

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

Mohammad Rastegari

Vicente Ordonez

Joseph Redmon

Ali Farhadi









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

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

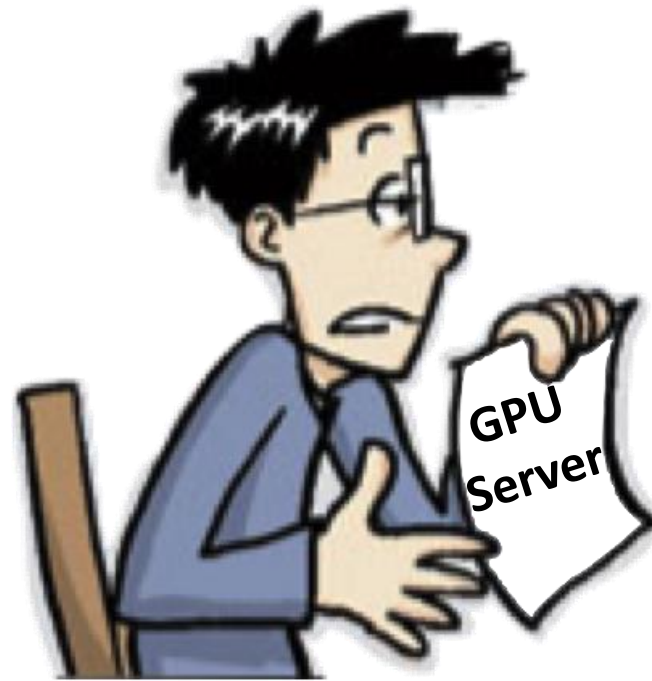


What should I learn to do well in computer vision research?



DEEP LEARNING

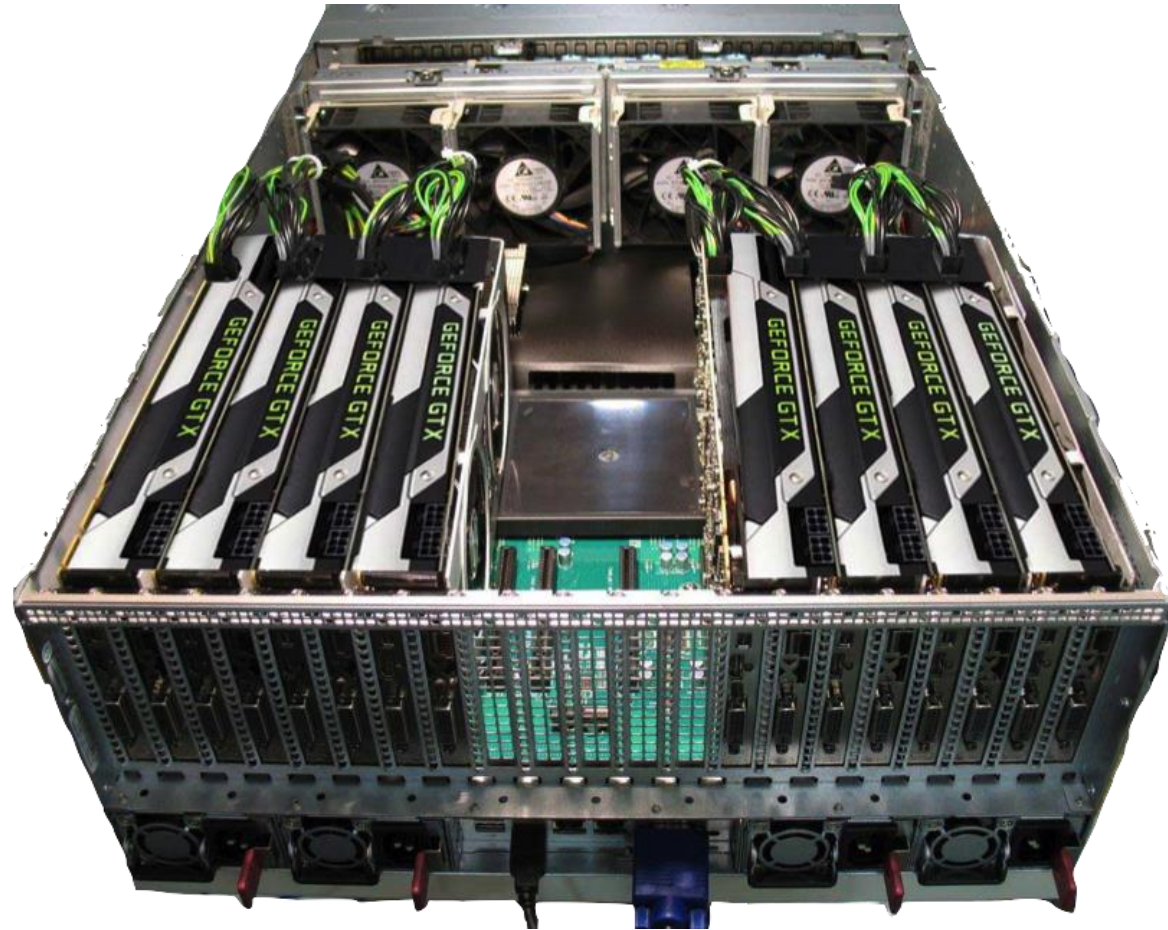




Ohhh No!!!

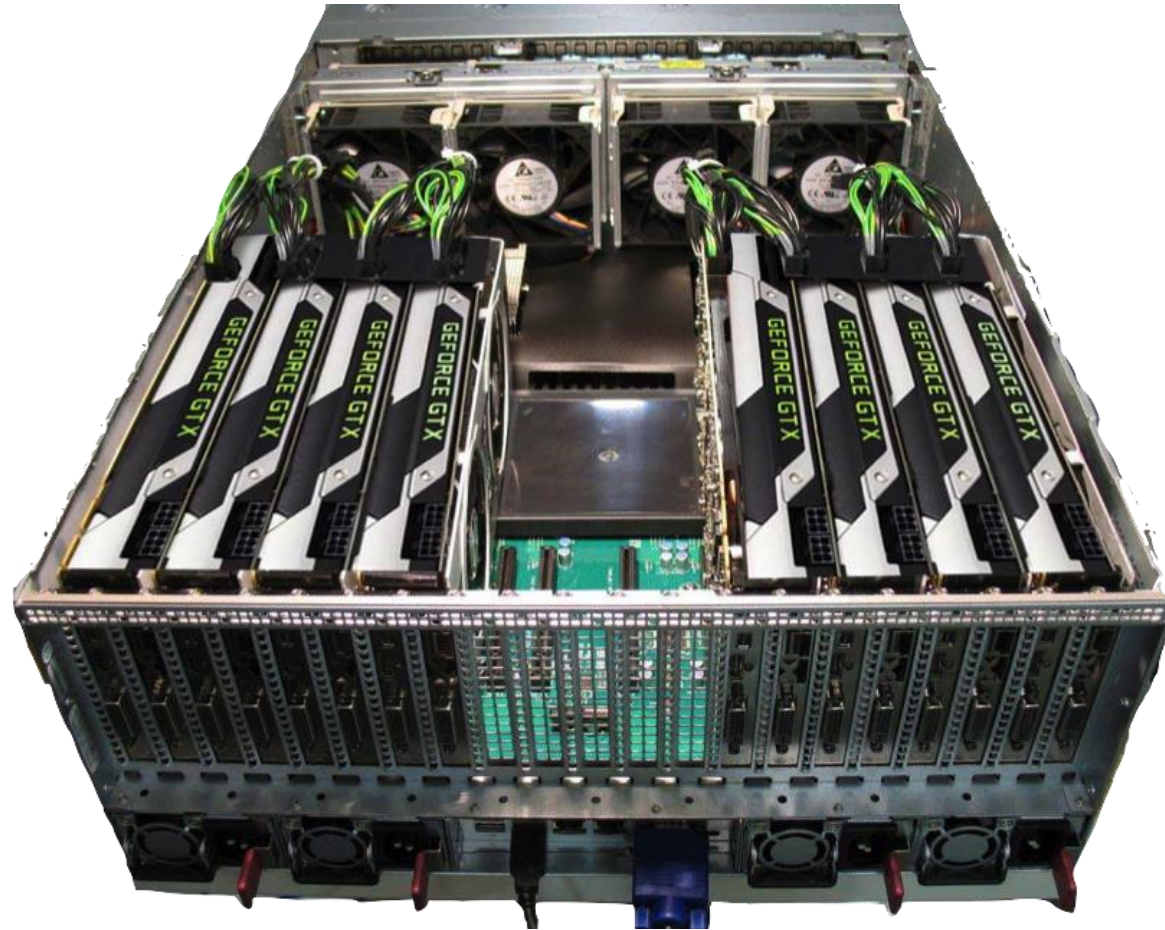


State of the art recognition methods



State of the art recognition methods

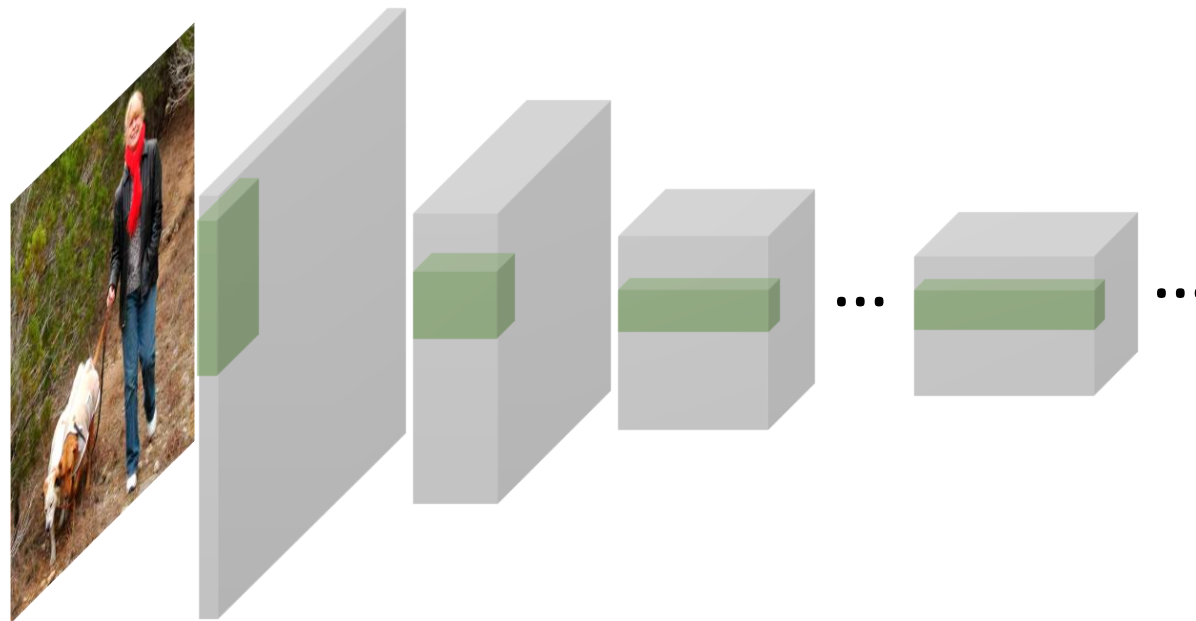
- Very Expensive
 - Memory
 - Computation
 - Power



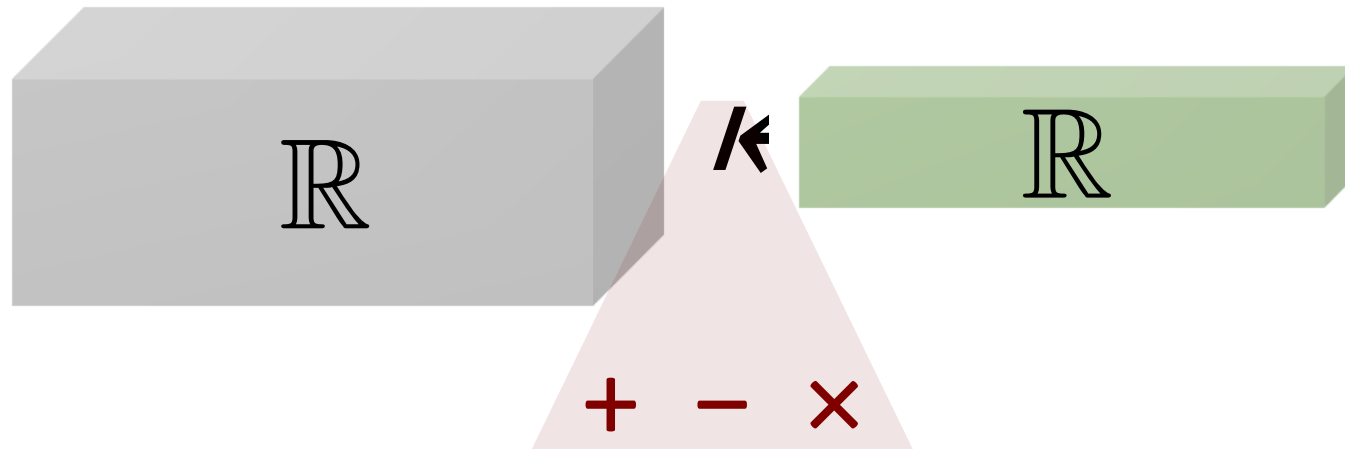




Convolutional Neural Networks







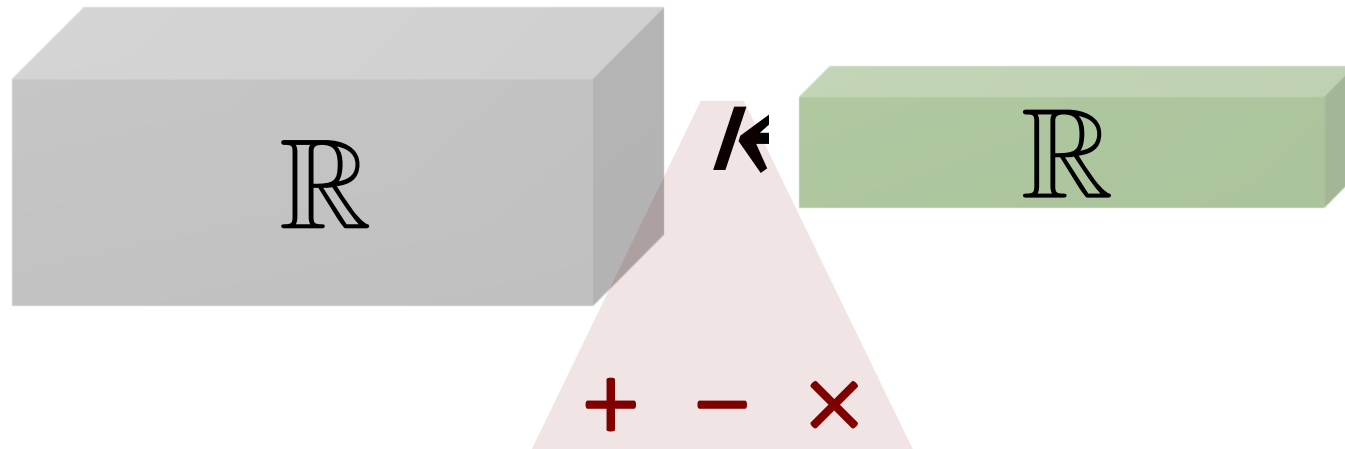
Number of Operations :

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

Inference time on CPU :

- AlexNet → ~3 fps
- VGG → ~0.25 fps

GPU !



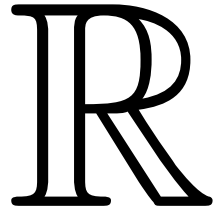
Number of Operations :

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

Inference time on CPU :

- AlexNet → ~3 fps
- VGG → ~0.25 fps

Lower Precision

A large, stylized, black-outlined letter 'R' in a serif font, positioned centrally on the page.

32-bit

Lower Precision

Reducing Precision

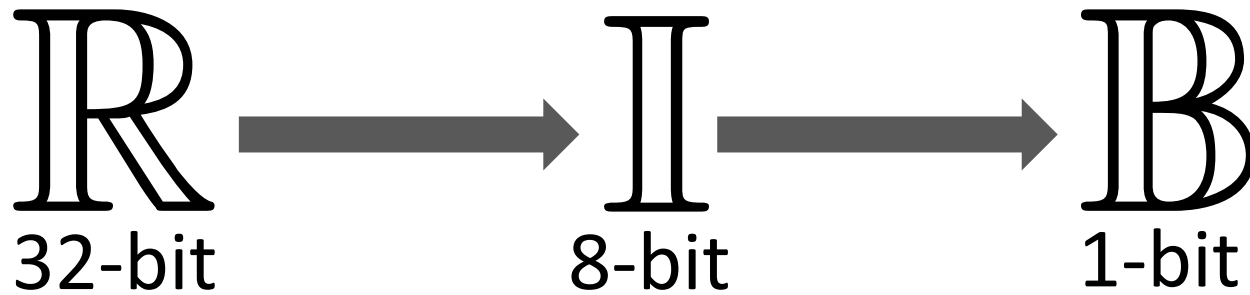
- Saving Memory
- Saving Computation



Lower Precision

Reducing Precision

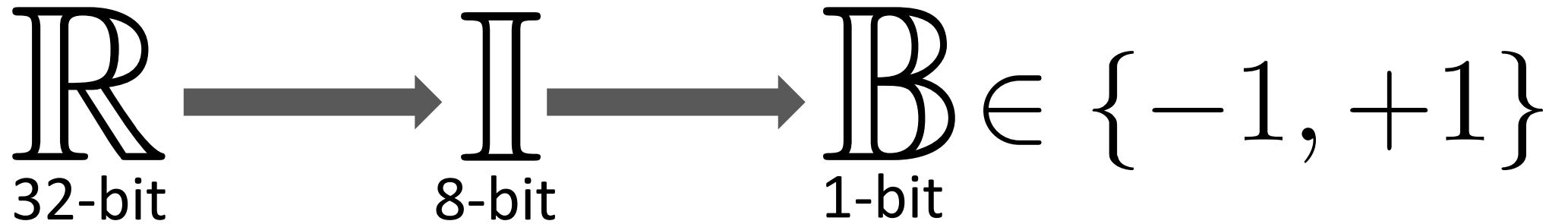
- Saving Memory
- Saving Computation



Lower Precision

Reducing Precision

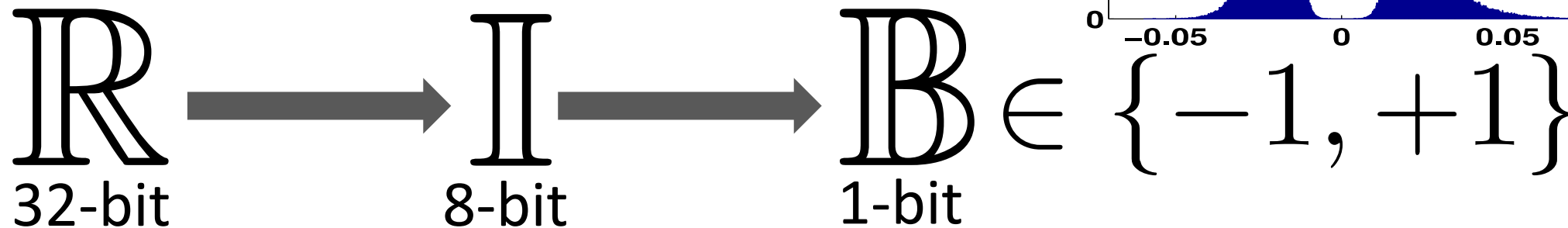
- Saving Memory
- Saving Computation



Lower Precision

Reducing Precision

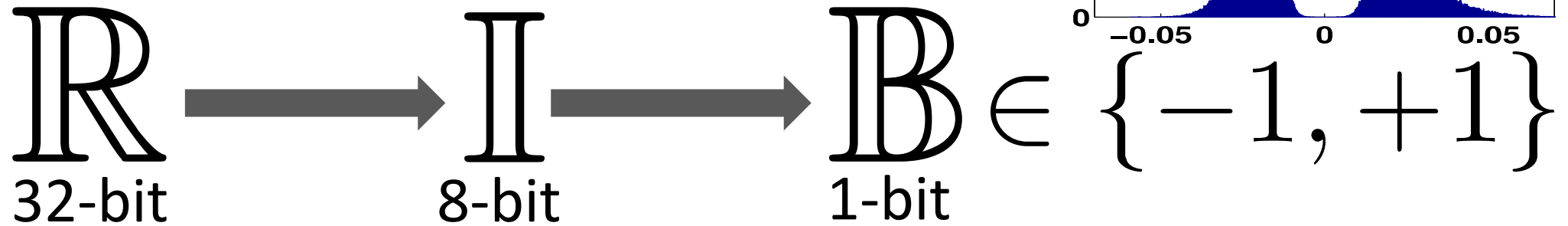
- Saving Memory
- Saving Computation



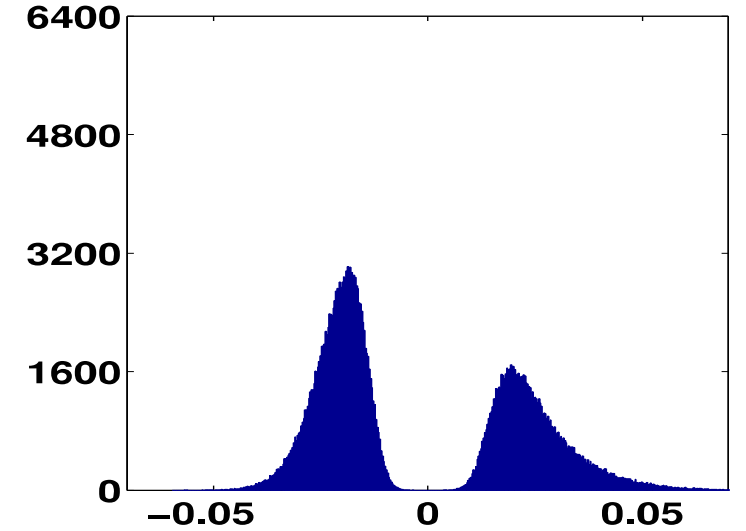
Lower Precision

Reducing Precision

- Saving Memory
- Saving Computation



[Han et al. 2016]

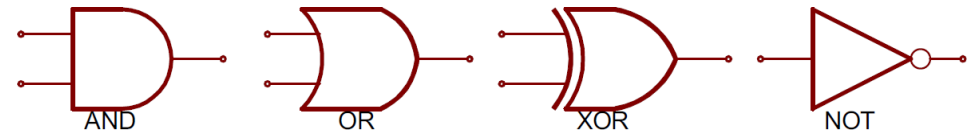


$\{-1,+1\}$	$\{0,1\}$
MUL	XNOR
ADD, SUB	Bit-Count (popcount)

Why Binary?

