# Appendices

[Appendix A: Plotting the Bethe-Bloch Equation 111](#_Toc14519599)

[Appendix B: Plotting Binary Cross-Entropy 118](#_Toc14519600)

[Appendix C: The Anatomy Of An AliROOT Analysis Task 119](#_Toc14519601)

[Appendix D: Software Environment, Packages & Utilities 124](#_Toc14519602)

[Appendix E: Running and Monitoring Root Analysis Tasks 126](#_Toc14519603)

[Appendix F: Data Extraction, Data Quality and Data Pre-processing 130](#_Toc14519604)

[Appendix G: Old Methods Section 132](#_Toc14519605)

[Appendix H: Old Results Section 136](#_Toc14519606)

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

### Utilities

#### Makefiles

all: gridfiles.md5

gridfiles.xml: query.sh

./$< > $@

gridfiles.md5: gridfiles.xml

xsltproc /alice/data/util/xml2md5.xsl $< > $@

download: $(shell cut -c 49- files.md5)

/alice/data/%:

mkdir -p $(dir $@)

alien\_cp alien:$@ file:$@

#### User specified aliases in ~/.bashrc

# User specific aliases and functions

alias initialize\_aliroot='/cvmfs/alice.cern.ch/bin/alienv enter VO\_ALICE@AliPhysics::vAN-20180902-1'

alias my\_alice='alienv -w /alice/gviljoen/alice/sw enter VO\_ALICE@AliPhysics::latest'

#### Remote Editing

#### Rsync

rsync -av --stats --progress --include="\*/" --include="\*.txt" --exclude="\*.C" --exclude="\*.cxx" --exclude="\*.h" --exclude="\*.root" --exclude="\*.ps" --exclude="\*.d" --exclude="\*.so" --exclude "\*.proc" gviljoen@hep01.phy.uct.ac.za:/alice/gviljoen/trdpid/adj\_sim/test .

X11 Forwarding

Atom packages:

* Remote Atom Server
* PlatformIO-IDE-Terminal

##### Killing a process being listened to on the remote port 52698:

List processes that are owned by me:

ps aux | grep gviljoen

Find the sshd process being listened to on port 52698 and kill it, by running:

kill -9 $processid

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: Running and Monitoring Root Analysis Tasks

Once one is happy with the analysis task defined, one first needs to enter AliPhysics, by using one of the user-defined aliases, e.g.:

initialize\_aliroot

Then, one gets a token from alien, to access the grid. This token will be valid for 24 hours. Since my CERN username is not the same as my username on HEP01, the command is:

alien-token-init username

Once the above commands have been run, one can run the analysis task on the grid, by setting the following parameters in the analysis macro (ana.C):

Bool\_t local = kFALSE;

Bool\_t gridTest = kFALSE;

Adding the appropriate run number and output directory:

alienHandler->AddRunNumber(265377);

alienHandler->SetGridWorkingDir("new-wd-momentum-test");

alienHandler->SetGridOutputDir("outDir265378");

Setting the run mode and starting the analysis:

alienHandler->SetRunMode("full");

//alienHandler->SetRunMode("terminate"); //this is run for merging stages

mgr->StartAnalysis("grid");

Assuming that one has added the appropriate CERN certificates, one can then view, manage and download the output of one’s jobs on the MonALISA grid monitoring site for ALICE see Figure 39 for an example screenshot of user job monitoring and Figure 40 for the user interface for viewing the directory structure for the ALICE grid, in particular the user’s working directory:

