Particle Identification at the EIC dRICH Detector Using Deep Learning

by

Omar Hassan

A thesis submitted in partial fulfillment of the requirements of

PHYS 4678 *& a* Bachelor's of Science (Hons) *degree*

in the

Department of Physics & Astronomy

University of Manitoba

Winnipeg, Manitoba

May 1, 2023

Abstract

Particle Identification at the EIC dRICH Detector Using Deep Learning

by

Omar Hassan

Supervisors:

Dr. Wouter Deconinck (Department of Physics & Astronomy)

Dr. Zhiyang Zhou (Department of Statistics)

The identification of particle types at the dual-radiator Ring Imaging Cherenkov Detector detector of the Electron-Ion Collider is a critical part of the facility's operation. We seek to investigate the potential of implementing deep learning techniques to solve this problem. A convolutional neural network is used with a sparse input for improved memory efficiency, utilizing Minkowski Engine. With the magnetic fields enabled we find that our model performs best with protons and electrons, with average accuracies of 94.71% and 97.68% respectively, this is followed by kaons at 81.41%, and the worst accuracies belonging to protons at an average of 6.19%. We discuss the discrepancy of accuracy which vary with particle types, alongside improvements to our model that potentially increase its computational efficiency and accuracy.

Contents

Abstract 1

Contents 2

List of Figures 4

List of Tables 5

Acknowledgements 6

1 Introduction 7

2 The Electron-Ion Collider 8

2.1 Cherenkov Radiation 9

2.2 dRICH Detector 10

3 Data Generation & Analysis 11

3.1 Generation 11

3.2 Analysis & Cleaning 12

4 Sparse Tensors 13

4.1 The Sparse COO Format 14

4.2 Minkowski Engine 14

5 Convolutional Neural Networks 15

5.1 Linear Layer 15

5.2 Loss & Gradient Descent 16

5.3 Dropout & Overfitting 17

5.4 Convolution & Pooling Layers 18

5.4.1 Convolution 18

5.4.2 Pooling 19

5.5 Hyperparameter Optimization 19

6 Results 20

6.1 The Dense Formulation 20

6.1.1 Model 20

6.1.2 Performance 21

6.2 The Sparse Formulation 22

6.2.1 Model 22

6.2.2 Performance 23

6.3 Discussion & Limitations 24

7 Conclusion & Future Work 25

8 Appendix 27

References 28

List of Figures

1 Schematic of the planned EIC facility [3]. 8

2 Cherenkov radiation cone produced when a charged particle exceeds the speed of light in a medium [3]. 9

3 a) The dRICH detector. b) A single sector from the dRICH detector showing the aerogel & C_2F_6 gas radiators [4]. 10

4 The different particle types and their respective radii. The difference in ring size by radiator (gas or aerogel) and magnetic fields is shown in the plots [4]. . 11

5 A sample output of 9 electron events from the generated data. 12

6 A sample output of 9 electron events after the filtration process. 13

7 How data is stored in a sparse COO format [8]. 14

8 Visualization of a stochastic gradient descent [16], the black line is the path downwards to the minima through different iterations. The two θs represent the weights and biases. $J(\theta)$ is the loss function. 17

9 a) The standard neural network with all neurons online b) The neural network after applying dropout, with a number of neurons offline [18] 18

10 Visualization of the model, including the different layers. 20

11 Plots of the loss by epoch for 100 epochs, as expected the longer we train the data the lower the loss and consequently, the accuracy. 21

12 Visualization of the model, including the different layers. 22

13 a) Loss by epoch without dropout layers b) Loss by epoch with dropout layers . 23

14 Sample of filtered kaon events. 24

15 Sample of filtered proton events. 25

16 The computational efficiency of an octree based model vs the full voxel model [21]. 27

17 The memory efficiency of an octree based model vs the full voxel model [21]. . 27

List of Tables

1 The parameters used for generating the data and their respective values. 11

2 The hyperparameters used and their respective values (dense case). 21

3 The hyperparameters used and their respective values (sparse case). 22

4 Accuracies per particle type when the magnetic field is disabled and enabled. The uncertainty is given by the sample standard deviation, averaged over 10 samples. 23

Acknowledgements

I would like to start off by thanking the faculty and staff who enabled my education for the past few years, I have learned more than I could possibly imagine during this degree. Professors like Dr. Gerald Gwinner, Dr. Khodr Shamseddine, Dr. Michael Gericke, Dr. Danielle Pahud, Dr. Roy Roshko, and Sakib Rahman, all supported me in my degree. And a big thank you to Dr. Samar Safi-Harb & Dr. Ruth Cameron for taking the time to help me progress through my thesis.

I would like to thank my supervisors, Dr Wouter Deconinck to whom I am beyond grateful, I have not only developed so much under your tutelage and guidance, but I have also gained the confidence needed to pursue my higher studies. And to Dr. Zhiyang Zhou, I could not have done this without you and I am very thankful that you chose me as your student and agreed to supervise me.

My research journey first started with Dr. Jesko Sirker, who deserves a special mention here. You have always been ready to support me and my development as a scientist, ready to provide me with opportunities to work with you ever since 2018 up until now. I am very grateful to you for that.

My peers and friends have made this entire experience fulfilling and worth it. Friends like Devon, Tommy, Kulvir, Yug, Hemakshi, and the people in OPUS. There are bound to be people that I missed, but all of you have had an impact on me and allowed me to be the best version of myself that I can be.

Finally, to my family. Your support and belief in me were ever present, I would not be here without you. Thank you, Dad, Mom, and Mohammed.

1 Introduction

Particle identification is a critical part of the operation of any particle accelerator. A great deal of work has been done in developing and designing detectors that can capture events with a high level of accuracy and efficiency. We wish to apply deep learning techniques for particle identification using the data gathered from these detectors, with a particular focus on convolutional neural networks. The end goal is to develop a model that can provide continuous particle identification with high accuracy and low latency; latency is defined here as the data pileup occurring due to lower computational speeds. Python is a good interface for prototyping our model but deployment will be done in C++ to improve the latency.

Machine learning is an immensely powerful field with far-reaching applications. Within experimental high energy physics, these applications have already started materializing with many examples demonstrating the power of machine learning, where it has been used as a tool for particle track reconstruction and optimizing detector design for our dRICH detector [1]. There are plenty more future applications and we hope to add to that with our usage for particle identification.

The goal of this project is to propose a method for particle identification that meets the specified criteria (high accuracy and low latency). The process by which the data is generated and filtered is mentioned, including instructions at the project GitHub repository. I will also present an introduction to neural networks and the mathematics behind how they work, including a brief discussion on why we chose convolutional neural networks and what sets them apart from other deep learning techniques.

A significant part of this project is aimed at minimizing latency and improving memory efficiency, consequently, a discussion about sparse tensors and their advantages is held, with emphasis on why a sparse formulation for our model is preferred.

The work we have done involves a working model for the complete case, involving particle identification between pions, electrons, kaons, and protons in the case that magnetic fields are enabled. The limitations are discussed alongside future improvements to the model both in terms of computational and memory efficiency.

