Trigger-Level Multiple Electron Event Classification with LDMX using Artificial Neural Networks

Jacob Lindahl

Department of Physics, Division of Particle Physics, Lund University

May 11 2023

Master thesis supervised by Lene Kristian Bryngemark and Ruth Pöttgen

Abstract

Artificial neural networks is a powerful tool for classifying and identifying patterns in large amounts of data. One of the possible tasks of these networks is classification of data into categories. LDMX is a fixed target experiment that is designed to search for light dark matter particles using missing energy and momentum of electrons passing through and scattering of a target. In the current design the number of electrons passing through is counted using the trigger scintillator system.

In this thesis, four different types of artificial neural networks were trained to count the number of electrons present in events using simulated data from both the electromagnetic calorimeter and trigger scinitllator to explore the viability of using neural networks at LDMX. These were convolutional neural networks, recurrent neural networks, graph neural networks, and a combined convolutional and recurrent neural network. Three datasets were generated: one using only data from the electromagnetic calorimeter, one that combined the electromagnetic calorimeter and the trigger scinitllator, and one that combined the electromagnetic calorimeter and a modified design for the trigger scinitllator data. To be viable the accuracy of the models needed to be equal to or exceed 0.99999. None of the models were able to exceed or match this accuracy with the highest accuracy reached by a convolutional and recurrent model training with the electromagnetic calorimeter and a modified design for the trigger scinitllator data of 0.962 and 0.9575. The graph neural network was not able to be trained with data in time and neither was all of the combined convolutional and recurrent neural networks. The different benefits and drawbacks of the models were then evaluated and compared to each other.

Popular Science Description

Dark Matter remains an elusive adversary for physicists. Few topics in physics have inspired more theories and debate while simultaneously remaining inscrutable, dodging our collective stabs at an effective explanation. Due to Dark Matter not interacting with light or other forces such as the strong nuclear force that binds protons and neutrons together, the only real way of confirming its effect on our universe is through its gravitational effects. The Standard Model of physics that describes all the matter made up of atoms and other particles only describes 5 % of our universe with 25 % of all the matter and energy made out of Dark Matter and the rest made out of some "Dark Energy". This means that a significant portion of our universe is only accessible indirectly meaning the true nature of our universe is obscured to us.

Many theories that go beyond the Standard Model of physics predict the existence of a type of particles called Weakly Interacting Massive Particles, or WIMPs, that have very small interactions with regular matter, if any, where weak refers to the weak nuclear force responsible for nuclear decay. These particles can have masses ranging from 100 TeV to below 1 GeV corresponding to roughly 100000 times the mass of a proton to below the proton mass, respectively. Numerous experiments exist to try to discover these particles. One type of theory that can involve the existence of WIMPs or possibly other types of particles with other interaction types is the Dark Sector. In this hypothesis there can exist entirely different types of forces and particles that cannot interact with us except through some mediator particle or particles. One version of the Dark Sector hypothesis treats one of these sectors analogously to electromagnetism and can interact with regular charged matter through a "dark photon" mediator particle that can then decay to Dark Matter. This would then allow for the existence of Dark Matter particles below 1 GeV in mass a region referred to as "Light Dark Matter". It is in this region that the Light Dark Matter eXperiment is designed to search for and possibly (finally) find Dark Matter.

LDMX is a proposed fixed-target missing momentum experiment. The experiment involves measuring if Dark Matter particles have been produced in the wake of an electron beam being fired at a tungsten target. Dark Matter can then be generated when the electron interacts with the target and emits a dark photon. When measuring the total amount of energy and momentum of the electron there will then be a certain amount of missing energy and momentum that can then be used to determine the properties of the mediator particle and the resultant Dark Matter. Dealing with the large amounts of events in the detector requires the use of *triggers*. A trigger is a real-time decision that determines whether an observed event should be saved to be analysed for a possible "signal" event involving the emission of a dark photon later. With a beam rate or incoming event rate of 37.2 MHZ meaning approximately 37 million events per second only 5 KHZ or 5 thousand events will be saved to avoid filling the experiment with data that is not relevant or unlikely to contain a signal event. For single electron events where only one electron is involved triggers have been designed that can veto, or remove, non-relevant events. However, for multiple electron events these vetoes are harder to design and often require more processing than for a single electron event and the possibility of accidentally vetoing a multiple electron signal event remains. Being able to distinguish between different types of events as involving one versus two or more electrons is then something that is of vital importance for ensuring that the proper analysis can be performed and no Dark Matter signal indicators are missed.

Machine Learning has steadily become a reliable tool for doing analysis on large sets of data such as for the identification of objects in images or finding relationships between different data points. With products such as ChatGPT demonstrating the power of large language models, Machine Learning continues to be an active area of research and development in all aspects of society. This thesis aims to see how Machine Learning in the form of Artificial Neural Networks can be trained on the data from the Electromagnetic Calorimeter and the Trigger Scintillator that register the energy of charged particles moving through it and positional data on the beam of incoming electrons. Artificial Neural Networks is a type of Machine Learning structure that relies on a set of nodes arranged in layers that are connected. These connections are controlled with a set of weights and these are continuously updated while training. In this case it would be trained using simulated data on events with up to four electrons in order to determine how an Artificial Neural Network can be used when designing vetos. As numerous types of architecture for these networks exist, this thesis restricted itself to four types: Convolutional Neural Networks, Recurrent Neural Networks, Graph Neural Networks, and a Convolutional and Recurrent Neural Network. Combining Machine Learning with the LDMX could then help end our quest for the true nature of Dark Matter and finally lay our nemesis to rest.

Contents

List of acronyms

ANN Artificial Neural Network

CNN Convolutional Neural Network

DM Dark Matter

Ecal Electromagnetic Calorimeter

GNN Graph Neural Network

GRU Gated Recurrent

LDMX Light Dark Matter eXperiment

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

WIMPs Weakly Interacting Massive Particles

TNR True Negative Rate

TPR True Positive Rate

List of Figures

Acknowledgments

I would first of all like to express my appreciation and admiration to my supervisors Lene Kristian Bryngemark and Ruth Pöttgen for their endless patience and guidance during this project. I would also like to thank my family for always standing by me and my friends for listening to me ramble on about physics without end.

1 Introduction

Dark Matter has a long and storied history as the elusive boogeyman of physics. That there is more matter in the universe than we can observe has been proposed as solutions for astronomical observations since the 1600s, but one of the first estimates of the amount of "dark matter" or more appropriately unseen matter was presented by Lord Kelvin in 1884 using stellar velocity to conclude [1]. The first actual observation to suggest that there has to be more matter in the universe than can be observed through electromagnetism was Knut Lundmark in 1930 [2] with Jan Oort and Fritz Zwicky arriving at similar hypotheses in 1932 and 1933, respectively using observations of galaxies [1]. Further observations have over time demonstrated that there is a significant portion of the matter in our universe that only interacts with its surroundings via gravity while seemingly not interacting via other forces such as electromagnetism[3], the only evidence of its existence its influence on everything from stellar clusters to galaxy formation [1], [3].

There are numerous models to explain what exactly Dark Matter is and what role it plays in cosmological evolution with everything from supersymmetry to quantum gravity providing explanations with massively diverging consequences for the history and future of our universe [1]. Some of these theories, however, suggest that these crafty particles can have non-zero couplings that open up for the possibility of observing them in experiments such as at colliders.

One of these experiments is the Light Dark Matter eXperiment that is designed to search for Dark Matter in the Light Dark Matter mass range of below 1 GeV [1]. The experiment will need to handle large amounts of events at a high rate. This means that the simple selection criteria used for the one electron events might have a poor resolution requiring an effective veto of events that are not relevant. This requires the use of algorithms that can based on the incoming data decide whether or not the event is worth keeping. Multiple electron events are particularly difficult. Machine Learning provides a possible way forward of identifying these types of events relying on data from the detector.

This thesis aims to investigate how machine learning, specifically Artificial Neural Networks, can be used to identify multiple electron events in the Light Dark Matter eXperiment relying specifically on data from the Electromagnetic Calorimeter and the Trigger Scinitillator using simulated data. Specifically, Convolutional Neural Networks, Recurrent Neural Networks, Graph Neural Networks, and a Convolutional and Recurrent Neural network combination were used to study and classify multiple electron events. The aim was to evaluate the different models limitations and weaknesses and see if Machine Learning can be a viable approach for when designing trigger systems in the Light Dark Matter eXperiment. Machine Learning has so far conquered the worlds of generative art and text, so how effective can it be in the search for Dark Matter?

2 Theory

2.1 Dark Matter

Dark Matter has proven itself to be one of the most powerful explanations for a series of cosmological observations that the Standard Model cannot account for. One of the clearest signals for Dark Matter comes from examining galactic rotational curves [1]. These curves plot the rotational speed of stars rotating around a galaxy's core. The rotational speed of stars around the galactic core, where most of the mass in a galaxy is observed, is expected to be higher closer to the galactic core due to its gravitational pull[1]. However, observational studies have shown that the rotational speed does not decrease but rather plateaus further away from the core contrary to Newtonian gravity and general relativity. This means that there must be more mass present than can be observed as without some modifications to our theories of gravity some mass or energy must be present for the stars to have such high rotational speeds [1], [3]. Dark Matter then becomes the only viable alternative theory to explain this phenomenon. Other evidence such as the Bullet Cluster as can be seen in Figure 1 show that there must be more invisible matter to explain the distribution of visible matter throughout our universe with many versions of galaxy formation relying on the existence of early density perturbations. If there some amount of Dark Matter present this could then explain why galaxys' started to form [1], [3]. Comparing the amount of matter and energy that correspond to baryonic matter, made out of fermions and gauge bosons, with the total amount of matter and energy in the universe, the Standard Model of physics accounts for approximately 5% [3]. 68.5 % of the remaining energy content of the universe is made of Dark Energy with 26.5 % made of Dark Matter that seemingly does not interact

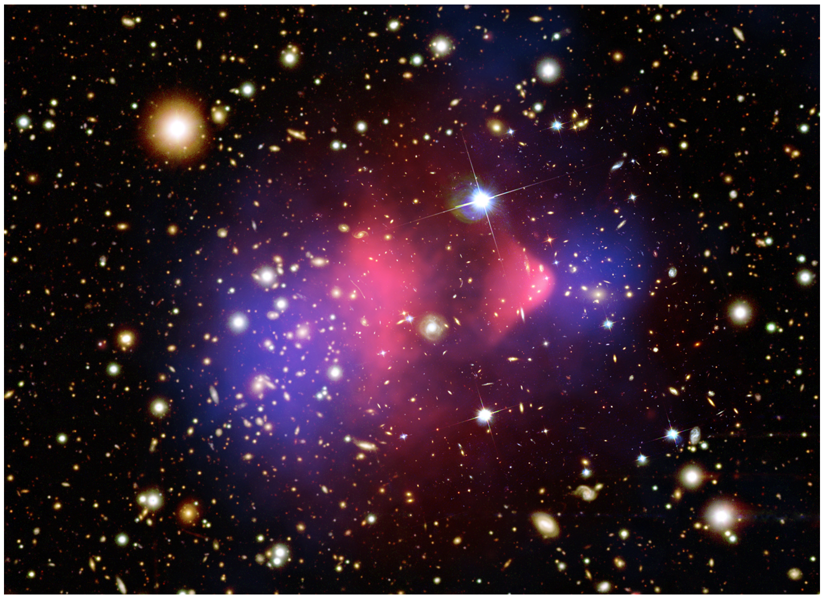


Figure 1: The Bullet cluster that shows hot intracluster gas as pink and the necessary Dark Matter distribution as blue [4].

with electromagnetism or other types of Standard Model forces such as the Strong Nuclear Force with its interactions seen through its gravitational contributions [3].

Looking at cosmological observations, limits on masses and lifetimes can be placed on this new type of matter. These particles need to be non-relativistic and since they do not interact with electromagnetic forces, should have small if any coupling strength to photons [3]. From the mass distributions of clusters of galaxies, the early epoch structure formation in the universe, and the distribution at the recombination temperature of the universe the variation of the dark matter density in a comoving volume has to be low if any at all across the history of the universe. This then implies that the lifetime of these particles must exceed the current age of the universe [3]. As no evidence has been found for Dark Matter decaying into Standard Model particles this implies that the lifetime should be even longer if this is a mechanism for their decay and so an assumption placed on most Dark Matter models is that the particles are stable [1], [3]. Alternative explanations for the origin of Dark Matter within the Standard Model have centered on neutrinos due to not experiencing electromagnetic or strong interactions. However, most of these models require the introduction of further neutrino species such as a sterile neutrino with the mass of the currently known species considered too small to account for the mass discrepancy [1], [3]. Other models usually extend the Standard Model such as through supersymmetric theories or go fully beyond the Standard Model, such as quantum gravity.

One theory for Dark Matter generation in the universe is that it is a thermal relic. In this scenario a stable heavy particle χ and its antiparticle $\bar{\chi}$ were once in thermal and chemical equilibrium with the cosmic plasma at high temperatures [3]. This implies that there is some type of interaction between Dark Matter and baryonic matter although the exact mechanism is left unspecified. The exact nature of χ depends on the specific assumptions of the model for χ with the stability of χ usually indicating that χ and $\bar{\chi}$ are produced together or that χ is its own antiparticle [3]. In this model the number of Dark Matter particles then changes only due to the creation of particle-antiparticle pairs and there is an inter-conversion between light Standard Model particles and Dark Matter as:

$$\chi + \bar{\chi} \leftrightarrow \text{Particles}. \tag{1}$$

Eventually the pair production stops as there are no more particles with enough energy to produce a Dark Matter pair and eventually the annihilation of pairs of particles stops as the density of particles becomes too low [3]. At this point the remaining Dark Matter particles freeze-out or stop being in thermal and chemical equilibrium with the cosmic plasma [3]. After this point the number of particles will remain constant forming the thermal relic that is then the Dark Matter that is observed today.

If Dark Matter is then made of stable, heavy particles it must be consistent with the current Dark Matter density of $\Omega_{DM} \sim 25\%$. If the annihilation cross section behaves approximately as $\sigma_0 \sim \frac{1}{m_\chi^2}$, where m_χ is the mass of the χ particle, then if $m_\chi \sim 100$ GeV the total cross section ends up being on the order of magnitude of the weak interaction called Weakly Interacting Massive Particles or WIMPS [1], [3]. If the annihilation cross section is too low then the number density of Dark Matter will be too large so an upper limit can be placed on the mass of these particles of $m_X \leq 100$TeV [3]. A lower bound on the Dark Matter particles mass comes from cosmological considerations and sets a mass of $m_X \geq 2$GeV as the lower limit. With a mass below that the density of Dark Matter would explode in size and overclose the universe [5]. This theory has since been supplanted by other models that push the possible mass of Dark Matter particles far lower to around the MeV scale [1], [6]. Theories containing these types of Dark Matter particles with masses below 1 GeV are generally referred to as Light Dark Matter models and are then allowed for in certain models such as Dark Sector Dark Matter [1], [7].

A Dark Sector theory is a hypothesis where there exists light, weakly-coupled particles that have not been observed yet and do not interact with Standard Model forces except gravity [7]. In this type of theory many different sectors could exist with their own respective quantum fields and forces that are all existing alongside the quantum fields contained in the Standard Model, they just cannot be observed directly [7]. One possible extension is a so-called Dark Photon. In this model a new abelian $U(1)$ gauge boson that through kinetic mixing with the photon can couple feebly to charged particles [7]. The way that these Dark Sector particles can interact with particles outside their sector would then be through a mediator particle A' that is of some mass m_A [7]. This type of mediation particle could then give insight into the Dark Sector by coupling to a charged particle and then decaying into Dark Sector particles. This type of dark bremsstrahlung emission would then result in there being missing momentum in detectors which could possibly be detected and used to test for the existence of Dark Matter particles [7], [8]. Probing this hypothesis for the existence of Dark Matter is the goal of the Light Dark Matter eXperiment.

2.2 Light Dark Matter eXperiment

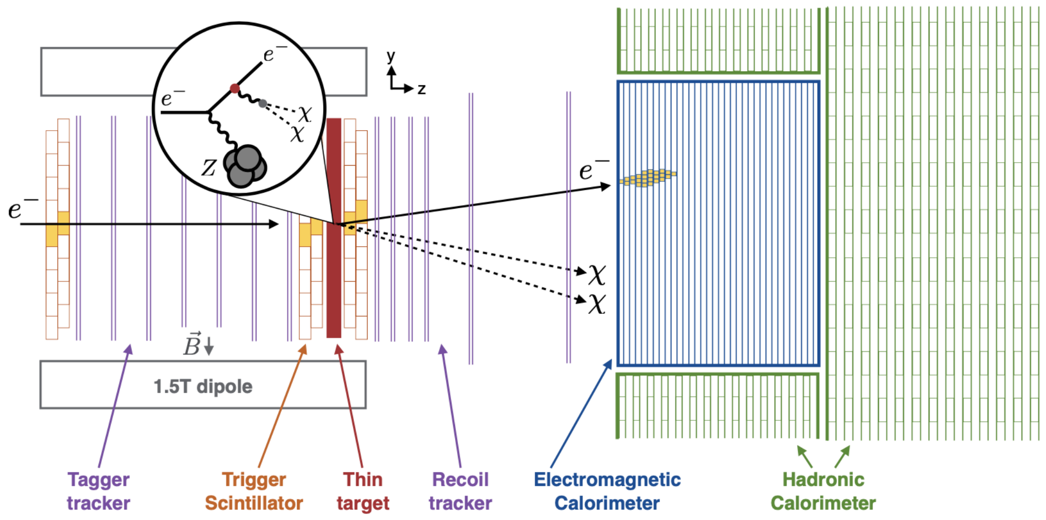


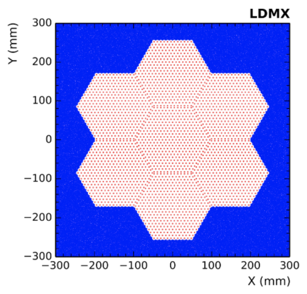
Figure 2: The proposed LDMX detector design with an example DM production process highlighting how the different subsystems will react to a signal type event [8].

