Algorithm-Accelerator Co-Design for Deep Learning Specialization

Zhiru Zhang

School of ECE, Cornell University

csl.cornell.edu/~zhiruz

EMC^2 Workshop @ NeurIPS, 12/13/2019

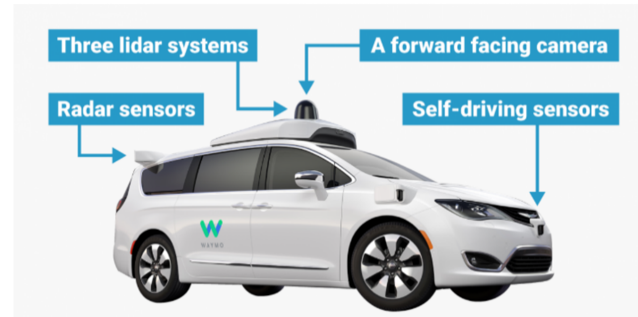
Cornell University

CSL

Need for Efficient Deep Learning







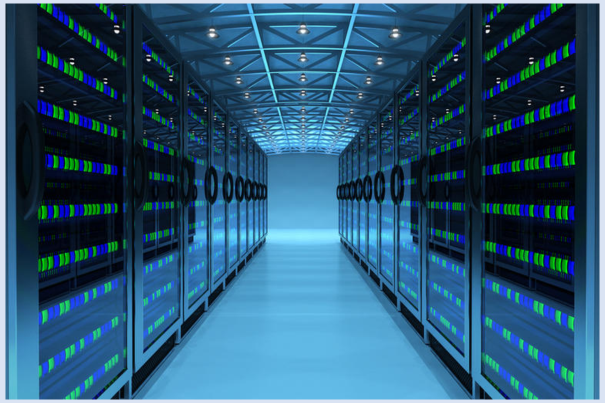
- Strong demand for **faster** DNNs with **better energy efficiency**

Specialized DNN Processors are Ubiquitous

Mobile

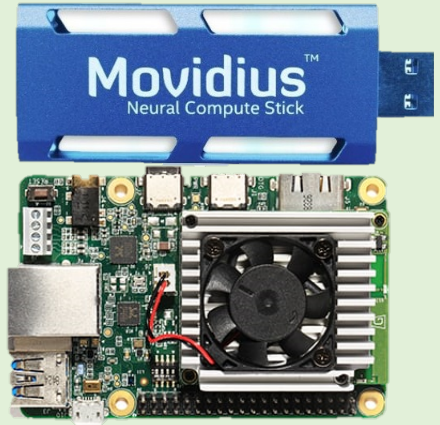
Apple (A12)
Samsung (Exynos 9820)
Huawei (Kirin 970)
Qualcomm (Hexagon)

Cloud



Google (TPU)
Microsoft (Brainwave)
Xilinx (EC2 F1)
Intel (FPGAs, Nervana)
AWS Offerings

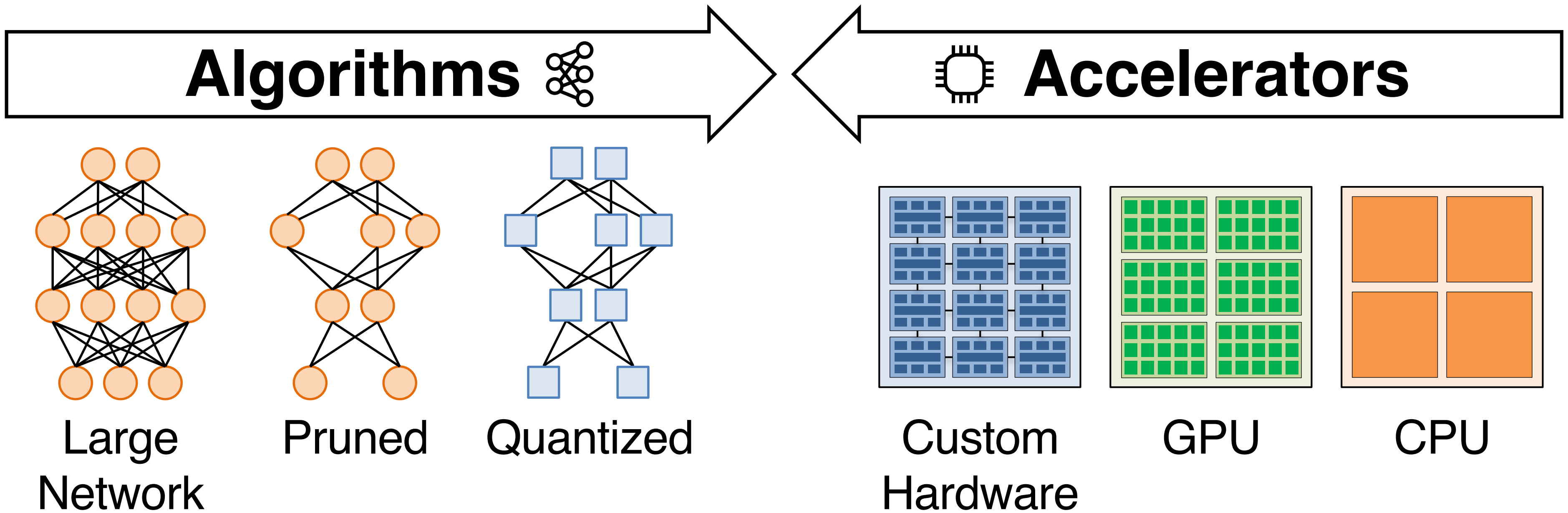
Embedded



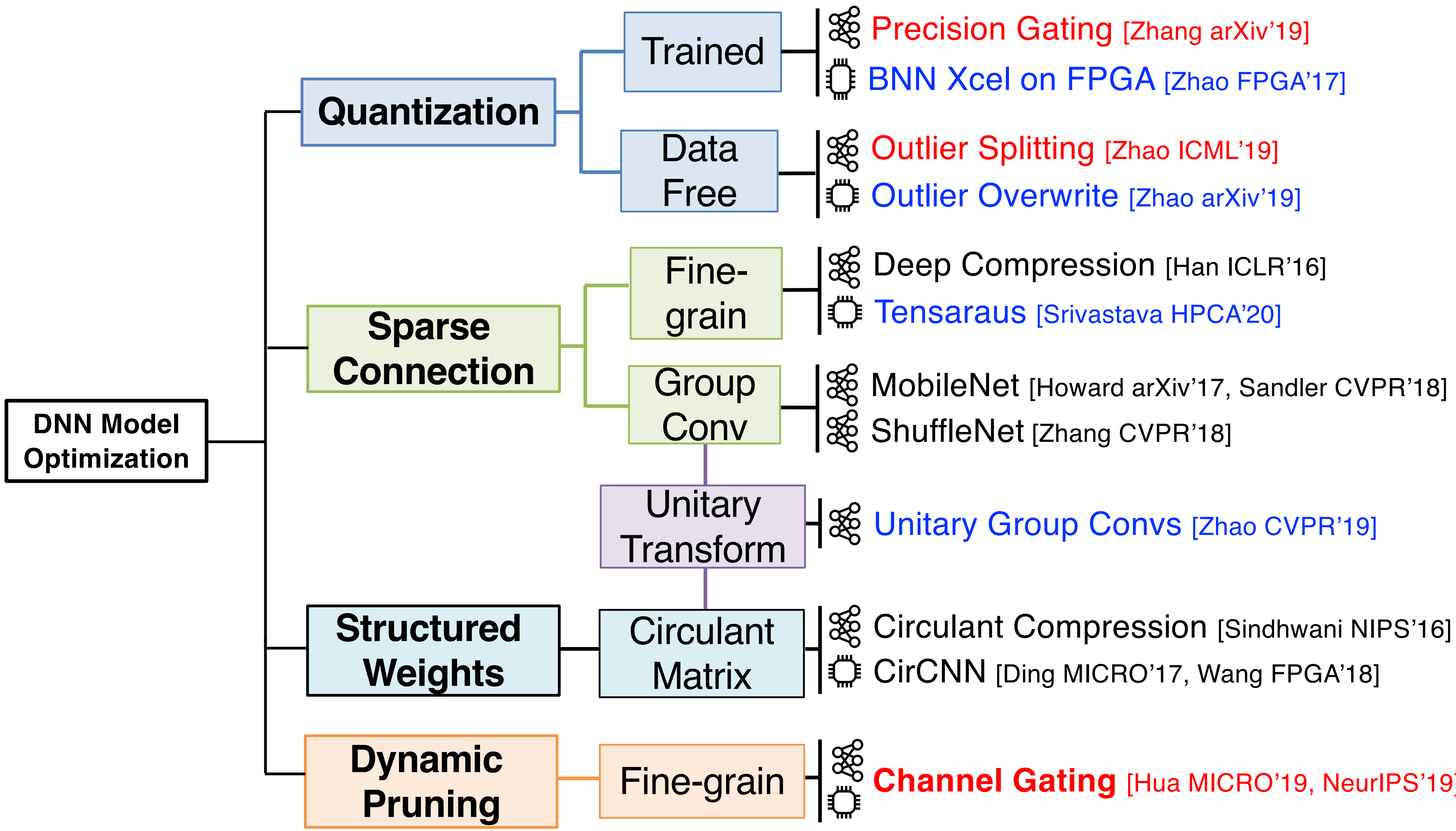
Google (Edge TPU)
Intel (Movidius)
Deephi/Xilinx (Zynq)
ARM (announced)
Many Startups

Co-Design for Deep Learning Specialization

- DNN performance and efficiency remain a big challenge
- Specialization necessitates algorithm and hardware co-design



