

# MicroBooNE-NOTE-1019-PUB

## Convolutional Neural Networks Applied to Neutrino Events in a Liquid Argon Time Projection Chamber

MicroBooNE Collaboration

July 4, 2016

### **Abstract**

We present several studies of convolution neural networks applied to data coming from the MicroBooNE detector, a liquid argon time projection chamber (LArTPC). The algorithms studied include the classification of single particle images, the localization of single particle and neutrino interactions in an image, and the detection of a simulated neutrino event overlaid with cosmic ray backgrounds taken from real detector data. The purpose of these studies was to demonstrate the potential of these networks for particle identification or event detection with simulated neutrino interactions and to address a number of technical issues that arise when applying this technique on data from a large LArTPC located near the surface. The results of these studies can be used to guide similar applications on detector neutrino data. We developed and validated techniques and approaches that demonstrate successful application of these networks for particle identification or event detection on simulated data and can be used to guide similar application on detector data.

# 1 Introduction

This work explores the application of a machine learning algorithm, convolution neural networks (CNNs), to event reconstruction in neutrino scattering experiments which utilize liquid argon time projection chambers (LArTPCs). Our case-in-point is the MicroBooNE experiment located at Fermi National Accelerator Laboratory (FNAL). MicroBooNE is the first large-scale LArTPC detector running in the United States. The experiment has a total mass of 170 tons and an active volume of  $2.3 \times 2.5 \times 10.4 \text{ m}^3$ . The system consists of two subdetectors: a time projection chamber (TPC) for tracking and a light collection system based on photomultiplier tubes. In October 2015, the detector began observing interactions of neutrinos coming from the FNAL Booster Neutrino Beam (BNB).

LArTPCs are state-of-the-art detectors of much interest to the neutrino community because of their capability to produce high resolution images of neutrino interactions (as will be described in section 2.1). Such images open the door to new high-precision studies that rely upon differentiating single electrons traversing the detector from photons that convert to an electron/positron pair in the detector. Because of this great promise for precision studies, MicroBooNE is the first detector in a series of LArTPCs planned for the future FNAL neutrino program. The next generation will include the SBND [1], ICARUS-T600 [2] and DUNE [3] detectors.

Performing physics analyses with LArTPC data depends upon accurate categorization of the tracks produced by charged-particles that travel through the TPC. Such categorization includes sorting tracks by particle type, deposited energy, and angle with respect to the known incoming neutrino direction. In the case of MicroBooNE and the future SBND and ICARUS-T600 experiments, all of which are located near the surface, this task is complicated substantially as tracks must further be classified into those produced by a neutrino interaction or of those produced by cosmic rays.

Because MicroBooNE is the first multi-ton LArTPC running on the surface in a neutrino beam and is only the second hundred-ton-scale LArTPC to take data, event reconstruction represents relatively uncharted territory. Many approaches are being applied. In this work, we present an approach based on CNNs.

This work addresses the following technical questions related to applying CNNs to the study of LArTPC images:

- Do CNNs, which have been developed for information-dense natural images, work for LArTPC event images, which are sparser?
- Current networks are designed to work with images that are smaller (approximately  $300 \times 300$  pixels) than those produced by a large LArTPC (approximately  $3000 \times 9600$ ), such as MicroBooNE. How does one balance the desire to use as high a resolution as possible with the constraints on image-size imposed by current hardware limitations? What is a possible scheme to downsize the image?
- And with a downsizing scheme in place, what is its effect on the network performance?
- How does one coordinate the multiple views of an event that LArTPC detectors produce?

We document a number of strategies to overcome the challenges listed above.

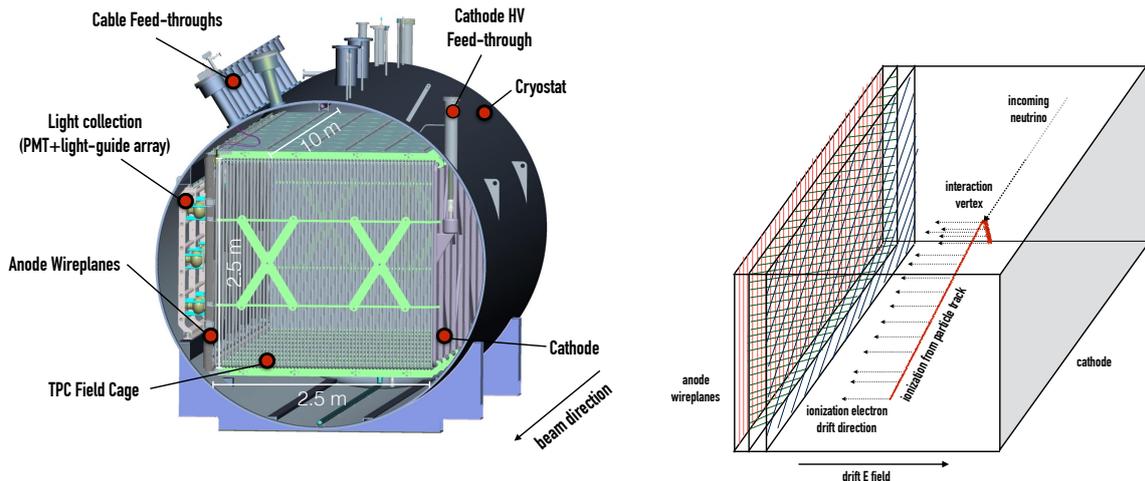


Figure 1: *Left.* An illustration of the MicroBooNE detector. Various components of the detector are labeled. *Right.* Cartoon illustrating the operation of the TPC. The TPC consists of a cathode plane and anode wireplanes. The cathode is biased to  $-70$  kV which places an electric field between the planes. When a neutrino interaction occurs in the TPC, often times charged particles are emitted, which travel across the detector and ionize liquid argon. The ionization electrons then drift towards the anode wire planes where the position and amount of ionization is measured.

We first present a brief introduction to the MicroBooNE LArTPC and then to CNNs. We then discuss the application of CNNs to LArTPCs in general. This is followed by three demonstrations that make use of the MicroBooNE simulation and cosmic ray data. The demonstrations use publicly available, open-source software packages.

## 1.1 The MicroBooNE Detector

MicroBooNE employs as its detector, a large liquid argon time projection chamber. Figure 1 shows a diagram of the MicroBooNE LArTPC, which is typical of most beam-based LArTPC experiments. The detector consists of a cryogenic vessel which houses the TPC and the light collection system, the latter an array of photomultiplier tubes (PMTs) and four acrylic light-guides. The TPC structure is composed of a rectangular, stainless-steel cathode plate set parallel to three sets of rectangular, anode wireplanes. The distance between the cathode plate and the closest wireplane is  $2.5$  m. The height and length of the cathode and anode wireplanes are similar,  $2.3$  m high and  $10.4$  m long. The space between them define the box-shaped, active volume of the detector. We describe the MicroBooNE TPC with a right-handed coordinate system where the  $Z$ -axis is aligned with the long dimension of the TPC and the direction of the neutrino beam. The beam travels in the  $+Z$  direction. The  $X$ -axis runs normal to the wire planes, beginning at the anode and moving in the direction of the cathode. The  $Y$ -axis runs parallel to the anode and cathode, in the vertical direction. The PMT array is mounted to the inside cryostat wall closest to the anode wireplanes. The phototubes are set behind the wireplanes, outside the space between the cathode and anode, and face towards the center of the TPC volume in the  $+X$  direction.

The TPC and PMT system together provide the data necessary to reconstruct the tracks of charged particles traveling through the TPC. Figure 1 contains a cartoon illustrating the basic operating principles of the TPC, which we describe here. Charge particles traversing the liquid argon produce ionization, and the resulting electrons drift to the TPC wire chambers over a time period of milliseconds due to an applied high voltage, -70 kV, on the cathode plate. The TPC wire chamber is located on the  $-X$  side of the detector and is composed of three planes of wires, which provides three views of the ionization pattern in the detector. For all three planes, the wires lie parallel to the  $YZ$  plane. The plane furthest from the TPC volume are oriented vertically along the  $Y$ -direction and serves as a collection plane, i.e., the electric field terminates on these wires, meaning that the ionization electrons collect on these wires. This plane is called the  $Y$  plane. The other two planes, called  $U$  and  $V$ , observe signals by induction as the electrons travel past them before being collected on the  $Y$  plane. The  $U$  and  $V$  plane are oriented  $\pm 60$  degrees with respect to the  $Y$ -axis. There are 2400 wires that make up the  $U$  and  $V$  planes, and 3456 wires that make up the  $Y$  plane. Electronics on each wire record the charge and drift-time associated with hits on a wire. Scintillation light is produced as well and can be observed by the PMTs within nanoseconds of the interaction time. Thus, the light tells us the time when a track passes through the detector. Because the drift speed of electrons in the TPC is measured,  $0.11 \text{ cm}/\mu\text{s}$  at the 273 kV/cm operating field [17], the time elapsed between the observation of the light and the observation of the charge at the wires provides the position of charge depositions in  $X$ . The scintillation light also is used to build a prompt event trigger.

An event in the detector is defined as a 4.8 ms period during which waveforms from the TPC wires and the PMTs are recorded. This period is a little over twice the time it takes for electrons to drift from the cathode to anode. This allows the electronics to capture one full drift period around some moment of interest, plus half a drift period before and after. The moment when the electronics begins to capture an event is initiated by a trigger. Triggers are sent to record events both in time with the neutrino beam and out of time, the latter used to record cosmic-ray only events for background studies. The TPC front-end electronics consist of custom ASICs capable of operating at liquid argon temperature. The ASICs amplify and shape the wire signals with a configurable shaping time set to 2 microseconds. Downstream of the ASICs, digitizers, located outside cryostat, sample the amplified wire signal at a frequency of 2 MHz, or for a total of 9600 samples per event, and output digital waveforms for each wire over an 12-bit range. The PMTs are readout by a separate set of front-end electronics with a 60 ns shaping time. The PMT signals are digitized at 64 MHz. An unbiased-readout window of 23.4 microseconds around the time of the event trigger is recorded from all the PMTs.

During an event, cosmic ray particles create dozens of tracks throughout the event window and different parts of the TPC. For those events recorded in time with the neutrino beam, some fraction capture a neutrino interaction as well. Many of these interactions will produce charged particle tracks ejected from the interaction vertex. One of the primary challenges of reconstruction will be to select these neutrino interaction tracks out of all the cosmic ray particle tracks. And once identified, the constituent neutrino interaction tracks must each be identified as a particle type, only then can information about the interaction neutrino, its flavor and energy, be inferred. This is the challenge of reconstruction in LArTPCs. However, because of the high frequency sampling of the detector by the TPC electronics, the data produced has a photographic quality, similar to images of bubble chamber events. This motivates the use of convolutional neural nets, a

type of artificial neural net, which because of their architectures, have had much success in parsing the information in images.

## 2 Background on Convolutional Neural Networks

Convolutional neural networks (CNNs) can be thought of as a modern advancement on feed-forward neural networks (FFNNs), the latter a commonly employed technique in high energy physics analyses. CNNs can be seen as a special case of FFNNs; one designed to deal with spatially localized data, like those found in images. The core principle behind the development of CNNs was to restrict the number of learnable parameters, which decreases the difficulty of training. This allows for more complicated network architectures, including operations that go beyond those performed by the networks' individual neurons. These expansions have allowed CNNs to become the state-of-the-art solution to many different kinds of problems, most notably photographic image classification. CNNs have even begun to find uses in other neutrino experiments [5][6].

Consider event sample classification, a typical analysis performed by FFNNs in high-energy physics. Here the goal is to assemble a set of  $N$  features, calculated for each data event, and use them to separate the events in one's data into  $C$  classes. For example, one might aim to separate the data into a signal sample and a background sample. For many experiments, a common approach to reconstruction is to take raw data from various instruments and distill it into several reconstructed quantities, often focused around characterizing observed particles, that can be used to construct these  $N$  features. This information can then be used to train an FFNN. However, the process of developing reconstruction algorithms can often be a long, complicated task. In the case of LArTPCs, we can employ CNNs, which will do much of this work automatically by learning its own set of features from data. This ability of CNNs is due to its architecture which differs substantially from that of FFNNs.

In FFNNs, the input feature vector,  $\vec{x}$  is passed into one or more hidden layers of neurons. These hidden layers eventually feed into an output layer containing  $C$  neurons, which together produce a  $C$ -dimensional output vector,  $\vec{y}$ . Figure 2 contains a diagram illustrating a possible network architecture for a FFNN. One characteristic feature of these networks is that each layer is “fully-connected” meaning that each neuron receives input from every neuron in the preceding layer. In principle, one has the freedom to define what each neuron does with this input. In practice, however, it is common for the neuron to perform a weighted sum of the inputs and then pass this sum into something known as an activation function. In other words, for a given neuron, whose label is  $i$ , and an input vector,  $\vec{x}$ , the output of the neuron,  $f_i(\vec{x})$  is

$$f_i(\vec{x}) = \sigma(\vec{w}_i \cdot \vec{x} + b_i), \quad (1)$$

where  $w_i$  are known as the weights of neuron,  $i$ ,  $b_i$  is the bias of neuron  $i$ , and  $\sigma$  is some choice of activation function. Common choices for  $\sigma$  include the sigmoid function or tanh. Currently, most CNN models use an activation function known as the Rectified Linear Unit, or ReLU. Here  $\sigma_{\text{ReLU}}$  is defined as

$$\sigma(\vec{x}) = \begin{cases} \vec{w}_i \cdot \vec{x} + b_i & \vec{w}_i \cdot \vec{x} + b_i \geq 0 \\ 0 & \vec{w}_i \cdot \vec{x} + b_i < 0. \end{cases} \quad (2)$$

The way these networks learn is that one chooses values of the weights and biases for all neurons such that, for a given  $\vec{x}$ , the network produces some desired,  $\vec{y}$ . Going back to the signal vs. background example, the goal is to choose values for the network parameters such that, for a given event, the FFNN is able to map an  $N$ -dimensional feature vector

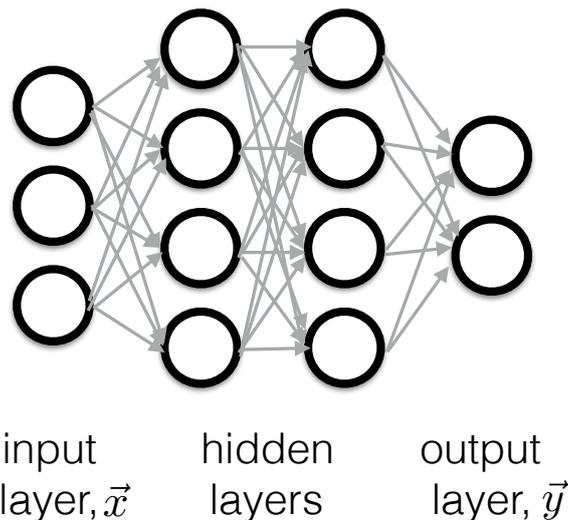


Figure 2: Feed-forward neural network architecture. The network is arranged in layers. There is an input layer consisting of a  $N$ -dimensional vector,  $\vec{x}$ , of features. In this case we have  $N = 3$ . There are one or more hidden layers. In this diagram, two are shown. Finally, there is the output layer, a  $C$ -dimensional vector,  $\vec{y}$ , with  $C$  commonly chosen to be the number output classes. In this case we have  $C = 2$ .

of reconstruction quantities,  $\vec{x}$ , to  $\vec{y} = (1, 0)$ , if the event is a signal event, or  $\vec{y} = (0, 1)$ , if the event is a background event. For CNNs, the way neurons are connected is different.

For CNNs, sets of hidden neurons are still grouped together in layers. However, a neuron in a given layer is only asked to receive the inputs of a local region of neurons in the preceding layer. Instead of an  $N$ -dimensional input vector, the input for a given neuron is laid out as a  $M \times N \times L$  tensor. This is a natural representation of the pixel values in an image. For an image with height,  $M$ , and width,  $N$ , the third dimension is the number of channels, e.g. three color channels for red-green-blue (RGB) images. The output of a given neuron looking at a local volume of pixel values centered around pixel,  $(m_i, n_i)$ , of input tensor,  $X$ , is given by

$$f_i(X) = \sigma\left(\sum_{m=m_i-K}^{m_i+K} \sum_{n=n_i-K}^{n_i+K} \sum_l^L W^{mnl} X_{mnl} + b\right), \quad (3)$$

where the summation is over all channels,  $L$ , and over  $-K$  to  $K$  elements in the height and width dimensions around the input pixel element with index  $(m_i, n_i)$ . Figure 3 illustrates this connection structure. In the figure, a neuron is receiving input from a local volume, highlighted in gray, of the input tensor, represented by the 3D grid on the left. The volume is centered around the input pixel,  $(m_i, n_i)$  highlighted in red. In a CNN, this neuron acts only on one sub-region at a time, but, in turn, it acts on all sub-regions of the input as it is scanned over the height and width dimensions in steps called “strides”. During this scan, the output values of the neuron are recorded and arranged into a 2D grid. The grid of output values, therefore, forms another image known as a feature map. In figure 3, this feature map is represented by the right-most grid.

The neuron in this architecture acts like an image filter. The value of the weights of the neuron can be thought as representing some pattern. For example, patterns might

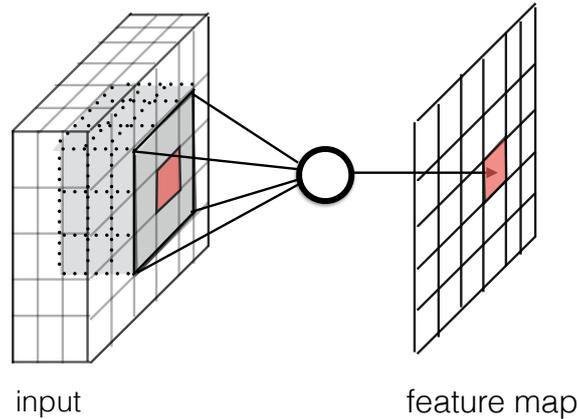


Figure 3: The input and output connections of one neuron in a convolutional neural network (CNN). The neuron looks at a local volume of pixel values arranged in an  $M \times N \times L$  tensor. The input consists of a  $M \times N$  image with  $L$  color channels, represented by the 3D grid on the left. In a CNN, one such neuron acts only on one sub-region at a time but, in turn, it acts on all sub-regions of the input as it is scanned over the height and width dimensions. By arranging the outputs of the neuron as it is being scanned across into a 2D grid, one forms a feature map, represented by the right most grid.

include different orientation of edges or small patches of color. If the local region of the input matches that pattern, the activation is high, otherwise the activation is low. Having scanned the feature across the input, the resulting output is an image indicating where that pattern can be located. In other words, the pattern has been filtered into the output image. By allowing  $F$  such neurons, or filters, to process the input, one gets  $F$  feature maps which can be arranged to make another tensor. Also, note that the operation of scanning the neuron across the input is a convolution between the input and the filter. The collection of filters that operate on an input tensor and produce a set of feature maps is known as a convolutional layer.