The Light Dark Matter eXperiment (LDMX) is a proposed fixed-target missing momentum experiment and is designed to search for light thermal Dark Matter relics [8]. LDMX aims to explore the Light Dark Matter mass range but could possibly be extended to help examine other types of models [8]. The experiment is designed such than an electron beam is incident on a fixed tungsten target. Tagger trackers and trigger scintillators track the path of the beam before the target and a recoil tracker is placed afterward with an electromagnetic calorimeter (Ecal) and Hadronic calorimeter (Hcal) placed further downstream of the beamline as can be seen in Figure 2 [8]. When using simulations of the detector, a Cartesian coordinate system is used with the z coordinate directed along the beamline, the y coordinate directed vertically as seen in Figure 2, and the x coordinate is directed perpendicular to the z and y coordinate. The goal of the experiment is to measure the production of Dark Matter (henceforth known as DM) via the interaction of the electron beam and the target resulting in a dark bremsstrahlung process. The electron will then either radiate DM a DM particle-antiparticle pair directly or through some mediator particle that potentially decays to a particle-antiparticle pair [6], [8]. When this process occurs the electron will lose a certain amount of energy dependent on the DM particles mass and gain transverse momentum relative to the beam line due to recoiling from creating the DM particles [6]. The mediator or DM particles will then escape detection and the missing energy and momentum of the electron then indicates the presence of a signal event where DM particle have been produced. The exact properties of the produced DM particles can then be reconstructed from examining the missing energy and momentum signal.

In order to determine if an event is a signal event or not, Standard Model contributions have to be rejected. Events that involve the emission of a photon, a bremsstrahlung event, will appear to be missing momentum but can be rejected by looking at the total amount of energy in the Ecal [6]. Complex events that include the production of hadronic particles can then be rejected by using information from the Hcal as well [6], [8]. For events that involve multiple electrons, the situation is more complicated as even if DM is produced the total energy will exceed the energy of the veto for a single electron signal in the Ecal. This means other triggers need to be designed for multiple electrons events [8]. One possible way of designing a veto for multiple electrons would be to combine information from the Ecal and the trigger scintillators to indicate how many electrons are present in the event. Understanding the design of the Ecal and the Trigger Scintillator is then vital for designing new triggers to be used to inform the veto for events.

2.2.1 The Electromagnetic Calorimeter and the Trigger Scintillator

The Ecal is designed with 34 layers of Si-W. These sensing layers are made of 7 hexagonal modules arranged in a flower pattern with a central module surrounded by the remaining 6 [6], [9]. Each module consists of 432 sensing pads of 0.56cm^2 in size [6]. With this structure the Ecal has a high granularity and should then be able to effectively resolve showers and handle more than one incident electron and an example can be seen in Figure 3 [6], [9].



(a) A front side view of the Ecal.

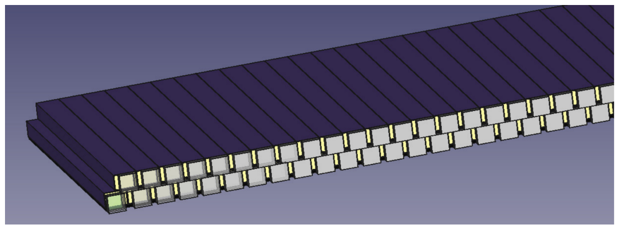

(b) A side view of the Trigger Scintillator.

Figure 3: The Ecal from a front side view with the cell centers of the modules marked in red [9]. The Trigger Scintillator proposed design showing the offset design of the scintillator bars is show in (b). In the final design the Trigger Scintillator will be placed vertically in the path of the beam [9].

The Trigger Scinitillator is designed to be able to determine how many electrons are incident in the beam [9]. It is to be constructed out of two planes of scintillators bars in the form of plastic or LYSO that are offset but overlapping in design [8], [9]. The Trigger Scintillator gives vertical positional data of where the incident electrons connected with it. The high granularity of the Ecal and the Trigger Scinitillator positional data could then possibly be combined in order to create a new way of counting incoming electrons which is important for when considering the triggers necessary for an effective veto system. A vital aspect is then how LDMX currently counts the number of electrons in each event.

2.2.2 Electron Counting

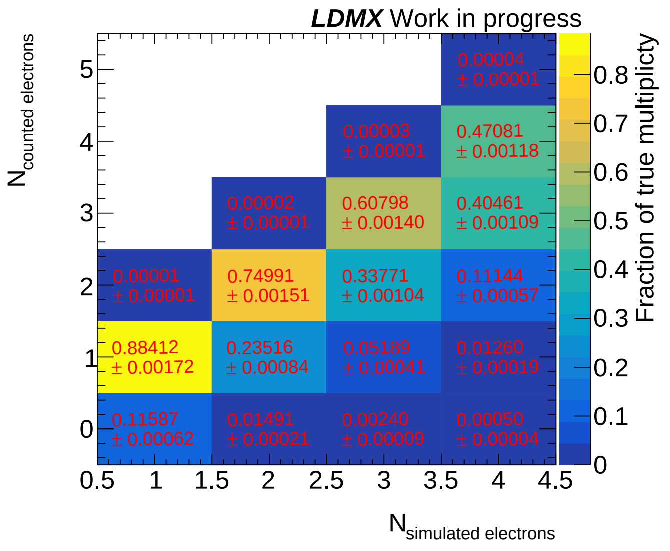


Figure 4: Plot showing the electron counting algorithm confusion matrix [10]. The confusion matrix shows the fraction of events that have been sorted into each respective class.

The current method for counting the number of electrons in the event is based on the Trigger Scintillator as seen in Figure 3. The algorithm for determining the number of electrons starts of by looking at the number of hits in the Trigger Scintillator [10]. When the electron or electrons passes through the system this registers as a hit on the Trigger Scintillator that the algorithm then uses nearest neighborhood clustering to group together with another hit. This then creates a series of separate clusters throughout the system. An amplitude cutoff also reduces the number of viable hits leaving behind a smaller number of clusters. The algorithm then applies straight line tracking across the three Trigger Scintillator modules starting from one of the clusters and attempts to find where the line then overlaps with another cluster. The cluster then needs to be within a certain tolerance or distance of this straight line and if it is then connected to the other cluster. If the cluster is shared is then also determined by looking at the Chi squared value of the straight line tracking. The number of tracks then gives the number of counted electrons in the event. In this way the number of electrons can be counted in the detector. This algorithm depends on only using the Trigger Scintillator data from the detector [10]. Since the veto for signal events relies on detecting missing energy and momentum in the detector being able to effectively count electrons becomes important [6]. At the moment the algorithm is skewed toward under counting the number of electrons in events as can be seen in Figure 4. This figure shows the confusion matrix for the algorithm where the fraction of events that have been sorted into each respective class is shown. The diagonal shows how large the fraction is of events that was correctly classified for each respective type with the lowest performance for four electron events with a fraction of 0.47081. The fractions above this diagonal show the fraction of events that have been overcounted and is comparatively much smaller when compared with the rate of undercounting as can be seen in the fractions below the diagonal. One aspect of the current veto system is based of the amount of energy in the detector.

If the total amount of energy is below 1.5 Gev [6] and there is a large amount of momentum and energy missing then this indicates a signal event might have occurred and the event is stored. If there are multiple electrons, the energy will be much higher than the veto energy and so if the Trigger Scintillator algorithm indicates that there are 3 electrons when there are actually only 2 this means that the energy of each electron might appear much lower and the event is stored as a "signal" event even when this is a false event. The safer option is then to have a bias towards undercounting the number of electrons as this might miss a couple of events but will not flood the data storage with spurious events. This is why the confusion matrix in Figure 2 is skewed towards undercounting events. With further upgrades introducing higher intensity beams to the LDMX this electron counting system will only become more vital as there will be more and more incident electrons that need to be dealt with at trigger level. Since there is no possibility of recovering these discarded events having a high accuracy model for electron counting then becomes vital to catching multielectron signal events.

This type of algorithm has so far relied on using the Trigger Scintillator but instead of solely relying on this system, data from the Ecal could be used to create a new combined trigger. In order to explore this type of algorithm, one of the ways to design new algorithms when there is a large amount of data available is Machine Learning where the computer can update its own algorithm based on input. This can then create new algorithms through iteration that might be useful for this problem [11]. To explore the viability of this approach we focused on a specific type of Machine Learning called Artificial Neural Networks to see how effective these alternative models can be at counting electrons instead.

2.3 Artificial Neural Networks

An Artificial Neural Network (ANN) are a type of machine learning algorithm inspired by the human brain [11], [12]. ANNs can "learn" from data in that its performance P in some task T can be improved with regards to some experience E [11], [12]. The task T consists of classes of tasks such as classification and linear regression. The performance P is some kind of measure of the overall performance of the ANN on the task. For a classification task this could be the accuracy of the network with regards to a specific class. The experience E is often a dataset collecting examples for the network to "learn" from [11], [12]. There are many different types of learning but the most relevant here are Supervised and Unsupervised learning. In Supervised learning, the experience E consists of a set of data points with labels. A data point could, for example, be an image with a label of "car". Unsupervised learning, on the other hand, only includes the data points with no labels [11], [12]. With a typical network, we then want to provide an output function $y(\mathbf{x_n})$ based on the input $\mathbf{x_n}$. The dataset is then represented via n patterns consisting of the input $\mathbf{x_n}$ and the target d_n. The goal of the network then becomes to have the output function equal the target such that $y(\mathbf{x_n}) = d_n \; \forall n$ [11], [12]. Structurally an ANN consists of a series of nodes with weighted connections called "synaptic weights" or ω. One set of nodes forms a layer within the network. The first layer is usually responsible for initially processing the input data and is therefore referred to as the input layer and the final layer from which the output of the network is processed is called the output layer. In between these two layers there can also be "hidden" layers. These layers increase the depth of the network and are usually required in order for the network to be able to tackle more complex tasks [12].

A typical unit of an ANN then takes as input some set of data $\{x_k\}$ that are then passed through a set of nodes with weights $\{\omega_k\}$ and a weighted summation is then performed such that:

$$a = \sum_{k=1}^{N} \omega_k x_k + b, \tag{2}$$

where b is input from a bias node. This is then passed through an activation function $\varphi(a)$ which then gives the output function $y = \varphi(a)$. The activation function $\varphi(a)$ is a free parameter of the network and the exact one has to be optimized for, similar to the loss function. Multi-layer networks use the same principle as the single unit but nodes on the same layer are evaluated together with a continuous evaluation in series from layer to layer. The weighted sum from the previous layer feeds into a node on the next layer that then outputs the $\varphi(a)$ value. This process is then repeated until the final output layer is reached where the final nodes then output the final $y = \varphi(a)$ [11], [12]. The exact number of nodes, number of layers, loss function, and activation function are all then parameters that have to be optimized for when designing the network and are called hyperparameters. Numerous different types of model architectures exist for different types of tasks and performance considerations with trial and error remaining the most common method for optimizing the the network.

2.3.1 Loss Function Optimization

In order to train the network such that $y(\mathbf{x_n}) = d_n \; \forall n$ holds, the weights ω need to updated in order to increase the performance of the network. However, in order to actually determine the performance of the network a measure needs to be chosen which is usually called a loss function $E(\omega)$. This function compares the output function $y_n = y(\omega, \mathbf{x_n})$ and the target d_n. This function is minimized, with the exact choice of function depending on the type of task being considered [11]. In this thesis we mainly consider multi category classification tasks. In this type of task the target is some type of category such as "House" or "Face" or a number; $d = 1$ for class 1 and $d = 2$ for class 2 and so on [11], [12]. The classes are often encoded via "one-hot encoding" where we set $d_{ni} = 1$ if the pattern n is in the considered class C_i such that each label becomes a vector. This means if we have three classes, then if a pattern $n \in C_1$ we write the label as $(1, 0, 0)$ [11], [12]. The loss function for this type of task then becomes the categorical cross-entropy error function of the form:

$$E(\omega) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{c} d_{ni}\ln y_{ni}, \tag{3}$$

where N is the number of data points, c is the number of classes, and y_{ni} is the output function [11]. In order to then optimize this loss function and update the weights ω we need a way of determining how the error changes with ω_k. Therefore we start of with considering the derivative of the loss function with regards to ω_k and use the chain rule to write:

$$\frac{\partial E}{\partial \omega_k} = \sum_n \frac{\partial E}{\partial y_k}\frac{\partial y_n}{\partial a_n}\frac{\partial a_n}{\partial \omega_k}, \tag{4}$$

where we can use the definition from Eq.(3) and that $y = \varphi(a)$ to give:

$$\frac{\partial E}{\partial \omega_k} = \sum_n \frac{\partial E}{\partial y_k}\varphi'(a_n)x_{nk}. \tag{5}$$

The Gradient Descent Method uses randomly initialized weights and then updates each weight in the network with:

$$\omega_k(t+1) = \omega_k(t) - \eta\sum_n \frac{\partial E}{\partial y_n}\varphi'(a_n)x_{nk}, \tag{6}$$

where $\omega_k(t)$ is the weight value at some time step t and η is a positive parameter called the learning rate. The learning rate is another hyperparameter of the network and needs to be optimized for as well [11], [12]. The Gradient Descent method can often result in the loss becoming stuck in plateaus or local minima with the consequence of the weights no longer updating properly. In order to mitigate this problem and reduce computational time we could also introduce Stochastic Gradient Descent. Now the loss function is treated as the mean of the losses of individual patterns E_n such that:

$$E = \frac{1}{N}\sum_n E_n.$$

If we then divide the training dataset into a series of minibatches of size P we can write the updating of the weights as:

$$\Delta\omega_k = -\frac{1}{P}\sum_{p}^{P}\eta\frac{\partial E_p}{\partial \omega_k}. \tag{7}$$

Iterating over all patterns in every minibatch an "epoch" has been performed [11]. Since the updating on the weights is dependent on the patterns in the minibatch the loss function has a higher probability of exiting local minima or plateus. Both the number of epochs and the minibatch size are further hyperparameters that need to be optimized for [11]. This method can be further improved by using earlier time step values with one of the most common methods called Adaptive moment estimation or "Adam". In this method three learning parameters η, β_1, and β_2 are introduced as well as two running averages $\mathbf{m}$ and $\mathbf{v}$ that are defined as:

$$m_k(t+1) = \beta_1 m_k(t) + (1-\beta_1)g_k(t), \tag{8}$$
$$v_k(t+1) = \beta_2 v_k(t) + (1-\beta_2)[g_k(t)]^2, \tag{9}$$

where $g_k(t)$ is the gradient as in Eq.(7). We then also write $\hat{m}_k(t) = \frac{m_k(t+1)}{1-\beta_1^t}$ and $\hat{v}_k(t) = \frac{v_k(t+1)}{1-\beta_2^t}$ [11]. $\hat{m}_k$ is then a weighted average of the gradients g_k while $\hat{v}_k$ is the weighted average of g_k^2. The updating of the weights $\omega_k(t+1)$ then becomes:

$$\omega_k(t+1) = \omega_k(t) - \eta \frac{\hat{m}_k(t+1)}{\sqrt{\hat{v}_k(t+1)} + \epsilon}, \tag{10}$$

where ϵ is a small number [11], [12]. This method has the benefit of that in a plateau region where g_k changes little between iterations, $|\hat{m}_k| \sim \sqrt{v_k}$ and the weights update as $\omega_k(t+1) \approx \omega_k(t) - \eta \cdot \text{sgn}(g_k)$ which avoids having negligible or vanishing $\Delta\omega_k$. Meanwhile, near the global minima of the loss function the sign of g_k will vary between updates while g_k^2 remains constant. This means that $|\hat{m_k}| << \sqrt{\hat{v}_k}$ and $\Delta\omega_k$ tends towards zero [11], [12]. This method was the one utilized for the networks used throughout the thesis.

2.3.2 Generalization and Performance

When training the network, the goal is for the network to perform well on new data not included in the training set, i.e. if the goal of the network is image recognition the network should be able to identify objects in a new image not just one of the training images. This is called the networks ability to generalize [11], [12]. The network can become overfitted when training reducing its performance on new data. In order to determine the generalization of the network, the dataset can be split into a training and a validation dataset where the network is trained on the training dataset and then the performance of the network is also tested on the validation dataset. If the network has become overfitted the validation performance will be low with multiple methods available to try and increase performance on the validation dataset.

In the course of training the network and trying to determine if the network is overfitted or underfitted, one important consideration is the bias-variance tradeoff [11]. The bias measures how the average of different models deviates from the target while the variance measures how the output function depends on the dataset [11], [12]. To have low bias a flexible model with a large number of nodes is needed but this means that it can be fitted to a specific dataset very well resulting in a higher variance. Reducing variance usually means removing nodes which can increase the bias as the model becomes more limited [11]. This tradeoff is unavoidable but its impact can be minimized or controlled with numerous methods. One way of increasing validation performance (reducing overfitting) is to add regularization terms to the loss function which then becomes:

$$\tilde{E}(\omega) = E(\omega) + \alpha\Omega(\omega), \tag{11}$$

where α is the regularization strength and Ω is the regularization term [11], [12]. Many different versions of Ω exist but most try and adjust the curvature of the loss function and balance the bias and variance of the network [11]. Another method to increase validation performance includes Dropout regularisation wherein some of the nodes are dropped for each pattern being considered with a probability p of keeping the node [11]. This can prevent overfitting by suppressing outliers affecting the training of the network and therefore making it more viable for applying to new data in the validation set [11]. A final method considered for this thesis is early stopping. In this method the performance of the validation error is monitored and the training is stopped when the validation error starts to increase. This can stop the model from becoming overfitted with the training error possibly still high. However, the method can make the network overfitted on the validation dataset instead as it now stringly influences the training. In turn, this can require the introduction of a third dataset called the test dataset [11], [12] but none was introduced for this thesis. All of these different methods have introduced further hyperparameters that also need to be specified. Many different types of models exists with different architectures and parameters but the types of networks considered for this work were: Convolutional Neural Networks (CNNS), Recurrent Neural Networks (RNNs), Graph Neural Networks (GNNs), and a combined CNN and RNN type network.

