XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

Mohammad Rastegari

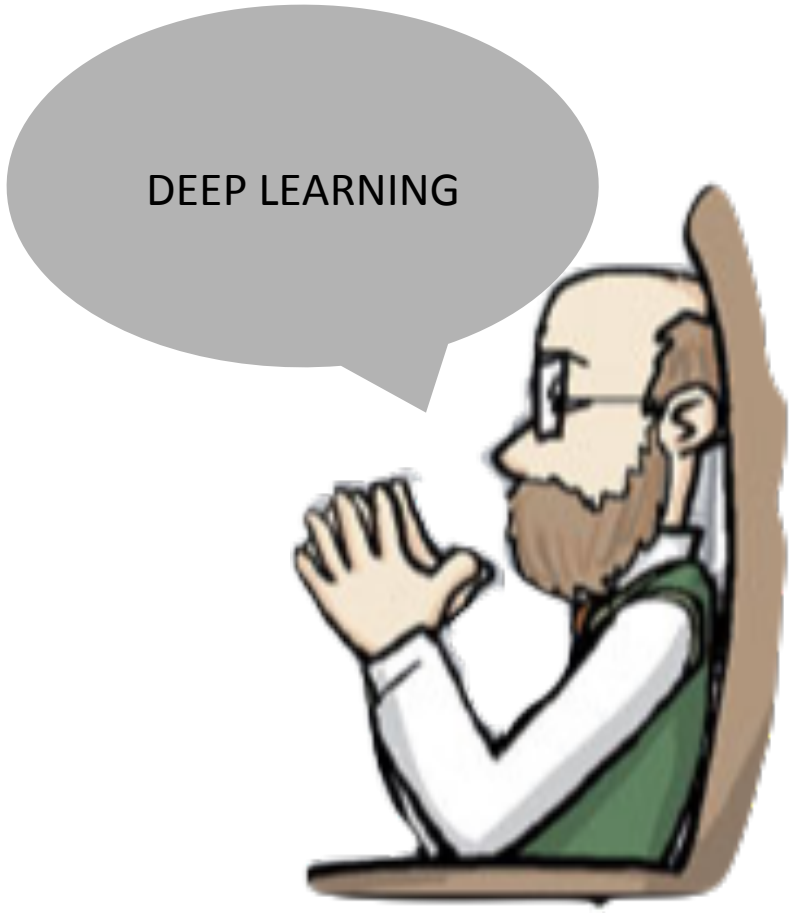
Vicente Ordonez

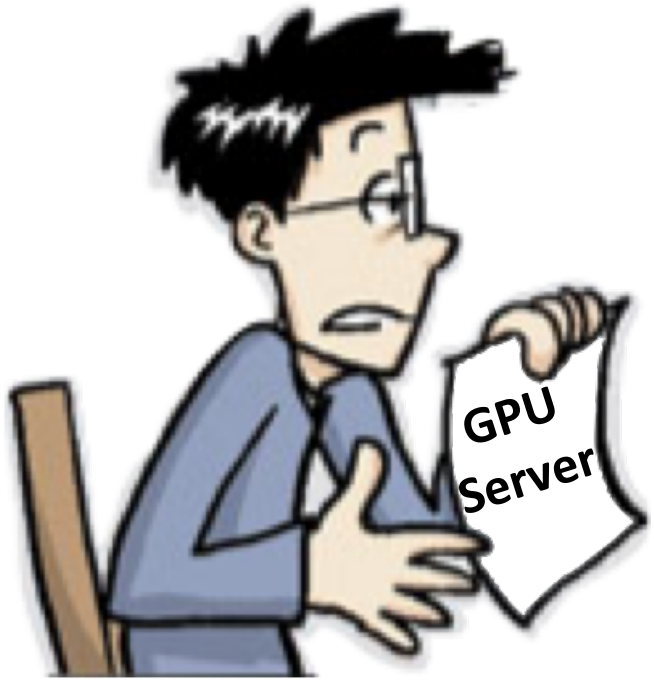
Joseph Redmon

Ali Farhadi

I want to research on a topic with DEAP LEARNING in it?

What should I learn to do well in computer vision research?

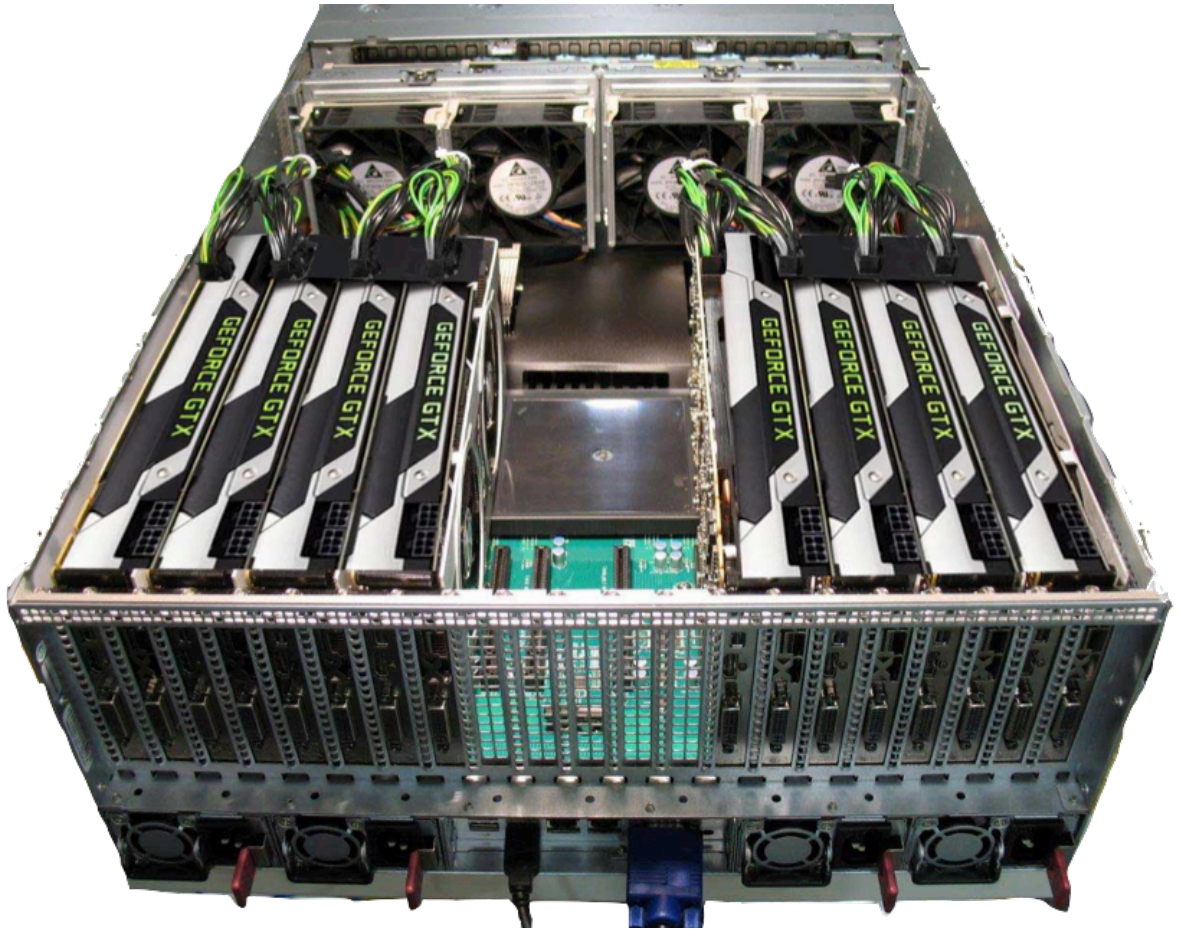

DEEP LEARNING


GPU
Server


Ohhh No!!!

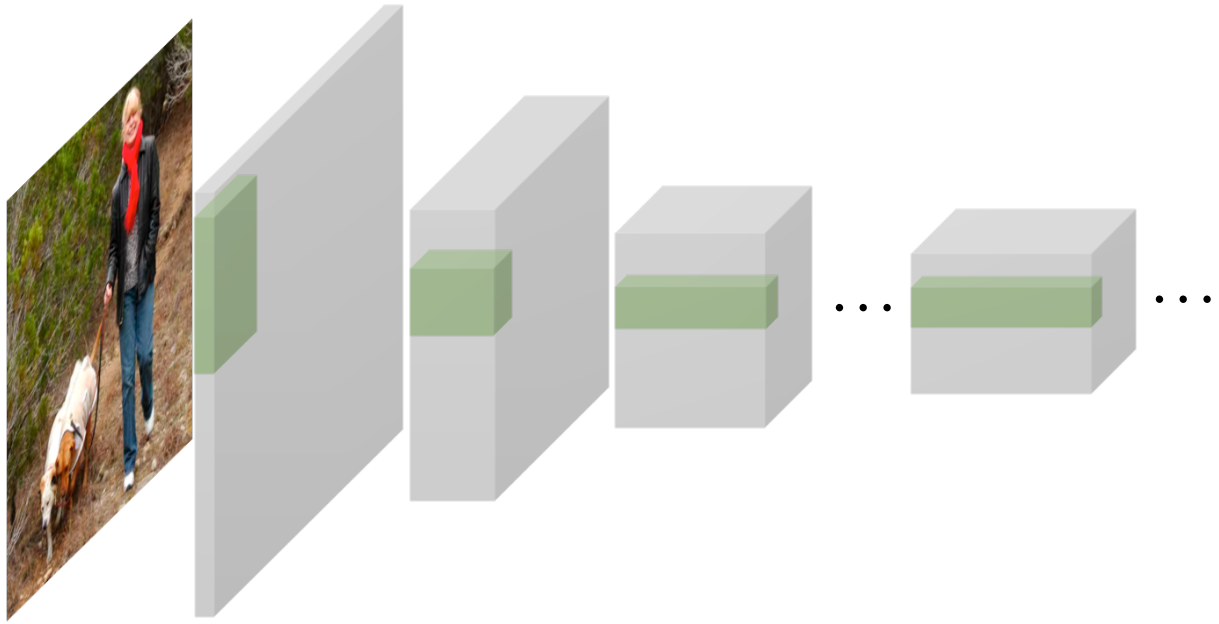
State of the art recognition methods

- Very Expensive
  - Memory
  - Computation
  - Power

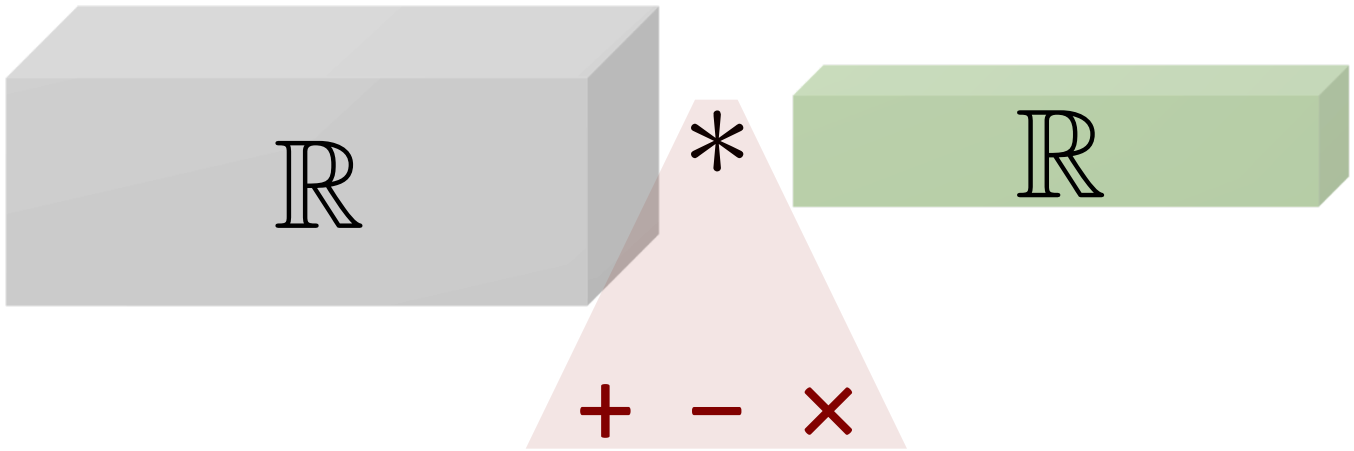



World
Make America
Great Again

Convolutional Neural Networks



GPU !



Number of Operations :

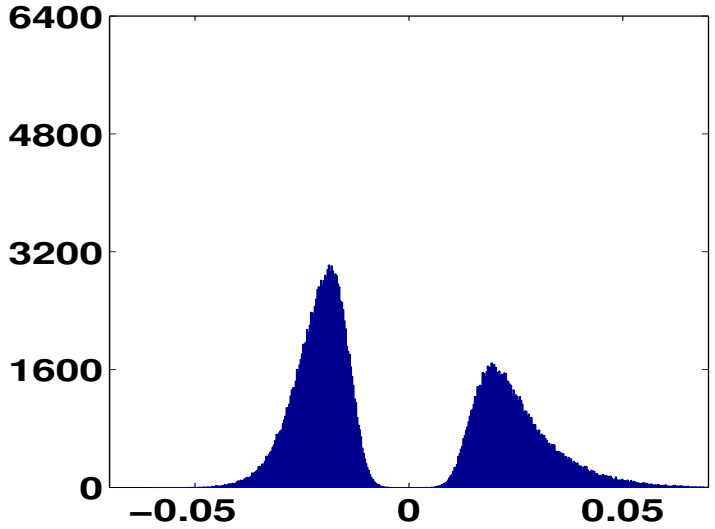
- AlexNet →1.5B FLOPs
- VGG → 19.6B FLOPs

Inference time on CPU :

- AlexNet →~3 fps
- VGG → ~0.25 fps

Lower Precision

[Han et al. 2016]



Reducing Precision

- Saving Memory
- Saving Computation

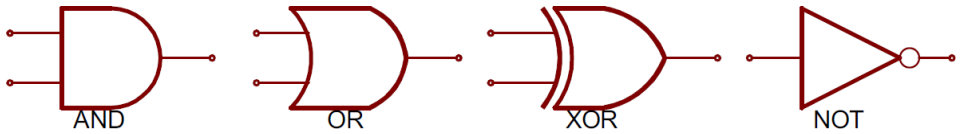
$$\mathbb{R} \rightarrow \mathbb{I} \rightarrow \mathbb{B} \in \{-1, +1\}$$

32-bit 8-bit 1-bit

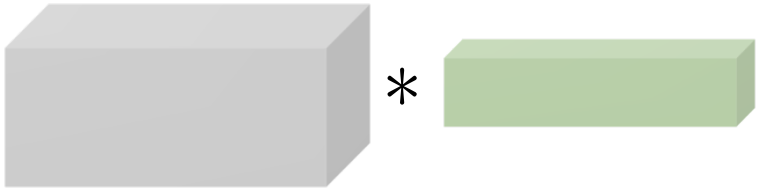
{-1,+1}	{0,1}
MUL	XNOR
ADD, SUB	Bit-Count (popcount)

Why Binary?

