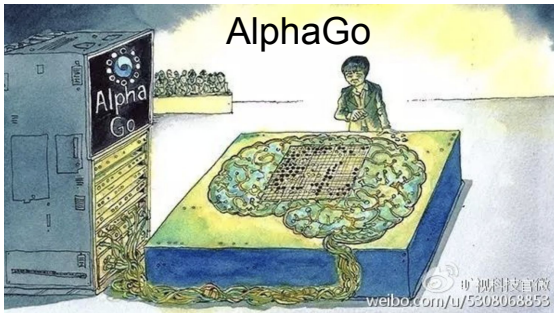
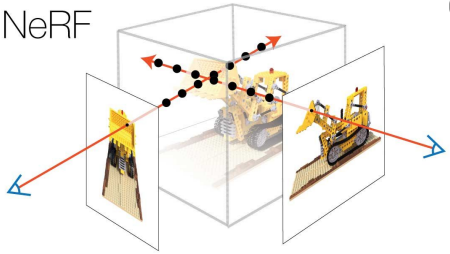
Hardware-software codesign for Computer Vision

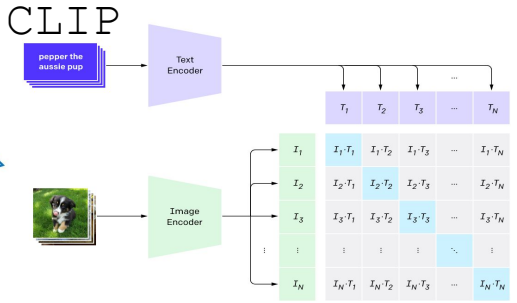
Object Detection and AI-ISP with Edge AI

zsc@megvii.com

2022.4

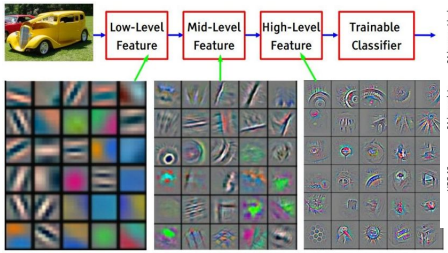




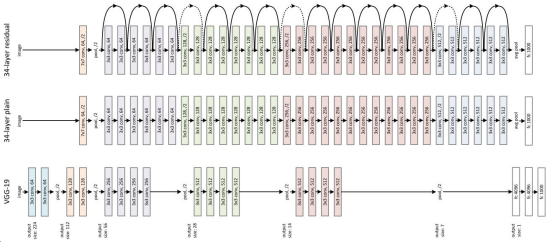


ImageNet

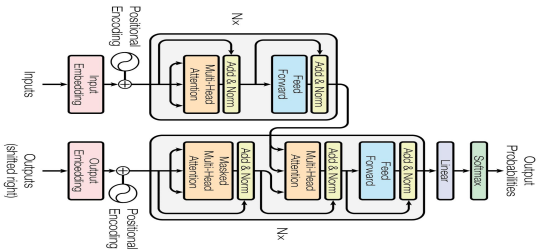
Convolutional NN



ResNet



Transformer

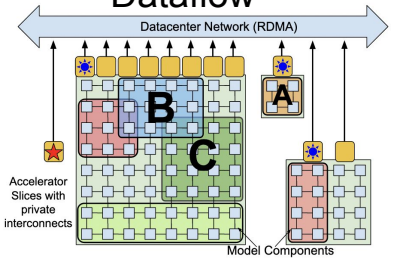


GPU

Accelerators

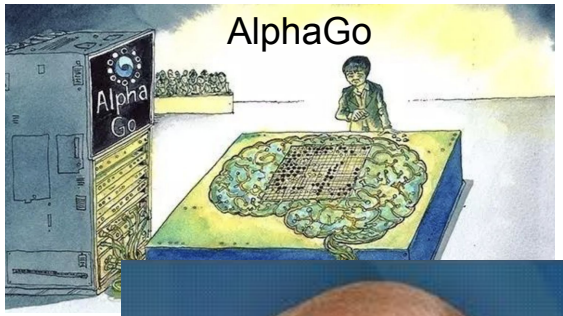
Quantization

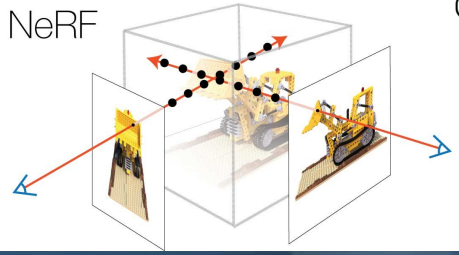
Dataflow

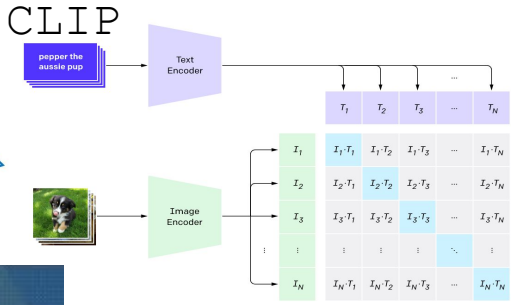


2011 *"BigBang"*

"Roaring 20s"

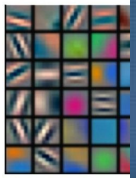




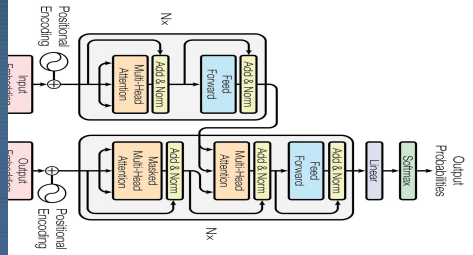


ImageNet

Con

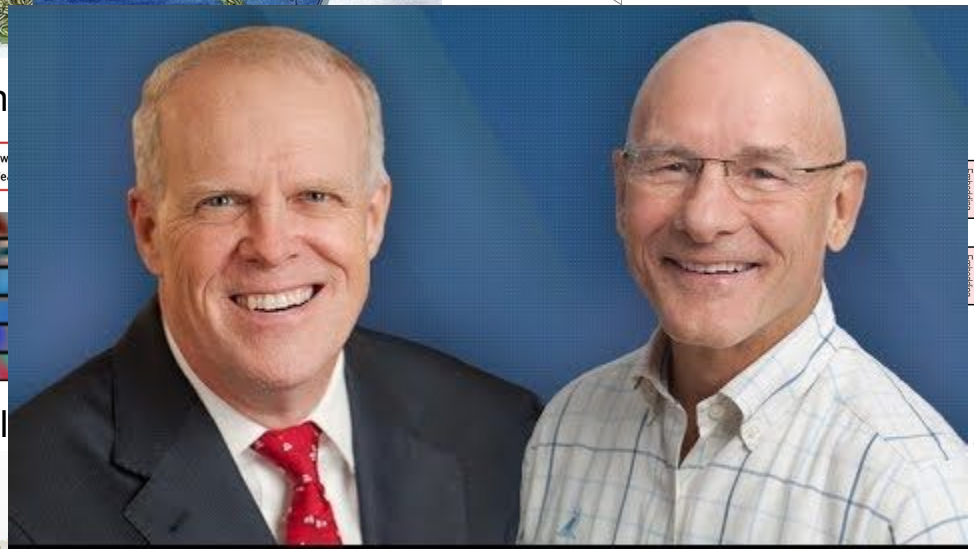


Transformer



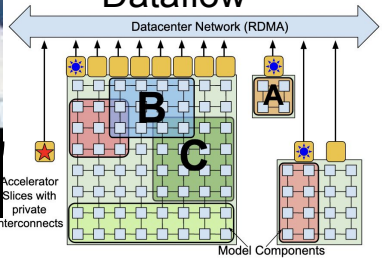
GPU

Accel



New *Computer Architecture Golden Age*

Dataflow



W

αW^B

2011 *"BigBang"*

2018 Turing Award

"Roaring 20s"

Outline

- Codesign for Real-time 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

About me

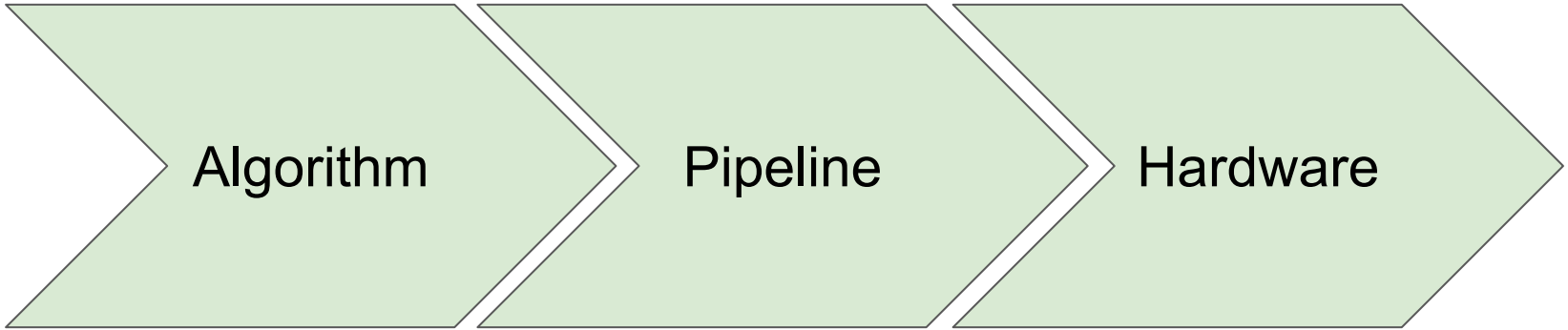
https://zsc.github.io/

Compiler Optimization	Machine Learning	Neural Network & Architecture
• Source-to-source transformation • Cache simulation	• Natural Language Question & Answer • Indoor Navigation with INS • 3D reconstruction	• Quantized Neural Network • Edge AI Devices • Reinforcement Learning

2007	2008	2009	2010	2011	2012	2013	2014	2015 ~ now

Gap between Cloud and Edge

Algorithm-Pipeline-Hardware: the Codesign Triathlons



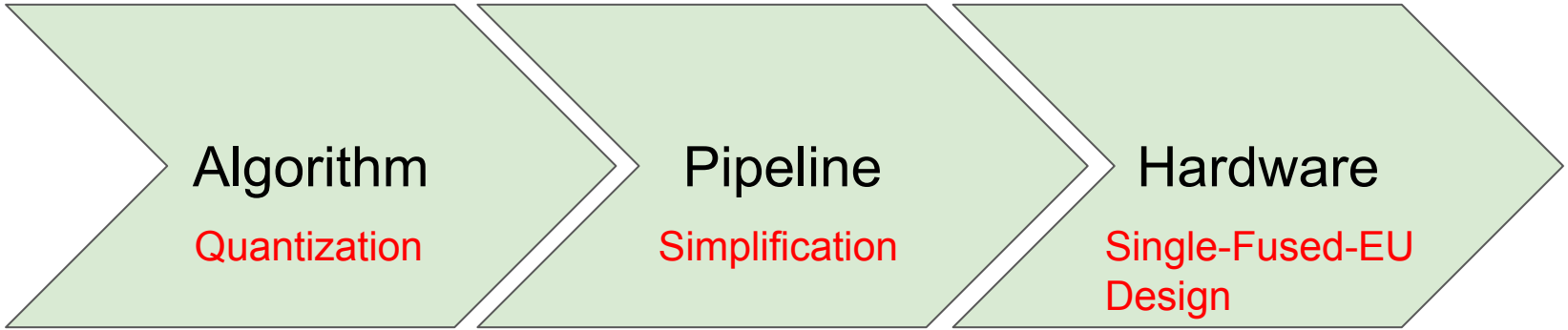
- Sets the quality upper bound
- Need triagable metrics
- Weekly update

