Network Compression (**Zhuo Su**)

Monday

RNN, LSTM and Applications (**Changchong Sheng**)

Tuesday

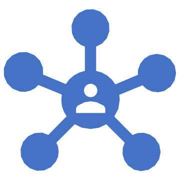
Generative Adversarial Networks (GANs) **(Lam Huynh)**

Wednesday

25.11.2019

# Network Compression

------ Zhuo Su



What is network compression?

Why we need to compress the network?

How to compress network?

What is network compression?

What is network?

What is compression?

Deep Neural Network

Data → → Targets

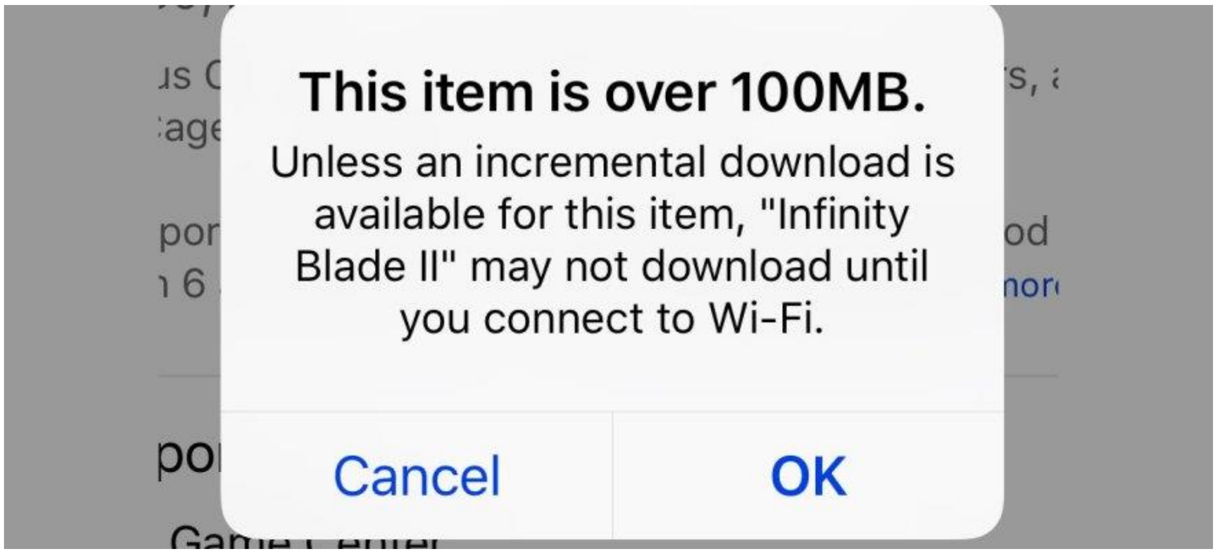
Dog

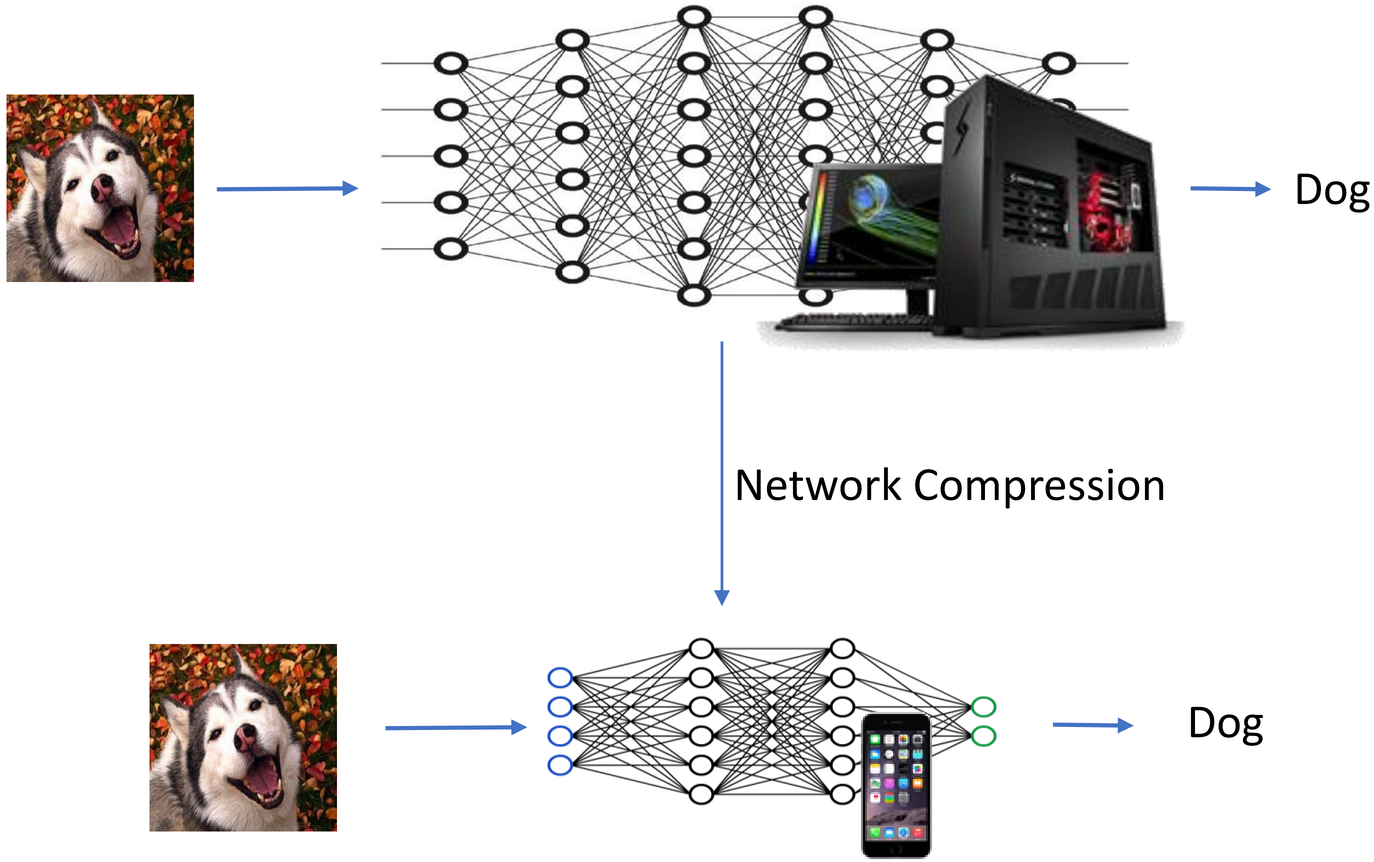
# Deep Neural Network

1. Limited computation power

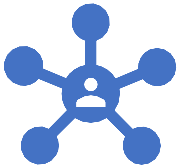
2. Limited storage and memory

3. Limited battery capacity


This item is over 100MB.
Unless an incremental download is available for this item, "Infinity Blade II" may not download until you connect to Wi-Fi.
Cancel
OK


Dog
Network Compression
Dog

- ☑ What is network compression?
  - ☑ What is network?
  - ☑ What is compression?
- ☑ Why we need to compress the network?



## How to compress the deep network?

How to compress network?

# Outline

- Network Pruning
- Knowledge Distillation
- Parameter Quantization
- Architecture Design
- Dynamic Computation

# Network Pruning

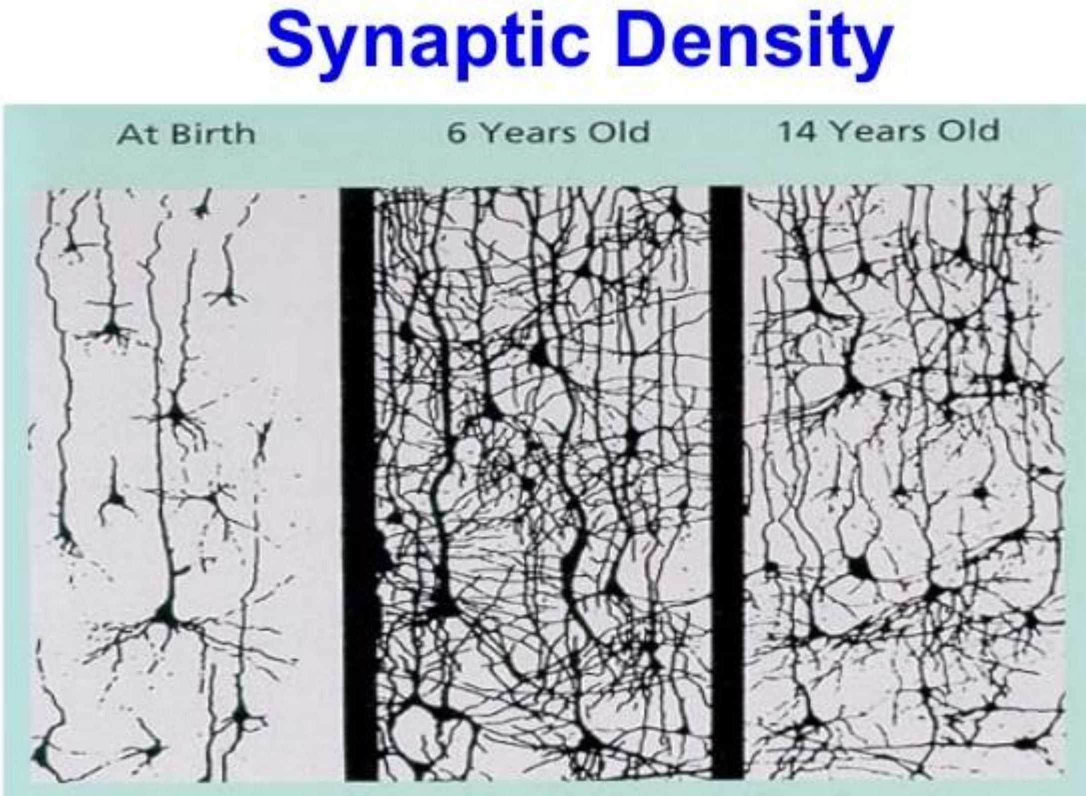
Network Pruning

- Networks are typically over-parameterized (there is significant redundant weights or neurons)
- Prune them!

