Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

KM3NeT Neutrino Detection using Deep Learning

Author: Arumoy Shome (2636393)

1st supervisor:	Dr. Adam Belloum	
2nd supervisor:	Dr. Ben van Werkhoven	(Netherlands eScience Center)
2nd reader:	Dr. FINDME	

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

October 16, 2020

Abstract

Neutrinos are highly elusive subatomic particles which can only be detected with the help of large neutrino detectors. The KM3NeT neutrino telescope is one such detector currently being constructed at the bottom on the Mediterranean Sea. Due to its large volume and the presence of background noise, "event trigger" algorithms are utilized by the data acquisition pipeline of the detector to sift through the noise. Although the state of the art event trigger algorithms are able to perform data filtration in real time, the quality of filtration is poor. A GPU Pipeline was developed to improve the quality of filtration without compromising the runtime performance. The goal of this paper is to improve upon this GPU Pipeline using Artificial Neural Networks. The paper explores the possibility of replacing parts of the GPU Pipeline using Multi Layer Perceptrons and Graph Convolusional Neural Networks. The results are promising and new avenues of research are discovered through this work.

Keywords*: Neutrino detection, Deep learning, Graph Neural Networks, Geometric Learning, KM3NeT.*

Acknowledgements

I would like to thank Dr. Adam Belloum and Dr. Ben van Werkhoven for introducing me to the fascinating field of neutrino detection and giving me the opportunity to be a part of the KM3NeT research initiative in collaboration with Nikhef and The Netherlands eScience Center. Many thanks to Dr. Roel Aaij, Dr. Ronald Bruijn and Brian O Fearraigh for their technical support on the physics side of things. Finally my gratitude and best wishes to my colleague Shruti Rao who is also defending her thesis in the coming months.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

Perhaps the most elusive subatomic particle known to science is the Neutrino. With no electrical charge (thus aptly named) and mass smaller than any other elementary particles, neutrinos pass through matter making them virtually undetectable. Neutrinos are sought after by researchers, especially those in the fields of Astrophysics and Astronomy since they may help gain insights into astronomical events such as the birth of a neutrino star or a supernova.

When neutrinos experience a change in the density of the matter they are traveling through (such as going from air into water), they experience a change in velocity and emit an electron and a photon. This phenomenon is known as Cherenkov Radiation (1) and is an indirect method which can be used to detect neutrinos. This remains the premise for some of the worlds largest neutrino observatories built to date such as the Sudbury Neutrino Observatory (Ontario, Canada), The Super-Kamiokande (Gifu Prefecture, Japan) and The IceCube Neutrino Observatory (Antarctica).

The KM3NeT or the Cubic Kilometer Neutrino Telescope is the next generation neutrino telescope, currently being constructed at the bottom of the Mediterranean Sea. The goal of this research infrastructure is two fold. First, to study high energy neutrinos originating from celestial events in the galaxy. And second, to study the properties of the neutrino particles produced in the Earth's atmosphere (2). The first goal will be realized with the KM3NeT/ARCA (Astroparticle Research with Cosmics in the Abyss) telescope and the second with KM3NeT/ORCA (Oscillation Research with Cosmics in the Abyss) (2). This paper talks exclusively about KM3NeT/ARCA.

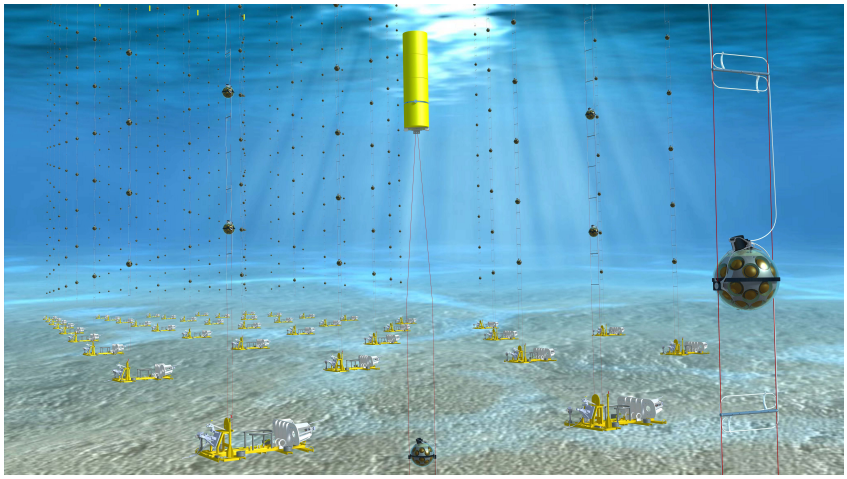

Figure 1.1: Artist's impression of the ARCA detector *source: https://www.km3net.org*

Figure 1.2: An optical detector (DOM) *source: https://www.km3net.org*

1.1 Situation of Concern

The ARCA telescope comprises of two "blocks" with a total volume of $1km^3$. Each block consists of 115 spherical detector units (DOMs) and each DOM consists of 31 Photo Multiplier Tubes (PMTs) in various spatial arrangement. Figure 1.1 shows an artist's impression of ARCA and figure 1.2 depicts a DOM along with the PMTs inside it. The PMTs are highly sensitive to light (photons), and thus are used to detect the Cherenkov Radiation emitted from the neutrino particle interactions. All hits are recorded by the PMTs in the form of analog signals and those above a certain threshold are digitized. The digital signals from all PMTs are arranged in 100ms "timeslices" and sent to the on-shore facility for further processing (3).

Unfortunately, there are several sources of noise like bioluminescense, decay of Potassium 40 (^{40}K) and atmospheric Muons (4). Due to the high level of noise, data is generated at an extremely high rate of 25GB/sec (2) and must be filtered and selectively stored for further analysis. The state of the art for this task are known as "Event Trigger" algorithms (2, 3) which can filter timeslices containing just noise thus only allowing important timeslices containing neutrino hits to pass through. The existing event trigger algorithms namely *L1* and *L2* although able to conduct the filtration in near real time, lack the ability to do so with high accuracy thus often failing to save important timeslices (5). Efforts have already been made to improve the existing event trigger algorithms. Karas et al. (2019) proposed and implemented a GPU powered pipeline which utilizes correlation and graph community detection to identify time slices that may contain neutrino hits whilst Post et al. (2019)

suggest an alternate using convolutional neural networks.

1.2 User Requirements

The primary users of ARCA are researchers who want to study high energy particles from outer space. The stakeholders are all member institutes involved in the project and by extension all scientists from these institutes who will be working with the data collected. The requirements of the primary users and stakeholders with respect to the data acquisition pipeline are as follows.

UR1. **The accuracy of filtration must be extremely high.**

Time slices which are deemed important by event trigger algorithms are stored for further analysis and research. Failure to store timeslices containing information from neutrino events can lead to loss of important data and hinder new discovery. Since majority of the data generated is noise, the pipeline must be able to prevent storage of unnecessary timeslices containing only noise in the on-shore facility.

UR2. **Filtration should occur in real time.**

The state of the art event trigger algorithms are able to process data in real time. The proposed alternative should be able to provide better data filtration quality whilst maintaining or improving upon its predecessor's runtime performance.

1.3 Research Question

This report presents research to improve upon the GPU Pipeline proposed by Karas et al. (2019) to combat the limitations of the L1 and L2 event trigger algorithms. Specifically this project seeks to answer the following research questions.

RQ1. **Can the existing GPU pipeline be improved using Neural Networks?**

Improvement may be achieved by reducing the processing time of the pipeline or improving the accuracy with which relevant timeslices are identified. This project focuses on achieving improvement via accuracy. The task of validating the runtime performance of the methods proposed in this paper is left to a separate project. In order to answer **RQ1**, the following additional questions are formulated.

RQ2. Can the *Hit Correlation Step* be replaced with a Multi Layer Perceptron?

The first step of the GPU Pipeline is the Hit Correlation Step. A novel trigger criterion was proposed which when given a pair of points, can quantify their level of correlation with an accuracy of 80%. The first phase of this project focuses on exploring an improvement to this trigger criterion using a Multi Layer Perceptron (MLP).

RQ3. Can the *Graph Community Detection Step* be replaced with a Graph Convolusional Neural Network?

The output of the Hit Correlation Step is used to create a graph structure where hits from neutrino and noise are represented as nodes connected with an undirected edge carrying the probability of correlation as its weight. The Constant Potts clustering algorithm, which operates on the principles of Graph Community Detection (6), was used to separate the graph into communities of neutrino and noise hits. The second phase of this project focuses on achieving a better accuracy for clustering neutrino and noise hits into separate communities in a given timeslice using Graph Convolusional Neural Networks (7).

The following chapters present the research efforts carried out to improve The GPU Pipeline using Artificial Neural Networks. In Chapter 2, the steps taken to prepare the dataset is presented followed by its statistical analysis and visual exploration. Since this research initiative is based on the seminal work conducted by Karas et al. (2019), an overview of the GPU Pipeline is presented in Chapter 3 in addition to other related projects. Chapters 4 and 5 present detailed analysis of the neural networks created to replace segments of The GPU Pipeline. Drawing from the results of the replacement models, practical recommendations and directions for further research are laid out in Chapter 6.

This report is intended for the primary users of ARCA with the hope to aid in the development of the successor to the state of the art event trigger algorithms. The report may also be used by deep learning practitioners working in the field of neutrino detection. This report assumes the reader posses a background in Computer Science or Artificial Intelligence and thus is familiar with concepts such as statistics, linear algebra and optimization. The reader is expected to have basic understanding of the guiding principals of Deep Learning such as Feed-Forward Neural Networks, backpropagation, binary classification and model evaluation metrics.

2

The Data

At the time of undertaking this project, the KM3NeT Neutrino Telescope was still under construction, thus simulated data provided by Nikhef was used. The data itself was provided in two parts namely *events* and *noise* datasets, both of which came from different sources and in different formats. The events dataset was provided as a 42MB *HDF5* (Hierarchical Data Format) file consisting of the */data/mc_hits* and */data/mc_info* tables. For the purposes of this project, the two tables were combined such that each row in the mc_hits table contains its corresponding 'event_id' from the mc_info table. A *label* column was added containing a value of '1' and the resulting table (henceforth referred to as the *events* dataset) was saved as a CSV file.

