1 Project Summary

This project has been completed in collaboration with the Deep Underground Neutrino Experiment (DUNE) under the direct supervision of William Dallaway and Professor Nikolina Ilic. In this paper, we provide an overview of theoretical physics concepts, DUNE's objectives, and the GNN NuGraph, as well as its limitations within the DUNE context. We further discuss the modifications we made to NuGraph to enhance training, validation, and test performance, along with the theoretical rationale behind these changes. The modifications that we made to the model showed vast improvement on test accuracy and overall model preformance. Moreover, we highlight additional ways in which the model can be improved, and attribute these modifications to future work.

2 Introduction

Before discussing the main components of this project, we provide a brief overview of neutrino particle physics, mixing probabilities particular in the context of tau neutrinos, and given a summary of the goals and methodologies of DUNE.

2.1 Neutrino Physics

Neutrinos are extremely lightweight, electrically neutral subatomic particles that interact predominantly via weak nuclear and gravitational forces. Initially thought to be massless, experimental evidence supported by theoretical models like the Standard Model and grand unified theories has confirmed that neutrinos possess small, non-zero masses. Neutrino oscillations, where a neutrino can change between electron, muon, and tau flavors, demonstrate these non-zero masses and remain a key puzzle in particle physics. The neutrino mass eigenstates ν_1, ν_2, and ν_3, with masses m_1, m_2, and m_3 respectively, form the basis for neutrino states, allowing for combinations that yield the distinct flavors ν_e, ν_μ, and ν_τ. This linear combination can be unambiguously defined by the 3×3 unitary mixing matrix, commonly referred to as the PMNS matrix:

$$\begin{pmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{pmatrix} = \underbrace{\begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{e\tau 2} & U_{\tau 3} \end{pmatrix}}_{\text{PMNS}} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} \tag{1}$$

The presence of non-zero off-diagonal entries and distinct mass eigenstates (m_i) leads to neutrino oscillations. Experimental evidence indicates that the oscillation probability is related to the squared mass differences ($\Delta m^2_{ij} = m^2_i - m^2_j$). The space of 3×3 unitary matrices is generally nine-dimensional, but in the PMNS matrix, five degrees are absorbed into phases of lepton fields. Consequently, the PMNS matrix can be uniquely described by three mixing angles (θ_{12}, θ_{13}, and θ_{23}) and a single phase angle (δ_{CP}) that quantifies charge parity violations. In this case, the PMNS matrix can then be decomposed into the product of three rotation matrices about each axis:

$$\begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{e\tau 2} & U_{\tau 3} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix}}_{\substack{\text{Atmospheric Neutrinos} \\ \nu_\mu \longleftrightarrow \nu_\tau}} \overbrace{\begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta} & 0 & c_{13} \end{pmatrix}}^{\substack{\text{Reactor/Accelerator Neutrinos} \\ \nu_\mu \longleftrightarrow \nu_e}} \underbrace{\begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{Solar Neutrinos} \\ \nu_e \longleftrightarrow \nu_X}} \tag{2}$$

2.2 Tau Neutrinos (ν_τ)

Tau neutrinos (ν_τ) pose significant challenges for detection due to kinematic constraints, making their appearance rare at typical beam energy levels. Moreover, the associated tau leptons exhibit multiple decay modes, complicating their identification as they can mimic ν_e and ν_μ charged current events or neutral current events. These inherent difficulties contribute to the prediction of approximately 800 tau neutrino events per year in the high-energy neutrino regime. The oscillation probability from ν_μ to ν_τ can be derived from 1, and provides an oscillation probability of $\nu_\mu \to \nu_\tau$ is given by

$$P(\nu_\mu \to \nu_\tau) = \sin^2(2\theta 23) \sin^2\left(\frac{\Delta m^2_{23} L}{4E}\right) \tag{3}$$

Because the near detector is in close proximity to the source beam, which primarily generates non-ν_τ particles, it is anticipated that very few, if any, ν_τ charged current events will be observed. In particular, the oscillation probability of an initially ν_μ flavoured neutrino transitioning to the ν_τ flavour is given by

$$P(\nu_\mu \to \nu_\tau) = \sin^2(2\theta_{23}) \sin^2 \frac{\Delta m^2_{41} L}{4E} \tag{4}$$

where Δm^2_{41} is the mass difference of a new mass eigenstate in the $3+1$ scenario [1]. This process is referred to as short-baseline sterile-driven ν_τ appearance. In summary, the high-energy beam mode will offer chances to detect ν_τ charged current events in the near detector and measure ν_τ charged current cross sections in the far detector modules.

