

Implementation of a Machine Learning Regression Algorithm for Energy Reconstruction of Neutrino-induced Particle Showers using a Scintillating Fibres Tracker at the SND@LHC

Shania Mitra [I6176442]

Supervisors: Dr. Elena Graverini, Dr. Jacco de Vries and MSc. Paul de Bryas

December 13, 2020



Abstract

SHiP and SND@LHC are two burgeoning experiments, as part of CERN, designed to study novel neutrino and BSM physics. Machine Learning (ML) algorithms need to be developed to extract hit information detected by SciFi planes and reconstruct energies of particle showers generated within the Sampling Calorimeter. This thesis implements real-life scenario into an existing ML algorithm for SND@LHC for particle shower reconstruction. Current algorithms do not tackle the ‘ghost-hits’ problem, however due to their presence in the real detector, these need to be taken into account before an algorithm can find immediate application in the detector. The ghost hit coordinates were appended to existing hit data to observe the increase in energy bias. Additionally, to eliminate the problem of ghost hits altogether, new side-view images of the SciFi planes were produced and fed into the algorithm to observe the change in energy bias and to optimize the network. The ghost hits failed to be implemented in the network, however, the optimized network successfully managed to minimize/eliminate energy bias to a value of 0.06%. The elimination/minimization of ghost hits will lead to greater adaptability and robustness of this ML algorithm to be adapted by the SND@LHC.



Contents

1	Introduction and Motivation	3
1.1	Background	3
1.2	SHiP and SND@LHC	3
1.2.1	A Brief Introduction	3
1.2.2	SND@LHC structure	5
1.3	Machine Learning, CNNs and Energy Reconstruction of Electromagnetic Showers	7
1.3.1	Convolutional Neural Networks – a brief introduction	8
1.3.1.1	Convolutional Layer	8
1.3.1.2	Padding	10
1.3.1.3	ReLu (Rectified Linear Unit) Layer	10
1.3.1.4	Pooling Layers	10
1.3.1.5	Dropout layers	10
1.3.1.6	Batch Normalisation	10
1.3.1.7	Fully Connected Layer	11
1.3.1.8	Training Process	11
1.4	Overall Aims and Hypotheses	11
1.5	Importance of the Research	12
2	Methodology	12
2.1	Preprocessing Raw Data	12
2.2	Original CNN Architecture	13
2.3	Minimising bias and testing Border Effects	13
2.4	The Ghost Hits Problem	14
2.4.1	Implementing Ghost Hits into current Algorithm	14
2.4.2	Side-view, xz and yz images of TT planes and CNN optimisation	15
3	Results and Discussion	15
3.1	Minimising bias and testing Border Effects	15
3.1.1	Minimising Bias with differing Loss Functions	15
3.1.2	Testing Border Effects	18
3.2	Tackling Ghost Hits Problem	18
3.2.1	Ghost Hits Implementation	18
3.2.2	Image Compression and Optimising CNN for new images	21
3.2.2.1	Using 4 CMBR Blocks	21
3.2.2.2	Using 3 CMBR Blocks and altering input and output features	23
3.2.3	Overall Performance of the pre-Optimised and Optimised Network for side-view, xz and yz images	24
3.2.4	Comparison of events with low biases and high biases	24
4	Further Research and Conclusion	26
5	Critical Reflection	28
6	Acknowledgements	28
7	References	28
A	Appendix	31

1 Introduction and Motivation

1.1 Background

The fundamental structure of matter has been found to lie in elementary particles which interact with one another, with four fundamental forces. One of the greatest achievements in High-Energy Physics has been the discovery and expounding of the Standard Model (SM) of Particle Physics. However, despite the elegance of the SM, many unanswered questions remain, such as:

- **Matter and Anti-matter Asymmetry in the Universe:** The SM remains incompatible with the concordant model of cosmology (Λ -CDM) which assumes that the Big Bang produced equal numbers of particles and anti-particles, contradicting experimental results [1].
- **Non-Baryonic Dark Matter:** Cosmological observations propose that Dark Matter (DM) particles, if discovered, would not consist of Baryons and that current SM particles would couple weakly to those in the Dark Sector due to various constraints. Thus, a need arises for potential DM candidates [2].
- **Neutrino masses/oscillations:** Despite the Higgs Mechanism accommodating massive gauge bosons and fermions in the SM, while maintaining local gauge invariance, it does not do this with the same mechanism for both, and thus, a Yukawa coupling term is introduced to solve this. The Spontaneous Symmetry Breaking for this coupling leads to the coupling of Higgs to fermions varying greatly, due to the dependence on fermion masses. For neutrinos to acquire a mass like other SM fermions, the Spontaneous Symmetry Breaking mechanism requires a right-handed neutrino to give them mass, however right-handed neutrino states are unobserved as of yet, despite it experimentally being proven that neutrinos possess mass [3] [4]

In particular, neutrinos play a vital role in subatomic physics. Neutrino masses, due to their small size, may relate to BSM physics and the existence of new, unexplored mass scales in particle physics, and provide us information about these mass scales, perhaps holding a clue to the fermion mass generation problem. Moreover, big bang nucleosynthesis depends on neutrino interactions and on the number of light neutrino species. Neutrinos of low mass in the order of 1 eV could constitute hot dark matter and may be pivotal to our understanding of galaxy formation. They may also play an important role in the Matter-Antimatter asymmetry in the universe, as the observed excess of baryons over anti-baryons may be related to heavy Majorana neutrino decay [5]

Thus, the study of neutrinos are crucial for our understanding of particle physics, and the energy reconstruction of particle showers at new detectors may help us understand and carry out neutrino physics at various neutrino-fluxes and energy scales.

Various reasons may exist for why physics Beyond the Standard Model (BSM) has not yet been observed. These include the possibilities that BSM physics may be too heavy to be produced at current LHC collisions, at collision energies of 13 TeV, or, that BSM physics may be too weakly coupled to the SM to be seen at the existing general-purpose experiments [6]. Thus arises a need for a new general-purpose experiment.

This current puzzle of the lack of observation of BSM physics and in particular, neutrino physics is hoped to be solved by the Search for Hidden Particles or SHiP experiment.

1.2 SHiP and SND@LHC

1.2.1 A Brief Introduction

SHiP is a proposed general purpose, fixed target experiment at the CERN SPS, designed to complement LHC experiments in the search for new BSM physics. Planned to be located downstream of a beam-dump facility, it is made of a high-density proton target, followed by a hadron stopper and muon shield. These filter all particles except a few muons, electrons, ν_τ and hidden long-lived particles, from secondary particle decays or proton interactions [7]. SHiP aims to explore very weakly interacting particles predicted by BSM physics and to study the properties of tau neutrinos. Primarily, Heavy Neutral Leptons predicted by the Neutrino Minimal Standard Model (ν MSM) [8] where right-handed counterparts N_1, N_2, N_3 are added to SM, will be looked into. Additional particles that will be studied include those in the Dark Sector, Supersymmetric Goldstone Bosons and Light Dark Matter (LDM). [9].

Its overall goal involves searching for two types of signatures predicted:

- **Decay Spectrometer (DS)** – Through the decay into visible particles in hidden sector spectrometer
- **Scattering Neutrino Detector (SND)** – By measuring neutrino scattering cross-sections

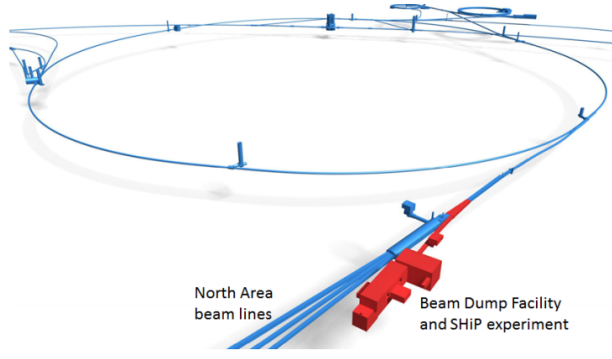


Figure 1: Proposed Beam-Dump Facility at SPS [10]

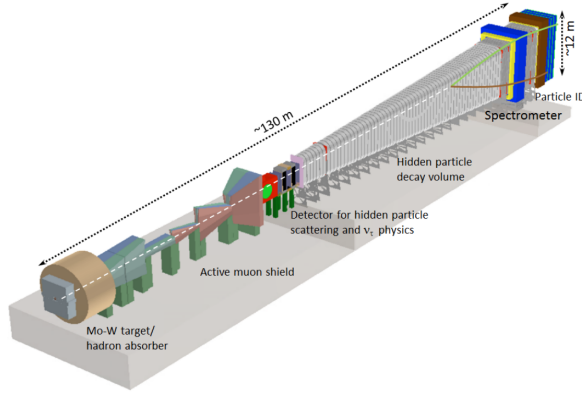


Figure 2: Overall SHiP experimental setup with SND and DS present [11]

The SHiP is planned to be located at the North Area of SPS. In the meantime, a parallel experiment using the same ideas and technologies as the SND detector of SHiP is planned to be built in conjunction with the pre-existing LHC, known as SND@LHC. It will be located at an LHC interaction point, later decided to be LHC-IP1 and will be placed off-axis with respect to it [12].

The primary focus of both the SND detector at SHiP and the SND@LHC experiment is the study of SM neutrinos. SND@LHC will collect data during Run 3 of LHC.

Preexisting experiments have studied neutrino interactions at energies below 350 GeV and energy regions between 350 GeV and 10 TeV remain unexplored. Over the past few decades, neutrino oscillation studies have been designed to be optimised over existing energy ranges upto 350 GeV, and no ν_e measurements have been made above 10 GeV. For ν_τ , the DONUT experiment reported 9 ν_τ events [13] and the OPERA experiment has reported 5 ν_τ events [14] [15] [16] [17] [18]. With so few events, it has proven challenging to precisely estimate the ν_τ and $\bar{\nu}_\tau$, as well as, due to lower energy ranges, cross-sections of ν_e and ν_μ at higher energies.

If an intense flux of neutrinos is used in combination with an intense proton beam, the neutrino-beam would produce a high flux of ν_e and ν_μ , as well as possibly ν_τ . At SHiP, 2×10^{20} protons from the SPS at CERN with 400 GeV energy will be used. Despite the neutrino energies still being limited to 350 GeV, the high flux of neutrinos may point to ν_τ further studies.

Conversely, although the SND@LHC will not possess such a high flux of neutrinos, it would extend energy scales of experiment upto a TeV scale, enabling novel neutrino cross-section studies.

The entirety of this thesis shall focus on the SND@LHC experiment, in particular the Pilot Run Detector.

1.2.2 SND@LHC structure

The SND@LHC consists of two overall parts, as shown in Figure 3, an Emulsion Spectrometer, and a Muon and Timing detector. Firstly, the Emulsion Spectrometer contains four emulsion brick walls (the emulsion target) interlaid with four Target Tracker (TT) planes, each made of Scintillating Fibres (SciFi). The brick walls are split into dimensions of 3x4 cells, with each cell containing an Emulsion Cloud Chamber (ECC). ECCs possess a high spatial resolution, enabling the identification of electrons through observation of particle showers in the brick [?]. This Emulsion Target is as illustrated in Figure 4.

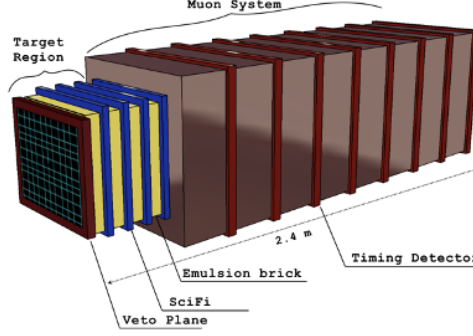


Figure 3: Proposed Neutrino Detector Structure [12]

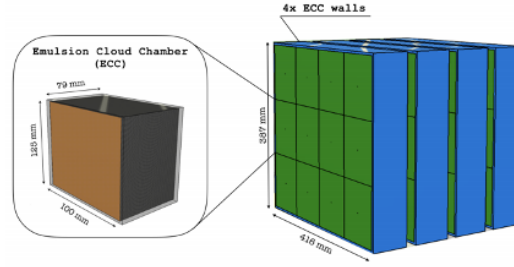


Figure 4: Emulsion target layout [12]

Emulsions, despite having very good spatial resolution, have the disadvantage of needing to be extracted from the detector and developed in order to obtain information about the particle shower and its origin. This makes them unsuitable for real-time analysis. Being able to reconstruct events in which a particle has initiated a particle shower in the SND@LHC is however of extreme importance, both for neutrino physics and to spot early hints of possible presence of DM. The SciFi trackers thus provides an opportunity to study particle showers in real-time, and disentangle piled up events using time and energy information [12].

The Scintillating Fibre Tracker acts as a scintillator which absorbs energy from an incoming particle and emits absorbed energy as photons. These photons are then transferred to Silicon PhotoMultipliers (SiPMs). SiPMs are arrays of Avalanche Photo-Diodes (APDs), also known as pixels. An individual channel consists of multiple pixels, and groups of these channels constitute a multi-channel SiPM. An incoming photon creates an avalanche of electrons in these pixels, and from this, a readout time window can be taken. The collected charge is then integrated to construct a signal cluster. These signal clusters can then be identified with their readout time window (timestamp) [19].

This is as illustrated in Figure 5.

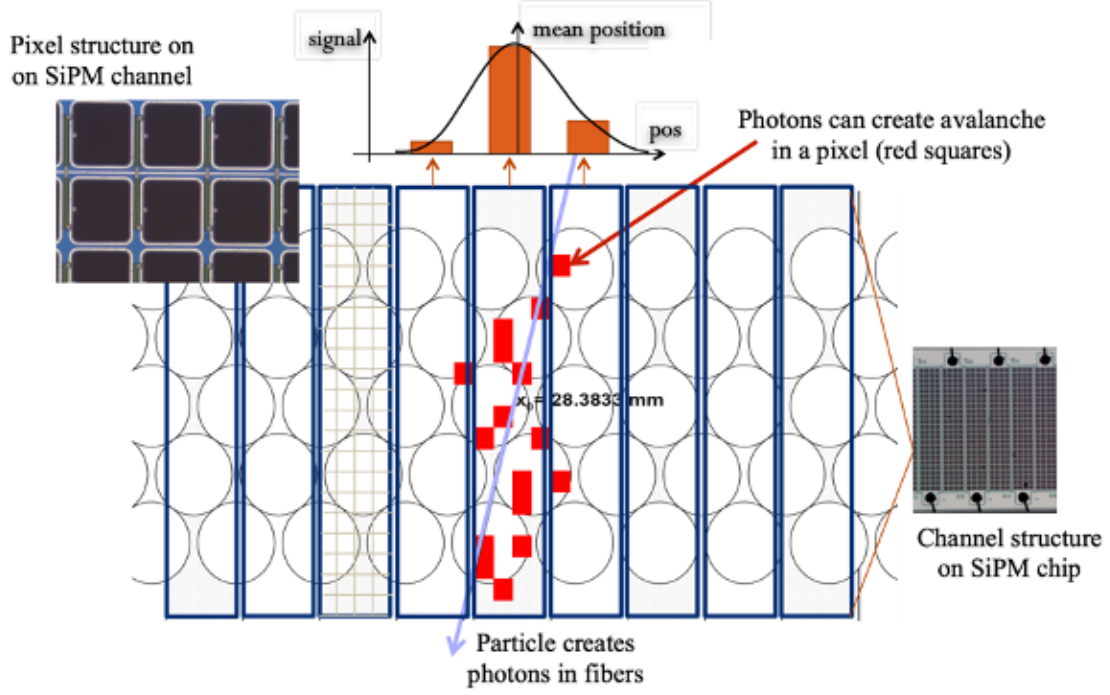


Figure 5: Signal Cluster Detection Schematic adopted from [19]

As Sampling Calorimeters contain arrays of thin counters separated by layers of absorbers, the SND@LHC also acts as a sampling calorimeter due to the SciFi Target Trackers serving as counters and emulsion bricks as absorbers.

