Deep Learning Based Classification Methods of Compton Camera Based Prompt Gamma Imaging for Proton Radiotherapy

Carlos A. Barajas[1], Gerson C. Kroiz[1],
Matthias K. Gobbert[1], and Jerimy C. 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–2021–1, hpcf.umbc.edu > Publications

Abstract

Proton beam radiotherapy is a method of cancer treatment that uses proton beams to irradiate cancerous tissue, while simultaneously sparing doses to healthy tissue. In order to optimize radiation doses to the tumor and ensure that healthy tissue is spared, many researchers have suggested verifying the treatment delivery through the use of real-time imaging. One promising method of real-time imaging is the use of a Compton camera, which can image prompt gamma rays that are emitted along the beam's path through the patient. However, because of limitations in the Compton camera's ability to detect prompt gammas, the reconstructed images are often noisy and unusable for verifying proton treatment delivery. Machine learning is able to automatically learn patterns that exist in numerical data, making it a promising method to analyze Compton camera data for the purpose of reducing noise in the reconstructed images. First, we provide motivation for training deep neural networks over standard ensemble techniques. We then present the usage of supervised deep neural networks to detect and exploit these patterns so that we can remove and correct the various problems that exist within our data.

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 [30]. 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 surrounding healthy tissues [25]. Due to uncertainties in the range of the beam, relative to important organs in the body, it is difficult to make optimal use of the Bragg peak during treatment.

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 Compton camera, it is possible to reconstruct their origin. It has been suggested that the range of the proton beam in the patient could be verified by using a Compton camera to image the prompt gamma rays emitted during proton treatment delivery. There are a couple hurdles in achieving this goal. The Compton camera does not explicitly record the sequential order of the prompt gammas that interact with the camera's internals twice (a double scatter) or three times (a triple scatter). In addition, it often records false events, which mislabel scatterings of separate, distinct, prompt gamma rays as originating from a single gamma. These problems make reconstructions based on Compton camera data noisy and unusable for practical purposes [25,26,32].

Neural networks, in general, represent repeated non-linear data transformations which map an input record to an expected outcome. Shallow networks like the ones seen in [32] and [21] used

1 layer and 2 layer neural networks respectively to perform less intensive classifications based on simpler prompt gamma simulation data. Instead of using shallow networks we approach these problems by leveraging several different deep fully connected neural networks which consist of well over 100 layers each.

We justify our usage of deep learning by demonstrating that traditional ensemble methods, like random forests, do not have the ability to generalize on our data. We also show that the classification accuracy of random forests pale in comparison to our neural networks' ability to perform the same task. We provide an in-depth analysis as to why the data we use in this work is different from our previous work in [6]. We also implement a fully connected residual block to prevent back propagation stagnation typically associated with deep neural networks. For network design and creation we use a network architecture generator, written in Python3 which uses Keras and Tensorflow, to build our deep fully connected neural networks. We use this generator in this work and in [18] to make networks which specialize in determining whether a given double is correctly ordered or is actually falsely coupled singles misreported as a double by the Compton camera. We train our doubles network on 2.2M events with 10 features per record. We use the mentioned generator in this work and in [3] to make networks which specialize in determining whether a given triple is: a true triple which has one of six possible orderings, a "double-to-triple" where a single is incorrectly attached to a true double of arbitrary ordering yielding six possible cases, a "false triple" which is three singles misreported as a true triple by the Compton camera. We attempt to classify a given triple into one of these 13 classes. In [3] we specifically focus on the predictions of doubles-to-triples. The jump from classifying 12 classes to 13 classes is surprisingly difficult and requires significantly more configuration and training time because false triples are so similar to doubles-to-triples which themselves are similar to triples. We refer to the triples/doubles-to-triples/false triples data configuration as triples only. We train our triples only networks on 1.8M events. We test all of our networks on 20 different MCDE simulation data sets which consist of varying dose rates.

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 could be used to overcome the limitations of proton beam therapy but the presence of false events and misordered interactions in recorded data prevents practical usage. Section 4 gives a brief overview of machine learning ensemble methods, properties and operation of neural networks in the context of deep learning, and how neural networks have been used in similar works. Section 5 details how the data has to be handled, changed, and labelled for deep learning viability. Section 6 catalogues the hardware and software used for all research activities. Section 7.3 details the performance of random forests used for prompt gamma classification. Section 7.2 through Section 7.4 describes how all variations of the networks performed on different proton beam sets. Section 8 presents our conclusions from this work.

2 Proton Beam Therapy

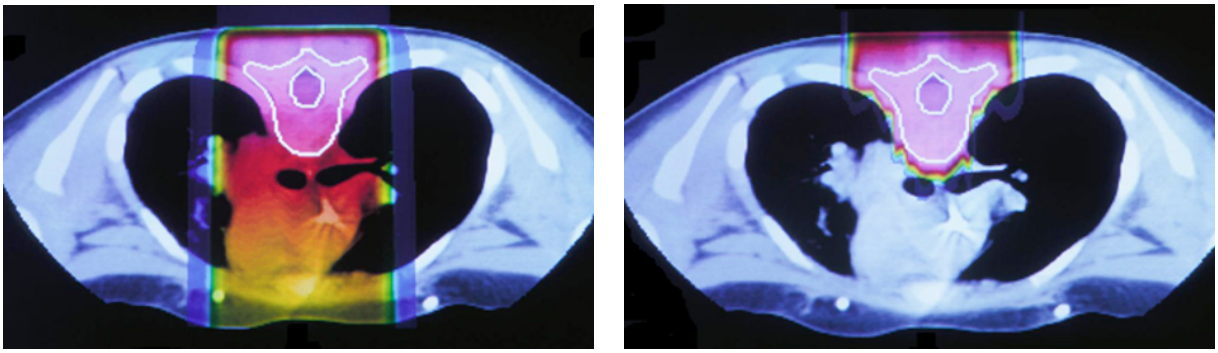
Proton beam therapy was first proposed as a cancer treatment in [30]. To a first order approximation, the radiation dose emitted by a proton beam is inversely proportional to the kinetic energy of the particles within the beam. The beam's particles lose kinetic energy as they traverse the patient, the 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 dose rapidly reaches its maximum [7]. This point of maximum dose is called the Bragg peak and the discovery and additional details associated with it can be seen in [7]. One of the most important things about the Bragg peak is

that 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 areas solely affected by cancerous tumors allowing vital organs beyond the tumor to be spared [1, 12, 25].

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, which contains a tumor that should be irradiated. Since the heart, which is at the bottom center, is still healthy, one should avoid delivering radiation to it. 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 doses of radiation to be delivered to cancerous tissues with minimal damage to healthy tissues. This can lead to better patient outcomes [25].

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. 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. 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 [25].

Figure 2.2 compares two possible beam trajectories through a cross-section of the chest [25]. In this case the heart, outlined in purple, is positioned at the top-center of the figure and a tumor, outlined in green is located next to it. The optimal trajectory can be seen in the left image and uses a single beam, which is represented as the space between the dashed white lines, to fully irradiate the tumor before hitting the heart. 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 strip partially overlaps the heart which means that there is a possibility that the heart could be irradiated. In practice professionals opt for the trajectory in the right image which uses two beams instead. This new trajectory is considered suboptimal because it delivers a small dose of radiation to the lungs [25].



(a) (b)

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

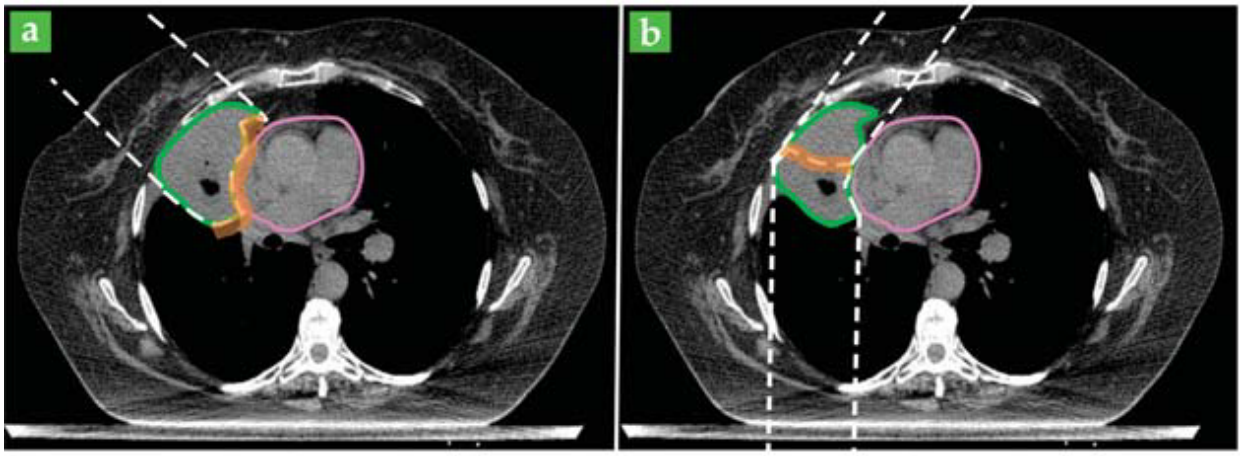


Figure 2.2: (a) Optimal proton beam trajectory. (b) Suboptimal trajectory necessary to protect heart.

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 [13, 25, 26]. 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. A Compton camera is a multi-stage detector that uses the principles of Compton scattering, detailed in [11], to produce 2D and 3D images of gamma ray and x-ray sources [27, 28].

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 and some of them enter 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 (also called a 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 [20].

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 [20]:

- Reconstruction methods typically require that all interactions in an event be correctly 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. 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 as a single event. This results in readouts that do not correlate to any actual physical event which we refer to as bad 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 bad events. Also, prompt gamma rays are more likely to scatter at higher energy levels, leading to more multi-scatter events, which, as explained above may be misordered in the output data. These two effects greatly diminish the accuracy of Compton camera reconstructions at high energy levels, making them unusable [20].

3.2 The Definition of Event Types

Multi-scatter events can be classified into five categories: true triples, doubles-to-triples, doubles, false triples, and false doubles. 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 and should also be removed before reconstruction. 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 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. Those originating from the same prompt gamma ray are 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. 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 matrix represents an entire event, while each row represents one 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 contains only two rows rather than three because double events only contain two interactions.

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 Euclidean distances $\delta r_{i,j}$ where $i, j = 1, 2, 3$ and δ_{xy} is the Euclidean distance between interaction x and interaction y. 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.

4 Machine Learning

4.1 Ensemble Methods

Work in [6], [5], and [4] show initial success in multi-class classification using various fully connected neural networks. In conjunction with the previous technical reports, this report studies

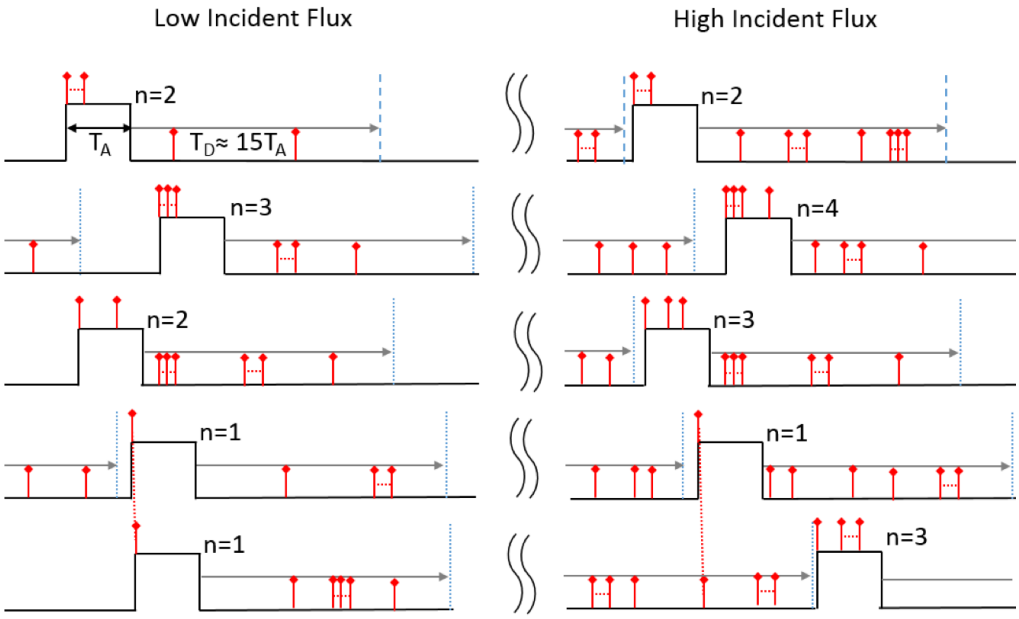


Figure 3.1: An illustration of events.

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	$\delta r_{1,2}$
e_2	x_2	y_2	z_2	$\delta r_{2,3}$
e_3	x_3	y_3	z_3	$\delta r_{3,1}$

(b)

Figure 3.2: (a) The initial input format representing a single event. (b) The appended input format including distances.

random forest models to hopefully provide motivation as to whether further work into deep learning techniques has greater potential to improve testing accuracy rather than standard classification techniques such as logistic regression, naive bayes, k-nearest neighbours, decision trees, and support vector machine. Each of these classification strategies' accuracies depend on the characteristics of the data as well as hyperparameters of the respective strategy. It is common for may of these classification models to produce similar results on the same data sets. As such, results from the random forest studies can provide initial motivation for further studying the other standard classification techniques.

An important conclusion in [32] was that naive Bayes showed great promise in classifying true triples but was significantly slower than neural networks despite yielding better accuracies. They note that it took several days to get a result but the ordering task requires a performance time of a couple minutes at most for usage in a real-time setting.

Random forests are a commonly used ensemble method that average the results from decision trees for classification, regression, and other forms of machine learning. The studies in this report use random forest models trained for classification accuracy. Within the random forest, each individual decision tree in itself is another form of machine learning classification, where based on characteristics of a sample, the sample is classified. Typically, random forests outperform single decision trees as they can average the results of many decision trees. The averaging of the individual decision trees directly tackles any overfitting that occurs when training individual decision trees.

