



# AI for AI: Automatically Generate Deep Neural Network Accelerator in FPGA for Data-center and Edge.

Presented By



Wang Jun Song (王均松)

IBM Research, China



- **Automation Tool -- AccDNN:**

- An end-to-end automation tool for generating convolutional neural network in FPGA without programming
- Some commercial pilots usecases.
- Cloud access for academic research and IBM's production roadmap

- **ELB-NN: Extremely low bit-width neural network**

- Model compression and its efficient implementation in FPGA

# AccDNN: An Automation Tool

**This work has been accepted in ICCAD'2018, San Diego, and wins the Best Paper Award.**

- **Goal for programmability :**
  - A tool to generate DNN accelerator without FPGA programming and keep RTL level performance

## DNN Application Design and Deployment Steps with AccDNN:

1. Design the specific deep neural network.
2. Training the network using GPU accelerator.
3. Use AccDNN to generate FPGA implementation
4. Deploy the recognition application using FPGA accelerator

***Do it Automatically!***

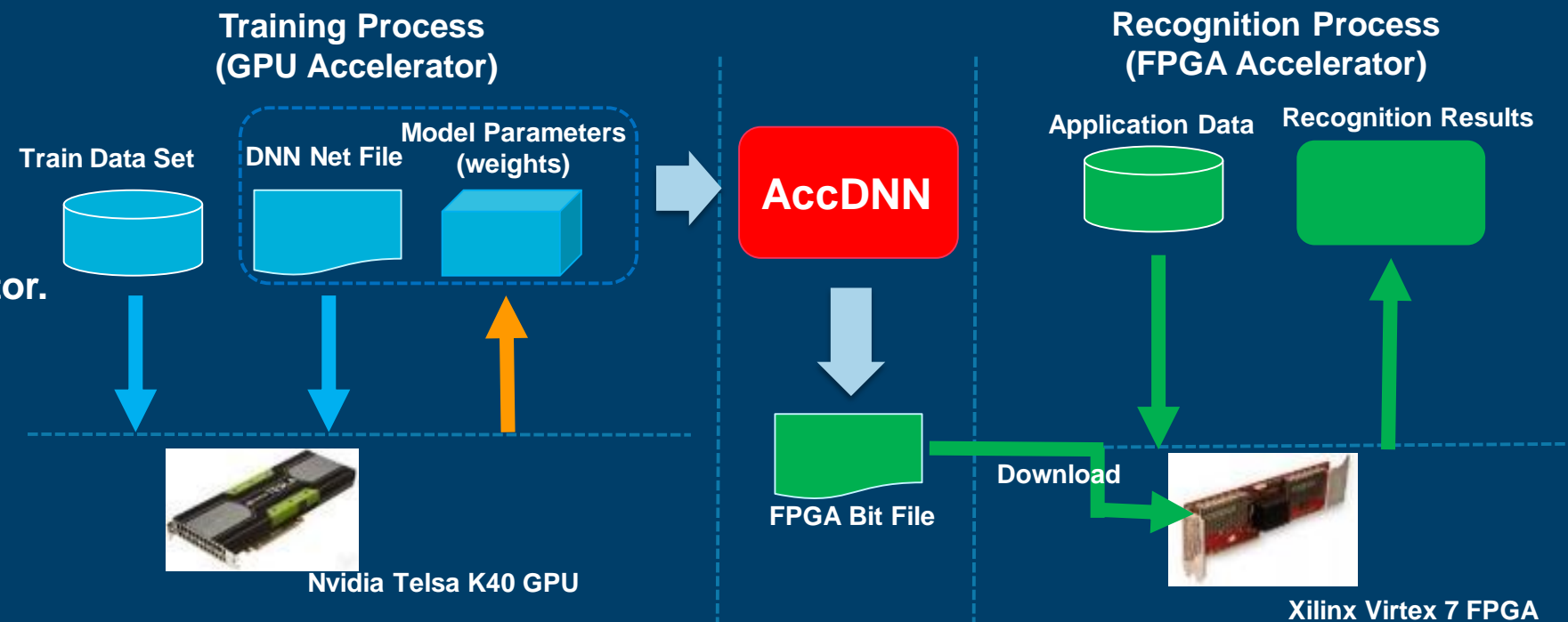
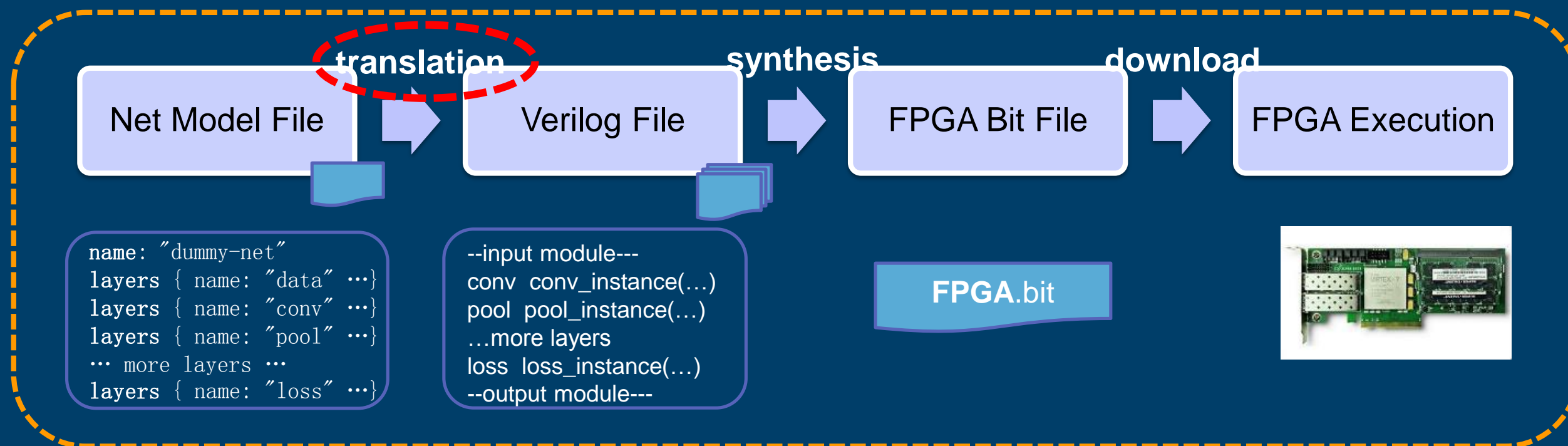


Illustration of training and recognition under *Caffe* framework

The critical stage is the translation.

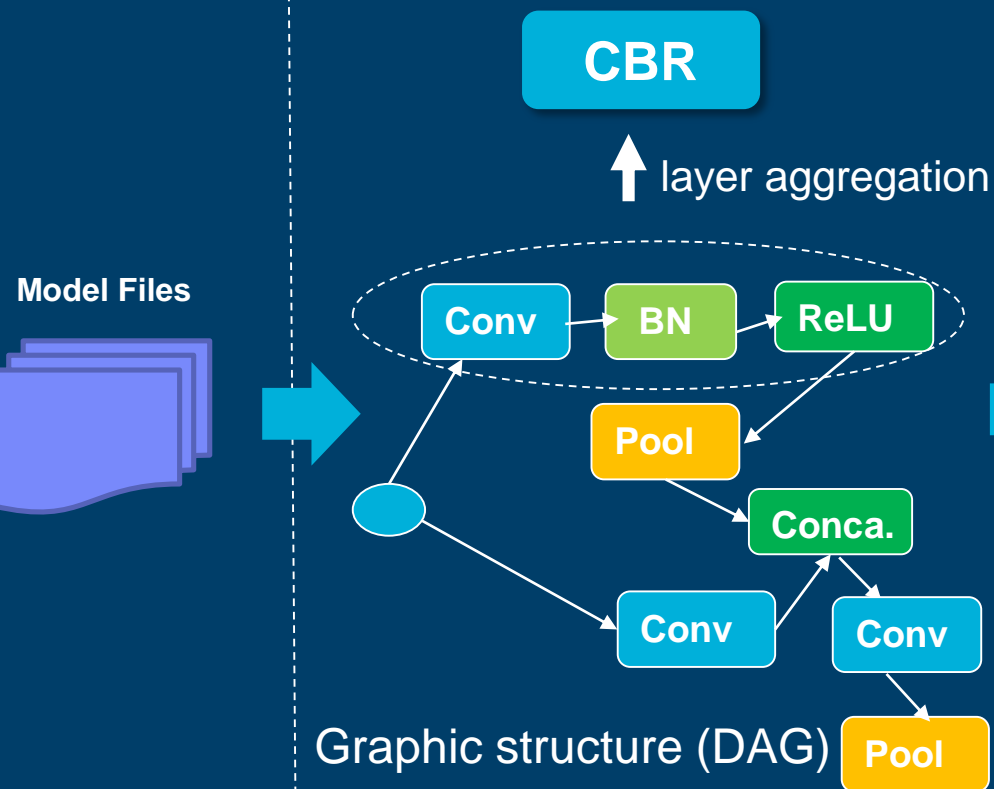
The other stages could be completed with Xilinx EDA toolchains, like Vivado.



## Parse and aggregation

- Graphic model for data flow
- Layer aggregation for efficient computation

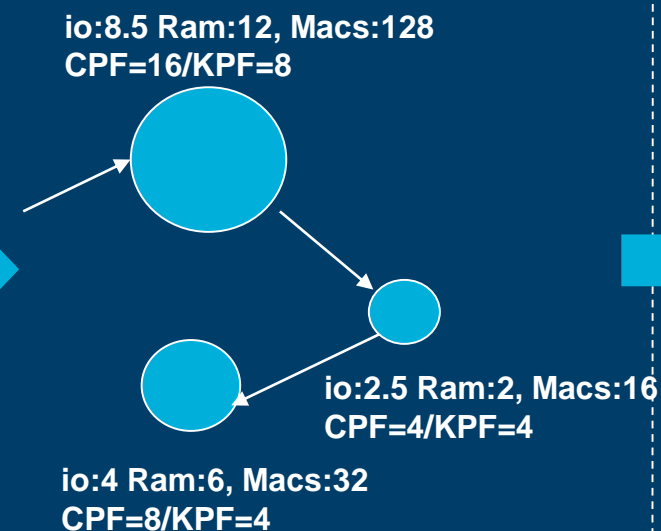
Model Files



## Resource Allocation

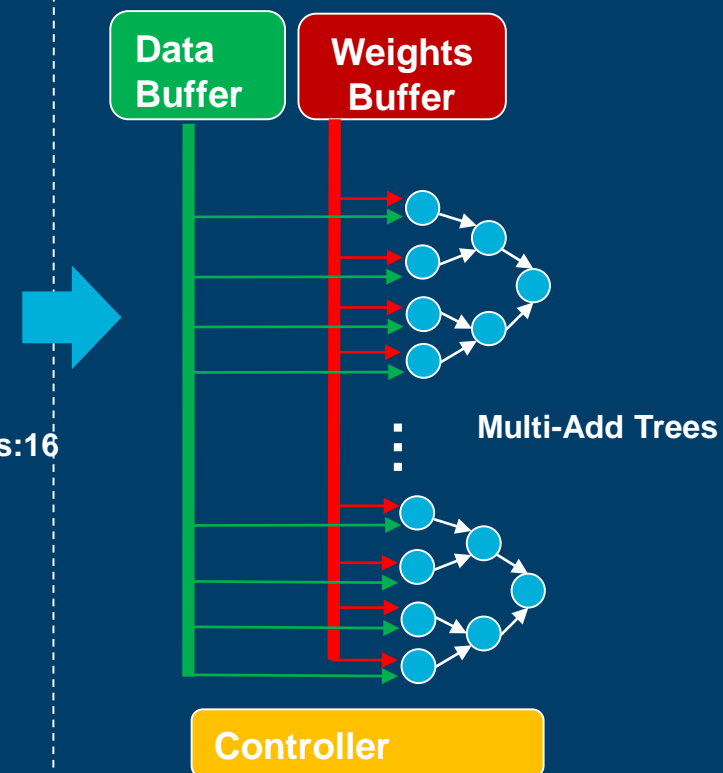
- I/O bandwidth, RAM and MACs allocation
- CPF/KPF calculation
- Pipeline balance

CPF: # of channels to parallelization  
KPF: # of kernels to parallelization



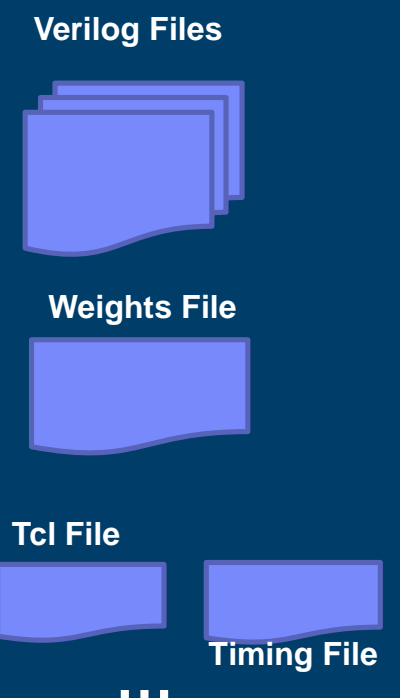
## Mapping & Route

- Mapping computation to customized BLAS (Vector multi-add, vector max/min, exp)
- Add controller logic
- Route pipelines



## Code Generation

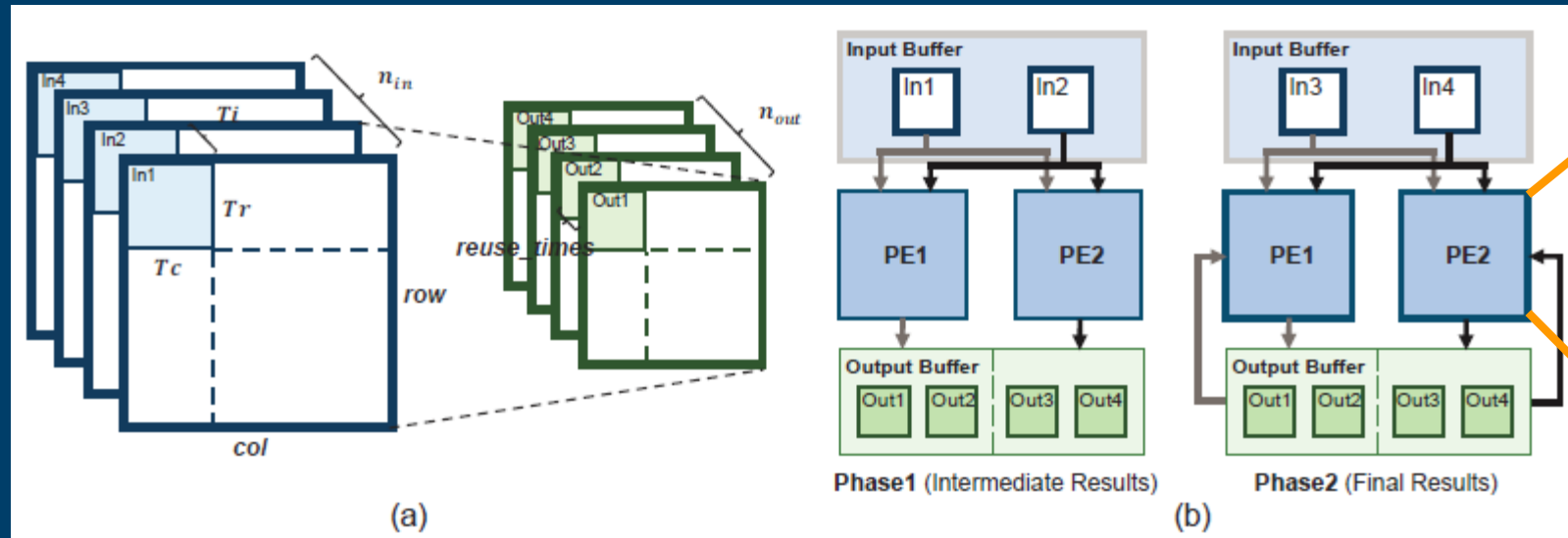
- Verilog files
- Weights file
- Tcl file for IP cores and timing file
- ...