- Binary Instructions

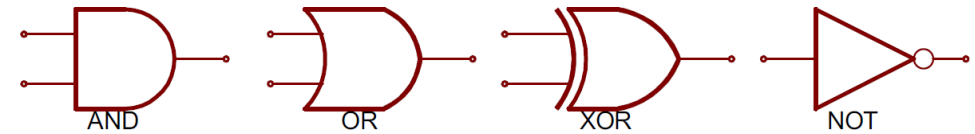
- AND, OR, XOR, XNOR, PoPCount (Bit-Count)



Why Binary?

- Binary Instructions

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



- Low Power Device





k



Operations

Memory

Computation

\mathbb{R}

k

\mathbb{R}

+ - ×

1x

1x

 k 

Operations

Memory

Computation

 \mathbb{R} k \mathbb{R}

+ - ×

1x

1x

 \mathbb{R} k \mathbb{B}

+ -

~32x

~2x

 k 

Operations

Memory

Computation

 \mathbb{R} k \mathbb{R}

+ - ×

1x

1x

 \mathbb{R} k \mathbb{B}

+ -

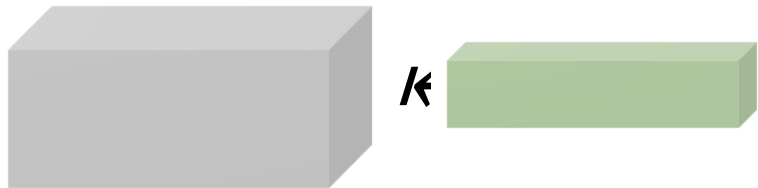
~32x

~2x

 \mathbb{B} k \mathbb{B} XNOR
Bit-count

~32x

~58x



	Operations	Memory	Computation
$\mathbb{R} \quad \leftarrow \quad \mathbb{R}$	+ - ×	1x	1x
<div style="border: 2px solid green; padding: 5px; display: inline-block;"> $\mathbb{R} \quad \leftarrow \quad \mathbb{B}$ Binary Weight Networks </div>	+ -	~32x	~2x
<div style="border: 2px solid brown; padding: 5px; display: inline-block;"> $\mathbb{B} \quad \leftarrow \quad \mathbb{B}$ XNOR-Networks </div>	XNOR Bit-count	~32x	~58x

 k 

Operations

Memory

Computation

 \mathbb{R} k \mathbb{R}

+ - ×

1x

1x

 \mathbb{R} k \mathbb{B}

+ -

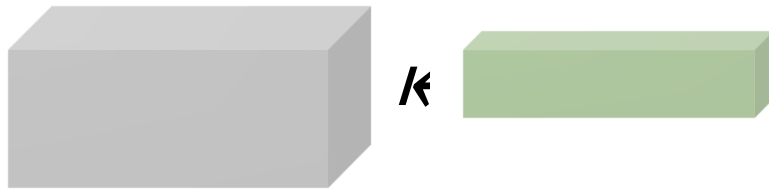
~32x

~2x

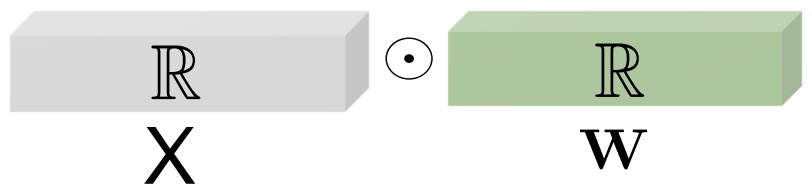
 \mathbb{B} k \mathbb{B} XNOR
Bit-count

~32x

~58x





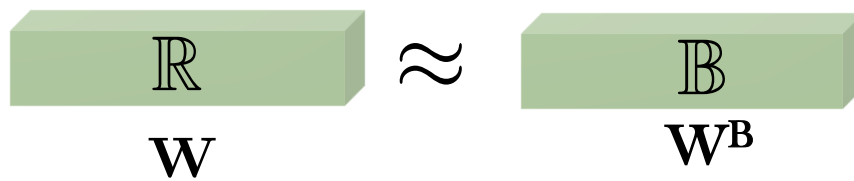


 \mathbb{R} X  \mathbb{R} W \approx  \mathbb{R} X  \mathbb{B} $W^{\mathbb{B}}$



$$\begin{array}{c} \text{R} \\ \text{X} \end{array} \odot \begin{array}{c} \text{R} \\ \text{W} \end{array} \approx \begin{array}{c} \text{R} \\ \text{X} \end{array} \odot \begin{array}{c} \text{B} \\ \text{W}^{\text{B}} \end{array}$$

$$\begin{array}{c} \text{R} \\ \text{W} \end{array} \approx \begin{array}{c} \text{B} \\ \text{W}^{\text{B}} \end{array}$$



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

Quantization Error

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

$$\left\| \begin{array}{c} \mathbf{W} \\ \mathbb{R} \end{array} - \begin{array}{c} \mathbf{W}^B \\ \mathbb{B} \end{array} \right\| \approx 0.75$$

Optimal Scaling Factor

$$\frac{\mathbb{R}}{w} \approx \alpha \frac{\mathbb{B}}{w^{\mathbb{B}}}$$

Optimal Scaling Factor

$$\begin{array}{c} \mathbb{R} \\ \mathbf{W} \end{array} \approx \alpha \begin{array}{c} \mathbb{B} \\ \mathbf{W}^{\mathbf{B}} \end{array}$$

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

Optimal Scaling Factor

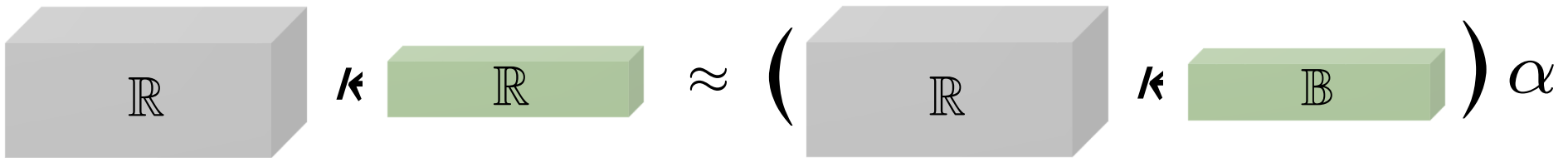
$$\begin{array}{c} \mathbb{R} \\ \mathbf{W} \end{array} \approx \alpha \begin{array}{c} \mathbb{B} \\ \mathbf{W}^{\mathbb{B}} \end{array}$$

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

$$\begin{array}{l} \mathbf{W}^{\mathbb{B}*} = \text{sign}(\mathbf{W}) \\ \alpha^* = \frac{1}{n} \|\mathbf{W}\|_{\ell_1} \end{array}$$

$$\mathbb{R} \kappa \mathbb{R} \approx \left(\mathbb{R} \kappa \mathbb{B} \right) \alpha$$

How to train a CNN with binary filters?

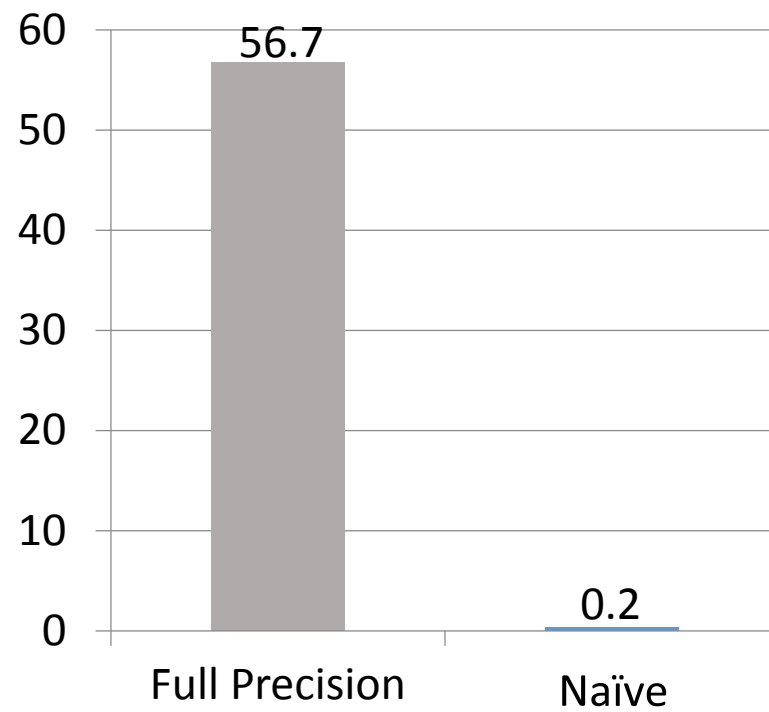


Training Binary Weight Networks

Naive Solution:

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

AlexNet Top-1 (%) ILSVRC2012



W



...



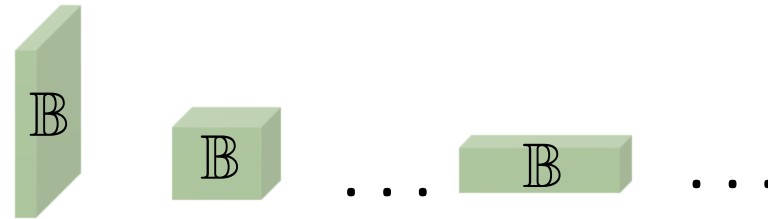
...

\mathbb{W}



Binarization

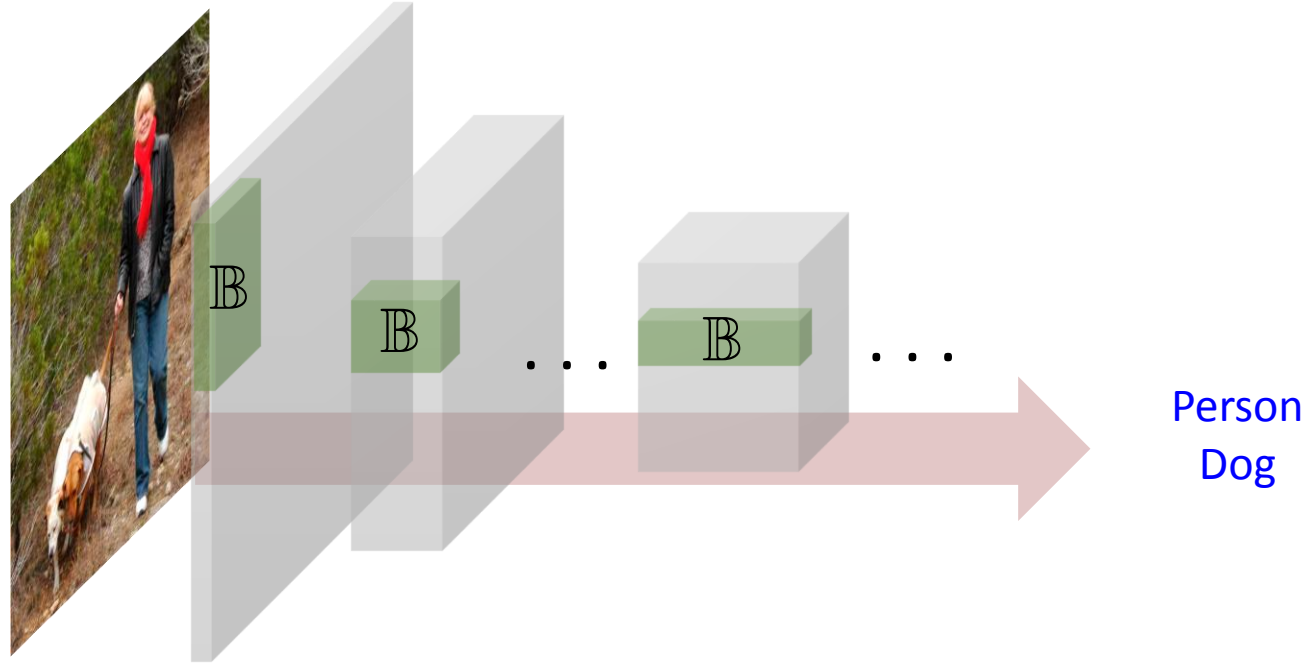
$\mathbb{W}^{\mathbb{B}}$



W



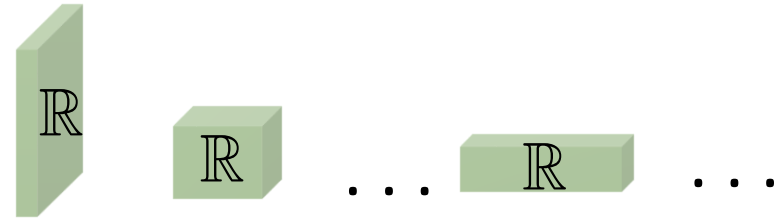
Binarization



Binary Weight Network

Train for binary weights:

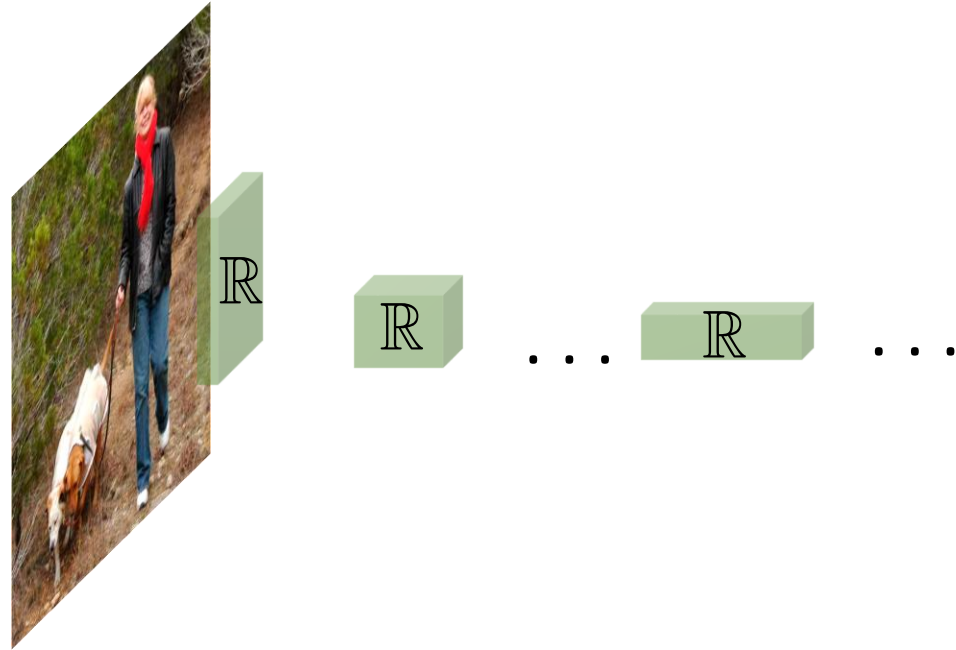
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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:

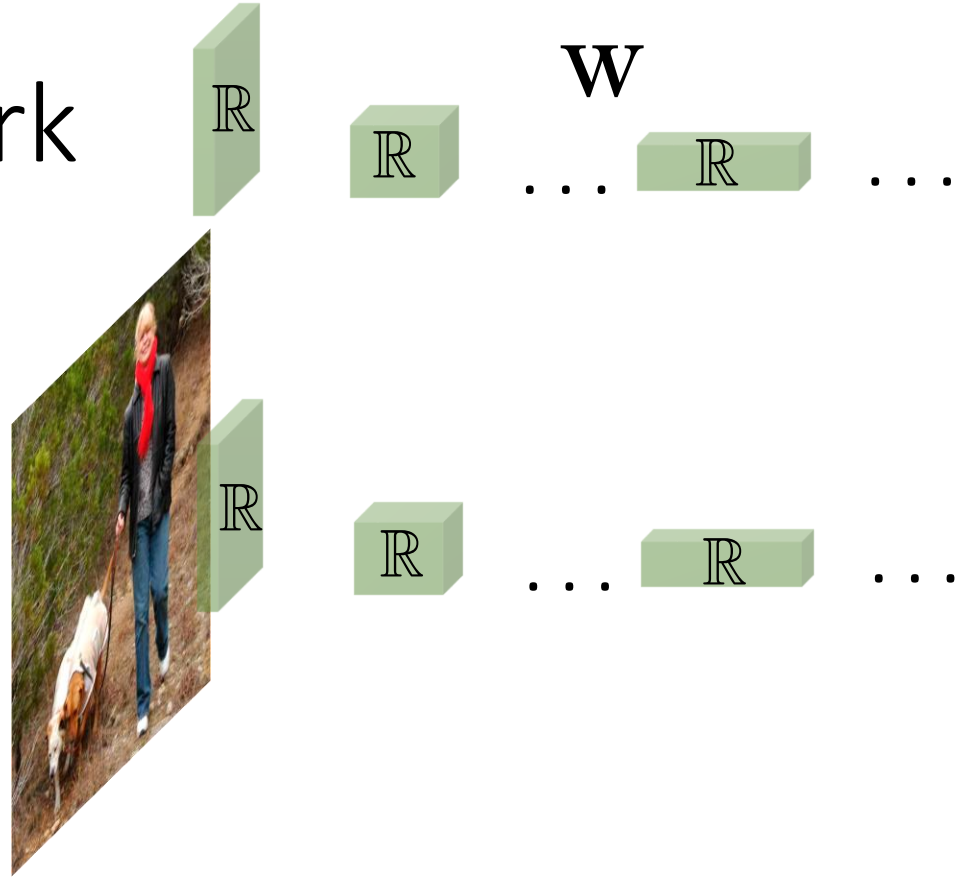
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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:

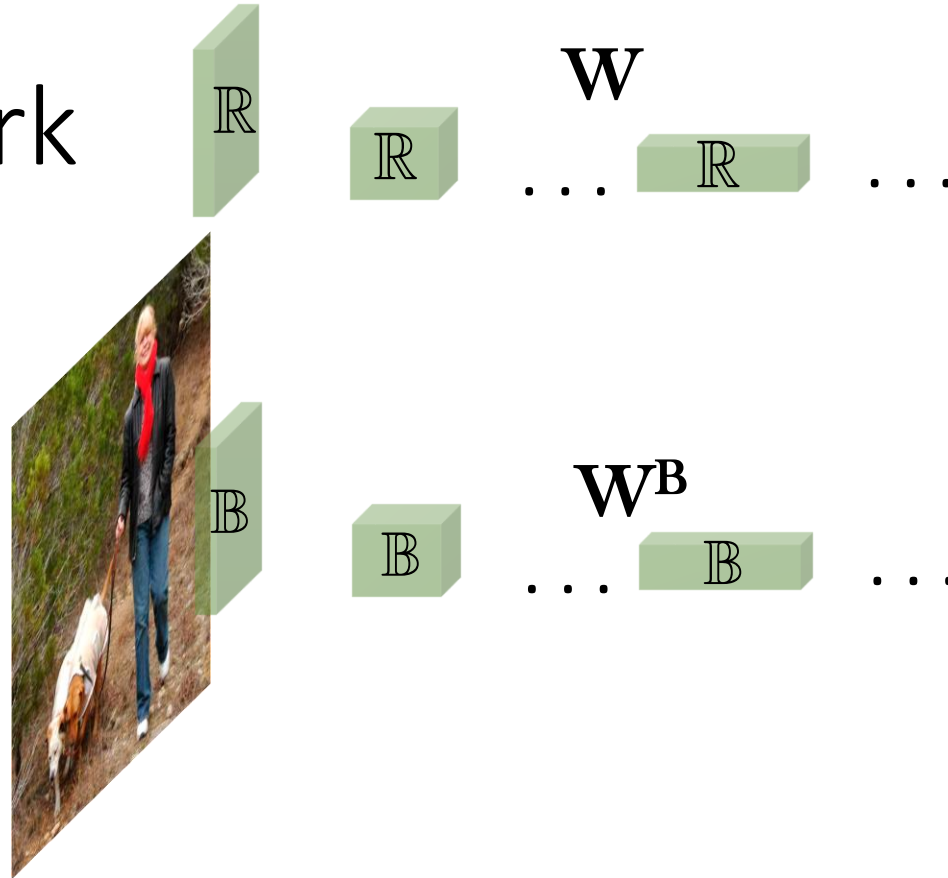
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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:

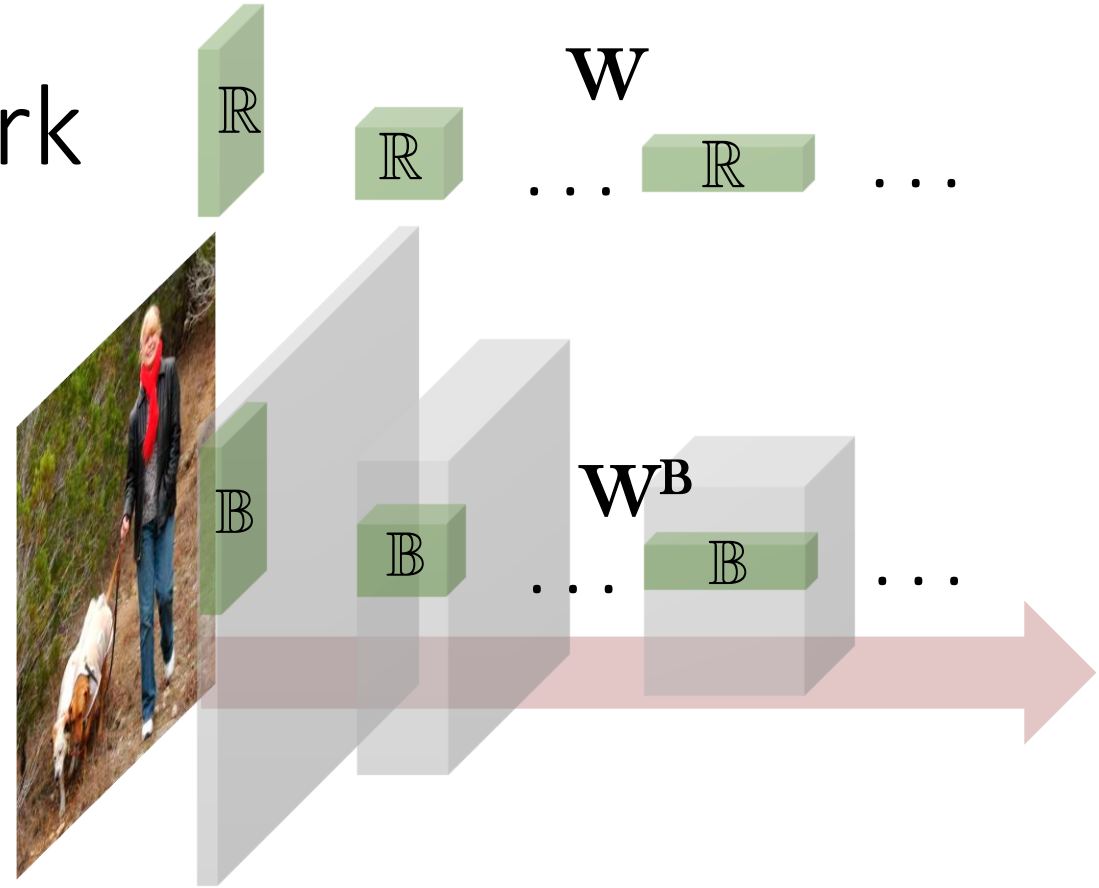
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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:

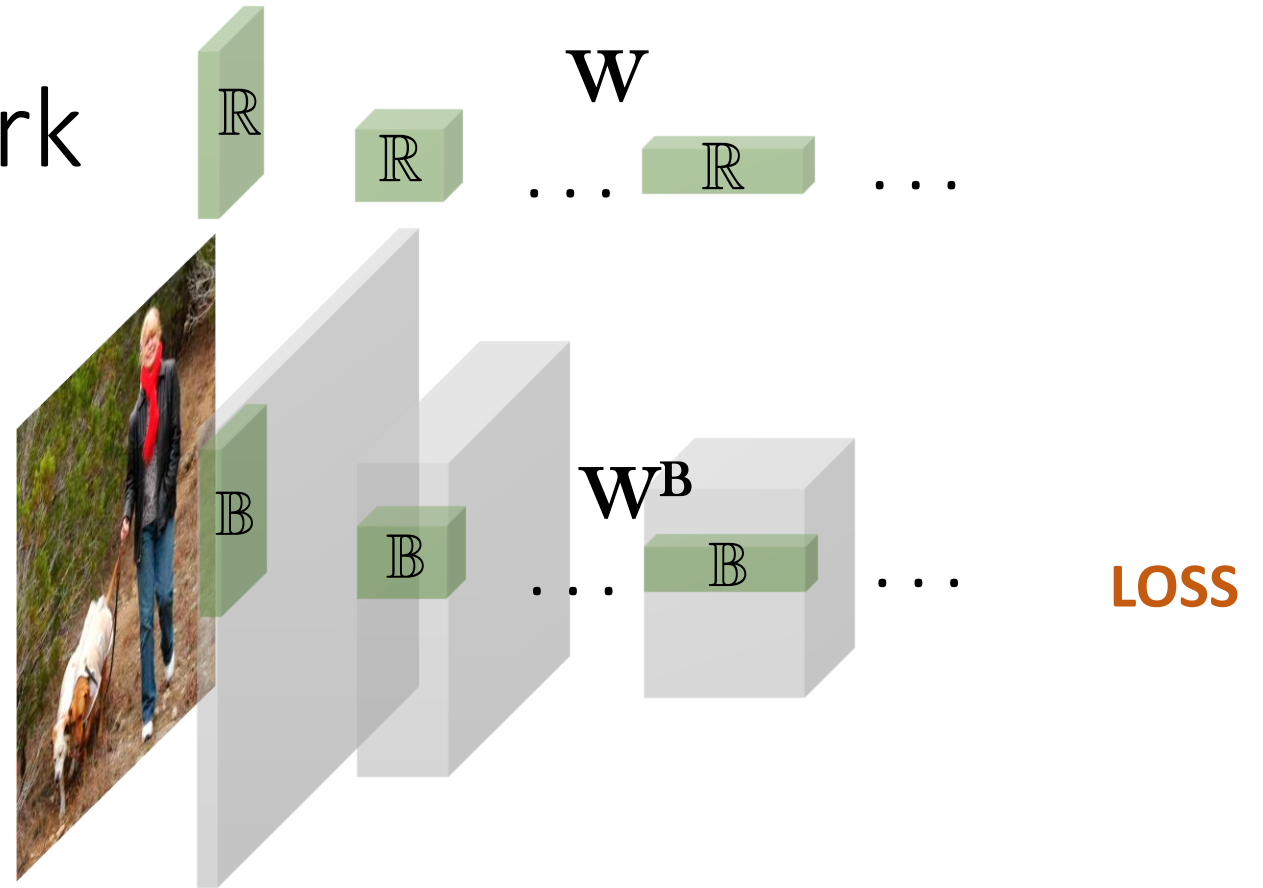
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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:

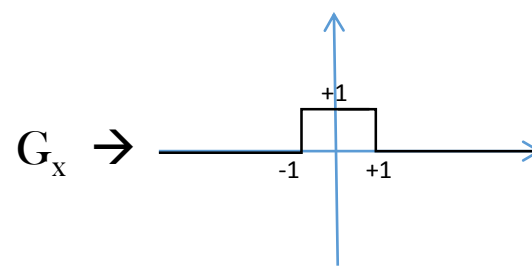
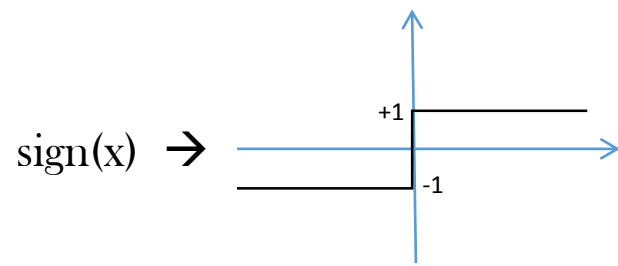
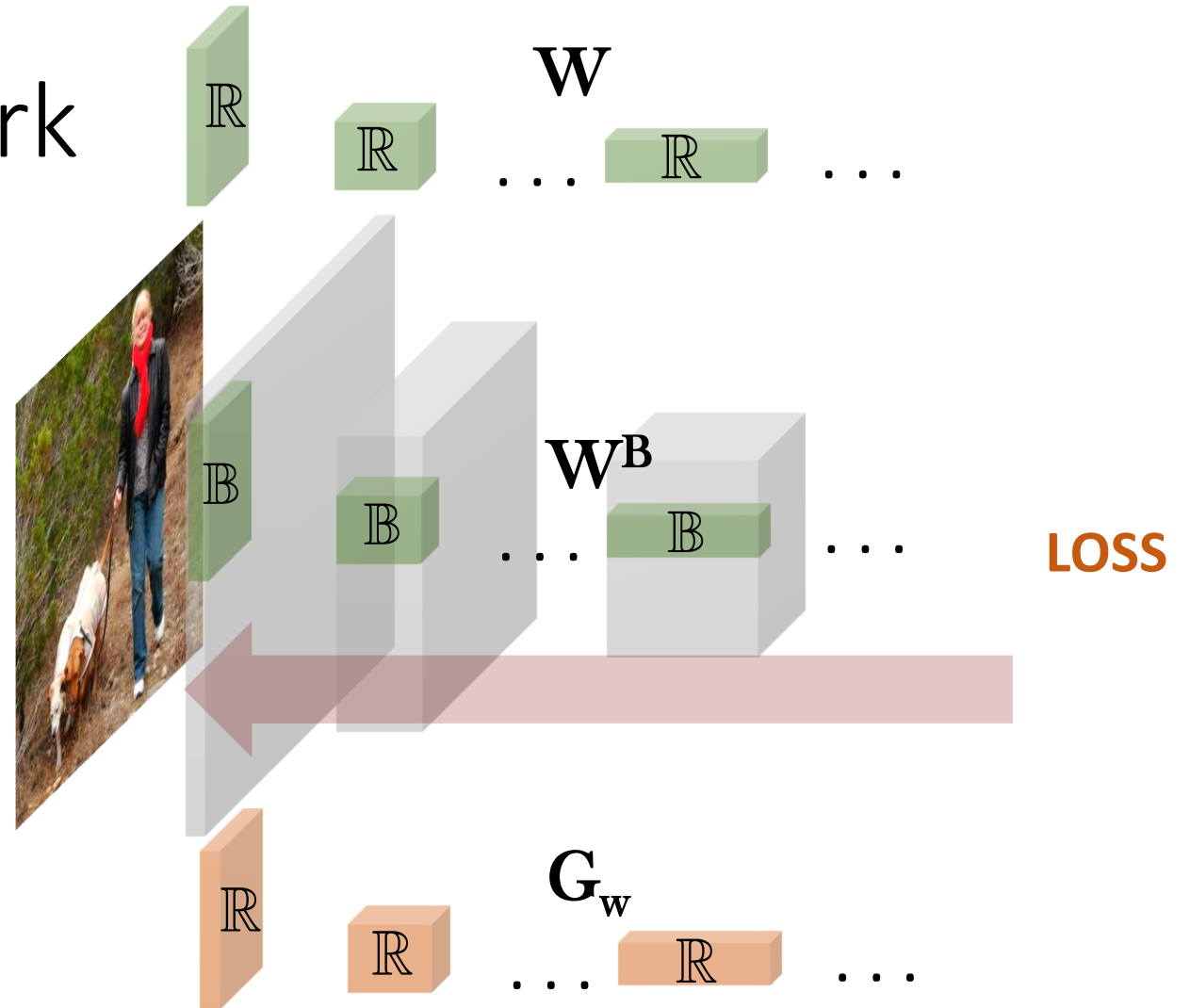
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \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. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} = \text{Backward pass with } \alpha, \mathbf{W}^B$
9. Update \mathbf{W} ($\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$)

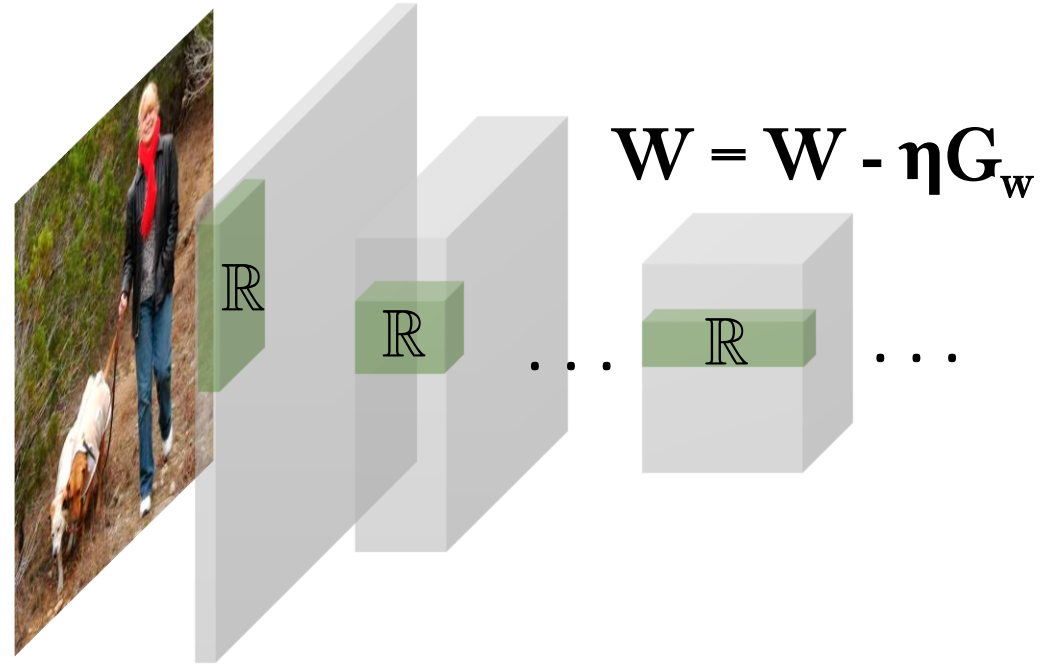


[Hinton et al. 2012]

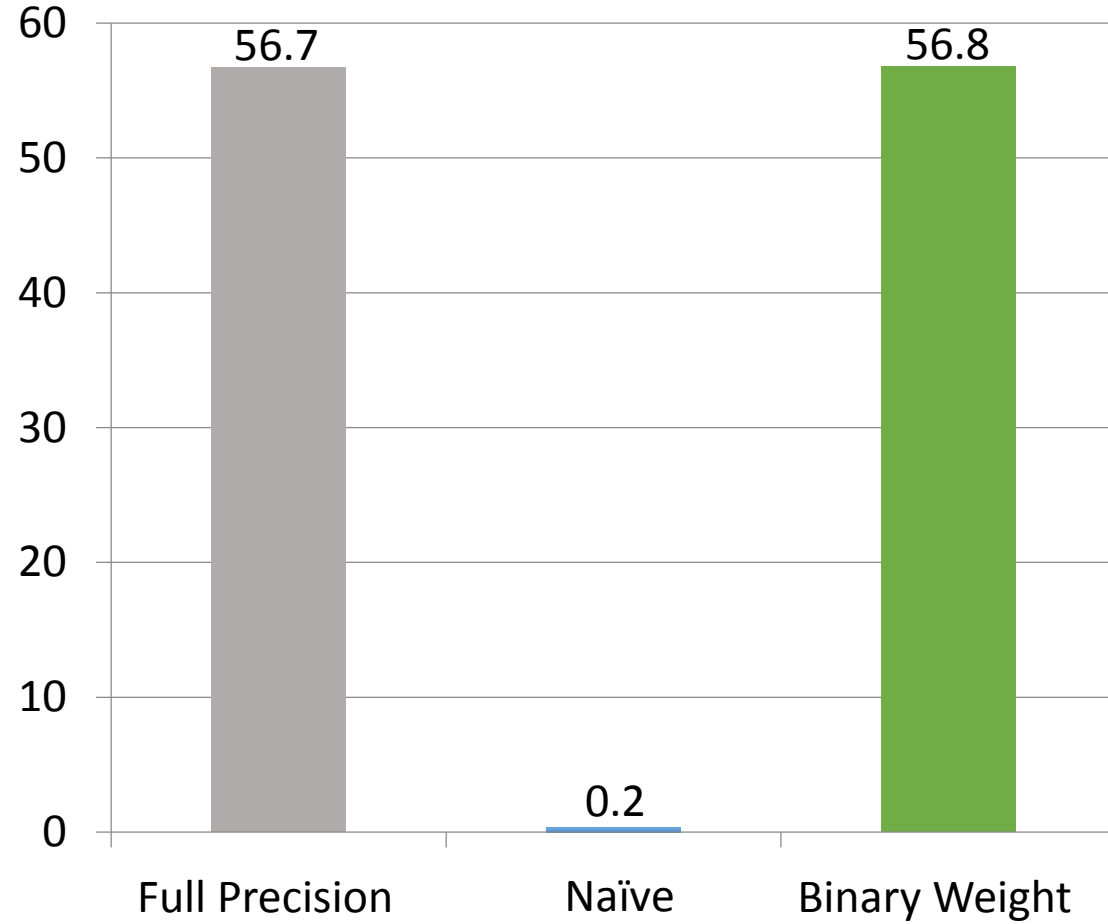
Binary Weight Network

Train for binary weights:

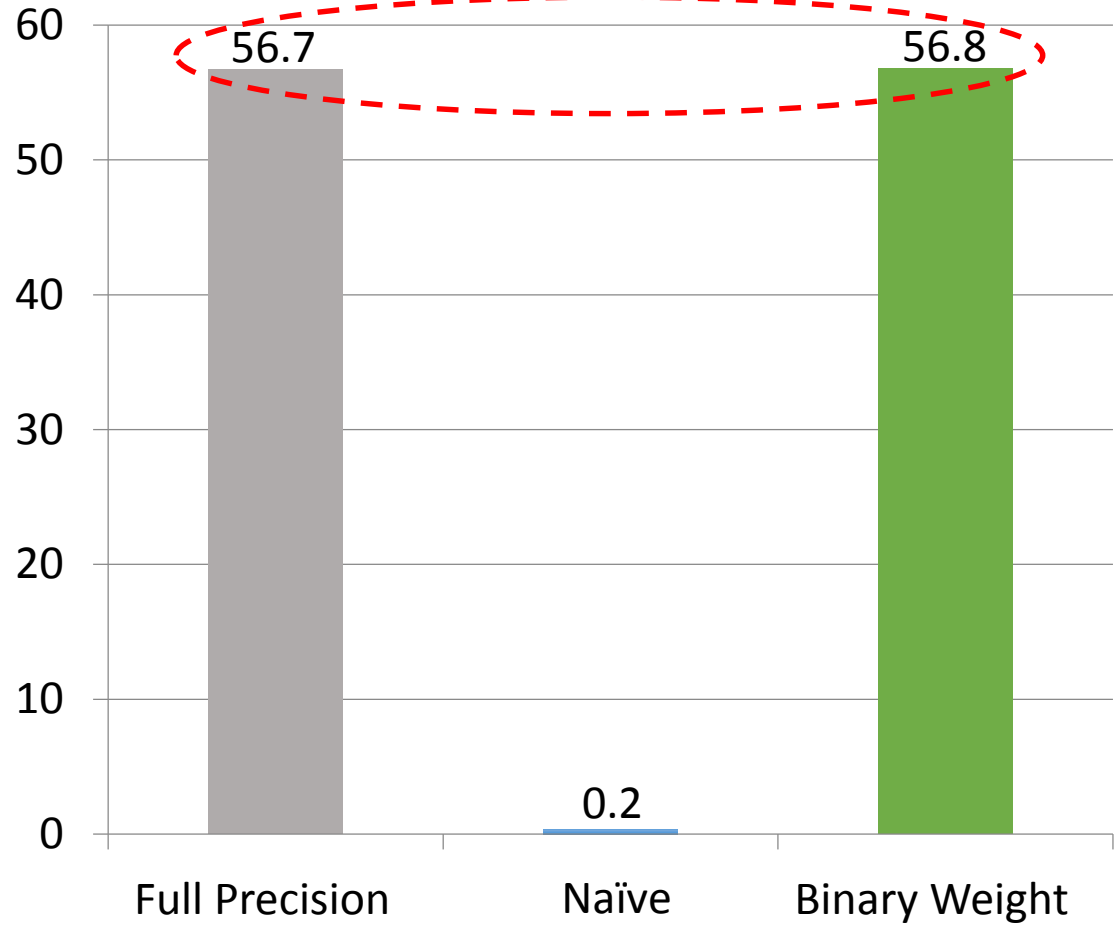
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \mathbf{W}^B
9. Update \mathbf{W} ($\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$)