1990

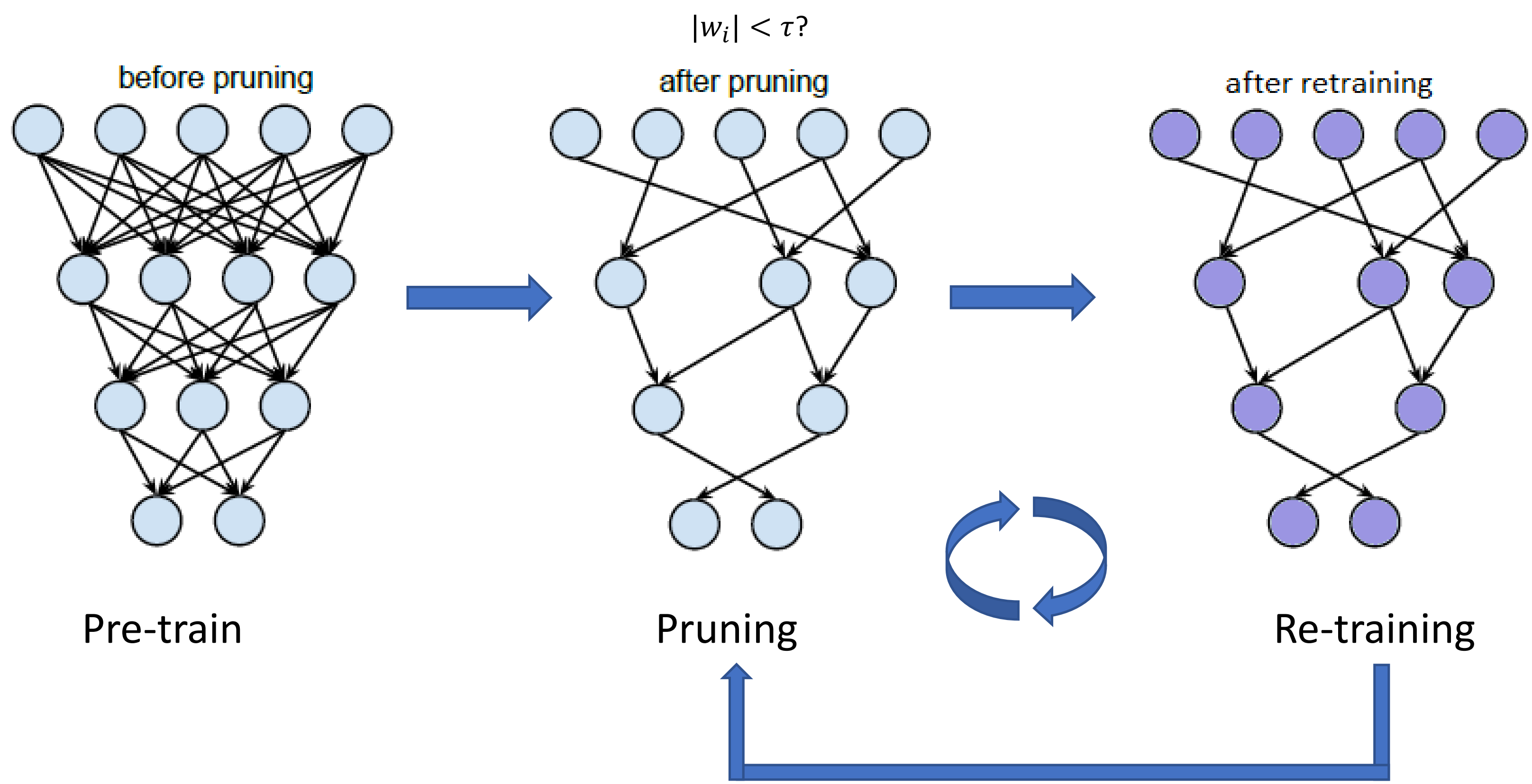
*Optimal Brain Damage*

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733


Synaptic Density
At Birth
6 Years Old
14 Years Old

# Network Pruning

Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015.



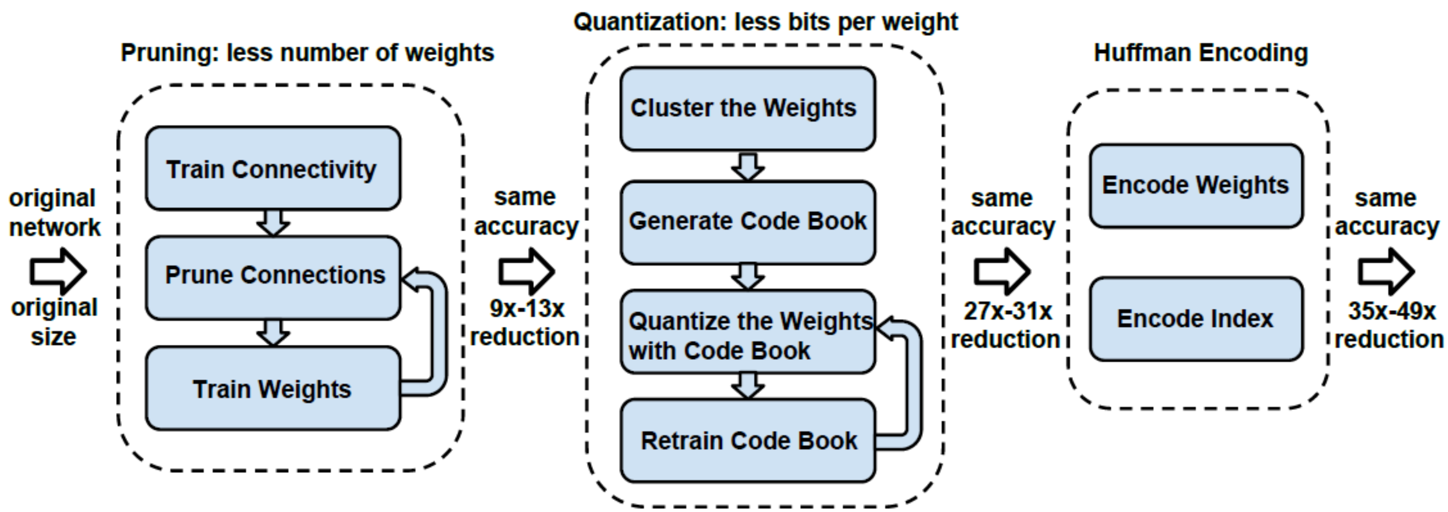
# Network Pruning

| Network | Top-1 Error | Top-5 Error | Parameters | Compression Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 267K | |
| LeNet-300-100 Pruned | 1.59% | - | **22K** | **12**× |
| LeNet-5 Ref | 0.80% | - | 431K | |
| LeNet-5 Pruned | 0.77% | - | **36K** | **12**× |
| AlexNet Ref | 42.78% | 19.73% | 61M | |
| AlexNet Pruned | 42.77% | 19.67% | **6.7M** | **9**× |
| VGG-16 Ref | 31.50% | 11.32% | 138M | |
| VGG-16 Pruned | 31.34% | 10.88% | **10.3M** | **13**× |

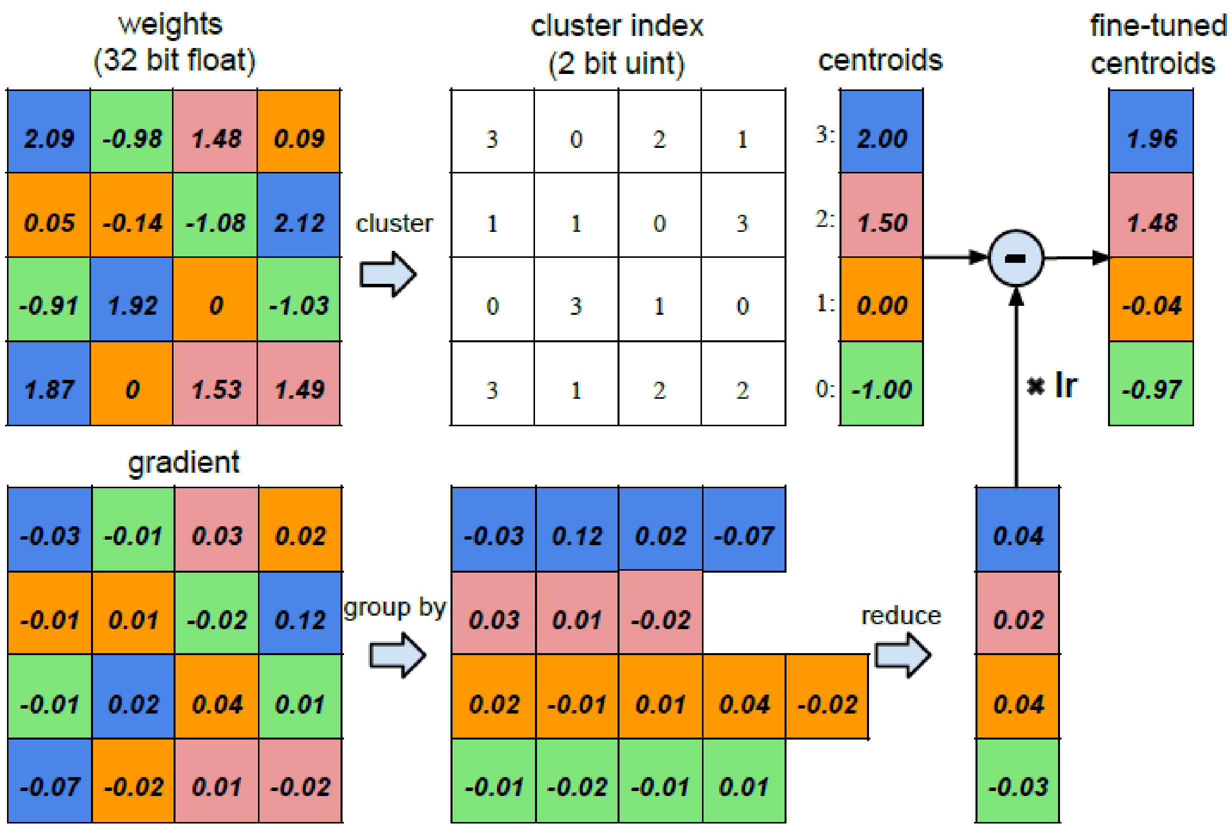
Experiments on ImageNet

# Network Pruning

Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, ICLR 2016.



# Network Pruning



## Network Pruning

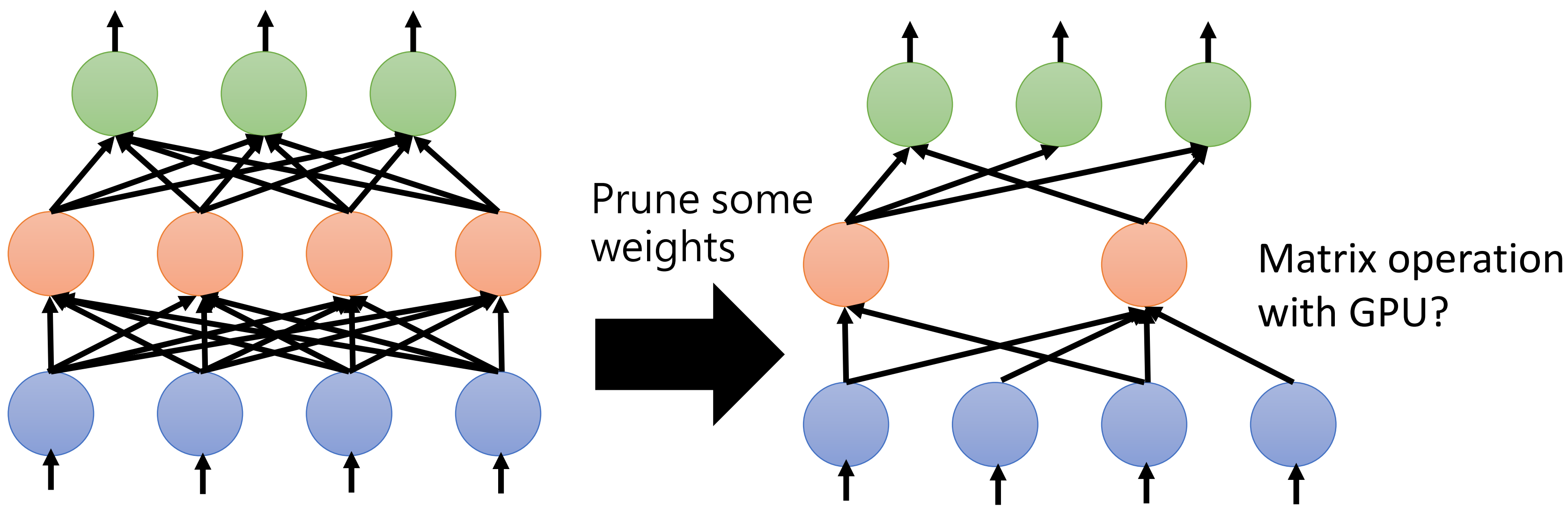
| Network | Top-1 Error | Top-5 Error | Parameters | Compress Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 1070 KB | |
| LeNet-300-100 Compressed | 1.58% | - | **27 KB** | 40× |
| LeNet-5 Ref | 0.80% | - | 1720 KB | |
| LeNet-5 Compressed | 0.74% | - | **44 KB** | **39**× |
| AlexNet Ref | 42.78% | 19.73% | 240 MB | |
| AlexNet Compressed | 42.78% | 19.70% | **6.9 MB** | **35**× |
| VGG-16 Ref | 31.50% | 11.32% | 552 MB | |
| VGG-16 Compressed | 31.17% | 10.91% | **11.3 MB** | **49**× |

