Synetgy: Algorithm-hardware Co-design for ConvNet Accelerators on Embedded FPGAs

Yifan Yang[1,2], Qijing Huang[1], Bichen Wu[1], Tianjun Zhang[1], Liang Ma[3], Giulio Gambardella[4], Michaela Blott[4], Luciano Lavagno[3], Kees Vissers[4], John Wawrzynek[1], and Kurt Keutzer[1]

[1]University of California, Berkeley, [2]Tsinghua University,
[3]Politecnico di Torino, and [4]Xilinx Research Labs

Outline

- Introduction
- ConvNet Design
- Hardware Accelerator Design
- Experimental Results
- Conclusion

Embedded Computer Vision

Applications

Robots

Drones

Autonomous Vehicles

Security cameras

Mobile phones

CV Kernels/Tasks

Image Classification

Object Detection

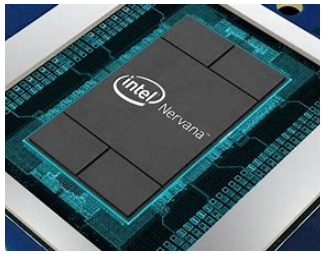
Semantic Segmentation

Embedded Platforms

CPU

GPU

FPGA



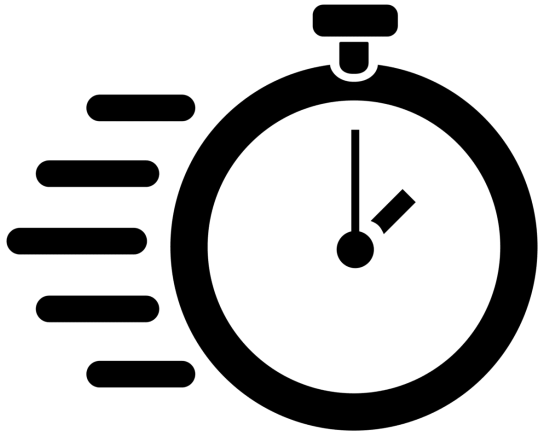
ASIC

Goals for Embedded CV

Accuracy

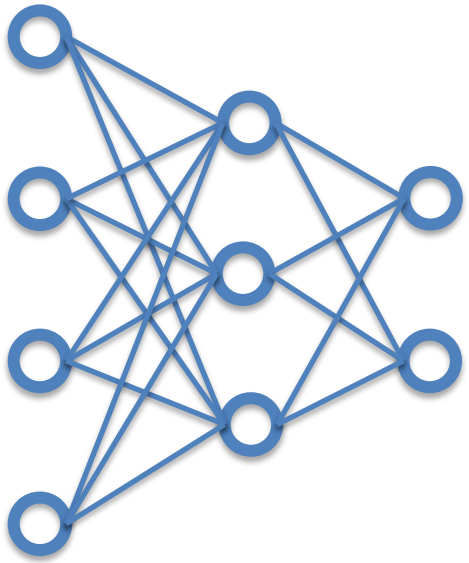
- Essential metric for applications like security cameras and autonomous vehicles

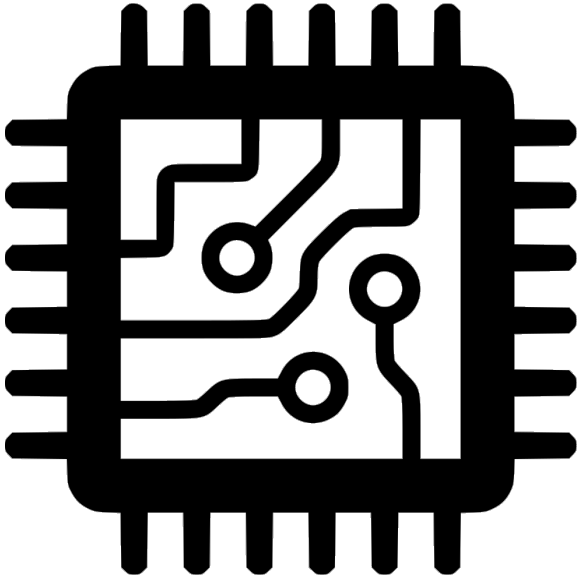
Efficiency



- Inference speed and power consumption constrain the deployment of CV tasks

How to improve accuracy and efficiency?



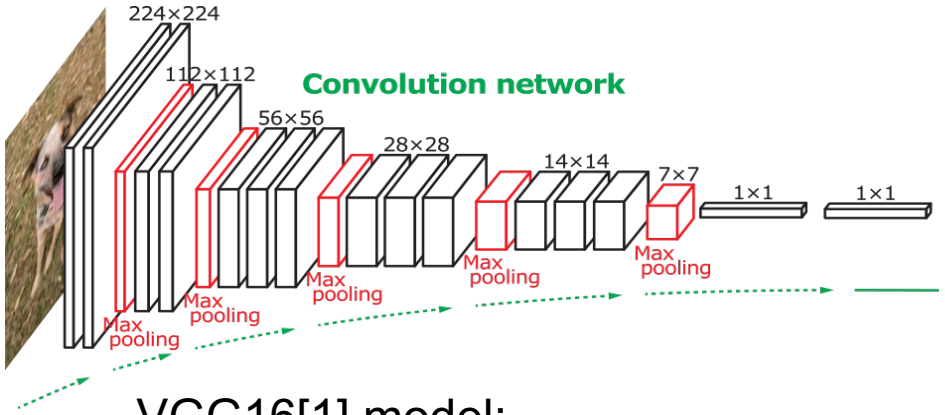


Design better ConvNet

Design better hardware

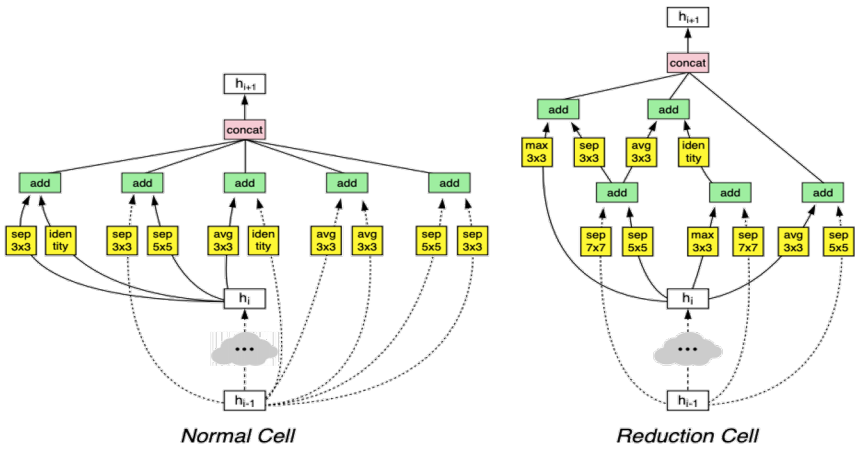
ConvNet Design

- CV community has evolved ConvNets for good accuracy – efficiency has been less important
- Efficiency proxies have been FLOPs and model size, ignoring hardware friendliness



VGG16[1] model:

- Parameter size: 552 MB
- Computation: 15.8 GOPs/image



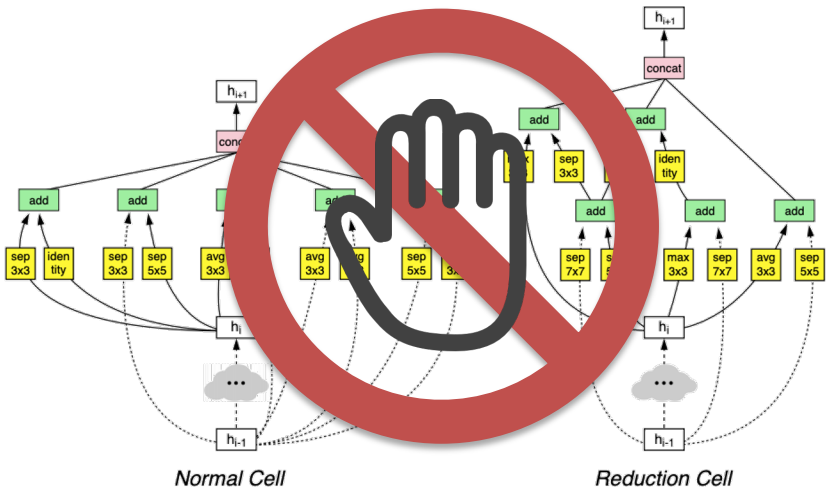
NasNet[2] model:

- Parameter size: 5.3 MB
- Computation: 1.28 GOPs/image

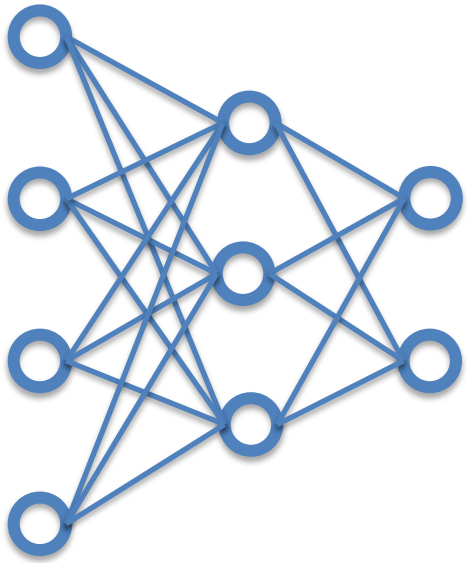
[1] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[2] Zoph, B., Vasudevan, V., Shlens, J. and Le, Q.V. Learning Transferable Architectures for Scalable Image Recognition. *arXiv e-prints*. 1707-7012.

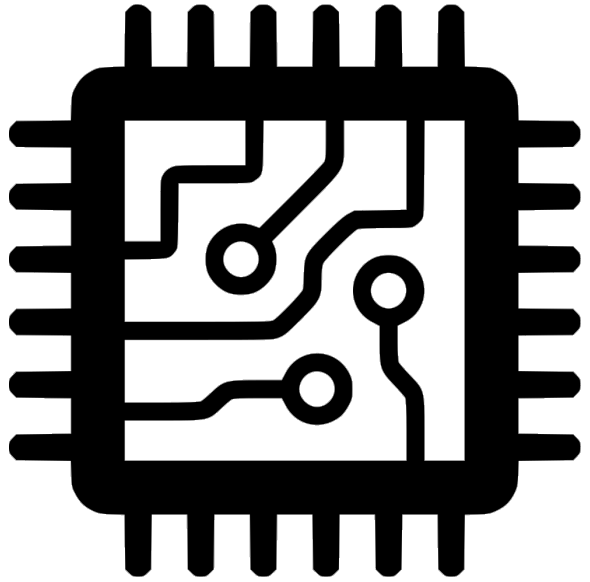
Hardware Design

- Most work only supports off-the-shelf network designs
- Most effort has focused on reducing precision and on pruning
- Often this throughput improvement comes at the expense of *lower accuracy*



Can we close this gap?