An example of the physics taking place within the SND is the generation of Electromagnetic (EM) showers within the Emulsion Spectrometer.

As neutrinos of all the three species enter the SND@LHC, the detector is designed to be able to differentiate between the flavours. It also allows for the search for LDM through the study of scattering signatures of electrons and nuclei. Among these, ν_e are distinguished by the production of EM showers. An incoming ν_e scatters an electron present in the detector material, which, in the case of the SND@LHC, is the emulsion brick wall. This electron traverses through the material with a high energy, using two processes. In the first process it loses a large fraction of this energy to photons through bremsstrahlung and in the second, through the production of electrons and positrons by pair-production. Therefore, a single electron triggers an avalanche of photons, electrons and positrons, known as an electromagnetic shower [20]. This is shown in Figure 6.

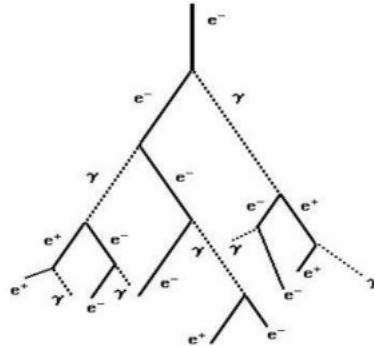


Figure 6: Electron-initiated EM shower [20]

This EM shower then produces a signal within the SciFi Target Tracker planes by triggering SiPMs, as shown previously in Figure 5.

Every individual SciFi plane can provide either x or y hits each and not a combination of the two. Ghosts

hits occur when the algorithm is unable to correlate each x-coordinate to a y-coordinate on each TT plane and vice versa, due to reconstructed hits that do not correspond to a real hit, due to various ‘false’ (x,y) coordinates of hits, as seen in Figure 7. This would lead to a significantly less accurate energy prediction due to these Ghost hits being absent from training and testing datasets, and higher computation processing times due to there being additional datapoints within an event present to consider when reconstructing the energy of an event.

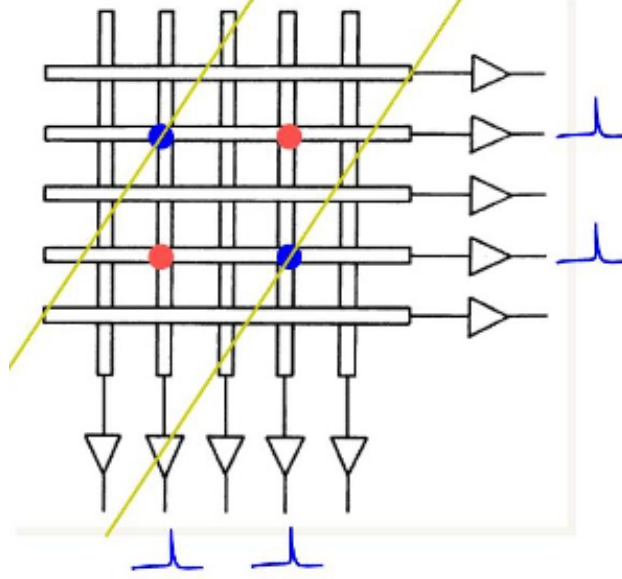


Figure 7: Representation of ghost hits in SciFi trackers. Blue hits are real hits and red hits are ghost hits [21].

EM shower reconstruction used to find the energy of the electron causing the shower, can provide valuable information about the scattering mechanism (energy of incoming neutrinos etc), other particles present (possible LDM signatures) and the overall detector’s operation as a sampling calorimeter.

1.3 Machine Learning, CNNs and Energy Reconstruction of Electromagnetic Showers

Traditionally, algorithms were developed based on domain knowledge, implemented within software and the resulting programs analysed and generated data, however, this becomes increasingly difficult and labour-intensive due to having to analyse complex datasets with various input variables. Rather than programming computers to analyse complex data and produce desired results, Machine Learning (ML) algorithms use large amounts of data to build models which can be applied to predict the behavior of new, previously unseen data. They can detect anomalies and/or generate simulated data with little human intervention [22].

Emulsions have to be extracted manually to observe events and thus is a time-consuming process despite possessing high resolution. To overcome this, the TT planes are made of Scintillating Fibres (SciFi), to enable EM shower reconstruction in real time and a future task of the SND@LHC will be to analyse the event and particle information obtained by the SciFi Target Tracker planes. This data will be used for the energy reconstruction of particle showers generated within the sampling calorimeter, and of the individual electrons propagating the shower.

Conventional baseline tracking algorithms, such as those used in OPERA, rely on known origins of the shower, however, for this algorithm for SND@LHC, no a priori knowledge of shower origin will be used [23]. Additionally, higher instantaneous luminosities and for detectors with coarse calorimeter segmentation, the overlap of calorimeter energy deposits from charged and uncharged particles leads to high occupancy, proving to be a challenge for particle energy reconstruction [24], and thus, a novel ML algorithm must be developed.

Energy reconstruction in High Energy Physics (HEP) converts detector readouts into kinematics of interactions. Initially, events and particle types are identified and the reconstructed energies are then assigned to these by an energy reconstruction process [25]. Thus, one of the challenges faced by the SND@LHC is a good energy estimation of the particle showers generated within the sampling calorimeter.

In this paper, deep learning and deep Convolutional Neural Network (CNN) methods are applied to energy

reconstruction, by feeding hit information per event as images to the CNN.

The Machine Learning (ML) algorithm carrying out this process was written originally by S. Shirobokov, found in [26], and further elaborated by P. de Bryas in [27] and [28], but initially, a good understanding of CNNs is necessary.

1.3.1 Convolutional Neural Networks – a brief introduction

The overall aim of a Convolutional Neural Network is for the network to differentiate between the images fed into it and discern the unique features that make it fall within a certain class. For a CNN, an image is seen as an array of pixel values. For a coloured image of dimensions $n_x \times n_y$, it computes an $n_x \times n_y \times 3$ array of numbers, the 3 referring to RGB values. Therefore, the overall image comprises of three images, one for each R, G and B channel. Conversely, a grayscale image only has a singular channel and so, a grayscale image of dimension $n_x \times n_y$ would have a representative array of shape $n_x \times n_y \times 1$.

The numbers within these representative arrays are assigned a value between 0 to 255, to indicate brightness intensities. These pixel inputs are the inputs available to the CNN. This representative array is fed into the network, and the output is the probability of the image being within a certain class. A CNN performs image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers, as will be explained below.

CNNs are important in fields including image and pattern recognition, speech recognition, natural language processing etc. The improved network structures of CNNs lead to savings in memory requirements and computation complexity requirements and, at the same time, give better performance for applications [29].

In summary, CNNs are fed an image as its input, and it passes this image through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, to get an output. This output can be a single class, or the probability of multiple classes which best classify the image.

It is crucial to understand how CNNs and certain aspects within them function, as shown below:

1.3.1.1 Convolutional Layer The first layer of a CNN is usually a convolutional Layer. The input to this conv layer is an array of the shape of the image. The conv layer consists of a kernel, also known as a filter or neuron, which slides over all areas of the input image array, and the specific region it covers at a time is known as the receptive field. The kernel itself is an array of numbers, of dimensions for example 5×5 . It is important to note that the depth of this filter has to be the same as the depth of the input (this ensure consistency in calculations), so the dimensions of this filter is $5 \times 5 \times 3$. The numbers within a kernel array are known as weights or parameters, and in the first conv layer, are initialized randomly.

As the kernel convolves around the input image, it multiplies values of the kernel weights with the pixel values in the image. These mathematical products are then summed to produce a single number. This is then repeated over the entire input array by moving the kernel by a certain amount each time, and this certain amount is known as the stride of the kernel. Every unique position on the input produces a number, and after convolving the kernel over the entire input, a singular, smaller array of numbers remains, known as an activation map or feature map (as seen in Figure 8).

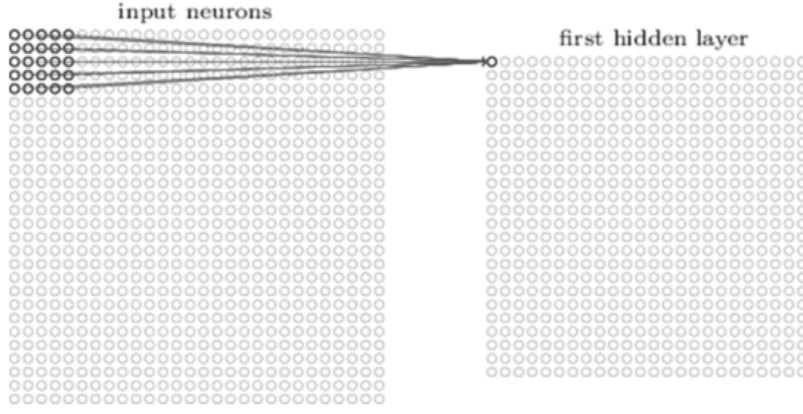


Figure 8: Visualisation of 5×5 filter convolving around input volume and producing activation map. [30]

Additionally, by using more kernels, more activation maps are produced, and thus, the output volume contains higher spatial dimensions. Therefore, more filters lead to better preservation of spatial dimensions and thus more information is retained.

Every individual kernel/filter can be looked at as feature identifiers. Each kernel identifies one feature of an image, such as edges, curves, colours etc. A kernel for detecting a curvature may have a structure such as that seen in Figure 9.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Figure 9: Pixel representation of a kernel for detecting curvature [31]



Figure 10: Visualisation of curvature kernel [31]

When the weights within the curvature kernel are multiplied by the original pixel values within a receptive field of an original image, an example of the process taking place is as shown below in Figure 11.

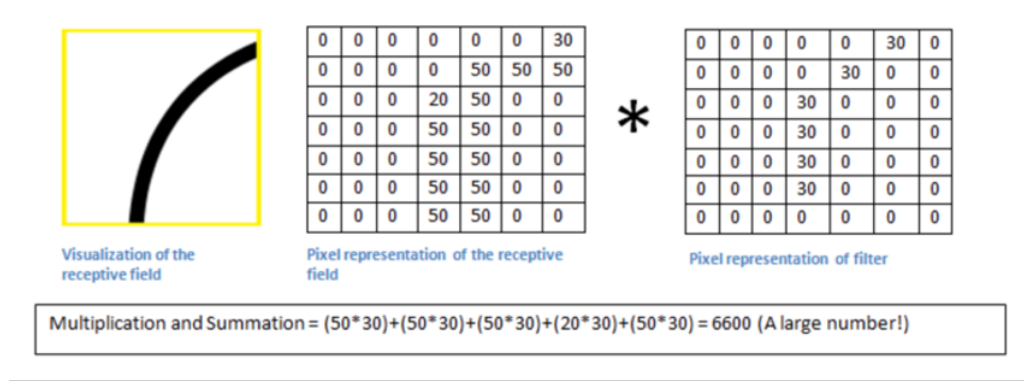


Figure 11: Feature identification in a receptive field [31]

Thus, the activation map contains information about which features are retained in which part of the image. As the curvature filter parses through the entire image, it identifies areas with curvature and those values within the activation map will be high. The CNN then repeats this process for various kernels corresponding to different features, and so the greater the number of filters, the greater the depth of the activation map, and more information about the input image is identified and retained.

Deeper within the network, the activation maps represent higher, more complex features. By the end of the Neural Network, certain filters are able to identify complex objects within the input image, such as handwriting, compound shapes and so on.

1.3.1.2 Padding Padding involves the addition of a string of 0s or 1s to the ends of the input volume. As a higher number of convolutional layers are applied, valuable information is lost about the original input volume as the size of the input volume decreases quickly. Padding is applied to the convolutional layer in order to extract low level features while minimizing the loss of information.

1.3.1.3 ReLu (Rectified Linear Unit) Layer The purpose of a ReLu Layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations). The advantages are that the network is able to train faster, without significant differences to the accuracy. ReLu layers convert all negative activations, which may stem from the initialization of negative weights, to 0, increasing nonlinearity.

1.3.1.4 Pooling Layers Pooling layers, in particular Maxpooling, usually use a filter and a stride of the same length and applies this filter to the input volume, outputting the maximum number in every region it parses. The basic reasoning for applying a pooling layer is that once a feature is identified within an original input volume, it will have a high activation value, and its exact location within the image is not as crucial information as its location with respect to other features. As this layer reduces spatial dimension, it serves to reduce computation time and helps to control/minimize the overfitting problem. The overfitting problem dictates that the model cannot perform well on validation and test datasets due to it over-learning the training dataset and its noise, using its ability to generalize [32].

1.3.1.5 Dropout layers Dropout layers drop certain activations in the layer by setting them to zero. This helps ensure that the network is still able to predict with relatively good accuracy despite activations being dropped out, and this further helps reduce the aforementioned overfitting problem.

1.3.1.6 Batch Normalisation Batch normalization helps with adjusting and scaling activations. In order to speed up learning, if certain activation values range from 1 to 1000 and another from 0 to 1, batch normalization reduces the shift of values within the hidden layers. An example of this is when a network designed to classify an image into a cat or not a cat is trained only with black cats, and then tested with coloured cat images, it will not perform well. This is most likely due to the differences in scales of activation values for both the coloured and black cats. If the algorithm learned some X to Y mapping, and if the scale of distribution of X changes, then the learning algorithm may need to be retrained by trying to scale the distribution of X to fit the distribution of Y [33].

1.3.1.7 Fully Connected Layer A Fully Connected Layer is the final layer in which the input volume from the previous layers is taken in and the output is an N-dimensional vector containing the number of classes the network must choose from, to make its prediction. It looks at the output of the previous layer (which should represent the activation maps of high-level features) and determines which features most correlate to a particular class or value.