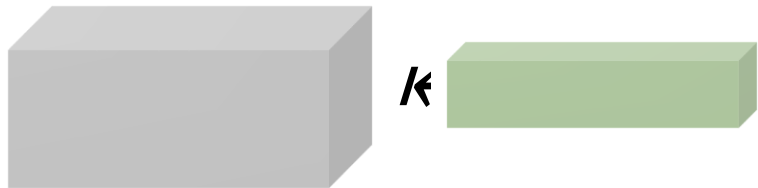


AlexNet Top-1 (%) ILSVRC2012



AlexNet Top-1 (%) ILSVRC2012





Operations

Memory

Computation

\mathbb{R}

k

\mathbb{R}

+ - ×

1x

1x

\mathbb{R}

k

\mathbb{B}

+ -

~32x

~2x

\mathbb{B}

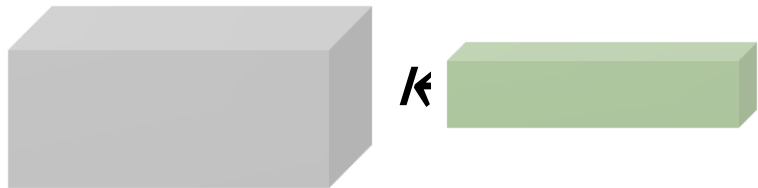
k

\mathbb{B}

XNOR
Bit-count

~32x

~58x



Operations

Memory

Computation

\mathbb{R}

k

\mathbb{R}

+ - ×

1x

1x

\mathbb{R}

k

\mathbb{B}

+ -

~32x

~2x

\mathbb{B}

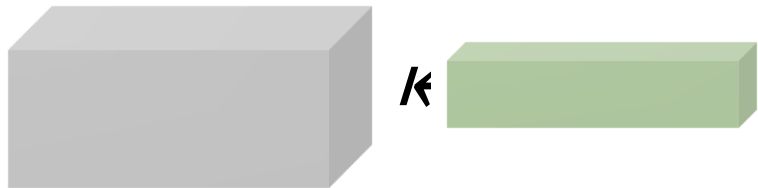
k

\mathbb{B}

XNOR
Bit-count

~32x

~58x



Operations

Memory

Computation

\mathbb{R}

k

\mathbb{R}

+ - ×

1x

1x

\mathbb{R}

k

\mathbb{B}

+ -

~32x

~2x

\mathbb{B}

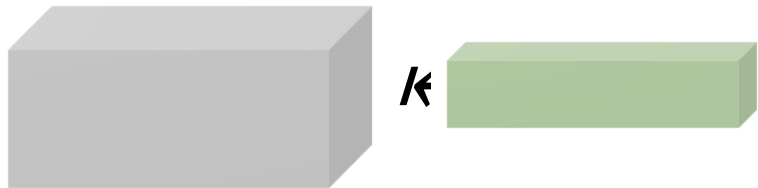
k

\mathbb{B}

XNOR
Bit-count

~32x

~58x



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

XNOR-Networks

Binary Input and Binary Weight (XNOR-Net)

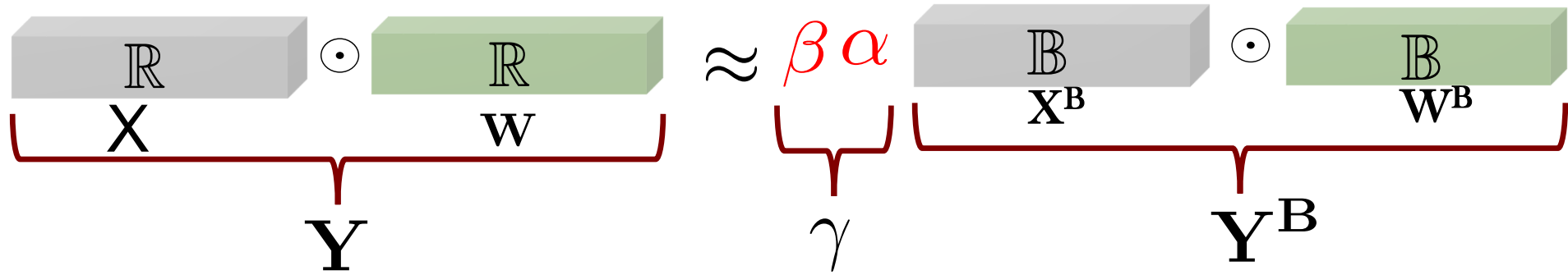


Binary Input and Binary Weight (XNOR-Net)

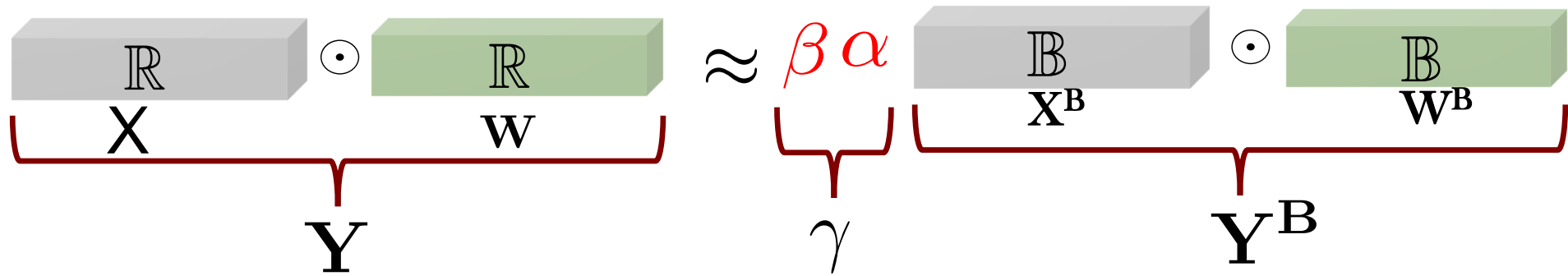
The diagram illustrates the approximation of a real-valued dot product with a binary dot product. On the left, a gray rectangular block labeled \mathbb{R} with X below it is followed by a green rectangular block labeled \mathbb{R} with W below it. A small circle with a dot inside is positioned between them. This is followed by an approximation symbol \approx and a red β . On the right, a gray rectangular block labeled \mathbb{B} with X^B below it is followed by a green rectangular block labeled \mathbb{B} with W^B below it. A small circle with a dot inside is positioned between them, followed by a red α .

$$\begin{matrix} \mathbb{R} \\ X \end{matrix} \odot \begin{matrix} \mathbb{R} \\ W \end{matrix} \approx \beta \begin{matrix} \mathbb{B} \\ X^B \end{matrix} \odot \alpha \begin{matrix} \mathbb{B} \\ W^B \end{matrix}$$

Binary Input and Binary Weight (XNOR-Net)



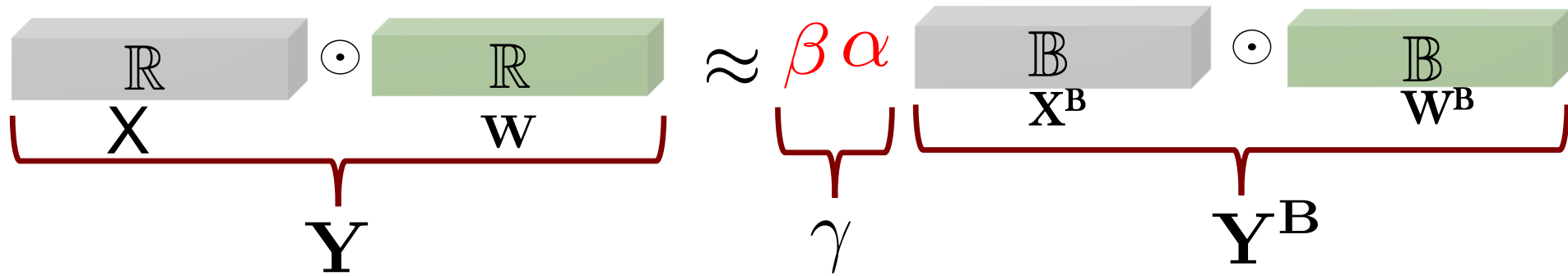
Binary Input and Binary Weight (XNOR-Net)



$$Y \approx \gamma Y^B$$

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

Binary Input and Binary Weight (XNOR-Net)

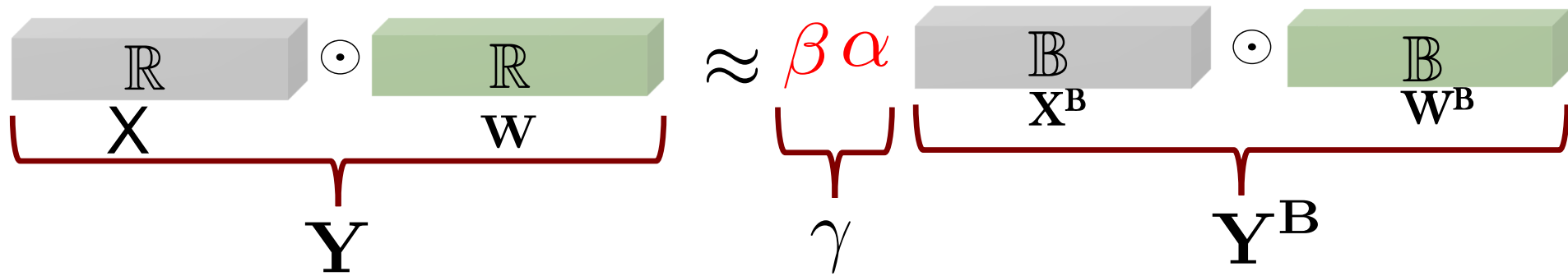


$$\mathbf{Y} \approx \gamma \mathbf{Y}^B$$

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

$$\mathbf{Y}^{B*} = \text{sign}(\mathbf{Y}) \quad \gamma^* = \frac{1}{n} \|\mathbf{Y}\|_{l_1}$$

Binary Input and Binary Weight (XNOR-Net)



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

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

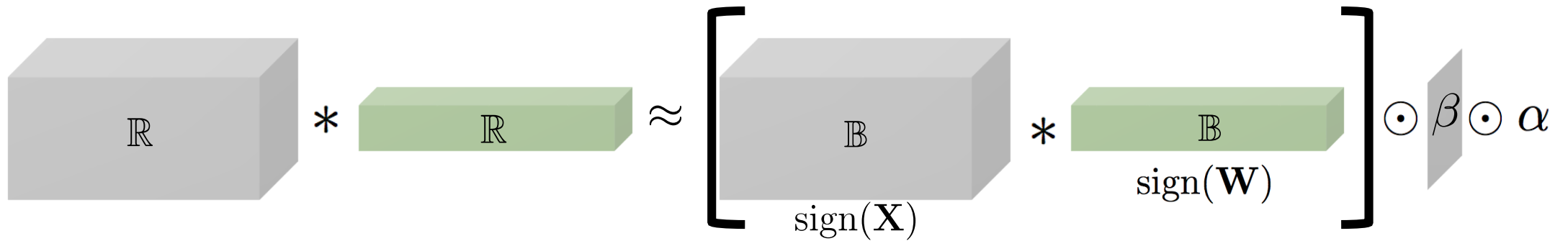
$$\mathbf{Y}^{\text{B}*} = \text{sign}(\mathbf{Y}) \quad \gamma^* = \frac{1}{n} \|\mathbf{Y}\|_{\ell_1}$$

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

$$\alpha^* = \frac{1}{n} \|\mathbf{W}\|_{\ell_1} \quad \beta^* = \frac{1}{n} \|\mathbf{X}\|_{\ell_1}$$

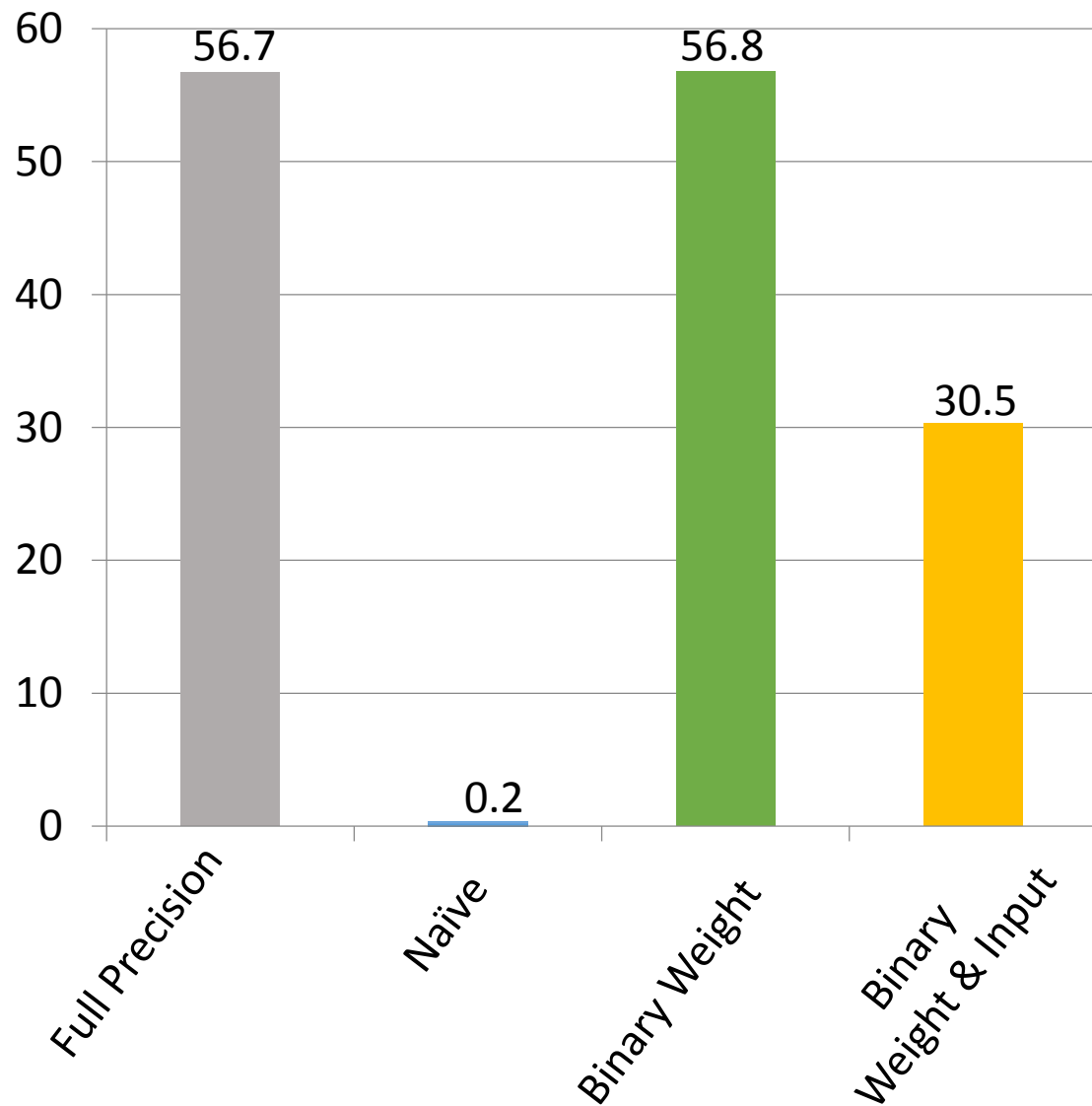
$$\mathbb{R} * \mathbb{R} \approx \left[\mathbb{B} * \mathbb{B} \right] \odot \beta \odot \alpha$$

The diagram shows a sequence of operations on tensors. On the left, a gray 3D block labeled \mathbb{R} is multiplied ($*$) by a green 3D block labeled \mathbb{R} . This is approximately equal (\approx) to a large bracketed expression. Inside the bracket, a gray 3D block labeled \mathbb{B} is multiplied ($*$) by a green 3D block labeled \mathbb{B} . Below the first \mathbb{B} is the label $\text{sign}(\mathbf{X})$, and below the second \mathbb{B} is the label $\text{sign}(\mathbf{W})$. The result of the bracketed multiplication is then element-wise multiplied (\odot) by a gray 2D plane labeled β , which is then element-wise multiplied (\odot) by another gray 2D plane labeled α .

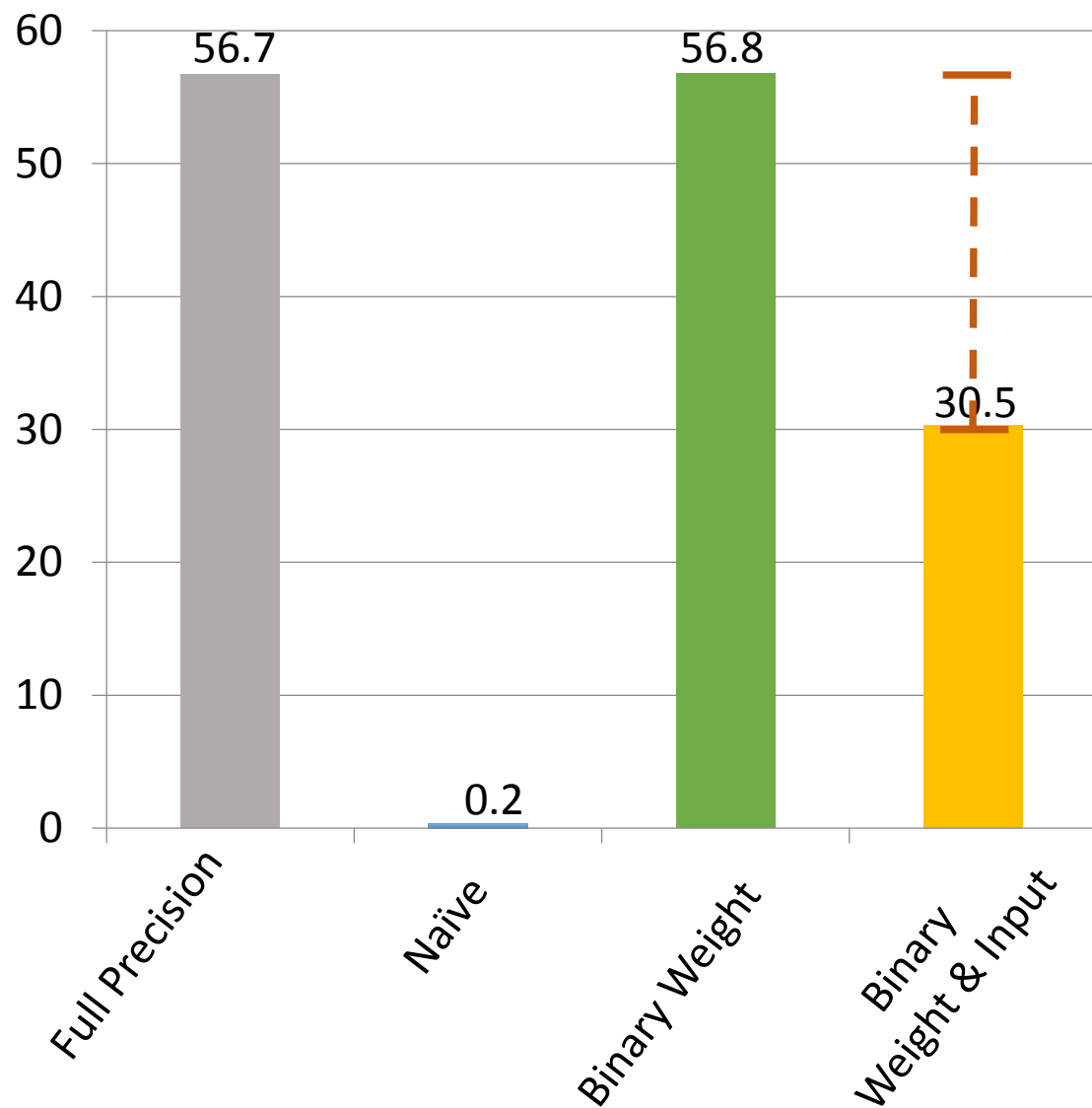


1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^B = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with α, \mathbf{W}^B
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with α, \mathbf{W}^B
9. Update \mathbf{W} ($\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$)