Design better ConvNet

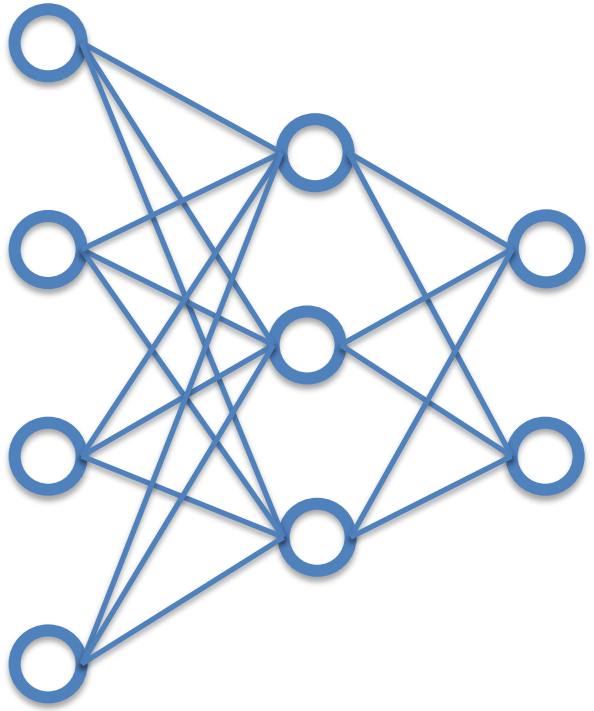
Design better hardware

Outline

- Introduction
- ConvNet Design
- Hardware Accelerator Design
- Experimental Results
- Conclusion

ConvNet Design Strategies

- Strategy 1: Use efficient models
- Strategy 2: Simplify operators
- Strategy 3: Quantize

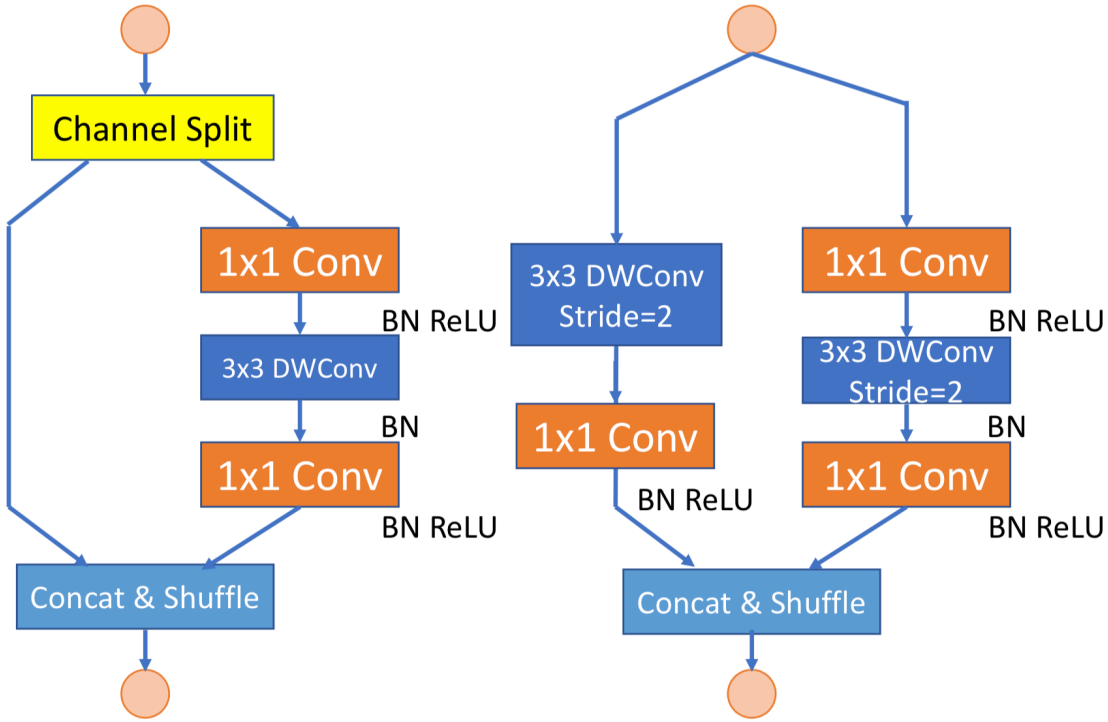


Design better ConvNet

CNN Strategy 1: Use efficient models

- **ShuffleNetV2-1.0**x [1] as our starting point
- Compared to VGG16:
  - 65x fewer OPs
  - 48x fewer parameters
  - Near equal accuracy on ImageNet

	MACs	#Params	Top-1 Acc
ShuffleNetV2-1.0x	146M	2.3M	69.4%
VGG16	15.3G	138M	71.5%



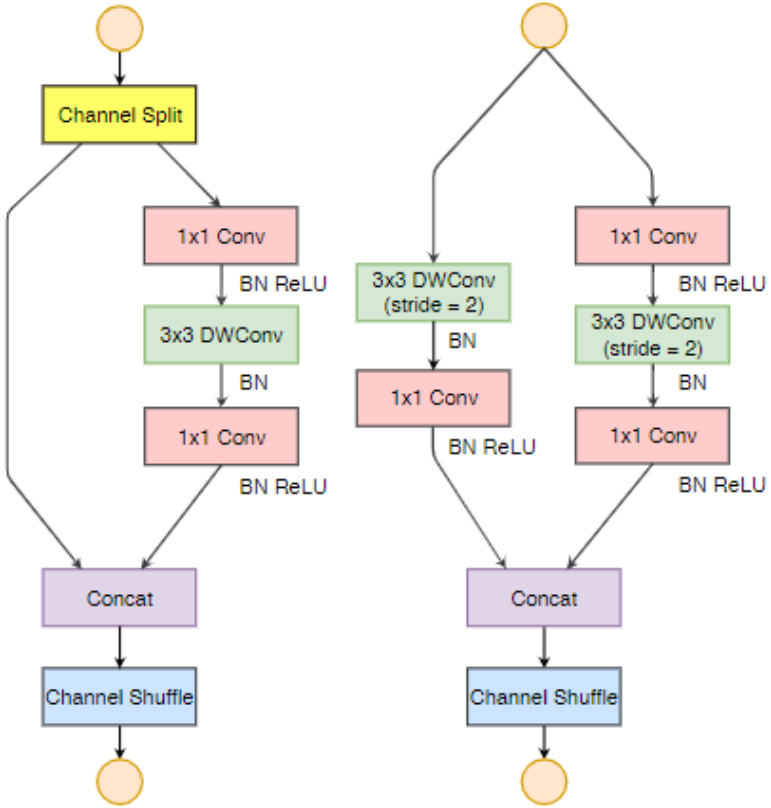
ShuffleNetV2 building blocks

[1] Ma, N., Zhang, X., Zheng, H. and Sun, J. {ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design}. arXiv e-prints. 1807-11164.

ShuffleNetV2

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Macro-architecture

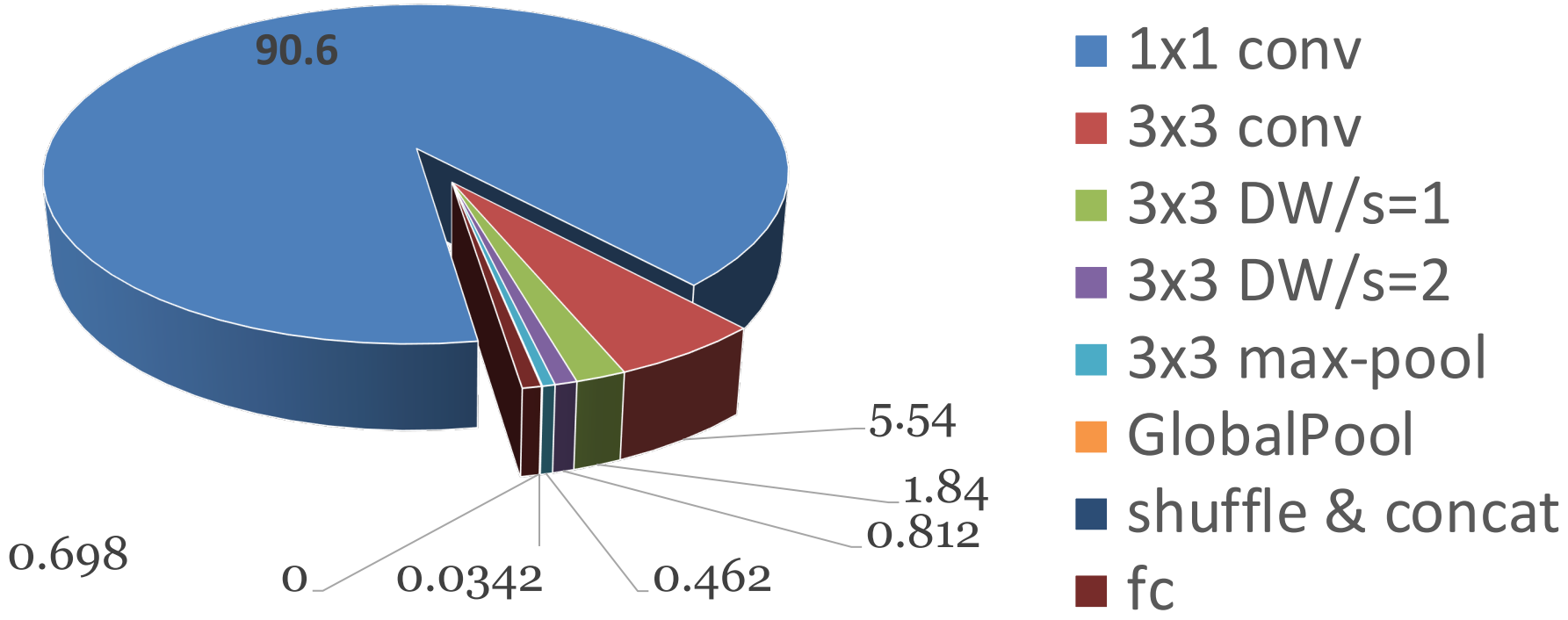


Building Block

[1] Ma, N., Zhang, X., Zheng, H. and Sun, J. {ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design}. arXiv e-prints. 1807-11164.

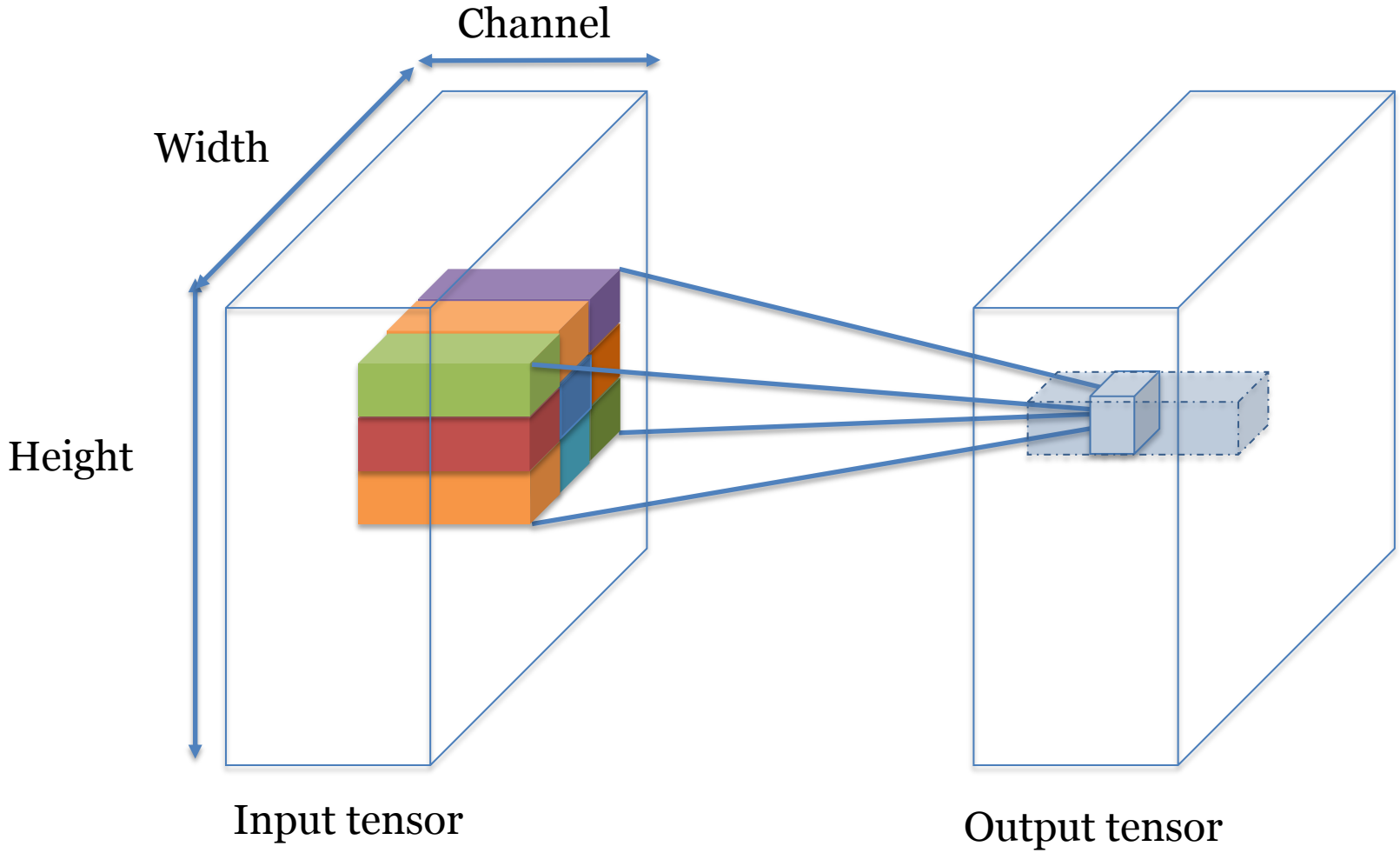
CNN Strategy 2: Simplify operators

- Can we reduce the number of operator types?
- Can we make the operation more hw-friendly?



