CVN Deep Learning in HEP: Feature Selection in Minimal Bias NOvA Data

Buzz Walter

May 10, 2022

Abstract

Big data continues to become more prevalent in high energy physics (HEP) due to the needs of experimental design. Many experiments, such as NOvA, collect cosmic ray data outside of the primary role of the experiment, and they could be used to investigate new physics. These data are unfiltered and would require more attention than there are resources for, so it's critical that state-of-the-art data analysis techniques be employed to explore it. Here we investigate using a convolutional autoencoder on simulated data and report the best models.

1 Introduction

Computer vision has achieved great heights since convolutional vision networks (CVNs) have taken over in the image recognition domain, and this success hasn't gone unseen in other disciplines, experimental particle physics included. Aurisano et al [1] showed that deep CVNs with narrow widths are both computationally efficient and effective in classifying neutrino interactions in the NuMI Off-Axis ν_e Appearance (NOvA) experiment. Growth in computer vision since the AlexNet model's breakthrough in 2012 has only continued, so its becoming more natural to turn to these methods for solutions when we want to simplify our analysis pipeline.

Identifying rare or unexpected events in experimental physics is the sole purpose of a number of experiments, but many other efforts such as those in NOvA are done in tandem with their main objective. While NOvA focuses on neutrino oscillations, those 10 μs bursts of neutrinos which arrive approximately every 2 s only take up a fraction of the time the far detector is active. The detector was designed to be sensitive to other events such as a supernova signal or other cosmic ray data. While a supernova signal would swiftly be recorded due to both its significance and SNEWS, other cosmic ray data involved in new exciting physics might go unseen if it isn't tended to.

Using the Neutrinos at the Main Injector (NuMI) beam, the NOvA experiment

hopes to discover valuable information about matter - antimatter asymmetry in the universe. Neutrinos are a small fundamental particle in the Standard Model of particle physics. They interact via one of the few forces of our world, the weak interaction. This interaction has the property that it violates the otherwise common symmetry of matter and anti-matter in interactions. Our world is overwhelmingly composed of matter rather than antimatter, so in trying to understand this fact, we are inevitably forced to investigate neutrino interactions in further detail. One process which characterizes this matter - antimatter asymmetry is that of flavor mixing in neutrinos. Neutrinos come in three different types, all with their own antiparticles. When a neutrino travels across distances, whether that be in a vacuum or through matter, they evolve through a combination of the three separate states as though they weren't ever distinctly one of these exact flavor states. NOvA employs two separate detectors, the Near Detector and Far Detector, one located at Fermilab and the other in Ash River, Minnesota, 810 km away respectively. This forces the NuMI beam created at Fermilab to undergo these oscillations. The changes in the composition of the beam are measured at the Far Detector, and the goal is to hone down key parameters which characterize this mixing of neutrino flavors.

Unsupervised learning was originally used in place of supervised learning simply due to the inability to process large data sets, but ever since the revolution in GPU computing, virtually all ML models are trained using supervised learning. It still finds use however in tasks such as anomaly detection where the data contain entirely unexpected events to the model architect or the size of the dataset is incredibly limited. Here, we arguably find ourselves in the latter position if we take an experiment first approach trying to steer away from theoretical bias. Ideally the lofty goal with an unsupervised approach is to let the model establish features on its own that separate the data.

There are many approaches which can be taken in unsupervised learning, but the use of a clustering algorithm alongside an autoencoder was chosen due to its simplicity and similarity to previous successful CVN approaches in HEP. An autoencoder is a machine learning algorithm which uses the input data as the output target to regress to. It is often used in getting rid of noise in data, but is also applicable to feature engineering. In feature engineering, one tries to develop the best mapping to distribute the data according to the task the modeler wishes to solve. In other words, if we look at the data in a certain representation, then it might be easier to see the characteristic we're searching for. The features, which are represented by the mappings we're interested in, are located within the bottleneck, or lower dimensional representation, within the autoencoder. After training, the data represented according to this mapping can be used as the input to other standard unsupervised methods.

