Efficient and Accurate CNN Models at Edge Compute Platforms

Mohammad Rastegari

September 2018

XNOR.AI

AI is confined to the cloud

far from the users at the edge

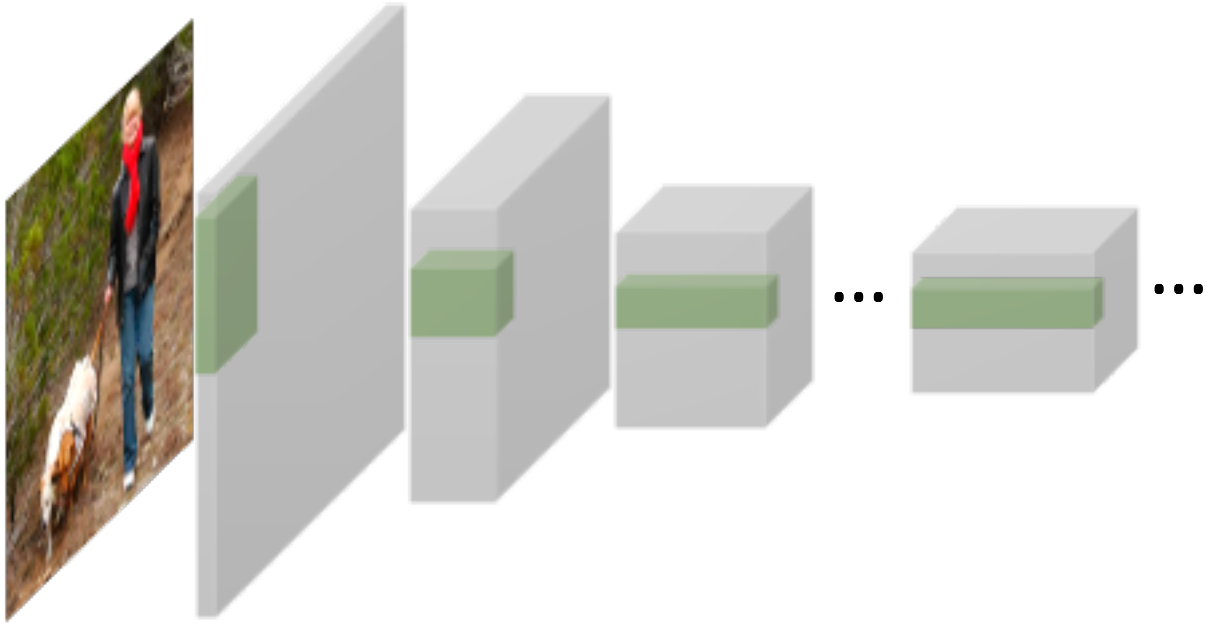
Bridging the growing divide between AI models dependent on the cloud and devices running at the edge

Deep learning models reliant on the cloud

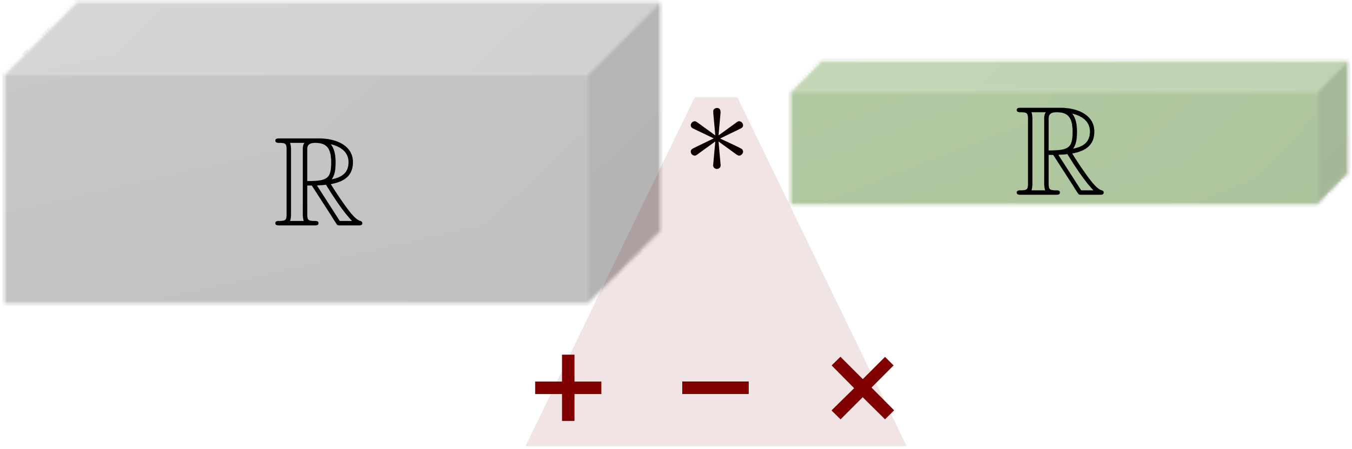
Growing demand for edge devices

Intelligent cameras that preserve privacy, security and bandwidth at home

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

Solutions

- Lower Precision (Quantization)
  - Fixed point, binary (XNOR-Net)
- Sparse Models
  - Lookup based CNN, Factorizations
- Compact Network Design
  - Mobile Net
- How to improve the accuracy?
  - Label Refinery

Lower Precision

Reducing Precision

- Saving Memory
- Saving Computation

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

32-bit 8-bit 1-bit

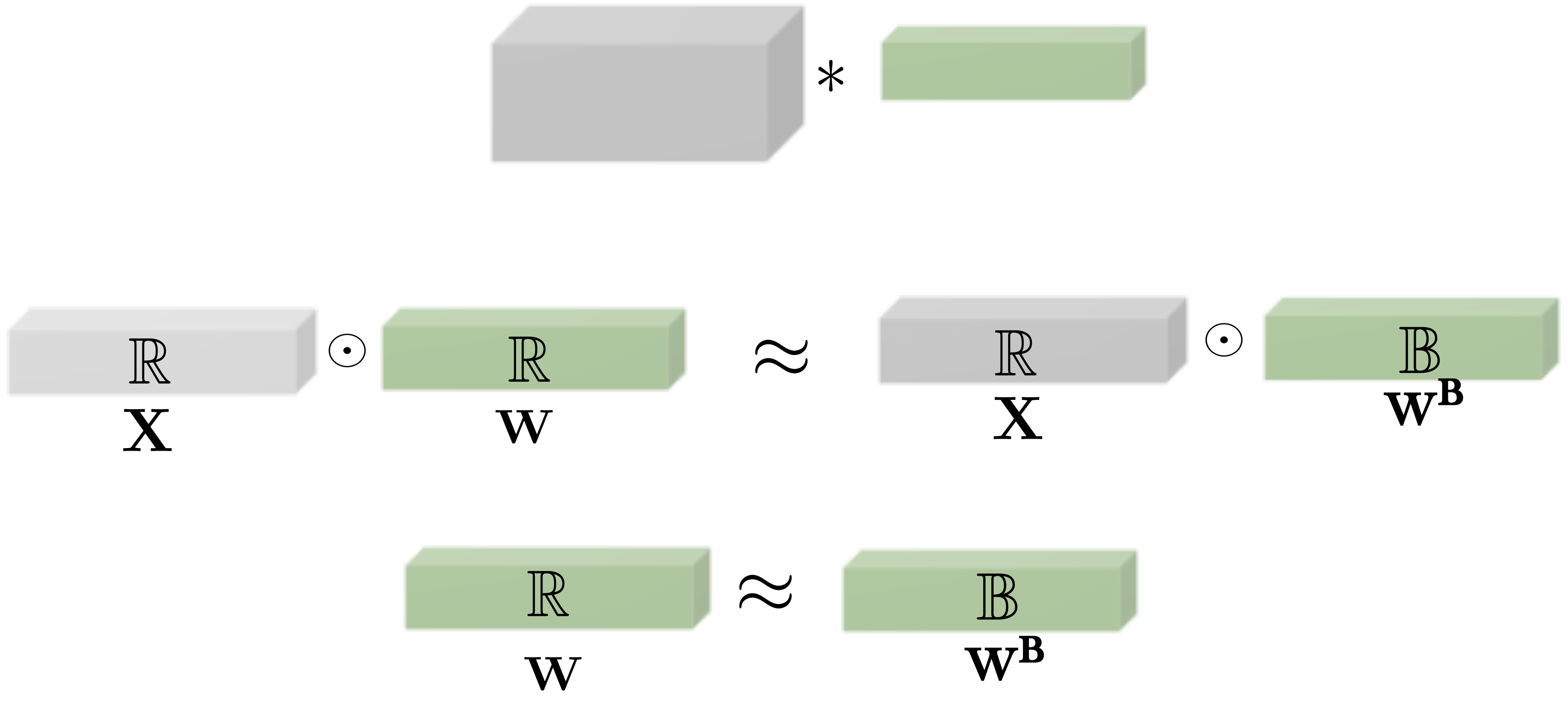
{-1,+1}	{0,1}
MUL	XNOR
ADD, SUB	Bit-Count (popcount)

Why Binary?

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

- Low Power Device

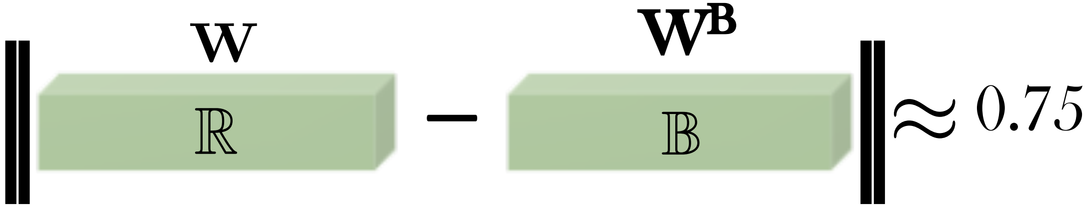
- Easy to Implement in hardware



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

Quantization Error

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



Optimal Scaling Factor

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

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

Binary Input and Binary Weight (XNOR-Net)

$$\underbrace{\overset{\mathbb{R}}{\mathbf{X}} \odot \overset{\mathbb{R}}{\mathbf{W}}}_{\mathbf{Y}} \approx \underbrace{\beta\,\alpha}_{\gamma} \underbrace{\overset{\mathbb{B}}{\mathbf{X^B}} \odot \overset{\mathbb{B}}{\mathbf{W^B}}}_{\mathbf{Y^B}}$$

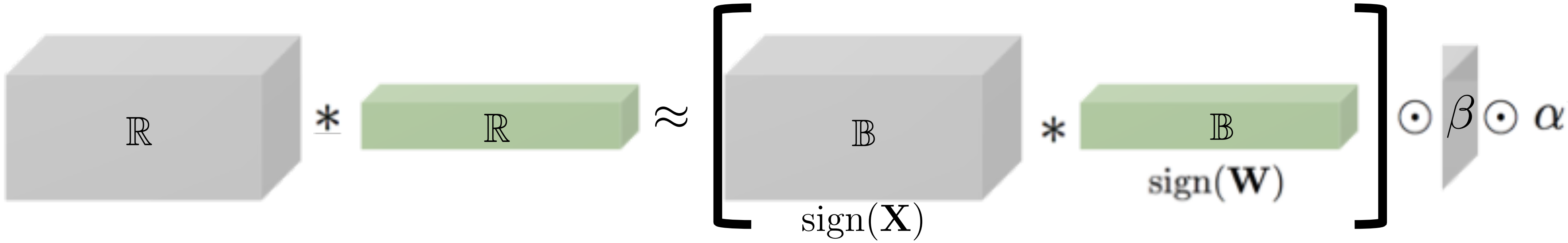
$$\mathbf{Y} \approx \gamma\ \mathbf{Y^B}$$

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

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

$$\boxed{\mathbf{X^B}^* = \mathrm{sign}(\mathbf{X}) \quad \mathbf{W^B}^* = \mathrm{sign}(\mathbf{W})} \qquad \boxed{\alpha^* = \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \qquad \beta^* = \frac{1}{n}\|\mathbf{X}\|_{\ell 1}}$$

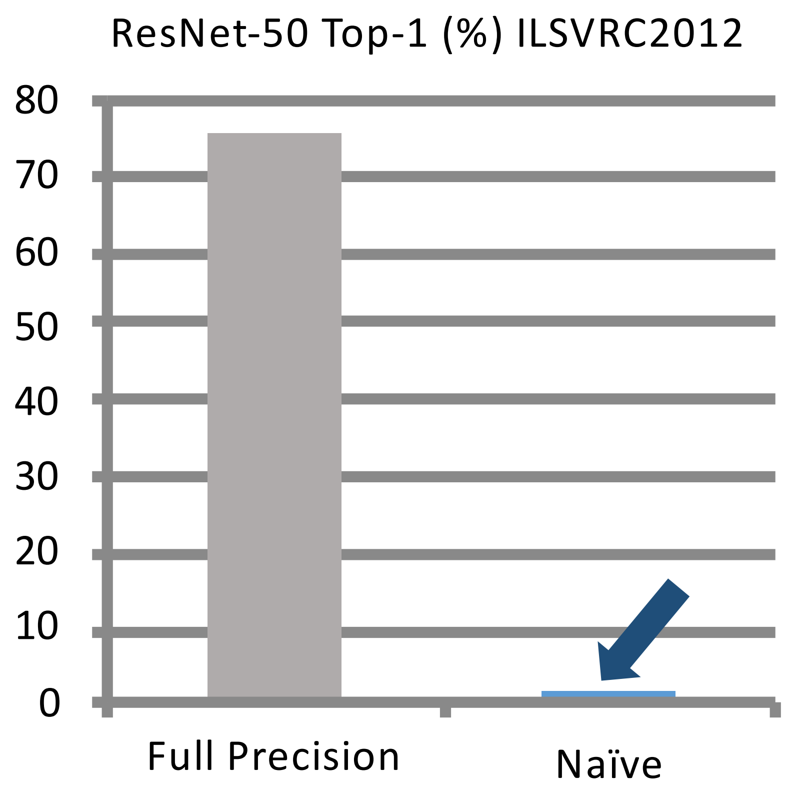
How to train a CNN with binary filters?