Topics of this Talk



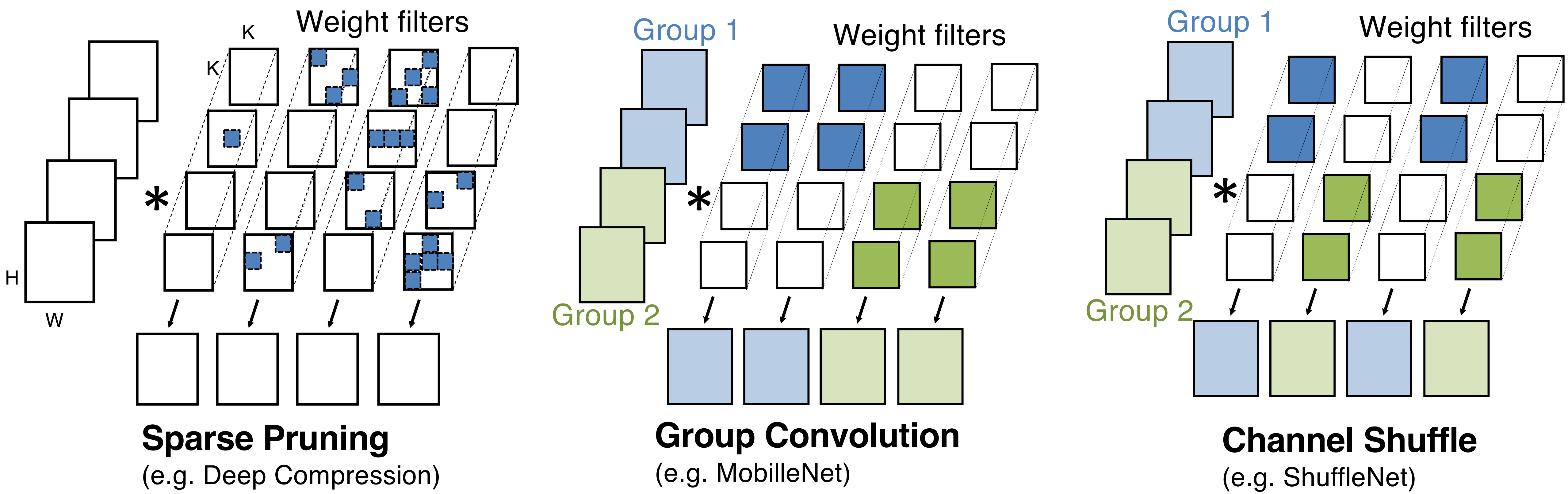
Channel Gating Neural Networks

Exploiting Dynamic Sparsity

Channel Gating Neural Networks
Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, Edward G. Suh
Conference on Neural Information Processing Systems (NeurIPS), December 2019

Boosting the Performance of CNN Accelerators with Dynamic Fine-Grained Channel Gating
Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, Edward G. Suh
International Symposium on Microarchitecture (MICRO), October 2019

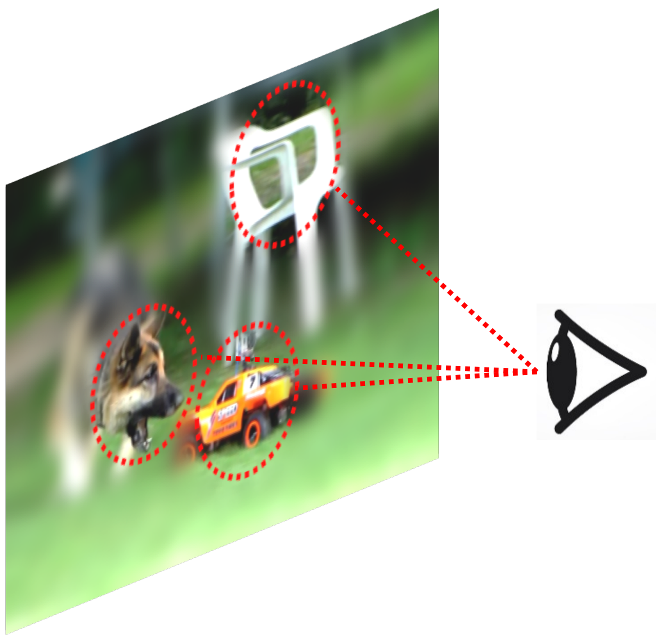
Static Pruning for DNNs



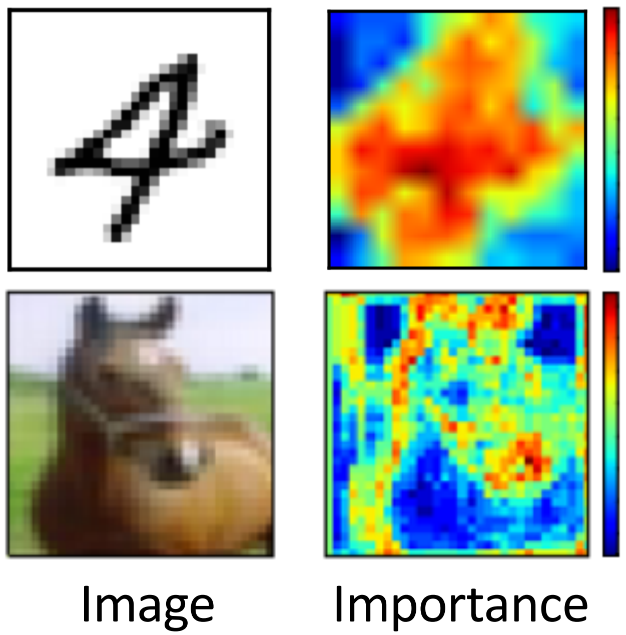
- Sparse pruning (fine-grain) makes the model unstructured
- Group conv (coarse-grain) often incurs nontrivial accuracy loss
- **Inference time of pruned model is agonistic to the actual input**

Dynamic Pruning using Input-Dependent Properties?

Human visual recognition focuses on salient regions



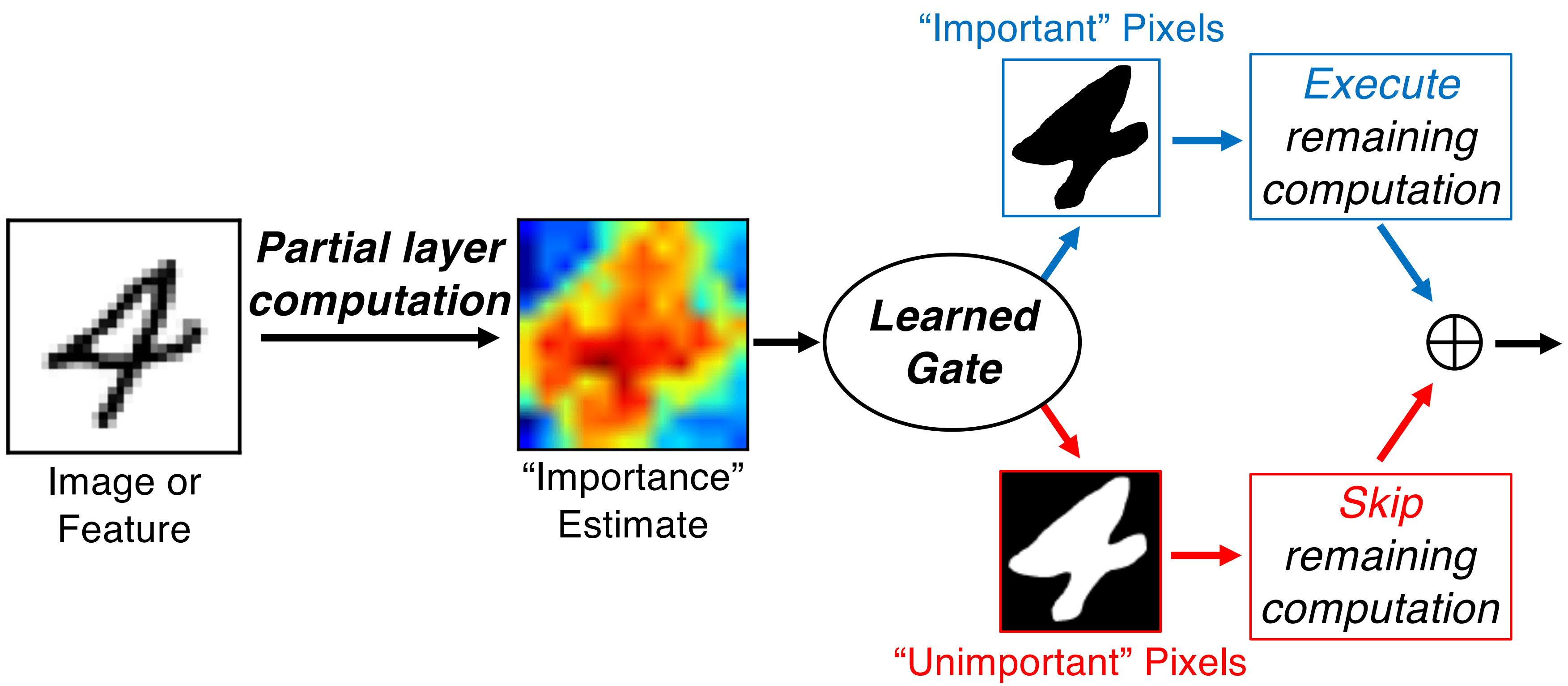
Basic Idea: dynamically reduce compute effort in unimportant regions of input features



Channel Gating for CNNs: High-Level Ideas

For each pixel in an output feature map:

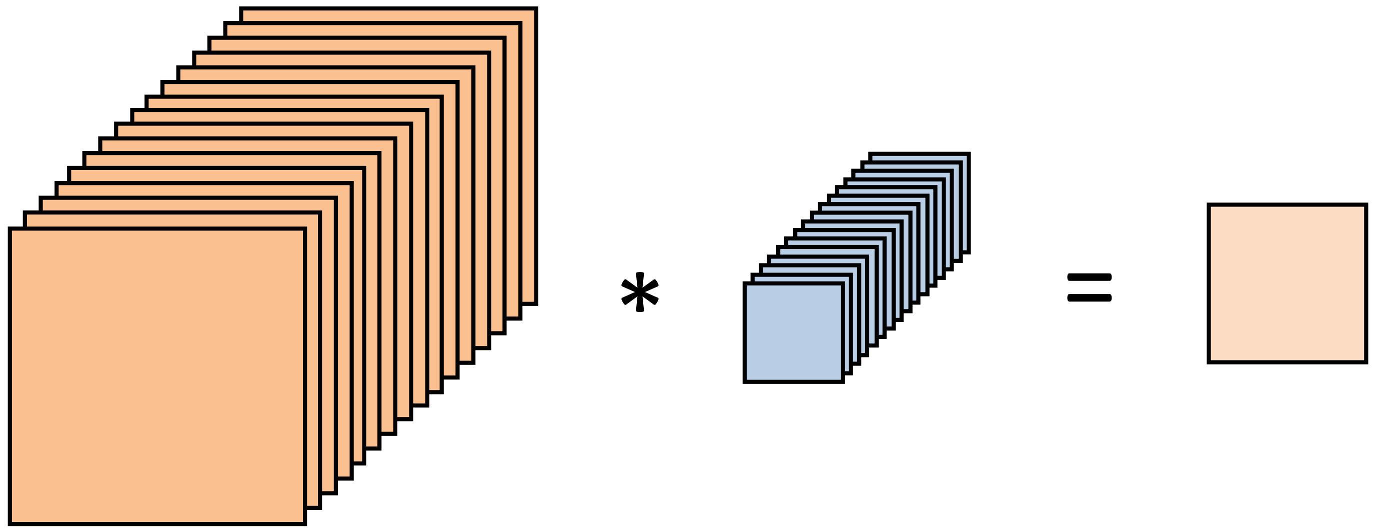
1. Estimate its "importance" from partial layer computation
2. Skip remaining computation if deemed "unimportant"



Dynamic pruning exploits input-dependent characteristics
Fined-grained pruning avoids overly aggressive compression that degrades accuracy

Rationale of Using Partial Layer Computation

- Approximating output features with partial sums
  - Partial sum and final convolution result are often highly correlated in CNNs



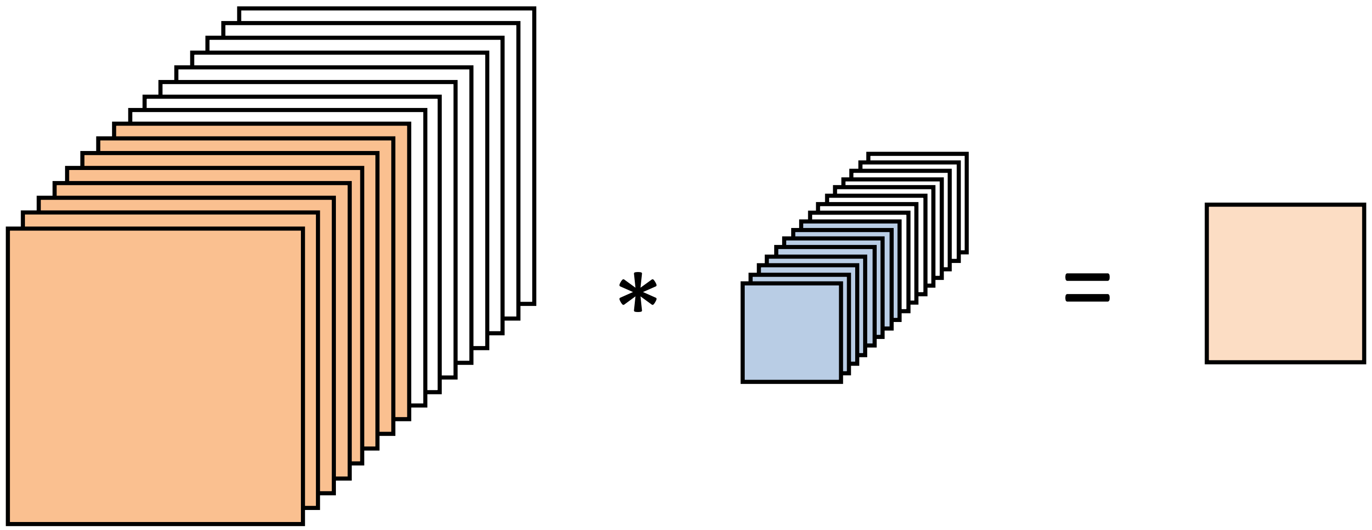
p = 100% input channels = c

partial sum = $\sum_{k=0}^{p} w^k x^k$; final sum = $\sum_{k=0}^{c} w^k x^k$

Correlation(partial sum, final sum) = **1.00**

Rationale of Using Partial Layer Computation

- Approximating output features with partial sums
  - Partial sum and final convolution result are often highly correlated in CNNs



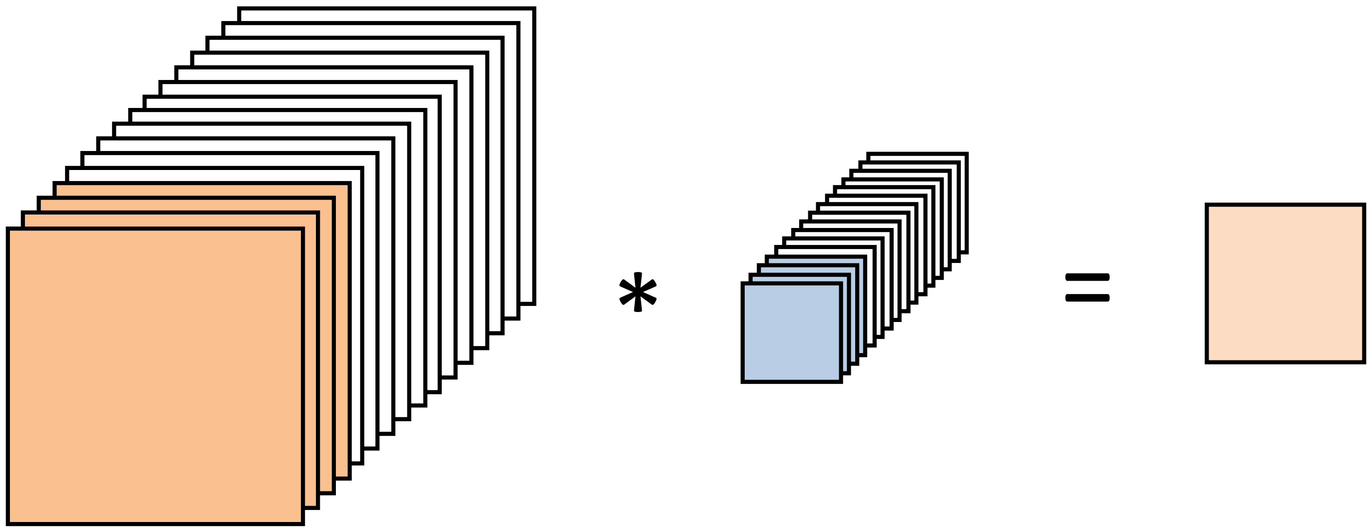
p = 50% input channels

partial sum = $\sum_{k=0}^{p} w^k x^k$; final sum = $\sum_{k=0}^{c} w^k x^k$

Correlation(partial sum, final sum) = **0.86**

Rationale of Using Partial Layer Computation

- Approximating output features with partial sums
  - Partial sum and final convolution result are often highly correlated in CNNs



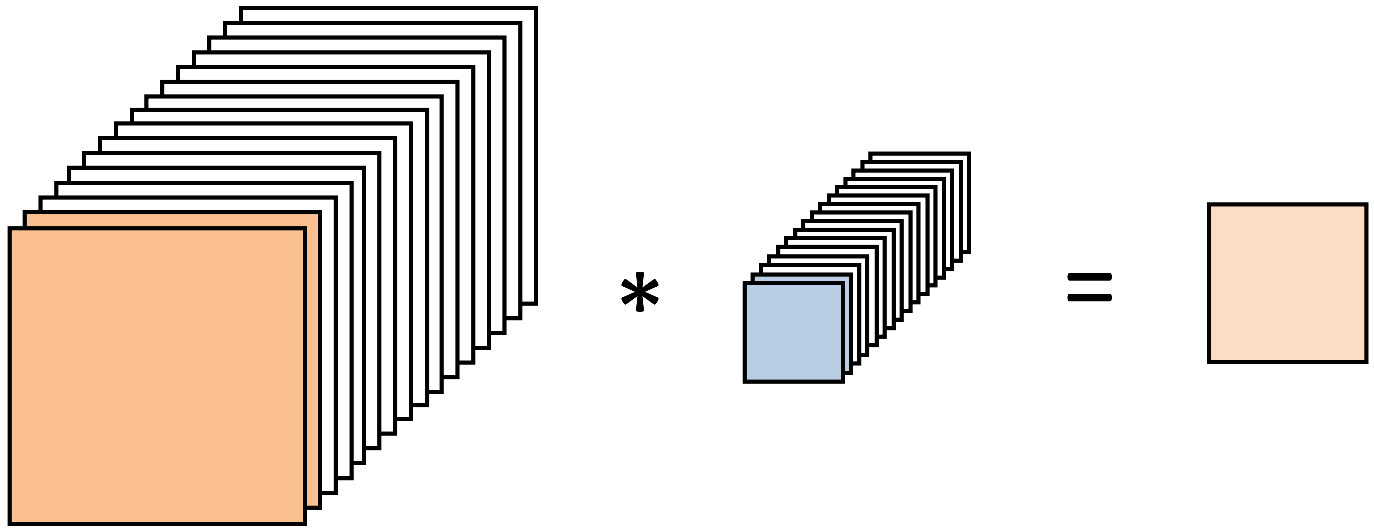
p = 25% input channels

partial sum = $\sum_{k=0}^{p} w^k x^k$; final sum = $\sum_{k=0}^{c} w^k x^k$

Correlation(partial sum, final sum) = **0.72**

Rationale of Using Partial Layer Computation

- Approximating output features with partial sums
  - Partial sum and final convolution result are often highly correlated in CNNs



