2020

A Machine Learning Approach to Estimate the Annihilation Photon Interactions Inside the Scintillator of a PET Scanner

Sai Akhil Bharthavarapu

West Virginia University, sb0168@mix.wvu.edu

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Part of the [Computer Sciences Commons](#), and the [Data Science Commons](#)

Recommended Citation

Bharthavarapu, Sai Akhil, "A Machine Learning Approach to Estimate the Annihilation Photon Interactions Inside the Scintillator of a PET Scanner" (2020). *Graduate Theses, Dissertations, and Problem Reports*. 7849.

<https://researchrepository.wvu.edu/etd/7849>

This Problem/Project Report is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Problem/Project Report in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Problem/Project Report has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

A Machine Learning Approach to Estimate the Annihilation Photon Interactions Inside the Scintillator of a PET Scanner

Sai Akhil Bharthavarapu

Problem Report submitted to the
Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Gianfranco Doretto, Ph.D., Chair
Donald Adjero, Ph.D.
Alexander V Stolin, Ph.D.
Raymond R. Raylman, Ph.D.
Powsiri Klinkhachorn, Ph.D.

Lane Department of Computer Science and Electrical Engineer-
ing
West Virginia University
Morgantown, WV
November 30, 2020

Keywords: PET Scanner, Machine learning, Deep learning, Positrons
Copyright 2020 Sai Akhil Bharthavarapu

ABSTRACT

A Machine Learning Approach to Estimate the Annihilation Photon Interactions Inside the Scintillator of a PET Scanner

Sai Akhil Bharthavarapu

Biochemical processes are chemical processes that occur in living organisms. They can be studied with nuclear medicine through the help of radioactive tracers. Based on the radioisotope used, the photons that are emitted from the body tissue are either detected by single-photon emission computed tomography (SPECT) or by positron emission tomography (PET) scanners. SPECT uses gamma rays as tracer but gives a weaker contrast and spatial resolution compared to a PET scanner which uses positrons as tracer. PET scans show the metabolic changes occurring at the cellular level in an organ or a tissue. This detection is important because diseases begin at the cellular level and PET scans can detect these changes at a very early stage. To detect these changes machine learning plays an important role, mainly in sensing the position of positron emissions from the tissue.

In this work, we first generated a dataset of images representing the photon distribution on the PET photodetectors, using the Geant4 Application for Tomographic Emission (GATE) simulation software package and the DETECT2000 software environment. Second, we designed and developed a fully connected and two different convolutional neural network models to correlate the depth of interaction (DOI) with the shape of the photon distribution on the photodetectors, and to detect the position of the annihilation photon interactions inside the PET scanner. Experimental results show that the top performing network can detect the annihilation photon interactions or event positions with higher accuracy than current methods, on a very large dataset.

Acknowledgements

Firstly, I would like to thank my family, especially to my mother - Sudha Gandhi Bharthavarapu, father - Durga Prasad Bharthavarapu and my sister - Meghana Bharthavarapu, for all the moral and financial support, which drove me all the way to continue my studies. Their love and sacrifices are the main reasons which made it possible to continue my higher studies abroad, for which I'm very grateful to them.

I express my deepest gratitude to my research advisor, Dr. Gianfranco Doretto, for his invaluable assistance, support and encouragement throughout my research. This thesis would not have been possible without his intellectual support and the resources he made available to me. I'm grateful to him for believing me and providing me an opportunity to work in his enthusiastic research group and making my Master's program a memorable experience. I would also like to thank my committee members, Dr. Ray Raylman, Dr. Alexander Stolin, Dr. Adjeroth, and Dr. Powsiri Klinkhachorn for their valuable advices and support that assisted me in betterment of my work.

I would like to thank Dr. Brian Powell for providing me an opportunity to work as a Graduate Teaching Assistant in the course CS101. I would also like to thank the Lane Department of Computer Science and Electrical Engineering for giving me the opportunity to pursue my Master's degree in such an academically vibrant environment. I would also like to thank all my lab mates, especially Stanislav Pidhorskyi, graduate student at WVU, who has helped me a lot in my problem report work. Thanks would not be sufficient for all the help they offered me.

I would like to specially thank Sai Pushpitha Vudata for being of constant support and for always believing in me. She was always there for me and continuously provided encouragement and was always willing and enthusiastic to assist in anyway she could throughout my journey at WVU. I admire the way she keeps the environment lively with positive energy all the time. I thank Hemanth Bommisetty for sharing knowledge on the common interests and also for entertaining our group. I really enjoyed their company which puts a smile on my face all the time. I would also like to thank Sujana Kasani for guiding me throughout the journey.

I wouldn't imagine my life at WVU without these people - Sai Satish Guda, Vivek Komarina, Surekha Pachipulusu, Pratyusha Vakkantula, Ranaa Mahveen, RamPinni, Sashank Jammalamadaka, Ashish Maradapa, Pavan Koppula, Akash Mehta, Savan Suri, Nihaarika Alahari, Anudeep Paraitham, Sahiti Guda, Sasanka Katreddi, who showered unlimited love and affection on me. I never forget the dinners and game nights that has put us all together as a family.

Last but not the least, I would like to thank my undergrad friends - Dileep Chintalapudi, Pranay Vasana, Rahul Guddeti, Dinesh Power, Krupa Kiran Jasti, Gowtham Pancharla, Puneeth Valmiki, Ramesh Korva, Vallabha Bava, Prudhvi Gutta, Viswa Charan Maddali, Varun Chintala, for their encouragement. Any omission in this acknowledgement does not mean lack of gratitude.

Contents

Chapter 1: Introduction	1
1.1 Problem and Motivation	1
1.2 Thesis Contributions	2
1.3 Thesis Outline	2
Chapter 2: Literature Review	3
2.1 Design of PET Scanners	3
2.2 Estimating Event Positions using Deep Learning	4
Chapter 3: Data Generation	6
3.1 Detector Geometry	6
3.2 Dataset	10
Chapter 4: Network Architectures	13
4.1 Architecture - 1	13
4.1.1 Batch Normalization	13
4.1.2 Activation Function (RELU)	14
4.1.3 Loss Function (MSE)	15
4.1.4 Stochastic Gradient Descent	16
4.1.5 Dropout	16
4.2 Architecture – 2	18
4.2.1 Convolution Layer	18
4.2.2 Pooling Layer	21
4.2.3 Fully Connected Layer	21
4.3 Architecture - 3	23
Chapter 5: Results and Evaluation	26
5.1 Data Distribution	26
5.2 Results with Architecture – 1	29

5.2.1 Experiment-1	30
5.2.2 Experiment - 2	33
5.2.3 Experiment - 3	37
5.3 Results with Architecture – 2	42
5.3.1 Experiment - 1	42
5.3.2 Experiment - 2	46
5.3.3 Experiment - 3	50
5.4 Results with Architecture - 3	55
5.4.1 Distribution Plots of Errors on the Validation Set	55
5.4.2 Distribution Plots of Absolute Errors on Validation Set	59
Chapter 6: Conclusions and Future Work	65

List of Figures

1	Schematic diagram of the AnnPET Scanner [1].	6
2	FlowChart of Photon Transport.	7
3	x - y Distribution Plot of Generated Data.	8
4	x - z Distribution Plot of Generated Data.	9
5	x - y Distribution Plot of HSC Data.	10
6	x - z Distribution Plot of HSC Data.	11
7	2D Image of the light distribution.	12
8	Graphical representation of a ReLU function.	15
9	Structure of Architecture-1.	17
10	Description of Architecture-1.	18
11	Example of convolutional operation with a kernel of 3×3 , no padding and stride 1.	19
12	Image after zero padding of size $P = 2$	20
13	Illustration of stride when applying convolution. The three colors show three adjacent positions where the kernel was applied. The stride in this case was $S = 2$	20
14	Architecture-2.	22
15	Description of Architecture-2.	23
16	Architecture-3.	25
17	Description of Architecture-3.	25
18	Distribution plot along the x coordinate of the data labels.	27
19	Distribution plot along the y coordinate of the data labels.	28
20	Distribution plot of the z coordinate of the data labels.	29
21	Architecture-1 - Experiment-1. Distribution plot of the total absolute error on validation set.	30
22	Architecture-1 - Experiment-1. Distribution plot of the absolute error along the x coordinate on the validation set.	31
23	Architecture-1 - Experiment-1. Distribution plot of the absolute error along the y coordinate on the validation set.	32

