Proceedings of Machine Learning Research vol 107:431–450, 2020

Butterfly-Net2: Simplified Butterfly-Net and Fourier Transform Initialization

Zhongshu Xu XZS1600@MAIL.USTC.EDU.CN
University of Science and Technology of China

Yingzhou Li YINGZHOU.LI@DUKE.EDU
Duke University

Xiuyuan Cheng XIUYUAN.CHENG@DUKE.EDU
Duke University

Abstract

Structured CNN designed using the prior information of problems potentially improves efficiency over conventional CNNs in various tasks in solving PDEs and inverse problems in signal processing. This paper introduces BNet2, a simplified Butterfly-Net and inline with the conventional CNN. Moreover, a Fourier transform initialization is proposed for both BNet2 and CNN with guaranteed approximation power to represent the Fourier transform operator. Experimentally, BNet2 and the Fourier transform initialization strategy are tested on various tasks, including approximating Fourier transform operator, end-to-end solvers of linear and nonlinear PDEs, and denoising and deblurring of 1D signals. On all tasks, under the same initialization, BNet2 achieves similar accuracy as CNN but has fewer parameters. And Fourier transform initialized BNet2 and CNN consistently improve the training and testing accuracy over the randomly initialized CNN.

Keywords: Convolutional neural network, Butterfly Algorithm, signal processing, inverse problem

1. Introduction

Deep convolutional neural network (CNN) has been widely applied to solving PDEs as well as inverse problems in signal processing. In both applications, spectral methods, namely involving forward and backward Fourier transform operators, serve as a traditional solution. Spectral methods have been a classical tool for solving elliptic PDEs. For image inverse problems, primarily image restoration like denoising and deblurring, a large class of PDE methods consider the nonlinear diffusion process (Perona and Malik, 1990; Tsiotsios and Petrou, 2013) which are connected to wavelet frame methods (Cai et al., 2012; Dong et al., 2017). The involved operator is elliptic, typically the Laplace operator.

Apart from the rich prior information in these problems, the conventional end-to-end deep CNN, like U-Net (Ronneberger et al., 2015) and Pix2pix (Isola et al., 2017), consists of convolutional layers which have fully trainable local filters and are densely connected across channels. The enlarged model capacity and flexibility of CNN improves the performance in many end-to-end tasks, however, such fully data-driven models may give a superior performance on one set of training and testing datasets, but encounter difficulty when transfer to another dataset, essentially due to the overfitting of the trained model which has a large amount of flexibility (Plotz and Roth, 2017; Abdelhamed et al., 2018). Also, as indicated in Chollet (2017); Jin et al. (2015); Mamalet and Garcia

(2012); Wang et al. (2017, 2018), the dense channel connection can be much pruned in the post-training process without loss of the prediction accuracy.

This motivates the design of structured CNNs which balance between model capacity and preventing over-fitting, by incorporating prior knowledge of PDEs and signal inverse problems into the deep CNN models. The superiority of structured CNNs over ordinary CNNs has been shown both for PDE solvers (Fan et al., 2019c) and for image inverse problems (Gilton et al., 2020). Several works have borrowed ideas from numerical analysis in deep models: Li et al. (2019) introduces Butterfly-Net (BNet) based on the butterfly algorithm for fast computation of Fourier integral operators (Ying, 2009; Candès et al., 2009; Demanet et al., 2012; Li et al., 2015a,b); Khoo and Ying (2019) proposes to use a switching layer with sparse connections in a shallow neural network, also inspired by the butterfly algorithm, to solve wave equation based inverse scattering problems; Fan et al. (2019c) and Fan et al. (2019b) introduce hierarchical matrix into deep network to compute nonlinear PDE solutions; Fan et al. (2019a) proposes a neural network based on the nonstandard form (Beylkin et al., 1991) and applies to approximate nonlinear maps in PDE computation. Problem-prior informed structured CNNs have become an emerging tool for a large class of end-to-end tasks in solving PDEs and inverse problems.

This paper introduces a new Butterfly network architecture, which we call BNet2. A main observation is that, as long as the Fourier transform operator is concerned, the switch layer in Butterfly algorithm can be removed while preserving the approximation ability of BNet. This leads to the proposed model, which inherits the approximation guarantee the same as the BNet, but also makes the network architecture much simplified and inline with the conventional CNN.

We also investigate the Fourier transform (FT) initialization. FT initialization adopts the interpolative construction in Butterfly factorization as in Li and Yang (2017) to initialize BNet2. Since BNet2 now is a conventional CNN with sparsified channel connections, FT initialization can also be applied to CNN to realize a linear FT. We experimentally find that both BNet2 and CNN are sensitive to initialization in problems that we test on, and FT initialized networks outperform their randomly initialized counterpart in our settings. The trained network from FT initialization also demonstrates better stability with respect to statistical transfer of testing dataset from the training set.

In summary, the contributions of the paper include: (1) We introduce BNet2, a simplified structured CNN based on Butterfly algorithm, which removes the switch layer and later layers in BNet and thus is inline with the conventional CNN architecture; (2) FT initialization for both BNet2 and CNN serves as an initialization recipe for a large class of CNNs in many applications; (3) FT initialized BNet2 and CNN inherit the theoretical exponential convergence of BNet in approximating FT operator, and the approximation can be further improved after training on data; (4) Applications to end-to-end solver for linear and nonlinear PDEs, and inverse problems of signal processing are numerically tested, and under the same initialization BNet2 with fewer parameters achieves similar accuracy as CNN; (5) FT initialized BNet2 and CNN outperforms randomly initialized CNN on all tasks included in this paper.

2. Butterfly-Net2

The structure of BNet2 inherits a part of design in BNet but makes it more simple and similar to CNN. The specific difference between BNet and BNet2 will be presented in Remark 2. For completeness, we will first recall the CNN under our own notations and then introduce BNet2.

Towards the end of this section, the numbers of parameters for both CNN and BNet2 are derived and compared.

Before introducing network structures, we first familiarize ourselves with notations used throughout this paper. The bracket notation of n denotes the set of nonnegative integers smaller than n, i.e., $[n] = \{0, 1, \ldots, n-1\}$. Further, a contiguous subset of $[n]$ is denoted as $[n]_i^k$, where k is a divisor of n denoting the total number of equal partitions and i indexed from zero denotes the i-th partition, i.e., $[n]_i^k = \{\frac{n}{k}i, \frac{n}{k}i+1, \ldots, \frac{n}{k}(i+1)-1\}$. While describing the network structure, X and Y denote the input and output vector with length N and K respectively, i.e., $X = X\big([N]\big)$ and $Y = Y\big([K]\big)$. Z, W, and B denote hidden variables, multiplicative weights and biases respectively. For example, $Z^{(\ell)}\big([n], [C]\big)$ is the hidden variable at ℓ-th layer with n spacial degrees of freedom (DOFs) and C channels; $W^{(\ell)}\big([w], [C_{in}], [C_{out}]\big)$ is the multiplicative weights at ℓ-th layer with w being the kernel size, C_{in} and C_{out} being the in- and out-channel sizes; $B^{(\ell)}\big([C_{out}]\big)$ denotes the bias at ℓ-th layer acting on C_{out} channels. Activation function is denoted as $\sigma(\cdot)$, which is ReLU in this paper by default.

2.1. CNN Revisit

A one dimensional CNN can be precisely described using notations defined as above. For a L layer CNN, we define the feedforward network as follows.

- **Layer 0:** The first layer hidden variable $Z^{(1)}\big([\frac{N}{2w}], [2r]\big)$ with $2r$ channels is generated via applying a 1D convolutional layer with kernel size $2w$ and stride $2w$ followed by an activation function on the input vector $X\big([N]\big)$, i.e.,

$$Z^{(1)}\big(j, c\big) = \sigma\Big(B^{(0)}\big(c\big) + \sum_{i \in [2w]} W^{(0)}\big(i, 0, c\big) X\big(2wj + i\big)\Big), \tag{1}$$

  for $j \in [\frac{N}{2w}]$ and $c \in [2r]$.

