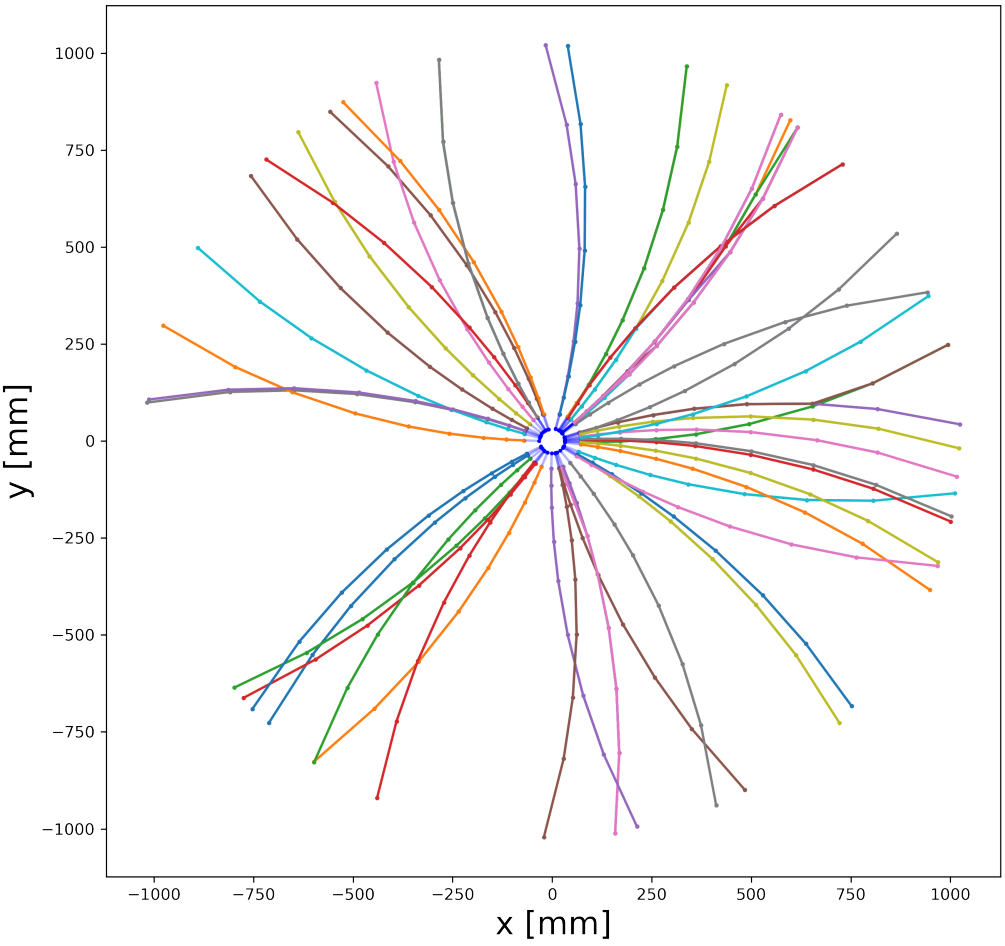
The use of neural networks for track finding at the High Luminosity LHC with the ATLAS trigger system

Nicolai Søderberg (bsh468) and Simon Munch Johannsen (lzt526)

Supervised by: Stefania Xella

15-06-2022



Name of Institute: Niels Bohr Institute
Authors: Nicolai Søderberg and Simon Munch Johannsen
Title: The use of neural networks for track finding at the High Luminosity LHC with the ATLAS trigger system
Supervisor: Stefania Xella
Submitted: 15.06.2022
Defense: 21.06.2022

Abstract

Due to the significant scale of data produced in High Luminosity LHC, traditional particle tracking methods will require more time reconstructing particle tracks. To confront this problem, various different machine learning techniques have been tested. A method that has produced promising results is a Graph Neural Network, which can predict possible track candidates through the coordinates of different hits in the detector. In this investigation, we apply the use of a neural network to classify potential track segments and from this aim to reconstruct the trajectories of individual particles. This network is trained and tested using the public data-sets provided from the TrackML Particle Tracking Challenge. The aim of this investigation is to analyse how fast this kind of neural network can predict possible track segments as well as how accurately it can predict them. Finally we aim to reconstruct the actual trajectories of individual particles using the predicted track segments from the neural network.

Contents

1 Introduction **5**

2 Graph Neural Networks (GNN) **6**
2.1 GNN structure 6
2.2 Edge Classification 7

3 Multilayer Perceptron (MLP) **8**
3.1 Hidden Layers 8
3.2 Determining the hidden layer dimensions 9

4 Data **10**
4.1 Data filtering 11
4.2 Data after filtering 12

5 Finding Possible Edges **15**
5.1 Restricting the curvature of the particle 15
5.2 Restricting the Z intercept of the edge 16

6 Results **18**
6.1 Testing the MLP 18
6.1.1 Using a pT range of $[1, 1.25]$ GeV 18
6.1.2 Using a pT range of $pT \geq 1.0$ GeV 20
6.2 Tracking Particles 22

7 Conclusion **23**

8 References **24**

A Full sized images **25**

1 Introduction

Experiments in High Energy Physics (HEP) have led to significant scientific discoveries in the past, one example is the discovery of the Higgs Boson. This discovery was made through particle accelerators, reflecting the importance of running these experiments. However as the search for more information continues, the amount of data produced increases. The ATLAS detector, for example, more than 1000 events per second, which is about 2 gigabyte per second. This eventually leads to several petabytes of raw data per year [1], therefore, in the search for new scientific discoveries, it can be like searching for a needle in a haystack. To analyse this data one needs to identify the tracks of the particles, reconstructing their path inside the detector. Each detector records where every particle that travelled through it,these points are recorded as "hits" in the detector. For a single event, the total number of hits recorded can reach up to $\mathcal{O}(10^5)$ at the high luminosity LHC and the amount of individual particles is usually around $\mathcal{O}(10^4)$. Connecting these hits together for individual particles eventually leads to the entire path a particle has traveled. Traditionally, particle tracking is done by searching for combinations of possible track segments, which is done by 'hand crafted' criteria [2]. Another method examines the hits in each detector layer and constructs possible tracks using Kalman Filters [2]. Both these methods perform well for today's LHC conditions, but will scale poorly if the amount of data increases.

This is where machine learning techniques can be used. By training models to find particle tracks on data, neural networks can predict tracks segments based on its training. Graph Neural Networks receives data in the form of hit coordinates, know as nodes, and track segments of the data, called edges. Combining edges between each node produces the full trajectory of all the particles in a data-set.

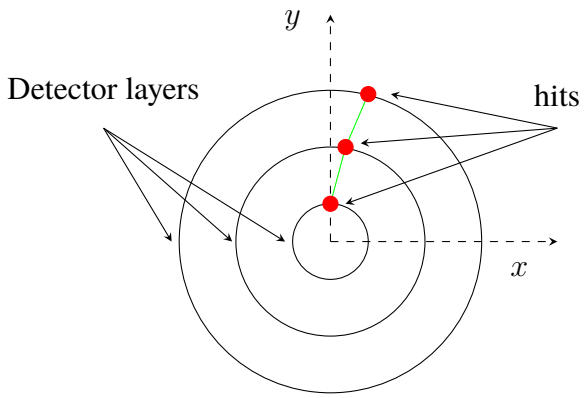


Figure 1: Example of a particle's track in a detector in an x-y plane. The black lines represent each of the detector layers where a particle passes through. The red dot in each layer highlights where on the detector the particle hits, resulting in a node. The connecting green lines illustrate an edge between two nodes.