The *noise* data was generated using a Python library written and maintained by Nikhef, *k40gen*. `k40gen.Generators(21341, 1245, [7000., 700., 70., 0.])` was used to create an instance of a generator where the first two arguments are random seeds followed by a list of rates at which single, double, triple and quadruple hits should be generated. The generator instance is then passed into `k40gen.generate_40()` method which returns a (4, n) array containing as rows *time (t)*, *dom_id*, *pmt_id* and *time over threshold (tot)*). The position coordinates (ie. x, y and z coordinates) for each datapoint was provided in a *positions.detx* file which was parsed using the Numpy Python package (8) and added to the *noise* array. The Python library Pandas (9) was used to convert the array into a (n, 4) dataframe. A `label` column was added containing a value of '0' and the dataframe was saved as a 3.9GB CSV file.

To create the *main* dataset for the project, the events and noise datasets were combined. Both datasets were read into memory as Pandas dataframes and their columns were renamed for consistency. The two dataframes were concatenated and sorted based on the time column. Rows with negative time were dropped along with columns which

were not relevant to this project. The `time` column was discretised into 15000ns bins and the resulting values were added to the `timeslice` column. The resulting dataframe was saved as a 1.9GB CSV file. The main dataset was explored using statistical analysis and visualizations to observe any patterns and "local trends" that may be present. Due to the high number of data points, a random sample of 10% of the data was taken in order to draw reasonable conclusions from the plots.

Table 2.1: Description of columns

Column	Data type	Unit	Description
x, y, z	float	meters (m)	The position within the detector where the hit was detected, they represent the x,y,z coordinates of the hit respectively.
t	float	nano seconds (ns)	The time at which the hit was detected.
label	int	NA	The type of hit, '0' represents noise and '1' represents a neutrino hit
event_id	int	NA	The id of the event to which the hit is related to. The id itself does not have any meaning, it is simply used to identify hits that originated from the same event.
timeslice	int	NA	The id of the timeslice to which the hit belongs. The id itself does not have any meaning, it is simply used to group hits into discrete bins.

Table 2.2 presents the descriptive statistics of the dataset. The dataset consists of 7 columns (or features) and roughly 4.5 million rows. Table 2.1 provides more information on the columns of the dataset. The dataset does not contain any `nan` or `null` values except for the `event_id` column where rows containing noise hits are not associated with any event. Next, the correlations amongst the features are checked using the "Pearson" correlation and depicted by a correlation matrix in Figure 2.1. No significant correlations are observed between x, y, z and t which indicates that ML models may not be able to learn anything from the dataset without the aid of feature engineering. The distribution of the `label` column is presented in Figure 2.2. A severe class imbalance was noted between events and noise hits. The dataset contains 489906 instances of events compared to over 4 million instances of noise. An effective strategy to handle the class imbalance was devised during training of models to prevent *overfitting*.

	pos_x	pos_y	pos_z	time	label	event_id	timeslice
pos_x	1.00	-0.01	-0.00	-0.00	0.00	-0.02	-0.00
pos_y	-0.01	1.00	0.00	0.00	-0.00	0.00	0.00
pos_z	-0.00	0.00	1.00	-0.00	0.00	-0.01	-0.00
time	-0.00	0.00	-0.00	1.00	-0.00	-0.01	1.00
label	0.00	-0.00	0.00	-0.00	1.00	nan	-0.00
event_id	-0.02	0.00	-0.01	-0.01	nan	1.00	-0.01
timeslice	-0.00	0.00	-0.00	1.00	-0.00	-0.01	1.00

Figure 2.1: Correlation matrix of features

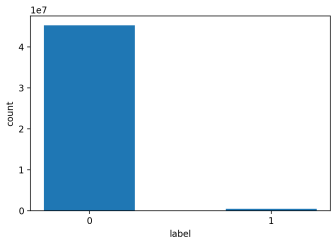

Figure 2.2: Distribution of `label` column

Table 2.2: Descriptive statistics

	x	y	z	t	label	event_id	timeslice
count	4.58e+7	4.58e+7	4.58e+7	4.58e+7	4.58e+7	489906	4.58e+7
mean	1.16e-02	-1.59e-02	1.17e+02	5.00e+07	1.06e-02	2862.00	3.33e+03
std	5.12e+01	6.22e+01	4.86e+01	2.89e+07	1.02e-01	1667.61	1.92e+03
min	-9.46e+01	-1.15e+02	3.77e+01	0.00e+00	0.00e+00	0.00	0.00e+00
25%	-4.50e+01	-5.79e+01	7.40e+01	2.50e+07	0.00e+00	1392.25	1.66e+03
50%	1.30e+00	-4.18e+00	1.21e+02	5.00e+07	0.00e+00	2887.00	3.33000e+03
75%	4.04e+01	4.85e+01	1.60e+02	7.50e+07	0.00e+00	4304.75	5.00000e+03
max	9.62e+01	1.05e+02	1.96e+02	1.01e+08	1.00e+00	5734.00	6.77e+03

The dataset is derived from synthetically generated data using simulations. As such, it is likely that the event hits in each timeslice may occur at a specific time such as at the beginning, middle or end of the timeslice. Having such a pattern in the dataset may bias the model since it may learn this pattern and thus fail to generalize. If this pattern does exist in the dataset, corrective measures need to be taken such that the event hits in each timeslice are uniformly distributed. To verify the existence of such patterns in the dataset, the mean time of event hits across all events was visualized as a scatter plot as depicted by Figure 2.3. A uniform distribution is noted with no visible patterns indicating no bias exists in the dataset and it is deemed suitable for further analysis.

The dataset is discretized into 6759 timeslices of which 2783 timeslices contain only noise hits. This is corroborated by Figure 2.5 which presents a skewed distribution where many timeslices contain few to no event hits and few timeslices contain a high number of event hits. Figure 2.4 depicts a scatter plot of *timeslice 615* which contains the largest number of event hits. It is observed that event hits occur close to each other in space and time

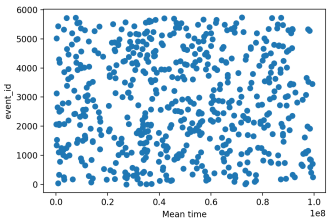

Figure 2.3: Verification of Bias

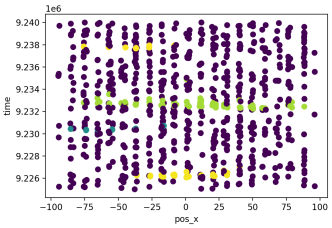

Figure 2.4: Distribution of Timeslice 615

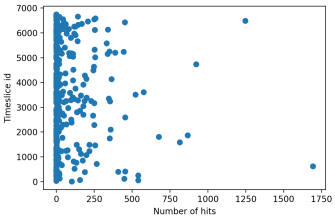

Figure 2.5: Distribution of event hits per timeslice

(represented by the yellow, blue and green points) whilst background hits are uniformly distributed in space and time (represented by the purple points).

3

Related Work

This chapter presents an overview of the existing body of work which are related to this project. Additionally, an overview of the GPU Pipeline by Karas et al. (2019) is also presented here as the research presented in this paper improves upon this seminal work. The chapter concludes with the limitations of the GPU Pipeline and motivates the need of a new pipeline.

3.1 Event Trigger Algorithms

Adrian et. al (2016) present *L0, L1, L2*, the state of the art event trigger algorithms currently used by the KM3NeT data acquisition pipeline. The L0 trigger applies a threshold to the analog signals in the PMTs and poses as the first level of filtration before the data is transferred to the on-shore facility. At the on-shore facility, the L1 filtration is applied by identifying coincidences of two or more L0 hits originating from different PMTs within the same DOM in a time window of 10ns. The time window is deduced by studying the scattering of light in deep sea. As a final filtration to reduce the occurrence of random coincidences from background noise, the L2 filtration is applied which is able to reduce the random coincidences by half by taking into account the known spatial orientation of the PMTs. An improvement upon the above event trigger algorithms was first proposed by Bakker et al (2011). In their paper, the authors introduce the *Match 3B Criterion* to identify causally related hits. The criterion operates by observing the distance between two hits and comparing it to the distance that a photon is known to travel in sea water.

3.2 Deep Learning in Particle Physics

Deep Learning has been an integral part of Particle Physics experiments (10, 11, 12, 13, 14) including but not limited to real time analysis and particle property prediction (10). Study of exotic particles such as *Higgs boson*, *anti matter*, *dark matter* and *quarks* is a highly sought after topic in this field. However, exotic particles are extremely difficult to detect due to the presence of negligible charge and mass. Additionally, once created, they are highly unstable and quickly decay into more stable forms (11). Thus to study exotic particles, the conditions prevalent during the *Big Bang* are recreated in large collision experiments such as the ones conducted in the *Large Hadron Collidor (LHC)*. Since the particles cannot be detected directly, the collision chambers are fitted with a plethora of sensors and detectors to collect the direction and momentum of each particle in order to recreate the collision event (11). These experiments produce data in the order of magnitude of *Petabytes* majority of which is just noise. To give an example, the LHC produces 10^{11} particles out of which only 300 may be a Higgs boson (11). Due to the availability of an abundance of data and the "data hungry" nature of Neural Networks, the adoption of *Deep Learning (DL)* for solving large scale problems in the field was only a natural step. Based on a study on the use of Machine Learning in neutrino experiments conducted by Psihas et al (2020), DL models such as *Multi Layer Perceptrons (MLPs)* and *Convolusional Neural Networks (CNNs)* have been applied to a variety of physics problems such as design, hardware triggers, energy estimation, reconstruction and signal selection. Many ML models have replaced their statistical predecessors since they provide better results.

A notable application of DL was in the discovery of the *Higgs boson* particle from data generated by the LHC collision experiments (10). The same paper makes use of CNNs and *Recurrent Neural Networks (RNNs)* for identification of *beauty-quark* particles where the data was represented in a graph structure. Sadowski et al. (2015) employed ML techniques to reduce the dimensionality of the raw dataset and compared the performance of shallow and deep MLPs for identification of Higgs boson particles. CNNs have been widely adopted for event recreation tasks since the data is often represented in the form of images. Terwilliger et al. (2017) present such an application where CNNs were used for *neutrino vertex reconstruction* which is a technique used to identify the origin of neutrino event using detector data represented as images.

All particle physics problem share the problem of having a high noise to signal ratio in their datasets and *Machine Learning (ML)* techniques have been used to tackle this problem (15, 16, 17, 18). In the deep sea environment in which the *ANTARES* neutrino

telescope resides, *Random Forest* and *Boosting Trees* were used for signal classification (18). Li et al. (2018) created a liquid neutrino detector toy model to replicate the conditions posed by the *JUNO* detector. The image data generated by the toy model was used to train CNNs to perform signal-noise discrimination and the model was able to outperform the existing solutions. Choma et al. (2018) used *Graph Neural Networks* to perform signal-noise detection using data collected by the *IceCube* detector and the model was able to outperform physics based methods and CNNs. Above the ground, Mulmule et al. (2020) proposed the use of *MLPBNN* a Bayesian extension of MLPs for signal-background detection of anti-neutrinos in the *ISMRAN* experiment. The performance of the model is better compared to the previously used statistical models based on likelihood estimates.