- **Layer $\ell = 1, 2, \ldots, L-1$:** The connection between the ℓ-th layer and the $(\ell+1)$-th layer hidden variables is a 1D convolutional layer with kernel size 2, stride size 2, $2^\ell r$ in-channels and $2^{\ell+1} r$ out-channels followed by an activation function, i.e.,

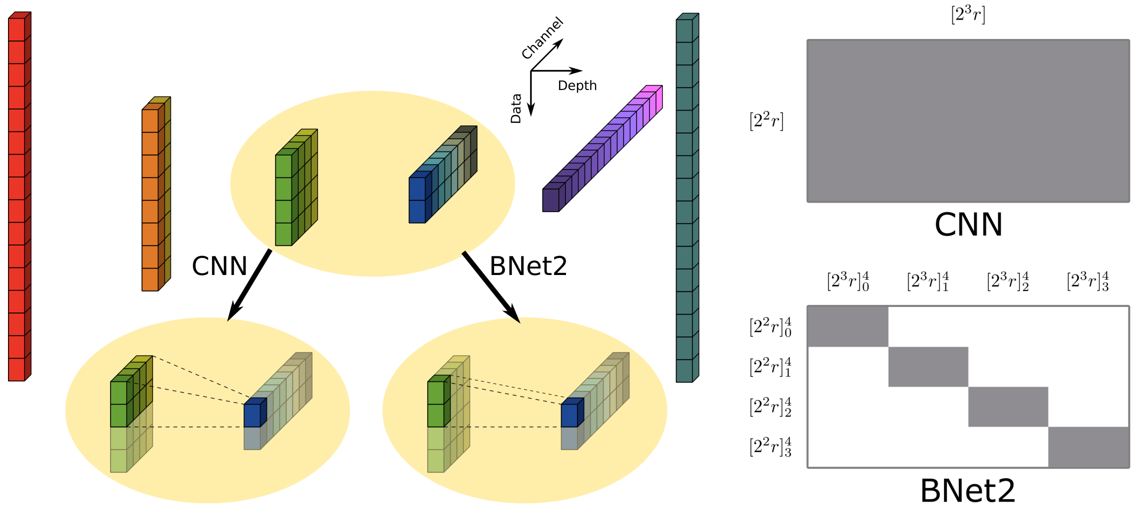
$$Z^{(\ell+1)}\big(j, c\big) = \sigma\Big(B^{(\ell)}\big(c\big) + \sum_{k \in [2^\ell r]} \sum_{i \in [2]} W^{(\ell)}\big(i, k, c\big) Z^{(\ell)}\big(2j + i, k\big)\Big), \tag{2}$$

  for $j \in [\frac{N}{2^{\ell+1} w}]$ and $c \in [2^{\ell+1} r]$. The first and second summation in (2) denotes the summation over in-channels and the spacial convolution respectively.

- **Layer L:** The last layer mainly serves as a reshaping from the channel direction to spacial direction, which links the L-th layer hidden variables with the output Y via a fully connected layer, i.e.,

$$Y\big(c\big) = \sum_{k \in [2^L r]} \sum_{i \in [\frac{N}{2^L w}]} W^{(L)}\big(i, k, c\big) Z^{(L)}\big(i, k\big), \tag{3}$$

  for $c \in [K]$. If Y is not the final output, then the bias and activation function can be added.



(*a*) Network Structure (*b*) Channel Connectivity

Figure 1: The comparison between CNN and BNet2.

In the above description, readers who are familiar with CNN may find a few irregular places. We will address these irregular places in Remark 1 after the introduction of BNet2.

CNN is the most successful neural network in practice, especially in the area of signal processing and image processing. Convolutional structure, without doubt, contributes most to this success. Another contributor is the increasing channel numbers. In practice, people usually double the channel numbers until reaching a fixed number and then stick to it till the end. Continually doubling channel numbers usually improves the performance of the CNN, but has two drawbacks. First, large channel numbers lead to the large parameter number, which in turn leads to overfitting issue. The second drawback is the expensive computational cost in both training and evaluation.

2.2. Butterfly-Net2

BNet2, in contrast to CNN, has the identical convolutional structure and allows continually doubling the channel numbers. For the two drawbacks mentioned above, BNet2 overcomes the second one and partially overcomes the first one.

The Layer 0 in BNet2 is identical to that in CNN. Hence, we only define other layers in the feed forward network as follows.

- **Layer $\ell = 1, 2, \ldots, L-1$:** The $2^\ell r$ in-channels are equally partitioned into 2^ℓ parts. For each part, a 1D convolutional layer with kernel size 2, stride 2, in-channel size r and out-channel size $2r$ is applied. The connection between the ℓ-th layer and the $(\ell+1)$-th layer hidden variables obeys,

$$Z^{(\ell+1)}(j,c) = \sigma\Big(B^{(\ell)}(c) + \sum_{k\in[2^\ell r]_p^{2^\ell}} \sum_{i\in[2]} W^{(\ell)}(i,k,c) Z^{(\ell)}(2j+i,k)\Big), \tag{4}$$

for $j \in [\frac{N}{2^{\ell+1}w}]$, $c \in [2^{\ell+1} r]_p^{2^\ell}$, and $p \in [2^\ell]$.

- **Layer L:** The $2^L r$ in-channels are equally partitioned into 2^L parts. For each part, a 1D convolutional layer with kernel size $\frac{N}{2^L w}$, in-channel size r and out-channel size $\frac{K}{2^L}$ is applied. The last layer links the L-th layer hidden variables with the output Y, i.e.,
$$Y(c) = \sum_{k \in [2^L r]_p^{2^L}} \sum_{i \in [\frac{N}{2^L w}]} W^{(L)}(i,k,c) Z^{(L)}(i,k), \tag{5}$$
for $c \in [K]_p^{2^L}$ and $p \in [2^L]$. If Y is not used as the final output directly, then the bias term and activation function can be added.

Remark 1 *This remark addresses two irregular places in the CNN and BNet2 described above against the conventional CNN. First, all convolutions are performed in a non-overlapping way, i.e., the kernel size equals the stride size. Regular convolutional layer with a pooling layer can be adopted to replace the non-overlapping convolutional layer in both CNN and BNet2. Second, except the Layer 0 and Layer L, all kernel sizes are* 2, *which can be generalized to other constant for both CNN and BNet2. We adopt such presentations to simplify the notations in Section 3, the FT initialization.*

Remark 2 *This remark addresses the difference between BNet and BNet2. As seen in Section 2.2 and Appendix A, BNet2 deletes Switch Layer and Conv-T Layers in BNet and expands Conv Layers to all layers. So the Switch Layer and Layer L in BNet are actually combined to be the Layer L in BNet2. These difference make the structure of BNet2 simpler than BNet. On the other hand, the removal of Switch Layer and Conv-T Layers makes BNet2 directly a sparsified regular CNN, while BNet does not has such a property.*

In (4), the in-channel index k and the out-channel index c of BNet2 are linked through the auxiliary index p, whereas in the CNN, the in-channel index k and the out-channel index c are independent (see (2)). Figure 1 (b) illustrates the connectivity of k and c in CNN and in BNet2 at the 2-nd layer. Further, Figure 1 (a) shows the overall structure of CNN and BNet2. If we fill part of the multiplicative weights in CNN to be that in BNet2 according to (4) and set the rest multiplicative weights to be zero, then CNN recovers BNet2. Hence, any BNet2 can be represented by a CNN. The approximation power of CNN is guaranteed to exceed that of BNet2. Surprisingly, according to our numerical experiments, the extra approximation power does not improve the training and testing accuracy much in all examples we have tested.