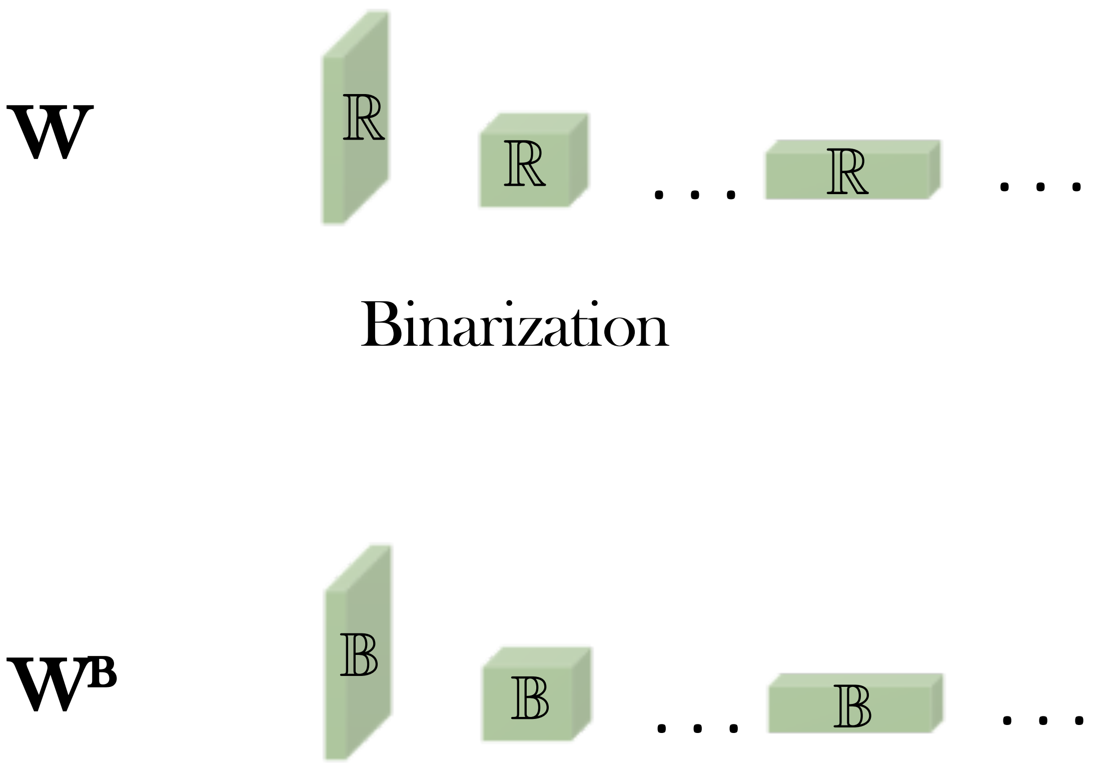


Training Binary Weight Networks

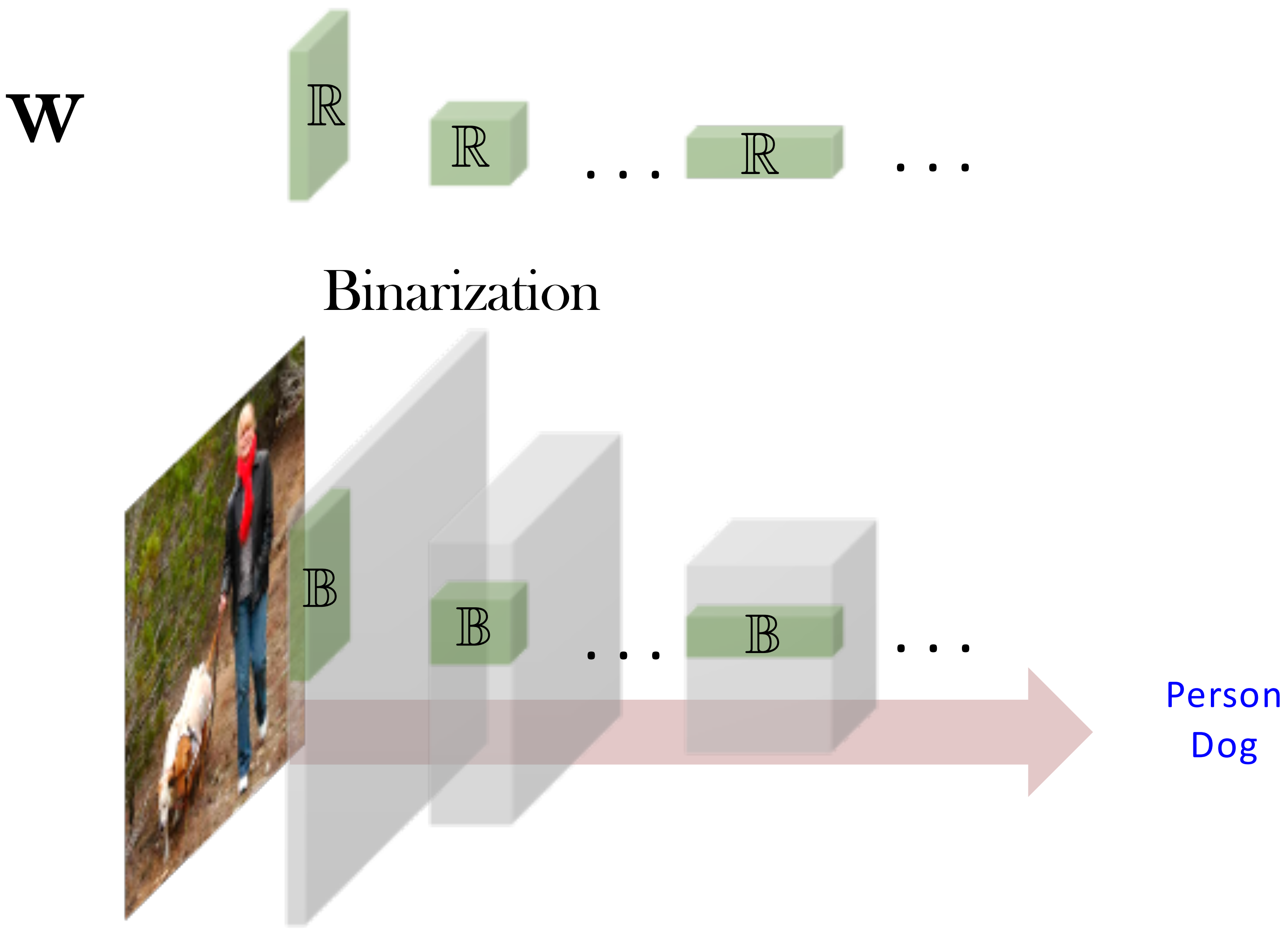
Naive Solution:

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


ResNet-50 Top-1 (%) ILSVRC2012
80
70
60
50
40
30
20
10
0
Full Precision
Naïve



W
ℝ
ℝ
. . .
ℝ
. . .
Binarization
W^B
𝔹
𝔹
. . .
𝔹
. . .



$\mathbf{W}$
$\mathbb{R}$
$\mathbb{R}$
. . .
$\mathbb{R}$
. . .
Binarization
$\mathbb{B}$
$\mathbb{B}$
. . .
$\mathbb{B}$
. . .
Person
Dog

Binary Weight Network

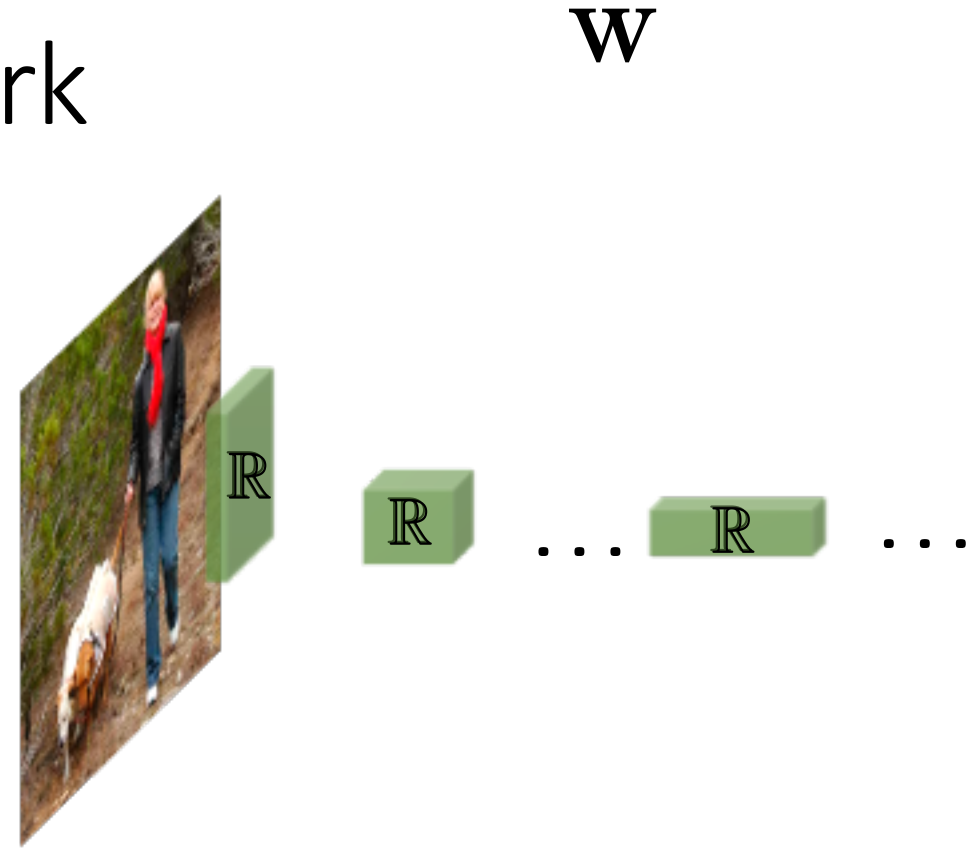
Train for binary weights:

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:

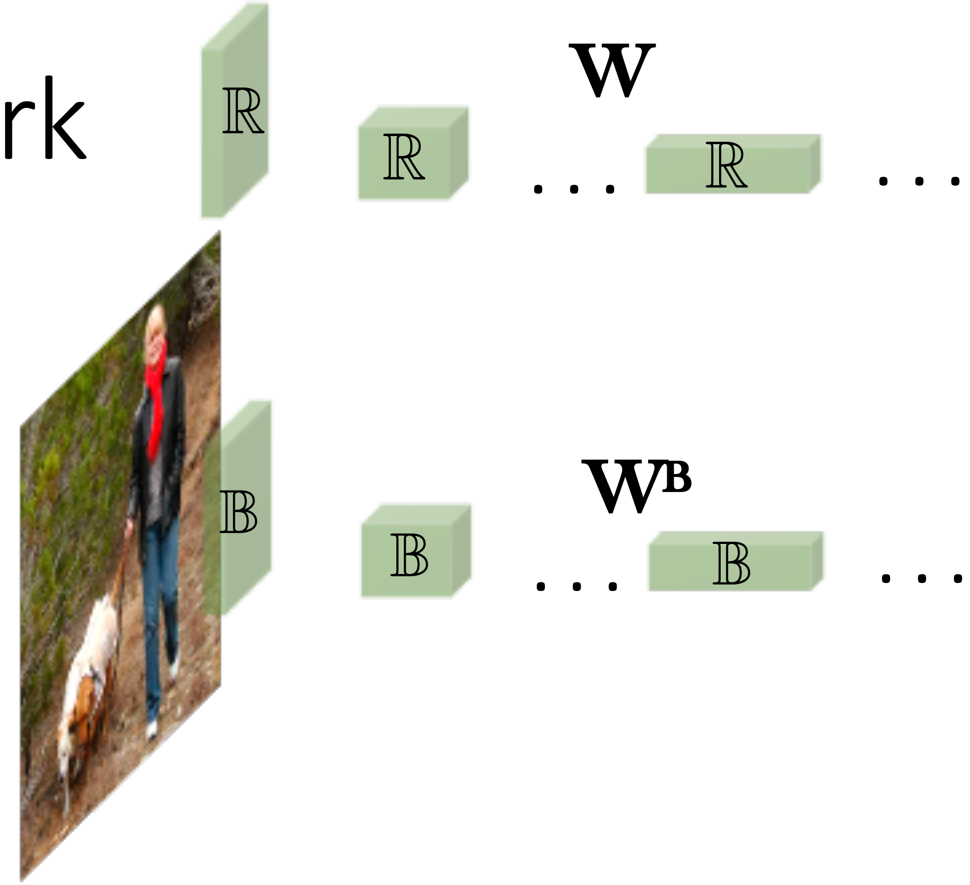
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial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



Binary Weight Network

Train for binary weights:

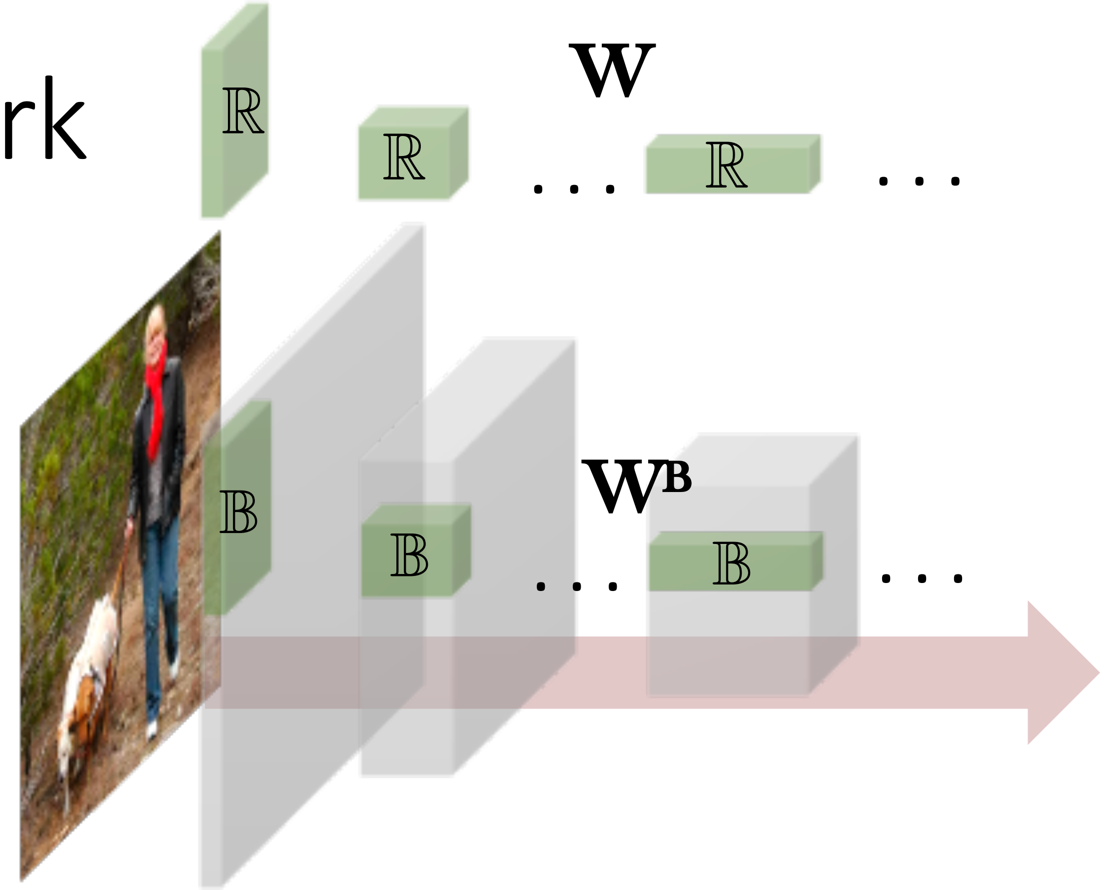
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial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



Binary Weight Network

Train for binary weights:

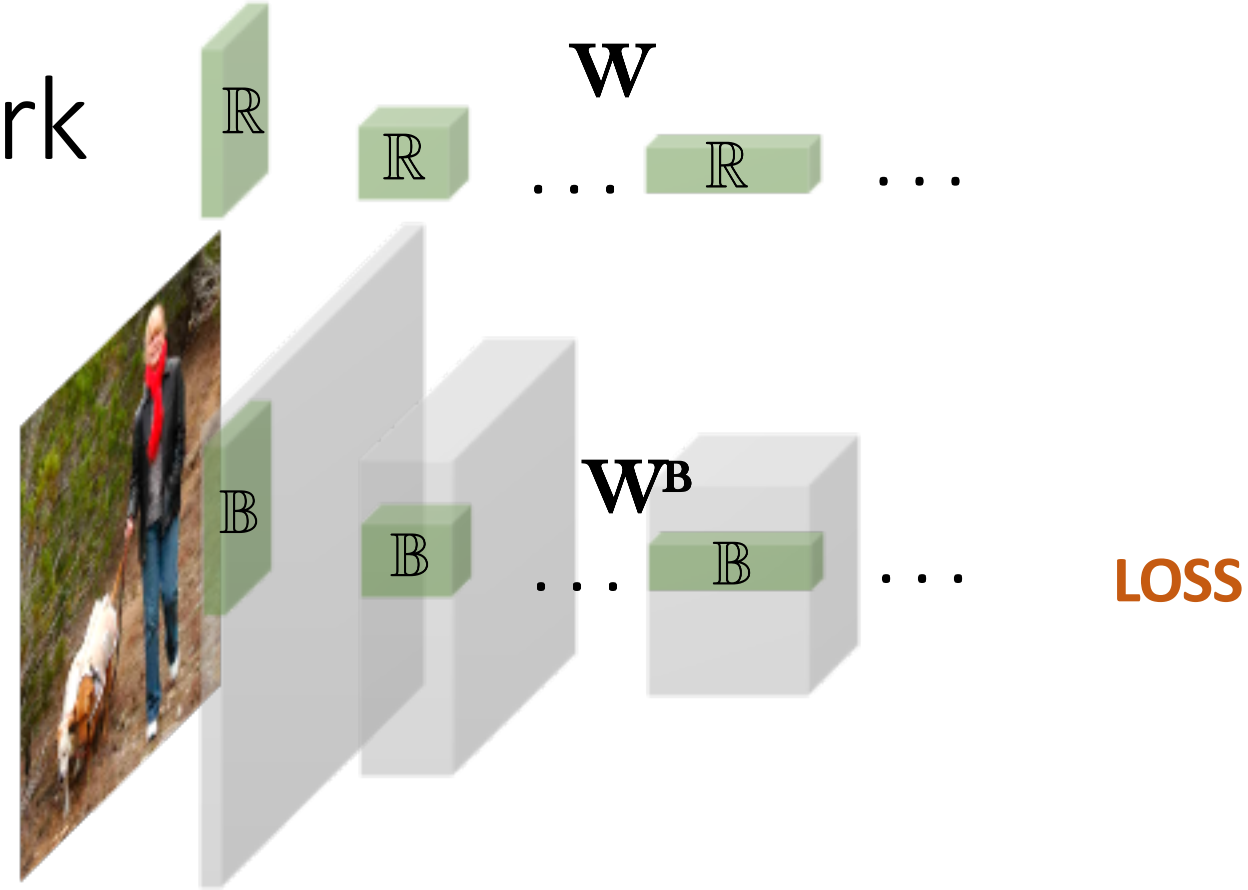
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial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



