Improvements to the Deep Learning Classification of Compton Camera Based Prompt Gamma Imaging for Proton Radiotherapy

Jonathan N. Basalyga[1], Carlos A. Barajas[1], Gerson C. Kroiz[1],
Matthias K. Gobbert[1], Paul Maggi[2], and Jerimy Polf[2]

[1]Department of Mathematics and Statistics, University of Maryland, Baltimore County
[2]Department of Radiation Oncology, University of Maryland School of Medicine

Technical Report HPCF–2020–29, hpcf.umbc.edu > Publications

Abstract

Real-time imaging has potential to greatly increase the effectiveness of proton beam therapy for cancer treatment. One promising method of real-time imaging is the use of a Compton camera to detect prompt gamma rays, which are emitted along the path of the beam, in order to reconstruct their origin. However, because of limitations in the Compton camera's ability to detect prompt gammas, the data are often ambiguous, making reconstructions based on them unusable for practical purposes. Deep learning's ability to detect subtleties in data that traditional models do not use make it one possible candidate for the improvement of classification of Compton camera data. The base network can be made cheaper via reducing hidden layer count while maintaining comparable classification performance. Additionally, even a simple training schedule can show improvements in the training process. Several variations of the network showed promise in their ability to classify multiple beam energies. However more improvements need to be made to the network for the performance on multiple beam energies to meet our goal of 90% classification accuracy.

Key words. Proton beam therapy, Prompt gamma imaging, Compton camera, Machine learning, Deep learning.

1 Introduction

Proton beams' primary advantage in cancer treatment as compared to other forms of radiation therapy, such as x-rays, is their finite range. The radiation delivered by the beam reaches its maximum, known as the Bragg peak, at the very end of the beam's range. Little to no radiation is delivered beyond this point. By exploiting the properties of the Bragg peak it is possible to only irradiate cancerous tissues, avoiding any damage to the healthy surrounding tissues [11]. However, without some way to image proton beams in real time, limitations exist in our ability to take full advantage of the dose delivery properties of the proton Bragg peak. This is due to uncertainties in the beam's position in the body relative to important organs that should not be irradiated.

The Compton camera is one method for real time imaging, which works by detecting prompt gamma rays emitted along the path of the beam. By analyzing how prompt gamma rays scatter through the camera, it is possible to reconstruct their origin. However, the raw data the Compton camera outputs does not explicitly record the sequential order of the interaction data which represents scatterings of a single prompt gamma ray. In addition, it often records false events, which mislabel scatterings of distinct prompt gamma rays as originating from a single ray. These problems make reconstructions based on Compton camera data noisy and unusable for practical purposes [11].

We approach these problems by leveraging several deep learning techniques. We used neural networks which, in general, represent data transformations. The network is trained by passing data through it, then updating it systematically so as to reduce the loss of its output compared with

some desired output. Doing this properly can create a model that exploits subtleties in the data which traditional models are unable to use [6].

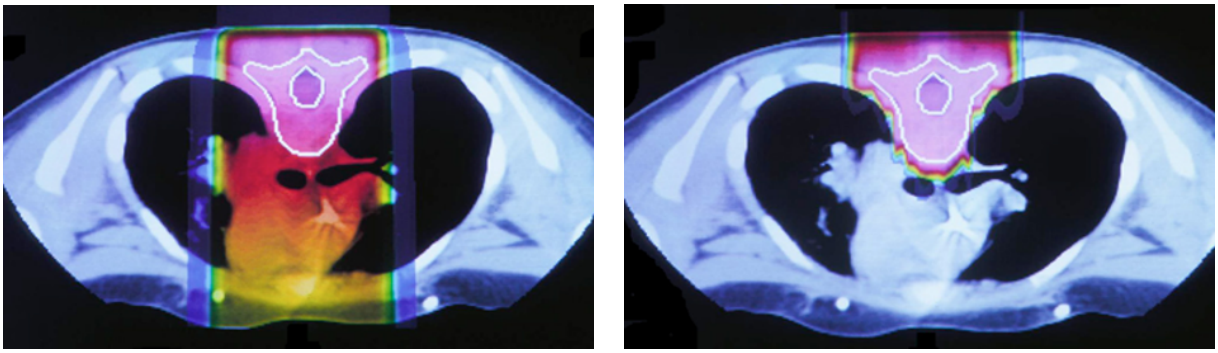
In order to reduce the noise present in Compton camera data, we train a neural network to process the data so as to reduce false events and correctly order interactions within events. We start with the network first proposed in [3] and then make a variety of changes to it. We then evaluate how these networks affect the quality of prompt gamma based proton beam reconstructions. We cannot see any strict improvements in the network's accuracy performance but we are able to make computations cheaper while maintaining competitive performance. We also test and train the network on different datasets to see if there is any difference in performance.

The remaining sections of this work are organized as follows: Section 2 introduces proton beam therapy for cancer treatment and its current limitations. Section 3 discusses how Compton camera imaging can be used to overcome the limitations of proton beam therapy and explains how the presence of false events and misordered interactions in Compton camera data limits its practical usage. Section 4 gives a brief overview of neural networks. Section 5 discusses all variations and changes we made to the network and training process used to generate later results. Section 6 details the original and current version of the preprocessing routines which must be used before our data can be fed into the network. Section 7 describes how all variations of the networks performed on different beam energies. Section 8 presents our conclusions from this work.

2 Proton Beam Therapy

Proton beam therapy was first proposed as a cancer treatment by Robert Wilson in 1946 [13]. To a first order approximation, the radiation dosage emitted by a proton beam is inversely proportional to the kinetic energy of the particles within the beam. Because the beam's particles lose kinetic energy as they traverse the patient, the amount of radiation delivered by the beam is low at its entry point, gradually rising until the beam nears the end of its range, at which point the delivered dosage rapidly reaches its maximum. This point of maximum dosage is called the Bragg peak. Little to no radiation is delivered beyond the Bragg peak. These characteristics of proton beam therapy give it a distinct advantage over x-rays. Exploiting its finite range, medical practitioners can confine the radiation of the beam to solely areas affected by cancerous tumors. Vital organs beyond the tumor can be spared [1, 7, 11].

Figure 2.1 shows two horizontal cross-sections of the chest comparing how radiation is delivered by x-ray therapy and by proton beam therapy. At the top of each image is the vertebral body,



(a) (b)

Figure 2.1: (a) X-ray treatment as compared to (b) proton beam treatment.

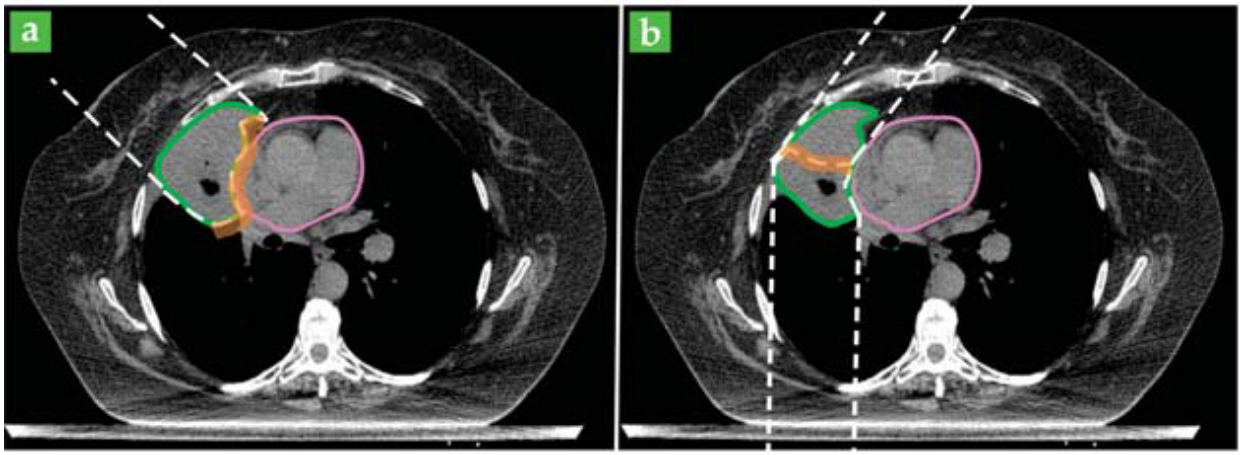


Figure 2.2: (a) Optimal proton beam trajectory. (b) Suboptimal trajectory necessary to protect heart.

which contains a tumor that should be irradiated. Since the heart, which is at the bottom center, is still healthy, any radiation delivered to it should be avoided. In the case of x-ray therapy the heart lies directly in the path of the x-rays. For proton beam therapy, however, all radiation is confined to just the vertebral body. The greater level of precision that proton beam therapy possesses allows for higher dosages of radiation to be delivered to cancerous tissues with minimal damage to healthy tissues. This can lead to better patient outcomes [11].

While the characteristics of proton beam therapy explained above would in principle greatly reduce the negative effects of radiation therapy, there are still practical limitations. In current practice the patient's body is imaged before undergoing treatment in order to map the position of the tumor. Because proton beam therapy consists of multiple sessions over a period of one to five weeks, the relative size and position of the tumor within the patient's body may change as surrounding tissues swell, shrink, and shift as a response to radiation. Therefore, whenever using proton beams, a safety margin must be added to the position of the Bragg peak in order to fully irradiate the tumor. This rules out certain beam trajectories that would otherwise minimize damage to healthy tissue [11].

Figure 2.2 compares two possible beam trajectories through a cross-section of the chest [11]. In this case the heart, outlined in purple, is positioned at top-center of the figure and a tumor, outlined in green is located next to it. The optimal trajectory, shown in the left image, uses a single beam, which is represented as the space between the dashed white lines, to fully irradiate the tumor while stopping before reaching the heart. However, due to uncertainty in the exact location that the Bragg peak occurs (and the beam stops), a safety margin is added to the optimal beam extent to ensure the tumor always receives the prescribed dose even in the presence of day-to-day changes in patient setup and patient internal anatomy. This safety margin is represented in the figure as an orange strip at the end of the beam. This partially overlaps with the heart, which would mean to possibly irradiate the heart. Therefore, in practice the trajectory in the right image using two beams is used. Because this trajectory passes through the lungs, delivering a small dose of radiation to them, it is considered suboptimal [11].

3 Compton Camera Imaging

3.1 Introduction to the Compton Camera

In order to exploit the full advantages of proton therapy, many researchers are investigating methods to image the beam in real time as it passes through the patient's body [11]. One proposed method for real time imaging is by detecting prompt gamma rays that are emitted along the path of the beam using a Compton camera.

As the proton beam enters the body, protons in the beam interact with atoms in the body, emitting prompt gamma rays. These prompt gamma rays exit the body, some of which entering the Compton camera. Modules within the Compton camera record interactions with energy levels above some trigger-threshold. These modules have a non-zero time-resolution during which all interactions are recorded as occurring simultaneously. For each interaction (that is, Compton scatter) an (x, y, z) location and the energy deposited are recorded. The collection of all interaction data that a camera module collects during a single readout cycle is referred to as an event. [10].

In principle it is possible to use the data that the Compton camera outputs (paired with a suitable reconstruction algorithm) in order to image the proton beam, however this has been shown to only be feasible at low energy levels. At the higher energy levels more typical of proton beam therapy, reconstructions of the beam are far too noisy to be helpful. This is a result of two main limitations in how the Compton camera records events [10]:

- Reconstruction methods typically require that all interactions in an event be chronologically ordered by their occurrence. However, as noted above, due to the camera's non-zero time-resolutions, the camera records all interactions within an event as occurring simultaneously. Therefore, the order of interactions that it outputs is arbitrary.
- Reconstruction methods also assume that all interactions in an event correspond to the same prompt gamma ray. However, since the Compton camera classifies all scatters occurring in the same module during the same readout cycle as belonging to the same event, should two prompt gamma rays enter the same module of the Compton camera during the same readout cycle, the camera would record the resulting interactions in a single event. This results in events that do not correlate to any actual physical event, which are referred to as false events.

