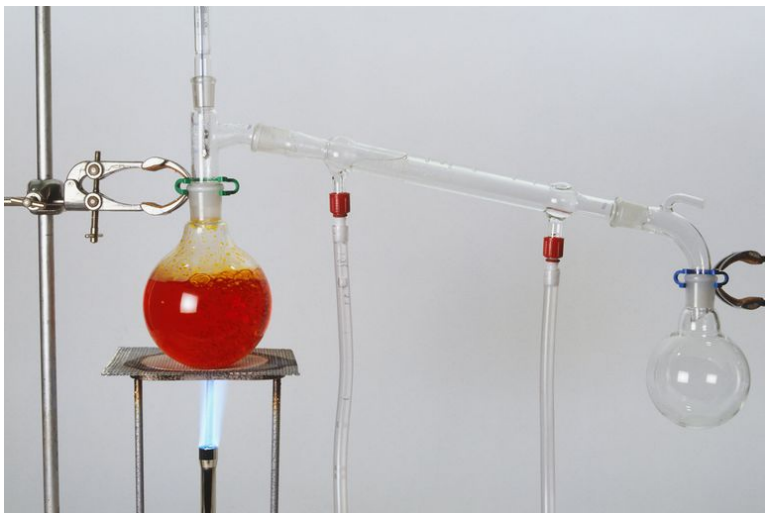
Knowledge Distillation

Knowledge Distillation

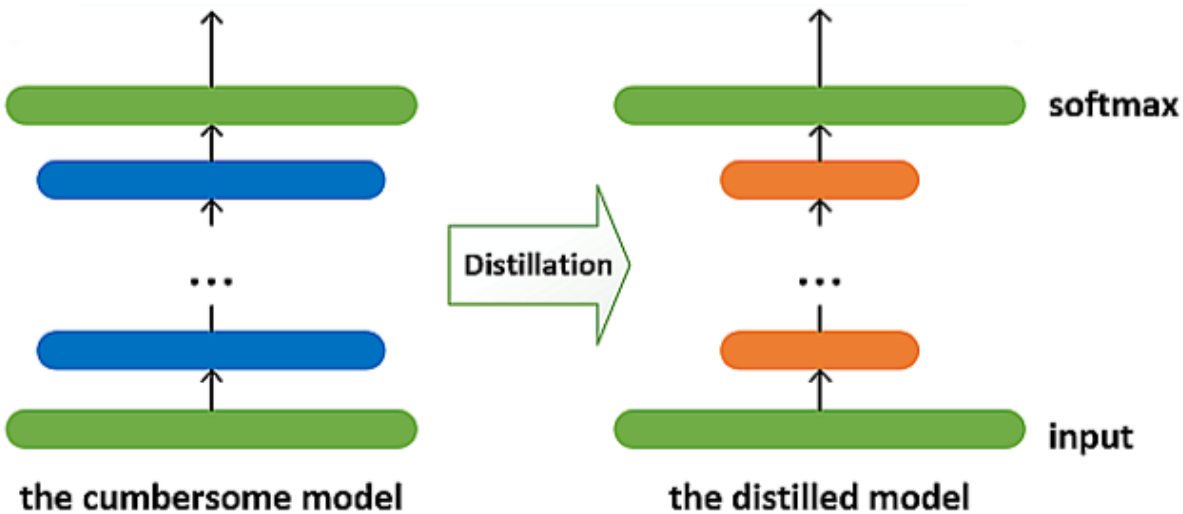
(Hinton et al, 2015)

Distilling the Knowledge in a Neural Network

transfer knowledge from a cumbersome (teacher) model to a small distilled model that is more suitable for deployment



Knowledge Distillation...



Cumbersome model

- an ensemble of models
- a very large model $\rightarrow$ obviously, can be used for network compression

Idea

naive approach

match the predictions of the distilled model with those of the cumbersome model

knowledge distillation: idea

do not focus only on the correct class label

- the trained model assigns probabilities to all incorrect answers
- relative probabilities of incorrect answers tell us a lot about how the cumbersome model tends to generalize

Example

$$\text{prob(man)} \gg \text{prob(monkey)} \gg \text{prob(car)}$$

- the model thinks monkey is closer to man than car

Matching the Probability Distributions over Classes

- softmax layer in multi-class classification

$$\text{prob of class } i : \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

  - z_i: input to softmax
- minimize cross entropy: $-\sum_{k=1}^{|V|} p(y=k|x) \log q(y=k|x)$
  - p: distribution for cumbersome (teacher) model
  - q: distribution for distilled model

problem

- the cumbersome model almost always produces the correct answer with very high confidence
  $\rightarrow$ incorrect answers have very small probabilities
  $\rightarrow$ very little influence on the cross-entropy cost function during transfer

Modifying the Softmax

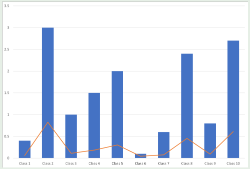
solution: add temperature

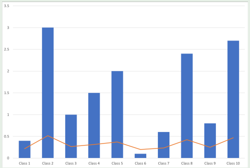
soft targets : class probability $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$

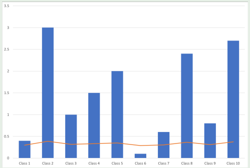
- T: temperature (original: $T = 1$)

Example

$T = 1, 3, 10$





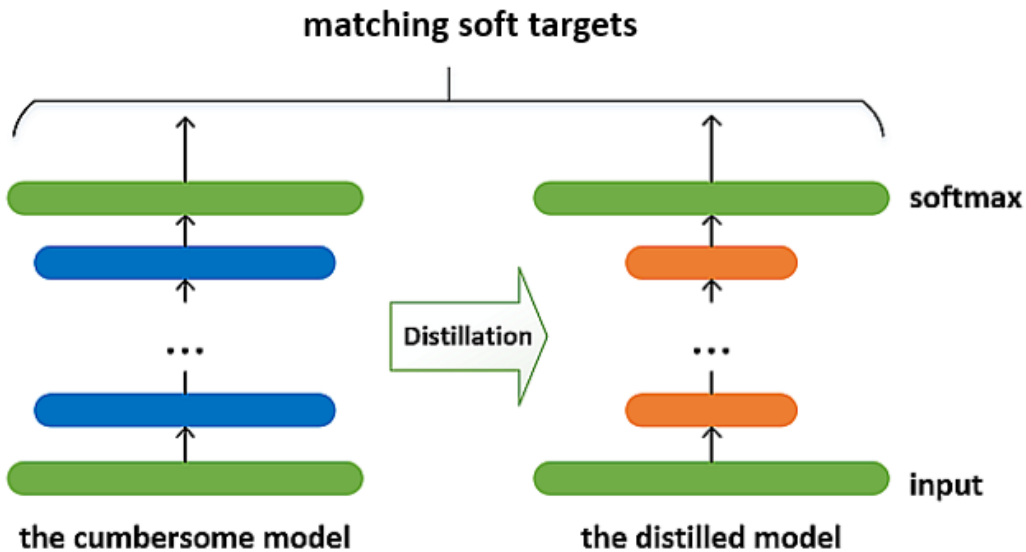


- the higher T, the softer the distribution

Knowledge Distillation with No Label Information

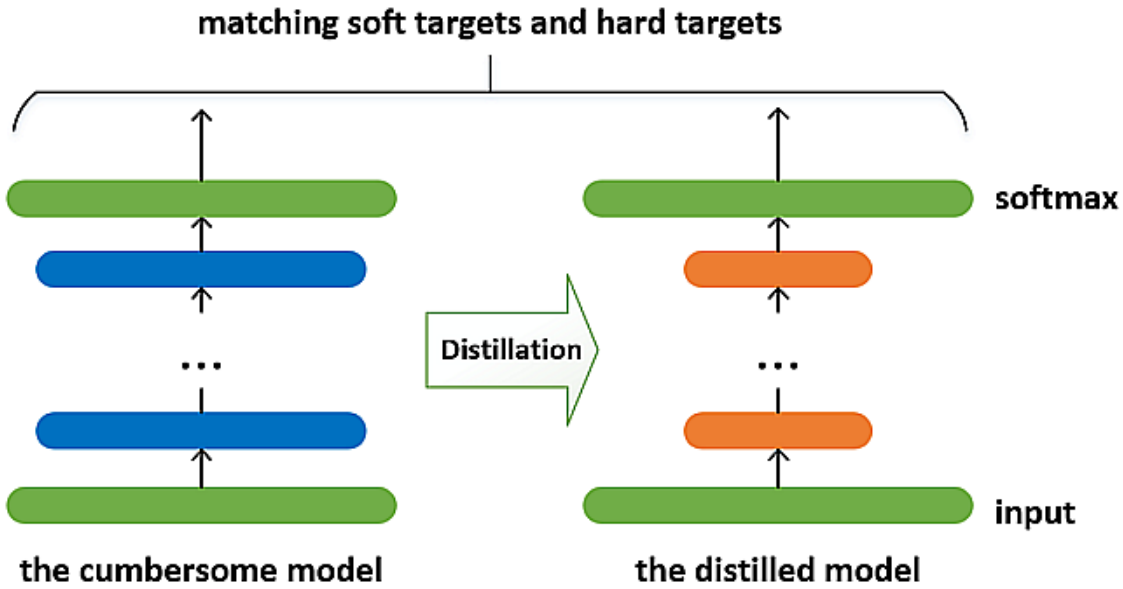
no label information

match soft target distributions on the training set or a transfer set



- match the soft target distribution produced by cumbersome model and distilled model (with a high temperature in softmax)
- after training, the distilled model uses a temperature of 1

Knowledge Distillation with Label Information



(average of two objective functions) $L = (1-\alpha)\ell_1 + \alpha\ell_2$

1. cross entropy with soft targets from cumbersome model

$$\ell_1(\theta_{\text{distilled}}) = -\sum_{k=1}^{|V|} p(y=k|x;\theta_{\text{teacher}}) \log q(y=k|x;\theta_{\text{distilled}})$$

   - use a high temperature

2. cross entropy with correct labels (hard targets)

$$\ell_2(\theta_{\text{distilled}}) = -\sum_{k=1}^{|V|} 1(y=k) \log q(y=k|x;\theta_{\text{distilled}})$$

   - distilled model use a temperature of 1

Distillation + Quantization

(Polino et al., 2018)

Model compression via distillation and quantization

quantized distillation

- a larger teacher network is distilled to a quantized student network

simply replace the original loss with distillation loss

Quantized Distillation

quantizing deep network

at iteration t

1. quantization: turn full-precision weight $\mathbf{w}^t$ to $\hat{\mathbf{w}}^t$
2. forward propagation using $\hat{\mathbf{w}}^t$ and compute the classification loss
3. backpropagate the gradient $\nabla \ell(\hat{\mathbf{w}}^t)$
4. update (full-precision) weight as $\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \nabla \ell(\hat{\mathbf{w}}^t)$

Quantized distillation

at iteration t

1. quantization: turn full-precision weight $\mathbf{w}^t$ to $\hat{\mathbf{w}}^t$
2. forward propagation using $\hat{\mathbf{w}}^t$ and compute the distillation loss
3. backpropagate the gradient $\nabla \ell(\hat{\mathbf{w}}^t)$
4. update (full-precision) weight as $\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \nabla \ell(\hat{\mathbf{w}}^t)$

Experimental Results

Table 6: WMT13 dataset BLEU score and perplexity (ppl). Teacher model: 84.8M param, 340 MB, 5.8 ppl, 34.7 BLEU. Details about model size are reported in Table 26.