2.3.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of network that specifically handles data in grids where spatial relations are important and gets its name from the convolution operation that is included in the [11],

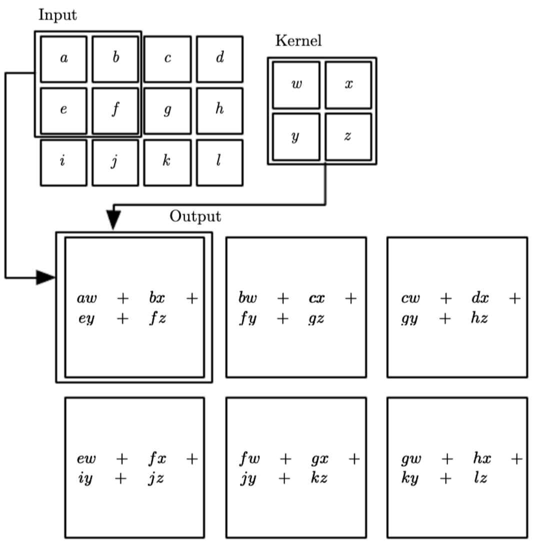


Figure 5: An example computation of a convolutional neural network in 2D using a kernel on a 2D data image [12].

[12]. This means that they are often used for image recognition with one of the earliest examples designed by Yann LeCun et al [13] in 1989. A convolution operation is of the form:

$$H(y) = (I * K)(y) = \int K(y-x)I(x)\mathrm{d}x, \tag{12}$$

where I(x) is some function and K(y-x) is some kernel where $H(y)$ is sometimes referred to as a feature map [11], [12]. Most neural network implementations prefer to work with the cross-correlation where the kernel is flipped such that $K(y-x) \to K(x-y)$ while introducing a change of variables $x \to x+y$ to write:

$$H(y) = \int K(x)I(x+y)\mathrm{d}x.$$

Usually the data is discretized and so for a 3D image $\mathbf{I}$ as input the cross-correlation sum becomes [11]:

$$H(i,j,k) = (I * K)(i,j,k) = \sum_{m,n,l} I(i+m, j+n, k+l)K(m,n,l). \tag{13}$$

The kernel acts as a feature detector over the image that picks up details in the image and filters it and the exact form of the kernel is part of the training of the network such that the network can learn to pick out the most important details [11], [12].

In a neural network, the layers usually require every output unit to interact with every input unit with all the nodes between layers connected [11], [12]. However, CNNs often use "local" kernels that are smaller than the input dimensions, creating a network with spare interactions. This reduces the number of parameters and can create networks that are more efficient in both memory and in the number of operations. For example, in a regular network with M input and N output there are $M \times N$ parameters which means it has a runtime of $O(M \times N)$ [12]. But in a sparsely connected network with a maximum number of connections K it only has $K \times N$ parameters with a $O(M \times N)$ runtime [12]. So by adjusting the size of K the efficiency of the network can be increased even while detecting features in the image [11], [12]. CNNs also employ parameter sharing where the functions in a model can share parameters. Usually a network uses every weight once while computing the output of the considered layer, but a CNN uses the corresponding unit of the kernel for the input. This means that for a 4×4 matrix if a 2×2 matrix is applied it would usually result in 16 weights but these are reduced to just 4 with parameter sharing [12]. This has the benefit of further reducing memory but also results in the layer becoming equivariant to translation [12]. An important type of layer for a CNN is the

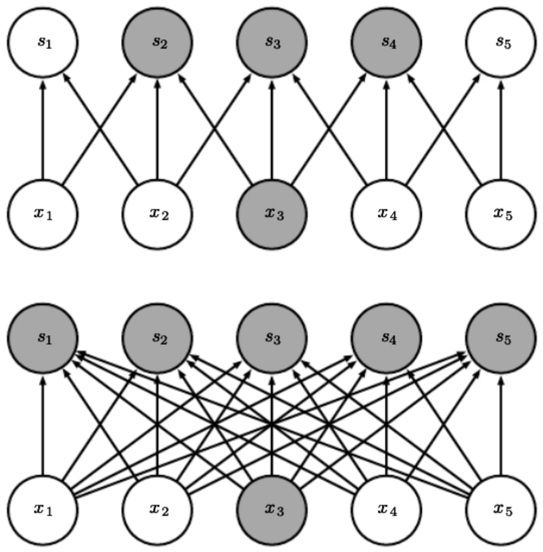


Figure 6: The lower set of nodes show a fully connected network and the upper set shows a sparsely connected network [12].

Pooling layer. The Pooling layer takes as input the output from several nodes in for example a rectangular area of nodes and applies some type of summation operation to them [12] with one of the most common being the max Pooling layer [11]. This layer then uses an activation function of the form $\max_k\{x_k\}$ where it picks the input with the largest weight [11], [12]. This results in the network becoming translationally invariant and is therefore better for networks trying to determine if a feature is present in an image rather than exactly where it is and also reduces the number of hidden nodes in the network [11], [12].

The architecture of a typical CNN then takes the form of an input layer that gets passed to the convolutional layer with the exact number of layer and pooling layers depending on the requirements of the network [11], [12]. The output from the convolutional layers can be passed to a flattening layer that reduces the output to a one dimensional vector or a multi-layer network that is fully connected to handle the feature map from the convolutions [11]. Depending on the type of task that the CNN is designed for, a final layer of nodes can be used. For a multi-categorical task the final number of nodes is set equal to the number of categories being considered [11], [12].

2.3.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of networks that are designed to handle sequential data such as text or time series data [11], [12]. First suggested in 1986 by Rutherford et al [14], RNNs differ from other types of networks by enabling feedback connections. The previous types of networks considered so far have been feed forward with no way of propagating updated weights to earlier nodes in the network, but with RNNs backpropagation can be implemented across the network [12]. Similar to how the main operation of CNNs is the convolution, recursion with the ability of a node feeding its weight back into itself being vital for the network [12].

There are neural networks that can tackle certain types of sequential data, but they lack the feedback connections that enable more complex modelling with the tradeoff being that they are easier to train than true feedback connection networks [11]. One illustrative type of RNN is the "Simple Recurrent Network" where some input data in the form of a sequence, for example $x(0)$, $x(1)$, etc until $x(T)$ where T symbolizes the final timestep in the sequence, is treated by the network [11], [12]. The most basic version of this network consists of one input node, one hidden node, and one output node with corresponding weights U and V between the input and hidden nodes and hidden and output nodes, respectively [11], [12]. A feedback weight

W is also added for the hidden node such that in this type of model the output function $y(t)$ becomes:

$$y(t) = g_o(Vh(t)), \tag{14}$$
$$h(t) = g_h(Ux(t) + Wh(t-1)), \tag{15}$$

where g_o is the output activation function and g_h is the hidden layer activation function [11]. Every timestep output of the network then depends on the previous steps thereby allowing for the sharing of weights across the sequence [11], [12]. This type of network can be unfolded in time such that if three timesteps are considered it will resemble a neural network with 3 hidden layers [11]. In order to train this type of network the loss function becomes:

$$E(\omega) = \sum_{t=0}^{T} E(t, \omega) \tag{16}$$

where the summation goes over every timestep t [11]. The gradient of the loss function then needs to be computed for every timestep and every weight. However, for certain weights such as for W and U taking the derivative at each timestep requires using the chain rule and summing over earlier timesteps k such that:

$$\frac{\partial E(t)}{\partial W} = \sum_{k=0}^{t} \frac{\partial E(t)}{\partial y(t)} \frac{\partial y(t)}{\partial h(t)} \frac{\partial h(t)}{\partial h(k)} \frac{\partial h(k)}{\partial W}. \tag{17}$$

This means that a RNN is trained by unfolding it in time and then performing backpropagation as in a standard neural network and is denoted as backpropagation-through-time [11], [12]. This type of propagation suffers from vanishing gradients for long time sequences where due to the series of chain rule multiplications of gradients, if one of them is small it results in the total gradient becoming small and possibly making training slow. The opposite problem could occur when the gradients becoming large referred to as exploding gradients [11], [12]. There are many types of RNNs that implement other methods for long sequences to avoid this problem with backpropagation through time. The two versions considered here are Long Short-Term Memory (LSTM) and Gated Recurrent Network (GRU).

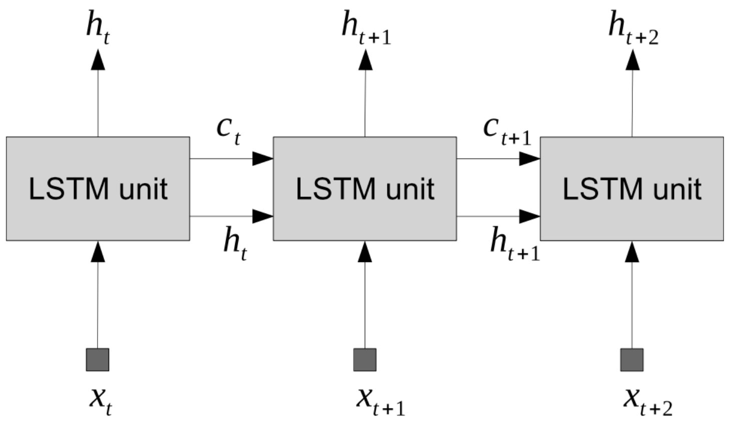


Figure 7: An unfolded LSTM network with the internal memory $c(t)$ and hidden node output $h(t)$ shared between units [11].

An LSTM unit was first introduced in 1997 [15] in order to handle long term dependencies of sequences [15]. In an LSTM network instead of having a node with recurrence that can be unfolded in time, it is now replaced by an LSTM unit [11], [15]. This new unit has a hidden node output $h(t)$ and an extra value $c(t)$ called the internal memory of the unit as can be seen in Figure 9 [11], [15]. These LSTM units have internal recurrence but also introduces gates that are part of the computational process [12], [15]. These are called forget f, input i, and output o gates that are then used in order to compute the output from the LSTM unit and will range in value from 0 to 1 [11], [15]. If we once again consider the simple example of one hidden node the gates equations take the form:

$$i(t) = \sigma(x(t)U^i + h(t-1)W^i), \tag{18}$$
$$f(t) = \sigma(x(t)U^f + h(t-1)W^f), \tag{19}$$
$$o(t) = \sigma(x(t)U^o + h(t-1)W^o), \tag{20}$$

where U^y and W^y with $y \in \{i, f, o\}$ are new weights and $\sigma()$ refers to the logistic function [11]. To then use these gates to compute the internal memory $c(t)$ and the hidden node output $h(t)$, a candidate value for $c(t)$, $\tilde{c}(t)$, is computed:

$$\tilde{c}(t) = \tanh(x(t)U^c + h(t-1)W^c), \tag{21}$$

where U^c and W^c are once again new weights for the internal memory [11], [15]. The internal memory $c(t)$ and $h(t)$ then becomes:

$$c(t) = c(t-1)f(t) + \tilde{c}(t)i(t), \tag{22}$$
$$h(t) = \tanh(c(t))o(t). \tag{23}$$

The internal memory is then calculated by processing the previous timestep through the forget gate and the candidate memory through the input gate while $h(t)$ is calculated by processing the internal memory through the tanh function and the output gate [11], [15]. The combinations of these gates then allows for the computation of long term dependencies where in practice many LSTM units are used [11], [15]. The LSTM network can avoid vanishing and exploding gradients as the derivative for $\partial c(t)/\partial c(t-1)$ becomes finite [11], [12].

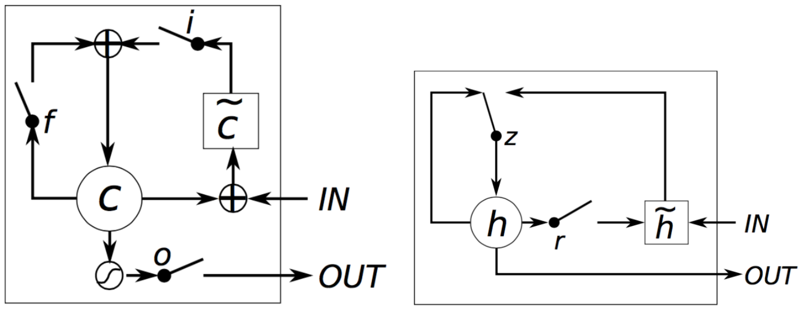


Figure 8: A comparison between the LSTM computational process involving its f, i, and o gates and the GRU computational process involving r and z gates [11].

The GRU type of network was first introduced in 2014 by Kyunghyun Cho et al [16] and uses two gates instead of three gates as for the LSTM without any internal memory $c(t)$ and a comparison can be seen in Figure 8 [16]. It consists of a reset gate r that determines how to combine values across timesteps and the update gate z that adjusts how much of the current value to retain [11], [16]. Their respective equations analogously to LSTM then become [11], [16]:

$$r = \sigma(x(t)U^r + h(t-1)W^r), \tag{24}$$
$$z = \sigma(x(t)U^z + h(t-1)W^z). \tag{25}$$

Instead of computing a candidate for the internal memory, a candidate is computed for the hidden node output $\tilde{h}(t)$ such that we have:

$$\tilde{h}(t) = \tanh(x(t)U^h + (h(t-1)r)W^h), \tag{26}$$
$$h(t) = (1-z)\tilde{h}(t) + zh(t-1), \tag{27}$$

where we can then recover the simple recursive network by setting z to 0 and r to 1 [11], [16]. These two types of networks were then used in the thesis when training RNNs. Another version of a neural network that acts on another type of data representation are the Graph Neural Networks (GNNs).

2.3.5 Graph Neural Networks

A graph $\mathbf{G}$ is a type of data structure which consists of a pair $(\mathbf{N}, \mathbf{E})$ of nodes $\mathbf{N}$ and edges $\mathbf{E}$ [17]. Nodes could represent a data point such as an object or some concept while the edges describe the relationships or connections between the points [17]. There are different types of graphs with directed graphs referring to graphs with directed edges in the form of vectors and undirected having no vector edges just connections [17]. There are also different types of tasks that can be performed on graph structures in the form of graph-level that acts on the entire graph or node-level that only acts on the nodes [17]. A state $\mathbf{x}_n$ can be attached for every node n that depends on the neighbors of the node. This then contains a possible representation of the concept represented by the nth node. An output or decision about the concept associated with the node denoted $\mathbf{o}_n$ can also be produced [17]. We can write these as:

$$\mathbf{x}_n = f_{\mathbf{w}}(\mathbf{l}_n, \mathbf{l}_{co[n]}, \mathbf{x}_{ne}, \mathbf{l}_{ne[n]}), \tag{28}$$

$$\mathbf{o}_n = g_{\mathbf{w}}(\mathbf{x}_n, \mathbf{l}_n), \tag{29}$$

where $\mathbf{l}_n$ refers to the lables of the node n, $\mathbf{l}_{co[n]}$ are the labels of the edges of the node, $\mathbf{x}_{ne}$ are the states of the nodes in the neighborhood, and $\mathbf{l}_{ne[n]}$ is the labels of the neighboring nodes [17]. f_ω is a parametric function that describes how the node n depends on its neighbors and is called the local transition function while g_ω is a parametric function that describes how to handle the output from the node [17]. The exact type of neighborhood is dependent on how the model is defined. In order to compute an output from the overall graph, the states, labels, node labels, and outputs are combined into respective vectors, $\mathbf{x}$, $\mathbf{o}$, $\mathbf{l}$, and $\mathbf{l}_N$ [17]. Then a global state and output can be written as:

$$\mathbf{x} = F_\omega(\mathbf{x}, \mathbf{l}), \tag{30}$$

$$\mathbf{o} = G_\omega(\mathbf{x}, \mathbf{l}_N), \tag{31}$$

where F_ω is a global transition function and G_ω is a global output function [17]. $\mathbf{x}$ and $\mathbf{o}$ should then be uniquely defined with an output for every node. In order to compute the state the state is iterated over in time t such that:

$$\mathbf{x}(t+1) = F_\omega(\mathbf{x}(t), \mathbf{l}), \tag{32}$$

where training the network becomes akin to solving a series of nonlinear equations [17]. A feedforward network, for example, is then trained until the network becomes smaller than some error ϵ such that $||\mathbf{x}(t) - \mathbf{x}(t-1)|| < \epsilon$ [17]. We can compare this with a RNN network where the encoding of the GNN in a feedforward network resembles the RNN while we can see a CNN as a special type of GNN where the graph is arranged in a regular grid rather than a more loose structure allowed for in other graph data structures [17]. There are many types of implementations for a GNN type network, but when considering graph classification there exists many implementations of graph convolutional layers that apply operations similar to how a convolutional layer in a CNN works.

One version is an implementation of the Weisfeiler-Leman Algorithm for labelled graphs as in a classification task [18] In this type of network the updates of $\mathbf{x}(t)_i$ are performed as:

$$\mathbf{x}(t)_i = \mathbf{W}^1(t-1)\mathbf{x}_i(t-1)\mathbf{W}^2(t-1) \sum_{j \in N(i)} e_{j,i} \cdot \mathbf{x}_j \tag{33}$$

where $\mathbf{W}^1$ and $\mathbf{W}^2$ are parameter matrices and $e_{i,j}$ are the edge weights. The summation goes over the neighborhood of the nodes [18]. A Relu or sigmoid output function is then used with the non-linear function allowing for the processing of a greater variety of graphs [18]. This type of layer is then combined with multiple others to extract features.

