WHEN NEURAL ODES MEET NEURAL OPERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Differential equation-based neural networks perform well in a variety of deep learning fields. Among those many methods, neural ordinary differential equations (NODEs) are one of the most fundamental work. NODEs have been applied to general downstream tasks such as image classification, time series classification, and image generation. The ODE function of NODEs can be understood as a special type of differential operators, which had been overlooked before. In this paper, therefore, we study the feasibility of modeling NODEs (or the ODE function of NODEs) as neural operators. Our neural operator-based methods are more rigorous than existing approaches when it comes to learning the differential operator (or the ODE function). To this end, we design a new neural operator structure called branched Fourier neural operator (BFNO), which is suitable for modeling the ODE function. It shows improved performance for several general machine learning tasks, as compared to existing various NODE models.

1 INTRODUCTION

Neural networks based on differential equations now show state-of-the-art performance in various deep learning fields (Chen et al., 2018; Song et al., 2020; Kidger et al., 2020; Morrill et al., 2021; Choi et al., 2021; Jhin et al., 2022; Kim et al., 2022). One exemplary work is neural ordinary differential fquations (NODEs (Chen et al., 2018)). NODEs are continuous-depth neural networks that learn the dynamics of hidden state $\mathbf{h}(t)$ from data. The following ODE function, $f(\mathbf{h}(t), t; \boldsymbol{\theta}_f)$ with learnable parameters $\boldsymbol{\theta}_f$, resides in their core parts:

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t; \boldsymbol{\theta}_f). \tag{1}$$

We then solve the initial value problem (IVP) in Equation 2 with the initial value $\mathbf{h}(t_0)$ with ODE solvers to produce the hidden state $\mathbf{h}(t_1)$ which can be used for downstream tasks:

$$\mathbf{h}(t_1) = \mathbf{h}(t_0) + \int_{t_0}^{t_1} f(\mathbf{h}(t), t; \boldsymbol{\theta}_f)dt. \tag{2}$$

Neural operators, which are often used for modeling operators for partial differential equations (PDEs), deal with mappings from a function space to another — these function spaces can be infinite dimensional under the regime of neural operators (Li et al., 2020b; Cao, 2021; Gupta et al., 2021; Rahman et al., 2022). Fourier neural operators (FNOs), one of the most prominent models in this realm, parameterize operators on the Fourier domain using kernel integral operators, which show fast and accurate inference. Interestingly, neural operators have been recently adopted to many other fields, e.g., FNOs have been recently used for designing a better vision transformer (Guibas et al., 2021).

Our approach: The ODE function in Equation 1 can be considered as applying the differential operator to $\mathbf{h}(t)$ in order to represent the right-hand side of ODEs. Therefore, we redesign NODEs by defining the ODE function f using the neural operator method. Researchers have habitually used conventional neural networks, such as convolutional neural networks (CNNs) and fully-connected networks (FCNs), to learn the differential operator (or the ODE function). In comparison with them, we employ neural operators that are able to more accurately learn the differential operator. However, we found that the naïve adaptation of recent neural operator methods, such as FNO, to NODEs does not improve model accuracy. Therefore, we propose our own neural operator architecture.

For empirical evaluations, we apply our method to various applications including image classification, time series classification, and image generation. For each task, we compare our method with state-of-the-art NODE and non-NODE-based baselines. In particular, our main competitors are enhanced NODE models, such as heavy ball NODE (HBNODE). Our branched Fourier neural operator Neural ODE (BFNO-NODE) outperforms all those advanced NODE models and some other non-NODE-based baselines. We summarize our contributions as follows:

1. **Branched Fourier neural operator (BFNO):** We design the ODE function of NODEs using our proposed branched Fourier neural operator (BFNO) method. Our BFNO is specialized for being used as ODE functions. In particular, we do not use the fixed architecture of FNOs, i.e., applying a low-pass filter to the Fourier transform outcome, followed by a convolutional operation, since it is rather restrictive to be used for general machine learning tasks. Therefore, our BFNO uses dynamic global convolutional operations with multiple kernels. To our knowledge, we are the first using neural operators to design the ODE function of NODEs.
2. **High performance in general machine learning tasks:** In comparison with existing non-operator-based approaches, our BFNO is able to more accurately learn the ODE function, which is basically a differential operator, for various tasks. To this end, we experiment with various general machine learning tasks of image classification, time series classification, and image generation. Our method outperforms various existing NODE and non-NODE-based methods by 7 percent point in image classification, 3 percent point in time series classification about test accuracy, and 9.2% in image generation (in terms of those tasks' standard metrics).

2 Related Work

Continuous-depth neural networks: Continuous-depth neural networks allow greater modeling flexibility and have been investigated as a potential replacement for traditional deep feed-forward neural networks (Weinan, 2017; Haber & Ruthotto, 2017; Lu et al., 2018). Neural ODEs (Chen et al., 2018) can model continuous depth neural networks. By the continuous depth property, NODEs describe the change of the hidden state $\mathbf{h}(t)$ over time. In particular, NODEs use the adjoint sensitivity method to address the main technical challenge of training continuous-depth networks. Many continuous-depth models that use the same computational formalism, such as neural controlled differential equations (NCDEs (Kidger et al., 2020)), neural stochastic differential equations (NSDEs (Liu et al., 2019)), and so on, have been proposed. In addition, NODE-based models with enhanced ODE function architectures have achieved significant improvements in various tasks in comparison with the original NODE design. Augmented NODEs (ANODEs (Dupont et al., 2019)) suggested solving the homeomorphic limitation of ODEs with adding extra dimensions. Second-order NODEs (SONODEs (Norcliffe et al., 2020)) showed that they can expand the adjoint sensitivity approach to the second-order ODEs efficiently. In order to further enhance the training and inference of NODEs, heavy ball NODEs (HBNODEs (Xia et al., 2021)) model the dynamics with the conventional momentum method. Adaptive moment estimation NODEs (AdamNODEs (Cho et al., 2022)) use an enhanced momentum-based method to define the ODE function. In this paper, we propose a novel continuous-depth NODE architecture with the infinite dimensional property based on our proposed BFNO. Our enhanced NODE architecture, called BFNO-NODE, outperforms existing NODE enhancements.

