NAAES Final Report

Eugenio Beaufrand - Eric Hsieh

August 2020

1 Problem Statement

The problem at hand is waveform identification using machine learning. The data provided for this problem is labelled waveforms of 7 features containing a varying number of timesteps. The approaches that will be used to attempt to identify waveforms will be in the domain of deep neural networks, and the goal is to find a combination of model architecture and hyperparameters that achieves best results. Some constraints that must be considered are inference time once a model has been trained, and that a trained model's architecture must be compatible for use with the Intel Neural Compute Stick 2.

The real-world application for this project is to identify incoming waveforms for military usage. Intercepting various transmissions with a machine learning algorithm to classify them as friendly or enemy signals will allow quicker responses. By utilizing a machine learning algorithm to classify waveforms, there is no need to maintain a large and potentially incomplete database of various signals. By being able to classify which kind of emitters a waveform comes from, decision-making speed can be improved for what kind of jamming techniques to use in order to avoid detection.

2 Existing Solutions

1. Original Projection (Aces_train.py):

   The initial existing algorithm for this project is contained in the file aces_train.py, and is a Deep Neural Network (DNN) that classifies waveforms into given categories. The input data is Pulse Descriptor Words (PDW) that detailed different aspects of a given waveform, such as pulse width, amplitude, frequency, bandwidth, and time of arrival. These become the main features for the machine learning algorithm. The goal is to classify all the input waveforms into 14 different emitters, grouped into 4 classes: A, B, C, and D. Each class's emitters all produce similar outputs, and the classification can be measured by two metrics: the emitter accuracy and the mode accuracy.

   In addition, the algorithm also seeks to preserve as much data as possible. Therefore, it uses projection before averaging its features; this allows the initial data to still be kept before averaging. This is important as averaging a continuous function, such as a signal, can remove domain-sensitive information.

2. Low Cost Projection Algorithm (Aces_lcpa_train.py):

   A modification of the original aces_train.py to classify using a Low-Cost Projection Algorithm (LCPA)

   The LCPA approach aims to lower cost, where cost is how long it takes to validate dwells and make inferences on waveforms, which also results in faster training time. The difference in the algorithm is that it takes the waveform data and projects it in a different manner. The output of this projection is a vector of dimension 102 which is fed into a network.

3 New Solutions, Improvements and Contributions

1. Bug fixes to low cost projection algorithms projection function (lcpa):

   Early on in the course of the project, it was noticed that the projection outputs of waveforms were too similar, and that the projection wasn't differentiating waveforms as was intended. Upon noticing this the code was updated to fix the projection and the results improved significantly, with dataset robust_scenario1 seeing an increase in accuracy from 50% to 80%.

2. Vectorization of low cost projection algorithm logic (lcpa):

   During this project, an issue was the long time it took to perform validation when classifying waveform dwells using the lcpa method, taking 13 ms/dwell. By rewriting the code in the projection step using numpy vectorized expressions, an improvement of 80% time reduction was obtained, and now validation takes 3 ms/dwell.

3. Hyperparameters and layers tuning:

   A large portion of this project was spent exploring the many different potential permutations of hyperparameters to increase model accuracy while managing the size of the model and the time it takes to train and perform validation. The hyperparameters considered were number of hidden layers, number of units per hidden layer, dropout per layer, the addition of extra features, and where applicable number of convolutional, pooling, and recurrent layers.

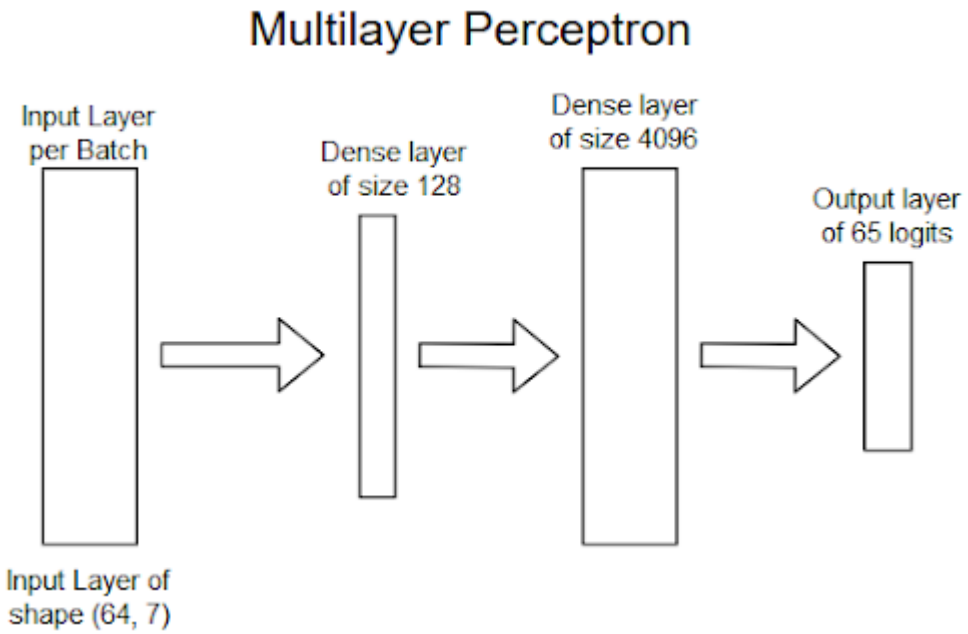
4. Keras implementation:

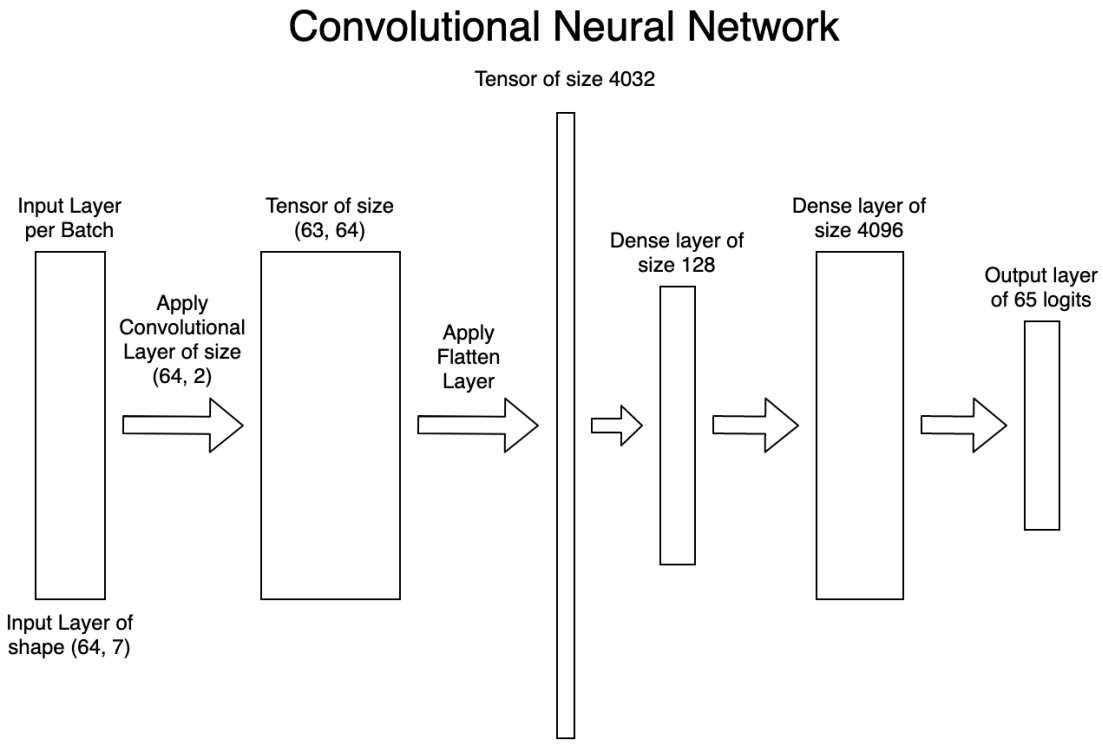
   Throughout the course of the project, it was agreed upon that creating a Keras version of the aces classifier would benefit the speed at which the team could create new models and test them. As a result, a keras aces classifier was created, as well as supplementary scripts that facilitate taking the saved model output of the classifier to intermediate representation and performing inference of the Intel Neural Compute Stick 2.

