Progress Review

Theoretical Physics BSc Project

University College London

Rahul Appreciation-Learning Classifier Robustness in Neutrino Experiments

Author
I.J.S. Shokar - 17066988

Supervisor
Dr. C Backhouse

January 14, 2020

Introduction

The project I am working on is looking at the robustness of classifiers used to identify neutrino interaction events. The initial couple of weeks of the project were orientated around the literature review, and better understanding the neural networks that we will be looking at and the two Monte Carlo (MC) simulations, GENIE [1] and GiBUU [2], that produce the training data for the machine learning algorithms at the NOνA experiment, at Fermilab.

The classifier used for particle identification is a Convolutional Neural Network (CNN) [3], where the detector data is an image. The CNN used at NOνA can be seen as a 'black-box', as the weighing of feature maps cannot be interpreted by humans. Because of this, understanding what exactly the network is using to make its classification predictions cannot be extracted.

One concern with using a model that cannot be understood is that the network may be overfitting to any idiosyncrasies of the training models, rather than being general to model real interactions correctly. This would be a problem as the MC generators each model the events in a unique way, while neither of them perfectly model reality. By overfitting to these errors the model would not be able to generate well to other generators, or to detector data.

To determine whether the classifier is overfitting to the statistics of the model that it was trained on we are training and testing the CNN on data from two different generators, GENIE- which simulates the initial iteration of a neutrino with nuclei in the detector and the resulting scattered products to the surface of the nucleus and GiBUU- which simulates the evolution of a many-body system in the presence of potentials and a collision term, with the addition of neutrino-induced interactions.

Software

The initial part of the project involved becoming familiar with the various technologies that are being used to analyse the data at NOνA. The data and software used to analyse the data are stored on the High Energy Physics group's Linux cluster of machines, running Scientifc Linux, meaning I initially needed to understand the Linux command line in order to navigate the file system and software packages.

NOνASoft is a software package, developed by Fermilab, that builds on the ROOT data analysis software [4], developed by CERN. CafAna is the framework that NOνASoft uses to run functions and produce plots by specifying cuts on the data. The analysis framework is written in C++, a language I had no prior experience of, so a proportion of the time was spent trying to interpret pre existing C++ scripts in the NOνASoft, as well as the doxygen documentation.

Training data

The MC data is stored as ROOT files, as well as HDF5 files that contain the training data. The image data comes in the form of an array and the interaction targets in the form of a 'pdg' value. The event labels are stored as part of 'mc' branch of the HDF5 file, while the input images are stored in a 'training' branch.