		4 bits
Student Model	PM Quant. (No bucket)	21.38 BLEU - 12.61 ppl
81.6M param - 326 MB	PM Quant. (with bucket)	27.73 BLEU - 7.4 ppl
30.22 - 30.21 BLEU	Quantized Distill.	35.32 BLEU - 6.48 ppl

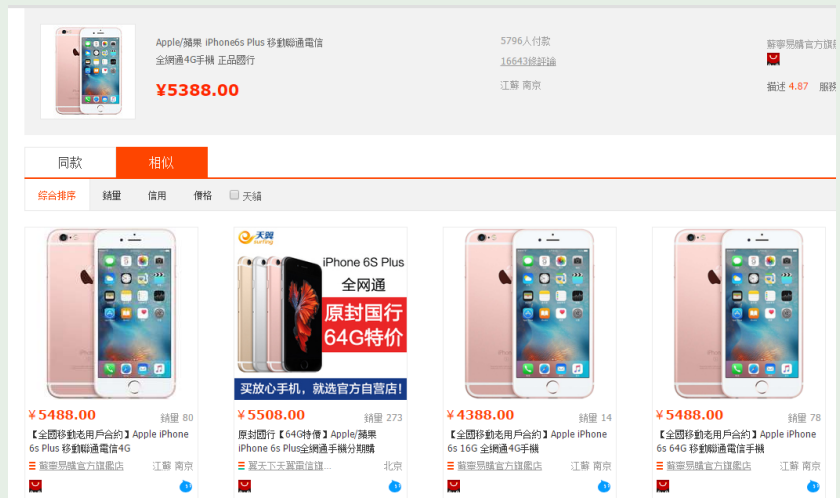
- quantized distillation with 4bits of precision has higher BLEU score than the teacher, and similar perplexity.

Low-Rank Approximation

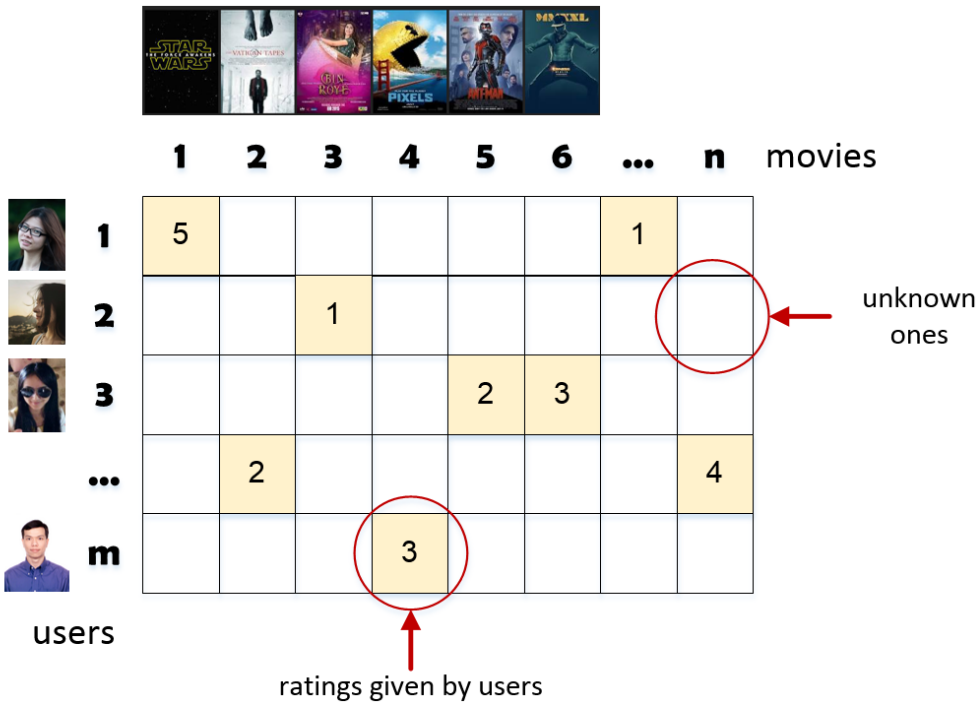
Popular Application: Recommender Systems

predict users' habit or preference based on their historical data

Example (News recommendation, product recommendation)



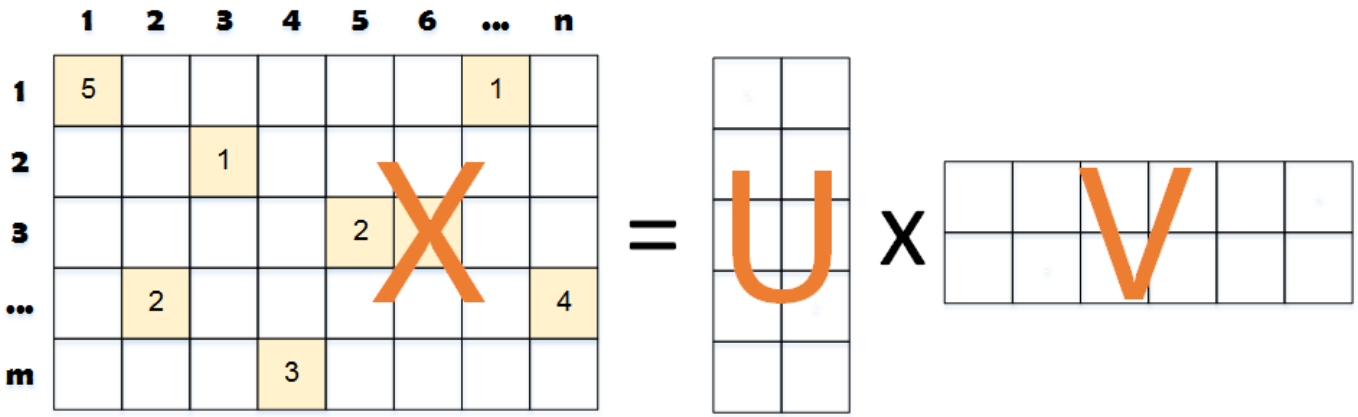
Recommender Systems as Matrix Learning



- data represented as a matrix
- rating prediction $\leftrightarrow$ filling in the missing matrix entries (matrix completion)

Low-Rank Assumption

- the matrix is assumed to be low-rank



- U: contains latent feature vectors for users
- V: contains latent feature vectors for items

Low-Rank Approximation for Deep Networks

fully-connected layers

- $\mathbf{W} \in \mathbb{R}^{m \times n}$
- represent $\mathbf{W}$ as a low-rank product of two smaller matrices
- $\mathbf{W} \simeq \mathbf{UV}$ ($\mathbf{U} \in \mathbb{R}^{m \times r}, \mathbf{V} \in \mathbb{R}^{r \times n}$, rank r)

complexity

- space: from mn to $(m+n)r$
- time for computing $\mathbf{Wx}$: from mn to $(m+n)r$

control the rank $\rightarrow$ control the size of the parameterization

How to Learn U, V?

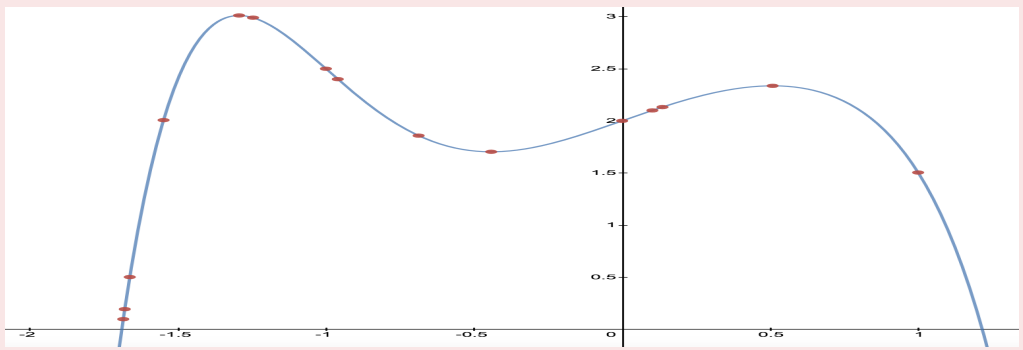
train U, V directly

- may lead to performance degradation

(Denil et al., 2014)

Predicting Parameters in Deep Learning

assumption: weights are smooth over input



- given only a few weight values, it is possible to accurately predict the remaining weight values
- $\rightarrow$ no training for those values

Example

- w_α: observed values of the weight vector w
- use kernel ridge regression to predict the parameter vector over the entire weight vector

$$w = \underbrace{k_\alpha^T (K_\alpha + \lambda I)^{-1}}_{U} \underbrace{w_\alpha}_{V}$$

empirically

- can train networks with vastly fewer parameters (e.g. $\approx 5\%$ subset) while achieving the same performance as full network

neural networks are heavily over-parametrized

CNN

- input: $D_F \times D_F \times M$
  - M: number of input feature maps
- output: $D_F \times D_F \times N$
  - N: number of output feature maps
- convolution kernel: $D_K \times D_K \times M \times N$

cost of computing the convolution

- $D_K^2 M N D_F^2$

the most expensive operations in CNN are the convolutions (esp. in the first few layers)

Low-Rank Approximation: Convolutional Layers

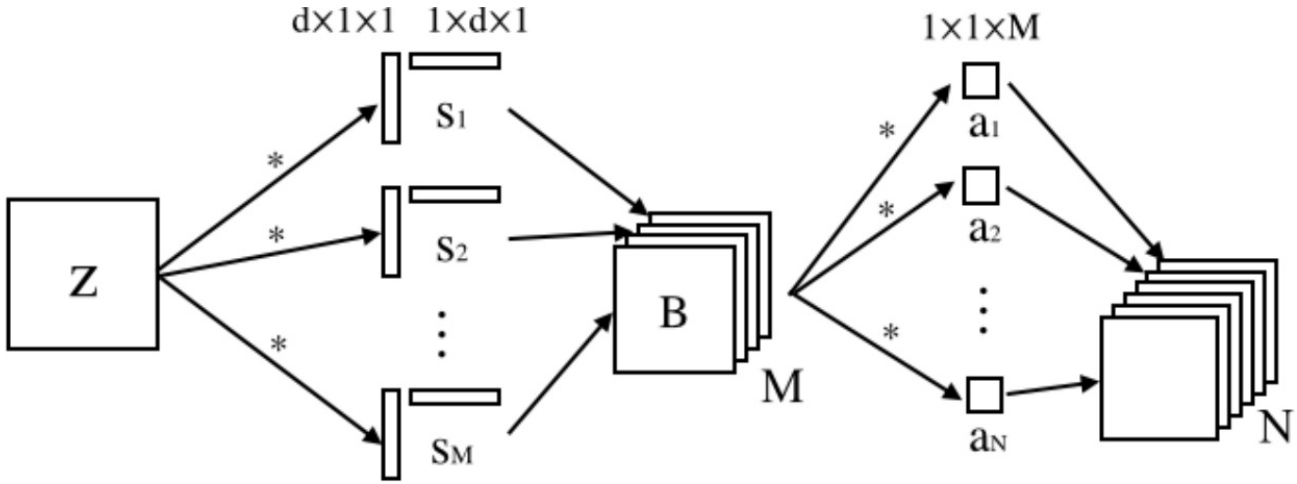
(Jaderberg et al., 2014)

