Neural Networks for Particle Reconstruction in High Energy Physics Detectors

Tiansheng Dai[1,*], Xiaobin Xu[2], Bo Hu[3], Kunhao Yue[4], Xinao Niu[5]
{daitiansheng0831@gmail.com[1], 1993508926@qq.com[2],
3462345845@qq.com[3], 221840136@smail.nju.edu.cn[4], 221870174@smail.nju.edu.cn[5]}

Computer Science and Technology, Xidian University, Xian, China[1]
Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China[2]
Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China[3]
Mathematics, Nanjing University, Nanjing, China[4]
Mathematics, Nanjing University, Nanjing, China[5]
*corresponding author

Abstract. The core of the Particle Flow (PFlow) algorithms in High Energy Physics experiments lies in reconstructing the intrinsic nature and kinematic properties of both charged and neutral particles, utilizing the data provided by detectors. Through effective trajectory analysis methods for various particle types, it becomes evident that the magnetic field significantly influences the accuracy of final reconstruction efforts. In this paper, we present several computer vision models aimed at enhancing PFlow algorithms. We employ a Multilayer Perceptron (MLP) to address scenarios where no magnetic forces interfere with particle tracks. Subsequently, we conduct a comprehensive discussion and comparison of the experimental results obtained from various enhanced Convolutional Neural Network (CNN) models, such as Polar Coordinate Processing, Parameterized Differential Operators (PDOs), Cylindrically Sliding Windows (CSW), tailored to the characteristics of our datasets.

Keywords: Particle reconstruction, High-energy detectors, Convolutional neural networks, Polar processing, Image fusion

1 Introduction

The general-purpose high-energy collider experiment aims to measure the trajectories of charged particles and the energy deposits in calorimeter clusters. Under the influence of a magnetic field, the tracks of charged particles exhibit deflection compared to those of neutral particles, providing crucial information necessary for reconstructing, verifying, and scaling the energy of the particles involved in the event. These particles primarily include charged and neutral hadrons, photons, electrons,

CONF-MLA 2024, November 21, Adana, Turkey

DOI 10.4108/eai.21-11-2024.2354620

muons, and neutrinos. Some other particles have a lifetime too short to be directly measured in the experiment, necessitating their reconstruction from decay products. This experiment is of significant importance for the study of energetic particles, which underscores the urgent need for research into PFlow algorithms.

PFlow algorithms are specifically designed to offer an effective means of reconstructing the nature and kinematic properties of particles within the detector acceptance during collisions in high-energy physics experiments. The key challenge lies in distinguishing neutral particles from charged ones by analyzing the detected tracks. One of the major challenges in developing PFlow algorithms is designing a model that can effectively differentiate between charged and neutral particles. The algorithm must accurately track particles by fusing images from three independent detectors: the Electromagnetic Calorimeter (ECAL), the Hadronic Calorimeter (HCAL), and the Tracker. For instance, the position of a charged particle will shift due to deflection, which must be accounted for in the particle track identification process.

In this paper, we adopt various computer vision approaches to address this fundamental issue in PFlow algorithms. We propose several neural network architectures, including MLP, U-Net and CNN. As for CNN, we make many attempts, such as utilizing Polar Coordinate Processing, employing PDOs, adding CSW and introducing masks for selectively image processing, to improve the design of the models from different perspectives and compare the results in the experiment. Although some related studies exist, most of them focus on particle separation [1]. Our work is distinct in its emphasis on image fusion and particle reconstruction. We explore different concrete models and investigate the effects of varying particle emission angles during data generation.

2 Related Work

Our work is situated within the domain of multi-modal image fusion, specifically in the context of PFlow reconstruction. This task requires the fusion of images from three distinct detectors—ECAL, HCAL, and Tracker—to produce ground truth images. To address this challenge, we will first provide a brief overview of relevant image fusion techniques, with a particular emphasis on their application to PFlow reconstruction.

2.1 Image Fusion Methods

Traditional image fusion methods rely on mathematical transformations to manually analyze activity levels and design fusion rules in either the spatial or transformation domains. These approaches, often referred to as traditional fusion methods [2–5], face significant limitations, such as crude feature fusion strategies and inadequate differentiation in feature extraction [6]. Consequently, there has been a growing trend towards the adoption of deep learning methods in this field. For example, CNNs offer significant potential for image fusion, as they can jointly implement image transformations, activity level measurements, and fusion rules (or portions thereof) through the learned representations within CNN [6].

2.2 Image Fusion Methods Applied to PFlow Reconstruction

In the specific context of PFlow reconstruction, deep learning algorithms have also seen widespread application, particularly those based on calorimeter images, such as CNNs, GNNs, and DeepSets. Among these, GNNs have emerged as the predominant deep learning architecture applied in particle physics. At present, the most commonly used deep learning networks in particle physics are GNNs. Theoretically, particles and their interactions are well-suited to being modeled as graph nodes and edges. In practical applications, GNNs have demonstrated strong performance across a range of high-energy physics tasks, including charged particle tracking [7], jet classification [8], and clustering [9].

In the subsequent sections, we will explore the capabilities of various deep learning algorithms, including those traditionally used in image segmentation, such as U-Net, to address the specific challenge of multi-modal image fusion within the context of PFlow reconstruction.

3 Simulated Dataset

This study utilizes a dataset generated through Monte Carlo simulations to replicate the behavior of particles passing through various detectors (Tracker, ECAL, and HCAL) in high-energy physics experiments. We have adjusted the simulation to reflect two key contributions: modifications to the magnetic field and the implementation of an angle wrapping strategy.

3.1 Dataset Overview

The dataset includes 10,000 sets of Trkp, Trkn, ECAL, HCAL, and Truth images, each with a resolution of 56×56 pixels. Trkp and Trkn represent the trajectories of positively and negatively charged particles recorded by the Tracker detector system, respectively. All images are stored in TIFF format.

3.2 Simulation Parameters

We modified the magnetic field affecting the deflection of charged particles traversing the Tracker, ECAL, and HCAL layers.

3.3 Angle Wrapping Strategy

The azimuthal angle ϕ represents the direction of a particle's path in the transverse plane relative to the central axis of the detector, typically constrained to $[-\pi, \pi]$. Magnetic field deflections may cause ϕ values to exceed this range. To preserve data integrity, we implemented an angle wrapping strategy ensuring ϕ values remain within $[-\pi, \pi]$.

Control data sets were generated:

1. No angle wrapping (labeled as 'No').

2. Wrapping ϕ to $[-\pi, \pi]$ (labeled as 'Yes') using:

$$\phi_f = (\phi_i + \pi) \bmod (2\pi) - \pi$$

where ϕ_i and ϕ_f are the initial and final angles, respectively.

Experiments evaluated the impact of angle wrapping on model performance.

3.4 Image Storage

After simulating each event, the system generates and stores five main types of images: Truth, Trkp, Trkn, ECAL, and HCAL. These images are saved in TIFF format.

The simulation code used in this study has been uploaded to a GitHub repository, making it available for other researchers to reference and replicate. The code can be accessed at `https://github.com/Robin0526/Neural-Networks-for-Particle-Reconstruction-in-High-Energy-Physics-Detectors`.

4 Experiment Procedure

This section aims to compare the performance of various Neural Networks in particle energy reconstruction tasks and to investigate the specific impacts of deflection and angle wrapping strategies on model performance.

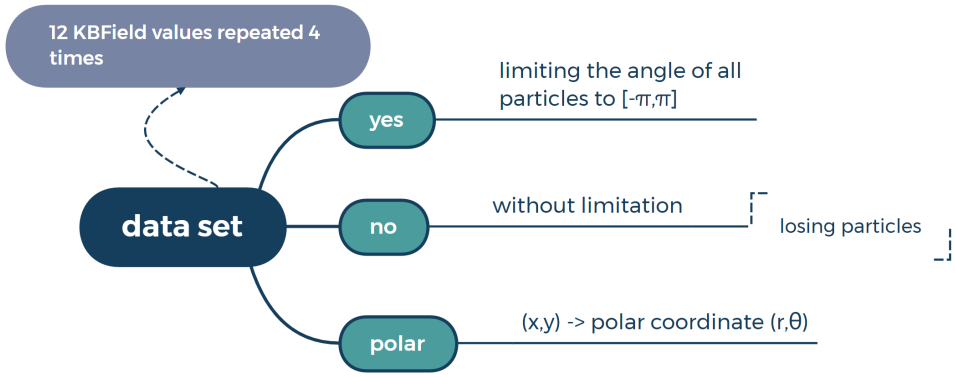

Fig. 1. Composition of the datasets

4.1 Data Generation and Preprocessing

Figure 1 illustrates the composition of our dataset. Data were generated under three distinct conditions labeled as 'Yes', 'No', and 'Polar'. For each condition, data points were created corresponding to 12 different magnetic field strengths: 0.00, 0.05, 0.10, 0.25, 0.35, 0.50, 0.75, 0.90, 1.00,

1.20, 1.40, and 1.50. To enhance data diversity and robustness, each data point was replicated four times.

The input data includes four types of images: Trkn, Trkp, ECAL, and HCAL, with the corresponding Truth values serving as labels. Each image type undergoes independent min-max normalization. The processed dataset is then randomly shuffled and divided into training, testing, and validation sets in a 7:2:1 ratio.

4.2 Training Process

We conducted comparative experiments on 'Yes' and 'No' datasets. For each dataset, we evaluated the performance on three types of models: MLP, U-Net, and CNNs (A broad category of CNN models, including several variants). These models will be detailed in Section5.

All experiments were conducted using the TensorFlow framework with CUDA acceleration. The training parameters were set to 100 epochs, an initial learning rate of 0.001, and a batch size of 32. The loss function used was Mean Squared Error (MSE), optimized using the Adam algorithm [10].