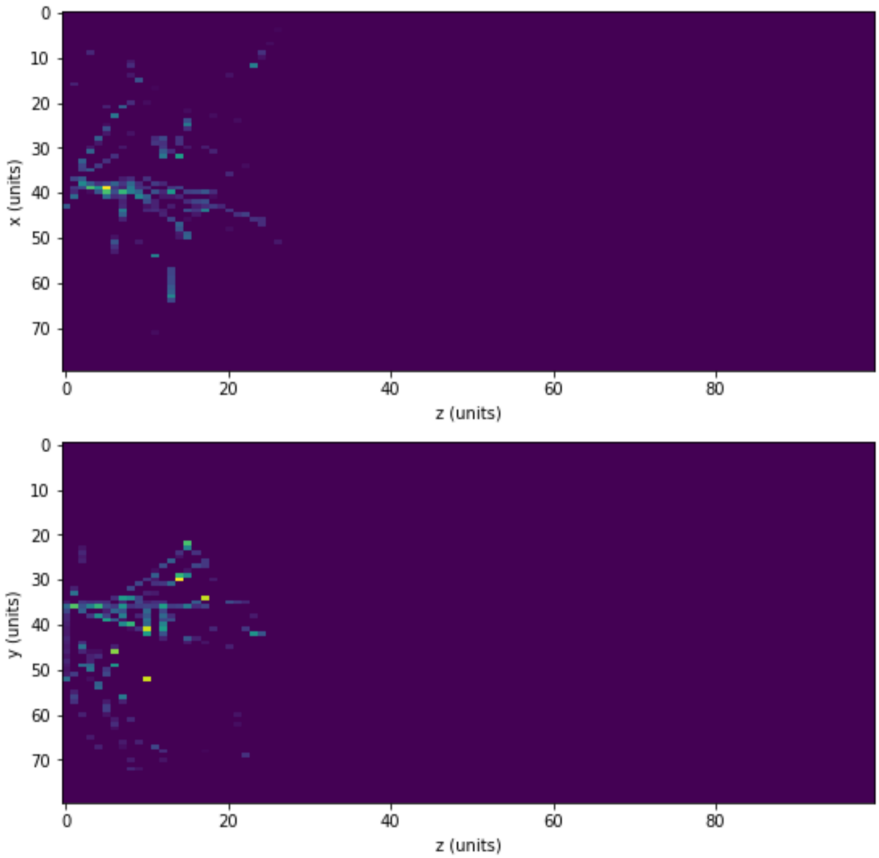

Figure 1. *Plot showing an event from the two planes of the detector (z-x and z-y) .*

I am assuming that the data stored in these HDF5 files has already undergone some sort of preselection cut as the number of subevents stored in the 'mc' branch is larger than that of the 'training' branch. Because of this, the events could not simply be matched up based on their index value, but on the following values: 'run', 'subrun', 'evt' (event), 'subevt' (sub event) and 'cycle' to indicate which subevent is being referenced.

After reshaping the image maps from a 1D array to two 100 x 80 pixel images representing the z-x and z-y planes of the detector (see figure 1 for plots of an event in the two detector planes), the images were written to a 4D tensor with dimensions:

- event index (determined by the order that the images were added),
- plane (z-x or z-y),
- z value,
- x or y value.

The HDF5 files also store data about each event, and this was used to apply the basic quality and preselection cuts. The cuts were outlined in the NOνA doxygen with the ν_μ cuts as kNumuQuality, kNumuContainFD2017 and the ν_e Cuts as 'kNue2017NDFiducial',

'kNue2017NDContain' , 'kNue2017NDFrontPlanes' , 'kNue2017NDNHits', 'kNue2018NDEnergy', 'kNue2017NDProngLength'.

Much like the the code already for the NOνA project and NOνAsoft, these are written in C++ and use the branch like structure of the ROOT files. A package has been developed for Python, called Pandana, however, I am unfamiliar with the syntax associated with the package and the documentation seems to focus on network analysis, so I deemed it a simpler option to simply implement the cuts using if statements with a loop over all the events.

While this meant that the computational process is much slower, I didn't see that as a problem as the cuts need only be applied once, for each generator, and the events that pass can be stored and then used as model inputs from then onwards. In order to verify that the cuts that I had made were correct, a function had been written in NOνASoft, 'MakeEventListFile', that listed the events that pass a specific cut, in order to be able to cross reference with the results I obtained.

It may be useful to learn and then implement the Pandana approach if more training data needs to be used in the future, or if the network was to be trained dynamically in the future.

While Keras does have a C++ API, as well as the option to export a model from Python, as Python is the language I am more comfortable with I decided that it was worthwhile interpreting the cuts and implementing them in Python rather than using the pre-existing C++ functions.

Models

The data that passed the preselection cuts, now in the form of image maps and labels indicating the interaction type, were now able to be passed to a model for training. Initially this was done on a smaller scale, using just 10 of the available 1150 files in order to ensure that all components were working properly. The interaction labels are one hot encoded into three classes: neutral current (NC), charged current (CC) ν_e (and $\overline{\nu_e}$) , CC ν_μ (and $\overline{\nu_\mu}$) as the classifier makes predictions for if an event belongs to each class.

Initially I produced a small network containing 6 sequential blocks containing convolution, max pooling and drop out layers in Keras. I did this to understand whether there were any errors in the formatting of the data as I was familiar with what was required for my rudimentary model. This worked with no hiccups, although the classifier was not particularly accurate.

