

EMBEDDED MACHINE LEARNING LECTURE 05 - NEURAL ARCHITECTURE DESIGN

Bernhard Klein & Holger Fröning

bernhard.klein@ziti.uni-heidelberg.de

Computing Systems Group, Institute of Computer Engineering
Heidelberg University

Achievement Data Migration heiCO In Summary



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

heiCO
CAMPUSonline

May 9th, 2024
Continued availability of all essential functionalities, except for changing degree programs and majors.

[Click here](#) to visit the heiCO system.

heiCO
CAMPUSonline

Alternative System
For access from remote, set up VPN!

May 21st, 2024
Opening of alternative system with **read-only access**.

[Click here](#) to visit the heiCO system.

heiCO
CAMPUSonline

June 1st, 2024
heiCO-System is available as usual.
Achievement data / grades will be visible from **June 15th at the earliest, adjustments** will continue beyond June 15th.

[Click here](#) to visit the heiCO system.



DOs

- Viewing achievements/credits
- Searching for courses, viewing information and dates
- Viewing exam dates
- Viewing student dossier
- Downloading and printing documents in the student dossier
- Downloading and printing proofs and certificates



Alternative System



DON'Ts

- Register for a course
- Register for an exam

Related Links

All information regarding the achievement data migration:

<https://www.uni-heidelberg.de/en/teaching-services-portal/heidelberg-campus-online/achievement-data-migration-may-21-to-may-31-2024>

heiCO FAQs:

<https://www.uni-heidelberg.de/en/faqs-students>

Manuals:

<https://www.uni-heidelberg.de/en/teaching-services-portal/heidelberg-campus-online/heico-downloads>

heiCO-System:

<https://heico.uni-heidelberg.de/heico/ee/ui/ca2/app/desktop/#/login>

heiCO-Website:

<https://www.uni-heidelberg.de/en/teaching-services-portal/heidelberg-campus-online>

