Integral Neural Networks

(CVPR 23 Award Candidate)

Kirill Solodskikh*† Azim Kurbanov*† Ruslan Aydarkhanov†
Irina Zhelavskaya Yury Parfenov Dehua Song Stamatios Lefkimmiatis

Huawei Noah's Ark Lab

Presenter: Seonghak KIM

Introduction

† Kolmogorov superposition theorem
¶ universal approximation theorem

- **DNN (Deep Neural Networks)**
  - **Characteristic**
    - Approximating continuous multivariate function † ¶ → huge representation
    - Large number of parameters and computations for better performance
    - Limit applications (memory- and computation-constrained devices) → pruning, quantization, NAS
  - **Discretized representations**
    - Natural signals such as images or audio signals, which inevitably discretized.
    - Matrix multiplications and discrete convolutions
    - Modification of size of NN (→ performance degradation)
      - Although pruning can generate efficient models, it requires to fine-tune the on whole training datasets.
      - Many tasks (e.g., autonomous driving) require different response speeds on same hardware according to various conditions (e.g., driving speed and weather condition).
      - Multiple model for all possible scenarios and store them together → resources ↑

→ self-resizing model without performance degradation

Introduction

- **INN (Integral Neural Networks)**
  - **Continuous representation**
    - Integral operators
    - High-dimensional hypercube to present the weights of one layer as a continuous surface
    - Numerical quadrature approximation (continuous networks → discretization)
    - At inference, arbitrary size with various discretization intervals (while preserving original performance)

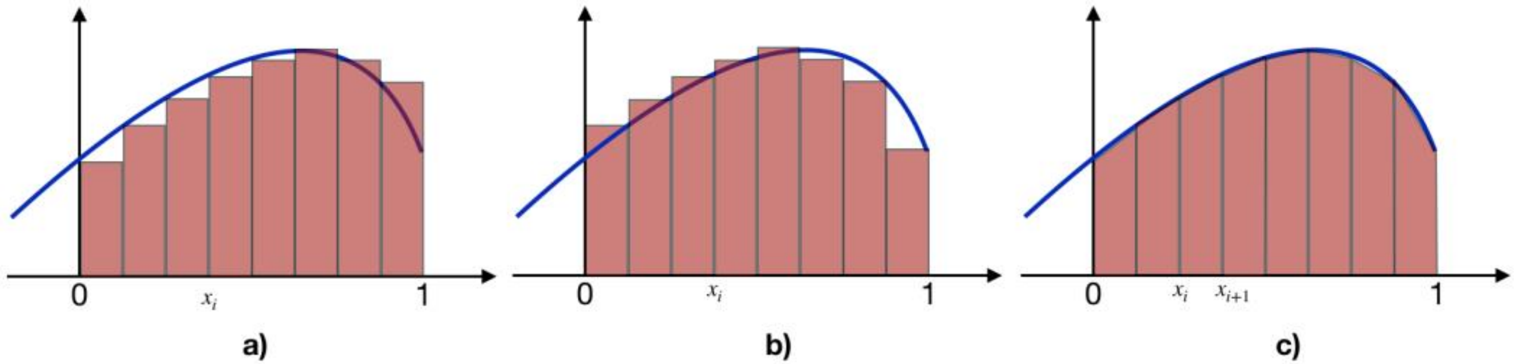


Figure 2. Different integration quadratures: a) left Riemann quadrature, b) right Riemann quadrature, c) trapezoidal quadrature. Riemann qudratures are first-order methods, while the trapezoidal quadrature is a second-order method. The trapezoidal quadrature computes the integral more precisely than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

Methods

- **Preliminary**
  - **Full-connected and convolution layer → numerical integration**
    - Let $W(x), S(x)$ be univariate functions

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

    - $\vec{w}_q = \left(q_0 W(x_0), \dots, q_n W(x_n)\right)$
    - $\vec{s} = (S(x_0), \dots, S(x_n))$
    - $\vec{q} = (q_0, \dots, q_n)$: weights of the integration quadrature
    - $\vec{P}^x = (x_0, \dots, x_n)$: segment partition $(0 < x_0 < x_1 < \cdots < x_{n-1} < x_n < 1)$
    - $(\vec{P}^x, \vec{q})$: numerical integration method

