Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Deep Learning for Neutrino Particle Detection using PointNet

Author: Shruti Rao (2636454)

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

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

December 7, 2020

Abstract

Particle physics involves examination of sub-atomic particles and their interactions. The main challenge in this field often lies in the separation of background noise from event signals. Most neural networks in the field use CNNs for particle classification. But this often leads to loss of information when converting data to images. This thesis examines the application of PointNet - a 3D classification network for KM3NeT neutrino data. The thesis has a two-fold interest. First, it wishes to investigate the role of 3D deep learning in neutrino identification. Next, it wishes to apply the network on KM3NeT data to save neutrino information while discarding background noise. The data is split into three datasets and trained individually. Feature engineering is performed and the resulting point clouds are converted to 3D meshes. A majority voting ensemble technique is used to combine predictions from the three models. The network showed promising results with a 95% recall for the positive class and perfect precision. The model also demonstrated perfect recall for the noise class. Being the first known work of its kind, results from the thesis indicate PointNet to be a viable methodology for future neutrino research.

Keywords: *PointNet*, *neutrino detection*, *classification*, *3D deep learning*, *KM3NeT*

Acknowledgements

I would like to thank Dr. Adam Belloum and Dr. Ben van Werkhoven for introducing me to the project topic and ensuring its finalisation. I would especially like to thank Dr. Belloum for his constant guidance and advice throughout the course of the thesis. Further, I would like to thank Dr. Roel Aaij, Dr. Ronald Bruin and Brían Ó Fearraigh from Nikhef for assisting with the Physics-based technicalities of the topic. Also, I would like to acknowledge Dr. Aaij's assistance with accessing the Viltstift AMD GPUs, that allowed for rapid experimentation. Finally, I would like to thank my co-worker Arumoy Shome for the exchange of ideas and discussions that enriched the thesis.

Contents

List of Figures

LIST OF FIGURES

List of Tables

1

Introduction

Physics has accounted for many fundamental properties of the universe. Yet, several questions regarding the elementary constituents of matter remain unanswered. For instance, it is well known that when neutron stars collide, they produce supermassive stars or black holes (2). However, there is not much information on what the cores of such stars or black holes comprise of. Similarly, it has also been established that majority of the universe is comprised of dark matter, with little indication as to what dark matter really constitutes (3). What is however known is that all of these phenomena have one particle in common - the neutrino (2). Neutrinos are elusive, weakly interacting particles that were discovered first by Pauli in the 1930s and the only known particle making up dark matter (4, 5). Understanding neutrinos has become increasingly significant for researchers in recent times as it represents much of the unknown universe. Experiments are being conducted to understand the mass of neutrinos, reasons for their oscillation, their ability to change forms, and the role they play in the birth and continuum of the universe (6).

With simultaneous advancement in hardware and computing power, the ability to detect and understand neutrinos has drastically increased (4, 5, 7, 8, 9, 10). Large particle accelerators are typical in particle physics experiments whereby protons and anti-protons are collided at high speeds to try to recreate neutrino particles. Through this process, petabytes of data are often produced and have to be analysed for signs of tracks, rings, jets and showers associated with neutrino interactions (11). These experiments so far have made use of physics algorithms that have worked well in detecting particles to a certain degree. However, these techniques fall short when it comes to identifying new particles or studying previously unknown behaviour that has not been defined by the algorithm parameters (12). Traditional algorithms are also unable to keep up with large volumes of

rapidly changing, high dimensional data. Thus, reliance on such algorithms have limited the potential for new discoveries (12).

On the other hand, Neural Networks (NNs) have faced several cycles of hype over the past decade or so. Early attempts at incorporating NNs were often unsuccessful due to limited understanding, large computation hours, hardware limitations, and lack of powerful architectures (13). Early applications that were developed were highly sensitive to errors and data quality (13). Due to these reasons, Neural Networks were not a favoured solution. But, recent advances in computing led to better storage of data, faster execution times and improved error handling. These factors directly contributed to enabling real-time processing of data. Also, since storage of large datasets was now possible, networks could be trained with larger datasets. Additionally, general theoretical research was advanced to address the concept of NNs being a black-box. Advances in computational theory led to development of powerful learning algorithms, optimisation techniques and robust architectures (14). These factors combined led to new interest in application of NNs to complicated problems, including those in particle physics (12).

1.1 Background

In order to understand the motivation behind the work done by this thesis and ongoing research, it is essential to understand the fundamental properties of neutrinos. It is also important to examine some significant existing applications of Neural Networks in working detectors. These can help identify scope for improvement for this thesis and for future research.

1.1.1 Neutrinos

Neutrinos are fundamental particles of the universe and quite different in nature from other commonly known particles. They carry no charge, are extremely small in mass and travel through matter undetected (4). Neutrinos come in three types (flavours) and can change between types and masses (4). On Earth, neutrinos are produced by nuclear reactors, natural radioactive changes in the atmosphere and particle accelerators. The Sun produces neutrinos via nuclear fission that occurs in its core. They are also generated from the births, deaths and collisions of stars and supernovae explosions (5).

Detecting neutrinos are extremely hard because they rarely interact. Around a trillion neutrinos pass through the Earth every second yet, approximately only one neutrino reacts with matter on Earth, once a day (7, 15). A neutrino travelling near the nucleus of an

atom emits weak bosons (W and Z type) (16). These in turn react with the nucleus of another atom to produce a multitude of particles, including charged particles which can be detected. However, the probability of the weak bosons hitting a nucleus of another atom is extremely small. This is because weak bosons have a very short lifetime of 1×10^{-27} seconds and travel a very short distance of less than 0.001 of the size of a proton (16, 17, 18). These two reasons make it very hard for charged particles to be produced and in turn for neutrinos to be detected.

There are a few ways of detecting neutrinos, with under-water detectors being most popular. In water, neutrino particles travel undisturbed and may travel faster than light in that medium. They may react with matter particles in water and create a charged *lepton* that produces a light known as *Cherenkov light*. These flashes of light are detected by photomultiplier (PMT) tubes that can infer direction, energy and flavour of the neutrino (17). *Super Kamiokande* is a water based detector that uses *Cherenkov light* to detect neutrinos (19). *IceCube* is another experiment located in the South Pole that uses a cubic kilometre of ice embedded with PMT tubes to detect neutrino events (9). *MiniBooNE* detector uses pure mineral oil that allows low energy muons and protons, invisible in water, to be detected (20).

Neutrinos are important because understanding their origin can help resolve many physics unknowns. Neutrinos could be used to probe and examine matter that present radiation cannot pass through. Since they travel through space practically unaffected, physicists believe that neutrinos can help learn about galactic cores and supernovae.

1.1.2 Neural Networks in Neutrino Physics

Neural Networks were first acknowledged in physics around 1988, in the field of particle physics (21). Particle physics comprises the study of the fundamental building blocks of nature. It largely involves low-level pattern recognition and physics process determination. According to Denby (1999), low-level pattern recognition tasks include finding tracks made by particles and process determination encompasses obtaining properties such as spatial topology and energy emissions of particles. Studies of such processes require work to be done either in real-time or offline. Denby (1999) described particle physics processes to be characterised by large magnitudes of background noise with small, rarer occurrences of real events at any given point in time (21).

Denby (1999) found Neural Networks in particle physics that have been used in both real-time and offline applications. Overall, NNs have had challenges being recognised as a statistical tool within the community of particle research. The main challenge in particle

physics lies in the fact that such experiments often have to deal with new and unknown phenomena. NNs in such instances have to be developed based on unknown, and guessed parameters. Models trained on such parameters then further reflect these unknowns and inaccuracies (21).

Despite these accepted fallacies, there are a few large-scale detector experiments that have incorporated Neural Networks. *Fermilab* has a muon trigger that applies low-level pattern recognition techniques (17). *Fermilab* also uses NNs to analyse proton-anti-proton collisions and measures top-quarks and lepto-quarks (17). The *Hera* accelerator has a prototype experiment that studies momentum from colliding particles (22) and a secondary experiment called *ZEUS* that uses a form of feedforward network to identify deeply inelastic neutral current events (23). The *CMS* experiment at the Large Hadron Collider (LHC) uses a neural network based trigger to identify electrons from protons (24).

1.2 KM3NeT

KM3NeT (Cubic Kilometre Neutrino Telescope) is a next generation neutrino telescope project, currently being constructed under the Mediterranean Sea [1]. The research unit comprises two main telescopes - the ARCA (Astroparticle Research with Cosmics in the Abyss) and the ORCA (Oscillation Research with Cosmics in the Abyss) (25). The ARCA telescope is to be used for extraterrestrial neutrino particles originating from the likes of supernovae or colliding stars (25). The ORCA telescope will be used to examine neutrinos travelling through the Earth's atmosphere (25). KM3NeT is another example of the underwater detectors discussed in Section 1.1.1, and relies on optical sensors to detect *Cherenkov light* (25). These highly sensitive photomultiplier tubes also acquire a significant amount of noise generated by other background factors, mainly Potassium-40 decay. The detector therefore records all photon particles as hits and solutions are in place to try and isolate relevant hits from noise hits.

1.2.1 Event Triggers

Data acquisition for the offshore KM3NeT detectors is based on a preset threshold, whereby all analogue signals that exceed the threshold are sent to shore. This is known as the level-zero (L0) filter. Next, event based level-one (L1) trigger is employed to filter out relevant events. Specifically, the occurrence of two or more L0 hits within a certain time and spatial distance are considered significant for L1 (25).

[1] https://www.km3net.org/

1.2.2 GPU Pipeline

A separate GPU-based data processing pipeline was proposed as a viable solution to the event-based triggers by Karas (2019). The pipeline comprises of three components. The first component of the pipeline implements hits correlation based on closeness of hits in time and space, given a threshold. The second phase of the pipeline uses the correlation from the previous component to cluster related hits together. The final phase of the pipeline then classifies communities into significant and insignificant ones, based on the presence of neutrino events (26).

1.3 Research Questions

Typically, particle physics research relies on building physics-derived algorithms to detect neutrinos. Previously described event triggers and the GPU pipeline for the KM3NeT are two such examples (Sec: 1.2). While these algorithms are physics driven, they may fall short when trying to capture complex information from imbalanced, skewed data. This could be due to three main reasons:

1. Often, physics algorithms make use of theoretical parameters and criterion that may not be optimised across varied datasets.

2. Algorithms designed to work on smaller, lower dimensional data may not be able to model increasingly higher dimensional data without suffering from information loss.

3. Several physics algorithms are highly customised and specific to the problem or data at hand, lacking in ability to be generalised (21).

Examination of literature on the topic has shown popularity of Convolutional Neural Networks (CNNs) for neutrino research. This also faces drawbacks, mainly from information loss and high compute times. Converting detector data to images and learning to detect neutrinos from them will lead to loss of information that may have been useful. It is imperative to understand if existing methodologies such as physics-based triggers and CNNs can be replaced by new models and data representations. Point clouds are the simplest representation of 3D geometric objects and could be one such means to represent neutrino data in a novel manner. PointNet, a point cloud specific architecture could be used to train on such data. The following research questions were formulated to address the feasibility of using PointNet for neutrino identification amidst background noise:

RQ1.0 *Can PointNet, a geometric neural network architecture be trained to identify timeslices that contain neutrino event hits from timeslices that contain only background noise?*

The KM3NeT detector gathers large volumes of data in real-time and sends it onshore in chunks of time (timeslices) (25). It would be computationally infeasible to store all of this data when most of it may be irrelevant to research. Therefore, an effective method is required to identify timeslices that contain neutrino hits so that only they may be saved and the rest gets discarded.

RQ1.1 *Can PointNet achieve a Recall score of greater than 0.9 for identifying timeslices with event hits?*

KM3NeT stakeholders defined 90% recall score for the model, which is the ability of the algorithm to learn from the data. To provide valuable contributions, PointNet would have to score 90% or higher for Recall.

RQ2.0 *Can the KM3NeT dataset be effectively represented using 3D meshes?*

PointNet typically learns from 3D meshes while the KM3NeT data comprises of `x, y, z` coordinates, `time` and metadata. If these attributes can be modelled to form a surface-based 3D mesh, it could serve as a suitable training data for PointNet as a 3D mesh stores more information about geometry than just 3D coordinates.

RQ2.1 *Which meshing algorithm would be most suitable for representing the data?*

The dataset is challenging as it does not contain any discerning features. Event hits are characterised by how close they occur in time and space, compared to noise hits. Yet this closeness is not very prominent. Conversion of points to 3D meshes will result transformation of data. It is essential to ensure that the transformation is such that much of the valuable information regarding event hits is maintained, if not enhanced. An ideal algorithm would allow minute differences amongst event hits to stand out more than noise.

RQ3.0 *Can PointNet be extended to obtain energy properties from neutrino events?*

Inferring energy from neutrino events can help identify and study the processes that formed it. PointNet is a classification specific architecture but as it learns from both local and global structure of point clouds, it may be able solve other tasks related to point cloud recognition including regression (1).

1.4 Research Outcomes

With research in particle physics turning to AI, a rigorous assessment of state-of-the-art networks is required to identify those best for the task at hand. This thesis aims to provide an in-depth investigation of one such state-of-the-art neural network - PointNet. This thesis contributes to the current state of knowledge by assessing the potential of 3D Neural Networks for research in neutrinos. In order to achieve this goal, an ensemble of data from the KM3NeT ORCA detectors comprising of neutrino hits and noise are examined. Noise from the data is minimised using feature engineering, converted to 3D meshes and trained with PointNet. The thesis also briefly explores the validity of PointNet with 3D and 4D coordinate data. Finally, the thesis lays the foundation for energy inference for KM3NeT data using non-linear regression techniques. At present, there are no known studies that use 3D deep learning to identify neutrinos amidst background noise. Moreover, no known studies attempt to represent neutrino data as 3D meshes.

Results from this thesis can not only be used by the KM3NeT project, but also by physicists interested in applying their own data to PointNet. It could also be used by deep learning experts to understand the gaps between what particle physics needs and what Neural Networks can deliver, and work on developing more streamlined solutions for the community.

The thesis first addresses concepts relevant to 3D point-set based learning and PointNet architecture in Chapter 3. Next, Chapter 4 describes data generation and insights gained from dataset exploration. Chapter 5 discusses individual components of the pipeline, Chapter 6 evaluates them and Chapter 7 analyses obtained results. Chapter 8 summarises additional research on an alternative PointNet pipeline with 3D and 4D coordinate data. This Chapter also briefly discusses introductory work on energy inference using regression techniques. The thesis discusses limitations and recommendations in Chapter 9 and concludes by re-addressing the research questions in Chapter 10.

2

Relevant Literature Study

Particle identification and categorisation is important in particle physics. Common practice for characterising such particles includes reconstruction of clusters, tracks, jets, rings and showers associated with particle interactions (27). Compared to several other domains that have seen significant adoption of deep learning, particle physics has remained relatively conservative in adopting deep learning models. This chapter summarises the few neutrino studies and experiments based on the type of network employed. Observations from these efforts serve as points of study for this thesis.

2.1 Feed-Forward Networks

Szadkowski et al. (2014) proposed a three-layer Neural Network (NN) to perform pattern recognition and classify proton old showers and neutrino young showers. Amidst background noise of cosmic rays, detecting the very infrequently occurring neutrino showers has been the main challenge for the field. The NN was set up to identify both young and old showers using simulated Monte Carlo events. The network was trained on 245,760 different patterns grouped into 160 events and presented extremely promising results. Noise was perfectly rejected and the NN was able to identify 161 patterns out of the 160, with a single false-positive. Thus on simulated data, the authors showed the ability of the NN to detect young showers with very low error rates (28). While Szadkowski et al. (2014) presented the first known implementation of NNs on neutrino data, the paper faced a few shortcomings. First, the paper did not indicate the rationale behind the selection of various hyperparameters, network architecture and activation functions. Next, the paper did not mention the nature of the test cases to indicate performance of the network across varied test data. Finally, only error rates were discussed as a metric however, precision-recall

(PR) and Receiver Operating Characteristics (ROC) curves could have proved a better measure of network performance in regard to the classification task it performed.

2.2 Convolutional Neural Networks (CNNs)

Detectors are often built to produce high-resolution images of particle movements and interactions. Acciarri et al. (2016) used CNN architectures to reconstruct neutrino scattering interactions in such images. They explored the use of CNNs for detector images that were very information sparse and often empty. 22,000 events per type of particle were used for training the CNN in batches. Both high and low-resolution images were provided as separate demonstrations to mimic realistic scenarios. The authors combined two CNN architectures - Faster-RCNN for particle detection, followed by AlexNet for particle classification (7). Results can be deemed quite promising as the authors noted the combined network's ability to distinguish track-like particles from shower particles very well. For high-resolution images, track-like particles had *87.2%* accuracy and shower-like particles had *81.3%* accuracy. For low-resolution images, the score was lower with *85.8%* accuracy for track-like images and *77.3%* for shower-like images (7). Based on the accuracy scores, it was concluded that there was reasonable localisation of both shower and track particles for high and low resolution images. However, the authors did not report precision-recall (PR) scores or the area under ROC curves that are imperative in classification scenarios.

Neutrino event classification experiments often also involve tagging and identification of on-beam event images for a neutrino interaction. Acciarri et al. (2016) additionally developed a methodology that identified neutrino interactions on single-plane images and cropped them around the interaction region. They then applied the network described in their previous work to classify particles in the cropped images (8). Acciarri et al. (2016) defined two classes for the classification task - Monte Carlo neutrino events and purely cosmic events and trained with InceptionResNet (8). The authors reported an *80%* accuracy score during training but reported lower validation scores of *78%*. Acciarri et al. (2016) additionally reported performance via selection efficiency for neutrino events to be a positive *80.1%*. They believed that this efficiency would improve if all three planes were used instead of a single plane (8).

Acciarri et al. (2017) continued on their study from (8) by extending their work from single-plane images to three-planes and combining it with optical detector data (29). High definition input images of simulated neutrino images and cosmic images (as background) were used. A new truncated network based on ResNet was designed. The authors discussed

the compromise of having fewer layers learning fewer filters, but preserving resolution, allowing for exposure to detailed features. Distribution of neutrino classification scores showed a very good separation between the two types of events. The selection efficiency improved to *85%* (from *80.1%* with one plane images) (29).

Image based detection and classification of neutrino particles were examined further through means of different representations of data and architectures. Adams et al. (2018) continued on the work of Acciari et al. (2016, 2017) by developing a Convolutional Neural Network that could predict objects in image data at the pixel level (30). Adams et al. (2018) trained U-ResNet, a deep semantic segmentation network via supervised learning. First, they used transfer learning techniques by training the first half of the network on the dataset from a previous work that contained single particle images (8). Then, they developed a new loss factor called pixel-wise loss (PL) weighing factor. This factor was multiplied by a single pixel's loss contribution to the total loss of an image. Thus, complex sections of the image obtained a higher weighted pixel loss, allowing the network to focus it's training on those regions. The process was monitored using the Incorrectly Classified Pixel Fraction (ICPF) metric. The ICPF indicates the average value of incorrectly classified pixel per image over all images on all events in a sample (31). The network was trained on 100,000 images and then tested on 20,000 images. U-ResNet achieved an average ICPF of *6.0* for electron neutrinos and *3.9* for muon neutrinos. They noted that U-ResNet could classify pixels from low energy and simple topologies fairly well with mean ICPF scores of *2.3* and *3.9* respectively. The authors additionally obtained real detector data and compared the results of their network with those obtained by physicists. However, physics methodology had a better, lower mean ICPF score of *1.8* for the electron samples while the network scored a mean ICPF of *2.6* (30). The authors justified this difference in performance to be due to lack of specialised physics determined features. Despite this, the study lays foundation for a new methodology for training and examining data from a new pixel level perspective.

Aurisano et al (2016) developed a technique called Convolutional Visual Network (CVN) based on CNNs to reconstruct neutrino energy and flavour. The authors designed the network to have two distinct views of the same image, rather than representing a single image in multiple colour channels. The network was trained using mini-batches on 3.7 million simulated neutrino events and tested on 1 million samples (27). To measure CVN's performance, it was first compared against existing metrics. Measurement-optimised efficiency scores were obtained from existing physics metrics and compared against that of the CVN. CVN scored an efficiency of *58%* versus the existing *57%* efficiency for muon neutrino

interactions. The authors felt that while the improvement was modest, it was still in the positive direction. CVN however scored *40%* efficiency over the pre-existing metric of *35%* for electron neutrinos. Additionally, the authors computed a Figure of Merit (FOM) to assess the performance of signal identification over background noise for oscillation parameters. Overall, the CVN obtained a range of efficiency scores from *17.4%* at the lowest to *66.4%* at the highest for various parameters. The authors found the results quite promising since they performed minimal event reconstruction and found positive performance with a single algorithm (27). Moreover, the CVN developed was used on atypical images, specifically the readout of a calorimeter. This study therefore opened up the possibility of using a different form data which might be extendable to other detectors as well

2.3 Graph Neural Networks (GNNs)

IceCube is a neutrino observatory at the South Pole that searches for high energy neutrino events (32). It observes two classes of such events - neutrino interactions within the detector and high energy cosmic interactions in the upper atmosphere (32). Choma et al.(2018) in their study on data from IceCube proposed that the irregular geometry of the detectors could be modelled as a graph with vertices as sensors and edges as learned functions of the sensors' spatial coordinates. They stated large asymmetry between positive and negative events to be the main challenge. The authors proposed the use of Graph Neural Networks (GNNs) for this work. The authors considered muon neutrinos as positive signals and the rest as background noise. As the background was much larger in terms of magnitude than the signal, a high rejection power was required. The GNN was initialised as a fixed, weighted, directed graph. 25,250 events were generated as signal and 109,491 events were generated as background. The performance of the classification algorithm was noted against physics results and CNN scores (33). As per Choma et al. (2018), physics-derived metrics reported *0.987* signal to noise ratio for events per year and CNN reported *0.937* signal-to-noise ratio. The GNNs showed clearly superior results by reporting *2.980* signal-to-noise ratio for events per year. The GNN outperformed physics metrics by identifying three times more positive events (33).

Based on studies discussed, many Neural Networks trained for neutrino detection are variations of CNNs (12, 28, 34). Most of the problems discussed involved identification of particles from background noise and then classification, often in the form of images (7, 8). However, typical CNN architectures requires conversion of data to images, resulting in significant volumes of data (1). Further, in trying to quantize data, several unwanted

features (artefacts) may render on the images distorting results (1, 35). In order to avoid such pitfalls, this thesis makes use of PointNet, a network that directly consumes point clouds without requiring conversion to a regular input like 3D voxels (1). Due to the varied nature of the datasets and questions being answered, different metrics got reported. None of the studies reported Precision-Recall (PR) scores that are useful to understand the learning abilities of classifiers (36). This thesis instead focuses on measuring recall, especially for the positive (events) class and employs a combination of metrics to ensure that any effects of class imbalance are mitigated.

3

Relevant Concepts

Point clouds are collections of points in space that represent geometric shapes in their simplest forms, so data structures they are a set of unordered vectors of points (37). PointNet expects its input points from Euclidean Space, hence this section addresses key concepts pertaining to point sets in $\mathbb{R}^n$. Understanding the properties of point sets is essential in order to understand the suitability of KM3NeT data and the rationale behind data preparation. This section also discusses key concepts underlying PointNet architecture.

3.1 Properties of Input Point Sets

Qi et al. (2017) identified three main properties that input point sets must follow.

1. **Unordered**: Point sets are unordered and so 3D input data accordingly must be invariant to $N!$ permutations in the order that it is provided to the network.

2. **Interaction among points**: Point sets originating from a space with a distance metric demonstrates meaningful relationship amongst neighbouring points. This will allow PointNet to capture and learn local and global structures.

3. **Invariant under transformations**: Unordered point sets should be invariant to transformations. Learned representations such as rotation or translation of input data should remain unaffected by the applied transformations in terms of global structures (1).