2 The Electron-Ion Collider

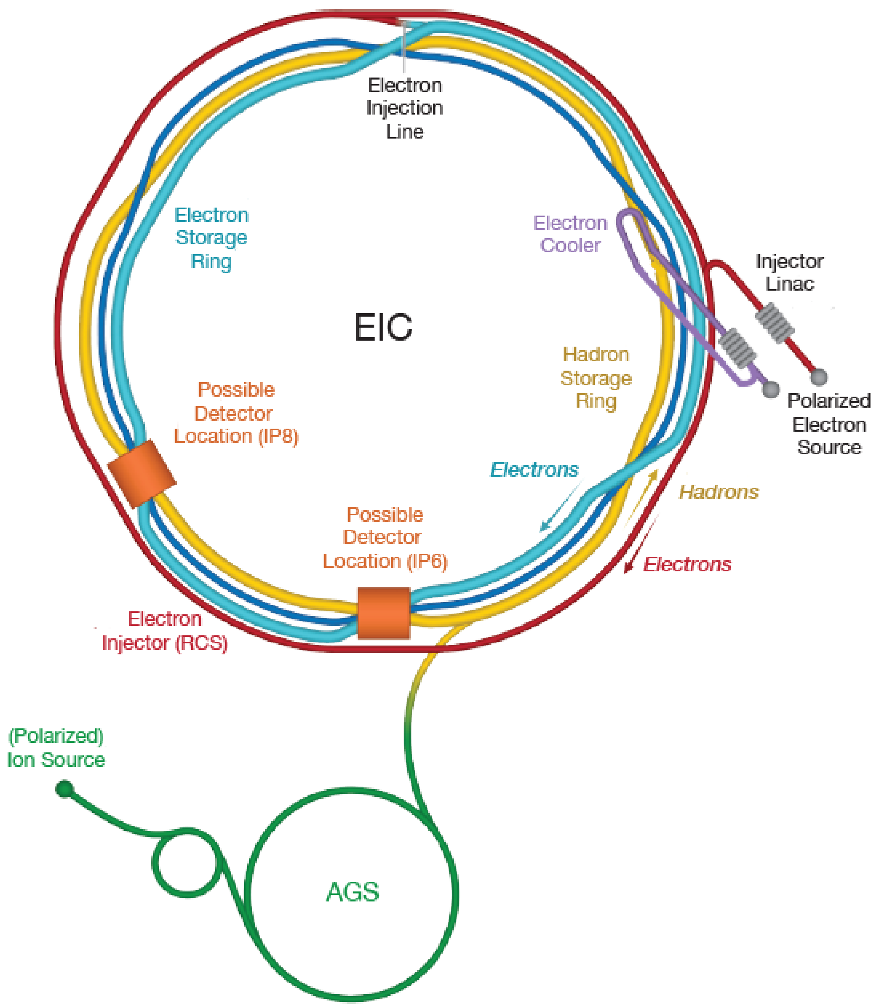


Figure 1: Schematic of the planned EIC facility [3].

The Electron-Ion Collider (EIC) is a proposed particle accelerator by Brookhaven National Laboratory (BNL) where electrons collide with protons and nuclei. This accelerator is a high polarization ($\sim 70\%$ for proton and electron beams) and high luminosity ($\sim 10^{33} - 10^{34}\text{cm}^{-2}\text{s}^{-1}$) accelerator allowing for the study of newer areas of physics as well as far-reaching applications beyond physics. It houses our dRICH detector as part of the EPIC detector. Part of the questions the EIC wishes to address are the origin of the nucleon mass, the nucleon spin, and the emergent properties of a dense system of gluons [2].

2.1 Cherenkov Radiation

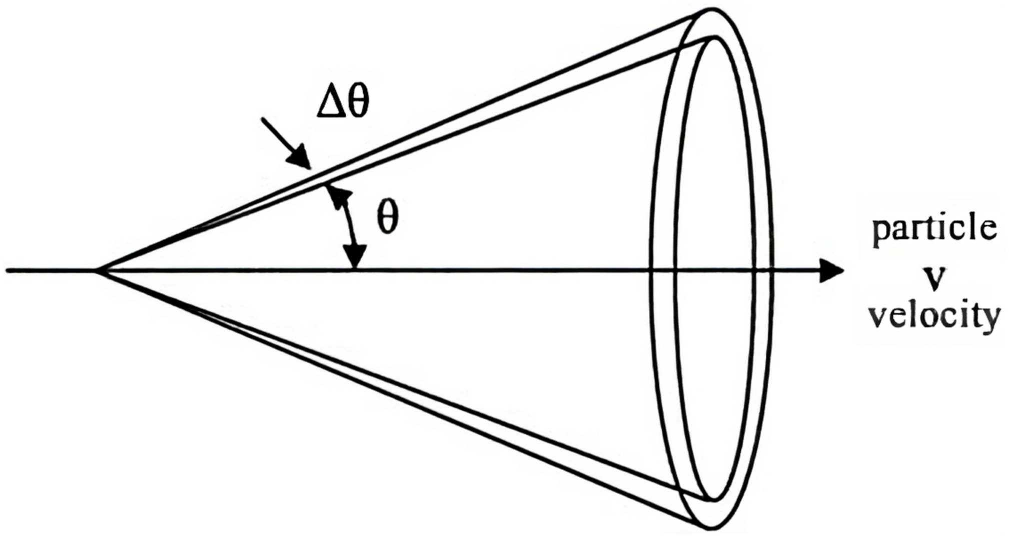


Figure 2: Cherenkov radiation cone produced when a charged particle exceeds the speed of light in a medium [3].

The working principle behind our detector involves Cherenkov radiation. Cherenkov radiation is emitted when charged particles exceed the speed of light in a medium (phase velocity of light). Since Cherenkov radiation typically occurs with high-energy particles, the radiation produced will consequently have higher frequencies, with the majority of the radiation lying on the ultraviolet spectrum. The emission angle is given by

$$\theta = \cos^{-1}\left(\frac{1}{n\beta}\right), \tag{1}$$

where β is defined as the ratio of the speed of the particle to the speed of light,

$$\beta = \frac{v_p}{c}. \tag{2}$$

When the phase velocity of light is exceeded, a cone of light is produced which is used by the standard RICH detectors to determine different particle types. This is because the radii of the cones when collected at a detector vary by particle type due to their mass. A popular example of this phenomenon is the blue glow seen with nuclear reactors underwater.

2.2 dRICH Detector

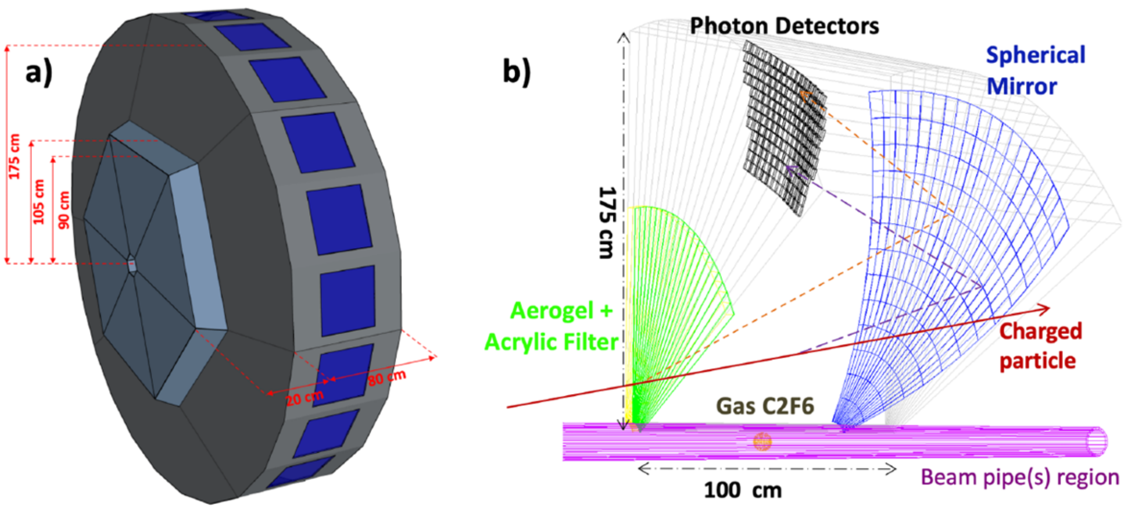


Figure 3: a) The dRICH detector. b) A single sector from the dRICH detector showing the aerogel & C_2F_6 gas radiators [4].

The dual-radiator Ring Imaging Cherenkov Detector shown in Figure 3 is a particle identification (PID) detector with powerful hadron identification properties (π/K/p separation better than 3σ apart) from $\sim$ 3 GeV/c to $\sim$ 60 GeV/c in the ion-side end cap of the EIC detector. It also provides a remarkable electron and positron identification (e/p separation) from a few hundred MeV up to about 15 GeV/c [2]. The detector has 6 identical sectors with aerogel and gas radiators in each sector. These radiators produce one ring each with Cherenkov radiation, this radiation is then reflected onto the photon detectors by a mirror, producing the characteristic rings we see from a RICH detector.

The radii of the rings produced vary by several factors, each particle produces a different ring size when given the same preset energy, this is self-evident since particles with larger masses will travel with smaller velocities at the same energy. This principle allows us to determine the particle type by the ring sizes.

