

# Efcnt DL

M.S. CV

# Acknowledgements



**Marco Pedersoli**

**Assistant Professor**

**ETS Montreal**

**email**

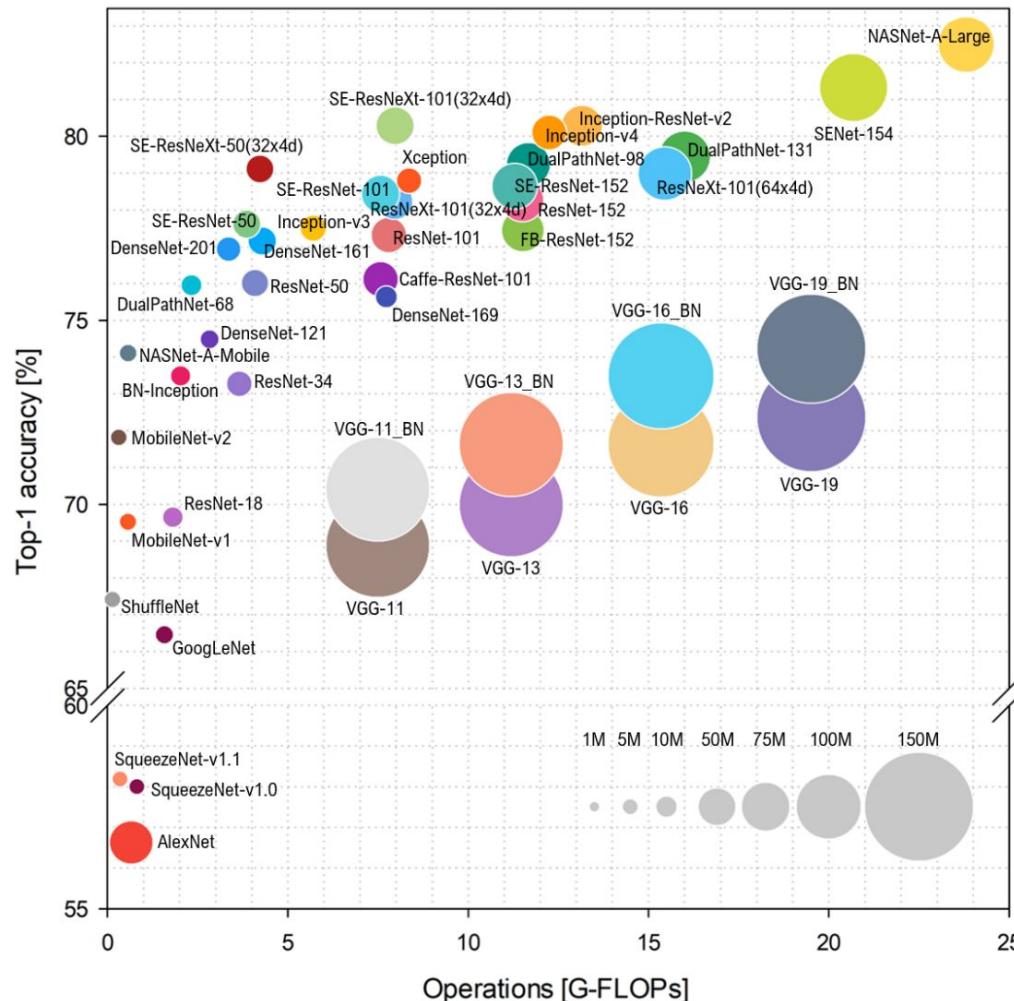
**phone: +1 (514) 396-8742**

**office: A-3480**

# Outline

- **Motivation**
- Weight pruning
- Low rank approximation
- Quantization
- Conditional / adaptive computation
- Distillation
- Architectures

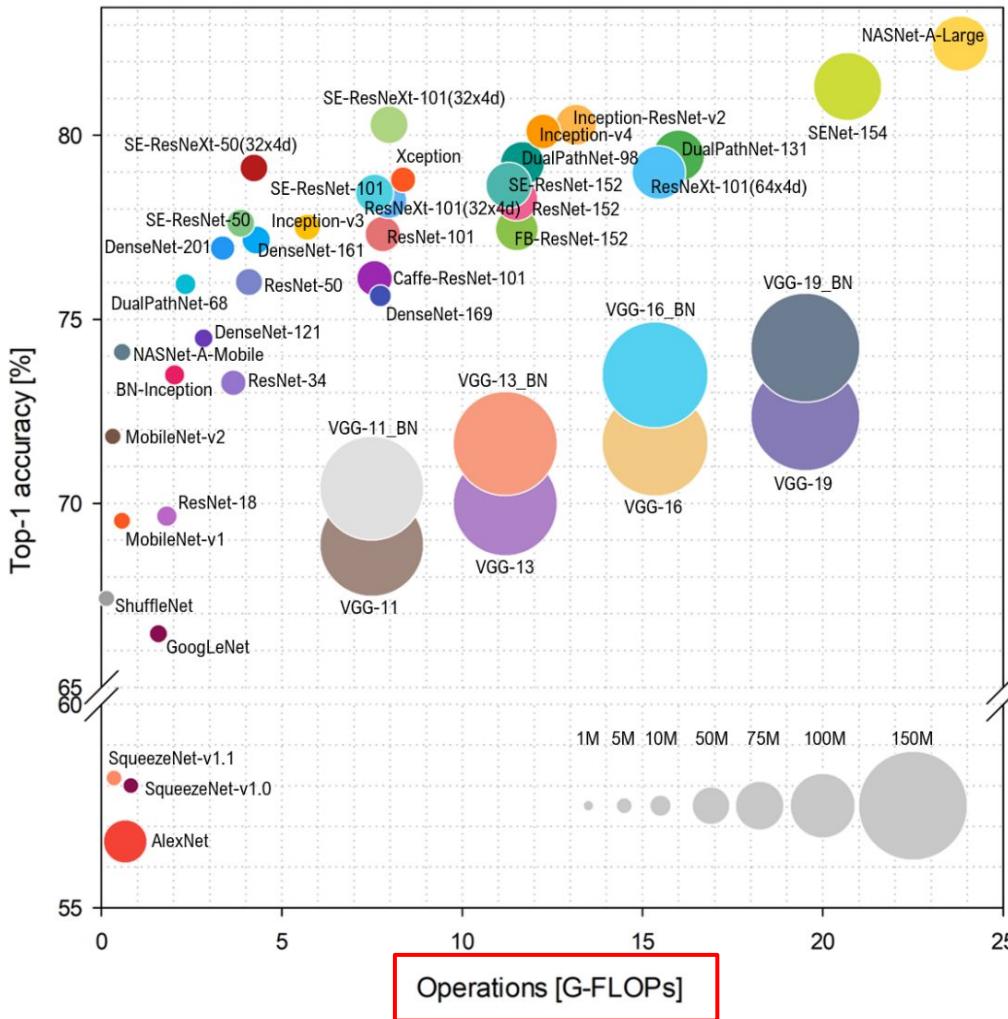
# Motivation



# Motivation

**GFLOP:** Giga-floating point operations

- 1 neural network weight = 1 floating point
- Summing two weights: 1 FLOP
- Matmult =  $O(n^3)$  FLOAT



# Motivation | Computational power

## IMAGE RECOGNITION

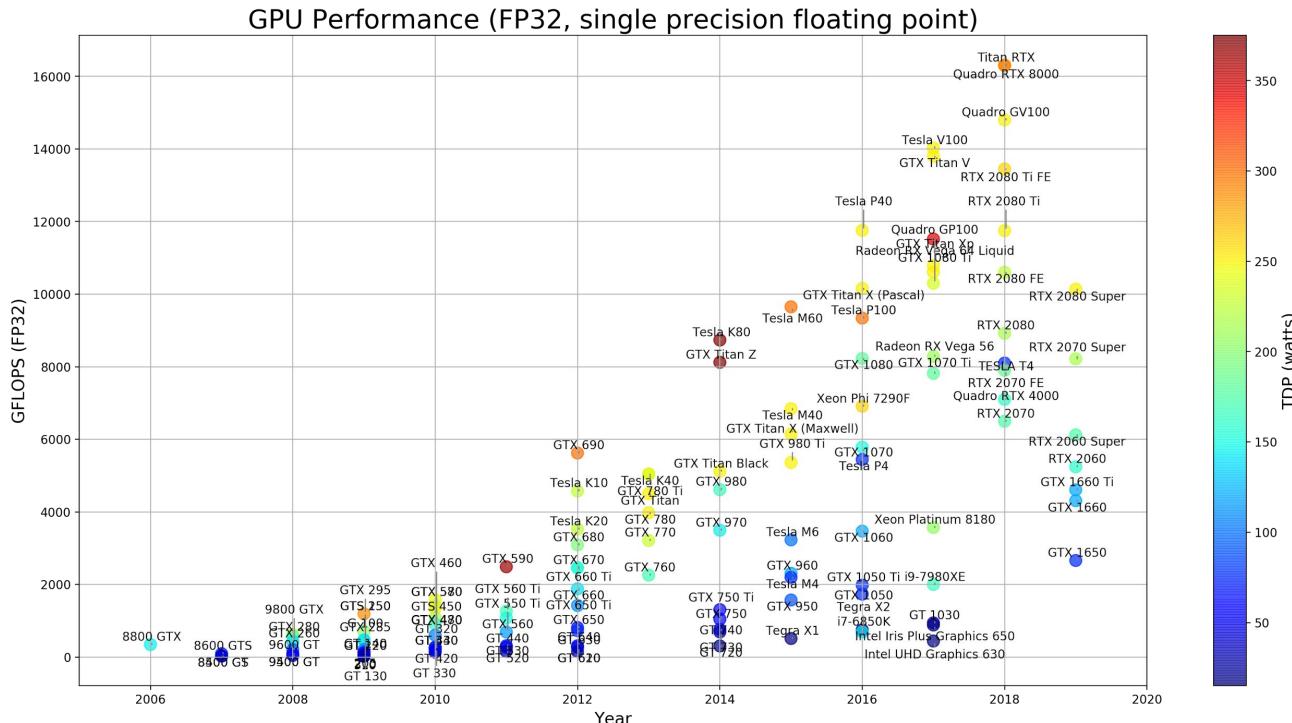


## SPEECH RECOGNITION



[Dally, NIPS'2016 workshop on Efficient Methods for Deep Neural Networks]

# Motivation | Energy consumption



source: <https://blog.inten.to/hardware-for-deep-learning-part-3-gpu-8906c1644664>

# Motivation | Energy consumption



## iPhone 11

3,110 mAh battery

(up from 2,942 mAh on iPhone XR)

4GB RAM

(up from 3GB on iPhone XR)

## iPhone 11 Pro

3,046 mAh battery

(up from 2,658 mAh on iPhone XS)

4GB RAM

## iPhone 11 Pro Max

3,969 mAh battery

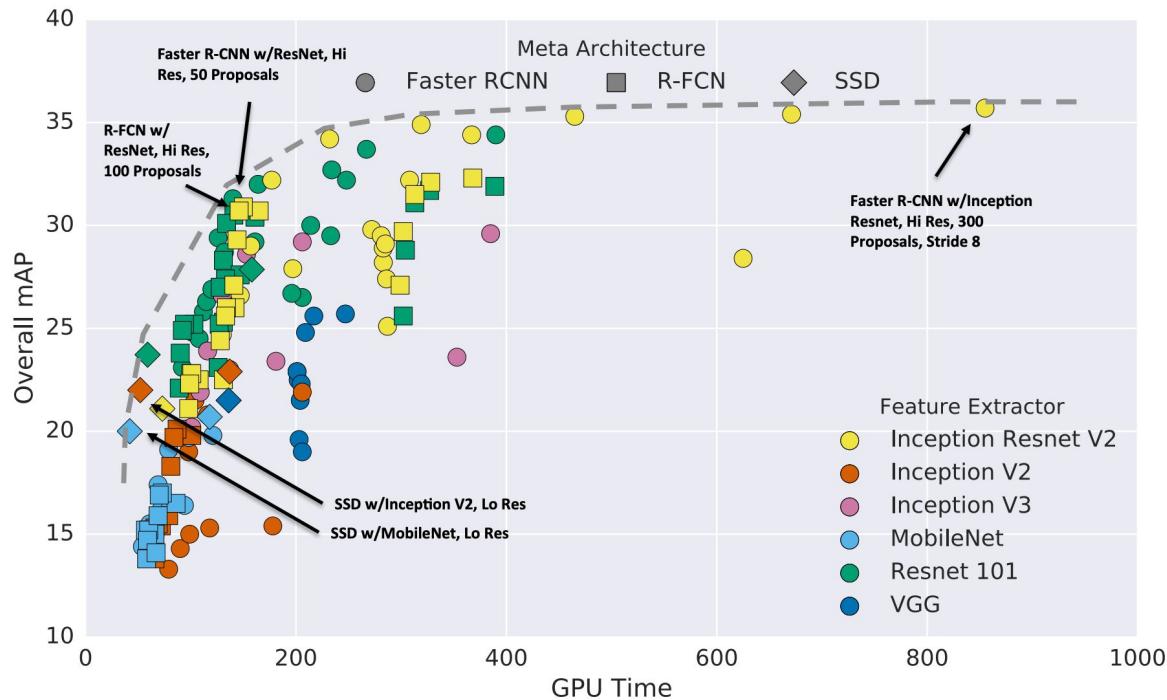
(up from 3,174 mAh on iPhone XS Max)

4GB RAM

$$250\text{W} = 25\text{A} * 10\text{v}$$

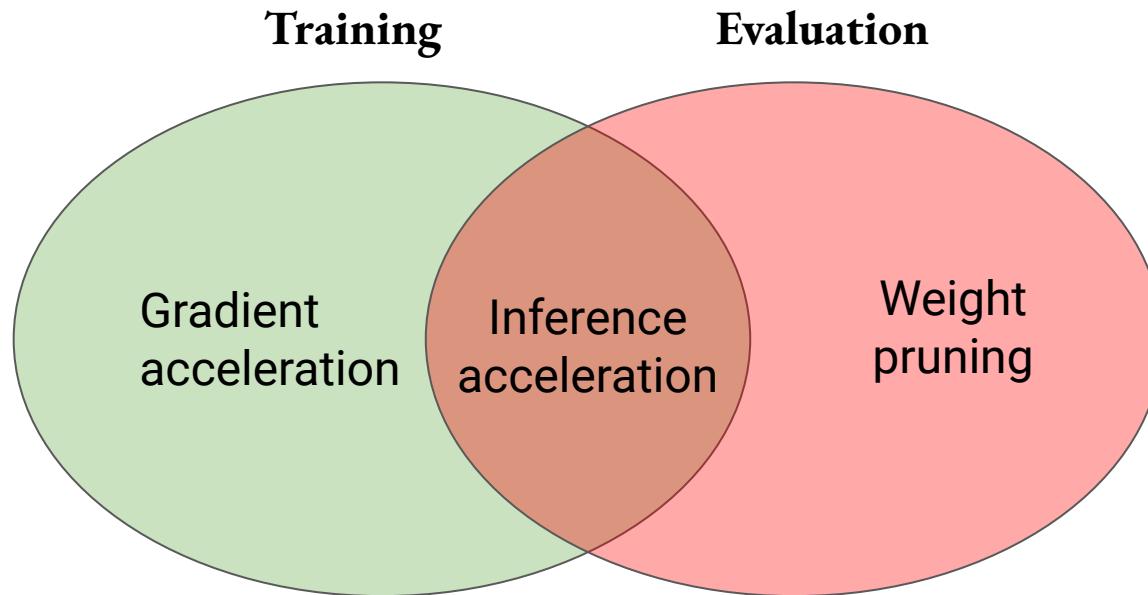
$$3110\text{mAh} / 25000\text{mA} = \mathbf{7.19\text{min}}$$

# Motivation | Speed



[Huang, J., et al. Speed/accuracy trade-offs for modern convolutional object detectors. CVPR 2017.]