3.3 Deep Learning in KM3NeT

Under the umbrella of the KM3NeT research initiative, three scientific works of relevance to this project exist. The earliest being the paper by De Sio et al. (2019) which presents the application of CNNs for tasks such as event-type and particle identification, energy/direction estimation, source identification, signal/background discrimination and data analysis. The ML models provided better results compared to the reconstruction models. Moreover, they alleviate the requirement for reconstruction models altogether by extracting relevant features from the raw data directly. Post et al. (2019) applied CNNs along with *Long Short Term Memory (LSTM)* (20) for event triggering using simulated KM3NeT data. The model performs a multi-class classification amongst hits originating from noise, neutrinos and *atmospheric muons*. The model has promising results as it is able to attain an accuracy of 80%. Karas et al. (2019) built upon the idea of causally related hits by Bakker el al. (2011) and present a data processing pipeline to filter timeslices containing neutrino events from those containing only noise. The pipeline is able to achieve this filtration by processing the data in 3 steps, illustrated in Figure 3.1 and described in more detail below. Figure 3.2 provides a visual representation of the various shapes and arrangements of the data as it passes through the pipeline.

3.3.1 The GPU Pipeline by Karas et al. (2019)

The first step of The GPU pipeline is the Hit Correlation step which proposes *The Pattern Matrix Criterion (PMC)* to identify hits which may have originated from the same neutrino event, referred to as *causally related* hits. From domain knowledge, it is known that related hits occur close to each other both in space and time. Specifically, the space and time

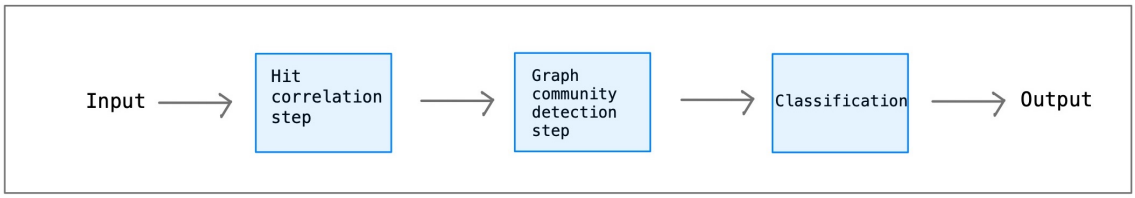

Figure 3.1: Overview of Karas Pipeline

difference of related hits is found to be 100m and 300ns respectively (2). The PMC operates by creating a correlation criterion based on the probability that the aforementioned space and time difference occurs between two related hits. Pairs of hits (from both events and noise) are passed to the PMC as input, the output being an adjacency matrix where related hits are assigned a high score whilst unrelated hits are assigned a lower score. The algorithm is evaluated with a dataset containing 130 event hits and 5000 noise hits and results in the range of 0.3 - 0.375 is reported for the recall, precision and F1 metrics and an accuracy of 80% is achieved. Due to the stocastic nature of hits, unrelated hits such as noise-event or noise-noise pairs may occur within this predefined range thus being falsely identified as related. This is rectified in the next step of the pipeline.

The second step of the Karas pipeline is the Graph Community Detection (GCD) step. The main dataset is represented as a graph where hits are represented as nodes. All nodes are connected with each other and carry as weight the probability of being related using the output of the PMC. The *Constant Potts Model (CPM)* (21) is used to group the nodes into separate communities (or clusters) of related and unrelated hits. The model is tested using a dataset consisting of 130 event hits and 5000 noise hits and performs exceptionally well as it is able to group most event hits into a single community and the noise in another.

Since the GCD step directly operates on the data derived using the Hit Correlation step, communities may still contain noise nodes. Thus the third and final step of the pipeline classifies given communities as *event* or *noise* communities based on the exclusive presence of event nodes. This can be done by observing two properties of the graph namely the size of the communities and the density of edges within the communities. Communities consisting exclusively of event nodes are of small sizes and have high edge density whilst communities consisting a mix of event and noise nodes or only noise nodes will be relatively larger and have lesser edge density. The two parameters which aid in the classification are the Probability Threshold (PT) of the PMC and the CPM resolution parameter (γ) of the GCD step. The PT and γ are grid searched to determine their optimal thresholds, all hits

above the specified thresholds are classified as event communities and the rest as noise communities.

3.3.2 Limitations of The GPU Pipeline

Although The Karas Pipeline is able to identify timeslices with neutrino event hits more accurately compared to its predecessors (5), the pipeline still has certain limitations which hinders its performance. The space and time difference based on which the PMC determines if two hits are causally related to one another is static. This may result in related hits which do not meet these thresholds to be incorrectly given a low score. Alternatively, due to the stocastic nature of hits, event-noise and noise-noise pairs which fall within the thresholds may also occur and these will be incorrectly given a high score. The communities created by CPM is influenced by the edge weights provided by the PMC thus any limitations of the PMC cascades down into the later steps of the pipeline. Furthermore, the classification step also operates on static values of the PT and γ and thus is unable to identify communities of size smaller than 20 hits.

Instead of deriving the correlation thresholds manually, a Neural Network can be used to learn the optimal thresholds instead. This idea is further explored in Chapter 4 where a MLP is used to classify related and unrelated hits. The choice of a MLP is motivated by existing literature which demonstrate the successful results obtained with the application of MLPs for signal-noise discrimination. Recently, the field of Geometric Deep Learning has gained popularity with some exciting developments of models which are designed to operate upon graph structures. Once such development is the Graph Convolusional Neural (GCN) Network proposed by Kipf et al (2016). The limitations of the GCD and classification steps may be alleviated by using a GCN to classify event and noise nodes and is explored in Chapter 5. The choice of a GCN is motivated by the graph data structure of the GCD step and existing literature validating the application of GCNs for neutrino detection.

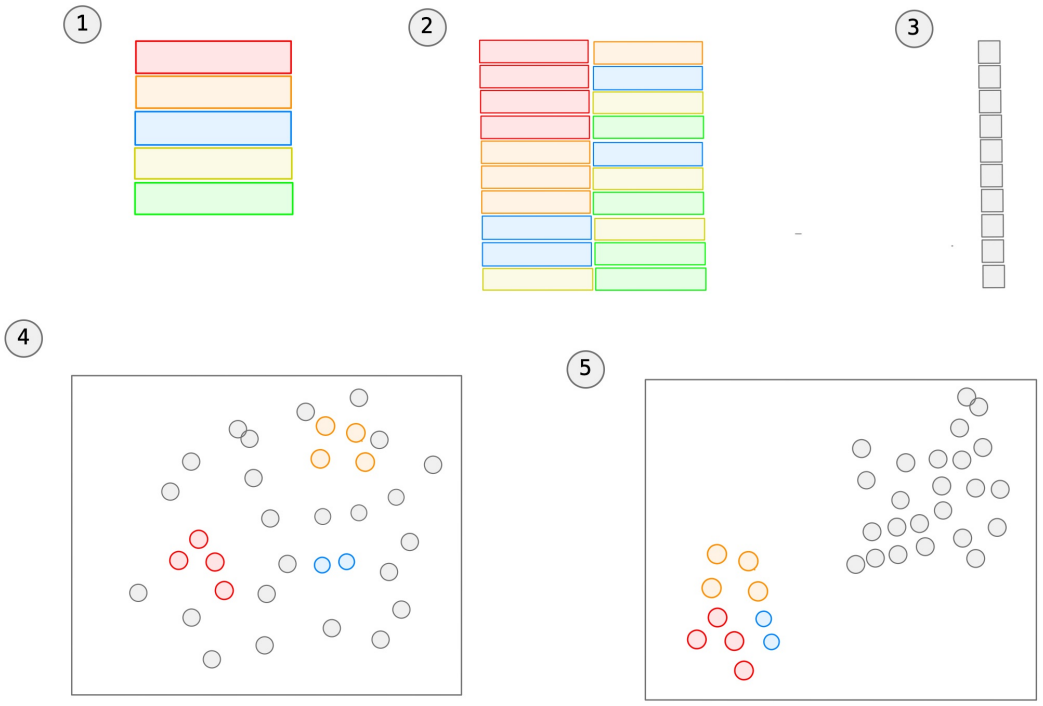

Figure 3.2: **(1)**. The dataset of the project where each row represents a hit consisting of the $(x, y, z, t)vector$. Here an example set containing 5 hits is shown. **(2)**. The input to the Hit Correlation Step is all unique pairs of hits. **(3)**. The output of the Correlation Step is a vector containing the probability that the pairs of hits are related to each other. **(4)**. (1) and (3) are used to construct a graph representation of the dataset where event (red, orange and blue) and noise (grey) hits are represented as nodes and are connected by an undirected edge carrying as weight the probability of being related (for simplicity, the edges are not shown). **(5)**. The graph from (4) acts as the input to the Graph Community Detection step, the output of which is another graph where event and noise nodes are grouped together and thus are linearly separable from each other.

4

Replacement for Hit Correlation Step

This chapter presents the replacement created using a Multi Layer Perceptron (MLP) for the *Hit Correlation Step* of the GPU Pipeline proposed by Karas et al. (2019). It is observed that a MLP is able to identify causally related hits with a higher accuracy, precision and recall compared to the PMC. The chapter begins by explaining how the data is created followed by its visual examination. The training and testing procedure for the model is explained next. The chapter concludes with discussions of the experiment results and next steps.

4.1 Data Preparation

Figure 4.1 summarizes the MLP dataset creation process. With an input data of shape $(n, 4)$ (n rows and 4 columns representing *x, y, z, and t*), an output data of shape $(\sum_{k=n-1}^{1} k$, 9) is obtained. A significant rise in the number of rows is observed since each row representing a single hit is paired with all rows that follow. The output dataset consists of 9 columns due to the presence of *x, y, z and t* columns of two hits plus the label column.

The label column is populated based on the values of the `event_id` column of the two hits. The row is assigned a label of 1 if the two hits have the same event id. This signifies that they originated from the same neutrino event and hence are causally related to each other. If the event id of the two hits are not the same then they are assigned a label of 0.