At the higher energy levels typically used in treatment, proton beams emit a larger number of prompt gamma rays per unit time, increasing the likelihood of false events. Also, prompt gamma rays are more likely to scatter at higher energy levels, leading to more multi-scatter events, which, as explained above will be unordered. These two effects greatly diminish the accuracy of Compton camera reconstructions at high energy levels, making them unusable [10].

3.2 The Representation of Events

Multi-scatter events can be classified into five categories. A False Triple event consists of three interactions which all originate from separate prompt gamma rays that happened to enter the same module of the camera at the same time. These should be removed from the data before reconstruction. Similarly, False Double events contain two interactions originating from separate prompt gamma rays. These too should be removed. A Double to Triple event contains two interactions corresponding to the same prompt gamma ray, and one interaction from a different prompt gamma ray. The non-corresponding interaction should be removed before reconstruction. The two remaining categories of events are true double and true triple events, which, once properly ordered, can be used for reconstruction.

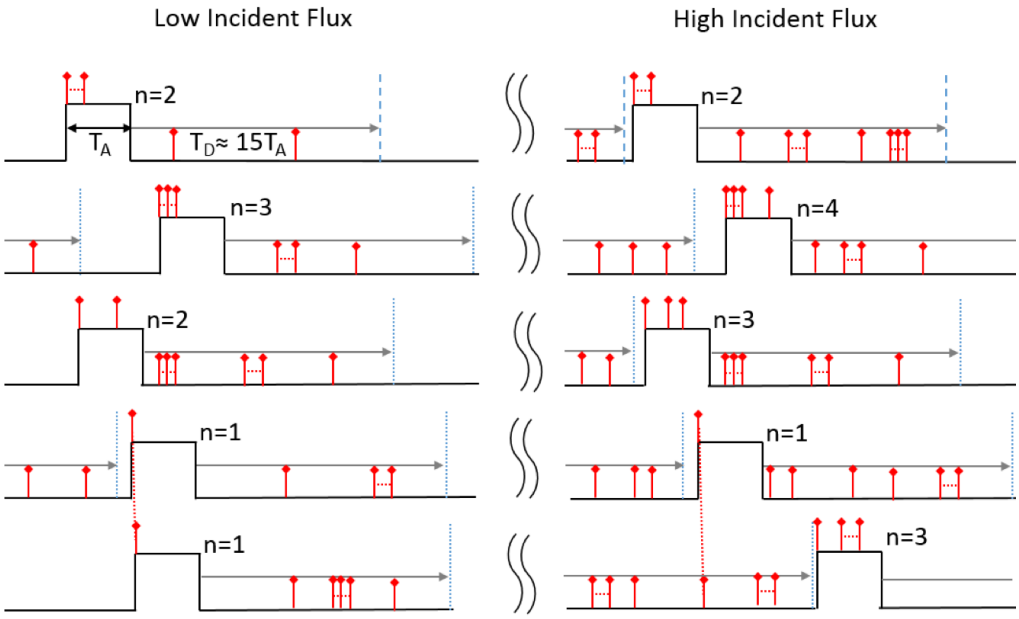


Figure 3.1: An illustration of events.

Figure 3.1 shows a schematic of the Compton camera as it records events. The left side shows events produced at low energy levels and the right shows higher energy levels. Each row represents an independent module of the camera. The red arrows represent scatters, with those originating from the same prompt gamma ray being connected by a dotted line. A single readout cycle within a module of length T_A is represented by a raised pulse. The value n is how many interactions occur during the readout cycle. Looking at just the left side, the first two rows show a True Double and True Triple event, respectively. The third row shows a False Double event consisting of two scatters originating from different prompt gamma rays. The fourth and fifth rows show two True Single events that consist of separate scatters by the same prompt gamma ray. The right side representing higher energy levels shows a far greater proportion of false events.

The raw data output by the Compton camera contains the information shown in Figure 3.2 (a). The entire matrix represents an entire event, while each row represents a single interaction. There are three rows because an event can contain up to three interactions. The variable e_i represents the energy level of the ith interaction, where $i = 1, 2, 3$, while (x_i, y_i, z_i) represents the corresponding position. Note that data representing double events still contains three rows even though double events contain only two interactions. This is necessary, since the networks we introduce in Section 7 expect input data to be a consistent size. To account for the missing interaction the third row will be zeroed out. Similarly, single events would have two rows zeroed out, however, single events have only one possible ordering, so there is no need to classify them.

To improve the performance of our networks, we find it useful to use the appended data shown in Figure 3.2 (b). In this version we add the distances $\delta r_{i,j}$ and the differences in energy levels $\delta e_{i,j}$ between the ith and jth interactions, where $i, j = 1, 2, 3$. Since these values have physical significance with regards to the ordering of interactions, explicitly including them in the data makes it easier for the networks to learn. For double events, the non-relevant $\delta r_{i,j}$ and $\delta e_{i,j}$ are set to zero. We use the value zero rather than `NaN` because our networks require that all data consist of real numbers.

e_1	x_1	y_1	z_1
e_2	x_2	y_2	z_2
e_3	x_3	y_3	z_3

(a)

e_1	x_1	y_1	z_1	$\delta e_{1,2}$	$\delta r_{1,2}$
e_2	x_2	y_2	z_2	$\delta e_{2,3}$	$\delta r_{2,3}$
e_3	x_3	y_3	z_3	$\delta e_{3,1}$	$\delta r_{3,1}$

(b)

Figure 3.2: (a) The initial input format representing a single event. (b) The appended input format including distances and energy distances between each interaction.

4 Deep Learning

We propose to train a neural network to process the data output by the Compton camera by removing false events and properly ordering the interactions within events.

The structure of a fully connected neural network is shown in Figure 4.1 [2]. The network contains three main components: an input layer which accepts the data, hidden layers which each perform some transformation on the data, and an output layer which returns the transformed data in some prescribed format [6]. We would like to train the neural network to transform the provided data in some useful way. In the case of the data output by the Compton camera, we would like the neural network to transform each multi-scatter event so that it contains only interactions originating from the same prompt gamma ray, and so that these interactions are in the correct order.

Figure 4.2 shows the training and testing process for a neural network during what is called supervised learning [2]. Supervised learning refers to training the neural network using labels for the data which provide what the transformed data is supposed to look like. By feeding data into the network and comparing it with the corresponding labels using a suitable loss function, we can calculate the current loss of the neural network. The neural network can then be updated using an optimization function. After training the network, it is then tested on data it has not seen before. If the network performs well on data it was not trained on, this indicates that the network's model generalizes well and can be used on real-life data.

To improve the network's performance, it is typical to train the network on all available data multiple times. One pass through all the training data is referred to as an epoch. Often, the network will be trained for hundreds or thousands of epochs. It is standard practice to set aside some data with which to evaluate the network after each epoch. These data are called the validation data. By evaluating the network at the end of each epoch, it is possible to plot how the network's

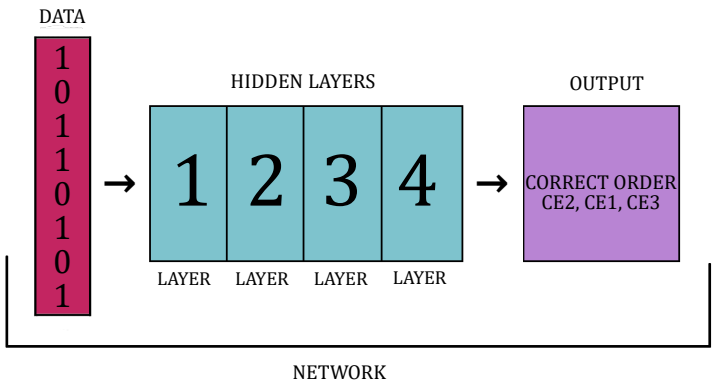


Figure 4.1: The structure of a fully connected neural network.

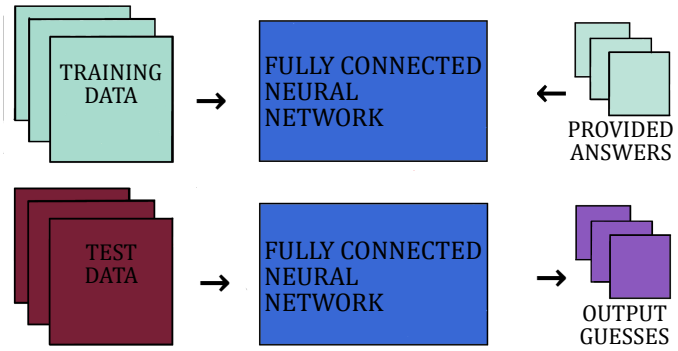
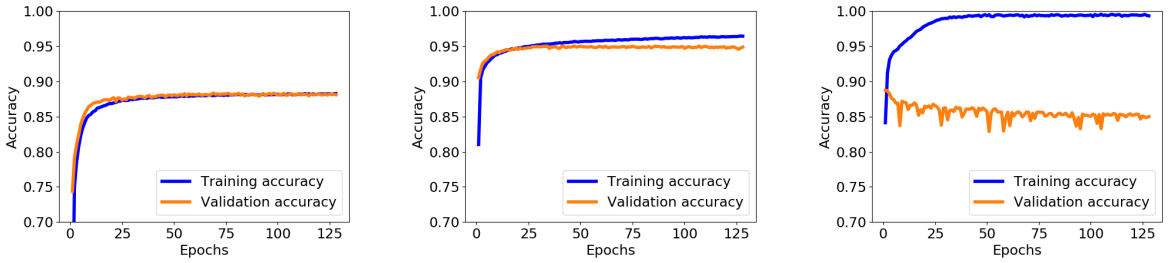


Figure 4.2: The training and testing process for a fully connected neural network.

performance improves over the training process, giving insight into whether or not the network has been fully trained. After the network has finished training, a final data set separate from the training data and validation data is used to test the network. This data set is referred to as the test data.

One of the primary difficulties in deep learning is training a network so that it properly fits the data it is being applied to. When there is still information in the data that has not been incorporated into the network's model this is referred to as underfitting. This occurs because the network has either not been trained enough, or because the network size is too small to fully process the data. When the network performs very well on the data it has been trained on but does not generalize to data that it has not been trained on, this is called overfitting. Overfitting occurs because the network has begun directly mapping inputs to outputs, that is, "memorizing" the data. Since large networks have a greater capacity to store information about the data, they are more likely to overfit. Therefore, using a larger network does not necessarily lead to better performance [6].

Figure 4.3 compares the accuracy curves of three networks which originating from a textbook example as they are being trained [6]. In each plot the blue curve represents training accuracy, that is, how well the network performs on training data at each epoch, while the orange curve represents validation accuracy, which measures how well the network performs on a validation set composed of data the network was not trained on. The left plot shows training and validation curves typical of underfitting. Since the network still has not fully internalized distinguishing features in the training data, it performs just as well on the validation data as it does on the training data. The plot on the right shows an overfitting curve. Here there is a large gap between training accuracy and validation accuracy, indicating a lack of generalization. The plot in the center shows what the training and validations curves of a suitable fitting look like. The training and validation curves are just beginning to diverge, but are still very close. This occurs because the network has incorporated as much information from the model as it can, so any additional training either has no effect on the validation accuracy, or even lowers it.



(a) underfitting (b) suitable fitting (c) overfitting

Figure 4.3: The training and validation accuracy curves representative of underfitting, a suitable fitting, and overfitting.