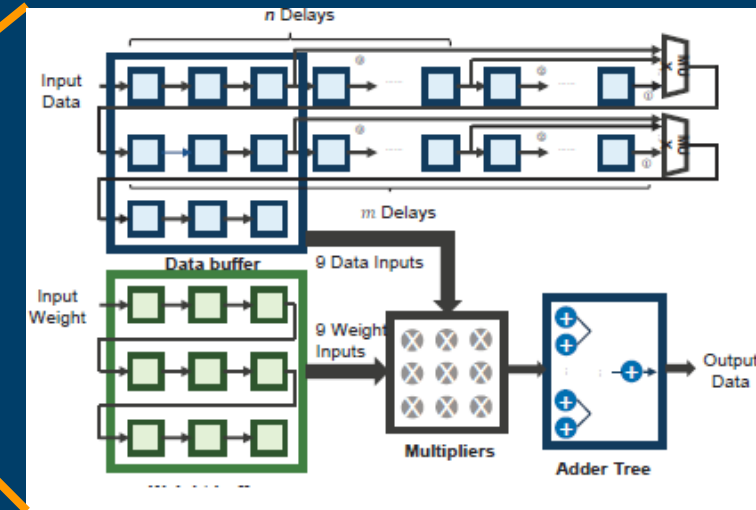
CPF: parallelism in processing multiple input feature maps for each output feature map.

KPF: parallelism in processing multiple output feature maps.

## Tiling and reuse of feature maps in CONV layers[1]

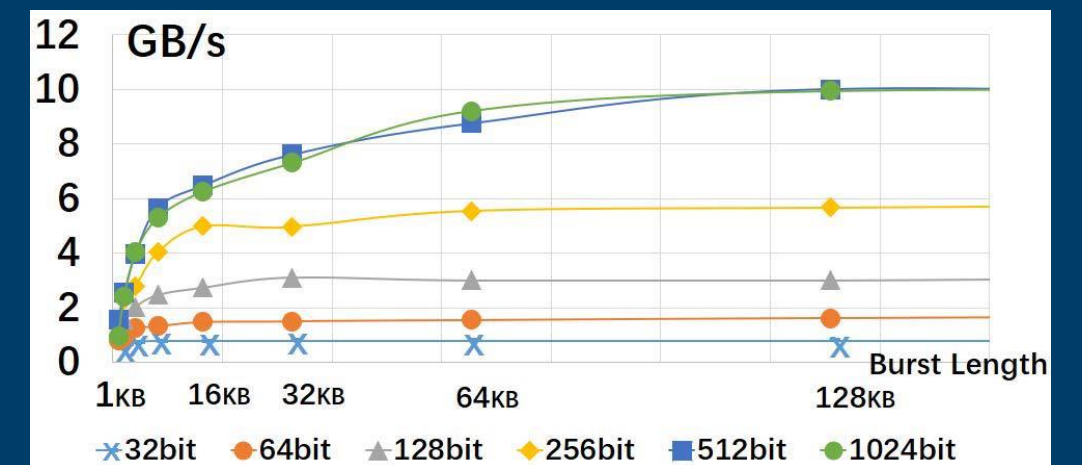


## Line buffer based convolver in PE



- Tiling and reuse of feature map could provide balance between the computation and bandwidth resource
- Usually suffer from the noncontiguous access of off-chip memory, results in lower bandwidth utilization, needs well data layout for DRAM space.
- Diversity convolution layers needs different tiling patterns, design exploration

## Effective bandwidth vs. burst length[2]



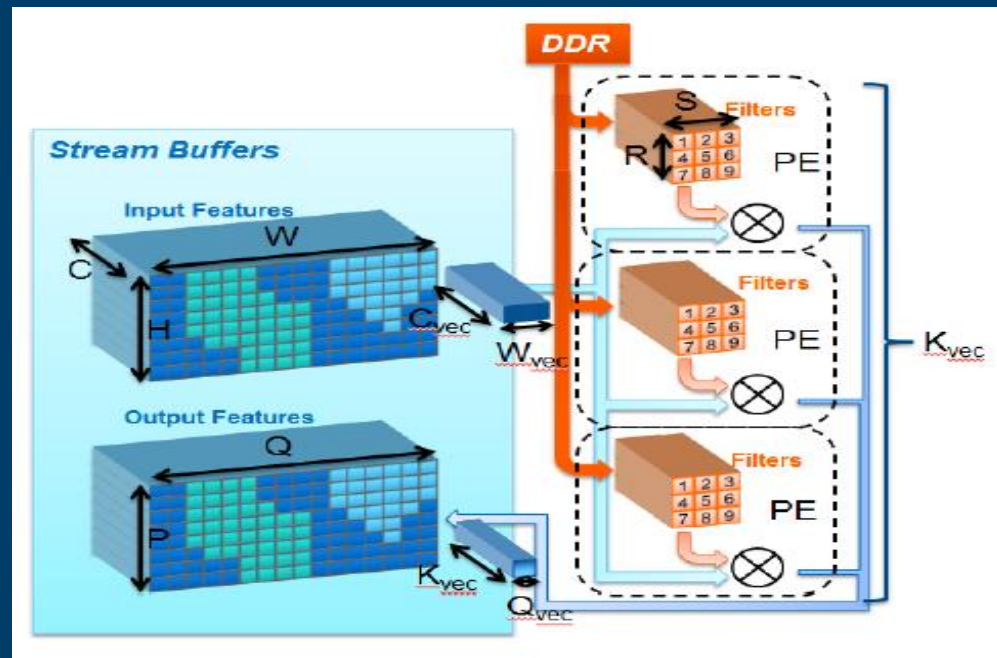
[1]Jiantao Qiu, JieWang, Song Yao, et al. Going deeper with embedded fpga platform for convolutional neural network. In Proc. of FPGA, pages 26–35. ACM, 2016.

[2]Chen Zhang, Zhenman Fang, Peipei Zhou, et al. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. In Proc. of ICCAD,2016.



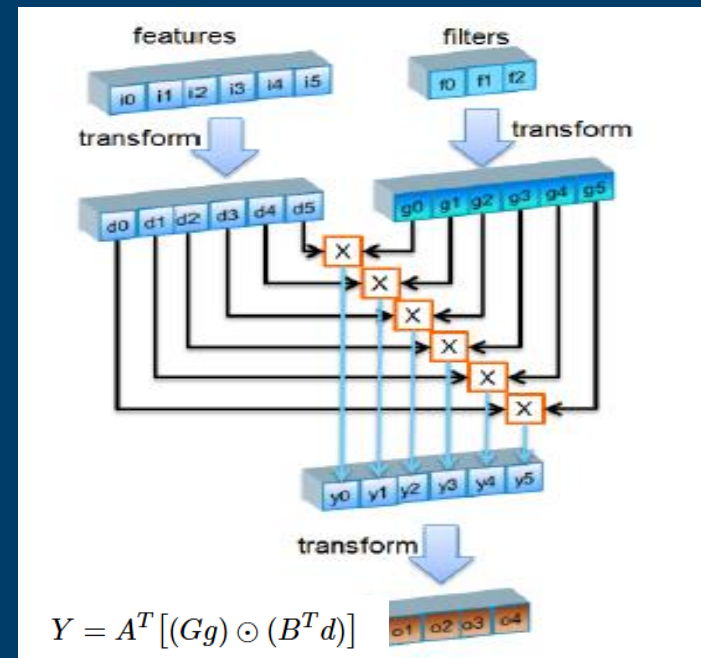
# Architectures of Convolution Acceleration (prior arts)

## On-chip memory caching



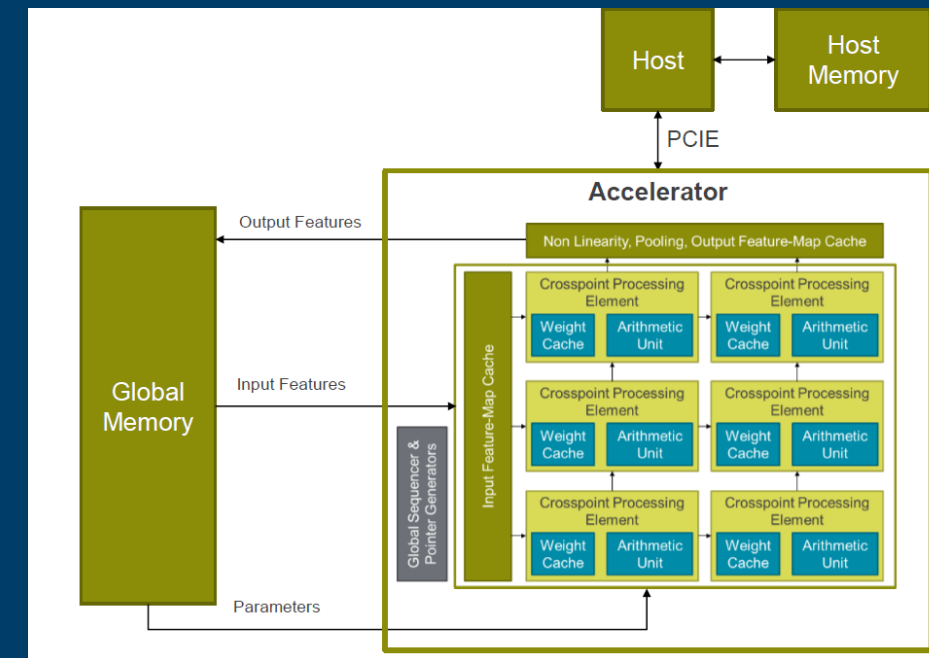
Caching all feature-maps on-chip to significantly reduce memory bandwidth by benefit larger on-chip memory in Intel's high-end FPGAs, and the convolution becomes a **compute-bound** problem. but **can not scale up large feature map (such as high resolution inputs) or low-end FPGAs.**

## Winograd accelerator F(4,3)



- Winograd is efficient for small size and stride kernels. A typical F(4,3) could reduce the multiplier from 12 to 6, 2x speed up.
- Also the transformed data for the filter can be pre-computed, it still needs 2x more bandwidth.
- Needs extra computation resource for A, B, and G operations.

## SuperTile for faster DSP



- Weights cached in DSP supertiles in fast slow domain
- Convolution operations time-folded to slow down data memory
- Forms the core of reconfigurable neural-network processors

[1] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, Gordon R. Chiu, An OpenCLTM Deep Learning Accelerator on Arria 10, FPGA'17



1) **An end-to-end automation tool** which provides an integrated design flow from deep learning frameworks to FPGA board-level implementations.

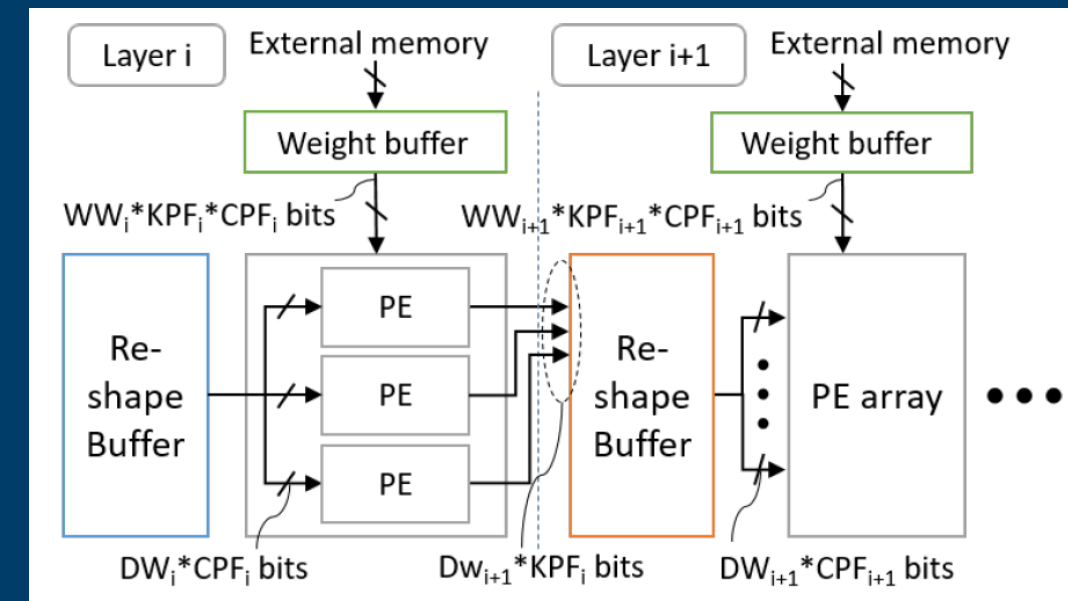
2) **A flexible support of quantization** to address the limited resource issues, Our design supports flexible quantization for weights and activations either within a layer or across layers in DNN. It also supports binary and ternary networks.

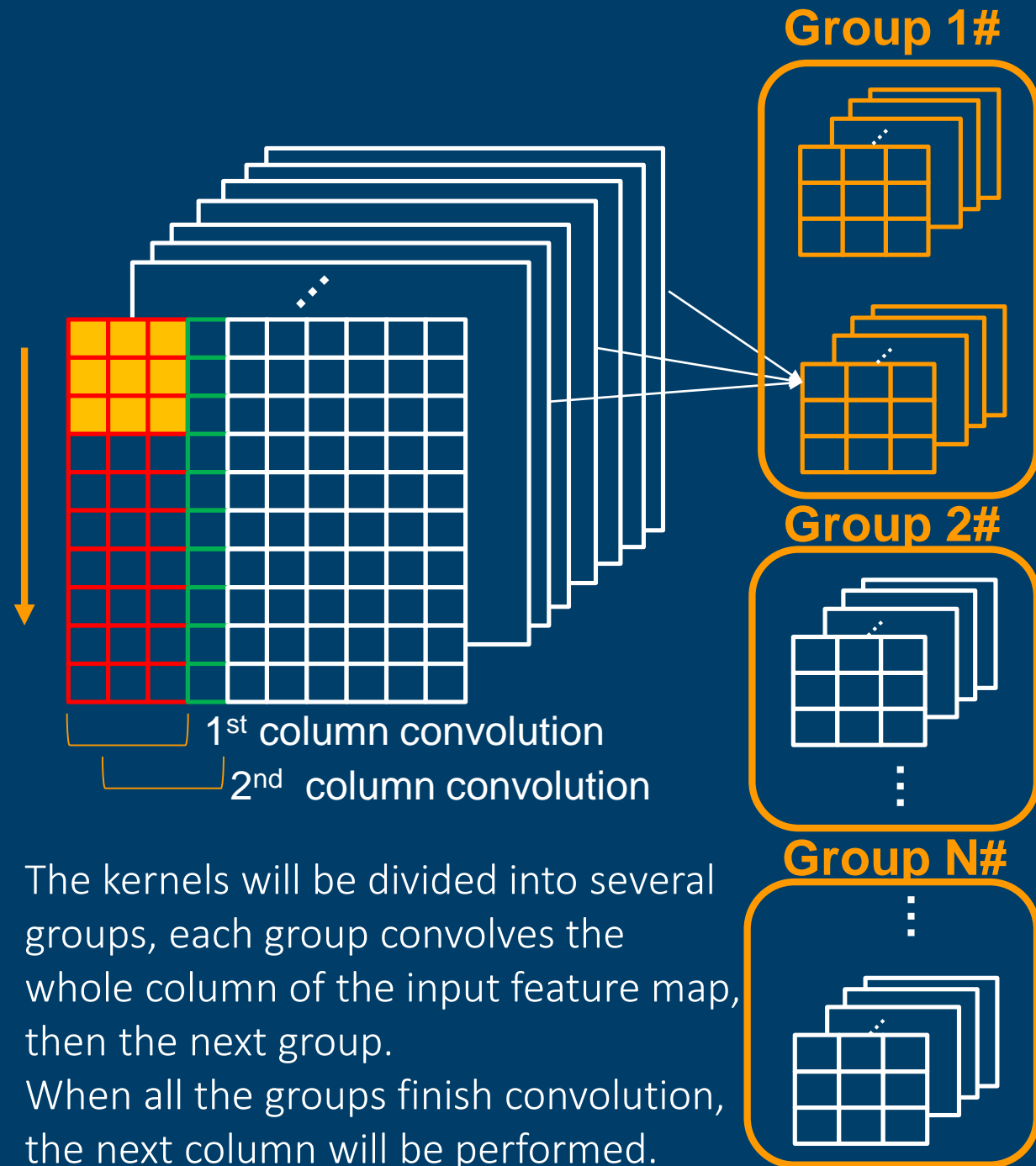
3) **A fine-grained layer-based pipeline architecture** that can achieve high throughput even without batch processing.