→ Integral of a product of two univariate functions ≈ dot product of two vectors (w/ specific integration method)

Methods

- **Convolution layer**
  - **Multichannel signal → Multichannel signal (i.e., $\mathbb{R}^{d\times C_{in}} \to \mathbb{R}^{d\times C_{out}}$)**
    - $F_W(\lambda, x^{out}, x^{in}, \mathbf{x}^s)$: weight of layer represented by integrable function
      - $\mathbf{x}^s$: scalar or vector representing the dimensions
      - λ: vector of trainable parameters
    - $F_I(x^{in}, \mathbf{x}^s)$: input images
    - $F_O(x^{out}, \mathbf{x}^{s'})$: output images

$$\boxed{\int_0^1 W(x)S(x)dx \approx \vec{w}_q \cdot \vec{s}}$$

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

- **Fully-connected layer**
  - **Vector → vector (i.e., $\mathbb{R}^d \to \mathbb{R}^d$)**

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

- **Activation function**

$$\mathcal{D}(\mathrm{ActFunc}(x), P_x) = \mathrm{ActFunc}(\mathcal{D}(x, P_x))$$

Methods

¶ Fubini theorem
† Leibniz theorem

Evaluation and Backpropagation

Evaluation (Forward pass)

- Integral kernel $F(\lambda, x)$ → discretization → conventional layer for numerical integration
- (∵) weights of a quadrature can be fused into the weight matrix of the vanilla layer
- E.g., fully-connected layer, $F_W(\lambda, x^{out}, x^{in})$ and $F_I(\lambda, x^{in})$: continuous function

$$\int_0^1 F_W(\lambda, x^{out}, x^{in}) F_I(x^{in}) dx^{in} \approx^{\P} \sum_{i=0}^{n} \underbrace{\boxed{q_i F_W(\lambda, x^{out}, x_i^{in})}}_{W_{ji}} F_I(x_i^{in})$$

➔ *Composite quadrature can be represented as a forward pass of the corresponding discrete operator.*

Backpropagation

$$\begin{aligned}\frac{\partial}{\partial \lambda}\int_0^1 F_W(\lambda, x^{out}, x^{in}) F_I(x^{in}) dx^{in} \;\&\approx^{\dagger} \int_0^1 \frac{\partial F_W(\lambda, x^{out}, x^{in})}{\partial \lambda} F_I(x^{in}) dx^{in} \\ &= \sum_{i=0}^{n} q_i \frac{F_W(\lambda, x^{out}, x_i^{in})}{\partial \lambda} F_I(x_i^{in})\end{aligned}$$

➔ *Evaluation of the integral operator with the kernel $\frac{\partial F(\lambda,x)}{\partial \lambda}$ using the same quadrature as in the forward pass.*

Methods

¶ Fubini theorem
† Leibniz theorem

- **Evaluation and Backpropagation**
  - **Evaluation (Forward pass)**

$$\int_0^1 F_W(\lambda, x^{out}, x^{in}) F_I(x^{in}) dx^{in} \approx^{\P} \sum_{i=0}^{n} \underbrace{\boxed{q_i F_W(\lambda, x^{out}, x_i^{in})}}_{W_{ji}} F_I(x_i^{in})$$

