Integral Neural Networks

Kirill Solodskikh*† Azim Kurbanov*† Ruslan Aydarkhanov†
Irina Zhelavskaya Yury Parfenov Dehua Song Stamatios Lefkimmiatis
Huawei Noah's Ark Lab
{kirillceo, azimcto, ruslancto}@garch.me
{zhelavskaya.irina1, parfenov.yury, dehua.song, stamatios.lefkimmiatis}@huawei.com

Abstract

We introduce a new family of deep neural networks, where instead of the conventional representation of network layers as N-dimensional weight tensors, we use a continuous layer representation along the filter and channel dimensions. We call such networks Integral Neural Networks (INNs). In particular, the weights of INNs are represented as continuous functions defined on N-dimensional hypercubes, and the discrete transformations of inputs to the layers are replaced by continuous integration operations, accordingly. During the inference stage, our continuous layers can be converted into the traditional tensor representation via numerical integral quadratures. Such kind of representation allows the discretization of a network to an arbitrary size with various discretization intervals for the integral kernels. This approach can be applied to prune the model directly on an edge device while suffering only a small performance loss at high rates of structural pruning without any fine-tuning. To evaluate the practical benefits of our proposed approach, we have conducted experiments using various neural network architectures on multiple tasks. Our reported results show that the proposed INNs achieve the same performance with their conventional discrete counterparts, while being able to preserve approximately the same performance (2% accuracy loss for ResNet18 on Imagenet) at a high rate (up to 30%) of structural pruning without fine-tuning, compared to 65% accuracy loss of the conventional pruning methods under the same conditions. Code is available at gitee.

1. Introduction

Recently, deep neural networks (DNNs) have achieved impressive breakthroughs in a wide range of practical applications in both computer vision [13, 20, 32] and natural language processing [7] tasks. This state-of-the-art performance is mainly attributed to the huge representation capacity [2] of DNNs. According to the Kolmogorov superposition theorem [14] and the universal approximation theorem [29], a DNN is capable of approximating uniformly any continuous multivariate function with appropriate weights. To achieve better performance, a large number of parameters and computations are assigned to the DNN [10, 40], which seriously limits its application on memory- and computation-constrained devices. Hence, numerous approaches have been proposed to compress and accelerate neural networks, including pruning [26, 38], quantization [35, 41] and neural architecture search [34, 37].

DNNs are particularly successful in dealing with challenging transformations of natural signals such as images or audio signals [27]. Since such analogue signals are inevitably discretized, neural networks conventionally perform discrete representations and transformations, such as matrix multiplications and discrete convolutions. However, for such kind of representations the size of neural networks cannot be adjusted without suffering severe performance degradation during the inference stage, once the training procedure is completed. Although several network pruning methods [26, 38] have been proposed to extract crucial channels from the trained model and generate efficient models, they either suffer from a significant accuracy degradation or require to fine-tune the model on the whole training database. Along with the development of hardware, there have been diverse edge devices with various capacities for memory and computation, from ordinary processors to dedicated neural network accelerators. The model size for different devices varies significantly [4]. Moreover, many tasks (e.g. autonomous driving) require different response speeds on the same hardware according to various scenarios or conditions (e.g. driving speed and weather condition). The conventional way to deal with such problems is to design multiple model architectures for all possible scenarios and store them together. However, the downside of such

*The authors contributed equally to this work.
†Currently affiliated with Garch Lab.

strategy is that it requires huge resources for training and memory space for storage. Hence, it is crucial to design neural networks that feature a self-resizing ability during inference, while preserving the same level of performance.

Inspired by the inherently continuous nature of the input signals, we challenge the discrete representation of neural networks by exploring a continuous representation along the filters and channel dimensions. This leads to a new class of networks which we refer to as Integral Neural Networks (INNs). INNs employ the high-dimensional hypercube to present the weights of one layer as a continuous surface. Then, we define integral operators analogous to the conventional discrete operators in neural networks. INNs can be converted into the conventional tensor representation by numerical integration quadratures for the forward pass. At the inference stage, it is convenient to discretize such networks into arbitrary size with various discretization intervals of the integral kernels. Since the representation is composed of integral operators, discretizing the continuous networks could be considered as the numerical quadrature approximation procedure [9]. The estimated values with various discretization intervals are close to the integral value when the interval is small enough. Hence, when we discretize an INN with different intervals to generate networks of various sizes, it is capable of preserving the original performance to some extent without the need of additional fine-tuning. Such kind of representation of neural networks can play a crucial role in dealing with the important problem of efficient network deployment in diverse conditions and hardware setups.

To evaluate the performance of INNs, extensive experiments were conducted on image classification and super-resolution tasks. The results show that the proposed continuous INNs achieve the same performance with their discrete DNN counterparts, when the training procedure is finished. Moreover, such kind of networks approximately preserve the performance at a high rate of structural pruning without the aid of additional fine-tuning.