There are two rings formed per particle since this is a dual-radiator RICH detector. The radii of the rings from the gas radiator are smaller as shown in Figure 4. This figure also shows that the problem of particle identification is more difficult with magnetic fields enabled as the variance in ring radii is greater, this will make the training process more challenging.

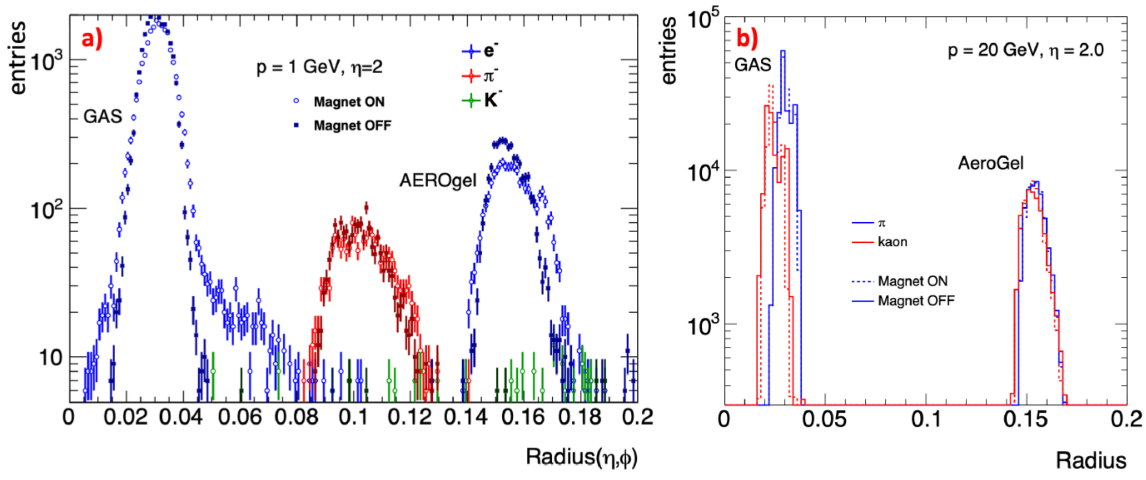


Figure 4: The different particle types and their respective radii. The difference in ring size by radiator (gas or aerogel) and magnetic fields is shown in the plots [4].

3 Data Generation & Analysis

3.1 Generation

The accelerator and dRICH detector are simulated using the DD4hep toolkit by CERN [6]. Data is generated using this toolkit with certain specified parameters, outputting a ROOT file with the events. The parameters we specify and their values are seen in Table 1.

Table 1: The parameters used for generating the data and their respective values.

Parameters	
Particle types	pi+, e-, kaon+, proton
Particle energy	5 GeV
Range of θ angles for gun	3° to 50°
Range of ϕ angles for gun	50° to 85°
Number of events	10000

Where the gun distribution function is a cos(θ) function. The ϕ angle controls the sectors included in our particle generation. It is not strictly accurate as particles bleed into sectors which are excluded. After generation of data, we send the ROOT files to the next stage of data filtration.

3.2 Analysis & Cleaning

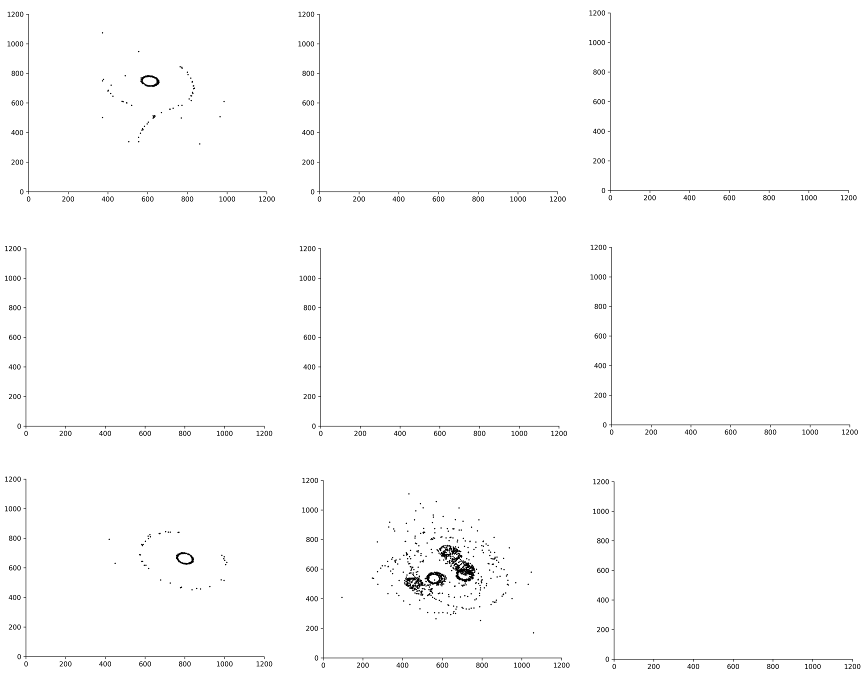


Figure 5: A sample output of 9 electron events from the generated data.

The generated data contains a lot of empty events and events with insufficient data, which can be seen in Figure 7. As such, we need to clean the dataset to produce a newer dataset mostly comprised of rings from which the model can learn. The criteria for removal are all empty events, events below 50 points, and events above 500 points. Empty events do not have a contribution, events with under 50 points are unlikely to yield a complete ring, and events over 500 points would be very messy events which will throw off the model when training.

A sample of 9 events, after we filter based on the specified criteria is shown in Figure 6. There are mostly rings and while not perfect, they're a far better dataset compared to the unfiltered dataset. This process is quite costly however, $\approx 60\%$ of the events are empty thereby eliminating the majority of the events, and after the other criteria are applied, only a fraction of the data remains. The number of points per event will also be affected once the quantum efficiency of the detectors (30%), dead areas between pixels (88%), and safety factor (70%) are taken into account.

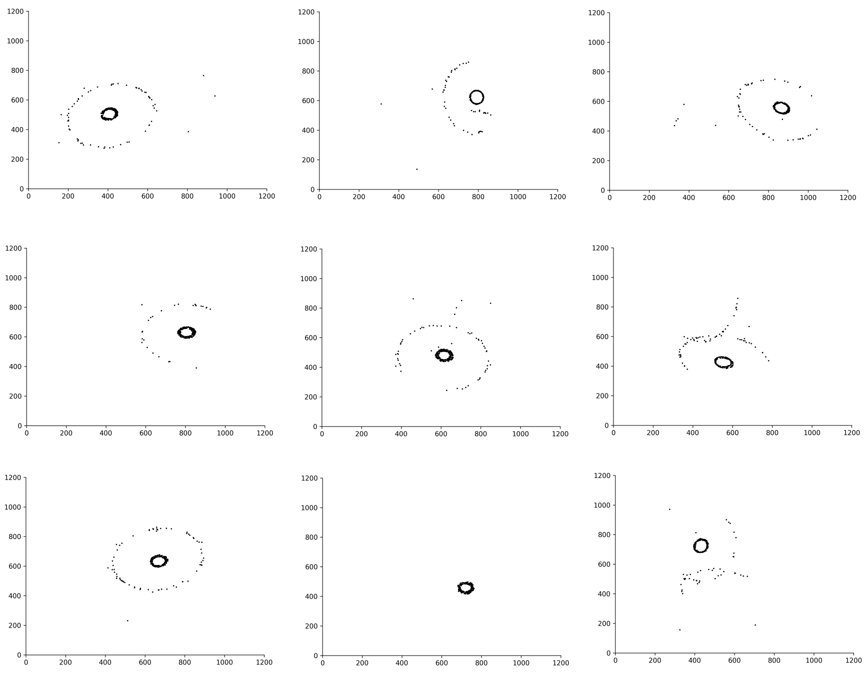


Figure 6: A sample output of 9 electron events after the filtration process.

