-Pb collisions at . This method allows for particle identification of light particles and hadrons [37]. **Error! Bookmark not defined.**](#_Toc19377360)

[Figure 34: Normalised distribution of charge deposition for electrons and pions in a single TRD chamber [37]. **Error! Bookmark not defined.**](#_Toc19377361)

[Figure 35: Pion efficiency as a function of electron efficiency for the various particle identification methods discussed [37]. 97](#_Toc19377362)

[Figure 36: Momentum dependence of pion efficiency for various methods (where electron efficiency is at 90%) **Error! Bookmark not defined.**](#_Toc19377363)

[Figure 10: Illustration of the descent towards zero, of the Binary Cross Entropy Loss Function as ŷ, or , approaches the true y. 124](#_Toc19377364)

[Figure 11: 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) [41]. 103](#_Toc19377365)

[Figure 12: Illustration of mathematical equivalence of implementing a convolution with unit stride followed by downsampling to implementing a convolution with stride = 2. 153](#_Toc19377366)

[Figure 18: An illustration of rejection or acceptance of the null hypothesis, under the assumed distributions of and , when t falls in the critical region **Error! Bookmark not defined.**](#_Toc19377367)

[Figure 42 **Error! Bookmark not defined.**](#_Toc19377368)

[Figure 43 143](#_Toc19377369)

[Figure 44 144](#_Toc19377370)

[Figure 45 145](#_Toc19377371)

[Figure 46 146](#_Toc19377372)

[Figure 58: Particle Identification Model Architecture 140](#_Toc19377373)

[Figure 59: Training vs Validation Accuracy 141](#_Toc19377374)

[Figure 60: Training vs Validation Loss 141](#_Toc19377375)

[Figure 61: t-statistic selection allowing for 90% electron efficiency 142](#_Toc19377376)

[Figure 62: Combined output probabilities for true electrons (1, blue) and true pions (0, red) 124](#_Toc19377377)

[Figure 67: Focal Loss where true class is 1. **Error! Bookmark not defined.**](#_Toc19377378)

[Figure 68 134](#_Toc19377379)

[Figure 69 134](#_Toc19377380)

[Figure 70 135](#_Toc19377381)

[Figure 71 136](#_Toc19377382)

[Figure 72 137](#_Toc19377383)

[Figure 73: Summary of incrementally trained (with uncalibrated dataset, partitioned by increasing momentum range) Convolutional Neural Network **Error! Bookmark not defined.**](#_Toc19377384)

[Figure 74: Summary of previous work done on Particle Identification using calibrated data from the TRD **Error! Bookmark not defined.**](#_Toc19377385)

[Figure 13: Training-time VAE **Error! Bookmark not defined.**](#_Toc19377386)

[Figure 14: Training-time VAE with reparameterization trick to enable backpropagation **Error! Bookmark not defined.**](#_Toc19377387)

[Figure 15: Testing time VAE **Error! Bookmark not defined.**](#_Toc19377388)

[Figure 16: Gan Densities during training, close to convergence, P(x) is shown in black, G(z) in blue and D(G(z)) in red **Error! Bookmark not defined.**](#_Toc19377389)

[Figure 17: Gan Densities during training, once the Algorithm has converged, G(z) matches P(x) perfectly and D(G(z)) outputs 0.5 everywhere **Error! Bookmark not defined.**](#_Toc19377390)

[Figure 75: Eta distributions for real and Geant4 simulated data 148](#_Toc19377391)

[Figure 76: estimate (TPC) distributions for real and Geant simulated data, before cut 148](#_Toc19377392)

[Figure 77: nsigma pion distributions after applying cut 149](#_Toc19377393)

[Figure 78: Momentum distributions for both real and Geant4 simulated data, before cut 149](#_Toc19377394)

[Figure 79: Momentum distributions after cut 149](#_Toc19377395)

[Figure 80: Training loss and accuracy curves for training and validation data 155](#_Toc19377396)

[Figure 81: Model architecture for distinguishing real from Geant4-simulated data **Error! Bookmark not defined.**](#_Toc19377397)

[Figure 82: Distribution of ADC values for simulated data 150](#_Toc19377398)

[Figure 83: Distribution of ADC values for real data 150](#_Toc19377399)

[Figure 84: Distribution of the number of pads with no data per image for simulated data 151](#_Toc19377400)

[Figure 85: Distribution of the number of pads with no data per image for real data 151](#_Toc19377401)

[Figure 86: Distribution of mean ADC value per image for simulated data 151](#_Toc19377402)

[Figure 87: Distribution of mean ADC value per image for real data 152](#_Toc19377403)