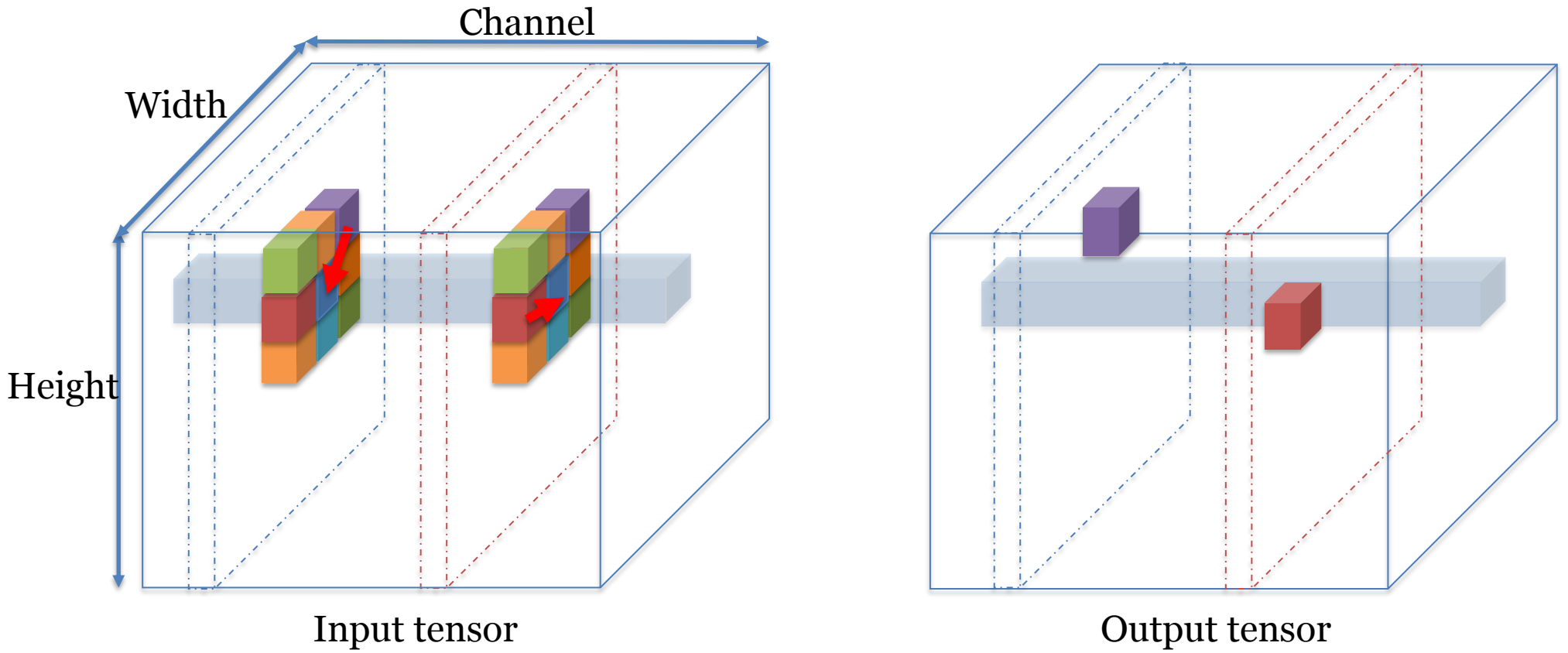
CNN Strategy 2: Simplify operators

- Can we replace 3x3 convolutions?



The Shift Operation

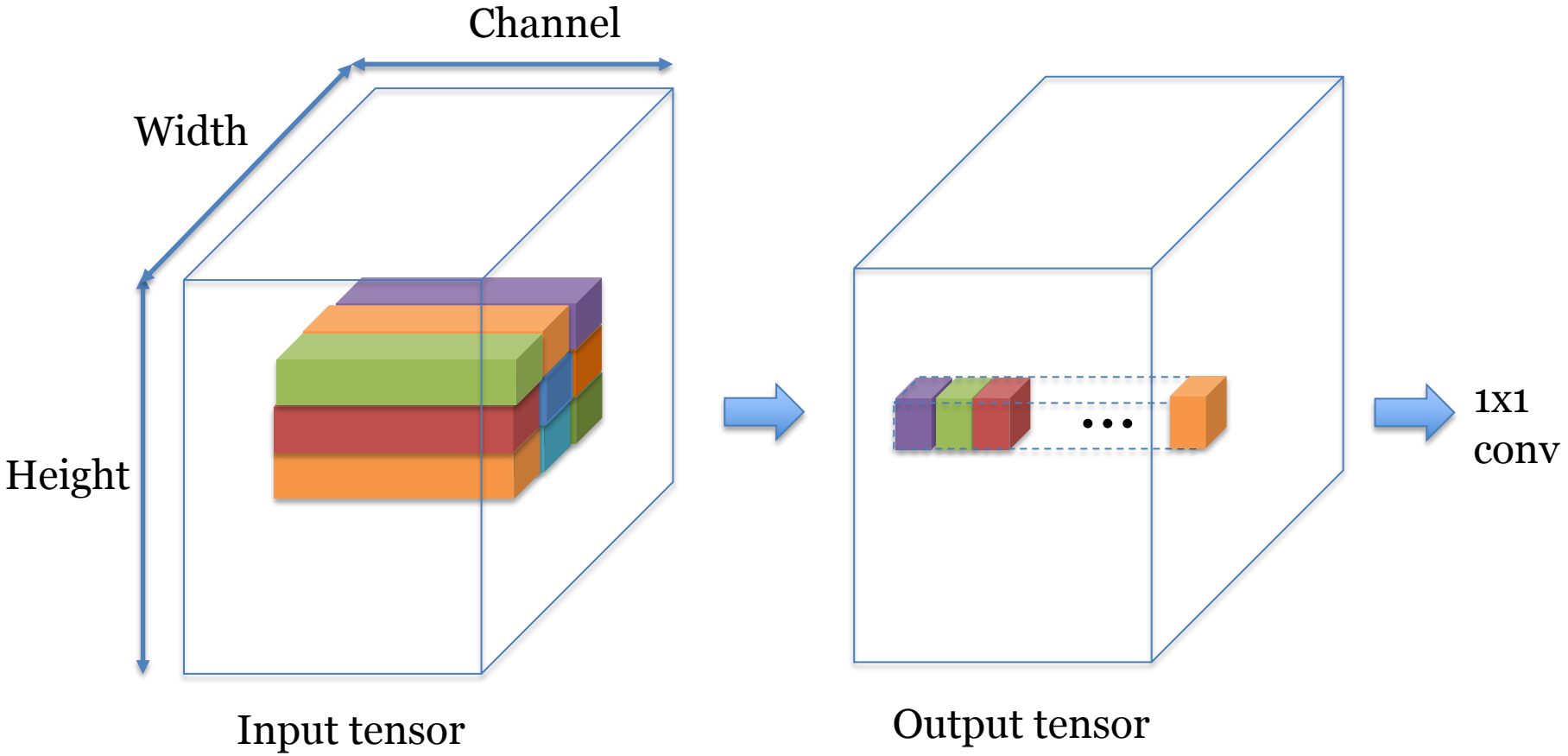
- The shift operation moves a neighboring pixel to the center position



[1] Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J. and Keutzer, K. {Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions}. arXiv e-prints. 1711-8141.