4) **An unified and flexible Processing Engine (PE)** that provides a two dimensional parallelism scheme for implementing major layers in DNNs including convolutional layer and fully-connected layer.

5) **An automatic resource allocation management scheme (A-REALM)** that provides resource allocation across network layers based on the external memory access bandwidth, data reuse behaviors, computation resource availability, and network complexity

Layer based pipeline structure





**Example:** kernel size is 3\*3, stride 1,

This design only need a ring buffer of 4 columns for the input feature map. If the feature map size is 224\*224, it can save up to 98.2% on-chip RAM (reduced from 224 columns to 4 columns).

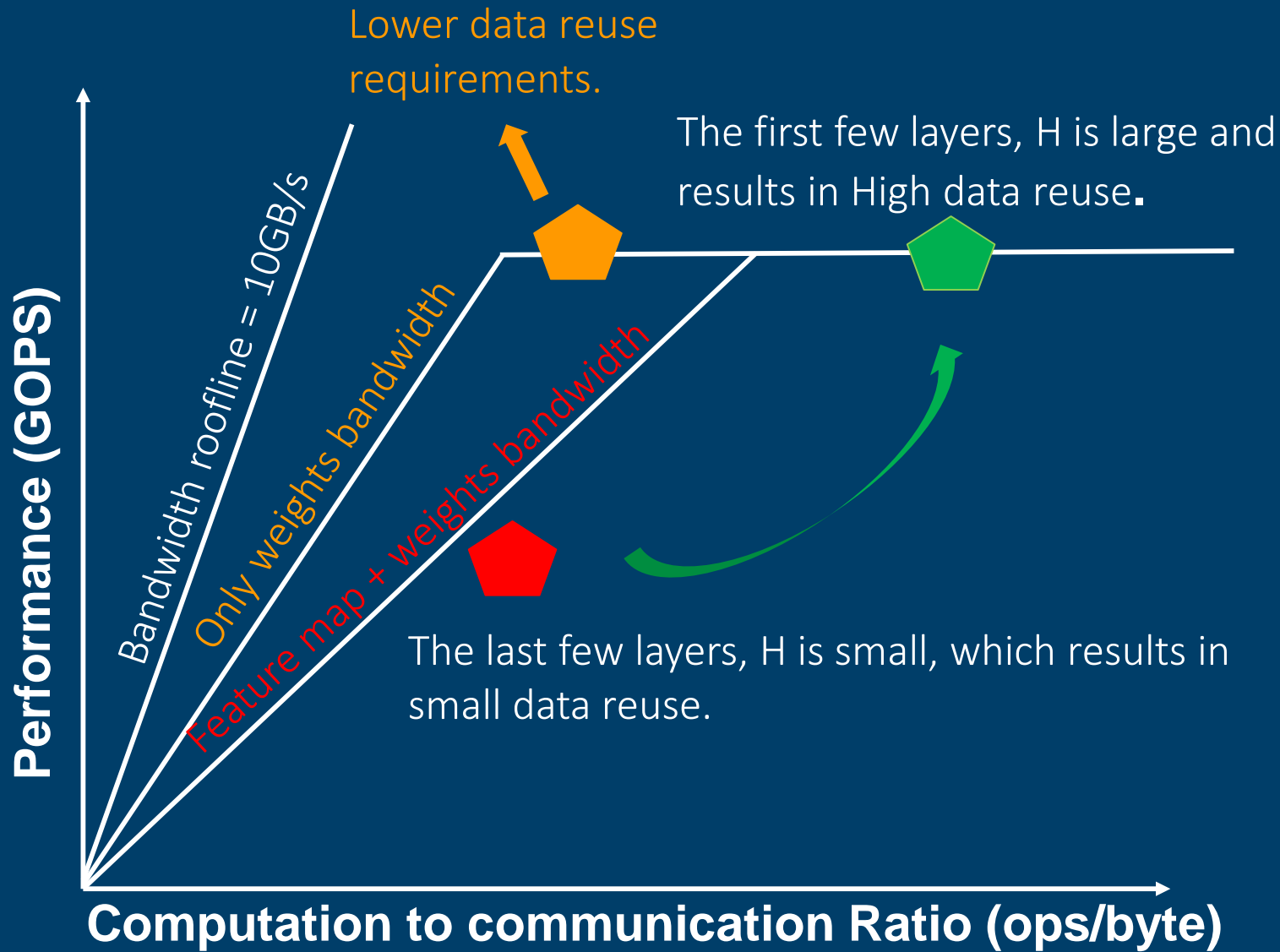
## Advantage of Layer plus Column based Pipeline:

- A layer-based pipeline architecture that can achieve high throughput for satisfying the overwhelming streaming input data in edge-computing applications (even without batch processing).
- Eliminate the bandwidth consumption of feature map load and restore, usually feature map load and restore is not efficient for bandwidth utilization because of its noncontiguous access.
- Column-based cache scheme can lower the latency caused by multi-layer pipeline stages, and it can also greatly saves memory space when deploying large-scale CNNs with high resolution image inputs.

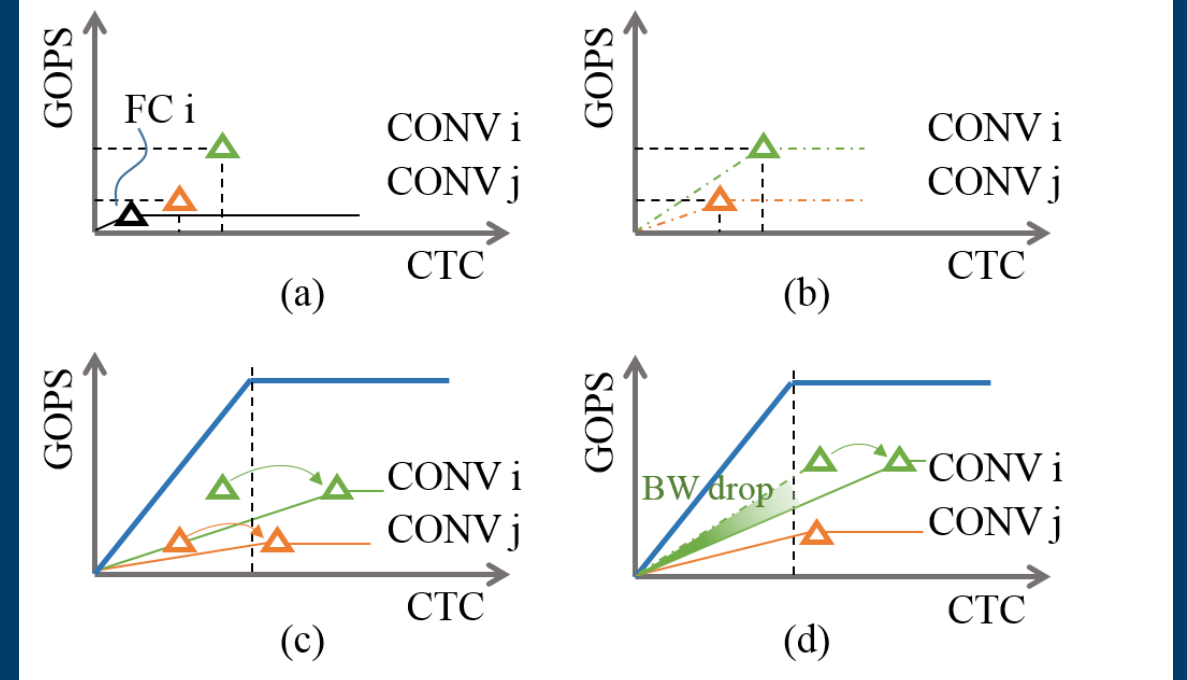
The kernels will be divided into several groups, each group convolves the whole column of the input feature map, then the next group.

When all the groups finish convolution, the next column will be performed.

# Extend Single Column to Multiple Columns for Higher Data Reuse



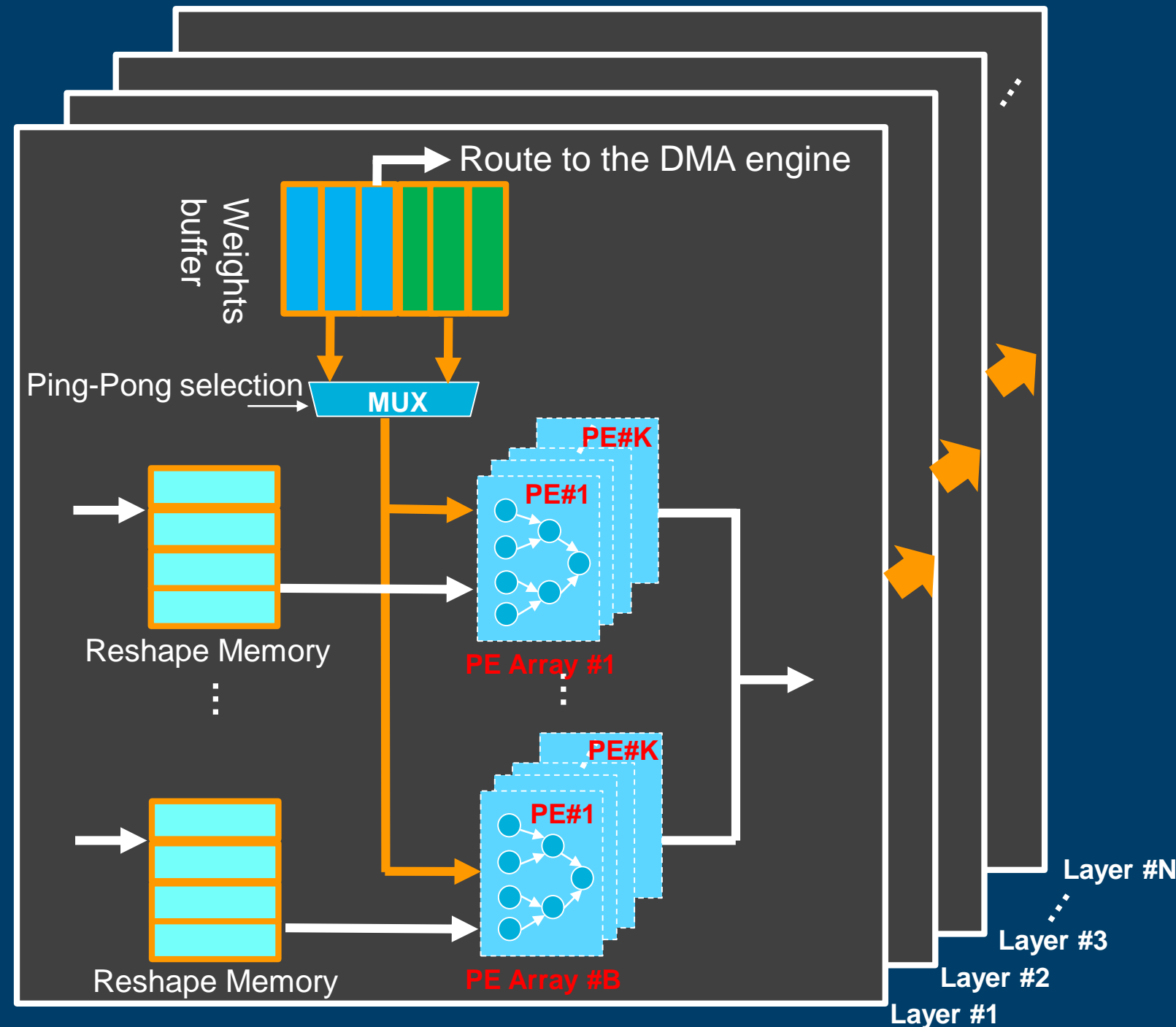
**Use multiple columns, needs more RAMs, Each kernel group convolves multiple column of the input feature map, then the next kernel group**



## Algorithm 2 Bandwidth resource allocation in A-REALM

- 1: Set available memory bandwidth:  $BW_{total}^{conv}$
- 2: Set available on-chip memory for input feature map:  $mem_{total}^{rb}$
- 3: Set single DSP's bandwidth usage:  $BW_R$
- 4: Initialize  $Col_i = 1$ ; size of reshape buffer (e.g.  $width_i^{rd}$ ,  $depth_i^{rd}$ , and  $width_i^{wr}$  according to  $KPF_i$  and  $CPF_i$ )
- 5: Allocate bandwidth  $BW_i$  for layer  $i$  to best satisfy its  $R_i$  demand:  $BW_i = \frac{PF_i \times BW_R}{H_i^{out} \times Col_i}$
- 6: **while**  $\sum_{i=1}^n BW_i \geq BW_{total}^{conv}$  //CONV layer bandwidth overuse
- 7:     Select layer  $i$  in CONV layer with maximum  $BW_i$
- 8:      $depth_i^{rd} + = \frac{H_i^{in} \times C_i^{in} \times Stride_i}{CPF_i}$ ,  $depth_{i+1}^{rd} + = \frac{H_i^{out} \times C_i^{out}}{CPF_{i+1}}$
- 9:     **if**  $\sum_{i=1}^n f(width_i^{rd}, depth_i^{rd}, width_i^{wr}) \leq mem_{total}^{rb}$
- 10:          $Col_i = Col_i + 1$  //Cache one more column
- 11:          $BW_i = BW_i \times \frac{Col_i}{Col_i + 1}$
- 12:     **else**
- 13:         //restore if no enough memory
- 14:          $depth_i^{rd} - = \frac{H_i^{in} \times C_i^{in} \times Stride_i}{CPF_i}$ ,  $depth_{i+1}^{rd} - = \frac{H_i^{out} \times C_i^{out}}{CPF_{i+1}}$
- 15:         **break**
- 16:     **endif**
- 17: **endwhile**

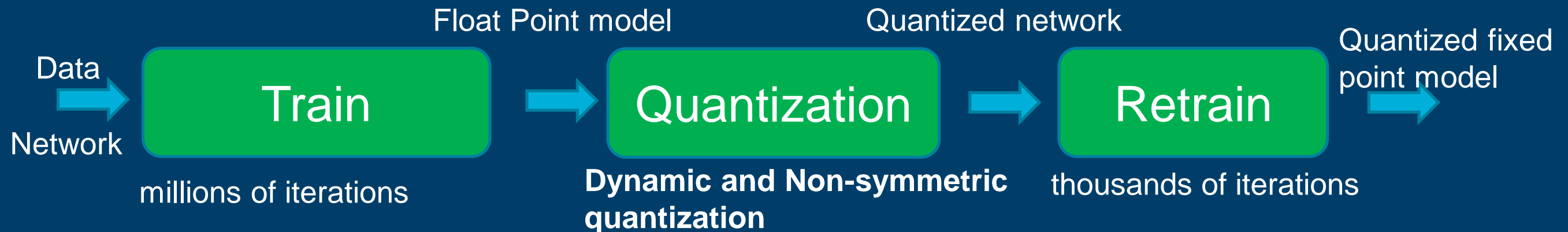
# Layer based Pipeline Structure Mapped to FPGA



- The size of the PE (processing unit) is decided by the CPF, if CPF is 4, the PE will be a 4-elements vector Multiply-Add-Tree, and the number of PEs is decided by the KPF.
- We also use the double buffer to cache the weights from the off-chip DRAM, each double buffer will be routed to a DMA channel for weights fetching.
- The data in the reshape memory will be broadcasted to each PE, and each stream has its own data buffer.
- The weights will be dispatched to each PEs. All PE arrays share the weights.
- The K outputs will keep together to feed to the next layer.
- The number of PE arrays depends on the number of streams (batch size).