Binary Weight Network

Train for binary weights:

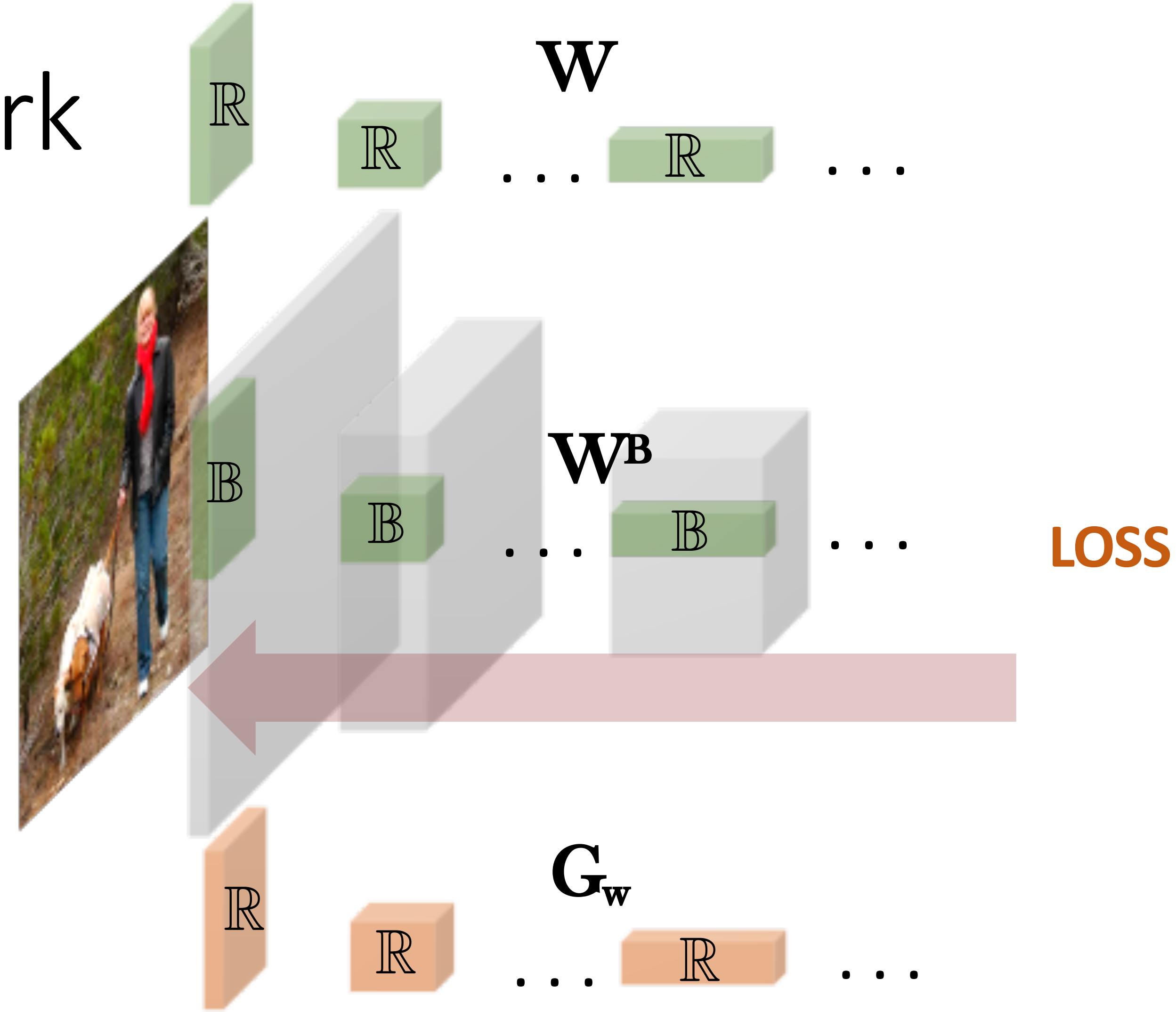
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial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



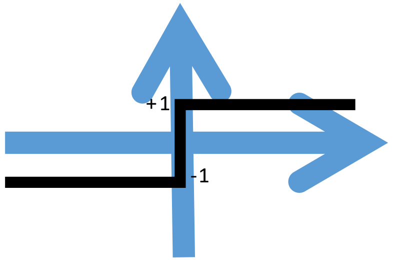
Binary Weight Network

Train for binary weights:

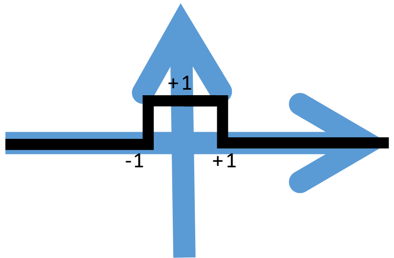
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial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



sign(x) →



Gx →

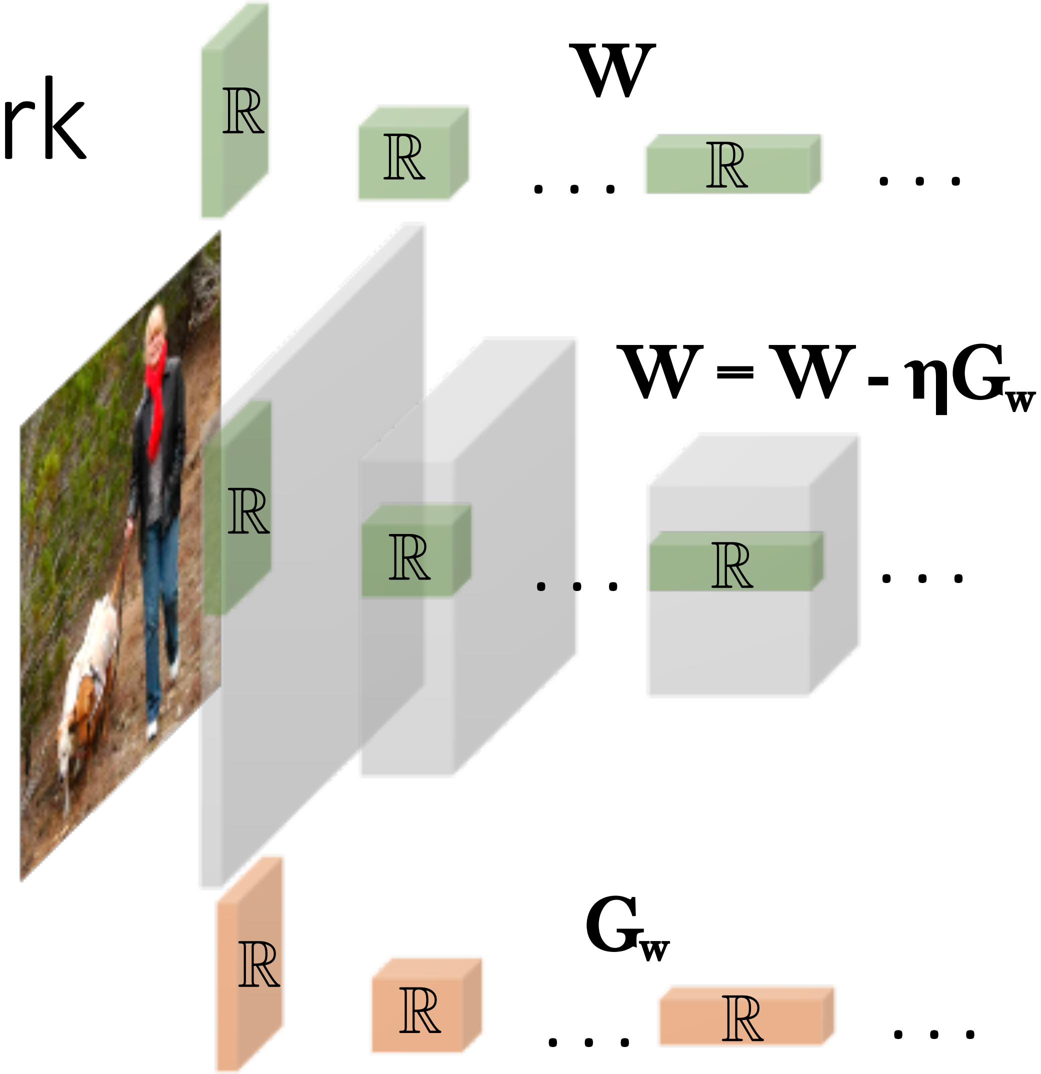


[Hinton et al. 2012]

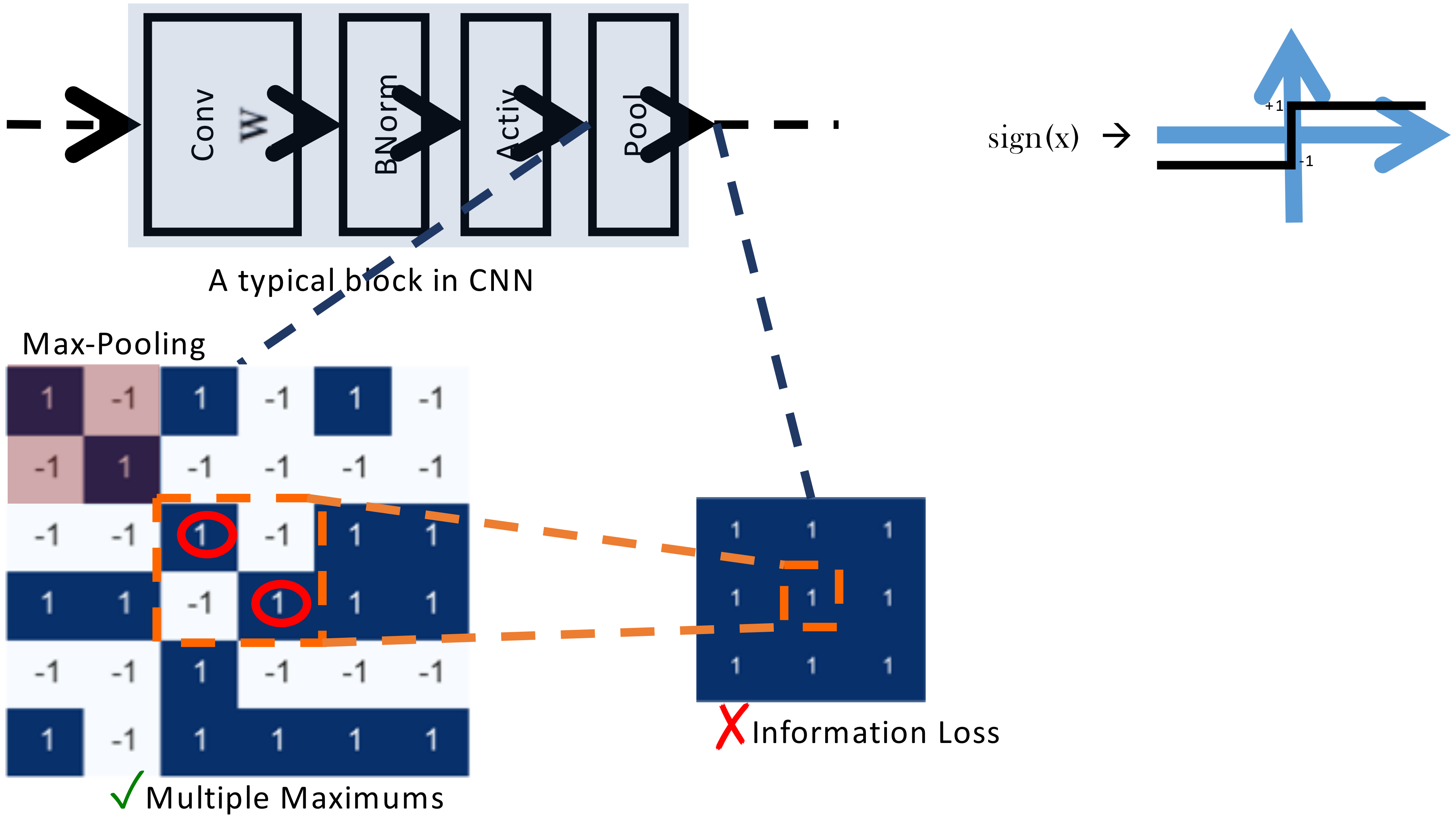
Binary Weight Network

Train for binary weights:

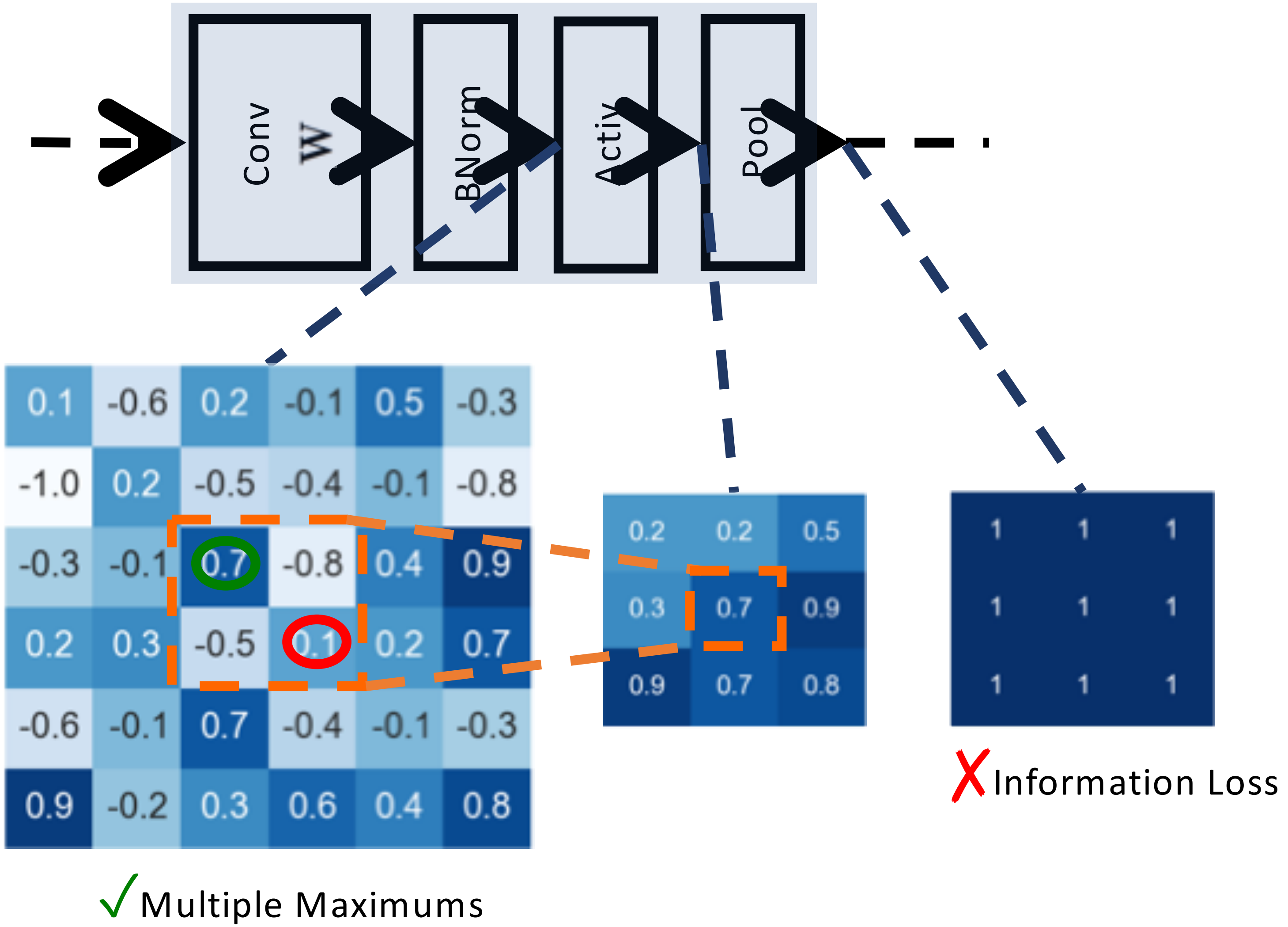
1. Randomly initialize $\mathbf{W}$
2. For $iter = 1$ to N
3. Load a random input image $\mathbf{X}$
4. $\mathbf{W^B} = \text{sign}(\mathbf{W})$
5. $\alpha = \frac{\|W\|_{\ell 1}}{n}$
6. Forward pass with $\alpha, \mathbf{W^B}$
7. Compute loss function $\mathbf{C}$
8. $\frac{\partial \mathbf{C}}{\partial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



