IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.0122113

Impact of Deep Learning Optimizers and Hyperparameter Tuning on the Performance of Bearing Fault Diagnosis

SEONGJAE LEE[1] AND TAEHYOUN KIM[1] (Member, IEEE)
[1]Department of Mechanical and Information Engineering/Smart Cities, University of Seoul, Seoul 02504, Korea
Corresponding author: Taehyoun Kim (e-mail: thkim@uos.ac.kr).

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1F1A1060231).

ABSTRACT Deep learning has recently resulted in remarkable performance improvements in machine fault diagnosis using only raw input vibration signals with no signal preprocessing. However, research on machine fault diagnosis using deep learning has primarily focused on model architectures, even though optimizers and their hyperparameters used for training can have a significant impact on model performance. This paper presents extensive benchmarking results on the tuning of optimizer hyperparameters using various combinations of datasets, convolutional neural network models, and optimizers with varying batch sizes. First, we set the hyperparameter search space and then trained the models using hyperparameters sampled from a quasi-random distribution. Subsequently, we refined the search space based on the results of the first step and finally evaluated model performances using noise-free and noisy data. The results showed that the learning rate and momentum factor, which determine training speed, substantially affected the model's accuracy. We also discovered that the impacts of batch size and model training speed on model performance were highly correlated; large batch sizes led to higher performances at higher learning rates or momentum factors. Conversely, model performances tended to be high for small batch sizes at lower learning rates or momentum factors. In addition, regarding the growing attention to on-device AI solutions, we assessed the accuracy and computational efficiency of candidate models. TICNN was the most efficient model in terms of computational efficiency and robustness against noise among the benchmarked candidate models.

INDEX TERMS Bearing fault diagnosis, Convolutional neural network, Deep learning, Hyperparameter tuning, Noise-robustness, Optimization

I. INTRODUCTION

In modern industries, rolling element bearings (REBs) are an essential part of rotating machinery operations, and 40–50 % of equipment failures are caused due to damaged REBs [1]. Because such device defects can lead to economic loss or loss of life, research on condition monitoring of REBs has continuously gained attention. Many machine condition monitoring studies have performed fault detection using data-driven vibration analysis.

The traditional data-based fault detection approach comprises four stages [2]: data acquisition, feature extraction, feature selection, and classification. The vibration signal of a rotating machine can be acquired using an accelerometer, and the sampling rate and resolution can vary depending on the performance of the accelerometer. Data characteristics are extracted in the feature extraction step by converting the collected time domain data into frequency or time-frequency domain data. Fast Fourier transform (FFT) is a representative method for converting time domain data into frequency domain data, and short-time Fourier transform (STFT) and wavelet transform (WT) are also commonly used for generating time-frequency domain data from time domain data. In the feature selection step, techniques, such as principal component analysis [3]–[5], extract

latent features from high-dimensional data. Finally, in the classification stage, machine learning (ML) methods, such as support vector machine (SVM) [4], [6], random forest (RF) [7], [8], and k-nearest neighbor (kNN) [9], [10], are used to classify machine faults. However, the disadvantage of the traditional approach is that the feature extraction and selection steps, depending on the domain expertise, may affect the fault detection performance.

Recently, deep learning has achieved remarkable performance improvements in various fields, such as image processing, natural language processing, and time series forecasting [11]. In particular, deep learning models achieve higher performance than conventional data-based approaches even when raw inputs are used without feature extraction and selection [12]. Furthermore, in bearing fault diagnosis, many researchers have obtained approximately 100 % accuracy without preprocessing using convolutional neural networks (CNN) [13]–[25], recurrent neural network (RNN) [26]–[28], and Transformer [29], [30]. Studies have also been conducted to diagnose the failures of multi-domain data measured under different working loads or to develop a model that can classify bearing faults even in a noisy environment.

Deep learning models are generally trained using the stochastic optimization method [31], and the training speed and final model performance vary significantly depending on the optimizer type and hyperparameter tuning. However, there is no theoretical basis for determining the best optimizer and hyperparameter tuning method, and empirical studies have been conducted only in representative fields of deep learning [32]. In particular, although research on model architectures has been intensively conducted in bearing fault diagnosis, there are fewer studies on the theoretical or empirical verification of the effects of optimizers and hyperparameter tuning on model performance. Furthermore, in model development studies, the description of the optimization methods and training environments used in them is often unclear.

Regarding these problems, this paper presents a bearing fault diagnosis benchmarking study of robustness against noisy data using two open-access datasets, seven CNN-based models, and four deep learning optimizers with varying batch sizes. The benchmarking and performance verification processes were as follows. First, we set the hyperparameter search space for each optimizer. We used stochastic gradient descent (SGD), Momentum, RMSProp, and Adam as optimizers. Hyperparameter set for each optimizer was sampled from a quasi-random log-uniform distribution. Further, we trained a model on each hyperparameter set and refined the hyperparameter search space for each model and optimizer. Finally, after training the models in the refined hyperparameter search space, we evaluated the results using noise-free and noisy data.

The contributions of this study are summarized as follows:

- Based on the experimental results, we confirmed that the optimizer hyperparameter tuning significantly affected the robustness of models against noise. For all models, the accuracies of results using noisy data varied substantially depending on the hyperparameter of the optimizers. In particular, the learning rate and momentum factor, the hyperparameters that determine training speed, substantially affected the accuracy of both noise-free and noisy data.
- We also noticed that the impacts of the batch size and model training speed on model performance were often correlated. For example, the model performance was relatively low in many cases if the learning rate or the momentum factor was high when the batch size was small. On the contrary, a lower learning rate or momentum factor resulted in good model performance when the batch size was large.
- Seven candidate models were assessed in noise-free and noisy environments, considering the on-device AI system with limited resources. The evaluation results showed that a convolutional neural network with training interference (TICNN) was the best model for computational efficiency and noise-robustness.

The remainder of this paper is organized as follows. Section II introduces related studies on bearing fault diagnosis, deep learning optimizer evaluation, and benchmarking studies. In Section III, we describe our benchmarking approach and the datasets, models, optimizers, and hyperparameters used in the experiments. Section IV presents benchmarking results and a review of the results. Finally, we summarize and conclude the paper in Section V.

II. RELATED WORK

A. BEARING FAULT DIAGNOSIS

Most early bearing fault diagnosis methods detected fault through features manually derived using conventional signal processing techniques, such as FFT, STFT, and WT. Randall and Antoni [33] reviewed a brief history of the bearing fault diagnosis and signal processing methods used in fault detection. Nandi et al. [1] provided vibration frequencies for each REB fault type and signal analysis of general faults that can occur in electrical motors.

In recent decades, data-driven fault diagnosis approaches using ML have gained considerable attention. In particular, classification methods, such as SVM and RF, are actively utilized for bearing fault diagnosis. In many cases, ML-based approaches are preceded by feature extraction based on signal processing techniques.

For example, Kankar et al. [34] presented a bearing fault diagnosis using SVM and an artificial neural network, whose inputs are signal features such as kurtosis and crest factors. Recently, research on extracting complex features using techniques such as wavelet transform and modal decomposition has also been conducted. In [35], Kankar et al. diagnosed failures by using wavelet decomposition and SVM techniques. Zhang et al. [6] and Qin et al. [8] extracted data characteristics using empirical modal decomposition, whereas SVM and RF were used for fault detection, respectively. Lu et al. [10] used variational modal decomposition for a kNN-based bearing fault diagnosis model.

Modern deep learning models have gained considerable attention in fault diagnosis because they can achieve high performance without prior feature extraction. Among them, CNN is the most commonly used structure, and 1D and 2D CNNs have been widely studied. The 1D CNN receives raw vibration signal inputs to diagnose device failures. Zhang et al. [13], [14] proposed a model that maximizes the robustness against noise by widening the kernel size of the first convolutional layer. Recently, fault diagnosis models that use structures, such as residual connection, dilated convolution, and capsule networks, have also been actively developed [15], [16], [18]. 2D CNN-based models use signal-to-image mapping, STFT or WT, to generate 2D signals and perform fault prediction from the signals [19], [20], [22], [23].

On the contrary, RNNs, in conjunction with CNNs, are also widely used in fault diagnosis. For example, input signals are first passed through a CNN, and then an RNN diagnoses the failure through the signals. Shenfield et al. [26] proposed a hybrid model of long short-term memory (LSTM) and a CNN-based model described in [13]. In addition, Jin et al. [27] proposed an adaptive anti-noise neural network framework using a 1D CNN, gate recurrent unit, and attention module. Currently, Transformer-based architectures have also been applied to fault diagnosis. For example, Feng et al. [30] presented a lightweight combination model of a 1D CNN and a Transformer structure, and Ding et al. [29] proposed a Transformer structure that processes time-frequency domain data.

B. DEEP LEARNING OPTIMIZERS AND HYPERPARAMETER TUNING

Most deep learning models are trained by stochastic optimization, which minimizes the loss from data in mini-batch units of a specific size. The most basic and widely used optimization algorithm for model training is SGD. The Momentum algorithm [36] was devised to improve the slow learning speed of SGD. Currently, the RMSProp [37] and Adam [38] algorithms are widely used because they adaptively change the learning rate during the training process.

The importance of properly tuning the optimizers and their hyperparameters in deep learning model training was also emphasized. Grid search is the simplest and most basic hyperparameter tuning method. It is an exhaustive method for selecting the best hyperparameter by evaluating the performance of all the combinations. Although this method is simple and easy to implement and parallelize, its search time increases exponentially even if the hyperparameter dimension increases only slightly. Random search is a widely used hyperparameter tuning method owing to its high efficiency [32], [39]–[41]. It randomly samples hyperparameter combinations within a predetermined range.

Research on hyperparameter tuning of deep learning optimizers has been conducted in various fields. However, there is no theoretically verified the best optimizer or hyperparameter tuning method [41], and benchmarking and empirical studies on specific datasets and models are predominant. Choi et al. [32] presented large-scale optimization hyperparameter tuning results for image classification and language-modeling tasks. Schmidt et al. [41] also presented hyperparameter tuning results for each optimizer in image classification, image generation, and character prediction tasks.

In addition to large-scale benchmarking studies, studies have been conducted to verify the effectiveness of deep learning optimizers and hyperparameter tuning on a specific target application. For example, Verma et al. [42] verified the performance of Adam and RMSProp optimizers based on a learning rate change in a COVID-19 classification problem using computed tomography images. Saleem et al. [43] compared the performances of six optimizers, including SGD and Adam, for CNN models applied in plant disease classification. Şen et al. [44] presented results of the grid search for the hyperparameters of the Adam optimizer in an electrocardiogram classification. With regard to fault diagnosis, Rezaeianjouybari and Shang [45] verified the effects of SGD and Momentum's cyclic learning rate and cyclic momentum on bearing fault detection.

III. METHODOLOGY

This paper presents benchmarking results of the impact of hyperparameter tuning on deep learning optimizers with noisy vibration data. Fig. 1 depicts the entire process of hyperparameter tuning and benchmarking performed in this study. We benchmarked 56 configuration options from combinations of two open datasets, seven CNN models, and four optimizers. The deep-learning model in each configuration was trained for batch sizes of 16, 64, and 128.

As the first step of our benchmarking study, we specified the optimizer's search space for each hyperparameter and performed model training 64 times. In this step, hyperparameters were sampled from a quasi-random uniform distribution on a log scale. Second, we

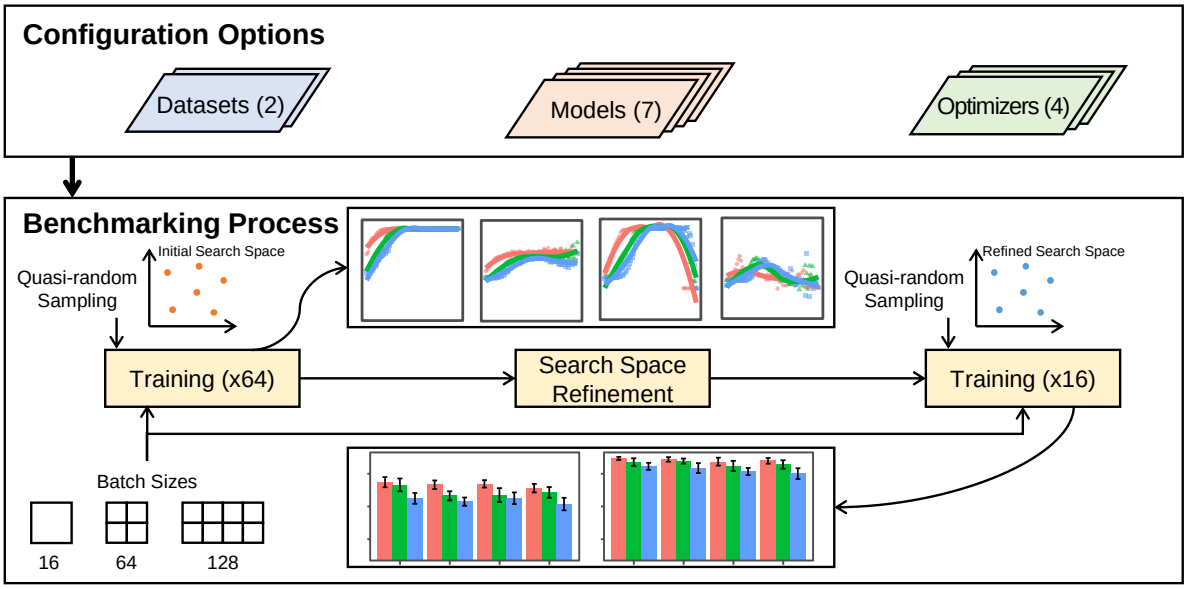


FIGURE 1. Schematic workflow of the benchmarking process.

refined the search space for each model and optimizer based on the experimental results acquired from the first step and then repeated the model training 16 times. Based on the experimental results from the second step, we evaluated the test accuracies of the noise-free and noisy data for different batch sizes and optimizers.

A. DATASETS

This benchmarking study used the Case Western Reserve University (CWRU) and Society for Machinery Failure Prevention Technology (MFPT) datasets. Both datasets are widely used in bearing fault diagnosis and provide vibration signals for various bearing faults and motor load environments.

The CWRU dataset [46] is the most widely used public dataset for bearing fault diagnosis. This dataset contains vibration data measured using an accelerometer with 12 and 48 kHz sampling rates from a 2 HP reliance electric motor. Fig. 2 shows the test rig where the dataset was collected. Two types of bearings were installed at the test rig: a 6205-2RS JEM SKF bearing at the drive end and a 6203-2RS JEM SKF bearing at the fan end. Data were collected for each bearing's inner raceway and outer raceway and ball fault situation. In addition, bearing faults were artificially synthesized using the electro-discharge machining method to include data with various crack sizes and motor loads. Our study used the drive end bearing data collected under 0.7457, 1.4914, and 2.2371 kW motor loads at 12 kHz.

