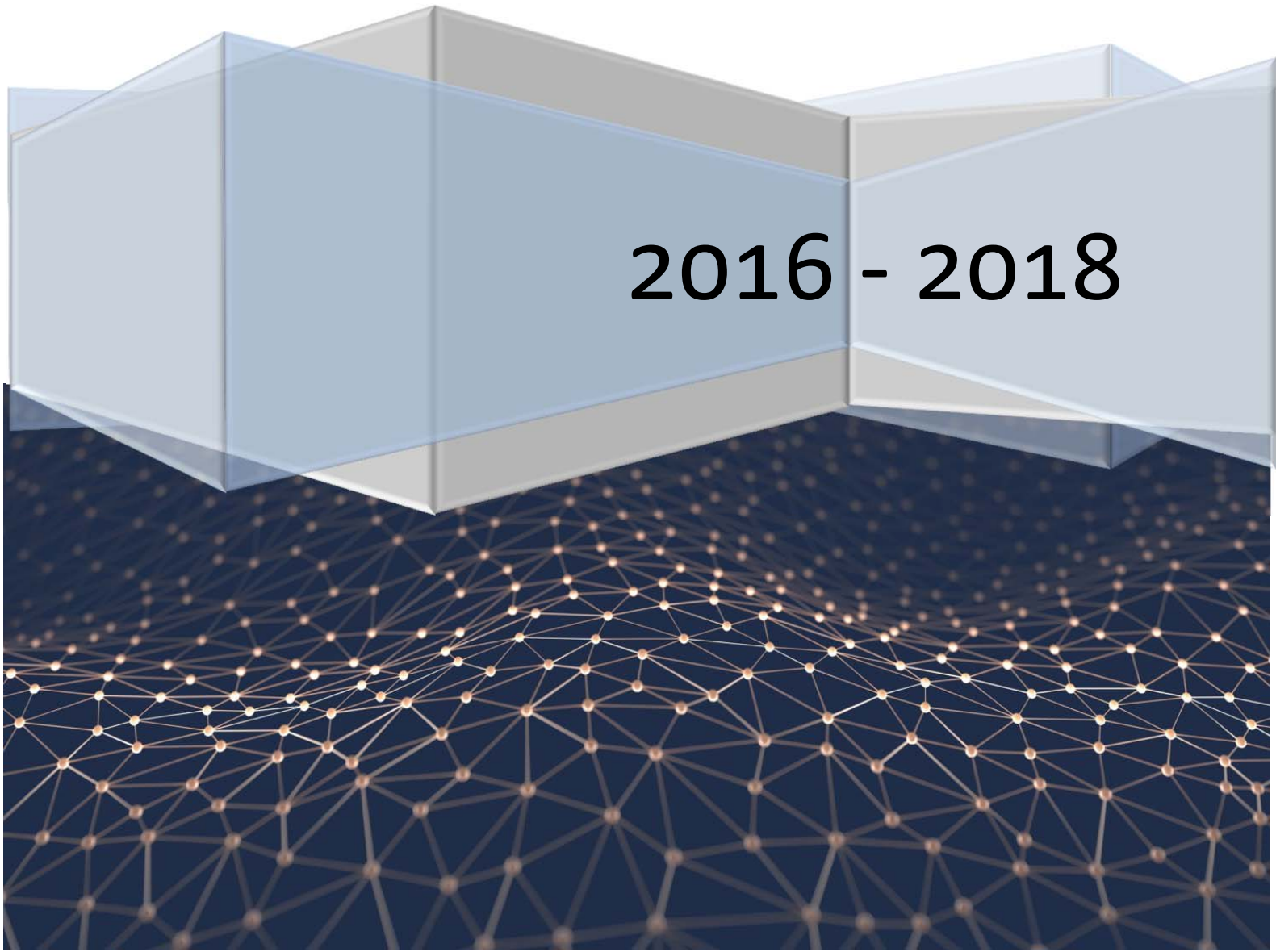
# The LUCID Project

## Neural Networks for Particle Analysis

**Abhishek Shenoy**



# Table of Contents


Abstract ..... 1

Introduction ..... 1

- LUCID ..... 1
- Timepix Chip ..... 1
- Particle Physics ..... 2
  - Particle Tracks ..... 2
  - Energy Values ..... 2
  - Common Particles ..... 3
- Data Collection ..... 4
  - Configs – LUCID Configuration Files ..... 4
  - PTRs – Payload Task Request Files ..... 4
  - Data Format ..... 5
  - Server-Side Management ..... 5
- LUCID Data API ..... 6
- Data Analysis ..... 7
  - Blobbing (Observation) ..... 7
  - Feature Extraction (Interpretation) ..... 7
  - Classification (Decision) ..... 9
  - Conclusion ..... 10

Research ..... 10

- Introduction ..... 10
- Nodes ..... 10
  - Perceptrons ..... 10
  - Sigmoid Neurons ..... 11
- Activation Functions ..... 11
  - Linear ..... 11
  - Rectified Linear (ReL) ..... 12
  - Softplus ..... 12
  - Logistic (Sigmoid) ..... 12
  - Hyperbolic Tangent ..... 12
  - Softmax ..... 12
- Error Functions ..... 13
  - Mean Absolute Error Function (MAE) ..... 13
  - Mean Squared Error Function (MSE) ..... 13
  - Sum Absolute Error Function (SAE) ..... 13

Sum Squared Error Function (SSE) ..... 13
Cross-Entropy Error Function (CEE) ..... 13
Optimisation Algorithms ..... 14
Gradient Descent (GD) ..... 14
Newton-Raphson Method ..... 16
Conjugate Gradient (CG) ..... 18
Architectures ..... 20
Deep Feed Forward Network ..... 20
Deep Convolutional Network ..... 20
Deep Residual Network ..... 21
Benchmark Classifiers ..... 22
Decision Tree ..... 22
Random Forest ..... 22
K-Nearest Neighbour ..... 23
Support Vector Machines ..... 23
Development ..... 24
Training Data Production ..... 24
Method Selection ..... 24
LUCID Neural Trainer ..... 25
Neural Network Construction ..... 27
Architecture Selection ..... 27
Inputs ..... 27
Output Encoding ..... 27
Hyperparameter Selections ..... 28
Final Neural Network ..... 30
Test Data Production ..... 32
Test Results ..... 32
GridPP ..... 32
Accuracy Tests ..... 32
Mini Investigation – South Atlantic Anomaly ..... 35
Conclusion ..... 38
Evaluation ..... 39
Energy Analysis ..... 39
Sub-System Analysis ..... 39
Convolutional Analysis ..... 39
Summary ..... 39
LUCID Dashboard ..... 40

Data Visualisation ..... 40
Tableau Software ..... 40
Plotly Library ..... 40
ArcGIS Platform ..... 40
Google Maps API ..... 40
Dashboard ..... 41
Data Files ..... 42
Radiation Map ..... 42
LUCID Tracker ..... 43
Data Graphs ..... 43
Documentation ..... 44
References ..... 45
Figure Table ..... 47

# Abstract

The purpose of this project is to demonstrate the construction of a more accurate and contemporary analysis algorithm for analysing particle tracks generated by Timepix chips using machine learning. The process of research, investigation and development of the analysis algorithm is described and explained with respect to particle physics, computational heuristics and mathematical statistics. The novel analysis algorithm will be a neural network that can be integrated into a web platform to classify particles from a range of detector chips for a variety of research projects. The main research project that this paper will focus on is the LUCID (Langton Ultimate Cosmic Ray Intensity Detector) project aiming to analyse radiation data collected in the Low Earth Orbit.

# Introduction

## LUCID

Figure 1 - TechDemo Sat-1 (TDS-1)

LUCID (Langton Ultimate Cosmic Ray Intensity Detector) [1] is a device created by students from Simon Langton to detect radiation in space. The payload is a part of the Space Environment Suite [2] on the satellite TechDemoSat-1 (TDS-1). The satellite was launched on the 8th of July 2014 from the Baikonur Cosmodrome in Kazakhstan. Currently the project is funded by the Institute for Research in Schools (IRIS) with the courtesy of Dr Parker (Director of IRIS). Surrey Satellite Technology Limited has developed the student design in collaboration with successive years of students and scientists at CERN.

Figure 2 - LUCID - 5 x Timepix Chips

There are 5 Timepix chips on the device created by the Medipix collaboration; 4 of them on the outer sides of the open cube and the 5th one at the bottom of the open cube as shown in Figure 2.

## Timepix Chip

To understand the basis of particle analysis, it is essential to understand how the detector actually works. This will help to understand the properties and the limitations of the particle tracks.

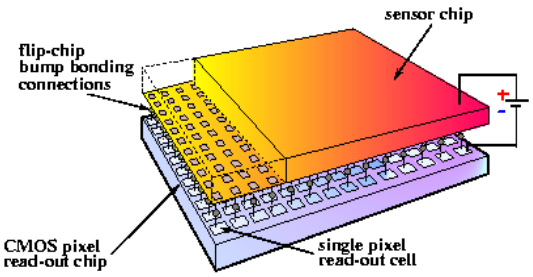



Figure 3 - Timepix Chip

Timepix chips (Figure 3) [3] have a pixel matrix of 256 x 256 (65,536 pixels) with each of the square pixels having only a side length of *55 $\mu m$*. The entire active area of the silicon sensor chip is 2cm$^2$. Upon collision of an ionising particle onto the surface of the chip, a charge is deposited into the silicon layer. The charge is gradually removed by the potential difference supplied across the detector called the threshold voltage. This process takes place by using CMOS circuits (complementary metal-oxide semiconductor) which are built into the chip.

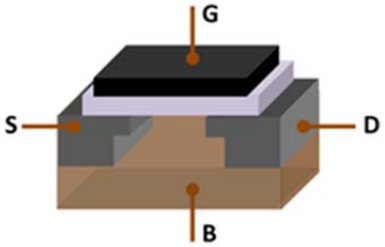



Figure 4 - MOSFET

The CMOS circuits consist of billions of p-type and n-type MOSFETs (metal–oxide–semiconductor field-effect transistors) to create logic gates and other circuitry. The three contacts on a MOSFET (shown in Figure 4) [4] allow it to be used like a switch. The source (S) supplies the electricity and depending on the electrical voltage on the gate terminal (G), the voltage through the drain terminal (D) can be altered. The gate is separated from

the body by an insulating layer (white). Increasing the potential difference on the gate allows electrons to flow between the source and the drain terminals. Using this mechanism, each of the pixels can then record the time it takes for the charge to be reduced below a certain threshold and the output is known as Time over Threshold (ToT).

Higher energy particles deposit a greater charge than lower energy particles, so the Time over Threshold is much greater as it takes longer for the charge to be reduced below the configured threshold. Therefore, Time-over-Threshold (ToT) is known as the energy value and is measured by counting the clock ticks. The Timepix chip has a single energy threshold and is stored using 4 bits for the whole pixel matrix. All of the data is accumulated in a 13-bit counter per pixel (1kB per pixel). Each pixel overflows at 11810 counts and the chip data is read out only after exposure to prevent dead time and the loss of data.

## Particle Physics

### Particle Tracks

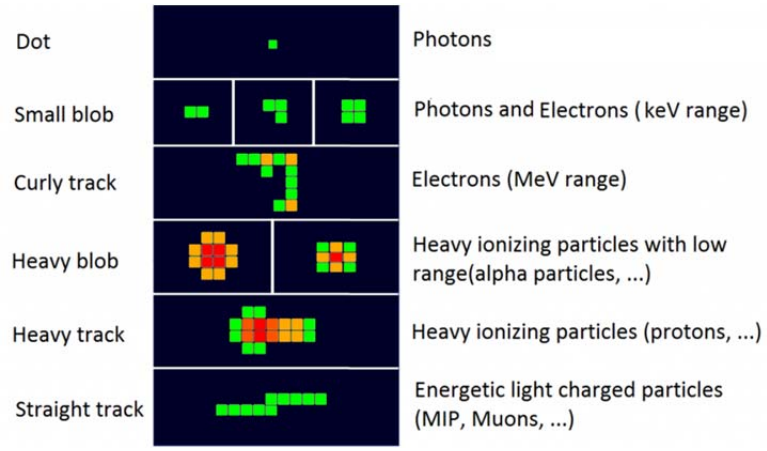



Figure 5 - Particle Tracks

There are several variations in the particle tracks received by LUCID however, the main properties of the tracks are shown in (Figure 5). These are very useful in identifying common particles however they are not very help to judge unrecognised particles.

The energy values are very important in understanding the direction of travel before collision and the point of collision of the particle. These can be used to identify unfamiliar particles. However, these properties are difficult to describe abstractly or mathematically. Abstraction can also lead to incorrect inferences about the type of particle and therefore it is simpler and more accurate to determine a particle by only its track shape rather than the energy values. The energy value may be useful upon deeper analysis, since the properties of the particle are directly correlated with the energy deposit from the particle onto the detector chip.

### Energy Values

The mean energy loss of a moving particle is also known as the stopping power of the material it is travelling through. This energy loss is calculated using the Bethe Formula (Figure 6Figure 6 - Bethe Formula). For electrons, the energy loss requires relativistic corrections due to their small mass. Fast charged particles moving through matter interact with the electrons of the atoms in the material. The interaction excites or ionizes the atoms, leading to an energy loss of the travelling particle.

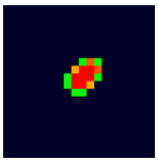
$$-\left\langle \frac{dE}{dx} \right\rangle = \frac{4\pi}{m_e c^2} \cdot \frac{nz^2}{\beta^2} \cdot \left( \frac{e^2}{4\pi\varepsilon_0} \right)^2 \cdot \left[ \ln\left( \frac{2m_e c^2 \beta^2}{I \cdot (1-\beta^2)} \right) - \beta^2 \right]$$

Figure 6 - Bethe Formula

| | |
|---|---|
| $v = Velocity$ | $c = Speed\ of\ Light$ |
| $z = Relative\ Charge$ | $e = Electron\ Charge$ |
| $E = Energy$ | $x = Distance$ |
| $n = Electron\ Density\ of\ Material$ | $I = Mean\ Excitation\ Potential$ |
| $c = Speed\ of\ Light$ | $\beta = v/c$ |
| $\mathrm{m}_0 = Rest\ Mass$ | $\varepsilon_0 = Vacuum\ Permittivity$ |

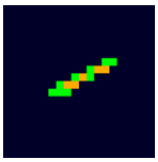
## Common Particles

### *Alpha*



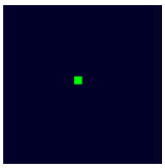
Alpha particles have one of the most distinct particle tracks as they form round circular blobs. They are heavy particles made up of 2 protons and 2 neutrons and therefore have a relative mass of 4. The +2 relative charge means that upon collision, more of the kinetic energy of the particle is deposited into the silicon layer. The velocity of alpha particles also tends to be much lower as the particle has to have more kinetic energy for its mass. From the Bethe Formula, as the velocity increases the amount of energy deposited decreases.

### *Electron*



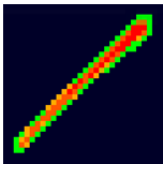
The relativistic Bethe formula is required to calculate the energy deposit of electrons as the equation needs to account for the fact that electrons have a much smaller mass than heavy particles like protons and that the electron is interacting with identical particles in the atoms of the material it is travelling through. The interaction between the incoming electron and the atomic electron allows for the possibility of exchange of the two particles.

### *Gamma*



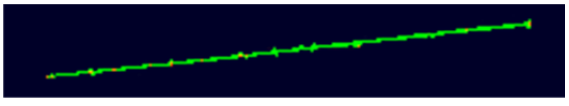
Since gamma particles are a part of the electromagnetic spectrum, these particles travel at the speed of light and therefore calculating the energy deposit for gammas requires the relativistic version of the formula. Since these particles are relativistic, the energy deposit increases logarithmically due to the transversal component of the electric field.

### *Proton*



Protons have a relative charge of +1 and a relative mass of 1. The high charge means that they deposit a lot of energy. However due to the lower mass, they tend to have a high velocity for the same amount of kinetic energy and therefore do not deposit as much energy as a heavy ion or an alpha particle.

### *Muon*



These are minimum ionising particles that have approximately 200 times as much mass as an electron however they are still 10 times lighter than a proton. Muons travel at high velocities and this means that they undergo time dilation as predicted by special relativity allowing them to travel long distances even though their decay time is much smaller. Due to their high velocity, they tend to deposit very little energy into the chip but at the same time their high mass makes the particle tracks thicker than those of beta particles. Due to the high velocity of muons, the particle tracks they produce are highly linear.

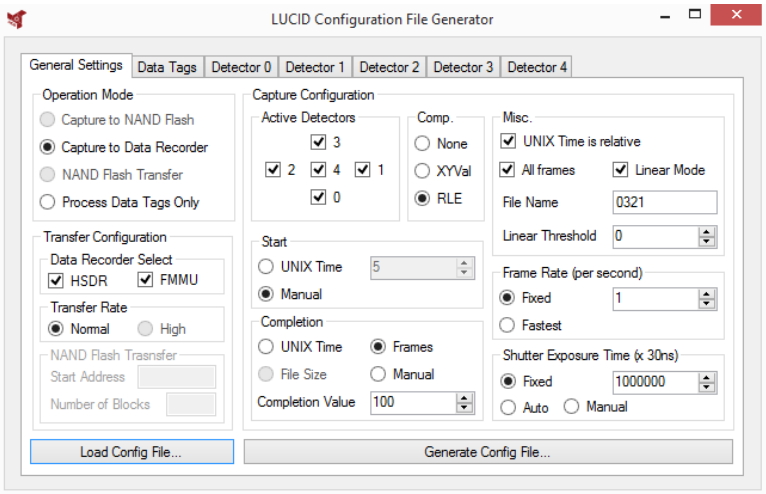
### *Other*

These particles are unrecognised or cannot be classified as any of the above particles due to its irregular particle track. These are often heavy ions, multiple particles, particle decay or just unknown particles. Misclassifications are relatively easy for this category due to the varied range of possibilities and therefore this category will mainly be used as a discriminator between recognised and unrecognised particles.

# Data Collection

## Configs – LUCID Configuration Files

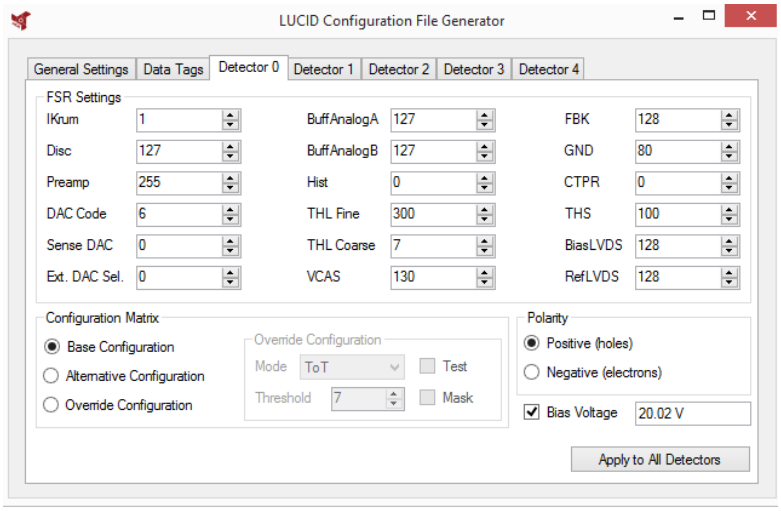
The LUCID configuration file assigns the settings for LUCID. This includes the following:




The general settings include the active detectors, the compression format, the start time, the definition of completion, the configuration file name, the frame rate and the shutter exposure time.

For each detector, there are these individual settings that can be adjusted. The majority of these settings are left as default however the following are changed depending on the objective:

The bias voltage is the voltage across the chip used to calculate the energy value of the deposited charge from a particle.




## PTRs – Payload Task Request Files

A PTR file is sent to LUCID for each run specifying the time at which the data should be captured and the configuration file to be used for each set of frames (a data file). The files are generated approximately a week before the designated run and are submitted by uploading to an FTP server hosted by SSTL. PTR files, unlike configuration files, cannot be re-used for multiple runs, as they require the absolute start time for each capture, rather than relative to the start of the run.

PTR files are written in XML and consist of a number of <PTR> tags, each of which schedules LUCID to either one of the following:

(1) Transfer a particular configuration file into the memory to be used for future captures
(2) Schedule a data capture using the currently active configuration file at a specific time.

To simplify the task of routinely writing PTR files - which will often need to schedule hundreds of captures during a run - a piece of software has been developed to generate the files automatically. The PTR generator works on the model of templates, skeleton PTR files which describe a certain pattern of data capture, which the software fills out with the appropriate dates and times for a specific run. The generator can either be used as a graphical application or a command line utility [5].