4 Sparse Tensors

Following the evident memory cost of using a dense formulation of our data, a new formulation that is both computationally and memory efficient is required. This leads to the introduction of the sparse formulation for data storage. This formulation relies on the criteria that most of the data given is 0. Since the majority of our detector pixels are unactivated (0) with only a few hits. This formulation works so well as it addresses the memory cost by dropping all the data points which are 0, which is a trivial operation. In our case, for a matrix of 1200x1200, we will have no more than 500 data points (from our filtration criteria) per event. This means our matrix density is at most 0.03%, which solidly meets the criteria. To calculate the memory usage for both cases, we assume a matrix size of 1200x1200 and a data type of float (4 bytes), this means the dense matrix uses $1200 * 1200 * 4 = 5760000$ bytes, whereas the sparse matrix will use $(2 * 8 + 4) * 500 = 10000$ bytes, there is at minimum a 576 fold memory savings [7]. This shows that a sparse formulation is the best way forward.

In the case of dense tensors, data is passed through a pooling layer (more on this later) for memory saving, reducing our image dimensions from 1200x1200 to 100x100.

4.1 The Sparse COO Format

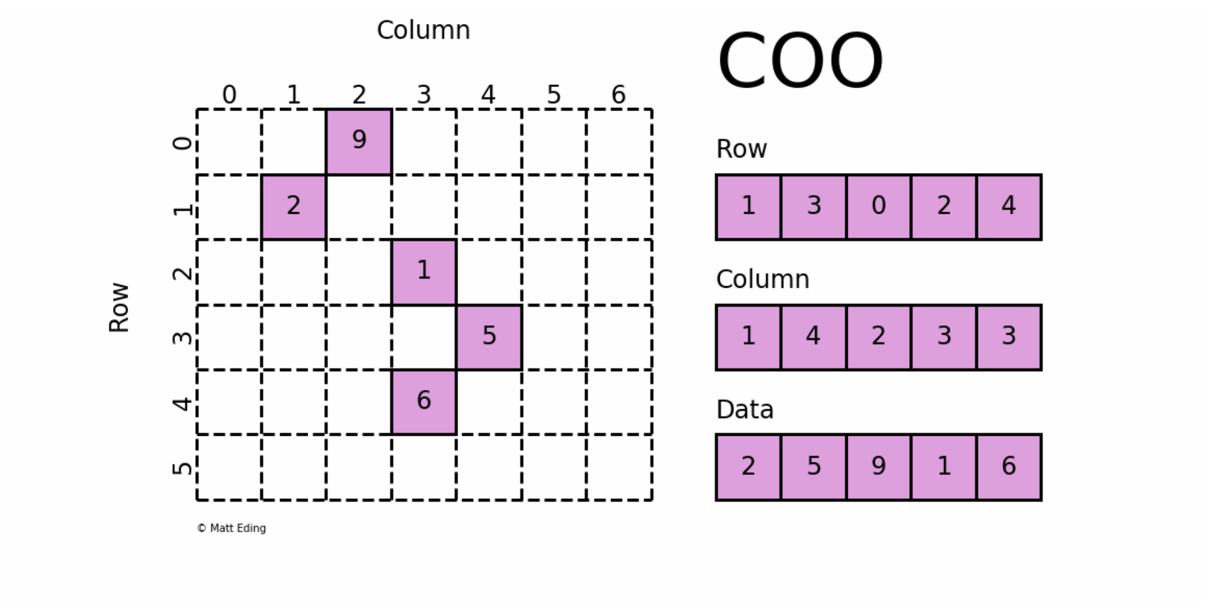


Figure 7: How data is stored in a sparse COO format [8].

There are a number of different formats for storing sparse data, most commonly used are the CSR/CSC and the COO formats. CSR has the computational edge while the COO format is the easiest to use [8]. Minkowski Engine utilizes the COO format so this will be our focus. The COO format works by storing the values at a certain point alongside its row and column coordinates, this is shown in Figure 7. The benefits of this have been discussed, it is more memory efficient allowing us to keep only the non zero values.

4.2 Minkowski Engine

The need for an external library arises from the lack of native sparse convolutions in the popular machine learning frameworks (PyTorch, TensorFlow, Keras). There are a number of frameworks that extend support for convolutions with sparse input. SparseConvNet by Meta, Minkowski Engine by Nvidia, and O-CNN by Microsoft are discussed in this paper. SparseConvNet was the first library we attempted to use, however, we quickly dropped it due to the lack of documentation. Minkowski Engine is the library we used for the sparse formulation. It is well-documented and straightforward to work with. This is a voxel-based approach that uses hash tables for indexing [10], whereas O-CNN uses an octree for indexing. Using octree's for indexing allows for several advantages over Minkowski Engine which are discussed later [11].

5 Convolutional Neural Networks

Convolutional neural networks are remarkably powerful when it comes to image recognition, they are highly suited for this project which relies on differentiating particles based on ring size. A notable example of their power at image recognition can be seen with the MNIST dataset [12] where the state of the art models perform with an error rate of 0.09% [13].

I will first introduce neural networks to give a brief idea of how they operate, after which I will introduce the types of layers that define a convolutional neural network (CNN). We use the Python interface for PyTorch as our framework [14], and to prevent linearity we use a rectified linear unit activation function (ReLU) between the layers:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{3}$$

Where x is a neuron. A neuron is an individual data point with a certain value known as the activation. An epoch is a full circulation around the entire dataset, a batch is a sample of the dataset. We go through an epoch by taking consecutive batches until the dataset is complete, the data is then shuffled and the next epoch is started until we complete the specified number of epochs.

There are 3 splits for the dataset: training (80%), validation (10%), and test dataset (10%). The training and test datasets are self-explanatory, while the validation dataset is used to ensure overfitting isn't occurring (discussed in section 5.3). Information about the loss per epoch and the test accuracy is collected for the analysis.

5.1 Linear Layer

The fully connected layer is a linear transformation, taking as input a flattened array and outputting the next layer after applying the weights and biases. The weights are a trained parameter that specifies how strong the connections between the neurons in one layer are to the next. The biases are another trained parameter that provides room for inactivity until the activation is greater than the bias.

$$a^l = xa^{l-1} + b, \tag{4}$$

where a^l is the flattened vector of activations of the neurons at layer l, x is the matrix of weights, and b is the vector of biases.

$$\begin{bmatrix} a_0^1 \\ a_1^1 \\ \vdots \\ a_n^1 \end{bmatrix} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,n} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,n} \end{bmatrix} \begin{bmatrix} a_0^0 \\ a_1^0 \\ \vdots \\ a_n^0 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}. \tag{5}$$

5.2 Loss & Gradient Descent

The weights and biases are trained through gradient descent during backpropagation. First, we define a loss function,

$$L(x) = (t-x)^2. \tag{6}$$

This is the quadratic loss function where t is the target and x is the activation of the neuron. The average of the loss function across all the neurons is taken to be our loss for a certain batch/epoch. The loss measures how far off our activation is from the target. This loss function is the simplest case whereas for our model, we use a cross-entropy loss function,

$$L(x,y) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} w_c log \frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})} y_{n,c}, \tag{7}$$

for x input, y target, N batch size, and C classes. More information on this loss function can be found in the PyTorch documentation [15].

The process of backpropagation is the optimization of the weights and biases by minimizing the loss function using an optimizer. This optimization can occur at different points during the training, most commonly at every epoch such as gradient descent or every batch such as stochastic gradient descent. Gradient descent is an iterative method that is used for minimizing the loss function by finding the local minima of the function, it can be seen in Figure 9.

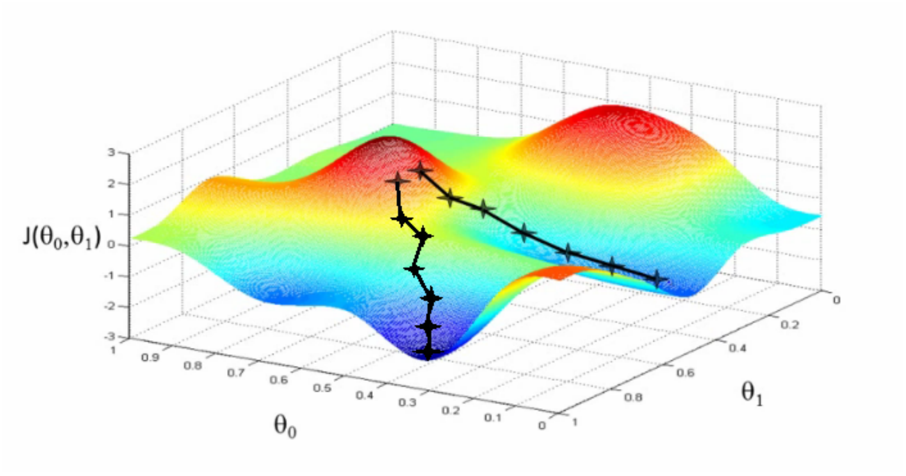


Figure 8: Visualization of a stochastic gradient descent [16], the black line is the path downwards to the minima through different iterations. The two θs represent the weights and biases. $J(\theta)$ is the loss function.