5 Network Design Options

The studies in this work use a distributed-memory cluster of compute nodes with large memory, and connected by a high-performance InfiniBand network. Both the 2018 and 2013 GPU nodes feature two multi-core CPUs, while the 2018 GPU node has four GPUs and the 2013 GPU nodes have two GPUs. The following specifies the details:

- **2018 GPU node**: 1 GPU node containing four NVIDIA Tesla V100 GPUs (5120 computational cores, 16 GB onboard memory) connected by NVLink and two 18-core Intel Skylake CPUs. The node has 384 GB of memory (12×32 GB DDR4 at 2666 MT/s).
- **2013 GPU nodes**: 18 hybrid CPU/GPU nodes, each two NVIDIA K20 GPUs (2496 computational cores, 5 GB onboard memory) and two 8-core Intel E5-2650v2 Ivy Bridge CPUs (2.6 GHz clock speed, 20 MB L3 cache, 4 memory channels). Each node has 64 GB of memory (8×8 GB DDR3). The nodes are connected by a QDR (quad-data rate) InfiniBand switch.

These nodes are contained in the cluster taki of the UMBC High Performance Computing Facility (HPCF), whose webpage at `hpcf.umbc.edu` can provide more details.

All studies and preprocessing using one or more of the following python packages with the respective version:

- Python 3.7.6,
- Tensorflow 2.4.0 and the bundled Keras ,
- Numpy 1.18.1,
- Scipy 1.4.1,
- Pandas 1.1.0.dev0+690.g690e382 (configured for icc 19.0.1.144 20181018),
- mpi4py 3.0.3.

5.1 Activation Functions

Activation functions are one of the core parts of neural networks. Consider a single fully connected layer. By definition, the unknowns of a fully connected layer are a weight matrix A with a bias vector b. For simplicity we will use only a single record for x. Take some record x and do a matrix

vector product such that $y = Ax + b$. To add another layer we repeat this operation again with a weight matrix K and bias vector d such that $Ky + d = z$. Now we have two fully connected layers with no activator. When we expand z we get $z = KAx + (Kb + d)$. With a simple replacement of $F = KA$ and $t = Kb + d$ we have $z = Fx + t$. Since we are only interested in finding out how x becomes z there is no need to solve for K, A, d, or b, we can solve for F and t instead of the other unknowns. In order to increase the problem complexity and artificially enforce the importance of all unknowns we use non-linear functions called "activation functions". We take a non-linear function like tangent and apply it element wise to a matrix or vector. By using an activation between our two layers and expanding z we get $z = K\tan(Ax + b) + d$ where tangent is applied element-wise to $Ax+b$. By weaving these non-linear functions into our compositions we introduce non-linearity and create a situation where the weights of both A and K need to be solved for. For a more in-depth explanation about the underlying mathematics associated with fully connected networks see [12]. The struggle we have now is choosing a non-linear function to use.

In [3] we used a Scaled Exponential Linear Unit (SeLU) proposed in [9] and stated as

$$s(x) = \begin{cases} \lambda x & x > 0, \\ \lambda\alpha e^x - \lambda\alpha & x \leq 0. \end{cases} \tag{5.1}$$

The constants α and λ could be treated as hyperparameters but [9] actually computed optimal values with proof in the publication. One of the major benefits of SeLU is that it has a self-normalizing property. By bundling the normalization into the activator we actually make the network cheaper by removing all batch normalization which occurs between layers.

Through experimentation with hyperparameter settings we found that normalization shows no noticeable accuracy benefits. Since batch normalization seems to have little, if any, impact on our outcomes the self-normalizing property of SeLU is more of a hindrance. Why bother with the expense of e when we can opt for a cheaper activator function like ReLU or Leaky ReLU.

ReLU, otherwise known as

$$r(x) = \begin{cases} x & x > 0, \\ 0 & x \leq 0, \end{cases} \tag{5.2}$$

is one of the more commonly used activators and is discussed in [6]. Notice how it has no mathematical operations but is still a non-linear function making it extremely cheap compared to SeLU. When neural networks use ReLU and become sufficiently deep they experience a "dying" effect. The neurons gradually become 0 as the network feeds network forward and during back propagation these 0 neurons cause a 0 gradient. This makes ReLU incompatible with our desire to create a very deep network. Instead we opt for Leaky ReLU

$$l(x) = \begin{cases} x & x > 0, \\ \beta x & x \leq 0, \end{cases} \tag{5.3}$$

which has single multiplication with a small positive constant β. We consider β to be a hyperparameter which can be tuned but we decided to use Keras' default value of 0.3. This non-zero β fixes the dying gradient problem seen with ReLU. With Leaky ReLU we get a much cheaper activator which uses only scalar multiplication instead of scalar multiplication, exponentiation, and subtraction combined.

5.2 Wide Network to Long Network

In [3], we hit a GPU memory limit using the due to the size of our network. The network itself was several GB on disk and took ten to twenty seconds to load onto the GPU. In order to use the

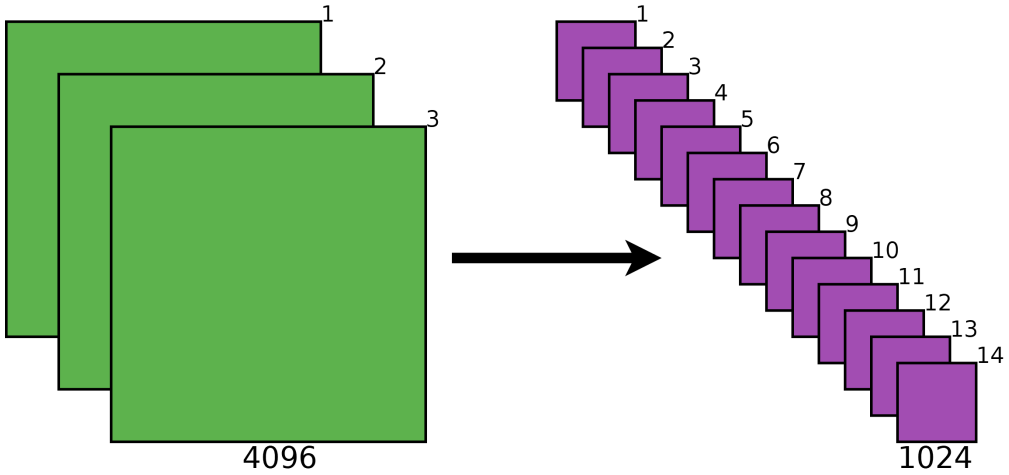


Figure 5.1: An example of transitioning from a network with few layers with many neurons per layer to a network with many layers and few neurons per layer.

network, we were limited in our choices for batch size and could not run any power of two beyond 4096 and expect the run to finish. If we did a run which had near perfect memory usage the run would fail within several epochs due to small amounts of memory leakage present in Keras and Tensorflow.

To allow for more creative changes to the network, we decreased the neurons per layer from 4096^2 to 1024^2 and increased the number of layers from 24 to 256. This causes us to go from about 400 million total neurons to about 268 million total neurons. This design shift is captured visually in Figure 5.1. Our longer network has fewer total neurons but the data is passed through significantly more non-linear functions. Ideally, our long network should be just as capable of learning from the data as the wider network. With a long network that has considerably fewer neurons per layer, we can start shrinking and cutting layers without needing to worry about whether the network should be longer than it is now. This allows us to easily make the network cheaper to train and test with very minor changes.

With the wide network, we are forced to remove layers or shrink dimension. If we shrink the dimension of the wide network then we have to ask, "would the network perform better if it was longer with same neurons per layer?", which further increases the complexity and scope of our hyperparameter searches. With the long network, we are confident that network elongation would serve little benefit for a fully connected network which maintains a constant neurons per layer even if the neurons per layer was decreased.

5.3 Alternative Training Methods

We attempt to determine if alternative training routines for our network could be of any benefit. Consider that since the creation of Generative Adversarial Networks (GANs) by Goodfellow in [8], other researchers have found that special training processes help improve training quality. Initially, the discriminator was trained on false data and real data simultaneously. However after years of researchers working with GANs, many rule of thumbs for the training process have bubbled to the surface. When training the discriminator we see in [4] that one should train on real data and false data separately on a per epoch basis.

The parallel we draw is that given false inputs and real inputs we feed one input category at a

time to the GAN's discriminator. If we train first on triples, then doubles, then doubles to triples, and so on, we would be training our network in a similar fashion to how the a discriminator is trained. We aim to create a generator which does this automatically in conjunction with Keras to train a network.

The generator performed several operations, given only the input data, output data, and batch size. First the generator converted the one hot encoded output into an array of index values. The value of every entry in the new array was the index of the 1 in the one hot encoded vector. Now the generator knows that if a 1 was present in the first 6 places, it was a triple event. The following 2 entries were doubles. The next 6 entries after the doubles were double to triples. The last entry represents false events. Then the generator isolates all data associated with a particular input category. This isolation process creates a view of the input category. The new data structure has a starting index of 0 and a final index equal to one less than the number of entries. The generator then creates an incremental array which starts at 0 to one less than the number of records. We treat this new array as a collection of indices of the view and shuffle it. We then batch up the array of indices. If the number of entries does not evenly divide by the batch size, then the final batch is a batch of the remaining entries. The generator proceeds to iterate over the batched indices. Given a batch of indices the generator returns a view of a batch of input/output data by indexing the input category view with the batch of indices. It continues this batching until it has exhausted all batches under an input category at which point it moves on to the next input category. When all input categories have been handled we know one epoch has elapsed. We repeat the entire process again until all epochs have completed.

5.4 An Efficient Prism Shaped Network

One of the major hurdles of our network is briefly detailed in Section 5.2. The network is very long and, as seen in Section 7.1, takes a very long time to train. Here we play with the idea that we can cut away layers but maintain accuracy. It was shown in [5] that deep networks actually develop redundancy and hold duplicate bits of information everywhere. By shrinking the network, thereby reducing the redundancy, we can speed up the time it takes to train while retaining all the important knowledge the network has acquired. This work motivates the results in Section 7.2.

To go beyond simply shrinking the network, we also intend to use all of the tools possible to improve training times and accuracy out of the network. In a typical optimization problem we use algorithms to determine the optimal direction to move in and we have a separate algorithm to determine how far in that direction we should move. In the case of a neural network our optimizer only chooses the optimal direction. The step size is actually retitled to be "learning rate" in the context of neural networks. With this idea in mind, we can draw some basic conclusions about how learning rate should affect the iterative or "epoch" process. A constant learning rate which is sufficiently small should give you convergence to your minimum but you pay a price in time. It will take significantly more operations at a small constant learning rate than a large learning rate. If you have a large learning rate of, say 1, your network will move huge jumps in accuracy and loss. When the learning rate is very large during the entire training process, it is likely that you will be unable to reach the minimum. This is because your current point in the loss landscape is close enough to the minimum that the learning rate pushes you past the minimum rather than closer to it. What type of learning schedule we use is a deep rabbit hole in its own right. With a training schedule we are essentially creating an algorithm that takes some data about our neural network's status at that epoch and then returns the learning rate to be used at that point. The starting learning rate we leave coarse. In general, the tightening occurs when a run without a schedule would normally stagnate with the given accuracy. The exact values and number of tightenings are,

themselves, hyperparameters. We explore this idea in Section 7.4.

We then take the lessons learned and create a shorter network with a simple learning schedule which produces the results in Section 7.5.

6 Preprocessing

6.1 Method for Class Generation from the Proton Data

This section explains how we create classes for the network from the Compton camera simulation data. This was already used in [3], though not documented. Let T, D, and S be the number of triples, doubles, and singles respectively. Keep in mind that we have significantly more singles than doubles and significantly more doubles than triples. We use the number of triples as our basis for how many events per class we should have. Triples only have 6 possible orderings and to keep our input classes balanced, we know that there must be $T/6$ many events per class.