Figure 1 is a simplified image of a particle's track in the detector. As the particle passes through the first layer in the detector it leaves a node in the position it hit. As it travels further it reaches other layers of the detector and continues to leave nodes. A particle may curve as well, which is due to its charge in the magnetic field in the detector. Then by connecting two nodes together, an edge can be constructed and by connecting these edges together, the full path of the particle can be modelled. This image, however, only illustrates one particle, in a single event there are

around $\mathcal{O}(10^4)$ different particles. This means that there will be a large amount of nodes and therefore edges to connect.

2 Graph Neural Networks (GNN)

In order to reduce the run time of classical particle tracking algorithms, various machine learning techniques have been tested. One method that proved to be efficient at finding possible track segments is Graph Neural Networks. This network is given a set of nodes and edges built from the hits left by particles in the tracking detectors and can then predict whether an edge is good or bad. The model learns this from training sets where each edge has a label of one or zero, a one implying a true track segment while a zero means a false segment. A true edge has a label of 1, which means that the two nodes, at the extremity, belong to the same particle. A false edge has a label of 0. This implies that the two nodes, at the extremity, belong to different particles. The trained model can then predict good edges, which have a value of one and bad edges which have a value of zero. Graph neural networks have multiple implementations such as node classification. This aims at classifying different nodes that belong together [3], while edge classification aims at predicting where individual particles travelled. However, the goal for both implementations is to correctly identify individual particles and their tracks. This investigation focuses on reconstructing particle tracks using edge classification.

2.1 GNN structure

The structure of the GNN consists of an input of features that are given to a neural network, and then produce a predicted label. The input consists of two nodes and their position in Cartesian and cylindrical coordinates (x,y,z,ϕ,r), so called features. Where ϕ and r are calculated by:

$$\phi = \arctan(y/x)$$
$$r = \sqrt{x^2 + y^2}$$

The neural network would then produce a label for each individual input. The process can be illustrated by:

$$(\mathbf{x}_i, \mathbf{x}_j) \longrightarrow f(\mathbf{x}_i, \mathbf{x}_j, \alpha) \longrightarrow Y_{ij}$$

Where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the features of two nodes. The function f represents a neural network which works on the two hit locations and contains trainable parameters, α. This process then produces a $Y_{ij} \in [0, 1]$. A cutoff value then determines a binary result for Y_{ij}, thus giving each edge a specific label of:

$$Y_{ij} = \begin{cases} 1 & \text{for a good edge} \\ 0 & \text{for a bad edge} \end{cases}$$

The trainable parameters α, represents the neurons in the network. Different neural networks use a different amount of neurons. Each neuron has a certain mathematical function which is determined by training the neural network. The training is done by fitting the network to a training set which includes the inputs, the coordinates of the two nodes making an edge, called the training features. The training set also contains the correct labels for each input, this is known as the training labels. During the training process, α is continuously updated by using stochastic gradient descent, which is an efficient method of fitting linear classifiers under convex loss functions [4]. This algorithm runs through the training data, for each input the parameters are updated by:

$$\alpha = \alpha - \eta \left[\beta \frac{\partial R(\alpha)}{\partial \alpha} + \frac{\partial L(\alpha^T \mathbf{x}_i + b, Y_{ij})}{\partial \alpha} \right] \quad [4]$$

Here α is the trainable parameters, while the function L represents the loss function for each training step and $\mathbf{x}_i$ is the input features. The loss function calculates how far apart the predicted value, $\alpha^T \mathbf{x} + b$ is from the actual value, the training label, Y_{ij}. β is the exponential decay factor, which has a value between 0 and 1. This parameter helps determine the weight change produced by the current and previous gradient descents. This factor is important as it prevents the weights from oscillating, when if the learning rate is too high. This learning rate is represented by the variable η. For classification, the usual learning rate is given by

$$\eta^{(t)} = \frac{1}{\beta(t_0 + t)} \quad [4]$$

Where t is the time step, where this is a total of $n_{inputs} \times n_{iterations}$, where $n_{iterations}$ is the amount of times the data is passed through the neural network. Using this stochastic gradient descent, the neural network can be trained to predict particle track segments or edges.

2.2 Edge Classification

The GNN is trained to classify which edges that are good and which are bad. It does this using the neural network. More specifically, a multi-layer perceptron classifier, imported from scikit-learn [5], is used to predict these binary labels for each edge. This type of network is discussed in the next section. The neural network is given a set of possible edge segments, then predicts which ones are good and which ones are bad. The edges that are useful after this classification are the good edges. By taking these good edges, a graph can be constructed, which will show the predicted trajectory of all the particles in the event. To identify individual tracks, however, the good edges have to be organised into consecutive segments. The segments are constructed by comparing different edges. Since each edge goes from one node to another, one can look at the coordinates of an outgoing node and compare which edge has the same incoming node. If they match, a segment can be constructed. By doing this for all the good edges that the network labelled, the individual particle tracks can be reconstructed.

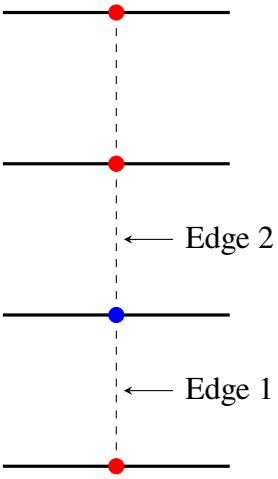


Figure 2: Image of how the individual particle tracks are constructed using the good labelled edges from the network.

Figure 2 illustrates how the segments are constructed. The red node belonging to edge 1 is the incoming node, while the blue node is its outgoing node. For edge two, the incoming node is the blue node. Since edge 1 has the same outgoing node as the incoming node of edge 2, their edges can be connected, creating a segment of a particle track. By doing this for each good edge produced by the network, all of the individual tracks can be found.

3 Multilayer Perceptron (MLP)

A multilayer perceptron is a type of feed forward neural network. It consists of three main layers, the input layer, the output layer, and the hidden layers that are in between the input and output layers [6]. The input layer receives the features of the data, which for this MLP, consists of two connecting node's coordinates. The output layer will then consists of labels, 1 or 0. Where a label of 1 would imply a good edge, meaning the two connecting nodes are from the same particle, where as a label of 0 is a bad edge. For reconstructing the particle tracks, only the good edges are useful.

3.1 Hidden Layers

The use of hidden layers in the MLP allows for nonlinear transformations of the inputs and work as different mathematical functions, which are designed to produce a specific output [6]. The illustration from figure 3 highlights how an MLP is constructed. The input layer is given some kind of features and from that produces an output through the output layer.

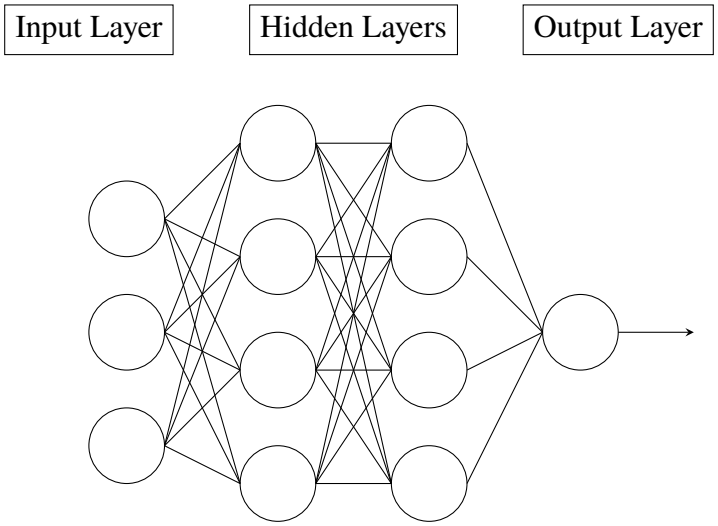


Figure 3: Illustration of an MLP with a hidden layer dimension of $(4, 4)$.