2.3. Parameter Counts

Parameter counts are explicit for both CNN and BNet2. The numbers of bias are identical for two networks. It is $2r$ for Layer 0, $2^{\ell+1}r$ for Layer ℓ and 0 for Layer L. Hence the overall number of biases is
$$\sharp\{\text{bias}\} = \sum_{\ell \in [L]} 2^{\ell+1} r = (2^{L+2} - 2) r. \tag{6}$$

The total number of multiplicative parameters are very different for CNN and BNet2. For CNN, the parameter count is $2r \cdot 2w$ for Layer 0, $2^\ell r \cdot 2^{\ell+1} r \cdot 2$ for Layer ℓ, and $2^L r \cdot K \cdot \frac{N}{2^L w}$ for Layer L. Hence the overall number of multiplicative parameters for CNN is
$$\sharp\{W_{CNN}\} = 4rw + \frac{rNK}{2^L w} + \sum_{\ell=1}^{L-1} 2^{2\ell+2} r^2 = 4rw + \frac{rNK}{w} + \frac{2^{2L+2} - 2^4}{3} r^2. \tag{7}$$

While, for BNet2, the parameter count is $2r \cdot 2w$ for Layer 0, $2^\ell r \cdot 2r \cdot 2$ for Layer ℓ, and $2^L r \cdot \frac{K}{2^L} \cdot \frac{N}{2^L w}$ for Layer L. Hence the overall number of multiplicative parameters for BNet2 is

$$\sharp\{W_{BNet2}\} = 4rw + \frac{rNK}{2^L w} + \sum_{\ell=1}^{L-1} 2^{\ell+2} r^2 = 4rw + \frac{rNK}{2^L w} + \frac{2^{L+2} - 2^3}{3} r^2. \tag{8}$$

If we assume $N \sim K \sim 2^L$ and $r \sim w \sim 1$, which corresponds to doubling channel number till the end, then we have

$$\sharp\{\text{parameter in CNN}\} = O(N^2) \quad \text{and} \quad \sharp\{\text{parameter in BNet2}\} = O(N). \tag{9}$$

Let us consider another regime, i.e., $K \sim 2^L$ and $r \sim \frac{N}{2^L w} \sim 1$, which can be viewed as an analog of doubling the channel number first and then being fixed to a constant 2^L. Under this regime, the total numbers of parameters can be compared as,

$$\sharp\{\text{parameter in CNN}\} = O(\frac{N}{K} + K^2) \quad \text{and} \quad \sharp\{\text{parameter in BNet2}\} = O(\frac{N}{K} + K), \tag{10}$$

where both $\frac{N}{K}$ come from $4rw$ term. Hence, in both regimes of hyper parameter settings, BNet2 has lower order number of parameters comparing against CNN. If the performance in terms of training and testing accuracy remains similar, BNet2 is then much more preferred than the CNN.

3. Fourier Transform Initialization

A good initialization is crucial in training CNNs especially in training highly structured neural networks like BNet2. It is known that CNN with random initialization achieves remarkable results in practical image processing tasks as shown in Krizhevsky et al. (2012). However, for synthetic signal data as in Section 4, in which the high accuracy prediction is possible through a CNN with a set of parameters, we notice that CNN with random initialization and ADAM stochastic gradient descent optimizer is not able to converge to that CNN.

In this section, we aim to initialize both BNet2 and CNN to fulfill the discrete FT operator, which is defined as

$$\mathcal{K}(\xi, t) = e^{-2\pi\imath \xi \cdot t}, \tag{11}$$

for $\xi \in [K]$ and $t \in \frac{[N]}{N}$, where N denotes the number of discretization points and K denotes the frequency window size. When the network is initialized as an approximated discrete FT, we call it **FT initialization**. Discrete FT is the traditional computational tool for signal processing and image processing. Almost all related traditional algorithms involve either FT directly or Laplace operator, which can be realized via two FTs, see Buades et al. (2005); Chan and Shen (2005). Hence, if we can initialize a neural network as such a traditional algorithm involving discrete FT, the training of the neural network would be viewed as refining the traditional algorithm and makes its data adaptive. In another word, neural network solving image processing and signal processing tasks can be guaranteed to outperform traditional algorithms, although it is widely accepted in practice. This section is composed of two parts: preliminary and initialization. We will first introduce related complex number neural network realization, Chebyshev interpolation, FT approximation in the preliminary part. The receipt of the FT initialization for both BNet2 and CNN is then introduced in detail, mainly in (16), (17), and (18).

3.1. Preliminary

Fourier transform is a linear operator with complex coefficients. The realization of complex number operations via ReLU neural network is detailed in Appendix B together with the definition of the extensive assign operator $\overset{\diamond}{=}$.

An important tool is the Lagrange polynomial on Chebyshev points. The Chebyshev points of order r on $[-\frac{1}{2}, \frac{1}{2}]$ is defined as,

$$\left\{ z_i = \frac{1}{2}\cos\left(\frac{i\pi}{r}\right) \right\}_{i\in[r]}. \tag{12}$$

The associated Lagrange polynomial at z_k is

$$\mathcal{L}_k(x) = \prod_{p\neq k} \frac{x - z_p}{z_k - z_p}. \tag{13}$$

If the interval $[-\frac{1}{2}, \frac{1}{2}]$ is re-centered at c and scaled by w, then the transformed Chebyshev points obeys $z_i' = wz_i + c$ and the corresponding Lagrange polynomial at z_k' is

$$\mathcal{L}_k'(x) = \prod_{p\neq k} \frac{x - z_p'}{z_k' - z_p'} = \prod_{p\neq k} \frac{\frac{x-z_k'}{w} + z_k - z_p}{z_k - z_p} = \widetilde{\mathcal{L}}_k(\frac{x - z_k'}{w}), \tag{14}$$

where $\widetilde{\mathcal{L}}_k(\cdot)$ is independent of the transformation of the interval.

Recall the Chebyshev interpolation representation of FT as Theorem 2.1 in Li et al. (2019). We include part of that theorem with a small modification here with our notation $\widetilde{\mathcal{L}}_k$ for completeness.

Proposition 3 (Theorem 2.1 in Li et al. (2019)) *Let L and r be two parameters such that $\pi e K \leqslant r2^L$. For any $\ell \in [L]$, let $A^{\ell+1}$ and $B^{L-\ell-1}$ denote two connected subdomains of $[0, K)$ and $[0, 1)$ with length $K \cdot 2^{-\ell-1}$ and $2^{\ell+1-L}$ respectively. Then for any $\xi \in A^{\ell+1}$ and $t \in B^{L-\ell-1}$, there exists a Chebyshev interpolation representation of the FT operator,*

$$\left| e^{-2\pi\imath\xi\cdot t} - \sum_{k=1}^{r} e^{-2\pi\imath\xi\cdot t_k} e^{-2\pi\imath\xi_0\cdot(t-t_k)} \widetilde{\mathcal{L}}_k\left(\frac{t-t_k}{2^{L-\ell-1}}\right) \right| \leqslant \left(2 + \frac{2}{\pi}\ln r\right)\left(\frac{\pi e K}{r2^{L+1}}\right)^r, \tag{15}$$

where ξ_0 is the centers of $A^{\ell+1}$, $\{t_k\}_{k\in[r]}$ are the Chebyshev points on $B^{L-\ell-1}$.

Obviously, part of the approximation, $e^{-2\pi\imath\xi_0\cdot(t-t_k)}\widetilde{\mathcal{L}}_k\left(\frac{t-t_k}{2^{L-\ell-1}}\right)$, admits the convolutional structure across all $B^{L-\ell-1}$. This part will be called the **interpolation part** in the following. It is the key that we can initialize CNN and BNet2 as a FT.

3.2. Fourier Transform Initialization for CNN and BNet2

Since all weights fit perfectly into the structure of BNet2, we will only introduce the initialization of BNet2 in detail. Assume the input is a function discretized on a uniform grid of $[0, 1)$ with N points and the output is the discrete FT of the input at frequency $[K]$. Throughout all layers, the bias terms are initialized with zero. In the description below, we focus on the initialization of the multiplicative weights. Without loss of generality, we further assume $N = w2^L$.

- **Layer 0:** For $\ell = 0$, we consider $A_i^1 = [K]_i^2$ and $B_j^{L-1} = \frac{[N]_j^{2^{L-1}}}{N}$ for $i \in [2]$ and $j \in [2^{L-1}]$, which satisfies the condition in Proposition 3. An index pair (i, k) for i being the index of A_i^1 and k being the index of the Chebyshev points can be reindexed as $[2r]$. Hence we abuse the channel index c as $c = (i, k)$. Then fixing c, the interpolation part is the same for all B_j^{L-1}, which is naturally a non-overlapping convolution. Hence we set

$$W^{(0)}\big(Nt, 0, c\big) \overset{\diamond}{=} e^{-2\pi\imath\xi_0\cdot(t-t_k)}\widetilde{\mathcal{L}}_k\Big(\frac{t-t_k}{2^{L-1}}\Big) \tag{16}$$

for $t \in \frac{[\frac{N}{2^{L-1}}]}{N}$, $c = (i, k)$, and t_k is the Chebyshev point on B_0^L or B_1^L. Then after applying the 1D convolutional layer as (1), the first hidden variable $Z^{(1)}\big(j, c\big)$ represents the input vector interpolated to the Chebyshev points t_k on B_j^{L-1} with respect to A_i^1. The following layers recursively apply Proposition 3 to the remaining $e^{-2\pi\imath\xi\cdot t_k}$ part.