Neural operators: Expressing and analyzing changes in phenomena over time and space as PDEs are an important issue in natural science and engineering. However, solving PDE problems is a mathematically difficult task. In the field of numerical analysis, these problems are approached using various numerical methods (Smith & Smith, 1985; Reddy, 2019). These methods perform well in many areas where PDE problems need to be analyzed, but there are fundamental challenges: i) it takes a lot of computational cost to solve the problem, ii) the higher its required accuracy, the longer its computation time, and iii) finally, it is difficult to analyze various input functions.

To overcome these limitations, neural operators (Lu et al., 2019; Kovachki et al., 2021) have recently been proposed in the field of deep learning. Neural operators aim to learn the (inverse) differential operator of PDEs. Furthermore, they also have the infinite dimensional property.

Recently, models with repetitive operator-based layers have been proposed in the field of neural operators (Li et al., 2020b;c; Wen et al., 2022). Typically, FNOs first define a kernel integral operator layer in the Fourier domain and repeatedly stack multiple such layers. In the field of computer vision, there was a study that improved the vision transformer (Dosovitskiy et al., 2020) using FNOs. Adaptive Fourier neural operator (AFNO) was proposed in Guibas et al. (2021) by replacing the self-attention layers of the vision transformer with adaptive FNO layers. However, these operator architectures do not perform well when they are used in the ODE function in our preliminary experiments. Therefore, we propose our branched Fourier neural operator concept.

3 Preliminaries

Neural ordinary differential equations (NODEs): In order to determine $\mathbf{h}(t_1)$ from $\mathbf{h}(t_0)$, NODEs solve the integral problem in Equation 2, where the ODE function $f(\mathbf{h}(t), t; \boldsymbol{\theta}_f)$ is a neural network used to approximate $\frac{d\mathbf{h}(t)}{dt}$ (cf. Equation 1). Once f is trained, NODEs rely on numerical ODE solvers to solve the integral problem, such as the explicit Euler method, the Dormand-Prince (DOPRI (Dormand & Prince, 1980)) method. ODE solvers discretize the time variable t and transform an integral into numerous additions. One distinguished characteristic of NODEs is that the gradient of loss w.r.t. $\boldsymbol{\theta}_f$, denoted $\nabla_{\boldsymbol{\theta}_f} C = \frac{dC}{d\boldsymbol{\theta}_f}$, where C is a task-dependent cost function, can be calculated by a reverse-mode integration, which has $\mathcal{O}(1)$ space complexity. This gradient calculation method is known as the adjoint sensitivity method.

Operator learning and discretization of function: Let S be a bounded, open set which is a subspace of $\mathbb{R}^d$, and let $P = P(S; \mathbb{R}^{d_p})$ and $Q = Q(S; \mathbb{R}^{d_q})$ be Banach spaces which define a function mapping from input $x \in S$ to output whose dimensions are d_p and d_q respectively. Then, a non-linear operator $\mathcal{L} : P \to Q$, where P and Q are the continuous function spaces, can be defined. By discretizing with finite numbers of observations, the function spaces can be represented in grid forms. At the end, neural operators learn a parameterized mapping $\mathcal{L}_\theta : P \to Q$ such that:

$$\mathcal{L}_\theta(P)(x) = Q(x), \qquad \forall x \in S. \tag{3}$$

For the given two functions, $P(x)$ and $Q(x)$ with $x \in S$, the operator $\mathcal{L}_\theta$ can map these function spaces. As such, NODE's time-derivative operator $\mathcal{D}_f = \frac{d(\cdot)}{dt}$ can be understood as a function mapping from $\mathbf{h}(t)$ to $\frac{d\mathbf{h}(t)}{dt}$, where t denotes a temporal coordinate. This differential operator can be regarded as a special case of the general operator representation. Therefore, we can design the ODE function f of NODEs as an operator-based network.

Kernel integral operators: As a way of expressing an operator, one can define the kernel integral operator $\mathcal{K}$ as following:

$$\mathcal{K}(q)(x) := \int_S \kappa(x, y) q(y) dy, \qquad \forall x \in S, \tag{4}$$

where κ is a continuous kernel function. If we use the Green's kernel $\kappa(x, y) = \kappa(x-y)$, Equation 4 becomes the following special global convolution operator:

$$\mathcal{K}(q)(x) := \int_S \kappa(x - y) q(y) dy, \qquad \forall x \in S. \tag{5}$$

Fourier neural operators (FNOs): For an efficient parameterization of the kernel κ, FNOs rely on the Fourier transform as a projection of a function onto the Fourier domain (Li et al., 2020a). Given an input function $\mathbf{g}(x)$, for instance, FNOs use the following kernel integral operator $\mathcal{K}$ parameterized by ψ (Li et al., 2020a):

$$(\mathcal{K}(\psi)\mathbf{g})(x) = \mathcal{F}^{-1}(\mathbf{R}_\psi \odot \mathcal{F}(\mathbf{g}))(x), \qquad \forall x \in S, \tag{6}$$

where $\mathcal{F}$ denotes the fast Fourier transform and $\mathcal{F}^{-1}$ its inverse. $\odot$ denotes the elementwise multiplication, and $\mathbf{R}_\psi$ denotes a tensor representing a global convolutional kernel. The global convolutional kernel $\mathbf{R}_\psi$ is the only trainable parameter, i.e., $\psi \equiv \mathbf{R}_\psi$.

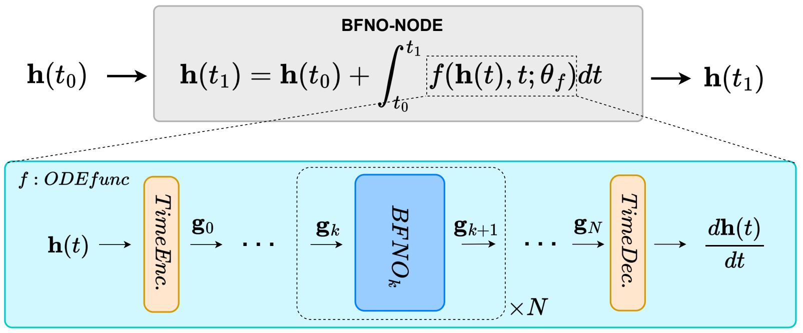


Figure 1: The overall architecture of BFNO-NODE

4 Proposed Methods

In this section, we describe our motivations, followed by the detailed model architecture.