Better model performance was observed when the model was trained with the difference between hits in time and space. As a result, the final dataset of shape $(\sum_{k=n-1}^{1} k, 5)$ was obtained - the first 4 columns being the difference of *x, y, z and t* vectors of the paired hits and the last column being the label. The data is additionally scaled between $[0, 1]$

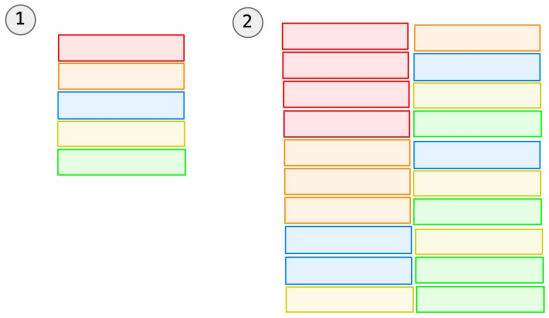

Figure 4.1: Overview of MLP dataset creation procedure. **(1)** The main dataset where each row represents a hit. Here an example set containing 5 rows is shown. **(2)** The MLP dataset is generated from the main dataset consisting of all unique pairs of hits. Algorithmically, this is done by pairing each hit with the subsequent hits below it as demonstrated with the use of colors. **(3)** The difference between the hits is taken (dark orange) and a label is assigned to each row (gray).

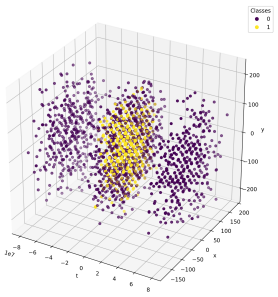

Figure 4.2: Distribution of MLP training dataset.

due to empirical evidence showing improvement of model performance and prevention of vanishing gradients during training (22, 23).

4.1.1 Preparation of Training Data

The main dataset is highly skewed, with the **majority or negative class** being hits from background noise and the **minority or positive class** being hits from neutrino events. Thus, the training set created is also skewed with the minority class being related hits and majority class being unrelated hits. To maximize the number of positive examples in the training set, a random sample was taken from the top 5 timeslices of the main dataset with the most number of event hits. The model is required to classify related and unrelated hits which can be done by observing the space and time difference between the given points. Since this phenomenon is consistent across the entire main dataset, training using a sample does not introduce any bias into the model.

The training set still however contains a skewed distribution of examples, and training the model with such a dataset will result in a model that is biased to the majority class. To combat this problem, the majority class is *undersampled* such that the number of examples for each class is the same. Figure 4.2 shows the distribution of a random sample of the training set. It is observed that the related hits occur close to one another in space and time whilst the noise hits are scattered throughout which should aid the model to learn.

A fraction of the training set is kept as a holdout or validation set to evaluate the model's training. Table 4.1 presents the distribution of the training and validation sets.

Table 4.1: Distribution of MLP Training and Validation Datasets.

	Total examples	Positive examples	Negative examples
Training	48,434	24,217	24,217
Validation	23,856	11,928	11,928

4.1.2 Preparation of Testing Data

Whilst the training dataset contains equal number of examples for each class, the testing dataset maintains its skewed distribution since this represents realistic data which the model will be required to classify. Four variants of the testing dataset with varying levels of examples of related hits were created as listed in Table 4.2. In practice, the pipeline will observe timeslices which contain no to very few related hits, thus the performance of the model on TS1 and TS2 are of vital importance.

Table 4.2: Distribution of MLP Test Datasets.

	Total examples	Positive examples	Negative examples
TS1	774,390	–	774,390
TS2	5,829,405	10	5,829,395
TS3	5,880,735	1176	5,879,559
TS4	364,231	8,372	355,859

4.2 Model Description

The expectation of the model is to identify if two given points are causally related to each other or not. As revealed through data exploration in Chapter **??**, hits originating from neutrino events occur close to each other in space and time. Thus, the expectation from the model is to learn this phenomenon by training over pairs of points and classifying unseen data as related or unrelated.

The model architecture is summarized in Table 4.4. It consists of an input layer, two hidden layers and an output layer. The network is fully connected with 4 neurons in the input layer, 16 neurons in the first hidden layer, 8 in the second hidden layer and finally 1 neuron in the output layer. The optimal value of all parameters stated above were identified

Table 4.3: GCN Model Parameter Summary.

Loss	BCELoss
Optimizer	Adam
Learning rate	0.001
Hidden activation function	ReLU
Output activation function	Sigmoid
Training batch size	16
Testing batch size	32

either empirically or from recommendations presented in literature. The number of epochs used to train the model is not mentioned above since this parameter is largely determined by the dataset, batch size and the learning rate, thus its value varied per experiment.

The parameters of the model are summarized in Table **??**. The Adam optimizer with a learning rate of 0.001 is used to optimize the loss function. This combination is considered a reasonable start for many optimization problems and has been empirically proven to outperform other algorithms such as *Stocastic Gradient Descent (SGD)*, *RMSProp* and *Adagrad* (24). Being a binary classification task, the *Binary Cross Entropy Loss (BCELoss)* (also known as *Log Loss*) was selected as the loss function since it has been established as the standard loss function for binary classification tasks (25). As the BCELoss function expects an input in the range of $[0, 1]$, the *Sigmoid* activation function was chosen for the output layer. The *ReLU* activation function was chosen for the hidden layers since it is generally considered a good start for most problems (26, 27). A batch size of 16 is used for the training and validation sets whilst a larger batch size of 32 is used for the testing set since these value are often recommended as good defaults by machine learning practitioners (22**?**).

Table 4.4: MLP Model Architecture Summary.

Layer Position	Type	Activation	In features	Out features
1	Linear	ReLu	4	16
2	Linear	ReLU	16	8
3	Linear	Sigmoid	8	1

4.3 Model Evaluation

The model is evaluated using metrics typical to binary classification tasks. Additionally, metrics geared towards skewed datasets are considered as well.

1. **Accuracy**. The accuracy is the ability of a model to classify unseen data correctly and is mathematically defined by Equation 4.1.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ examples} \tag{4.1}$$

2. **Learning Curve**. The Learning curve is a line plot of the loss over the training epochs. A model with a good fit results in a loss curve which approaches 0 with time.

3. **Confusion Matrix (CM)**. For highly skewed data, accuracy is not a good metric for evaluating the model performance (28) since it may achieve a high score by simply predicting the majority class. Thus the CM is used to visualize the number of *true positive (TP)*, *true negative (TN)*, *false positive (FP)* and *false negative (FN)* predictions of the model.

4. **Recall**. The recall is the ability of the model to correctly identify the minority class. For this problem, the recall of the model is given precedence over its precision. This is because the model should be able to identify all instances of the positive class since this determines if the timeslice will ultimately be saved or not. The recall is mathematically defined by Equation 4.2.

$$\frac{TP}{TP + FN} \tag{4.2}$$

5. **Precision**. The precision is the ability of the model to not misclassify an instance of the negative class (ie. classify it as the positive class). Although this should also be high, it is often inversely proportional to recall. The precision is mathematically defined by Equation 4.3.

$$\frac{TP}{TP + FP} \tag{4.3}$$

6. **F1 score**. The F1 score is the harmonic mean of the precision and recall, thus it is a single metric to summarize the model's performance based on its precision and recall. The F1 score is a value between $[0, 1]$ with a value close to 1 indicating high precision and recall. The F1 score is mathematically defined by Equation 4.4.

$$\frac{2 * (Precision * Recall)}{Precision + Recall} \tag{4.4}$$

7. **F2 score**. Since recall is given precedence for this problem, the F2 score can be considered a better alternative to the F1 score as it gives higher importance to the recall through the β parameter. The F2 score is mathematically defined by Equation 4.5. Thus the F2 score with $\beta = 1$ is equivalent to the F1 score.

$$\frac{(1 + \beta^2) * Precision * Recall}{(\beta^2) * (Precision + Recall)} \tag{4.5}$$

8. **Receiver Operating Characteristic (ROC) curve** is a plot of the *true positive rate (TPR)* and the *false positive rate (FPR)* (mathematically defined by Equations 4.6) across various discrimination probability thresholds of the model. The ROC curve can be interpreted as the fraction of correct predictions for the positive class (along the y-axis) versus the fraction of error in predictions for the negative class (along the x-axis). The area under the ROC curve (ROCAUC) can be used to summarize the ROC curve with a singular value. Thus, a highly skilled model has a ROC curve which arches from $(0, 0)$ to $(1, 1)$ with a ROCAUC between 0.5 and 1.0. Whilst the ROC curve of a model without any skill is a straight line from $(0, 0)$ to $(1, 1)$ with a ROCAUC of 0.5.

$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ FPR &= \frac{FP}{FP + TN} \end{aligned} \tag{4.6}$$

9. **Precision-Recall (PR) Curve** can be considered a better alternative to the ROC curve since the ROC curve can be overly optimistic of the model's skill when dealing with highly skewed data since it considers the model's performance for both the classes. In contrast, the PR curve is a diagnostic plot of the model's precision and recall across various discrimination probability thresholds of the model. Thus the PR curve only considers the model's performance with regards to the positive class.

Similar to the ROC curve, the PR curve can also be summarized by the area under the curve or the PRAUC. A skilled model, thus has a PR curve which bows towards $(1, 1)$ whilst a model with no skill is a horizontal line.

4.4 Results

Figure 4.3 shows the learning curve of the model on the training and validation tests. The curves approach zero with time and remain in close proximity to each other indicating a good fit. Table 4.5 summarizes the model's performance across the various test sets (see Section 4.1.2). In general, the model performs well with high accuracy and recall across all test sets. The model achieves almost perfect recall for TS2 and TS3 which have extremely low positive examples which speaks to its strength. The poor precision scores can be attributed to the skewed nature of the datasets. Since the FPs are relatively high compared to the TPs in the datasets, the precision falls drastically. This is also corroborated by the fact that the recall in TS4 increases due to a better FP:TP ratio (see Figure 4.16).

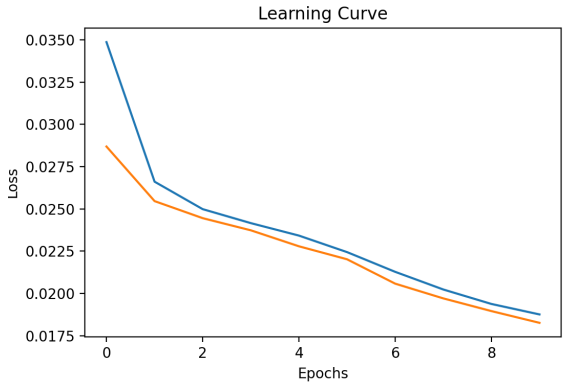

Figure 4.3: Learning Curve of MLP Training and Validation Datasets.