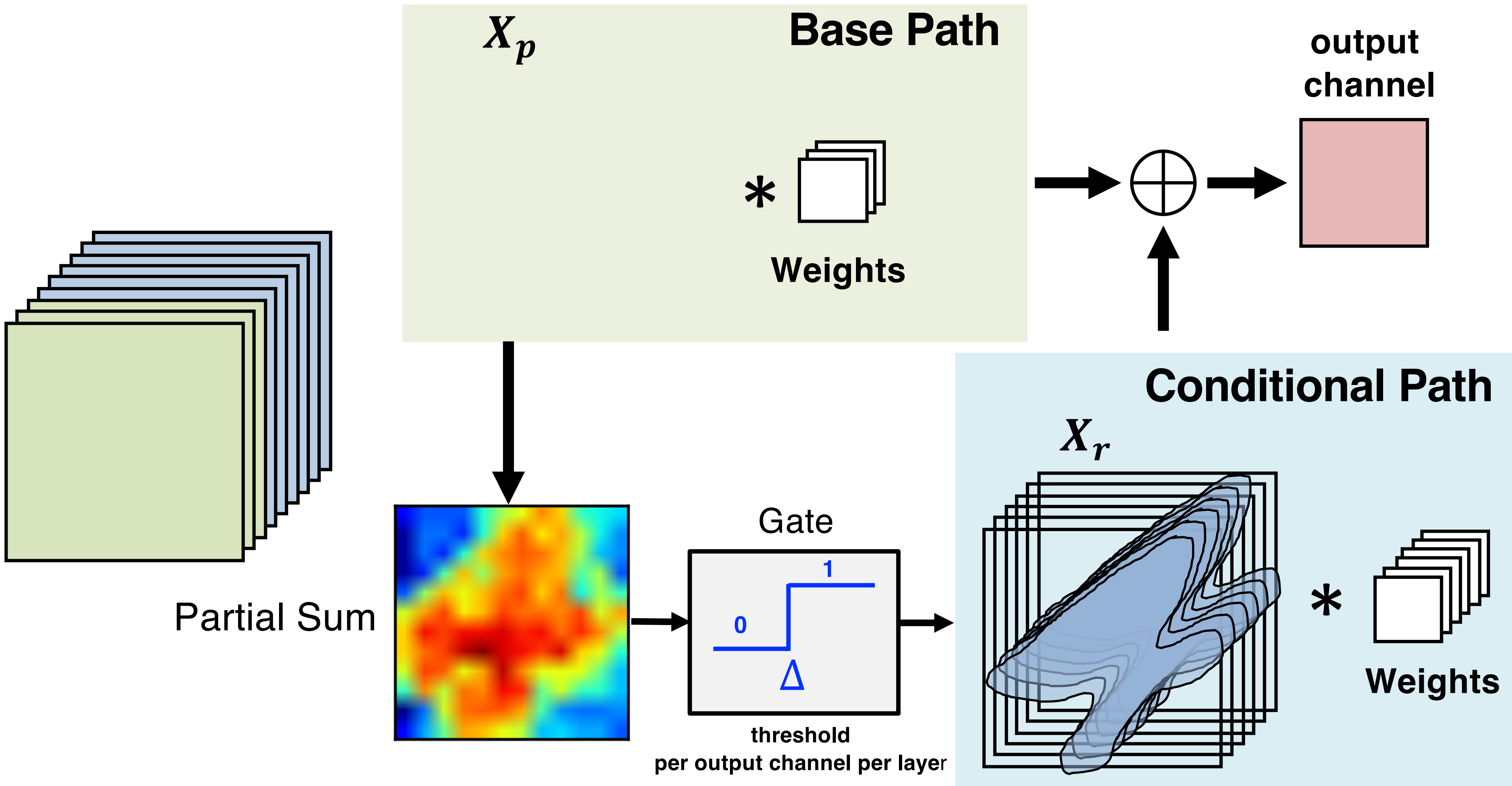
p = 12.5% input channels

partial sum = $\sum_{k=0}^{p} w^k x^k$; final sum = $\sum_{k=0}^{c} w^k x^k$

Correlation(partial sum, final sum) = **0.56**

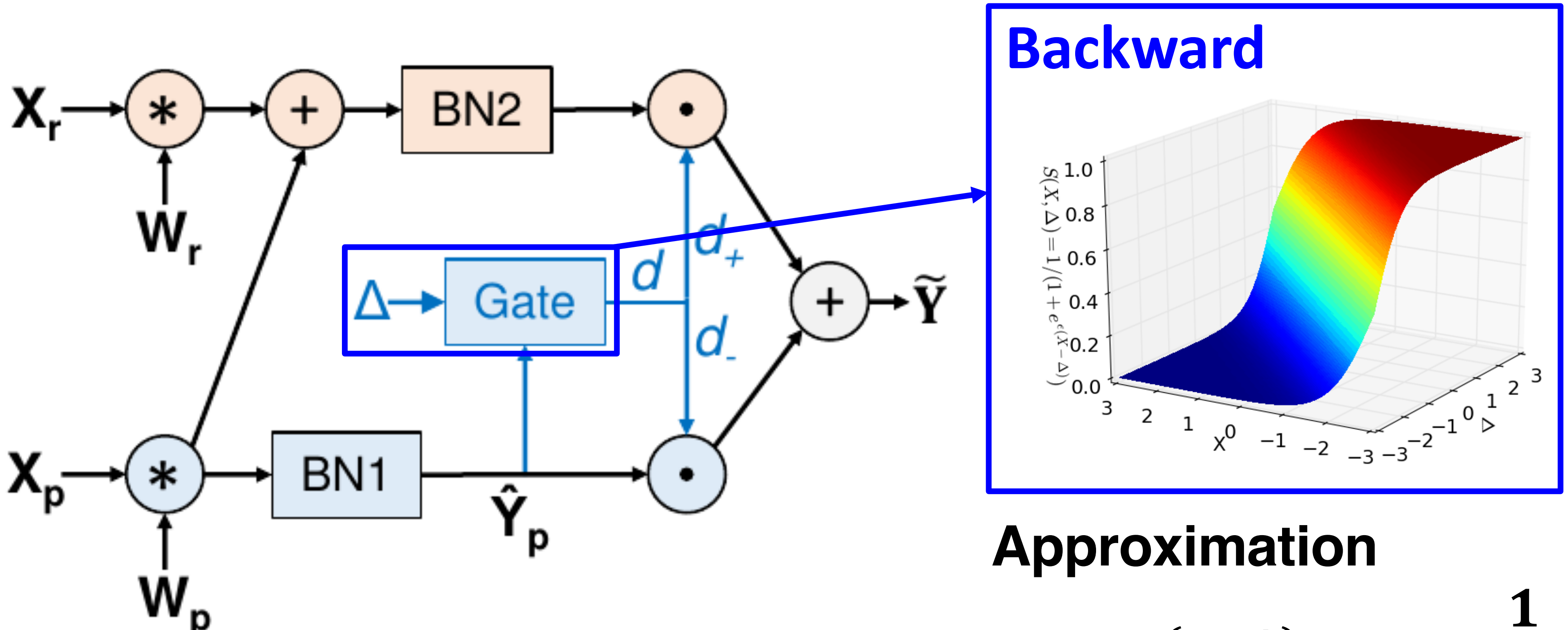
Conv Layer with Channel Gating

1. Obtain partial sum by performing convolution over the first *p* input channels (**base path**)
2. The **gate** outputs a binary decision by comparing partial sum with a **learnable threshold**
3. Skip convolution over remaining *r* channels if decision = 0 (**conditional path**)



Training Channel Gating Networks

- **Single-pass training** to learn effective gating policy
  - Each building block in the training computation graph must be differentiable



Extended computation graph of one conv layer for training

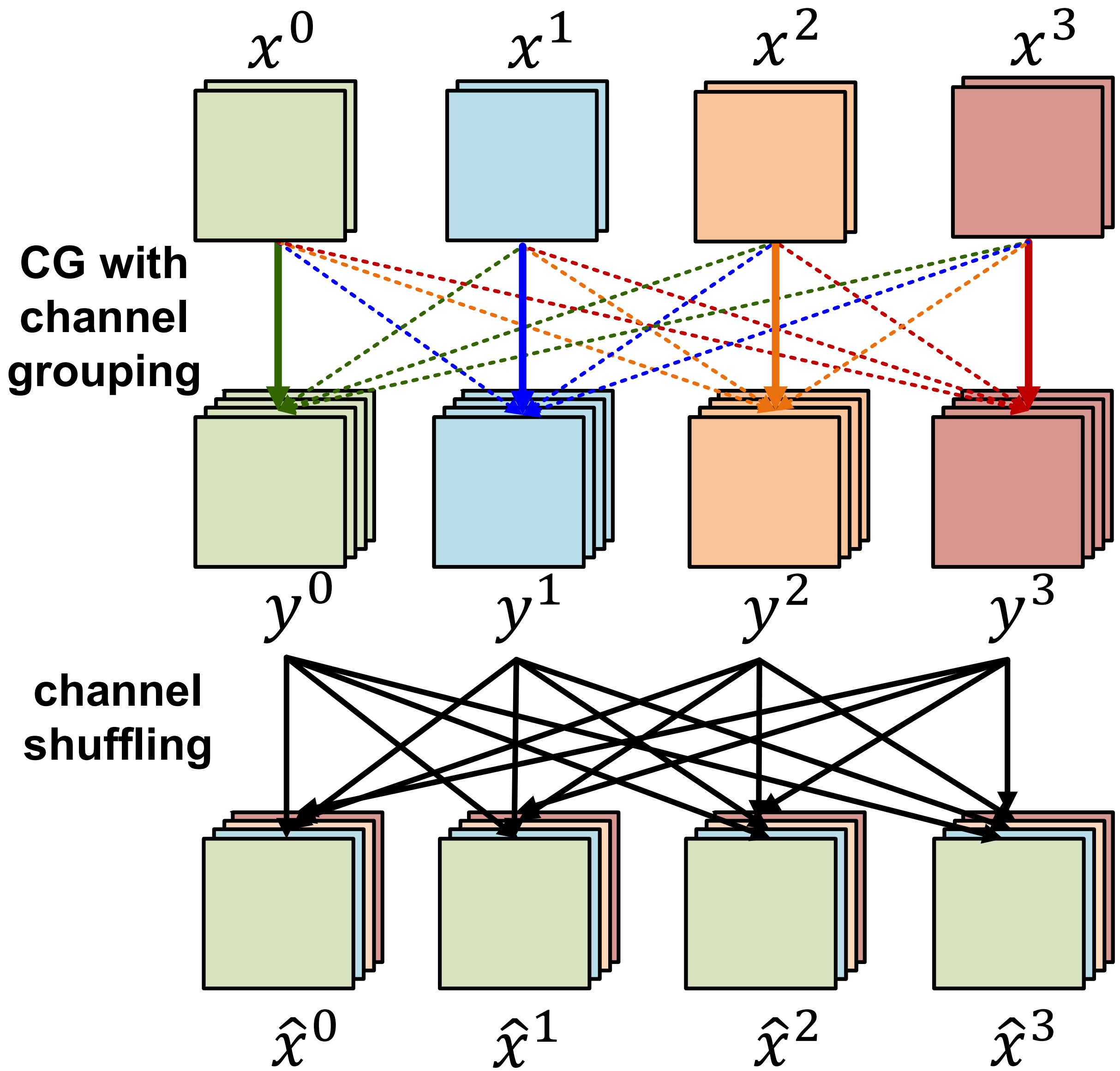
Approximation

$$S(X,\ \Delta) = \frac{1}{1 + e^{\varepsilon(X-\Delta)}}$$

Results on CIFAR-10

ResNet-18	Base path Fraction	Test Error	FLOPs Saved
Baseline	1	5.4%	-
Channel Gating	1/8	5.44%	**81.8%** **(5X+ reduction)**
Channel Gating	1/16	5.96%	**87.4%** **(~8X reduction)**