- Memory/storage bandwidth & CPU
- Determinism
- Quarterly update

- Stable ISA for decoupled evolution
- Physical measurements
- Yearly update

Algorithm-Pipeline-Hardware: the Codesign Triathlons

For Codesign of Real-time 1080p Processing on a Low-end FPGA



- Sets the quality upper bound
- Need triagable metrics
- Weekly update

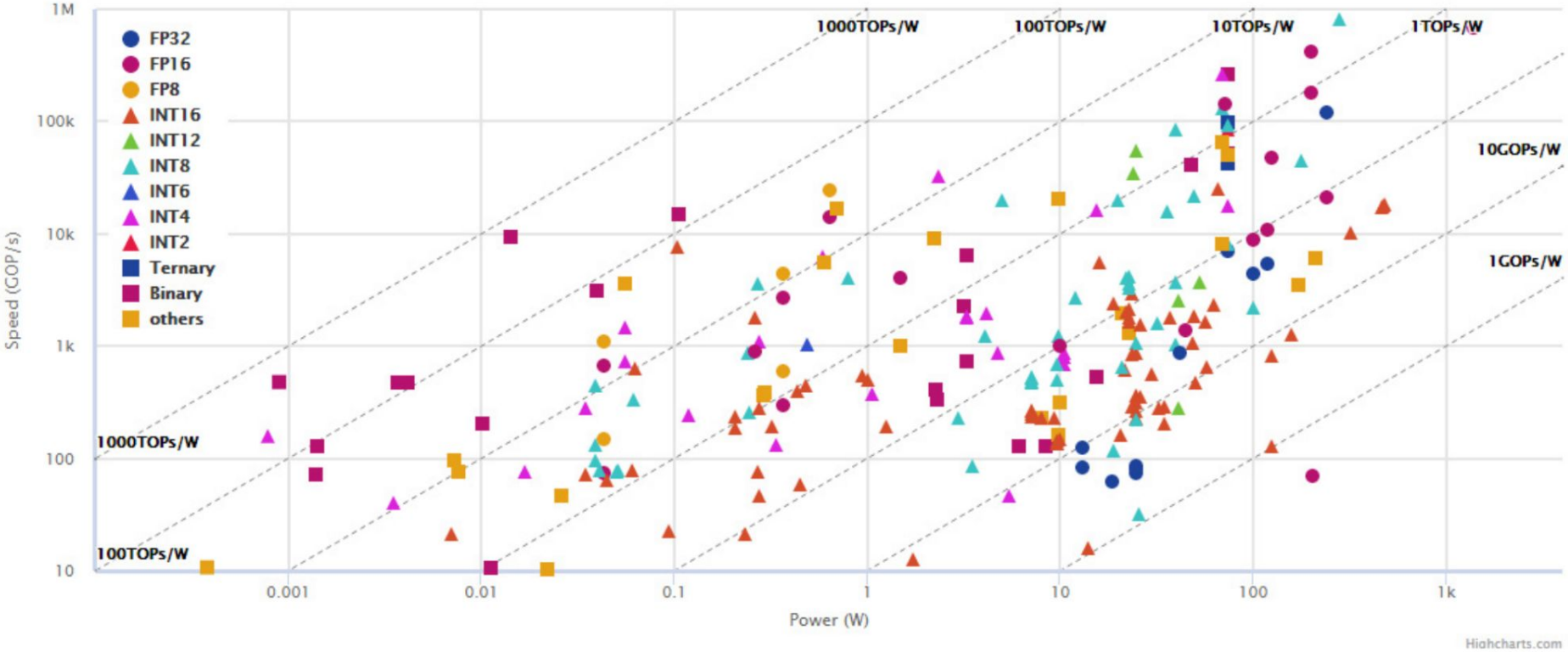
- Memory/storage bandwidth & CPU
- Determinism
- Quarterly update

- Stable ISA for decoupled evolution
- Physical measurements
- Yearly update

Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network 
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

High TOPs generally due to low-bit QNN

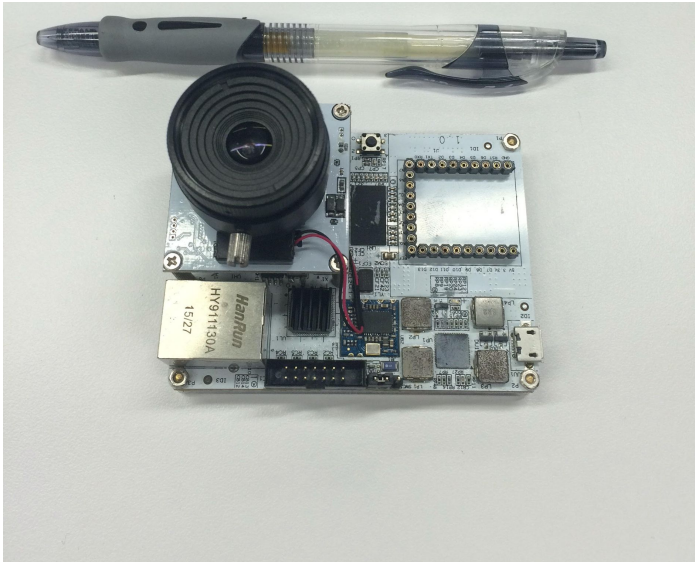


Source: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

Zynq 7020 FPGA

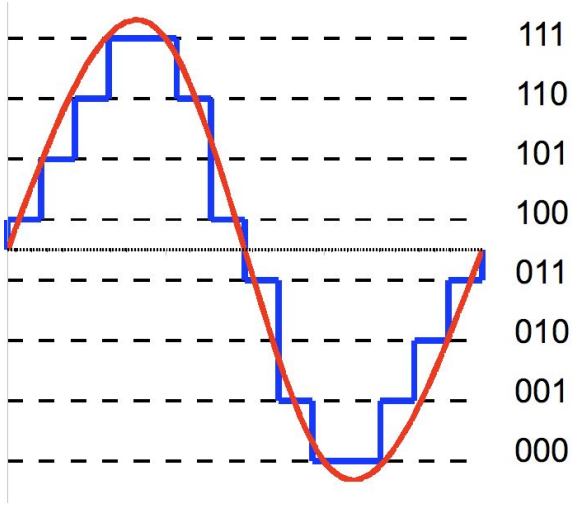
- costs 18 USD, ~20Gops float16 ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network (2020)
- FHD (1080p) is ~40 times the 224x224 resolution
- Need a 1000 FPS backbone for Real-time (25FPS) processing!

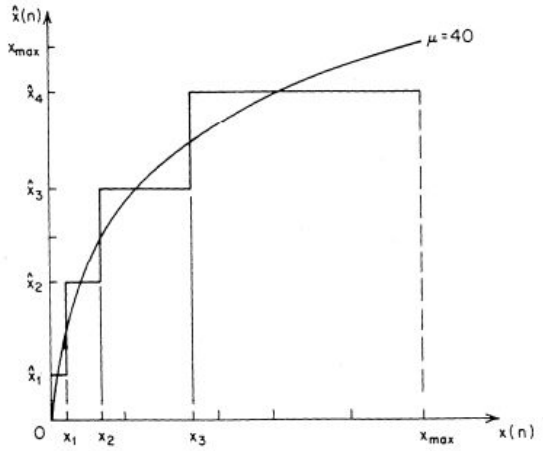
	Z-7020
Logic Cells (K)	85
Block RAM (Mb)	4.9
DSP Slices	220
Maximum I/O Pins	200
Maximum Transceiver Count	-



Fixed Points: Quantization

- Linear Quantization $$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$
  - Essentially rounding
  - Con: dynamic range of floating point numbers are problematic
- Logarithmic Quantization
  - Quantize log(x) instead of x
  - Achieve 12 bit quality with 8 bits
  - Con: addition not efficient on existing platforms





ig. 5.16 Distribution of quantization levels for a μ-law 3-bit quantizer ith $\mu = 40$.

Benefits of Quantized Neural Networks

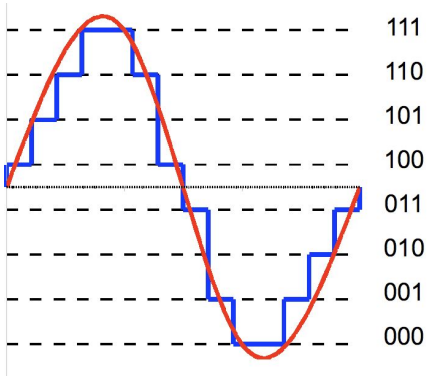
- Reduce weight size
- Reduce feature size
  - larger feature library size
- Speedup computation (usually require hardware support)
  - *O(Bit-width)* if SIMD, *O(Bit-width^2)* if special hardware
  - Saves bandwidth (intra-chip and off-chip), eases P & R
  - Dense computation
- Distributed training: reduces communication of weights

$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \, bitcount[and(c_m(\boldsymbol{x}), c_k(\boldsymbol{y}))]$$

	INT8	INT4	Binary
Tesla T4	130 T	260 T	
Megvii Zynq 7020	0.05 T	0.2 T	3.2 T

Differentiable Quantization

- Bengio ’13: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation
  - REINFORCE algorithm
  - Decompose binary stochastic neuron into stochastic and differentiable part
  - Injection of additive/multiplicative noise
  - Straight-through estimator



Gradient vanishes after quantization.

Quantization also at Train time

- Neural Network can adapt to the constraints imposed by quantization
- Exploits “Straight-through estimator” (Hinton, Coursera lecture, 2012)

$$x \approx \hat{x}$$

$$\Rightarrow$$

$$\frac{\partial}{\partial x} \approx \frac{\partial}{\partial \hat{x}}$$

- Example

$$\textbf{Forward: } q \sim Bernoulli(p) \qquad q \approx \mathbf{E}[q] = p$$

$$\textbf{Backward: } \frac{\partial c}{\partial p} = \frac{\partial c}{\partial q}.$$

Simple Implementation: zero_grad(q - p) + p

STE Alternative: Local Reparameterization Trick

Learning Discrete Weights using the local reparameter trick(Shayer, ICLR 2018)

$$\bigtriangledown L(W) = E_{W \in S}[\sum_{i=1}^{N} l(f(x_i, W), y_i)]$$

- local reparameterization trick (Kingma&Welling, 2014)

$$E_{p(x)}[f(x)] \qquad x = g(\epsilon, \theta)$$ usually Gaussian

$$\bigtriangledown_\theta E_{p(\epsilon)}[f(g(\epsilon, \theta))] \approx \sum_{i=1}^{m} \bigtriangledown_\theta f(g(\epsilon_i, \theta))$$ Monte Carlo approximation

- sampling weights -> sampling pre-activations

$$W_{ij,l} \sim N(\mu_{ij,l}, \sigma_{ij,l}) \qquad z_{i,l} \sim N(\sum_j \mu_{ij,l} h_{j,l-1}, \sum_j \sigma^2_{ij,l} h^2_{j,l-1})$$

Bit Neural Network

- Matthieu Courbariaux et al. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. http://arxiv.org/abs/1511.00363
- Itay Hubara et al. Binarized Neural Networks https://arxiv.org/abs/1602.02505v3
- Matthieu Courbariaux et al. Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1. http://arxiv.org/pdf/1602.02830v3.pdf
- Rastegari et al. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks http://arxiv.org/pdf/1603.05279v1.pdf
- Zhou et al. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients https://arxiv.org/abs/1606.06160
- Hubara et al. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations https://arxiv.org/abs/1609.07061

Binarizing AlexNet

Theoretical

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%53.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Scaled binarization