A more in-depth explanation of the ensemble method can be described in [8]. For the studies in this report, the random forests models are from `scikit-learn 0.23.dev0` (`sklearn`). All hyperparameter studies use hyperparameters defined in the `sklearn` library.

With the random forest classifier from `sklearn` we studied a randomized search on many of the possible hyperparameters. The results to both studies are shown in Section 7.3.

4.2 Neural Networks

4.2.1 Deep Learning

We propose to train a neural network to process the data output by the Compton camera.

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 [10]. In the case of the data output by the Compton camera, we would like the neural network to read a multi-scatter event and determine which interactions originate from the same prompt gamma ray, and what the correct order of these interactions are.

Figure 4.2 shows the training and testing process for a neural network using supervised learning [2]. Supervised learning refers to training the neural network using labels for the data. These labels provide a clear cut answer as to what the the output data should look like when the hidden layers have finished their work. 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 neural network generalizes well and can be used on additional unseen 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 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

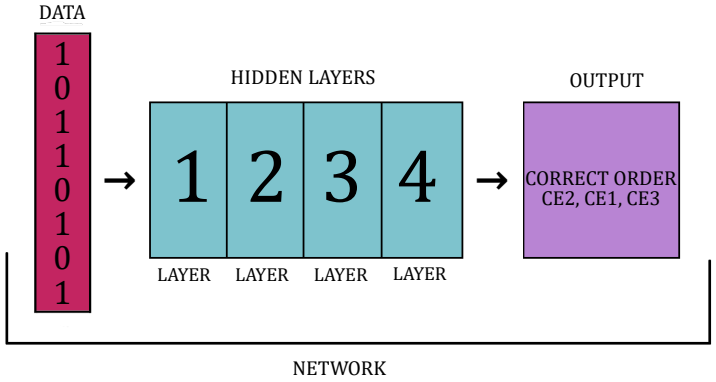


Figure 4.1: The structure of a fully connected neural network.

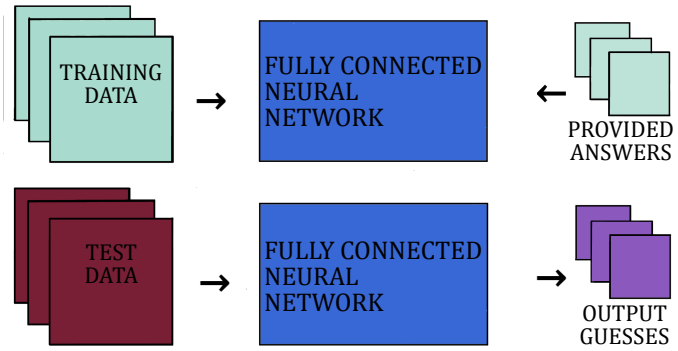
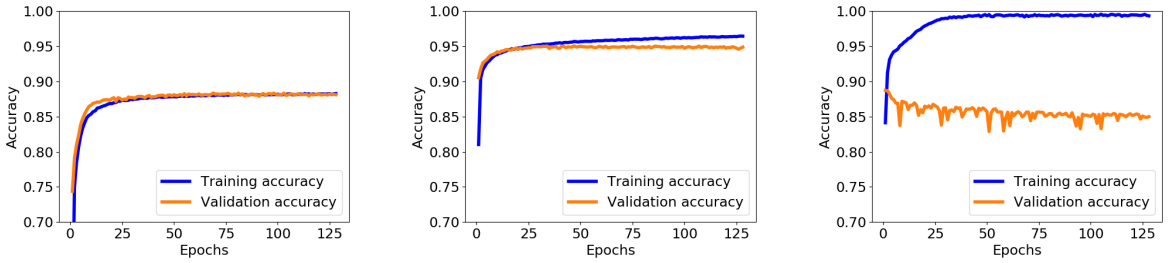


Figure 4.2: The training and testing process for a fully connected neural network.

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. Figure 4.3 compares the accuracy curves of three networks which originate from a textbook example as they are being trained [10]. 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. When there is still information in the data that has not been incorporated into the network's model this is referred to as underfitting which can be seen in the left plot. 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. This occurs because the network has either not been trained enough, or because the network size is too small to fully capture all patterns within the data. When the network performs very well on the data it has been trained on but does not perform well on data that it has not been trained on, this is called overfitting which can be seen in the right plot. Here there is a large gap between training accuracy and validation accuracy, indicating a lack of generalization. Overfitting occurs because the network has begun directly mapping inputs to outputs which is equivalent to "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 [10]. 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.

Neural networks have shown promise in the handling of Compton camera data in other works. In Zoglauer et al. [32] it was shown that a 1 layer fully connected neural network can be fed raw camera data and some computed values to determine the correct ordering for true triples. The authors conclude that their neural network performs competitively to classical sequencing techniques. In that same work the authors attempted to use a Bayesian approach but it took several days to run and cannot be considered a reasonable replacement to the classical or neural network approaches. In Muñoz et al. [21] they use a 2 layer fully connected neural network with a binary output layer to determine if any given event should be used for reconstruction. The authors were able to successfully



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

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

decrease the amount of bad data being used for reconstruction and increase the amount of good data used for reconstruction. They note a relative increase of 35% for the good to bad data ratio. The emphasis of our work is both in the complexity of the network and the classification power it provides. The "deep" in deep learning stems from the fact that a neural network with many layers is considered deep and networks with very few layers are considered to be shallow. Both [32] and [21] use very shallow networks, 1 layer and 2 layers respectively, which were capable of classifying much simpler scenarios. Our proposed networks used in Section 7 contain over 100 layers for any given network and correct more noise causing scenarios than the other works through several different methods.

4.2.2 Activation Functions

Activation functions are one of the core parts of neural networks [10]. 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 directly instead of solving for all 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 [29]. The struggle we have now is choosing a non-linear function to use.

In [6] we used a Scaled Exponential Linear Unit (SeLU) proposed in [17] and stated as

$$s(x) = \begin{cases} \lambda x & x > 0, \\ \lambda\alpha e^x - \lambda\alpha & x \leq 0. \end{cases} \tag{4.1}$$

The constants α and λ could be treated as hyperparameters but [17] 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 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 the Rectified Linear Unit (ReLU) or Leaky ReLU. ReLU, otherwise known as

$$r(x) = \begin{cases} x & x > 0, \\ 0 & x \leq 0, \end{cases} \tag{4.2}$$

is one of the more commonly used activators and is discussed in [10]. 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 [19]. The neurons gradually become 0 as the network feeds forward. This makes ReLU incompatible with our desire to create a very deep network. Instead we opt for Leaky ReLU

$$l(x) = \begin{cases} x & x > 0, \\ \beta x & x \leq 0, \end{cases} \tag{4.3}$$

which has single multiplication with a small positive constant β. We consider β to be a hyper-parameter which can be tuned but we decided to use Keras' default value of 0.3. This non-zero β fixes the dying forward propagation seen with ReLU [15]. With Leaky ReLU we get a much cheaper activator which uses scalar multiplication instead of scalar multiplication, exponentiation, and subtraction combined.

Any neural network which uses a one-hot binary multi-class output uses the Softmax function as the activation function on the output layer. Given some n by 1 vector x we define Softmax as

$$s(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \forall i \in \{1, \ldots, n\}. \tag{4.4}$$

When it is used on the output layer of a neural network it produces a probability for each output node that can be interpreted as the network's confidence that a given input maps to the respective node [9]. We use the Softmax function for the output layer on all networks in Section 7.

4.2.3 Fully Connected Residual Blocks

The network used in Section 7 is a deep fully connected neural network. Neural networks, especially fully connected ones, break down once they start becoming notably deep and complex. One of the first problems is that the values start to become very small during the forward propagation process. This leads to zeros and like-zero values becoming more prominent as you go deeper and deeper. A partial fix to this forward propagation issue is to use Leaky ReLU over the traditional ReLU. The second problem occurs during back propagation. During back propagation we start to see the gradient becoming like-zero causing little to no update to existing weights which causes learning stagnation. This phenomenon is discussed more intimately in [14] where they detail these effects. The major breakthrough solution to this problem is also proposed in [14] where they create ResNet, a network built from "residual blocks". We use their original implementation of residual blocks as the conceptual basis for our fully connected residual blocks. A visual representation of the fully

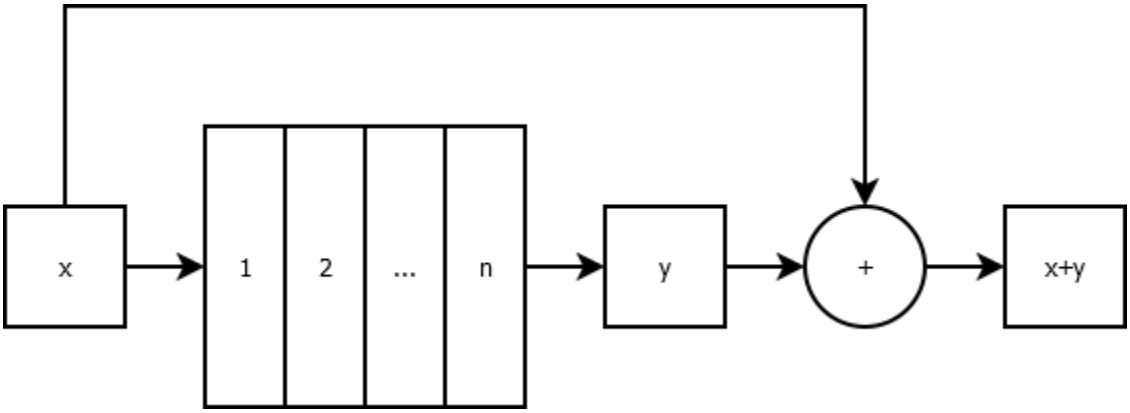


Figure 4.4: Our fully connected residual block takes an input and passes it through n layers eventually adding it to the output of the n layers.

connected residual block can be seen in Figure 4.4. Consider some record x. We pass it as an input to a small group of n layers with their own activators. The result of the layer digestion we can call y. Finally, we concatenate x and y. The concatenation in our case, and the case of the original ResNet, is addition. This addition operation helps push non-zero values through the forward propagation process which helps keep input data to each block fresh and non-zero. This also helps prevent vanishing gradients during the back propagation process.

The residual block method in [14] concatenates the data before the activation function is applied. The second version of ResNet, called ResNetV2, concatenates after the activation function. The original residual blocks were designed using convolutional layers in an image classification network. We create residual blocks using only fully connected layers with post-activation concatenation for the classification of prompt gamma events. The fully connected residual blocks allows us to create a thin yet super deep fully connected neural network which avoids the aforementioned problems.

4.2.4 Learning Rate Schedulers

To talk about learning rate schedulers first we have to mention a small piece about our optimizer. In Section 7 we use the Adam optimizer for all of our neural networks. What Adam is and how it works can be seen in detail in [16]. The only piece of the document we need to consider is

$$\theta_i = \theta_{i-1} - \alpha \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}. \tag{4.5}$$

For clarity we want to point out that θ is the variable which is being updated. In the context of a neural network, θ, could be a neuron or an entire weight matrix depending on the neural network layer type and implementation of the optimizer. This update function uses special step direction, $\hat{m}_i/(\sqrt{\hat{v}_i} + \epsilon)$, which is designed to be adaptive and based on momentum of the bias vector and weight matrix [16]. The update function also uses a constant step length α which is commonly referred to as the learning rate in the context of neural networks.

Here we want to focus on α from Equation 4.5. In Equation 4.5 the α stays constant. With a learning rate scheduler we allow the α to change as the epochs increase. Let L be a function which produces the learning rate to be used at the ith epoch. Keras passes the epoch number and the learning rate from the previous epoch as inputs to L. So long as there is a function which takes these inputs and returns a learning rate to be used at the current epoch then Keras cares very little what else is incorporated to make this happen.

With these parameters we can design a simple step function which tightens the learning rate after a third, two thirds, and five sixths of the total number of epochs have been performed. Let p be the total number of epochs and t_i the learning rate for any given epoch. Then to tighten as described we would use

$$t_i = L(i, t_{i-1}) = \begin{cases} 10^{-3} & i \leq \frac{p}{3} \\ 10^{-4} & \frac{p}{3} < i \leq \frac{2p}{3} \\ 10^{-5} & \frac{p}{3} < i \leq \frac{5p}{6} \\ 10^{-6} & \frac{5p}{6} < i. \end{cases} \tag{4.6}$$

We choose to use this step scheduler in this work because it is simple and showed promise in [6]. The scheduler listed in Equation 4.6 was used to determine the learning rate during training for all networks in Section 7.

5 Preprocessing

5.1 Standardization and Normalization Techniques

In order to give our network a better chance at learning we decided we would focus more heavily on preprocessing and pre-computed values. All methods we used are a part of sklearn's preprocessing library or `sklearn.preprocessing` [22].