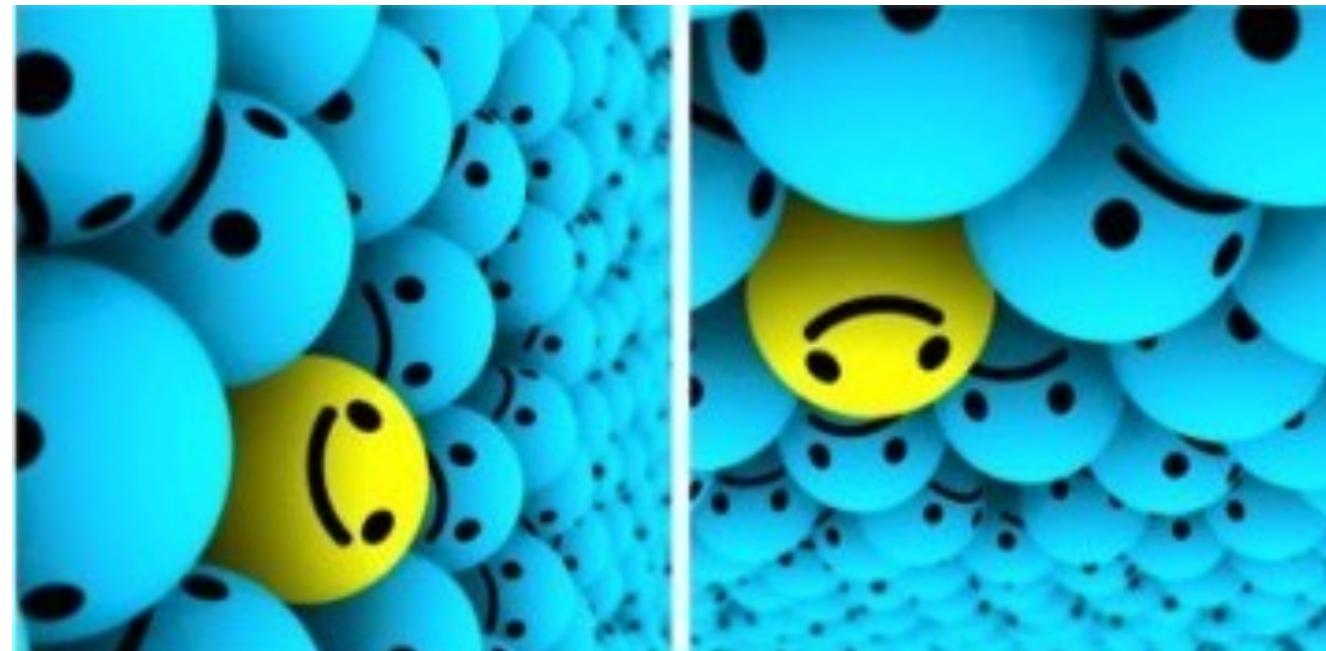
Website Universität Heidelberg – Study

<https://www.uni-heidelberg.de/en/study>

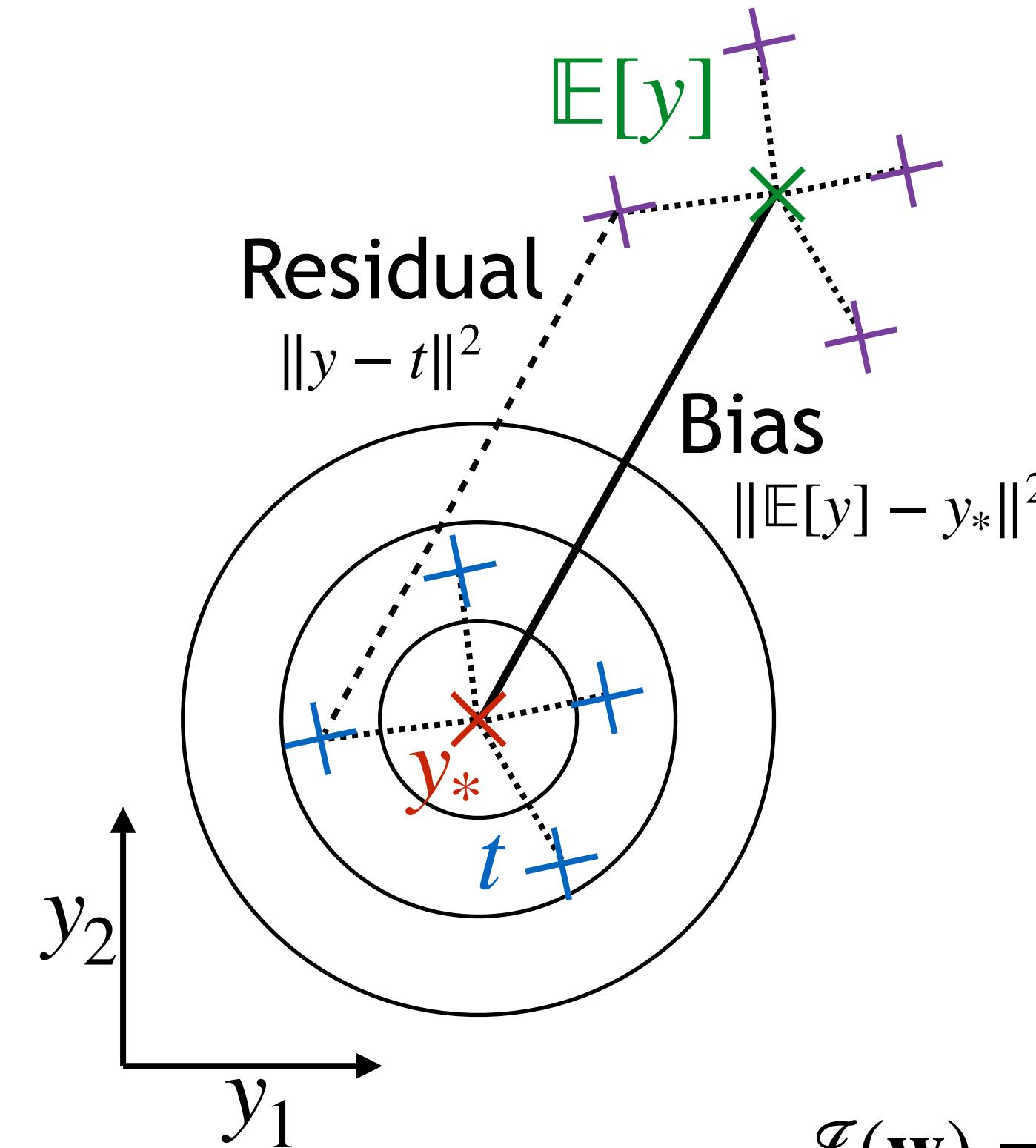
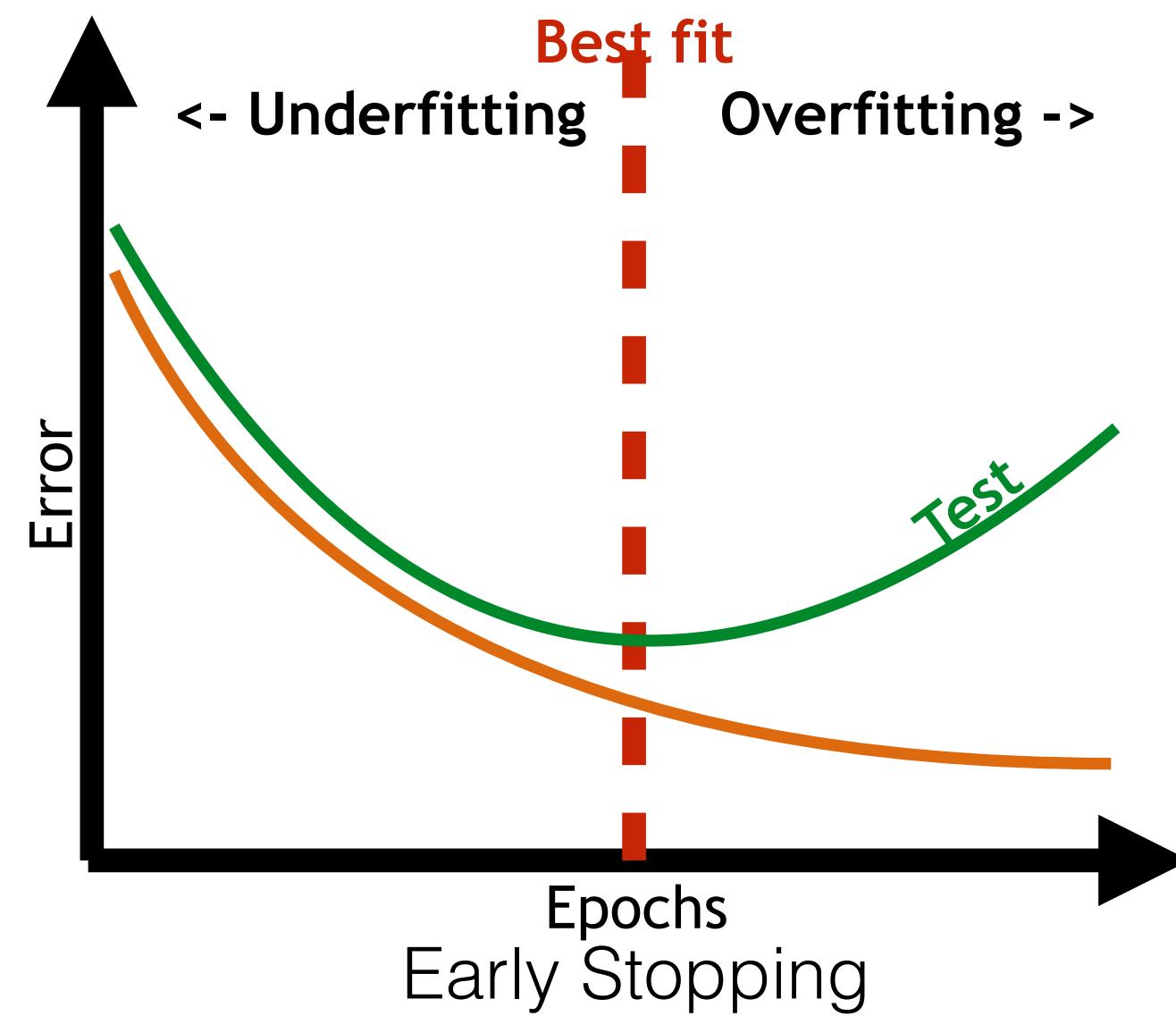
heiCO-Helpdesk:

heico-support@uni-heidelberg.de

RECAP: REGULARIZATION

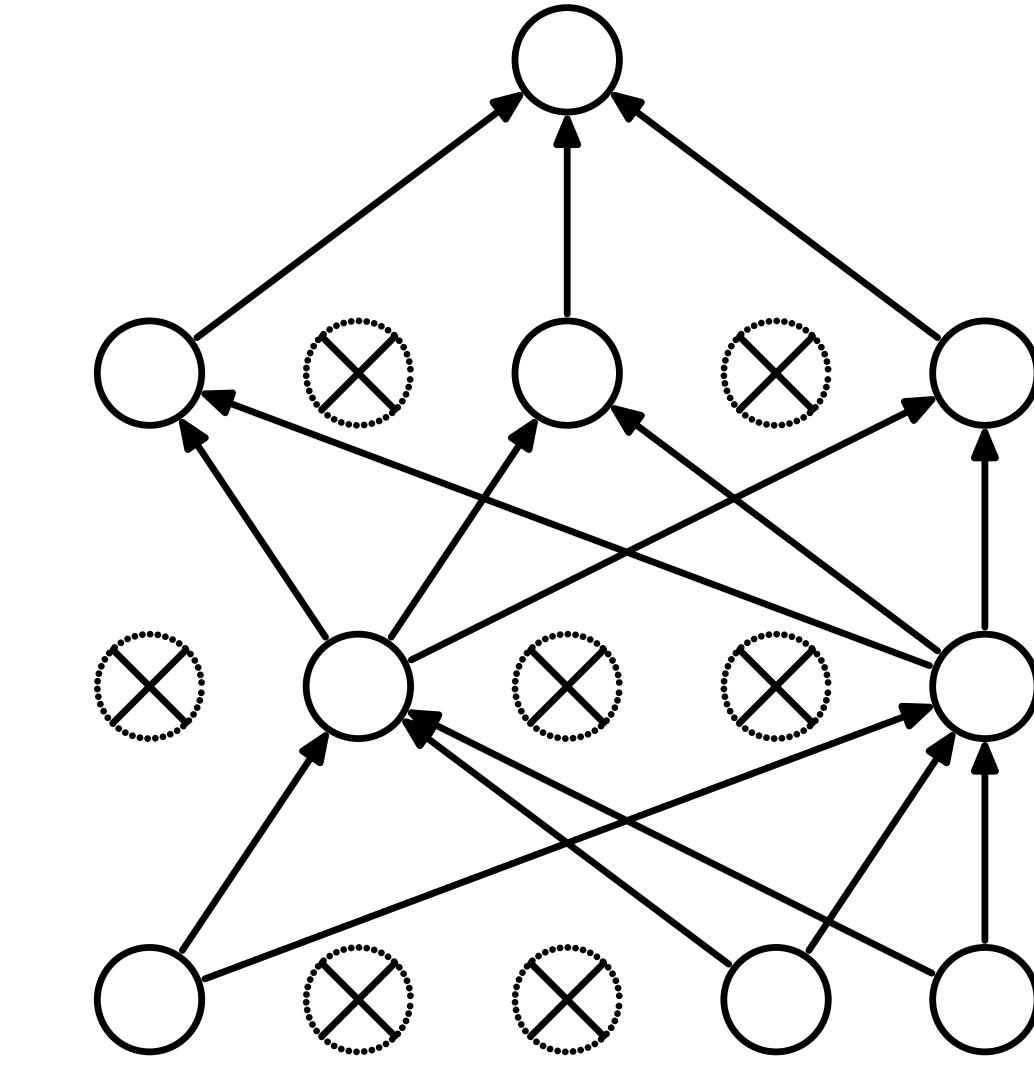


Data Augmentation



$$\mathcal{J}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(y(\mathbf{w}, \mathbf{x}_n), t_n) + \lambda \mathcal{R}(\mathbf{w})$$

Weight Decay L1/L2



Dropout

NORMALIZATION

BATCH NORMALIZATION

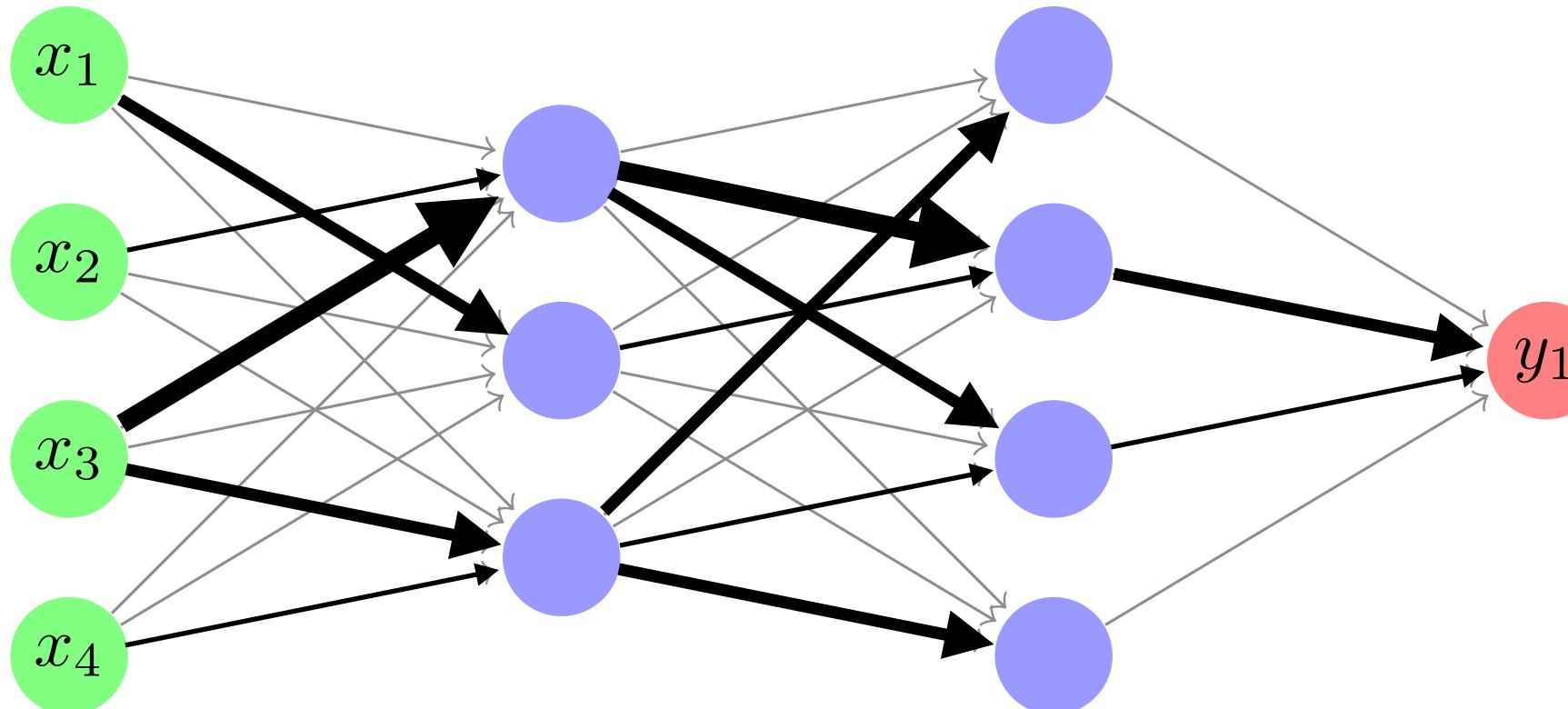
Batch normalization is a method of adaptive reparametrization, motivated by the difficulty of training deep models

Coordinating updates across different layers is difficult, due to second-(and higher)-order interactions