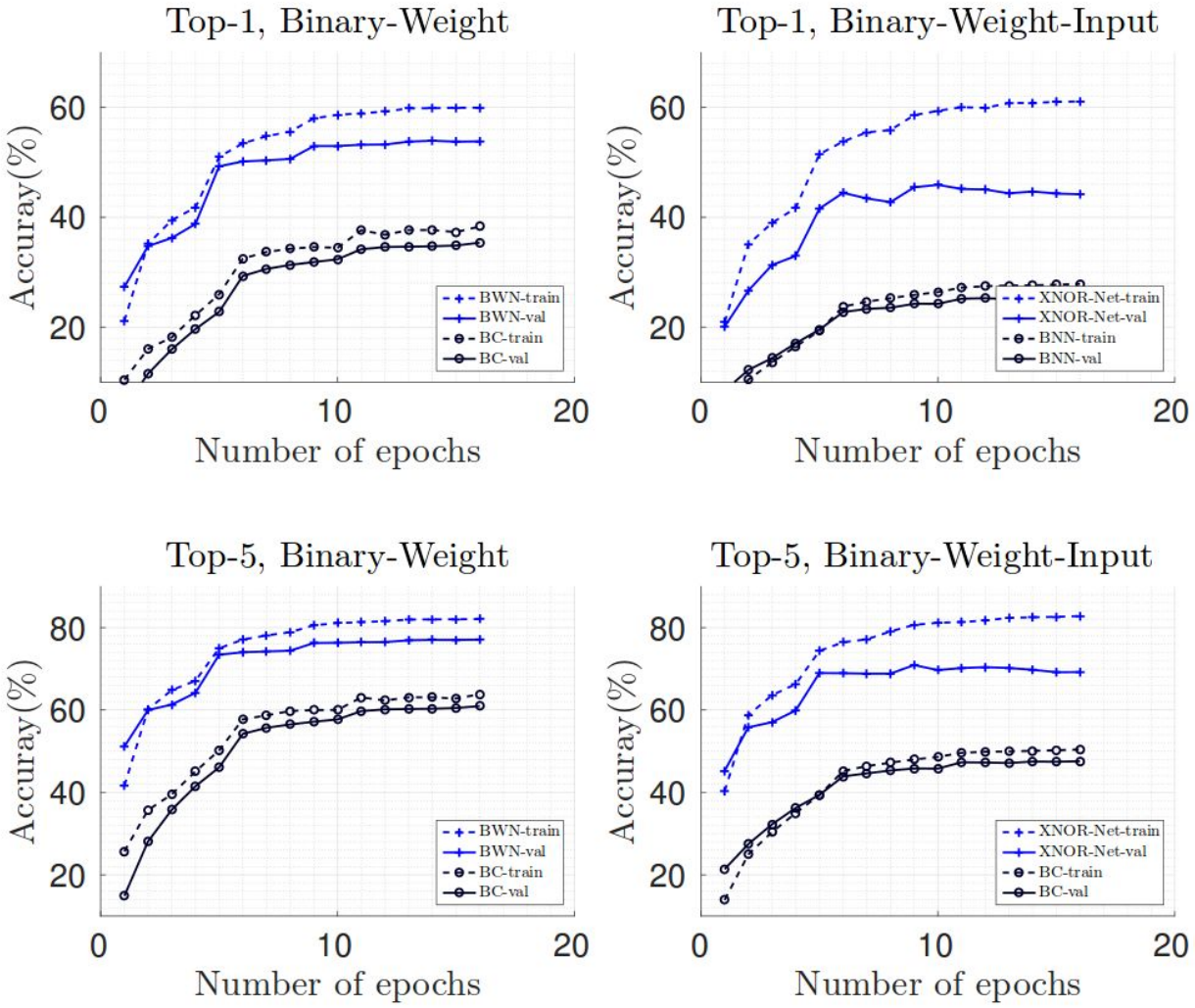
$$\min_{\Lambda \in \text{diagonal}, B \in \{1,-1\}^{m \times n}} \|\Lambda B - W\|_F^2$$

- Sol:

$$\|\Lambda B - W\|_F^2 = \|(\Lambda B) \circ B - W \circ B\|_F^2$$
$$= \|\Lambda (B \circ B) - W \circ B\|_F^2$$
$$= \|\Lambda \mathbf{1} - W \circ B\|_F^2$$

= Varaince of rows of W o B

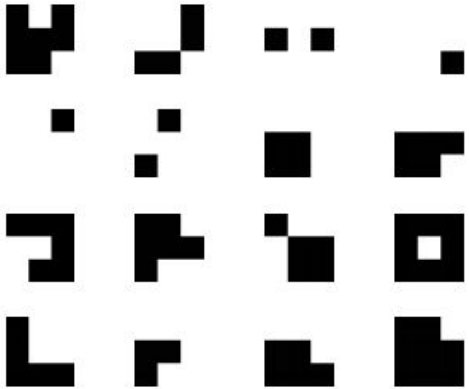
XNOR-Net


Top-1, Binary-Weight
Accuray(%)
Number of epochs
BWN-train
BWN-val
BC-train
BC-val
Top-1, Binary-Weight-Input
XNOR-Net-train
XNOR-Net-val
BNN-train
BNN-val
Top-5, Binary-Weight
Top-5, Binary-Weight-Input

Binary weights network

- Filter repetition
  - 3x3 binary kernel has only 256 patterns modulo sign.
  - 3x1 binary kernel only has only 4 patterns modulo sign.
  - Not easily exploitable as we are applying CHW as filter

Figure 2: Binary weight filters, sampled from of the first convolution layer. Since we have only 2^{k^2} unique 2D filters (where k is the filter size), filter replication is very common. For instance, on our CIFAR-10 ConvNet, only 42% of the filters are unique.



Binarizing AlexNet

Theoretical

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
▪ Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%53.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Scaled binarization is no longer exact and not found to be useful

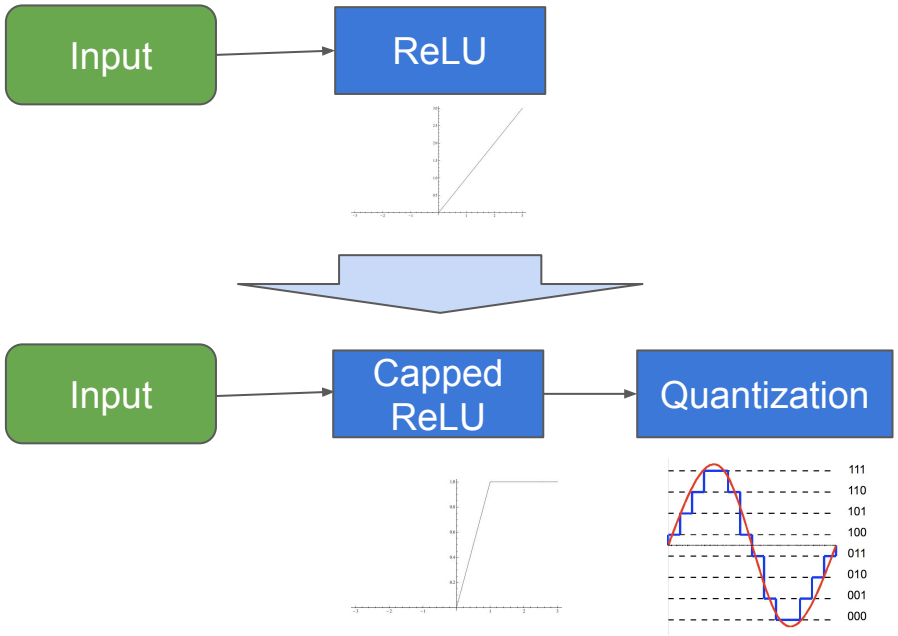
$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha, \mathbf{B}, \beta, \mathbf{H}}{\operatorname{argmin}} \|\mathbf{X}^\mathsf{T}\mathbf{W} - \beta\alpha\mathbf{H}^\mathsf{T}\mathbf{B}\|$$

The solution below is quite bad, like when Y = [-4, 1]

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}^\mathsf{T}) \operatorname{sign}(\mathbf{W}) = \mathbf{H}^{*\mathsf{T}}\mathbf{B}^*$$

Quantization of Activations

- XNOR-net adopted STE method in their open-source our code

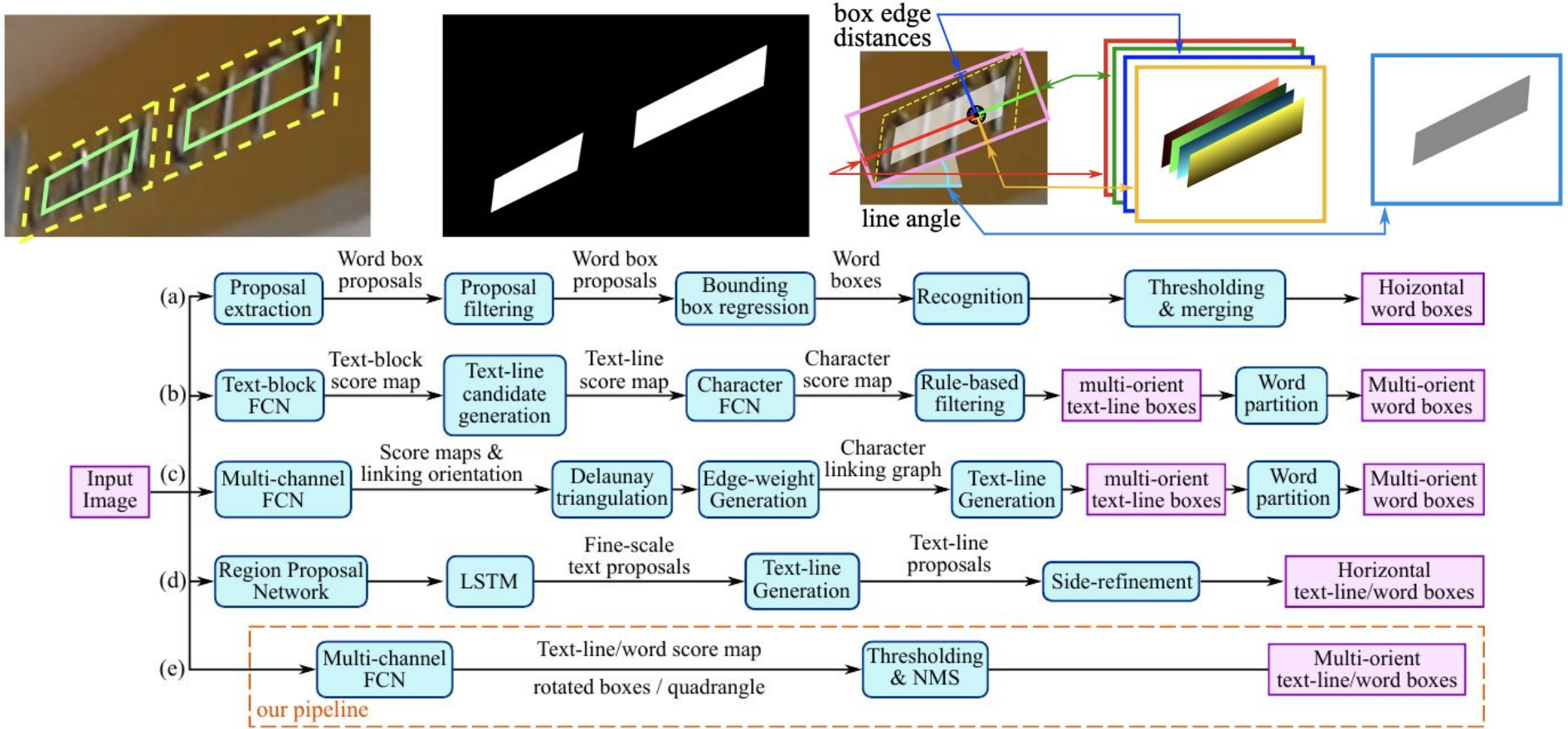


Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

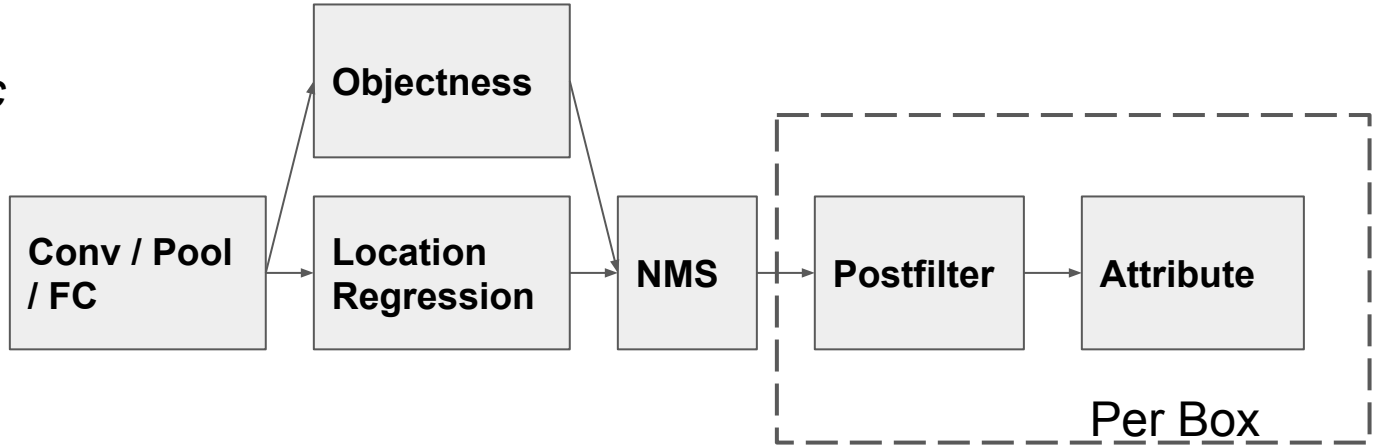
Computer Vision Pipelines: OCR

EAST: An Efficient and Accurate Scene Text Detector, CVPR '17

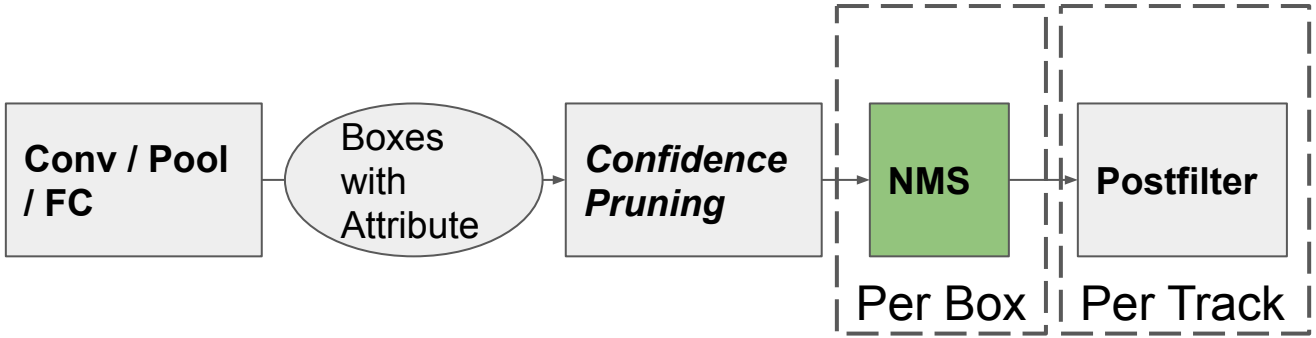


Pipeline Simplification

Before Algorithmic Simplification



After Algorithmic Simplification



Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator 
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