24	Architecture-1 - Experiment-1. Distribution plot of the absolute error along the z coordinate on the validation set.	33
25	Architecture-1 - Experiment-2. Distribution plot of the total absolute error on the validation set.	34
26	Architecture-1 - Experiment-2. Distribution plot of the absolute error along the x coordinate on the validation set.	35
27	Architecture-1 - Experiment-2. Distribution plot of the absolute error along the y coordinate on the validation set.	36
28	Architecture-1 - Experiment-2. Distribution plot of the absolute error along the z coordinate on the validation set.	37
29	Architecture-1 - Experiment-3. Distribution plot of the total absolute error on the validation set.	38
30	Architecture-1 - Experiment-3. Distribution plot of the absolute error along the x coordinate on the validation set.	39
31	Architecture-1 - Experiment-3. Distribution plot of the absolute error along the y coordinate on the validation set.	40
32	Architecture-1 - Experiment-3. Distribution plot of the absolute error along the z coordinate on the validation set.	41
33	Architecture-2 - Experiment-1. Distribution plot of the total absolute error on the validation set.	43
34	Architecture-2 - Experiment-1. Distribution plot of the absolute error along the x coordinate on the validation set.	44
35	Architecture-2 - Experiment-1. Distribution plot of the absolute error along the y coordinate on the validation set.	45
36	Architecture-2 - Experiment-1. Distribution plot of the absolute error along the z coordinate on the validation set.	46
37	Architecture-2 - Experiment-2. Distribution plot of the total absolute error on the validation set.	47
38	Architecture-2 - Experiment-2. Distribution plot of the absolute error along the x coordinate on the validation set.	48
39	Architecture-2 - Experiment-2. Distribution plot of the absolute error along the y coordinate on the validation set.	49
40	Architecture-2 - Experiment-2. Distribution plot of the absolute error along the z coordinate on the validation set.	50

41	Architecture-2 - Experiment-3. Distribution plot of the total absolute error on the validation set.	51
42	Architecture-2 - Experiment-3. Distribution plot of the absolute error along the x coordinate on the validation set.	52
43	Architecture-2 - Experiment-3. Distribution plot of the absolute error along the y coordinate on the validation set.	53
44	Architecture-2 - Experiment-3. Distribution plot of the absolute error along the z coordinate on the validation set.	54
45	Architecture-3. Distribution plot of the total error on the validation set. .	56
46	Architecture-3. Distribution plot of the error along the x coordinate on the validation set.	57
47	Architecture-3. Distribution plot of the error along the y coordinate on the validation set.	58
48	Architecture-3. Distribution plot of the error along the z coordinate on the validation set.	59
49	Architecture3. Distribution plot of the absolute error along the x coordinate on the validation set.	60
50	Architecture3. Distribution plot of the absolute error along the y coordinate on the validation set.	61
51	Architecture3. Distribution plot of the absolute error along the z coordinate on the validation set.	62
52	Architecture-3. Plot of the training and validation losses.	63

List of Tables

1	Architecture-1 - Experiment-1. Results with 600 nodes in hidden layer-1 and 400 nodes in hidden layer-2.	33
2	Architecture-1 - Experiment-2. Results with 400 nodes in hidden layer-1 and 300 nodes in hidden layer-2.	37
3	Architecture-1 - Experiment-3. Results with 400 nodes in hidden layer-1 and 300 nodes in hidden layer-2, after adding dropout regularization.	41
4	Architecture-1. Results with 600 nodes in hidden layer-1 and 400 nodes in hidden layer-2, after searching for the best learning rate.	42
5	Architecture-2 - Experiment-1. Summary of the average errors.	46
6	Architecture-2 - Experiment-2. Summary of the average errors.	50
7	Architecture-2 - Experiment-3. Summary of the average errors.	54
8	Architecture-2 - Experiment-2. Summary of the average errors after changing the learning rate and the learning rate scheduler.	55
9	Architecture-3. Summary of the average errors.	62
10	Comparison between the architectures on the validation set 1.	64

Chapter 1: Introduction

1.1 Problem and Motivation

Medical imaging analysis plays a crucial role in the abnormality detection in various organs like eye, lung, brain, breast, etc. Small animals like rodents are used as models for disease in biomedicine. To detect the biochemical changes in a human or in animals, radioactive tracers are used in nuclear medicine [1]. Positron Emission Tomography (PET) scan is an imaging test that allows doctors to check for diseases in your body at the cellular level unlike the Computerized Tomography (CT) scans and Magnetic Resonance Imaging (MRI) scans which can detect the changes later only after a disease alters the structure of the organs or tissues. PET scans are mostly used to inspect blood flow, oxygen intake, or the metabolism of organs and tissues. PET scans show the problems at the cellular level giving doctor the best view of complex systemic diseases. PET scans are mostly used to detect cancer, heart problems and brain disorders, including problems with the central nervous system.

Positron emitters such as ^{11}C and ^{18}F are used as radioactive tracers which are either swallowed, inhaled, or injected into a vein in your arm depending on what part of the body is being examined. After the material is injected into the subject, a positron is emitted after the radiotracer is decayed. This positron is then annihilated with an electron and produces two photons traveling in opposite directions with 511 keV energy. A PET scanner detects the line of response where the positron annihilation occurred by detecting the two photons.

Cancer cells have a higher metabolic rate than noncancerous cells. Due to this significant level of substance action, disease cells appear as brilliant spots on PET scans. PET scans reveal areas with decreased blood flow. This is because healthy tissues will take in more tracer than an unhealthy tissue that has decreased blood flow. Different colors and different degree of brightness on the scans indicates different functioning levels of the tissue.

PET scan images display bright spots where the radioactive tracer is collected. The areas of greater intensity called “hot spots” indicate where large amounts of radioactive tracer have accumulated and where there is a high level of chemical or metabolic activity. Less intense areas called “cold spots” indicate a smaller concentration of radioactive tracer and less activity.

Machine learning approaches are used to estimate the 3D positions of events by correlating the depth of interaction with the shape of the photon distribution onto the photodetectors. This is a regression task which estimates the position of the coordinates in x , y and z directions since the output in all the directions is a continuous value. We plan to tackle this task by using deep learning approaches. Generally deep learning methods improve the performance when large amounts of data for training is available. Artificial Neural Networks (ANNs) are used in the deep learning approaches which are inspired by the organization of a human brain. ANNs contains cells or neurons which performs a single operation and interact with others to make a decision. The most difficult aspect in a machine learning approach is to extract the features from a given raw input. This feature extraction has been

made easy with deep learning because neural networks are capable to discover the relevant patterns which will then contribute to produce the prediction of an output. It is well known that deep learning performs far better than traditional shallow machine learning techniques when dealing with big data. With increase in the computing power and large amount of data available to train neural networks, there has been a great development in the area of Convolutional Neural Networks (CNNs), which are suitable for processing our raw input data, which is arranged in 2D arrays, and can be treated as images. The main feature of CNNs is weight sharing. Moreover, CNNs are more efficient in terms of memory and complexity, and as we will see, they will be the most effective tool to address out regression task.

1.2 Thesis Contributions

The contributions of this problem report are summarized as follows:

- The data generation using the GATE simulation and DETECT2000 software environment.
- A fully connected and two convolutional neural network models were developed which estimates the position of the positron annihilation based on the image of the photon distribution on the photodetectors.
- Extensive evaluation and analysis of the results of all the three models.

1.3 Thesis Outline

Chapter 2 presents a detailed review of the existing literature on the topics related to this work. Chapter 3 presents a detailed description of the geometry of the PET scanner and how the data is generated using the GATE software package and using the DETECT2000 software environment. Chapter 4 describes the architectures that we used to detect the 3D position of the event in all the x , y and z directions. Chapter 5 presents the performance results of the architectures used. The produced results are compared among the architectures. Finally, Chapter 6 discusses the conclusion and future work.

Chapter 2: Literature Review

In this chapter we discuss the existing methodologies for designing a PET Scanner. This chapter also discusses the existing work for estimating the coordinates of the positron emission using machine learning techniques.