2.3 DUNE Colaboration

The DUNE experiment aims to measure oscillation probabilities for muon neutrinos or their antineutrinos transitioning into electron neutrinos, essential for understanding neutrino flavor mixing and potential violations of charge parity symmetry. Central to DUNE are a high-intensity neutrino beam, a near detector providing initial energy spectra measurements, and a far detector for detailed interaction reconstruction, located 1300 km away underground. A schematic diagram highlighting the main hardware components of the experiment is included in Figure (1). The far detector comprises four liquid argon time projection chamber modules, facilitating precise event reconstruction, with two modules utilizing single-phase technology and one utilizing dual-phase, while the fourth remains under development. The near detector serves as the control, minimizing systematic errors and uncertainties in neutrino interaction reconstruction, crucial for accurately predicting far detector observations.

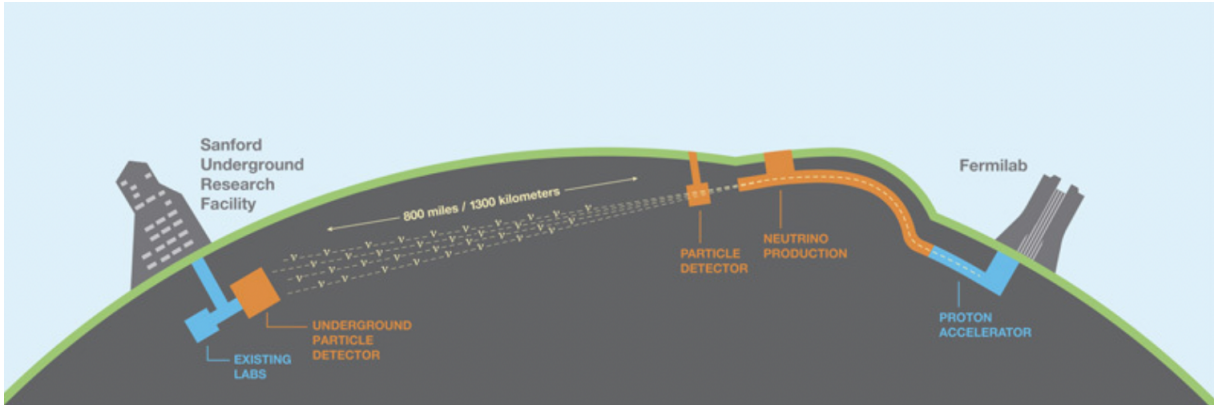

Figure 1: A schematic diagram highlighting the key components of the DUNE physics experiment. The data which will eventually be given to NuGraph is collected in whole at the far detector located at the Stanford Underground Research Facility.

The experiment also aims to detect supernova neutrinos and study nucleon decay, necessitating detectors with large, stable reference masses. The far detector's single-phase technology immerses all components in liquid argon, enabling precise event reconstruction based on ionization patterns. Conversely, the dual-phase technology amplifies ionization signals through avalanches, offering enhanced signal-to-noise ratios and longer drift lengths for improved resolution. Both technologies employ photon detectors to capture scintillation light emitted by ionized particles, essential for event reconstruction and drift correction. The success of DUNE hinges on its detectors' ability to provide high-resolution images of neutrino interactions, crucial for uncovering new physics beyond the standard model and advancing our understanding of neutrino properties.

3 NuGraph

The team at CERN directly utilized the model as found on the Github, structuring the ProtoDUNE dataset to fit the parameters required by the model. This repository hosts NuGraph which is designed to reconstruct particle interactions within neutrino physics detector environments. Its main purpose is to classify particle types based on detector hits using semantic segmentation. Additionally, it offers secondary functionalities including background hit rejection, event classification, clustering, and vertex reconstruction. In this section, we describe the orginal model as forked, and discuss the shortcoming of and issues with this model in the context of training on the ProtoDUNE dataset.