4.1 Motivations

The ODE function f in Equation 1 is to learn the time-derivative operator of the hidden state $\mathbf{h}(t)$, in which case we can consider that f is a special operator suitable for defining NODEs. Therefore, our proposed NODE concept can be written as follows:

$$\mathbf{h}(t_1) = \mathbf{h}(t_0) + \int_{t_0}^{t_1} \mathcal{D}_f(\mathbf{h}(t), t; \boldsymbol{\theta}_f)\mathrm{d}t, \tag{7}$$

where $\mathcal{D}_f$ is the time-derivative operator. However, we found that FNO (Equation 6) is not suitable for use in general machine learning tasks (see Section 5.4 for detailed discussion). Therefore, we propose our own extended FNO method, called branched Fourier neural operator (BFNO).

Many NODE-based papers have habitually used conventional architectures, e.g., fully-connected layers, convolutional layers, and rectified linear units (ReLUs), to define the ODE function (Dupont et al., 2019; Norcliffe et al., 2020; Xia et al., 2021; Cho et al., 2022). In comparison with them, our method provides a more rigorous way for defining the ODE function (or the time-derivative operator). Our main goal in this paper is to enhance the model accuracy by learning the ODE function more accurately.

4.2 BFNO-NODE

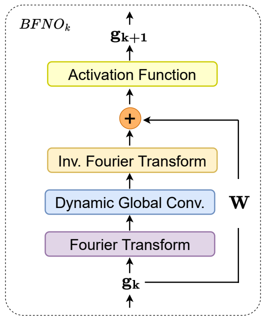
The overall model architecture of our proposed BFNO-NODE is shown in Figure 1. There are three major points in our design in contrast to the original NODE design: i) the neural operator-based ODE function, ii) the BFNO layer, and iii) the dynamic global convolution.

Neural operator-based ODE functions: We propose the following ODE function with our newly designed BFNO layer:

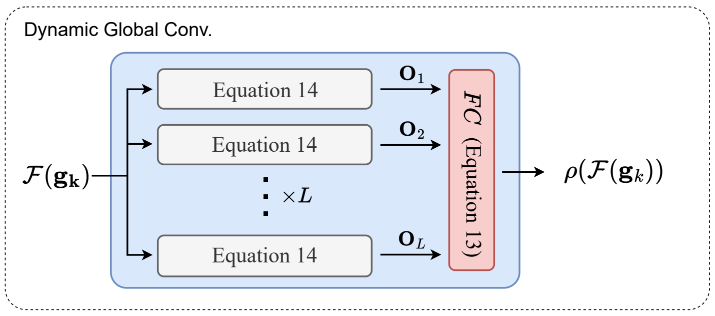
$$\begin{aligned} \mathbf{g}_0 &= TimeEnc(\mathbf{h}(t), t), \\ \mathbf{g}_{k+1} &= BFNO_k(\mathbf{g}_k), 0 \leq k \leq N-1, \\ \frac{d\mathbf{h}(t)}{dt} &= TimeDec(\mathbf{g}_N), \end{aligned} \tag{8}$$

where $BFNO_k$ is the k-th BFNO layer. Both the time-dependent encoder ($TimeEnc$) and the time-dependent decoder ($TimeDec$) consist of a fully-connected layer, respectively. The size of encoded vector $\mathbf{g}_k$, $0 \leq k \leq N$, is a hyperparameter. We use $\dim(\mathbf{g})$ to denote the size of $\mathbf{g}_k$. The output size of the decoder is the size of $\mathbf{h}(t)$. The size of $\mathbf{h}(t)$, denoted $\dim(\mathbf{h})$, is also a hyperparameter.

Branched Fourier neural operator (BFNO) layers: Figure 2 (a) shows the structure of the BFNO layer. The update process between the input value $\mathbf{g}_k$ and the output value $\mathbf{g}_{k+1}$ of the k-th BFNO



(a) BFNO layer



(b) Dynamic global convolution

Figure 2: The detailed proposed architecture. We perform the Fourier transform, the dynamic global convolutional operation, and the inverse Fourier transform to process $\mathbf{g}_k$ in conjunction with one additional transformation by the transformation matrix $\mathbf{W}$. (cf. Equation 12)

layer is expressed as follows:

$$\mathbf{g}_{k+1}(x) = BFNO_k\big(\mathbf{g}_k(x)\big), \tag{9}$$
$$\Rightarrow \sigma\big((\mathcal{K}(\mathbf{h}, t; \psi)\mathbf{g}_k)(x) + \mathbf{W}\mathbf{g}_k(x)\big), \tag{10}$$
$$\Rightarrow \sigma\big(\mathcal{F}^{-1}(\rho(\mathcal{F}(\mathbf{g}_k)))(x) + \mathbf{W}\mathbf{g}_k(x)\big), \tag{11}$$

where the first term is a kernel integral operator parameterized by ψ, and the second term is a linear transformation parameterized by $\mathbf{W}$. σ is an activation function. ρ is the dynamic global convolutional operation for the input $\mathbf{g}_k$, which we will describe shortly.

Under the regime of our proposed BFNO, $\mathbf{g}$ is theoretically a spatial function. However, infinite-dimensional spatial functions cannot be processed by modern computers and therefore, all neural operator methods implement their discretized versions, which is also the case for our method. As a result, our actual implementations are as follows:

$$\mathbf{g}_{k+1} = \sigma\big(\mathcal{F}^{-1}(\rho(\mathcal{F}(\mathbf{g}_k))) + \mathbf{W}\mathbf{g}_k\big), \tag{12}$$

where $\mathbf{g}_k$ (resp. $\mathbf{g}_{k+1}$) is an input (resp. output) tensor. $\mathbf{W} \in \mathbb{R}^{\dim(\mathbf{g})\times\dim(\mathbf{g})}$ is a linear transformation matrix, whose input and output sizes are all the same as that of $\mathbf{g}_k$.