2. Related Works

Continuous representations Continuous parameter representations have already been utilized in deep neural network architectures. In [31], the authors propose to use continuous convolutions to extend CNNs for non-uniform grid data. In [30], the weights of convolution layers were continuously parameterized in each section defined by the filter index and the channel index. Such parameterization allows for a flexible image upsampling but not pruning, since filters and channels are not connected by a continuous function. In [5] the authors proposed continuous-depth neural networks based on ordinary differential equations. The step size of the numerical ODE solver defines the number of layers, i.e., depth. In our work, we have constructed the integral neural networks based on Riemann integrals, which leads to continuous-width neural networks. In [24] the authors proposed deep neural networks with layers defined as functional operators. Such networks are designed for learning PDE solution operators, and its layers are continuously parameterized by MLPs only along the kernel dimensions. A re-discretization was investigated in terms of training on smaller data resolution and testing on higher input resolution. However, the proposed framework in [24] does not include continuous connections between filters and channels dimensions.

Structured pruning of neural networks Structured network pruning aims at reducing the redundancy of a model by removing channels with negligible impact on its performance. The authors in [33] were the first to propose a structured sparsity learning (SSL) method to regularize the structures according to the magnitude of weights. However, smaller norm weights do not always play a less informative role in the performance. Hence, [38] explored the valid criteria using a direct simplification of the channel-to-channel computation graph. Further, in [26] a collaborative channel pruning method that takes the inter-channel dependency into consideration was introduced. On the other hand, loss-aware pruning methods [17, 23] have also been widely investigated by researchers. However, pruning methods require network fine-tuning to improve the accuracy of small networks. To solve this issue, in the next section we propose a self-resized DNN with a continuous underlying representation, which allows network compression without the need of any fine-tuning.

3. Neural Networks and Integral Operators

The intuition behind our proposed idea is based on the observation that fully-connected and convolution layers could be considered as numerical integration of specific integrals. To illustrate this, we consider the following example. Let $W(x), S(x)$ be univariate functions, then we have [11]:

$$\int_0^1 W(x)S(x)dx \approx \sum_{i=0}^{n} q_i W(x_i)S(x_i) = \vec{w}_q \cdot \vec{s}, \qquad (1)$$

where $\vec{w}_q = \big(q_0 W(x_0), \ldots, q_n W(x_n)\big)$, $\vec{s} = \big(S(x_0), \ldots, S(x_n)\big)$, $\vec{q} = (q_0, \ldots, q_n)$ are the *weights of the integration quadrature*, and $\vec{P}^x = (x_0, \ldots, x_n)$ is the *segment partition* that satisfies the following inequality: $0 = x_0 < x_1 < \ldots < x_{n-1} < x_n = 1$. The pair $(\vec{P}^x, \vec{q})$ is called a *numerical integration method* [16]. General numerical integration methods are built using different approximations of input functions (we refer to the examples depicted in Fig. 2). From Eq. (1) we can see that the integral of a product of two univariate functions can be approximated by the dot product of two vectors using a

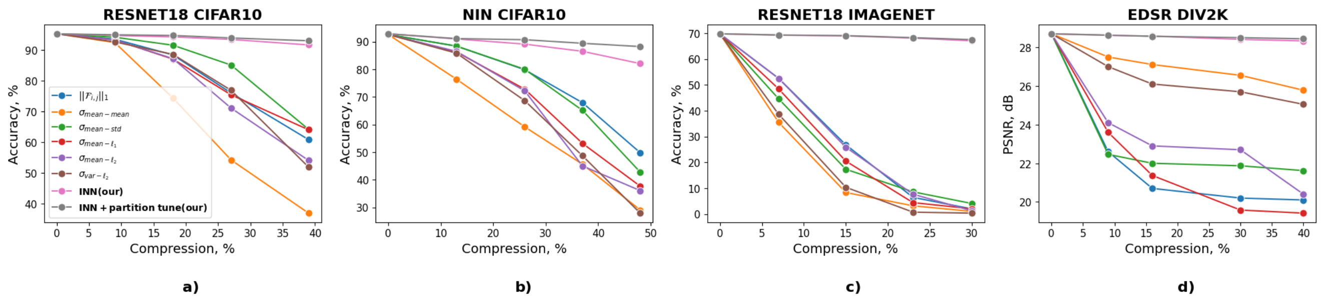

Figure 1. Visualization of different channels selection methods without fine-tuning compared with our proposed integral neural networks. a) ResNet-18 on Cifar10. b) NIN architecture on Cifar10. c) ResNet-18 on ImageNet. d) 4x EDSR on Div2k validation set. By compression we denote the percentage of deleted parameters.

specific numerical integration method. The size of vectors $\vec{w}_q$ and $\vec{s}$ can change to arbitrary values by selecting a larger or smaller partition $\vec{P}^x$. For more details on the numerical integration of multiple integrals we refer to Appendix A. The proposed use of these integrals for representing basic network layers, such as convolution and fully-connected ones, allows for various segment partition lengths along filters and channels dimension or height and width. This leads to the generation of layers with the desired number of filters, channels, height, and width.

3.1. DNNs layers as integral operators

Commonly used linear network layers can be presented as integral operators with a specific integral kernel. While such layers act as linear operators on the real linear space $\mathbb{R}^k$, integral operators act as linear operators on the linear space of integrable functions $\mathbf{L}^2$. Therefore, not all input data could be considered as continuous integrable functions in a meaningful way. Nevertheless, digital images and audio signals are discretizations of analog signals and, therefore, they can be naturally used in integral networks.