Channel Gating Composes with Grouping and Shuffling



For output group y^k, input group x^k is on the base path and the rest is conditional

Shuffle grouping for the next layer

More Results on CIFAR-10

- Channel gating applies to a variety of DNN models

Baseline	# of Groups	Target Threshold	Top-1 Error Baseline (%)	Top-1 Error Pruned (%)	Top-1 Accu. Drop (%)	FLOP Reduction
ResNet-18	8	2.0	5.40	5.44	0.04	5.49×
	16	3.0	5.40	5.96	0.56	7.95×
Binary VGG-11	8	1.0	16.85	16.95	0.10	3.02×
	8	1.5	16.85	17.10	0.25	3.80×
VGG-16	8	1.0	7.20	7.12	-0.08	3.41×
	8	2.0	7.20	7.59	0.39	5.10×
MobileNetV1	8	1.0	12.15	12.44	0.29	2.88×
	8	2.0	12.15	12.80	0.65	3.80×

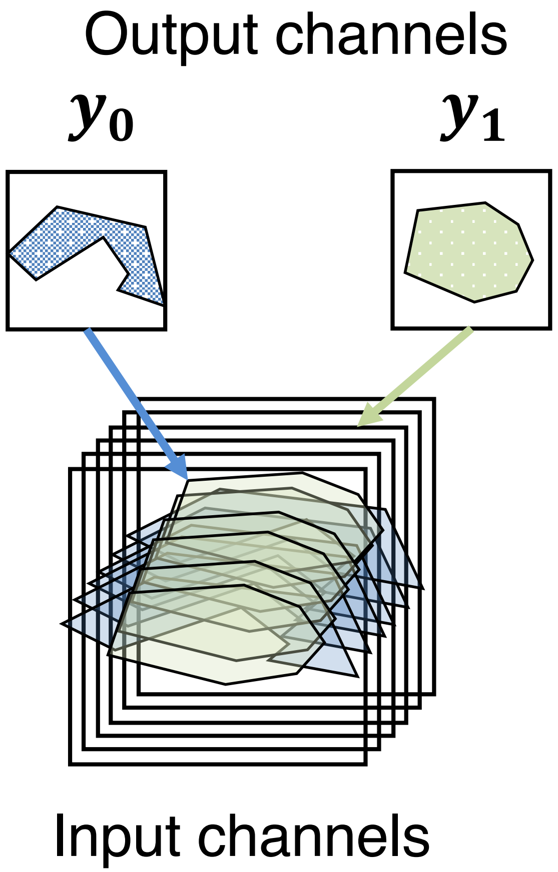
Results on ImageNet

AlexNet	Dynamic	Test Error (Top 5)	FLOP Reduction
Baseline	/	19.4%	1x
SnaPEA (ISCA'18)	Y	30.4%	2.11x
Channel Gating	Y	**20.0%**	**2.65**x

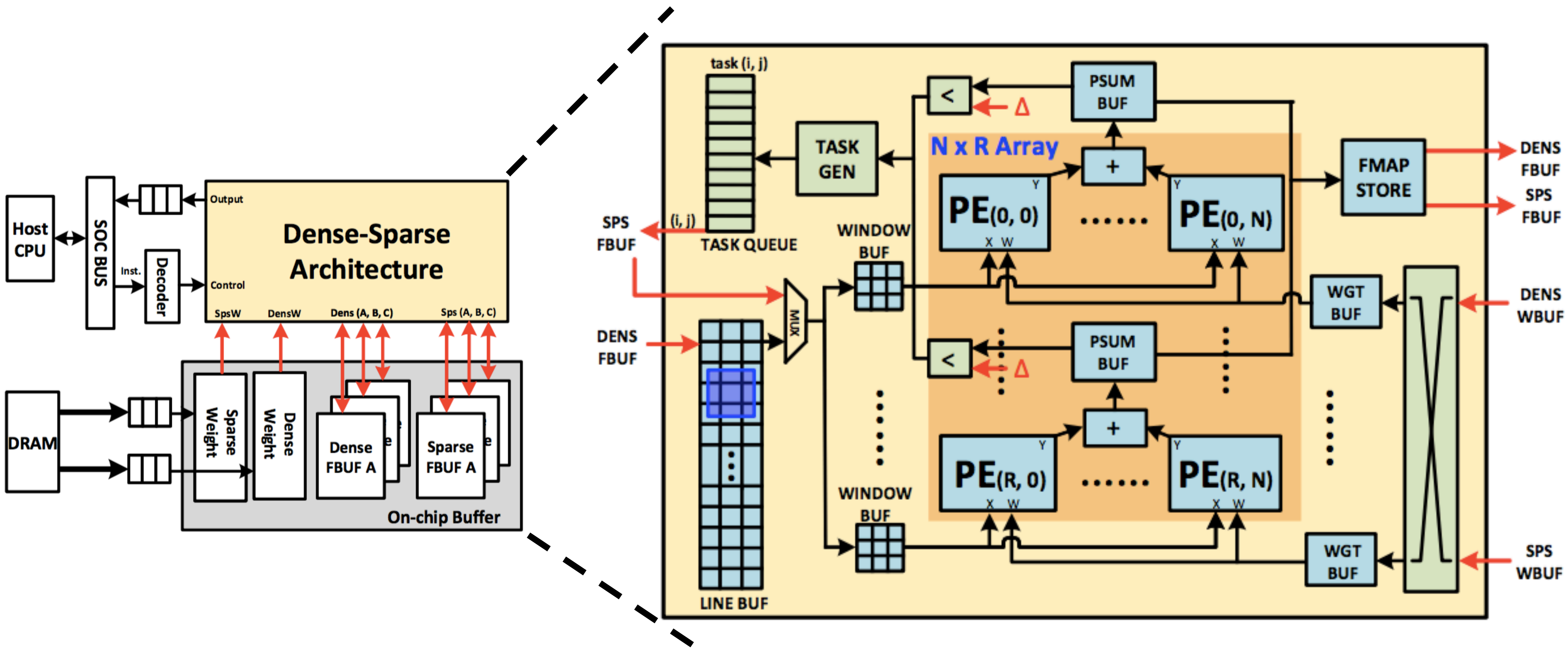
ResNet-18	Dynamic	Test Error (Top 1)	FLOP Reduction
Baseline	/	30.8%	1x
Soft Filter Pruning (IJCAI'18)	N	32.9%	1.72x
Network Slimming (ICCV'17)	N	32.8%	1.39x
Collaborative Layers (CVPR'17)	Y	33.7%	1.53x
Discrimination-aware Pruning (NIPS'18)	N	32.7%	1.85x
Channel Gating	Y	**31.7%**	**2.03**x
Channel Gating + KD	Y	**31.0%**	**2.82x**

Sampled Feature Convolution in Conditional Path

- Compute in conditional path is sparse
  - Output activations are sparse
  - Their spatial locations vary dynamically
- But regularity is preserved along channel dimension
  - Per output activation, the input channels in the conditional path ($\boldsymbol{X_r}$) are either used altogether or entirely skipped



Accelerator Architecture for Channel Gating Networks (CGNet)

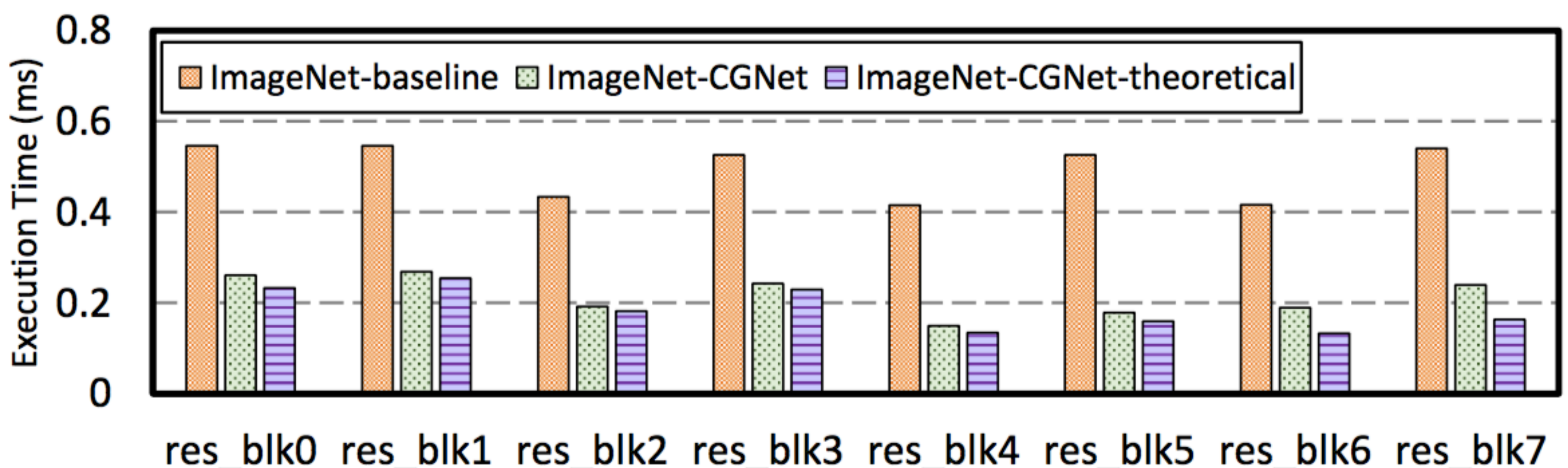


- Base (**dense**) & conditional (**sparse**) paths reuse the same systolic array
- The whole accelerator is designed in **HLS** C++ (by two PhD students)