- **Quantization is always import for FPGA**

- Fixed point multiplier is efficient for FPGA, such as 16bits/8bits.
- Reduce the DDR I/O bandwidth requirement, board size & cost & power budget.
- Dynamic quantization is essential for deep learning.
- Non-symmetric quantization, activations and weights use different quantization/bitwidth, usually activation is more sensitive to the numerical precision than weight.



**Use quantized weights in forward path while keeping float weights when updating gradients in the backward path.**



# AccDNN Performance Results



## Comparison : Embedded FPGAs for edge-devices

Reference	[1]	[2]	DNNBuilder
Categories	Edge-computing platforms		
FPGA chip	Zynq XC7Z045	Zynq XC7Z045	Zynq XC7Z045
Frequency	150 MHz	100 MHz	200MHz
Network	VGG	VGG	VGG
Precision	Fix16	Fix16	Fix16 (Fix8)
DSPs (used/total)	780/900	824/900	680/900
DSP Efficiency	44.0%	69.6%	96.2%
Performance (GOPS)	137	230	262 (524)
Power Efficiency (GOPS/W)	14.2	24.4	36.4 (72.8)

### Zynq XC7Z045

- LUT: 218,600
- FF: 437,200
- BRAM: 545
- DSP: 900

**Peaking at  
526 GOPS**

- [1] J. Qiu et al. Going deeper with embedded FPGA platform for convolutional neural network. In *FPGA*, 2016.  
 [2] Q. Xiao et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *DAC*, 2017.

## Comparison : High-performance FPGAs for cloud computing

Reference	[5]	[6]	[10]	DNNBuilder
Categories	Cloud-computing platforms			
FPGA chip	Arria10-1150	Arria10-1150	Stratix-V GXA7 + CPU	KU115
Frequency	303 MHz	385 MHz	200 MHz & 2~3 GHz(CPU)	235 MHz
Network	Alexnet	VGG	Alexnet	VGG
Precision	Float16	Fix16	Fix16 in FPGA	Fix16 (Fix8)
DSPs (used/total)	2952/3036	2756/3036	512/512 in FPGA	4318/5520
DSP Efficiency	77.3%	84.3%	-	99.1%
Performance (GOPS)	1382	1790	781	2011 (4022)
Power Efficiency (GOPS/W)	30.7	47.8	-	90.2 (180.4)

### KU115

- LUT: 663,360
- FF: 1,326,720
- BRAM: 2160
- DSP: 5520

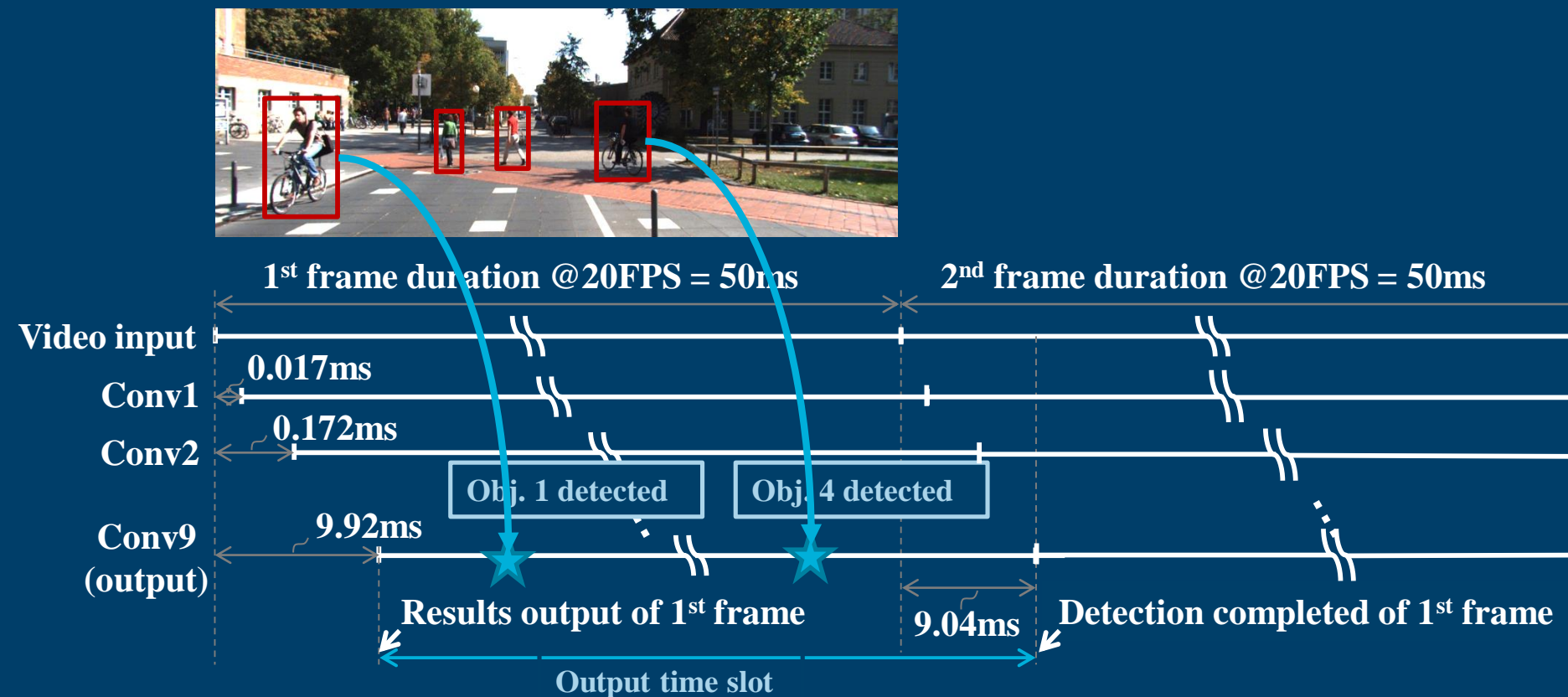
**Peaking at  
4218  
GOPS**

- [5] U. Aydonat et al. An OpenCL deep learning accelerator on Arria 10. In *FPGA*, 2017.  
 [6] J. Zhang et al. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In *FPGA*, 2017.  
 [10] H. Zeng et al. A framework for generating high throughput CNN implementations on FPGAs. In *FPGA*, 2018.

# Faster Response of Real-time Object detection for Video



- Start the computation immediately after the video frame input.
- Complete the detection immediately after the video frame finished, only 9ms delay.
- With the same computation capability, recurrent accelerator should have ~45ms, the detection delay is reduced up to 80.1%.
- For some critical objects (strict response time), could be detected immediately after the object streamed into the accelerator.





# Use Case: Traffic Sign Detection in Night Environment

- Detection for traffic sign as far as possible and in night time: Extremely small object detection, 15pixels\*15pixels, should detect ahead 60m.
- Different weather situation , shape distortion due to the angle between the camera and sign.
- Low-end embedded system, 25FPS requirements.



**Ideal case**



**Complicated illumination**

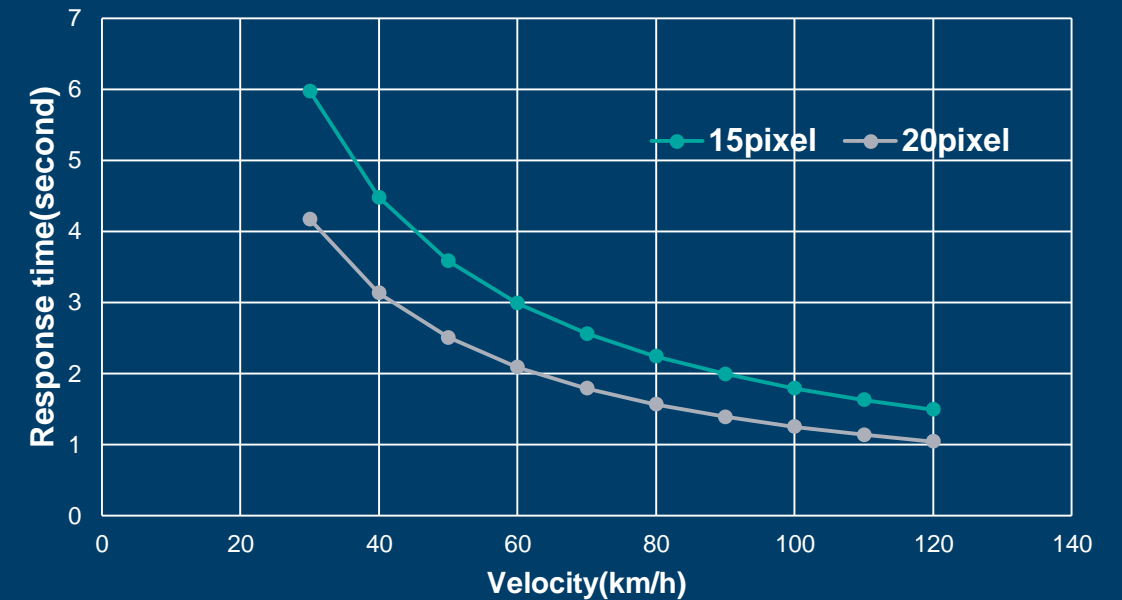


**Shape distortion**

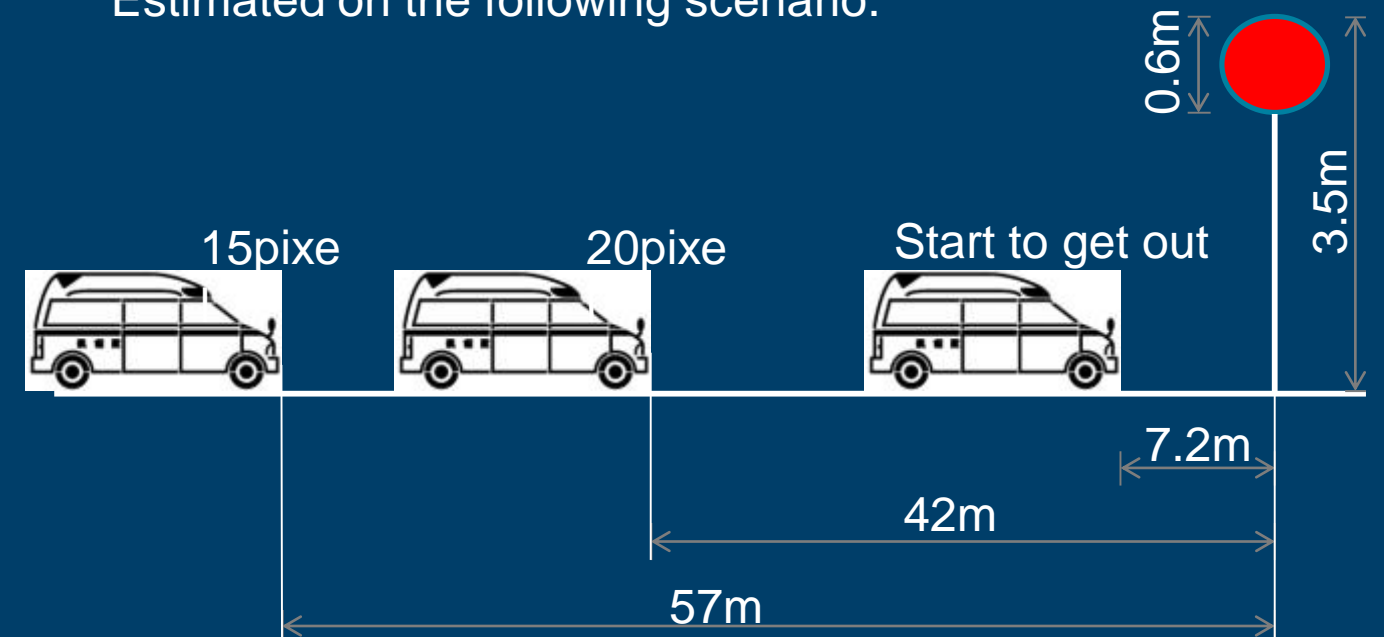


**Extremely small object**

**Reponse time vs. Velocity**

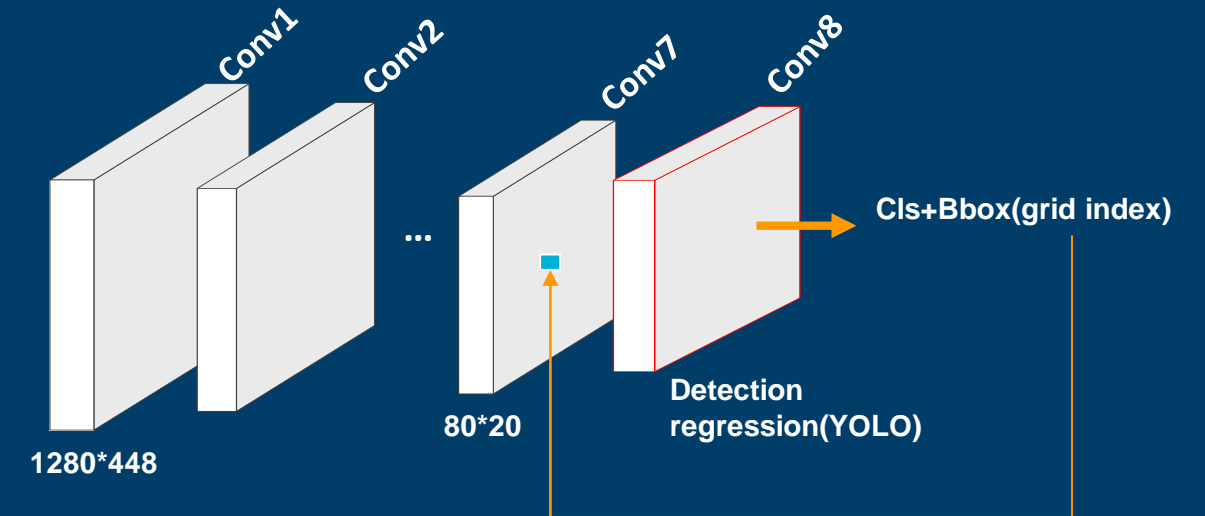


Estimated on the following scenario.



## Performance efficient light neural network models

- Use deep CNN to handle shape distortion, illumination invariance, and use the low level feature to detect with higher spatial resolutions
- High resolution input ( $1280 \times 448$ ) without any down-sampling for small object detection.
- Manual anchor selection, object size is range from  $15 \times 15$  to  $70 \times 70$ , we select 8 square anchors, from 1.0 to 4.5 with interval of 0.5 at feature map  $1/16$ .
- Detection + classification, detect major categories(warning, prohibitory, ...) then classify its sub-category/meaning, and improve the precision as well.