In the case of image data, the autoencoder naturally takes the form of a convolution autoencoder (CVA) just in the same way that these convolutional operations, which are central to CVNs, are effective in classification of image data.

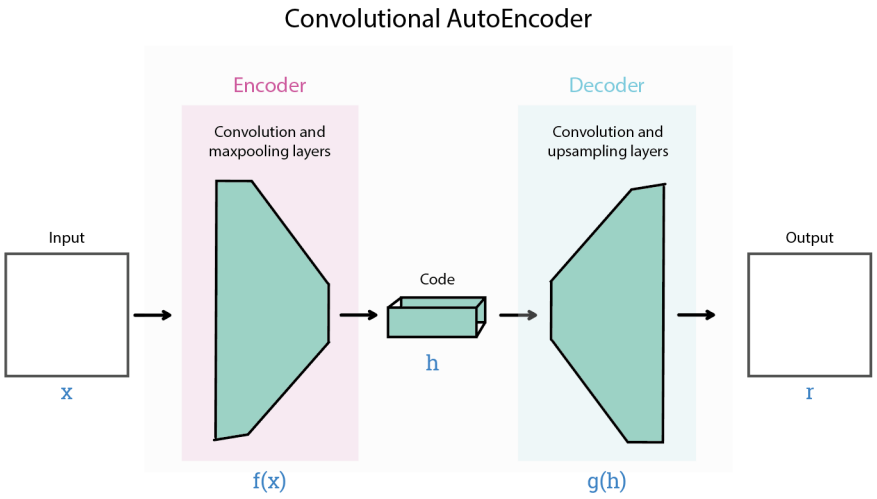

Figure 1: A representation of a CVA which encodes the data, x, via f to h. The training process requires decoding via g.

One of the nice properties of convolutions is that they have a built in invariance to object location in the images, and the features which they often find are interpretable as geometric characteristics. These both lend themselves to the feature engineering process. Fig. 1 depicts this process. At the bottleneck the image data becomes reduced in dimension and denoised which produces a better representation of the input data for the purposes of classification.

2 Deep Learning Theory

2.1 Basic Feed-Forward Network

A general feed-forward neural network can be described as a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. The input data $x \in \mathbb{R}^n$ gets passed through a series of linear functions followed by non linear activation functions which are non linear functions that have derivatives that stabilize far from the origin. Let the data have k samples. For example, for a single feed forward network, we have

$$\begin{cases} Z = h(XW) \\ Y = ZV \end{cases} \tag{1}$$

where X is a $k \times n+1$ matrix of the data appended with a column of ones, W is a $n+1 \times j$ matrix representing the weights associated with the input layer, and likewise V is a $j+1 \times m$ weight matrix. The nonlinear activation function $h(z)$ is applied across the whole matrix XW and appended with a column of ones to form Z, a $k \times j+1$ matrix. The output we get is then $k \times m$. By introducing

a loss function, $l(x, t)$ to regress to our target data $t \in \mathbb{R}^m$, we can train the model via Backpropogation.

Backpropogation updates the weights according to a chosen method, stochastic gradient descent (SGD) or adaptive moment estimation (Adam) among others.

$$\begin{cases} V_{j,m} \longleftarrow V_{j,m} - \rho_V \frac{\partial L}{\partial V_{j,m}} \\ W_{n,j} \longleftarrow W_{n,j} - \rho_W \frac{\partial L}{\partial W_{n,j}} \end{cases} \tag{2}$$

To summarize the formalities in plain english, a feed-forward network approximates the target data which one intends to predict from the input data. The error by which it is off is then used to parametrize the training procedure. To train the model, we use Backpropagation which updates the parameters of the model with the strict goal of minimizing the error by which the model is off.