## Data Format

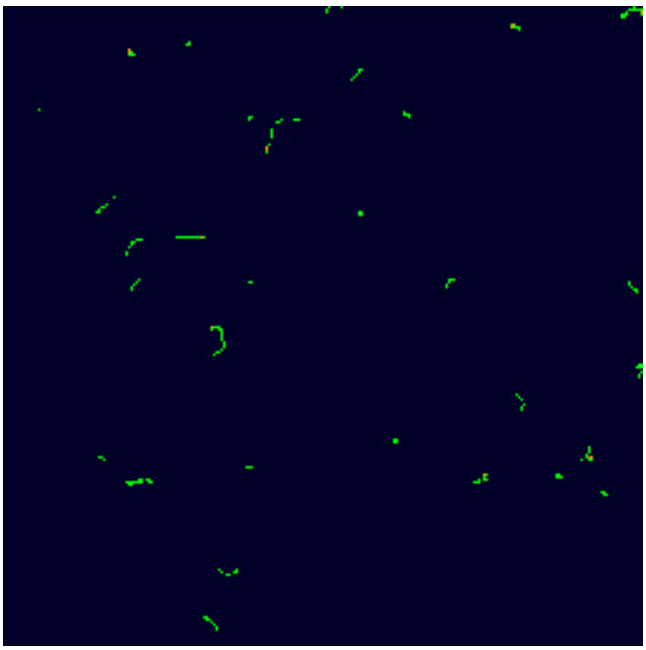


Figure 7 - PNG of a Data Frame

| | | |
|---|---|---|
| 14 | 41 | 10.0 |
| 37 | 82 | 49.0 |
| 38 | 81 | 22.0 |
| 38 | 82 | 17.0 |
| 38 | 180 | 19.0 |
| 39 | 80 | 5.0 |
| 39 | 180 | 40.0 |
| 40 | 80 | 7.0 |
| 40 | 181 | 19.0 |
| 41 | 79 | 33.0 |
| 44 | 76 | 30.0 |
| 49 | 98 | 31.0 |
| 49 | 99 | 26.0 |
| 49 | 190 | 21.0 |
| 50 | 17 | 135.0 |
| 50 | 18 | 143.0 |
| 50 | 19 | 8.0 |
| 50 | 96 | 16.0 |
| 50 | 190 | 15.0 |
| 50 | 191 | 19.0 |
| 51 | 18 | 22.0 |
| 51 | 19 | 17.0 |
| 51 | 94 | 19.0 |
| 51 | 95 | 44.0 |
| 51 | 112 | 7.0 |

Figure 8 - Partial XYC of a Data Frame

The three columns of the XYC file in Figure 8 represent the x-coordinate, the y-coordinate and the energy (Time over Threshold) value, hence the name XYC.

The XYC can be converted to an image (Figure 7) simply by using a conversion program which plots each pixel depending on its energy value to create an image of the data frame.

## Server-Side Management

The server has 12GB of RAM and 16 Intel Xeon cores clocked at 2.53GHz, and runs Ubuntu 16.04 LTS. It has 146GB of storage for the OS and 21TB for the data. Every 8 days, a cron job is used to check an FTP server hosted by SSTL to download any new LUCID data to the server.

Each data frame is stored as an individual file inside its designated data run folder on the server for the specific data file ID, frame number and the detector number.
(https://starserver.thelangton.org.uk/data/LUCID/xyc/data_run/id/frame_channel.txt)

Example: https://starserver.thelangton.org.uk/data/LUCID/xyc/2016-12-16/0747002227/frame1c0.txt

The above URL returns the XYC file as plaintext in a browser.

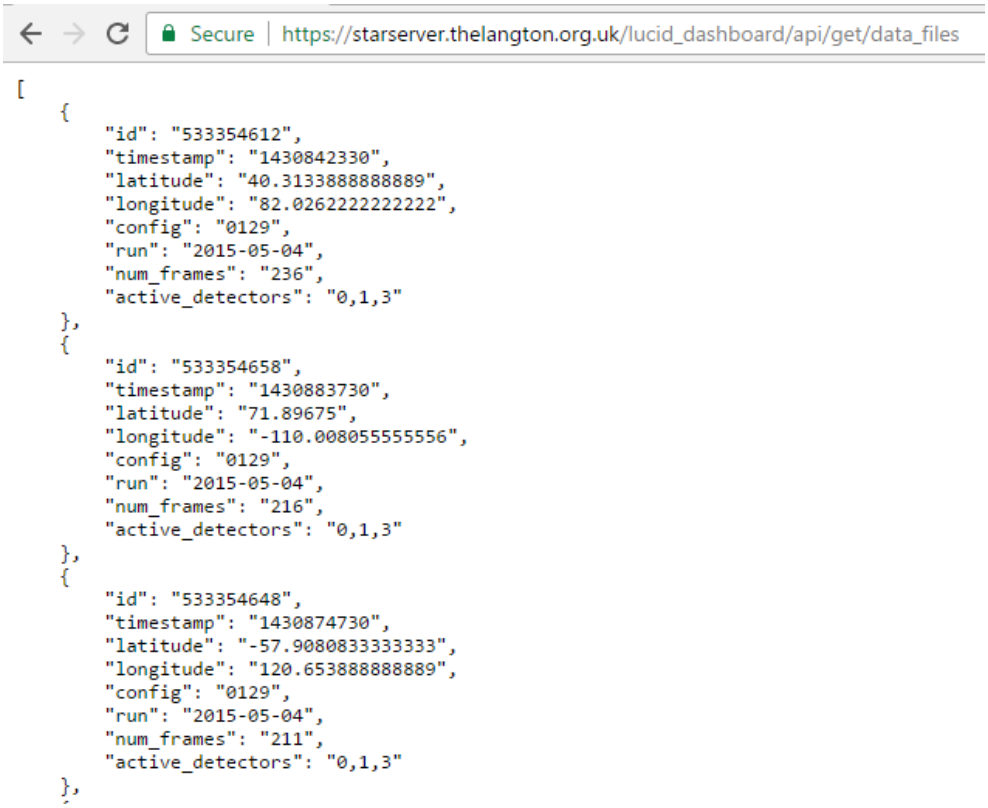
The metadata of all the data files is stored in a SQLite database consisting of the following fields:

- ID – The ID of the data file
- Filename – The name of the data file with the storage path
- Latitude, Longitude – The location of where the data file was created
- Timestamp – The time when the data file was created
- Config – The configuration file used for the data file
- Active Detectors – The detectors that were used in the data file
- Number of Frames – The total number of data frames collected for the data file
- Data Run – The date the 8 day data run began (when the data file was created)

## LUCID Data API

To access the LUCID data, I created a REST (Representational State Transfer) API (Application Programming Interface) which returns the data in JSON (JavaScript Object Notation) format when a specific URL is typed. This is done by using PHP and MySQLi to access the SQLite database where all of the metadata is stored.
The base URL for the API: http://starserver.thelangton.org.uk/lucid_dashboard/api/

### get/data_files




This returns all the information about all the LUCID data files ever collected using the GET request via the HTTP protocol.

Each single block of text in the curly braces refers to a single data file returning the location, the timestamp, the total number of frames taken, the configuration file used, the active chips out of the 5 detectors and the data run (the 2-day time period during which the data file was created).

### get/frames?data_file={id}


The ID of the data file from the previous URL can then be used to view the frames of the data file by using the above GET request.

Each single block of text in the curly braces refers to a data frame recorded by the chips at the given timestamp and location.

The channels array refers to the data frame from each individual chip, out of the active detectors. The key of the dictionary is the chip number and the value is the XYC frame in a serialised format.

## Data Analysis

To gain valuable insight, the data needs to be processed in a way by which useful data can be extracted from the raw format and the vast amounts of data. The main use of LUCID data is for the exploration of particle physics using particle analysis. The existing data analysis algorithm is currently used by several entities including CERN@school and the TAPAS (Timepix Analysis Platform at School) web application by IRIS. The aim of this section is to identify the significant parts of the current analysis algorithm so that the new algorithm is more accurate.

### Blobbing (Observation)

#### Blobbing

To analyse the frames of LUCID data, it is necessary to identify the individual particles in a data frame. To accomplish this, the blobbing program simply takes a 256x256 NumPy array (representing a frame) as a parameter and returns a list of lists of tuples where each sub-list is a blob and the tuples it contains are the coordinates of every single pixel within that blob.

A recursive algorithm is used to check each hit pixel one by one, adding the pixel to a new cluster if it is not already a part of an existing cluster. It then finds all adjacent pixels, adds those to the same cluster, and then finds all the adjacent pixels to those adjacent pixels, deeper and deeper until there are no more adjacent pixels to be added to the original cluster. A blobbing radius is also given as a parameter to classify two blobs within the specified radius as one particle.

### Feature Extraction (Interpretation)

#### Common

To interpret and extract the key features of the clusters, it was necessary to calculate some metrics for the particle tracks in a way that would help to determine the properties of the particles.

The following metrics are calculated from the cluster that is provided as a list of pixels to the program:

Number of Pixels → This is calculated as the length of the pixels list.
Centroid → This is calculated using the **find centroid** function.
Radius → This is calculated using the **calculate radius** function.
Diameter → This is twice the calculated radius.
Density → This is calculated using the **calculate density** function.
Line Residual, Best Fit Theta → This is calculated using the **calculate non-linearity ("squiggliness")** function.
Centre of Circle, Curvature Radius, Circle Residual → This is calculated using the **find best fit circle** function.
Average Neighbours → This is calculated using the **find average neighbours** function.
Width → This is the number of pixels divided by the diameter provided that the number of pixels is not 1. For single pixels, the width is set to 0.

#### Find Average Neighbours

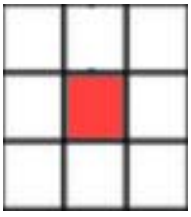


Figure 9 - Average Neighbours

For each individual pixel (red pixel in Figure 9) in the cluster, the number of neighbouring pixels is counted by iterating through the surrounding pixel coordinates (all the white pixels in Figure 9) and checking if they exist in the cluster. The pixels that exist are then stored in a list as these are the ones that have an energy deposit. The mean of the list is then calculated and the function returns the average number of neighbours.

**Find Centroid – (Centroid)**

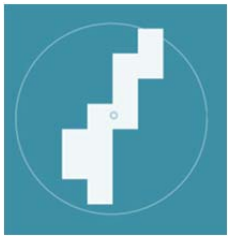


Figure 10 - Centroid

The centroid of the blob is the centre of the particle track. This centroid is found by calculating the mean of the x values and the mean of the y values of the pixels in the blob. The mean x-value is the presumed to be the x-coordinate of the centre point and the mean y-value is presumed to be the y-coordinate of the centre point. This coordinate is not the centre of the smallest enclosing circle but in fact, it is the centre of mass with no weighting on energy values (shown in Figure 10 - Centroid).

**Calculate Radius – (Radius & Diameter)**

$$Radius = \sqrt{(x_c - x_n)^2 + (y_c - y_n)^2}$$

The Euclidean distance between the coordinates of each pixel to the centroid is calculated. The radius is set to the highest distance from the centroid. The diameter is then simply twice the calculated radius.

**Calculate Density – (Density)**

$$Density = \frac{Pixel\ Count}{\pi r^2}$$

The density can be greater than 1 as the blob's radius passes through the centre of the outer pixels rather than around them. If the blob is only one pixel in size, then the density is by default set to 1.

**Calculate Non-Linearity – (Line Residual & Best Fit Theta)**

This function returns the return angle theta anticlockwise from the x-axis, with the line passing through the cluster centroid. First, all the coordinates of the pixels within the cluster are split into separate lists of X and Y values. Single pixel gammas are by default given an angle and line residual of 0 as these are completely linear.

$$b = \frac{\sum XY - \frac{\sum X \sum Y}{n}}{\sum X^2 - \frac{(\sum X)^2}{n}}, a = \bar{Y} - b\bar{X}\ where\ \bar{X} = \frac{\sum X}{n}\ \&\ \bar{Y} = \frac{\sum Y}{n}$$

For all other clusters, the above least squares regression function is used to calculate the line of best fit in the form $y = a + bx$. From the line of best fit, the best-fit angle $\theta$ (anticlockwise from the x-axis) of the line-of-best-fit can be calculated using simple trigonometry. However, all of this is done by the sci-kit python library and therefore the mathematics behind this has only been shown for understanding the algorithm.

$$Line\ Residual = \sum_{n=0}^{Pixel\ Count} (x_n sin\theta - y_n cos\theta + x_c sin\theta - y_c cos\theta)^2$$

This best fit angle is used to calculate the line residual value (also known as the “squiggliness” of the particle track) using the equation above. The line residual value is calculated by the sum of the square of the distance from each pixel coordinate to the line travelling through the centroid where the line is calculated using the best fit angle $\theta$. The equation above shows the simplest form of the line residual where $(x_c, y_c)$ is the coordinate of the centroid and $(x_n, y_n)$is the coordinate of the pixels in the cluster where $n$ is the index of the pixel in the cluster.

**Find Best Fit Circle – (Curvature Radius & Circle Residual)**

This function is used to perform circle regression however for single pixel gamma particles, this cannot be done and therefore single pixels are by default given values of zero.

As the cluster is a very poor initial guess for the centre of the circle, multiple test circles are generated using the calculated best fit angle from the regression line calculation.

For each of these test circles, the program performs circle regression on the cluster using least squares. The Euclidean distance between the data points and the mean circle centred at $(x_c, y_c)$ is calculated. The mean distance is then subtracted from all of the calculated distances. This value is optimised using the least squares function from the SciPy python library. The optimisation process returns an optimised centre point.

This optimised centre point is used to calculate the distance of all the points from the centre of the circle once again. The mean distance is set as the new radius and the residual is calculated by the summation of the squares of the difference between the distance from each point to the optimised centre point and the mean radius.

$$(Mean\ Radius)\ R_\mu = \frac{\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2}}{n}$$

$$Circle\ Residual = \sum_{n=0}^{Pixel\ Count} (\sqrt{(x_c - x_n)^2 + (y_c - y_n)^2} - R_\mu)^2$$

The optimised test circles are then organised in order of the magnitude of the residual. The aim of the optimisation is to reduce the circle residual as much as possible and therefore the circle with the least residual is chosen as the best fit circle.

## Classification (Decision)

**LUCID Algorithm**

As shown in the introduction, most of the common particles have particle tracks with unique properties that allow them to be identified. To do this algorithmically, the LUCID algorithm imports the common module and works out various metrics (numeric values based on each pixel's position, such as radius and density) for each blob, and checks them against predefined bounds for each particle type.

```
1.  if num_pixels < 4:
2.      return 'gamma'
3.  if num_pixels < 8:
4.      return 'beta'
5.  if (density > 0.75 and num_pixels > 11) or (avg_neighbours > 6 and curvature_radius > 8):
6.      return 'alpha'
7.  if (line_residual / radius) < 0.1 and radius > 40:
8.      return 'muon'
9.  if ((curvature_radius > 50 or line_residual < circle_residual) and width > 1.5) or avg_neighbours
    > 3.5:
10.     return 'proton'
11. else:
12.     return 'beta'
```

The simplicity of the above algorithm allows it to define alpha, beta and gamma particles with a high degree of accuracy. However, it also means that occasionally protons and MIPs are misclassified as beta particles. Moreover, as there is no option for unknown particles, any unrecognised particle is automatically classified as a beta particle. The limitations of the current analysis algorithm are clearly outlined above hence suggesting the need for a more robust flexible analysis algorithm that will be future-proof and will account for any new particle classes.

### Conclusion

The algorithm currently being used [6] is highly inaccurate and does not account for many of the factors involved in producing the specific particle tracks. To improve the analysis program, I will be using machine learning for data classification. So when given another dataset, the program will be able to deduce the type of particles within the test data. There are many possible improved alternatives to the current analysis algorithm however before choosing a superior algorithm, it is important to understand the basics of machine learning, as this is the chosen method of improvement.

There are two types of machine learning: supervised and unsupervised learning. Supervised learning uses labelled data for classification and regression whereas unsupervised learning uses unlabelled data for clustering, dimensionality reduction and association rule learning.

As unsupervised learning is not the most appropriate for classification, it is best to use supervised learning to ensure maximum accuracy for the data that has already been collected. For supervised learning, the metric calculations are rather useful as these provide useful data about the geometrics of the particle track. These extracted features can be used to determine the class of the particles in the test dataset. It would also be possible to use the raw images of each blob with their corresponding labels to train a classifier to accurately predict the class.

The most suitable machine learning method to solve this task was a neural network as the hyperparameters provide the flexibility for continual modification and the various architectures will allow me to explore the field of artificial intelligence. However, I will also be using other supervised learning techniques as a method of evaluating the outcome of the development of my neural networks.

# Research

## Introduction

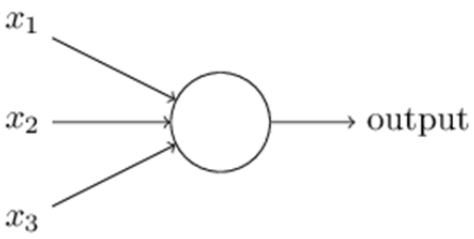
Neural networks are a machine learning method used to solve data classification problems. Neural networks learn by creating a training model using a training dataset containing an array of inputs with their expected outputs. This type of machine learning is known as supervised machine learning. The training dataset is passed through a series of interconnected nodes and the features of the training dataset are interpreted and learned by the mathematical modification of the neural weights for each individual node. This neural model can then check a given input to give an accurate representation of the expected output.

## Nodes

### Perceptrons

A perceptron is a node that accepts a series of binary inputs, weights them according to their relative importance and then produces a binary output based on a threshold.

$$f(x) = \begin{cases} 1 & if \quad \sum_{i=0}^{m} w_i\, x_i + b \geq t \\ 0 & if \quad \sum_{i=0}^{m} w_i\, x_i + b < t \end{cases}$$




- $x$ is a binary input (0 or 1)
- $m$ is the number of inputs provided to the perceptron
- $b$ is the bias (tilting the result to one side depending on the magnitude of the value)
- $w$ is the weight (the relative importance of the input) supplied to the algorithm for each input

- $t$ is the threshold which changes the output value upon being exceeded
- $f(x)$ is the output given by the perceptron (can only be a 0 or a 1)

The equation above shows the mathematics behind a single perceptron. This in parallel with other perceptrons can be used for decision-making. The bias and the weights are independent and can be initialised randomly or with zero value.

### Sigmoid Neurons

A sigmoid neuron is similar to a perceptron however, this node can take on any values between 0 and 1 (it is not a binary classifier). This is because the sigmoid neuron uses a sigmoid graph for classification and therefore the output values are not binary. The mathematics behind this neuron is very similar to that of the perceptron.

$$f(x) = \sigma\left(\sum_{i=0}^{m} w_i\, x_i + b\right) \quad where \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

- $x$ is an input (any value between 0 and 1)
- $m$ is the number of inputs provided to the perceptron
- $b$ is the bias (tilting the result to one side depending on the magnitude of the value)
- $w$ is the weight (the relative importance of the input) supplied to the algorithm for each input
- $f(x)$ is the output given by the sigmoid neuron (can be any value between 0 and 1)

Instead of a threshold, the sigmoid neuron simply outputs the final output from the neuron using a sigmoid function. The function that is used for a neural network node is called the activation function or the transfer function. The sigmoid function $\sigma(z)$ that is being used in the mathematical equation (shown above), is the logistic function, hence this sigmoid neuron is also called the logistic neuron. The input $z$ into the activation function $\sigma(z)$ can also be mathematically expressed as the dot product of two vectors $w$ and $x$ using the dot product formula [7]. This would simplify the expression of the logistic neuron to the following:

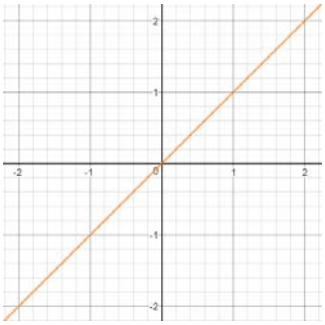
$$f(x) = \sigma(w \cdot x + b) \quad where \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

## Activation Functions

An activation function is just a function that is assigned to each node which determines the presence of a particular feature and then uses the output to make a decision. Its purpose is to create a partition between the data points to make classification as accurate as possible. A sigmoid neuron utilises the sigmoid function while a basic perceptron uses a threshold value. The most commonly used activation functions for a sigmoid neuron are the logistic function or the hyperbolic tan function. The perceptron is fundamentally a neuron that uses the unit step (Heaviside function) or the sign function (signum function) as its activation function where the threshold is the value at which the output changes.

### Linear

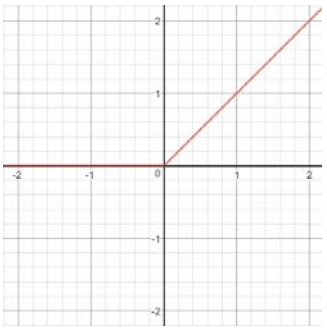
$f(x) = x$



Activation functions cannot be linear because neural networks with a linear activation function are effective only one layer deep (performing linear regression). Real world problems are generally non-linear and to make the input data nonlinear, a nonlinear mapping called the activation function is used. Non-linearity is needed in activation functions because its aim in a neural network is to produce a nonlinear decision boundary via non-linear combinations of the weight and inputs.

### Rectified Linear (ReL)

$$f(x) = \max(0, x)$$



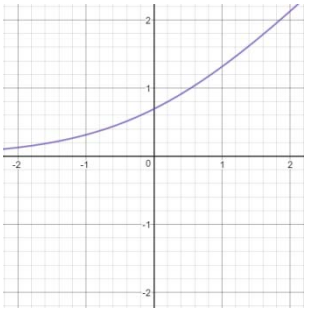
ReLUs (Rectified Linear Units) have two benefits.

1. They are very fast.
2. They do not suffer from the vanishing gradient problem.

ReLUs are faster to compute because they do not require any normalization or any exponential computation (such as those required by the logistic and hyperbolic tangent activations). It is also useful as it is an approximation for the softplus function.

### Softplus

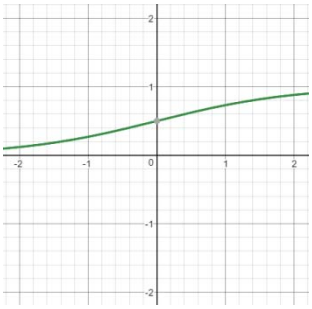
$$f(x) = \ln(1 + e^x)$$



The softplus function is differentiable everywhere unlike the rectifier linear function and does not completely saturate. Although theoretically softplus has an advantage, empirically ReLUs are not only much faster but they have also shown to work with a greater accuracy.

