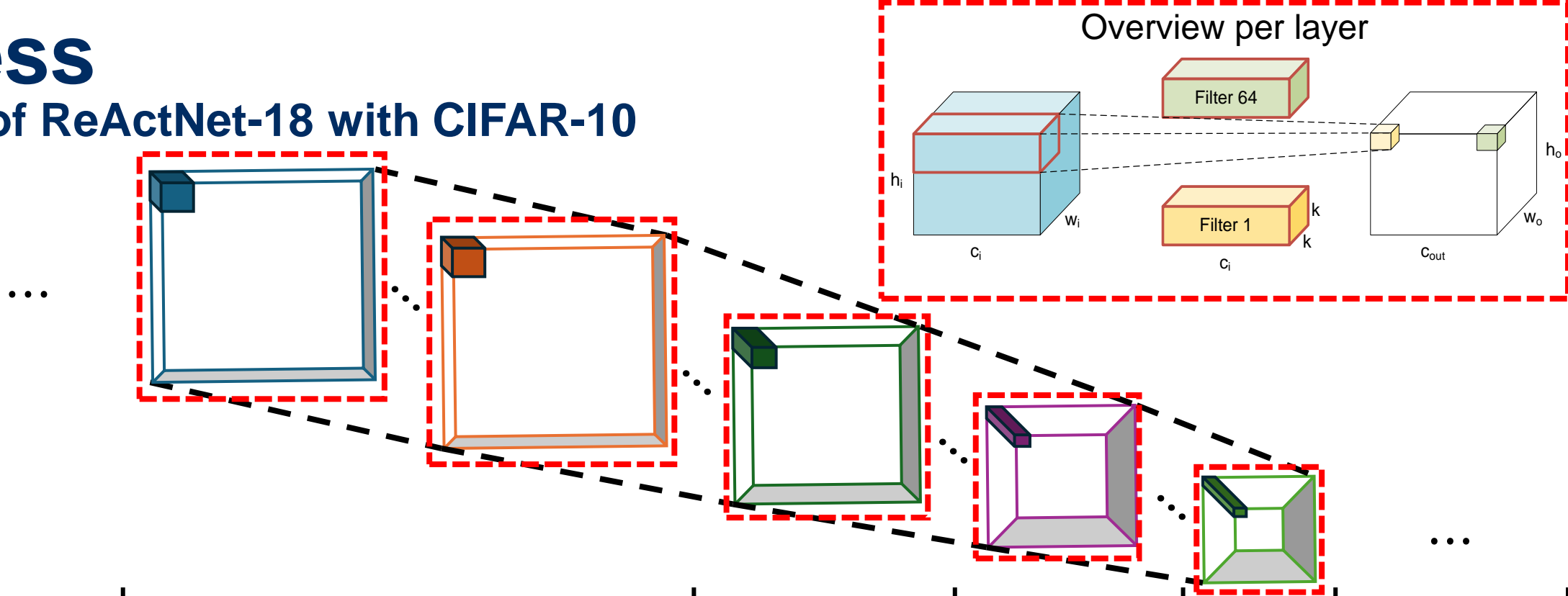


Lightweight DNN with Majority Voter

Hyungdong Park, Inguk Yeo
Department of Computer Engineering

Progress

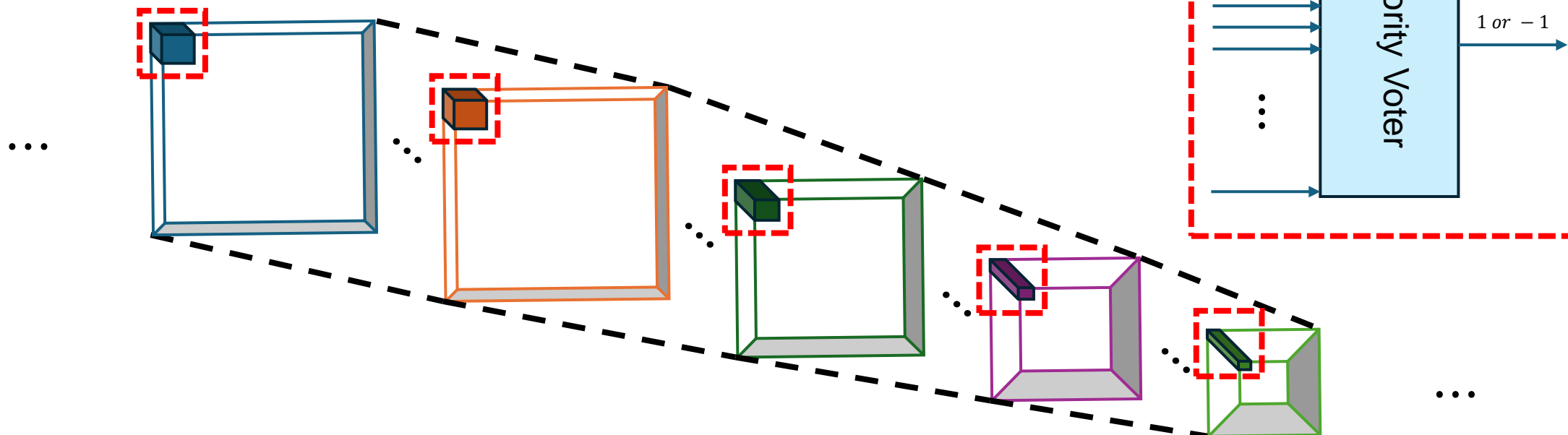
- Overview of ReActNet-18 with CIFAR-10



Num of channels	64	64	128	256	512	Pooling & FC
Operations	\otimes	XNOR & Bit-Count				
Activations and weights	\mathbb{R}	\mathbb{R} (After BN) & Binarized values (1 or -1)				\mathbb{R}
# units of XNOR		$(c_i \times k \times k) \times (c_{out} \times h_o \times w_o)$				
# units of Popcount		$c_{out} \times h_o \times w_o$				

Progress

- Overview of ours with CIFAR-10



Num of channels	64	64	128	256	512	Pooling & FC
Operations	\odot	Majority Voter				
Activations and weights	\mathbb{R}	\mathbb{R} (After BN) & Binarized values (1 or -1)				\mathbb{R}
# units of XNOR		$(c_i \times k \times k) \times (c_{out} \times h_e \times w_e) \rightarrow \text{Majority Voter}$				
# units of Popcount		$c_{out} \times h_e \times w_e \rightarrow \text{Majority Voter}$				

Progress

- Results with CIFAR-10

Models	Top-1 Accuracy (%)	Top-5 Accuracy (%)
ReActNet-18	93.380	99.800
ReActNet-18 with Majority Voter	84.930	99.250
Bi-RealNet-18	88.770	98.250
Bi-RealNet-18 with Majority Voter	30.070	79.690

Progress

- Our strategy for retraining to increase accuracy...1

- A straightforward application of the Majority Voter results in a decline in accuracy. Therefore, to achieve our goal of enhancing inference speed, additional techniques must be incorporated.
- Our Majority Voter employs the same functionality as the Sign function used in ReActNet. Hence, we will leverage techniques such as the **Straight-Through Estimator (STE)** and **ApproxSign()**