Like FFNNs, one can arrange a CNN as a series of layers where the output of one convolutional layer is passed as input to another convolutional layer. With such a stack of layers, CNNs learn to identify high-level objects through combinations of low-level patterns. Furthermore, because a given pattern is scanned across the image, translation-invariant features are identified. For example, one can find a vertex by locating edges of various orientations near one another.

This architecture has a couple of advantages. First, the parameters of the neurons, or filters, are learned. Like FFNNs, training the network involves an algorithm which chooses the parameters of the neurons such that the network learns to map an input image to some output number or vector. This ability to learn its own features is one of the reasons CNNs are so powerful and have seen such widespread use. When approaching a new problem, instead of having to come up with a set of features to extract from the data, with CNNs, one can simply collect as much data as possible and start training the network. Second, note that the number of parameters per layer scales like the volume of the input region. This can be much smaller than the number of parameters per layer of an FFNN, especially when considering the number of parameters it would take to use an unrolled image as input to the network. This efficient use of parameters allows CNNs to be composed of many, successive convolutional layers. The more layers one uses, the more

complex an object can be represented. And the current understanding from research into CNNs is that "deep" networks with a very large number of layers (on the order of one hundred or more) tend to work better than shallow networks.

## 2.1 Application in LArTPCs

Application of a CNN is relatively straightforward for LArTPCs, because the data they produce is essentially a set of images containing charged particles' energy depositions and trajectories projected on a 2D plane view. The two axes of an image are the wire number and readout drift time. The first dimension moves across the detector, upstream to downstream in the neutrino beam direction, while the second dimension is a proxy for the  $X$  direction<sup>1</sup>. In MicroBooNE, anode wires are separated by a 3 mm pitch and carry a signal waveform digitized at a 2 MHz frequency and presently set with a 2  $\mu$ s shaping time. For a full-size image, one pixel corresponds to 0.55 mm along the time axis given the measured drift velocity of 0.11 cm/ $\mu$ s. The pixel values of the image represent the analog-to-digital-converted (ADC) charge on the wire at the given time tick. ADC counts are encoded over a 12 bit range. This scheme can produce high spatial resolution images for each wire plane with good charge resolution as well. In figure 4, we show two images (in false color) of cosmic rays viewed by the  $Y$  plane (Hence showing a  $Z$  vs.  $X$  projection) as examples of the high quality information obtained by an LArTPC. The task of reconstruction is to parse these images and determine information about the neutrino interaction that produced the tracks observed.

The attractiveness of the Deep Learning approach is reduced need for hand-engineered pattern recognition algorithms for the reconstruction of LArTPC images. Likely, different algorithms will be required for specific types of physics processes. Such algorithms can take many person-hours to develop and then optimize for efficient computation. Instead, Deep Learning potentially offers a different strategy, one in which networks are left to discover its own patterns to recognize the various processes and other physical quantities of interest. Furthermore, because GPUs have been optimized for this type of image processing, analyzing with a CNN takes on the order of tens of ms per event. However, this approach is still relatively unexplored, and so in this work we describe three studies or demonstrations applying Deep Learning to LArTPC images.

---

<sup>1</sup>This assumes a constant electron drift velocity. Space charge leads to distortions of this ideal and must be corrected for.

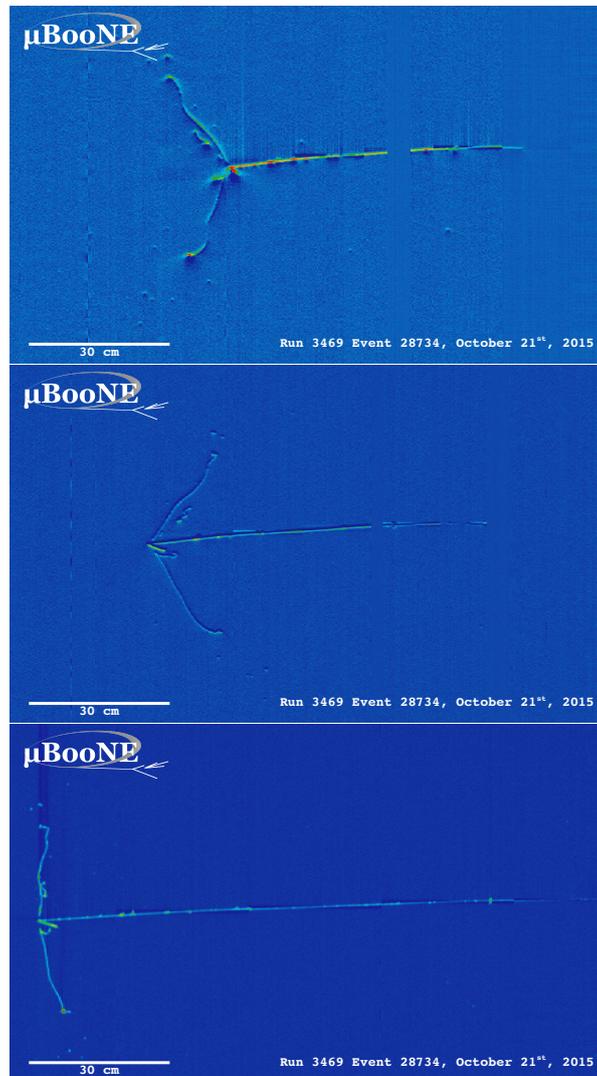


Figure 4: Example neutrino interaction observed in the MicroBooNE detector. The same interaction is shown in all three wire plane views. The top image is from wire plane  $U$ ; the middle from wire plane  $V$ ; and the bottom from wire plane  $Y$ . The image is at full resolution and is only from a portion of the full event view.

Table 1: A summary of the publicly-available code used in each of the demonstration exercises.

Software	ref.	Purpose	Used in Demonstrations
LArSoft	[22]	Simulation and Reconstruction	1-3
uboonecode	[25]	Simulation and Reconstruction	1-3
LArCV	[27]	Image Processing and Analysis	1-3
Caffe	[36]	CNN Training and Analysis	1-3
AlexNet	[14]	Network Model	1,2
GoogLeNet	[15]	Network Model	1
Faster-RCNN	[16]	Network Model	1,2
Inception-ResNet-v2	[32]	Network Model	2
ResNet	[33]	Network Model	3

## 2.2 Demonstration Steps

We demonstrate the applicability of the Deep Learning technique through the following tests:

- Demonstration 1– Classification and detection of a simulated single particle in a single-plane image;
- Demonstration 2– Neutrino event classification and interaction localization within a single-plane image;
- Demonstration 3– Neutrino event classification with a full 3 wire-plane model.

The first study shows that a CNN can be applied to LArTPC images despite the fact that its nature is quite different from the photographic images for which the technique is developed. In this study we crop and downsize a sub-region of a whole event image which is an important step for a large LArTPC detector such as MicroBooNE due to computing resource limitations. The second study is an initial trial for localizing neutrino interactions within a 2D event image using only one plane of the three views. In the last case, we construct a network model that combines full detector information including all three views and optical data, using a data augmentation technique.

## 2.3 CNNs Models and Software Used

For the demonstrations performed in this work, we use several prevalent networks that have been shown to perform well at their given image processing task, typically measured by having nearly the best performance in some computer vision competition at the time of publication. Our implementation of the networks and the analysis of their outputs uses open-source software along with some custom code that is publicly available. We summarize the networks and software we use in table 1. For demonstration 1 we used the AlexNet [14] and GoogLeNet [15] for image the classification task. Demonstration 2 introduces a simplified Inception-ResNet-v2 by C. Szegedy et al. [32]. For the demonstration 3, we have designed our own CNN based on ResNet [33]. Demonstrations 1 and 2 involve a network that can locate an object within a sub-region of an image (“Region of

Interest,” or ROI, finding). For this task we used a network known as the Faster-region convolutional neural network, or Faster-RCNN [16].

All of the above are CNNs with various depths of layers. The choice of AlexNet for demonstrations 1 and 2 is motivated by the fact that this relatively shallow model is often used as a standard in the field of Deep Learning ever since it was first introduced for the annual Large Scale Vision Recognition Challenge (LSVRC) in 2012. GoogLeNet, which has 22 layers compared to the 8 layers in AlexNet, is another popular model which we employed to compare against AlexNet in demonstration 1.

For ROI-finding within an image, we chose Faster-RCNN [16] because it is a prevalent and state-of-the-art object detection network for determining the spatial location of objects inside an image. This network localizes individual instances of some object of interest within an image. It is capable of identifying multiple instances of different classes within the same image.

We also use truncated versions of two networks, ResNet [33] and Inception-ResNet-v2 [32], to perform neutrino event classification in demonstration 2 and 3. Both networks use a type of sub-network architecture known as residual convolutional modules [33]. These modules are believed to help the network achieve higher accuracy and learn faster. (For a description of a residual model, see ref. [33] or Appendix A for a brief introduction.)

Throughout all studies in this work, we used one of the most popular open-source CNN software frameworks, Caffe [36], for CNN training and analysis. Input data is in a ROOT file format[4] created by the LArCV software framework[27], which we developed to act as the interface between the LArSoft and Caffe. LArCV is also an image data processing framework and is used to further process and analyze images as described in the following sections. One can find our custom version of Caffe that utilizes the LArCV framework in [28].

The computing hardware used in this study involves a server machine equipped with two NVIDIA Titan X GPU cards[29], chosen in part due to their large amounts of on-board memory (12 GB). We have two such servers of similar specifications at the Massachusetts Institute of Technology and at Columbia University, which are used for training and analysis tasks.

### 3 Demonstration 1: Single Particle Classification and Detection

In this demonstration, we investigate the ability of CNNs to classify images of single particles located in various regions of the MicroBooNE detector. This demonstration also outlines a strategy to deal with a technical problem for both this task and others in large LArTPCs, like MicroBooNE. The issue is as follows. The detector creates three images per event, one from each plane, which at full resolution has dimensions of either 2400 wires  $\times$  9600 digitized time-ticks for the induction planes, U and V, or 3456 wires  $\times$  9600 time-ticks for the collection plane, Y. Future LArTPC experiments will produce similar size or even larger images. Such images, as is, are too large to use for training a CNN of reasonable size on any GPU card currently available in the mainstream commercial market. To mitigate this problem one must crop or downsize the image, possibly both.

Therefore, in Demonstration 1, we perform the following tasks:

- Five particle classification ( $e^-$ ,  $\gamma$ ,  $\mu^-$ ,  $\pi^-$ , proton) using AlexNet and GoogLeNet within a cropped, high-resolution image

- Comparison of the above with a low-resolution image (downsized by factor of two in both pixel axes)
- Two two-particle separation studies:  $e^-/\gamma$  and  $\mu^-/\pi^-$

The first bullet point shows the first demonstration of CNN’s capability to interpret LArTPC images. The second point is a comparison of how the downsizing factor affects our classification capability. The third bullet point focuses on particular cases that are interesting to the LArTPC community.

### 3.1 Sample Preparation

For this study, we generated images using a single particle simulation built upon LArSoft [22], a simulation and analysis framework written specifically for LArTPC experiments. LArSoft uses GENIE[24] for neutrino event generation and Geant4[23] for particle tracking as well as drift electron simulations inside the LAr medium. Each event for this demonstration simulates only one particle:  $e^-$ ,  $\gamma$ ,  $\mu^-$ ,  $\pi^-$ , or proton. Particles are uniformly generated inside the TPC active volume and directed isotropically with momenta ranging from 100 MeV to 1 GeV, except for protons which were generated with a kinetic energy between 100 MeV to 788 MeV. Particle momenta were generated uniformly within the limits. Then they are passed to Geant4 for tracking.

Once the events are simulated by Geant4, the MicroBooNE detector simulation, also built upon LArSoft, is used to generate signals on all the MicroBooNE wires. This is done with the 3D pattern of ionization deposited by charged particles with the TPC. The detector simulation takes the deposited charge and produces raw wire signals on the wires at certain times using a model for the electron drift that includes diffusion and ionization lost to impurities. This model provides a distribution of charge at the wire planes which then go into simulating the expected raw, digitized signals on the wires. These raw signals are based on 2D simulations of charge propagating either past a wire (in the case of the induction planes) or propagating into the wires (in the case of the collection planes). We then convert these raw wire signals into an image.

The conversion of wire signals to an image, both for the simulation and for detector data used in later demonstrations, proceeds through the following steps. First, the digitized, raw wire signals are sent through a noise filter [19]. Then, they are passed through an algorithm which converts the filtered, raw signals into calibrated signals. The algorithm is designed to take a raw pulse coming from a certain amount of charge deposition into a pulse with a correspondingly height and width [18]. The values of the calibrated, digitized waveforms are then put into an image-like format. To do this, we simply populate a 2D matrix with one axis being time and the other being a sequence of wires. Because the Y plane has more wires than the U and V planes, a full event image for each plane has 3456 pixels. For the U and V planes, which only have 2400 wires, the first 2400 pixels are filled, the rest are set to zero. Note that when we refer to the input to the network as an image, we do not imply that data from the detector must be converted into an image format, i.e. we are not required to define a color scale which converts the wire signal values into 3-channel, 8-bit RGB values. Instead, we are free to define an image as a tensor of any rank and arbitrary dimensions where each element is either 16 or 32 bits.

A full MicroBooNE event has a total of 9600 time ticks. For this and the other demonstrations, we use a 6048 time tick subset ranging from tick 800 to 7200 with

respect to the first tick of the full event. This subset includes time before and after the expected signal region. A reason to include these extra time ranges is to demonstrate that the analysis methodology employed in this study can be used for neutrino analysis in the future with the same hardware resource we have today. Particles are generated at a time which corresponds to time-tick number 800 in the recorded readout waveform, and all associated charge deposition is expected to reach the readout wire plane by time-tick 5855 given the drift velocity. Finally this simulation includes a list of unresponsive wires, which carry no waveform. In total, about 830 (or 10%) of the wires in the detector are unresponsive. The number of such wires on the  $U$  plane is about 400, on the  $V$  about 100, and on the  $Y$  plane about 330. This list of wires is created based on real detector data.

Following a sample generation of images composed of 3456 wires and 6048 time ticks, we downsize by a factor of two in wire and six in time to produce a resulting image size of 1728 by 1008 pixels. We downsize by merging pixels through a simple sum of pixel values. Although this causes a loss of resolution and, therefore, a loss of potentially useful information, particularly along the wire axis, it is necessary due to hardware memory limitation. However the loss of information is smaller in the time direction because digitization period is shorter than the electronics shaping time, which is 2 microseconds, or 4 time ticks.

Next, we run an algorithm called HiResDivider[27] in LArCV that crops out a sub-image of 576 by 576 pixels. The algorithm works in three dimensions and provides images from each plane, although this study uses only the  $Y$  plane view. The algorithm first sub-divides the whole detector active volume into a number of equally sized 3D boxes. Then, it identifies a specific 3D box that contains the start of a particle trajectory. Next, it determines and crops a corresponding portion of each plane view, which generates an image of size 576 by 576 pixels in each plane. This method assumes that a point or region-of-interest is specified upstream of the process chain in order to identify one such 3D box. For this study we used a simulated particle start-point information to identify a particular 3D box. For future algorithms running on data, the same technique developed here can be applied following a neutrino vertex detection by another CNN or a reconstruction path in LArSoft. Example event displays of  $Y$  plane view are shown in figure 5. This study uses only the  $Y$  plane view, the collection plane.

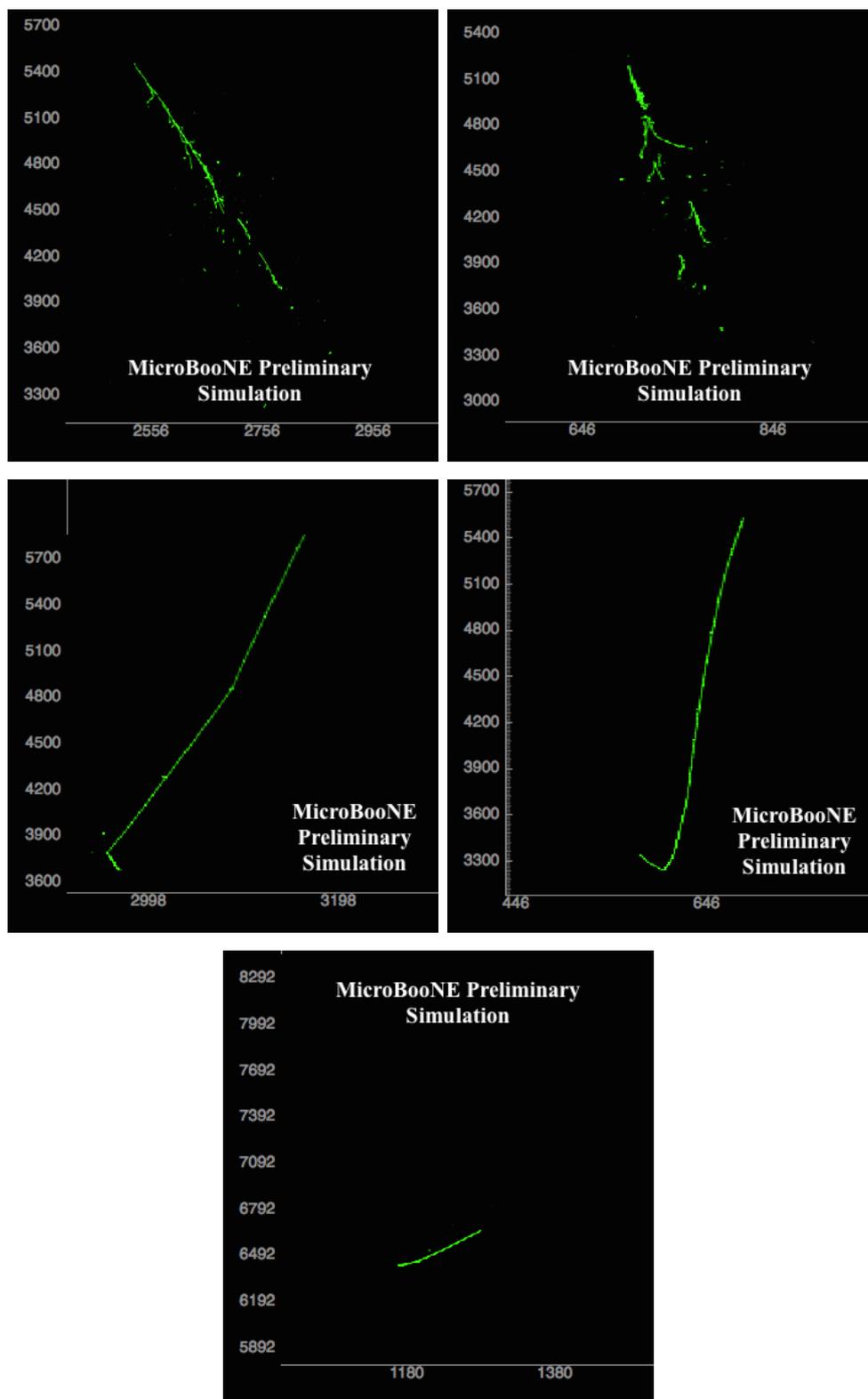


Figure 5: Example event display for 3D box projection images for each particle type on the collection plane. The vertical axis denotes time-tick numbers and the horizontal axis shows wire numbers. From left top in clockwise direction,  $e^-$ ,  $\gamma$ ,  $\mu^-$ , proton, and  $\pi^-$  respectively.