Experiments on ImageNet

## Network Pruning

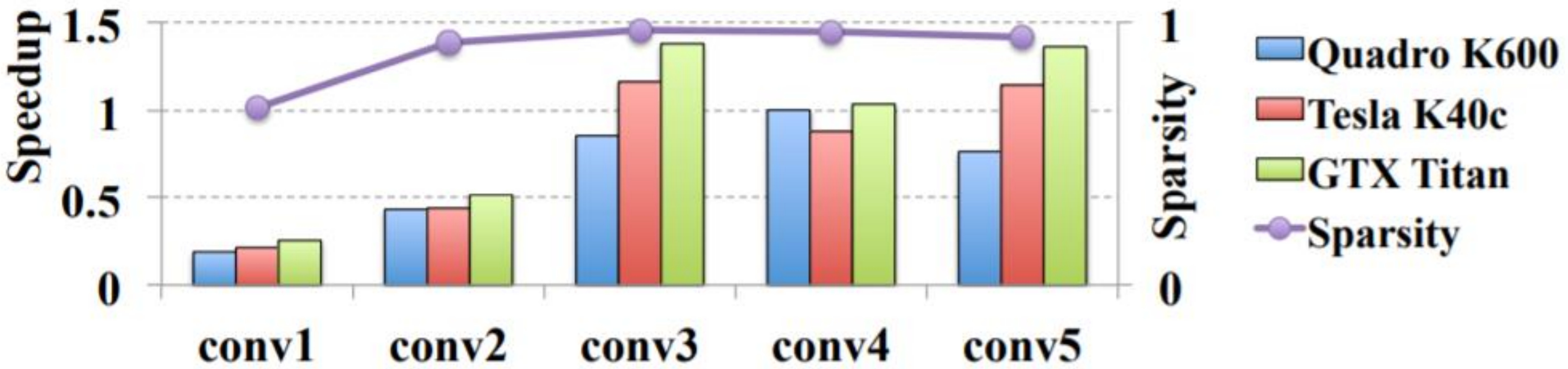
### Weight pruning

The network architecture becomes irregular.




**Hard to implement, hard to speedup ……**

Weight pruning



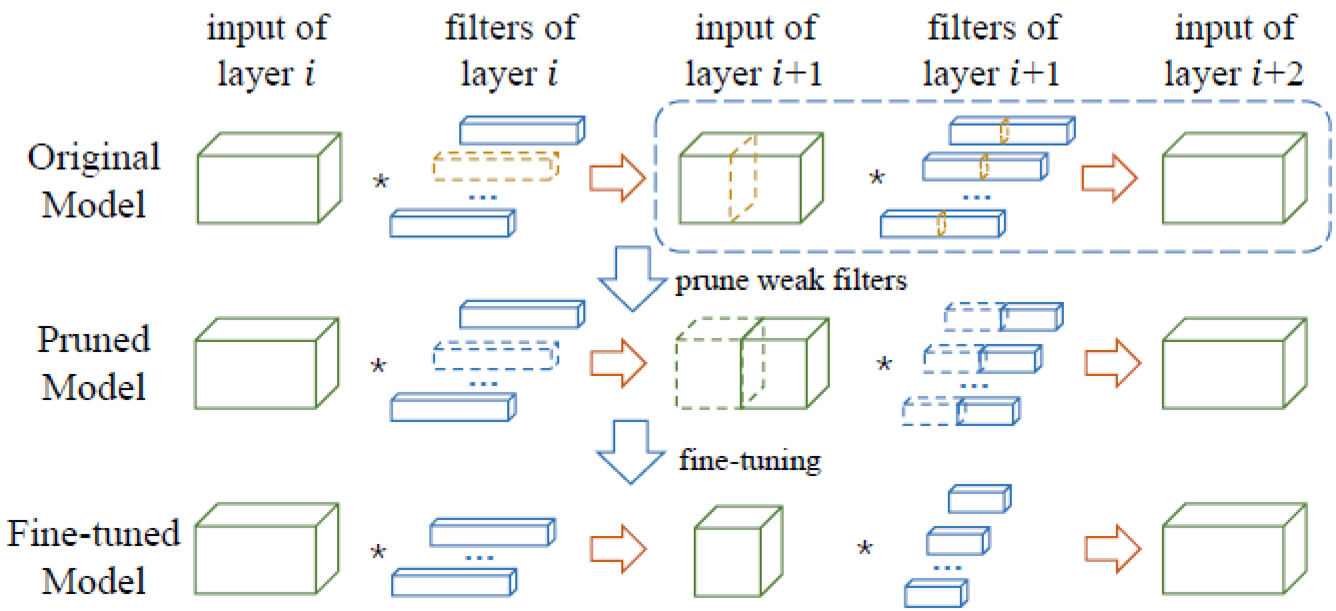

https://arxiv.org/pdf/1608.03665.pdf

ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression, ICCV 2017

# Prune the whole filter

---ThiNet

# ThiNet


input of
layer i
filters of
layer i
input of
layer i+1
filters of
layer i+1
input of
layer i+2
Original
Model
prune weak filters
Pruned
Model
fine-tuning
Fine-tuned
Model

# ThiNet

| Model | Top-1 | Top-5 | #Param. | #FLOPs[1] | f./b. (ms) |
|---|---|---|---|---|---|
| Original[2] | 68.34% | 88.44% | 138.34M | 30.94B | 189.92/407.56 |
| ThiNet-Conv | 69.80% | 89.53% | 131.44M | 9.58B | 76.71/152.05 |
| Train from scratch | 67.00% | 87.45% | 131.44M | 9.58B | 76.71/152.05 |
| ThiNet-GAP | 67.34% | 87.92% | 8.32M | 9.34B | 71.73/145.51 |

Experiments on ImageNet based VGG-16

# Network Pruning

Rethinking the Value of Network Pruning, ICLR 2019

Scratch-E: the same number of epochs as the large network

Scratch-B: double the number of epochs

| Dataset | Unpruned | Strategy | | Pruned Model |
|---|---|---|---|---|
| ImageNet | VGG-16 | | VGG-Conv | VGG-GAP |
| | 71.03 | Fine-tuned | $-1.23$ | $-3.67$ |
| | 71.51 | Scratch-E | $-2.75$ | $-4.66$ |
| | | Scratch-B | $+\mathbf{0.21}$ | $-\mathbf{2.85}$ |
| | ResNet-50 | | ResNet50-30% | ResNet50-50% |
| | 75.15 | Fine-tuned | $-6.72$ | $-4.13$ |
| | 76.13 | Scratch-E | $-5.21$ | $-2.82$ |
| | | Scratch-B | $-\mathbf{4.56}$ | $-\mathbf{2.23}$ |