[Figure 88: 3D Histograms of Four Randomly Sampled Geant4 Pion Tracklets 103](#_Toc19377404)

[Figure 89: 3D Histograms of Four Randomly Sampled Real Pion Tracklets 104](#_Toc19377405)

[Figure 90: Average Pulse-height as a function of time-bin for Real and Geant4-simulated pion tracklets 152](#_Toc19377406)

[Figure 91: Encoder **Error! Bookmark not defined.**](#_Toc19377407)

[Figure 92: Decoder **Error! Bookmark not defined.**](#_Toc19377408)

[Figure 93: Four examples of simulated data created using a Variational Autoencoder iii](#_Toc19377409)

[Figure 94: Training accuracy and loss curves for training vs validation data 156](#_Toc19377410)

[Figure 95: Accuracy Paradox 114](#_Toc19377411)

[Figure 96: Example of a 2D-Convolutional Network training to high validation accuracy 115](#_Toc19377412)

[Figure 97: Example of an LSTM Network training to high validation accuracy 115](#_Toc19377413)

[Figure 98: Example of a 1D-Convolutional Neural Network training to high validation accuracy 115](#_Toc19377414)

[Figure 99: Running a successful model for twice the number of epochs results in minimal gains and eventually, results in overfitting 116](#_Toc19377415)

[Figure 100: No Dropout vs Same Model with too much Dropout 117](#_Toc19377416)

[Figure 101: Gaussian Noise with σ=0.2 117](#_Toc19377417)

[Figure 102: Real images 118](#_Toc19377418)

[Figure 103: Autoencoder outputs 119](#_Toc19377419)

[Figure 104: An illustration of Mode Collapse 119](#_Toc19377420)

[Figure 105: Illustrating the effect of Batch normalization to prevent output images from looking highly similar 120](#_Toc19377421)

[Figure 106: 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 121](#_Toc19377422)

[Figure 107: Least squares GAN after 94600 epochs 121](#_Toc19377423)

[Figure 108: Least squares GAN after 387600 epochs 122](#_Toc19377424)

[Figure 109: Least squares GAN after 449000 epochs 122](#_Toc19377425)

**Error! Reference source not found.** summarises the results from the final stage of model-building, in order to allow for comparison with work that had been done in the past on particle identification using the TRD (for that reason **Error! Reference source not found.**, which was also shown in Section **Error! Reference source not found.** is repeated here for comparison).

## The Structure & Organization of this Dissertation

Chapter **Error! Reference source not found.**: **Error! Reference source not found.** 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 understanding the data used in this thesis, followed by a quick overview of the Quark Gluon Plasma. Next, the European Organization for Nuclear Research (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 packages for data analysis (ROOT) and simulation (Geant4).

Chapter **Error! Reference source not found.**: **Error! Reference source not found.** 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 **Error! Reference source not found.**: **Error! Reference source not found.** introduces Deep Learning within the larger context of Machine Learning and Artificial Intelligence, 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 **Error! Reference source not found.**: **Error! Reference source not found.** 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, which is important in understanding the Results section of this dissertation.

Chapter **Error! Reference source not found.**: 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 **Error! Reference source not found.** is the **Error! Reference source not found.** section of this dissertation.

Chapter **Error! Reference source not found.** is the **Error! Reference source not found.** section of this dissertation, where only the most successful results are discussed.

Chapter **Error! Reference source not found.** contains the **Error! Reference source not found.**, including a deeper analysis of results, as well as an outline of future work which can be done in this area.