# Objective: reduce complexity



# Reduction of the complexity

During training

- Regularize the model to reduce complexity or design more efficient models
  - Directly construct an efficient model
  - Requires full training of the model each time
  - Ex: weight pruning, quantization, low-rank approximation of the weights...

After training

- Modify certain parts of the network to make them more efficient and/or “fine-tuning”.
  - No need to retrain
  - Lower improvement margin

# Compression vs Reduction of calculations

## Compression

- Reduce the number of parameters to fit in embedded system
- Reduce the number of parameters to:
  - Limit the overfitting
  - Learn a better model, particularly with small databases

## Reduction of the number of calculations

- Reduce the amount of calculations to make the algorithm more efficient
- This can be achieved by reducing the amount of parameters but no guarantee

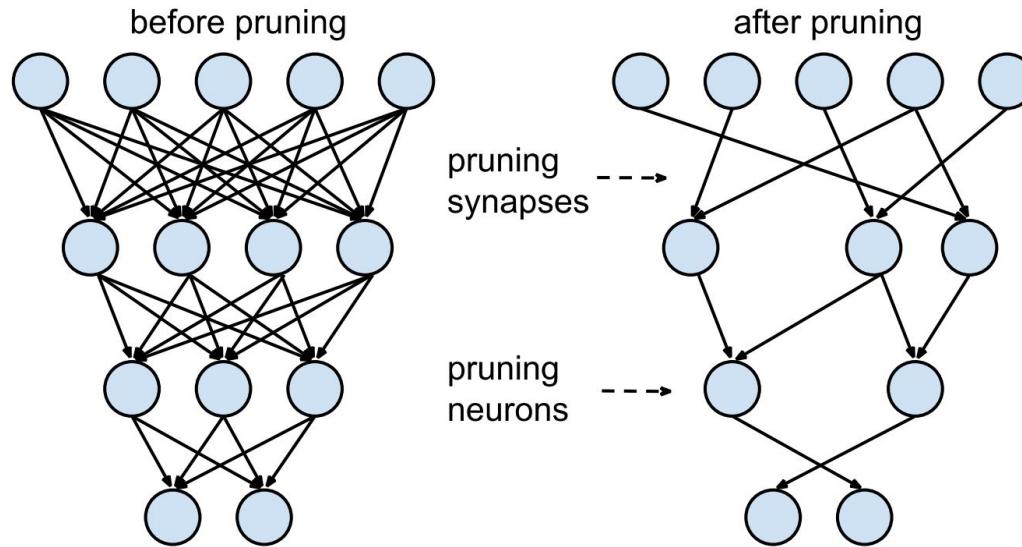
# Reducing calculations on GPU

- GPUs are optimized to execute parallel operations very fast (e.g. matrix multiplication)
- Important: to achieve good performance operations have to be **local**
- Reducing the number of operations != faster execution
  - It is important to keep a dense representation!

# Outline

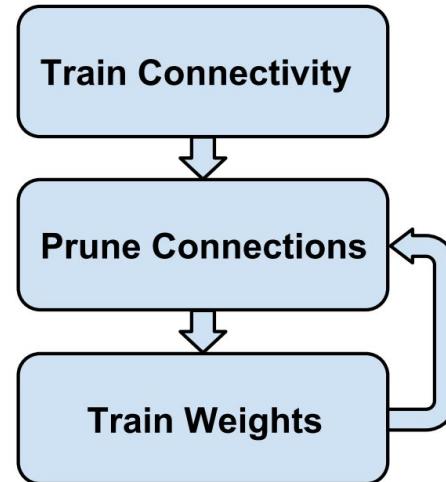
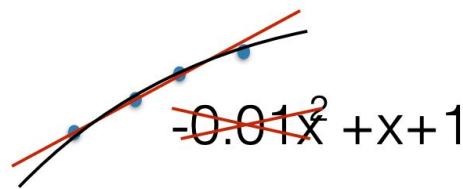
- Motivation
- **Weight pruning**
- Low rank approximation
- Quantization
- Conditional / adaptive computation
- Distillation
- Architectures

# Weight pruning



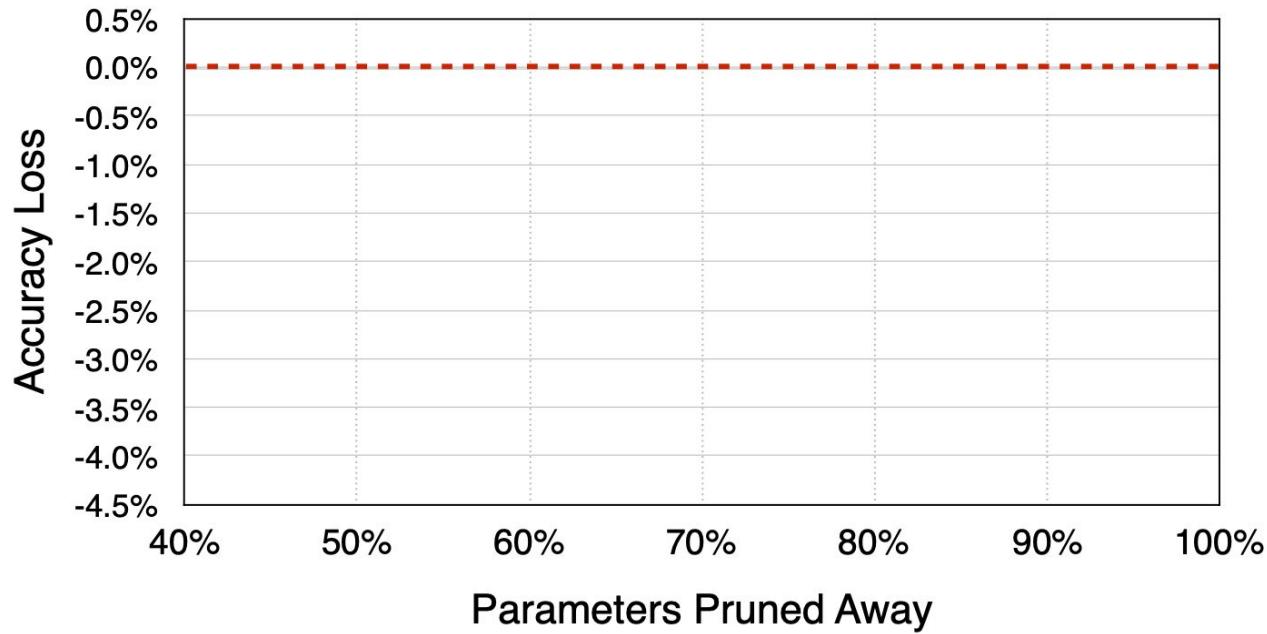
[Lecun et al. NIPS'89]  
[Han et al. NIPS'15]

# Weight pruning

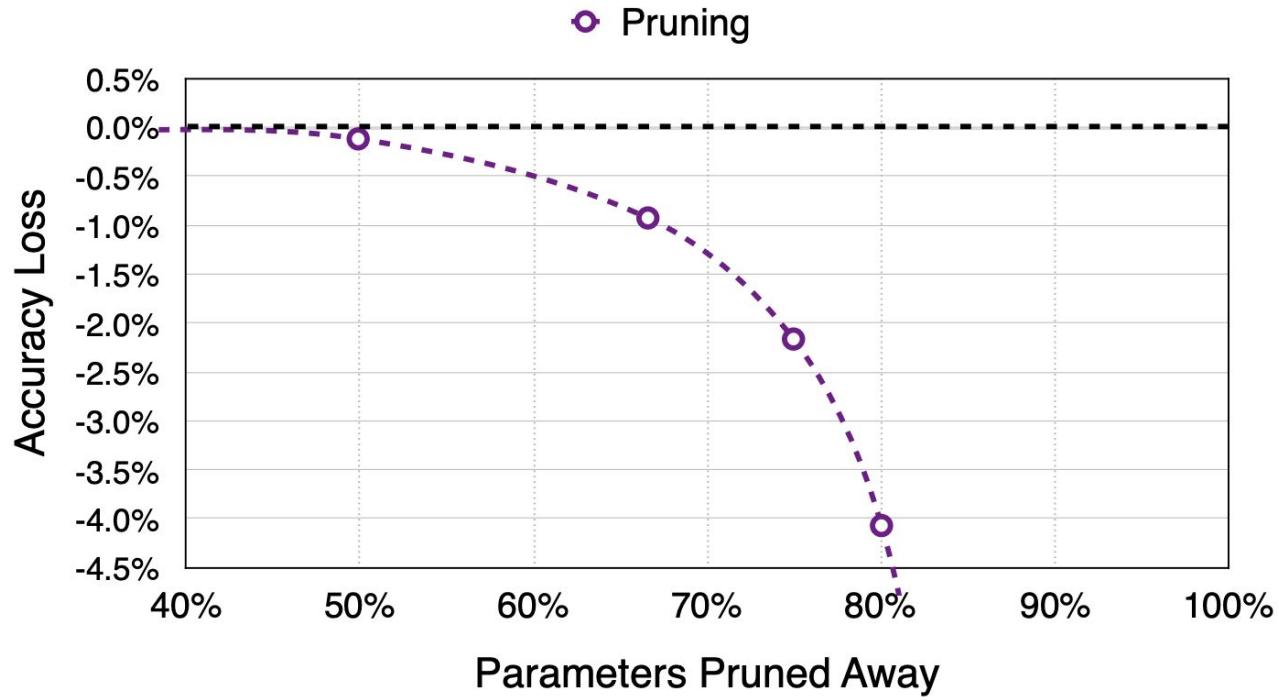
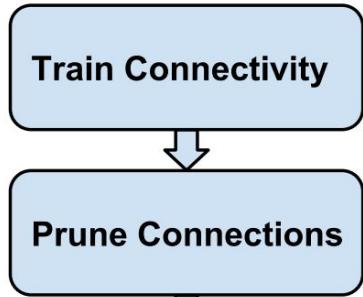


# Weight pruning

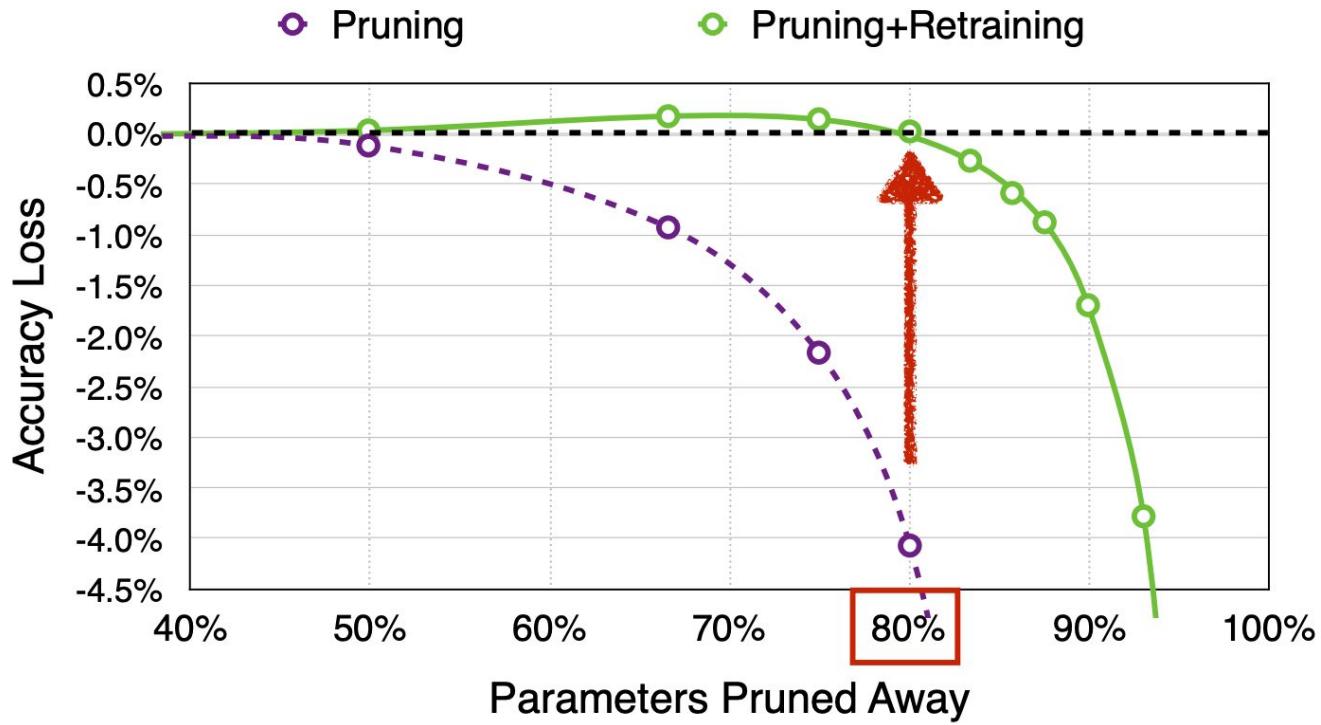
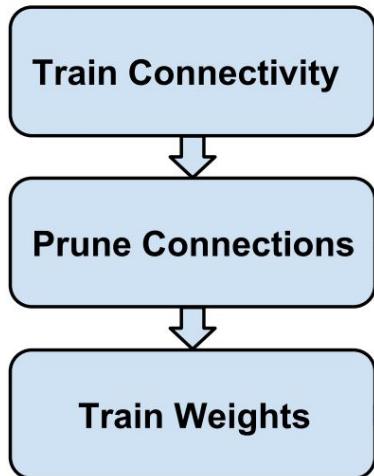
Train Connectivity