- Binary Instructions
  - AND, OR, XOR, XNOR, PoPCount (Bit-Count)



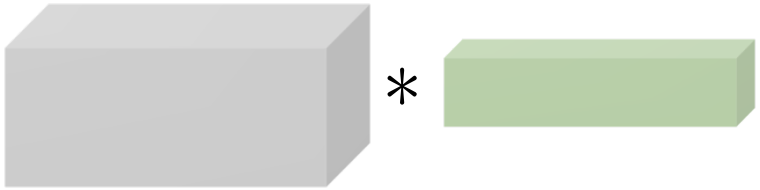
- Low Power Device



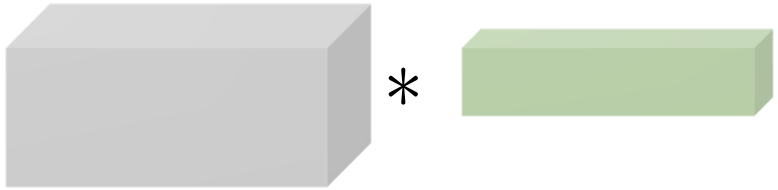
	Operations	Memory	Computation
$\mathbb{R}$ * $\mathbb{R}$	+ − ×	1x	1x

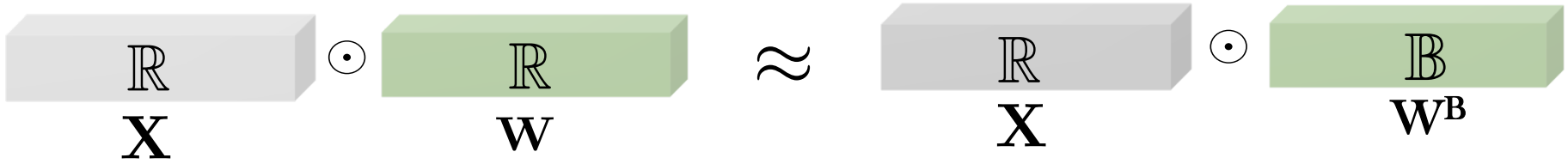
Binary Weight Networks

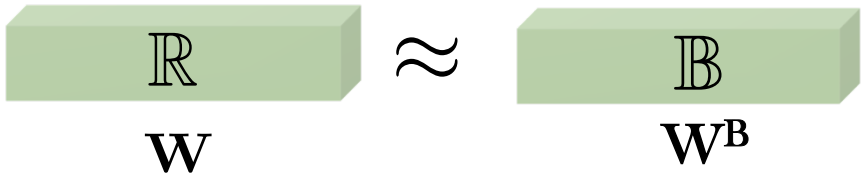
XNOR-Networks



*	Operations	Memory	Computation
$\mathbb{R}$ * $\mathbb{R}$	+ − ×	1x	1x
$\mathbb{R}$ * $\mathbb{B}$	+ −	~32x	~2x
$\mathbb{B}$ * $\mathbb{B}$	XNOR Bit-count	~32x	~58x



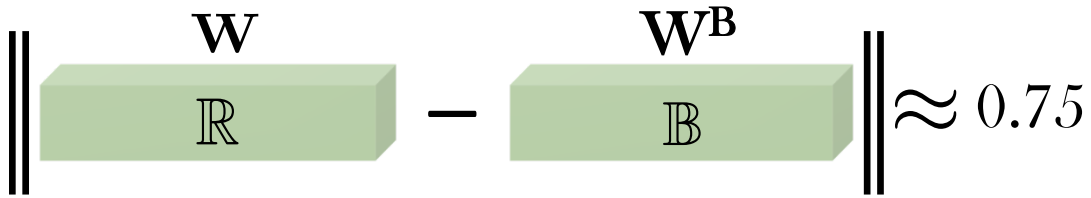




$$\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$$

Quantization Error

$$W^B = \text{sign}(W)$$

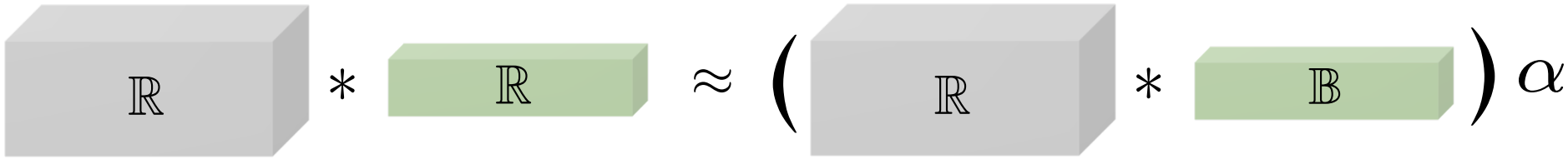


Optimal Scaling Factor

$$\alpha^*, \mathbf{W^{B*}} = \arg\min_{\mathbf{W^B},\alpha} \{||\mathbf{W} - \alpha\mathbf{W^B}||^2\}$$

$$\begin{aligned} \mathbf{W^{B*}} &= \text{sign}(\mathbf{W}) \\ \alpha^* &= \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \end{aligned}$$

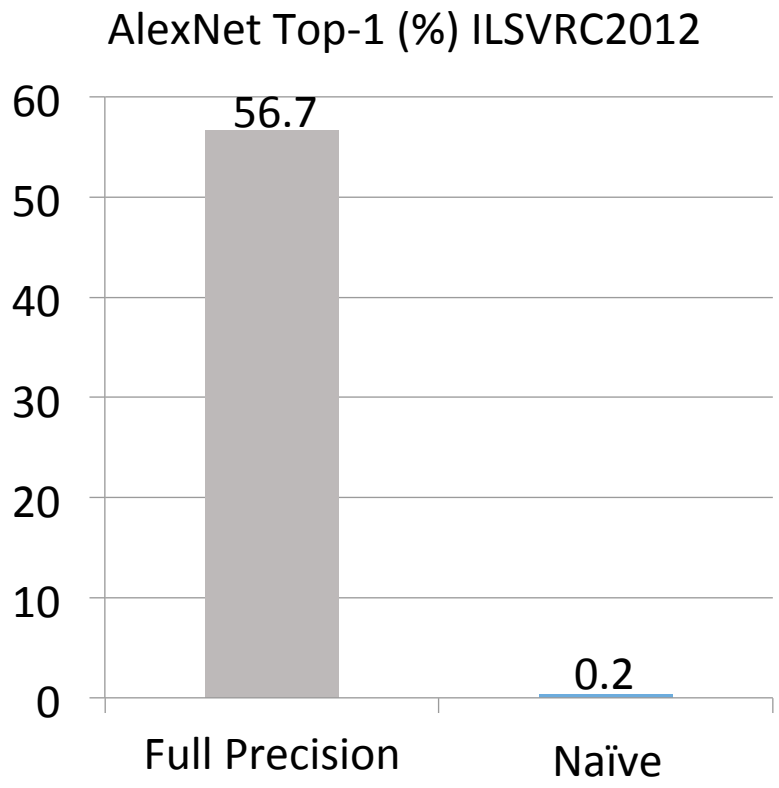
How to train a CNN with binary filters?

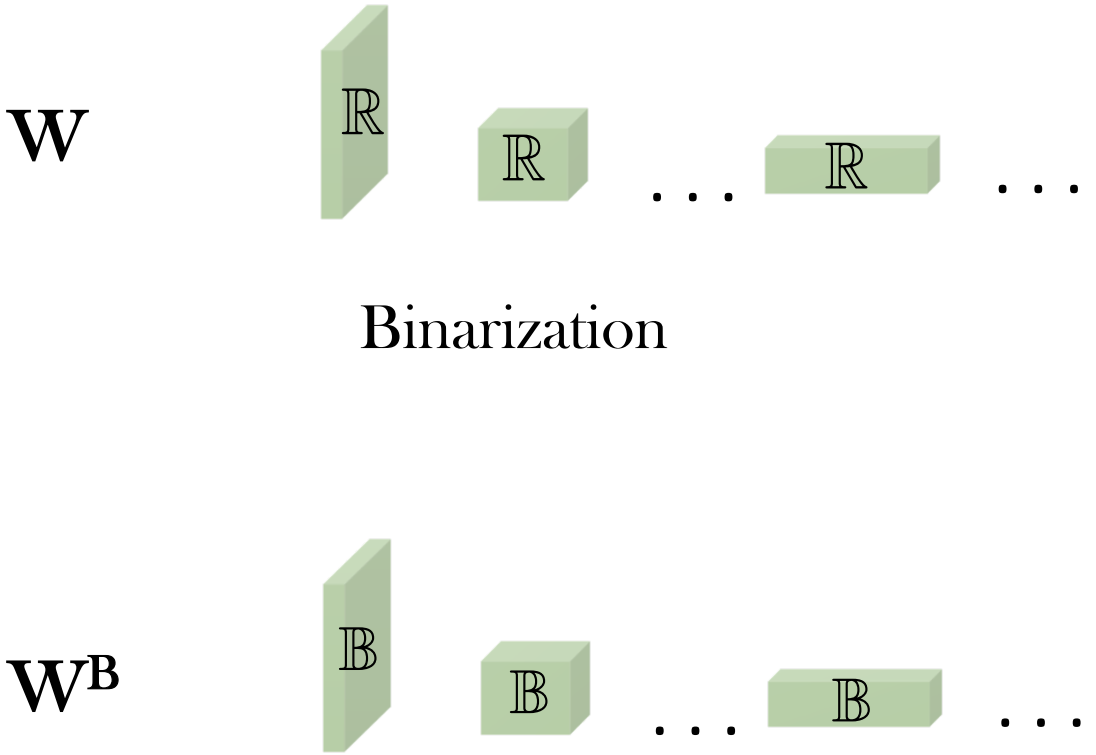


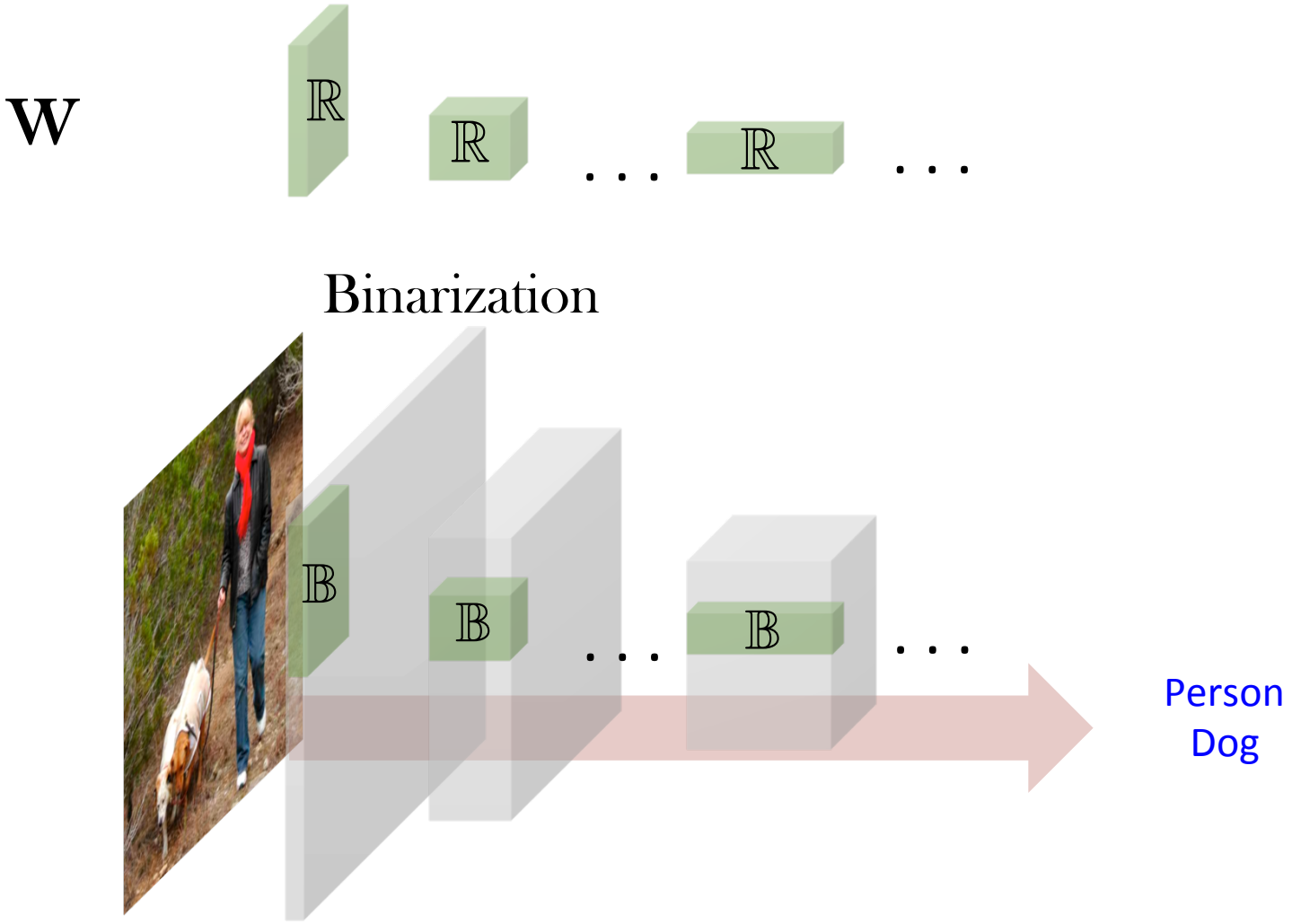
Training Binary Weight Networks

Naive Solution:

1. Train a network with real value parameters
2. Binarize the weight filters


AlexNet Top-1 (%) ILSVRC2012
60
50
40
30
20
10
0
56.7
0.2
Full Precision
Naïve

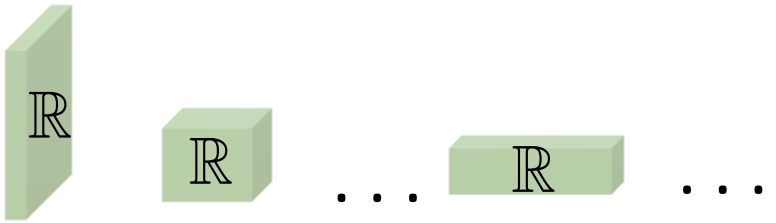

W
ℝ
ℝ
. . .
ℝ
. . .
Binarization
$\mathbf{W}^{\mathbf{B}}$
𝔹
𝔹
. . .
𝔹
. . .