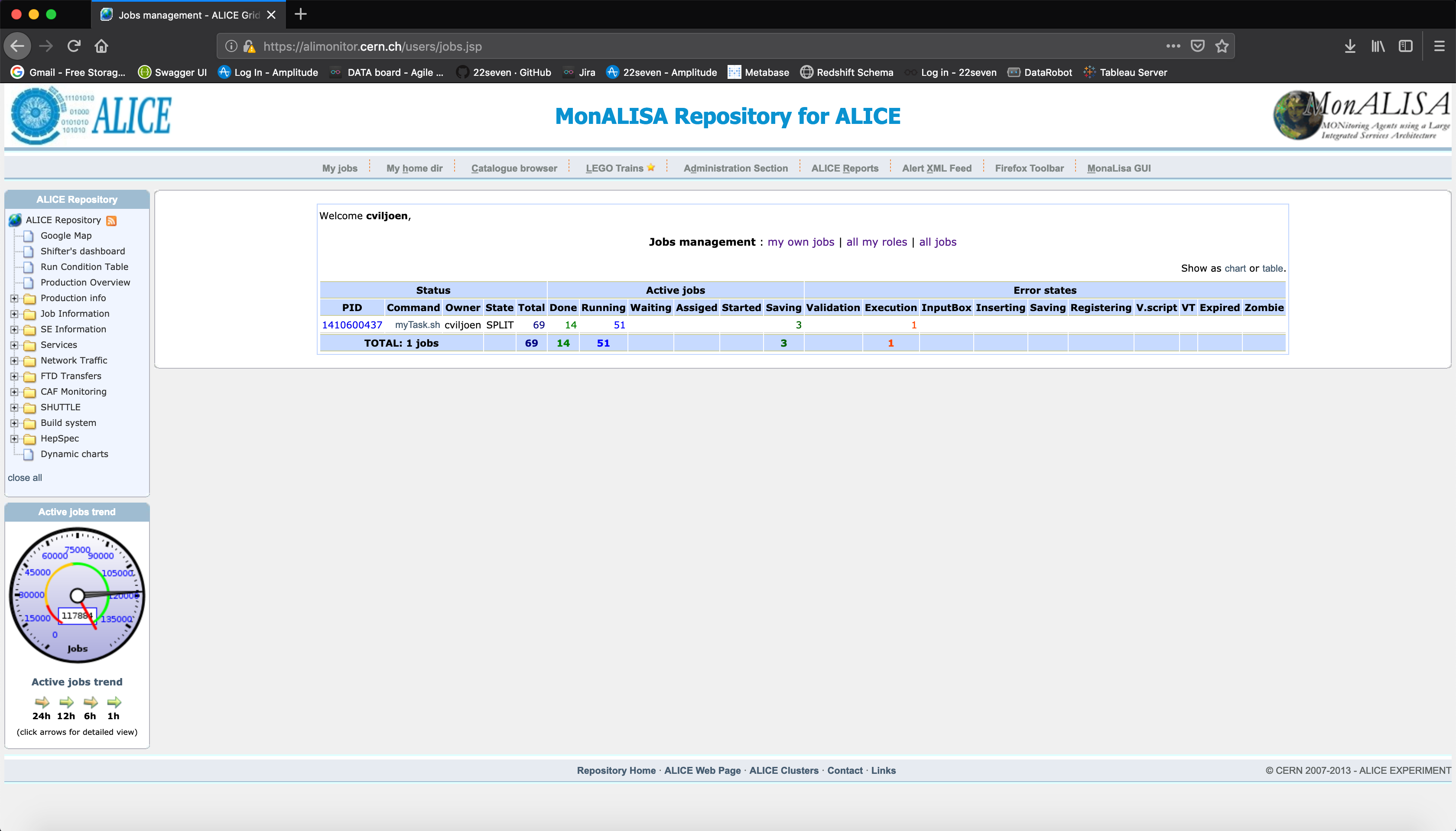


Figure 39: MonALISA Alice grid monitoring site, user jobs at url: <https://alimonitor.cern.ch/users/jobs.jsp>

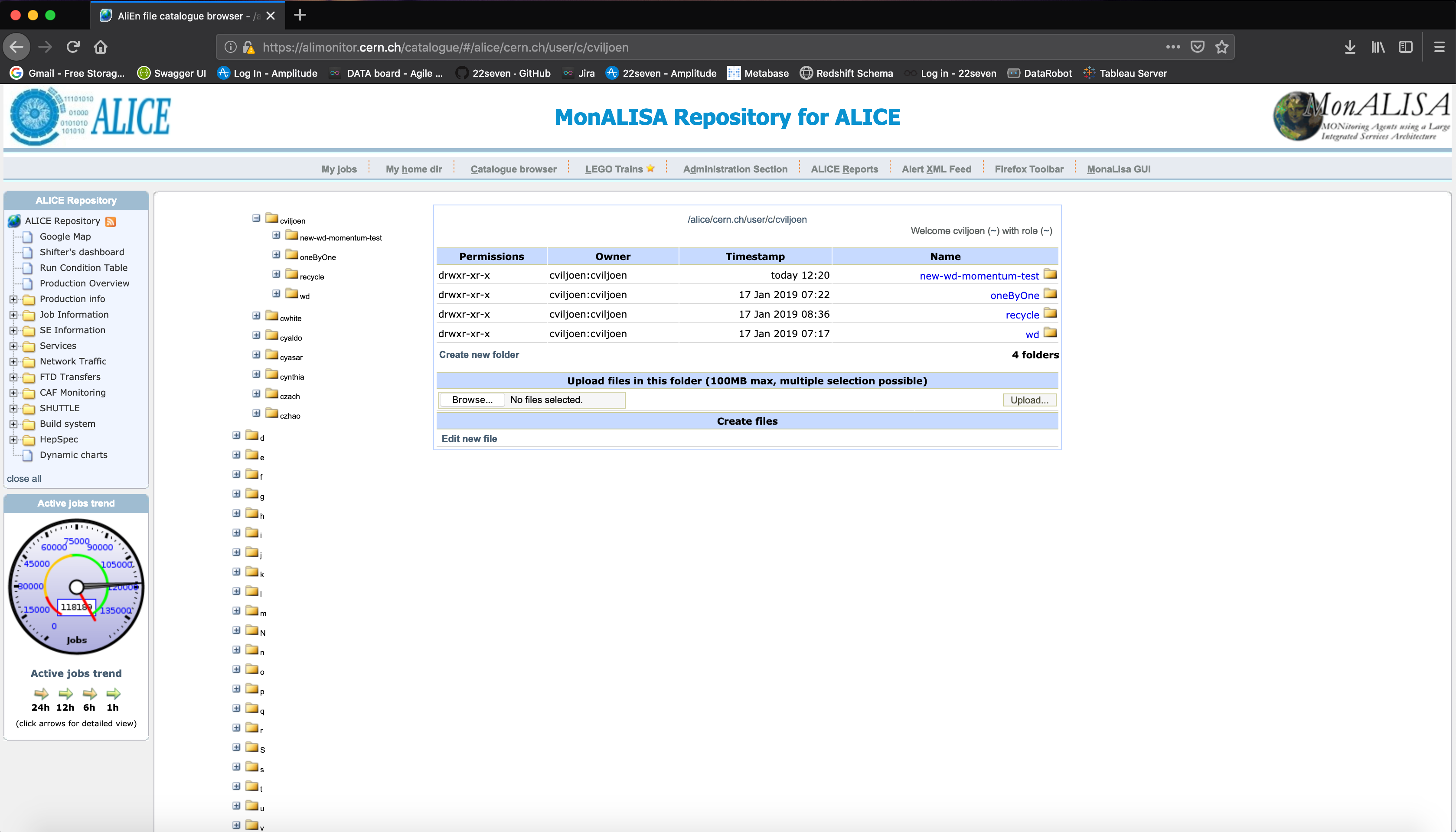


Figure 40: User working directory structure on MonALISA at url: <https://alimonitor.cern.ch/catalogue/#/alice/cern.ch/user/c/cviljoen>

