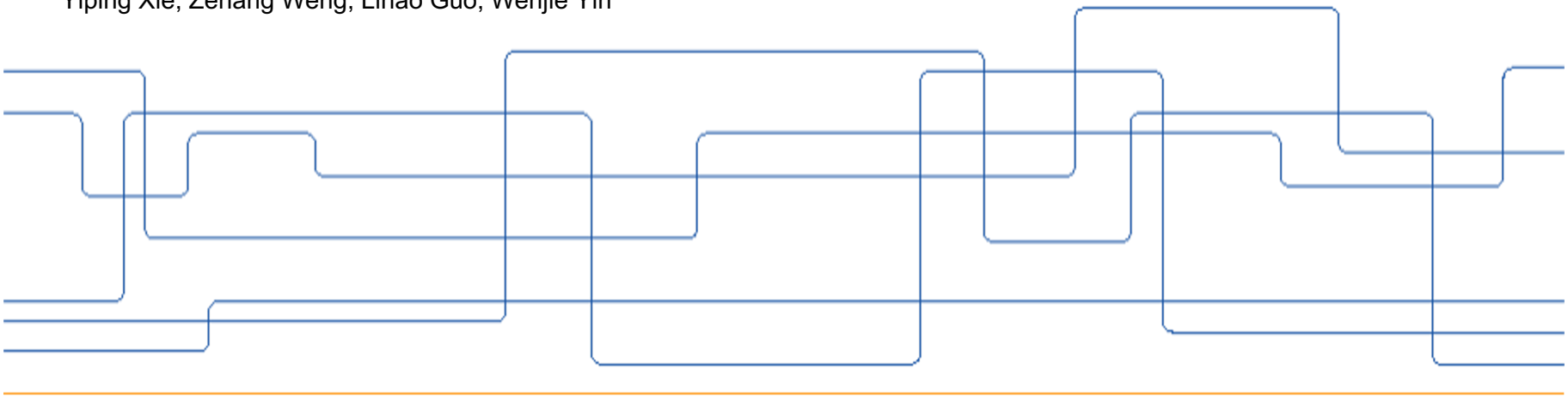
MLoNs Special Topic: Model Compression

Group 1

Yiping Xie, Zehang Weng, Lihao Guo, Wenjie Yin



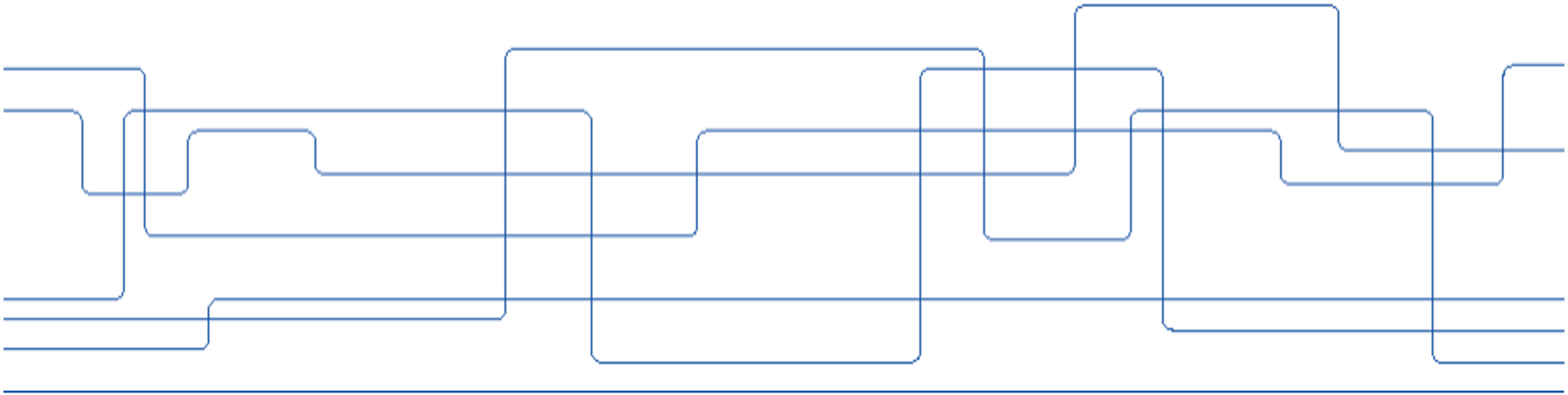
Outline

Part 1: Introduction

Part 2: Methods and Case Studies

Part 3: Q&A

Part 1: Introduction



Introduction

Dilemma of Deep Learning

Mobile Devices:

Not good calculation

Wearable Devices:

Cannot calculate DL

Computing Centre:

Not affordable

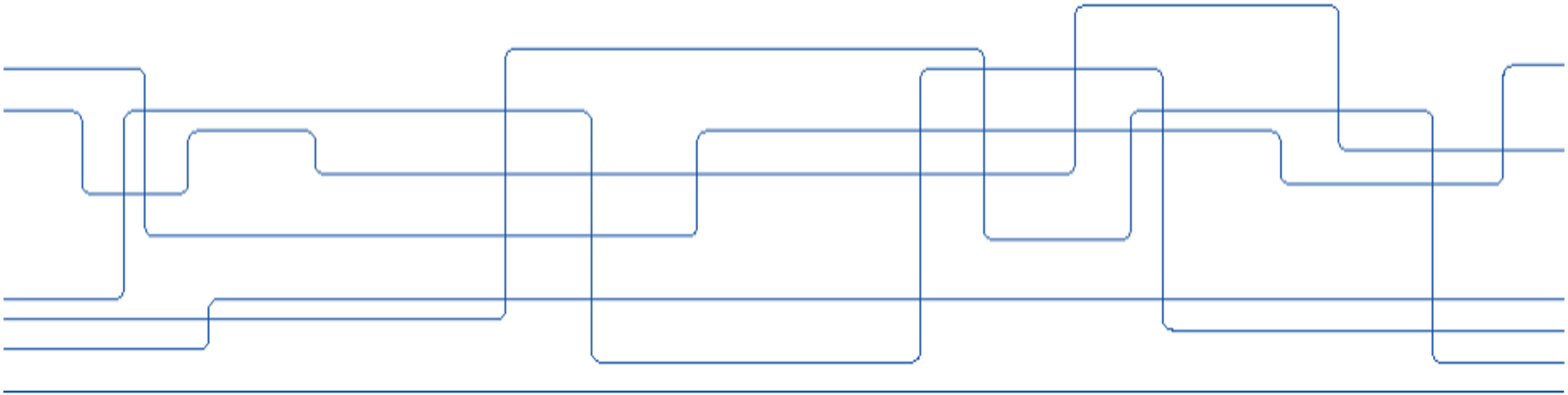
Introduction

Theoretical Basis

- **Necessarily**
  - In many network structures, the number of parameters is too much.
- **Feasibility**
  - There is significant redundancy in many deep neural networks.
  - Using only a small portion (5%) of the weights is sufficient to predict the remaining weights [1].
- **Purpose**
  - Minimize the model complexity
  - Reduce the space required for model storage
  - Accelerate model training and speculation.

[1] Denil, Misha, et al. "Predicting parameters in deep learning." Advances in neural information processing systems. 2013.

Part 2: Methods



Outline

Part 2: Methods and Case Studies

- Method: Quantization
- Method: Pruning
- Case study: Deep compression

- Method: Low rank factorization
- Method: Knowledge distillation
- Method: Compact network design
- Case study: Mobilenet

Methods

Quantization

- Bundling weights together by clustering them or rounding them off
- Generally, the parameters of the neural network model are represented by 32-bit length floating numbers.
- In fact, it does not need to retain such high precision

Methods

Binary Quantization

Based on Sign function

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Based on the probability distribution

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2}))$$

Courbariaux M, Hubara I, Soudry D, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1[J]. arXiv preprint arXiv:1602.02830, 2016.

Methods

Binary Quantization

Based on Sign function

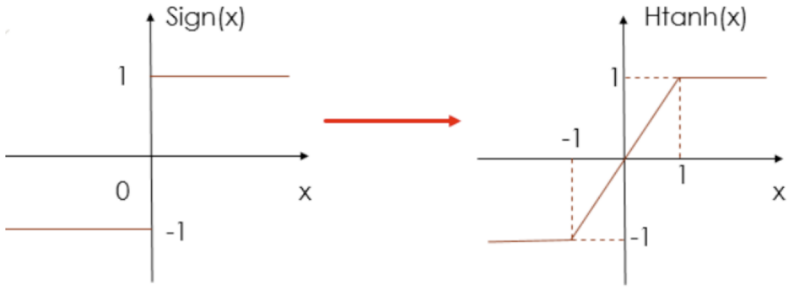
$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Based on the probability distribution

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

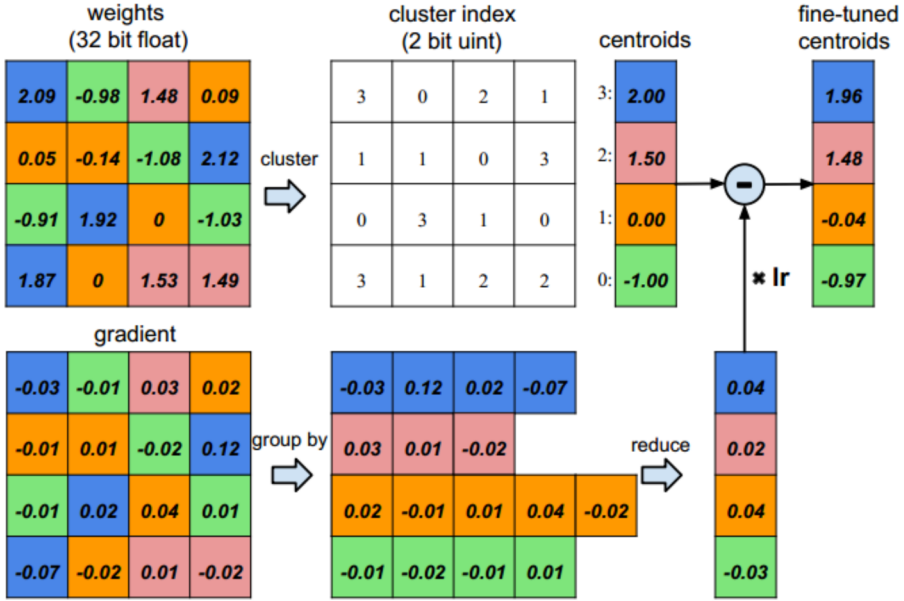
$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2}))$$

$$\text{Htanh}(x) = \text{Clip}(x, -1, 1) = \max(-1, \min(1, x)).$$



Courbariaux M, Hubara I, Soudry D, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1[J]. arXiv preprint arXiv:1602.02830, 2016.