1.3.1.8 Training Process The overall training process in the neural network works through the mechanism of back propagation. This is done in four steps: the forward pass, the loss function, the backward pass and the weight update. The forward pass is when the input image into the network is passed through all the layers within it. Usually, the weights in the first forward pass are randomly initialized and have a consistent value, as they do not favour any particular number, and due to this, the network is unable to make any fruitful conclusions and output a prediction. This is then passed onto the loss function, where the training label (true value) is compared to the predicted label. The aim of this is to minimize the amount of loss, the difference between the actual label and predicted label. This is an optimization operation, and if the variable L is the loss of the network, to investigate which weights contributed the most to the loss, the derivative $\frac{dL}{dW}$ is computed by initiating a backward pass. The final step in the backpropagation mechanism is the weight update. The weights are updated in a way which attempts to then minimize the computed gradient.

Despite the current ML Algorithm providing a sound basis for the eventual implementation into the SND, there remain various limitations and omissions of real-life phenomena within it. This thesis aims to address these limitations and overcome a few of them in order to refine the algorithm to enable implementation into the SND.

1.4 Overall Aims and Hypotheses

In this research a dataset generated using FairSHiP will be utilised. FairSHiP is a framework used for the reconstruction for SHiP, which in turn utilizes ROOT and FairROOT. FairROOT [34], [35] is used for data analysis, object-oriented simulation and reconstruction. Despite the SND@LHC containing both EM and hadronic showers, the dataset used in this thesis contained only EM showers. Additionally, the simulated dataset corresponds to data simulated with a complete detector description with the layout of the SND@LHC Pilot Run Detector [27].

The overall goals this thesis attempts to carry out are as follows:

Firstly, the energy bias between the true and predicted energies will be minimised by experimenting with normalised and unnormalised loss functions of L1Loss and MSELoss, and the effects of training the CNN with a wider energy range than the testing energy range will be observed. Secondly, ghost hits will be implemented into the current dataset to observe the effect on the energy bias. This is crucial to the eventual implementation of this algorithm into the SND@LHC pilot detector, as previous research, done by [27] used unrealistic (x,y) combinations of hits, thus eliminating ghost hits altogether, however, this will not be the case for real-life scenario, which shall include ghost hits. The current SciFi tracker 'front-facing' images will be transformed into side-view images of the SciFi Tracker planes. The CNN will be optimised to attempt to bypass the problem of ghost hits altogether.

It is postulated that the network will predict electron energies with higher accuracies when using MSELoss than L1Loss. This is due to MSELoss possessing a squared term, resulting in predictions farther from actual values being penalised more heavily in comparison to less deviated predictions. An additional hypothesis is that utilising the normalised loss function will lead to better energy predictions by the network, when compared to its unnormalised counterpart. This is as the loss is the relative error between the predicted and true energies, leading to a larger relative error for higher energies. By normalising the loss function by the true energy, the losses are thus scaled to eliminate biases the network may have towards or against high or low electron energies due to larger absolute values. Additionally, by training the network with a wider range of energies than the testing energy range, a decrease in bias would indicate that energy reconstruction at border energy regions leads to worse energy reconstruction due it being too constrained at these regions.

For the ghost hits implementation, it is hypothesised that the network's energy reconstruction accuracy would worsen, due to the presence of many additional data points. Finally, it is hypothesised that the bias would improve for the side view, xz and yz images, post-optimisation of the CNN, used as input for the algorithm. This is due to the higher resolution in the x and y directions increasing the energy reconstruction accuracy [36].

The aim of this thesis is to alter the existing proof-of-concept algorithm and take the first step towards its practical use in realistic scenarios. Thus, for the sake of this study, the physical processes behind the data generated are not as crucial as the data being analysed itself and the usage of this data for real-time analysis in the SND@LHC detector.

1.5 Importance of the Research

This research attempts to combine particle tracking with measuring energy data for the EM showers, by applying a ML algorithm to tracker data to generate calorimetric results. This is previously unmapped territory as this is the first time energy reconstruction with a sampling calorimeter has been using ML has been attempted, as far as we know. This leads to a better understanding of the nature of the particles travelling through the emulsion bricks. The real-time calorimetric data also aids in differentiating between signatures for various particles/events.

The study of EM showers brings great insight into the development of SHiP and SND@LHC, neutrino background rates from LHC interaction points and detecting the possible presence of dark matter. With eliminated ghost hits and increased precision at higher energies, this algorithm can be commissioned, tested on real data, and eventually implemented into the SND@LHC Pilot Run. The SND@LHC also serves as an important test detector for SHiP and so implementing this algorithm into SND@LHC would provide important information for the future construction of the SHiP detector.

2 Methodology

2.1 Preprocessing Raw Data

The overall aim of this algorithm is the energy reconstruction of electrons passing through the Target Tracker planes. The SND@LHC Pilot Run detector will have four such TT planes present, with emulsion bricks interleaved between them. Using FairShip, these simulated events were collected in a ROOT file. The first challenge of this algorithm was converting this raw hit and event data into readable input images for the CNN to use. This was done by preprocessing the simulated raw data based on the capabilities of the SciFi Trackers and for appropriate track reconstruction of the EM shower.

The algorithm was altered based on the specific objectives to be achieved (mentioned below), and then trained with these changes using GPUs from EPFL and Nikhef. The dataset generated by FairShip is Preprocessed and, using Pytorch, converted into the tensor images which are eventually fed into the CNN to obtain energy predictions. Other python packages used for the development of the algorithm include sklearn for machine learning methods and pickle for generating .pkl files containing event information.

The unreduced geometry was initially set to resemble the real SND@LHC Pilot run detector. The number of TT planes, n_{planes} was chosen to be 4 as will be the case at the SND@LHC Pilot Run Detector, and dimensions of each plane were $d_x = 52.0$ cm, $d_y = 42.0$ cm and the d_z width of each plane was approximately 5.0 cm per plane. The x and y dimensions were set to be higher than the true detector dimensions of the SND@LHC Pilot Run of 42×39.9 cm² [12], to ensure all hits are incident on the planes. The distance between each Target Tracker plane was set to be 10.0 cm, and so this was the width of the emulsion layers within which the EM showers were produced and propagated. In the existing algorithm, the energy range of the fired electrons was set to 200 to 400 GeV for both the testing and training range

The number of events to be preprocessed was chosen to be 180,000 and the number of events per chunk chosen to contain 5000 events each. Firstly, the raw data was preprocessed to remove all hits caused by uncharged particles. This was due to the inability of the SciFi tracker to detect or analyse such particles. Additionally, the SciFi tracker cannot detect hits by particles possessing energies lower than that of 1e-7 GeV and so this is the minimum energy threshold required to filter them out. However, for this thesis, the energy threshold was set to a higher value of 0.0001 GeV, due to the lower resolution of input images leading to an oversaturation of hits within them. Subsequently, particles with negative momentum values were also discarded, due to them being back scattered and thus not propagating through the Emulsion Spectrometer of the detector. Finally, in order to ensure appropriate shower reconstruction based on a sufficient number of hits, events with less than 5 hits were not included in the preprocessed data. Overall, the filtered raw data in the root files were then stored as Pandas DataFrames, making them easier to utilize within python.

Using these pickle files, various individual events were displayed in order to observe their overall hit patterns, and compute the tensor shape of the input image. The conversion of the Pandas Dataframes to an image was done by the function `digitize_signal` (see Figure A.1. in Appendix)

As the GPU computing power was not sufficient to utilize the unreduced dimensions of the images, the images were required to be downsized. This downsizing was carried out by selecting a window within which 99% of the hits were contained. This was done because reducing the overall resolution per unit area would lead to a loss of information and hit separation, as a large number of hits in the EM shower are concentrated with minimal separation from each other. Therefore, the reduced dimensions of the TT plane were computed to include 99% of the hits, and these new dimensions were then stored in a new parameters file, while retaining the same geometry of 4 planes and d_z width used previously.

These reduced dimensions were then utilized to produce reduced pickle files, and the events containing 99% of hits were then used to train the CNN.

2.2 Original CNN Architecture

The original CNN structure used to carry out energy reconstruction involved constructing the network out of blocks (see Figure A.2. in Appendix). These blocks were then used with various input arguments in order to construct varying CNN architectures. This was advantageous as it enabled easier alteration of the CNN structure and tuning of hyperparameters. The primary block structures, which were a Convolutional Block and Fully Connected Block were as those seen below in Figure 12.

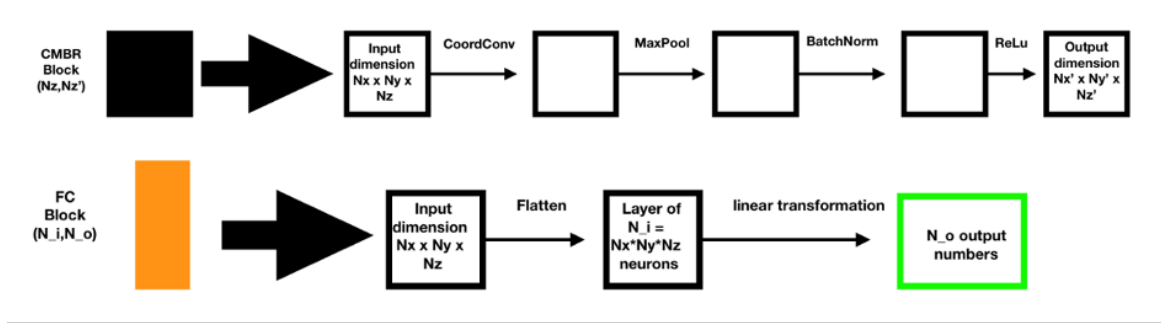


Figure 12: Different block structures used in CNN [27]

The CMBR (Convolutional, MaxPool, Batch Normalisation, Rectified Linear Unit) block, as the name suggests, contains a convolutional layer, along with a MaxPooling layer. Following this, a Batch Normalisation layer was implemented into the block, and finally, a ReLu layer was added, to complete the structure of the hidden layer. The block required two parameters to be specified: the N_z dimension of the input (the number of 2D images) and the output parameter which is the desired N_z dimension of the output.

The Fully Connected block then was responsible for using inputs from the last CMBR block and outputting a singular prediction value. This block also required two parameters to be specified: the total number of output pixels ($N_x \times N_y \times N_z$) from the last CMBR block, and the other parameter is the number of predicted outputs, which will remain 1 through the entirety of this thesis as only a singular energy prediction value for each event is aimed for. In order to implement linearity and obtain a singular value from a 2D input, the Flatten module is applied, to convert the 2D input into a 1D list of pixels.

The network had 4 CMBR blocks overall, with the first block using the number of images in the input as the input parameter, which was 4 due to the number of images produced by the `digitize_signal` function, yielding one image per TT plane. The overall architecture of the network was as shown in the classes `SNDNet` and `Block` in Figure A.2. in the Appendix.

2.3 Minimising bias and testing Border Effects

Within the dataset generated, all energy values were seen to range between 200-400 GeV. Using the original CNN structure and preprocessed data, bias analysis was carried out by initially testing with different loss functions to minimize the overall bias between the true and the predicted values of the energy output by the network. Additionally, better performance of energy reconstruction is also given by a lower standard deviation between the predicted and true energy labels. This would indicate a lower spread in the energy difference between predicted and true energy values.

The measure of the estimation accuracy of the network used in this thesis was the Relative Error (RE), the formula for which is in Equation (1) below:

$$RE = (A - E)/A \quad (1)$$

where A is the True value and E is the predicted value

The Mean Relative Error (MRE) for n events is as seen in Equation (2), and thus the bias was the MRE of the predicted and true energy values for all the test events.

$$RE = (RE_1 + + RE_n)/n \quad (2)$$

where RE_1, RE_2 etc are the relative errors of the 1st, 2nd event etc.

The two loss functions tested in this paper were in-built PyTorch functions, L1Loss and Mean-Square-Error Loss or MSELoss.

L1Loss is a measure of the Mean Absolute Error (MAE), with a formula as shown in Equation (3)

$$loss(x, y) = |x - y| \quad (3)$$

where x is the actual value and y is the predicted value.

L1Loss measures the numerical distance between the actual and predicted value. The absolute value is considered, as without eliminating negative values, the negative errors will cancel positive errors.

MSELoss is a measure of the Mean Squared Error (MSE), with a formula as shown in Equation (4) below:

$$loss(x, y) = (x - y)^2 \quad (4)$$

where x is the actual value and y is the predicted value.

MSELoss, like L1Loss, is applicable to regression problems such as that in this thesis and so both were chosen for trial.

Additionally, the network was trained and tested by normalizing the loss functions, to observe the impact of normalisation of the loss function on the bias.

Finally, as a part of the Border Effects studies, the testing energy range was decreased to a range of 220-380 GeV, with a 20% reduced margin on each extreme. This was done to examine whether training with a wider energy range than that within the test dataset would yield better prediction results for the test data.

2.4 The Ghost Hits Problem

A primary aim of this thesis was to address the ghost hits problem in the algorithm, as these shall be observed at the real SND@LHC detector. There were two ways of tackling this: by implementing the ghost hits into the current algorithm to observe how well/poorly the network predicts energy values and the second would be to use side-view, xz and yz TT plane images of the TT planes, instead of the current ‘front-on’ images to eliminate the problem altogether, and to optimize the CNN for such images.

2.4.1 Implementing Ghost Hits into current Algorithm

The current hit information was stored in Pandas DataFrames as a long list of entries with each row corresponding to a hit, and the columns of the data frame corresponding to various information about that hit on the TT plane, such as the momentum in the x,y,z directions, the (x,y,z) coordinates of the hits etc.

In order to implement the ghost hits into the original code, during the production of the reduced pickle files during step 6 of Preprocessing, containing the Pandas DataFrame of the TT responses, the x and y hits were first each grouped according to the TT plane they were found on. This was done by grouping the z coordinates of each hit into the four different TT planes, based on the width and starting position of the plane. Then, every x coordinate of a hit within a plane was matched to every y coordinate within the same plane, and these new constructed ghost hits were appended to the original data frame, containing only x,y and z coordinate information.

This newly appended dataframe was then used to produce the reduced pickle files, which were then eventually be used for training.

2.4.2 Side-view, xz and yz images of TT planes and CNN optimisation

By altering the original `digitize_signal` function, the four TT images containing (x,y) hit information were then transformed to produce two images of (x,z) and (y,z) responses respectively. These images thus produced 4 lines in the z-direction with various hits in the x and y directions for each of the images. As empty pixels with a value 0 were considered redundant information, the image was compressed to be 4 pixels wide in the z direction, with each TT plane a single pixel wide, stacked against each other. The resolution in the x and y directions in the 2 images respectively were augmented to 2100 pixels, which is approximately a fourth of the true x and y resolution found in the real SND@LHC pilot detector of $50 \mu m$ (8400 pixels) [12], and this could be done due to the lowered resolution in the z direction of the image, enabling a higher image resolution to be used due to the compression.