Compare with ThiNet

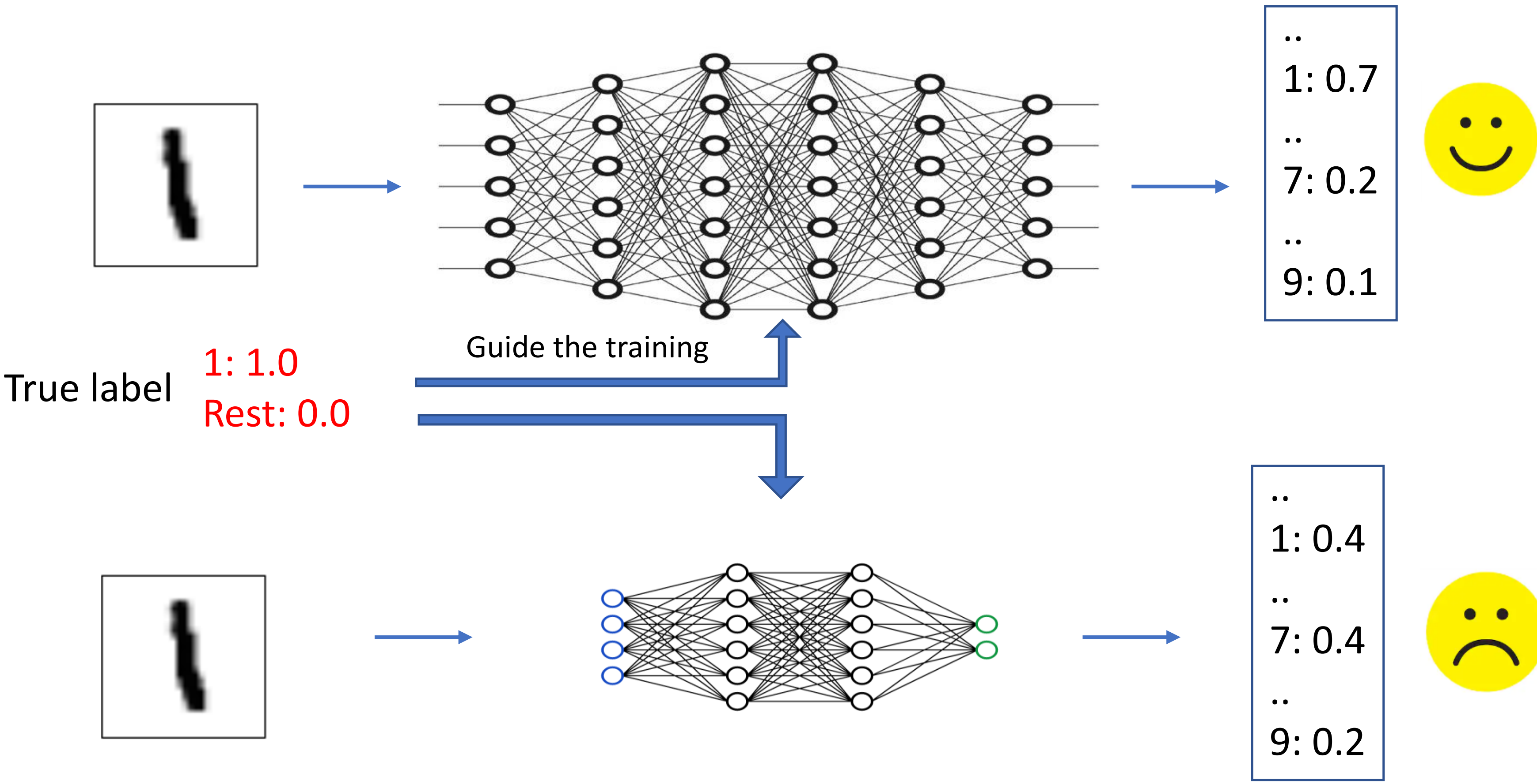
# Knowledge Distillation

# Knowledge Distillation

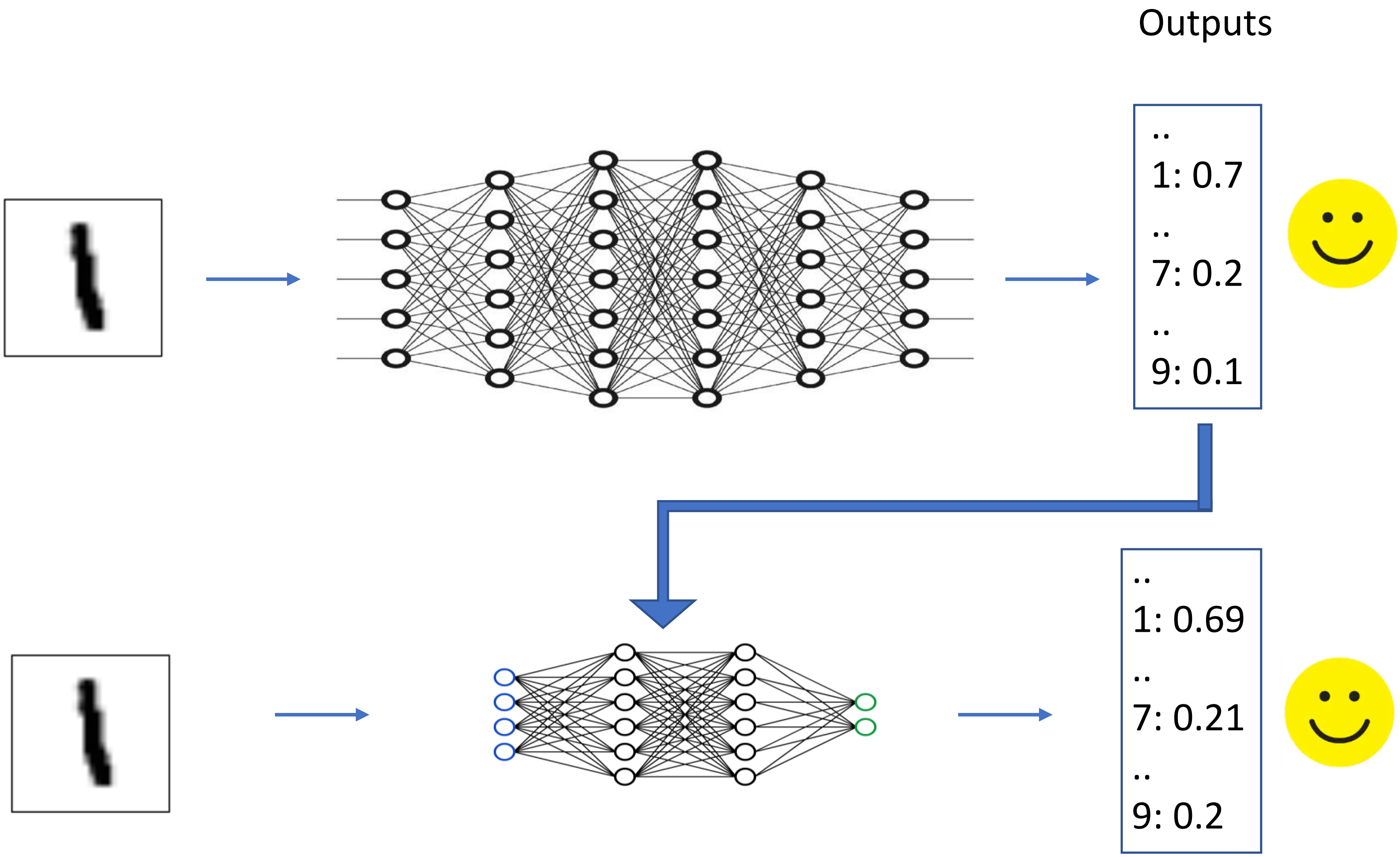
Do Deep Nets Really Need to be Deep? NIPS 2014

Distilling the Knowledge in a Neural Network, arXiv 2015

Outputs

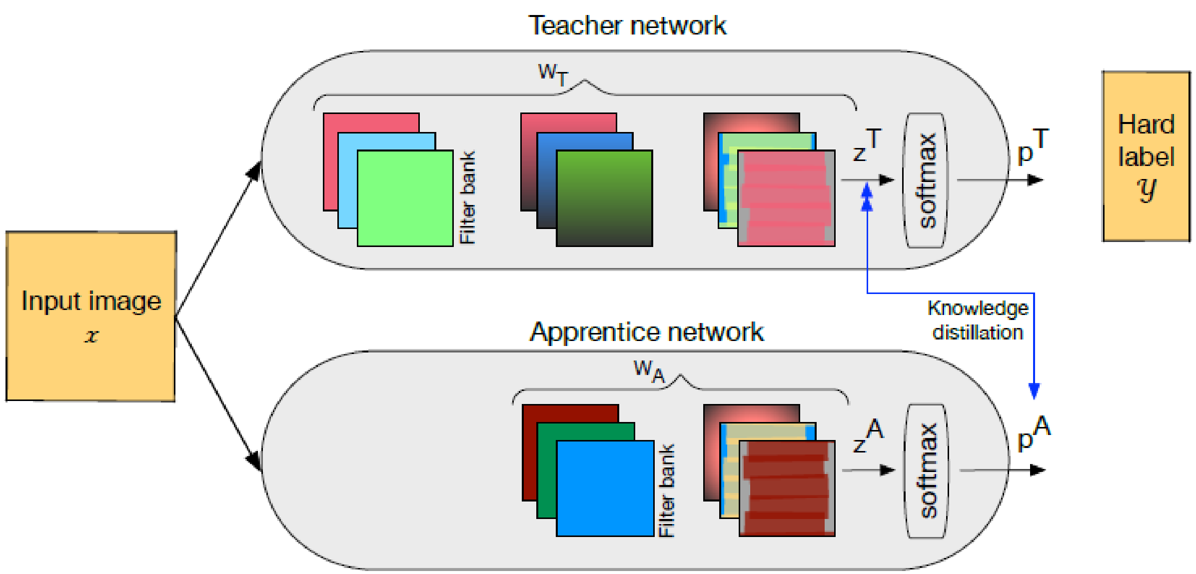


# Knowledge Distillation



# Knowledge Distillation

Apprentice: Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy, ICLR 2018



# Knowledge Distillation

Distilling the Knowledge in a Neural Network, arXiv 2015

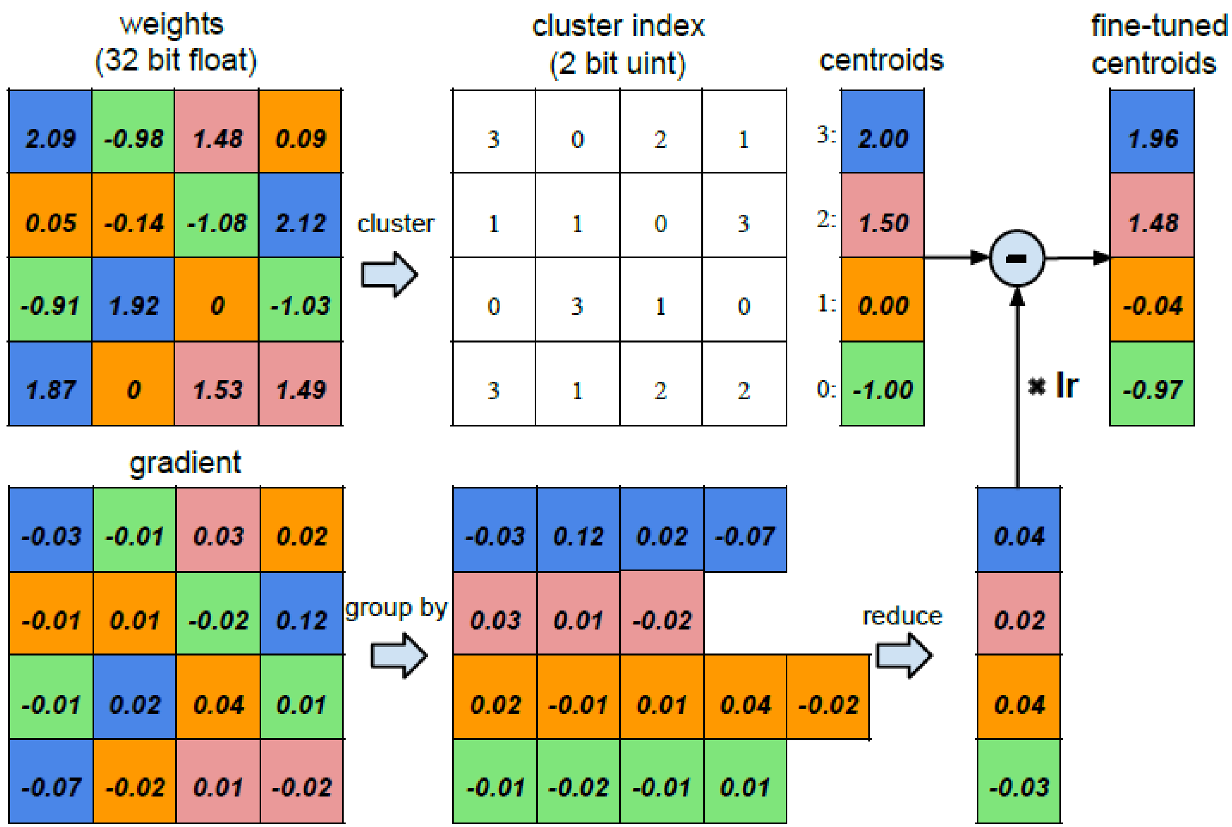
| System | Test Frame Accuracy |
|---|---|
| Baseline | 58.9% |
| 10xEnsemble | 61.1% |
| Distilled Single model | 60.8% |