"MNISC": a Minimalist Architecture for Accelerator

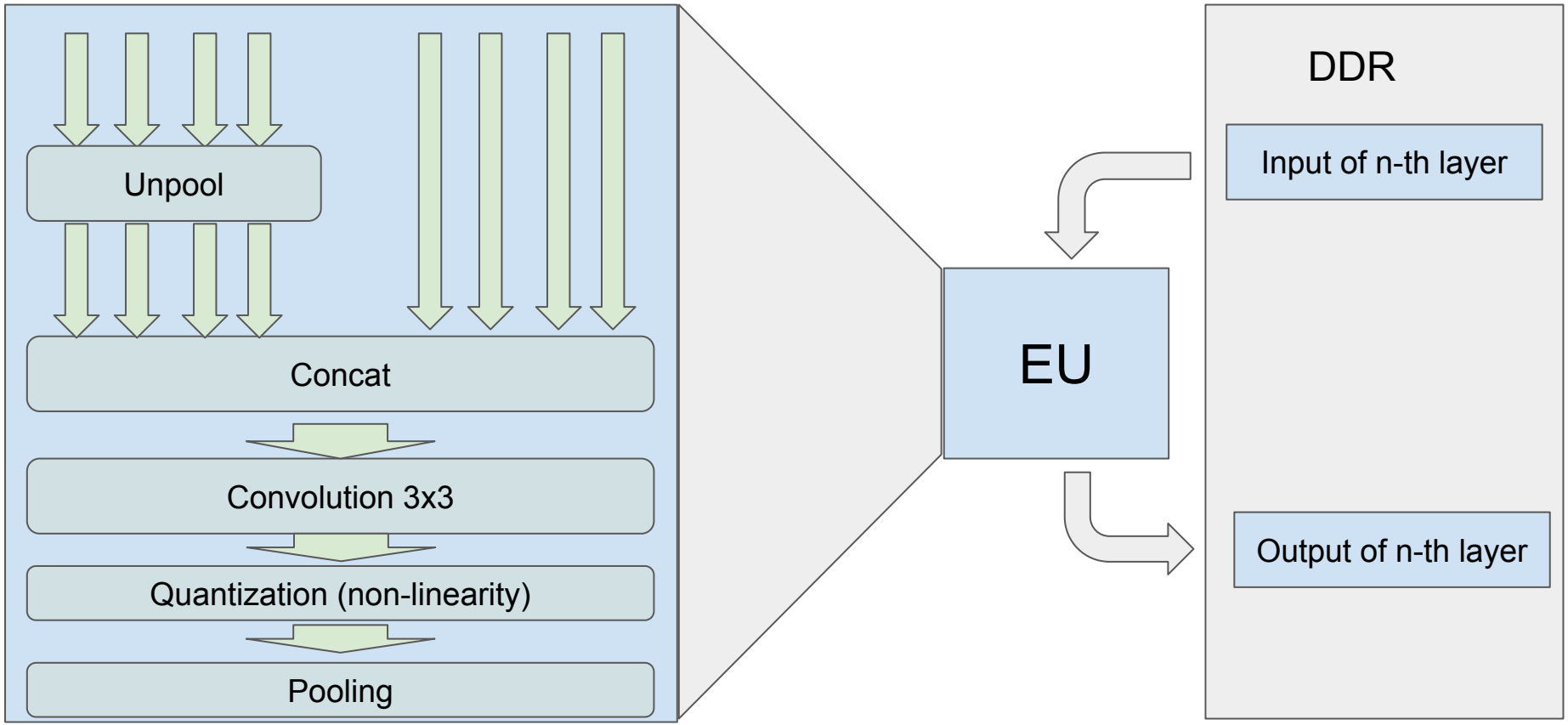
- Minimal design, low resource requirement
  - Simple EU design
  - Read/write DDR for each "layer"
  - Deployable to Zynq 7020
- Native support int2/int4

Three ways of Upsampling

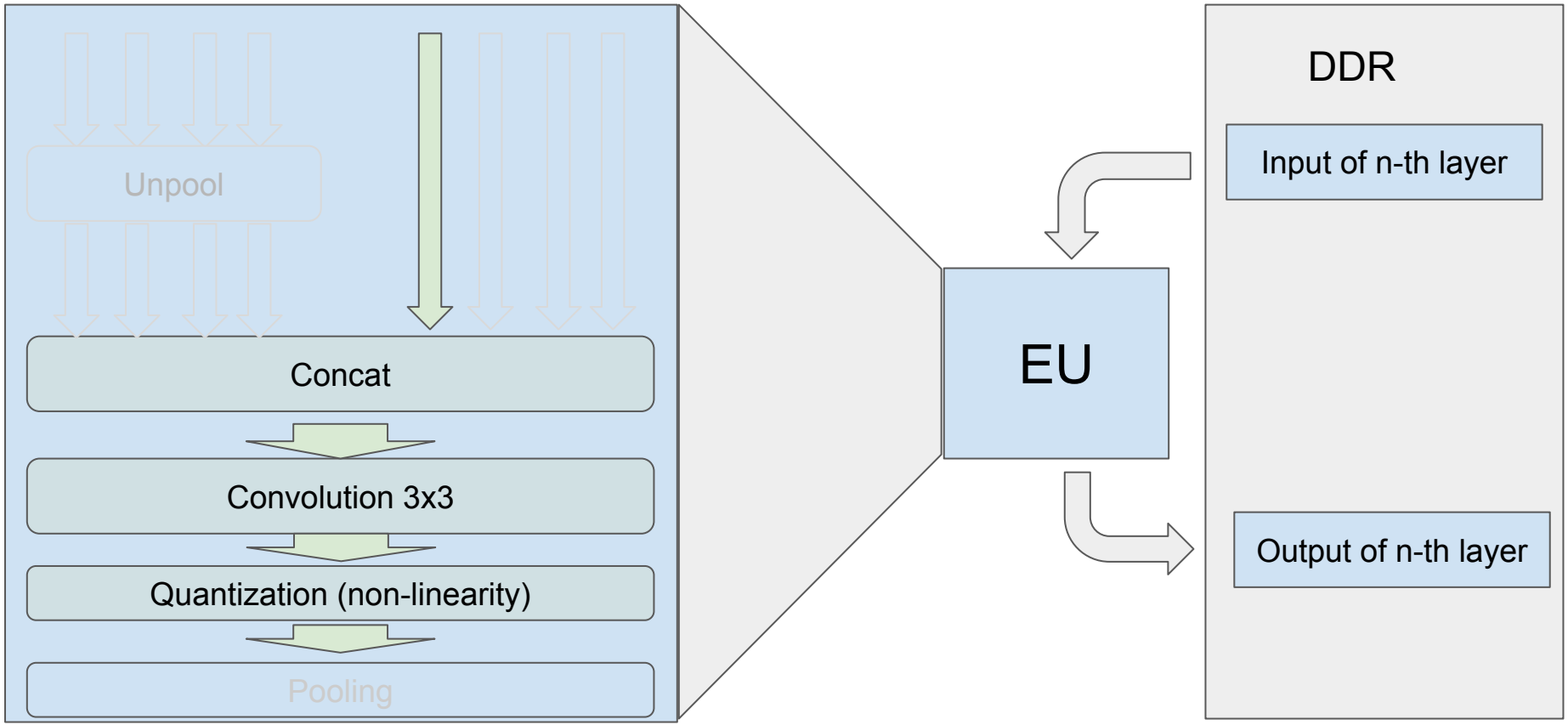
- Transposed Convolution (fractional stride)
  - Motivated by implementation readiness: use a "Backward" operator as forward for free
- Pad zero + convolution
  - Easy to implement
- Reshape operation ("subpixel")
  - The more "efficient" software way

- Pro & Cons
  - checkerboard effects: we noticed in 2015, the effects will go away with more training/larger layer/extra layer, etc.
  - In several tasks, we found the three ways above are quite the same
  - Decide to only support the 2nd upsampling method

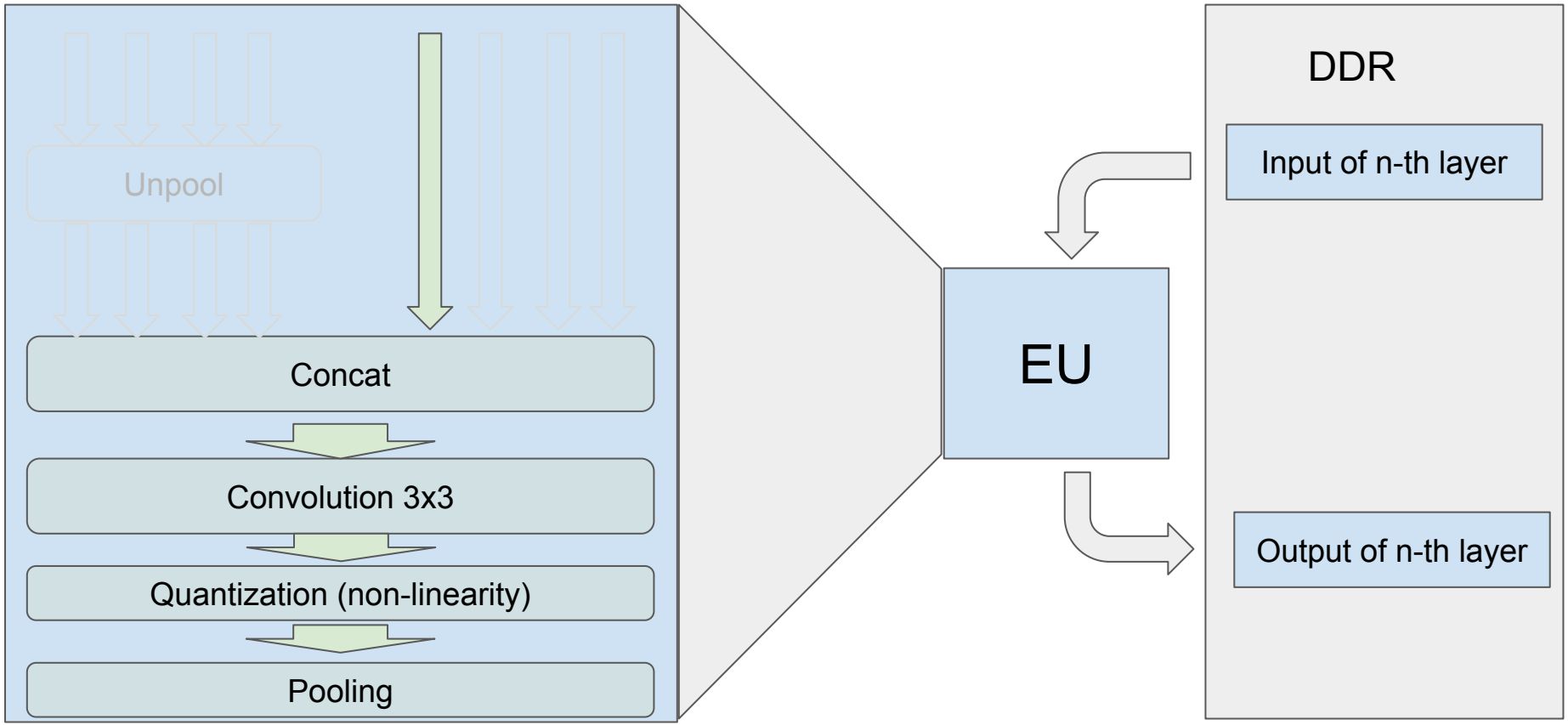
MNISC: one EU is all you need for a U-Net