Now we will proceed to detail the process through which we balance and generate our data. Our first goal, as seen in Figure 6.1, is to take the simulated camera data and split it into the five input categories: triples, double to triples, doubles, false doubles, and false triples. First, we load in a perfect data file produced by a proton beam simulation. Then we split the data file into triples, doubles, and singles. From our D doubles, we randomly select $2T/6$ doubles and set them aside for later use. We then randomly select T doubles from the remaining $r = D - 2T/6$ doubles and T many singles to combine into T double to triple events. We use some distance calculations to ensure that the selected singles are in the same module as their doubles while also being a reasonable distance away from the doubles. If we do not force a single to be a minimum of 1 pixel away from the double, then the camera never would have detected it in the first place. To make false doubles we have to combine two singles which are at least 1 pixel away and in the same module. To do this, we randomly choose $T/6$ many singles to create $T/6/2 = T/12$ false doubles. To make false triples, we have to combine three singles which are at least 1 pixel away from each other and in the same module. We randomly choose $T/4$ many singles to create $T/4/3 = T/12$ many false triples. For the purpose of future reordering, we ensure that for the double to triples, the single is always the third interaction.

At this point we have gone to great lengths to ensure that our data set has the correct proportions so that, post-shuffling, it is completely balanced. Now we must split our categories into classes. We know that there are six different interaction arrangements for our triples. We know that every arrangement is just as likely as any other. We take, then shuffle the triples such that there are $T/6$ triples for each arrangement as seen in Figure 6.2. Next we take the $2T/6$ doubles and swap the interactions for half of them giving us $T/6$ perfect doubles and $T/6$ swapped doubles. We know that there also exist six different arrangements for double to triple events. Similar to how we handled the triples we take the T double to triples and put $T/6$ of them into each arrangement. We do not swap false doubles or false triples because doing so serves no purpose. It is important to know that we have $T/12$ false doubles and $T/12$ false triples as they both fall under the false category giving us $T/6$ false events in total.

We then normalize the data by feature using sklearn's normalizer.

6.2 Updated Method

While looking through the original data we realized that the data generator produced triples and doubles which were technically impossible. Some of the scatters for double and triples were not in the same module as their other scatters. The design of the Compton camera is such that it would

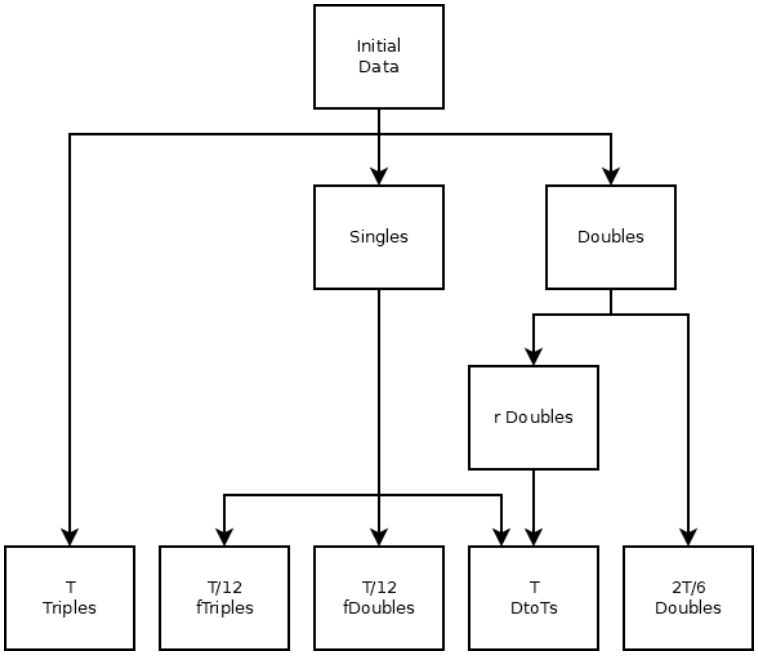


Figure 6.1: We leave some $2T/6$ doubles and all T triples untouched. We pair some of the remaining r doubles with a single to make T many double to triple events. We also use singles to generate $T/6$ false events.

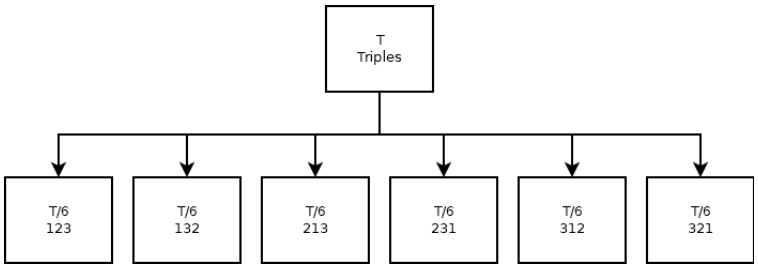


Figure 6.2: We split the T many triples into their 6 different orderings such that each ordering has $T/6$ many events.

never produce data that had interactions from different modules for an event. The proton beam simulator does not have this restriction. To accommodate this difference we removed all events which had inter-module interactions after separating the data file into doubles, triples, and double to triples.

After shuffling and data generation has finished we then compute and add the module number to each event.

6.3 Complete Rewrite and Streamlining Process

The original preprocessing methods that were used in [3] and detailed in Section 6.1 had several problems. The largest problem is that the preprocessing method systematically destroyed data. It is not clear when the bug started happening and we are not sure what results are based on this bad data. The preprocessing code had grown so large and unwieldy that it was hard to pin down at what point in the preprocessing the problem was happening. The quickest option was to

completely rewrite the majority of the preprocessing code. We took an array first approach which means we used numpy vectorized operations wherever and whenever possible. This idea caused the time taken to preprocess the data to become significantly better than its pure Python counterpart while also being easier to debug. Additionally the preprocessing code became much shorter when speaking of a line by line comparison.

7 Results

7.1 Dominant Classifications for Single and Multi Beam Studies

In this subsection we will be using the most recent classification system introduced in [3]. For completeness we will restate the general ideas of the system. Let the data structure for the input record be as seen in Section 3.1. Given an event whose interactions are marked as 1, 2, and 3, we reorder them or decouple them. When we say a record is a 213 this means that the second interaction should actually be the first interaction, the first should be the second, and the third should be the third. When we say a record is a 314 this means that the third interaction should be the first, the first should be the second, and the second interaction should be decoupled. We use the case of 124 and 214 to mean a double and swapped double respectively. We use the special case of 412 and 421 to mean a double to triple where the third interaction should be decoupled but the classification is different than a true double. By extension is a 213 is classified as a 412 this means that we have improperly decoupled the third interaction and also computed the incorrect ordering of the first two interactions.

Figure 7.1 is the training and accuracy plot for the network detailed in Section 5.2 trained on 150MeV data for 4096 epochs. The training accuracy continually increases as the epochs elapse with no noticeable dip in accuracy. There is small decrease in the rate of increase around 500 epochs but this is most likely due to a suboptimal dropping of neurons during the training phase. The validation accuracy shows some abnormal trends compared to the training accuracy. Only some neurons are available when training the network but all neurons are available during the validation

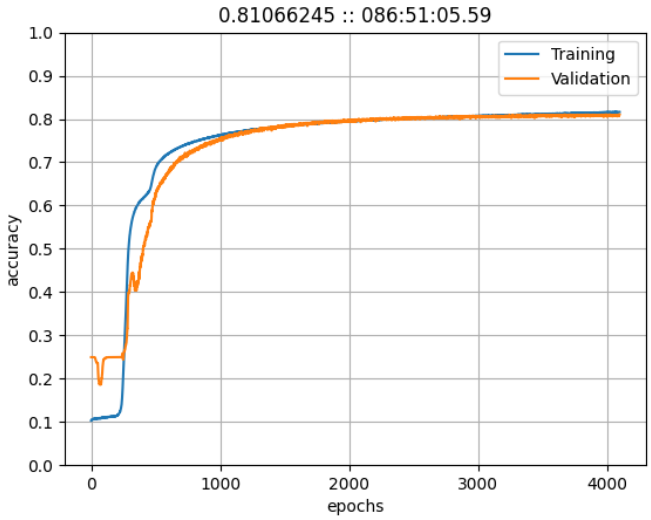


Figure 7.1: The validation and training accuracy for the 256 layer network with 1024^2 neurons per layer trained for 4096 epochs using 150MeV beam data.

Table 7.1: Confusion matrix for a 150MeV trained network classifying 150MeV beam data normalized using a 150MeV normalizer. The leftmost column is the correct input class and the percent in the each proceeding column represents the amount of data put into the class at the top.

	123	**132**	**213**	**231**	**312**	**321**	**124**	**214**	**412**	**421**	**134**	**314**	**234**	**324**	**444**
123	69.3	4.9	2.5	3.5	2.1	7.1	0.0	0.0	1.0	1.3	0.2	0.3	4.4	2.4	1.0
132	6.7	62.8	3.2	5.6	6.3	3.8	0.0	0.0	0.4	0.4	1.5	1.5	3.1	3.7	1.1
213	3.6	2.2	68.2	7.4	6.6	3.6	0.0	0.0	0.8	1.4	2.2	1.8	0.5	0.3	1.4
231	2.9	2.4	5.1	73.3	3.2	6.3	0.0	0.0	1.3	1.2	1.0	1.4	0.4	0.3	1.1
312	3.5	4.8	6.3	3.6	65.8	5.0	0.0	0.0	0.3	0.5	1.5	2.6	2.9	2.1	1.1
321	5.5	2.5	2.3	6.3	3.0	72.0	0.0	0.0	1.0	1.9	0.2	0.3	2.1	2.1	0.9
124	0.0	0.0	0.0	0.0	0.0	0.0	73.8	20.0	0.0	0.0	0.0	0.0	0.0	0.0	6.3
214	0.0	0.0	0.0	0.0	0.0	0.0	17.2	76.5	0.0	0.0	0.0	0.0	0.0	0.0	6.2
412	0.4	0.3	0.2	0.8	0.2	0.5	0.0	0.0	93.9	0.8	0.0	0.1	0.0	0.0	2.8
421	0.2	0.2	0.3	0.4	0.3	0.7	0.0	0.0	0.0	94.2	0.3	0.0	0.0	0.0	3.4
134	0.5	0.2	0.5	0.4	0.3	0.3	0.0	0.0	0.0	0.3	89.6	4.1	0.0	0.0	3.8
314	0.2	0.3	0.5	0.7	0.5	0.6	0.0	0.0	0.0	0.1	2.3	90.7	0.3	0.0	3.7
234	1.2	0.5	0.5	0.3	0.7	0.7	0.0	0.0	0.0	0.0	0.0	0.3	89.4	2.1	4.4
324	1.0	0.9	0.3	0.9	0.6	1.2	0.0	0.0	0.0	0.0	0.0	0.0	2.2	88.5	4.3
444	0.3	0.2	0.3	0.5	0.2	0.5	2.9	3.1	2.0	2.1	1.8	1.9	2.1	1.9	80.3

process. The large dips in accuracy are most likely related to suboptimal neuron configurations when all the available neurons are used. Around 700 epochs validation accuracy starts to become jittery which can be seen in as a thicker line when compared to training accuracy. The jittering behavior does not improve but it also does not get worse as the epochs progress. The validation accuracy peaks at 81% near the end of the training process. The network that produced Figure 7.1 also produced the results seen in Tables 7.1, 7.2, and 7.3.

Table 7.1 is the confusion matrix for the network detailed in Section 5.2 trained on a 150MeV beam classifying 150MeV beam data which was normalized with a 150MeV normalizer (150MeV/150MeV). The leftmost column is the correct input class and the percent in each proceeding column represents the amount of data put into the class at the top of the column. The darkest entry in each row is the dominant classification of the input class. The dominant classification for each input class is the class itself. The dominant classification for the triples has a large lead over all of the other potential classifications and averages 70% for all cases of triples. Doubles have a high average classification accuracy of 75% but still fail to break into our 90% accuracy threshold. The double to triples have an average classification accuracy of 90% with less than 5% being mistaken as true triples and less than 5% being classified as false data. With the false data being correctly classified with an 80% accuracy we are successfully trashing the bulk of our bad data. Everything comes together to show that we are recovering a lot of data via reordering and also scrapping a lot of bad data. The network proposed in Section 5.2 is confirmed to classify at least as accurately as the network used in [3].