The next stage was to run the small dataset using the MobileNet [5] architecture, developed by Google. Currently at NOνA the ResNet [6] architecture is being used, however the computing power available at NOνA is far greater to that available to me, meaning the training would take significantly longer. MobileNet is designed for implementation on mobile devices with a reduced computing power and thus reduced the training time significantly. Literature stated that MobileNet does experience a reduction in classification performance [5], however the increase in performance time is significant and more valuable to a project with a limited timeframe, which is looking to see if new methods can be implemented, as opposed to producing a final production model.

The network produces probabilities for each category indicating the confidence level that the network has in belonging to each label class. By taking the category that the model is

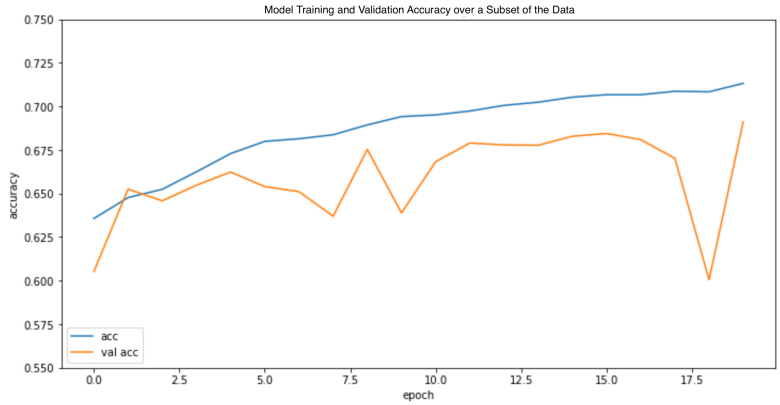

Figure 2. *Plot of the training (blue) and validation (orange) accuracies over 20 epochs .*

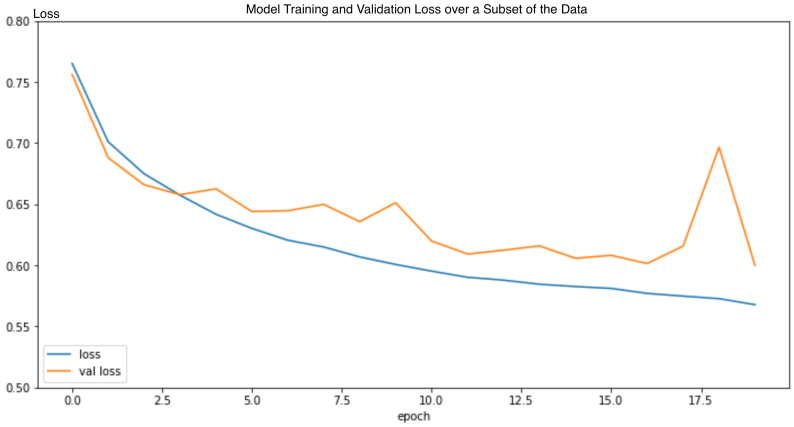

Figure 3. *Plot of the training (blue) and validation (orange) losses over 20 epochs.*

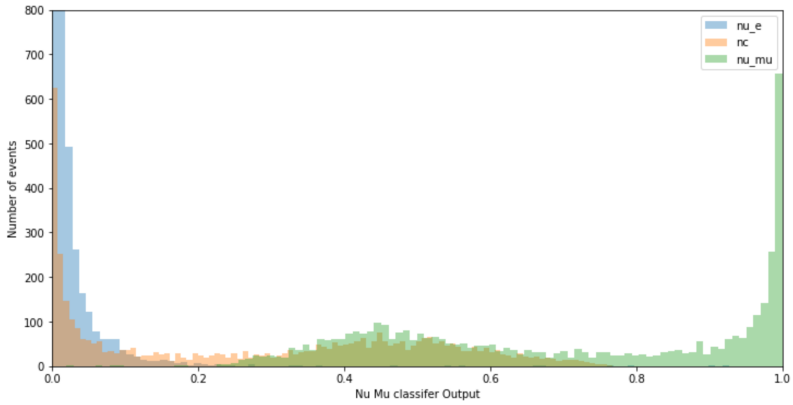

Figure 4. *Distribution in CC ν_μ Classifier output for ν_μ interactions (green), ν_e (blue) and NC interactions (orange).*