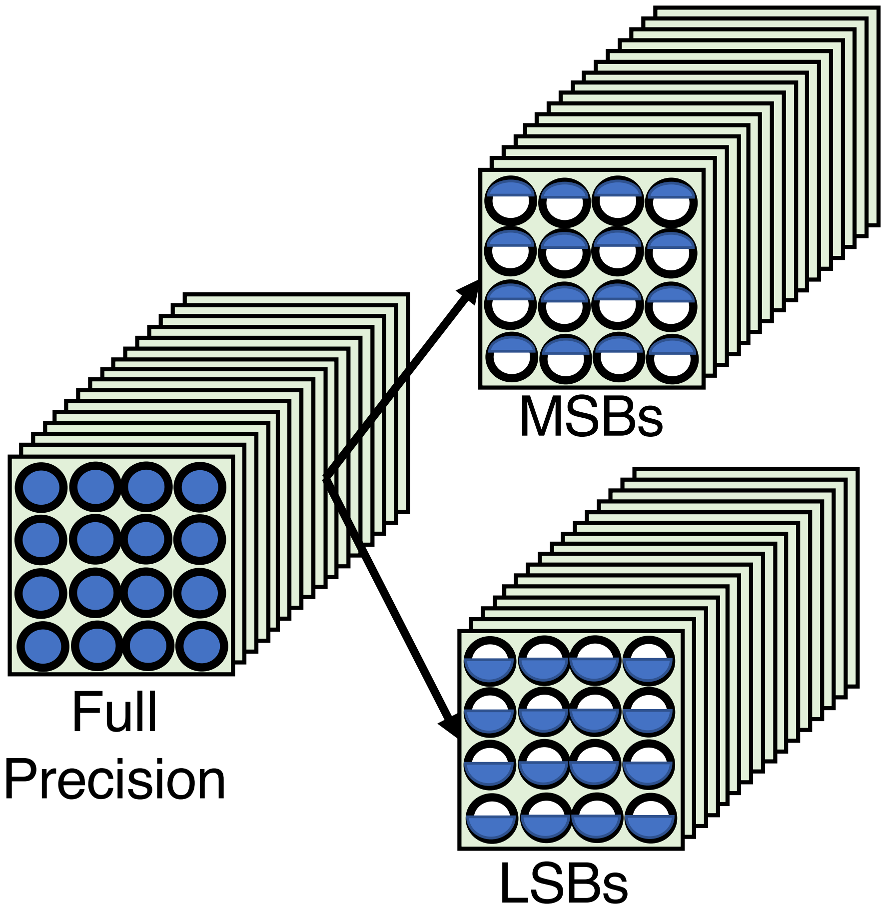
Preliminary ASIC Evaluation (8-bit ResNet-18 on ImageNet)

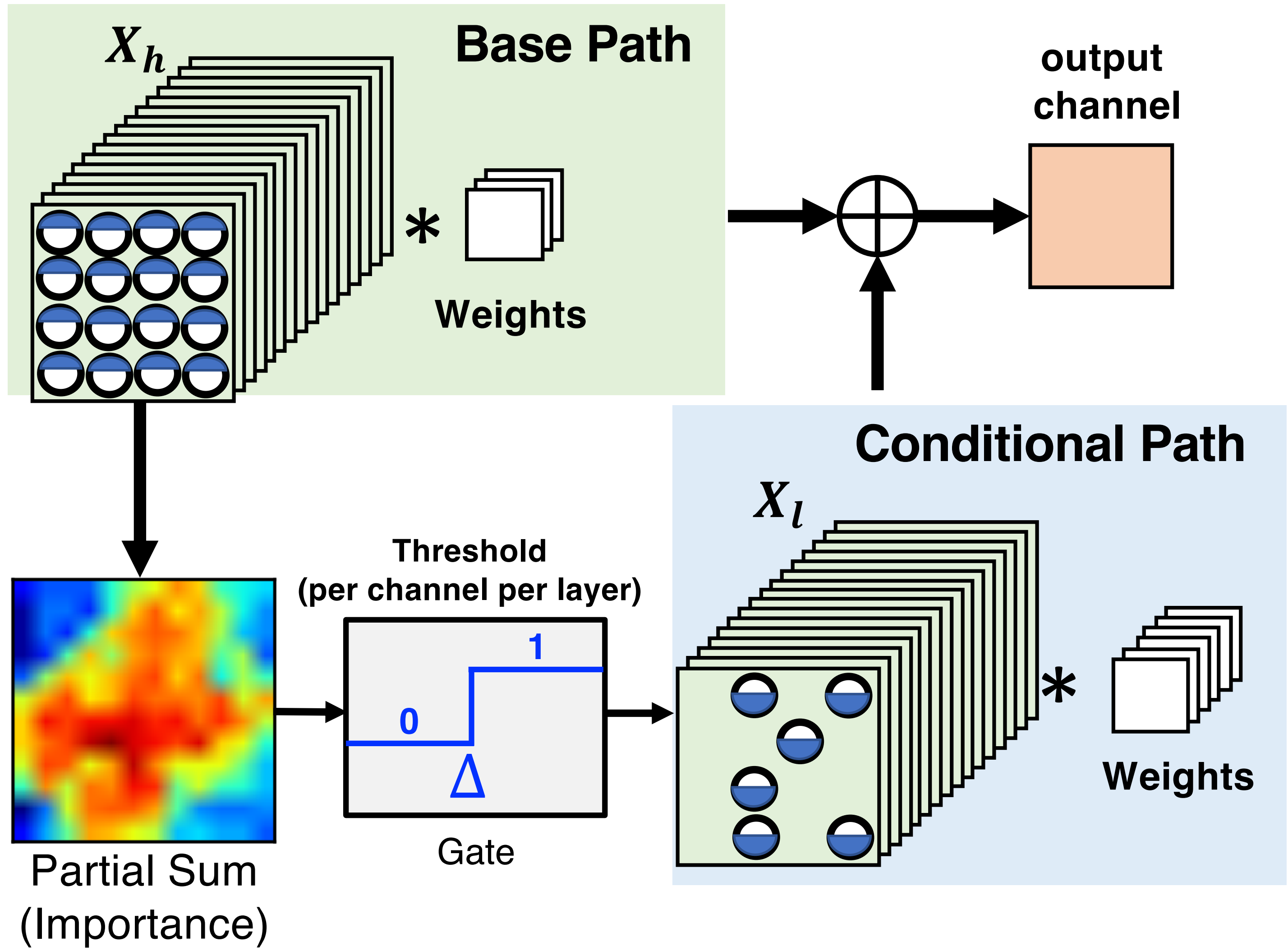
Platform	ASIC		Nvidia GTX 1080Ti	Intel i7 7700k
	Baseline	CGNet		
Freq. (MHz)	800	800	1923	4200
Power (Watt)	0.202	0.256	225	91
ImageNet Throughput (fps)	253.8	580.6	1563.7	13.8
Energy/frame (mJ)	0.796	0.441	143.9	6594.2

CGNet is 2.3× faster with 1.8× higher energy efficiency compared to a baseline accelerator w/o dynamic pruning (~20% area overhead)



Ongoing Work: Dynamic Quantization with Precision Gating





Precision Gating (PG): Preliminary Results

	Ours				Baselines						
	Precision Gating					UQ	PACT	Fix-Threshold			
	B/B_{hb}	Sp.	B_{avg}	Acc	Bits	Acc.	Acc.	B/B_{hb}	Sp.	B_{avg}	Acc.
ShiftNet-20	5/3	**55.5**	**3.9**	**89.1**	8	89.1	89.0	5/3	48.8	4.0	74.3
CIFAR-10	5/3	**96.3**	**3.1**	**88.6**	4	87.3	87.5	5/3	67.8	3.6	67.0
(fp 89.4%)	3/1	**71.9**	**1.6**	**84.5**	2	77.8	82.9	3/1	10.1	2.8	64.3
ResNet-18	4/3	**78.2**	**3.2**	**91.7**	8	91.6	91.2	4/3	58.7	3.4	88.3
CIFAR-10	3/2	**90.1**	**2.1**	**91.2**	4	91.1	90.9	3/2	71.0	2.3	74.2
(fp 91.7%)	2/1	**71.5**	**1.3**	**90.6**	2	84.0	90.1	2/1	21.6	1.8	71.9
ShuffleNet	6/4	**57.2**	**4.8**	**59.7**	8	59.1	59.1	6/4	52.6	4.9	33.6
ImageNet	6/4	**62.2**	**4.7**	**59.3**	6	57.8	57.1	6/4	58.5	4.8	32.7
(fp 59.0%)	5/3	**41.9**	**4.1**	**58.0**	5	57.0	56.6	5/3	40.4	4.2	27.7

- Comparing PG against prediction-based execution (PBE) [Song et al. ISCA'18]
  - PBE does zero prediction with fixed threshold
  - $B_{avg} = B_{hb} + (1 - Sp\%) \times (B - B_{hb})$
- Using a similar bitwidth, PG is 25+% more accurate than PBE on ShuffleNet for ImageNet

Outlier Channel Splitting

Improving DNN Quantization without Re-training

Improving Neural Network Quantization without Retraining Using Outlier Channel Splitting