2.2 Convolutional Networks

As far as image input is concerned, the data are indexed by two separate axes of pixel coordinates, so in applying the images to a network, they are first flattened. For example, if one uses a 32×32 image, then $n = 1024$. Increasing the depth only adds more cases to Eq. 1 and Eq. 2. In the instance of CVNs, the network gets divided into two sections. First, data passes through convolutional layers which adhere to a slightly different mathematical operation in the first case of Eq. 1, namely a discrete convolution

$$(f * g)_{p,q,r} = \sum_i \sum_j f_{i,j,r} g_{p+i,q+j} \tag{3}$$

where the (p, q) pixel is operated on by the patch f which associates the adjacent pixels via the indices i, j, and r indexes the output channels from the convolutional layer. Again, in practice this operation is accommodated to flattened data to facilitate the methods described above. The weights for these layers are simply the patch's elements. A nonlinear function is still applied in the same manner as before in the hidden layer.

After passing through the convolutional layers the data are output to a regular feed-forward section, which is subject to the whole host of model architectures that have been covered in many textbooks and papers predating CVNs.

2.3 Convolutional Autoencoders

Autoencoders take the implementation in Sec. 2.1 with $m = n$ in the mapping as well as $t = x$. This means the output representation in Fig. 1 is meant to best approximate the data itself. In doing so the bottleneck portion represented by h serves to encode the data. This process has some limitations on the networks that can be used since the output must end up with the same dimensionality as

the input. Furthermore, the encoding must reduce the dimension of the pixel coordinates of the data at the bottleneck, or code layer, which is at the center of the autoencoder.

In implementing a CVA on image data we use the method mentioned in Sec. 2.2. Specifically we only use the first section of this approach, the convolutional layers themselves. These serve as the structure for the encoding f. Again, the encoding must form a bottleneck to reduce the dimension of the data. In this case, that means f must reduce the pixel dimensions. There are a variety of ways to do this, however the real challenge is the process of decoding since there isn't a straightforward way of inverting a convolution. This is a mathematical limitation due to the fact that convolutions in general are not invertible.

The convolutional transpose is the operation used to upsample the data. Similar to how a convolution applies a patch to a subset of an image around a pixel to produce a single number, the convolutional transpose takes a single pixel and associates a patch of numbers with it through scalar multiplication of the patch, treating it as a vector in a vector space. In the same manner as the convolutional operation in Sec. 2.2

$$(f *^t g)_{i,j,k} = \sum_r f_{i,j,k,r} g_{p,q,r}. \tag{4}$$

Now, associated with the (p, q) pixel of the input feature r, we have the number of i, j combinations available due to the convolutional patch times the number of output channels specified by the layer. Using these free parameters, we can construct layers that will expand the dimension of the mapping back up to its original single channel form.

2.4 k-Means Clustering

Given a set, S, of n dimensional data points, k-means clustering aims to partition the set S into k sets S_i. This is done by picking mean based centers for each cluster that will minimize distances from the cluster mean, or in the statistical sense, the within cluster variance, i.e.

$$\text{argmin}_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \tag{5}$$

where μ_i is the mean of the ith subset. The algorithm achieves this by first randomly selecting k centers from the dataset and forming the sets S_i by selecting which center a given data point is closest to. Then the means of those data sets are computed and used as the new centers. This process is repeated for an appropriate number of iterations to converge on clusters. The convergence is not guaranteed for all data sets since some are not uniquely separable by this metric, and even when a dataset is mostly separable by this metric, it doesn't necessarily produce the exact same answer every time simply due to the random choice of the initial centers.

3 Application to NOvA Data

3.1 Detector Design

NOvA consists of an approximately 600 tonn Near Detector placed 1 km from the neutrino source, and a 14 kton Far Detector located 810 km away in Ash River, Minnesota. A roughly 10 μs, 750 kW beam of muon neutrinos is produced from the NUMI beam line, and the Far Detector measures the number of muon neutrinos and electron neutrinos characterizing the oscillation which has occurred during the traversal while the Near Detector characterizes the beam [2]. NOvA is used for a variety of other studies besides neutrino flavor mixing, e.g. the Near Detector is also used for separate studies on ν - nucleus interactions.

