Predicting depth-of-interaction in LYSO scintillators with Random Forest

Completed for the Certificate in Scientific Computation and Data Sciences
Fall 2023

Firas Abouzahr
Bachelor of Science
Department of Physics
College of Natural Sciences

[illegible]

Karol Lang
Professor of physics
Department of Physics

Predicting depth-of-interaction in LYSO scintillators with Random Forest

Firas Abouzahr
Research Advisor: Professor Karol Lang

August 2nd, 2022

Abstract

We trained categorical and regression random forest models to predict depth-of-interaction in 30 mm LYSO scintillator crystals for double-ended readout PET. The impact of energy thresholds was explored, and the accuracy (categorical) and RMSE (regression) continuously improved as we cut closer to the photopeak. Mean minimal depth calculations illustrated that normalized count differences were the most important feature, while time was the least important. Transformations to improve input variables were explored, and it was found that training regression RF with 3-dimensional UMAP embedding of charge, channel ID, and NCD improved the centering of the prediction distributions around their proper truth DOI. We also observed that z-score transformations helped create more considerable statistical differences between the charge distributions of adjacent DOIs. Training with the z-score transformed charge greatly improved regression results to have an RMSE of 1.42 mm when the maximal depth of the decision trees was increased to 20 with 2σ photopeak cut data. Although NCD proved to be the most important feature, we conclude that the need for transformed and additional data beyond NCD to accurately predict DOI illustrates that the typical statistical method of estimating DOI solely on NCD neglects important information that could improve the resolution. Our final regression model is able to predict DOI with an average resolution of 0.97 mm at an efficiency of 72.6%. Efforts are underway to help increase the efficiency of the resolutions.

Acknowledgments

I am very thankful to my research advisor Professor Karol Lang for his mentorship and support during this project, the countless other projects I have embarked on in his research group, and for teaching me what it means to be a physicist and a researcher.

Thank you to Marek Proga for designing and commissioning the experimental apparatus as well as for providing many of the schematics in this report. Thank you to Alex Kuo, Aryan Ojha, and Michael Gajda for brainstorming ideas and engaging in helpful conversations with me throughout this project. Thank you to Will Matava, Michael Gajda, and Dr. John Cesar for carrying out the depth-of-interaction experiments from which the data used in this report comes from.

Thank you to the TOF-PET for Proton Therapy (TPPT) consortium for providing the instrumentation used to conduct our DOI experiments.

Finally, thank you to the Department of Statistics and Data Science for offering the certification in Scientific Computation and Data Sciences. The coursework along with this research semester has allowed me to gain invaluable skills in computational and data analysis skills.

Contents

1 Introduction

1.1 Positron Emission Tomography

Positron emission tomography is a molecular imaging technique that works on the basis of detecting radiation from positron-emitting radioisotopes within a patient's tissue [1]. These radioisotopes can either be injected into a patient in the form of radiopharmaceuticals, which then accumulate in regions of high metabolic function, such as a tumor, or directly activated in the patient's tissue with a particle beam. Positrons emitted from these radioisotopes will then annihilate with electrons in the tissue into two 511 kev back-to-back gamma rays. Typical PET scanners are composed of crystal scintillator arrays coupled to SiPMs, a type of light sensor. When a gamma deposits its energy into the crystal scintillator, it produces optical photons, which are then detected by the SiPMs. A single PET module and a schematic of a full brain-PET scanner are pictured in Figure 1.

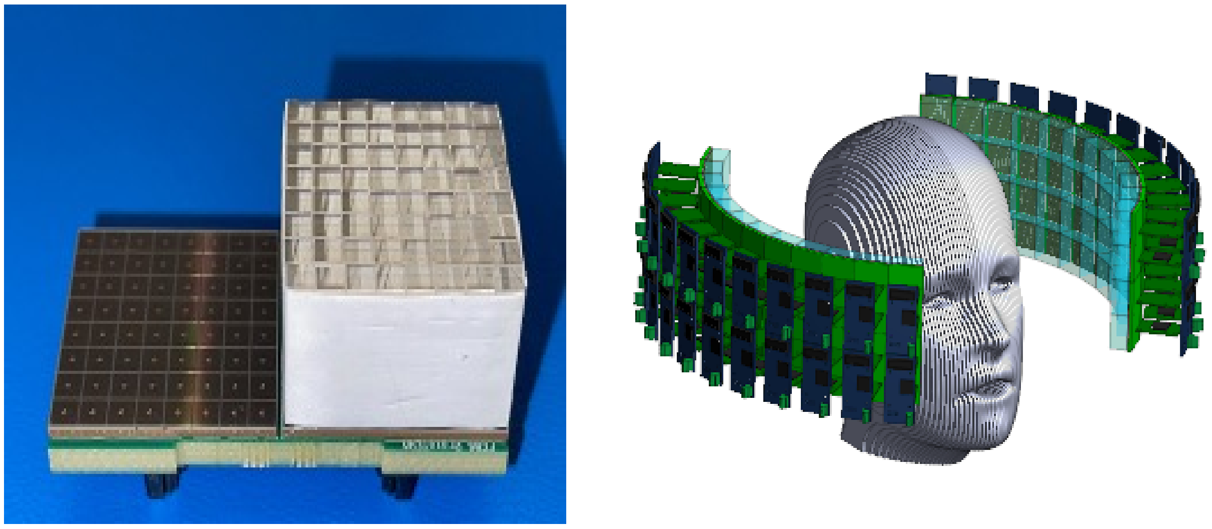

Figure 1: On the left, a single PET module with half of the SiPM pixels exposed and the other half coupled to a 8$\times$8 LYSO scintillator array. On the right, is a schematic of a full PET scanner composed of these modules. The images are provided courtesy of Marek Proga for the the TOF-PET for Proton Therapy consortium [2].

By configuring scanners to surround a patient, either through a cylindrical geometry or into two-halves, a PET scanner can detect the back-to-back annihilation gammas in coincidence. Each detected gamma pair forms a line-of-response (LOR), a line in space and time that connects two detector channels (single crystal-SiPM pixels). The LORs, together with detector parameters such as timing resolution, are utilized to localize and then reconstruct an image of the positron source, such as the tumor where the radioisotopes accumulated [3].

1.2 Depth of Interaction

Depth of interaction (DOI) is the depth at which a gamma ray travels into a scintillator before it deposits its energy. DOI can significantly improve PET imaging by correcting parallax error, the mispositioning of LORs that connect two back-to-back gammas from an annihilation event [4, 5]. Without DOI information, we may assume that each gamma interacted at the center of the respective crystals they entered, but physically, we know that the gammas are likely to interact at a continuum of different depths. For instance, one of the photons may deposit its energy at the front of a crystal while the other deposits its energy closer to the back of another crystal. In this case, naively drawing the LOR to connect the centers of the crystals would incorrectly localize where the annihilation event occurred. Figure 2 illustrates how depth-of-interaction affects the placement of the LOR. PET scanners with double-ended readout, SiPMs coupled to both sides of the scintillator arrays, can be utilized to extract DOI information [6]. When a gamma interacts in a crystal with a double-ended readout, one can compare the response from the SiPMs coupled to each end of the crystal to deduce the DOI. The measure of how well a system can estimate DOI is cited as its DOI resolution. The statistical analysis that can be carried out to reconstruct DOI from double readout is discussed in Section 1.3.

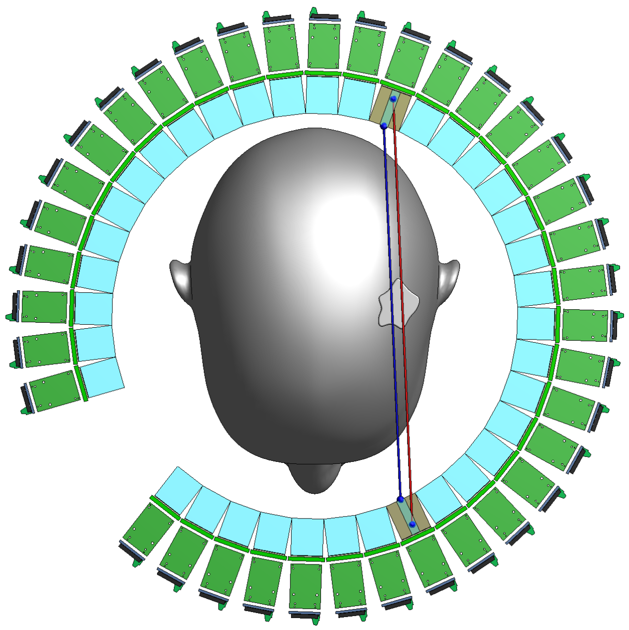

Figure 2: A diagram illustrating the importance of DOI information in properly assigning LORs. The red and blue lines illustrate how different DOI-encoding effect the placement of the LOR. The diagram is provided courtesy of Marek Proga.

1.3 Statistical Analysis

1.3.1 Energy Response

When a gamma ray enters a scintillator, it can undergo two main processes: photoelectric absorption or a Compton scatter. Photoelectric absorption occurs when a gamma is fully absorbed in a scintillator, thereby depositing its total 511 keV energy. In an energy spectrum, the photoelectric absorption of a gamma-ray appears as a Gaussian peak, known as the photopeak, centered around 511 keV. When a gamma Compton scatters, it only deposits some of its energy, manifesting as a continuous distribution known as the Compton Continuum.