A gradient descent optimizer is not used here, we use a stochastic gradient descent (SGD) method instead, which is defined by

$$w = w - \eta \nabla L_i(w), \tag{8}$$

η is the learning rate of the algorithm, specifying the step size to the local minima. $L_i(w)$ is the loss function for the current batch which is used to approximate the true gradient, and w are the weights and biases which are increased/decreased. We don't specifically use this stochastic gradient descent, but rather an extension of it. The optimization algorithm used in this project is the Adam optimizer [17]. This process is repeated at every sample allowing us to inch closer to the local minima of the graph.

5.3 Dropout & Overfitting

An issue that often occurs during the training process is when the model over-fits, this is when the model starts learning the training dataset too well. This issue causes the model to only perform well on the training dataset, failing when presented with new data. The conventional way to test for overfitting is by running a validation dataset side by side with the training dataset, and measuring the loss for each. If the model is overfitting, we expect the validation loss to lag

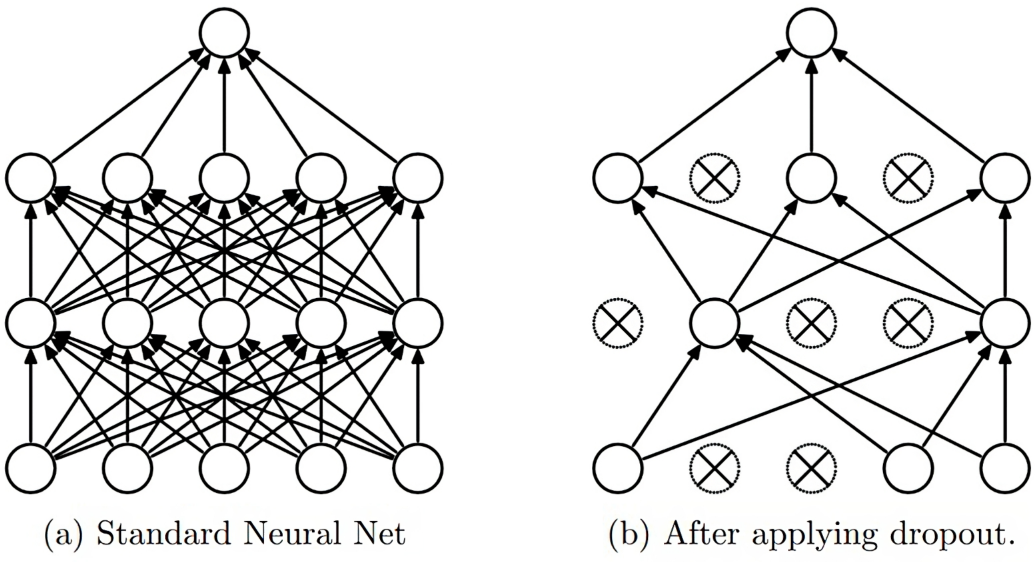


Figure 9: a) The standard neural network with all neurons online b) The neural network after applying dropout, with a number of neurons offline [18]

behind the training loss or even stop decreasing entirely.

A solution to this is the introduction of dropout layers to our model. Dropout layers are a regularization technique commonly used to address overfitting, the way dropout layers work is by deactivating some neurons between the hidden layers. This approach is highly successful in addressing overfitting. Typically dropout layers are controlled by a dropout percentage, however I am not able to specify the dropout percentage as the function used in Minkowski Engine lacks this option.

5.4 Convolution & Pooling Layers

A CNN is different from other deep learning techniques due to two types of layers, the convention used here for a layer in a CNN is defined as a convolution layer followed by a pooling layer, which is considered as one layer.

5.4.1 Convolution

The convolution layer is defined by the convolution operator, which is the backbone of a CNN model. The main idea behind this layer is to provide a method of feature extraction from an image using filters/kernels. These filters are multiplied element-wise by a block of the image to produce an output neuron, a sliding window of this filter produces all the output neurons for the next layer. Padding increases the dimension by adding new elements of value 0 at the border of the matrix, stride specifies how fast the sliding window moves across the neurons. There are two types of convolutions, same and valid convolutions. Same provides an output image of the same size as the input, while valid decreases the image size. The equation determining the output size from the input size for padding size p, stride size s, and kernel size k is given by

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1. \tag{9}$$

The convolution function we use for an input of (N, C_{in}, h, w) and an output of $(N, C_{out}, h_{out}, w_{out})$ is given by

$$out(N_i, C_{out}) = bias(C_{out}) + \sum_{k=0}^{C_{in}-1} weight(C_{out}, k) \star input(N_i, k), \tag{10}$$

where $\star$ is the cross-correlation operator. C_{out} specifies the number of filters used. These filters are not hand-picked but rather trained by the model [19].

5.4.2 Pooling

Pooling is a layer used for dimension reduction while retaining the important features of the image. The equation describing the average pooling method we used is given by

$$out(N_i, C_j, h, w) = \frac{1}{k^2} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} input(N_i, C_j, s \cdot h + m, s \cdot w + n). \tag{11}$$

This assumes a square filter and stride. With C channels, h height, and w width. k is the filter size while s is the stride size. The output layer is reduced by a factor determined by k and s [20].

5.5 Hyperparameter Optimization

Hyperparameters are the parameters we specify to control the training process of the model, these are different from the parameters trained by the model and they do not change during the training process. We seek to optimize the hyperparameters to provide the minimum loss, of particular importance is the learning rate, number of epochs, and quantization size (specific to Minkowski Engine).

Manually optimizing hyperparameters is a tedious process which is unlikely to lead to the optimal hyperparameters. Hyperparameter optimization methods offer an automated and methodical way by which we can find the best hyperparameters to minimize our loss. As such we use a technique known as Bayesian hyperparameter optimization. This method uses prior results to better predict the future hyperparameters. We use the open source scikit-optimize for this process [9].

6 Results

6.1 The Dense Formulation

6.1.1 Model

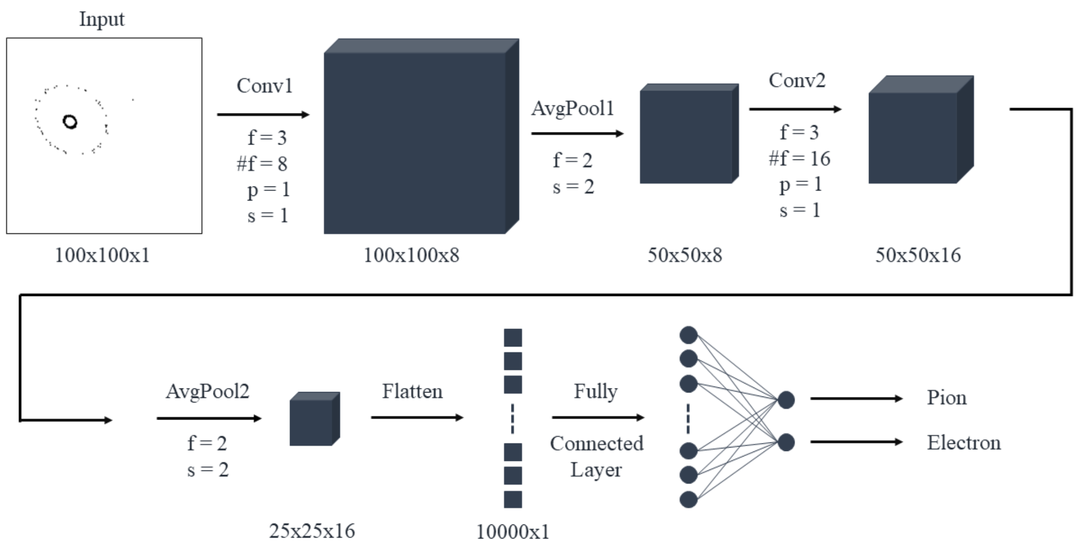


Figure 10: Visualization of the model, including the different layers.