The altered `digitize_signal` function to produce the side-view images of the TT planes is as shown in the Figure A.3. in the Appendix:

These new images were then used to train the network to judge the efficacy of energy prediction for side-view, xz and yz images of TT planes. The CNN structure and hyperparameters were adjusted in order to minimize the bias, and the CNN architecture was tuned and altered to find an optimal structure to train future side-view images in the real detector. The altered network structures are explained in the following paragraphs.

To train these new input images, the CNN was altered to minimize bias and maximise efficiency. Initially, these new images were trained with the original CNN Block architecture of 4 Blocks. Furthermore, due to the non-square nature of the new compressed image, the kernel size was changed to (2,4), to increase the number of features included in the x and y direction per image respectively. The module `MaxPool2d` was changed to (1,4) to avoid pooling in the already compressed z-direction. This CNN, slightly altered from the original, architecture can be seen in the Figure A.4. in the Appendix.

To observe the bias at various epochs, the network was trained with 10000 and 30000 events. The CNN architecture was subsequently simplified and trained with the same number of events, to compare performance and the accuracy of the prediction energy values with respect to the true energy labels within the data.

The simplification and alteration of the CNN structure involved reducing the number of CMBR blocks from the original 4 blocks to 3 blocks and reducing the number of input and output features to 32 for each block. The new CNN architecture used is as seen in the new SNDNet and Block classes in Figure A.5. in the Appendix

3 Results and Discussion

3.1 Minimising bias and testing Border Effects

One of the two main aims of this thesis were to minimize the bias between predicted and expected values by experimenting with differing loss functions and testing with a smaller energy range, with a 20% lower energy margin than the training energy range on each extreme. The results obtained for this are as follows.

3.1.1 Minimising Bias with differing Loss Functions

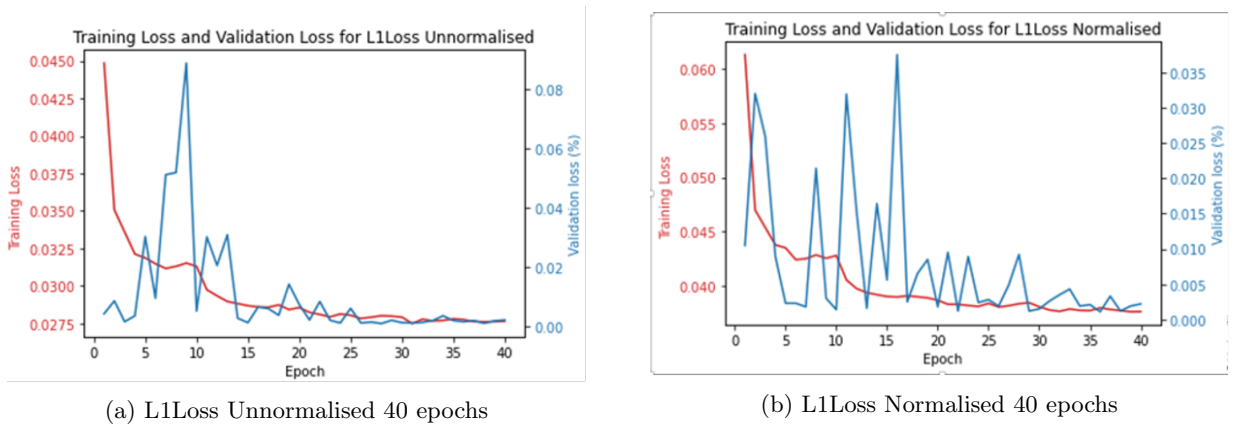


Figure 13: Training and Validation Loss for L1Loss

Figures 13a and 13b depict the evolution of the training loss and validation loss for L1Loss with respect to the number of epochs the network propagates over and, as expected for both the normalised and unnormalised functions, the training loss decreases over the number of epochs, indicating that the algorithm improves the

accuracy with which it predicts the energy of the electrons of the training sample. Similarly, the validation loss, which corresponds to the network's ability to reconstruct the electron energy on unseen data also decreases as the number of epochs progress for both the unnormalised and normalised functions.

In Figure 13a, the validation loss fluctuates and oscillates before eventually stabilizing around epoch 27, however, it is interesting to note that beyond epoch 30, despite maintaining its stable nature, the validation loss seems to still fluctuate slightly with seemingly slight increases in value at epochs 35 and 40. In Figure 13b, the oscillation in value is far more distinct and consistent as the epochs progress, and although the validation loss decreases, it never completely stabilizes within the 40 epochs. It is also seen that there is a correlation between the high bump visible in Figure 13a at epoch 10 for the validation loss, and the bump in the training loss at approximately the same epoch. The validation loss fluctuations lessen beyond this value, indicating that maybe a new feature was learnt and so the energy reconstruction of the algorithm improved in accuracy and performed better on the unseen data.

Conversely, in Figure 13b, there is no clear correlation between the bumps in the training loss curve and validation loss curve, perhaps indicating that no new features were learnt and applied to the validation set, so although the network performs better on the training data, it lacks the ability to generalize. This is known as over-fitting and it is suspected that for L1Loss normalised, the network performs too well on the training data while not generalizing its learning [37].

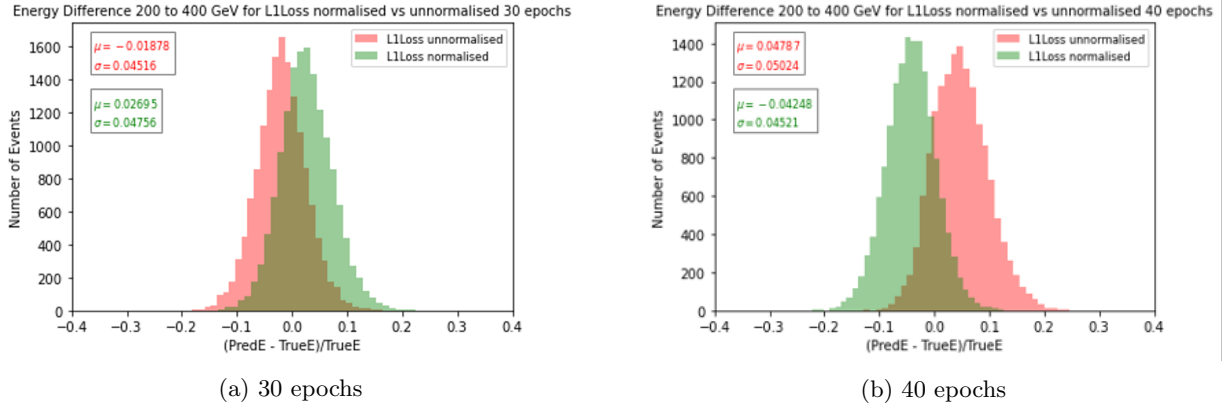


Figure 14: Energy difference when tested and trained with an energy range of 200 to 400 GeV for L1Loss normalised and unnormalised

Figure 14 depicts comparison graphs of the bias between predicted and expected electron energy values for L1Loss normalised and unnormalised. In general, the biases for both the L1Loss functions are lower at 30 epochs than at 40 epochs. This may be, as previously mentioned, due to an increase in the validation loss at epoch 40, when compared to 30. As for both L1Loss normalised and unnormalised the training loss seems to stabilise at epoch 25-30, at epoch 40 the network's ability to generalise may deteriorate, a phenomena known as overtraining. Thus, it was found that for the L1Loss function in combination with the original CNN structure, the optimum number of epochs for training to avoid the overtraining problem is 30 epochs.

The standard deviations for the L1Loss unnormalised and normalised curves at 30 epochs was found to be 4.51% and 4.75% respectively. This signifies that there is approximately a 5% spread of the relative error between predicted and true energy label from the mean. It was also found that L1Loss unnormalised performed better at electron energy reconstruction than L1Loss normalised. This can be seen as the bias between the predicted and true energy values for L1Loss unnormalised was found to be approximately 1.88%, whereas for L1Loss normalised, the bias was found to be 2.69%.

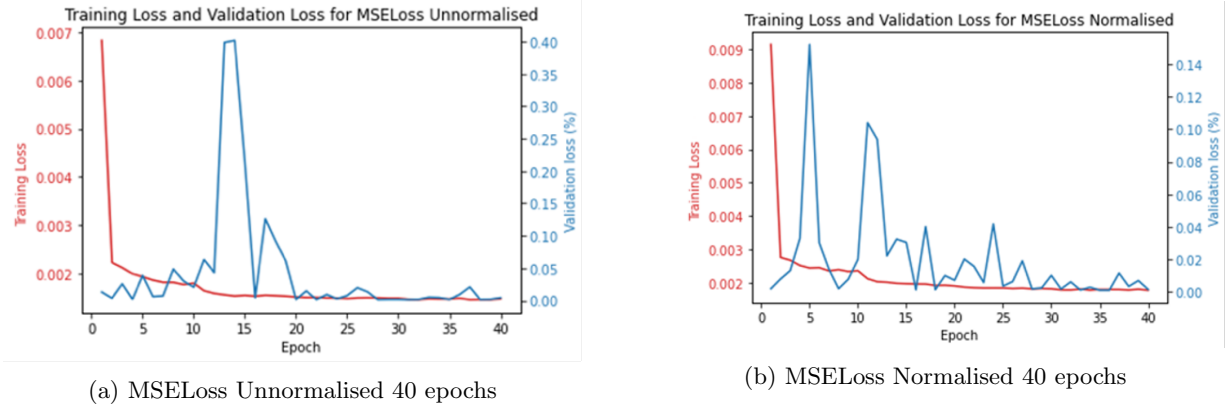


Figure 15: Training and Validation Loss for MSELoss

In Figure 15 the graphs for the training and validation loss for MSELoss unnormalised and normalised are seen. As was observed with the L1Loss function, for both 15a and 15b, the training loss and validation loss decrease over the number of epochs, indicating improvement in energy reconstruction. One of the most prominent aspects observed was the steep gradient at which the training loss decreases between the first and second epochs in both 15a and 15b. This signifies that at the first two epochs, the network using MSELoss function undergoes the most learning and realizes more features. It is after this second epoch that the training loss decreases with a relatively lower gradient, indicating a more gradual learning process on the training data until epoch 10. Beyond epoch 10 the training loss begins stabilising, for 15a it completely stabilises around epoch 20, whereas for 15b this takes place around epoch 30.

The validation loss observed for 15a is noteworthy to mention. It initiates at a relatively lower value, and abruptly spikes to a value of approximately 0.4% at epoch 14 and 0.15% at epoch 17. At epoch 14, validation loss takes a value approximately 200 times, and at epoch 17 the validation loss is approximately 5 times greater than the training loss at the corresponding epochs. Beyond epoch 20, it decreases further and stabilizes between epochs 27 and 35, after which the validation loss increases and loses its stable nature slightly.

For 15b the validation loss is far more volatile and despite decreasing overall, it does not seem to stabilise completely within the 40 epochs. It is also interesting to note that for this figure, the validation loss at 30 epochs is higher than that of 40 epochs.

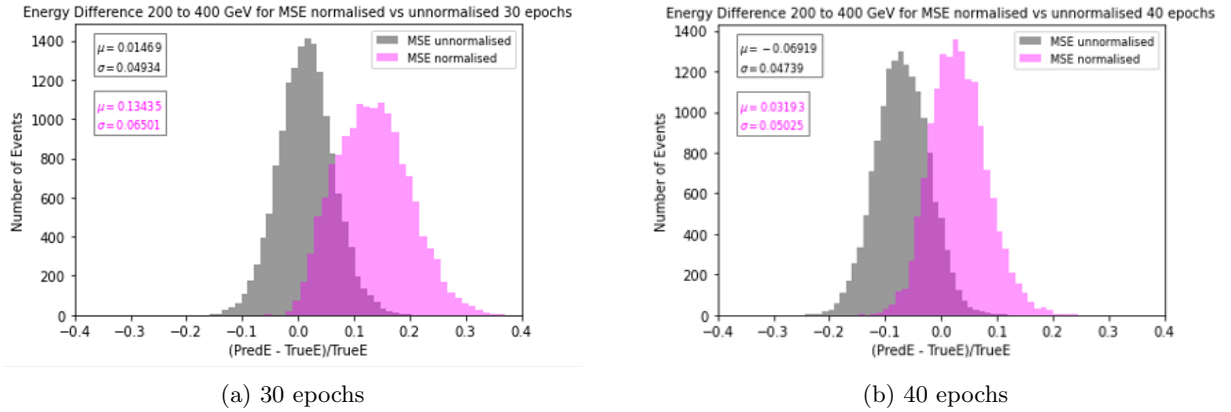


Figure 16: Energy difference when tested and trained with an energy range of 200 to 400 GeV for MSELoss normalised and unnormalised

In Figure 17 it can be observed that the patterns in difference in biases for MSELoss between the four curves are slightly different to those for L1Loss. Firstly, like both L1Loss normalised and unnormalised, generally MSELoss unnormalised seems to perform better at 30 epochs than 40 epochs, as its bias at 30 epochs is 1.46% as opposed to the bias at 40 epochs, seen to be 6.91%. Conversely, MSELoss normalised seems to perform better at 40 epochs, with a bias of 3.19%. It performs relatively badly on the test dataset at epoch 30, with a bias of 13.4% observed between the true and predicted electron energy values. This behaviour may be explained due to its validation loss at 30 epochs being higher than that of 40 epochs, perhaps due to the network still being in the process of learning new features to apply to the unseen data at 30 epochs.

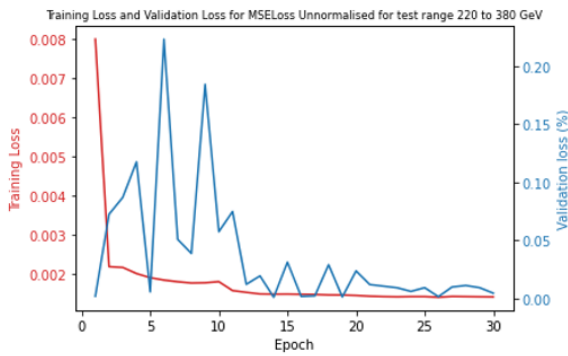
Overall, MSELoss unnormalised at 30 epochs was found to be the loss function with the lowest bias between the predicted and true energy values, and this loss function was subsequently used in all following bias analysis in this thesis. The spread for MSELoss unnormalised at 30 epochs is seen to be approximately 4.93%, whereas for MSELoss normalised at 30 epochs this spread is 6.50%, indicating that there is greater variation in the prediction values from the mean for the normalised loss function.

One of the most noticeable trends among the loss curves for the four varying loss functions was the general volatility of the validation loss curves observed. A possible reason for random unexplained spikes could be that they are an unavoidable consequence of the mini-batch gradient descent within the Adam optimizer used in the current network. Certain mini-batches may contain less-than-desirable data for optimization, and may not be representative of entire datasets, inducing sudden spikes in the loss, in our case the validation loss [38].