By training the model, using stochastic gradient descent, the hidden layers and their functions are weighted to produce similar outputs as the training labels. Stochastic gradient descent was used to train the model as it is more efficient on larger data-sets. This is based on both time it takes to train the model as well as producing a higher score according to [4]. The disadvantages of using this method however is that it requires tuning of different hyper-parameters, such as number of iterations. However, for this investigation the amount of iterations only affected the model if they were too low, thus not allowing the hidden layers to converge. Using 300 iterations, the model converged with $\mathcal{O}(10^5)$ inputs. The other weakness of stochastic gradient descent is the fact that it is poor at dealing with feature scaling. However, since each feature is a coordinate in the detector, feature scaling should not have a significant effect on the model in this case.

3.2 Determining the hidden layer dimensions

Since there is no exact formula to determine the best dimension of hidden layers for an MLP, the best way to determine it was to optimize the score on a control training and test set and determine which configuration of hidden layers would score the highest. A training set containing $128,000$ possible edges was used to fit different models, afterwards these models were tested on a test set containing around $7,000$ possible edges. These models were then rated using a score function, which calculates the amount of correct predictions the MLP has done and compares it to the total amount of inputs in the data.

$$Score = \frac{\#\text{Of correctly predicted edges}}{\text{Total number of edges}}$$

From figure 4 it is easy to see that the dimension that had the highest score is the $200 : 100$ dimension having a score just above 0.964 of correctly predicted labels. It is important to notice however that the lowest score of these points is just around 1.2% less than the highest while their hidden layers dimensions are much smaller. These data underline that the hidden layer

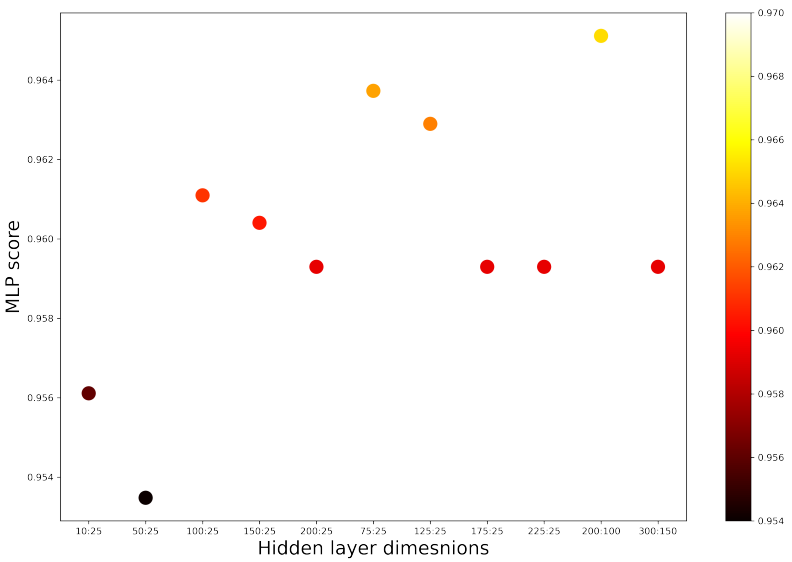


Figure 4: A variety of different hidden layer dimensions on the x-axis, with their corresponding test scores on the y-axis

dimensions do have some affect on the overall test score, however, it is not a significant factor. Using this data the hidden layers with a $(200, 100)$ dimension were used in creating the actual MLP for particle tracking.

4 Data

The data used in this investigation was the Kaggle TrackML Particle Tracking Challenge [7]. Their data-sets are public for everyone and are a simulation of proton-proton collisions [8] at the high luminosity LHC. It uses an approximate geometry for the detector planned for that phase for ATLAS, and simulating the production of top and anti-top quarks. The challenge was constructed to innovate methods of particle tracking. The data-sets used in this investigation consist of *detectors*, *hits*, *particles*, and *truth* information.

Detectors The detectors file is a description of the detector used in the simulation. It contains the coordinates of each detector in Cartesian coordinates. The entire detector is also grouped into three categories, volume ID, layer ID, and module ID. The volume ID is a numerical label for each detector group. There are in total nine different detector groups, however this investigation only looked at the most central detector groups. These were labelled 8, 13, and 17. Figure 5 demonstrates this configuration of the detector. The layer ID is again a numerical label for the detector layer inside each group [7]. The module ID is another label for each module inside the layer of each detector.

Hits The hits file contains information about where each particle hit a detector. After a hit is registered by a detector, it is labelled with a hit ID. Each hit ID produced the location of this hit

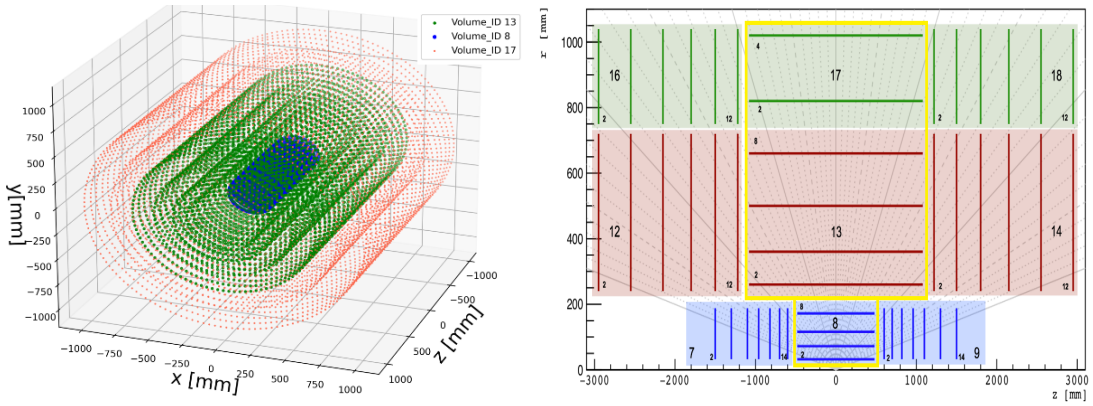


Figure 5: Illustration of the different detector volumes. Volume 8 consists of pixel detectors, and 13, 17 are silicon strips. The image on the **left** is a 3D image of the detector volumes used in this investigation. The image on the **right** illustrates the complete detector projected in the r-z plane. The outlined yellow area highlights the 8^{th},13^{th}, and 17^{th} volume ID. This image is adapted from a figure in [7].

in Cartesian coordinates as well as in which volume, layer, and module the hit is.

Particles The particles file provides information about each individual particle. Every unique particle in the data-set contains a specific particle ID. The file contains the initial position of each particle in the (x, y, z) coordinates, in mm, as well as its momentum, in $\frac{\text{GeV}}{\text{c}}$, in the (x, y, z). It also reports the charge of the particle, by either having a charge of 1 for positive and -1 for negative. Furthermore, the particles file also highlights the amount of detectors each unique particle hits.

Truth The last type of file used is the truth file. The truth file connects the hits and the particles files in the sense that it contains the hit ID of each hit as well as where the particles originate from. For each particle ID, this file also consists of the particle's true momentum in three directions, (x, y, z). The importance of each hit, is reflected in the truth file as well. Each hit has a weight from zero to $\mathcal{O}(10^{-4})$. The particles that have three hits or less are given a weight of zero.

4.1 Data filtering

The raw data from [7], can consist of up to $\mathcal{O}(10^5)$ hits and $\mathcal{O}(10^4)$ different particles. To reduce the amount of data the network operated on, this was achieved by using various filters.

1. The first filter removes all the hits which have a particle ID of 0 as according to [7] the these hits do not originate from a reconstructable particle, and they should therefore be identified as noise from the detector.
2. For the sake of simplicity, the second filter removes all particles which are not in the volumes highlighted in figure 5, 8, 13, or 17.
3. The third filter removes all hits with a weight of 0. This been done to remove all the particles leaving 3 hits or less, a particle has to have 4 hits or more to get a weight and therefore become interesting to reconstruct.

4. The fourth filter removes one hit per particle if the particle has hit the same layer more than once in one volume. This can happen due to double sensitive layers.