The MFPT dataset [47] provides vibration data measured for normal and two types of defected bearings: inner and outer raceway faults. The vibration data were collected under nine load conditions at a sampling rate

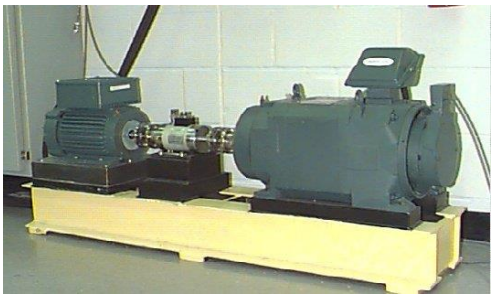


FIGURE 2. Test rig of the CWRU dataset [46].

of 97,656 Hz.

For the benchmarking study, we preprocessed the data in three stages. First, we generated data from the raw vibration signal using overlapping. We then labeled the data according to fault type. Finally, we divided the generated dataset into training, validation, and testing sets.

Both bearing datasets consist of large raw vibration signals measured over a predefined time interval under specific bearing conditions. For example, the inner raceway fault data of the CWRU dataset are composed of nine raw signals, and the length of each signal is approximately 120,000. If these raw signals are used for training directly, the size and computational burden of a model increase, whereas the number of the training data samples is insufficient. Therefore, it is common to divide the raw signals into specific sample lengths for training. For example, an augmentation technique that overlaps and slices the raw signals have often been used to secure sufficient training data. Fig. 3 shows an example of data

augmentation using overlapping.

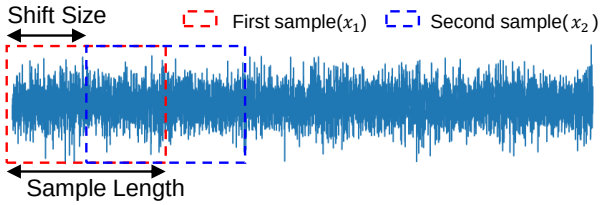


FIGURE 3. Data augmentation using overlapping.

In this study, we sliced the data with the largest input size of the candidate models, 4,096, and a shift size of 2,048. However, the models evaluated in this study have different input sizes. For this reason, to train each model using the same data, we used only the front part of the data when the input size of the model was smaller than 4,096. For example, the model that had an input size of 2,048 was trained using only the first 2,048 elements of data.

After generating the data, we labeled each sample according to the fault type. Because the CWRU dataset provided fault data for three different crack sizes and three fault types, the data were labeled with nine fault classes and a normal class. Table 1 presents the class classification method for the CWRU dataset. However, the MFPT dataset provided only two types of faults, the inner and outer raceways, and did not separate the detailed characteristics of each fault. Therefore, the data were classified into three classes: normal, inner raceway faults, and outer raceway faults. Table 2 shows the class classification method for the MFPT dataset and the number of data for each class.

Finally, to avoid overfitting, we divided the labeled data into training, validation, and test datasets with a split ratio of 60%:20%:20%. We also performed stratified sampling to ensure that the distribution of the sample ratio for each class among the different splits was the same. Tables 1 and 2 also list the number of samples for each dataset in the benchmarking study.

TABLE 1. Data labels and the number of samples for the CWRU dataset.

Fault Type	Fault Diameter (mm)	Class Label	Number of Samples		
			Train	Validation	Test
Normal	-	0	424	141	141
Ball	0.1778	1	104	35	35
	0.3556	2	105	35	35
	0.5334	3	105	34	34
Inner Raceway	0.1778	4	105	35	35
	0.3556	5	104	35	35
	0.5334	6	104	35	35
Outer Raceway	0.1778	7	104	35	35
	0.3556	8	104	35	35
	0.5334	9	104	35	35

B. CANDIDATE MODELS

Seven candidate models were employed in our benchmarking study: four 1D CNNs, two 2D CNNs, and one CNN + RNN model. Table 3 summarizes the candidate models and their concepts.

TABLE 2. Data labels and the number of samples for the MFPT dataset.

Fault Type	Class Label	Number of Samples		
		Train	Validation	Test
Normal	0	513	171	171
Inner Raceway	1	294	98	98
Outer Raceway	2	807	269	269

TABLE 3. Summary of candidate models.

Category	Model	Concept
1D CNN	WDCNN [13]	• Wide First Convolutional Layer
	TICNN [14]	• Wide First Convolutional Layer • Convolutional Kernel Dropout
	DCN [15]	• Wide First Convolutional Layer • Dilated Convolution [48] • Residual Connection [49] • Squeeze-and-Excitation Block [50]
	SRDCNN [16]	• Wide First Convolutional Layer • Dilated Convolution [48] • Residual Connection [49] • WaveNet-like Architecture [51]
2D CNN	STIM-CNN [20]	• Signal-to-Image Mapping
	STFT-CNN [22]	• STFT
1D CNN + RNN	RNN-WDCNN [26]	• Integration of WDCNN and LSTM

Because vibration signals are one-dimensional, 1D CNNs are the most popular models for bearing fault diagnosis. The main idea of early 1D CNN-based architectures was to widen the first kernel size of the convolutional layer. This technique has been broadly applied in many studies of 1D CNN-based fault detection. We used deep convolutional neural networks with wide first-layer kernels (WDCNN) [13] and TICNN [14] as representatives of the 1D CNN model.

WDCNN [13] is an early 1D CNN-based model that enhances domain adaptation and anti-noise ability for bearing fault diagnosis. This model showed that widening the kernel size of the first convolutional layer could suppress the high-frequency noise of vibration signals.

TICNN [14] is the successor of WDCNN. This model has a similar architecture to the WDCNN but uses kernel dropout for the first convolutional layer. Kernel dropout randomly drops the value of the convolutional filters. The study mentioned that the kernel dropout of the first layer could add noise to the input data. Fig. 4 shows the kernel dropout for a 1D convolutional filter of size three.

Dilated convolution and residual connection are also frequently used in fault detection CNN. Dilated convolution was originally proposed for image segmentation [48]. It can increase the receptive field of a convolutional kernel while keeping the parameter constant by creating holes in the convolutional kernel. Fig. 5(a) illustrates a 1D dilated convolution of a 3x1 filter. The receptive field size of this filter was five while only using the parameters of the 3x1 filter.

Residual connection adds the input layer values to the output layer values of a neural network by element. This technique can address the accuracy degradation with re-

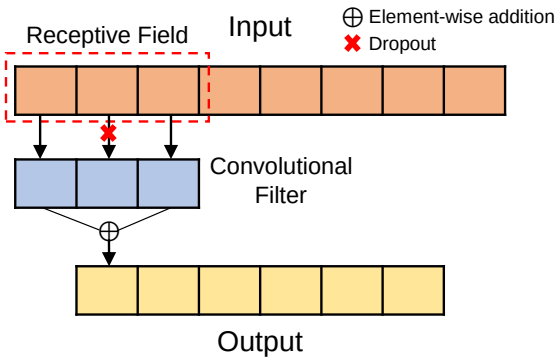


FIGURE 4. Kernel dropout of TICNN.

spect to the depth of the neural network. Fig. 5(b) shows a residual connection applied to a convolutional layer. Beginning with ResNet [49] for image classification, numerous modern CNN architectures use the residual connection. In this study, we used a one-dimensional dilated convolution network with residual connection (DCN) and a stacked residual dilated convolutional neural network (SRDCNN) as representatives using dilated convolution and residual connection.

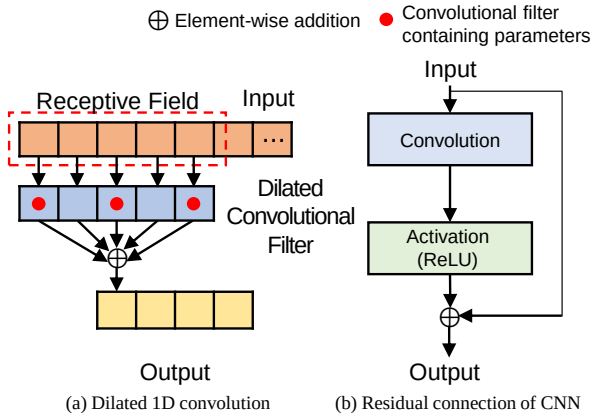


FIGURE 5. Dilated convolution and residual connection.

DCN [15] used dilated convolution and residual connection to enhance the feature learning ability. The essential blocks of DCN were the residual connection and dilated residual connection blocks. The residual connection block was a combination of a normal convolutional layer and residual connection, and the dilated residual connection block was composed of dilated convolutional layers with a residual connection. In addition, the squeeze-and-excitation block, proposed by SeNet [50], was used for the DCN architecture. The DCN effectively diagnosed the faults of two bearing datasets in multi-domain and noisy environments.

SRDCNN also used both residual connection and dilated convolution. The building block of the SRDCNN was similar to that of the WaveNet [51] architecture, which was based on the input gate layer of the LSTM and dilated convolution. Fig. 6 shows the building block of SRDCNN.

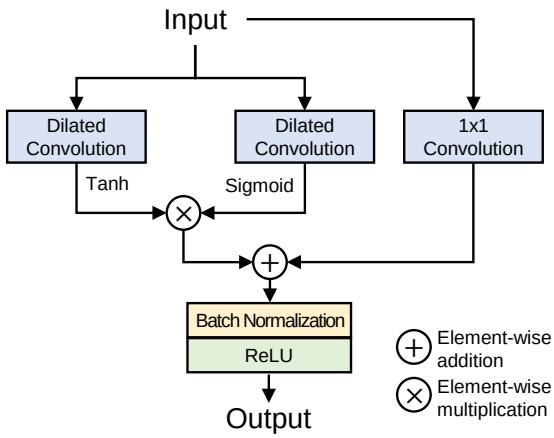


FIGURE 6. Building block of SRDCNN.

2D CNN architectures have also been intensively studied. Unlike 1D CNNs, 2D CNNs must convert inputs to 2D data because raw vibration signals are one-dimensional time-series data.

Signal-to-image mapping (STIM) is the simplest data conversion algorithm that reshapes a 1D signal to a 2D vibration image. Fig. 7(a) illustrates the STIM algorithm, which converts a 9×1 1D signal to a 3×3 2D signal. Among the previous studies of 2D fault diagnosis CNNs using STIM [20], [21], we selected the simple 2D CNN by Zhao et al. [20] (STIM-CNN) for the candidate model. The STIM-CNN had two convolutional layers and a fully connected layer that achieved higher accuracy with fewer parameters than LeNet-5.

Signal processing algorithms, such as STFT and WT, can also be used to generate 2D inputs [19], [22], [24]. These algorithms require additional computational resources to derive time-frequency domain information. Fig. 7(b) shows a 2D spectrogram generated by the STFT. This study used the model proposed by Zhang et al. [22] (STFT-CNN) as another 2D CNN candidate. The STFT-CNN used a 2D spectrogram from STFT as inputs, and the scaled exponential linear unit was used to avoid dead nodes.

Because vibration signals are a time series, RNNs have been actively applied to diagnose bearing faults [26], [27]. As the candidate model of the RNN-based architecture, we selected the RNN-WDCNN [26] as shown in Fig. 8. The model consists of two paths. One path passes the input to the WDCNN [13] directly. In another path, the input passes the convolutional layer and the LSTM layer sequentially. Finally, the two outputs from the paths are concatenated.

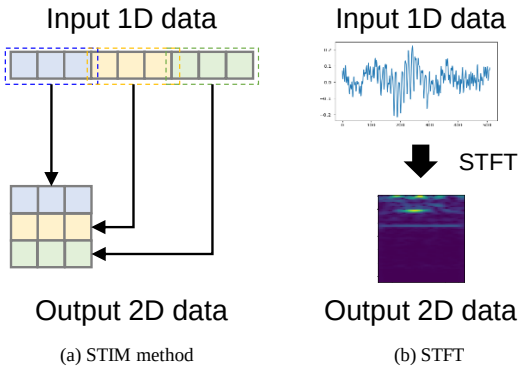


FIGURE 7. Data generation for 2D CNN.

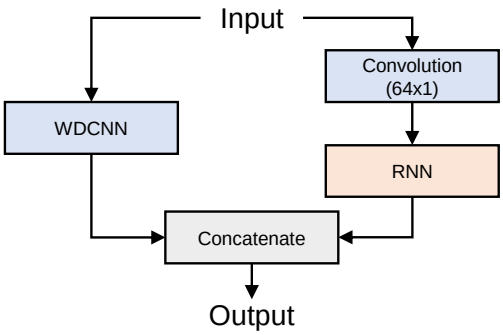


FIGURE 8. RNN-WDCNN architecture.

C. OPTIMIZATION METHODS

We used four optimizers: SGD, Momentum [36], RMSProp [37], and Adam [38] in this benchmarking study. They are the most popular optimizers for deep learning model training, which update parameters using the gradient. Before describing the parameter update rule for each optimizer, we define the model parameters, loss function, batch size, and i^{th}input as θ, $f(\cdot;\theta)$, $L(\cdot)$, n, (x_i, y_i), respectively. Algorithms 1–4 describe the optimizers' update rule.

SGD is the simplest deep learning optimization method that updates parameters in the direction where the loss function decreases the fastest, that is, in the opposite direction of the gradient. This method requires only one crucial hyperparameter, the learning rate, which determines the step size of the parameter update. However, a fixed learning rate may lead to inefficiencies in the model training. For example, a low learning rate can easily reduce the training speed. However, the models often fail to converge to the optimal loss under a high learning rate.

Momentum [36] was proposed to accelerate the training speed of SGD. It introduced the information of past gradients v to update the parameter, as described in Algorithm 2. In this algorithm, the momentum factor γ determines the contributions of the previous gradient.

Algorithm 1 SGD

Require: learning rate $\eta > 0$

repeat
  $\mathbf{g} = \frac{1}{n}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$
  $\theta = \theta - \eta\mathbf{g}$
until Converge

Algorithm 2 Momentum

Require: learning rate $\eta > 0$, momentum factor $0 \leq \gamma < 1$

$v = 0$
repeat
  $\mathbf{g} = \frac{1}{n}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$
  $v = \gamma v - \eta\mathbf{g}$
  $\theta = \theta + v$
until Converge

The larger the value of γ, the more the previous gradient affects the training. In general, frequently selected values for the momentum factor are 0.5, 0.9, and 0.99.

Learning rate is the most significant hyperparameter of optimizers; however, tuning the learning rate for a specific task is difficult. In recent studies, algorithms that adaptively change the learning rates of parameters, such as RMSProp and Adam, have been introduced to address this problem.