### Discussion of Most Successful Model from Stage Two of Model Building

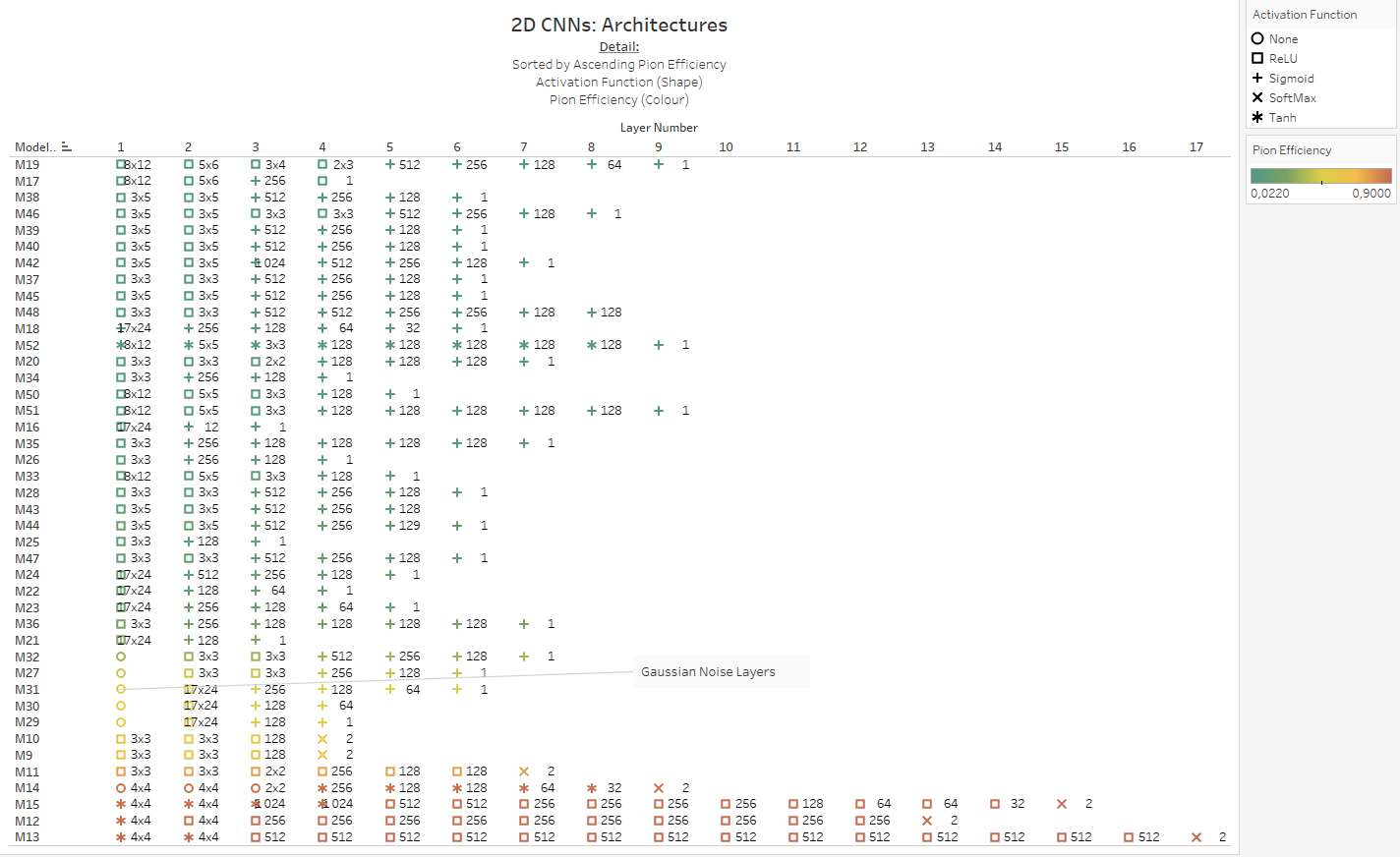


Figure 83

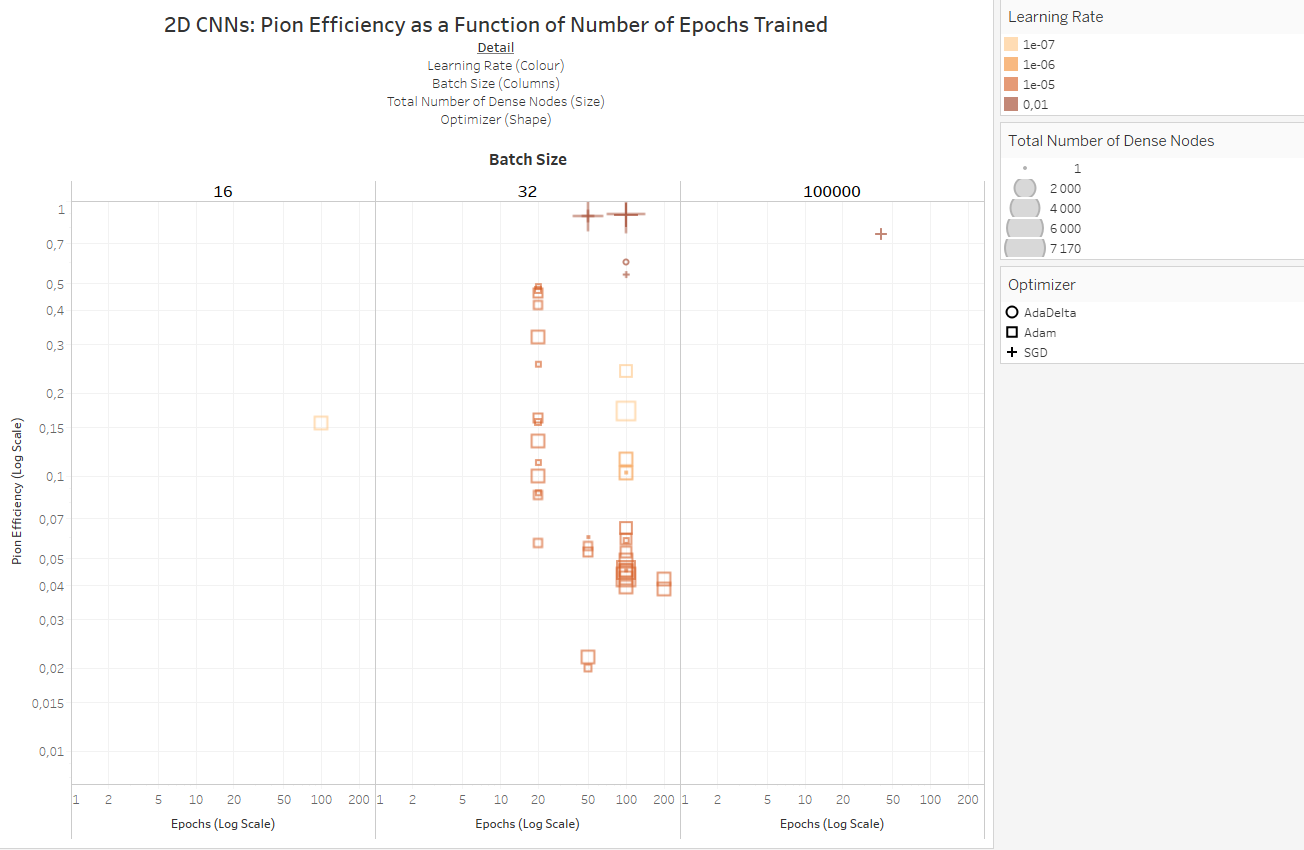


Figure 84

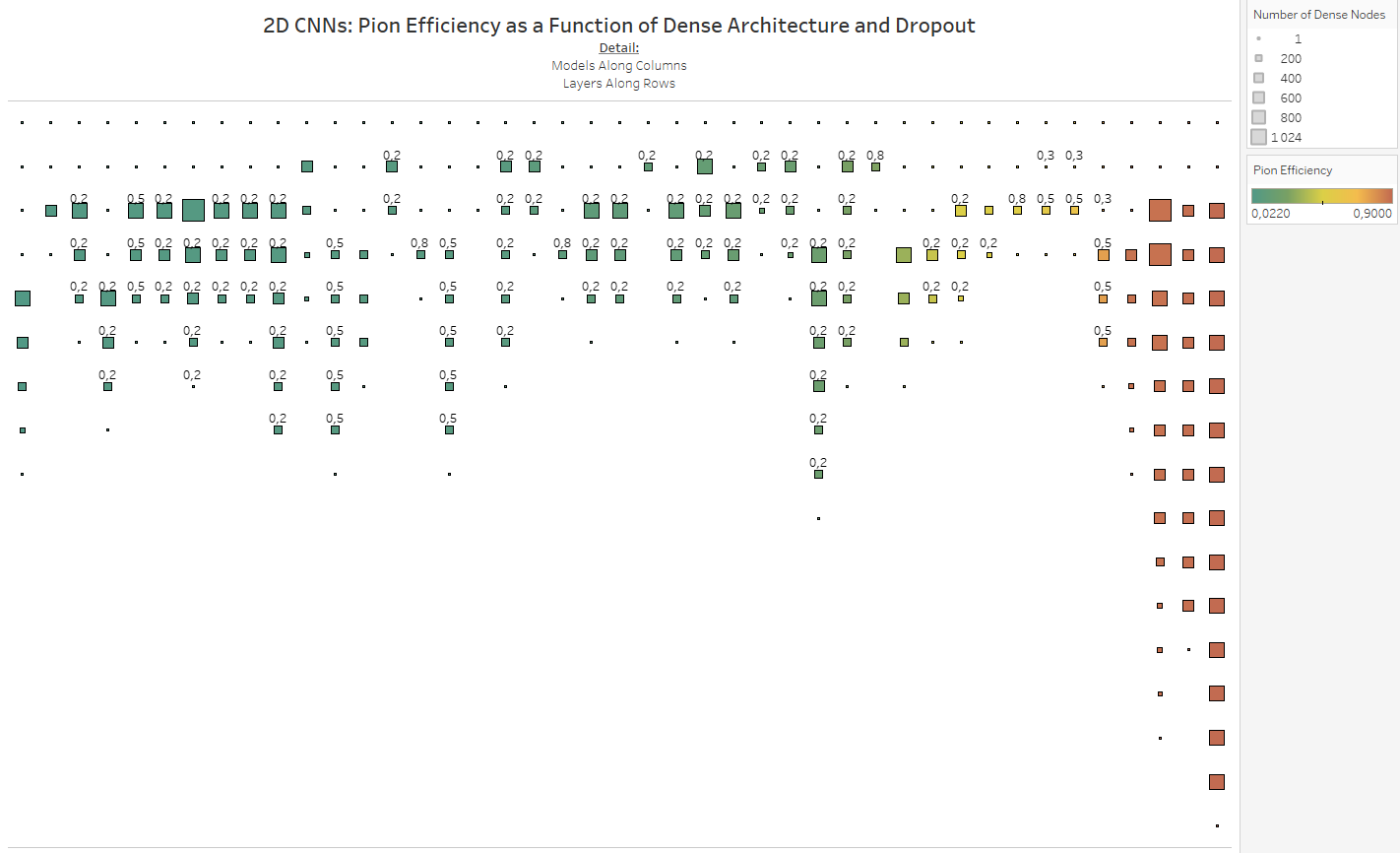


Figure 85

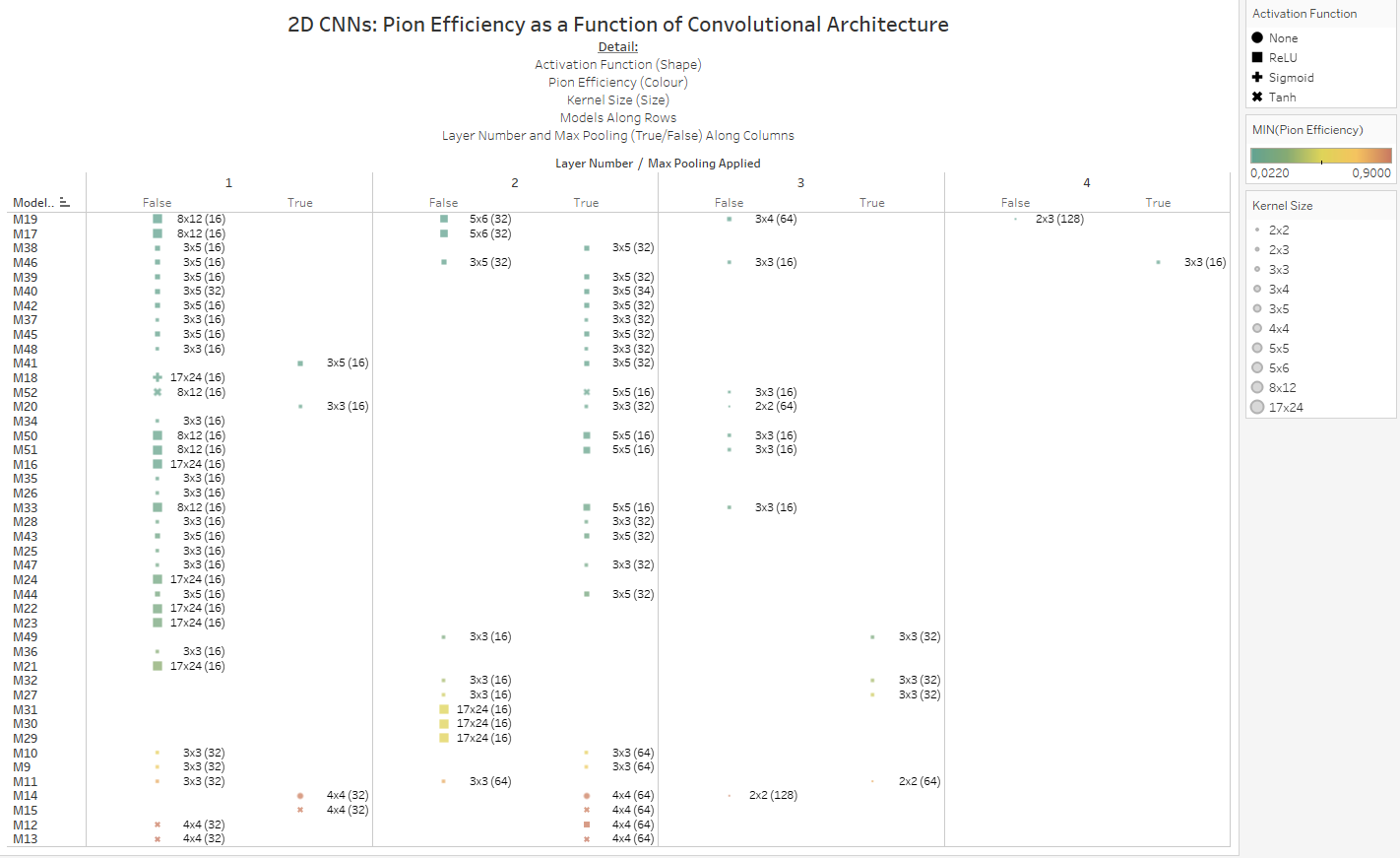


Figure 86

### 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 , is now a function of both a noise vector and the label of the image, i.e. , whereas 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 [43].