W
ℝ
ℝ
. . .
ℝ
. . .
Binarization
𝔹
𝔹
. . .
𝔹
. . .
Person
Dog

Binary Weight Network

Train for binary weights:

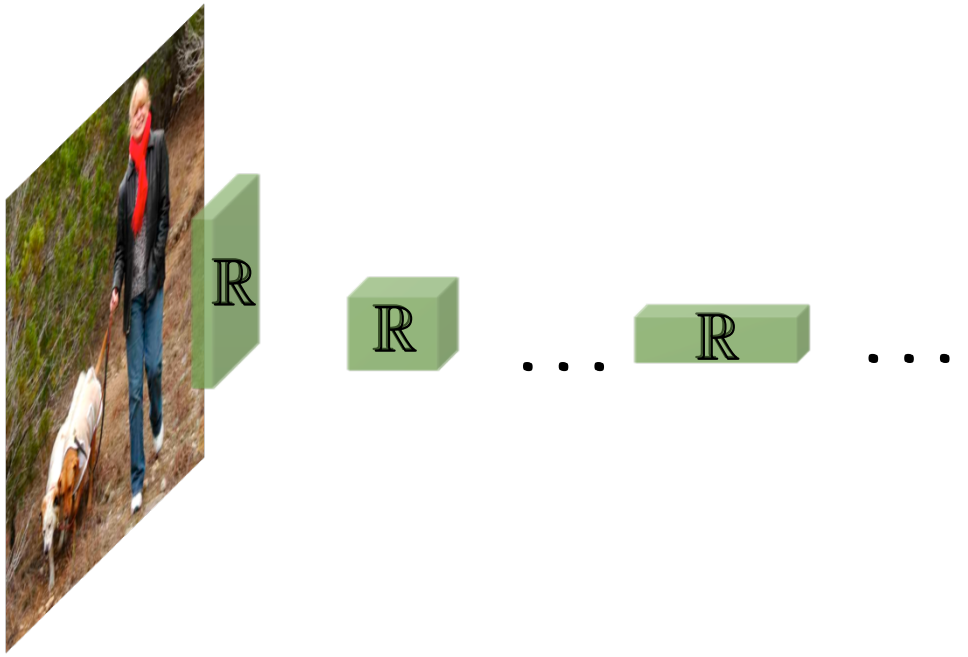
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$



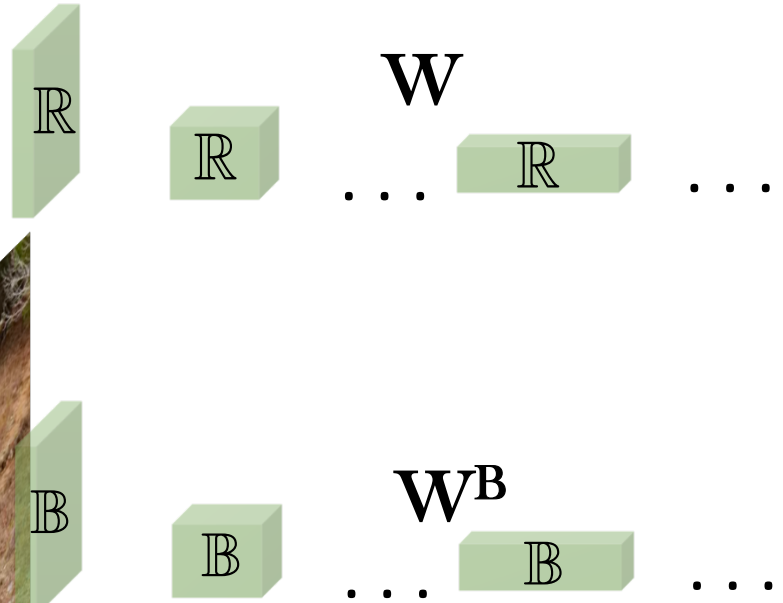
Binary Weight Network

Train for binary weights:

1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$



Binary Weight Network



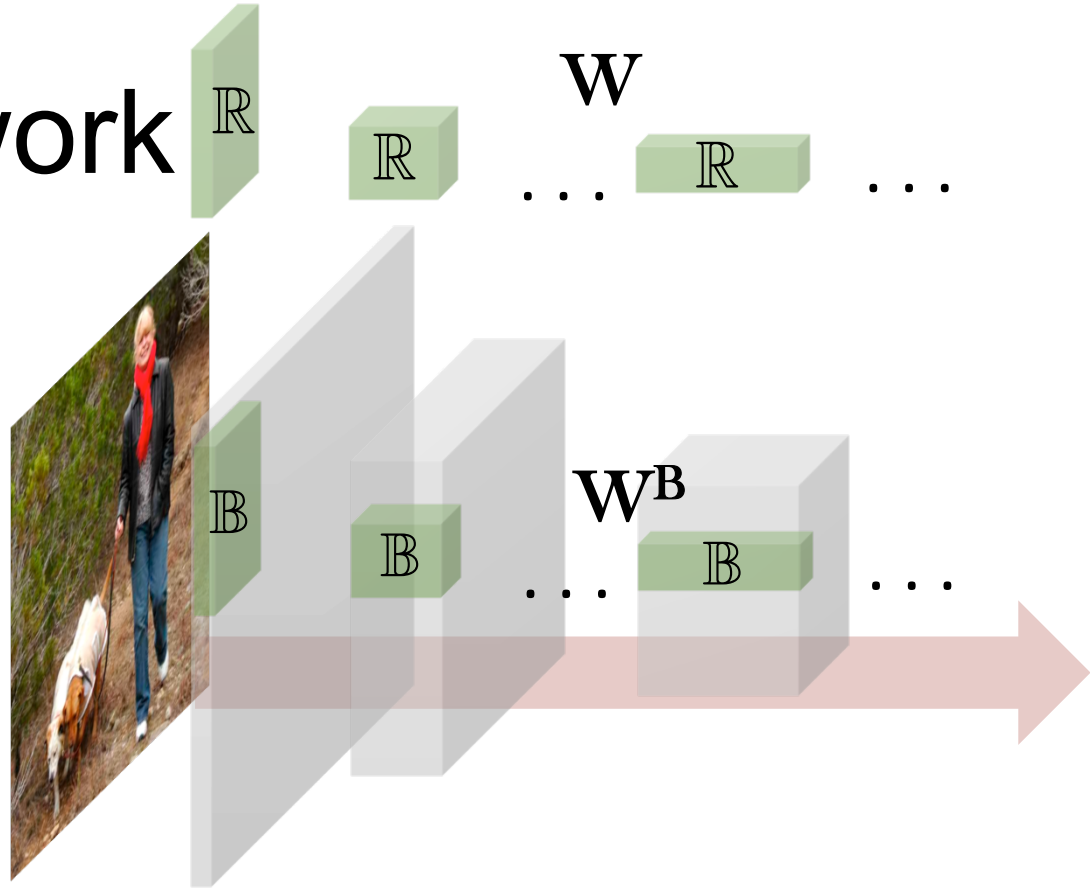
Train for binary weights:

1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$

Binary Weight Network

Train for binary weights:

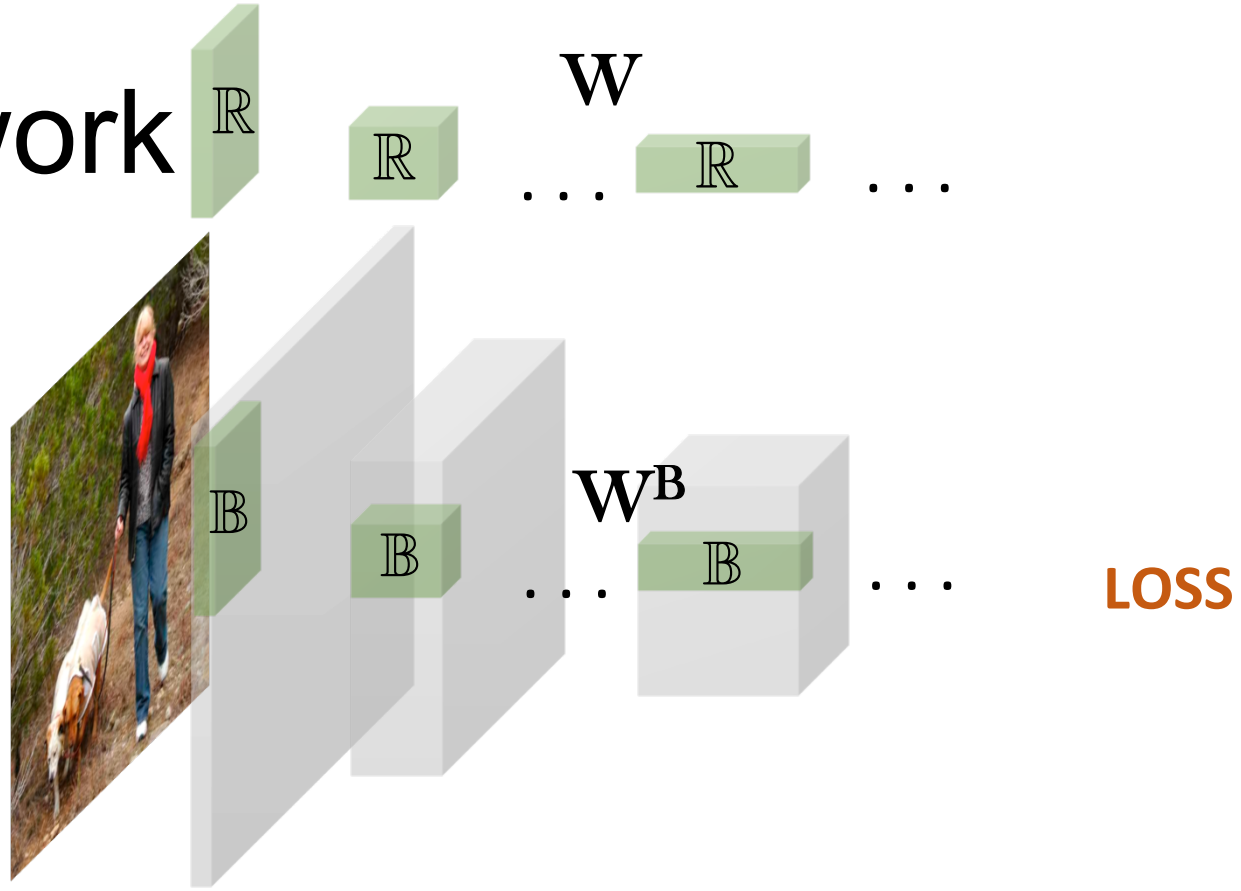
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$



Binary Weight Network

Train for binary weights:

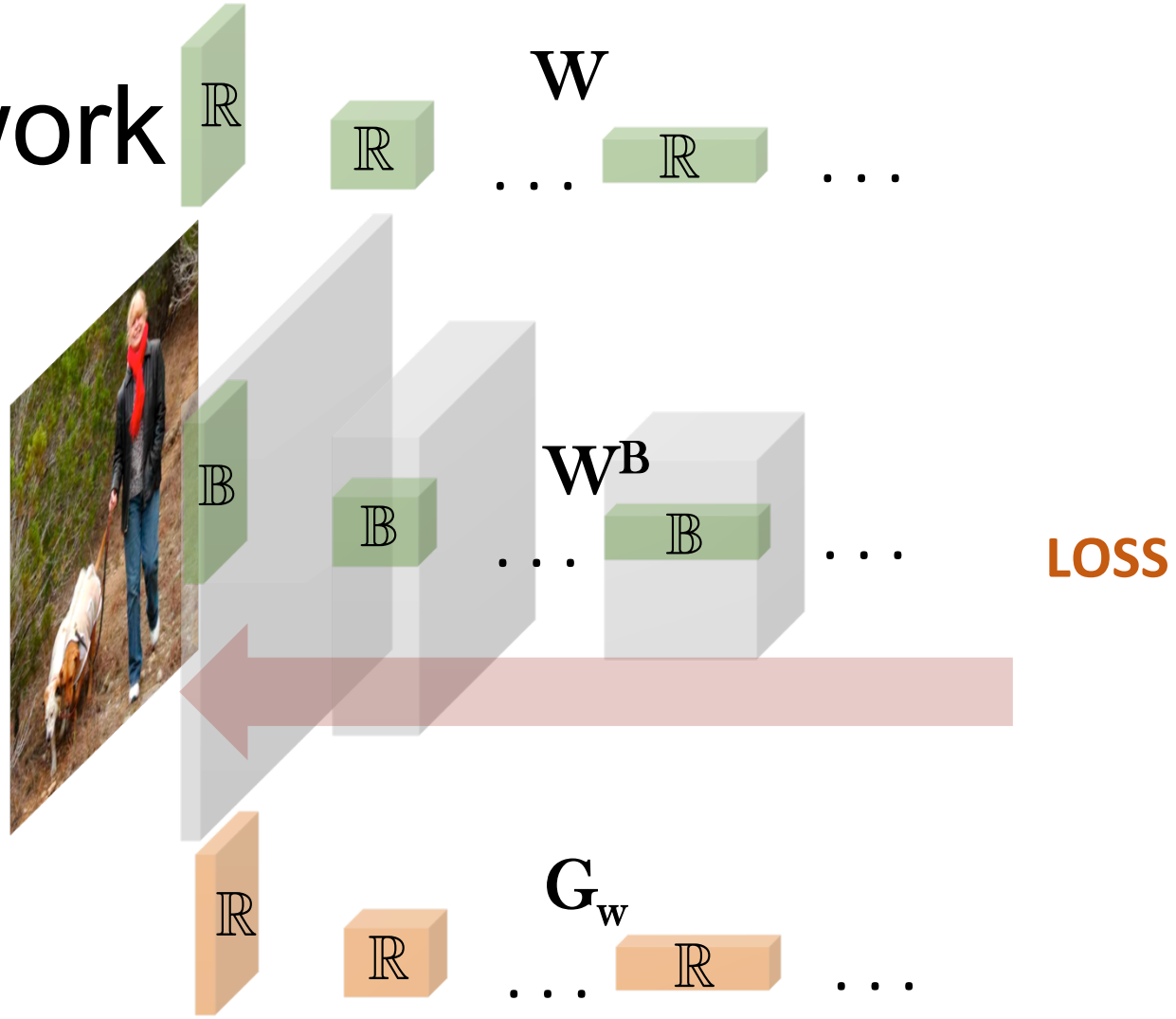
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}}$ = Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$

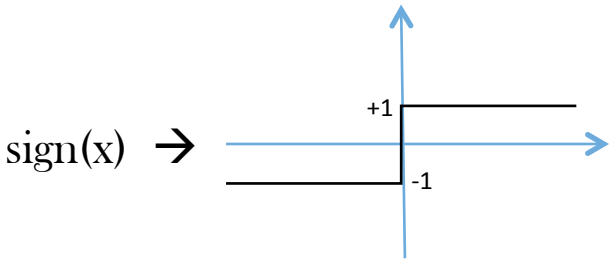


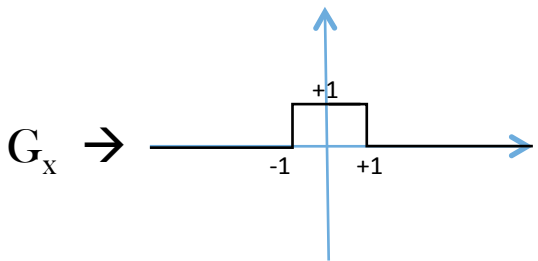
Binary Weight Network

Train for binary weights:

1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$





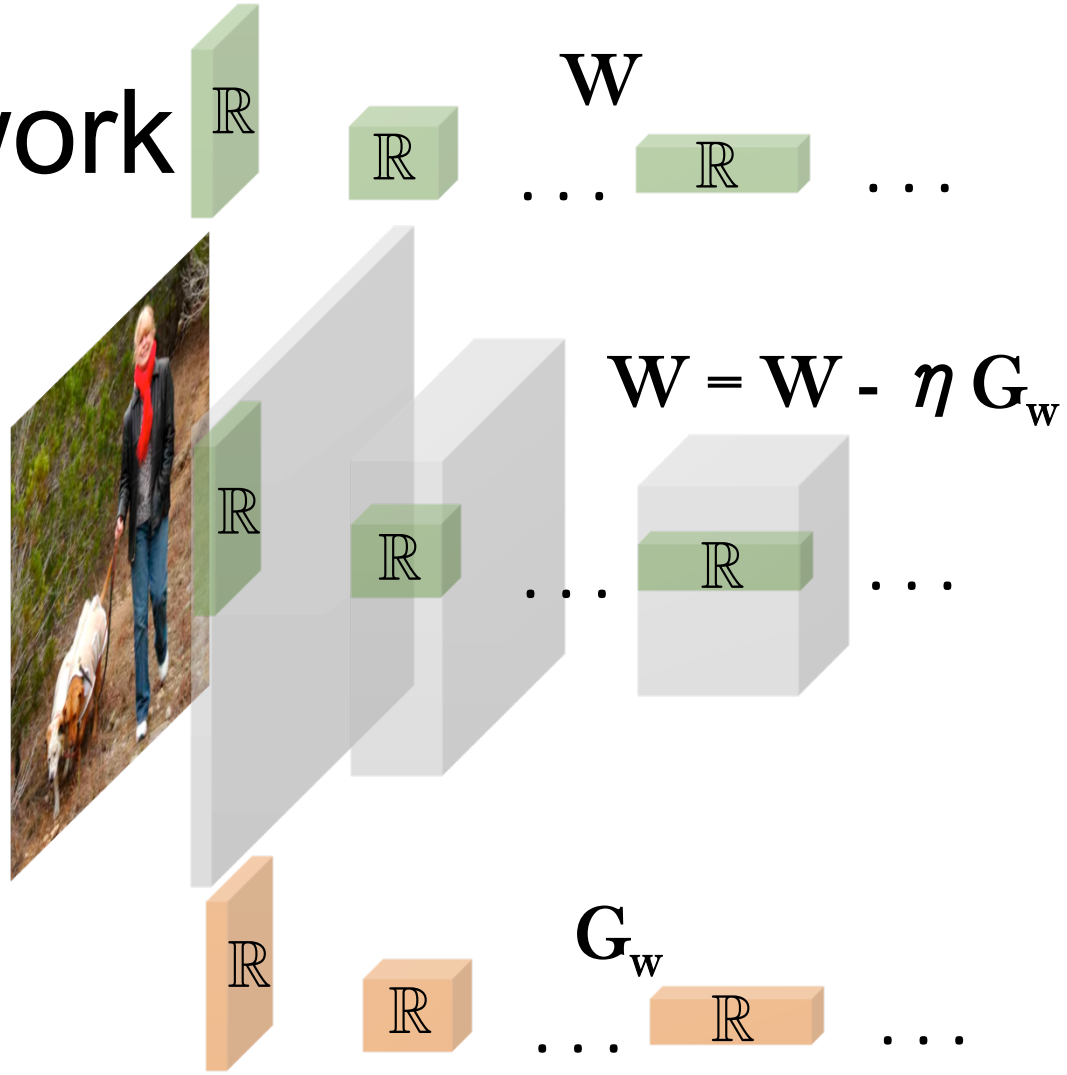


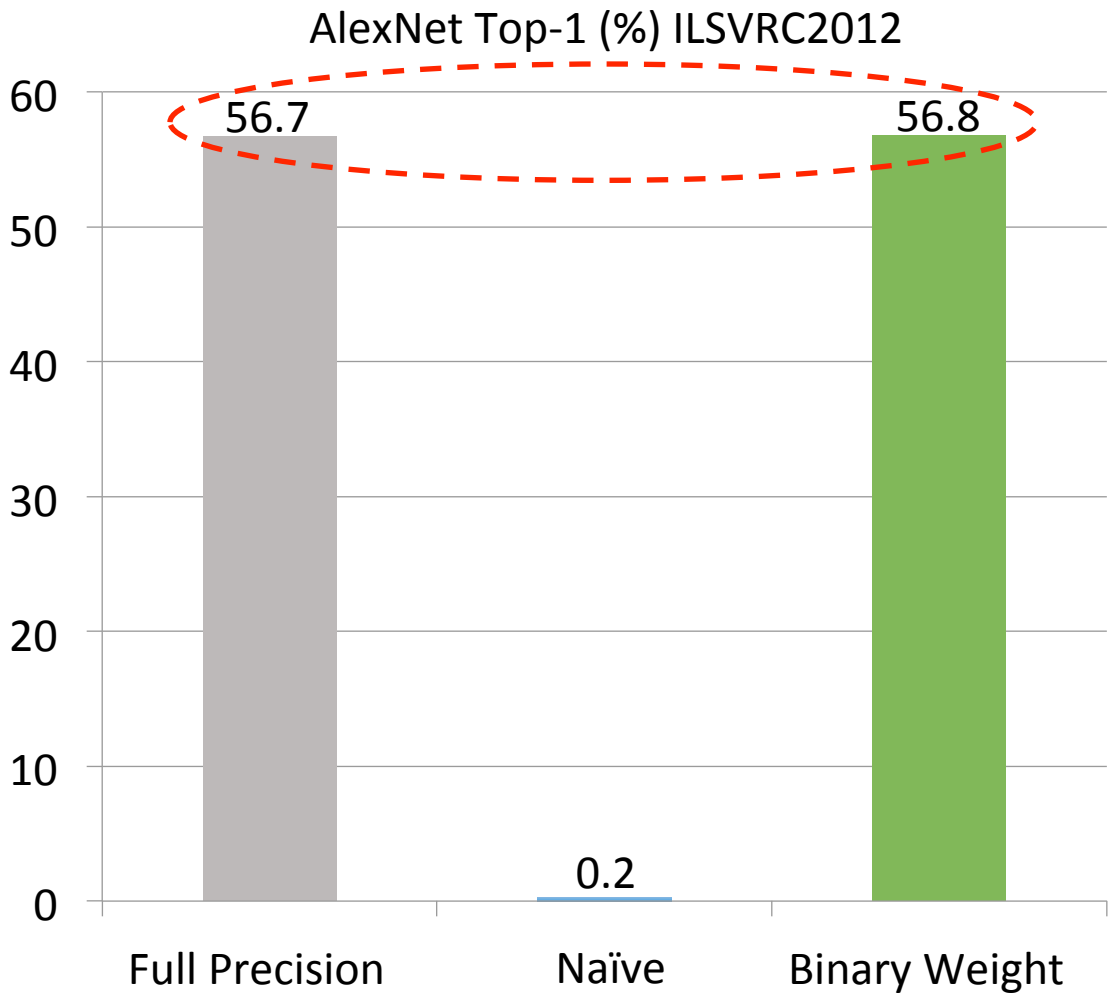
[Hinton et al. 2012]

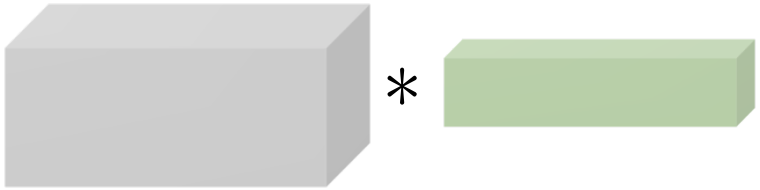
Binary Weight Network

Train for binary weights:

1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}}$ = Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$

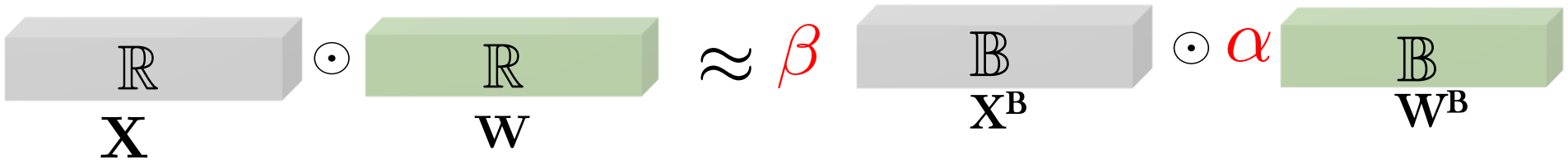



AlexNet Top-1 (%) ILSVRC2012
60
50
40
30
20
10
0
56.7
0.2
56.8
Full Precision
Naïve
Binary Weight

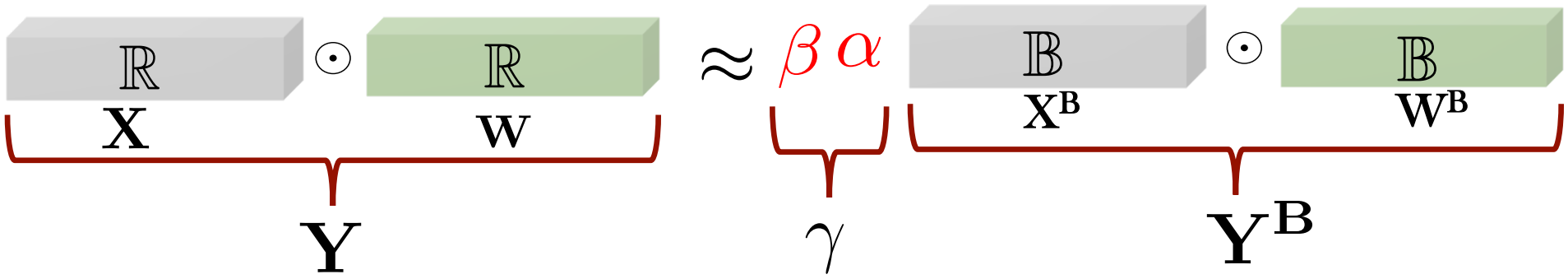


	Operations	Memory	Computation
$\mathbb{R}$ * $\mathbb{R}$	+ − ×	1x	1x
$\mathbb{R}$ * $\mathbb{B}$	+ −	~32x	~2x
$\mathbb{B}$ * $\mathbb{B}$ XNOR-Networks	XNOR Bit-count	~32x	~58x

Binary Input and Binary Weight (XNOR-Net)



Binary Input and Binary Weight (XNOR-Net)



$$\mathbf{Y} \approx \gamma\, \mathbf{Y^B}$$

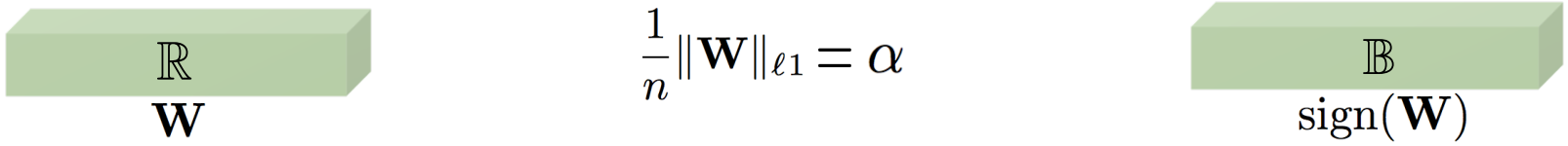
$$\mathbf{Y^{B*}}, \gamma^* = \arg\min_{\mathbf{Y^B},\gamma} \|\mathbf{Y} - \gamma \mathbf{Y^B}\|^2$$

$$\mathbf{Y^{B*}} = \text{sign}(\mathbf{Y}) \quad \gamma^* = \frac{1}{n}\|\mathbf{Y}\|_{\ell 1}$$

$$\mathbf{X^{B*}} = \text{sign}(\mathbf{X}) \quad \mathbf{W^{B*}} = \text{sign}(\mathbf{W})$$

$$\alpha^* = \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \qquad \beta^* = \frac{1}{n}\|\mathbf{X}\|_{\ell 1}$$

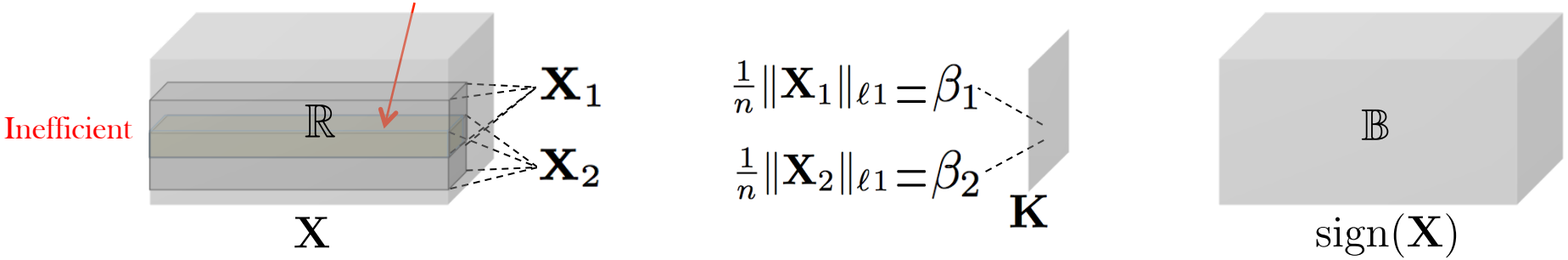
(1) Binarizing Weights



(2) Binarizing Input

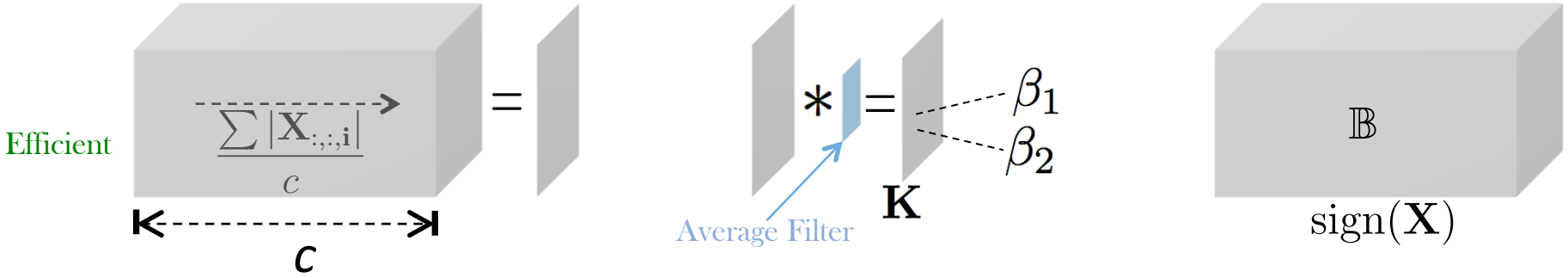
Redundant computation in overlapping areas