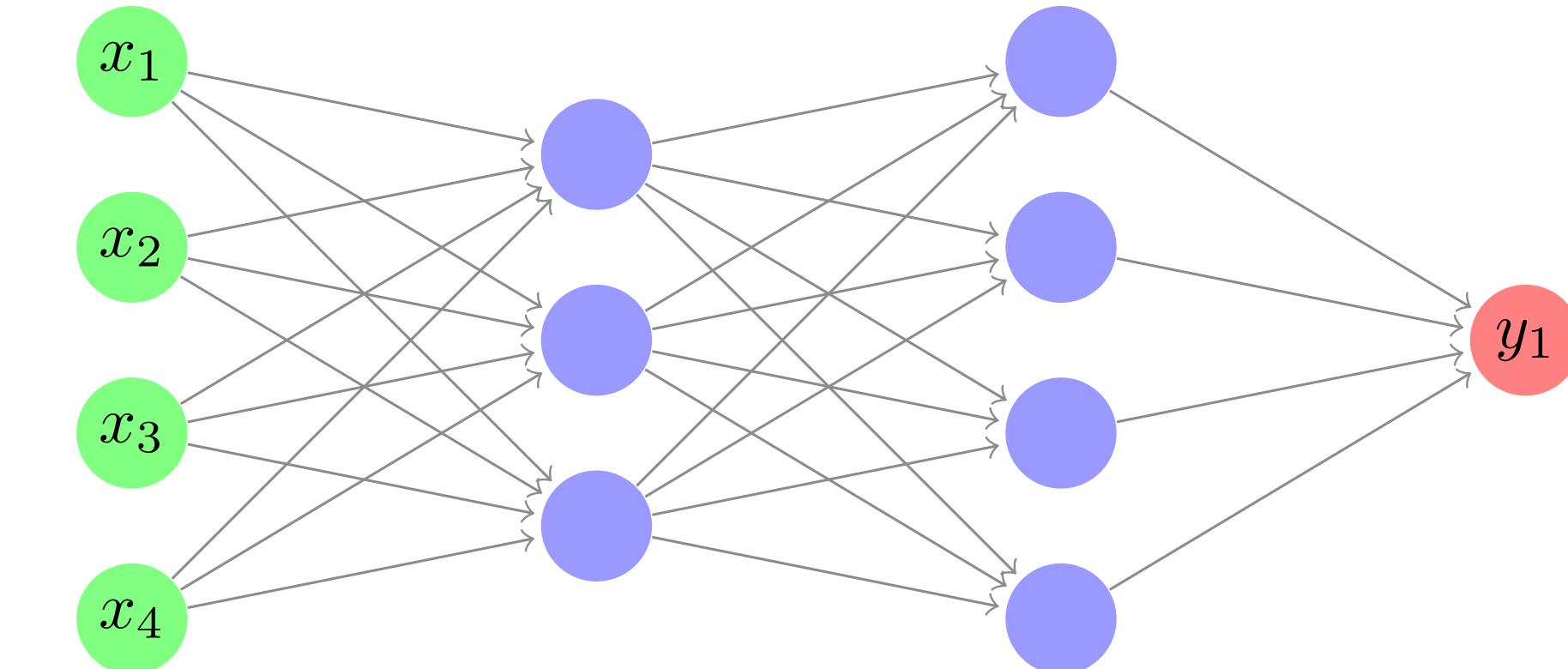
Reparametrization can help to coordinate updates across many layers

E.g., by standardizing the activations

Standardization refers to rescaling data to have a mean of zero and a standard deviation of one, i.e., a standard Gaussian



Raw signal, high interdependency between distributions
=> slow and unstable training



Normalized signal, mitigated interdependency
between distributions => improved training

BATCH NORMALIZATION

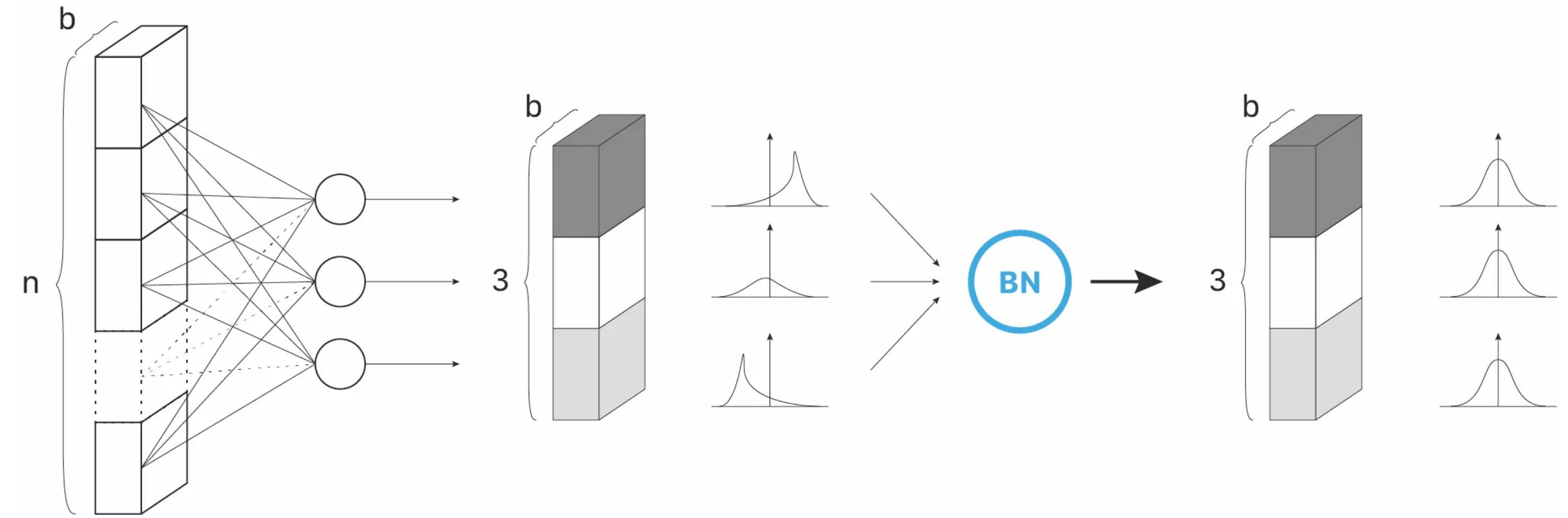
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
 Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$


Example of 3 hidden layers with a batch size b

ϵ is only used for numerical stability (avoid division by zero)

γ and β are trainable parameters to scale and shift the distribution

Counter-intuitive as we re-introduce non-zero mean and non-unit variance

But helpful for learning dynamics, as the mean of x_i is determined by complicated interactions between the parameters of the previous layers, while the mean of scaled and shifted \hat{x}_i is solely determined by $\beta \Rightarrow$ a parametrization that is much easier to learn with GD