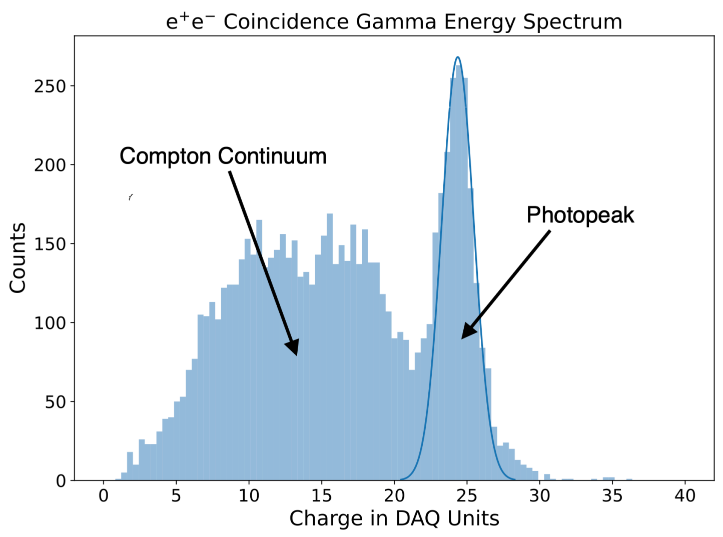

Figure 3: An example of an energy spectrum collected by one of our single PET channels. Charge on the horizontal-axis is proportional to energy.

The Compton continuum and the photopeak are labeled on an example spectra from our DOI experiments with a Gaussian fit to the peak in Figure 4. Events falling in the Compton Continuum makeup "blurred" information since one cannot easily reconstruct their trajectory from the original annihilation event [1]. For this reason, it is common to impose photopeak cuts, in which analysis and image reconstruction are only conducted with events that fall within a specified number of standard deviations inside of the Gaussian fit to the photopeak. A detector's ability to identify photoelectric events is parameterized by its energy resolution, Equation 1 [1].

$$\text{Energy Resolution} = \frac{2\sqrt{2\ln(2)}\sigma}{\mu} = \frac{\text{FWHM}}{\mu} \tag{1}$$

Where the σ is the standard deviation and μ is the mean of the Gaussian fit to the photopeak.

1.3.2 Normalized Count Differences and DOI Resolution

The importance of energy resolution and photopeak cuts for PET with DOI-encoding is in the form of normalized count differences (NCD). NCD is the common statistical method used to reconstruct DOI information from double readout PET, Equation 2 [6, 7].

$$\text{NCD} = \frac{S_1 - S_2}{S_1 + S_2} \tag{2}$$

Where S_1 is the charge readout from the SiPM on one end and S_2 is the charge from the other. Literature has shown that NCD is an effective means to statistically distinguish between different DOIs [4, 6]. Figure 4 illustrates the resultant NCD spectra at each DOI from 28 μm crystal data with a 2σ photopeak cut for a specific channel pair. The separation between the NCD peaks allow us to decipher between different DOIs and hence evince that in a clinical setting, NCD from double ended readout PET could be used to correct parallax error. The distinction between DOIs with

NCD also motivates that machine and deep learning algorithms may be able to use NCD as well as other information to predict DOI rapidly and effectively.

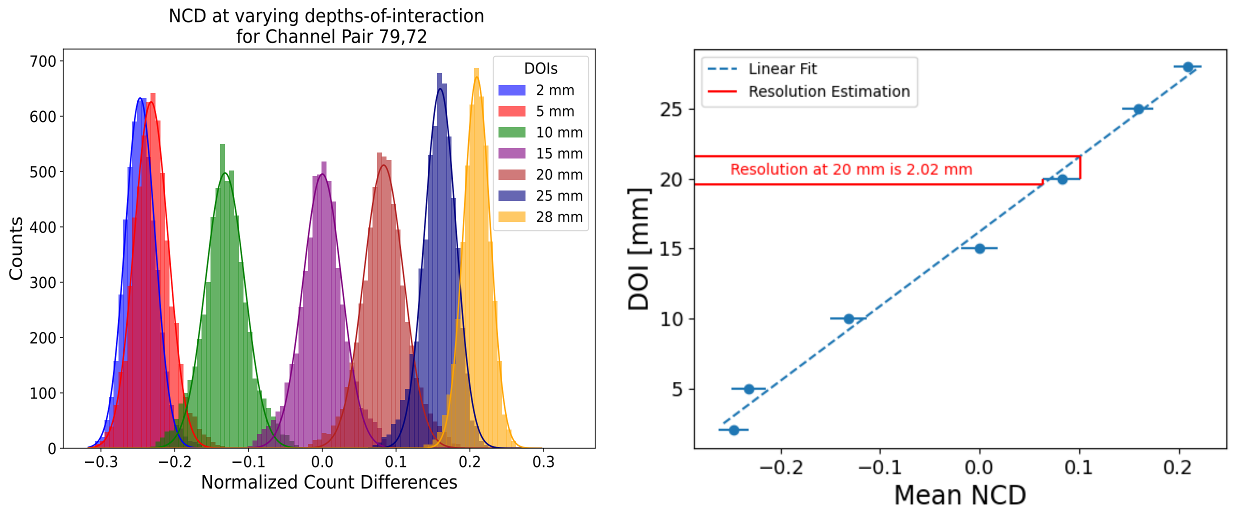

Figure 4: The left panel shows the NCD spectra at each DOI with 28 μm data for a single double-ended channel pair. The NCDs are calculated with data that falls within 2σ of the photopeak. The right panel shows the corresponding D0I vs mean NCD from the Gaussian fits. The red lines illustrate the means by which DOI resolution was determined at 20 mm DOI.

Similarly to energy resolution, we may also quote the performance of our detector's ability to estimate DOI by computing its DOI resolution. Regarding the NCD spectra shown in Figure 4, the DOI resolution can be computed by plotting the truth DOI vs NCD peak locations obtained from the Gaussian fit to the data. The horizontal error bars are then given by the full-width-half-max (FWHM) from the Gaussian fits. The distance difference from each end of the FWHM bars at a given DOI to the best linear fit is then cited as the DOI resolution at that depth. The right panel of Figure 4 illustrates an example of this methodology. Table 1 lists the DOI resolutions obtained using this method, from a not yet published paper, using the same data set we will use later in this report.

DOI (mm)	Resolution (mm)
2	2.69
5	2.99
10	3.67
15	3.68
20	3.68
25	3.02
28	2.65
Average	3.20

Table 1: DOI Resolution from 28 mum rough crystals at various DOIs (see 2.1) computed using the method illustrated in Figure 4

Another typical method to cite DOI is to histogram truth values minus the predicted values and then fit a Gaussian to the resultant distribution. The FWHM of the Gaussian is then cited as the

DOI resolution [6]. This is the method by which we will aim to report our resolutions using the predicted values from machine learning.

1.4 Motivation and Predictions

The importance of high-resolution images and the ability of double-ended PET to estimate DOI calls for rapid and efficient methods to do so. Machine learning (ML) is quickly being adopted in various fields of science to identify physical phenomena and has already been employed in other areas of PET imaging [8, 9]. With analysis of observables from double-ended PET illustrating that different DOIs can be effectively separated (e.g., via NCD), we hypothesize that machine learning can be utilized to identify DOI. With this as motivation, we report results from training categorical and regression random forest (RF) models to identify DOI from experimental data recorded from a miniature double-ended PET scanner (see 2.1). We predit that RF will accurately and efficiently estimate DOI and even improve on the resolutions cited in 1 and other leading papers. Finally, we will conclude the importance of different features, hypothesizing that NCD will be the most important feature in determining DOI following from literature, as well as processing methods and RF parameters for DOI identification.

2 Methods

2.1 Experimental Setup

A tabletop experiment was configured such that a single double-ended PET module composed of 30 mm long LYSO crystals was irradiated at known depths along the length of the crystals. A Ge-68 line source was placed in front of this double-ended module with two steel blocks between it and the module. The steel collimator ensured that annihilation gammas could only interact within a $\pm$ 0.5 mm spatial window in the crystals. The module was placed on a linear stage such that translating the module left and right allowed the collimated gammas to enter the crystals at different depths along the 30 mm length. The SiPM readout and pre-processing of the data was done using PETsys front-end electronics (FEM) [10]. This setup is summarized in Figure 5.

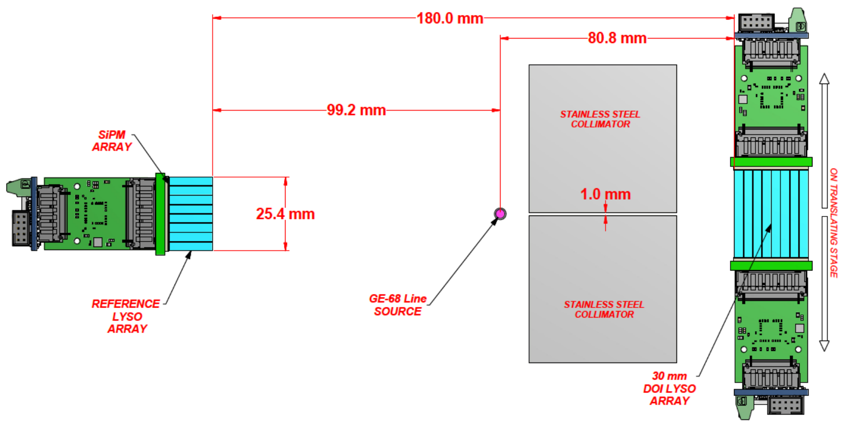

Figure 5: The experimental setup to irradiate a double-ended LYSO PET module at specific depths. This schematic is provided courtesy of Marek Proga.