3.1 Model Architecture and Structure

The NuGraph project revolves around developing and training a graph neural network designed for the detection of tau neutrinos. The key components of the model reside primarily in the `numl/NuGraph/scripts` directory, with particular emphasis on the `train.py` and `NuGraph2.py` scripts. These scripts orchestrate the training and inference processes, respectively, utilizing various libraries and utilities.

The model leverages PyTorch extensively, employing its `torch` library for tensor operations, optimization, and neural network construction. Within `torch.nn`, the `Sequential` module is prominently utilized for constructing sequential layers. Additionally, specialized components such as the `SimpleConv` message passing operator and functions from `torchmetrics` for metric computations are integrated into the model architecture.

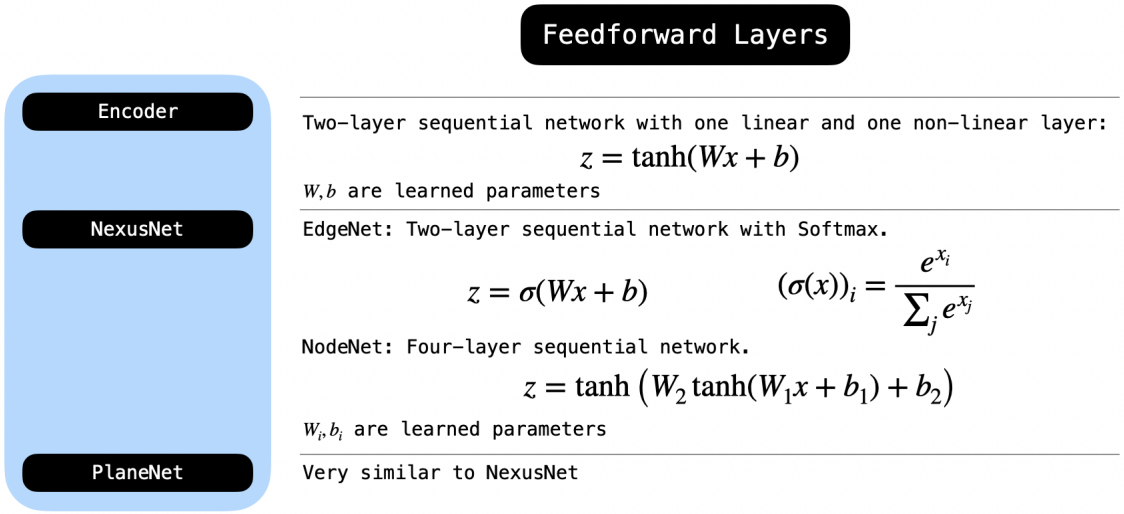

Figure 2: A diagram which visually depicts the feedforward layers of the network. The encoder layer is used to predict the event label of a given sample, and is the centre of focus of our development.

Among the utility modules, `PositionFeatures.py`, `FeatureNorm.py`, and `RecallLoss.py` play crucial roles. The `RecallLoss` function, for instance, serves as the optimization objective, minimizing the loss $\mathcal{L}$ by adjusting model parameters θ. It computes a weighted cross-entropy loss, where the weight is determined by the recall of model predictions relative to ground truth labels.

The training pipeline, orchestrated by `train.py`, encompasses two main functions: `configure` and `train`. The former initializes arguments and model parameters from command-line inputs, while the latter orchestrates dataset configuration, invokes the main model (`NuGraph2.py`), logs training progress for Slurm, and employs the PyTorch Lightning `Trainer` module for network training.

