DUNE VDTPC Raw Signal Waveform Analysis using Machine Learning

Malachi Clark, University of Wisconsin - Madison, Fermilab SIST

May 2022 - August 2022

Abstract: Inside of the DUNE Vertical Drift Time Projection Chamber of the SURF far detector, there are plans for photon detectors to be used as triggers for particle drift mappings to be recorded. Using data from a DUNE photon detector prototype, we aimed at training a machine learning model to be able to classify raw signal vs. noise waveforms as well as give the predictions for the number of photo-electrons captured by the photon detector. We ended the project by adapting a model, 1D-CNN ROI Finder, and getting a 98.98% accuracy with a loss of 0.0223 on the classification problem. We didn't gain as much success with the regression model gaining a best RMSE of 1.05837; ended up switching that model to a multi-class classification and ran into some unexpected issues after getting only a 55% accuracy.

1 Background

1.1 The Experiment

The experiment that I worked under was the Deep Underground Neutrino Experiment, or DUNE. In DUNE a high intensity neutrino beam will be sent from Fermilab for detection by the Sanford Underground Research Facility (SURF) in South Dakota [Fit]. This experiment will be able to tell researchers and scientists a lot more about neutrinos, which we currently know little about. It has goals of giving us more information about neutron stars and black holes, the origin of matter, and the unification of nature's forces [LLC].[1] Inside of the far detector in South Dakota, there are plans for there to be four chambers, one of which being the Vertical Drift Time Projection Chamber (VDTPC). The VDTPC is a Liquid Argon Time Projection Chamber (LArTPC) where particles and electrons from neutrino interactions will be drifted for mappings and data collection. For that data to be collected, photon detectors are used as triggers. The current prototype of the photon detectors that will be placed inside of the DUNE VDTPC is where I get my data from for analysis.

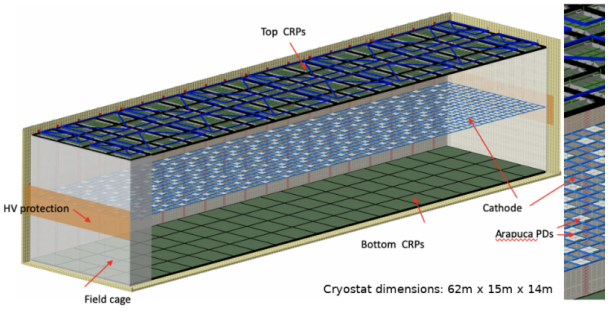

Figure 1: This is an image of a Vertical Drift LArTPC. This is where the data my model will do it's analysis on will come from when DUNE is up and running. It works similarly to the Horizontal Drift LArTPC but the drift occurs vertically instead of horizontally.

1.2 Motivation

The goal behind this project was to apply a machine learning algorithm to photo-electron predictions and detections. I was to create a model to classify waveforms[2] as signal or noise and another model to predict the numerical amounts of photo-electrons (PEs) captured within a waveform. Although applying numerical algorithms and manipulations would be similar and more precise, they wouldn't be as efficient as a machine learning model prediction, and the motivation behind this project was to make signal detections and predictions more efficient. There is no need for our numbers to be extremely precise for the regression model due to the numbers not being used for scientific calculations, so efficiency was more favorable than accuracy.

[1] If you want to learn more about the experiment and how it works visit the DUNE website linked in the references.
[2] This refers to the shape of the graph in a 2D format seen in Fig. 2

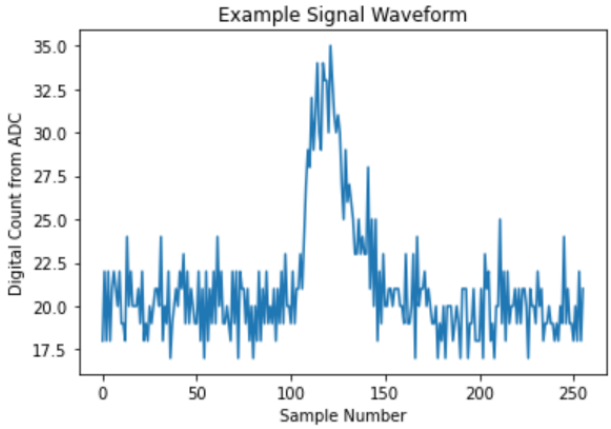

Figure 2: This is an example raw signal waveform that I used to train, validate, and test my model. The x-axis is the Sample Number which has a value of about 13 ns and the y-axis is the Digital Count from the ADC which is around 2 V/($2^{16} bits$).

2 Methods & Implementation

2.1 Data Collection

My data comes from a prototype of a photon detector where an LED emulates what would occur within the far detector in terms of photon discharge, not to the same intensity but enough for our purposes. To dive deeper on the mechanics behind the prototype.