5. The fifth filter removes all particles that have a transverse momentum, pT, less than 1GeV, where pT is given by the formula:

$$pT = \sqrt{p_x^2 + p_y^2}$$

   We investigate two different pT filters. First, for simplicity, the particles that have a range of $pT \in [1, 1.25]$ GeV are considered. Afterwards we look at all the particles with $pT \geq 1$ GeV.

4.2 Data after filtering

When using a dataset with 100 proton collisions, which is 100 events there is an average of 109675 hits in total as seen table 1. As each filter processes the data, the two most significant filters that remove the most is the first filter and the last filter.

Events used	50	100
Avg input hits	111104.0	109675.0
Avg hits post F1	(60401.0, 54.39%)	(59670.0, 54.44%)
Avg hits post F2	(47387.0, 42.57%)	(46656.0, 42.46%)
Avg hits post F3	(46079.0, 41.39%)	(45377.0, 41.29%)
Avg hits post F4	(33752.0, 30.32%)	(33253.0, 30.26%)
Avg hits post F5	(1809.0, 1.62%)	(1775.0, 1.61%)
Total particle count	16052	31473
Total nodes count	90428	177519
Maximum possible true edges	74376	146046

Table 1: Average amount of hits passed through each filter for 50 and 100 events, with filter 5 meaning $pT \in [1, 1.125]$ GeV. The total amount of unique particles and nodes is illustrated in row 7 and 8 respectively.

The third filter is the filter with the lowest impact. The reason why this weight equals 0 filter doesn't have a larger impact is because the most of the hits where the weight is equal to 0 gets removed when filter 1 is implemented. The fifth filter has the largest impact as it reduces the remaining $\sim 30\%$ of the total hits down to just $\sim 1.6\%$. This interval in pT has been selected based on the typical minimum pT value used by the ATLAS experiment, and a wish to focus on a small amount of hits to start with. This reduces the final average hits down to 1775, from all of the 100 events. The impact of the other pT filter, of 1 GeV and above, is illustrated in table 2. Due to the larger range, the fifth filter in table 2 reduces the final amount of hits to just below 5%.

Events used	50	100
Avg hits after filter 5	(5202.0, 4.67%)	(5105.0, 4.64%)
Total particle count	45162	88488
Total nodes count	260114	510504
Maximum possible true edges	214952	422016

Table 2: Average amount of hits passed through filter 5 for 50 and 100 events, with filter 5 being $pT \in [1, \infty]$ GeV. The total amount of unique particles and nodes is illustrated in row 2 and 3 respectively.

All the drops from filter one to four can be seen in table 1, and the only difference between table 1 and table 2 is the fifth filter.

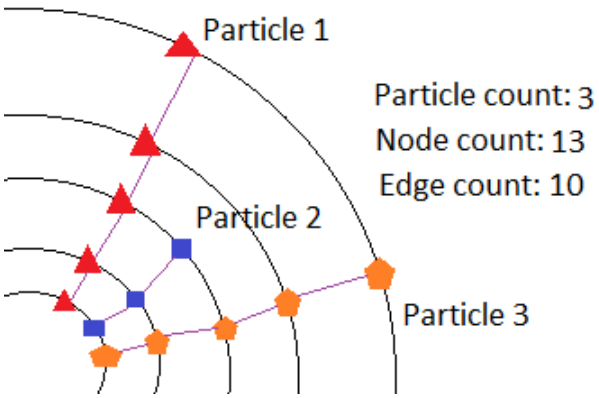


Figure 6: Image to understand how the maximum amount of true edges has been calculated

In figure 6 one can see how it is possible to calculate the maximum amount of true edges there can be with a set amount of particles and nodes. In figure 6 there are 3 different particles, with a total amount of 13 nodes. Since the particles can only go one way, and a true edge are between two identical particles, one can calculate the maximum amount of true edges by subtracting the particle count from the node count. This amount of true edges is the expected amount of true edges in the data, given the amount of nodes and particles.

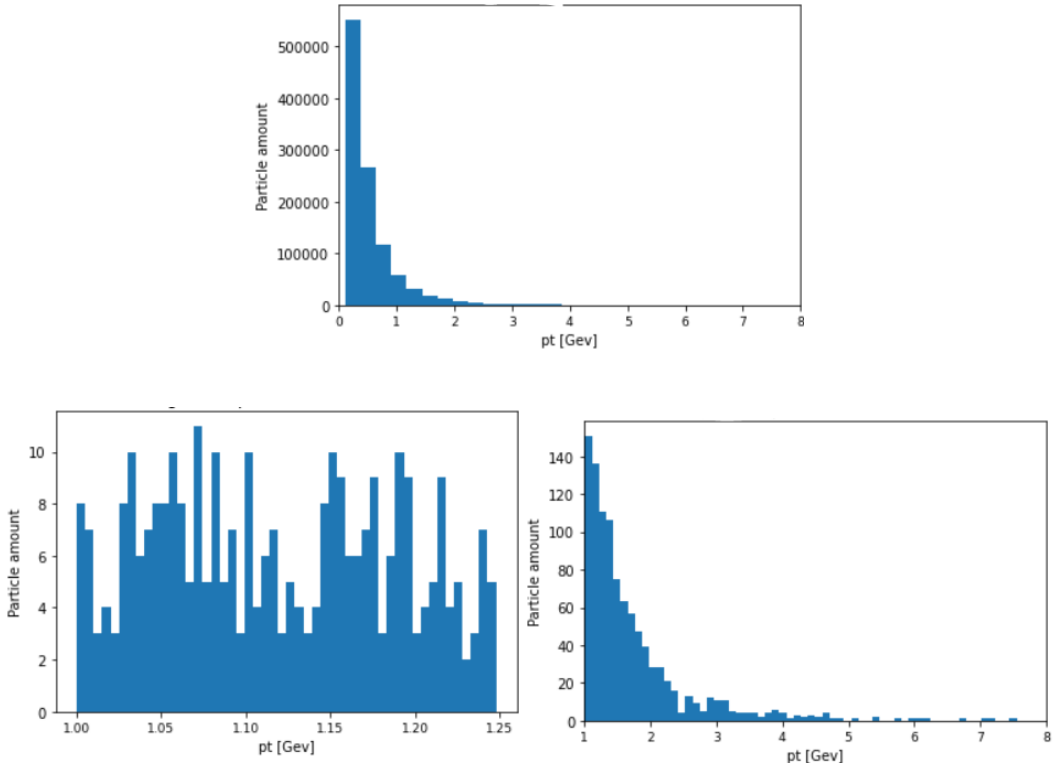


Figure 7: **Top:** This histogram shows the pT distribution before any filters, with a total particle count at 1079146. **Bottom left:** In this histogram the pT distribution of $pT \in [1, 1.25]$ GeV, with a total particle count at 31473. **Bottom right:** In this histogram the pT distribution of $pT \in [1, \infty]$ GeV. The y-axis of all histograms shows the amount of particles for every bin, with 100 events used, with a total particle count at 88488.

In figure 7 one can see how the fifth filter reduces the amount of hits in the two intervals. This can be compared with the top histogram, which illustrate the transverse momentum before any filters. In the histograms before the filters and where $pT \in [1, \infty]$ there is a limit on the x-axis at $pT = 8GeV$ to make the images clearer. In both instances there where 725 hits at $pT > 8GeV$. It is clear that by applying a lower cutoff at 1GeV most of the data is removed.

5 Finding Possible Edges

5.1 Restricting the curvature of the particle

The implementation of the GNN has the purpose to classify good and bad edges, to do this the data has to be converted into possible edges to be tested. This could be done by comparing each node to each other in the data, which constructed n_{nodes}^2 possible edges. To limit this amount of possible edges, two different filters are applied. The first is the value of $|\frac{\Delta\phi}{\Delta r}|$, which restricted how much the particle can change direction for a distance of Δr. Here we also apply the condition that $\Delta r > 0$ as the edges all travel outward. Since all the particles in this investigation have a pT of 1 GeV or higher the particle trajectories are not expected to have a large curvature. A similar method was used in [9]. However [9] restricted the curve of the particle in the r-z plane as the particles travel in a straight line in this plane.