The exact nature of our allows us additional assumptions. We take advantage of the fact that each event has 3 interactions for triples and 2 interactions for doubles. All notation in the section will be for the triples only data. For all normalization we reshape, otherwise referred to as "wrap", the data such that we normalize over all x, y and z rather than x_1, x_2, x_3 and so on. From a mathematical standpoint, let $E \in \mathbb{R}^{n \times 3 * f}$ be a matrix containing n events with 3 interactions and f features per interaction. For normalization we reshape E such that it becomes $R \in \mathbb{R}^{3 * n \times f}$ and then normalize by column. This reshaping process can be hard to visualize so instead consider Figure 5.1 where we have $E \in \mathbb{R}^{2 \times 9}$. We know that E has 3 features per interaction and that our new matrix R after reshaping will be $R \in \mathbb{R}^{6 \times 3}$. Now to expand upon this notation let E^p be a submatrix of E where all of the events in E^p belong to module p. For the pth module let E^p_{ij} be value of the jth feature of the ith event of that module. Then by extension R^p_{ij} is the reshaped version of E^p where we are looking at the jth feature of the ith row after E^p has been reshaped. Finally we will say that $\hat{R}^p$ is the normalized version of the module submatrix R^p and $\hat{E}^p$ would be the unwrapped version of $\hat{R}^p$. We use the notation for by module normalization and corrections but we do not use by module arithmetic for the studies in Section 7. Since we are not adjusting the data by module in this work you can assume that all data is contained in a single module for the

$$E = \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{14} & E_{15} & E_{16} & E_{17} & E_{18} & E_{19} \\ E_{21} & E_{22} & E_{23} & E_{24} & E_{25} & E_{26} & E_{27} & E_{28} & E_{29} \end{bmatrix} \overset{\text{reshape}}{\rightarrow} \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{14} & E_{15} & E_{16} \\ E_{17} & E_{18} & E_{19} \\ E_{21} & E_{22} & E_{23} \\ E_{24} & E_{25} & E_{26} \\ E_{27} & E_{28} & E_{29} \end{bmatrix} = R$$

Figure 5.1: A demonstration of the data wrapping process performed by numpy.

purposes of preprocessing. If we were to normalize by module we would still feed all of our data into the machine learning algorithm as a whole NOT by module.

The `MaxAbsScaler` scales and translates by feature such that the maximal absolute value of each feature is between -1 and 1. First we wrap the data into R and start normalizing by module. We know that any element in $\hat{R}^p$ can be written as

$$\hat{R}^p_{ij} = \frac{R^p_{ij}}{\max\limits_{k\in\{1,\ldots,3*n\}} |R^p_{kj}|}. \tag{5.1}$$

This method of normalization was how we adjusted all spatial values with wrapping in Section 7.

The `PowerTransformer` uses the Yeo-Johnson method, detailed in [31], to normalize the data. The idea is that we can use parametric transformations to attempt to map data to a normal distribution. After transforming the data the transformer applies zero-mean, unit variance normalization. We let $\psi(\lambda_j, R^p_{ij})$ represent the power transformation function and note that the value λ_j is computed by sklearn using a maximum likelihood estimation for the jth column of R^p. Now we can compute any element of $\hat{R}^p$ as $\hat{R}^p_{ij}$ with

$$\hat{R}^p_{ij} = \psi(\lambda_j, R^p_{ij}) = \begin{cases} \frac{(R^p_{ij}+1)^{\lambda_j}-1}{\lambda_j} & \text{if } \lambda_j \neq 0, R^p_{ij} \geq 0, \\ \ln(R^p_{ij}+1) & \text{if } \lambda_j = 0, R^p_{ij} \geq 0 \\ -\frac{(-R^p_{ij}+1)^{2-\lambda_j}-1)}{2-\lambda_j} & \text{if } \lambda_j \neq 2, R^p_{ij} < 0, \\ -\ln(-R^p_{ij}+1) & \text{if } \lambda_j = 2, R^p_{ij} < 0. \end{cases} \tag{5.2}$$

The interesting part of this transformer is that it maps to something close to the Gaussian distribution. We used this transformer to normalize our energy deposition values using wrapping in Section 7.

5.2 Events and Their Classes for Deep Learning

For the ease of discussion we will talk about our data as doubles and triples only because these two scenarios have separate methods for data cleaning and classification. This is because the number of elements in the doubles data is less than the number of elements in the triples only data because they have one less interaction. The number of elements in the doubles and false doubles are the same. Our triples only are the true triples, doubles-to-triples, and false triples and they all have the same number of elements. These three categories provide a combined total of 13 classes that are used for classifying our data. Let each interaction be represented by 1, 2, or 3 for any given event. When a triple is correctly ordered we call that a 123 event. When a triple is misordered it is represented as 132, 213, 231, 312, 321. All possible orderings can also be seen in Figure 5.2. To explain the labelling system more, consider the 312 event from Figure 5.2. In the 312 event the data which should be interaction 1, $[e_1, x_1, y_1, z_1]$, shows up as interaction 2. Similarly, $[e_2, x_2, y_2, z_2]$ shows up as interaction 3 but should be interaction 2. Lastly, $[e_3, x_3, y_3, z_3]$ shows up as interaction 1 but should be interaction 3. For doubles-to-triples one of the interactions is actually a single which was incorrectly coupled to a double. When talking about a double-to-triple we still have three interactions but instead use 1, 2, or 4. A 124 event is a correctly ordered double-to-triple where the last interaction is a falsely coupled single. When a double-to-triple is misordered it is represented as 214, 134, 314, 234, 324. For example a 314 means that the $[e_1, x_1, y_1, z_1]$ shows up under the third interaction, $[e_2, x_2, y_2, z_2]$ shows up under the the first interaction, and a falsely coupled single shows up under the second interaction. False interactions are labelled as a 444 event

Class	Interaction 1				Interaction 2				Interaction 3			
123	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2	e_3	x_3	y_3	z_3
132	e_1	x_1	y_1	z_1	e_3	x_3	y_3	z_3	e_2	x_2	y_2	z_2
213	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1	e_3	x_3	y_3	z_3
231	e_2	x_2	y_2	z_2	e_3	x_3	y_3	z_3	e_1	x_1	y_1	z_1
312	e_3	x_3	y_3	z_3	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2
321	e_3	x_3	y_3	z_3	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1
124	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2	single			
214	e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1	single			
134	e_1	x_1	y_1	z_1	single				e_2	x_2	y_2	z_2
314	e_2	x_2	y_2	z_2	single				e_1	x_1	y_1	z_1
234	single				e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2
324	single				e_2	x_2	y_2	z_2	e_1	x_1	y_1	z_1
444	single				single				single			

Figure 5.2: We can see the input class as the left column and the actual interaction data in the proceding columns. The input class directly coordinates to how the energy and spatial data are ordered in the data file for training. For example for a 312 event $[e_1, x_1, y_1, z_1]$ shows up as interaction 3 when it should show up as interaction 1. Similarly, $[e_2, x_2, y_2, z_2]$ shows up as interaction 1 but should be interaction 2. Lastly, $[e_3, x_3, y_3, z_3]$ shows up as interaction 2 but should be interaction 3. For doubles-to-triples one of the interactions is actually a completely a single which was incorrectly coupled to a double.

and all three interactions are actually falsely coupled singles. The arrangement of the data for each class in their respective interactions can be seen in Figure 5.2.

For doubles we only have three classes: 12, 21, and 44. A correctly ordered double is a 12 and a misordered double is a 21. A false double is two falsely joined singles and is labelled 44.

6 Hardware Used

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.

7 Results

The results section of this report is a collection of various studies, tests, networks, configurations, and data samples that were used and done over the last year of our research. In Section 7.1 we show that the old data we used in [5] is not usable for continued studies requiring us to use different data for the reminder of our studies. In Section 7.2 we do a hyperparameter study on the simpler but important doubles data set from our latest data and also show the network's learning capabilities and performance when incorporating false data. In Section 7.3 we do a random forest study on the true triples to show the effectiveness of an ensemble method on our data. Lastly, in Section 7.4 we do multiple studies which gradually add more event types and the classes associated with those events from the triples only data.

Due to the nature of the simulations we must combine events across multiple experimental data sets for all of the data sets used for training and validation. DtoT events do not occur in large quantities, if at all, in lower dose rates as is explained in [24]. True triples occur most frequently at lower dose rates and less frequently at higher dose rates. We use a 150MeV proton beam with a 0kMU dose rate for the true triples and a 150MeV beam with a dose rate of 100kMU for the doubles-to-triples. It is important to remember that the dose rate should only affect how well the Compton camera detects and determines interactions and events. It should not have any impact on what underlying properties "make" a true triple, DtoT, true double, or false event [24]. These reasons are enough for us to justify the merging of data sets from different simulated experiments using different dose rates in order to ensure we have balanced classes.

For any given subsection we used 20% of the data for validation and the remaining 80% for training. We used the Keras `tensorflow.Keras.Models.Model.fit` method for training and validation.

7.1 Comparing Original Data to New Data

There were quite a few issues with our previous data set used in [5]. The first issue is that, perhaps due to a configuration bug in the data generator, we cannot create similar data sets. In this section the data set used in [5] will be called "old data" and the data we use for the remaining results will be called "latest data". Any attempt to validate, train, or test on newly generated data sets results in a failure to learn. This is demonstrated by examining the distributions and viewing training plots.

Consider Figure 7.1. Notice how the old data has a similar skewed shape to the latest data but a completely different distribution of data among the bins. All three energies values have nearly 90% of their data falling within the first bin in the old data. This is in contrast to our current data which has around 70% of the energies in the first bin. We can see that the remaining bins for the

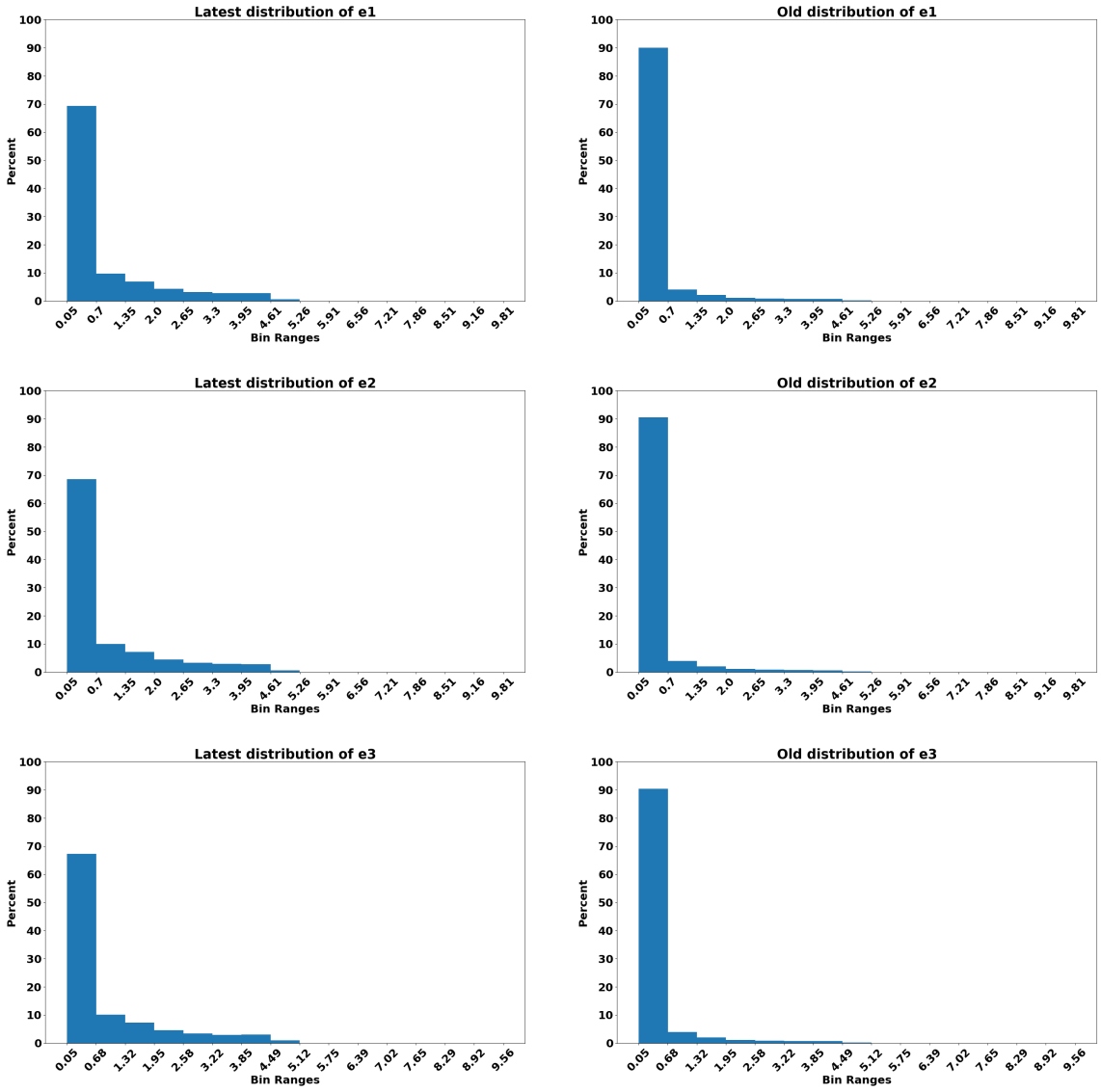


Figure 7.1: Each histogram represents the percent of records which fall within the particular bin range for any given energy feature for both the old data and latest data.

new data also contain more data than the old data. This difference in energy distributions implies that there is some unknown difference between the simulations of old and the ones we are using now.