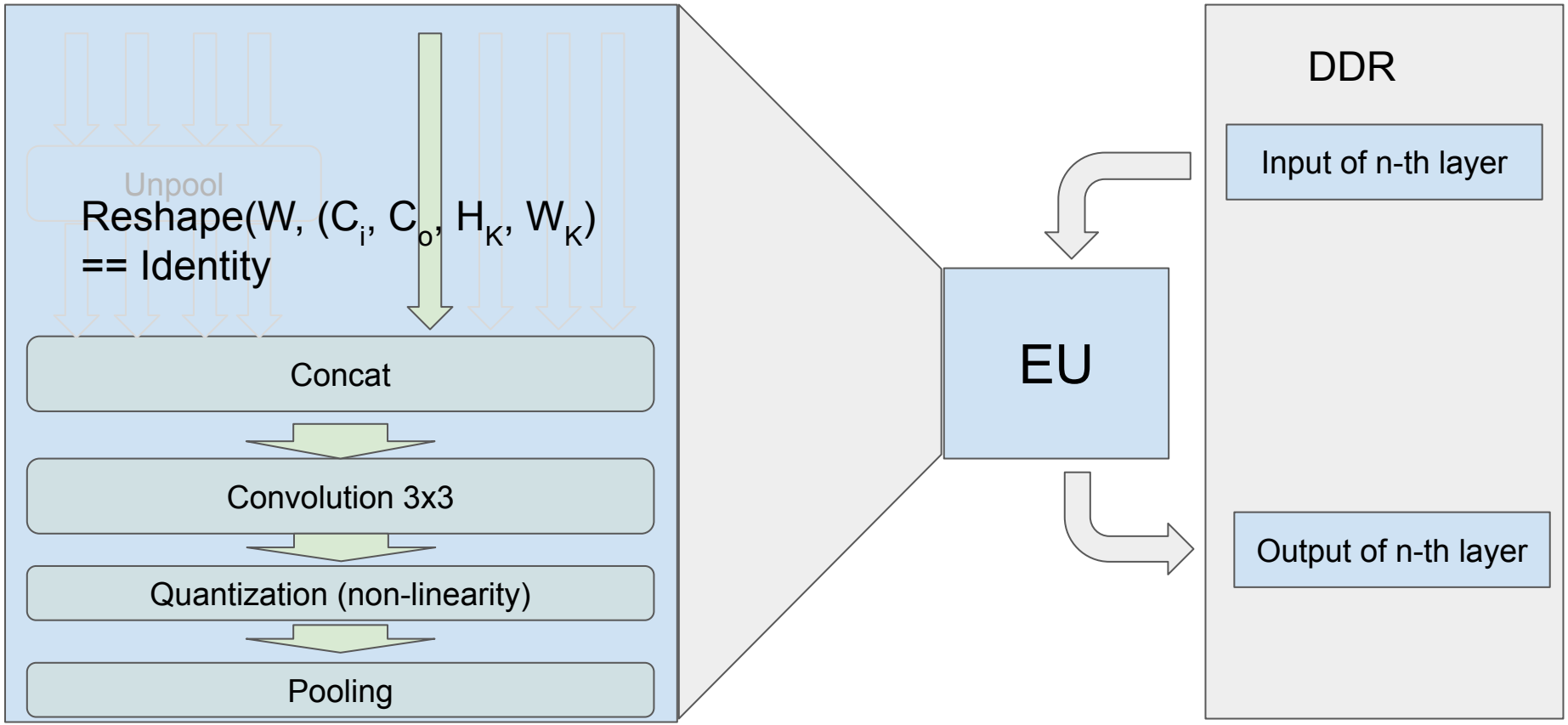
MNISC: vanilla Convolution



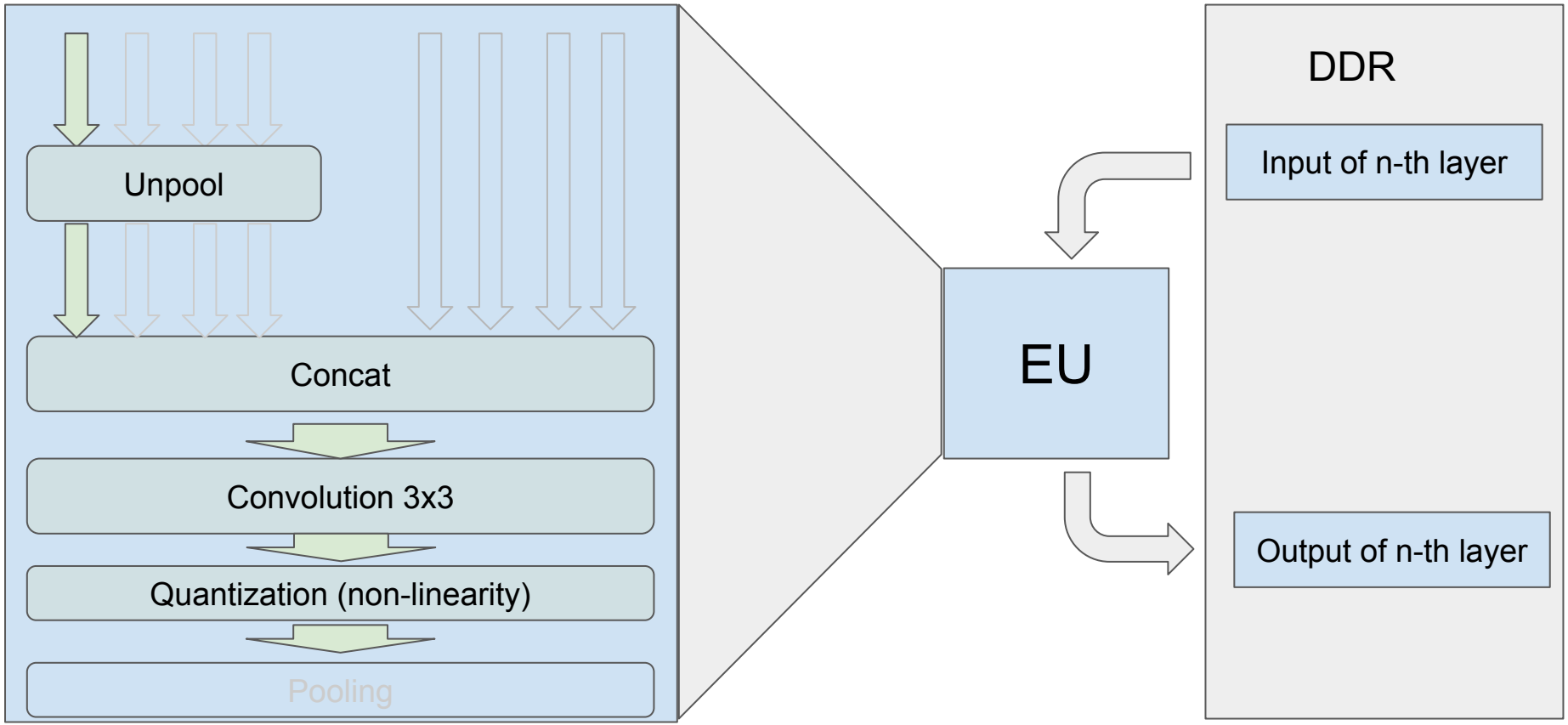
MNISC: Convolution + Pooling



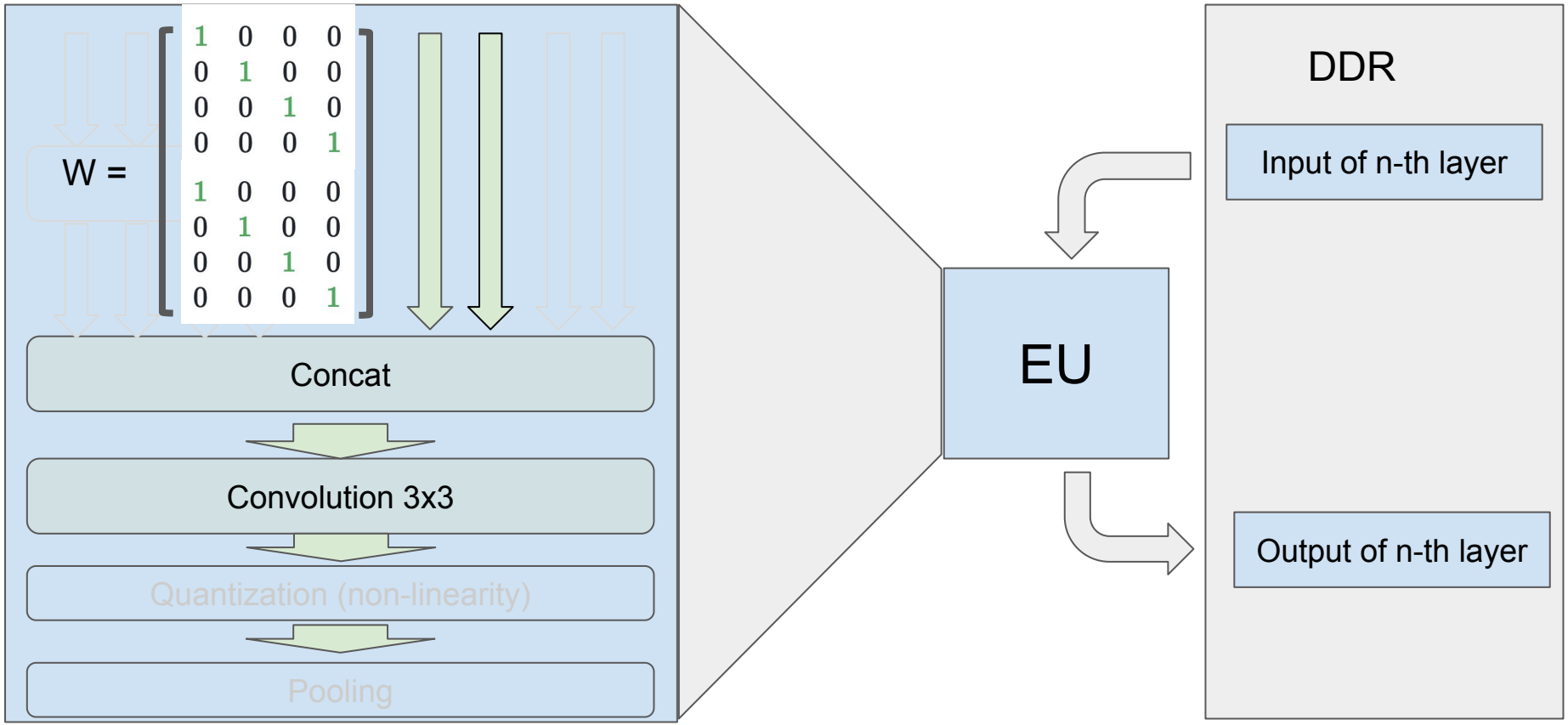
MNISC: only Pooling (setting Conv weights)