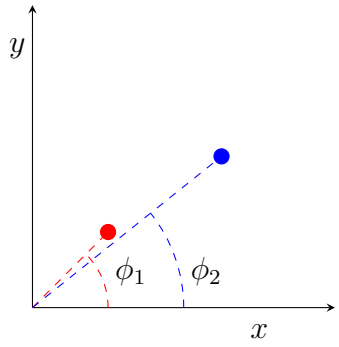


Figure 8: Illustration of how the curve of the particle can be quantified by the ϕ angles in the x-y plane. ϕ_1 belongs to the red node, while ϕ_2 is from the blue node. ϕ is calculated by $\arctan\left(\frac{y}{x}\right)$.

Figure 8 illustrates how the angle of each hit is calculated. The red line connecting each node to the origin illustrates the radius of that node, denoted by r. The angles of the nodes are represented by the arches, the red being ϕ_1 and the blue is ϕ_2. By using these values, the curve of the particle was measured by $\frac{\phi_2-\phi_1}{r_2-r_1} = \frac{\Delta\phi}{\Delta r}$. The absolute value for this limit was chosen to be smaller than $0.00030035\frac{\text{rad}}{\text{mm}}$. This was decided based on the expected amount of true edges using the formula from figure 6 where the amount of true edges is equal to the difference between the number of nodes and the number of particles. This had an efficiency of 94.4% meaning that of all the true edges, 94.4% of them were passed through the filter. While this value is not optimal as it removes some of the true edges, it also reduced the amount of possible edges from $\mathcal{O}(10^6)$ to $\mathcal{O}(10^4)$ for one event.

Figure 9 illustrates the effect of applying the $\frac{\Delta\phi}{\Delta r}$ filter. The image on the left reflects the amount of true edges there are in total, before applying the filter. It is noticeable that there are some bins, on the left image, that exceed the aforementioned value of this filter. This clarifies why the efficiency is 94.4%, however, increasing the value for $\frac{\Delta\phi}{\Delta r}$ would also limit the amount of false edges removed from by the filter. The image on the right of figure 9 underlines that most of the true edges are still passed through the filter, however, the cut is visible at the 0.0003 mark on the graph.

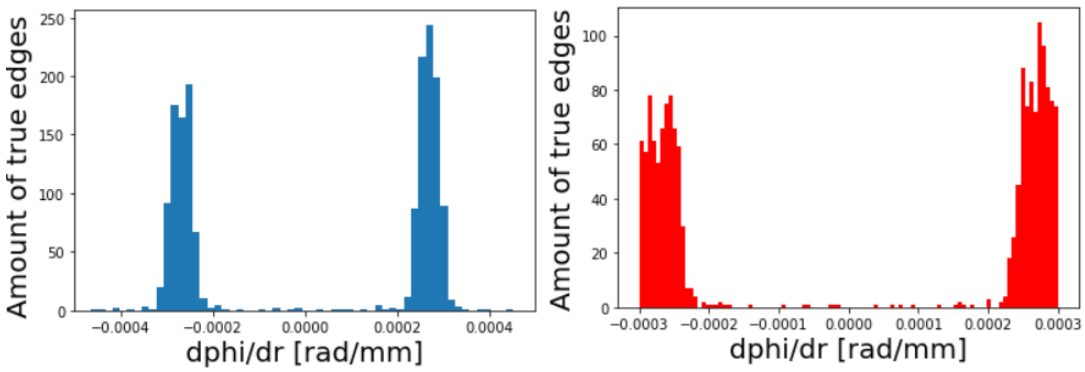


Figure 9: **Left** Image illustrating the range of $\frac{\Delta\phi}{\Delta r}$ and the amount of true edges in a single event. The image on the **right** is the distribution of $\frac{\Delta\phi}{\Delta r}$ for true edges after the filter.

5.2 Restricting the Z intercept of the edge

The last filter is used to restrict the value of of the Z intercept of possible edges, calculated by

$$Z_0 = Z_i - r_i \frac{Z_f - Z_i}{r_f - r_i}$$

Where i describes the initial coordinate, while f is the final coordinate and r is radius of the detector layer. This method was used in [9] "to eliminate highly oblique edges". Figure 10 demonstrates this z-intercept.

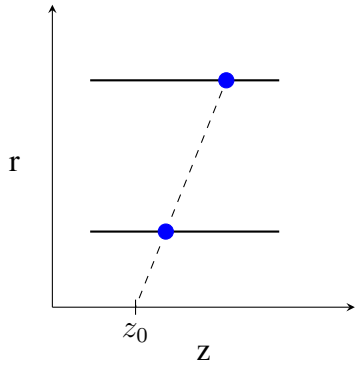


Figure 10: Illustration of Z_0 for two arbitrary nodes. The edge connecting the two nodes in the r-z plane has to be close enough to the origin, where the protons collide.

An upper cut to the absolute value of Z_0 is applied as a filter to allow 100% of the true edges in the data. The value of this filter was at 249 mm. This had an efficiency of 100% as all of the true edges were passed through. The filter also removed any extreme false edges. This value for Z_0 was determined by calculating the amount of true edges passed through a range of different Z_0 values.

Figure 11 demonstrates the limit for Z_0. It is clear that the graph reaches 100% near the 100 mm mark, but ceases to increases after this. This illustrates that as long as the value of the cut for

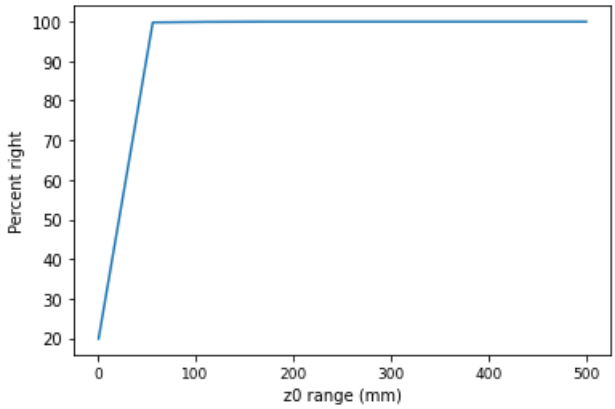


Figure 11: Optimization of Z_0, to find the limit of Z_0, which produces 100% true edge output. Z_0 is the z-axis intercept, considering the particle originates at the origin $(0, 0, 0)$ it is expected that the z-intercept should be around this point.

Z_0 is 100 mm or larger, all the true edges should be passed by the filter. However, the chosen absolute value for the upper cut is 249 mm, which allowed for a good amount of false edges as well as all of the true edges for training the data set.

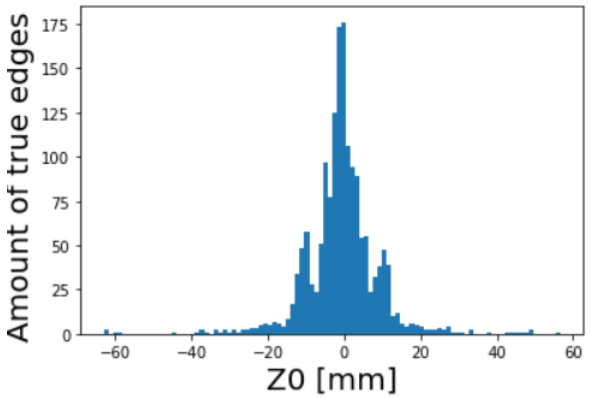


Figure 12: The range of Z_0 values that all the true edge in the data we have.