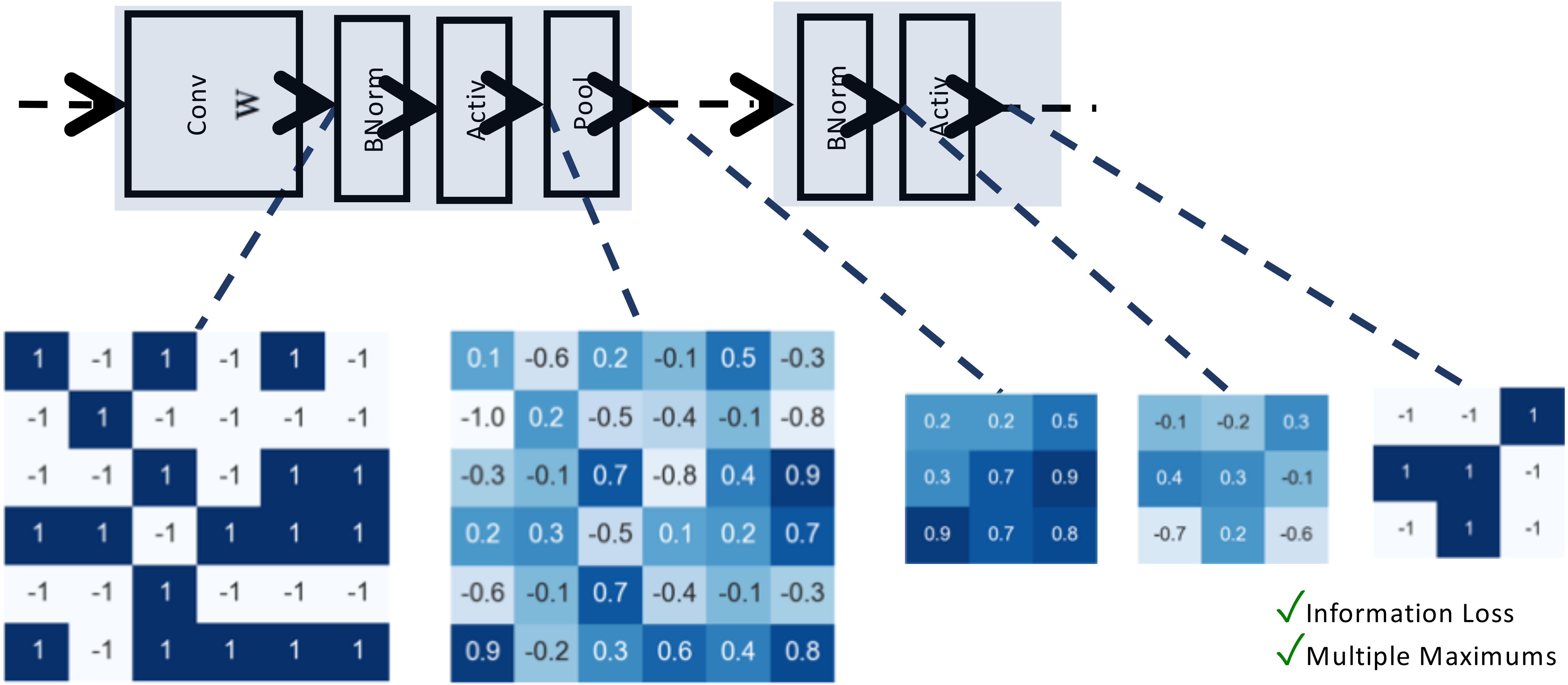
Network Structure in XNOR-Networks

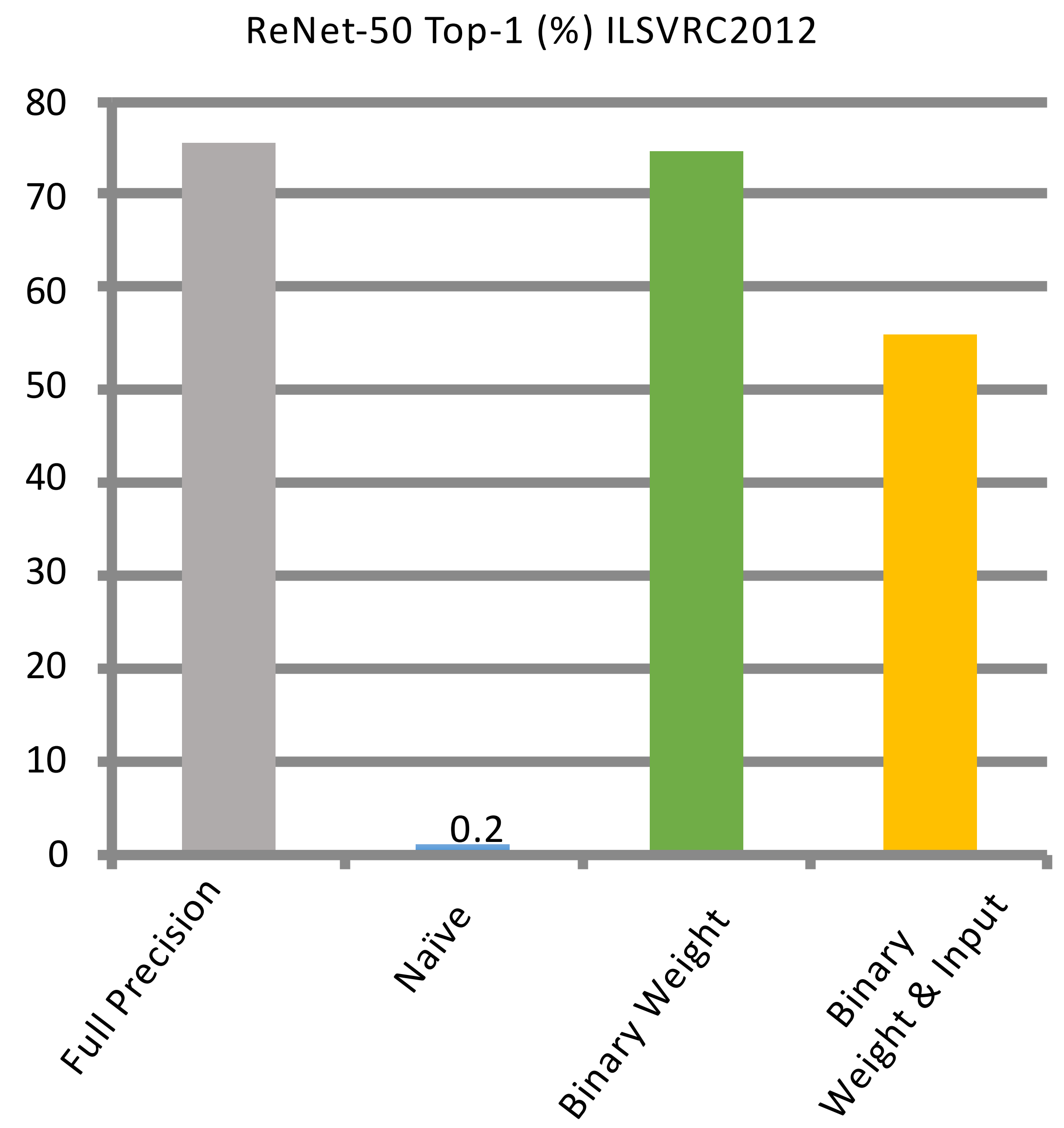


Network Structure in XNOR-Networks



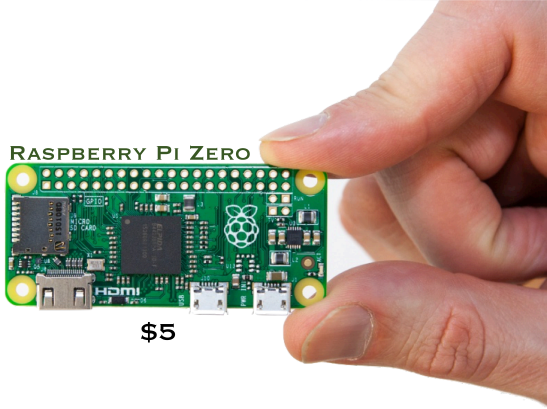
Network Structure in XNOR-Networks

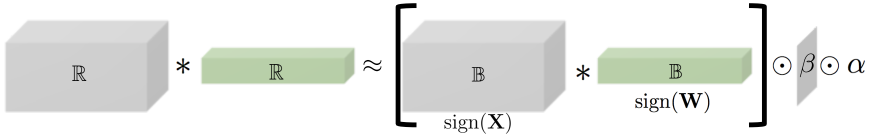




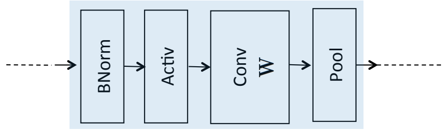
[XNOR-Networks, Rastegari et al, ECCV2016]

XNOR.AI

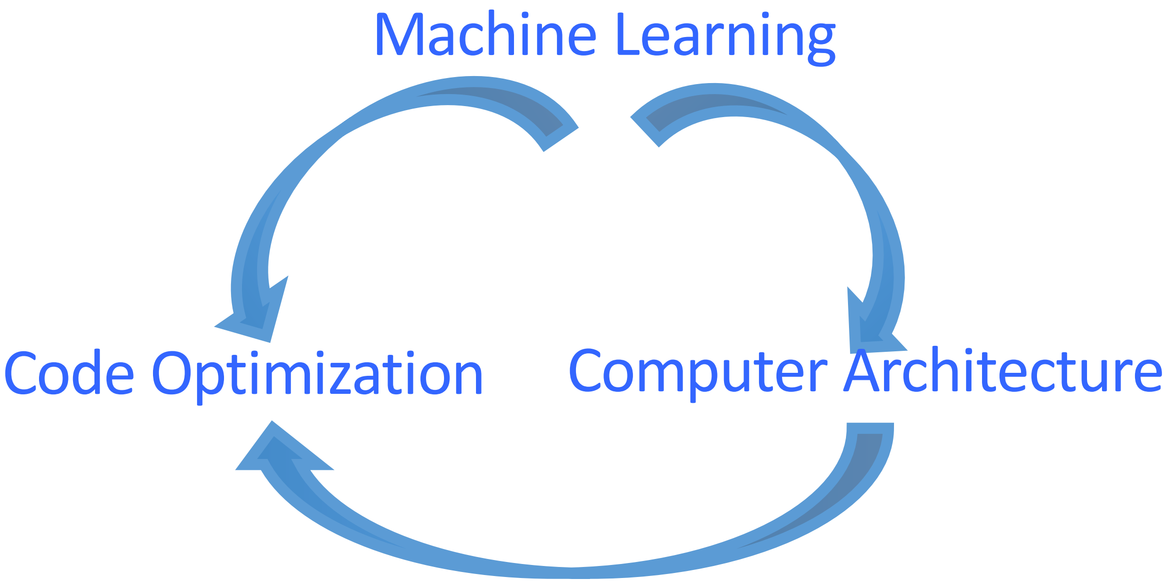


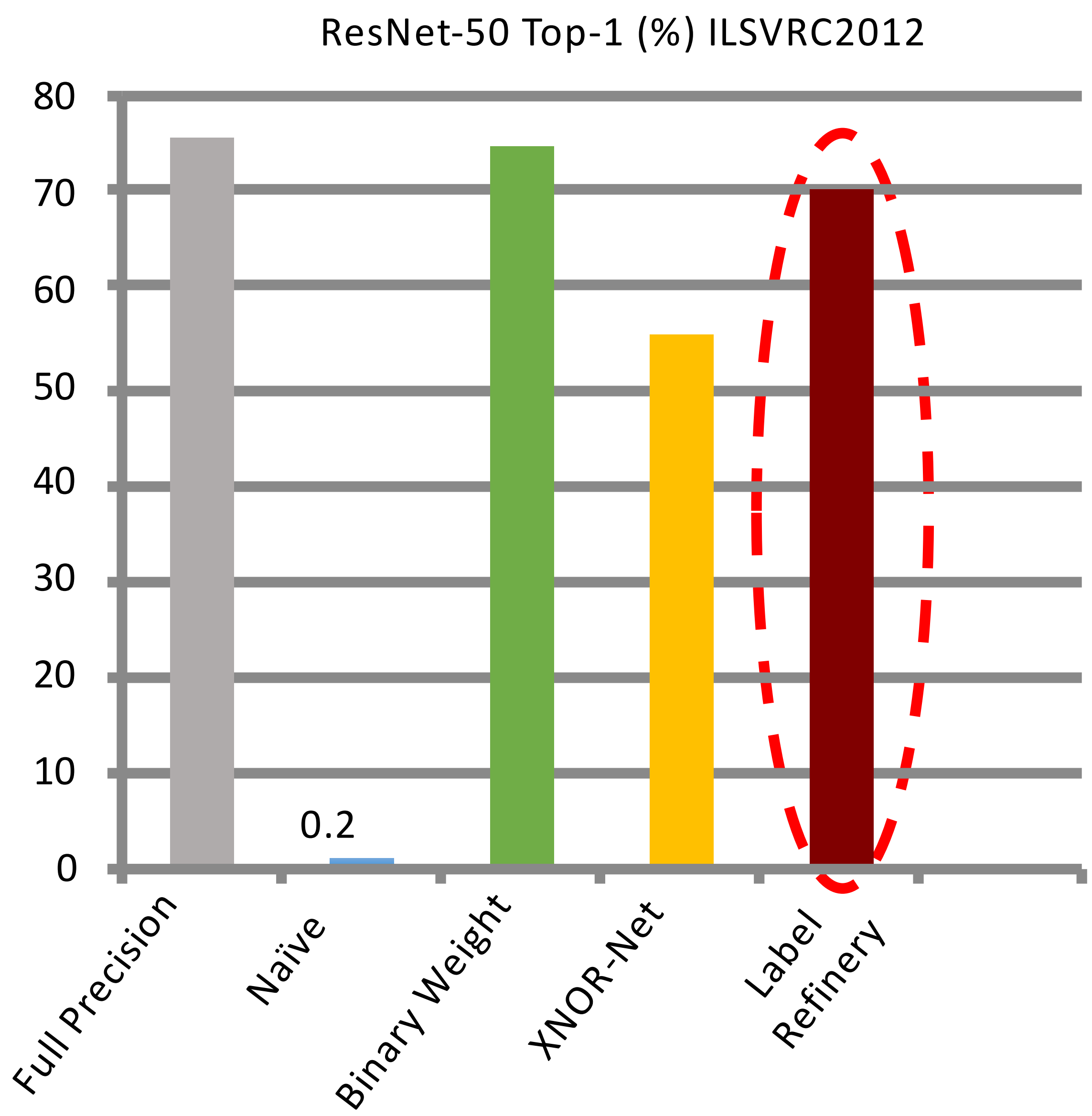


Randomly initialize $\mathbf{W}$
For $iter = 1$ to N
Load a random input image $\mathbf{X}$
$\mathbf{W}^{\mathbf{B}} = \text{sign}(\mathbf{W})$
$\alpha = \frac{\|W\|_{\ell 1}}{n}$
Forward pass with $\alpha, \mathbf{W}^{\mathbf{B}}$
Compute loss function $\mathbf{C}$
$\frac{\partial \mathbf{C}}{\partial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



