# Design and Analysis of a Convolutional Neural Network for Optimizing Particle Detectors at Jefferson Lab

22 April 2022

**Grace Johns and Dr. Edward Brash**

Senior Capstone Final Report

# 1 Abstract

Machine learning has become a popular tool for data analysis in recent years in almost all fields of research. This is especially true for the field of physics, as machine learning has the potential to lead to faster and more accurate results than traditional mathematical algorithms. Convolutional Neural Networks (CNNs) are a type of machine learning algorithm most commonly used for sorting and categorizing images, but also, due to their construction, are very adaptable and flexible for most types of input data. This made it an excellent choice for the focal plane polarimetry experiment at Jefferson Lab, an experiment where the angles of protons as they went through a given material and scattered were measured and used to calculate the polarizations and momenta of the protons. A CNN, given the initial measurements that represent the incoming track of the protons, can be trained to calculate the outgoing scattering angles the protons will have when they exit the material. This experiment strived to create a functional CNN for the Jefferson Lab polarimetry experiment and find the best parameters to maximize its accuracy.

# 2 Introduction

With its abilities to handle large amounts of data and learn which parameters are best to use for classifications and predictions, machine learning has become a popular tool for scientists in many different fields. The purpose of this project was to see how well a convolutional neural network (a type of machine learning program) would work for an experiment conducted a few years ago at Jefferson Lab. If it resulted in a higher accuracy than the mathematical algorithm used for the experiment, it would potentially be able to be adopted as the main data analysis method for experiments similar to this one in the future.

A convolutional neural network (CNN) was chosen instead of other kinds of machine learning programs, because it is primarily used for classifying images and other kinds of data where the positions of the data points relative to each other matter. The data from the Jefferson Lab experiment has this feature, which makes a CNN an excellent choice. The CNN also has a relatively simple design, making it a manageable program to learn how to code within the project time limit, even with no prior experience of any kind with machine learning. Convolutional neural networks are also very flexible. Since the initial weights and biases do not need to be known, as long as there is a sufficient amount of training and testing data and it is formatted clearly, a CNN will be able to classify the data. During the training stage, the CNN will calculate the most important characteristics of the data through multiple trials and adjustments. This means a it has the potential to be more efficient and more accurate than calculating by hand or with a traditional minimization algorithm.

A simple CNN example for classifying hand written digits was used to learn how to code CNNs, and also to gain expectations of how the accuracy and loss change as the CNN trains. The example CNN had overall accuracy of about ninety-eight percent, and it was expected that the CNNs for this experiment would not reach that high of an accuracy, as they would not be able to be fine-tuned as much as the example was in the time provided for the project.

# 3 Theory

## 3.1 Overview of Focal Plane Polarimetry Experiments at Jefferson Lab

At Jefferson Lab (JLab), there have been a series of experiments where high energy electrons impinge on a stationary target at such a velocity that protons break free from the nucleus of the target atoms [P$^+$18, P$^+$12, P$^+$17]. The polarization (*i.e.* the degree of spin alignment) of the outgoing protons was then measured. Based on the polarization of the incoming electrons, the electric and magnetic form factors of the proton were measured. The angles of the outgoing protons, following a second scattering, are essential for measuring the polarization components of the scattered proton, and it is necessary to have the uncertainties in these angles to be as small as possible. One of the sources of uncertainty in the angle measurements is related to the physical alignment of the detectors.

To align the detectors, scattered protons were sent through the detectors freely with no second scattering. Ideally, the measured polarization components of these protons should then be zero, as they are colliding with nothing, but warping and tilting of the detectors can cause misalignment, resulting

in a perceived non-zero polarization. The problem then becomes finding the initial misalignments of the detectors so that they can be taken into account when making systematic corrections.

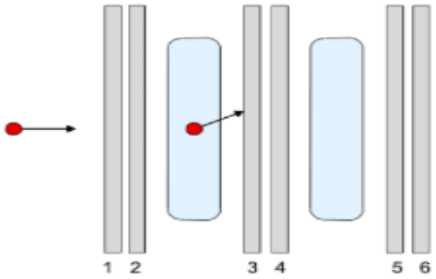


Figure 1: This figure shows the orientation of the six detectors and the scattering material as a proton (red) travels through it.

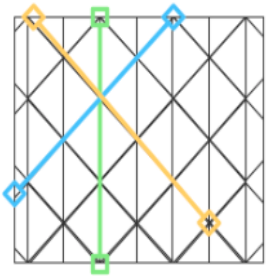

Figure 2: This figure shows the orientation of the wires within a detector chamber. Green represents the x wires; blue represents the u wires; yellow represents the v wires.

In the initial analysis, a minimization function dependent on the physical misalignments was created, and an algorithm was developed to determine these misalignments. This technique was limited in its ability to accurately calculate the misalignments due to the fact that only a few variables were taken into account. A deep learning program, however, has the capability to handle thousands of inputs easily, and can hopefully better determine the angles of second scattering more accurately.

The experimenters also utilized a simulation of the experiment to calculate expected values. This simulation, like the experiment itself, consisted of a scattered proton traveling through two blocks of material and six detectors. As shown in Figure 1, the detector chambers are grouped in pairs with a single block of material between each pair. The inside of each chamber has a mesh of wires, with a third of them oriented vertically (x), a third oriented at forty-five degrees relative to the vertical orientation (u), and the last third oriented at a negative forty-five degrees relative to the vertical wires (v). The wires are numbered, and as a particle passes through a chamber, it drifts towards the wires nearest to it. The wire number for each orientation and for each chamber is recorded as well as the time it takes the proton to drift to each wire (dx, du, dv).

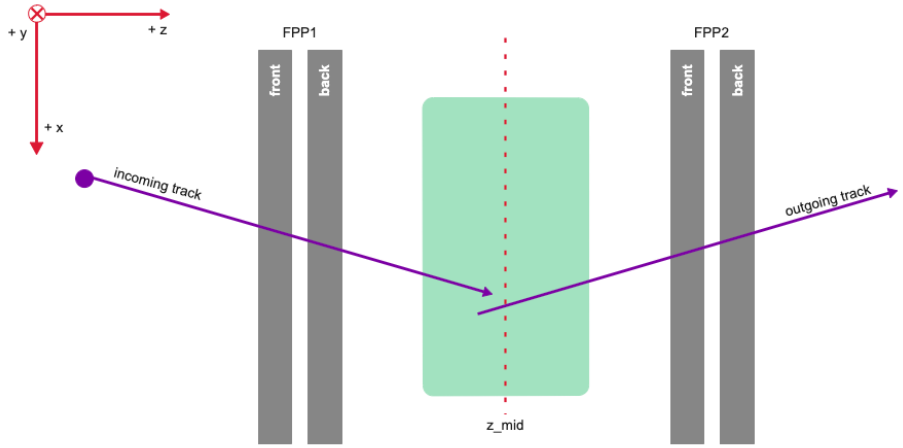


Figure 3: This figure shows the orientation of the coordinate system, as well as an example of a golden track for a proton (purple) travelling through it.

The coordinate system for this experiment is oriented such that the positive x-axis is pointed vertically towards the ground, the positive y-axis is pointed into the page so that it is parallel to the detector chambers, and the positive z-axis is pointed to the right so that it is perpendicular to the detector chambers, as shown in Figure 3. The projection of a particle's track onto the x and y axis

can be calculated using the wire numbers and drift values of the particle as it passes through a pair of detector chambers, as well as the angles of the track respective to the x-z and y-z plane.