Table 4.5: Summary of MLP performance across test sets.

	Accuracy	Precision	Recall	F1	F2	ROCAUC	PRAUC
TS1	0.80	–	–	–	–	–	–
TS2	0.91	0.00	1.00	0.00	0.00	0.99	0.00
TS3	0.92	0.00	0.98	0.00	0.01	0.98	0.01
TS4	0.92	0.19	0.83	0.31	0.48	0.96	0.33

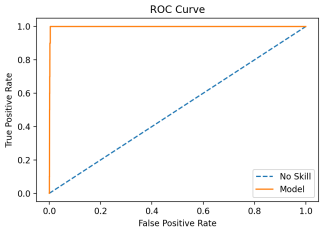

Figure 4.4: ROC Curve for TS2.

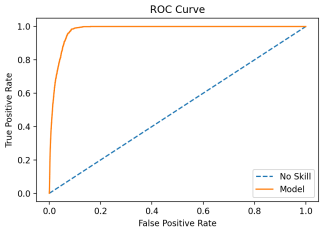

Figure 4.5: ROC Curve for TS3.

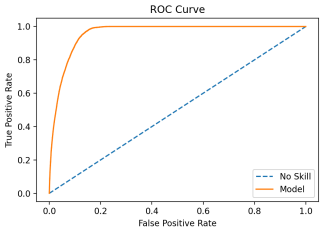

Figure 4.6: ROC Curve for TS4.

Figure 4.7: ROC Curves for MLP Test Datasets.

The ROC curves paint a different picture (see Figure 4.7) with exceedingly large areas under the curve. This however is misleading since ROC curve takes into account the performance for both the positive and negative classes and thus tend to be overly optimistic of the model's performance on skewed datasets (28, 29). In such circumstances, The PR curves are considered better at gauging a model's skill when dealing with skewed datasets (28). However, in this case it too is deemed misleading as it significantly undermines the model's performance (see Figure 4.11). This again is attributed to the precision approaching zero due to the extremely skewed FP:TP ratio which results in the PRAUC to also be zero.

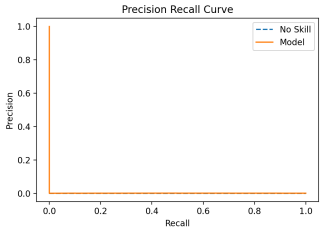

Figure 4.8: PR Curve for TS2.

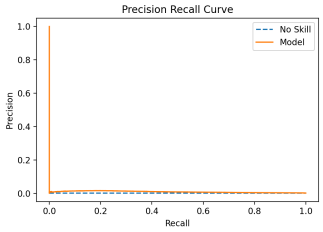

Figure 4.9: PR Curve for TS3.

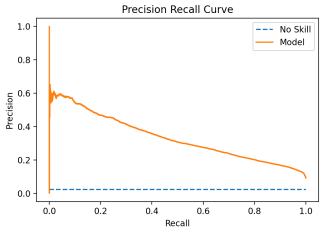

Figure 4.10: PR Curve for TS4.

Figure 4.11: PR Curves for MLP Test Datasets.

Figure 4.16 depict the confusion matrices for the various test datasets. The confusion matrices highlight the strengths and weaknesses of the model better than the ROC and PR curves. The model has a very low number of FNs and FPs which is extremely valuable as this ensures that the model does not miss hits from neutrino events and also does not classify hits from noise sources incorrectly as hits from neutrino events.

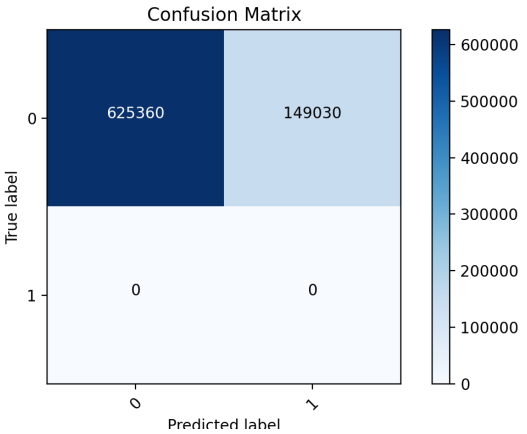

Figure 4.12: Confusion Matrix for TS1.

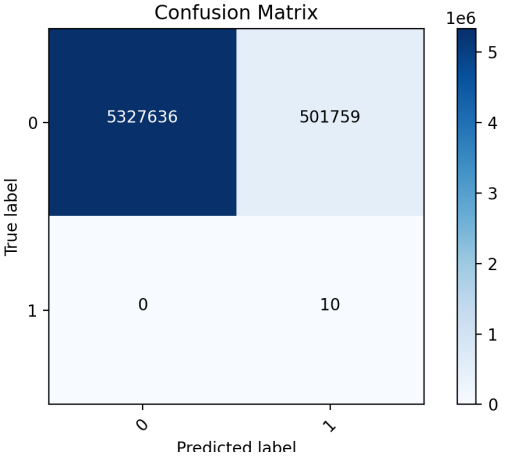

Figure 4.13: Confusion Matrix for TS2.

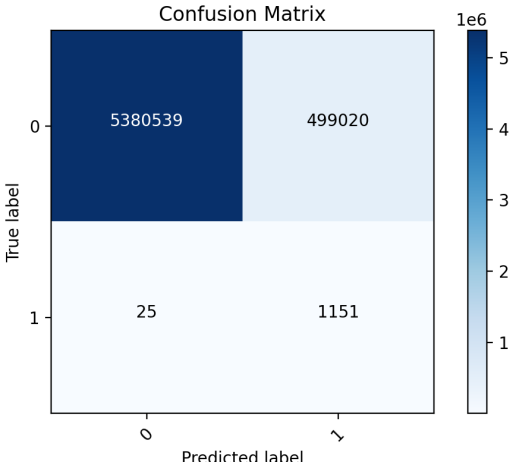

Figure 4.14: Confusion Matrix for TS3.

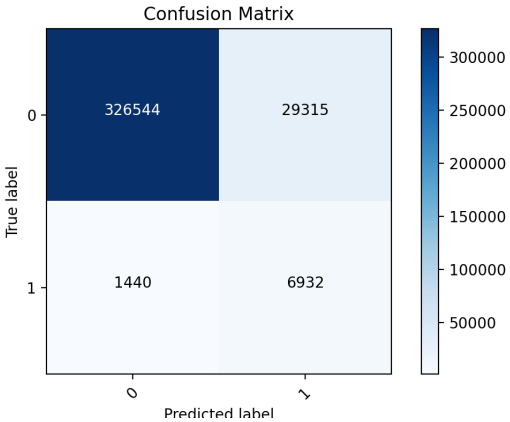

Figure 4.15: Confusion Matrix for TS4.

Figure 4.16: Confusion Matrices of MLP Test Datasets.

5

Replacement for Graph Community Detection Step

This chapter presents the replacement created using a Graph Convolusional Neural Networks (GCNs) as proposed by Kipf et al. (2016) for the *Graph Community Detection Step* of the GPU Pipeline. It is observed that a GCN is able to identify event nodes very well however is severely biased to them thus unable to distinguish between event from noise nodes. The chapter begins with an overview of GCNs and how they have been applied to this problem. The data preparation and model evaluation are touched upon next. The chapter concludes with discussion of the results.

5.1 Primer on Graph Convolusional Neural Networks

GCNs are designed to operate on data consisting of entities and their relations, commonly referred to as *graphs*. A graph $G = (V, E)$ consists of a set *nodes (V)* and a set of *edges (E)*. Each node may or may not be connected to one or many nodes. These are referred to as the *neighbors* of the node. A graph with all nodes connected to one another is called a *fully connected graph*. An edge may have attributes associated with it, the two most common attributes being *weight* and *direction*. An edge $(u, v) \in E$ between two nodes u and v may be *directed* which denotes a sense of hierarchy amongst the nodes, an example being a graph which models how Twitter users follow one another. An edge may also *undirected* such as a graph which models the friendship amongst the users of a social network (since friendship is mutual). An edge may also have a weight to signify a stronger or weaker connection amongst nodes. Nodes may also posses attributes associated with themselves, commonly known as *node embeddings*. The complexity of the node embedding

may range from a simple scalar quantity to a multi-dimensional tensor, and depends on how the dataset is modeled as a graph.

Graphs are primarily classified into two variants namely *homogeneous* and *heterogeneous* graphs. Homogeneous graphs have the same type of entities and relations represented as nodes and edges respectively. For example, a graph representing the social network consisting of people and their connections is a homogeneous graph. In contrast, Heterogeneous graph consist of different types of nodes and edges. For example, a graph representing a person's likes and dislikes in regards to food items. Here, two entities, namely people and food are represented as nodes. The edges also come in two variants ie. a 'like' and a 'dislike'.

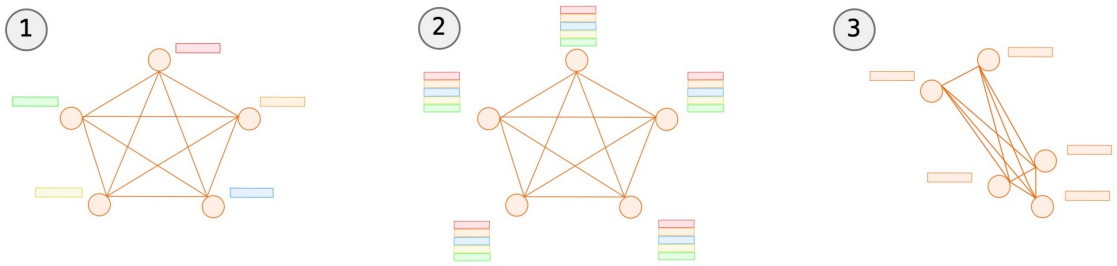

Figure 5.1: Message passing paradigm of GCNs. **(1)** An example of a fully connected graph with 5 nodes. The edge weights are hidden for simplicity. **(2)** Each node sends its embedding to its neighbors. **(3)** The aggregated embedding becomes the new node embedding thus changing the structure of the graph.