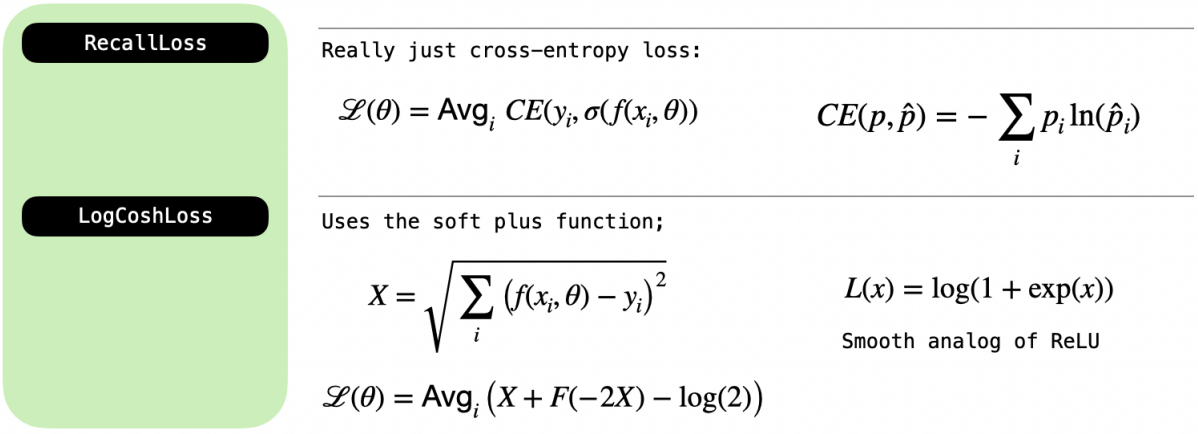

Figure 3: A diagram which visually depicts the loss functions used by each part of the model. The `RecallLoss` is used to train the Encoder layer, while the `LogCoshLoss` is used by `NexusNet` and `PlaneNet`.

Within the `nugraph` directory, the `data` and `models` subdirectories contain essential components. `data` encapsulates data configuration methods, including the `H5DataModule` responsible for extracting pertinent information from input data files. On the other hand, `models` comprises various network architectures and associated functionalities, such as data encoding (`encoder.py`), decoding (`decoder.py`), and network

architecture components (`nexus.py`).

Before training begins, the training dataset is partitioned into batches of samples. During each epoch of training, the model loops through these batches, updating the model parameters are each batch. This marks the completion of a single iteration. The batching process is handled by the `BalancedSampler` class. This class is designed to create balanced batches for training or evaluation while maintaining the relative frequency of each class within the entire dataset. Upon initialization, the sampler calculates the number of batches based on the specified batch size (which is an input parameter to the model) and determines the fraction of the dataset length to be used for balancing. It then separates the dataset into outliers, consisting of samples with the largest sizes, and the bulk of the dataset. These outliers are distributed among bins to ensure each bin receives a fair share, while the remaining samples are allocated to each bin to fill them up to the batch size limit, preventing overfilling. The methodologies used to batch the data can have significant impacts on the success of training, and is discussed in further detail in Section 3.3.

3.2 Model Training

The model is trained using the `train.py` script. Due to the large number of parameters and variables needed to be specified in order for training to run, we utilize bash scripts to easly and consistently run training jobs. In particular, passing user information, runtime, GPU and RAM requests, as well as passing training parameters such as batchsize, number of training epochs, and training features is all heavily simplified through the use of this training script. The main command within the bash script `run_train.sh` is as follows;

```
singularity exec --nv {NUML_PATH}/numl:v23.9.0.sif python {NUGRAPH_PATH}/train.py
--name {LOG_NAME} --logdir /scratch/{USERNAME} --data-path /project/6079563/{FILE_TO_TRAIN}
--batch-size 32 --in-feats 8 --event --semantic --epochs 80
```

This command passes all required training parameters required to properly train the model. The above command further stipulates that the model is to train on both the event and semantic labels. To begining training the model, we run the following command within the Canada compute cluster, calling `run_train.sh`

```
sbatch -J {job-name} run_train.sh {log-directory} {dataset-directory}          (5)
```

For example, the boiler-plate command that we utilized for our training purposes is:

```
sbatch -J murdock_test run_train.sh murdock_log MCprod/training_hdf5/new_all_beam.gnn_processed.h5
```
(6)

To resume training from the previously logged checkpoint, a very similar command to 5 is to be called, in our case calling `run_resume`.

Running either of the above commands will initiate training using the specified training sample, and will output training logs and checkpoints into the specified {`log-directory`}. Moreover, we can use the Tensorboard interface to visualise these metrics and track the network's training progress. By accessing tensorboard by launching a local host in the cluster home directory, we are able to access the current status of training, along with *all* outputted metrics. This real-time allows us to make inferences on the quality of training more accurately, and ultumately contribute to our ability to maximize the models preformance.

3.3 Shortcomings

In this section, we provide a breif descriptions of some of the pitfalls that NuGraph is subject to in the context of our dataset in particular. Each sample that we have to train the model with has one of four event labels associated with it; muon neutrino, electron neutrino, tau neutrino, or neutral current sample. To train the model, we are using training samples generated from ProtoDUNE. Unfortunetely, the data collected from this testing module is heavilty skewed *away* from tau neutrinos.