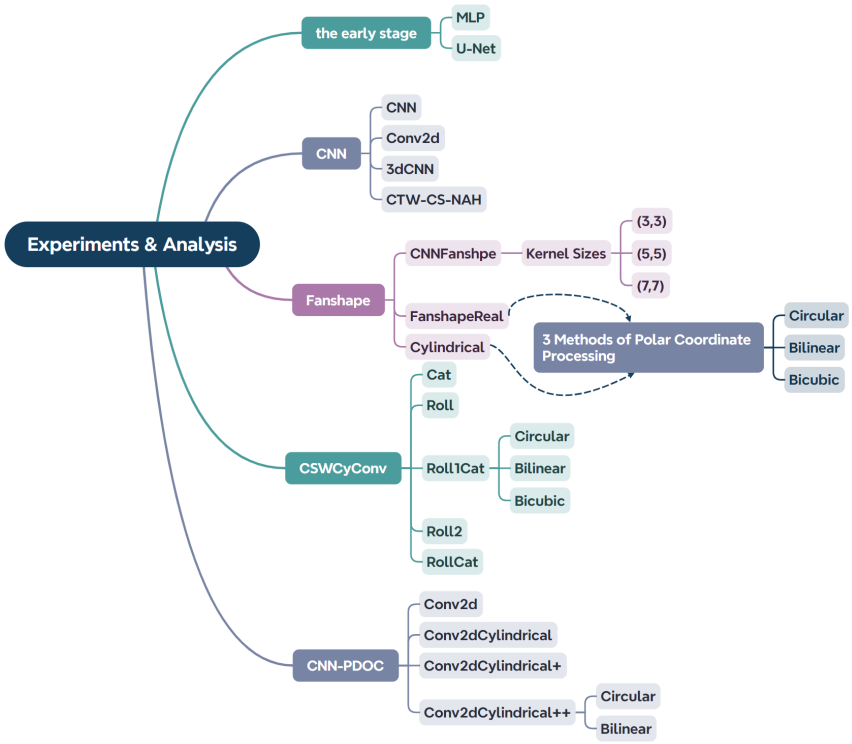

Fig. 2. Composition of the experiments

4.3 Evaluation and Visualization

Model performance was evaluated by calculating the standard deviation (STD) of the pixel-wise differences between the predicted and true values. A smaller STD indicates better reconstruction accuracy.

Figure 2 illustrates the composition of the experiments, focusing on the evaluation of different models and angle wrapping strategies. Initially, we summarized the performance of different models under varying magnetic field strengths using line plots to compare their STD values. Subsequently, for CNN models, we further analyzed the impact of angle-wrapping strategy by comparing the trends in STD values.

5 Experiments and Analysis

In this section, we present a variety of deep neural network models developed to reconstruct particle properties in high-energy collision events accurately. Each model is tailored to handle specific aspects of the data, from simple scenarios with no magnetic field deflection to more sophisticated approaches involving multi-dimensional convolutions and graph-based learning. In this study, we investigate the model reconstruction outcomes across two datasets: one with angle wrapping strategy and another without. Given that particles emitted out of the angle range of $-\pi$ to π are omitted in the reconstruction prior to specification, we anticipate superior reconstruction performance for the dataset incorporating angle specifications, which indicates that the model effectively captures the inherent periodicity of emission angles.

5.1 MLP

To better understand the characteristics of the dataset, we begin with a simple scenario where KBField $= 0$ (indicating zero magnetic field). In this case, simulated particles do not experience any deflection, meaning the pixel points in the Trkp, Trkn, ECAL, and HCAL images correspond directly to those in the truth images. Therefore, we use the pixel values at corresponding positions from the first three images to predict the pixel values of the truth image.

Given the characteristics of the dataset, we chose the MLP model for our initial attempts. This choice is due to its simplicity, ease of implementation, and suitability for handling such structured input data. Additionally, the MLP model offers computational efficiency, allowing for quick processing and training. While MLP itself is a simple model, it can serve as a building block for more complex models.

5.1.1 Framework of MLP Model

The architecture MLP model is structured as follows. The input layer consists of four nodes, each corresponding to a pixel value extracted from the Trkn, Trkp, ECAL, and HCAL images at the same (x, y) coordinate. The model includes three hidden layers, comprising 256, 128, and 64 neurons, respectively, all of which utilize the Rectified Linear Unit (ReLU) activation function. These hidden layers are specifically designed to capture essential patterns within the data. Finally, the output layer

generates a regression prediction that corresponds to the true value of each pixel, enabling accurate reconstruction of the target variable.

5.1.2 Model Assessment

We employ various metrics to assess the model's effectiveness, as illustrated in the three figures below.

Figure 3 displays a 2D histogram of true versus predicted values, excluding instances where the true values are zero. The majority of points align closely along the diagonal, indicating a strong model fit. Figure 4 presents a scatter plot of predicted versus true values, showing a clustering around the horizontal zero line, which further supports the model's accuracy. Additionally, Figure 5 provides histograms of the differences, including zero values.

However, the MLP model performs poorly in the presence of a magnetic field bias. To achieve image reconstruction under such conditions, we require a more sophisticated model. Therefore, we adopted the U-Net model.

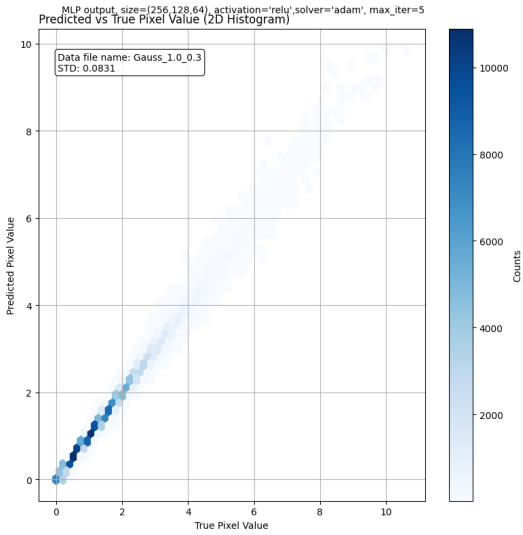

Fig. 3. Hist2d of true vs predicted (excluding zero)

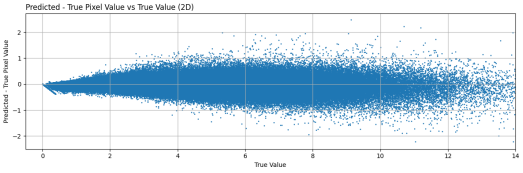

Fig. 4. Scatter plot of predicted vs true

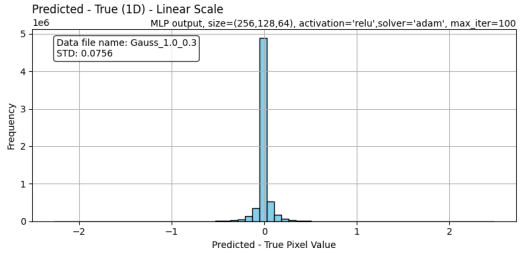

Fig. 5. Histograms of predicted-true

5.2 U-Net

5.2.1 Framework of U-Net Model

The U-Net model is distinguished by its symmetric encoder-decoder structure and skip connections between the encoding and decoding paths. These skip connections effectively preserve high-resolution spatial features, ensuring that critical detail information is not lost during the image

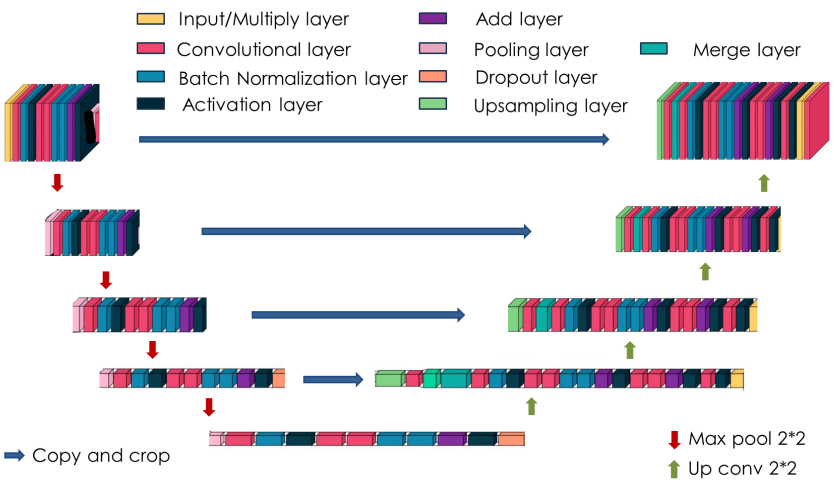

Fig. 6. U-Net Architecture

fusion process. Although the U-Net model is not traditionally used in the field of image fusion, we have attempted to construct a U-Net model for particle reconstruction.

The input to the U-Net model is a four-channel image with dimensions of (56, 56, 4), where each channel corresponds to image data from the ECAL, HCAL, Trkn, and Trkp detectors, respectively. The model architecture, as illustrated in Figure 6, consists of three main components: the encoder path, the bottleneck layer, and the decoder path, incorporating residual blocks and attention mechanisms.

In the encoder path, a series of residual blocks are used, each composed of two convolutional layers followed by batch normalization and ReLU activation. The introduction of residual blocks enhances the model's feature extraction capabilities, enabling it to better handle the complex feature combinations present in multi-channel image fusion. At the network's core, the bottleneck layer represents the deepest part of the architecture, containing the highest number of filters, and serves as a crucial connection between the encoding and decoding paths. The decoder path gradually restores the spatial dimensions through upsampling layers, integrating attention blocks at this stage, allowing the network to focus on the most relevant parts of the feature maps. The final output is generated through a convolutional layer with a linear activation function, producing a single-channel output for the image reconstruction task.

Figure 7 demonstrates the error distribution of the U-Net model under both unbiased and biased conditions (with magnetic field strengths of 0.50 and 0.80). The distributions appear to follow a Gaussian pattern. Under biased conditions, the distributions flatten and become wider, indicating a decline in model performance.

However, the overall STD of the U-Net model remains high, leading to suboptimal reconstruction performance. Consequently, we developed a series of CNN models to further enhance the

performance.

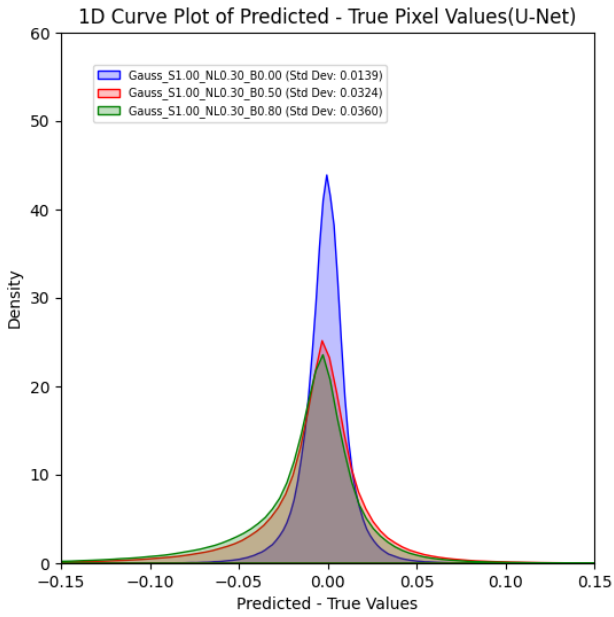

Fig. 7. Error Distribution of U-Net on Different KBField

5.3 CNN

From the perspective of the entire experiment, this section presents four models in ascending complexity: a conventional CNN, a 3D convolutional neural network (3D-CNN), a hybrid model (CTW-CS-NAH) that integrates cylindrical translation windows (CTW), cylindrical near-field acoustic holography (CSA-NAH) and 3D-CNN, and a PyTorch-based deep learning model utilizing nn.Conv2d. Below is a detailed description of the four models employed in this section:

5.3.1 Overview of Four Models

Conventional CNN The traditional architecture starts with two convolutional layers, each containing 32 filters of size 3×3, which are designed to capture local features such as edges and textures. The following third convolutional layer contains 64 filters of size 5×5, aimed at capturing more complex and larger-scale patterns in the images. Then another convolutional layer with 32 filters is included in the final part for further feature refinement, followed by a single-filter convolutional layer of size 1×1 to generate the final reconstructed image. This network is designed for the purpose of efficiently

integrating information from four channels. In addition, it is able to capture features in the images to achieve accurate reconstruction.

nn.Conv2d This model implements a standard CNN based on PyTorch for processing 2D image data. Through multiple convolutional and upsampling layers, the model extracts the characteristics of the images and generates the output. With the help of custom dataset classes and data loaders, the code can load and process large-scale image data. The Adam optimizer and mean squared error loss function ensures that the model notices the features in the images. Finally, the model's predictions and ground truth values for the test set are saved as NumPy arrays for subsequent analysis and evaluation.