GCNs learn by utilizing a message passing paradigm which is summarized in Figure **??**. During each training epoch, all nodes propagate their embedding to their neighbors. The collected embeddings are then aggregated (for example using a sum, difference or mean) which becomes the new embedding for the node. This procedure is done for all nodes of the graph, for each training epoch. The number of layers in the network determine how far the messages are sent. For example, for a network with a single layer, each node aggregates the embeddings from their immediate neighbors. With 2 layers, the node also aggregates embeddings from the neighbors of its immediate neighbors and so forth.

5.2 Data Preparation

The graphs for the testing and training of the network are constructed from a combination of the main dataset and a modified version of the MLP dataset, Figure 5.2 illustrates this procedure. A fully connected graph is constructed, and its node embeddings and node

labels are derived from the main dataset. Each node is thus assigned a (x, y, z, t) vector as its node embedding. The node is assigned a label of 1 if it is an event hit, else a label of 0 to denote noise.

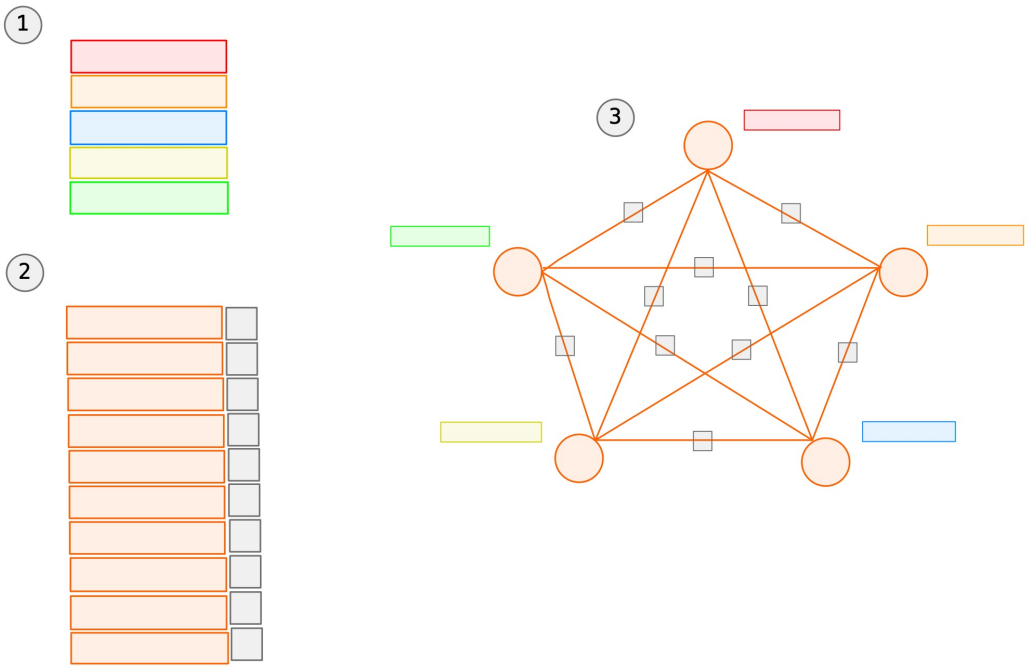

Figure 5.2: Overview of GCN dataset creation procedure. **(1)** an example of the main dataset with 5 hits. **(2)** the corresponding modified MLP dataset created. Although the dataset will contain $n^2 - n = 25$ rows, only the unique rows (10 in this example) are shown in the illustration for simplicity. **(3)** The graph representation is created where rows of (1) become the node embeddings and the label column of (2) become the edge weights.

A modified MLP dataset (see 4.1) with a shape of $(n^2 - n, 5)$ is created such that each hit is paired with all other hits except itself. The label column from this dataset is then used as the edge weights of the graph. Edges between event nodes from the same event thus are assigned a weight of 1 and all other edges are assigned a weight of 0.

Since the main dataset and the MLP dataset are highly skewed, the GCN dataset is also skewed with majority of the nodes being noise. Similar strategy as used in the creation of the MLP training set (see 4.1) is used. The training set is a graph with approximately 1000 nodes equally distributed amongst the classes. The skewed nature of the data is maintained in the testing sets. The model is evaluated with 3 test sets, each with varying

levels of event nodes. In practise, the pipeline will observe timeslices with no to very few events, thus the performance of the model on test set 1 and 2 should be given importance. The various test sets and their distribution are summarized in Table **??**.

Table 5.1: Distribution of GCN testing datasets.

	Total examples	Positive examples	Negative examples
TS1	1000-1500	–	1000-1500
TS2	1000-1500	10-20	990-1480
TS3	1000-1500	200-250	800-1250

5.3 Model Description and Evaluation

Table 5.2: GCN Model Parameter Summary.

Loss	BCELoss
Optimizer	Adam with learning rate of 0.001
Hidden Activation	ReLu
Output Activation	Sigmoid

The model is expected to classify nodes of an unseen graph as event or noise nodes. Since causally related nodes are connected with edges carrying a high weight, the model is expected to group them together resulting in a final graph with separate clusters of causally related nodes and noise nodes. The parameters of the model are summarized in Table 5.2. The rational for selecting the parameters remains the same as that of the MLP model (see 4.2) since both models perform binary classification. The difference comes from the model architecture which is summarized in Table 5.3. The GCN model comprises of an input layer, two graph convolusional layers and an output layer. The network is fully connected with 4 neurons in the input layer, 16 in both graph convolusional layers and 1 neuron in the output layer. A dropout layer is added between the two Gconv layers to prevent overfitting (30). The same evaluation metrics are used (see Section 4.3) since the GCN dataset is also highly skewed in nature.

5.4 Results

A good fit is achieved by the model during training as seen in Figure 5.3. In addition to the model's performance on the various test sets as summarized by Table 5.4, the node

Table 5.3: GCN model architecture summary.

Layer position	Type	Activation	In features	Out features
1	GConv	ReLU	4	16
2	dropout	–	–	
3	GConv	RelU	16	2
4	Linear	Sigmoid	2	1

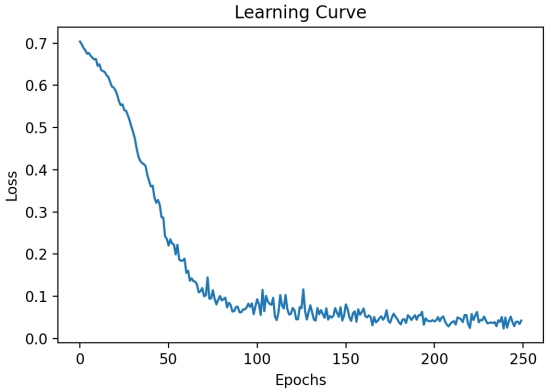

Figure 5.3: Learning Curve for GCN.

embedding of the training and testing graphs are also inspected using t-SNE (31). Figure 5.6 shows the node embedding of the training before and after training. It is interesting to note that the model is able to learn as clusters of similar nodes are noticed after training.

Inspecting Figure 5.17 one can see the model's inability to cluster similar nodes in the testing sets. Very scarce communities are observed in TS1 and TS2 with the model only being able to cluster the event nodes in TS3. The model is biased to the minority class as is seen in the Confusion Matrices depicted in Figure **??**. The model is able to identify all event nodes in TS2 and TS3 perfectly however has a high number of FPs in all 3 test sets. This indicates that the presence of the high edge weights assigned to causally related nodes greatly aids the model in identifying the event nodes. However, since all other edges

Table 5.4: Summary of GCN performance across test sets.

	Accuracy	Precision	Recall	F1	F2	ROCAUC	PRAUC
TS1	0.52	–	–	–	–	–	–
TS2	0.58	0.04	1.00	0.08	0.18	0.87	0.06
TS3	0.67	0.40	1.00	0.57	0.77	0.81	0.36

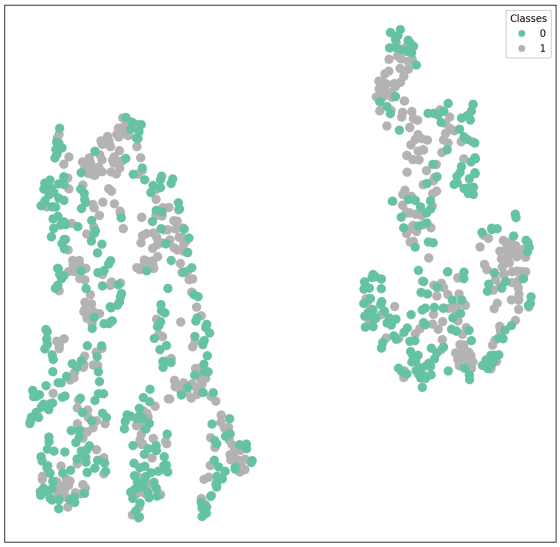

Figure 5.4: TSNE for training set before training.

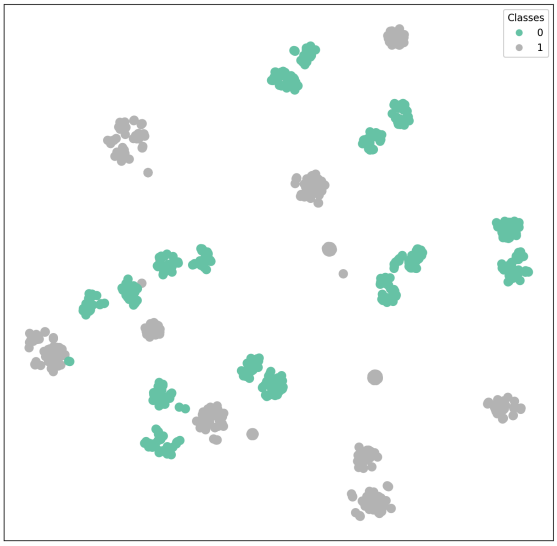

Figure 5.5: TSNE for training set after training.

Figure 5.6: TSNE for GCN training dataset.

are assigned a weight of 0, the model does not quite learn to identify the noise nodes.

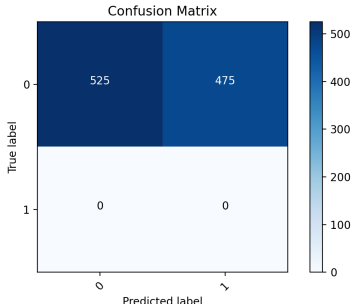

Figure 5.7: CM for TS1.

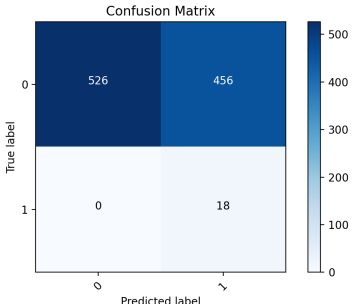

Figure 5.8: CM for TS2.