The Far Detector which can be seen in Fig. 2 consists of a number PVC cells of dimension 3.9 cm $\times$ 6.6 cm $\times$ 15.5 m filled with scintillator and an optical fiber cable. A single pixel represents the deposition in a single cell due to the charged particle passing through the scintillator. The optical fiber is attached to an avalanche photo-diode array which collects the signal in a cell. The cells are calibrated to get a good estimate of the true deposited energy. Fig. 2 shows an 8-cell wide module which alternates horizontal and vertical views, X-Z and Y-Z, two separate $2D$ representation of the interaction.

In full, these are combined to form slices which are clusters of energy deposition in space and time across 80 cells and 100 planes deep. An interaction represented in this manner can be seen in Fig. 3. Effectively, the readout is a 80×100 pixel map with a single 8-bit channel for energy deposition.

3.2 Model Data

The approach used to show that unsupervised learning for the detector data could be done required some assumptions to be made since the actual data this would be applied to would require more effort to manipulate than time would allow for this term. For computational ease given that GPU computation was not available for this project, we had to generate $32 \times 32 \times 1$ shower tracks and curvilinear tracks. These can be seen depicted in Fig. 4.

To generate the simulated sample, first positive gaussian sampled noise was generated. This was then added to tracks which were both linear and curved. The deposition was given a random exponential decay to mimic the energy deposition as the track loses kinetic energy. For many tracks this is negligible. Also random decay times were assigned to produce a subset of curves with abrupt terminations. These tracks were all produced with the same initial condition i.e. starting at the $(0, 0)$ pixel moving along a single axis. The resulting curvilinear tracks were however rotated according to a uniform sampling of angles from 0 to 2π to mimic the roughly isotropic nature of non-beam spill data.

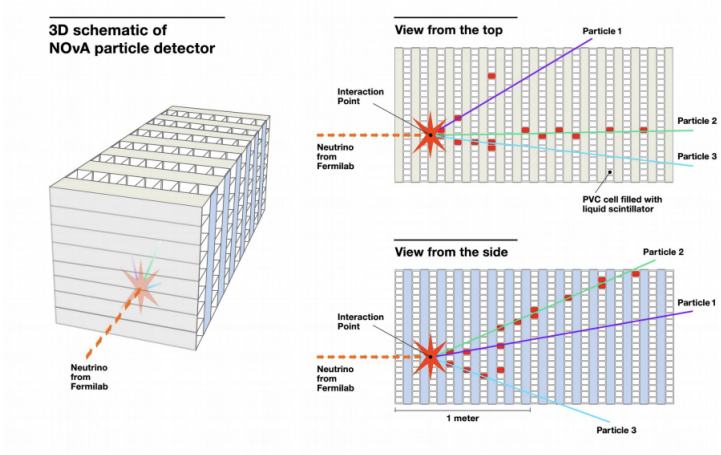

Figure 2: The NOvA detector depicted with alternating views from interleaved parallel arrays of cells. The actual correspondence between the $2D$ readout and the cells picking up energy deposition can be seen here.

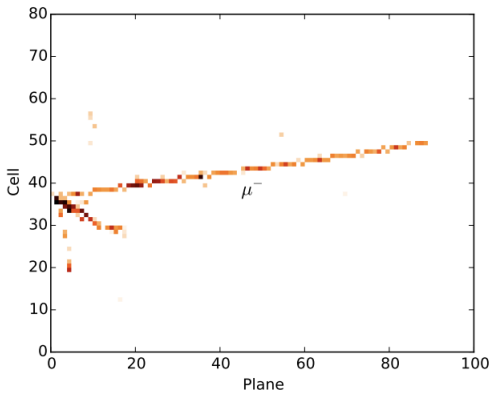

Figure 3: A slice from the NOvA detector readout for one view. Pixels are representative of a cell at a certain plane.