For this project, the only tracks that will be considered are ones that are caused by a single proton travelling through only the first four detectors and scattering once in the material, in such a way that its scattering angle is small enough for the particle to travel through the all four detectors. These tracks are called "golden tracks," and an example track is shown in Figure 3.

## 3.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of machine learning program that, given an input image, assigns importance to various characteristics of the image, and differentiates one image from another [Sah18]. It is comprised of convolutional layers, pooling layers, and a fully connected neural network. The convolutional layers are matrices the same depth of the input image, with arbitrary weights assigned to each element, that move across the input image with a given stride length starting in the top left corner of the input image and moving across and down until it has covered the entire image. It creates a new matrix of data that is either the same size or smaller than the initial input. For example, as shown in Figure 4, the convolutional layer could be a three by three matrix where all the diagonal elements are assigned a weight value of one, and the rest are zeroes. When it moves across a five by five input image, it multiplies its weights by the corresponding pixel values, adds the results together, and creates a new convolved feature that is the same depth as the original image, but now only three by three pixels. If desired, padding can be added to the input image so that the convolved image is the same size as the input image, but this is only necessary if there are important characteristics on the edges of the input image. The weights of the convolutional layers are assigned randomly, and, during the training stage, the CNN tests to see which weights work best for the classification using a minimization function. Any number of convolutional layers can be added to the CNN, though typically more layers will lead to a higher run time. The size of the input data gives a general indication of how many convolutional layers may be needed; larger sized input data usually requires more layers, while smaller input data usually requires less.

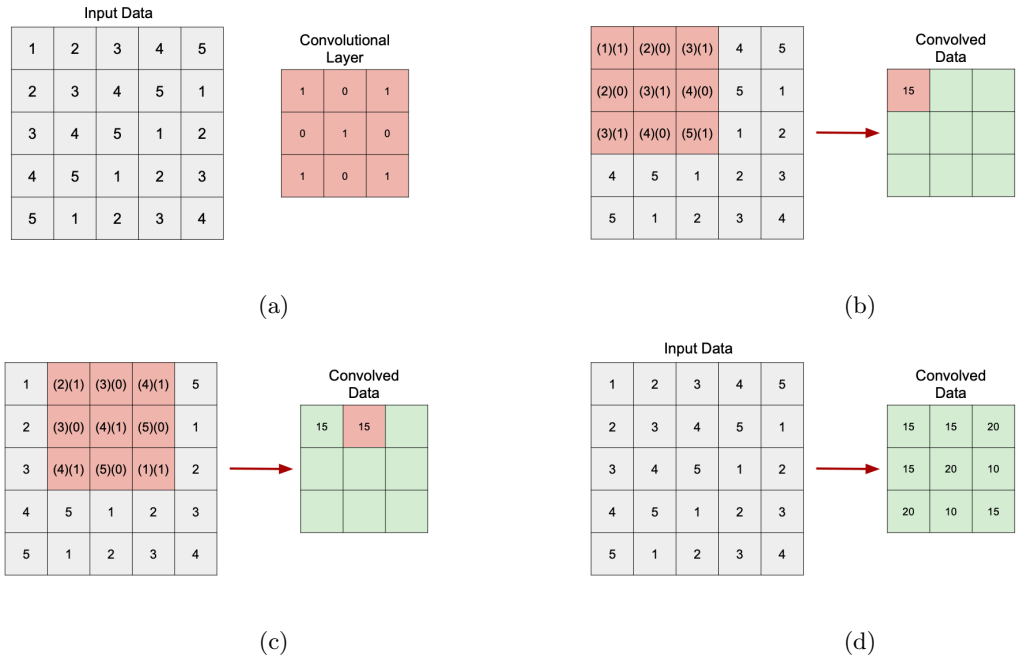


Figure 4: Example of a convolutional layer (red) moving across an input (gray) and producing convolved data (green).

The pooling layers are also matrices like the convolutional layers that move across the given image. But instead of being composed of weights and biases, the layers "pool" the data together, making the

convolved images smaller in one of two ways: It either returns the maximum value of the portion of the image it covers, or returns the average value of the pixels it covers. There can be multiple pooling layers or just one, but after the input image goes through all the layers, it is flattened into a column vector and connected to the neurons in the first layer of the neural network. Different numbers of neurons in the first layer should be tested to see what works best for the CNN, as well as how many layers of neurons will be in the neural network. Lastly, these neuron layers connect to the output nodes, the number of which depends on the number of categories the image could be sorted into, and the CNN classifies the image.

The CNN is first trained on a set of data that has the the input data as well as the corresponding correct classification for each data point. The input data goes through the CNN and its prediction is compared to the actual classification. If the prediction is incorrect, then the CNN adjusts its weights and biases to better match the correct answer and runs again with the corrections. The number of times the CNN goes over all the data (*i.e.* the number of epochs), as well as the number of times the CNN is allowed to correct itself (*i.e.* the batch size), are both parameters that can be adjusted during the creation of the CNN. A second of data, completely separate from the training data set, is used to test the CNN's accuarcy.

# 4 Methods

Python was the only coding language used for this experiment. The NumPy, Matplotlib, pandas, scikit-learn, and Keras python libraries were used to organize the data, create and run the CNN, and analyze the results. Keras was used the most, as it has the primary functions needed to create and run machine learning programs. For computing power, the actual neural network training and testing was run on a JuputerLab server housed in Luter 345. This server comprises a quad-core (40 thread) CPU with more than a terabyte of available disk space, which was more than enough for this project.

From the simulation of the polarimetry experiment, two CSV files were provided with a total of 80,000 data points of golden tracks. In the first file, each data point had the x and y projections for the incoming track (x0f1, y0f1) and the outgoing track (x0r1, y0r1), the angle projections onto the x-z and y-z planes for the incoming (tthetaf1, tphif1) and outgoing tracks (tthetar1, tphir1), and the final scattering angles of the outgoing track relative to the incoming track (theta, phi). The midpoint of the scattering material (zmid1), the position where the particle scatters in the material (zclosef), and the closest distance between the incoming and outgoing tracks (sclosef) were also included in the first data set. In the second file, each data point had the wire numbers (n1ua, n1xa, n1va, etc.) and drift distances (d1ue, d1xe, d1ve, etc.) for each of the four detectors, the final scattering angles (theta, phi), the x and y projections for the incoming track (x0f1, y0f1), the position of scattering (zclosef), and the closest distance between the incoming and outgoing tracks (sclosef).

```
df1.columns =['x0f1', 'y0f1', 'tthetaf1', 'tphif1','x0r1', 'y0r1', 'tthetar1', 'tphir1',
              'zmid1', 'theta', 'phi', 'zclosef', 'sclosef']

df2.columns =['x0f1','y0f1','n1ua','n1xa','n1va','n2ua','n2xa','n2va','n3ua','n3xa','n3va',
              'n4ua','n4xa','n4va','d1ue','d1xe','d1ve','d2ue','d2xe','d2ve','d3ue','d3xe','d3ve',
              'd4ue','d4xe','d4ve','theta','phi','zclosef','sclosef']
```

Figure 5: This figure shows the labels of the data points in the two CSV files, df1 and df2, containing the simulation data.