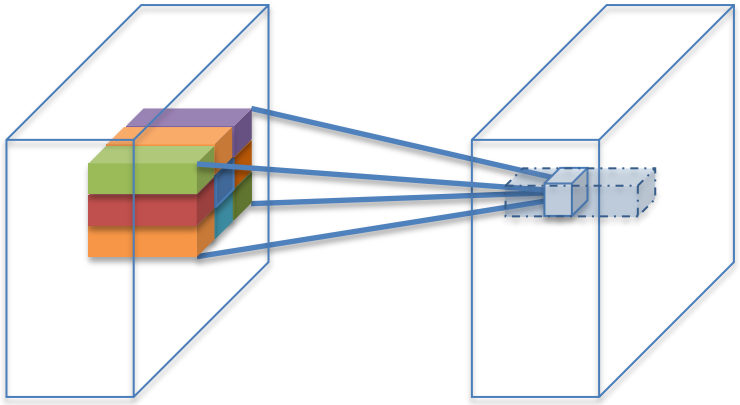
The Shift Operation

- 1x1 conv aggregates spatial information along the channel dimension



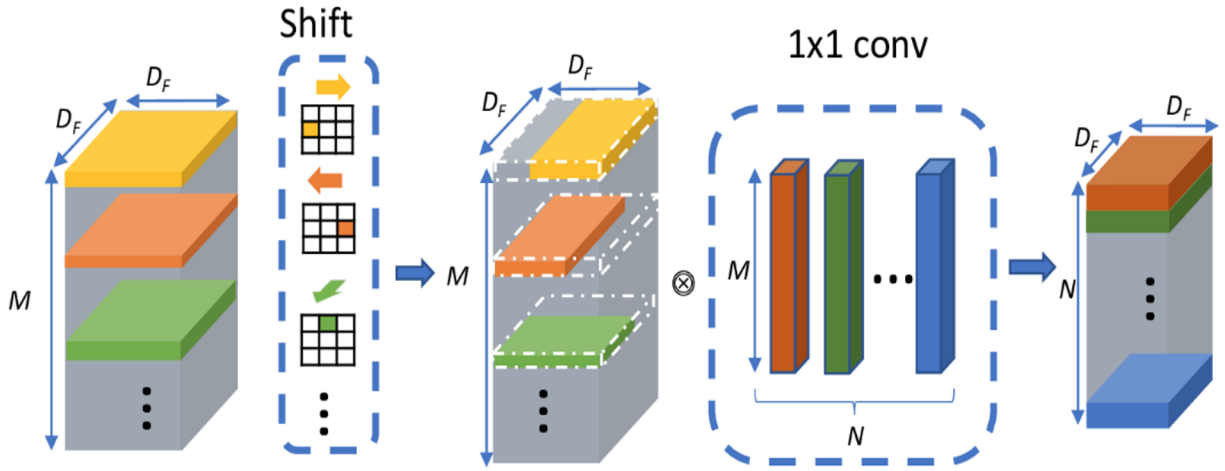
Replace 3 x 3 Conv

- **3x3 conv:**
  - Aggregates neighboring pixels
  - Mixes channel info



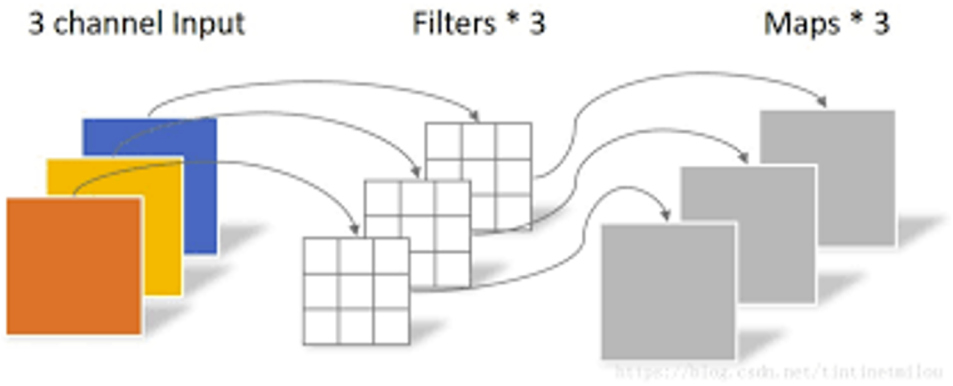
Input tensor Output tensor

- **shift**: Re-aligns pixels
- **1x1 conv**: Mixes channel info

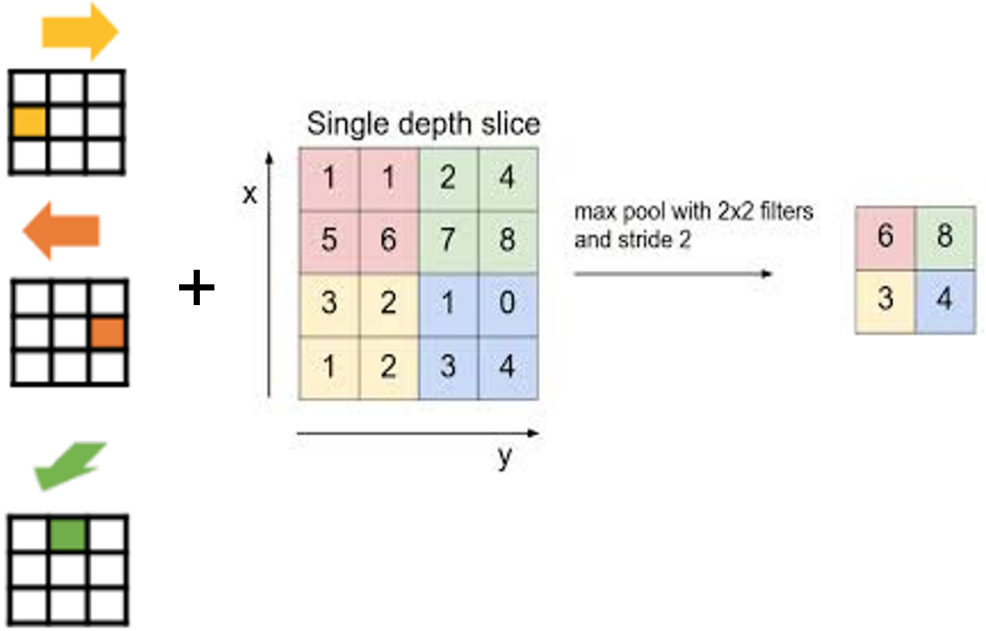


Replace 3 x 3 DW Conv

- **3x3 DW conv w/ stride 2:**
  - Aggregates neighboring pixels
  - Downsamples

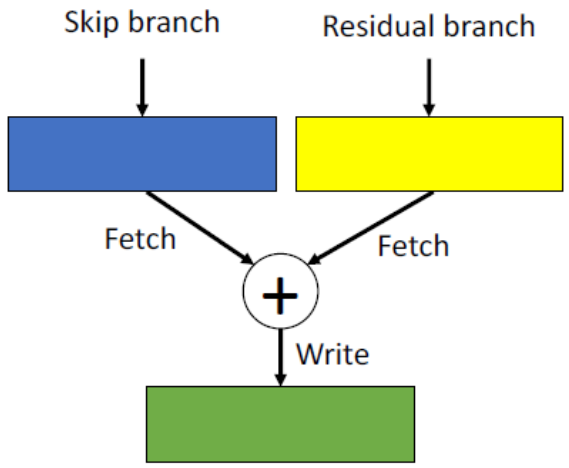


- **shift**: Re-aligns pixels
- **2x2 pooling w/ stride 2**: Downsamples

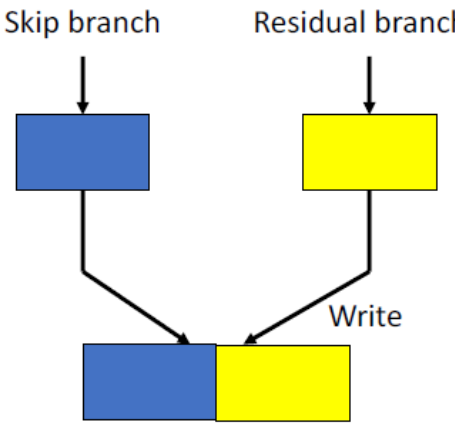


Strategy 2: Concatenative Connection

- Concatenative skip connection
  - Achieve similar accuracy
  - Less CPU-FPGA data movement
  - Less on-chip synchronization and buffer
  - Quantization friendly



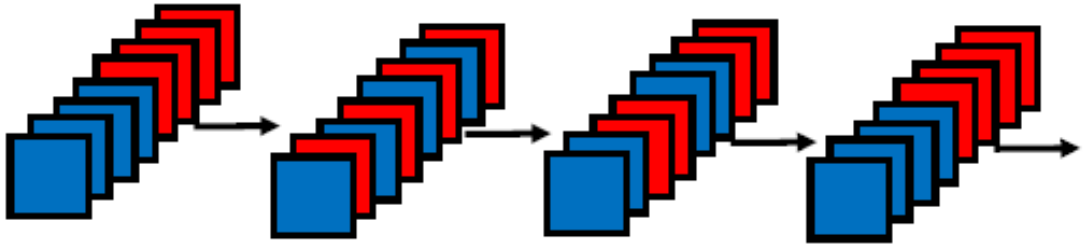
(a) Additive skip connection



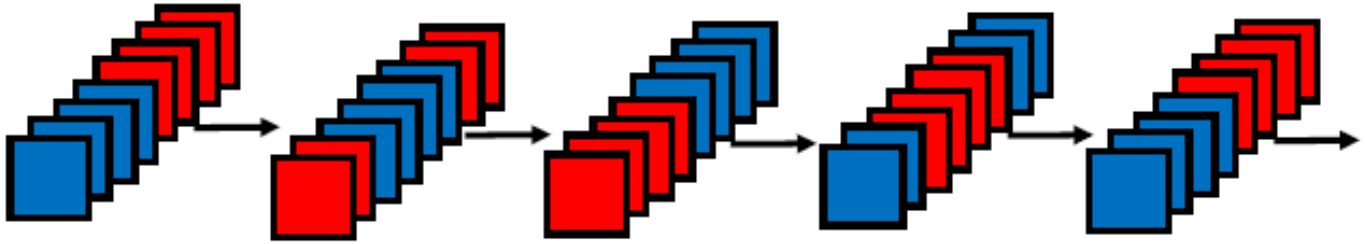
(b) Concatenative skip connection

Strategy 2: Use hw-friendly operators

- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



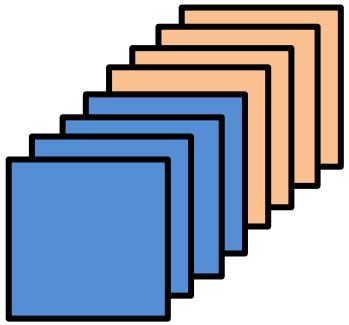
(a) Transpose based channel shuffle



(b) Our channel shuffle

Strategy 2: Use hw-friendly operators

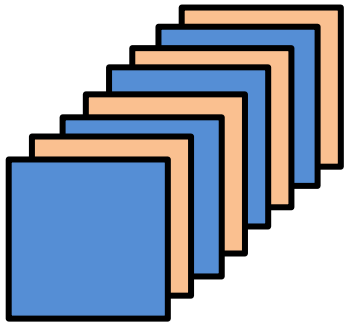
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Transpose based channel shuffle

Strategy 2: Use hw-friendly operators

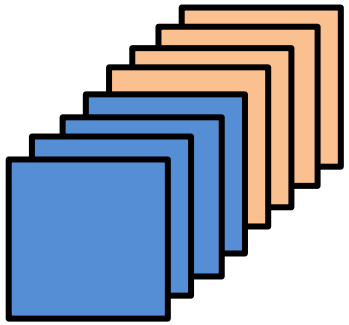
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Transpose based channel shuffle

Strategy 2: Use hw-friendly operators

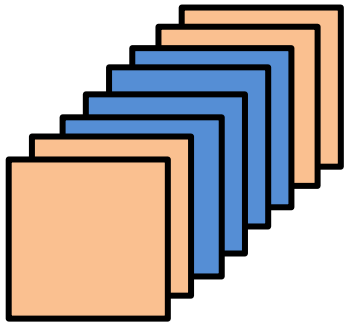
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Our channel shuffle

Strategy 2: Use hw-friendly operators

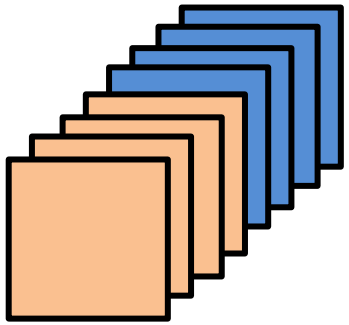
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Our channel shuffle

Strategy 2: Use hw-friendly operators

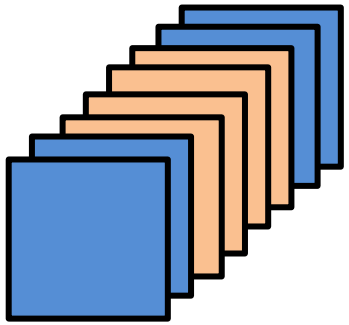
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Our channel shuffle

Strategy 2: Use hw-friendly operators

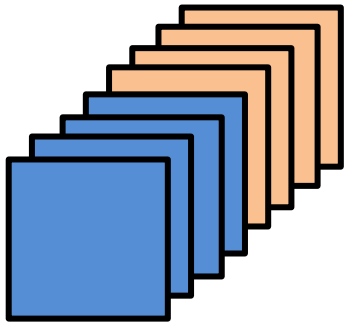
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Our channel shuffle

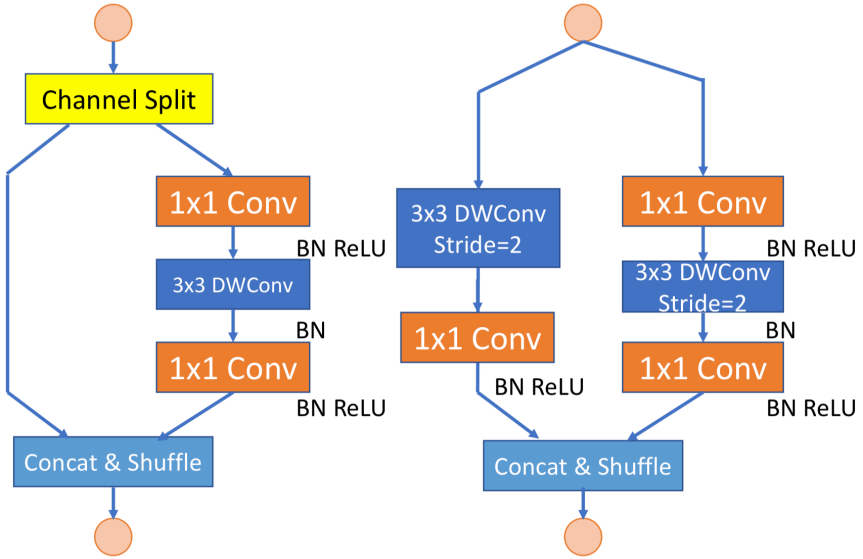
Strategy 2: Use hw-friendly operators

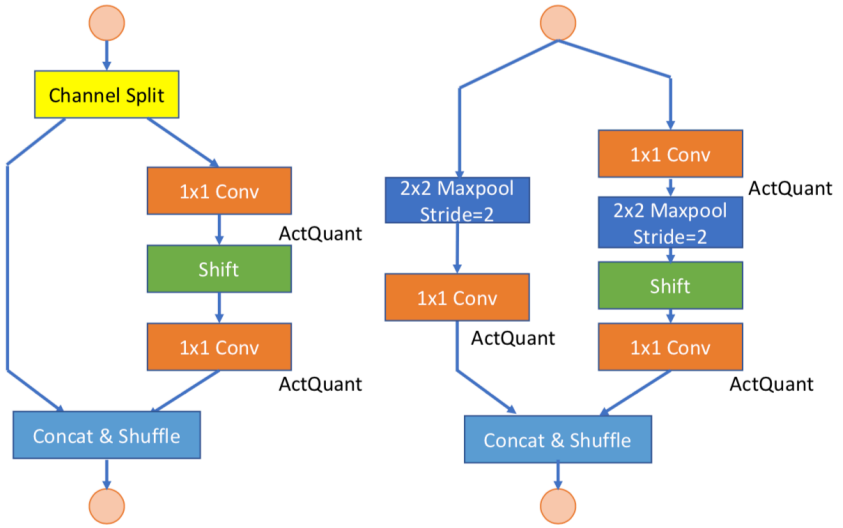
- 3x3 max-pooling -> 2x2 max-pooling
- Hw-friendly channel shuffle



Our channel shuffle

ShuffleNetV2 -> DiracDeltaNet





- Accuracy (full precision): **69.4%**
- Operators involved:
  - 1x1 convolution
  - 3x3 convolution
  - 3x3 DW convolution
  - 3x3 max pooling
  - Channel split/shuffle/concat

- Accuracy (full precision): **69.7%**
- Operators involved:
  - 1x1 convolution
  - 2x2 max pooling
  - Channel split/shuffle/shift/concat

[1] Ma, Ningning, et al. "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design." arXiv preprint arXiv:1807.11164 (2018).