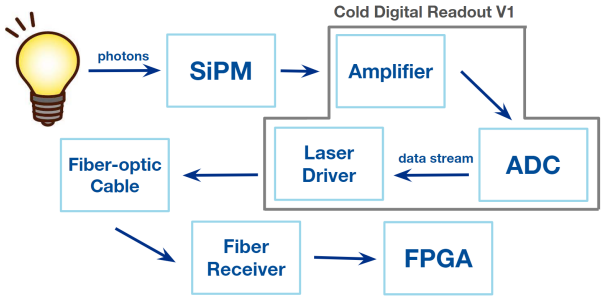

Figure 3: This is a representation of the prototype where I get my data from, showing how we go from an LED to our data being stored.

The LED is mounted inside of an ARAPUCA, a light tight enclosure, where the LED is given a pulse of electric current for the LED to light up and give off photons. Those photons are then detected by the Silicon Photo Multiplier (SiPM), passed to the Cold Digital Readout board, where the signal is amplified and an analog to digital conversion occurs and is data streamed into a laser driver, then outputs to a fiber-optic cable, routed to a fiber receiver and finally arrives at an FPGA board where the data is captured.

2.2 Classification Data Manipulations

The original data set has 153600 waveforms with variant PE values. However, in the data set, majority of the signals occur within the same sample number values which is not reminiscent of how the data would be retrieved once DUNE is up.

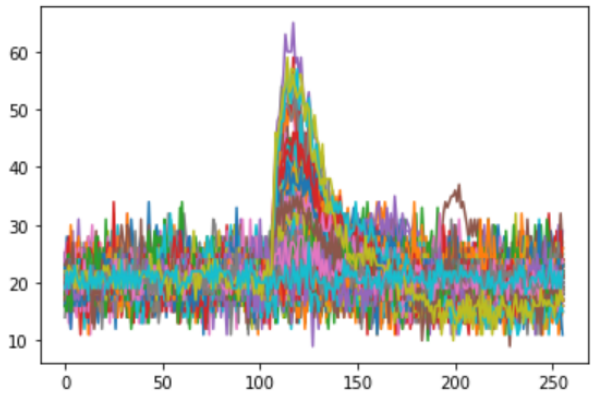

Figure 4: This is an image of 1000 random waveforms showing the bias in signal peak placement from the prototype.

Along with that, our data did not come separated, so there needed to be some specific choices made in determining the split and how we handle signal values. To solve the problem of having the peak signals show up in the same sample number indices, we rolled all of the values within the waveform a random number between 80 and -80, to ensure that we wouldn't have split peaks and would have random placed peaks, which is how the data from DUNE would look. Moving to our unseparated data issue, our waveforms have Gaussian noise which is assumed by the Non-Negative Least Squares (NNLS) method, which turns out to give the numerical predictions of PEs captured within the waveform. So I took all of the waveforms and ran them through the NNLS method, getting me the predictions for all the waveforms. Once I had those values, we arbitrarily determined that if the values passed a threshold of 0.45, they could be determined as a signal, because there may be some cases of signals where the detection could be split into two 0.45 bins for a variety of reasons. With that threshold, we now had a split of signals and noise, with signals representing 69497 waveforms and noise representing 84103 waveforms

within the complete data set and then we were able to label our data. With that we had dealt with the issue of having an unseparated data set.

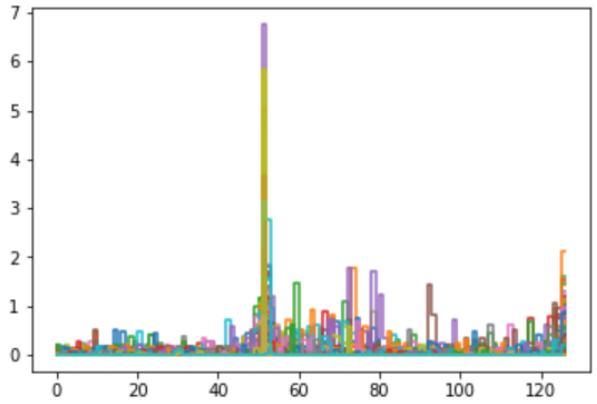

Figure 5: This is an image of 1000 random waveforms after the nnls transformation showing the PE amounts that were captured.

After shuffling our labeled data set and doing some other syntactical manipulations, we ran our model with a 65%, 20%, 15%, train, validation, and test split, respectively for the classification model.

2.3 1D-CNN ROI Finder - Classification

We sought out with the goal of creating our own model but found out there was a project running parallel to ours that had already created a model for this specific problem, however they lacked a working photon detection waveform data set at the time. We then applied that model to the classification problem of determining signal vs. noise. The model is named the 1D-CNN ROI Finder (One Dimensional Convolutional Neural Network Region of Interest Finder). The model is a sequential model, which just means that every layer is connected to one another. It consists of three 1D Convolution (CONV) layers: The first CONV block has 16 filters, ReLU activation, and max pooling, The second CONV block has 32 filters, ReLU activation, max pooling, and a dropout of 10%, the third CONV block has 64 filters, ReLU activation, max pooling, and a dropout of 20%. The model is then flattened, put through our dense function which has the sigmoid activation, getting us our noise or signal prediction.