Inefficient

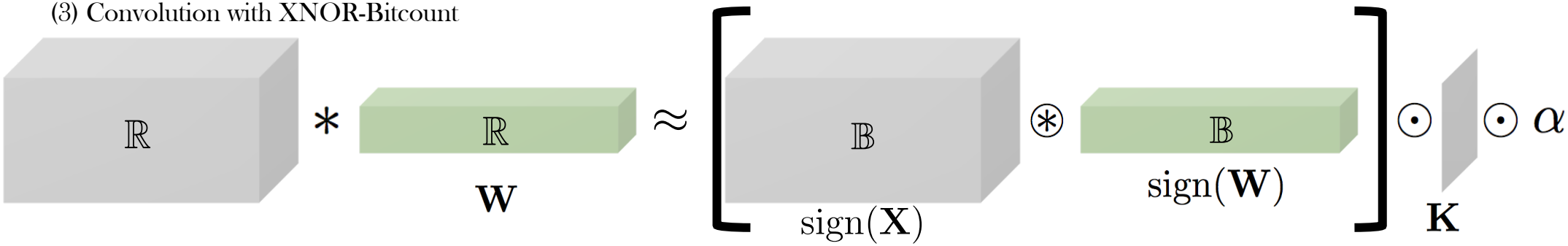


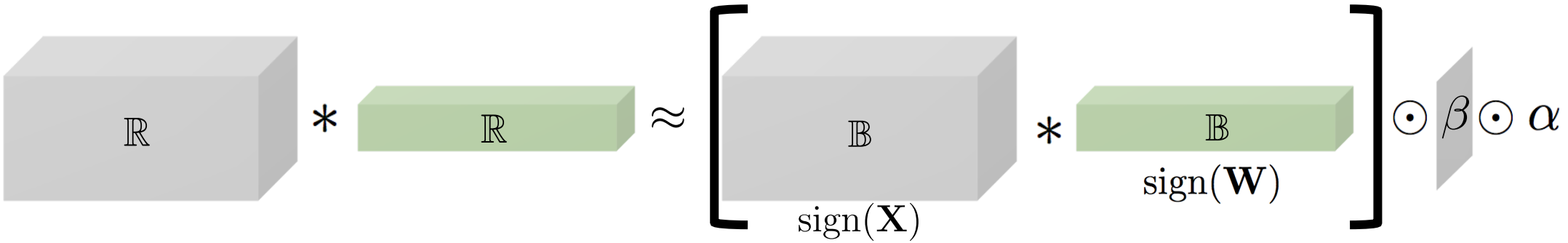
(2) Binarizing Input

Efficient

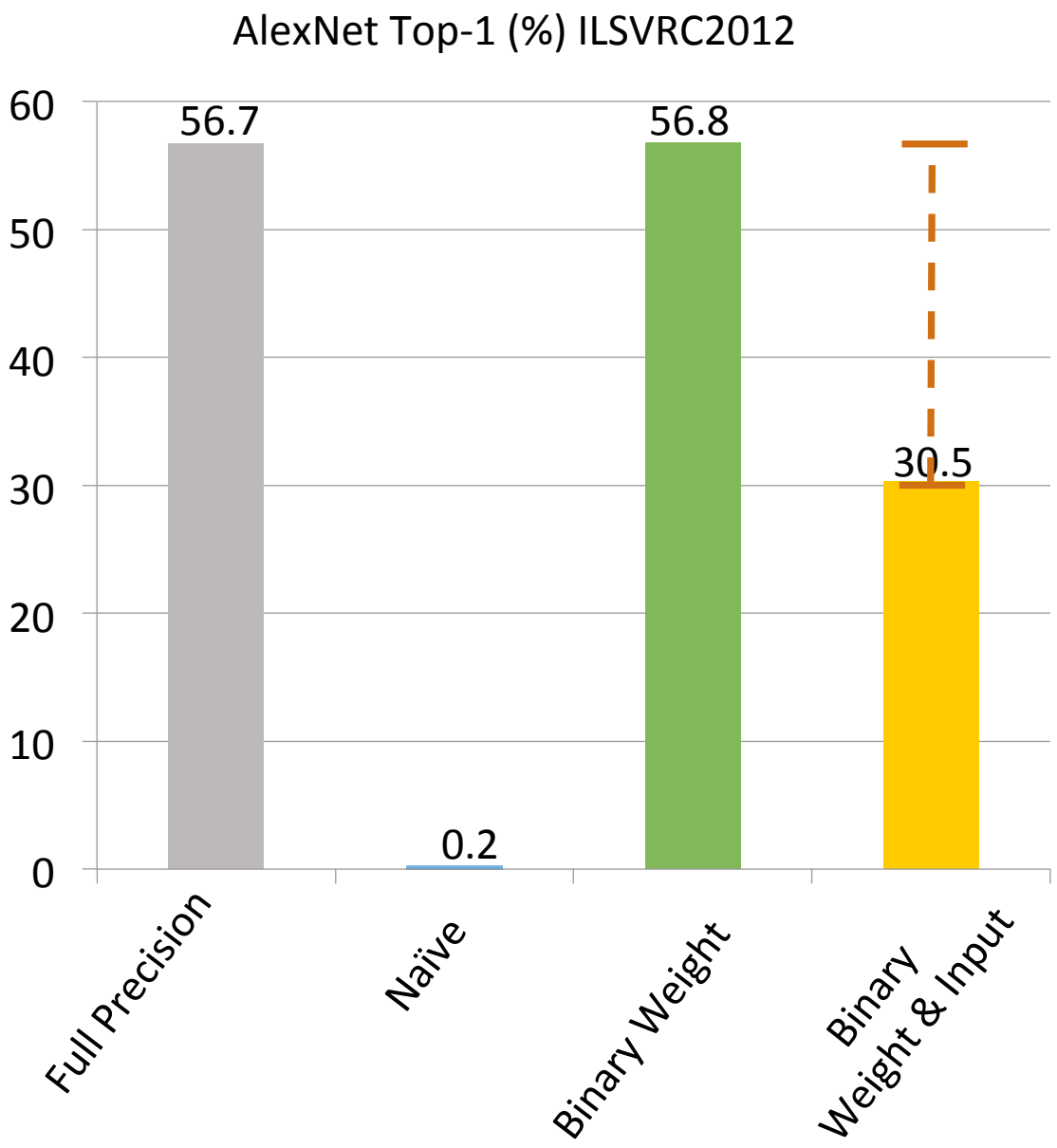


(3) Convolution with XNOR-Bitcount

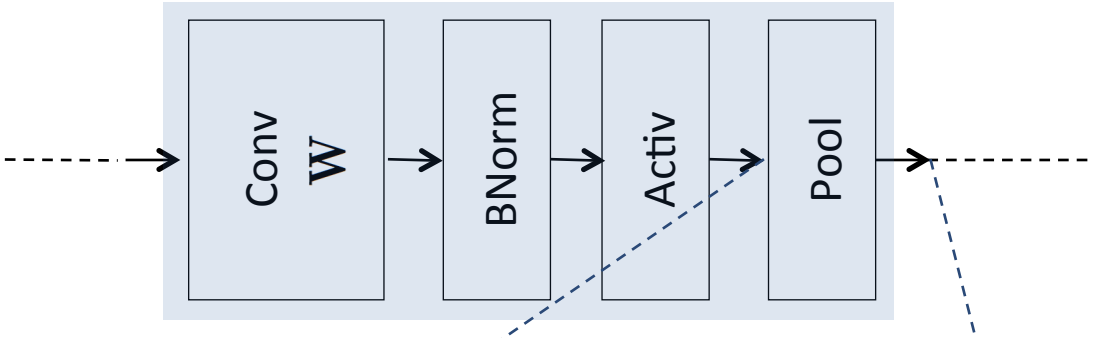




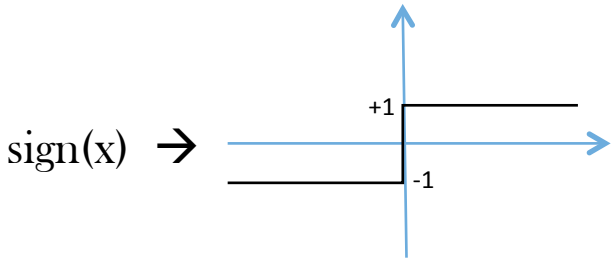
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$


AlexNet Top-1 (%) ILSVRC2012
60
50
40
30
20
10
0
56.7
0.2
56.8
30.5
Full Precision
Naïve
Binary Weight
Binary Weight & Input

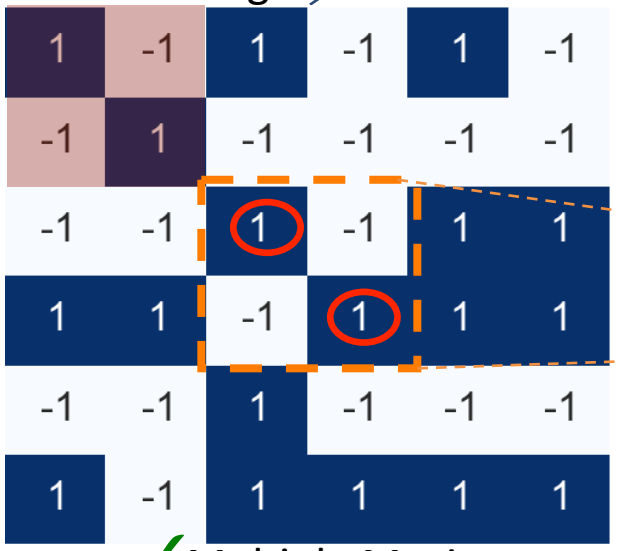
Network Structure in XNOR-Networks



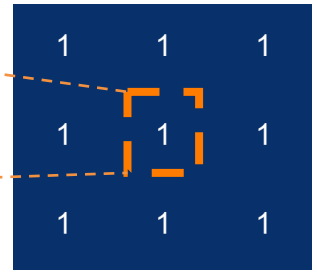
A typical block in CNN



Max-Pooling

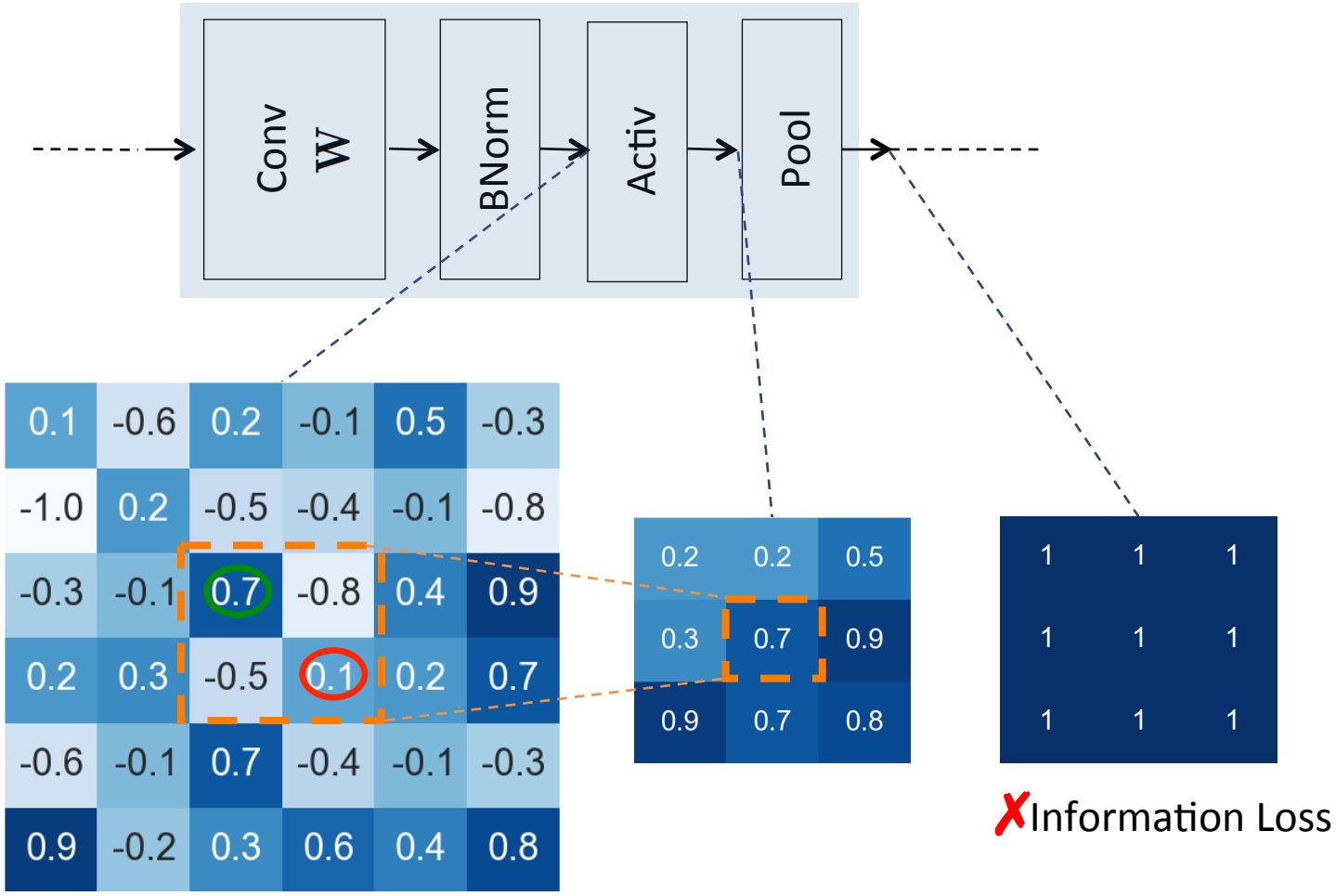


✓Multiple Maximums



✗Information Loss

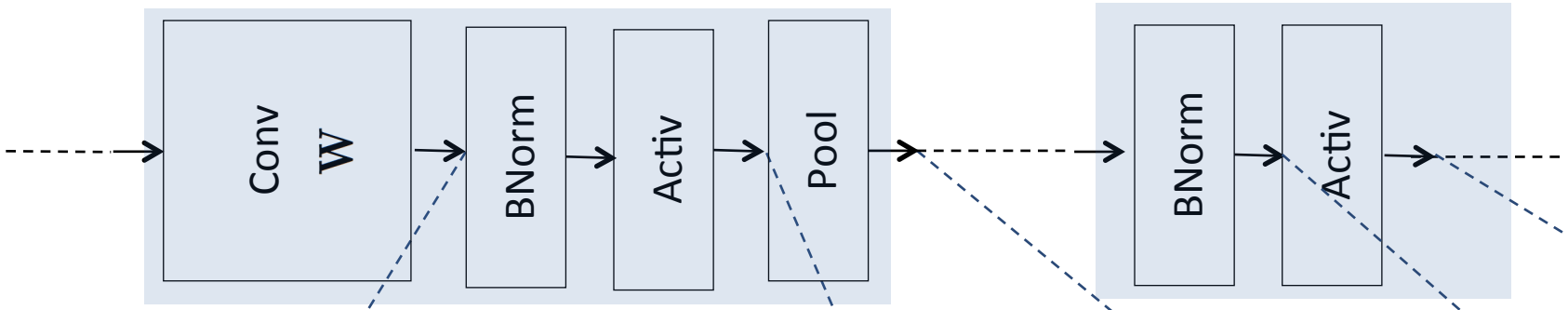
Network Structure in XNOR-Networks



✓Multiple Maximums

✗Information Loss

Network Structure in XNOR-Networks



1	-1	1	-1	1	-1
-1	1	-1	-1	-1	-1
-1	-1	1	-1	1	1
1	1	-1	1	1	1
-1	-1	1	-1	-1	-1
1	-1	1	1	1	1