2.1 Design of PET Scanners

In biomedicine, small animals like rodents are typically used as models for diseases. Dissection of a large number of animals has been done which were of help for examining the progression of these models. This *application* has been reduced since there has been a tremendous increase in the use of high-resolution scanners applied to the imaging of small animals [2, 3, 4, 5]. Works like [2, 6, 3] have focussed on designing high-resolution PET scanners for imaging of small animals. The most important requirement in this specific application is the spatial resolution. With previous PET systems, the achievable Full Width Half Max (FWHM) resolution was 3-4 mm which is sufficient for the imaging of medium-sized animals but is inadequate for small animals. To achieve higher spatial resolutions, Bloomfield et al. [2] proposed a method which resulted in air gaps due to the positioning of the detectors in the PET scanners. The results were not as good as expected even with the decrease in the detector diameter due to the gaps at the joints of the detector elements. Lecomte et al. [6] proposed a PET scanner design which uses the avalanche photodiode (APD) to resolve the problem by using small discrete detectors. This has reduced the spatial resolution to 2 mm FWHM, making it ideal for imaging small animals with higher resolution. In both works, there are some parts of the region where there is no or much less scintillator material at the joints of the detectors. This has reduced the detection sensitivity.

James et al. [3], proposed methods to increase the spatial resolution and sensitivity by tapering the gap between the detector modules. Extra scintillation material is used for the tapered detector modules which result in a higher sensitivity of the scanner. Though, compared to the traditional detectors there was a slight increase in the sensitivity across the field of views, maximum sensitivity couldn't be achieved. The construction of such small scintillator elements requires labor and is expensive and complex. This is also the limitation of the method that was proposed in [7]. Sanchez et al. [8] made the scintillating crystals monolithic (non-pixelated) and proposed methods which mitigate the edge effects with the monolithic crystals, by giving a special trapezoidal shape to the crystals. They also designed a PET scanner which has the capability to determine the depth – of – interaction (DOI) based on the width of the distribution of the light onto the monolithic LYSO scintillation crystals. However, PET scanners with the pixelated design require a lot of small scintillating crystals which increase the cost and complexity of the scanner design. In addition to this, there is also a loss in the sensitive area when some extra material is used in arranging the blocks and rings eventually reducing the sensitivity of the scanner.

To address the problems of the cost and complexity, España et al. [9] proposed a method which exhibits a low spatial resolution at lower costs compared to the systems based on the

pixelated crystals. For rodent brain imaging, a spatial resolution of less than a millimeter is required. To achieve this spatial resolution, monolithic scintillators were used in designing a PET scanner. The geometry was very compact due to the imaging of a rat brain. Even after using the monolithic scintillator, there are few significant regions where there are some gaps at the joints of the detector elements. The sensitivity is also approximately 10 times lower than the then existing PET systems. This limitation is mainly due to the use of the very thin monolithic scintillation crystals.

The unique capability of the monolithic scintillator-based detectors is that the DOI can be correlated to the shape of the light distribution impinging upon the photodetectors. The parallax error needs to be considered when parameterizing the line of response (LOR) which leads to uncertainties in the spatial resolution. A possible solution to overcome this parallax error is to restrict the transaxial field of view of PET systems [10]. This method causes loss of efficiency. There are a number of methods proposed to determine the depth of interaction based on a pixelated scintillator. Moses et al. [11] proposed an approach which determines the depth of interaction by measuring the ratio of the scintillation light detected at the photo detectors. This method is expensive due to the use of the expensive photo detectors and their associated electronics.

Lerche et al. [12] proposed methods to determine the depth of interaction on the condition of not increasing the cost of the PET system. The reduction in cost has been achieved due to the use of the position sensitive photo multiplier tubes with large sensitive areas and continuous crystals. Due to the limitations in the computing power, [12] can't simulate the transport path of photons, which are in the order of thousands.

To address the problems with different types of the scintillator elements, van Dam et al. [13] has proposed methods to investigate the depth of interaction decoding approaches which do not require any detector modifications and prior knowledge about the light transport. This work also reported accurate measurements of depth of interaction at the center of the detector but a degraded performance was noted at the edge of the detector due to the gaps where the detector modules meet since the continuous modules are arranged in the rings.

One of the main challenges involved in a PET scanner is to determine the depth of the event position by mapping the light distribution of the photons on the scintillator. Prior works [14, 15] have proposed methods which show a strong relation between the depth of interaction and the ratio of the count of photons to the peak intensity of the light distribution. The centroids of the scintillation light distribution are used in determining the event position and the spread of the light distribution helps in defining the depth of interaction. However, the trajectories of each scintillation photon can't be simulated since it is a computationally intensive process.

2.2 Estimating Event Positions using Deep Learning

PET is an imaging technique in which a patient is injected with a positron emitting radioactive isotope. The positron that is emitted from the decay of the isotope collides with

the electron in the body tissue which produces two 511 keV photons travelling in opposite directions. A PET composed of several detector rings is used to detect these pairs of annihilation photons. Delorme et al. [16] proposed a multilayer neural network to estimate the coordinates of the positron position. A fully connected neural network with 3 hidden layers and two outputs was used as architecture. This method is limited to only estimating the event position but couldn't estimate the depth of interaction.

Lecomte [17] proposed a method to achieve high sensitivity and maintain uniform spatial resolution over the field of view in a PET scanner, which uses a phoswich approach to extract depth of interaction information from the crystals. As the scintillators become narrower, there is a problem of degradation in the sensitivity of the image resolution. One solution is to use larger ring diameters. Another solution is to use shorter scintillators or crystals. Casey [18] proposed a method to overcome the loss of sensitivity, which needs to determine the depth of interaction within the scintillators. In that work, a BGO crystal was stacked on a GSO crystal.

There is lack of literature on the depth – of – interaction (DOI) and the event position estimation using neural networks in a PET scanner. Clement [19] proposed a method for exploiting the division of light in a scintillating crystal to map the point of interaction of annihilation photons in 3D using neural networks. He also developed a multilayer neural network, which maps the light division with the light beam from the output of the photon interaction onto a scintillating crystal. This multilayer neural network was trained to estimate 2D coordinates of the light beam position. Three such neural networks were trained each learning 2 out of 3 spatial coordinates depending on which crystal had scintillated. However, using three neural networks increased the number of parameters to be trained in the system. Using one network decreases the computational complexity associated with the system. Peter et al. [20] proposed a method which extracts the position information from the sampled distribution of the scintillation photons using neural networks.

In this work, we developed a convolutional neural network which can directly estimate the event position and the depth of interaction all at the same time by mapping the distribution of the photon intensity onto the scintillator crystals.

Chapter 3: Data Generation

3.1 Detector Geometry

The simulated Annular PET scanner contains a 7.2 cm long annulus of scintillator called LYSO with an outer diameter of 8 cm and inner diameter of 5 cm as shown in Figure 1 (figure taken from [1]). Twelve facets with dimensions 1.9 cm x 7.2 cm were placed around the outer surface of the annulus equidistantly. The distance between the two outer SiPMs is 7.5 cm. SiPMs are silicon photomultipliers which are photodetectors. Simulation of the positron annihilation onto the detectors was performed by the Geant4 Application for Tomographic Emission (GATE) software package. It is very difficult to design new PET scanners without the simulation because of its cost and complexity in building the designs.

Figure 1: Schematic diagram of the AnnPET Scanner [1].

The image data acquisition from the scanner requires the simulations of the annihilation photons with the scintillator and the transport of the photons onto the SiPM arrays.

Figure 2: FlowChart of Photon Transport.

The Figure 2 (figure taken from [1]) shows the flowchart of the simulation of positron emissions, their annihilations and photon interactions in the scintillator. All these tasks were performed by the GATE software package. We used the DETECT200 software environment to model the creation of the scintillation photons and their transportation to the outer surface of the annulus on to the SiPM arrays. DETECT2000 uses Monte Carlo simulations to reproduce the simulation of the optical behavior of the scanner. It is also used to study the characteristics of the PET detectors.

The first step was to create the annihilation photon interaction points in the scintillator. The ^{22}Na point source was scanned which resulted in 200,000 events. With the help of the GATE software package, the event positions were generated within one sector of the facet. The points were generated in such a way that they are uniformly distributed. One facet sector angle θ ranges from 0 to $\tan^{-1} \left[\frac{2 \times 4.2625}{39.972} \right]$, where 4.2625mm is the pitch size and 39.972mm is the distance from the center of the annulus to one end of the facet, which is the outer radius. Figure 3 shows the uniform distribution plot of the event positions on a single facet. This is the x - y view of the event positions. Figure 4 shows the distribution plot of the event positions in the x - z plane.

$$\begin{aligned}
 x &= \sqrt{\text{radius}} * \cos \theta , \\
 y &= \sqrt{\text{radius}} * \sin \theta .
 \end{aligned}$$

Figure 3: x - y Distribution Plot of Generated Data.