In Figure 41, a screenshot shows how subjobs belonging to a masterjob can be tracked by clicking on the process ID on the MonALISA jobs management webpage:

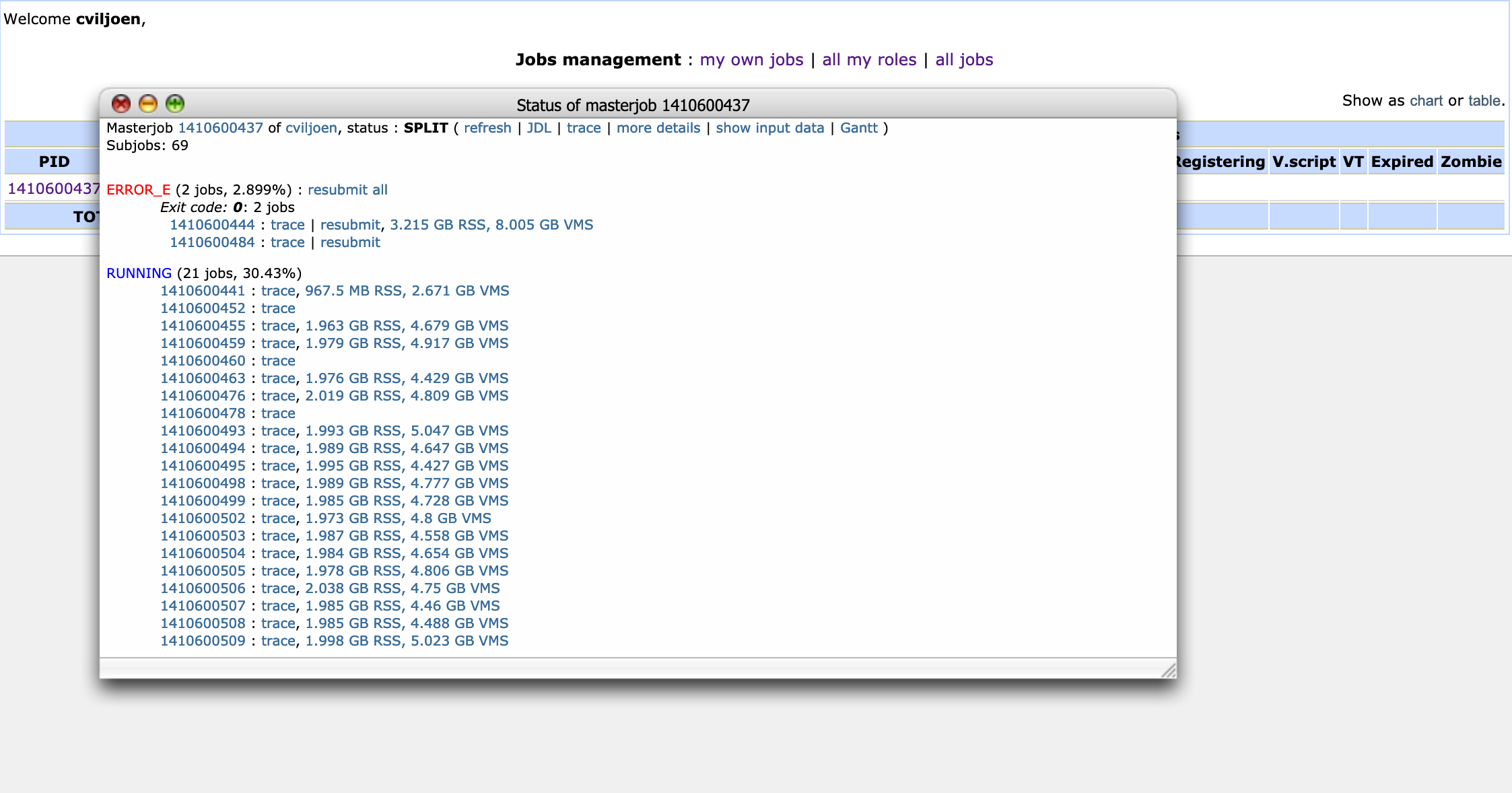


Figure 41: Tracking the status of subjobs of a master-job, by clicking on the process id (PID)

One can resubmit errored subjobs by browsing through the various error states in the “Status of masterjob” view and clicking on “resubmit all” for all processes that are in a specific error state.

The trace of a subjob (see Figure 42 for an example screenshot) can give hints as to what caused a specific subjob to fall into an error state. In this case the job has an error state “ERROR\_E”, i.e. “Error in Execution”, since the job is using too much memory (memory and storage limits are allocated to each user and overusing either can downgrade the priority of a user’s jobs).

The alien shell can be accessed by running

aliensh

This gives access to the alien terminal, which is not strictly a bash terminal, but has similar commands, for instance the shell command to forcefully and recursively remove a directory:

rm -rf directory

would be achieved on an alien terminal by running:

rmdir directory

Killing a job is done in a similar fashion to the normal shell workflow, i.e. running

ps

To list the currently active processes and

kill $(process-id)

To kill a process and its attendant subprocesses, in case you figured out that you made a mistake and want to terminate a running process early for whatever reason.