### Logistic (Sigmoid)

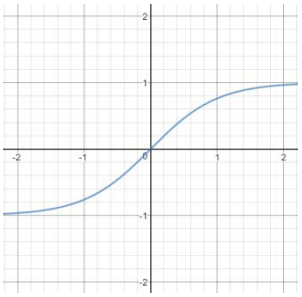
$$f(x) = \frac{1}{1+e^{-x}}$$



For the logistic function, the data usually needs to be normalised to prevent large inputs from resulting in a shallow gradient near zero. This will cause very slow or no learning during backpropagation as the weights will be updated with small values.

### Hyperbolic Tangent

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



The hyperbolic tangent function also requires the data to be normalised to prevent large inputs from resulting in a shallow gradient near zero. This will cause very slow or no learning during backpropagation as the weights will be updated with small values.

### Softmax

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{N} e^{x_k}} for\ j = 1, \ldots, N$$

The softmax function is also known as the normalized exponential function and it is a generalization of the logistic function that converts a K-dimensional vector into a vector of real values in the range (0, 1). This activation function is only used for the output layer of a neural network to provide the class predictions where 1 can be interpreted as 100% confidence and 0 can be interpreted as 0% confidence.

## Error Functions

Error functions are also known as loss or cost functions and they are used to map values onto a real number that represents a "cost" associated with the input value. The aim of a neural network is to solve this optimisation problem by trying to minimise the output value of the cost function.

### Mean Absolute Error Function (MAE)

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |a(x_i) - y_i|$$

The mean absolute error is the mean of the absolute errors where the absolute error is determined by the magnitude of the difference between the prediction output from the neural network $a(x_i)$ and the correct expected value $y_i$.

### Mean Squared Error Function (MSE)

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (a(x_i) - y_i)^2$$

The mean squared error is the mean of the square of the absolute error where the absolute error is determined by the magnitude of the difference between the prediction output from the neural network $a(x_i)$ and the correct expected value $y_i$.

### Sum Absolute Error Function (SAE)

$$SAE = \sum_{i=1}^{n} |a(x_i) - y_i|$$

The sum absolute error is the sum of the absolute error where the absolute error is determined by the magnitude of the difference between the prediction output from the neural network $a(x_i)$ and the correct expected value $y_i$.

### Sum Squared Error Function (SSE)

$$SSE = \sum_{i=1}^{n} (a(x_i) - y_i)^2$$

The sum squared error is the sum of the square of the absolute error where the absolute error is determined by the magnitude of the difference between the prediction output from the neural network $a(x_i)$ and the correct expected value $y_i$.

### Cross-Entropy Error Function (CEE)

$$CEE = -\frac{1}{n} \sum_{i=1}^{n} y_i ln(a(x_i)) + (1 - y_i) ln(1 - a(x_i))$$

Cross-entropy [8] is used as an error measure when a network's outputs can be thought of as representing independent hypotheses where each node stands for a different concept, and the node activations can be understood as representing the probability that each hypothesis is true. In that case, the output vector represents a probability distribution, and the error function represents the distance between the network prediction and the expected output. Therefore, the cross-entropy error function is primarily used when the softmax activation function is used for the output layer.

## Optimisation Algorithms

A neural network learns by minimising the error function for a given dataset, where the error function evaluates how well certain data fits a given neural model of stored weights. The error function is minimised by adjusting the synaptic weights and biases which can then be congregated into a single n-dimensional vector $w$.

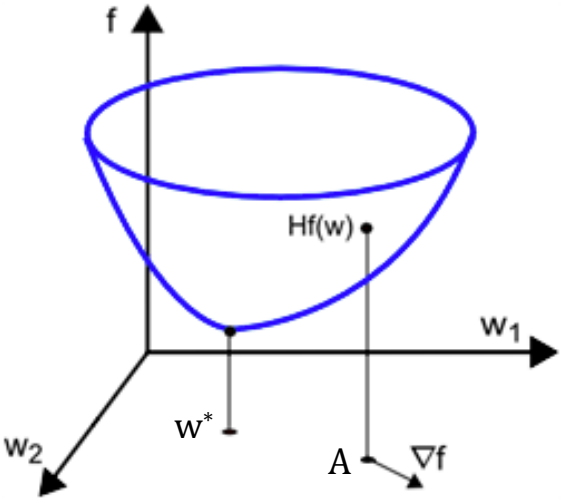



Figure 11 - Error Surface

Figure 11 refers to the error function $f(w)$ where $w$ is the combined weight vector. At any point $A$ on the surface plot, the following first and second order vectors can be derived:

$$\nabla_i f(w) = \frac{df}{dw_i} \ (i = 1, \dots, n)$$

$$\mathrm{H}_{i,j} f(w) = \frac{d^2 f}{dw_i \cdot dw_j} \ (i, j = 1, \dots, n)$$

The aim of deriving these vectors is to use them to find the vector $w^*$ where the error function has a minimum value. As the graph is non-linear and multidimensional, the only method of finding the minimum point is to use search algorithms based on the gradient vector and/or the hessian matrix. The algorithms used for this learning procedure are known as the training optimisation algorithms of a neural network. Below I have listed, explained and evaluated all the commonly used training algorithms [9] that I have considered in my research where $x_n$ is the weight vector $w_i$.

In the context of the neural network, the weight vector $w_{i,j}$ (where $w_{i,j}$ is $x_n$) is initialised with small random values. The input is forward passed into the neural network and the prediction values are stored. The error is computed for the difference between the prediction and the actual expected values using an error function. The new weights for the layers are computed by calculating the gradients of the cost function using a gradient optimisation algorithm. The weights are then updated using an update rule. This process of optimising and updating the weights is known as backpropagation. This is repeated until the error from the improvement is minimal. This is the basis of the neural network machine-learning algorithm.

### Gradient Descent (GD)

The gradient descent is the simplest algorithm for iteratively finding the minimum point of a function. It is a first-order method meaning that it only uses the first derivative of the function.

#### *One-Dimensional Method*

$$x_{n+1} = x_n - \alpha \, f'(x_n)$$

A point $x_0$ is picked as a starting value and the gradient is calculated for the current value. $g_n = f'(x_n)$ To improve the conjecture, the point needs to move by $\alpha$ (the training rate) in the direction of that gradient:

$$x_{n+1} = x_n - \alpha \, g_n$$

This iteration is repeated until the function is close enough to zero (until $f(x_n) < t$ for some small threshold).

#### *Multi-Dimensional Method*

$$x_{n+1} = x_n - \alpha \, \nabla f(x_n)$$

The multi-dimensional method is derived from the one-dimensional method where the first derivative is replaced with the gradient of the original function $f'(x) \to \nabla f(x)$. For both methods, $\epsilon$ is a fixed training rate with a small value greater than 0.

A point $x_0$ is picked as a starting value and the gradient is calculated for the current value. $g_n = \nabla f(x_n)$ To improve the conjecture, the point needs to move by $\alpha$ (the learning rate) in the direction of that gradient:

$$x_{n+1} = x_n - \alpha\, g_n$$

This iteration is repeated until the function is close enough to zero (until $f(x_n) < t$ for some small threshold).

### *Variations of Update Rules*

In all of the below update rules [10], the variables are defined as the following:

$\epsilon$ – Small value for numerical stability also known as the smoothing constant
$x_n$ – The weights of the neural network
$\nabla f(x_n)$ – The gradient of the error function
μ – Momentum constant
γ – Decay constant
$\alpha$ – Learning rate
$m_n$ – 1st Order Moment of $\nabla f(x_n)$
$g_n$ – 2nd Order Moment of $\nabla f(x_n)$

Stochastic Gradient Descent

$$x_{n+1} = x_n - \alpha \nabla f(x_n)$$

Negative → Outliers can lead to an inefficient learning process.

Stochastic Gradient Descent with Momentum

$$v_{n+1} = \alpha \nabla f(x_n) + \mu v_n$$

$$x_{n+1} = x_n - v_{n+1}$$

Negative → Slower convergence than more advanced methods below.

Adaptive Gradient Descent (AdaGrad)

$$g_{n+1} = g_n + \nabla f(x_n)^2$$

$$x_{n+1} = x_n - \frac{\alpha}{\sqrt{g_{n+1} + \epsilon}} \nabla f(x_n)$$

Negative → Update size is lowered very aggressively.

Adaptive Delta (AdaDelta)

$$g_{n+1} = \gamma\, g_n + (1-\gamma) \nabla f(x_n)^2$$

$$r_{n+1} = \gamma\, r_n + (1-\gamma)\, {v_{n+1}}^2$$

$$v_{n+1} = -\frac{\sqrt{r_n + \epsilon}}{\sqrt{g_{n+1} + \epsilon}} \nabla f(x_n)$$

$$x_{n+1} = x_n + v_{n+1}$$

Negative → Moving averages are biased by initialization of the decay constants.

Root Mean Squared Propagation (RMSProp)

$$m_{n+1} = \gamma\, m_n + (1-\gamma)\, \nabla f(x_n)$$

$$g_{n+1} = \gamma\, g_n + (1-\gamma)\, \nabla f(x_n)^2$$

$$v_{n+1} = \mu v_n - \frac{\alpha}{\sqrt{g_{n+1} - m_{n+1}{}^2 + \epsilon}} \nabla f(x_n)$$

$$x_{n+1} = x_n + v_{n+1}$$

Negative → Updates are scaled differently across batches.

Adaptive Moment Estimation (Adam)

$$m_{n+1} = \gamma_1 m_n + (1 - \gamma_1) \nabla f(x_n)$$

$$g_{n+1} = \gamma_2 g_n + (1 - \gamma_2) \nabla f(x_n)^2$$

$$\hat{m}_n = \frac{m_n}{1 - \gamma_1{}^n}$$

$$\hat{g}_n = \frac{m_n}{1 - \gamma_2{}^n}$$

$$x_{n+1} = x_n - \frac{\alpha}{\sqrt{\hat{g}_n + \epsilon}} \hat{m}_n$$

Negative → It is much slower than gradient descent at converging for easy problems.

### *Limitations*

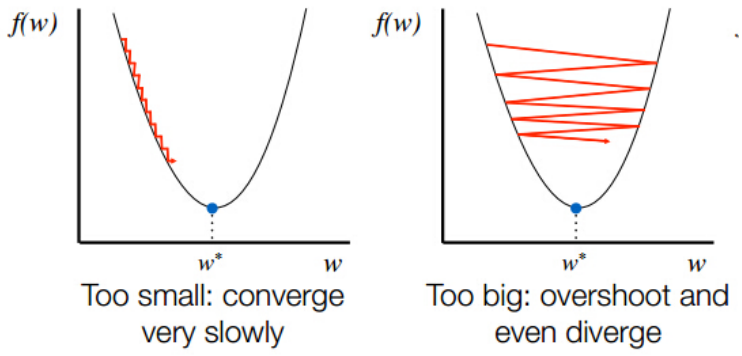



Figure 12 - Convergence & Divergence

The gradient descent method only uses the first derivative and this is a major limitation of this method as we are assuming the error surface to be a plane instead of a curved surface. This can cause the training process to be extremely slow as it reaches a local minimum. This method also requires numerous iterations for functions that have long, narrow valley structure and therefore gradient descent is not used in such cases. Moreover, if the learning rate is too slow the convergence will be too slow and if the learning rate is too big, it may diverge.

## Newton-Raphson Method

The Newton-Raphson method is an iterative method for finding roots (where $f(x) = 0$) to a differentiable equation such as a polynomial. For the purpose of a training algorithm for a neural network, the aim of the algorithm is to find the minimum point. This can be performed by applying the Newton-Raphson method to the derivative of the original function, to find the roots of the derivative (solutions to $f'(x) = 0$), also known as the stationary points of the function.

### *One-Dimensional Method*

The one-dimensional Newton-Raphson method is derived from the formula for a tangent at a given point ($x_1$, $y_1$).

$$1)\ y - y_1 = m\,(x - x_1)$$

$$2)\ y - f(x_1) = f'(x_1)\,(x - x_1) \qquad 3)\ \frac{y - f(x_1)}{f'(x_1)} = x - x_1 \qquad 4)\ \frac{y - f(x_1)}{f'(x_1)} + x_1 = x$$

To work out the root of the rearranged equation, y is set to 0.

$$x = x_1 - \frac{f(x_1)}{f'(x_1)}$$

This gives the following iteration formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

By iterating through this formula, the output value will eventually converge to a point at which y = 0 (which is the root of the function). To find the minimum point of a given function, the following formula is derived from the equation above where the original function f(x) is twice differentiable:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

The above formula is resultant from the following:

Given a function $f(x)$, to find the minimum point we compute the second-order Taylor Series [11] for the function $f(x)$ where $x = x_0 + \varepsilon$ where $x_0$ is some point and ε is some value such that $f(x_0 + \varepsilon) = 0$.

$$f(x_0 + \varepsilon) \approx f(x_0) + f'(x_0)\,\varepsilon + f''(x_0)\,\frac{\varepsilon^2}{2}$$

For $f(x_0 + \varepsilon)$ to be a minimum point the derivative of $f(x_0 + \varepsilon)$ has to be equal to 0. Therefore, differentiating the Taylor Series approximation gives the following equation:

$$\frac{d}{dx}\left(f(x_0) + f'(x_0)\,\varepsilon + f''(x_0)\,\frac{\varepsilon^2}{2}\right) = 0$$

$$2f'(x_0) + 2f''(x_0)\,\varepsilon + f'''(x_0)\,\frac{\varepsilon^2}{2} = 0$$

Using only up to the second derivative from the differentiated Taylor Series, the following equation is produced.

$$f'(x_0) + f''(x_0)\,\varepsilon = 0$$

$$\varepsilon = -\frac{f'(x_0)}{f''(x_0)}$$

The above equation gives the minimum point of any given quadratic function. However, it does not work for other non-linear functions. The formula needs to be iterative for it to output the approximations to the minimum point. Therefore, the final formula computes the minimum point for a one-dimensional function. $f: \mathbb{R} \to \mathbb{R}$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

### *Multi-Dimensional Method*

For a multidimensional function $f: \mathbb{R}^n \to \mathbb{R}$, the same formula is used but the derivatives are replaced with gradients and the second derivatives are replaced with Hessians (matrix of second derivatives).

$$f'(x) \to \nabla f(x)$$

$$f''(x) \to H(f)(x)$$

This gives the following formula for a multi-dimensional function:

$$x_{n+1} = x_n - \frac{\nabla f(x_n)}{H(f)(x_n)}$$

### *Limitations*

- The required derivative can be very difficult to calculate.

- The method will fail to converge at the minimum point unless the function has a quadratic Taylor expansion near an optimum.
  - Stationary points will result in the derivative being 0, giving a zero division error.
  - An inaccurate preliminary estimate may prevent the algorithm from converging.
  - To prevent non-convergence, the function can be bound within a small known interval.

### *Conclusion*

The Newton-Raphson method tends to converge faster than conjugate gradient methods however, it is very complex and computationally expensive to compute the Hessian matrix. Moreover storing the full Hessian matrix uses $\Theta(n^2)$ memory, which is infeasible for high-dimensional functions such as the loss functions of neural networks and other statistical models with large numbers of parameters.

## Conjugate Gradient (CG)

The conjugate gradient algorithm is in between the gradient descent and the Newton-Raphson algorithms in terms of speed and convergence. The conjugate gradient algorithm [12] is a multi-dimensional optimisation method where the one-dimensional concept is similar to gradient descent. It does not use the Hessian matrix and so it is not as computationally expensive as the Newton-Raphson method. However, it does calculate conjugate directions implicitly with respect to the Hessian matrix [13] producing faster convergence than gradient descent.

### *Multi-Dimensional Method*

$$f(x) = \frac{1}{2}\, x^T A x + \mathrm{b}^{\mathrm{T}}\mathrm{x} + \mathrm{c}, \qquad \mathrm{x} \in \mathbb{R}^{\mathrm{n}}, \qquad \mathrm{A} \in \mathbb{R}^{\mathrm{nxn}}, \qquad \mathrm{b}, \mathrm{c} \in \mathbb{R}^{\mathrm{n}}$$

$$\nabla f(x) = Ax + b$$

By evaluating $-\nabla f(x)$ at any given location, a vector pointing towards the direction of steepest descent is calculated. An initial guess $x_0$ is picked, the gradient $-\nabla f(x_0)$ is computed, and we move in that direction by some step size $\alpha$. Unlike normal gradient descent, a line search is performed to find the best step size $\alpha$ instead of having a fixed value.

$$g(\alpha) = f(x + \alpha d_0) \; where \; d_0 = -\nabla f(x_0)$$

$$g(\alpha) = \frac{1}{2}\,(x + \alpha d_0)^T A (x + \alpha d_0) + \mathrm{b}^{\mathrm{T}}(x + \alpha d_0) + \mathrm{c}$$

This is a quadratic equation with the variable $\alpha$ and it is therefore assumed that the function has a minimum point given that it is not at a saddle point. The minimum occurs when the differential of the equation equals 0.

$$g'(\alpha) = ({d_i}^T A d_i)\alpha + {d_i}^T (A x_i + b) = 0$$

Solving the above equation for $\alpha$ gives the following solution where $\alpha$ is positive as the direction is the negative of the gradient: (Although the equation for $\alpha$ can be used to calculate $\alpha$, a line search is used as it is simpler to perform.)

$$\alpha = -\frac{{d_i}^T (A x_i + b)}{{d_i}^T A d_i}$$

$$x_1 = x_0 - \alpha \nabla f(x_0)$$

If this was a simple gradient descent process, this equation could be iterated to find the minimum value however by moving $a_i$ in direction $d_i$, it is possible that the equation never converges as moving $a_{i+1}$ in direction $d_{i+1}$ may destroy the work from the previous iteration. To prevent this from happening, the directions need to be conjugate to each other. Two vectors x and y are conjugate to each other with respect to some semi-definite matrix $A$ if $x^T A y = 0$. Semi-definite matrices are matrices which obey $x^T A x \geq 0$ where $x$ is real.

Let the conjugate direction be called $d_1$ which starts with the gradient at $x_1$ and subtracts anything that counteracts the previous direction (which in this case is $d_0$) where $\beta_0$ is the magnitude of how much it counteracts. This gives the following equation:

$$d_1 = -\nabla f(x_1) + \beta_0 d_0$$

As $d_0$ and $d_1$ are conjugate to each other, the following must be true:

$$d_1{}^T A d_0 = 0$$

$$d_1{}^T A d_0 = (-\nabla f(x_1) + \beta_0 d_0)^T A d_0 = -\nabla f(x_1)^T A d_0 + \beta_0 d_0{}^T A d_0 = 0$$

Reorganising this to work out $\beta_0$:

$$\beta_0 = \frac{\nabla f(x_1)^T A d_0}{d_0{}^T A d_0}$$

Choosing the value of β gives us a direction conjugate to all previous directions. Iterating this will keep giving us conjugate directions. After generating each direction, it is possible to find the best $\alpha$ for that direction and update the current estimate of position.

Since the difference between $x_{k+1}$ and $x_k$ is in the direction $d_k$ with a magnitude of some constant $c$, the following equations can be obtained:

$$x_{k+1} - x_k = c d_k$$

$$\therefore \nabla f(x_{k+1}) - \nabla f(x_k) = (A x_{k+1} + b) - (A x_k + b) = A(x_{k+1} - x_k) = c A d_k$$

$$\therefore A d_k = c^{-1}(\nabla f(x_{k+1}) - \nabla f(x_k))$$

By substituting the above equation into the equation for calculating β, the simplified form can be written as:

$$\beta_k = \frac{\nabla f(x_{k+1})^T (\nabla f(x_{k+1}) - \nabla f(x_k))}{d_k{}^T (\nabla f(x_{k+1}) - \nabla f(x_k))}$$

The final update rule can be written as the following:

$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k$$

### *Variations of Update Rules*

The different variations in the conjugate gradients are due to the various methods [14] of calculating $\beta_k$. The one showed in the proof above is the Hestenes-Stiefel conjugate gradient. These formulae are equivalent for a quadratic function, but for nonlinear optimization, the formula is chosen based on heuristics.

Fletcher-Reeves Conjugate Gradient

$$\beta_k = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)}$$

Polak-Ribiere Conjugate Gradient

$$\beta_k = \frac{[\nabla f(x_{k+1}) - \nabla f(x_k)]^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)}$$

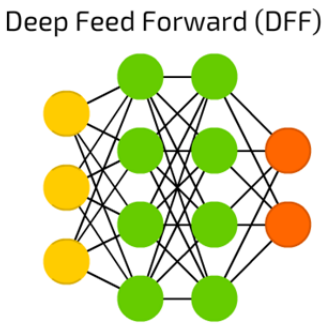
### *Limitations*

- Although more accurate, the conjugate gradient algorithm is very slow for general neural networks.
- Relatively difficult to understand intuitively.
- For small networks, it is quicker and simpler to use gradient descent.

## Architectures

The fast development in the field of machine learning and artificial intelligence means that different neural architectures are being invented very frequently. This means that there are several variations of names and abbreviations for the same or similar neural architectures [15].

### Deep Feed Forward Network




A deep feed forward network is also known as a multi-layer feed forward neural network and it is the basic neural network architecture used to build up other neural networks with far more advanced uses. In a deep feed forward network, there are three types of layers: the input layer, the hidden layers and the output layer. The input layer receives inputs from a training dataset and propagates it to the hidden layers. The neural model is created by adjusting the weights on the connections to the hidden layers by backpropagation until the error of the neural network is minimal for the corresponding outputs of the given dataset. The neural model is stored as a series of weights and can be used again to classify inputs for an unknown output.