S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015

<https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338>

BATCH NORMALIZATION - HINTS

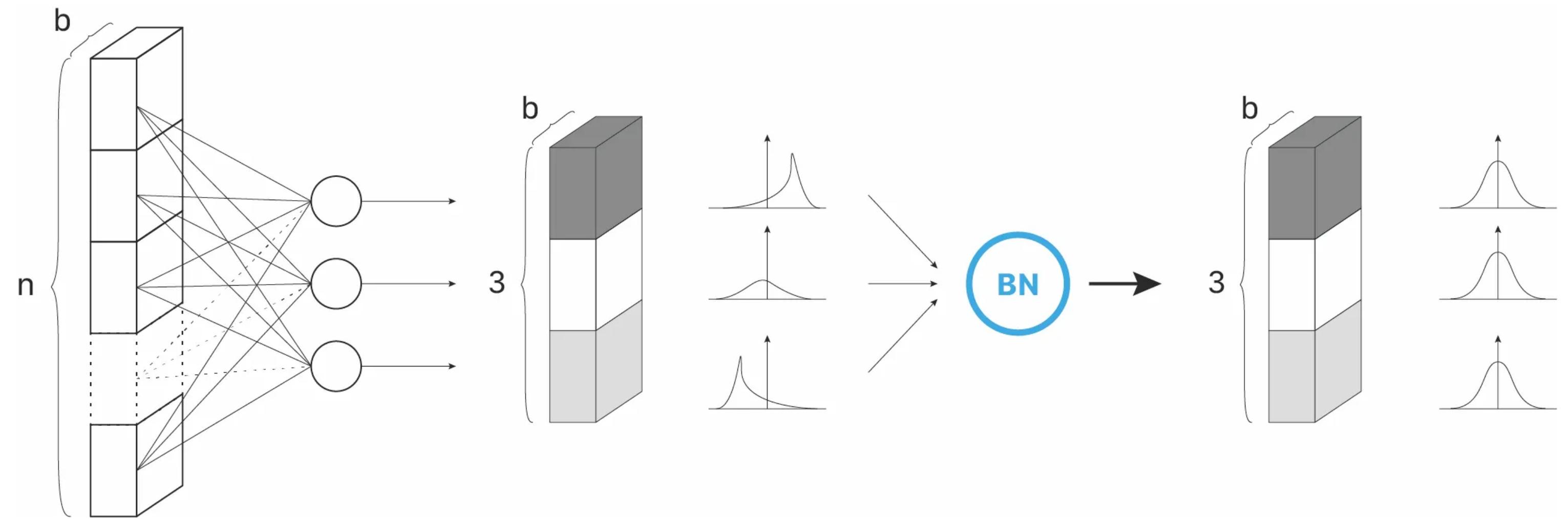
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
 Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$


Example of 3 hidden layers with a batch size b

1. Replace $\mathbf{Wx} + \mathbf{b}$ with \mathbf{Wx} , as bias is now redundant with β

2. BN can be applied before or after the non-linearity

Before: original paper; after: often works out better

3. Don't use in combination with dropout
 As a rule of thumb

4. At test time, replace μ and σ with running averages (or other approximations)

Enabling arbitrary mini-batch sizes: $\mu \Rightarrow \mu'$, $\sigma \Rightarrow \sigma'$

WHY DOES BN WORK OUT?

The shifting of distribution between training and testing set is called “covariate shift”

Theory 1: BNs reduce the so-called internal co-variate shift (original paper) [1]

Even though μ' and σ' are approximations, they (usually) make the test set “similar enough” to the training set => better generalization

Other work also refers to covariate shift among different training samples and as weights are learned

Other theories:

Theory 2: BN mitigates the interdependency between layers during training [2]

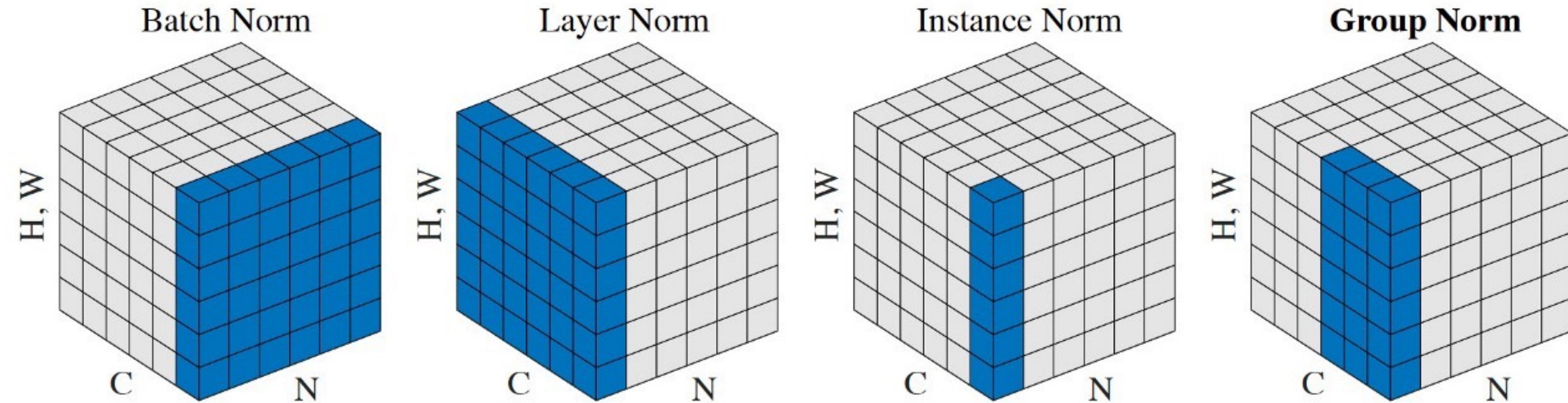
Theory 3: BN smoothes the optimization landscape [3]

[1] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015

[2] <https://www.youtube.com/watch?v=Xogn6veSyxA>

[3] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry, “How Does Batch Normalization Help Optimization?”, arxiv.org/abs/1805.11604

NORMALIZATION



BN is highly dependent on mini-batch size. Alternatives:

LN operates along channel dimensions

IN performs BN-like computations for each sample

GN is independent of batch size (similar to LN, IN), but shows comparable performance to BN

CONVOLUTIONAL ARCHITECTURES

RECAP: ANN TRENDS

	LeNet 5	AlexNet	Overfeat fast	VGG-16	GoogLeNet v1	ResNet 50	ResNet 152
Top-5 error [%]	n/a	16.4	14.2	7.4	6.7	5.3	4.5
# CONV layers	2	5	5	13	57	53	155
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M	58M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G	11.3G
# FC layers	2	3	3	3	1	1	1
Weights	58k	58.6M	130M	124M	1M	2M	2M
MACs	58k	58.6M	130M	124M	1M	2M	2M
Total weights	60k → 61M → 146M → 138M			7M	25.5M	60M	
Total MACs	341k → 724M → 2.8G → 15.5G			1.43G	3.9G	11.3G	

FORWARD PATH ONLY. ADDITIONAL LAYERS (POOLING, BATCH NORMALIZATION, ...) AND ACTIVATION FUNCTION NOT INCLUDED.