RMSProp uses an exponentially weighted moving average of the gradient to adapt to the learning rate. This method modifies Adagrad and performs better in non-convex environments. RMSProp has four hyperparameters: η, α, γ, and ϵ. η is the initial learning rate, and α is the smoothing factor of the exponentially weighted moving average. γ is the momentum factor, which is the same as the Momentum algorithm. ϵ is a numerical constant that prevents division error by zero.

Recently, Adam [38] has been the most popular adaptive learning rate algorithm. It uses first- and second-order moments of the gradient to adaptively change the learning rate. Adam may be regarded as a combination of RMSProp and Momentum; however, a significant distinction is that the moments of the gradient are rescaled. Adam has four hyperparameters: η, β_1, β_2, and ϵ, which indicate the initial learning rate, smoothing factors of the first- and second-order moments, and numerical constant, respectively. The commonly used default values for the hyperparameters are $\eta = 0.001, (\beta_1, \beta_2) = (0.9, 0.999)$, and $\epsilon = 10^{-8}$.

D. MODEL TRAINING AND HYPERPARAMETER TUNING

As described in the previous sections, we performed a two-stage benchmarking test for $2 \times 7 \times 4 = 56$ different configuration options using two datasets, seven models, and four optimizers. In the first stage, we trained the models 64 times for each optimizer and batch size in the

Algorithm 3 RMSProp

Require: learning rate $\eta > 0$, smoothing factor $0 \leq \alpha < 1$, momentum factor $0 \leq \gamma < 1$, numerical constant ϵ

$v = 0$, $m = 0$
repeat
  $\mathbf{g} = \frac{1}{n}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$
  $v = \alpha v + (1-\alpha)\mathbf{g} \bigotimes \mathbf{g}$
  $m = \gamma m + \frac{\eta}{\sqrt{v}+\epsilon}\mathbf{g}$
  $\theta = \theta - m$
until Converge

Algorithm 4 Adam

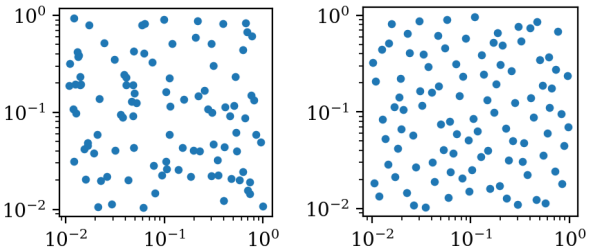
Require: learning rate $\eta > 0$, smoothing factor $0 \leq \beta_1, \beta_2 < 1$, numerical constant ϵ

$v = 0$, $m = 0$, $t = 0$
repeat
  $\mathbf{g} = \frac{1}{n}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$
  $m = \beta_1 m + (1-\beta_1)\mathbf{g}$
  $v = \beta_2 v + (1-\beta_2)\mathbf{g} \bigotimes \mathbf{g}$
  $\hat{m} = \frac{m}{1-\beta_1^t}$
  $\hat{v} = \frac{v}{1-\beta_2^t}$
  $\Delta\theta = -\eta \frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon}$
  $\theta = \theta + \Delta\theta$
  $t = t + 1$
until Converge

initial search space of hyperparameters. Next, we refined the hyperparameter search space based on these results and trained the models 16 times again for the refined search space in the second stage. Finally, we evaluated the noise-robustness and the impacts of optimizers and batch size on the model performance.

The initial hyperparameter search space was set based on a study on large-scale optimizer hyperparameter tuning conducted in [32]. Table 4 summarizes the initial search space. Similar to most previous studies on hyperparameter tuning, we sampled the hyperparameters γ of Momentum, (α, γ) of RMSProp, and (β_1, β_2) of Adam from $(1-\gamma)$, $(1-\alpha, 1-\gamma)$, and $(1-\beta_1, 1-\beta_2)$, respectively, following a quasi-random uniform distribution. Because hyperparameters have multiplicative effects on updating the parameters [39], [40], we sampled the hyperparameter set from a log scale. For example, when the search space of the learning rate was $[10^{-4}, 10^{0}]$, the hyperparameters were sampled from $10^{\mathcal{U}[-4,0]}$, where $\mathcal{U}$ denotes a uniform distribution.

Pseudo-random sampling can generate a biased distribution of hyperparameters. Therefore, we used a quasi-random distribution to sample the hyperparameters [52]. Fig. 9 shows the difference between the pseudo-random and quasi-random sampling for two variables in the range $[10^{-2}, 10^{0}]$. The sampling results show that the samples from the quasi-random distribution had a lower discrepancy than those from the pseudo-random distribution.



(a) Pseudo-random sampling (b) Quasi-random sampling

FIGURE 9. Comparison between pseudo-random and quasi-random sampling

We trained each model for 100 epochs for each sampled hyperparameter set and selected the result with the lowest validation loss. Cross-entropy was used as the cost function L given by (1). We let C, $y_i^{(j)}$, and $f(x_i;\theta)^{(j)}$ be the number of class, one-hot encoded label for j^{th} class of i^{th} data, and softmax output for j^{th} class of i^{th} data, respectively.

$$L(f(x_i;\theta), y_i) = -\sum_{j}^{C} y_i^{(j)} \log f(x_i;\theta)^{(j)} \tag{1}$$

After model training, we evaluated the models using both the test accuracy of the noise-free and noisy data. The test accuracy of the noise-free data represents the general performance of the model. By contrast, we used data with additive white Gaussian noise to investigate the model performance in a harsh industrial environment often exposed to noise interference. Additive white Gaussian noise was generated as follows. First, we calculated the signal power using (2), where N and x_i denote the length of the signal and i^{th} element of the signal, respectively. The power of the noise was then derived from (3) with a specific signal-to-noise ratio (SNR), denoted as SNR_{db}. Finally, Gaussian noise from the normal distribution $N(0, \sqrt{P_{\text{noise}}})$ was added to the original signal. To evaluate the robustness of the models against noisy data, we used the average accuracy using the noisy data with SNR $[-4, -2, 0, 2, 4]_{\text{db}}$.

$$P_{\text{signal}} = \frac{1}{N}\sum_{i}^{N} x_i^2 \tag{2}$$

$$\text{SNR}_{\text{db}} = 10\log_{10}\frac{P_{\text{signal}}}{P_{\text{noise}}} \tag{3}$$

IV. EXPERIMENTAL RESULTS AND DISCUSSION

This section presents the benchmarking results of a two-stage hyperparameter tuning for bearing fault diagnosis

TABLE 4. Initial hyperparameter search space.

SGD		Momentum		RMSProp		Adam	
Hyperparameter	Search Space	Hyperparameter	Search Space	Hyperparameter	Search Space	Hyperparameter	Search Space
η	$[10^{-4}, 10^{0}]$	η	$[10^{-4}, 10^{0}]$	η	$[10^{-4}, 10^{-1}]$	η	$[10^{-4}, 10^{-1}]$
		$1-\gamma$	$[10^{-3}, 10^{0}]$	$1-\alpha$	$[10^{-3}, 10^{0}]$	$1-\beta_1$	$[10^{-3}, 10^{0}]$
				$1-\gamma$	$[10^{-3}, 10^{0}]$	$1-\beta_2$	$[10^{-4}, 10^{-1}]$
				ϵ	$[10^{-10}, 10^{0}]$	ϵ	$[10^{-10}, 10^{0}]$

using various deep learning models and discusses the impact of the tuning results in both noise-free and noisy environments on model performance. To this end, we first tuned the hyperparameter in the initial search space, as presented in Table 4. Subsequently, we refined the search space and trained the models again based on the refined search space.

Table 5 summarizes the experimental setup. We used PyTorch [53] and PyTorch Lightning [54] to implement and train the models on two NVIDIA GeForce RTX2080 GPUs in parallel. For reproducibility, we also fixed a random seed in all the experiments.

TABLE 5. Hardware and software specifications for the experiment.

Type	Specification
OS	Ubuntu 18.04
CPU	Intel® Core™ i9-10900K @ 3.70GHz
RAM	128GB
GPU	NVIDIA GeForce RTX 2080 SUPER x2
CUDA version	11.2
CUDNN version	7.6.5
pyTorch version	1.12.1
pyTorch Lightning version	1.7.7

A. CASE 1: CWRU DATASET

In the training results of the first stage in the CWRU dataset, we observed that the smoothing factors and the numerical constant ϵ of the adaptive learning rate methods, that is, RMSProp and Adam, had only minor effects on the model performance. Therefore, we focused on the impacts of the learning rate and momentum factor.

Figs. 10–15 show the relationship between hyperparameter and model performance on the CWRU dataset with the optimizers and varying batch sizes through scatter plots and trend lines. The results showed that the model performances in both noise-free and noisy environments were significantly affected by the learning rate η for all optimizers and the momentum factor γ for Momentum and RMSProp, which were the dominant hyperparameters for determining the training speed of the models. For example, when η was high, the step size of the parameter update increased. High momentum factor γ also accelerated the model training.

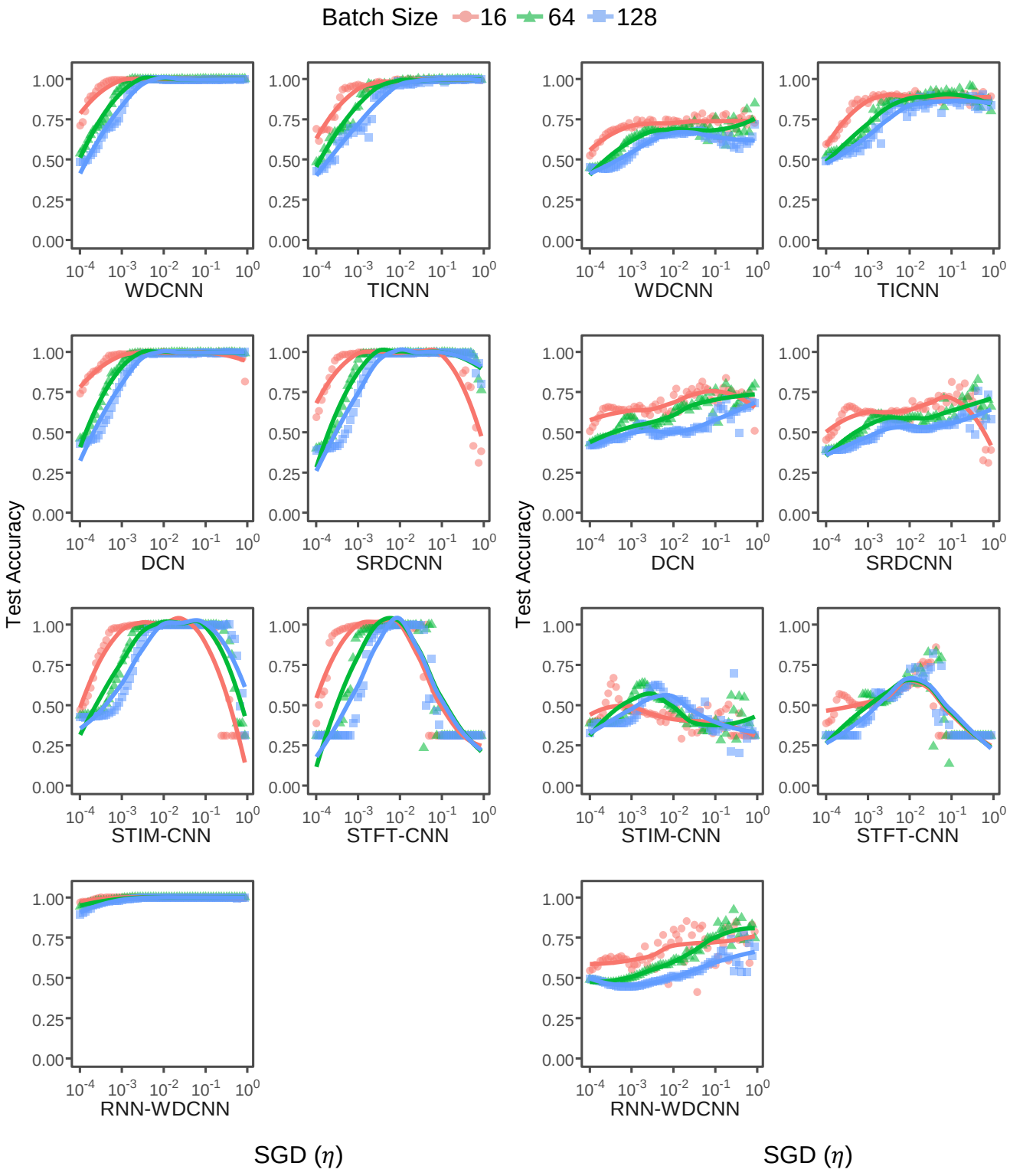
As shown in Fig. 10, in SGD, the learning rate significantly affected the performance of all models, except for RNN-WDCNN. For instance, in WDCNN and TICNN, when the learning rate was lower than 10^{-3}, the performance decreased in noise-free and noisy environments. DCN and SRDCNN showed patterns similar to those of WDCNN and TICNN in a noise-free environment. However, when the learning rate was higher than 10^{-1}, the performance of SRDCNN rapidly decreased in both environments, and the performance of DCN also decreased slightly in the noisy environment.

The results for 2D CNN models, STIM-CNN and STFT-CNN, showed slightly different patterns compared with 1D CNN models. The models had clear ranges of learning rates, producing the best model performance. In a noise-free environment, STIM-CNN and STFT-CNN achieved the maximum performance at a learning rate of around 10^{-2}. As for the noisy environment, STFT-CNN's performance was the best at a learning rate of around 10^{-1}; however, the best learning rate for STIM-CNN varied according to the batch size. Among all candidate models, RNN-WDCNN was the least affected by the learning rate. In a noise-free environment, the accuracy of RNN-WDCNN was approximately 100 %, and the higher the learning rate, the higher the accuracy in a noisy environment.

Figs. 11 and 12 show the effects of the learning rate and momentum factor, respectively, on the model performance using the Momentum optimizer. We observed that as the learning rate increased, the model performance declined more rapidly than SGD in noise-free and noisy environments when using the Momentum optimizer. For example, the performances of all candidate models in both environments decreased at learning rates higher than 10^{-2}. Moreover, with a higher momentum factor, that is, approximately 1, models trained in a small batch size generally exhibited poor test accuracy. This was because the Momentum optimizer trained models faster than SGD by employing previous gradients.

In most cases, when the learning rate was high, a small batch size led to poor model performances. Conversely, at lower learning rates, the model performance increased when the batch size was small. However, in 2D CNN models, the correlations of the learning rate and batch size on the model performance were unclear because their overall performances were low in a noisy environment.

