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

Summary

- It is crucial to design neural networks that feature a self-resizing ability during inference, while preserving the same level of performance.
- Instead of tensors, the weights of INNs are represented as continuous functions.

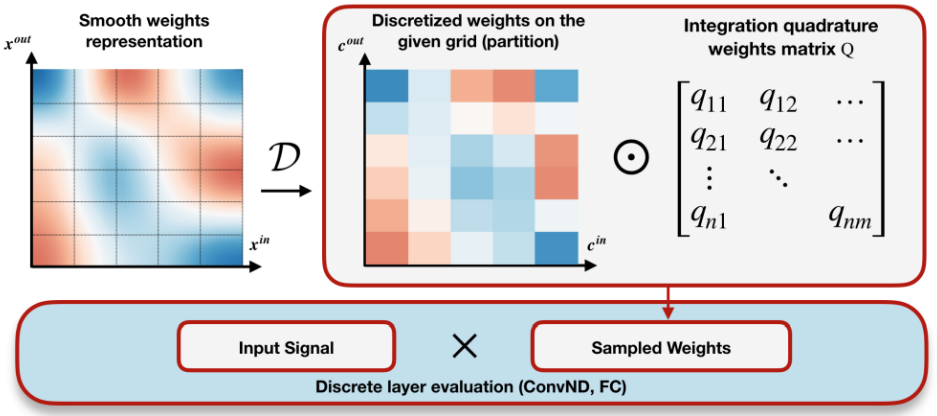


Figure 3. Visualization of the integral layer evaluation. Continuous weights go through discretization along the variables x^{in}, x^{out} and adjusted by an element-wise product with the integration quadrature Q.

Layers in NNs can be considered as numerical integration

- Fully-connected and convolution layers could be considered as numerical integration of specific integrals.
- Utilizing continues weight functions instead of tensors leads to the generation of layers with the desired number of filters, channels, height, ar width.

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

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].

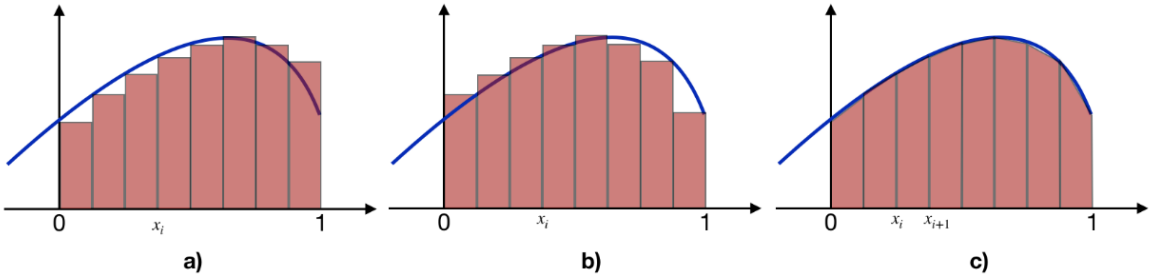


Figure 2. Different integration quadratures: a) left Riemann quadrature, b) right Riemann quadrature, c) trapezoidal quadrature. Riemann qudratures are first-order methods, while the trapezoidal quadrature is a second-order method. The trapezoidal quadrature computes the integral more precisely than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

Fully-connected layer as as integral operator

- Input and Output functions are represented by the integrable functions

$$F_I(x^{in}), F_O(x^{out})$$

- The weights of this layer are represented by an integrable function

$$F_W(\lambda, x^{out}, x^{in})$$

- Utilizing continues weight functions instead of tensors leads to the generation of layers with the desired number of filters, channels, height, and width.

Fully-connected operator

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

Convolution operator:

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

Smooth representation of weights

- Parameterize weight function of integral layer by a sum of interpolation kernels of finite support

$$F_W(\lambda, x) = \sum_{i=0}^{m} \lambda_i u(xm - i)$$

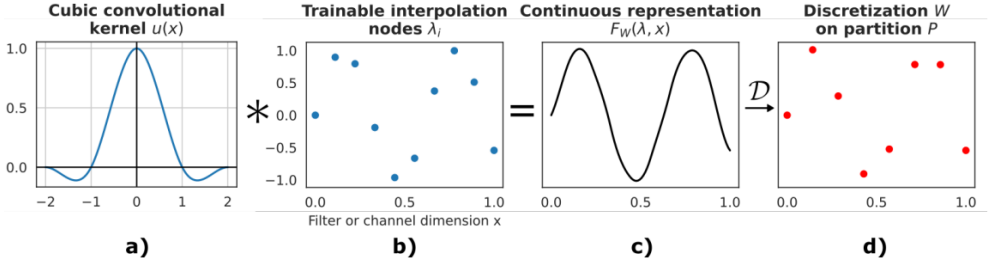
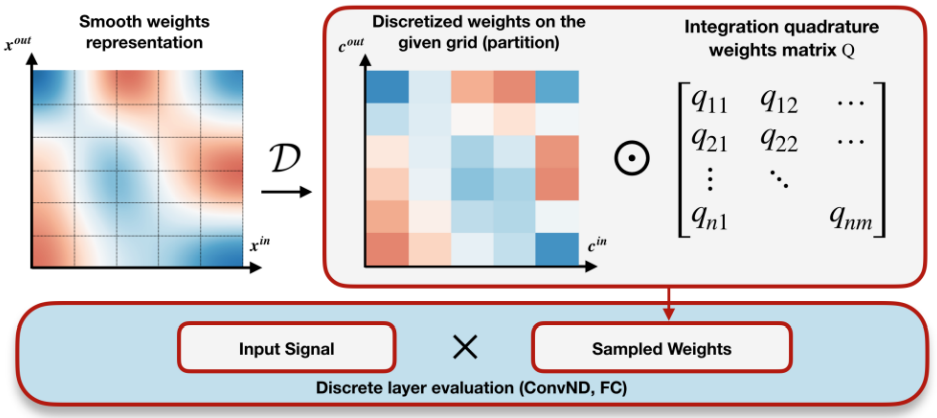