Now we look to Figure 7.2. These plots show the percentage of data distributed for the first set of spatial coordinates. We admit that a difference in a couple percentage points can be treated with some skepticism given that the simulations rely on Monte Carlo methods and do not produce identical results even when given identical parameters. Notice that the distributions for x_1 and z_1 are nearly identical in their shape and percentages. The y_1 values, however, have different concentrations with similar shapes and ranges. To the uninitiated it is difficult to determine whether the simulations' randomness is to blame for these concentration differences. The differences in the energy distribution concentration and the y distribution concentration is too large to blame on

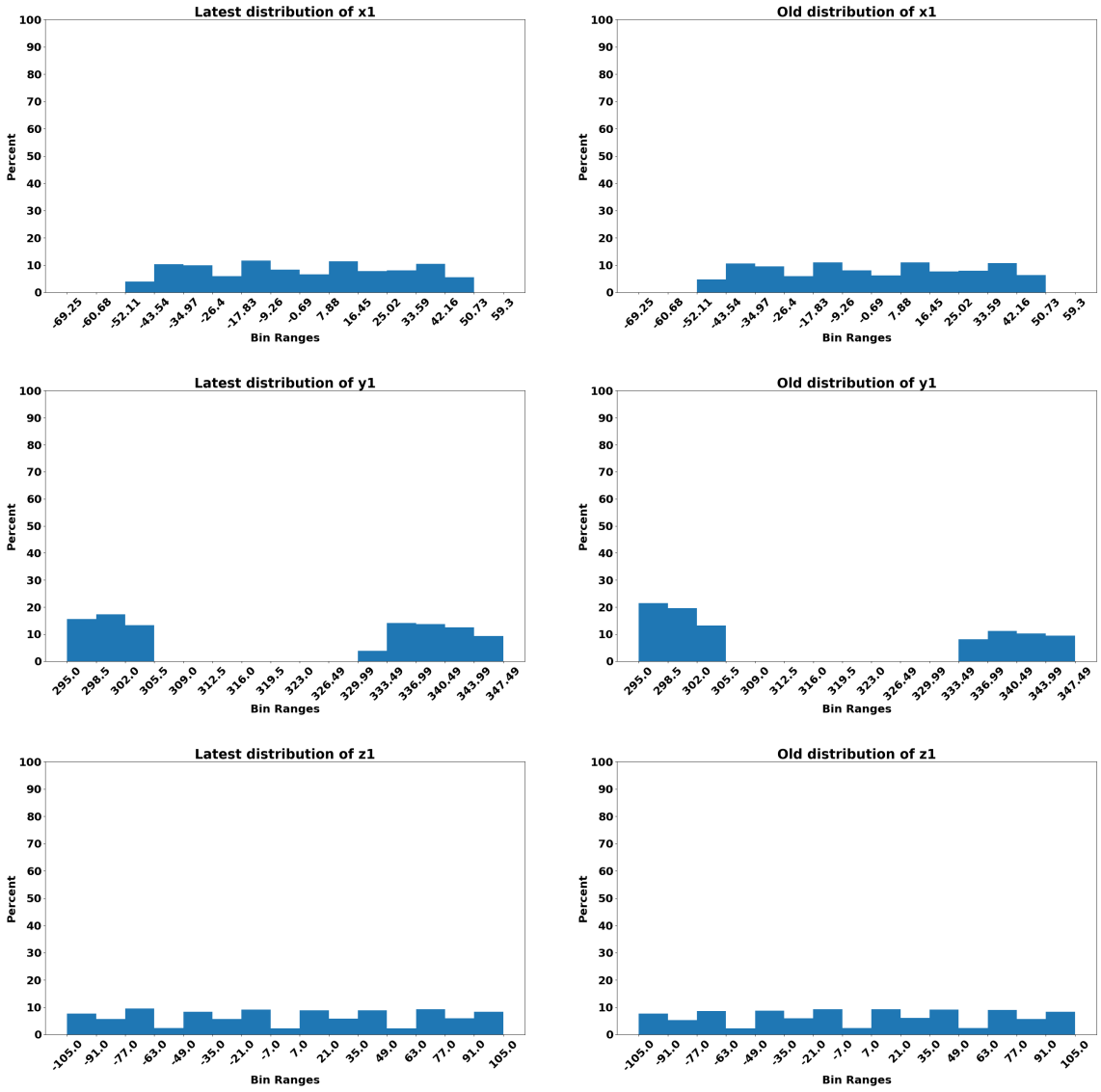


Figure 7.2: Each histogram represents the percent of records which fall within the particular bin range for any given spatial feature for both the old data and latest data.

randomness. These differences in the data, whatever the deeper specifics may be, is simply too great for the network to predict on one set when trained on the other.

When we look at Figure 7.3 this point is demonstrated. We train on the old data and validate on the latest data at every epoch and as a result the network's validation accuracy is significantly lower than the training accuracy. This usually means that the validation set, the latest data, is either not similar enough to the training data, or that the latest data is more complex. What it means to be more complex in this context is vague but most likely has to do with the underlying simulation differences between the old data and the latest data. Perhaps there are more defining patterns in the new data which do not exist in the old data. Maybe the old data contains simpler patterns which do not generalize well when the energies and y values have different concentrations in regards to their distributions seen in Figure 7.1 and Figure 7.2.

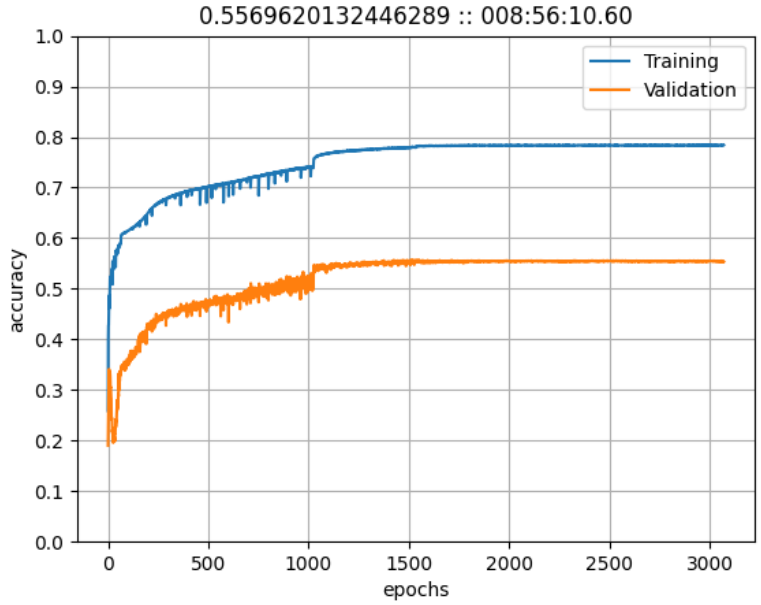
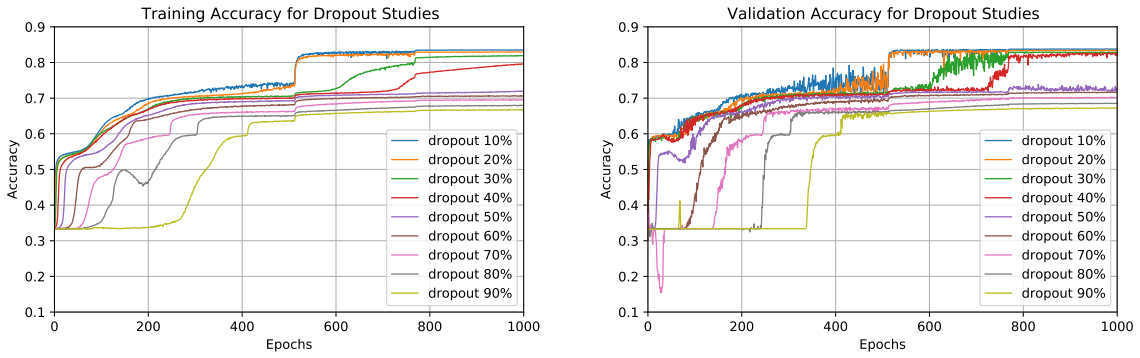


Figure 7.3: Training and validation plot for a fully connected network trained on the old data and validated on the latest data.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	63.8	2.6	9.6	2.6	5.7	4.4	0.9	0.9	1.3	0.0	3.1	3.5	1.7
132	2.7	62.5	2.7	17.0	3.1	3.1	0.4	0.4	1.3	2.2	2.2	1.8	0.4
213	10.7	3.1	63.8	3.6	10.3	2.7	0.9	1.3	1.3	1.8	0.0	0.0	0.4
231	2.7	11.2	3.1	60.7	1.8	8.0	1.8	4.9	0.4	2.2	0.4	0.9	1.8
312	1.3	6.2	9.4	1.8	67.9	2.2	0.0	1.3	2.2	4.5	1.3	0.4	1.3
321	3.6	0.9	2.7	8.5	3.1	68.8	1.8	8.5	0.4	0.0	0.4	0.9	0.4
124	0.4	0.4	0.9	0.4	0.0	0.0	36.9	7.1	0.0	0.0	0.0	0.0	53.8
214	0.4	0.9	0.4	0.9	0.0	0.4	1.3	38.7	0.0	0.0	0.0	0.0	56.9
134	0.0	0.0	0.4	0.4	0.0	0.9	0.0	13.8	28.0	0.4	0.0	0.0	56.0
314	0.9	0.4	0.0	0.0	0.0	1.3	0.0	0.0	10.2	31.1	0.0	0.0	56.0
234	0.4	0.0	1.3	0.4	0.9	0.0	0.0	0.0	0.0	8.0	31.1	0.0	57.8
324	0.0	0.9	0.0	2.2	0.0	0.0	0.0	0.0	0.0	0.0	2.2	41.3	53.3
444	0.4	1.3	0.9	0.9	0.0	0.0	4.9	5.8	6.7	4.9	2.2	5.8	66.1

Figure 7.4: Confusion matrix for a fully connected network trained on triples, double to triples, and false data from a 150MeV using old data. The testing data used is the new data which shares the same balanced classes.

This idea that the old data is less complex is painted clearer by the confusion matrix seen in Figure 7.4. In [5] we trained, tested, and validated on the old data. Now when validating and testing on the latest data we see similar triple accuracy for the new data as we saw for the old data in [5]. Yet the doubles-to-triples and false data numbers have fallen significantly. The majority of the doubles-to-triples are being improperly classified as false. This shows that the doubles-to-triples and the false data are the reason for a drop in accuracy when classifying the latest data. The doubles-to-triples and false events for the old data came from manually stitching singles to doubles and other singles together respectively. The doubles-to-triples and false events for the latest data



(a) (b)

Figure 7.5: (a) Training and (b) Validation accuracy for dropout values over the span of 1000 epochs

were generated by the simulation software only. It seems that the old method of attaching singles to other events does not mimic the idea of Compton camera bad detection. A more full breakdown of doubles-to-triples and false events could be done to see why this is the case but it is beyond the scope of this work.

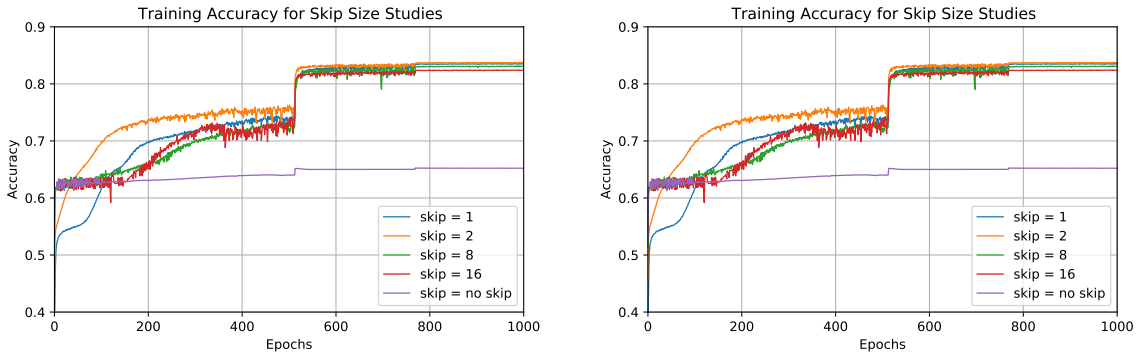
7.2 Doubles Hyperparameter Studies

We conducted hyperparameter studies that train and classify double events on 150MeV beam data for true doubles events. The hyperparameters studied consisted of the following: dropout rates, number of neurons per layer, batch sizes, batch normalization, number of layers, and size of the residual blocks. For the studies, our goal was to minimize time per epoch and to maximize training and validation accuracy. Rather than running a large grid search of all hyperparameters, which would be fairly time consuming, we tested each hyperparameter individually. The results and the conclusions of these studies were used in [18].

7.2.1 Dropout Studies

For the dropout studies, we studied dropout values from 0.1 to 0.9 with increments of 0.1.

In Figure 7.5, we see the training and validation accuracy for the varying dropout values over the span of 1000 epochs. From both figures, it appears that as the dropout value increases, the accuracy decreases. We see that for dropout values closer to 0, the training and validation accuracy plateau slightly above 80% whereas for dropout values closer to 1, the accuracies do not reach 70%. At the end of 1000 epochs, we see that the training and validation accuracies vary around 20%, where a dropout rate of 90% results in $\approx 65\%$ and a dropout rate of 20% results in $\approx 83\%$. We also see that the lower dropout rates learn faster than the larger dropout rates. This makes sense as larger dropout rates mean that more neurons are excluded from update cycles, resulting in less significant updates. Additionally, at 500 epochs, we see a significant jump for all dropout rates. While not shown, the dropout value did not change the average epoch training time by more than 0.03 of a second. As such, for future studies, we used a dropout rate of 20%.



(a) (b)

Figure 7.6: (a) Training and (b) Validation accuracy for skip size studies over the span of 1000 epochs

7.2.2 Residual Block Size Studies

To study the impact of residual block size on a network with 24 layers, we use a block size between 1 and 24, where 24 is equivalent to using no residual blocks.

Figure 7.6 shows the training and validation accuracies for a selected sample of residual block size values within the domain previously explained. The main observation is that there is a significant difference in training and validation accuracy when a residual block is used vs not. The two plots in the figure indicate that smaller residual block sizes perform slighter better than larger block sizes. However, this difference of $\approx 1\%$ is much less significant than the $\approx 20\%$ accuracy difference between residual block and no residual block. As previously explained, the large increase in accuracy at 500 epochs is a consequence of the learning rate scheduler. While not shown, there was not a significant difference in average training time per epoch for each of the residual block sizes. As such, for future studies, we used residual blocks of size 2, which produced the highest training and validation accuracies.