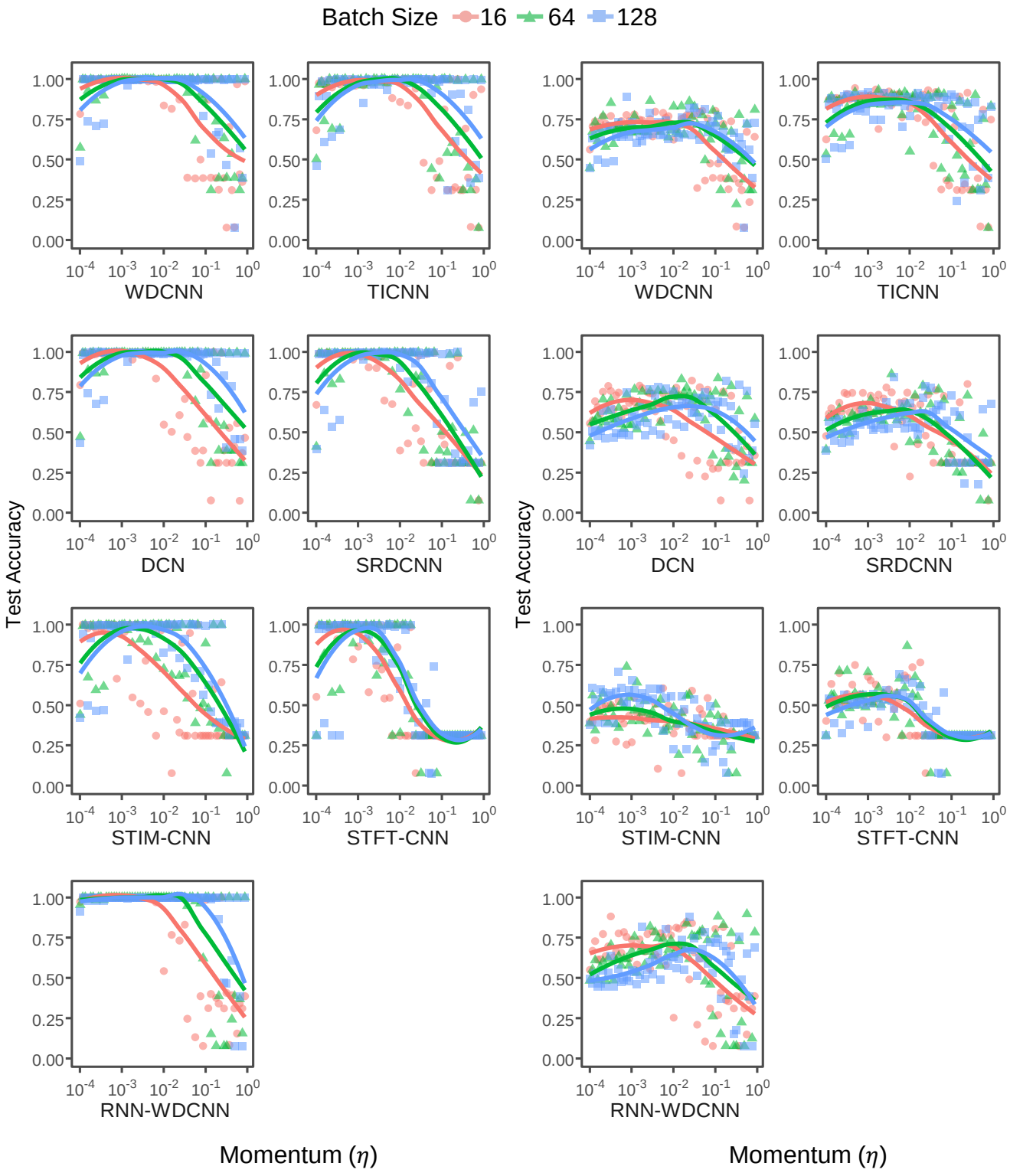
Based on the results illustrated in Fig. 13, we can observe that the model performance degradation followed by increasing the learning rate was the most outstanding



(a) Dataset with noise

(b) Dataset without noise

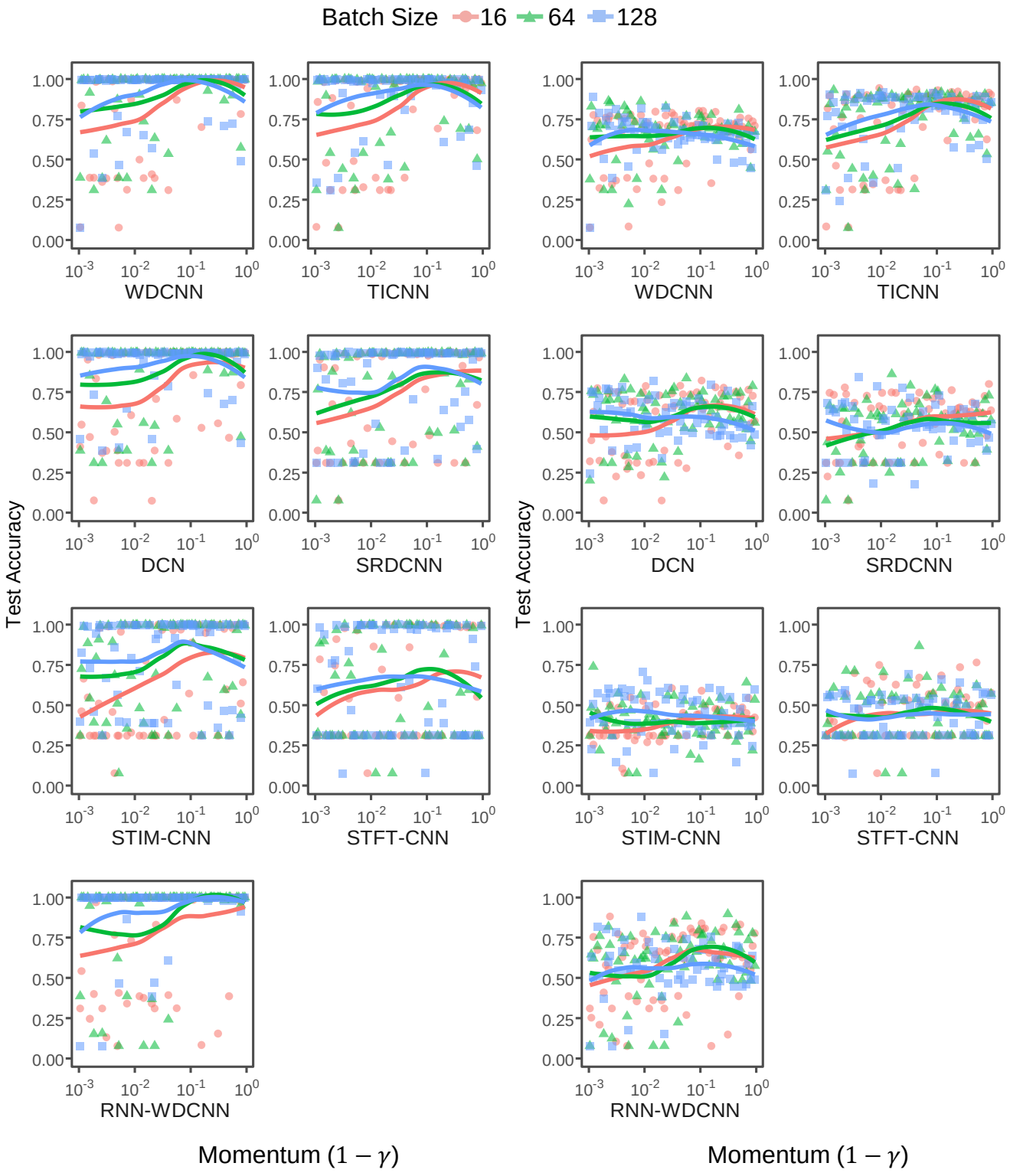
FIGURE 10. Hyperparameter tuning using the CWRU dataset and SGD optimizer (learning rate).



(a) Dataset with noise

(b) Dataset without noise

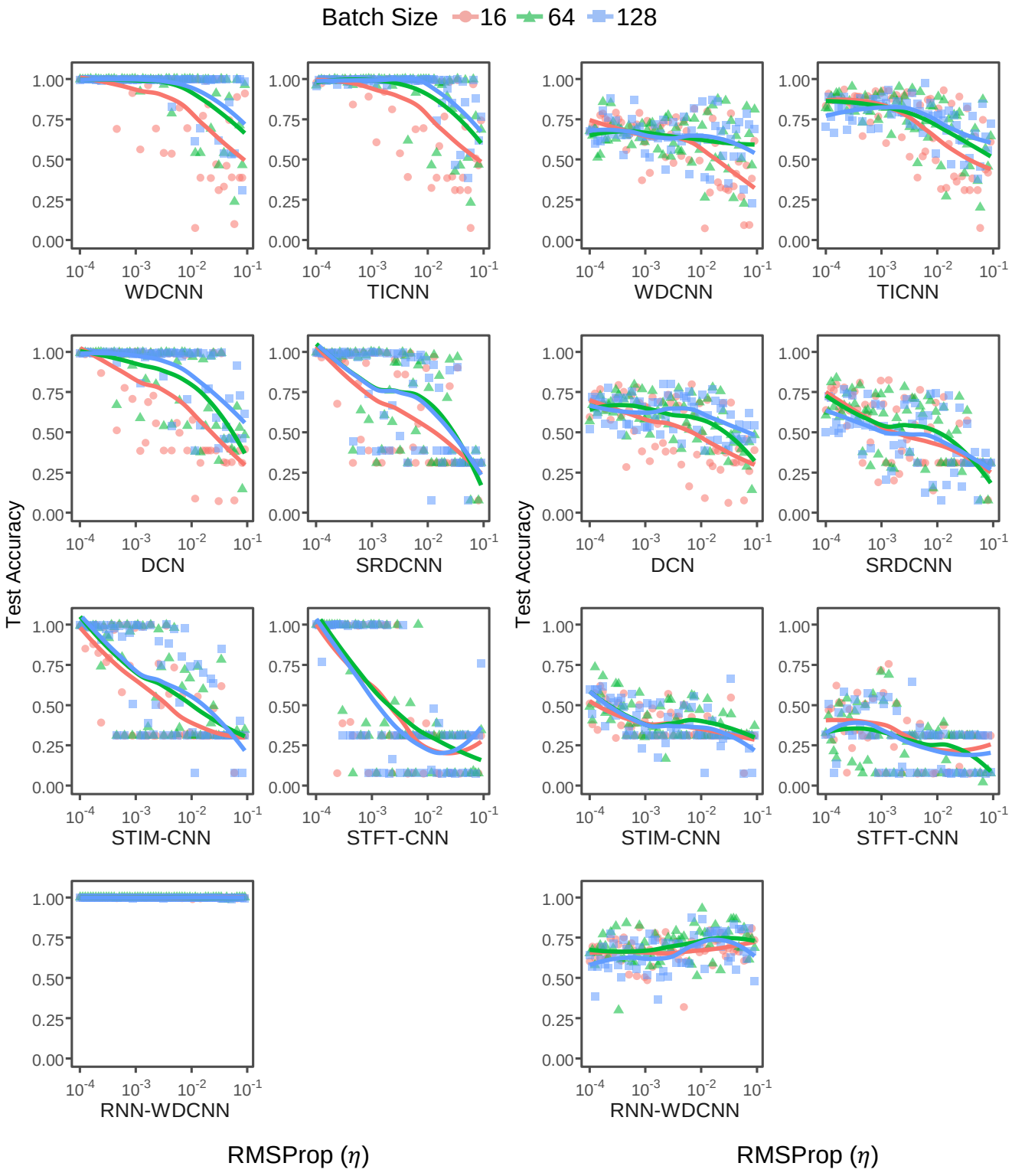
FIGURE 11. Hyperparameter tuning using the CWRU dataset and Momentum optimizer (learning rate).



(a) Dataset with noise

(b) Dataset without noise

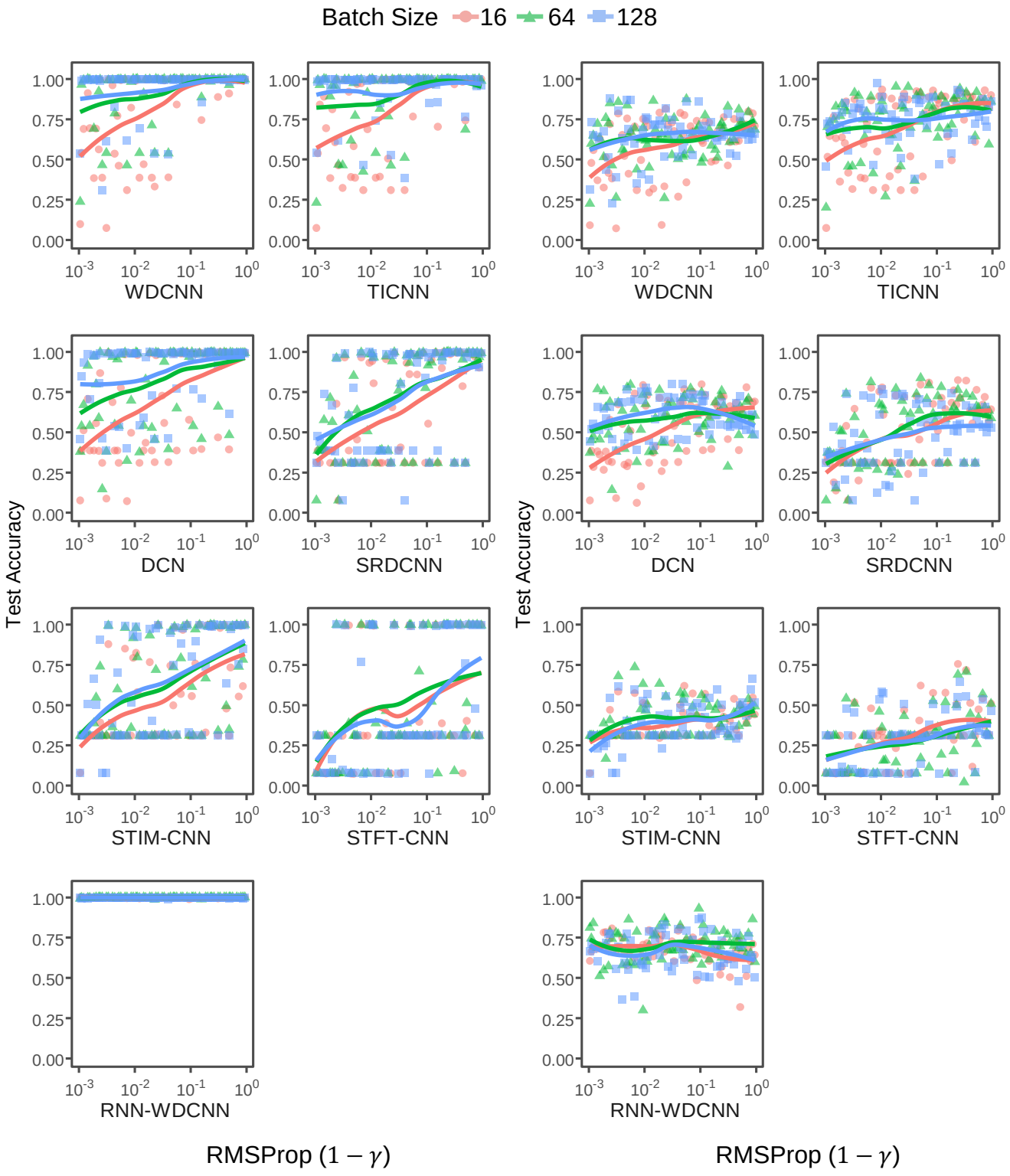
FIGURE 12. Hyperparameter tuning using the CWRU dataset and Momentum optimizer (momentum factor).