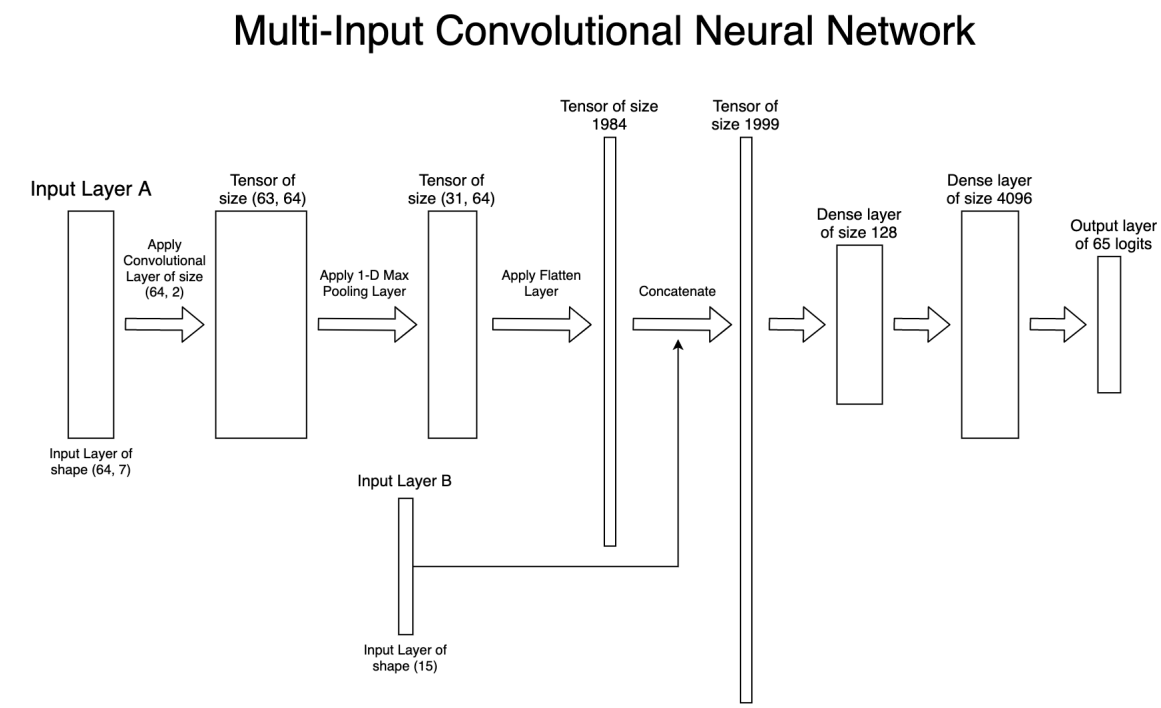
5. Architectures:

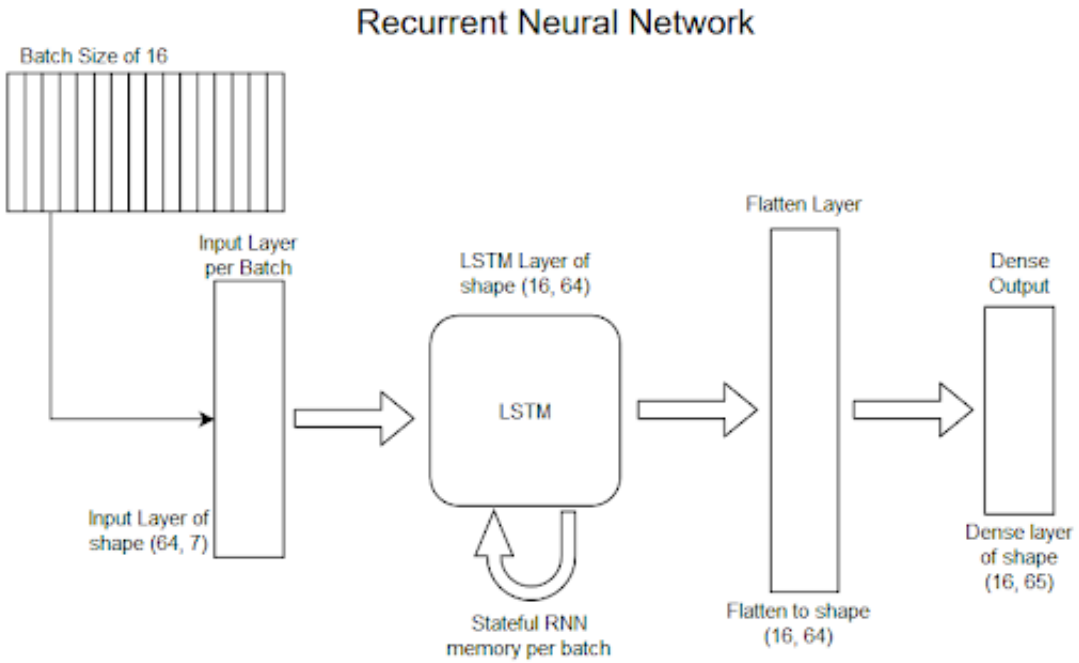
   In addition to the standard Multi-Layer Perceptron model that was provided when starting the project, Convolutional Neural Network Models, Recurrent Neural Networks Models, and Multiple Input Neural Network Models were experimented with.