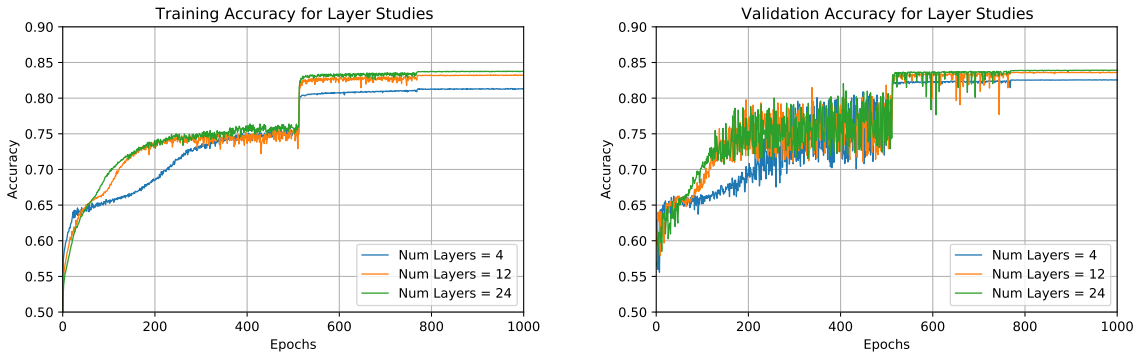
7.2.3 Number of Layers Studies

For the number of layer studies we studied different layer configurations from 4 to 24 layers. Figure 7.7 shows the training and validation accuracies for a selected sample of layer sizes within the range of 1-24. While we tested all values between 4 and 24, the tables only plot these three values as they represent the largest gaps in the model's accuracy. While the change in accuracy based on the number of layers is less than 5%, it appears that using more layers results in higher accuracy. This makes sense as more layers allows for greater complexity of the model therefore improving model's classification abilities.

7.2.4 Learning Rate Studies

The previous hyperparameter studies focused on smaller networks in order to maintain or slightly improve training and validation accuracy while possibly improving training time. However, for the learning rate studies, we trained on much larger neural networks. The hyperparamters were kept the same as the previous studies, except that the number of layers was set to 128.

One method of improving training and validation accuracy is via step schedulers which are discussed in Section 4.2.4. To optimize this kind of learning rate scheduler, we ran coarse learning



(a) (b)

Figure 7.7: (a) Training and (b) Validation accuracy for number of layer studies over the span of 1000 epochs

	0kMU				100kMU				180kMU		
	12	21	44		12	21	44		12	21	44
12	84.3	8.8	6.9	12	84.2	9.0	6.8	12	84.7	8.3	7.0
21	8.8	84.7	6.5	21	9.2	84.2	6.6	21	8.3	84.8	6.8
44	5.9	5.9	88.2	44	5.6	5.9	88.5	44	5.4	5.9	88.7

Figure 7.8: Confusion matrix for 0kMU, 100kMU, and 180kMU dose rates. The fully connected network was trained with 150MeV beams on true and false doubles data using a step-scheduler over 5000 epochs.

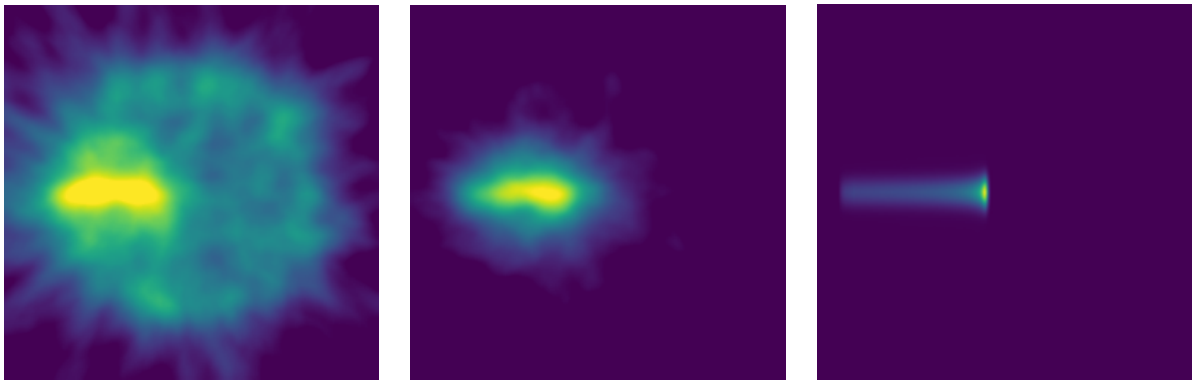
rates for 5000 epochs and determined at which epoch would be ideal for reducing the size of the learning rate. In a recursive order, we optimized a scheduler based on this method starting with a learning rate of 10^{-3} and ending training with a learning rate of 10^{-6}. We noticed that for larger learning rates, i.e., 10^{-2}, the model did not properly train as the learning rates were too coarse, which eventually lead to `NaN` values. For smaller learning rates, i.e., 10^{-7}, the learning rates were too small to have a significant impact on the training and validation accuracies. As such, the step schedulers did not use any values larger than 10^{-3} or smaller than 10^{-6}.

Through the use of a constant learning rate of 10^{-3}, we noticed that after a few hundred epochs, the model's accuracy, which peaked around 70%, started dropping, indicating that the learning rate was too coarse for continued training. As such, we used the baseline accuracy of the neural network as the peak of the training and validation accuracy before both metrics tanked. In comparison, using the learning rate step scheduler improved the training and validation accuracy more than 15%.

7.2.5 Maximizing Training and Validation Accuracy on True and False Doubles

For these studies, we added a third class, false doubles, and accompanying data. The hyperparameter values were set based on the previous studies, except for the learning rate value, where a new study was conducted to find a new optimal step-based learning rate scheduler.

Figure 7.8 displays three confusion matrices, one for the differing dose rates. 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 cells are shaded based on their current



(a) Uncleaned (b) Cleaned (c) Dose

Figure 7.9: Comparison of reconstructed prompt gamma rays with (a) uncleaned double data and (b) cleaned double data to the (c) dose delivered by the proton beam.

accuracy relative to the largest accuracy in each submatrix. The darkest entry in each row is the dominant classification of the input class. Within the matrices, there are three different labels: 12, 21, and 44. Label 12 represents the case where both interactions are correctly ordered and no corrections need to be made. Label 21 represents the scenario where the second interaction should be first and the first interaction should be second. The third label 44, is a false event, where the two interactions are actually two singles.

For each of the confusion matrices, the dominant classification for each input class is the class itself. This indicates that the model is able to correctly classify most samples. The matrix values between the dose rates do not appear to change by more than 1%, where higher dose rates have slightly higher accuracies. We see that the neural network classifies false events with around a 4% higher accuracy for all three dose rates compared to the true doubles events.

7.2.6 Reconstructions for True and False Doubles Data

Based on the classifications of the data from the neural network, we created a new data set of cleaned data, where events classified as false events were removed and the misordered events were reordered.

We cleaned the data with best performing neural network and then did reconstructions, seen in Figure 7.9, of the (a) uncleaned data, (b) cleaned, and (c) original dose. Ultimately, we want the reconstruction images to resemble the original dose as shown in Figure 7.9(c). When comparing the Figure 7.9(a) and (b), we see that the cleaned data has removed a significant portion of the noise as seen in uncleaned reconstruction. When we look at the difference between Figure 7.9(b) and Figure 7.9(c) there still is visual noise in the cleaned data that can be removed.

7.3 Random Forest Studies

A small discussion of the random forest method can be seen in Section 4.1. Training a neural network is very time consuming and is not always the right tool for the job. In some cases simpler systems like random forests are often preferable and feature better explainability than neural networks. We train over a collection of hyperparameters using sklearn's `RandomForestClassifier` for classification and sklearn's `RandomizedCVSearch` for hyperparameter tuning given the collection of hyperparameters mentioned in Table 7.1. Only the random forest with the best accuracy will be

Hyperparameter	Values
`min_samples_split`	2, 5, 10
`min_samples_leaf`	1, 2, 4
`max_depth`	10, 20, 30, 40, 50
`max_features`	`auto`, `sqrt`, `log2`
`bootstrap`	true and false

Table 7.1: This table is the collection of all hyperparameter names and values used when training random forest in Section 7.3. The left column is the name of the hyperparamter and the right column is the collection of possible values that the hyperparameter could possess when searching for the best random forest.

Hyperparameter	Value
`min_samples_split`	5
`min_samples_leaf`	2
`max_depth`	50
`max_features`	`auto`
`bootstrap`	false

Table 7.2: This table is the collection of all hyperparameter names and values used in the best random forest in Section 7.3.

considered and evaluated.

Figure 7.10 is a confusion matrix which is generated by a random forest, whose parameters are listed in Table 7.2, classifying the MCDE test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The doubles-to-triples and 444 entries are entirely 0

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	60.1	13.5	14.6	4.5	2.9	4.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	14.1	53.3	3.6	5.1	20.2	3.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	13.9	4.3	57.5	17.1	4.9	2.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	4.3	8.1	10.6	60.0	3.7	13.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	4.8	8.9	7.8	3.0	63.7	11.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	8.8	3.6	4.7	13.3	12.7	57.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.10: Confusion matrix for a random forest trained on true triples data from a 150MeV 0kMU beam and tested on the MCDE model test1 150MeV 20k beam. The doubles-to-triples and 444 entries are entirely 0 because any classification of non-true triple data by a network not trained with it is meaningless. This rows and columns which are entirely zeroes are left on the confusion matrix for consistency with the results in Section 7.4.3.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	60.6	14.6	15.1	3.7	2.6	3.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	12.8	52.9	3.5	4.7	22.7	3.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	14.6	3.8	54.6	19.1	5.3	2.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	2.8	7.3	10.7	61.3	3.6	14.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	4.5	7.8	7.4	2.9	63.5	14.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	8.7	3.8	4.8	12.4	13.4	57.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.11: Confusion matrix for a random forest trained on true triples data from a 150MeV 0kMU beam and tested on the MCDE model test1 150MeV 100k beam. The doubles-to-triples and 444 entries are entirely 0 because any classification of non-true triple data by a network not trained with it is meaningless. This rows and columns which are entirely zeroes are left on the confusion matrix for consistency with the results in Section 7.4.3.

because any classification of non-true triple data by a network not trained with it is meaningless. This rows and columns which are entirely zeroes are left on the confusion matrix for consistency with the results in Section 7.4.3.

We see that the dominant classification for each row is the input class itself. For the 123 class we see that the second and third highest classifications result in around 30% of the data and the remaining 10% is spread among the remaining classes. The observation that the 2nd and 3rd highest classifications soak around 30% of the data is consistent for each input class but there is not consistent reason reason why any input class may be put into the dominant incorrect classes. The 123 class' top classifications are 123, 213, 132 and the 312 class' top classifications are 312, 321, 132. Why is the 2nd highest classification of 123 213 and the 3rd highest classification 132? Why is the 2nd highest classification of 312 321 and the 3rd highest classification 132? There is no answer which currently can consistently explain what the input class, say 123, has in connection to a dominant misclassification like say 213.

Figure 7.11 is a confusion matrix which is generated by a random forest, whose parameters are listed in Table 7.2, classifying the MCDE test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The DtoT and 444 entries is entirely 0 because any classification of non-true triple data by a network not trained with it, is meaningless. This rows and columns which are entirely 0 are left on the confusion matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.10 also hold in their entirety for the triples and in Figure 7.11 with exceptions for the exact percentages mentioned.

Figure 7.12 is a confusion matrix which is generated by a random forest, whose parameters are listed in Table 7.2, classifying the MCDE test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The DtoT and 444 entries is entirely 0 because any

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	62.7	14.9	11.8	3.6	1.9	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	11.5	58.7	2.2	4.6	19.0	4.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	15.2	4.3	55.4	18.8	3.9	2.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	4.8	8.7	9.2	56.1	4.8	16.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	4.8	8.2	6.5	2.2	64.8	13.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	8.2	4.6	6.3	10.6	13.5	56.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.12: Confusion matrix for a random forest trained on true triples data from a 150MeV 0kMU beam and tested on the MCDE model test1 150MeV 180k beam. The doubles-to-triples and 444 entries are entirely 0 because any classification of non-true triple data by a network not trained with it is meaningless. This rows and columns which are entirely 0 are left on the confusion matrix for consistency with the results in Section 7.4.3.

classification of non-true triple data by a network not trained with it, is meaningless. This rows and columns which are entirely 0 are left on the confusion matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.10 also hold in their entirety for the triples and in Figure 7.12 with exceptions for the exact percentages mentioned.

We also have four additional MCDE testing tests at each of the previously used dose rates. All of the conclusions made about Figures 7.10, 7.11, and 7.12 all hold for the unlisted results mentioned. There are differences in the exact percentages but the general relationships discussed are identical.

As we will see in Section 7.4.1 the random forest performed considerably worse than the neural network. The maximum classification percentage of the random forest on the test1 20kMU beam was 63.7% and the worst was 53.3% for the dominant classification. The maximum classification percentage of the neural network on the test1 20kMU beam was 87.6% and the worst was 85.8% for the dominant classification. The random forest performed around 30% worse than the neural network. The random forest model when saved to disk using the `joblib` library with compression level 3 was 11 gigabytes. The neural network when saved using Keras' `Model.save` method uses 203 megabytes. The neural network outperforms the random forest in classification while using considerably less disk space. Both methods are fast enough to be used for data processing. We need a minimum of 80% classification accuracy for each class with a preferred accuracy greater than 95% for real world usage.

7.4 Expanding Three Scatter Data Sets

Due to the problems discussed in Section 7.1 we wanted to slowly expand our network's classification coverage. First starting with only true triples, then adding in doubles-to-triples, finishing with the inclusion of false triples. This allows us to gauge how the increase in data complexity is impacting the network's ability to learn.