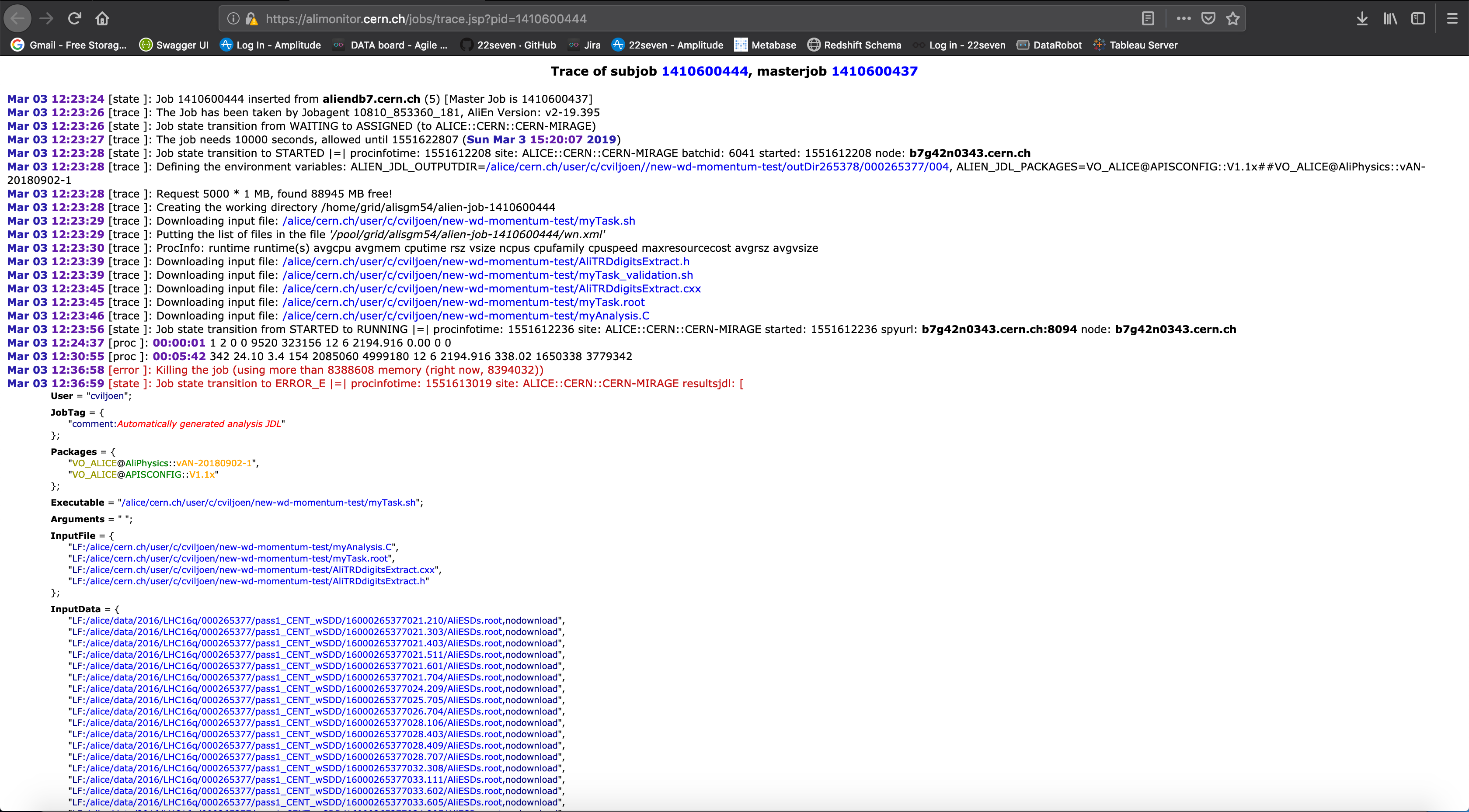


Figure 42: Example trace of a subjob on MonALISA

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

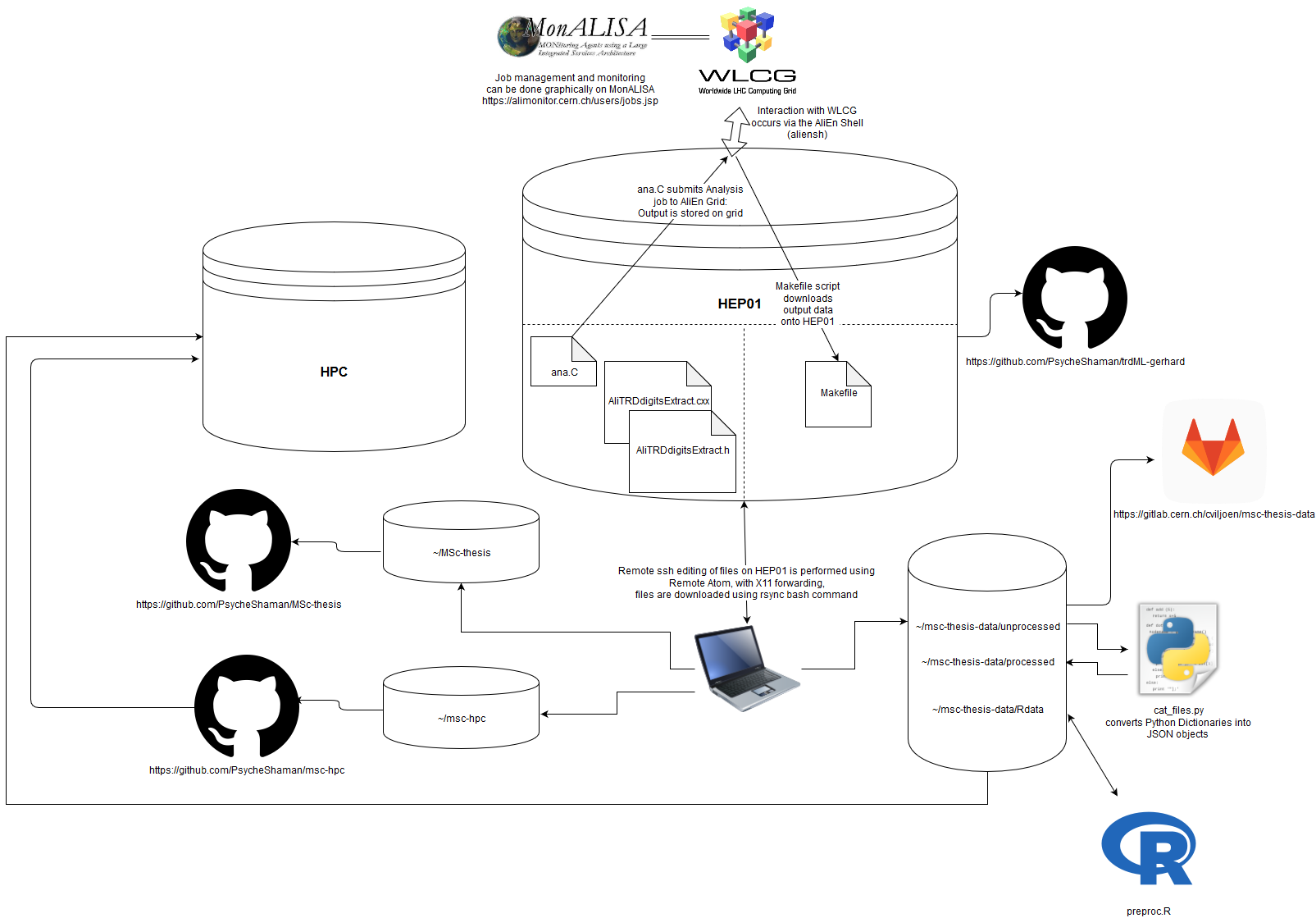


Figure 43: Data pre-processing environment and repository logic

### 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

### UCT HPC Cluster

Compiler variables set in ~/.R/Makevars

CC = gcc -std=gnu99

Install.packages command needs to be modified to write packages in a directory where there are permissions and where the CRAN mirror is set, dependencies=TRUE allows R to read the Makevars compiler variables.

install.packages(pkgs="keras",lib="/scratch/username",

repos="https://cloud.r-project.org",dependencies=TRUE)

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

All electron tracks were included and a pion sample twice the size of the electron sample was added, before partitioning data into a training and test set.