MNISC: Deconvolution

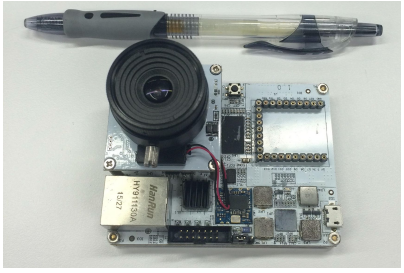


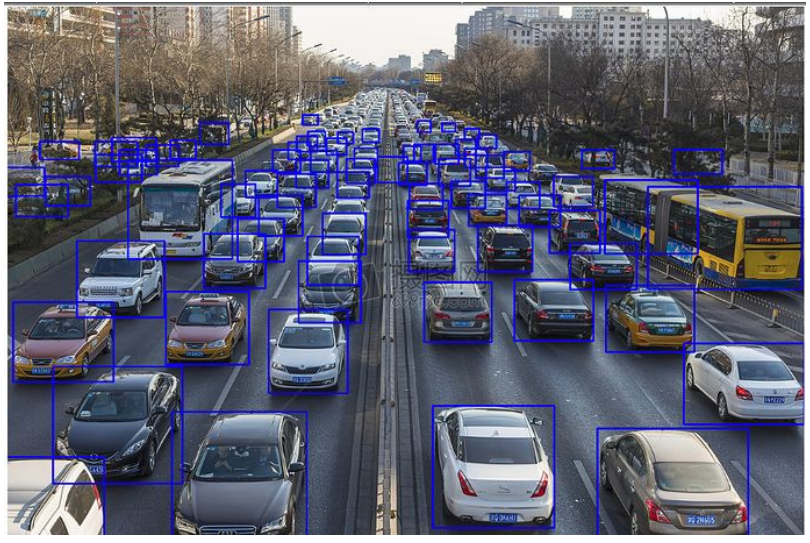
MNISC: Residual Add (setting Conv weights)



Realtime Processing on a low-end FPGA

Zynq 7020





Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization 
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

Software-hardware Codesign : Portfolio of Number formats

- Int8, int16 (signed/unsigned)
- Int1, Int2, int4
- Deterministic
- Low in resource
- Breaks Compatibility
  - int8 has too small dynamic range
  - int1/int2/int4 has problem with negation: either asymmetric and no "safe negation", or symmetric but no "zero"

- float8, float16, bfloat16, float32
- Complex
  - overflow/underflow, denormal numbers
  - Rounding rules (Rounding to nearest / directed rounding)
- Breaks Associativity
  - Kahan summation algorithm
  - Pairwise summation

Floating Point Formats

bfloat16

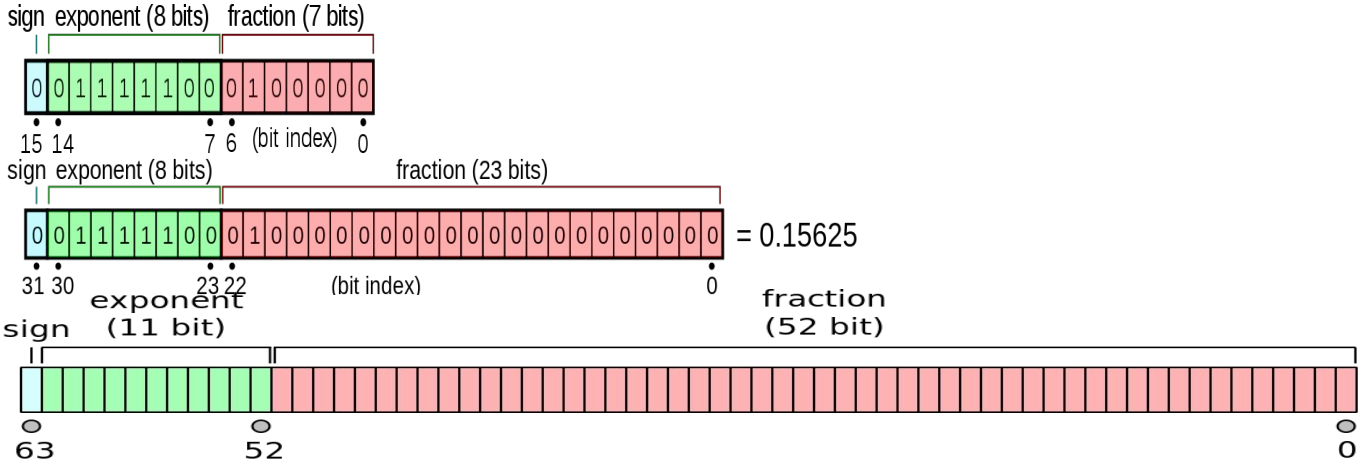
float32

float64

float8 ?

CSD

Posit



$$x = \pm \sum_{r=1}^{R} a(r) 2^{q(r)} \text{ , 其中 } a(r) = 0, 1 \quad \text{(unsigned power-of-2)}$$

$$\text{或 } x = \sum_{r=1}^{R} a(r) 2^{q(r)} \text{ , 其中 } a(r) = -1, 0, 1 \quad \text{(signed power-of-2)}$$

QAT vs. PTQ

	Bitwidth	Dataset	Impacts to training
QAT	<8W8A	Full training dataset	Need modify model.py
PTQ	8W8A (wo/ distillation)	Small (1~1000) calibration set	Just train float model

Best of both worlds: PTQ+Fine-tuning

	Bitwidth	Dataset	Workflow and impacts to model-trainer
QAT	**<8W8A**	Full training dataset	One stage: model-trainer need modify model.py
PTQ	8W8A (wo/ distillation)	**Small (1~1000) calibration set**	**Two stage: model-trainer just trains float model**
PTQ +Finetuning	**<8W8A**	**Small (1~1000) calibration set + Synthetic Data**	**Two stage: model-trainer just trains float model**

Features and Weights need different quantization

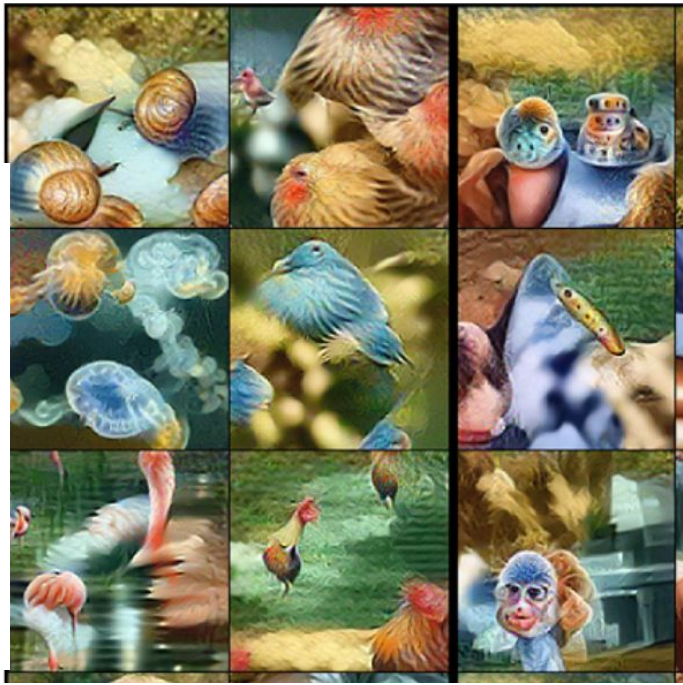
	Saturation	Zero-point	Distribution
Feature Quantization	Can, may be good	Can be non-zero. Easily compensated by pre-computing bias	Need compute over real/synthetic calibration dataset
Weight Quantization	Cannot	Need be zero, otherwise involve summation over feature	Just compute statistics

Selected PTQ works

- 2019 DFQ: Data-Free Quantization through Weight Equalization and Bias Correction
- 2019 The Knowledge Within: Methods for Data-Free Model Compression
  - Synthetic Data
- 2020: ZeroQ: A Novel Zero Shot Quantization Framework
- 2020 AdaRound: Up or Down? Adaptive Rounding for Post-Training Quantization (Qualcomm)
- 2020 AdaQuant: Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming
- 2020 BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction
- 2021 Zero-shot Adversarial Quantization

The Knowledge Within: Methods for Data-Free Model Compression 2019

	Settings	*Reference*	BNS	$\mathcal{I}$	$BNS + \mathcal{I}$	$\mathcal{G}$
ResNet-18 [8] - ImageNet, fp32 accuracy - 69.75						
Calibration	8w8a, 10^1	69.63 (0.03)	69.55 (0.01)	69.6 (0.02)	69.57 (0.04)	68.94 (0.02)
	4w4a, 10^2	54.72 (0.06)	55.29 (0.1)	38.25 (0.1)	55.49 (0.06)	53.02 (0.1)
KD	4w4a, 10^2	68.63	67.98	62.8	68.06	63.98
	4w4a, 100^2	68.68	68.14	63.1	67.95	63.58
MobileNet-V2 [28] - ImageNet, fp32 accuracy - 71.88						
Calibration	8w8a, 10^1	71.26 (0.05)	71.34 (0.03)	71.2 (0.01)	71.32 (0.04)	71.17 (0.02)
	4w4a, 10^3	15.1 (0.1)	16.17 (0.1)	10.55 (0.04)	16.1 (0.06)	13.36 (0.04)
KD	4w4a, 10^3	68.5	66.4	53.13	66.07	36.95
DenseNet-121 [12] - ImageNet, fp32 accuracy - 74.65						
Calibration	8w8a, 10	74.41 (0.01)	74.41 (0.03)	74.22 (0.02)	74.23 (0.02)	74.16 (0.02)
	4w4a, 10	45.27 (0.04)	43.54 (0.15)	40.46 (0.1)	43.88 (0.09)	46.89 (0.03)
KD	4w4a, 10	71.08	71.26	63.98	70.72	63.59



AdaQuant: Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming 2020

- Per layer optimization

$$\left(\hat{\Delta}_w, \hat{\Delta}_x, \hat{V}\right) = \underset{\Delta_w, \Delta_x, V}{\arg\min} ||WX - Q_{\Delta_w}(W+V) \cdot Q_{\Delta_x}(X)||^2,$$

- Per-layer bit allocations with integer programming
- Batch Normalization Tuning

	AdaQuant	Mixed-Precision (IP)	BN tuning	Bias Tuning
Light-Pipeline	×	✓	✓	×
Heavy-Pipeline	✓	✓	✓	✓

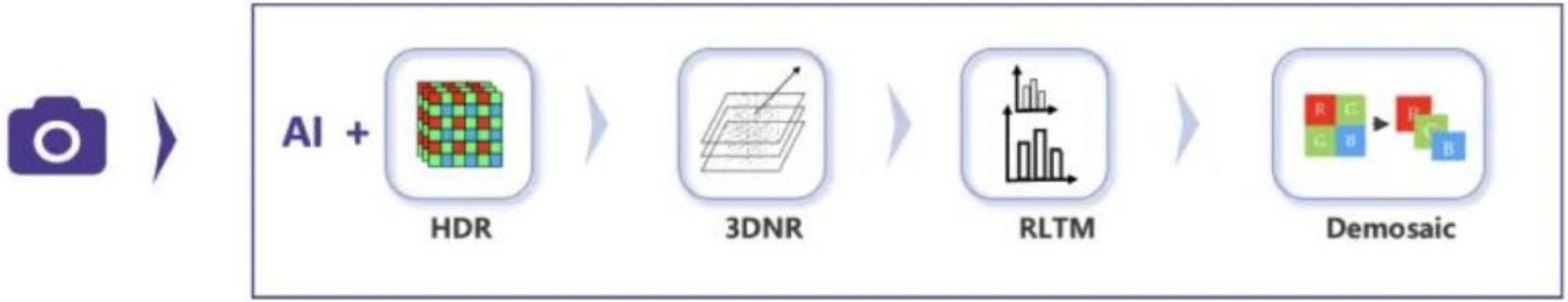
Table 1: Comparison between light and advanced pipelines.

Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and On-Device Training
  - Dataflow, Spatial Computing and Beowulf cluster

Computer Vision Pipelines: ISP

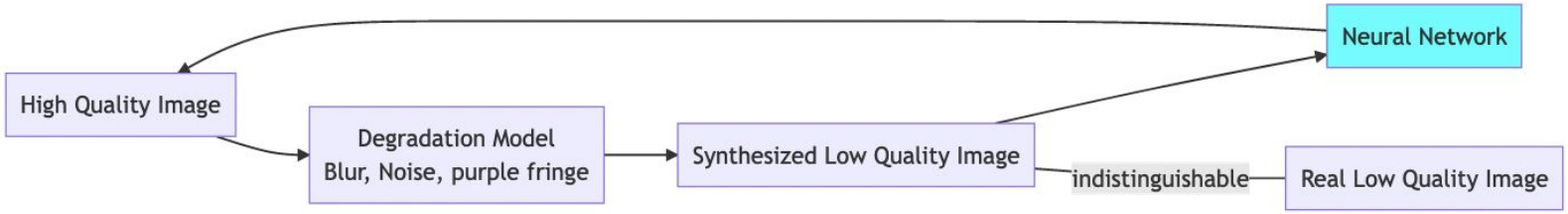
Computer Vision Pipelines: AI-ISP



Why AI-ISP?