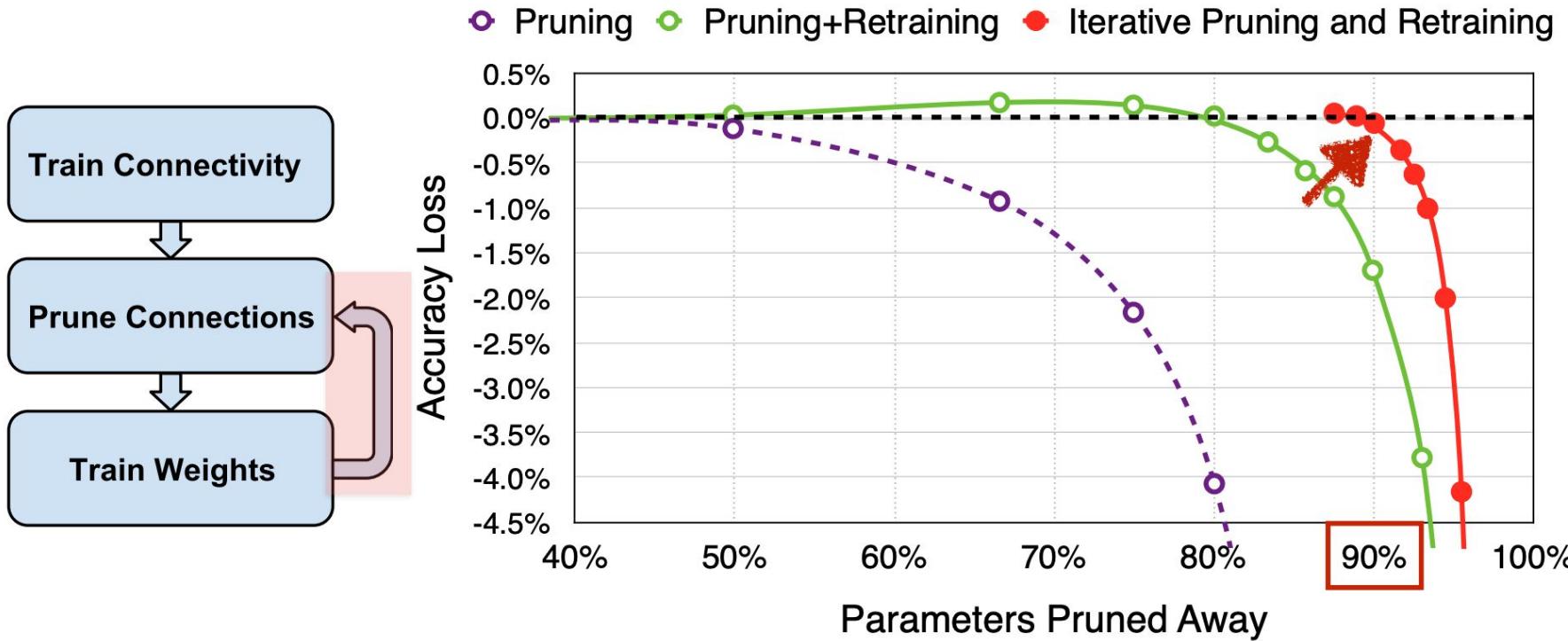
# Weight pruning



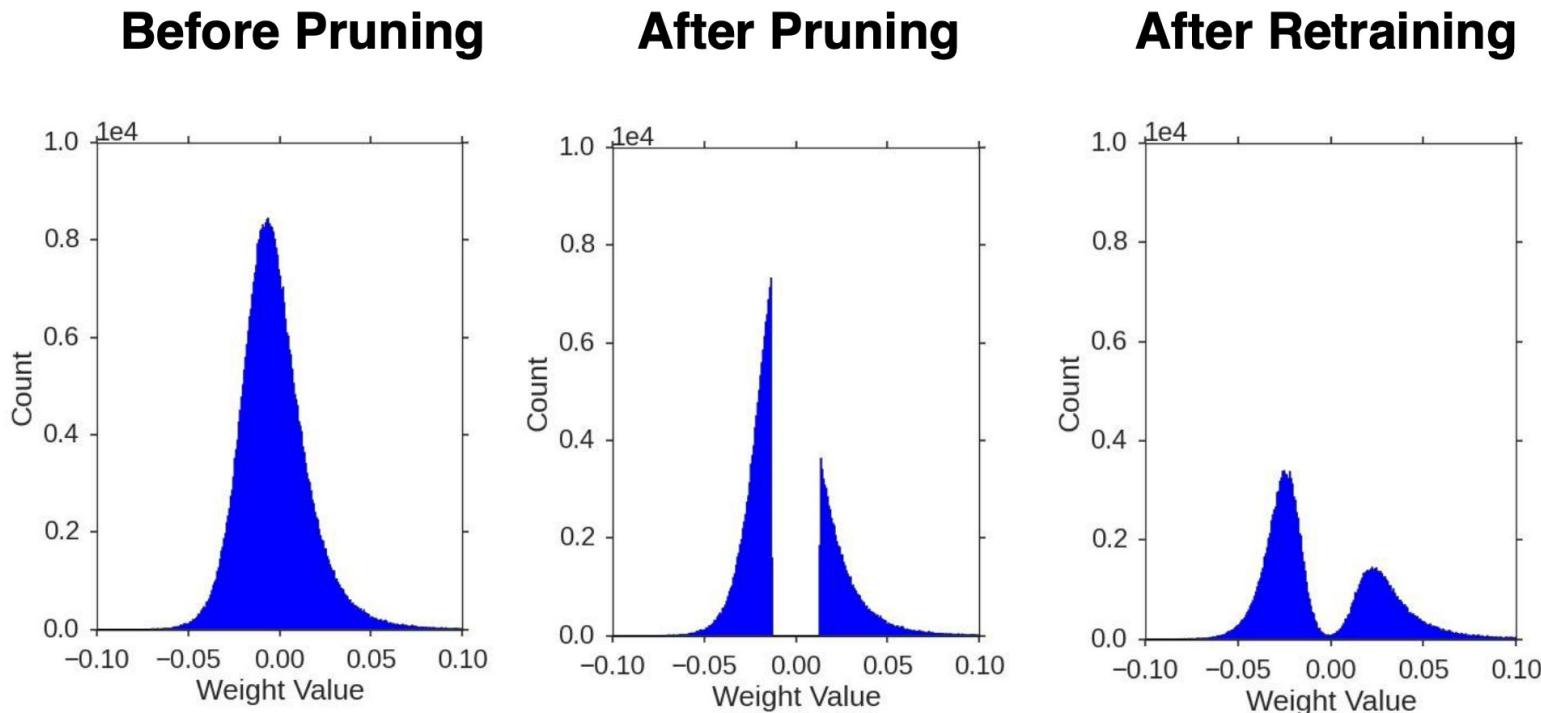
# Weight pruning



# Weight pruning

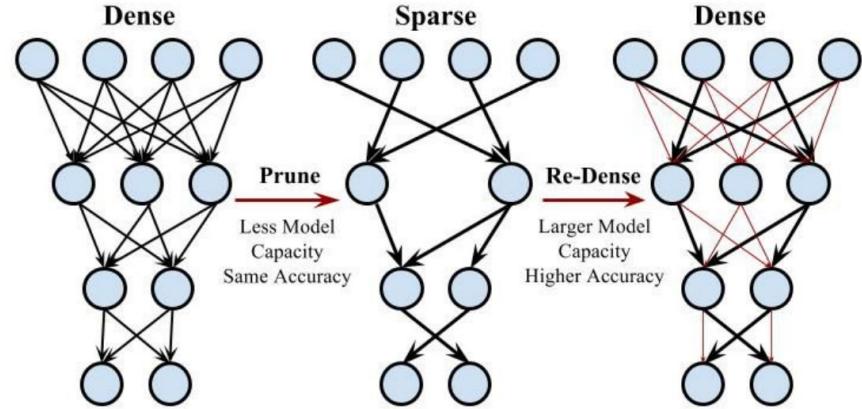


# Weight pruning



# Dense Sparse Dense

- Phase I: prune less important weights
- Phase II: re-initialize weights that are 0 and continue training



Baseline	Top-1 error	Top-5 error	DSD	Top-1 error	Top-5 error
AlexNet	42.78%	19.73%	AlexNet_DSD	41.48%	18.71%
VGG16	31.50%	11.32%	VGG16_DSD	27.19%	8.67%
GoogleNet	31.14%	10.96%	GoogleNet_DSD	30.02%	10.34%
SqueezeNet	42.39%	19.32%	SqueezeNet_DSD	38.24%	16.53%
ResNet18	30.43%	10.76%	ResNet18_DSD	29.17%	10.13%
ResNet50	24.01%	7.02%	ResNet50_DSD	22.89%	6.47%

# Optimal Brain Damage

- Big networks -> Overfitting
- Small networks -> Underfitting

How to find the optimal size?

- Regularization, early stopping, etc.
- In this work: pruning = better generalization and reduction of calculations
- Problem: when just looking at weight values, retraining is needed, which is slow

[LeCun '90]

# Optimal Brain Damage

- Taylor approximation of the loss function

$$\delta E = \sum_i g_i \delta u_i + \frac{1}{2} \sum_i h_{ii} \delta u_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta u_i \delta u_j + O(||\delta U||^3)$$

$$g_i = \frac{\partial E}{\partial u_i} \quad \text{and} \quad h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j}$$

- Only considers diagonal terms
- Convergence is achieved with  $g_i = 0$
- $h_{ii}$  can be found by backpropagation
- The weights with smallest  $h_{ii}$  can be set to 0

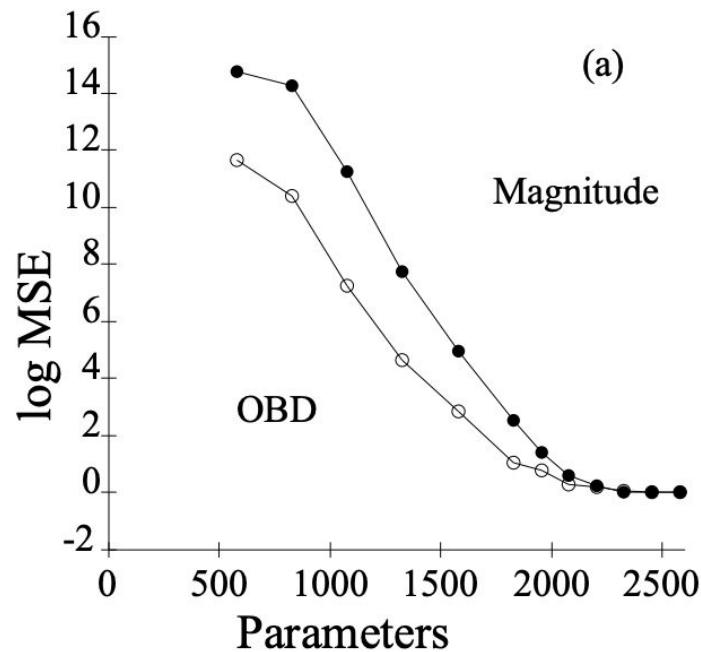
[LeCun '90]

# Optimal Brain Damage

After training, one supreses weights based on:

- Weight magnitude
- Optimal brain damage

With the same number of parameters, optimal brain damage reduces the target function better than the magnitude

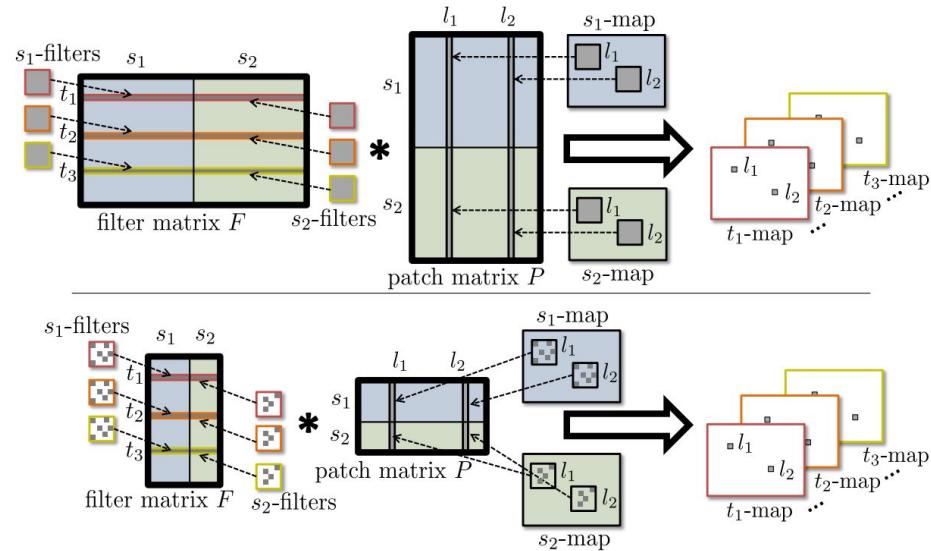


[LeCun '90]

# OBD for ConvNets

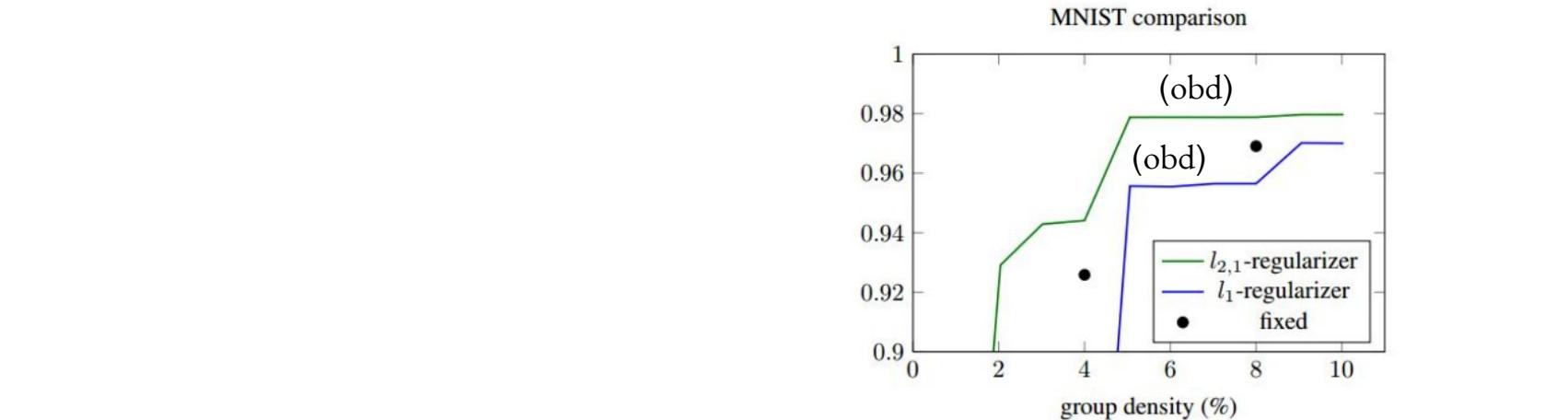
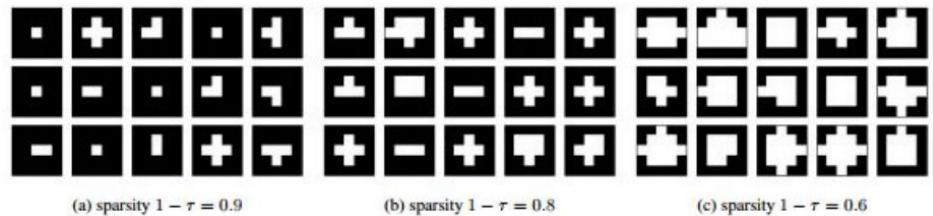
Reduce the cost by reducing the number of parameters of the convolution filters

- Convolution can be considered a matrix multiplication
- For a real advantage one requires dense matrices -> reduce groups!