(a) Dataset with noise

(b) Dataset without noise

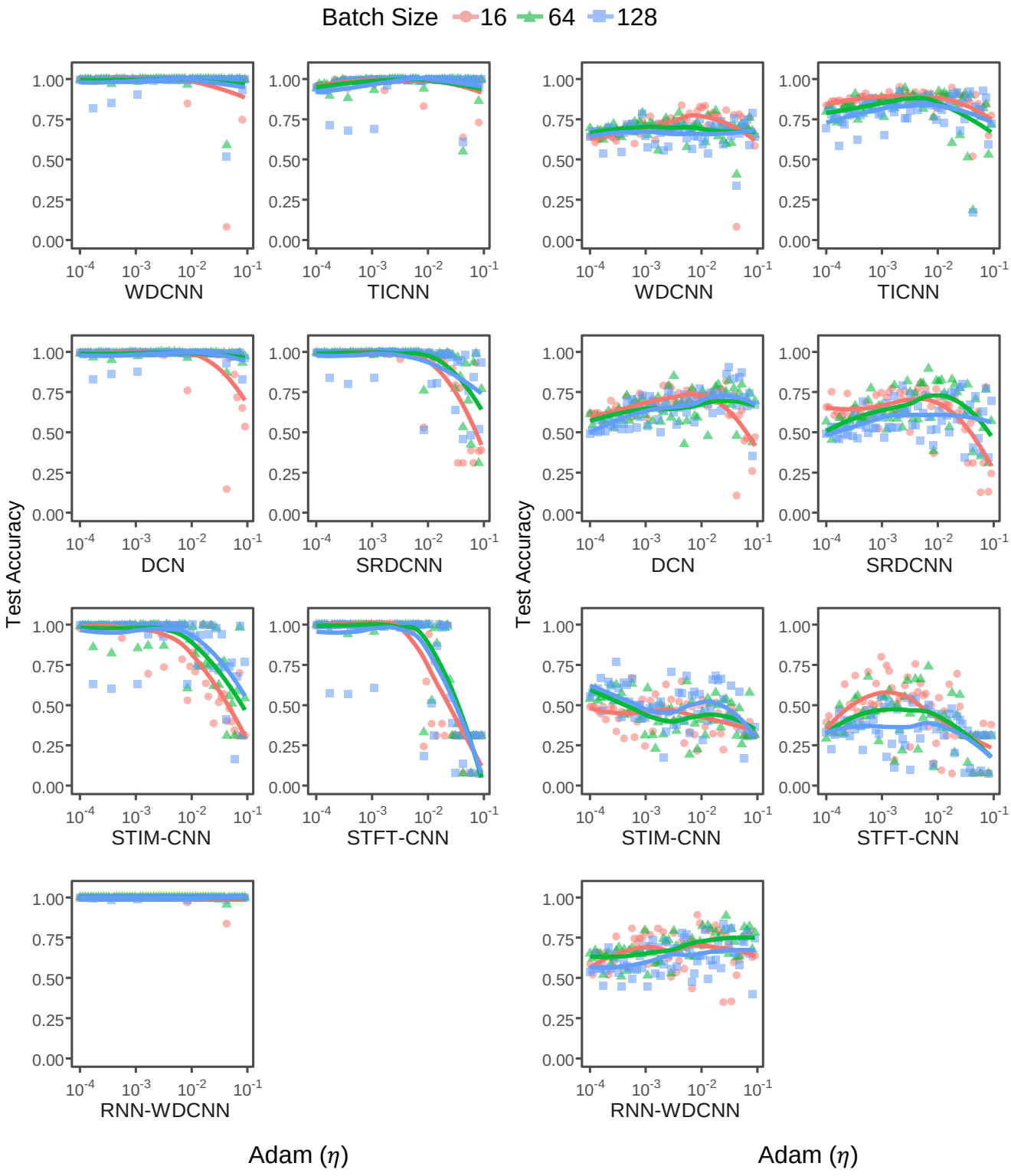
FIGURE 13. Hyperparameter tuning using the CWRU dataset and RMSProp optimizer (learning rate).



(a) Dataset with noise

(b) Dataset without noise

FIGURE 14. Hyperparameter tuning using the CWRU dataset and RMSProp optimizer (momentum factor).



(a) Dataset with noise

(b) Dataset without noise

FIGURE 15. Hyperparameter tuning using the CWRU dataset and Adam optimizer (learning rate).

TABLE 6. Refined hyperparameter search space for the CWRU dataset.

Model	SGD	Momentum		RMSProp ($\alpha = 0.99$, $\epsilon = 10^{-8}$)		Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
	η	η	$1-\gamma$	η	$1-\gamma$	η
WDCNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
TICNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
DCN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
SRDCNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-3}, 10^{-2}]$
STIM-CNN	$[10^{-3}, 10^{-2}]$	$[10^{-3}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$
STFT-CNN	$[10^{-3}, 10^{-1}]$	$[10^{-3}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
RNN-WDCNN	$[10^{-2}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-1}]$

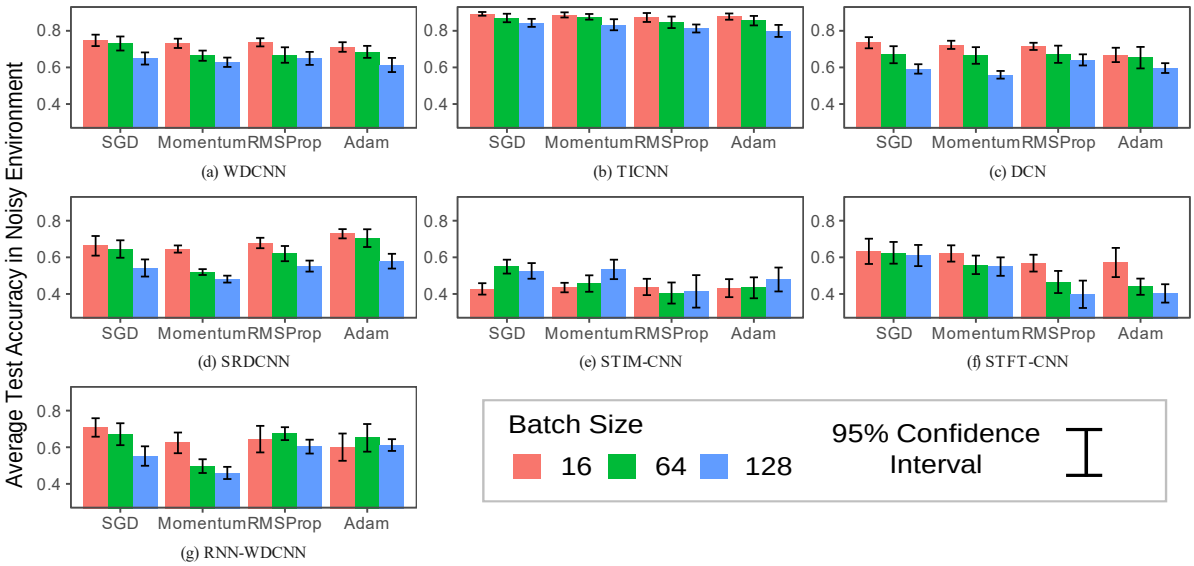


FIGURE 16. Model performance in a noisy environment with reduced search space using the CWRU dataset.

in the RMSProp optimizer. Similar to the Momentum optimizer, a high learning rate or momentum factor reduced the model performance in both environments, as shown in Figs. 13 and 14. Furthermore, performance reduction followed by the learning rate increment when using the RMSProp optimizer was higher than that of using the Momentum optimizer.

However, unlike the Momentum case, RNN-WDCNN exhibited consistent performance in all hyperparameter ranges. RNN-WDCNN maintained approximately 100 % accuracy when using the noise-free data and exhibited approximately 70–75 % average accuracy when using the noisy data in all hyperparameter ranges. Although some studies have shown the effectiveness of the RMSProp optimizer for training small RNN-based models [55]–[57], research on the effect of RMSProp learning rate is still limited. In our experiments, we found that the performance of RNN-WDCNN was less sensitive to learning rate variation.

Fig. 15 illustrates the performances of the models under various learning rates using the Adam optimizer. In the Adam case, the performances of DCN, SRDCNN, STIM-CNN, and STFT-CNN declined when the learning rate approached 10^{-1}. In addition, the performance was worse for a small batch size when the learning rate was approximately 10^{-1}. However, among the optimizers, the Adam optimizer was the least sensitive to variations in hyperparameter because WDCNN, TICNN, and RNN-WDCNN maintained their performances regardless of the changes in the learning rate, in most cases.

The hyperparameter tuning results in the first stage show that the hyperparameters had significant impacts on the model performance. In particular, the learning rate and momentum factor, which determine the model training speed, significantly affected the model accuracy in noise-free and noisy environments. In general, the model performance decreased when the model training speed was high with small batch sizes or when the model training speed was low with large batch sizes. Based on this result, we refined the hyperparameter search spaces, including those which did not result in model performance reduction. Because the hyperparameters α and ϵ of RMSProp and β_1, β_2, and ϵ of Adam had trivial effects on model performance, we set them to the default values. In most cases, the model performance rapidly decreased when the momentum factor exceeded

0.9. Therefore, we used $10^{-1} \leq 1-\gamma \leq 10^{0}$ for the new search space. Table 6 summarizes the refined search space for the hyperparameters.

We trained each model 16 times using the refined search space in the second stage and compared the model performance for different optimizers and batch sizes. Because the average test accuracies of all the models in a noise-free environment were higher than 95 %, only the results for the noisy environment were further analyzed.

Fig. 16 shows the model training results in the refined search space for the noisy data. The value in the bar chart is the average test accuracy of the 16 results in the noisy data, and the error bar shows the 95 % confidence interval. The results indicate that no optimizer achieved the best performance across all the models. However, it can be seen that the batch size and noise-robustness correlate, except for STIM-CNN and RNN-WDCNN. A small batch size led to high model performance; however, RNN-WDCNN showed such a tendency only when using SGD and Momentum optimizers. The relationship between batch size and model performance was not clear in STIM-CNN. 2D CNNs were less stable and had lower accuracies than 1D CNNs in noisy environments. In particular, among all candidate models, TICNN achieved the highest and most stable performance.

B. CASE 2: MFPT DATASET

We conducted the same experiment described in Section IV-A using the MFPT dataset. The results also show that smoothing factors, namely, α, β_1, β_2, and the numerical constant ϵ had trivial impacts on model performance. Thus, similar to the first stage experiment using the CWRU dataset, we investigated the effects of the learning rate and momentum factor on the model performance in detail.

Figs. 17–22 show the hyperparameter tuning results for the first stage. In the initial search space, the parameters η for all optimizers and γ for Momentum and RMSProp optimizers significantly affected the model performance. This result shows that the learning rate and momentum factor were critical hyperparameters in both datasets.

Fig. 17 shows the relationship between the learning rate and model performance using SGD. Similar to the CWRU case in Fig. 10, each model had a specific learning rate range to produce high model performance. The performances of WDCNN and DCN rapidly decreased at learning rates lower than 10^{-3}. The performance of TICNN decreased when the learning rates were lower than 10^{-2}. By contrast, 2D CNNs maintained their performance at learning rates between 10^{-3} and 10^{-1} when using noise-free data. For all models, at high learning rates, training with large batch sizes resulted in a higher accuracy than training with small batch sizes. By contrast, a small batch size led to a higher model performance at a low learning rate.