The LYSO array had crystals of four types of surface treatment: 0 μm (polished), 5 μm, 14 μm, and 28 μm roughnesses. Larger μm values correspond to rougher surfaces. For each crystal type, we acquired data at seven DOIs: 2 mm, 5 mm, 10 mm, 15 mm, 20 mm, 25 mm, 28 mm. While our PET modules consist of 64 total crystals and 64 coupled SiPMs on each side, only a select few channel pairs were coupled to the rough crystals; hence, we will focus our attention on those. Prior knowledge (from a not yet published paper) illustrates that 28 μm rough crystals yield the best DOI results, so our ML training and testing will focus on 28 μm data.

2.2 Random Forest

This report will focus on utilizing random forest regression to predict DOI as a continuous variable. We will also cite some results from categorical regression predicting the seven truth DOIs as discrete variables to provide auxiliary information about the overall behavior of RF. However, for this discussion we focus on the principal behind random forest regression since this will be the primary objective. Random forest regression works on the basis of generating randomized assortments of regression trees [11, 12, 13]. Decision trees are non-parametric, hierarchical learning methods composed of three main components: root, decision, and leaf nodes. The root node marks the first split into other nodes. The decision nodes split based on a condition, for example if the the vector element x_i is less then 1, the node goes left and if it is greater than 1, it splits right. Decision nodes can split into other decision nodes or final outcomes, known as leaf nodes. Regression trees work utilizing a similar tree structure to predict continuous values. Regression trees are built by partitioning prediction space into discrete regions [12, 14]. The regions correspond to an average value of the prediction variables, $\hat{y}$, that meet binary splitting conditions based on the input variables, $\hat{x}$. The splits are chosen to reduce the training error, often given by the residual sum square (RSS), Equation 3.

$$S = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{3}$$

Where y$_i$ is a given truth value and f(x_i) is the predicted value obtained through fitting to the independent variables, x_i.

Growth of a tree ends when a stopping criterion is met such as the maximum allowed depth, the number of splits a single tree can take [14]. Finally, the random forest is generated by growing multiple regression trees, each from random samples of the training data. A final prediction by a RF model is then the average from its constituent trees [11]. Our study will focus on methods to increase the statistical relevance of experimental observables to improve the RF's ability to distinguish between DOIs rather than on the optimization of the algorithm itself. Nonetheless, brief comments about how varying the number of trees and the maximal depth impacted our performance will be noted in the results. We will use the TensorFlow python package for our random forest models [15].

2.2.1 Data Processing, Feature Importance, AND Performance Assessment

We will perform DOI-specific processing and generic data transformations to improve DOI estimation. Namely, we will cut data on the photopeak as described in Section 1.3 and explore how different photopeak cuts impact RF's performance. Secondly, we will explore feature importance and methods to transform features to be statistically better predictors of DOI. Specifically, we consider dimensionality reductions using the UMAP (Uniform Manifold Approximation & Projection) python package and z-score transformations using the sklearn processing python package [16, 17]. Other transformations were attempted, but for conciseness, we will discuss only the aforementioned transformations as they were the ones that proved to make meaningful improvements. Overall assessment of algorithm performances will be done through standard means such as accuracy in classification and root-mean-square error (RMSE) in regression (Eqaution 4), a measure of the mean difference between predicted and truth data, both of which are easily extracted from TensorFlow. Feature importance will be mainly explored through mean minimum depth, the average, smallest number of splits needed for a feature to reach a prediction, which can be obtained from TensorFlow [15, 18]. The final assessment of the RF performance for DOI predictions will be the resultant DOI resolution, which will be given by the full-width-half-maximum (FWHM) of the Gaussian fit to the truth minus the predicted value distributions. We will use pandas for data read-in, NumPy and sklearn for statistics, SciPy for curve fitting, and matplotlib for visualization [17, 19, 20, 21, 22].

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{4}$$

Where N is the sample size and y_i and $\hat{y}_i$ are the observed and expected ith data points in the sample, respectively.

3 Results and Discussion

The following data visualizations and results all stem from 28 μm roughness data. Initial trainings were conducted with the following features: left and right SiPM channel ID (ChannelIDL/ChannelIDR), left and right collected charge values (ChargeL/ChargeR), left and right detection times (TimeL/TimeR) as well as their time difference (delta_t), and normalized count differences (NCD). The parameters were all set to their defaults given by the TensorFlow package, this includes 300 trees and a maximum allowed depth per tree of 16.

3.1 The Effect of Energy Cutting

Without any energy cut imposed, accuracy of the classifier was rather poor yielding 44.36% total accuracy as illustrated by the left panel of Figure 6. The right panel of Figure 6 illustrates the distributions of predicted DOIs by RF regression, which have large, non-uniform spread and a RMSE of 7.1 mm. The poor results stem from not filtering data from the Compton continuum. We also note that both the classification and regression models disproportionately confused 2- and 5 mm DOIs. Feature importance calculations illustrate that NCD is overwhelmingly the most important feature (see Section 3.2 below). Thus following Figure 4, which shows large overlap between the 2 and 5 mm NCD Gaussian distributions, it is easy to understand why the algorithm struggles to discern the two smallest DOIs. This is likely to be an effect of the left SiPMs' low sensitivity to distinguish between charge deposition near to them, which could be improved by adjusting their gains.

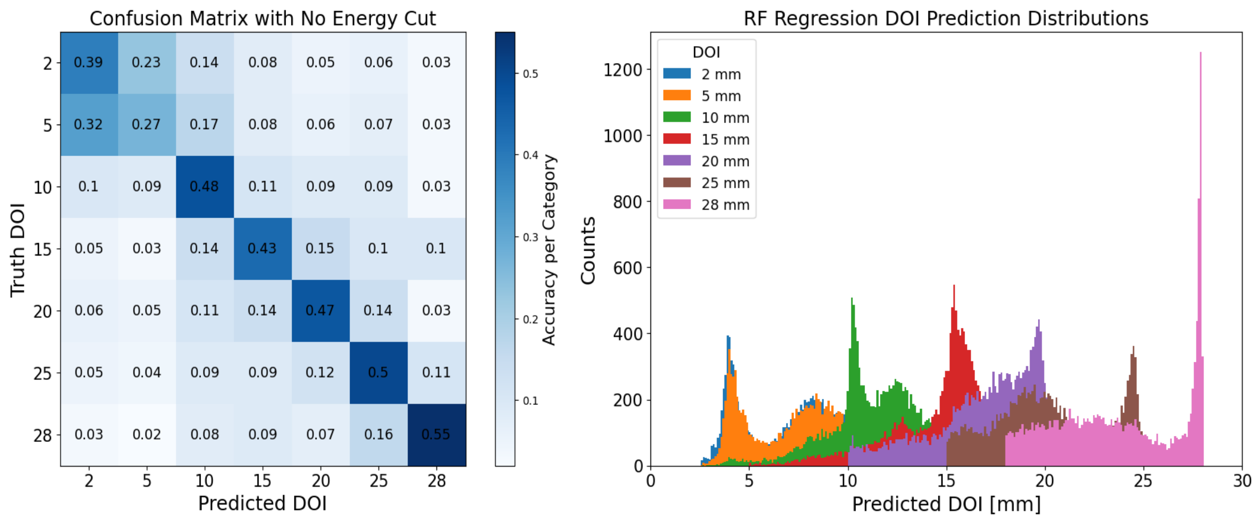

Figure 6: On the left, the confusion matrix from random forest classification on non-cut DOI data. The total classification accuracy is 44.36%. On the right, the prediction distributions per DOI achieved from random forest regression. The total RMSE is 7.1 mm. Both models illustrate why energy cuts are necessary.

Imposing an energy cut on the data immediately boosts performance. The left panel Figure 7 illustrates overall accuracy from classification and RMSE from regression models as a function of the number of standard deviations from the photopeak we cut the data. The right panel illustrates the sample size as a function of each standard deviation cut. The results show that the stricter energy cut we impose (e.g., lesser number of standard deviations from the photopeak), the more effective RF can identify DOI. However, we also see that it also dramatically reduces the sample size of data as shown in the right panel of Figure 7. Although, it is expected to lose a significant amount of statistics from a photopeak cut since it removes almost all of the Compton continuum, imposing too stringent of cuts such that we start losing information from photoelectric events should be avoided.