#### Adversarial Autoencoder

Adversarial Autoencoders match the aggregated posterior of the latent space vector from an autoencoder with an arbitrary prior distribution , 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 [44].

#### Bidirectional Generative Adversarial Network

Bidirectional Generative Adversarial Networks (BiGANs) make use of an encoder function which maps data into a latent feature space for the generative model, i.e.: . Here the discriminator has access to both the (simulated or real) , as well as its latent encoding 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 [45].

#### 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 [46].

#### 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 [47].

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:

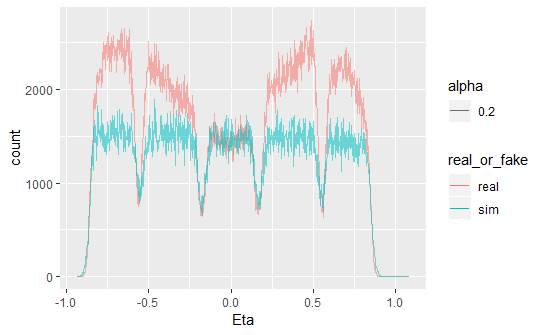


Figure 87: Eta distributions for real and Geant4 simulated data

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

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

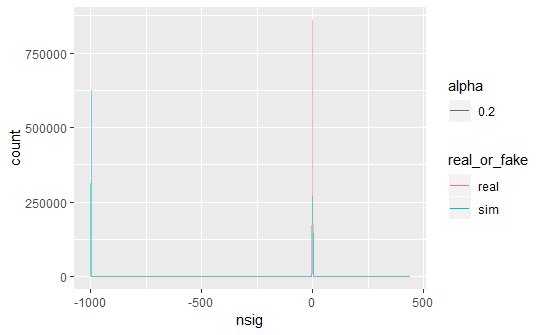


Figure 88: estimate (TPC) distributions for real and Geant simulated data, before cut