4 Diagrams of Architectures Used









5 Model Research Methodology

1. Hidden layer sizes

   Searching for combinations of different hyperparameters can improve the accuracy of the the initial algorithm (aces_train.py). The algorithm used two hidden layers of size 128 initially. Varying these layer sizes would often increase accuracy at the cost of speed. Searches on all combinations of two-layered MLP hidden layer sizes in multiples of 128 resulted in an optimal size of 128 for hidden layer 1 and 4096 for hidden layer 2.

2. Number of Hidden Layers

   Varying the number of hidden layers was also experimented with. Both the original projection method and the low cost projection algorithm started out with two hidden layers. Additional hidden layers were attempted, but this resulted in a decrease in accuracy and an increase in time taken compared to a network of just two hidden layers.

3. Learning Rate

   Adjusting the learning rate of the network changes the rate at which the network learns. By increasing it, the network will make larger changes to the network weights when learning, but it may not converge to an optimal value. Decreasing it may cause it to take more time to learn a given model, which can impact the time taken to converge. When adjusting the parameters of the low cost projection algorithm to dataset robust_scenario3, breaking the 40% accuracy mark was met with difficulty by just tuning hidden layers, so decreasing the learning rate was attempted. However, adjusting the learning rate did not have a sizable impact on the accuracy.

4. Dropout

   By varying dropout rate, a random ratio of neuron outputs are ignored. By introducing RNG into this network, potential improvements may be found in the internal representation being learned and may help with overfitting. However, increasing the dropout weight (by decreasing the ratio of neurons kept) did not seem to improve the accuracy for this implementation.

5. Number of Convolutional Layers

   Whether or not to use convolutional layers and how many according to its impact on accuracy and trade-offs with time to train and test. In this project One dimensional convolutions are being used. These types of convolutions are also known as temporal convolutions which are commonly used in instances where data is in the form of having time-steps. This is why this idea was proposed as having potential. This layer type creates a filter of a desired size and passes it along the features in the time-steps to create a feature map. This feature map can often be more telling than simply dense layers. All of the models in this project with convolutional layers also make use of Max Pooling, which provides the network with spatial variance, helps prevent overfitting, and decreases model size.

6. Number of Recurrent Layers

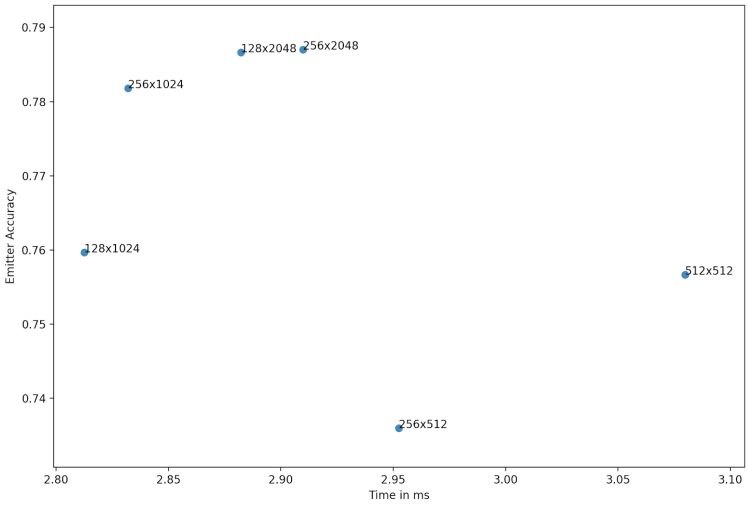
   Only one recurrent layer per implementation was used. Recurrent layers; especially LSTM, can be very time-intensive. Therefore, for the purpose of maintaining a good speed to accuracy ratio, it was attempted to optimize the use of a single layer.

7. Extra Features

   It was proposed throughout the project that adding the mean and standard deviations of each feature throughout the waveform time-steps for each waveform might be helpful in the algorithm to learn the data better. This was implemented in some of the convolutional architectures and was helpful in achieving slightly higher accuracy.

6 Results

1. Accuracy vs Time Trade-off Plot for LCPA:



2. Tables of architecture top performance by data set:

Table 1: Robust scenario 1

	original projection	lcpa	cnn*	rnn
emitter accuracy	58.6%	79.8%	64.5%	43.6%
type accuracy	100.0%	99.2%	99.9%	99.9%

Table 2: Robust scenario 2

	original projection	lcpa	cnn*	rnn
emitter accuracy	51.3%	33.2%	60.1%	35.2%
type accuracy	100.0%	99.7%	100.0%	99.9%

Table 3: Robust scenario 3

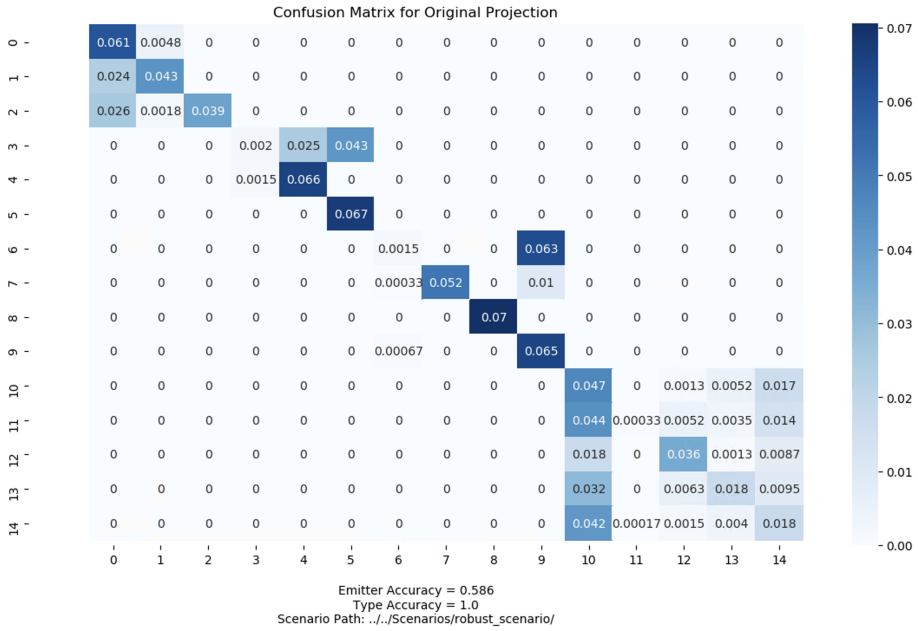
	original projection	lcpa	cnn*	rnn
emitter accuracy	53.3%	41.5%	50.3%	40.4%
type accuracy	99.9%	99.5%	99.9%	97.6%