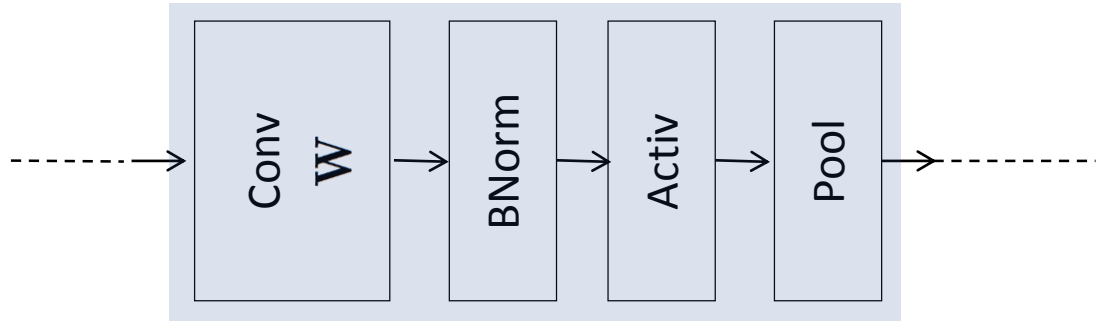
AlexNet Top-1 (%) ILSVRC2012



AlexNet Top-1 (%) ILSVRC2012

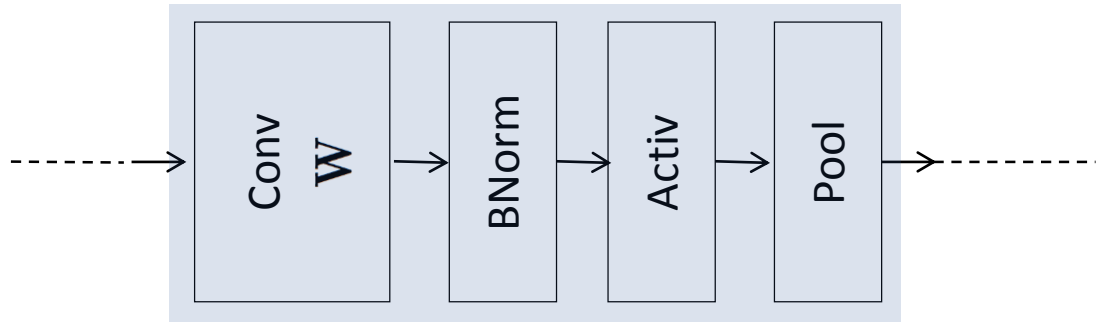


Network Structure in XNOR-Networks

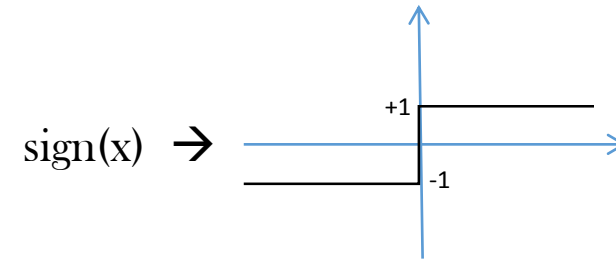


A typical block in CNN

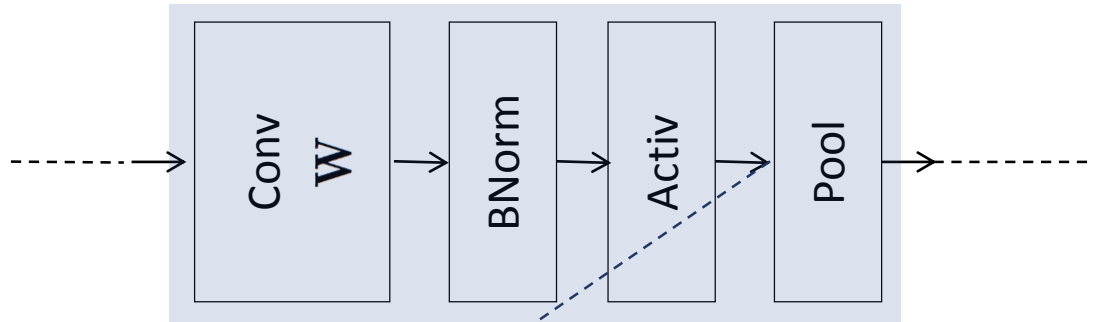
Network Structure in XNOR-Networks



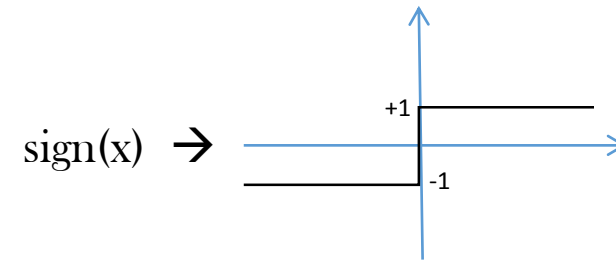
A typical block in CNN



Network Structure in XNOR-Networks

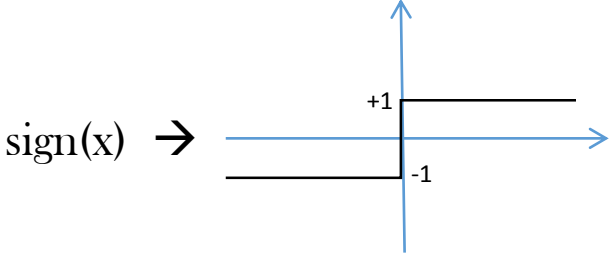
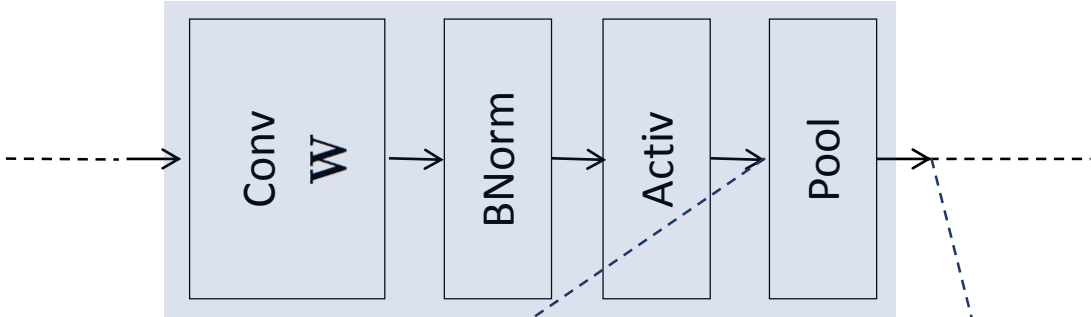


A typical block in CNN



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

Network Structure in XNOR-Networks



A typical block in CNN

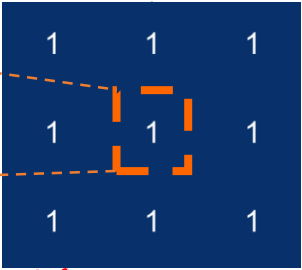
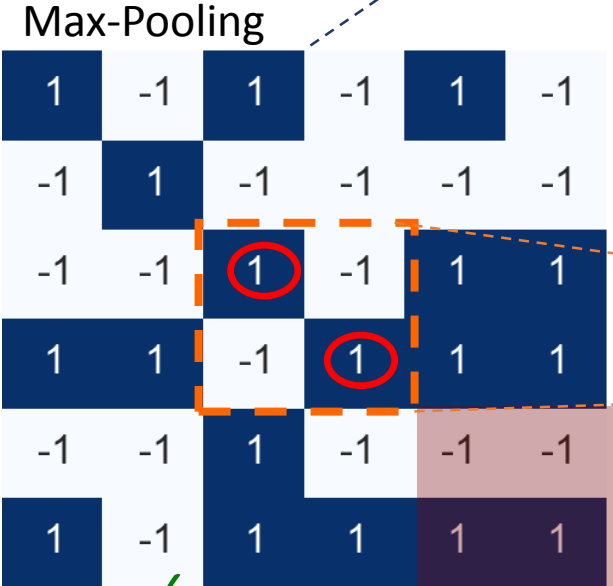
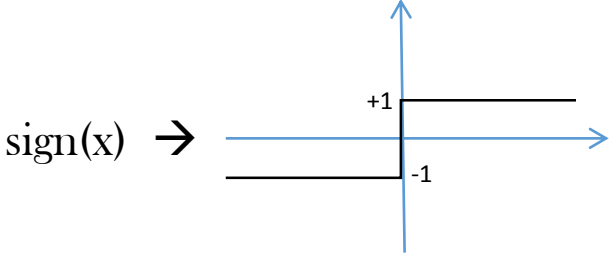
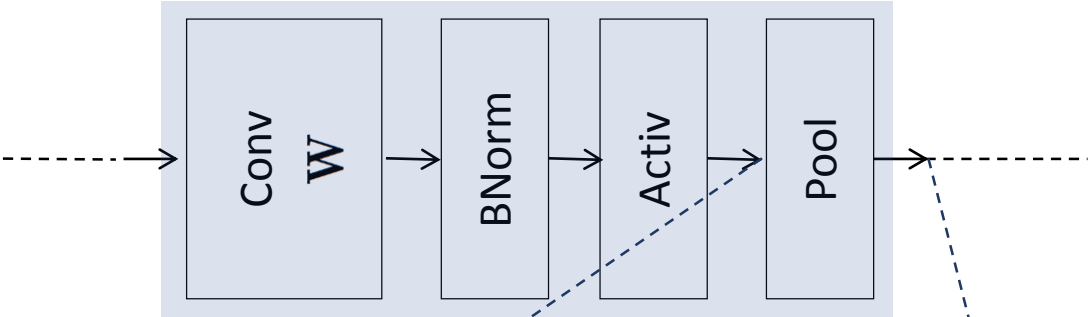
Max-Pooling

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

1	1	1
1	1	1
1	1	1

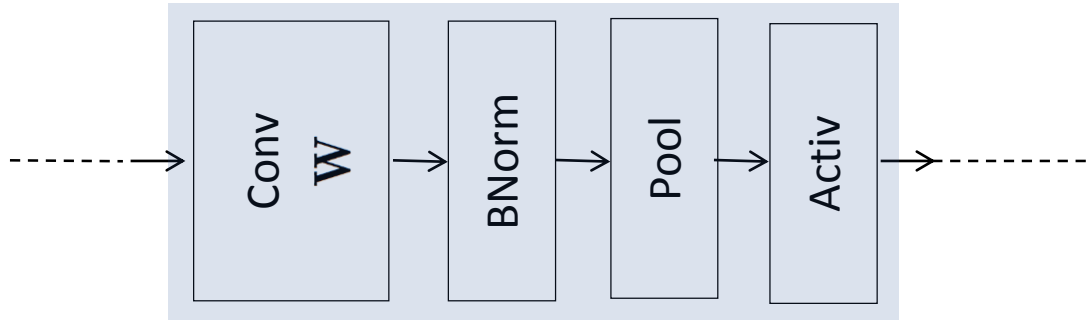
X Information Loss

Network Structure in XNOR-Networks

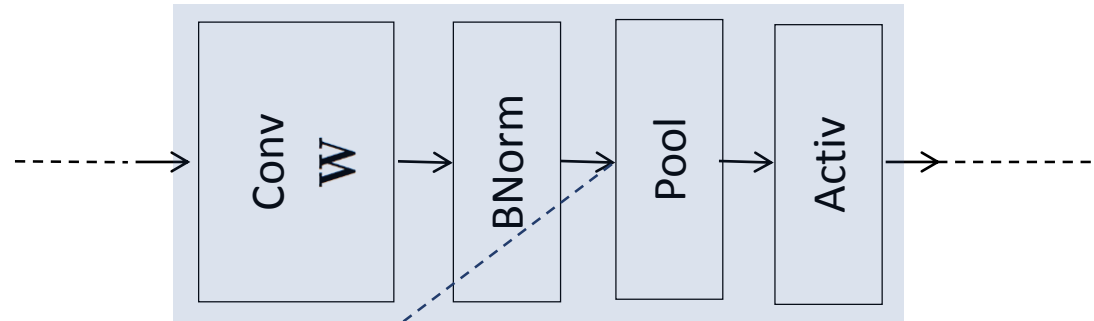


✗ Information Loss

Network Structure in XNOR-Networks

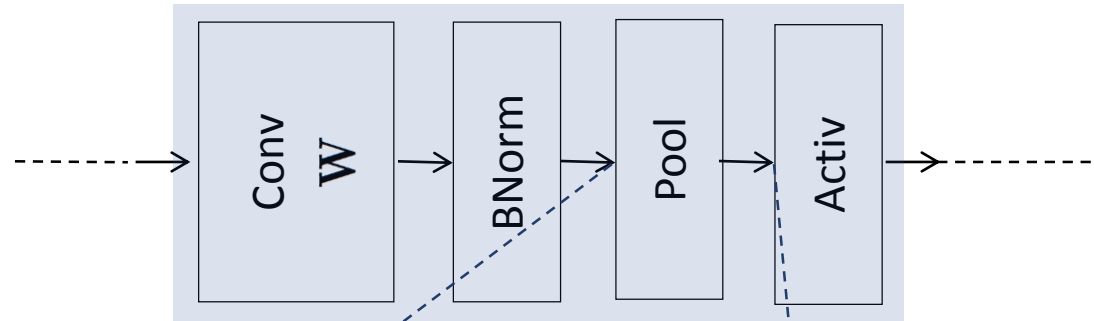


Network Structure in XNOR-Networks



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

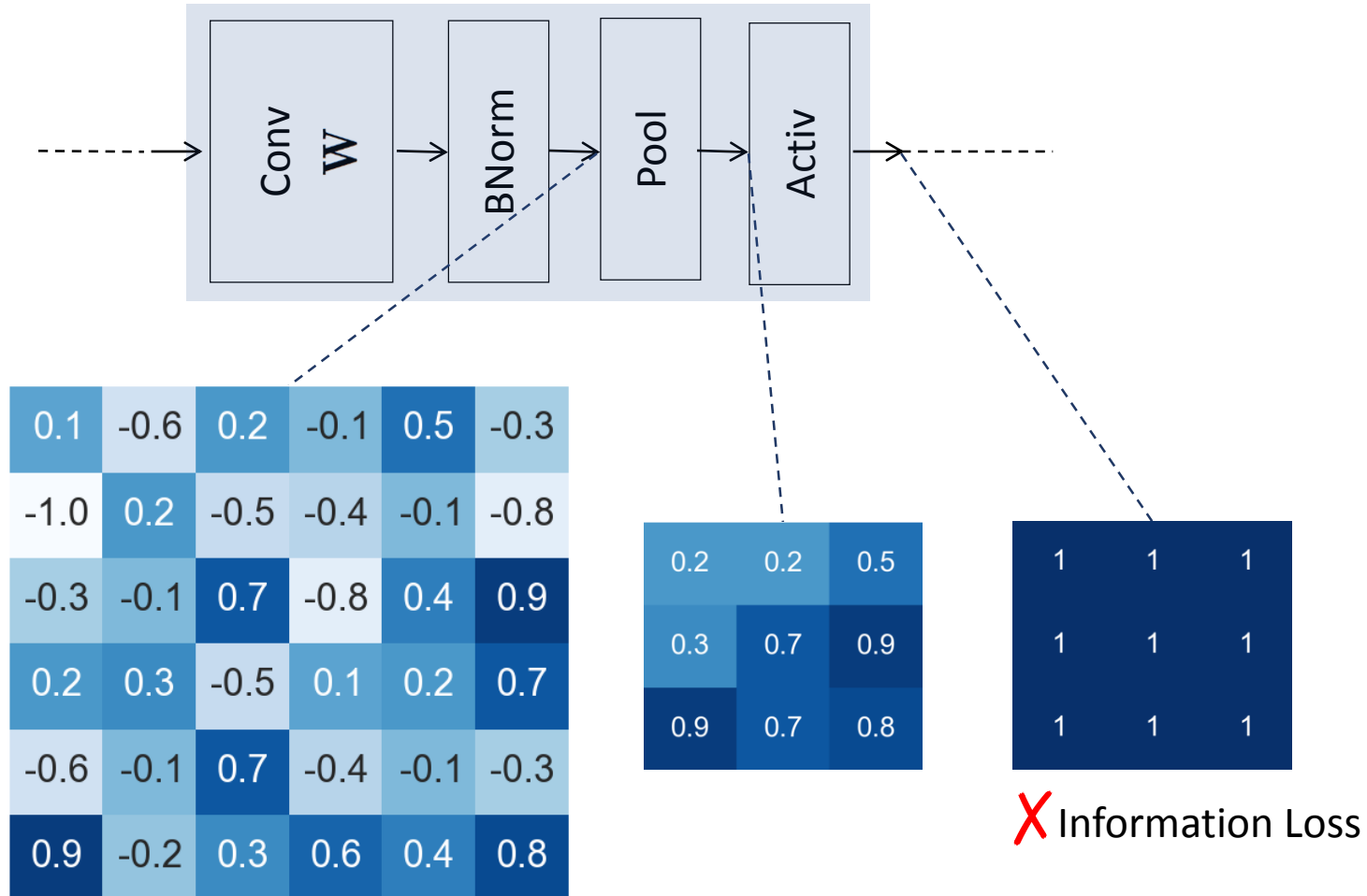
Network Structure in XNOR-Networks



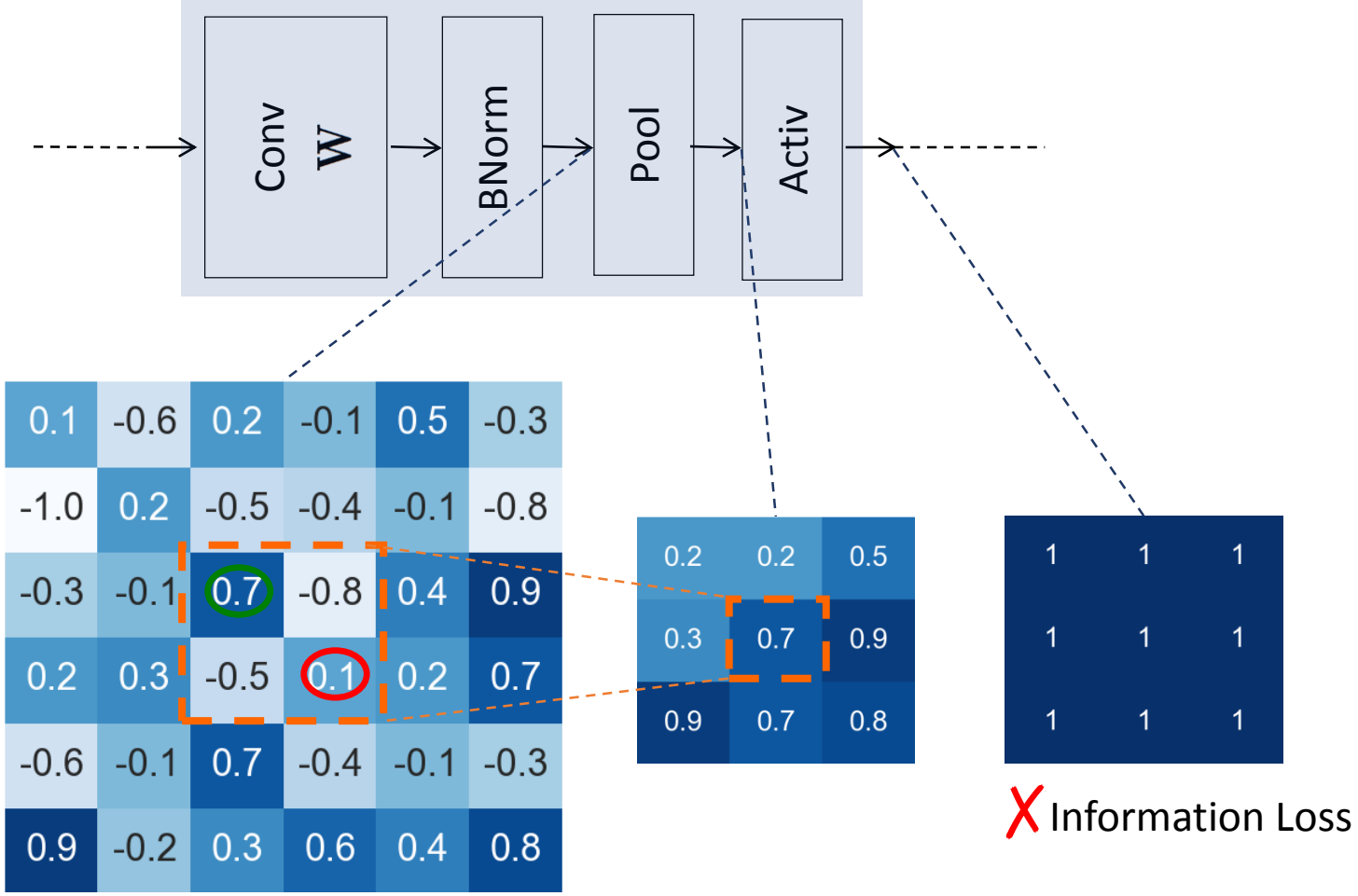
0.1	-0.6	0.2	-0.1	0.5	-0.3
-1.0	0.2	-0.5	-0.4	-0.1	-0.8
-0.3	-0.1	0.7	-0.8	0.4	0.9
0.2	0.3	-0.5	0.1	0.2	0.7
-0.6	-0.1	0.7	-0.4	-0.1	-0.3
0.9	-0.2	0.3	0.6	0.4	0.8