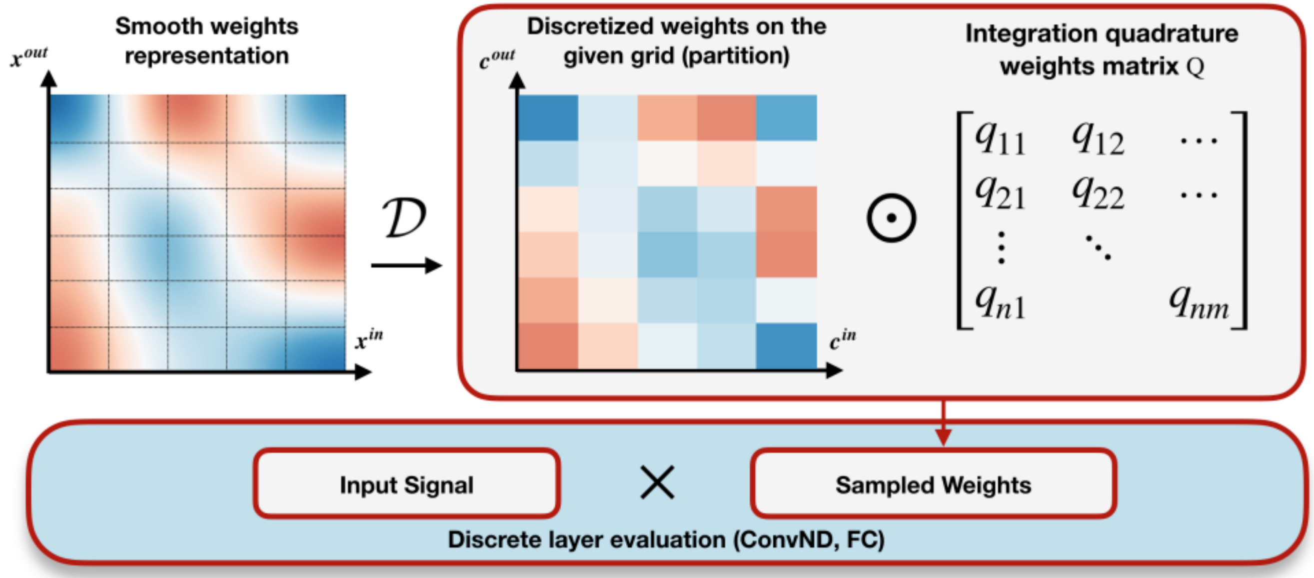


Figure 3. Visualization of the integral layer evaluation. Continuous weights go through discretization along the variables x^{in}, x^{out} and adjusted by an element-wise product with the integration quadrature Q.

Methods

- **Continuous parameters representation $F(\lambda, x)$**
  - **Linear combination**
    - Richer and more generalized continuous parameter representation → sample discrete weights
    - Interpolation kernels with uniformly distributed interpolation nodes on the line segment [0,1]

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

- m: number of interpolation nodes
- λ_i: node's value

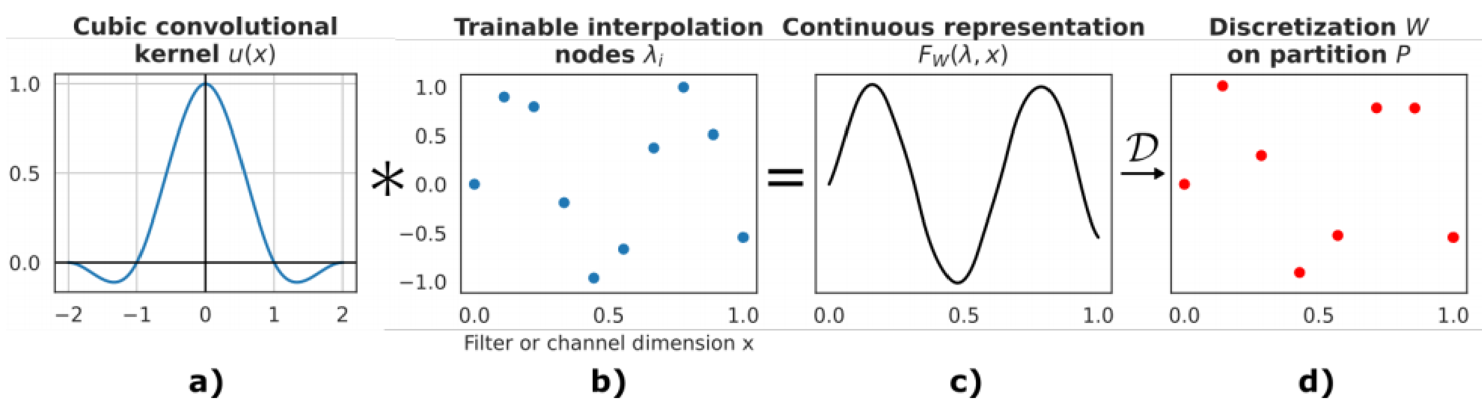


Figure 4. Visualization of continuous parameter representation and sampling along one dimension. The continuous representation (c) is the result of a linear combination of a cubic convolutional kernel (a) with interpolation nodes (b). During the forward phase it is discretized (d) and combined with an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

$$\text{cf.} \sum_{i=0}^{n} \underbrace{q_i F_W(\lambda, x^{out}, x_i^{in})}_{W_{ji}} F_I(x_i^{in})$$