Table 7.2 is the confusion matrix for the network detailed in Section 5.2 when trained on a 150MeV beam classifying 70MeV beam data which was normalized with a 150MeV normalizer (70MeV/150MeV). The leftmost column is the correct input class and the percent in each proceeding column represents the amount of data put into the class at the top of the column. The darkest entry in each row is the dominant classification of the input class. The dominant classification for each input class is the class itself except for some double to triple orderings. The classification accuracy for triples have an average accuracy of 67% with data misordering being the primary problem. The classification accuracy for double events is about 75%. The dominant classification for 412, 421, and 134 is not the class themselves but actually some form of triple. The dominant classification for the 314 class is itself and the percent classification is a meager 28%. The second highest classification percent for 314 is 25% for 231.

For triples the 213 case has considerably higher accuracy than the 150MeV classification despite no change to the network or normalization process. The double classification accuracy for the 70MeV/150MeV case is the same as the 150MeV/150MeV case. The double to triple classification

Table 7.2: Confusion matrix for a 150MeV trained network classifying 70MeV beam data normalized using a 150MeV normalizer. The leftmost column is the correct input class and the percent in the each proceeding column represents the amount of data put into the class at the top of the column.

	123	**132**	**213**	**231**	**312**	**321**	**124**	**214**	**412**	**421**	**134**	**314**	**234**	**324**	**444**
123	71.5	5.2	1.9	4.8	2.4	10.5	0.0	0.0	0.0	0.0	0.0	0.2	2.8	0.8	0.0
132	8.0	62.7	3.2	9.4	7.0	5.4	0.0	0.0	0.0	0.0	0.0	0.3	1.6	2.1	0.3
213	3.9	2.5	66.5	13.7	7.5	5.1	0.0	0.0	0.0	0.0	0.0	0.3	0.3	0.0	0.2
231	2.3	1.9	4.0	80.6	2.9	6.8	0.0	0.0	0.0	0.1	0.0	0.2	0.7	0.2	0.4
312	4.5	7.4	6.2	6.2	64.2	6.7	0.0	0.0	0.0	0.0	0.0	0.5	2.2	2.2	0.1
321	8.2	2.7	3.0	9.8	2.0	71.6	0.0	0.0	0.0	0.0	0.0	0.0	1.5	1.3	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	75.3	21.6	0.0	0.0	0.0	0.0	0.0	0.0	3.1
214	0.0	0.0	0.0	0.0	0.0	0.0	20.9	76.1	0.0	0.0	0.0	0.0	0.0	0.0	3.0
412	9.3	1.0	7.1	23.5	0.6	14.0	0.0	0.0	16.9	0.0	0.0	0.0	6.8	0.0	20.8
421	8.5	0.3	9.9	15.4	0.7	22.9	0.0	0.0	0.0	12.1	0.0	9.1	5.6	3.5	12.2
134	1.3	6.3	13.7	22.7	8.7	0.6	0.0	0.0	0.0	0.0	14.8	0.0	6.7	5.3	19.9
314	0.8	5.3	8.6	25.5	13.9	2.9	0.0	0.0	0.0	0.0	0.0	28.0	4.2	2.6	8.2
234	8.1	3.0	1.2	2.1	2.5	8.4	0.0	0.0	0.0	0.0	0.0	0.0	58.5	15.3	1.1
324	4.6	4.7	0.6	3.2	1.2	6.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	77.1	2.1
444	0.6	0.3	0.6	2.0	0.4	1.1	21.4	16.8	0.0	0.0	0.0	0.5	2.2	2.9	51.3

accuracy suffered greatly compared to the 150MeV/150MeV classification accuracy. The majority of the events under 234 and 324 are correctly classified but the percentages are far below the 90% accuracy seen in the 150MeV/150MeV classification. In the cases where double to triple events are misclassified it would be preferred if they were just thrown out entirely. Unfortunately we do not have the ability to throw out double to triple events prior to the classification process because if there was we would not need the network to pick them out among the triples. An optimistic view is that a small number of recovered doubles to triples is a technical improvement even if it is not preferred.

Table 7.3 is the confusion matrix for the network detailed in Section 5.2 when trained on a 150MeV beam classifying 70MeV beam data which was normalized with a 70MeV normalizer (70MeV/70MeV). The leftmost column is the correct input class and the percent in each proceeding column represents the amount of data put into the class at the top of the column. The darkest entry in each row is the dominant classification of the input class. The dominant classification for each input class is not the input class itself for most classes. The majority of all triple classes ended up misclassified under other classes. The only positive take away from triple classification is that the majority of all triples stay under the larger category of triples rather than say double to triples or false events. The classification accuracy of doubles has an average of about 72%. The 412, 421, 134, 314, and 324 have dominant classifications which are not the classes themselves. The classification percentage for 134, 314, 324 is less than 50% with the second highest classification, also not the correct class, being around 30%. For 412 and 421 the dominant classification is incorrect but is at least correctly labeled as a double to triple event. Additionally the dominant classification contains the majority of the events in the class. Only the 234 double to triple event has the correct dominant classification. The dominant classification also contains the majority of the data and the second highest classification was at least a double to triple event.

The double classification accuracy is a little lower than the accuracy for the 70MeV/150MeV and 150MeV/150MeV cases. Despite being a little lower the accuracy for doubles is still comparable to other normalizer scenarios despite all other input classification accuracies being ruined by this new normalization technique. The majority of correctly ordered data was ruined by the misclassifications. There are a multitude of reasons both known and unknown why doubles are comparatively unaffected. The most obvious known reason is that the by feature normalization plays nicer with doubles over other input classes. All doubles have all zeros for the third interaction meaning that non-doubles have their third interaction normalized with a bunch of non-measured zeros causing a skewed mean and standard deviation compared to the first two interactions.

While the false data classification accuracy is comparable to the 70MeV/150MeV scenario at

Table 7.3: Confusion matrix for a 150MeV trained network classifying 70MeV beam data normalized using a 70MeV normalizer. The leftmost column is the correct input class and the percent in the each proceeding column represents the amount of data put into the class at the top of the column.

	123	**132**	**213**	**231**	**312**	**321**	**124**	**214**	**412**	**421**	**134**	**314**	**234**	**324**	**444**
123	7.1	0.2	0.2	27.9	0.1	53.0	0.0	0.0	0.0	0.0	0.0	0.0	11.1	0.5	0.0
132	0.9	1.2	0.5	43.2	0.2	46.0	0.0	0.0	0.0	0.0	0.0	0.2	5.5	2.0	0.5
213	0.6	0.0	3.8	52.7	0.5	32.4	0.0	0.0	0.0	0.0	0.0	0.2	9.8	0.0	0.0
231	0.9	0.0	0.5	61.0	0.0	32.2	0.0	0.0	0.0	0.0	0.0	0.2	5.0	0.0	0.2
312	0.9	0.2	0.5	45.3	1.8	43.8	0.0	0.0	0.0	0.0	0.0	0.2	5.4	1.2	0.7
321	0.7	0.0	0.2	29.4	0.0	62.1	0.0	0.0	0.0	0.0	0.0	0.0	7.2	0.5	0.1
124	0.0	0.0	0.0	0.0	0.0	0.0	68.3	28.1	0.0	0.0	0.0	0.0	0.0	0.0	3.6
214	0.0	0.0	0.0	0.0	0.0	0.0	19.9	76.2	0.0	0.0	0.0	0.0	0.0	0.0	3.8
412	0.0	0.0	0.0	15.8	0.5	7.9	0.0	0.0	0.7	0.0	0.0	0.0	74.2	0.0	0.9
421	0.4	0.4	0.1	9.0	0.1	28.4	0.0	0.0	0.0	0.9	0.0	0.8	57.1	0.4	2.4
134	0.1	0.0	1.0	44.2	0.4	4.3	0.0	0.0	0.0	0.0	0.0	0.0	28.6	1.5	19.9
314	0.1	0.1	0.3	43.8	0.2	13.5	0.0	0.0	0.1	0.0	0.0	6.2	30.9	0.4	4.6
234	0.2	0.0	0.0	3.8	0.0	29.7	0.0	0.0	0.0	0.0	0.0	0.0	61.5	3.6	1.3
324	0.2	0.3	0.6	11.8	0.6	44.5	0.0	0.0	0.0	0.2	0.0	0.5	9.2	29.9	2.4
444	0.0	0.0	0.0	1.2	0.0	1.5	18.3	12.7	0.0	0.0	0.0	0.1	10.8	0.6	54.8

51%, it suffered greatly compared to the 150MeV/150MeV case. The neural network seems to struggle greatly to correctly throw out bad data when not presented with false data from the original beam energy. This does not mean that the network is not generalizing correctly but may instead imply that the false events for the 70MeV are vastly different than those generated by a 150MeV beam. To better gauge the generalizability of the network it must be trained on both beam energies first and then tested on the two data sets separately.

Figure 7.2 is the network detailed in Section 5.2 trained on 150MeV data and 70MeV for 4096 epochs. The data was normalized independently meaning that when normalizing by feature the 150MeV data was normalized separately from the 70MeV data. The hyperparameters of the network are identical to the network which generated Figure 7.1. The training accuracy is very smooth for the entire training process and with no noticeable dips in accuracy. Training accuracy caps at the end of training with a peak accuracy 82%. The validation accuracy starts out much higher than the training accuracy for the first couple hundred of epochs. Around the 250 epoch mark the validation accuracy is overtaken by the training accuracy and remains consistently lower than the training accuracy the remainder of the run. The training accuracy does experience small jittering behavior in the data file but this is normal during the iterative process. However the validation accuracy becomes noticeably jittery around the 300 epoch marks and becomes much worse till around the 1500 epoch mark. Near the 1500 epoch marker the jittering starts to improve until the end where the line is only noticeably more jittery than the training accuracy.

The training accuracy in Figure 7.2 is nearly identical in path, shape, and behavior to the training accuracy in Figure 7.1 but the peak validation accuracy is lower than what is shown in Figure 7.1. The validation line in Figure 7.2 has similar behavior to the validation line in Figure 7.1 until the wiggly behavior starts in the multi beam case. At that point the validation accuracies no longer share similar behavior except that both improve until the end of training.

Attempting to train the network on multiple beam energies does show some success but there are clear problems with generalization compared to training only on a single beam energy. Some new preprocessing methods could be designed to try and more properly blend the data sets to help remove the jittering in validation accuracy and maybe even improve generalizability of the network.

7.2 Training Smaller Networks

Since networks using fewer layers require less memory and are typically faster to train for a given number of epochs, we would like to explore how small our network can be while still performing at a similar accuracy to what is seen in Section 7.1. Figure 7.3 shows the training and validation

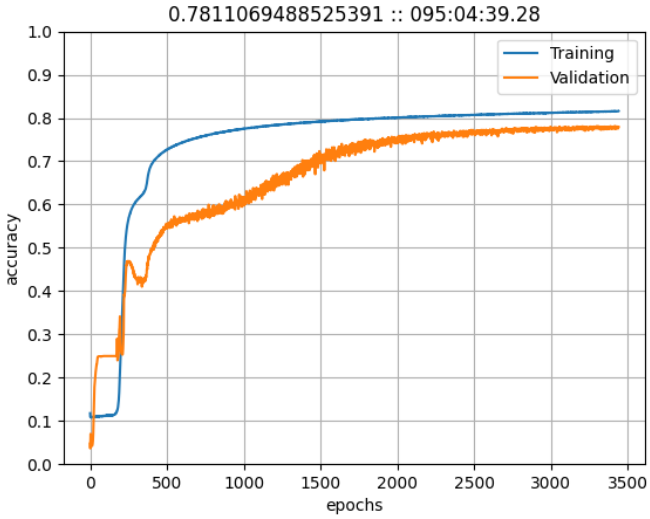


Figure 7.2: The validation and training accuracy for the 256 layer network with 1024^2 neurons per layer trained for 4096 epochs using 150MeV and 70MeV beam data.