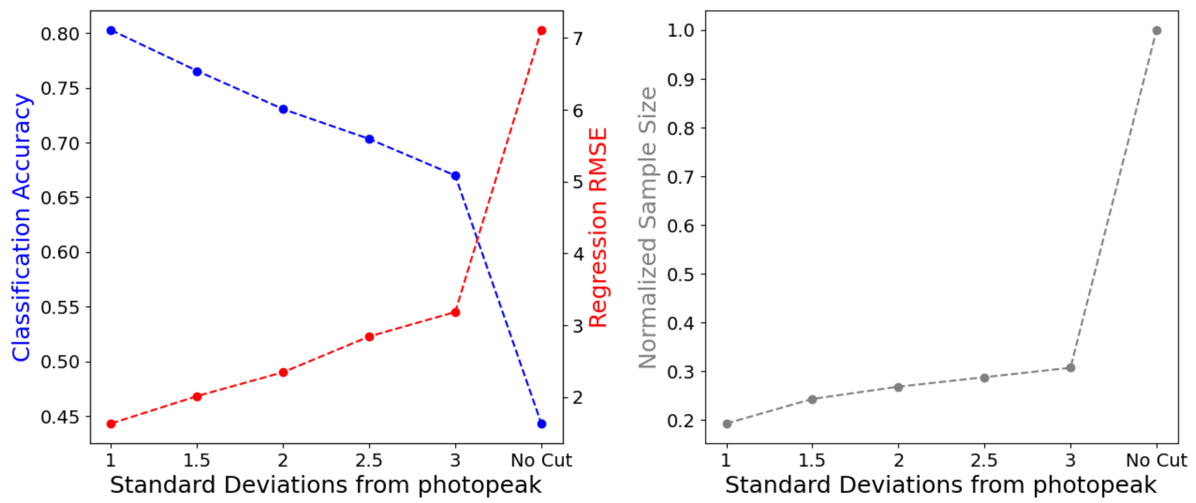

Figure 7: The left panel illustrates the classification accuracy and RMSE from regression as a function of the number standard deviations that we cut data away from the photopeak. The right panel displays the normalized number of data points in the training set after each photopeak cut.

Within a clinical setting, reduction of statistics means a patient would have to be subjected to longer PET scans, and hence longer radiation exposure times. Based on this practical concern as well as efficiency, we conclude that although accuracy continues to increase with stricter cuts, cutting beyond 2σ is likely an unreasonable requirement. For the rest of our results, we will focus on data with a 2σ photopeak cut, which yielded a 73.0% classification accuracy and RMSE of 2.31 mm. The left plot of Figure 8 is the confusion matrix for the 2σ cut, we see an overall strong accuracy per category. We note that while the accuracy amongst most categories is greatly strengthened compared to Figure 6, the RF models still disproportionately struggle with 2- and 5 mm data. In the regression predictions, we see the distributions have reduced variance, albeit still poor, compared to that from the non-cut data but are not centered well on their corresponding truth DOIs.

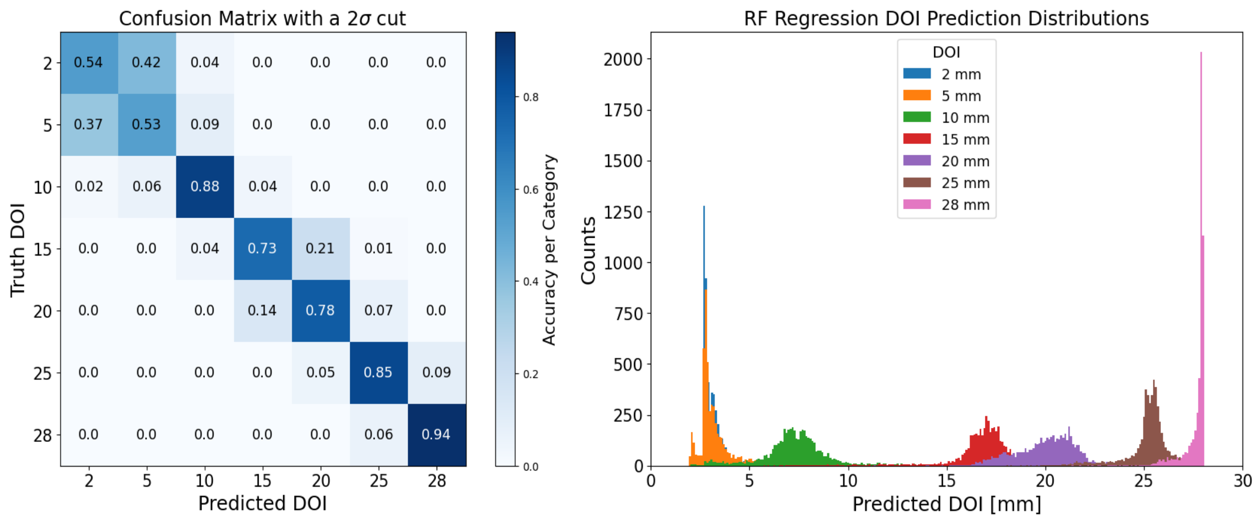

Figure 8: On the left, the confusion matrix from random forest classification on 2σ photopeak cut DOI data. The total classification accuracy is 73.0%. On the right, the prediction distributions per DOI achieved from random forest regression from the same cut data yielding total RMSE of 2.32 mm.

3.2 Feature Impact and Tested Transformations

3.2.1 Feature Importance

Both non-cut and cut data training and testing yielded the same feature importance calculated via the inverse mean minimum depth illustrated in Figure 9. The result illustrates that detection times are the least important result, and it was found that removing time, but not time difference, actually improves the result of the discrete accuracy from 72.85% to 76.05% and the RMSE from 2.32 to 1.79 mm. This stems from the time being an arbitrary numerical value, which likely posed no helpful information when building the forest. Beyond timing, mixing and matching different combinations of features did not improve regression or classification. NCD proved to be the most important feature, which agrees strongly with literature that have found NCD is the best method to estimate DOI [7]. However, observing that time difference is the third most important feature, we also draw an important conclusion that additional information is needed beyond just NCD to reliably predict DOI.

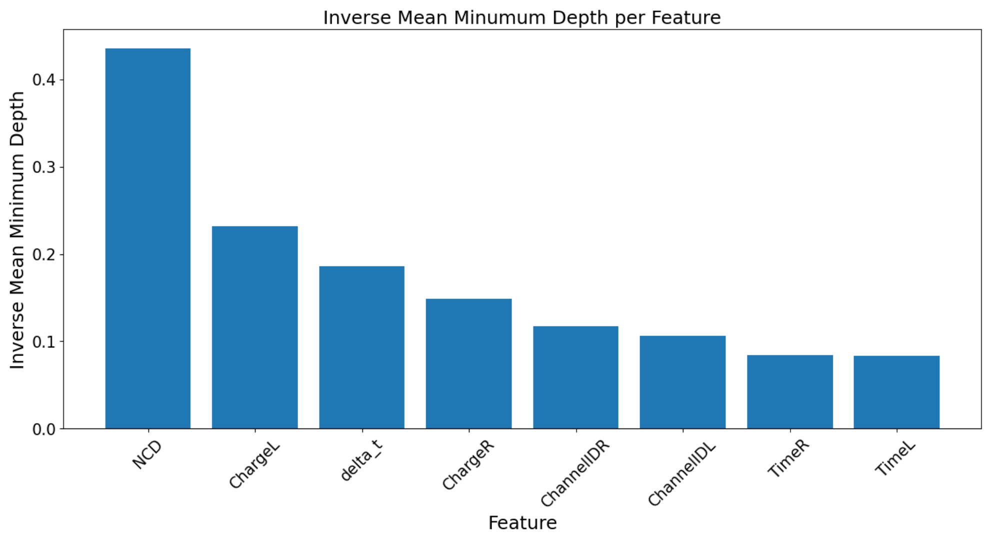

Figure 9: The inverse mean minimal depths for the original training features. The trend holds true for other common feature importance metrics as well.

3.2.2 UMAP Dimensionality Reductions

Next, attention was turned to methods to transform features to improve their importances. Based on Figure 8, we pay particular attention on methods to help improve RF's ability to distinguish between 2- and 5 mm DOI. The first attempt was using UMAP to collectively transform the data into a lower dimensional phase space. The motivation being that UMAP can cluster the data in such a way that further distinguishes between all of the DOIs. This was attempted with various dimensionality reductions (n = 2,3,4,...), metrics, and combinations of our input features. Careful attention was paid to ensure both training and testing data clustered approximately in the same way, namely by setting a stagnant random seed.