The model consists of two layers where we use the convention of one layer to include a convolution layer and a pooling layer. The n-dimensional matrices are flattened into a vector where they will be passed onto a linear layer to be classified as either a pion or an electron. There are

ReLU activation functions used between the layers in the model. The hyperparameters used are shown in Table 2:

Table 2: The hyperparameters used and their respective values (dense case).

Hyperparameters	
batch size	64
epochs	100
learning rate (η)	0.001

6.1.2 Performance

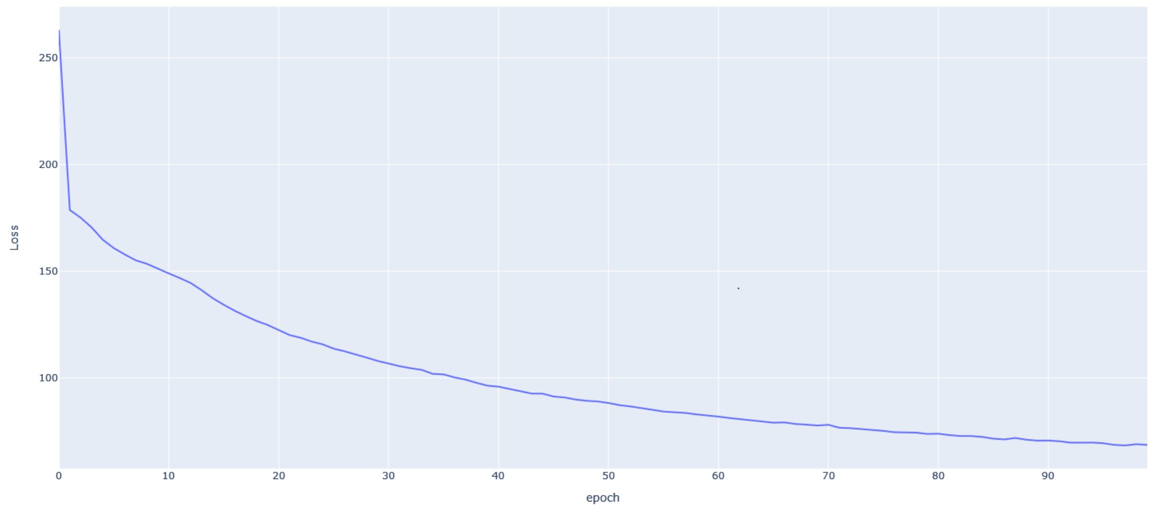


Figure 11: Plots of the loss by epoch for 100 epochs, as expected the longer we train the data the lower the loss and consequently, the accuracy.

The loss plot per epoch number is shown in Figure 11. The accuracy is defined as the ratio of the number of successful classifications with the total number of attempts, and it is $\approx 97\%$ for our model. The accuracy increased by 12.95%, while the loss decreased by 73.9%. There is an exponential decrease for the loss, this is expected as towards the end of training, the improvements are not as significant as at the start of the training process. Ultimately, this model performs quite well, however, we will not spend much more time on this method as it is an inefficient method, despite it's good performance and high accuracy.

6.2 The Sparse Formulation

6.2.1 Model

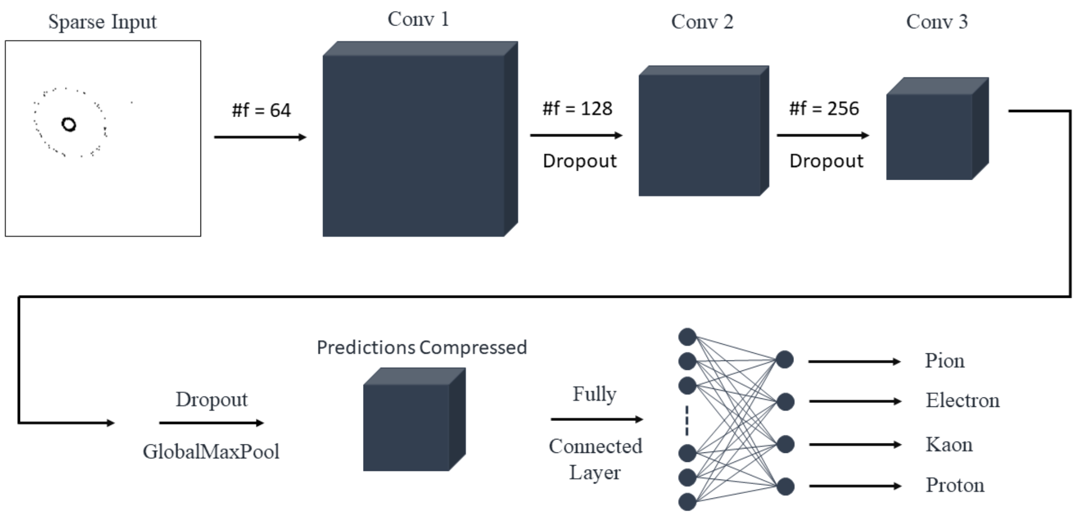


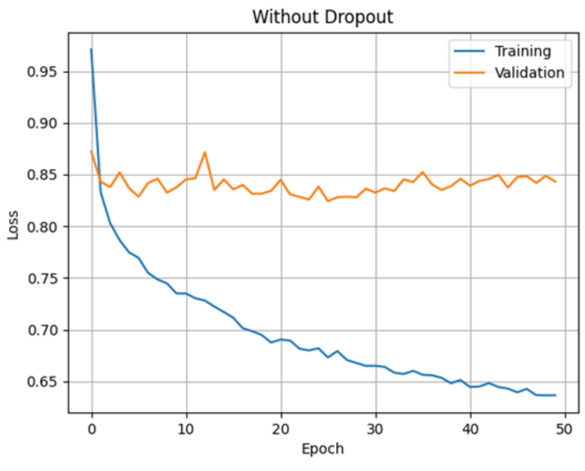
Figure 12: Visualization of the model, including the different layers.

The sparse formulation is the main focus of this paper so we will dedicate more time to discussing its features and performance. We have three convolution layers with both ReLU and dropout layers between them. This is then passed onto a Minkowski Engine function, GlobalMaxPooling, that reduces the prediction data from (batch size * event size) to the (batch size). This reduction of the prediction size allows us to match it to the size of the targets, which allows us to carry out the loss calculations. Otherwise, a size mismatch would prevent our model from working, more information on the size of the Minkowski Engine sparse tensor can be found in the documentation [10].

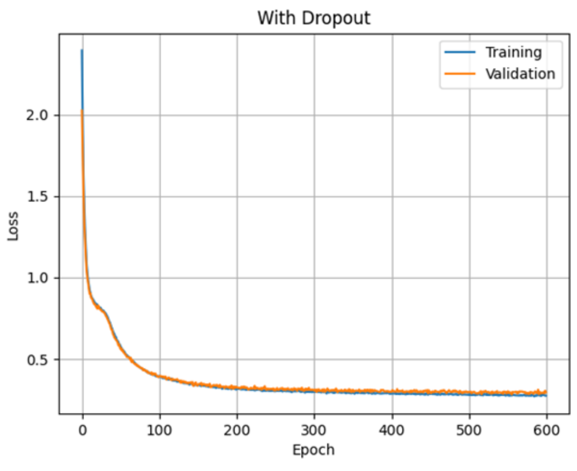
Table 3: The hyperparameters used and their respective values (sparse case).

Hyperparameters	
batch size	128
epochs	600
learning rate (η)	1e-4
quantization size	5e-6

6.2.2 Performance



a)



b)

Figure 13: a) Loss by epoch without dropout layers b) Loss by epoch with dropout layers

We first investigate the effect that dropout layers had on potential overfitting. Figure 13 showcases the validation and training loss with and without the dropout layers. As we would expect, overfitting is observed without dropout layers, this can be seen as the validation loss is stagnant while the training loss is decreasing drastically. Once dropout layers are added, however, this issue is addressed as the validation and training loss decrease at a roughly equal rate.