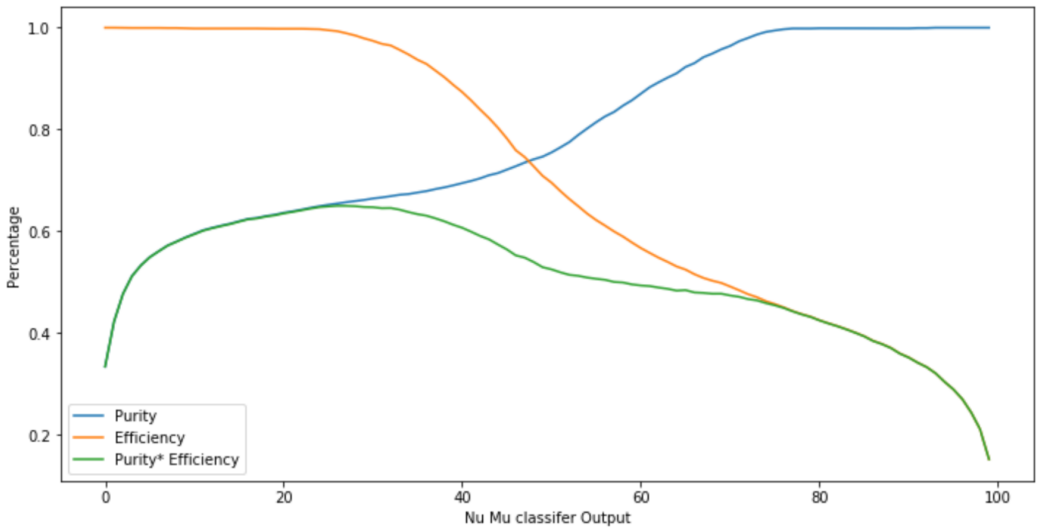

Figure 5. *CC ν_μ classifier efficiency (orange), purity (blue), and their product (green).*

most confident of we can compare this to the MC truth data to obtain a naive metric using a classification matrix.

When using the MobileNet architecture with the small subset of the data to access the performance we can look at the classifier output for truth CC ν_μ interactions, figure 4, and see that the classifier mostly identified ν_μ interactions correctly, however for many interactions it struggled to discern between CC ν_μ and NC interactions. The purity efficeincy plot in figure 5 can be used to compare other models directly to see when we have found the optimal model.

Generator Method

While it was possible to write the required data to an array when working with a relatively small dataset, this was not possible when using all of the available files in the directory, 1150 Genie HDF5 files, so a data frame was produced for the events that passed the cuts, storing the following data about the event, but not the image map itself:

- 'run' ,
- 'subrun',
- 'evt' (event),
- 'subevt' (sub event),
- 'cycle',
- 'train_index': location index within the 'training branch' of the image,
- 'label' : pdg interaction value,
- 'file': path to the relevant HDF5 file.