2.3.6 CNN and RNN

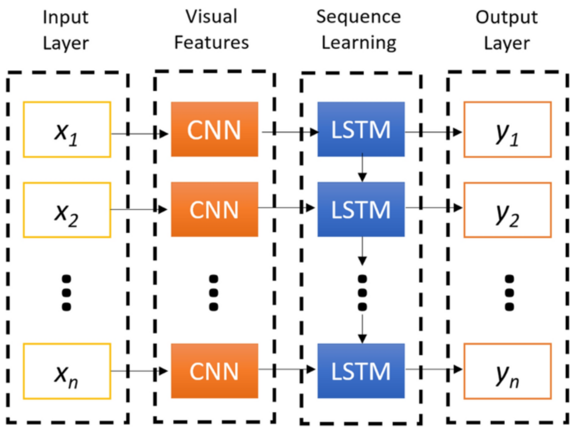


Figure 9: An example of a combined CNN and LSTM network working on microRNA structures [19].

For types of data that are sequential in nature but have a grid-like structure such as a film, applying a normal CNN can sometimes be ineffectual in propagating the weights. An alternative approach is to instead rely on a model that combines RNN and CNN architecture to propagate the sequential nature of the data across for example the frames in a film [20]. In this type of structure the key unit is a recurrent convolutional layer that combines the feature map from a CNN layer and then passes it to an RNN layer where longterm dependencies are calculated [20]. These can then be combined with Pooling layers to form a full network. So in the case of a film each frame would be treated as a 2D structure and a 2D convolutional layer would act on the frame and this is then passed to an RNN layer such as an LSTM or GRU [20]. This then could act as an alternative to 3D convolution and possibly treat the evolving nature of the data better than an RNN or CNN could do separately.

3 Method

3.1 Data Generation

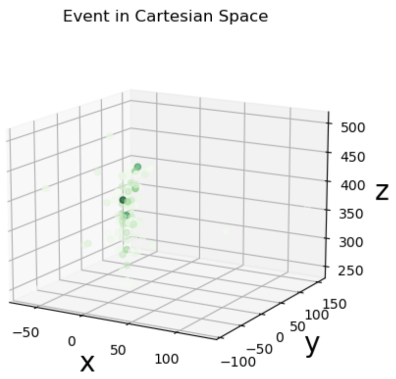
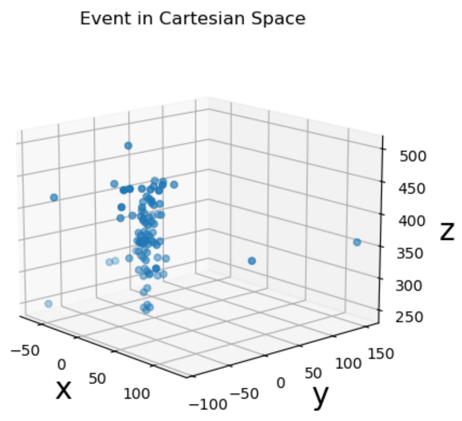


Figure 10: Plots of a one electron event in Cartesian space with the energy of each hit used for a color map for the intensity of the hits.

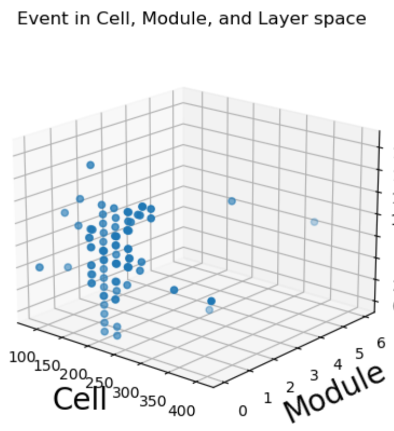
In order to train a neural network a sufficient amount of data needs to be generated in order to optimize the weights. The exact amount of data that is necessary is usually dependent on the type of task but the general consensus is that more training data is better [12]. Due to the LDMX still being designed there is currently no experimental data available to train the networks with and instead simulated data had to be used. The LDMX collaboration employs an event processing and simulation framework built in C++ called ldmx-sw [6], [8] built upon Geant4 [21]. With this software the current version of the detector geometry and its interactions with the electron beam can be simulated. The version of the detector used for the simulations was version 14 as the most current version as of time of writing.

Initially smaller datasets ranging up to 20000 events in size were used to test the feasibility of the different networks and ensure that they worked as intended for the data being considered. For this thesis events with up to four electrons were used in order to test the different performances of the networks on a sufficient variety of events. Further events could be chosen but that is left for future research endeavours. To scale up the number of data points, 10 million data points of each multiplicity was generated using the lightweight distributed computing system (LDCS) [22]. These were generated in files of 10000 events. However, due to some computational errors some of these files were corrupted so in reality there was on average 7 million events per multiplicity. Ultimately due to time constraints only 1.5 million of each of type was used when training the networks. 100,000 of each class of event was used for the validation data set. The data was generated as a series of ROOT [23] files and then extracted with an example event shown in Figure 10.

3.2 Preprocessing



(a) Event in Cartesian space.



(b) Event in cell, module, and layer space.

Figure 11: Plots of a one electron event in Cartesian and cell,module, and layer space using the same event as in Figure 10.

The data from the LDMX-sw simulation software allows for easy access to positional data of the different hits in the Ecal. However, in Leo Östman Master Thesis [24] that investigated the use of the Ecal for classifyihng events using Machine Learning, attempts to make the situation more realistic meant discretizing the Ecal positional data into cells with a certain set size. In order to try and avoid this slightly artificial construction for this thesis the specific Ecal locations in terms of cell, module, and layer number were used. This required further processing of the data in terms of converting a so-called EcalID produced in each event for each hit. For each type of network the energy, cell number, module number, and layer number was used as input data and a comparison with the event in Cartesian space is seen in Figure 11. The mapping of the module on a layer with one cell marked can be seen in Figure 12. One dataset was then prepared using only the data from the Ecal. The data from the Trigger Scintillator were also used with two different datasets prepared.

The Trigger Scintillator does not measure the energy as a particle passes through it instead measuring

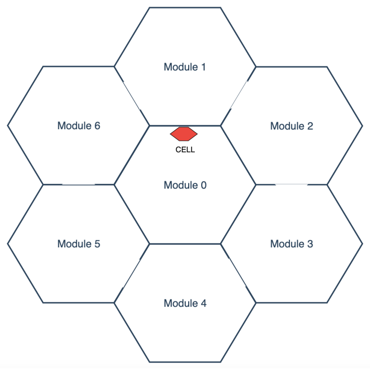
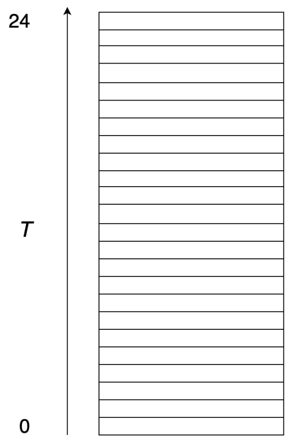
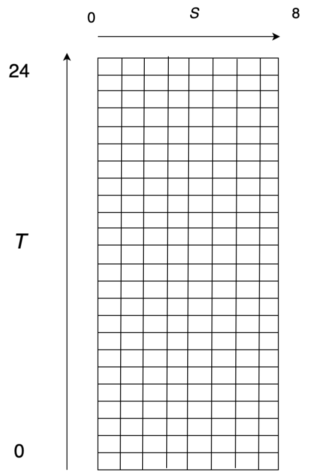


Figure 12: The module mapping from 0 to 6 for some layer L with some cell marked in red.

a small amount of energy after the particle has passed. As a result the energy of the hit or hits from the Trigger Scinitillator was set equal to 1 MeV to ensure the hit is still recognized by the network when training even though the exact energy of the electron at that point is not available. The initial data on the position of the beam electrons on the Trigger Scintillator is given in Cartesian coordinates and so needs to be made compatible with the new cell, module, and layer coordinate system. The Trigger Scintillator currently consists of 24 horizontally stacked bars so the Cartesian coordinates of the hit were translated into a coordinate $T \in [0, 24]$ as can be seen in Figure 13 (a). The Ecal has 34 layers so for the training of the network the Trigger Scintillator was treated as a 35th layer that is located before the Ecal. The module number was set equal to 0 for the hits on the Trigger Scintillator while the cell number was set equal to the coordinate T. For the second dataset a hypothetical extra layer is added to the Trigger Scintillator in the form of 8 vertical bars arranged behind the current layer. This means that the hits on the Trigger Scintillator can now be converted to a set of two coordinates: $T \in [0, 24]$ and $S \in [0, 8]$ as can be seen in Figure 13 (b). The T coordinate is then placed as the cell coordinate while S coordinate is used as the module number. As there are only 7 modules on the Ecal an extra "module" is added for the 35th layer so that S then takes the place of the module coordinate for the Trigger Scintillator hit. These data points then needed to be further processed for each respective neural network architecture for it to be able to train and predict with the data.



(a) Trigger Scintillator mapping with 24 horizontal bars.



(b) Mapping with an extra layer of 8 vertical bars.

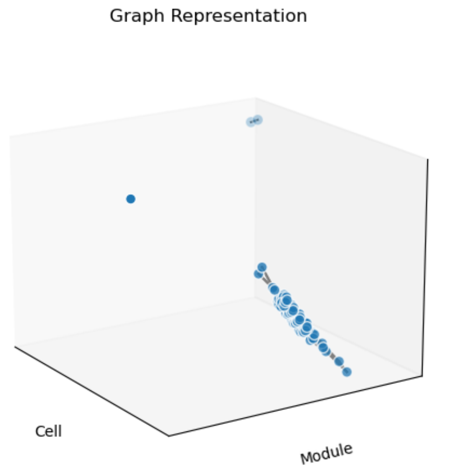
Figure 13: Trigger Scintillator mapping for only 24 horizontal bars with $T \in [0, 24]$ in (a) and with an extra 8 vertical bars with $S \in [0, 8]$ in (b).

For the CNN, the data was represented as an array of dimensions (35,8,450) where we use 35 for the number of layers that now include the Trigger Scintillator, 8 modules including an extra to accommodate the Trigger Scinitillator, and 450 is the maximum cell number which is higher than the 432 cells available on each module [8], [9] but was chosen due to the CNN handling of the arrays becoming easier to predict when training the network. Each point of the array where a hit occurred is then filled with the corresponding energy. This array can be seen analogous to a voxel (3D pixel) image with the energy corresponding to the intensity of the voxel. For the CNN and RNN combined network the same array is used except that instead of treating the array as a 3D image filled with voxels, each layer can be considered as a frame in a film with (8,450) describing the dimensions of each frame. The CNN and the CNN and RNN combined network were then trained on these arrays. For these types of networks the data must be in the same format to ensure it still has predictive power.

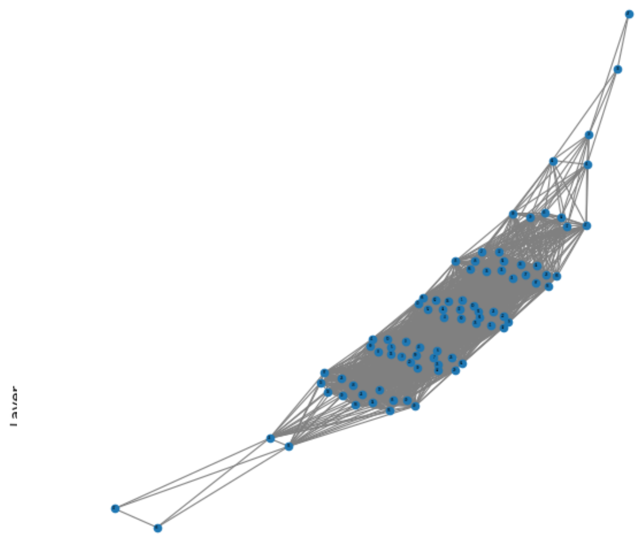
RNN type networks are designed to deal with sequential data and as a result the data had to be transformed into sequential form. Since the hits in the Ecal are arranged in 3D space, several hits can exist on the same layer coordinate. This prevents the layer from being used as a time coordinate and instead the maximum number of hits for each type of event was used as the time coordinate. This means that for example the two electron events where the maximum number of hits was found to be 250 consists of a sequence of 250 time steps with 155, 360, and 450 for one electron, three electron, and four electron events, respectively. This was done so that the network could use each hit as a sequential data point but does remove some information about when the hits actually occurred in the event. The network then standardizes the number of sequential data points by padding the sequences with zeros that the network is then told to ignore. The value of each hit is treated as a combination of the cell, module, layer, and energy using the following formula:

$$H = L \cdot 10,000,000 + M \cdot 100,000 + C \cdot 1000 + E \tag{34}$$

where H is the total value of the hit, L is the layer, M is the module, C is the cell, and E is the energy. When performing the calculation it was normalized using 3564500 as the highest value without energy included. This is common practice when dealing with large numbers for neural networks as it is better for memory and computational times.



(a) Total event graph.



(b) Zoomed in region with the largest number of nodes in the event.

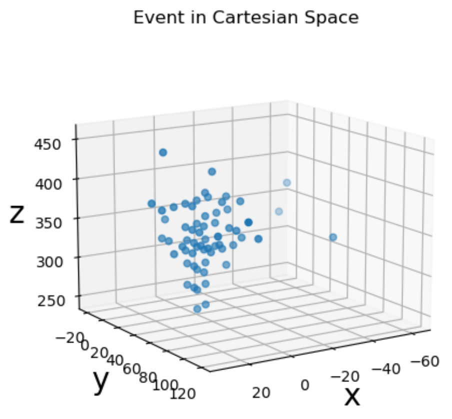
Figure 14: Plots of a one electron event as a graph data structure in 3D. The picture on the right highlights the region with the largest number of nodes in the graph to demonstrate the connections of the nodes.

Graphs require a bit more processing than the other types of data due to having to calculate the edges of the graph. Each hit was treated as a node in the graph with the features of each node set equal to the cell, module, and layer coordinates with the energy added as a final feature. Each node also had the position in cell, module, and layer space recorded as this allows for plotting and geometric relations between the nodes. The edges of each node was then calculated using geometric distance. For this a set radius had to be employed for the neighborhood of each point with the value set equal to 40. This allowed for most of the nodes to be connected with only the strongest outliers ending up outside the neighborhood of the majority of nodes in the networks. The edges of graphs can be directed or undirected in that if there is a relation between the nodes the edge can be pointed towards another. For the graphs considered here the edges were undirected as the points are treated as a point cloud. An example can be seen in Figure 14 where the majority of the nodes end up interconnected but some of the outliers end up with no or only a few edges. The visualization of the event using graphs also does not correspond perfectly to the events seen in Figure 11 as the 3D positions of the nodes is calculated using a minimization algorithm for their distribution. Each graph also has a label added to it for its specific event type which then allows for the training of a graph classifier network on the data.

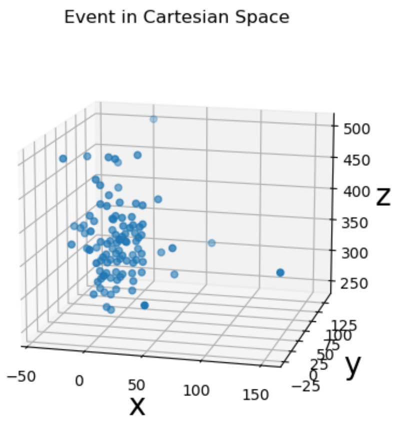
3.3 Event Types

For this work only the Ecal and Trigger Scintillator data were used so there are mainly three different types of events that need to be considered: signal events, bremsstrahlung events, and non-bremsstrahlung events. The neural networks can be more or less effective with certain types of events as the variation in event type adds an extra nuance for the networks to work through beyond just the four different types of classes. For example, due to the bremsstrahlung events involving emitted photons that will register as hits in the Ecal there is a possibility that a one electron bremsstrahlung event that emits one photon can be confused with a two-electron non-bremsstrahlung event. Similar issues could then occur if more photons are involved thereby increasing the possibility of missclassification. The same type of problem could of course arise for signal events that involve the emission of photons with an added difficulty of the incident angle of the electron being different from an event that has not been deflected due to the emission of a dark photon. In order to properly analyze the strengths and weaknesses of the different neural networks, their performance on bremsstrahlung events compared with non-bremsstrahlung events can then give insight into whether or not they can handle the variation introduced by the different event types. A set of events can be seen in Figure 15 with one electron events of each type and one two electron event. To then determine how many events involve the release of photons that can actually be detected at LDMX, the distribution of electron energy versus photon energy was investigated with the result seen in Figure 16 for 20,000 single electron events. The diagonal line is then the highest energy photon in each event with the photon and electron energy at that point adding up to the 4.5 GeV used to generate the event. The vast majority of the photons are low energy with most being below 10 MeV in energy. These photons are so low energy that they might be difficult for the detector to resolve as separate electromagnetic showers. Therefore a cut was applied to the energy of the photon of 10 MeV so that only events with photons above that energy counted as bremsstrahlung events. The number of photons was then recorded as well which could then be used when examining missclassified events but was not used while training the networks.

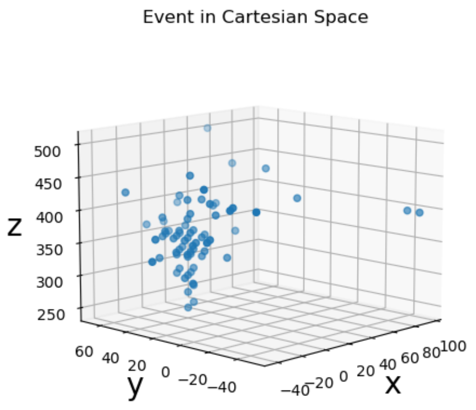
For signal and non-bremsstrahlung events the main variation will be in incident angle which would be the main source of difference between the two types of events if the network was designed to specifically distinguish between them. However, since the goal of the network is to distinguish between events with different number of electrons this variation should be able to be absorbed by the weights. For events that involve multiple electrons there is possible difficulty when two or more electrons enter the Ecal in close proximity. If they enter the Ecal or the Trigger Scintillator with only a small separation between them they might register as the same hit possibly tricking the neural networks into thinking that the event is, for example, a two electron when it is a three electron event. Therefore one of the things that was looked at when it came to missclassified events was how close the incident electrons were on the Ecal with the distance between electrons and the average distance recorded. Looking at the event type and distance between electrons can then give insight into what the weaknesses of the respective neural networks are and where they can be more effective.