We now carry out a comparison of the average accuracies of all particles, in the cases of disabled and enabled magnetic fields, respectively. Table 4 displays the average accuracies by particle type and status of the magnetic fields.

Table 4: Accuracies per particle type when the magnetic field is disabled and enabled. The uncertainty is given by the sample standard deviation, averaged over 10 samples.

	Accuracy (%)	
Particle	Mag Field Disabled	Mag Field Enabled
Pion	97.39 ± 0.52	94.71 ± 1.17
Electron	98.33 ± 0.54	97.68 ± 1.03
Kaon	87.86 ± 2.49	81.41 ± 2.06
Proton	10.00 ± 31.62	6.19 ± 6.27

The magnetic field adds noise to the system, and as expected, this decreases the accuracy by quite a bit, particularly for the kaon and proton. Consistently, our model was able to detect electrons better than any other particle. On the opposite end of the spectrum, protons performed unfavorably with significant fluctuations (standard deviation is larger than the value itself).

6.3 Discussion & Limitations

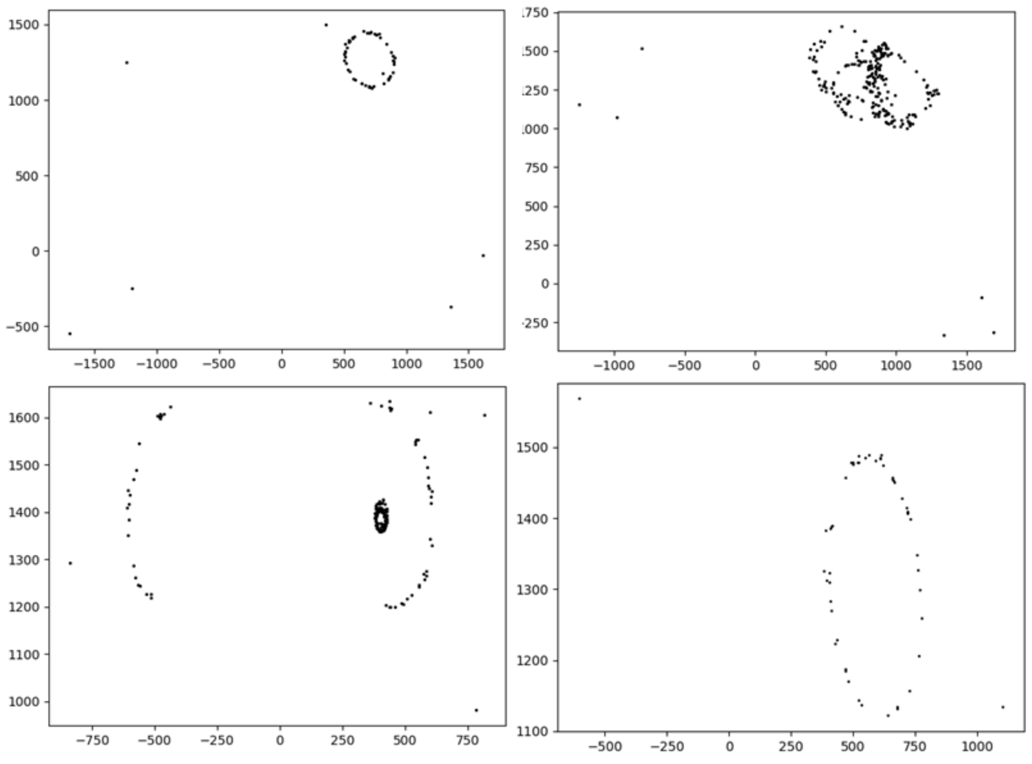


Figure 14: Sample of filtered kaon events.

The limitations mostly come from the data cleaning and analysis, there is a lot of room for improvement when it comes to filtration. The model doesn't perform as poorly on kaons as it does the protons, however, there is still a 20% error rate that needs addressing. To better understand what goes wrong, we can sample a few events which are shown in Figure 14. There are visible rings which are the ones the model can both train on and identify. Despite this, some events are too noisy, while others are rings that seem to be shifted apart. Both of these cases weaken the predictive ability of the model during the testing phase. These events are what cause the 20% failure rate. Part of this noise can be attributed to the magnetic field, which can be seen in Table 4.

In the case of protons, the accuracy is far worse. Figure 15 shows a few proton events which help to explain the incredibly low accuracy for protons. The filtered data is simply unusable

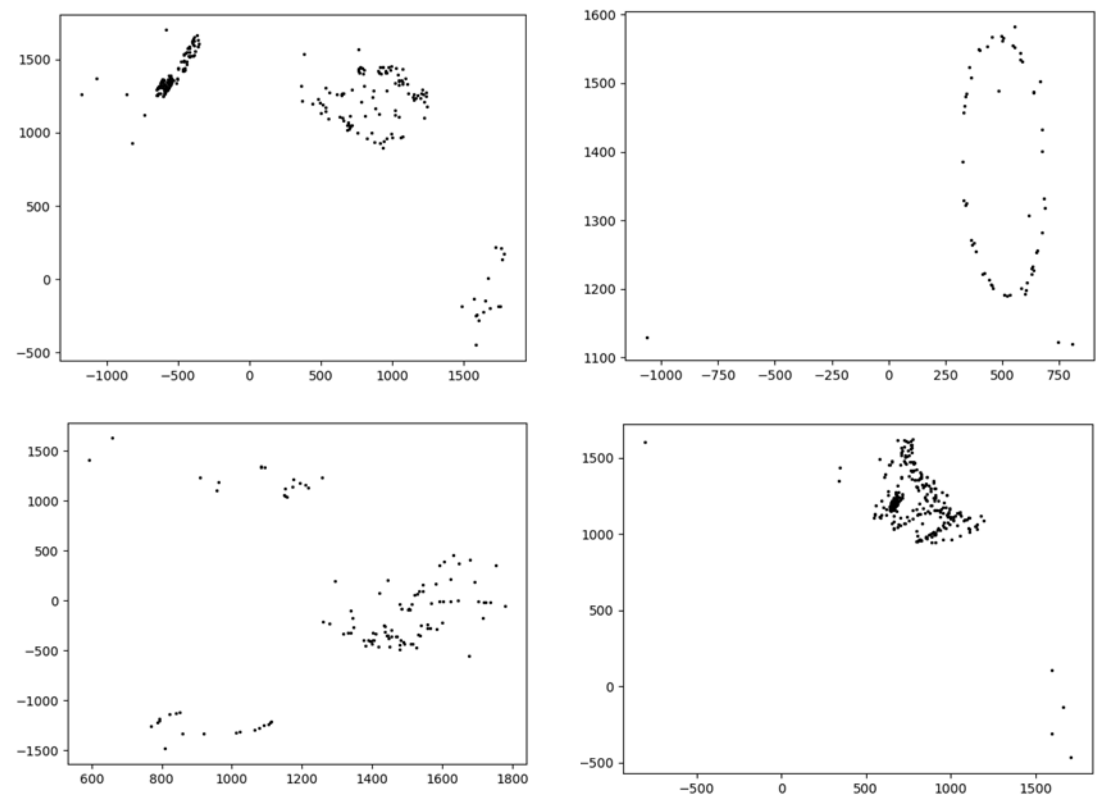


Figure 15: Sample of filtered proton events.

with no discernable pattern for the majority of the events. This problem is compounded by the lower number of events after filtration. The results for kaons and protons suggest that perhaps a separate filtration method should be used for different particles, which would provide the model with the best training data possible.

Another big limitation is that we have a small dataset, more data will give our model a better training process. This issue may be difficult to address as adding a larger number of events (with magnetic fields enabled) causes the program to crash, possibly due to memory issues in Python. This issue was present on my local machine and Narval (Compute Canada).

7 Conclusion & Future Work

The goal of this project was to apply deep learning techniques to the problem of particle identification. We have successfully demonstrated the power of CNN's when identifying either pions or electrons under the presence of a magnetic field, $(94.71 \pm 1.17)\%$ and $(97.68 \pm 1.03)\%$ for pions and electrons respectively. Kaons and particularly protons require better filtration, after which we can .