# OBD for ConvNets

- L1 Regularization
- vs OBD



# Weight pruning | Human brain



Newborn

**50 Trillion Synapses**



1 year

**1000 Trillion Synapses**



Teenager

**50 Trillion Synapses**

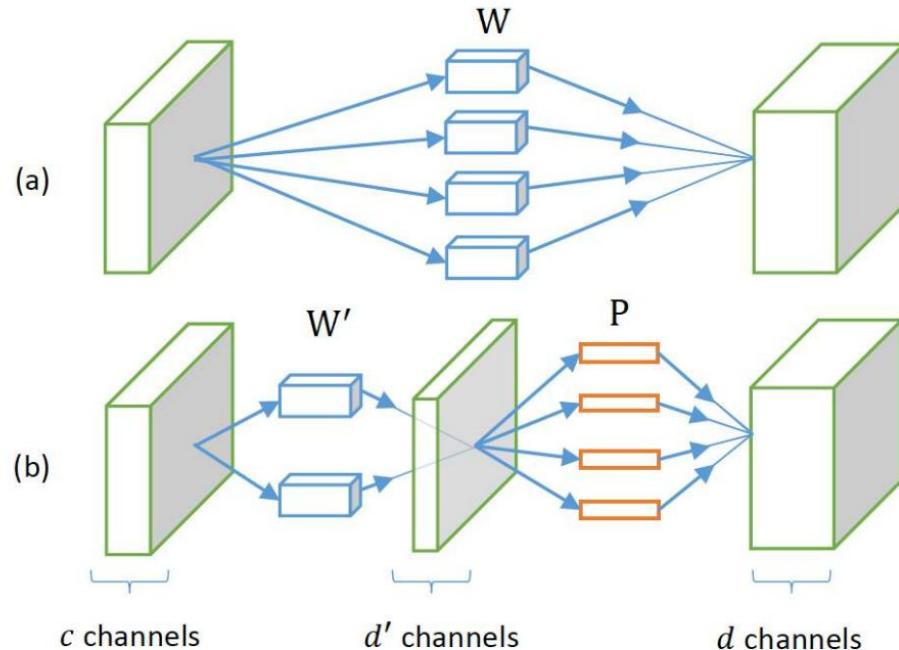
[C.A. Walsh. P. Huttenlocher (1931-2013). *Nature* 2013]

# Outline

- Motivation
- Weight pruning
- **Low-rank approximation**
- Quantization
- Conditional / adaptive computation
- Distillation
- Architectures

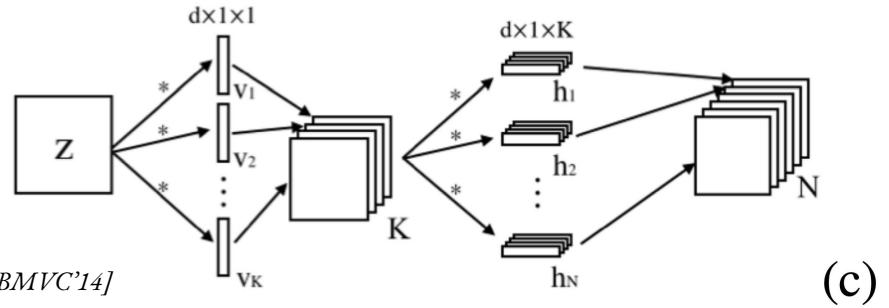
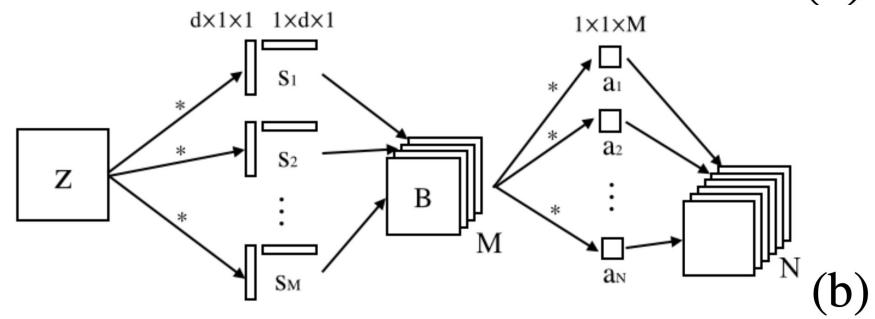
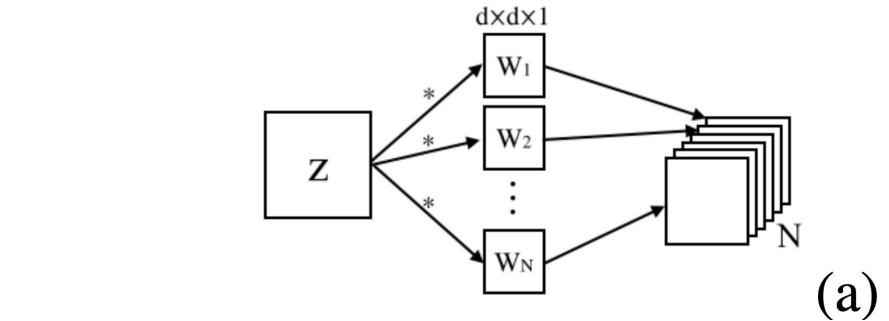
# Low-rank approximation

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with  $d$  filters with filter size  $k \times k \times c$  to
  - A layer with  $d'$  filters ( $k \times k \times c$ )
  - A layer with  $d$  filter ( $1 \times 1 \times d'$ )



# Low-rank approximation

- (a) Original conv filters
- (b) Approximation 1
  - line vector \* column vector \* M
- (c) Approximation 2
  - column vector \* K \* line vector \* N



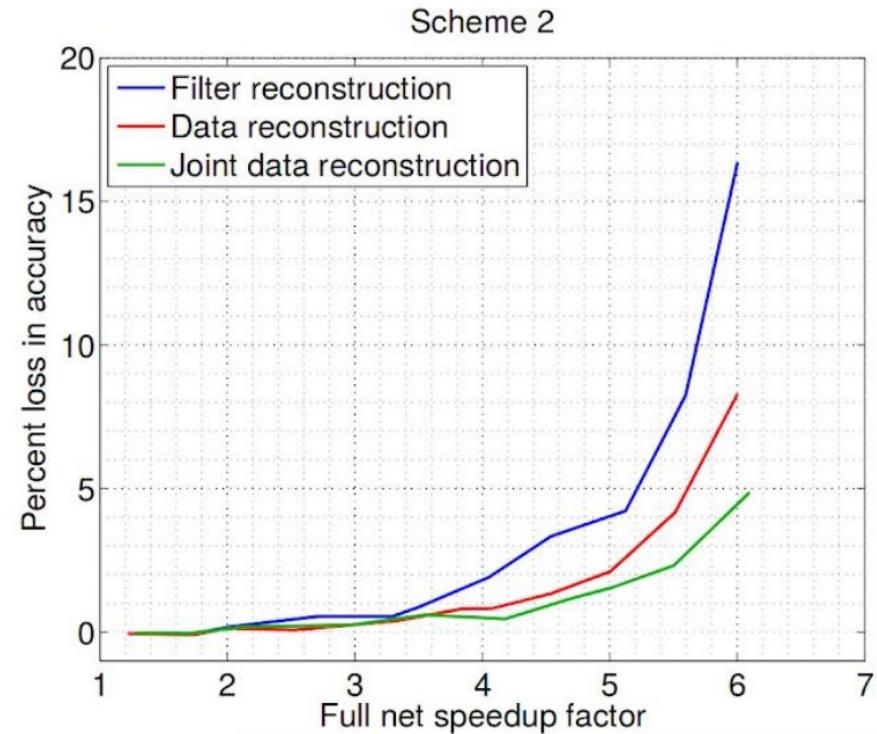
[“Speeding up Convolutional Neural Networks with low rank expansions”, Jaderberg et. al. BMVC’14]

# Speed vs Precision

Approximate trained network.

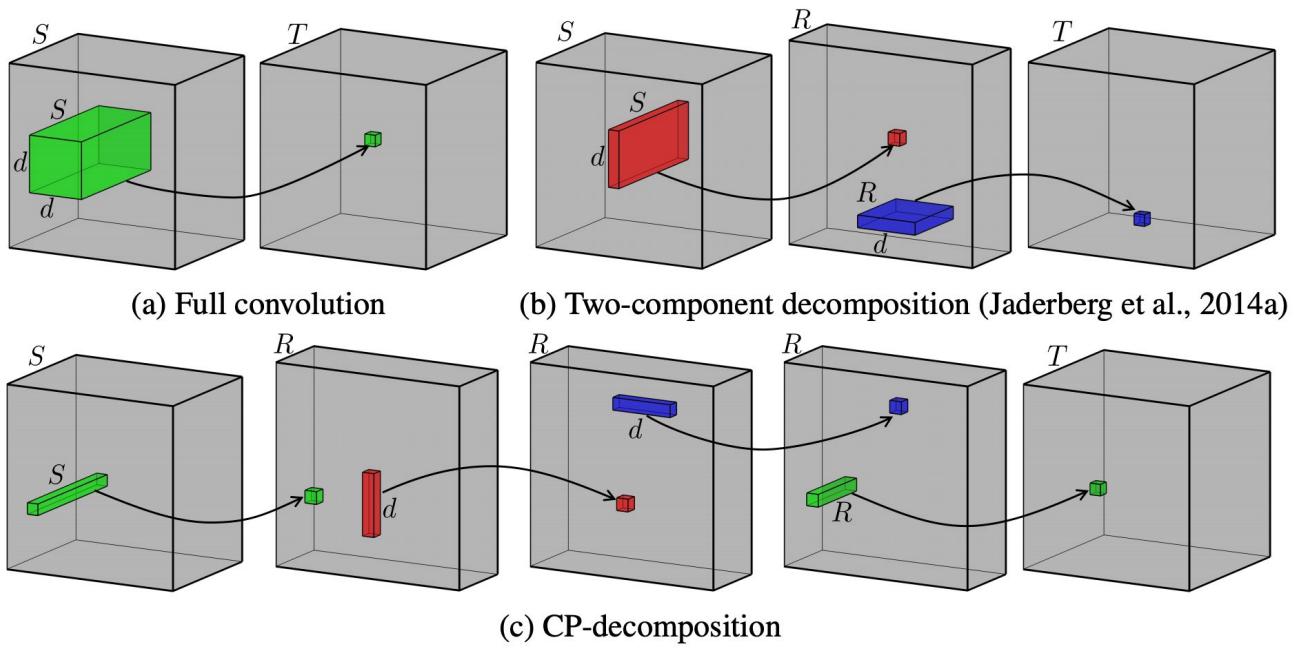
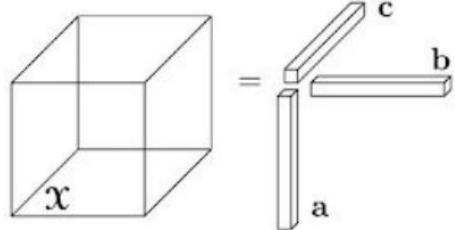
Strategies:

- Optimize filter reconstruction
- Optimize output reconstruction
- Both of them



[“Speeding up Convolutional Neural Networks with low rank expansions”,  
Jaderberg et. al. BMVC’14]

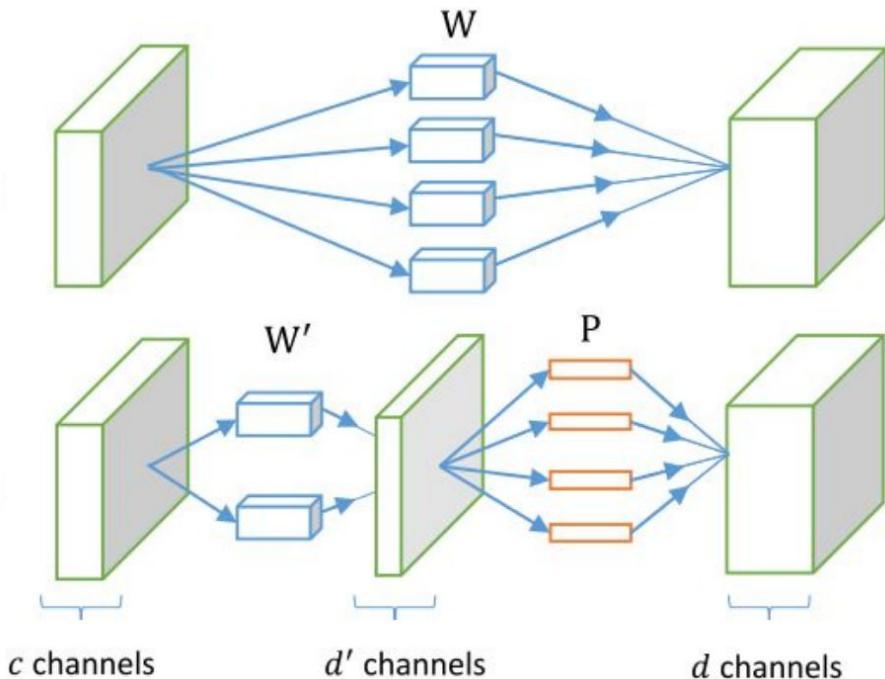
# Improvement I



[Lebedev, Vadim, et al. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition." ICLR (2015).]