Speeding up Convolutional Neural Networks with Low Rank Expansions

method 1: operate on each input channel separately

- approximate each filter W_n^c as linear combination of M rank-1 filters s_m^c's

$$W_n^c \simeq \sum_{m=1}^{M} a_n^{cm} s_m^c$$



(from (Jaderberg et al., 2014)). Here, $C = 1$

Method 1 (Jaderberg et al., 2014)

$$W_n^c \simeq \sum_{m=1}^{M} a_n^{cm} s_m^c$$

$$W_n * z = \sum_{c=1}^{C} W_n^c * z^c \simeq \sum_{c=1}^{C} \sum_{m=1}^{M} a_n^{cm} (s_m^c * z_c)$$

- cost $O(MC(d^2 + N)H'W')$
- compare with direct computation: $O(NCd^2H'W')$
  - efficient if $M \ll d^2$ (i.e., number of basis filters is less than filter area)

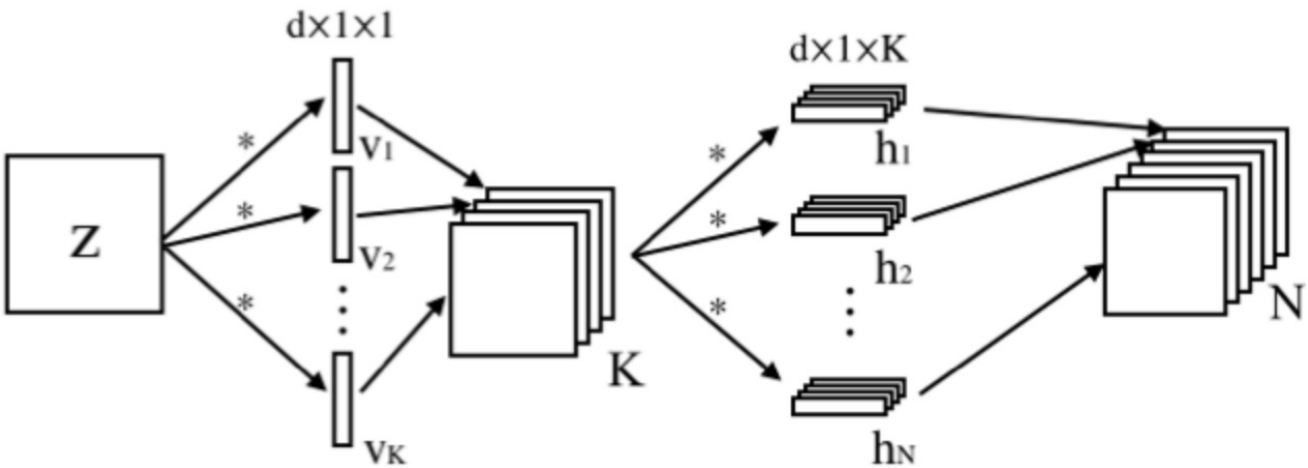
Method 2 (Jaderberg et al., 2014)

method 1: operate on each input channel separately

method 2: operate on multiple channels simultaneously

- consider 3D filters throughout
- factorize each convolutional layer as two convolutional layers

Method 2 (Jaderberg et al., 2014)...



(from (Jaderberg et al., 2014))

- first convolutional layer has K $d \times 1 \times C$ filters
- second convolutional layer has N $1 \times d \times K$ filters
- cost: $O(K(N + C)dH'W')$
- compared to direct convolution: $O(NCd^2H'W')$
  - efficient if $K(N + C) \ll NCd$

Weight Tensor not Low-Rank

maybe the whole weight tensor is not low-rank

... but sub-structures are low-rank

(Denton et al., 2014)

Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation

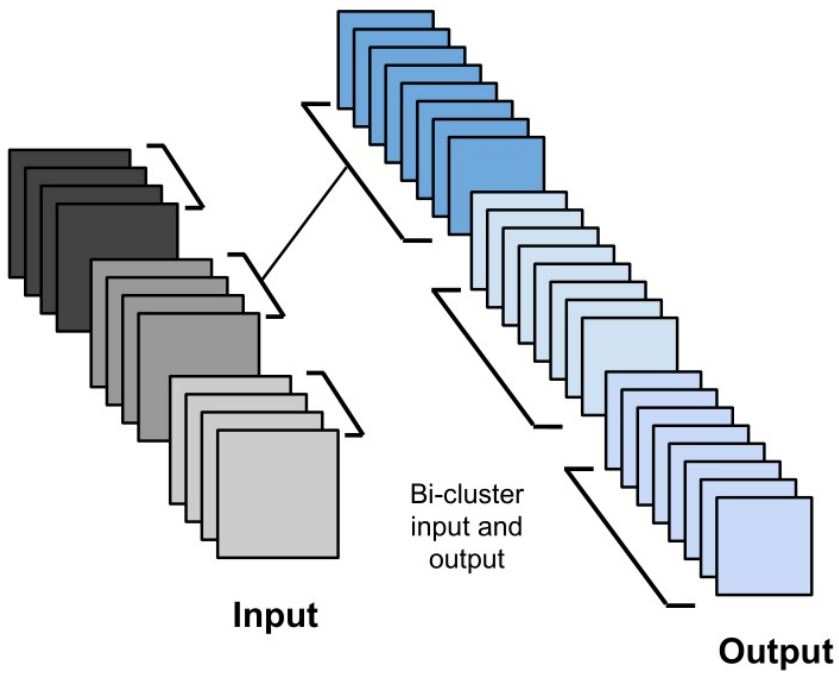
biclustering

partition the weight tensor into sub-tensors (clusters), and fit each sub-tensor with a low-rank approximation

Procedure

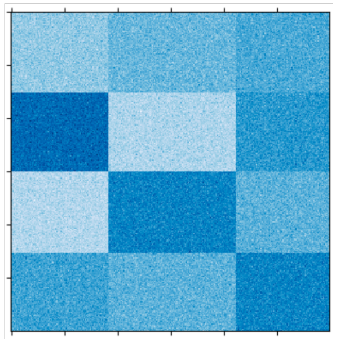
convolution weights: 4-dimensional tensor $W \in \mathbb{R}^{C \times X \times Y \times F}$

- consider $W_C \in \mathbb{R}^{C \times (XYF)}$, cluster rows $\rightarrow C_1, \ldots, C_a$
- consider $W_F \in \mathbb{R}^{(CXY) \times F}$, cluster columns $\rightarrow F_1, \ldots, F_b$
  - break the original W into ab sub-tensors $\{W_{C_i, F_j}\}$



[from (Denton et al., 2014)]

Procedure...



- each sub-tensor contains similar elements
  $\rightarrow$ easier to fit with a low-rank approximation
- find a low-rank approximation of each sub-tensor (3-tensor)
  $W_S \in \mathbb{R}^{C \times (XY) \times F}$
- (optional) fine-tuning the last FC layer

how to approximate a tensor $W \in \mathbb{R}^{m \times n \times k}$?

Low-Rank Approximation of Sub-Tensor (1)

1. singular value decomposition (SVD)

1. fold all but two dimensions together to convert it into a 2-tensor
2. perform SVD on the resulting matrix

Example

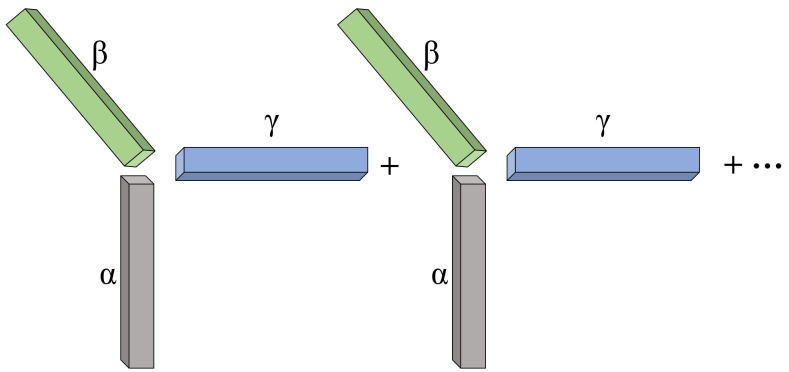
1. approximate $W_m \in \mathbb{R}^{m \times (nk)}$ as $\bar{U}\bar{S}\bar{V}^T$
2. can be further compressed by applying SVD to $\bar{V}$

Low-Rank Approximation of Sub-Tensor (2)

2. Greedy rank-1 tensor pursuit

1. approximate W by rank-1 tensor $\alpha \otimes \beta \otimes \gamma$
   - can be solved efficiently by alternating least squares
2. find the residual $(W - \alpha \otimes \beta \otimes \gamma)$, repeat K times

$$W = \sum_{k=1}^{K} \alpha_k \otimes \beta_k \otimes \gamma_k$$



Biclustering

- the previous procedure is only used for the higher convolutional layers
- simpler procedure for the first convolutional layer

A More Disciplined Low-Rank Approximation

1. need to first perform biclustering
2. need to use greedy algorithm to decompose sub-tensor

are there more disciplined approaches?

(Lebedev et al., 2015)

Speeding up convolutional neural networks using fine-tuned CP-decomposition

CP-decomposition for a d-dimensional array $\mathbf{W}$

$$W(i_1, \ldots, i_d) = \sum_{r=1}^{R} A_1(i_1, r) A_2(i_2, r) \cdots A_d(i_d, r)$$

- size of W: $n_1 \times \cdots \times n_d$
- size of CP-decomposition: $(n_1 + \cdots + n_d)R$
- can be computed by standard packages (e.g., nonlinear least squares)

Decompose Convolution Kernel using CP-Decomposition

- input tensor $U(\cdot,\cdot,\cdot) \in \mathbb{R}^{X\times Y\times S}$
- output tensor $V(\cdot,\cdot,\cdot) \in \mathbb{R}^{(X-d+1)\times(Y-d+1)\times T}$

$$V(x,y,t) = \sum_{i=x-\delta}^{x+\delta}\sum_{j=y-\delta}^{y+\delta}\sum_{s=1}^{S} K(ix+\delta, jy+\delta, s, t)U(i,j,s)$$