3D-CNN Traditional CNNs mainly focus on learning features in the planar space, which possibly limit the ability to fully capture the spatiotemporal relationship present in the data. Therefore, we develop a 3D-CNN model to break the limitations of 2D convolutional networks. An additional dimension is introduced, treating the raw data as volumetric and applying 3D convolution operations. Specifically, the network begins with two 3D convolutional layers that both contain 32 filters of size 3×3×3, which is followed by another 3D convolutional layer with 64 filters, also using 3×3×3 filter sizes, in order to enable the model to capture more complex and multi-scale patterns in the volumetric data. The final convolutional layer applies a 3×3×3 filter to compress the feature map into a single channel, so it can preserve the volumetric format instead of flattening or reshaping it into 2D. This architecture successfully improves the function of the model to learn complex spatial patterns in volumetric data, contributing to prediction accuracy.

CTW-CS-NAH Combined with CTW and stacking to handle 3D data, this model applies a 3D convolutional neural network-based autoencoder model. The input data is mapped to a cylindrical coordinate system with the introduction of the CTW transformation, so it is capable of capturing geometric information in 3D space. This is followed by 3D-CNN-based autoencoder layers for feature extraction and reconstruction. With the help of cylindrical near-field acoustic holography (NAH), which is referred as CS3C-NAH, the model is capable of dealing with complex structures and geometric features in 3D data as well as helping improve robustness. This innovative method allows the model to better adapt to the characteristics of 3D data[11].

5.3.2 Results Analysis

Initially, we experimented with both the CNN and nn.Conv2d models, generating comparative plots of their STD values with and without controlling phi within the range $[-\pi, \pi]$, and comparing the STD images when both models were set to 'Yes'. The results are presented in figs. 8 to 10 and table 1.

From fig. 10 and table 1, it is clear that the nn.Conv2d model exhibits much lower STD values than the conventional CNN. We attribute this result to the superior feature extraction capability of the PyTorch-based Standard CNN model, which includes more convolutional layers, upsampling layers to improve image reconstruction resolution, and a more flexible and efficient data-loading mechanism (through CustomDataset and DataLoader). Additionally, during training, the model

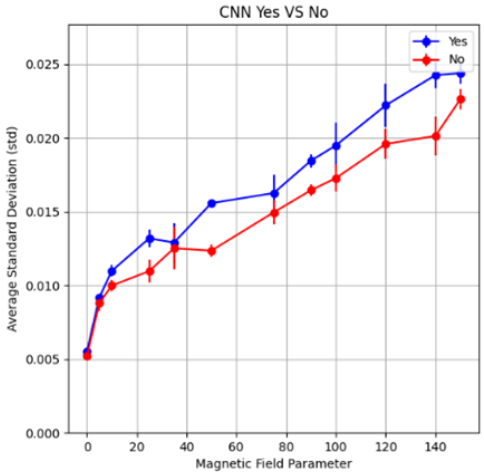

Fig. 8. CNN Yes VS No

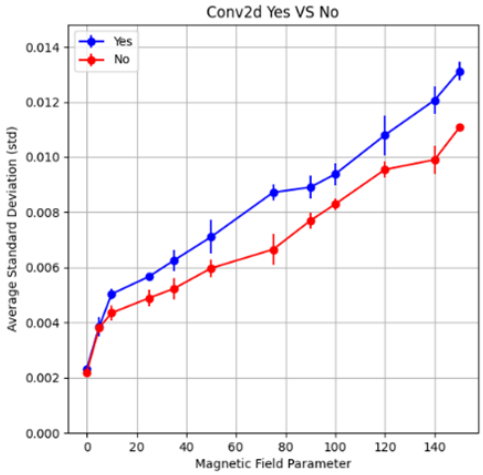

Fig. 9. Conv2d Yes VS No

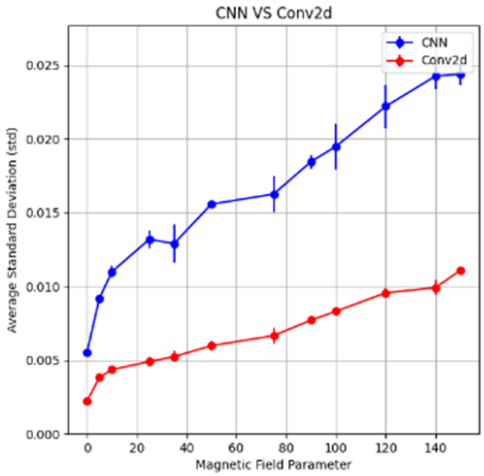

Fig. 10. CNN VS Conv2d

handled the alignment of target and predicted image sizes effectively. Furthermore, PyTorch provides greater customization and flexibility, facilitating further adjustments and experimentation, making this model more advantageous for complex tasks. However, the overall result is not ideal. Even though

Table 1: STD Comparison of CNN and Conv2d

	CNN	Conv2d
0	0.005491874	0.0023165008
5	0.009711155	0.003498844
10	0.010969961	0.0050339295
25	0.013173911	0.0056292927
35	0.012891463	0.0062466157
50	0.015354893	0.0071801706
75	0.016224669	0.008718811
90	0.018442026	0.009806737
100	0.019447459	0.009883293
120	0.022187682	0.0107982615
140	0.022474126	0.012066324
150	0.024374973	0.013125393

the nn.Conv2d model produced generally lower STD values, the 'Yes' values were consistently higher than the 'No' values, indicating that neither model successfully learned the transformation expressed by the code "phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi". This led us to explore other approaches. Considering that 3D CNNs have significant advantages over traditional 2D CNNs in processing 3D data, particularly in capturing local features and structural information in 3D space, we constructed 3D-CNN and CTW-CS-NAH models. While both models employed 3D convolutions, CTW-CS-NAH theoretically offers additional advantages by transforming the input data into a cylindrical coordinate system, allowing it to better capture geometric and structural information in 3D space. This transformation effectively handles the complex structures and geometric features in 3D data, improving model performance and robustness. Moreover, by better utilizing the geometric information in 3D data, the model typically exhibits lower loss values and higher prediction accuracy during training and testing. Below are STD comparison plots with and without controlling phi within the range $[-\pi, \pi]$, along with the complete comparison of both models.

From fig. 13, it is evident that the 3D-CNN model has lower STD values but exhibits noticeable oscillations. The CTW-CS-NAH model shows higher STD values, but the gap between the 'Yes' and 'No' plots is smaller compared to the 3D-CNN model, with the 'Yes' and 'No' images being closer to each other. Furthermore, both models exhibit regions where the 'Yes' STD is lower than the corresponding 'No' STD, indicating that both models achieved the target in certain ranges. We believe the higher STD observed in the CTW-CS-NAH model may be attributed to the increased complexity and uncertainty introduced by the cylindrical transformation. This transformation can lead to significant changes in the input data distribution, particularly where certain features (such as image edges) may be stretched or compressed during the transformation. This introduces uncertainty at the input layer, making it difficult for the model to quickly converge to a stable solution, thus increasing output fluctuations and, consequently, the STD. Additionally, the NAH method is designed to handle nonlinear and uneven signals, and its adaptive smoothing based on local features could also

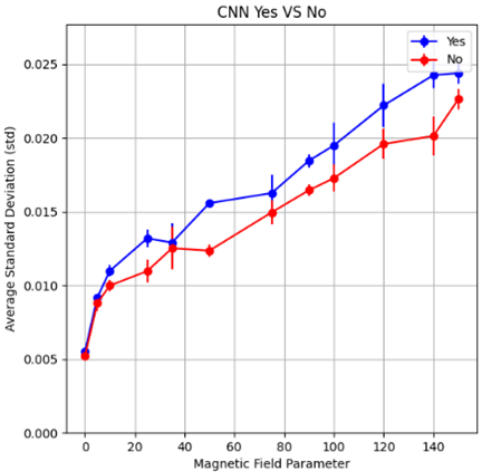

Fig. 11. 3DCNN Yes VS No

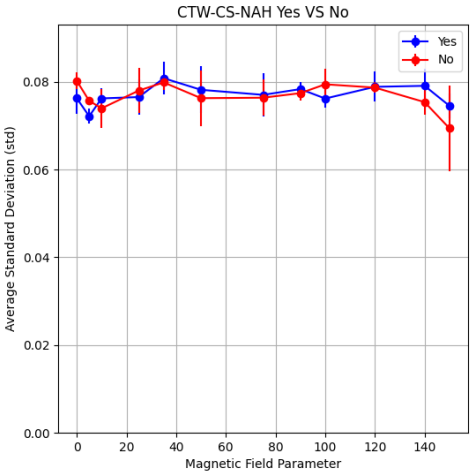

Fig. 12. CTW-CS-NAH Yes VS No

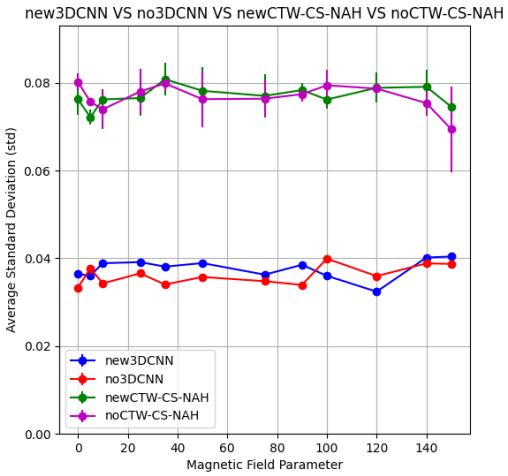

Fig. 13. new3dcnn VS no3dcnn VS newtwcsanah VS noctwcsanah

increase uncertainty in the output, especially in cases where the input data contains noise or uneven feature distribution. These factors collectively contribute to increased output variability, resulting in higher STD. In contrast, the 3D-CNN model is more stable during training due to its focus on

extracting local features through convolutional layers, thus yielding lower STD.