Experiments on speech recognition

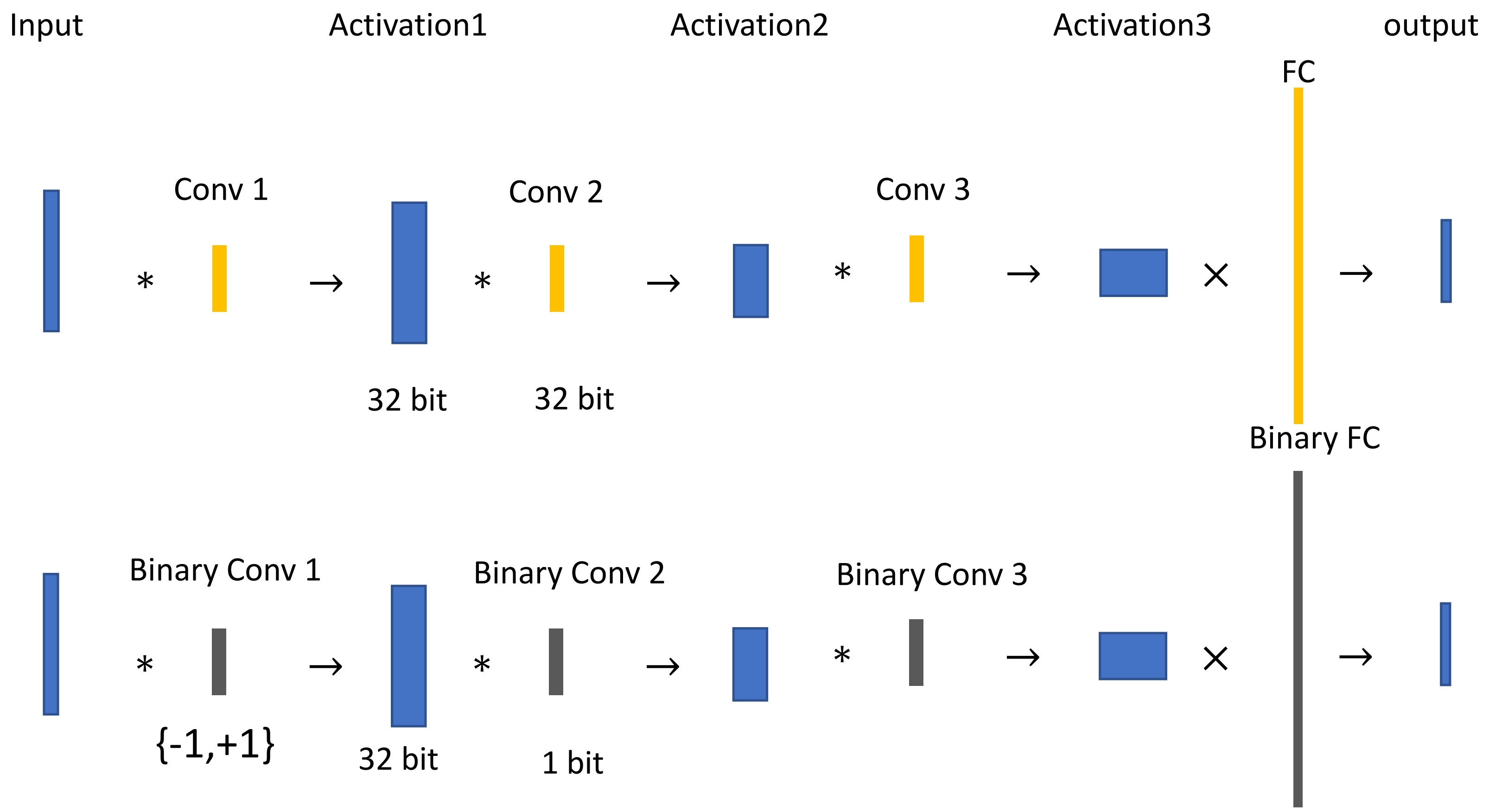
# Parameter Quantization

# Parameter Quantization

## Weight clustering



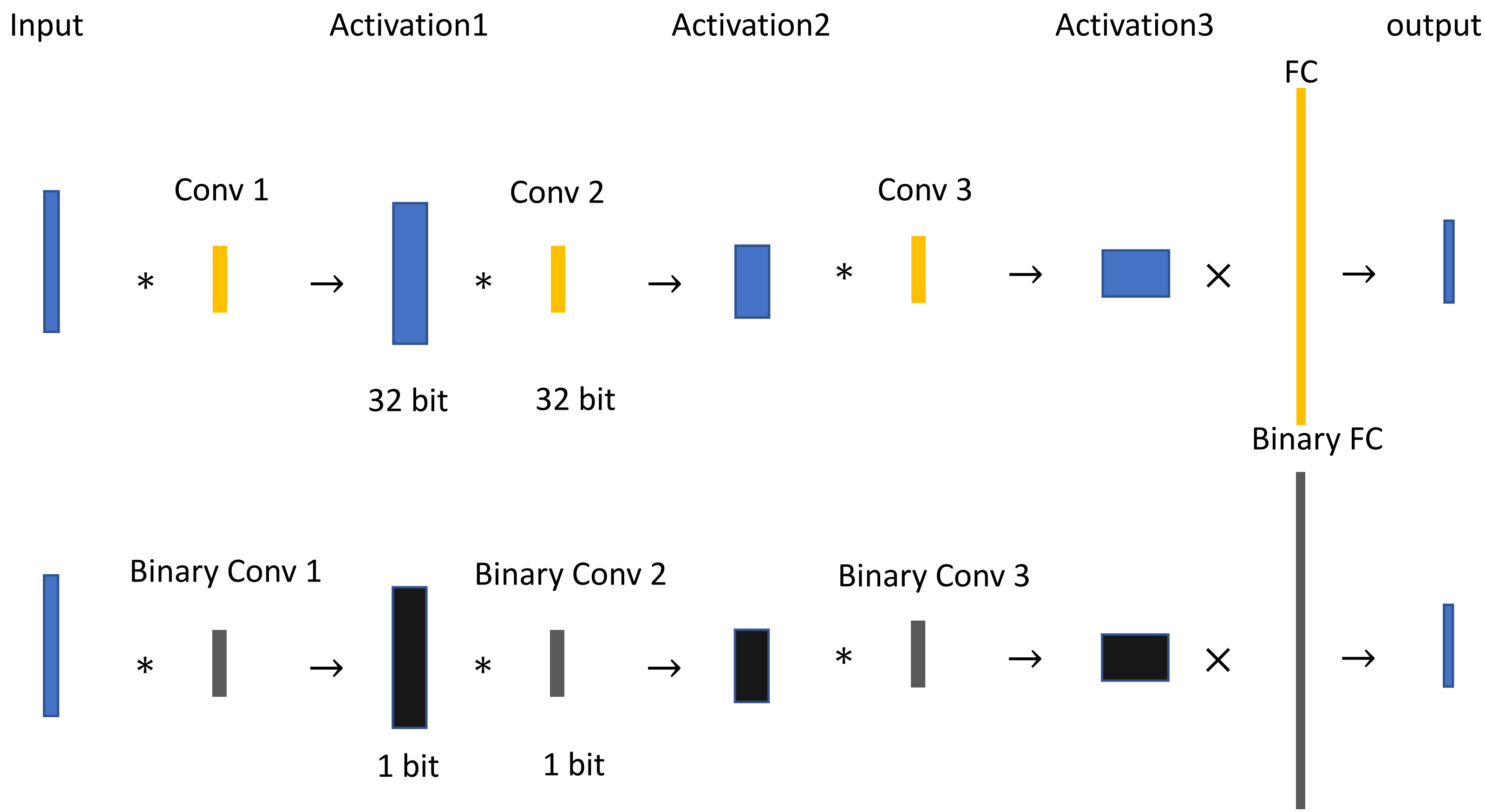
# Parameter Quantization




Multiplication -> Adding or subtraction

Less memory and faster!    ≈ 1/32 of the original model size

# Parameter Quantization




Multiplication -> Bit operation: $\mathbf{x} \cdot \mathbf{y} = \mathrm{bitcount}(\mathrm{and}(\mathbf{x}, \mathbf{y})), x_i, y_i \in \{0, 1\} \, \forall i.$

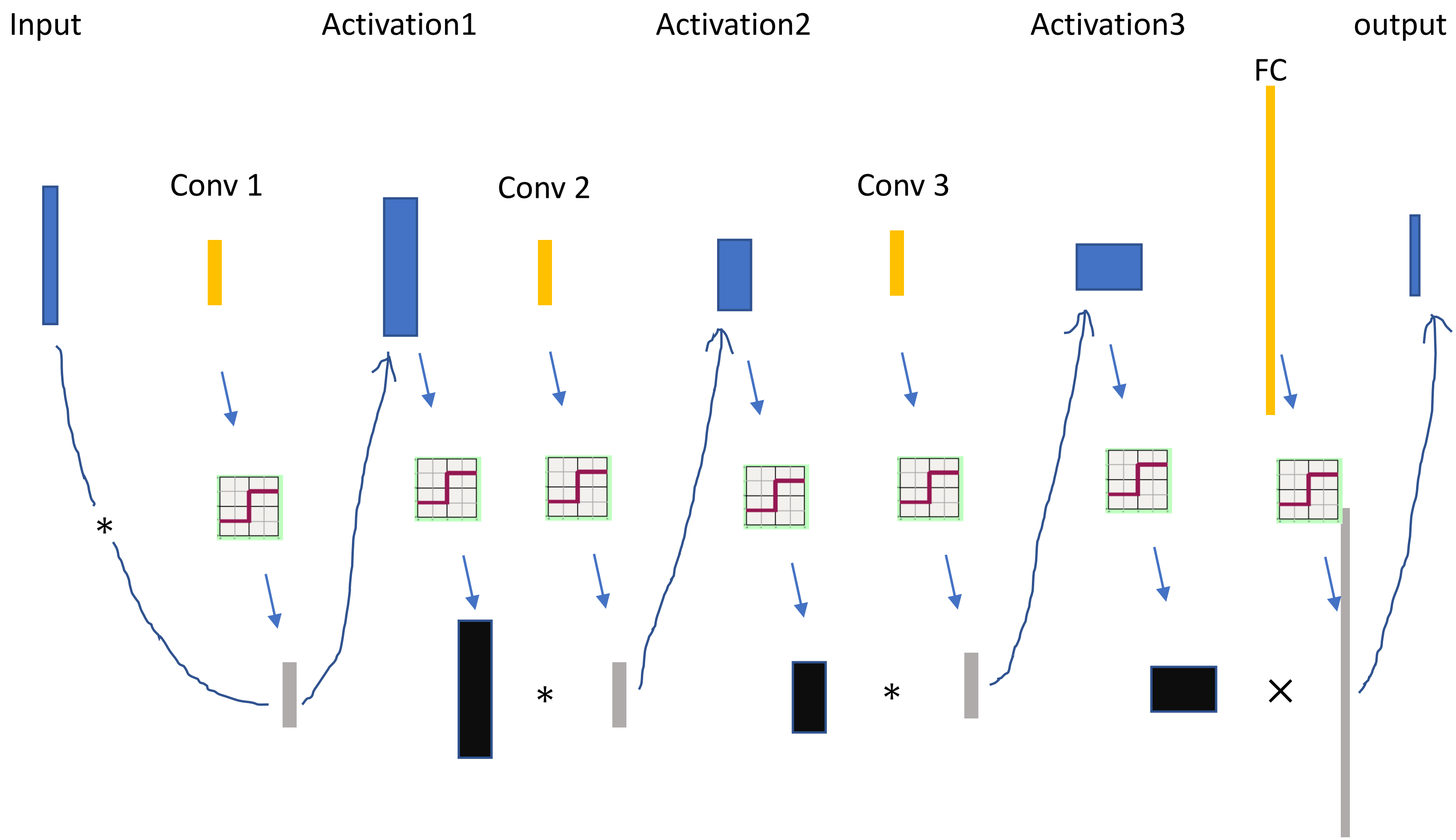
Less memory and faster!! ≈1/32 of the original memory needed

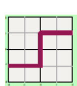
Parameters are discrete

How to train this binary network end to end?

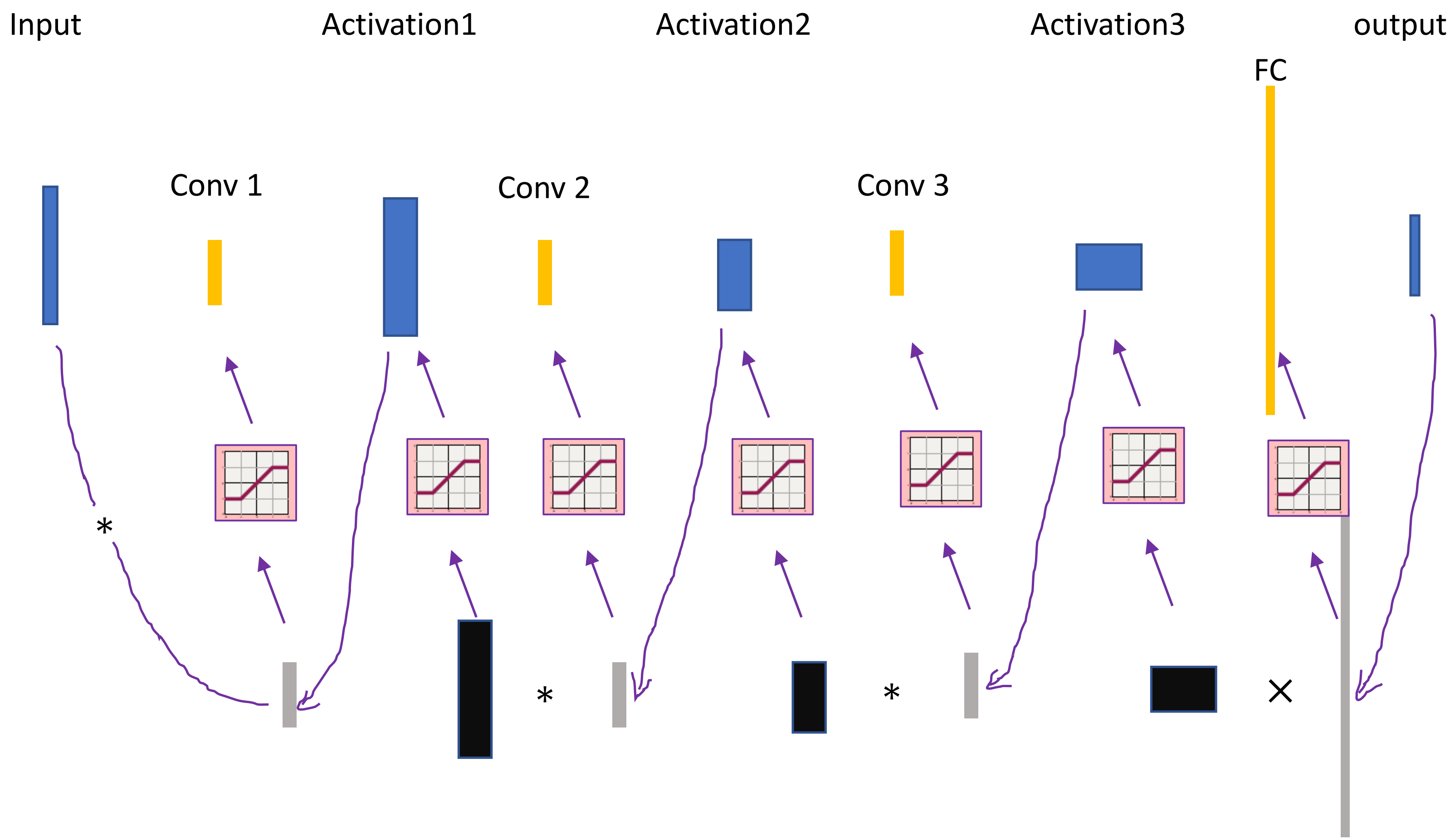
# Parameter Quantization

https://arxiv.org/abs/1602.02830




Forwarding  $w = sign(w), \quad \frac{\partial_{sign(w)}}{\partial_{w}} = 0, verywhere$