Xnor.ai IP



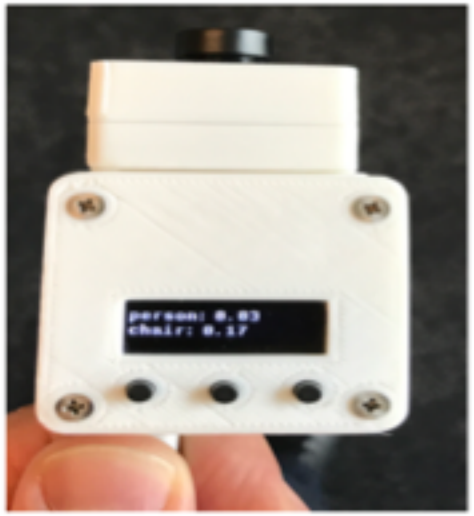


ResNet-50 Top-1 (%) ILSVRC2012
80
70
60
50
40
30
20
10
0
0.2
Full Precision
Naïve
Binary Weight
XNOR-Net
Label Refinery

State-of-the-Art AI: all the way to Pi Zero

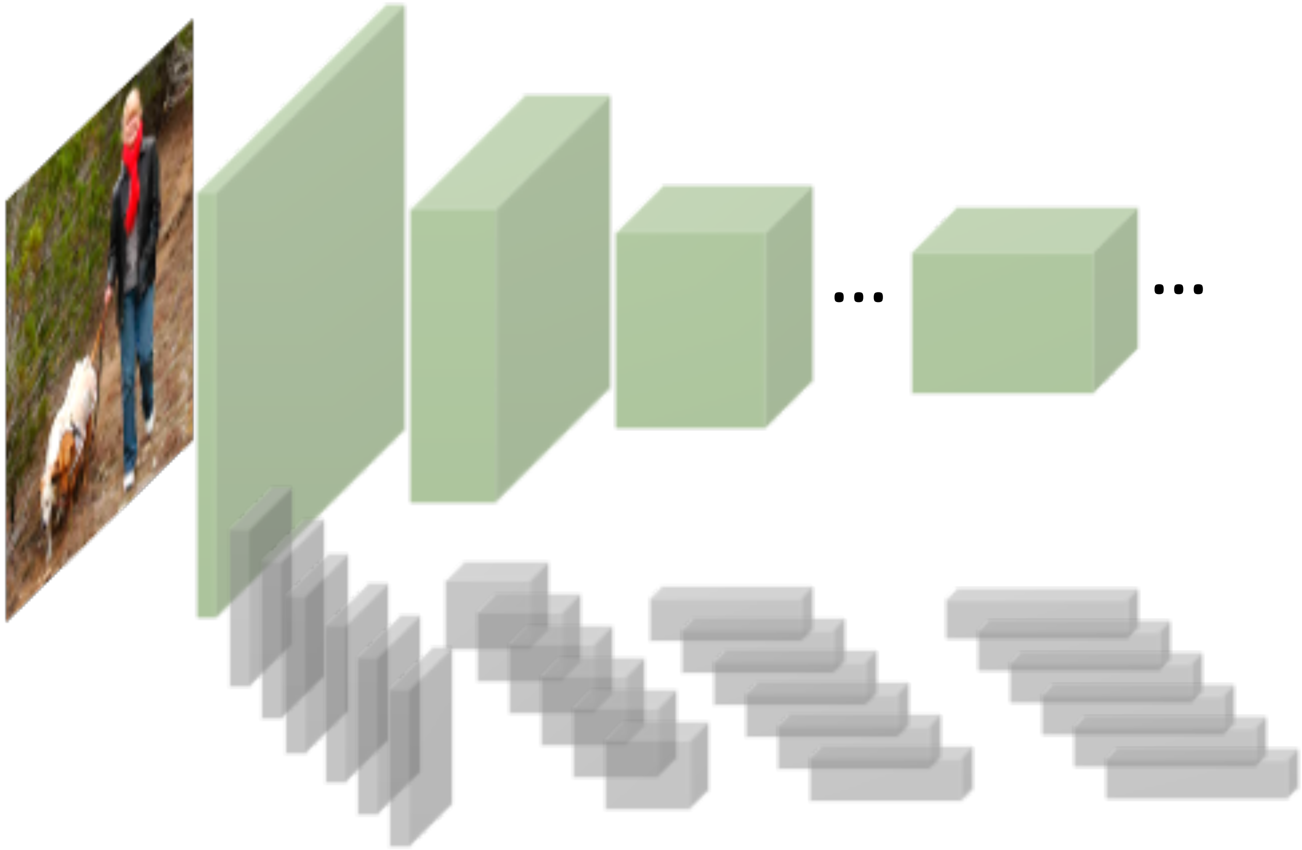
XNOR $5 deep learning machine…
… on Raspberry Pi Zero

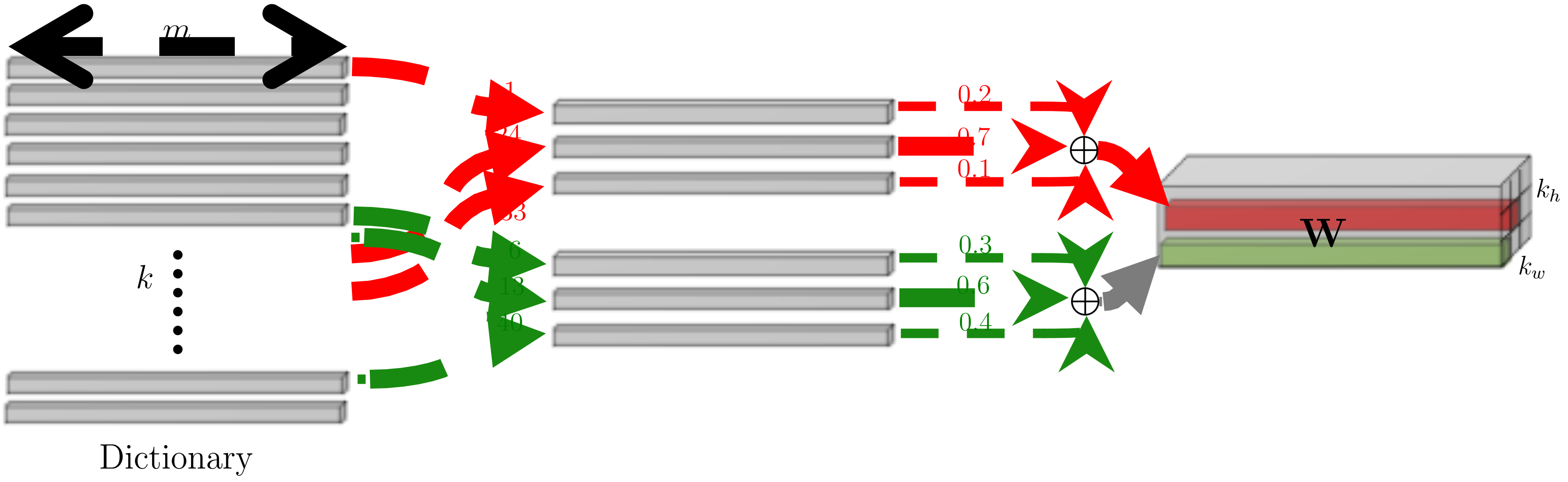
Modular AI at Edge



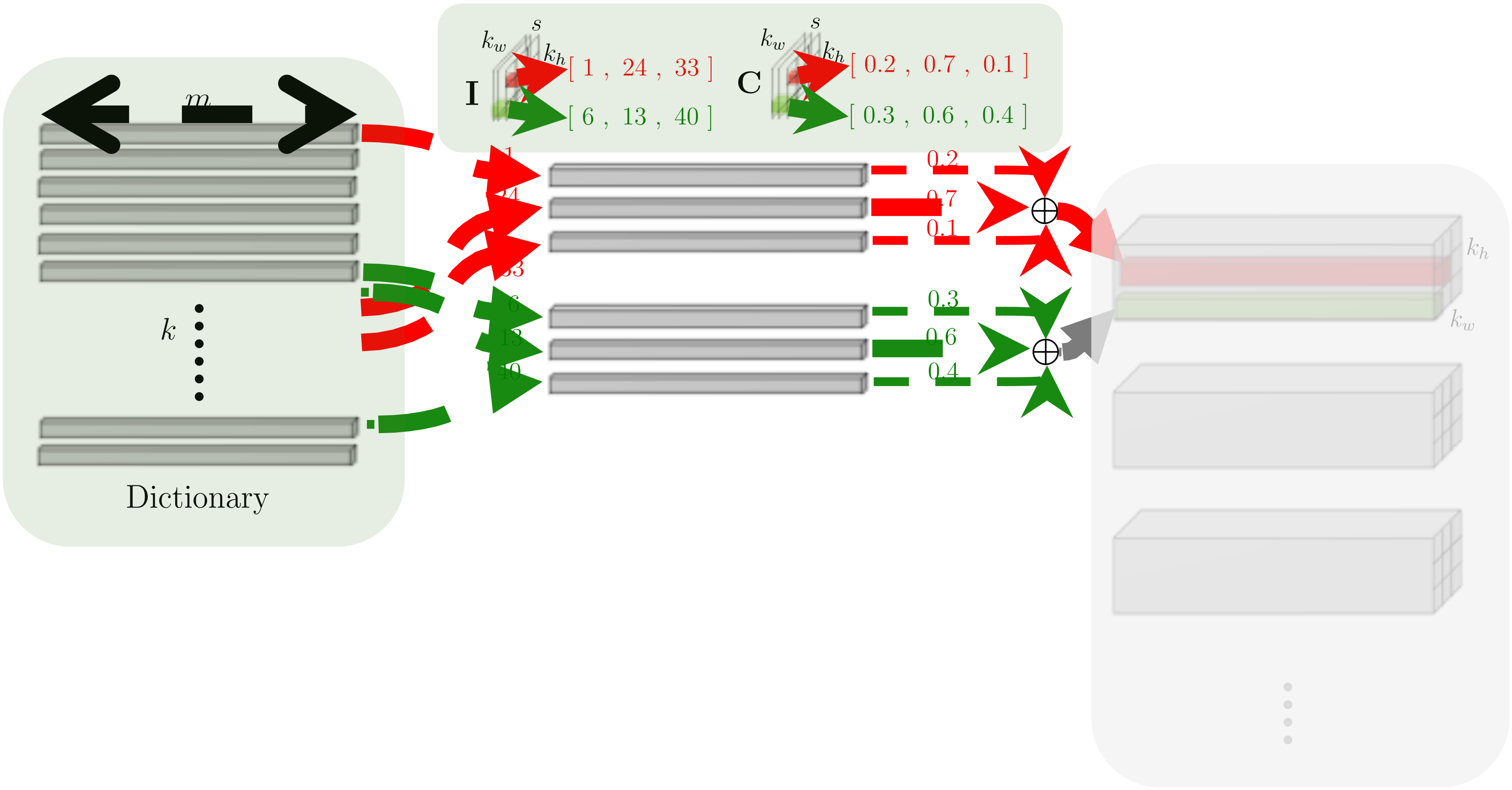
Lookup Based CNN

[LCNN Bagherinezhad et al, CVPR 2017]

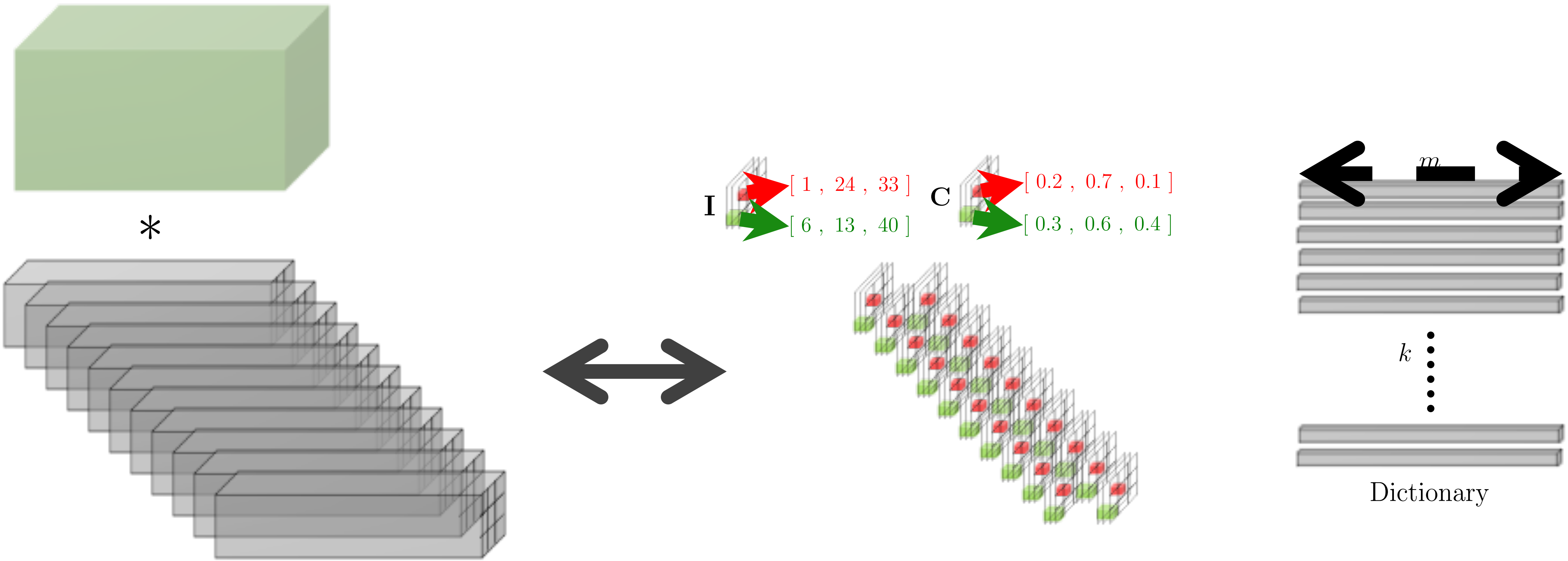




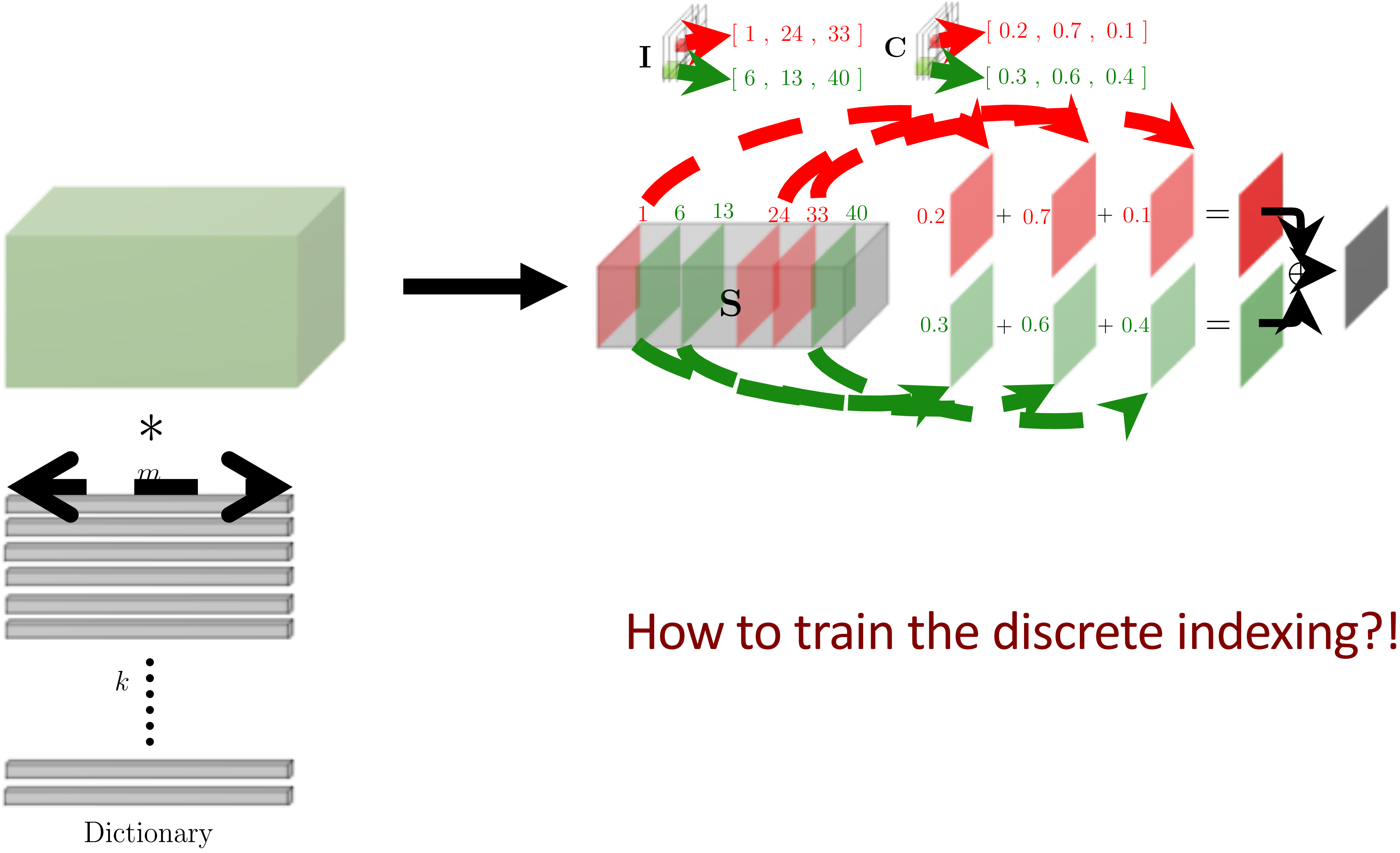
m
k
Dictionary
0.2
0.7
0.1
0.3
0.6
0.4
W
k_h
k_w