Finally, we plotted all the 'Yes' images from each model for comparison and summary.

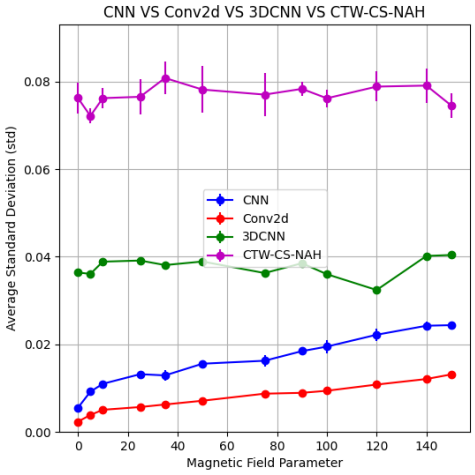

Fig. 14. CNN VS Conv2d VS 3DCNN VS CTW-CS-NAH

From fig. 14, it is clear that while neither the CNN nor nn.Conv2d models successfully learned the transformation "phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi", we found that nn.Conv2d was highly effective at reducing STD. During our experiments, the loss of nn.Conv2d dropped to 0.00005 after just 10 epochs, which is an impressive performance. For the 3D-CNN and CTW-CS-NAH models, although both exhibited relatively higher STD values, especially the CTW-CS-NAH model, we believe they show greater potential for learning to control phi within the range $[-\pi, \pi]$. Unlike the previous two models, the 'Yes' and 'No' images for these models display an interwoven pattern, and the STD does not increase with increasing kBField; instead, it tends to decrease as kBField increases, which is an interesting observation.

5.4 Fanshape and Cylindrical

We refer to the work presented in [12], which aims to more effectively capture the rotational and spiral structures of tropical cyclones. The model proposed in that study first converts satellite images of tropical cyclones from Cartesian to polar coordinates and then employs a CNN for feature extraction. Given that the LHC experiment, which our study is based on, involves a cylindrical geometry similar to, but not identical to, the structure of tropical cyclones, We have progressively developed three innovative models—Fanshape, FanShapeReal, and Cylindrical—drawing on the insights from this work. The following sections outline the methods applied in these models and provide a brief description of each.

5.4.1 Image Processing in Polar Coordinates

In this experiment, all models use polar coordinate transformation to process images. The purpose of introducing polar coordinate transformation is to better handle rotationally symmetric data, thereby improving the performance of the model in specific tasks. Through polar coordinate transformation, the model can more effectively capture the rotationally symmetric features in the image.

5.4.2 Introduction of Masks

Masking is a key technique in image processing. By selectively processing specific areas of the image, the model's ability to capture specific features can be enhanced. After applying a mask, the convolution operation can be effectively restricted to specific areas of the image, allowing the model to focus on areas of interest. This technique is particularly useful when processing data with special geometries. For example, when processing typhoon images, masks can help the model extract image features more effectively, thereby improving the performance of the model [13].

5.4.3 Overview of Three Models

Fanshape In this model, we define a fan-shaped convolutional layer (FanShapeConv2D), where convolution kernel weights are created in the build method, but no fan-shaped mask is applied. In this method, we do not use masks for convolution operations.

FanShapeReal A fan-shaped mask (Fan_mask) is created, and in the call method, this mask is used to apply masking to the convolution kernel. The convolution kernel is masked by the fan-shaped mask before being applied. Therefore, only the weights within the fan-shaped region are used.

Cylindrical In this model, we define a cylindrical convolutional layer (CylindricalConv2D). A cylindrical mask (Cylindrical_mask) is created in the build method, and in the call method, this mask is used to apply masking to the convolution kernel. The cylindrical mask is created based on radius constraints. This ensures that the convolution kernel is effective only within the cylindrical region.

5.4.4 Results Analysis

The size of the convolution kernel can affect the model's performance and training. Larger kernels expand the receptive field, enabling the model to capture broader features. This is beneficial for processing global structures and contextual information in images. However, larger kernels also increase the number of parameters, which can make the model more complex and prone to overfitting. Conversely, smaller kernels have a limited receptive field, which is more suitable for capturing local features, making them useful for detailed image analysis. Smaller kernels also reduce the number of parameters, leading to a simpler model that is easier to train but may struggle to capture broader features. To determine the most suitable convolution kernel, we experimented with kernel sizes of (3,3), (5,5), and (7,7) across the three models and generated comparative results for each:

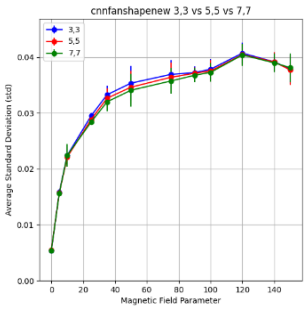

Fig. 15. Fanshape (3,3) VS (5,5) VS (7,7)

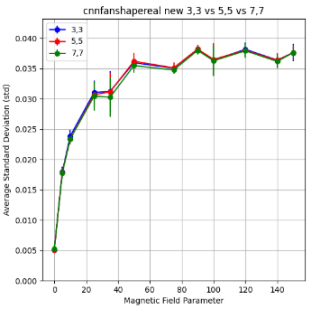

Fig. 16. FanshapeReal (3,3) VS (5,5) VS (7,7)

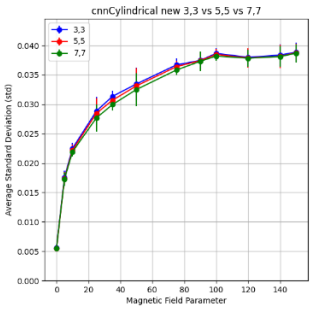

Fig. 17. CNNCylindrical (3,3) VS (5,5) VS (7,7)

From the figs. 15 to 17, we can draw a similar conclusion: the STD is highest for the (3,3) kernel, moderate for the (5,5) kernel, and lowest for the (7,7) kernel. Since the (5,5) kernel provides a balance between computational complexity and feature extraction capability, capturing features over a sufficient range without excessive information loss, we ultimately chose this kernel for subsequent experiments. We plotted comparative graphs and STD tables for the Fanshape, FanshapeReal, and Cylindrical models using the (5,5) kernel, with and without code to constrain the value of phi to the range $[-\pi, \pi]$. The results are as follows:

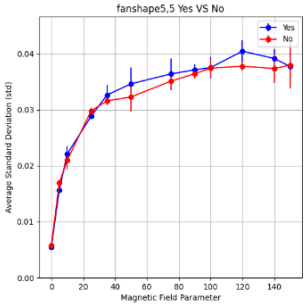

Fig. 18. Fanshape (5,5) Yes VS No

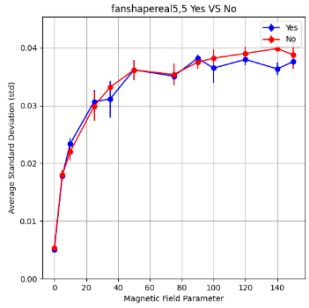

Fig. 19. FanshapeReal (5,5) Yes VS No

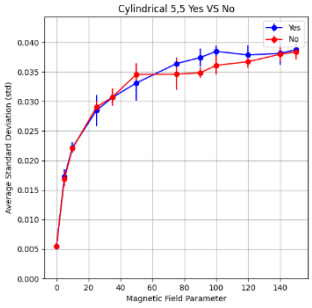

Fig. 20. CNNCylindrical (5,5) Yes VS No

From the fig. 21 and table 2, when using the same polar coordinate processing method and a (5,5) convolution kernel, we observe that although FanshapeReal and Cylindrical show some improvements over Fanshape, the STD results alone do not clearly indicate which model performs better. However, from the fig. 19, we see that, for most cases, the STD is lower when the code ”phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi” is applied compared to when it is not, which is quite interesting. This led us

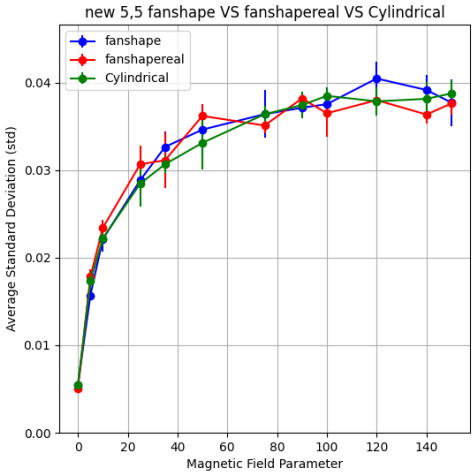

Fig. 21. Fanshape VS FanshapeReal VS CNNCylindrical with (5,5) Kernel Size

Table 2: STD Comparison of Fanshape, Fanshapereal and Cylindrical

	Fanshape	Fanshapereal	Cylindrical
0	0.0054750843	0.0050113676	0.0054712333
5	0.015689295	0.017815063	0.0173143
10	0.022113267	0.023390392	0.02218945
25	0.028838407	0.030668806	0.028454734
35	0.03264405	0.031127691	0.030673325
50	0.034627117	0.03618995	0.0331151
75	0.036390625	0.03509658	0.036384415
90	0.03712367	0.038192295	0.037428554
100	0.037532486	0.03649251	0.038483374
120	0.040464997	0.037991293	0.037856884
140	0.039155573	0.036352392	0.038140904
150	0.037717547	0.037607875	0.03875772

to further experiment and refine the FanshapeReal model, focusing our attention on improving the polar coordinate transformation method.