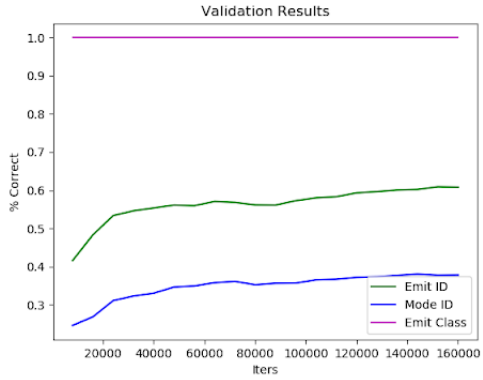
3. Best Performing Models

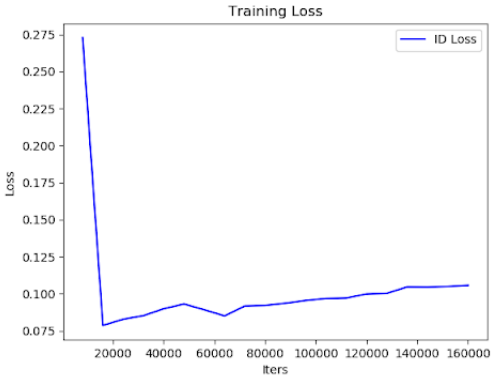
   (a) Best model with original projection:

   Architecture: MLP, 2 Dense Layers (128,4096) -> Dense Output Layer (65 logits)

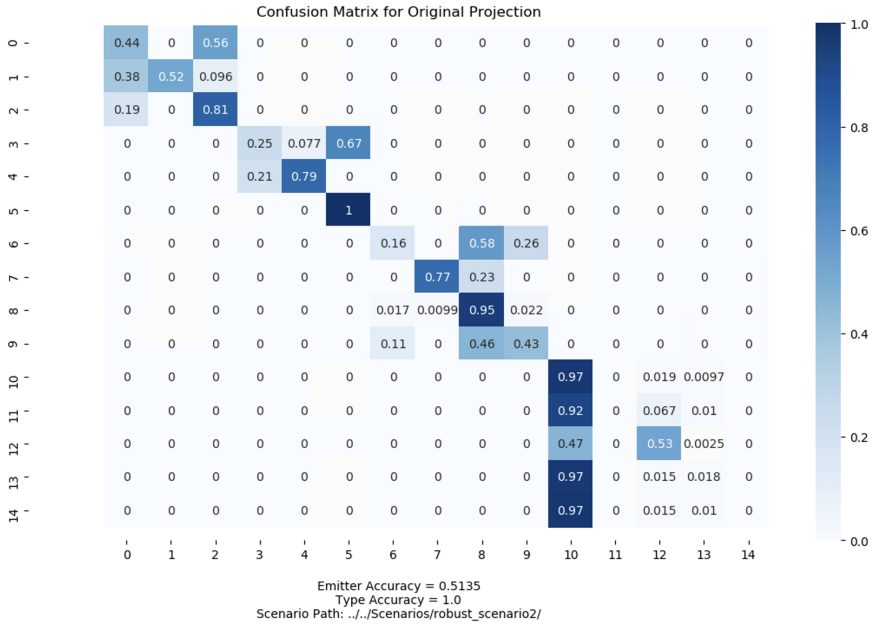
Robust Scenario 1 Results:

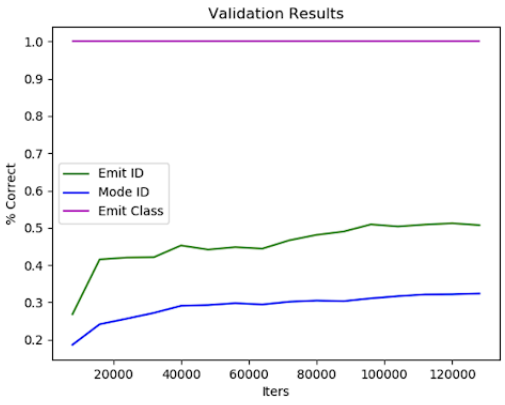




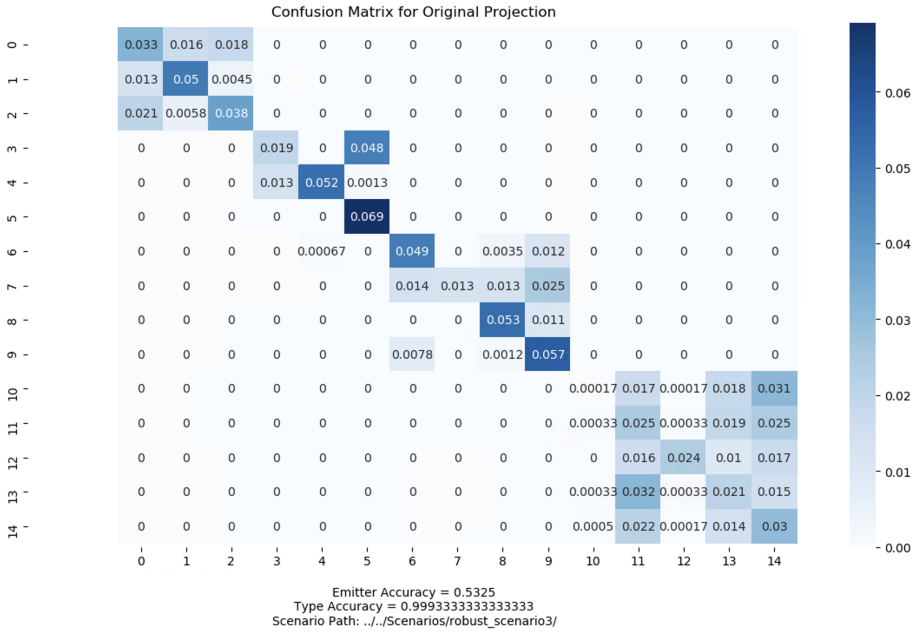


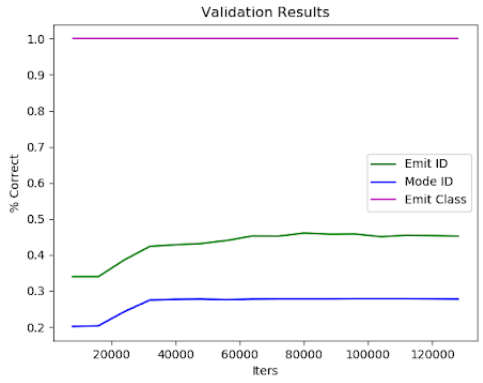
Robust Scenario 2 Results:





Robust Scenario 3 Results:

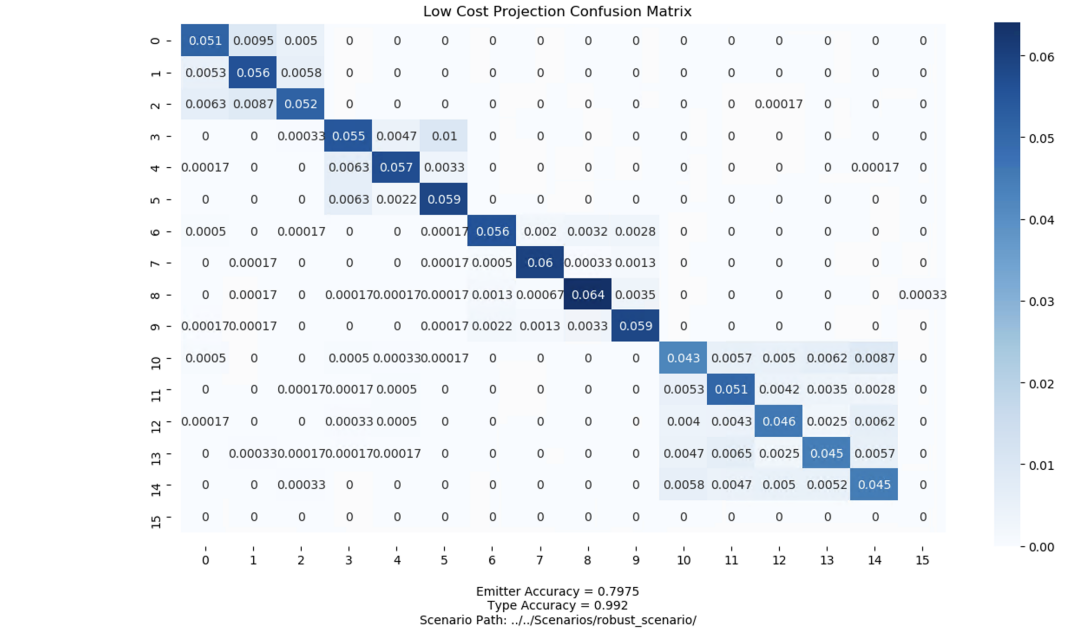


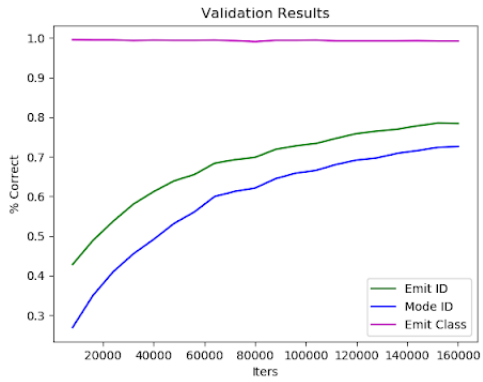


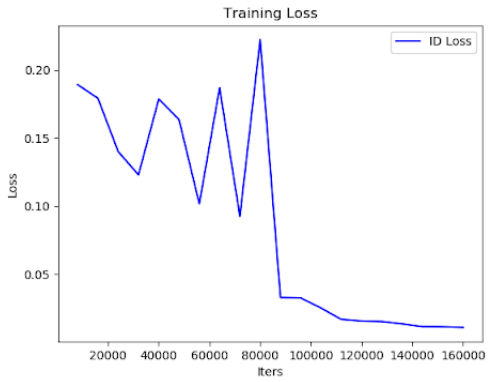
(b) Best model with low cost projection algorithm:

Architecture: MLP, 2 Dense Layers (128,4096) -> Dense Output Layer (65 logits)

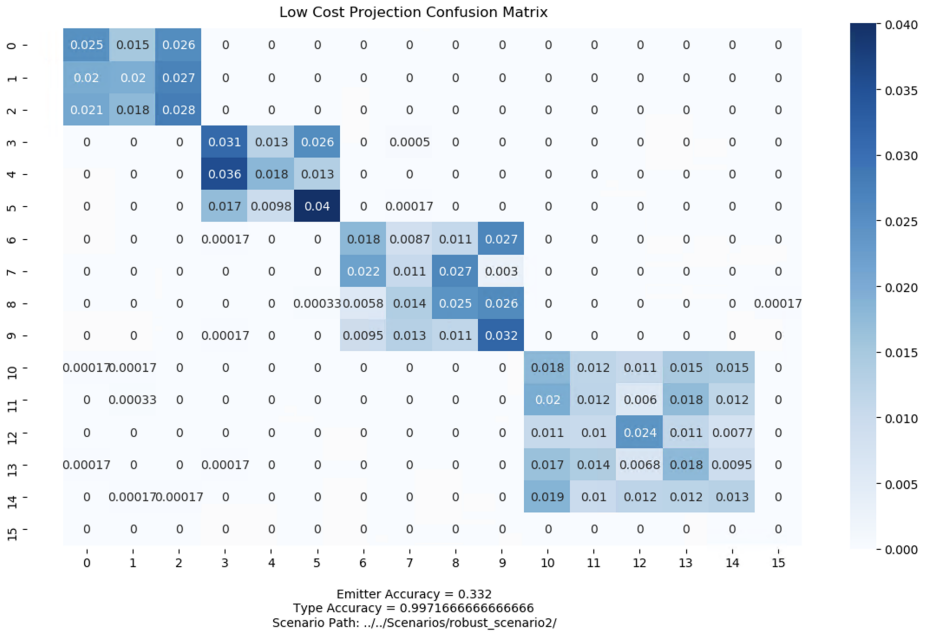
Robust Scenario 1 Results:

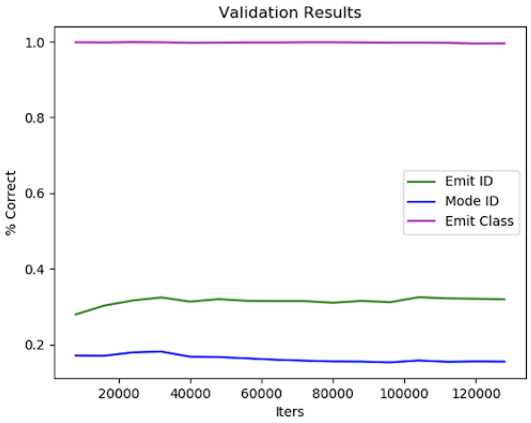




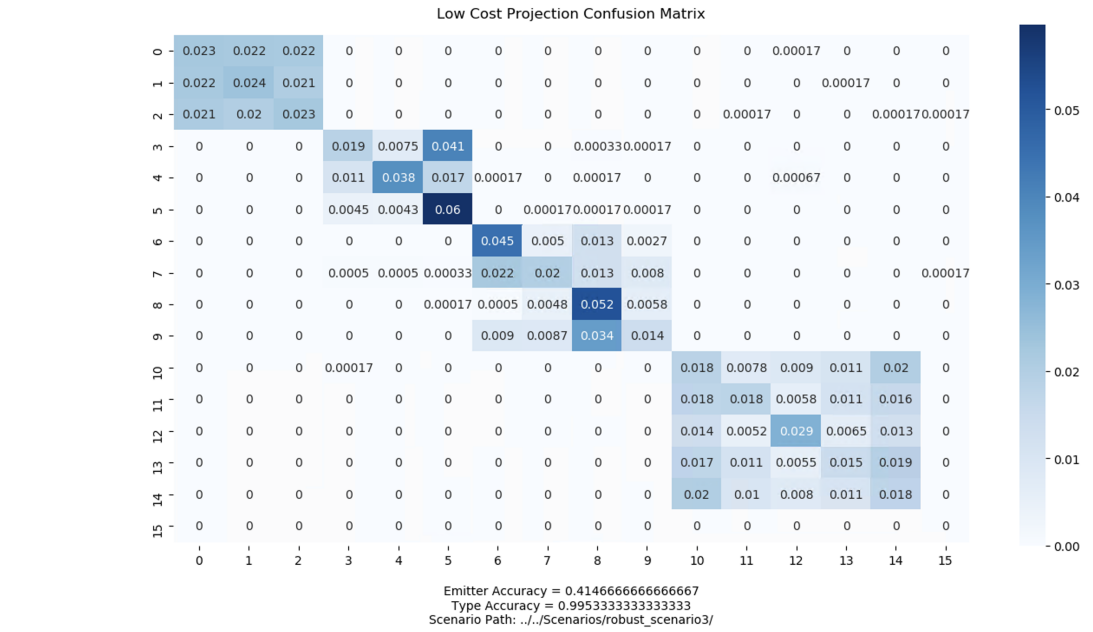


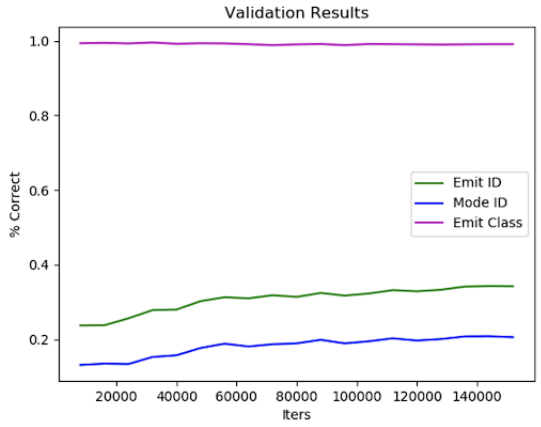
Robust Scenario 2 Results:





Robust Scenario 3 Results:

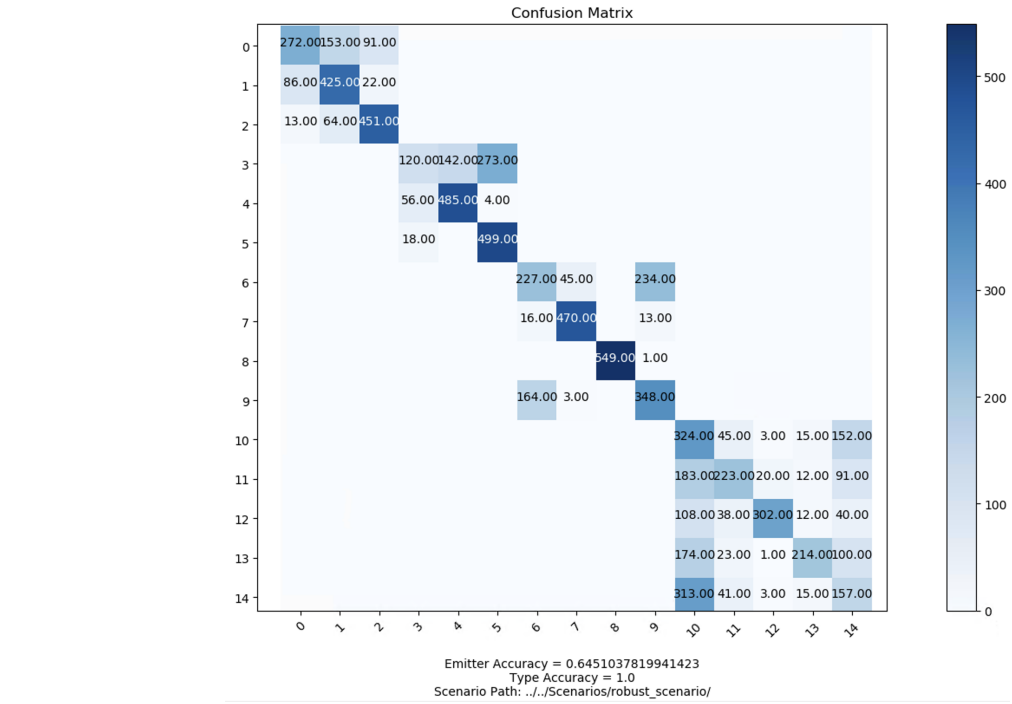


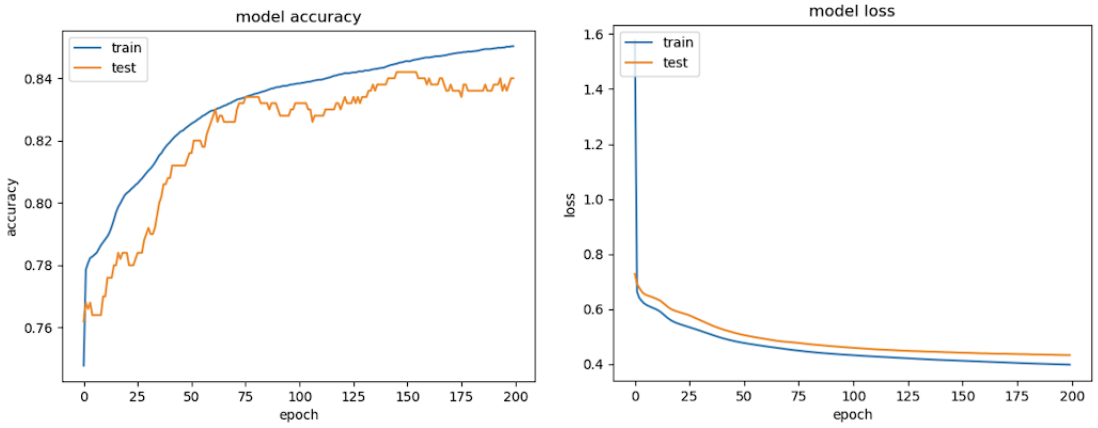


(c) Best model with convolutional architecture (using extra features)