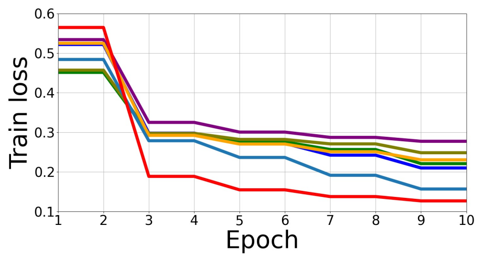
Dynamic global convolution ρ: We note that in our method, the dynamic global convolutional operation ρ is defined by the following method:

$$\rho(\mathcal{F}(\mathbf{g}_k)) = FC(\mathbf{O}_1, \mathbf{O}_2, \cdots, \mathbf{O}_L), \tag{13}$$
$$\mathbf{O}_i = \mathbf{R}_i \odot \mathcal{F}(\mathbf{g}_k), \tag{14}$$

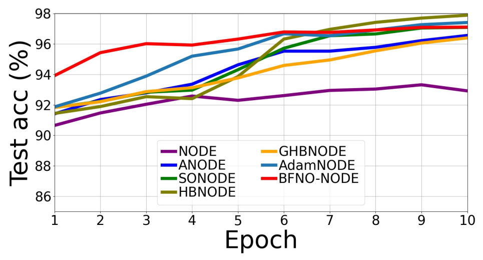
where FC means a fully-connected layer, and $\mathbf{O}_i$ is the elementwise multiplication between the i-th global convolutional kernel $\mathbf{R}_i$ and $\mathcal{F}(\mathbf{g}_k)$. Therefore, FC is to learn how to dynamically aggregate the L different global convolutional processing outcomes, i.e., $\{\mathbf{O}_1, \mathbf{O}_2, \cdots, \mathbf{O}_L\}$, where L is a hyperparameter. The learnable parameters (kernels) in $\mathbf{R}_i$, $1 \leq i \leq L$ and FC constitute our proposed dynamic global convolutional operation with respect to the input $\mathcal{F}(\mathbf{g}_k)$.

5 Experiments

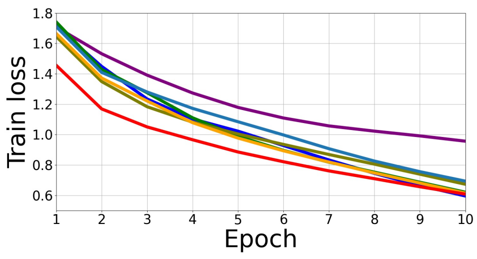
In this section, we compare our model with existing baseline models for three general machine learning tasks: image classification, time series classification, and image generation. In comparison with the experiment set of HBNODE (Xia et al., 2021), our experiment set covers more general machine learning tasks and several more datasets. Our software and hardware environments are as follows: Ubuntu 18.04 LTS, Python 3.6, TorchDiffEq, PyTorch 1.10.2, CUDA 11.4, NVIDIA Driver 470.74, i9 CPU, and NVIDIA RTX A6000.



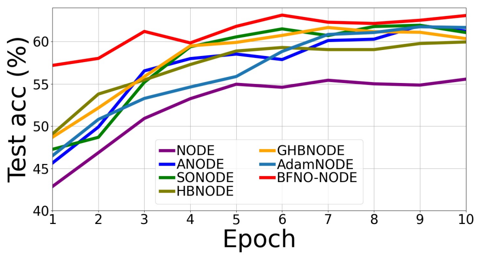
(a) Train loss in MNIST



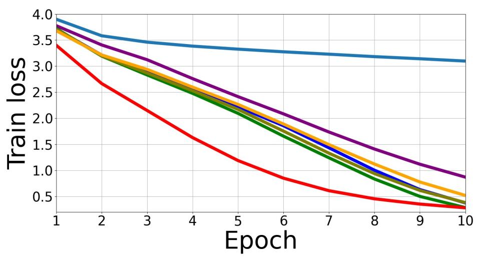
(b) Test accuracy in MNIST



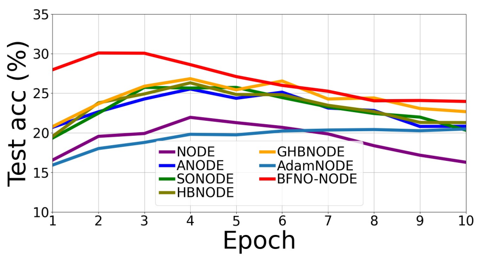
(c) Train loss in CIFAR-10



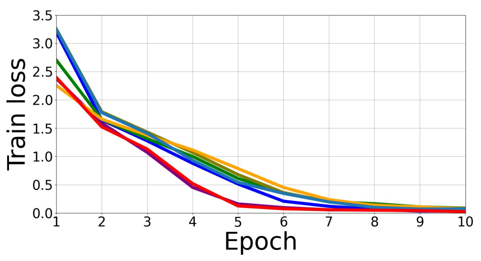
(d) Test accuracy in CIFAR-10



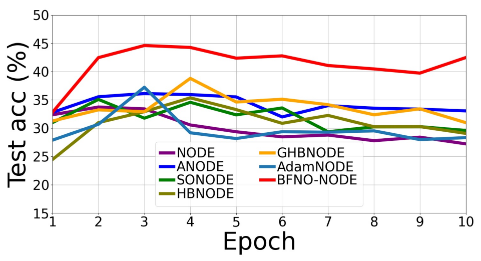
(e) Train loss in CIFAR-100



(f) Test accuracy in CIFAR-100



(g) Train loss in STL-10



(h) Test accuracy in STL-10

Figure 3: Train loss and test accuracy of various methods

5.1 Image Classification

Datasets: We test baselines and our model with the following four image classification benchmarks: MNIST (LeCun et al., 2010), CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009), and STL-10 (Coates et al., 2011).

Experimental environments: In this experiment, NODE (Chen et al., 2018), ANODE (Dupont et al., 2019), SONODE (Norcliffe et al., 2020), HBNODE, GHBNODE (Xia et al., 2021), and AdamNODE (Cho et al., 2022) are used as baselines, which covers a prominent set of enhancements for NODEs.

For MNIST, we use a learning rate of 0.0001 and a batch size of 32, and for other datasets, a learning rate of 0.001 and a batch size of 64. In all datasets, we use DOPRI with its default tolerance settings to solve the integral problem. For HBNODE and GHBNODE, the damping parameter γ is set to

Table 1: Time series classification (HumanAct.)

Method	Accuracy
RNN-Δ_t	0.797 $\pm$ 0.003
RNN-Impute	0.795 $\pm$ 0.008
RNN-D	0.800 $\pm$ 0.010
GRU-D	0.806 $\pm$ 0.007
RNN-VAE	0.343 $\pm$ 0.040
ODE-RNN	0.829 $\pm$ 0.016
HBNODE-RNN	0.821 $\pm$ 0.015
Latent-ODE (RNN enc.)	0.835 $\pm$ 0.010
Latent-ODE (ODE enc.)	0.846 $\pm$ 0.013
Latent-BFNO-NODE (ODE enc.)	**0.874 $\pm$ 0.004**

Table 2: Time series classification (PhysioNet)

Method	AUROC
RNN-Δ_t	0.787 $\pm$ 0.014
RNN-Impute	0.764 $\pm$ 0.016
RNN-D	0.807 $\pm$ 0.003
GRU-D	0.818 $\pm$ 0.008
RNN-VAE	0.515 $\pm$ 0.040
ODE-RNN	0.833 $\pm$ 0.009
HBNODE-RNN	0.513 $\pm$ 0.001
Latent-ODE (RNN enc.)	0.781 $\pm$ 0.018
Latent-ODE (ODE enc.)	0.829 $\pm$ 0.004
Latent-BFNO-NODE (ODE enc.)	**0.834 $\pm$ 0.005**

$sigmoid(\pi)$, where π is trainable and initialized to -3. For all NODE-based baselines, we use 3 convolutional layers to define their ODE functions and for our method, we use 2 BFNO layers and $L = 2$. For fair comparison, their model sizes (in terms of the number of parameters) are almost same. (cf. Table 4 in Appendix B).