0.2	0.2	0.5
0.3	0.7	0.9
0.9	0.7	0.8

Network Structure in XNOR-Networks

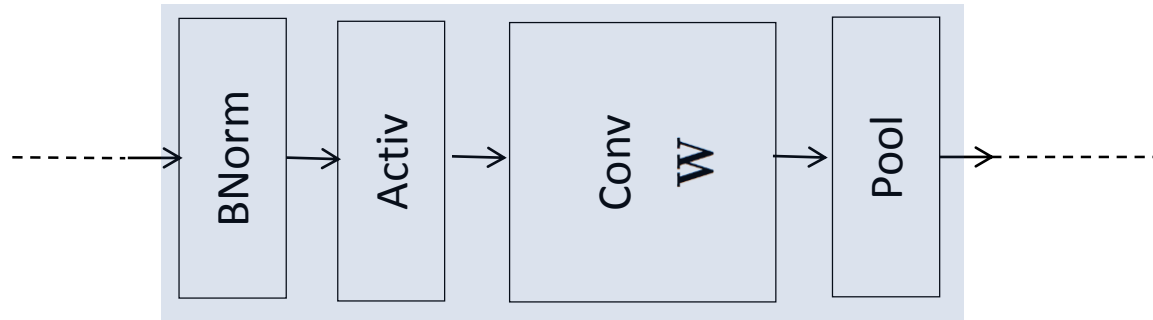


Network Structure in XNOR-Networks

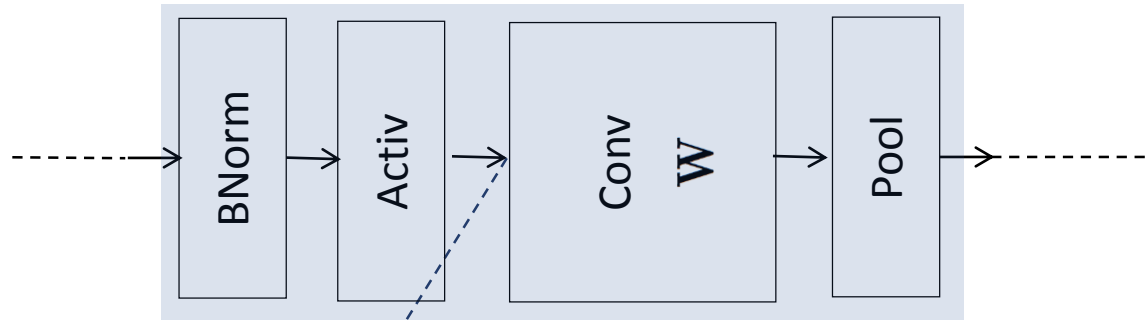


✓ Multiple Maximums

Network Structure in XNOR-Networks

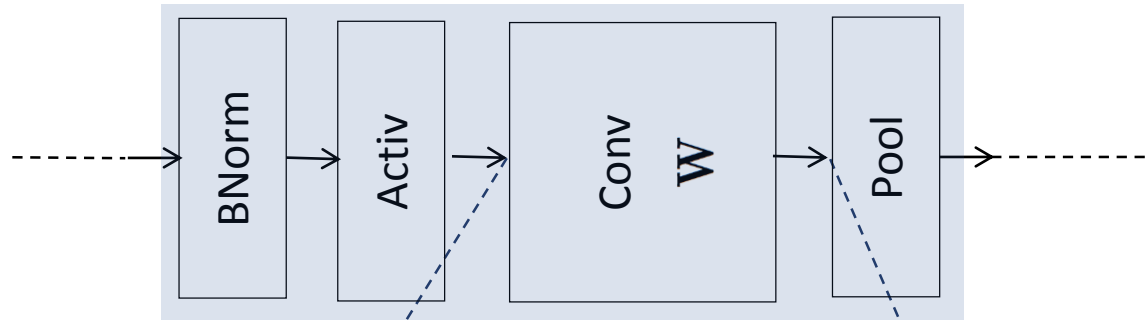


Network Structure in XNOR-Networks



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

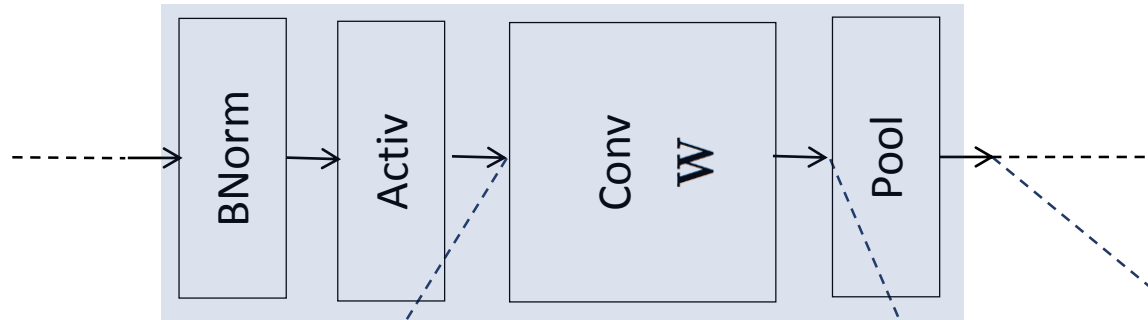
Network Structure in XNOR-Networks



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

Network Structure in XNOR-Networks

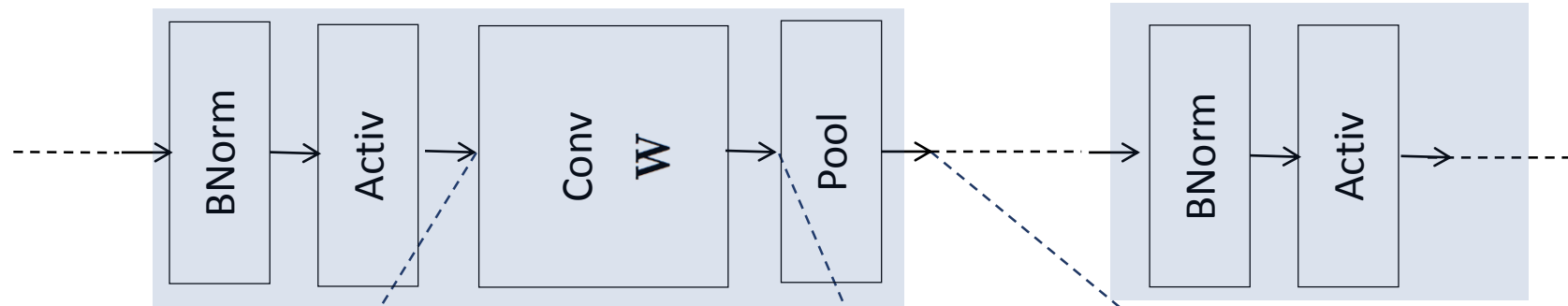


1	-1	1	-1	1	-1
-1	1	-1	-1	-1	-1
-1	-1	1	-1	1	1
1	1	-1	1	1	1
-1	-1	1	-1	-1	-1
1	-1	1	1	1	1

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

Network Structure in XNOR-Networks

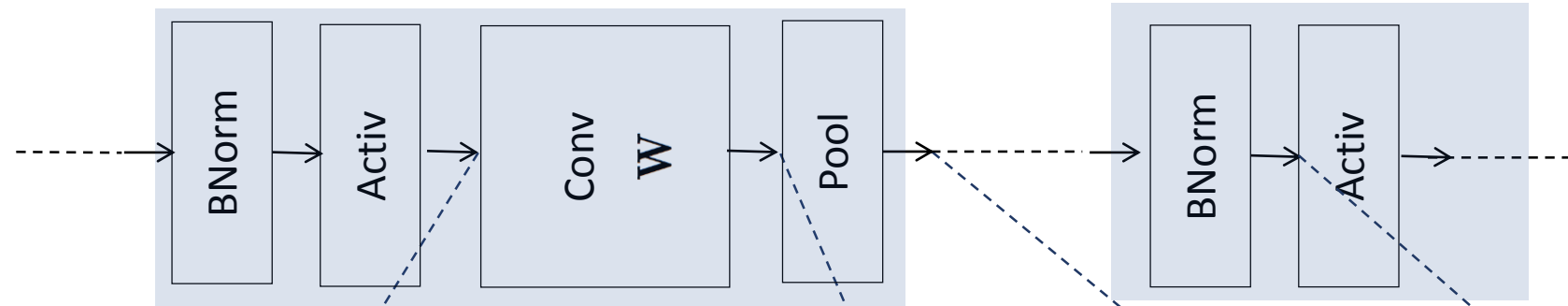


1	-1	1	-1	1	-1
-1	1	-1	-1	-1	-1
-1	-1	1	-1	1	1
1	1	-1	1	1	1
-1	-1	1	-1	-1	-1
1	-1	1	1	1	1

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

Network Structure in XNOR-Networks



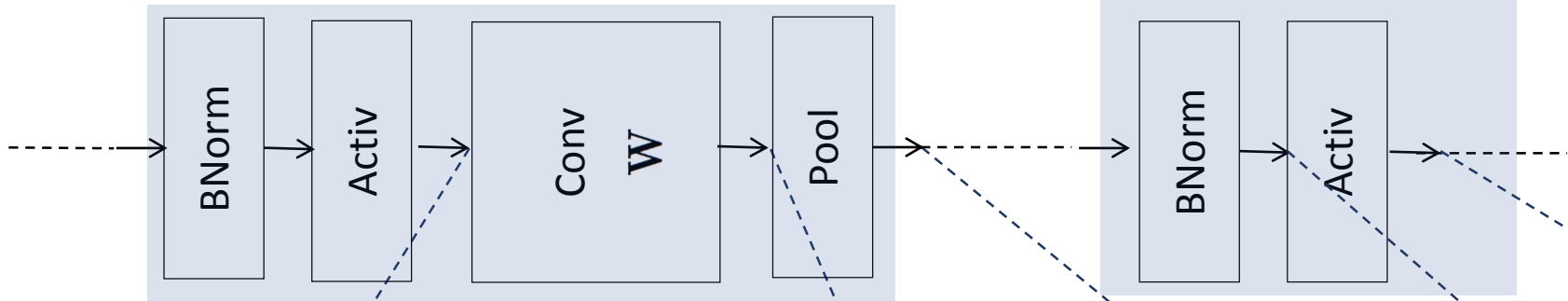
1	-1	1	-1	1	-1
-1	1	-1	-1	-1	-1
-1	-1	1	-1	1	1
1	1	-1	1	1	1
-1	-1	1	-1	-1	-1
1	-1	1	1	1	1

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

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

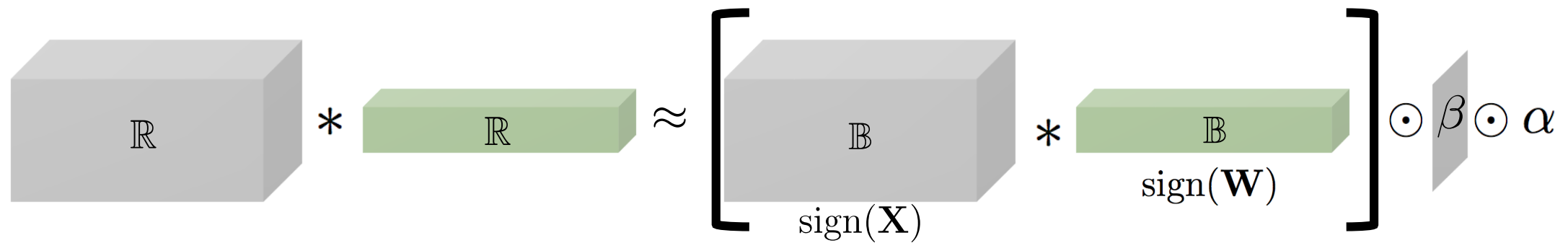
0.1	-0.6	0.2	-0.1	0.5	-0.3
-1.0	0.2	-0.5	-0.4	-0.1	-0.8
-0.3	-0.1	0.7	-0.8	0.4	0.9
0.2	0.3	-0.5	0.1	0.2	0.7
-0.6	-0.1	0.7	-0.4	-0.1	-0.3
0.9	-0.2	0.3	0.6	0.4	0.8

0.2	0.2	0.5
0.3	0.7	0.9
0.9	0.7	0.8

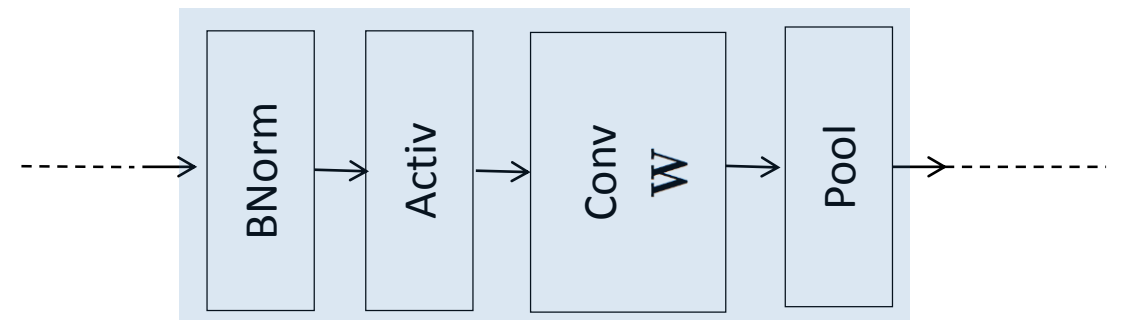
-0.1	-0.2	0.3
0.4	0.3	-0.1
-0.7	0.2	-0.6

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

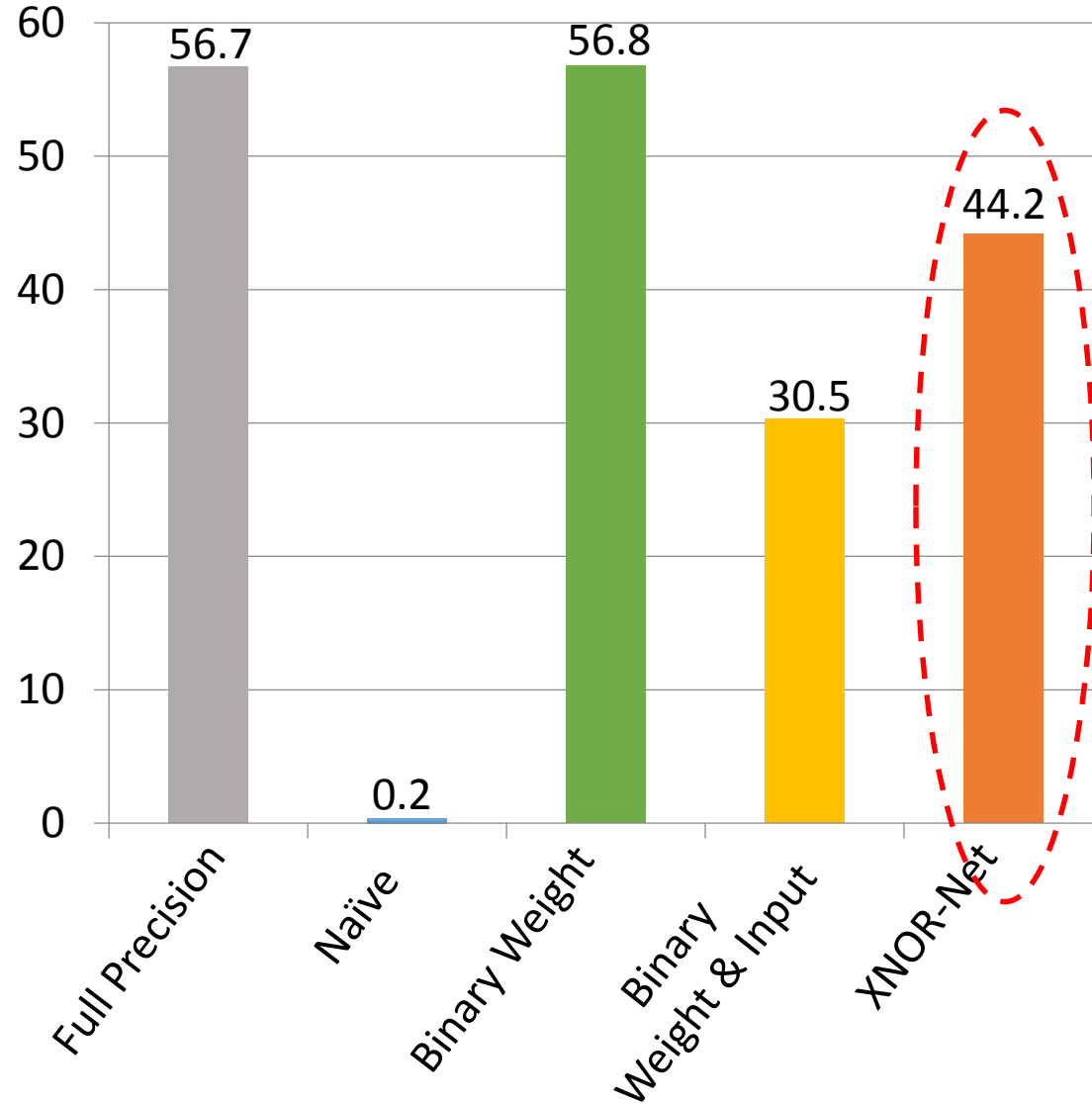
- ✓ Information Loss
- ✓ Multiple Maximums



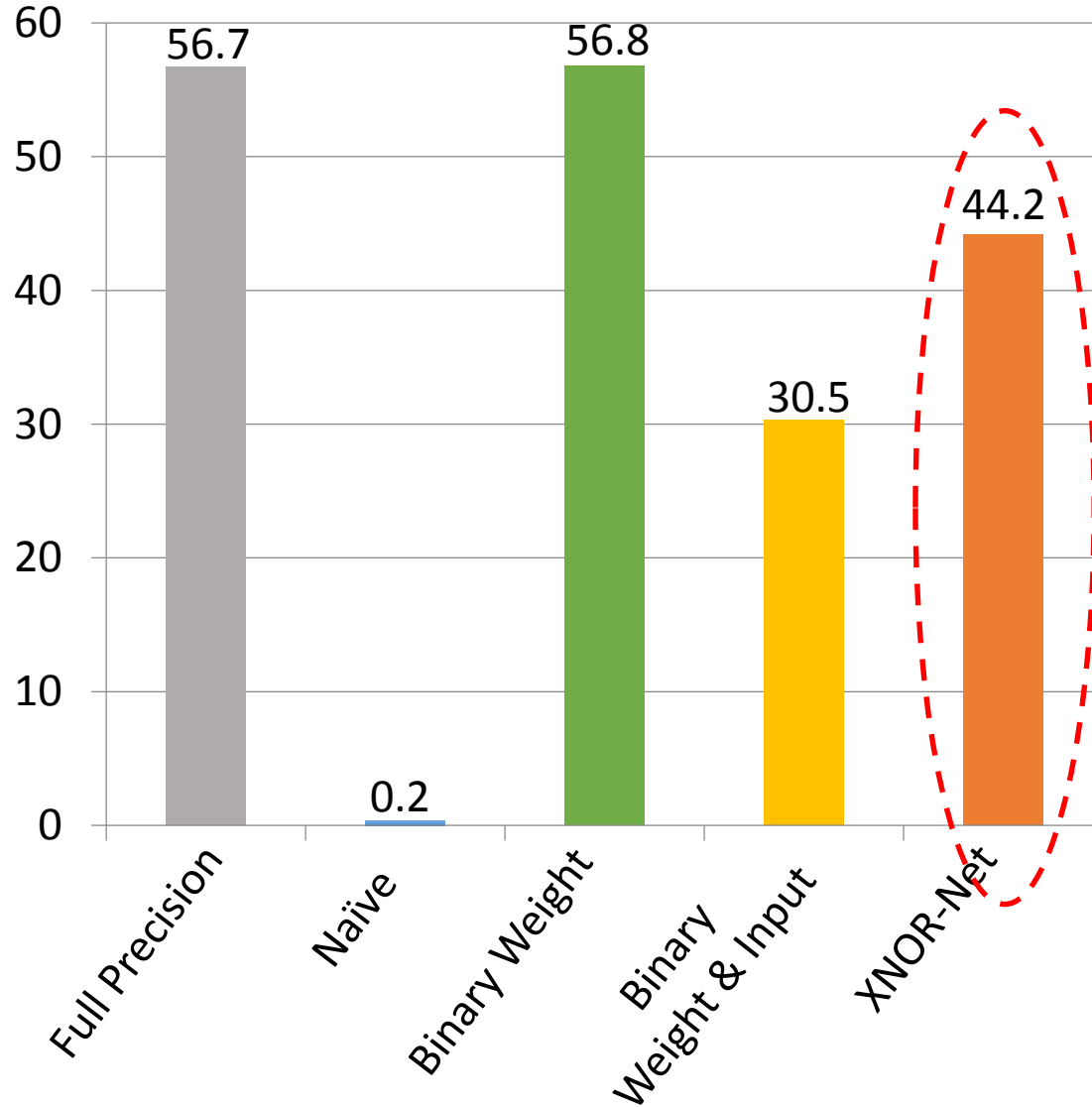
1. Randomly initialize \mathbf{W}
2. For $iter = 1$ to N
3. Load a random input image \mathbf{X}
4. $\mathbf{W}^{\mathbb{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|\mathbf{W}\|_{\ell_1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbb{B}}$
7. Compute loss function \mathbf{C}
8. $\frac{\partial \mathbf{C}}{\partial \mathbf{W}} =$ Backward pass with $\alpha, \mathbf{W}^{\mathbb{B}}$
9. Update \mathbf{W} ($\mathbf{W} = \mathbf{W} - \frac{\partial \mathbf{C}}{\partial \mathbf{W}}$)