#### *Layered Structure*

For a neural network with 2 hidden layers with the activation functions being sigmoid function for the hidden layers and then softmax in the output layer generates the following equations:

$$x_i = Input\ at\ Node\ i$$

$$f_j = \sigma(w_j \cdot x_i + b_j) \quad where \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

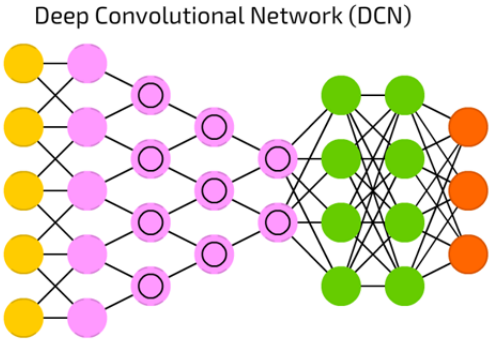
$$f_k = \sigma(w_k \cdot f_j + b_k) \quad where \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$y_l = \rho(w_l \cdot f_k + b_l) \quad where \quad \rho(z) = \frac{e^{z_k}}{\sum_{p=1}^{N} e^{z_p}} for\ l = 1, \dots, N$$

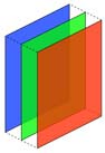
$$y_l = Output\ at\ Node\ l$$

The above equations intuitively explain how a neural network works. Although these equations explain the structure of a neural network they do not demonstrate how they are trained (how the weights are updated).

### Deep Convolutional Network




A deep convolutional neural network [16] is essentially a neural network for image classification where the input is an image matrix (a 3-dimensional matrix with dimensions $h * w * d$).



Each image input matrix is such that the height and width is the height and width of the image and the depth is the number of colour components. (RGB is 3)

There are two different types of layers for image processing in a deep convolutional network: the convolutional layer and the subsampling layer.

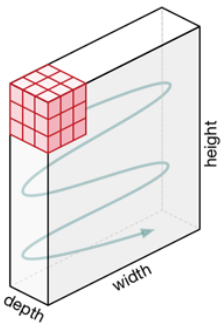



Figure 13 - Convolution

#### *Convolutional Layers*

Convolutional layers run a convolution kernel on the 3D tensor where the convolutional kernel is also 3-dimensional with a small value for width and height but the depth being the same as the input image. In Figure 13 [17], the red cube is slid across the entire volume of the input image and at each position, the 27 elements are summed up and written to the output. The weights here belong to the convolution kernel and as there are

27 elements, there will be 27 weights for this kernel. As in a deep neural network, the weights describe the strength of the connection between an input and output. Using only this convolution kernel, the output matrix then has the same width and height as the original but the depth is now 1. However, each convolutional layer has more than one convolution kernel and therefore the output volume is a matrix with the same width and height but the depth being the number of convolution kernels.

### *Subsampling Layers*

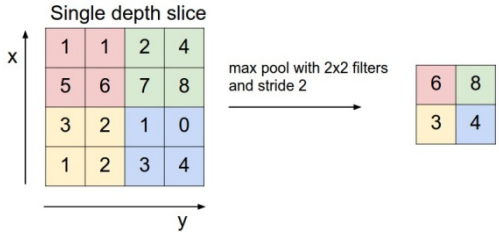



Figure 14 - Subsampling

Subsampling layers are needed to reduce the size of the input to allow focussing on significant features. There are multiple ways to subsample, but the most popular are max pooling, average pooling and stochastic pooling. For subsampling layers, a fixed subsampling region is defined (Figure 14).

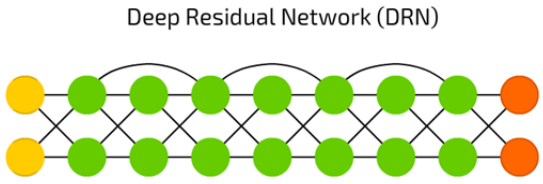
- Max pooling takes the maximum value in the region.
- Average pooling takes the average in the region.
- Stochastic pooling uses the normalisation of an activation function to generate probabilities to weight the values in the region.

Taking an example, if a pooling layer has a subsampling region of 2x2 and the input consists of a 16x16 matrix then the output would be a 8x8 matrix, meaning that 4 pixels (each 2x2 square) of the input matrix are combined into a single output pixel using one of the pooling processes (Figure 14).

### *Fully-Connected Layers*

The last layer of the convolutions and pooling is usually connected to one or more fully connected layers, the last of which is the output layer representing the target data. Training is performed using modified backpropagation that takes the subsampling layers into account and updates the convolutional filter weights based on all values to which that filter is applied. This is essentially a deep forward neural network attached to the end of an image reduction process.

## Deep Residual Network




Deep residual networks are neural networks made up of interconnected blocks of neural networks. Therefore, the old input for the first layer can be re-entered into the neural network three layers later so that different features are learned at each layer. This makes this neural network highly efficient at learning patterns. Moreover, the interconnected layers mean that removing a single block simply reduces the accuracy in a proportional manner. However, removing a layer in a deep feed forward neural network can change the accuracy completely. Deep residual networks tend to be very deep and can use convolutional and pooling layers for image classification tasks.

### *Residual Blocks*

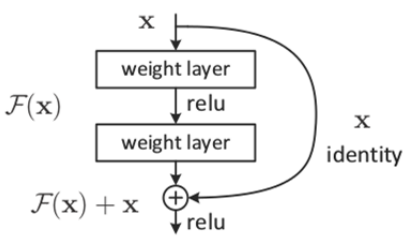



Figure 15 - Residual Block

Residual networks are made up of residual blocks. Each residual block has multiple weighted layers (Figure 15) [18] and as residual networks tend to be used for image classification, these weighted layers tend to be convolutional layers. The greater the number of blocks, the more the network learns and therefore the accuracy is usually relatively proportional to the depth of the network.

## Benchmark Classifiers

To evaluate the best method for the classification of particles, it was necessary to research other common methods for machine learning to compare the results with the neural networks. As these classifiers were not the focal point of the project, my research is more concerned about the evaluation of these classifiers rather than how they function.

### Decision Tree

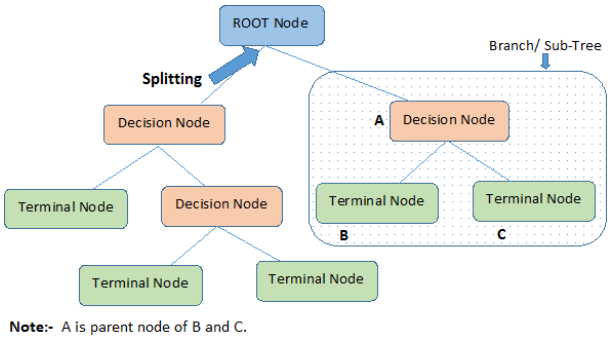



Figure 16 - Decision Tree

Classification and regression trees (CART) use decision trees for machine-learning. The number of features in the training data are used as the number of candidate splits. Using a cost function, the accuracy cost of each split is calculated and the split with the least cost is chosen. So, the primary root node of the decision tree tends to be the discriminating feature for most of the data (Figure 16) [19]. The algorithm is recursive and therefore the groups can be sub-divided in the same manner. The other features are used as the decision nodes to improve the class prediction accuracy.

Advantages:

- ✓ Decision trees quickly identify most significant variables and relation between multiple variables.
- ✓ It requires less data cleaning compared to some other modelling techniques and it is not greatly influenced by outliers or missing values.
- ✓ It can handle both numerical and categorical variables.
- ✓ Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages:

- Over-fitting is one of the most practical difficulties for decision tree models. However, it can be solved by setting constraints on model parameters and pruning.
- While working with continuous numerical variables, decision tree loses information when it categorizes variables.

### Random Forest

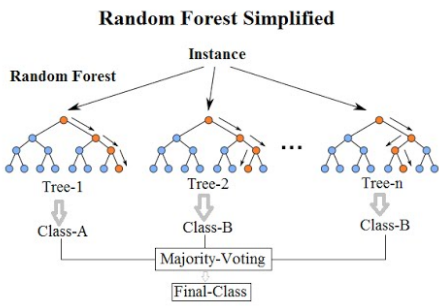



Figure 17 - Random Forest

A problem with decision trees is that they use a “greedy” algorithm to minimise error. Hence, multiple decision trees are often used to validate the accuracy (known as bagging). However even with bagging, the decision trees can have many structural similarities and in turn have high correlation in their predictions and therefore random forest classifiers are used as an ensemble method to improve accuracy. The random forest classifier once again uses multiple decision trees (Figure 17) [20] but it limits the search to a random sample of features so that the learned sub-trees are less similar and hence the resulting predictions have less correlation.

Advantages:

- ✓ Random forests are useful for both classification and regression.
- ✓ They balance errors due to skewed datasets and imbalances in class.
- ✓ They can easily handle high dimensionality data.
- ✓ Bootstrap sampling provides an effective method for estimating missing data.

Disadvantages:

- Overfitting can lead to inaccurate predictions for noisy datasets.
- Regression problems are only resolved within the range of the training dataset.

## K-Nearest Neighbour

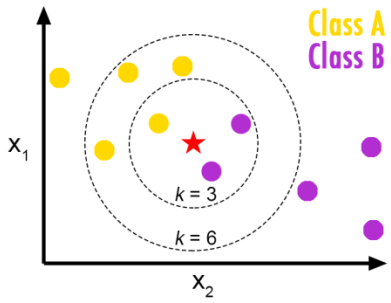



Figure 18 – K-Nearest Neighbour

The k-nearest neighbour classifier is very easy to interpret and to use for classification. The training data is plotted as a series of vectors and when given a test data, the algorithm simply determines k closest data points from the training dataset (Figure 18) [21].This is fundamentally used as the prediction where k is a hyperparameter to be determined. The classifier itself can use various distance measures for finding the nearest neighbour. The most common ones include Euclidean, Hamming, Minkowski and Manhattan distance measures.

Advantages:

- ✓ It is very robust to noisy training data.
- ✓ Highly effective if the training data is large.

Disadvantages:

- It can be very computationally expensive as the distance between each query instance and all the training samples need to be calculated.
- Optimal hyperparameters such as the number of nearest neighbours (K) and the type of distance measure are difficult to determine.
- When the class distribution is skewed “majority voting" classification can be inaccurate as they tend to be common among the k-nearest neighbours due to their large number.

## Support Vector Machines

Unlike neural networks, which have multiple output neurons, a SVM always has one single output. However, this does not mean that you cannot do multi-class classification with a SVM. Multi-class classification is the usual reason for having multiple output neurons in a neural network however for a SVM the classes can be encoded as ordinal values between a specific range.

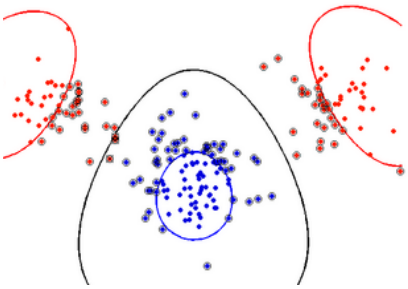


Figure 19 - RBF Kernel

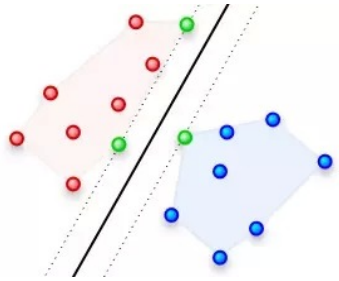


Figure 20 - Linear Kernel

The SVM algorithm tries to find the optimal separating hyperplane (multi-dimensional plane with n-1 dimensions where n is the number of dimensions of the training data) that maximizes the margin of the training data. Instead of an activation function, the SVM uses a kernel, which acts like an activation function. The kernel is usually a linear function (Figure 20) [22] however, for the purpose of particle classification I will be using the radial basis function (Figure 19) [23] assuming that the data is not linearly separable. The higher the value of the kernel coefficient, the SVM will try more to exact fit the training dataset. Adapting this value is used to calibrate the generalization error and prevent over-fitting. When a linear function is used, the gradient descent optimisation method is used to minimise the loss and improve classification accuracy.

Advantages:

- ✓ It works really well with clear margin of separation.
- ✓ It is effective in high dimensional spaces.
- ✓ It uses support vectors to make it more memory efficient.

Disadvantages:

- Performance is weaker when dataset has target classes that overlap.
- SVM does not directly provide probability estimates. These are calculated using an expensive five-fold cross-validation.

# Development

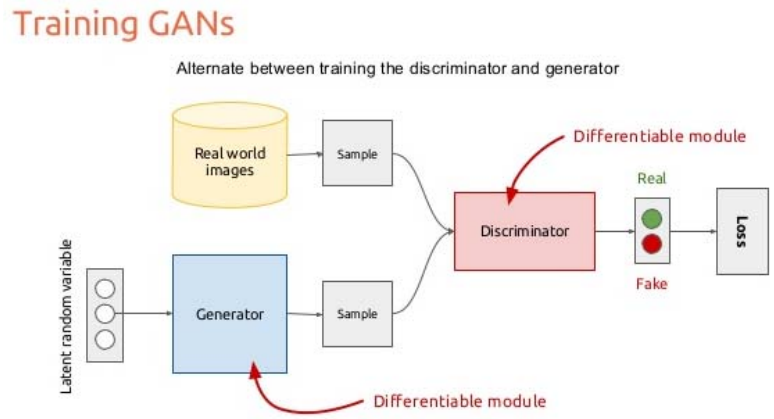
## Training Data Production

### Method Selection

To produce the training data for the neural network, I had several options:

1. Use LUCID data with a platform (like Zooniverse [24]) for user input classification ("crowd" science or citizen science) to create a database matching particle tracks to the classification provided by users. The advantage of this is that it would use real-life data with human predictions for classification. The disadvantages are that human opinions can be diverse in debatable situations and therefore the classifications made from multiple people can lead to inaccuracies in the training data. Moreover, this method requires a lot of time as it has to be done manually and cannot be done computationally.

2. Generative modelling of different particle tracks using Generative Adversarial Networks (GANs) (Figure 21) [25]. So providing the GAN with several alpha particle tracks would allow us to generate numerous alpha particle tracks with slight variances resembling the real life distribution. This can then be used as the training data for our classification neural network. The disadvantages to this method are that the GAN can require a lot of training to provide the reliable training data needed for the classification neural network and that the generated distribution of data is not always fully accurate to the real-life distribution.




The generator generates data that the discriminator validates. The generator improves itself using the loss output via backpropagation. The real life data is entered into the discriminator to train the discriminator to learn the features that make the data valid.

Figure 21 - Generative Adversarial Network

3. Using data frames created by a human as training data would provide the classification neural network with accurate but minimal data. The limitations of this also includes the fact that human-made particle tracks tend to be almost perfect to the expected tracks and therefore the neural network may fail in recognising more complex tracks. This method takes the longest time by far to generate any substantial or useful training data.

For this part of the project, I decided to use a custom-built platform to collect user classifications for a large quantity of particle tracks since this was the quickest method and required the least data processing. I have recognised and considered the drawbacks of this method as well as the other methods and my conclusion was to develop my neural classification model in the most efficient and quickest way possible.
A platform like Zooniverse is not suitable as there is simply too much LUCID data for it to be manually uploaded and edited into a questionnaire for users to fill in. Instead, the simplest method was to create a custom web application that would dynamically generate the questionnaire and ask users for their classification response. Moreover, the Zooniverse project builder was far more complex than necessary for generating some training data. A custom platform provided the flexibility and the mobility needed to generate the training data quickly and efficiently.

## LUCID Neural Trainer

### *Database Design*

To store the training data, I had to consider the type of storage and the format of the data. The simplest method was to use a SQLite database as it required the least preliminaries in setting up.
The training data mainly needs to contain the blob data with the corresponding user classification, which will be used for training the neural network. There were several methods for storing the data in an SQLite database. The following schematics are shown below:

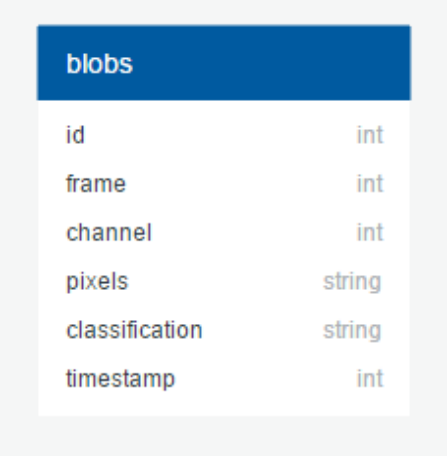



Figure 22 - DB Schema 1

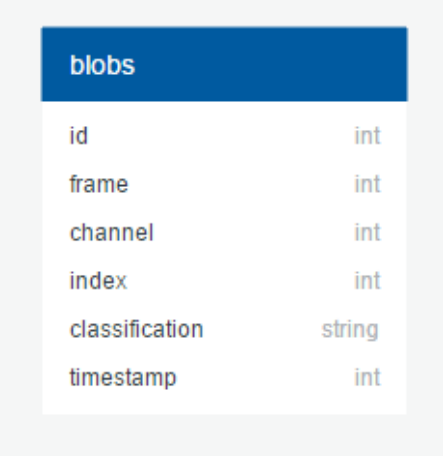



Figure 23 - DB Schema 2

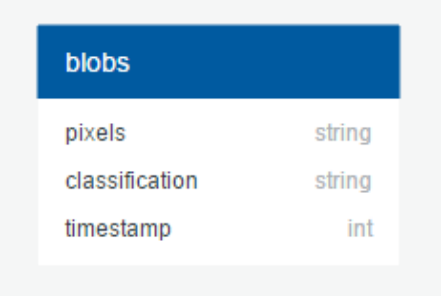



Figure 24 - DB Schema 3

DB Schema 1

**Advantages**

- Data can be reaccessed and checked for redundancies using id, frame and channel.
- The data will never be redundant as the pixels themselves are stored.
- The pixels can be updated when the blobbing algorithm is changed.
- The data can be checked for duplicates.

**Disadvantages**

- It uses the most storage of all the schematics.

DB Schema 2

**Advantages**

- Data can be reaccessed and checked for redundancies using id, frame and channel.
- It uses the least storage as the index of the blob is stored not the blob itself.
- The data can be checked for duplicates.

**Disadvantages**

- The database becomes redundant if the blobbing algorithm is changed.
- Accessing the blob during the training process would be very slow.

DB Schema 3

**Advantages**

- Uses less storage than DB Schema 1

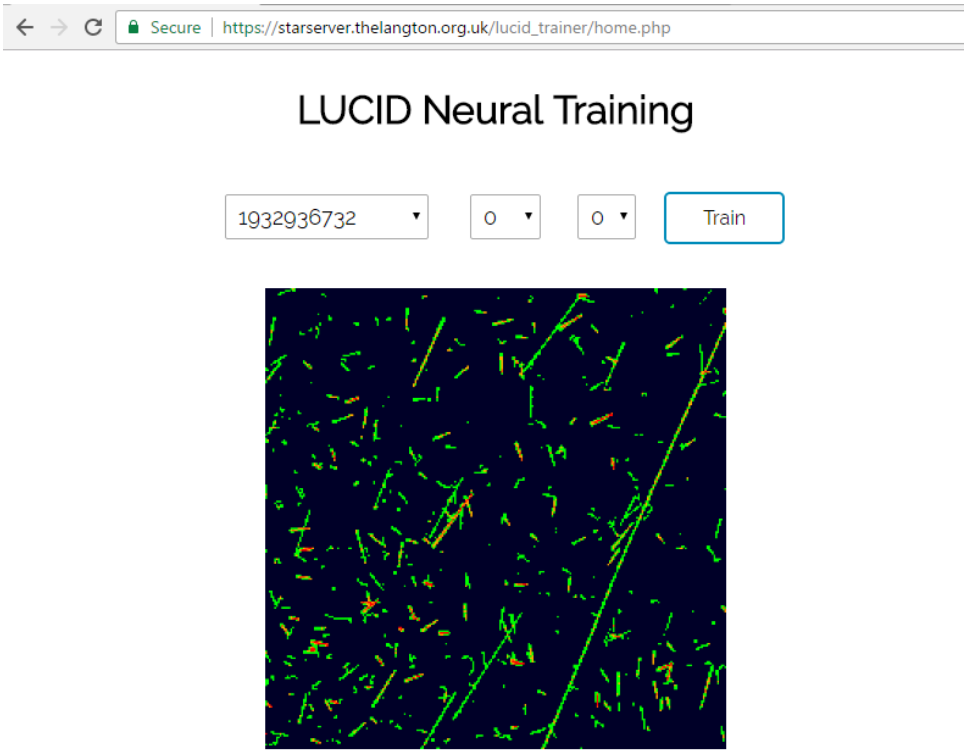
**Disadvantages**

- The data cannot be rechecked in the original frame and therefore cannot be updated or checked for duplicates.
- It still uses a lot of storage for busy frames.

Conclusion

From the above evaluations, I decided to use DB Schema 1 as the only disadvantage it had was that the storage space would be too large for a large amount of data. Given that a large storage space does not necessarily affect my investigation in any way, this method is most appropriate for storing the training data.