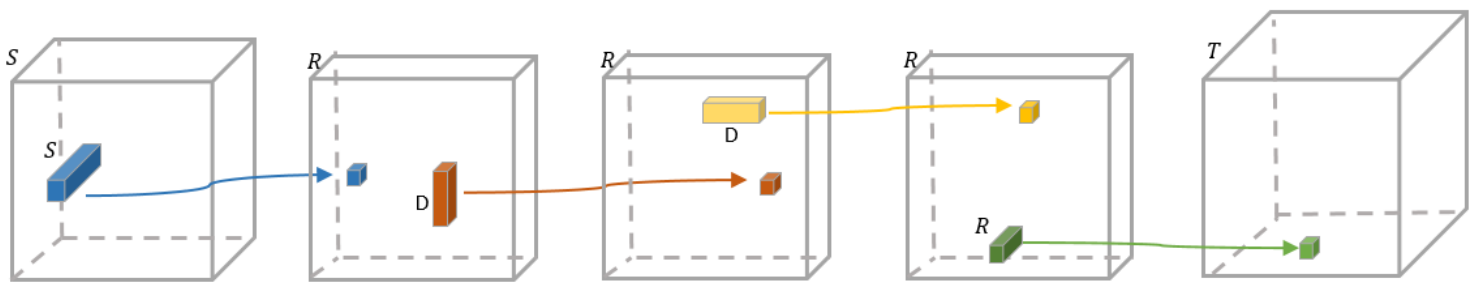
- CP-decomposition on 4D convolution kernel tensor

$$K(i,j,s,t) = \sum_{r=1}^{R} K^x(ix+\delta, r)K^y(jy+\delta, r)K^s(s,r)K^t(t,r)$$

$$V(x,y,t) = \sum_{r=1}^{R} K^t(t,r)\,\underbrace{\Big(\sum_{i=x\delta}^{x+\delta} K^x(ix+\delta,r)\,\underbrace{\Big(\sum_{j=y\delta}^{y+\delta} K^y(jy+\delta,r)\,\underbrace{\Big(\sum_{s=1}^{S} K^s(s,r)U(i,j,s)\Big)}_{1st}\Big)}_{2nd}\Big)}_{3rd}\Big)}_{4th}$$

Using CP-Decomposition in Deep Networks

1. decompose the 4D convolution kernel tensor using CP-decomposition
   - composition of four convolutions with small kernels
   - computation based on 1D-array that spans either one pixel in all input maps, or a 1D spatial window in one input map



2. fine-tune the entire network using backpropagation

Experimental Results

accuracy results for the second layers of CharNet and AlexNet

	CharNet, R=16		CharNet, R=64		CharNet, R=256		AlexNet	AlexNet	AlexNet
	NO FT	FT	NO FT	FT	NO FT	FT	R=140	R=200	R=300
RANDOM	–	9.70	–	7.64	–	6.13	–	–	–
GREEDY	24.15	2.64	4.99	0.46	1.14	-0.31	65.04	7.29	4.76
NLS	**18.96**	**2.16**	**1.93**	**0.09**	**0.31**	**-0.52**	**3.21**	**0.97**	**0.30**

- R: rank; FT: fine-tuning; "random": random initialization
- numbers correspond to accuracy drop (original networks achieve 91.24% (CharNet) and 79.95% (AlexNet))
- greedy algorithms perform worse than more nonlinear least squares (NLS)

Efficient Designs (mainly for CNNs)

Fully Connected Layer

usual CNN architecture

convolution $\rightarrow \cdots \rightarrow$ convolution $\rightarrow$ fully connected layer $\rightarrow$ output

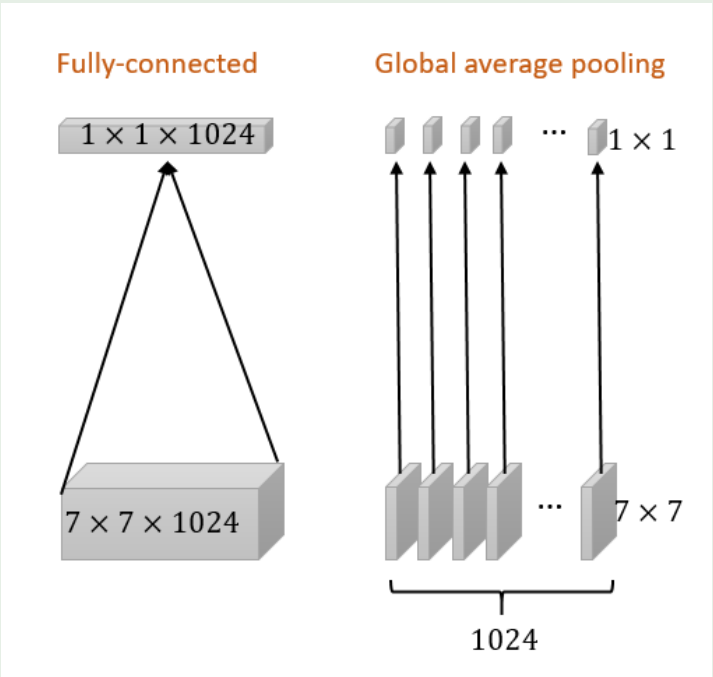
most of the model parameters are in fully connected layers

- get rid of the FC layer
- GoogleNet, ResNet

Global Average Pooling

- reduce each $h \times w$ feature map to a single number by simply taking the average of all hw values

Example



number of parameters

- fully-connected layer: $7 \times 7 \times 1024 \times 1024 = 51.3\text{M}$
- global average pooling: 0

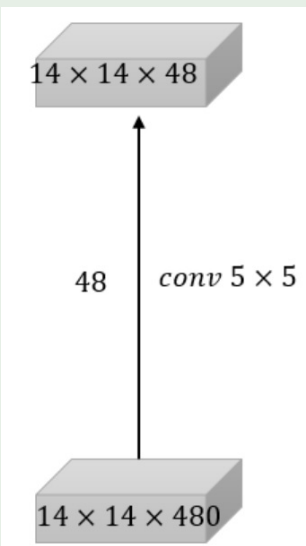
Many Input Feature Maps

cost: $D_K^2 M N D_F^2$

many input feature maps

- large space and time complexities, particularly when the convolutional operation being performed is large

Example



- # parameters: $5 \times 5 \times 480 \times 48 = 576\text{K}$
- # operations:
  $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9\text{M}$

1×1 Convolution

how to reduce the number of feature maps?

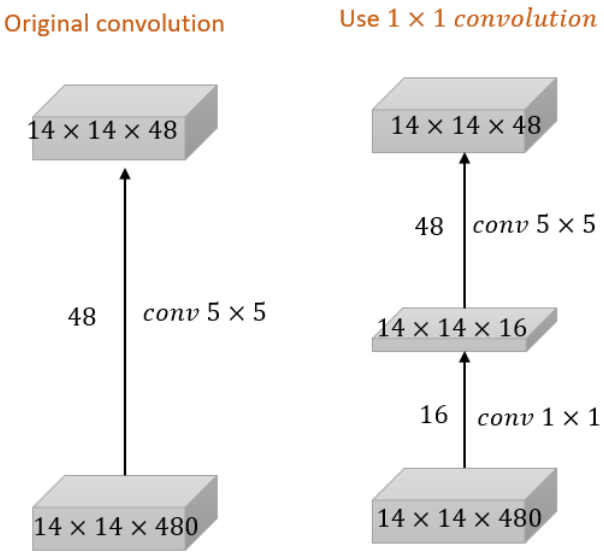
pooling?

- can only reduce width and height of feature maps

1×1 convolution

- perform convolution without looking at neighboring pixels

Example



original:

- # parameters: 576K
- # operations: 112.9M

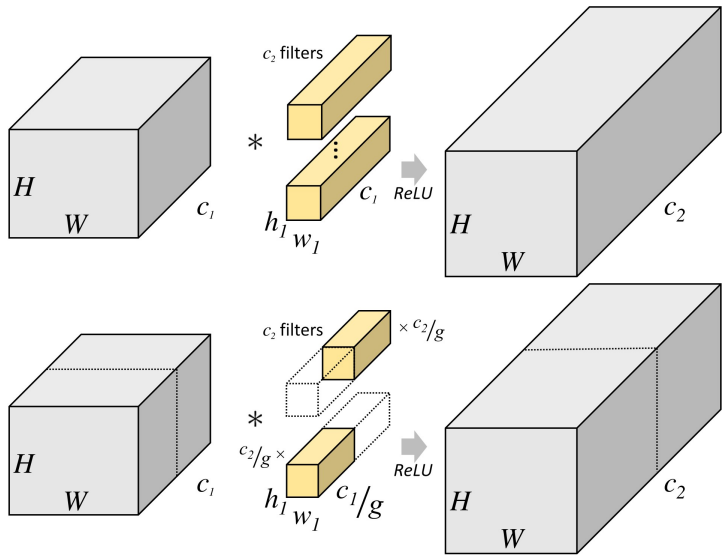
use 1×1 convolution

- # parameters: $1 \times 1 \times 480 \times 16 + 5 \times 5 \times 16 \times 48 = 26.88\text{K}$
- # operations: $(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = 5.3\text{M}$

- dimension reduction $\rightarrow$ reduce computation
- can also be used to increase the number of feature maps

Group Convolution

group the channels



[from `https://blog.yani.io/filter-group-tutorial/`]

- perform convolution independently for each grouped channel
- with two filter groups
  - each filter is exactly half the number of parameters
  - each output channel only relates to the input channels within the group $\rightarrow$ computation also halved

MobileNet

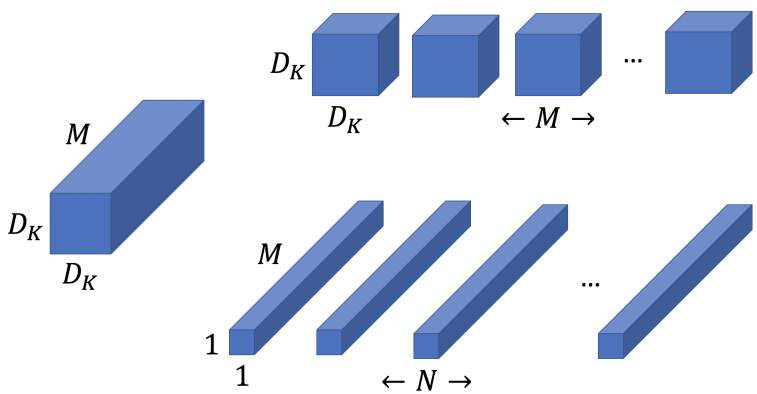
(Howard et al., 2017)