Class imbalances were accounted for by allowing error in electron classification to contribute proportionately more to the loss function.

As per the approach followed by the current classification neural network in production at the TRD, timebins were compressed by summing across three timebins at a time, to create 8 new features, which were added to the 24 timebins already available to the neural network as input features.

Class weights were accounted for as follows:

class\_weights = class\_weight.compute\_class\_weight('balanced',

np.unique(y\_train),

y\_train)

class\_weights = {0:class\_weights[0],1:class\_weights[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:

### 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,

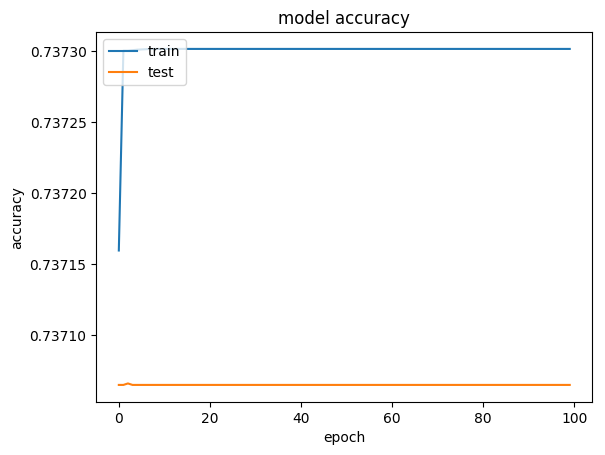
validation\_split=0.1,

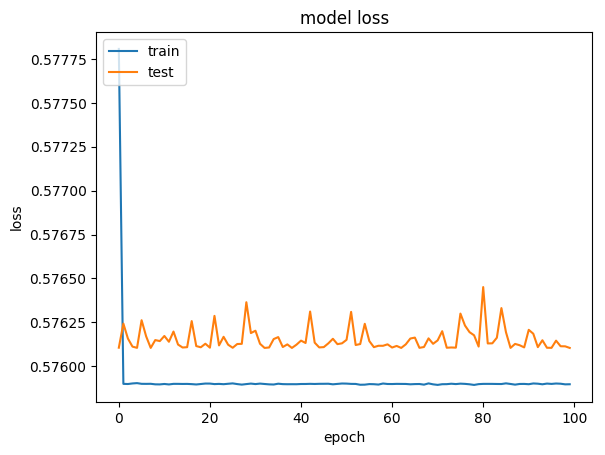
shuffle=True)