3.2 PointNet Architecture

According to Qi et al. (2017), PointNet can perform both classification and segmentation tasks. This thesis only focuses on the classification network due to its relevance to the research problem. Figure 3.1 shows an overview of the PointNet architecture. The classification network accepts `n` points as input and applies input and feature transformations. A shared multi-layer perceptron (MLP) is used to map points from the `x, y, z` dimensions to 64 dimensions. This step is duplicated to then map points from 64 dimensions to 1024 dimensions. Next, point features are aggregated though *max pooling* to generate a global features vector in $\mathbb{R}^{1024}$. Max pooling is a technique that is used to down-sample input via sample discretization to a more abstract output (38). Finally, a MLP is used to map global feature vector to k classification scores (1).

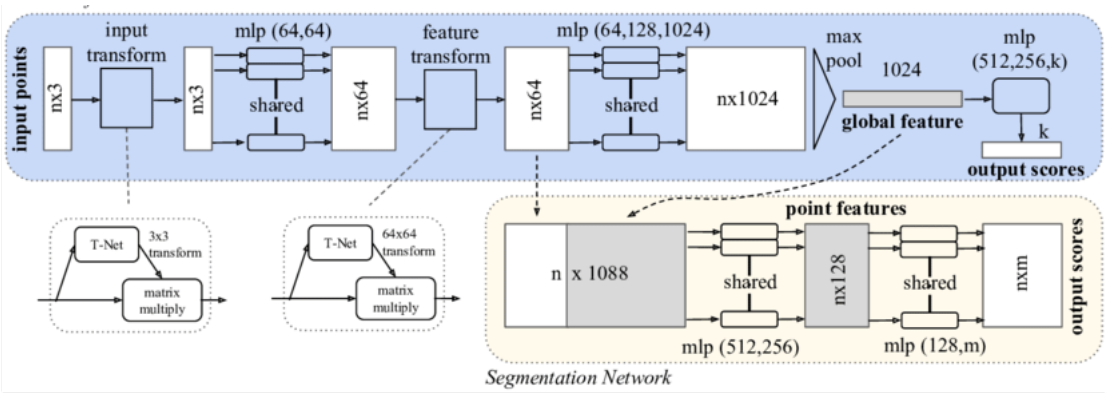


Figure 3.1: PointNet Architecture for Classification and Segmentation (1). *mlp* indicates multi-layer perceptron; numbers in brackets are the layer sizes.

Feature and input transformations are significant to the architecture, and is additionally described under permutation and transformation invariance.

3.2.1 Permutation Invariance

Point clouds are unstructured, numerical sets and invariant to permutations ie., for N points, there are $N!$ valid permutations (section 3.1). Symmetric functions are therefore used to make PointNet invariant to input permutations (1). Specifically, max pooling is used when `n` input points are mapped to higher dimensional space and then used directly for classification (1). This can be seen in Figure 3.2

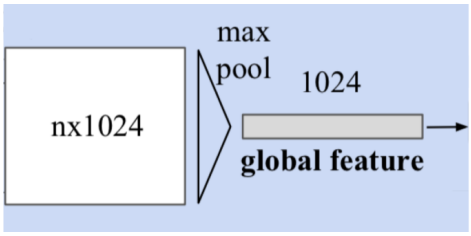
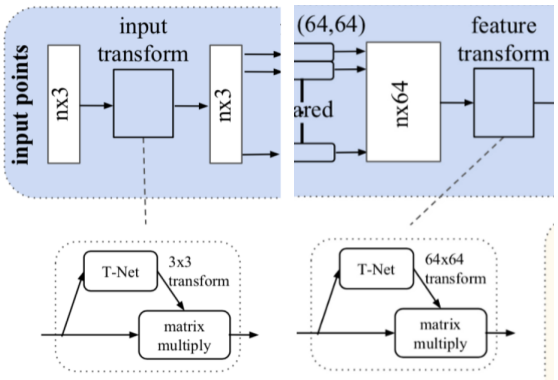


Figure 3.2: Symmetric Function with Max Pool Applied to Global Features (1)

3.2.2 Transformation Invariance

Point clouds must be invariant to certain geometric transformations (section 3.1). If point clouds undergo transformation, then it is necessary that the classification labelling must be invariant as well. To ensure that this is the case, input and feature transformation sub-networks are used to provide normalisation to objects. *T-Net* - a regression network is tasked with predicting a $n \times n$ transformation matrix.



(a) Input Transformation

(b) Feature Transformation

Figure 3.3: Input and Feature Transforms within PointNet

For input transformations, n input points are represented as a vector and mapped to the embedding spaces. Geometric transformation then becomes easy to apply and involves matrix multiplying each point with a transformation matrix. Here, the T-Net predicts a 3×3 transformation matrix, which is then matrix multiplied with the $n \times 3$ input (Figure 3.3).

For feature transformation of the 64-dimensional embedding space, the corresponding T-Net predicts a 64×64 transformation matrix (Figure 3.3). The increased number of trainable parameters leads to the potential for overfitting and instability during training,

so a regularisation term is added to the loss function to constrain the feature matrix to be close to the orthogonal matrix:

$$\mathcal{L}_{reg} = ||I - AA^T||_F^2 \tag{3.1}$$

where A is the matrix predicted by the T-Net.

So, PointNet is capable of learning from unordered, raw point clouds as long as the properties of point sets are maintained. This is relevant to the thesis and validated when preparing training data. Additional transformation functions within the network further ensure that the data respects invariance to permutations and transformations. These concepts are used by this thesis in the upcoming pipeline to ensure absolute accordance with rules governing point sets.

4

Data Generation and Exploration

Nikhef provided simulated data from three sources that were used to generate the complete KM3NeT dataset - the *K40 noise generator*, *HDF5 hits and events* tables and a *positions* file. This chapter describes the steps taken to combine noise and events data to produce the complete dataset. The chapter then discusses the quality and properties of the complete dataset by means of visual data exploration.

4.1 Noise Generation

Noise for the dataset was generated using the *k40gen* [1] package. *k40gen* is a standalone background noise generator developed by Nikhef to generate a random array simulating Potassium-40 decay underwater. An instance of the generator was created with two random seeds - `21341` and `1245`. `7000`, `700`, `70`, and `0` were specified as the rates at which single, double, triple, and quadruple hits were to be generated respectively. Noise hits were generated from `0` nanoseconds (ns) till `100000000` ns. The ORCA (Oscillation Research with Cosmics in the Abyss) counting scheme was also provided as a numbering scheme for the photomultiplier (PMT) IDs within the Digital Optical Modules (DOMs). The DAS-5 cluster [2] was used to install and generate the background noise array, totalling 5GB. The returned array contained information regarding `time`, `DOM ID`, `PMT ID` and `time-over-threshold` for noise hits.

Additionally, the positions file (`positions.detx`) contained all the spatial positions corresponding to the noise hits from the *k40gen* array and had to be merged. Spatial positions

[1] https://gitlab.nikhef.nl/roelaaij/k40gen
[2] https://www.cs.vu.nl/das5/

from `positions.detx` were identified and mapped to the corresponding noise hits using the following formula:

$$noise[pos\_idx] = 31 \times (noise[dom\_id] - 1) + noise[pmt\_id]) \tag{4.1}$$

Finally, a class label of `0` was also added for all hits to indicate noise for future classification.

4.2 Event Hits Generation

Neutrino events data was stored in a HDF5 (Hierarchical Data Format version 5) file [1] consisting of an event hits table (`mc_hits`) and a corresponding event information table (`mc_info`). `mc_hits` included information pertaining to the DOMs (`dom_id`), PMTs (`pmt_id`), positions (`pos_x, pos_y, pos_z`) and directions (`dir_x, dir_y, dir_z`) of event hits. It also contained time over threshold (`tot`) values and the time the hits were registered (`time`). A class label of `1` was added to this data to indicate an event hit. Only the `mc_hits` table was used for the classification project undertaken in this thesis.

4.3 Data Combination

Before the background noise and event hits data could be combined, a few preliminary cleanup and validation measures were taken. First, it was noticed that the PMT IDs for event hits were from `0` till `6417`. This indicated that it followed a global numbering scheme, while PMT IDs for noise hits had been generated using the ORCA numbering scheme. To enforce uniformity, PMT IDs for event hits were set from `1` till `31`:

$$pmt\ id = pmt\ id - 31 \times (dom\ id - 1) \tag{4.2}$$

Next, `time` for some noise hits were negative, so these data points were deemed invalid and deleted from the dataset. As both events and noise data were simulated, it was important to ensure that the `time` values associated with event hits were not biased. To check for such a bias, the mean time of occurrence for all event hits was calculated and plotted for any evidence of hits clusters. Based on the evenly distributed scatter of points in Figure 4.1, it was concluded that there were no biases in the data. Noise and event hits were merged to form the complete KM3NeT dataset after negative time was deleted, PMT IDs were adjusted and data validity was confirmed.

[1] https://www.hdfgroup.org/solutions/hdf5/

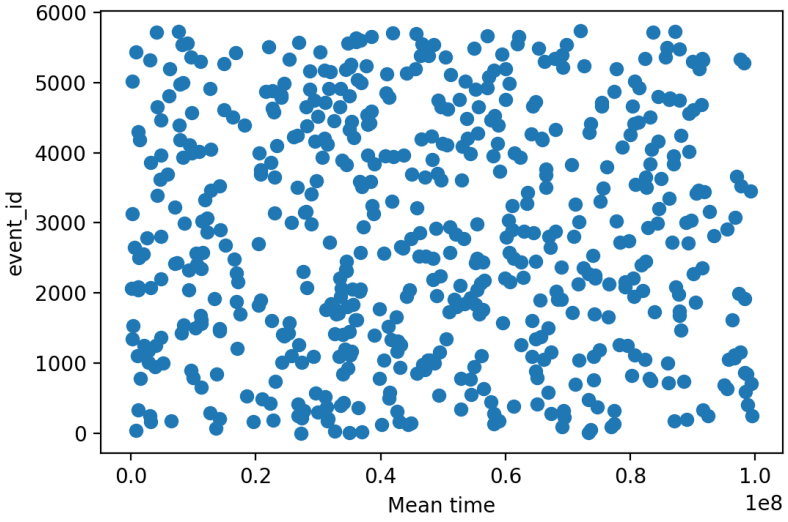


Figure 4.1: Scatter Plot of Mean Time for All Event Hits

4.4 Key Attributes

The KM3NeT detector records and sends data in chunks (timeslices) for processing at the offshore facility (25). The thesis's main aim was to train a network that would classify timeslices into those that contained only noise and those that contained event hits amidst noise. Therefore, the dataset was sequentially binned into groups of 15000 ns and assigned a number for identification. For example, timeslice `0` contained all hits that occurred between 0 ns and 15000 ns. 15000 ns was used for binning as neutrino events typically occur between 100 ns and 15000 ns (25). Selecting a value on the higher end of the range allowed for fewer groups to be created and by extension, faster processing. The binning resulted in a total of 6759 groups. The complete dataset was found to be 4GB in size, with 45,820,220 rows and 12 attributes, summarised in Table 4.1.

Domain knowledge from Nikhef indicated `dom_id, pmt_id` and direction of PMT tubes (`dir_x, dir_y, dir_z`) to be metadata that could be ignored. Additionally, work by Karas (2019) identified time-over-threshold (`tot`) to be insignificant to the classification problem (26). The remaining attributes - spatial coordinates (`pos_x, pos_y, pos_z`), `time`, `label` and timeslices (`group`) were identified as key variables for the remainder of the classification project.

Attribute	Description
`dom_id`	[Unique ID for sensor module.]
`pmt_id`	[Unique ID for photomultiplier (PMT) tubes within DOMs.]
`pos_x, pos_y, pos_z`	[Spatial coordinates (in meters) of hit within the detector.]
`dir_x, dir_y, dir_z`	[Direction of PMT tubes within DOMs to look for Cherenkov Light.]
`tot`	[Time-over-threshold (ToT) indicates the amount of light transformed to charge which is interpreted as the length of the square wave pulse over a given threshold (26).]
`time`	[Time at which the hit was recorded.]
`label`	[0 or 1 class label indicating whether hit is from noise or event respectively.]
`group`	[Timeslice numbers starting from 0 for the purpose of identification.]

Table 4.1: KM3NeT Data Attributes and Description

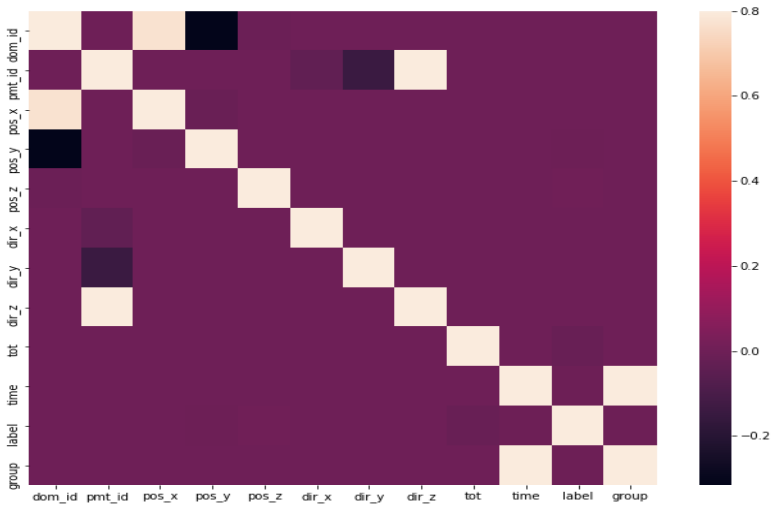


Figure 4.2: Correlation Heatmap Between Variables

4.5 Visual Analysis

Python libraries were used to explore the quality of the dataset and gain further insight on the key variables and the relationships that may exist between them. The dataset was found to have no missing values or outliers. All hits occurred between `0` ns till `101591357` ns. The dataset showed severe imbalance between event hits and noise as, for every 1 event

hit, there were 93 noise hits. Finally, the correlation heatmap in Figure 4.2 indicated that the key variables had no relevant relationship with each other. Significant variables were further examined to identify existence of useful properties.

Timeslices (`group`):

Timeslice `0` was found to have the highest number of hits (12,454), all of which were noise. Timeslice `615` contained the highest number of event hits (around 8500) with the lowest noise-to-event hits ratio of 4:1. Figure 4.3 shows the distribution of number of points per timeslice. Timeslice `0` had the highest occurrence of hits while some timeslices contained very few hits. These timeslices were edge cases and were ignored for the rest of the project. However, majority of the other timeslices contained between 6000 to 8000 hits, providing a more or less consistent sample for the training.

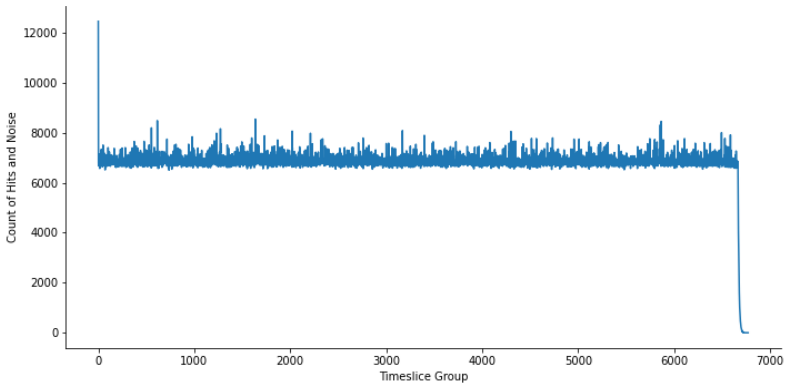
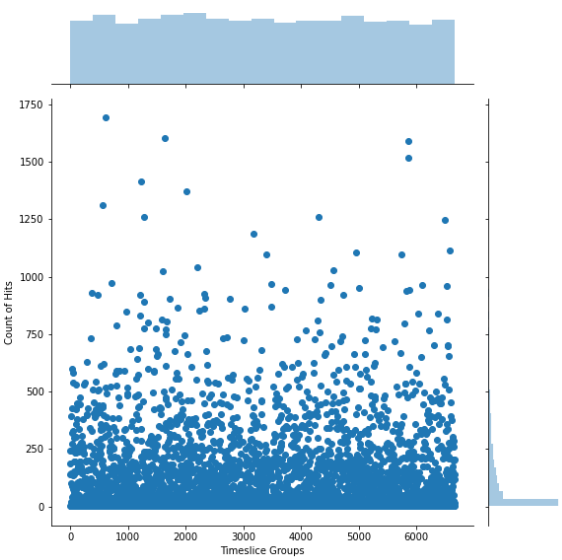
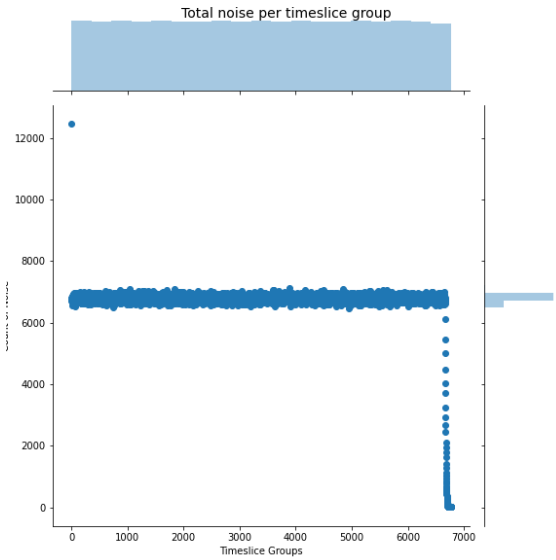


Figure 4.3: Count of Total Points per Timeslice

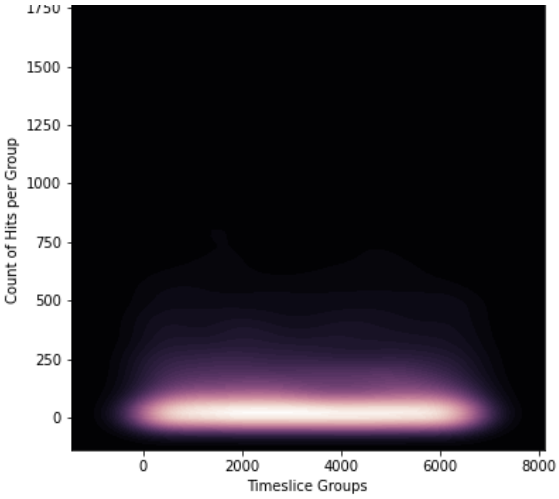
In accordance with the classification task, timeslices were separated into those that contained only noise and those that contained both event hits and noise. First, class imbalance between event hits and noise within timeslices were visually examined. Figure 4.4a shows that most timeslices with both events and noise had between 0 to 250 event hits. On the other hand, Figure 4.4b shows noise timeslices had around 7000 points on average. The outliers in this Figure (4.4b) demonstrate a timeslice that had a significantly large number of noise hits and timeslices towards the end that had comparatively fewer noise hits. These anomalies could be attributed to the nature of the *k40gen* random noise generator. Figure 4.4c and Figure 4.4d use kernel density estimation (KDE) to produce a continuous density estimate using a Gaussian Kernel and provides an alternate visualisation of the severe class imbalance by highlighting the densest regions (39).



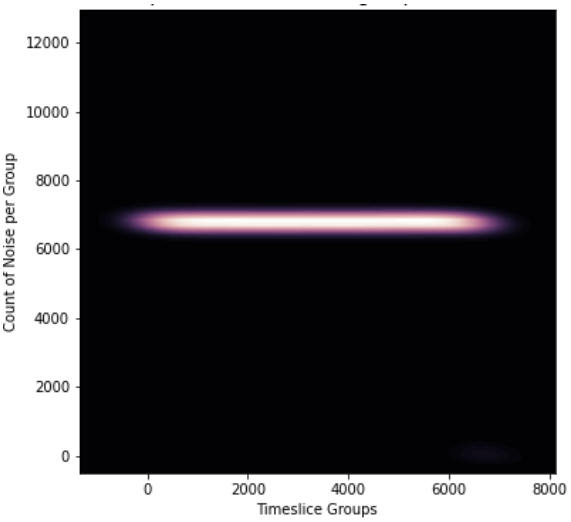
(a) Count of Event Hits For Timeslices with Both Noise and Events



(b) Count of Noise Hits For Timeslices with Only Noise



(c) Kernel Density Estimation for Only Event Hits



(d) Kernel Density Estimation for Only Noise Hits

Figure 4.4: Comparing Count of Hits and Noise Between Timeslices

3D Spatial Coordinates (`pos_x, pos_y, pos_z`):

Figure 4.5 uses a "swarm" technique to show distribution of noise and event hits across time for timeslice `615`, the group with the highest incidence of event hits. The points are non-overlapping and adjusted to better show distribution (39). Orange points indicate that event hits occur in groups coinciding with neutrino events amidst the noise.

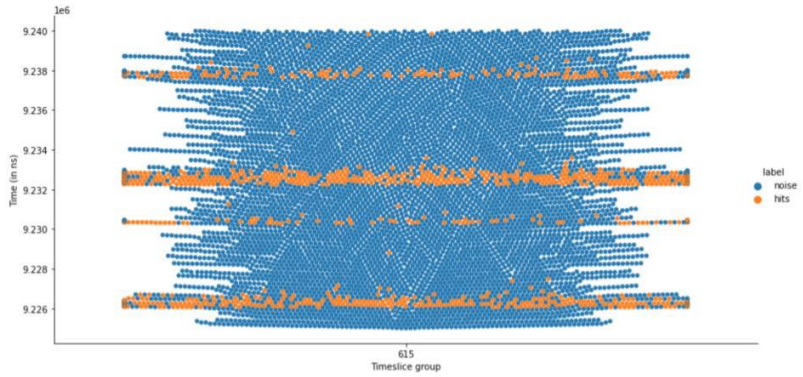
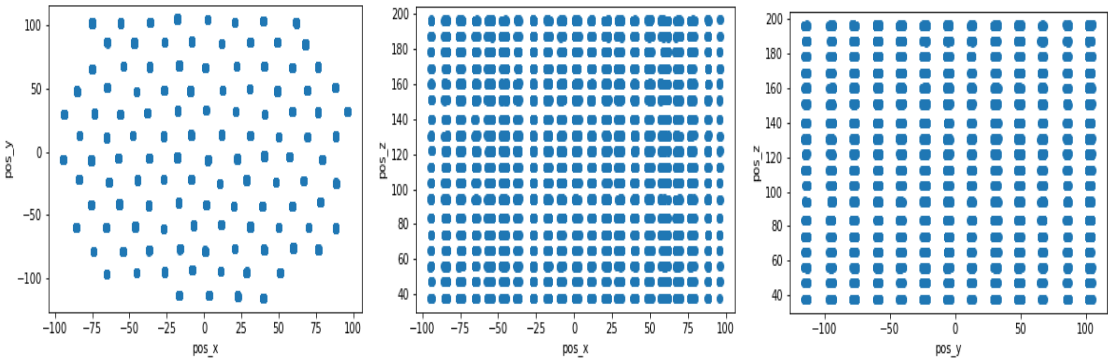


Figure 4.5: Distribution of Noise and Event Hits in Timeslice 615

Bi-variate plots in Figure 4.6 show the relationship between the `x, y, z` points for timeslice `615`. These 3D coordinates don't give much information or patterns that may indicate presence of event hits, confirming the need for `time` as part of the dataset.



(a) Scatter plot for `pos_x` against `pos_y`

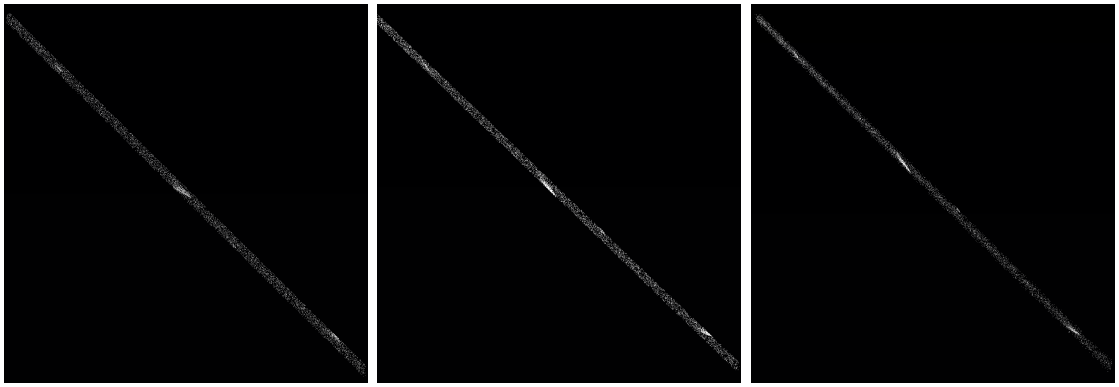
(b) Scatter plot for `pos_x` against `pos_z`

(c) `pos_y` against `pos_z`

Figure 4.6: Bi-variate Plots Showing Relationship Between 3D Coordinates

Figure 4.7 shows an enhanced 3D render of timeslice `615` generated using MeshLab [1]. The timeslice is first visualised as `x`, `y` and `time`, then `x, z` and `time` and finally `y, z,` and `time`. The three representations show no relevant differences between each other. However, visual enhancements via MeshLab allow for event clusters to stand out, as indicated by the bright clusters of points.