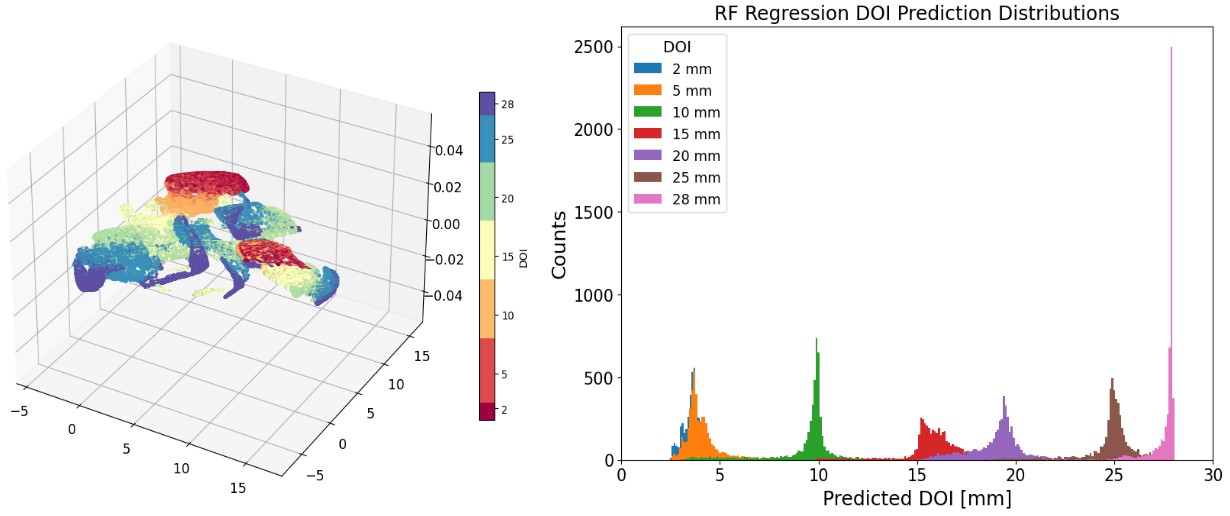

Figure 10: Dimensionality reduction of charge, NCD, and ChannelID features into a 3-dimensional phase space using UMAP with the Chebyshev metric. Training our random forest models with these embeddings increased the symmetry of the prediction distributions.

UMAP clustering helped improve the symmetry of the prediction distributions (right panel of Figure 10) but only improved the RMSE minimally to about 1.65 mm. This result was achieved by clustering Charge, ChannelID, and NCD features into 3-dimensional phase space using the Chebyshev metric (Equation 5).

$$d(\vec{x}, \hat{\vec{x}}) = \max_i |x_i - \hat{x}_i| \tag{5}$$

Where $\vec{x}$ and $\hat{\vec{x}}$ are two vectors of which we are computing the distance between. The Chebyshev metric, which yielded the best result, was chosen since it can maximize the distinguishability between DOIs due to its selection of the maximum possible difference along any coordinate direction [23]. Clustering with any time feature (time or Δt) hurt the clustering and subsequently RF training. An example of this resultant UMAP clustering is illustrated in the left panel of Figure 10.

3.2.3 The z-score Transformation

Beyond UMAP clustering, careful attention was still needed to help make the features more distinct at adjacent DOIs. Numerous methods were attempted including developing novel weighting functions similar to NCD, however, these attempts proved futile. More generic transformations such as log and z-score transformations were studied as well. The z-score transformation, particularly imposed on charge, was found to greatly improve results. A subtly to applying this transformation was that it had to be imposed on a channel-by-channel basis due to the gain differences of each SiPM. Figure 11 illustrates that after the transformation the 2- and 5 mm charge distributions are more discernible compared to the direct-observable charge distributions before the transform. Although this transform increased the distinction between the charge of adjacent DOIs, we note that optimizing the SiPMs' gains could lessen the need for such a manipulation.

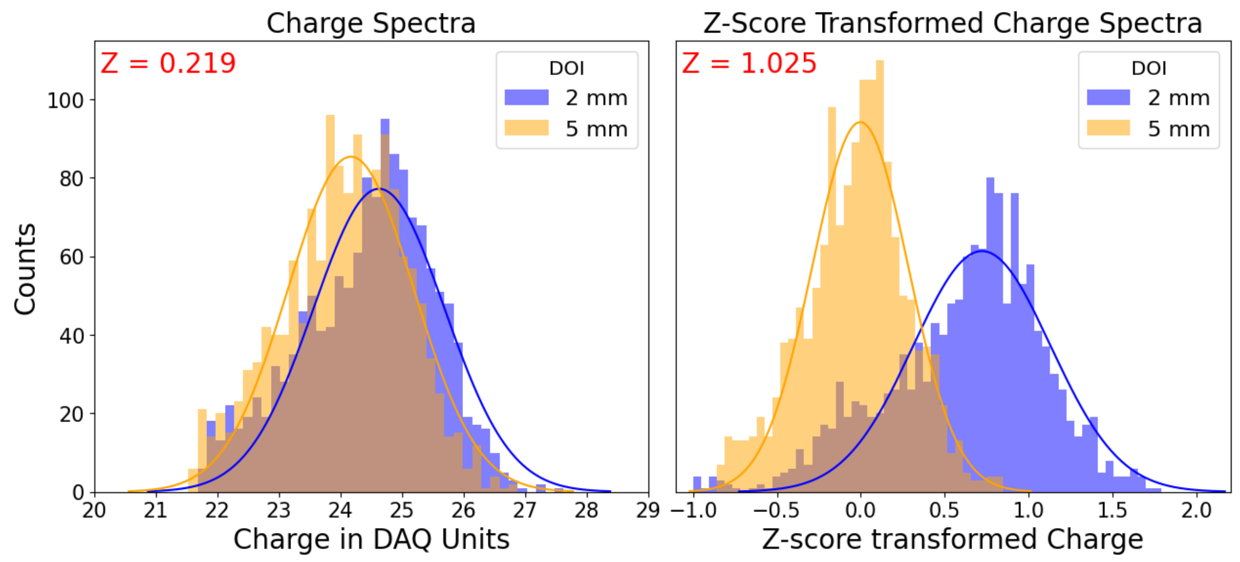

Figure 11: An example of a channel's right charge distribution for 2- and 5-mm DOIs before and after a z-score transformation. We see that after the z-score transformation, there is a more discernible difference between charge distributions in the 2- and 5-mm DOIs.

Quantitatively, we can assess how significantly two Gaussians differ from each other using the Z-test, Equation 6.

$$Z = \frac{\mu_2 - \mu_1}{\sqrt{\sigma_1^2 + \sigma_2^2}} \tag{6}$$

Where Z is the Z-test score and μ_i and σ_i are the respective means and variances of each Gaussian distribution, respectively. The larger the Z-test score, the more significantly two distributions differ. The Z-test score is shown in the top lefts of Figure 11 for the 2- and 5 mm charge distributions before and after the transformation. Including the z-score transformed charge into our training features reduced the RMSE to 1.59 mm and increased the accuracy of classification to 85.3%.

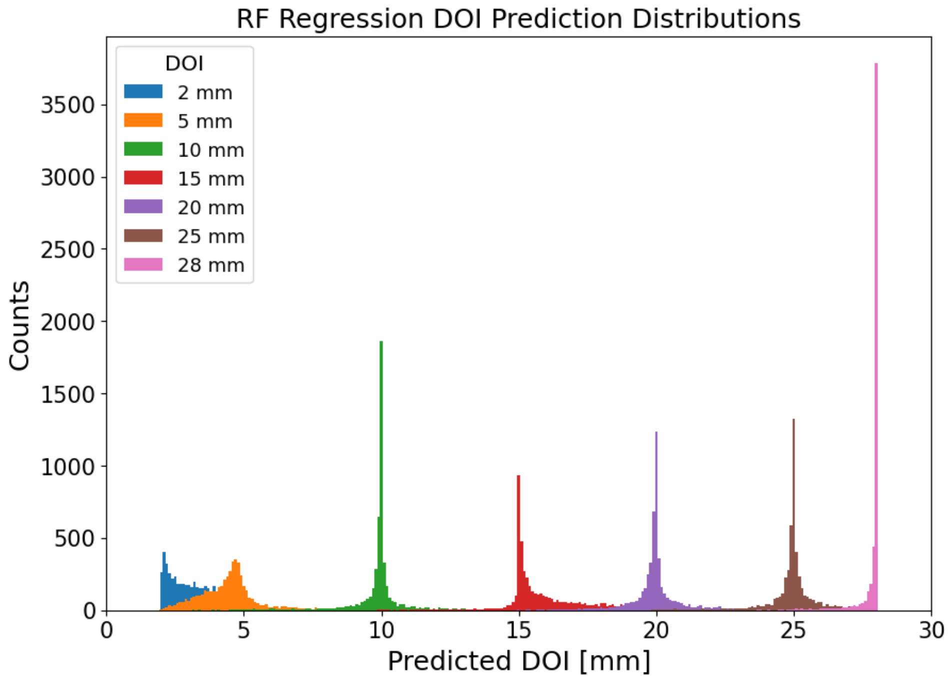

Figure 12: The final prediction distributions from our RF regression model. These distributions will be used to compute the DOI resolutions per depth.

We found that after incorporating z-score transformations, the UMAP embeddings no longer made significant contributions to the regression performance. We conclude that the model generally needed additional features to make reduce the RSS, generate better decision splits, and hence make more accurate predictions. Initially, the UMAP embeddings were able to provide additional information to better grow the regression trees. However, with the z-transformed charge added to the model, the UMAP embeddings provided no additional means to reduce the RSS and better grow the trees that the transformed charge did not already provide. The final regression model we report encompasses NCD, z-score transformed charge, time differences, and channelIDs with a maximum depth of 20 and 300 trees. Prior to changing the maximum depth from 16 to 20, the RMSE was 1.59 mm. With the maximum depth at 20, we report a final RMSE of 1.42 mm. The need for more splittings in the model illustrates that the model was not reliably partitioning prediction space with only 16 splits. Considering charge and NCD were transformed to create the other variables in the predictor space, we considered removing them to reduce mutli-colinearity across the data set but removing them negatively impacted the performance. Generally, colinearity will not hurt RF performance but can make calculations of feature importance unreliable [24]. The need to include both charge and its transform points to the model requiring many features and splits to estimate DOI, which is further shown by the need for increased depth. Interestingly, this again illustrates that the common the statistical analysis that only uses NCD to estimate DOI is neglecting other important information related to DOI.