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

replace standard convolution by depthwise separable convolution (depthwise convolution + pointwise convolution)

depthwise convolution

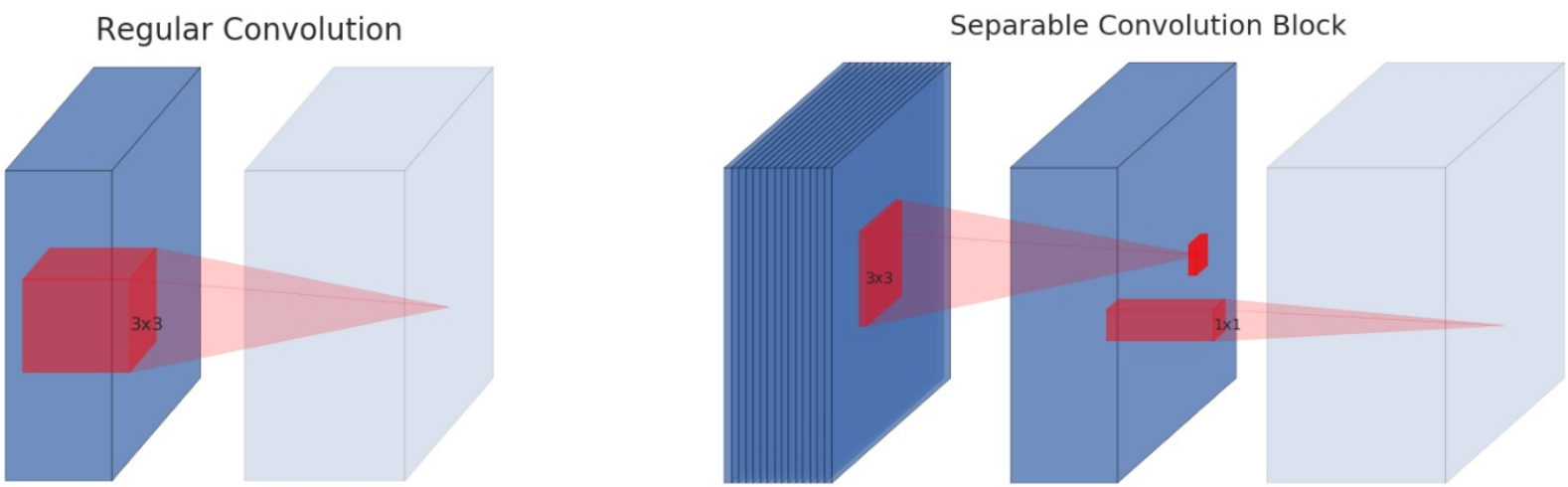
- one filter per channel (depth)



pointwise convolution

- 1×1 convolution
- combines information from channels

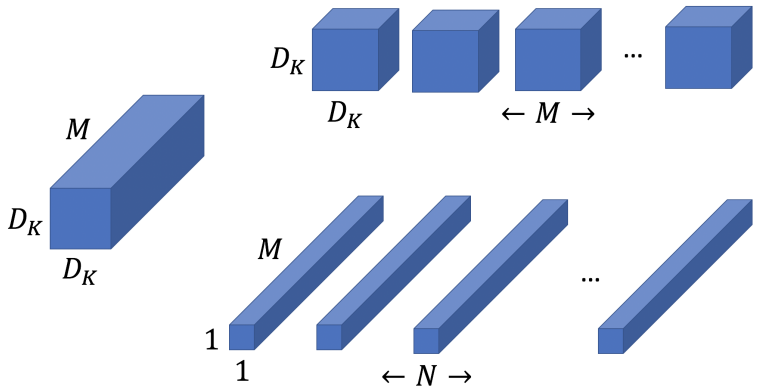
Depthwise Separable Convolution



[from (Sandler et al, 2018)]

Cost

computing the standard convolution: $D_K^2 M N D_F^2$
depthwise separable convolution



- depth-wise convolution: $D_K^2 M$
- 1×1 convolution: M
- total: $D_K^2 M D_F^2 + M N D_F^2$

- speedup: $D_K^2 N / (D_K^2 + N)$
  - e.g., $D_K = 3, N = 32$, speedup $= 7$
- empirically, 9 times less work than comparable neural nets with the same accuracy

MobileNetV2

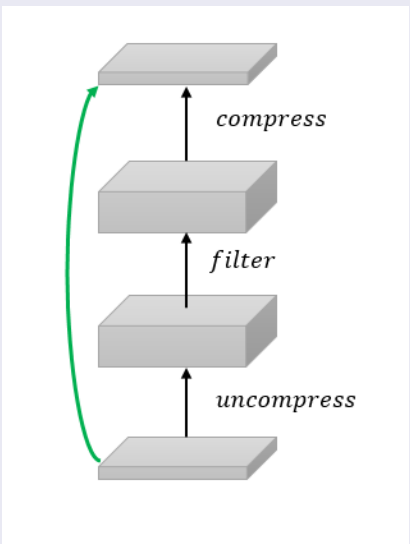
dilemma

- reduce number of computations → prefer small tensors
- extract more information → prefer large tensors

(Sandler et al., 2018)

MobileNetV2: Inverted Residuals and Linear Bottlenecks

inverted bottleneck structure

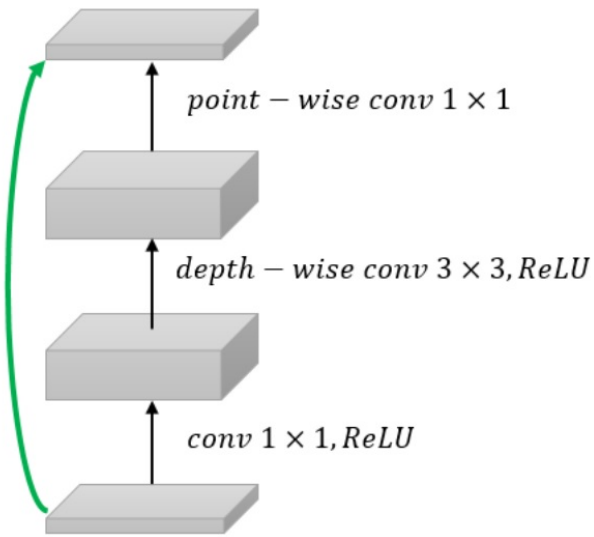


- compressed representation
  → uncompress (increase expressiveness)
  → filter
  → compress → compressed representation
- residual connections as in ResNet

Inverted Bottleneck Structure

input: a low-dimensional tensor

1. (uncompress) expand it to high dimension, using 1×1 convolution
2. filter with depthwise convolution
3. (compress) project features back to a low-dimensional tensor, using 1×1 convolution (without nonlinearity)



- 1×1 convolutions (for expansion and projection) have learnable parameters
- the model can learn how to decompress/filter/compress

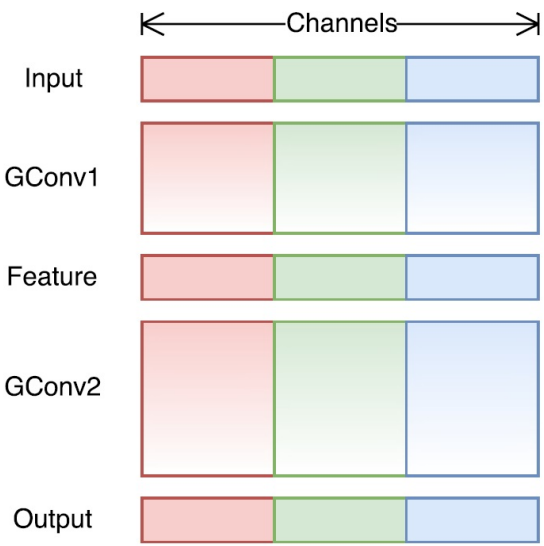
1×1 Convolution (conv1x1) is Expensive

conv1x1 becomes the most expensive bottleneck

(Zhang et al, 2017)

ShuffleNet: An extremely efficient convolutional neural network for mobile devices

replace conv1x1 by grouped conv1x1



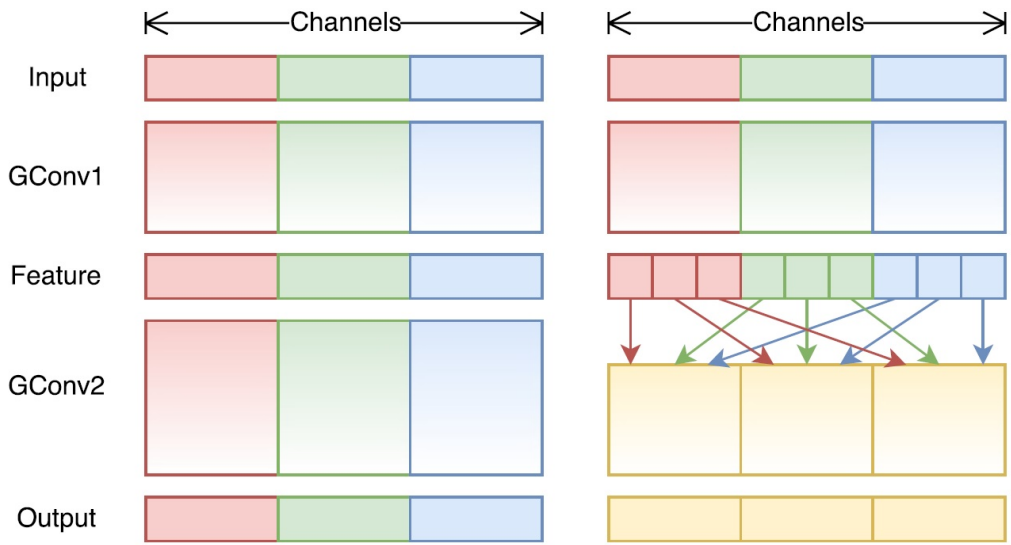
[from (Zhang et al, 2017)]

- each output channel only relates to the input channels within the group
- blocks information flow between channel groups

Channel Shuffle

shuffles the order of the channels among groups

- allow group convolution to obtain input data from different groups $\rightarrow$ input and output channels will be fully related



[from (Zhang et al, 2017)]

Neural Architecture Search

Motivation

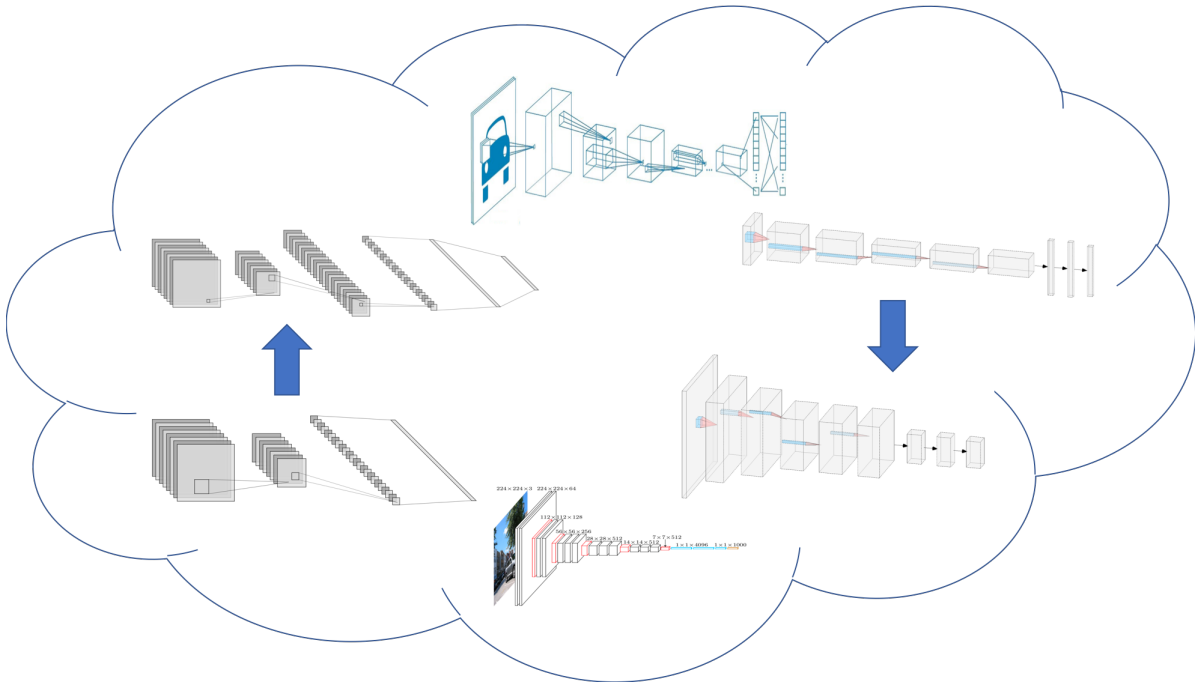
conventional model compression are heuristic

- sub-optimal, time-consuming

can we find a compressed model automatically?