Hyperparameter	Value
Learning Rate	See Section 4.2.4
Dropout	45%
Inter-Layer Activation	LeakyReLU
Number of Hidden Layers	256
Neurons per Layer	256
Residual Block Size	8 Layers
Number of Residual Blocks	32
Final Activation	Softmax
Batch Size	4096

Table 7.3: These hyperparameters determine the structure of the deep fully connected network used in Section 7.4.1

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	86.8	4.2	1.8	2.7	2.8	1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	3.1	87.3	1.9	2.2	2.6	2.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	1.8	3.4	86.5	3.6	2.0	2.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	2.8	2.0	3.1	87.5	3.0	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	2.9	1.8	2.0	2.8	87.6	2.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	1.8	3.2	3.7	1.9	3.5	85.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.13: Confusion matrix for a fully connected network trained on true triples from a 150MeV 0kMU beam The testing data used is the from the MCDE model test1 data 150MeV 20kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The doubles-to-triples and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

7.4.1 Triples Only

The true triples data set is the simplest of the three data sets used for training in Section 7.4. Every event in the input data is a triple which could be used for reconstruction if the interactions were correctly ordered. As discussed in Section 5.2 only the 123 class can be used for reconstruction. Any triple we correctly order from the 132, 213, 231, 312, 321 classes will yield a positive contribution to our reconstructed image. Any triple ordering other than 123 will pollute the reconstruction and only add noise to the final image. If the network can properly classify the true triples data set then we know that the network can correctly identify underlying patterns which indicate the real ordering of any given true triple. If the network can understand patterns within the true triple data set then it may be possible for the network to understand more complex scenarios like doubles-to-triples and false data in terms of three interaction events.

Figure 7.13 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 20kMU beam data.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	87.5	3.4	2.0	3.3	2.0	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	2.6	87.7	2.3	2.5	3.1	1.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	2.5	3.0	85.4	4.6	2.2	2.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	3.0	1.4	3.1	88.3	2.6	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	1.8	1.3	2.8	2.6	88.6	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	1.7	3.2	3.4	2.3	4.5	84.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.14: Confusion matrix for a fully connected network trained on true triples from a 150MeV 0kMU beam The testing data used is the from the MCDE model test1 data 150MeV 100kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The doubles-to-triples and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

The first column of the table is the input class and every proceeding column is the percentage of input which was assigned to the class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The doubles-to-triples and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

We see that the dominant classification for each row is the input class itself. The remaining cell accuracies in each row are within 3% of the remaining cells in the respective row. Less than 16% of all remaining data was miclassified during the classification process. This means that the neural network is capable of understanding how the patterns within the input data map to a given output class. Given these results we were confident that we could integrate more complicated data and maintain optimistic expectations of the neural network's performance.

Figure 7.14 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 100kMU beam data. The first column of the table is the input class and every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy is relative to the largest percentage present in entire confusion matrix. The DtoT and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is is meaningless. This row and column is left on the confusion matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.13 also hold in their entirety for the true triples in Figure 7.14 with exceptions for the exact percentages mentioned.

Figure 7.15 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 20kMU beam data. The first column of the table is the input class and every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy is relative to the largest percentage present in entire confusion matrix. The DtoT and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is is meaningless. This

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	85.6	6.0	1.7	2.2	2.6	1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
132	2.4	88.5	1.9	1.4	3.8	1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
213	1.2	2.7	88.4	3.6	2.4	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
231	3.1	1.9	4.1	84.8	4.1	1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
312	2.7	2.7	1.0	2.2	89.2	2.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
321	1.9	1.9	4.3	1.4	1.9	88.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
124	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
314	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
234	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
324	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.15: Confusion matrix for a fully connected network trained on true triples from a 150MeV 0kMU beam The testing data used is the from the MCDE model test1 data 150MeV 180kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The doubles-to-triples and 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

row and column is left on the confusion matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.13 also hold in their entirety for the true triples in Figure 7.15 with exceptions for the exact percentages mentioned.

In the real world we will not know if a given input is a true triple or not. If a given input happens to be a true triple then we have no way to determine its real ordering. Given this, these results on their own do not tell us whether or not the network in its current stage can aid in real world reconstructions. These results to give us the ability to have optimistic expectations about the neural networks ability to classify more complex data which are of closer to real world scenarios.

We also have four additional MCDE testing tests at each of the previously used dose rates. All of the conclusions made about Figures 7.13, 7.14, and 7.15 all hold for the unlisted results mentioned. There are differences in the exact percentages but the general relationships discussed are identical.a

The neural network performed considerably better than the random forest in Section 7.3. The maximum classification percentage of the random forest on the test1 20kMU beam was 63.7% and the worst was 53.3% for the dominant classification. The maximum classification percentage of the neural network on the test1 20kMU beam was 87.6% and the worst was 85.8% for the dominant classification. The random forest performed around 30% worse than the neural network. The random forest model when saved to disk using the `joblib` library with compression level 3 was 11 gigabytes. The neural network when saved using Keras' `Model.save` method uses 203 megabytes. The neural network outperforms the random forest in classification while using considerably less disk space. Both methods are fast enough to be used for data processing. We need a minimum of 80% classification accuracy for each class with a preferred accuracy greater than 95% for real world usage.

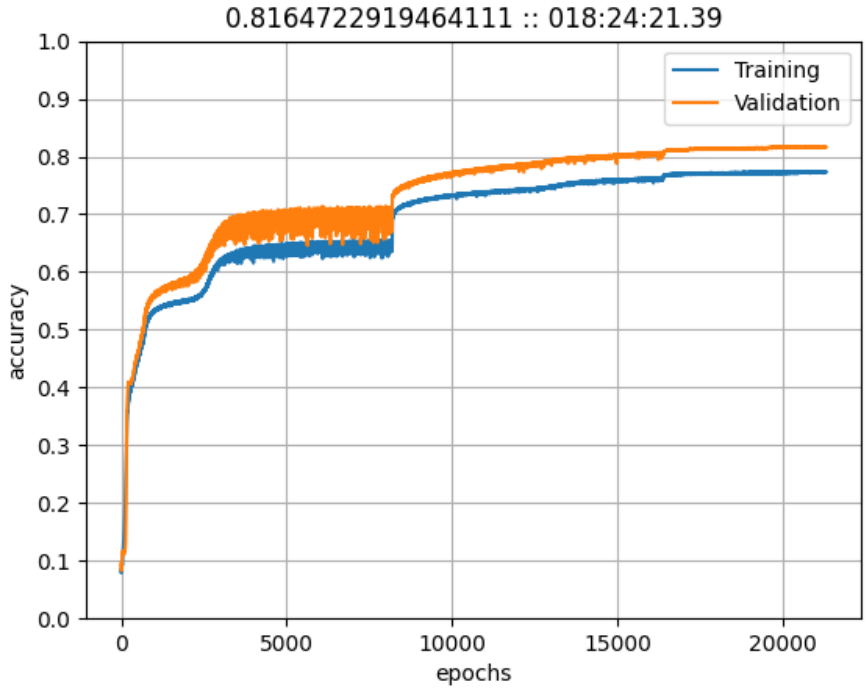


Figure 7.16: Training and validation plot for a fully connected network trained on triples from a 150MeV 0kMU beam and double to triples from a 150MeV 100kMU using ≈21k epochs. The validation data is 20% of the initial data.

7.4.2 Triples and Doubles-to-Triples

Our previous network configuration showed promising results for classifying true triples which has only 6 possible output classes. In this section we expand our data set to include true triples and doubles-to-triples. As stated in Section 5.2 the DtoT events have 6 possible classes: 124, 214, 134, 314, 234, 324. By merging the DtoT events with the true triples we are doubling the amount of output classes that our neural network will have to classify. The DtoT events have the added difficulty of having a single interaction which does not belong in the triplet. It is up to our neural network to determine which of the three interactions does not belong and also determine the correct order of the remaining pair. This also raises the issue that the network could start to put true triples, of any ordering, into the DtoT classes which removes their third interaction while also possibly destroying the ordering of the remaining pair.

Our network was trained on 400k triples and 400k doubles-to-triples making 800k total events. We used 20% of the data for validation and the remaining 80% for training. We used the Keras `tensorflow.Keras.Models.Model.fit` method for training and validation.

In Figure 7.16 we trained a deep fully connected neural network with the structural hyper-parameters listed in Table 7.3. In order to account for the increased data complexity born from using the DtoT with true triples we made two training changes compared to the training regime in Section 7.4.1. First we increased the number of epochs significantly to approximately 21k epochs. We also increased the gap of the tightening compared to training regime in Section 7.4.1 and the gap increases between each step. This was done to allow as much learning as possible at each stage of training while trying to account for possible plateaus.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	74.0	3.6	1.8	2.2	2.5	1.4	8.2	0.6	0.2	0.1	4.1	1.5	0.0
132	3.4	74.2	2.2	1.7	2.0	2.5	0.3	0.0	7.6	0.7	1.2	4.2	0.0
213	2.0	3.1	73.4	2.4	1.5	2.2	0.9	7.9	4.9	1.7	0.1	0.1	0.0
231	2.5	2.1	3.4	73.1	2.9	1.8	0.0	0.2	1.7	3.7	7.8	0.9	0.0
312	2.5	2.0	1.6	2.5	73.1	3.2	4.9	1.6	0.7	7.6	0.1	0.3	0.0
321	1.7	2.5	3.1	1.9	3.2	72.8	1.3	3.7	0.0	0.4	0.9	8.4	0.0
124	3.4	0.4	0.6	0.1	2.2	1.6	78.7	8.3	0.9	0.9	0.6	2.1	0.0
214	0.6	0.2	3.9	0.1	1.5	2.5	8.8	78.5	0.4	1.9	0.7	0.9	0.0
134	0.5	3.8	2.9	1.6	0.3	0.2	0.9	0.6	79.4	7.1	1.9	0.7	0.0
314	0.0	0.5	1.6	3.6	4.6	0.4	0.8	1.6	8.3	76.7	0.7	1.3	0.0
234	2.6	1.6	0.1	5.0	0.1	0.4	0.7	1.5	2.1	0.8	76.3	8.6	0.0
324	0.9	3.5	0.1	0.4	0.1	4.1	1.6	0.3	0.6	0.9	6.8	80.7	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.17: Confusion matrix for a fully connected network trained on triples from a 150MeV 0kMU beam and double to triples from a 150MeV 100kMU. The testing data used is the from the MCDE model test1 data 150MeV 20kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

We can see that the first huge spike in accuracy starts at 0 epochs where the network immediately obtains 40% validation and training accuracy. The slope of the line becomes less steep as the network is trying to fit with the randomly dropped neurons. Eventually the network seems to have fit the neurons well enough to plateau just before 5k epochs. This plateau holds until the sudden spike in accuracy at 8192 epochs where the learning rate is tightened from 10^{-3} to 10^{-4}. The network continues to learn at a slow but steady rate until a tiny spike in accuracy occurs at 16384 epochs where the learning rate tightens from 10^{-4} to 10^{-5}. The accuracy plateaus after this point increasing by fractions of a percent. The final tightening had little impact on the overall accuracy of the network.

An interesting observation is that the validation accuracy is considerably higher than the training accuracy. The exact reason for this occurrence is not completely known but this occurs for us when then dropout rate is greater than 0. One possible reason is that the training data is passed through the network with the dropout layers on but the validation data is passed through the network with the dropout layers off. The network gets to classify the validation data using all neurons rather than just some of the neurons. This increases the number of neurons present during validation which may allow the network to perform better on the unseen than the training data.

Figure 7.17 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 20kMU beam data. The first column of the table is the input class and every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

We see that the dominant classification for each row is the input class itself. The second highest classification percent for most classes of triple is the DtoT version of itself. This means that the network correctly identifies the first two interactions the best and, at times, struggles to correctly identify whether the third interaction truly belongs in the triplet. If we want to make light of this

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	73.7	4.2	1.7	2.5	2.2	1.0	6.7	1.0	0.3	0.0	5.2	1.3	0.0
132	2.6	74.4	1.9	1.6	1.5	1.6	0.1	0.0	9.7	0.8	1.4	4.3	0.0
213	2.3	3.3	72.7	3.0	2.6	1.2	1.0	6.5	5.6	1.5	0.1	0.1	0.0
231	2.5	1.0	3.2	74.2	2.8	2.7	0.1	0.4	1.9	3.7	6.9	0.7	0.0
312	1.5	1.8	2.1	2.7	75.9	3.1	3.9	1.2	0.7	6.6	0.0	0.4	0.0
321	1.0	2.7	3.0	2.3	4.9	70.9	1.7	4.5	0.1	0.4	0.3	8.2	0.0
124	4.4	0.3	0.4	0.0	2.4	1.3	79.1	8.0	0.9	0.7	0.7	1.7	0.0
214	0.5	0.2	4.2	0.3	1.1	3.4	7.6	78.5	0.8	1.9	0.7	0.8	0.0
134	0.2	4.7	2.2	1.4	0.4	0.0	0.9	0.7	78.6	8.2	2.0	0.5	0.0
314	0.1	0.7	1.3	2.6	4.2	0.2	0.5	1.4	8.7	78.2	0.7	1.5	0.0
234	2.7	1.5	0.4	4.1	0.1	0.3	0.6	0.7	1.4	0.8	78.9	8.5	0.0
324	1.0	2.2	0.2	0.8	0.5	3.7	1.5	0.6	1.0	0.7	7.7	80.1	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.18: Confusion matrix for a fully connected network trained on triples from a 150MeV 0kMU beam and double to triples from a 150MeV 100kMU. The testing data used is the from the MCDE model test1 data 150MeV 100kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

idea we can say that this percentage of classification still produces a valid true double which does have some use during reconstruction. Even more interesting is that the third highest classification yields another valid double! This tells us that the network is also likely to throw away the first interaction and keep the remaining two interactions and correctly order them. Take the 213 input class. The top 3 dominant classifications are 213 at 73.4%, 214 at 7.9%, and 134 at 4.9%. Given a collection of 213 events, which are 100% unusable, we get 73% back as perfectly ordered triples and 13% as perfectly ordered doubles. The remaining 14% pollute our data. This is a large amount of data recovered which would otherwise cause noise during reconstruction. The other classes of triple, like 123, where the third highest classification is another triple which only contributes more noise.

For the DtoT this behavior is not bi-directional. We see that the second highest classification for each DtoT event is actually a reverse ordering. The neural network can determine which of the three interaction does not belong to the pair but, after doing so, is unsure of the ordering given the remaining two interactions. The third highest classification is a triple where the two true interactions are correctly ordered but the third interaction stays attached. It is more difficult to spin this behavior in a more positive light. An incorrect classification of a DtoT is likely to only generate a misordered double or noise. It may be useful to pass a classified DtoT to the doubles classification network to help recover some of the misordered doubles but it is also likely that this idea only compounds the errors. Regardless we recognize that any DtoT is unusable for reconstruction and, as such, any improvement is welcomed.