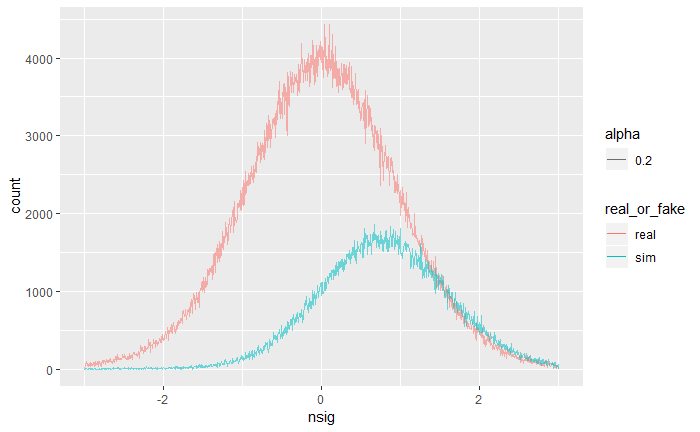


Figure 89: nsigma pion distributions after applying cut

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

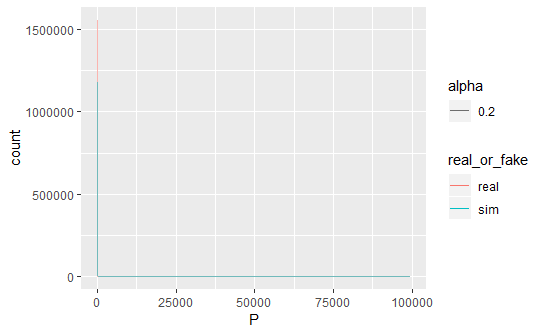


Figure 90: Momentum distributions for both real and Geant4 simulated data, before cut

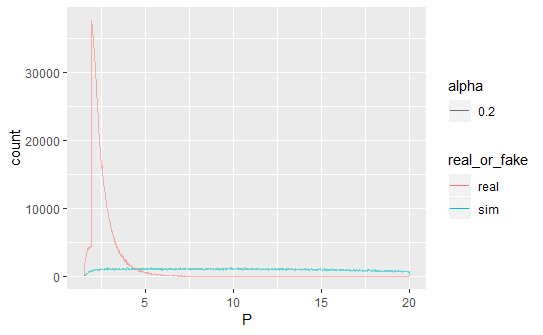


Figure 91: Momentum distributions after cut

Distinguishing Geant4 simulated data from real data proved to be trivial compared to distinguishing real pions from real electrons.