- NN solution to Inverse Problem

Explicit model



Implicit model:
Deep Image Prior (2017)

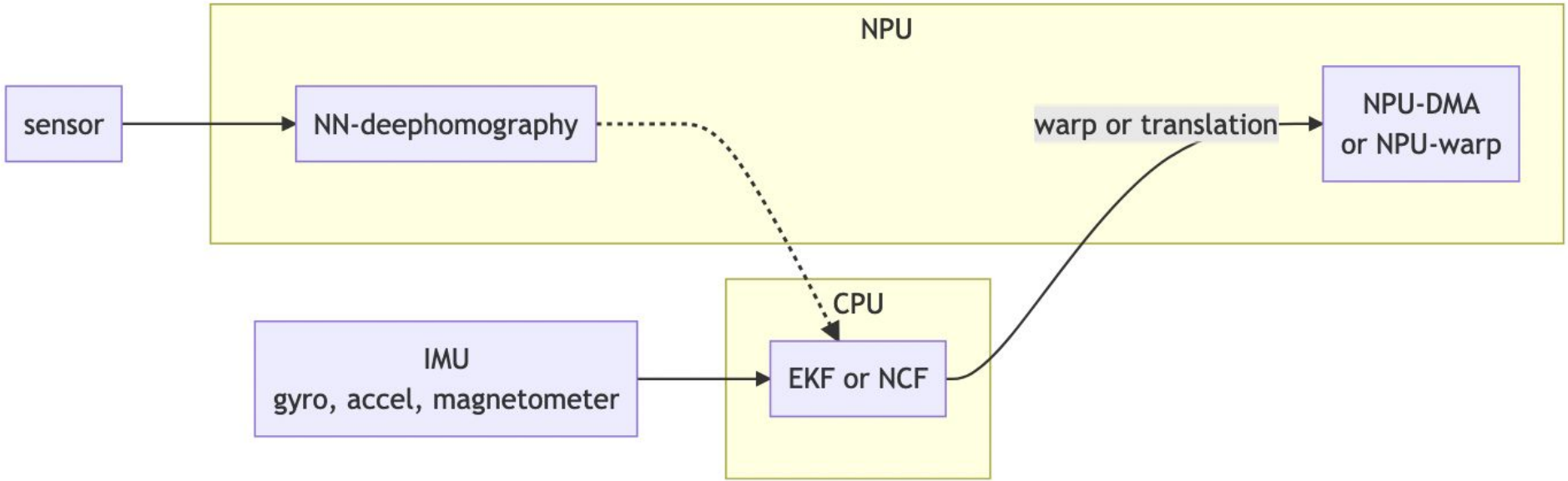
(a) Input (white=masked) (b) Encoder-decoder, depth=6

Why AI-ISP?

- *Reconfigurable*: Allow Post-tape-out changing of ISP pipeline (fix missing features, change stage orders)
- Case study 1: Black level

Why AI-ISP?

- Case study 2: Electronic Image Stabilization via cooperation with a homography-prediction network



AI-ISP is complex: requires DSL

Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and Quantized Training
  - Dataflow, Spatial Computing and Beowulf cluster

Quantized Training

- Int8 OPS is often larger than float16/float32 OPS

OPS	AGX Xavier	AGX Orin	H100
Int8	34T	254T	2000T
Float16 / Float32	8.5T	5.2T	1000T / 500T

DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients

- Uniform stochastic quantization of gradients
  - 6 bit for ImageNet, 4 bit for SVHN
- Simplified scaled binarization: only scalar
  - Forward and backward multiplies the bit matrices from different sides.
  - Using scalar binarization allows using bit operations
- Floating-point-free inference even when with BN
- Future work
  - BN requires FP computation during training
  - Require FP weights for accumulating gradients

$$\text{uniform-quant}_k(\mathbf{W}) \stackrel{\text{def}}{=\!=} 2\max(|\mathbf{W}|)\,\text{quant}_k\left(\frac{\mathbf{W}}{2\max(|\mathbf{W}|)}\right)$$

Floating Point Scaling Factor

Integer

Quantization Methods

- Deterministic Quantization

$$\mathrm{Q}_k^{det}(\mathbf{X}) = \alpha\,\mathrm{Q}_k(\tilde{\mathbf{X}}) + \beta \approx \mathbf{X}$$

- Stochastic Quantizaiton

$$\mathrm{Q}_k^{stoc}(\mathbf{X}) = \alpha\,\mathrm{Q}_k(\tilde{\mathbf{X}} + \frac{\xi}{2^k - 1}) + \beta \approx \mathbf{X}$$

$$\xi \sim U(-\frac{1}{2}, \frac{1}{2})$$

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \beta}{\alpha}$$

$$\alpha = \max(\mathbf{X}) - \min(\mathbf{X})$$

$$\beta = \min(\mathbf{X})$$

Injection of noise realizes the sampling.

Quantization of Weights, Activations and Gradients

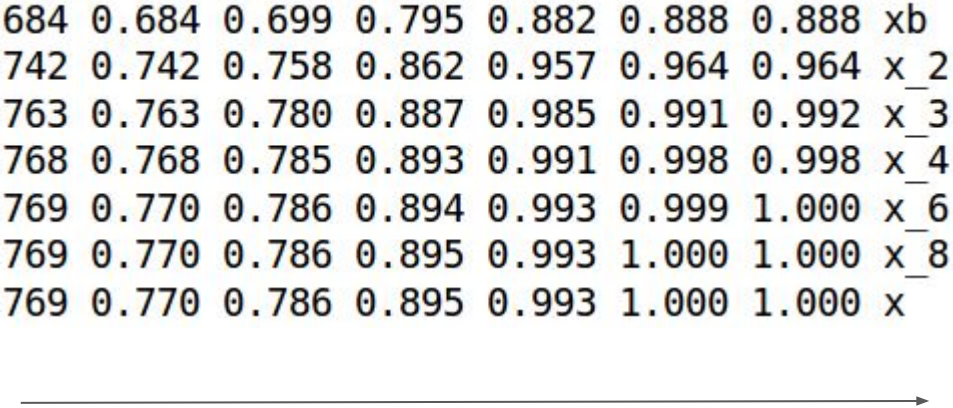
- A half #channel 2-bit AlexNet (same bit complexity as XNOR-net)

Quantization Method	Balanced Determini	Unbalanced Deterministic	Stochastic
Weights	**0.469**	0.346	0.120
Activations	0.315	**0.469**	diverge
Gradients	diverge	diverge	**0.469**
XNOR-net		0.442*	
BNN		0.279*	

Quantization Error measured by Cosine Similarity

- Wb_sn is n-bit quantizaiton of real W
- x is Gaussian R. V. clipped by tanh

Wb_s1	Wb_s2	Wb_s3	Wb_s4	Wb_s6	Wb_s8	W
0.684	0.684	0.699	0.795	0.882	0.888	0.888 xb
0.742	0.742	0.758	0.862	0.957	0.964	0.964 x_2
0.763	0.763	0.780	0.887	0.985	0.991	0.992 x_3
0.768	0.768	0.785	0.893	0.991	0.998	0.998 x_4
0.769	0.770	0.786	0.894	0.993	0.999	1.000 x_6
0.769	0.770	0.786	0.895	0.993	1.000	1.000 x_8
0.769	0.770	0.786	0.895	0.993	1.000	1.000 x



Saturates

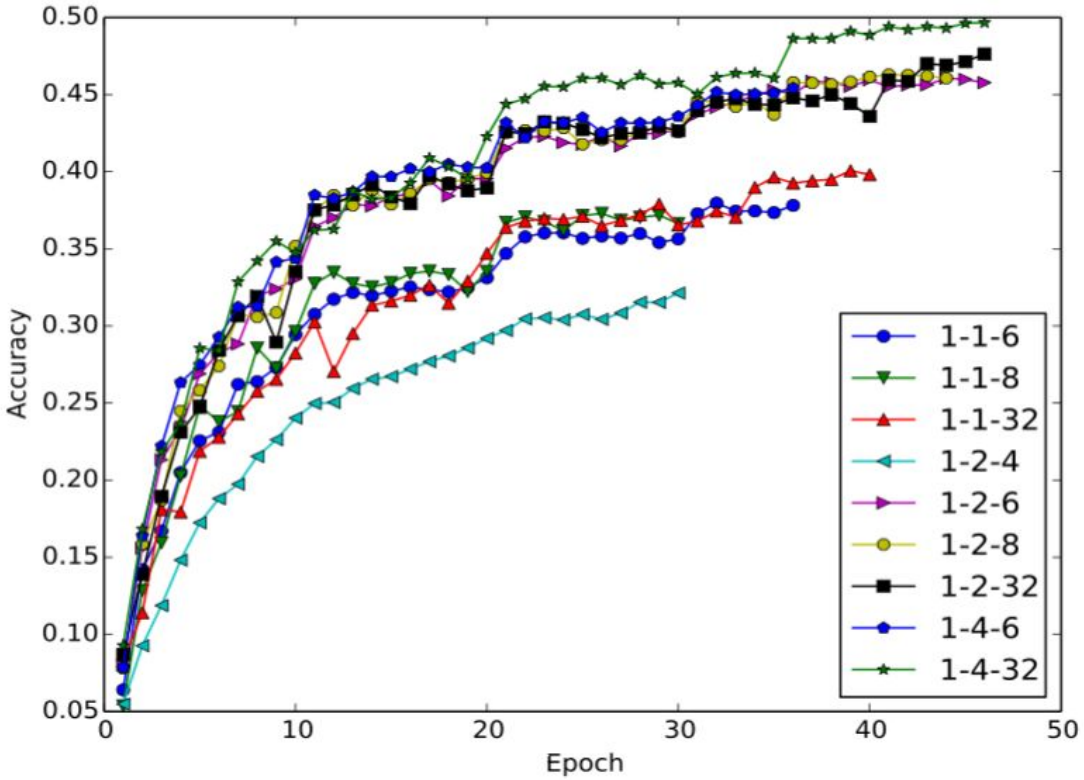
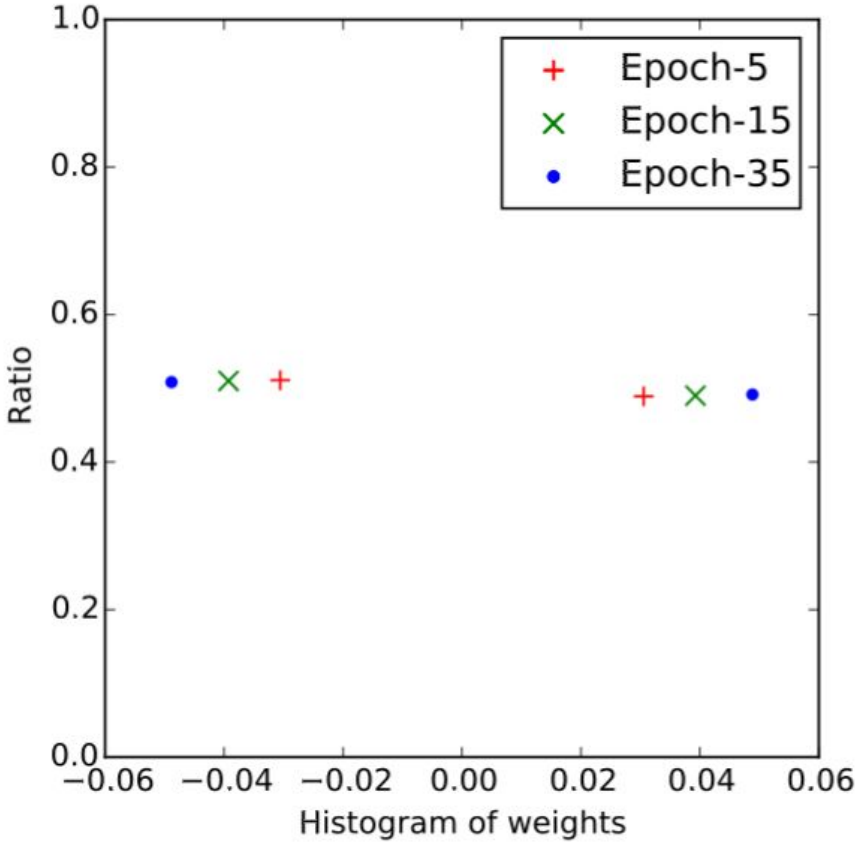
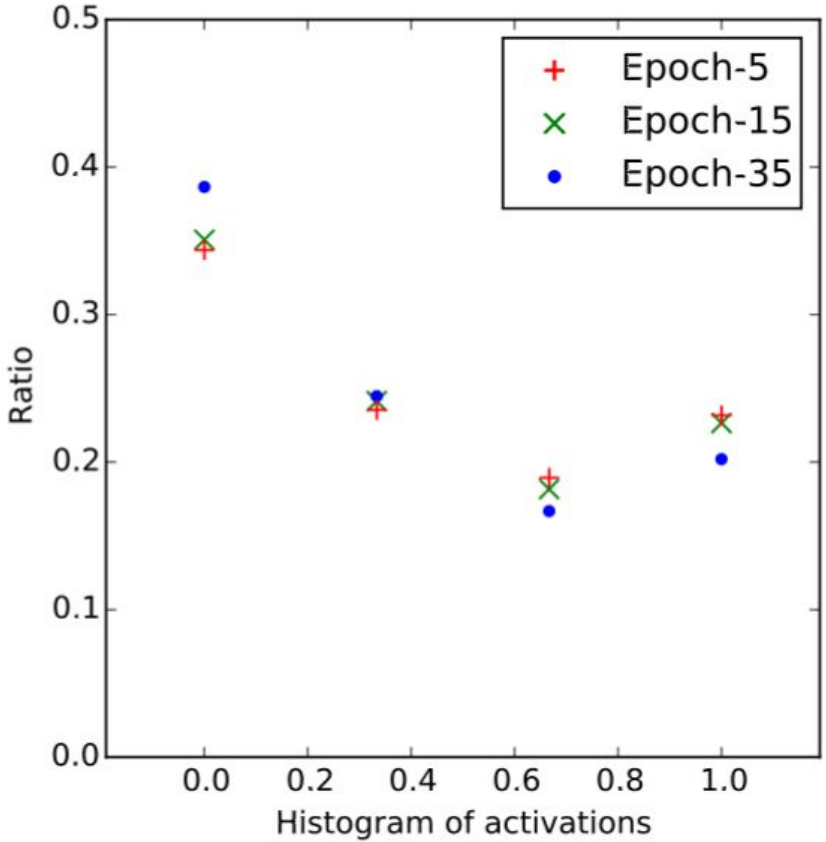