Since the original cyclic polar coordinate transformation method tends to lose information, preserving the accuracy of image features is particularly important in fan-shaped and Cylindrical-Conv2Ds. As convolution operations rely on the relationships between neighboring pixels, we believe that using improved polar coordinate transformation and interpolation methods can enhance the

precision of these operations. We added resampling and interpolation methods to the original code, further dividing the polar transformations into three types: the original cyclic assignment, bilinear interpolation, and bicubic interpolation. Bilinear interpolation considers the weighted average of four neighboring pixels, producing smoother results. This method effectively reduces edge effects and artifacts, improving image quality. Bicubic interpolation, on the other hand, considers a larger neighborhood of 16 pixels, resulting in finer and smoother outputs. Although more computationally intensive, it provides higher precision. However, because it takes into account more neighboring pixels, bicubic interpolation may introduce artifacts such as ringing effects, especially near the edges of the image. To evaluate the impact of these changes, we generated Yes and No comparison graphs for bilinear and bicubic interpolation, and a comparison of Yes images for cyclic assignment, bilinear, and bicubic methods, along with their corresponding STD tables. We observed several interesting phenomena:

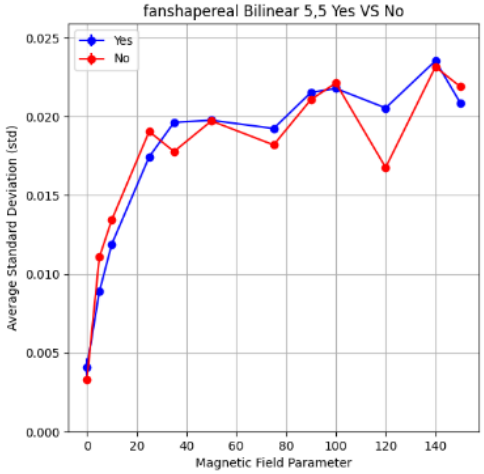

Fig. 22. FanshapeReal Bilinear Yes VS No

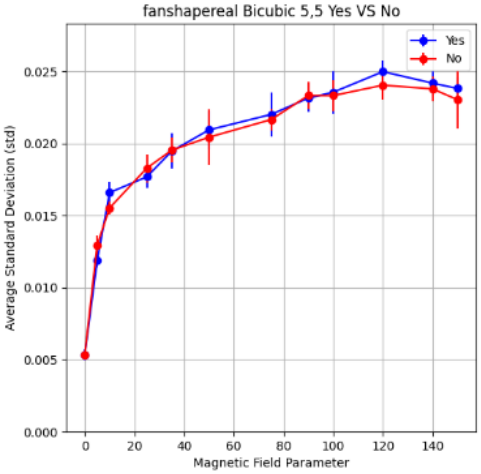

Fig. 23. FanshapeReal Bicubic Yes VS No

In the fig. 22, we notice that when using the Bilinear polar coordinate transformation, in the range of kBField = 0.05 to kBField = 0.25, the STD is lower when the code "phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi" is applied compared to when it is not. This result meets our requirements within this range. Additionally, from the fig. 24, we observe that both bilinear and bicubic interpolation methods reduce the STD compared to simple cyclic assignment, verifying that bilinear and bicubic interpolation can better smooth images and reduce noise and errors during polar coordinate transformation, thereby improving model training and prediction accuracy. In contrast, the simple cyclic assignment may cause discontinuities or jumps in certain areas of the image, introducing more errors. As a result, after using bilinear and bicubic interpolation, the experimental STD is much smaller than with simple cyclic assignment. However, when comparing bilinear and bicubic interpolation, we find that the STD

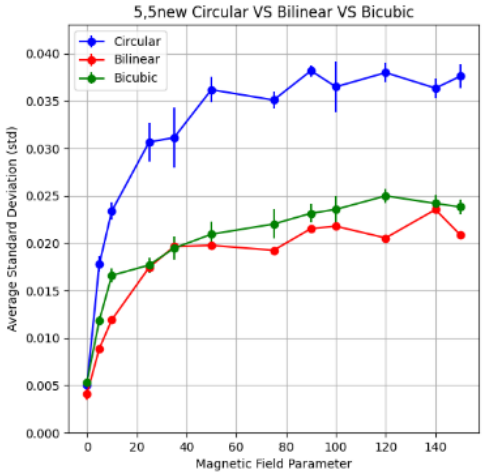

Fig. 24. Fanshape Circular VS Bilinear VS Bicubic with (5,5) Kernel Size

Table 3: STD Comparison of Circular, Bilinear and Bicubic

	Circular	Bilinear	Bicubic
0	0.0050113676	0.0040768925	0.005337935
5	0.017815063	0.008892259	0.011868012
10	0.023390392	0.011909938	0.016592793
25	0.030668806	0.017441515	0.017687324
35	0.031127691	0.019633738	0.019468237
50	0.03618995	0.019767828	0.020942265
75	0.03509658	0.019245364	0.022016276
90	0.038192295	0.021533381	0.023149991
100	0.03649251	0.021784123	0.023559364
120	0.037991293	0.020548724	0.024985474
140	0.036352392	0.023540195	0.024187502
150	0.037607875	0.02087024	0.023820218

for bilinear interpolation is lower than that for bicubic interpolation, although bicubic interpolation produces more stable and smoother images, with less oscillation. This indicates that both bilinear and bicubic interpolation have their advantages and disadvantages in image interpolation: bilinear interpolation better smooths the image, reducing noise and error, thus lowering the STD; while bicubic interpolation better preserves the overall structure and stability of the image, reducing oscillations and improving the visual quality. Although bilinear interpolation has a lower STD, its stronger smoothing

may result in the loss of certain details and edge information, potentially affecting the visual quality of the image.

In summary, although bilinear interpolation has certain shortcomings, it undoubtedly outperforms the other two polar coordinate transformation methods. Therefore, we decided to apply bilinear interpolation to another model we developed, the Cylindrical model, which uses a cylindrical network. To ensure the rigor and comprehensiveness of the experimental results, we also applied bicubic interpolation. Below are the figs. 25 to 28 generated after successfully applying these methods.

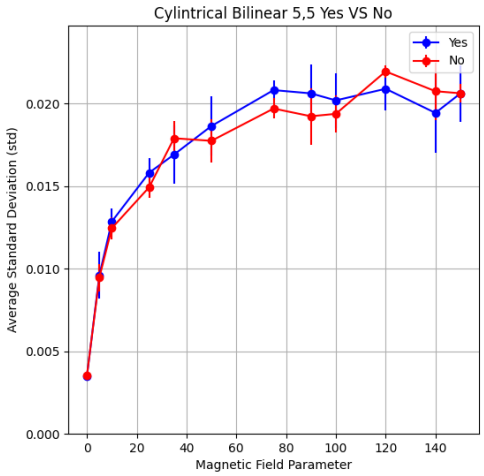

Fig. 25. Cylindrical Bilinear Yes VS No

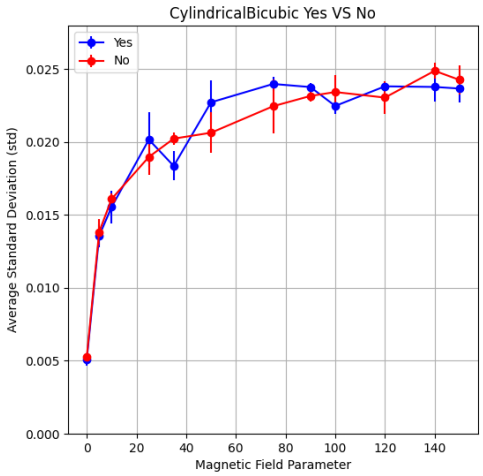

Fig. 26. Cylindrical Bicubic Yes VS No

From the fig. 27, we can observe that the superior ability of bilinear interpolation to reduce the STD has been validated once again. Additionally, the fig. 25 demonstrates that when bilinear interpolation is applied to the Cylindrical model, the STD is lower in both the kBField $>$1.00 and kBField = 0.35 conditions when the code ”phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi” is used, compared to the case without this line of code. This marks a significant improvement over the initial results in fig. 20 where the Yes images were almost entirely above the No images. Furthermore, the fig. 28 shows that when both the Cylindrical and fanshapereal models use bilinear interpolation, the Cylindrical model exhibits not only a more stable performance but also a lower STD in most cases compared to the fanshapereal model. This suggests that the cylindrical network in the Cylindrical model, when using bilinear polar coordinate transformations, is indeed better suited for the LHC experiments. Overall, while neither the Cylindrical nor the fanshapereal model fully met our expectations, both models demonstrated good performance in specific areas and can serve as a solid foundation for further research by other investigators.

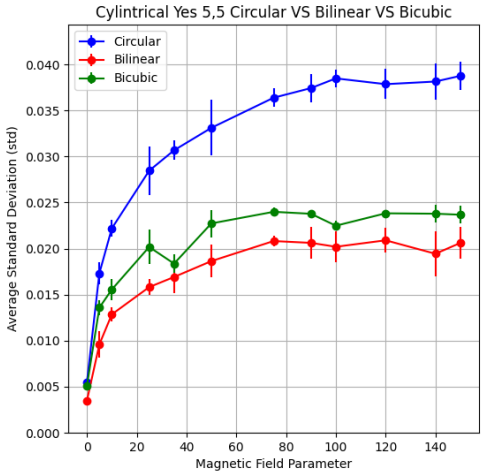

Fig. 27. Cylindrical Circular VS Bilinear VS Bicubic with Yes dataset and (5,5) Kernel Size

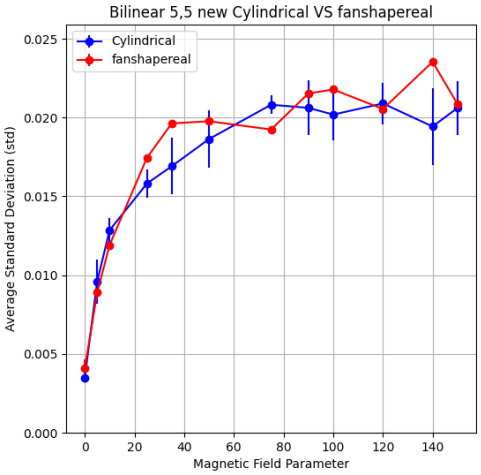

Fig. 28. Bilinear Cylindrical VS FanshapeReal with No dataset and (5,5) Kernel Size

5.5 CSWCyCnov

5.5.1 CSW Mechanism

After converting the input image from Cartesian coordinates to polar coordinates, we introduces the CSW mechanism to better process polar images. CSW is capable of simulating the effect of "rolling the image into a cylinder" by connecting the top and bottom of the input image, which not only allows the convolution kernels to maintain image continuity when crossing boundaries but also addresses the boundary issues that often happen in traditional convolution layers in polar or cylindrical coordinate systems. CSW also expands the receptive field of the boundary units, enabling the convolutional layers to extract complex features from rotated images, which improves feature extraction efficiency as a result. Moreover, this mechanism ensures the robustness of features after image rotation.

Introduction to CyConv Layer The CyConv layer is a cylindrical version compared with the traditional convolutional layer. The CyConv layer uses a convolution kernel of the input image along a cylindrical shape, which not only processes features in the central region due to the cylindrical wrapping property, but also effectively converts the rotation problem into a translation problem. Consequently, This design enables the network to better capture rotation-invariant features [14].