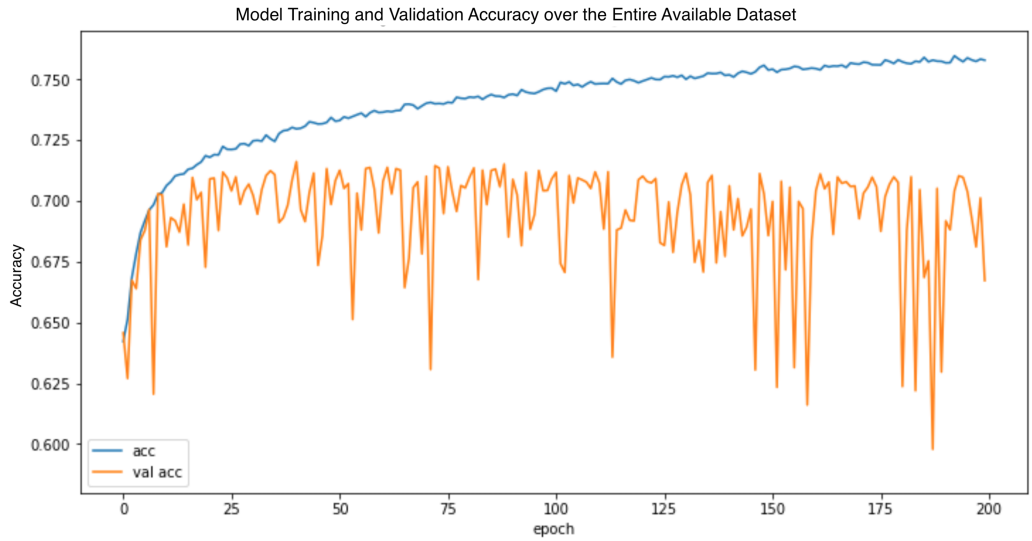

Figure 6. *Plot of the training (blue) and validation (orange) accuracies over 200 epochs .*

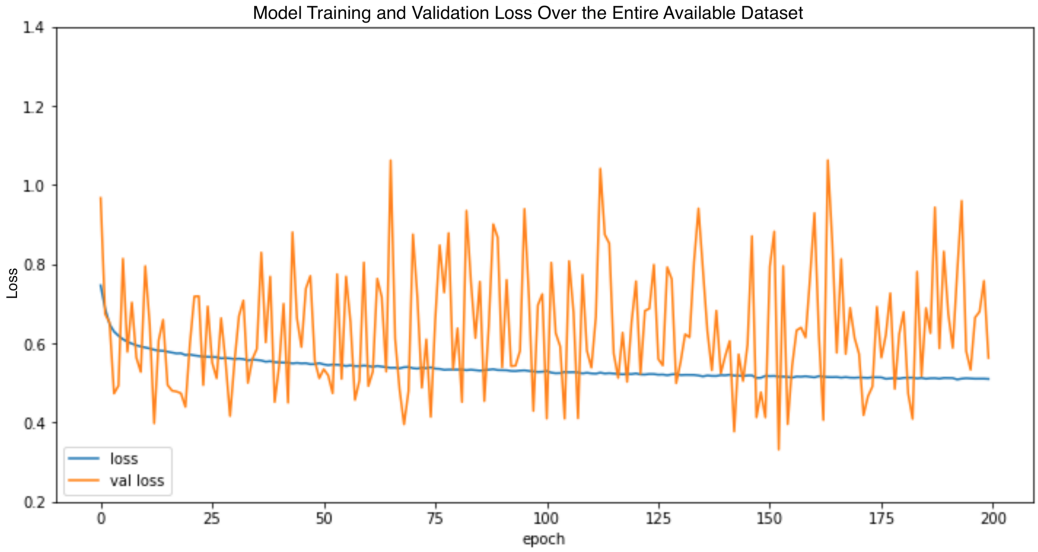

Figure 7. *Plot of the training (blue) and validation (orange) losses over 200 epochs.*

The data frames were then concatenated to produce three: one for training containing 335,456 sub events, one for training validation containing 67,092 sub events, and one for testing the model containing 93,248 sub events

As the data was not stored locally, the data could not be loaded into the model all at once, a generator method is used to load the data to the model in various batch sizes. This sources the image data from the relevant HDF5 file using the location stored in the 'file' column of the data frame and the 'train_index' value from that row to identify the array in question. This is then reshaped the same way as before and written to a 4D tensor identical to the one for the