[1] https://www.meshlab.net/

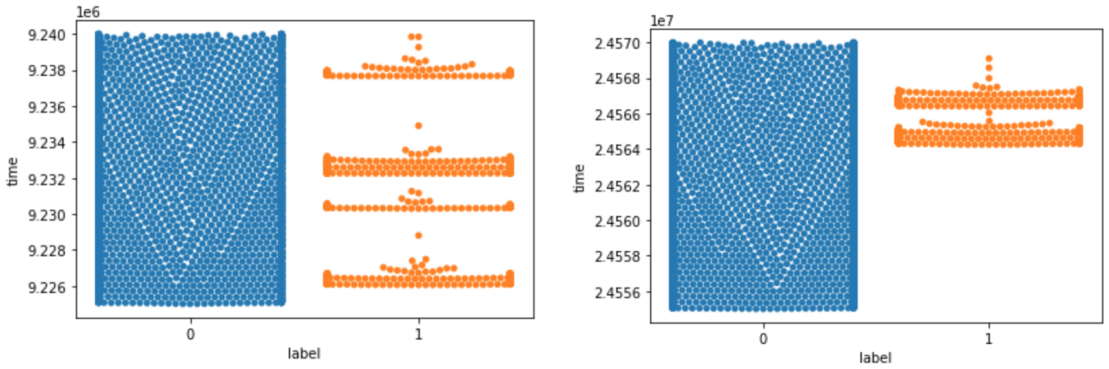


(a) 3D Plot of `x, y, time` **(b)** 3D Plot of `x, z, time` **(c)** 3D Plot of `y, z, time`

Figure 4.7: 3D Plots Showing Combinations of `x, y, z, time` for Timeslice 615

Time (`time`)

Time plays a significant role in the identification of neutrino hits amidst noise. A swarm plot of `time` against `label` was plotted for two events timeslices to note the relation between time and type of hit (40). Figure 4.8 shows an even distribution of noise hits across time, while event hits again form localised clusters.



(a) `time` and `label` for Timeslice 615 **(b)** `time` and `label` for Timeslice `1637`

Figure 4.8: Swarm Plots Showing Relationship Between Time and Hit Labels

Based on the combined results from plots of spatial positions, `time` and `label`, it was evident that a combination of these attributes would be required to obtain sufficient information on event hits. Further, the dataset had unrelated variables, lacked significant patterns, and had a high density of points. It would be very likely that the network would not be able to learn anything significant. Therefore, feature engineering may be required to add new information to the dataset before training could occur.

5

The Pipeline

The design goal of the pipeline is to accept timeslices as input and output a `0` or `1` indicating whether they need to be discarded or saved respectively. Two classes were identified: `class_0` comprised timeslices with just noise and `class_1` included timeslices with event and noise hits. Henceforth, the thesis refers to `class_1` timeslices as event timeslices, but they contain both noise and event hits.

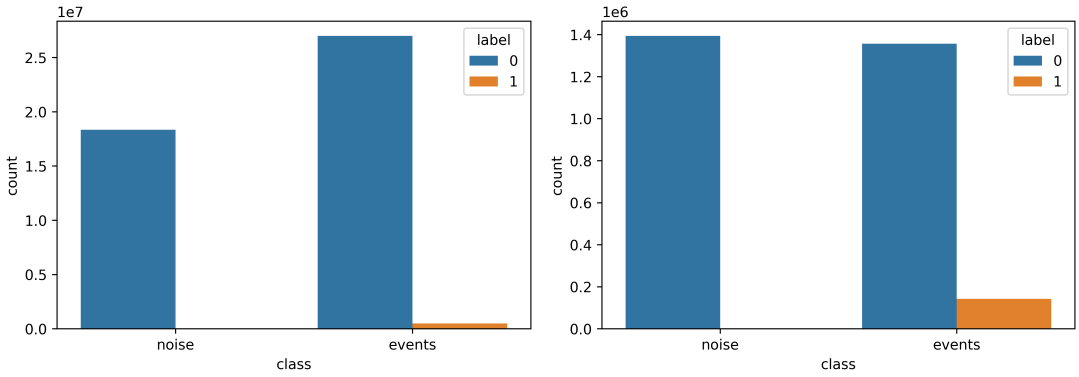
Stakeholder knowledge and data exploration (Chapter 4) identified 6 key variables - `pos_x, pos_y, pos_z, time, group` (timeslice) and `label`. Since neutrinos are identified by both spatial and time differences between each other, an ideal dataset would contain all four variables (25). However, generating a 4D mesh with `time` was not feasible, as the process is complicated and relatively under-developed (41). Instead, `time` was combined with spatial coordinates to create three permutations of the dataset - (`x, y, time`), (`x, z, time`) and (`y, z, time`). The following chapter discusses each component of the pipeline in detail. All three datasets were processed and transformed in the same manner, so details described in the following pipeline are applicable to all three datasets.

5.1 Generation of Point Clouds

The first step in the pipeline was to build 3D point clouds for each timeslice and save them as `.xyz` files. In order to do so, a few preliminary steps were undertaken. First, timeslices were divided into two classes whereby one contained only noise and the other contained both noise and event hits. Most timeslices with only noise had an average of 6500 hits per timeslice, but there were a few timeslices that had a single noise hit. Likewise, timeslices with event hits had an average of 6900 hits per timeslices, but some timeslices had only 3 event hits. Discussion with domain scientists indicated that such groups represented an

unrealistic scenario. Therefore, they were excluded from the data as they would not be able to provide a quality training sample.

Figure 5.1 shows class imbalance before and after selection of the largest timeslices per class. It shows imbalance both across the noise and events class; and also within the events class. Sub-figure 5.1a shows that the noise timeslices had fewer noise hits compared to the noise hits in event timeslices. Moreover, within the event timeslices, there is significant disparity between the number of noise and event hits. Sub-figure 5.1a indicates that the class imbalance needed to be improved as it may affect training and bias the classifier towards the majority class. Further, the entire dataset was not required for training. So, in order to continue with a smaller, quality training data, the top 200 timeslices were taken for each class. For the events class, timeslices were sorted in a descending order based on the number of event hits they contained. The largest 200 of these timeslices were then selected. Similarly, the top 200 noise timeslices were also selected.



(a) Count of Noise and Event Hits from All Timeslices

(b) Count of Noise and Event Hits for 200 Noise and Event Timeslices

Figure 5.1: Class Imbalance Before and After Selection of Top 200 Timeslices Per Class

Sub-figure 5.1b shows the class imbalance after selecting the top timeslices. The noise hits are now equal between the two classes. Further, there is some improvement between the event and noise hits within the events class. This indicated that taking the largest timeslices helped make the dataset more uniform and balanced for training.

Data for the 200 event timeslices were further observed. It was seen that there were a total of 27,485,996 hits, of which only 489,906 were event hits. Positive hits formed only 1.78% of the data. The maximum number of hits a timeslice was 1692 and the smallest number of hits in a timeslice was 487. On average, timeslices had around 700 event hits. In contrast, these timeslices had 7000 noise hits on average. All timeslices were saved as

point clouds under their respective classes - `class_0` comprising of noise point clouds and `class_1` comprising of event and noise point clouds.

5.2 Feature Engineering

Feature engineering is the process through which data can be transformed in a manner that either brings new information to light or provides more structure to the data, making learning easier for the network (42). Feature engineering was included in the pipeline due to three main reasons.Visual data exploration (Section 4) revealed unrelated variables with no significant patterns. Next, visual inspection of both point clouds and their corresponding 3D meshes showed lack of potential structure. Finally, preliminary results from training on data without feature engineering showed the need for additional information (Appendix 10).

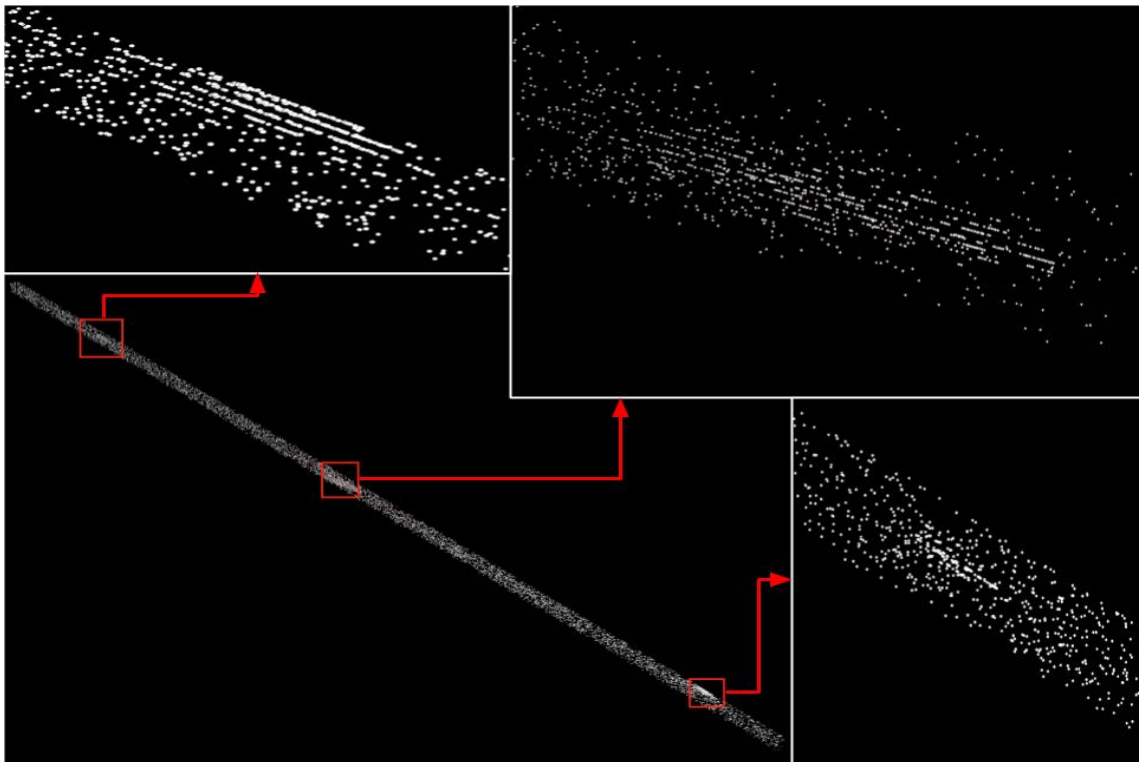


Figure 5.2: Point Cloud of the Largest Event Timeslice

Figure 5.2 shows a timeslice that contained the highest number of event hits. The point cloud has three event regions highlighted in red that are very hard to identify. These

regions were visualised under a highly zoomed perspective using MeshLab [1] and shows very minimal differences between event hits and the surrounding noise (43). Figure 5.3 on the other hand shows a point cloud of the timeslice with the highest number of noise. Based on the observed point clouds, it was decided that the data could be enhanced by elimination of points. Certain noise hits could be identified as outliers and removed in order for clusters of event hits to stand out more.

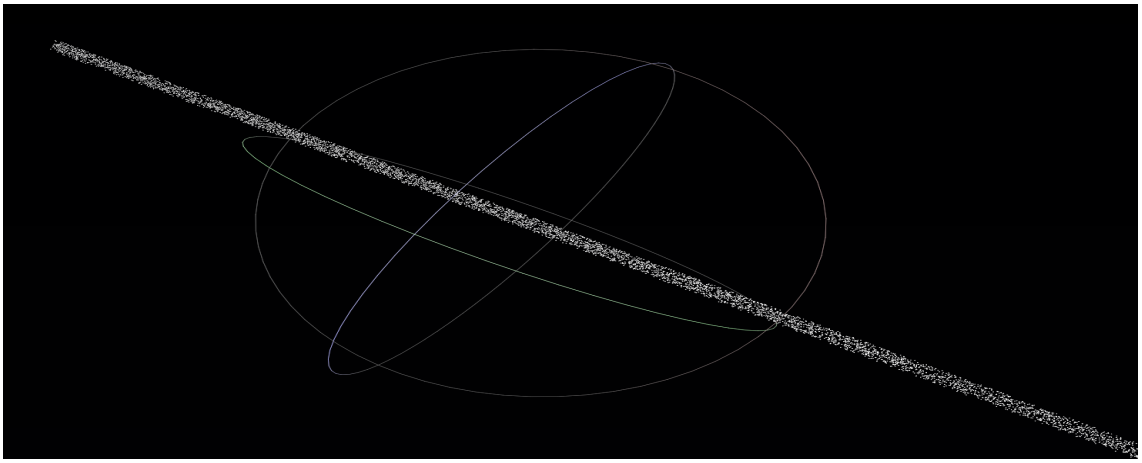


Figure 5.3: Point Cloud of the Largest Noise Timeslice

Removing outliers from point clouds are a common task and often manually removed using point cloud processing software (43). Surface fitting-based methods are a popular means of outlier removal. Here, a triangulated surface gets generated and criterion such as Moving Least Squares with Lagrangian operators are used to identify "rough" features from irregular regions (44). Discontinuous operators-based method is another reliable technique whereby regions of points are detected using density depth maps and removed based on visibility conflict (45). All methods described have the advantage of being conceptually accurate (46). Discontinuous operators based methods however work only on specific outliers and are inapplicable elsewhere (45). Surface based methods can work well on logical shapes but may not make sense for irregular, unknown shapes. They can also be very time consuming due to the typically large size of point clouds (44, 47). This thesis required a technique that could work well on irregular point clouds. Moreover, the focus was on providing fast processing, rather than precise outlier detection. This is because the transformed point clouds would ultimately be passed onto PointNet that should perform additional computations to differentiate between the two classes. Based on these

[1] https://www.meshlab.net/#description

requirements, a simple radius-based outlier detection method was finalised (46).

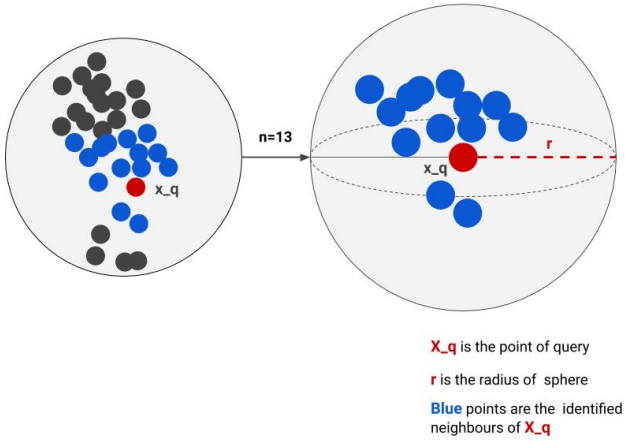


Figure 5.4: Demonstration of the Principle of Radius-Based Outlier Removal on A Set of Points

The principle of radius-based outlier filter (RBOF) is that if the number of points in the sphere of radius `r` centred at the query point `X_q` is lower than a threshold `n`, then `X_q` will be marked as an outlier and removed (48). Figure 5.4 explains the concept of radius-based outlier removal where the red point (`X_q`) is the point of interest. The number of neighbours is specified to be 13. Based on a specified radius `r`, there are 13 blue points that lie within the sphere of `X_q`. Thus, `X_q` is identified as outlier. Visual analysis of point clouds such as in Figures 5.2 and 5.3 helped determine that noise timeslices had roughly evenly spaced points. Therefore, the goal of RBOF was that fewer outliers would be detected and removed. Event timeslices however showed clustering of points, so here, the algorithm would detect and remove more outliers, especially the noise hits around the event clusters.

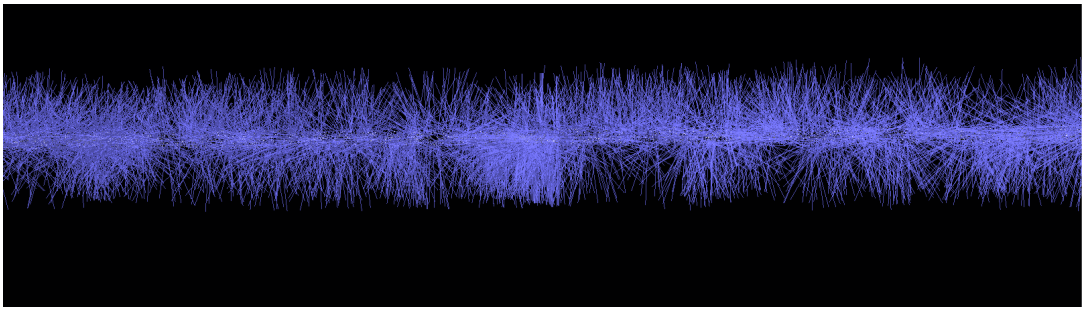


Figure 5.5: Normals computed for a Timeslice with Nearest Neighbours Specified as 10

Open3D[1] was used to identify outliers and perform feature engineering. First, all `.xyz` files from each class were transformed from an array of 3D coordinates to an `o3d.geometry.PointCloud` geometry. Next, normals were computed for each point because surface reconstruction based 3D meshing requires normals of the point cloud (49). The number of neighbours used to estimate normals was set to `300`, due to the large number of points per point cloud.

Figure 5.5 shows the normals for a timeslice with the largest number of event hits, when computed with the default `10` neighbours. The purple markers indicate the direction of the normal. Ideally, the markers for all points must face in the same direction, which is not the case in Figure 5.5. Figure 5.6 shows the same timeslice, recomputed with `300` nearest neighbours. The markers now approximately point in the same direction. The highlighted regions in the figure correspond to event hits and show interesting behaviour. The normals computed at these locations all point at different directions.

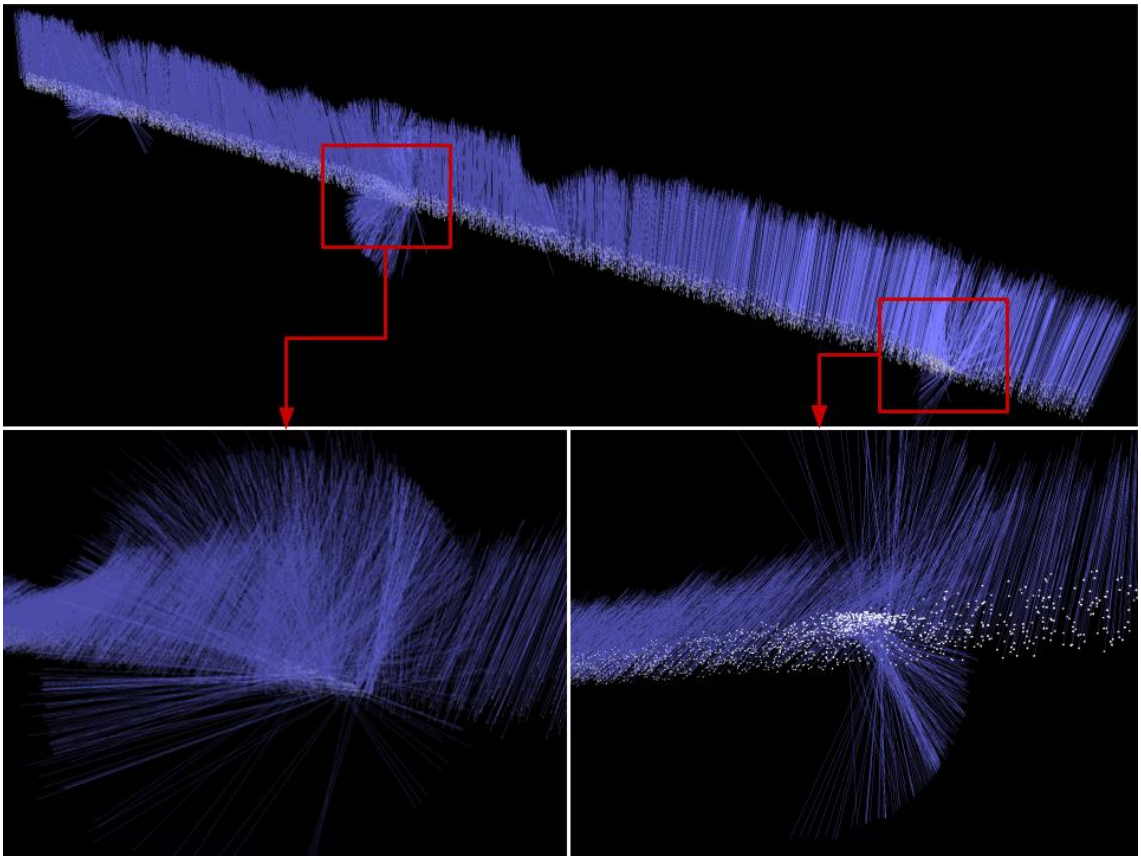


Figure 5.6: Normals computed for a Timeslice with Nearest Neighbours Specified as 300

With the point clouds normalised, the radius-based outlier detection was finally per-

[1]http://www.open3d.org/

formed. The minimum amount of points that the sphere should contain (`nb_points`) was left as the default value of `32` neighbours. The `radius` of the sphere that would be used for counting the neighbours was set to vary for each point cloud because of their irregular shapes and densities. A fixed radius value would not generalise well and produce inaccurate results. In order to identify a suitable radius, the nearest neighbour algorithm was used to compute the distance from a point to its nearest neighbour in the point cloud (50). Once an array of all the distances were obtained, they were averaged and multiplied by a factor of 3.6 to account for scaling (51).

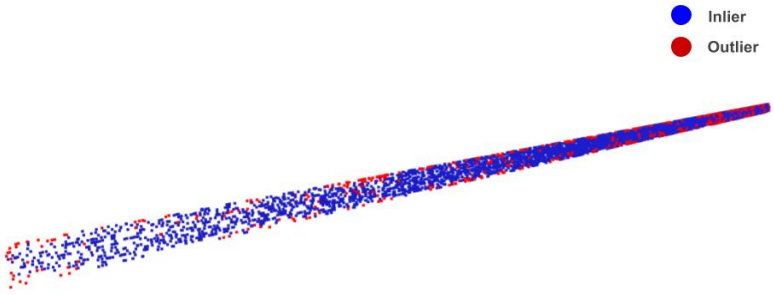
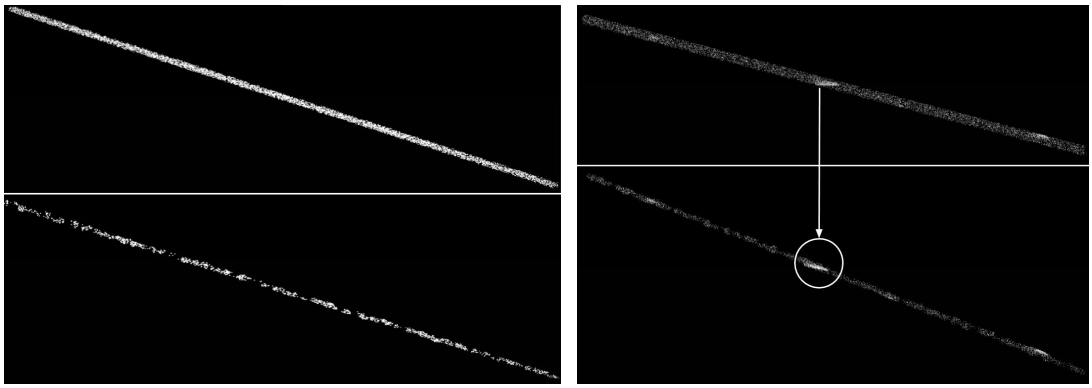


Figure 5.7: Result of Radius-Based Outlier Detection on Timeslice with Only Noise

Figure 5.7 shows the result of applying the outlier removal on a timeslice with only noise. The algorithm identified inliers in blue and outliers in red.