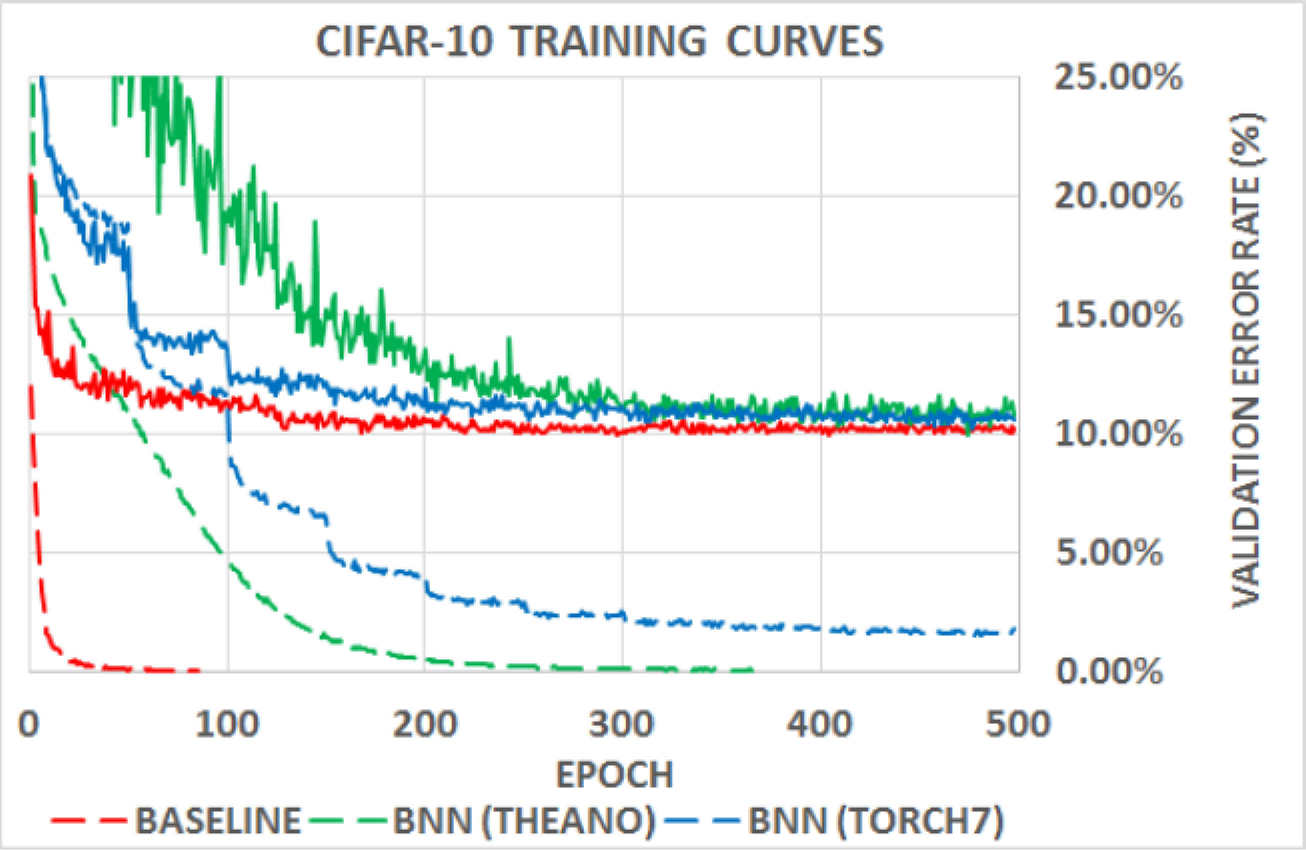
# Parameter Quantization




Backpropagation

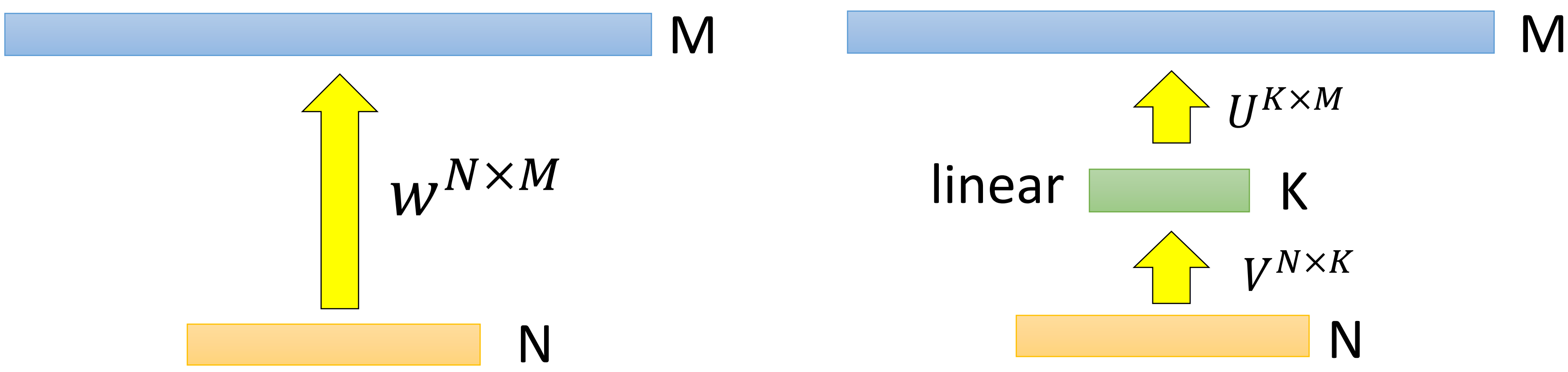
$$\frac{\partial_{sign(w)}}{\partial_w} = \begin{cases} 1 & if\ |w| < 1 \\ 0 & else \end{cases}, not\ everywhere$$

# Parameter Quantization



# Architecture Design

*Fully connected layer…*




$$w^{N \times M} \approx V^{N \times K} \cdot U^{K \times M}$$

Number of parameters:

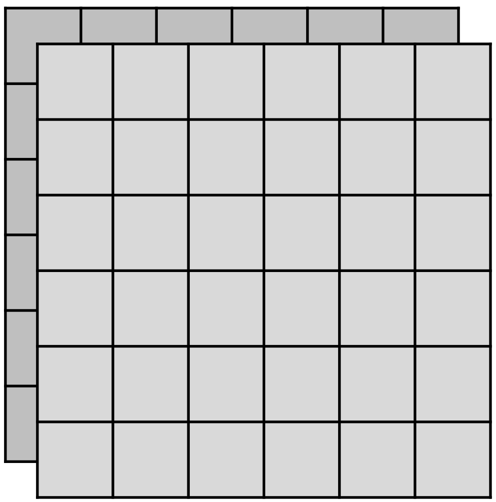
$N \times M$ $\quad\quad$ $K \times (N + M)$ Less parameters

MACC (Multiply-accumulate, how many multiplication operations):

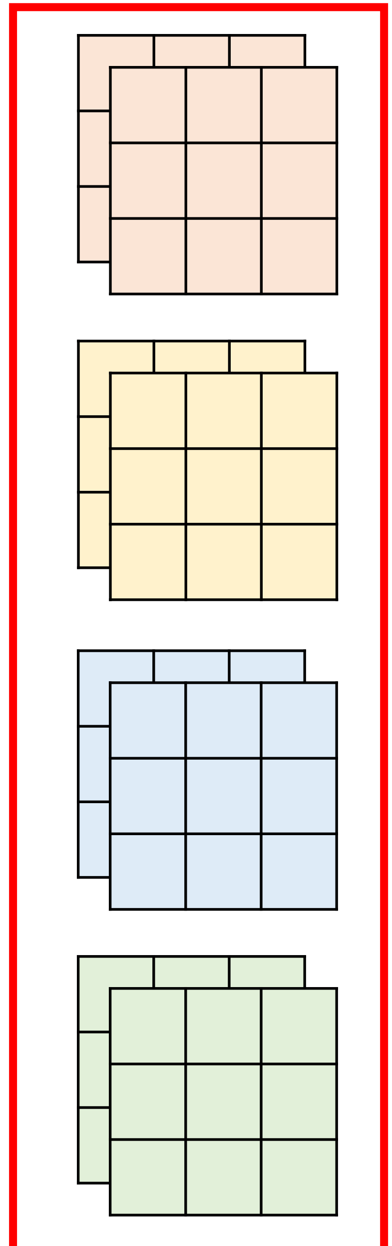
$N \times M$ $\quad\quad$ $K \times (N + M)$ Less MACC