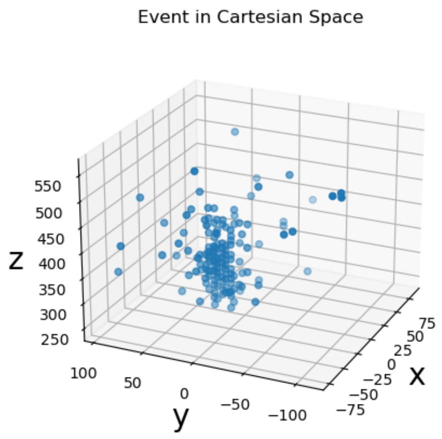
(a) One electron event that passes the veto.



(b) One Electron bremsstrahlung event with 4 emitted photons.



(c) One electron non-bremsstrahlung event.



(d) Two electron non-bremsstrahlung event.

Figure 15: Four different types of events in Cartesian space. Left-to-right they show a one electron signal event, a one electron bremsstrahlung event, a one electron non-bremsstrahlung event, and a two-electron non-bremsstrahlung event.

3.4 Neural Network Models

3.4.1 Model Implementation

In order to implement the different neural networks different libraries in python were used. For the CNN, RNN, and the CNN and RNN combined network the Keras and Tensorflow libraries [25], [26] were used to implement them. For the GNN the PyTorch Geometric [27] library was used as it offered the flexibility over Keras of being able to have variable size graph structures [25], [27]. The different models code implementations can be seen at [28]. There are some structural similarities between all networks used for this thesis. This includes adding a flattening layer and a dense layer with a number of nodes equal to the number of classes before the output is recorded. The output layer of each network also used a soft-max activation function of

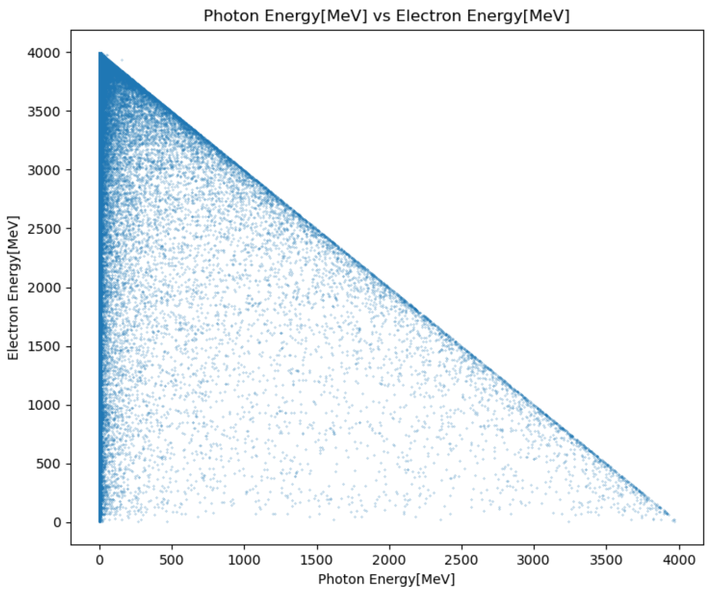


Figure 16: The distribution of photon energy and electron energy for 20,000 one electron events.

the form:

$$y_i(\mathbf{x}_n) = \frac{e^{a_{ni}}}{\sum_{i'} e^{a_{ni'}}}, \tag{35}$$

where a_{ni} is the argument passed to the ith node from the pattern n and input data $\mathbf{x}_n$ [11]. This gives the probability of the input belonging to some class and if the network is then applied to some input data the highest probability is used to assign the data point to a specific class.

In order to deal with the large size of the dataset the events were saved in chunks of 1000 as a series of files for each type of data as the entire dataset could not be loaded into the RAM at once. This meant that when loading the data the size could be kept manageable and within the available RAM on the cluster. Parallelisation was used for the computations as multiple processors could be used when on the cluster as facilitated by Keras and helped to reduce computational times. However, each file had to be loaded piece by piece which required the use of a generator that would shuffle the collection of files, pick the number of files required for the mini batch size, and then load each file. The split into validation and training set was set at 0.3 and 0.7, respectively, from the total amount of data. In order to measure the performance of the model while training, the training set accuracy and loss was used while the validation set accuracy and loss was used to implement early stopping. Since the type of task for these networks is a classification task one aspect of the performance of the model is the True Positive Rate (TPR) and its True Negative Rate (TNR) [11]. These are defined as:

$$TPR = \frac{TP}{P}, \tag{36}$$

$$TNR = \frac{TN}{N}, \tag{37}$$

where TP is the amount of correctly identified events, P is the total number of events of a specific type, TN is the amount of events identified as another type, and N is the number of events of this other type. In the case of two classes these two have a clear definition with P standing for one specific class and N for the other. The False Positive Rate (FPR) and False Negative Rate(FNR) can also then be defined as:

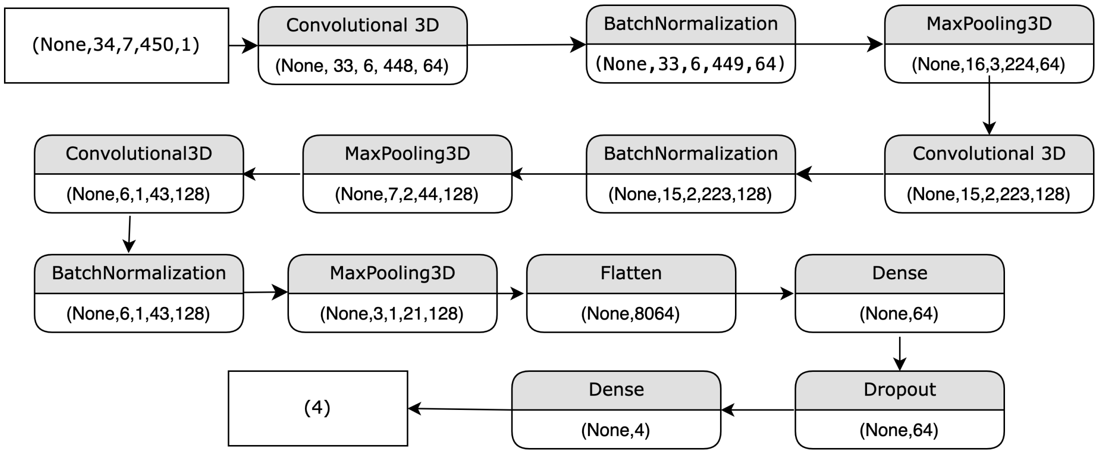
$$FPR = 1 - TNR, \tag{38}$$

$$FNR = 1 - TPR. \tag{39}$$

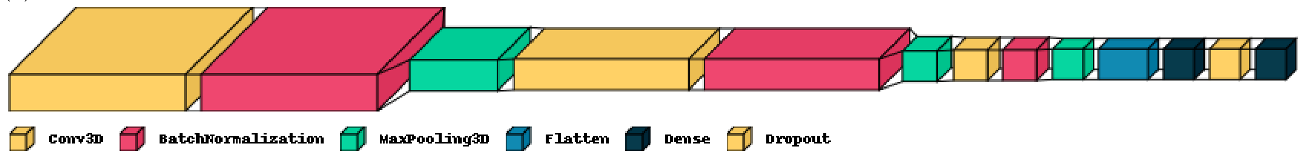
The FPR can be seen as the rate of overcounting for a specific class of events while the FNR can be considered the rate of underfitting for a specifc class. For more than two classes a specific TPR and TNR can be defined for each class which can be used to make a confusion matrix that then describes the performance of the network for each respective class [11]. The LMDX typical beam electron arrival rate is of the order 37.1 MHz. The trigger system is meant to reduce this to a rate of 5 kHz that can then be stored [8]. This implies that the network should have a maximum overall TNR of the order $\sim 10^{-5}$ so the aim would be to have one missclassification for every 100,000 event being processed.

4 Results

4.1 CNN



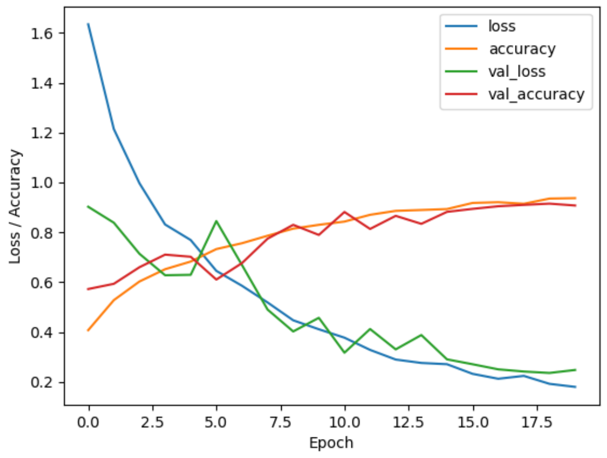
(a) The network architecture for the CNN used for the Ecal dataset.



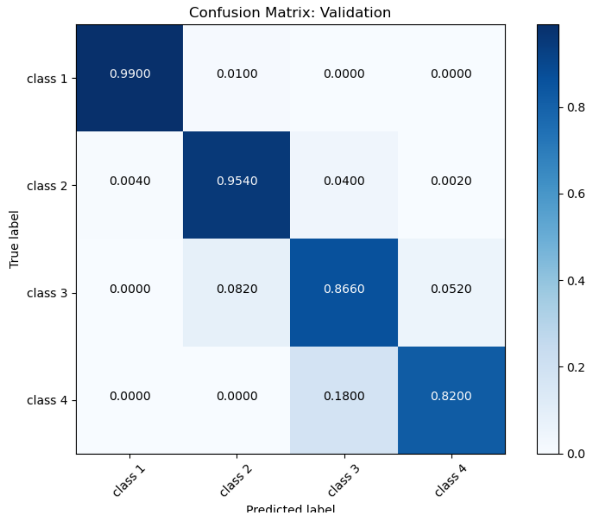
(b) The network architecture as visualized using visualkeras [29].

Figure 17: The network architecture for the CNN that was most successful with the Ecal dataset consisting of 3 convolutional layers combined with batch normalization and maxpooling layers. These were then followed by a flattening and dense layer. A dropout layer was added to control overfitting.

The most successful architecture for a CNN training with the Ecal dataset can be seen in Figure 17 with (a) showing the structure and flow of the model and (b) showing a visual representation of the model. It consists of 3 convolutional 3D layers each followed by BatchNormalization and MaxPooling layers. This is then followed by a Flattening layer, a densely connected layer, and finally a densely connected output layer with 4 nodes provides the probabilities for each class. The convolutional layers had 64,64, and 128 hidden units respectively while the dense layer had 64 units. This meant that the model had 715,140 where 714,500 are trainable and 640 are non-trainable, the non-trainable arising from the Batch Normalization. The validation accuracy of this model was 0.9075 with a validation log loss of 0.248. The maximum TPR was for the single electron events with 0.990 and the lowest was for 4 electron events with 0.820 as can be seen in Figure 18. Looking at the total amount of overcounting in the model we can see that there is a total FPR of 0.102 with the highest FPR for four electron events. The total amount of undercounting in terms of FNR is 0.266 so the model is overall skewed towards undercounting the number of electrons in events. 20 epochs were used for training with a learning rate of 0.0001 and a 32 minibatch size. It also used a dropout rate of 0.6. As can be seen in Figure 18 (a) the training was relatively stable with relatively few spikes in either variables. Since the validation loss is still quite spiky this suggests that other regularization techniques such as L1 or L2 could have been used besides dropout. More data could also make the training of the model more smooth. The model size was 8.89 MB with an evaluation speed of 0.761 seconds which like all of the evaluation speeds were calculated as an average of 1000 evaluations of events as seen in Table 1.

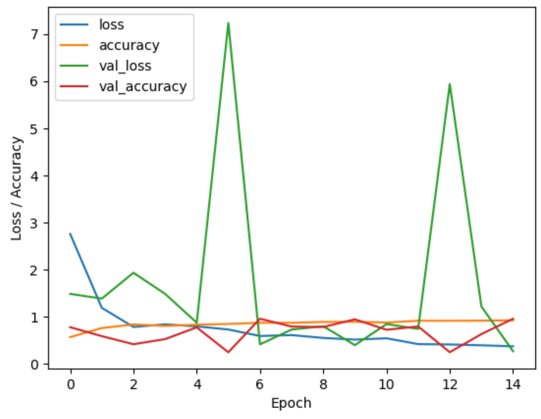


(a) The training history of the network training on the Ecal dataset over 20 epochs.

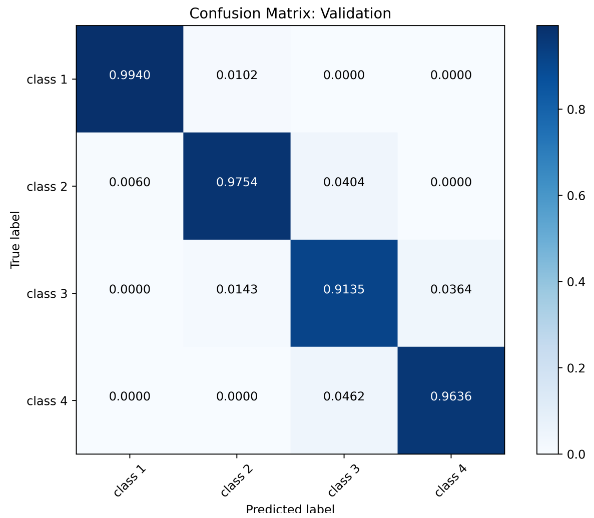


(b) The confusion matrix of the validation dataset for the Ecal dataset.

Figure 18: The confusion matrix and training history for the Ecal dataset. The maximum TPR was for the single electron events with 0.996 and the lowest was for 3 electron events with 0.7974



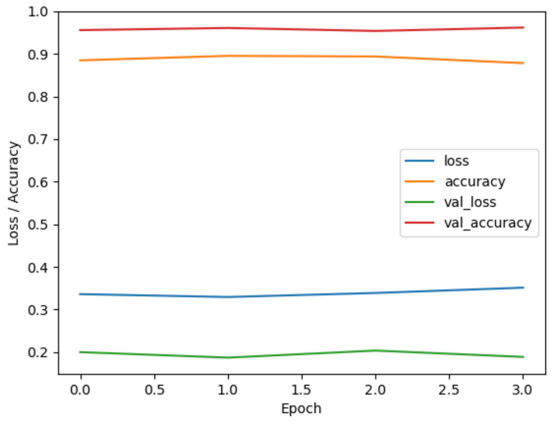
(a) The training history of the network training on the Ecal and Trigger Scintillator dataset over 15 epochs.



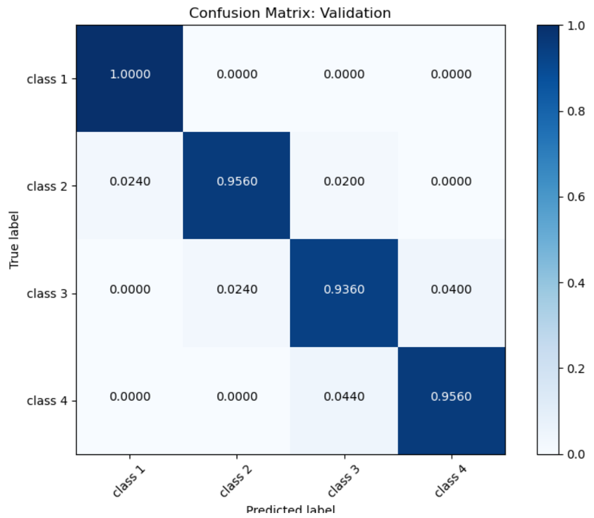
(b) The confusion matrix for the Ecal and Trigger Scintillator dataset.

Figure 19: The confusion matrix and training history for the Ecal and Trigger Scintillator dataset. The highest TPR was for the single electron events again with 0.994 while the lowest was for 3 electron events with 0.9135.

For the Ecal and Trigger Scintillator dataset two more convolutional layer was added with a BatchNormalization and MaxPooling layer added as well. The network now had 64,64,64,64, and 64 hidden units while the dense layer had 128. A dropout rate of 0.6 was once again used but the total number of parameter in this model is 150,212 with 640 non-trainable. The validation accuracy was found to be 0.961 with a validation log loss of 0.105. The highest TPR was for the single electron events again with 0.994 while the lowest was for 3 electron events with 0.9135 as seen in Figure 19 (b). The total FPR for this model was 0.117 which is higher than the Ecal model with a total FNR of 0.0665. The model is then skewed towards overcounting electrons in the events. The training was performed with 15 epochs and a learning rate of 0.005 and a minibatch size of 32. The model also used L1 regularization with a value of 10^{-4} and a dropout rate of 0.6. The training history can be seen in Figure 19 (a) which demonstrates a far more spiky training in terms of the validation loss and accuracy in comparison with the Ecal model. This suggests that the training was unstable which like the Ecal model training could have been controlled with more data for training or other regularizes. The model size here was 2.24 MB with an evaluation speed of 0.3707 seconds almost twice as fast as the Ecal model.



(a) The training history of the network training on the Ecal and modified Trigger Scintillator dataset over 15 epochs.