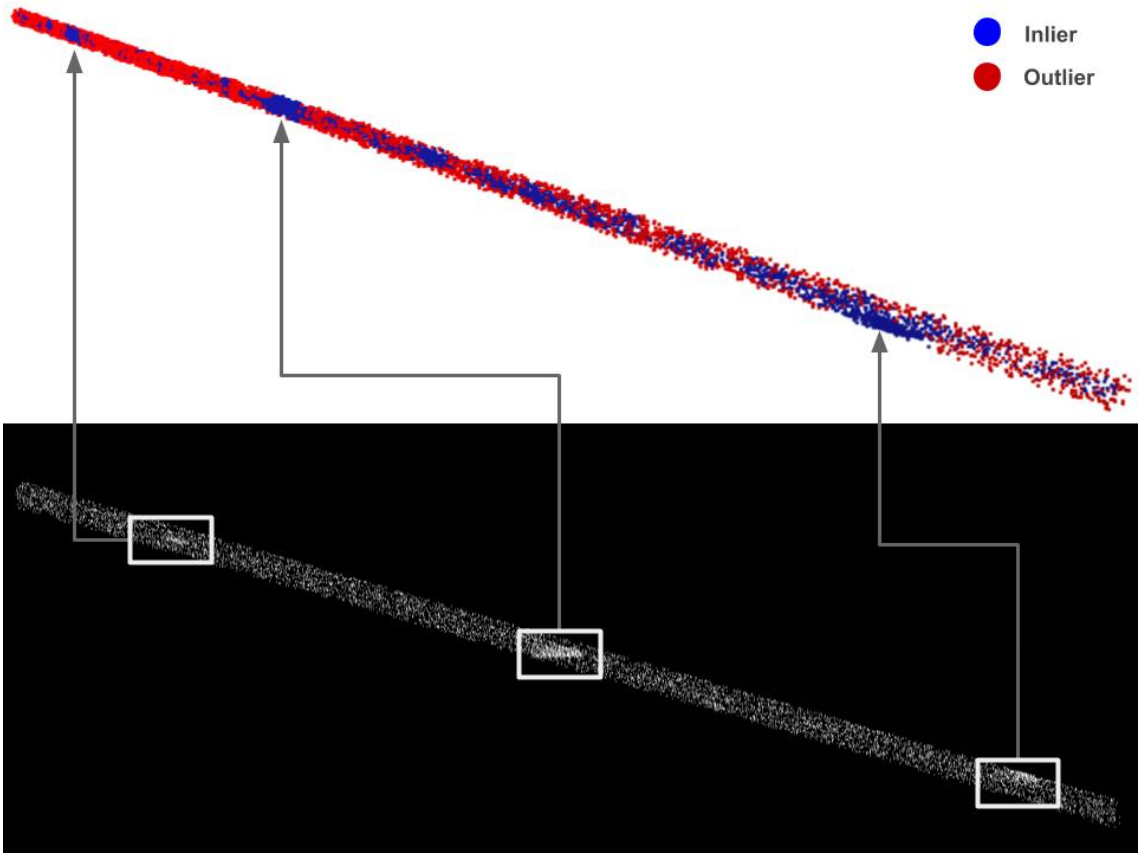
(a) Noise Timeslice Before and After Outlier Removal

(b) Event Hits Timeslice Before (Top) and After (Bottom) Outlier Removal

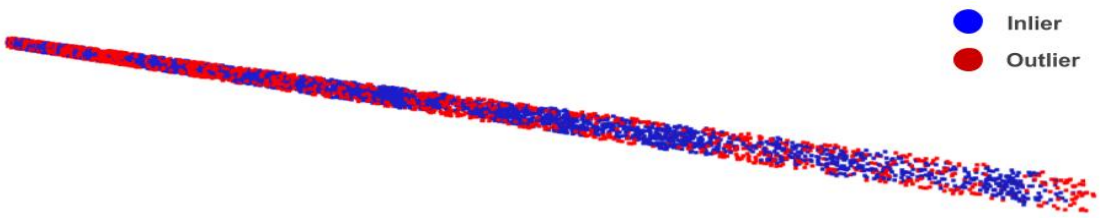
Figure 5.8: Noise and Event Hits Timeslices Before (Top) and After (Bottom) Radius-Based Outlier Detection

Figure 5.8 shows an example of a noise and event hits timeslices both before and after removal of outliers. As expected, the RBOF found a higher incidence of outliers in the

event timeslice. The resulting point cloud is more simplified with the event hit clusters intact, as highlighted in Figure 5.8b. Similarly, Figures 5.9a and 5.9b shows the result of the algorithm on timeslices with event hits. As expected, event timeslices show higher number of outliers due to the presence of event clusters.



(a) Timeslice with Highest Event Hits



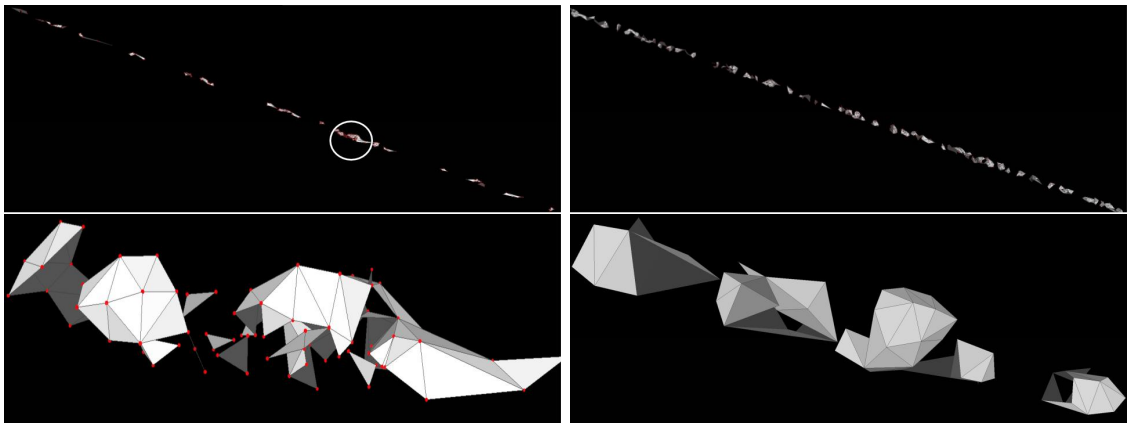
(b) Another Event Hits Timeslice

Figure 5.9: Result of Radius-Based Outlier Detection On Different Event Timeslices

5.3 3D Mesh Generation

With the feature engineering completed, `.xyz` files were converted to 3D meshes. Meshes are a collection of geometric properties - vertices, faces and edges that describe a 3D object (52). There are several algorithms that can convert 3D coordinates to meshes but two of the most popular algorithms were considered and evaluated (52).

Ball Pivoting Algorithm (BPA) "rolls" a virtual ball across the point cloud "surface". It forms a triangle from the three nearest points and then rolls along that triangle to another set of three points where it forms a new triangle (53). In this manner, the algorithm continues until all points are converted to a mesh. BPA requires the radius of the ball to be specified such that it is slightly larger than the average space between points (53). For this thesis, using an average radius value would cause problems in the event hit clusters as the distance between points in those regions may be less than the size of the ball. This would lead the ball to roll over those points and ignore them. Given the irregular, unknown symmetry with sets of close points in event clusters, it was likely that BPA would miss the details that could indicate presence of event hits.



(a) Event Timeslice: Highlighted Area Corresponds to Event Hits Cluster

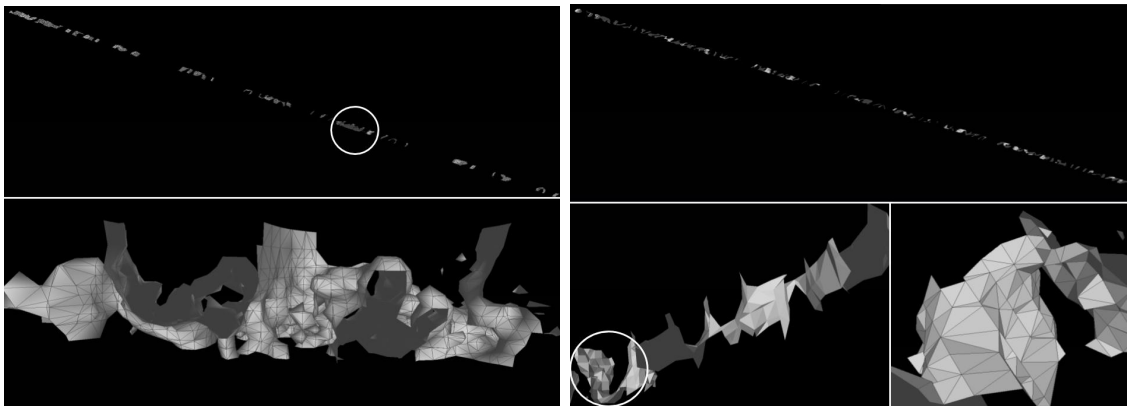
(b) Noise Timeslice

Figure 5.10: Noise and Event Timeslices After Applying Ball Pivoting Algorithm

Figure 5.10 shows the transformed point clouds after applying BPA. Figure 5.10a shows an events timeslice where event hits (in red) are overlaid with the corresponding 3D mesh for comparison. The region in focus shows an event cluster. The algorithm rolled over and missed several points, leaving gaps in the mesh. On the other hand, Figure 5.10b shows the effect of BPA on a noise timeslice. Due to the generally evenly spaced nature of noise, the algorithm was able to capture most points. However, the two meshes don't

show significant differences between each other since the event clusters are not particularly enhanced. It indicated that BPA would not be sufficient for the PointNet to learn.

Poisson Surface Reconstruction is an implicit meshing algorithm by Kazhdan et al. (2006) that envelops points in a smooth "cloth". It tries to fit a surface from the original point set by creating an entirely new point set representing an iso-surface linked to the normals (54). As it tries to create a watertight surface, it seemed a more promising solution for the detailed event regions. Poisson Surface Reconstruction required several parameters for the octree that is used for the reconstruction (54) - depth, scale and fit. The octree *depth* specifies the level of detail of re-construction and is the most significant parameter. The *scale* describes the ratio between the diameter of the cube used for reconstruction and the diameter of the samples' bounding cube. *fit* lets the re-constructor use linear or non-linear interpolation to estimate the positions of iso-vertices (54).



(a) Event Timeslice: Highlighted Area Corresponds to Event Hits Cluster

(b) Noise Timeslice: Highlighted Area Corresponds to Noisy Artefact

Figure 5.11: Noise and Event Timeslices After Applying Surface Poisson Reconstruction

Default parameters of depth `10`, octree scale `1.1` and `non-linear` interpolation were used (43). Figure 5.11 shows the same timeslices as before, rendered with Poisson Surface Reconstruction. Poisson Surface Reconstruction shows a more detailed mesh for the event hits region in the event timeslice in Figure 5.11a. Figure 5.11b shows the algorithm rendered on the noise timeslice. The surface is more or less flat and even. However, the Reconstruction algorithm produced some noisy artefacts, as highlighted in the image.

Based on the difference in mesh detail between BPA and Poisson Surface Reconstruction, Poisson Surface Reconstruction did a better job of capturing event hits. The noisy artefacts generated in noise timeslices could pose a problem, however PointNet should be able to better distinguish between the two classes based on other features.

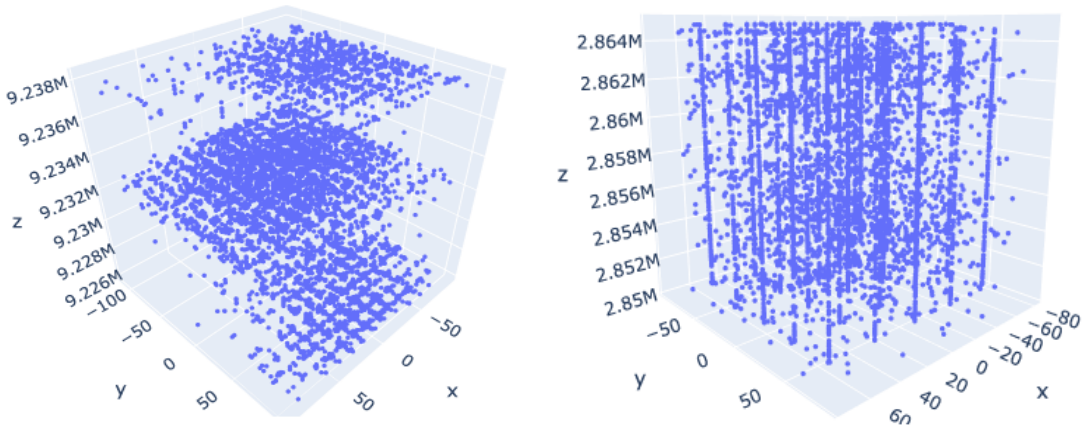
5.4 PointNet

PointNet was implemented in PyTorch [1] and a few modifications were made to work with the KM3NeT Data. Input batch (training examples) for PointNet can be 1D or 2D convolutions (1). A convolution involves the application of a filter to the input that results in an activation function (36). For the thesis, 1D convolutions were used as it can help reduce dimensions and computational costs (55). In this case, the 1D convolution was the Multi-Layer Perceptron (MLP) with shared weights and kernel of size 1.

Pooling layers allow for down-sampling feature maps by means of a summary (36). Two common pooling methods include max pooling which summarises the most activated presence of a feature; and average pooling which summarises the average presence of a feature (36). Instead of using a global max pool function suggested in the paper by Qi et al. (2017), an average pool function was applied to the transformed features (1). This was done because max pool was found to cause some overfitting to the KM3NeT data.

The paper by Qi et al. (2017), used Cross Entropy Loss, a typical loss function for classification (1). However, thesis experiments showed that the network learned better with the negative log-likelihood (NLL) loss. As the NLL loss requires log probabilities of each class, the Log Sigmoid activation was also applied to the output layer (56). The remainder of the architecture was left unchanged.

5.4.1 PointNet Transformations



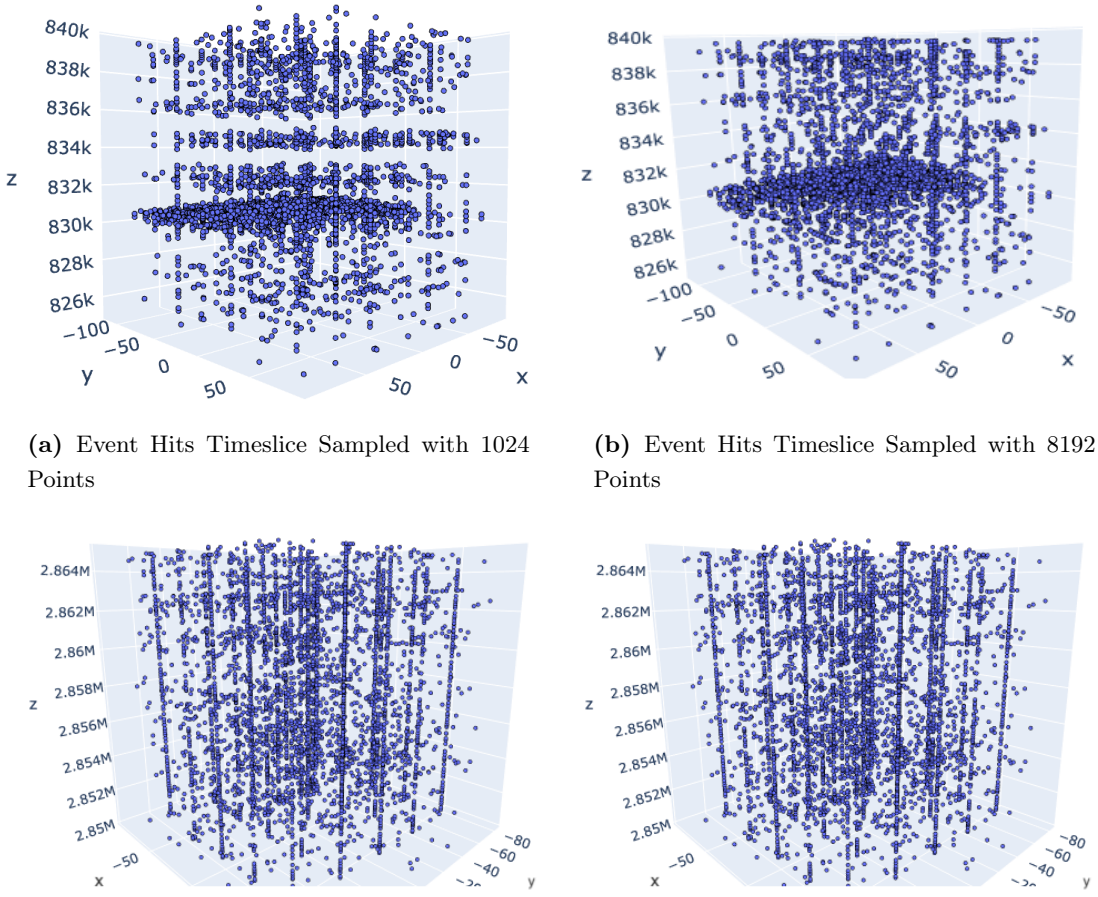
(a) Events Timeslice With Vertices from Mesh **(b)** Noise Timeslice With Vertices from Mesh

Figure 5.12: Noise and Event Timeslices With Non-Uniform Distribution of Mesh Vertices

[1] https://pytorch.org/

Qi et al. (2017) stated three requirements for training point clouds with PointNet (Section 3). Point clouds should be unordered and PointNet has to be invariant to permutations of the input set. Next, the network must be invariant to rigid transformations. Finally, the network should also capture interactions among points (1). The thesis implemented the recommended transformations to the input point clouds during training and testing to ensure conformity of data to the above requirements (1).

Points are not uniformly distributed across the 3D object's surface, hindering PointNet's ability to classify them. For example, Figure 5.12 shows non-uniformity in the distribution of vertices from the meshes for both noise and event class.



(a) Event Hits Timeslice Sampled with 1024 Points

(b) Event Hits Timeslice Sampled with 8192 Points

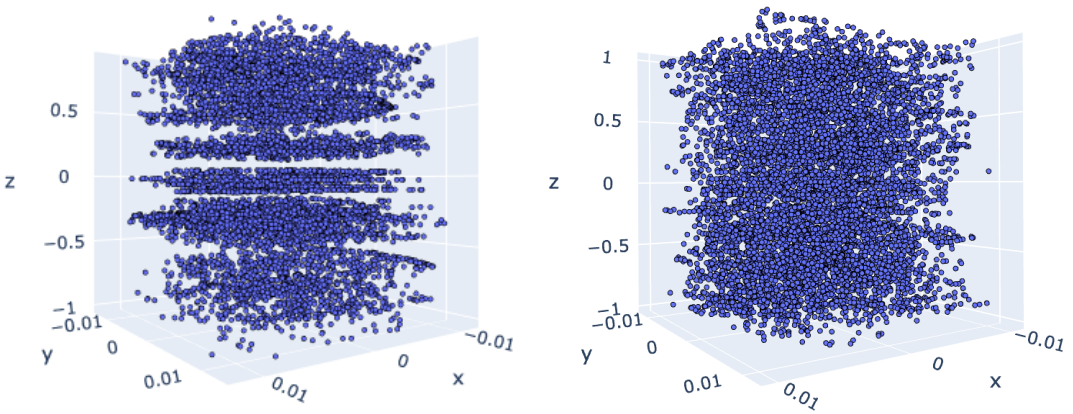
(c) Noise Timeslice Sampled with 1024 Points

(d) Noise Timeslice Sampled with 8192 Points

Figure 5.13: An Example of Noise and Event Timeslices With Different Levels of Sampling

PointNet also requires a fixed number of points to be sampled per point cloud (1). While Qi et al. (2017) uniformly sampled 1024 points on mesh faces, it was found that this

number was not sufficient for the KM3NeT Data. Figure 5.13 shows both classes of point clouds randomly sampled with 1024 points and then with 8192 points (points have to be in multiples of 1024). While noise timeslices were relatively unaffected, event timeslices showed better structure with greater points. The thesis sampled one point per chosen face. Additionally, as faces had different areas, a probability was assigned to choose a particular face proportional to its area. That is, faces with higher area had higher probability of being chosen as it represented more of the surface (1). In this manner, a total of 8192 point were sampled per point cloud.



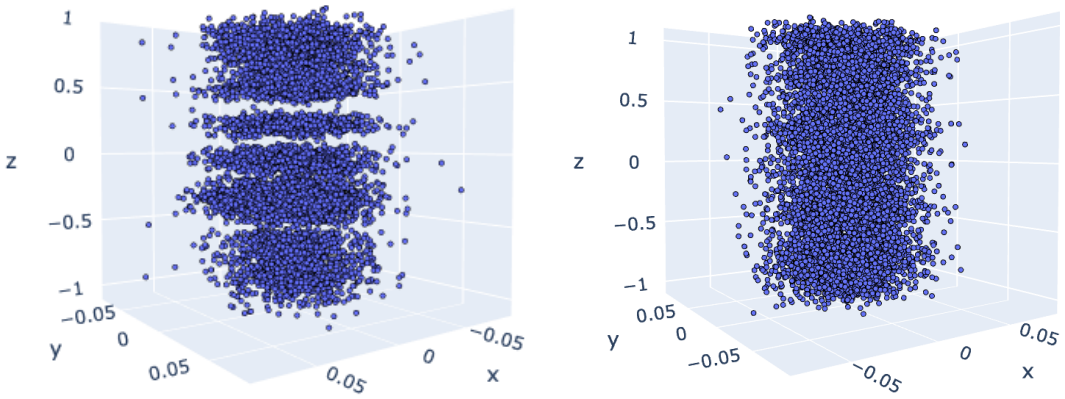
(a) Event Hits Timeslice **(b)** Noise Timeslice

Figure 5.14: An Example of Noise and Events Timeslice After Normalisation

Point clouds in the KM3NeT dataset have different sizes and are often placed in different parts of the coordinate system. Qi et al. (2017) normalised point clouds along a unit sphere (1). The thesis also applied similar translations. Point cloud were translated to the origin by subtracting the mean from all its points, and then normalising them into a unit sphere (1). Figure 5.14 shows the effect of this normalisation for both classes.

Qi et al. (2017) also augmented point clouds during training by adding jitter and randomly rotating the objects (1). Therefore, the event and noise point clouds were randomly rotated around the Z-axis. Jitter via Gaussian noise was also added with `0` mean and `0.02` standard deviation (1). Figure 5.15 shows the effect of these augmentations for both classes.

The application of the recommended augmentations resulted two fairly distinct point clouds and was considered effective. The noise timeslice formed a dense block while the events timeslice formed a collection of points with gaps in between.



(a) Event Hits Timeslice **(b)** Noise Timeslice

Figure 5.15: An Example of Noise and Event Timeslices After Jitter and Random Rotation

Figure 5.16 shows an overview of the pipeline that was used for each of the three datasets - `(x y time)`, `(x z time)` and `(y z time)`. First, 3D coordinates were generated and separated into two classes. Feature engineering using radius-based outlier filter eliminated several points, enhancing event clusters. Surface Poisson Reconstruction was applied to convert the coordinates to 3D meshes. PointNet transformation functions were also added to enhance the point clouds on the fly during training and testing.