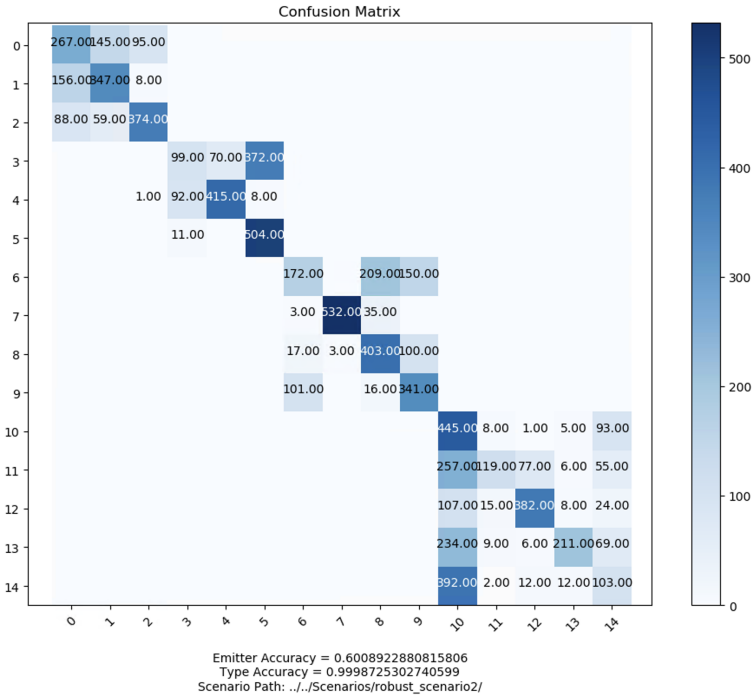
Architecture: Convolutional Neural Network, One Convolutional Layer (Input size: 64, kernel size 2) -> Max Pooling Layer -> Concatenation of Max Pooling Output and Extra Features -> 2 Dense Layers (128,4096) -> Dense Output Layer (65 logits)

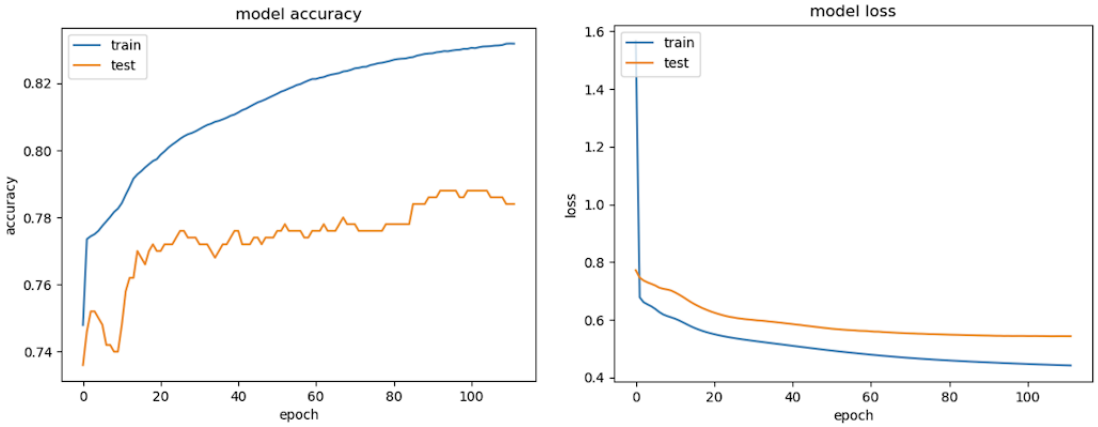
Robust Scenario 1 Results:



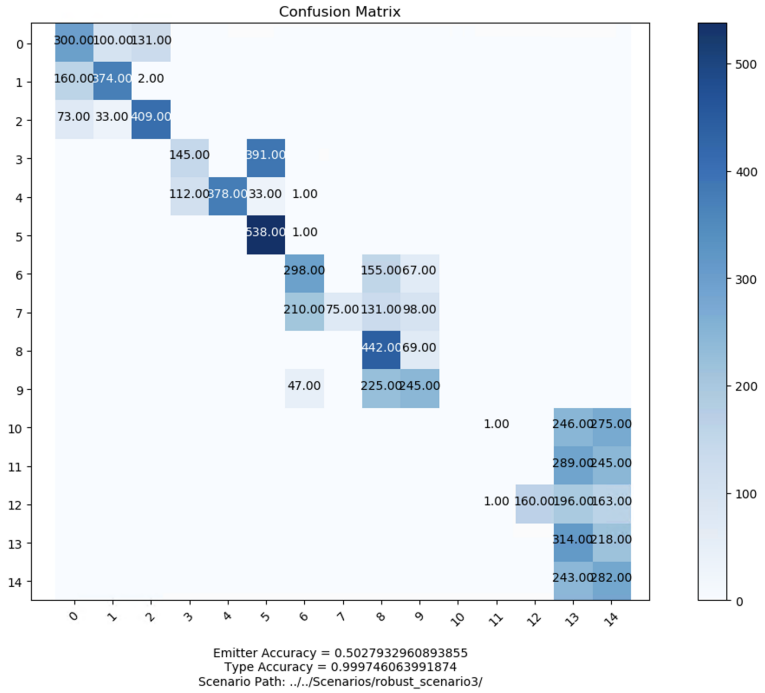


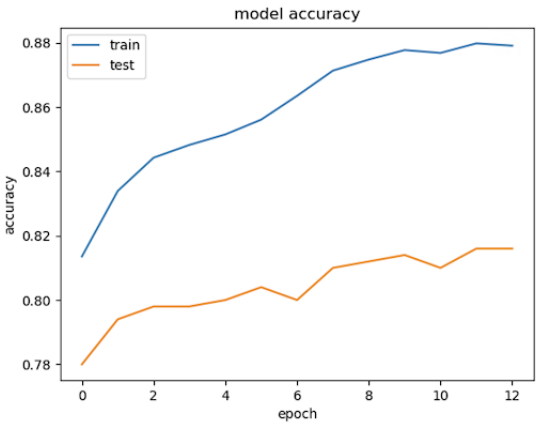
Robust Scenario 2 Results:

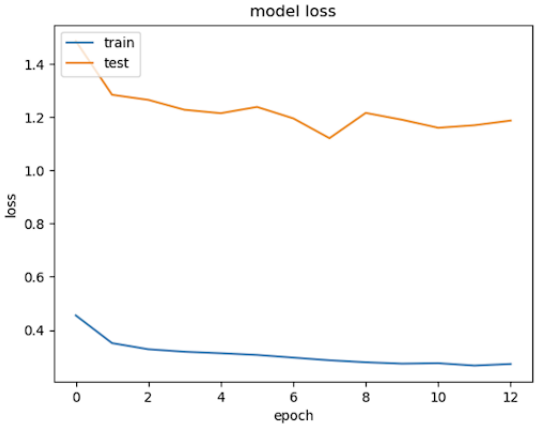




Robust Scenario 3 Results:



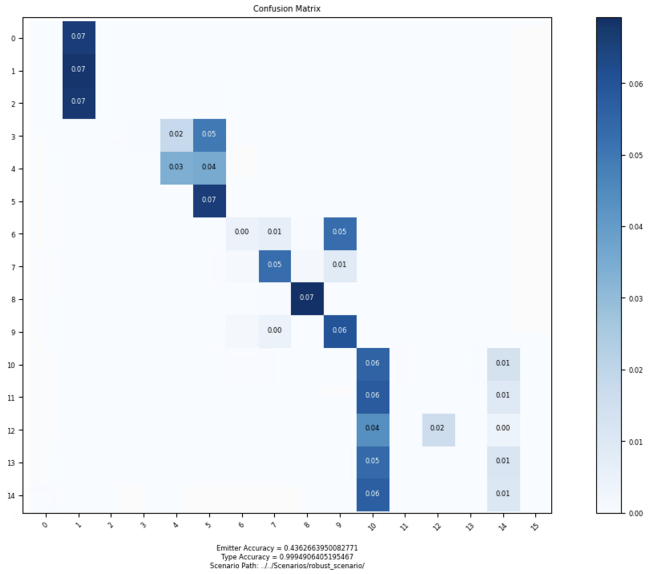




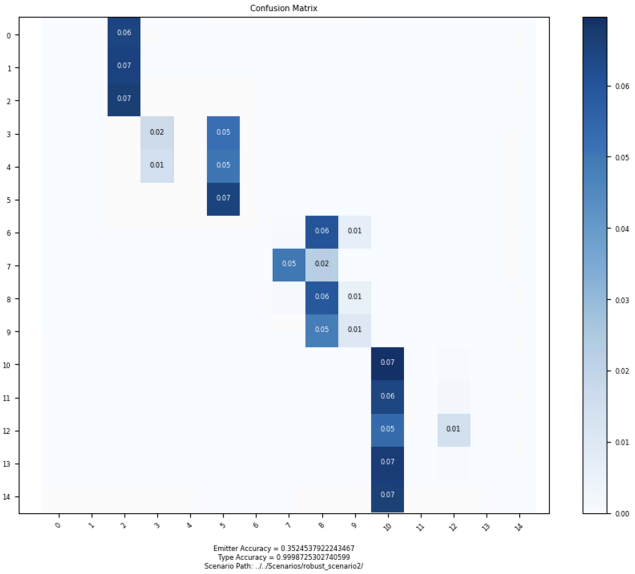
Best model with Recurrent Architecture:

Architecture: LSTM Layer -> Dense Output Layer (65 logits)

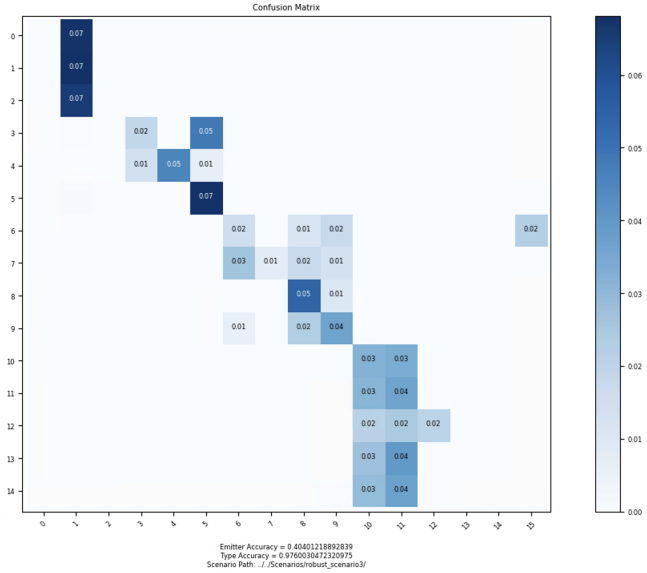
Robust Scenario 1 Results:



Robust Scenario 2 Results:



Robust Scenario 3 Results:



7 Conclusions

From the models tested, the Convolutional Neural Network with the added features architecture was most consistently the top performing network for the three scenarios or very close in accuracy to the top performing network. The Convolutional Neural Network with added features architecture attained an average 58.3% accuracy across the three robust scenarios.

The next best performing architecture was the original projection, yielding reasonably good accuracy on all three scenarios, however typically accuracy below the Convolutional Neural Network with added features architecture. The original projection attained an average 54.4% accuracy across the three robust scenarios.

With regards to the Recurrent architecture and the low cost projection algorithm network, they both typically performed much less favorably than the other two. The low cost projection algorithm attained an average accuracy across the three robust scenarios of 51.5%, although it must be noted that for robust scenario one this algorithm performed extraordinarily well (80% accuracy), while for the other two scenarios it performed relatively poorly (below 42% accuracy). The recurrent architecture attained an average accuracy across the three robust scenarios of 39.7%.

The recurrent network has not been thoroughly tested and optimized yet. There may be potential to outperform the other network architectures because its stateful capabilities should theoretically synergize well with the continuous-time data. However, it has not been performing as such due to many factors, such as training time, hyperparameters, dataset, and overall architecture. Therefore, the results with the recurrent neural network may not reflect the potential of the recurrent network as a whole.

For the data given, it is possible that the way the scenarios generated at hand don't lend themselves to learning the data best. The possibility of generating longer waveforms has been discussed, and that this might have a favorable effect upon classification accuracy. A good course of action will be to investigate if these discussions have any merit by testing a dataset with longer waveforms. Another good question to answer will be if generated data's classification accuracy corresponds to real data classification accuracy.

Compared to the initial neural network architecture, the new architectures have improved the overall accuracy. The convolutional neural network with added features architectures performed close to, if not better than, the initial multi-layer perceptron model with original projection on all scenarios. Additionally, with a bit more tuning and longer training, the recurrent network may very well see significant increases in accuracy.

8 Next Steps

In general, an important step will be to verify that model results being seen indicate how well a model can learn real waveform data. The artificially generated data may not necessarily tell how well the neural network can learn real waveform data. As a result, the current understanding of how the architectures perform with respect to one another may not correspond to their true differences in performance when learning real data.

Another area of interest is it to create a dataset with longer waveforms. It was proposed that longer waveforms might give a better ability to learn. If this turns out to be true, since this is a setting that can be tuned when learning real data, this would be valuable for increasing accuracy.

8.1 Recurrent Neural Network

For the recurrent neural network, the optimal next step would be implementing Backpropagation through time (BPTT), a gradient-based training approach, as opposed to the current training method, which is just iteration. At the moment, the current training method in the recurrent architecture implementation will keep increasing its weights as long as the next value is greater or equal to the previous value. This method is also extremely time-intensive. An advantage of BPTT is that it boasts significantly faster training speeds than most other optimization techniques, and will also be able to benefit from working with a stateful LSTM RNN, since they both utilize memory from previous outputs.

In addition, the RNN still needs loss and validation plots to be implemented to match the data from the other networks. Furthermore, the validation curves don't necessarily match up with the numbers as it was not done through Keras model.fit function, so this needs to be re-examined. Finally, the LSTM RNN network's accuracy has been increasing as epochs increase as well; if the RNN could be run faster, such as by running it through the GPU, then potentially it would be worth it to run the model for many additional epochs to see if accuracy continues to increase.

Another item that could be improved upon is that the training loss and validation plots, which currently display emitter accuracy and training loss not ignoring singleton arrays. When plots that do ignore singleton arrays are created, it may be easier to discern improvements in model function.