**July 2018 Interim Report – Brendon Matusch**

**Summary**

This project is an effort to optimize discrimination between alpha particles and neutrons, using machine learning, based on various data collected during the PICO-60 experiment. Since I started work on this project in late June, I have developed several deep learning techniques for this task, and compared their effectiveness to the neural network applied in the PICO-60 paper, as well as the Acoustic Parameter (AP) frequency analysis function, which is the main tool used for this task in the PICO-60 project.

**Key Results**

I have tested many different techniques and configurations. On this page, I have highlighted four of the most successful, starting very similar to the neural network analysis in the PICO-60 paper and progressively making use of lower-level data representations to achieve higher performance. The class-wise standard deviation is used as a performance metric; this is defined on page 2.

Based on this metric, **performance was improved 49% relative to the PICO-60 neural network, and 45% relative to AP.** In addition, **performance was improved 40% relative to AP on a data set without any fiducial, pressure, or audio wall cuts.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Neural network architecture** | **Fiducial/pressure/audio wall cuts applied?** | **Input data format** | **Performance compared to PICO-60 NN** | **Performance compared to AP** |
| 2-layer perceptron with 50% dropout regularization | Yes | Position-corrected 8-band Fourier transform (piezo\_E\_PosCor) | 46% lower class-wise standard deviation | 19% lower class-wise standard deviation |
| 3-layer perceptron with 25% dropout regularization | Yes | Non-corrected 8-band Fourier transform (piezo\_E) with positional input (X, Y, Z) | 46% lower class-wise standard deviation | 26% lower class-wise standard deviation |
| 3-layer perceptron with no regularization | Yes | Non-corrected custom 20-band Fourier transform with positional input (X, Y, Z) | 49% lower class-wise standard deviation | 45% lower class-wise standard deviation |
| 3-layer perceptron with no regularization | No (data set is much larger and has greater acoustic variation) | Full resolution custom Fourier transform (50,001 bands) with positional input (X, Y, Z) | Not applicable (network does not meaningfully function on this data set) | 40% lower class-wise standard deviation |

**Performance Analysis**

The performance of neural networks relative to AP and the network used in the PICO-60 paper was done using the class-wise standard deviation. This is a function defined as *C* below, where *N* and *A* are the sets of outputs of the binary discriminator in question (AP or a neural network), corresponding to the sets of neutrons and alpha particles respectively.

The first step is to calculate the standard deviation *S* of the union of the two sets. This gives an indication of the scale of the overall distribution. When *N* and *A* are divided by *S*, they are normalized so that the standard deviation of their union is equal to 1.

While the neural network’s outputs are bounded in the range of 0 to 1 with a sigmoid activation, AP has a significantly wider range. Normalization of the union prevents this from creating a bias where AP would produce a higher standard deviation with a similarly proportioned error.

The second step is to calculate the mean of the standard deviations of the normalized sets of neutrons and alpha particles individually. This is an indication of how tightly clustered or widely dispersed the discriminator’s predictions are. Very consistent predictions of *x* for neutrons and *y* for alphas, with minimal variance off of those specific values, will produce a low class-wise standard deviation.

This performance metric is used rather than a simple accuracy score because it takes in to account not only whether the discriminator is correct or incorrect, but how confident it is. The intent is for a decisive discriminator, which produces a wide separation between the two classes, to be preferred over one that produces a nebulous cloud of outputs with a seemingly arbitrary decision boundary.

**Input Data Formats**

Several formats for the input data to the neural network were tested, including some that were unsuccessful.

* Position-corrected 8-band Fourier transform: The audio frequency information present in the ROOT data file as piezo\_E\_PosCor. It is already corrected for the position of the bubble in the tank. This is used to calculate AP, and is used as input data for the neural network in the PICO-60 paper.
* Non-corrected 8-band Fourier transform: The same as above, except that it is not corrected for the position of the bubble. This was briefly used as the only input for a neural network, with some success.
* Non-corrected 8-band Fourier transform with positional input: The non-corrected Fourier transform combined with the position of the bubble in the vessel. (They are combined by simply concatenating them on the first layer.) The intent with this input format is that the neural network can learn to perform position corrections internally, rather than having it done as a preprocessing step. This avoids destruction of information and should allow for more flexibility, but may increase the complexity of the learning task.
* Non-corrected custom 20-band Fourier transform with positional input: The same as above, except that I reimplemented the Fourier transform process using NumPy and increased the resolution from 8 bands to 20.
* Full resolution custom Fourier transform with positional input: The banding process is removed from the custom Fourier transform. Instead, the 100,002 raw values are input directly into the neural network alongside the position of the bubble. There is no destruction of information; the first layer (which reduces the data down to 12 neurons) can act as a generalization of the banding process.
* Raw waveform: This is the most end-to-end input format possible. The samples of the unprocessed audio recording are input directly into a convolutional neural network. This drastically increases the complexity of the learning task, but should allow the network the flexibility to perform an approximate generalization of a Fourier transform as it is relevant to discrimination between neutrons and alphas. Some early tests of this technique indicated that it easily overfits on noise, but is much more difficult to train a network on for meaningful accuracy, compared to the Fourier transform formats.
* Raw waveform with positional input: Later on, positional input was added to the format above. Rather than being simply concatenated (which would not work with a convolutional neural network), the audio waveform is processed by several convolutional layers and the positional data is concatenated as an additional input near the end, before the first dense layer. When this was tested, it made very little difference; it seemed the network was not able to make the connection between the position and the raw waveform data nearly as well as it could with the Fourier transform.
* Image data: Rather than audio, the images collected during bubble formation are used as inputs for this technique. They are used as inputs for a 2-dimensional convolutional neural network. This was tested only briefly, and produced a very high degree of overfitting and poor accuracy.

**Network Architectures**

Many different network configurations have been tested, but they can be divided up into a few major categories.

* Shallow perceptron: This was the type of network used for all Fourier transform analysis. It consists of a small number of dense layers (usually just 2 or 3) with small numbers of neurons. Learning based on the Fourier transform data is not an extremely complex task, so it does not require large architectures. Dropout and L2 regularization are used to alleviate overfitting.
* Moderately deep fully convolutional network: This consists of ~5 one-dimensional convolutional layers followed by a single dense layer. The lack of stacked dense layers helps to minimize the number of parameters, and preserves spatial information until the very end.
* Very deep fully convolutional network: This basic architecture is inspired by the M34 model, which was designed specifically for processing raw waveforms. Depending on the specific configuration, it can have ~20–40 convolutional layers. Either a global average pooling layer or a single dense layer is used at the very end; a dense layer was found to provide much greater flexibility.
* Image processing convolutional network: This consists of ~4 two-dimensional convolutional layers and ~3 dense layers with dropout regularization. It is a fairly standard convolutional model intended for processing small image windows containing a bubble.