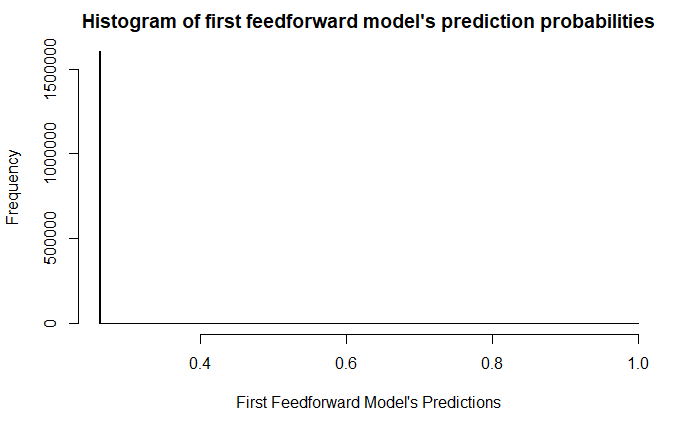
### Recurrent Neural Networks

Appendix H: Old Results Section

#### Model 1

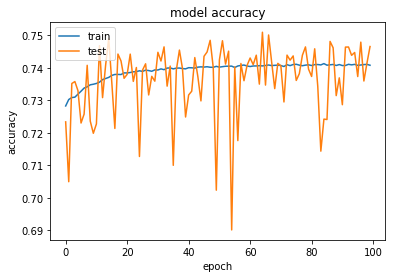


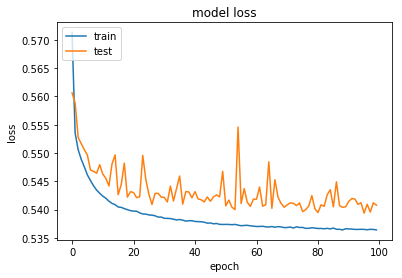


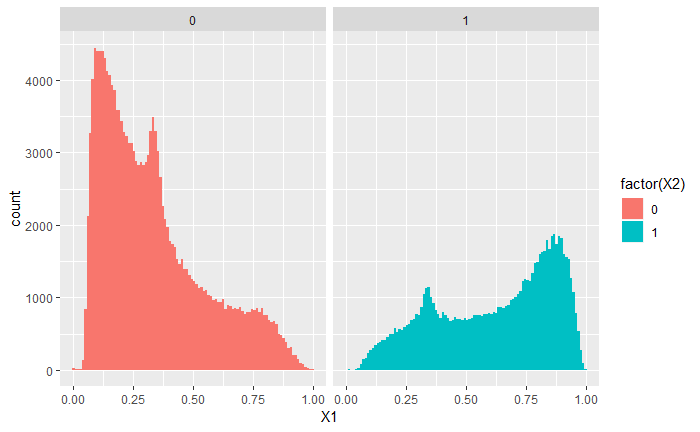


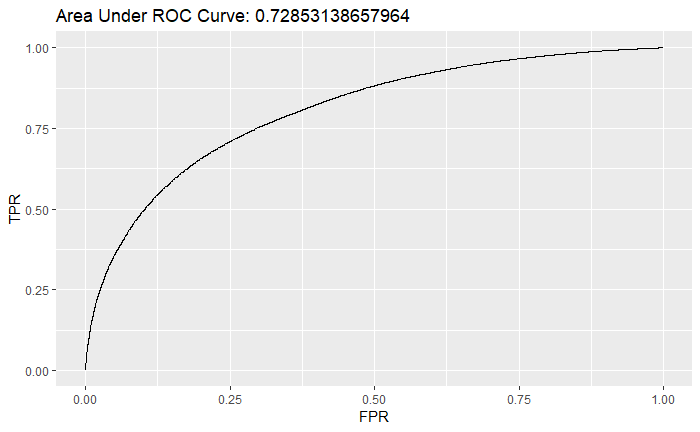
#### Sequential Model Building

##### Stage 1

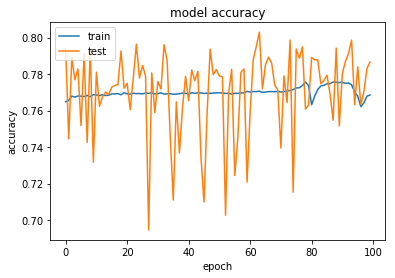


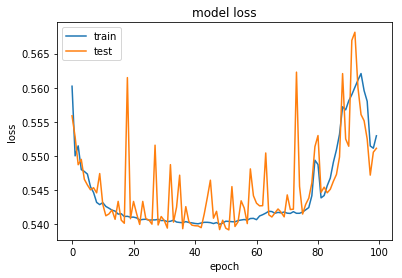


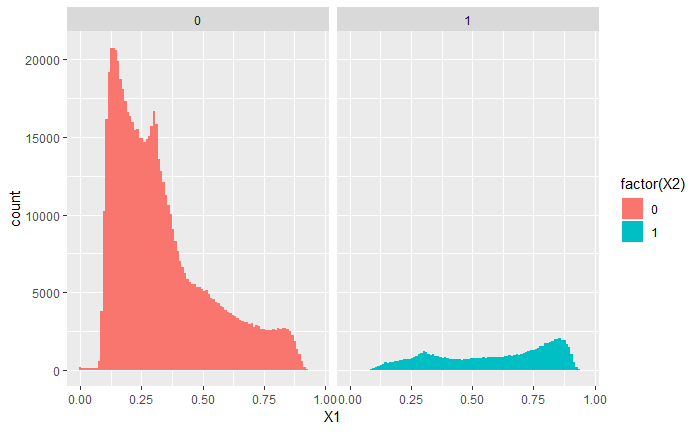


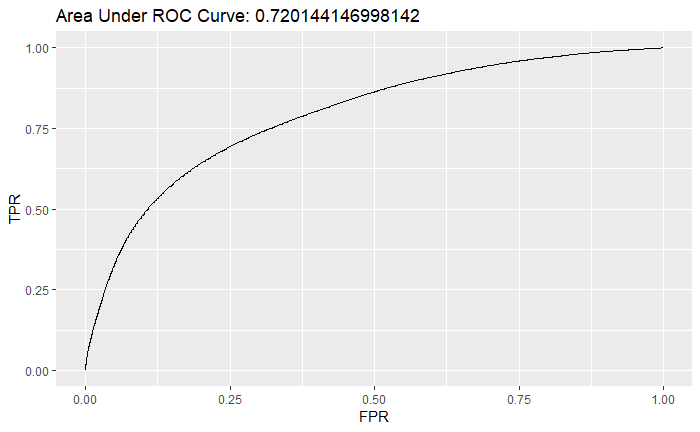


##### Stage 2

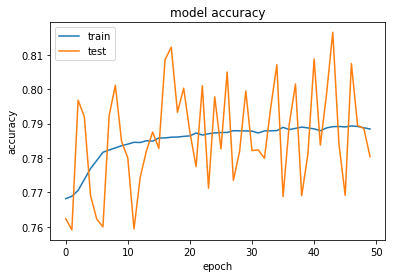


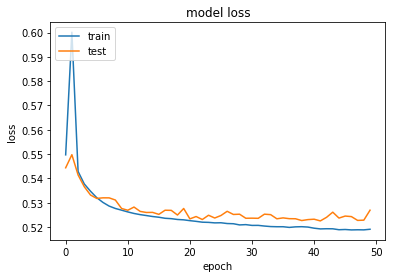


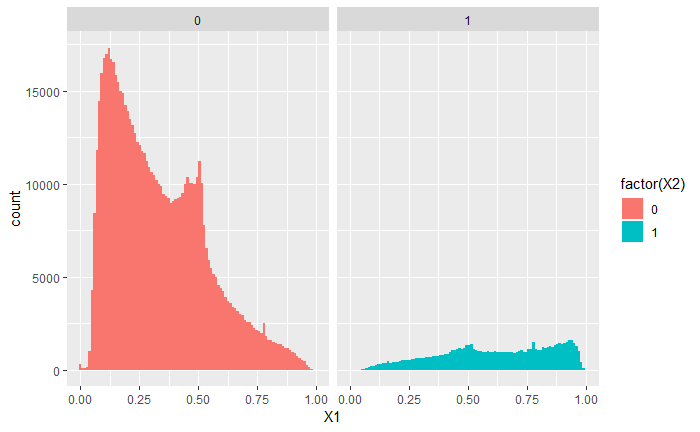


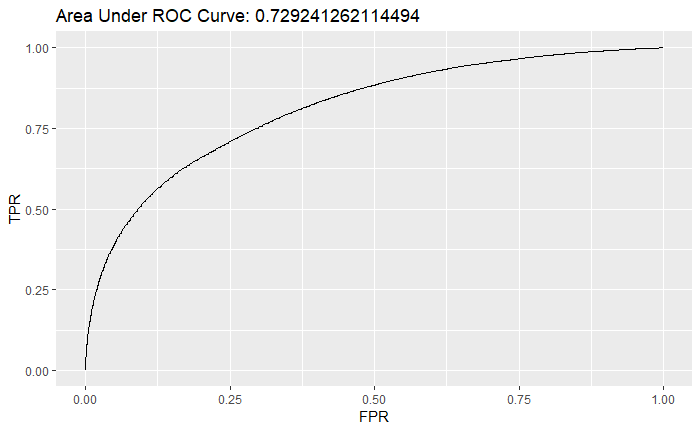