Introduction to Torch.cat and Torch.roll In PyTorch, torch.cat is a function used to deal with multiple tensors along a specified dimension, allowing for the formation of a larger tensor without

changing the internal structure. Torch.roll is another function used to cyclically shift the elements of a tensor along a specified dimension. By setting the shift steps and dimension, the elements of the tensor will be rearranged, which is of great use for handling cyclic data or in Recurrent Neural Networks (RNN). Both of them play important roles in deep learning model construction and data processing, providing greater flexibility in handling and manipulating tensor data.

Implementation of the CSW Mechanism Using torch.roll and torch.cat, we implemented a method to process cyclic data, which can be considered as an approximation of CSW Mechanism.

Specifically, torch.roll is added to cyclically shift the input tensor. In the CyConv2d class, the line rolled_input = torch.roll(input, shifts=-pad, dims=3) shifts the input tensor along the width dimension (i.e., the 3rd dimension), which allows the input data to "wrap around" and handle edge pixels during convolution, similar to sliding a window on a cylindrical surface. Then torch.cat connects the shifted input tensor with the results of the convolution. In the CyConv2d class, the line "combined_out = torch.cat([conv_out, rolled_input], dim=1)" concatenates the convolution output with the shifted input tensor, which makes the convolution results combined with the original input data, enhancing the expressiveness of the convolution operation. A subsequent 1×1 convolution layer is applied to adjust the number of channels in the concatenated tensor so that the output tensor has the expected number of channels.

This code uses a method to process cyclic data with the introduction of both torch.roll and torch.cat, which successfully improves the ability to handle edge information in images.

5.5.2 Model Introduction

This study used seven models in total:

Cat To implement this mechanism, we first designed a custom CNN model. The input to the model consists of four different types of image data stacked to form a four-dimensional tensor X, which is then standardized. The core of the model replaces traditional convolutional layers with custom CylindricalConv2D. The entire network consists of five convolutional layers, with ReLU activation functions following the first four layers, and the final convolutional layer producing the output.

CyConv2d achieves cylindrical convolution by performing special padding on the input tensor along the width dimension. During the forward pass, the padding size is first calculated, and then the input tensor is cyclically padded using the torch.cat function, meaning the last few columns of data are added to the front of the tensor, and the first few columns are appended to the end. The padded tensor is then subjected to standard 2D convolution. The weights of this layer are initialized using Xavier uniform distribution to ensure the effectiveness of the convolution operation.

Roll The use of torch.cat for padding the input tensor yielded suboptimal results. We hypothesize that this is because torch.cat merely manually concatenates the start and end of the input tensor, failing to adequately simulate the effect of "rolling the image into a cylinder." Therefore, we replaced torch.cat with torch.roll. torch.roll essentially performs a cyclic shift operation on the data, which intuitively aligns with the requirement of simulating a cylindrical structure.

Specifically, we used the torch.roll function in CyConv2d to cyclically shift the input tensor along the width dimension, allowing the convolution operation to process data with periodic or circular structures.

Roll2 When addressing the image fusion problem for the Emcal, Hcal, and Tracker detectors, we observed that since the detector images exhibit periodic boundary properties, rolling along a single dimension may not sufficiently capture the periodic patterns in both dimensions. Therefore, we extended the periodic convolution operation to roll along both the width and height dimensions, aiming to more accurately simulate the periodic boundary conditions in detector images.

Roll1cat Rolling the image along only one dimension may lead to information loss and boundary effects. To enhance the fusion of images, we introduced concatenation between the original input and the convolution output, followed by a 1×1 convolution layer to adjust the number of channels. This modification achieves more comprehensive feature fusion, as it preserves the global information from the original input while combining the local features extracted by the convolution. This improves the model's ability to capture complex image features and addresses the challenge of fusing images from different detectors.

Rollcat Both of the above modifications to Roll effectively reduced the model's error. Therefore, we attempted to combine these two improvements. In CyConv2d, we used torch.roll to cyclically shift the input tensor along both the width and height dimensions, simulating periodic expansion. After the convolution operation, we used torch.cat to concatenate the convolution results along the channel dimension.

Roll1catBilinear In previous iterations, we employed a nested loop method to convert images from Cartesian coordinates to polar coordinates. This method involved calculating the radial distance and angle for each pixel and mapping it to a new coordinate position. While this method is straightforward, it relied on integer indexing for interpolation, which limited its accuracy, particularly in high-precision physical detector data, leading to distortion or blurring. To improve the accuracy of the coordinate transformation, we adopted bilinear interpolation. This method calculates a weighted average of the four nearest pixels in the polar coordinate grid, allowing for more precise handling of image details and reducing the distortion caused by integer mapping. This improvement increased the smoothness between pixels during the transformation, making the image conversion from Cartesian to polar coordinates more natural.

Roll1catBicubic To further enhance the accuracy of interpolation, we ultimately adopted bicubic interpolation. Compared to bilinear interpolation, bicubic interpolation uses a larger neighborhood of pixels for weighted calculations, preserving more detail and achieving higher precision during the polar coordinate transformation. This method is particularly well-suited for handling the complex image structures present in high-energy physics detector images. It effectively reduces error during the polar coordinate transformation, ensuring the quality and consistency of image fusion.

5.5.3 Results Analysis

After generating a series of comparison images between the models for both 'Yes' and 'No' conditions, we observed several interesting phenomena. We present the images below(figs. 29 to 34).

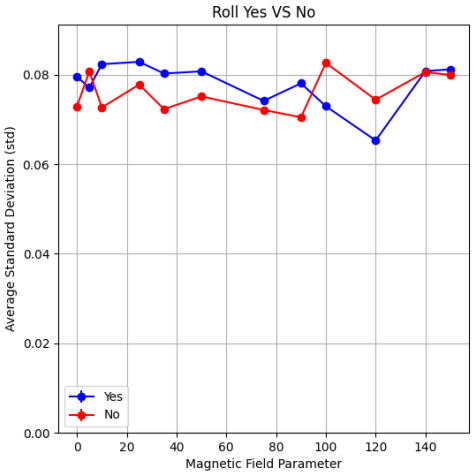

Fig. 29. Roll Yes VS No

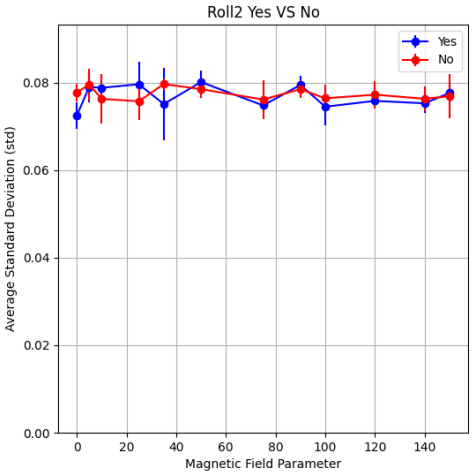

Fig. 30. Roll2 Yes VS No

Firstly, we noticed a common pattern in these images: the lines representing 'Yes' and 'No' are intertwined, with the 'Yes' STD values being larger in some instances and the 'No' values being larger in others. We believe that these models, to varying degrees, have learned some of the differences between the images generated with and without controlling the range of phi to $[-\pi, \pi]$ using the phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi code. However, an intriguing observation was made when closely examining the roll1cat Bilinear Yes vs No(fig. 33) and roll2 Yes vs No images(fig. 30)—they look almost identical. After further consideration, we believe that the cyclic shift operation of torch.roll allows the convolution kernel to "see" the other side of the image when processing edges, preventing edge information loss. Meanwhile, the torch.cat concatenation operation enhances the expressiveness of the convolution. Perhaps both methods improve the convolution's ability to process edge information, capturing similar features and thus producing similar results. That said, explaining how these features are captured theoretically remains a challenge. Additionally, when closely observing the roll1cat Bicubic Yes vs No and roll1cat Yes vs No images, they also look very similar, if not identical. To further explore this, we created a comparison plot of roll1cat Circular vs Bilinear vs Bicubic Yes images, along with a corresponding table showing the exact STD values for each point. The results are shown in fig. 35.

In fig. 35, three lines are present, but only two are visible. This is because the Circular line (blue) and the Bicubic line (green) almost completely overlap. The first and third columns in the STD table also show very close values (though not identical, which rules out the possibility of using the wrong

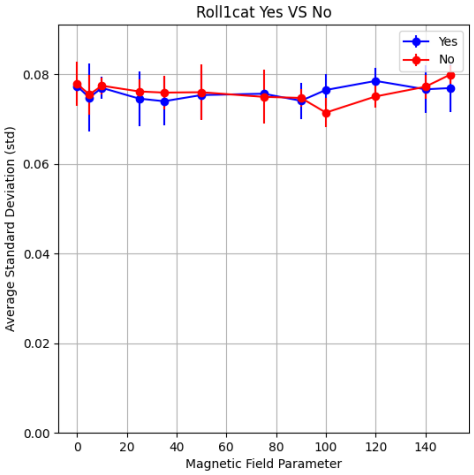

Fig. 31. Roll1cat Yes VS No

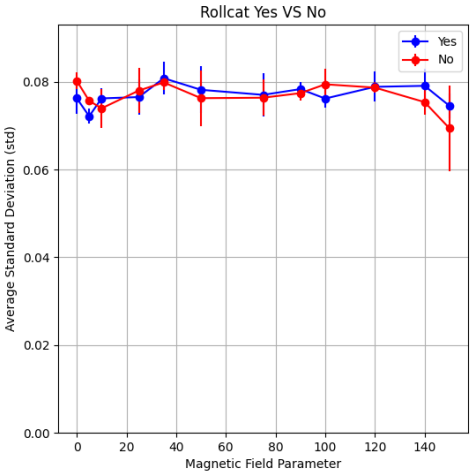

Fig. 32. Rollcat Yes VS No

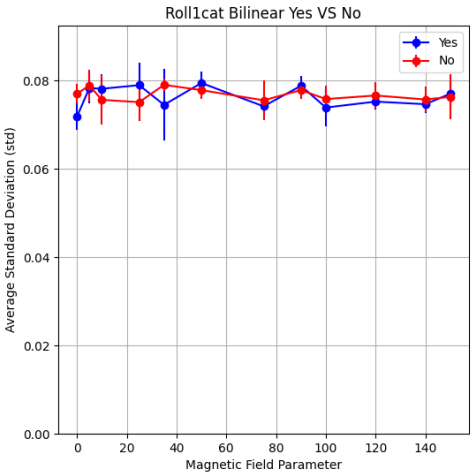

Fig. 33. Roll1cat Bilinear Yes VS No

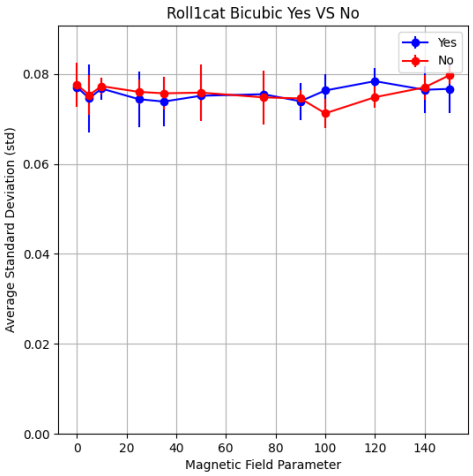

Fig. 34. Roll1cat Bicubic Yes VS No

dataset or running the wrong code). After rigorous verification, our code indeed implemented bicubic interpolation using map_coordinates. Although bilinear and bicubic polar coordinate transformations successfully reduced STD and improved model learning capacity in the earlier fan-shape category,