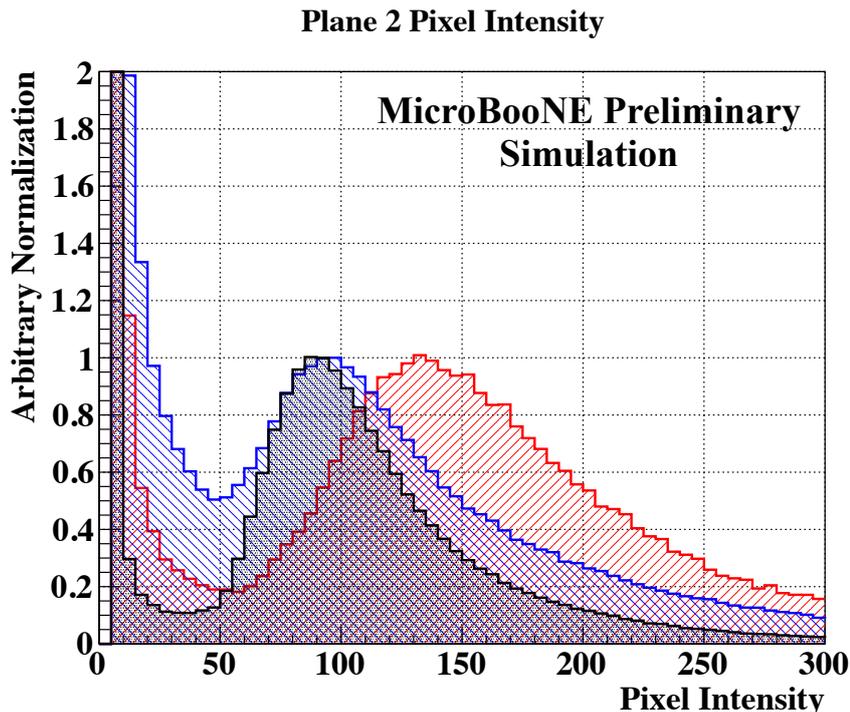


Figure 6: Pixel intensity distribution (pixel value resulting from amplitude after merging operations of source waveform) for the collection plane. Red, blue, and black curves correspond to proton,  $e^-$ , and  $\mu^-$  respectively. The vertical axis represents a relative population of pixels but normalized with an arbitrary unit to match the signal peak amplitude.

Finally, we applied a filter to remove some images that contain almost no charge deposition information. This is typically due to a particle immediately exiting from the detector active volume or the 3D box identified by HiResDivider algorithm. Some small fraction of such cases were also due to dead wires that carry no signal in the detector. For this filtering method, an image was considered empty based on the number of pixels that carried a pixel intensity (PI) value above some threshold. Figure 6 shows the PI distribution on the  $Y$  plane view for electrons and muons after running a simple, threshold-based signal-peak-finding algorithm, on each column of pixels which represent downsized wire signal waveforms. A threshold on the filled pixel count is set to 40 for all simulated particles except for protons, for which set to 20, due to the short distance they travel.

Besides the images, the simulation also provides the needed truth information to train our networks. We save a set of 2D region-of-interest boxes on each plane that is derived based on simulated particle energy deposition profile. A box is stored for each particle that encapsulates where a particle deposited energy in the TPC. These boxes store this region both in the detector coordinates and in the image coordinates. For later demonstrations, we also store a box that combines the single particle information and describes an entire neutrino interaction.

Finally, in preparing the single-particle training sample, we filtered out some proton events based on simulation information indicating there was a nuclear inelastic scattering process that generated a neutron that in turn hit another proton. Such images contain multiple protons with space between them. Filtering those images helped the network

to learn a single proton image, which is aligned with our intention in this single-particle study. After all filtering, we split the outcome sample into 24,000 events per particle type for training and about 6,000 per particle type for training validation (or a total of 120,000 and 30,000 images for the training and validate sets).

## 3.2 Network Training

For the particle classification studies in this demonstration, we train the networks, AlexNet and GoogLeNet, to perform this task. Training a network in essence involves adjusting the parameters of the network such that it outputs the right answer, how ever defined by the user, when given an image. In practice, training a network involves sending a randomly selected small collection of images, called a *batch*, along with a label for each image to the GPU card where images are sent through the network for classification. For a given input image, a network makes a prediction for its label. Both the label and the prediction typically takes the form of a vector of positive real numbers where each element of the vector represents each class of object to be identified. For an image which contains only one example, the label vector will have only one element with a value of one, and the rest being zero. For the training images, the provided label is regarded as “truth” information. The network outputs the predicted label with each element filled with numbers between 0 and 1 based on how confident the network is that the image contains each class. This prediction is often normalized such that the sum over all labels is set to be 1. In this work, we refer to the elements of this normalized, predicted vector simply by “score.” Based on the provided labels of the images and the computed scores, a measure of error, called *loss*, is computed and used to guide the training process.

The goal of training is to adjust the parameters of the network such that the loss is minimized. The loss we use is the negative natural logarithm of the squared-magnitude of the difference vector between the truth label and the prediction. Minimization of the loss is performed using stochastic gradient descent (SGD) [30]. We attempt to minimize the loss and maximize the accuracy of the network during training. In order to avoid the network becoming biased towards recognizing just the images in the training sample, a situation known as over-training, we monitor the accuracy computed for a test sample which does not share any events with the training set.

During the course of properly training a network, we monitor the accuracy computed for the test sample and watch to see if it follows the accuracy curve of the training sample. Both accuracy and loss are plotted against a standard unit of time, referred to as an epoch, which is the ratio of the number of images processed by the network for training to the total number of images in the whole training set. In other words, the epoch indicates how many times, on average, the network has seen each training example. It is standard to train over many epochs. For AlexNet we send 50 images as one batch to the GPU to compute and update the network weights, while 22 are chosen for GoogLeNet. These batch sizes are chosen based on the network size and GPU memory limitation. After the network is loaded into the memory of the GPU, we choose a batch size that maximizes the usage of the remaining memory on the GPU. Figure 7 shows both the loss and accuracy curves during the training process of the five particle classification task. The observed curves are consistent with what one would expect during an acceptable training course.

In addition, for the classification tasks, we also trained networks for lower resolution images by downsizing the training and test images by a factor of two in both wires

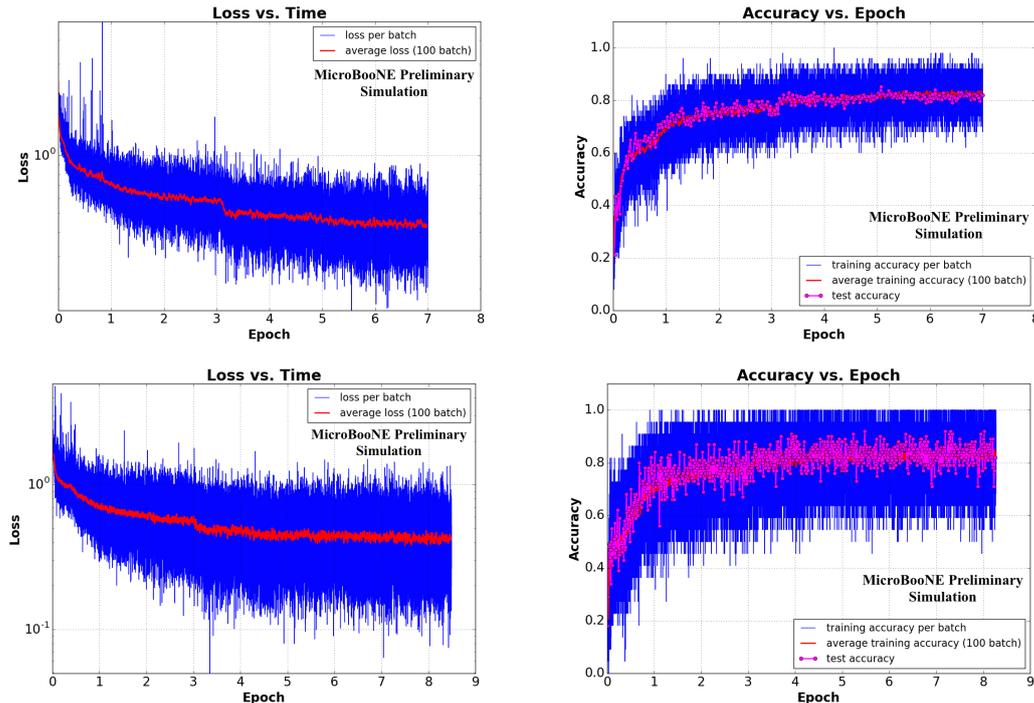


Figure 7: The figure shows network training loss (left column) and accuracy (right column) for AlexNet (top row) and GoogLeNet (bottom row). The horizontal axis denotes elapsed time since the start of training. Blue data points are computed within the training process per batch of images. Red data points are computed by taking an average of 100 consecutive blue data points. The magenta data points on the accuracy curve is computed using a validation sample to identify over training.

and time-ticks to study how the network performance changes. Finally, we trained both AlexNet and GoogLeNet for a two-particle classification task using only the  $\mu^-$  and  $\pi^-$  samples. This is to compare how performance differs when the network is exposed to a richer variety of features.

For the single particle detection, we train the Faster-RCNN network designed for object localization within images. The output of this network differs from the classification networks described above. For a given image, the Faster-RCNN network returns  $N$  classification predictions along with a bounding box for each image, which indicates where in the image the network thinks an object is located. The number of boxes returned,  $N$  is programmable. Because the output of the network differs, the inputs provided during training does so as well. To train the network, we provide for each training image an truth label and a “ground truth” bounding box. (The designation of “ground” indicates the fact that the truth is defined by the user and might not necessarily correspond to ideal box, if it even exists.) For our MicroBooNE images, this is a defined rectangular sub-region of a plane view that contains all charge deposition based on simulation information.

One nice feature of the Faster-RCNN network is that its architecture modular, meaning that it is composed of three parts: a base network that is meant to provide image features, a region proposal network that uses the features to identify generic “objects” and put a bounding box on it, and a classification network that looks at the features inside the bounding box to classify what is in the bounding box. For more details about how

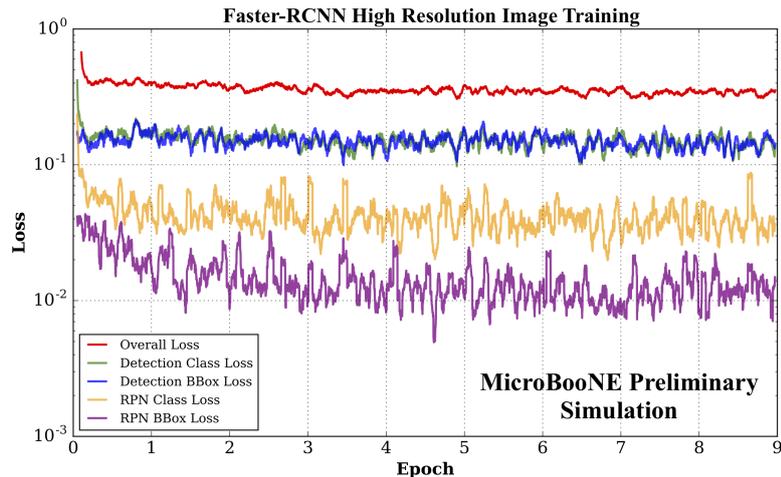


Figure 8: The figure shows the training loss for the 4 different cost functions minimized by SGD in the Faster-RCNN. Five colored lines correspond to a detection network class, detection network bounding box, RPN class, RPN bounding box, and an overall loss. We find the region proposals improve over a period of roughly 5 epochs and the region proposed class begin to stagnate after one epochs. The detection losses slowly decreased over time.

each portion of the network operates see [16]. This modular design means that there is freedom to choose the base network. This is because the portion of the network that finds objects in the image, puts bounding boxes around them, and then classifies the object inside the box is intended to be appended a base network which provides relevant image features. In this study, we used AlexNet trained for five particle classification described above as the base network. We append the Faster-RCNN portion of the network after the fifth convolutional layer of AlexNet. Because the Faster-RCNN network will be asked to identify the same classes of objects as our base network, we transfer the parameters from the classification portion of AlexNet, composed of the last few layers, and use them to initialize the classification portion of the Faster-RCNN network. We train the whole network in the approximate joint training scheme [16] with stochastic gradient descent. Because the Faster-RCNN has two tasks, find objects and classify them, the loss that must be minimized comes from both tasks. This multi-task loss over the course of our training is shown in figure 8. We assess the performance of the detection network in the next section.

### 3.3 Five Particle Classification Performance

Figure 9 shows the classification performance study using the network trained on 5 particle types described in Sec.3.1 using images of size 576 by 576 pixels. Each plot in the figure corresponds to a sample of a specific particle type, and the distribution shows the fraction that is classified as certain particle types defined for the network. The class that is chosen by choosing the one with the highest score. A result using images that have been further downsized by a factor of 2 are shown in figure 10. The classification performance for each particle type as well as the most mis-identified particle types, and their mis-identification rates, is summarized in table 2 and 3.

Table 2: Five particle classification rate

The table shows 5 particle classification performance. The very left column describes the image type and network where “HiRes” refers to a standard 576 by 576 pixel image while “LoRes” refers to a downsized image of 288 by 288 pixels. Right five columns denote classification performance per particle type. Quoted errors are purely statistical and assume a binomial distribution.

Image, Network	Classified Particle Type				
	$e^-$ [%]	$\gamma$ [%]	$\mu^-$ [%]	$\pi^-$ [%]	proton [%]
HiRes, AlexNet	$73.6 \pm 0.8$	$81.3 \pm 0.8$	$84.8 \pm 0.5$	$73.1 \pm 0.8$	$87.2 \pm 0.5$
LoRes, AlexNet	$64.1 \pm 0.9$	$77.3 \pm 0.8$	$75.2 \pm 0.6$	$74.2 \pm 0.8$	$85.8 \pm 0.5$
HiRes, GoogLeNet	$77.8 \pm 0.8$	$83.4 \pm 0.7$	$89.7 \pm 0.4$	$71.0 \pm 0.8$	$91.2 \pm 0.4$
LoRes, GoogLeNet	$74.0 \pm 0.8$	$74.0 \pm 0.9$	$84.1 \pm 0.5$	$75.2 \pm 0.8$	$84.6 \pm 0.6$

Table 3: Five particle classification negatives

The table shows most frequently misidentified particle type for the 5 particle classification task. Following table 2, the very left column describes the image type and network where “HiRes” refers to a standard 576 by 576 pixel image while “LoRes” refers to a downsized image of 288 by 288 pixels. The right five columns denote the classification performance per particle type. Each table element denote most frequently mistaken particle type and its mis-identification rate.

Image, Network	Classified Particle Type				
	$e^-$ [%]	$\gamma$ [%]	$\mu^-$ [%]	$\pi^-$ [%]	proton [%]
HiRes, AlexNet	$\gamma$ 23.0	$e^-$ 16.2	$\pi^-$ 8.0	$\mu^-$ 19.8	$\mu^-$ 7.0
LoRes, AlexNet	$\gamma$ 29.3	$e^-$ 17.6	$\pi^-$ 11.7	$\mu^-$ 16.5	$\mu^-$ 7.9
HiRes, GoogLeNet	$\gamma$ 19.9	$e^-$ 15.0	$\pi^-$ 5.4	$\mu^-$ 22.6	$\mu^-$ 4.6
LoRes, GoogLeNet	$\gamma$ 21.0	$e^-$ 21.3	$\pi^-$ 9.4	$\mu^-$ 19.3	$\mu^-$ 9.1

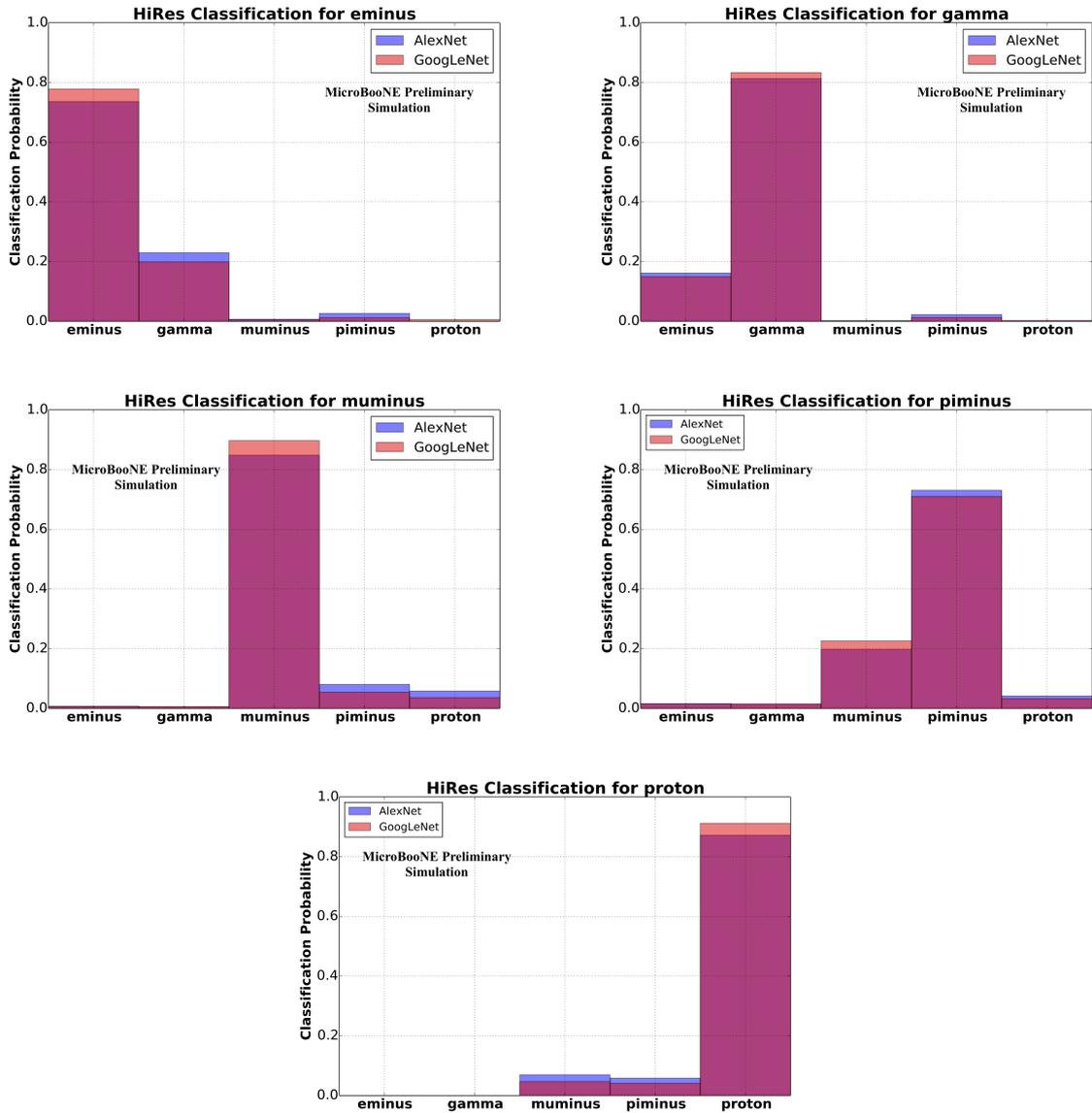


Figure 9: High resolution five class single particle results. Both models struggle with electron and gamma separation as well as distinguishing pions from muons. The GoogLeNet performs better than the AlexNet in all case except for a pion.