Figure 4: x - z Distribution Plot of Generated Data.

The 3D positions thus created by the GATE simulation were used by the DETECT2000 software to create the scintillation photon initiation points. The number of photons created was directly proportional to the amount of energy deposited in the scintillator, which is approximately 32,000 MeV. DETECT2000 tracks the transport of the photons which were created inside the scintillator to the SiPM arrays. DETECT2000 generates the photon distributions detected by the SiPM arrays for each annihilation photon interaction. A coordinate system has been established after rotation of the axes. The SiPM surfaces were then parallel to the y - z plane, and the event position penetration depths were parallel to the x axis. The y - and z - coordinates were then separated into the histogrammed bins which were fitted with Gaussian functions. The data is included with the photons from either side of the facet with the maximum photon distribution. The photon distributions which are outside the geometry of the scanner and inside the inner surface of the annulus are discarded. The centroids of the Gaussian fits on the scintillator detectors were defined as the event position which gives the y - and z - coordinates. The x - coordinates which are the depth of the event positions are estimated by calculating the ratio of the count of the photons in the light distribution to the light intensity. The light distribution for each event is assumed to be a Gaussian distribution

as follows

$$N = I \int e^{-\frac{x^2}{2\sigma^2}} dx ,$$

where N is the total number of photons in the light distribution, I is the amplitude of the distribution, σ is the spread of the Gaussian distribution. Methods have been proposed to determine the relation between the depth of interaction and the Gaussian distribution of the light intensity [10].

3.2 Dataset

The dataset that was provided to us contains images of the light distribution on the scintillator detectors. At the beginning we were provided with the dataset containing the images and an excel sheet containing the 3D coordinates of the photon interactions inside the scintillator. A total of 199 images were simulated for one 3D coordinate of the events. There is a total of 821 coordinates in the excel sheet which constitutes to a total number of 163,379 images in the dataset. The Figures 5 and 6 below show the distribution of the event positions in the x - y and x - z directions of the data provided.

Figure 5: x - y Distribution Plot of HSC Data.

Figure 6: x - z Distribution Plot of HSC Data.

From the plots shown above, we can see that the data is not uniformly distributed. Much of the points are concentrated towards the end of the annulus, where there is the scintillator detector. This distribution made the training of the neural network harder. Indeed, it was reducing its ability to generalize in the regions away from the scintillator detector. Also, the grayscale images that were provided to us had a resolution of 13×16 . This needed to be corrected to 12×16 because the size of a single facet is 4×16 . The facet with the maximum light distribution is placed at the center, with one facet to the left and one facet to the right to compose the image representing the signal sensed by the scintillator detector. Finally, we also noticed that the repetition of several images per event position (199), were severely reducing the ability of the neural network to generalize. For these reasons we generated a new dataset with a uniform spatial distribution, revised resolution, and with single simulated image per event position.

The new dataset that we generated contains 200,000 event positions, containing one image per event position. The number of images per event position was limited to one. This is important to make sure that the network can generalize the predictions over the full extension of the scintillator rather than overspecializing around certain points only. We split the generated data into a training set made of 80% of the dataset, and a validation set

made of the remaining 20% of the dataset. All the experiments were performed on the newly created dataset. Figure 7 shows an image of the light distribution produced for one point.

Figure 7: 2D Image of the light distribution.

Chapter 4: Network Architectures

4.1 Architecture-1

We first describe the specialized functions implemented by some of the layers used in this architecture.

4.1.1 Batch Normalization

Batch normalization is a technique used for improving the training speed, performance and stability of neural networks. It is used to normalize the input layer by centering and scaling the activations. It was initially proposed to solve problems related to the internal covariate shift. The internal covariate shift is the change in the distribution of the network activations due to the change in the network parameters during training [21]. In neural networks, the output of the first layer is fed into the second layer, the output of the second layer is fed into the third layer and so on. When the weight parameters of a layer change, so does the distribution of the inputs to the subsequent layers. The shifts in these inputs create problems to the neural networks, especially deep neural networks that have large number of layers. The method of batch normalization was proposed to reduce the shifts in these inputs and to speed up the training and produce reliable models.

Batch normalization is achieved by fixing the mean and variance of each layer's inputs through a normalization step. This normalization can be conducted over the whole training set but to use it with the stochastic optimization methods, it is restricted to a mini batch in the training process.

Let B denote a mini batch of size m of the entire training set. The mean and variance of B are

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i, \quad \text{and} \quad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2.$$

For a layer of the network with d -dimensional input, $x = (x^{(1)}, \dots, x^{(d)})$, each dimension of its input is then normalized separately,

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}},$$

where $k \in [1, d]$ and $i \in [1, m]$, and $\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are the mean and variance along each dimension, respectively.

To restore the representation power of the network, a transformation step then follows as

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)},$$

where $\gamma^{(k)}$ and $\beta^{(k)}$ are subsequently learned in the optimization process. The output of the BN layer is then fed into the next network layer while the normalized output remains internal to the current layer.

4.1.2 Activation Functions

One of the most attractive properties of ANNs is its ability to adapt to the changing characteristics of the modeled system because of their property of being a universal approximator. Activation functions decides whether a neuron should be activated or not by calculating the weighted sum of the inputs and then adding the bias to it. More importantly, they introduce a non-linearity to the output of a neuron [22]. The weights and biases need to be updated during training based on the error calculated at the output of the neuron. Computationally, this process is known as backpropagation. Backpropagation happens through computational graph of a neural network, which includes the activation functions, which are supplied with the gradients according to the error of the downstream layers. Such gradients are then used to update weights and biases.

Rectified Linear Unit (ReLU) is the most commonly used activation function within the context of deep neural networks. ReLU is a half rectified function from the bottom as seen in Figure 8. The ReLU function $R(z) = \max\{0, x\}$ is 0 when z is less than 0 and $R(z)$ is equal to z when z is greater than or equal to 0.

Figure 8: Graphical representation of a ReLU function.

The range of the function is $[0, \infty)$. Both the ReLU function and its derivative are monotonically non-decreasing. The ReLU function does not activate all the neurons at the same time. Because it is piecewise linear, and because only a certain number of neurons are activated (i.e., $z > 0$), the ReLU function is computationally more efficient when compared to the other activation functions like sigmoid and tanh. More importantly, it does not saturate when it is active, which helps against the so called vanishing gradient problem, which limits the ability of the network to be trained.

4.1.3 Loss Function (MSE)

Our training goal is to minimize the loss for a neural network by optimizing the parameters or weights. The loss is calculated using a cost function which penalizes the residual error between the target value and the predicted value by a neural network. We then use the gradient descent method to update the weights so that the loss is minimized.

When we have a regression task, a typical design choice is to use the Mean Squared Error (MSE). This loss is calculated by taking the mean of the squared differences between

the predicted and the actual values. The MSE loss is never negative because we always square the errors. More precisely, we use the following expression

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 ,$$

where y_i is the actual (label) value, and $\hat{y}_i$ is the value predicted by the neural network with the current set of parameters.

4.1.4 Stochastic Gradient Descent

Gradient Descent is an optimization algorithm that is used to update the weights or parameters of a machine learning prediction model. The model is used to predict the target labels and the residual errors on them are used to guide the updates of the weights or parameters of the model in order to reduce the training loss and thus the errors themselves. So, the goal of the gradient descent algorithm is to minimize the training loss function. This works by iteratively updating the model parameters along the negative direction of gradient, which is the direction towards which the loss should decrease the most, i.e.,

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} ,$$

where η is the step size or learning rate, w are the weight parameters, and L is the loss function.

If the learning rate is too high, then the iteration converges quickly but it might skip the true local minimum. If the learning rate is too low, then it might take a lot of time for the model to converge. Sometimes it is difficult to find the local minimum in these cases. We used a learning rate of 0.0002 for this model. Due to memory limitations, these gradients are calculated on a smaller set of training sample called the mini-batch, and then the parameters are updated. This method is called mini-batch gradient descent, which is a version of the so called Stochastic Gradient Descent.

4.1.5 Dropout

Generally, deep neural networks can overfit a dataset if it does not have enough data samples. In such situations, the model learns the noise which results in poor performance of the model on the validation dataset. One approach to avoid such overfitting problem is to fit different neural networks on the same dataset, and then average the predictions from all the models. While this method is not feasible in practice, it is possible to achieve a similar effect with a regularization technique called dropout.