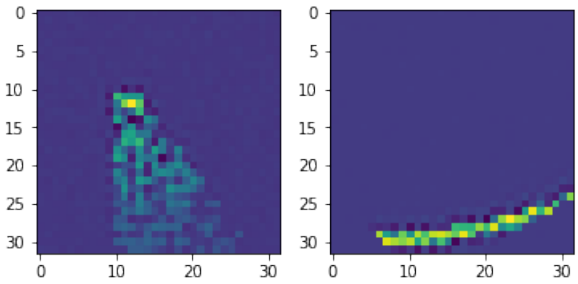

Figure 4: Simulated showers and tracks are shown as $32 \times 32 \times 1$ pixel maps.

Similarly with the shower tracks, we generated the cone axial vector representing the shower in a standard direction at a random point in the pixel plane. Random hits were then allowed at pixels within the cone, and the energy deposition decayed exponentially with the distance from the vertex. Then we rotated the samples according to a uniform distribution of angles.

3.3 Model Architecture

The model architectures followed a general recipe for dimensionality reduction and then decoding. As can be seen in Fig. 5, at each layer in the encoding process 3×3 patch filters are used with a stride of 1 to preserve the pixel dimensions along with a variable number of features. Ideally the feature number is reduced at the bottleneck. To get the dimensionality reduction in pixels every 3×3 patch is followed by a 2 max pooling layer. This halved the width of the image at every other layer. Ultimately this limits the maximum number of 3×3 layers to 5. As far as the nonlinear activation functions for CVA layers go, the more common ReLU or tanh were available for all of the layers except for the last one which was given a softmax activation function to preserve the standardized output from 0 to 1.

For the decoding process, convolutional transposes of patch size 2×2 and stride of 2 successively double the pixel dimensions. The associated feature numbers are assigned in the inverse order to the encoding process so that we end up with an output image of $32 \times 32 \times 1$. For regression we use mean squared error (MSE) as our loss function. A container function that helps the forward pass was used to store output at various depths in the autoencoder. Using the bottleneck output to the k-means clustering algorithm with $k = 3$ served to attempt to separate noise and the two separate main classes of the sample. This algorithm was applied to each feature at the bottleneck to produce different clustering.

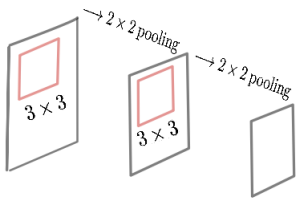

Figure 5: A depiction of the encoding process for the CVA applied to the pixel map. Several 3×3 convolution patches with 1×1 padding are followed successively by 2×2 max pooling.

4 Results

A number of models were constructed to test performance on a test dataset formed from the simulation process. This data were also generated from the simulation process, which traditionally introduces bias into model performance, however for the purposes of our experiment using the CVA, we are using an unsupervised approach to show that this model functions well enough to be extended to actual data. The model architectures in this thesis are suited to this sort of approach since unsupervised training is geared towards small data sets. The autoencoder was trained on 12119 images of showers and 10000 curvilinear tracks, which can be seen in the Appendix, produced as described in Sec. 3.3. The random number of showers roughly around 10000 was due to the fact that a certain number of the samples generated through this method needed to be removed due to edge cases in the simulation producing spurious data as can be seen in Fig. 6. This data was standardized before training by subtracting the mean and dividing by the standard deviation. This is a common preprocessing step which facilitates stability in the numerical procedures.

After the initial preprocessing of the data, the training and testing were done via test scripts that automated tests like the ones seen in Fig. 7. Upon testing another 10000 images of noise were included. A class based design was used to generate the models in pytorch, and a container for the forward pass was used to extract the output at the bottleneck layer. The class architecture allowed for using different optimizing techniques like Adam and SGD as well as the different nonlinear activation functions for the interior autoencoder layers. Through looking at the different non linear activation function available for the model, it was found that using tanh for the interior layers' activation function and sgd for the optimization technique produced larger separation in the clustering. Also