Neural architecture search (NAS)

Components in NAS: 1. Search Space



what architectures to search?
how fine-grained?

1. macro search
2. micro search

Components in NAS: 2. Search Strategy

search space is often very large. how to search?

- reinforcement learning (RL)
- evolutionary algorithm
- gradient-based methods
- ...

Components in NAS: 3. Performance Measure

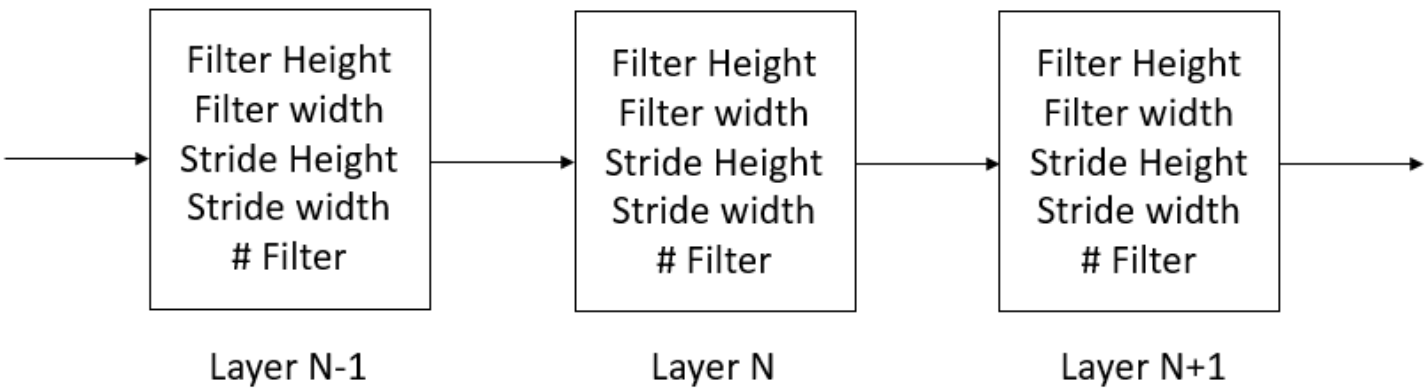
how to measure performance of the architecture on unseen data?

what is "performance"?

Macro Search

search for detailed structure of the network

Example (CNN)

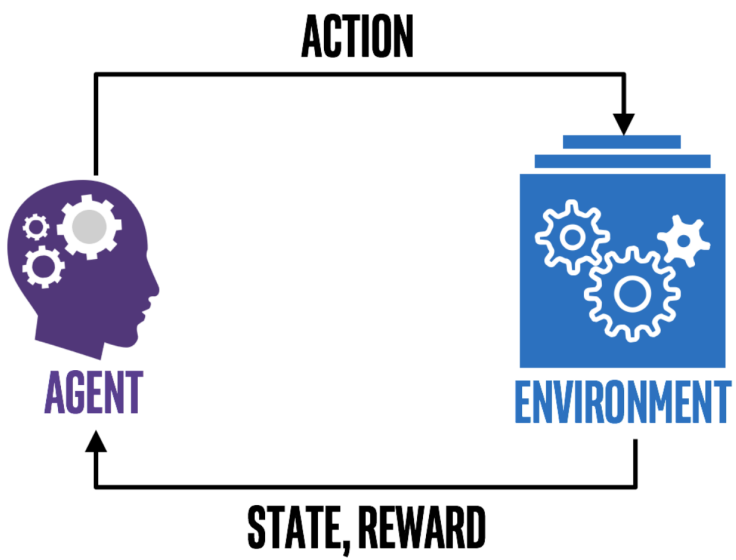


NAS

(Zoph & Le, 2016)

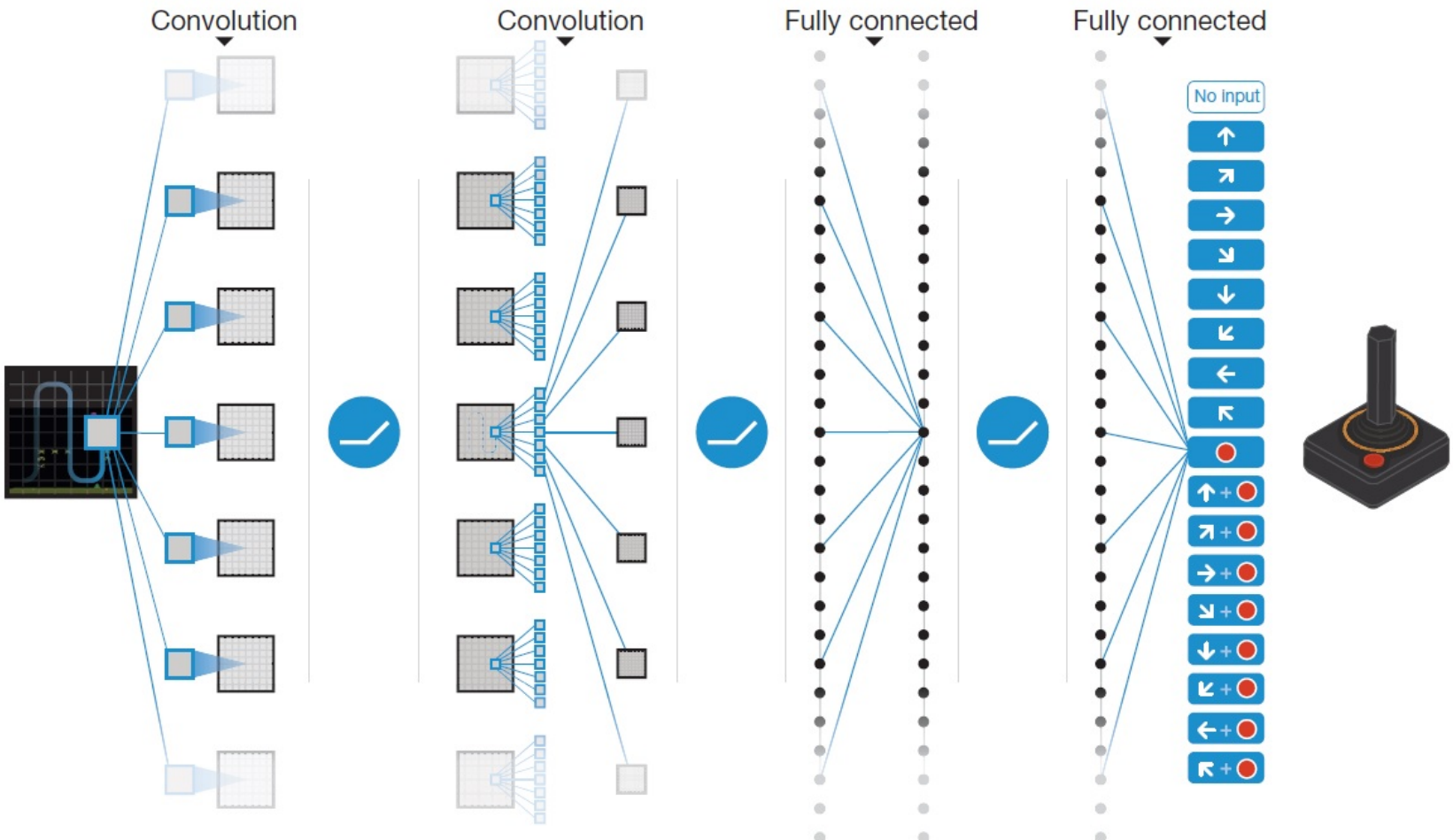
Neural architecture search with reinforcement learning

use RL to design every operation in the entire network



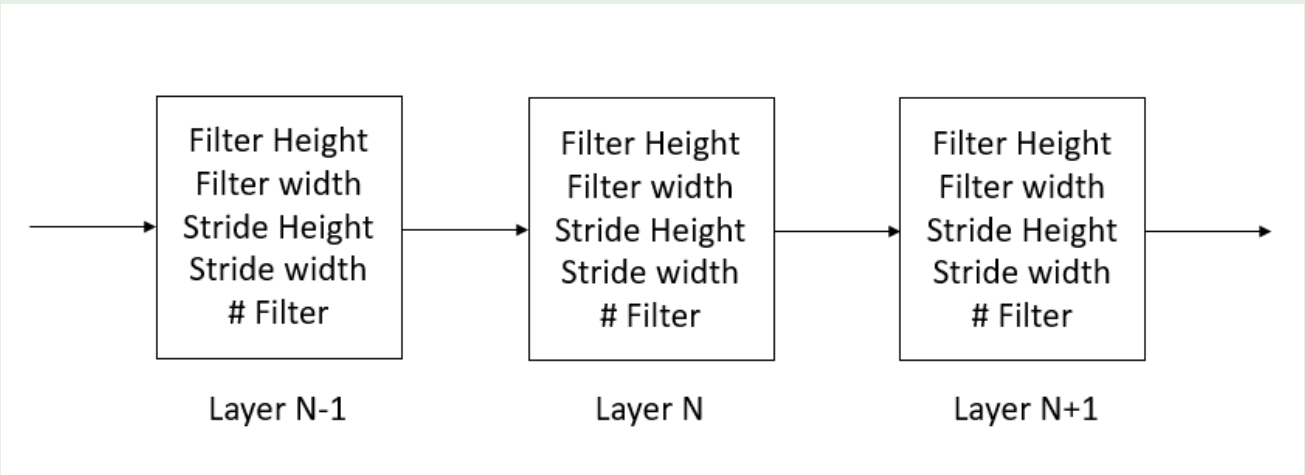
- agent maximizes the long-term reward from the actions

Example RL Application



Definition of RL Components in NAS

Example (search for a simple CNN)