Convolution or cross-correlation layer The convolution layer defines a transform of a multichannel signal to another multichannel signal. In the case of integral operators, weights of this layer are represented by an integrable function $F_W(\lambda, x^{out}, x^{in}, \mathbf{x^s})$, where $\mathbf{x^s}$ is a scalar or vector representing the dimensions, over which the convolution is performed, and λ is a vector of trainable parameters. Input and output images are represented by integrable functions $F_I(x^{in}, \mathbf{x^s}), F_O(x^{out}, \mathbf{x^{s'}})$ and are connected through the weight function in the following way:

$$F_O(x^{out}, \mathbf{x^{s'}}) = \int_\Omega F_W(\lambda, x^{out}, x^{in}, \mathbf{x^s}) F_I(x^{in}, \mathbf{x^s} + \mathbf{x^{s'}}) dx^{in} d\mathbf{x^s}. \quad (2)$$

Fully-connected layer A fully-connected layer defines a transform of a vector to a vector by means of matrix multiplication. The weights of this layer are represented by an integrable function $F_W(\lambda, x^{out}, x^{in})$. Similar to the convolution operator, λ defines a vector of trainable parameters of the integral kernel. The input and output functions are represented by the integrable functions $F_I(x^{in}), F_O(x^{out})$, respectively, and are connected via the weight function as follows:

$$F_O(x^{out}) = \int_0^1 F_W(\lambda, x^{out}, x^{in}) F_I(x^{in}) dx^{in}. \quad (3)$$

Pooling and activation functions Pooling layers also exhibit a meaningful interpretation in terms of integration or signal discretization. Average pooling could be interpreted as a convolution along the spatial dimensions with a piece-wise constant function. MaxPooling could be interpreted as a way of signal discretization. Activation functions in integral networks are naturally connected with activation functions in conventional networks by the following equation:

$$\mathcal{D}\big(\text{ActFunction}((x), P_x\big) = \text{ActFunction}(\mathcal{D}(x, P_x)\big), \quad (4)$$

where by $\mathcal{D}$ we denote the *discretization operation* that evaluates a scalar function on the given partition P_x. This equation implies that applying an activation function on a discretized signal is equivalent to discretizing the output of the activation function applied to the continuous signal.

Evaluation and backpropagation through integration For fast integral evaluation, the integral kernel goes through a discretization procedure and is then passed to a conventional layer for numerical integration. It turns out that any composite quadrature may be represented by such a conventional layer evaluation. For backpropagation through integration we use the chain-rule to evaluate the gradients of the trainable parameters λ as in discrete networks. The validity of the described procedure is guaranteed by the following lemma, whose proof can be found in Appendix A.

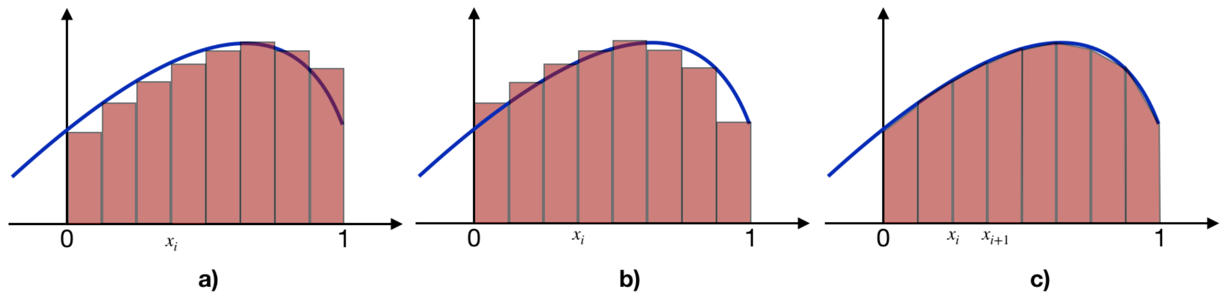

Figure 2. Different integration quadratures: a) left Riemann quadrature, b) right Riemann quadrature, c) trapezoidal quadrature. Riemann qudratures are first-order methods, while the trapezoidal quadrature is a second-order method. The trapezoidal quadrature computes the integral more precisely than the Riemann quadratures with a fewer required number of points in the segment partition.

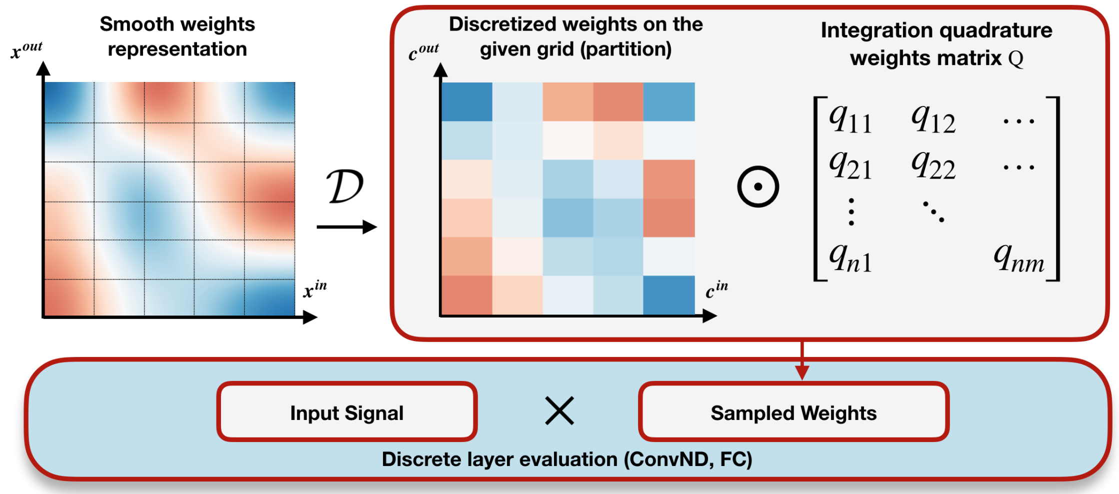
Smooth weights representation
x^{out}
x^{in}
$\mathcal{D}$
Discretized weights on the given grid (partition)
c^{out}
c^{in}
$\odot$
Integration quadrature weights matrix Q
$\begin{bmatrix} q_{11} & q_{12} & \cdots \\ q_{21} & q_{22} & \cdots \\ \vdots & \ddots & \\ q_{n1} & & q_{nm} \end{bmatrix}$
Input Signal
$\times$
Sampled Weights
Discrete layer evaluation (ConvND, FC)