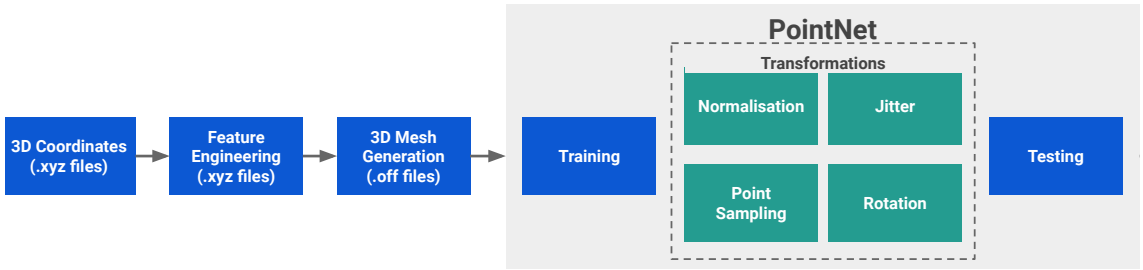


Figure 5.16: KM3NeT Pipeline Components

6

Evaluation Methodology

The selected timeslices had between 500 to 1692 event hits to capture a variety of event sizes for training. Figure 6.1 shows the directory structure for the dataset. Timeslices were organised by classes and separated into individual `training` and `testing` folders as required by PointNet.

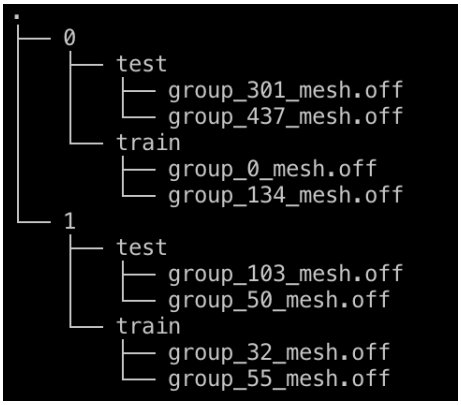
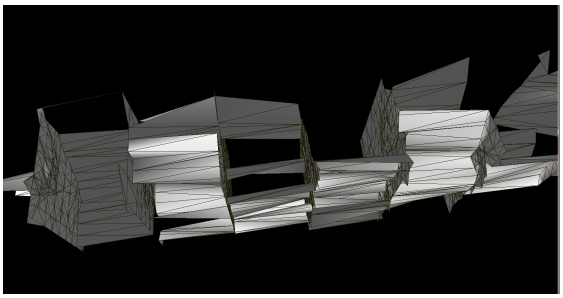


Figure 6.1: Required Directory Structure for PointNet

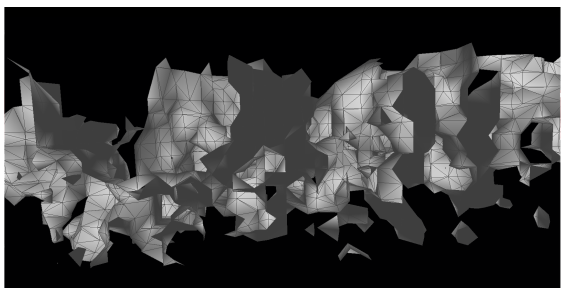
An 80-20 train-test split was used to randomly select 80% of the files under each class for training and 20% for testing. No validation set was required because there was no need to choose an appropriate model from rivalling approaches (57, 58). This chapter discusses the preparation of the training and testing data, along with two chosen edge cases. The chapter also describes the ensemble methodology used for obtaining final predictions and remarks on the use of L1 trigger on the dataset for comparison against PointNet.

6.1 Training and Testing Data

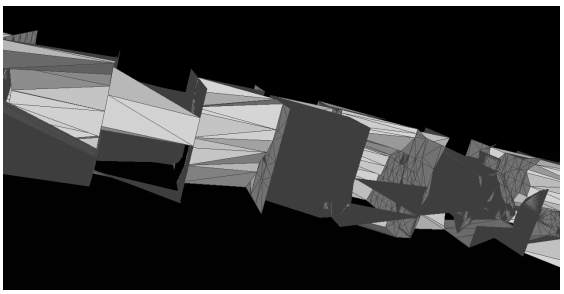
160 timeslices were selected for each class as training data. Both classes had meshes with approximately 4800 vertices on average, and 8500 triangles. A total of 40 timeslices were selected as testing data for each class. Both classes had meshes with approximately 4800 vertices on average and 7500 triangles. The testing data represented much of what the detector would gather in real world, and send to the pipeline for processing. However, two additional edge cases were chosen to evaluate the extent of the model's learning capacity. Most event timeslices have around 700 event hits on average. Therefore, a timeslice with 100 event hits and 6796 noise hits was selected. As an extreme case, a group with 39 event hits and 6645 noise hits was selected.



(a) 3D Mesh of Timeslice with 100 Event Hits



(b) Typical Example of Timeslice with Event Hits



(c) Typical Example of Timeslice with Noise

Figure 6.2: Edge Case: Comparison of Mesh Detail for Timeslices with Low Event Hits

Figure 6.2a shows the detail captured by the mesh for a processed point cloud with 100 event hits. Figure 6.2b on the other hand is an example of a typical event cluster rendered with details and rounded surfaces. Figure 6.2c is an example of a typical noise region, rendered as smoother surfaces by Surface Poisson Reconstruction. Figure 6.2a resembles the noise point cloud in Figure 6.2c with smooth surfaces. This indicated that the event hits details were lost in conversion and PointNet may not be able to classify it as an event timeslice.

Similarly, Figure 6.3 shows another example of a point cloud with 39 event hits. The mesh representation contains very few distinguishing features and flat surfaces, unlike a typical event cluster mesh (Figure 6.2b). Since the event cluster resembled the mesh representing noise hits, it was likely that PointNet would not be able to classify it as an events class.

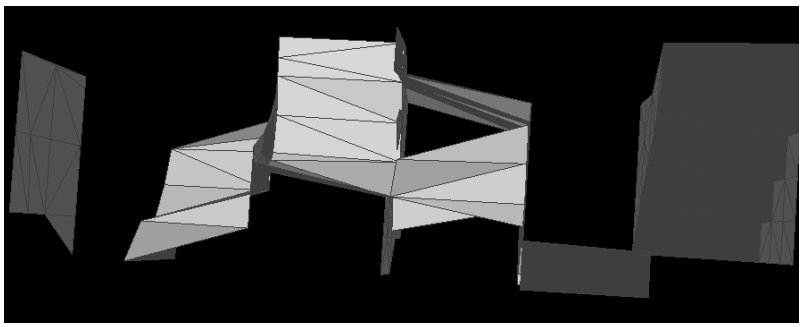


Figure 6.3: 3D Mesh of Timeslice with 39 Event Hits

6.2 Model Specifications

The model was first run for `200` epochs to find the optimal training epoch value. Figure 6.4 shows that the loss settled around 100 to 120 epochs. Therefore, the rest of the experiments were conducted for `120` epochs. The loss was found to be between `0.008` and `0.003` during training and considered sufficient for the thesis.

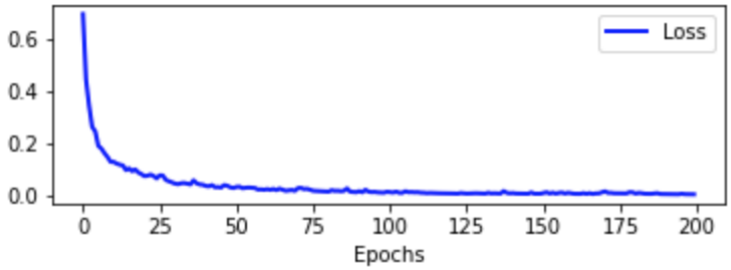


Figure 6.4: Loss Plotted for 200 Training Epochs

Model parameters were experimented to obtain optimal performance, but it was noted that the default parameters in the paper by Qi et al. (2017) were best suited for the dataset (1) (Appendix 10). Table 6.1 shows the parameters used for training. For each batch, the model was evaluated on the testing data to obtain a signal about the model's ability to

generalise. The dataset was trained with PyTorch using Google Colab NVIDIA Tesla K80 GPU and on ViltStift AMD MI50 GPUs for approximately 3.5 hours.

regularisation loss with weight	0.001
Initial Learning Rate	0.001
Dropout probability	0.3
Batch Size Training	32
Batch Size Testing	64
Epochs	120

Table 6.1: Final Model Parameters Used for Training on KM3NeT Data

6.3 Ensemble Methods for Results

As `x, y, z,` and `time` variables were split into three permutations of (`x, y, time`), (`x, z, time`) and (`y, z, time`), each dataset was trained separately. The results from each of these datasets were then combined using majority voting ensemble techniques (36). Voting ensembles were selected as a suitable methodology for two main reasons. First, while there were three datasets, they represented the same information, except in slightly different manner. Majority voting ensembles place equal value on the models being combined to make predictions which was relevant in this case (59). Second, a voting ensemble is considered appropriate when no single model performs comparatively worse or better than the others (59). Results showed that two of the three models were quite similar in performance.

Two voting ensemble techniques were used to generate the final predictions - hard voting and soft voting. A hard voting ensemble sums the votes for class labels from all models and presents the class with the most vote as the final prediction (60). A soft voting ensemble sums the predicted probabilities for class labels and presents the label with the highest sum probability (60). A soft voting ensemble additionally requires a probability threshold to finalise the label as positive. A high threshold of 90% was used due to the physics requirement of minimising false positives and correctly classifying event timeslices as much as possible.

The existing L1 trigger (Section 1.2.1) was also applied on the dataset. The results from the L1 trigger were contrasted against PointNet to identify if any improvements could be noted. The L1 trigger is known to save noise timeslices as relevant, so PointNet would have to perform better in that instance. The training and testing data was therefore prepared

to ensure that a good range of event sizes were covered. Further, two edge cases for event timeslices were also prepared to establish the extent of the network's learning capacity. Majority voting ensemble was seen as a robust, simple ensemble technique and used to obtain final predictions.

7

Classification Results

Accuracy score is typically used to identify optimal models, however as a metric it is often not interpretable and discriminative (61). Moreover, an accuracy score is almost always biased towards the majority class. The goal was to ensure that timeslices with neutrinos did not get mislabelled as noise. Therefore, *recall* was the most significant metric for the classification task. Based on the highly imbalanced nature of the dataset and the physics goal, a few other metrics were also required (62). This chapter establishes the relevant metrics selected for classification and discusses the results obtained through plots and summaries.

Precision measures the percentage of results that are relevant. It is the ability of the classifier to not label a negative sample as positive (62). Precision is given by the number of true positives (T_p) over the number of true positives (T_p) plus the number of false positives (F_p):

$$P = \frac{T_p}{T_p + F_p}$$

Recall refers to the percentage of total relevant results correctly classified, and is the ability of the classifier to find all positive samples (62). It is given by the number of true positives (T_p) over the number of true positives (T_p) plus the number of false negatives (F_n):

$$R = \frac{T_p}{T_p + F_n}$$

F1 Score is a weighted harmonic mean of the precision and recall (62). It is given by:

$$F1 = 2 \times \frac{P \times R}{P + R}$$

Receiver Operating Characteristic (ROC) Curves depict the trade-off between true positive rate and false positive rate for different probability thresholds (62).

Precision-Recall (PR) Curve shows the trade-off between precision and recall for different thresholds (62). A high area under the curve represents both high recall and high precision.

Confusion Matrix visualises the number of true positives, false positives, true negatives and false negatives (63) .

Each of the three datasets - (`x, y, time`), (`x, z, time`), and (`y, z, time`) were trained using the same PointNet model parameters (Table 6.1), and their results combined using ensemble techniques. Table 7.1 shows the final classification accuracy scores for each of the three datasets and the results after applying two majority voting ensemble methods.

`x, y, time`	**95%** (loss = 0.003)
`x, z, time`	**90%** (loss = 0.006)
`y, z, time`	**99%** (loss = 0.005)
Ensemble 1: Hard Voting	**97%**
Ensemble 2: Soft Voting	**90%**

Table 7.1: Accuracy Scores

A classification report was generated for each dataset and the two ensembles to obtain detailed performance report on precision, recall and F1 Scores. ROC Curves and PR Curves were both plotted to compare against each other. This is because ROC curves are more appropriate for balanced classes and PR curves are better suited for imbalanced datasets. Keeping in mind the physics goal, analysis was more focused on the model's performance in (mis)classifying event timeslices (`class_1`).

7.1 Dataset 1: `x, y, time`

The `x, y, time` model reported a 95% accuracy. Classification report in Table 7.2 shows that the model did not mislabel any noise (`class_0`) as event `class_1` timeslices. Further, **90%** of the total relevant results were correctly classified as `class_1`. Support indicates that 40 test cases were used for each class.

`x, y, time`				
	precision	recall	F1-score	support
`class_0`	0.91	1.00	0.95	40
`class_1`	1.00	0.90	0.95	40

Table 7.2: `x, y, time`: Classification Report for `class_0` and `class_1`

Figure 7.1 shows the relevant plots of metrics for the dataset. The ROC curves are towards the top left of the graph and indicate a desirable, high true positive rate against the baseline performance plotted on the diagonal. Complementing the ROC, the Precision-Recall (PR) curves show a high area under the curve, indicating high precision and recall. Precision was mostly stable for increasing recall, but it fell dramatically for `class_1` after 85% recall. The confusion matrix shows that 4 of the 40 timeslices with event hits were incorrectly labelled as noise, but none of the noise timeslices were incorrectly labelled.

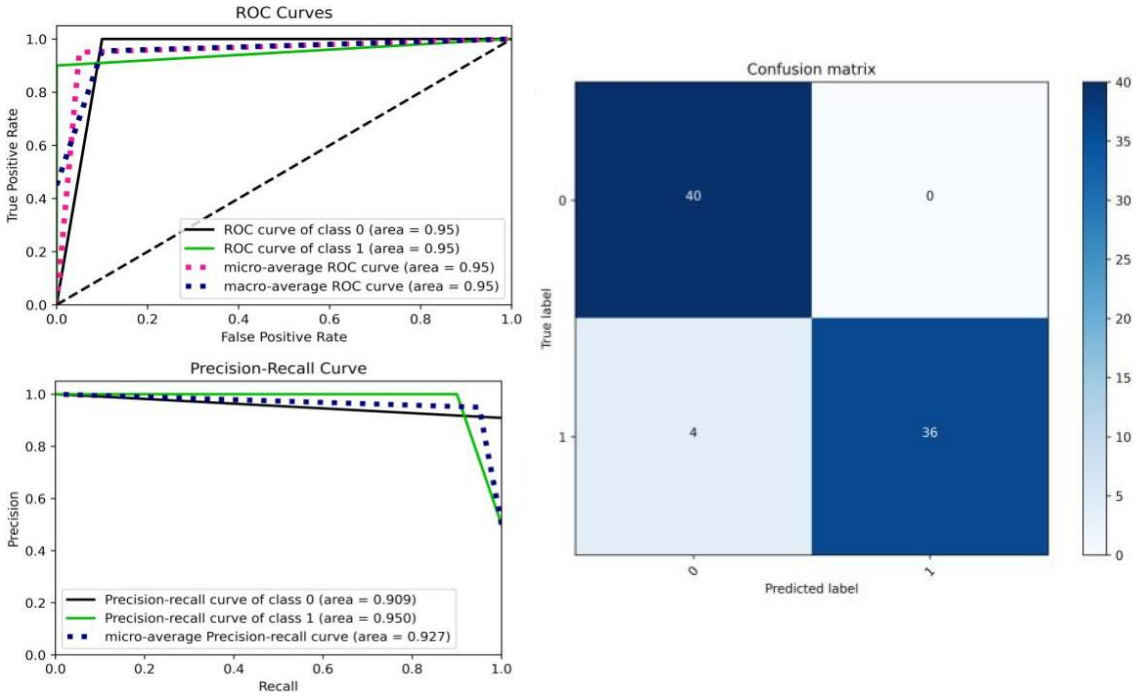


Figure 7.1: `x, y, time`: Classification Plots

7.2 Dataset 2: `x, z, time`

`x, z, time`				
	precision	recall	F1-score	support
`class_0`	0.83	1.00	0.91	40
`class_1`	1.00	0.80	0.89	40

Table 7.3: `x, z, time`: Classification Report for `class_0` and `class_1`

The `x, z, time` model reported a lower accuracy of 91%. This is because it wrongly

classified a higher percent of `class_1` samples (Table 7.3). The model was however much better at detecting noise and obtained a **perfect recall**.

Figure 7.2 shows the relevant plot of metrics for the dataset and indicates poorer performance compared to the previous model. The ROC and PR curves both show lower areas under the curve. The confusion matrix shows that out of the 40 test cases, 8 of the event timeslices class were incorrectly labelled as noise. Again, none of the noise timeslices were wrongly classified.

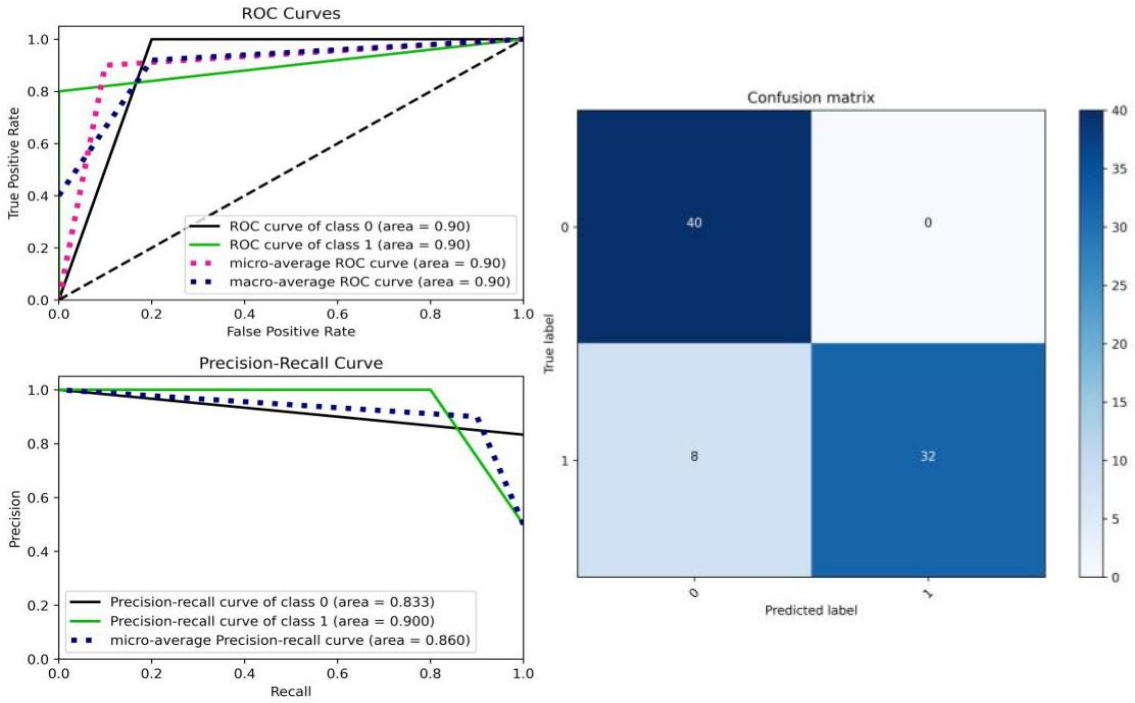


Figure 7.2: `x, z, time`: Classification Plots

7.3 Dataset 3: `y, z, time`

The `y, z, time` model reported 99% accuracy score. The classification report in Table 7.4 showed that the model did not incorrectly classify events as noise 98% of the time. More importantly, it was able to correctly find and classify **all** event timeslices.

Figure 7.3 shows the relevant plot of metrics for the dataset. The ROC and PR curves indicate very desirable results. The confusion matrix shows that out of the 40 test cases, all of the event timeslices were correctly labelled. Although one instance of noise timeslice was incorrectly classified as a timeslice with events.

y, z, time				
	precision	recall	F1-score	support
class_0	1.00	0.97	0.99	40
class_1	0.98	1.00	0.99	40

Table 7.4: `y, z, time`: Classification Report for `class_0` and `class_1`

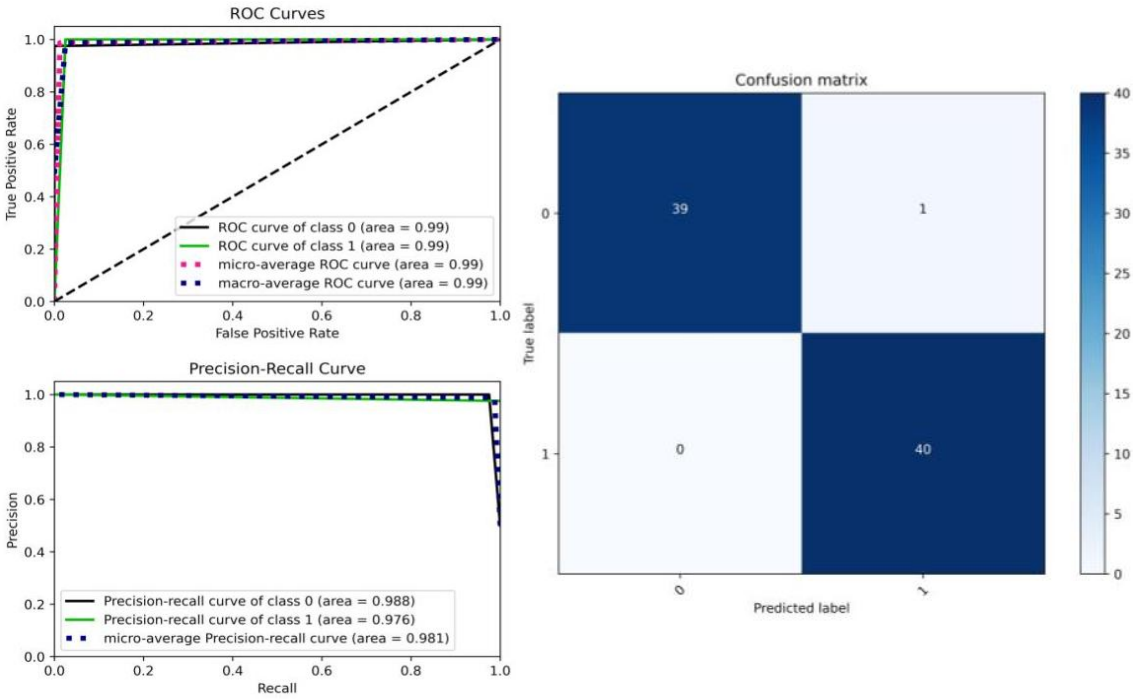


Figure 7.3: `y, z, time`: Classification Plots

Between the three datasets, the `y, z, time` model performed better than the other two, in terms of recall. It also had the highest F1 score for the positive class. The `x, z, time` model performed poorly by wrongly classifying the most number of event timeslices.

7.4 Majority Voting Ensemble

Next, results from the three models were combined using hard and soft voting. The classification reports in Tables 7.5 and 7.6 show high precision, recall and F1 scores for hard voting. Soft voting on the other hand predicted much lower scores. It scored especially low on recall for `class_1`. The supporting plots in Figure 7.4 for hard majority voting corroborate to the precision and recall scores. Additionally, the final models were tested on the two edge cases - timeslices with very low event hits. All three models predicted

both event timeslices to be noise timeslices. These results confirmed the visual inspection of the input meshes to PointNet (Section 6.1). Figure 7.5 shows the poorer performance of soft majority voting especially since 8 of the event timeslices were labelled as noise.