RECAP: CONVOLUTION

$$\mathbf{O}[z][u][x][y] = \sum_{k=0}^{C-1} \sum_{i=0}^{S-1} \sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux + i][Uy + j] \cdot \mathbf{W}[u][k][i][j] + \mathbf{B}[u]$$

ofmap \mathbf{O} , ifmap \mathbf{I} , filters (weights) \mathbf{W} , and biases \mathbf{B}

$$E = (H - R + U)/U$$

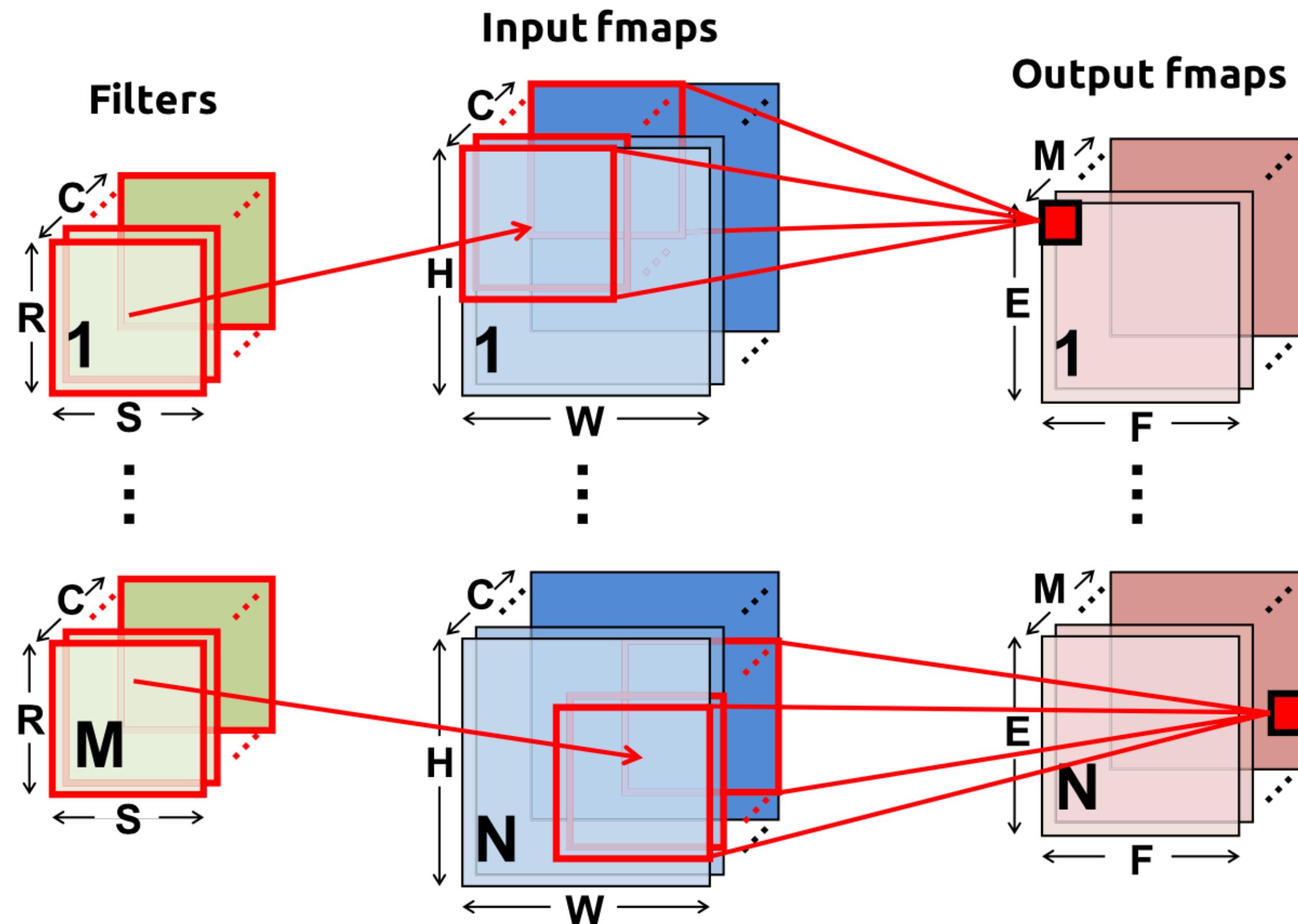
ofmap = output filter map (output activations)

$$F = (W - S + U)/U$$

ifmap = input filter map (input activations)

N	Batch size (3D fmmaps)	$0 \leq z \leq N$
M	number of 3D filters / number of ofmaps	$0 \leq u \leq M$
C	number of ifmap/filter channels	
H / W	ifmap plane height/width	
R / S	filter plane height/width	
E / F	ofmap plane height/width	$0 \leq x \leq E, 0 \leq y \leq F$
U	stride	