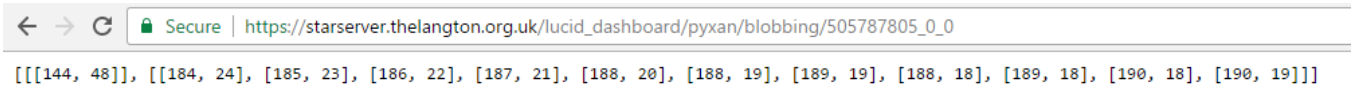
### *Web Application*



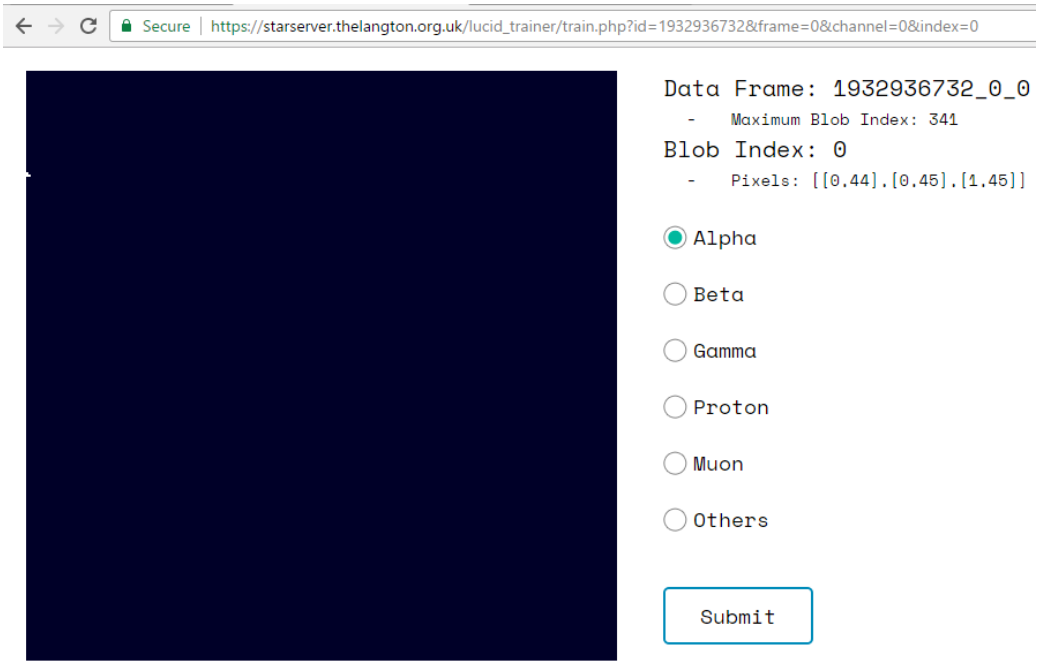

In the home page, the user can select a data file ID. After selecting the data file, the user will then be able to choose a frame number and a channel.

When the user clicks the train button (after having selected the parameters), the user will be redirected to the training questionnaire for the specific frame.

If the frame is empty, the user will be redirected back to the home page.




To display the blob images in the questionnaire, I needed a REST API for the blobbing algorithm as all the scientific programming could only be done in Python and not in PHP. The blobbing API takes the id, frame and channel as the GET parameters and returns a JSON array containing the blobs as arrays of coordinates. This enabled me to use the JSON blobbing data in the PHP questionnaire.




The questionnaire page loops through all the blobs in a specified frame asking the user for the classification. When all the blobs in the specific frame are classified, the user is redirected to the home page to train a new data file.
All the classifications are stored in a SQLite database via AJAX (Asynchronous JavaScript and XML) requests to another PHP file passing the id, frame, channel, pixels and the classification as the POST variables.

# Neural Network Construction

## Architecture Selection

As a part of my experimentation process, I had constructed all the various neural architectures that I had researched. However, from my preliminary constructions, the simplest neural architecture to construct was the deep neural network as the data that I had collected was in the most suitable format for training it and the trial and improvement method for reconstructing the architecture was a much simpler process. The convolutional neural network required the collected data to be pre-processed into images leading to a large single file containing all the image data. This became rather computationally expensive to process during the training of the neural network. The deep residual network simply lacked computing power and the auto-encoders were rather inconsistent in their results leading to invalid evaluations. Therefore, the main neural architecture I decided to focus on was the deep neural network.

## Inputs

### *Selection*

The metrics I selected for the inputs of the neural network included the following:

| | | |
|---|---|---|
| ➢ Number of Pixels<br>➢ Density<br>➢ Radius | ➢ Curvature Radius<br>➢ Line Residual<br>➢ Circle Residual | ➢ Width<br>➢ Average Neighbours |

These values are the ones that were synthesised in the Feature Extraction (Interpretation) process. Metrics specific to each individual particle such as the best-fit angle, centroid and centre of circle were not used, as they would not help generalising the class of the particle.

### *Normalisation*

Once the metrics have been calculated from the data, they usually need to be normalised. Normalisation is the process by which the all the data are made to be within a specific range usually between 0 and 1. However, during my development process, I found that it was not necessary depending on the type of activation function that was used as it only resulted in poorer accuracy. It was also very difficult to normalise some of the metric values as they could vary from zero to a very large number.

## Output Encoding

### *Single Output*

If we were to have a single output then we would encode the data using single nominal values usually between 0 and 1. However, this nominal value may be interpreted as an ordinal value by a machine-learning algorithm.

```
particle_outputs = {
    "gamma": [0.0],
    "beta": [0.2],
    "muon": [0.4],
    "proton": [0.6],
    "alpha": [0.8],
    "others": [1.0]
}
```

This means that the classes may be treated with respect to the magnitude of their assigned value even though the values are being used as labels. This can alter the output value as the different particles are not spectrally distributed. The advantage of this method is that it is very simple to interpret a single output neuron.

### *Multiple Outputs*

```
particle_outputs = {
    "gamma":    [1, 0, 0, 0, 0, 0],
    "beta":     [0, 1, 0, 0, 0, 0],
    "muon":     [0, 0, 1, 0, 0, 0],
    "proton":   [0, 0, 0, 1, 0, 0],
    "alpha":    [0, 0, 0, 0, 1, 0],
    "others":   [0, 0, 0, 0, 0, 1]
}
```

As single outputs can cause inaccuracies, it is sometimes necessary to use a method called one hot encoding to provide multiple outputs from the neural network. One-hot encoding is used to transform categorical features into a numerical format that works better with classification and regression. It is essentially an array of 0s with one 1 in the array where its position in the array denotes the class. This allows for multi-class classification where each of the neurons in the output layer gives each value in the array.

## Hyperparameter Selections

### *Dropout*

Dropout is when one or more neural network nodes will switch off at every training step depending on a provided probability so that it will not interact with the network (its weights cannot be updated, nor affect the learning of the other network nodes). With dropout, the learned weights of the nodes become less sensitive to the weights of the other nodes and become less dependent on the other nodes they are connected to. In general, dropout helps the network to generalize better. It also increases accuracy because dropout decreases the influence of a single node.

### *Regularisation*

Regularisation limits the magnitude of the neural weights by adding an extra cost for weights to the error function. There are two types of regularisation: L1 and L2 regularisation. L1 regularization uses the sum of the absolute values of the weights whereas L2 regularization uses the sum of the squared values of the weights.

Dropout and regularisation are used to prevent overfitting to the training dataset. Although I had implemented dropout rates and regularisation in my program, my program was not overfitting and the test accuracy was not much lower than the training accuracy. This meant that it was not necessary to use dropouts or regularisation for my neural network. Several attempts at these two techniques resulted in a much lower training and test accuracy and led to greater loss.

### *Training Algorithm*

The best training algorithms were the various forms of gradient descent as these were the fastest to compute while providing a high degree of accuracy. After several iterations, the accuracy of the neural network was found to be more dependent on the learning rate than the type of gradient descent algorithm used.
After numerous combinations, the best algorithm was found to be the Adam optimiser with a learning rate of 0.001.

### *Error Function*

The numerous error functions have different derivatives and different purposes. Probabilistically, the cross-entropy (CEE) arises as the natural cost function to use if the sigmoid or softmax activation function is used in the output layer of the neural network. This is used to maximise the likelihood of classifying the input data correctly. If the target is continuous and normally distributed, then using mean-squared (MSE) usually maximises the accuracy.
For classification, cross-entropy tends to be more suitable than mean-squared however, for regression problems mean-squared should be used. As the particle analysis problem is a classification problem, the cross-entropy cost function worked the best. This was evaluated through the process of trial-and-improvement.

### *Layers & Nodes*

There are only one input and output layers in any neural network. The input layer has the same number of nodes as the number of input metrics selected and therefore there are overall 8 inputs into the neural network. The output layer has the same number of nodes as the number of output classes. Since one-hot encoding is being used for the training data of the neural network, there will be multiple nodes for the output layer. From the dataset created using LUCID Trainer, there are 6 different output classes for the different particles the neural network will be classifying.

#### Activation Functions

The activation functions that were tested were the hyperbolic tangent function, the sigmoid function and the rectified linear function for the hidden layers. After several iterations, the RELUs failed to provide accuracy higher than 80%. This was most likely because the inputs were not normalised and as some of the metric may have been less useful than others when the values were large. The hyperbolic tangent function has values between -1 and 1

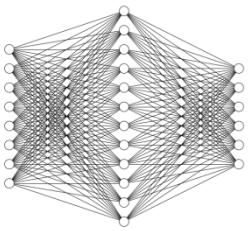
and therefore it also seemed to lack in accuracy. Therefore, the sigmoid function was used as it was able to compensate for the large values due to the "vanishing gradient" and it was in the range 0 to 1.
As the analysis problem is a multi-class classification problem, the softmax function was used for the output layer.

### Hidden Layers

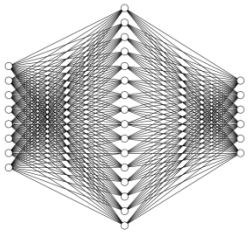
To construct the neural network, it was essential to understand how many hidden layers to have in the neural network. A large number of layers is used for greater learning capacity however this can also result in overfitting as the neural network can end up memorising the training data in the form of the neural weights. Experimenting with the number of hidden layers, showed that a single hidden layer was too shallow for any learning at all and any more than 2 layers resulted in high overfitting.

### Number of Nodes

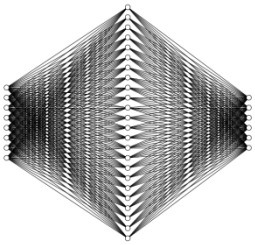
This process was purely trial-and-improvement however, there were some preliminary factors that allowed the decision to be made clearer. As there were more inputs than outputs, it was more appropriate to have a decreasing number of nodes in the hidden layers. This was checked during development as primary tests.
Here are the main neural structures used to form the basis of the final selection. There were many other structures that were tested however due to the iterative nature of the modification process, only the key structures that affected the final decision are shown.



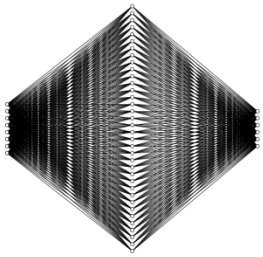
[ 8, 12, 6 ]



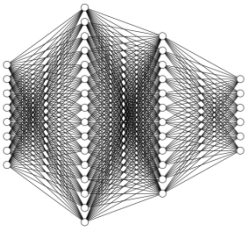
[ 8, 16, 6 ]



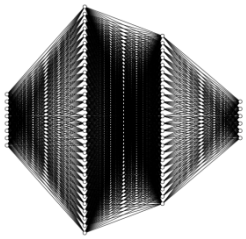
[ 8, 24, 6 ]



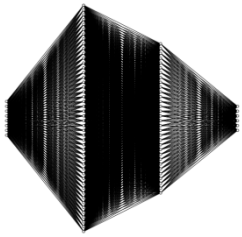
[ 8, 32, 6 ]



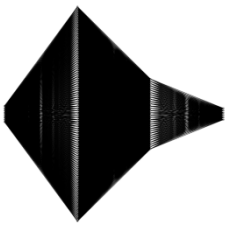
[ 8, 16, 12, 6 ]



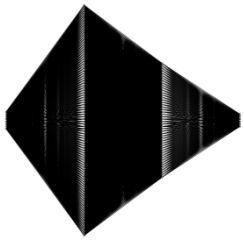
[ 8, 32, 24, 6 ]



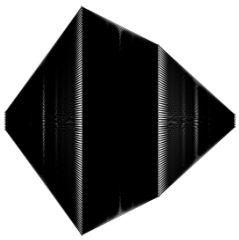
[ 8, 48, 32, 6 ]



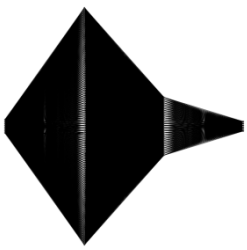
[ 8, 96, 32, 6 ]



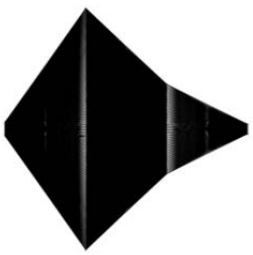
[ 8, 96, 48, 6 ]



[ 8, 96, 64, 6 ]



[ 8, 128, 32, 6 ]



[ 8, 128, 48, 6 ]

From the development process, it was noticed that adding any more than 128 nodes in any of the layers resulted in overfitting. Moreover, only 2 hidden layers provided good enough accuracy regardless of the number of nodes. For a greater depth, a low number of nodes simply resulted in the inability to learn while a high number of nodes resulted in overfitting.

## Final Neural Network

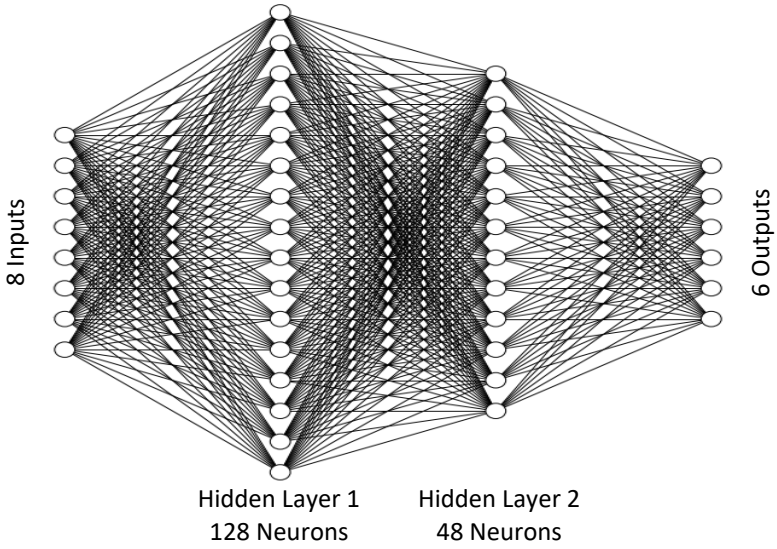
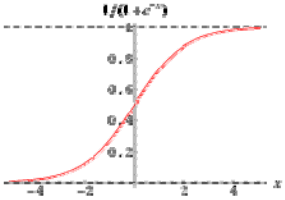



Figure 25 - Final Neural Network

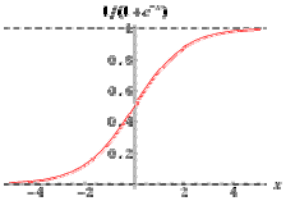
Figure 25 shows the structure of the final neural architecture. Figure 26 was the auto-generated meta-graph of the layers from the Python Tensorflow code. Layers 1 and 2 multiply the weights matrix by the output from the previous layer, the biases are added and then propagated through the sigmoid function. The output layer is the same however, the sigmoid function is replaced with the softmax function.

The following activation functions were used in the final neural network:

**Hidden Layer 1 – Sigmoid**



**Hidden Layer 2 – Sigmoid**



**Output Layer – Softmax**

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; for \; j = 1, \ldots, k$$

Generalised version of the sigmoid function.

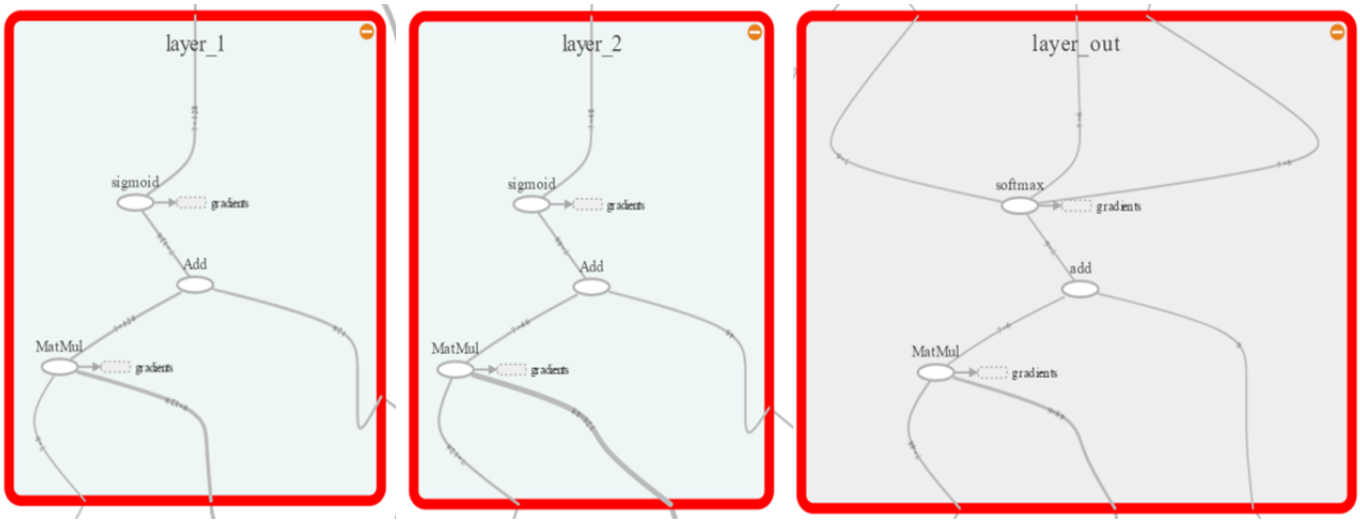
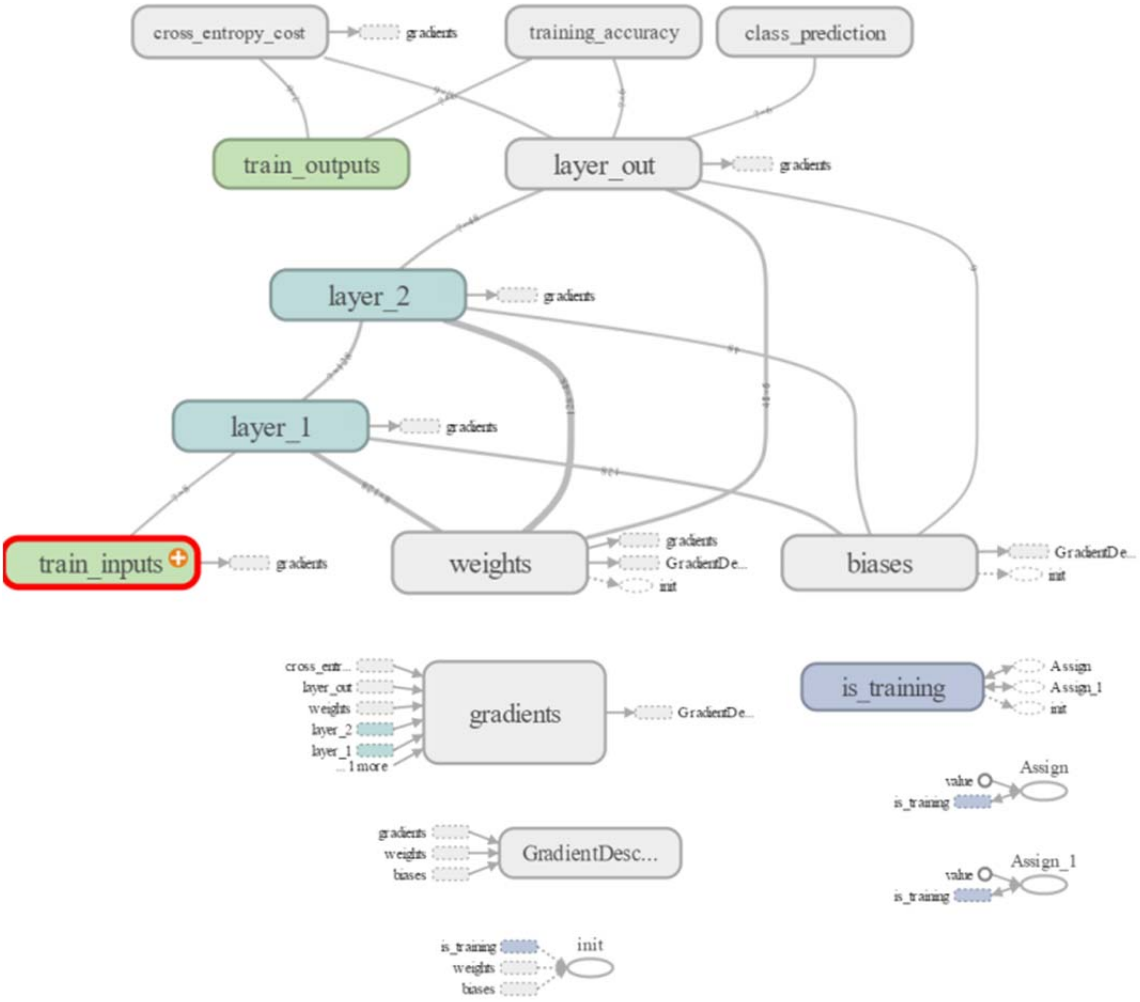
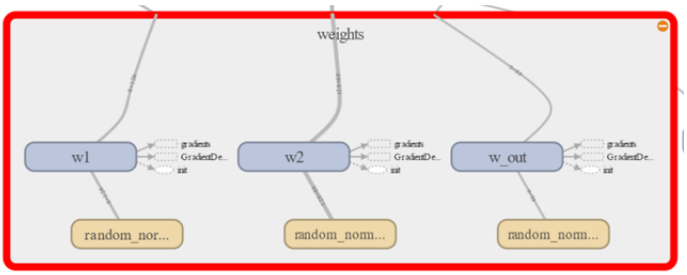


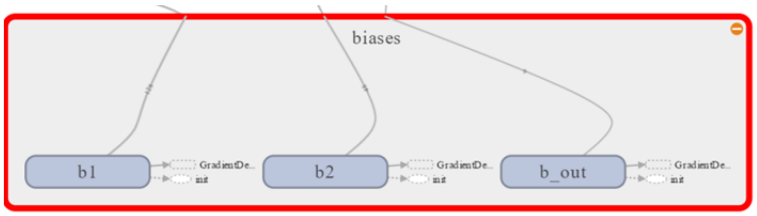

Figure 26 - Meta Graph of Layers

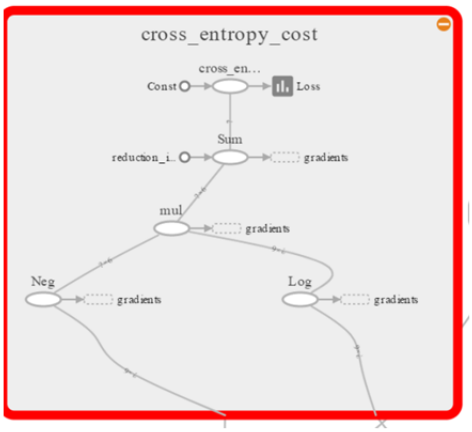
### *Tensorflow Meta Graphs*

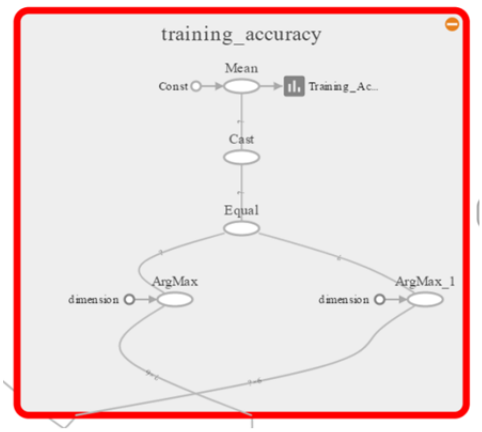
The following meta-graph shows the structure of the whole neural network and how the neural network works in the simplest manner.