Methods
Quantization



- First cluster the weights or gradients
- Generate index based on clustering results
- Get the results based on the index

Methods

Gradient Quantization for Distributed Deep Learning

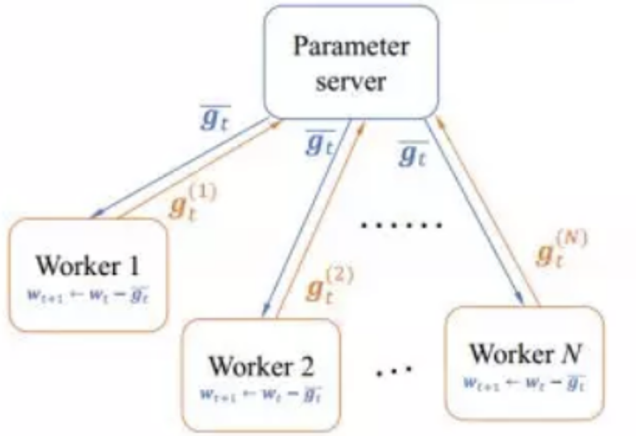


Figure 1: Distributed SGD with data parallelism.

- Distributed SGD is fast, but it is limited by communication cost;
- Gradients communication between servers and workers is expensive;
- Reduce the communication cost by gradient quantization.

Wen, Wei, et al. "Terngrad: Ternary gradients to reduce communication in distributed deep learning." Advances in neural information processing systems. 2017.

Methods

Pruning

- Network are typically over-parameterized (there is redundant weights or neurons， too complex)

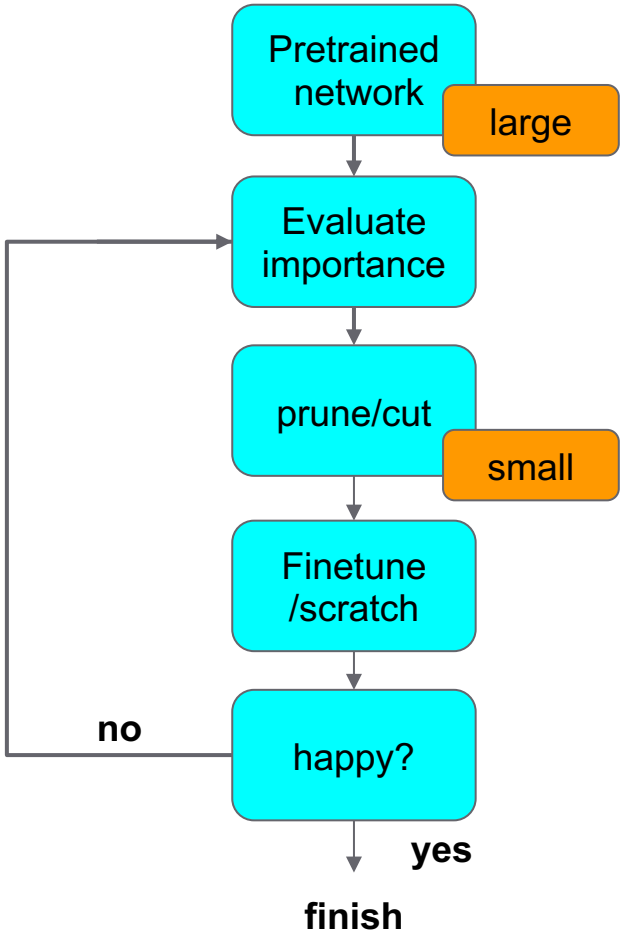
Methods

Pruning

- **Importance:**
  - **Weight importance?**
    L1,L2…
  - **Neuron importance?**
    The number of times it wasn't zero on a given dataset

- **Accuracy drop after pruning:**
  - *Finetune for recovering performance*

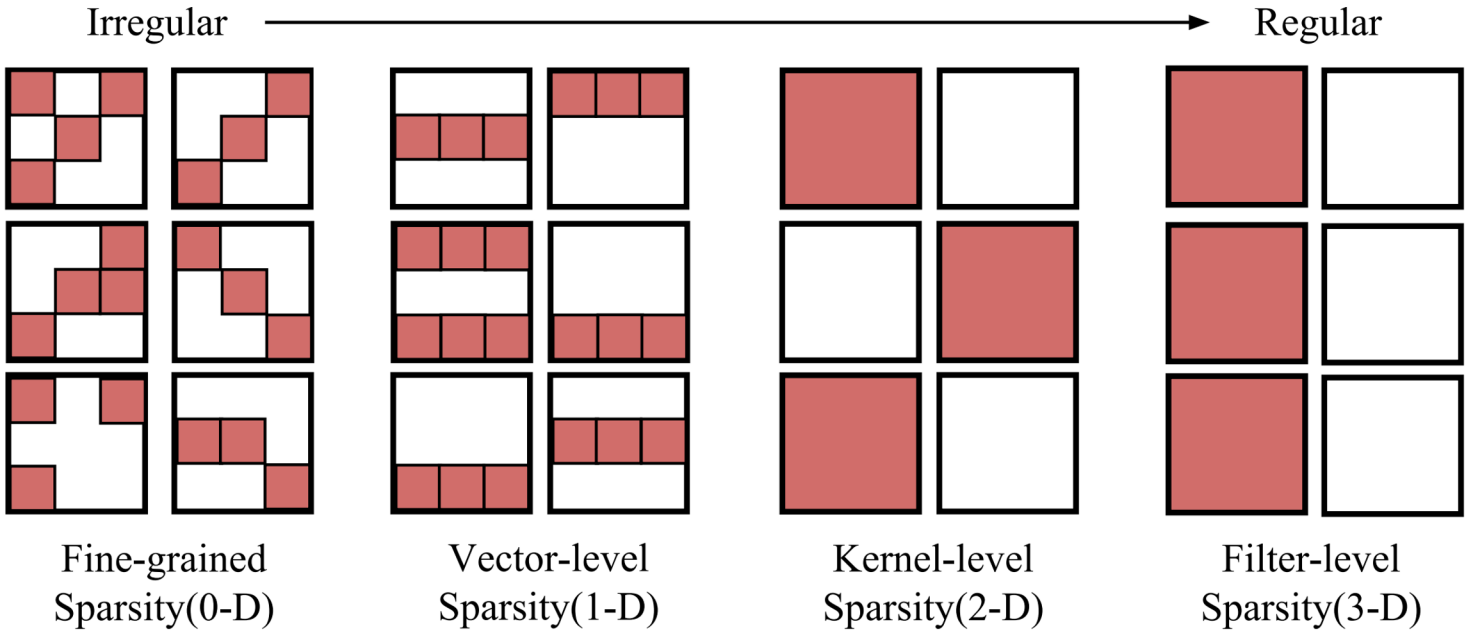
- **Happy:**
  - *Network size*



Methods

Pruning

2. Structural pruning and non structural pruning



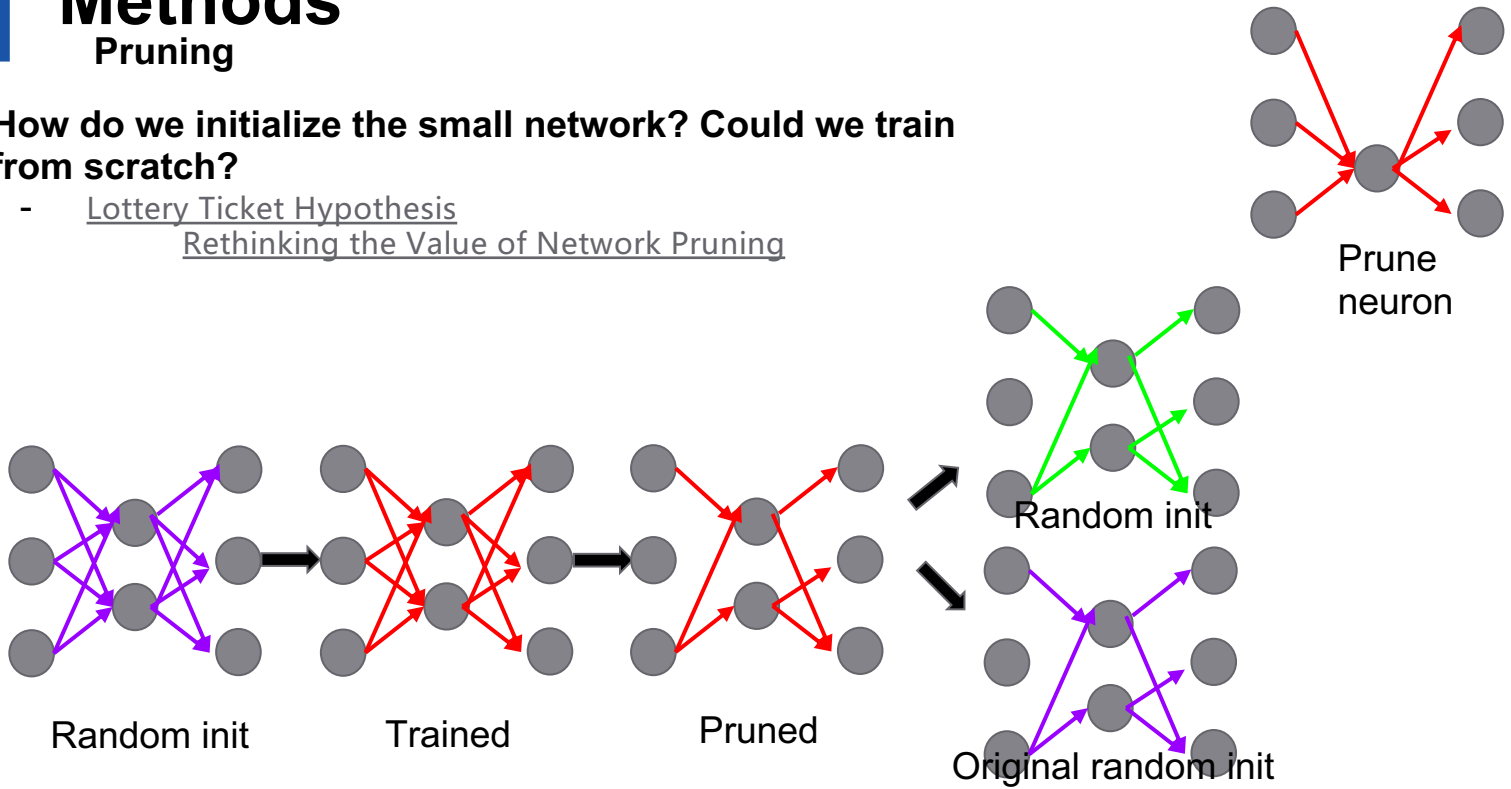
Huizi Mao et al, 2015, Exploring the Regularity of Sparse Structure in Convolutional Neural Networks

Methods

Pruning

How do we initialize the small network? Could we train from scratch?