Hard Voting: `(x y time), (x z time) (y z time)`				
	precision	recall	f1-score	support
`class_0`	0.95	1.00	0.98	40
`class_1`	1.00	0.95	0.97	40

Table 7.5: Hard Voting: Classification Report for `class_0` and `class_1`

Soft Voting: `(x y time), (x z time) (y z time)`				
	precision	recall	f1-score	support
`class_0`	0.93	1.00	0.91	40
`class_1`	1.00	0.80	0.89	40

Table 7.6: Soft Voting: Classification Report for `class_0` and `class_1`

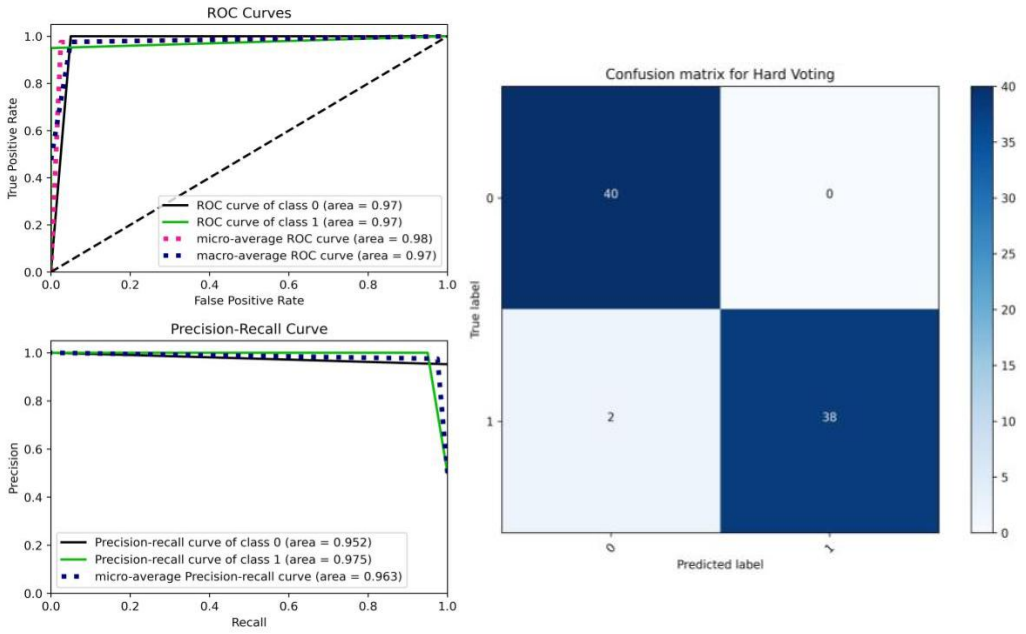


Figure 7.4: Hard Voting Ensemble Results

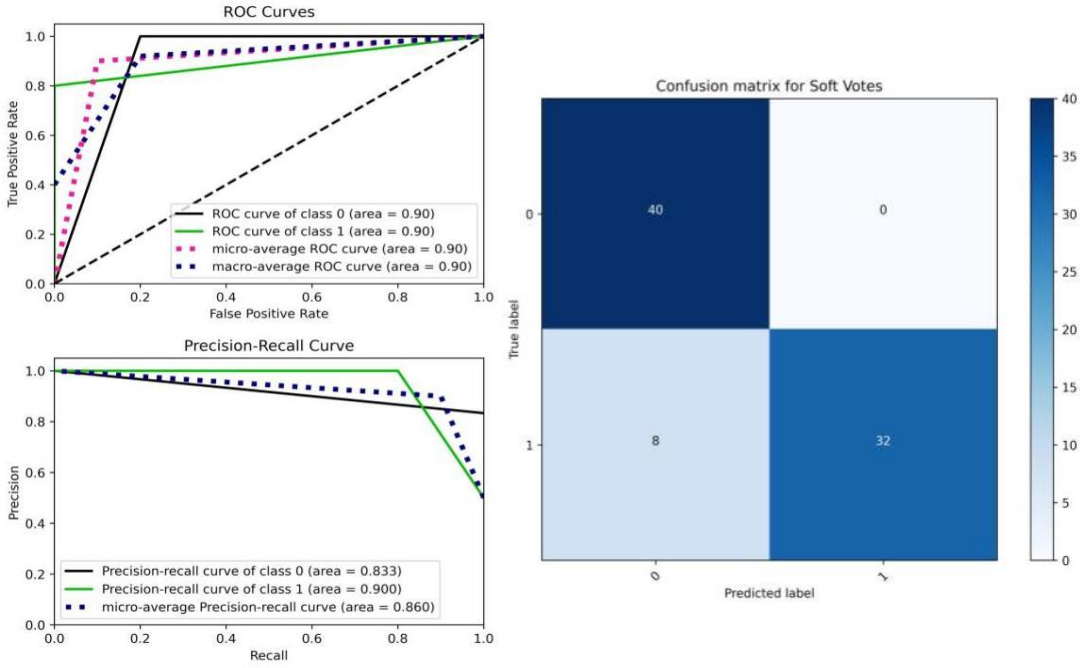


Figure 7.5: Soft Voting Ensemble Results

7.5 Comparison Against L1 Trigger

The L1 trigger was applied to same test cases used for evaluating PointNet. Figure 7.6 indicates that it can correctly classify all 40 event timeslices. However, it wrongly classified 5 noise timeslices. Like the PointNet model, it was unable to classify either of the two edge cases. Therefore, PointNet demonstrated better performance than L1 Trigger, especially in terms of minimising false positives.

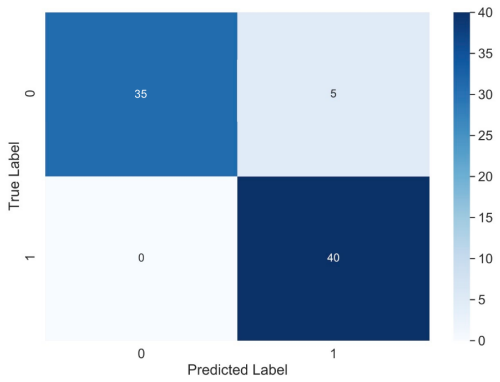


Figure 7.6: Confusion Matrix From L1 Trigger to Testing Data

7.6 Other Performance Metrics

Execution time for the pre-processing pipeline and the deep learning model was measured for the ensemble. Additionally, throughput and energy efficiency were calculated for the PointNet model. The ensemble comprised of the three datasets - (`x, y, time`), (`x, z, time`) and (`y, z, time`). Each dataset contained 400 timeslices, so a total of 1200 timeslices were processed. As each timeslice contained around 8000 rows, the complete ensemble had approximately 9,600,000 rows of data. The pre-processing was run on a Macbook Pro with 8GB of memory and 2 cores. All deep learning performance metrics were measured on Google Colab's Tesla T4 GPUs.

Python's `cProfile` [1] was used to obtain execution times for the complete pre-processing pipeline, starting from generation of 3D coordinates, to finally separating data into training and testing datasets. The total execution time was recorded to be *3771.85 seconds* for the entire ensemble of three datasets. Figure 7.7 shows the execution time for individual components of the pipeline. As expected, the main bottleneck arises from the conversion of 3D coordinates to 3D meshes.

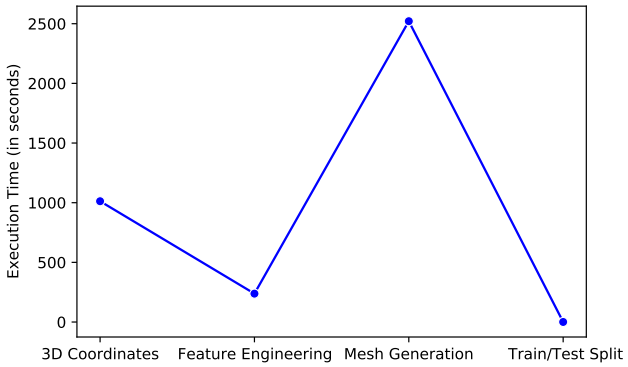


Figure 7.7: Combined Execution Time for 3 Data sets for Each Stage in Pre-Processing Pipeline for the three datasets

Next, execution time for PointNet was obtained, keeping in mind that execution time for deep learning typically involves only the feedforward functions (64, 65). The three datasets were trained and evaluated in parallel, after which the results were combined. The mean execution time was measured to be *94950.47 milliseconds (ms)* or *1.5 minutes* and the total time for 120 epochs was approximately *3.2 hours*.

The throughput of a neural network can be stated as the maximum number of input the network can process in unit time and is an important metric for scalability (66). The

[1] https://docs.python.org/3/library/profile.html

following formula was used to determine the throughput (66):

$$\frac{(number\ of\ batches \times max\ batch\ size)}{(total\ time\ in\ seconds)}$$

The maximum batch size for the GPU was found by performing a binary search and was *64*. It was seen that the network could process *0.34* input instances per second.

Finally, energy efficiency for the model was also calculated. Deep learning has the ability to solve complex tasks on specialised hardware, but are often energy intensive (67). Energy metrics are not frequently reported in deep learning, but is significant in the present global climate (67). Table 7.7) shows the energy metrics for the PointNet model and was found using Python's `carbontracker`[1]. The reported 62.4 g of CO2 emissions are relatively acceptable in comparison to an average of 4.5 kg CO2 emitted by standard deep learning models (68).

Actual consumption for 1 epoch(s)	
Time	0:01:38
Energy	0.001770 kWh
CO2eq	0.520728 g
This is equivalent to: 0.004325 km travelled by car	
Predicted consumption for 120 epoch(s)	
Time	3:16:20
Energy	0.212393 kWh
CO2eq	62.487393 g
This is equivalent to: 0.518998 km travelled by car	

Table 7.7: Energy Metrics and Carbon Footprint for PointNet

The execution time for a single timeslice with 8500 hits including 700 event hits was also measured. The complete pipeline took 3.5 minutes from pre-processing and then obtaining a classification from the trained model.

7.7 Analysis

Hard voting ensemble was chosen as the final result of PointNet classification project based on recall scores. Despite being considered naive, hard voting was suitable in this thesis since all three datasets more or less equally contributed to the learning. Since the three

[1]https://pypi.org/project/carbontracker/

datasets are spatial combinations, none of them would have more valuable information about the neutrino events than the other. Soft voting resulted in lower performance as it was affected by the `x, z, time` recall scores. But since all three dataset models are equally important, a majority voting was more appropriate in finalising the results.

Overall, the model was able to perform well in classifying timeslices into noise versus events. **97%** of of the predictions were accurate overall. The final results showed that the model was able to correctly classify all noise timeslices and had a **95%** recall for the positive class. All results showed that the models were better at classifying noise timeslices than event timeslices. This could be because of the feature engineering itself. As the network is tuned to handle class imbalances and overfitting, feature engineering could be insufficient in certain cases for timeslices with events. Out of 40, it only mis-classified 2 event timeslices as noise. While the recall for the positive class is not perfect, it met and exceeded the stakeholder requirement of 90%. It also improved over the L1 trigger's performance in terms of identifying false positives. These scores have to be credited to the feature engineering and a high sampling of points per point cloud. Several experiments were conducted with the exact same network, but with no feature engineering. Here the model could not achieve more than 65% accuracy in identifying positive classes (Appendix 10). Additional experiments with the number of points sampled per point cloud showed that the higher the points sampled, the better. With the default 1024 points sampled and feature engineering, the model was able to perform only slightly better than before with 72% accuracy. It was clear that due to the highly irregular shape of the mesh, with very low occurrences of significant areas, increasing the number of points sampled, increased the probability that points would be taken from the event cluster parts of the mesh.

In terms of execution time, a single incoming timeslice will take 3.5 seconds to be classified. This is not ideal in real-time processing, where there may be incoming timeslices at every second. The proposed GPU pipeline in comparison is significantly faster (26).

This chapter examined results from the three datasets, focusing on the recall for positive class (`class_1`). Results from the three datasets were combined using two majority voting techniques - hard and soft voting. Hard voting was chosen as the preferred, final result based on the best recall and precision scores. The results also showed improvements over the L1 trigger. There are no other similar approaches that can allow the project to make performance comparisons against, but the results can serve as a benchmark for future improvements. Additional execution, throughput and energy metrics also cannot be directly compared with other models. Instead, they can be used by KM3NeT to evaluate the feasibility of the model given their resources.

8

Additional Research

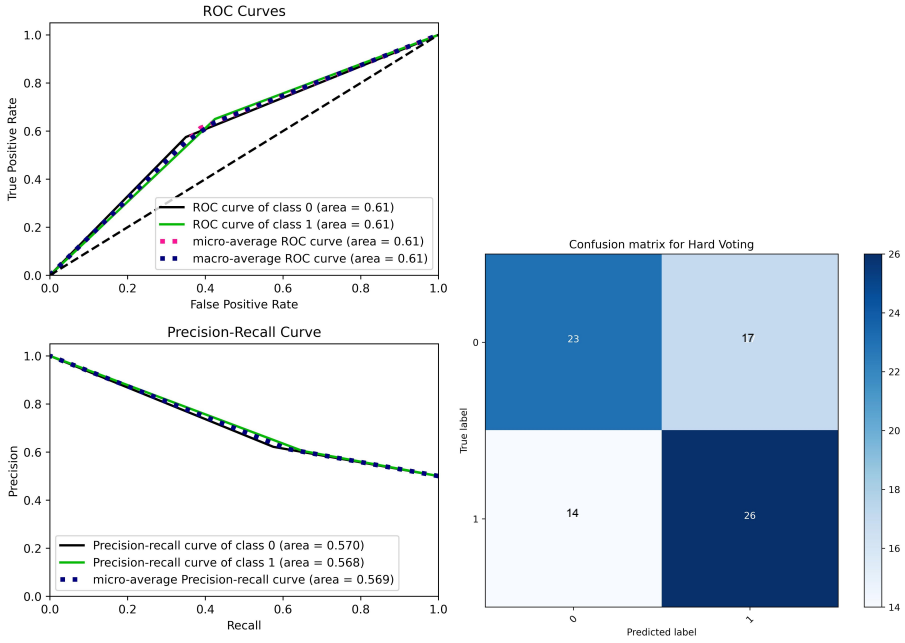
The 3D mesh-based classification pipeline was considered successful in learning to differentiate between event and noise timeslices. However, feature engineering was required to be able to learn sufficiently (Appendix 10). Qi et al. (2017) had highlighted that PointNet could work with just 3D coordinates (1). Therefore, additional research was conducted to explore an alternate 3D point-based pipeline. If successful, it would simplify the computational costs and research time. The 3D point-based pipeline was next expanded to a 4D PointNet. Instead of generating three datasets out of `x, y, z` and `time`, they could be directly used as 4D points to train PointNet. The final section of this chapter provides initial exploration of the energy inference requirement (Section 1.3) and lays the path for future research.

8.1 Alternate Pipeline: 3D Points-based PointNet

The thesis pipeline generated three sets of data and their corresponding `.xyz` files. It performed feature engineering and generated 3D meshes that were used for training and evaluation. As an alternate approach to the pipeline, the thesis attempted to determine if PointNet could perform successful classification with just feature engineered `.xyz` files.

As before, three combinations of the KM3NeT dataset were prepared - (`x, y, time`), (`x, z, time`) and (`y, z, time`). Feature engineering using radius-based outlier detection was once again used to simplify the point clouds. These points were then directly passed on to PointNet. Within the network, the point clouds were transformed in the same manner as before. 8192 points were equally sampled per cloud. They were normalised, rotated around the Z-axis and enhanced with random noise. Each of the three datasets were trained with the exact same parameters as before (Table 6.1, Appendix 10).

However, results showed that this model was unable to learn from the point clouds. It could not achieve more than 61% accuracy. While the predictions are not random, they are enough for physics requirements. Additionally, the precision for the positive class was found to be **0.60** and the recall was **0.65**, indicating only 65% of the positive results were correctly classified. This is further shown by the decreasing Precision-Recall (PR) curves in Figure 8.1a. The almost diagonal curves of the Receiver Operating Characteristic (ROC) curve in Figure 8.1a demonstrates a nearly random classifier, meaning that the model predicted with mostly random chance (62). The confusion matrix in Figure 8.1b shows that 14 instances of event timeslices were wrongly labelled as noise.



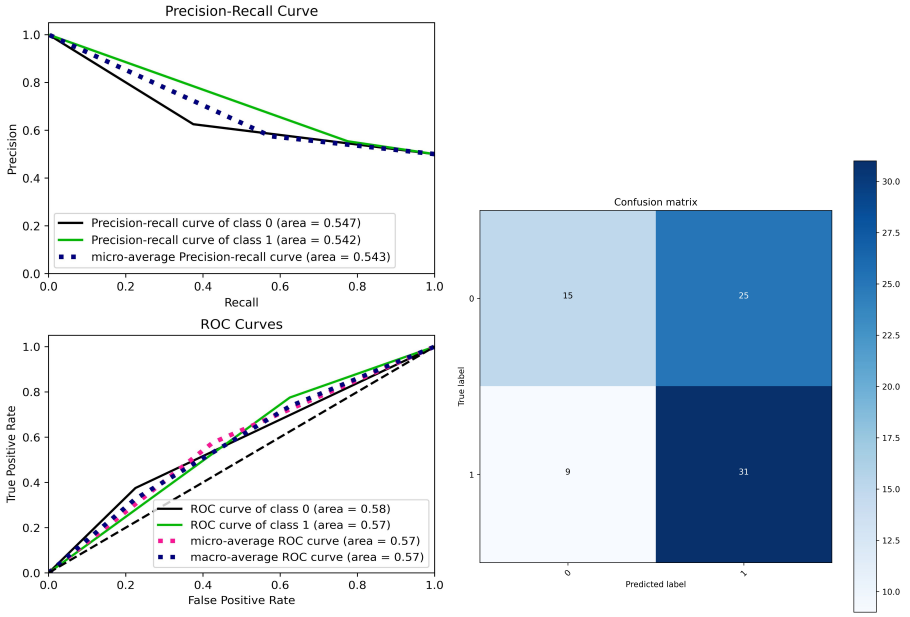
(a) ROC and PR curves **(b)** Confusion Matrix

Figure 8.1: Results of Classification using 3D Points

While several experiments using PointNet are run on 3D points (69, 70, 71), this was not applicable to the KM3NeT data. It could be because the points alone did not contain sufficient information for the network to learn from. As the network also randomly samples a fixed number of points per point cloud, it was likely that the relevant event hits were not selected by the random sampler. While it would have been computationally beneficial to successfully train a model without 3D meshes, these results provide justification for the mesh generation step in the pipeline. It was evident that the 3D meshes added the required level of detail necessary for PointNet to learn.

8.2 Alternate Pipeline: 4D PointNet

Experiments were also conducted to see if PointNet could learn from the `x, y, z, time` attributes at once, instead of generating three datasets. As 4D meshes are not technically feasible, the points were only pre-processed using radius-based outlier detection. The initial layers of the architecture were modified to accept 4 inputs (Appendix 10). Training showed that the model was able to perform only slightly better than before. It achieved an accuracy of **57%** with **78%** recall for the positive class. Figure 8.2a shows low areas under the ROC and PR curves. The confusion matrix in Figure 8.2b indicates that the classifier was better at classifying event timeslices over noise.



(a) ROC and PR curves **(b)** Confusion Matrix

Figure 8.2: Results of Classification using 4D Points

8.3 Regression Analysis for Energy Inference

The final segment of the research question was focused on inferring energy values from timeslices classified as `class_1` (1.3). Obtaining secondary properties such as the energy of an event is important as they can help identify lesser understood events such as decay of ark matter particles (25). While PointNet can be theoretically used for regression tasks, there is no known research that demonstrate this yet (1). PointNet was specifically built for object classification and segmentation tasks, so tuning the architecture for regression tasks would

involve significant architectural changes. This indicated that a different architecture more suited for regression should be used instead of modifying PointNet. Additionally, the thesis pipeline started with the 3D coordinates and time values. However, these attributes were no longer represented by the same values when the point clouds were converted to 3D meshes. Thus, after classification, it would be challenging to map the 3D meshes to their respective energy values and proceed with training for regression. Due to these reasons, building a model for regression was more appropriate for a separate research project and was not attempted further. Instead, the remainder of the thesis investigated the inference of energy values using two non-linear techniques - decision trees and random forests bootstrapping. That is, experiments were not considered an extension of the classification pipeline and are only intended to lay the ground for future work.

8.3.1 Data Preparation

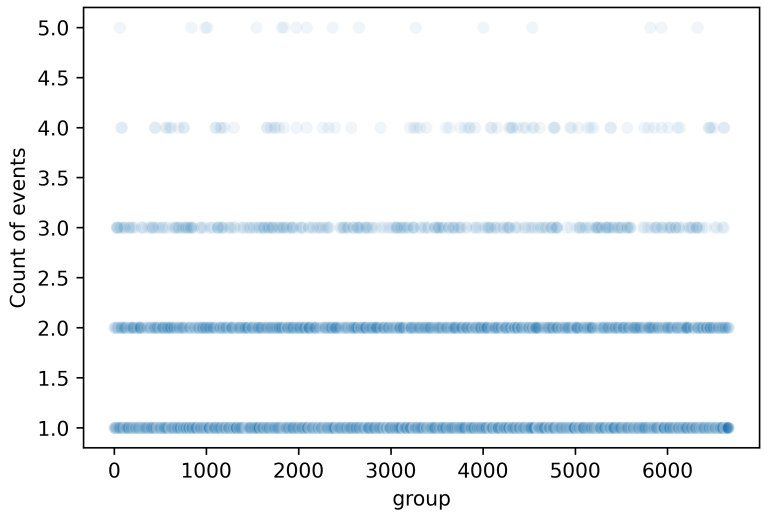


Figure 8.3: Number of Energy Events Per Timeslice

HDF5 files containing `mc_hits` and `mc_info` tables were used (Section 4). The energy events in the data were specifically in the range of 10 to 100 Gigaelectron-volts (GeV). `mc_info` contained additional energy information corresponding to the event hits in the `mc_hits` table. It contained energy values (`nu_E`), the type of neutrino (`type`) and the start (`nu.hits.start`) and end times of the event hits (`nu.hits.end`). The `mc_hits` and `mc_info` tables were combined and grouped into timeslices of 15000 ns to obtain the complete events dataset with corresponding energy values. The scatter plot in Figure 8.3

shows the count of events that occurred within each timeslice. Most of the timeslices contained a single event while very few timeslices contained more than 3 events.

Target and predictor variables were defined. `energy` was set to be the target variable that the model would predict. `pos_x, pos_y, pos_z` and `time` were selected as the predictor variables that would be used to predict `energy`. The predictor variables were selected using uni-variate feature selection based on the *k-best* scores from *mutual information*. Mutual information (MI) measures the dependency between variables and is zero when the two random variables are independent (72). The correlation heatmap in Figure 8.4 and Table 8.1 show that `energy` had no significant relation with any other variables.

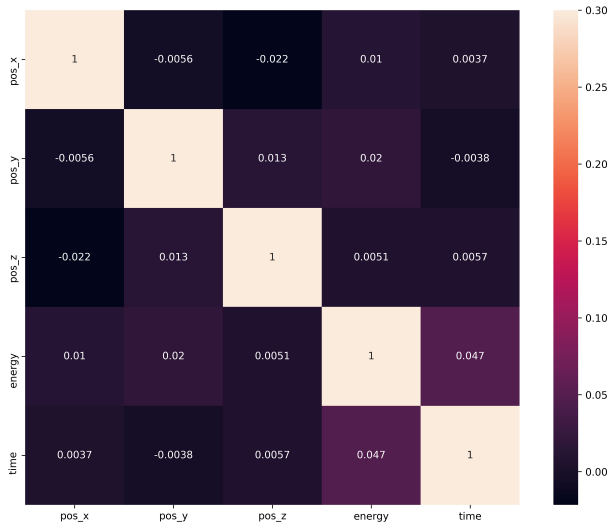


Figure 8.4: Correlation Heatmap of the Events Dataset

`energy` vs `time`	0.05
`energy` vs `pos_x`	0.01
`energy` vs `pos_y`	0.02
`energy` vs `pos_z`	0.005

Table 8.1: Correlation Coefficients: `energy` with `pos_x, pos_y, pos_z` and `time`

While all other key variables were found to have skewness close to 0, the target variable was **0.88**. This indicated a high positive skew, affirmed via the non-normal density and probability plots in Figure 8.5.