*Convolutional layer…*

Input feature map

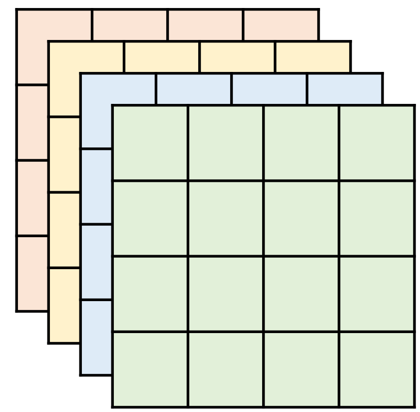


2 channels



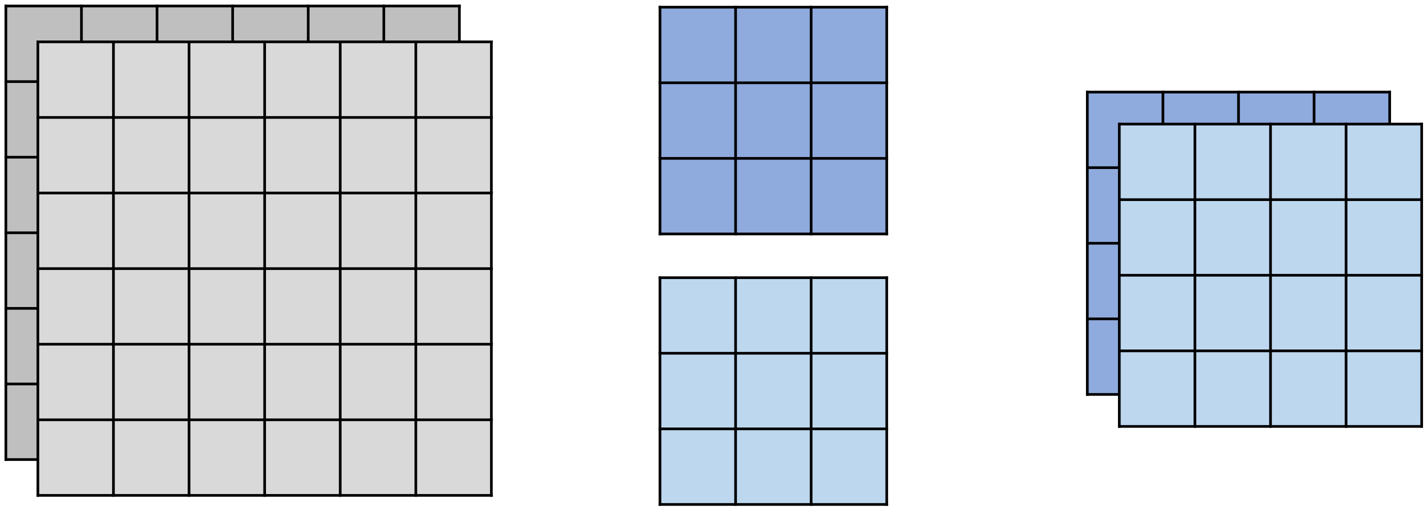
$$3 \times 3 \times 2 \times 4 = 72$$

parameters

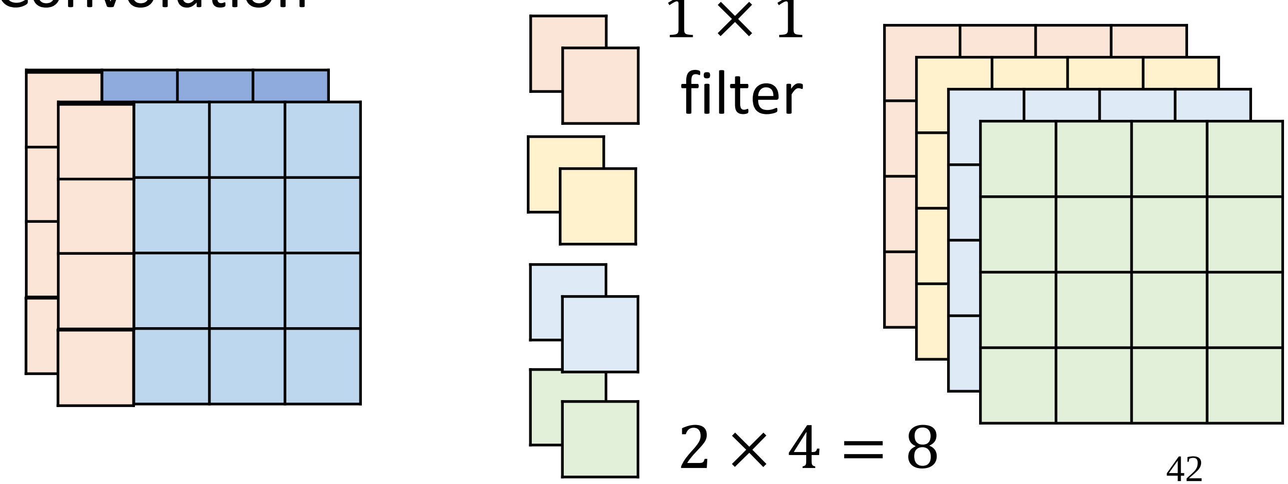


## 1. Depthwise Convolution

$$3 \times 3 \times 2 = 18$$



## 2. Pointwise Convolution

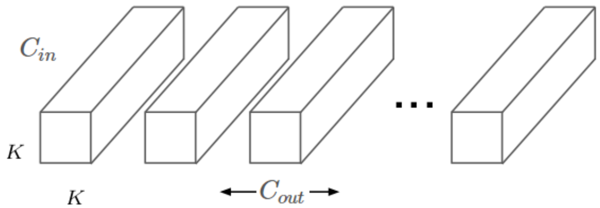


# Architecture Design

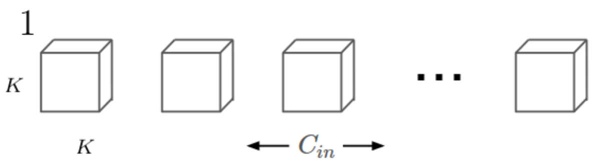
## *Convolutional layer…*

Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

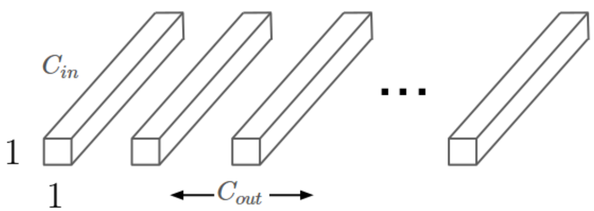
#FLOPS=2*#MACC




(a) Standard Convolution Filters




(b) Depthwise Convolution Filters




(c) Pointwise Convolution Filters

| | Parameters | MACC (Multiply-accumulate) |
|---|---|---|
| 1 | $K \times K \times C_{in} \times C_{out}$ | $K \times K \times C_{in} \times C_{out} \times \mathrm{W} \times \mathrm{H}$ |
| 2 | $K \times K \times 1 \times C_{in}$ | $K \times K \times 1 \times C_{out} \times \mathrm{W} \times \mathrm{H}$ |
| 3 | $1 \times 1 \times C_{in} \times C_{out}$ | $1 \times 1 \times C_{in} \times C_{out} \times \mathrm{W} \times \mathrm{H}$ |

Compression rate: $\dfrac{2 + 3}{1}$ | $\dfrac{1}{C_{out}} + \dfrac{1}{K \times K}$ | $\dfrac{1}{C_{out}} + \dfrac{1}{K \times K}$

# Architecture Design

Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

Table 1. MobileNet Body Architecture

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1<br>Conv / s1 | $3 \times 3 \times 512$ dw<br>$1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$<br>$14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

# Architecture Design

Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| Conv MobileNet | 71.7% | 4866 | 29.3 |
| MobileNet | 70.6% | 569 | 4.2 |
| GoogleNet | 69.8% | 1550 | 6.8 |
| VGG 16 | 71.5% | 15300 | 138 |

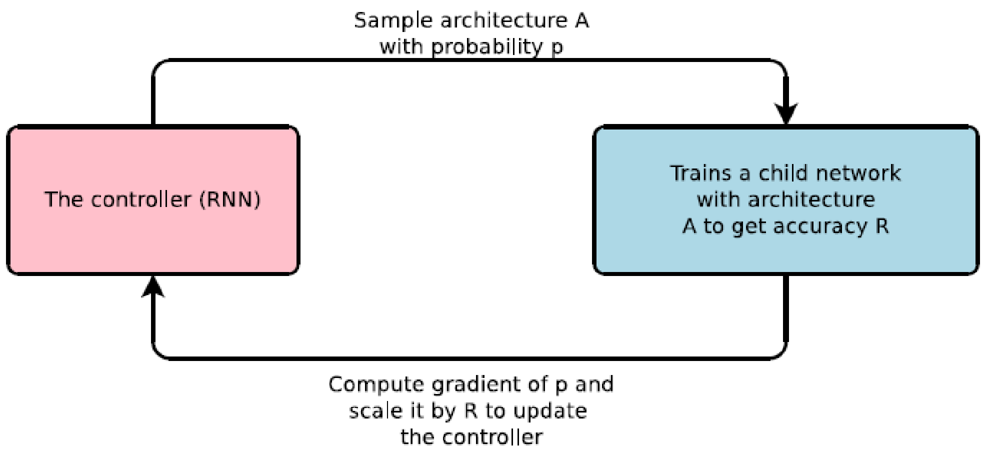
Experiments on ImageNet

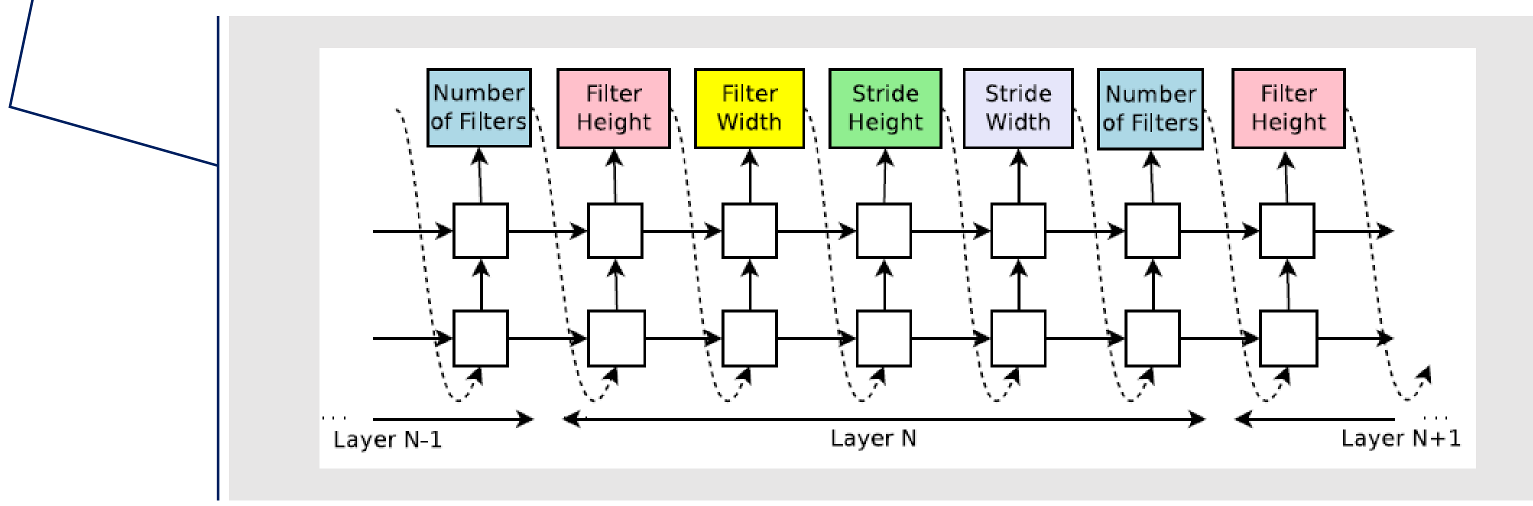
https://zhuogege1943.com/2019/06/16/Going-with-small-and-fast-networks-1/ for more information.

## Architecture Design

*Automatically architecture search…*

Neural Architecture Search with Reinforcement Learning, ICLR 2017



# Architecture Design

Neural Architecture Search with Reinforcement Learning, ICLR 2017

# Architecture Design

Neural Architecture Search with Reinforcement Learning, ICLR 2017

| Model | Depth | Parameters | Error rate (%) |
|---|---|---|---|
| Network in Network (Lin et al., 2013) | - | - | 8.81 |
| All-CNN (Springenberg et al., 2014) | - | - | 7.25 |
| Deeply Supervised Net (Lee et al., 2015) | - | - | 7.97 |
| Highway Network (Srivastava et al., 2015) | - | - | 7.72 |
| Scalable Bayesian Optimization (Snoek et al., 2015) | - | - | 6.37 |
| FractalNet (Larsson et al., 2016) | 21 | 38.6M | 5.22 |
| with Dropout/Drop-path | 21 | 38.6M | 4.60 |
| ResNet (He et al., 2016a) | 110 | 1.7M | 6.61 |
| ResNet (reported by Huang et al. (2016c)) | 110 | 1.7M | 6.41 |
| ResNet with Stochastic Depth (Huang et al., 2016c) | 110 | 1.7M | 5.23 |
| | 1202 | 10.2M | 4.91 |
| Wide ResNet (Zagoruyko & Komodakis, 2016) | 16 | 11.0M | 4.81 |
| | 28 | 36.5M | 4.17 |
| ResNet (pre-activation) (He et al., 2016b) | 164 | 1.7M | 5.46 |
| | 1001 | 10.2M | 4.62 |
| DenseNet ($L = 40, k = 12$) Huang et al. (2016a) | 40 | 1.0M | 5.24 |
| DenseNet($L = 100, k = 12$) Huang et al. (2016a) | 100 | 7.0M | 4.10 |
| DenseNet ($L = 100, k = 24$) Huang et al. (2016a) | 100 | 27.2M | 3.74 |
| DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b) | 190 | 25.6M | 3.46 |
| Neural Architecture Search v1 no stride or pooling | 15 | 4.2M | 5.50 |
| Neural Architecture Search v2 predicting strides | 20 | 2.5M | 6.01 |
| Neural Architecture Search v3 max pooling | 39 | 7.1M | 4.47 |
| Neural Architecture Search v3 max pooling + more filters | 39 | 37.4M | 3.65 |