- Lottery Ticket Hypothesis
  Rethinking the Value of Network Pruning



Eran Malach et al, Lottery Ticket Hypothesis
Zhuang Liu et al. Rethinking the Value of Network Pruning

Case Study
Deep Compressing

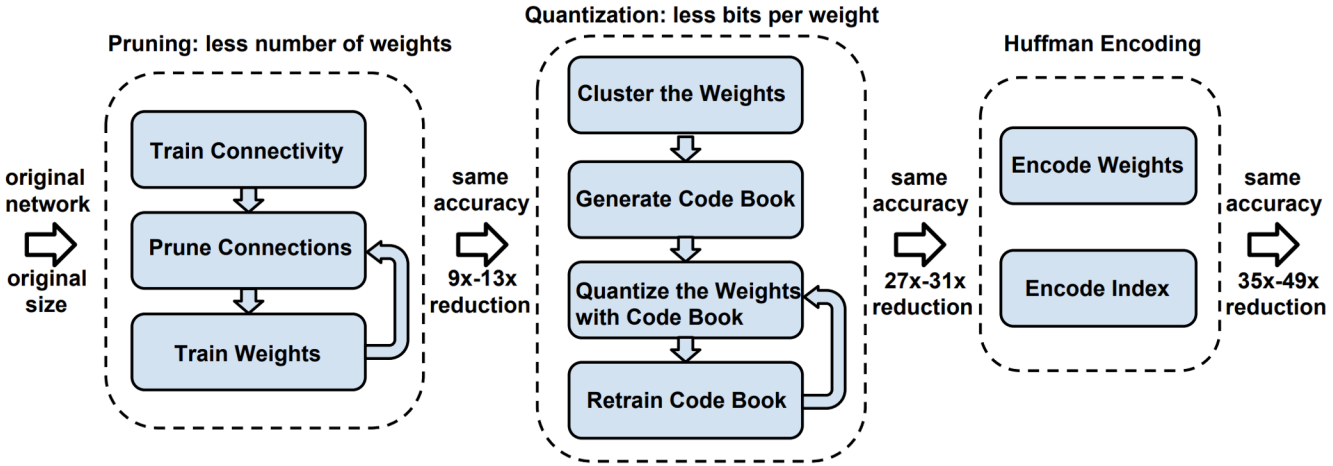


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.

Case Study
Deep Compressing

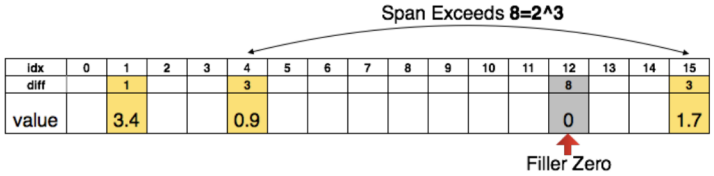


Figure 2: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow.

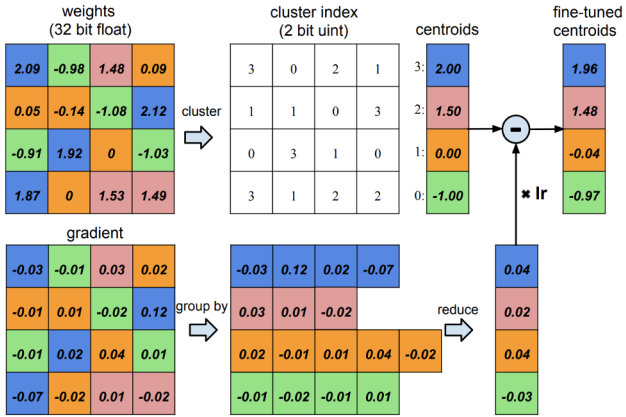


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

Case Study
Deep Compressing

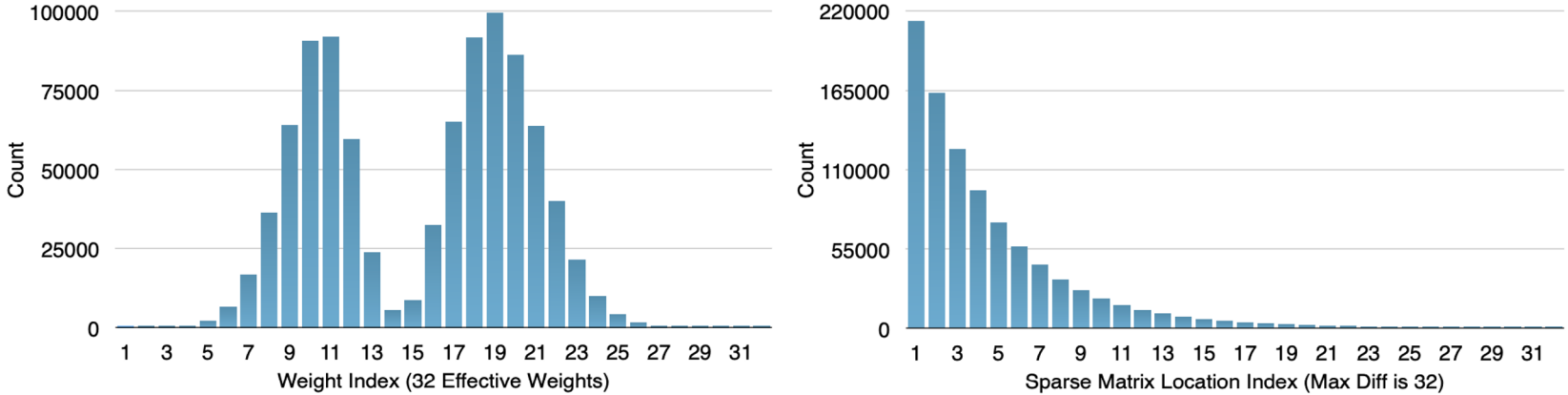


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

Case Study
Deep Compressing

Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

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**×

Case Study
Deep Compressing

Table 4: Compression statistics for AlexNet. P: pruning, Q: quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

Case Study
Deep Compressing

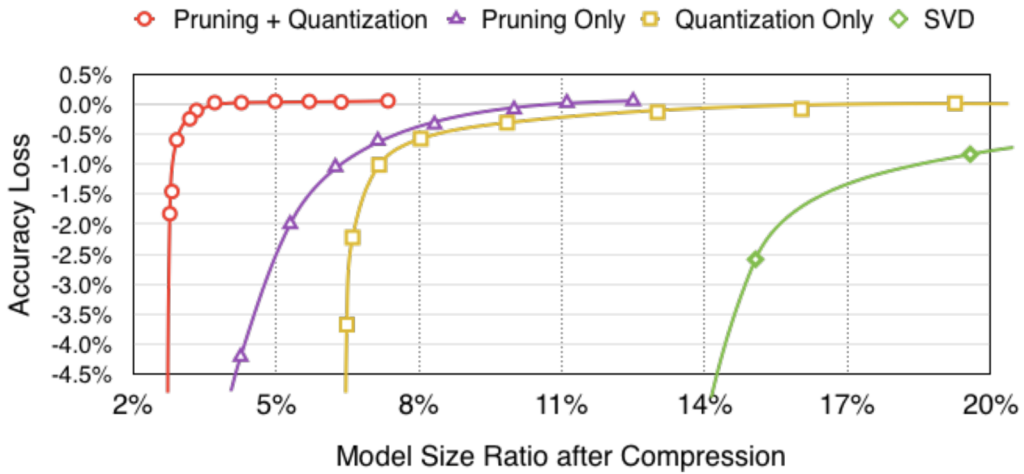


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

Case Study
Deep Compressing

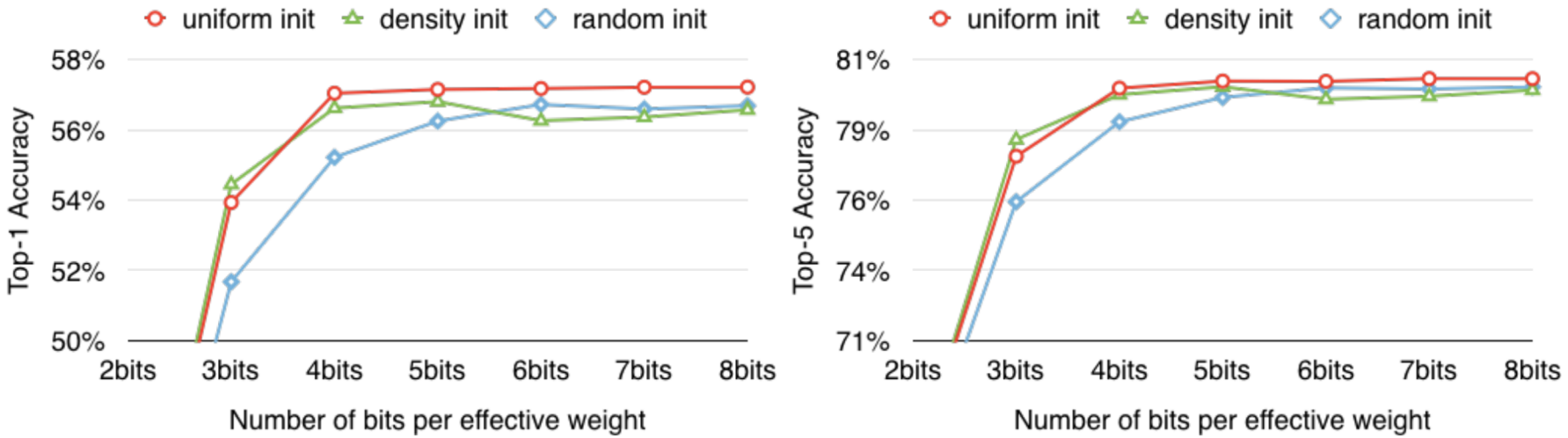
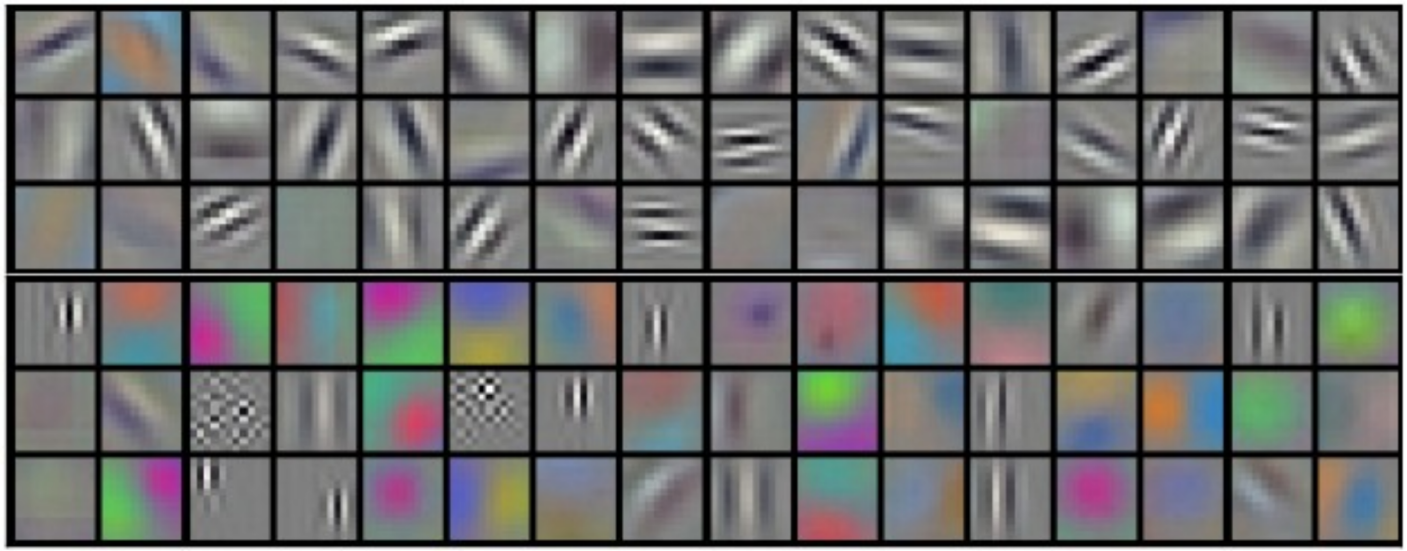