- In a convolution layer the filters are pixels from the image of the waveform. The layer takes a window and scans it over the image. The window size comes from the amount of filters.
- ReLU activation takes any negative number and makes it 0, and any positive number is left as what it was.
- Max pooling takes a similar shape as the convolution layer where it is a window, but instead of applying a filter to the window, it just takes the max number from the number of values left from the filters.
- Dropout takes a percentage of outputs and sets them to 0, it helps to generalize the model.
- Flatten takes a multi-dimensional array and flattens it so it's only one dimensional. [ex: 5x5 to 1x5]
- Dense fully connects the layers together, giving us our neural network.

The optimizer that we use is Adam which uses two gradient descent momentums and RMSP. As well the loss we use is binary_crossentropy considering we have only two outputs for classification, noise and signal.

```
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_74 (Conv1D)           (None, 127, 16)           64
_________________________________________________________________
batch_normalization_10 (Batc (None, 127, 16)           64
_________________________________________________________________
max_pooling1d_48 (MaxPooling (None, 63, 16)            0
_________________________________________________________________
conv1d_75 (Conv1D)           (None, 30, 32)            2592
_________________________________________________________________
batch_normalization_11 (Batc (None, 30, 32)            128
_________________________________________________________________
max_pooling1d_49 (MaxPooling (None, 15, 32)            0
_________________________________________________________________
dropout_30 (Dropout)         (None, 15, 32)            0
_________________________________________________________________
conv1d_76 (Conv1D)           (None, 7, 64)             18496
_________________________________________________________________
batch_normalization_12 (Batc (None, 7, 64)             256
_________________________________________________________________
global_max_pooling1d_24 (Glo (None, 64)                0
_________________________________________________________________
dropout_31 (Dropout)         (None, 64)                0
_________________________________________________________________
flatten_24 (Flatten)         (None, 64)                0
_________________________________________________________________
reg_out (Dense)              (None, 1)                 65
=================================================================
Total params: 21,665
Trainable params: 21,441
Non-trainable params: 224
_________________________________________________________________
```

Figure 6: This shows the summary of the layers inside of the 1D CNN model that we ran our data set on.

2.4 Classification Results

The model worked extremely well on the first run of the model; it ran for 100 epochs out of 100. It had an accuracy of 98.98% and a loss of 0.0223. Below are the confusion matrices for the model being ran against the test data, the test data being data the

model has not seen before. In order to read these matrices some information is necessary. The 0 represents the label corresponding with noise, the 1 representing the label for a signal detection within the waveforms. When you see the box crossover between, for example, 0 on the y axis and 0 on the x-axis, it means that the model predicted there had no PE/signal detection and there actually was no signal present, showing a correct prediction. Or another example, if you see the box crossover of 0 on the y axis and 1 on the x-axis, it means that the model predicted there was a signal detected but there was actually no PE detected in the waveform, showing a false prediction.

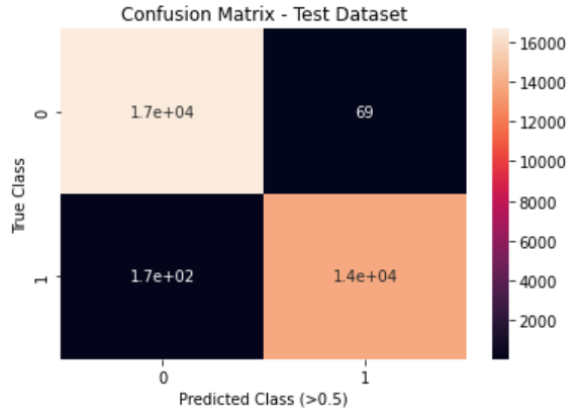

Figure 7: This is a confusion matrix where the model had a greater than 50% positivity rate in it's prediction. This CM has an average false positive rate of less than 1%

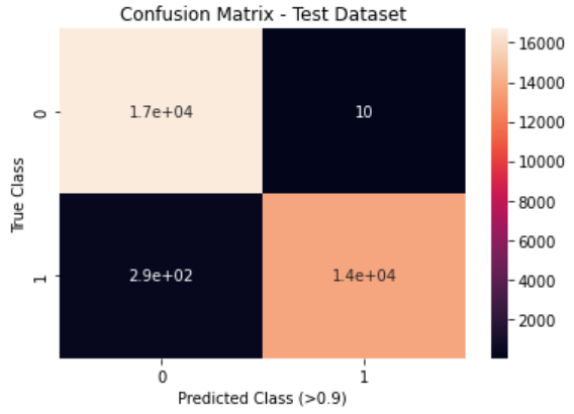

Figure 8: This is a confusion matrix where the model had a greater than 90% positivity rate in it's prediction. This CM also has an average false positive rate of less than 1%

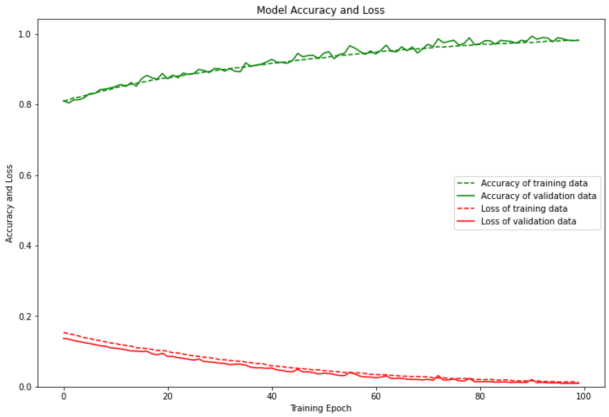

Figure 9: This plot shows the model accuracy and loss over the course of 100 epochs for the first run of the model. The first run of the 1D-CNN model on the data set ran for 100 epochs, gaining a 98.98% accuracy and a 0.0223 loss.