Two CNNs were constructed using this data. Both were designed to predict only the scattering angle theta, as it is much easier to create separate CNNs for the two scattering angles rather than trying to manipulating it into predicting both at once. The theta angles data was reformatted into bins, so instead of each point being an angle value, it was instead a single truth value (*i.e.* one) in a 58 element row matrix of zeroes. This would allow the CNN to use a classification prediction function, rather a regression prediction function. A stochastic gradient descent algorithm with a learning rate of 0.01 and momentum of 0.9 was used to optimize the parameters, and a categorical crossentropy function was used to compute the loss of the CNN.

For the first CNN, CNN 01, the input data consisted of the x and y values and the two angle

projections for the incoming and outgoing tracks arranged in a two-by-four matrix. The input data was padded with zeroes before being processed by the CNN. There was one two-by-two convolutional layer, which condensed the four-by-six padded input data into a three-by-five matrix. A two-by-two max pooling layer was then added, which further condensed the data into a two-by-four matrix. The data was flattened into a eight element row vector, and fed into the first layer of the dense neural network which consisted of 4 neurons. The second, and last layer, of the neural network consisted of 58 neurons used to classify the input data into one of the 58 output bins for the theta angle. The convolutional layer and the first neural network layer both used a Rectified Linear Unit (ReLU) activation function, and an He uniform kernel initializer. The last neural network layer used a softmax activation function to classify the data.

For the second CNN, CNN 02, the input data consisted of the wire numbers and drift distances for each of the four detectors arranged in a four-by-six matrix, then padded with zeroes. It also consisted of a single convolutional layer and pooling layer, the only difference being that the convolutional layer was a three-by-three matrix, instead of the two-by-two matrix for CNN 01. Its data was flattened to a 15 element row matrix, which was then connected to a dense neural network layer consisting of five nodes. Like for CNN 01, the last neural network consisted of 58 neurons which classified the data with a softmax activation function. The convolutional layer and first neural network layer used the same activation functions and kernel initializers as CNN 01.

| | | | |
|---|---|---|---|
| x0f1 | y0f1 | tthetaf1 | tphif1 |
| x0r1 | y0r1 | tthetar1 | tphir1 |

Figure 6: The input data formation for CNN 01.

| | | | | | |
|---|---|---|---|---|---|
| n1ua | n1xa | n1va | d1ue | d1xe | d1ve |
| n2ua | n2xa | n2va | d2ue | d2xe | d2ve |
| n3ua | n3xa | n3va | d3ue | d3xe | d3ve |
| n4ua | n4xa | n4va | d4ue | d4xe | d4ve |

Figure 7: The input data formation for CNN 02.

The input data, consisting of 40,000 data points for each CNN, was split into fifths, such that four-fifths were used to train the CNNs and the remaining fifth was used to test the CNNs. This process, called k-fold validation, would be repeated five times, so each run tested on a different k-fold of the data. The batch size for both CNNs was 64, and the number of epochs varied from as low as 8 epochs, to as high as 1200 epochs, depending on the run.

# 5 Data

For each k-fold of the CNNs, the accuracies and losses of the training and testing stages as they went through each epoch were recorded and graphed. The loss is a metric in machine learning that measures how well the program learns. It decreases as the CNN finds better and better parameters for correctly classifying its inputs. The accuracy measures how many of the inputs the CNN correctly classifies.

For CNN 01, the accuracy remained around 85% for 8, 10, 50, 100, 200, 500, 800, and 1000 epochs, and the only steady decrease in loss seemed to occur for only one of the k-folds arrangement of the data. The graphs for 100 and 500 epochs are shown in Figures 8 and 9. The graphs for the other epochs can be found in the appendix.

For CNN 02, the accuracy varied among the k-folds, ranging from 70 to 75%, but stayed mostly constant throughout the epochs. It was run with 20, 50, 100, and 500 epochs, and none of the runs showed a decrease in loss. The graphs for 100 and 500 epochs are shown in Figures 10 and 11. The graphs for the other epochs can be found in the appendix.