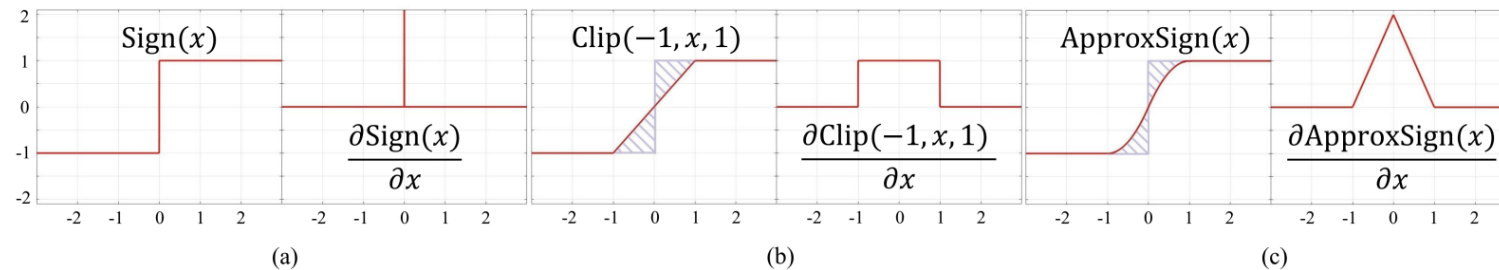
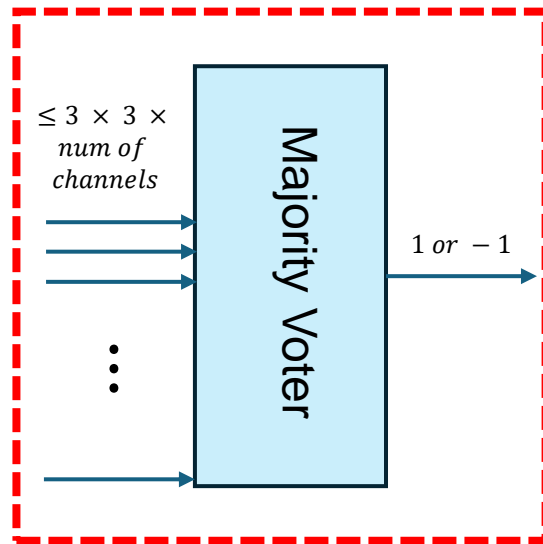


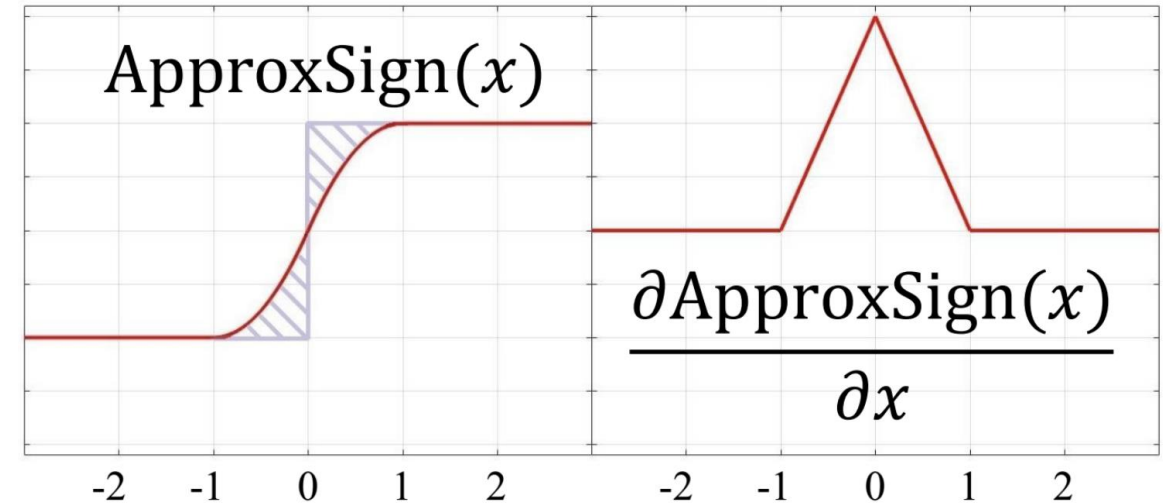
Fig. 5. (a) Sign function and its derivative, (b) Clip function and its derivative for approximating the derivative of the sign function, proposed in [7], (c) Proposed differentiable piecewise polynomial function and its triangle-shaped derivative for approximating the derivative of the sign function in gradients computation.