Our model was working extremely well on it's predictions gaining false positive rates of less than 1% consistently. After running the model a bunch of times and constantly seeing these results, we were shocked to see that the model was this accurate without any changes from the original 1D-CNN model[3]. We wanted to verify that there was nothing dubious going on within the model or the data set; what that means for us considering our model was so accurate, was to do everything possible to try and break the model.

2.5 Classification Verification

We went through a variety of changes within the model, the data set manipulations, as well as the processes we used for separation to verify everything was working properly. First we needed to verify that the things going wrong weren't happening in the algorithm, so in the upcoming confusion matrix, I trained the model on 5 signal waveforms and 5000 waveforms of noise, the result we got was really good because it meant that the model performed really well on noise and horribly on peaks which was expected. This lead us to believe that something was up with the data set or that our problem wasn't as complex as we believe and the model is just extremely accurate.

[3]To learn about the development and original motivation of the 1D CNN model creation, take a look at Ben Hawk's Presentation [Haw]

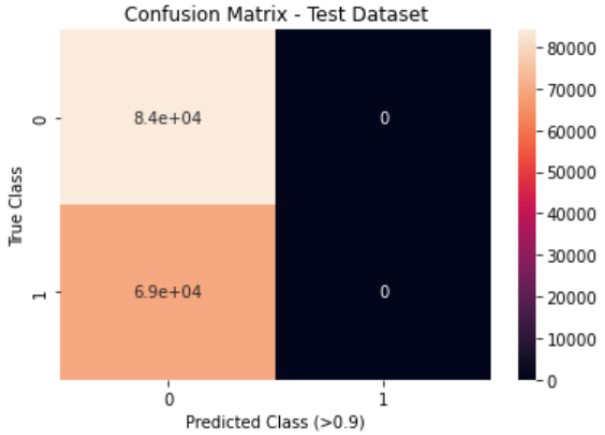

Figure 10: This is a confusion matrix where the model had a greater than 90% positivity rate in it's prediction. The model this CM comes from was trained on 5 signal waveforms and 5000 noise waveforms. The result represented in this CM is expected, verifying that the algorithm works properly.

We continued the model verification process by mislabeling half the the noise data as signals to see how that would change the results represented in the confusion matrix. We expected that the model would start to confuse some noise waveforms as signals. So we shouldn't see much change within the noise predictions, but a major shift in how positive the model is in it's prediction as well as the accuracy at which the model can predict signal waveforms.

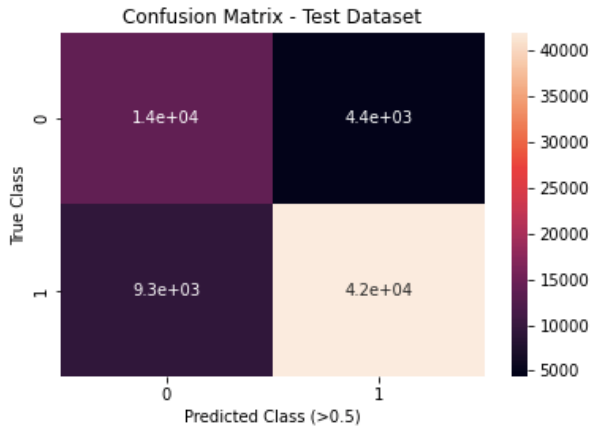

Figure 11: CM where the model prediction positivity was greater than 50%. If you look at the predictions of the noise and signal classifications, they get a little less accurate as they usually are. This is expected for the model.

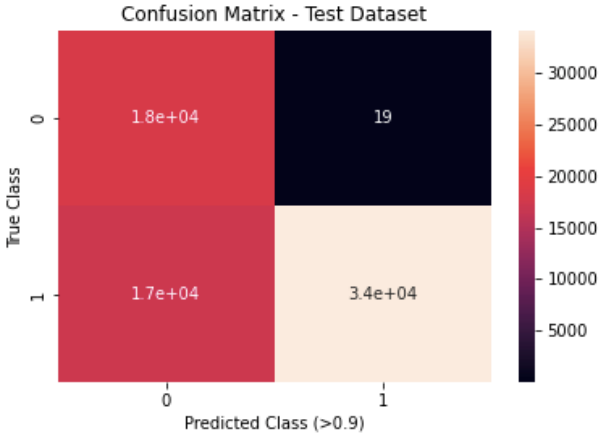

Figure 12: CM where the model prediction positivity was greater than 90%. If you look at the predictions of the noise and signal classifications, We get that the model is now unreliable in terms of classifying signals because some of the noise data was labeled as being a signal in training but in fact wasn't. This is expected for the model.