It was evident that some transformation was required to bring the target as close to normal as possible. Three types of transformation functions were applied - *log*, *squared*,

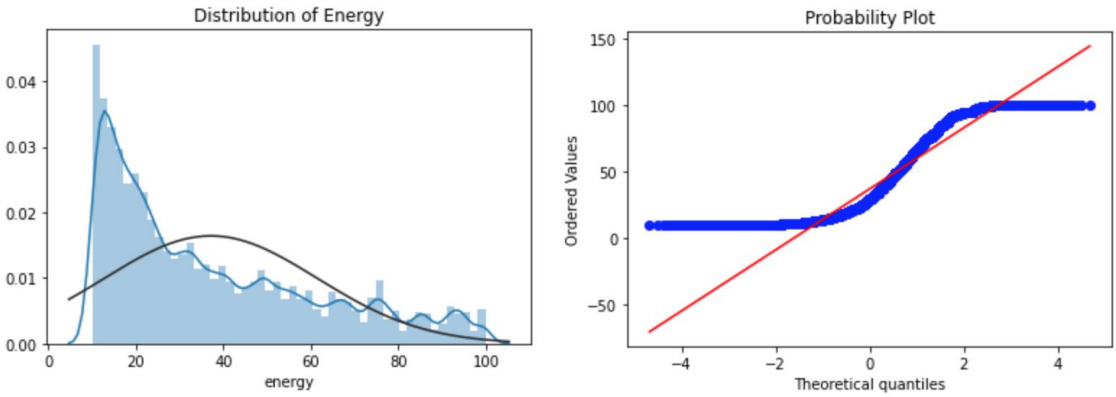
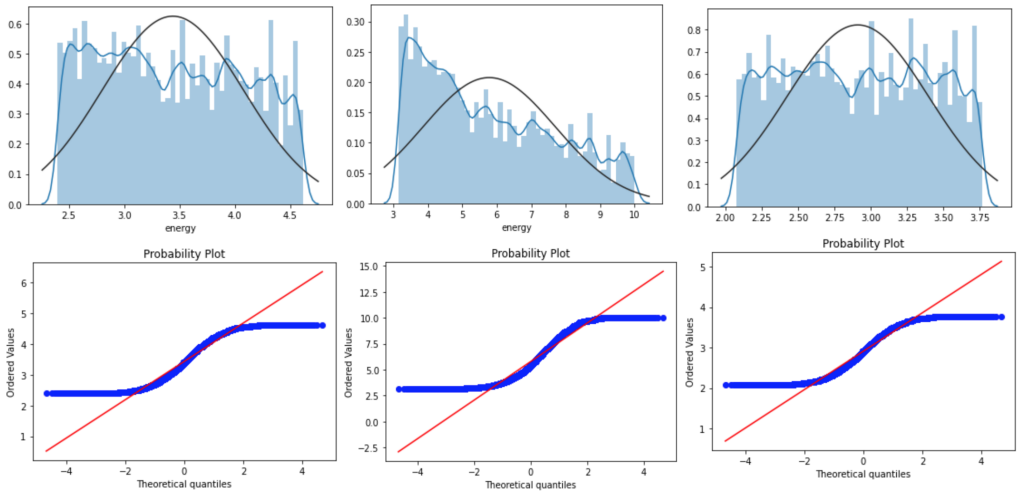


Figure 8.5: Density and Probability Plot

and *Box-Cox*, and results were compared (73, 74). Log and squared transformations involve taking the logarithm and square roots respectively. Box-Cox transformation makes use of λ to approximate the best fitting values for the data (75).

$$y = \begin{cases} (\frac{x^{\lambda-1}}{\lambda}, \forall\lambda > 0 \\ log(x), \forall\lambda = 0 \end{cases} \tag{8.1}$$

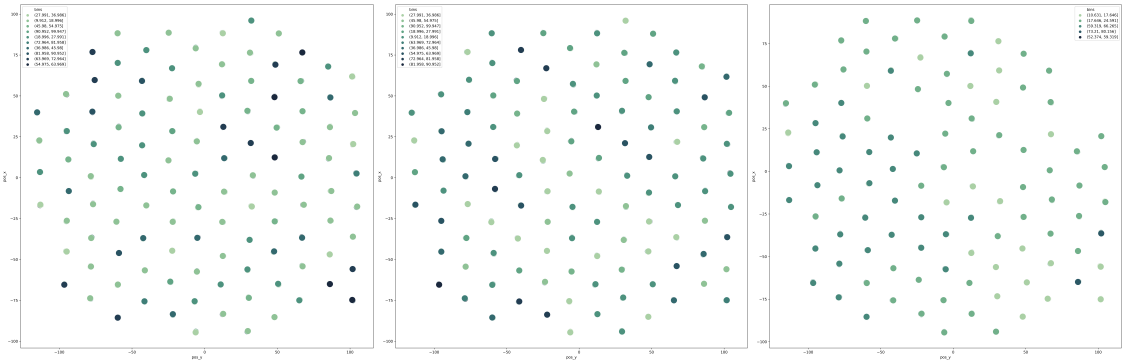


(a) Log Transformation **(b)** Squared Transformation **(c)** Box-Cox Transformation

Figure 8.6: Transformations of the Target Variable `energy`

Figure 8.6 shows the application of each function. The skew after log transformation was **0.12** and the corresponding plot shows a more bell-shaped curve indicating some normalisation (Figure 8.6a). The squared transformation did not result in useful improvements,

indicated by the still right skewed data (Figure 8.6b), and a skew value of **0.49**. Better results were obtained from the Box-Cox transformation that had the lowest skew of **0.02** and the corresponding plot in Figure 8.6c showed a distribution aligned towards normal. Thus the target variable was transformed using Box-Cox.



(a) Training Data **(b)** Testing Data **(c)** Additional Evaluation Data

Figure 8.7: Distribution of Energy for Events in Training, Testing and Additional Evaluation Data

50 timeslices from the dataset were randomly selected and kept aside for evaluation. The dataset was then split into training and testing components using a 66/34 split. Figure 8.7 shows the distribution of energy corresponding to events within the training, testing and additional evaluation data. Here, energy values were binned into 10 groups for a simpler visualisation. Sub-figures 8.7a and 8.7b show that both the training set and testing set contained a good balance between low and high energy events. The additional evaluation set in Sub-figure 8.7c shows that most of the events had low energy, especially under 30 GeV. This indicated that the model could be evaluated against a challenging dataset since lower energy events do not have many hits associated with them.

Standard regression metrics, Mean Squared Error (MSE) and coefficient of determination (R2) were chosen for performance measurement. MSE indicates the average squared difference between the estimated value and actual value (76). R2 metric provides an indication of the goodness of fit of a set of predictions to the actual values (76). In order to get a complete unbiased picture, both metrics were used.

8.3.2 Decision Trees Regressor

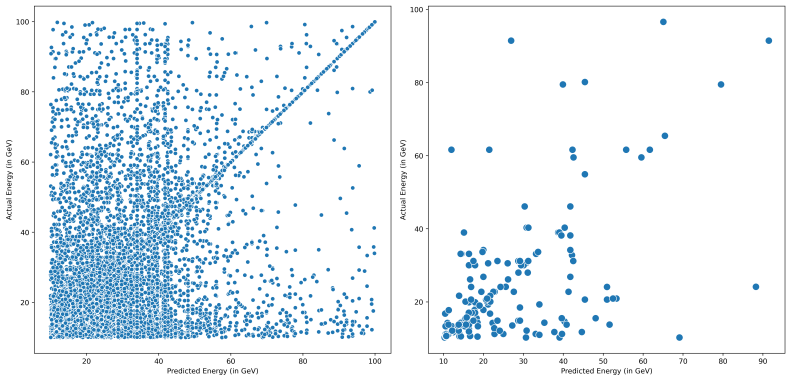
Decision Trees are a non-parametric, supervised learning method where the model makes predictions by learning simple decision rules inferred from data features (77). They were

chosen because they require very little data preparation and are computationally efficient (77). 342,920 samples were selected for training and 146,966 samples were used for testing. Sklearn's `DecisionTreeRegressor` was used to initialise a decision tree, with parameters based on experimentation. Tree depth was `20` and the minimum samples required to be at a leaf node was set to be `1`.

	Training	Testing	Additional
MSE	100.3	116.3	269.0
R2	0.83	0.80	0.31

Table 8.2: Decision Tree Regression Metrics

MSE and R2 were used as metrics of performance (Table 8.2) for the training, testing and additional 50 timeslices. The model obtained a **0.80** R2 score on the testing data indicating a good fit. However, the MSE value on the testing set was **116**, which indicated large errors. After training and testing, the model was evaluated on the 50 timeslices. Here, it showed an even larger magnitude of error of **200** and R2 scores of **0.32**. Therefore, while the model performed well during training and testing, with no data leakage, it had lower performance on the 50 holdout timeslices. With no data leak in place, this has to be attributed to the complexity of the examples provided and overfitting.



(a) Testing Data **(b)** 50 Evaluation Timeslices

Figure 8.8: Decision Trees Regressor: Actual vs Predicted Energy Values on Testing and Evaluation Data

Figure 8.8 shows the plot of actual versus predicted energy values on the testing and evaluation data. The model was better at predicting lower energy values. High incidence of points in the upper left triangle of Sub-figure 8.8a further indicate that the model was biased towards predicting lower energy values for higher energy events.

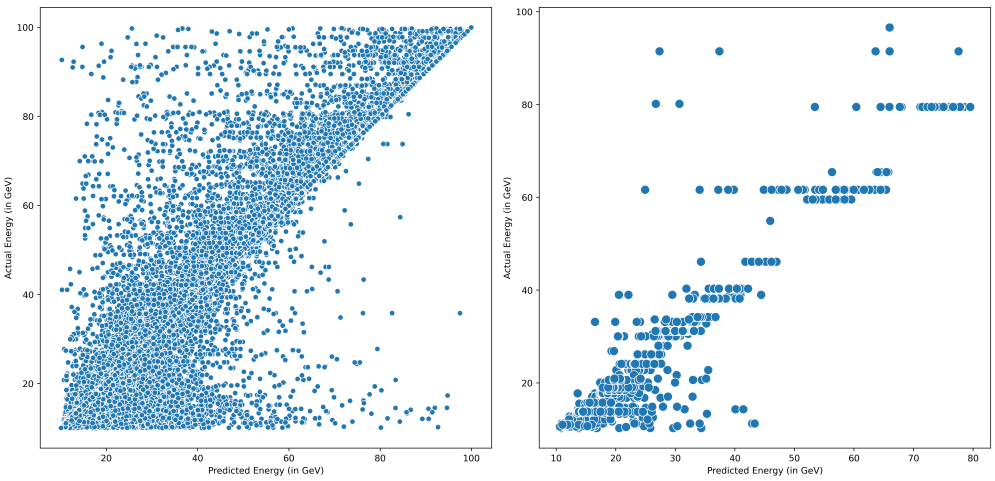
8.3.3 Random Forest Bootstrapping Regressor

A random forest is an estimator that fits a number of decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. With the same train/test setup, 50 trees were chosen to build the model.

	Training	Testing	Additional
MSE	2.6	15.93	79.8
R2	0.99	0.97	0.80

Table 8.3: Random Forest (Bootstrap Aggregation) Metrics

This time, both the MSE and R2 results were very promising (Table 8.3). MSE showed lower errors of **15.93** on the testing data and the high R2 value of **0.97** showed good model fit. However, there is evidence of overfitting due to the difference between MSE scores during training and test. Additionally, evaluation on the 50 data samples gave significantly better results than the other models. While the MSE indicated higher errors, the R2 value of **0.80** showed a good fit with the data. The model was able to predict energy up to 12.138 GeV for events containing only 32 hits.



(a) Testing Data **(b)** 50 Evaluation Timeslices

Figure 8.9: Random Forest Regressor: Actual vs Predicted Energy Values on Testing and Evaluation Data

Figure 8.9 shows the predicted versus actual energy values on the training and 50 holdout timeslices. The model is much better at predicting across all levels of energy due to the closeness of the points to the diagonal. Sub-figure 8.9a indicates that the model still

predicted some high energy events as lower energy events.

8.3.4 Regression Analysis

Overall, random forests performed the best out of the applied models. The minimum energy predicted was 12.138 GeV for events containing only 32 hits. This could be attributed to the algorithm itself. They are ensemble algorithms comprising multiple trees (62). The models are diverse since each tree is learnt on a random sample of the data and at each node, a random set of feature are considered (62). Decision trees are known to overfit and learn the data. This was noted in the experiment due to the difference in R2 values between training and testing (Table 8.2). While the tree was pruned by lowering the maximum depth of the tree, it still showed overfitting. It could be because the problem itself is too hard for the tree to learn and not suited for the Decision tree's learning rules. The algorithm may also require more complex pruning techniques such as weight-based pre-pruning (62, 77). In both cases, the model was biased towards predicting high energy events as low energy events. This was atypical to results from other energy inference research (25, 78, 79) where models were better at predicting higher energy events associated with a larger number of hits. This bias may be due to the splitting parameter set in the experiments. A large value for splitting trees results in a deeper tree with "cleaner" nodes and higher variance, but a lower value limits the splits, leading to higher bias and lower variance (62).

8.4 Summary

Figures 8.10 and 8.11 summarise the two alternate pipeline approaches examined in this chapter. Figure 8.10 outlines the alternate 3D points-based pipeline. Three permutations of the KM3NeT dataset were obtained and processed using radius-based outlier filter, but no 3D meshes were generated. This pipeline was explored to try and achieve faster processing, but it did not result in suitable learning. Figure 8.11 outlines the second alternative pipeline. The pipeline employed 4D data, ie., it did not make use of the three dataset permutations. Again, no 3D meshes were generated. This approach too demonstrated the network's inability to classify between the two classes. Finally, energy inference experiments indicated random forests as the best candidate for regression-based approach. Lack of through testing with varied, larger datasets however, indicate more work in the area.

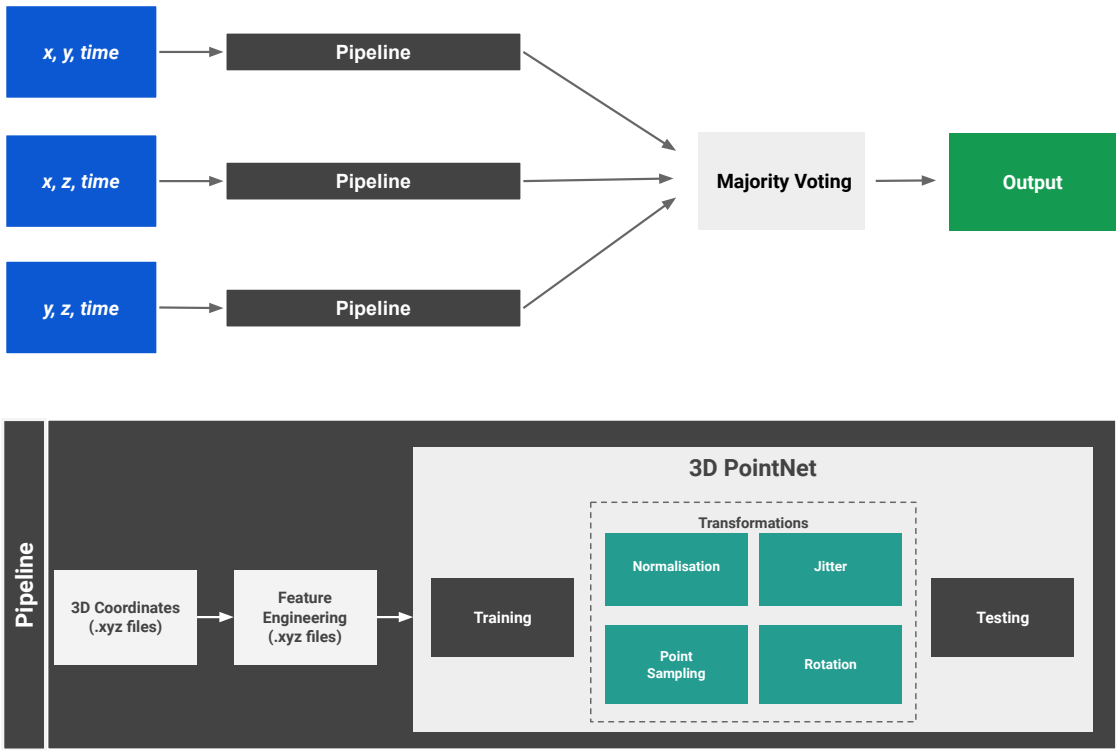


Figure 8.10: Alternate Pipeline I: 3D Points-based PointNet

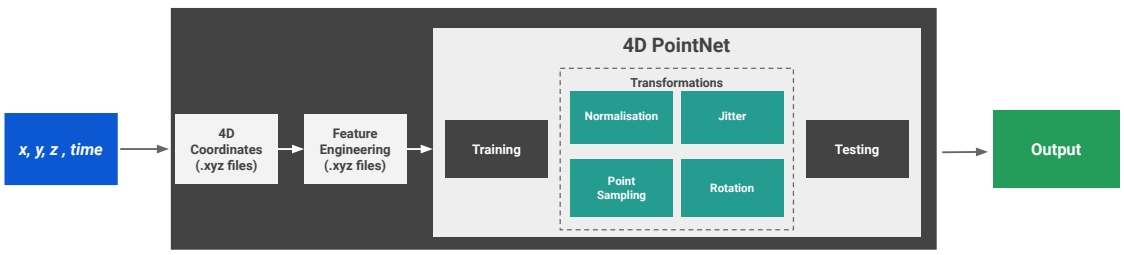


Figure 8.11: Alternate Pipeline II: 4D PointNet

9

Limitations and Recommendations

The thesis was able to successfully conclude on the role of PointNet and point-based learning for neutrino detection. However, a few key drawbacks need to be addressed as part of future work.

Despite feature engineering, PointNet was unable to accurately classify the edge cases where timeslices had very few event hits. This was identified to be due to the loss of information at the feature engineering step involving the surface based reconstruction technique (Section 6). Surface based techniques like the Poisson Reconstruction make use of statistical assumptions about the underlying point cloud model (80). These may not be reliable or available in largely unordered clouds like the KM3NeT data. An alternate would be to use a more sophisticated algorithm such as PointCleanNet - a deep learning method that classifies and discards outlier samples (80). The approach is considered to be efficient and robust to varying amounts of noise and outliers within large densely-sampled point clouds (80).

The Surface Poisson Reconstruction algorithm also requires several parameter specifications. Often, these parameters are arbitrary and depend on the specific dataset being used (54). There is no way to ensure that the parameters used in this thesis may work well on a different version of the dataset. This lack of generalisability is a limiting factor of the Surface Poisson Reconstruction algorithm.

Another drawback of PointNet is that it is reliant on feature engineering to learn from the data. But given the complicated, irregular geometry of the KM3NeT data, it is unlikely that the network will be able to learn without additional features. This is since PointNet requires equal number of points to be randomly sampled per point cloud while training (1). The random sampling of points does not guarantee that sufficient event hits will be picked up, considering that they are very infrequently occurring compared to noise. PointNet2 is a

next generation improvement over PointNet (35). It learns hierarchical features similar to Convolutional Neural Networks and observes non-uniform densities in natural point clouds (35). Therefore, PointNet2 does not require equal, random sampling from every cloud. This could be a recommended alternative to PointNet pipeline.

PointNet was deemed unsuitable for regression as its architecture was primarily developed for classification. The thesis alternatively conducted research into energy inference using non-linear regression techniques. However, these experiments are preliminary. Results showed that they were partial towards predicting lower energy values, even for high energy events. The bias could be addressed through different parameters or by using boosting methods (81).

Additional testing would also be required to validate the energy predictions obtained. Only 3D coordinates and time were used for regressing energy values, therefore the predictions are only derived from these attributes. Research on energy inference use techniques such as Maximum Likelihood Estimation (MLE) to estimate various parameters. These parameters are then used to train decision trees (78). Some studies also address uncertainties arising from DOM sensitivity, water properties, simulation and statistical uncertainties (78, 79, 82). The drawback of these solutions is the significant reliance on a-priori information which is what the thesis attempted to correct.

In terms of performance, the overall execution time for the pipeline from preprocessing to evaluation was around 3 minutes. This is not ideal in real-time processing pipelines. However, long execution times are a known fallacy of deep learning models that employ complicated algorithms. Moreover, the pre-processing pipeline was evaluated on the CPU with only two cores. Improvements to the execution time could be obtained by porting the pre-processing pipeline to the GPU. Also, significant speedups could be obtained by parallelising the pre-processing of the three datasets.

10

Conclusion

Particle detectors are often the source for studying and collecting data in particle physics. These instruments gather exabytes of data that need to be thoroughly probed for relevant signals. While the state-of-the-art in hardware has significantly improved and allowed for detailed data collection, traditional physics controls are often unable to keep up with the large, high dimensional, irregular data. With this gap between information extraction techniques and information gathering systems, attention has shifted to artificial intelligence. Research is required to explore the feasibility of state-of-the-art architectures to particle data before they can be incorporated into the processing pipeline. This thesis examined one such state-of-the-art deep learning architecture - PointNet, and its ability to learn from 3D mesh representation of neutrino data.

The goal of the thesis was to build a classification network that could label a timeslice as `0` or `1`, such that noise could be discarded and events could be saved. The three research questions were revisited based on the results from the thesis. In order to answer **RQ1.0** and **RQ 1.1**, **RQ 2.0** and **RQ 2.1** were first resolved.

RQ2.0 *Can the KM3NeT dataset be effectively represented using 3D meshes?*

3D mesh representations of the feature engineered point clouds demonstrated that meshes are a valid representation of the KM3NeT data. Moreover, secondary experiments that used 3D and 4D point clouds showed insufficient learning through means of low precision and recall scores (Section 8). 3D meshes likely outperformed point-based learning because the meshes added more information to the point clouds in the form of mesh faces and normals. Moreover, PointNet requires a fixed number of points to be randomly sampled per point cloud. With 3D meshes, the thesis was able to sample per mesh face, retaining much of the shape of the data.

RQ2.1 *Which meshing algorithm would be most suitable for representing the data?*

Poisson Surface Reconstruction was found to be most suitable for representing KM3NeT data. Based on visual examinations, it showed greater detail, especially around event clusters (Section 5). It was however not able to capture details in timeslices with few event hits. Ball-Pivoting Algorithm was also used to reconstruct point clouds. While BPA was found to be significantly faster in building meshes, it was too simplistic for the detailed event clusters (Section 5).

RQ1.0 *Can PointNet, a geometric Neural Network architecture be trained to classify timeslices that contain neutrino event hits from timeslices that contain only background noise?*

PointNet can be successfully trained to classify timeslices of the KM3NeT data. This was indicated via high recall scores for the positive class and the Precision-Recall curves, measured via Hard Voting (Section 7). It also outperformed the existing L1 Trigger, which had a higher false positive rate. However, PointNet was not able to correctly classify timeslices with events containing a few hits, for example 30 to 40 event hits.

RQ1.1 *Can PointNet achieve a Precision-Recall score of greater than 0.9 for identifying timeslices with event hits?*

Hard-voting predicted a 0.95 recall and perfect precision scores for `class_1` containing event hits. Therefore, it met stakeholder expectations. However, it was only able to do so with feature engineering and mesh representations. Without feature engineering, it was unable to learn between classes (Appendix 10), and without meshes, it scored no more than 0.70 on recall and precision (Section 8).

RQ3.0 *Can PointNet be extended to obtain energy properties from neutrino events?*

The thesis does not recommend extending PointNet for regression tasks. As PointNet is built for classification, it would require several changes to make it suitable for regression. Further, mapping the energy values to the completely randomised and transformed point clouds after classification tasks would be inefficient. Experiments are instead conducted using random forests. The trained model can predict a minimum energy of 12.138 GeV for events containing 32 hits.

The pipeline in Figure 10.1 was the finalised thesis pipeline. Three permutations of the KM3NeT dataset were obtained and processed using radius-based outlier filter. They were then converted to 3D meshes and trained using PointNet. Several transformation functions

were applied to the data during training and testing. Majority voting was used to ensemble results and obtain final output.

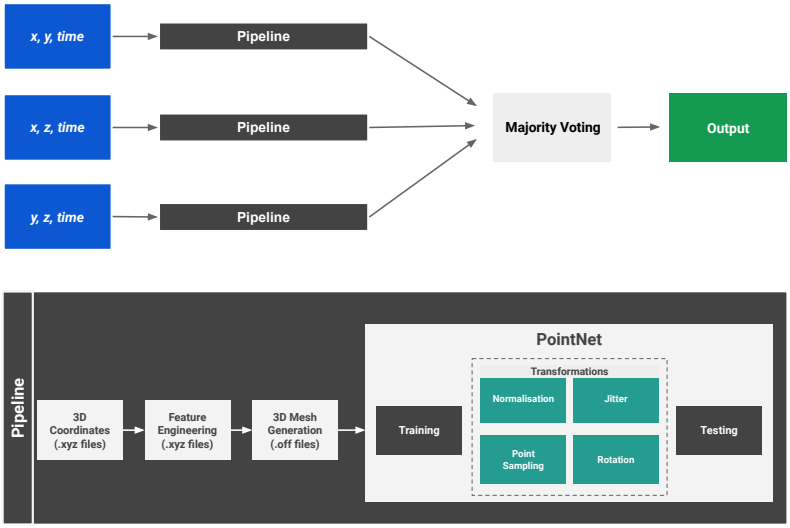


Figure 10.1: The Complete Process for KM3NeT Timeslice Classification

Despite the noted limitations, this thesis lays the groundwork for 3D point-based deep learning for neutrino identification amidst noise. The thesis is the first known application of 3D point-based learning for neutrino detection. It is also the first known work to use 3D meshes to represent neutrino data and achieve high precision and recall scores.