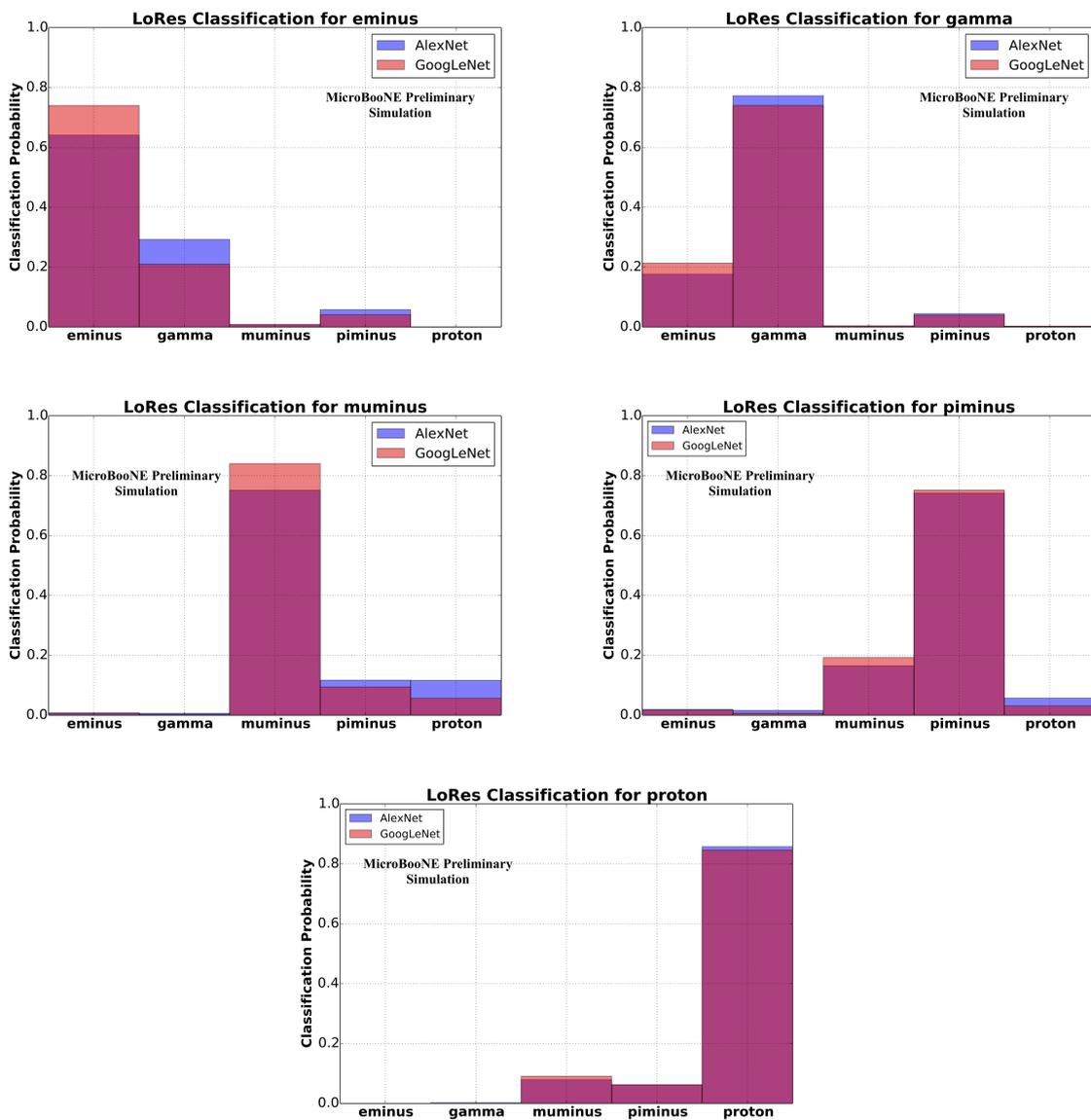


Figure 10: Low resolution five class single particle results. Overall a classification performance is worse than high-resolution image except for  $\pi^-$ .

**Learning Geometrical Shapes** First of all, we see the network has learned about the geometrical shape of showers and tracks irrespective of image downsizing. Both figure 9 and 10 show that the network is more likely to be confused among track-shaped particles ( $\mu^-$ ,  $\pi^-$ , and proton) and also among shower-shaped particles ( $e^-$  and  $\gamma$ ) but less across those categories.

**Learning  $dE/dx$**   $e^-$  and  $\gamma$  have similar geometrical shapes, i.e. they produce showers, but differ in the energy deposited per unit length, called  $dE/dx$ , near the starting point of the shower. The fact the network can separate these two particles fairly well means the network has learned about the difference of  $dE/dx$  at the shower start. It is also worth noting that downsizing an image made the result worse, likely because downsizing, which involves combining neighboring pixels, smears out the  $dE/dx$  information.

**Network Depth** We expect that GoogLeNet is capable of learning more features than AlexNet overall, because of its more advanced, deeper model. Our results are consistent with this expectation and can be seen in the results table. When comparing how AlexNet and GoogLeNet are affected by downsizing an image, it is interesting to observe that GoogLeNet performs better for those particles with higher  $dE/dx$  ( $\gamma$  and proton), which suggests GoogLeNet was using the  $dE/dx$  information more effectively than AlexNet.

### 3.4 $\pi^-/\mu^-$ Separation

We used a set of 3400  $\pi^-$  and 3400  $\mu^-$  sample from the high resolution validation sample and ran the network to classify them into each particle type. The left plot in figure 11 shows an efficiency vs. purity (EP) distribution for a  $\mu^-$  selection from the total of 6800 images. The efficiency is the percentage of true muon images that were labeled correctly as a muon. The purity is the fraction of true pion images that were labeled incorrectly as a muon. Data points for the efficiency and purity are from using different cut values of the  $\mu^-$  classification score to determine what was a muon. The cuts range from 5 to 95%. The lowest purity corresponds to the lowest selection score cut. Blue and red data points are for AlexNet and GoogLeNet. The lack of data points at the higher score end is because there were no events that passed those score cuts.

**Training with a more Feature-Rich Sample** Figure 11 also contains orange and cyan data points that are obtained from GoogLeNet and AlexNet networks trained with the five-particle sample. However, because in this case we are interested in  $\pi^-/\mu^-$  only, the  $\pi^-/\mu^-$  classification score is computed by re-normalizing the sum of  $\pi^-$  and  $\mu^-$  classification score to 1 for these networks. The fact that the network trained with five-particle classes performs better than the network trained only on two particle classes, i.e. solely on a  $\pi^-/\mu^-$  sample, might appear counter intuitive, but this is one of the CNN's strengths: being exposed to other particle types, it can learn and generalize more features. It can then use this larger feature space to better separate the two particles. Quoting a specific data point that lies in the outer most part of the orange curve, the 5-particle classification network achieved 94.6% selection efficiency with 79.2% sample purity.

**$\mu^-/\pi^-$  Indistinguishability** The right plot of figure 11 shows the score distribution of  $\mu^-$  and  $\pi^-$  from GoogLeNet trained for the five-particle classification task with a higher

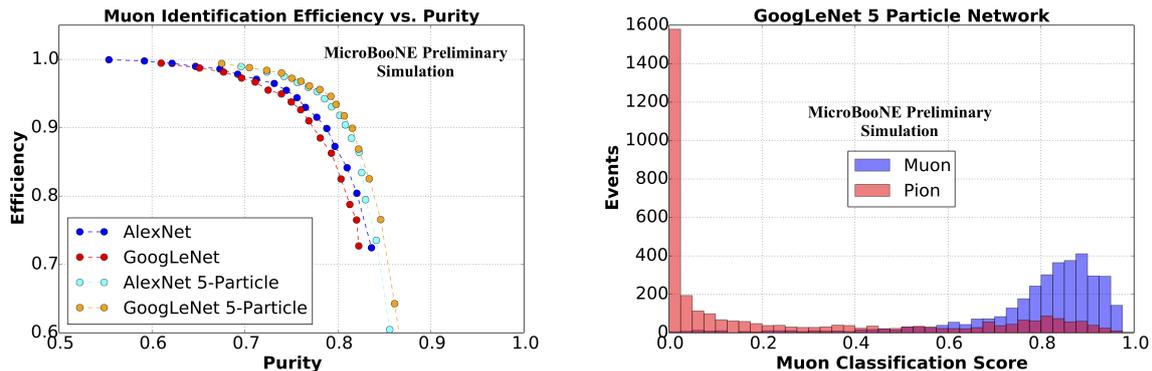


Figure 11: Left:  $\mu^-$  selection EP curve for 1:1  $\mu^-$  and  $\pi^-$  mixture (6800 events total). Blue and red data points are AlexNet and GoogLeNet respectively trained with a sample set that only contain  $\pi^-$  and  $\mu^-$  as described in Sec.3.4. Orange and cyan data points are from GoogLeNet and AlexNet respectively, trained for 5 particle classification. Right:  $\mu^-$  score distribution from GoogLeNet trained for 5 particle classification where score is re-normalized for  $\mu^-/\pi^-$  separation purpose.

resolution image from the previous study. It is interesting to note that there is a small, but distinct, set of  $\pi^-$  events that follow the  $\mu^-$  distribution. This makes sense since the  $\pi^-$  has a similar mass to the  $\mu^-$  and decays into  $\mu^-$ . As a result, some  $\pi^-$  can look very similar to a  $\mu^-$ . A typical way to distinguish  $\pi^-$  is to look for a nuclear scattering, which occurs more often for  $\pi^-$  than for a  $\mu^-$ . There can also be “kink” in the track at a point where the  $\pi^-$  decays-in-flight into a  $\mu^-$ , although this is generally quite small. When neither is observable, the  $\pi^-$  looks like a  $\mu^-$ , however when there is a kink or visible nuclear interaction involved,  $\pi^-$  is distinct. This can be seen by a very sharp peak for  $\pi^-$  in the right figure. The same reason explains why there is no  $\mu^-$  above 95% (with the statistics of this sample) because  $\mu^-$  can never be completely distinguished from those small fraction of  $\pi^-$  that do not carry any kink nor visible nuclear interaction.

### 3.5 $e^-/\gamma$ Separation

We show a similar separation study for  $e^-$  and  $\gamma$  as we did for  $\mu^-/\pi^-$ . This time, however, we only show the result using five-particle classification, since we saw those networks seem to perform better, presumably for similar reasons. The left plot in figure 12 shows  $e^-$  selection efficiency and purity in a 1:1 mixture of 5200 events taken from the validation set. The outer-most point achieves selection efficiency of 94.3% with purity of 71.9%, although one might want to ask for a better separation with less efficiency depending on the goals of an analysis.

**$e^-/\gamma$  Indistinguishability** The right plot in figure 12 shows an electron classification score distribution for both  $e^-$  and  $\gamma$ . The separation is not as strong compared to  $\pi^-/\mu^-$ : the two types are essentially indistinguishable in the range of scores from 0.3 to 0.6. We note that our high-resolution image has a factor of two in wire and six in time compression applied, and hence this might be the highest separation achievable. It may be interesting to repeat more studies across different downsizing levels (including no downsizing) and

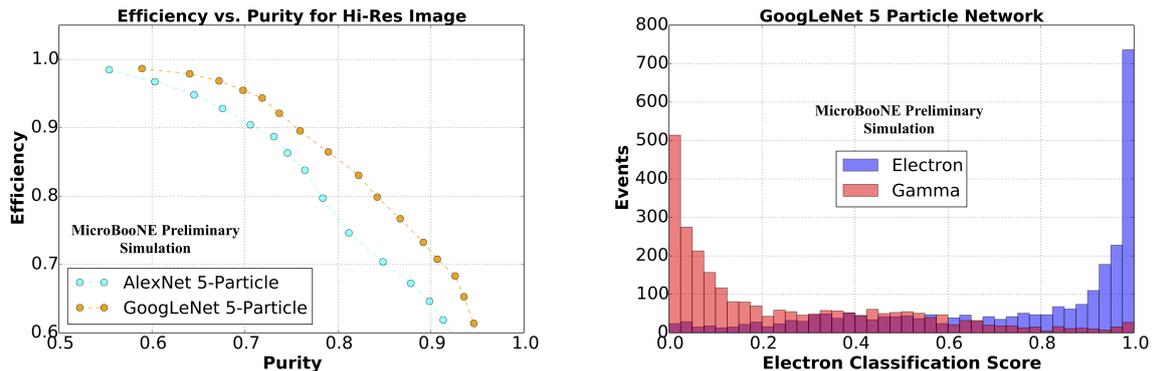


Figure 12: Left:  $e^-$  selection EP curve for 1:1  $e^-$  and  $\gamma$  mixture (5200 events total). Orange and cyan data points are from GoogLeNet and AlexNet respectively, trained for 5 particle classification. Right:  $e^-$  score distribution from GoogLeNet trained for 5 particle classification where score is re-normalized for  $e^-/\gamma$  separation purpose.

study how this separation power changes. However, that is beyond the scope of this publication.

### 3.6 Particle Detection Performance

To assess the Faster-RCNN detection performance on the single particle sample, we let the Faster-RCNN network infer a set of bounding boxes per class for each high resolution event image containing one particle. Typical detection examples can be seen in figure 13. As it is done for all studies in this section, this study used the same training and validation sample described in sec. 3.1.

To quantify the Faster-RCNN detection performance on the single particle sample we infer a set of bounding boxes per class for each high resolution single particle image. To quantify this performance we compute the intersection over the union of the ground truth bounding box and the predicted box with the highest network score. This is the standard performance metric used by object detection networks to compare with one another. Intersection over union (IoU) is defined for a pair of boxes in the following way: the intersection area between two boxes is first computed by calculating the overlap area and then divided by the difference between the total area of the two boxes and their intersection area. Specifically for two boxes with area  $A_1$  and  $A_2$ ,

$$\text{IoU} = \frac{A_1 \cap A_2}{A_1 + A_2 - A_1 \cap A_2}. \quad (4)$$

This quantity is unity when the predicted box and the ground truth box overlap perfectly. In other words, the predicted network box is of the same pixel dimensions. In figure 14 we plot the IoU for the different five-particle classes. We separate the detected sample into the five different particle types and break down each sample by their top classification score. The true class label is in the title of the plot, and the legend lists 1) the five particle types that were detected for the sample, 2) the number of detections in the histogram for that class, and 3) the class-wise fraction of all detections. For this plot we make a cut on the network score of 0.5. We observe good detection accuracy and ground truth bounding box overlap on the muon and proton classes. If we consider classification only, muons and

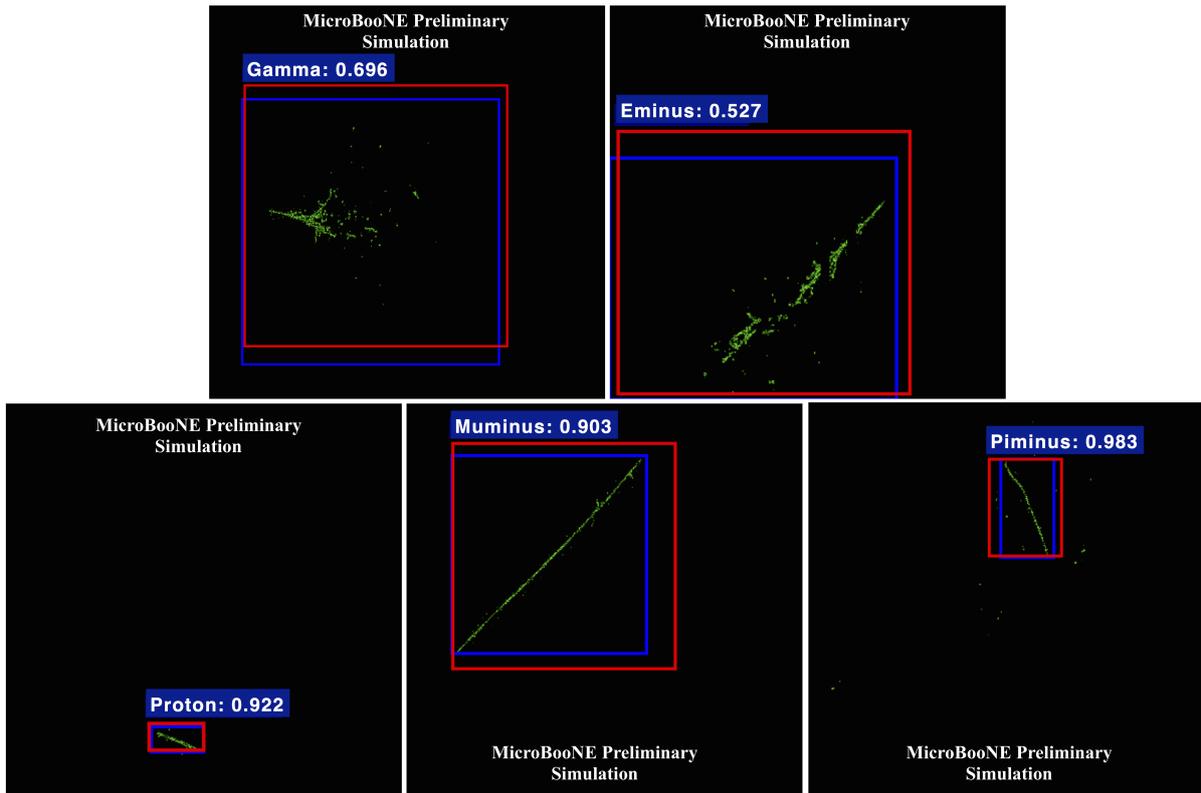


Figure 13: Example detections for each of the five particle classes using high resolution images (576 by 576 pixels). The blue box is the true bounding box. The red box is the network inferred bounding box. The detection score and inferred class sits atop the red box on the top left corner. It's interesting to note the network's ability to capture a shower-type particle's ionization charge within the detection box

protons have the smallest contamination from other particle types. This could be a result of the strong classification performance of the AlexNet classifier model revealed in figure 9. The electron and  $\gamma$  samples had expectedly mutual contamination between the two as previously revealed by the pure AlexNet classifier. We also find a small contamination of piminus detection in the electron and  $\gamma$  samples at the low IoU range indicating that some piminus have features shared with electrons and gammas. This is consistent with lower energy gammas and electrons appearing track like in liquid argon. It is also interesting to note that both classes' IoU are similar, meaning the network is able to encapsulate the charge that spreads outwards as the shower develops. This means the model values the shower-like nature of electron and  $\gamma$  as essential and uses these learned features for classification. Lastly, the  $\pi^-$  particle exhibits the least number of detections above a threshold of 0.5. We also find the largest contamination from  $\mu^-$ .