Figure 8: Accuracy of different initialization methods. Left: top-1 accuracy. Right: top-5 accuracy. Linear initialization gives best result.

Methods

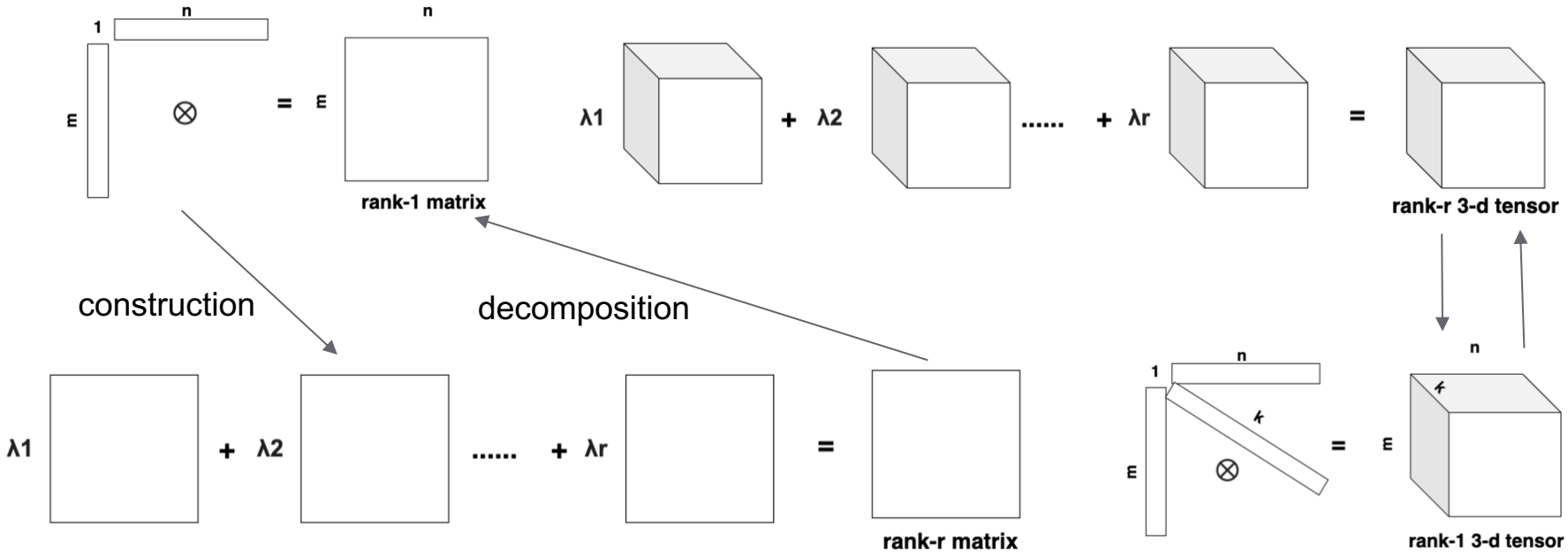
Low Rank - Weight Matrice



https://cs231n.github.io/convolutional-networks/

Methods

Low Rank - Decomposition



Methods

Low Rank - Decomposition

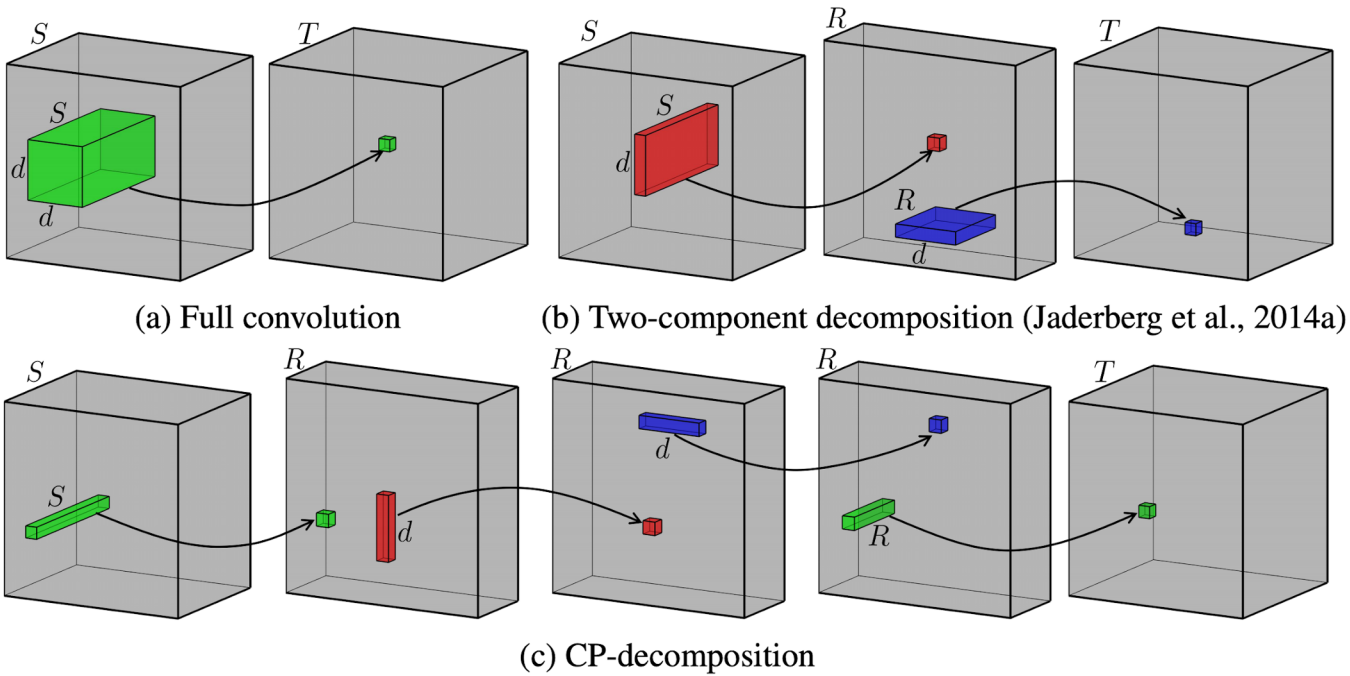
$$F^{n_1 \times \cdots \times n_d} \cong F^{n_1} \otimes \cdots \otimes F^{n_d}$$

$$\mathcal{A} = \sum_{i=1}^{r} \lambda_i \mathbf{a}_i^{(1)} \otimes \mathbf{a}_i^{(2)} \otimes \cdots \otimes \mathbf{a}_i^{(d)}$$