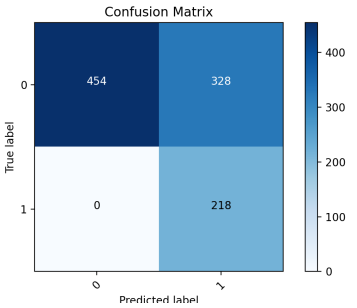

Figure 5.9: CM for TS3.

Figure 5.10: CM for GCN Test Datasets.

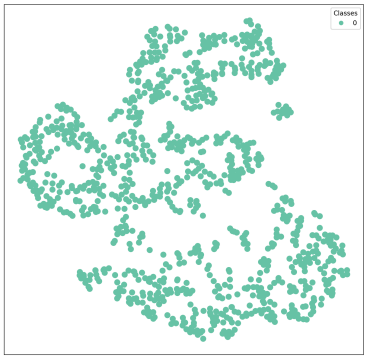

Figure 5.11: TSNE for TS1 before.

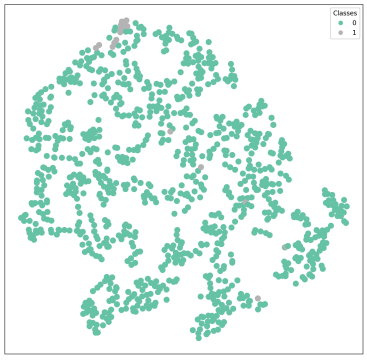

Figure 5.12: TSNE for TS2 before.

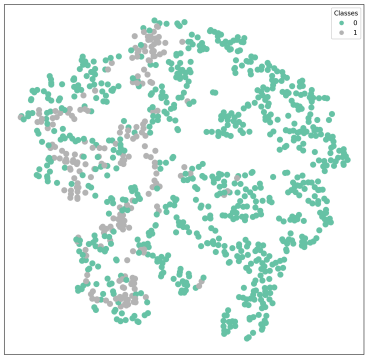

Figure 5.13: TSNE for TS3 before.

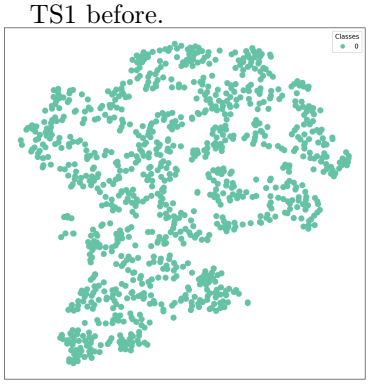

Figure 5.14: TSNE for TS1 after.

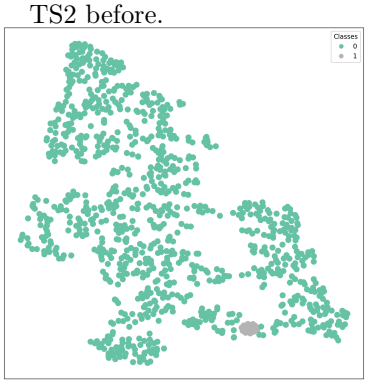

Figure 5.15: TSNE for TS2 after.

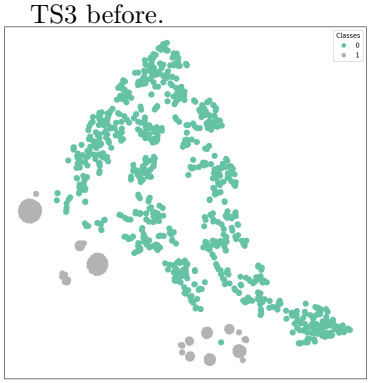

Figure 5.16: TSNE for TS3 after.

Figure 5.17: TSNE for GCN Test Datasets with naive edge weights.

5. REPLACEMENT FOR GRAPH COMMUNITY DETECTION STEP

6

Recommendations

This chapter presents practical recommendations for the readers who wish to use the new data processing pipeline presented in this report. The chapter also presents alternative paths of research which remain unexplored and general improvements that can be made to the pipeline in the future.

6.1 On the MLP

The Multi Layer Perceptron presented in Chapter 4 is capable of identifying causally related hits with a higher accuracy, precision and recall compared to the Pattern Matrix Criterion presented by Karas et al. (2019). It is therefore considered a viable successor to the PMC. Although experiments were done to identify the optimal parameters such as the batch size, optimization function, learning rate, height and depth of the network, further experimentation is recommended before it is integrated into the Data Acquisition Pipeline. The model also showed an increase in the number of FPs when the number of positive class examples increased. It may be possible to correct this bias by adding some regularization into the model (**?**).

Table 6.1: Advanced edge weight scheme for GCN.

Node type(s)	Edge weight
noise-noise	1.0
event-event (causally related)	1.0
event-event (causally unrelated)	0.5
event-noise	0.1

6.2 On the GCN

The GCN obtained in Chapter 5 has a perfect recall however is also significantly biased to the positive class (event nodes) and thus unable to identify the negative class (noise nodes). The origin of the problem can be traced back to the edge weights being applied to the graph. The weights supplied by the MLP only take into account edges between causally related and unrelated hits. In reality however, unrelated hits consists of various sub categories: 1. noise-noise hits 2. noise-event hits and 3. causally unrelated event-event hits. Although from a physics point of view the pairs of hits listed above are in fact causally unrelated, this *naive* edge weight scheme does not allow the GCN to learn adequately. The root cause of this conflict is based in the fact that the two models have contradictory goals. Whilst the MLP is trained to identify causally related and unrelated hits, the GCN is required to identify hits originating from neutrino events and noise. The solution is simple and requires assigning edge weights based on the nodes that it connects as summarized in Table 6.1. The GCN model is trained using the *advanced* edge weight scheme and the results are promising as observed in Figure 6.8 and 6.4. The model now has perfect discriminatory skills for both classes even in the extremely skewed datasets like TS2.

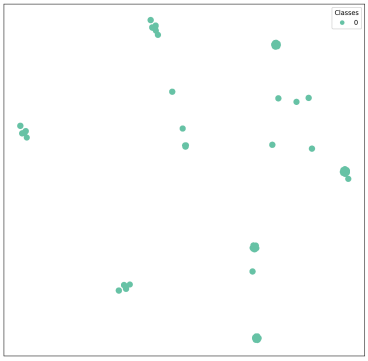

Figure 6.1: TSNE for TS1 with advanced edge weights.

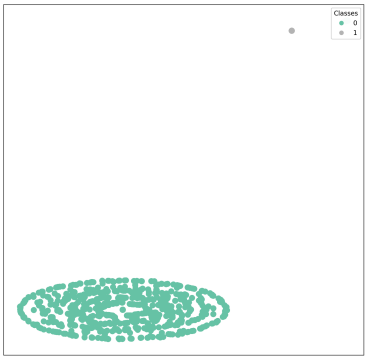

Figure 6.2: TSNE for TS2 with advanced edge weights.

Figure 6.3: TSNE for TS3 advanced edge weights.

Figure 6.4: TSNE for GCN test datasets with advanced edge weights.

Although the GCN is now capable of identifying event and noise nodes with immaculate accuracy, precision and recall, it however classifies all examples as false positives in TS1 which contains no event nodes. The model is impeded by the high weights on edges between causally related event nodes and noise nodes, which may be possible correct by

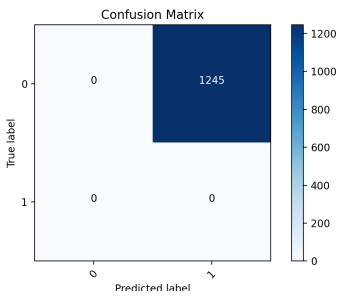

Figure 6.5: CM for TS1 (advanced edge weights).

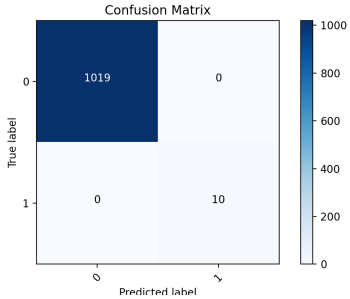

Figure 6.6: CM for TS2 (advanced edge weights).

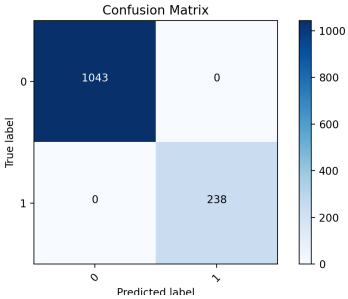

Figure 6.7: CM for TS3 (advanced edge weights).

Figure 6.8: CM for GCN test datasets with advanced edge weights.

framing the data as a multi-edge heterogeneous graph. By creating different types of edges corresponding to the various types of connections that two nodes may posses, each carrying the corresponding weights, the network may be able to correct its bias to the positive class. In order to obtain the advanced edge weight, the MLP must be modified such that it performs multi-class classification on the edge types. For a classification problem of n edge types, the output of the MLP will thus become a `(n,)` vector containing the expected probability for each class. These probabilities can then be used as the edge weights as illustrated in Figure 6.9. The MLP may not be able to discern causally unrelated hits since they are spread evenly in space and time. Thus research to identify an appropriate model for the task is required. One such approach may be to use a GCN to perform classification of the edge types or regression on the edge weights (32).

Table 6.2: Summary of GCN performance across test sets with advanced edge weights.