##### Stage 3

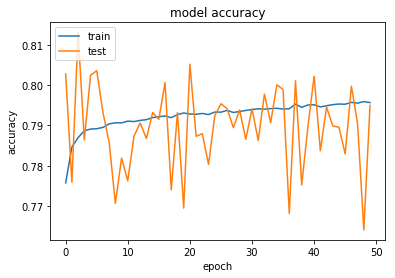


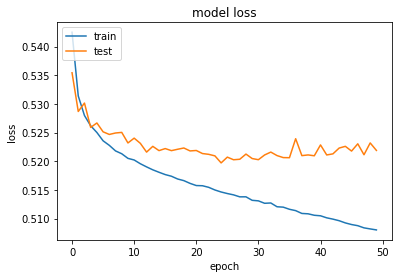






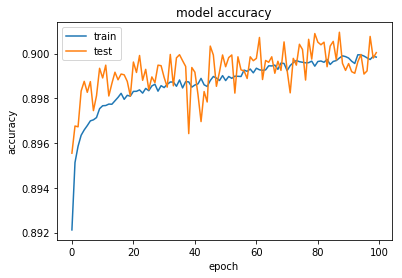
##### Stage 4

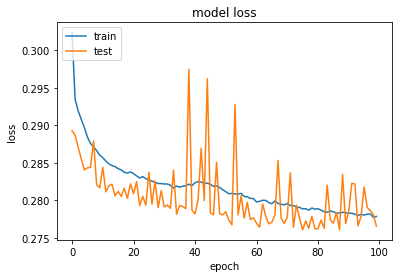




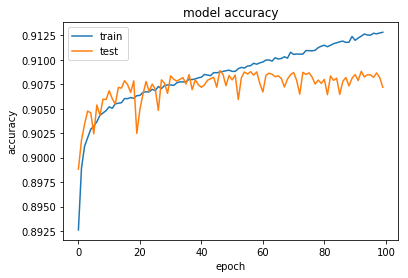
### Convolutional Neural Networks

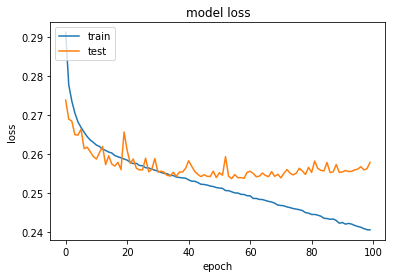
#### Stage 1



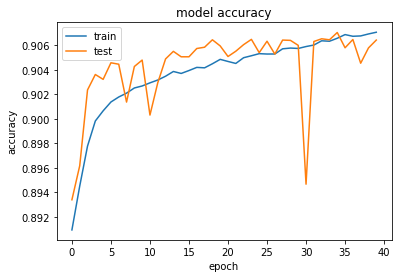


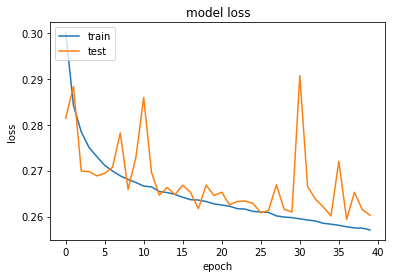
#### Stage 2





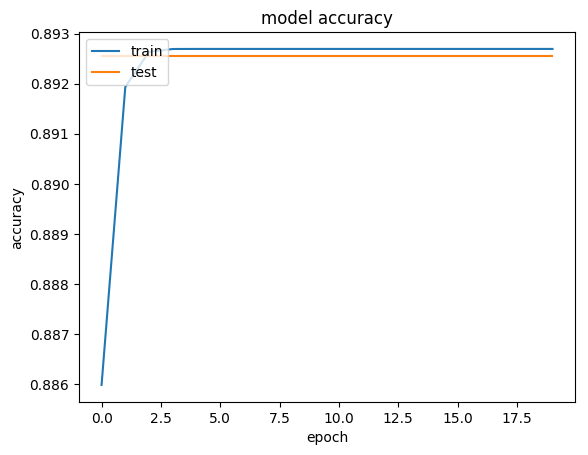
#### Stage 3

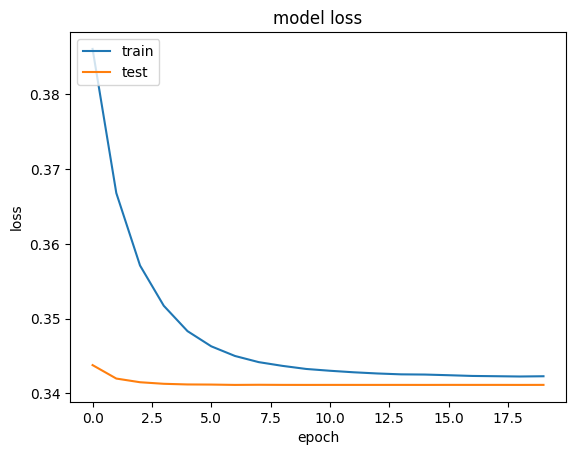




#### Stage 4

#### Stage 5





## Autoencoders



#### Boundary-Seeking Generative Adversarial Network

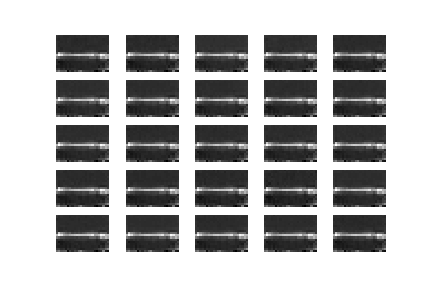