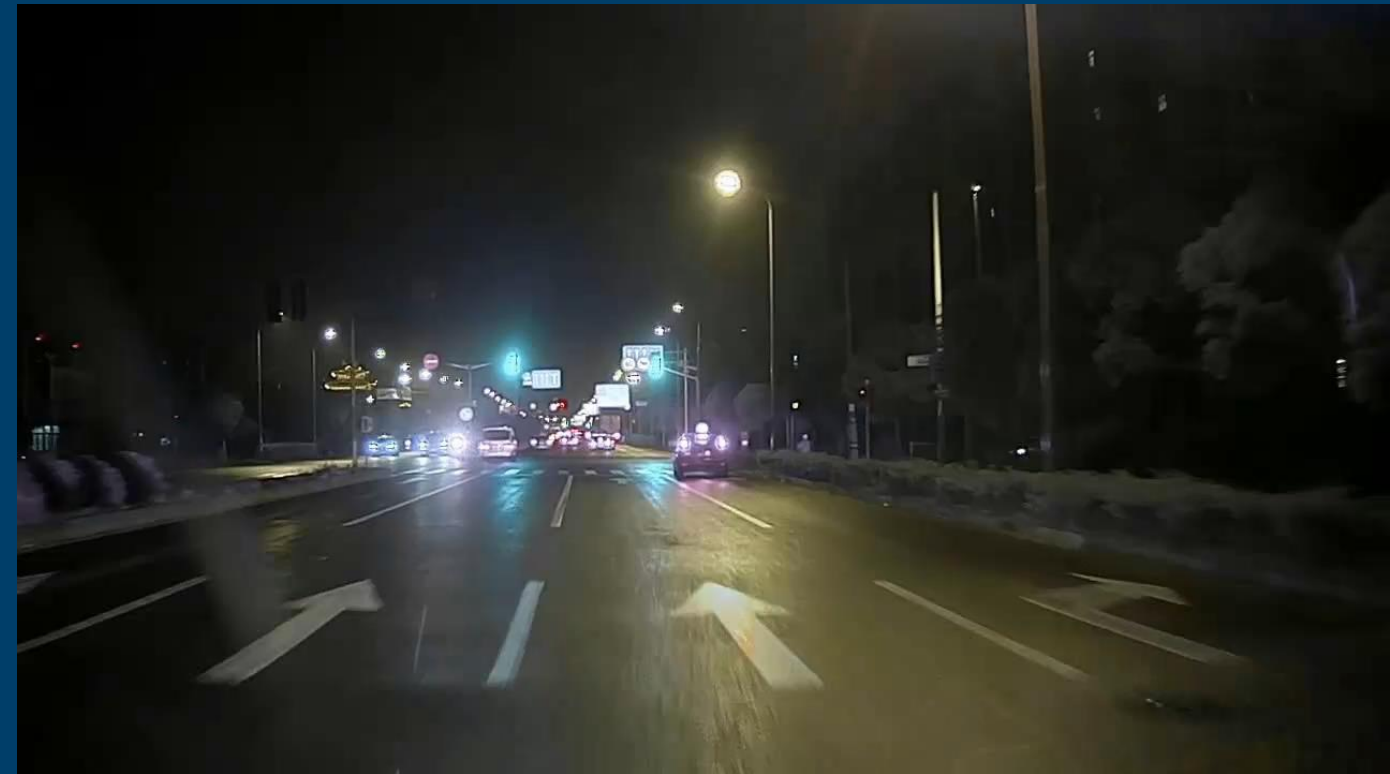
# Use Case: Traffic Sign Detection in Night Environment



Mapping two accelerators (detector plus classifier) to a single-embedded FPGA (Zynq XC7Z045), timing enclosure in 200MHz working frequency in this 28nm chip without sophisticated manual timing adjustment.

Network	Resource Utilization			Complexity (Giga oper)	Speed (images/s)	Performance (GOPs)	DSP efficiency
	LUT(218600)	BRAM(545)	DSP(900)				
Detector	52921(25%)	278(51%)	604(67%)	11.2	18	201.6	83.7%
Classifier	27451(13%)	93(17%)	140(16%)	0.137	161	44.1	78.8%
Total	109223(50%)	473(87%)	744(83%)	---	27.7	245.7	82.3%

- Use the Kalman filter to do the on-line object tracking, using real-time velocity information.



We provide a try version with limited function in cloud (<https://crl.ptopenlab.com:8800/accdnn>). Anyone would like to have a try, please contact us for the account.

## Step 1: Choose targeted hardware and

Select the percentage of your total FPGA resource could be used for accelerator.

PL SIDE MEMORY	LUTS	18KB BLOCK RAMS	DSP SLICES
DDR3 SODIMM 1GB	218600	1090	900

Select Target FPGA resource usage

- 30%
- 30%
- 40%
- 50%
- 60%
- 70%
- 0%

Progress bar: Set Name (checked), Select Hardware (current), Upload Network Model, Estimate Resource

## Step 2: Upload DNN model file

Should upload the .prototxt file

try-yolo-kitt-640-relu-512-quantized.prototxt

Constraints

- Support model trained by Caffe framework. To keep the precision, we suggest you to use the Caffe Ristretto to quantize your model, and upload the quantized model.
- Support convolutional layer, max pooling layer, fully connected layer, batch normalization layer and ReLU.
- The total number of convolutional and fully connected layers in the network should be less than 10.

Progress bar: Set Name (checked), Select Hardware (checked), Upload Network Model (current), Estimate FPGA Resource, Upload Weight File

## Step 3: Analyze DNN, optimize resource allocation and predict the performance of

PowerAI Interference Engine

45742434@qq.com | Current Language: English

← Create New Project

Overview

NAME YOLO-TINY-MODEL  
USAGE 70%  
BOARD Xilinx ZC706 Evaluation Board  
NETWORK MODEL FILENAME tiny-yolo-kitti-640-relu-512-quantized.prototxt

Estimated Result

Batch Size: 1

Auto-optimized results, including the resource usage, performance.

Throughput: 27.74325284090909 images/s, DSP efficiency: 99.27%

NAME	TYPE	CPF	KPF	MACS	DSP	WEIGHTS	BRAM16E	DELAY(US)	DDR_BW(MB/S)
conv1	Convolution	4	4	129761280	20	576	25	32440.32	145.45
pool1	Pooling		16	0	0	0	22	1126.40	
conv2	Convolution	8	4	259522560	36	4608	34	36044.80	581.82
pool2	Pooling		32	0	0	0	29	261.60	

optimize

Set Name Select Hardware Upload Network Model Estimate FPGA Resource Upload Weight File

Prev Next

## Step 4: Upload the weights file (caffe .caffemodel file) to generate the Accelerator IP

PowerAI Interference Engine

45742434@qq.com | Current Language: English

← Create New Project

Overview

NAME YOLO-TINY-MODEL  
USAGE 70%  
BOARD Xilinx ZC706 Evaluation Board  
NETWORK MODEL FILENAME tiny-yolo-kitti-640-relu-512-quantized.prototxt

Upload Weight File

Please upload caffe trained weight .caffemodel file.

Choose File

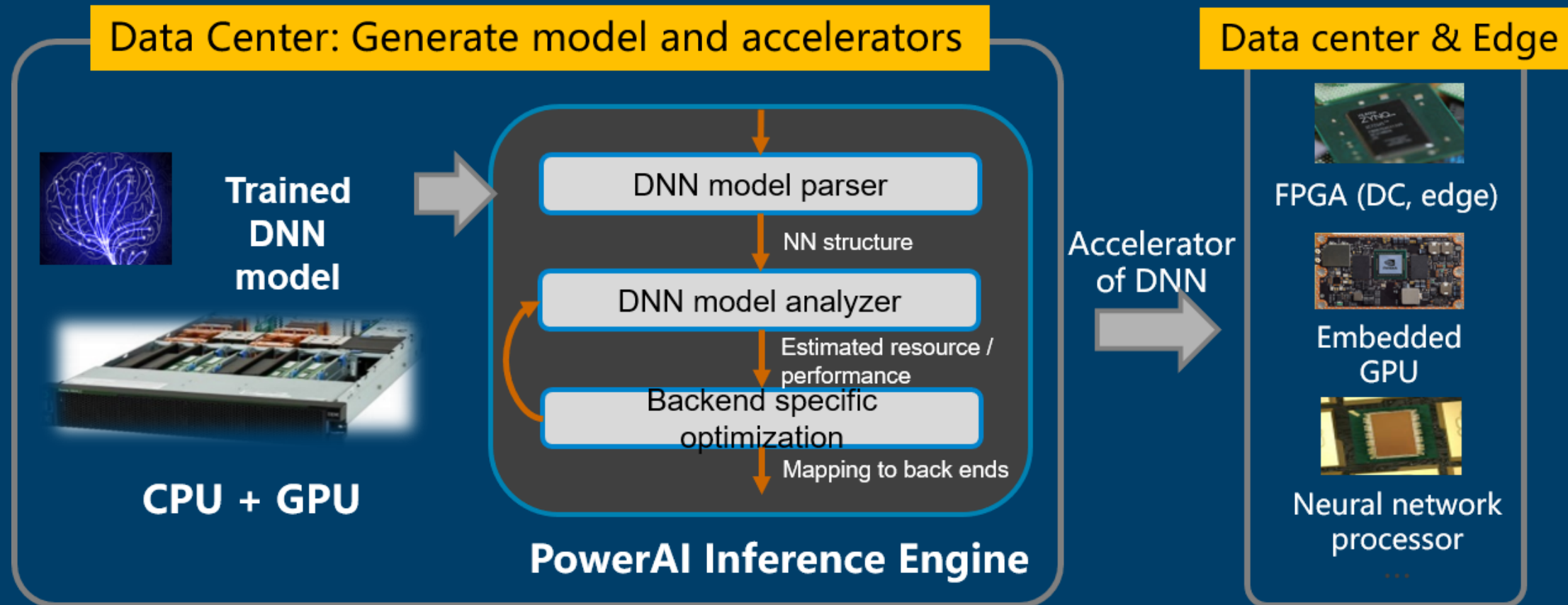
tiny-yolo-kitti-640-relu-512.caffemodel

Set Name Select Hardware Upload Network Model Estimate FPGA Resource Upload Weight File

Prev Create



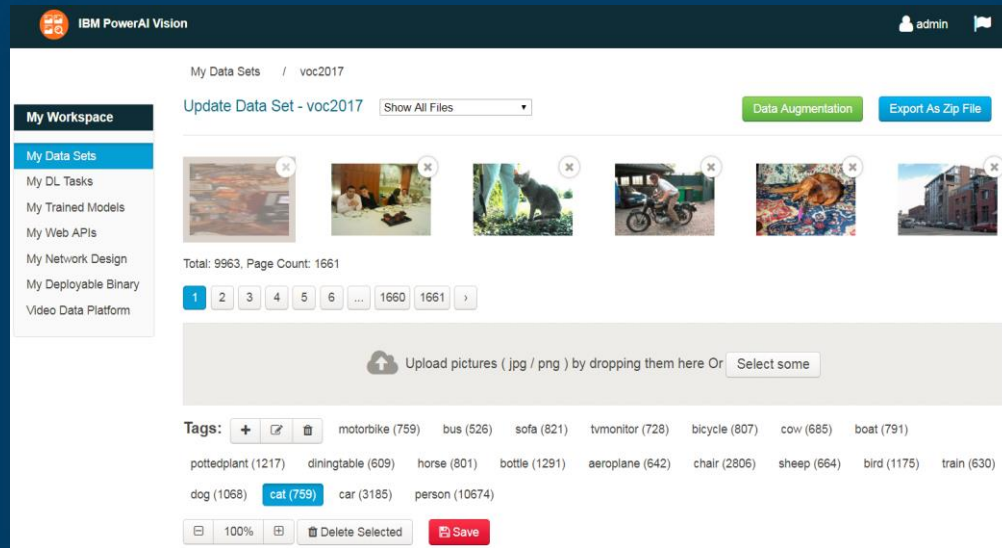
## IBM Inference Engine : Analyze, Generate, and Optimize Deep Neural Network Accelerator automatically



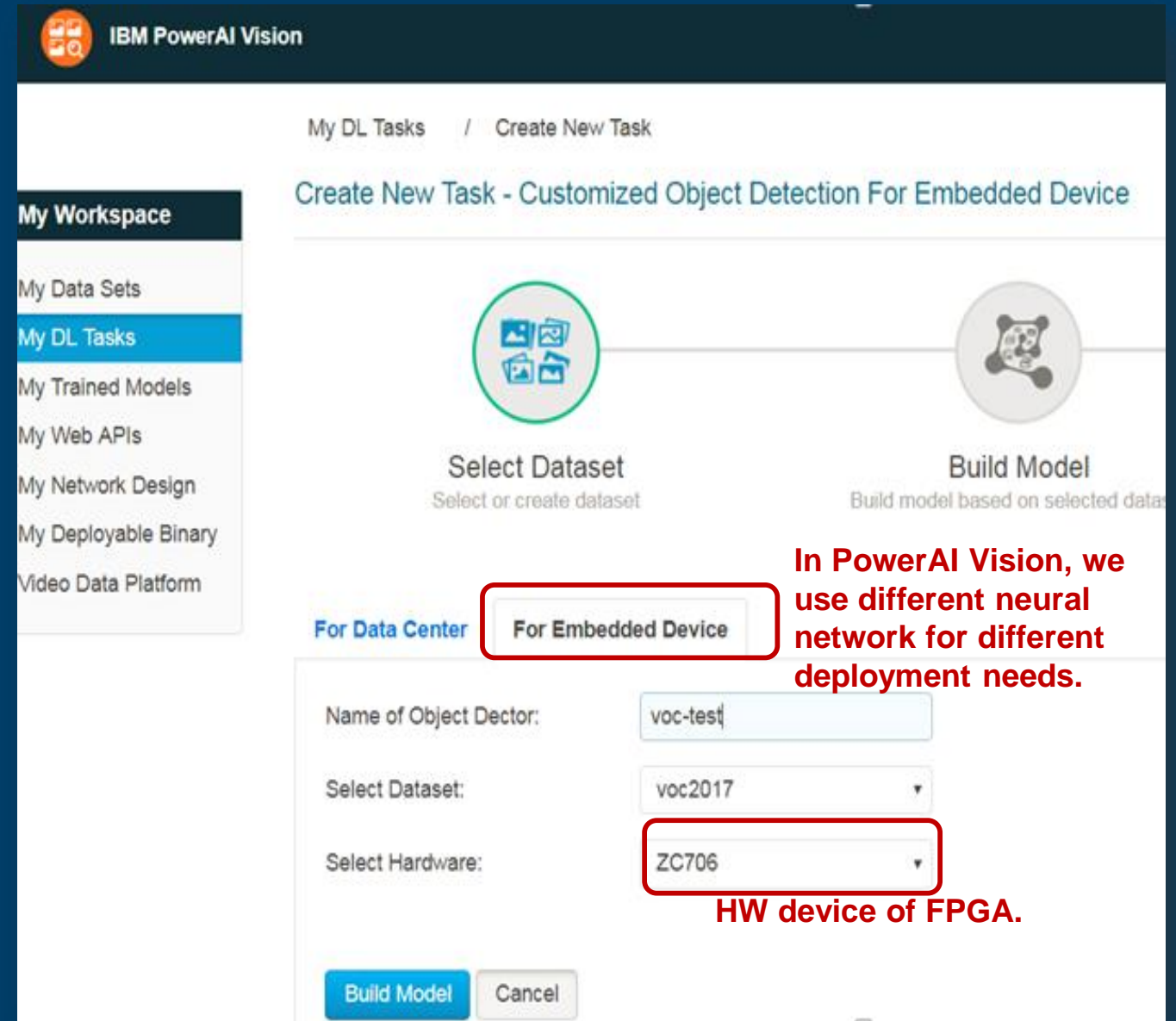
**Our Goal: PowerAI Inference Engine is toolset to provide all these capabilities for inference **automatically**.**

- Analyze DNN model and predict resource requirement, performance
- Convert and generate code packages based on different backend implementation for different HW architecture
- Optimize DNN model with different neural network compression technologies

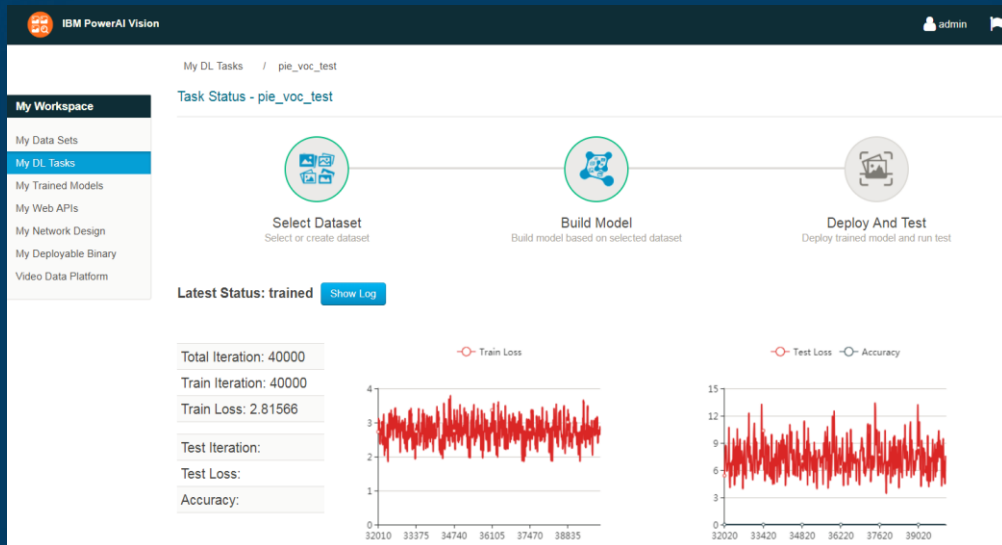
## Step 1: Upload data set and labeling



## Step 2: Create new task, choose targeted embedded platform



## Step 3: Training/visualization



(Demo video: <https://ibm.box.com/s/um180x4nj0f8uyz246goagrvq8ukj144>)