https://en.wikipedia.org/wiki/Tensor_rank_decomposition

Methods

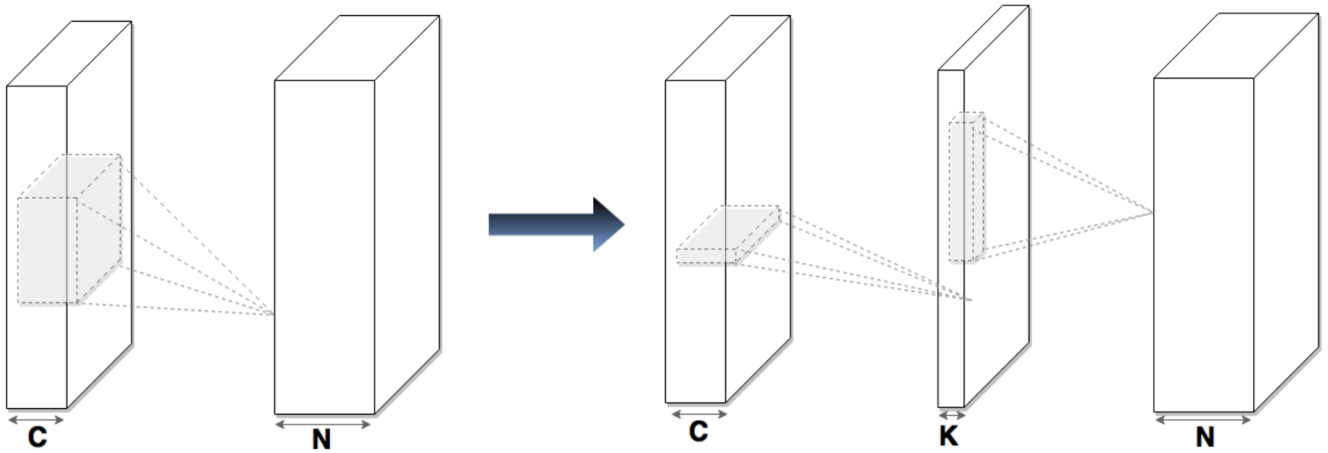
Low Rank - Convolution



Vadim Lebedev et al, 2014

Methods

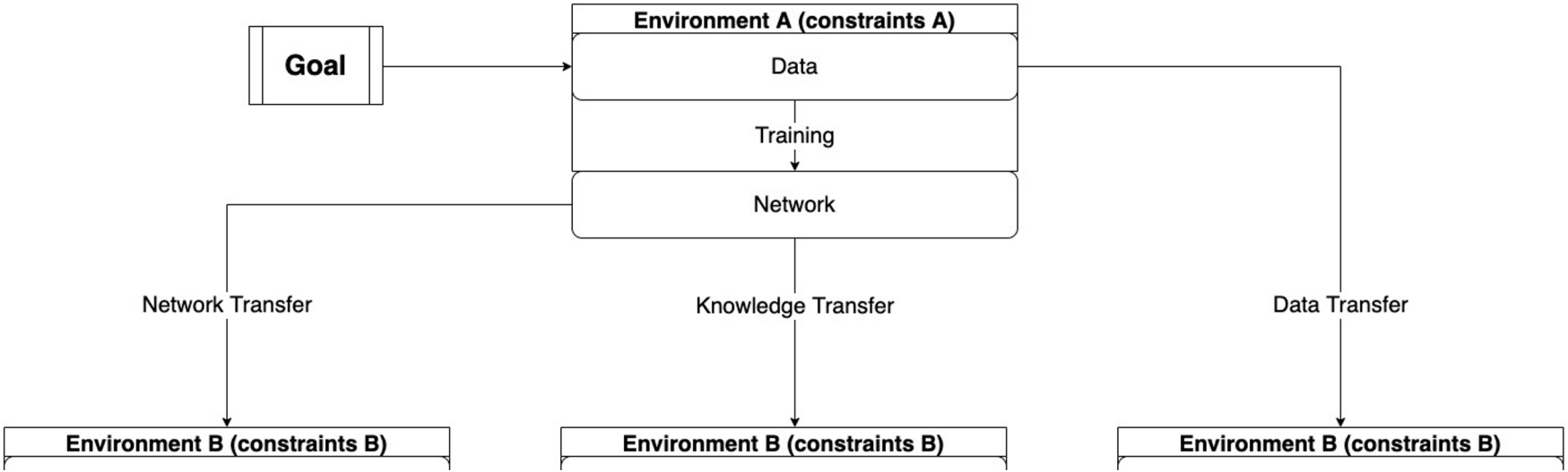
Low Rank - Regularization



Cheng Tai et al, 2014

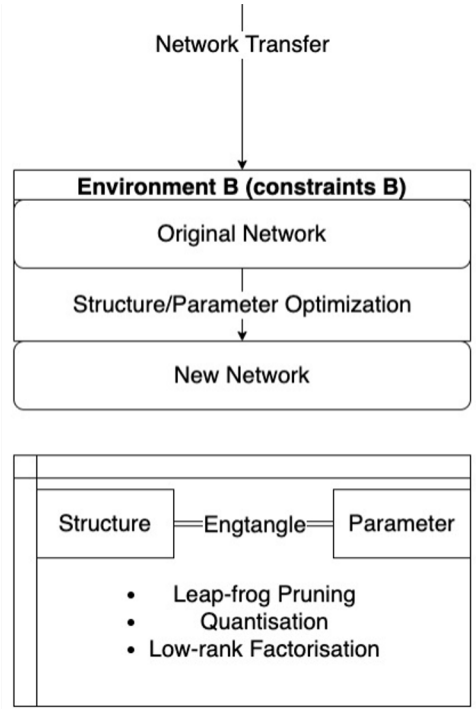
Methods

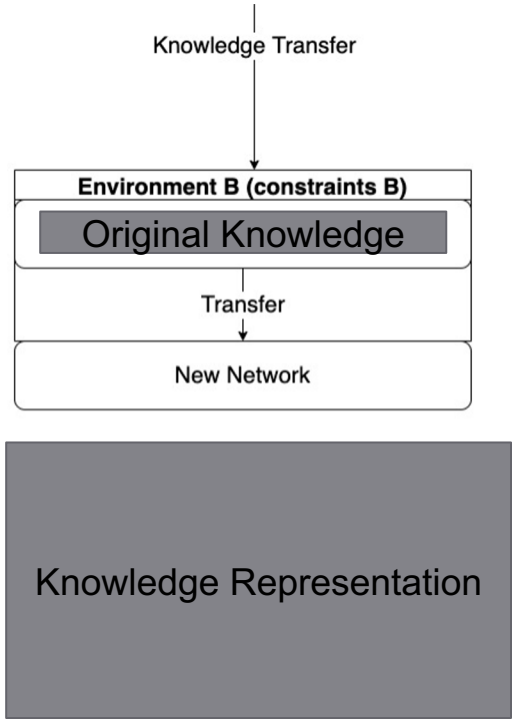
Rethinking The Deployment

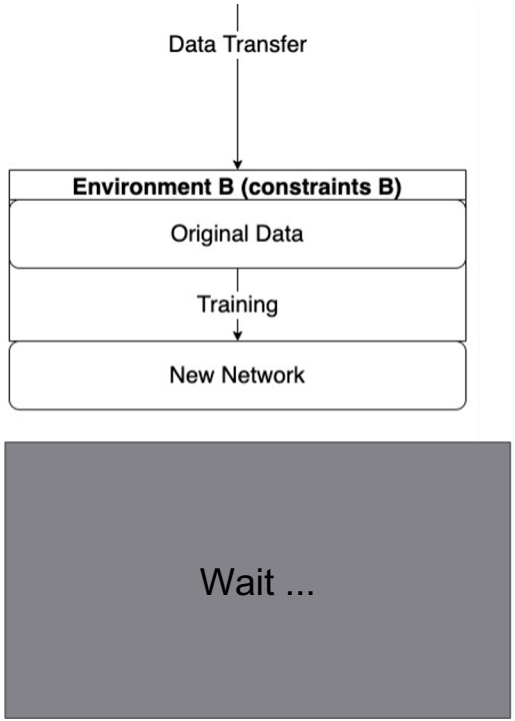


Methods

Transfer Schemes

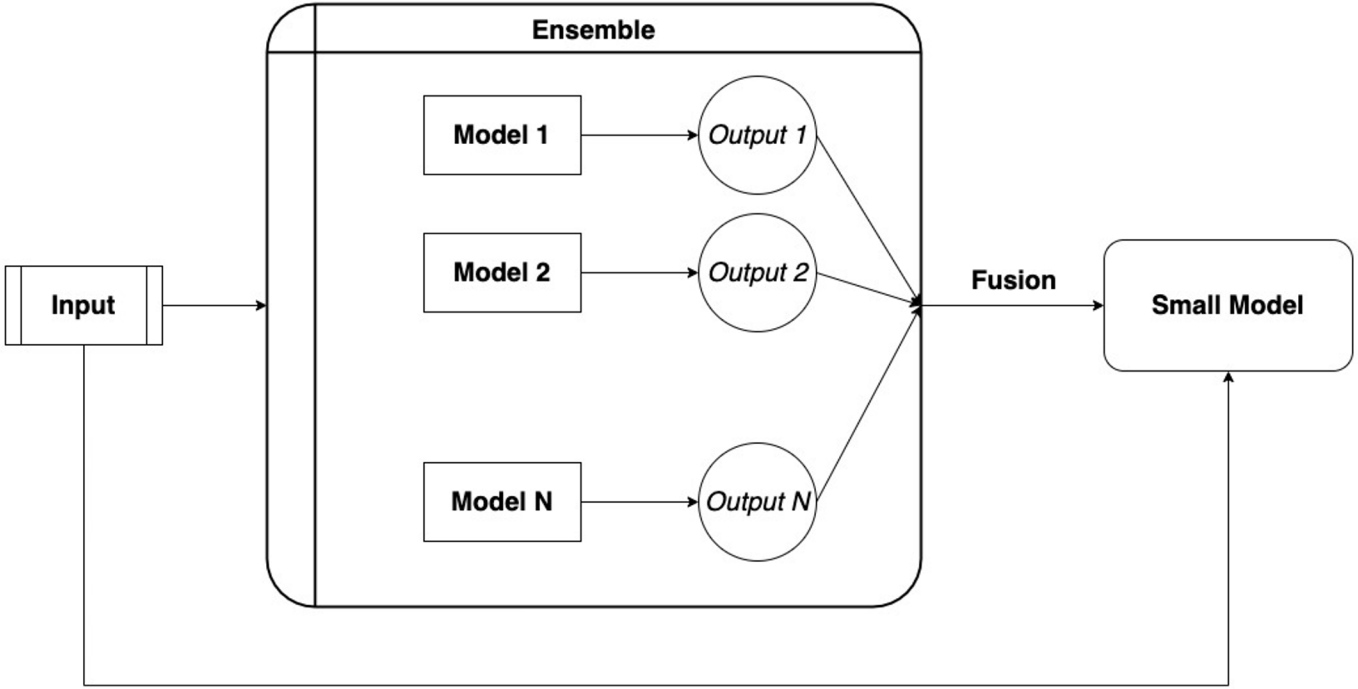






Methods

Knowledge Distillation - Ensemble

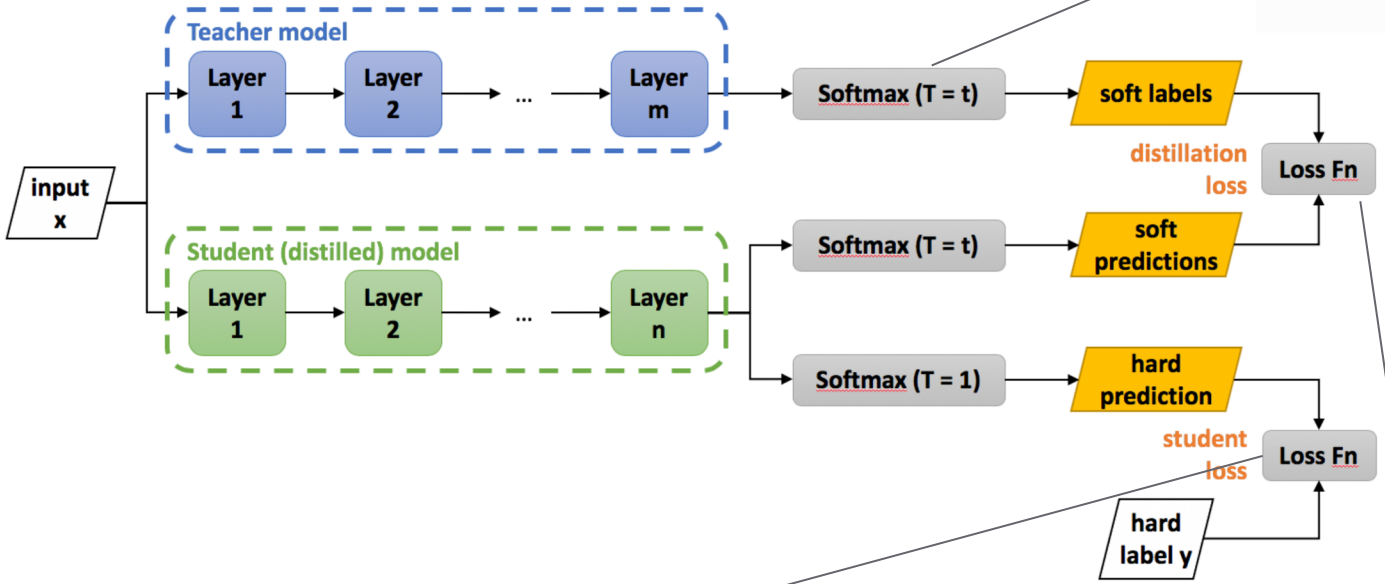


Rich Caruana et al, 2006

Methods

Knowledge Distillation - Generalization on Classification

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



$$\mathcal{L}(x; W) = \alpha * \mathcal{H}(y, \sigma(z_s; T = 1)) + \beta * \mathcal{H}(\sigma(z_t; T = \tau), \sigma(z_s, T = \tau))$$

Hinton et al, 2015

https://github.com/NervanaSystems/distiller/

Methods

Knowledge Distillation - Performance

Table 9: CIFAR10: Teacher and distilled model accuracy, full precision

Model name	Test accuracy	# of parameters	Size (MB)
Teacher model	89.7 %	5.3 millions	21.3 MB
Smaller model 1	84.5 %	1.0 millions	4.00 MB
Distilled model 1	88.8 %	1.0 millions	4.00 MB
Smaller model 2	80.3 %	0.3 millions	1.27 MB
Distilled model 2	84.3 %	0.3 millions	1.27 MB
Smaller model 3	71.6 %	0.1 millions	0.45 MB
Distilled model 3	78.2 %	0.1 millions	0.45 MB

Antonio Polino et al, 2018

Methods