CONVOLUTION OPERATION



Output filter height

$$E = \frac{H - R + U + 2P}{U}$$

For padding P

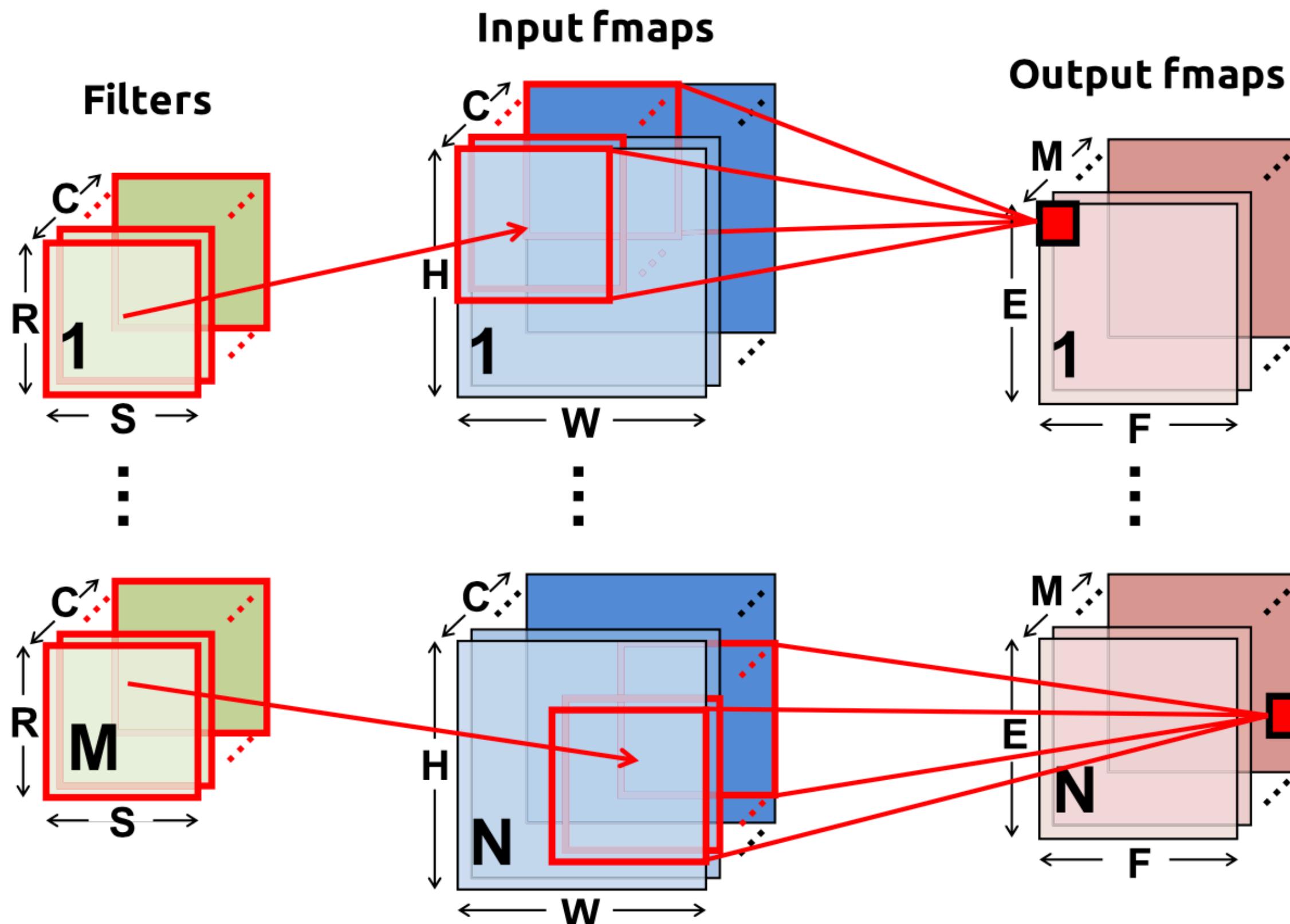
Output filter width

$$F = \frac{W - S + U + 2P}{U}$$

For padding P

Output filter depth (channels)
equals numbers of filters M

CONVOLUTION: NUMBER OF PARAMETERS



Convolution

$$\text{Weights } W_c = RSCM$$

$$\text{Biases } B_c = M$$

$$\text{Total } P_c = W_c + B_c$$

Compare to FC

Number of neurons O

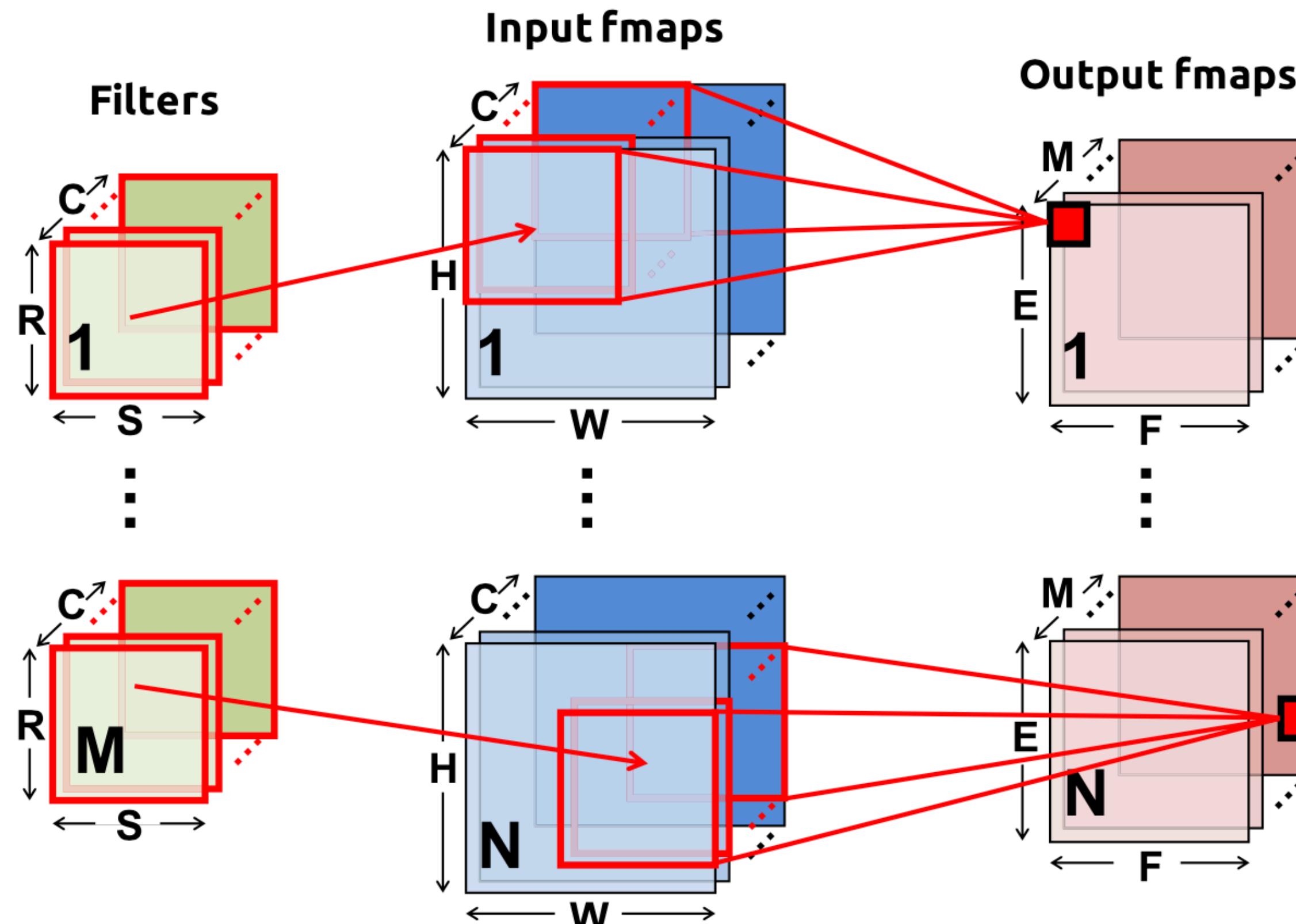
$$\text{Weights } W_f = WHC \cdot O$$

$$\text{Biases } B_f = O$$