Figure 4. Visualization of continuous parameter representation and sampling along one dimension. The continuous representation (c) is the result of a linear combination of a cubic convolutional kernel (a) with interpolation nodes (b). During the forward phase it is discretized (d) and combined with an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

- Use cubic convolutional kernels

- For smooth representation of 2D tensors, use a linear combination of 2D kernels:

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



Results

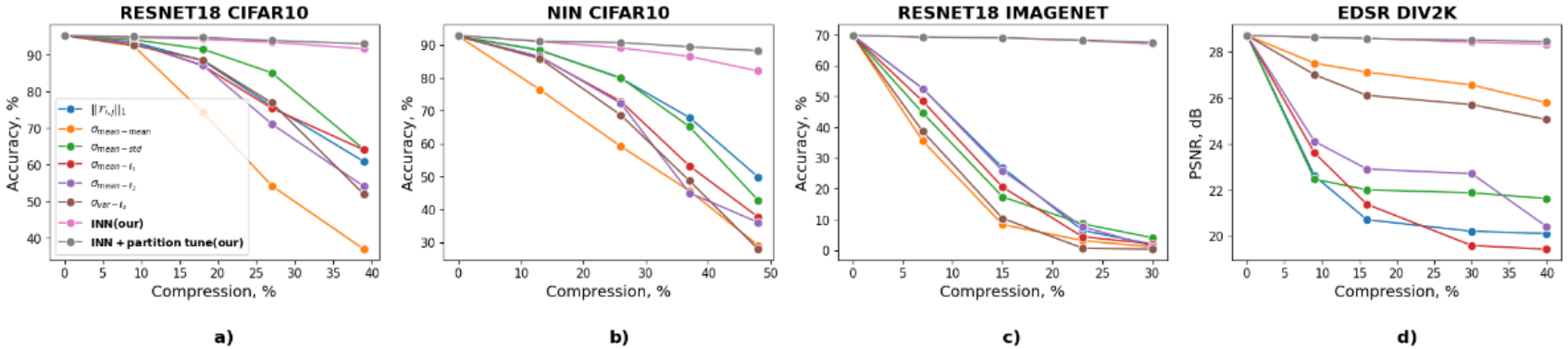


Figure 1. Visualization of different channels selection methods without fine-tuning compared with our proposed integral neural networks. a) ResNet-18 on Cifar10. b) NIN architecture on Cifar10. c) ResNet-18 on ImageNet. d) 4x EDSR on Div2k validation set. By compression we denote the percentage of deleted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

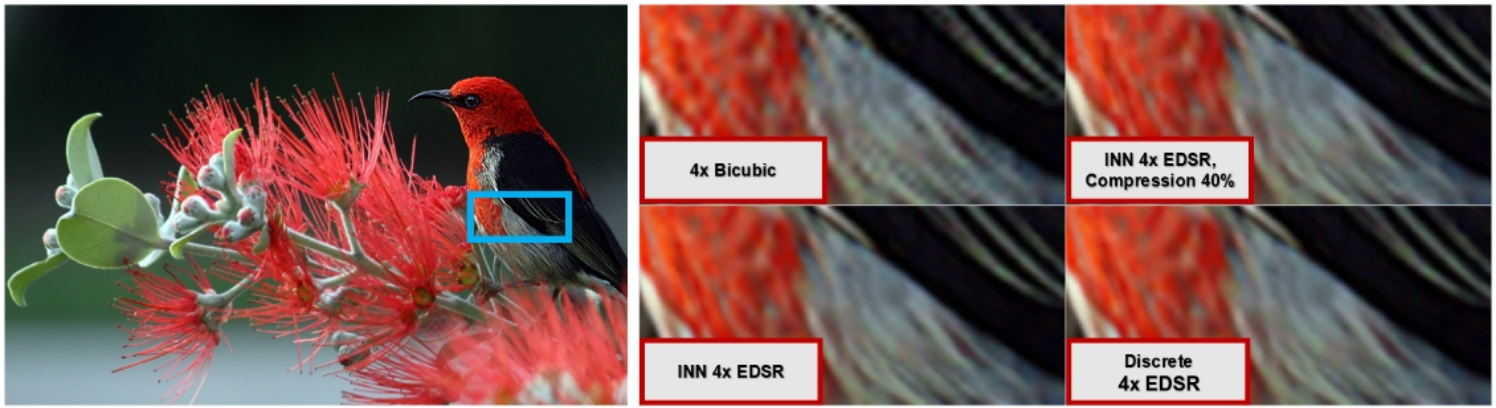


Figure 7. Example of 4x image super-resolution with 4 methods: bicubic interpolation, EDSR discrete neural network, EDSR integral neural network of full-size and pruned by 40%.

Results

" 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 Image-net) 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"

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.

References

https://openaccess.thecvf.com/content/CVPR2023/papers/Solodskikh_Integral_Neural_Networks_CVPR_2023_paper.pdf

https://openaccess.thecvf.com/content/CVPR2023/supplemental/Solodskikh_Integral_Neural_Networks_CVPR_2023_supplemental.pdf

https://www.youtube.com/watch?v=MdSVyi00r3E&ab_channel=KirillSolodskikh

https://inn.thestage.ai/