Figure 7.18 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 100kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entry is entirely 0 because any classification of false data by a network not trained with it is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	71.9	5.3	2.6	1.2	2.6	1.7	8.2	0.7	0.0	0.0	3.4	2.4	0.0
132	2.4	76.2	1.4	1.0	1.0	1.9	0.2	0.0	7.2	0.7	1.4	6.5	0.0
213	1.4	1.9	73.0	3.4	1.7	2.2	0.5	9.4	4.3	1.7	0.5	0.0	0.0
231	2.7	1.2	4.6	70.1	4.3	2.7	0.2	0.2	1.0	4.3	8.2	0.5	0.0
312	1.4	2.9	0.7	2.2	74.0	3.1	5.5	1.2	0.7	7.7	0.0	0.5	0.0
321	1.4	2.2	3.6	1.2	3.4	72.8	2.2	5.1	0.0	0.0	1.0	7.2	0.0
124	3.8	0.6	0.2	0.1	2.4	1.7	79.6	7.9	1.2	0.5	0.6	1.4	0.0
214	0.8	0.0	3.2	0.3	1.5	2.5	8.1	79.3	0.5	1.7	1.2	0.9	0.0
134	0.2	4.2	2.1	1.2	0.7	0.0	0.7	0.7	78.9	8.2	2.1	1.0	0.0
314	0.1	0.5	0.7	3.2	4.4	0.3	0.7	2.0	7.5	78.2	0.6	1.7	0.0
234	2.9	1.0	0.5	3.1	0.1	0.6	0.6	0.9	1.3	0.6	79.3	9.1	0.0
324	1.4	2.7	0.1	0.6	0.2	3.8	2.1	0.3	1.2	0.7	7.8	79.2	0.0
444	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 7.19: Confusion matrix for a fully connected network trained on triples from a 150MeV 0kMU beam and double to triples from a 150MeV 100kMU. The testing data used is the from the MCDE model test1 data 150MeV 180kMU beam data. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entries are entirely 0 because any classification of data by a network not trained on similar data is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.17 also hold in their entirety for the triples and DtoT in Figure 7.18 with exceptions only in the exact percentages mentioned.

Figure 7.19 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.3, classifying the MCDE model test1 150MeV 180kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix. The 444 entry is entirely 0 because any classification of false data by a network not trained with it is meaningless. The rows and columns containing only zeroes remain in the matrix for consistency with the results in Section 7.4.3.

The conclusions made about Figure 7.17 also hold in their entirety for the triples and DtoT in Figure 7.19 with exceptions for the exact percentages mentioned.

We also have four additional MCDE testing tests at each of the previously used dose rates. For these additional but unlisted results, the conclusions made about Figures 7.17, 7.18, and 7.19 all hold true. There are differences in the exact percentages but the general relationships discussed are identical. The results and the conclusions of these studies were used in [3].

7.4.3 Triples, Doubles-to-Triples, and False Data

The results in Section 7.4.2 showed that the neural network configuration was capable of handling true and partially true data. At this point we aim to expand the data to include all possible classes which means the usage of false data on top of true triples and DtoTs. The false data is simply labeled as 444 and only adds one additional class but the data itself is completely worthless in regards to reconstruction value as is discussed in Section 3. Any amount of data which slips through the classification process cannot be recovered or used for anything and will pollute the reconstruction results. In order to get enough simulated false data we had to do hundreds of simulations and merge all of the false data from them. This provides us with enough false data to have balanced classes. The 444 class has as much data as any of the other classes. While not

Hyperparameter	Value
Learning Rate	See Section 4.2.4
Dropout	45%
Inter-Layer Activation	LeakyReLU
Number of Hidden Layers	512
Neurons per Layer	256
Residual Block Size	8 Layers
Number of Residual Blocks	64
Final Activation	Softmax
Batch Size	8192

Table 7.4: The hyperparameters which determine the structure of the deep fully connected network used in Section 7.4.3

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	77.3	3.2	1.4	2.1	2.7	1.6	6.9	0.5	0.2	0.0	2.9	0.8	0.4
132	2.8	77.5	2.0	1.5	2.3	2.5	0.3	0.0	6.5	0.4	1.0	2.9	0.3
213	1.4	2.6	77.7	2.8	1.6	2.2	0.6	6.4	3.3	1.1	0.1	0.0	0.2
231	2.9	1.7	2.8	78.4	2.9	1.6	0.1	0.2	0.8	2.5	5.5	0.5	0.2
312	2.7	1.2	1.5	2.2	79.8	2.3	3.1	1.2	0.5	5.3	0.0	0.2	0.1
321	1.5	2.7	3.1	1.7	3.2	78.4	0.9	2.5	0.0	0.2	0.5	5.1	0.3
124	3.5	0.2	0.6	0.1	2.8	1.8	74.0	8.3	0.4	0.4	0.4	1.3	6.2
214	0.4	0.3	3.7	0.3	1.5	2.8	7.3	76.8	0.3	1.2	0.4	0.3	4.8
134	0.4	3.8	3.0	2.1	0.4	0.1	0.5	0.4	75.8	7.4	1.1	0.7	4.1
314	0.0	0.6	1.8	4.0	5.4	0.3	0.8	0.8	6.2	73.8	0.1	0.8	5.2
234	2.4	1.5	0.1	5.7	0.2	0.9	0.2	0.7	1.3	0.4	72.9	7.8	5.9
324	1.1	3.2	0.2	0.5	0.2	5.2	1.3	0.3	0.6	0.7	6.4	75.9	4.4
444	0.3	1.3	0.3	0.9	0.0	0.3	4.7	5.0	1.9	3.4	5.0	6.0	70.8

Figure 7.20: Confusion matrix for a fully connected network trained on triples, double to triples, and false data from a 150MeV train for approximately 18k epochs and tested on the MCDE model test1 150MeV 20k beam. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix.

immediately obvious, we do end up with some conceptual contention for what it means to have truly balanced classes. We have 6× more triples and 6× more DtoT than we do false triples but triples and DtoTs are both are split into 6 different classes. What counts as balance is a hard question in the context of our problem so we opt for an equal number of records in each class over other possible interpretations.

Our network was trained on 840k triples, 840k DtoT, and 140k false triples making 1.8M total events. We used 20% of the data for validation and the remaining 80% for training. We used the Keras `tensorflow.Keras.Models.Model.fit` method for training and validation.

Figure 7.20 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.4, classifying the MCDE model test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix.

We see that the dominant classification for each row is the input class itself. The second highest classification percent for most classes of triple is the double-to-triple version of itself. This means

that the network correctly identifies the first two interactions the best and, at times, struggles to correctly identify whether the third interaction truly belongs in the triplet. If we want to make light of this idea we can say that this percentage of classification still produces a valid true double which does have some use during reconstruction. Even more interesting is that the third highest classification yields another valid double! This tells us that the network is also likely to throw away the first interaction and keep the remaining two interactions and correctly order them. The true triples are very rarely mistaken as false data. Take the 213 input class. The top 3 dominant classifications are 213 at 77.7%, 214 at 6.4%, and 134 at 3.3%. Given a collection of 213 events which are 100% unusable. We get 78% back as perfectly ordered triples and 9% as perfectly ordered doubles. The remaining 13% pollute our data. This is a large amount of data recovered. The other classes of triple, like 123, where the third highest classification is another triple which only contributes more noise.

For the DtoT we see that the second highest classification for each DtoT event is actually a reverse ordering. The neural network can determine which of the three interaction does not belong to the pair but, after doing so, is unsure of the ordering given the remaining two. In hopes of alleviating the DtoT event being cast as misordered doubles it may be useful to pass classified doubles-to-triples to the doubles classification network but it is also likely this idea only compounds the errors. The third highest classification for some inputs is a triple where the two true interactions are correctly ordered but the third interaction, a improperly coupled single, stays attached. This makes the DtoT appear as a valid triple and creates noise during reconstruction. Other DtoT have false events as their third highest classification. If a DtoT is classified as a false event and is thrown away then this is still more optimal than having it incorrectly used as a true triple. In the context of reconstruction no data is better than bad data. Regardless we recognize that a DtoT without adjustments is unusable for reconstruction and, as such, any improvement is welcomed.

The false data classification is a more interesting story. Any improper classification of false data means that we are guaranteed to see noise occur in our reconstruction. The dominant classification of false events is false events which yields a very positive outcome. The majority of the incorrect classifications of the false data fall directly into the DtoT classes. Very few false events fall into the true triple categories. A false triple being classified as a DtoT means that we now have a false double being passed through the system. There is no reason to believe that a false triple with an interaction removed will be fundamentally different than a naturally occurring false double. If we were to then pass that false double to the double classification network it has an 80% chance of being removed entirely. This means that of the roughly 26% of false events classified as DtoT events, assuming an 80% removal rate via the double classification network, only around 5% will make it through to the final reconstruction process. Of course it would be most optimal if the triple classification network did not need additional assistance but, in general, these results are not nearly as bad they seem on the surface when you follow this train of thought. Of course additional experimentation and testing will be needed in order to fully verify these ideas.

Figure 7.21 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.4, classifying the MCDE model test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix.

The conclusions made about Figure 7.20 also hold in their entirety for the triples, DtoT, and false data in Figure 7.21 with exceptions to the exact percentages mentioned.

Figure 7.22 is a confusion matrix which is generated by a fully connected neural network, whose parameters are listed in Table 7.4, classifying the MCDE model test1 150MeV 20kMU beam data. The first column of the table is the input class and the every proceeding column is the percentage

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	79.1	2.3	1.5	2.2	1.7	1.5	5.6	0.7	0.1	0.0	3.9	1.0	0.2
132	2.6	76.0	2.0	1.7	2.8	1.9	0.1	0.0	7.7	0.4	1.2	3.5	0.1
213	1.7	2.5	76.4	3.7	2.0	2.0	0.4	5.5	3.9	1.2	0.3	0.1	0.2
231	3.0	1.4	2.8	80.7	2.2	1.7	0.1	0.2	1.3	1.8	4.4	0.1	0.2
312	1.8	0.8	2.1	2.0	82.4	2.6	2.2	0.5	0.2	4.7	0.1	0.2	0.2
321	1.7	2.6	3.7	1.5	3.5	76.5	1.2	3.9	0.0	0.4	0.2	4.4	0.4
124	5.0	0.3	0.4	0.0	2.9	1.4	76.0	7.0	0.6	0.4	0.2	1.1	4.7
214	0.5	0.2	4.1	0.2	1.5	3.5	7.6	75.0	0.5	1.1	0.5	0.3	4.9
134	0.2	4.3	2.3	1.7	0.5	0.1	0.6	0.5	75.4	7.9	1.0	0.3	5.3
314	0.1	0.4	1.4	3.4	4.9	0.2	0.3	0.8	7.0	75.4	0.2	1.0	4.9
234	3.1	1.5	0.5	5.4	0.1	0.5	0.4	0.5	1.0	0.4	73.6	7.1	5.8
324	1.0	2.3	0.1	0.7	0.4	5.1	1.1	0.2	0.6	0.4	6.5	75.3	6.2
444	0.6	0.2	0.4	0.5	0.8	0.5	4.2	3.8	5.1	4.0	3.9	3.5	72.6

Figure 7.21: Confusion matrix for a fully connected network trained on triples, double to triples, and false data from a 150MeV train for approximately 18k epochs and tested on the MCDE model test1 150MeV 100k beam.

	123	132	213	231	312	321	124	214	134	314	234	324	444
123	74.5	5.0	2.2	2.2	2.9	1.7	6.7	0.5	0.0	0.0	2.9	1.4	0.0
132	2.2	77.4	1.9	1.2	2.2	1.9	0.0	0.0	6.2	0.5	1.0	4.6	1.0
213	0.7	2.2	77.6	3.1	1.2	1.7	0.5	8.7	3.1	1.0	0.2	0.0	0.0
231	2.4	1.9	3.9	76.6	3.9	1.7	0.2	0.5	0.5	2.4	5.1	0.7	0.2
312	2.7	1.7	1.0	1.9	80.2	2.9	2.7	1.0	0.2	5.1	0.2	0.0	0.5
321	1.4	2.2	3.6	1.7	3.1	79.0	1.4	3.4	0.0	0.0	0.5	3.6	0.0
124	3.8	0.3	0.3	0.2	2.3	1.9	76.4	6.9	0.6	0.4	0.4	1.0	5.5
214	0.8	0.1	3.6	0.4	1.5	3.2	6.6	76.9	0.3	0.9	0.2	0.5	5.1
134	0.2	4.8	2.7	1.4	0.9	0.1	0.5	0.4	75.9	7.5	1.3	0.6	3.7
314	0.1	0.1	1.0	3.2	6.0	0.4	0.3	1.0	7.1	74.4	0.4	0.6	5.5
234	3.3	1.1	0.3	4.2	0.3	0.9	0.2	0.5	0.7	0.4	75.6	6.9	5.6
324	1.7	2.9	0.1	0.5	0.3	5.4	1.5	0.2	0.3	0.2	7.4	73.1	6.4
444	0.3	0.7	0.3	0.6	0.4	0.3	4.6	3.9	4.1	4.5	5.1	4.3	70.8

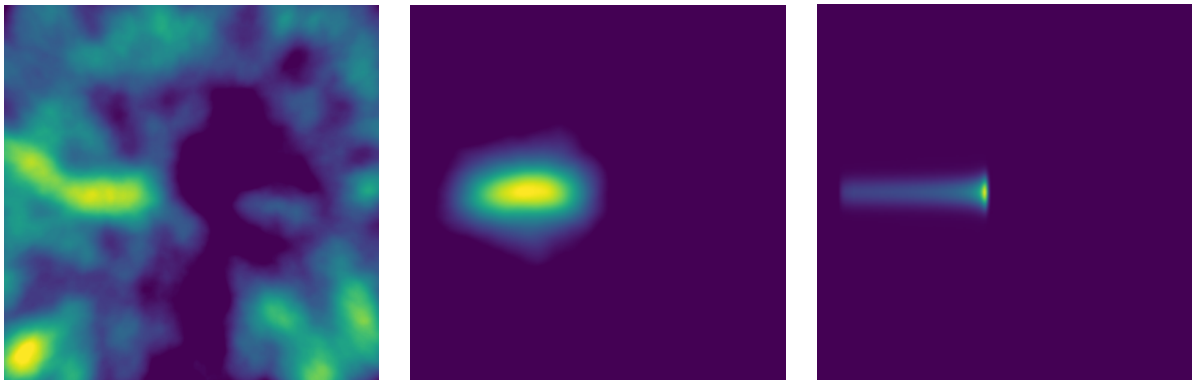
Figure 7.22: Confusion matrix for a fully connected network trained on triples, double to triples, and false data from a 150MeV train for approximately 18k epochs and tested on the MCDE model test1 150MeV 180k beam.

of input which was assigned to class at the top of column. Each cell is colored by accuracy relative to the largest percentage present in entire confusion matrix.