In order to evaluate in various aspects, we consider the following evaluation metrics: i) the convergence speed of train loss, and most importantly ii) test accuracy. The number of function evaluations (NFEs) is an effective metric to measure the complexity of the forward and backward computation for NODE-based models. However, our work's main focus is not to reduce NFEs but to improve accuracy. Thus, we refer readers to Appendix A for additional analyses on NFEs.

Experimental results: Figure 3 shows the train loss curve and the test accuracy in each dataset. Our method, highlighted in red, shows the fastest convergence speed while achieving smaller loss values from the beginning of the training process in many cases. For MNIST and CIFAR-100, our method shows far better train curves than those of other baselines. Other baselines' train loss values are much higher than those of our method in MNIST. Our method also show good results for the test accuracy. In CIFAR-10, CIFAR-100 and STL-10, our method method consistently outperforms other baselines from the beginning of the training process to the end. For MNIST, however, HBNODE shows a higher accuracy than ours (although its final loss is worse than ours).

5.2 Time Series Classification

Datasets: HumanActivity (Kaluža et al., 2010) and Physionet (Silva et al., 2010) benchmark datasets are used to train and evaluate models for time series classification. The HumanActivity dataset includes the information from five persons with four sensors at their left ankle, right ankle, belt, and chest while performing a variety of activities (such as walking, falling, lying down, rising from a lying position, etc). We let them to repeat five times in order to gather trustworthy data. In this experiment, we classify each person's input into one of the seven activities.

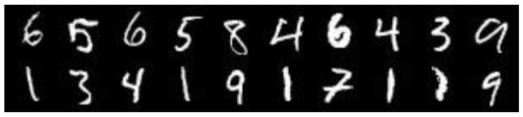
PhysioNet consists of 8,000 time series samples and is used to forecast the mortality of intensive care unit (ICU) populations. The dataset had been compiled from 12,000 ICU stays. They documented up to 42 variables and removed brief stay of less then 48 hours. Additionally, the data recorded in this way have a timestamp of the time elapsed since admission to the ICU. In this task, the patient's life or death is determined based on records.

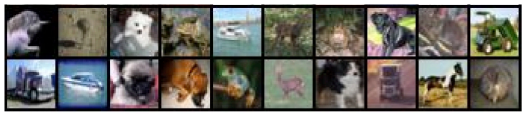
Experimental environments: We consider a variety set of RNN-based models, ODE-RNN, and Latent-ODE, following the evaluation protocol in Rubanova et al. (2019). In addition, we build one more baseline by replacing the ODE function of ODE-RNN with the HBNODE-based design, denoted HBNODE-RNN in our experimental result table. Our model, Latent-BFNO-NODE, is implemented by replacing the ODE function of Latent-ODE (ODE enc.) with the BFNO-NODE-based design. In this process, the hyperparameter L is fixed to 1 for efficiency.

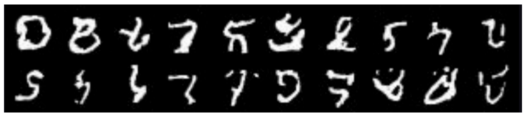
We use accuracy for HumanActivity (since the dataset is balanced). AUROC is used for Physionet, considering its imbalanced nature.

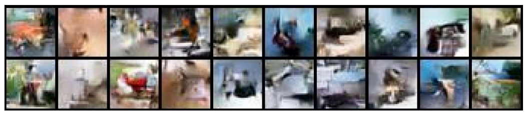
Table 3: Negative log-likelihood (in bits/dim) on test images. Lower is better.

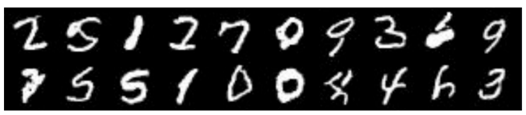
Method	MNIST	CIFAR-10
REALNVP (Dinh et al., 2016)	1.06	3.49
I-RESNET (Behrmann et al., 2019)	1.05	3.45
GLOW (Kingma & Dhariwal, 2018)	1.05	3.35
FFJORD (Grathwohl et al., 2018)	0.99	3.40
FFJORD-RNODE (Finlay et al., 2020)	0.97	3.38
FFJORD-BFNO-RNODE	**0.88**	**3.33**


(a) Real images (MNIST)


(b) Real images (CIFAR-10)


(c) FFJORD-RNODE (MNIST)


(d) FFJORD-RNODE (CIFAR-10)


(e) FFJORD-BFNO-RNODE (MNIST)

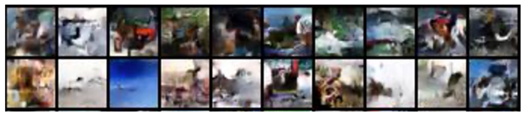

(f) FFJORD-BFNO-RNODE (CIFAR-10)

Figure 4: Real and produced samples from models. MNIST (left) and CIFAR-10 (right).

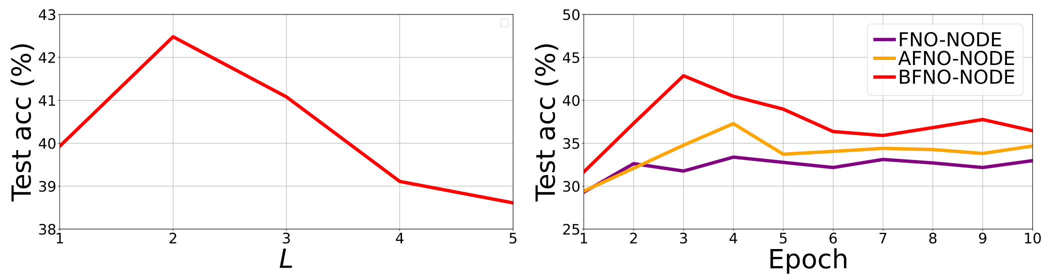
Experimental results: In Table 1, we summarize the results for HumanActivity. In general, RNN-based models do not show good accuracy. RNN-VAE shows the worst accuracy. NODE-based models significantly outperforms them. Our proposed Latent-BFNO-NODE shows the best accuracy, followed by Latent-ODE. Similar patterns can be observed in Table 2 for PhysioNet.