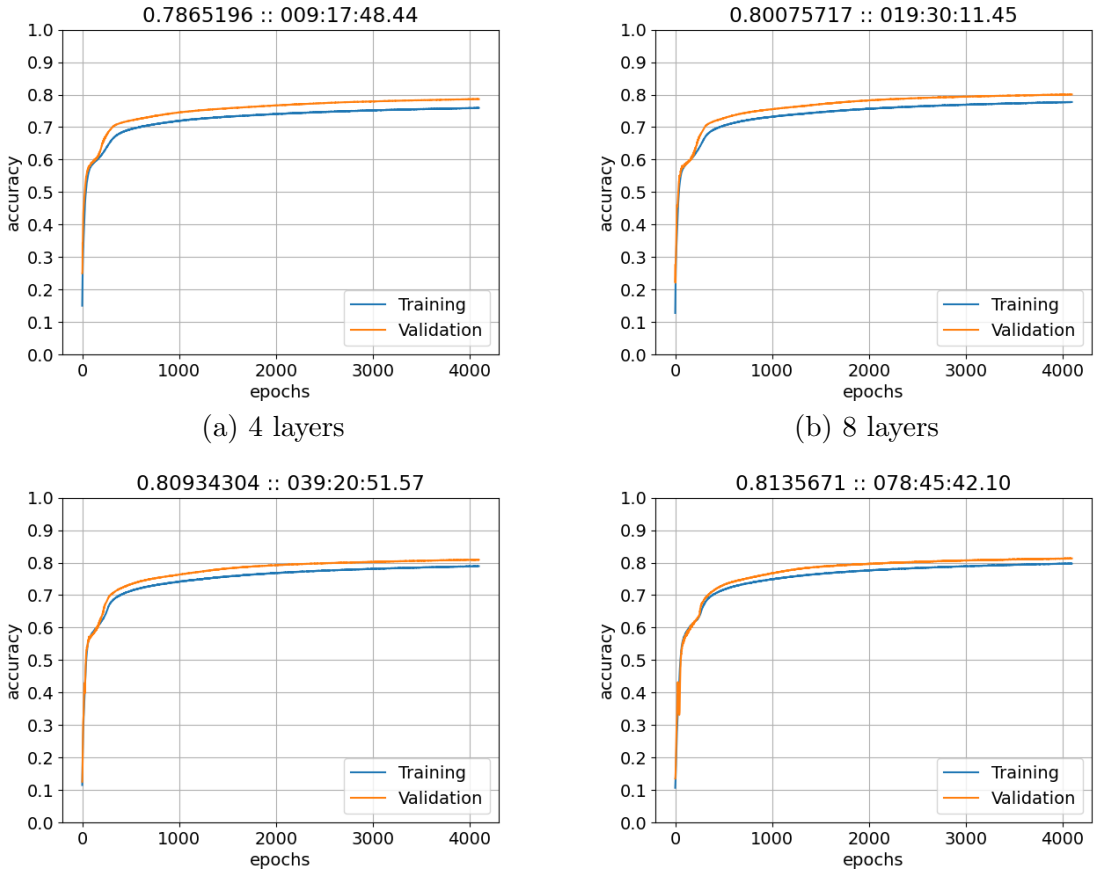
accuracy curves for four different numbers, each employing a different number of layers. These networks are trained for 4096 epochs using a constant learning rate of 5×10^{-6}. Notice that all of the networks achieve a validation accuracy above 78%, with the 8, 16, and 32 layer networks all having an accuracy above 80%. This is comparable to the validation accuracy achieved by the 256 layer network trained in Section 7.1. In the case of the 4 layer network, the accuracy curves have not yet plateaued after 4096 epochs. It is possible that with more training time it too could reach an accuracy above 80%. We therefore conclude that smaller network sizes than 256 layers can be used for further studies.

7.3 Higher Learning Rates

Motivated by the results in Section 7.2, we explore the effects that different learning rates have on the training of smaller networks to determine whether we can train our networks in fewer epochs without sacrificing accuracy.

Figure 7.4 shows the training and validation accuracy curves of four different 8 layer networks trained over 8000 epochs, each with a different learning rate, varying from 1×10^{-5} to 5×10^{-4}. Notice that as learning rate increases, accuracy plateaus earlier. With a learning rate of 1×10^{-5} the accuracy is still increasing after 8000 epochs of training. The other three networks do plateau, with the 5×10^{-4} learning rate network plateauing in well under 1000 epochs. However, the 5×10^{-4} learning rate network also has a lower peak accuracy of 78.65%, as compared to the 79.97% and 79.90%, respectively, peak accuracies of the 1×10^{-4} and 5×10^{-5} learning rate networks. This indicates that the network is stuck at some local minimizer of the loss. We can see from this that 5×10^{-4} is too coarse of a learning rate to achieve maximum accuracy after training.

The corresponding training and validation accuracy curves using 16 layer and 32 layer networks are shown in Figure 7.5 and Figure 7.6, respectively. When layers are kept constant, the change in the validation accuracy curves as learning rate increases shows the same trend as seen in the the 8 layer case: higher learning rates cause the validation accuracy to plateau earlier. However, these networks using higher layer counts begin to show signs of overfitting. This can be seen by



(a) 4 layers

(b) 8 layers

(c) 16 layers

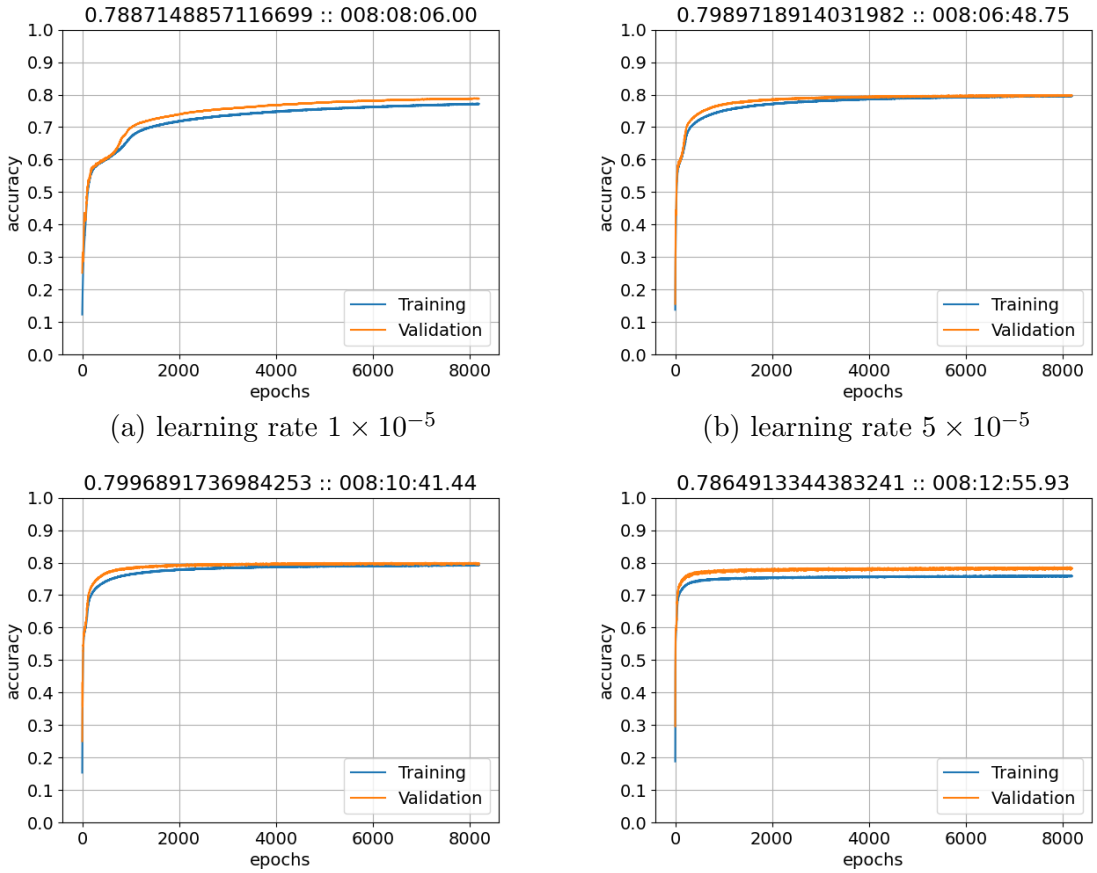
(d) 32 layers

Figure 7.3: Training accuracy and validation accuracy curves for networks of different layer counts with a learning rate of 5×10^{-6}, trained for 4096 epochs.

observing the training accuracy curves. In both Figure 7.5 and Figure 7.6, the training accuracy curves of the 5×10^{-5} and 1×10^{-4} learning rate networks both continue increasing after validation accuracy has peaked. This is particularly prominent in the 32 layer case shown in Figure 7.6. Larger networks have a greater capacity to store information particular to the training dataset that does not generalize to larger datasets, so it makes sense that the these networks with higher layer counts would exhibit overfitting. However, notice that for the 5×10^{-4} learning rate case, for both the 16 layer and 32 layer networks, training accuracy plateaus without showing signs of overfitting. This is because, with such a coarse learning rate, the networks are unable to integrate as much information specific to the training dataset, even though the networks have enough storage capacity to do so.

7.4 Learning Rate Schedules

In Section 7.3 it was observed that, while coarser learning rates can allow a network to be trained to in a fewer number of epochs, if the learning rate is too coarse then the network will plateau at a lower accuracy. One method to combat this is to vary learning rates during training, using a coarse learning rate only at the beginning of training, then using a finer learning rate later on to prevent the network from skipping over local minimizers of the loss. The set of different learning



(a) learning rate 1×10^{-5}

(b) learning rate 5×10^{-5}

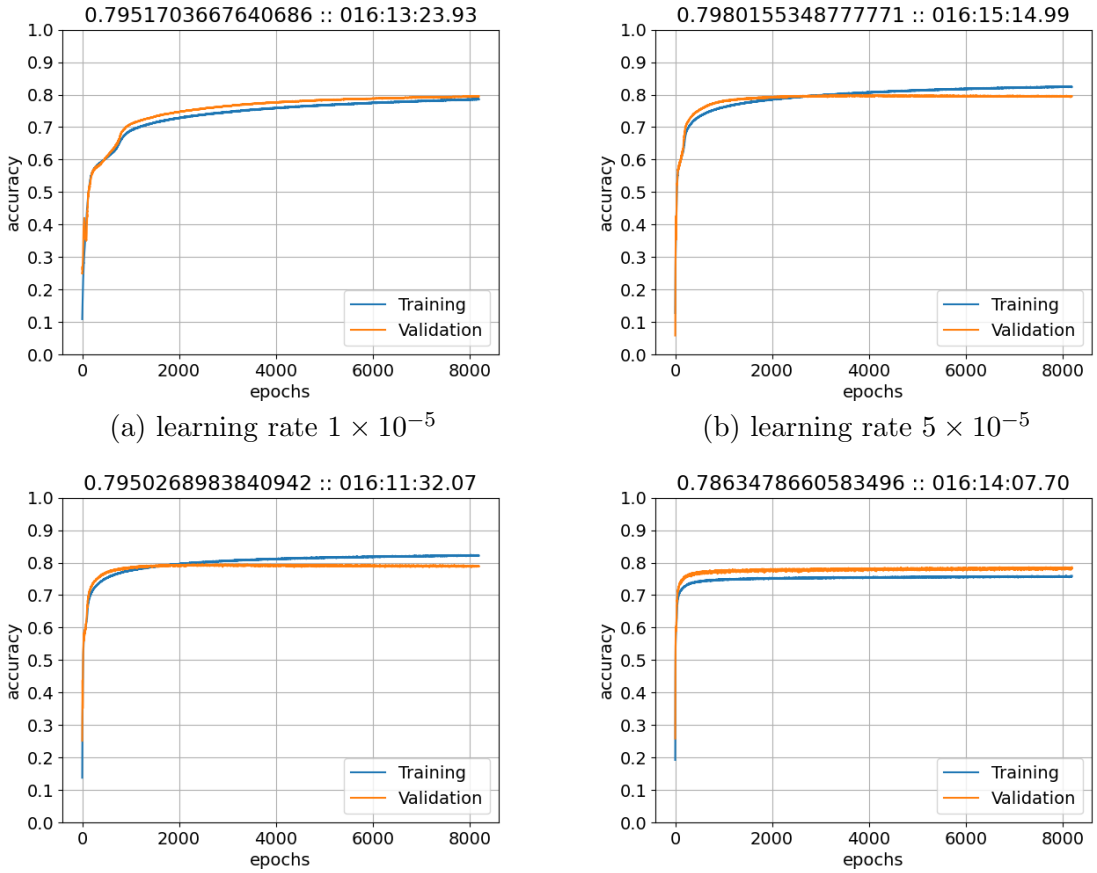
(c) learning rate 1×10^{-4}

(d) learning rate 5×10^{-4}

Figure 7.4: Training accuracy and validation accuracy curves for 8 layer networks, trained with different learning rates for over 8000 epochs.

rates the network employs during training is referred to as a learning rate schedule. It is possible that an appropriate learning rate schedule can allow networks to be trained more quickly to higher accuracies.