**Step 4: Click *simulate deploy* to deploy the trained model onto GPU server for simulation testing**

**Step 5: Upload an image to do simulation testing (on GPU server or FPGA server in future)**

**Allow user to do simulation testing for the trained model on GPU server**

Name / Id	Usage	Categories	Accuracy	Created At	Operation
pie_voc_test-model	Object Detection	sofa, chair, person, car...	0.43257	2018-05-24 00:24:21	Simulate Deploy Actions
test-model	Object Detection	prohibitionsign, mandato...	0.00008	2018-05-22 18:26:30	Simulate Deploy Actions



Upload a picture to classify:

Or input image URL to classify:

Result:

train: 0.107

person: 1



**Step 7: Generated binary is ready for downloading onto FPGA device**

**Step 6: Click to convert and generate deployable binary for FPGA device**

**Download generated binary package**

Name / Id	Device Type	Status	Description	Created At	Operation
ec6fa03a-1d1a-407f-86d8-ff54c710b811	ZC706	uploaded		2018-05-22 20:48:52	Actions
ea05ab03-6283-4694-a5d8-d4fe8406bf61	ZC706	uploaded		2018-05-17 17:09:44	Rename Delete Download



Name / Id	Usage	Categories	Accuracy	Created At	Operation
pie_voc_test-model	Object Detection	sofa, chair, person, car...	0.43257	2018-05-24 00:24:21	Actions
test-model	Object Detection	prohibitionsign, mandato...	0.00008	2018-05-22 18:26:30	Rename Delete Export As Zip File Generate Deployable Binary
image-class-model	Image Classification	damaged, normal	0.97917	2018-05-22 15:26:56	Simulate Deploy Actions
detector-model	Object Detection	sofa, chair, person, car...	0.47253	2018-05-19 11:35:15	Simulate Deploy Actions



**Xilinx ZC706 Evaluation Board**



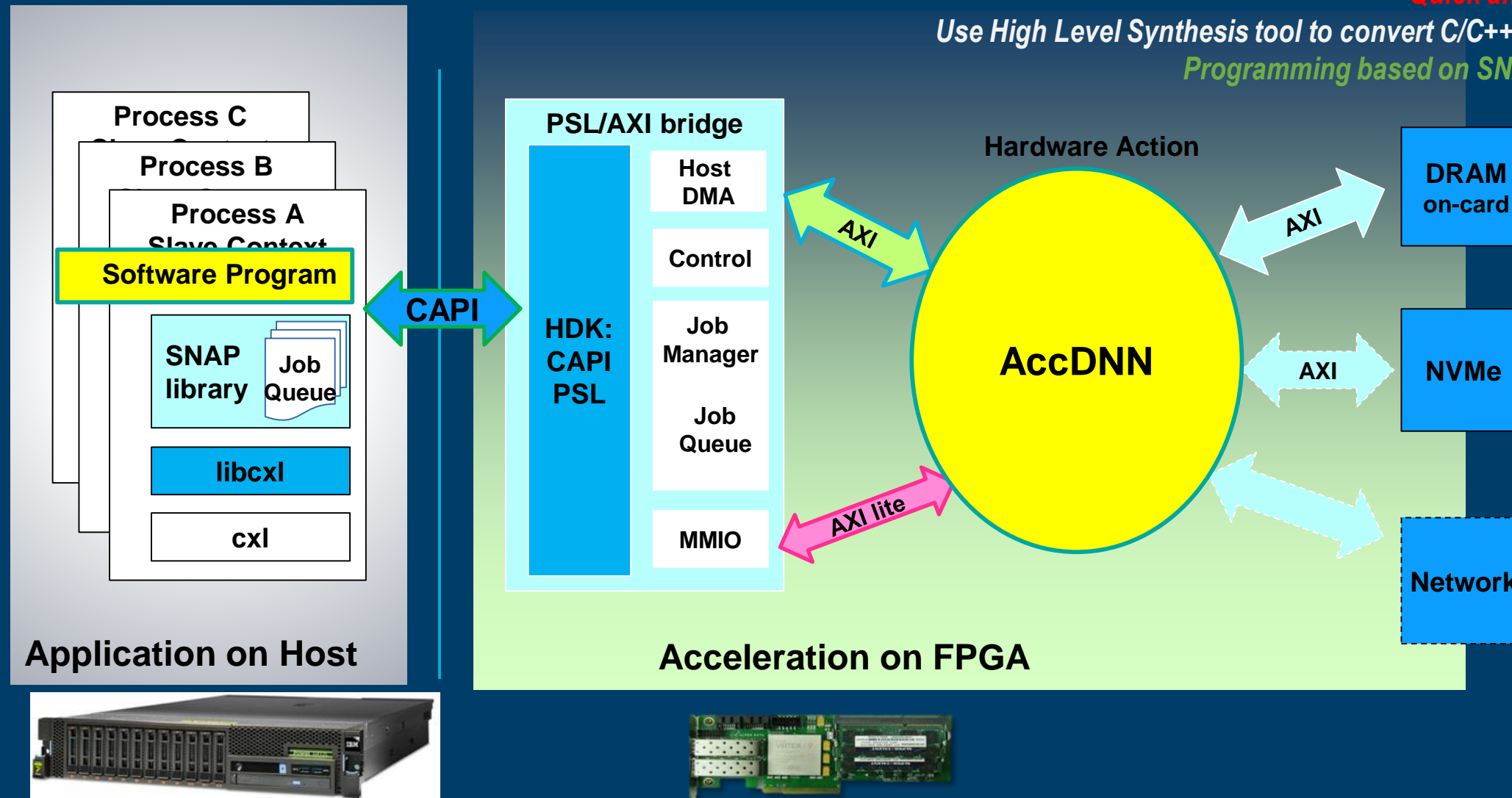
**DeepRed, full SDK available**

## SNAP framework: Spend Time on Algorithms Not on Coding

<https://github.com/open-power/snap>

*Quick and easy developing platform*

*Use High Level Synthesis tool to convert C/C++ to RTL, or directly use RTL Programming based on SNAP library and AXI interface*



# Hybrid Extremely Low Bit-width Neural Network (ELB-NN)

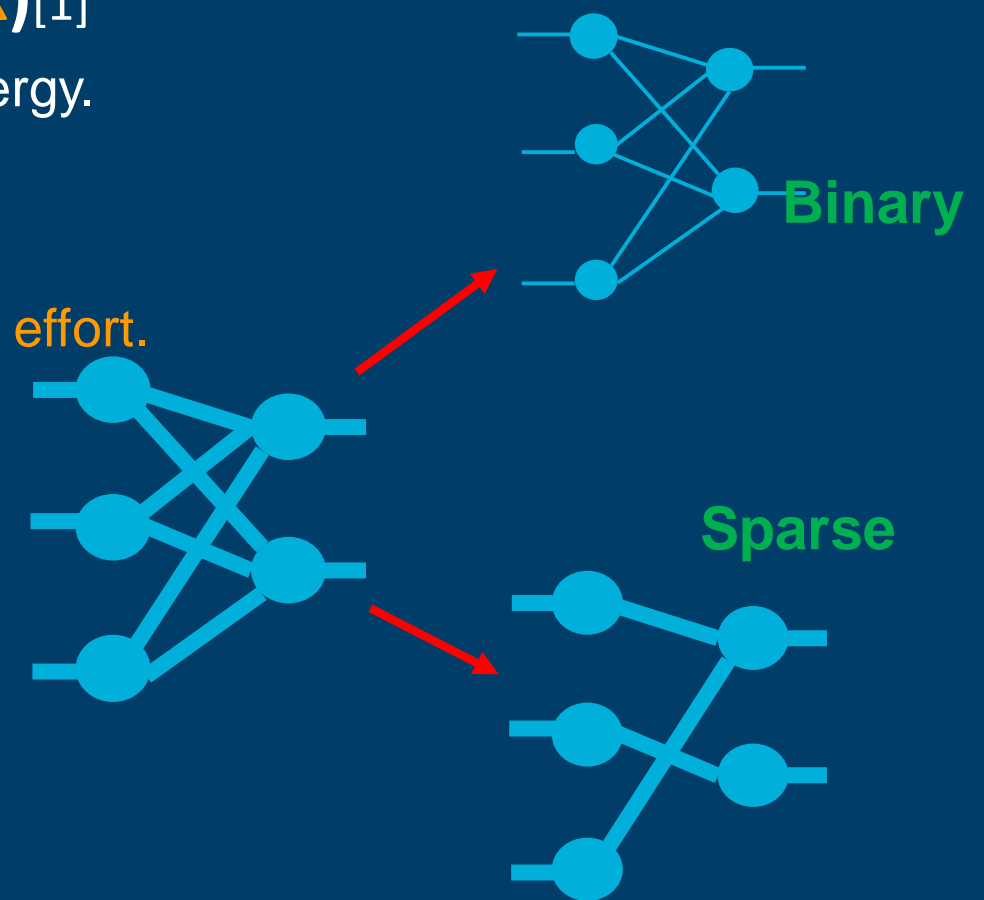
This work has been published in FPL'2018, Dublin, Ireland

## Pruning, quantized representation and Huffman coding (**2~3x**)<sup>[1]</sup>

- Significantly reduce the model size, I/O bandwidth, and data transmission energy.
- Without loss of accuracy.
- Still requires high precision computation – not save computation too much
- Extra cost of sparse representation, unregulated computation, extra decoding effort.

## Binary or ternary neural network (**20x+**)<sup>[2][3][4]</sup>

- Largely reduce the model size and I/O bandwidth.
- Extremely higher throughput
- Without multiplier, only use xnor plus popcount (only for binary)
- Performance of full binary network drops from 57.1 to 35.1%, Alexnet at larger scale network.
- Still keep float precision of activations in ternary network to maintain accuracy, which is not efficient in FPGA.



[1] Han et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", ICLR 2016

[2] Courbariaux et al. "Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1"

[3] Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks"

[4] Fengfu Li et al. "Ternary Weight Networks"

## FPGA on edge device is lack of computation resource, on-chip memory and bandwidth because of the strict constraints on power budget, form factor, and cost.

- Scarce DSP block, even 7045(900 DSPs) can only provide 720GOPS(INT8@200MHz) peak performance.
- Limited on-chip memory, results in frequent data swap.
- Limited DDR bandwidth, usually less than 32bits DDR3, even share the DDR width PS side.

COMPARE	Reset	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045
Logic Cells (K)		74	85	125	275	350
Block RAM (Mb)		3.3	4.9	9.3	17.6	19.1
DSP Slices		160	220	400	900	900
Maximum I/O Pins		150	200	250	362	362
Maximum Transceiver Count		4	-	4	16	16

<https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productTable>



## ELB-NN neural networks adapt AI to edge-devices perfectly

- ELB-NNs use lower bit-width of weights & activations (less comp. & mem. demands)
- Embedded FPGAs deliver improved latency, energy efficiency, and flexibility
- Could leverage the massive LUTs resource in FPGA, DSP block won't be the bottleneck

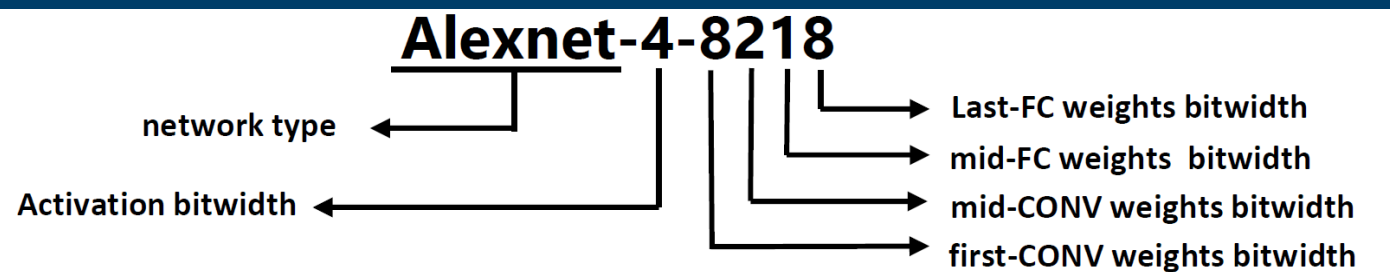
Theoretical peak performance for low precision (evaluated on Xilinx XC7Z045, 200MHz)



- **How to maintain the accuracy with ELB quantization or tradeoff between accuracy and throughput?**
- **How to develop a FPGA design and accelerate ELB-NNs with high efficiency?**



## Experimental results on Alexnet benchmark with various precision/bit-width



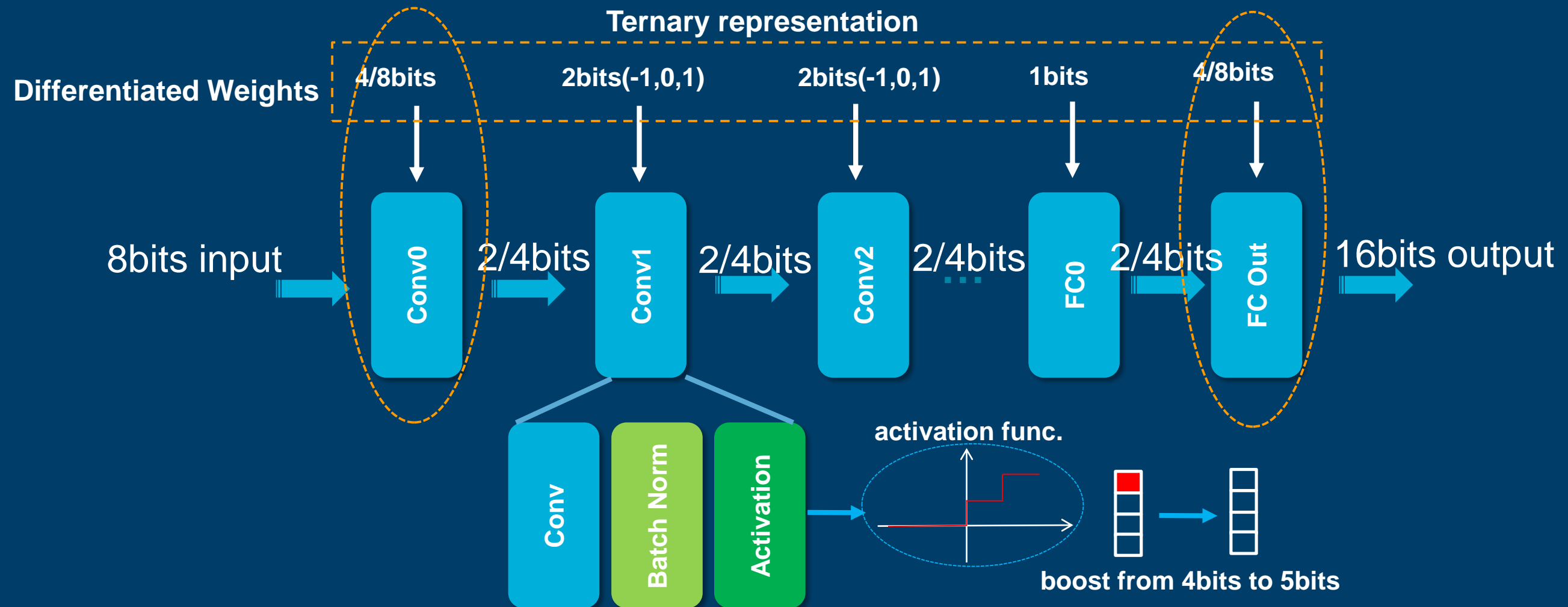
**We need hybrid ELB-NN solutions**

Network precision	Accuracy (Top-1)
Alexnet with float32	55.9% [6]
Alexnet-8-8888	54.6%
Alexnet-8-8228	53.3%
Alexnet-8-8218	52.6%
Alexnet-8-8118	51.1%
Alexnet-4-8218	49.3%
Alexnet-2-8218	46.1%
Alexnet-4-8218 (w/o group)	53.2%
Alexnet-4-8218 (extended)	54.5%

- Increasing the network complexity/scale to significantly brings back accuracy
- Activation precision has more significant impact to the final classification accuracy
- Promising accuracy with binary/ternary weights in mid-layers

# Example of Hybrid ELB-NN

- Use relatively higher precision in first or last layer, since it is more sensitive to accuracy but less convolutional operation. Also its number of weights is relatively small, not won't eat more one-chip memory and I/O bandwidth.
- Utilize the sign bit after the ReLU to improve the activation precision without affording any extra bit.
- Ternary representation provides tradeoff of memory bandwidth, logic resource, and accuracy.