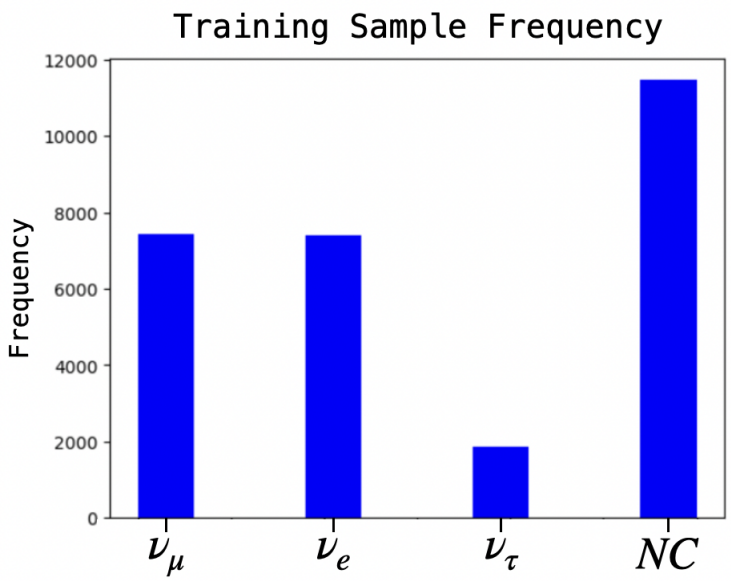

Figure 4: A bar graph displaying the frequency of each sample type within the training dataset. Note the relatively low frequency of the tau neutrino samples, with a large dominance on neutral current samples. This skewed frequency severely effects the models ability to accurately and consistency classify particular classes, particularly in the case of tau neutrino samples.

Included in Figure (4) is a bar graph illustrating the frequency distribution of each sample type within the training dataset. It is noteworthy that the tau neutrino samples exhibit a notably low frequency, contrasting sharply with the predominant occurrence of neutral current samples. This skewed frequency distribution significantly impacts the model's capacity to accurately and consistently classify specific classes, particularly evident in the case of tau neutrino samples. The disparity in sample frequencies poses a considerable challenge for the model, potentially leading to biases and inaccuracies in classification outcomes, especially for underrepresented classes like tau neutrinos. Addressing this imbalance in the dataset is crucial for improving the model's robustness and generalization performance across all sample types.

As described in Section (3.1), the model uses a batching module called `BalancedSampler`.

4 Model Refinements and Training Results

In this section, we discuss the modifications made to the general-purpose model to ensure better preformance on the tasks required for DUNE. Moreover, we highlight the results of these modifications and indicate overall preformance of the modified models in comparison to the vanila model across a variety of metrics.

4.1 Modifications

In efforts to combat the issues resulting from a heavily imbalanced dataset, we implemented three distinct modifications to the architecture of the network itself. In particular, we test the effect of implementing a modified, weighted loss function, the effects of expanding the model's width within the event decoder, and finally the reprecussions of altering the batching process of the data before training starts. Each of these methods has the same goal of improving the model's training accuracy on tau neutrino data samples, and does so in distinctly different ways. After implementing and testing these modifications, a variety of results were observed, all of which show the model to exceed its previous preformance on the validation and test datasets. These three modifications are summarized bellow in full detail, included with overarching theoretical descriptions and motivations.

4.1.1 Weighted Loss Function

A machine learning model is trained by feeding input data into a function which is a composition of linear layers and nonlinear activation functions which are applied in sequence. The linear layers consist of large matrices called weights, and affine transformation vectors, called biases. Each of the input samples has a "correct" output, which the model is intended to learn to predict over time. They was the model learns to do this is by minimizing an error function, called the loss. NuGraph utilizes a variety of loss functions, but in training the model, it utilizes a cross entropy loss function, referred to locally as the `RecallLoss`. This is given by, as highlighed in Figure 3, the equation

$$L(\theta) = \text{Avg}_i CE(y_i \sigma(f(x_i, \theta))) \qquad CE(p, \hat{p}) = -\sum_i p_i \ln(\hat{p}_i)$$

where x_i is an input sample, y_i is its associated event label and, θ are the parameters of the model. The function CE is referred to as the cross-entropy, and it measures how accurately the model is able to predict the true label of a given training sample. The cross-entropy takes as input p and $\hat{p}$ which represent the models guess, and the true sample labels resprectively. Finally, the function σ.