0.1	-0.6	0.2	-0.1	0.5	-0.3
-1.0	0.2	-0.5	-0.4	-0.1	-0.8
-0.3	-0.1	0.7	-0.8	0.4	0.9
0.2	0.3	-0.5	0.1	0.2	0.7
-0.6	-0.1	0.7	-0.4	-0.1	-0.3
0.9	-0.2	0.3	0.6	0.4	0.8

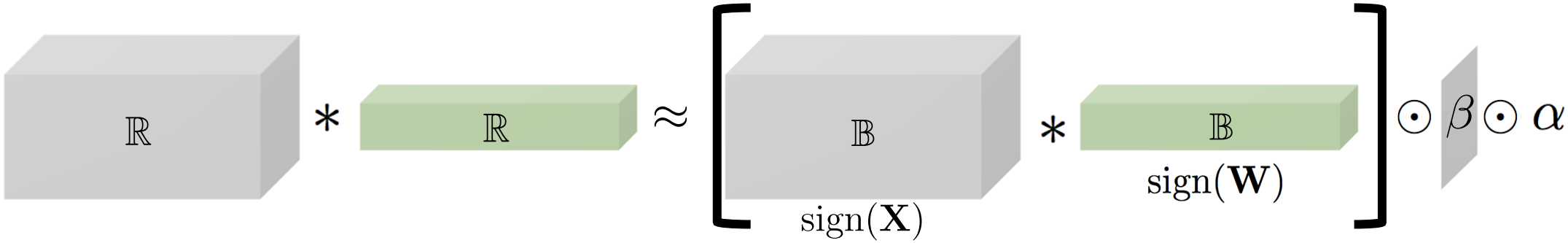
0.2	0.2	0.5
0.3	0.7	0.9
0.9	0.7	0.8

-0.1	-0.2	0.3
0.4	0.3	-0.1
-0.7	0.2	-0.6

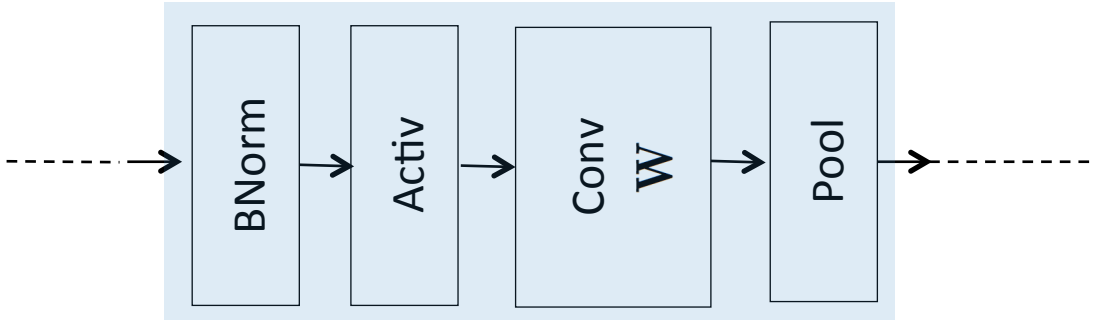
-1	-1	1
1	1	-1
-1	1	-1

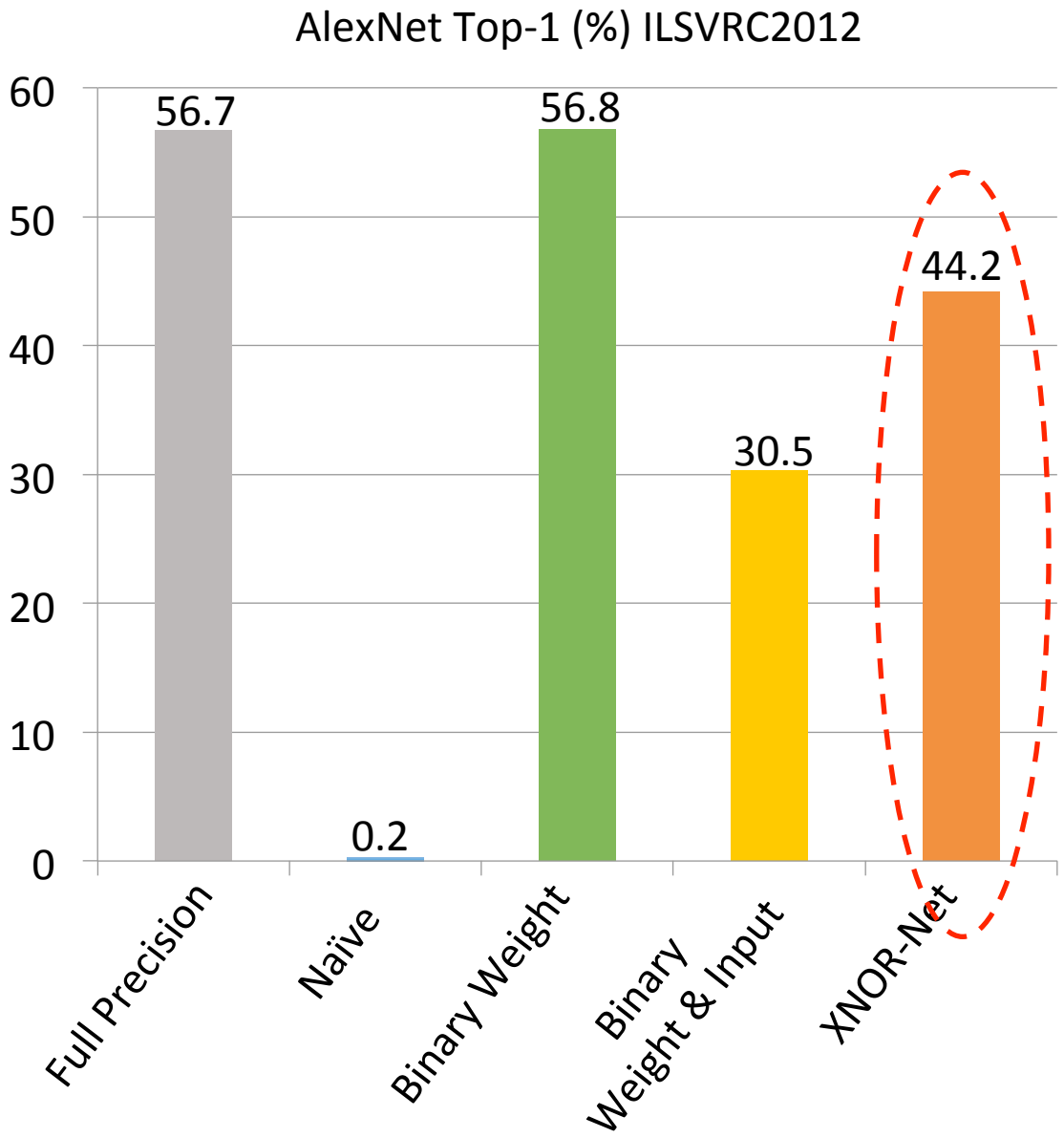
✓Information Loss

✓Multiple Maximums

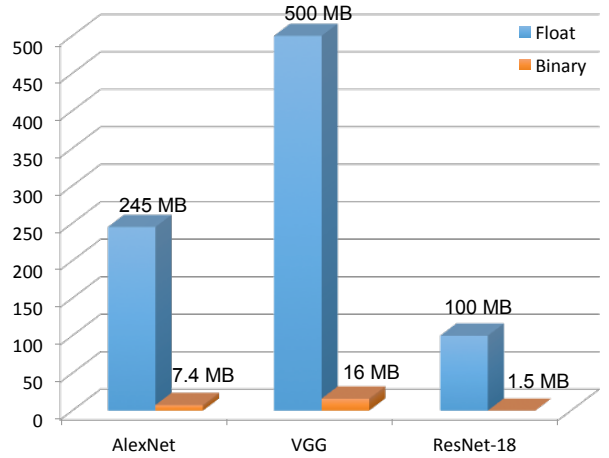


1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}}$ = Backward pass with $\alpha, \mathbf{W^B}$
9. Update $\mathbf{W}$ $(\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}})$