The conclusions made about Figure 7.20 also hold in their entirety for the triples, DtoT, and false data in Figure 7.22 with exceptions to the exact percentages mentioned.

In Figure 7.23 we clean the data using our neural network and then reconstruct the (a) uncleaned, (b) cleaned, and (c) real dose. The cleaned image has a clear beam shape and a clear stopping point which is similar to the dose image. The uncleaned image is extremely noisy and we cannot make any conclusions about the beams shape or depth. More detailed discussions, conclusions, and results about the application, usage, and impact of the neural network on real-world testing and physics is detailed in [23].

We also have four additional MCDE testing tests at each of the previously used dose rates. All of the conclusions made about Figures 7.17, 7.18, and 7.19 all hold for the unlisted results mentioned. There are differences in the exact percentages but the general relationships discussed



(a) Uncleaned (b) Cleaned (c) Dose

Figure 7.23: Comparison of reconstructed prompt gamma rays with (a) uncleaned triples data and (b) cleaned triples data to the (c) dose delivered by the proton beam.

are identical.

8 Conclusions

In Section 7.1 we highlighted that there were quite a few issues with our previous data set used in [5]. The first issue is that, perhaps due to a configuration bug in the data generator, we cannot create similar data sets. This is demonstrated by examining the distributions in Figure 7.1 . Additionally we see that any attempt to validate, train, or test on newly generated data sets results in a failure to learn which is seen in the training plots in Figure 7.3. Figure 7.4 shows that the doubles-to-triples and the false data are the reason for a drop in accuracy when classifying the latest data. The doubles-to-triples and false events for the old data came from manually stitching singles to doubles and other singles together respectively. The doubles-to-triples and false events for the latest data were generated by the simulation software only. It seems that the old method of attaching singles to other events does not mimic the idea of Compton camera bad detection. A more full breakdown of doubles-to-triples and false events could be done to see why this is the case but it is beyond the scope of this work.

We did several hyperparameter studies on different sets of doubles data to determine the optimal size for the residual blocks, number of layers, dropout rate, learning rate schedule, neurons per layer, and batch sizes in Section 7.2. The results and the conclusions of these studies were used in [18]. We determined many optimal hyperparameters for the doubles classification network and used this optimal set to produce Figure 7.8 which displays three confusion matrices, one for each of the different dose rates. For each of the confusion matrices, the dominant classification for each input class is the class itself. This indicates that the model is able to correctly classify most samples. The matrix values between the dose rates do not appear to change by more than 1%, where higher dose rates have slightly higher accuracies. We see that the neural network classifies false events with around a 4% higher accuracy for all three dose rates compared to the true doubles events. We cleaned the data with best performing neural network and then did reconstructions, seen in Figure 7.9, of the (a) uncleaned data, (b) cleaned, and (c) original dose. Ultimately, we want the reconstruction images to resemble the original dose as shown in Figure 7.9(c). When comparing the Figure 7.9(a) and (b), we see that the cleaned data has removed a significant portion of the noise as seen in uncleaned reconstruction. When we look at the difference between Figure 7.9(b)

and Figure 7.9(c) there still is visual noise in the cleaned data that can be removed.

In Section 7.3 we train several random forests using sklearn and pick the best performing random forest to generate the confusion matrix seen in Figure 7.10. The best random forests hyperparameters can be seen in Table 7.2 We see that the dominant classification for each row is the input class itself. For the 123 class we see that the second and third highest classifications result in around 30% of the data and the remaining 10% is spread among the remaining classes. The that the 2nd and 3rd highest classifications soak around 30% of the data is consistent for each input class but there is no pattern between the how any input class may related to the dominant incorrect classes. The 123 class' top classifications are 123, 213, 132 and the 312 class' top classifications are 312, 321, 132. Why is the 2nd highest classification of 123 213 and the 3rd highest classification 132? Why is the 2nd highest classification of 312 321 and the 3rd highest classification 132? There is no answer which currently can consistently explain what the input class, say 123, has in connection to a dominant misclassification like 213. The random forest performed considerably worse than the neural network in Section 7.4.1. The maximum classification percentage of the random forest on the test1 20kMU beam was 63.7% and the worst was 53.3% for the dominant classification. The maximum classification percentage of the neural network on the test1 20kMU beam was 87.6% and the worst was 85.8% for the dominant classification. The random forest performed around 30% worse than the neural network. The random forest model when saved to disk using the `joblib` library with compression level 3 was 11 gigabytes. The neural network when saved using Keras' `Model.save` method uses 203 megabytes disk space. The neural network outperforms the random forest in classification while using considerably less disk space. Both methods are fast enough to be used for data processing. We need a minimum of 80% classification accuracy for each class with a preferred accuracy greater than 95% for real world usage.

In Section 7.4 we decided that, based on the results in Section 7.1, we wanted to slowly expand our network's classification coverage. First starting with only true triples, then adding in doubles-to-triples, finishing with the inclusion of false triples. This allows us to gauge how the increase in data complexity is impacting the network's ability to learn. The results of our expansion is best highlighted in Figure 7.13 for triples, Figure 7.17 for triples/doubles-to-triples, and Figure 7.20 for triples/doubles-to-triples/false. In the most complex set, the triples/doubles-to-triples/false results, we see that the dominant classification for each row is the input class itself. The second highest classification percent for most classes of triple is the double-to-triple version of itself. This means that the network correctly identifies the first two interactions the best and, at times, struggles to correctly identify whether the third interaction truly belongs in the triplet. If we want to make light of this idea we can say that this percentage of classification still produces a valid true double which does have some use during reconstruction. Even more interesting is that the third highest classification yields another valid double! This tells us that the network is also likely to throw away the first interaction and keep the remaining two interactions and correctly order them. The true triples are very rarely mistaken as false data. Take the 213 input class. The top 3 dominant classifications are 213 at 77.7%, 214 at 6.4%, and 134 at 3.3%. Given a collection of 213 events which are 100% unusable. We get 78% back as perfectly ordered triples and 9% as perfectly ordered doubles. The remaining 13% pollute our data. This is a large amount of data recovered. The other classes of triple, like 123, where the third highest classification is another triple which only contributes more noise. For the DtoT we see that the second highest classification for each DtoT event is actually a reverse ordering. The neural network can determine which of the three interaction does not belong to the pair but, after doing so, is unsure of the ordering given the remaining two. In hopes of alleviating the DtoT event being cast as misordered doubles it may be useful to pass classified doubles-to-triples to the doubles classification network but it is also likely this idea only compounds the errors. The third highest classification for some inputs is a triple where the two

true interactions are correctly ordered but the third interaction, a improperly coupled single, stays attached. This makes the DtoT appear as a valid triple and creates noise during reconstruction. Other DtoT have false events as their third highest classification. If a DtoT is classified as a false event and is thrown away then this is still more optimal than having it incorrectly used as a true triple. In the context of reconstruction no data is better than bad data. Regardless we recognize that a DtoT without adjustments is unusable for reconstruction and, as such, any improvement is welcomed. The false data classification is a more interesting story. Any improper classification of false data means that we are guaranteed to see noise occur in our reconstruction. The dominant classification of false events is false events which yields a very positive outcome. The majority of the incorrect classifications of the false data fall directly into the DtoT classes. Very few false events false into the true triple categories. A false triple being classified as a DtoT means that we now have a false double being passed through the system. There is no reason to believe that a false triple with an interaction removed will be fundamentally different than a naturally occurring false double. If we were to then pass that false double to the double classification network it has an 80% chance of being removed entirely. This means that of the roughly 26% of false events classified as DtoT events, assuming an 80% removal rate via the double classification network, only around 5% will make it through to the final reconstruction process. Of course it would be most optimal if the triple classification network did not need additional assistance but, in general, these results are not nearly as bad they seem on the surface when you follow this train of thought. Of course additional experimentation and testing will be needed in order to fully verify these ideas. In Figure 7.23 we clean the data using our neural network and then reconstruct the (a) uncleaned, (b) cleaned, and (c) real dose. The cleaned image has a clear beam shape and a clear stopping point which is similar to the dose image. The uncleaned image is extremely noisy and we cannot make any conclusions about the beams shape or depth. We also have four additional MCDE testing tests at each of the previously used dose rates. All of the conclusions made about Figures 7.17, 7.18, and 7.19 all hold for the unlisted results mentioned. There are differences in the exact percentages but the general relationships discussed are identical. More detailed discussions, conclusions, and results about the application, usage, and impact of the neural network on real-world testing and physics is detailed in [23].

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] Carlos A. Barajas, Gerson C. Kroiz, Matthias K. Gobbert, and Jerimy C. Polf. Using deep learning to enhance compton camera based prompt gamma image reconstruction data for proton radiotherapy. *Proc. Appl. Math. Mech. (PAMM)*, submitted (2021).

[4] Jonathan N. Basalyga, Carlos A. Barajas, Matthias K. Gobbert, Paul Maggi, and Jerimy Polf. Deep learning for classification of Compton camera data in the reconstruction of proton beams in cancer treatment. *Proc. Appl. Math. Mech. (PAMM)*, 20(1):e202000070, 2021.

[5] Jonathan N. Basalyga, Carlos A. Barajas, Gerson C. Kroiz, Matthias K. Gobbert, Paul Maggi, and Jerimy Polf. Improvements to the deep learning classification of Compton camera based prompt gamma imaging for proton radiotherapy. Technical Report HPCF–2020–29, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2020.

[6] 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.

[7] W. H. Bragg and R. Kleeman. On the ionization curves of radium. *London Edinburgh Dublin Philos. Mag. J. Sci.*, 8(48):726–738, 1904.

[8] Lo Breiman. Random forests. *Mach. Learn.*, 45:5–32, 2001.

[9] John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990.

[10] François Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

[11] Arthur H. Compton. A quantum theory of the scattering of x-rays by light elements. *Phys. Rev.*, 21(5):483–502, 1923.

[12] 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.

[13] M. Fontana, J.-M. Ley, D. Dauvergne, N. Freud, J. Krimmer, J. M. Létang, V. Maxim, M.-H. Richard, I. Rinalida, and É. Testa. Monitoring ion beam therapy with a Compton camera: Simulation studies of the clinical feasibility. *IEEE Trans. Radiat. Plasma Med. Sci.*, 4(2):218–232, 2020.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[15] Muhammad Khalid, Junaid Baber, Mumraiz Khan Kasi, Maheen Bakhtyar, Varsha Devi, and Naveed Sheikh. Empirical evaluation of activation functions in deep convolution neural network for facial expression recognition. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 204–207, 2020.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[18] Gerson C. Kroiz, Carlos A. Barajas, Matthias K. Gobbert, and Jerimy C. Polf. Exploring deep learning to improve Compton camera based prompt gamma image reconstruction for proton radiotherapy. In *The 17th International Conference on Data Science (ICDATA'21)*, accepted (2021).

[19] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *Commun. Comput. Phys.*, 28(5):1671–1706, 2020.

[20] Paul Maggi, Steve Peterson, Rajesh Panthi, Dennis 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.*, 65(12):125004, 2020.

[21] Enrique Muñoz, Ana Ros, Marina Borja-Lloret, John Barrio, Peter Dendooven, Josep F. Oliver, Ikechi Ozoemelam, Jorge Roser, and Gabriela Llosá. Proton range verification with MACACO II Compton camera enhanced by a neural network for event selection. *Sci. Rep.*, 11(1):9325, 2021.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[23] Jerimy C. Polf, Carlos A. Barajas, Gerson C. Kroiz, Stephen W. Peterson, Paul Maggi, Dennis S. Mackin, Sam Beddar, and Matthias K. Gobbert. A study of the clinical viability of a prototype Compton camera for prompt gamma imaging based proton beam range verification. In *AAPM Virtual 63rd Annual Meeting*, submitted (2021).

[24] Jerimy C. Polf, Paul Maggi, Rajesh Panthi, Stephen Peterson, Dennis Mackin, and Sam Beddar. The effects of Compton camera data acquisition and readout timing on PG imaging for proton range verification. *IEEE Trans. Radiat. Plasma Med. Sci.*, pages 1–1, 2021.

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

[26] H Rohling, M Priegnitz, S Schoene, A Schumann, W Enghardt, F Hueso-González, G Pausch, and Fiedler. Requirements for a Compton camera for *in vivo* range verifications of proton therapy. *Phys. Med. Biol.*, 62:2795–2811, 2017.

[27] V. Schönfelder, A. Hirner, and K. Scheider. A telescope for soft gamma ray astronomy. *Nucl. Instruments Meth.*, 107:385–394, 1973.

[28] R. W. Todd, J. M. Nightingale, and D. B. Everett. A proposed γ camera. *Nature*, 251:132–134, 1974.

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

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

[31] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[32] Andreas Zoglauer and Steven E. Boggs. Application of neural networks to the identification of the Compton interaction sequence in Compton imagers. In *IEEE Nuclear Science Symposium Conference Record*, 2007.