3.3 DOI Resolution

As discussed in Section 2.2.1, DOI resolution is given by the FWHM of the Gaussian fit to the truth minus predicted distributions. This is done for every channel individually and then averaged across each DOI. As illustrated by Figure 12, we can see the distributions are not entirely Gaussian and have long outlier tails that skew toward adjacent DOIs. Moreover, we can see that the minimum and maximum DOIs, 2 and 28 mm, suffer from edge effects. This results from the model being built with the assumption that DOI cannot go below 2 mm or above 28 mm DOI and hence partitioned the prediction space on this basis. For future studies, this can be combated by taking data at the true crystal extremities, 0 and 30 mm, so that resolution can be estimated at 2 and 28 mm. Generally, tree-based learning algorithms struggle with extrapolation, so this is not a viable option to combat this issue [25]. Alternatively, we can fit half-Gaussians to 2 and 28 mm prediction distributions but for this report, we will skip citing resolutions at these DOIs, and it will handled in future studies. We cite our final average resolutions accompanied by the prediction efficiency, the percent of predictions that fell within 2.5σ of the Guassian fit to the truth minus predicted distributions. The results are displayed in Table 2.

DOI (mm)	Resolution (mm)	Efficiency (%)
5	1.635	85.5
10	0.48	73.4
15	1.55	63.3
20	0.55	70.0
25	0.52	69.0
Average	0.97	72.6

Table 2: DOI Resolution and efficiency achieved using RF regression to predict DOI. The uncertainties in the resolutions were all within 1% as calculated by the standard error multiplied by 2.355 to get uncertainty in the FWHM ($\delta x = 2.355\frac{\sigma}{\sqrt{N}}$). The average uncertainty computed by retraining this model with different seeds was below 5% for all numerical values report.

In each case, we report excellent resolutions. The commonly reported resolutions range from 2 to 4 mm [6, 7, 26]. We can see that the lowest resolutions occur at 5 and 15 mm DOIs, which follows from the predictions at these DOIs having the widest distributions and being skewed heavily towards their adjacent DOIs as shown in Figure 12. This may partially stem from SiPM pairs' gain differences and could possibly be improved in experiment. The large skewness of the 15 mm DOI predictions towards 20 mm accounts for it having the lowest efficiency as well. The other DOIs have resolutions less than 1 mm, which is illustrated by the delta-like peaks of their prediction distributions at these DOIs. Work is being continued to increase the Gaussianity and hence the efficiency of the estimated resolutions. Nonetheless, with near 70% efficiency at most DOIs, we can predict DOIs with this model at a resolution higher than most, or perhaps all, current publications.

4 Conclusion

The final DOI resolutions reported are incredibly high with an average value of 0.97 mm at 72.6% efficiency and a final RMSE of 1.42 mm. Hence our hypothesis that random forest can be used to predict DOI at high resolutions, better than the results in Table 1 as well as other published results, proved correct. However, we also hypothesized this could be done with high-efficiency which was only partially achieved. The hypothesis that NCD was the most important feature was shown to be true no matter what additional features were added to the model. This work opens a new modality of predicting depth-of-interaction in radiation detectors through random forest regression and illustrates important considerations about constructing these models. As shown by feature importances and the need for transformed data to improve the RMSE of our regression model, additional information is required beyond just NCD to reliably predict DOIs. Our final regression RF model is able to estimate DOI more than precisely enough to help significantly correct parallax error.

5 Appendix A: The Pipeline

A streamlined and comprehensive directory of python scripts was created for this machine learning project and can be found here on its respective Github. The data processing pipeline and subsequent random forest training, testing, and model assessment is described below. The results from this report were obtained from one jupyter notebook, from which the optimized python scripts discussed below were adapted. For clarity, I describe the pipeline from the individual python scripts as this is more universal than jupyter notebook. The readme.md on the homepage of the Github repository goes through the entire pipeline of the notebook.

5.1 Data Processing

The data processing pipeline includes five python scripts. MasterProcessor.py, is imported into the four other scripts, and is simply used as a place to define certain reoccurring variables names like file names for which the processed data is saved under. This file also imports the Header files, DOI_header.py and analysis_header.py, which define various functions, such as the photopeak fitting function, used throughout the processing codes. For each DOI measured and for every crystal roughness type, there is a separate data file. The first step in the pipeline is to call train_and_test.py, which reads-in every DOI file and concatenates them into two datasets, a training and a testing set. At the beginning of train_and_test.py, we define three variables: number_to_train_with, number_to_test_with, and shuffle. The first two variables are integers and define how many data points we want to sample per DOI for the training and testing data. Shuffle is a boolean and when set to True, scrambles the created training and testing data. By reading in each DOI data file through a loop, we sample the specified number of data points for each DOI for training and testing and concatenate them into their own pandas DataFrames and then finally save them into their own files.

Before we call train_and_test.py, there are no data files in the Processing directory:

```
Processing Firas$ ls
MasterProcessor.py    clean.sh      readme.md
UMAP.py          train_and_test.py      __init__.py
photopeakcut.py     z_transform.py
```

Now, setting the three variables at the beginning of the script to number_to_train_with = 50000, number_to_test_with = 20000, and shuffle = True and calling train_and_test.py:

```
Processing Firas$ python3 train_and_test.py
Original Sample Sizes:
 Training Set: 350000
 Testing Set: 140000
```

The script prints out the sample sizes of the training and testing datasets produced for reference. Now, in the directory we have two files, one for training (trainingdata_28um.csv) and one for testing (testingdata_28um.csv) our random forest:

```
Processing Firas$ ls
MasterProcessor.py  train_and_test.py    UMAP.py
photopeakcut.py     trainingdata_28um.csv
__init__.py     readme.md     z_transform.py
clean.sh      testingdata_28um.csv
Firass-MacBook-Pro:Processing Firas$
```

train_and_test.py also added two new columns to these data files not present in the original data sets, normalized count differences (NCD) and detection time difference (delta_t). Other than these two newly defined variables, the files generated by this script have no filtering done to them. The next step in the pipeline is to conduct an energy cut on the datasets. To do this, we use photopeakcut.py, which has two main functions. The first function getphotopeakcuts() is used to generate a look-up-table (LUT) of photopeak cuts. getphotopeakcuts() loops through all of the SiPM channels present in our data files, finds and fits to their respective energy spectrum photopeaks at each DOI, and then saves the lower- and upper-limit photopeak cut for each photopeak. The number of standard deviations from the mean that the two limits correspond to is given by the input parameter sigma. Once a photopeak cut LUT has been generated, we can use the next function, energycut(), to actually impose photopeak cuts on our training and testing datasets. The script by default imposes the energy cut on trainingdata_28um.csv and testingdata_28um.csv generated by train_and_test.py.

```
Processing Firas$ ls
MasterProcessor.py  train_and_test.py    UMAP.py
photopeakcut.py     trainingdata_28um.csv
__init__.py     readme.md     z_transform.py
clean.sh      testingdata_28um.csv
```

I've included a couple nice features to make this as streamline as possible. Notice that the script tells you that no LUT was found and thus it must be generated and saved. Secondly, the use of the wonderful python package tqdm allows us to view a loading bar as we impose our energy cuts. After imposing the cut, two new files are generated trainingdata_28um_?σ_cut.csv and testingdata_28um_?σ_cut.csv, where the ? indicates the value of sigma. Below is an example of running photopeakcut.py with sigma=2.

```
Processing Firas$ python3 photopeakcut.py

Trying to read-in the 28 um,2sig photopeak LUT...
```

```

A photopeak LUT has yet to generated for 28 um data with a 2sig cut.
Generating and saving the 28 um,2sig LUT:
100%|============================================| 8/8 [00:00<00:00, 92.49it/s
    ]
100%|============================================| 8/8 [00:00<00:00, 95.25it/s
    ]

Imposing a 2sig photopeak cut on training and testing datasets... this may
    take a few minutes!
100%|==================================================================|
    350000/350000 [05:04<00:00, 1149.37it/s]
100%|==================================================================|
    140000/140000 [01:46<00:00, 1310.79it/s]
Sample Sizes after the 2sig cut:
 Training Set: 93171
 Testing Set: 37319
```