We then mislabeled half of the signal data as noise. We expect that the model should work extremely well at predicting noise waveforms but horrible at determining signal waveforms because the model is learning the completely wrong information in regards to signal waveforms. This should plummet the positivity rate of the prediction of the model in guessing signal waveforms.

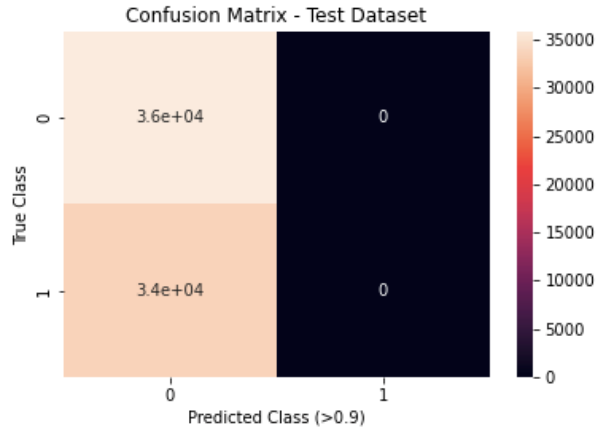

Figure 13: CM where the prediction be greater than 90%. The predictions for noise are perfect and the predictions for signals are completely inaccurate, not even getting one peak prediction correct. This is expected of the model, considering the model wasn't given a variety of data to train on.

After verifying that the 1D-CNN model itself

worked properly, we needed to make sure that nothing inside of the data set was going wrong either. In the upcoming confusion matrix we took the max values from the raw signals– the original values from the ADC and not from the NNLS separation– then we separated it using an arbitrary value of 9 to determine signals vs. noise. We did this because we had suspicions that the model was learning our data separation techniques instead of the images themselves. We didn't have any expectations in regards to the output but if we were to assume that the model was working properly, we should assume that this data separation technique should get us similar accuracy.

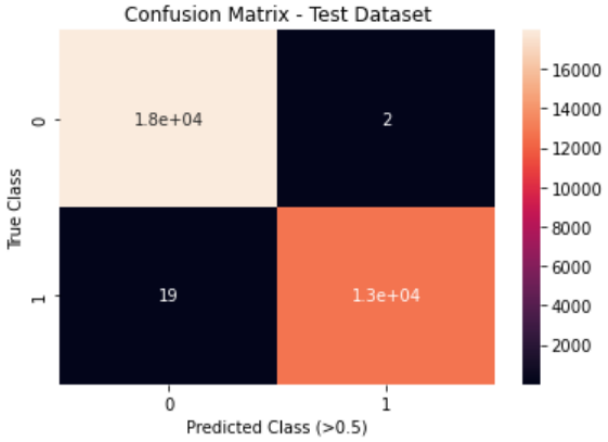

Figure 14: CM where the model prediction positivity was greater than 50%. The model does extremely well on the tests. It's presumed the model learned to take the maximums and learned the threshold.

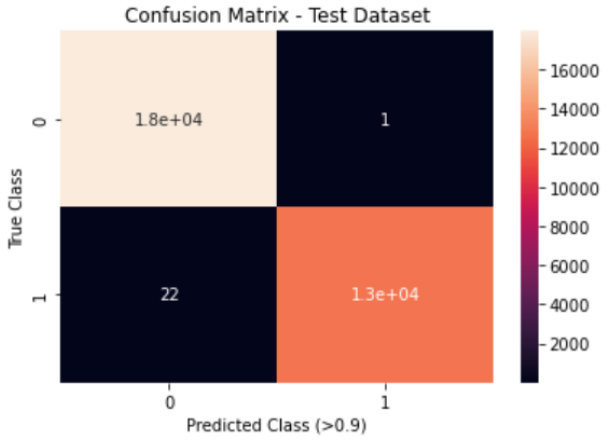

Figure 15: CM where the model prediction positivity was greater than 90%. Does pretty similar in regards to classification accuracy, so It can be deemed that it's not the NNLS manipulation.

We also did different variations of data splits and separations to test skews and how that would affect how the model predicted.

- We looked at the histogram of peaks and filtered out all the waveforms that had maxes of greater than 10 in the data set, we got very similar accuracy in the tests. This lead us to believe the accuracy was not coming from our data distribution.
- We did manipulations where we tested on varying, reasonable, data set sizes and skews, still receiving accuracy of greater than 95% consistently.
- For all of our tests besides this one, we scaled our data to ensure that there wasn't over-fitting as well as to ensure that large peak values wouldn't skew our data. For this test we unscaled our data so those peaks had a chance to skew and possibly cause over-fitting but we still received an accuracy of greater than 95%.

We even tried ridiculous variations to see if anything weird would start to reveal itself. In this next confusion matrix we trained our model on no signal waveforms and 10000 noise waveforms. We expect the model to have almost perfect accuracy in terms of noise predictions and completely incorrect accuracy in terms of signal predictions, and we see that represented in the matrix, extremely similar to our test earlier.