The utilization of cross-entropy is abundant across the deep learning field, and typically yields the strongest training results [5]. This loss function, however, fails to take into account the relative frequency of each sample. To highlight the consequences of this, consider a dataset which consists of 99% of one sample type, and the remaining 1% of another. Then the model could simply label *all* samples as the frequency dominant label, obtain 99% classification precision, and obtain an extremely low loss, likely only a local minima within the loss landscape. To account for this, we implemented a *weighted* loss function;

$$CE_{weighted}(p, \hat{p}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} w_c p_{i,c} \ln(\hat{p}_{i,c}) \tag{7}$$

where N is the total number of samples, and C is the number of classes. In our case, $C = 4$. Moreover, the weight $w_{i,c}$ is given by

$$w_c = \frac{N}{n_c C} \tag{8}$$

where n_c is the frequency of class c within the entire sample. This weighted cross-entropy loss function entices the model to pay more attention to minority classes or rare samples within the dataset during the training process. By assigning higher weights to underrepresented classes, the loss function effectively penalizes the model more for misclassifications in these classes compared to the majority classes. Consequently, the model is encouraged to improve its performance specifically on the minority classes, aiming to reduce the misclassification rate and better capture the patterns present in these less frequent samples. In essence, it helps address the issue of class imbalance by ensuring that the model does not disproportionately favor the majority classes and instead learns to generalize well across all classes, leading

to a more balanced and accurate predictive model.

4.1.2 Model Expansion

As discussed in Section 4.1.1, the NuGraph encoders consist of sequentially-applied linear layers and nonlinear tanh activations. This is explicitly displayed in Figure (2). The Encoder layer, which is directly used and trained for event classification, consists of only a single linear layer, encapsulating 3500 model parameters. In comparison to NexusNet, which is trained and optimized over 125,000 paramters, is extremely low. Having such a comparably lower number of parameters, the model is subject to under parameterization, meaning that there is not enough room for expresivity for the model to truly learn all underlying patterns that appear across the dataset.

In efforts to increase exressivity, we doubled the depth of the encoder layer by introducing an additional linear and nonlinear layer. In particular, the event encoder now consists of the following sequential layers;

$$z = \tanh(W_2(\tanh(W_1 x + b_1)) + b_2) \tag{9}$$

By doubling the depth of the encoder, the model gains additional parameters and increased complexity, enabling it to capture more nuanced features and relationships within the data. This approach aligns with the scalability principle observed in graph neural networks, where increasing the model's scale often leads to improved performance across various tasks, as documented in prior research [3]. Overall, expanding the depth of the model enhances its ability to learn and generalize from the data, ultimately improving its performance in event classification tasks.

Furthermore, deeper networks can facilitate more effective gradient propagation during the training process. Nearly all deep learning models are trained by adjusting the network parameters as to optimize the loss function over the loss landscape. This is achieve by iteratively applying the gradient descent algorithm. Networks which have a low number of parameters are highly subject to the vanishing gradient problem in which the model gets trapped in a local minimum or saddle point, rather than the global optimum. While shallow networks may suffer from the vanishing gradient problem, where gradients diminish rapidly as they propagate through successive layers, deeper architectures can resolve this issue by providing multiple pathways for error signals to flow backward through the network. This can result in more stable and efficient training dynamics, allowing the model to converge to better optima and achieve higher performance on the task at hand. In fact, the AdamW optimizer, which is the one utilized by NuGraph, is a gradient descent algorithm specifically designed to handle a large number of parameters; expanding the depth of NuGraph plays directly into the strengths of the chosen optimizer.

The AdamW optimizer offers several advantages that can be particularly beneficial in the context of deep neural networks and addressing the vanishing gradient problem. One key advantage is its adaptive learning rate mechanism, which dynamically adjusts the learning rates of individual model parameters based on their historical gradients. This adaptive learning rate scheduling allows AdamW to effectively solve complex optimization landscapes and converge to globally optimal solutions more efficiently, even in the presence of vanishing gradients.