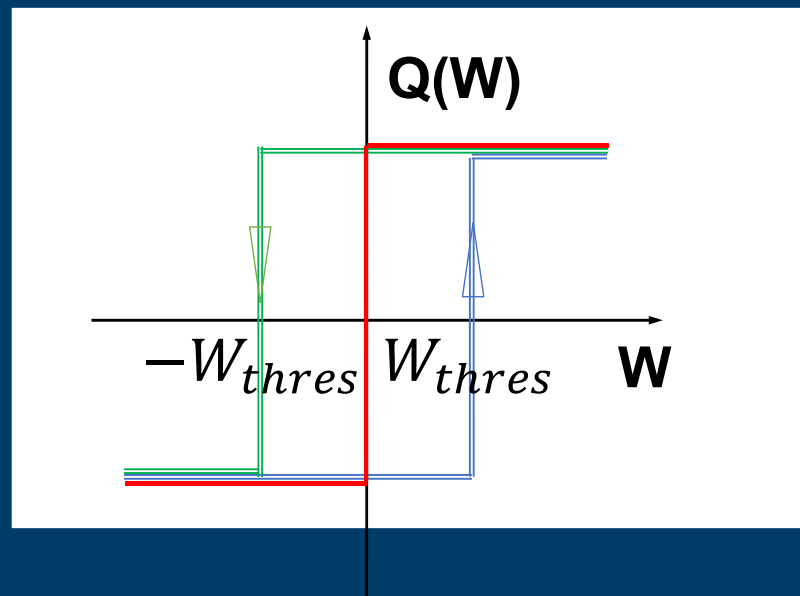
# How to Train a Hybrid ELB-NN

## Extended the caffe-Ristretto to support hybrid ELB-NN

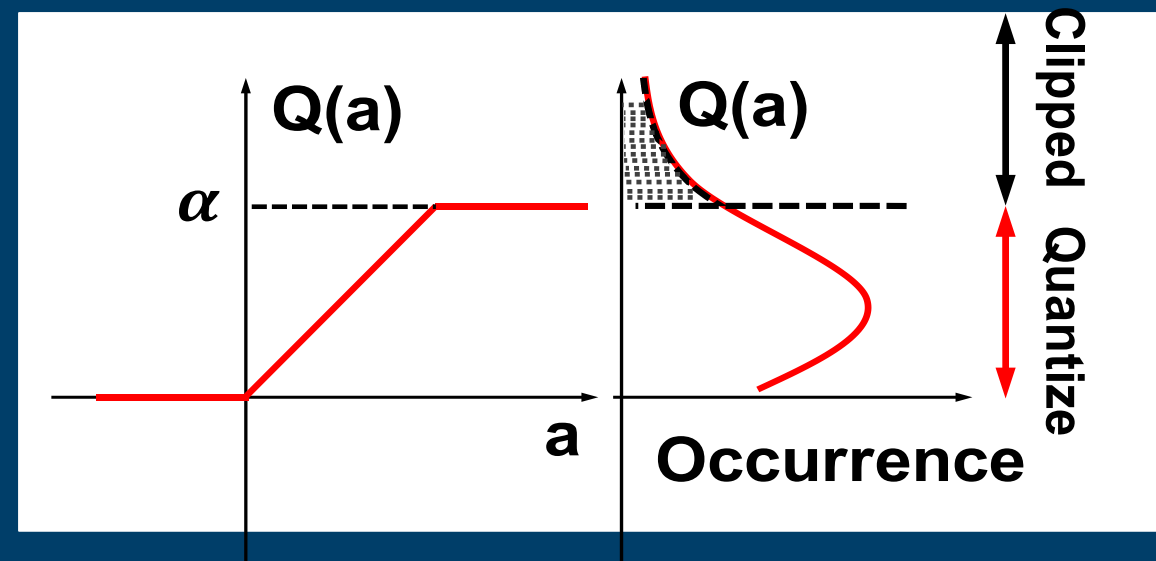
- Binary/ternary weights with scale factor
- Activation (clip & quantization)

## Enhance the training stability and accuracy

### Hysteresis



### PACT[1]



[1] PACT: Parameterized Clipping Activation for Quantized Neural Networks, <https://arxiv.org/abs/1805.06085>

TABLE II: AccELB performance evaluated on Xilinx ZC706 board

Network	Utilization (include SDK, around 11% LUT and BRAM)				Batch size	Bandwidth (GBytes/s)	Complexity (GOP)	Speed (imges/s)	Performance (TOPS)
	LUT	FF	BRAM	DSP					
Alexnet-8-8888 (baseline)	86262(39%)	51387(12%)	303(56%)	808(90%)	2	10.8	1.45	340	0.493
Alexnet-8-8218	103505(47%)	90125(21%)	498(91%)	550(61%)	5	3.35	1.45	856.1	1.24
Alexnet-4-8218	105673(48%)	94149(22%)	463(85%)	880(98%)	8	3.35	1.45	1369.6	1.99
Alexnet-4-8218 (w/o group)	127393(58%)	105328(24%)	435(80%)	839(93%)	7	4.30	2.61	1198.5	2.59
Alexnet-4-8218 (extended)	124317(57%)	101558(23%)	481(88%)	783(87%)	7	3.4	4.22	599.2	2.53

- Extended Alexnet- 8218 can reach up to **599.2** images/s, which surpasses the baseline (Alexnet-8-8888) by **1.76x** while still keeping the same accuracy.
- Significant reduction of 68% bandwidth resulting less power consumption. Half of DDR hardware cost can be saved,

Network precision	Accuracy (Top-1)
Alexnet with float32	55.9%[6]
Alexnet-8-8888	54.6%
Alexnet-8-8228	53.3%
Alexnet-8-8218	52.6%
Alexnet-8-8118	51.1%
Alexnet-4-8218	49.3%
Alexnet-2-8218	46.1%
Alexnet-4-8218 (w/o group)	53.2%
Alexnet-4-8218 (extended)	54.5%

TABLE III: Comparison with existing FPGA-based low precision DNN accelerators

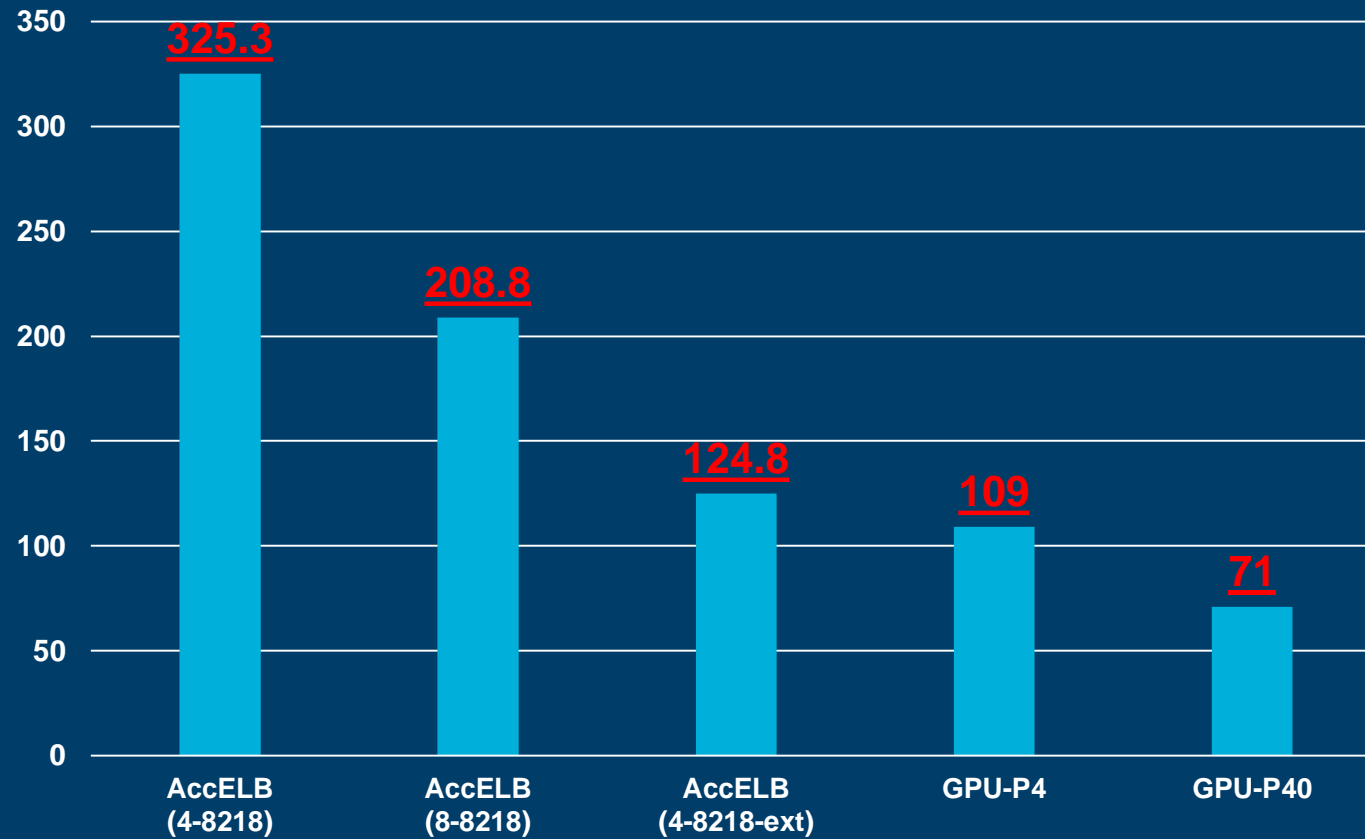
Reference	[15]	[16]	[19]	[5]	AccELB(1)-VGG16	AccELB(2)-VGG16
FPGA chip	XC7Z020	XC7Z045	XCKU115	Arria10 1150	XC7Z045	XC7Z045
Frequency	143MHz	200 MHz	125 MHz	N/A	200MHz	200 MHz
Network Type	Binary	Binary	Binary	Binary	Hybrid(4-8218)	Hybrid(2-8118)
kLUTs (used/total)	46.9/53.2	82.9/218.6	392.9/663	N/A	113.0/218.6	138.0/218.6
Performance (TOPS)	0.21	9.1	14.8	9.8	3.43	10.3
efficiency (GOPS/kLUT)	3.95	41.6	22.3	N/A	15.6	47.1

- LUT efficiency(GOPS per kilo LUTs), outperforms the most efficient work in [16] by 13%, even if we use one more bits for the activation and the code is automatically generated.
  - The major reason is that the generated code is RTL level, and the PE is well optimized and also written in RTL.
- 
- [15] R. Zhao et al. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In FPGA, 2017.
  - [16] Y. Umuroglu et al. Finn: A framework for fast, scalable binarized neural network inference. In FPGA. ACM, 2017.
  - [19] Fraser Nicholas et al. Scaling binarized neural networks on reconfigurable logic. arXiv:1701.03400, 2017.
  - [5] N. Eriko et al. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. FPT, 2016

# Comparison to GPU solution



Power Efficiency(images/s/watt)



Network precision	Accuracy (Top-1)
Alexnet with float32	55.9%[6]
Alexnet-8-8888	54.6%
Alexnet-8-8228	53.3%
Alexnet-8-8218	52.6%
Alexnet-8-8118	51.1%
Alexnet-4-8218	49.3%
Alexnet-2-8218	46.1%
Alexnet-4-8218 (w/o group)	53.2%
Alexnet-4-8218 (extended)	54.5%

- A clear tradeoff between accuracy and efficiency is be observed.
- Extended Alexnet-4-8218 (the same accuracy as INT8) still outperforms the most efficient GPU (P4) solution so far by 14%.

---

# Backup

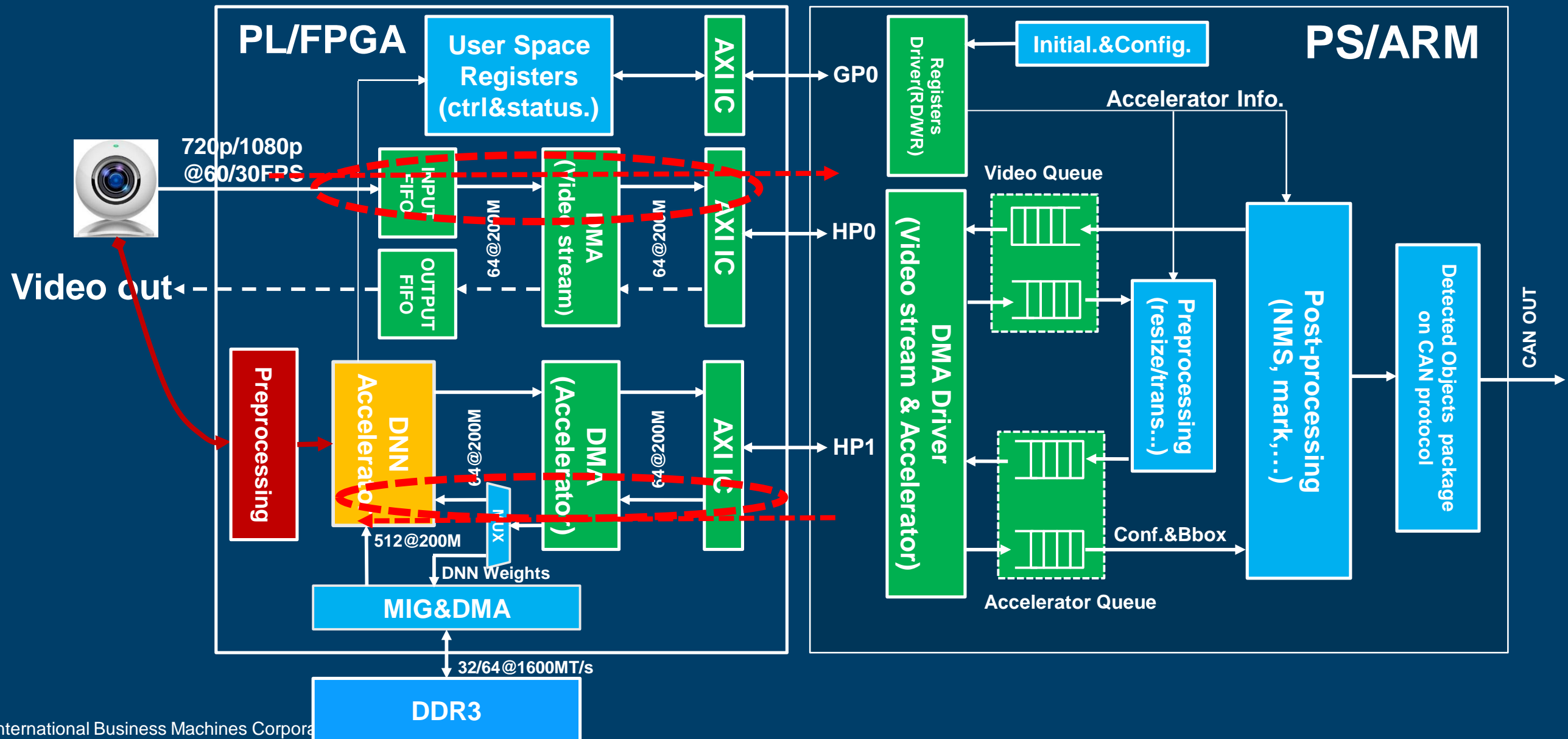


- Very limited I/O bandwidth due to the power budget, cost, size, (usually 16/32bits DDR3)
- High resolution input, e.g. 720P/1080P, results in larger feature map size (can't down-sampling in some cases).
- Faster response, millisecond level latency
- No batching can be explored, any batching operation will introduce more latency.
- Task is usually simple, like detecting face, traffic sign,...
- In many cases, the objects are very small, such as the view from the drone.