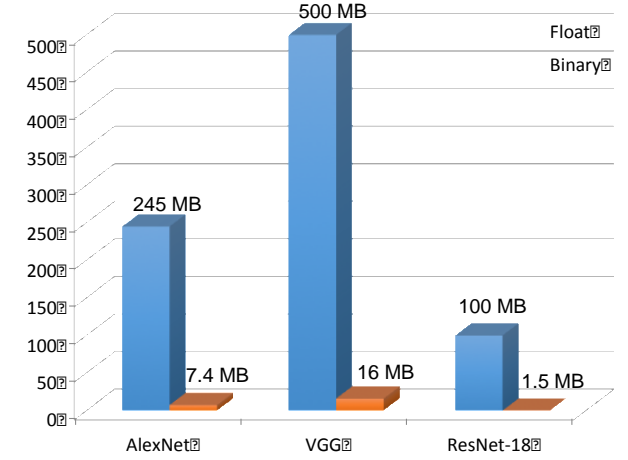
AlexNet Top-1 (%) ILSVRC2012



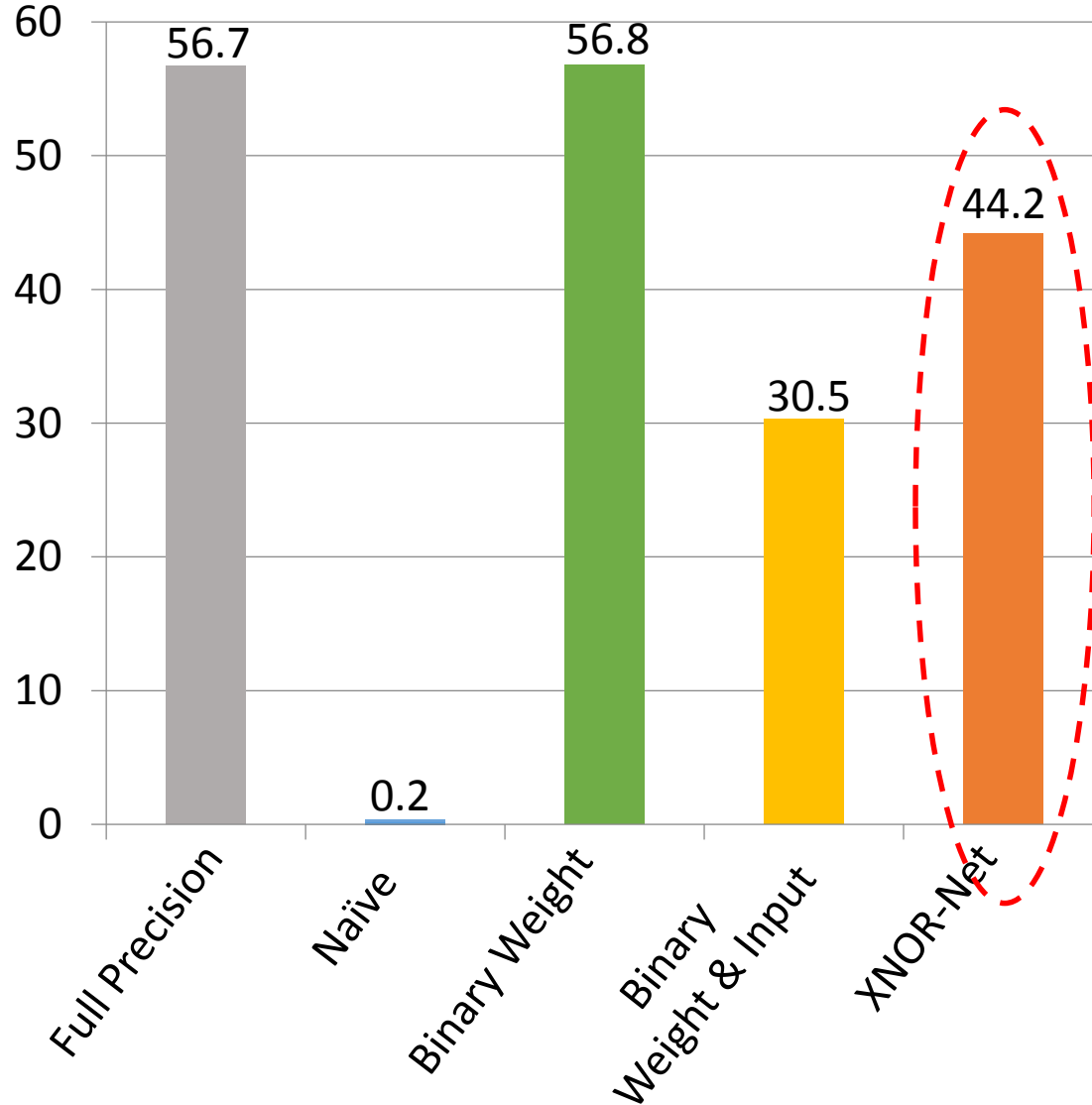
AlexNet Top-1 (%) ILSVRC2012



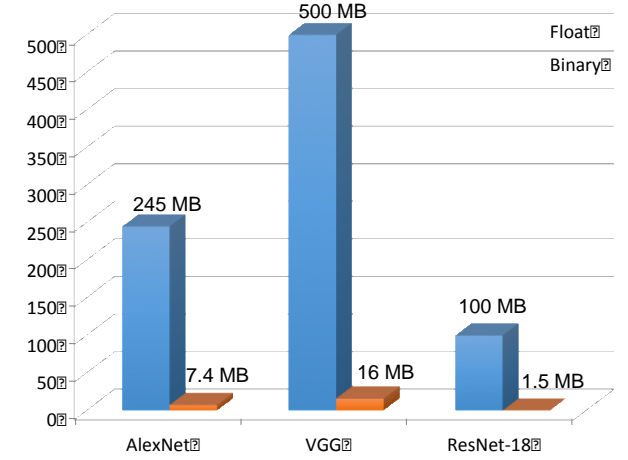
✓ 32x Smaller Model



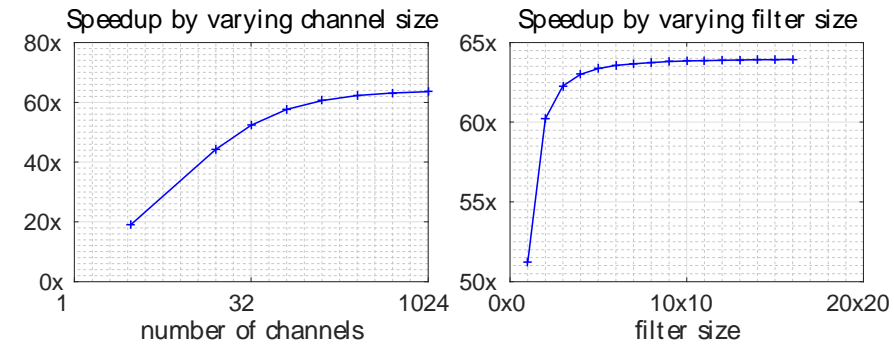
AlexNet Top-1 (%) ILSVRC2012



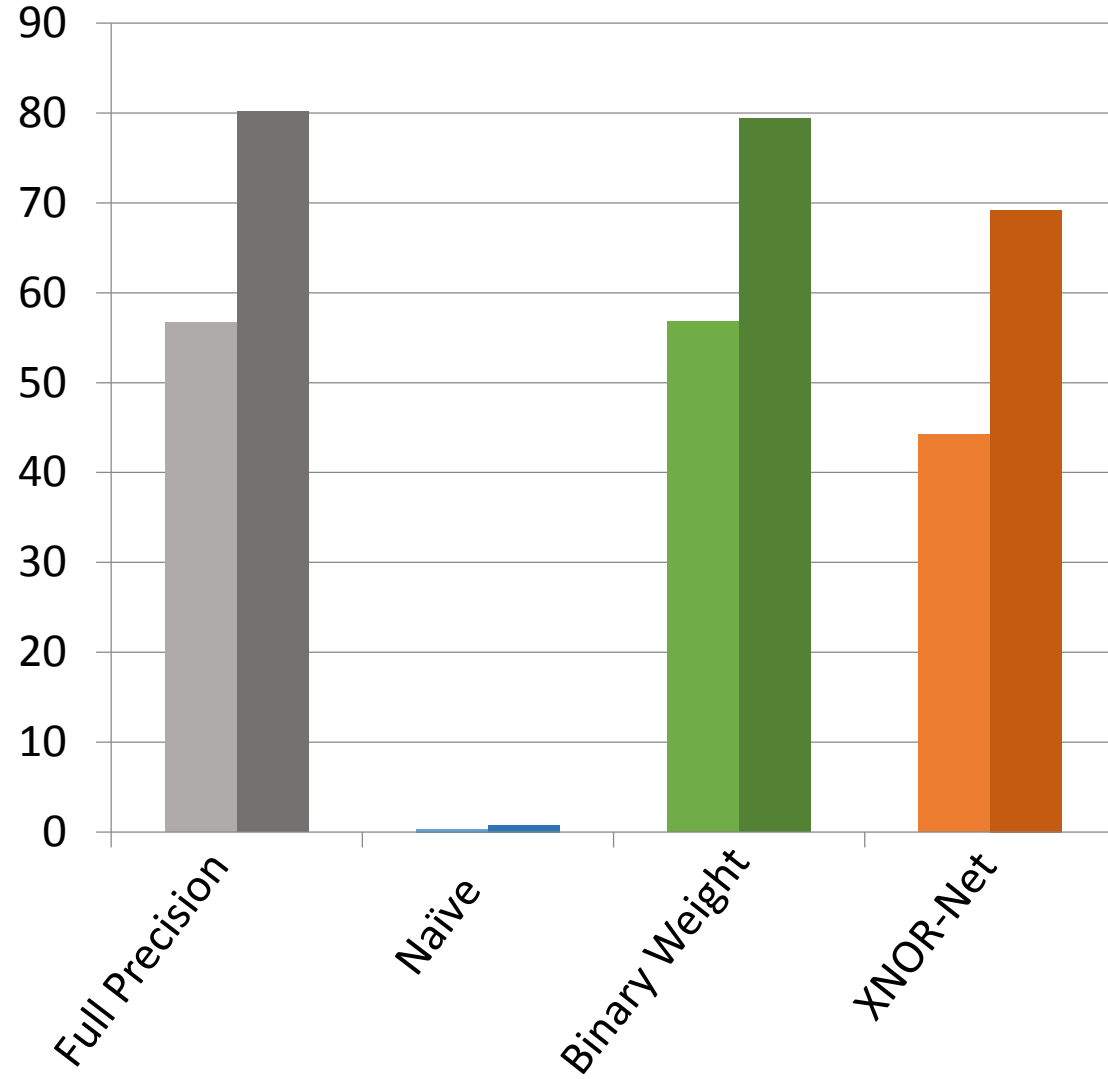
✓ 32x Smaller Model

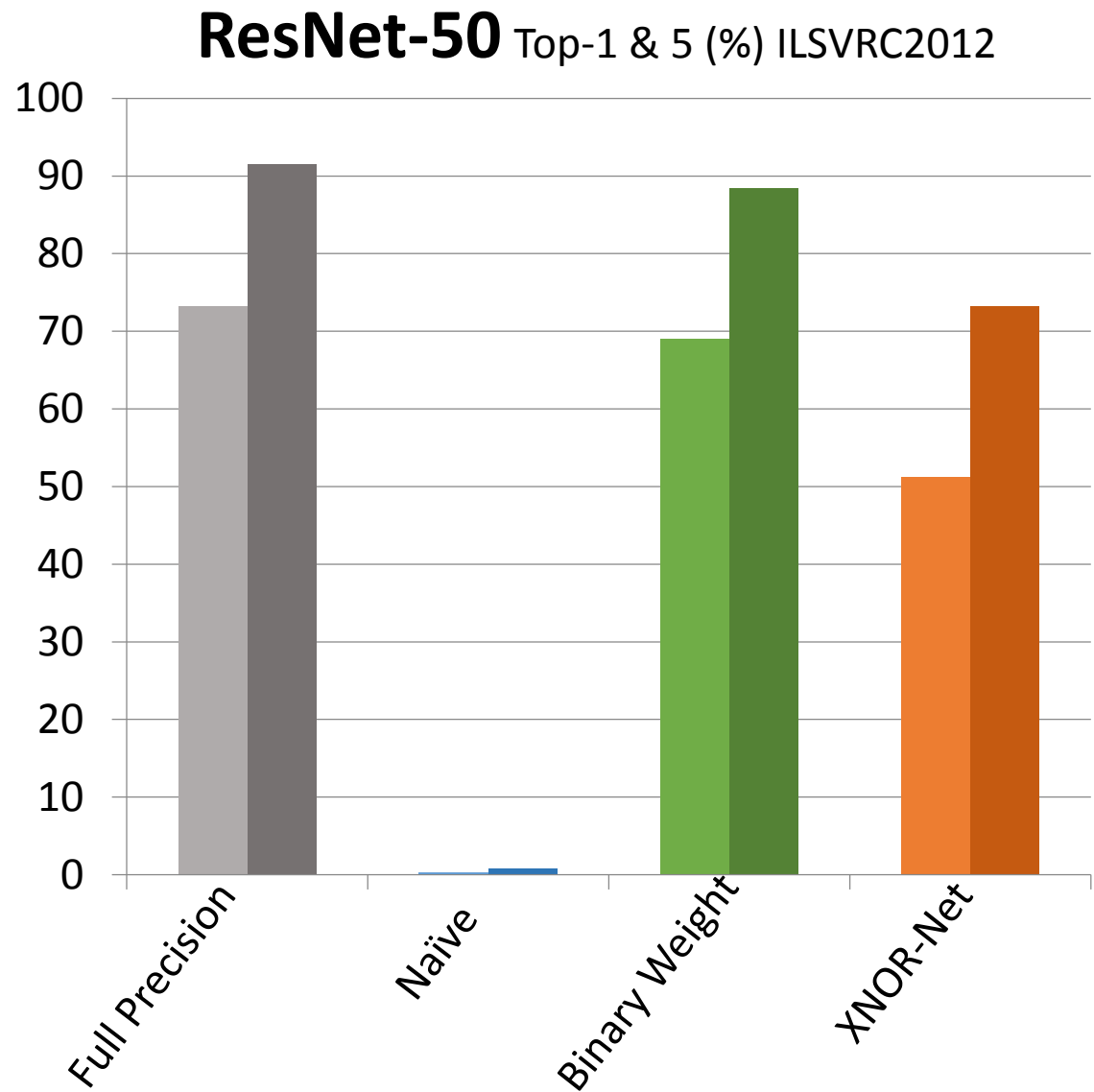
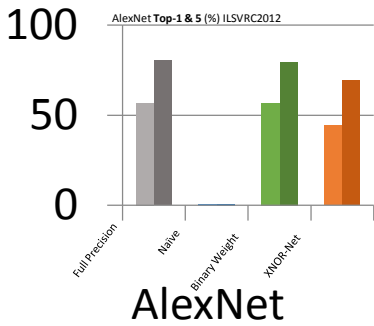


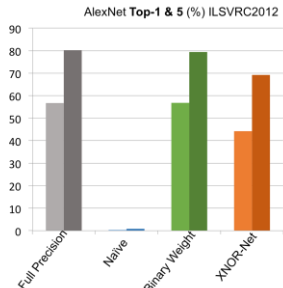
✓ 58x Less Computation



AlexNet Top-1 & 5 (%) ILSVRC2012

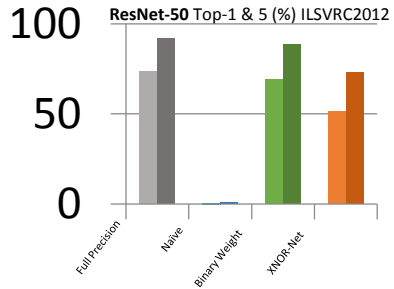
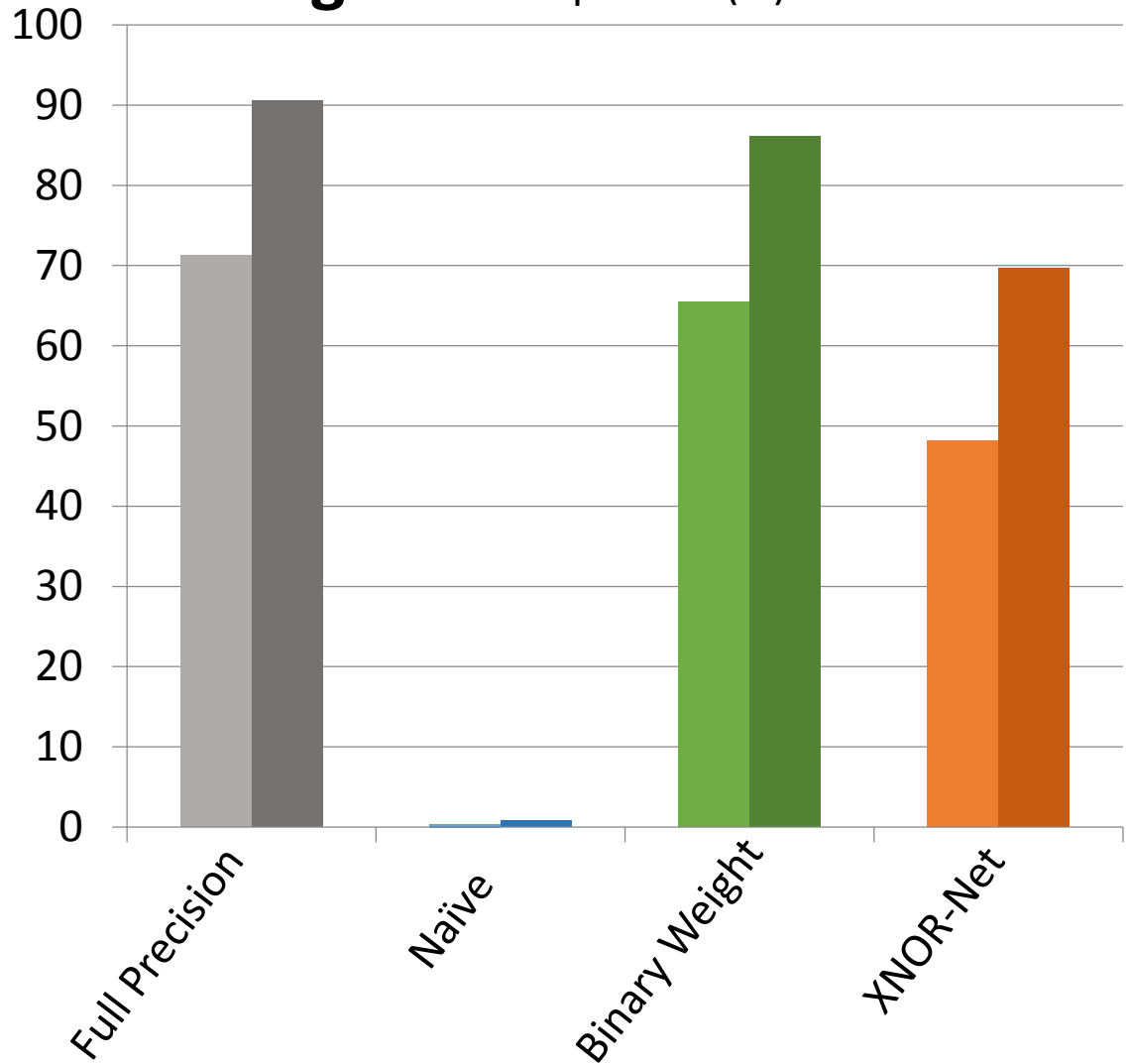