Methods

- **Continuous parameters representation $F(\lambda, x)$**
  - **Fully-connected layer**
    - Two dimensional weight tensor represented by linear combination of two-dimensional kernels on a uniform 2D grid within the square $[0,1]^2$

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

    - Sampling continuous representations on partitions $\vec{P}^{out}, \vec{P}^{in} \rightarrow W_q$

$$W_q[k,l] = q_l W[k,l] = q_l F_W(\lambda, P_k^{out}, P_k^{in})$$

      - $\vec{P}^{out} = \{kh^{out}\}_k, \vec{P}^{in} = \{lh^{in}\}_l$: uniform partitions with steps h^{out}, h^{in}

      ➔ Various partition size make diverse sized model
  - **Trainable partition**
    - Non-uniform sampling → improve numerical integration w/o partition size ↑
    - Training the separable partitions

$$\vec{P} = \mathrm{cumsum}(\vec{\delta}_{\mathrm{norm}})$$

      - $\vec{\delta}_{\mathrm{norm}} = \frac{\vec{\delta}^2}{\mathrm{sum}(\vec{\delta}^2)}, \ \vec{\delta} = (0, \delta_1, \dots, \delta_n)$

Methods

- **Training INN**
  - **Conversion of DNNs to INNs**
    - Benefits of use from pre-trained discrete networks by converting into integral networks
    - Better initialization for training of integral networks
    - *Permutation* by total variation minimization along a specific dimension of the weight tensor
      - *cf*. optimal "route" in TSP as optimal permutation→ slices along the c^{out} = "cities", total variation = "distance" b/w cites

$$2\text{opt}: \min_{\sigma \in S_n} \sum |W[\sigma(i)] - W[\sigma(i+1)]|$$

- σ: permutation, $\sigma(i)$: new position of i element by the permutation
- S_n: set of all permutations of length n

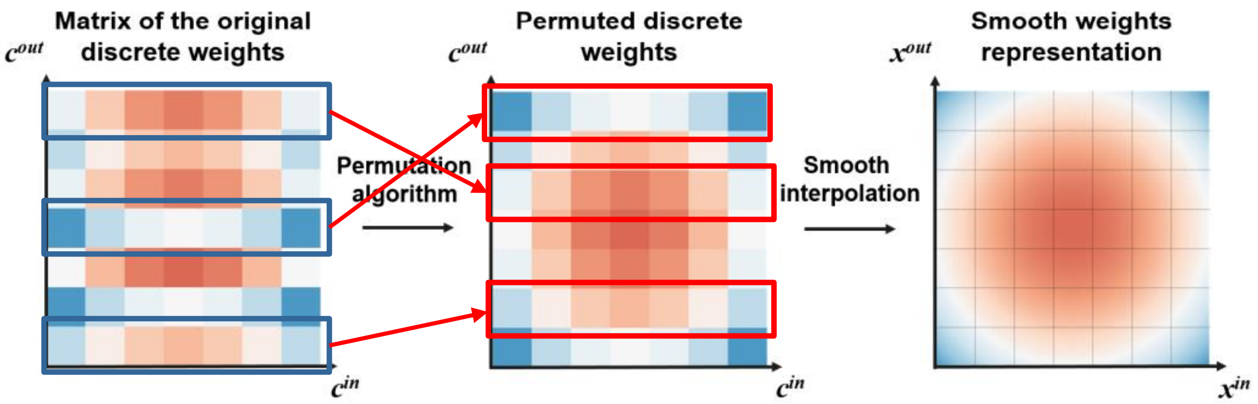
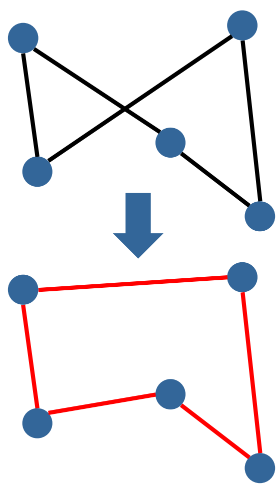


Figure 5. Toy example illustrating the permutation of filters in a discrete weight tensor in order to obtain a smoother structure.