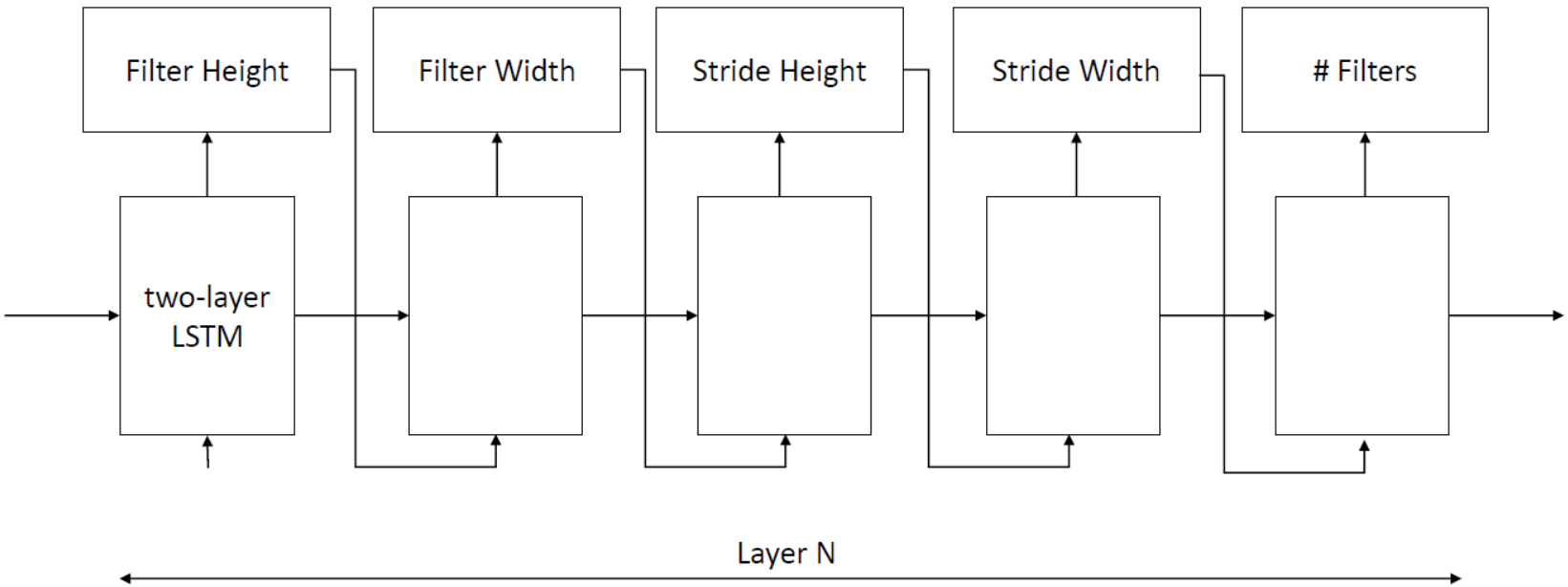
actions: decisions to build child model architecture

- for each layer, predict filter height, filter width, stride height, stride width, number of filters

reward: after taking a given number of T actions, evaluate validation accuracy of the child network

Definition of RL Components in NAS...

agent: a recurrent network controller outputting the actions



- different choices output from softmax classifier
- current predicted action depends on the previous actions
- sample predictions (from softmax classifier) and feed to the next time step as input

Learning Procedure

1. controller samples architecture according to probabilities from softmax
2. trains a child network with this architecture
3. obtains validation accuracy
4. updates controller
   - policy gradient method: directly optimizes controller parameter

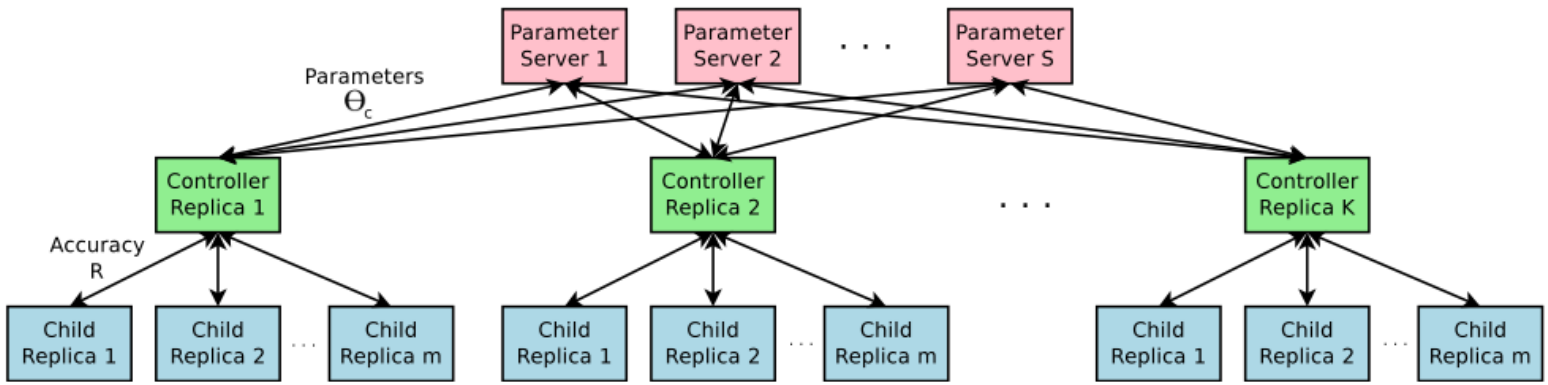
policy gradient

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{t-1};\theta_c)R_k$$

- m: number of different architectures the controller samples in one batch
- T: number of hyperparameters the controller has to predict to design a neural network architecture
- R_k: accuracy of child network

Distributed Training

training is expensive



[from (Zoph & Le, 2016)]

- each controller replica samples m architectures and run the multiple child models in parallel
- accuracy of each child model is recorded to compute the gradients, which are then sent back to the parameter servers

Inefficiency of Macro Search

NAS is computationally expensive on large datasets

- CNNs often identifies repeated motifs (e.g., inception module in GoogleNet)

micro search

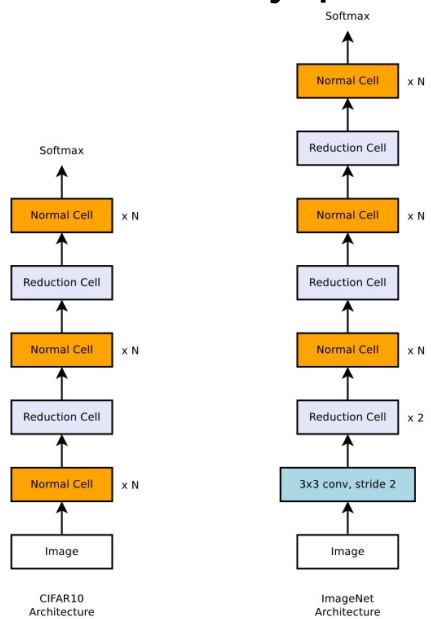
controller designs modules, which are then stacked in series to construct the network

Macro Search: NASNet (for CNN)

(Zoph et al., 2017)

Learning Transferable Architectures for Scalable Image Recognition

- overall CNN architecture manually predetermined



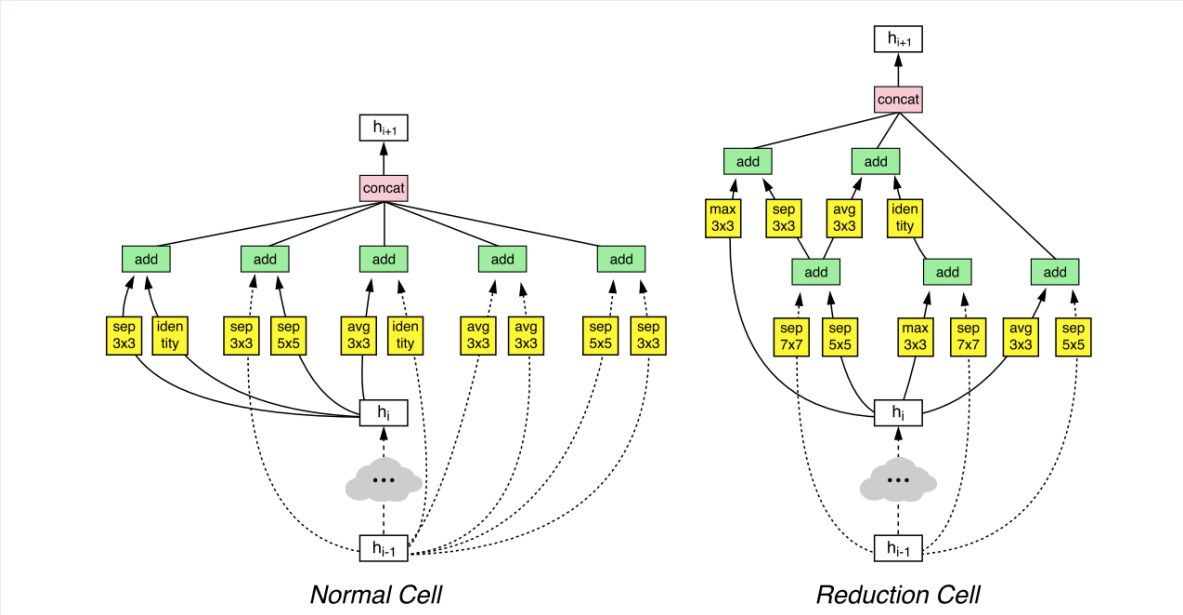
[from (Zoph et al, 2017)]

- module: generic convolutional cell (two types)
  1. normal cell: feature map of the same dimension as input
  2. reduction cell: feature map where the feature map height and width is reduced by a factor of two

Search for Cells in NASNet

- structures of the cells are searched by controller RNN
- actions (operations to apply)
  - identity; 1×3 then 3×1 convolution; 1×7 then 7×1 convolution; 3×3 average pooling; 3×3 max pooling; 5×5 max pooling; 3×3 depthwise-separable conv; 5×5 depthwise-separable conv; etc

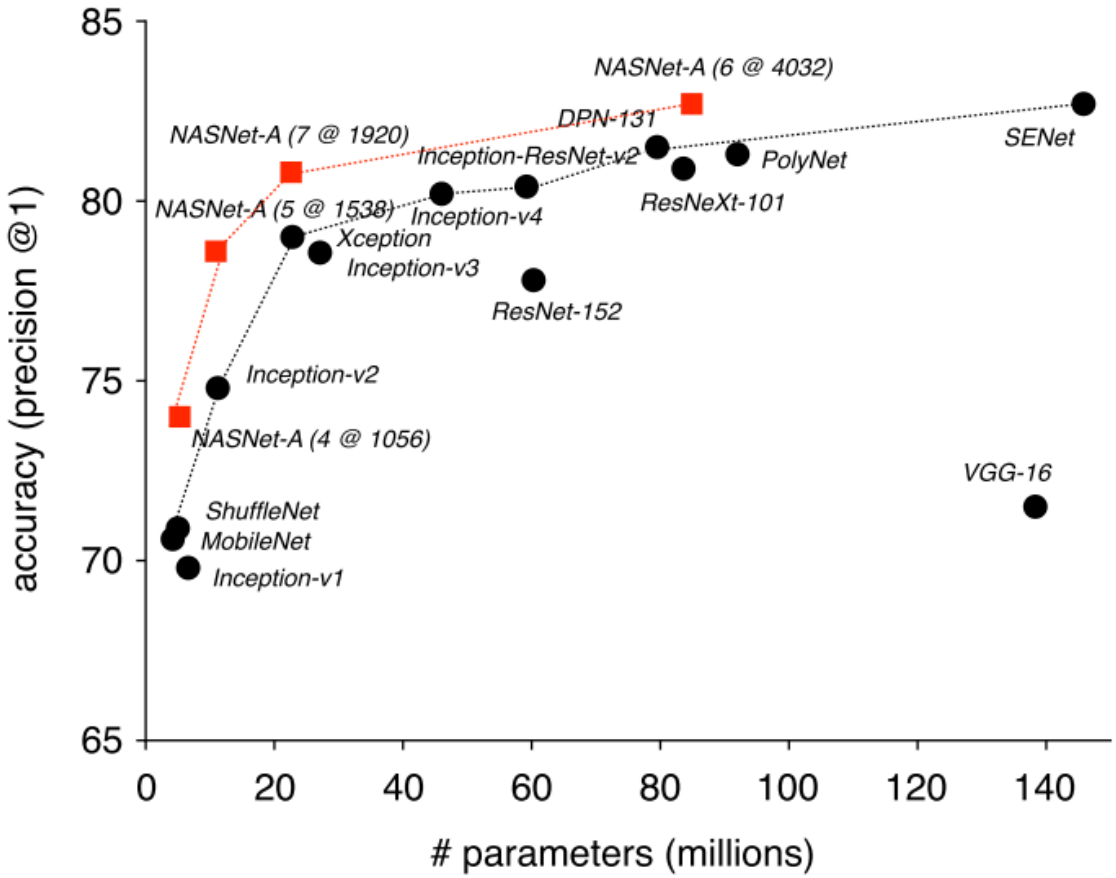
Example (best convolutional cells with CIFAR-10)



[from (Zoph et al, 2017)]

Experimental Results

accuracy vs number of parameters on ImageNet



- faster search: 7x less expensive than standard NAS
- smaller model

NAS for Compressed Networks

NAS +

- sparsification
- quantization
- ...

network sparsification/quantization

reduce number / quantize weights from a pre-trained network

this is an easier problem!

no need to search for an entirely different architecture

what NAS can offer?