Figure 12 illustrates how small the range of Z_0 is for all the true edges. Most of the edges are distributed around the origin, which is expected as that is where the protons collide. This provides a better understanding of why the efficiency of this filter is 100%, as none of the true edges are anywhere near the limit for this filter. By applying these two filters reduces the ratio of true to false edges to $1.19 : 1$, which means when training the network, the data will not be over-saturated with false edges.

6 Results

The results from this investigation were promising. After training the model, it was able to predict the label of each input in a single event at an average time of 16.98 ± 1.16 ms. This was for an event with a pT range layer of $[1, 1.25]$ GeV. For the test event with no upper limit on the pT range, the average run time was 48.10 ± 3.12 ms. These tests were both run on an intel core i7 10^{th} gen 10700kf CPU with 8 cores. The increase in run time can be explained due to the larger amount of inputs. The second set had just over $5,000$ more inputs than the other. However, these low run times illustrate the efficiency of a graph neural network for edge classification.

6.1 Testing the MLP

6.1.1 Using a pT range of $[1, 1.25]$ GeV

Using the optimized hidden layer dimension distinguished above, the model was trained on 95 events, which in total contained around $254,467$ possible edges, $138,122$ being true and $116,345$ false edges. Afterwards the model was tested on a single test event which contained $2,813$ possible edges. The model managed a score of 0.969, predicting $1,531$ good edges in a test set that had $1,512$ true edges. To check how many of the true edges it predicted correctly we used the formula:

$$Purity = \frac{\#\text{of true and good edges}}{\#\text{true edges}} \quad [2]$$

From this formula we found that the purity of the model was at 0.978. This value implies that the MLP labelled 97.8% of all the true edges as good in the data. This meant that out of the total $1,512$ true edges, the MLP labelled 19 of them as bad. This along with the fact that the model also labelled 53 edges that were false as good, underlines the score of 0.969.

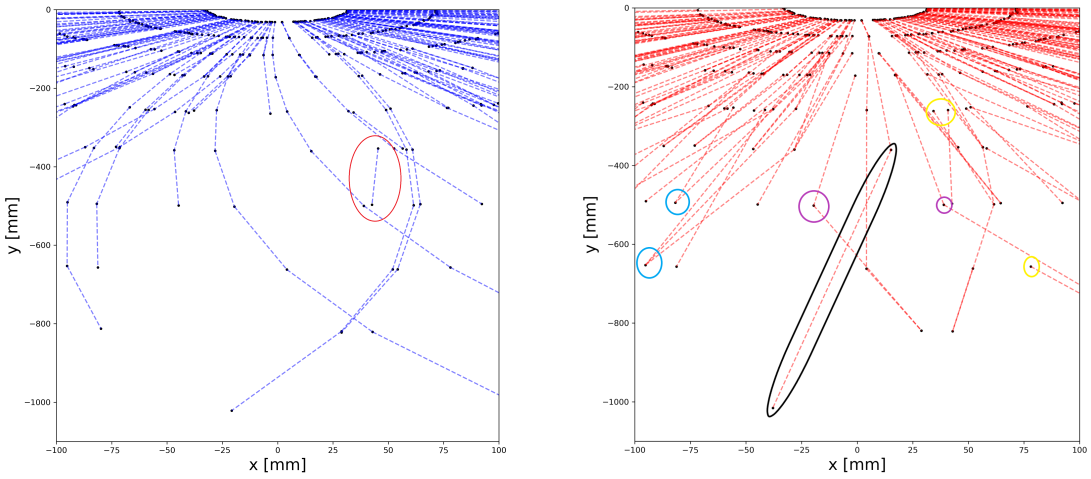


Figure 13: **Left** The image on the left is a close up of the good edges that the model has predicted. It has been zoomed in to get a better understanding of which kind of edges the model labels as good. The edge in the red circle on the left is an example of the 53 edges labelled good but were false. **Right** The right image represents the bad edges, in the same range, that the model predicted. The highlights made on the right show some of the characteristics that the bad edges have. The entire image of these figures can be seen in the appendix.

Figure 13 illustrates the way in which the neural network predicts good and bad edges. An important difference between the two images is the fact edges are connecting to different nodes. The image on the right, of the bad edges, has multiple edges that reach the same node, this is highlighted in blue. Another distinction between the two images is the fact that the good edges do not skip a detector layer. Each good edge is connected to a node in the next layer, which is ideal for a particle travelling outward, as it will hit every detector in its path. A good example of a bad edge that skips several detector layers is enclosed by the black figure. This edge starts at the sixth layer and reaches the tenth, meaning it skips three layers. There is also a difference in how the good and bad edges change direction. The good edges curve in the same direction, without any abrupt changes, while the bad edges change direction much more suddenly. Some of these sudden and sharp direction changes are represented by the purple circles. On the left of figure 13 also has a good edge, which is obviously false. This is highlighted by the red circle. The edge is labelled good, but should be labelled bad as it is not connected to anything else. This illustrates that the model is not perfect, but the significant difference between the good and bad edges emphasizes that the model is still able to predict edges with a decent consistency.

6.1.2 Using a pT range of $pT \geq 1.0$ GeV

This model was trained on a training set which contained 47 events. This training set consisted of $136,144$ possible edges, with $73,855$ edges being true and $62,289$ false edges. The model was constructed with the same configuration as the $pT \in [1, 1.25]$ GeV. This meant that the hidden layer dimensions were $(200, 100)$ and ran on 300 iterations. This test set was also constructed from a single event, it contained $8,271$ possible edges. The total number of true edges in this set were $4,306$. The purity of this model was also extremely high, having a purity of 0.992 only missing 35 true edges. However, the model had a score of 0.954, which means it labelled a high amount of false edges as good. The exact amount of false edges that the model labelled good was 344.

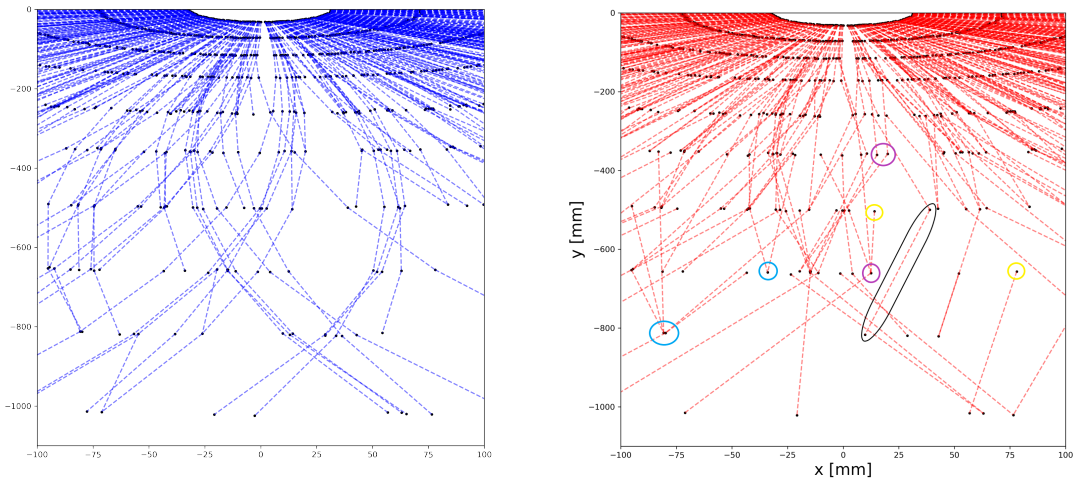


Figure 14: The image on the **left** illustrates the edges that the network labelled good, in the same range as 13. The image on the **right** demonstrates the bad edges labelled by the network. Again the highlighted points reflect the characteristics of the bad edges. The entire image of these figures can be seen in the appendix.