4.1.3 Frequency-Balanced Batches

As discussed in Section (3), NuGraph batches data according to the `BalancedSampler`, which prioritizes maintaining the relative frequency of the entire dataset within each batch. For a batchsize of 32, this results in 7-8 muon and electrong neutrino samples, 1 or 2 tau neutrino samples, and the remaining being filled with neural current (NC) samples. As is displayed by the preformance of the original model, the model being exposed to far fewer tau neutrino samples has sever impact on its ability to identify them. Obtaining more tau samples is difficult, but also does not follow the true abundance of tau neutrinos in nature. In

order to yield strong event classification preformance on tau samples. we introduce a new sampling method.

To address the imbalance in the relative frequency of each class within the dataset, we employ a technique known as random oversampling [4, 2], specifically targeting the tau samples. We introduce a new sampler, called the `FrequencySampler`. By artificially suplementing the occurrence of tau samples during each training epoch, we ensure that the model encounters each electron, muon, and neutral current (NC) sample precisely once per epoch. In contrast, tau samples are presented to the model at a notably higher frequency, approximately seven times per epoch.

This approach strategically exposes the model to tau samples more frequently, in turn enhancing its ability to learn and generalize from this class of data. By increasing exposure to tau samples through oversampling, we aim to improve the model's training effectiveness on these smaples, ultimately seeking improvements in its performance during testing and validation phases.

4.2 Results

Throughout this project, we trained both the original, unmodified NuGraph network taken directly from the aforementioned Github, as well as numerous variations of this model. We successfully implemented each of the above modifications in isolation, allowing the model to successfully train.

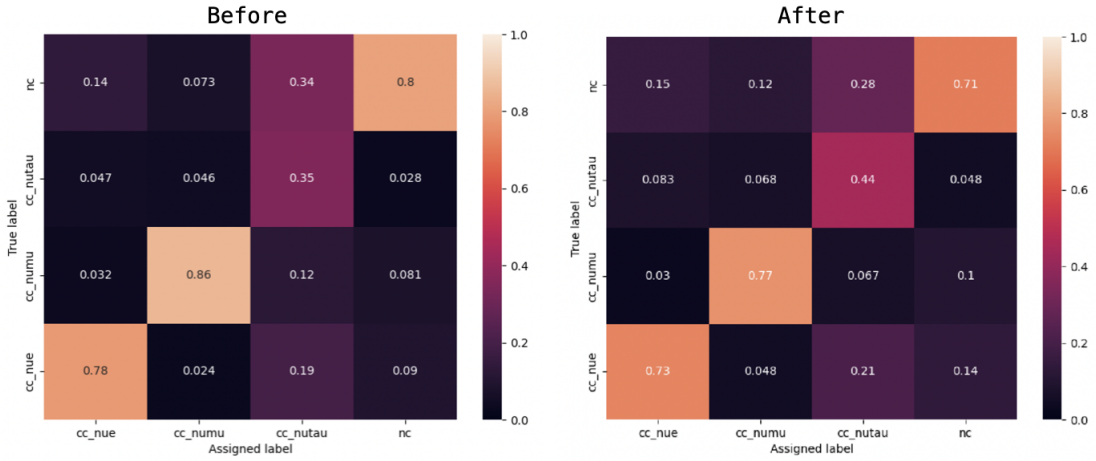

Figure 5: (Left) The precision matrix of the original model at the end of training. (Right) The precision matrix of the modified model after 25 training epochs. A model with perfect precision would have a unit anti-diagonal with all other entires being 0.

Included in Figure 5 is the precision matrix of the original model (left), and the modified model (right) after introducing the changes highlighted in Sections 4.1.1 and 4.1.2. In particular, make note of the anti-diagonal element of this matrix corresponding to tau samples being labelled correctly. The original model's results, which were observed at the *end* of training shows only a 35% when labelling tau samples. With only four classes to choose from, this does not show much better prefomance than simply randomly guessing (which would be represented by a prediction certainty of 25%). Moreover, the original model labelled tau samples as such almost precisely as often as labelling as a NC. This highlights the issues noted in Section 4.1.1 regarding the effect of having one class significantly dominate over another.

Following implementing the above modifications, the models' prefomance on tau samples has *significantly*. Specifically, after completely only 25 of the 80 epochs used to train the original model, we have observed a significant boost in preformance, with the model correctly classifying tau samples 10% more frequently.