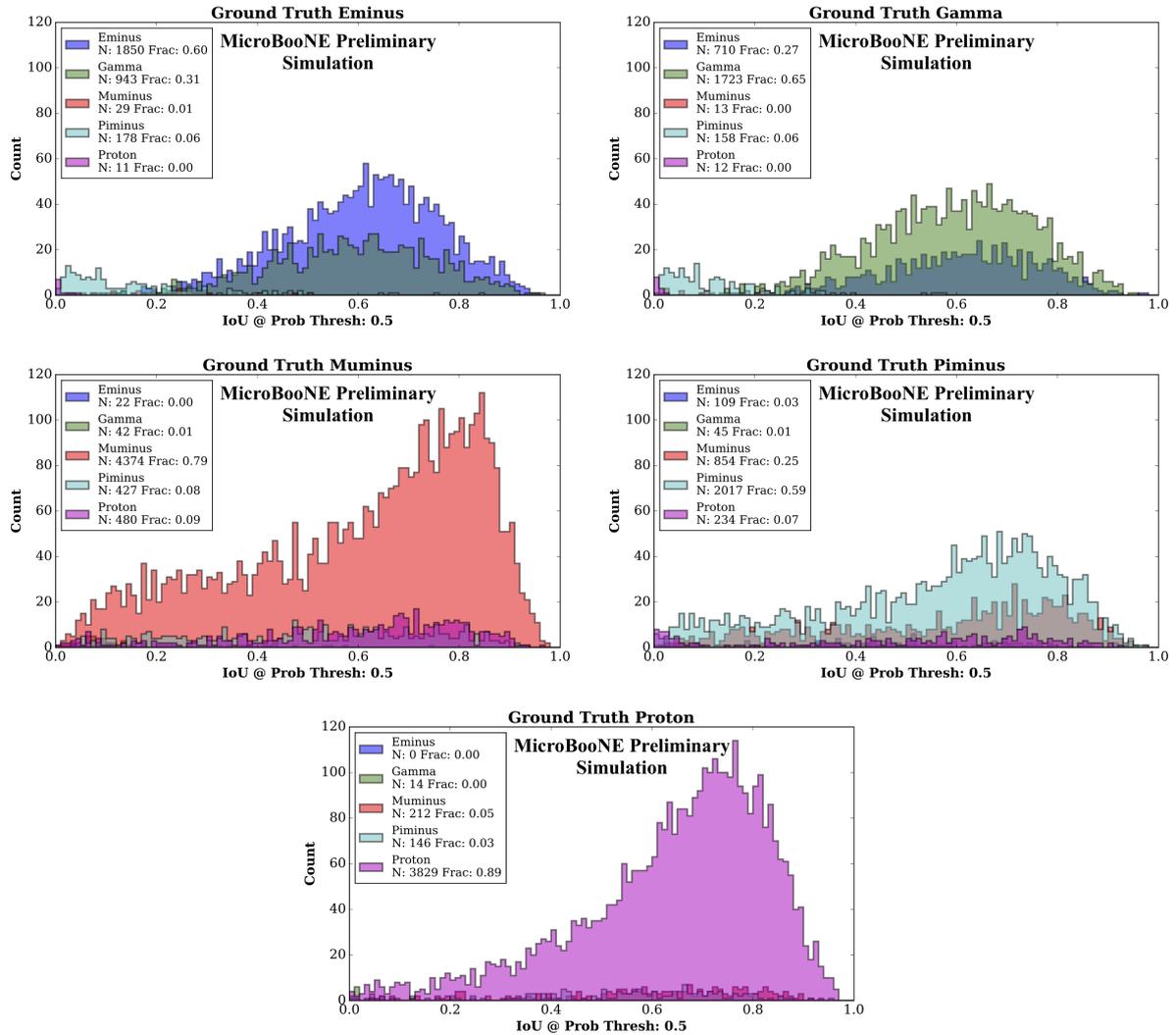


Figure 14: Intersection over union distributions for each of the 5 classes. The test sample is first separated by true particle class, and then broken down by the set of detected particle types.

### 3.7 Summary of Demonstration 1

Our first demonstration shows that two CNN models, AlexNet and GoogLeNet, can learn particle features from LArTPC images to a level potentially useful for LArTPC particle-identification tasks. We also find that, as one might naively expect, downsizing an image has a negative effect upon particle detection performance of  $e^-$ ,  $\gamma$ ,  $\mu^-$ , and proton. The exception are  $\pi^-$  images which is yet to be further understood. GoogLeNet trained for five particle classification showed the best performance among tested cases for  $e^-/\gamma$  (94.3% efficiency and 71.9% purity) and  $\mu^-/\pi^-$  (94.6% efficiency and 79.2% purity) separation tasks.

## 4 Demonstration 2: Single Plane Neutrino Detection

In this section, we report on the study of neutrino event classification and detection using a single plane image. This is an important step for demonstrating that we can localize a neutrino interaction within an event image. If successful in identifying neutrino interactions, a high-resolution image then can be cropped around the interaction region, after which one can apply techniques, such as those in demonstration 1, at a higher resolution. For the first iteration of this neutrino interaction task, we perform a detection using only a single plane image, as combining multiple (wire,time) 2D-planes to uniquely localize a neutrino location is non-trivial. This is because the location of a neutrino interaction in all three planes requires the prediction of a bounding box in each plane that should be constrained to represent the same 3D box in the detector. Future work will ask the network to predict such a volume and project the resulting 2D bounding into the plane image.

We take a two-task approach:

1. Neutrino event selection
2. Neutrino interaction (ROI) detection within an event image

Our strategy for the 1st step is to train the network with Monte Carlo neutrino interaction images overlaid with data cosmic background images. Accordingly, we define two classes for the classification task: 1) Monte Carlo neutrino overlaid with data cosmic events, and 2) purely data cosmic events. For this to succeed, the Monte Carlo signal waveform, which is affected by particle generation, detector response simulation, and calibration, needs to match the real data. Otherwise a network may find a Monte-Carlo-specific feature to identify neutrino interactions, which may work for the training sample but may not work at all, or in a biased way, for real data.

For a neutrino event classification task, we use a simplified version of Inception-ResNet-v2 by Google Research [32], one of the state-of-the-art networks in the field of Deep Learning. The network is composed of three different types of modules that are stacked on top of one another. Because our images are larger ( $864 \times 756$ ) than that used by the original network ( $299 \times 299$ ), we must shrink the network so that we can train the network with the memory available on one of our GPUs. The three modules are labeled A, B, and C. For details on the modules and the original network, please refer to [32]. Here, we only list our modifications: we reduced the number of Inception-A modules and Inception-C modules from 5 to 3, and the number of Inception-B modules from 10 to 5.

For neutrino detection training, we use AlexNet as the base network for the Faster-RCNN, object detection network, similar to what we did for demonstration 1. However, unlike our approach in demonstration 1, we train AlexNet+Faster-RCNN from scratch, instead of using the parameters found by training the network from a previous task. We found that we were not able to train AlexNet to reasonable level of accuracy through fine-tuning. Accordingly, the AlexNet+Faster-RCNN model trained in this study is specialized for detecting neutrino-vertex-like objects, instead of distinguishing neutrinos against cosmics. The study demonstrates successful neutrino interaction localization nevertheless.

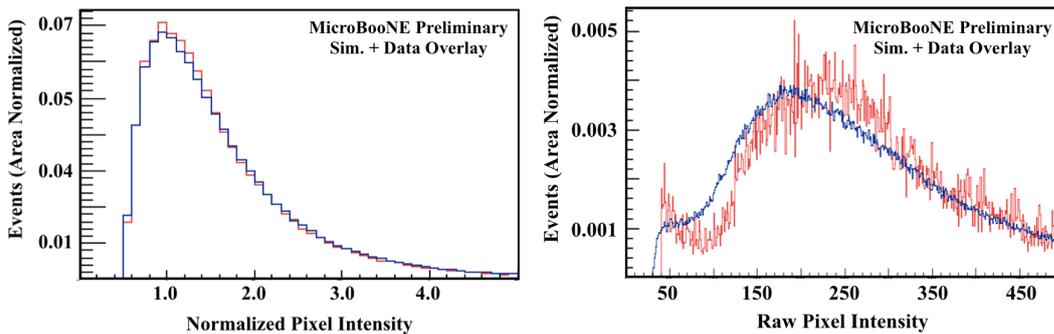


Figure 15: Right: raw PI value distribution for a MIP peak found by a simple peak finding algorithm. Red is from data images and blue is from MC cosmic images. Left: scaled PI distributions to match between data and simulation where the scale factors applied to the data and MC are set such that the peaks are normalized to 1. A threshold is applied at 0.5 on the scaled PI value.

## 4.1 Data Sample Preparation

For this study, we generated simulated neutrino events without dead wires first, and then we overlay on it, an event capture by the detector when there was no beam going through the detector. This image should contain only cosmic ray tracks and no beam neutrino interactions. Since the current simulation is lacking on models of noise that exist in data, we consider the use of real detector data for the background as an important factor for this technique demonstration. This is because the real data will have noise features and unresponsive wires throughout the image that a final application of the network must contend with. But this advantage does come at a cost. We note that there might be differences in the wire response modeling between the simulation and the real data that the network could use to identify neutrino interactions in the training sample. However, the topology of many neutrino interactions should be distinct enough to be used by the network. Further study to quantify the effect must be performed in order to apply the technique for high level physics analysis. However, the goal of this work is to demonstrate that this technique from computer vision can be applied to this task rather than to precisely quantify the expected performance.

Upon overlaying two images, we remove the signals from certain wires in the simulation image based on the bad wire record which is inspected on a per-event basis for the cosmic data image. We use the same software as before to process the simulated neutrino images and the cosmic background event samples used for overlay. We create images with a downsizing factor of 4 for wires and 8 for time-ticks which makes an event image 756 (wires) by 864 (time-ticks).

In order to prevent the network from keying on simulation-specific features, we must calibrate the signal response between data and simulation. We have run a simple analysis algorithm to find a minimum ionizing peak (MIP) in data and simulation cosmic events. The distribution of raw PI values is shown in figure 15 (right) for raw images of data and simulated cosmic background events. We applied a scaling factor such that the peak amplitude becomes 1.0, and threshold is 0.5 to reduce unmatched low PI noise components. The resulting PI distributions are shown in the left of the same figure. The  $Y$  (collection) plane's scaling factor was 0.00475 for simulation and 0.005 for data. These scaling factors are applied to the simulated neutrino and the cosmic background events

respectively prior to an image overlay. An example event display image of an uncalibrated simulated neutrino as well as overlaid image are shown in figure 16.

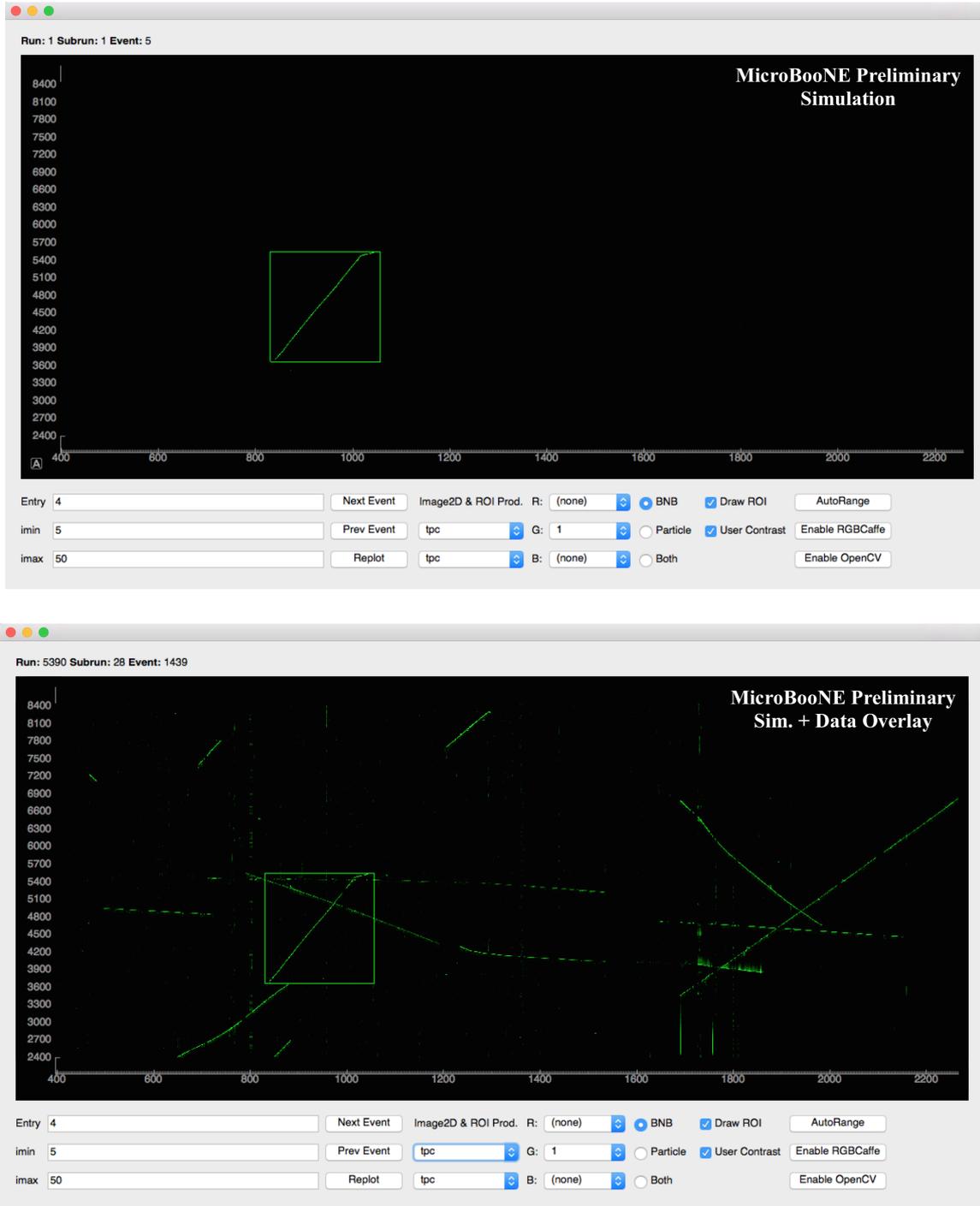


Figure 16: Top: example event display of a neutrino interaction on  $Y$  plane view. Square boxes indicate ROIs per plane created based on charge deposition profile using simulation information. Bottom: the same event overlaid with data cosmic image. The bounding box is used to train the detection network layer.

We prepared an approximately 1:1 mixture of cosmic-only images and simulated-neutrino-overlaid images for both training (totaling 101,191 images) and validation (totaling 32,220 images).

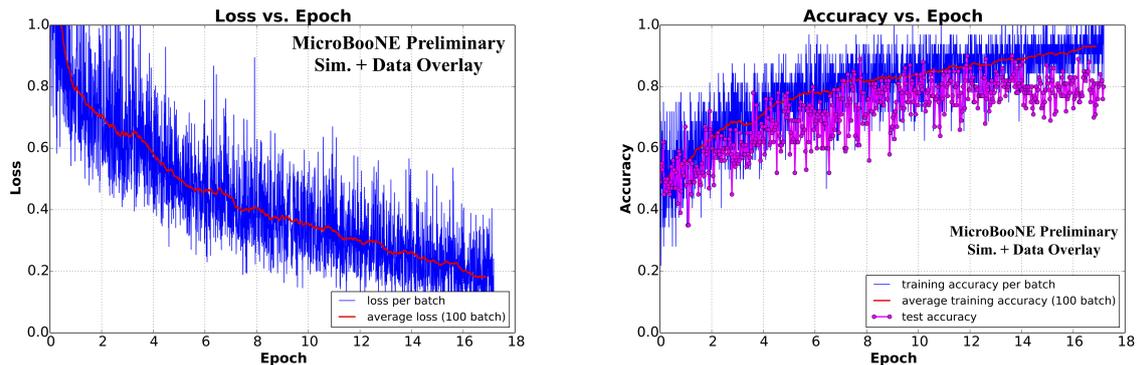


Figure 17: Left: 1-plane classification loss curve for our simplified version of Inception-ResNet-v2. Right: accuracy curve monitored during training. In both plots blue data point was computed from the training process while red is our interpretation by taking an average of 100 consecutive blue data points. The magenta curve on the right plot is an accuracy computed using a test sample.

## 4.2 Training

We used our reduced Inception-ResNet-v2 network for classification training of two labels: neutrino vs. cosmic events. At the data preparation stage during the training, we performed a random cropping of an image to a slightly smaller size along the time (vertical) axis to avoid an issue of over-training. This is an example of data augmentation, which is a standard techniques employed in the field. Presenting a randomly modified version of an image each time it is given to the network is widely believed to help reduce over-training. Figure 17 shows the classification accuracy reached the level of 80%. The slightly lower accuracy of a test sample relative to the training sample points to a slight, but acceptable, over-training. Slight over-training is common practice as it is a sign that the number of parameters in a model, which typically scales with the model’s capacity to learn, is just a bit larger than needed and, therefore, considered near-optimal.