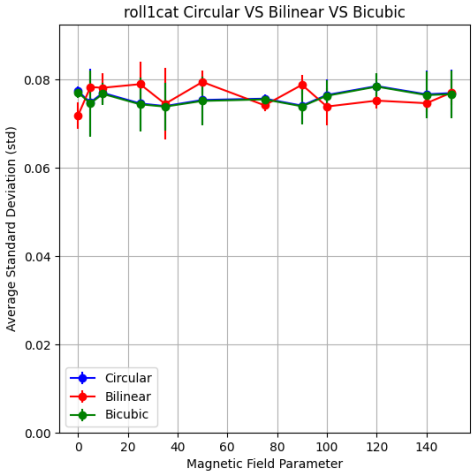

Fig. 35. Roll1cat Circular VS Bilinear VS Bicubic

Table 4: STD Comparison of Circular, Bilinear and Bicubic

	Circular	Bilinear	Bicubic
0	0.07736404	0.071788125	0.07709758
5	0.07481675	0.07831088	0.07459799
10	0.0769816	0.07814097	0.07675308
25	0.0745811	0.07896046	0.07483935
35	0.07399389	0.07449576	0.073866785
50	0.07537529	0.07494992	0.07515803
75	0.075673856	0.07416574	0.07550721
90	0.074095726	0.078812346	0.07393851
100	0.07650898	0.07388787	0.076327816
120	0.07852839	0.07522836	0.07840324
140	0.07665938	0.07464537	0.076487705
150	0.07693428	0.077027045	0.07669269

they produced unexpected and even puzzling results in this section: the final result of using bicubic interpolation in polar coordinates was almost identical to the simplest circular assignment, and the performance of roll1cat with bilinear interpolation was very similar to that of roll2. Despite searching the literature, we were unable to find an explanation for this anomaly.

In conclusion, this section's models, through the combination of CyCNN and CyConv2d, and the use of torch.roll and torch.cat, implemented enhanced convolution operations. These operations

allow the convolution kernel to ”see” the other side of the image when processing edges, preventing edge information loss. From the resulting images, it is evident that the code has, to varying extents, learned the significance of the ”phi = np.mod(phi + np.pi, 2 * np.pi) - np.pi” code in the dataset generation. However, the observed phenomena, where roll1cat with bilinear interpolation performed similarly to roll2, and roll1cat with circular and bicubic interpolations produced very similar results, raise questions. These theoretical challenges certainly warrant further investigation by researchers.

5.6 PDOs

PDOs are a novel approach in CNNs, where a linear combination of differential operators, such as gradient and Laplace operators replaces traditional convolutional kernels. PDOs are particularly suited for handling complex geometries, including unstructured grids and spherical signals, by calculating differential operators within local neighborhoods to enable efficient convolution operations [15].

PDOs significantly reduce the number of parameters required in convolutional layers, thereby improving the parameter efficiency of the model while maintaining or enhancing performance. Compared to traditional methods, PDOs are better at processing spherical or non-Euclidean data, avoiding projection distortion, and combining local and global information, which allows the model to flexibly adapt to complex data structures.

We considered introducing a convolutional kernel based on PDOs into the model, replacing the traditional convolution kernel to effectively process information captured by the detector. We designed four models: Conv2D, Conv2DCylindrical, Conv2DCylindrical+, and Conv2DCylindrical++Bilinear.

5.6.1 Overview of Four Models

Conv2D The model framework consists of several key components. First, the input data is four-channel image information (EMCal, HCal, Trkn, Trkp), which is processed by a customized PDO-Conv2D layer. This layer is based on PDOs to extract geometric information from the image by calculating unit convolution, x-direction gradient, y-direction gradient, and Laplacian features. The extracted features are further abstracted by three standard convolution layers (using 32 3×3 convolution kernels, 64 5×5 convolution kernels, and 32 3×3 convolution kernels respectively), and then a 1×1 convolution layer to generate the final single-channel output. The model is trained with the Adam optimizer and is suitable for multimodal data fusion and feature extraction tasks for complex geometric shapes.

The PDOConv2D layer is the core of the model. It performs convolution operations through unit convolution, x and y gradients, and Laplacian operators, each corresponding to a different convolution kernel. The final output is a linear combination of these results. Compared with traditional convolutional layers, PDOConv2D can capture complex geometric features more effectively, especially when processing irregular grids and multi-directional gradient information, improving the model's perception of local structures and edges.

Conv2DCylindrical In high-energy physics experiments, particle detectors such as the ECAL, HCAL, and Tracker (Trkn and Trkp) capture the movement and energy deposition of particles from

various directions, forming three-dimensional spatial information. This data typically contains three-dimensional structures, not just two-dimensional plane data. Therefore, we improved the model's PDOConv2D layer by adding z-direction gradient processing, making it more suitable for fusing images obtained from the detectors.

This model captures particle movement, energy deposition, and potential trajectory curves across the x, y, and z directions. This capability allows for a more comprehensive and accurate reconstruction of particle dynamics and energy distribution.

Conv2DCylindrical+ We introduced a dz_kernel to improve the model's performance by enhancing z-direction gradient calculations. Here, the z-direction gradient indicates changes in the "depth" of the input data, not the conventional z-axis. The z-direction gradient convolutional kernel detects differences in the image's z-direction by calculating changes between adjacent pixels, focusing on variations in the channel dimension.

The calculation of the z-direction gradient greatly enhances the model's capability to manage complex geometric structures and three-dimensional spatial data. This leads to more comprehensive and accurate predictions. It also boosts overall precision and adaptability.

Conv2DCylindrical++ and Conv2DCylindrical++Bilinear To better handle the cylindrical symmetry present in data from high-energy physics experiments, we further refined the above models. These models use bilinear interpolation to transform images from Cartesian to polar coordinates and implement a true cylindrical network by utilizing a custom CylindricalPDOConv2D layer. Specifically, the CylindricalPDOConv2D layer performs convolution operations using PDOs to extract x and y gradients, Laplace features and adds z-direction gradients (convolved through the dz_kernel). After convolution, the model applies ReLU activation functions and pooling layers to reduce the feature map size, followed by standard convolution and upsampling layers to restore resolution and generate the final single-channel output.

By converting Cartesian coordinates to polar coordinates, the model can more naturally process cylindrical data, which is particularly important for handling detector data in high-energy physics experiments. The polar coordinate system is well-suited to circumferential or cylindrical data, avoiding distortions that may occur when converting to a two-dimensional plane. The CylindricalPDOConv2D layer enables the model to capture multidimensional geometric features in the data, particularly depth information and local variations in cylindrically symmetric data.

The models are divided into two groups. The first group includes Conv2D, Conv2DCylindrical, and Conv2DCylindrical+, which, although utilizing PDO convolution to capture gradient information in different directions, still rely on standard 2D convolution and are not specifically tailored for cylindrical data or geometry. The second group includes Conv2DCylindrical++ and Conv2DCylindrical++Bilinear, which utilize the custom cylindrical convolution layer CylindricalPDOConv2D to process input images using PDOs and incorporate polar coordinate transformations to handle periodic data.

5.6.2 Results and Analysis

We first generated the respective Yes and No images for the models in the first group, as well as a horizontal comparison of the Yes images for the first three models. The figs. 36 to 38 are as follows.

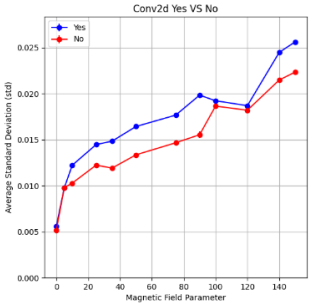

Fig. 36. Conv2d Yes VS No

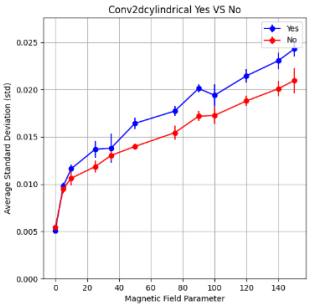

Fig. 37. Conv2dCylindrical Yes VS No

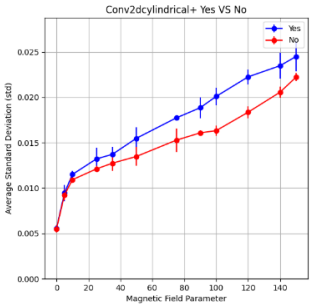

Fig. 38. Conv2dCylindrical+ Yes VS No

From the fig. 39, it is evident that although the differences in STD between the three models are generally small, the images produced by the Conv2DCylindrical and Conv2DCylindrical+ models are noticeably smoother than those from the Conv2D model. This is primarily because Conv2DCylindrical and Conv2DCylindrical+ use more PDOs, including dx, dy, dz, and Lap, in their custom convolutional layers. These operators allow the models to capture more details and edge information from the images.

Additionally, these models incorporate more convolutional and upsampling layers, further enhancing their expressive capability and resulting in smoother predictions. In contrast, the Conv2D model lacks the dz operator in its custom convolutional layer, and its overall structure is relatively simpler. This makes it less capable of capturing smooth features in the images, which results in a less smooth standard deviation map. When comparing Conv2DCylindrical and Conv2DCylindrical+, it is clear that Conv2DCylindrical+ produces the least oscillations, supporting the earlier hypothesis that z-direction gradient calculations enhance the model's ability to handle complex geometries and three-dimensional spatial data.

However, since none of these three models involve polar coordinate transformations, it remains difficult to assess whether constraining the phi between $-\pi$ and π in the original image simulation code has any meaningful impact. To address this, we conducted additional experiments using the Conv2DCylindrical++ model, which employs cyclic assignment, and the Conv2DCylindrical++Bilinear model, which utilizes bilinear polar coordinate transformation. We generated figs. 40 to 42 and table 5 for further observation.