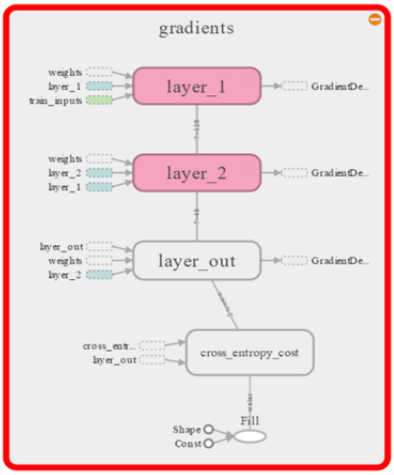


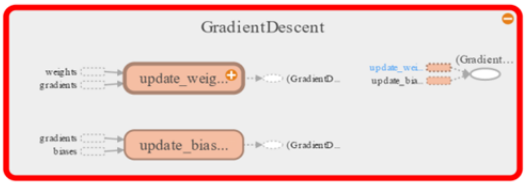



### Test Data Production

The test data had to be different from the training data just so that the neural network did not overfit to the training data. Overfitting means that the accuracy of the neural network is only valid for the training data inputted into the neural network and therefore the output would not be of the same accuracy for another dataset. This is mainly because the model may have been made too complex and therefore accounts for idiosyncrasies in the training data set. To prevent overfitting, the test data had to be chosen randomly and had to represent a real-life dataset.

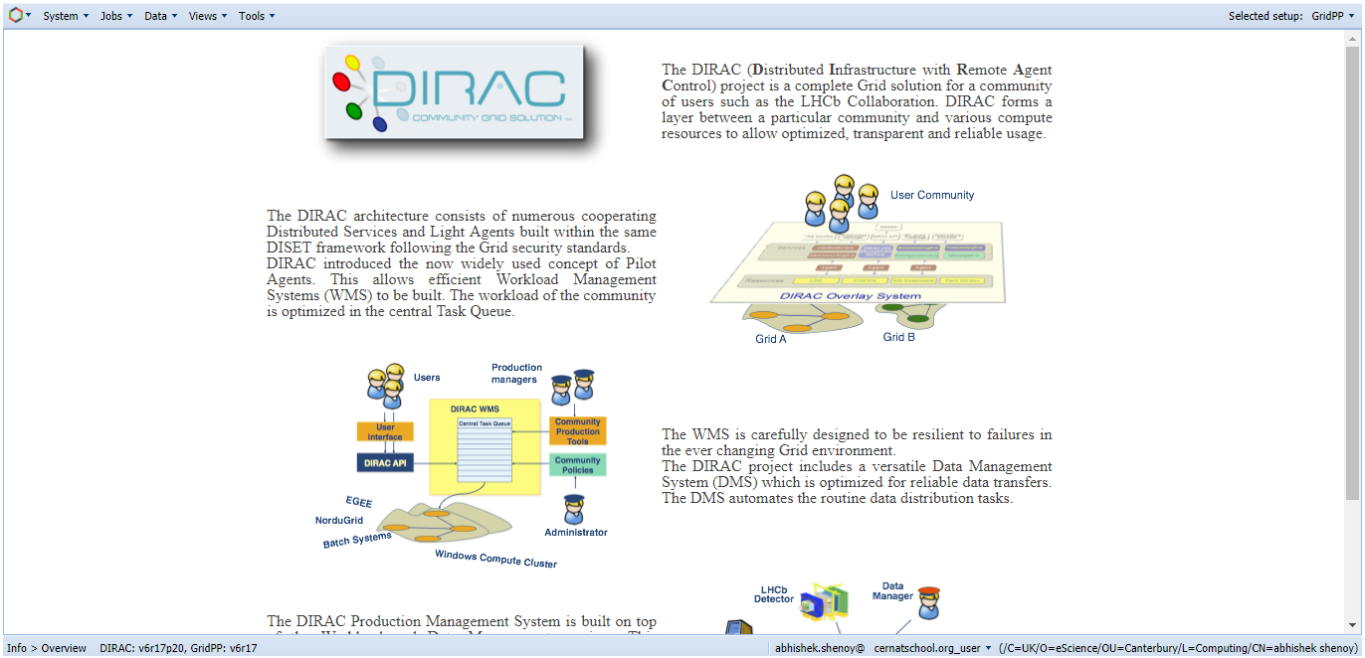
The data used to test the neural network was simply generated by dividing the database from the LUCID Neural Trainer randomly into two sets of data: a training dataset and a test dataset; where the test dataset is smaller than the training dataset. Then using the metrics program, the metrics of the blobs were calculated. To calculate the accuracy after training the neural network with the training dataset, the test data was inputted into the neural network and the output values were computed and stored. Using the output value, the representative class was determined for each measured particle track and these were then compared with the original classes stored in the database.

## Test Results

### GridPP

"The GridPP Collaboration is a community of particle physicists and computer scientists based in the United Kingdom and at CERN, supporting tens of thousands of CPU cores and petabytes of data storage across the UK and it was used to discover the Higgs boson at CERN's Large Hadron Collider. Such a large-scale distributed computing grid is used to solve data-intensive problems." [26]

Doing research as a part of the LUCID project, I was able to get an e-science certificate to be able to access the grid and as of 2016 and 2017, I am the youngest member of the Particle Physics Computing Grid. Being a member of the GridPP has enabled me to train and test several of these neural networks concurrently as individual jobs on the computing grid.




### Accuracy Tests

To check the accuracy of the current algorithm, I ran the current analysis algorithm on all of the blobs in the database created by the LUCID Neural Trainer. The total number of correct classifications was divided by the total

number of particles to calculate the accuracy of the algorithm. This was very important, as it would allow me to evaluate the success of my neural network and the improvements necessary to further develop the classifier.

During the training process of the deep neural network, the training accuracy and the loss of the error function was recorded. After the whole training process, this data was converted to a CSV file to allow easy data visualisation. The graphs below were created using the Tableau software from the CSV files generated by Tensorboard.

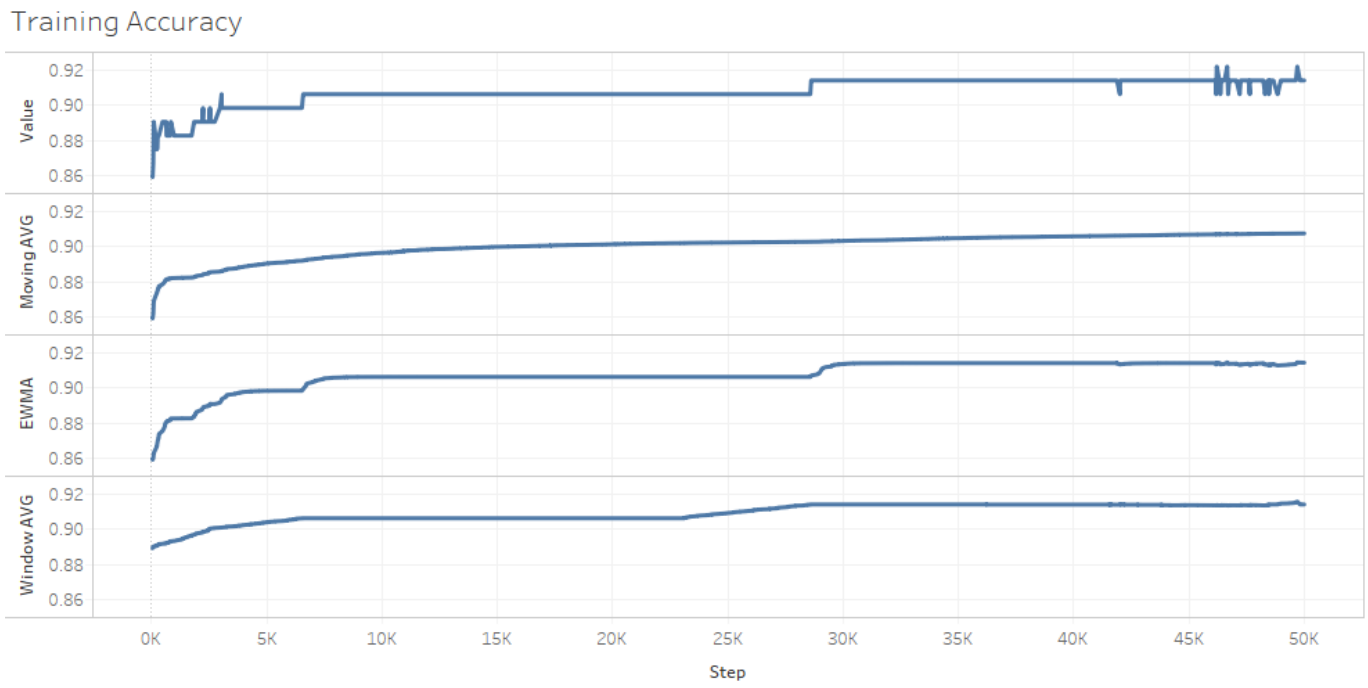



Figure 27 - Training Accuracy Graph

Figure 27 shows the increase in accuracy of the neural model as the model was trained over 50,000 epochs. The moving average, exponentially weighted moving average and window average show the smoothened version of the accuracy graph representing the approximate average training accuracy rather than the absolute value for a specific training instance.

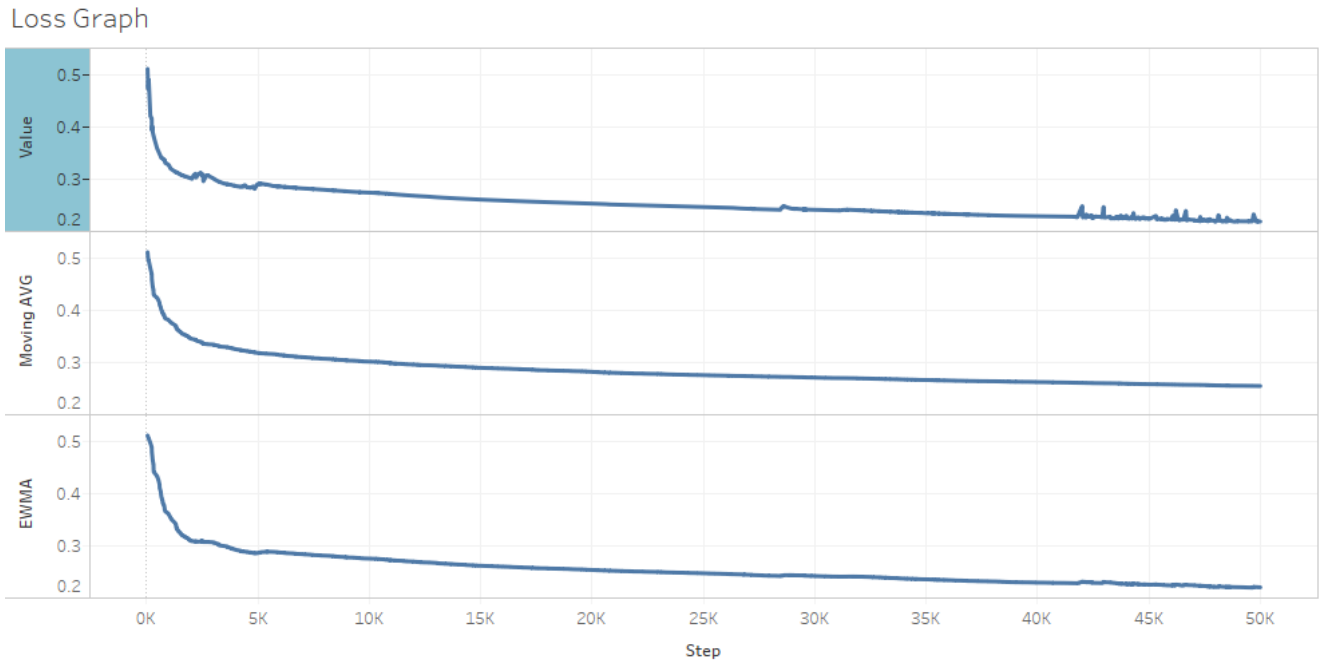



Figure 28 - Training Loss Graph

Figure 28 shows the error from the error function as the model was trained over 50,000 epochs. The moving average and the exponentially weighted moving average show the smoothened version of the loss graph representing the approximate average training loss rather than the absolute value for a specific training instance.

To compare the accuracy of the neural network with the accuracy of the benchmark classifiers, I simply ran the entire database through each of the classifiers and printed out the accuracy tests. Figure 29 shows a summary of the results in the form of a bar graph.

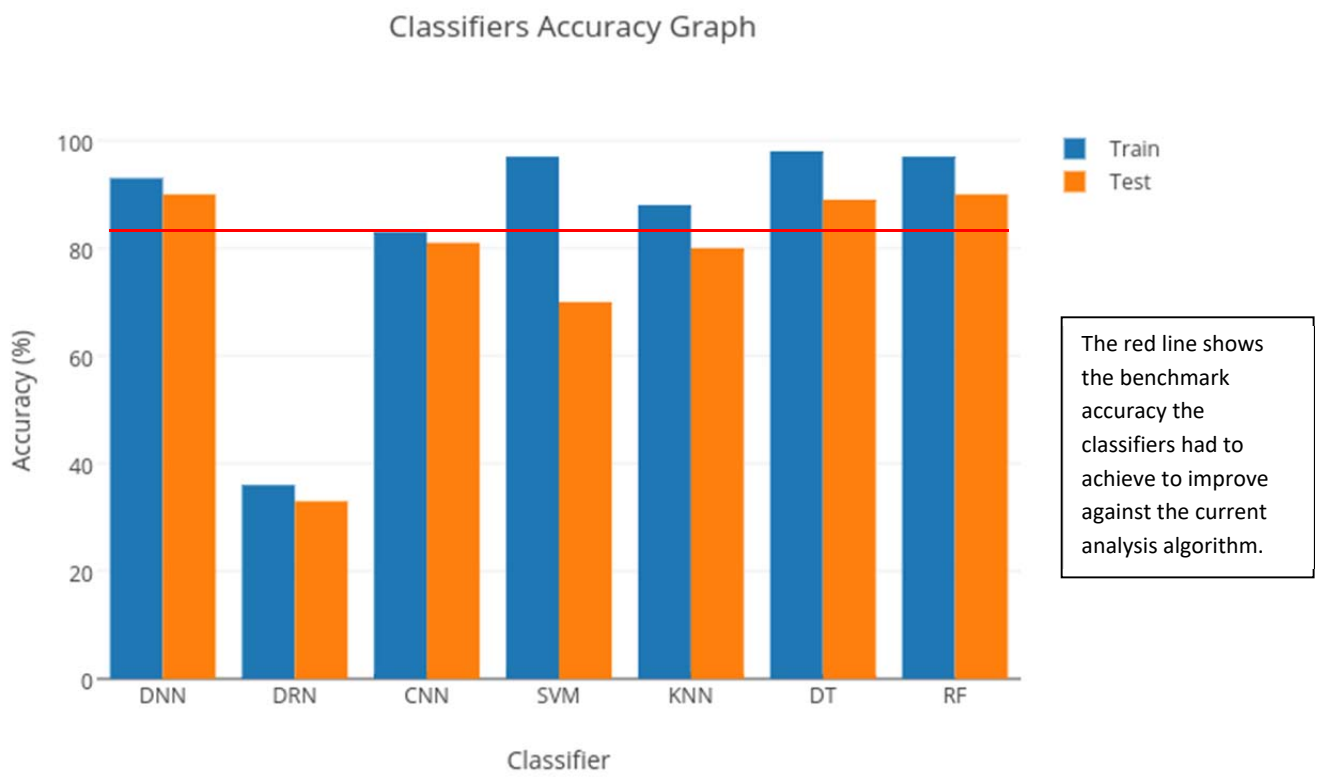



**Figure 29 - Classifiers Accuracy Graph**

### *Justification*

The hyperparameters of the deep feed-forward neural network (DNN) were fully optimised through the process of trial and improvement. The only way of increasing the accuracy is to provide more labelled data to continue training the neural model.

Deep residual networks (DRNs) require a lot of computational power. Even training a relatively shallow residual network of 20 layers for 10 epochs, for 5 days on a multi-core CPU server with a 2.53GHz clock speed and 12GB RAM would not outperform a GPU training for 12 hours. A high-end GPU such as the Nvidia Tesla or the Nvidia Quadro GP100 can speed up training by almost 15 times and provide a high degree of accuracy. Not only that, they can also train much deeper residual networks with a low training time due to the high amounts of memory and the ability to parallel process. However, these GPUs are very expensive to buy and relatively expensive for electricity usage. The inaccessibility to a GPU meant that I could not explore this neural architecture far enough to exploit its full potential.

The problem of the lack of computational power also applied to training the convolutional neural networks (CNNs) however, the problem was much less difficult as CNNs are not deep in comparison to DRNs and therefore required less computational power to train. The training time was still quite long, approximately 4 days continuously running on the server for a mere 300 epochs. However, the high accuracy showed that the CNN has much greater potential as it uses the raw images for processing rather than the metrics calculated. The newly

trained CNN model almost performs as well as the current analysis algorithm making this classifier highly suitable for further development.

SVM and KNN hyperparameters were not optimised and therefore they did not perform as well as the other neural networks yet they still managed to provide a moderate degree of accuracy. The results from the SVM show that the SVM had clearly overfit to the training data and therefore its overall predictive power was lost. By optimising the hyperparameters, this classifier could be thoroughly improved. The KNN, on the other hand did not overfit extensively and still provided a high degree of accuracy. Therefore, optimising the K parameter and increasing the training dataset would largely improve the accuracy of the classifier.

Decision trees (DT) and random forests (RF) had a surprisingly high accuracy. These tend to overfit to the training data but the high test accuracy suggests that these two classifiers can be plausibly used for analysing particles. It was more intriguing to notice that although the random forest classifier (the ensemble of decision trees) had a higher accuracy than a single decision tree, it was not greatly more accurate than the single decision tree. This also led to the conclusion that the generated training data was not sufficient or fully accounting and that more data would definitely increase the accuracy of all the classifiers.

## Mini Investigation – South Atlantic Anomaly

To test the program for a real life phenomenon, I devised an experiment using LUCID to analyse the type of radiation in space from the data files that have already been collected as well as new ones that have been configured with specific configuration files for specific regions over the Earth.

### *Hypothesis*

It is possible to prove the existence of the South Atlantic Anomaly using the neural network on LUCID data. On greater analysis of the data collected, it is possible to understand more details about the correlation between the location and the type and intensity of radiation over the Earth.

### *Experiment*

There were two methods for observing the South Atlantic Anomaly:

1) Collect new data files with new configuration files to map the particle counts
2) Use existing data files to map the particle counts

**New Data Files**

For the first method, to test for the South Atlantic Anomaly, the shutter exposure time had to be quite small to prevent the chip from receiving too many particle hits and whiting out the chip. The other settings of the chip were kept constant as they were not necessary to change.