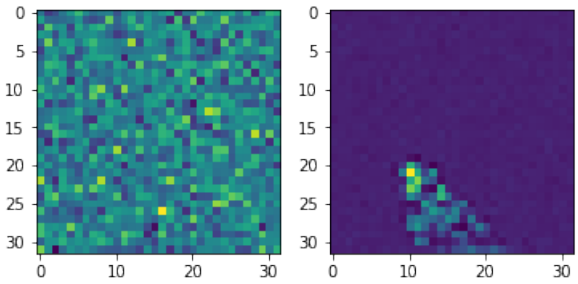

Figure 6: Two simulation images generated from the original simulation procedure. The image on the left is thought to have resulted from a poor random choice of initials conditions and rotation angle.

the looking at the number of epochs we found that 100 gave better convergence than 75, 50, or 25 epochs.

The two key architecture tests seen in Fig. 7 explored the scaling of the number of features and the depth of the CVA. From the clustering, we consistently see that noise is picked up over the full dataset which is expected for an autoencoder, however the shower samples get confused for noise at just about every level of the the feature number test. The best performing features from each of these architecture is reported with the hope of possibly combining certain ones to further increase separation, so the trend we see in the reduction of curves confused for noise isn't that convincing of the value of increased feature number. Nonetheless, It was determined that for both sampling purposes and at least equivalent performance, using a larger number of features is preferred. In practice, these computations would be done on GPUs which could very easily accommodate scaling of the number of features. As far as depth was concerned, it was very clear that going for deeper layers improved performance, especially in moving away from the confusion of showers for noise. There seemed to be a general trend that if we used a very shallow architecture such as [5], it seemed to better separate the curvilinear tracks where 38.6% were confused while the deeper networks separated showers more effectively.

The best performance we were able to get was from the $[80, 50, 30, 15, 5]$ architecture. This produced a number of features which performed well on certain task. In particular, the third feature of this architecture manages to get the noise selected percentage of showers down to 4%, however in this representation 38% of curvilinear tracks were confused. The best performance on curvilinear track though was the slightly different model of $[35, 20, 10, 4, 2]$ which only confused roughly 19.6% of the curved tracks for noise. Besides noise confusion, the other clustering was less concerning because that's where interesting features can be picked up on. For example, it was found with the curve clustering that,

aside from those which were confused with noise, one cluster always ended up containing tracks with higher curvature, so some geometric features were being picked up on by the autoencoder. This is a common result for networks however they normally occur only at shallow layers in the network and not quite as deep as the bottleneck. The fact that the encoded data was processed through $k-$means could also be contributing to this though.

These results suggest that the deeper networks with a greater number of features seem to produce the most variety and separation in the data. By looking at combining features it could be feasible to tease out certain aspects of the data. For example, one could intend to use the $[80, 50, 30, 15, 5]$ architecture along with the $[35, 20, 10, 4, 2]$ and explore their intersections and compliments to go further in the direction of getting great performance and separate geometries rather than just a single one.

5 Conclusion & Future Outlooks

Several different models were explored in trying to develop an unsupervised approach to identifying features in an unbiased way. This resulted in CVA architectures which, when combined with k-means clustering, produced features that performed well on different geometries of the data. In particular the $[80, 50, 30, 15, 5]$ layer architecture did well to separate the shower samples while the $[35, 20, 10, 4, 2]$ layer architecture performed better of curved tracks. There still is room to improve on exploring the curvilinear samples since the best performance still confused 19.6% of tracks for noise. Getting access to a GPU could greatly improve the experimental efficacy of this endeavor since different models could be explored. Also, for sake of investigation, certain aspects of the autoencoder were constrained. Some of these constraints such as the patch size and structure of the convolutional transposes could be altered. Lastly, this is just one of many different unsupervised approaches that can be used, so it would be interesting to see other methods used in both the feature engineering and the classification process.