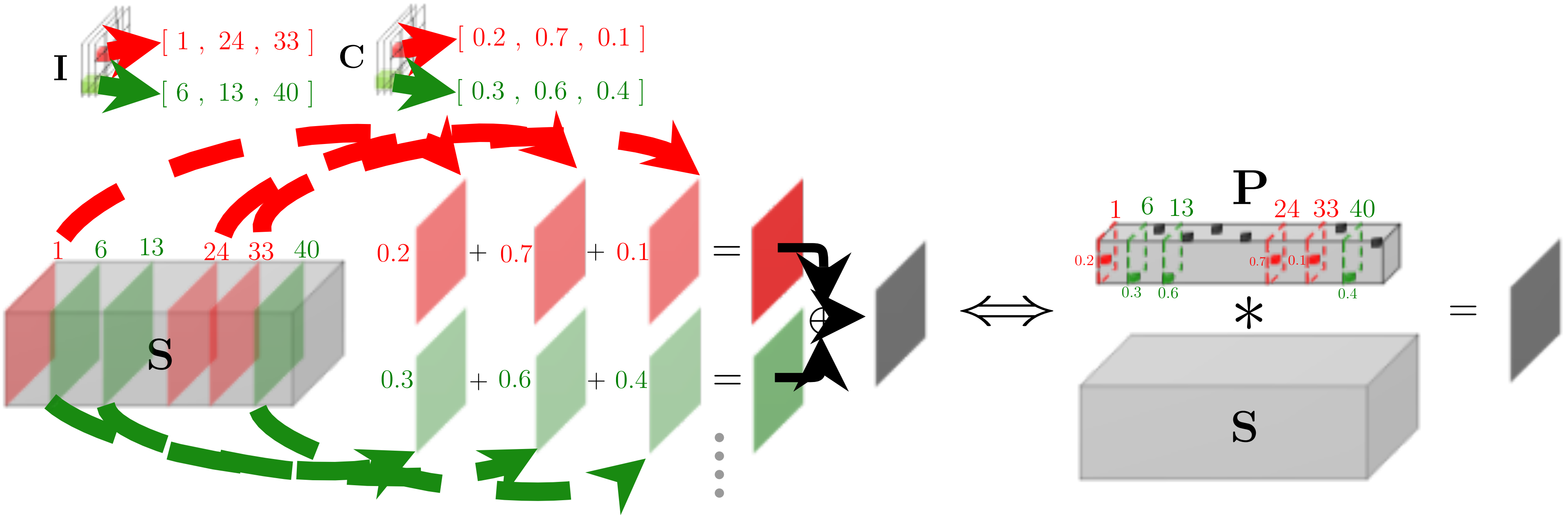
m
k
Dictionary
I
C
[1 , 24 , 33]
[6 , 13 , 40]
[0.2 , 0.7 , 0.1]
[0.3 , 0.6 , 0.4]

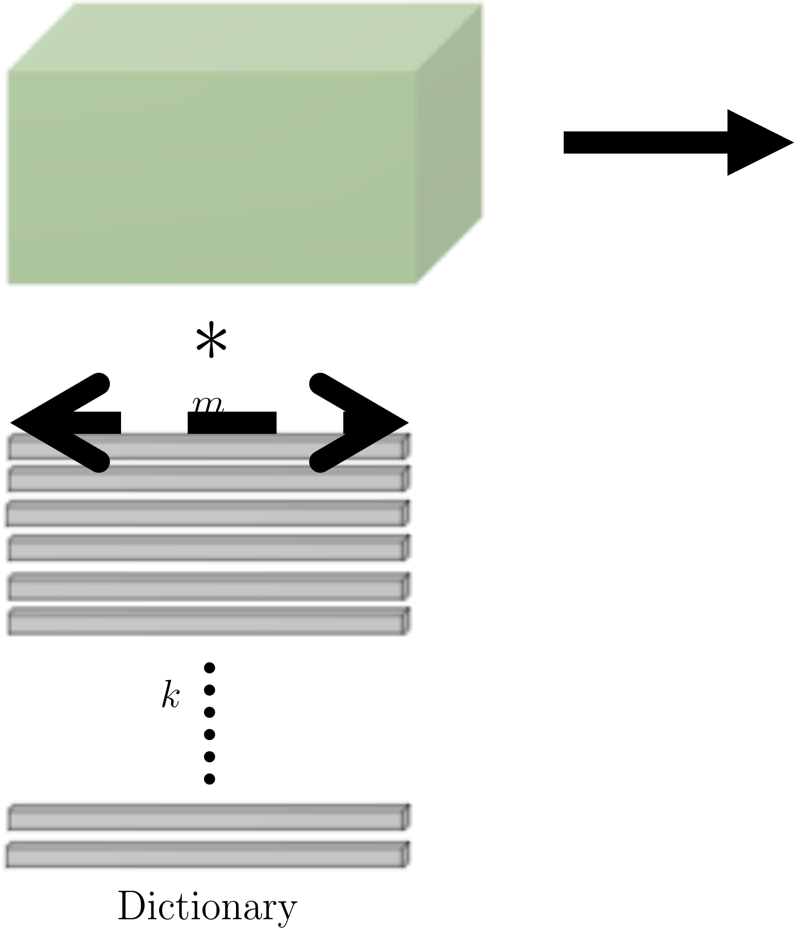


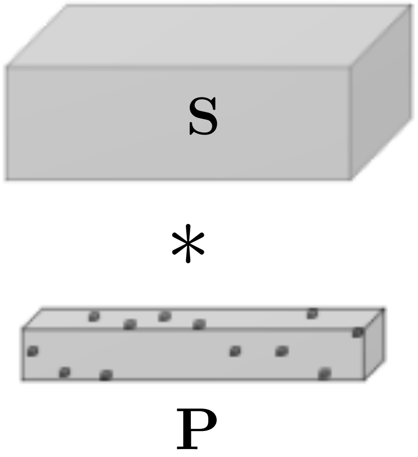
*
I
[1 , 24 , 33]
[6 , 13 , 40]
C
[0.2 , 0.7 , 0.1]
[0.3 , 0.6 , 0.4]
m
k
Dictionary



How to train the discrete indexing?!!!!


I
[1 , 24 , 33]
[6 , 13 , 40]
C
[0.2 , 0.7 , 0.1]
[0.3 , 0.6 , 0.4]
1 6 13 24 33 40
S
0.2 + 0.7 + 0.1 =
0.3 + 0.6 + 0.4 =
P
0.2 0.7 0.1
0.3 0.6 0.4
S





$$\frac{\partial(L + \lambda \|\mathbf{P}\|_{\ell_1})}{\partial \mathbf{P}} = \frac{\partial L}{\partial \mathbf{P}} + \lambda\, \mathrm{sign}(\mathbf{P})$$

ImageNet Classification Result

	AlexNet		
Model	speedup	Top-1	Top-5
CNN	1.0x	56.6	80.2
XNOR-Net[2]	8.0x	44.2	69.2
LCNN-fast	**37.6x**	44.3	68.7
LCNN-accurate	3.2x	**55.1**	**78.1**

	ResNet-18		
Model	speedup	Top-1	Top-5
CNN	1.0x	69.3	90.0
XNOR-Net[2]	10.6x	51.2	73.2
LCNN-fast	**29.2x**	51.8	76.8
LCNN-accurate	5x	**62.2**	**84.6**

How to improve the accuracy of compact models?

Components in a Supervised Learning System

- Data
  - ImageNet, MSCOCO, SUN, …
  - Data Augmentations
- Model
  - SVM, CNN
  - Optimization Techniques (SGD,ADAM, RMSProp,…)
- Label
  - ?!!