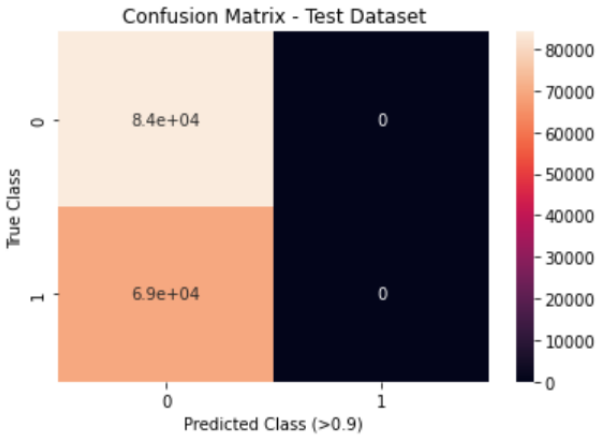

Figure 16: CM where the model prediction positivity was greater than 90%. Does exactly what the model should and does extremely well on classifying noise and absolutely horrible on classifying signals.

After completing all of these tests in different variations and changes, we determined that the model

performs at that accuracy because of all the steps that we took to ensure that the model would run the best it could. When they were developing 1D-CNN they took all the precise steps in making sure there would be no over-fitting, the model would be generalized, the model had a robust data set to train on, and that it was adaptable for similar problems.

2.6 Regression Data Manipulations

The goal of the regression was to be able to have the model predict the amount of photo-electrons captured inside of a waveform. We started the data set for the regression by taking the originally split signal waveform detections and appending some 0 PE waveforms so we could have the complete spectrum of PE amounts to train our regression model with. We then did the NNLS on those waveforms, summed up the values outputed in a standard range and used those values to label the waveforms with. We wanted to have the regression be able to do the NNLS for varying amounts of PE, but we were constrained by the range of signal data we had.

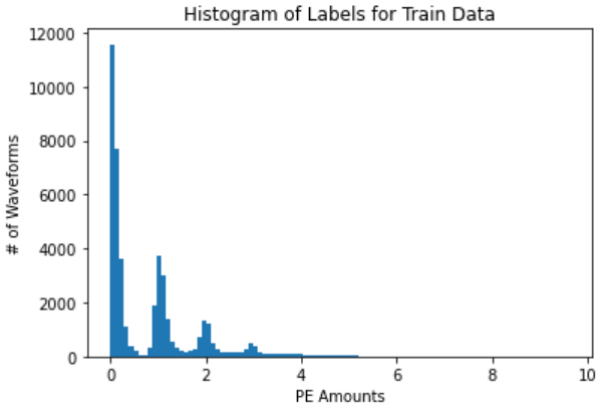

Figure 17: This is a histogram of the labels used for the train data set. There's a huge skew in the data towards waveforms with 0 PE detections. When looking at PE label amounts of 2+, the amount of waveforms showing those values completely drop off and are in the minority of the data set.

We noted that our data set came with a skew towards 0 PEs even without adding the 0 PE waveforms. So we created another data set without the 0 PE waveforms because we had fears that the model would skew it's predictions towards 0 due to waveforms with that value being so prevalent within the data set.

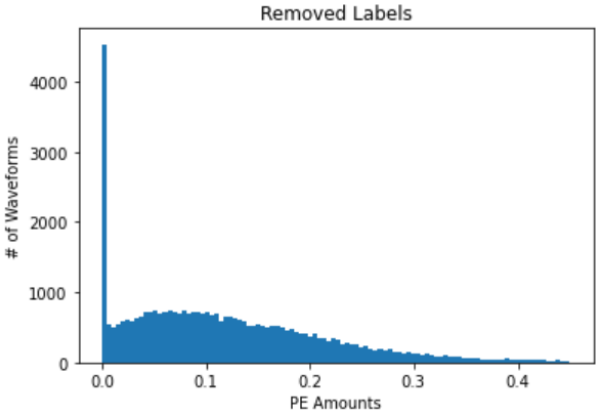

Figure 18: This histogram represents the range from 0 PE to 0.45 PE being captured by it's corresponding waveform. This histogram represents about 30,000 waveforms being removed from this specific data set.

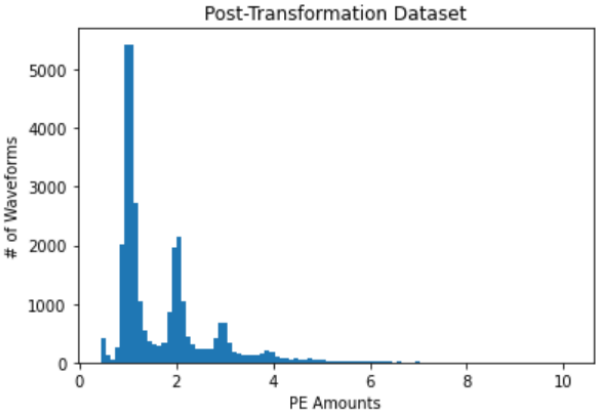

Figure 19: This histogram depicts the data set after removing the waveforms with PE values between 0 and 0.45. This data set will be used for some of training with our 1D-CNN regression model.