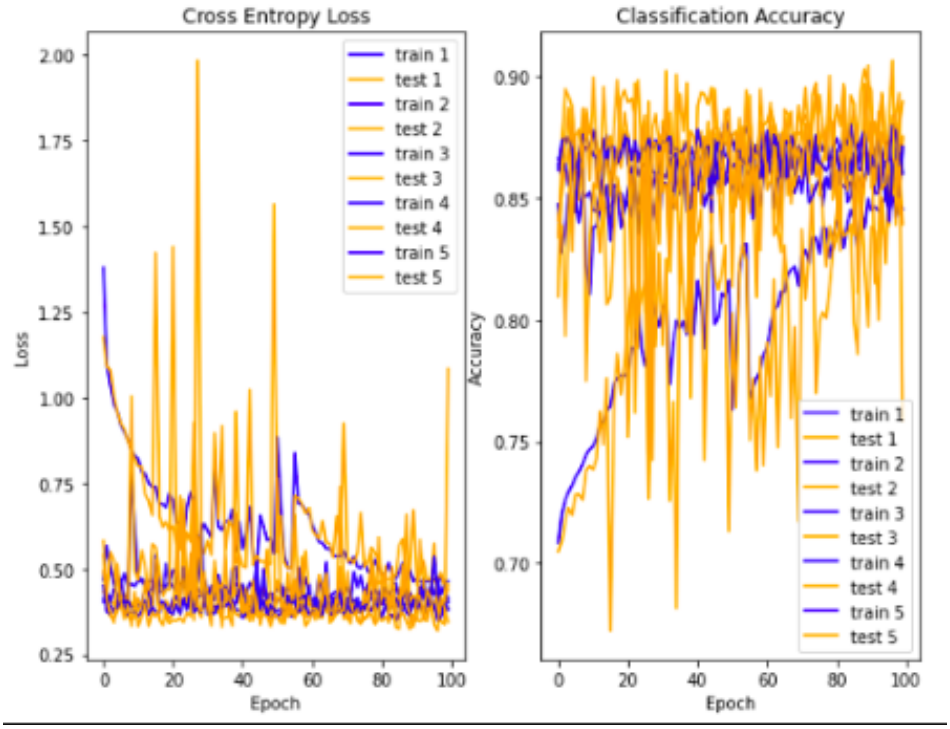


Figure 8: CNN01, 100 epochs

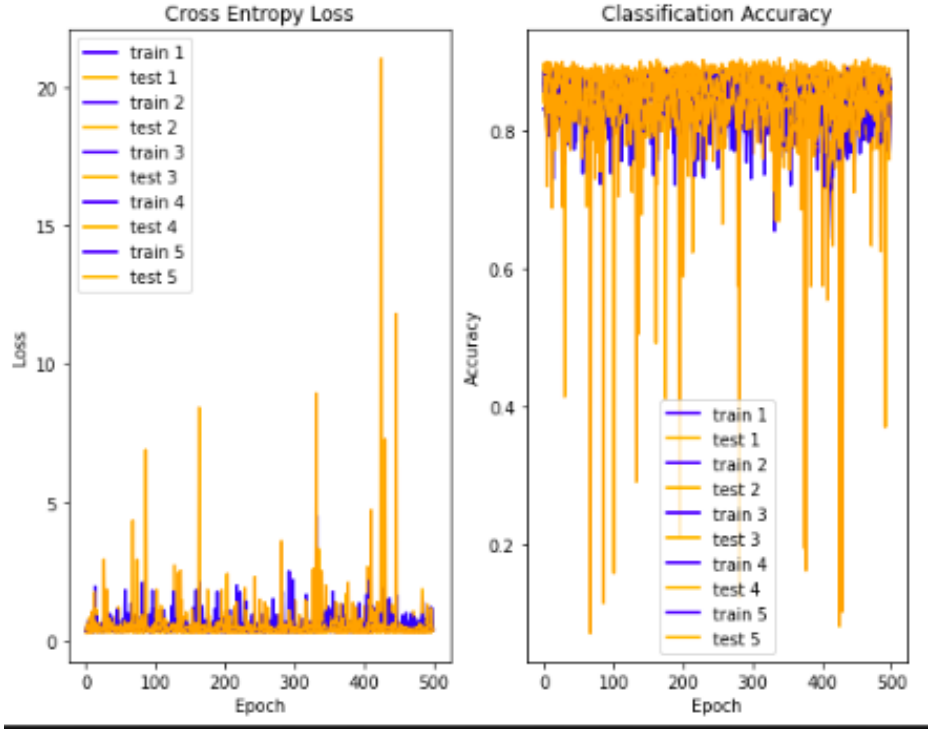


Figure 9: CNN 01, 500 epochs

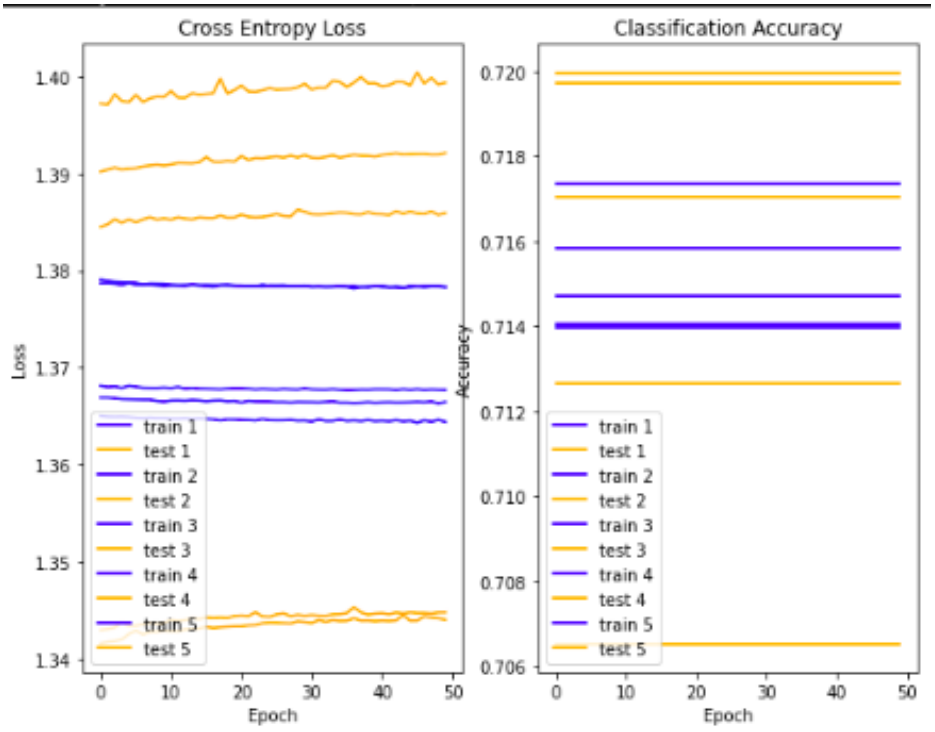


Figure 10: CNN 02, 50 epochs

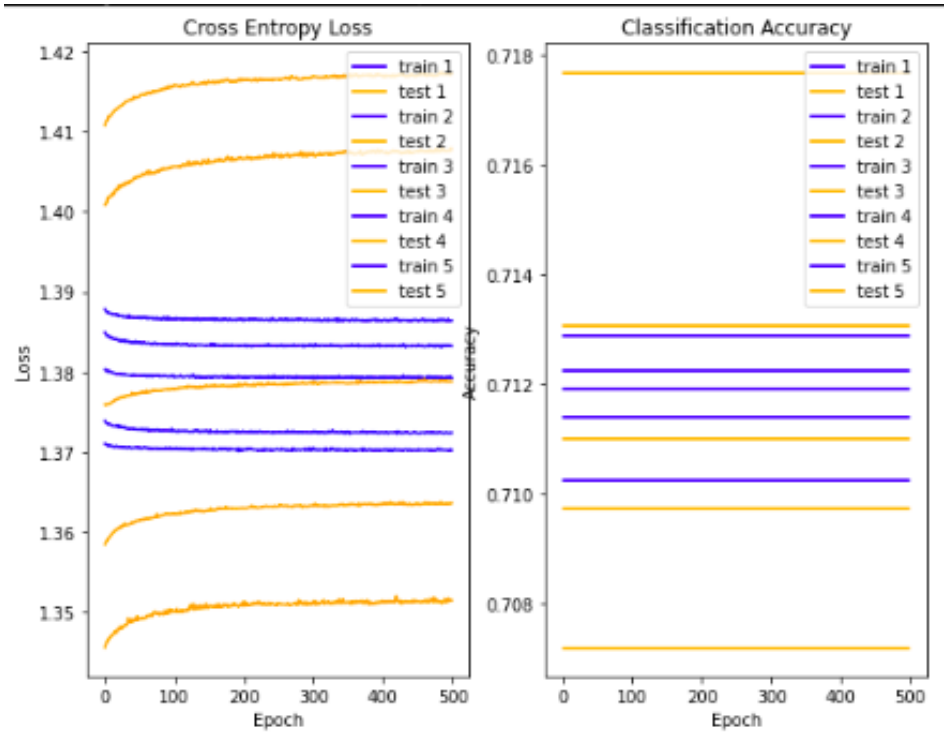


Figure 11: CNN 02, 500 epochs

# 6 Discussion and Conclusion

The results of these tests were interesting, if a bit disappointing. It was expected that as the number of epochs were increased, the accuracy would improve and the loss would decrease, in a way similar to what happened for the hand-written digit classification example (Figure 12). Instead, there was minimal accuracy increase and loss decrease for CNN 01, and neither for CNN 02, which indicates that neither CNN was improving its predictions during the training and testing stages.

This could be due to a number of reasons. Firstly, it could be that the initial formats of the input data are not spatially significant enough for the convolutional layers to be able to pick out important features for classification. That is, perhaps the data needs to be formatted in such a way so that it is visually an image of the particle moving through the detectors, rather than a matrix of its incoming and outgoing track values or a collection of the wires it hits.

Another reason the CNNs did not perform well could be the softmax activation function that was used to classify the data and the categorical cross entropy function that was used to measure the loss. Both of these were used to classify the data into one of the 58 bins created for the theta scattering angle, but perhaps using a linear activation function for classification and mean squared error function for loss calculation would be a better metric. These would allow the CNNs to learn to predict the value of the final angle, rather than trying to classify which bin the data belongs to.