- **Layer $\ell = 1, 2, \ldots, L-1$:** We concern $A_i^{\ell+1}$ and $B_j^{L-\ell-1}$ for $i \in [2^{\ell+1}]$ and $j \in [2^{L-\ell-1}]$ at the current layer. The hidden variable $Z^{(\ell)}\big(j', c'\big)$ represents the input interpolated to the Chebyshev points on $B_{j'}^{L-\ell}$ with respect to $A_{i'}^{\ell}$, where $c' = (i', k')$ and k' is the index of Chebyshev points. $A_i^{\ell+1}$ is a subinterval of $A_{\lfloor \frac{i}{2} \rfloor}^{\ell}$ and $B_j^{L-\ell-1}$ covers $B_{2j}^{L-\ell}$ and $B_{2j+1}^{L-\ell}$. For a fixed $c = (i, k)$, the interpolation part is the same for each index j. The convolution kernel, hence, is defined as,

$$W^{(\ell)}\big(p, c', c\big) \overset{\diamond}{=} e^{-2\pi\imath\xi_0\cdot(t_{k'}-t_k)}\widetilde{\mathcal{L}}_k\Big(\frac{t_{k'}-t_k}{2^{L-\ell-1}}\Big) \tag{17}$$

where $c' = (\lfloor \frac{i}{2} \rfloor, k')$, $c = (i, k)$, $t_{k'}$ and t_k are Chebyshev points on $B_{2j+p}^{L-\ell}$ and $B_j^{L-\ell-1}$ respectively, and $p \in [2]$.

- **Layer L:** This layer concerns A_i^L and B_0^0 for $i \in [2^L]$. All previous layers take care of the interpolation part. And the current layer applies the FT operator on each A_i^L. The hidden variable $Z^{(L)}\big(j, c'\big)$ represents the input interpolated to the Chebyshev points on B_0^L with respect to A_i^{ℓ}, where $c' = (i, k')$ and k' is the index of Chebyshev points. Define the channel index c as an index pair (i, k), where $i \in [2^L]$ is the index of A_i^L and $k \in [\frac{K}{2^L}]$ is the index for uniform points $\xi_k \in A_i^L$. Then the multiplicative weights are initialized as,

$$W^{(L)}\big(0, c', c\big) \overset{\diamond}{=} e^{-2\pi\imath\xi_k\cdot t_{k'}}. \tag{18}$$

Since BNet2 can be viewed as a CNN with many zero weights, such an initialization can be used to initialize CNN as well. When we set the weights as above and set the rest weights to be zero, the CNN is then initialized by the FT initialization.

Remark 4 *As mentioned in Remark 1, a few irregular places in the current CNN and BNet2 description can be modified to match conventional CNN. The FT initialization can be updated accordingly. First, when convolutions are performed in a non-overlapping way without pooling layer, we can enlarge the kernel size and embed zeros to eliminate the impact of the overlapping part. Second, when the kernel sizes are a constant different from 2, the generalization of the initialization is feasible as long as the bipartition is modified to a multi-partition.*

The approximation power of FT initialized CNN and BNet2 can be analyzed in an analogy way as that in Li et al. (2019) and the sketch proof in Appendix C.

Theorem 5 *Let N and K denote the size of the input and output respectively. The depth L and channel parameter r satisfies $\pi e K \leqslant r 2^L$. Then there exists a BNet2/CNN, $\mathcal{B}(\cdot)$, approximating the discrete FT operator such that for any bounded input vector f, the error satisfies,*

$$\|\mathcal{K}f - \mathcal{B}(f)\|_p \leqslant C_{r,K} \left(\frac{r^{1-\frac{1}{p}} \left(\frac{2}{\pi} \ln r + 1\right)}{2^{r-2}} \right)^L \|f\|_p, \tag{19}$$

where $C_{r,K} = {(2+4/\pi \ln r)^3 (\pi e K)^r}/{(2r)^{r-1}}$ is a constant depending only on r and K, for $p \in [1, \infty]$.

Theorem 5 is validated numerically in next section. In terms of function approximation, Li et al. (2019) showed that for BNet a result in the type of (19) implies that a large class of functions can be well approximated with network complexity depending on the effective frequency bandwidth instead of the input dimension. Based on Theorem 5, such a function approximation result applies to BNet2 and CNN as well. The approximation analysis can be extended to vector-valued output functions. Numerically, we apply BNet2 to the computation of the Laplace operator energy, which has scalar output (Section 4.2.1) , as well as end-to-end solvers of PDEs (Section 4.2.3, 4.2.3) and signal processing inverse problems (Section 4.3), both of which have vector-valued output.

4. Numerical Results

This section presents numerical experiments to demonstrate the approximation power of CNN and BNet2, and compare the difference between FT initialization and random initialization. Thus, four different settings, CNN with random initialization (CNN-rand), CNN with FT initialization (CNN-FT)[1], BNet2 with random initialization (BNet2-rand), and BNet2 with FT initialization (BNet2-FT) are tested on three different sets of problems: (1) approximation of FT operator; (2) energy and solution maps of elliptic equations; (3) 1D signals de-blurring and de-noising tasks.

4.1. Approximation of Fourier Transform Operator

This section repeats experiments as in the original BNet Li et al. (2019) on BNet2, namely approximation power before training, approximation power after training, and transfer learning capability.

4.1.1. Approximation Power Before Training

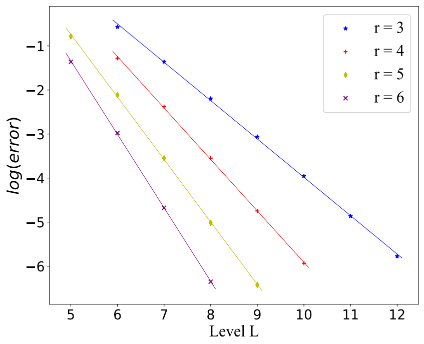
The first experiment aims to validate the exponential decay of the approximation error of the BNet2 as either the depth L increases or the number of Chebyshev points r increases. We construct and initialize a BNet2 to approximate a discrete FT operator, which has length of input $N = 16,384$ and length of output K representing integer frequency on $[0, K)$. The approximation power is measured under the relative operator p-norm, i.e., $\epsilon_p = {\|\mathcal{K}-\mathcal{B}\|_p}/{\|\mathcal{K}\|_p}$, where $\mathcal{B}$ and $\mathcal{K}$ denote BNet2 and FT operator respectively.

In Table 1, we fix the number of Chebyshev points being $r = 4$ and varying L for two choices of K. All errors with respect to different norms decay exponentially as L increases. The decay rates for different Ks remain similar, while the prefactor is slightly larger for larger K.

1. The Layer L is often combined with feature layers. Hence for both CNN, Layer L as in BNet2 is adopted.

N	$K = 64$				$K = 256$			
	L	ϵ_1	ϵ_2	ϵ_∞	L	ϵ_1	ϵ_2	ϵ_∞
	6	3.48e−02	5.25e−02	6.30e−02	8	3.80e−02	7.26e−02	6.94e−02
	7	2.18e−03	4.18e−03	6.36e−03	9	2.39e−03	6.05e−03	6.95e−03
16384	8	1.37e−04	2.84e−04	5.30e−04	10	1.54e−04	4.31e−04	5.73e−04
	9	8.96e−06	1.79e−05	4.08e−05	11	1.05e−05	2.89e−05	4.37e−05
	10	6.41e−07	1.16e−06	3.11e−06	12	7.64e−07	1.86e−06	3.30e−06

Table 1: Relative error of BNet2 before training with $r = 4$ in approximating FT operator.



r	$K = 64$			$K = 128$			$K = 256$		
	k_1	k_2	k_∞	k_1	k_2	k_∞	k_1	k_2	k_∞
3	-0.90	-0.87	-0.82	-0.90	-0.85	-0.82	-0.91	-0.85	-0.82
4	-1.18	-1.14	-1.04	-1.18	-1.16	-1.16	-1.17	-1.15	-1.08
5	-1.44	-1.42	-1.34	-1.48	-1.43	-1.39	-1.44	-1.40	-1.36
6	-1.72	-1.67	-1.60	-1.72	-1.69	-1.60	-1.72	-1.70	-1.61

Figure 2: (Left plot) Exponential convergence rate when $K = 64$, $p = 2$. (Right table) Convergence rate of BNet2 before training in approximating FT operator for rs. k_1, k_2, and k_∞ are the logarithms of convergence rates under different norms, $N = 16384$.