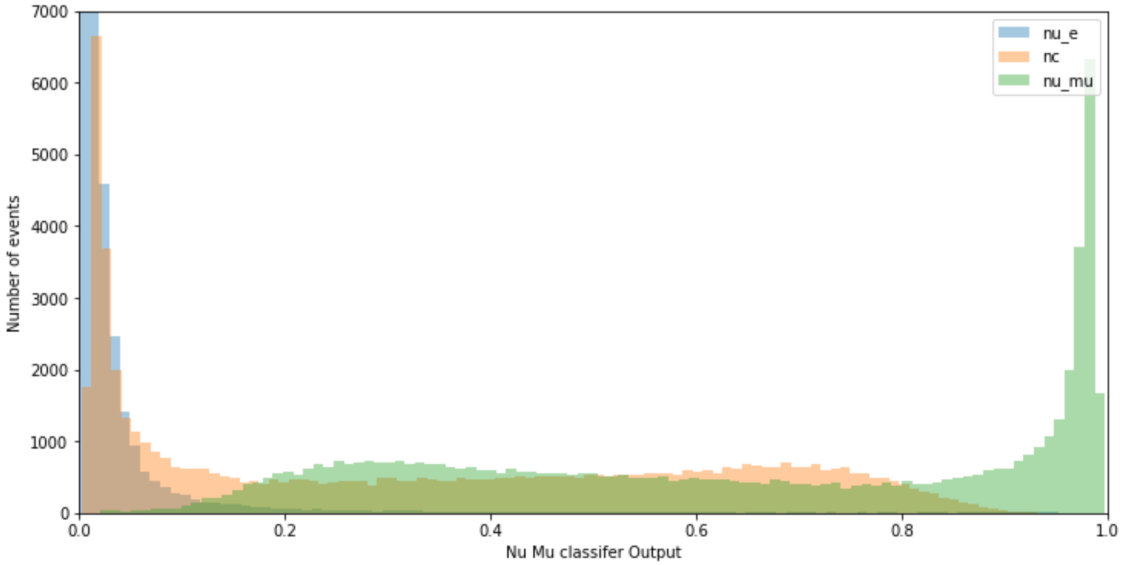

Figure 8. *Distribution in CC ν_μ Classifier output for ν_μ interactions (green), ν_e (blue) and NC interactions (orange) for the entire available dataset.*

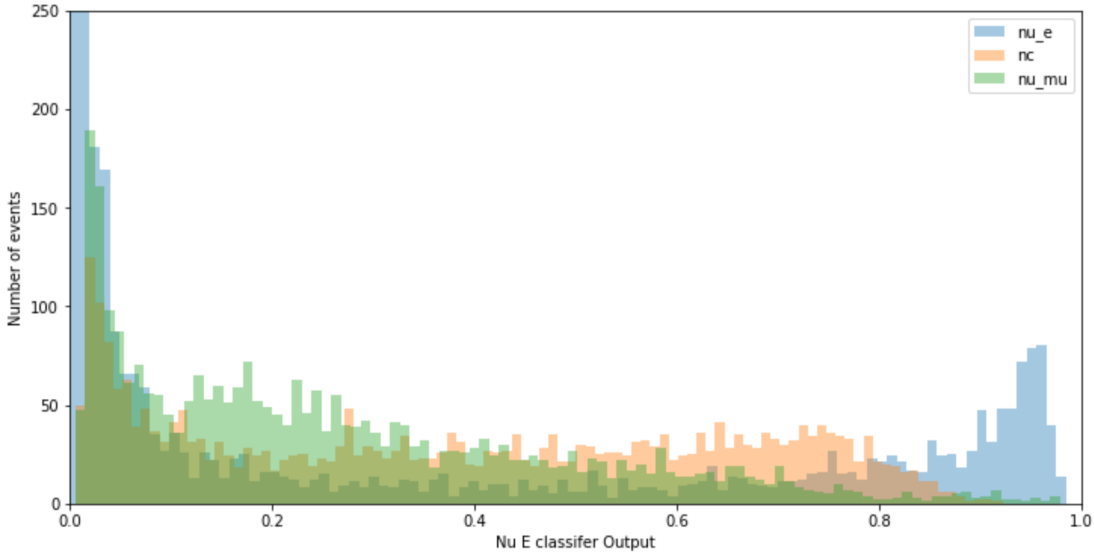

Figure 9. *Distribution in ν_e interactions (blue), ν_μ interactions (green), and NC interactions (orange) for the entire available dataset.*