Lastly, the accuracy of the CNNs were measured using a built in function the keras python library had, which measured the accuracy by calculating how many predictions the CNNs got exactly correct. It did not have a way to measure if the predictions were close to the correct answer, that is, if it were a bin or two away from the correct one. A custom function for calculating the accuracy of the predictions, then, might provide a better understanding of how the CNNs are performing and what else may need to be changed in order to improve overall accuracy.

Any of the changes listed above would be excellent next steps for this project. Once a working CNN has been created for the theta scattering angle, a second one could easily be made for the phi scattering angle using the first as a base. This project also has the potential to be applied to other kinds of experiments that share similarities with the Jefferson Lab focal plane polarimetry experiment used here.

# References


[P+12] A. J. R. Puckett et al. Final Analysis of Proton Form Factor Ratio Data at $\mathbf{Q^2 = 4.0}$, 4.8 and 5.6 GeV$^2$. *Phys. Rev. C*, 85:045203, 2012.

[P+17] A. J. R. Puckett et al. Polarization Transfer Observables in Elastic Electron Proton Scattering at $Q^2$ =2.5, 5.2, 6.8, and 8.5 GeV$^2$. *Phys. Rev. C*, 96(5):055203, 2017.

[P+18] A. J. R. Puckett et al. Technical Supplement to "Polarization Transfer Observables in Elastic Electron-Proton Scattering at $Q^2$ = 2.5, 5.2, 6.8, and 8.5 GeV$^2$". *Nucl. Instrum. Meth. A*, 910:54–78, 2018.

[Sah18] Sumit Saha. A comprehensive guide to convolutional neural networks–the eli5 way, 2018. Last accessed 21 October 2021.


# 7 Appendix

The code for this project can be found on github at johnsrgrace/minst. The graphs for all of the CNN runs are shown on the next few pages.