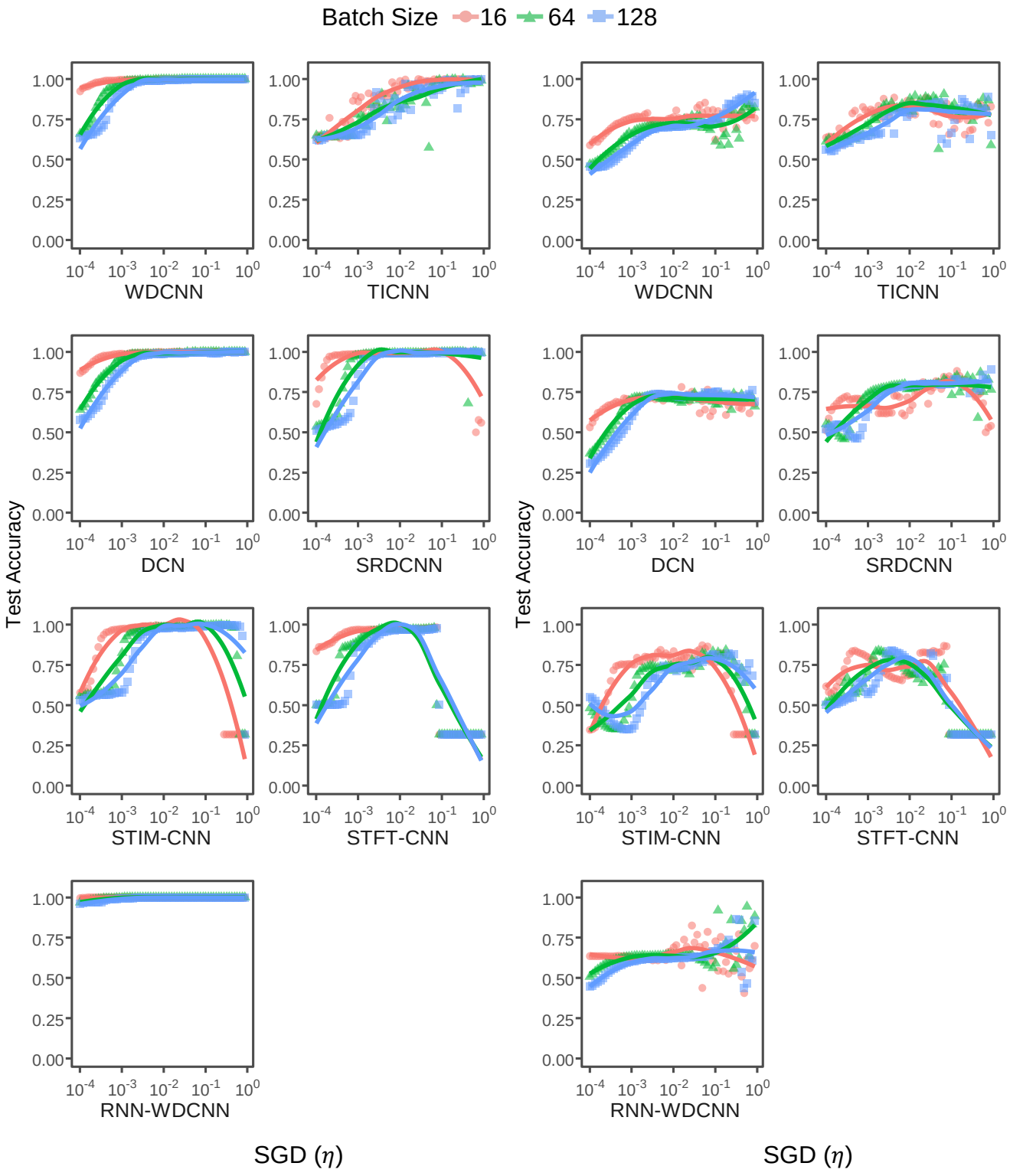
In both datasets, RNN-WDCNN exhibited the highest performance in the entire learning rate range in a noise-free environment and a learning rate of approximately 1 in a noisy environment. However, the performance of TICNN was more sensitive to the learning rate when using the MFPT dataset than the CWRU dataset. TICNN achieved the highest test accuracy when the learning rate was higher than 10^{-2} in noise-free and noisy environments using the CWRU dataset. However, for the MFPT dataset, the highest test accuracy was achieved when the learning rate was approximately 1 in a noise-free environment. The other models had similar tendencies for both datasets.

The hyperparameter tuning results showed similar patterns in the two datasets using the Momentum optimizer. As shown in Fig. 18, the test accuracies of all the models decreased rapidly when the learning rates were higher than 10^{-2}. At high learning rates, training results with small batch sizes generally had lower accuracies than those with large batch sizes. In particular, the 2D CNN models, STIM-CNN and STFT-CNN, exhibited noticeable performance reduction at high learning rates compared to the other models. In addition, when the momentum factor was approximately 1, the performances of the models decreased in most cases, as shown in Fig. 19. However, the performance reduction owing to the increasing momentum factor was smaller than that in the learning rate case.

Similar to Momentum, the performances of the models decreased when the learning rate increased in RMSProp, as shown in Fig. 20. However, the degree of performance reduction caused by the increase in learning rate was smaller using the MFPT dataset than the CWRU dataset. In addition, whereas the performances of WDCNN, TICNN, and DCN exhibited little variation based on the learning rate in a noisy environment, the performances of SRDCNN, STIM-CNN, and STFT-CNN were more sensitive to the learning rate. Similar to the experiment using the CWRU dataset, the performance of RNN-WDCNN was stable at all learning rates in both noise-free and noisy environments.

Fig. 21 shows the impact of the momentum factor on the model performance. When the momentum factor was approximately 1, the model performance decreased; however, the effect of the momentum factor was smaller than that of the learning rate.

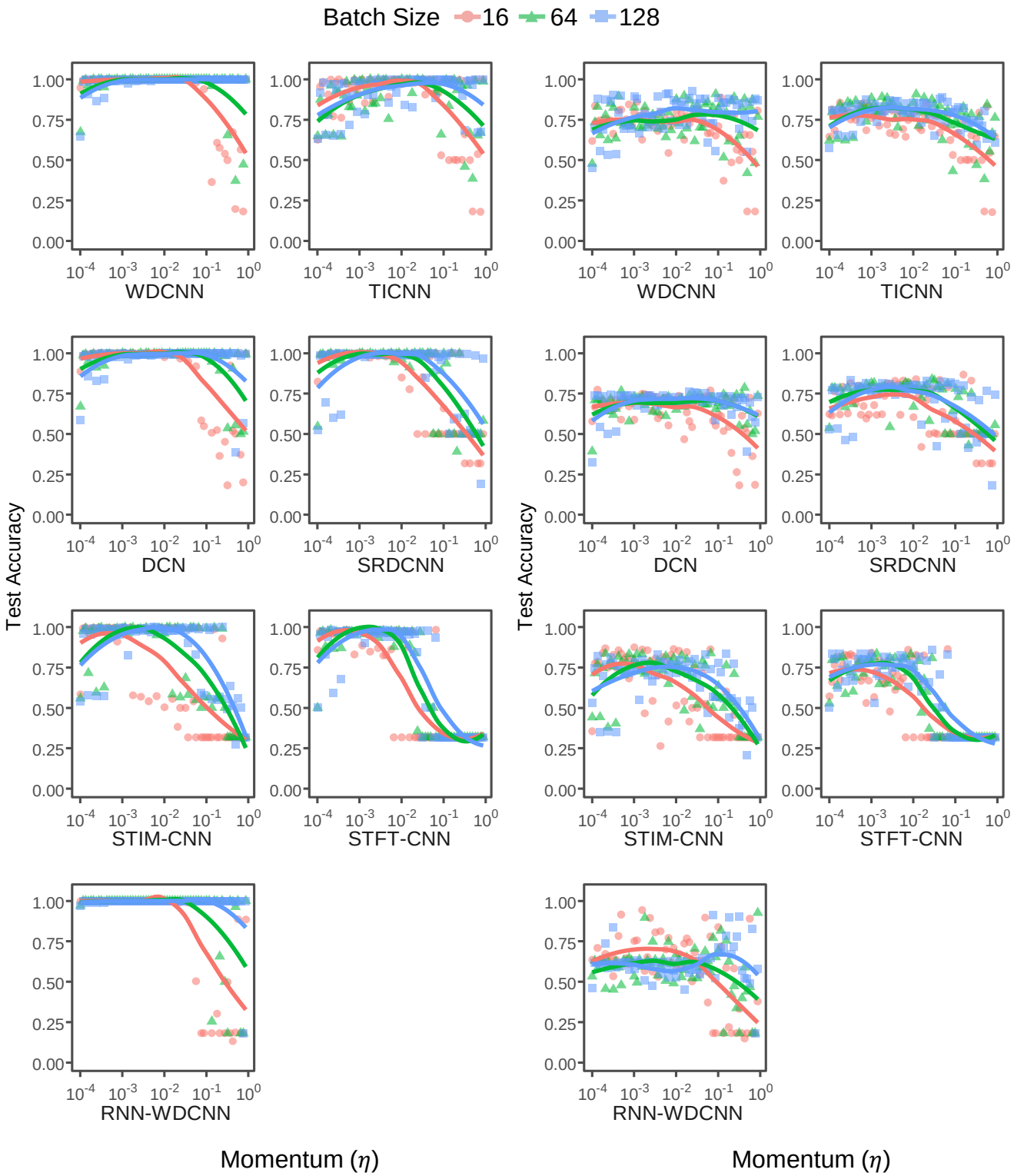
Fig. 22 shows the relationship between the learning rate of the Adam optimizer and model performance on the MFPT dataset. The results showed that, among the four optimizers, the learning rate of Adam had the least impact on the model performance. However, for SRDCNN, STIM-CNN, and STFT-CNN, when the learning rates were higher than 10^{-2}, the performance decreased regardless of the existence of noise. In addition, the model performance of TICNN decreased when the learning rate was less than 10^{-3}. Similar to the other



(a) Dataset with noise

(b) Dataset without noise

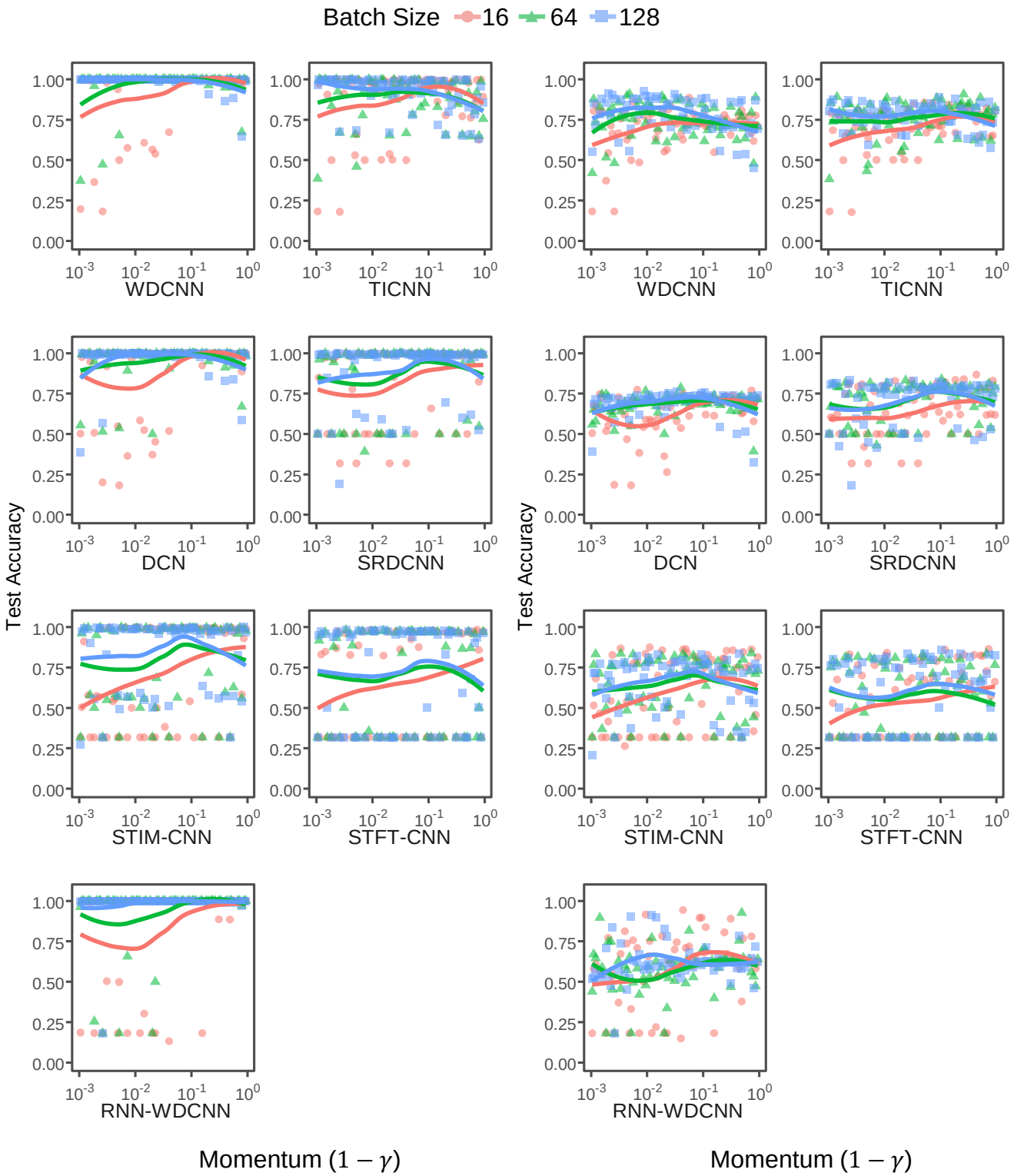
FIGURE 17. Hyperparameter tuning using the MFPT dataset and SGD optimizer (learning rate).



(a) Dataset without noise

(b) Dataset with noise

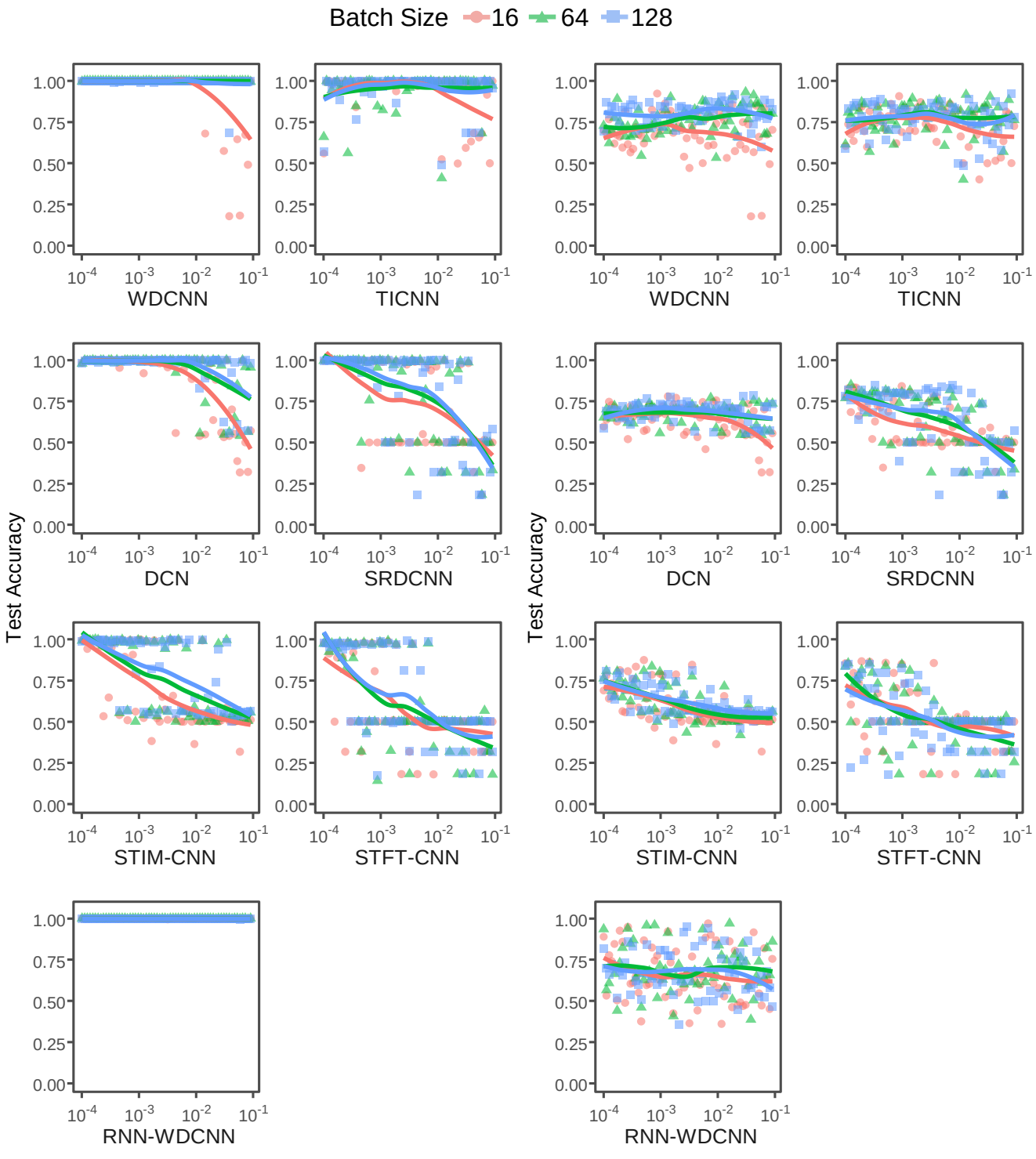
FIGURE 18. Hyperparameter tuning using the MFPT dataset and Momentum optimizer (learning rate).