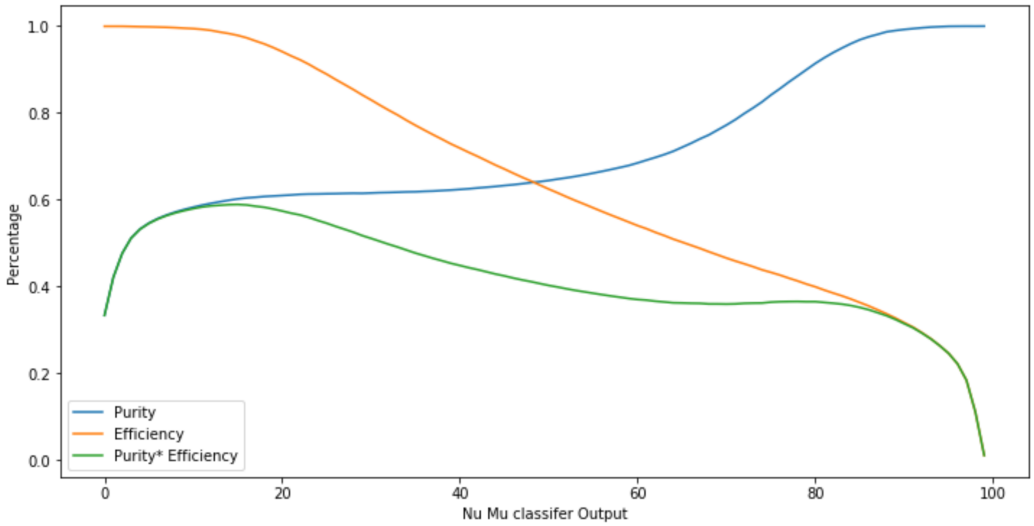

Figure 10. *CC ν_μ classifier efficiency (orange), purity (blue), and their product (green).*

entire dataset, with the only difference being the event index dimension is the size of the batch size that is specified, rather than the entire dataset. The default batch size is 32, as is the case with the results produced by NOνA [7]. The target values are again one-hot encoded based on the 'label' value for the row. This generation takes place for the training and validation data frames, and the network is trained, taking a number of days, before inference on the test data.

When training the model on all the available GENIE results we can see that the classification of the particles is still relatively poor. This can be seen from both figures 6 and 7 that show the validation accuracy and loss oscillating dramatically, as well as the classifier distributions in figure 8 which, surprisingly, performs worse than the model trained on only 10 of the GENIE files at identifying between between CC ν_μ and NC interactions for truth ν_μ interactions. The purity efficiency plot in figure 10 confirms this poorer performance with the product curve closer to the origin than that of figure 5. As well as this the classification of truth ν_e interactions, shown in figure 9, are very poor, with the majority of the interactions given a near zero confidence of being a ν_e interaction. The hyper parameters of this initial model training are clearly not optimal.

Term 2 Plan

Initially the first few weeks of term will look at optimising the current network, to find the hyper parameters that yield the best results. As seen from the results produced, the test loss varies significantly and a variable learning rate optimiser, such as 'Adam' as well as a momentum term may avoid the function finding local minima.

Then we will look at applying the network, training and testing process, to the GIBUU simulations. These vary slightly as the GIBUU training needed to be weighted, meaning finding

a way to implement this in Keras.

By February, we should be in a position to see if we can increase the robustness of network using data from both domains together. With the next stage to apply to Domain Adversarial [8] training using both simulations. I would hope that by the end of February we would be in a stage to apply the domain adversarial training to the detector data, and be able to analyse the results to see if any improvements in accuracy over various domains are made, before writing up the project and results for the end of March deadline.

References

[1] C. Andreopoulos , A. Bellb, D. Bhattacharyab et al. *The GENIE Neutrino Monte Carlo Generator*, 2009.

[2] O. Lalakulich, K. Gallmeister, U. Mosel et al. *Neutrino nucleus reactions within the GiBUU model*, 2011.

[3] C. Szegedy, C. Hill , Y. Jia et al. *Going deeper with convolutions*, 2014.

[4] R. Brun, F. Rademakers. *ROOT - An object oriented data analysis framework*, 1997.

[5] A. Howard, M. Zhu, B. Chen et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017.

[6] K. He, X. Zhang, S. Ren et al. *Deep Residual Learning for Image Recognition*, 2015.

[7] A. Aurisano, A. Radovic, D. Rocco et al. *A Convolutional Neural Network Neutrino Event Classifier*, 2016.

[8] Y. Ganin, E. Ustinova, H. Ajakan et al. *Domain-Adversarial Training of Neural Networks*, 2016.