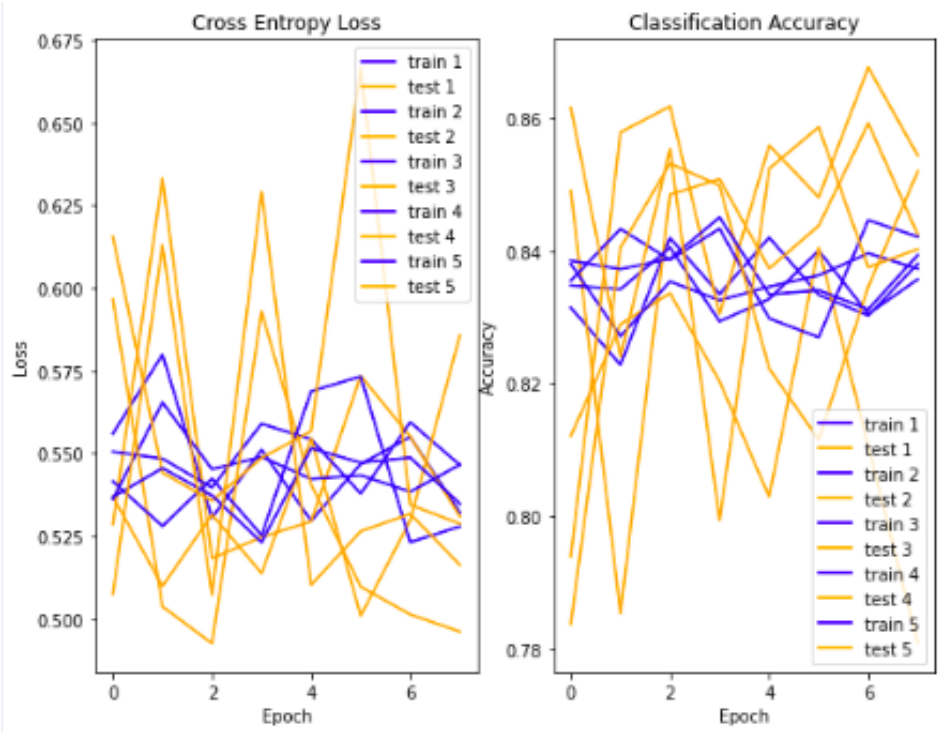


Figure 12: CNN01, 8 epochs

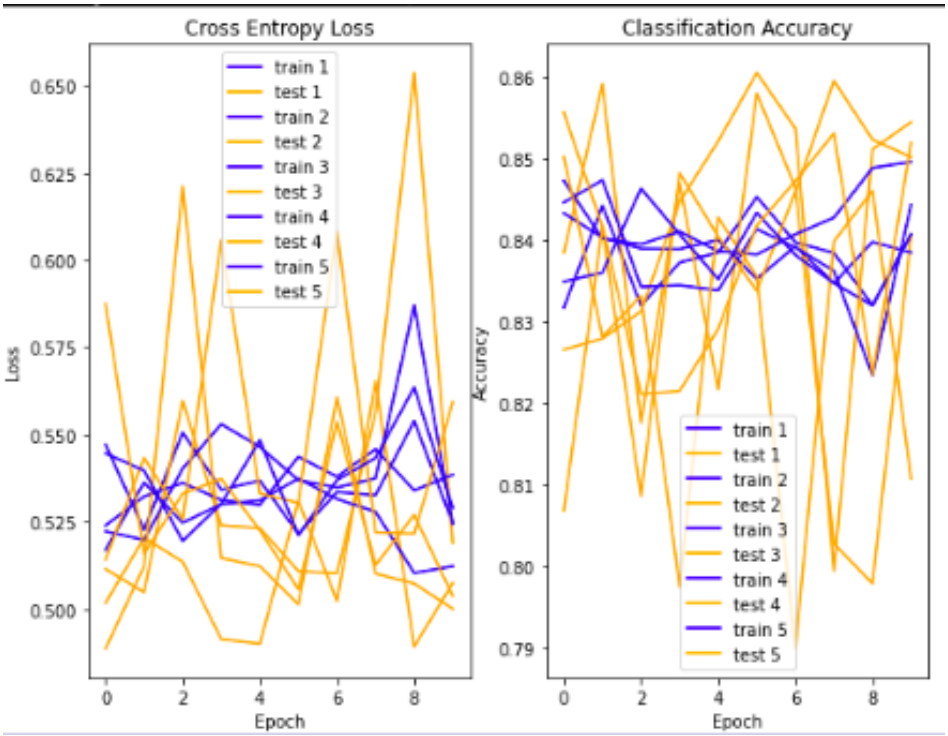


Figure 13: CNN 01, 10 epochs

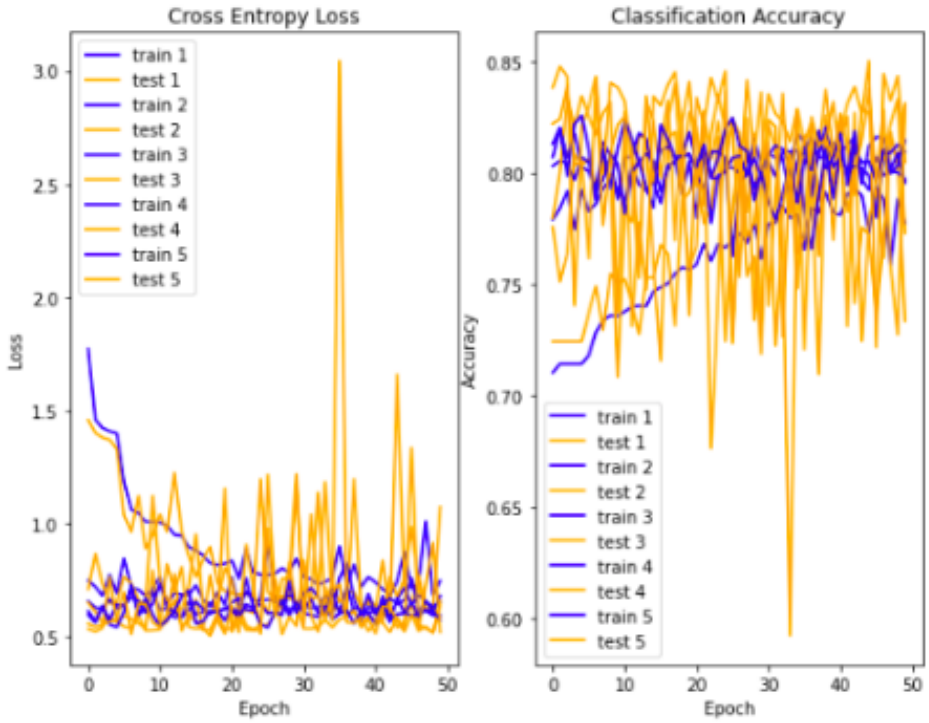


Figure 14: CNN 01, 50 epochs

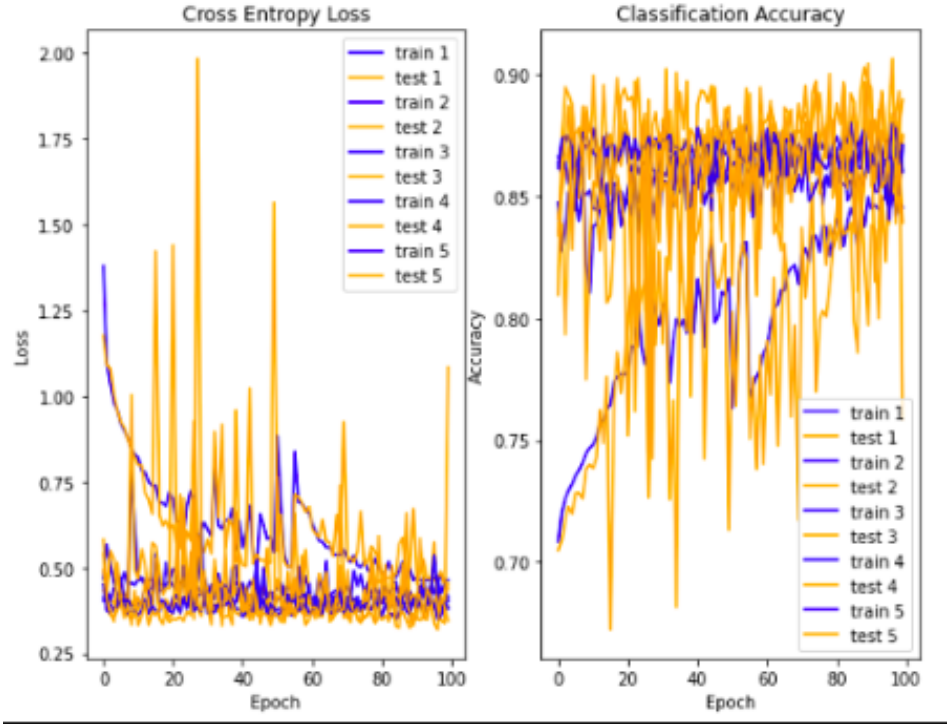


Figure 15: CNN 01, 100 epochs