The training and validation accuracy curves for a 16, 32, and 64 layer network resulting from the first learning schedule we explore are shown in Figure 7.7. This learning schedule begins with a constant learning rate of 1×10^{-3}, then, after 100 epochs, reduces the learning rate to 5×10^{-4}. After another 100 epochs, the learning is again reduced to 1×10^{-4}, and this pattern is continued until the learning rate reaches 1×10^{-6} at 600 epochs, from which point the learning rate stays constant for the remainder of training. The step wise nature of this learning schedule is especially prominent in the accuracy curves of the 16 and 32 layer networks. Notice that, at first, each time the learning rate is decreased, both training and validation accuracy show a sudden increase. However, before the 500th epoch, changes in learning rate no longer seem to make a difference. Both the 16 and 32 layer networks plateau around the 500th epoch at a validation accuracy of around 78.5%, which is lower than the peak validation accuracies reached by the networks of corresponding size trained with constant learning rates shown in Section 7.3. So, despite the increase in training speed, this particular learning schedule has a negative effect on accuracy. The peak validation accuracy of the 64 layer network is much lower, at only 75.7%. Notice that the step wise nature of the learning schedule is less clear in the 64 layer network's accuracy curves. Unlike the 16 and 32 layer networks,



(a) learning rate 1×10^{-5}

(b) learning rate 5×10^{-5}

(c) learning rate 1×10^{-4}

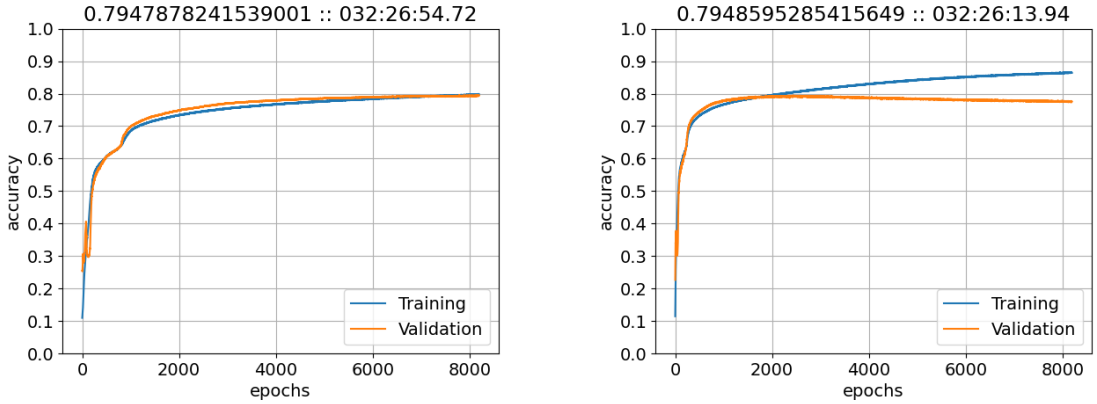
(d) learning rate 5×10^{-4}

Figure 7.5: Training accuracy and validation accuracy curves for 16 layer networks, trained with different learning rates for over 8000 epochs.

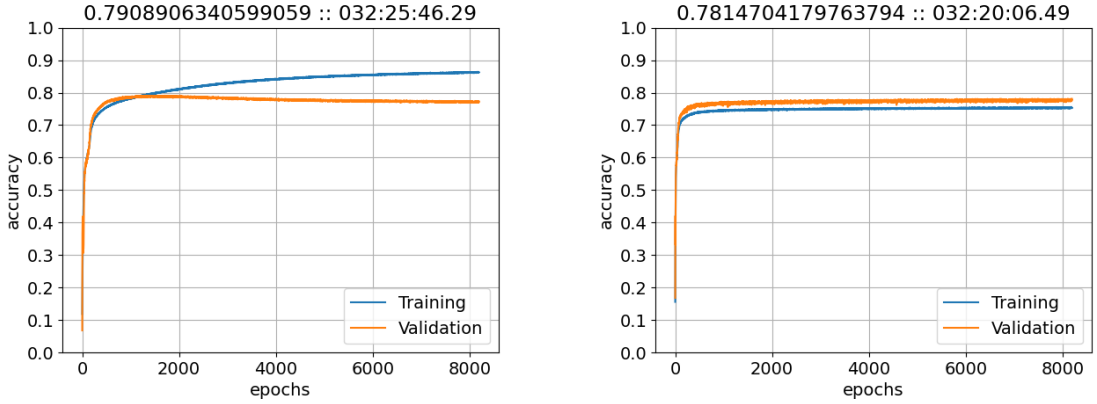
the accuracy never reaches a plateau for each learning rate. This indicates that a learning schedule that decreases the learning rate more slowly might produce better results.

We use a learning schedule that decreases the learning rate in the same pattern as the previous learning schedule, but doing so every 200 epochs rather than every 100 epochs, in order to produce the training and validation accuracy curves shown in Figure 7.8. In this case all three network sizes perform similarly, reaching a peak accuracy slightly over 79%. This is on par with the performance seen in Section 7.3. Notice that for the l6 and 32 layer networks, accuracy takes longer to peak for later steps in the learning rate than it does for earlier steps.

For this reason we test another learning schedule using a variable step size shown in Figure 7.9. Specifically, the learning rate begins at 1×10^{-3}, at 100 epochs decreases to 5×10^{-4}, at 300 epochs decreases to 1×10^{-4}, at 600 epochs decreases to 5×10^{-5}, and continues this pattern until the learning rate reaches 5×10^{-6} at 1500 epochs, remaining constant afterwards. This learning schedule performs similarly to the previous one, however notice that for the 32 layer case the peak validation accuracy reaches 80%, which is higher than what any network used in Section 7.3 achieves. Also notice that for the 64 layer network the accuracy still does not plateau for the early learning rate steps. It is possible that the 64 layer case requires a significantly larger number of epochs for each step of the learning rate in order to achieve its maximum possible accuracy, which may be higher than what is seen in the 32 layer case.

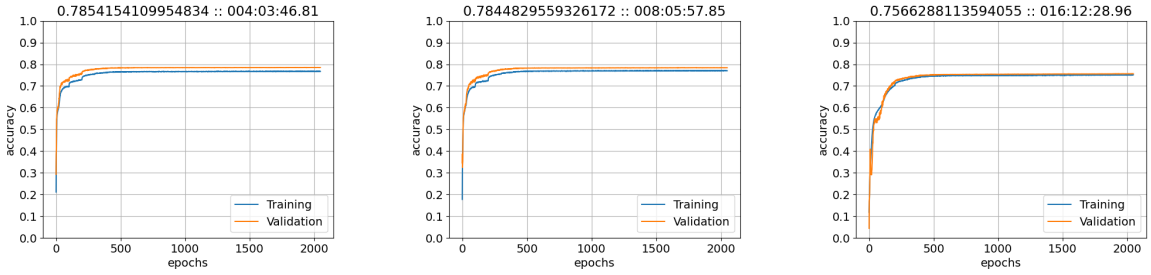


(a) learning rate 1×10^{-5} (b) learning rate 5×10^{-5}



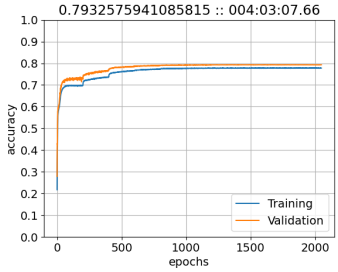
(c) learning rate 1×10^{-4} (d) learning rate 5×10^{-4}

Figure 7.6: Training accuracy and validation accuracy curves for 32 layer networks, trained with different learning rates for over 8000 epochs.

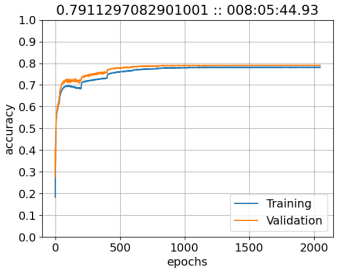


(a) 16 layers (b) 32 layers (c) 64 layers

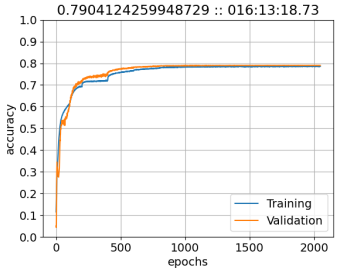
Figure 7.7: Training accuracy and validation accuracy curves for networks of different layer counts, trained for over 2000 epochs, with learning rate starting at 1×10^{-3} and decreasing every 100 epochs until it reaches 1×10^{-6} at 600 epochs, then remaining constant from then onward.



(a) 16 layers

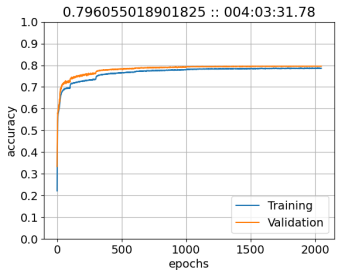


(b) 32 layers

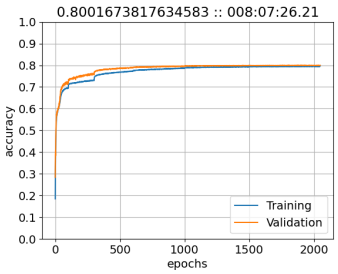


(c) 64 layers

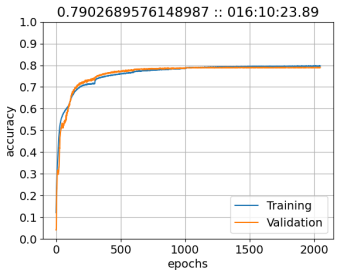
Figure 7.8: Training accuracy and validation accuracy curves for networks of different layer counts, trained for over 2000 epochs, with learning rate starting at 1×10^{-3} and decreasing every 200 epochs until it reaches 1×10^{-6} at 1200 epochs, then remaining constant from then onward.



(a) 16 layers



(b) 32 layers



(c) 64 layers

Figure 7.9: Training accuracy and validation accuracy curves for networks of different layer counts, trained for over 2000 epochs, with learning rate starting at 1×10^{-3} and decreasing with an increasing step size until it reaches 5×10^{-6} at 1500 epochs, then remaining constant from then onward.