We would like to continue working on improving our data filtration process, filtering differently by particle type if necessary. Another area which requires some revision is the choice of Minkowski Engine for sparse convolutions, for improved memory and computation efficiency we would ideally switch to O-CNN, it is far more efficient than the full voxel libraries (Minkowski Engine included) as demonstrated in Figures 16 & 17.

Finally, to further improve computational and memory efficiency, it would be best to move on from Python to a lower level language like C++ that has a PyTorch interface, this would likely address the memory issue faced when filtering large event numbers.

8 Appendix

The entire code for the project can be found on my GitHub repository, including instructions for data generation: https://github.com/S-tuberosum/Particle-Identication-Using-CNNs

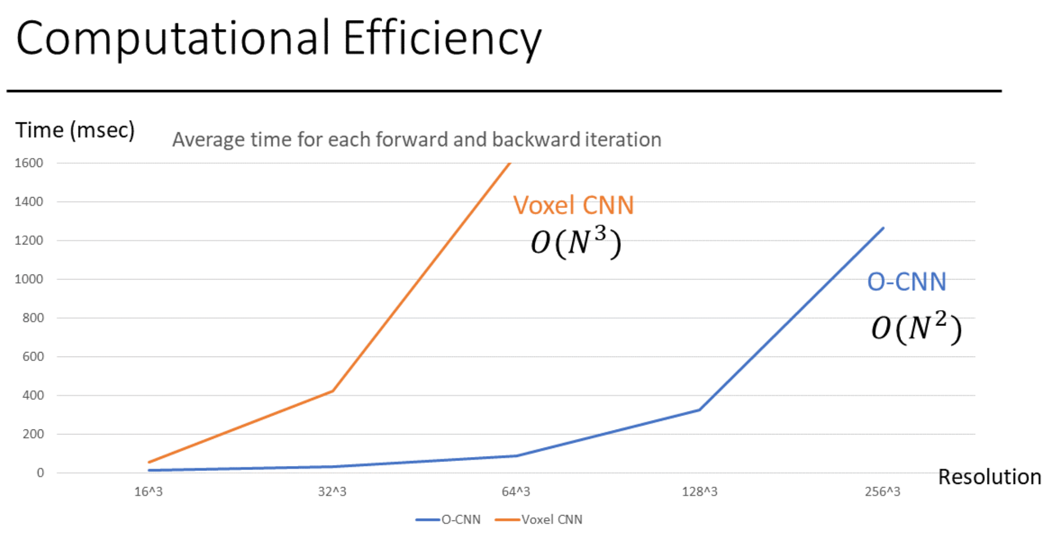


Figure 16: The computational efficiency of an octree based model vs the full voxel model [21].

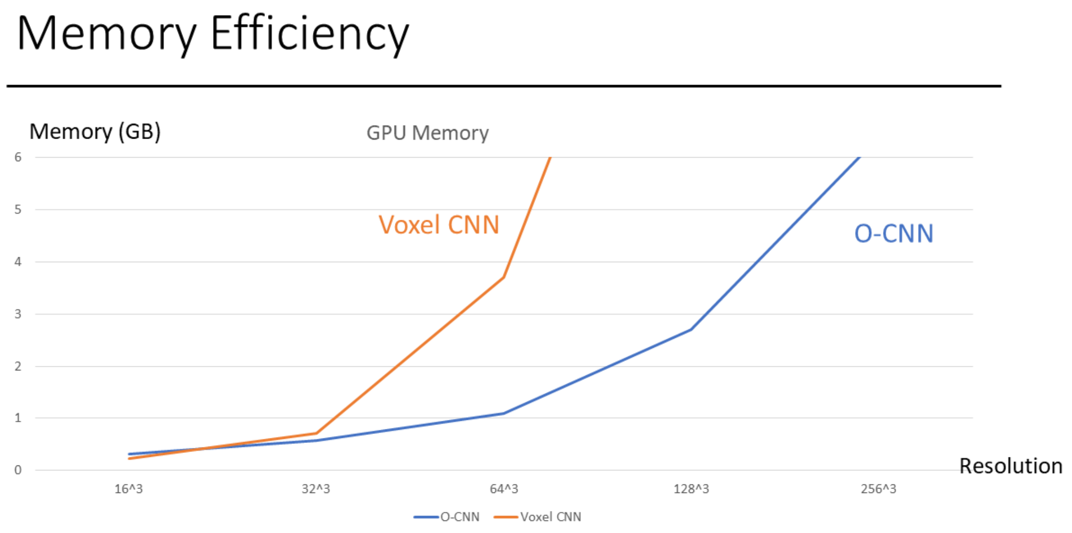


Figure 17: The memory efficiency of an octree based model vs the full voxel model [21].

References

[1] Cisbani, E. et al. (2020). *AI-optimized detector design for the future Electron-Ion Collider: the dual-radiator RICH case*. JINST 15 P05009. doi:10.1088/1748-0221/15/05/P05009

[2] Adam, Jaroslav. et al. (2021). *EIC Yellow Report*. United States. doi:10.2172/1764596

[3] Obodovskiy, I. (2019). *Radiation: Fundamentals, applications, risks, and safety*. Elsevier.

[4] Bock, F. et al. (2021). *ECCE Particle Identification*. Internal report.

[5] Sauli, F. (2023). Cherenkov ring imaging. In *Gaseous Radiation Detectors: Fundamentals and Applications* (Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology, pp. 399-429). Cambridge: Cambridge University Press. doi:10.1017/9781009291200.016

[6] Frank, M. et al. (2014). *DD4hep: A Detector Description Toolkit for High Energy Physics Experiments*. J. Phys.: Conf. Ser. 513 022010. doi:10.1088/1742-6596/513/2/022010

[7] PyTorch Foundation. (n.d.). *torch.sparse - PyTorch 2.0 documentation*. Retrieved April 18, 2023, from `https://pytorch.org/docs/stable/sparse.html#sparse-coo-docs`

[8] Eding, M. (2019, April 25). *Sparse matrices*. Sparse Matrices · Matt Eding. Retrieved April 7, 2023, from `https://matteding.github.io/2019/04/25/sparse-matrices/#coordinate-matrix`

[9] scikit-optimize. (n.d.). *Sequential model-based optimization in Python*. Retrieved April 27, 2023, from https://scikit-optimize.github.io/stable/

[10] Choy, C. et al. (2019). *4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 2019 pp. 3070-3079. `https://doi.org/10.48550/arXiv.1904.08755`

[11] Wang, P. et al. (2017). *O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis*. ACM Transactions on Graphics, Vol. 36, No. 4, Article 72. https://wang-ps.github.io/O-CNN_files/CNN3D.pdf

[12] Deng, L. (2012). *The mnist database of handwritten digit images for machine learning research*. IEEE Signal Processing Magazine, 29(6), 141–142. doi:10.1109/MSP.2012.2211477.

[13] An, S. et al. (2020). *An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition*. ArXiv, doi:10.48550/ARXIV.2008.10400

[14] Paszke, A. et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[15] PyTorch Foundation. (n.d.). *CrossEntropyLoss - PyTorch 1.13 documentation*. Retrieved December 19, 2022, from https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[16] Ng, A. (n.d.). *Neural Networks and Deep Learning: Gradient descent*. DeepLearning.AI. Retrieved December 18, 2022, from https://www.coursera.org/lecture/neural-networks-deep-learning/gradient-descent-A0tBd

[17] Kingma, Diederik P. et al. (2014). *Adam: A Method for Stochastic Optimization*. ArXiv, doi:10.48550/ARXIV.1412.6980

[18] Srivastava, N. et al. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. from https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[19] PyTorch Foundation. (n.d.). *Conv2d - PyTorch 1.13 documentation*. Retrieved December 19, 2022, from https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[20] PyTorch Foundation. (n.d.). *AvgPool2d - PyTorch 1.13 documentation*. Retrieved December 19, 2022, from `https://pytorch.org/docs/stable/generated/torch.nn.AvgPool2d.html`

[21] Wang, P. (2017). *O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis*. [PowerPoint slides]. `https://onedrive.live.com/view.aspx?resid=97002CD884825EBF!2092&ithint=file%2cpptx&authkey=!ADgc5oQjg_dQbaQ`