In the table in Figure 2, we calculate the logarithms of rates of convergence for different rs and Ks under different norms. The table shows that for all choices of K the convergence rates measured under different norms stay similar for any fixed r. And the convergence rate decreases as r increases.

All of these above convergence behaviors agree with the analysis in this paper and Li et al. (2019). And all rates we obtained are better than the corresponding theoretical ones. In summary, when approximating FT operator using FT initialized BNet2, the approximation accuracy decays exponentially as L increases and the rate of convergence decreases as r increases.

4.1.2. Approximation Power After Training

The second numerical experiment aims to demonstrate the approximation power of the four networks in approximating FT operator after training.

Each data point used in this section is generated as follows. We first generate an array of K random complex numbers with each component being uniformly random in $[-a, a]$. The zero frequency is a random real number. Second, we apply a Gaussian mask with width (standard deviation) $G_{\text{width}} = 2$ and center $G_{\text{center}} = 0$(low frequency data) or $G_{\text{center}} = 56$(high frequency data) on the array. The array then is complexly symmetrized to be a frequency vector and the inverse discrete FT is applied to obtain the real input vector. The constant a is chosen such that the two norm of the output vector is close to 1. Examples of low and high frequency input can be seen in Appendix E.

In this experiment, we have input length being $N = 128$, output length being $K = 8$, level number being $L = 5$, channel parameter being $r = 3$. All networks are trained under the infinity data setting, i.e., the training data is randomly generated on the fly. ADAM optimizer with batch size 256 and exponential decay learning rate is adopted. For FT initialized networks, the maximum training steps is 10,000, whereas for random initialization we train 20,000 steps. The reported relative error in vector two norm is calculated on a testing data set of size $16,384$ with the same distribution as the training data set. Default values are used for any unspecified hyper parameters.

Network	♯ Parameters	Low Frequency		High Frequency	
		Pre-Train Rel Err	Test Rel Err	Pre-Train Rel Err	Test Rel Err
BNet2-FT	9252	2.44e−3	1.33e−5	2.61e−3	1.29e−5
BNet2-rand		1.38e+0	8.82e−3	1.25e+0	8.50e−3
CNN-FT	49572	2.44e−3	9.29e−6	2.61e−3	7.54e−6
CNN-rand		5.09e+0	4.63e−2	2.73e+0	2.20e−2
CNN-BNet2(FT-trained)		1.33e−5	6.18e−6	1.29e−5	4.06e−6

Table 2: Relative errors of networks in approximating FT operator before and after training. The last row use the trained parameters in the first row as it's initialization.

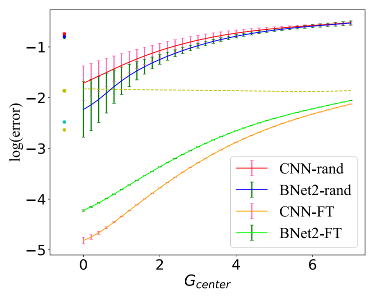
Table 2 shows the pre-training and testing relative errors for BNet2-FT, BNet2-rand, CNN-FT, CNN-rand and CNN-BNet2(FT-trained) on both low and high frequency training set. Comparing the results, every network have similar performance on both data set, BNet2 and CNN have similar performance for both initializations, while BNet2 has only about $1/5$ parameters as CNN. Hence those extra coefficients in CNN do not improve the approximation to FT operator. On the other hand, FT initialization achieves an accuracy better than that of BNet2-rand and CNN-rand after training. After training FT initialized networks, extra two digits accuracy is achieved for both BNet2-FT and CNN-FT. The CNN with trained FT initialized BNet2 performs slightly better than CNN-FT, but the improvement is not as significant as FT initialization. We conjecture that the local minima found through the training from the FT initialization has a narrow and deep well on the energy landscape such that the random initialization with stochastic gradient descent is not able to find it efficiently.

4.1.3. Transfer Learning Capability

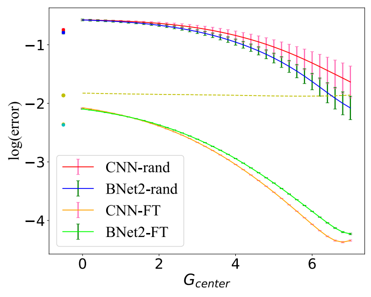
This numerical experiment compares the transfer learning capability of four networks. The training and testing data are generated in a same way as in Section 4.1.2 with different choices of G_{center} and G_{width}. We have three training sets: low frequency training set ($G_{\text{center}} = 0$ and $G_{\text{width}} = 2$), high frequency training set ($G_{\text{center}} = 7$ and $G_{\text{width}} = 2$) and mixture training set (no Gaussian mask). A sequence of testing sets of size $16,384$ are generated with $G_{\text{center}} = 0, 0.2, 0.4, \dots, 7$ and $G_{\text{width}} = 2$.

The networks used here have the same structure and hyper-parameters as in Sec 4.1.2 while the channel parameter $r = 2$ instead of 3 here. Each experiment is repeated 20 times. Then the mean and standard deviation of the error in two norm are reported below.

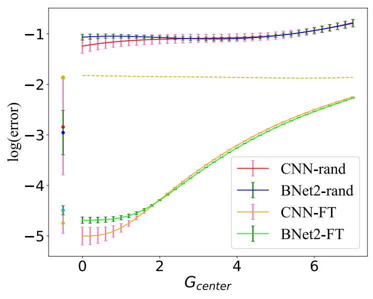
As in Figure 3, for both initializations, BNet2 and CNN have similar accuracy especially on testing sets away from the training set. Taking the FT initialization before training as a reference, we also notice that even if randomly initialized networks can reach the accuracy of the reference on some testing sets, they lose accuracy on transferred testing sets. On the other side, FT initialized networks after training maintain accuracy better than that of the reference on all testing sets. In terms of the stability after training, BNet2-FT and CNN-FT are much more stable than BNet2-rand



(*a*) low frequency training set



(*b*) high frequency training set



(*c*) mixture training set

Figure 3: Figures show the transfer learning results of four networks trained on three different training sets. The horizontal axis represents testing sets with different G_{center}. The testing results on the mixture testing set are plotted as the isolated error bars at the left of each plot. Each error bar represents the mean and standard deviation across 20 repeating experiments. The horizontal dash lines indicate the testing error of FT initialized networks before training.

and CNN-rand, which is due to the randomness in initializers. This phenomenon also emphasizes the advantage of FT initialization in stability and repeatability.

4.2. Energy and Solution Map of Elliptic PDEs

This section focus on the elliptic PDE of the following form,

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(a(x)\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right) + bu^3(x) = f(x), \quad x \in [0,1), \tag{20}$$

with periodic boundary condition, where $a(x) > 0$ denotes coefficients and b denotes the strength of nonlinearity. Such an equation appears in a wide range of physical models governed by Laplace's equation, Stokes equation, etc. Equation (20) is discretized on a uniform grid with N points.

4.2.1. Energy of Laplace Operator

In this section, we aim to construct an approximation of the energy functional of 1D Poisson's equations, i.e., $a(x) \equiv 1$ and $b = 0$. The energy functional of Poisson's equation is defined as the negative inner product of u and f, which can also be approximated by a quadratic form of the leading low-frequency Fourier components. Hence, Here we adopt BNet2-FT, BNet2-rand, CNN-FT, and CNN-rand with an extra square layer, which is called task-dependent layer.

In this numerical example, the input f has the same distribution as that in Section 4.1.2. All other hyper parameters of networks and the training setting are also identical to that in Section 4.1.2.

	♯ Parameters	Pre-Train Rel Err	Test Rel Err
BNet2-FT	9268	2.11e−3	8.10e−6
BNet2-rand		7.97e−1	4.62e−3
CNN-FT	49588	2.11e−3	4.79e−6
CNN-rand		5.53e−1	6.21e−3

Table 3: Training results for networks in representing the energy of the Laplace operator.