Knowledge Distillation - Other Representation

- **Hidden layer**
  (Ping Luo et al, 2016)
- **Attention Transfer**
  (Sergey Zagoruyko et al, 2016)
- **Reinforcement Learning**
  (Anubhav Ashok et al, 2018)

Methods

Codistillation - Decentralized Learning

Algorithm 1 Codistillation

Input loss function $\phi(\texttt{label}, \texttt{prediction})$
Input distillation loss function $\psi(\texttt{aggregated\_label}, \texttt{prediction})$
Input prediction function $F(\theta, \texttt{input})$
Input learning rate η
for `n_burn_in` steps **do**
  for θ_i in `model_set` **do**
    `y, f = get_train_example()`
    $\theta_i = \theta_i - \eta \nabla_{\theta_i} \{\phi(y, F(\theta_i, f))\}$
  end for
end for
while not converged **do**
  for θ_i in `model_set` **do**
    `y, f = get_train_example()`
    $\theta_i = \theta_i - \eta \nabla_{\theta_i} \{\phi(\texttt{y}, F(\theta_i, f)) + \psi(\{\frac{1}{N-1} \sum_{j \neq i} F(\theta_j, f)\}, F(\theta_i, f))\}$
  end for
end while

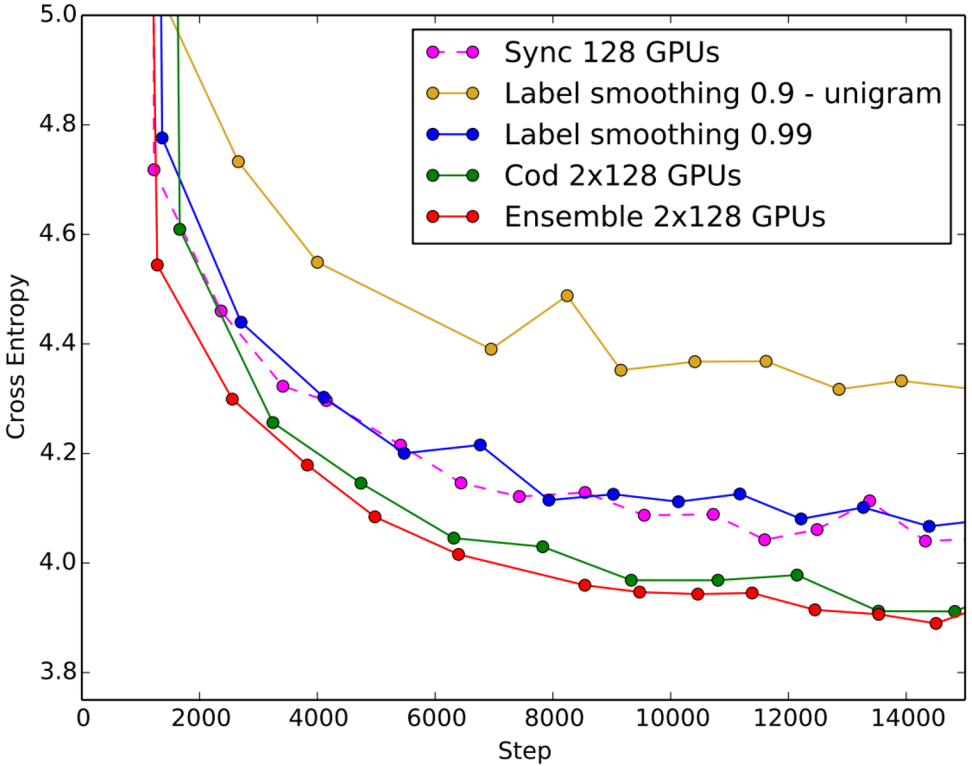
Subset

Communication

Rohan Anil et al, 2018

Methods

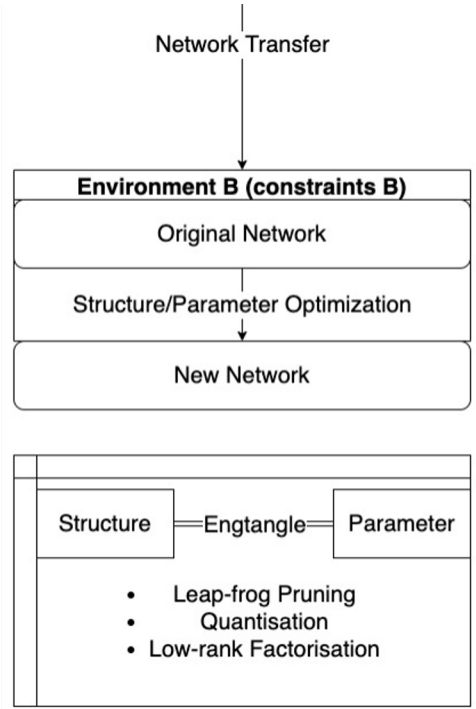
Codistillation - Distributed Learning

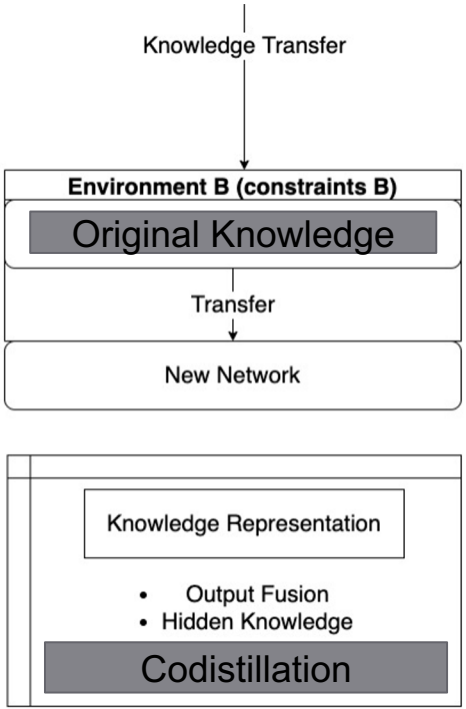


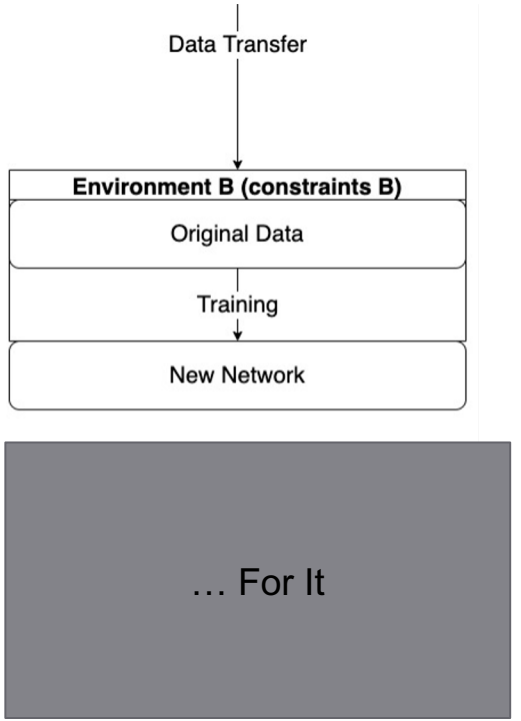
Rohan Anil et al, 2018

Methods

Transfer Schemes



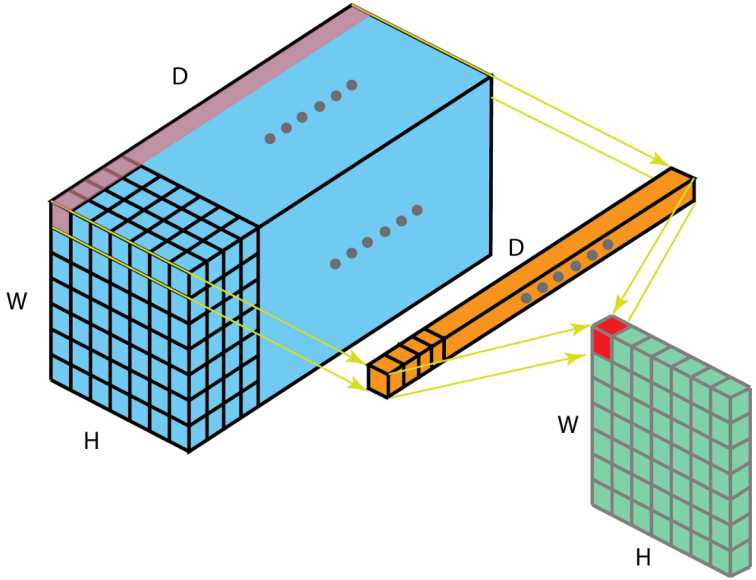




Methods

Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use **1x1 convolution** (a 1x1 filter has 9X fewer parameters than a 3x3 filter)