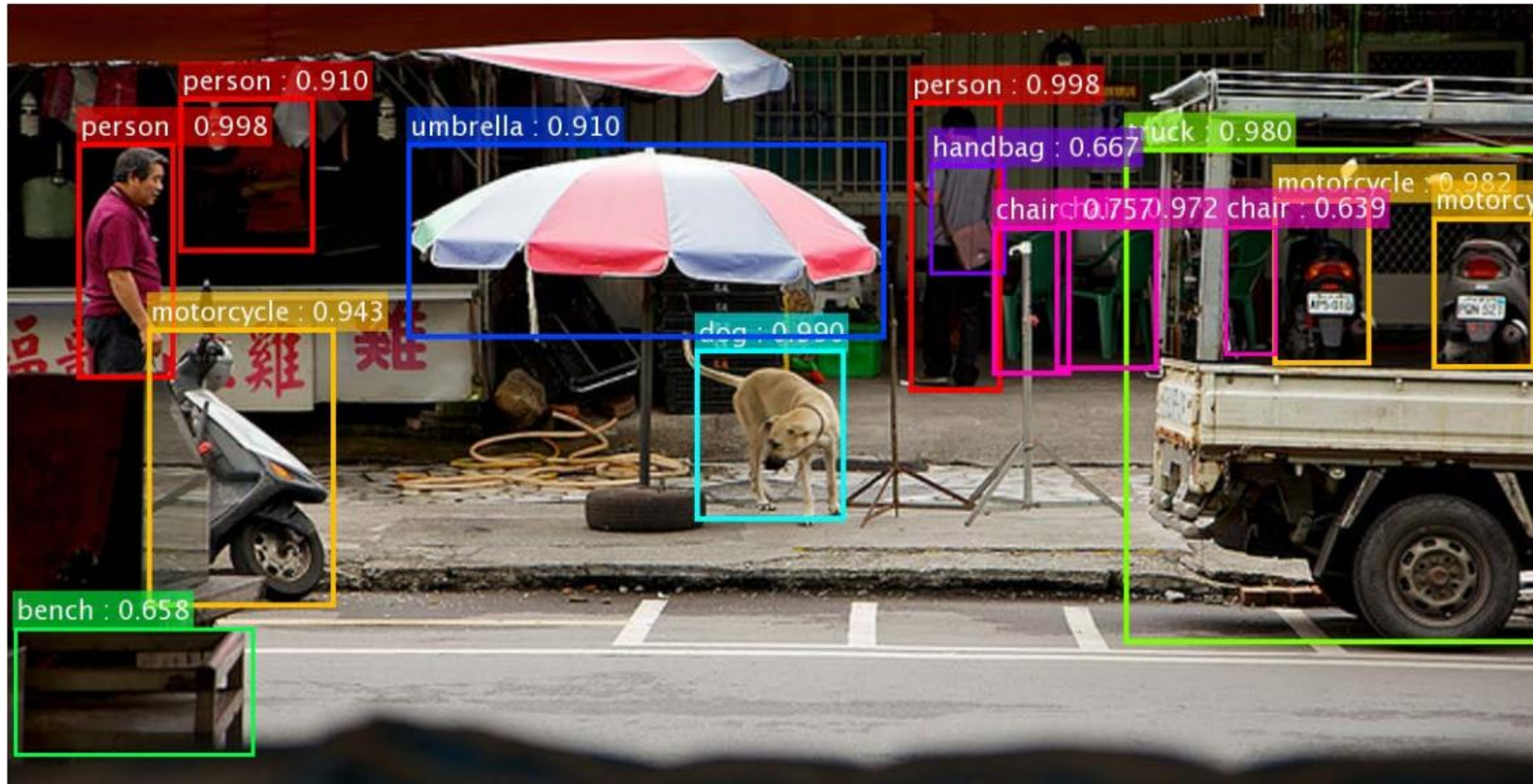


AlexNet

GoogLeNet Top-1 & 5 (%) ILSVRC2012



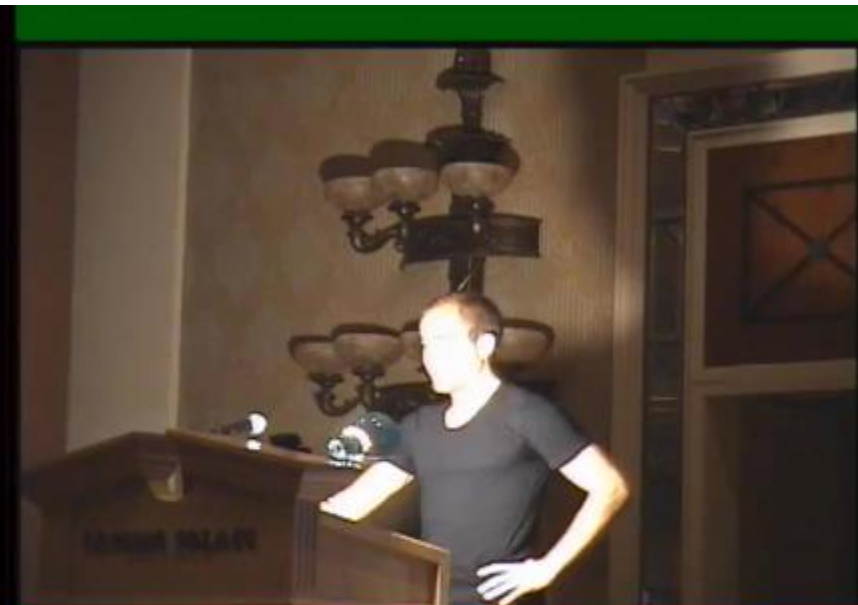
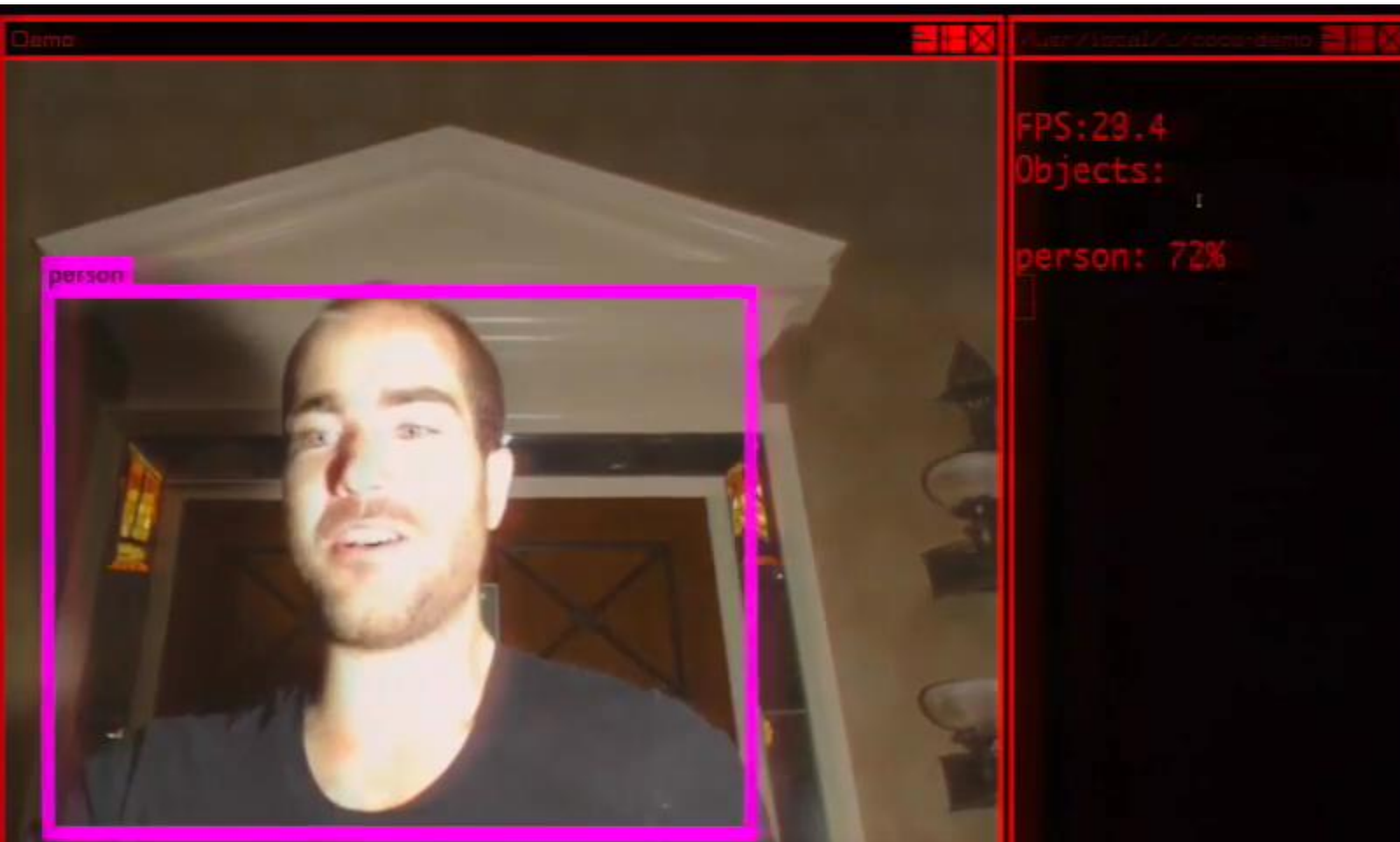
Object Detection



[He et al, 2015]

YOLO: Fastest Object Detector

[Redmon et al. CVPR 2016]

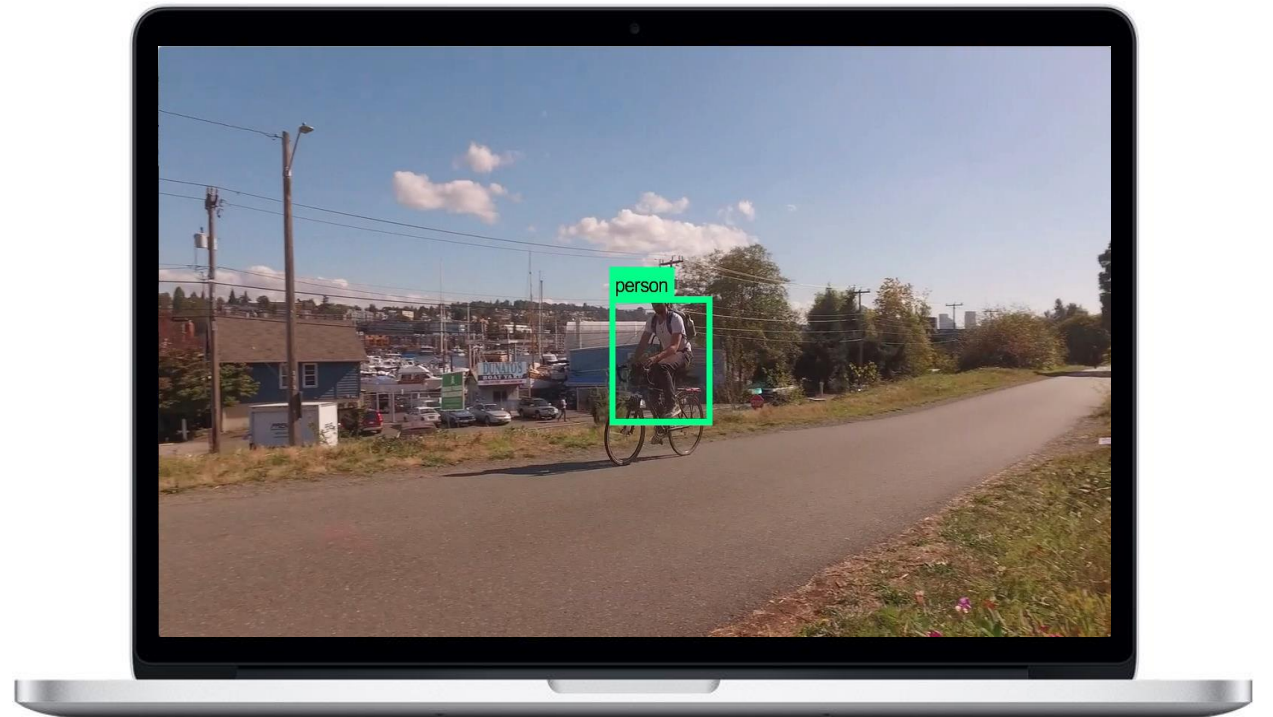


CVPR 2016



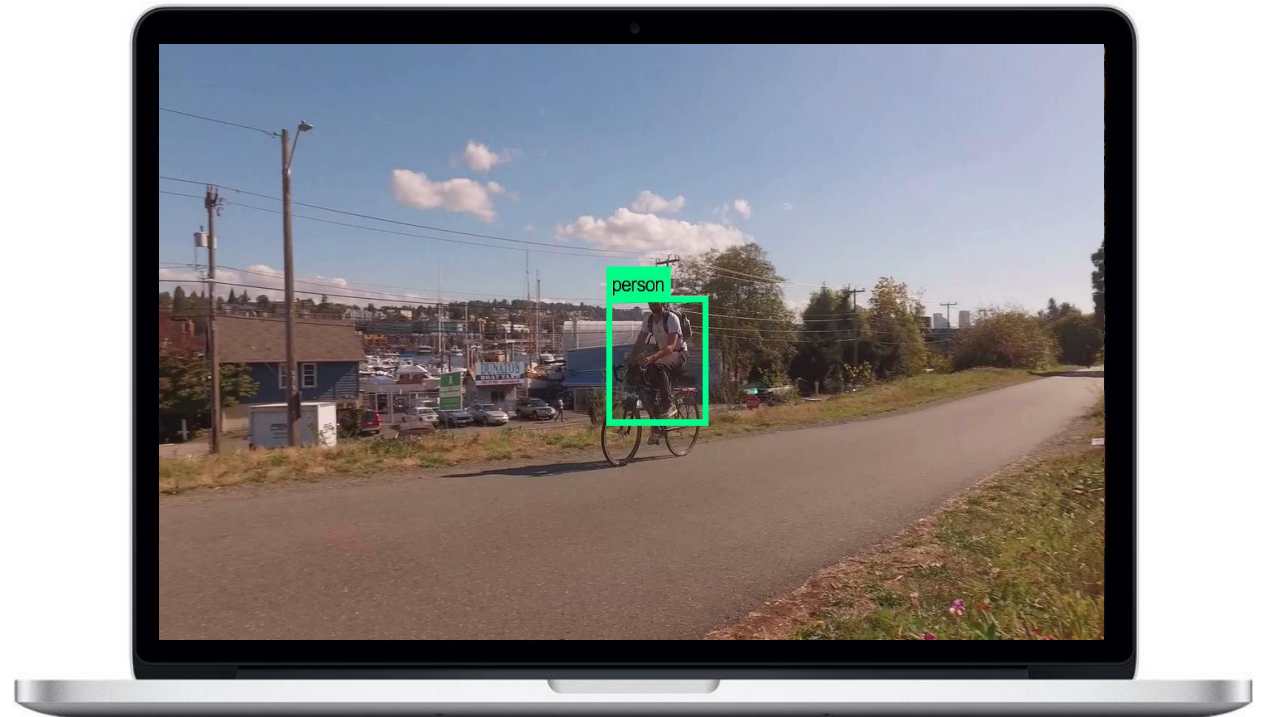


YOLO on CPU (NOT GPU)
Fastest Object Detector
[Redmon et al. CVPR 2016]



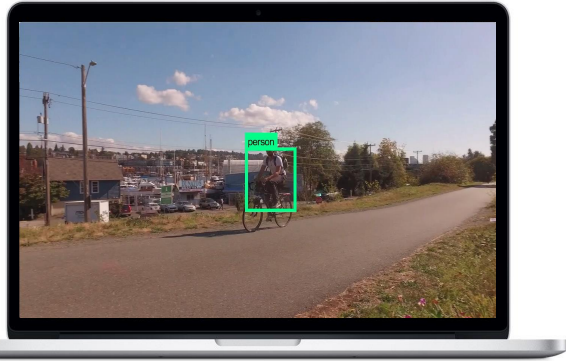


YOLO on CPU (NOT GPU)
Fastest Object Detector
[Redmon et al. CVPR 2016]





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

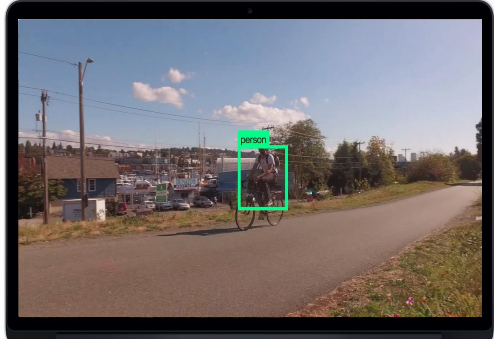


Our Method

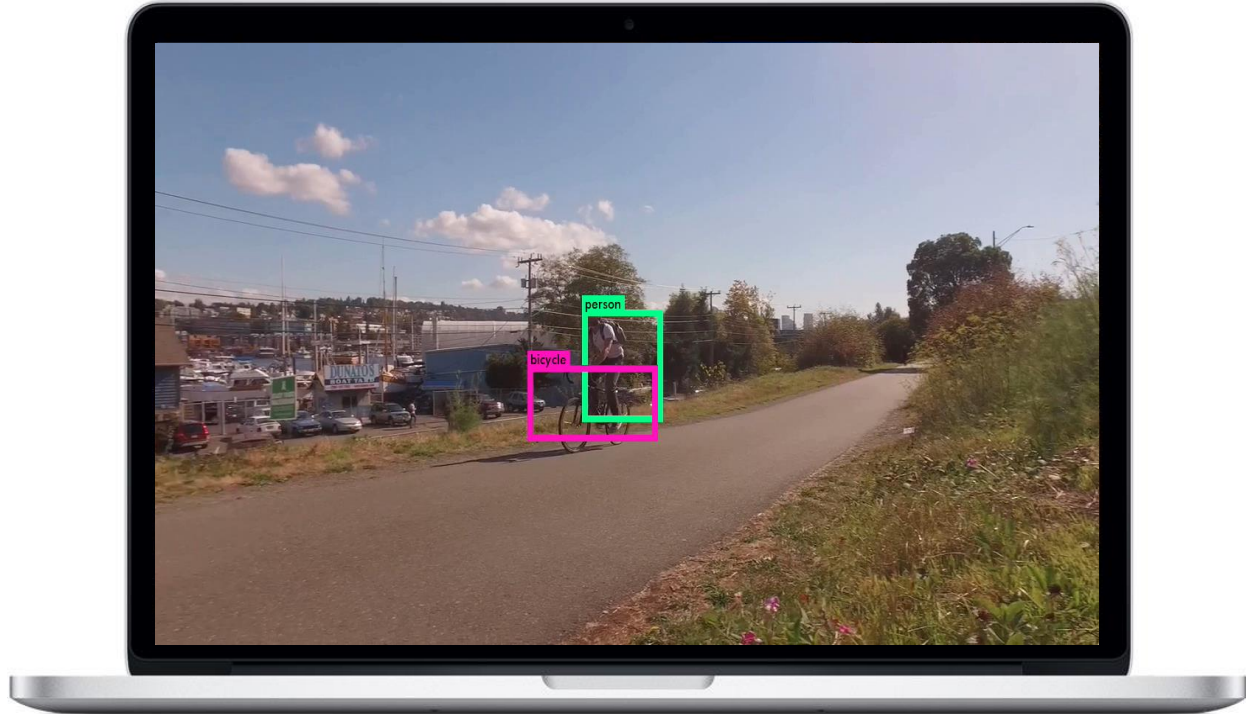




YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]

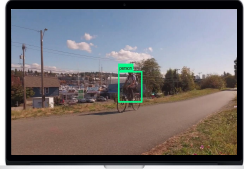


Our Method On CPU (NOT GPU)

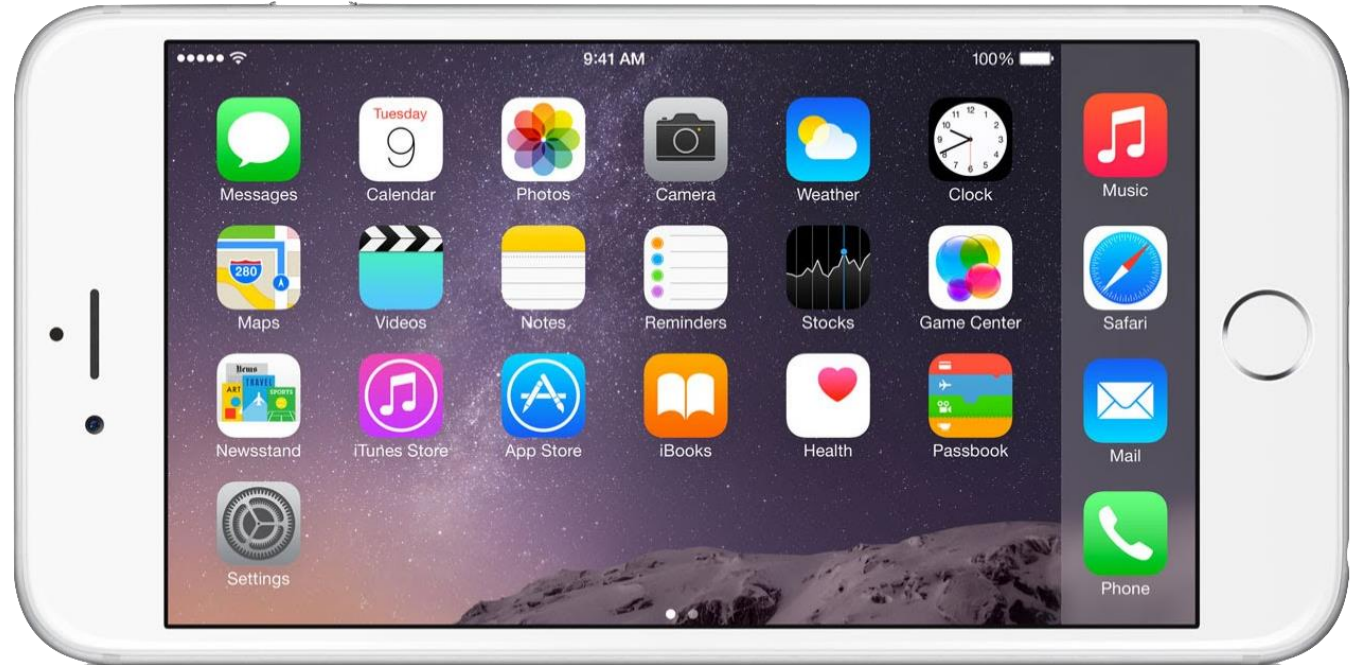
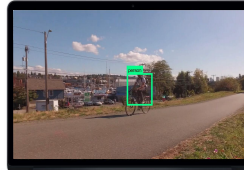




YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]



Our Method
On CPU (NOT GPU)

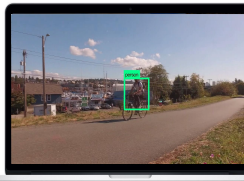




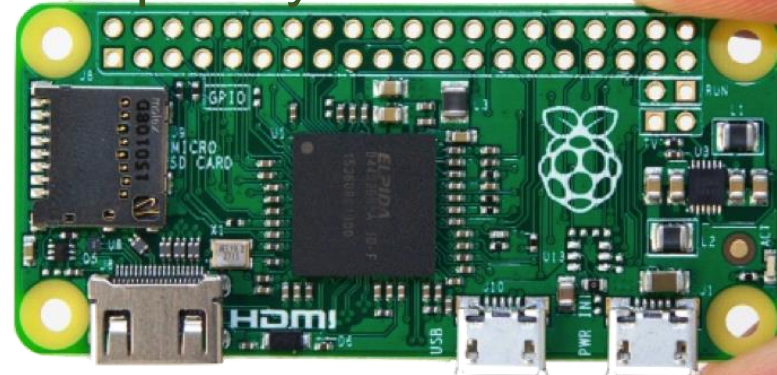
YOLO
Fastest Object Detector
[Redmon et al. CVPR 2016]



Our Method



Raspberry Pi Zero



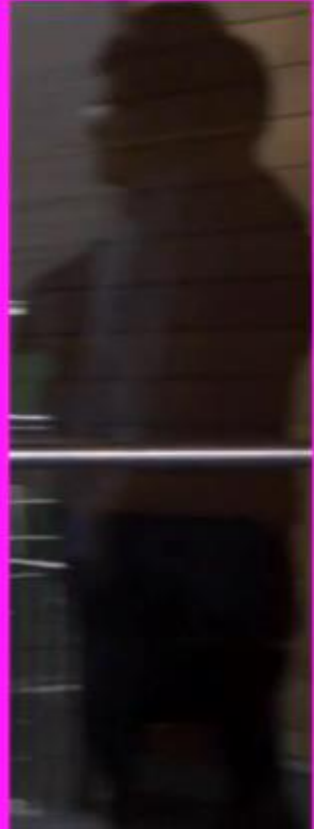
\$5

Thank You !

person



person



p

Thank You !



Carlo C. del Mundo

Dmitry Belenko