Table 3 shows the results for energy of 1D Laplace operators, which has similar property as Table 2. Hence all conclusions in Section 4.1.2 apply here.

4.2.2. End-to-end Linear Elliptic PDE Solver

In this section, we aim to represent the end-to-end solution map of linear elliptic PDEs by an encoder-decoder structure. The linear elliptic PDE is (20) with high contrast coefficient $a(x)$ as,

$$a(x_i) = \begin{cases} 10, & \lfloor \frac{8i-N}{2N} \rfloor \equiv 0 \,(\mathrm{mod}\, 2) \\ 1, & \lfloor \frac{8i-N}{2N} \rfloor \equiv 1 \,(\mathrm{mod}\, 2) \end{cases}, \tag{21}$$

for x_i being the uniform point in $[0, 1)$ and $b = 0$.

It is well known that the inverse of linear constant coefficient Laplace operator can by represented by $\mathcal{F}^\star \mathcal{D}^{-1} \mathcal{F}$ where $\mathcal{F}$ denotes the FT and $\mathcal{D}$ is a diagonal operator. Therefore, we design our network in the same sprite. Our network contains three parts: a BNet2/CNN encoder with input length $2N$,output length K_{en}, a $K_{en} \times K_{de}$ fully connected dense layer with bias terms and activation function, and a BNet2/CNN decoder with input length K_{de}, output length N. Since the input f is real function, here we apply odd symmetry to it and initialize the first BNet2/CNN as FT to make the first part of the network serves as sine transform. Then we initialize D according to $a(x)$, and initialize the third part to be an inverse sine transform so that the overall network is an approximation of the inverse operator. In this example, we set $N = 64$, $K_{en} = 8$, and $K_{de} = 16$. Both BNet2s/CNNs are constructed with $L = 4$ layers and channel parameter $r = 3$.

Each training and testing data is generated as follows. We first generate an array of length K with $K - 1$ random numbers. The first entry is fixed to be 0 to incorporate the periodic boundary condition, whereas the following $K - 1$ entries are uniform sampled from $[-1, 1]$. Then an inverse discrete sine transform is applied to obtain the input vector. The reference solution is calculated through traditional spectral methods on a finer grid of $16N$ nodes. The training and testing data set contain $4,096$ and $5,000$ points respectively. Other settings are the same as in Section 4.1.2.

	♯ Parameters	Linear PDE			Nonlinear PDE		
		Pre-Train Rel Err	Train Rel Err	Test Rel Err	Pre-Train Rel Err	Train Rel Err	Test Rel Err
BNet2-FT	17856	5.16e−2	4.71e−3	4.86e−3	3.48e+0	1.97e−2	2.02e−2
BNet2-rand		9.75e+0	4.26e−2	4.43e−2	4.37e+2	1.00e+0	1.00e+0
CNN-FT	82368	5.16e−2	3.80e−3	3.96e−3	3.48e+0	1.36e−2	1.52e−2
CNN-rand		3.53e+0	2.02e−2	2.03e−2	5.65e+2	1.00e+0	1.00e+0

Table 4: Relative errors in approximating the solution map of the linear and nonlinear elliptic PDE.

Table 4 and Figures 4 show that in this end-to-end task, CNN performs slightly better than BNet2 at the cost of 5 times more parameters. Training from FT initialization in both cases provides one more digit of accuracy. Figures 4 further shows that BNet2-FT and CNN-FT significantly outperform BNet2-rand and CNN-rand near sharp changing areas in u.

4.2.3. End-to-end Nonlinear Elliptic PDE Solver

In this section, we focus on a highly nonlinear elliptic PDE (20) with $a(x) \equiv 1$ and $b = 10^3$. The reference solution for nonlinear PDEs in general is difficult and expensive to obtain. Hence, in this

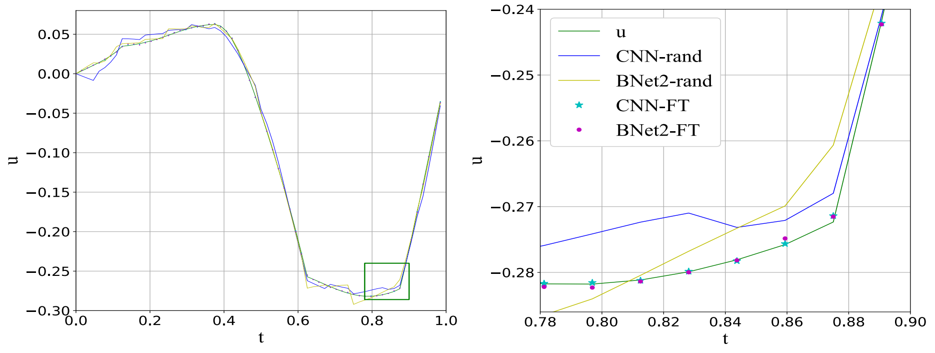


Figure 4: The left figure shows an example solution u and the output from four networks for the linear elliptic PDE. The right figure is a zoom-in of the green box in the left.

section, we apply the solve-train framework proposed in Li et al. (2020) to avoid explicitly solving the nonlinear PDE.

Denoting the nonlinear PDE as an operator acting on u, i.e., $\mathcal{A}(u) = f$, our loss function here is defined as

$$\ell\Big(\{f_i\}_{i=1}^{N_{\text{train}}}, \mathcal{A}, \mathcal{N}\Big) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \big\| f_i - \mathcal{A}\big(\mathcal{N}(f_i)\big)\big\|^2, \tag{22}$$

where $\mathcal{N}$ denotes the used neural network. The reported relative error is calculated on testing data $\{g_i\}_{i=1}^{N_{\text{test}}}$ as follows,

$$\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{\Big\| g_i - \mathcal{A}\Big(\mathcal{N}(g_i)\Big)\Big\|}{\|g_i\|}. \tag{23}$$

The same networks and other related settings as in Section 4.2.2 are used here.

Table 4 shows that under solve-train framework randomly initialized networks are not able to converge to a meaningful result, whereas FT initialized networks find a representation for the solution map with 2 digits accuracy. Partially, this is due to the extra condition number of $\mathcal{A}$ introduced by solve-train framework in training. Comparing BNet2-FT with CNN-FT, we find similar conclusions as before, i.e., CNN-FT achieves slightly better accuracy with higher cost in the number of parameters.

4.3. Denoising and Deblurring of 1D Signals

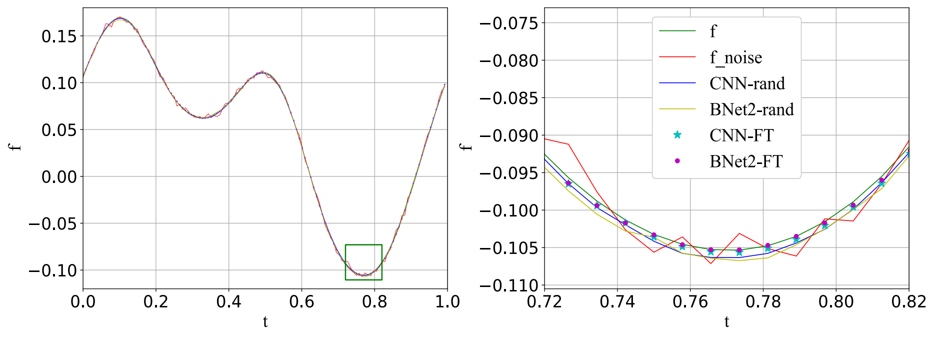
In this section, we aim to apply networks to the denoising and deblurring tasks in signal processing. An encoder-decoder structure is used in this experiment, which concatenates two networks, i.e., two BNet2-FT, two BNet2-rand, two CNN-FT or two CNN-rand. Such a structure with FT initialization reproduces a low pass filter.

The low frequency true signal f is generated as the input vector in Section 4.1.2. Two polluted signals, f_{noise} and f_{blur}, are generated by adding a Gaussian noise with standard deviation 0.002 and convolving a Gaussian with standard deviation 3, respectively. The mean relative errors of f_{noise} and f_{blur} are 0.0226 and 0.165 respectively.