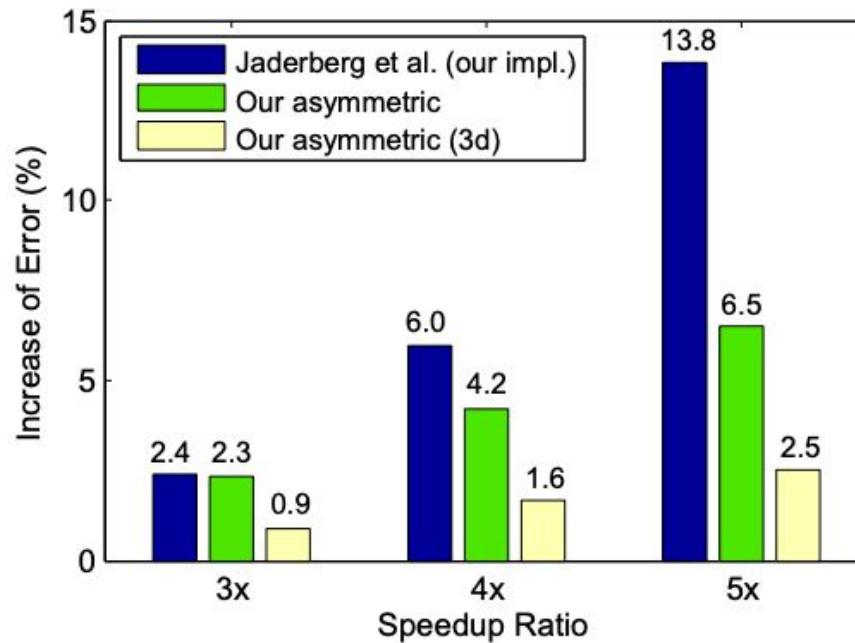
# Improvement II

- Simpler decomposition
- however, the non-linearity is considered!
- Reconstruction function: non-linear Mean square error.
- The effect of stacking multiple layers considered during optimization!



[“Efficient and Accurate Approximations of Nonlinear Convolutional Networks”, X. Zhang et al. CVPR’15]

# Improvement II



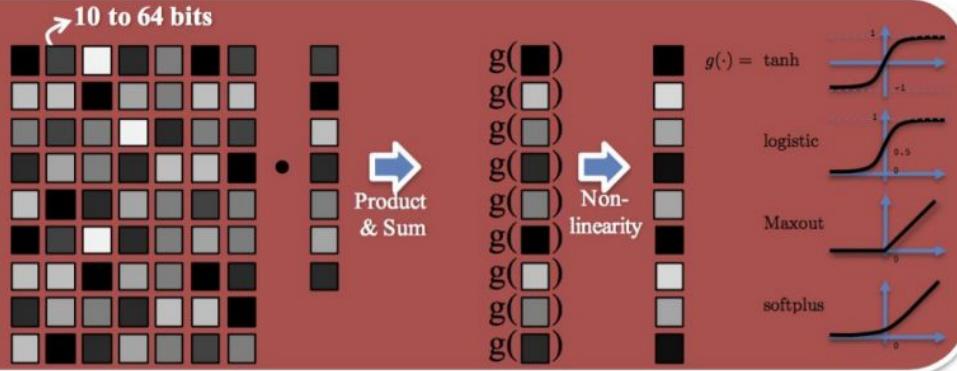
[“Efficient and Accurate Approximations of Nonlinear Convolutional Networks”, X. Zhang et al. CVPR’15]

# Outline

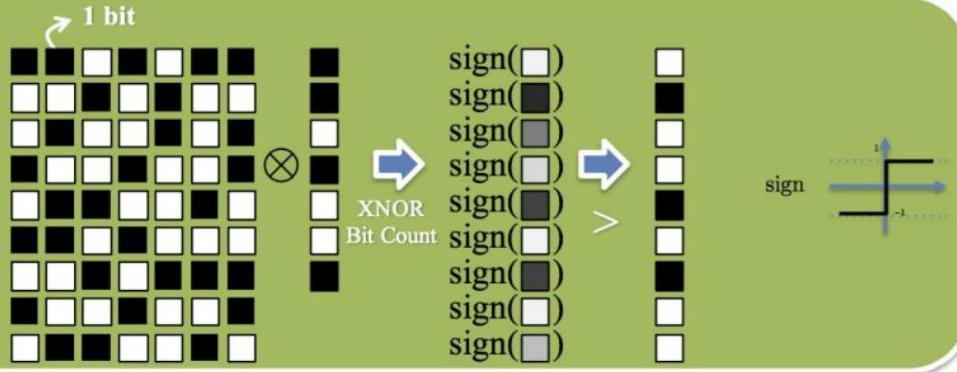
- Motivation
- Weight pruning
- Low-rank approximation
- **Quantization**
- Conditional / adaptive computation
- Distillation
- Architectures

# Quantization

Real-valued  
Networks

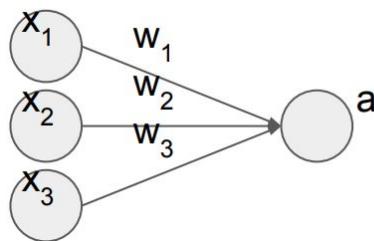


Binary  
Networks



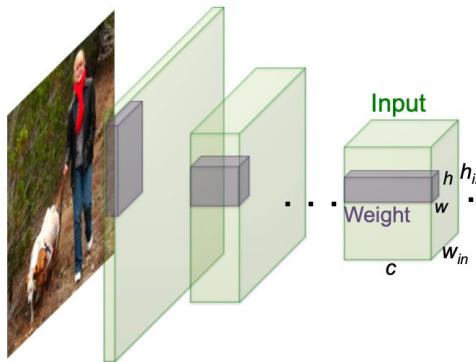
# Quantization | BinaryConnect

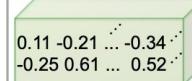
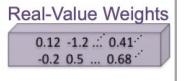
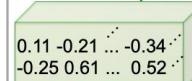
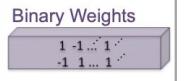
- Weights are mapped to the values **-1 and 1**. Only one bit per value is needed!
- No multiplications at inference time
- The **original values are kept** during **backpropagation**
- Quantization is a kind of regularization. Instead of converting values to 0 like dropout, it converts them to -1 or 1.



$$a = w_0 x_0 + w_1 x_1 + w_2 x_2$$
$$a_b = \pm x_0 \pm x_1 \pm x_2$$

# Quantization | XOR Networks



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	<b>Real-Value Inputs</b>  <b>Real-Value Weights</b> 	+ , - , $\times$	1x	1x	%56.7
Binary Weight	<b>Real-Value Inputs</b>  <b>Binary Weights</b> 	+ , -	$\sim 32x$	$\sim 2x$	%56.8
BinaryWeight Binary Input (XNOR-Net)	<b>Binary Inputs</b>  <b>Binary Weights</b> 	XNOR , bitcount	$\sim 32x$	$\sim 58x$	%44.2

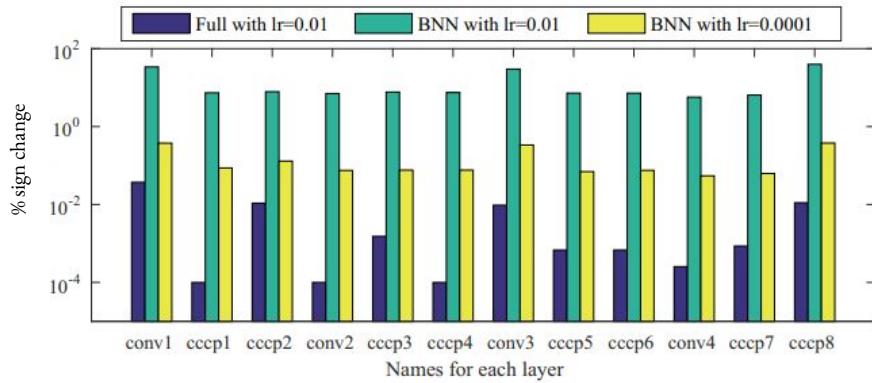
- Approximation of the convolution  $\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B}) \alpha$
- with  $\mathbf{B}^* = \text{sign}(\mathbf{W})$

# Quantization | How to train binary weights

- Reduce the learning rate
- Use a different regularization

$$J(\mathbf{W}, \mathbf{b}) = L(\mathbf{W}, \mathbf{b}) + \lambda \sum_{l=1}^L \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} (1 - (\mathbf{W}_{l,ij})^2)$$

- Add a scale coefficient to the last layer to avoid softmax saturation



# Quantization | Now we can train binary weights!

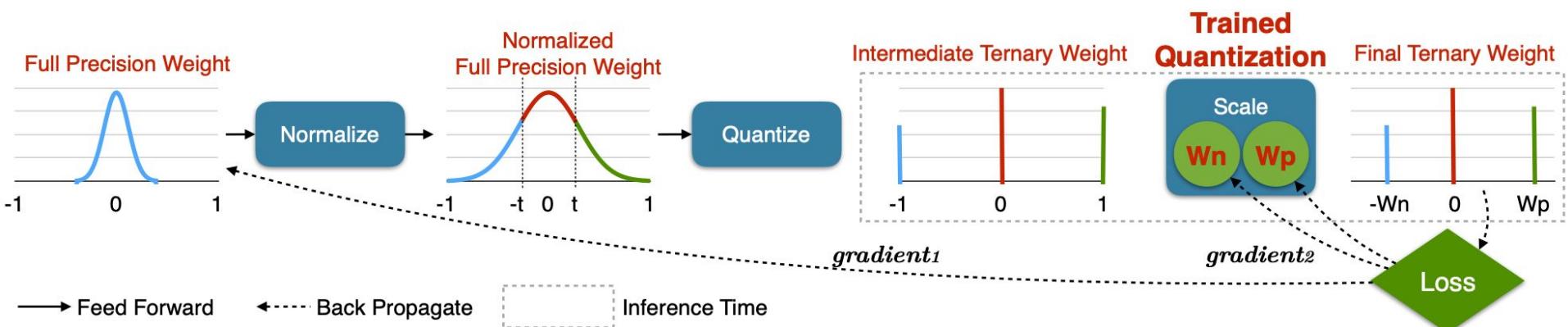
Methods	Base model	Bits of A	Last layer	Comp. rate	Model size (bef.)	Model size (aft.)	Accuracy(%)
BNN	AlexNet	1	Full	10.3×	232MB	22.6MB	50.4/27.9
XNOR-net	AlexNet	1	Full	10.3×	232MB	22.6MB	69.2/44.2
XNOR-net	ResNet-18	1	Full	70×(13.4×)	44.6MB	3.34MB	73.2/51.2
DoReFa-net	AlexNet	2	Full	10.3×	232MB	22.6MB	— /49.8
Our method	AlexNet	2	Binary	31.2×	232MB	7.43MB	71.1/46.6
Our method	NIN-net	2	Binary	189×(23.6×)	29MB	1.23MB	75.6/51.4

With the previous improvements one can achieve:

- A compression of x30-x190
- A big reduction in complexity
- A precision only 5-10 points inferior on AlexNet

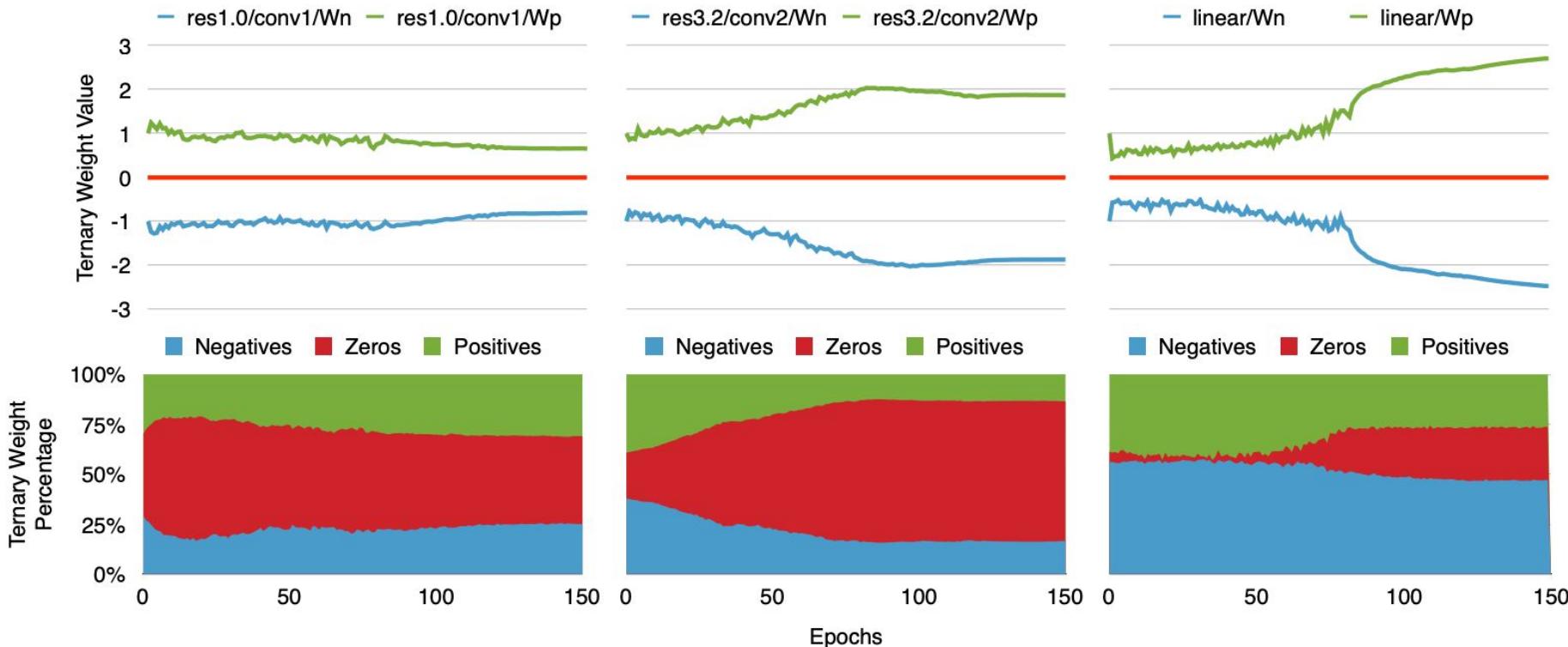
# Quantization | Trained Ternary Quantization

- Three possible values -1, 0, 1
- Zeros -> save computation at expense of performance



[Zhu, Han, Mao, Dally. Trained Ternary Quantization, ICLR'17]