100 Latent dimensions

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

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

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

Example output after 29000 epochs:



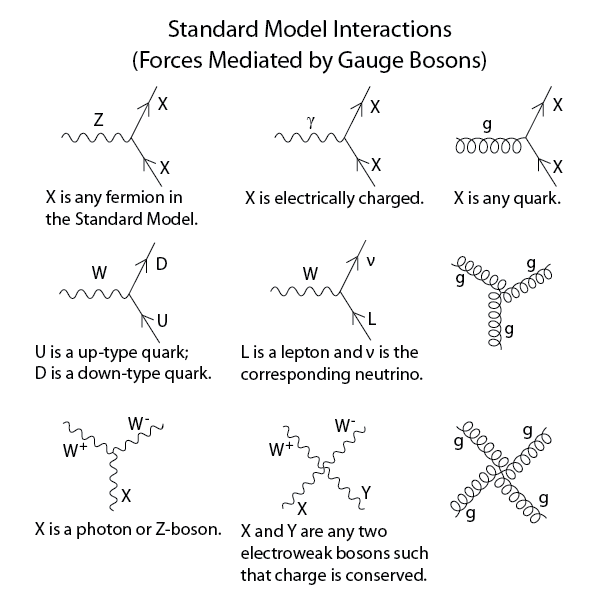
Appendix i: Standard Model Vertices

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 Table 1 carry the weak isospin charge involved in the weak interaction, they all participate in this interaction [1].

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

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



**Figure 1: Standard model interaction vertices [2]**