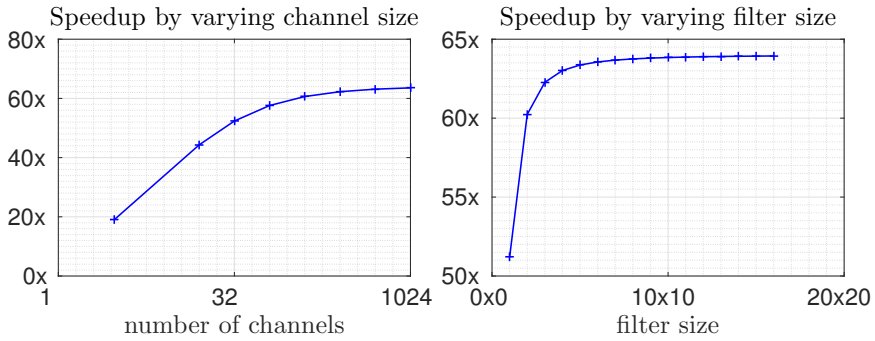


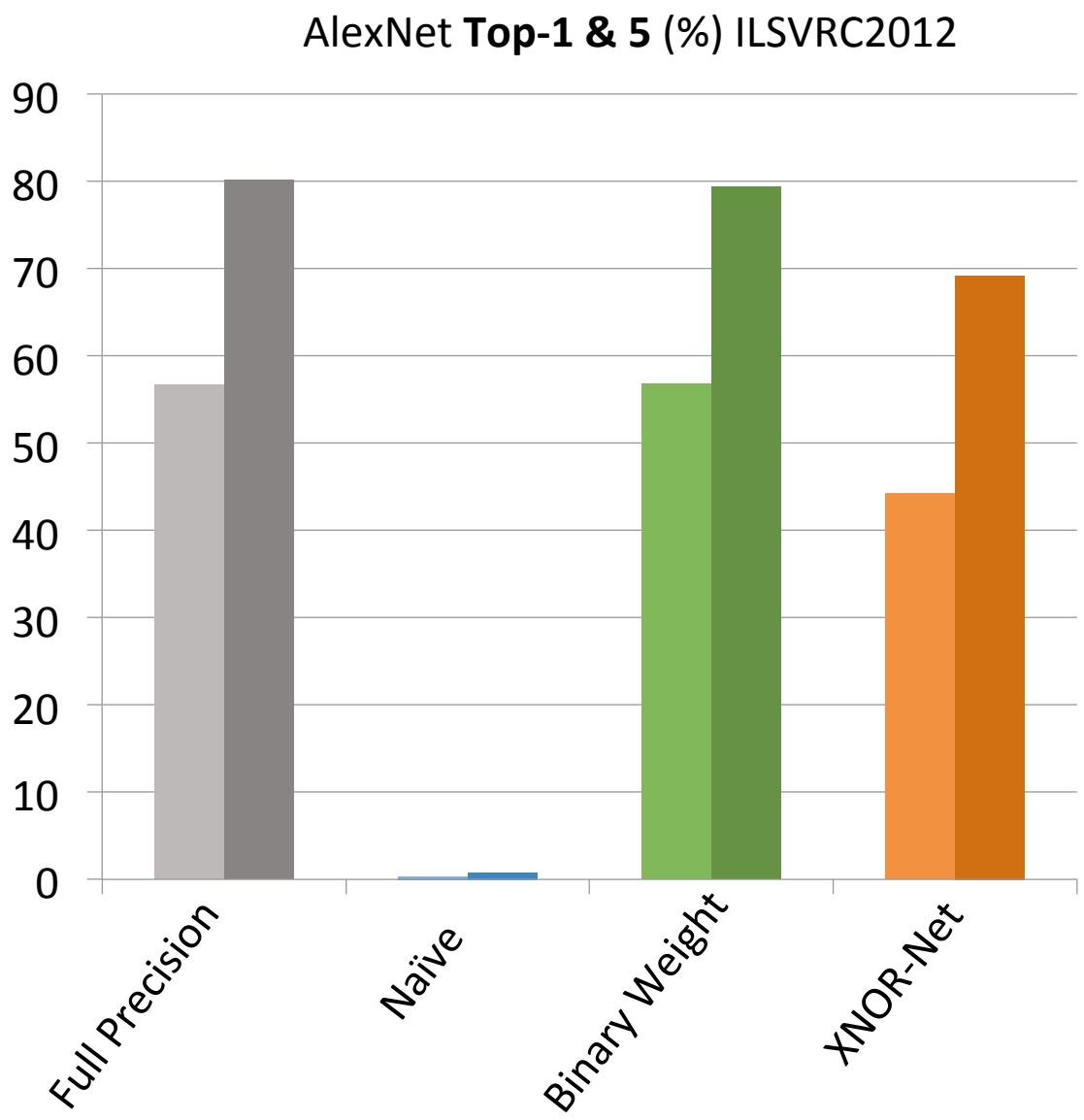


✓ 32x Smaller Model

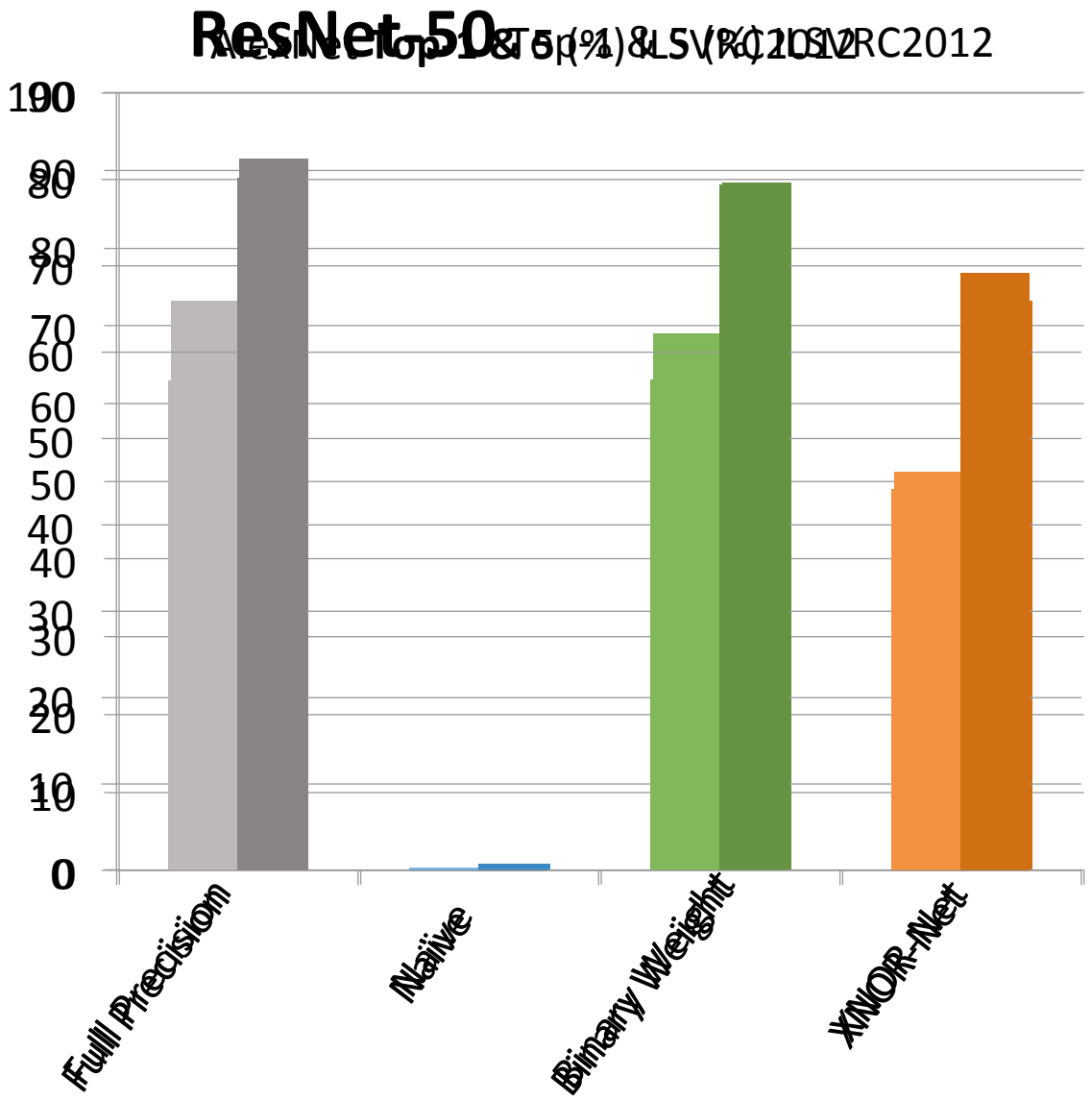


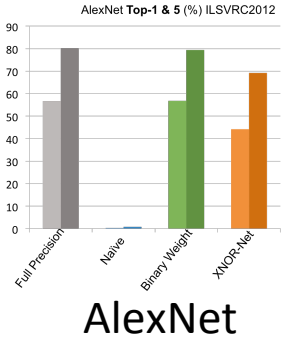
✓ 58x Less Computation




AlexNet **Top-1 & 5** (%) ILSVRC2012
90
80
70
60
50
40
30
20
10
0
Full Precision
Naive
Binary Weight
XNOR-Net

AlexNet


ResNet-50
AlexNet-Top1&Top5(%)&LSVRC2012
100
90
80
70
60
50
40
30
20
10
0
Full Precision
Naïve
Binary Weight
XNOR-Net