Figure 99 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 [34].

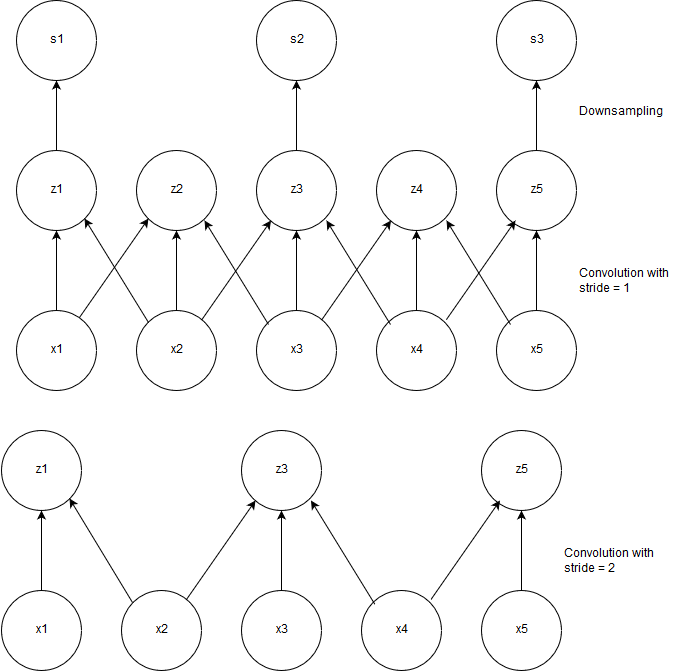


Figure 99: Illustration of mathematical equivalence of implementing a convolution with unit stride followed by downsampling to implementing a convolution with stride = 2.

A simplified diagram depicting this process can be seen in Figure 100.



Figure 100: 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 [13]

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

Geant4 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.

The task of distinguishing simulated from real data was performed using a 2D convolutional neural network, with architecture shown in Figure 101 (a).

Distinguishing Geant4 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 101 (b).

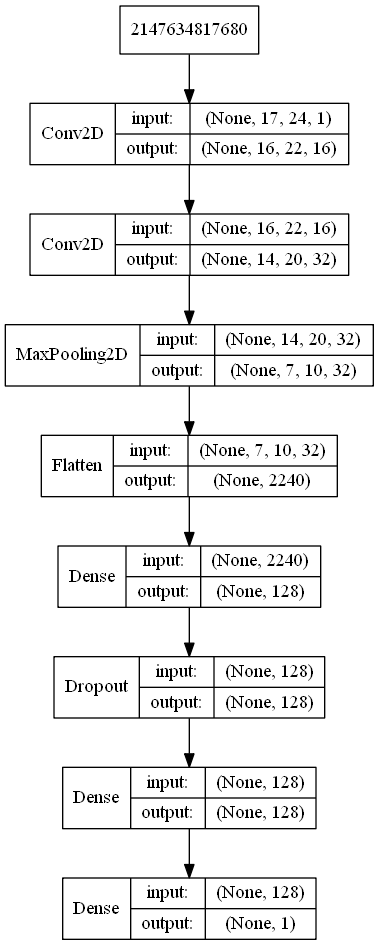
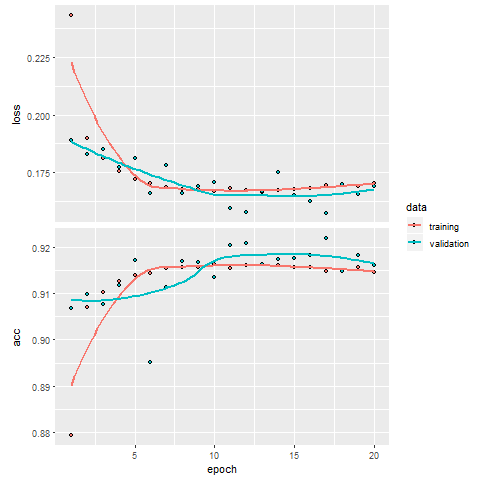
(a)  (b) 

Figure 101: Training curves (a) and model architecture (b) for distinguishing real from Geant4-simulated data

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

Table 3: Confusion Matrix for distinguishing between Geant4 vs Real Data

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

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 102.