For detection training, the base network used in the Faster-RCNN architecture is the AlexNet model, where the region-finding and classification layers of Faster-RCNN are appended to the final fifth convolutional layer of AlexNet. This is similar to what was done for the single particle detection network. In this instance, we modify the allowed output classes to two, one for neutrino events and the other for an all-inclusive “background” class which includes cosmic rays. We train the AlexNet convolution layers from scratch, contrary to the single particle case, by re-initializing their weights and biases. We also re-initialize the two large fully connected layers at the end of the model, which sit downstream of the region-finding portion of the network, with random samples from a Gaussian distribution centered at zero with 0.001 standard deviation. We have empirically found that re-initializing the last fully connected layers with Gaussian weights, rather than copying those from the classification stage, especially in the case of neutrino detection, helps ensure the detection specific layers learn both bounding box regression, and classification.

Finally, when we train the detection model, we only provide the network with cosmic+neutrino images and ignore the cosmic ray-only images. Each neutrino image contains a single truth bounding box around the neutrino interaction vertex, which encap-

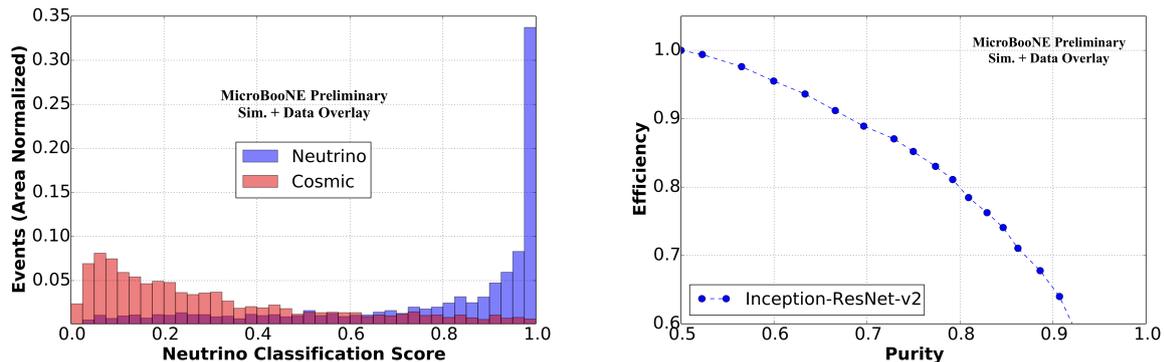


Figure 18: Left: neutrino score (horizontal axis) for the validation sample. The blue distribution is for neutrino events while red is for cosmic background events. Right: Efficiency vs. purity curve for equal number of cosmic background and neutrino signal events.

ulates the bounding boxes associated with each daughter particle of the interaction. We use the same strategy employed in the single particle demonstrations to find the bounding boxes for the daughter particles. We train with the standard batch size used by the model authors.

### 4.3 Performance

Figure 18 shows the neutrino classification score distribution from the validation sample. The distribution matches our intuition, having a sharp peak when there is a neutrino event. Cosmic background events are not expected to have a sharp peak since there is nothing to key-on within those images. The bottom plot in the same figure shows the overlaid neutrino event selection efficiency and purity for the case of having an equal number of neutrino events and cosmic-only events. A particular data point on this curve achieved 87.1% efficiency with 72.9% purity by selecting events with a neutrino score above 0.35. For the case of MicroBooNE, however, a cosmic and neutrino event do not have equal statistics. Such details are included in demonstration 3, which aims for a more realistic analysis implementation, with full detector information. Given that this is single-plane performance using real data cosmic background events, we expect that the performance will improve once all three views are used. In the next section, demonstration 3, we explore further improvements by using a network that combines three plane information together.

Examples images showing regions that the network has identified as being neutrino-like are shown in Figure 20-23. The different figures show examples of where the network correctly identifies neutrinos with strong confidence (figures 20 and figures 21), where the network identifies cosmic ray tracks as neutrinos but with low confidence (figure 22), and where the network labels cosmic ray tracks as neutrinos with high confidence (figure 23). Each image in those figures is a high-resolution sample of the event readout. The images are single channel and are represented with false color, with blue representing low PI values, green representing PI values produced by minimum ionizing particles, and red representing large charge depositions. The yellow box in the figures are the ground truth bounding boxes around the neutrino determined from Monte Carlo information. The red

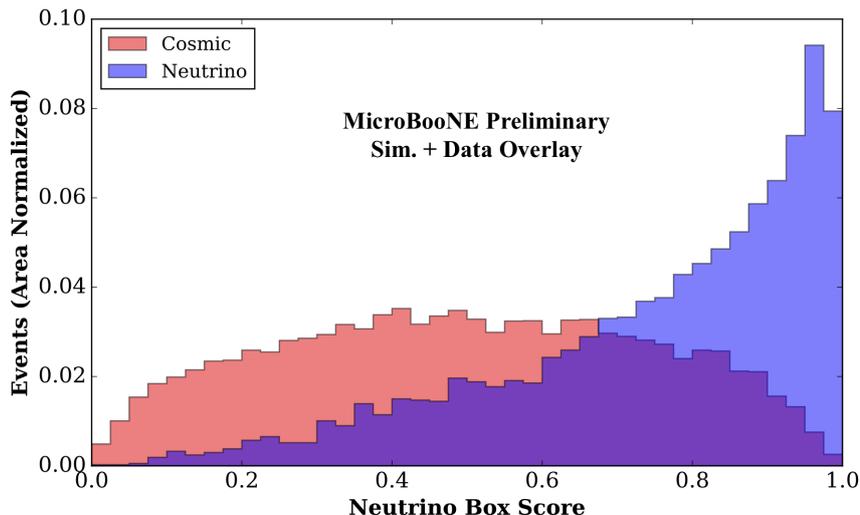


Figure 19: Neutrino detection box score distribution for neutrinos (blue) and cosmic (red), normalized by area. It shows there is some discrimination only by just detection network.

box is the network prediction for the region containing the neutrino and the network score is labeled in white text above. Figures 20 and 21 are showing the results where the CNN successfully located a neutrino in an image with a high score ( $>0.9$ ). Figure 23 shows two types of mistakes: 1) finding a high score ( $>0.9$ ) bounding box in a wrong location in a neutrino event, and 2) also in a cosmic background event where there is no neutrino. In either case, the bounding box is containing an interaction topology that could be mistaken as a neutrino event. Thus, these are not boxes drawn randomly. Finally, Figure 22 shows examples of how the CNN is drawing many boxes in a cosmic background event. Boxes shown in this figure are those with neutrino scores less than 0.1.

Figure 19 shows the distribution of neutrino bounding box scores predicted by FasterRCNN per event for both neutrino+cosmic and cosmic-only images. We can see that the network is successfully finding a more neutrino-like bounding box in neutrino events than cosmic background events. Moreover, because the network is not trained to specifically discriminate cosmic events, it finds a bounding box with a moderate score value among cosmic. This is a good sign, as it indicates that the network is not simply keying on a mere difference of data and simulation.

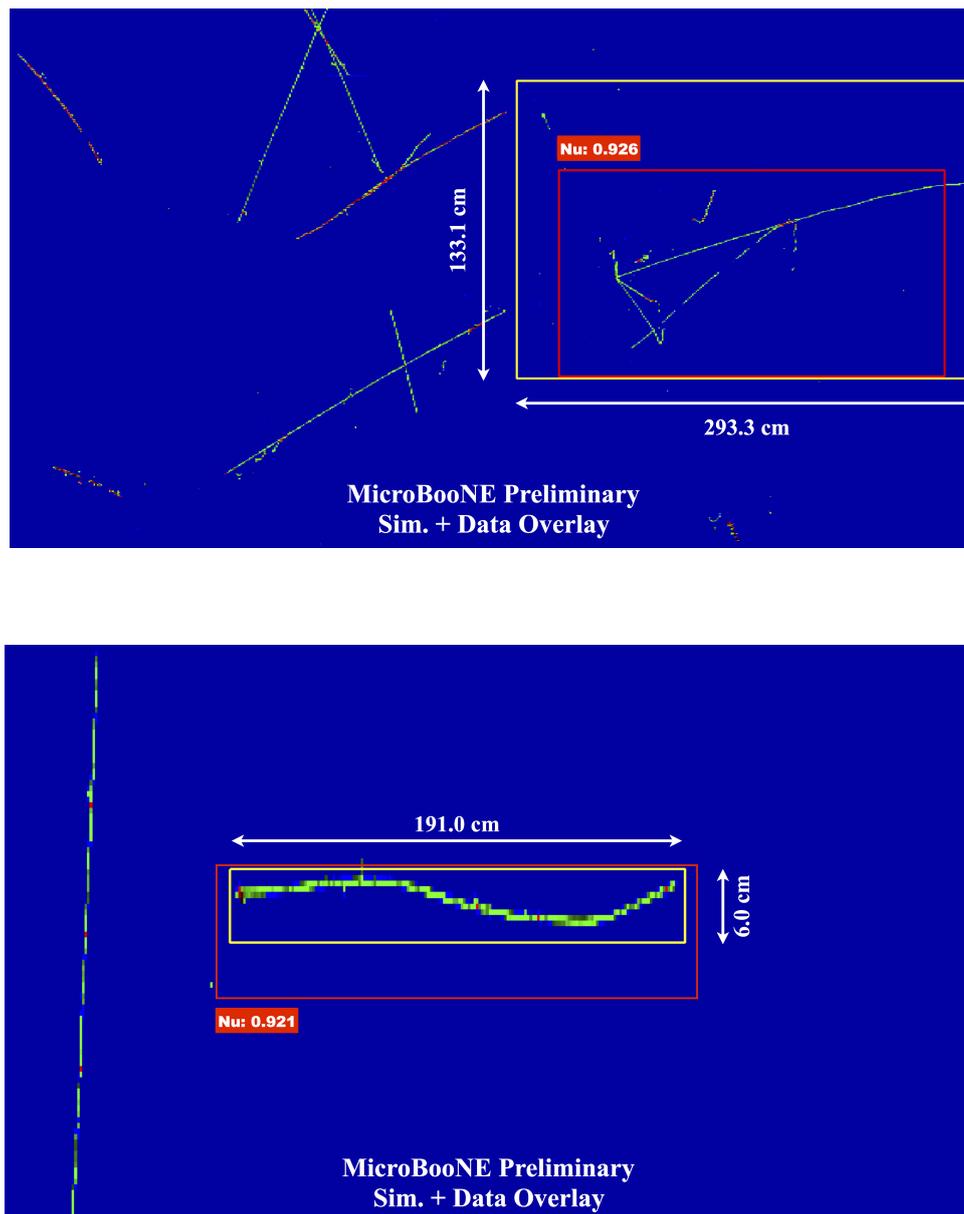


Figure 20: Detected neutrino bounding box within an event image. Top: A  $\approx 2$  GeV Neutrino interaction with a single muon, three protons, three piminus, one gamma, and one electron produced. Bottom: CC interaction with a single muon and proton produced. The red predicted box extends in the correct dimension to encapsulate the full interaction. The yellow box shows the truth bounding box. Both figures are zoomed-in from the original event display, and the length scale of the truth box are shown in cm.

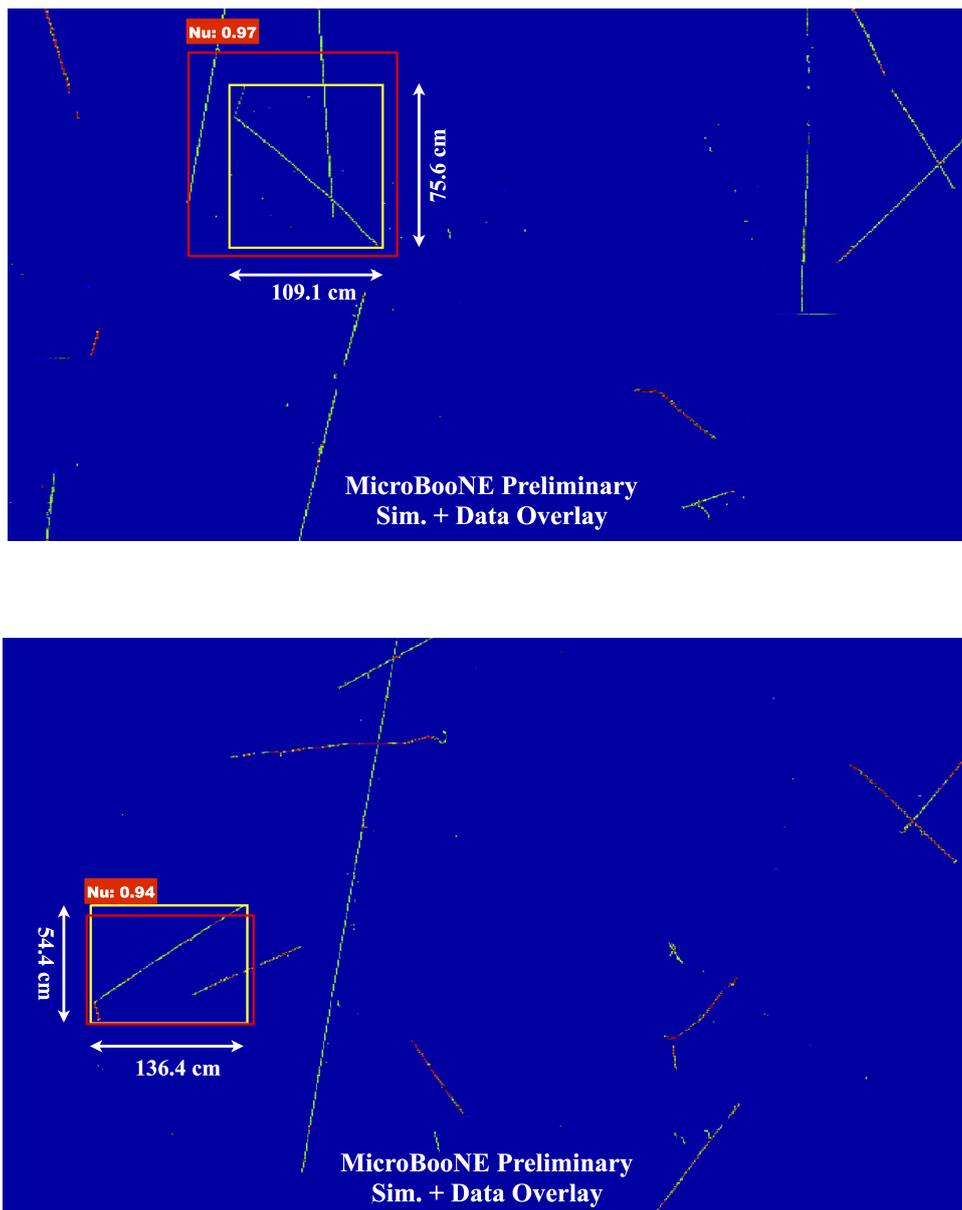


Figure 21: Detected neutrino bounding box within an event image. Top: Muon and charged pion produced from a  $\approx 1$  GeV neutrino. The detection box (in red) appears to capture a neighboring cosmic ray, but maintains the overall shape of the ground truth box (in yellow). Bottom: CC event with muon and proton produced. Both figures are zoomed-in from the original event display, and the length scale of the truth box are shown in cm.

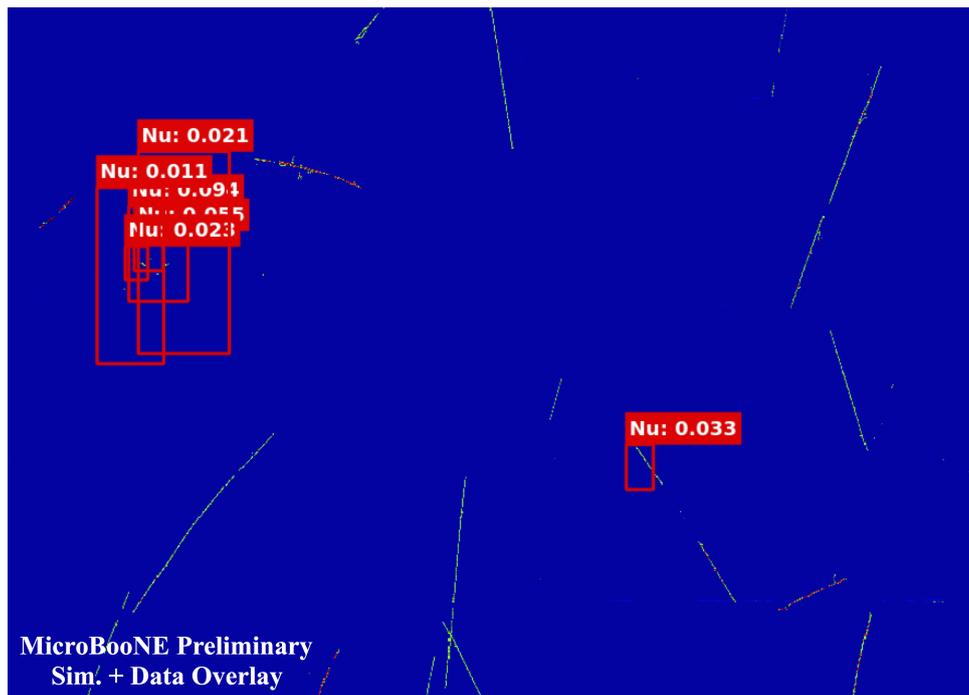
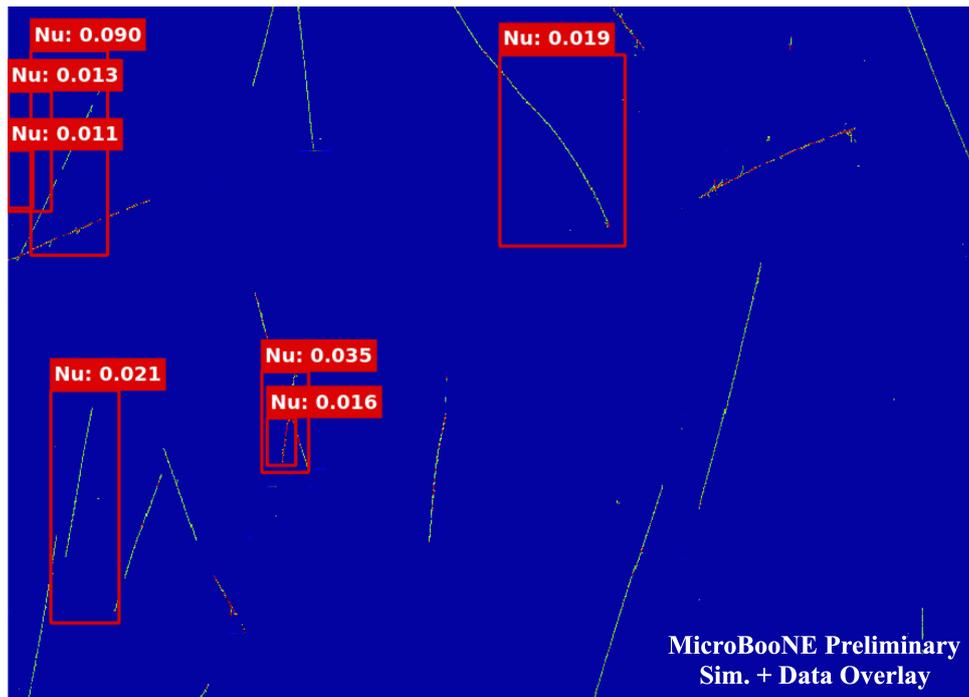


Figure 22: Example of cosmic background events (both top and bottom) with detected neutrino bounding boxes with score less than 0.1. One can see many proposals were made. Both figures show the full region of 6048 time-ticks and 3456 wires on the collection plane.