Figure 3. Visualization of the integral layer evaluation. Continuous weights go through discretization along the variables x^{in}, x^{out} and adjusted by an element-wise product with the integration quadrature Q.

Lemma 1 (Neural Integral Lemma) *Given that an integral kernel $F(\lambda, x)$ is smooth and has continuous partial derivatives $\frac{\partial F(\lambda,x)}{\partial \lambda}$ on the unit cube $[0,1]^n$, any composite quadrature can be represented as a forward pass of the corresponding discrete operator. The backward pass of the discrete operator corresponds to the evaluation of the integral operator with the kernel $\frac{\partial F(\lambda,x)}{\partial \lambda}$ using the same quadrature as in the forward pass.*

3.2. Continuous parameters representation

The richer and more generalized continuous parameter representation allows to sample discrete weights at inference time at any given resolution. We propose to compactly parameterize the continuous weights as a linear combination of interpolation kernels with uniformly distributed interpolation nodes on the line segment $[0,1]$: $F_W(\lambda, x) = \sum_{i=0}^{m} \lambda_i u(xm - i)$, where m and λ_i are the number of interpolation nodes and their values, respectively. For efficiency purposes, we suggest to exploit the available hardware and existing Deep Learning (DL) frameworks and rely on the cubic convolutional interpolation as shown in Fig. 4, which is used for efficient image interpolation on GPUs. Despite its slight deviation from the cubic spline interpolation, this approach is significantly faster and yet preserves the details better than linear interpolation. In case of multiple dimensions, we propose to define the interpolation on the cube $[0,1]^n$ with separable kernels, which is fully compatible with the existing DL frameworks.

The continuous representation is discretized into a standard weight tensor W which is used by the corresponding layer in the forward pass. A schematic visualization of a continuous parameter representation and discretization is depicted in Fig. 4.

Representation of weights for fully-connected and convolutional layers Fully-connected layers are defined by a two-dimensional weight tensor and, thus, we represent them

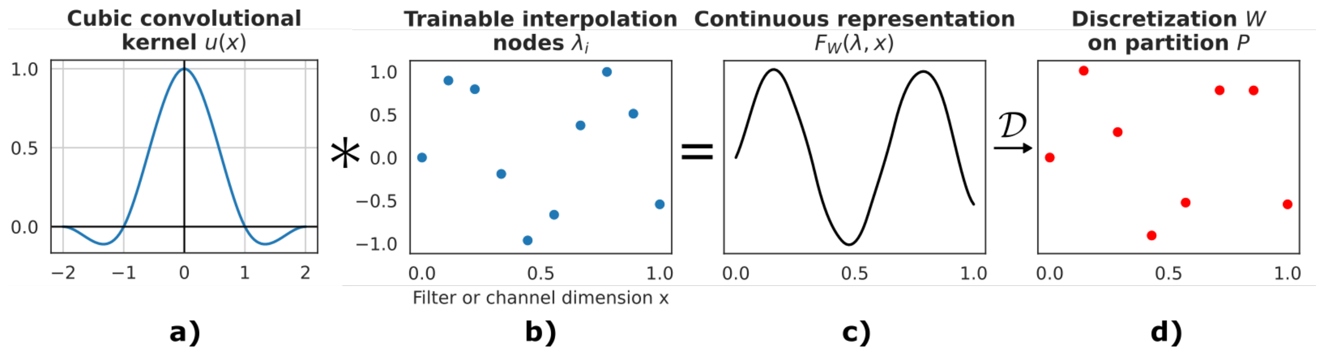

Figure 4. Visualization of continuous parameter representation and sampling along one dimension. The continuous representation (c) is the result of a linear combination of a cubic convolutional kernel (a) with interpolation nodes (b). During the forward phase it is discretized (d) and combined with an integration quadrature.

with a linear combination of two-dimensional kernels on a uniform 2D grid within the square $[0, 1]^2$:

$$F_W(\lambda, x^{out}, x^{in}) = \sum_{i,j} \lambda_{ij} u\big(x^{out}m^{out}-i\big)u\big(x^{in}m^{in}-j\big). \tag{5}$$

The discretized weight tensor W_q of the fully-connected layer is obtained by sampling the continuous representation on partitions $\vec{P}^{out}$ and $\vec{P}^{in}$ and by weighting the result according to the integration quadrature of Eq. (1):

$$W_q[k, l] = q_l W[k, l] = q_l F_W(\lambda, P^{out}_k, P^{in}_l). \tag{6}$$

Uniform partitions with steps h^{out} and h^{in} are defined as follows: $\vec{P}^{out} = \{kh^{out}\}_k$ and $\vec{P}^{in} = \{lh^{in}\}_l$. Fewer or more filters and channels at inference time are obtained with a varying partition size.