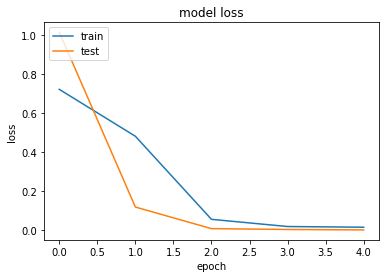
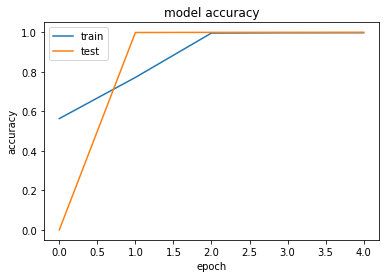


Figure 102: Training accuracy and loss curves for training vs validation data

In order to quantitatively compare the accuracy of the generative models built during the course of the project and Geant4 simulations, a Convolutional Neural Network was trained to 92% accuracy, with 75 000 observations of each type of simulated data combined labeled as 0 and an equivalent amount of real data (75 000 x 4) labeled as 1, used as the training set.

Each training set’s images were independently scaled to be in the range [-1;1].

Figure 103 shows the distribution of predictions made by this neural network on an independent test set of size 25 000 for each model type and 25 000 real observations.

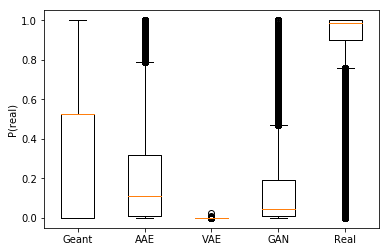


Figure 103: predictions made (by a neural network trained to discriminate 4 kinds of simulated data from real data) on an independent test set of each type of simulate dataset, as well as real observations. Most real observations received a prediction probability above 0.8, verifying that the model has decent statistical power to discriminate real observations from simulated data, although there appears to be a long tail of outliers that the model could not correctly identified as being real. By looking at this graph, it seems that VAE data was quite easily classified as simulated data (with no probability predictions reaching above 0.1) and can therefore be deemed as inaccurate, Geant4 data generally received higher probability predictions than the other methods of simulation with a median prediction score of above 0.5 and upper quartile reaching up to 1, but AAE and GAN data both have outlier observations which seem to have fooled this neural network to think they are real as well and can be deemed to be more accurate than VAE data, with AAE data’s output distribution slightly more skewed towards 1 than GAN data.

Table 4: For each simulated dataset, as well as real data: the tracklets which received the highest P(real) prediction from the neural network discussed above. This tells an entirely different story and shows that many of the Geant4 simulated tracklets are completely empty images (a phenomenon which *does* occur in real data, as discussed) and the top two observations appear qualitatively incorrect; although it is theoretically possible that such observations might have existed in the real dataset, it is probably because something went wrong. While it is still difficult to draw any absolute conclusions from looking at such a small sample, AAE data seems to be the only dataset whose generative procedure was capable of fully capturing the underlying data distribution. VAE data received consistently low probability scores and most samples look very similar, whereas the top GAN images appear to all be exactly the same image; while this indicates that the GAN managed to create at least one image which fools a discriminative network, it also shows that it failed to capture the underlying distribution as well as the AAE, which produced many observations which were hard to discriminate from real data, and which all look dissimilar.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | Geant | AAE | VAE | GAN | Real |
| 1 | P(real) = 1 | P(real) = 0.999 | P(real) = 0.026 | P(real) = 0.999 | P(real) = 1 |
| 2 | P(real) = 0.999 | P(real) = 0.999 | P(real) = 0.010 | P(real) = 0.999 | P(real) = 1 |
| 3 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.006 | P(real) = 0.999 | P(real) = 1 |
| 4 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.005 | P(real) = 0.999 | P(real) = 1 |
| 5 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.004 | P(real) = 0.999 | P(real) = 1 |
| 6 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.003 | P(real) = 0.999 | P(real) = 1 |
| 7 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.003 | P(real) = 0.999 | P(real) = 1 |
| 8 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.003 | P(real) = 0.999 | P(real) = 1 |
| 9 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.002 | P(real) = 0.999 | P(real) = 1 |
| 10 | P(real) = 0.526 | P(real) = 0.999 | P(real) = 0.002 | P(real) = 0.999 | P(real) = 1 |