[TSP]

Methods

- **Training INN**
  - **Optimization of continuous weights**
    - Training algorithm minimizes the differences b/w different cube partitions for each layer

$$|\mathrm{Net}(X, P_1) - \mathrm{Net}(X, P_2)| \leq |\mathrm{Net}(X, P_1) - Y| + |\mathrm{Net}(X, P_2) - Y|$$

      - $\mathrm{Net}(X, P_i)$: neural network evaluated on input data X with labels Y
      - P_1, P_2: two different partitions for each layer
    - Reduction of differences between the outputs of INNs of different sizes
      ➔ trained INNs has a similar performance when pruned to arbitrary sizes.

BREIL

Experiments

KAIST

● Comparison with discrete NNs

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.

➔ Performance: INN from pre-trained discrete net ≥ discrete net ≫ INN from scratch

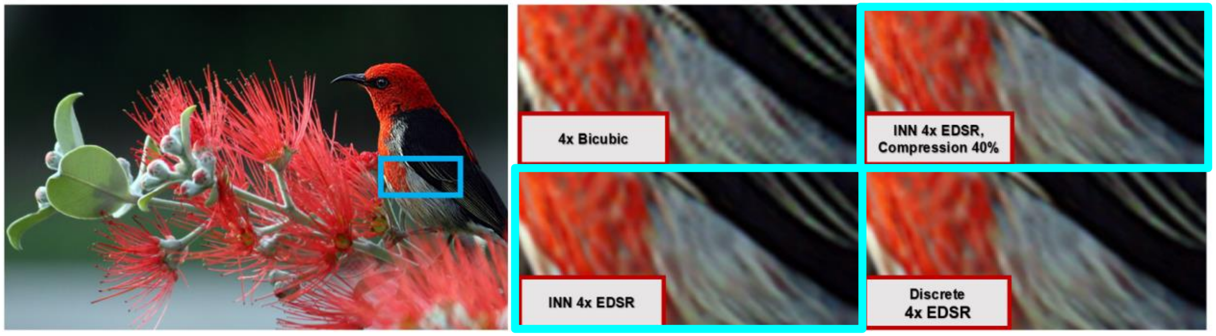


Figure 7. Example of 4x image super-resolution with 4 methods: bicubic interpolation, EDSR discrete neural network, EDSR integral neural network of full-size and pruned by 40%.

➔ Even after 40% pruning the INN preserves almost the same performance.