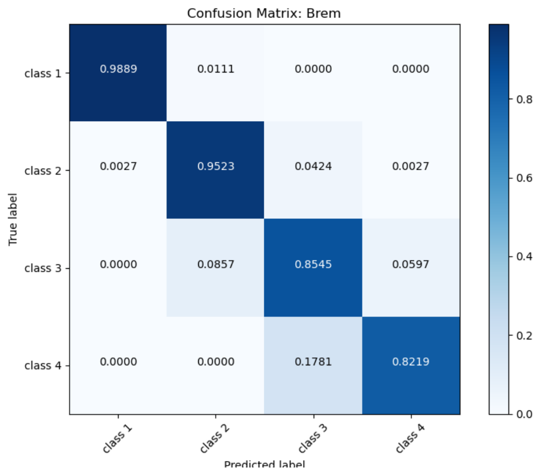


(b) The confusion matrix for the Ecal and modified Trigger Scintillator dataset.

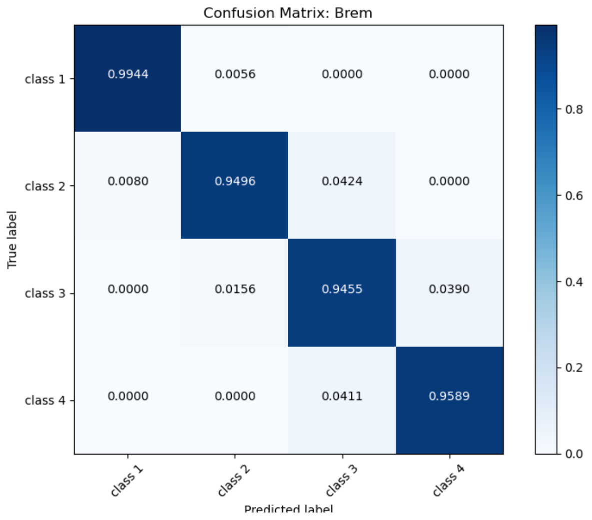
Figure 20: The training history and confusion matrix for the of Ecal and modified Trigger Scintillator dataset. The TPR maximum was for single electron events with 0.9766 with the lowest TPR for 3 electron events

For the Ecal and the modified Trigger Scintillator dataset the same number of convolutional layers and hidden units as for the Ecal and Trigger Scintillator model was used with the same number of trainable and non-trainable parameters. The validation accuracy was found to be 0.962 with a validation log loss of 0.144. The maximum TPR was for single electron events with 1.0 with the lowest TPR for 3 electron events with 0.9360 as can be seen in Figure 20 (b). The total FPR for this model was 0.06 much lower than both the other models. The FNR is also lower than the Ecal model with 0.092 but is slightly higher than for the Ecal and Trigger Scintillator model. A learning rate of 0.0005 was used in conjunction with a minibatch size of 16 and dropout rate of 0.4. The training was performed over 4 epochs and can be seen in Figure 20 (a). The training history for this model compared with the other CNN models is remarkably flat with no clear spikes. This suggests the model quickly learned the features of the model but the variation in validation performance was essentially oscillating around its current value. The model size was unchanged from the Ecal and Trigger Scintillator model at 2.24 MB with an evaluation speed of 0.2906 seconds faster than both of the other models.

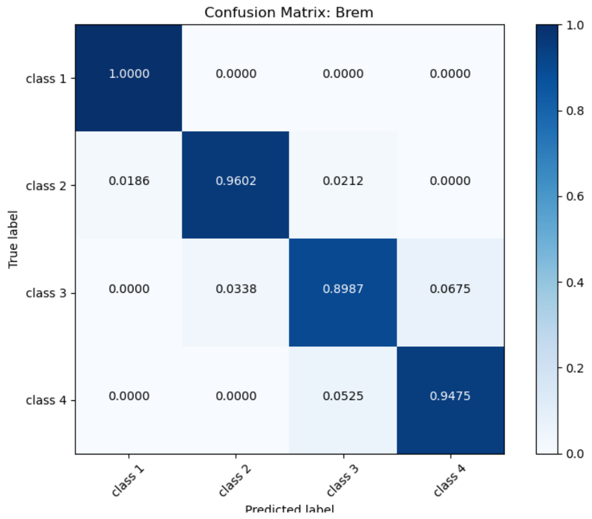
If we restrict the results to only the Brem events the corresponding confusion matrices can be seen in Figure 21. Overall the Ecal and Trigger Scintillator model had the highest validation accuracy of 0.9615 compared to 0.91 for the Ecal model and 0.9506 for the Ecal and modified Trigger Scintillator, respectively. This model also had the lowest total FPR with 0.0887 and total FNR at 0.0647. This also true for 3 and 4 electron events but the Ecal and modified Trigger Scintillator model has slightly higher accuracy for one electron and 2 electron events.



(a) The Brem confusion matrix for the Ecal dataset.



(b) The Brem confusion matrix for the Ecal and Trigger Scintillator dataset.



(c) The Brem confusion matrix for the Ecal and modified Trigger Scintillator dataset.

Figure 21: The confusion matrices of the different CNN models performance on only the Brem type of events in every dataset. The model with the highest accuracy was found to be the Ecal and Trigger Scintillator model with 0.9615. The model with the lowest total FPR was also the Ecal and Trigger Scintillator model with 0.087.

Table 1: The evaluation speed, model size, event size, validation accuracy, total FNR, and total FPR for the different data sets using the CNN architectures. ETS refers to the Ecal and Trigger Scintillator dataset and ETSX refers to the Ecal and modified Trigger Scintillator dataset.

Dataset	Evaluation Speed[s]	Model Size[MB]	Event Size[KB]	Accuracy	Total FNR	Total FPR
Ecal	0.761	8.89	2.1	0.9075	0.266	0.102
ETS	0.3702	2.24	2.2	0.961	0.0665	0.117
ETSX	0.2906	2.24	2.2	0.962	0.092	0.06

4.2 RNN

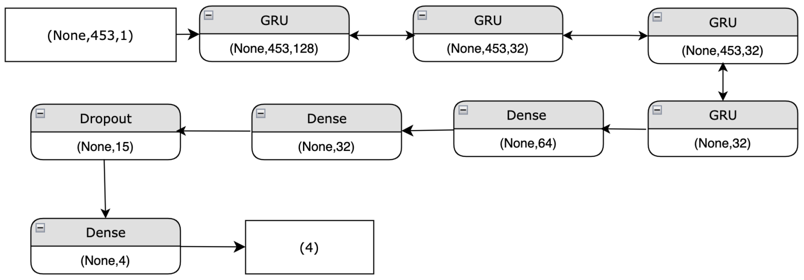
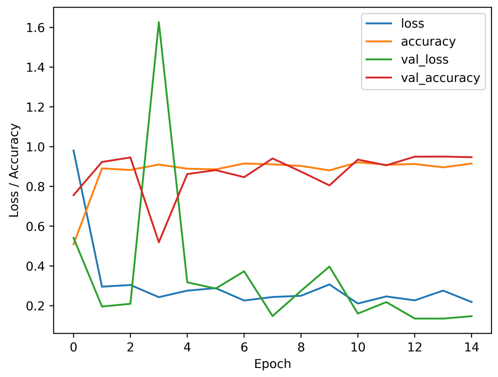
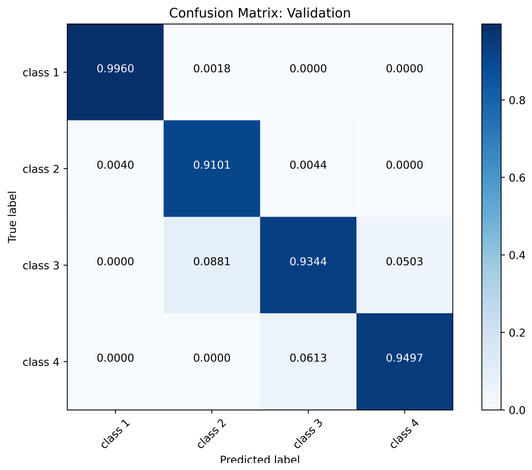


Figure 22: The RNN architecture that was used for the three different datasets. It consists of 4 GRU layers connected to two subsequent dense layers of fully interconnected nodes. The network finally outputs four probabilities, one for each type of class of electrons. The model used slightly different hyperparameters for each dataset but the overall structure remained the same.



(a) The training history of the RNN training with the Ecal dataset using 15 epochs.



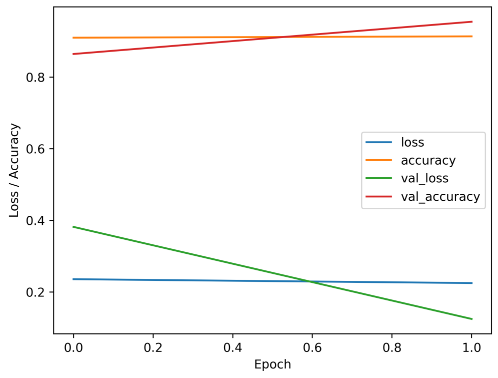
True label \ Predicted label	class 1	class 2	class 3	class 4
class 1	0.9960	0.0018	0.0000	0.0000
class 2	0.0040	0.9101	0.0044	0.0000
class 3	0.0000	0.0881	0.9344	0.0503
class 4	0.0000	0.0000	0.0613	0.9497

(b) The confusion matrix for the RNN trained on the Ecal dataset.

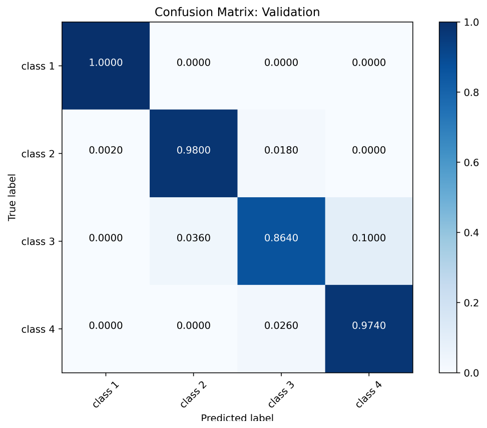
Figure 23: The training history of the RNN acting on the Ecal dataset and the confusion matrix for the validation data. The highest TPR is for the single electron events with 0.996 and the lowest for the two electron events with 0.9101.

The overall most successful RNN architecture can be seen in Figure 22. It consists of 4 GRU layers with 3 of the layers interconnected and one acting as an output layer. This is then followed by two dense layers with a final output layer. Dropout served to regulate the weights and avoid overfitting with the rate varied between each respective model. The network that was most successful for the Ecal dataset had 128, 32,32, and 32 hidden units for the GRU layers and 64 and 32 hidden units for the densely connected layers. The Ecal dataset had a maximum validation accuracy of 0.947 with a log loss of 0.1471. The maximum TPR was for the single electron events of 0.996 with the lowest for the 2 electron events with 0.9101. The total FPR of the model was 0.0565 with the largest FPR for 4 electron events. The total FNR of the model was 0.1534 suggesting that the model is skewed towards undercounting electrons. The training history and the confusion matrix can be seen in Figure 23 and during training 10 epochs were used with a learning rate of 0.0001. A dropout rate of 0.3 was used which had an impact of requiring a larger number of epochs. The validation accuracy and loss oscillate strongly over the epochs. Similar to the CNN models this could indicate that more data should be used in training or more regularizes to ensure the training of the network becomes less dependent on random chance to achieve a good result. Overall this model had 82,852 trainable parameters.

When saving the model it had an overall size of 4.1 MB with an event evaluation speed of 2.17 seconds as noted in Table 2.



(a) The training history of the RNN training with the Ecal and Trigger Scintillator combined dataset using 1 epoch.



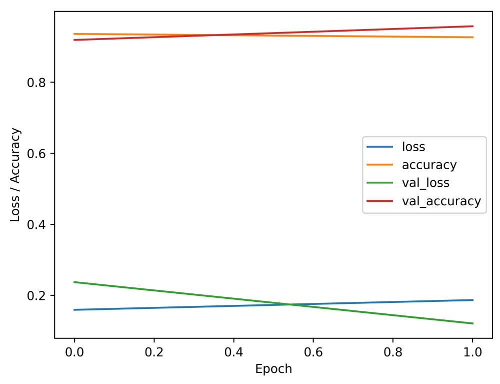
(b) The confusion matrix for the RNN trained on the Ecal and Trigger Scintillator dataset.

Figure 24: The training history of the RNN acting on the Ecal and Trigger Scintillator dataset and the confusion matrix for the validation data. The highest TPR is for the single electron events with 1.0 and the lowest is for the 3 electron events with 0.8640.

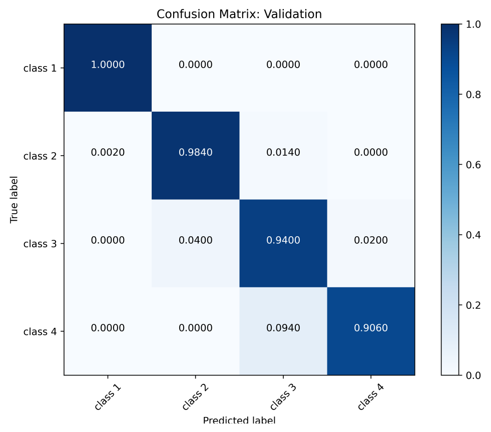
The network that was most successful on the Ecal and Trigger Scintillator dataset was found to be a model with 128, 64, 64, and 128 hidden units in the GRU layers but with otherwise an identical number of hidden units for the other layers. The dropout rate was increased compared to the Ecal model to 0.45 and the overall number of trainable parameters was found to be 144,868. The learning rate was set to 0.0005 and only 1 epoch was necessary for training with the training history and confusion matrix seen in Figure 24 (a). The maximum validation accuracy was found to be 0.9545 with the maximum TPR being found for the single electron events of 1.0. The lowest TPR was found for the 3 electron events with 0.8640. The total FPR was found to be 0.118, significantly higher than the Ecal model. The total FNR on the other hand was 0.064 which is much smaller than for the Ecal model but does suggest that this model tends towards overcounting electrons. The training history of the model compared to the Ecal model is very smooth only requiring two epochs to reach its current level of validation accuracy. It is unclear if any clear improvements could be made while training the model from the perspective of preventing oscillation or similar undesirable bahviour of the model. The overall model size was bigger than the Ecal model at 4.9 MB with an event evaluation speed of 2.023 seconds.

For the Ecal and modified Trigger Scintillator dataset the model with 128, 32, 32, and 128 hidden units in the GRU layers. The other layers were identical to the layers used for the Ecal and Trigger Scintillator model. The real difference between the model was that the learning rate was set to 0.0003 and the dropout rate was increased to 0.6. Two epochs were once again used as can be seen in Figure 25 (a) and is once again relatively smooth. Similarly to the Ecal and Trigger Scintillator model the training only required a small amount of epochs. Earlier attempts at training the model did demonstrate that the model oscillated between the current validation accuracy value so possible improvements would be to make the training more stable. The maximum validation accuracy was found to be 0.9575 with the highest TPR for the single electron events at 1.0 while the lowest was for 4 electron events at 0.906. Compared to the other models the total FPR was lower than the others at 0.034 but the total FNR was only lower than the Ecal model at 0.136. The model is then skewed towards undercounting electron especially compared to the other models. The model size was the same as for the previous model but the evaluation speed was far higher at 3.6320 seconds.

The models were once again tested on the Brem events in the validation dataset with the confusion matrices seen in Figure 26. The model with highest accuracy was the Ecal and modified Trigger Scintillator with 0.9551 while the Ecal model had an accuracy of 0.9436 while the Ecal and Trigger Scintillator model had an accuracy of 0.9487. The model had, however, the lowest TPR for the 4 electron events with 0.9087



(a) The training history of the RNN training with the Ecal and modified Trigger Scintillator dataset using 2 epochs.



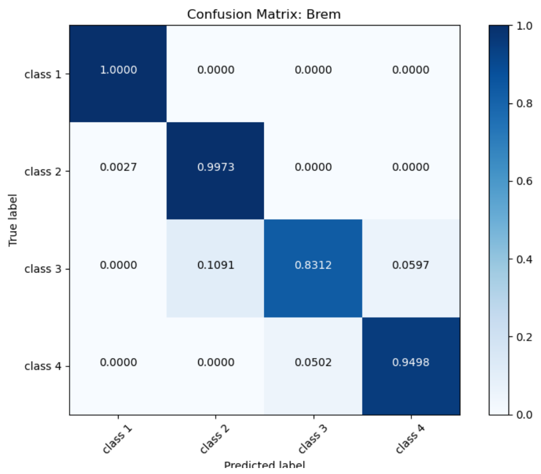
(b) The confusion matrix for the RNN trained on the Ecal and modified Trigger Scintillator with the extra granularity dataset.

Figure 25: The training history of the RNN acting on the Ecal and modified Trigger Scintillator with the extra granularity dataset and the confusion matrix for the validation data. The highest TPR is for the single electron events with 1.0 and the lowest is for the 4 electron events with 0.9060.

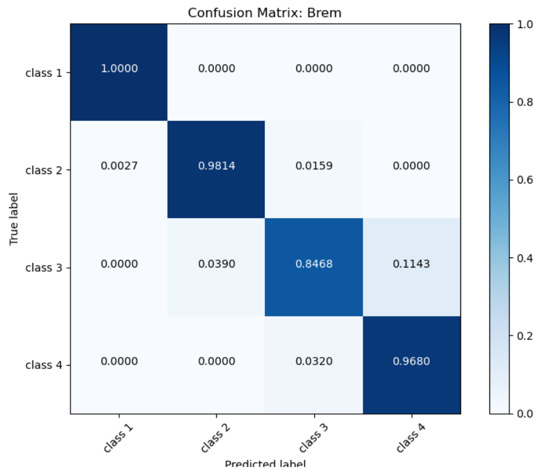
and only outperformed the other models for 3 electron events with a TPR of 0.9377. The model also had the lowest FPR of the three models while the Ecal and Trigger Scintillator model had a lower FNR.