The tqdm loading bars couldn't be rendered in LaTeX so I've replaced them with equal signs. It also did not like the σ or μ being printed in the code box so I've replaced them with "sig" and "u" for the purpose of the report but normally, shell would print the appropriate symbols. Notice the script prints out the training and testing sample sizes after the cut. Now with our most significant data processing done, we can move onto "optional" data transformation scripts, namely using UMAP.py or z_transform.py, both of which by default read-in the energy cut data (so trainingdata_28um_?σ_cut.csv and testingdata_28um_?σ_cut.csv) and then re-save the files with new columns of the transformed data. Of course, a simple change of variable name would allow us to also carry out the same transformations on unfiltered data. Before running any of our optional transformation scripts, our two energy cut files have the following columns:

```
Processing Firas$ python3
Python 3.9.6 (default, May  7 2023, 23:32:44)
[Clang 14.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> df = pd.read_csv("trainingdata_28um_2 _cut.csv")
>>> df.columns
Index(['TimeL', 'ChargeL', 'ChannelIDL', 'TimeR', 'ChargeR', 'ChannelIDR',
       'DOI', 'NCD', 'delta_t'],
      dtype='object')
```

Now, let's run UMAP.py. UMAP.py starts by allowing us to choose which features we want to cluster with UMAP, which by default is set to what was used in our report (and found to yield the best result): charge, NCD, and channelID. Then it actually calls upon umap.UMAP() given by the UMAP package with a few preset parameters: metric="chebyshev" and n_components = 3, again the same parameters our studies illustrated to yield the best results (see Section 3.2.2) [16]. UMAP.py also gives one the option to plot the clustering with the boolean variable, show_projection.

```
Processing Firas$ python3 UMAP.py
Fitting our data to
```

```
UMAP(n_components=3, output_metric='chebyshev', random_state=42, verbose=True)
Sun Nov 26 10:03:04 2023 Construct fuzzy simplicial set
Sun Nov 26 10:03:04 2023 Finding Nearest Neighbors
Sun Nov 26 10:03:04 2023 Building RP forest with 20 trees
Sun Nov 26 10:03:06 2023 NN descent for 17 iterations
   1  /  17
   2  /  17
  Stopping threshold met -- exiting after 2 iterations
Sun Nov 26 10:03:18 2023 Finished Nearest Neighbor Search
Sun Nov 26 10:03:20 2023 Construct embedding
Epochs completed:
    100%|====================================================|200/200 [00:45]
Sun Nov 26 10:04:18 2023 Finished embedding

Carrying out the same projection on the training data:
Sun Nov 26 10:04:55 2023 Worst tree score: 0.99050134
Sun Nov 26 10:04:55 2023 Mean tree score: 0.99159985
Sun Nov 26 10:04:55 2023 Best tree score: 0.99242253
Sun Nov 26 10:04:56 2023 Forward diversification reduced edges from 1397565 to
     379863
Sun Nov 26 10:04:57 2023 Reverse diversification reduced edges from 379863 to
    379863
Sun Nov 26 10:04:58 2023 Degree pruning reduced edges from 327222 to 327222
Sun Nov 26 10:04:58 2023 Resorting data and graph based on tree order
Sun Nov 26 10:04:58 2023 Building and compiling search function
Epochs completed:
    100%|====================================================|30/30 [00:02]
```

The text shown above are all a result of having verbose=True in umap.UMAP(). With show_projection = True, we also see a resultant clustering plot similar to the left panel of Figure 10. Now after runnning UMAP.py, we have new transformed columns in our datasets:

```
Processing Firas$ python3
Python 3.9.6 (default, May  7 2023, 23:32:44)
[Clang 14.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> df = pd.read_csv("trainingdata_28um_2 _cut.csv")
>>> df.columns
Index(['TimeL', 'ChargeL', 'ChannelIDL', 'TimeR', 'ChargeR', 'ChannelIDR',
       'DOI', 'NCD', 'delta_t', 'n1', 'n2', 'n3'],
      dtype='object')
```

The n1, n2, n2 represent our new phase space variables from the projection. We can also use z_transform.py to carry out a z-score transformation of our data. Using the function ztransform(), which itself calls upon scipy.stats.zscore, we can z-score transform any column of data we want from a pandas DataFrame [19, 21]. The script by default carries out this transform only on charge as this was proven to yield promising results.

```
Processing Firas$ python3 z_transform.py
Saving z-transformed data to files: trainingdata_28um_2 _cut.csv &
    testingdata_28um_2 _cut.csv
```

And again, these transformed features are added to the energy-cut data files:

```
Processing Firas$ python3
Python 3.9.6 (default, May  7 2023, 23:32:44)
[Clang 14.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> df = pd.read_csv("trainingdata_28um_2 _cut.csv")
>>> df.columns
Index(['TimeL', 'ChargeL', 'ChannelIDL', 'TimeR', 'ChargeR', 'ChannelIDR',
       'DOI', 'NCD', 'delta_t', 'n1', 'n2', 'n3', 'ChargeL_zscore',
       'ChargeR_zscore'],
      dtype='object')
```

5.2 Machine Learning Pipeline

Once, we have ran the required scripts in the Processing directory and, if desired, the two optional scripts, we can move onto machine learning! Since our machine learning is still at the research stage, the learning pipeline is set to train and test in the same scripts. Work is currently being done to streamline this directory such that saved random forest models can be applied to new datasets. For now the repository contains five main scripts. As with the Processing directory, this file has a “Master“ file that only contains variables that will be reused throughout the ML pipeline such as the features we wish to implement into our models. Next, we have RFassessment.py, which contains functions to assess how our classification and regression random forest models are performing. This includes functions to plot the confusion matrix, prediction distributions, and feature importance calculations. RFassessment.py is not called on its own but rather is imported into our actual machine learning scripts: RFclassification and RFregression.py which both give options to show assessment plots after testing. RFclassification and RFregression.py actually train and test random forest models using the data files generated in the Processing directory with the features defined in MasterLearning.py. The last script, DOI-resolution.py, uses the results saved from RFregression.py to fit Gaussians to the truth minus predicted distributions and then extract DOI resolutions. Let’s start with RFclassification. In MasterLearning.py we have set:

```
features = ["NCD","ChargeR","ChargeL","ChargeR_zscore","ChargeL_zscore","
   delta_t","ChannelIDL","ChannelIDR",'DOI']
```

RFclassification.py first calls tfdf.keras.pd_dataframe_to_tf_dataset() to convert our training and testing data files from pandas DataFrames to keras frames, which is more optimized for use with TensorFlow. After these are defined, the script creates a random forest model using TensorFlow: tfdf.keras.RandomForestModel() with certain parameters set to default according to the results of this report. With the parameter verbose=2, while the model is being trained, many statements are printed out such as the accuracy per tree. After the model has been trained, the script calls the TensorFlow functions model.compile(), model.evaluate(), and model.predict() to evaluate our model and then use it to predict the DOIs from our testing data. Ignoring the verbosity output by TensorFlow, when we call RFclassification.py we see:

```
Machine-Learning Firas$ python3 RFclassification.py
Training the model:

.
. this is where TensorFlow would print out information about the RF being
   built
.

Evaluating our model:
36/36 [==============================] - 1s 27ms/step - loss: 0.0000e+00 -
   Accuracy: 0.8209
36/36 [==============================] - 1s 27ms/step
loss: 0.0000
Accuracy: 0.8209

Accuracy for 2 mm DOI: 0.707
Accuracy for 5 mm DOI: 0.717
Accuracy for 10 mm DOI: 0.92
Accuracy for 15 mm DOI: 0.74
Accuracy for 20 mm DOI: 0.85
Accuracy for 25 mm DOI: 0.87
Accuracy for 28 mm DOI: 0.96
Total Accuracy:  0.824
```

RFclassification has two boolean variables defined: feature_importance and confusion_matrix. When feature_importance is set to True, the script calls the function getFeatureImportance() from RFassessment.py, which by default computes the mean inverse minimum depth, and will then show a plot like in Figure 9. With confusion_matrix set to True, the script calls ConfusionMatrix() from RFassessment.py to plot a confusion matrix just like in Figures 6 and 8. RFregression.py follows a very similar setup to RFclassification with task=tfdf.keras.Task.REGRESSION in the tfdf.keras.RandomForestModel() function to grow a regression random forest. An additional feature of this script is that it saves the prediction results to a file named regressionResults_28um.csv to estimate DOI resolution later on.

```
Machine-Learning Firas$ python3 RFregression.py
Training the model:

.
. this is where TensorFlow would print out information about the RF being
   built
.

Evaluating our model:
36/36 [==============================] - 1s 30ms/step - loss: 0.0000e+00 - mse
   : 1.9671
{'loss': 0.0, 'mse': 1.9671297073364258}

MSE: 1.9671297073364258
RMSE: 1.4025440126200768

Testing our model:
```

```
38/38 [==============================] - 2s 47ms/step

Saving regression results to file: regressionResults_28um.csv
```

As with RFclassification, RFregression also has two boolean variables we can set equal to True to help visualize the performance of our model. The first is feature_importance, which serves the same purpose as in classification. The second is prediction_spectra, which when set to True, will display a plot like in the right panels of Figures 6 and 8. Finally, after running RFregression.py, we will have a file named regressionResults_28um.csv. With this file generated, we can run DOI-resolution.py to estimate the average DOI resolutions per DOI. DOI-resolution.py has the boolean omit_2_28 = True because as noted in the report, for now, we do not cite the 2 and 28 mm DOI resolutions due to the edge effects.

```
Machine-Learning Firas$ python3 DOI-resolution.py

Average Resolution at 5 mm DOI: 1.83 mm with 88.7% effciency
Average Resolution at 10 mm DOI: 0.476 mm with 73.4% effciency
Average Resolution at 15 mm DOI: 1.554 mm with 63.3% effciency
Average Resolution at 20 mm DOI: 0.557 mm with 70.0% effciency
Average Resolution at 25 mm DOI: 0.467 mm with 67.7% effcienc
```