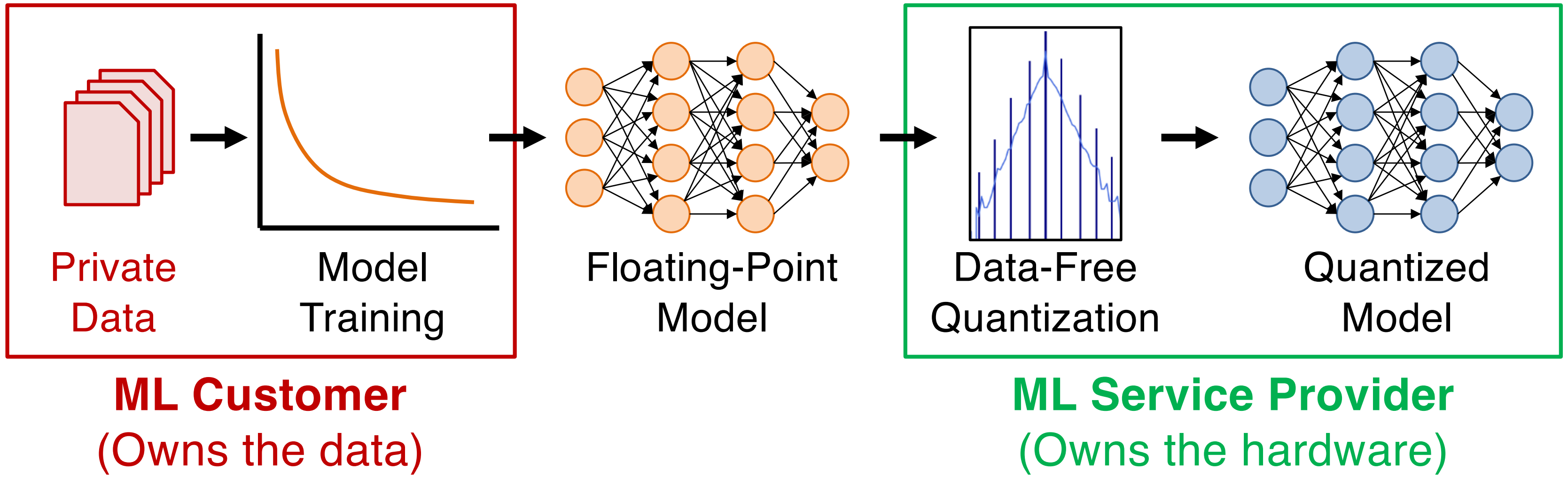
Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, Zhiru Zhang

International Conf. on Machine Learning (ICML), June 2019

https://github.com/cornell-zhang/dnn-quant-ocs

Quantization without Training Data

- DNN quantization techniques that require training are often discouraged by the current ML service model



- Reasons to prefer **data-free quantization**:
  1. ML providers typically cannot access customer training data
  2. Customer is using a pre-trained **off-the-shelf model**
  3. Customer is unwilling to retrain a **legacy model**
  4. Customer **lacks the expertise** for quantization training

The Outlier Problem

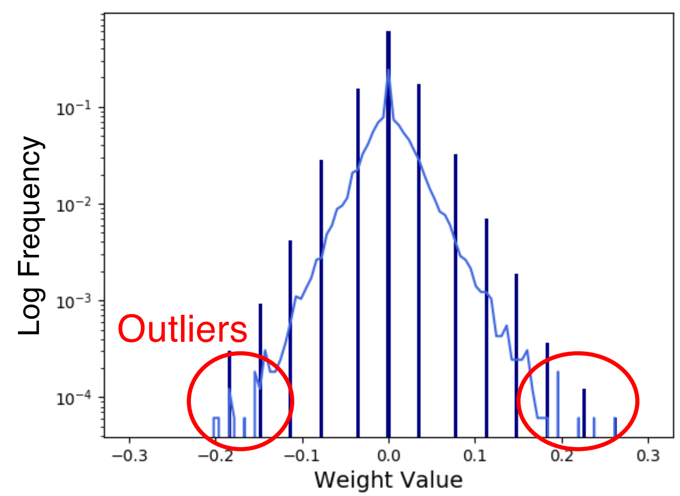
- DNN weights and activations are distributed in a bell curve that peaks near zero
  - Most values are close to zero, rare *outliers* are large

Uniform Quantizer

$$Q(x) = \text{round}\left(\frac{x(2^{k-1}-1)}{\max(|x|)}\right)\frac{\max(|x|)}{2^{k-1}-1}$$

– Grid points extend to $\max(|x|)$

– Outliers stretch the quantization grid, resulting in poor resolution

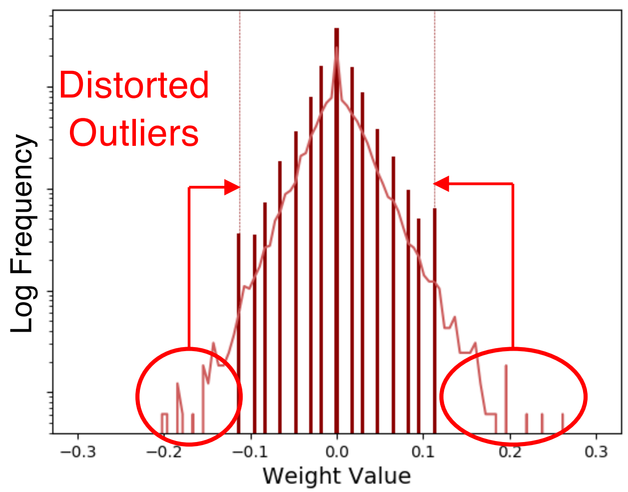


Prior Arts on Addressing Outliers

Clipping

Sung *arXiv'15*, Shin *ICASSP'16*, Migacz *GTC'17*, Banner *arXiv'18*

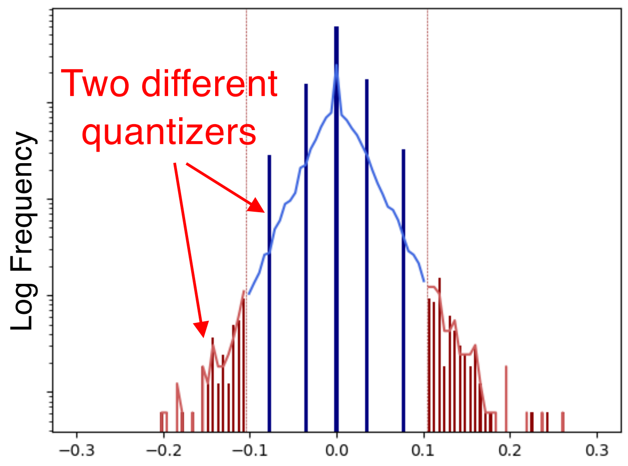
+ Reduces quantization noise
+ Used in industry solutions (TensorRT)
– Distorts outliers, accuracy loss



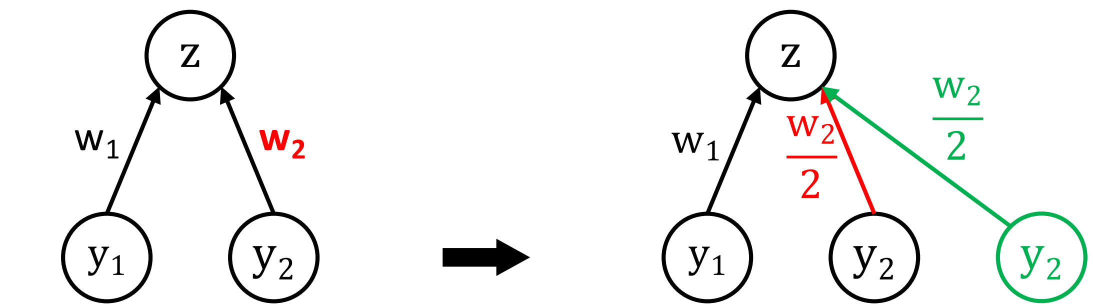
Outlier-Aware Quantization

Park *ECCV'18*, Park *ISCA'18*

+ Reduces quantization noise
+ Preserves outlier values
– Requires additional sparse hardware

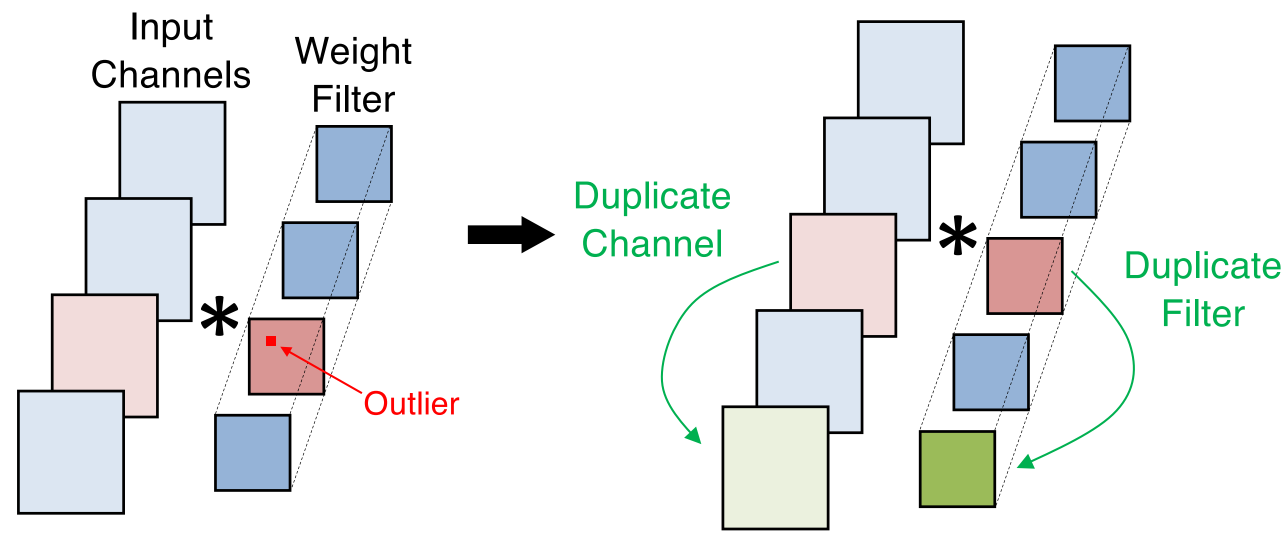


Our Proposal: Outlier Channel Splitting (OCS)