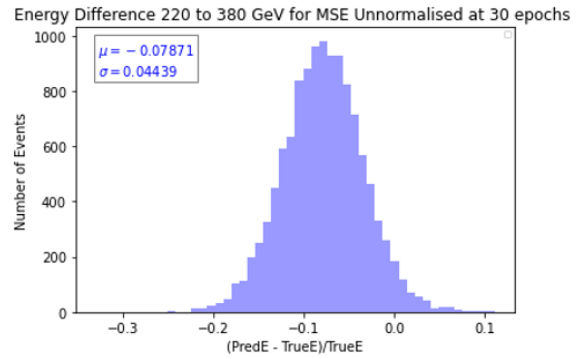
Possible ways to alleviate this problem may be by increasing the batch size from 150 to 300 in order to minimize the effects of smaller, individual mini-batches, or by increasing the learning rate to prevent the validation loss from being trapped within a local minima [38].

Furthermore, for both MSELoss and L1Loss at 30 epochs, the unnormalised loss function illustrated a better ability to make accurate predictions. Although the reasoning for this is not apparent, a possible reason may be that normalising the loss function decreases the loss value. This rings especially true for the MSELoss normalised. The squared term within the MSELoss function squares the normalised, lower loss values. Subsequently as the error value is less, the Convolutional Neural Network is adjusted less, indicating a smaller step-size between each back-propagation, and thus a lower learning rate, due to the learning rate being adaptive. This may cause the loss function to be trapped within a suboptimal solution, such as a local minima [38].

3.1.2 Testing Border Effects



(a) Training and Validation Loss for MSELoss Unnormalised



(b) Energy difference for MSE unnormalised

Figure 17: Energy difference when tested with a smaller energy range of 220 to 380 GeV for 30 epochs

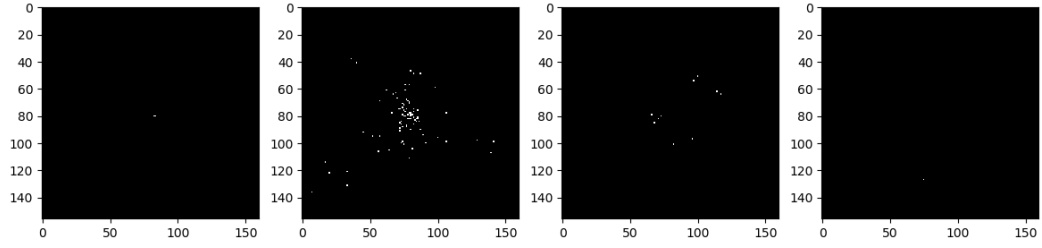
As seen in 17b, the bias between the predicted and expected electron energy values was found to be approximately 7.87%, and the spread of the energy difference values from the mean (bias) is approximately 4.44%.

This would mean that the overall range of energy differences between predicted and expected values deviates from the bias by 4.44%.

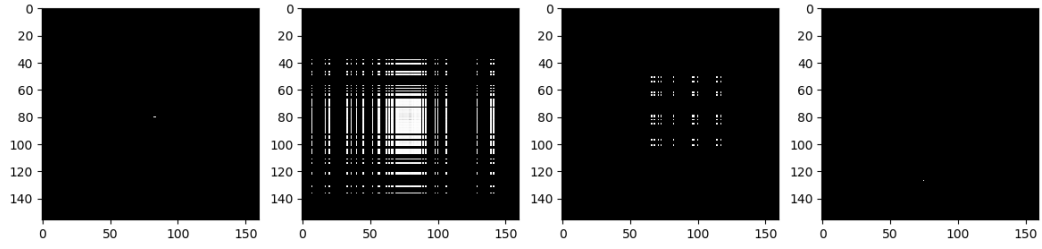
This illustrates that border effects are negligible and possibly do not impact the overall bias detrimentally. If the bias had significantly improved for a test range of 220 to 380 GeV to a lower value than that seen for the full testing energy range of 200 to 400 GeV, it would have been possible to conclude that the majority of the bias problem lay in the 20% border region ignored in the reduced test range. However, this was not found to be the case, and there was no reduction in the bias for a smaller test range. Thus, border effects do not seem to have a significant impact.

3.2 Tackling Ghost Hits Problem

3.2.1 Ghost Hits Implementation

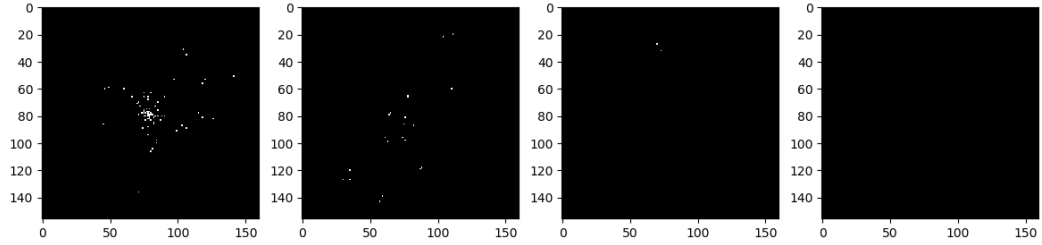


(a) No ghost hits present

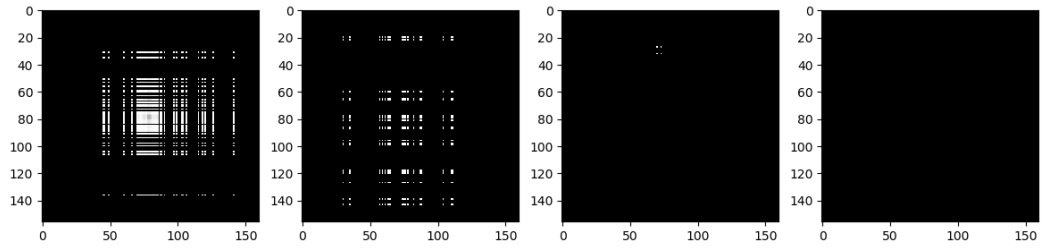


(b) Ghost hits implemented

Figure 18: Event 3 with no ghost hits (only first 100 hits) and ghost hits



(a) No ghost hits present



(b) Ghost hits implemented

Figure 19: Event 50 with no ghost hits (only first 100 hits) and ghost hits

In Figures 18 and 19, two random events were selected, which in our case was event 3 and event 50, and these were chosen to depict the output of the implementation of ghost hits in the algorithm. Only the first 100 hits were taken considered in order to visualize the hits and corresponding ghost hits with better clarity. The white lines represent pixels upon which ghost hits have been incident due to the detection of real hits. As can

be seen, for planes with a higher density of ghost hits incident on the barycentre, there is a higher density of white lines present. Interspersed within these white lines are dark bands, which appear to be x and y fibres upon which no hits have been incident. This is not unlikely as because only the first 100 hits from an event were chosen to be visualised, it is possible that not all pixels on the SciFi Tracker were triggered.

Despite seeming to be effective in applying ghost hits to the current TT response information, various problems were faced in the implementation of the code to the algorithm. Primarily, appending every combination of x coordinate to every combination of y coordinate per plane to the overall data frame led to a computational challenge.

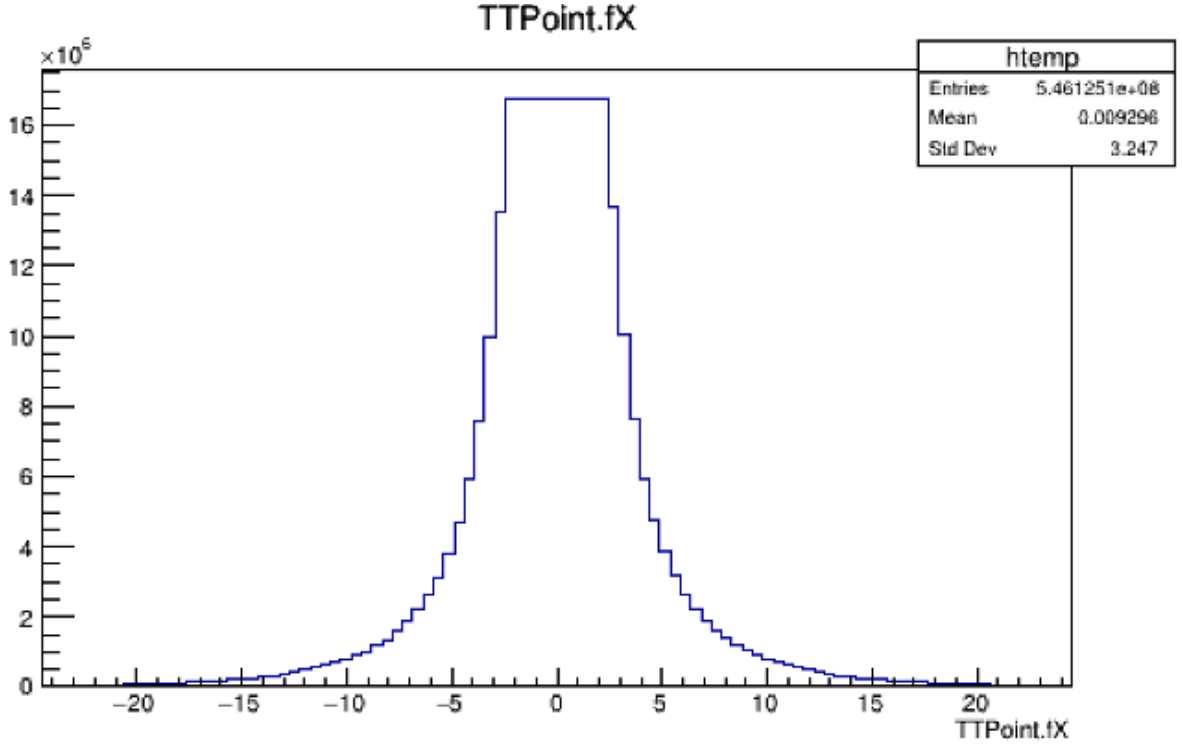


Figure 20: Visualisation of TTPoint.fX Leaf from dataset Tree

As can be seen in Figure 20 the number of entries within the TTPoint.fX leaf of the raw ROOT data file is approximately $5.46e08$ entries. As there are 200,000 total events contained within the ROOT file, the average number of hits per event is 2700. This means implementation of ghost hits would involve appending approximately 2700^2 , or 7 million new coordinates per event to the Pandas DataFrame during the Preprocessing stage. This was found to be unfeasible and computationally expensive and so, moving past this stage into training with these ghost hits was unsuccessful.

Alternately, the second approach to tackling the ghost hits problem would be to prevent corresponding x and y coordinates together, and rather, produce images of x and z coordinates, and y and z coordinates. This would eliminate the problem of ghost hits altogether. As such, these images were produced and compressed, and further bias analysis was carried with these images, to optimize the CNN for the new image inputs for these new images

3.2.2 Image Compression and Optimising CNN for new images

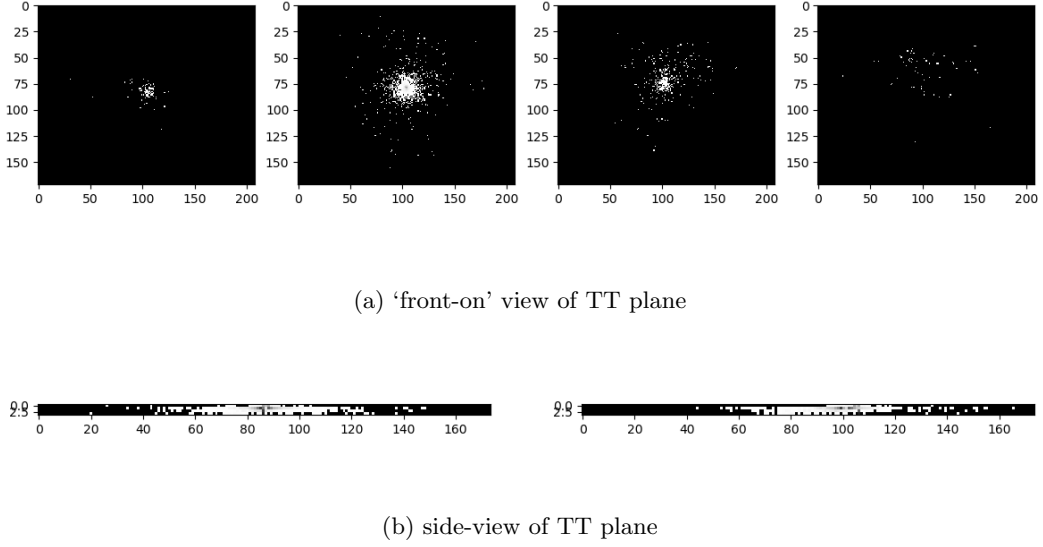


Figure 21: Event 7: original image vs. newly transformed, side-view, xz and yz planes image of TT planes

Figure 21b is shown above at a resolution of 174 pixels and not the actual 2100 pixels used to train the network. The 10 cm gap between each SciFi Tracker plane was compressed and removed, in order to make the z direction 4 pixels wide, as can be seen in Figure 21b. This was done for the sake of visualisation, as an image response shape of (2,4,2100) results only in a single line due to the every high number of pixels in the x/y-direction with respect to the z direction.

3.2.2.1 Using 4 CMBR Blocks

Using the Neural Network architecture as shown in Figure A.4 of the Appendix, the new side-view images were fed into the Neural Network. Initially, a dropout of 0.5 was implemented, to observe the impact of using a dropout in every block. The results of this can be seen in Figure 22

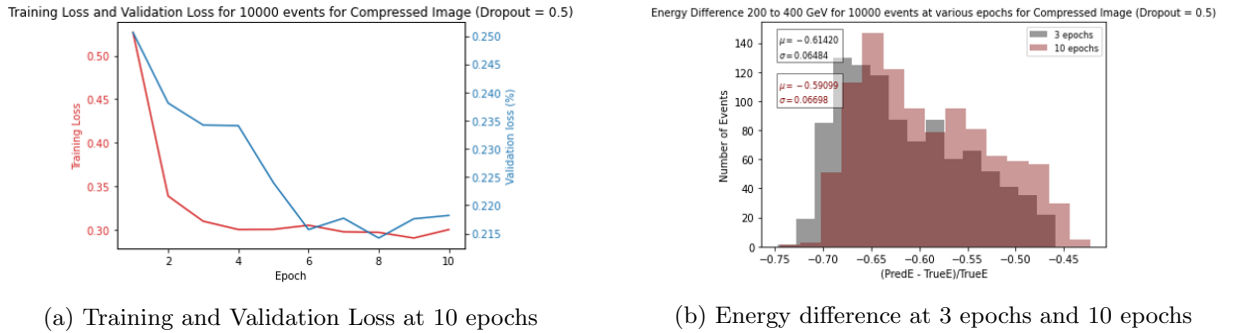


Figure 22: Results for side-view, xz and yz planes image with dropout = 0.5 for 10000 events

In Figure 22a, although both the validation loss and training loss seem to decrease over the 10 epochs, they are still very high in value, indicating that the network fails to learn very effectively in the first 10 epochs. The losses seem to initiate stabilising slightly by approximately the sixth epoch despite both loss values remaining relatively high. This may indicate the limitations of the current CNN structure to further learn and reduce the losses, or maybe due to the loss function discovering a local minimum.