5.3 Image Generation

Datasets: For our image generation task, we use MNIST and CIFAR10. These two datasets are the most widely used for conducting generative task experiments for NODE-based and invertible neural network-based models (Grathwohl et al., 2018).

Experimental environments: We consider the baselines in Finlay et al. (2020), which cover a prominent set of NODE-based and invertible neural network-based models. The ODE function $f(\mathbf{h}(t), t; \boldsymbol{\theta}_f)$ in these baselines consist of 4 convolutional layers with 3×3 kernels and softplus activations. These layers contain 64 hidden units, and the time t is concatenated to the spatial input vector $\mathbf{h}(t)$ (as a side channel). The Gaussian Monte-Carlo trace estimator is used to calculate the log-probability of a generated sample via the change of variable theorem. Using the Adam optimizer with a learning rate of 0.001, we train on a single GPU with a batch size of 200 and for 100 epochs. We use a model with the kinetic regularization proposed in Finlay et al. (2020), and we propose FFJORD-BFNO-RNODE by replacing ODE function f of FFJORD-RNODE with BFNO structure. Our model consists of 3 BFNO layers and softplus activation in between them. The hyperparameter L is fixed to 3. In addition, the transformation $\mathbf{W}$ of BFNO is replaced to a convolutional operation with a 3×3 kernel since this task is a generation.

Experimental results: In Table 3, it can be seen that the negative log-likelihood (NLL) of our model is better than those of all other baselines by large margins. In Figure 4, in addition, we show real images and fake images by our method and FFJORD-RNODE. All methods show acceptable fake images (although their quality is worse than other large-scale generative models, such as diffusion



(a) Ablation study on the number of parallelized global convolutions L (STL-10)

(b) Ablation study on the neural operator type (STL-10)

Figure 5: Ablation study on the two key design options of BFNO. Other figures are in Appendix F.

models (Song et al., 2020; Ho et al., 2020)). In general, NODE-based and invertible neural network-based models do not show better generations than diffusion models.

5.4 Ablation Studies

We conduct two key ablation studies: 1) checking the model accuracy by changing the number of parallelized global convolutions in BFNO, denoted L, and 2) comparing BFNO with existing neural operator methods, such as FNO and AFNO.

Accuracy by the number of parallelized global convolutions (L) in BFNO: As shown in Figure 2 (b), a BFNO layer has multiple parallelized global convolutions which will be later merged by a fully-connected layer. The number of parallelized global convolutions in BFNO, denoted L, is a key hyperparameter. Figure 5 (a) shows the model accuracy for various settings for L on image classification with STL-10. Too many or small settings lead to sub-optimal outcomes and $L = 2$ is the best configuration in this ablation study. We think that using too many convolutions results in overfitting, which degrades the model accuracy.

Comparison with FNO and AFNO: Figure 5 (b) compares our method with FNO and AFNO. As shown, our BFNO-NODE consistently outperforms them throughout the entire training epoch. FNO was developed to model PDE operators, and AFNO aims at improving the vision transformer by replacing its spatial mixer with a special neural operator. Since not being designed for NODEs, however, they fail to show effectiveness in our experiments when being used to define the ODE function of NODEs. Similar patterns are observed in other datasets as well, and the results are shown in Appendix F.

6 Conclusions

Enhancing NODEs by adopting advanced ODE function architectures has been one active research trend for the past couple of years. However, existing approaches did not pay attention to the fact that the ODE function learns a special differential operator. In this work, we presented how neural operators can be used to define the ODE function and learn the differential operator. However, the naïve adoption of existing neural operators, such as FNO and AFNO, does not enhance NODEs. To this end, we designed a special neural operator architecture, called branched Fourier neural operators (BFNOs). Our dynamic global convolutional method with multiple parallelized global convolutions significantly improves the efficacy of various NODE-based models for three general machine learning tasks: image classification, time series classification, and image generation. Since the role of the ODE function is, in fact, applying a differential operator to the input $\mathbf{h}(t)$, our operator-based approach naturally shows the best fit in our experiments.

Limitations: Although significantly enhancing the efficacy of NODEs for various tasks, our approach sometimes increases NFEs, resulting in larger computation in comparison with existing approaches (see Appendix A). It is not straightforward to achieve both the high efficacy and the low computation at the same time. However, we believe that there exists an operator-based ODE function definition which simultaneously enhances the efficacy and the efficiency of NODEs.

7 Ethics Statement

There is no particular ethically problematic element in our paper.

8 Reproducibility Statement

All implementations of our proposed method can be reproduced by referring to the attached README.md. The best hyperparameters and detailed model architectures are recorded in Appendix C, D, E. The source code used in our paper will be made available to contribute to the community.

References

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2019.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 34:24924–24940, 2021.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Seunghyeon Cho, Sanghyun Hong, Kookjin Lee, and Noseong Park. Adamnodes: When neural ode meets adaptive moment estimation. *arXiv preprint arXiv:2207.06066*, 2022.

Jeongwhan Choi, Jinsung Jeon, and Noseong Park. Lt-ocf: learnable-time ode-based collaborative filtering. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 251–260, 2021.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19 – 26, 1980.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *ICML*, 2020.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34:24048–24062, 2021.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Sheo Yon Jhin, Jaehoon Lee, Minju Jo, Seungji Kook, Jinsung Jeon, Jihyeon Hyeong, Jayoung Kim, and Noseong Park. Exit: Extrapolation and interpolation-based neural controlled differential equations for time-series classification and forecasting. In *Proceedings of the ACM Web Conference 2022*, pp. 3102–3112, 2022.

Boštjan Kaluža, Violeta Mirchevska, Erik Dovgan, Mitja Luštrek, and Matjaž Gams. An agent-based approach to care in independent living. In *International joint conference on ambient intelligence*, pp. 177–186. Springer, 2010.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

Jayoung Kim, Chaejeong Lee, Yehjin Shin, Sewon Park, Minjung Kim, Noseong Park, and Jihoon Cho. Sos: Score-based oversampling for tabular data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 762–772, 2022.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020c.

Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pp. 3276–3285, 2018.

James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pp. 7829–7838. PMLR, 2021.

Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes. *Advances in Neural Information Processing Systems*, 33:5911–5921, 2020.

Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

Junuthula Narasimha Reddy. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *Comput Cardiol*, 39:245–248, 2010.

Gordon D Smith and Gordon D Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

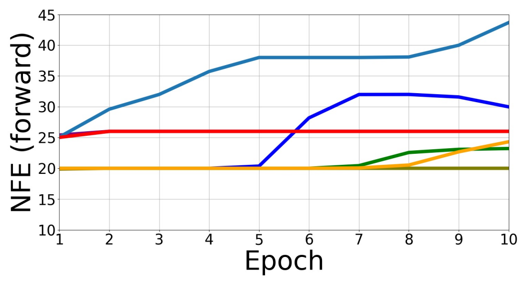
E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M Benson. U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

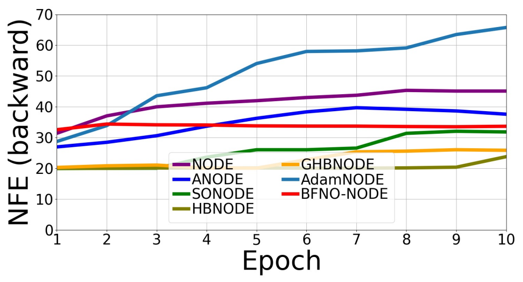
Hedi Xia, Vai Suliafu, Hangjie Ji, Tan Nguyen, Andrea Bertozzi, Stanley Osher, and Bao Wang. Heavy ball neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

A Forward and Backward NFE in Image Classification

In general, our BFNO-NODE's NFE values are comparable to those of NODE and ANODE. Since our main focus is not decreasing NFEs but increasing the model effectiveness by learning the differential operation f in a rigorous manner, we consider that it is normal that our method marks comparable NFEs to existing methods.

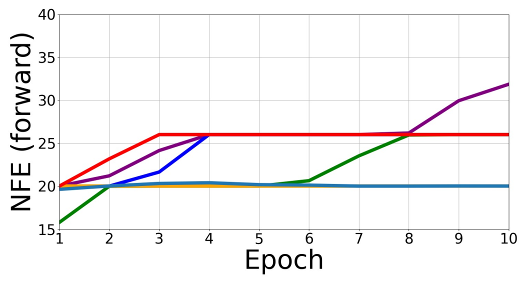


(a) Train forward NFE

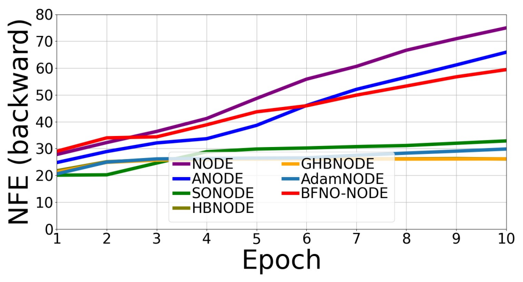


(b) Train backward NFE

Figure 6: Image classification MNIST result

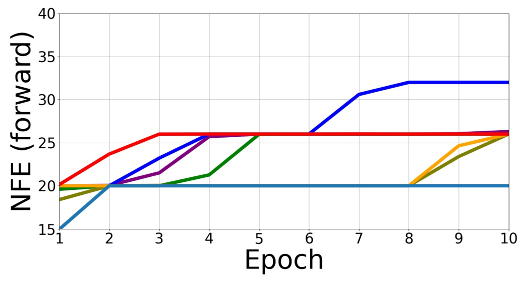


(a) Train forward NFE

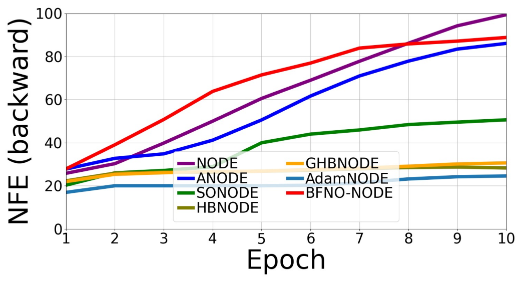


(b) Train backward NFE

Figure 7: Image classification CIFAR-10 result

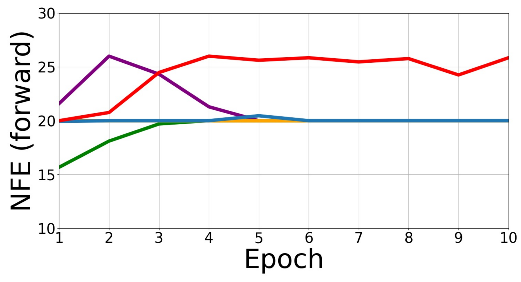


(a) Train forward NFE

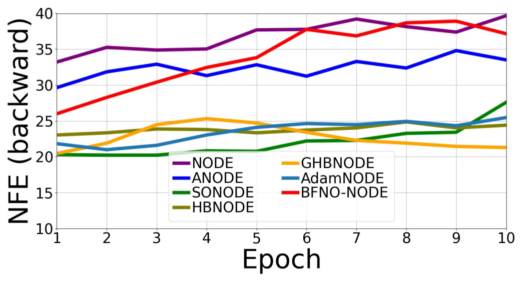


(b) Train backward NFE

Figure 8: Image classification CIFAR-100 result



(a) Train forward NFE



(b) Train backward NFE

Figure 9: Image classification STL-10 result

B Model Size

Table 4: Model size in terms of the number of parameters

	NODE	ANODE	SONODE	HBNODE	GHBNODE	AdamNODE	BFNO-NODE
MNIST	85,315	85,462	86,179	85,931	85,235	85,832	85,019
CIFAR-10	173,611	172,452	171,635	172,916	172,916	173,070	171,133
CIFAR-100	646,021	645,121	645,081	646,338	646,338	643,776	644,547
STL-10	521,512	520,180	521,532	521,248	521,247	525,946	514,291

In the image classification task, we confirm the model performance on four datasets: MNIST, CIFAR-10, CIFAR-100, and STL-10. To compare the performance of our model with six NODE baselines, we set the size of the models as shown in Table 4.