Table 2: The evaluation speed, model size, event size, validation accuracy, total FNR, and total FPR for the different data sets using the RNN architectures. ETS refers to the Ecal and Trigger Scintillator dataset and ETSX refers to the Ecal and modified Trigger Scintillator dataset.

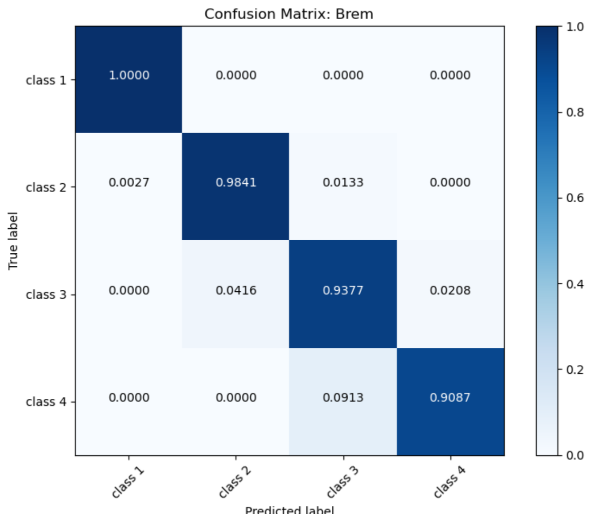
Dataset	Evaluation Speed[s]	Model Size[MB]	Event Size[KB]	Accuracy	Total FNR	Total FPR
Ecal	2.665	4.1	0.64	0.947	0.1534	0.0565
ETS	2.023	4.9	0.70	0.9545	0.06	0.118
ETSX	3.6320	4.9	0.70	0.9575	0.136	0.034



(a) The Brem confusion matrix for the Ecal dataset.



(b) The Brem confusion matrix for the Ecal and Trigger Scintillator dataset.



(c) The Brem confusion matrix for the Ecal and modified Trigger Scintillator dataset.

Figure 26: The confusion matrices of the different RNN models performance on only the Brem type of events in every dataset. The model with the highest validation accuracy was found to be the Ecal and modified Trigger Scintillator model with 0.9551. The model with the lowest total FPR was also the Ecal and modified Trigger Scintillator model with 0.0341 while the Ecal and Trigger Scintillator model had the lowest FNR at 0.0737.

4.3 GNN

Unfortunately, there was not enough time left in the project to properly optimize the GNN models. Although the actual training time for the GNNs was only around 5-6 hours the performance never rose above 0.45 in validation accuracy for either dataset for the GNN in time for the end of the project. The network architecture that was most successful is shown in Figure 27 and consists of three graph convolutional layers with 95 hidden units in each layer. This is then connected to a global mean pooling layer that aggregates the output from the previous layers. A dropout layer is placed in between this layer and the a dense layer with 32 hidden units. This is then passed to a final dense layer with 4 hidden units. Even though there was some initial success with this model, ultimately there was not enough time to properly train the model on the data or achieve a validation accuracy much better than guessing. All the models ended up oscillating in both training

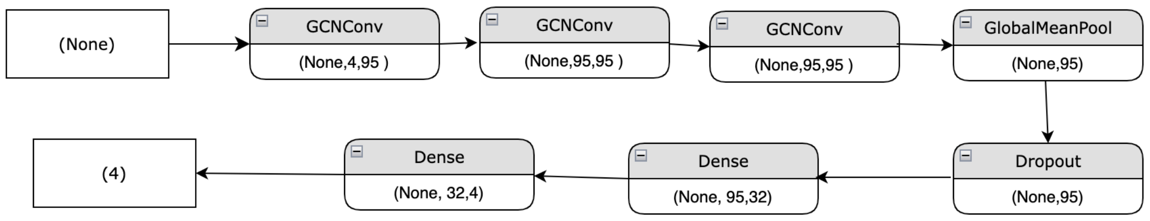


Figure 27: The network architecture that was attempted for the GNN. It consists of 3 graph convolutional layers with 95 hidden units in each. This is then passed through a global mean pooling layer that aggregates all the features from the convolutional layers until being passed to a dropout layer. This is then connected to two dense layer with 32 and 4 hidden units, respectively.

and validation accuracy but failed to converge in time for a proper result to be evaluated.

4.4 CNN and RNN

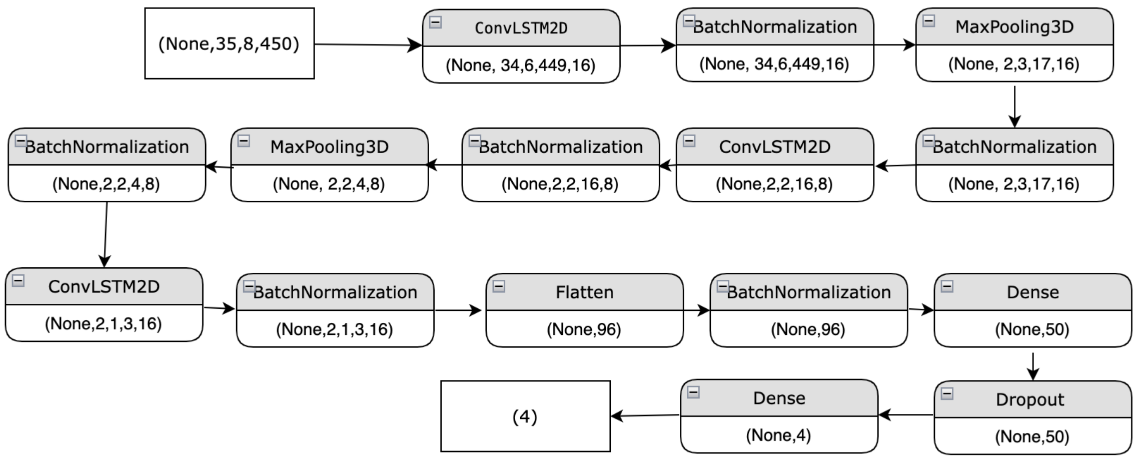
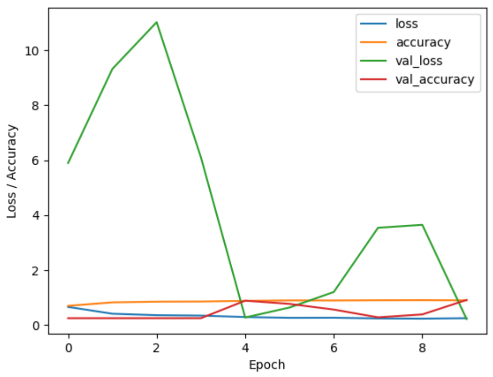
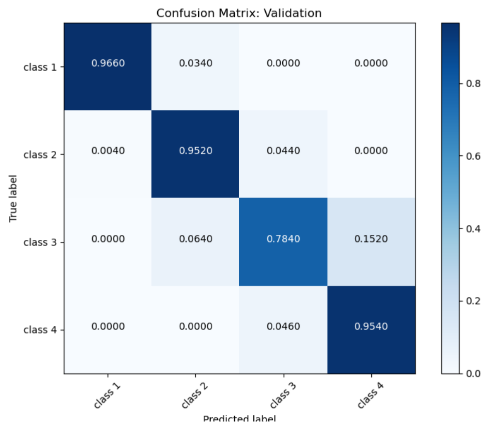


Figure 28: The network architecture that was attempted for the combined CNN and RNN. It consists of 3 convolutional-LSTM layers with MaxPooling between the the first and second and second and third layer. Batch Normalization follows each layer as well with a flatten layer after the convolutiona-LSTM layers. This is then followed by a dense layer and a dropout layer that connects to a final dense layer.

In comparison with the GNN models there was at least some success in training the combined CNN and RNN models. It was comparatively difficult to train to have a high validation accuracy so only the Ecal and Trigger Scintillator dataset was sucessfully trained in time. The most successful network architecture can be seen in Figure 28 which was for the Ecal and Trigger Scintillator dataset. This network consists of 3 convolutional-LSTM layers with MaxPooling after two of the layers with 16, 8, and 16 hidden units in each of the layers. The dense layer has 50 units while the final layer has 4 hidden units. The model has 8532 trainable and 128 non-trainable parameters with a dropout rate of 0.5. The learning rate used was 0.01 with a 64 minibatch size. Further more L1 and L2 regularizes were added to control the validation set performance with values 10^{-5} and 10^{-4}, respectively. The validation accuracy was 0.914 with a validation log loss of 0.2192. The highest TPR was for one electron events with 0.9660 while the lowest TPR was for 3 electron events at 0.7840 as can be seen in Figure 29 (b). The total FPR was found to be 0.23 while the total FNR was 0.114. The model is then severely skewed towards overcounting the number of electrons with the largest specific FPR for 4 electron events similar to the other models used for the CNN and RNN architechtures. The training history of the model as seen in Figure 29 (a) shows a that the validation loss spikes massively before decreasing and then increasing again. The validation accuracy behaves similarly remaining at 0.25 for several epochs before increasing until suddenly decreasing again. This suggests that the model was difficult to generalize from the training and no further adjustment of dropout or L1 and L2 values helped improve this performance. The model size was 1.72 MB with an evaluation speed of 0.882 seconds as can be seen in Table 3. The model performance on the Brem events can be seen in Figure 30 with an accuracy of 0.915. The highest TPR was for one electron events with 0.9639 while the lowest TPR was for 3 electron events with 0.787. The total FPR is 0.224 with a total FNR of 0.1162 suggesting that for Brem events the model is also skewed towards overcounting.



(a) The training history of the Ecal and Trigger Scintillator dataset using 10 epochs.



(b) The confusion matrix for the Ecal and Trigger Scintillator dataset.

Figure 29: The training history of the combined CNN and RNN acting on the Ecal and Trigger Scintillator dataset and the confusion matrix for the validation data. The highest TPR is for the single electron events with 0.966 and the lowest is for the three electron events with 0.7840.

An example training from each of the other datasets using the same model as for the Ecal and Trigger Scintillator dataset can be seen in Figure 31. Both of them demonstrate the same extreme variations in the validation loss as seen for the Ecal and Trigger Scintillator dataset. The Ecal dataset validation accuracy never breached 0.5 remaining 0.25 for several epochs before increasing. This suggests that more changes needed to be performed on the model to make it viable for this type of data even as the training accuracy increased to close to 0.95 each time the model was run. The Ecal and modified Trigger Scintillator dataset validation accuracy could rise above 0.8 on most iterations over the 10 epochs but was not stable. This is similar to the result on the Ecal and Trigger Scintillator dataset since the validation accuracy also oscillates except the training terminated at the right epoch for the model validation accuracy to remain high. The model seems more viable for this dataset than for the Ecal dataset suggesting less modification s might be necessary to the current model to receive a better performance.

Table 3: The evaluation speed, model size, event size, validation accuracy, total FNR, and total FPR for the different data sets using the combined CNN and RNN architecture. ETS refers to the Ecal and Trigger Scintillator dataset.

Dataset	Evaluation Speed[s]	Model Size[MB]	Event Size[KB]	Accuracy	Total FNR	Total FPR
ETS	0.882	1.72	2.2	0.914	0.114	0.23

4.5 Comparison

The performance of the models on the different datasets in terms of TPR for each class of event as well as FNR and FPR can be seen in Table 4. In terms of validation accuracy performance the most successful model was the CNN model for the Ecal and modified Trigger Scintillator dataset with 0.962. The highest

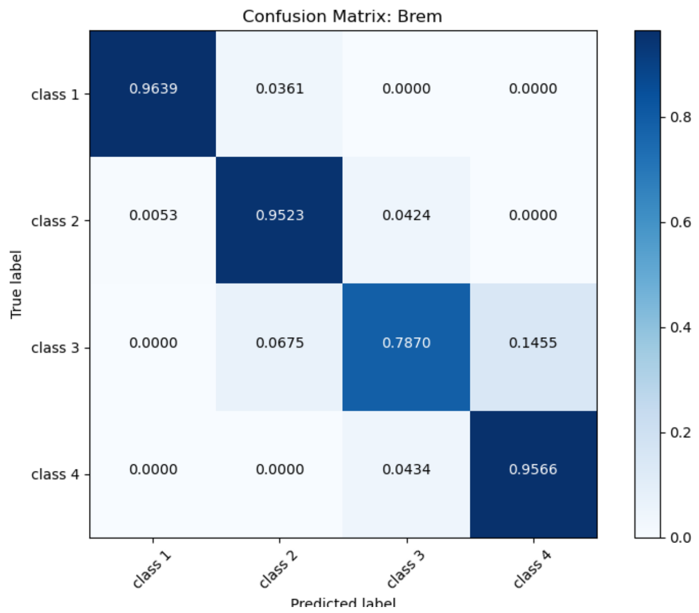
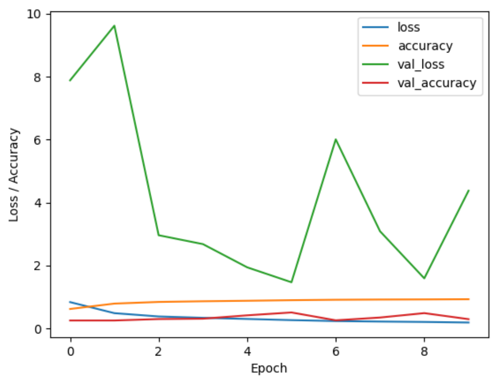
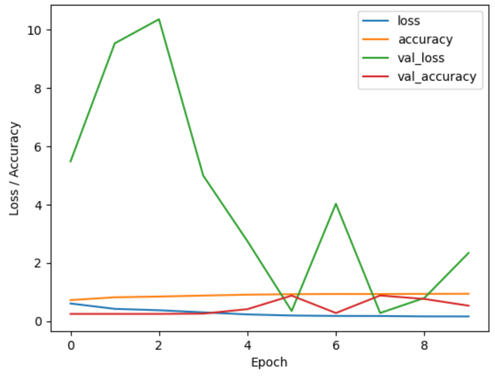


Figure 30: The confusion matrix for the combined CNN and RNN model acting on the Brem events in the Ecal and Trigger Scintillator dataset. The accuracy was found to be 0.915 with the highest TPR for one electron events with 0.9639 and the lowest for 3 electron events. The total FPR is 0.224 with a total FNR of 0.1162.





(a) The training history of the Ecal dataset using 10 epochs.

(b) The training history for the Ecal and modified Trigger Scintillator dataset using 10 epochs.

Figure 31: The training history of the combined CNN and RNN acting on the Ecal and the Ecal and modified Trigger Scintillator datasets over 10 epochs.

accuracy for the Brem events was however the Ecal and Trigger Scintillator CNN model. For 1 electron events the CNN and RNN models for the Ecal and modified Trigger Scintillator reached 1.0 while the RNN model for the Ecal and Trigger Scintillator also reached 1.0. For 2 electron events the RNN model for Ecal and modified Trigger Scintillator reached 0.984. The 3 electron events all had lower TPR than the other classes with highest value reached by the RNN for the Ecal and modified Trigger Scintillator with 0.940. The same holds for 4 electron events with the RNN model reaching 0.9784. The model with the lowest total FPR is the also the RNN model for the Ecal and modified Trigger Scintillator at 0.034. This pattern is however broken for the FNR which is lowest for the RNN model for the Ecal and Trigger Scintillator. In terms of evaluation speed the fastest model was the CNN model for the Ecal and modified Trigger Scintillator with a time of 0.2906 seconds. The smallest model size was reached for the combined CNN and RNN model with a size of 1.72 MB. None of these models, however, reached the necessary accuracy of 0.99999, but are any of these models still viable options for further training and refinement?

Table 4: The TPR for each type of network with respect to the 4 classes of electron events as well as the accuracy and the total FNR and FPR. ETS refers to the Ecal and Trigger Scintillator dataset and ETSX refers to the Ecal and modified Trigger Scintillator dataset.

Dataset	1 Electron	2 Electron	3 Electron	4 Electron	FNR	FPR	Accuracy
CNN[E]	0.990	0.9540	0.8660	0.820	0.266	0.102	0.9075
CNN[ETS]	0.9940	0.9754	0.9135	0.9636	0.0665	0.117	0.961
CNN[ETSX]	1.0	0.9560	0.9360	0.9560	0.09	0.06	0.962
RNN[E]	0.9960	0.9101	0.9344	0.9497	0.1534	0.0565	0.947
RNN[ETS]	1.0	0.98	0.8640	0.9740	0.06	0.118	0.9545
RNN[ETSX]	1.0	0.9840	0.940	0.9784	0.136	0.034	0.9575
GNN[ETS]	NAN	NAN	NAN	NAN	NAN	NAN	NAN
GNN[ETSX]	NAN	NAN	NAN	NAN	NAN	NAN	NAN
GNN[E]	NAN	NAN	NAN	NAN	NAN	NAN	NAN
CRN[E]	NAN	NAN	NAN	NAN	NAN	NAN	NAN
CRN[ETS]	0.966	0.952	0.7840	0.9540	0.1162	0.224	0.914
CRN[ETSX]	NAN	NAN	NAN	NAN	NAN	NAN	NAN

5 Discussion

5.1 Viability of the Models

Even though the goal of finding a model that could reach the accuracy goal of 0.99999 remains unfulfilled, each type of model considered so far has its own strengths and weaknesses with large variation between respective performance measurements. Examining the CNN models the model that was trained with the Ecal and modified Trigger Scintillator data had the overall highest accuracy and lowest evaluation speeds. However, this model remains the best option from the CNN type models because its total FPR is the lowest with a higher total FNR meaning it skews slightly towards undercounting events. This is one of the desired features of an electron counting algorithm in order to avoid flooding the detector systems with non-relevant data. The algorithm is, however, slightly worse at dealing with Brem events compared with the model for the Ecal and Trigger Scintillator data as seen in Figure 21 but this could possibly be accounted for by training with more data. For the RNN type of models the model for the Ecal and modified Trigger Scintillator also had the highest accuracy even if the difference between the other model for the Ecal and Trigger Scintillator is just 0.03. It also has the smallest amount of overcounting as well with the smallest total FPR but with a large FNR instead. Again this is the more desirable situation to have with an algorithm that undercounts rather than overcounts. The same performance holds for the Brem events as well as seen in Figure 26 with the. The GNN types of models unfortunately cannot be evaluated due to there not being enough time to perform the proper fine tuning of hyperparameters and model architecture that could have pushed the models to at least perform closer to the desired accuracy performance. The combined CNN and RNN although difficult to train properly in terms of validation performance still had promising results reaching 0.914 in accuracy even with only a small amount of fine tuning. The problem with this model for the Ecal and Trigger Scintillator dataset is that the total FPR is 0.23 compared with a total FNR of 0.114 which means the system will be prone to overcounting. This is something that more training time and fine tuning might have been able to reduce but until more research is performed remains an area of concern. The models were then not able to perform to the level that would be required in order to be implemented in the detector, but their performance does suggest that the approach of using Machine Learning to create new electron counting algorithms could yield effective new models with more time and research. Based on this limited study no verdict can be passed on the which type of dataset is better for these dataset to train with, but there seems to be a small increase in overall performance with the added granularity to the Trigger Scintillator. This might of course just be a product of the limited amount of training that was done and might disappear with more training but the possibility of increasing the performance by expanding the Trigger Scintillator capability remains a tantalizing possibility for future research. Ultimately the CNN and RNN models for the Ecal and modified Trigger Scintillator then show the most promise but more research needs to be conducted using the GNN and combined CNN and RNN before any clear verdict can be had on what type of ANN is approriate for electron counting.