In Figure 22b, the bias is extremely high for both 3 and 10 epochs, albeit slightly lower at 3 epochs, however this is not of significance, as the bias for both indicates an approximate 60% RE between the predicted and expected energy value, indicating the network has not learnt much.

Generally, the training loss is slightly higher than the validation loss. This would indicate underfitting and to counter this, the dropout layer was removed from the CNN structure.

Without the dropout layer, the results obtained are as followed:

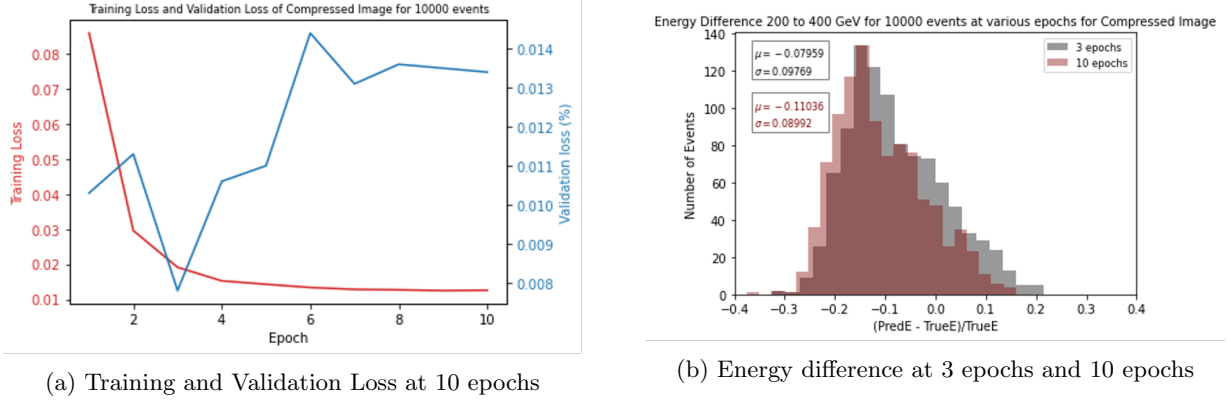


Figure 23: Results for side-view, xz and yz planes image for 10000 events (no dropout)

First and foremost, the bias as seen in Figure 23b has drastically improved both at 3 epochs and 10 epochs, decreasing from an approximate value of 60% to around 7.96% at epoch 3, and 11.0% at epoch 10. The training loss in Figure 23a follows a general decreasing trend, which is as expected, however, the validation loss decreases to its lowest value at epoch 3, after which it begins increasing, possibly explaining why the bias is lower at epoch 3 than at epoch 10.

As epochs progress, despite the training loss being higher, the fact that the validation loss is increasing may indicate overfitting of the data, as the network is becoming increasingly better at learning the training dataset, but simultaneously worsening its ability to perform on the validation dataset. This could be countered by increasing the size of the training dataset, or by changing the structure of the CNN.

Initially, increasing the number of events from 10000 events to 30000 events in the dataset was carried out, while maintaining the current CNN architecture, to observe the effect of an increased number of events in the training dataset on the bias. This is shown in Figure 24 below:

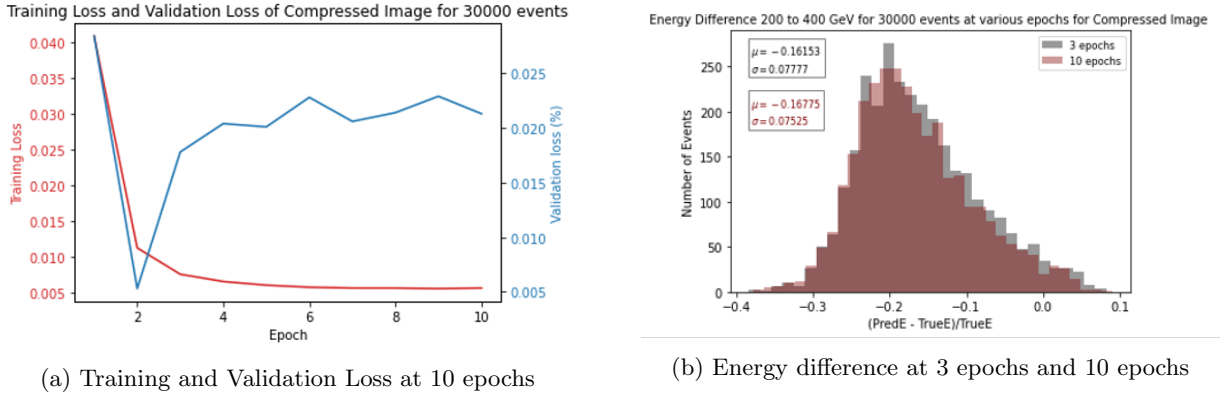


Figure 24: Results for side-view, xz and yz planes image for 30000 events

When comparing 24a and 23a, primarily it can be seen that the training loss stabilises at 0.005 for 24a around the 8th epoch, whereas for 23a for 10000 events this value is about 0.01. Similarly, the lowest validation loss reached at epoch 2 for Figure 24 was approximately 0.005, whereas for Figure 23 at epoch 3 it was 0.008. This indicates that by training with a higher number of events, the Neural Network may be able to learn more features, minimising loss more than with fewer events. Despite this however, the bias is higher for 30000 events than that for 10000 events for both epoch 3 and 10, both epochs yielding a bias of approximately 16.1% and 16.8% respectively, illustrating that using an increased number of events failed to improve the bias.

Thus it could be concluded that the problem of higher biases could be attributed to the CNN architecture itself, rather than the number of events used for training, and so the CNN architecture needed to be altered.

The alteration involved simplifying the CNN structure. This was done because a general trend of overfitting was observed using the CNN architecture in Figure A.4 of the Appendix. This would correspond to expectation, as due to the compression of the images to remove empty black pixels, and using 2 input images instead of the original 4, the network has a lower number of features to learn and the original network structure proves to be

too complex for the neural network to accurately predict electron energies. An over-complex network induces the risk of overfitting data.

3.2.2.2 Using 3 CMBR Blocks and altering input and output features

The simplified CNN architecture is as shown in Figure A.5 in the Appendix.

The results obtained by training the new CNN structure with the compressed images for 10000 events is as shown in Figure 25

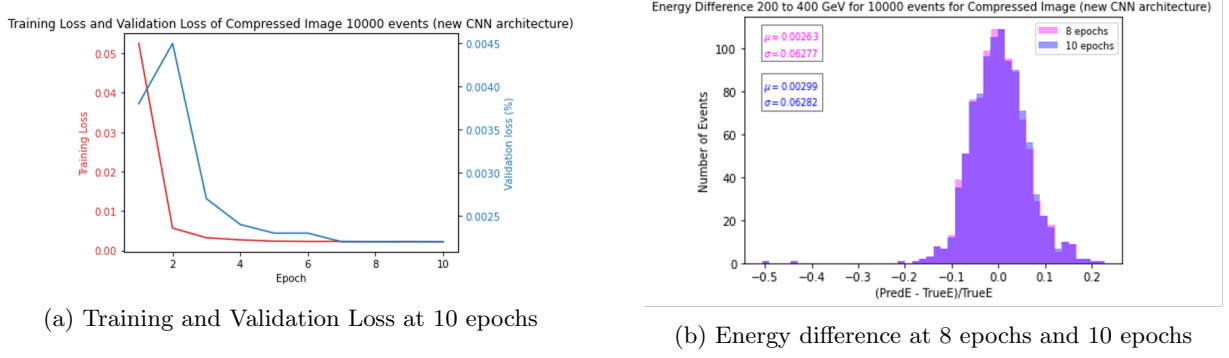


Figure 25: Results for side-view xz and yz planes image for 10000 events with simplified CNN architecture

As seen in Figure 25a, for 10000 events, the validation loss and training loss are extremely low, with the training loss converging to a value of approximately 0.005 and the validation loss converging to a value of approximately 0.002%. Thus post-alteration of the CNN, the ability of the network to predict the electron energies on seen and unseen data with high accuracy has improved.

The validation loss rises between the first and second epoch, after which it undergoes a steep descent beyond epoch 2. This could correlate to the sharp decrease of the training loss till epoch 2, perhaps indicating that it learns the most features during these epochs. Thus, beyond epoch 2, these newly learned features are applied to the validation dataset, possibly explaining why the validation loss decreases steeply from epoch 2 to epoch 3, after which it stabilises at approximately epoch 8.

This improved performance of the simplified network structure is reflected in the bias at epoch 8 and epoch 10. At epoch 8 and 10, the bias is seen to be at a value of 0.263% and 0.299% respectively, indicating a relatively low bias between predicted and true energy values. The standard deviation for both epoch 8 and 10 are seen to be approximately 0.56 to 0.57 %.

Finally, 30000 events were fed into the simplified CNN, and the results of this are as seen below in Figure :

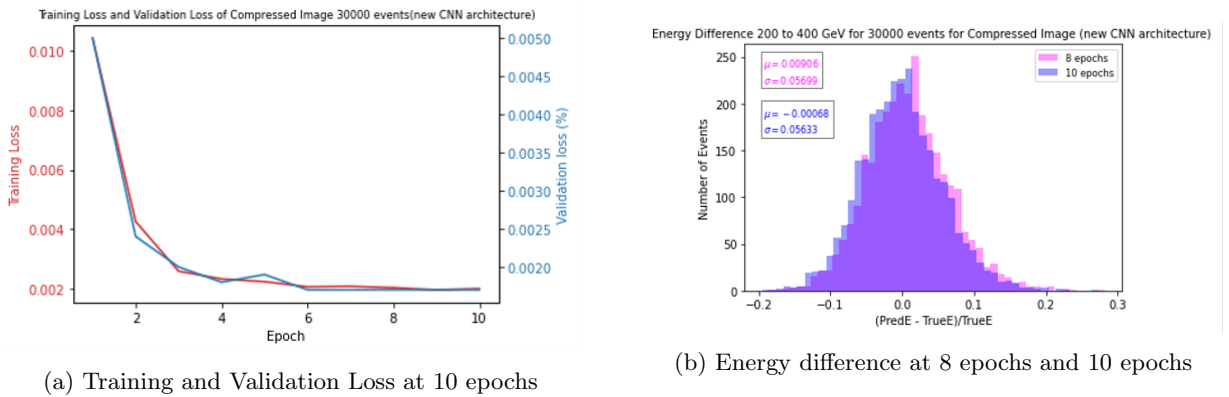


Figure 26: Results for side-view, xz and yz planes image for 10000 events with simplified CNN architecture

In figure 26a it can be seen that the network seems to learn well, with both the loss function and validation function following a near identical trend in their progression across epochs. The vast majority of learning seems to be in the first 2 epochs, after which the network continues to learn, albeit more gradually. Both the validation loss and training loss stabilize at epoch 9, at an approximate value of 0.002. As the lower the loss, the better the

model, it can be concluded that this model performs well and manages to accurately predict electron energies for the side-view, xz and yz images with the simplified CNN architecture.

This is also seen by the biases of the predictions with respect to the true values, at epochs 8 and 10. At epoch 8 and 10, the bias is seen to be at a value of 0.906% and 0.06% respectively, indicating that at epoch 10 for 30000 events, the bias is essentially eliminated, and the algorithm is seemingly successful in the energy reconstruction of electromagnetic showers.

3.2.3 Overall Performance of the pre-Optimised and Optimised Network for side-view, xz and yz images

To quantify how accurate this regression algorithm is for the side-view TT plane images for both the pre-optimised CNN structure and the simplified, optimised CNN structure, the **coefficient of determination** (R^2 -value) was calculated.

The R^2 value provides information about the goodness of a fit of a model, and is a statistical measure of how well the regression algorithm approximates actual data.

It is calculated as shown in Equation 5

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \quad (5)$$

where y_i is the actual value, \hat{y}_i is the predicted value and \bar{y} is the mean of the actual values.

The coefficients of determination, along with the respective Linear Regression plot are as shown below in Figure 27

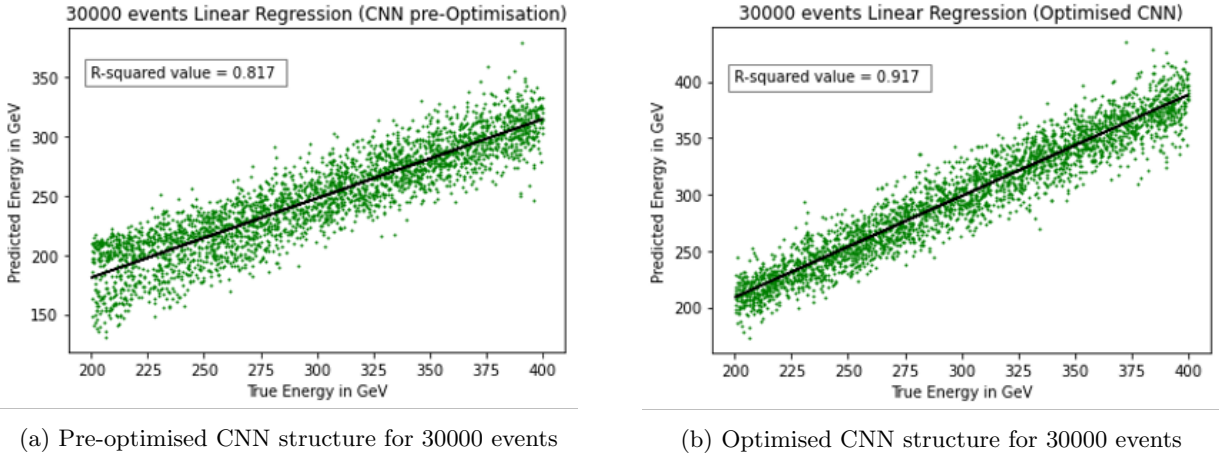


Figure 27: Regression Analysis and R-squared value for side-view, xz and yz planes image at 10 epochs

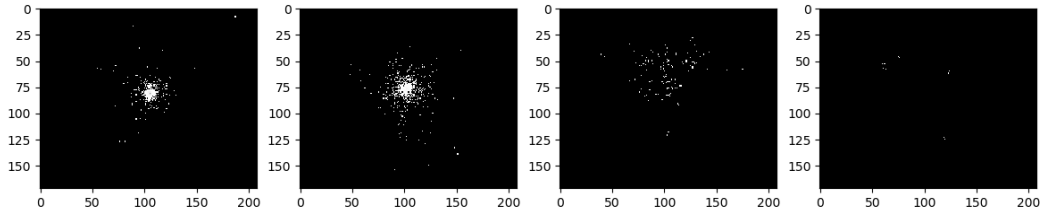
As can be seen in Figure 27, the relationship between the true energy values and the predicted energy values exhibits a relatively linear correlation.