# Quantization | Trained Ternary Quantization



# Quantization | Trained Ternary Quantization

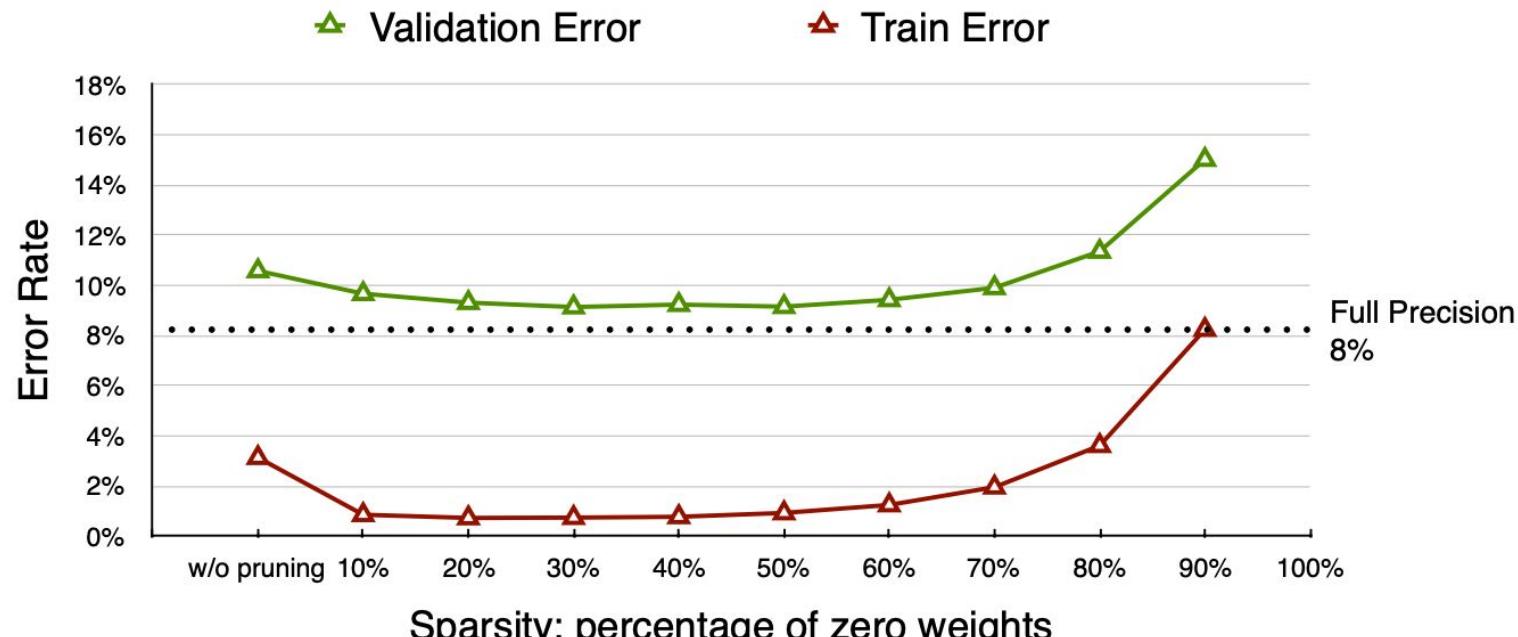
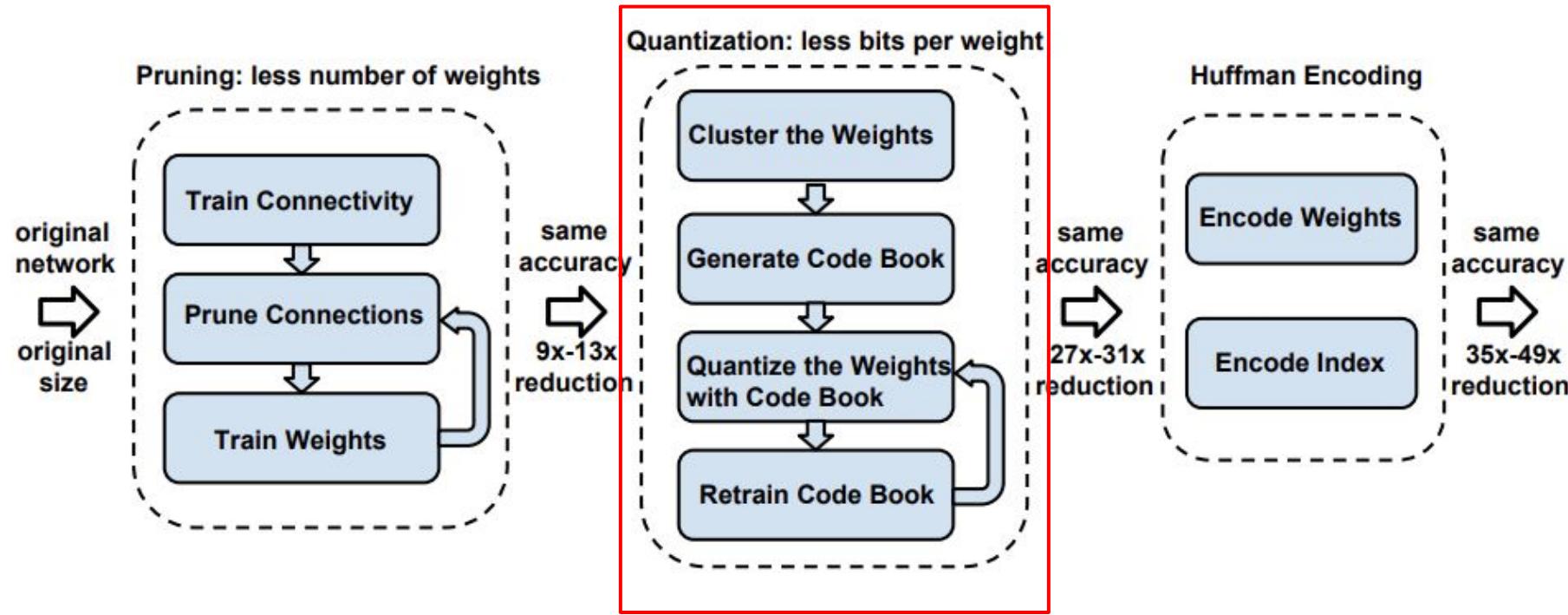
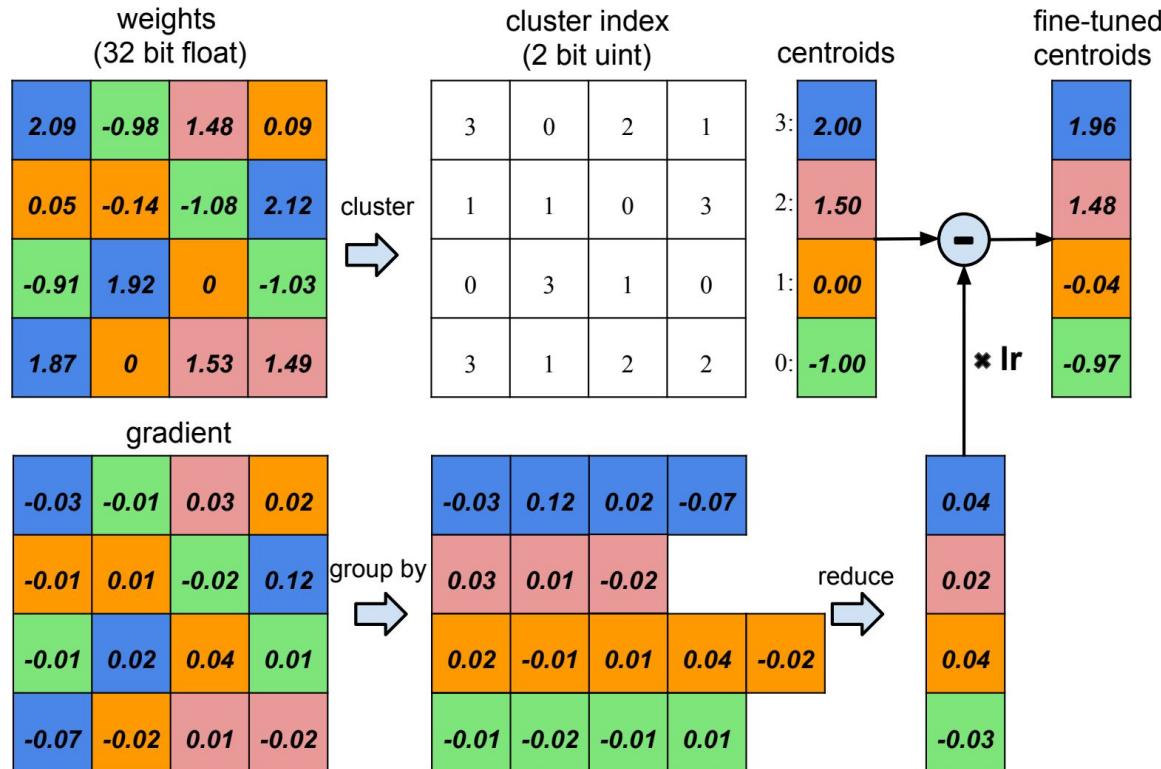


Figure 5: Accuracy v.s. Sparsity on ResNet-20

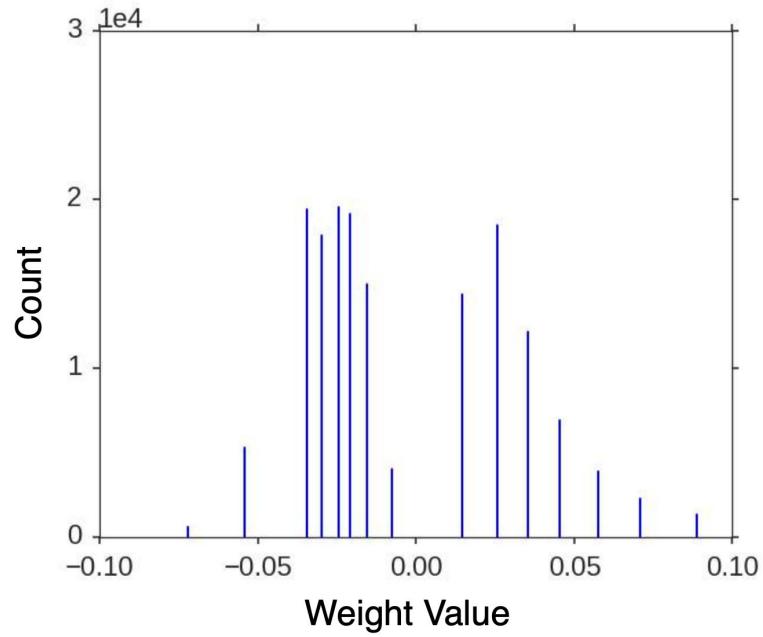
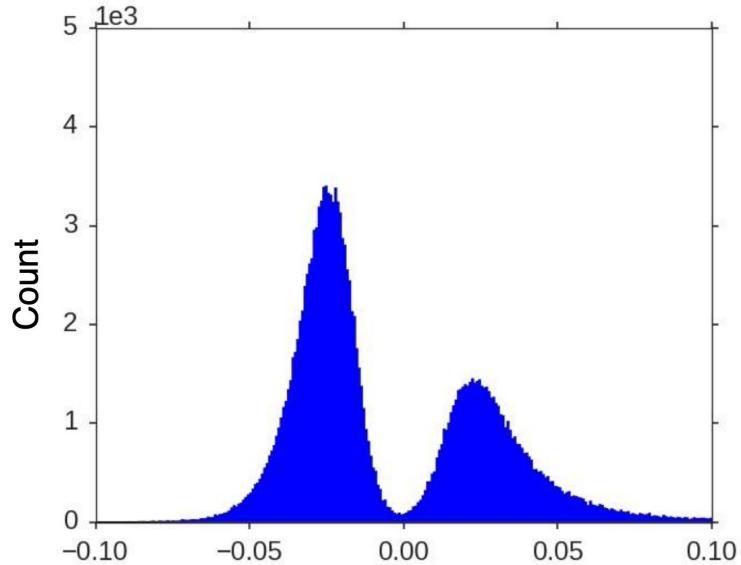
# Trained Quantization



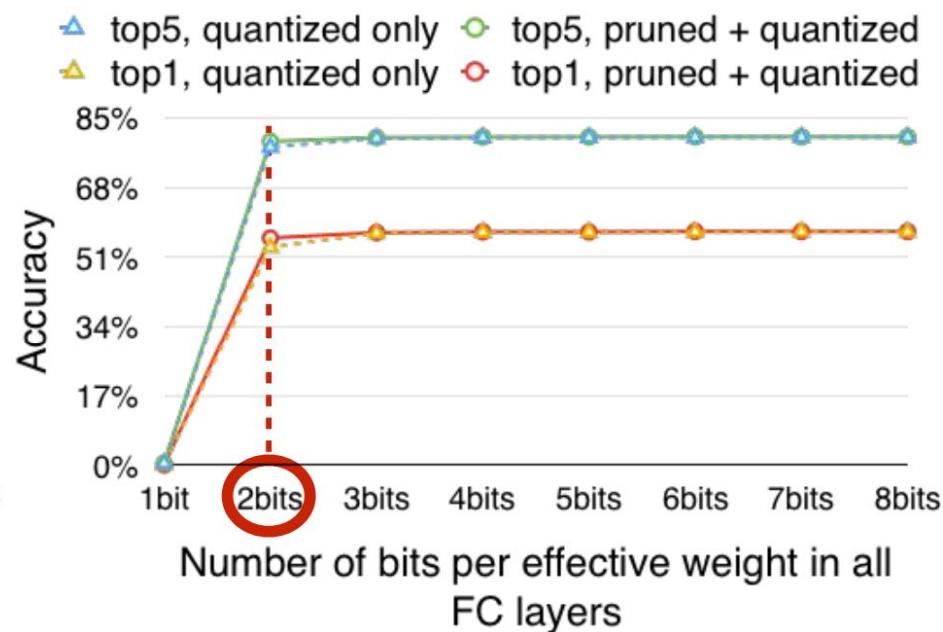
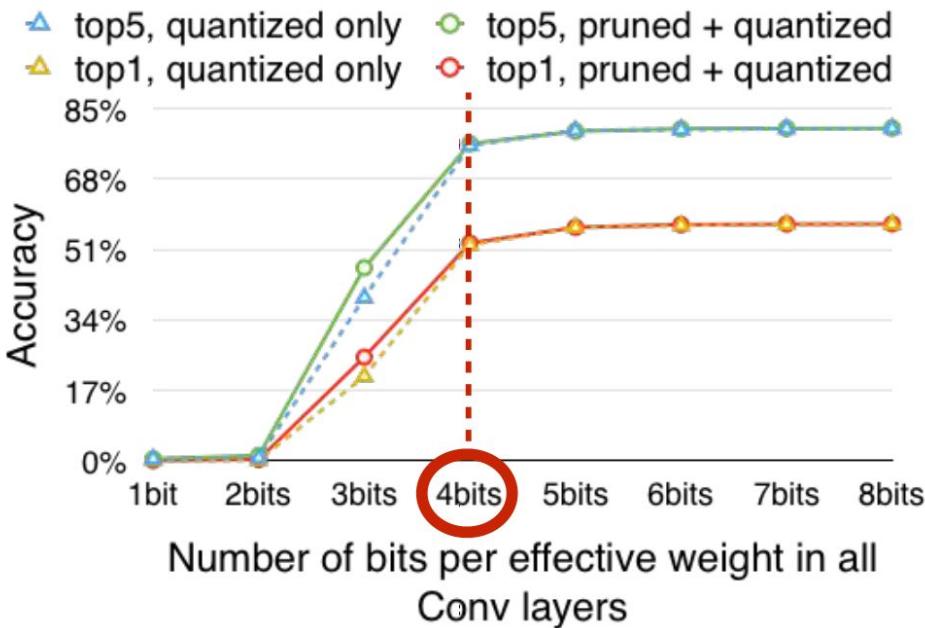
# Trained Quantization | Weight Sharing