(a) Dataset without noise

(b) Dataset with noise

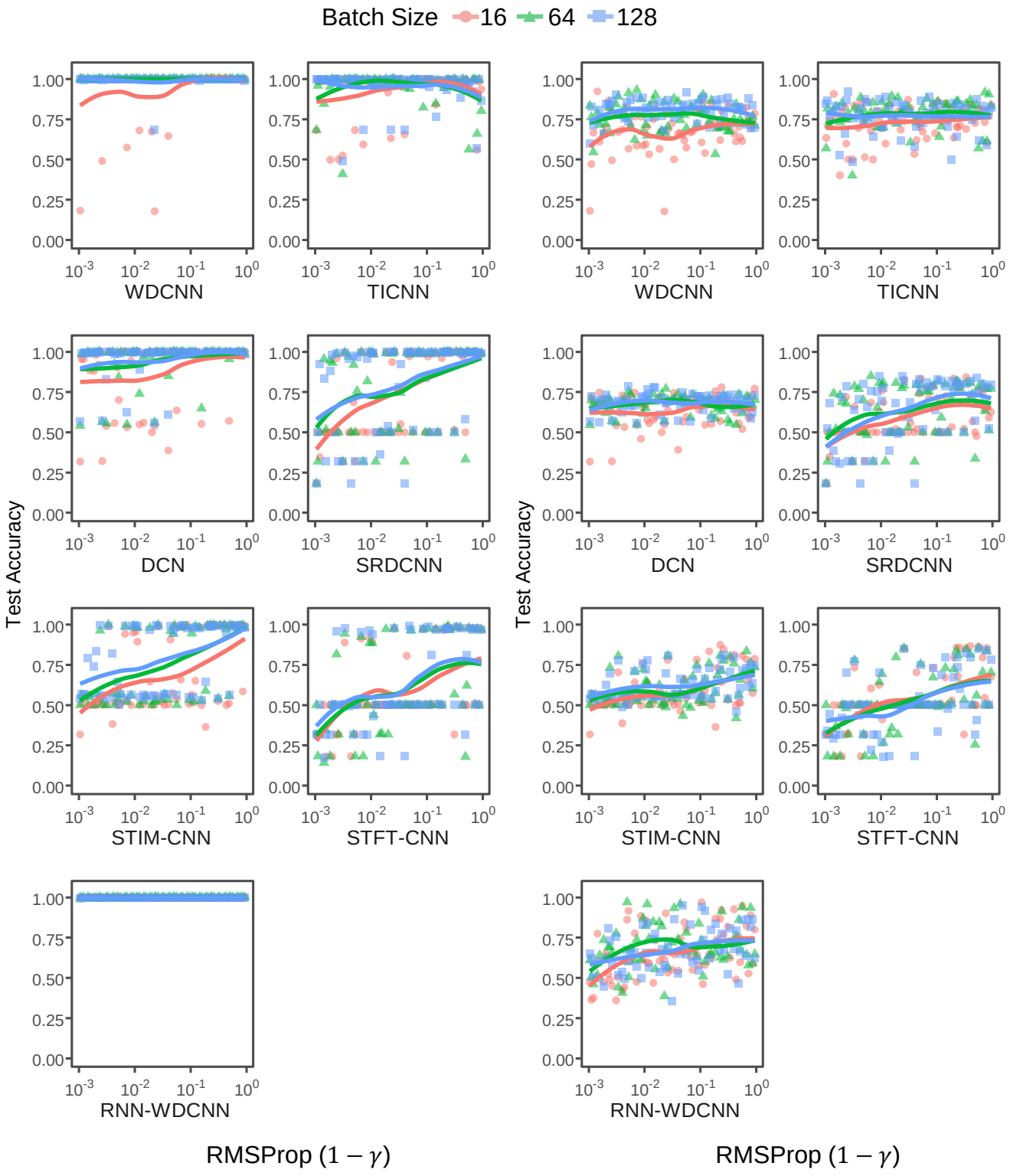
FIGURE 19. Hyperparameter tuning using the MFPT dataset and Momentum optimizer (momentum factor).



(a) Dataset without noise

(b) Dataset with noise

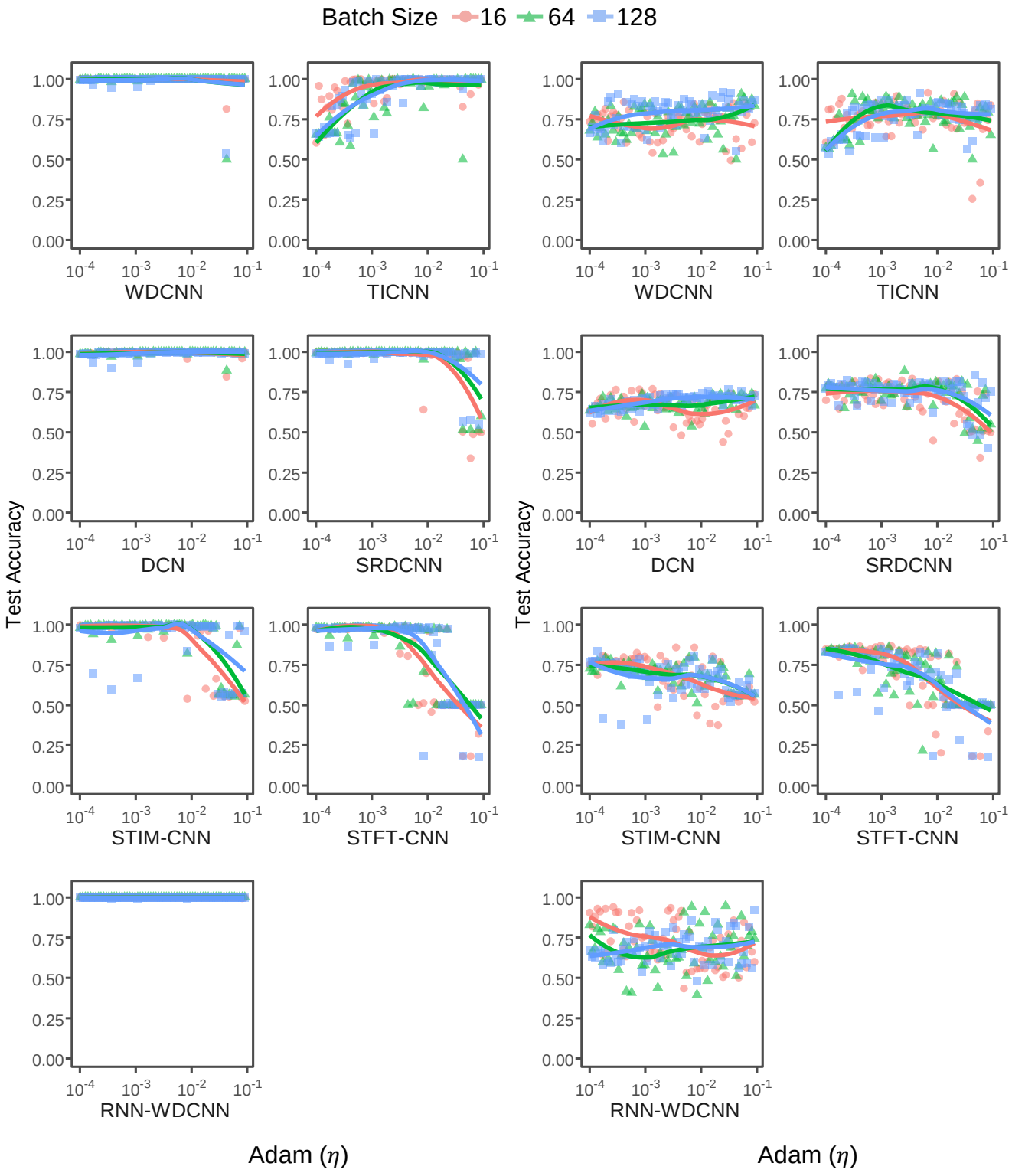
FIGURE 20. Hyperparameter tuning using the MFPT dataset and RMSProp optimizer (learning rate).



(a) Dataset with noise

(b) Dataset without noise

FIGURE 21. Hyperparameter tuning using the MFPT dataset and RMSProp optimizer (momentum factor).



(a) Dataset without noise

(b) Dataset with noise

FIGURE 22. **Hyperparameter tuning using the MFPT dataset and Adam optimizer (learning rate).**

TABLE 7. Refined hyperparameter search space for the MFPT dataset.

Model	SGD	Momentum		RMSProp ($\alpha = 0.99$, $\epsilon = 10^{-8}$)		Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)
	η	η	$1-\gamma$	η	$1-\gamma$	η
WDCNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
TICNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-3}, 10^{-1}]$
DCN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-1}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$
SRDCNN	$[10^{-2}, 10^{0}]$	$[10^{-3}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-3}, 10^{-2}]$
STIM-CNN	$[10^{-3}, 10^{-1}]$	$[10^{-4}, 10^{-2}]$	$[10^{-2}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$
STFT-CNN	$[10^{-3}, 10^{-1}]$	$[10^{-4}, 10^{-2}]$	$[10^{-2}, 10^{0}]$	$[10^{-4}, 10^{-3}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-3}]$
RNN-WDCNN	$[10^{-2}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-2}]$	$[10^{-1}, 10^{0}]$	$[10^{-4}, 10^{-1}]$

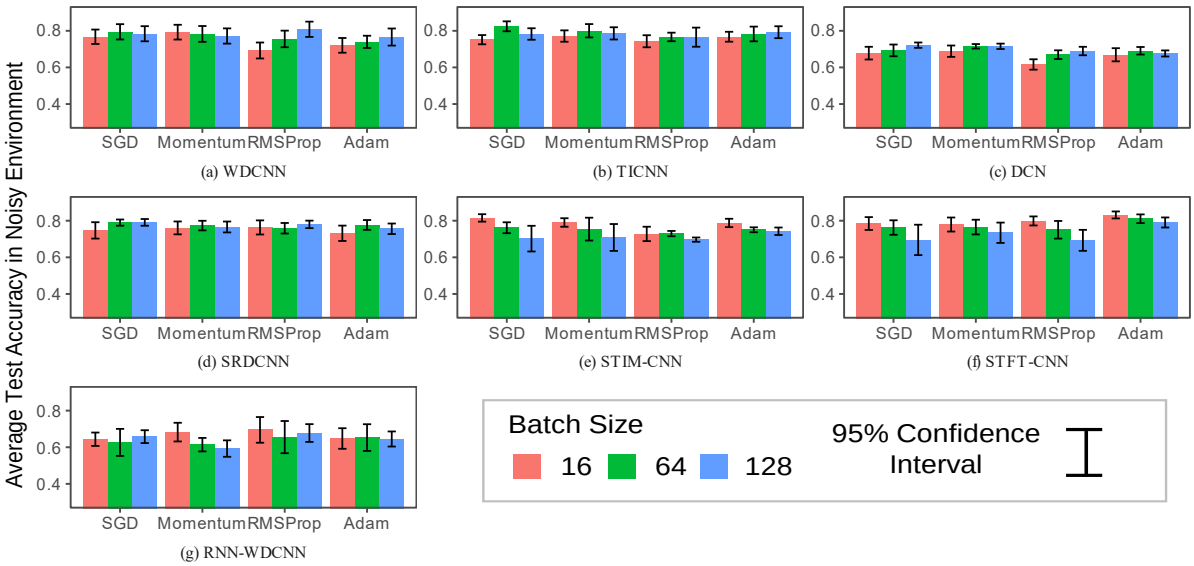


FIGURE 23. Model performance in a noisy environment with reduced search space using the MFPT dataset.

optimizers, at higher learning rates, the models achieved higher performances with larger batch sizes. By contrast, with smaller batch sizes, lower learning rates resulted in higher performances.

In the experimental results performed on the initial hyperparameter search space using the MFPT dataset, the overall relationship between hyperparameters and the model performance was similar in the two datasets. Furthermore, the model performance varied significantly depending on the hyperparameters. Therefore, based on the experimental results of the initial search space, we refined the hyperparameter search space by the same method used for the CWRU dataset. Table 7 summarizes the refined search space for the MFPT dataset.

For the refined search space, all models achieved test accuracy higher than 95 % in a noise-free environment; therefore, we analyzed in detail the performances of the models only for a noisy environment. Fig. 23 shows the results of the training repeated 16 times for each model using the MFPT dataset in the refined search space. Although decreasing the batch size was effective for noise-robustness using the CWRU dataset, only the 2D CNN models were more robust against noise when the batch size was small in the MFPT dataset. Except for TICNN, all models achieved higher test accuracies using the noisy data of the MFPT dataset than the CWRU dataset. TICNN was the most robust model against noise using the CWRU dataset but exhibited similar performance to WDCNN using the MFPT dataset. The performances of 2D CNNs were also improved, which was similar to 1D CNNs. However, unlike the pure CNN models, the performance of RNN-WDCNN using the MFPT dataset was not improved compared with using the CWRU dataset.

C. MODEL PERFORMANCE COMPARISON

After two-stage training using the two datasets, we summarized the training results and evaluated the model performance, as shown in Table 8. The performance of the models was evaluated by the average test accuracies and their 95 % confidence interval in both noise-free and noisy environments. The number of training results for each model was $4 \times 3 \times 16 = 192$, representing all cases for the four optimizers and three batch sizes.

Moreover, on-device AI systems have gained attention in machine fault diagnosis for reducing the inference

TABLE 8. Comparison of candidate models in terms of accuracy, model size, and computation load.