Experiments on CIFAR-10

Architecture Design

Learn more from the Survey paper:

https://arxiv.org/pdf/1808.05377

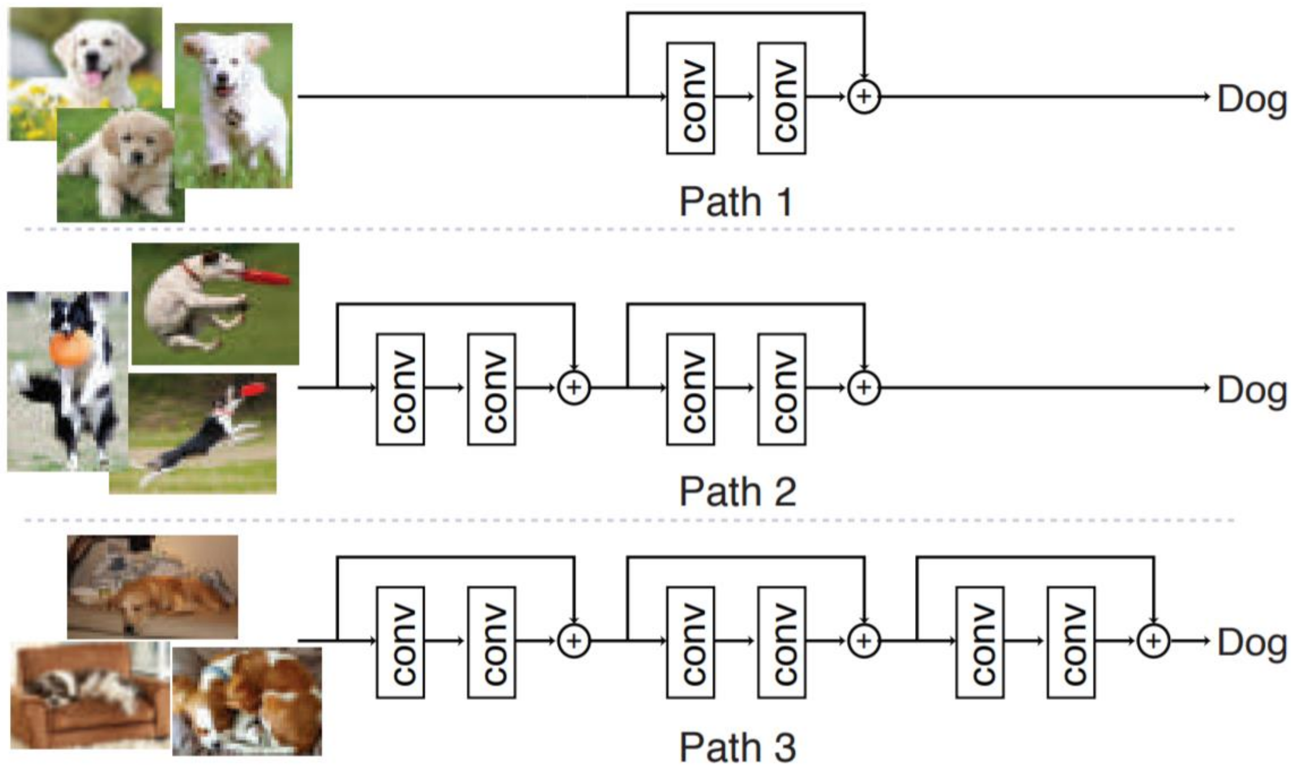
# Dynamic Computation

BlockDrop: Dynamic Inference Paths in Residual Networks, CVPR 2018

## Block-wise dynamic pruning

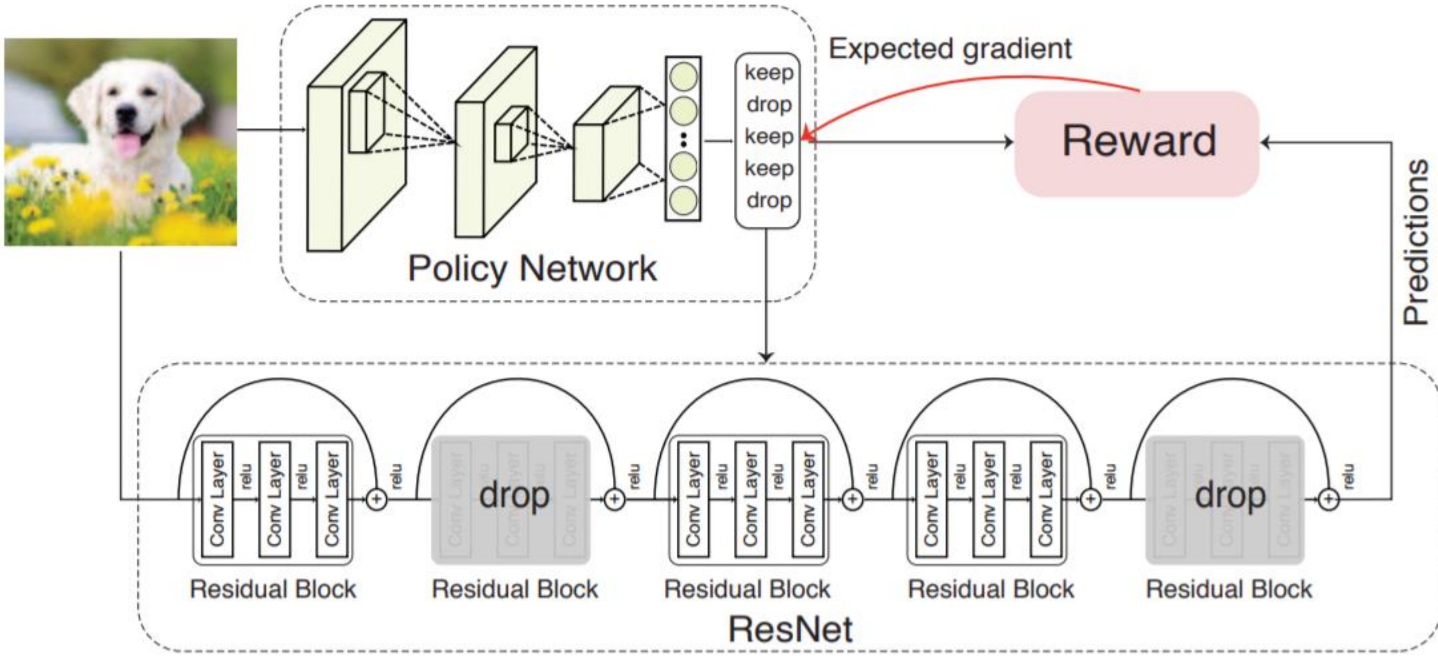
# Why?

Easier samples use fewer blocks



# Dynamic Computation

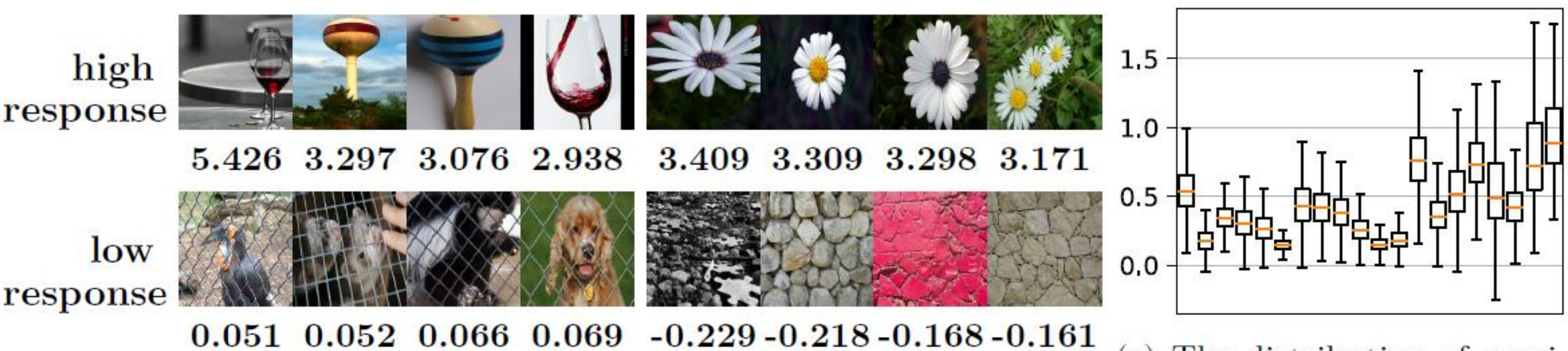
BlockDrop: Dynamic Inference Paths in Residual Networks, CVPR 2018



Dynamic Channel Pruning: Feature Boosting and Suppression, ICLR 2019

## Channel-wise dynamic pruning

# Why?

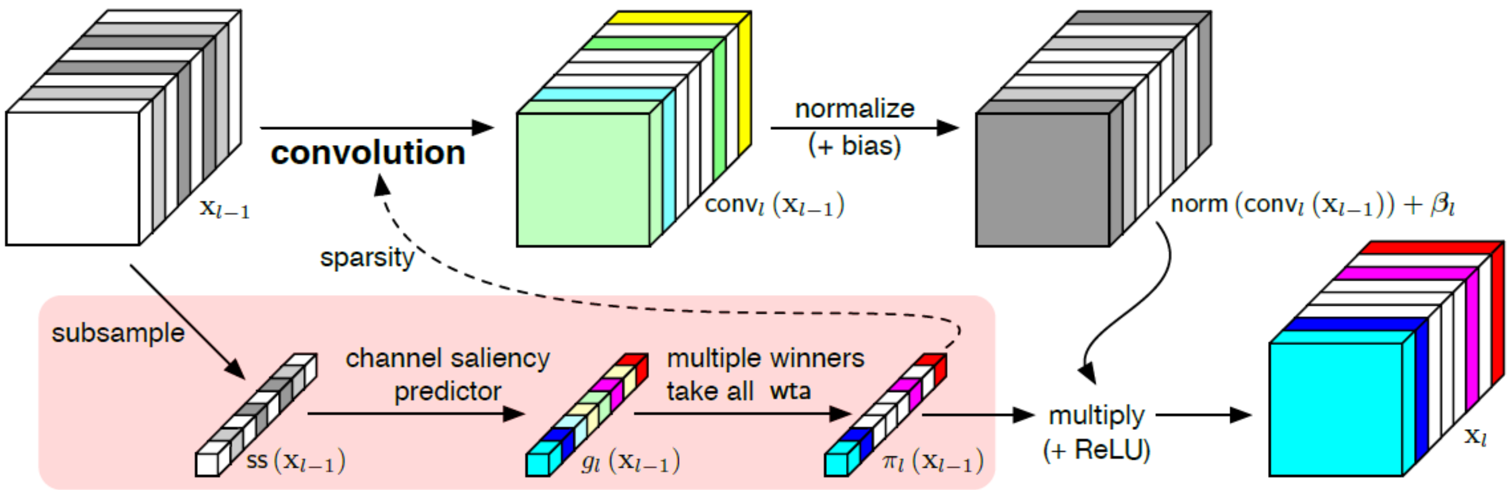



(a) Channel 114

(b) Channel 181

(c) The distribution of maximum activations of the first 20 channels

Dynamic Computation




End to end optimization

| Method | Dynamic | Δ top-5 errors (%) 3× | 4× | 5× |
|---|---|---|---|---|
| *Filter Pruning* (Li et al. (2017), reproduced by He et al. (2017)) | | — | 8.6 | 14.6 |
| *Perforated CNNs* (Figurnov et al., 2016) | | 3.7 | 5.5 | — |
| *Network Slimming* (Liu et al. (2017), our implementation) | | 1.37 | 3.26 | 5.18 |
| *Runtime Neural Pruning* (Lin et al., 2017) | ✓ | 2.32 | 3.23 | 3.58 |
| *Channel Pruning* (He et al., 2017) | | 0.0 | 1.0 | 1.7 |
| *AutoML for Model Compression* (He et al., 2018b) | | — | — | 1.4 |
| *ThiNet-Conv* (Luo et al., 2017) | | 0.37 | — | — |
| *Feature Boosting and Suppression* (FBS) | ✓ | 0.04 | **0.52** | **0.59** |

Experiments on ImageNet

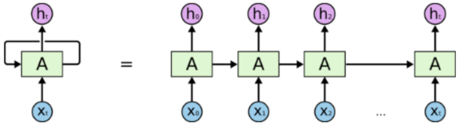
# Conclusion Remarks

- Network Pruning
- Knowledge Distillation
- Parameter Quantization
- Architecture Design
- Dynamic Computation

zhuo.su@oulu.fi

Next session:

# RNN, LSTM and Applications




An unrolled recurrent neural network.