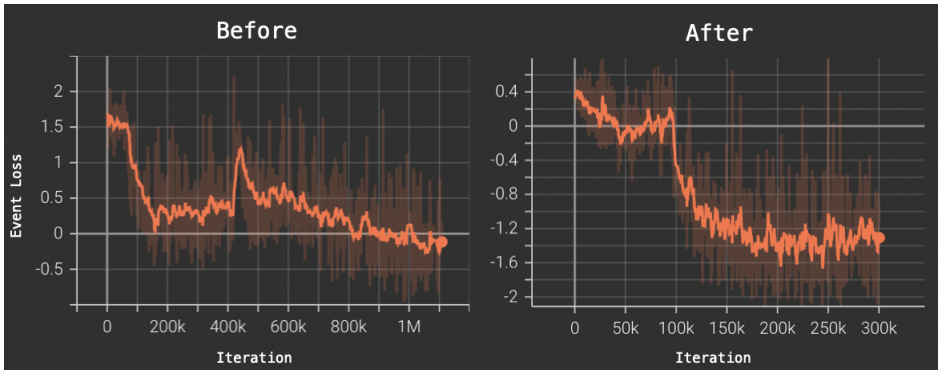

Figure 6: (Left) The loss of the original model. A model with perfect precision would have a unit anti-diagonal with all other entires being 0.

Figure 6 displays the loss over each iteration for both the original model (left) and the modified model (right). Steadily decreasing loss is inidcative that the methods implemented above indeed work with the existing NuGraph structure. Despite training for only a fraction of the original model's duration, the modified model exhibited a decreasing loss trend, indicating successful integration of the proposed methods within the NuGraph framework. Even with limited training time, the modified model achieved record minimum loss, indicative of high test performance.

Due to the considerable time and resources required for training this model, we were unable to conclude the training process after implementing the `FrequencyBatch` module. Nonetheless, initial training results suggest the method's effectiveness. With more training time, we are confident that we would have observed significantly improved overall performance, especially in classifying tau event samples. The challenges encountered in completing the training highlight the resource-intensive nature of training deep learning models, particularly with large datasets and numerous training epochs. Despite these challenges, the observed enhancement in model performance with the `FrequencyBatch` module underscores its potential for even greater impact with extended training.

5 Future Work

Moving forward, our focus will be on completing the training process with the `FrequencyBatch` module to fully assess its effectiveness. We'll also fine-tune model settings and explore ensemble learning methods to boost predictive accuracy and reliability. Additionally, we'll delve into advanced neural network designs tailored to our dataset and include domain-specific features to further enhance model performance. Ensuring that our model's predictions are understandable and deployable in real-world scenarios remains a priority, bridging the gap between theory and practical application.

Another promising direction for future work involves expanding the dataset by incorporating artificial data from ProtoDUNE. This expansion aims to balance the relative frequency of each class, thereby enhancing prediction performance. By continually exposing the model to a diverse range of samples from ProtoDUNE, rather than processing the same ones repeatedly, we can improve its ability to generalize and accurately classify tau neutrino events.

In summary, our contributions to the model could have the potential to significantly impact the DUNE team's ability to detect tau neutrinos following the completion of the entire experiment. By refining the model's training data and optimizing its performance, we aim to provide CERN and DUNE researchers with a powerful tool for uncovering valuable insights from the experimental data collected by DUNE.

References

[1] B. Abi et al. "Volume III. Introduction to DUNE". In: *JINST 15 T08009* (2020).

[2] Haseeb Ali et al. "A review on data preprocessing methods for class imbalance problem". In: (Oct. 2019), pp. 390–397. DOI: 10.14419/ijet.v8i3.29508.

[3] Ognjen Kundacina et al. *Scalability and Sample Efficiency Analysis of Graph Neural Networks for Power System State Estimation*. 2023. arXiv: 2303.00105 [cs.LG].

[4] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. "Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results". In: Apr. 2020, pp. 243–248. DOI: 10.1109/ICICS49469.2020.239556.

[5] Zhilu Zhang and Mert R. Sabuncu. "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels". In: *CoRR* abs/1805.07836 (2018). arXiv: 1805.07836. URL: http://arxiv.org/abs/1805.07836.