∫NNs

INTEGRAL NEURAL NETWORKS

CVPR2023, Award Candidate

K. Solodskikh[2*], A. Kurbanov[2*], R. Aydarkhanov[2],
I. Zhelavskaya[1], Y. Parfenov[1], D. Song[1], S.Lefkimmiatis[1]
Huawei Noah Ark Lab [1], Thestage.ai [2], *-equally contributed

paper

code

What is Integral Neural Networks?

- Integral Neural Networks (INNs) is the new class of neural networks which uses **integral operators instead of conventinal layers**
- INNs utilizes **smooth parameters representation** instead of tensor representation
- Such a representation allows fast resampling of pre-trained INN delivering **structured pruning without fine-tuning**
- The main idea could be touched through simple Riemann integral:

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

$$\vec{w}_q = \left(q_0 W(x_0), \ldots, q_n W(x_n)\right), \; \vec{s} = \left(S(x_0), \ldots, S(x_n)\right),$$

$\vec{q} = (q_0, \ldots, q_n)$ are the weights of the integration quadruature,

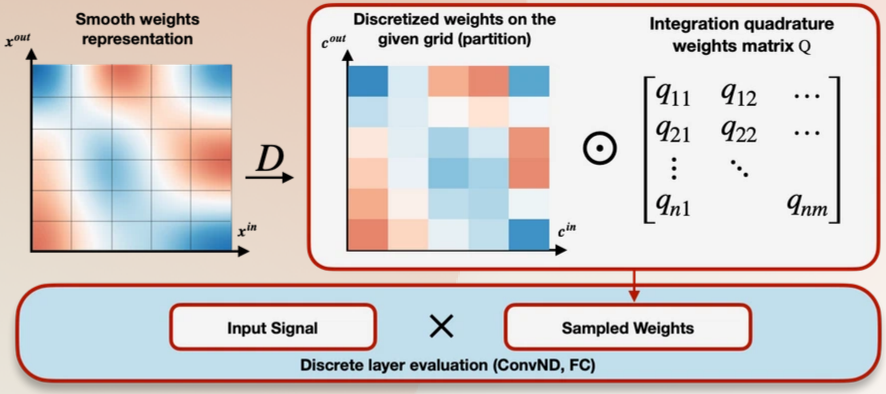
$$0 = x_0 < x_1 < \ldots < x_{n-1} < x_n = 1$$

How integral layers works?

- Integral layers are integral operators of specific type on linear space of integrable functions.
- Vanilla discrete layers coincide with numerical integration quadratures of corresponding integral layers.

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



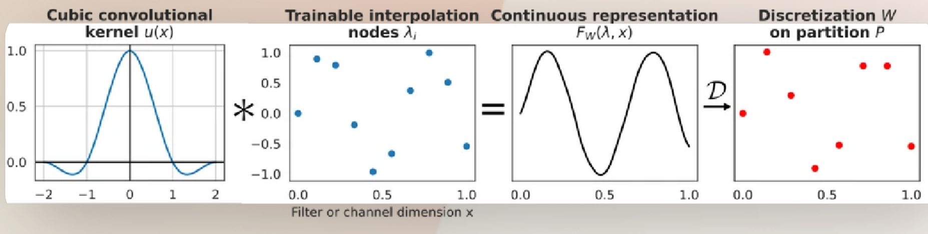
Smooth representation of weights

- Because of efficiency we propose to parametrize weight function of integral layer by a **sum of interpolation kernels of finite support**
- Specifically, we utilizing **cubic convolutional kernels**. Such a parametrization supported by main deep learning framewroks like **TensorFlow, PyTorch for signals and images resizing**:

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

- On forward pass weights goes through the discretization process and adjusted by quadrature weigths:

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



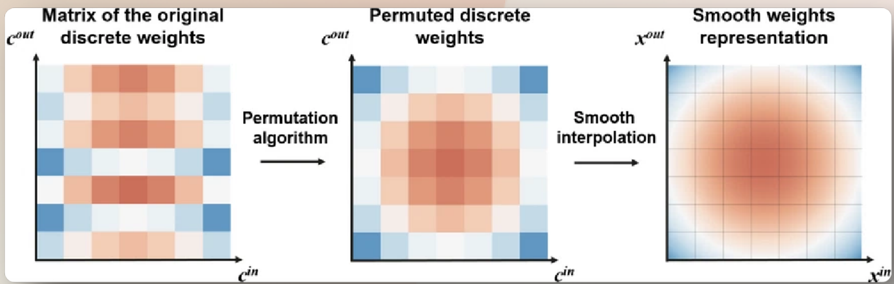
Backpropagation through integration

For backpropagation through integration we use the chain-rule to evaluate the gradients of the trainable parameters as in discrete networks. The validity of the described procedure is guaranteed by the combination of Fubini's theorem and Leibniz rule and can be formulated as the following simple lemma.

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.*

Conversion of DNN to INN

- Nowadays, there exists a large variety of pre-trained discrete networks
- It would be beneficial to have in place a process of converting such networks to integral ones
- 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
- To find a permutation, we build equivalent problem to the well-known Traveling Salesman Problem
- Resulted network has the same quality as initial discrete NN



Training of INNs

- Any available gradient descent-based method can be used for training the proposed integral neural networks
- We use Neural Integral Lemma to construct the training algorithm
- We train our networks with random number of output channels / rows from a predefined range
- Training INNs using such an approach allows for a better generalization of the integral computation
- Our training algorithm minimizes the differences between different cube partitions for each layer using the following objective:

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

Trainable Integration Grid

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.

INNs Framework

TORCHINTEGRAL

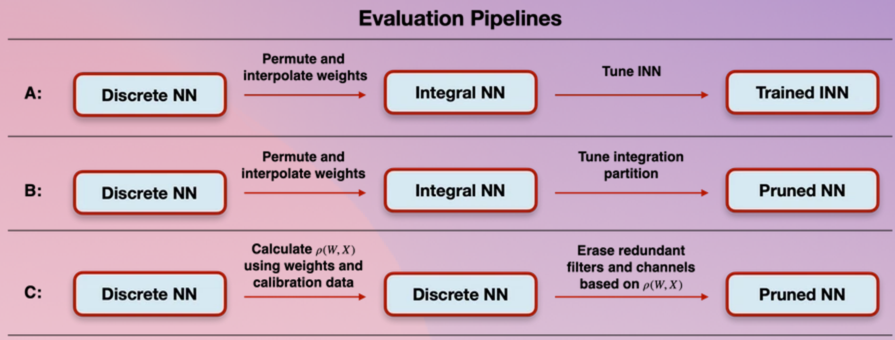
- TorchIntegral is the Python framework for numerical evaluation of **arbitrary integrals**
- Framework has general enough interface and supports **arbitrary integration quadratures**
- Support of integral layers and weights parametrization customization
- **Automatic conversion** of discrete DNNs to INNs

```python
import torch_integral
import torchvision.models as models

model = models.resnet18(pretrained=True)
# convert discrete model to INN
model = torch_integral.IntegralWrapper(
    init_from_discrete=True,
    quadrature='trapezoidal',
    parametrization='cubic_conv',
).wrap_model(model)
```

Experiments / Overview

- Comparison of discrete NNs with INNs
- Comparison of INNs trained from scratch and INNs initialized from pre-trained discrete network
- Comparison of INNs resampling and structured pruning without fine-tuning