As for convolutional layers, in this study we omit re-sampling convolution kernels along the spatial dimensions $\mathbf{x^s}$. Therefore, the continuous representation of weights could be viewed as already sampled at each spatial location t and defined by F_W with location dependent set of interpolation nodes $\lambda(t)$.

Trainable partition So far, we considered only uniform partitions with a fixed sampling step. However, non-uniform sampling can improve numerical integration without increasing the partition size. This relaxation of the fixed sampling points introduces new degrees of freedom and leads to a trainable partition. By training the separable partitions we can obtain an arbitrary rectangular partition in a smooth and efficient way. Such a technique opens up the opportunity for a new structured pruning approach. Combined with the conversion strategy of Section 4, this can reduce the size of pre-trained discrete DNNs without tuning the rest of the parameters. Instead of using a direct partition parameterization $\vec{P}$ we employ a latent representation by the vector $\vec{\delta} = (0, \delta_1, \ldots, \delta_n)$ so that the following holds: $\vec{\delta}_{norm} = \frac{\vec{\delta}^2}{\text{sum}(\vec{\delta}^2)}$, $\vec{P} = \text{cumsum}(\vec{\delta}_{norm})$. Such parameterization guarantees that the result is a correctly defined (sorted) partition $\vec{P}$ stretched over the whole segment $[0, 1]$.

4. Training Integral Neural Networks

Nowadays, there exists a large variety of pre-trained discrete networks. Therefore, it would be beneficial to have in place a process of converting such networks to integral ones. Such converted networks can serve as a better initialization for the training of integral networks. To this end, we propose an algorithm that *permutes* the filters and channels of the weight tensors in order to obtain a smooth structure in discrete networks. A visual illustration of this strategy is provided in Fig. 5. We also propose an algorithm to optimize the smooth parameters representation of INNs using gradient descent. This allows us to obtain a network which can be re-sampled (structurally pruned) without any fine-tuning at inference time.

Conversion of DNNs to INNs To find a permutation that leads to the smoothest structure possible, we minimize the total variation along a specific dimension of the weight tensor. This problem is equivalent to the well-known Traveling Salesman Problem (TSP) [15]. In our task, the slices along the c^{out} dimension in the weight tensor (i.e., filters) correspond to the “cities” and the total variation to the “distance” between those cities. Then, the optimal permutation can be considered as an optimal “route” in TSP terms. We use the 2-opt algorithm [6] to find the permutation of filters that minimizes the total variation along that dimension:

$$\min_{\sigma \in S_n} \sum \big|W[\sigma(i)] - W[\sigma(i+1)]\big|, \tag{7}$$

where W is the weight tensor, σ denotes the permutation, $\sigma(i)$ is the new position of the i-th element defined by the permutation, and S_n is a set of all permutations of length n. The permutation is performed in such a way so that the filters' permutation in the preceding layer matches the channels' permutation in the following layer. Since the model output stays exactly the same, our algorithm allows to initialize the integral neural network using a discrete one without experiencing any quality degradation.

Optimization of continuous weights Any available gradient descent-based method can be used for training the proposed integral neural networks. We use Lemma 1 to construct the training algorithm as described below. We train our networks with random c^{out} from a predefined range (c^{out} denotes the number of filters or rows in the convolution or the fully-connected layer). Discretization of x^{in} of the next layer is therefore defined by discretization of x^{out} of the previous layer. Training integral neural networks using such an approach allows for a better generalization of the integral computation and avoids overfitting of the weights to a fixed partition, since the size of the partition changes at every training iteration. Formally, our training algorithm minimizes the differences between different cube partitions for each layer using the following objective:

$$\begin{aligned}\big|\mathrm{Net}(X, P_1) - \mathrm{Net}(X, P_2)\big| \leq \big|\mathrm{Net}(X, P_1) - Y\big| \\ + \big|\mathrm{Net}(X, P_2) - Y\big|,\end{aligned} \tag{8}$$

where $\mathrm{Net}(X, P_i)$ is the neural network evaluated on input data X with labels Y, and P_1, P_2 are two different partitions for each layer. One can note that the optimization under stochastic sampling of the partition sizes leads to a reduction of differences between the outputs of integral neural networks of different sizes. Such an optimization therefore ensures that a trained integral neural network has a similar performance when pruned to arbitrary sizes.

5. Experiments

We have implemented a general framework for numerical integration using the PyTorch library [25]. Our framework allows to use custom integration quadratures for numerical integration on Nvidia GPUs. Our integral neural networks use PyTorch layers for fast evaluation of integral operators. One could use our framework for numerical integration of multiple integrals independently on the integral network. To validate INNs we have conducted experiments for image classification and image super-resolution tasks. For image classification the Cifar10 [12] and ImageNet [28] datasets were used. For the image super-resolution task we have selected the 4-x EDSR [19] and 3-x SRCNN [8] models trained on the Div2k dataset [1] and the 91-image dataset [36], respectively. Validation was performed on Set5 [3], Set14 [39] and B100 [21] datasets. A schematic description of the main validation pipelines is shown in Fig. 6. Details of the implementation and the experiments setup can be found in Appendix D.