Challenges with current labeling paradigm

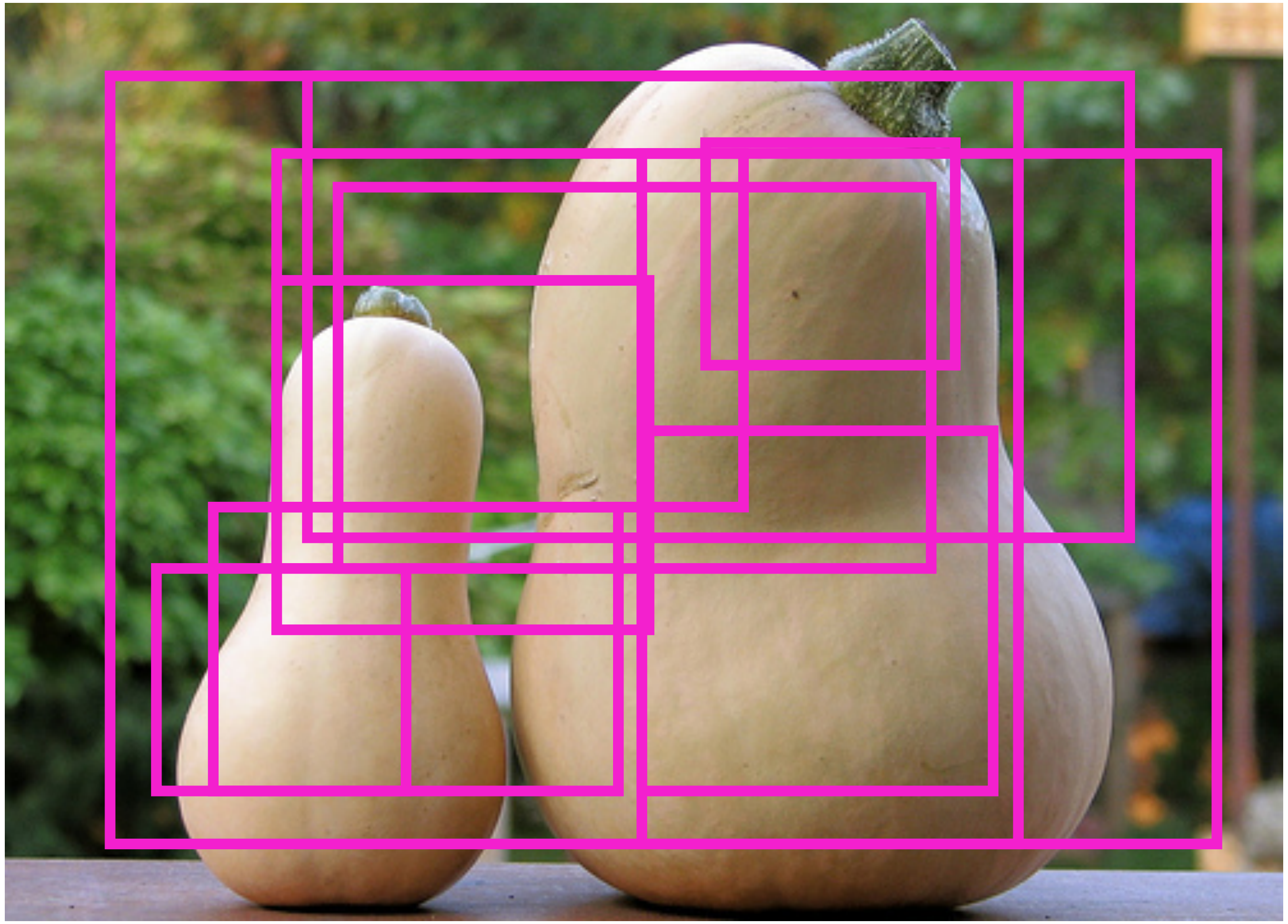
- Incomplete

Persian Cat

ball

Challenges with current labeling paradigm

- Random cropping



Challenges with current labeling paradigm

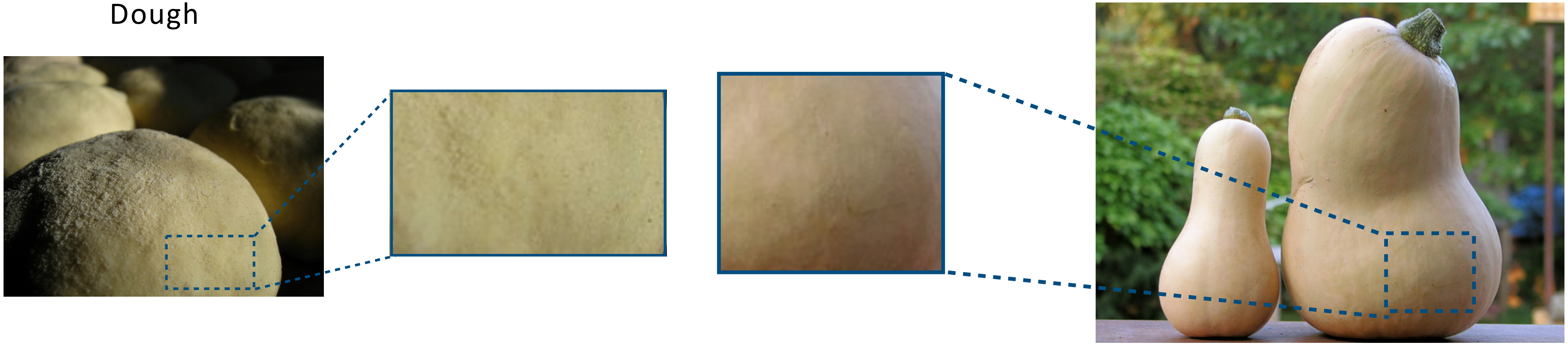
- Inconsistent

Challenges with current labeling paradigm

- Inconsistent

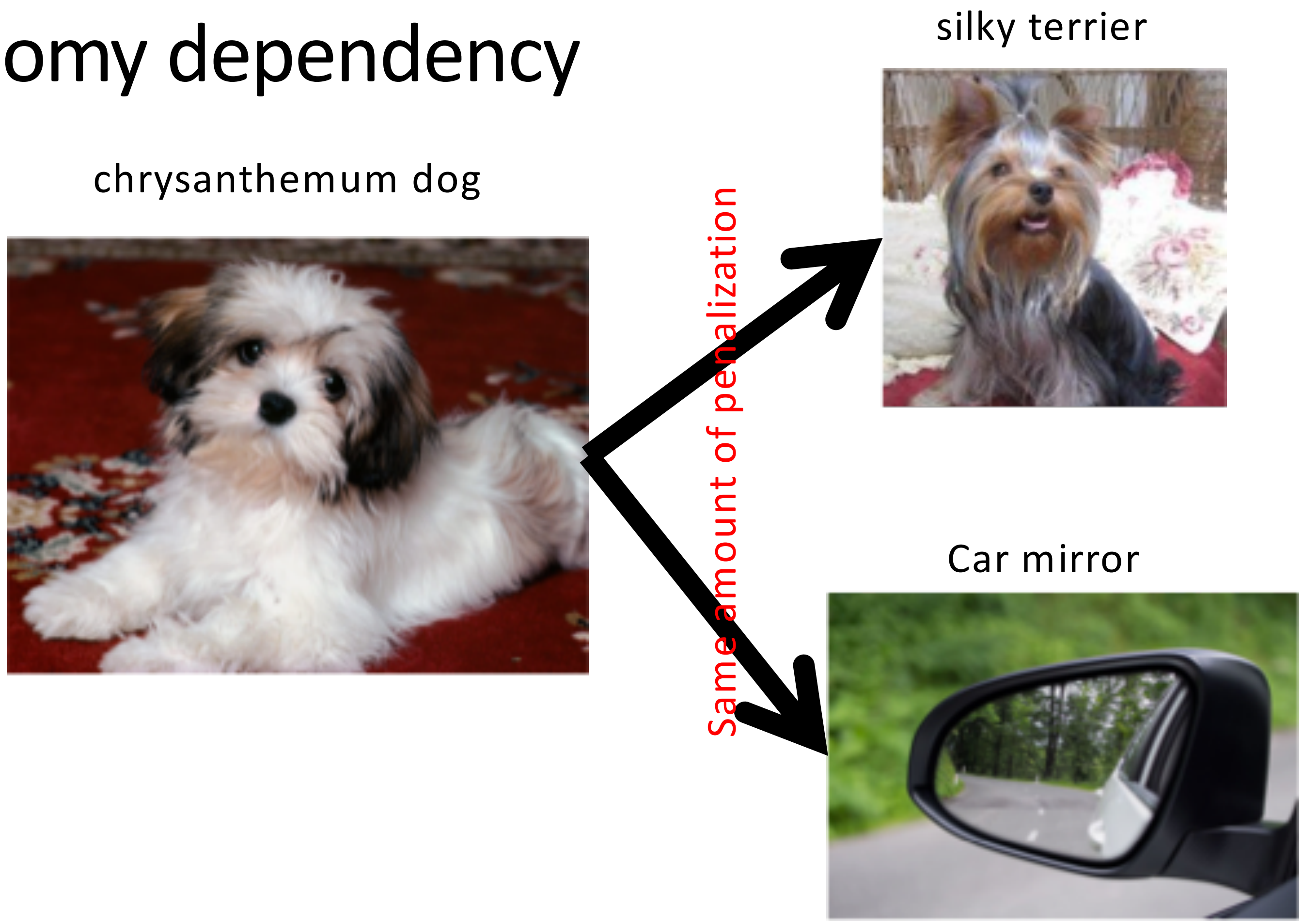
Butternut Squash

Dough



Challenges with current labeling paradigm

- Taxonomy dependency



Labels should be:

- Soft

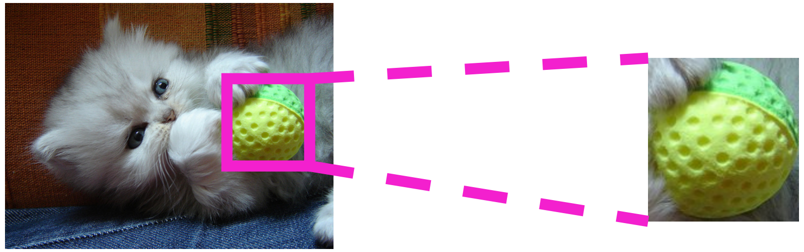
Cat → 80%
Ball → 20%

- Informative

Dog --> 60%
Cat --> 10%
Bear --> 30%

Dog --> 60%
Cat --> 30%
Bear --> 10%

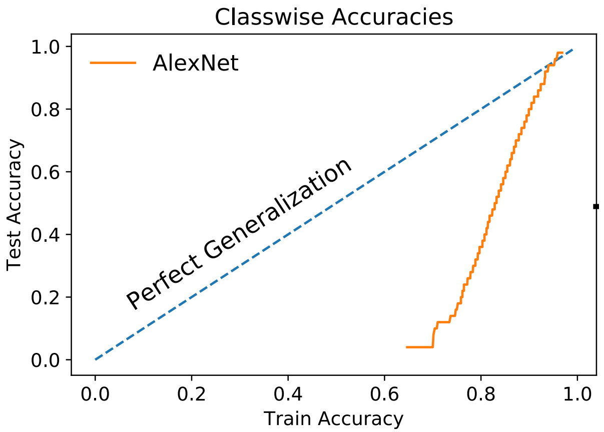
- Dynamic



Cat → 1 %
Ball → 99%

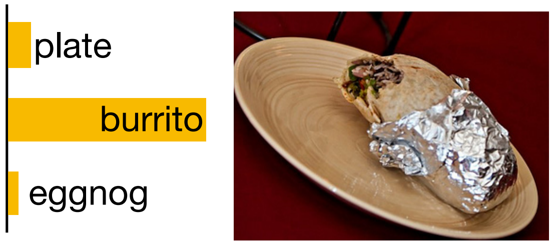
Label Refinery
[Bagherinezhad et al, 2018]

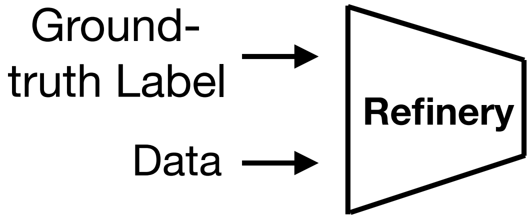
Top-1: 57.93



Label Refinery

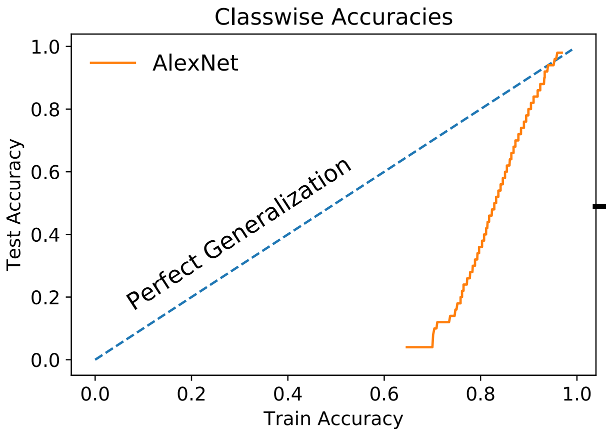
[Bagherinezhad et al, 2018]