Results from this thesis can be used by physicists at KM3NeT to assess the feasibility of adopting PointNet into the pipeline. They could also extend the architecture to classify the three neutrino flavours (25). The event trigger is an important aspect of the KM3NeT pipeline, both from a fiscal and physics perspective as it determines data that needs to be saved or discarded (25). The methodology developed in this thesis demonstrates a high recall and low false-positive rate. Thus, making use of this as a KM3NeT pipeline would both minimise the noise being saved and ensure that timeslices with events are saved with a high accuracy. Results from this thesis can also help ascertain the validity of novel point-based learning for particle physics data. The methods undertaken, the problems faced and the results obtained could serve as a starting point for others in particle physics wishing to adopt Neural Networks to their own work.

While this thesis serves as a starting point for examination of novel deep learning architectures for neutrino research, there is certainly more work required to understand how complex networks could be tuned to meet the end goals in the field. Additional studies would be required to understand performance of the pipeline on the GPU. Adopting deep

learning always presents a trade-off between superior accuracy of results and longer compute times. These trade-offs would require careful examination against existing methodology. Despite these gaps, PointNet at its current state has a very promising role in the future of neutrino research and by extension particle physics.

Appendix

PointNet Model Layers

The model layers and corresponding parameters used in the thesis pipeline (Section 5, 6, and 7) were generated using PyTorch. The following layers were also used for 3D points-based PointNet (Section 8.1

```
PointNet(
  (transform): Transform(
    (input_transform): Tnet(
      (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
      (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
      (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
      (fc1): Linear(in_features=1024, out_features=512, bias=True)
      (fc2): Linear(in_features=512, out_features=256, bias=True)
      (fc3): Linear(in_features=256, out_features=9, bias=True)
      (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                                          track_running_stats=True)
      (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                                          track_running_stats=True)
      (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
                                          track_running_stats=True)
      (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
                                          track_running_stats=True)
      (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
                                          track_running_stats=True)
    )
    (feature_transform): Tnet(
      (conv1): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
      (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
```

```
    (fc1): Linear(in_features=1024, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=256, bias=True)
    (fc3): Linear(in_features=256, out_features=4096, bias=True)
    (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
    (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
    (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
    (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
    (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
  )
  (conv1): Conv1d(3, 64, kernel_size=(1,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
  (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
                                        track_running_stats=True)
)
(fc1): Linear(in_features=1024, out_features=512, bias=True)
(fc2): Linear(in_features=512, out_features=256, bias=True)
(fc3): Linear(in_features=256, out_features=2, bias=True)
(bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
                                    track_running_stats=True)
(bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
                                    track_running_stats=True)
(dropout): Dropout(p=0.3, inplace=False)
(logsigmoid): LogSigmoid()
)
```

The following model summary describes the model used for the 4D PointNet experiment (Section 8.2).

```
PointNet(
```

```
(transform): Transform(
  (input_transform): Tnet(
    (conv1): Conv1d(4, 64, kernel_size=(1,), stride=(1,))
    (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
    (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
    (fc1): Linear(in_features=1024, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=256, bias=True)
    (fc3): Linear(in_features=256, out_features=16, bias=True)
    (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
                                       track_running_stats=True)
    (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
                                       track_running_stats=True)
    (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
                                      track_running_stats=True)
    (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
                                      track_running_stats=True)
    (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
                                      track_running_stats=True)
  )
  (feature_transform): Tnet(
    (conv1): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
    (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
    (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
    (fc1): Linear(in_features=1024, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=256, bias=True)
    (fc3): Linear(in_features=256, out_features=4096, bias=True)
    (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
    (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
    (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
    (bn4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
    (bn5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
  )
  (conv1): Conv1d(4, 64, kernel_size=(1,), stride=(1,))
  (conv2): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (conv3): Conv1d(128, 1024, kernel_size=(1,), stride=(1,))
  (bn1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
  (bn2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
  (bn3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
```

```
  )
  (fc1): Linear(in_features=1024, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=2, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
  (bn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
  (dropout): Dropout(p=0.3, inplace=False)
  (logsigmoid): LogSigmoid()
)
```

Model with No Feature Engineering

The model was trained without any feature engineering, and results were collected from hard voting. Table 10.1 indicates that the model was able to perfectly identify and classify event timeslices. However, it was not able to identify and label a single noise timeslice. So, despite the perfect recall for `class_1`, the model shows no learning ability. This is further confirmed by the Precision-Recall (PR) and Receiver Operating Characteristic (ROC) Curve in Figure 10.2. The ROC plot shows curves through the diagonal, indicating that the classifier was completely random and learnt nothing. The PR curve for `class_1` is also through the diagonal, indicating that its performance was equivalent to a model with no skill. The model could in a different instance predict the reverse, ie., perfectly classify noise and none of the event timeslices.

Hard Voting Results: `(x y time), (x z time) (y z time)`				
	precision	recall	F1-score	support
`class_0`	0.00	0.00	0.00	40
`class_1`	0.50	1.00	0.67	40

Table 10.1: No Feature Engineering: Classification Report for `class_0` and `class_1`

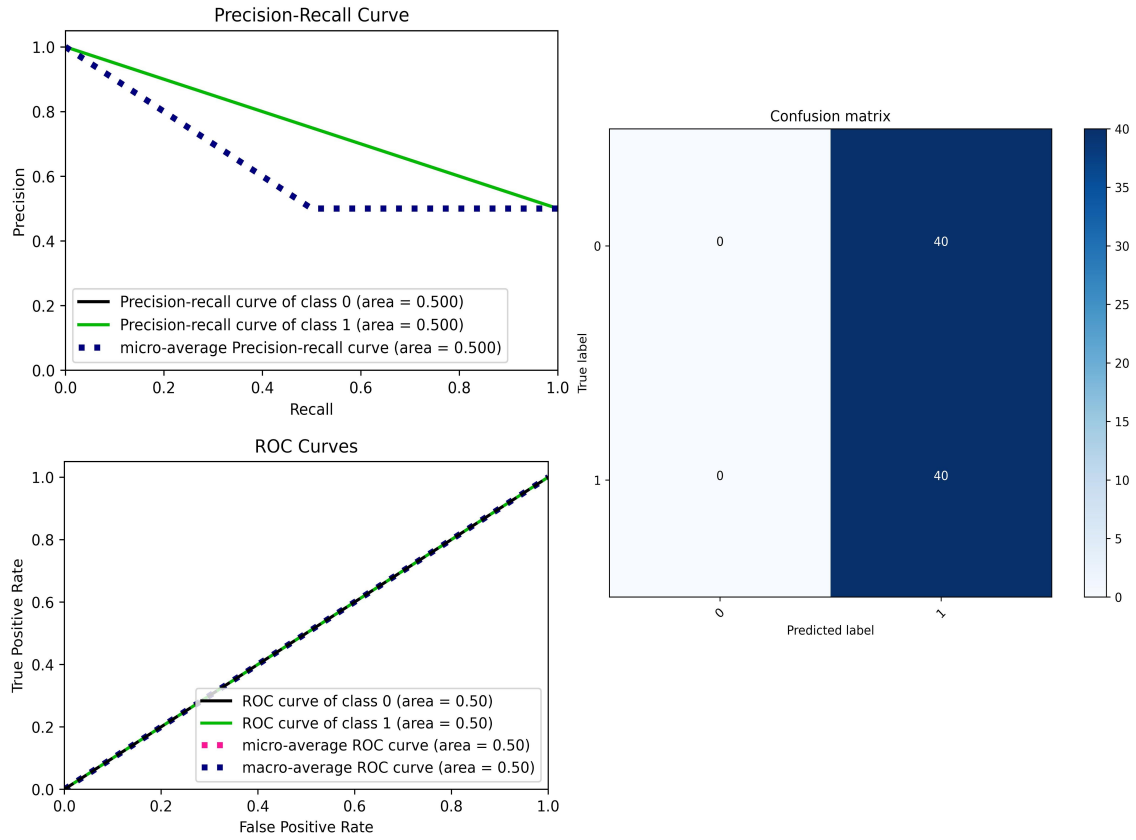


Figure 10.2: Classification Metrics: Model without Feature Engineering

References

[1] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. **Pointnet: Deep learning on point sets for 3d classification and segmentation**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. v, 6, 11, 12, 13, 14, 15, 35, 36, 37, 41, 54, 56, 65

[2] Jacob D Bekenstein. **Black holes and entropy**. *Physical Review D*, **7**(8):2333, 1973. 1

[3] Jonathan J Halliwell and Stephen William Hawking. **Origin of structure in the universe**. *Physical Review D*, **31**(8):1777, 1985. 1

[4] SM Bilenky. **Neutrino. History of a unique particle**. *The European Physical Journal H*, **38**(3):345–404, 2013. 1, 2

[5] Arthur Roberts. **The birth of high-energy neutrino astronomy: A personal history of the DUMAND project**. *Reviews of Modern Physics*, **64**(1):259, 1992. 1, 2

[6] FE Gray, C Ruybal, J Totushek, D-M Mei, K Thomas, and C Zhang. **Cosmic ray muon flux at the Sanford Underground Laboratory at Homestake**. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **638**(1):63–66, 2011. 1

[7] R Acciarri, MA Acero, M Adamowski, C Adams, P Adamson, S Adhikari, Z Ahmad, CH Albright, T Alion, E Amador, et al. **Long-baseline neutrino facility (LBNF) and deep underground neutrino experiment (DUNE) conceptual design report, volume 4 the DUNE detectors at LBNF**. *arXiv preprint arXiv:1601.02984*, 2016. 1, 2, 9, 11

[8] R Acciarri, C Adams, R An, J Asaadi, M Auger, L Bagby, B Baller, G Barr, M Bass, F Bay, et al. **Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber**. *Journal of instrumentation*, **12**(03):P03011, 2017. 1, 9, 10, 11

[9] A Albert, M André, M Anghinolfi, G Anton, M Ardid, J-J Aubert, J Aublin, B Baret, S Basa, B Belhorma, et al. **ANTARES and IceCube Combined Search for Neutrino Point-like and Extended Sources in the Southern Sky**. *arXiv preprint arXiv:2001.04412*, 2020. 1, 3

[10] Sebastiano Aiello, SE Akrame, F Ameli, EG Anassontzis, Michel Andre, G Androulakis, Marco Anghinolfi, G Anton, Miguel Ardid, Julien Aublin, et al. **Sensitivity of the KM3NeT/ARCA neutrino telescope to point-like neutrino sources**. *Astroparticle Physics*, **111**:100–110, 2019. 1

[11] D Britton and SL Lloyd. **How to deal with petabytes of data: the LHC Grid project**. *Reports on Progress in Physics*, **77**(6):065902, 2014. 1

[12] AL Edelen, SG Biedron, BE Chase, D Edstrom, SV Milton, and P Stabile. **Neural networks for modeling and control of particle accelerators**. *IEEE Transactions on Nuclear Science*, **63**(2):878–897, 2016. 1, 2, 11

[13] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. **Discrete cosine transform**. *IEEE transactions on Computers*, **100**(1):90–93, 1974. 2

[14] Léon Bottou. **Large-scale machine learning with stochastic gradient descent**. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. 2

[15] B Abi, R Acciarri, MA Acero, M Adamowski, C Adams, D Adams, P Adamson, M Adinolfi, Z Ahmad, CH Albright, et al. **The DUNE far detector interim design report volume 1: physics, technology and strategies**. *arXiv preprint arXiv:1807.10334*, 2018. 2

[16] John N Bahcall. *Neutrino astrophysics*. Cambridge University Press, 1989. 3

[17] P Adamson, C Andreopoulos, KE Arms, R Armstrong, DJ Auty, S Avvakumov, DS Ayres, B Baller, B Barish, PD Barnes Jr, et al. **Study of muon neutrino disappearance using the Fermilab Main Injector neutrino beam**. *Physical Review D*, **77**(7):072002, 2008. 3, 4

[18] Antonio Puccini. **On the Bosons' Range of the Weak Interaction**. *Computer Physics Communications*, 2018. 3

[19] Masahiro Tanaka, K Abe, C Bronner, Y Hayato, M Ikeda, S Imaizumi, H Ito, J Kameda, Y Kataoka, Y Kato, et al. **Search for proton decay into three charged leptons in 0.37 megaton-years exposure of the Super-Kamiokande**. *Physical Review D*, **101**(5):052011, 2020. 3

[20] AA Aguilar-Arevalo, M Backfish, A Bashyal, B Batell, BC Brown, R Carr, A Chatterjee, RL Cooper, P Deniverville, R Dharmapalan, et al. **Dark matter search in a proton beam dump with MiniBooNE**. *Physical review letters*, **118**(22):221803, 2017. 3

[21] Bruce Denby. **Neural networks in high energy physics: a ten year perspective**. *Computer Physics Communications*, **119**(2-3):219–231, 1999. 3, 4, 5

[22] GA Voss and BH Wiik. **The electron-proton collider HERA**. *Annual Review of Nuclear and Particle Science*, **44**(1):413–452, 1994. 4

[23] Halina Abramowicz, Allen Caldwell, and Ralph Sinkus. **Neural network based electron identification in the ZEUS calorimeter**. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **365**(2-3):508–517, 1995. 4

[24] Serguei Chatrchyan, V Khachatryan, AM Sirunyan, W Adam, B Arnold, H Bergauer, T Bergauer, M Dragicevic, M Eichberger, J Ero, et al. **Performance of the CMS Level-1 trigger during commissioning with cosmic ray muons and LHC beams**. *Journal of Instrumentation*, **5**, 2010. 4

[25] Adrian et al. Martinez. **A Letter of Intent for KM3NeT 2.0**, Apr 2017. 4, 6, 19, 25, 56, 63, 69

[26] Konrad Karas. **Data processing pipeline for the KM3NeT neutrino telescope**, 2019. 5, 19, 20, 53

[27] A Aurisano, A Radovic, D Rocco, A Himmel, MD Messier, E Niner, G Pawloski, F Psihas, Alexandre Sousa, and P Vahle. **A convolutional neural network neutrino event classifier**. *Journal of Instrumentation*, **11**(09):P09001, 2016. 8, 10, 11

[28] Zbigniew Szadkowski and Krzysztof Pytel. **Artificial neural network as a FPGA trigger for a detection of very inclined air showers**. *IEEE Transactions on Nuclear Science*, **62**(3):1002–1009, 2015. 8, 11

[29] MicroBooNE Collaboration. **Convolutional Neural Networks Applied to Neutrino Events in a Liquid Argon Time Projection Chamber**. *arXiv preprint arXiv:1611.05531*, 2016. 9, 10

[30] MicroBooNE Collaboration, C Adams, M Alrashed, R An, J Anthony, J Asaadi, A Ashkenazi, M Auger, S Balasubramanian, B Baller, et al. **Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber**. *Physical Review D*, **99**(9):092001, 2019. 10

[31] Liangpei Zhang, Ke Wu, Yanfei Zhong, and Pingxiang Li. **A new sub-pixel mapping algorithm based on a BP neural network with an observation model**. *Neurocomputing*, **71**(10-12):2046–2054, 2008. 10

[32] IceCube Collaboration et al. **Evidence for high-energy extraterrestrial neutrinos at the IceCube detector**. *Science*, **342**(6161):1242856, 2013. 11

[33] 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. 11

[34] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. **Searching for exotic particles in high-energy physics with deep learning**. *Nature communications*, **5**:4308, 2014. 11

[35] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. **Pointnet++: Deep hierarchical feature learning on point sets in a metric space**. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 12, 66

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. **Convolutional networks**. In *Deep learning*, **2016**, pages 330–372. MIT Press Cambridge, MA, 2016. 12, 35, 42

[37] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. **Dynamic Graph CNN for Learning on Point Clouds**. *ACM Trans. Graph.*, **38**(5), October 2019. 13

[38] Goodfellow Ian and Courville Aaron. **Deep Learning (Adaptive Computation and Machine Learning series)**, 2016. 14

[39] Michael Waskom and the seaborn development team. **mwaskom/seaborn**. *Journal of Machine Learning Research*, **12**:2825–2830, September 2020. 21, 22

[40] Aron Eklund. **Beeswarm: the bee swarm plot, an alternative to stripchart**. *R package version 0.1*, **5**, 2012. 24

[41] Ikuru Otomo, Masahiko Onosato, and Fumiki Tanaka. **Direct construction of a four-dimensional mesh model from a three-dimensional object with continuous rigid body movement**. *Journal of Computational Design and Engineering*, **1**(2):96–102, 2014. 25

[42] Isabelle Guyon and André Elisseeff. **An introduction to variable and feature selection**. *Journal of machine learning research*, **3**(Mar):1157–1182, 2003. 27

[43] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. **MeshLab: an Open-Source Mesh Processing Tool**. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. 28, 34

[44] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. **Parameterization-free projection for geometry reconstruction**. *ACM Transactions on Graphics (TOG)*, **26**(3):22–es, 2007. 28

[45] Yuankai Qi, Shengping Zhang, Lei Qin, et al. **Hedged Deep Tracking. Computer Vision and Pattern Recognition**, 2016. 28

[46] Xiaojuan Ning, Fan Li, Ge Tian, and Yinghui Wang. **An efficient outlier removal method for scattered point cloud data**. *PloS one*, **13**(8):e0201280, 2018. 28, 29

[47] Leif Kobbelt and Mario Botsch. **A survey of point-based techniques in computer graphics**. *Computers & Graphics*, **28**(6):801–814, 2004. 28

[48] Dawei Li, Yan Cao, Guoliang Shi, Xin Cai, Yang Chen, Sifan Wang, and Siyuan Yan. **An overlapping-free leaf segmentation method for plant point clouds**. *IEEE Access*, **7**:129054–129070, 2019. 29

[49] Niloy J Mitra and An Nguyen. **Estimating surface normals in noisy point cloud data**. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 322–328, 2003. 30

[50] Haihua Cui, Wenhe Liao, Xiaosheng Cheng, Ning Dai, and Changye Guo. **Flexible point cloud matching method based on three-dimensional image feature points**. *Advances in Mechanical Engineering*, **10**(9):1687814018795032, 2018. 31

[51] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. **Open3D: A modern library for 3D data processing**. *arXiv preprint arXiv:1801.09847*, 2018. 31

[52] Seok-Il Kim and Rixie Li. **Complete 3D surface reconstruction from unstructured point cloud**. *Journal of mechanical science and technology*, **20**(12):2034–2042, 2006. 33

[53] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. **The ball-pivoting algorithm for surface reconstruction**. *IEEE transactions on visualization and computer graphics*, **5**(4):349–359, 1999. 33

[54] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. **Poisson surface reconstruction**. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, **7**, 2006. 34, 65

[55] Min Lin, Qiang Chen, and Shuicheng Yan. **Network in network**. *arXiv preprint arXiv:1312.4400*, 2013. 35

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. **Pytorch: An imperative style, high-performance deep learning library**. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 35

[57] Matthew F Dixon, Nicholas G Polson, and Vadim O Sokolov. **Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading**. *arXiv preprint arXiv:1705.09851*, 2017. 39

[58] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, **112**. Springer, 2013. 39

[59] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. **Evolving diverse ensembles using genetic programming for classification with unbalanced data**. *IEEE Transactions on Evolutionary Computation*, **17**(3):368–386, 2012. 42

[60] Ian H Witten and Eibe Frank. **Data mining: practical machine learning tools and techniques with Java implementations**. *Acm Sigmod Record*, **31**(1):76–77, 2002. 42

[61] Mohammad Hossin and MN Sulaiman. **A review on evaluation metrics for data classification evaluations**. *International Journal of Data Mining & Knowledge Management Process*, **5**(2):1, 2015. 44

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. **Scikit-learn: Machine Learning in Python**. *Journal of Machine Learning Research*, **12**:2825–2830, 2011. 44, 45, 55, 63

[63] Tariq Jaffery and Shirley X Liu. **Measuring campaign performance by using cumulative gain and lift chart**. In *SAS Global Forum*, page 196, 2009. 45

[64] Luting Yang, Bingqian Lu, and Shaolei Ren. **A Note on Latency Variability of Deep Neural Networks for Mobile Inference**. *arXiv preprint arXiv:2003.00138*, 2020. 51

[65] David A Teich and Paul R Teich. **PLASTER: A Framework for Deep Learning Performance**. Technical report, Tech. rep. TIRIAS Research, 2018. 51

[66] Jussi Hanhirova, Teemu Kämäräinen, Sipi Seppälä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. **Latency and throughput characterization of convolutional neural networks for mobile computer vision**. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 204–215, 2018. 51, 52

[67] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. **Carbontracker: Tracking and predicting the carbon footprint of training deep learning models**. *arXiv preprint arXiv:2007.03051*, 2020. 52

[68] Emma Strubell, Ananya Ganesh, and Andrew McCallum. **Energy and policy considerations for deep learning in NLP**. *arXiv preprint arXiv:1906.02243*, 2019. 52

[69] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. **Pointnetlk: Robust & efficient point cloud registration using pointnet**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7163–7172, 2019. 55

[70] Liuhao Ge, Yujun Cai, Junwu Weng, and Junsong Yuan. **Hand pointnet: 3d hand pose estimation using point sets**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8417–8426, 2018. 55

[71] Alberto Garcia-Garcia, Francisco Gomez-Donoso, Jose Garcia-Rodriguez, Sergio Orts-Escolano, Miguel Cazorla, and J Azorin-Lopez. **Pointnet: A 3d convolutional neural network for real-time object class recognition**. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1584. IEEE, 2016. 55

[72] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. **Erratum: estimating mutual information [Phys. Rev. E 69, 066138 (2004)]**. *Physical Review E*, **83**(1):019903, 2011. 58

[73] Jason Osborne. **Notes on the use of data transformations**. *Practical assessment, research, and evaluation*, **8**(1):6, 2002. 59

[74] David Ruppert and Raymond J Carroll. **Data Transformations in Regression Analysis with Applications to Stock—Recruitment Relationships**. In *Resource Management*, pages 29–47. Springer, 1985. 59

[75] George EP Box and David R Cox. **An analysis of transformations**. *Journal of the Royal Statistical Society: Series B (Methodological)*, **26**(2):211–243, 1964. 59

[76] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*, **821**. John Wiley & Sons, 2012. 60

[77] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. **Classification and regression trees. Belmont, CA: Wadsworth**. *International Group*, **432**:151–166, 1984. 60, 61, 63

[78] Rasha Abbasi, Yasser Abdou, T Abu-Zayyad, J Adams, JA Aguilar, M Ahlers, K Andeen, J Auffenberg, X Bai, M Baker, et al. **Measurement of the atmospheric neutrino energy spectrum from 100 GeV to 400 TeV with IceCube**. *Physical Review D*, **83**(1):012001, 2011. 63, 66

[79] Giacomo D'Amico. **Flavor and energy inference for the high-energy IceCube neutrinos**. *Astroparticle Physics*, **101**:8–16, 2018. 63, 66

[80] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. **Pointcleannet: Learning to denoise and remove outliers from dense point clouds**. In *Computer Graphics Forum*, **39**, pages 185–203. Wiley Online Library, 2020. 65

[81] Tom Dietterich. **Overfitting and undercomputing in machine learning**. *ACM computing surveys (CSUR)*, **27**(3):326–327, 1995. 66

[82] Maicon Hieronymus, Bertil Schmidt, and Sebastian Böser. **Reconstruction of Low Energy Neutrino Events with GPUs at IceCube**. In *International Conference on Computational Science*, pages 118–131. Springer, 2020. 66