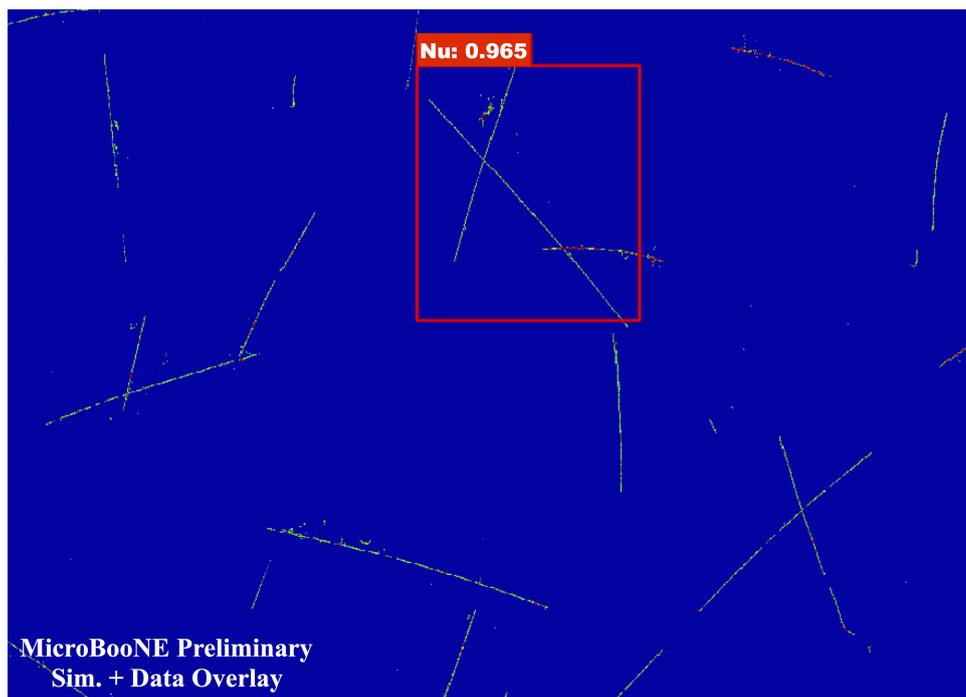
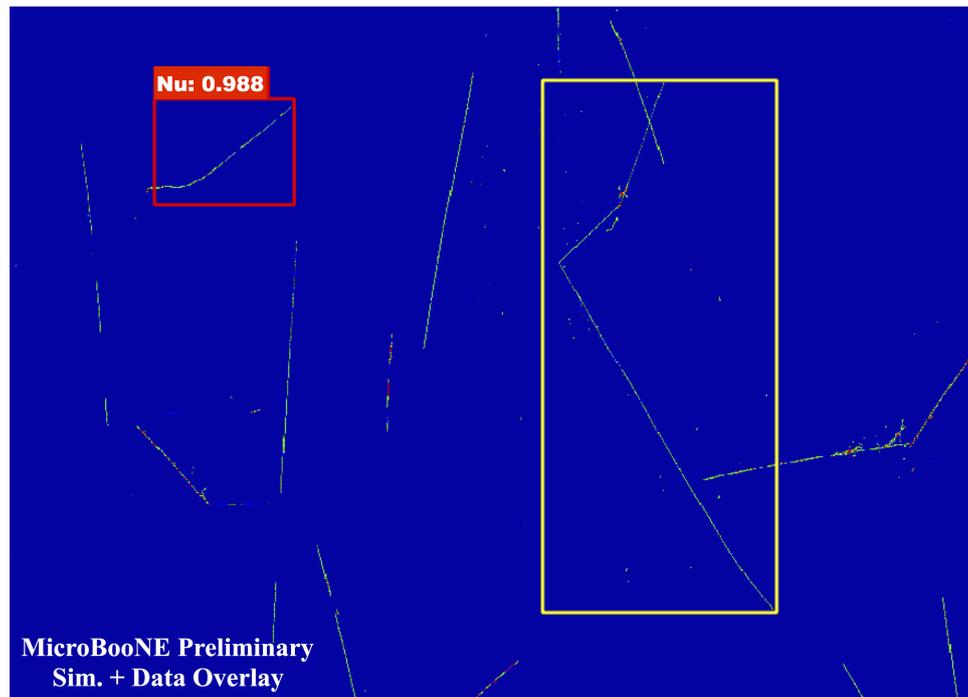


Figure 23: Top: an example of a true neutrino (yellow box) event where the network found the highest score bounding box in the wrong location (red box). Bottom: an example of cosmic-only event where the network found a bounding box (red) with high neutrino score. There is no neutrino interaction in this event. Both figures show the full region of 6048 time-ticks and 3456 wires on the collection plane.

## 4.4 Summary of Demonstration 2

In this study, we demonstrated how a CNN can be used to perform event classification using only one plane view from the MicroBooNE data, overlaid on simulated neutrino events. We use a simplified Inception-ResNet-v2 architecture. In particular, starting with equal number of signal and background events, we quoted a selection efficiency of 87.1% with a 72.9% purity for neutrino events by cutting on a neutrino score of 0.35 (figure 18). We also demonstrated that CNNs, in particular using AlexNet+Faster-RCNN model in this study, can successfully learn neutrino interaction features and localize them in an event view. These are important steps for an analysis chain using CNN techniques. Such tasks would be near the first part of an analysis chain, providing a method for event selection and locating interactions in a large LArTPC detector, the latter which can be cropped at a higher resolution and passed to downstream algorithms.

## 5 Demonstration 3: Neutrino Event Identification with 3 Planes + PMT Data

### 5.1 Network Design

The network we designed for the three-plane neutrino classification task is based on the ResNet network discussed in ref. [33]. This network features the repeated use of what is referred to as residual convolutional modules. These modules have been demonstrated to help networks train more quickly [32]. As described in more detail below, the input images were chosen to be  $768 \times 768$  pixels with 12 channels as a third dimension. This is a relatively large amount of input data for a network, compared to existing models. This compelled us to make a new network model based on a truncated ResNet network. This constraint arises because of the memory limitation of the GPUs, which have 12 GB. The more slowly one reduces the size of the output feature maps at each layer, the more memory that is required at each given layer. This means that there can be fewer layers and that each layer can learn fewer filters. More layers and filters means that the network can learn more and, in principle, attain a higher performance. However, by preserving the resolution of the feature maps, the network is exposed to more detailed features in the image. Fully exploring the space of the network configurations is reserved for future studies.

The full neutrino ID network is shown in figure 24. The network uses all three planes. The three planes are passed separately through the “stem” portion of the network which consists of three “standard” convolutional layers, along with a couple of “pooling” layers, which reduce the size of the output feature map. All convolutional layers (in both the stem and elsewhere in the network) use a technique known as “batch normalization” [35] and are followed by a ReLU. Note that the convolutional layers in the stem are the same for all three views. In other words, the parameters of those layers are shared. The output feature maps of the stem are then concatenated and passed through nine residual modules. This is followed by an average pooling layer that incorporates a technique known as “dropout” [34]. After the dropout layer, two linear, fully-connected layers are used to classify the output features and determine if the event has a neutrino or does not. For more details on the network and different component layers included, please see Appendix A.

### 5.2 Sample Preparation and Training

The network is asked to identify two types of images: a cosmic-only image and a cosmic+neutrino interaction image. We use off-beam events in which the detector is triggered externally but the data is read out in exactly the same way as for neutrino events for two purposes. One purpose is to use them to make a training sample of cosmic-only images by running them through a trigger algorithm used to select beam events. This trigger selects interesting events from the stream of beam data by choosing only those that have in-time flashes of light observed in the PMTs that are in coincidence with the expected arrival time of the neutrino beam. This extra trigger selection is necessary in order to make the intended type of background event where an empty beam event is still saved due to accidental coincidence between the beam window and a cosmic ray particle. A second purpose is to make the cosmic+neutrino sample. In this case we overlay onto the image a neutrino interaction generated using the BNB flux, the GENIE interaction

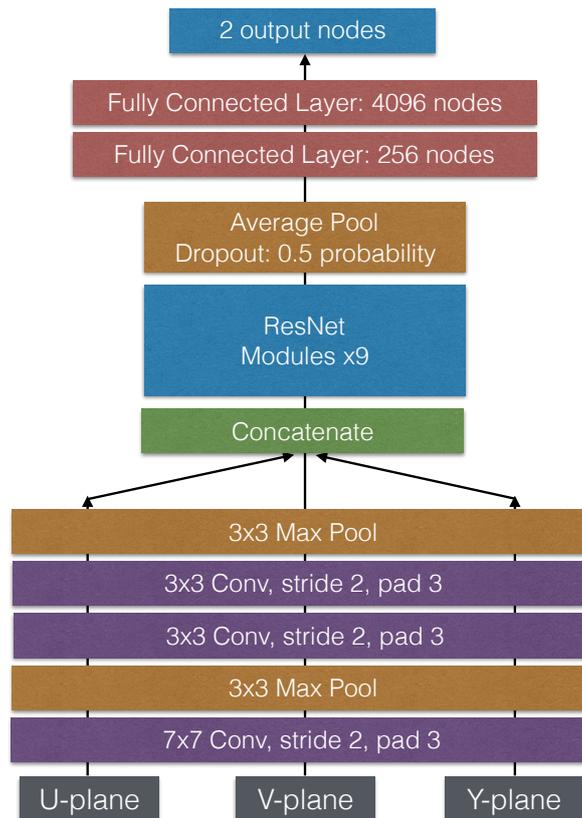


Figure 24: Neutrino ID Network. See text for more details.

model, and our full simulation chain. The time at which the neutrino event occurs is set to fall within the expected beam window. We then pass this merged event through the PMT trigger. Note that unlike the cosmic-only events, we do not require that the externally-triggered data event into which we overlay the MC neutrino has an in-time flash.

When selecting simulated BNB events to overlay, we applied a few quality cuts to identify the events of interest. The first is a cut that ensures that the vertex occurs inside the TPC. The second selects events with neutrino energy above 400 MeV. The third selects charged-current quasi-elastic (CCQE) events.

A full event in MicroBooNE consists of 6400 time ticks and 3 planes of waveforms coming from the 2400, 2400, and 3456 wires in the  $U$ ,  $V$ , and  $Y$ -planes, respectively. As discussed in the previous section, we are forced to down-sample the images to a lower resolution in order to fit our network model within the GPU memory limitation. For this first study, we attempt to choose an image size that was still fairly large ( $768 \times 768$ ) while still allowing us to construct a fairly deep network that could view all three planes. Unfortunately, as shown within demonstration 1, down-sizing has a negative effect on the performance.

In addition to the three TPC planes, we also provide the network with additional images that convey supporting information that we believe the network will find useful for classification. This was inspired by the algorithm, AlphaGo, which was used to play

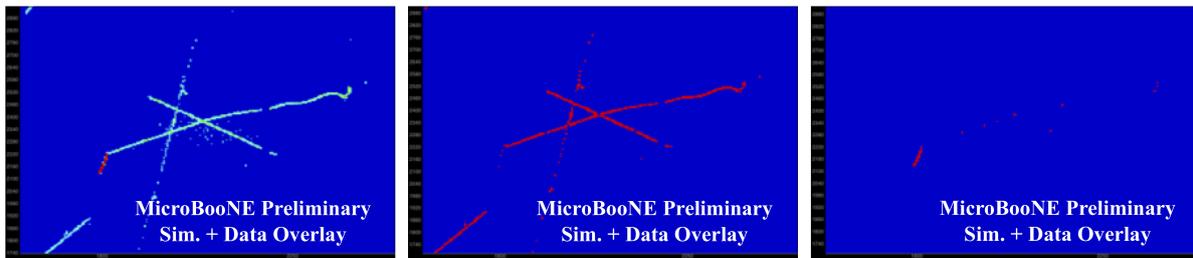


Figure 25: Comparison of a TPC image (left) and the MIP image (center) and HIP image (right). This example is for the collection plane. The box (drawn by hand) indicates the neutrino interaction. The MIP and HIP images are made by assigning pixel values of 1.0 to any pixel that has an PI value consistent with a minimum ionizing particle and heavily ionizing particle, respectively. For the neutrino interaction, one can see that the proton track has more HIP pixels, while the neutrino track has more MIP pixels in the beginning and the middle and HIP pixels at the end.

the board game Go [37]. One core feature of that algorithm is a convolutional network that helps predict the next move in a game based on the arrangement of pieces on the board. The network was trained with a very large set of board positions paired with the next expert move. However, in addition to pieces on the board, supporting information is also provided. Analogously, we provide three additional supporting images for each TPC plane: an image marking pixels with PI values consistent with a minimum ionizing particle (MIP), an image marked with PI values consistent with heavily ionizing particles (HIP), and, finally, an image weighted by the location and quantity of PMT pulses.

The images marking PI values consistent with MIP (minimum ionizing particle) and HIP (highly ionizing particle) are made by assigning a value of 1.0 to each pixel whose PI value falls within a certain range. Figure 25 shows an example of both type of images. For the MIP image, the PI value must fall between 10 and 45. For the HIP image, the PI value must be greater than 45. These two supporting images are provided in order to help the network identify the neutrino interaction vertex. The information to do this is indeed already in the image. However, because we down-sample the image fairly early in the network (through the use of large strided convolutions and pooling layers) in order to reduce the memory size of the network, this calorimetric information can get lost. Therefore, we provide it as a binary image.

The PMT-weighted image is provided to the network in order to help it ignore tracks that are inconsistent with the PMT signal in the beam window. Figure 26 compares an image of the TPC charge with its corresponding PMT-weighted image. The image is made by weighting the charge on each wire by (1) the amount of charge seen by each PMT and (2) the distance each PMT is to the wire in question. The weight for each wire,  $i$ , is given by

$$W_i = \sum_p^{N_{PMTS}} w_{ip}^{(d)} q_p. \quad (5)$$

In the above,  $w_{ip}^{(d)}$  is the PMT distance weight between wire,  $i$ , and PMT,  $p$ , and is defined by

$$w_{ip}^{(d)} = \frac{1}{D_{ip}^\alpha} / \hat{w}_i, \quad (6)$$

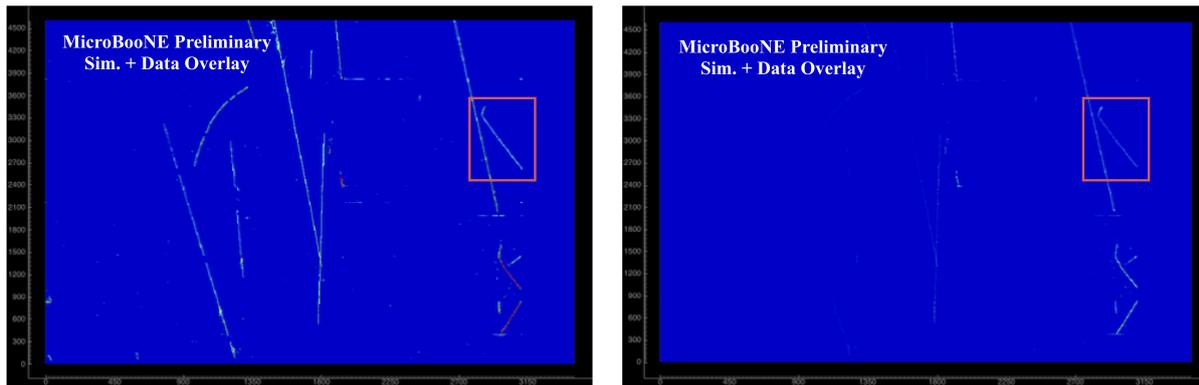


Figure 26: Comparison of a TPC image (left) and PMT-weighted image (right). This example is for the collection plane. The PMT-weighted image is made by weighting the TPC image using information about the amount and location of charge seen by the PMTs inside the beam window. As this example demonstrates, the PMT weighted image tells the network where it should focus its attention. The box (drawn by hand) indicates the neutrino interaction.

where  $D$  is the shortest distance between wire,  $i$ , and PMT,  $p$ , and  $\alpha$  is 2 for the Y-plane and 1 for the U- and V-planes. The factor,  $\hat{w}_i$  is the largest valued weight for wire,  $i$ , and is used to normalize the set of distance weights for a given wire. For the second factor in  $W_i$ ,  $q_p$  is the PMT charge weight. It is defined as

$$q_p = Q_p / \hat{Q}, \quad (7)$$

where  $Q_p$  is the amount of charge in PMT,  $p$ , inside the beam window, and  $\hat{Q}$  is the maximum  $Q_p$  used to normalize the set of weights. The PMT-weighted image acts like what is known in the deep learning field as a “soft-attention model.” It helps indicate to the network where it would be helpful to focus attention.

In total, we prepared 35146 training images and 14854 validation images. For each image set, half are cosmic-only and the other half are neutrino+cosmic.

The network was trained using the optimization algorithm RMSprop [31] with an initial learning rate of  $1.0 \times 10^{-4}$  and a weight decay value of  $1.0 \times 10^{-3}$ . The former controls the magnitude that the parameters of the network are allowed to update during each batch. The weight decay parameter adjusts a common constraint term added in the loss function to keep the parameters close to zero. Such a constraint is a common tool to fight over-training. A batch of size seven was used. The training was run for about 75 epochs, where an epoch marks when all the training images have had a chance to pass through the network. Note that each time the network saw a particular image, it was perturbed slightly. This technique is known as data augmentation and it is a common and very strong technique to prevent over-training of the network. The images are originally  $768 \times 768$ . During image preparation, all images were given 10 pixels worth of padding to both ends of the image in the time dimension, making the output image  $768 \times 788$ . The values for these padding pixels was set to zero in all channels. During training, the images are randomly cropped back down to  $768 \times 768$  before being passed to the network. This shifts the image in time, while preserving the wire-dimension. We found that without doing this random cropping, the network over-trained within several epochs.