CNN Strategy 3: Quantize

- Quantization has been mostly demonstrated on large networks. Is it effective on the small ones like DiracDeltaNet?
- We used existing quantization methods:
  - DoReFaNet [1] method for weights
  - Modified PACT [2] method for activations
- We achieved 4-bit weight and 4-bit activation precision with competitive accuracy

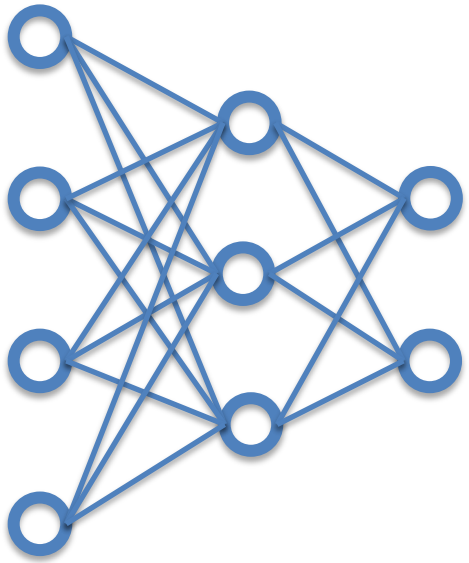
	Network	Pruning	Precision	Top-1 Acc
[3]	VGG16	Yes	8-8b	67.72%
Ours	DiracDeltaNet	No	4-4b	67.52%

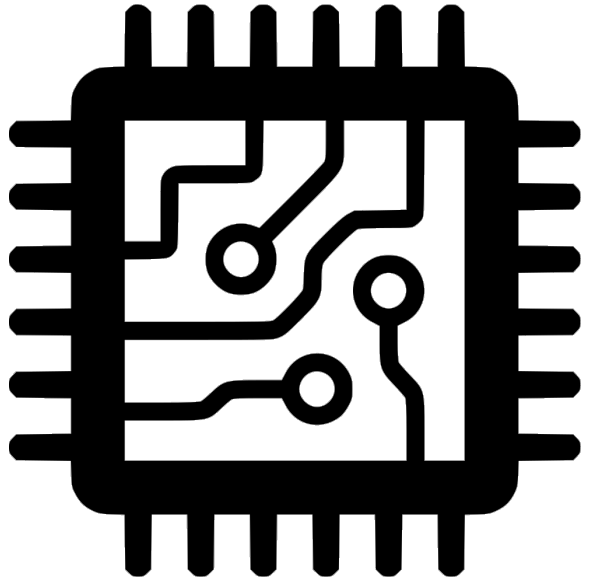
[1] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H. and Zou, Y. {DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients}. arXiv e-prints. 1606-6160.
[2] Choi, J., Wang, Z., Venkataramani, S., I-Jen Chuang, P., Srinivasan, V. and Gopalakrishnan, K.PACT: Parameterized Clipping Activation for Quantized Neural Networks.
[3] Guo, K., Han, S., Yao, S., Wang, Y., Xie, Y. and Yang, H. Software-Hardware Codesign for Efficient Neural Network Acceleration. IEEE Micro

Outline

- Introduction
- ConvNet Design
- Hardware Accelerator Design
- Experimental Results
- Conclusion

Can we close this gap?



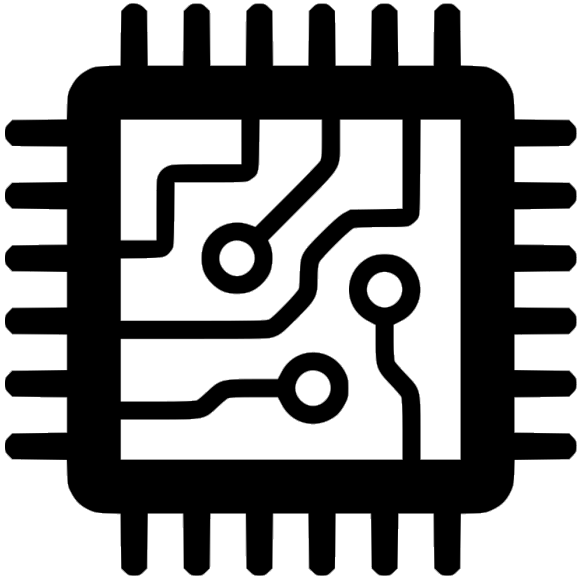


Design better ConvNet

Design better hardware

Hardware Design Strategies

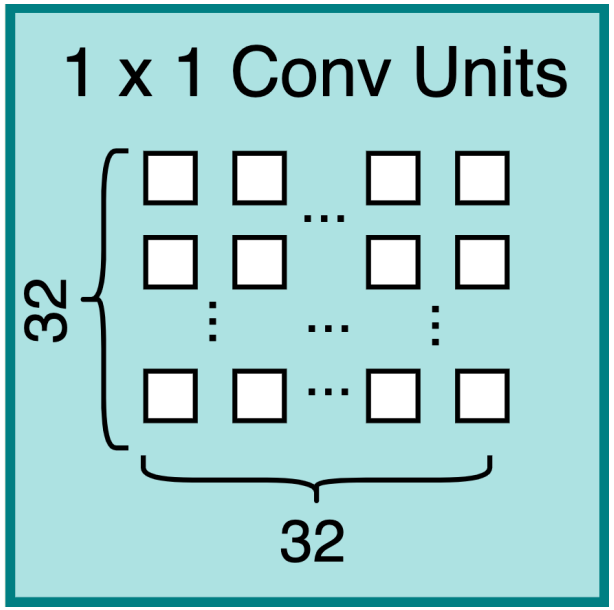
- Strategy 1: Specialize conv engine
- Strategy 2: Use dataflow architecture
- Strategy 3: Merge layers



Design better hardware

HW Strategy 1: Specialize conv engine

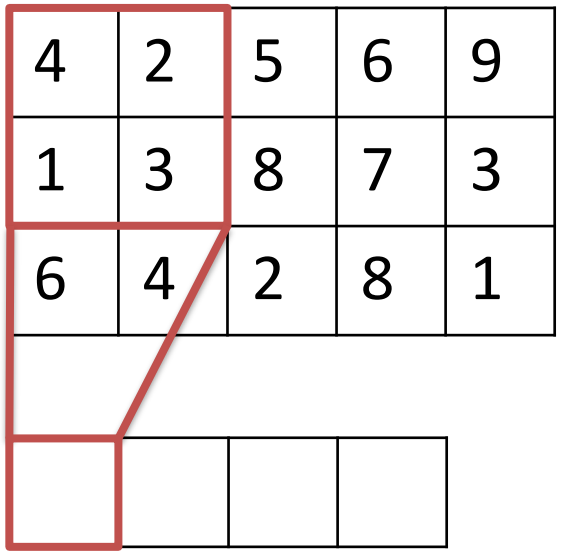
- 1x1 Conv Unit:
  - Supports matrix-vector multiplication
    - 4-bit inputs
    - 4-bit weights
    - 17-bit partial sums
  - Buffers weights and partial sums on-chip
  - Performs 32 x 32 MACs per iteration
  - Each input gets reused *output channel size* times



Strategy 2: Use dataflow architecture

- 1x1 conv
  - No line-buffer
- shift
  - 3x3 sliding window, II=1
- 2x2 max-pooling
  - 2x2 sliding window, II=2

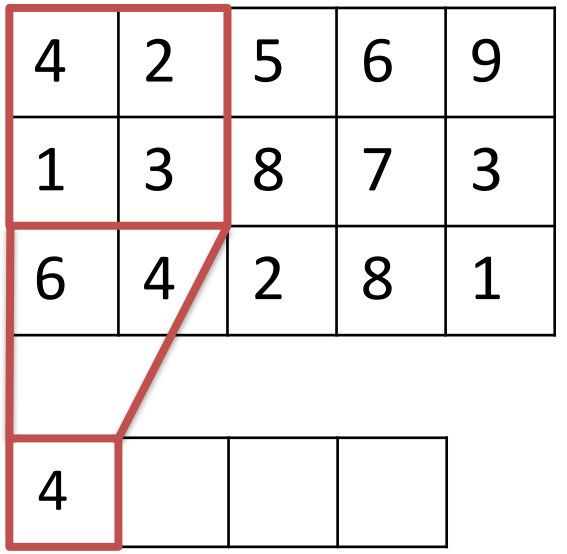
2x2 max-pooling example:



Strategy 2: Use dataflow architecture

- 1x1 conv
  - No line-buffer
- shift
  - 3x3 sliding window, II=1
- 2x2 max-pooling
  - 2x2 sliding window, II=2

2x2 max-pooling example:



Strategy 2: Use dataflow architecture

- 1x1 conv
  - No line-buffer
- shift
  - 3x3 sliding window, II=1
- 2x2 max-pooling
  - 2x2 sliding window, II=2

2x2 max-pooling example:

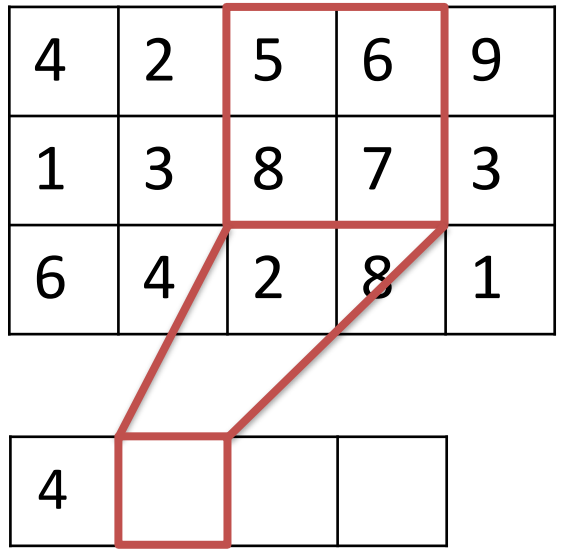
4	2	5	6	9
1	3	8	7	3
6	4	2	8	1

4			

Strategy 2: Use dataflow architecture

- 1x1 conv
  - No line-buffer
- shift
  - 3x3 sliding window, II=1
- 2x2 max-pooling
  - 2x2 sliding window, II=2

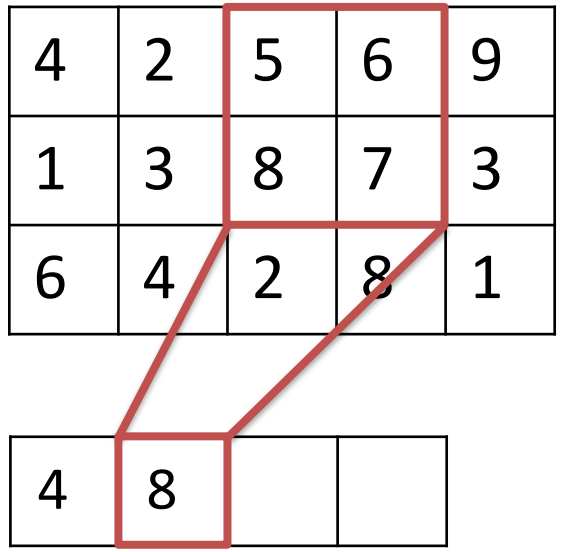
2x2 max-pooling example:



Strategy 2: Use dataflow architecture

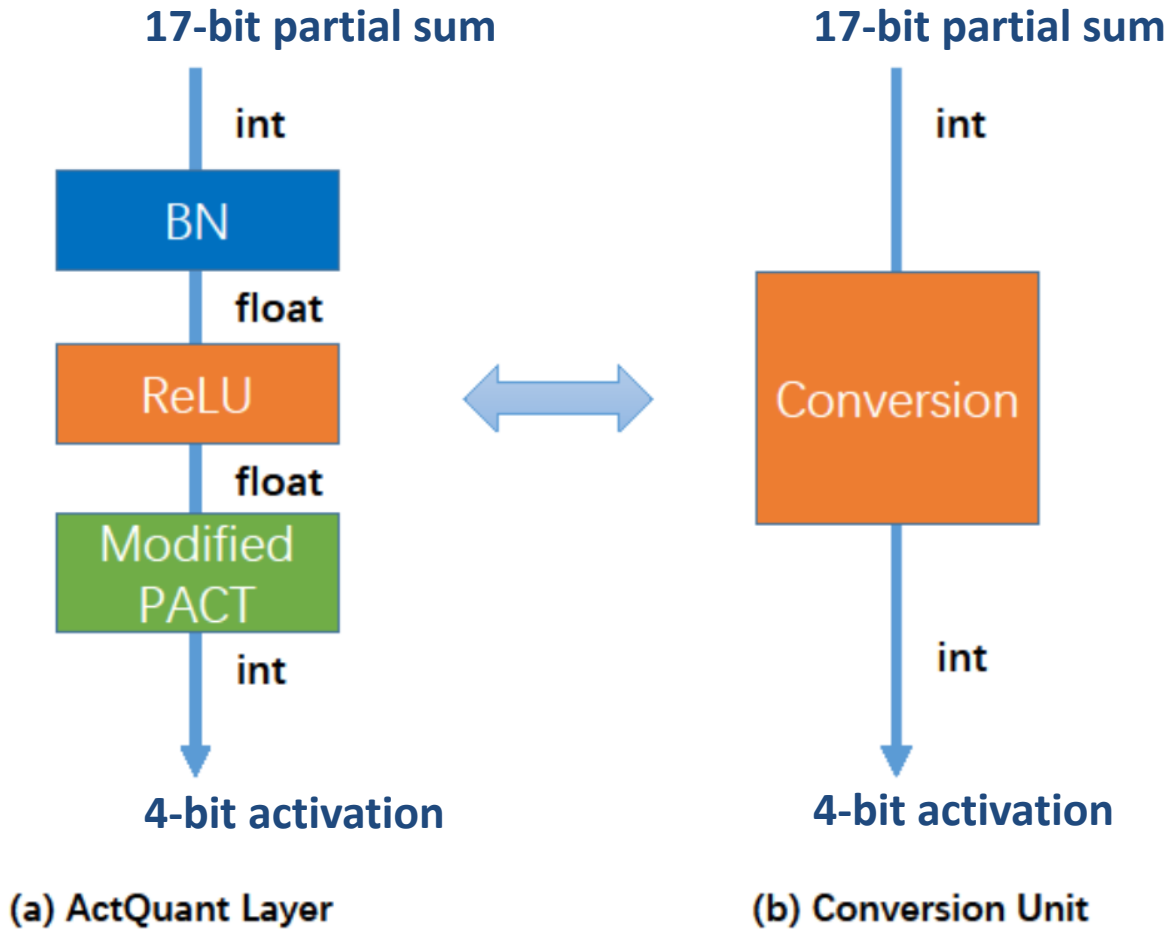
- 1x1 conv
  - No line-buffer
- shift
  - 3x3 sliding window, II=1
- 2x2 max-pooling
  - 2x2 sliding window, II=2