[1] ttps://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Methods

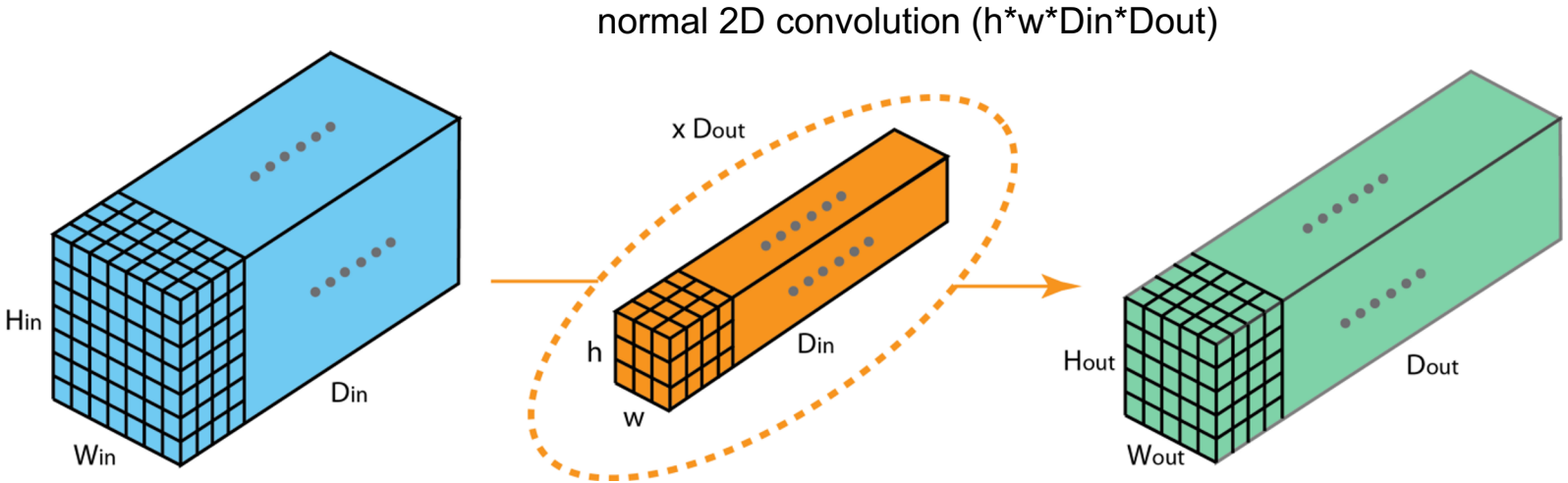
Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use 1x1 convolution (a 1x1 filter has 9X fewer parameters than a 3x3 filter)
  - use **Global Average Pooling** instead of Fully Connected layer

Methods

Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use 1x1 convolution (a 1x1 filter has 9X fewer parameters than a 3x3 filter)
  - use Global Average Pooling instead of Fully Connected layer
  - **group convolution**



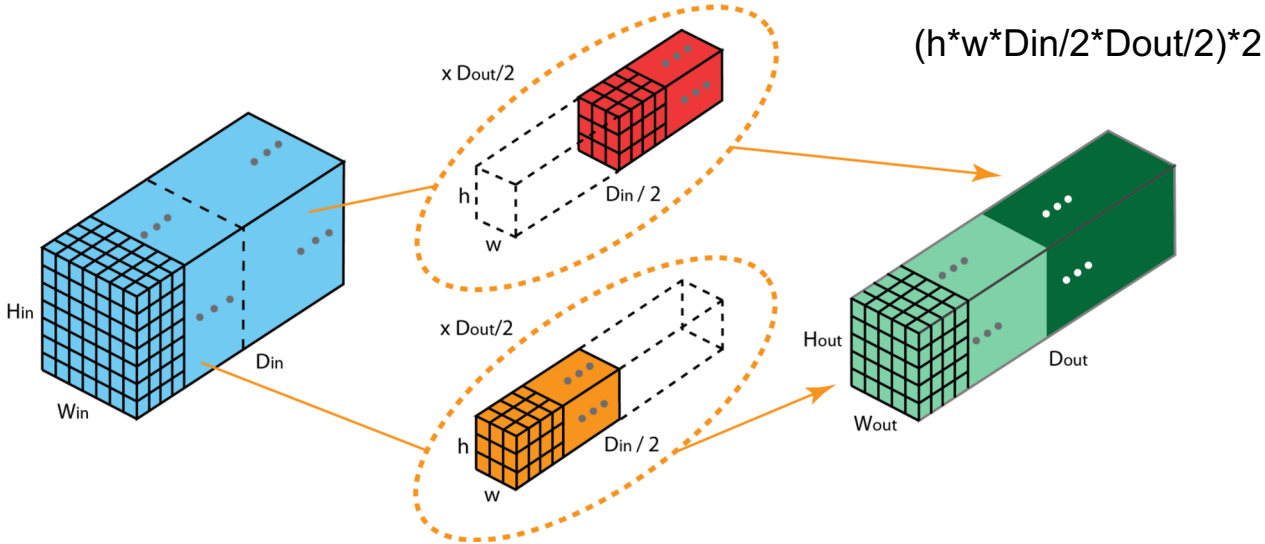
[1] ttps://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Methods

Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use 1x1 convolution (a 1x1 filter has 9X fewer parameters than a 3x3 filter)
  - use Global Average Pooling instead of Fully Connected layer
  - **group convolution**

(h*w*Din/2*Dout/2)*2

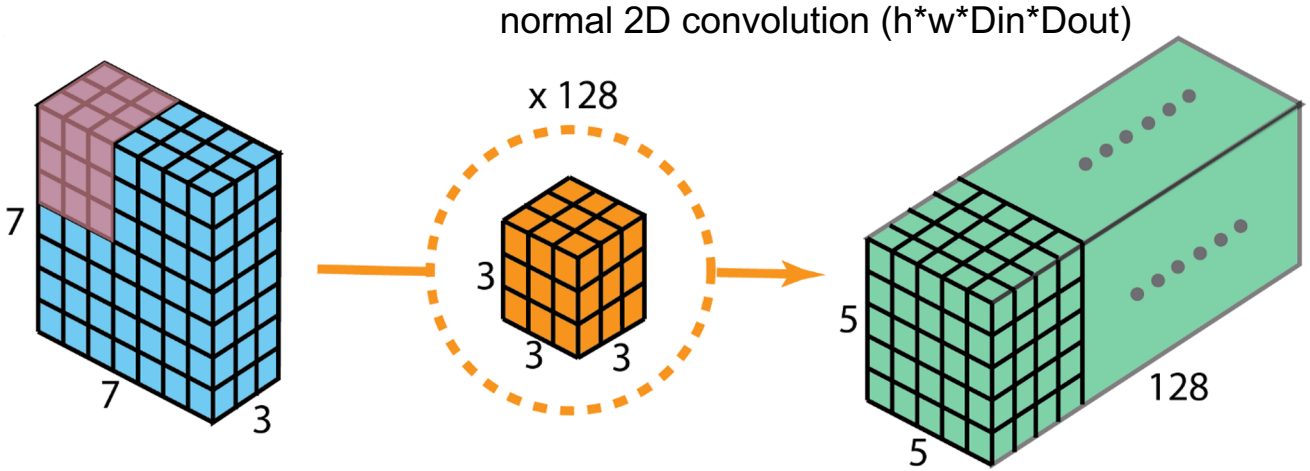


[1] ttps://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Methods

Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use 1x1 convolution (a 1x1 filter has 9X fewer parameters than a 3x3 filter)
  - use Global Average Pooling instead of Fully Connected layer
  - group convolution
  - **depth-wise separable convolution**



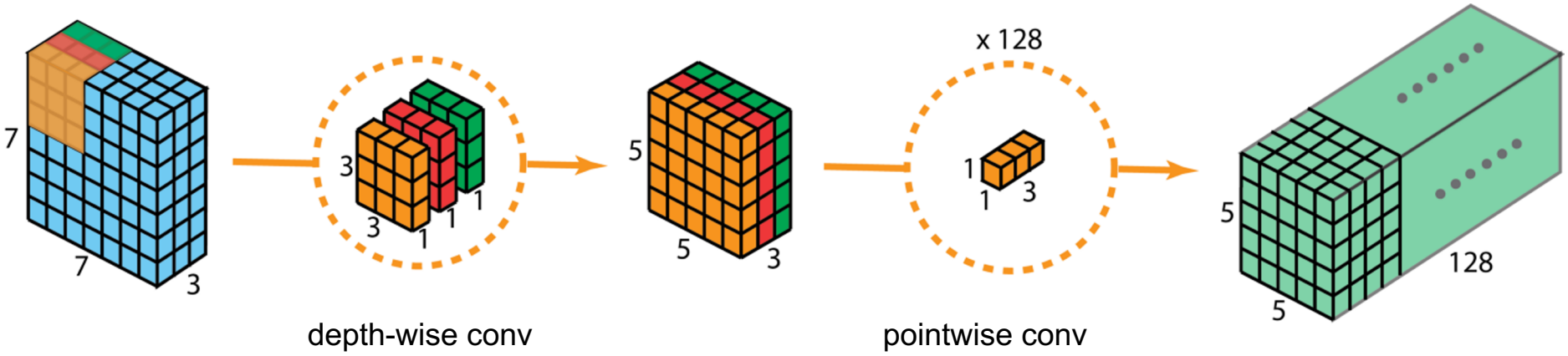
[1] ttps://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Methods