Pipeline A. Comparison with discrete NNs We have trained INNs with two different initializations: from scratch and from a converted pre-trained discrete network. We can

Algorithm 1 Training of the integral neural network.

```
Require: Integral neural network IntegralNet.
Require: Dataset $X$.
 1: $P^0$ := Partition(NumberOfInputChannels($X$))  ▷ Input partition
 2: $P^L$ := Partition(NumberOfOutputChannels($X$))  ▷ Output partition
 3: while not converge do
 4:     $x$ := SampleBatch($X$)
 5:     for each layer $l$ in Layers(IntegralNet) do  ▷ $l$ starts from 1
 6:         $P^l$ := Partition(min$^l$, max$^l$)  ▷ Generate uniform partition with random size
 7:         $W_q^l$ := DiscretizeWeight($\lambda^l$, $P^l$, $P^{l-1}$)  ▷ Obtain weight tensor according to Eq. 6
 8:         $b^l$ := DiscretizeBias($\lambda^l$, $P^l$)  ▷ Obtain 1D bias tensor similar to Eq. 6
 9:     end for
10:     $L$ := Loss(IntegralNet, $x$)  ▷ Forward pass
11:     $\frac{\partial L}{\partial \lambda}$ := Backward($L$)  ▷ Backward pass
12:     $\lambda$ := OptimizerStep($\lambda$, $\frac{\partial L}{\partial \lambda}$)  ▷ Update parameters
13: end while
```

see that the INN fine-tuned from the pre-trained discrete network has the same or higher performance as the corresponding discrete network and significantly outperforms the INN trained from scratch (see Table 1). The INN fine-tuned using Algorithm 1 can be re-sampled to a corresponding discrete network of any desired size. Figure 7 shows a comparison of the EDSR INN with its discrete counterpart. It can be seen that even after 40% pruning the INN preserves almost the same performance.

Pipeline B. Structured pruning without fine-tuning through conversion to INN In this experiment, we have pruned discrete networks through their conversion to INNs using the method described in Section 4, and tuned the integration partition on a few samples. Results are presented in Fig. 1 (denoted as *INN + partition tuning*). It is important to note that the permutation step is very important in our conversion algorithm. Indeed, we observe a higher accuracy drop when partition tuning is deployed without the permutation step. This is in line with the theoretical connection of variation and integration errors (see Appendix A) and with our empirical evaluation reported in Table 2.

Pipeline C. Structured pruning without fine-tuning of discrete NNs A straightforward approach to obtain a compact neural network is to remove some of the parameters of a trained network in a structured way. Typically, the importance of neurons or filters of each layer is assessed based on various criteria $\rho(W, X)$ that depend on the network weights W and data X, such as the ℓ_1-norm of the weights [18] or the contribution to the loss [22]. In our

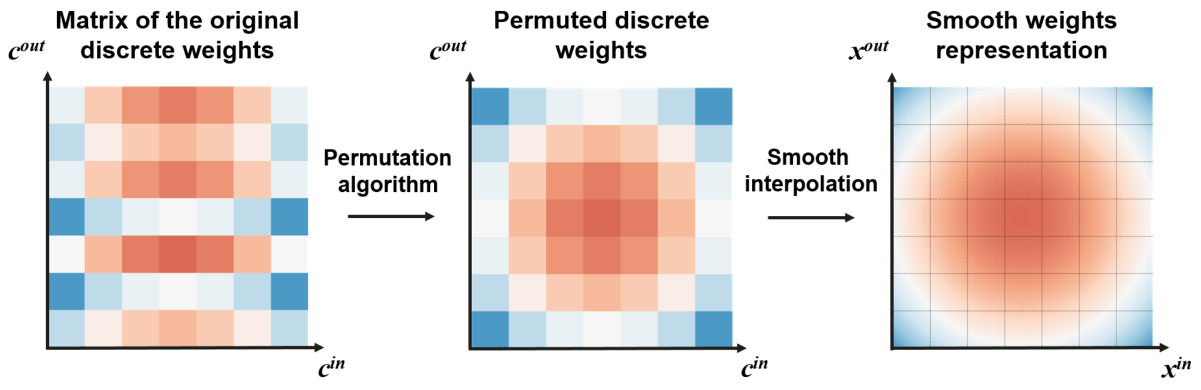
Matrix of the original discrete weights
c^{out}
c^{in}
Permutation algorithm
Permuted discrete weights
c^{out}
c^{in}
Smooth interpolation
Smooth weights representation
x^{out}
x^{in}

Figure 5. Toy example illustrating the permutation of filters in a discrete weight tensor in order to obtain a smoother structure.