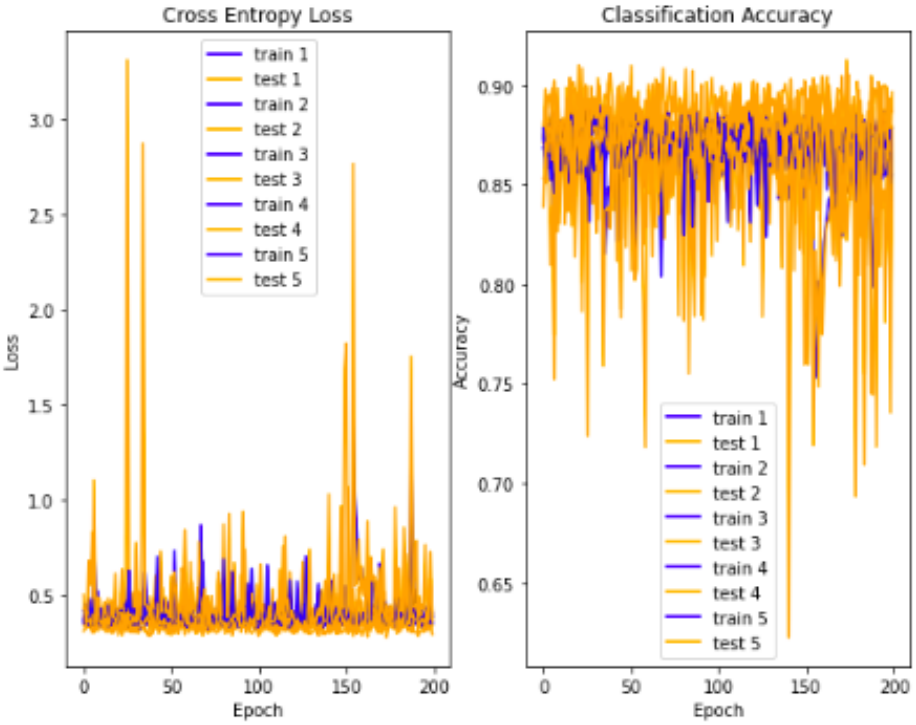


Figure 16: CNN 01, 200 epochs

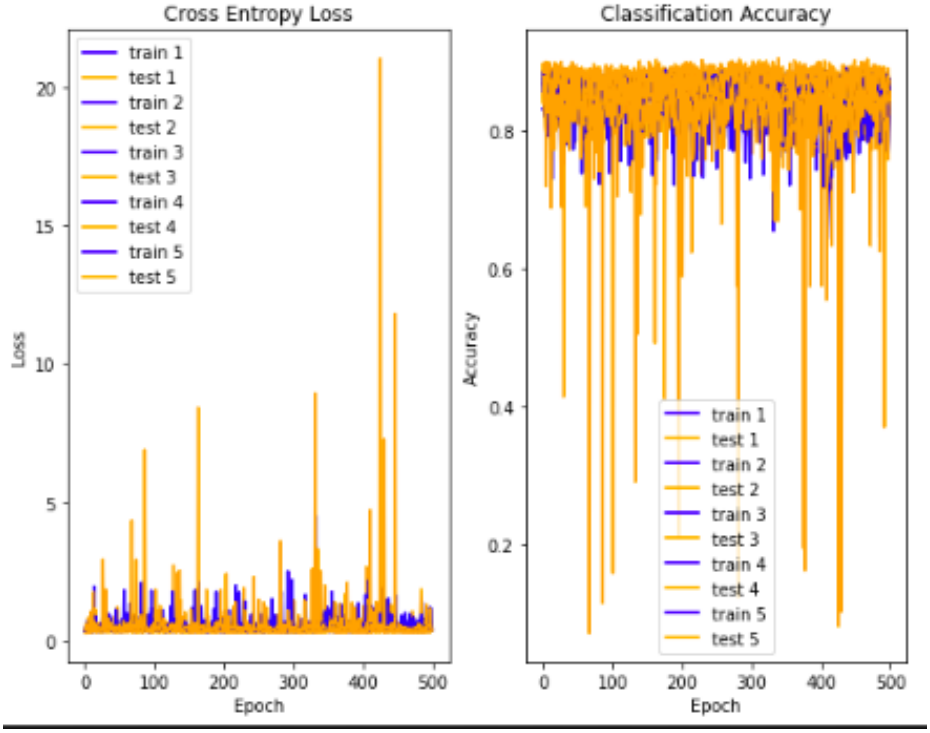


Figure 17: CNN 01, 500 epochs

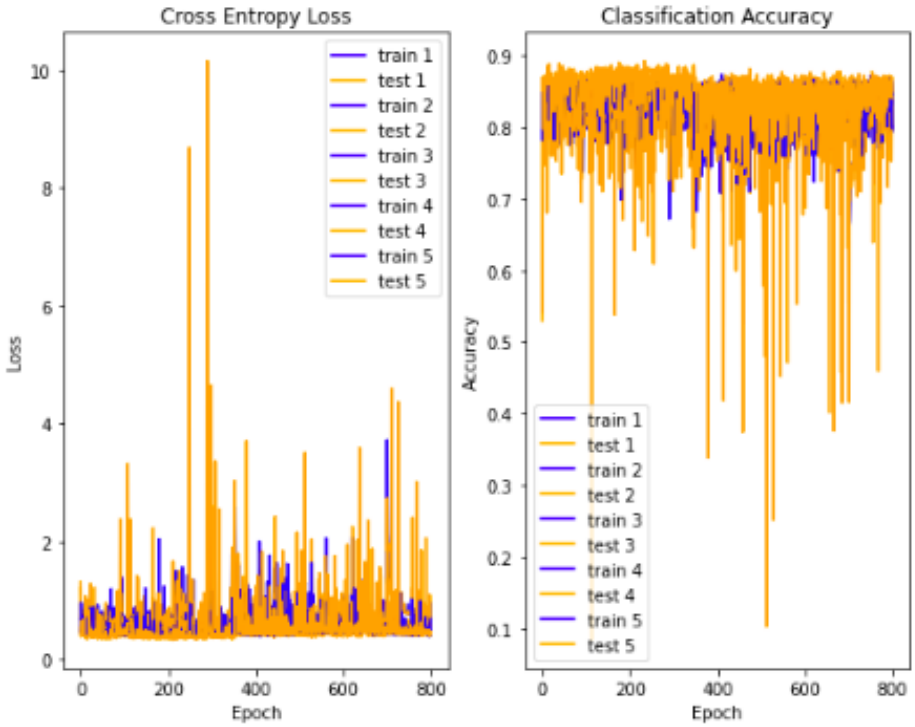


Figure 18: CNN 01, 800 epochs

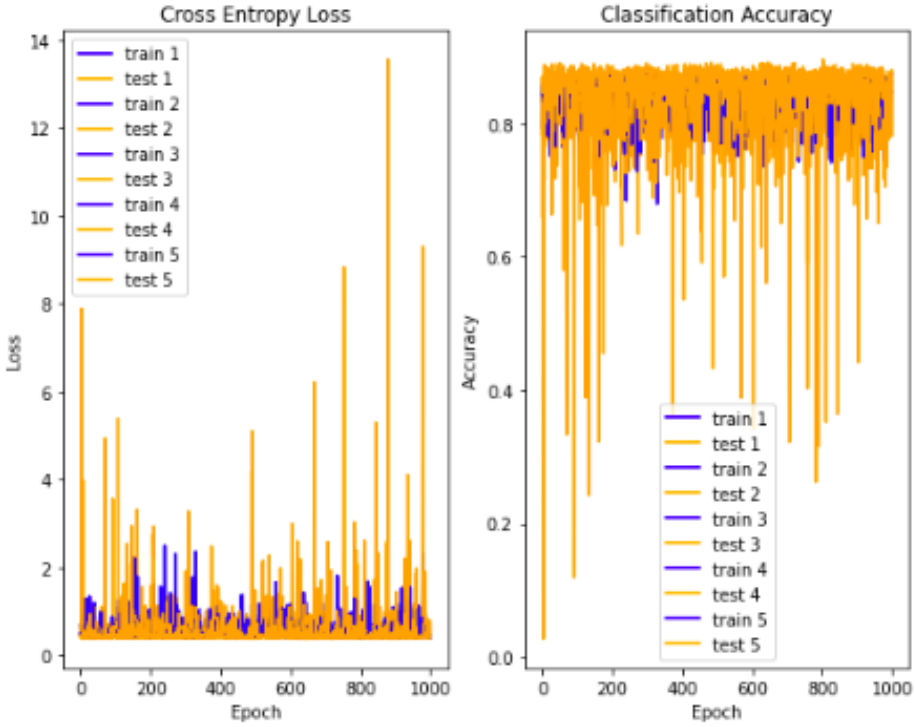


Figure 19: CNN 01, 1000 epochs