**Existing Data Files**

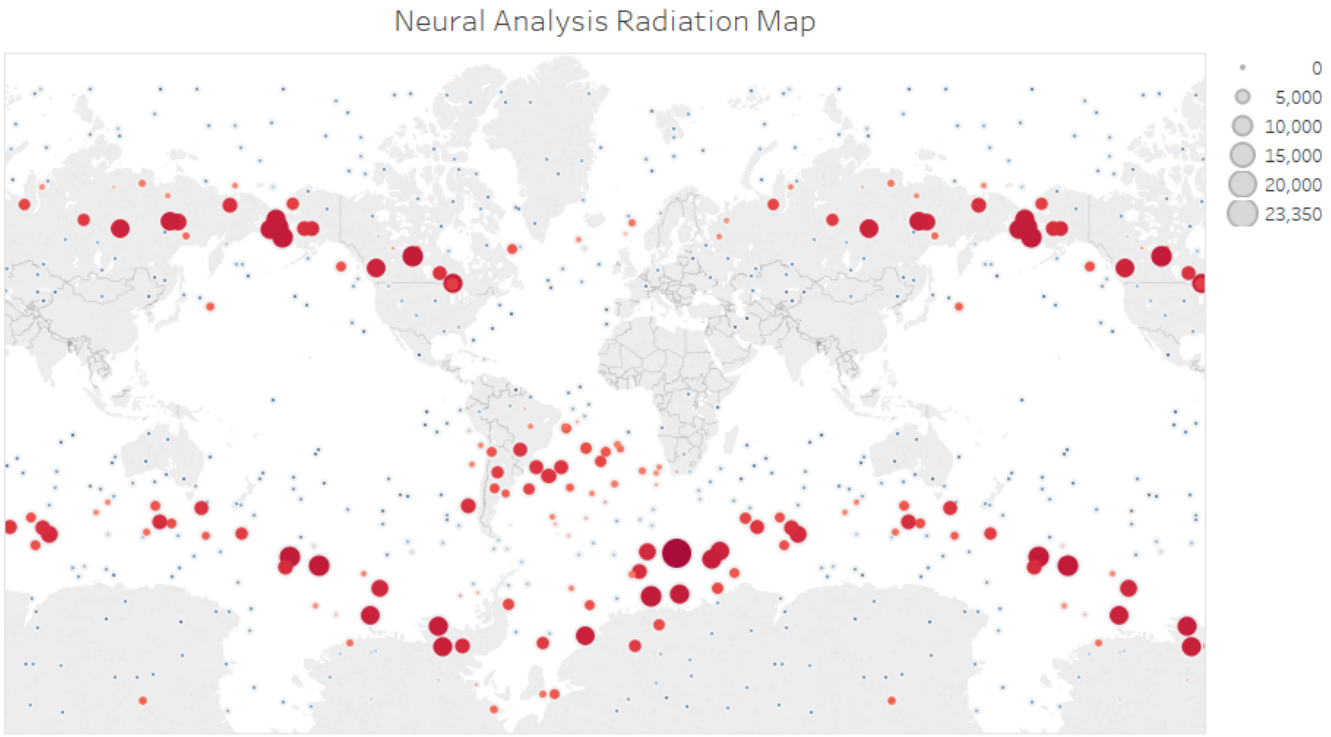
The second method was to analyse the existing data files using the data API in Python to get the selected data files. The latitude, longitude, number of frames and the counts of each particle in each data file will be written to a text file. By collecting the particle counts for all of the selected data files, a radiation map can be plotted which can then be used to observe the high radiation zones and the average levels of radiation over the Earth.

For plotting these maps, I decided to use a mixture of programming and Tableau. Tableau provided me with flexibility in data handling while the programmatic map generator allowed me to make continual revisions to the display of the standard radiation map.

### *Results*

Due to the vast amount of data collected from LUCID, it was only possible to analyse approximately 10 frames from 10% of all the data files to prove this hypothesis. Analysing these frames using the neural network was much slower than using the old analysis algorithm, as the neural model was computationally much more expensive.

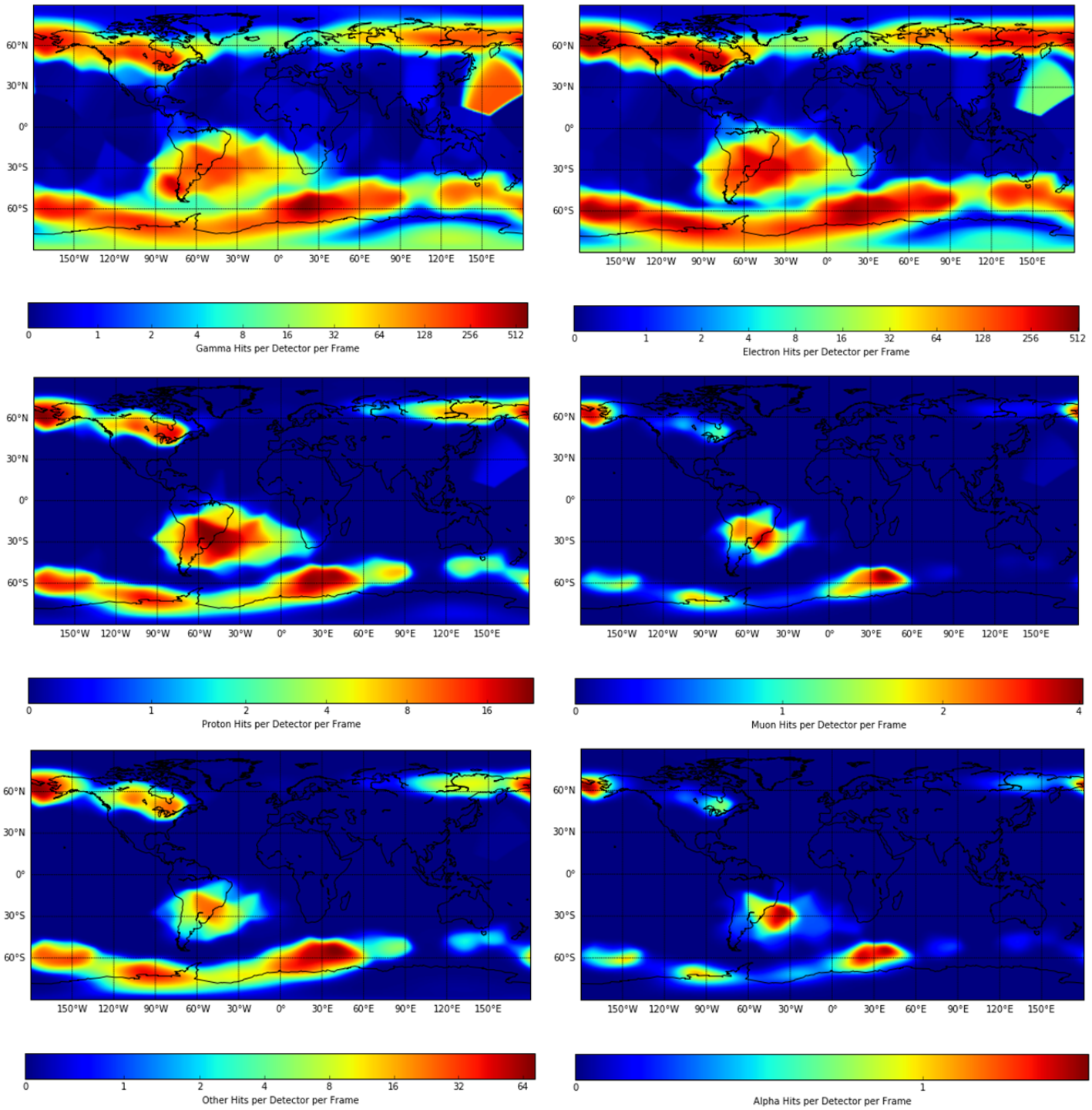
**Initial Counts Map**




This map shows the total particle counts from the selected data files. This initial map was created mainly to see the distribution of the data files. By looking at the map, it is possible to see that there is an even distribution of data files, yet the most prominent ones are the ones near the poles and in the South Atlantic. This map suggests that the neural network is successful in identifying general particles and that it performs as well as the old analysis algorithm for general count purposes.

However, this does not yet suggest that the neural network is successful at identifying the different particles that are expected from these high radiation zones. To test the model, the counts of each individual particle classes were added and stored in a CSV file. These were then plotted as interpolated logarithmic colour graphs to show the distribution of the various particles.

**Interpolated Particle Counts Map**




From the newly analysed frames, the results clearly suggest that the neural network is very successful at correctly identifying various particles in the South Atlantic Anomaly. The proportion of each particle type in the various radiation zones is highly accurate and is as predicted by several research papers and institutes. [27] [28]

### *Conclusion*

The results generated from the map generator strongly support the hypothesis of this investigation. Apart from providing evidence for the South Atlantic Anomaly, it also provides confidence in the LUCID data and the neural network designed for analysis. This whole endeavour opens scope for new research projects using the collected data and the analysis algorithm.

## Conclusion

The deep neural network accurately classifies all of the common particles very easily and even uncommon ones are recognised to a high degree of accuracy. The advantage of this technique is that the model can be retrained on a new dataset from its current checkpoint allowing gradual improvement in accuracy over time. As the outputs are one-hot encoded, the decimal outputs are used as confidence values for each class that the particle could belong to.

The convolutional network that had been created and tested has the most potential out of all the machine learning techniques as it allows for the addition of more features and has the ability to increase drastically in accuracy for this classification problem.

To develop on the newly created deep neural network and to improve the classification accuracy, I combined the benchmark classifiers with high test accuracies (above 90%) alongside the neural network and the current algorithm so that the new system can be used to provide multiple predictions. The most common prediction from this set of prediction will be the most accurate prediction for the particle cluster.

The neural model and the benchmark classifiers with their respective APIs have been made available on GitHub for use by any institute:
https://amshenoy.github.io/lucid_neural_analysis
https://amshenoy.github.io/lucid_classifiers

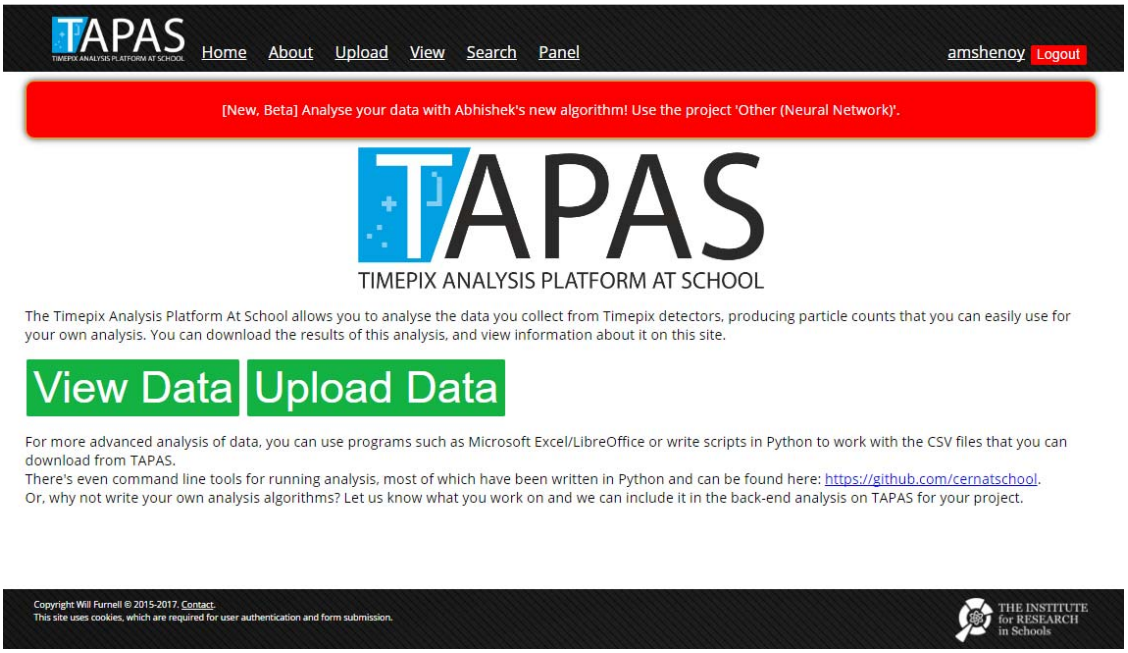



Figure 30 - TAPAS

Having successfully built the neural network and ensured its accuracy, it was integrated into TAPAS (Timepix Analysis Platform At School) to analyse particle tracks collected by students all around the UK for particle physics research projects (Figure 30).

The integration of the neural network into TAPAS clearly shows the need for a good analysis algorithm to analyse particle tracks from radiation detectors in various different fields.
Due to the vast amounts of LUCID data, the particle data collected has many applications. The LUCID data could potentially be used to find out the source of radiation in space as well as the dose of radiation that astronauts have to experience in space. Apart from this, it may even be possible to create a computer-simulated model that could predict future radiation in space.

## Evaluation

To improve the overall accuracy, the amount of labelled data used to train the neural network needs to be increased. Since a human generates the labelled data, the process of training data production requires a lot of time and is not very efficient. This was the main problem in trying to achieve high accuracies with all the machine learning classifiers. Below I have outlined analysis methods that can be used to improve the analysis accuracy.

### Energy Analysis

The deep neural network does not account for the distribution in energy values. This is because analysis using energy values is highly difficult as the pre-processing is dependent on shape and energy deposit and it is harder to describe parametrically. Although this problem could have been solved by using a convolutional neural network for image classification like the one I had tested, it would have been very slow to train and run for analysis on a web platform. Moreover, the LUCID Trainer was mainly developed for the deep neural network using metric analysis and therefore the energy values had been removed during the process of clustering. Hence, the convolutional neural network I developed did not analyse energy values.

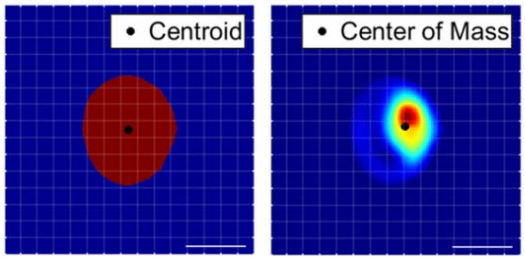



Figure 31 - Centre of Energy Deposit

For a metric-based system, the "centre of mass" (Figure 31) can be calculated along with the centroid and this can be used as another one of the inputs in the neural network. The current centroid calculation method only uses equally weighted pixels without energy values however by adding energy values as the weighting, the collision point of the particle can be determined and this may help a neural network to perform better classification.

### Sub-System Analysis

The neural network can be further extended into a system of neural networks or a system of general machine learning classifiers whereby which the particles can even be given sub-classes for a greater discrepancy between alike particles.

Protons → High Energy, Low Energy

Electrons →High Energy, Low Energy

MIPs →Muons, Electrons (Misclassified as MIPs)

Others → Pions, Alpha, Heavy Ions, Particle Crossovers

The above sub-classes are only examples however a full sub-system analysis could almost implement a broad particle analysis spectrum.

### Convolutional Analysis

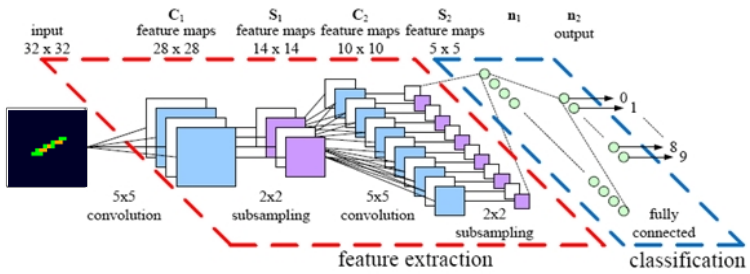



Figure 32 - Convolutional Analysis

Convolutional networks (Figure 32) are very useful as they can not only analyse energy values but they can also provide a deeper insight into what make particles different to each other. This can be visualised by viewing the filters learned by the network after training.

### Summary

For a reasonably simple deep neural network, I was very successful at solving this classification problem. The main difficulty in the process of the neural constructions was that the computing power was rather limited for iteratively editing the design of the network to ensure the best accuracy. As an improvement, I would definitely like to use a GPU for training simply to provide more training iterations and therefore allowing the possibility for increased accuracy.

# LUCID Dashboard

## Data Visualisation

To understand more about LUCID data, the simplest way was to use data visualisation software alongside data mining techniques to display practical data in a format that conveys a correlation or relationship between multiple factors. To do this, I intended to create an online web application to allow the presentation of LUCID data in a user-friendly and useful manner. For presenting the data, I had to research different data analysis APIs and software to successfully evaluate and choose the most suitable ones for my web application.

### Tableau Software

Tableau is a business intelligence and data-analytics software company based in Seattle. The proprietary Tableau software provides an easy way of visualising data from a range of statistical formats and a range of database management systems (DBMS) namely the popular ones being MySQL, CSV, SQLite and JSON. The software is mainly used for offline data visualisation and most of the analysis features are only provided offline however the graphs that are produced can be exported as an image or can be viewed in a web browser using Tableau Server for premium users.

### Plotly Library

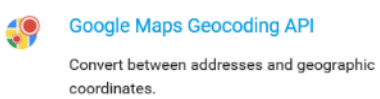
The JavaScript Plotly library is entirely open-source and the advantage of using JavaScript over the Python library is that it can be readily viewed on a website. The facility of Plotly to be integrated into various programming languages including Python and JavaScript, gives it much more functionality than Tableau and provides data visualisation with various forms of interactivity. Many more data formats can be easily integrated simply by standardising the data in a programming language. Plotly graphs also provide high amounts of user-interaction making this API highly suitable for an online web application.

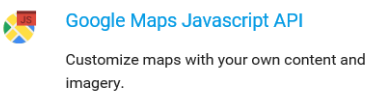
### ArcGIS Platform

The ESRI (Environmental Systems Research Institute) software company provides ArcGIS for location-based data analysis. Although it is very useful as a location-based analysis software, it lacks many of the features for data analysis. Moreover, the software itself is more designed for geological analysis rather than data visualisation.

### Google Maps API

The Google Maps API provides a range of features however as the LUCID data is based on locational comparisons, I will only need to use two of the Google Maps APIs including the Maps JavaScript API and the Maps Geocoding API.




The Google Maps API is very flexible, is fully apt at presenting geo-locational data, and provides features like place searches, proximity to an address, marker attachments and embedding on a webpage. This makes the API very useful for displaying the LUCID data on a webpage.

## Dashboard

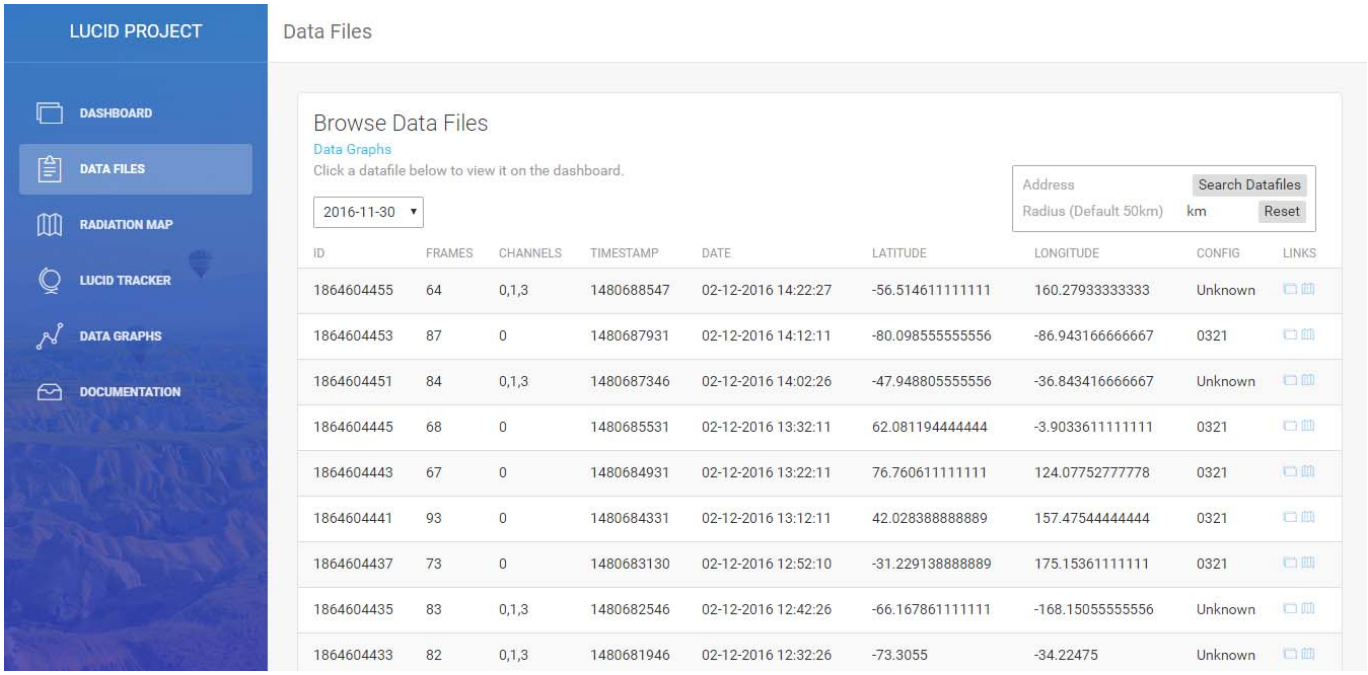
The LUCID Dashboard is an online web application designed to display LUCID data in a viewable, easy-to-analyse format, providing various methods of looking through all of the collected data files. I made the application using PHP as the backend scripting language and, HTML (Hypertext Mark-up Language), JS (JavaScript) and CSS (Cascading Style Sheet) for serving it as a user-friendly website in a web browser. The fundamental structure of the web application was based on the user interface of an existing admin dashboard [29] complying with the MIT license.