Experiments

● Structured pruning w/o fine-tuning through conversion to INN

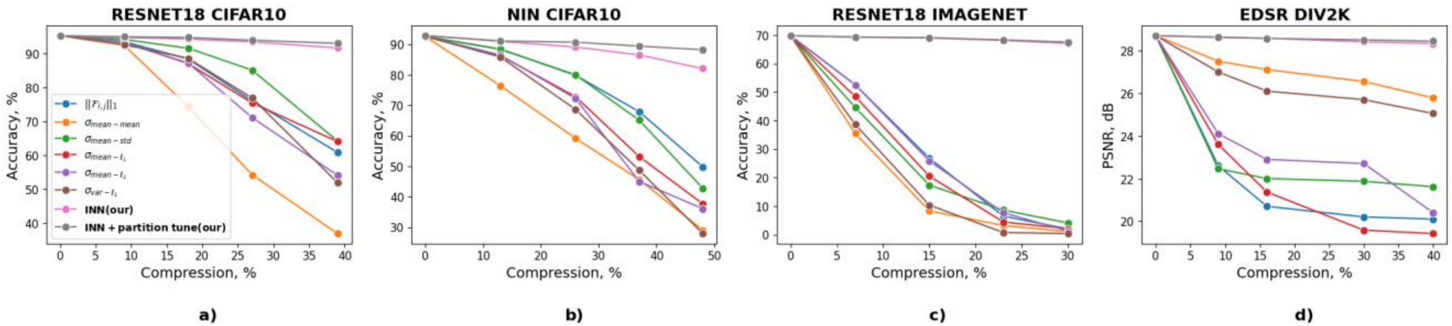


Figure 1. Visualization of different channels selection methods without fine-tuning compared with our proposed integral neural networks. a) ResNet-18 on Cifar10. b) NIN architecture on Cifar10. c) ResNet-18 on ImageNet. d) 4x EDSR on Div2k validation set. By compression we denote the percentage of deleted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

→ INNs significantly outperform other alternative equipped with the ability of pruning w/o fine-turning.

	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 %.

→ w/o permutation, higher accuracy drop when partition tuning is applied.

Experiments

● Trainable partition

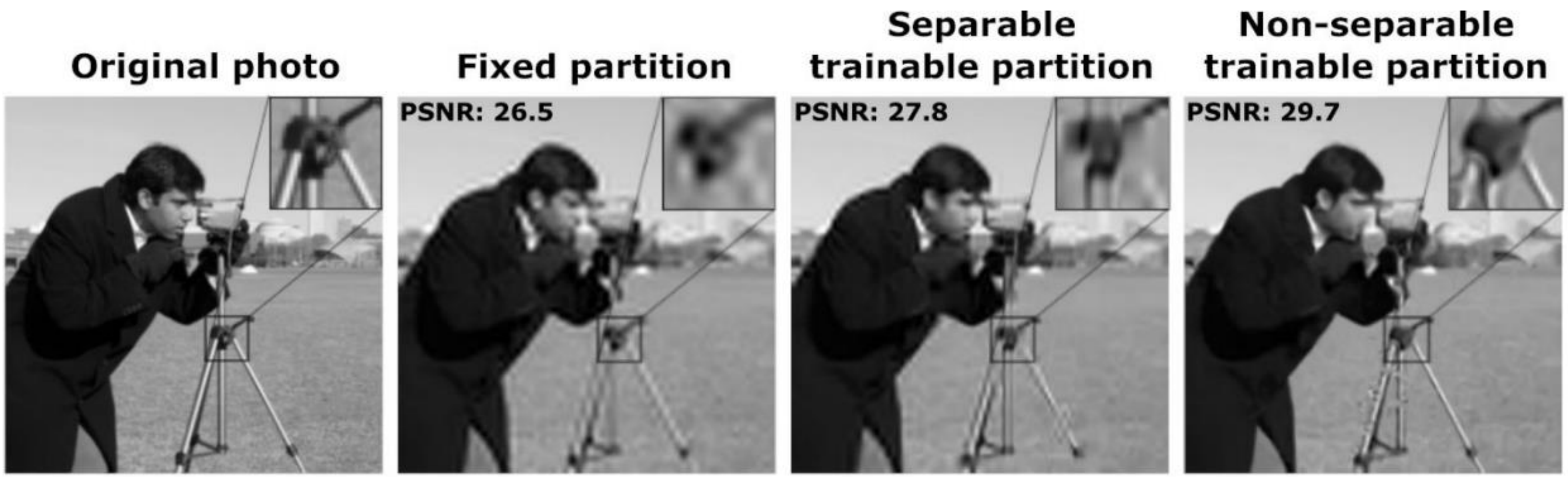


Figure 8. Image reconstruction with 3 methods (from left to right): original image, interpolation kernels with fixed partition, with separable trainable partition and non-separable trainable partition.

➔ Additional flexibility to enrich the signal reconstruction leads to higher quality representation

Conclusions

- **Conclusions**
  - **Integral representation of neural networks**
    - It generate conventional neural networks of arbitrary shape by a re-discretization of the integral kernel.
    - Same performance as their discrete DNN counterparts, while being stable under pruning w/o the fine-tuning.

- **Open problems**
  - **Nyquist theorem**
    - How to select the number of sampling points
  - **Adaptive integral quadrature**
    - Non-uniform partition estimation
  - **Training from scratch**
    - Absence of batch-normalization layers

Thank you.