Figure 1: Prediction accuracy of AlexNet variants on Validation Set of ImageNet indexed by epoch number. "W-A-G" gives the specification of bitwidths of weights, activations and gradients. E.g., "1-2-4" stands for the case when weights are 1-bit, activations are 2-bit and gradients are 4-bit. The figure is best viewed in color.



(a)

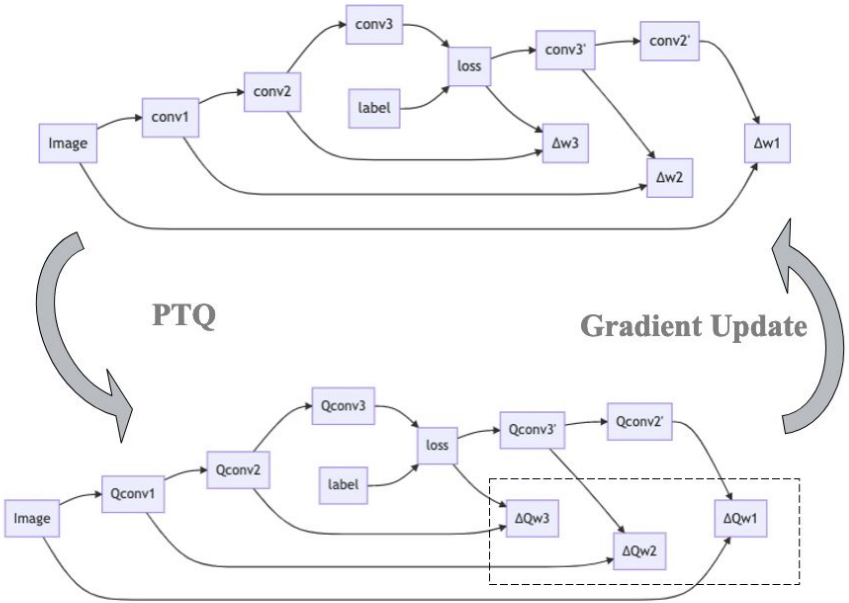


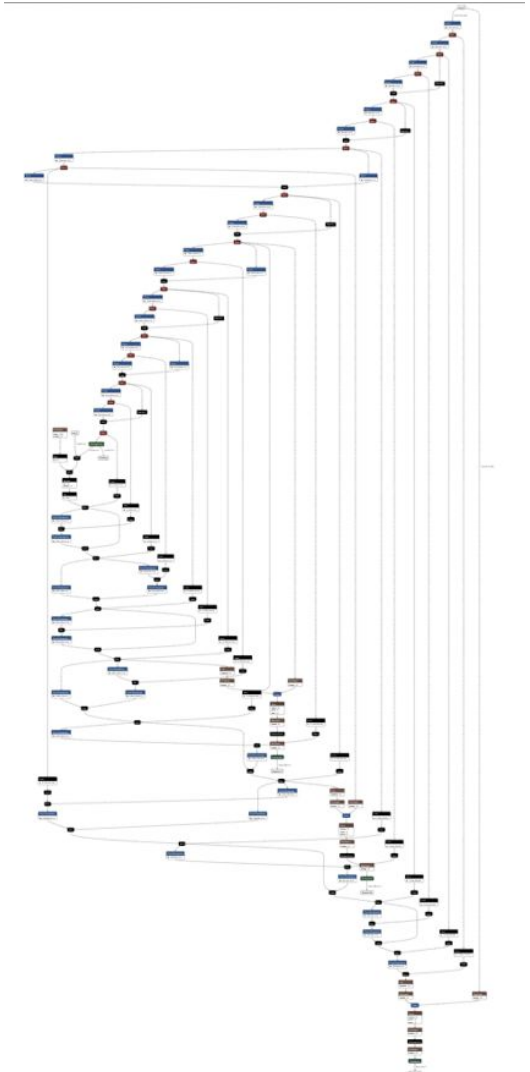
(b)

Figure 3: (a) is histogram of weights of layer “conv3” of “1-2-6” AlexNet model at epoch 5, 15

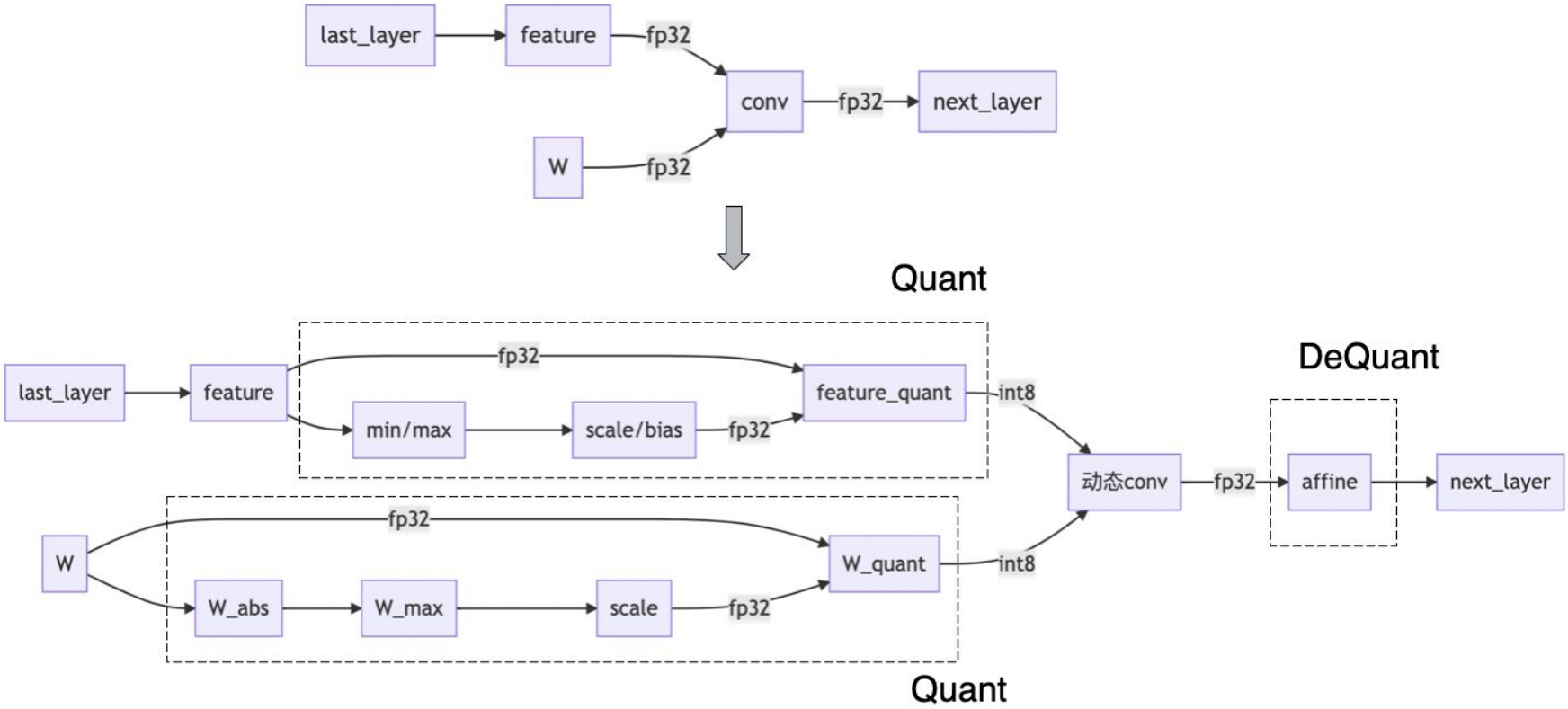
Quantized Training Method 1: Static Quantization

- Training as Inference, the "Delta-W" U-Net

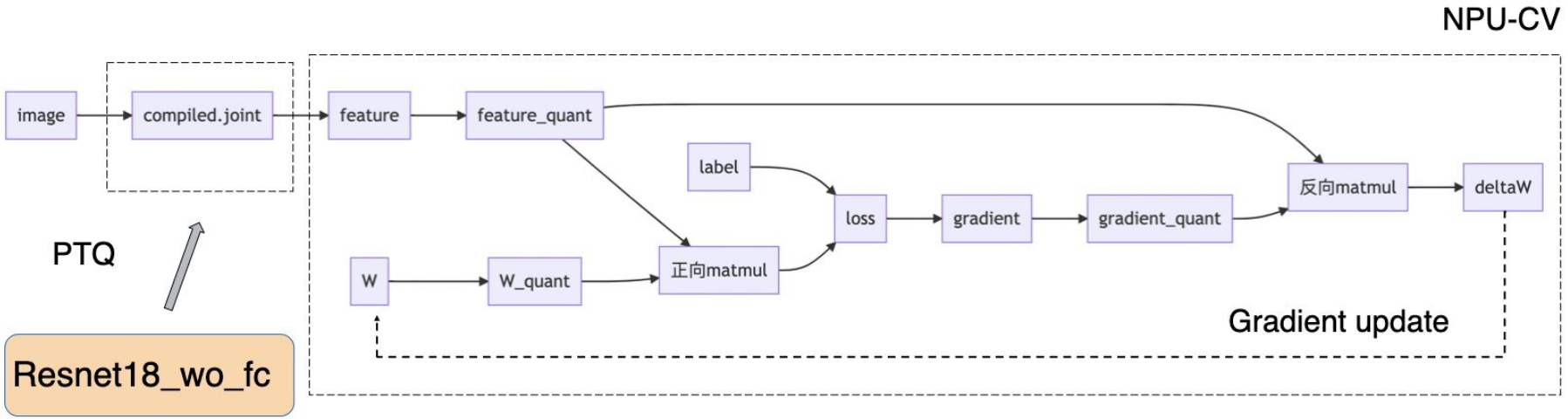




Quantized Training Method 2: Dynamic Quantization



Quantized Training Method 2:
Dynamic Quantization (finetune only)



Outline

- Codesign for Realtime 1080p Processing on a Low-end FPGA
  - Quantized Neural Network
  - Pipeline Simplification
  - a Minimalist Architecture for Accelerator
- Codesign for "General" Neural Networks and AI-ISP
  - Compatibility and Post-Training Quantization
  - AI-ISP and Domain Specific Language
- What's next?
  - Few-shot Learning and Quantized Training
  - Dataflow, Spatial Computing and Beowulf cluster