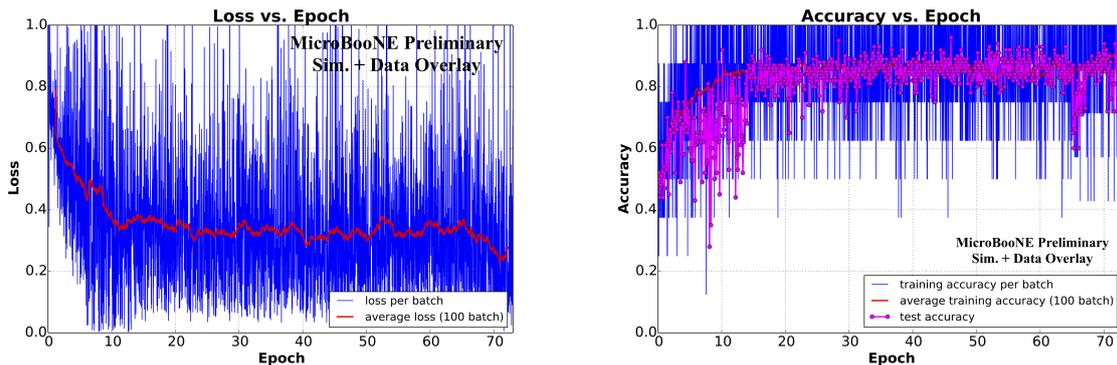


Figure 27: Loss vs. epoch (left) and Accuracy of training and validation data set vs. epoch (right). The dip around epoch 70 was due to a change in the training parameters, made to try and help training.

Figure 27 shows the loss curve along with the training and validation set accuracy which reached a little over 85%, an improvement over the single plane training. The training and validation accuracies are close in value, indicating that the network had not yet over-trained when the training was stopped. The training took about two days on a single Titan X GPU. Note that the dip in the validation set accuracy around epochs 65 and 70 occurred because we stopped the training and changed the learning rate in order to see if we were in a local minimum of the loss function. This did not seem to make a noticeable difference in the training for validation accuracy in the end.

## 5.3 Results

### 5.3.1 Performance on Validation Data Set

To analyze the performance of the network, we use the validation data set. This set was not used to optimize the parameters of the network, but rather monitor the performance over time. To do this, we pass through every image in the validation data set through the network and record its neutrino class score. Unlike the training stage, which position the crop randomly, at test time, we crop out only the zero padding before evaluating the score.

Figure 28 shows the distribution of neutrino scores for true cosmic-only images (the red histogram) and true cosmic+neutrino images (the blue histogram). Both histograms are area normalized separately. One can see that there is a good separation between the two types of event. Figure 29 quantifies this separation. The curve plots the fraction of cosmic-only events removed versus the neutrino+cosmic event efficiency. A cut on the neutrino score is varied to generate the curve. Requiring 90% cosmic-only rejection, one gets about 75% neutrino+cosmic event efficiency. Requiring 90% neutrino+cosmic event efficiency, 75% of cosmic-only events are rejected. Because the number of cosmic-only events is expected to be much higher than events that contain neutrinos in the data, a selection that used this network will likely require approximately 99% or higher cosmic-only event rejection for which the neutrino+cosmic efficiency is 60%. The point in the purity vs. efficiency curve closest to 100% in both categories is about 85% efficiency with 85% cosmic-only rejection. Note that the efficiency is relative to the selected neutrino

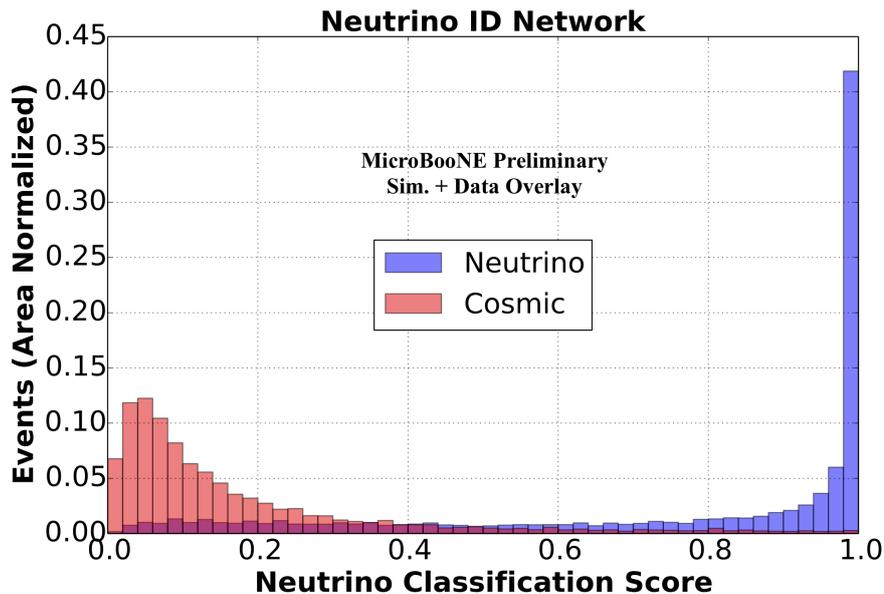


Figure 28: Distribution of neutrino class score for cosmic-only data (red) and simulated neutrino events overlaid on data cosmic (blue) images. The cosmic-only and neutrino+cosmic distributions are area normalized separately. Events come from the validation data set (which is not used to train the network).

events for the training sample: CCQE interactions with neutrino greater than 400 MeV.

## 6 Conclusion

In this work, we have successfully trained several neural networks to perform particle classification, particle and neutrino detection, and neutrino event identification. In particular, a single particle classification has shown a performance of identifying  $e^-$  with an efficiency of 94% and a 71.9% purity from an equal mixture of  $e^-/\gamma$ , and  $\mu^-$  with an efficiency of 94.6% with a 79.2% purity from a mixture of  $\mu^-/\pi^-$  with equal statistics. Using full detector information, demonstration 3 showed a capability of neutrino event selection with 85% neutrino+cosmics event efficiency with 85% cosmic-only rejection.

While these results show that CNNs can be applied to analyze LArTPC images and has the potential for good performance, there is much more to be explored. For one, our studies overlaid simulated neutrino interactions onto off-beam data events. This was to train the networks with a realistic depiction of detector noise, something a successful application must contend with. However, before being able to quantify the network performance with high enough accuracy for physics-level analyses, we will need to perform careful, separate analyses to ensure that simulated signal waveforms carry the same features in real data. This is one example of the many systematic uncertainty studies that haven't been discussed as they are beyond the scope of this work. Another type of potential systematic error may arise from using simulated neutrino events which might cause a model-dependent bias to what a network learns. Careful study is required to quantify such bias, ensuring that it is small. Or one might, instead, aim to develop a model independent approach for detecting neutrinos in data. Such studies as those mentioned above are a requirement for physics analyses using this technique and developing

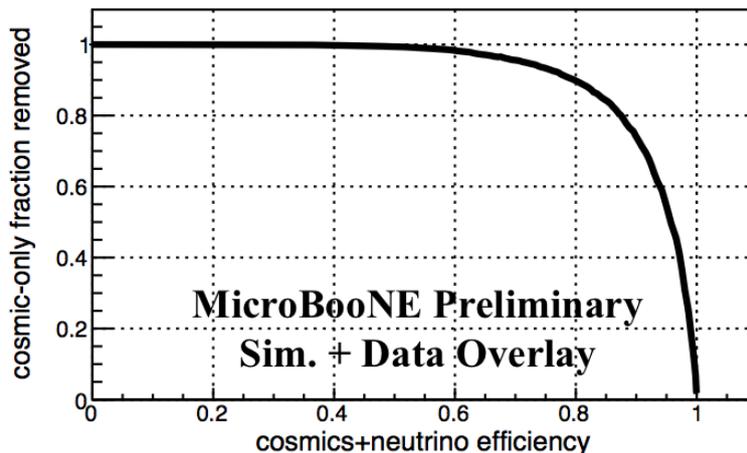


Figure 29: Fraction of cosmic-only events remained vs. neutrino event efficiency. Each point on the curve is for some cut value on the neutrino class score.

new methods to quantify this will be explored in future work.

Overall, this work takes the first steps in exploring CNNs and the broader discipline of Deep Learning to LArTPCs. We have learnt the following lessons:

- It is necessary to consider a strategy to either crop or down-size for a large LArTPC detector, and study the effect. We have shown one possible method (demonstration 1).
- All else being equal, it is best to limit the down-size, as the networks use the  $dE/dx$  information for classification. A factor of 2 downsizing of the images showed a clear negative effect (demonstration 1).
- For a particle classification task, using weights for a network trained for a greater variety of features might perform better. This was seen in  $\mu^-/\pi^-$  study where the network trained for five-particle classification outperformed largely over the same network trained for only  $\mu^-/\pi^-$  sample (demonstration 1).
- The Faster-RCNN architecture can be used for neutrino interaction detection in a large event image (demonstration 2).
- It is possible to combine different detector information to enhance the feature in image data. We have shown one method to combine PMT and TPC information as illustration (demonstration 3).
- A multi-view architecture can be employed to process multiple TPC plane-views for neutrino interaction selection, to greatly enhance the performance from using a single plane information only (demonstration 3).

There are certainly many other avenues, besides the ones listed here, to study and possibly improve performance. However, the proof-of-principle tests conducted in this work show that CNNs can deliver results worthy of further exploration and provide a useful starting-point for those interested in developing their own CNNs for reconstruction of neutrino events in Liquid Argon TPCs.

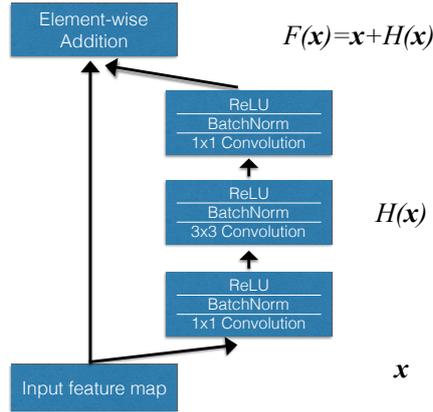


Figure 30: A ResNet module. This is the main unit used for the neutrino ID network.

## A Details of the Components in the Three-plane Neutrino ID Network

This appendix provides more details on the custom network described in Sec. 5.1.

The primary type of network layer employed is known as the residual convolutional module [33]. The basic idea is that instead of training the network to optimize a function,  $F(x)$ , over feature space,  $x$ , it is easier for a network to optimize residuals,  $H(x)$ , such that  $F(x) = x + H(x)$ . A diagram of the residual module employed in our network is given in figure 30. The insight of the authors [33] arose from their experience in training very deep networks where the number of layers surpasses 100 or more. What the authors found is that these very deep networks perform worse than shallower ones. This seemed counter-intuitive, because if additional layers were not useful, a network in principle should just learn to use the identity convolution to pass existing feature maps forward in the network and retain the same performance, all other things being equal. This insight led them to hypothesize that with deep networks it was difficult (or just require too much time) for convolutional layers to optimize to the identity if it needed to. Therefore, the residual module, which can be initialized near to  $H(x) \approx 0$ , allows a layer to start at the identity convolution and optimize if favorable.

Another proposed reason this network structure is useful is that during training the amount of error made by the network can be propagated back more easily through the network as weights are updated (this is called “back-propagation”). As can be seen in figure 30, information passing backwards can flow both through the convolutional layers and through a “bypass” connection directly back to the input feature map. The bypass protects the information from potentially small derivative values coming from the convolutional layers. Studies by a later group found that non-residual, very deep networks, while they do not necessarily perform worse than residual networks, train more slowly than residual networks. That residual networks train faster corroborates both hypotheses as to why residual network structures are effective and, thereby, more than warrant their use [32]. Based on these findings, we use these type of modules in our network.

Our version of the residual module follows that used by ref. [33]. We use a  $3 \times 3$  convolution that is both preceded and followed by  $1 \times 1$  convolutions. Each convoluta-

tional layer is followed by a ReLU. It is believed that this  $1 \times 1 - 3 \times 3 - 1 \times 1$  structure has about the same complexity as two  $3 \times 3$  convolutions but contains more non-linearities and less free parameters. This is desirable as it is believed to increase the separation power while reducing over-fitting. The convolutional layers also use a technique known as batch normalization, or BatchNorm [35].

BatchNorm is a technique where for a batch during training the activations of a layer are normalized to have a mean of zero and a variance of one. Such a layer is meant to reduce learning time by preventing the distribution of activations of each layer to change, something the authors refer to as “internal covariance shift”. In other words, during training the scale of the activations of a layer can change requiring layers to adjust and producing other problems. For example, the distribution of activations can become small. This will cause gradients during back-propagation to become small and producing a bottle-neck that stops the development of the layer and layers below it. By actively maintaining the activations to be normal, their values and their derivatives do not vanish. BatchNorm also seems to possibly provide a degree of regularization and thereby helps to reduce over fitting [35].

The final technique incorporated into the network is known as “dropout”, where parts of the feature map are set to zero randomly during each back-propagation pass. This technique is thought to be very effective in limiting over-training [34]. It is also thought to force the network to develop features that can stand on their own, instead of relying on two features that must be activated in conjunction with others in order to lead to a certain classification outcome. Reducing such “co-adapted features” promotes robustness of the network and has been observed to improve accuracy in some cases (but at times at the cost of longer training). The hypothesized reason for increased accuracy is thought to be due to the network behaving like an ensemble of networks. This might occur because the stochastic removal of features promotes multiple paths for classification.

## References

- [1] H. Chen *et al.* [MicroBooNE Collaboration], “Proposal for a New Experiment Using the Booster and NuMI Neutrino Beamlines: MicroBooNE,” FERMILAB-PROPOSAL-0974.
- [2] M. Antonello *et al.* [MicroBooNE and LAr1-ND and ICARUS-WA104 Collaborations], “A Proposal for a Three Detector Short-Baseline Neutrino Oscillation Program in the Fermilab Booster Neutrino Beam,” arXiv:1503.01520 [physics.ins-det].
- [3] R. Acciarri *et al.* [DUNE Collaboration], “Long-Baseline Neutrino Facility (LBNF) and Deep Underground Neutrino Experiment (DUNE) : Volume 1: The LBNF and DUNE Projects,” arXiv:1601.05471 [physics.ins-det].
- [4] Rene Brun and Fons Rademakers ROOT: An object oriented data analysis framework, Linux Journal 998July Issue 51, Metro Link Inc
- [5] E. Racah *et al.* Revealing Fundamental Physics from the Daya Bay Neutrino Experiment using Deep Neural Networks. *arXiv:1601.07621*.
- [6] A. Aurisano *et al.* A Convolutional Neural Network Neutrino Event Classifier. *arXiv:1604.01444*
- [7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009
- [8] Tsung-Yi Lin *et al.* Microsoft COCO: Common Objects in Context *arXiv:1405.0312*
- [9] PASCAL VOC Dataset, <http://host.robots.ox.ac.uk/pascal/VOC/index.html>
- [10] Navneet Dalal, Bill Triggs Histograms of Oriented Gradients for Human Detection. *CVPR* **1**, pages 886-893, 2005
- [11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229, 2013.
- [12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.
- [13] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.
- [14] Alex Krizhevsky *et al.* Imagenet Classification with Deep Convolutional Neural Networks. *NIPS* **25**, pages 1106–1114, 2012.
- [15] Christian Szegedy *et al.* Going Deeper with Convolutions. *arXiv 1409.4842*, 2014.
- [16] Ren, Shaoqing and He, Kaiming and Girshick, Ross and Sun, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks *NIPS* **28** pages 91-99, 2015.

- [17] MicroBooNE Collaboration. "Measurement of Average Drift Velocity Utilizing UV Laser Data." MicroBooNE Public Note 1009-PUB. <http://www-microboone.fnal.gov/publications/publicnotes/index.html>
- [18] MicroBooNE Collaboration. "TPC Signal Processing." MicroBooNE Public Note 1017-PUB. <http://www-microboone.fnal.gov/publications/publicnotes/index.html>
- [19] MicroBooNE Collaboration. "TPC Noise Filtering." MicroBooNE Public Note 1016-PUB. <http://www-microboone.fnal.gov/publications/publicnotes/index.html>
- [20] Software for LArTPC <http://larsoft.org>
- [21] MicroBooNE Software <https://cdcvs.fnal.gov/redmine/projects/uboonecode>
- [22] LArSoft release *redmine* **04.36.00** <https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/ReleaseNotes050800>
- [23] S. Agostinelli *et al.* GEANT4: A Simulation toolkit *NIM* **A506** 250-303, 2003
- [24] C. Andreopoulos *et al.* The GENIE Neutrino Monte Carlo Generator *NIM* **A614** 87-104, 2010
- [25] uboonecode release *redmine* **05.08.00** <https://cdcvs.fnal.gov/redmine/projects/uboonecode/wiki/ReleaseNotes050800>
- [26] Y. Jia *et al.* Caffe: Convolutional Architecture for Fast Feature Embedding *arXiv:1408.5093*, 2014
- [27] V. Genty *et al.* LArCV: Liquid Argon Computer Vision Software <https://github.com/LArbys/LArCV>
- [28] V. Genty *et al.* LArCV API adopted custom version of caffe <https://github.com/LArbys/caffe>
- [29] NVIDIA GeForce Titan X GPU Card Specification <http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications>
- [30] Y. LeCun *et al.* "Efficient BackProp." *Neural Networks: Tricks of the trade*. Springer 1998.
- [31] Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 4, 2012.
- [32] Szegedy *et al.* Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv:1602.07261v1*, 2016.
- [33] Kaiming He *et al.* Deep Residual Learning for Image Recognition. *arXiv:1512.03385*, 2015.
- [34] Srivastava *et al.* Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 1929-1958. 2014.

- [35] Ioffe et al. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proc. of the 32nd Intl. Conf. on Machine Learning 37, 2015.
- [36] Y. Jia et al. Caffe: Convolutional Architecture for Fast Feature Embedding *arXiv:1408.5093*, 2014
- [37] D. Silver et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489, 2016.
- [38] Collin, Genty, Terao, and Wongjirad. Particle Detection in LArTPC Data Using Deep Learning. MicroBooNE Internal Note DocDB-XX, 2016.
- [39] Genty, Terao and Wongjirad. LArCV: LArTPC Image Processing/Analysis Framework for Deep Learning <http://microboone-docdb.fnal.gov:8080/cgi-bin/ShowDocument?docid=5847>
- [40] Zheng et al. Improving the Robustness of Deep Neural Networks via Stability Training. *arXiv:1604.04326v1*, 2016. <http://arxiv.org/pdf/1604.04326v1.pdf>