Dropout is a regularization method which is similar to training different neural network models in parallel. During training, a random number of neurons in a given layer are ignored. This makes the single architecture looks like several different architectures.

Dropout can be implemented on any or all of the hidden layers. A new hyperparameter is introduced which requires the probability at which the neurons in a layer are to be dropped out. A typical probability value for dropout is 0.5 for the nodes in a hidden layer to retain the outputs, and a value close to 1 is generally used for the input layers. Dropout can't be used when training a network for making a prediction by starting from a pretrained model. We used a dropout probability of 0.6 in all the hidden layers of this architecture.

Architecture

Figure 9 shows the diagrammatic representation of Architecture-1, which is based on fully connected layers. The image with 12×16 resolution is flattened into a 1D Python `numpy` array. The inputs are fed into a fully connected hidden layer with 600 nodes. The inputs from the previous layer are connected to each node of the hidden layer. The outputs from this layer are then passed through ReLU activation functions. The outputs from the hidden-1 layer are then fed to the hidden-2 layer which has 400 nodes. The inputs to the hidden-2 layer are then passed through the ReLU activation functions. The outputs from the hidden-2 layer are then fed to the output layer with 3 nodes, which give out the coordinates of the estimated event position in all the three x , y and z directions.

Figure 9: Structure of Architecture-1.

The table in Figure 10 gives a clear description of each layer and the series of opera-

tions performed.

Layer	Operation
Input Layer	Image with resolution 12x16 – Converted into 1-D numpy array
Hidden Layer - 1	600 nodes
	Activation Function - ReLU
Hidden Layer - 2	400 nodes
	Activation Function – ReLU
Output Layer	3 nodes which estimates the x, y and z coordinates.

Figure 10: Description of Architecture-1.

4.2 Architecture-2

Convolutional neural networks (CNNs) have become more dominant in the field of computer vision tasks. Convolutional neural networks work like fully connected neural networks which have sharable weights and biases. CNNs can automatically learn the spatial features through backpropagation architectures that combine convolutional layers, pooling layers and fully connected layers [23]. These networks perform better on tasks where the input data are images due to the reduction in the number of parameters to estimate, and their reusability. The role of a convolutional neural network is to reduce the image into another form which is easier to process, without losing any the features necessary for the prediction.

4.2.1 Convolution Layer

A convolution layer is the basic part of a Convolutional Neural Network, which performs the combination of both the linear and non-linear operations for feature extraction, i.e., convolution operation followed by the activation function [18]. An array of numbers called a kernel is applied across an input array of the image called a tensor. Element-wise product between each element of the tensor and the kernel is calculated at each position of the tensor, and then summed up to an output array called a feature map. Figure 11 shows an example of a convolution by a kernel with size 3×3 , 0 padding and a stride of 1 being applied on the input image of size 5×5 .

Figure 11: Example of convolutional operation with a kernel of 3×3 , no padding and stride 1.

The outputs from this convolution layer are then fed through a non-linear activation function. Functions like the hyperbolic tangent, or sigmoid were among the early choices as activation functions. The most used activation function nowadays is the ReLU. The three parameters that we need to choose for a convolutional layer are the kernel size, the stride and the padding.

Convolutional Kernel

Each convolutional layer has a series of filters called the convolutional kernels. The filter is a matrix of integers that are applied to the input image pixels. Each pixel in the image is multiplied with each pixel of the kernel, the result is then summed up to form an output feature map. The kernel moves over the image horizontally column by column over the first row of the image. Then the kernel moves vertically down to the next row and continues to move over the pixels in that row. Eventually, the feature map becomes the input convolved with the kernel.

Padding

Sometimes we want to apply the convolutional layers on the input images, but we do not want the image to decrease its spatial dimensions. To achieve this, we apply a zero padding, which adds the zero pads to the input image at the borders. The size of the padding P is computed as

$$P = \frac{K - 1}{2},$$

where K is the size of the kernel.

Figure 12: Image after zero padding of size $P = 2$.

Figure 12 shows the input image of spatial dimension $32 \times 32 \times 3$ after zero padding of size P on the borders. So, when we apply a convolution on this padded image of spatial dimension $36 \times 36 \times 3$, with a kernel size of 5×5 and stride 1, then we will get an output image of spatial dimension $32 \times 32 \times 3$. When we apply the zero padding of P , then both the input and output volume will have the same spatial dimensions. The formula for calculating the output size for a given convolutional layer is as follows

$$O = \frac{W - K + 2P}{S} + 1 ,$$

where O is the output height/length, W is the input height/length, K is the filter size, P is the padding and S is the stride.

Stride

Stride controls the kernel movement on the input image. If the stride is 1, then the kernel moves over the image one pixel at a time. The amount by which the kernel moves over the image is called a stride. Stride is normally set in a way that the output volume (size) after applying the convolution is an integer, not a fraction.

Figure 13: Illustration of stride when applying convolution. The three colors show three adjacent positions where the kernel was applied. The stride in this case was $S = 2$.

Figure 13 shows an input image with size (volume) 7×7 to which it has been applied convolution with a kernel of size 3×3 with a stride 2. The output volume now is 3×3 after applying the convolution.

4.2.2 Pooling Layer

The outputs from the feature maps are sensitive to the location of the features in the input. One approach to address this sensitivity is to down sample the feature map. This makes the feature map more robust to the change in position of the features in the feature map. Pooling layers downsample the feature maps. The two most common pooling methods are Average Pooling and Max Pooling. The first method computes the average of the features over a specific region. The second method computes the maximum of the feature over a specific region. Downsampling can be achieved by applying the convolution across the image with a longer stride. A pooling layer is often added after a convolutional layer, right after the outputs from the convolutional layer are passed through the ReLU non-linear activation functions. The sequence of layers in the model looks like:

Input Image $\rightarrow$ Convolutional Layer $\rightarrow$ Nonlinearity $\rightarrow$ Pooling Layer

For pooling, we need to choose the pooling operation and the filter to be applied on the feature map. The size of the filter must be smaller than the size of the feature map. It is often applied with a 2×2 size and a stride of 2 pixels. This means that the pooling layer almost always reduces the dimension by a factor of 2, i.e., the dimension of the feature map is halved.

4.2.3 Fully Connected Layer

The outputs of the feature map from the last convolutional layer or the pooling layer is then flattened into a 1D array of numbers, which are then connected to one or more fully connected layers. The outputs are connected to each and every node of the fully connected layer. Once the features are extracted by the convolutional layers, and are down sampled by the pooling layers, the outputs are mapped by the fully connected layers to the final outputs of the network. The outputs from each fully connected layer is also passed to a non-linear activation function.

Figure 14: Architecture-2.

Figure 14 shows the diagrammatic representation of Architecture-2. The raw image of size 12×16 is padded with zeros and is converted to size 16×16 . A convolution layer is applied on the input images with resolution 16×16 and 1 channel, with a kernel size 4×4 , stride 2 and padding 1. The output from the first convolutional layer is an image with 128 channels. The outputs of the convolutional layer are then followed by ReLU activation functions. Only some neurons get activated from this convolutional layer. The outputs are then fed to the next convolutional layer with kernel size 4×4 , stride 2 and padding 1. The output channels of this convolutional layer are 256. The outputs from this convolutional layer are then batch normalized. The neurons are activated by using the ReLU function. The outputs of the ReLU function are then fed to the convolutional layer with input channels 256, output channels 512, kernel size 3×3 , stride 1 and padding 1. These inputs for the next layer are then batch normalized with 512 features. ReLU activation is performed on the outputs from the batch normalization. The outputs from some of the activated neurons from the previous layer are then fed to the last convolutional layer with input channels 512, kernel size 4×4 , stride 1 and padding 0. The outputs from the last convolutional layer are flattened, connected to the 3 nodes of the output layer by the help of a fully connected layer. This is the architecture used in mapping the spread of the light distribution to the depth of interaction and the event position.

Layer	Operation
Input Layer	Image with resolution 12x16 after padding gets converted into 16x16.
Convolutional Layer	(Kernel, Stride, Padding) – (4x4, 2, 1)
	Activation Function – ReLU
Convolutional Layer	(Kernel, Stride, Padding) – (4x4, 2, 1)
Batch Normalization	
	Activation Function - ReLU
Convolutional Layer	(Kernel, Stride, Padding) – (3x3, 1, 1)
Batch Normalization	
Convolutional Layer	(Kernel, Stride, Padding) – (4x4, 1, 0)
Fully Connected Layer	
Output Layer	3 nodes which estimates the event positions.