Compact network structure design

- Directly design more efficient but low-cost network architecture
  - use 1x1 convolution (a 1x1 filter has 9X fewer parameters than a 3x3 filter)
  - use Global Average Pooling instead of Fully Connected layer
  - group convolution
  - **depth-wise separable convolution**

(h*w*Din+1*1*Din*Dout)



depth-wise conv pointwise conv

[1] ttps://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Case Study
MobileNet V2

- For mobile and embedded vision applications
- Network based on MobileNet V1
  - Depthwise Separable Convolution

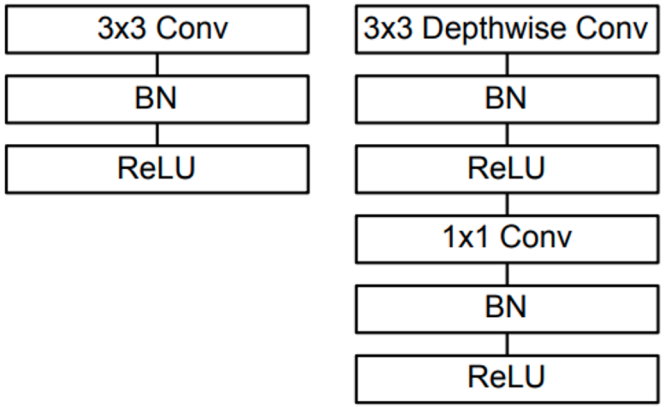


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Case Study

MobileNet V2

- For mobile and embedded vision applications
- Problems with MobileNet V1:
  - Manifold Collapse because of ReLU

Case Study

MobileNet V2

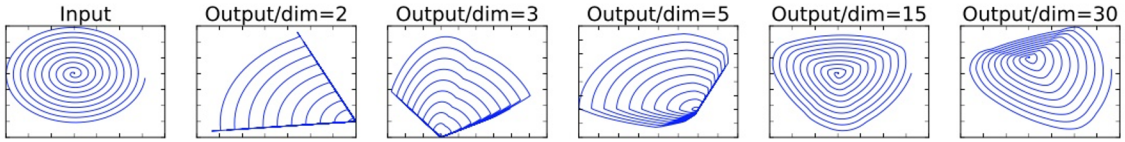
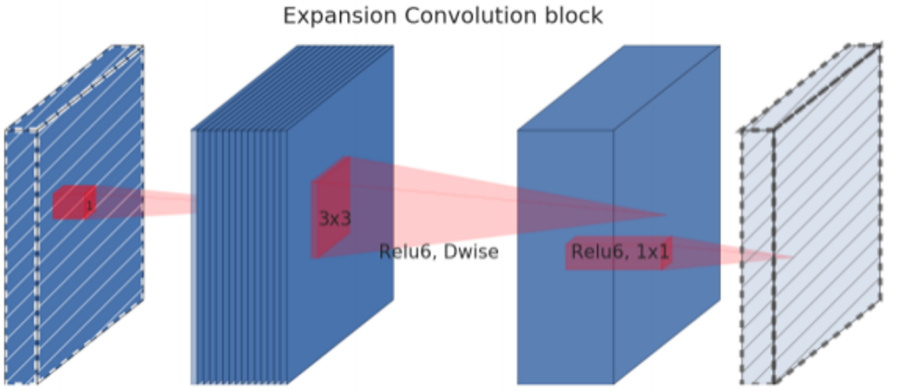


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

Case Study

MobileNet V2



Case Study

MobileNet V2

- For mobile and embedded vision applications
- Problems with MobileNet V1:
  - Manifold Collapse because of ReLU
  - No feature reuse

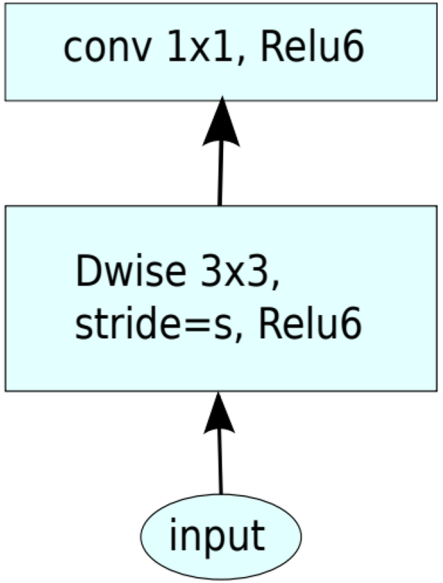
Case Study

MobileNet V2

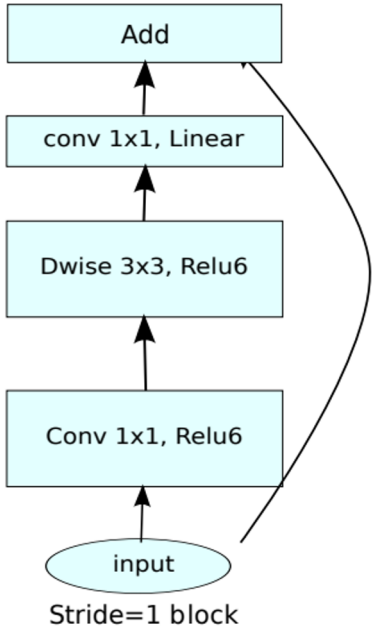
- For mobile and embedded vision applications
- Problems with MobileNet V1:
  - Manifold Collapse because of ReLU
  - No feature reuse
- Solution:
  - The inverted residual with linear bottleneck

Case Study

MobileNet V2



(b) MobileNet[27]



(d) Mobilenet V2

Case Study
MobileNet V2

- SSDLite:
  - Mobile friendly variant of Single Shot Detector (SSD)
  - Replace all the regular convolutions with separable convolutions (depthwise followed by 1 × 1 projection) in SSD prediction layers

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

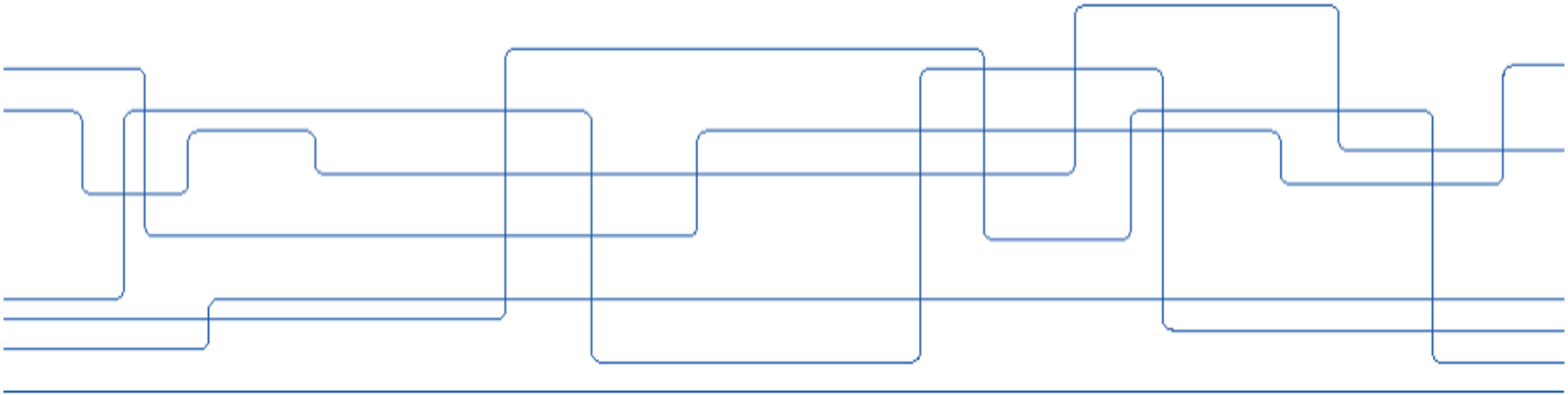
Case Study
MobileNet V2

- Performance

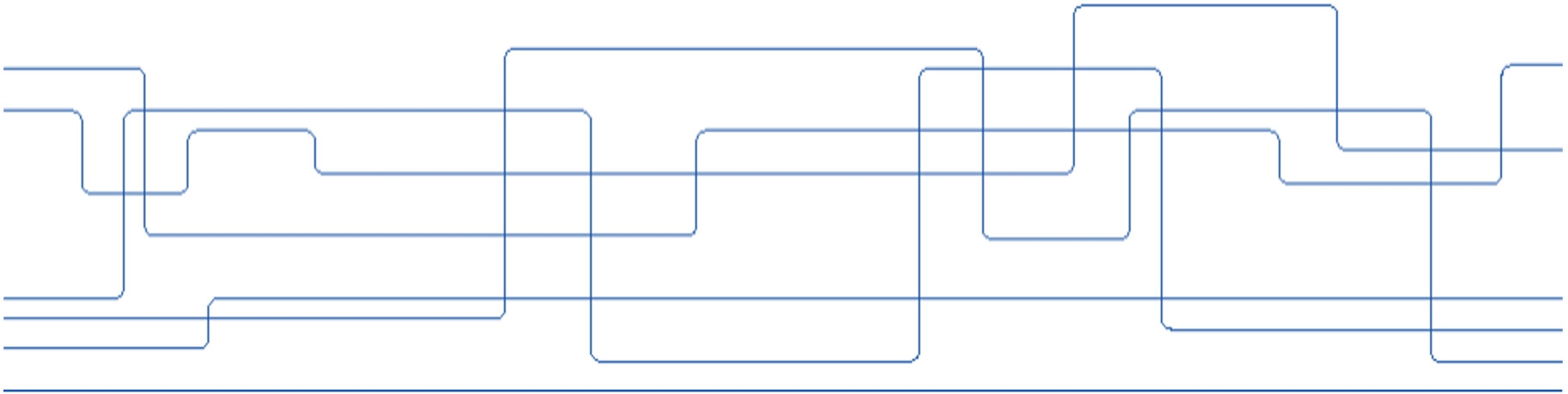
Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Table 6: Performance comparison of MobileNetV2 + SSDLite and other realtime detectors on the COCO dataset object detection task. MobileNetV2 + SSDLite achieves competitive accuracy with significantly fewer parameters and smaller computational complexity. All models are trained on `trainval35k` and evaluated on `test-dev`. SSD/YOLOv2 numbers are from [35]. The running time is reported for the large core of the Google Pixel 1 phone, using an internal version of the TF-Lite engine.

Part 3: Q & A



Thanks!



MLoNs Special Topic: Model Compression

Group 1

Yiping Xie, Zehang Weng, Lihao Guo, Wenjie Yin