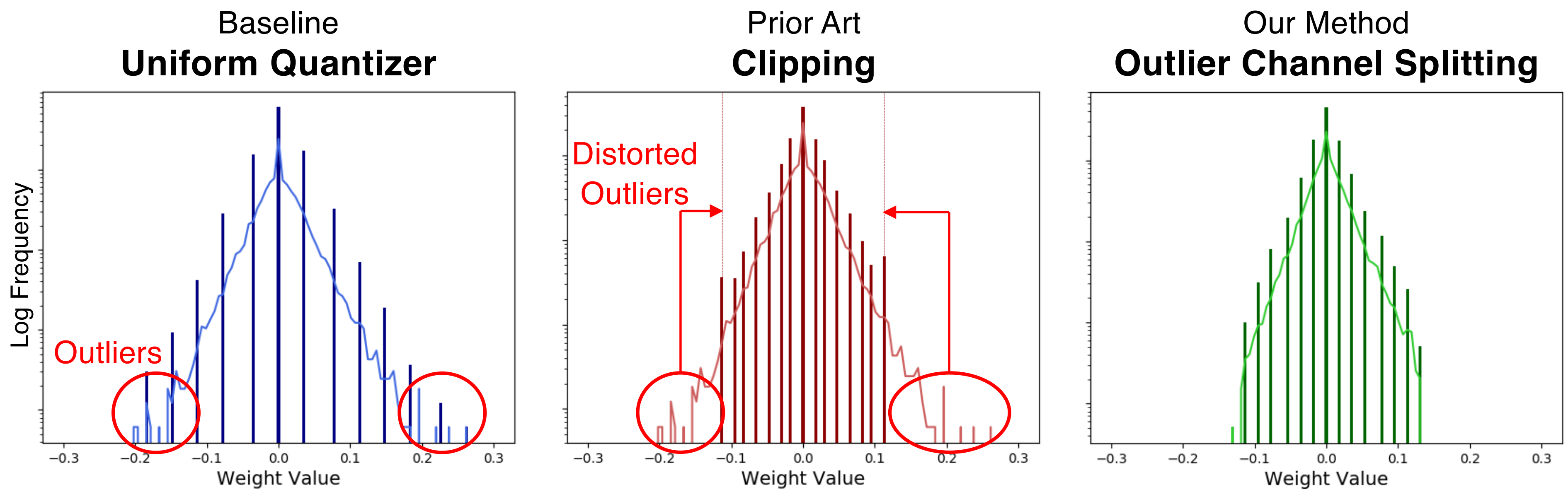
$z = w_1 y_1 + \mathbf{w_2 y_2}$

$z = w_1 y_1 + \frac{w_2}{2} y_2 + \frac{w_2}{2} y_2$



- OCS splits outlier weights, halving their values
  - **The network remains functionally equivalent**
  - But affected outliers are moved toward center of the distribution
  - Example: Duplicate node y_2 to halve the weight w_2

OCS vs. Prior Arts



– Poor quantizer resolution due to outliers

+ Reduces quantization noise
+ Used in NVIDIA TensorRT
– Distorts outliers

+ Reduces quantization noise
+ Removes outliers
– Model size overhead

- Improves quantization without retraining
- Outperforms existing methods with negligible size overhead (<2%) in both CNNs and RNNs
- Applies to both commodity CPUs/GPUs and custom accelerators

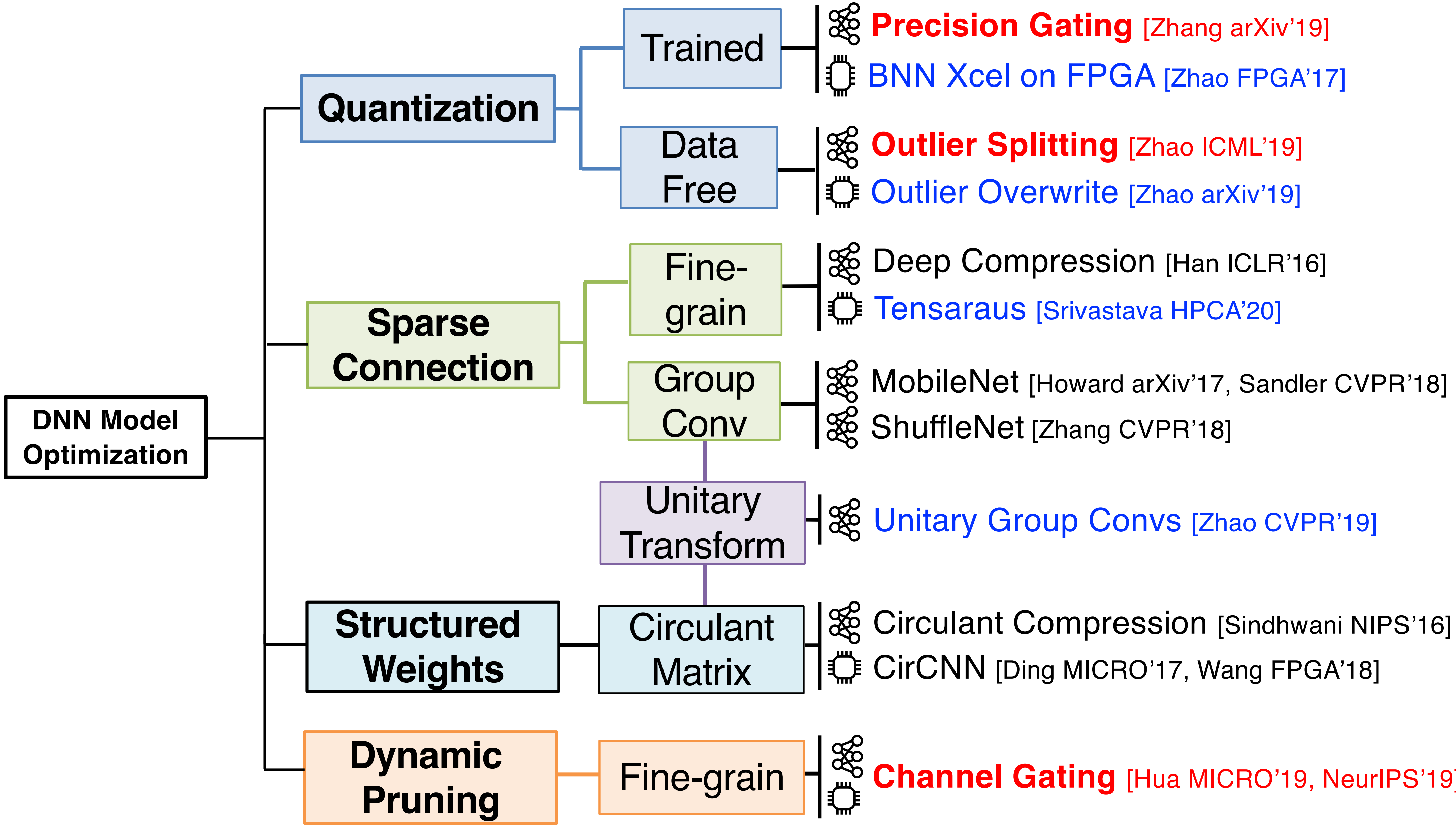
OCS Results on CNN Weights

Network (Float Acc.)	Wt. Bits	Clip None	Clip MSE	Clip ACIQ	Clip KL	Clip Best	OCS + Clip 0.01	OCS + Clip 0.02
VGG-16 BN (73.4)	6	70.8	**71.3**	71.2	63.2	71.3	71.8	71.8
	5	63.1	**66.9**	61.2	62.7	66.9	68.8	69.5
	4	0.2	53.5	34.2	**59.4**	59.4	63.8	63.8
ResNet-50 (76.1)	6	72.9	73.5	**74.3**	71.6	74.3	74.8	74.8
	5	14.5	69.1	**69.9**	69.4	69.9	71.0	71.9
	4	0.1	45.0	33.2	**62.9**	62.9	66.2	67.1
DenseNet-121 (74.4)	6	71.0	**71.4**	71.1	60.7	71.4	73.2	73.1
	5	46.9	**65.4**	61.4	54.6	65.4	70.0	70.7
	4	0.4	33.3	25.2	**42.6**	42.6	52.7	56.5
Inception-V3 (75.9)	6	58.3	**66.2**	62.3	63.0	66.2	70.5	71.7
	5	0.5	30.4	29.6	**40.5**	40.5	57.0	60.0
	4	0.1	0.2	0.1	**1.6**	1.6	2.1	2.3

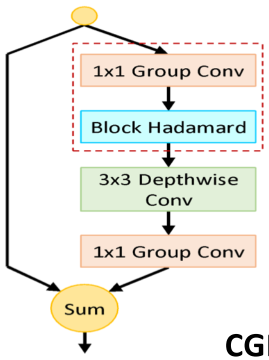
Best clip method is **bolded**

Blue = +1% or better vs. clip

Co-Design of DNN Algorithm & Hardware Yields Highest Efficiency



Related Research Efforts in My Group at Cornell



Application & Programming

Efficient Machine Learning

CGNet (NIPS'19), **HadaNet** (CVPR'19), **OCS** (ICML'19), **GraphZoom** (arXiv'19)

Intermediate Language

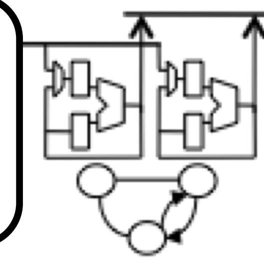
HeteroCL (FPGA'19), **T2S-Tensor** (FCCM'19)

```
// algorithm spec based on TVM
diff = hcl.compute(train.shape, [train],
lambda i, j: train[i][j]^test)
// quantization
hcl.downsize(diff, hcl.uint(49))
```

Compiler & CAD

Automatic Synthesis & Tuning

PRIMAL (DAC'19), **ASSIST** (DAC'19), **QuickEST** (FCCM'19) **LAMDA** (FCCM'19)



Hardware Design

Custom Accelerators

Tensaurus (HPCA'20), **Celerity** (IEEEMICRO'18)

Algorithms → ← **Accelerators**

Co-evolution of efficient ML and agile hardware design is generating a host of exciting research opportunities