From the figs. 40 to 42, an unexpected observation can be made: the shape of the STD lines for the Conv2dcylindrical++ and Conv2dcylindrical++Bilinear models are almost identical, with the only difference being that the STD values for Conv2dcylindrical++Bilinear are consistently lower than those for Conv2dcylindrical++. This suggests that while Bilinear interpolation reduces the STD

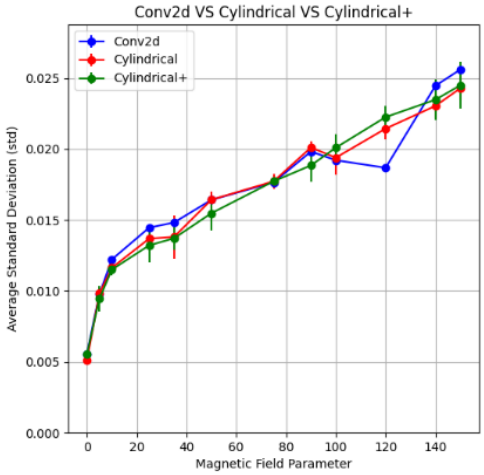

Fig. 39. Conv2d VS Cylindrical VS Cylindrical+

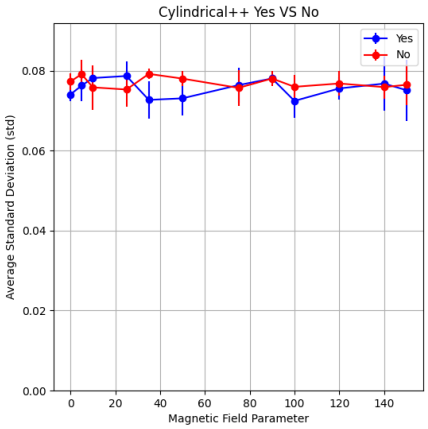

Fig. 40. Cylindrical++ Yes VS No

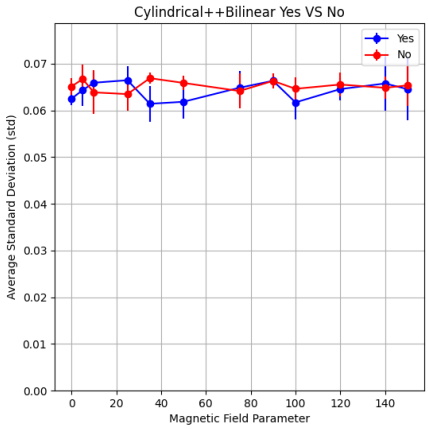

Fig. 41. Cylindrical++Bilinear Yes VS No

values, it does not provide further refinement in terms of performance improvement. Additionally, from the figs. 40 and 41, we can see that, except for the KBField range between 0.1 and 0.25 where the Yes STD is greater than No, in other regions Yes either shows a lower STD than No or both are very close. This indicates that the model has successfully learned the significance of controlling phi between $-\pi$ and π, as implemented in the imagecrafter code. It has, to some extent, captured the

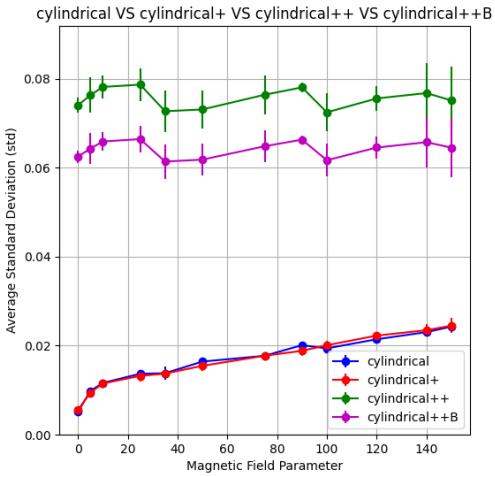

Fig. 42. Culindrical VS Cylindrical+ VS Cylindrical++ VS Cylindrical++Bilinear

Table 5: Comparison of STDs for different models

	Cylindrical	Cylindrical+	Cylindrical++	Cylindrical++B
0	0.0050794464	0.0055381292	0.07409182	0.062486786
5	0.009812146	0.009442591	0.076318786	0.06433553
10	0.011627518	0.011499854	0.07819866	0.06592123
25	0.013687121	0.01321098	0.07871339	0.066458255
35	0.013809831	0.013712202	0.072729975	0.0614118
50	0.016426623	0.01547892	0.073124886	0.061844237
75	0.017728161	0.017730132	0.07642967	0.064850524
90	0.020112518	0.018851042	0.07809214	0.066336796
100	0.019396875	0.02010132	0.07248047	0.061709754
120	0.02144609	0.022256888	0.07560325	0.06452671
140	0.023046046	0.023485519	0.07681223	0.0657614
150	0.024290603	0.024498535	0.075159684	0.064550065

difference between small particle emission ranges and the occurrence of particles on the opposite side, which is an encouraging result.

However, the STD values for both Conv2dcylindrical++Bilinear and Conv2dcylindrical++ models remain relatively high. Given that the code accurately implements all intended functionalities, we believe that further optimization and parameter adjustments could lead to versions with lower STD values. For example, in a prior experiment, reducing the batch size from 32 to 16 led to a loss

reduction to three-quarters of the value in the current version of the paper. Therefore, this approach holds considerable potential for future improvements.

6 Conclusions and Future Work

In conclusion, after developing nearly 30 models across five major categories, we have obtained several promising models that have, to some extent, learned to recognize the differences in images generated with and without constraining the value of phi between $-\pi$ and π in the imagecrafter code. For example, models such as the 3D-CNN and CTW-CS-NAH in the first category of CNN, fanshape-real, fanshaperealBilinear, and CylindricalBilinear in the second category of fanshape, all roll and cat combinations in the third category, and the Conv2dcylindrical++Bilinear and Conv2dcylindrical++ models in the fourth category. These models have overcome the challenge of consistently having the 'Yes' curves fall below the 'No' curves and have, to varying degrees, learned what it means for the shifted energy to appear on the opposite side.

Among these models, some, despite their strong performance—where 'Yes' curves are below 'No' in most cases—exhibit relatively high STD values, such as the Conv2dcylindrical++Bilinear and Conv2dcylindrical++ models. Given that the technical functionality of the code has been fully implemented (all code and experimental data are publicly available on GitHub), we believe that further tuning of parameters and settings, coupled with additional experiments, will ultimately lead to models with optimal performance and minimal STD values. Additionally, regardless of their ability to learn the significance of controlling phi, the nn.Conv2d model in the first category exhibits an extraordinary ability to reduce STD. At zero KBField, its STD is only 0.0022, which is two-fifths that of the standard CNN, and its loss quickly dropped to 0.00005 after 10 training epochs—an impressive result.

We hope that our experiments and this paper can serve as a foundation for future related research. In each experimental category, we systematically developed models with excellent functionality: some excel in 2D data analysis, others in 3D data analysis, some show outstanding performance on spherical surfaces, and others specialize in cylindrical data. We would be honored if future researchers could draw insights from our models and further develop them. Whether it leads to the creation of more effective models in other domains or the realization of models that fully comprehend the significance of shifted energy appearing on the opposite side, we would feel privileged.

Finally, our experiments stemmed from one question, but in the process, we generated many of our own questions:

- Given that in the third category, the images in fanshapereal5,5new circular vs bilinear vs bicubic show that introducing bilinear and bicubic interpolation significantly reduces STD compared to simple circular assignment, while also smoothing the images, why does the roll1cat model using Bilinear perform almost identically to the roll2 model? Additionally, why are the results for roll1cat using Circular and Bicubic interpolation so similar?

- In the fifth category, why are the STD patterns of Conv2dcylindrical++ and Conv2dcylindrical++Bilinear models almost identical, with the only difference being the STD values?

We hope that future researchers can resolve these questions, and we extend our deepest gratitude for their efforts.

References

[1] F. A. Di Bello, Sanmay Ganguly, Eilam Gross, Marumi Kado, Michael Pitt, Jonathan Shlomi, and Lorenzo Santi. Towards a computer vision particle flow. *The European Physical Journal C*, 81, 2020.

[2] Shutao Li, Bin Yang, and Jianwen Hu. Performance comparison of different multi-resolution transforms for image fusion. *Information Fusion*, 12(2):74–84, 2011.

[3] Shutao Li, Xudong Kang, and Jianwen Hu. Image fusion with guided filtering. *IEEE Transactions on Image processing*, 22(7):2864–2875, 2013.

[4] Yan Mo, Xudong Kang, Puhong Duan, Bin Sun, and Shutao Li. Attribute filter based infrared and visible image fusion. *Information Fusion*, 75:41–54, 2021.

[5] Gonzalo Pajares and Jesus Manuel De La Cruz. A wavelet-based image fusion tutorial. *Pattern recognition*, 37(9):1855–1872, 2004.

[6] Hao Zhang, Han Xu, Xin Tian, Junjun Jiang, and Jiayi Ma. Image fusion meets deep learning: A survey and perspective. *Information Fusion*, 76:323–336, 2021.

[7] Steven Farrell, Paolo Calafiura, Mayur Mudigonda, Dustin Anderson, Jean-Roch Vlimant, Stephan Zheng, Josh Bendavid, Maria Spiropulu, Giuseppe Cerati, Lindsey Gray, et al. Novel deep learning methods for track reconstruction. *arXiv preprint arXiv:1810.06111*, 2018.

[8] Isaac Henrion, Johann Brehmer, Joan Bruna, Kyunghyun Cho, Kyle Cranmer, Gilles Louppe, and Gaspar Rochette. Neural message passing for jet physics. 2017.

[9] Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, et al. Graph neural networks for particle reconstruction in high energy physics detectors. *arXiv preprint arXiv:2003.11603*, 2020.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[11] Jiaxuan Wang, Weihan Zhang, Zhifu Zhang, and Yizhe Huang. A cylindrical near-field acoustical holography method based on cylindrical translation window expansion and an autoencoder stacked with 3d-cnn layers. *Sensors*, 23(8):4146, 2023.

[12] Boyo Chen, Buo-Fu Chen, and Chun Min Hsiao. Cnn profiler on polar coordinate images for tropical cyclone structure analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 991–998, 2021.

[13] Boyo Chen, Buo-Fu Chen, and Chun-Min Hsiao. Cnn profiler on polar coordinate images for tropical cyclone structure analysis. In *AAAI Conference on Artificial Intelligence*, 2020.

[14] Jinpyo Kim, Wooekun Jung, Hyungmo Kim, and Jaejin Lee. Cycnn: A rotation invariant cnn using polar mapping and cylindrical convolution layers. *arXiv preprint arXiv:2007.10588*, 2020.

[15] Chiyu Max Jiang, Jingwei Huang, Karthik Kashinath, Prabhat, Philip Marcus, and Matthias Nießner. Spherical cnns on unstructured grids. *CoRR*, abs/1901.02039, 2019.