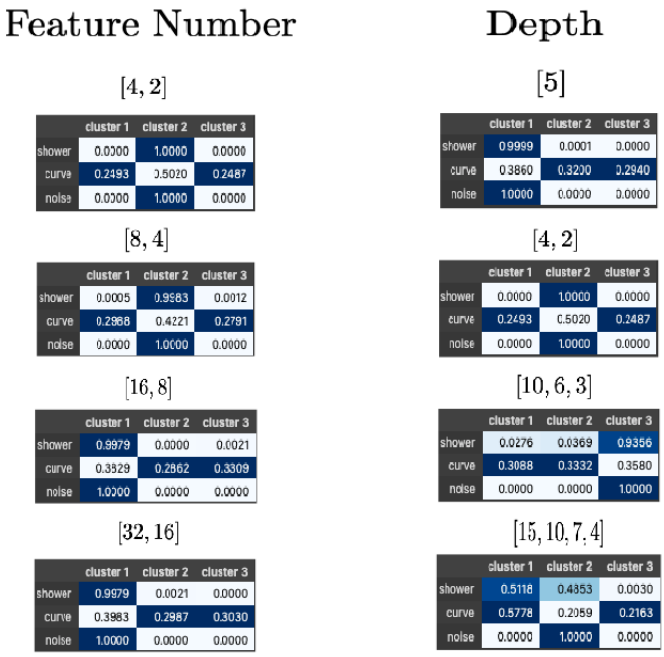

Figure 7: Two tests on the scaling of the number of features and the depth of the autoencoder were conducted. A confusion matrix is produced to show how the clusters were grouped and what their actual composition was according to the dataset. The labeling gives the order of the output channels, or features, at any given point in encoding process of the autoencoder in the 3×3 layers. We can see in the upper left table and second to upper right table that showers and noise were entirely classified together in the same cluster which was a problem that certain features in the architecture would yield.

References

[1] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. D. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa and P. Vahle, "A Convolutional Neural Network Neutrino Event Classifier," JINST **11**, no.09, P09001 (2016) doi:10.1088/1748-0221/11/09/P09001 [arXiv:1604.01444 [hep-ex]].

[2] D. S. Ayres *et al.* [NOvA], "The NOvA Technical Design Report," doi:10.2172/935497

[3] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", Neural Networks 2 (1989) 359 - 366.

[4] Goodfellow *et al.*, "Deep Learning", MIT Press, 2016.

[5] F. Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms." Spartan Books, 1961.

[6] G. James, D. Witten, T. Hastie, R. Tibshirani. "An Introduction to Statistical Learning : with Applications in R." New York: Springer, 2013.

Appendix

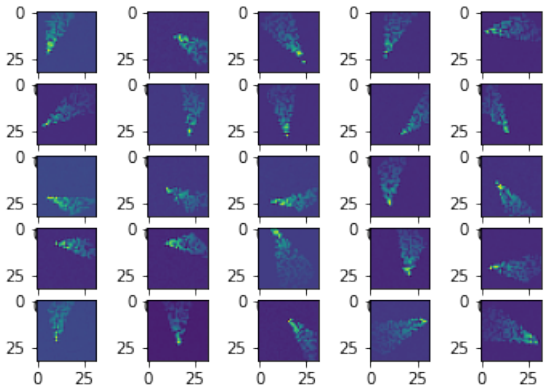

Figure 8: A set of simulated showers. Roughly the same geometry can be seen with varied vertices, angles of rotation, and pixel coordinates for deposition.

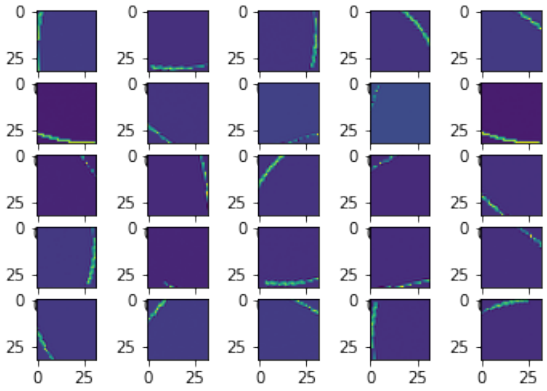

Figure 9: A set of curvilinear tracks. Linear tracks are shown amongst other tracks with a spectrum of curvatures.