The R^2 value indicates the extent to which the energy values are predictable based on the input true energy values. For the pre-optimised CNN, R^2 value was found to be 0.817. This shows that an approximately 81.7% variance in the predicted energy is predictable by the network when using the true energy as input labels. After optimising the CNN, the R^2 for a training with the same number of events and epochs was found to be 0.917 or 91.7%. Thus, it can be concluded that the predictive capacity of the model has improved for the optimised CNN architecture.

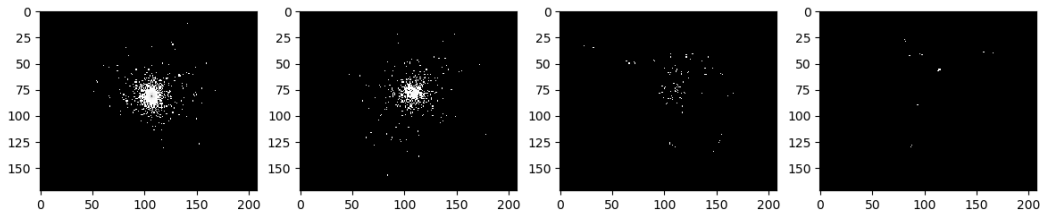
3.2.4 Comparison of events with low biases and high biases

While minimising the bias for various loss functions, specific indices of events were chosen based on the relative error between the predicted and true energy values. 3 events with a percent error lying beyond 20% and 3 events with a relative error within 0.01 were chosen to be visualised, to observe any existing TT response patterns with which the network predicts energies with better or worse accuracy.

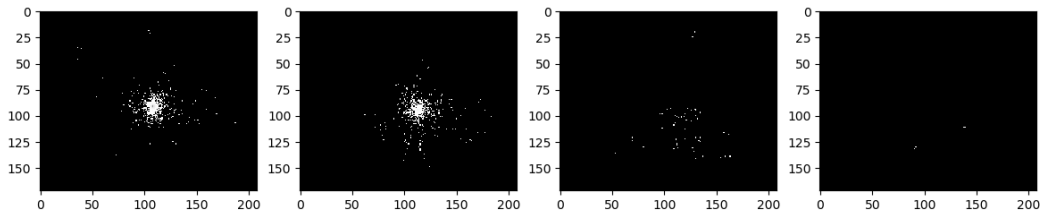
The events for which the relative error lies within 0.2 are as shown in Figure 28



(a) Event 1800



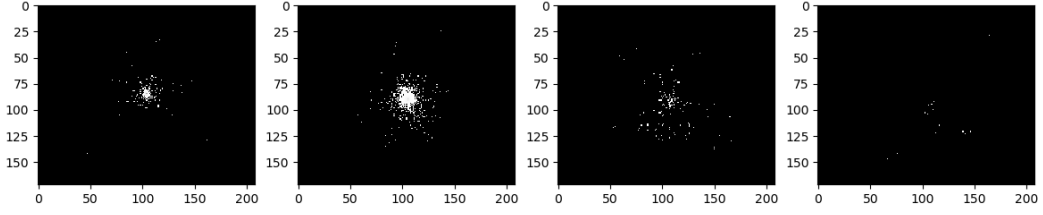
(b) Event 2081



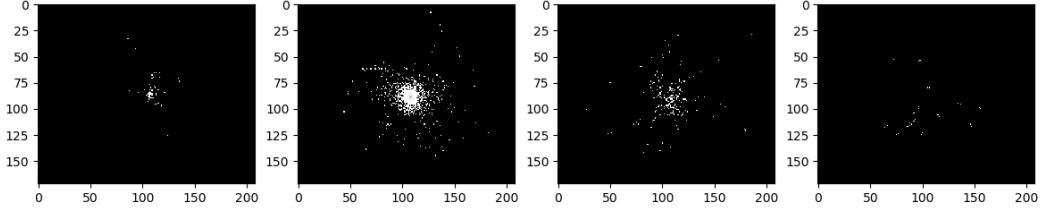
(c) Event 8684

Figure 28: Events with relative error over 20% of true energy value

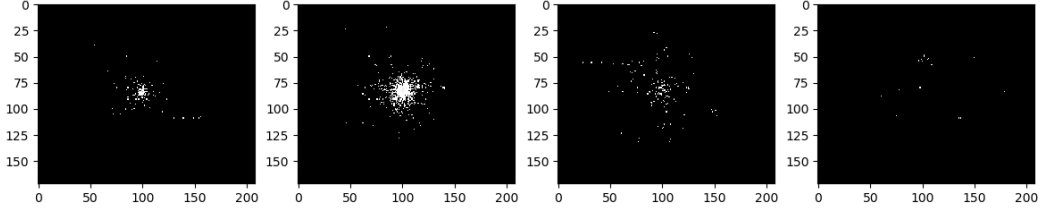
The events for which the relative error lies within 0.01 are as shown in Figure 29



(a) Event 325



(b) Event 5363



(c) Event 9103

Figure 29: Events with relative error within 1% of true energy value

The energy reconstruction for the events in Figure 29 are better than those in Figure 28. The overall hit pattern difference between these two sets of events is the image/plane on which the hits are incident. Events with better energy reconstruction appear to have the vast majority of their hits incident on the second TT plane than the first, whereas events with the higher relative error appear to have a more descending pattern of hit intensity across the planes, with the first TT plane having a relatively high number of hits than those in Figure 29.

This is a noteworthy observation as it would suggest that the ability of the network to reconstruct energies possesses a dependency on the number of hits on each plane. Since the CNN trains with hit patterns, it is not counter-intuitive to see that it considers the plane on which the hits are incident as an additional learned feature. This may occur as, during training, the algorithm comes across fewer events with hits in the first plane, and so its predictive power is lower for events with a higher number of hits in the first plane, than for those with more hits on the second plane.

4 Further Research and Conclusion

This thesis sought to reconstruct the energy of EM showers solely using SciFi data, with the inclusion of real-life phenomena. This was done by employing and implementing a Convolutional Neural Network to reconstruct the

energies of electrons showering through the calorimeter. Previous work had not addressed this problem of ghost hits and thus this was the main focus of this thesis. Research was carried out on four different loss functions and the bias they yield when utilized within the neural network: L1Loss normalised, L1Loss unnormalised, MSELoss normalised and MSELoss unnormalised. It was found that generally unnormalised loss functions tend to perform better for energy reconstruction within this algorithm at 30 epochs, with the exception of MSELoss normalised at 40 epochs, which yielded better results than those for itself at 30 epochs. It was also found that the loss function with the lowest overall bias was MSELoss unnormalised at 30 epochs, and this was subsequently used for all following bias analysis in this thesis.

The “border effects” study illustrated that there is no detrimental effect of border regions of energy on the overall bias between the predicted and actual energy values. Thus, the constraints on the energy range do not have an effect on the overall predictive capacity of the algorithm.

The primary aim of this thesis was the implementation of ghost hits and real-life scenario into the existing algorithm. It was found that implementing the corresponding ghost hits into current hit data for every event in the dataset proves to be computationally challenging and extremely slow. Thus, implementation of ghost hits into the current algorithm to observe the change in predictive capacity of the neural network was unsuccessful. Despite this, the real-life ghost hits scenario was tackled alternately by avoiding the problem altogether and transforming the existing algorithm’s front-view, xy Target Tracker plane images to two side-view, xz and yz and compressing them to minimize the number of empty black pixels present within them. Using these new images as input to the algorithm, these images were then successfully trained with and the overall CNN structure was simplified and optimized for these new images. The bias with these new images of 30,000 events decreased to a value of 0.06%, indicating that the bias between the predicted and true values had almost been eliminated with this optimised CNN structure.

Finally, it was observed that there lies a relationship between the accuracy of energy reconstruction and the number of hits on each plane. It was observed that the vast majority of predictions with a bias within 1% of the true value contain most of their hits on the second TT plane and fewer on the first.

The present study leaves a great deal of room for future research and improvement. Firstly, it would be interesting to study why generally the normalized loss function seem to perform worse than their unnormalised counterpart. A possible reason may be a decrease in the learning rate due to the lower step size and thus smaller updates of the network during back-propagation. To counter this, while using the normalized loss function, the learning rate of the algorithm may be increased to observe its impact on the bias. Additionally, when trying to implement ghost hits, the algorithm can be altered to save numpy arrays containing the hit information across events into pickle files, instead of Pandas DataFrames. Thus, a possible improvement could be to avoid appending to and looping through dictionaries and Pandas DataFrames as it proves to be computationally expensive and extremely slow, a possible reason why ghost hits implementation was unsuccessful in this thesis.

An observed characteristic of the images fed into the algorithm is that they are grayscale images, as opposed to the binary “hit vs. no hit” images which will exist in the actual detector. Within grayscale for this image, the various shades indicate the number of hits on a certain pixel. This is an additional feature which the current algorithm does not need to implement, and does not resemble the real-detector scenario. Thus, for future research, it is recommended to carry out bias studies with both the original xy TT plane images and compressed images with the implementation of binary black or white, rather than grayscale.

Finally, as was observed in section 3.2.4, there seems to be a correlation between the accuracy of energy prediction in the algorithm and which target tracker plane contains the most hits in an event. This is an undesirable feature for the CNN to learn, and possibly arises due to skewed training datasets with events with their vast majority of hits in the second TT plane. The CNN thus needs to be changed in order for the energy prediction to be less dependent on the TT plane on which a hit is incident, or the current CNN can be trained using different random states in order to shuffle the training and test datasets.

Overall, the aim of reconstructing energies for particle showers with the implementation of real-life scenario was achieved. With the new CNN optimised architecture in combination with side-view images as input, we are a step closer to implementing the existing Machine Learning Algorithm into the SND@LHC Pilot detector. This real-time analysis of data provides exciting prospects for further research into the fundamentals of the universe, with the discovery of new physics seemingly on the horizon.

5 Critical Reflection

This thesis was a steep learning curve for me. As it was essentially my first practical experience with programming, things such as Machine Learning and Convolutional Neural Networks had seemed like scary, obscure programming things which I thought I would never have been able to deal with, and to my pleasant surprise, by the end of the thesis I was proven wrong!

One of the biggest struggles I faced however, was this gap in programming knowledge and it is due to this that my progress has been slower during this thesis. I was unfamiliar with basic skills such as using GitHub and using a Linux environment. I had never worked with Machine Learning before, leading me to have to learn basic concepts such as learning rates, batch sizes etc. It is no doubt that working on this knowledge took away a great deal of time from doing the actual thesis work.

A learning experience for me has been to openly ask questions when stuck with a problem, rather than repeatedly trying to fix something to no avail. One of the biggest regrets I have is not asking certain questions earlier on during the thesis, instead of trying to unsuccessfully solve it myself as it would have meant utilizing time more efficiently and the overall work would have progressed faster.

I also found that the ebb and flow of working on this thesis was not consistent. There were definitely periods of time where I worked much harder than others and it was a personal challenge to constantly remain motivated for the entirety of the thesis. This problem arose partly due to the tedious nature of the work, often I would myself demotivated and exhausted after having been stuck on the same error for days. However, another contributing factor to this was being separated from my family for over a year due to the pandemic. Thus, an important part of my life was missing throughout the duration of this thesis, and this definitely impacted my efficiency and my work ethic at times.

Despite this, this thesis has been a positive learning experience and I enjoyed carrying it out immensely. Arguably the most enjoyable part was working with my supervisors Jacco, Elena, Paul and my partner-in-crime Joyce for the duration of this thesis. I have learnt a great deal about teamwork and collaboration, and having a set of supportive, likeminded (dare I say, friends) individuals guiding you through the work you're doing has been a pleasure. The open environment for asking questions and shooting ideas past each other was a dynamic way to carry out thesis research, and I am very grateful for that.

Overall, it was very rewarding to work on something so novel and something unfamiliar. I look forward to working further on this subject and reading about the SND@LHC and SHiP in newspapers. It's exciting to think I will be one among many looking forward to seeing some new discoveries made by these experiments.

6 Acknowledgements

I would like to express my utmost gratitude to my supervisors Dr. Elena Graverini and Dr. Jacco de Vries for their patience and guidance in helping me with this thesis, without which this would not have been possible.

I would like to extend my gratitude to EPFL and Nikhef for allowing me to utilise their GPU resources for extended period of time.

I would also like to thank Paul de Bryas for providing support and help with this thesis, and whose previous work it was a pleasure building on.

I offer a special thanks to my friend and partner-in-crime for this thesis Joyce, who it was a pleasure to work with.

My friends Pooja, Adrienne, Minnal and many others for constantly encouraging me and providing much needed moral support.

Most importantly, I'd like to thank my wonderful parents, who have always unconditionally supported me in my work through many sleepless nights and have always pushed me to strive for better.

7 References

- [1] Robson, B.A., 2018. The Matter-Antimatter Asymmetry Problem. *Journal of High Energy Physics, Gravitation and Cosmology*, 4(01), p.166.
- [2] Peter, A.H.G., Dark Matter: A Brief Review, arXiv (2012). arXiv preprint arXiv:1201.3942.