# A Typical Video Embedded System for Edge Computing



- Can video frame be feed into accelerator directly to alleviate the bandwidth pressure of memory?
- Can we utilize the video transmission time of video frame to reduce the acceleration latency?



# Spatial Complementary paired sparse kernel

## Paired sparse kernel

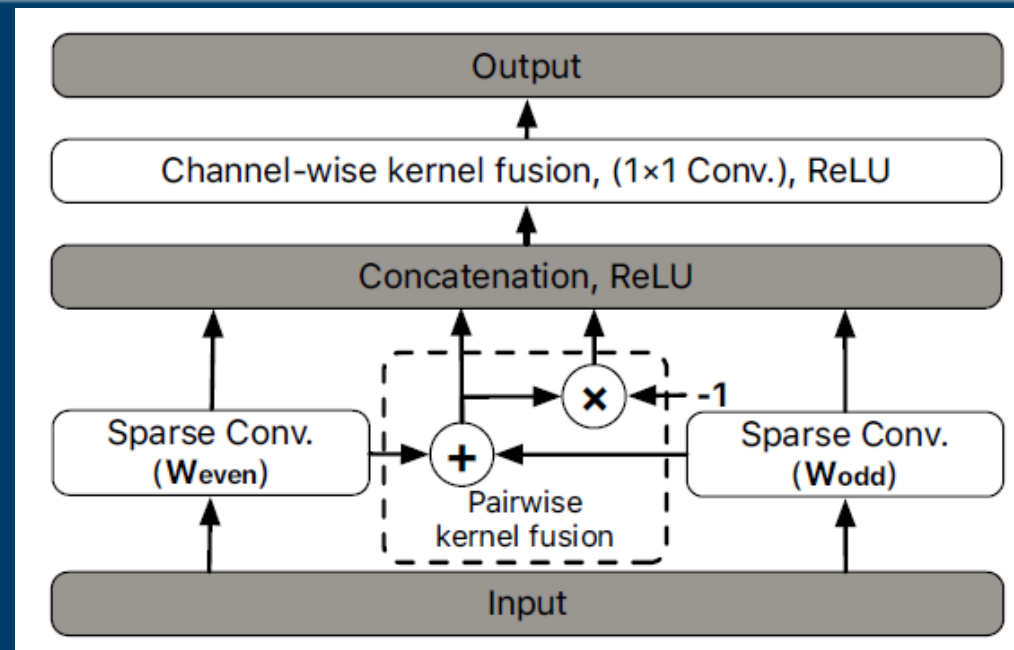
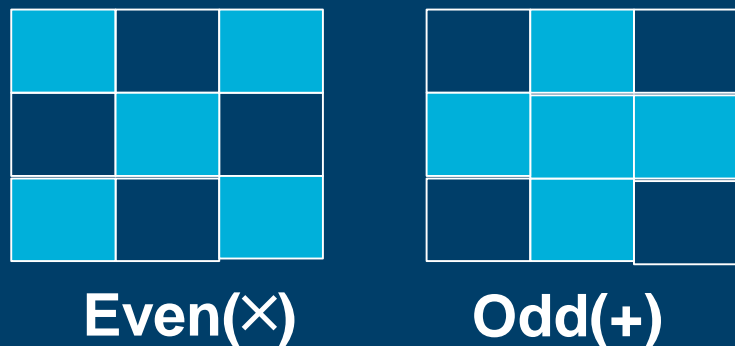
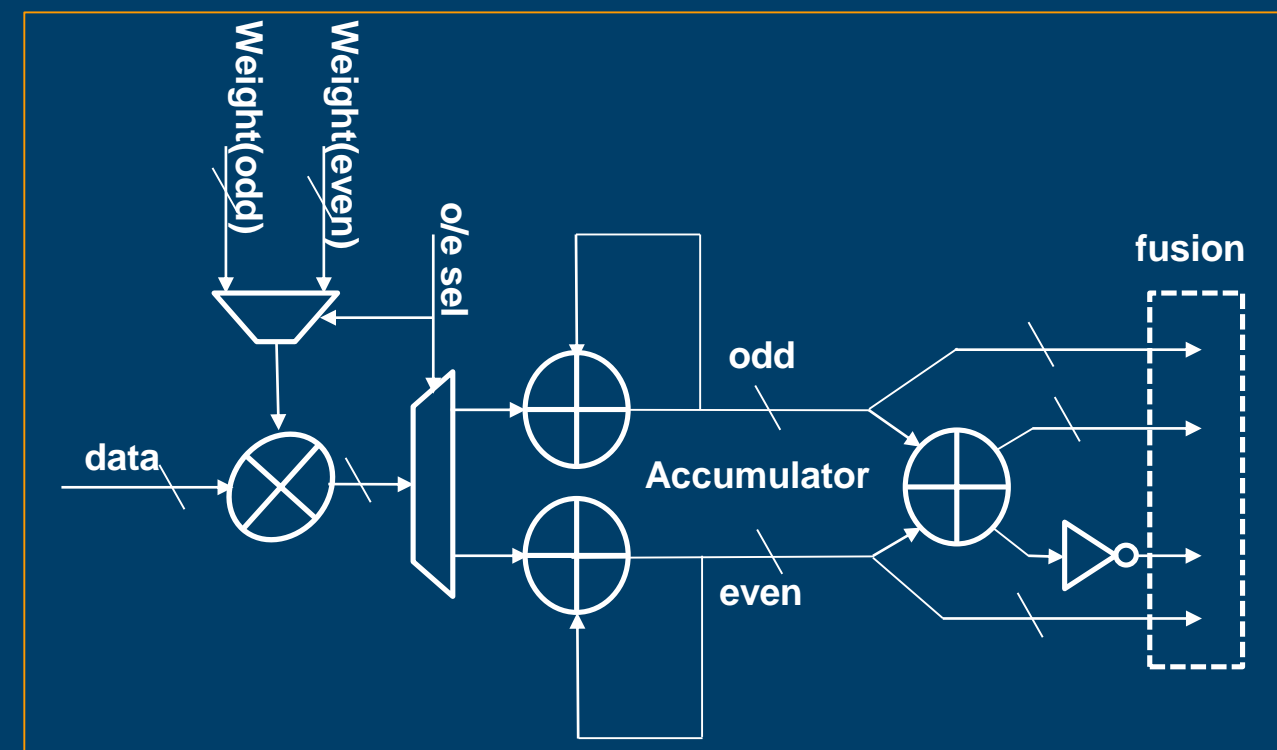
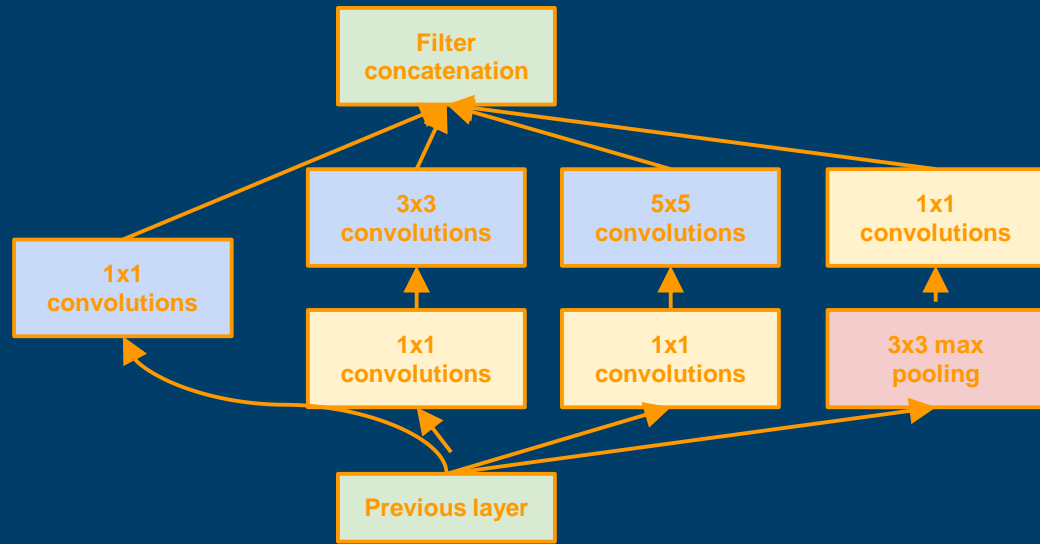


Table 5. Classification accuracy on ImageNet of our networks.

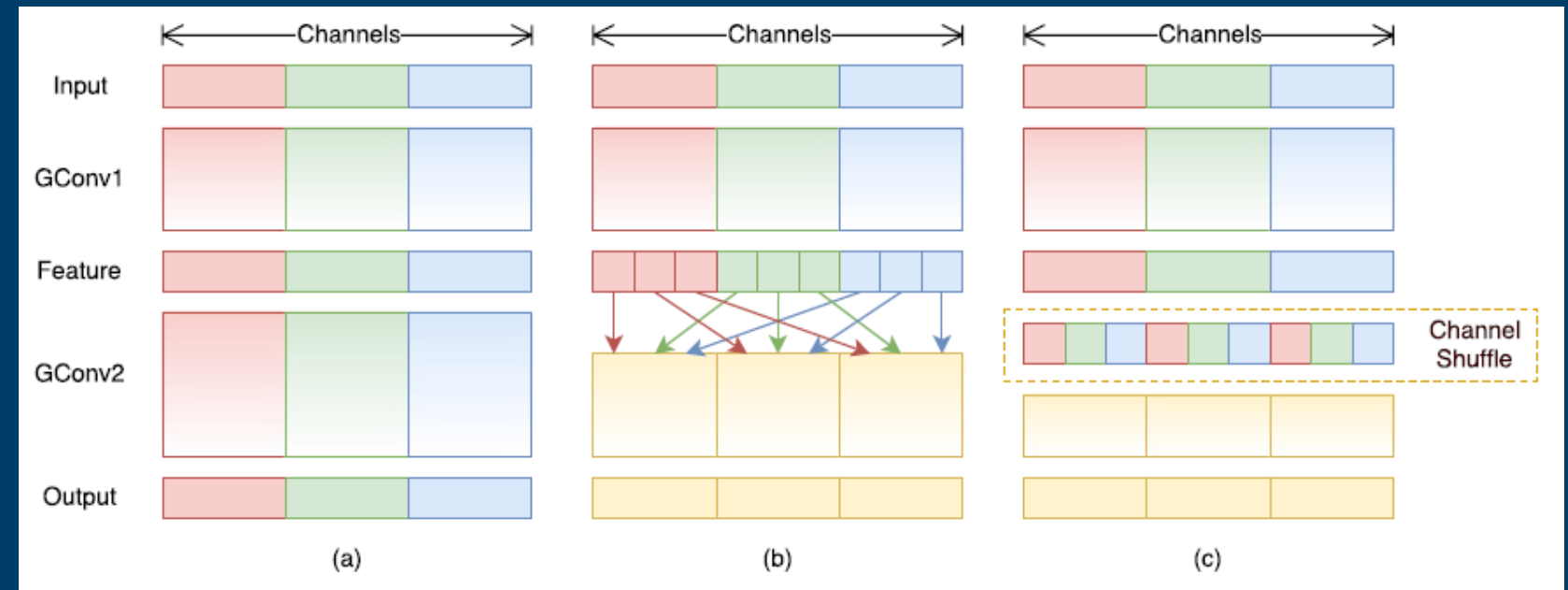
Network	Top-1 Acc.	Top-5 Acc.	FLOPs ↓	Params ↓ (Conv)	Params ↓ (Total)
AlexNet [17]	57.72%	80.46%	—	—	—
<i>scFusion-4</i>	58.64%	81.56%	1.88×	2.43×	1.02×
<i>scFusion-8</i>	54.35%	78.22%	2.63×	3.24×	1.03×
VGG-16 [31]	70.94%	89.92%	—	—	—
<i>scFusion-4</i>	71.52%	90.25%	2.40×	1.60×	1.04×
<i>scFusion-8</i>	69.90%	89.21%	4.64×	3.21×	1.08×
ResNet-50 [5]	75.21%	92.24%	—	—	—
<i>scFusion-4</i>	74.27%	91.96%	1.23×	1.21×	1.21×
<i>scFusion-8</i>	73.53%	91.52%	1.50×	1.44×	1.44×
<i>scFusion-8 (group)<sup>‡</sup></i>	72.90%	91.10%	2.38×	2.17×	2.17×

<sup>‡</sup>: Group of two  $1 \times 1$  convolutional layers in the residual block are set to 2.





## Inception



## ShuffleNet

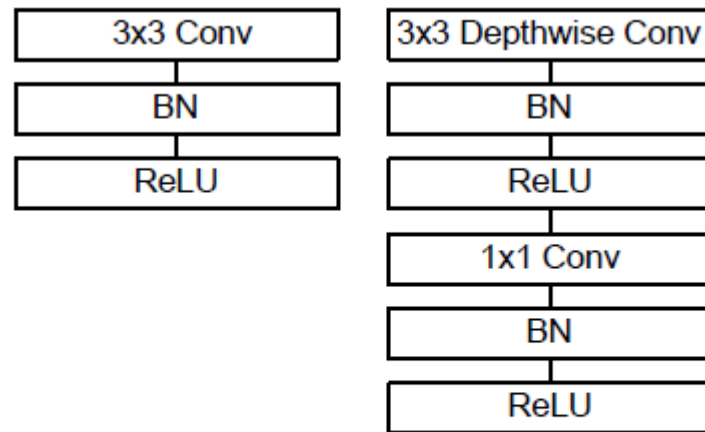
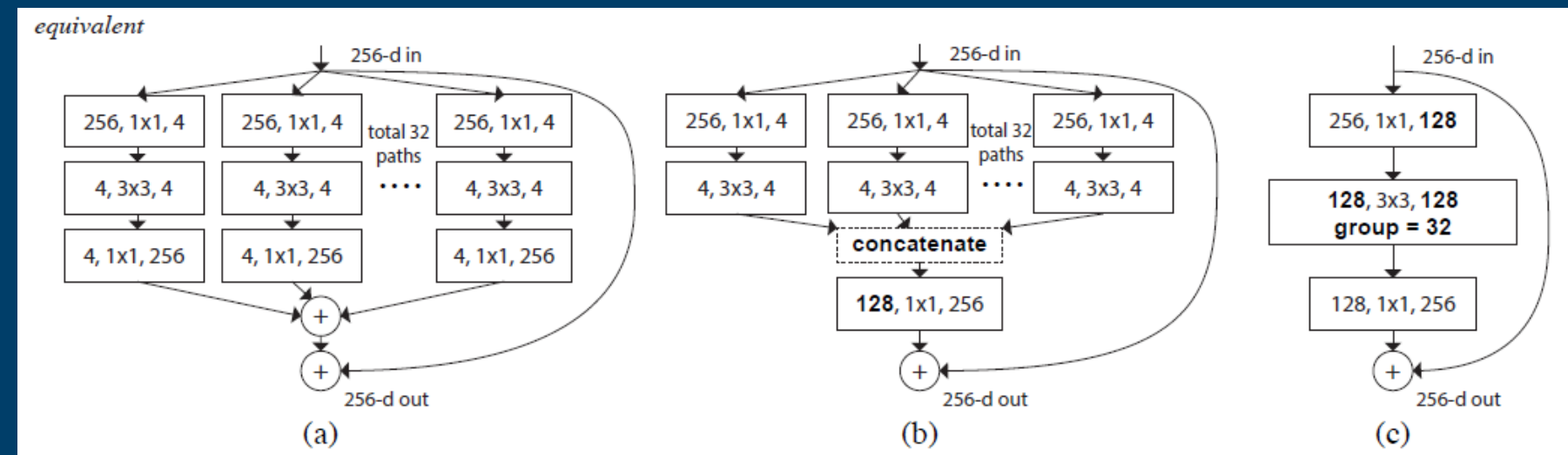


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

## MobileNet



## ResNeXt

**Adaptable.**  
**Intelligent.**