Figure 15: Description of Architecture-2.

The table in Figure 15 gives a clear description of each layer and the series of operations performed.

4.3 Architecture - 3

The ReLU function suffers from the dying ReLU problem, whereby some neurons stay inactive for whatever the inputs they are given. When this happens, there is no gradient flow going through them which prevents the neurons from training, ultimately affecting the performance of the network. In such situations, we use Leaky ReLU where the slope for a negative input is changed. Instead of setting the output to zero when $x < 0$, Leaky ReLU has a small slope. Figure ?? shows the graphical representation of the Leaky ReLU function, which is defined as

$$f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x) ,$$

where α is a small constant. The two main benefits of Leaky ReLU are:

- It fixes the dying ReLU problem since it does not have zero-slope parts.
- This speeds up the training process. Leaky ReLU is more balanced and therefore helps the model learn faster.

Figure 16 shows the diagrammatic representation of Architecture-3. The raw image of size 12×16 is padded with zeros and is converted to size 16×16 . This is fed to a convolutional layer which changes the image with 1 input layer to 64 output layers using a

kernel size of 1×1 , stride 1 and padding 0. The outputs from this convolutional layer are then fed into the ReLU activation function. The outputs from the activation function are then fed into the network inside a block.

Each block contains the following layers. The outputs from the previous layer are fed into the convolutional layer which doubles the input filters. Then the 1st set of biases are added to the outputs. 1st set of batch normalization is applied on these outputs. A LeakyReLU activation function is applied on the outputs from the previous convolutional layer with a slope of 0.2. A 2D convolution is applied on the output image which contains several input planes which blur the image. A `numpy` array is used as a mask which is applied onto the output image from the previous layer. This convolution has a kernel of size 3×3 , padding 1. Then the outputs from the previous 2D convolutional layer is sent into an average pooling layer with kernel 2×2 and stride 2. This is used to downscale the images which will be helpful in retrieving much information from the images. The 2nd set of biases are then added to the outputs and are then fed into the 2nd set of batch normalization on these outputs. The neurons are then activated using the ReLU activation function. This whole architecture comes under one block.

Three of such blocks are added one after the other. The outputs from the LeakyReLU after the application of the 1st set of batch normalization to the 1st convolutional layer in the block are fed to the fully connected layer. The outputs from the last convolutional layer from the last block are then fed to a fully connected layer with 256 nodes. The ReLU activation function is applied on the outputs from the previous fully connected layer. The 256 nodes from the previous layer are then fully connected to the 3 neurons in the output layer which gives the estimated coordinates of the event positions in the x , y and z directions.

This architecture with different learning rates are trained and tested on the validation set. This architecture performed better with the learning rate 0.0002. A learning rate scheduler technique is used which decreases the learning rate by the factor of 0.2 every 50 iterations.

Figure 16: Architecture-3.

The table in Figure 17 gives a clear description of each layer and the series of operations performed.

Layer	Operation
Input Layer	Image with resolution 12x16 after padding gets converted into 16x16.
Convolutional Layer	(Kernel, Stride, Padding) – (1x1, 1, 1)
	Activation Function – ReLU
Block * 2	
Convolutional Layer	(Kernel, Stride, Padding) – (3x3, 1, 1)
	First set of biases are added to the outputs
Batch Normalization	
	Activation Function - LeakyReLU
2D Convolution	A mask with weights as kernel with weights, Stride – 1, Padding – 1.
Convolutional Layer	(Kernel, Stride, Padding) – (3x3, 1, 1)
Average Pooling Layer	(Kernel, Stride) – (2x2, 2)
Convolutional Layer	(Kernel, Stride, Padding) – (3x3, 1, 1)
	Last set of biases are added to the output.
Batch Normalization	
	Activation Function – LeakyReLU
Fully Connected Layer	
Output Layer	3 nodes which estimates the event positions.

Figure 17: Description of Architecture-3.

Chapter 5: Results and Evaluation

This chapter summarizes the achieved results with the three architectures analyzed, and the data that was generated as part of the simulation.

5.1 Data Distribution

The simulated data is given by pairs, each of which has an image representing the light distribution on the PET detector, and a 3D position representing the coordinates x , y , and z of the event positions.

L2 Norm Calculation

The L2 norm of the residual error is used for calculating the error of a prediction. It is the Euclidean distance between the actual and predicted 3D position of the events. It is defined as

$$\|x\|_2 = \frac{1}{N} \sqrt{\sum_{i=1}^N (x_i - \hat{x}_i)^2},$$

where x_i is the actual (label) value and $\hat{x}_i$ is the predicted value. This gives the sum of the squares of the errors between the actual and predicted 3D positions of the photon interactions. The mean of the total error is then calculated by dividing the total error by the total number N of samples in the validation set.

Mean and Standard Deviation Calculation

The sample mean is calculated by dividing the sum of the samples by the total number of samples in the validation set:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i),$$

where x_i is the actual value and $\hat{x}_i$ is the predicted value.

The standard deviation is the square root of the average (minus 1 to correct the bias) of the squared differences from the mean:

$$\sigma = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N - 1}},$$

where σ is the standard deviation.

Figure 18: Distribution plot along the x coordinate of the data labels.

Figure 18 shows the distribution of the data labels along the x direction. The mean of the x coordinates is 35.093693 and the standard deviation is 2.9156187. The minimum value along this direction is 29.361 and the maximum value is 39.925. The range of the values along this direction is 10.563999.

Figure 19: Distribution plot along the y coordinate of the data labels.

Figure 19 shows the distribution of the data labels along the y direction. The mean of the y coordinates is -0.00790741 and the standard deviation is 4.360502 . The minimum value along this direction is -8.501 and the maximum value is 8.493 . The range of the values along this direction is 16.994 .

Figure 20: Distribution plot of the z coordinate of the data labels.

Figure 20 shows the distribution of the data labels along the z direction. The mean of the z coordinates is -1.9668504 and the standard deviation is 19.704615 . The minimum value along this direction is -36.1 and the maximum value is 32.1 . The range of the values along this direction is 68.2 .

5.2 Results with Architecture-1

This section presents the absolute error, and absolute errors in the x , y and z directions of the predictions on the validation set. A hyperparameter search was performed to select the best architecture. Tables 1, 2 and 3 present the results of the performed experiments using different hyperparameters.

The number of nodes in each layer is one of the parameters that we focused on while building these architectures. We also considered the regularization technique dropout with the probability 0.6 for both the input and the hidden layer-1.

5.2.1 Experiment-1

There are 600 nodes in the hidden layer-1 and 400 nodes in the hidden layer-2. Dropout is not included in this experiment. The total absolute error, the root mean squared error (RMSE) and the absolute errors in the x , y and z directions are presented in Table 1. Figure 41 represents the total error on the validation set with these hyperparameters.

Figure 21: **Architecture-1 - Experiment-1.** Distribution plot of the total absolute error on validation set.

Figures 42, 43 and 44 represent the distribution plots of the absolute errors in the x , y and z directions respectively.

Figure 22: **Architecture-1 - Experiment-1.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 23: **Architecture-1 - Experiment-1.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 24: **Architecture-1 - Experiment-1.** Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	0.949482
Mean of Absolute Error on x coordinate	0.210155
Mean of Absolute Error on y coordinate	0.260805
Mean of Absolute Error on z coordinate	0.825599
RMSE	0.591514

Table 1: **Architecture-1 - Experiment-1.** Results with 600 nodes in hidden layer-1 and 400 nodes in hidden layer-2.

5.2.2 Experiment-2

There are 400 nodes in the hidden layer-1 and 300 nodes in the hidden layer-2. The Total Error, RMSE and absolute errors in x , y and z directions are presented in the Table 2.

Figure 41 represents the total error on the validation set with these hyperparameters.

Figure 25: **Architecture-1 - Experiment-2.** Distribution plot of the total absolute error on the validation set.

Figures 42, 43 and 44 represent the distribution plots of the absolute errors in the x , y and z directions respectively.