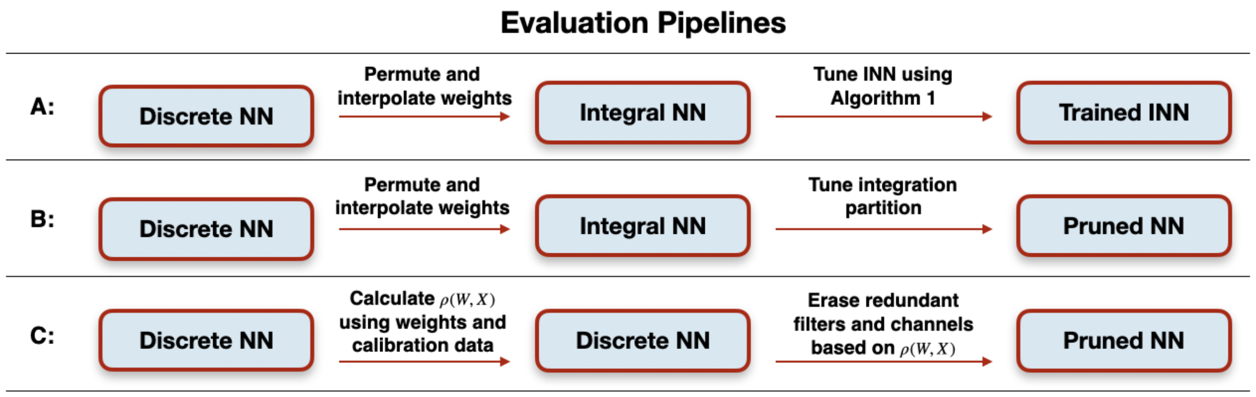
Evaluation Pipelines
A:
Discrete NN
Permute and interpolate weights
Integral NN
Tune INN using Algorithm 1
Trained INN
B:
Discrete NN
Permute and interpolate weights
Integral NN
Tune integration partition
Pruned NN
C:
Discrete NN
Calculate $\rho(W, X)$ using weights and calibration data
Discrete NN
Erase redundant filters and channels based on $\rho(W, X)$
Pruned NN

Figure 6. Illustration of the main validation experiments.

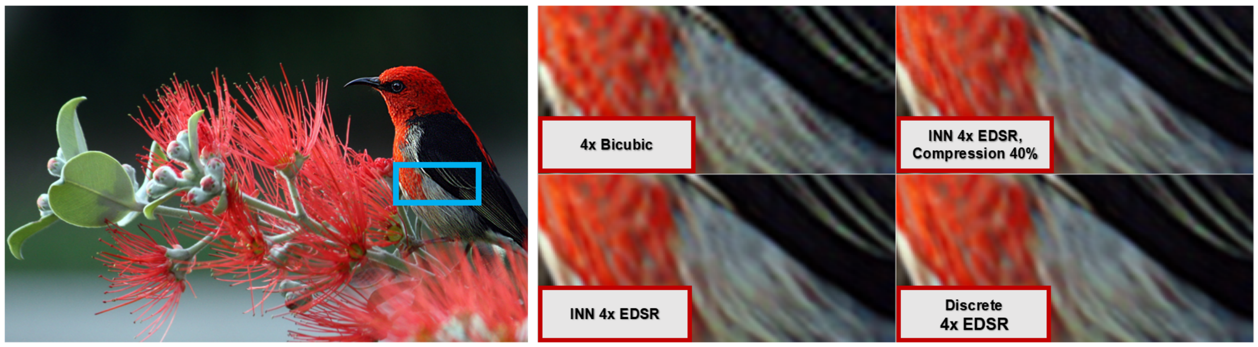

Figure 7. Example of 4x image super-resolution with 4 methods: bicubic interpolation, EDSR discrete neural network, EDSR integral neural network of full-size and pruned by 40%.

study, structural pruning is applied to convolutional layers as in [18]. The neurons to prune are selected based on the ℓ_1-norm of their kernels or various statistics of feature maps estimated on the whole dataset. Finally, the results from all 3 pipelines are presented in Fig. 1. From these results we can see that INNs significantly outperform other alternative methods equipped with the ability of pruning without fine-tuning. Further, we observe that the non-uniform trainable partition outperforms pre-trained INNs with uniform sampling, while the uniform sampling provides much faster and data-free partition of the re-discretization.

Trainable partition for the improvement of continuous representations Cubic convolution interpolation is typically applied to discrete data such as images or volume. However, the uniformly sampled partition can limit the reconstruction quality. Therefore, a trainable partition pro-

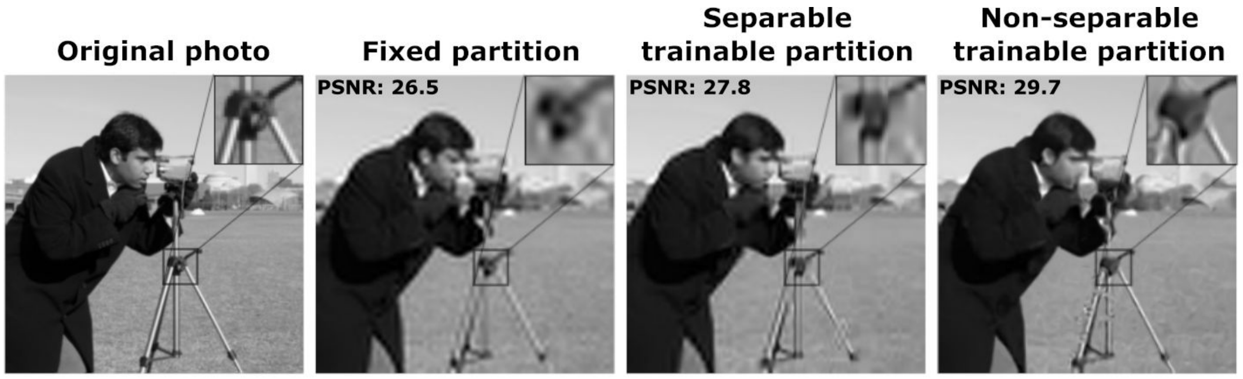
Original photo
Fixed partition
PSNR: 26.5
Separable trainable partition
PSNR: 27.8
Non-separable trainable partition
PSNR: 29.7

Figure 8. Image reconstruction with 3 methods (from left to right): original image, interpolation kernels with fixed partition, with separable trainable partition and non-separable trainable partition.