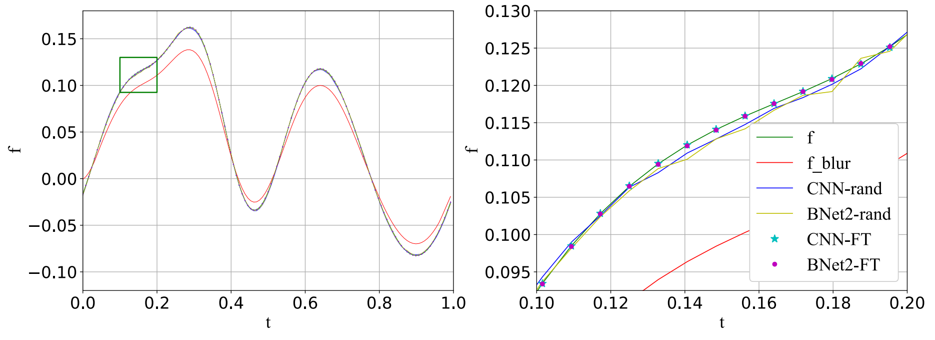
Regarding the encoder-decoder structure, the first part has input length $N = 128$ and output length $K = 8$ in representing frequency domain $[0, K)$. After that, the output of the first part is complex symmetrized to frequency domain $[-K, K)$. Then the second part has input length 16, output length 128. In both parts, we adopt $L = 4$ layers. The other hyper parameters are the same as that in Section 4.1.2. All relative errors are measured in two norm.

Network	♯ para	f_{noise}		f_{blur}	
		Pre-Train Rel Err	Test Rel Err	Pre-Train Rel Err	Test Rel Err
BNet2-FT	19,392	9.56e−2	7.54e−3	1.64e−1	8.02e−4
BNet2-rand		1.07e+0	1.52e−2	1.02e+0	1.07e−2
CNN-FT	83,904	9.56e−2	7.74e−3	1.64e−1	8.19e−4
CNN-rand		1.23e+0	1.28e−2	1.05e+0	9.95e−3

Table 5: Relative error of denoising and deblurring of 1D signals.



(*a*) Example of signal denoising



(*b*) Example of signal deblurring

Figure 5: (a) and (b) show an example of denoising and deblurring respectively. The right figures are zoom-in of green boxes in the left figures.

Table 5 lists all relative errors of four networks and Figure 5 shows the performance of four networks on an example signal. We observe that, for both tasks, the FT initialized networks have better accuracy than their randomly initialized counterparts. Under the same initialization, BNet2 achieves similar accuracy as CNN with much fewer parameters. Comparing two tasks, we notice that the improvement of FT initialization over random initialization is more significant on deblurring task than that on denoising task. For denoising tasks, as we enlarge additive noise level, BNet2-rand and CNN-rand perform almost as good as BNet2-FT and CNN-FT. However, for deblurring tasks, we always observe significant improvement from FT initialization.

Acknowledgments

The authors thank Bin Dong, Yiping Lu for discussion on image inverse problems. The work of YL is supported in part by National Science Foundation via grants DMS-1454939 and ACI-1450280. XC is partially supported by NSF (DMS-1818945, DMS-1820827), NIH (grant R01GM131642) and the Alfred P. Sloan Foundation.

References

Abdelrahman Abdelhamed, Stephen Lin, and Michael S Brown. A high-quality denoising dataset for smartphone cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1692–1700, 2018.

Gregory Beylkin, Ronald Coifman, and Vladimir Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on pure and applied mathematics*, 44(2):141–183, 1991.

Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A review of image denoising algorithms, with a new one. *Multiscale Model. Simul.*, 4(2):490–530, 2005. ISSN 15403459.

Jian-Feng Cai, Bin Dong, Stanley Osher, and Zuowei Shen. Image restoration: total variation, wavelet frames, and beyond. *Journal of the American Mathematical Society*, 25(4):1033–1089, 2012.

Emmanuel J. Candès, Laurent Demanet, and Lexing Ying. A fast butterfly algorithm for the computation of Fourier integral operators. *Multiscale Model. Simul.*, 7(4):1727–1750, jan 2009.

Tony Chan and Jianlong Shen. *Image processing and analysis variational, PDE, wavelet, and stochastic methods*. Society for Industrial and Applied Mathematics, 2005. ISBN 089871589X.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, pages 1800–1807. Institute of Electrical and Electronics Engineers Inc., nov 2017.

Laurent Demanet, Matthew Ferrara, Nicholas Maxwell, Jack Poulson, and Lexing Ying. A butterfly algorithm for synthetic aperture radar imaging. *SIAM Journal on Imaging Sciences*, 5(1):203–243, jan 2012.

Bin Dong, Qingtang Jiang, and Zuowei Shen. Image restoration: Wavelet frame shrinkage, nonlinear evolution pdes, and beyond. *Multiscale Modeling & Simulation*, 15(1):606–660, 2017.

Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. Bcr-net: A neural network based on the nonstandard wavelet form. *Journal of Computational Physics*, 384:1–15, 2019a.

Yuwei Fan, Jordi Feliu-Faba, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2): 21, 2019b.

Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019c.

D. Gilton, G. Ongie, and R. Willett. Neumann networks for linear inverse problems in imaging. *IEEE Transactions on Computational Imaging*, 6:328–343, 2020.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Flattened convolutional neural networks for feedforward acceleration, 2015. https://arxiv.org/abs/1412.5474.

Yuehaw Khoo and Lexing Ying. Switchnet: A neural network model for forward and inverse scattering problems. *SIAM Journal on Scientific Computing*, 41(5):A3182–A3201, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F Pereira, C J C Burges, L Bottou, and K Q Weinberger, editors, *Adv. Neural Inf. Process. Syst. 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Yingzhou. Li and Haizhao. Yang. Interpolative butterfly factorization. *SIAM Journal on Scientific Computing*, 39(2):A503–A531, 2017.

Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying. Butterfly factorization. *Multiscale Model. Simul.*, 13(2):714–732, jan 2015a.

Yingzhou Li, Haizhao Yang, and Lexing Ying. A multiscale butterfly algorithm for multidimensional Fourier integral operators. *Multiscale Model. Simul.*, 13(2):1–18, jan 2015b.

Yingzhou Li, Xiuyuan Cheng, and Jianfeng Lu. Butterfly-Net: Optimal function representation based on convolutional neural networks, aug 2019. http://arxiv.org/abs/1805.07451v4.

Yingzhou Li, Jianfeng Lu, and Anqi Mao. Variational training of neural network approximations of solution maps for physical models. *Journal of Computational Physics*, 409:109338, 2020. ISSN 0021-9991.

Franck Mamalet and Christophe Garcia. Simplifying ConvNets for fast learning. In *Lect. Notes Comput. Sci.*, volume 7553 LNCS, pages 58–65, 2012.

Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

Tobias Plotz and Stefan Roth. Benchmarking denoising algorithms with real photographs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Chourmouzios Tsiotsios and Maria Petrou. On the choice of the parameters for anisotropic diffusion in image processing. *Pattern recognition*, 46(5):1369–1381, 2013.

Gaihua Wang, Guoliang Yuan, Tao Li, and Meng Lv. An multi-scale learning network with depth-wise separable convolutions. *IPSJ Trans. Comput. Vis. Appl.*, 10(11), dec 2018. ISSN 1882-6695.

Min Wang, Baoyuan Liu, and Hassan Foroosh. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial ”bottleneck” structure, 2017. https://arxiv.org/abs/1608.04337.

Lexing Ying. Sparse Fourier transform via butterfly algorithm. *SIAM J. Sci. Comput.*, 31(3):1678–1694, jan 2009.

Appendix A. BNet Revisit

For a L layer BNet with an additional parameter L_t denoting the number of layers before switch layer, the feedforward network is as follows.

- **Layer 0:** The same as Layer 0 in CNN.
- **Layer $\ell = 1, \dots, L_t - 1$:** The same as Layer ℓ in BNet2.
- **Switch Layer:** This layer first applies many small dense layer to each part and then the role of spacial dimension and channel dimension is switched afterwards. We denote the hidden variables on switch layer as $Z^{(s)}$. The connection between the ℓ-th layer and the (s)-th layer hidden variables obeys,

$$Z^{(s)}\big(i, rj + c\big) = \sigma\Big(B^{(s)}\big(i, j, c\big) + \sum_{k \in [r]} D^{(s)}\big(i, j, c, k\big) Z^{(L_t)}\big(i, rj + k\big)\Big), \tag{24}$$

  for $j \in [2^{L_t}]$, $i \in [\frac{N}{2^{L_t} w}]$, and $c \in [r]$. Here $D^{(s)}$ and $B^{(s)}$ denote the weights and bias respectively.