Figure 26: **Architecture-1 - Experiment-2.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 27: **Architecture-1 - Experiment-2.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 28: **Architecture-1 - Experiment-2.** Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	0.949744
Mean of Absolute Error on x coordinate	0.224601
Mean of Absolute Error on y coordinate	0.258835
Mean of Absolute Error on z coordinate	0.821518
RMSE	0.592427

Table 2: **Architecture-1 - Experiment-2.** Results with 400 nodes in hidden layer-1 and 300 nodes in hidden layer-2.

5.2.3 Experiment - 3

The regularization technique dropout is added to both the input layer and hidden layer-1 with probability 0.6. There are 600 nodes in the hidden layer-1 and 400 nodes in the

hidden layer-2. The results are presented in Table 3. Figure 41 plots the distribution of the total error on the validation set with these hyperparameters.

Figure 29: **Architecture-1 - Experiment-3.** Distribution plot of the total absolute error on the validation set.

Figures 42, 43 and 44 represent the distribution plots of the absolute errors in the x , y and z directions respectively.

Figure 30: **Architecture-1 - Experiment-3.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 31: **Architecture-1 - Experiment-3.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 32: **Architecture-1 - Experiment-3.** Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	3.088526
Mean of Absolute Error on x coordinate	2.059363
Mean of Absolute Error on y coordinate	1.117341
Mean of Absolute Error on z coordinate	1.411967
RMSE	1.969960

Table 3: **Architecture-1 - Experiment-3.** Results with 400 nodes in hidden layer-1 and 300 nodes in hidden layer-2, after adding dropout regularization.

Experiment-1 gives the best results where we used 600 nodes in the hidden layer-1 and 400 nodes in the hidden layer-2. The mean of the total error for the experiment-1 is 0.9494. The mean of the total error when dropout was used is 3.088. The model loses

some of the capacity when the dropout technique is used because the model deactivates the neurons for each layer with the probability 0.6.

From all the experiments, the architecture in the experiment-1 has performed well. The learning rate is the parameter that we used to tune this architecture to get the best results. Over several learning rates that we tried we found that setting a learning rate of 0.004 has given the best results. Table 4 shows the performance of the best Architecture-1.

	Validation Set
Mean of the Total Error	0.8842
Mean of Absolute Error on x coordinate	0.1673
Mean of Absolute Error on y coordinate	0.2235
Mean of Absolute Error on z coordinate	0.7897
RMSE	0.5497

Table 4: **Architecture-1**. Results with 600 nodes in hidden layer-1 and 400 nodes in hidden layer-2, after searching for the best learning rate.

5.3 Results with Architecture-2

This section presents the total absolute error, absolute errors in x , y and z directions on the validation set by using Architecture-2. A hyperparameter search was performed to select the best architecture. The kernel size, stride and padding are the hyperparameters that we focused on while building the convolutional network. We changed these values and repeated the experiment thrice. Tables 5, 6 and 7 present the results of the experiments using different hyperparameters.

5.3.1 Experiment-1

Four convolutional layers were used in this architecture. The kernel size, stride and padding values (4,2,1) were used in the 1st, 2nd and 4th convolutional layer, and (3,1,1) were used in the 3rd convolutional layer. Stride 2 halves the resolution of the image which was helpful for extracting the features of the image. The set (4,1,0) was used in the last convolutional layer which flattens the image. The total error, RMSE and absolute errors in x , y and z directions are presented in Table 5. Figure 41 shows the total error distribution on the validation set with these hyperparameters.

Figure 33: **Architecture-2 - Experiment-1.** Distribution plot of the total absolute error on the validation set.

Figures 42, 43 and 44 represents the distribution plots of the absolute errors in the x , y and z directions respectively.

Validation Validation Distribution of X-Coordinate Absolute Errors

Figure 34: **Architecture-2 - Experiment-1.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 35: **Architecture-2 - Experiment-1.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 36: **Architecture-2 - Experiment-1.** Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	0.373281
Mean of Absolute Error on x coordinate	0.163156
Mean of Absolute Error on y coordinate	0.185713
Mean of Absolute Error on z coordinate	0.202385
RMSE	0.278950

Table 5: **Architecture-2 - Experiment-1.** Summary of the average errors.

5.3.2 Experiment-2

There are 5 convolutional layers in this architecture. The kernel size, stride and padding values (5, 1, 1) were used in the 1st, 2nd and 4th convolutional layers which decreases the

resolution of the image. The values set to (4, 2, 1) were used in the 3rd convolutional layer, which halves the resolution of the image. The set (4, 1, 0) is used in the last convolutional layer which flattens the image. The total error, RMSE and absolute errors in x , y and z directions are presented in Table 6. Figure 41 shows the distribution of the total error on the validation set with these hyperparameters.

Figure 37: **Architecture-2 - Experiment-2.** Distribution plot of the total absolute error on the validation set.

Figures 42, 43 and 44 represent the distribution plots of the absolute errors in the x , y and z directions respectively.

Validation Validation Distribution of X-Coordinate Absolute Errors

Figure 38: **Architecture-2 - Experiment-2.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 39: **Architecture-2 - Experiment-2.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 40: **Architecture-2 - Experiment-2**. Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	0.348037
Mean of Absolute Error on x coordinate	0.151523
Mean of Absolute Error on y coordinate	0.179085
Mean of Absolute Error on z coordinate	0.183844
RMSE	0.260126

Table 6: **Architecture-2 - Experiment-2**. Summary of the average errors.

5.3.3 Experiment - 3

Three convolutional layers were used in this architecture. The set $(4, 2, 1)$ was used in the first convolutional layer which halves the resolution of the image. The set $(5, 1, 1)$ was

used in the second convolutional layer which decreases the resolution of the image. The set (6,1,0) was used in the 3rd convolutional layer which flattens the image. The results are presented in the Table 7. Figure 41 represents the total error on the validation set with these hyperparameters.

Figure 41: **Architecture-2 - Experiment-3.** Distribution plot of the total absolute error on the validation set.

Figures 42, 43 and 44 represent the distribution plots of the absolute errors in the x , y and z directions respectively.

Figure 42: **Architecture-2 - Experiment-3.** Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 43: **Architecture-2 - Experiment-3.** Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 44: **Architecture-2 - Experiment-3.** Distribution plot of the absolute error along the z coordinate on the validation set.

	Validation Set
Mean of the Total Error	0.491954
Mean of Absolute Error on x coordinate	0.209314
Mean of Absolute Error on y coordinate	0.206913
Mean of Absolute Error on z coordinate	0.305411
RMSE	0.358927

Table 7: **Architecture-2 - Experiment-3.** Summary of the average errors.

From among all the architectures in the three different experiments, we selected the best performer from Experiment-2. We then trained the same architecture with different learning rates. The architecture performed better for a learning rate of 0.0004, which is double the one of the previous learning rate. We also uses a learning rate scheduler, whereby the learning

rate is decreased by a factor of 0.2 after every 50 iterations. The training is performed for 150 iterations. The results are presented in the Table 8.

	Validation Set
Mean of the Total Error	0.3292
Mean of Absolute Error on x coordinate	0.1417
Mean of Absolute Error on y coordinate	0.1718
Mean of Absolute Error on z coordinate	0.1734
RMSE	0.2480

Table 8: **Architecture-2 - Experiment-2.** Summary of the average errors after changing the learning rate and the learning rate scheduler.

5.4 Results with Architecture-3

5.4.1 Distribution Plots of Errors on the Validation Set

After training Architecture-3 we computed the distribution plots of the total error, and the errors along the x , y and z directions. The total error is calculated using the L2 norm between the actual position of the events and the predicted positions. Figure 45 shows the distribution plot of the total error. The mean of the total error is 0.25493044 and the standard deviation is 0.23525739.

Figure 45: **Architecture-3.** Distribution plot of the total error on the validation set.

Below there are the plots of the distributions of the errors on the x , y and z directions.

Figure 46: **Architecture-3.** Distribution plot of the error along the x coordinate on the validation set.

Figure 46 shows the distribution plot of the error along the x coordinate. The mean error is 0.0128 and the standard deviation is 0.1425.

Figure 47: **Architecture-3**. Distribution plot of the error along the y coordinate on the validation set.

Figure 47 shows the distribution plot of the errors along the y coordinate. The mean error -0.0076 and the standard deviation is 0.2258 .

Figure 48: **Architecture-3.** Distribution plot of the error along the z coordinate on the validation set.

Figure 48 shows the distribution plot of the error along the z coordinate. The mean error is -0.0173 and the standard deviation is 0.2203 .

5.4.2 Distribution Plots of Absolute Errors on Validation Set

This section simply shows the plots of the distributions of the absolute errors along the three coordinates x , y , and z .