Progress

- Our strategy for retraining to increase accuracy...2



Foward



$$F(a_r) = \begin{cases} -1 & \text{if } a_r < -1 \\ 2a_r + a_r^2 & \text{if } -1 \leq a_r < 0 \\ 2a_r - a_r^2 & \text{if } 0 \leq a_r < 1 \\ 1 & \text{otherwise} \end{cases}, \quad \frac{\partial F(a_r)}{\partial a_r} = \begin{cases} 2 + 2a_r & \text{if } -1 \leq a_r < 0 \\ 2 - 2a_r & \text{if } 0 \leq a_r < 1 \\ 0 & \text{otherwise} \end{cases}$$

Backpropagation


Progress

- Results with CIFAR-10


Models	Top-1 Accuracy (%)	Top-5 Accuracy (%)
ReActNet-18	93.380	99.800
ReActNet-18 with Majority Voter	84.930	99.250
Our RaActNet-18	92.090	99.610
Bi-RealNet-18	88.770	98.250
Bi-RealNet-18 with Majority Voter	30.070	79.690
Our Bi-RealNet-18	87.660	98.720

Update about plans

- PyTorch Modeling

 pminhtam / xnor_conv_pytorch_extension


[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)







 **xnor_conv_pytorch_extension** Public


[Watch](#) **1** [Fork](#) **0** [Star](#) **4**



[main](#) **2 Branches** **0 Tags**

[Add file](#) [Code](#)

 **Phạm Minh Tâm** [cpp : add jit.py](#) c363da0 · 2 years ago **30 Commits**

 cpp	cpp : add jit.py	2 years ago
 cuda	cuda : dell all redundant comment	2 years ago
 py	py: if_name__	2 years ago
 .gitignore	first commit	2 years ago
 README.md	README.md : add some references	2 years ago
 main.py	first commit	2 years ago

 **README**

C++/CUDA Extensions XNOR convolution in PyTorch






XNOR extension

In XNOR convolution, both the filters and the input to convolutional layers are binary. Now, by approximating the convolution operations with XNOR and bitcounting operations, we can gain massive speed-up and memory savings.

About

XNOR-Net with binary conv2d kernels with XNOR GEMM op, support both CPU and GPU.

[cpp](#) [cuda](#) [pytorch](#) [xnor-net](#) [gemm](#) [binary-convolutions](#) [xnor-convolutions](#) [binary-neural-networks](#) [binary-op](#) [pytorch-extension](#)

 [Readme](#)  [Activity](#)  [4 stars](#)  [1 watching](#)  [0 forks](#) [Report repository](#)

Releases

No releases published

Packages

No packages published

Update about plans

- PyTorch Modeling

~~1. BNN based on XNOR and **Popcount**~~

- ~~- Implementing **actual** XNOR and Popcount operations within hardware (GPU) using PyTorch and CUDA~~

~~2. BNN based on XNOR and **Majority Voter**~~

- ~~- Apply majority voter to standard convolution.~~

3. BNN based on XNOR and **Hierarchical Majority Voter**

- Example) M512 \sim M4 (M128, M128, M128, M128)

Thank you

Progress

- Appendix

Table 2. Comparison of the top-1 accuracy between the three variants (i.e., BN, w/o BN, BN-Free) of binary networks on CIFAR-10 and CIFAR-100. All networks are modified from ResNet-18 except for ReActNet-A, which is constructed from MobileNetv1.

Binary Network	CIFAR-10 (%)			CIFAR-100 (%)		
	BN	w/o BN	BN-Free	BN	w/o BN	BN-Free
XNORNet-18	90.21	71.75	79.67	65.35	45.30	53.76
Bi-RealNet-18	89.12	71.30	79.59	63.51	47.72	54.34
ReActNet-18	92.31	90.33	92.08	68.78	62.60	68.34
ReActNet-A	82.95	77.60	83.91	50.30	39.37	55.00