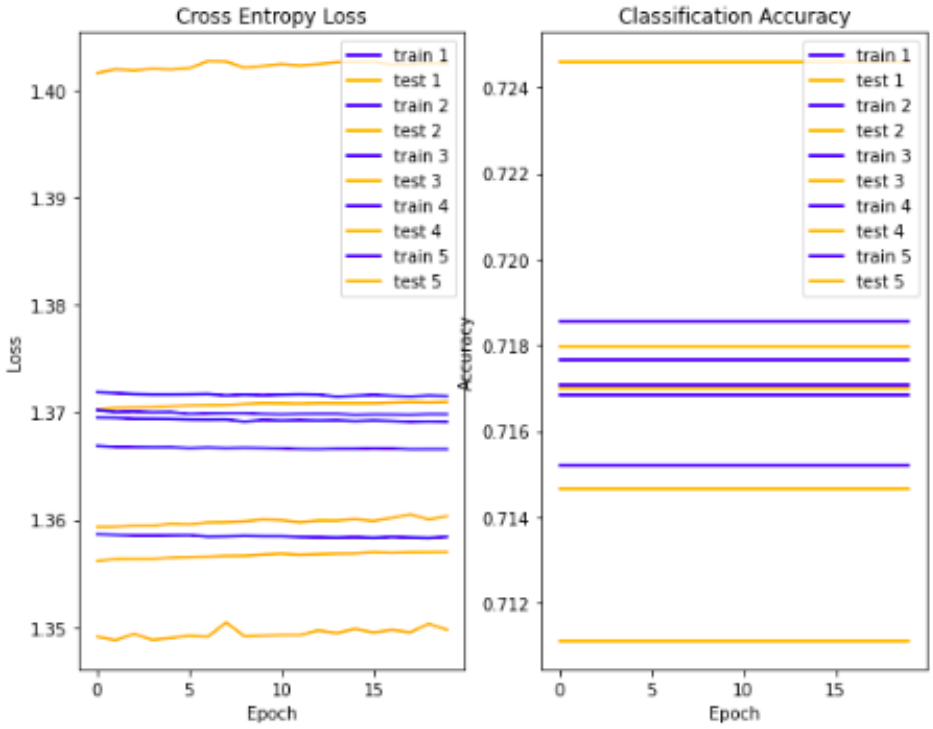


Figure 20: CNN 02, 20 epochs

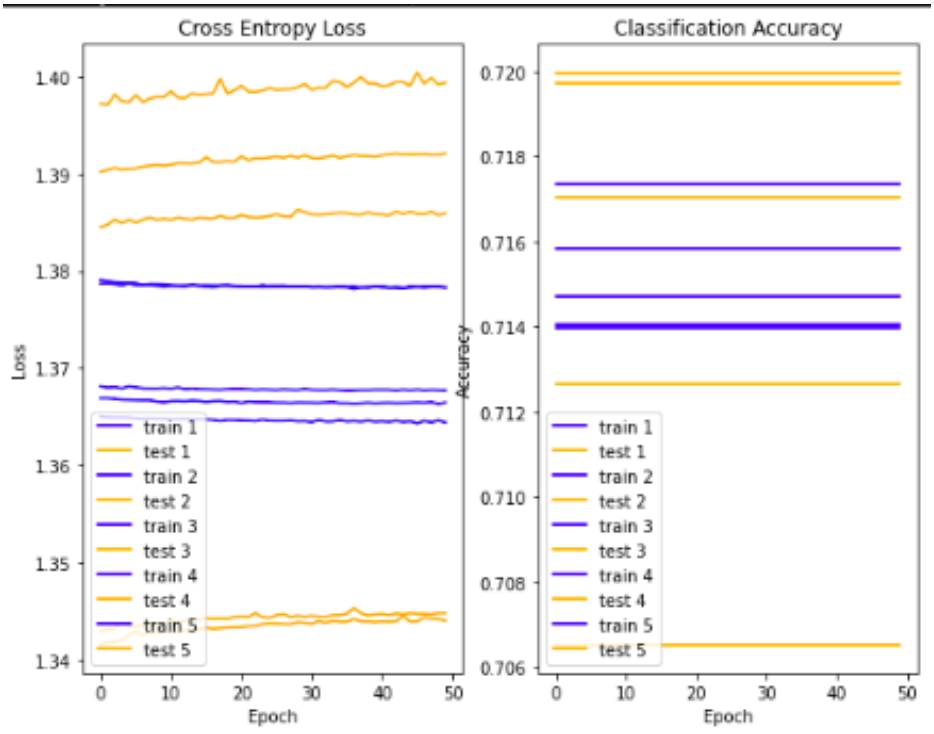


Figure 21: CNN 02, 50 epochs

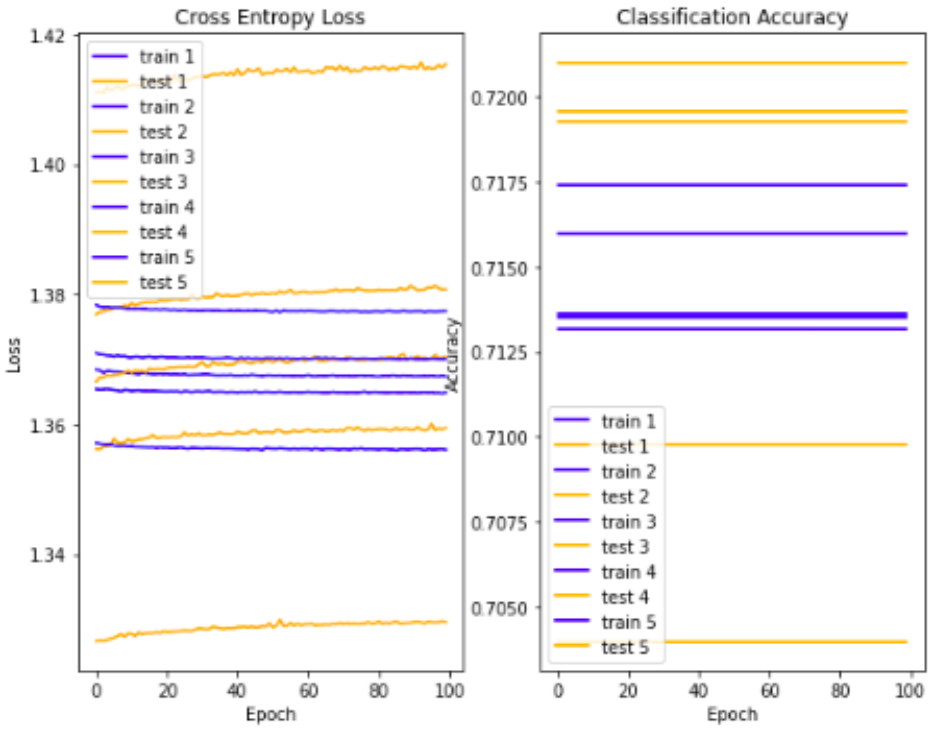


Figure 22: CNN 02, 100 epochs

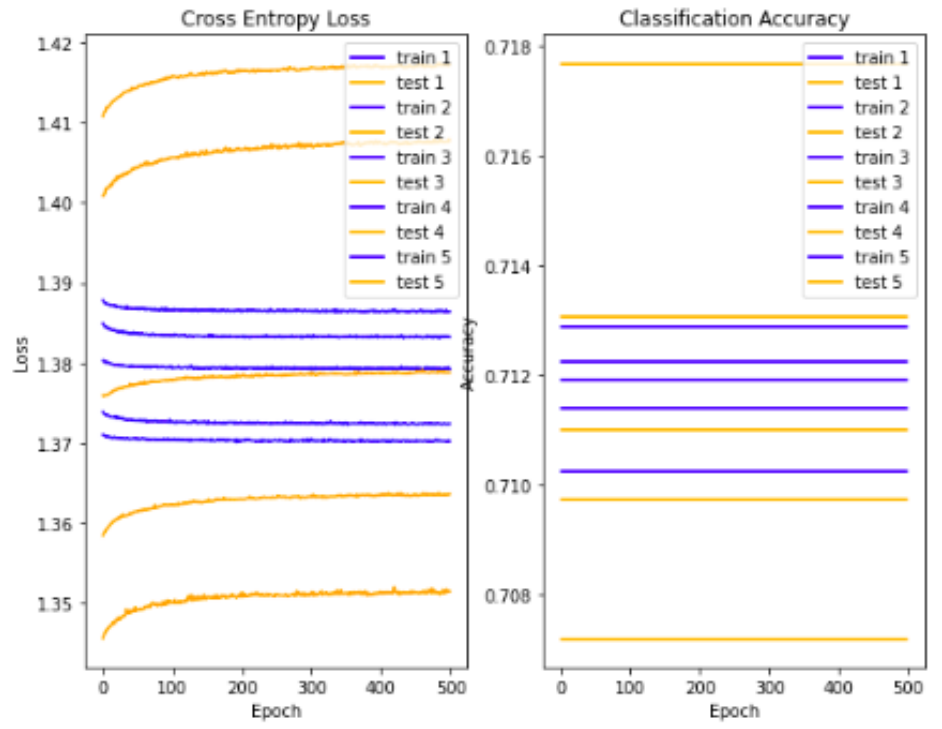


Figure 23: CNN 02, 500 epochs