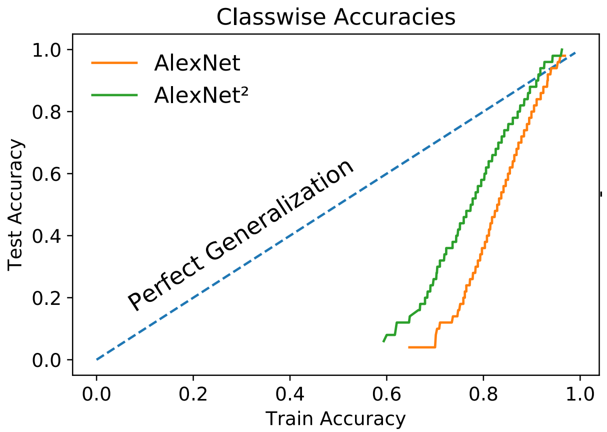




Top-1: 57.93

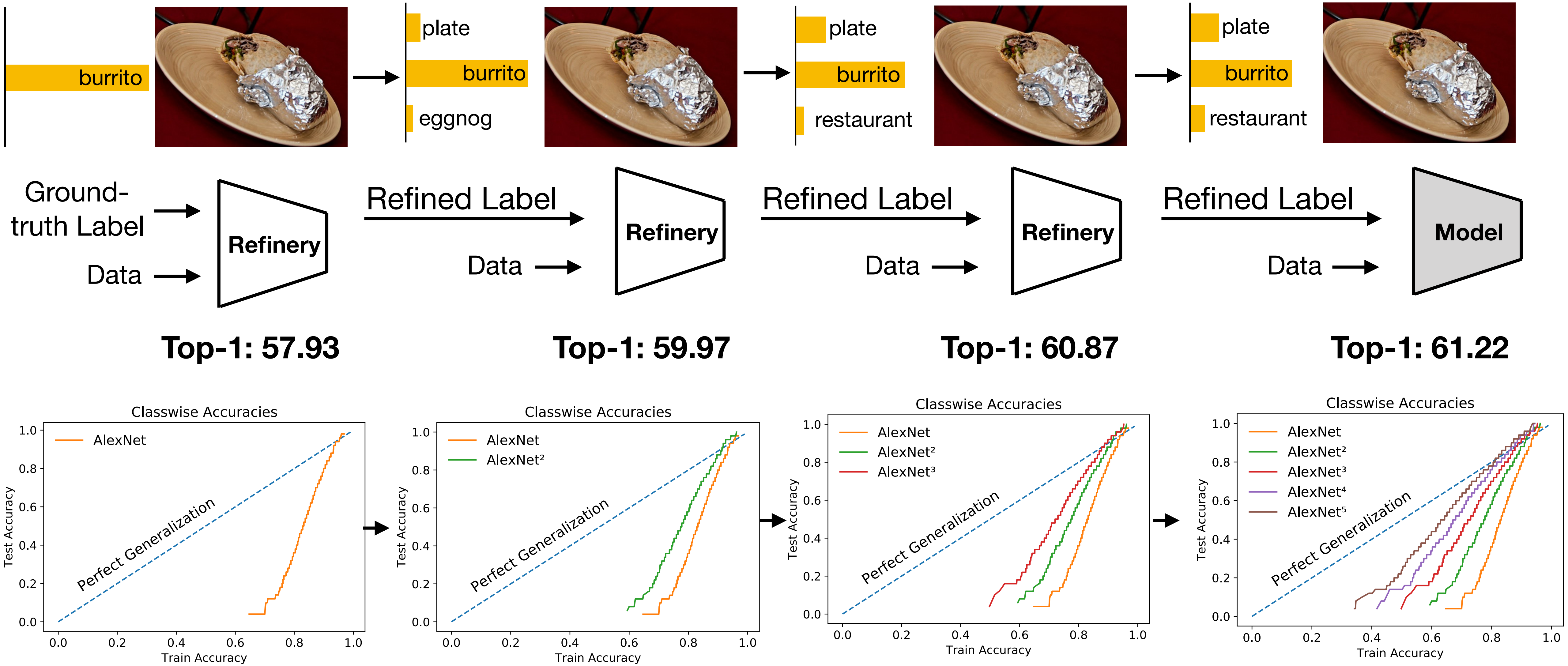
Top-1: 59.97

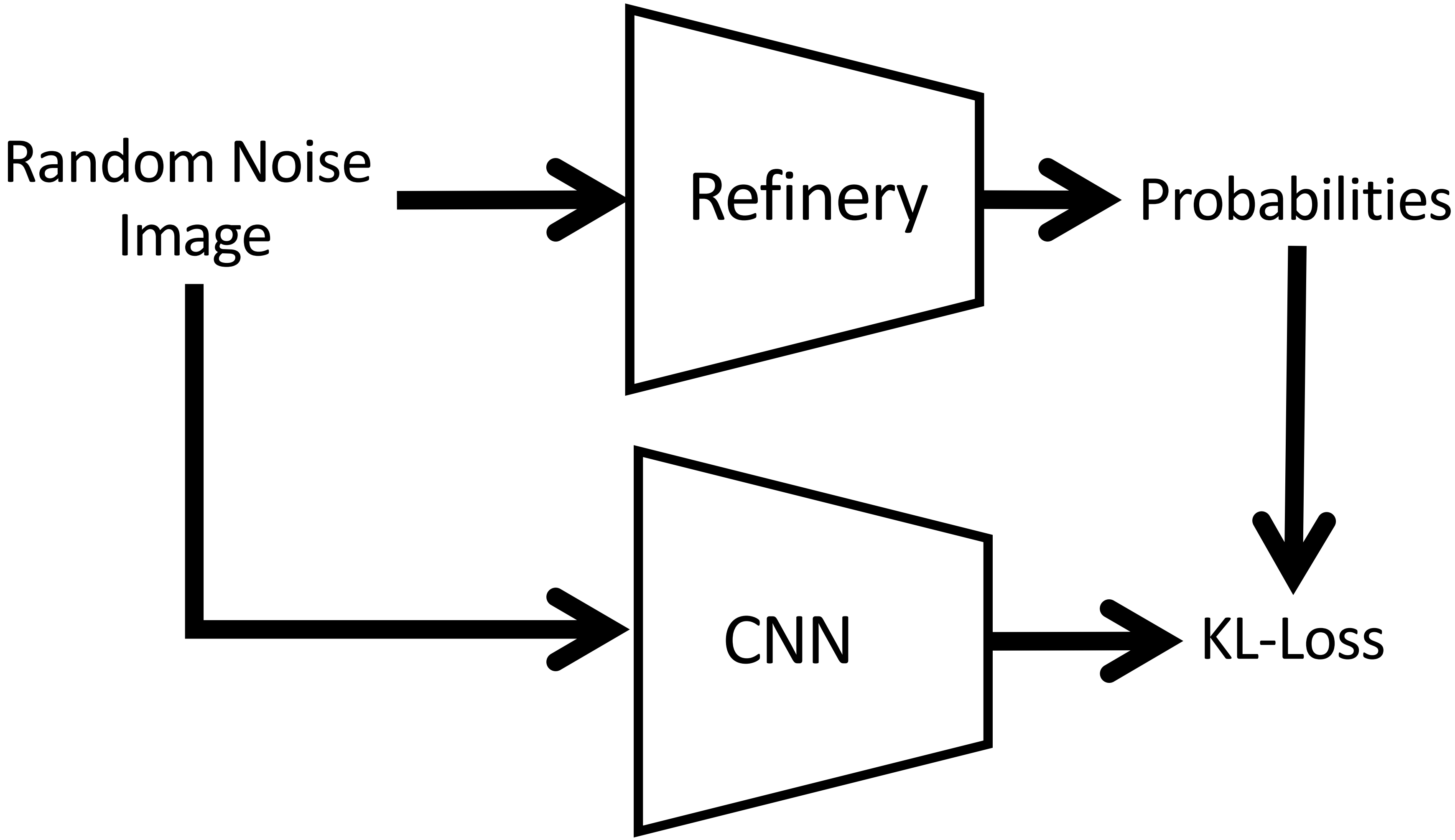


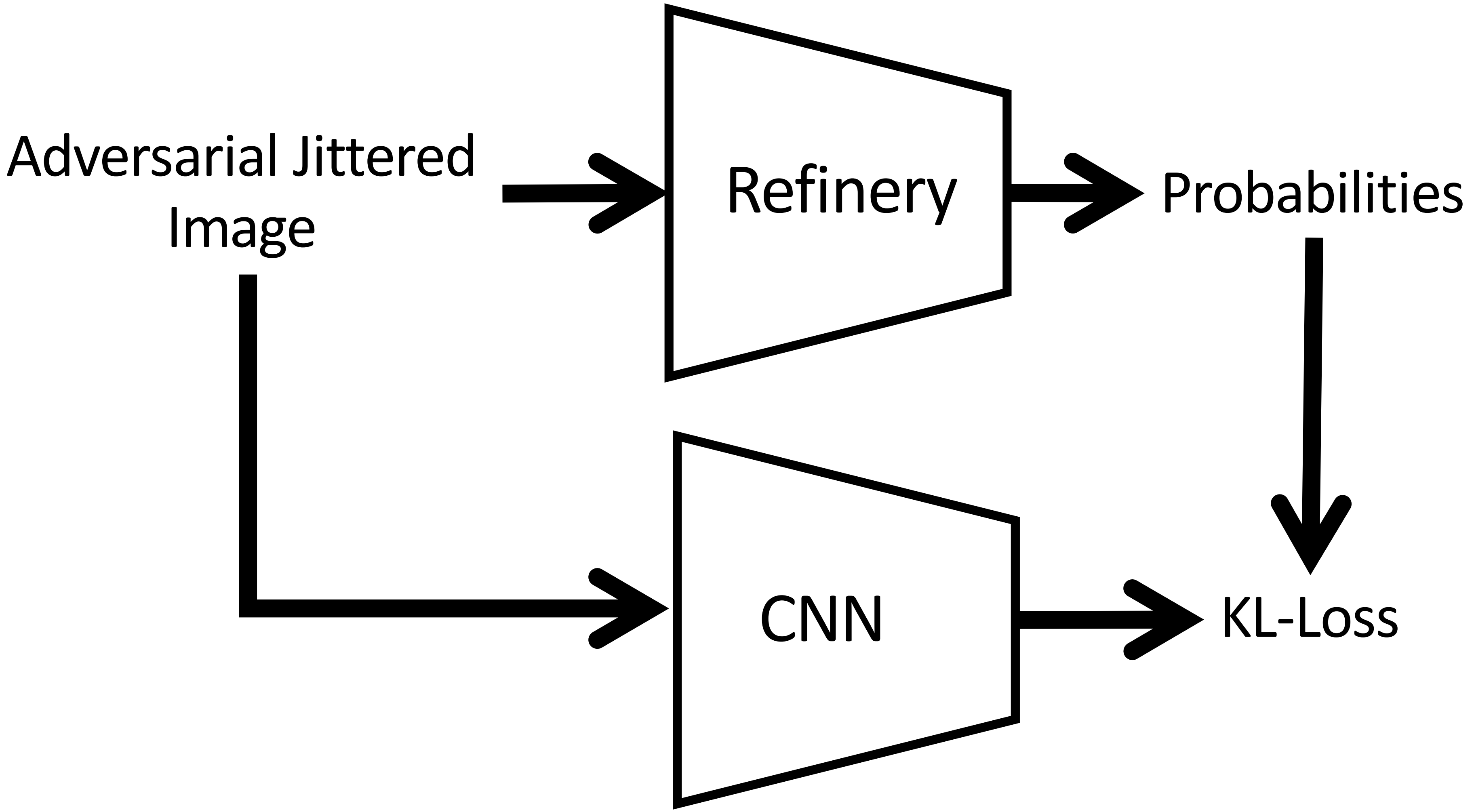


Label Refinery

[Bagherinezhad et al, 2018]




Random Noise Image
Refinery
Probabilities
CNN
KL-Loss

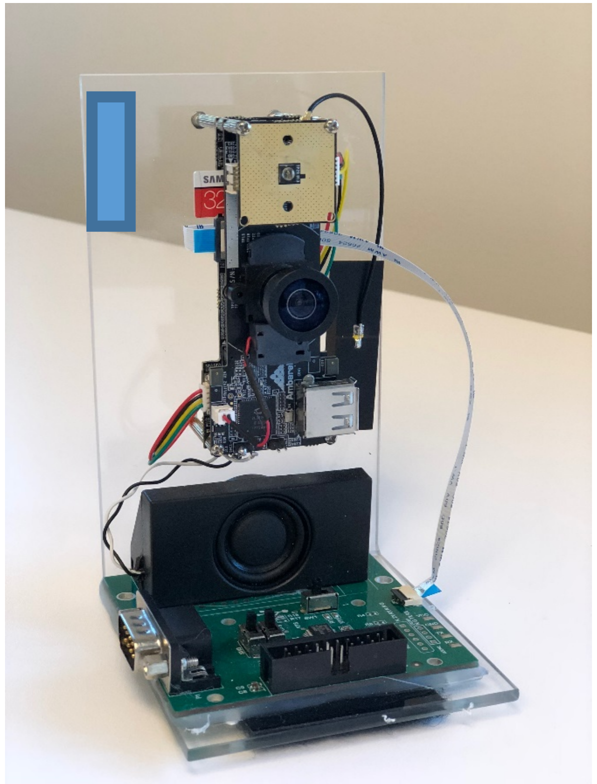

Adversarial Jittered
Image
Refinery
Probabilities
CNN
KL-Loss

Model	**Paper Number**		**Our Impl.**		**Label Refinery**	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
AlexNet [8]	59.3	81.8	57.93	79.41	**66.28**†	**86.13**†
MobileNet [28]	70.6	N/A	68.53	88.14	**73.39**	**91.07**
MobileNet0.75 [28]	68.4	N/A	65.93	86.28	**70.92**	**89.68**
MobileNet0.5 [28]	63.7	N/A	63.03	84.55	**66.66**†	**87.07**†
MobileNet0.25 [28]	50.6	N/A	50.65	74.42	**54.62**†	**77.92**†
ResNet-50 [5]	N/A	N/A	75.7	92.81	**76.5**	**93.12**
ResNet-34 [5]	N/A	N/A	73.39	91.32	**75.06**	**92.35**
ResNet-18 [5]	N/A	N/A	69.7	89.26	**72.52**	**90.73**
ResNetXnor-50 [32]	N/A	N/A	63.1	83.61	**70.34**	**89.18**
VGG16 [6]	73	91.2	70.1	88.54	**75**	**92.22**
VGG19 [6]	72.7	91	71.39	89.44	**75.46**	**92.52**
Darknet19 [33]	72.9	91.2	70.6	89.13	**74.47**	**91.94**

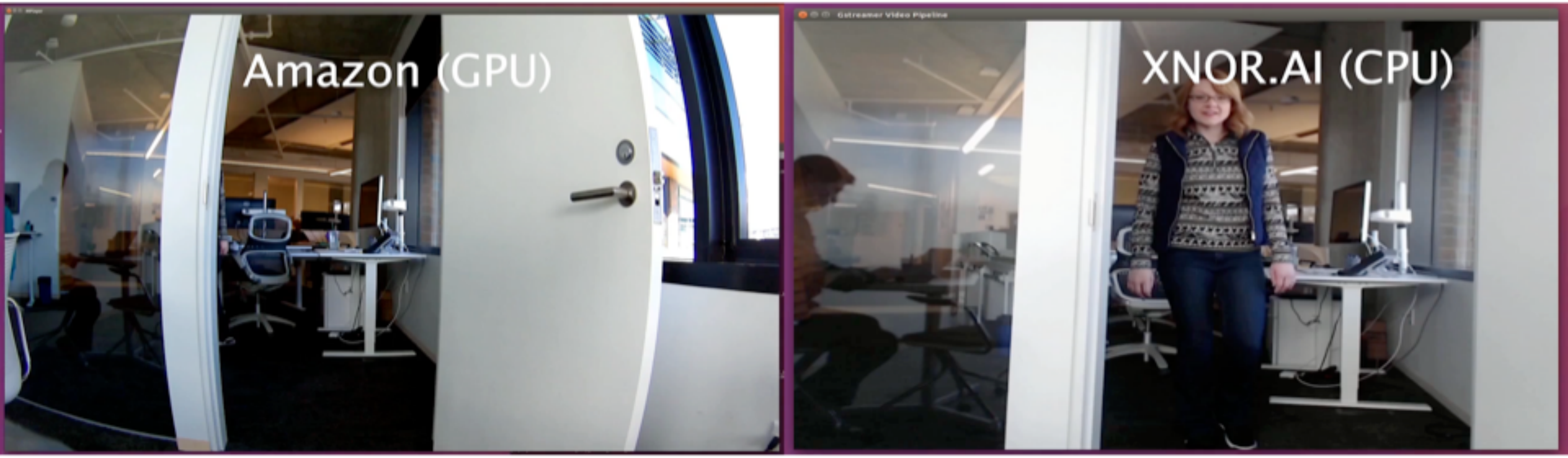
Low Power AI

S5L

- Very low power (~2x lower than Pi Zero)
- Standard AI model for object detection
  - 1 fps
- XNOR AI Model for object detection
  - 26 fps



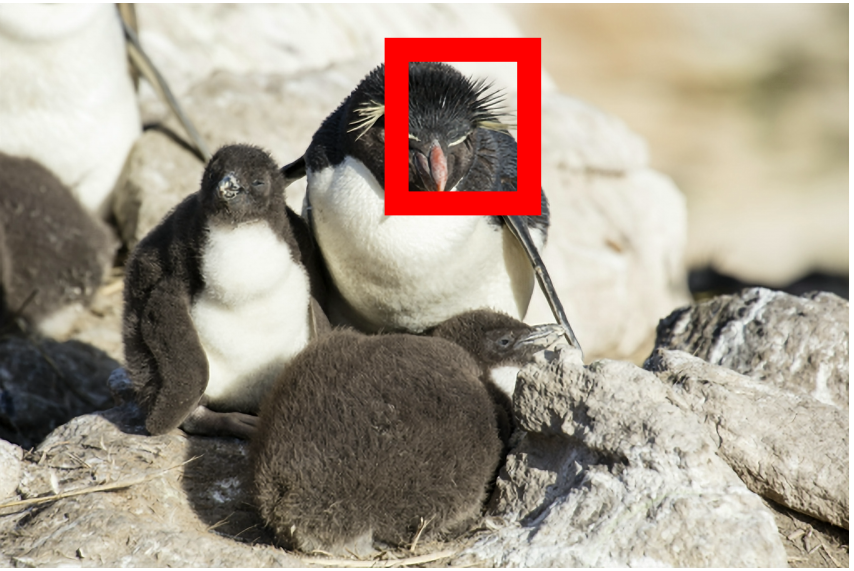
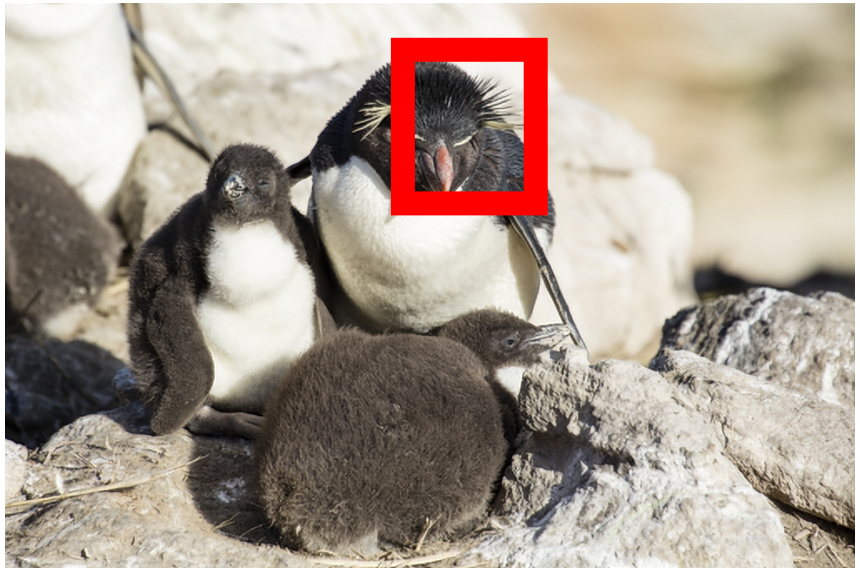
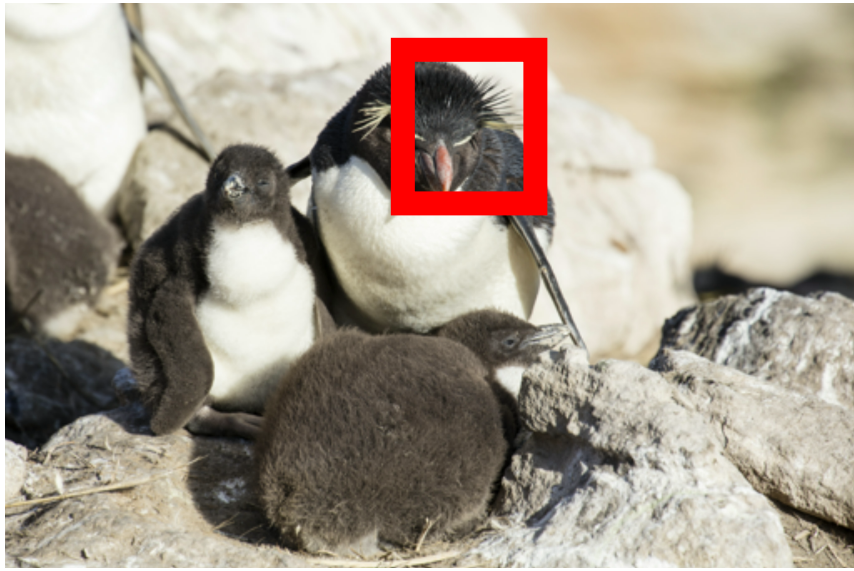
XNOR model on DeepLens



Full Precision Network

XNOR.AI

Bilinear

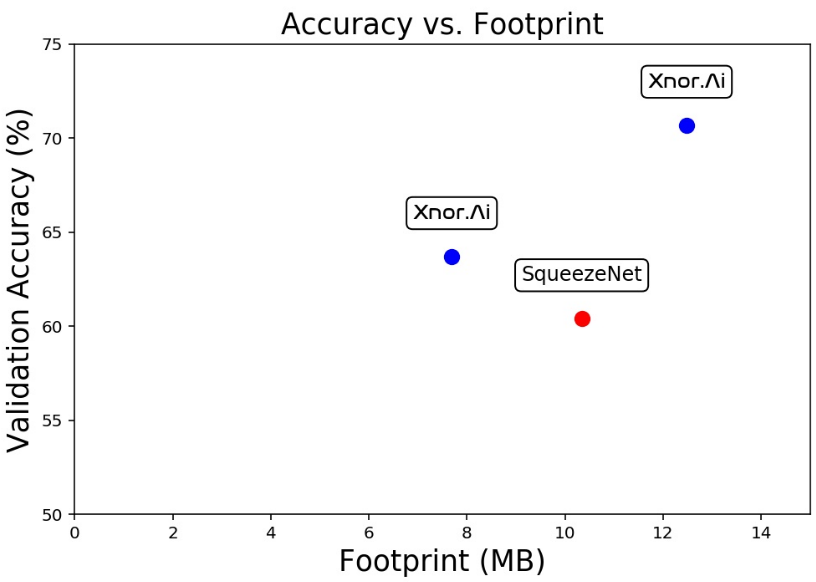


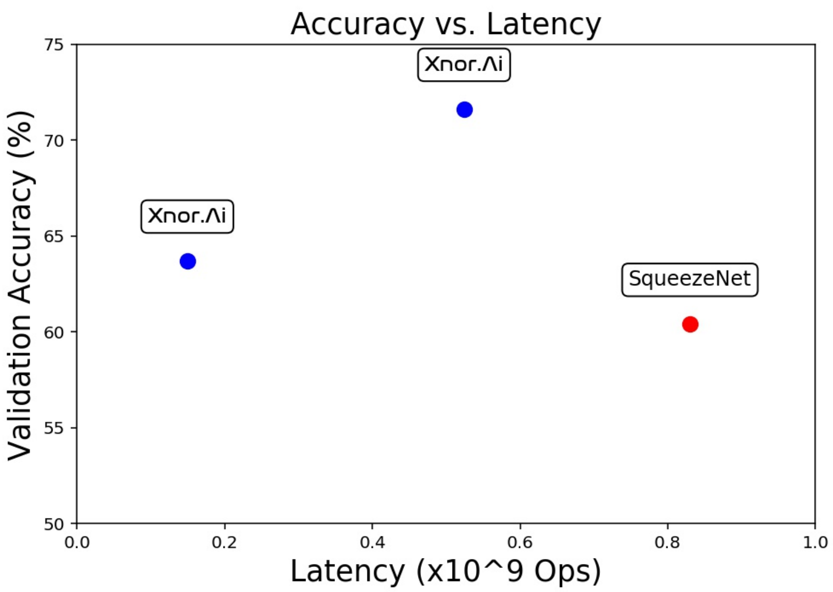
Thank you !!!

Learn more

www.xnor.ai

Competitive analysis: XNOR-Net models Vs SqueezeNet





Benchmarked results

XNOR.AI solutions are 2.5X faster than squeeze net at the same accuracy on a core i7 770K 4.2GHz Intel CPU