Dataset	Model	Discrete	INN	INN-init
Cifar10	NIN	92.3	91.8	92.5
	VGG-11	91.1	89.4	91.6
	Resnet-18	95.3	93.1	95.3
ImageNet	VGG-19	72.3	68.5	72.4
	ResNet-18	69.8	66.5	70.0
	ResNet-50	74.1	71.2	74.1

(a)

Dataset	Model	Discrete	INN	INN-init
Set5	SRCNN 3x	32.9	32.6	32.9
	EDSR 4x	32.4	32.2	32.4
Set14	SRCNN3x	29.4	29.0	29.4
	EDSR 4x	28.7	28.2	28.7
B100	SRCNN 3x	26.8	26.1	26.8
	EDSR 4x	27.6	27.2	27.6

(b)

Table 1. Comparison of INNs with discrete networks on classification and image super-resolution tasks for different architectures. **Discrete** refers to the conventional DNN, **INN** refers to the integral network trained from scratch, while **INN-init** refers to the integral network trained according to pipeline A indicated in Fig. 6. Table (a) indicates accuracy [%] for classification tasks, whereas table (b) indicates PSNR [dB] for super-resolution tasks.

	w Perm., %	w/o Perm., %
ResNet-18	**93.0**	91.3
NIN	**89.4**	84.71
VGG-11	**88.7**	85.2

Table 2. Tuning integration partition of INN with and without permutation step during conversion from pre-trained DNN. All models were compressed at 40 %.

vides additional flexibility to enrich the signal reconstruction and may lead to a higher quality representation. We assess the representation capability of interpolation kernels with fixed and separable trainable partitions by performing image reconstruction (Fig. 8). Additionally, we have tested the reconstruction with the use of a non-separable trainable partition parameterized by a set of independent 2D coordinates. Since the partition parameterization introduces additional parameters, we equalize the total number of parameters by adjusting the number of interpolation nodes and the partition size.

6. Conclusions and open problems

In this paper, we proposed a novel integral representation of neural networks which allows us to generate conventional neural networks of arbitrary shape at inference time by a simple re-discretization of the integral kernel. Our results show that the proposed continuous INNs achieve the same performance as their discrete DNN counterparts, while being stable under structured pruning without the use of any fine-tuning. In this new direction, the following questions/problems are worth to be further investigated:

- INNs open up new possibilities for investigating the capacity of neural networks. The Nyquist theorem can be used to select the number of sampling points.
- Adaptive integral quadratures. In this work, we have investigated only uniform partitions for training INNs. Investigating data-free non-uniform partition estimation could also have strong impact on INNs.
- Training INN from scratch requires improvement for classification networks. Current accuracy drop probably caused by absence of batch-normalization layers. Smooth analogue of normalization is required.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 6

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013. 1

[3] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. *Proceedings of the British Machine Vision Conference*, pages 135.1–135.10, 2012. 6

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2019. 1

[5] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 2

[6] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958. 5

[7] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, 2018. 1

[8] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016. 6

[9] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical algorithms*, 18(3):209–232, 1998. 2

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1

[11] D. Hughes-Hallett and W. G. McCullum. *Calculus*. Wiley, 2005. 2

[12] Alex Krizhevsky and Geoffrey E Hinton. Learning multiple layers of features from tiny images. *Technical report*, 2009. 6

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 1

[14] Vera Kurkova. Kolmogorov's theorem and multilayer neural networks. *Neural networks*, 5(3):501–506, 1992. 1

[15] Eugene L. Lawler. *The Traveling salesman problem: A guided tour of combinatorial optimization*. Chichester [West Sussex] ; New York : Wiley, 1985. 5

[16] J. J. Leader. *Numerical analysis and scientific computation*. CRC Press, 2022. 2

[17] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2018. 2

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 6, 7

[19] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017. 6

[20] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318, 2020. 1

[21] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423, 2001. 6

[22] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 6

[23] P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*, 2019. 2

[24] Zongyi Li Nikola Kovachki. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020. 2

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019. 6

[26] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122. PMLR, 2019. 1, 2

[27] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019. 1

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. 6

[29] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998. 1

[30] Assaf Shocher, Ben Feinstein, Niv Haim, and Michal Irani. From discrete to continuous convolution layers. *arXiv preprint arXiv:2006.11120*, 2020. 2

[31] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018. 2

[32] Zhihao Wang, Jian Chen, and Steven CH Hoi. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 1

[33] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. 2016. 2

[34] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 1

[35] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7308–7316, 2019. 1

[36] J. Yang, J. Wright, T. Huang, and Y. Ma. Image super-resolution as sparse representation of raw image patches. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8, June 2008. 6

[37] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020. 1

[38] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. 1, 2

[39] R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. *International conference on curves and surfaces*, pages 711–730, 2010. 6

[40] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 1

[41] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized neural networks with a full-precision auxiliary module. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1