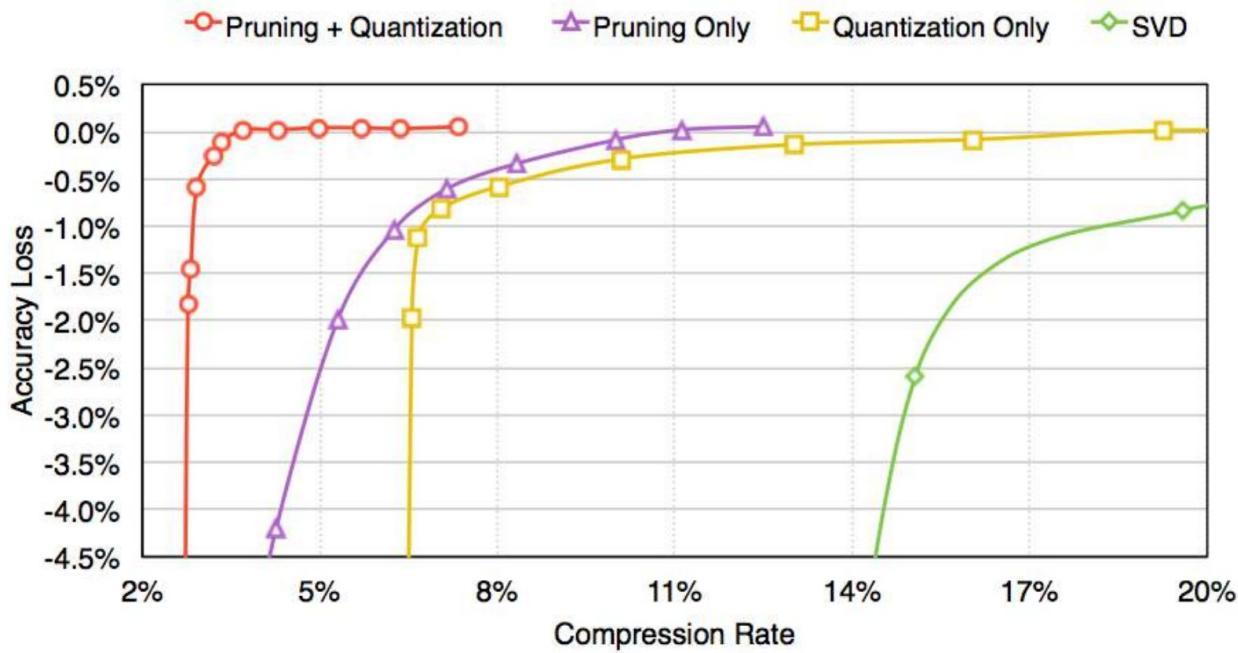
# Trained Quantization



# Trained Quantization



# Trained Quantization | Prunning



AlexNet on ImageNet

# Outline

- Motivation
- Weight pruning
- Low-rank approximation
- Quantization
- **Conditional / adaptive computation**
- Distillation
- Architectures

# Conditional Computation

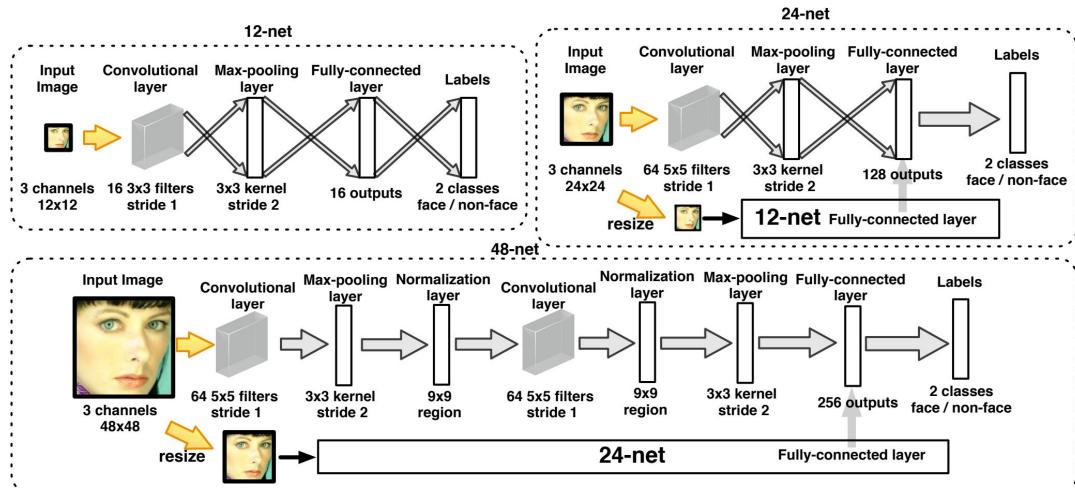
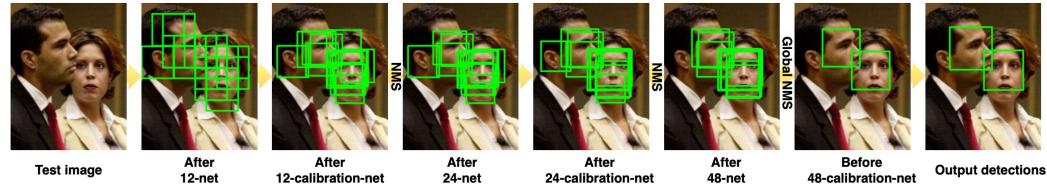
The amount of computation can be adapted to the complexity of the data

- E.g. a complex image requires more computation than a simple image
- Objective: trade-off between accuracy and computation



# Conditional Computation | Cascade

- Use a cascade of classifiers on images at different resolutions
- When a classifier is not confident enough, feed to the next one in higher resolution
- There is another network that refines the position of the bounding boxes
- Detection at 100 img/s

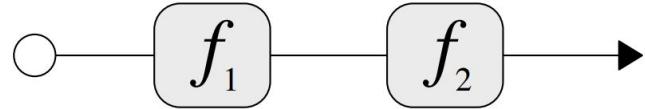


[“A convolutional Neural Network Cascade for Face Detection”, Haoxiang et al. CVPR’15]

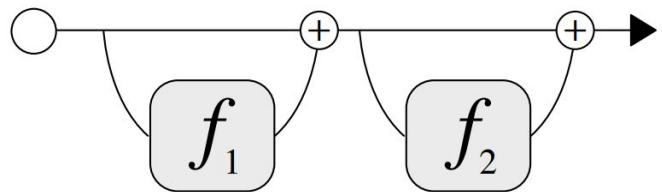
# Adaptive ResNet

- Skip-connections are used to avoid computations
- A gate mechanism learns whether a module should be skipped
  - The decision is discrete (probabilistic)
- Gradients are propagated with a “straight-through” estimator

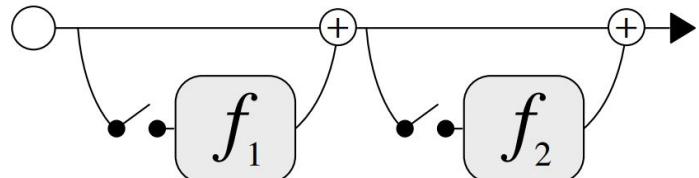
Traditional feed-forward ConvNet:



ResNet:

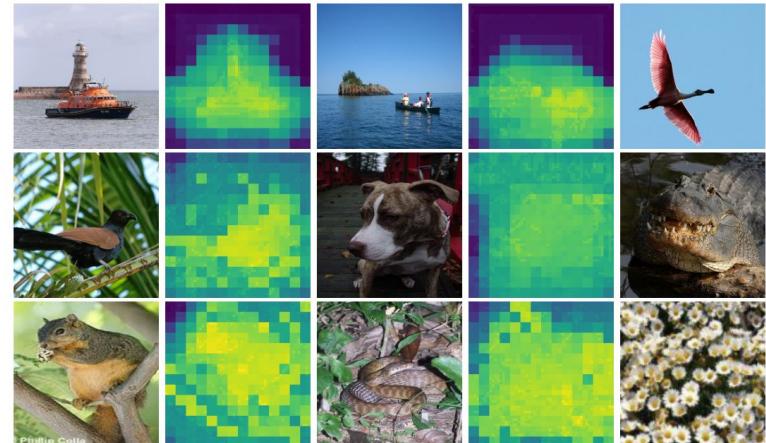


ConvNet-AIG:



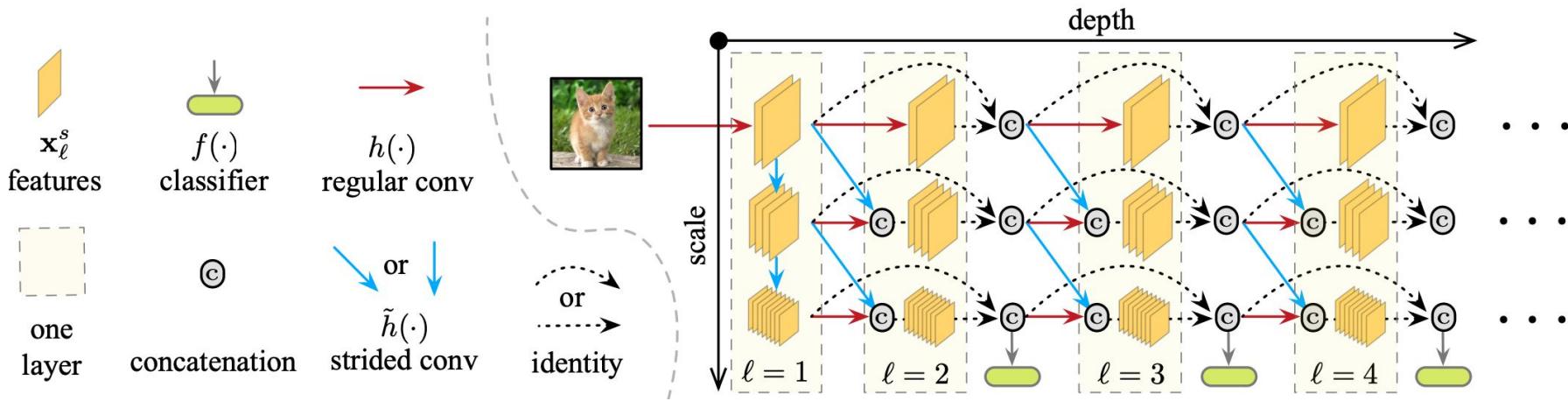
# Spatially adaptive computation

- The network has a “computation budget” that has to spend wisely on different spatial positions of the image
- Since computation becomes sparse, the improvement is limited (~50%)
- The network looks at the image similar to a human



[“Spatially Adaptive Computation Time for Residual Networks” Figurnov et al. CVPR’17]

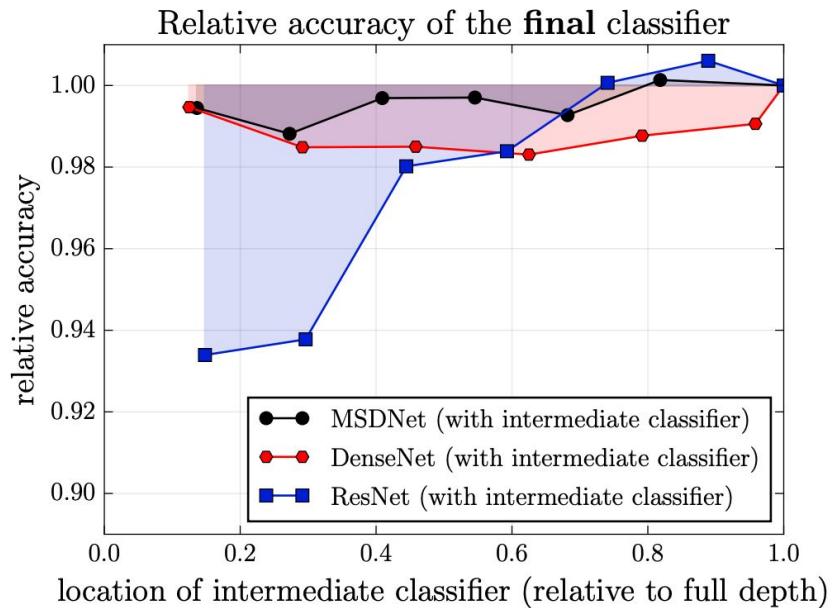
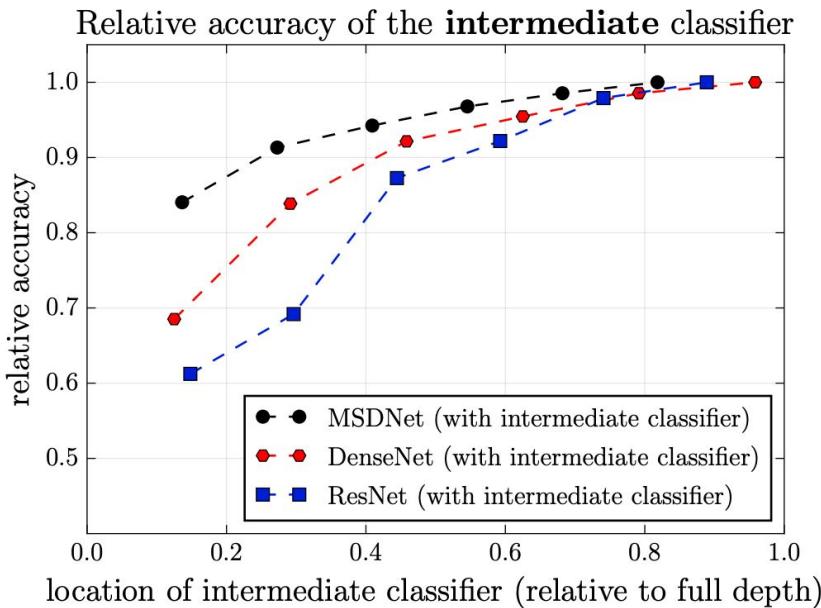
# Multi-scale Dense Network