C Detailed Settings for Image Classification Experiments

C.1 Architecture of the ODE function f

Table 5: MNIST structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$2 \times 28 \times 28$	$47 \times 28 \times 28$
2	ReLU($BFNO$)	$47 \times 28 \times 28$	$47 \times 28 \times 28$
3	ReLU($BFNO$)	$47 \times 28 \times 28$	$47 \times 28 \times 28$
4	$BFNO$	$47 \times 28 \times 28$	$47 \times 28 \times 28$
5	$TimeDec$	$47 \times 28 \times 28$	$1 \times 28 \times 28$

Table 6: CIFAR-10 structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$4 \times 32 \times 32$	$76 \times 32 \times 32$
2	ReLU($BFNO$)	$76 \times 32 \times 32$	$76 \times 32 \times 32$
3	ReLU($BFNO$)	$76 \times 32 \times 32$	$76 \times 32 \times 32$
4	$BFNO$	$76 \times 32 \times 32$	$76 \times 32 \times 32$
5	$TimeDec$	$76 \times 32 \times 32$	$3 \times 32 \times 32$

Table 7: CIFAR-100 structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$4 \times 32 \times 32$	$118 \times 32 \times 32$
2	ReLU($BFNO$)	$118 \times 32 \times 32$	$118 \times 32 \times 32$
3	ReLU($BFNO$)	$118 \times 32 \times 32$	$118 \times 32 \times 32$
4	$BFNO$	$118 \times 32 \times 32$	$118 \times 32 \times 32$
5	$TimeDec$	$118 \times 32 \times 32$	$3 \times 32 \times 32$

Table 8: STL-10 structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$4 \times 96 \times 96$	$99 \times 96 \times 96$
2	ReLU($BFNO$)	$99 \times 96 \times 96$	$99 \times 96 \times 96$
3	ReLU($BFNO$)	$99 \times 96 \times 96$	$99 \times 96 \times 96$
4	$BFNO$	$99 \times 96 \times 96$	$99 \times 96 \times 96$
5	$TimeDec$	$99 \times 96 \times 96$	$3 \times 96 \times 96$

C.2 Best Hyperparameters

For each of the reported results, we list the best hyperparameter as follows:

- For MNIST, learning rate = 0.0001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 2, N = 3;
- For CIFAR-10, learning rate = 0.001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 2, N = 3;
- For CIFAR-100, learning rate = 0.001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 2, N = 3;
- For STL-10, learning rate = 0.001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 2, N = 3;

D Detailed Settings for Time Series Classification Experiments

D.1 Architecture of the ODE function f

Table 9: Human Activity structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$15 \times 3 \times 50$	$500 \times 3 \times 50$
2	Tanh($BFNO$)	$500 \times 3 \times 50$	$500 \times 3 \times 50$
3	$TimeDec$	$500 \times 3 \times 50$	$15 \times 3 \times 50$

Table 10: PhysioNet structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$15 \times 3 \times 50$	$50 \times 3 \times 50$
2	Tanh($BFNO$)	$50 \times 3 \times 50$	$50 \times 3 \times 50$
3	$TimeDec$	$50 \times 3 \times 50$	$15 \times 3 \times 50$

D.2 Best Hyperparameters

For each of the reported results, we list the best hyperparameter as follows:

- For Human Activity, learning rate = 0.0001, relative tolerance = 0.001, absolute tolerance = 0.0001, L = 1, N = 1.
- For PhysioNet, learning rate = 0.0001, relative tolerance = 0.001, absolute tolerance = 0.0001, L = 1, N = 1.

E Detailed Settings for Image Generation Experiments

E.1 Architecture of the ODE function f

Table 11: MNIST structure

Layer	Desgin	Input size	Output size
1	$TimeEnc$	$2 \times 28 \times 28$	$100 \times 28 \times 28$
2	SoftPlus($BFNO$)	$100 \times 28 \times 28$	$100 \times 28 \times 28$
3	SoftPlus($BFNO$)	$100 \times 28 \times 28$	$100 \times 28 \times 28$
4	SoftPlus($BFNO$)	$100 \times 28 \times 28$	$100 \times 28 \times 28$
5	$TimeDec$	$100 \times 28 \times 28$	$1 \times 28 \times 28$

Table 12: CIFAR-10 structure

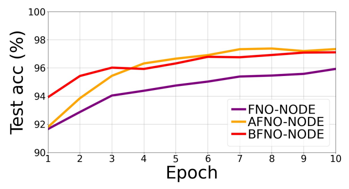
Layer	Desgin	Input size	Output size
1	$TimeEnc$	$4 \times 32 \times 32$	$100 \times 32 \times 32$
2	SoftPlus($BFNO$)	$100 \times 32 \times 32$	$100 \times 32 \times 32$
3	SoftPlus($BFNO$)	$100 \times 32 \times 32$	$100 \times 32 \times 32$
4	SoftPlus($BFNO$)	$100 \times 32 \times 32$	$100 \times 32 \times 32$
5	$TimeDec$	$100 \times 32 \times 32$	$3 \times 32 \times 32$

E.2 Best Hyperparameters

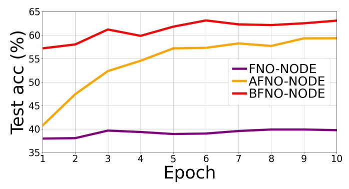
For each of the reported results, we list the best hyperparameter as follows:

- For MNIST, learning rate = 0.0001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 3, N = 1.
- For CIFAR-10, learning rate = 0.0001, relative tolerance = 0.001, absolute tolerance = 0.001, L = 3, N = 1.

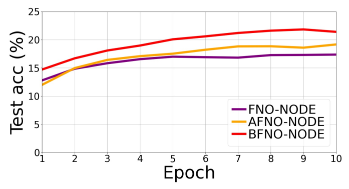
F Detailed Ablation Studies



(a) Ablation study on the neural operator type (MNIST)



(b) Ablation study on the neural operator type (CIFAR-10)



(c) Ablation study on the neural operator type (CIFAR-100)

Figure 10: Ablation study on the two key design options of BFNO

As shown in Figure 5, the performance of the BFNO structure as the ODE function f is superior to that of the other two baselines, FNO and AFNO in other datasets. In this section, we compare the performance of the models for three additional datasets used in the image classification task. The experimental settings for FNO-NODE, AFNO-NODE, and BFNO-NODE are the same as those in section 5.1. However, AFNO-NODE has too long training time in CIFAR-100. Therefore, we contrast the performance of small-sized models (# of parameters of FNO: 310807, AFNO: 311015, BFNO: 312503).