7.5 First Attempt at Using a Cheaper Network for Proper Classification

Figure 7.10 is the network detailed in Section 5.4 trained on a 150MeV beam. The training accuracy shoots to 75% in less than 100 epochs. The accuracy then continues to climb until the final epoch capping out with an accuracy of 80%. The validation accuracy follows same pattern but passes training accuracy at around 80 epochs and stays slightly higher than training accuracy until the final epoch capping out at 81%

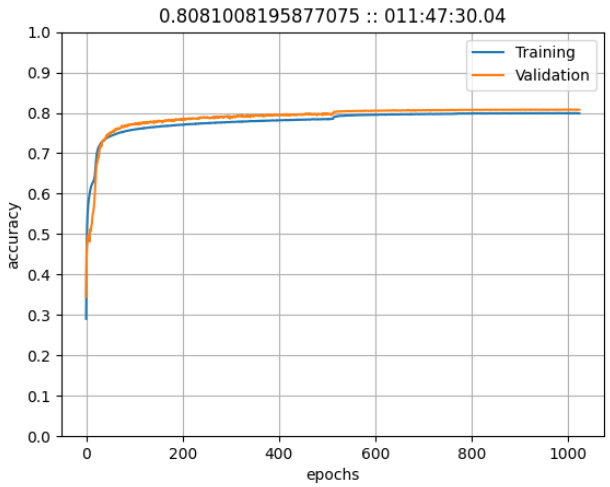


Figure 7.10: The validation and accuracy for the 64 layer network with 1024^2 neurons per layer trained 1024 epochs using 150MeV beam data.

The network itself is the first attempt to create a network with suitable accuracy using the results in Section 7.2 as a basis for improvement. We manage to maintain the same accuracy as seen in Section 7.1 but with about an eighth of the runtime. The improvement in runtime with comparable accuracy shows that a considerably smaller network can adequately classify the data. There is still the question of how the network will perform on multiple beam energies. It is possible that when using additional beam energies we will hit a limit where more beam energies lead to worse prediction because the data is too complex to be learned with only 64 layers. Though considering the number of neurons in our network it is likely this will take more beam energies than we would use for training.

7.6 Alternative Training via Isolated Input Categories

Figure 7.11 is the network detailed in Section 5.2 trained using the generator discussed in Section 5.3 on a 150MeV beam for 1300 epochs. The training accuracy starts out at less than 15% and remains that way for several hundred epochs. At around 500 epochs the training accuracy starts rising at a rapid rate hitting 50% before the 600 epoch marker. The improvement slows and the accuracy is at 60% at near 700 epochs. The training accuracy continues to improve at a much slower rate until the end of the training session with a peak accuracy of 75%. The validation accuracy starts out considerably higher at 25% and does not show any considerable improvements until the 500 epoch marker. At the 500 epoch marker, accuracy steadily increases until the last epoch where it obtains a final peak accuracy of 73%. Once validation accuracy passes 40% the validation accuracy becomes jittery.

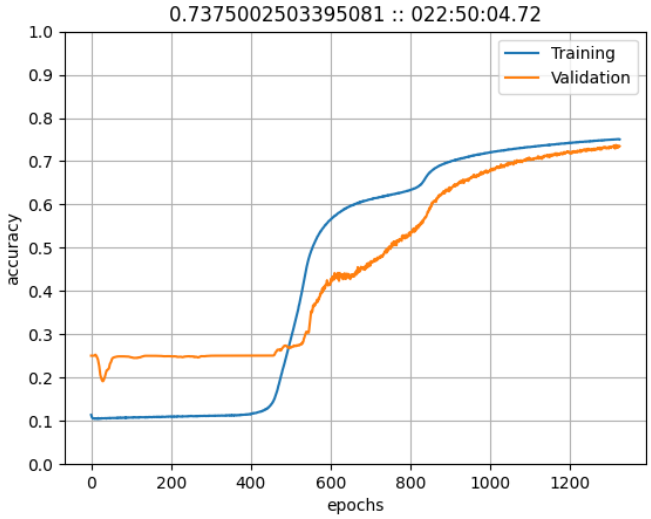


Figure 7.11: The validation and training accuracy for the 256 layer network with 1024^2 neurons per layer train for 1300 epochs using 150MeV beam data and the input category isolation generator.

The use of dropouts provides some explanation as to why learning seemed to stall in the earlier parts of training. Additionally the idea of feeding it input categories can also play a role. The generator is designed such that, as we feed a batch to the network it learns on one specific input category. After we exhaust that input category we feed the next input category. This means that the network may be over compensating when it starts to learn on a new input category. It then overcompensates repeatedly before a single epoch completes. After a sizable amount of the training time has passed, the network has honed in on weights which work for multiple input categories. This is when it starts to improve in accuracy and is making better and less drastic adjustments to the weights. We can say that while a different batch routine was initially not effective it ended up having comparable accuracy at around the same number of epochs. The accuracy around 1400 epochs in Figure 7.1 is around 75% compared to Figure 7.11 finishing at 73%. We can say that different batching routines show promise for at least comparable accuracy while also maintaining the stance that they could lead to improved accuracy.

8 Conclusions

The first step for this work is our conversion from a wide short network to a thin long network as discussed in Section 5.2. We tested this network on a single beam energy and multiple beam energies in in Section 7.1. For the single beam scenario training accuracy continually increases as the epochs elapse with no noticeable dip in accuracy. The validation accuracy shows some abnormal trends compared to the training accuracy. Around 700 epochs the validation line becomes much thicker than the training line which implies a small amount of jitter as the epochs progress. The jittering behavior does not improve but it also does not get worse as the epochs progress. The validation accuracy peaks at 81% which occurs near the end of the training process.

We proceeded to dig deeper into an accuracy breakdown using a confusion matrix on several scenarios. First we looked at how a 150MeV beam normalized with its own normalizer is classified by a network trained on a 150MeV beam. The dominant classification for each input class is the

class itself. We recover a lot of data via reordering and also remove a lot of bad data which would otherwise pollute our reconstruction. At the end we can confirm that the network proposed in Section 5.2 can classify at least as accurately as the network used in [3].

Then we looked at how a 70MeV beam normalized with a 150MeV normalizer is classified by a network trained on a 150MeV beam. The dominant classification for each input class is the class itself except for some double to triple orderings. The classification accuracy for triples have a lower accuracy of around 67% with data misordering being the primary problem. The dominant classifications for 412, 421, and 134 are not the classes themselves but actually some form of triple. In the cases where double to triple events are misclassified it would be preferred if they were just thrown out entirely. Unfortunately we do not have the ability to throw out double to triple events prior to the classification process because if there was we would not need the neural network in the first place. An optimistic view is that a small number of recovered doubles to triples is a technical improvement even if it is not preferred.

Finally we looked at how a 70MeV beam normalized with its own normalizer is classified by a network trained on a 150MeV beam. The dominant classification for each input class is not the input class itself for most classes. The majority of all triple classes ended up misclassified under other classes. The only positive take away from triple classification is that the majority of all triples stay under the larger category of triples rather than say double to triples or false events. For the double to triples we saw that 412 and 421 have an incorrect dominant classification but is still labeled as a double to triple event. Only the 234 double to triple event has the correct dominant classification of all input classes. The false data suffered greatly compared to the 150MeV/150MeV but is comparable to the 70MeV/150MeV with an accuracy of 51%.

Given the ideas in Section 5.4 we experimented with network length and training schedules. The networks trained in Section 7.2 show that it is feasible to use networks employing a relatively small number of layers. This allows for faster training time and more flexibility in terms of what hardware is used. For instance, the 2013 GPU nodes on which these networks were trained have already been surpassed in power by consumer GPUs. Section 7.3 establishes upper bounds for what constant learning rates can be used to achieve accuracy comparable to that in Section 7.1. It is also seen that networks with larger number of layers quickly begin to overfit when higher learning rates are used. The networks in Section 7.4 are trained for a shorter number of epochs, but, through the usage of a learning rate schedule show similar or higher levels of accuracy to what is shown Section 7.3. Learning schedules that decrease the learning rate more rapidly at the beginning of training, then more slowly later on seem to perform better than learning schedules that decrease the learning rate at a constant rate.

We continued to test these ideas in Section 7.5. We put together a network which uses just 64 layers and a simple step schedule that could maintain the same accuracy as seen in Section 7.1 but with about an eighth of the runtime. The improvement in runtime with comparable accuracy shows that a considerably smaller network can adequately classify the data. There is still the question of how the network will perform on multiple beam energies. It is possible that when using additional beam energies we will hit a limit where more beam energies lead to worse prediction because the data is too complex to be learned with only 64 layers. Though considering the number of neurons in our network it is likely this will take more beam energies than we would use for training.

As a side avenue we looked into how to improve the accuracy of the network without changing the configuration. We settled on creating a new training process using the Python generator detailed in Section 5.3 with the corresponding results in Section 7.6. The generator is designed such that, as we feed a batch to the network it learns on one specific input category. After we exhaust that input category we feed the next input category. This means that the network may be over compensating when it starts to learn on a new input category. It then overcompensates

repeatedly before a single epoch completes. The use of different batching routines show promise for at least comparable accuracy while also maintaining the stance that they could lead to improved accuracy.

Acknowledgments

This work is supported by the grant "CyberTraining: DSE: Cross-Training of Researchers in Computing, Applied Mathematics and Atmospheric Sciences using Advanced Cyberinfrastructure Resources" from the National Science Foundation (grant no. OAC–1730250). The research reported in this publication was also supported by the National Institutes of Health National Cancer Institute under award number R01CA187416. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258, CNS–1228778, and OAC–1726023) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See hpcf.umbc.edu for more information on HPCF and the projects using its resources. Co-author Carlos Barajas additionally acknowledges support as HPCF RA.

References

[1] Fernando X. Avila-Soto, Alec N. Beri, Eric Valenzuela, Abenezer Wudenhe, Ari Rapkin Blenkhorn, Jonathan S. Graf, Samuel Khuvis, Matthias K. Gobbert, and Jerimy Polf. Parallelization for fast image reconstruction using the stochastic origin ensemble method for proton beam therapy. Technical Report HPCF–2015–27, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015.

[2] Carlos A. Barajas. *An Approach to Tuning Hyperparameters in Parallel: A Performance Study Using Climate Data*. M.S. Thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County, 2019.

[3] Jonathan N. Basalyga, Gerson C. Kroiz, Carlos A. Barajas, Matthias K. Gobbert, Paul Maggi, and Jerimy Polf. Use of deep learning to classify Compton camera based prompt gamma imaging for proton radiotherapy. Technical Report HPCF–2020–14, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2020.

[4] Vladimir Bok and Jakub Langr. *GANs In Action*. Manning, 2019.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

[6] François Chollet. *Deep Learning with Python*. Manning, 2018.

[7] James Della-Giustina, Carlos Barajas, Matthias K. Gobbert, Dennis S. Mackin, and Jerimy Polf. Hybrid MPI+OpenMP parallelization of image reconstruction in proton beam therapy on multi-core and many-core processors. In *Proceedings of the Symposium on High Performance Computing*, HPC '18, pages 1–11. Society for Computer Simulation International (SCS), 2018. article 11.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

[9] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.

[10] Paul Maggi, Stephen W. Peterson, Rajesh Panthi, Dennis S. Mackin, Hao Yang, Zhong He, Sam Beddar, and Jerimy Polf. Computational model for detector timing effects in Compton-camera based prompt-gamma imaging for proton radiotherapy. *Phys. Med. Biol.*, online April 22, 2020.

[11] Jerimy C. Polf and Katia Parodi. Imaging particle beams for cancer treatment. *Phys. Today*, 68(10):28–33, 2015.

[12] Seth Weidman. *Deep Learning from Scratch*. O'Reilly Media, Inc., 2019.

[13] Robert R. Wilson. Radiological use of fast protons. *Radiology*, 47(5):487–491, 1946.