	Accuracy	Precision	Recall	F1	F2	ROCAUC	PRAUC
TS1	0.00	–	–	–	–	–	–
TS2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
TS3	1.00	1.00	1.00	1.00	1.00	1.00	1.00

6.3 On Independence of the Models

There is a high dependency on the Hit Correlation Step by the subsequent steps of the GPU Pipeline. The output of the PMC is used by these subsequent steps so any shortcomings of the PMC cascade down affecting the performance of the latter steps and the pipeline as a whole. This dependency is also present in the new pipeline as the performance of

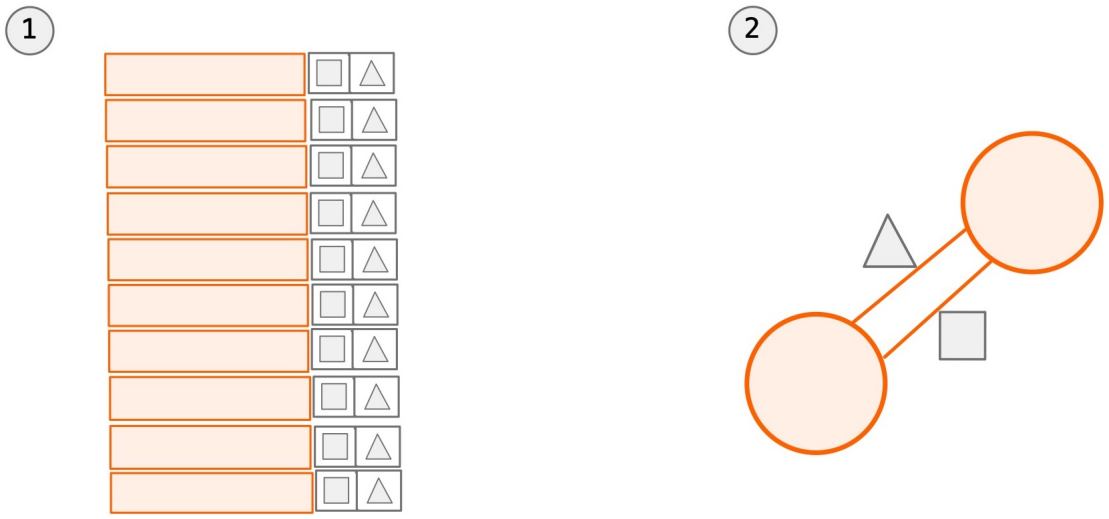

Figure 6.9: Heterogeneous graph for GCN.

the MLP directly dictates the performance of the GCN. This dependency is not ideal and an effective solution to make the models independent is required. One such solution is to explore the possibility of replacing the entire pipeline with a single GCN. The data can be framed such that the (x, y, z) vectors are used as the node embedding. If the t is scaled between $[0, 1]$ and the complement of Δt is assigned as the edge weights, it should result in a graph where edges between causally related nodes carry a high weight. After convolusion, a reasonable expectation is the presence of small and tightly connected communities of causally related nodes and large, weakly connected communities of noise nodes (illustrated in Figure 6.10). The GCN model can now be modified to perform graph classification (33) instead of node classification. The premise being that presence of small, densely connected communities indicate the timeslice is important and thus should be saved for further analysis. Alternatively, the Line Graph Neural Network proposed by Chen et al. (2017) built specifically for community detection can also be used.

6.4 On the Runtime Performance

The models presented in this report were tested in isolation. The intended usage however is to use them in tandem in order to identify timeslices containing neutrino event hits. Thus it is recommended that the models be tested as an integrated pipeline so that the results may be compared to that of the GPU Pipeline. Additionally, performance of the pipeline should be observed using various permutations of the individual parts of the new and the

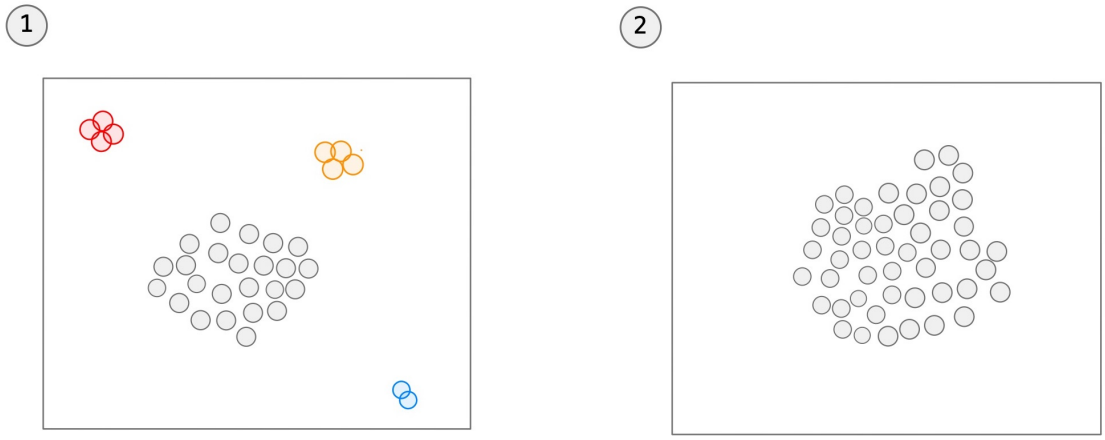

Figure 6.10: Embeddings of graph.

old pipeline to determine the best performing combination overall. As noted in Section 1.2, the runtime performance of the pipeline is crucial and should be able to perform filtration in near real time. As neural networks can be parallelized using the compute power of GPUs, the models should be capable of meeting the requirements. Several existing work also indicate the feasibility of scaling GCNs to parallel computation over large graphs (34, 35, 36), however the data preparation may pose a bottleneck. Of course at this point these are merely speculations and empirical proof still requires to be gathered.

7

Conclusion

This report presented the research undertaken to validate the application of Deep Learning for neutrino detection in the KM3NeT Neutrino Telescope. Chapter 1 presented an introduction to the problem space and its importance, the stakeholders and their requirements and finally the research questions this project sought to answer. In Chapter 2

Bibliography

[1] Annarita Margiotta, KM3NeT Collaboration, et al. **The KM3NeT deep-sea neutrino telescope**. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **766**:83–87, 2014. 1

[2] Silvia Adrian-Martinez, M Ageron, F Aharonian, S Aiello, A Albert, F Ameli, E Anassontzis, M Andre, G Androulakis, M Anghinolfi, et al. **Letter of intent for KM3NeT 2.0**. *Journal of Physics G: Nuclear and Particle Physics*, **43**(8):084001, 2016. 1, 2, 12

[3] Sebastiano Aiello, Fabrizio Ameli, Annarita Margiotta, Michel Andre, Giorgos Androulakis, Marco Anghinolfi, Antonio Marinelli, Gisela Anton, Miquel Ardid, Christos Markou, et al. **KM3NeT front-end and readout electronics system: hardware, firmware, and software**. *Journal of Astronomical Telescopes, Instruments, and Systems*, **5**(4):046001, 2019. 2

[4] Maarten Post. *"KM3NNeT" A neural network for triggering and classifying raw KM3NeT data.* PhD thesis, Universiteit van Amsterdam, 2019. 2

[5] Konrad Karaś. *Data processing pipeline for the KM3NeT neutrino telescope.* PhD thesis, Universiteit van Amsterdam, 2019. 2, 13

[6] Santo Fortunato. **Community detection in graphs**. *Physics reports*, **486**(3-5):75–174, 2010. 4

[7] Thomas N Kipf and Max Welling. **Semi-supervised classification with graph convolutional networks**. *arXiv preprint arXiv:1609.02907*, 2016. 4

[8] **NumPy - The fundamental package for scientific computing with Python**. 5

[9] **Pandas**. 5

[10] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. **Machine learning at the energy and intensity frontiers of particle physics**. *Nature*, **560**(7716):41–48, 2018. 10

[11] Peter Sadowski, Julian Collado, Daniel Whiteson, and Pierre Baldi. **Deep learning, dark knowledge, and dark matter**. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, pages 81–87, 2015. 10

[12] Chiara De Sio. **Machine Learning in KM3NeT**. **207**:05004, 2019. 10

[13] Fernanda Psihas, Micah Groh, Christopher Tunnell, and Karl Warburton. **A Review on Machine Learning for Neutrino Experiments**. *arXiv preprint arXiv:2008.01242*, 2020. 10

[14] Adam M Terwilliger, Gabriel N Perdue, David Isele, Robert M Patton, and Steven R Young. **Vertex reconstruction of neutrino interactions using deep learning**. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2275–2281. IEEE, 2017. 10

[15] D Mulmule, PK Netrakanti, LM Pant, and BK Nayak. **Machine learning technique to improve anti-neutrino detection efficiency for the ISMRAN experiment**. *Journal of Instrumentation*, **15**(04):P04021, 2020. 10

[16] Rui Li, Z You, and Yumei Zhang. **Deep Learning for Signal and Background Discrimination in Liquid based Neutrino Experiment**. In *J. Phys.: Conf. Ser*, **1085**, page 042037, 2018. 10

[17] Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael Bronstein, Spencer Klein, and Joan Bruna. **Graph neural networks for icecube signal classification**. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391. IEEE, 2018. 10

[18] Max Neff, Gisela Anton, Alexander Enzenhöfer, Kay Graf, Juergen Hössl, Uli Katz, Robert Lahmann, and Carsten Richardt. **Signal classification for acoustic neutrino detection**. *Nuclear Instruments and Methods in*

Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, **662**:S242–S245, 2012. 10, 11

[20] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. **Convolutional, long short-term memory, fully connected deep neural networks**. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584. IEEE, 2015. 11

[21] Vincent A Traag, Paul Van Dooren, and Yurii Nesterov. **Narrow scope for resolution-limit-free community detection**. *Physical Review E*, **84**(1):016114, 2011. 12

[22] Yoshua Bengio. **Practical recommendations for gradient-based training of deep architectures**. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012. 16, 18

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 16

[24] Sebastian Ruder. **An overview of gradient descent optimization algorithms**. *arXiv preprint arXiv:1609.04747*, 2016. 18

[25] Amichai Painsky and Gregory Wornell. **On the universality of the logistic loss function**. pages 936–940, 2018. 18

[26] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. **Activation functions: Comparison of trends in practice and research for deep learning**. *arXiv preprint arXiv:1811.03378*, 2018. 18

[27] Gang Wang, Georgios B Giannakis, and Jie Chen. **Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization**. *IEEE Transactions on Signal Processing*, **67**(9):2357–2370, 2019. 18

[28] Paula Branco, Luis Torgo, and Rita Ribeiro. **A survey of predictive modelling under imbalanced distributions**. *arXiv preprint arXiv:1505.01658*, 2015. 19, 22

[29] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from imbalanced data sets*. Springer, 2018. 22

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting**. *Journal of Machine Learning Research*, **15**(56):1929–1958, 2014. 28

[31] Laurens van der Maaten and Geoffrey Hinton. **Visualizing data using t-SNE**. *Journal of machine learning research*, **9**(Nov):2579–2605, 2008. 29

[32] Liyu Gong and Qiang Cheng. **Exploiting edge features for graph neural networks**. pages 9211–9219, 2019. 35

[33] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. **An end-to-end deep learning architecture for graph classification**. 2018. 36

[34] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. **High performance graph convolutional networks with applications in testability analysis**. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019. 37

[35] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. **Neugraph: parallel deep neural network computation on large graphs**. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 443–458, 2019. 37

[36] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. **Accurate, efficient and scalable graph embedding**. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 462–471. IEEE, 2019. 37