Experiments / INNs vs DNNs

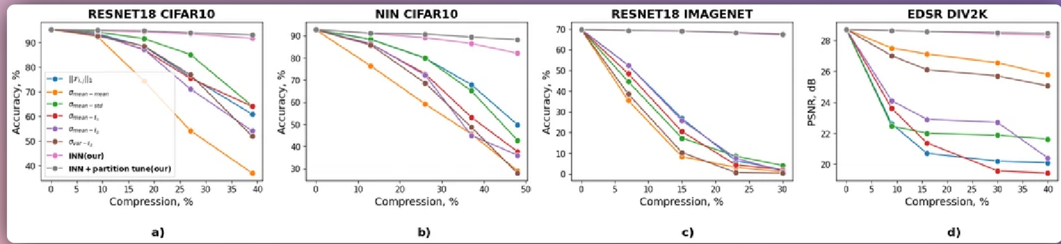
- Comparison of trained vanilla DNNs, INNs trained from scratch and INN initialized by our conversion algorithm (INN-init)
- INN with our initialization acheives **the same preformance** as corresponding vanilla DNN

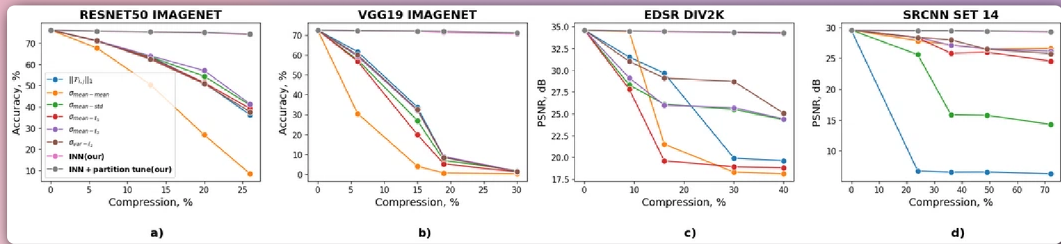
Dataset	Model	Discrete	INN	INN-init
Cifar10	NIN	92.3	91.8	92.5
	VGG-11	91.1	89.4	91.6
	Resnet-18	95.3	93.1	95.3
ImageNet	VGG-19	72.3	68.5	72.4
	Renset-18	69.8	66.5	70.0
	Resnet-50	74.1	71.1	74.1

Dataset	Model	Discrete	INN	INN-init
Set5	SRCNN 3x	32.9	32.6	32.9
	EDSR 4x	32.4	32.2	32.4
Set14	SRCNN 3x	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

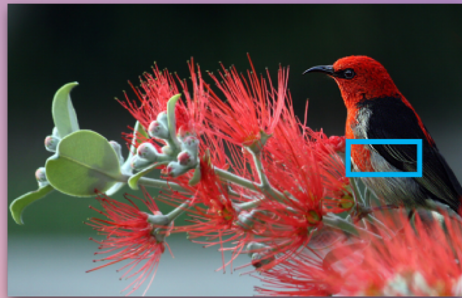
Experiments / Comparison with channel selection methods

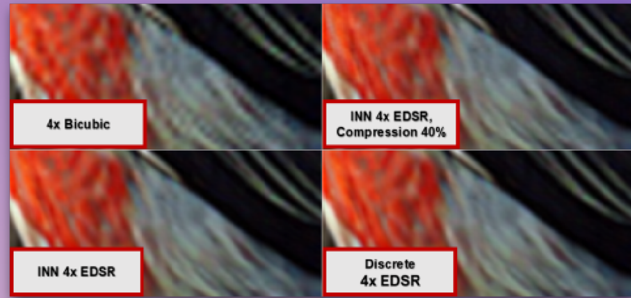
Resampling of pre-trained INNs **significantly outperforms channel selection methods** for structured pruning without fine-tuning.





Experiments / EDSR examples on DIV2K





Future steps

- 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.
- Explore other parameter permutation strategies that can improve the initialization from discrete networks and the pruning accuracy.
- 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.
- Convolutions in INNs could generate any number of channels for the output image. We propose to investigate such architectures in an **optical flow estimation** to provide a flexible sampling of intermediate frames.