The images in figure 14 are similar to those in figure 13. However a noticeable difference in these two figures is the amount of edges on each graph. The good edges are also similar to 13, however, there are a lot more reaching the final detector. This is also expected as there are more particles with a higher transverse momentum than in the previous results. The bad edges in figure 14 are also similar to the bad edges in 13. The similarities of the bad edges in figure 14 and those in 13 underlines that the neural network is consistent in labelling bad edges. On the left of figure 14 there are also some edges that stop before they reach all of the detectors, this can be due to the fact that they hit other detector volumes, which were filtered out in the second data filter.

Figure 15 provides a better understanding of why some of the good edges in figure 14 and figure 13 are shorter than others. The blue edges closest to the left hit the last part of the last detector, while the blue edges to the right do the same for the right side. This illustrates the range in which a trajectory requires in order to hit each of the ten detectors used in this investigation. The image also underlines that the further away from the origin the green edges are, the shorter their trajectory is as they become filtered. There are also some green edges that reach the last

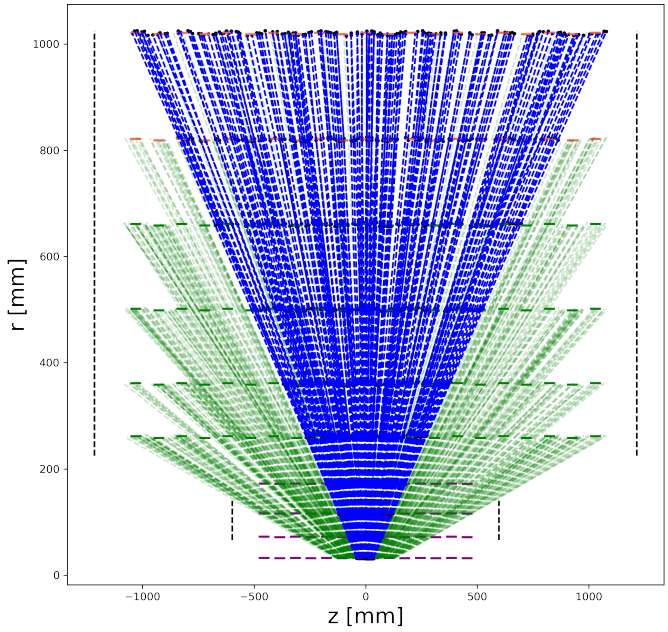


Figure 15: Image of all the good edges in the test event for $pT \geq 1$ projected in the r-z plane. The blue edges illustrate all those that hit all ten detectors, while the green edges do not hit all ten. The black lines on either side of the edges illustrate each of the nearest detectors that were filtered out.

detector, however, these edges do not hit all ten. These could represent some of the 344 edges that the model labelled good and could be the model predicting that they are good edges after passing some of the first detectors.

6.2 Tracking Particles

To show the candidate tracks constructed from good labelled edges, it was needed to connect the edges. By using the aforementioned method of comparing the incoming and out-coming node for each good edge the MLP classified, it was possible to create candidates of each particle's individual track.

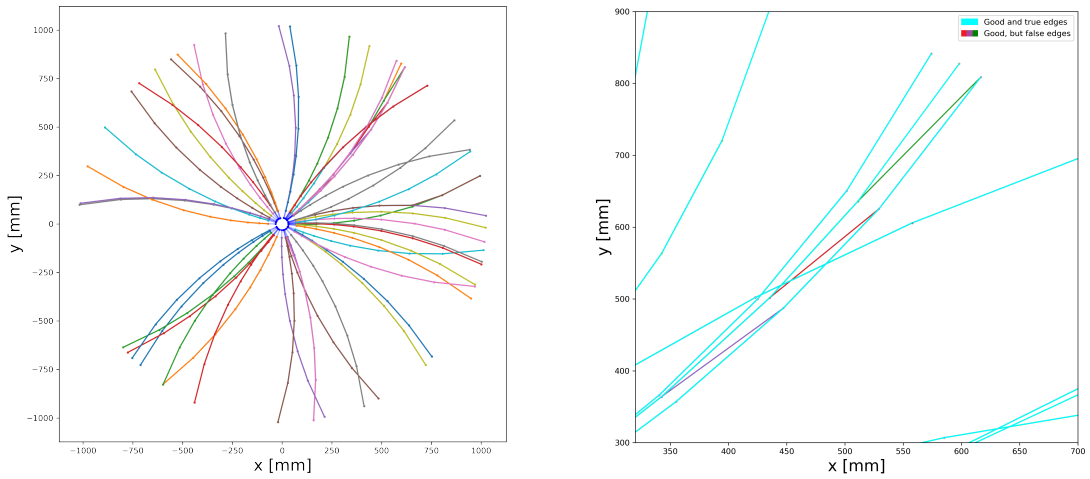


Figure 16: (**Left**) Reconstructed tracks for particles that hit each detector in the test event with $pT \in [1, 1.25]$GeV, each path represents the trajectory of a unique particle. The image on the **right** is a close up of the **left** image. It illustrates some of the errors that the network has made. The light blue curves reflect good edges that the network labelled good, while the lines with different colours are edges that the network labelled as a good edge, but were false. A full sized image of the figure on the right can be seen in the appendix.

Figure 16 illustrates how a GNN can be used in the reconstruction of particle tracks. Most of these reconstructed tracks are in their expected form as they all follow a curved trajectory in the x-y plane. However, since the network did not achieve a perfect score, there has to be some errors, this is reflected by the image on the right in figure 16 where the red, green, and purple lines represent edges that were labelled incorrectly as belongin to a track candidate. Figure 16 demonstrates that for edges that are extremely close together, the network has a difficulty of connecting the nodes together, even though it also finds the actual track of the particle. This type of error can account for why the model predicts more good edges than there are true ones.

7 Conclusion

In this work, we implemented a neural network, in the form of GNN using a multi-layer perceptron, to classify good and bad edges. The network was tested and trained using the public trackML data-sets [7]. From these data-sets, each edge is constructed by connecting two hits together, if the hits belonged to the same particle, the edge is a true edge and false if they belong to different particles, in order to guide the training. By focusing on the good edges that the network predicts, the trajectory of particles can be reconstructed. However, to reconstruct trajectories of particles, the edges are connected into track candidates. Those that had the same incoming and outgoing nodes were combined together.

The network illustrated a great efficiency by only requiring a time of $\mathcal{O}(10^1)$ ms to classify all the edges in a test set. The precision of the neural network was also reflected in the resulting purity and scores. These models were only trained on 95 and 47 events, but still managed to produce a purity of 97.8% and 99.2% respectively and a score of 0.969 and 0.954. The fact that the scores were lower than the purity implies that the model found more good edges than there were true edges. According to [2], to be able to compete with traditional tracking algorithms the network has to predict all the labels in less than one second. This was achieved for both models as they only required around 17 ms and 48 ms. However this was for around 2,800 and 8,200 edges respectively. In [8] they demonstrate that their neural network can classify $9,080.7 \pm 3,027.1$ edges in 3.83 ± 0.89 ms on a 12-core Intel Xeon CPU E5-2650 v4, which does have 4 more cores than the CPU we used. However, this is still almost 13 times faster than our result with $8,200$ edges, highlighting the fact that there is still significant room to improve.

This investigation focuses on volumes 8, 13, and 17 where the model could predict edges with good score and purity. An extension to this could be to explore how well the model would perform if it was trained on the whole detector. If the model can still predict edges with a high purity and score, then it would prove to be highly effective for particle track reconstruction. The only disadvantage with creating this model rather than the one we used is the fact that it needs to be trained on much more data, which will require much more time.

Our neural network only uses edge classification network when predicting potential edges. A possible improvement to this method could be to construct a GNN which uses both an edge classifier as well as a node classifier. This method would work in a way that both the edge classifier and node classifier have to agree whether the two nodes belong to the same particle. The node classifier, would predict which nodes are left by the same particles, while the edge classifier would connect the correct nodes. This method would help decrease the amount of good edges that were actually false from this investigation as it would require both classifiers to have the same prediction. One limitation this method could have is the run time, as it would consist of two networks rather than just one.