2.7 1D-CNN ROI Finder - Regression

For regression, the only things that were changed were the output layers, activation function, as well as the addition of hidden layers, which help the model learn more details within the image[4]. Specifically for the regression, we applied a linear output layer activation function so our prediction would just be the numerical prediction instead of the scaled value from 0 to 1, like it was in the classification. We also applied different amounts of hidden layers of varying neuron sizes depending on the specific model runs.

[4]If you are curious about the specifics behind hidden layers, the Deep Learning Book has a great chapter on neural networks. [GBC16]

Layer (type)	Output Shape	Param #
conv1d_9 (Conv1D)	(None, 127, 16)	64
max_pooling1d_6 (MaxPooling1	(None, 63, 16)	0
conv1d_10 (Conv1D)	(None, 30, 32)	2592
max_pooling1d_7 (MaxPooling1	(None, 15, 32)	0
dropout_6 (Dropout)	(None, 15, 32)	0
conv1d_11 (Conv1D)	(None, 7, 64)	18496
global_max_pooling1d_3 (Glob	(None, 64)	0
dropout_7 (Dropout)	(None, 64)	0
flatten_3 (Flatten)	(None, 64)	0
hidden_one (Dense)	(None, 64)	4160
hidden_two (Dense)	(None, 64)	4160
hidden_three (Dense)	(None, 32)	2080
hidden_four (Dense)	(None, 16)	528
reg_out (Dense)	(None, 1)	17

Total params: 32,097
Trainable params: 32,097
Non-trainable params: 0

Figure 20: This shows the summary of all the layers within the 1D-CNN regression variant.

2.8 Regression Results

When running our data set on the regression model, we found that our model ran quite poorly with a loss never reaching below 1... In some of the graphs the accuracy shows, but that metric doesn't correlate much with the actual accuracy of the model due to the model never being able to reach a prediction the exact same as the label, which are not in whole PE values but the decimal coming from the NNLS sums.

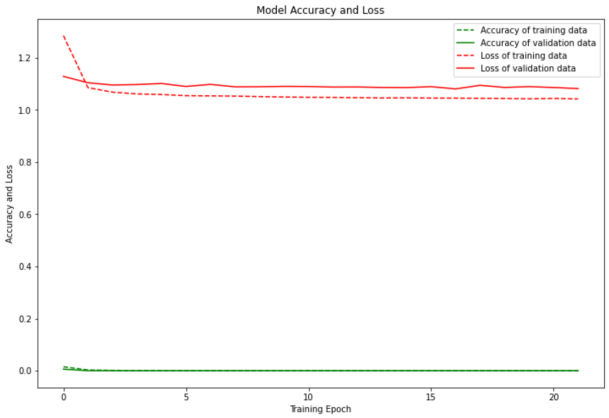

Figure 21: Within this plot we see the model accuracy and loss being plotted. This run was just base manipulations, the basic regression data set, as well as the model with no changes, in terms of hidden layer amounts. We got a Root Mean Squared Error (RMSE) of 1.05837

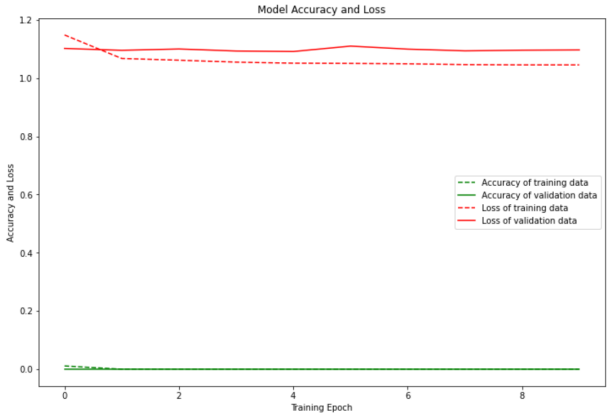

Figure 22: This run was the original regression data set along with a decrease in the amount of hidden layers, to see if there was an over-fitting problem. We see our RMSE increase to 1.11477, assuring it was not an over-fitting issue.

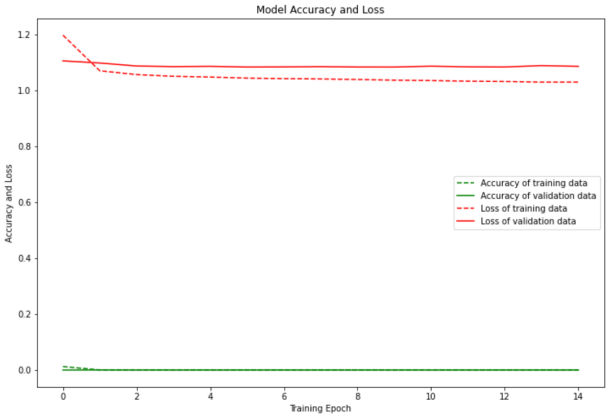

Figure 23: Within this plot we used the original model but the data set with 0 PE waveforms removed, because we saw our predictions were being skewed. We see our RMSE arrive at 1.08618, which is worse than the base set up.