2x2 max-pooling example:



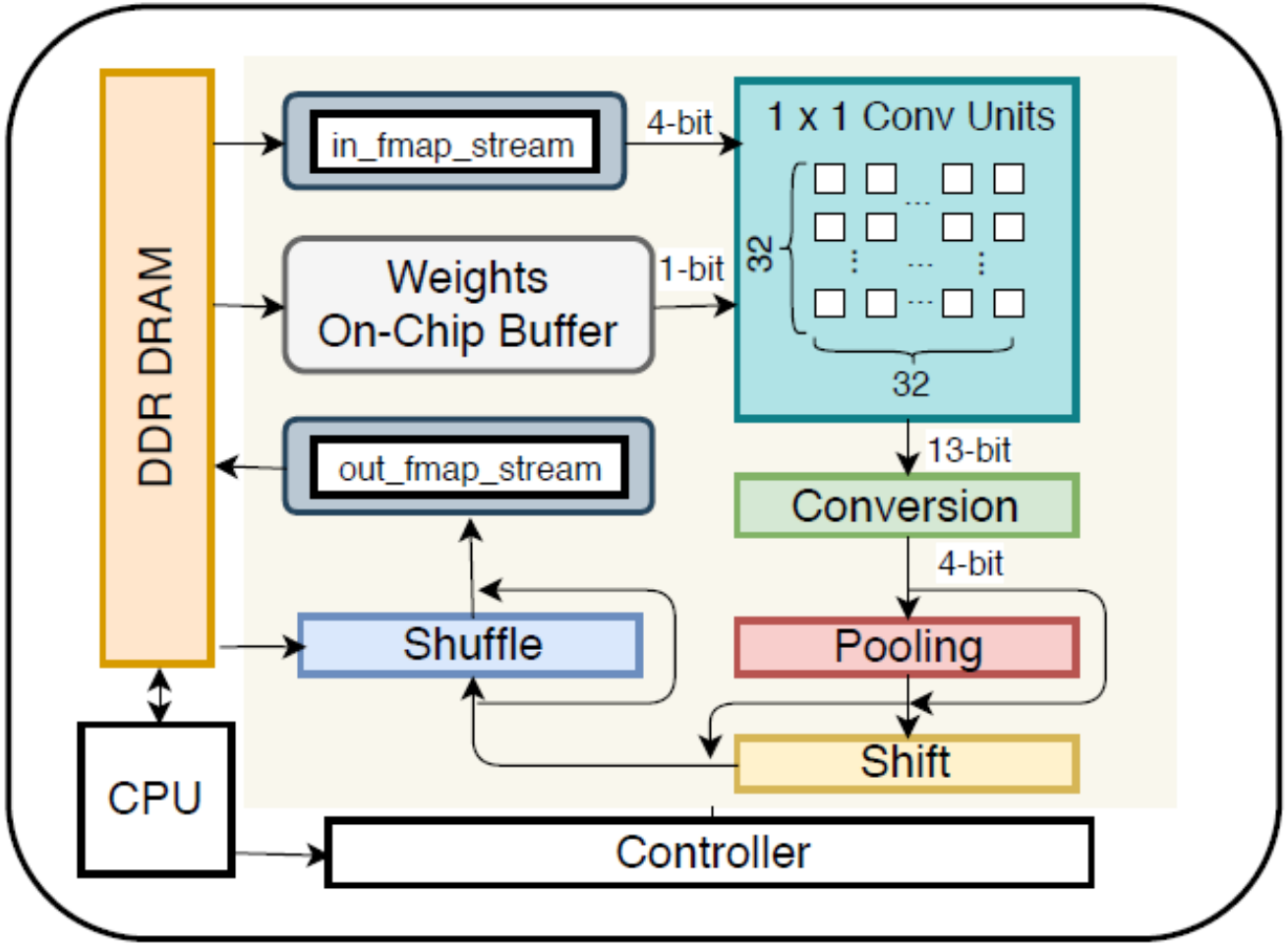
Strategy 3: Merge layers

- Conversion unit includes:
  - Batch Norm
  - ReLU
  - Modified PACT
- It performs 17-bit to 4-bit conversion
- It is implemented with comparators

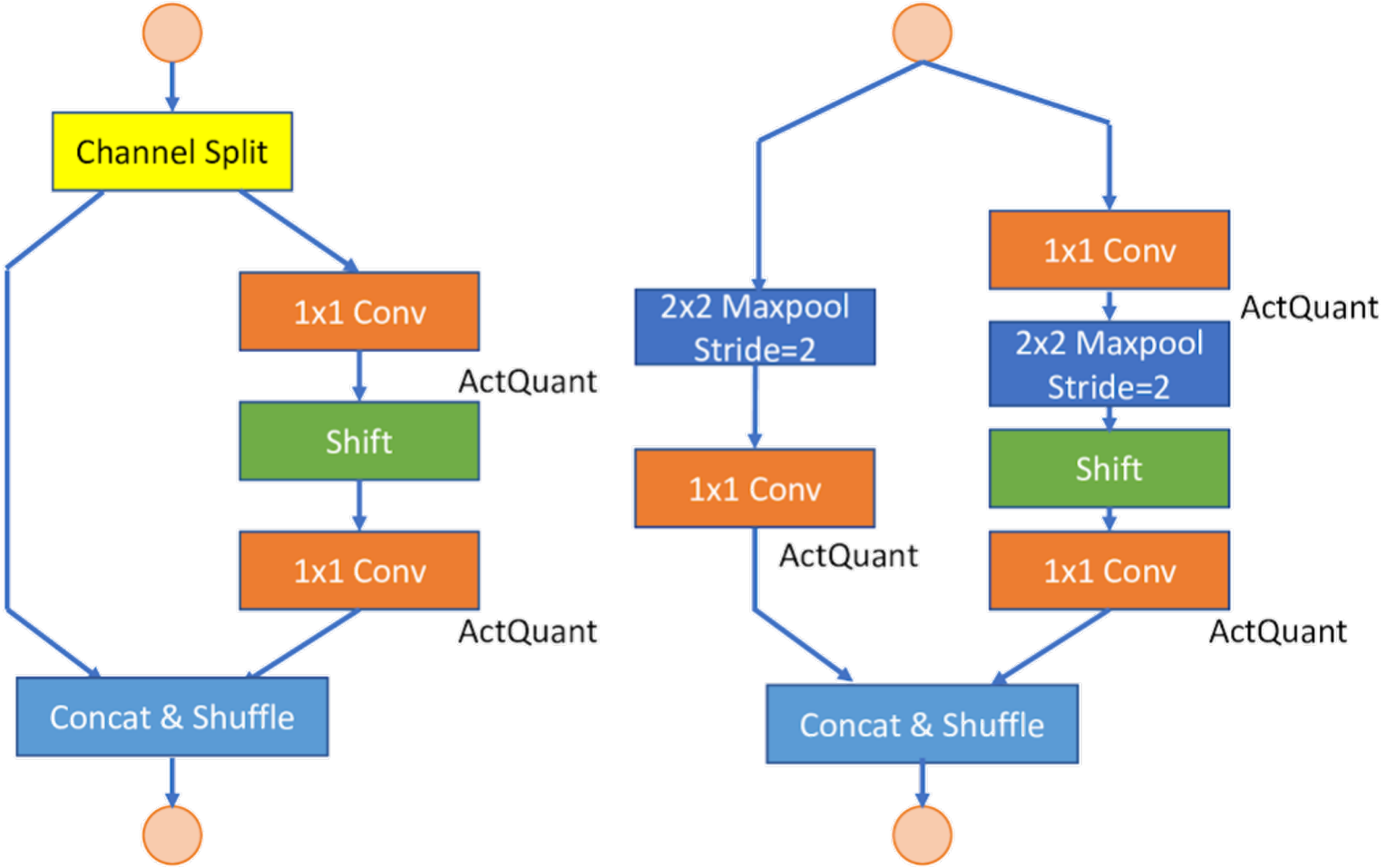


Synetgy Architecture Overview

- HW engine supports:
  - 1x1 conv
  - 2x2 max-pooling
  - shift
  - shuffle
- Layer-based design
- Implemented with Vivado HLS and PYNQ