8 References

References

[1] Vidal, Xabier Cid, and Ramon Cid Manzano. "Taking A Closer Look At LHC - LHC Data Analysis". Lhc-Closer.Es, https://www.lhc-closer.es/

[2] Choma, Nicholas et al. "Track Seeding And Labelling With Embedded-Space Graph Neural Networks". 2022, Accessed 8 June 2022.

[3] Duarte, Javier, and Jean-Roch Vlimant. "Graph Neural Networks For Particle Tracking And Reconstruction". 2020, Accessed 8 June 2022.

[4] "1.5. Stochastic Gradient Descent". Scikit-Learn, 2022, https://scikit-learn.org/stable/modules/sgd.html

[5] "Sklearn.NeuralNetwork.Mlpclassifier". Scikit-Learn, 2022, https://scikit-learn.org/stable/modules/generated/sklearn.neuralnetwork.MLPClassifier.html.

[6] Bento, Carolina. "Multilayer Perceptron Explained With A Real-Life Example And Python Code: Sentiment Analysis". Multilayer Perceptron Explained With A Real-Life Example And Python Code: Sentiment Analysis, 2021, https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.

[7] Rousseau, David et al. Participant Document: Particle Tracking And The Trackml Challenge. Trackml, 2018, https://www.kaggle.com/competitions/trackml-particle-identification/overview, Accessed 10 June 2022.

[8] DeZoort, Gage et al. "Charged Particle Tracking Via Edge-Classifying Interaction Networks". Computing And Software For Big Science, vol 5, no. 1, 2021. Springer Science And Business Media LLC, https://doi.org/10.1007/s41781-021-00073-z. Accessed 10 June 2022.

[9] Tüysüz, Cenk et al. "Hybrid Quantum Classical Graph Neural Networks For Particle Track Reconstruction". 2021, Accessed 9 June 2022.

A Full sized images

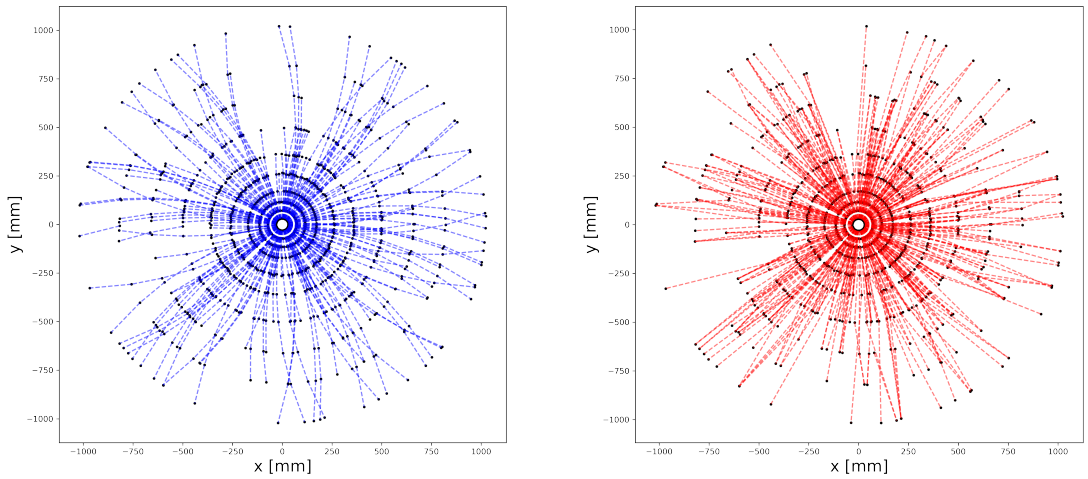


Figure 17: Full sized images of figure 13.

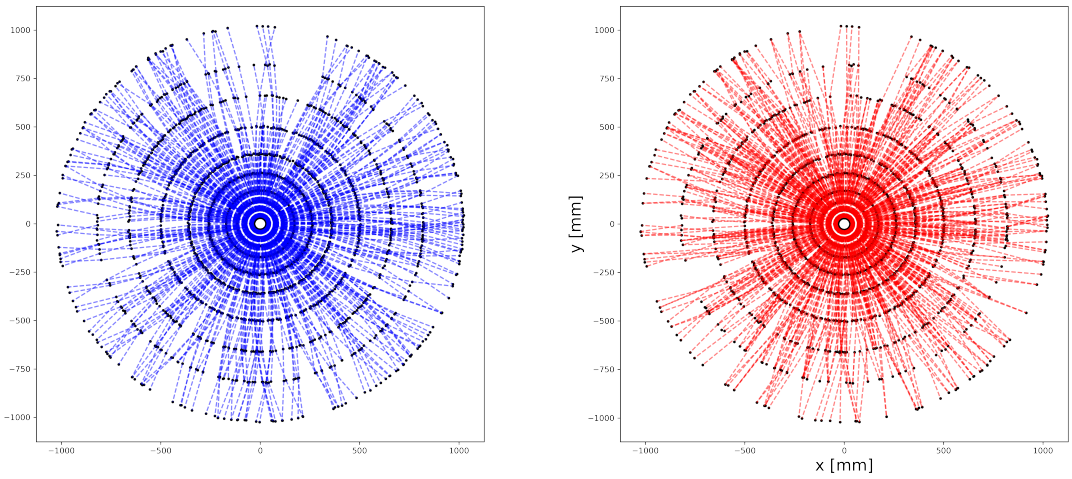


Figure 18: Full sized image of figure 14

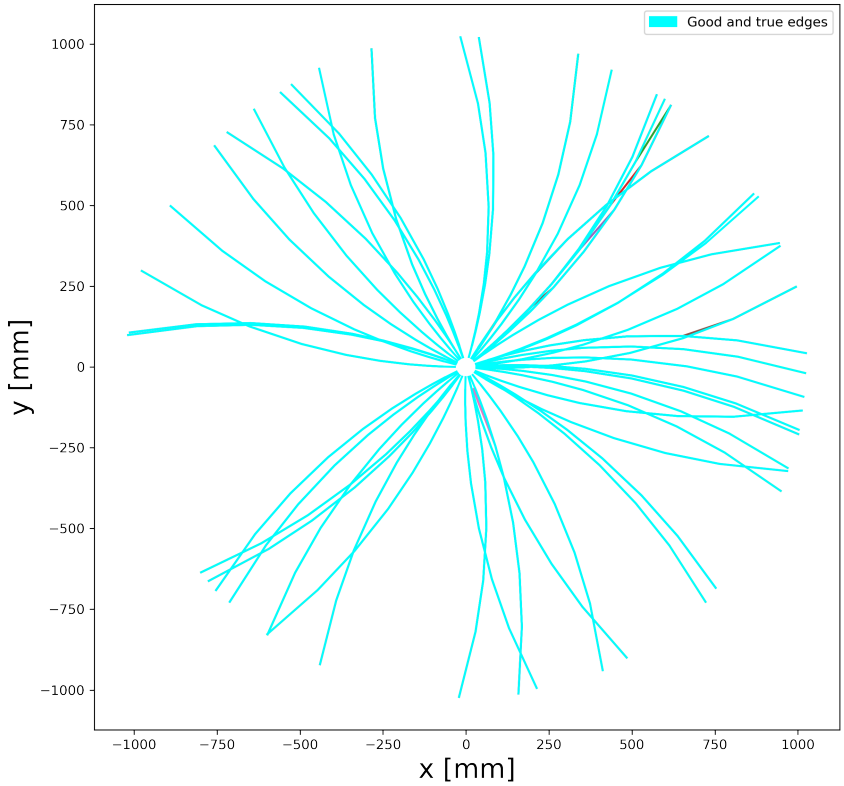


Figure 19: Full image of figure 16