After doing those tests, and variants of those tests, we see that our model was having an issue making predictions for PE values larger than 1. This makes sense considering our data sets are both skewed that way. We also see a fault occur in our decision in loss function, Mean Squared Error (MSE). We see a fault in this decision because when MSE tries to correct our model it wants to keep the lowest value on average. And when the data set is skewed towards values of 0 and 1, on average, a prediction in that range will be pretty close in terms of accuracy.

Looking at those results and coming to those conclusions, we determined that our data set wasn't ro-

bust enough for this specific problem, so we thought that considering our original classification model worked with extreme accuracy, it might do the same for a multi-class variation of this regression.

2.9 New Classification

In the new multi-class classification model we wanted to be able to classify the already separated signal waveforms into 3 bins: 0 PE, 1 PE, 2+ PEs. We took the data set from the original classification issue and only used the signal waveforms and separated them using their NNLS values. Our model reached its best accuracy of 55%, however an interesting issue arose when we saw our runs. We found that after each of our successful models had trained for their set amount of epochs, they arrived at the same precise accuracy independent of model shape. Unfortunately for us, we were unsure as to why this issue occurred and why the model trained the way it did on this specific data set.

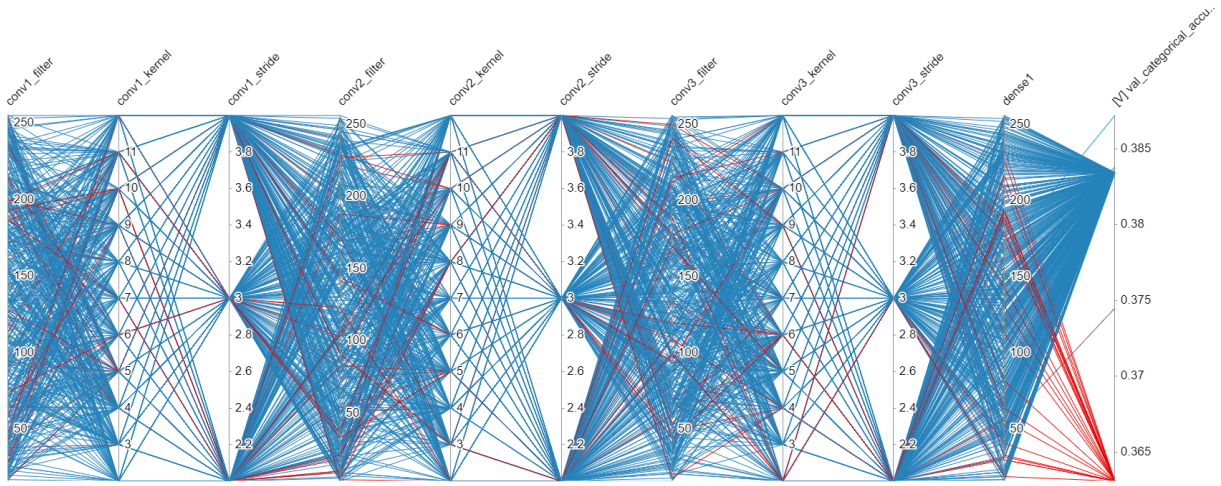

Figure 24: Depicted in this figure is around 1000 runs of the multi-class classification model where we see the interesting issue arise. In this specific plot we reach an accuracy of around 38%, about 5% above statistical randomness for this problem.

3 Future Work

One of the original goals of this project was to be able to use machine learning to predict the amount of photo-electrons captured within a waveform, and we were unable to successfully complete that. However, the next steps within this project are to take a look at the results of the new multi-class classification and figure out why the model performs the way that it does because that may tell us a lot about, not just the new classification model, but all the models we have built so far. But once that is looked at, the regression should be attempted again with a custom loss function, another model shape beyond 1D-CNN, possibly Residual Networks or Long Short-Term Memory Networks, and we should work on gaining and developing more signal data with larger PE values.

4 Acknowledgements

I would like to thank the SIST committee for giving me the opportunity to work for Fermilab under Jonathan Eisch in the Scientific Computing Division. I would like to thank Jonathan Eisch, for being an amazing supervisor, my collaborators, Ben Hawks, Hannah Parish, and Javier Campos, for their impactful help along the way, and my supportive mentor group and mentors.

References

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Fit] Rory Fitzpatrick. *5th UK LArTPC Software Analysis Workshop*. URL: https://indico.cern.ch/event/835190/contributions/3613901/attachments/1940522/3217390/fitzpatrickNNN19.pdf.

[Haw] Ben Hawks. *1D-CNN ROI Finder for LArTPC Waveforms*. URL: https://indico.fnal.gov/event/54666/contributions/242406/attachments/155674/202784/1D-CNN%20ROI%20Finder%20for%20LArTPC%20Waveforms%20-%20AI%20for%20Neturinos-1.pdf.

[LLC] Fermilab Research Alliance LLC. *DUNE at LBNF: Science Goals*. URL: https://lbnf-dune.fnal.gov/about/science-goals/.