The front-end of the dashboard needed to fulfil the following criteria:

- Display the latest LUCID data frames from the most recent data file
- Display the metadata of the data file
- Provide features that allow data analysis – (XYC for Energy Values, Energy Filter, Particle Type and Counts)

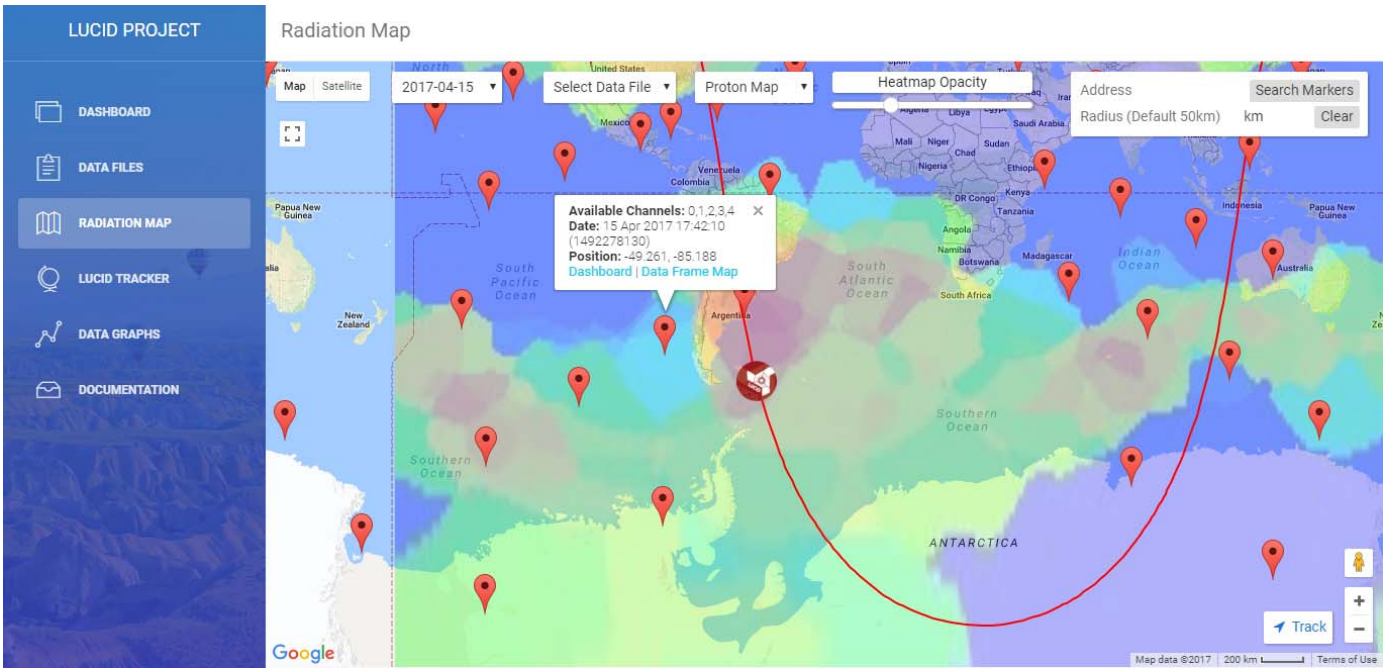

The details of the frame can also be viewed by clicking the info button and this shows the geo-location of the capture on a Google Maps popup. The XYC file of each frame is analysed using the saved neural network model and the counts of each particle are then displayed under the PNG frame. For fast page-loading, this analysis feature was disabled however the implementation clearly shows the broad uses of the neural network.

## Data Files



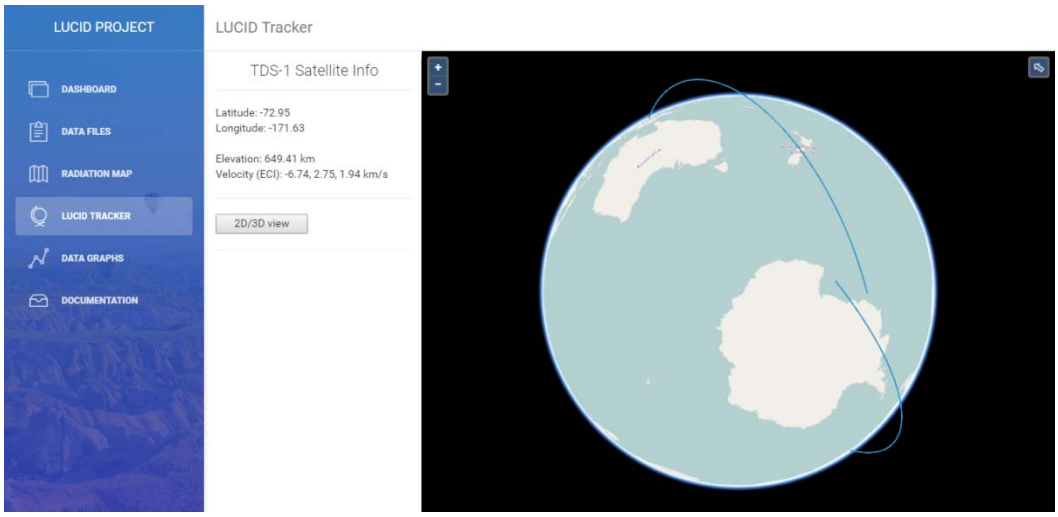

This page allows users to search for data files by their data run and also by the form of radial distance from a specific location. By clicking on the links in the link column, the user can be redirected to the dashboard or the radiation map to view their selected data file in further detail. All of the metadata about the data file that is stored in the server database is presented in a neat table for users to scroll through.

## Radiation Map



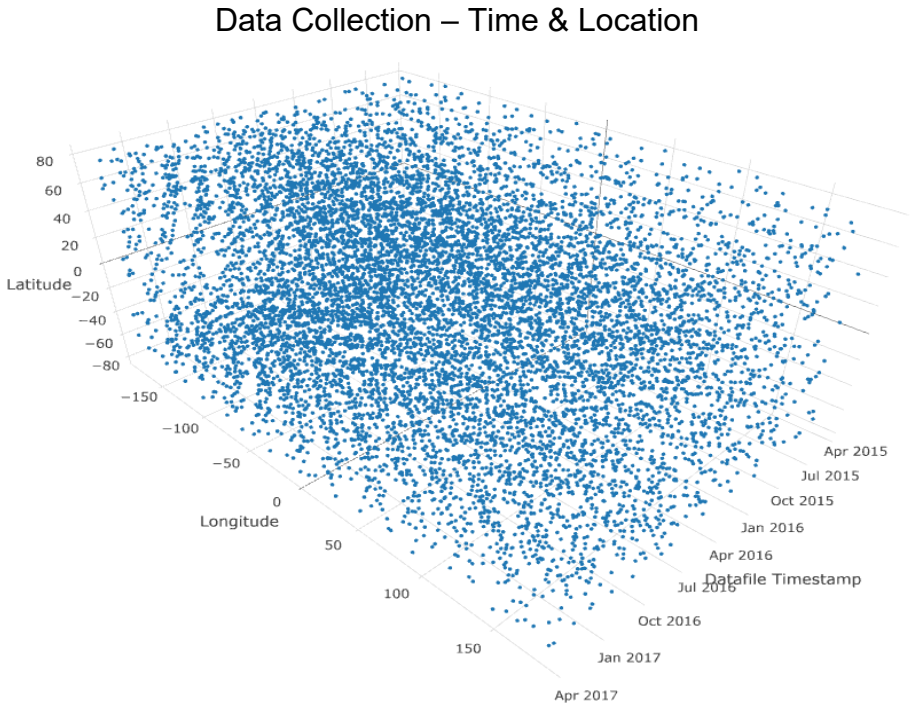

This page shows a pre-processed radiation map overlaid on the map of the Earth using the Google Maps API. Each marker is a data file in the selected run. Upon clicking the marker, the user is shown a small information window that shows the active detectors, the timestamp of when the file was captured and the geolocation. It also provides a link to the same data file on the dashboard and a link to all the frames of the data file on the same radiation map. LUCID is tracked by default however it can also be followed on the map by simply clicking the track button.

## LUCID Tracker



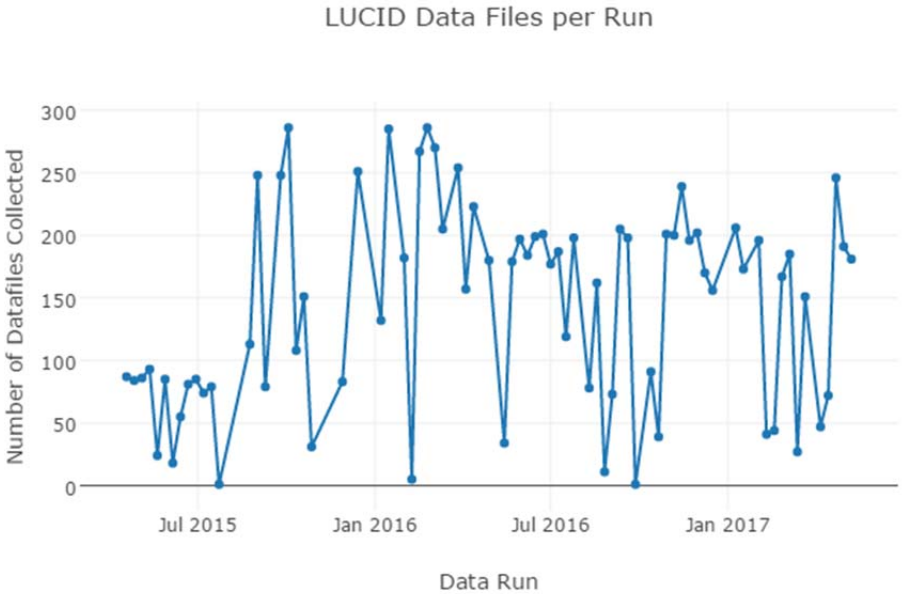

The LUCID tracker will mainly be used for demonstration purposes to show the location and trajectory of LUCID in 3D space. The user interface has been made such that the key details are presented on a side bar and the user if left to explore on the 3D globe.

## Data Graphs




It was important to be able to keep track of an overview of the data that was collected. To do this, the aim was to generate graphs for different aspects of the LUCID data so that users of the LUCID Dashboard can view these features easily.

The metadata of all the collected data files is processed on runtime using PHP and then converted to JSON so that it can be served into the Plotly JavaScript API.




Another example of a data graph is the plot of the data files collected per run as a time series graph.

All of these graphs are intended to provide real-time information about LUCID data collection.

## Documentation

### XYC File URL

https://starserver.thelangton.org.uk/lucid_dashboard/id_frame_channel.txt
https://starserver.thelangton.org.uk/lucid_dashboard/747002715_0_0.txt

### XYC Analysis

https://starserver.thelangton.org.uk/lucid_dashboard/pyxan/id_frame_channel.txt
https://starserver.thelangton.org.uk/lucid_dashboard/pyxan/747002715_0_0.txt

{"Alpha": 0, "Beta": 8, "Gamma": 5, "Muon": 0, "Other": 0, "Proton": 0}

https://starserver.thelangton.org.uk/lucid_dashboard/pyxan/xyc_input.html

Type your xyc here...

Submit

### Dashboard Usage & API

Brief Outline of all the Features ( [Optional] {Parameter} (Range) **Optimal )

| TYPE | DATA | DESCRIPTION | SIMPLE USAGE | ADVANCED USAGE | EXAMPLE |
|---|---|---|---|---|---|
| API | PNG | Display the PNG of a given frame | *{id}* _ *{frame}* _ *{channel}* [_x *{scale(1-20)}*] [_s].png ** frame / *{id}* / *{frame}* / *{channel}* / [ *{scale(1-20)}* ] / [s] / | gen_image.php?id=*{id}*&frame=*{frame}*&channel=*{channel}* [&scale=*{scale(1-20)}*] [&min_e=*{min_e(0-11810)}*] [&max_e=*{max_e(0-11810)}*] [&simple] | 1175485581_0_0_x4_s.png frame/1175485581/0/0/4/s gen_image.php?id=1175485581&frame=0&channel=0&scale=4&min_e=10&max_e=100&simple |
| API | XYC | Display the XYC of a given frame | *{id}* _ *{frame}* _ *{channel}*.txt ** xyc / *{id}* / *{frame}* / *{channel}* / | gen_xyc.php?id=*{id}*&frame=*{frame}*&channel=*{channel}* [&min_e=*{min_e(0-11810)}*] [&max_e=*{max_e(0-11810)}*] | 1175485581_0_0.txt xyc/1175485581/0/0 gen_xyc.php?id=1175485581&frame=0&channel=0&min_e=10&max_e=100 |
| IFrame | Map | Embed Radiation Map on another website - If the optional parameters are not provided then no markers are shown unless selected by dropdown. | /map/iframe/ ** | /map/iframe / [ *{selected_run}* ] / [ *{selected_file}* ] | <iframe src='/map/iframe' width="100%" height="100%"></iframe> |
| IFrame | Dashboard | Embed Dashboard on another website - If the optional parameters are not provided then the latest data is displayed in the iframe. | /view/iframe/ ** | /view/iframe / [ *{id}* ] / [ *{frame}* ] / [ *{channel}* ] | <iframe src='/view/iframe' width="100%" height="100%"></iframe> |
| IFrame | Data Graphs | Embed Data Graphs on another website - If the optional parameters are not provided then the first graph is displayed. | /data/graph/iframe/ ** | /data/graph/iframe / [ *{g_num}* ] / | <iframe src='/data/graph/iframe' width="100%" height="100%"></iframe> |
| Radiation Map | Address & Radius | The map is zoomed to a translucent circle which shows the data files within the radius of the specified location. Note* The latitude and longitude of a datafile is the latitude and longitude of the first frame. | Any place name recognisable by Google Maps can be entered in the address box. The radius can be any positive numerical value including decimals. | If an invalid address then the circle will not be created. If the radius is not set or is invalid then the default value of 50km is used. | Address: Canterbury, Kent \| Radius: 10 km |
| Data Files | Address & Radius | All the data files within the specified radius of a location from the selected run are returned. | Any place name recognisable by Google Maps can be entered in the address box. The radius can be any positive numerical value including decimals. | If an invalid address then no files will be returned. If the radius is not set or is invalid then the default value of 50km is used. / data / *{selected_run}* / *{address}* / [ *{radius 'metres'}* ] / | Address: Canterbury, Kent \| Radius: 10 km /data/all/Canterbury%2C%20Kent/10000/ |

# References


[1] Surrey Satellite Technology Limited, “LUCID,” SSTL, [Online]. Available: http://www.sstl.co.uk/Blog/February-2013/TechDemoSat-1-s-LUCID--a-novel-cosmic-ray-detector. [Accessed 30 November 2016].

[2] Surrey Satellite Technology Limited, “TechDemoSat-1 Payloads,” [Online]. Available: http://www.sstl.co.uk/Missions/TechDemoSat-1--Launched-2014/TechDemoSat-1/TechDemoSat-1--Payloads. [Accessed 30 November 2016].

[3] X. Llopart, R. Ballabriga, M. Campbell, L. Tlustos and W. Wong, “Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements,” in *Nuclear Instruments and Methods in Physics Research Section A*, Elsevier, 2007.

[4] “MOSFET as a Switch,” ELPROCUS, [Online]. Available: https://www.elprocus.com/mosfet-as-a-switch-circuit-diagram-free-circuits/. [Accessed 5 July 2017].

[5] C. Hewitt, “PTR Generator,” 2017. [Online]. Available: https://github.com/amshenoy/ptrgen.

[6] C. Hewitt, W. Furnell and A. Shenoy, “LUCID Utils,” 2017. [Online]. Available: https://github.com/amshenoy/lucid-utils.

[7] R. Pierce, “Vectors Dot Product,” Math Is Fun, [Online]. Available: https://www.mathsisfun.com/algebra/vectors-dot-product.html. [Accessed 2 March 2017].

[8] P. Roelants, “Cross-Entropy Cost Function,” [Online]. Available: http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/. [Accessed 6 March 2017].

[9] Artelnics, “5 Algorithms to train a Neural Network,” [Online]. Available: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network. [Accessed 11 March 2017].

[10] S. Ruder, “Optimizing Gradient Descent,” 19 January 2016. [Online]. Available: http://sebastianruder.com/optimizing-gradient-descent/. [Accessed 15 March 2017].

[11] Wolfram MathWorld, “Taylor Series,” Wolfram, [Online]. Available: http://mathworld.wolfram.com/TaylorSeries.html. [Accessed 16 March 2017].

[12] A. Gibiansky, “Conjugate Gradient,” 2 February 2014. [Online]. Available: http://andrew.gibiansky.com/blog/machine-learning/conjugate-gradient/. [Accessed 18 March 2017].

[13] A. Gibiansky, “Hessian Free Optimization,” 13 February 2014. [Online]. Available: http://andrew.gibiansky.com/blog/machine-learning/hessian-free-optimization/. [Accessed 22 March 2017].

[14] SDSU Library, “Conjugate Gradient Algorithms,” [Online]. Available: http://edoras.sdsu.edu/doc/matlab/toolbox/nnet/backpr58.html. [Accessed 25 March 2017].

[15] Asimov Institute, “Neural Network Zoo,” Asimov Institute, 14 September 2016. [Online]. Available: http://www.asimovinstitute.org/neural-network-zoo/. [Accessed 29 March 2017].

[16] A. Karpathy, “Convolutional Networks,” Stanford University, [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed 4 April 2017].

[17] M. Hollemans, “Convolutional Networks on the iPhone with VGGNet,” 30 August 2016. [Online]. Available: http://machinethink.net/blog/convolutional-neural-networks-on-the-iphone-with-vggnet/. [Accessed 7 April 2017].

[18] L. A. Santos, “Residual Net,” [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/residual_net.html. [Accessed 10 April 2017].

[19] Analytics Vidhya, “Complete Tutorial on Tree Based Modeling,” 12 April 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/. [Accessed 14 April 2017].

[20] “Random Forest Template,” TIBCO Software Incorporated, [Online]. Available: https://community.tibco.com/modules/random-forest-template-tibco-spotfirer. [Accessed 15 April 2017].

[21] B. DeWilde, “Classification of Hand-written Digits,” 26 August 2012. [Online]. Available: http://bdewilde.github.io/blog/blogger/2012/10/26/classification-of-hand-written-digits-3/. [Accessed 18 April 2017].

[22] M. Antonelli, “Support Vector Machine,” [Online]. Available: http://www.marioantonelli.it/svm. [Accessed 19 April 2017].

[23] “SVMDA - Eigenvector Documentation Wiki,” 16 May 2016. [Online]. Available: http://wiki.eigenvector.com/index.php?title=Svmda. [Accessed 20 April 2017].

[24] “Zooniverse,” [Online]. Available: https://www.zooniverse.org/about. [Accessed 22 April 2017].

[25] A. Karpathy, P. Abbeel, G. Brockman, P. Chen, V. Cheung, R. Duan, I. Goodfellow, D. Kingma, J. Ho, R. Houthooft, T. Salimans, J. Schulman, I. Sutskever and W. Zaremba, “Generative Models,” Open AI, 16 June 2016. [Online]. Available: https://openai.com/blog/generative-models/. [Accessed 22 April 2017].

[26] GridPP Collaboration, “GridPP,” [Online]. Available: https://www.gridpp.ac.uk/. [Accessed 18 May 2017].

[27] A. F. R. Laboratory, “Energetic Proton Maps for South Atlantic Anomaly,” 27 July 2008. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a485155.pdf. [Accessed 13 June 2017].

[28] Vernov, S. N., Gorchakov, E. V., Shavrin, P. I., & Sharvina, K. N.;, “Radiation Belts in the Region of the South-Atlantic Magnetic Anomaly,” [Online]. Available: http://articles.adsabs.harvard.edu//full/1967SSRv....7..490V/0000491.000.html. [Accessed 16 June 2017].

[29] CreativeTim, “Light Bootstrap Dashboard Design,” [Online]. Available: https://github.com/amshenoy/light-bootstrap-dashboard.

# Figure Table

Figure 1 - TechDemo Sat-1 (TDS-1) ..... 1
Figure 2 - LUCID - 5 x Timepix Chips ..... 1
Figure 3 - Timepix Chip ..... 1
Figure 4 - MOSFET ..... 1
Figure 5 - Particle Tracks ..... 2
Figure 6 - Bethe Formula ..... 2
Figure 7 - PNG of a Data Frame ..... 5
Figure 8 - Partial XYC of a Data Frame ..... 5
Figure 9 - Average Neighbours ..... 7
Figure 10 - Centroid ..... 8
Figure 11 - Error Surface ..... 14
Figure 12 - Convergence & Divergence ..... 16
Figure 13 - Convolution ..... 20
Figure 14 - Subsampling ..... 21
Figure 15 - Residual Block ..... 21
Figure 16 - Decision Tree ..... 22
Figure 17 - Random Forest ..... 22
Figure 18 – K-Nearest Neighbour ..... 23
Figure 19 - RBF Kernel ..... 23
Figure 20 - Linear Kernel ..... 23
Figure 21 - Generative Adversarial Network ..... 24
Figure 22 - DB Schema 1 ..... 25
Figure 23 - DB Schema 2 ..... 25
Figure 24 - DB Schema 3 ..... 25
Figure 25 - Final Neural Network ..... 30
Figure 26 - Meta Graph of Layers ..... 30
Figure 27 - Training Accuracy Graph ..... 33
Figure 28 - Training Loss Graph ..... 33
Figure 29 - Classifiers Accuracy Graph ..... 34
Figure 30 - TAPAS ..... 38
Figure 31 - Centre of Energy Deposit ..... 39
Figure 32 - Convolutional Analysis ..... 39