5.2 Computational Considerations

There are of course more considerations than just the performance of these models. One important aspect to consider is the computational times of these models when applied to a typical event. The result from looking at an average of 1000 evaluations for each model is seen as the evaluation speed of the models as in Table 1. The fastest speed of evaluation was for the CNN model acting on the Ecal and modified Trigger Scintillator with a time of 0.2906 seconds. Depending on the technical specifications of the trigger where this can be implemented this is something that would have to be taken into account in future research to ensure that the model is fast enough on the hardware to be effectively used. For some models performance might have to be sacrificed in order to make the processing more efficient. Similar concerns exist for the size of the models as the smallest network was the combined CNN and RNN model with 1.72 MB but if there are memory limits that need to be set in the detector this might be too large and require pruning the model of variables to make it more viable. Picking the right model for the trigger will then not only rely on which has the highest performance but which model has the highest likelihood of being a good fit for the trigger and its technical specifications.

5.3 Error and Improvement

Since none of the models achieved the validation accuracy that would be required there are of course multiple improvements that could be implemented. The clearest on is the use of more data. We unfortunately only had time to use 1.5 million events when around 10 million had been prepared. Actually having time to properly train the GNN and combined CNN and RNN would have possibly led to models with higher performance or at least would have more excluded these types of models as appropriate for this problem. Based on the training histories seen in for example Figure 19, the training was not smooth for most models with the exception of the RNN models for the two Trigger Scintillator datasets that are remarkably smooth. Every single model in training did, however, demonstrate that the validation performance such as the validation accuracy oscillates around certain values without improving steadily or decreasing steadily. Introducing more data for the network to train on could have allowed the network to increase its generalization by picking up more features instead of relying on techniques such as dropout that can make the training a bit more unstable. More epochs could also allow for a more complete picture of how to attempt to increase the performance of the model in training but unfortunately there was nit enough time to perform longer runs of each model.

Training the models with the full range of data would plausibly result in the model being more efficient at handling specific situations and outliers that may not have been captured in the smaller dataset. This might then also expose if the models truly were viable or if they were simply appropriate for this subset of data. Improving the generalization of the model by training with more data would have a large impact on making the model more viable and possibly avoid problematic features of the networks such as overcounting that would disfavour them even if their relative performance might be higher for certain multielectron events than the current electron counting algorithm. Further study could also have been done on how exactly the angle of incidence of the electrons on the Ecal affected the predictive capabilities of the models. Of course, with more data there is a chance that the model might adapt to the variation through training but it would still be instructive to determine how this feature affects a prediction. Very specific event types such as events with large amounts of Brem photons or multiple electron that pass a veto could also have been generated. Training on these events could have benefits in causing more exotic features to be picked out by the network which could have unforeseen benefits on the performance of the model. The data generated for this thesis is a mix of types and so by looking at datasets composed of just one type there could be some benefits with regards to understanding the weaknesses and strengths of the models. Something that needs further research is also how these types of networks can actually be implemented in the trigger and if some types of networks are more viable than others. For example, is there a type of data output that would be more efficient to use than others? This could then mean that networks such as for example pure CNN might not be viable and require the exploration of other avenues. A firmware implementation of an RNN for the computation of energy deposited at the ATLAS experiment [30] in 2023 demonstrates that the question of implementation is starting to move from an if to a when for some large detectors implying an influx of interest in the topic to come.

One significant part of the performance of these models is how the data was preprocessed. For the RNNs the data was processed with a formula that combined the layer, module, cell, and energy into one number. Each event then becomes a list of sequential data points that the RNN can attempt to process. Of course the exact form of this equation is just one attempt to combine the data from each hit into one value and there could be many different variations of this type of formula that could have vastly different outcomes on the performance of the model. For example, when initially normalizing the data the lower values were so suppressed with this formula that they were rounded to zero meaning that some events included almost nothing but zero which made classification essentially impossible for these events. The same situation could then be happening when using these formulas where other representations could have a far more distinct set of points.

When training the CNN and the RNN and CNN combined models the array used was of the form (34,7,450) or (35,8,450). This irregular shape proved to be cumbersome when dealing with the convolutional layers as it meant having to use kernel shapes that were also irregular which made finding the right shape more difficult. The number of module values meant that the shape for each layer had to respect this particular shape which hindered the type of kernel sizes possible. One avenue of research would then be to see if it would be worth adding extra dimensions to the array that, although they might never be filled with any data, could pad the shape of the array to be easier for the networks to interpret. A possible complication with this would be that if there are any average pooling operations expanding the array could mean that the values of the pooling operations are reduced possibly making feature detection harder not easier.

In terms of the graphs the module, layer, cell, and energy were used as labels of each node. There could be benefits in adding for example the amplitude of the hit to the available labels of each node. The edges are also undirected meaning there is no clear progression from each hit in the graph. A possible benefit would then be to add directed graphs between sequential nodes such that the evolution of the event is represented more directly in the structure of the graph. This could then help more distinguishing features to each graph. At the moment the edges are based of geometric radius around each node. The parameter was set at 40 which seemed to provide a good coverage of the connectivity of the nodes but could be fine tuned to be more optimal and only allow the closest nodes to connect to each other. All of these approaches could potentially have an impact on the performance of the networks and are worth exploring in a more long term study.

5.4 Understanding ANNs

An ANN is by its nature a black box. Information is passed to the ANN and it produces some sort of output based on an elaborate set of mathematical instructions we have encoded in it. The exact features and details that the networks manage to pick up remain unknown unless testing is performed on the model to try and determine what the weights correspond to in terms of features. This can be a time consuming process that requires a lot of time and effort for possibly little return. So how can we be certain that the networks are not just inventing patterns for these types of events in order to satisfy the performance goals? During an initial attempt at training the network the data was saved in a way that introduced numerical errors that were propagated throughout when training. This meant that there was no longer a clear distinction between the different types of events and instead the networks were trying to learn patterns from essentially noise. The networks ended up failing to accomplish this as the integrity of the data was compromised: there were no longer any clear patterns to distinguish. Once this was fixed the ANNs could learn from the data again as the patterns were brought back instead of just being noise. An ANN then follows the fundamental nature of most computation: input equals output. The better the data the better the corresponding model.

Understanding an ANN can then often boil down to instead understanding the principles of the model that was used to discern the features. For example, understanding how a CNN performs its operations and relies on filters and then seeing what features the filters are detecting is often easier than attempting to comprehend how all the connections and updates were made under training and what each weight corresponds to. Using

ANNs to design models then requires a willingness to give up immediate understanding of the model. Training the model can then be seen as the first step of designing models using ANNs and actually understanding how they work and determining what features they are detecting is the next and much more difficult step.

5.5 Why Machine Learning?

It might be questioned why Machine Learning is the right approach for this type of problem to begin with. When ANNs are used, the physical intuition about a problem can quickly be lost as the data is transformed and processed. The current approach for designing the algorithms that identify the number of electrons in the detector at LDMX relies on the physics of the electrons in the detector and can be justified based on known theories and techniques. With clear goals such as preventing overcounting there is a lot of control and direct input by a human researcher. However, when considering ANNs the algorithms that are produced no longer rely on any clear basis in physics but strip the problem down to just handling data points: the ANN does not care whether or not the data is produced in a nuclear reactor or from a social media platform, data is data to an ANN. Is relying on an ANN to design an algorithm then akin to using a hammer on a screw and hoping for the best? This begs the question if an ANN is then appropriate for dealing with problems where there is more information available for the problem but that is not possible to interpret from the data. For example, when the CNN was trained the network does not make any reference to the fact that these data points were generated by electrons or photons, the data points are indistinguishable beyond the coordinate and energy value of the hits. There is then a possibility that by removing the majority of information around the problem the ANN is reduced to finding relations between a seemingly arbitrary collection of points. Of course this is not the entire story of what is happening in the problem. The ANN can only train on the data that is passed to it so the degree of sophistication depends on the researcher implementing it. The full degree of physical information could have been passed to a network it would just have to be treated in some way that the network can interpret the data and process it. However, in doing so the real world applications of the network might be reduced since if an algorithm based on the network is to be implemented all the information available when training the network might not be available when receiving the output from the detector. At LDMX, when the data from the Ecal and the Trigger Scinitillator is passed for analysis the location of the hits in terms of cell, module, and layer is available from the event. A benefit of the models considered for this thesis is that they at least avoid using information that might not be available from the detector itself but from further analysis. An earlier version relied on using Cartesian coordinates and while it was easier to design a network for due it being more symmetric in its array shape for the CNN models, it is partway unrealistic and missed the goal of seeing what type of models are viable options for counting electrons.

A problem might arise when considering more standard physics problems such as for example dealing trajectories where the network might come up with a series of solutions that are outside physical reality and therefore must be rejected. Machine Learning in physics can then play the role of being an iterator, throwing out possible solutions or algorithms to deal with problems that can take away the busywork of designing systems with optimal tolerances. This, however, does not mean that physicists should trust the results implicitly but should compare the results to physical reality and make sure that the results that are being produced are reasonable. The goal of this thesis was mainly to explore the feasibility of using ANNs for electron counting. Based on the training using simulated data, that is of course much cleaner than what the actual data would be, it certainly seems like a viable option that should be considered as a tool for generating algorithms for this type of task. Machine Learning is then not a tool that should be wielded indiscriminately across physics: a screwdriver is still best for dealing with a screw. However, it should be considered a viable tool that can help in the design or conceptualizing of a solution to a problem where a lack physical intuition might be hampering efforts, something that can lay down a basis that other people can use to approach the problem from a better vantage point. The future of Machine Learning and ANNs in physics remains bright as a useful tool rather than catch-all solution.

6 Conclusion

This thesis attempted to explore whether Machine learning in the form of ANNs could be a viable tool for designing new algorithms for counting electrons passing through the detector. Although none of the models were ultimately successful in reaching the required accuracy of 0.99999 the CNN, RNN, and combined CNN and RNN models at least demonstrated that using the data from the simulated events was a viable method for training ANNs. The three different datasets were all successfully trained on for the CNN and RNN type models and a slight increase in performance was seen when using the modified Trigger Scintillator for the dataset. The RNN and CNN models for the Ecal and modified Trigger Scintitllator both demonstrated a tendency towards under counting events which is important to ensure non-relevant events are not saved on trigger level. Combining the Ecal data with the modified Trigger Scintillator and an ANN could then provide a method for designing a new trigger for multielectron events.

Even if there ultimately was no time to properly train the GNN model these types of models should not be dismissed but more research is necessary to determine whether or not they can be viable candidate models for the trigger. The same situation holds for the combination of CNN and RNN even if there was some results for the Ecal and Trigger Scintillator dataset that indicates the feasibility of training this type of network. The exact implementation of the data could also have been performed differently which could have had significant effect on the final outcome of the training. In conjunction with training on all of the available data there could be many methods for increasing the performance of these models and make models such as the GNN viable candidates as well.

ANNs can then be a powerful tool for counting electrons at the trigger level in the LDMX detector. With further research with regards to the optimization of these models necessary to determine if they can ever replace more traditional methods they nevertheless demonstrate the promise of ANNs as a tool for physics research. Taking the leap into using ANNs at LDMX should not be a question of if but when. Many areas of research remain unexplored but ANNs have proven themselves to at least be possibly effective tools even if the models remain unrefined.

References

[1] G. Bertone and D. Hooper, "History of dark matter," *Reviews of Modern Physics*, vol. 90, no. 4, Oct. 2018. DOI: 10.1103/revmodphys.90.045002. [Online]. Available: https://doi.org/10.1103/RevModPhys.90.045002.

[2] K. Lundmark, "Über die Bestimmung der Entfernungen, Dimensionen, Massen und Dichtigkeit fur die nächstgelegenen anagalacktischen Sternsysteme.," *Meddelanden fran Lunds Astronomiska Observatorium Serie I*, vol. 125, pp. 1–13, Jan. 1930.

[3] V. A. Rubakov and D. S. Gorbunov, *Introduction to the Theory of the Early Universe: Hot Big Bang Theory 2nd Edition*. World Scientific, 2018.

[4] M.Markevitch, *The bullet cluster*. [Online]. Available: %5Curl%7Bhttps://www.esa.int/ESA_Multimedia/Images/2007/07/The_Bullet_Cluster2%7D.

[5] B. W. Lee and S. Weinberg, "Cosmological lower bound on heavy-neutrino masses," *Phys. Rev. Lett.*, vol. 39, pp. 165–168, 4 Jul. 1977. DOI: 10.1103/PhysRevLett.39.165. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.39.165.

[6] T. Åkesson, N. Blinov, L. Bryngemark, *et al.*, *A high efficiency photon veto for the light dark matter experiment*, 2019. arXiv: 1912.05535 [physics.ins-det].

[7] R. Essig, J. A. Jaros, W. Wester, *et al.*, *Dark sectors and new, light, weakly-coupled particles*, 2013. arXiv: 1311.0029 [hep-ph].

[8] T. Åkesson, N. Blinov, L. Brand-Baugher, *et al.*, *Current status and future prospects for the light dark matter experiment*, 2022. arXiv: 2203.08192 [hep-ex].

[9] T. Åkesson, A. Berlin, N. Blinov, *et al.*, *Light dark matter experiment (ldmx)*, 2018. arXiv: 1808.05219 [hep-ex].

[10] L. K. Bryngemark, Personal conversation, Apr. 2023.

[11] M. Ohlsson and P. Eden, *Introduction to Artificial Neural Networks and Deep Learning: Lecture Notes*. Lund University, 2022.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] Y. LeCun, B. Boser, J. Denker, *et al.*, "Handwritten digit recognition with a back-propagation network," *Advances in Neural Information Processing Systems*, 1989.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Literate programming," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[16] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[17] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 20, no. 1, pp. 61–80, 2009.

[18] C. Morris, M. Ritzert, M. Fey, *et al.*, "Weisfeiler and leman go neural: Higher-order graph neural networks," *CoRR*, vol. abs/1810.02244, 2018. arXiv: 1810.02244. [Online]. Available: http://arxiv.org/abs/1810.02244.

[19] A. Tasdelen and B. Sen, "A hybrid CNN-LSTM model for pre-miRNA classification," *Scientific Reports*, vol. 11, no. 1, p. 14 125, 2021. DOI: 10.1038/s41598-021-93656-0.

[20] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3367–3375. DOI: 10.1109/CVPR.2015.7298958.

[21] S. Agostinelli, J. Allison, K. Amako, *et al.*, "Geant4—a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, pp. 250–303, 2003. DOI: `https://doi.org/10.1016/S0168-9002(03)01368-8`.

[22] L. K. Bryngemark, D. Cameron, V. Dutta, *et al.*, "Building a distributed computing system for LDMX," *EPJ Web of Conferences*, vol. 251, C. Biscarat, S. Campana, B. Hegner, S. Roiser, C. Rovelli, and G. Stewart, Eds., p. 02 038, 2021. DOI: `10.1051/epjconf/202125102038`. [Online]. Available: `https://doi.org/10.1051%2Fepjconf%2F202125102038`.

[23] R. Brun, F. Rademakers, P. Canal, *et al.*, *Root-project/root: V6.18/02*, version v6-20-06, Aug. 2019. DOI: `10.5281/zenodo.3895855`. [Online]. Available: `https://doi.org/10.5281/zenodo.3895855`.

[24] L. Östman, "Imaging using machine learning for the ldmx electromagnetic calorimeter," M.S. thesis, Lund University, Lund, May 2020.

[25] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[26] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[27] M. Fey and J. E. Lenssen, *Fast graph representation learning with pytorch geometric*, 2019. [Online]. Available: `%5Curl%7Bhttps://github.com/pyg-team/pytorch_geometric%7D`.

[28] J. Lindahl, *LMDX Machine Learning Project*. [Online]. Available: `https://github.com/Udcstb99/LDMXML.git`.

[29] P. Gavrikov, *Visualkeras*, `https://github.com/paulgavrikov/visualkeras`, 2020.

[30] G. Aad, T. Calvet, N. Chiedde, *et al.*, *Firmware implementation of a recurrent neural network for the computation of the energy deposited in the liquid argon calorimeter of the atlas experiment*, 2023. arXiv: `2302.07555 [physics.ins-det]`.