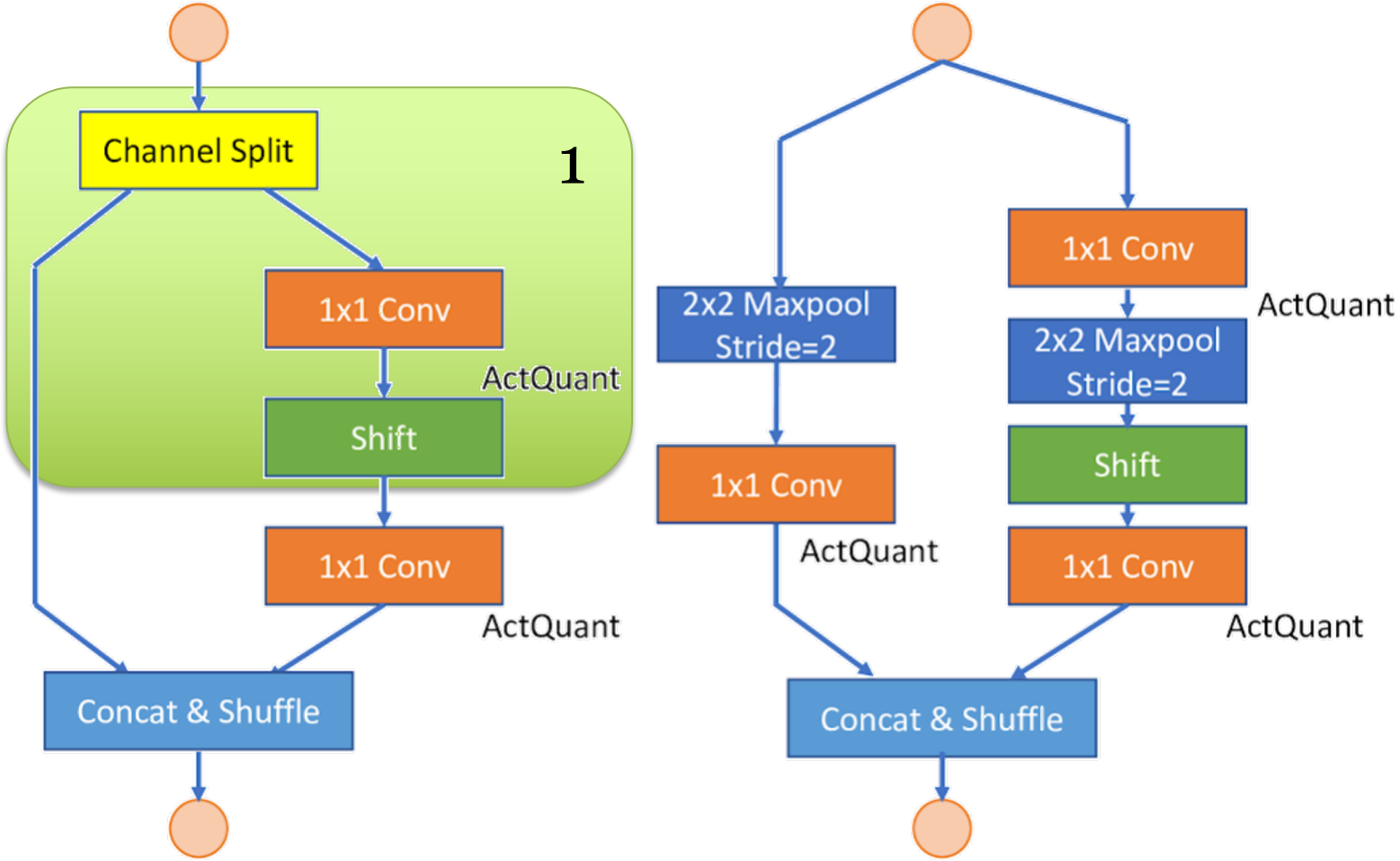


Execution Model



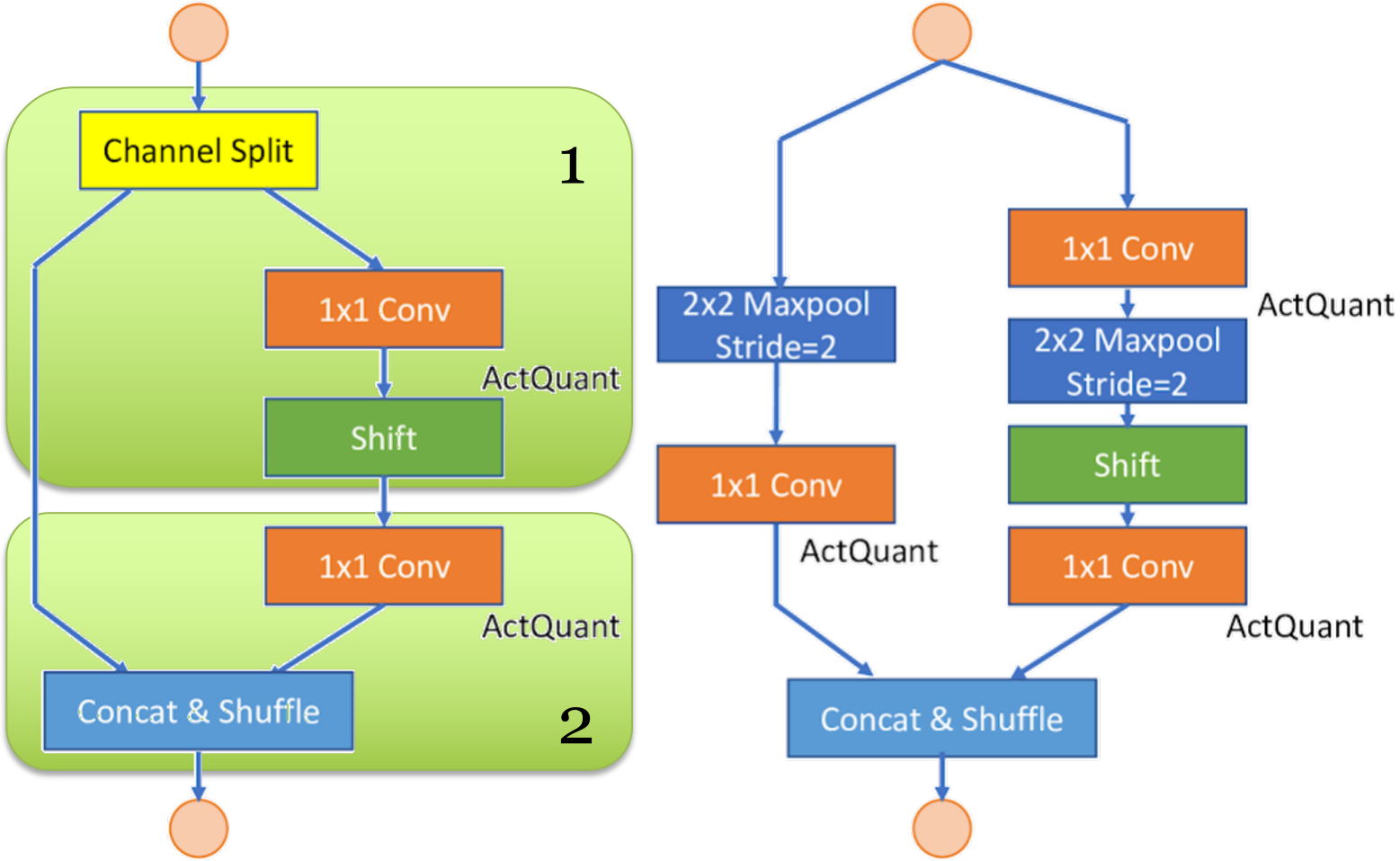
DiracDeltaNet Block

Execution Model



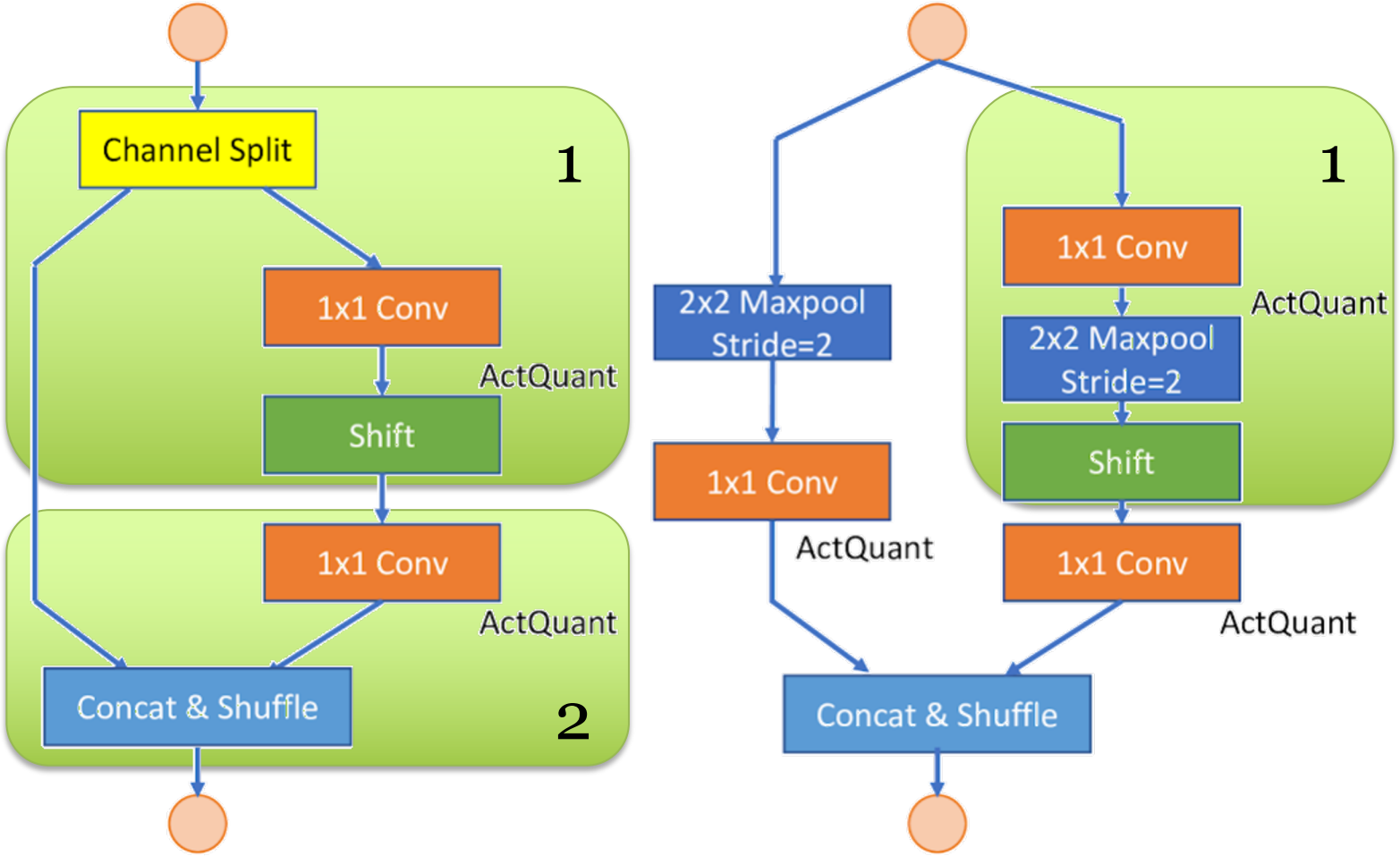
DiracDeltaNet Block

Execution Model



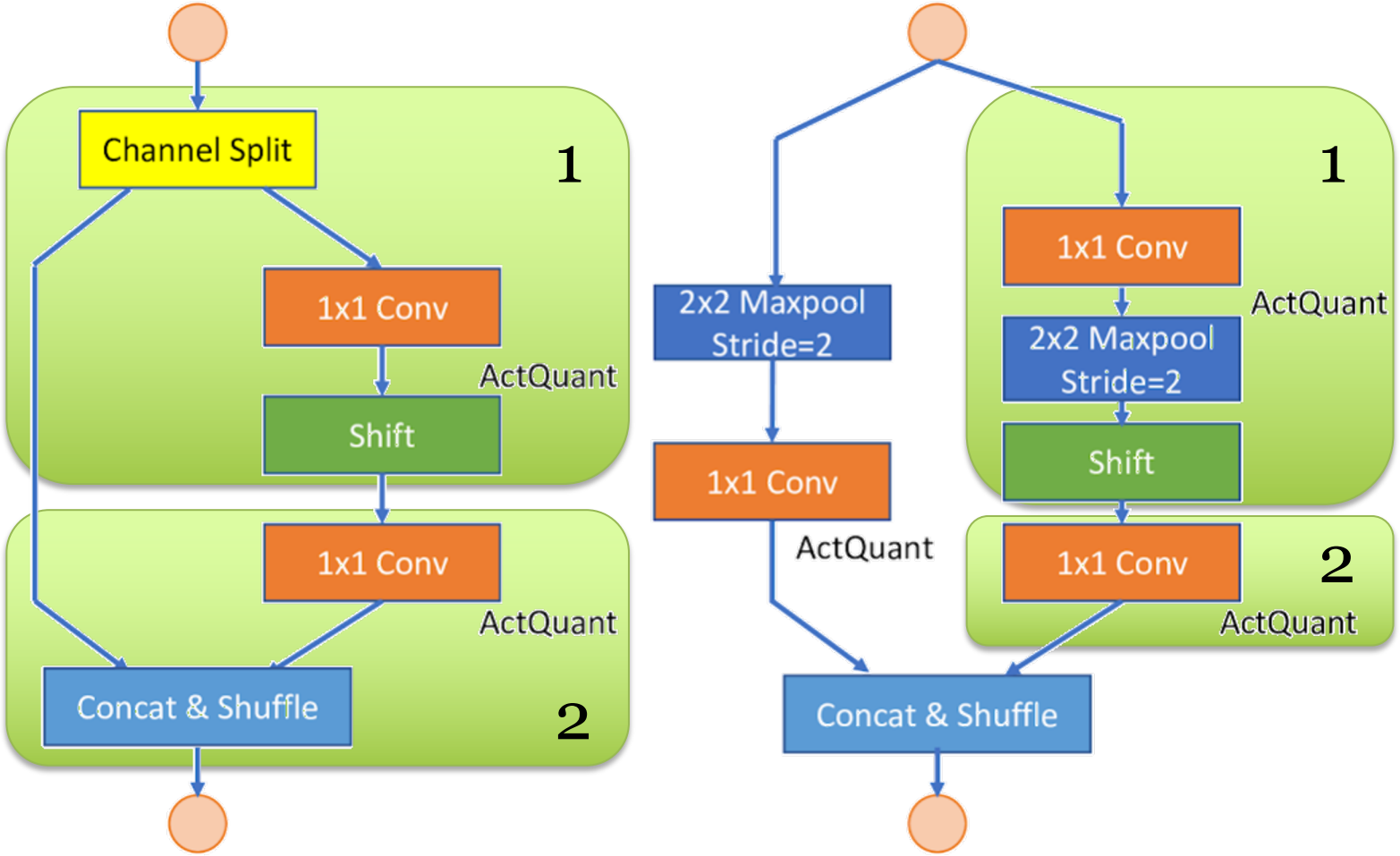
DiracDeltaNet Block

Execution Model



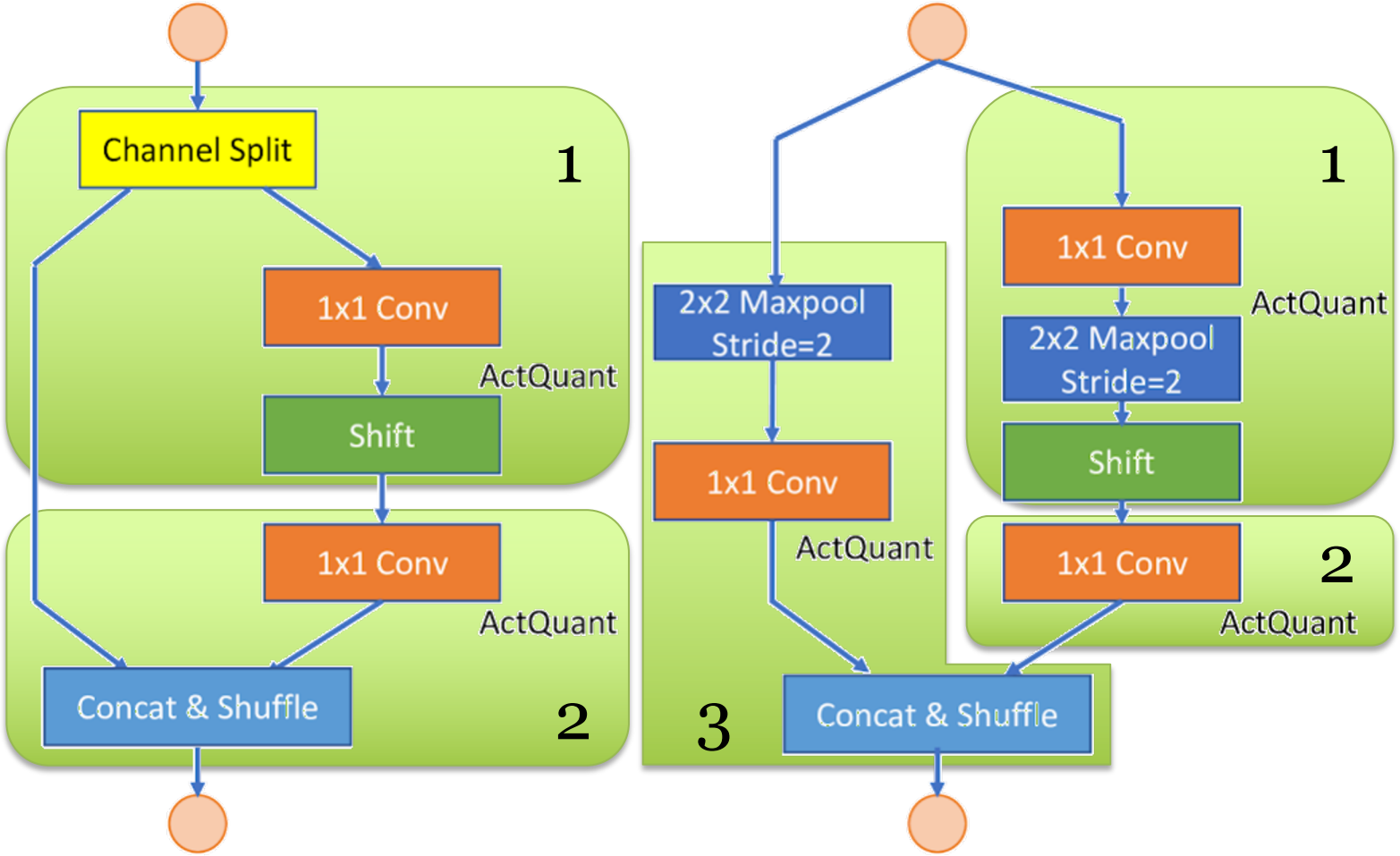
DiracDeltaNet Block

Execution Model



DiracDeltaNet Block

Execution Model



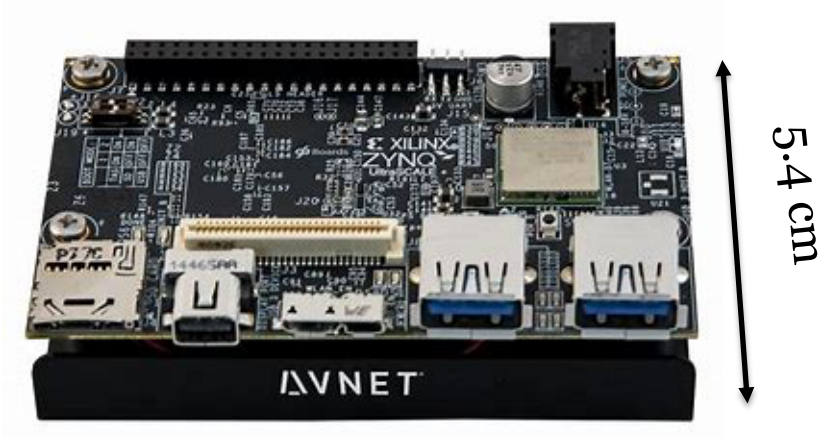
DiracDeltaNet Block

Outline

- Introduction
- ConvNet Design
- Hardware Accelerator Design
- Experimental Results
- Conclusion

Experimental Setup

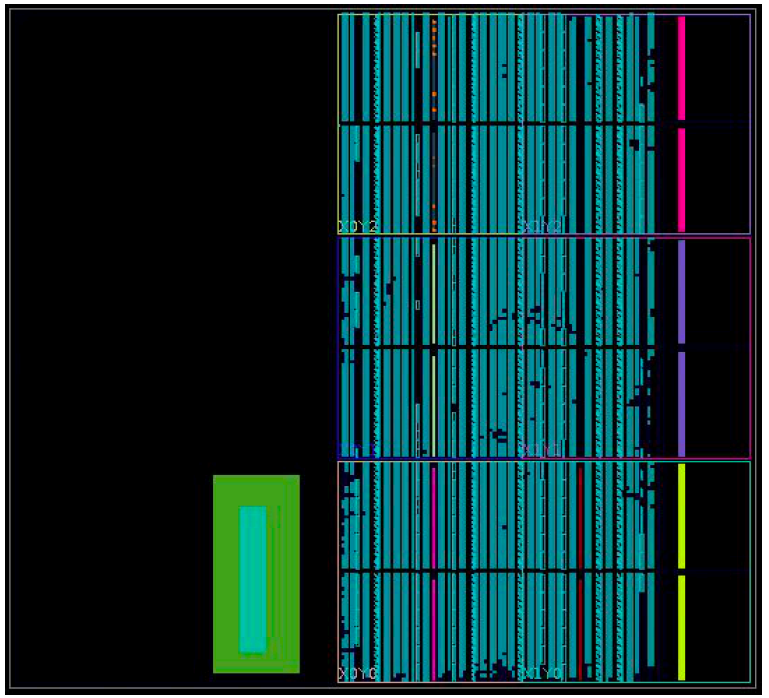
- Avnet Ultra96 Board
- With Xilinx ZU3EG FPGA – the second smallest device in the Ultrascale+ family



Resource Usage

LUT	FF	BRAM	DSP
51776(73.4%)	42257(29.9%)	159(73.6%)	360(100%)

- LUT: 4/4bit multiplications (1x1 conv)
- FF: fully-partitioned partial sums (1x1 conv)
- BRAM: line buffers and FIFOs (dataflow)
- DSP: 4/4bit multiplications (1x1conv)



On-chip Layout

Comparison with Previous Work

	Platform	Framerate	Top-1 Acc	Precision	Energy/ Frame (J)
[1]	Zynq 7Z020	5.7	**67.72%**	8-8b	0.526
[2]	Zynq 7Z045	4.5	64.64%	16-16b	0.666
[3]	Stratix-V	3.8	66.58%	8-16b	5.026
Ours	Zynq ZU3EG	**66.3**	67.52%	**4-4b**	**0.083**

- Equal top-1 accuracy
- 11.6x higher framerate
- 6.3x more power efficient