- Cascade of classifiers
- After each classifier one decides if stop the computation
- “Anytime classification”
- Limited budget classification

[Huang et al ICLR'18]

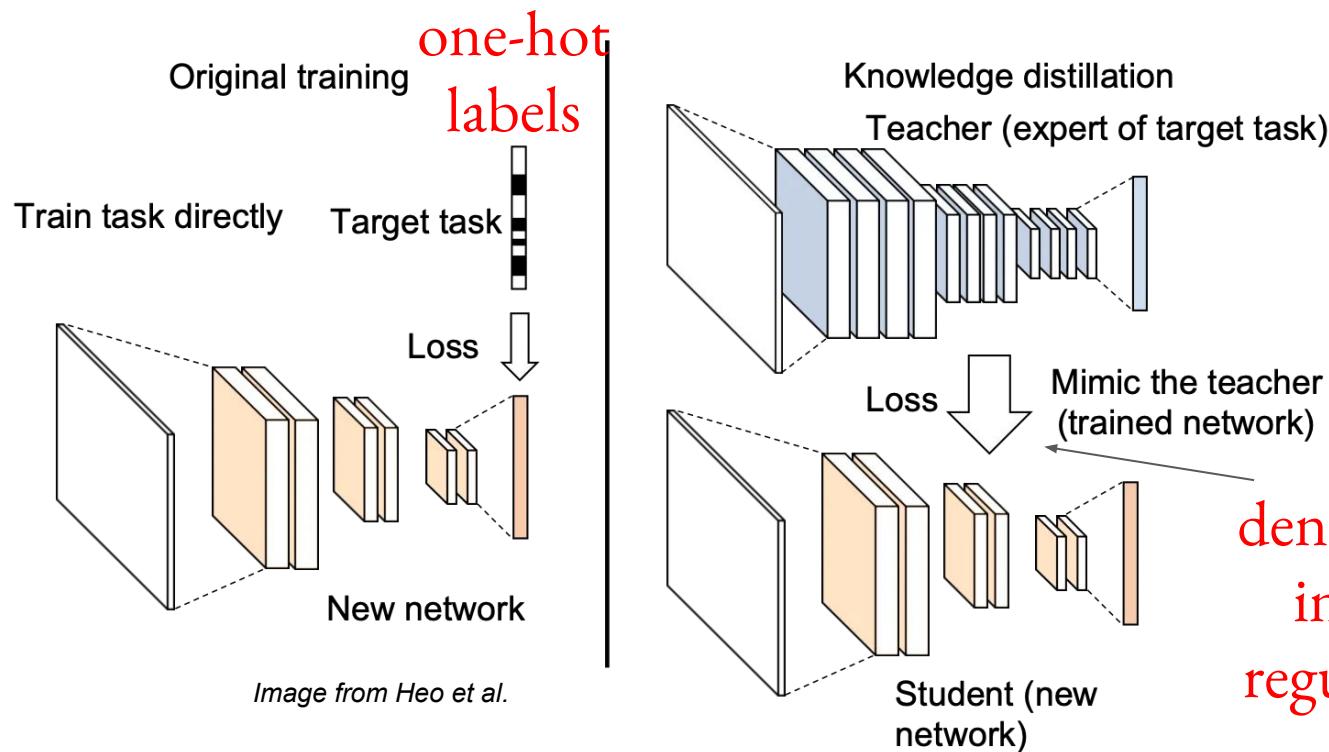
# Multi-scale Dense Network



# Outline

- Motivation
- Weight pruning
- Low-rank approximation
- Quantization
- Conditional / adaptive computation
- **Distillation**
- Architectures

# Distillation



**dense labels!!!!**  
**intuition:**  
**regularization**

# Distillation

- A (big) teacher model is first trained on a task
- Then, a (smaller) student learns to mimic the outputs of the teacher
- Can achieve better accuracy!!!

System	Test Frame Accuracy	WER
Baseline	58.9%	10.9%
10xEnsemble	61.1%	10.7%
Distilled Single model	60.8%	10.7%

# Distillation | Dark knowledge

cow	dog	cat	car
0	1	0	0

original hard targets

cow	dog	cat	car
$10^{-6}$	.9	.1	$10^{-9}$

output of geometric ensemble

cow	dog	cat	car
.05	.3	.2	.005

softened output of ensemble

## Temperature

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

# Outline

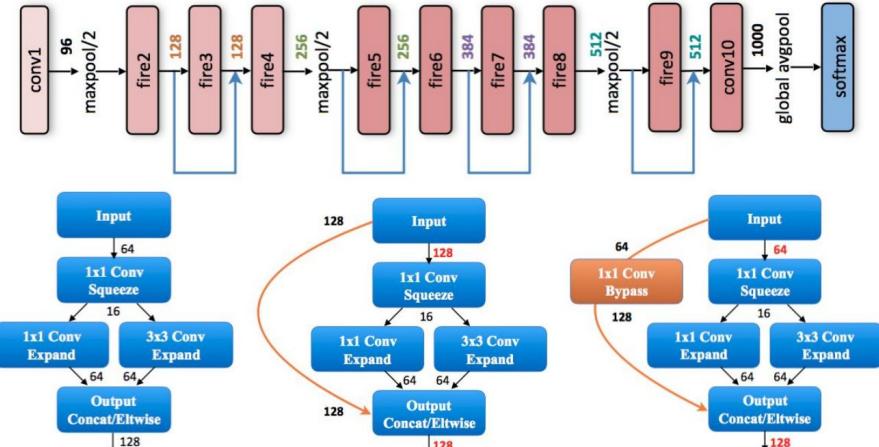
- Motivation
- Weight pruning
- Low-rank approximation
- Quantization
- Conditional / adaptive computation
- Distillation
- **Architectures**

# Architectures

- Instead of optimizing an existing architecture, **find ways to directly create more efficient architectures:**
  - Thinner
  - More efficient layers (without low-rank approx.)

# SqueezeNet

- Use 1x1 convolutions to reduce the number of channels
- Use 1x1 and 3x3 convolutions to recover the number of channels
- Move pooling to the end to improve performance



[Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016]

# SqueezeNet

- The target is to reduce the amount of parameters
- 50x less parameters than AlexNet at the **same error rate**
- **510x reduction with compression**

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	<b>50x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	<b>363x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	<b>510x</b>	57.5%	80.3%

[Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016]

# Mobilenet

Reduces the number of parameters and it is **more efficient!**

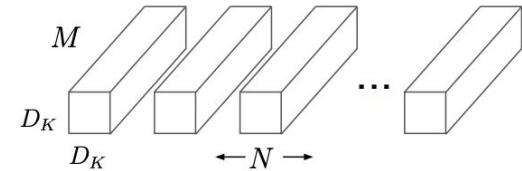
- New decomposition of the convolution
  - Depth-wise filters before
  - 1x1 convolution after
- 9x Faster than a 3x3 convolution at comparable performance

Table 9. Smaller MobileNet Comparison to Popular Models

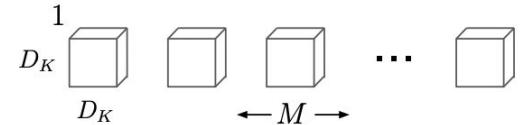
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
SqueezeNet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Table 2. Resource Per Layer Type

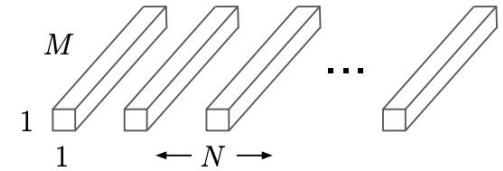
Type	Mult-Adds	Parameters
Conv $1 \times 1$	94.86%	74.59%
Conv DW $3 \times 3$	3.06%	1.06%
Conv $3 \times 3$	1.19%	0.02%
Fully Connected	0.18%	24.33%



(a) Standard Convolution Filters



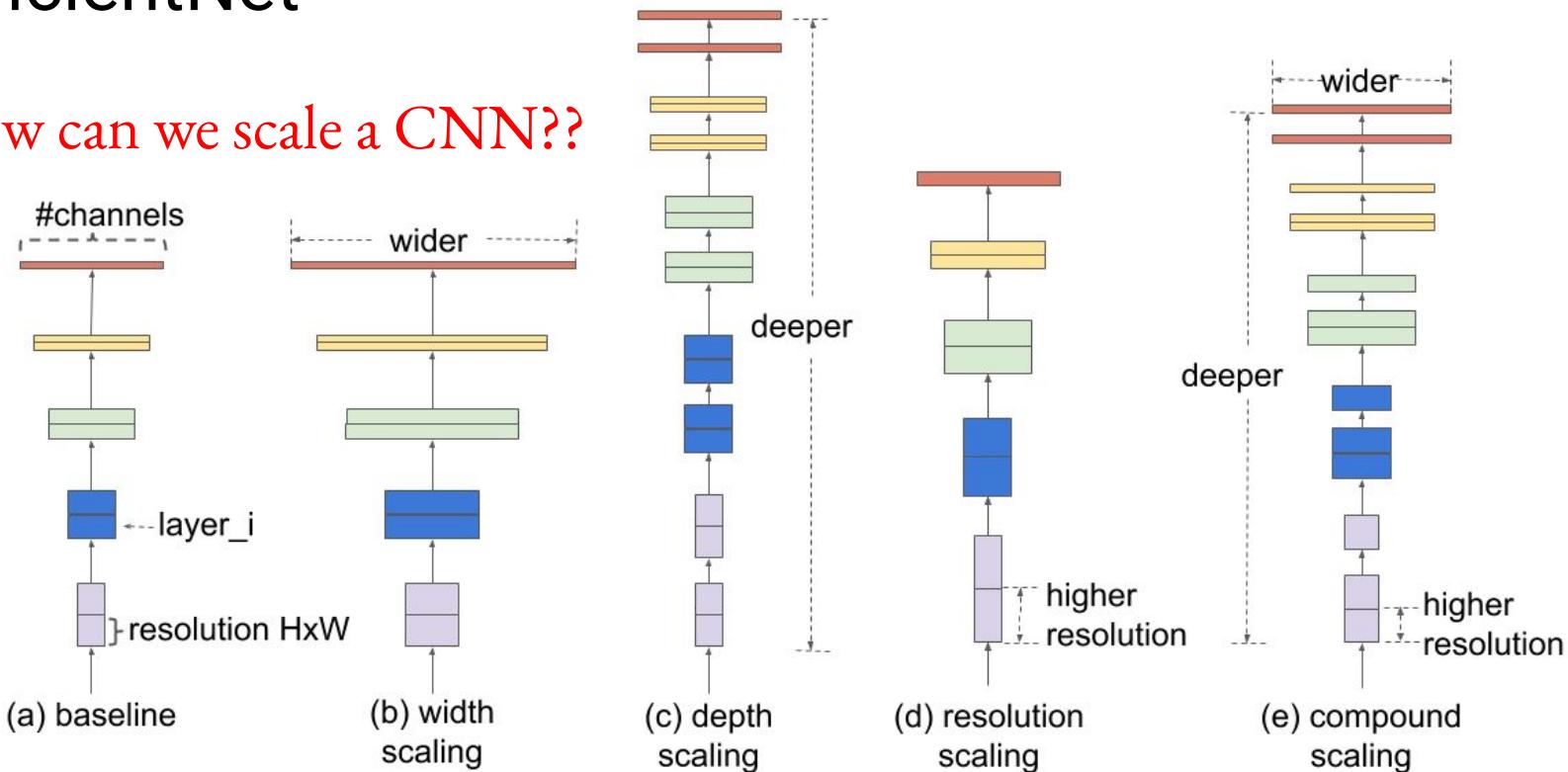
(b) Depthwise Convolutional Filters



(c)  $1 \times 1$  Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

# EfficientNet

# How can we scale a CNN??



[Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks.]

# EfficientNet

- Architecture found by NAS
  - FLOPS included in the optimization!
- + Proper scaling  
Found by hyperparameter search

$$\text{depth: } d = \underline{\alpha^\phi}$$

$\Phi$  = amount of resources

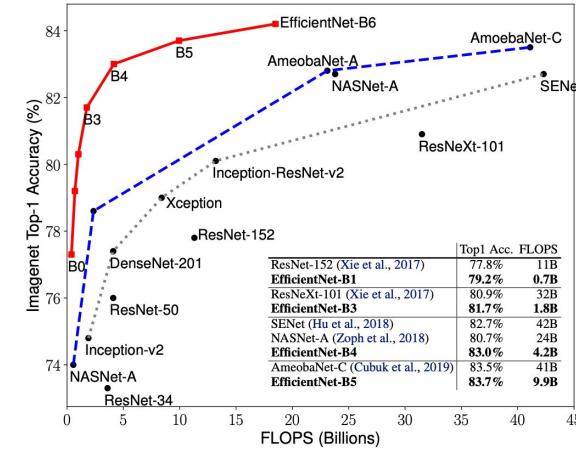
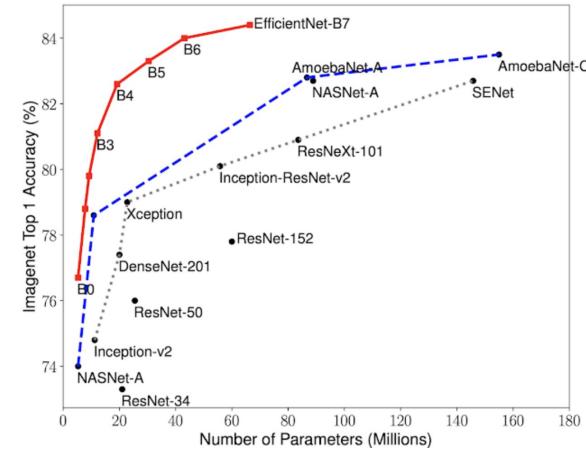
$$\text{width: } w = \underline{\beta^\phi}$$

$$\text{FLOPS} = 2^\Phi$$

$$\text{resolution: } r = \underline{\gamma^\phi}$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



# Thanks!