- **Layer $\ell = L_t, \dots, L - 1$:** The $2^\ell r$ in-channels are equally partitioned into 2^ℓ parts. A 1D transposed convolutional layer is applied. The connection between the ℓ-th layer and the $(\ell + 1)$-th layer hidden variables obeys,

$$Z^{(\ell+1)}\big(2j + i, c\big) = \sigma\Big(B^{(\ell)}\big(c\big) + \sum_{k \in [2^{L-\ell} r]_p^{2^{L-\ell-1}}} W^{(\ell)}\big(i, k, c\big) Z^{(\ell)}\big(j, k\big)\Big), \tag{25}$$

  for $j \in [\frac{N}{2^{L-\ell} w}]$, $i \in [2]$, $c \in [2^{L-\ell-1} r]_p^{2^{L-\ell-1}}$, and $p \in [2^{L-\ell-1}]$. Here we abuse notation $Z^{(L/2)} = Z^{(s)}$.

- **Layer L:** The last layer links the L-th layer hidden variables with the output Y, i.e.,

$$Y\big(c\big) = \sum_{k \in [r]} \sum_{i \in [\frac{N}{w}]_p^{2^L}} W^{(L)}\big(i, k, c\big) Z^{(L)}\big(i, k\big), \tag{26}$$

  for $c \in [K]_p^{2^L}$ and $p \in [2^L]$.

Appendix B. Complex valued network

In order to realize complex number multiplication and addition via nonlinear neural network, we first represent a complex number as four real numbers, i.e., a complex number $x = \operatorname{Re} x + \imath \operatorname{Im} x \in \mathbb{C}$ is represented as

$$\begin{pmatrix}(\operatorname{Re} x)_+ & (\operatorname{Im} x)_+ & (\operatorname{Re} x)_- & (\operatorname{Im} x)_-\end{pmatrix}^\top, \tag{27}$$

where $(z)_+ = \max(z, 0)$ and $(z)_- = -\min(z, 0)$ for any $z \in \mathbb{R}$. The vector form of x contains at most two nonzeros. The complex number addition is the vector addition directly, while the complex number multiplication must be handled carefully. Let $a, x \in \mathbb{C}$ be two complex numbers. The multiplication $y = ax$ is produced as the activation function acting on a matrix vector multiplication, i.e.,

$$\sigma\left(\begin{pmatrix} \operatorname{Re} a & -\operatorname{Im} a & -\operatorname{Re} a & \operatorname{Im} a \\ \operatorname{Im} a & \operatorname{Re} a & -\operatorname{Im} a & -\operatorname{Re} a \\ -\operatorname{Re} a & \operatorname{Im} a & \operatorname{Re} a & -\operatorname{Im} a \\ -\operatorname{Im} a & -\operatorname{Re} a & \operatorname{Im} a & \operatorname{Re} a \end{pmatrix}\begin{pmatrix}(\operatorname{Re} x)_+ \\ (\operatorname{Im} x)_+ \\ (\operatorname{Re} x)_- \\ (\operatorname{Im} x)_-\end{pmatrix}\right) = \begin{pmatrix}(\operatorname{Re} y)_+ \\ (\operatorname{Im} y)_+ \\ (\operatorname{Re} y)_- \\ (\operatorname{Im} y)_-\end{pmatrix}. \tag{28}$$

In the initialization, all prefixed weights are in the role of a instead of x. In order to simplify the description below, we define an extensive assign operator as $\overset{\diamond}{=}$ such that the 4 by 4 matrix A in (28) then obeys $A \overset{\diamond}{=} a$. Without loss of generality, (28) can be extended to complex matrix-vector product and $\overset{\diamond}{=}$ notation is adapted accordingly as well.

Appendix C. Sketch Proof of Theorem 5

The detail proof of Theorem 5 is composed of layer by layer estimations on the multiplicative weight matrices, which is analogy to the proof of Theorem 4.8 in Li et al. (2019). Here we omit the detail and discuss the relations and differences.

If we consider the approximation under condition $L \leqslant \log K$, then the bound in Theorem 5 is exactly the same as that in Theorem 4.8 in Li et al. (2019) with $L_\xi = 0$, where L_ξ denotes the number of Conv-T layers after switch layer. However, when we consider $L > \log K$, the number of partitions of the frequency domain in BNet is limited by K due to the existence of switch layer. Hence the bottleneck domain pair, A and B as in Proposition 3, are of length 1 and 1 respectively. The Chebyshev interpolation error is then

$$\left(2 + \frac{2}{\pi}\ln r\right)\left(\frac{\pi e}{2r}\right)^r, \tag{29}$$

which can be well controlled as we increase r. Therefore, in Theorem 4.8 in Li et al. (2019), the approximation error is also controlled in terms of r.

BNet2, different from BNet, dose not have the constraint from switch layer. The frequency domain can be partitioned into 2^L subdomains and each has length $\frac{K}{2^L}$. When the number of subdomains is larger than the number of output frequencies, only those subdomains containing output frequencies are constructed in the network and considered in the proof. Due to the fine partition of the frequency domain, in Proposition 3, the product of the lengths of domain pair A and B is always bounded by $\frac{K}{2^L}$ and the Chebyshev interpolation error is

$$\left(2 + \frac{2}{\pi}\ln r\right)\left(\frac{\pi e K}{r 2^{L+1}}\right)^r \tag{30}$$

for any L. Replacing the interpolation error in the proof of Theorem 4.8 in Li et al. (2019) by (30) throughout layers proves Theorem 5.

Appendix D. BNet against BNet2

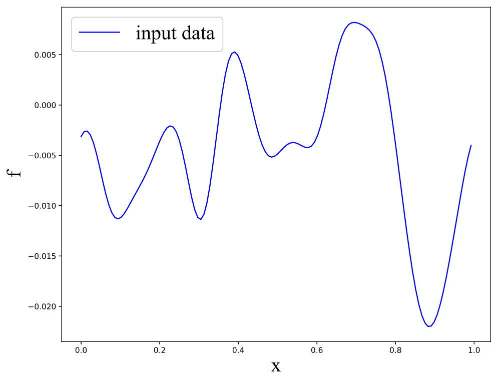
This numerical experiment compares BNet2 with BNet in a similar task as in Section 4.1.2. Comparing to that in Section 4.1.2, $L = 3$ are used here, while N, K and r remain the same. We train both BNet and BNet2 on the same training set with the same training hyper-parameters.

	♯ Parameters	training time	Pre-Train Rel Err	testing time	Test Rel Err
BNet2-FT	4596	68.3 s	1.28e−1	0.095 s	3.13e−4
BNet2-rand			1.06e+0		1.69e−2
BNet-FT	3876	95.4 s	8.01e−2	0.182 s	3.51e−4
BNet-rand			1.03e+0		1.84e−2

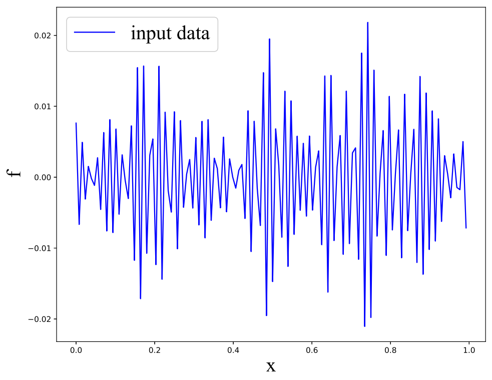
Table 6: Training results for BNet and BNet2

Due to the huge difference in architectures of BNet and BNet2, the numbers of parameters and pre-train relative errors are not identical but stay close to each other. After training, BNet2 achieves slightly better accuracy than that of BNet for both random initialization and FT initialization. This is likely due to the small difference in the number of parameters. Regarding the runtime, both the training and evaluation of BNet are more expensive than that of BNet2.

Appendix E. Extra Numerical Results



(a) Low frequency input example



(b) High frequency input example

Figure 6: (a) and (b) show examples of low and high frequency input used in Section 4.1.2.