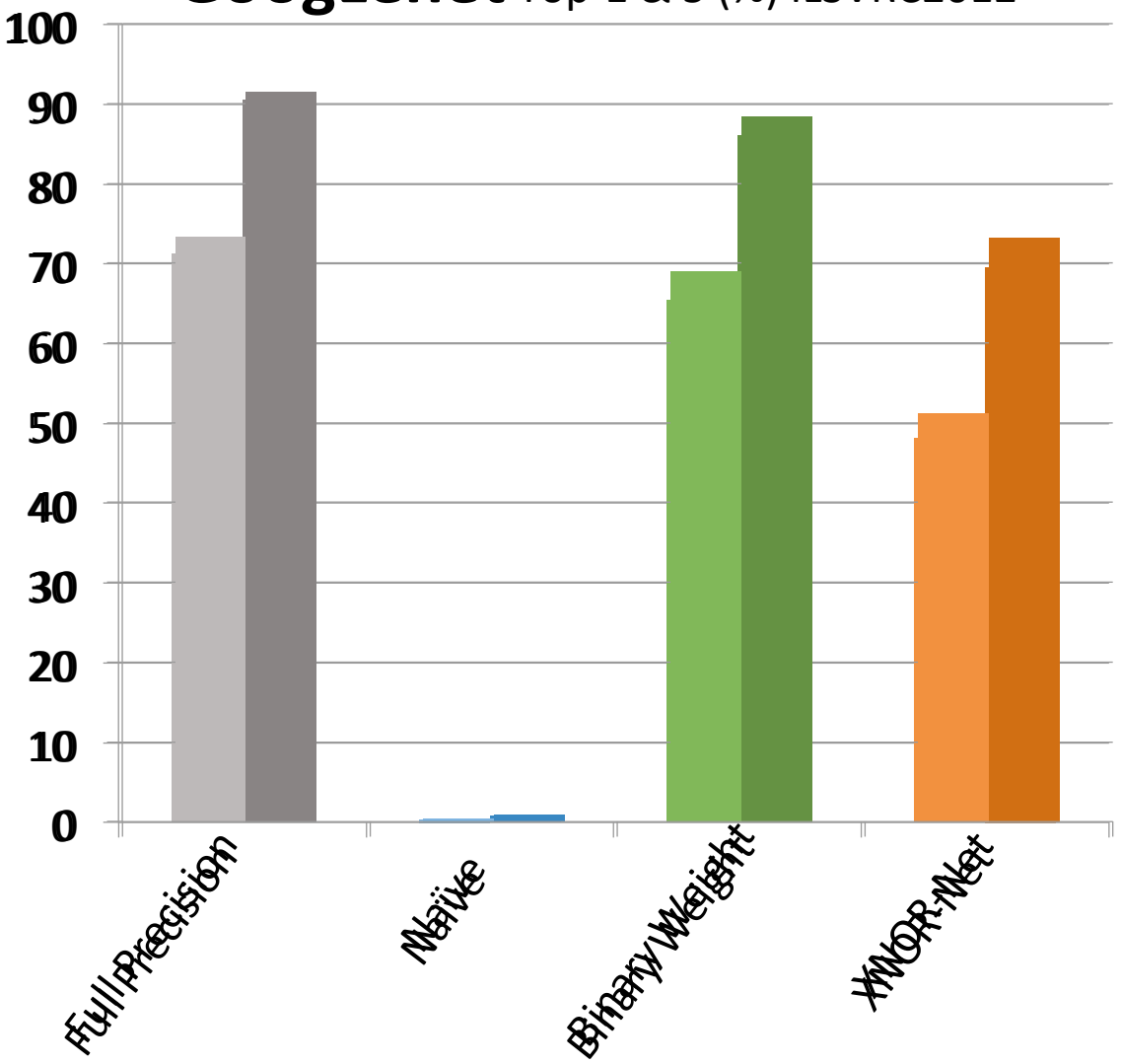


AlexNet

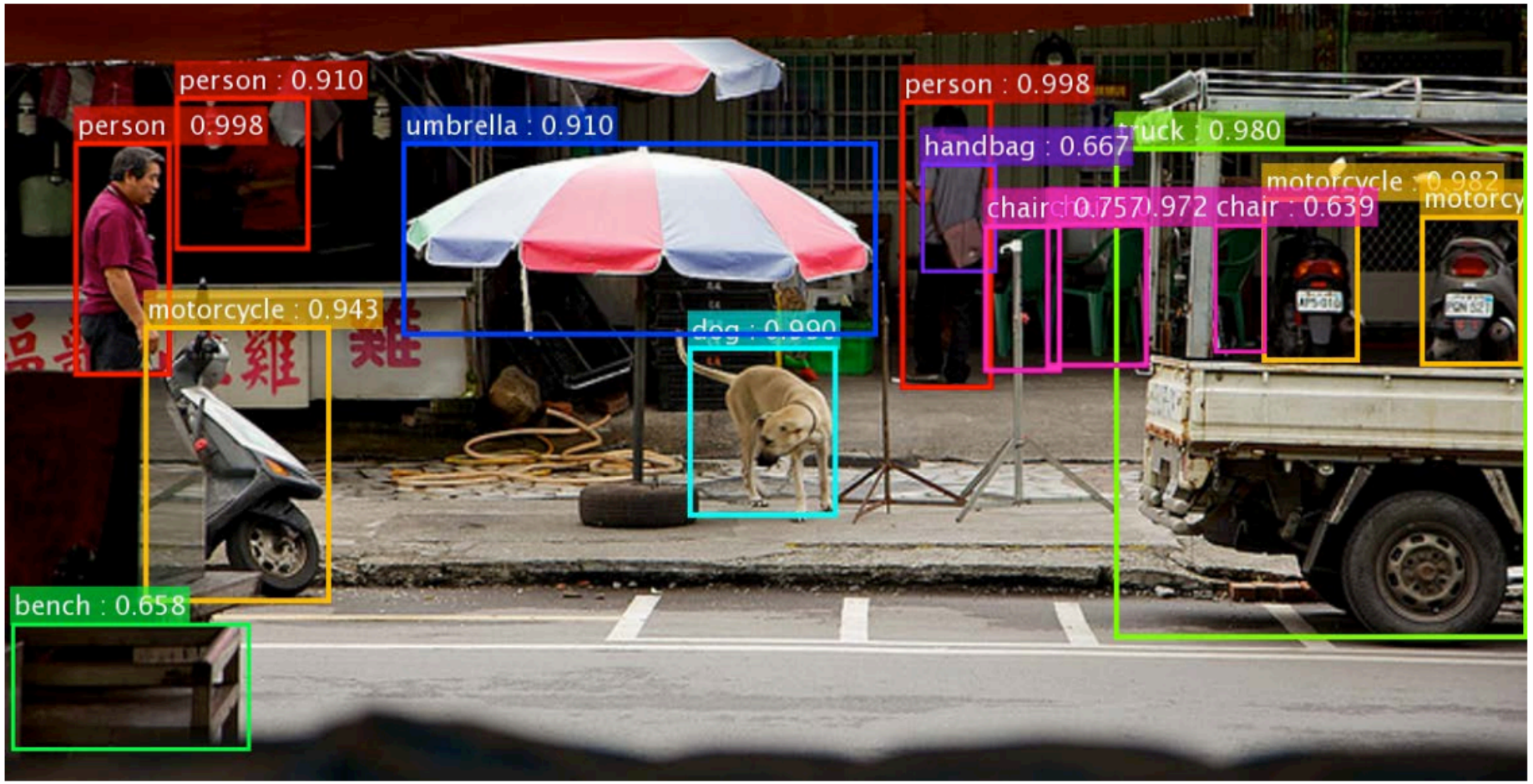
ResNet-50

GoogLeNet

Top-1 & 5 (%) ILSVRC2012



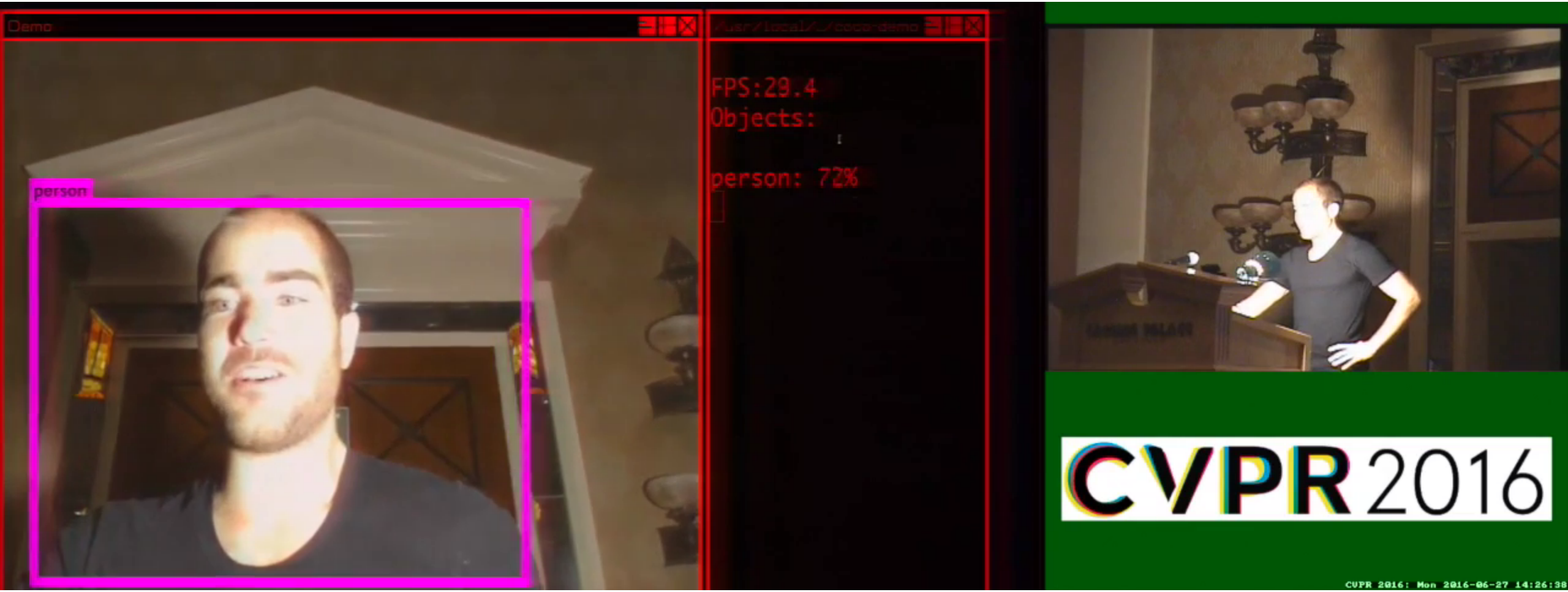
Object Detection



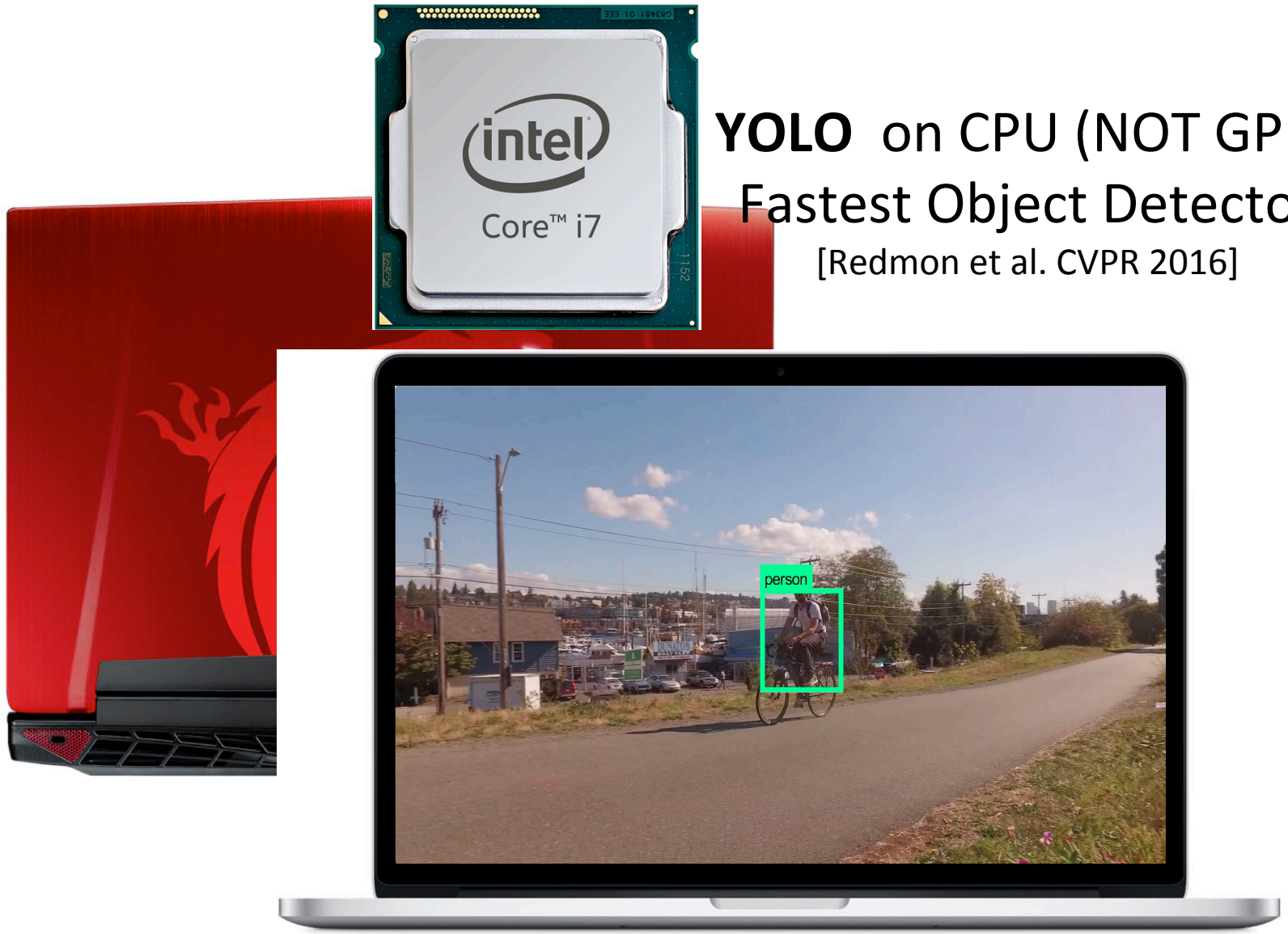
[He et al, 2015]

YOLO: Fastest Object Detector

[Redmon et al. CVPR 2016]




msi

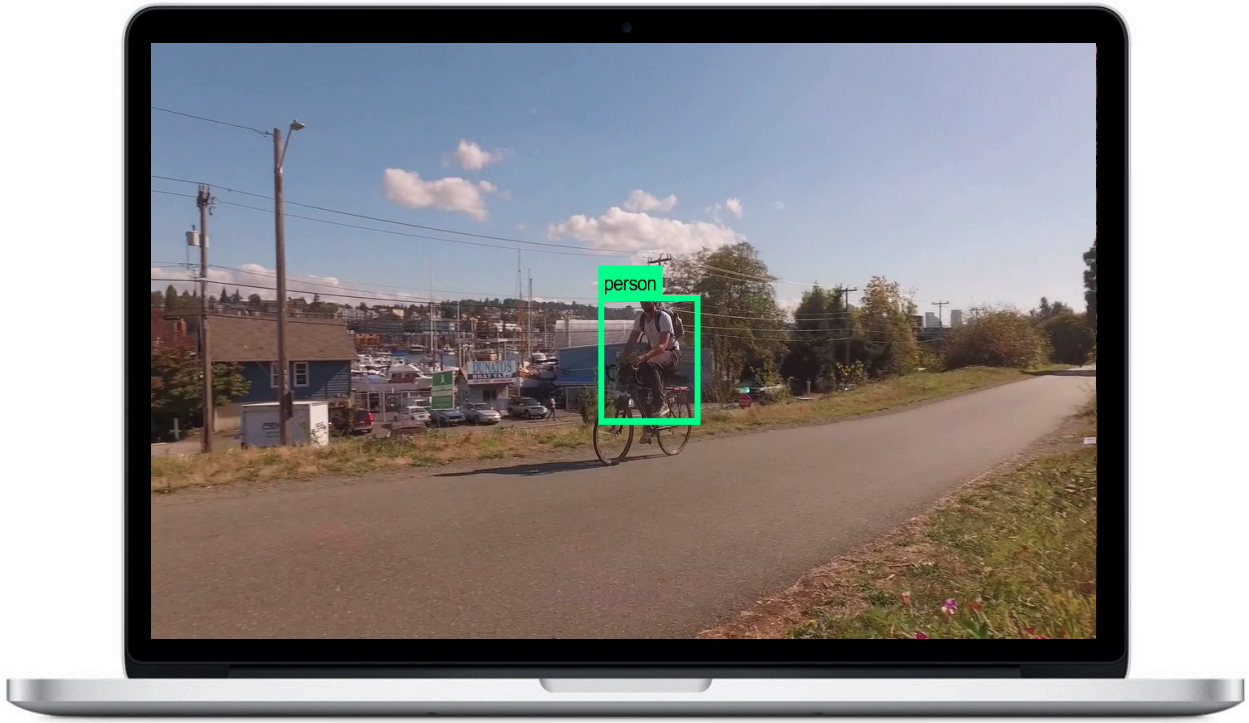


YOLO on CPU (NOT GPU)

Fastest Object Detector

[Redmon et al. CVPR 2016]

YOLO on CPU (NOT GPU)
Fastest Object Detector
[Redmon et al. CVPR 2016]

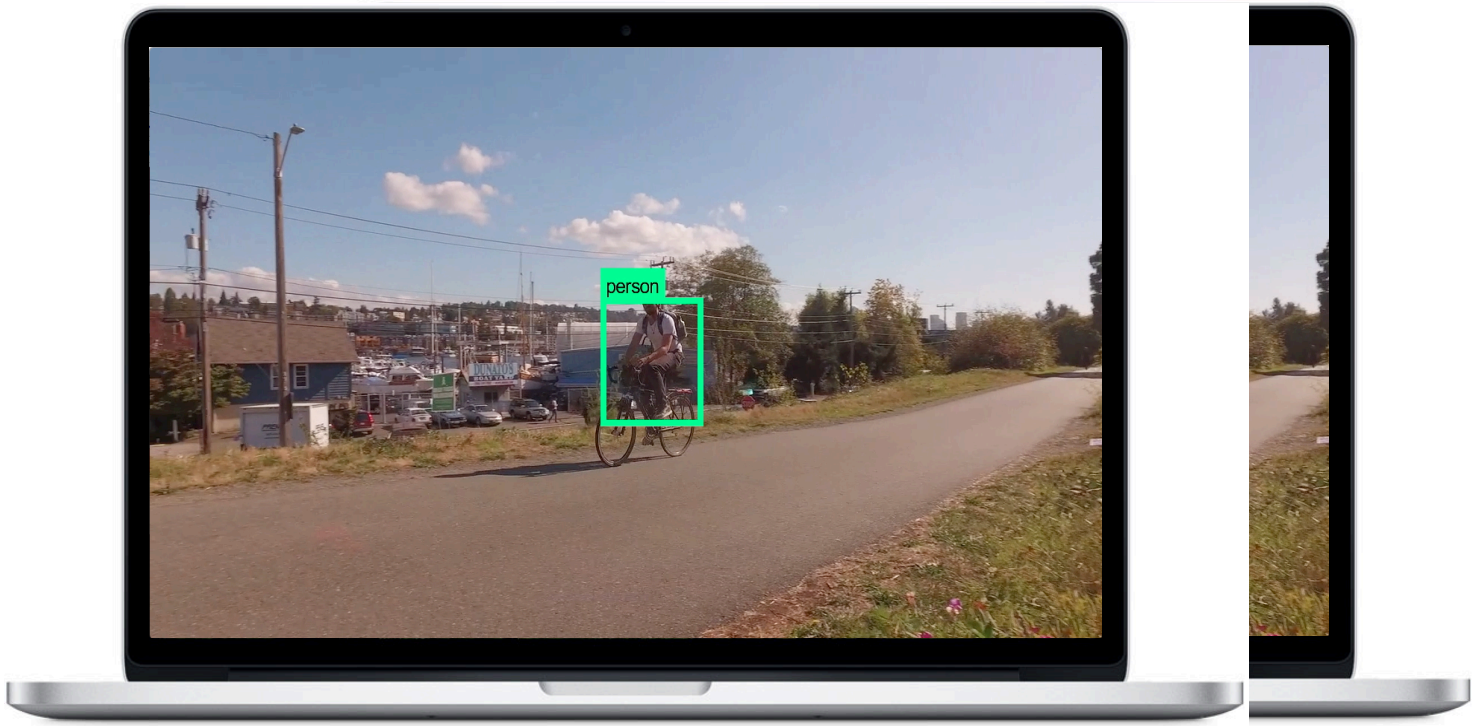


YOLO Method

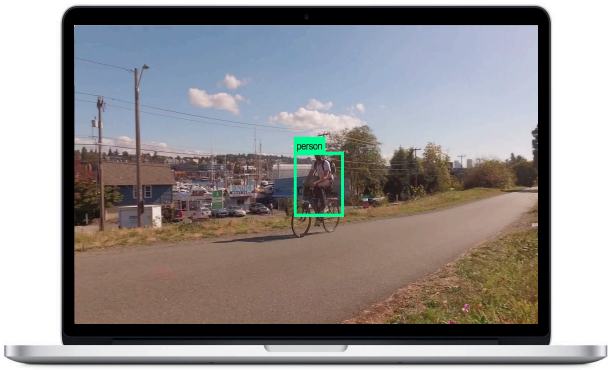
Real-Time on CPU(NOT GPU)

Fastest Object Detector

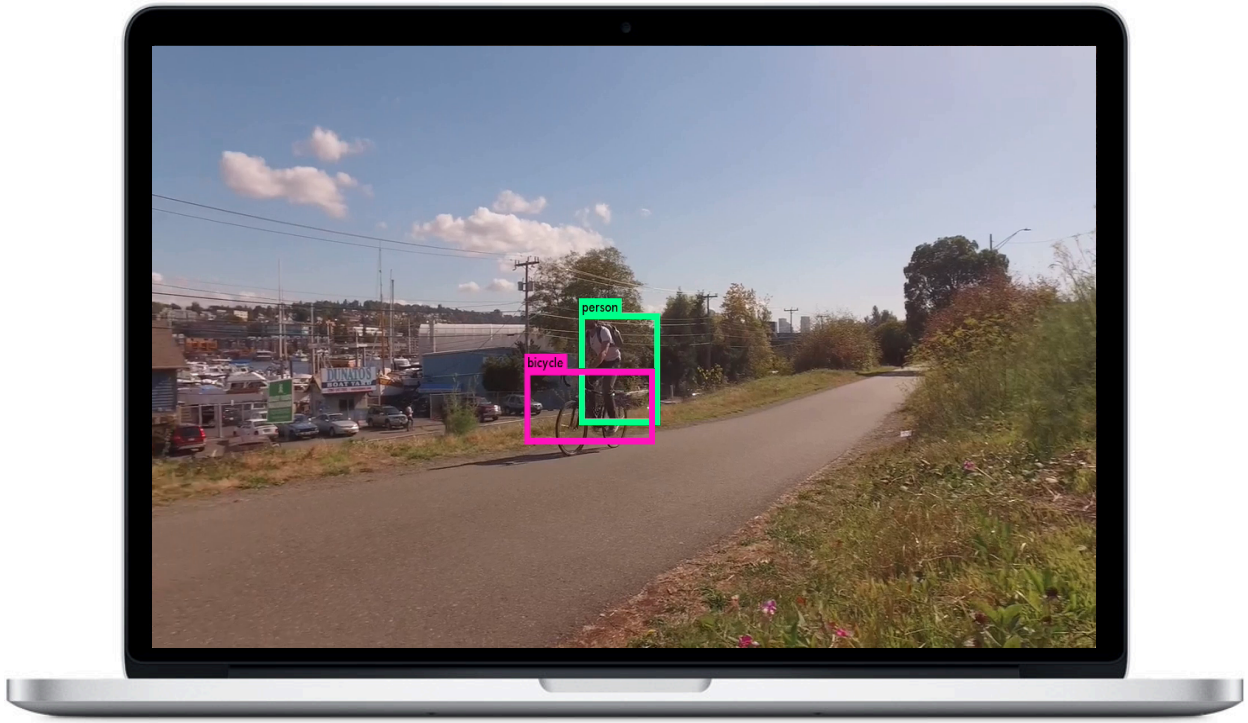
[Redmon et al. CVPR 2016]



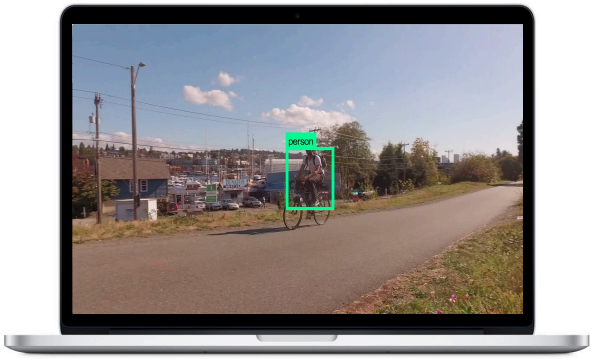
YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]



Our Method
On CPU (NOT GPU)

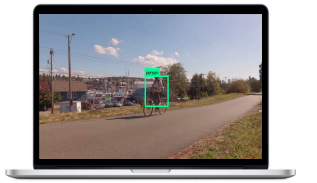


YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]

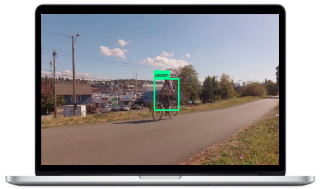


Our Method
On CPU (NOT GPU)

YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]



Our Method



Thank You !

Thank You !

GitHub

https://github.com/allenai/XNOR-Net

Carlo C. del Mundo

Dmitry Belenko