6 Appendix B: Photopeak Fitting

Here, we discuss the most key function written for this project, photopeak fitting, photopeakFit(), from analysis_header.py as well as the key function it calls upon scipy.optimize.curve_fit [21]. The function photopeakFit() is shown below:

```python
def photopeakFit(energy,bins,std_guess = 2,photopeakcut = 2):

    y,x = np.histogram(energy,bins[0],(bins[1],bins[2]))
    centers = (x[:-1] + x[1:]) / 2

    # this helps isolate the photopeak making it easier for the fitter to find
     the peak
    # Doing this is really only necessary for spectra with large compton edges
    , which in the case of rough crystals is true

    if std_guess == None:
        energy_temp = energy
    else:
        fitcut = centers[np.where(y == max(y))[0][0]] - std_guess
        energy_temp = energy[energy >= fitcut]

    # redefine as we will fit to this cut data
    y,x = np.histogram(energy_temp,bins[0],(bins[1],bins[2]))
    centers = (x[:-1] + x[1:]) / 2

    # guess where the photopeak lies, we can do this systematically
    guess = [max(y),centers[np.where(y == max(y))[0][0]],np.std(energy_temp)]
    try:
        p,c = curve_fit(gaussian,centers,y,p0=guess)
```

```
23         photopeak_counts = energy[(energy >= p[1] - photopeakcut*p[2]) & (
   energy <= p[1] + photopeakcut*p[2])]
24     except:
25         p = [-1,-1,-1]
26         photopeak_counts = np.array([])
27         print('Fit Failed')
28 
29     return p,photopeak_counts
```

The function photopeakFit() takes parameters energy (e.g., the array of charge values collected by a specific photopeak at a specific DOI), the bins for the energy spectrum, std_guess which we will describe in further detail momentarily, and photopeakcut, the number of standard deviations from the photopeak we want to cut at. The script starts by using numpy.histogram to histogram the energy. numpy.histogram returns the bin heights, defined as y in this function, and the bin edges, defined as x [20]. Right after, we define the centers of the bins, which is better used for fitting than the edges. Now, std_guess is an idea I came up with to help the fitter locate the photopeak in “noisy“ data, such as energy spectra with large Compton edges. Sometimes the curve_fit, will have a hard time locating the photopeak and will try to poorly fit across both the continuum and peak instead of just the peak as desired. To combat this, std_guess, which should roughly be an estimate of the actual photopeak's σ, is used to temporarily cut the data to the left of the photopeak and define energy_temp. This temporary energy is then fed to fitter instead of the entire spectrum. We then call upon scipy.optimize.curve_fit() a non-linear least squares fitter function to carry out the fit, which takes in required parameters: f, xdata, ydata [21]. f being the function we are fitting to and xdata and ydata being the data we are fitting with. Here, we input f=gaussian(), which is a gaussian function defined in analysis_header.py, xdata=centers, and ydata=y. The optional parameter p0 serves as a guess for the fitter. We compute an appropriate guess using numpy as in line 20. scipy.optimize.curve_fit() returns two variables, the fit parameters, we define as p, and the covariance matrix, that we define as c. Hence p contains the amplitude/coefficient, the mean, and the standard deviation of the Gaussian fit. photopeakFit() then return p as well as the data that falls within the photopeak according the value of photopeakcut.

References

[1] D. L. Bailey, J. S. Karp, and S. Surti, "Physics and instrumentation in pet," in *Positron emission tomography: basic sciences*, pp. 13–39, Springer, 2005.

[2] "Tof-pet for proton therapy (tppt) – in-beam time-of-fligh (tof) positron emission tomography (pet) for proton radiation therapy." TOF-PET for Proton Therapy (TPPT) – In-beam Time-of-Fligh (TOF) Positron Emission Tomography (PET) for proton radiation therapy.

[3] S. Tong, A. M. Alessio, and P. E. Kinahan, "Image reconstruction for pet/ct scanners: past achievements and future challenges," *Imaging in medicine*, vol. 2, no. 5, p. 529, 2010.

[4] L. MacDonald and M. Dahlbom, "Parallax correction in pet using depth of interaction information," *IEEE Transactions on Nuclear Science*, vol. 45, no. 4, pp. 2232–2237, 1998.

[5] I. Mohammadi, I. F. C. Castro, P. M. M. Correia, A. L. M. Silva, and J. F. C. A. Veloso, "Minimization of parallax error in positron emission tomography using depth of interaction capable detectors: methods and apparatus," *Biomedical Physics Engineering Express*, vol. 5, p. 062001, oct 2019.

[6] C. Layden, K. Klein, W. Matava, A. Sadam, F. Abouzahr, M. Proga, S. Majewski, J. Nuyts, and K. Lang, "Design and modeling of a high resolution and high sensitivity pet brain scanner with double-ended readout," *Biomedical Physics & Engineering Express*, vol. 8, no. 2, p. 025011, 2022.

[7] Z. Liu, M. Niu, Z. Kuang, N. Ren, S. Wu, L. Cong, X. Wang, Z. Sang, C. Williams, and Y. Yang, "High resolution detectors for whole-body pet scanners by using dual-ended readout," *EJNMMI physics*, vol. 9, no. 1, pp. 1–18, 2022.

[8] T. Wang, Y. Lei, Y. Fu, W. J. Curran, T. Liu, and X. Yang, "Machine learning in quantitative pet imaging," *arXiv preprint arXiv:2001.06597*, 2020.

[9] K. Gong, E. Berg, S. R. Cherry, and J. Qi, "Machine learning in pet: from photon detection to quantitative image reconstruction," *Proceedings of the IEEE*, vol. 108, no. 1, pp. 51–68, 2019.

[10] "Petsys electronics s.a.." Taguspark, Ed. Tecnologia I, 24 and 26, 2740-257, Portugal.

[11] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.

[12] C. M. University, "Random forests."

[13] M. R. Segal, "Machine learning benchmarks and random forest regression," 2004.

[14] manning college of information amp; computer sciences., "Trees."

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke,

V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020.

[17] 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," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] C. X. M. A. J. L. M. L. M. S. a. K. Ishwaran, H., "Minimal depth, fast unified random forests with randomforestsrc," 2023.

[19] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[21] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[22] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[23] S. Yang, G. Xu, H. Meng, and M. Wang, "Progressive neighbors pursuit for radar images classification," *Applied Soft Computing*, vol. 109, p. 107194, 2021.

[24] L. Toloşi and T. Lengauer, "Classification with correlated features: unreliability of feature ranking and solutions," *Bioinformatics*, vol. 27, no. 14, pp. 1986–1994, 2011.

[25] W.-Y. Loh, C.-W. Chen, and W. Zheng, "Extrapolation errors in linear model trees," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 2, pp. 6–es, 2007.

[26] X. He, C. Trigila, G. Ariño-Estrada, and E. Roncali, "Potential of depth-of-interaction-based detection time correction in cherenkov emitter crystals for tof-pet," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 7, no. 3, pp. 233–240, 2022.

7 Reflection

This report illustrates final results I obtained for predicting DOI using random forest with the data transformations and processing techniques that were successful. What this report does not show is the countless hours I explored other possible ways to improve the machine learning result. Although frustrating at times, this work was incredibly fun and enlightening. In reality, I learned more about statistics, machine learning, and physics during my failed attempts then I did from the few that worked.

I am very grateful to the SDS department for providing this opportunity to embark on a computational research project. Although my research has always involved computational tasks like simulation and analysis, they have always been auxiliary tasks to complete a physics goal. This was the first project of mine that focused on the computational side. I look forward to continuing research like this in my graduate endeavors as a physicist, where machine learning is becoming more and more important to identify physical phenomena. So truly this research has given me an awesome opportunity to start learning about machine learning early on in my career as a scientist. This project has also lead to conversations with my professor about continuing this project and building it into a publication considering we have obtained excellent results thus far.

I like to think I am pretty well-versed in my general coding skills and very well-versed in my knowledge of physics. I also learned quite a bit about computing and numerical methods during the coursework to earn the Scientific Computation and Data Science certificate. However, trying to understand how random forest works was definitely a challenge for me and something I am continuing to read about as I think towards the future of a possible publication. This was definitely the most difficult part of this entire report even though my explanation of random forest only spanned a paragraph, I read tens of papers to understand how RF works, what its feature importances mean, and how trees are grown to better improve the results.

Another wonderful outcome of this research is that it has forced me to become very organized with my codes and allowed me to become proficient with git. Prior to this project, many of my codes were sound in function but disorganized and not commented. Knowing that this project needed to have a clean and streamline pipeline as well as easy readability and reproducibility, I became very organized with my codes and maintained a nice Github (which looks awesome on my resume as well!). This has now been instilled in me as habit and on a completely different project, I have now begun to maintain a very organized Github for it as well.

In short this project helped me learn how to implement and optimize random forests and how different statistical transformations change data and when to use them for machine learning. I was also able to refine my coding skills, having fully coded and organized an entire code repository myself.