Model	Accuracy(%) and 95% Confidence Interval				No. of Params (M)	MFLOPs
	CWRU (without noise)	CWRU (with noise)	MFPT (without noise)	MFPT (with noise)		
WDCNN	99.81±0.13	68.46±3.60	99.95±0.06	76.46±4.03	0.05	1.48
TICNN	98.85±0.78	85.56±2.50	95.59±3.84	77.79±3.17	0.07	2.63
DCN	99.39±0.34	65.75±4.12	99.46±0.34	68.53±2.61	0.16	9.17
SRDCNN	98.76±2.51	61.33±4.93	99.40±1.47	76.62±2.88	0.82	254.90
STIM-CNN	97.53±3.24	46.08±5.22	97.19±3.27	74.72±3.96	3.53	28.50
STFT-CNN	97.32±4.83	53.69±6.64	96.10±3.53	76.63±4.45	2.60	17.15
RNN-WDCNN	99.90±0.27	60.81±3.60	99.94±0.16	64.92±5.28	0.86	76.65

time and data traffic between edge devices on a factory floor and a server. Because on-device AI systems have limited hardware resources, we used the number of parameters and the million floating point operations per second (MFLOPs) as the evaluation metrics of the candidate models.

Overall, 1D CNN models outperformed 2D CNNs in noise-free and noisy environments, particularly for the CWRU dataset with noise. Furthermore, the number of parameters of the 1D CNN models was smaller than that of the 2D CNN models, indicating that 1D CNN models require smaller memory storage than 2D CNN models. In a noisy environment, the performances of the models, except for TICNN and RNN-WDCNN, were notably higher for the MFPT dataset than for the CWRU dataset. TICNN was the only model with worse performance for the MFPT dataset than for the CWRU dataset. On the contrary, RNN-WDCNN exhibited a similar performance for the two datasets. The performance of RNN-WDCNN was higher than the 2D CNN models for the CWRU dataset but lower than the 2D CNN models for the MFPT dataset.

In our experiment, TICNN was the most robust against noise among the 1D CNN models. However, the test accuracy of TICNN on the MFPT dataset and noise-free environment was 95.59 ± 3.84 %, which was slightly lower than the accuracies of the other models. We speculated that this was because the kernel dropout rate used by TICNN was optimized only for the CWRU dataset; therefore, the performance of TICNN for the MFPT dataset in a noise-free environment was relatively low. WDCNN had a similar performance to TICNN for the MFPT dataset, but its performance declined considerably for the CWRU dataset with noise. DCN and SRDCNN exhibited comparable performances for both datasets. The accuracy of DCN was slightly higher than that of SRDCNN using the CWRU dataset with noise; however, using the MFPT dataset, SRDCNN was more robust against noise than DCN. Using both datasets with noise, these two models had lower performances than WDCNN and TICNN, which had more computations and parameters. For example, SRDCNN required 254.90 MFLOPs; thus, it performed at least 28–172 times more operations than the other 1D CNN models.

V. CONCLUSIONS

This study aimed to evaluate the impact of hyperparameter tuning on deep learning optimizers in bearing fault diagnosis. To this end, we selected two open-access datasets, four optimizers, and seven candidate models and performed hyperparameter tuning in two stages with varying batch sizes. Our experimental results showed the impact of hyperparameters and batch sizes on model performance.

In our benchmarking study, the learning rate and momentum factor were the key hyperparameters significantly affecting the model performance. In addition, we observed that large batch sizes led to high model performances at high learning rates or momentum factors. By contrast, the model performances were high for small batch sizes at low learning rates or momentum factors.

We also evaluated the performance and computational efficiency of each model. The evaluation results showed that TICNN was the best on-device AI solution candidate in terms of computational efficiency and noise-robustness.

REFERENCES

[1] S. Nandi, H. A. Toliyat, and X. Li, "Condition Monitoring and Fault Diagnosis of Electrical Motors—A Review," IEEE Transactions on Energy Conversion, vol. 20, no. 4, pp. 719–729, 2005.

[2] D.-T. Hoang and H.-J. Kang, "A survey on Deep Learning based bearing fault diagnosis," Neurocomputing, vol. 335, pp. 327–335, 2019.

[3] F. Wang, J. Sun, D. Yan, S. Zhang, L. Cui, and Y. Xu, "A Feature Extraction Method for Fault Classification of Rolling Bearing based on PCA," Journal of Physics: Conference Series, vol. 628, no. 1, p. 012079, 2015.

[4] L. Shuang and L. Meng, "Bearing Fault Diagnosis Based on PCA and SVM," 2007 International Conference on Mechatronics and Automation, pp. 3503–3507, 2007.

[5] J. Zhu, T. Hu, B. Jiang, and X. Yang, "Intelligent bearing fault diagnosis using PCA–DBN framework," Neural Computing and Applications, vol. 32, no. 14, pp. 10 773–10 781, 2020.

[6] X. Zhang, Y. Liang, J. Zhou, and Y. Zang, "A novel bearing fault diagnosis model integrated permutation entropy, ensemble empirical mode decomposition and optimized SVM," Measurement, vol. 69, pp. 164–179, 2015.

[7] Z. Wang, Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He, "Fault Diagnosis of a Rolling Bearing Using Wavelet Packet Denoising and Random Forests," IEEE Sensors Journal, vol. 17, no. 17, pp. 5581–5588, 2017.

[8] X. Qin, Q. Li, X. Dong, and S. Lv, "The Fault Diagnosis of Rolling Bearing Based on Ensemble Empirical Mode Decomposition and Random Forest," Shock and Vibration, vol. 2017, pp. 1–9, 2017.

[9] A. Sharma, R. Jigyasu, L. Mathew, and S. Chatterji, "Bearing Fault Diagnosis Using Weighted K-Nearest Neighbor," 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), vol. 00, pp. 1132–1137, 2018.
[10] Q. Lu, X. Shen, X. Wang, M. Li, J. Li, and M. Zhang, "Fault Diagnosis of Rolling Bearing Based on Improved VMD and KNN," Mathematical Problems in Engineering, vol. 2021, pp. 1–11, 2021.
[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[12] F. Jia, Y. Lei, J. Lin, X. Zhou, and N. Lu, "Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data," Mechanical Systems and Signal Processing, vol. 72, pp. 303–315, 2016.
[13] W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, "A New Deep Learning Model for Fault Diagnosis with Good Anti-Noise and Domain Adaptation Ability on Raw Vibration Signals," Sensors, vol. 17, no. 2, p. 425, 2017.
[14] W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, "A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load," Mechanical Systems and Signal Processing, vol. 100, pp. 439–453, 2018.
[15] H. Liang and X. Zhao, "Rolling Bearing Fault Diagnosis Based on One-Dimensional Dilated Convolution Network With Residual Connection," IEEE Access, vol. 9, pp. 31 078–31 091, 2021.
[16] Z. Zhuang, H. Lv, J. Xu, Z. Huang, and W. Qin, "A Deep Learning Method for Bearing Fault Diagnosis through Stacked Residual Dilated Convolutions," Applied Sciences, vol. 9, no. 9, p. 1823, 2019.
[17] M. Zhao, S. Zhong, X. Fu, B. Tang, and M. Pecht, "Deep Residual Shrinkage Networks for Fault Diagnosis," IEEE Transactions on Industrial Informatics, vol. 16, no. 7, pp. 4681–4690, 2019.
[18] T. Chen, Z. Wang, X. Yang, and K. Jiang, "A deep capsule neural network with stochastic delta rule for bearing fault diagnosis on raw vibration signals," Measurement, vol. 148, p. 106857, 2019.
[19] Z. Zhu, G. Peng, Y. Chen, and H. Gao, "A convolutional neural network based on a capsule network with strong generalization for bearing fault diagnosis," Neurocomputing, vol. 323, pp. 62–75, 2019.
[20] J. Zhao, S. Yang, Q. Li, Y. Liu, X. Gu, and W. Liu, "A new bearing fault diagnosis method based on signal-to-image mapping and convolutional neural network," Measurement, vol. 176, p. 109088, 2021.
[21] D. Yao, H. Liu, J. Yang, and X. Li, "A lightweight neural network with strong robustness for bearing fault diagnosis," Measurement, vol. 159, p. 107756, 2020.
[22] Y. Zhang, K. Xing, R. Bai, D. Sun, and Z. Meng, "An enhanced convolutional neural network for bearing fault diagnosis based on time–frequency image," Measurement, vol. 157, p. 107667, 2020.
[23] R. Magar, L. Ghule, J. Li, Y. Zhao, and A. B. Farimani, "FaultNet: A Deep Convolutional Neural Network for Bearing Fault Classification," IEEE Access, vol. 9, pp. 25 189–25 199, 2021.
[24] C.-Y. Lee, G.-L. Zhuo, and T.-A. Le, "A Robust Deep Neural Network for Rolling Element Fault Diagnosis under Various Operating and Noisy Conditions," Sensors (Basel, Switzerland), vol. 22, no. 13, p. 4705, 2022.
[25] Z. Chen, A. Mauricio, W. Li, and K. Gryllias, "A deep learning method for bearing fault diagnosis based on Cyclic Spectral Coherence and Convolutional Neural Networks," Mechanical Systems and Signal Processing, vol. 140, p. 106683, 2020.
[26] A. Shenfield and M. Howarth, "A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults," Sensors, vol. 20, no. 18, p. 5112, 2020.
[27] G. Jin, T. Zhu, M. W. Akram, Y. Jin, and C. Zhu, "An Adaptive Anti-Noise Neural Network for Bearing Fault Diagnosis Under Noise and Varying Load Conditions," IEEE Access, vol. 8, pp. 74 793–74 807, 2020.
[28] L. Y. Imamura, S. L. Avila, F. S. Pacheco, M. B. C. Salles, and L. S. Jablon, "Diagnosis of Unbalance in Lightweight Rotating Machines Using a Recurrent Neural Network Suitable for an Edge-Computing Framework," Journal of Control, Automation and Electrical Systems, pp. 1–14, 2022.
[29] Y. Ding, M. Jia, Q. Miao, and Y. Cao, "A novel time–frequency transformer based on self–attention mechanism and its application in fault diagnosis of rolling bearings," Mechanical Systems and Signal Processing, vol. 168, p. 108616, 2022.
[30] H. Fang, J. Deng, Y. Bai, B. Feng, S. Li, S. Shao, and D. Chen, "Clformer: A lightweight transformer based on convolutional embedding and linear self-attention with strong robustness for bearing fault diagnosis under limited sample conditions," IEEE Transactions on Instrumentation and Measurement, vol. 71, pp. 1–8, 2021.
[31] R.-Y. Sun, "Optimization for deep learning: An overview," Journal of the Operations Research Society of China, vol. 8, no. 2, pp. 249–294, 2020.
[32] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On empirical comparisons of optimizers for deep learning," arXiv preprint arXiv:1910.05446, 2019.
[33] R. B. Randall and J. Antoni, "Rolling element bearing diagnostics—A tutorial," Mechanical Systems and Signal Processing, vol. 25, no. 2, pp. 485–520, 2011.
[34] P. Kankar, S. C. Sharma, and S. Harsha, "Fault diagnosis of ball bearings using machine learning methods," Expert Systems with Applications, vol. 38, no. 3, pp. 1876–1886, 2011.
[35] ——, "Rolling element bearing fault diagnosis using wavelet transform," Neurocomputing, vol. 74, no. 10, pp. 1638–1645, 2011.
[36] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," Ussr computational mathematics and mathematical physics, vol. 4, no. 5, pp. 1–17, 1964.
[37] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," Cited on, vol. 14, no. 8, p. 2, 2012.
[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
[39] Y. Bengio, "Neural Networks: Tricks of the Trade, Second Edition," Lecture Notes in Computer Science, pp. 437–478, 2012.
[40] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," Journal of Machine Learning Research, 2012.
[41] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers," arXiv, 2020.
[42] P. Verma, V. Tripathi, and B. Pant, "Comparison of different optimizers implemented on the deep learning architectures for COVID-19 classification," Materials Today: Proceedings, vol. 46, no. Radiology 295 3 2020, pp. 11 098–11 102, 2021.
[43] M. H. Saleem, J. Potgieter, and K. M. Arif, "Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers," Plants, vol. 9, no. 10, p. 1319, 2020.
[44] S. Y. Şen and N. Özkurt, "Convolutional Neural Network Hyperparameter Tuning with Adam Optimizer for ECG Classification," 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), vol. 00, pp. 1–6, 2020.
[45] B. Rezaeianjouybari and Y. Shang, "An empirical study of machine learning and deep learning algorithms on bearing fault diagnosis benchmarks," in ASME International Mechanical Engineering Congress and Exposition, vol. 85611. American Society of Mechanical Engineers, 2021, p. V07AT07A050.
[46] "Bearing data center," 2021. [Online]. Available: https://engineering.case.edu/bearingdatacenter/
[47] E. Bechhoefer, "Condition based maintenance fault database for testing of diagnostic and prognostics algorithms," 2022. [Online]. Available: https://www.mfpt.org/fault-data-sets/

[48] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," arXiv preprint arXiv:1511.07122, 2015.
[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[50] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.
[51] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," arXiv preprint arXiv:1609.03499, 2016.
[52] O. Bousquet, S. Gelly, K. Kurach, O. Teytaud, and D. Vincent, "Critical hyper-parameters: No random, no cry," arXiv preprint arXiv:1706.03200, 2017.
[53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.
[54] W. Falcon, "Pytorch lightning," 2019. [Online]. Available: https://github.com/Lightning-AI/lightning
[55] F. Kamalov, L. Smail, and I. Gurrib, "Stock price forecast with deep learning," in 2020 International Conference on Decision Aid Sciences and Application (DASA), 2020, pp. 1098–1102.
[56] P. Nimmani, S. Vodithala, and V. Polepally, "Neural network based integrated model for information retrieval," in 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 2021, pp. 1286–1289.
[57] P. V. Matrenin, V. Z. Manusov, A. I. Khalyasmaa, D. V. Antonenkov, S. A. Eroshenko, and D. N. Butusov, "Improving accuracy and generalization performance of small-size recurrent neural networks applied to short-term load forecasting," Mathematics, vol. 8, no. 12, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/12/2169

SEONGJAE LEE received the B.S. degree in mechanical and information engineering from the University of Seoul, Korea in 2019.

Since 2021, he has been a Ph.D. student (integrated masters and doctoral program) in the Department of mechanical and information engineering and smart cities, University of Seoul, Korea. His research interests include parallel computing, numerical optimization, and deep learning.

TAEHYOUN KIM (M'06) received his B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1994, 1996, and 2001, respectively.

From 2001 to 2005, he was a research manager with the SoC Division, GCT Research, Inc. Since 2005, he has been a professor with the Department of Mechanical and Information Engineering, University of Seoul, Korea. His current research interests include real-time embedded systems, application-specific GPU optimization, and on-device AI systems.