- [3] Fukuda Y, Hayakawa T, Ichihara E, Inoue K, Ishihara K, Ishino H, Itow Y, Kajita T, Kameda J, Kasuga S, Kobayashi K. Evidence for oscillation of atmospheric neutrinos. *Physical Review Letters*. 1998 Aug 24;81(8):1562.
- [4] De Simone D. Study of neutrino interactions with the SHiP experiment [thesis]. 2017.
- [5] Akhmedov EK. Neutrino physics. arXiv preprint hep-ph/0001264. 2000 Jan 25.
- [6] Lantwin, O., 2017. Search for new physics with the SHiP experiment at CERN. arXiv preprint arXiv:1710.03277.
- [7] De Lellis, G., 2015. Search for Hidden Particles (SHiP): a new experiment proposal. *Nuclear and Particle Physics Proceedings*, 263, pp.71-76.
- [8] Asaka, T. and Shaposhnikov, M., 2005. The MSM, dark matter and baryon asymmetry of the universe. *Physics Letters B*, 620(1-2), pp.17-26.
- [9] Alekhin, S., Altmannshofer, W., Asaka, T., Batell, B., Bezrukov, F., Bondarenko, K., Boyarsky, A., Choi, K.Y., Corral, C., Craig, N. and Curtin, D., 2016. A facility to Search for Hidden Particles at the CERN SPS: the SHiP physics case. *Reports on Progress in Physics*, 79(12), p.124201.
- [10] Ahdida, C., Komatsu, M., Grenier, D., Redi, F., Dedenko, L., Cadeddu, S., Chernyavskiy, M., Hushchyn, M., Obinyakov, B., Rakai, A. and Lacker, H.M., 2019. SHiP Experiment- Progress Report (No. CERN-SPSC-2019-010).
- [11] C. Ahdida et al. [SHiP], JINST 14 (2019) no.03, P03025 doi:10.1088/1748-0221/14/03/P03025 [arXiv:1810.06880 [physics.ins-det]].
- [12] Ahdida C, Akmete A, Albanese R, Alexandrov A, Andreini M, Anokhina A, Aoki S, Arduini G, Atkin E, Azorskiy N, Back JJ. SND@ LHC. arXiv preprint arXiv:2002.08722. 2020 Feb 20.
- [13] Goncharov M, Spentzouris P, NuTeV Collaboration. Precise measurement of dimuon production cross-sections in muon neutrino Fe and muon antineutrino Fe deep inelastic scattering at the Tevatron. arXiv preprint hep-ex/0102049. 2001 Feb 26.
- [14] Acquafredda R, Adam T, Agafonova N, Sanchez PA, Ambrosio M, Anokhina A, Aoki S, Ariga A, Ariga T, Arrabito L, Aufranc C. The OPERA experiment in the CERN to Gran Sasso neutrino beam. *Journal of Instrumentation*. 2009 Apr 29;4(04):P04018.
- [15] OPERA collaboration. Observation of a first candidate in the OPERA experiment in the CNGS beam. *Phys. Lett. B*. 2010;691:138-45.
- [16] Agafonova N, Aleksandrov A, Anokhina A, Aoki S, Ariga A, Ariga T, Asada T, Autiero D, Badertscher A, Dhahbi AB, Bender D. New results on τ appearance with the OPERA experiment in the CNGS beam. *Journal of High Energy Physics*. 2013 Nov 1;2013(11):36.
- [17] Agafonova N, Aleksandrov A, Anokhina A, Aoki S, Ariga A, Ariga T, Asada T, Autiero D, Dhahbi AB, Badertscher A, Bender D. Evidence for τ appearance in the CNGS neutrino beam with the OPERA experiment. *Physical Review D*. 2014 Mar 5;89(5):051102.
- [18] OPERA collaboration, Agafonova N, Aleksandrov A, Anokhina A, Aoki S, Ariga A, Ariga T, Asada T, Bender D, Bertolin A, Bozza C. Observation of tau neutrino appearance in the CNGS beam with the OPERA experiment. *Progress of theoretical and experimental physics*. 2014 Oct 1;2014(10):101C01.
- [19] Blanc, F., 2013. Scintillating Fiber Trackers: Recent Developments And Applications.
- [20] Kaushik, V.S., 2002. Electromagnetic Showers and Shower Detectors. Internal Documents, University of Texas at Arlington.
- [21] Battaglia, M., Silicon Sensors for Collider Physics from Physics Requirements to Vertex Tracking Detectors. 2016
- [22] Bourilkov D. Machine and deep learning applications in particle physics. *International Journal of Modern Physics A*. 2019 Dec 20;34(35):1930019.
- [23] Shirobokov S, Filatov A, Belavin V, Ustyuzhanin A. Machine Learning for electromagnetic showers reconstruction in emulsion cloud chambers. In *J. Phys: Conf. Ser* 2018 (Vol. 1085, p. 042025).

- [24] Elagin A, Murat P, Pranko A, Safonov A. Probabilistic particle flow algorithm for high occupancy environment. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2013 Mar 21;705:93-105.
- [25] Baldi, Pierre, Jianming Bian, Lars Hertel, and Lingge Li. "Improved energy reconstruction in NOvA with regression convolutional neural networks." Physical Review D 99, no. 1 (2019): 012011.
- [26] Original algorithm from S. Shirobokov. <https://github.com/shir994/SNDtrackers>
- [27] De Bryas, P. (2020). *Development and commissioning of a machine learning algorithm for real time reconstruction of electromagnetic showers with a scintillating fibres tracker*. École polytechnique fédérale de Lausanne.
- [28] *Developed algorithm from P. de Bryas*. https://github.com/pauldebryas/SND_trackers_My_data
- [29] Hijazi S, Kumar R, Rowen C. Using convolutional neural networks for image recognition. Cadence Design Systems Inc.: San Jose, CA, USA. 2015:1-2.
- [30] Nielsen MA. Neural networks and deep learning. San Francisco, CA: Determination press; 2015 Sep 25.
- [31] Deshpande A. A Beginner's Guide To Understanding Convolutional Neural Networks [Internet]. 2016. Available from: <https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>
- [32] Dietterich T. Overfitting and undercomputing in machine learning. ACM computing surveys (CSUR). 1995 Sep 1;27(3):326-7.
- [33] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167. 2015 Feb 11.
- [34] Al-Turany, M., Bertini, D. and Koenig, I., 2006. CbmRoot: Simulation and analysis framework for CBM experiment. Computing in High Energy and Nuclear Physics (CHEP-2006), 1, pp.170-171.
- [35] Bertini, D., A-Turany, M., Koenig, I. and Uhlig, F., 2008. The FAIR simulation and analysis framework. In Journal of Physics: Conference Series (Vol. 119, No. 3, p. 032011). IOP Publishing.
- [36] Kannoja SP, Jaiswal G. Effects of varying resolution on performance of CNN based image classification: An experimental study. Int. J. Comput. Sci. Eng. 2018;6:451-6.
- [37] Koehrsen W. Overfitting vs. underfitting: A complete example. Towards Data Science. 2018.
- [38] Ede JM, Beanland R. Adaptive learning rate clipping stabilizes learning. Machine Learning: Science and Technology. 2020 Apr 28;1(1):015011.

A Appendix

```
def digitize_signal(event, params, filters=1):
    """
    Convert pandas DataFrame to image
    :param event: Pandas DataFrame line
    :param params: Detector configuration
    :param filters: Number of TargetTrackers in the simulation
    :return: numpy tensor of shape (n_filters, H, W).
    """
    shape = (filters,
             int(np.ceil(params.snd_params["Y_HALF_SIZE"] * 2 * CM_TO_MUM /
                          params.snd_params["RESOLUTION"])),
             int(np.ceil(params.snd_params["X_HALF_SIZE"] * 2 * CM_TO_MUM /
                          params.snd_params["RESOLUTION"])))
    response = np.zeros(shape)

    for x_index, y_index, z_pos in zip(np.floor((event['X'] + params.snd_params["X_HALF_SIZE"]) * CM_TO_MUM /
                                                params.snd_params["RESOLUTION"]).astype(int),
                                       np.floor((event['Y'] + params.snd_params["Y_HALF_SIZE"]) * CM_TO_MUM /
                                                params.snd_params["RESOLUTION"]).astype(int),
                                       event['Z']):
        response[params.tt_map[bisect_left(params.tt_positions_ravel, z_pos)],
                 shape[1] - y_index - 1,
                 x_index] += 1
    return response
```

Figure A.1: Original Digitize_signal function used for producing front-on, xy images

```

class Block(nn.Module):
    def __init__(self, in_channels, out_channels, k_size=3, pool=False):
        super().__init__()
        self.block = nn.Sequential()
        self.block.add_module("conv", CoordConv(in_channels, out_channels,
                                                kernel_size=(k_size, k_size), stride=1, with_r=True))

        if pool:
            self.block.add_module("Pool", nn.MaxPool2d(2))
        self.block.add_module("BN", nn.BatchNorm2d(out_channels))
        self.block.add_module("Act", nn.ReLU())
        # self.block.add_module("dropout", nn.Dropout(p=0.5))

    def forward(self, x):
        return self.block(x)

class SNDNet(nn.Module):
    def __init__(self, n_input_filters):
        super().__init__()
        self.model = nn.Sequential(
            Block(n_input_filters, 32, pool=True),
            Block(32, 32, pool=True),
            Block(32, 64, pool=True),
            Block(64, 64, pool=True),
            #Block(32, 32, pool=True),
            #Block(128, 128, pool=False),
            Flatten(),
            nn.Linear(256, 1),
            # nn.ReLU(),
            # nn.Dropout(p=0.5),
            # nn.Linear(512, 512),
            # nn.ReLU(),
            #nn.Linear(1280, 2)
        )

```

Figure A.2: Original CNN structure used for training with front-on, xy images


```

def digitize_signal(event, params, filters=1):

    shape = (filters,4,int(np.ceil(params.snd_params["X_HALF_SIZE"] * 2 * CM_TO_MUM /
                                params.snd_params["RESOLUTION"])))

    response = np.zeros(shape)

    first_layer_x = []
    first_layer_y = []
    second_layer_x = []
    second_layer_y = []
    third_layer_x = []
    third_layer_y = []
    fourth_layer_x = []
    fourth_layer_y = []

    for i in range(len(event['Z'])):
        if np.logical_and(event['Z'][i] >= -3041.0, event['Z'][i] <= -3037.0):
            first_layer_x.append(event['X'][i])
            first_layer_y.append(event['Y'][i])
        elif np.logical_and(event['Z'][i] >= -3032.0, event['Z'][i] <= -3027.0):
            second_layer_x.append(event['X'][i])
            second_layer_y.append(event['Y'][i])
        elif np.logical_and(event['Z'][i] >= -3022.0, event['Z'][i] <= -3017.0):
            third_layer_x.append(event['X'][i])
            third_layer_y.append(event['Y'][i])
        elif np.logical_and(event['Z'][i] >= -3012.0, event['Z'][i] <= -3007.0):
            fourth_layer_x.append(event['X'][i])
            fourth_layer_y.append(event['Y'][i])
        else:
            pass

    event_comp_x = pd.DataFrame({'z1': pd.Series(first_layer_x), 'z2': pd.Series(second_layer_x), 'z3': pd.Series(third_layer_x), 'z4': pd.Series(fourth_layer_x) })
    event_comp_x.fillna(0, inplace=True)
    event_comp_y = pd.DataFrame({'z1': pd.Series(first_layer_y), 'z2': pd.Series(second_layer_y), 'z3': pd.Series(third_layer_y), 'z4': pd.Series(fourth_layer_y) })
    event_comp_y.fillna(0, inplace=True)

    for x_one, x_two, x_three, x_four in zip(np.floor((event_comp_x['z1'] + params.snd_params["X_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                            np.floor((event_comp_x['z2'] + params.snd_params["X_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                            np.floor((event_comp_x['z3'] + params.snd_params["X_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                            np.floor((event_comp_x['z4'] + params.snd_params["X_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int)):

        # print(x_one)
        # print(x_two)
        # print(x_three)
        # print(x_four)
        response[0,0,x_one] +=1
        response[0,1,x_two] += 1
        response[0,2,x_three] +=1
        response[0,3,x_four] += 1

    for y_one, y_two, y_three, y_four in zip (np.floor((event_comp_y['z1'] + params.snd_params["Y_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                              np.floor((event_comp_y['z2'] + params.snd_params["Y_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                              np.floor((event_comp_y['z3'] + params.snd_params["Y_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int),
                                              np.floor((event_comp_y['z4'] + params.snd_params["Y_HALF_SIZE"])*CM_TO_MUM/
                                                    params.snd_params["RESOLUTION"]).astype(int)):

        response[1,0,shape[1] - y_one - 1] += 1
        response[1,1,shape[1] - y_two - 1] += 1
        response[1,2,shape[1] - y_three - 1] += 1
        response[1,3,shape[1] - y_four - 1] += 1

    # response = (response>=1).astype(int)
    # print(response)
    return response

```

Figure A.3: New Digitize_signal function used for producing side-view, xz and yz images

```

class Block(nn.Module):
    def __init__(self, in_channels, out_channels, k_size=2, pool=False):
        super().__init__()
        self.block = nn.Sequential()
#         self.block.add_module("conv", CoordConv(in_channels, out_channels,
#                                                    kernel_size=(k_size,k_size), stride=1, with_r=True,padding=1))
#         self.block.add_module("conv",nn.Conv2d(in_channels, out_channels,
#                                                    kernel_size=(2,4), stride=1, padding=1))

        if pool:
            self.block.add_module("Pool", nn.MaxPool2d((1,4)))
#         self.block.add_module("BN", nn.BatchNorm2d(out_channels))
        self.block.add_module("Act", nn.ReLU())
#         self.block.add_module("dropout", nn.Dropout(p=0.5))

    def forward(self, x):
        return self.block(x)

class SNDNet(nn.Module):
    def __init__(self, n_input_filters):
        super().__init__()
        self.model = nn.Sequential(
            Block(n_input_filters, 32, pool=True),
            Block(32, 32, pool=True),
            Block(32, 64, pool=True),
            Block(64, 64, pool=False),
#             Block(32, 32, pool=True),
#             #Block(128, 128, pool=False),
            Flatten(),
            nn.Linear(448, 1),
            #nn.ReLU(),
#             nn.Dropout(p=0.5),
#             #nn.Linear(512, 512),
#             #nn.ReLU(),
            #nn.Linear(1280,2)
        )

```

Figure A.4: Altered CNN structure used for training with side-view, xz and yz images, with same Block structure as the original but changed kernel size and MaxPooling

```

class Block(nn.Module):
    def __init__(self, in_channels, out_channels, k_size=2, pool=False):
        super().__init__()
        self.block = nn.Sequential()
#         self.block.add_module("conv", CoordConv(in_channels, out_channels,
# #                                     kernel_size=(k_size,k_size), stride=1, with_r=True,padding=1))
        self.block.add_module("conv",nn.Conv2d(in_channels, out_channels,
                                                kernel_size=(2,4), stride=1, padding=1))

        if pool:
            self.block.add_module("Pool", nn.MaxPool2d((1,4)))
#         self.block.add_module("BN", nn.BatchNorm2d(out_channels))
        self.block.add_module("Act", nn.ReLU())
#         self.block.add_module("dropout", nn.Dropout(p=0.5))

    def forward(self, x):
        return self.block(x)

class SNDNet(nn.Module):
    def __init__(self, n_input_filters):
        super().__init__()
        self.model = nn.Sequential(
            Block(n_input_filters, 32, pool=True),
            Block(32, 32, pool=True),
            Block(32, 64, pool=True),
            Block(64, 64, pool=False),
#             Block(32, 32, pool=True),
#             #Block(128, 128, pool=False),
            Flatten(),
            nn.Linear(448, 1),
            #nn.ReLU(),
#             nn.Dropout(p=0.5),
#             #nn.Linear(512, 512),
#             #nn.ReLU(),
            #nn.Linear(1280,2)
        )

```

Figure A.5: Simplified CNN structure used for training with side-view, xz and yz images, with fewer Blocks than as original and changed kernel size and MaxPooling