[1] Guo, K., Han, S., Yao, S., Wang, Y., Xie, Y. and Yang, H. Software-Hardware Codesign for Efficient Neural Network Acceleration. IEEE Micro, 37 (2). 18-25.
[2] Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., Wang, Y. and Yang, H. Going Deeper with Embedded {FPGA} Platform for Convolutional Neural Network, 2016, 26-35.
[3] Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S.B.K., Seo, J.S. and Cao, Y. Throughput-Optimized OpenCL-based {FPGA} Accelerator for Large-Scale Convolutional Neural Networks, 2016, 16-25.

Batch Size

Batch Size	1	2	4	8	16
Framerate (fps)	41.4	53.6	62.6	65.6	66.3

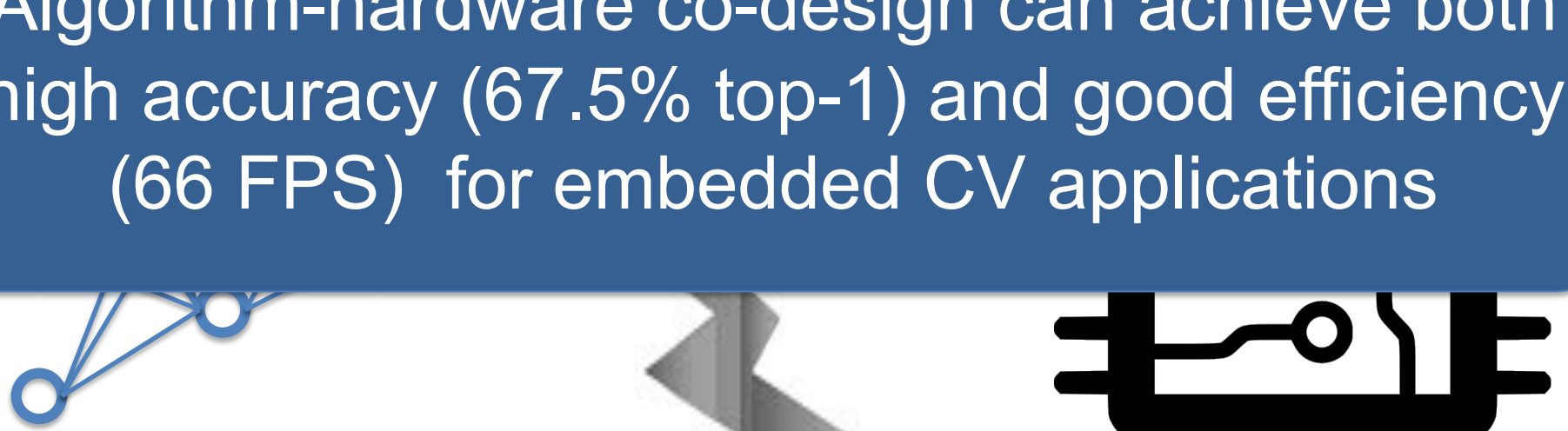
- Mitigate software API call (Python) overhead
  - Accelerator runtime (batch=1) 0.15ms
  - API call 0.40ms
- More activation and weight reuse leads to better performance

Outline

- Introduction
- ConvNet Design
- Hardware Accelerator Design
- Experimental Results
- Conclusion

Can we perform co-design to close this gap?

Algorithm-hardware co-design can achieve both high accuracy (67.5% top-1) and good efficiency (66 FPS) for embedded CV applications



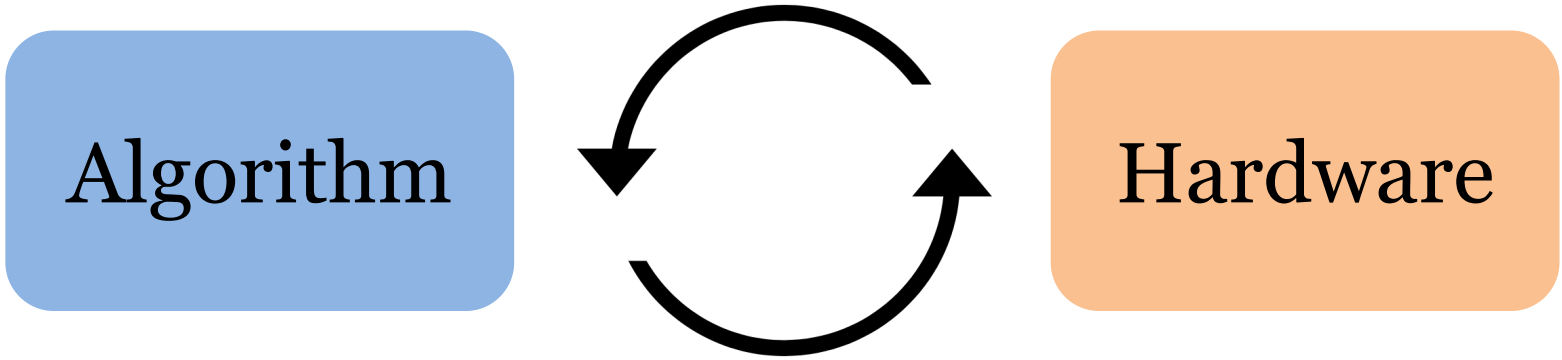
Design better ConvNet

Design better hardware

Future Work

Automate the co-design process

- More comprehensive deep neural network search
- More comprehensive HW design space exploration



Thank you!

Q&A