NAS + Sparsification

limitation for standard sparsification schemes

fixed pre-defined sparsity level for all layers

(He et al., 2018)

AMC: AutoML for Model Compression and Acceleration on Mobile Devices

- automatically compress a pre-trained network layer-by-layer
- advantage: allows variable sparsity for layers
- formulation as a RL problem

RL Formulation

action

- target sparsity for each layer

rewards

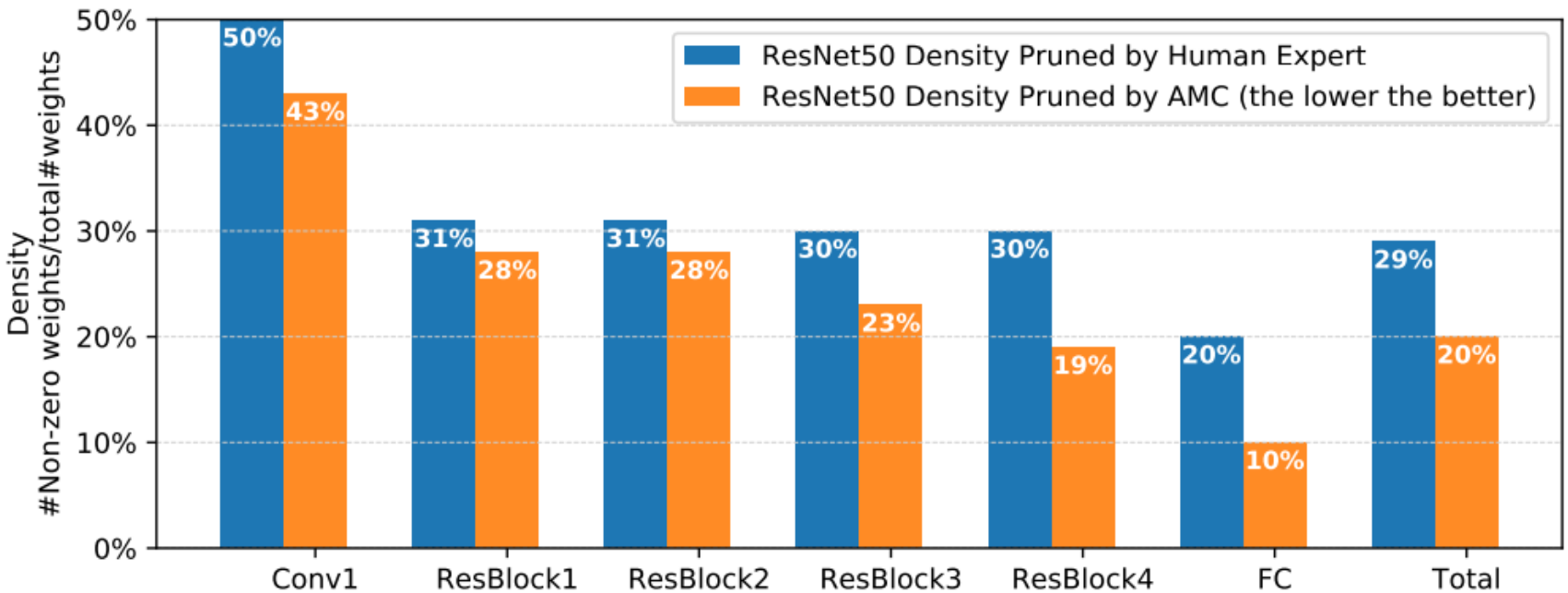
1. resource budget (e.g., model size) is given
   - achieve the best accuracy given the maximum amount of hardware resources
   - reward: $-$error
   - constrain the action space to ensure that the compressed model is below resource budget
2. no explicit resource budget
   - objective is to have low error and small model size
   - reward: $-\text{error} \times \log(\#\text{parameters})$

AMC Procedure

- for each layer
  1. obtain sparsity (action) of this layer
  2. compress (using fine-grained pruning or structured pruning)
- after compressing all layers: evaluate reward
- update controller

Empirical Results

AMC vs human experts on ResNet50



NAS + Quantization

(Wang et al., 2019)

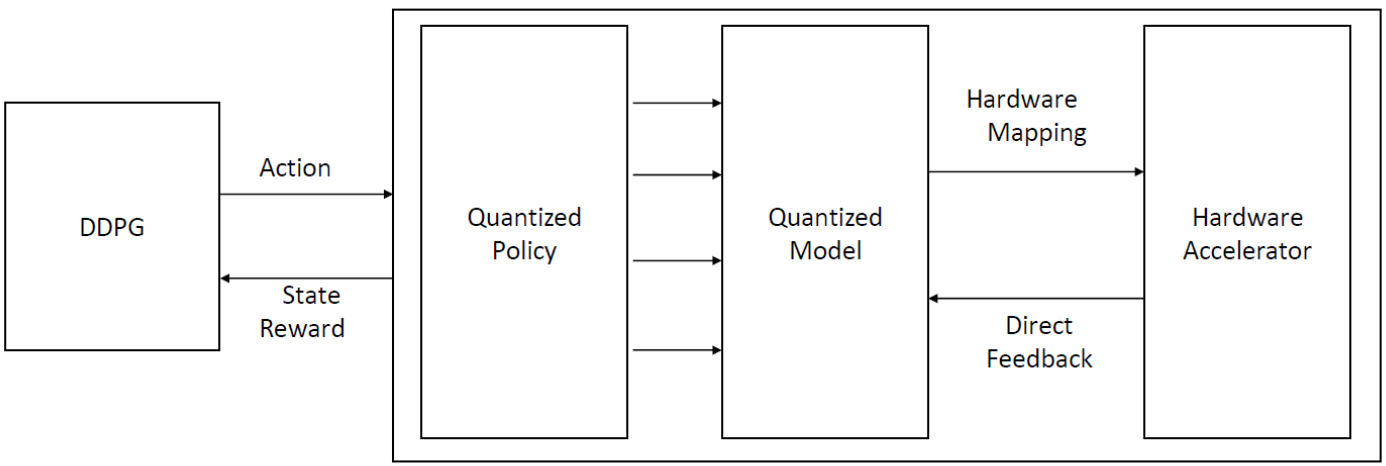
HAQ: Hardware-Aware Automated Quantization with Mixed Precision

advantages over standard quantization methods

given the amount of computation resources (i.e., latency, power, and model size), determine the bitwidth of both weights and activations for each layer automatically

Quantization as RL Problem

- state: characteristics of the layer (e.g., number of input/output channels)
- action: bits for each layer
  - resource constraints used to constrain the action space
  - after RL agent gives actions to all layers, measure the amount of resources that will be used by the quantized model
  - if resource budget exceeded, decrease bitwidth of each layer until the constraint is finally satisfied
- reward: accuracy $R = \lambda \times (\text{acc}_{\text{quantized}} - \text{acc}_{\text{orig}})$



Experimental Results

		Edge Accelerator						Cloud Accelerator					
		MobileNet-V1			MobileNet-V2			MobileNet-V1			MobileNet-V2		
	Bitwidths	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency
PACT [3]	4 bits	62.44	84.19	45.45 ms	61.39	83.72	52.15 ms	62.44	84.19	57.49 ms	61.39	83.72	74.46 ms
Ours	*flexible*	**67.40**	**87.90**	45.51 ms	**66.99**	**87.33**	52.12 ms	**65.33**	**86.60**	57.40 ms	**67.01**	**87.46**	73.97 ms
PACT [3]	5 bits	67.00	87.65	57.75 ms	68.84	88.58	66.94 ms	67.00	87.65	77.52 ms	68.84	88.58	99.43 ms
Ours	*flexible*	**70.58**	**89.77**	57.70 ms	**70.90**	**89.91**	66.92 ms	**69.97**	**89.37**	77.49 ms	**69.45**	**88.94**	99.07 ms
PACT [3]	6 bits	70.46	89.59	70.67 ms	71.25	90.00	82.49 ms	70.46	89.59	99.86 ms	71.25	90.00	127.07 ms
Ours	*flexible*	**71.20**	**90.19**	70.35 ms	**71.89**	**90.36**	82.34 ms	**71.20**	**90.08**	99.66 ms	**71.85**	**90.24**	127.03 ms
Original	8 bits	70.82	89.85	96.20 ms	71.81	90.25	115.84 ms	70.82	89.85	151.09 ms	71.81	90.25	189.82 ms

- reduce latency by 1.4X to 1.95X with negligible loss of accuracy compared with 8-bit quantization

Repositories for Fast Inference

Repositories for Fast Inference

Google

- Tensorflow Lite (`https://www.tensorflow.org/lite`)
  - hardware acceleration on supported devices, device-optimized kernels, and pre-fused activations and biases
  - supports 8-bit quantization
- learn2compress (`https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html`)
  - takes as input a large pre-trained TensorFlow model and automatically generates smaller ready-to-use on-device models
  - supports pruning, quantization, joint training and distillation

Facebook

- QNNPACK (`https://zhuanlan.zhihu.com/p/48078493`)
  - supports 8-bit quantization for mobile devices

Repositories for Fast Inference...

Intel

- Distiller (`https://github.com/NervanaSystems/distiller`)
  - an open-source Python package for neural network compression research
  - implements popular sparsification, quantization, distillation methods

Tencent

- NCNN (`https://github.com/Tencent/ncnn`)
  - hardware optimization for mobile devices
  - supports 8-bit, 16-bit representation
- PocketFlow (`https://github.com/Tencent/PocketFlow`)
  - an automatic model compression (AutoMC) framework with support for sparisifed or quantized models

Conclusion

- deep learning models are powerful
- but we don't want to carry big models around

take-home message

- capacity of deep network is usually larger than needed
- can be compressed without accuracy degradation