Figure 49: **Architecture3**. Distribution plot of the absolute error along the x coordinate on the validation set.

Figure 49 shows the distribution plot of the absolute errors along the x coordinate. The mean of the absolute error is 0.1059 and the standard deviation is 0.0962.

Figure 50: **Architecture3**. Distribution plot of the absolute error along the y coordinate on the validation set.

Figure 50 shows the distribution plot of the absolute errors along the y coordinate. The mean of the absolute error is 0.1383 and the standard deviation is 0.1786.

Figure 51: **Architecture3**. Distribution plot of the absolute error along the z coordinate on the validation set.

Figure 51 shows the distribution plot of the absolute errors along the z coordinate. The mean of the absolute error is 0.1328 and the standard deviation is 0.1766. Table 9 presents the summary results of the mean of the total error, the RMSE, and the errors in x , y and z directions.

	Validation Set
Mean of the Total Error	0.2549
Mean of Absolute Error on x coordinate	0.1059
Mean of Absolute Error on y coordinate	0.1383
Mean of Absolute Error on z coordinate	0.1328
RMSE	0.2002

Table 9: **Architecture-3**. Summary of the average errors.

Figure 52 shows the plot of the training and validation losses. From this plot we can say

that the model is learning and is also generalizing beyond the training set. The loss becomes nearly constant after 120 epochs, clearly indicating that the model is fully trained and the optimization can be stopped. Overfitting happens when the model learns the noise in the data to such an extent that it performs very badly on the new data. We do not observe here since the loss on the validation set does not start to increase. Underfitting happens when the model does not learn from the training data and it also does not generalize well on the new data. By looking at the plot we can say that underfitting is not happening and our model progressively learns from the training data. Since the curve of the validation set follows the training curve rather closely, the model generalizes well to the new data.

Figure 52: **Architecture-3**. Plot of the training and validation losses.

Comparison of the results with all the architectures

Table 10 presents the summary of the results after training and testing the three best architectures on the validation set 1, and it clearly shows that Architecture-3 is the best performer.

	Architecture-1	Architecture-2	Architecture-3
Mean of the Total Error	0.8842	0.3292	0.2549
Absolute Error on x coordinate	0.1673	0.1417	0.1059
Absolute Error on y coordinate	0.2235	0.1718	0.1383
Absolute Error on z coordinate	0.7897	0.1734	0.1328
RMSE	0.5497	0.2480	0.2002

Table 10: Comparison between the architectures on the validation set 1.

Chapter 6: Conclusions and Future Work

In this work, we have discussed the steps that need to be taken to generate simulated data inside a PET scanner. In order to train a neural network able to generalize to every input it is important to make to simulate event positions that are uniformly distributed across one section of the facet. Specifically, they must be uniformly distributed across both the x - y directions as well as the x - z direction. The dataset that we generated is constituted by pairs, each of which is given by image of the light distribution on the scintillation detector with the corresponding 3D coordinates of the event position.

We designed three architectures to estimate the 3D coordinates of the annihilation photon interactions from images representing the light distribution captured by the scintillator detector. Architecture-1 is a traditional multi layer perceptron type of neural network, made of fully connected layers. This simple architecture allows to predict the output with a mean total error of 0.8842. Subsequently, we improved with Architecture-2 which is a basic convolutional neural network. The mean total error decreased to 0.3292, which is a substantial 62.76% improvement over Architecture-1. We then made a number of changes to Architecture-2 and obtained Architecture-3 which gave a decreased mean total error of 0.2549, which is another substantial 22.56% improvement.

In future, we intend to test the models developed on new simulated data that will be generated. In addition, a PET prototype will be built soon and real data will be generated on which our models will be tested. Our long term goal is to adapt these models to work well on real data, by using transfer learning techniques. The ultimate goal is to develop a model which can estimate the event positions inside the body by 3D mapping the images on the scintillator.

References

- [1] A. V. Stolin, P. F. Martone, G. Jaliparthi, and R. R. Raylman, “Preclinical positron emission tomography scanner based on a monolithic annulus of scintillator: initial design study,” *Journal of Medical Imaging*, vol. 4(1), 2017.
- [2] P. M. Bloomfield, “The design and physical characteristics of a small animal positron emission tomograph,” *Phys. Med. Biol.*, vol. 40, p. 1105–1126, 1995.
- [3] A. L. G. et al., “First results from the high-resolution mouse spect annular scintillation camera,” *IEEE Trans. Med. Imaging*, vol. 24, p. 863–867, 2005.
- [4] T. M. Button, “Small animal mri at 0.35 tesla: growth and morphology of intra-organ murine tumors,” *Magnetic Resonance Imaging*, vol. 8, p. 505–509, 1990.
- [5] M. J. Paulus, “High resolution x-ray computed tomography: an emerging tool for small animal cancer research,” *Neoplasia*, vol. 2, p. 62–70, 2000.
- [6] R. Lecomte, “Initial results from the sherbrooke avalanche photodiode positron tomography,” *IEEE Trans. Nucl. Sci.*, vol. 43, p. 1952–1957, 1996.
- [7] S. S. J. et al., “Simulation study of spatial and sensitivity for the tapered depth of interaction pet detectors for small animal imaging,” *IEEE Trans. Med. Imaging*, vol. 55, p. N63–N74, 2010.
- [8] F. Sanchez, “Small animal pet scanner based on monolithic lyso crystals: performance evaluation,” *Medical Physics*, vol. 39, p. 643–653, 2012.
- [9] S. E. et al., “Digipet: sub-millimeter spatial resolution small-animal pet imaging using thin monolithic scintillators,” *Phys. Med. Biol.*, vol. 59, p. 3405–3420, 2014.
- [10] E. J. Hoffman, T. M. Guerrero, G. Germano, W. M. Digby, and M. Dahlbom, “Pet system calibrations and corrections for quantitative and spatially accurate images,” *IEEE Trans. Nucl. Sci.*, vol. 36, p. 1108–1112, 1989.
- [11] W. M. Moses and S. E. Derenzo, “Design studies for a pet detector module using a pin photodiode to measure depth of interaction,” *IEEE Trans. Nucl. Sci.*, vol. 41, p. 1441–1445, 1994.
- [12] C. W. L. et al., “Depth of γ -ray interaction within continuous crystals from the width of its scintillation light-distribution,” *IEEE Trans. Nucl. Sci.*, vol. 52, p. 560–572, 2005.
- [13] H. T. van Dam et al., “A practical method for depth of interaction determination in monolithic scintillator pet detectors,” *Phys. Med. Biol.*, vol. 56, p. 4135–4145, 2011.
- [14] R. P. et al., “Continuous doi determination by gaussian modelling of linear and non-linear scintillation light distributions,” *Proc. IEEE Nuclear Science Symposium Conference Record*, p. 3386–3389, 2011.

- [15] A. J. G. et al., “A pet design based on sipm and monolithic lyso crystals: performance evaluation,” *IEEE Transactions on Nuclear Science*, vol. 63(5), p. 2471–2477, 2016.
- [16] S. Delorme, R. Frei, C. Joseph, J.-F. Loude, and C. Morel, “Use of a neural network to exploit light division in a triangular scintillating crystal,” *Nucl. Instr. Methods Phys. Res.*, vol. A373, p. 111–118, 1996.
- [17] C. Carrier, C. Martel, D. Schmitt, and R. Lecomte, “Design of a high-resolution positron emission tomograph using solid state scintillation detectors,” *IEEE Transactions on Nuclear Science*, vol. 35, pp. 685–690, 1988.
- [18] M. E. C. et al., “Investigation of lso crystals for high spatial resolution positron emission tomography,” *IEEE Transactions on Nuclear Science*, vol. 44, pp. 1109–1113, 1997.
- [19] D. Clement, R. Frei, J.-F. Loude, and C. Morel, “Development of a 3d position sensitive scintillation detector using neural network,” *Proc. IEEE Med. Imag. Conf.*, 1998.
- [20] O. N. et al., “Deep learning vs. traditional computer vision,” *Advances in Intelligent Systems and Computing*, vol. 943, 2020.
- [21] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *ICML Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol. 37, p. 448–456, 2015.
- [22] A. Olgac and B. Karlik, “Performance analysis of various activation functions in generalized mlp architectures of neural networks,” *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 02 2011.
- [23] Y. R., N. M., D. R.K.G., and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, p. 611–629, 2018.