$$\text{Total } P_f = W_f + B_f$$

Output: $W = O, H = C = 1$

CONVOLUTION: NUMBER OF MACS



MAC = multiple-accumulate
as used in the dot product

$$c = \sum_n ab \Rightarrow n \text{ MACs}$$

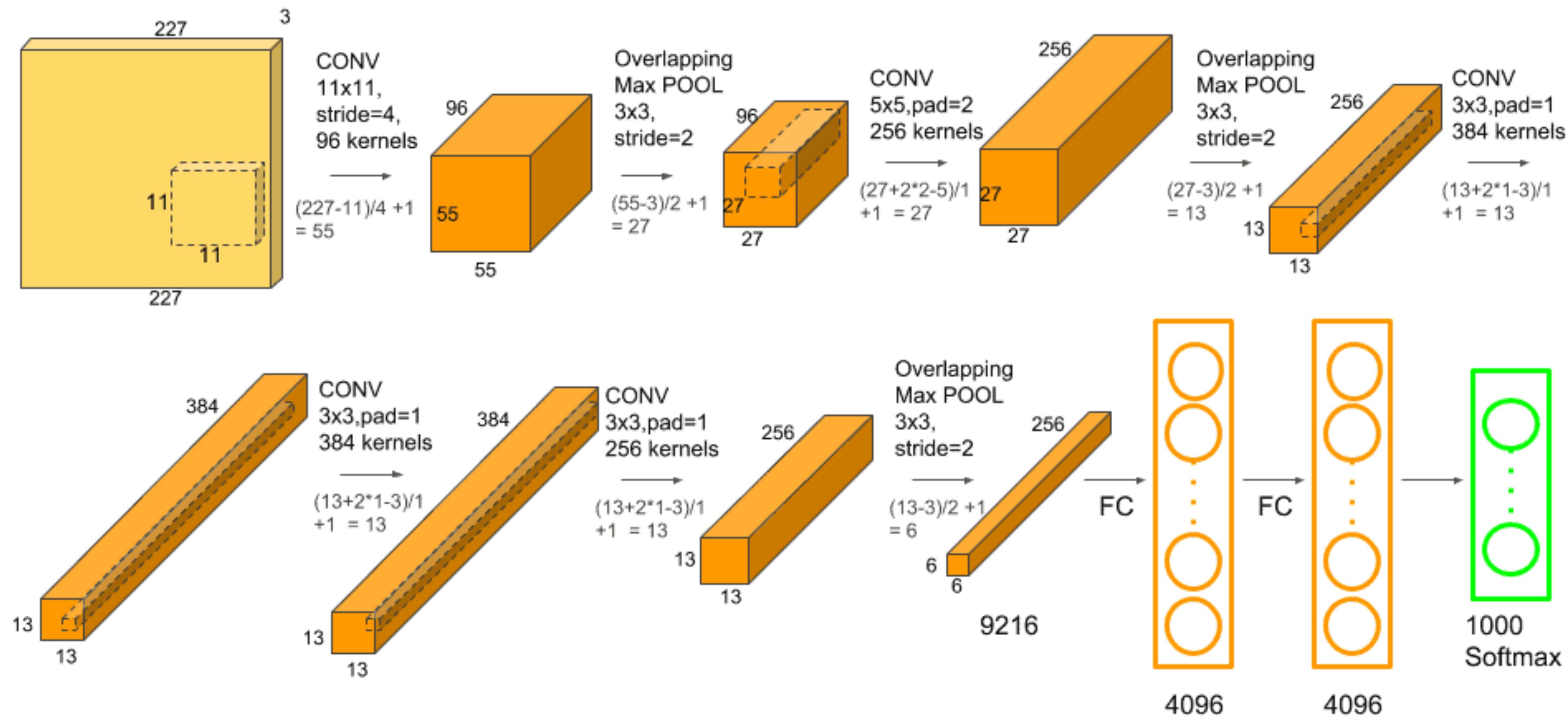
Convolution

$$\text{MAC}_c = (EF \cdot RSC) \cdot M$$

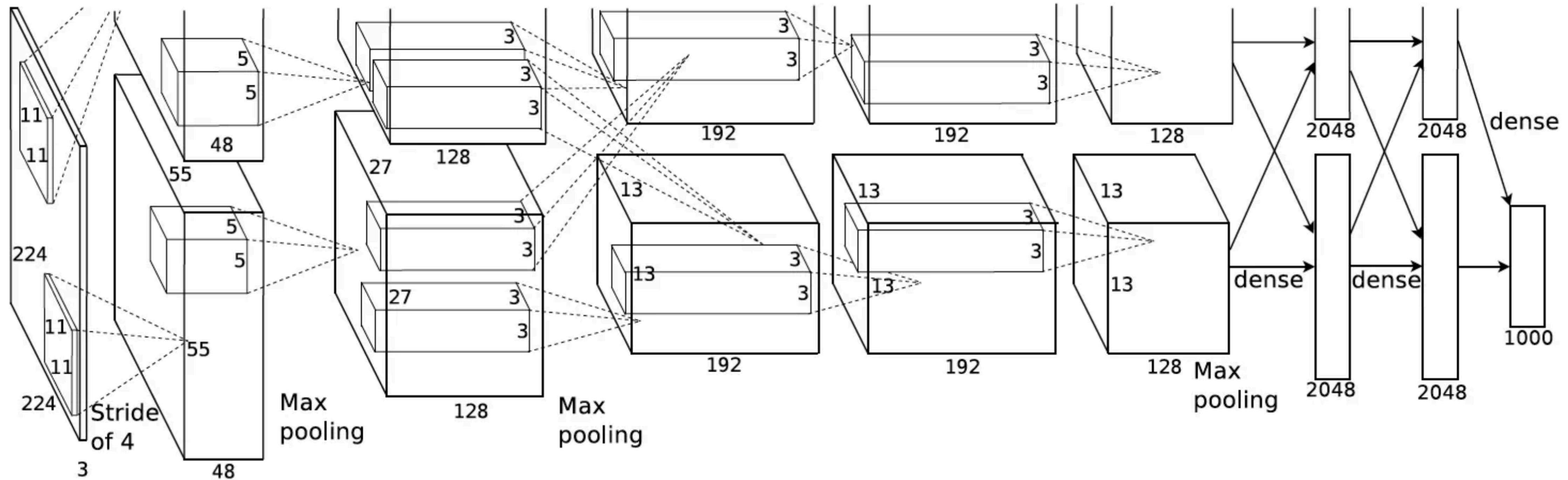
Compare to FC

$$\text{MAC}_f = W_f = WHC \cdot O$$

ALEXNET



ALEXNET



POOLING

Pooling layers reduce the size of the output map and create invariance to small transformations

Pool size = kernel size k , and stride s (usually $s = k$)

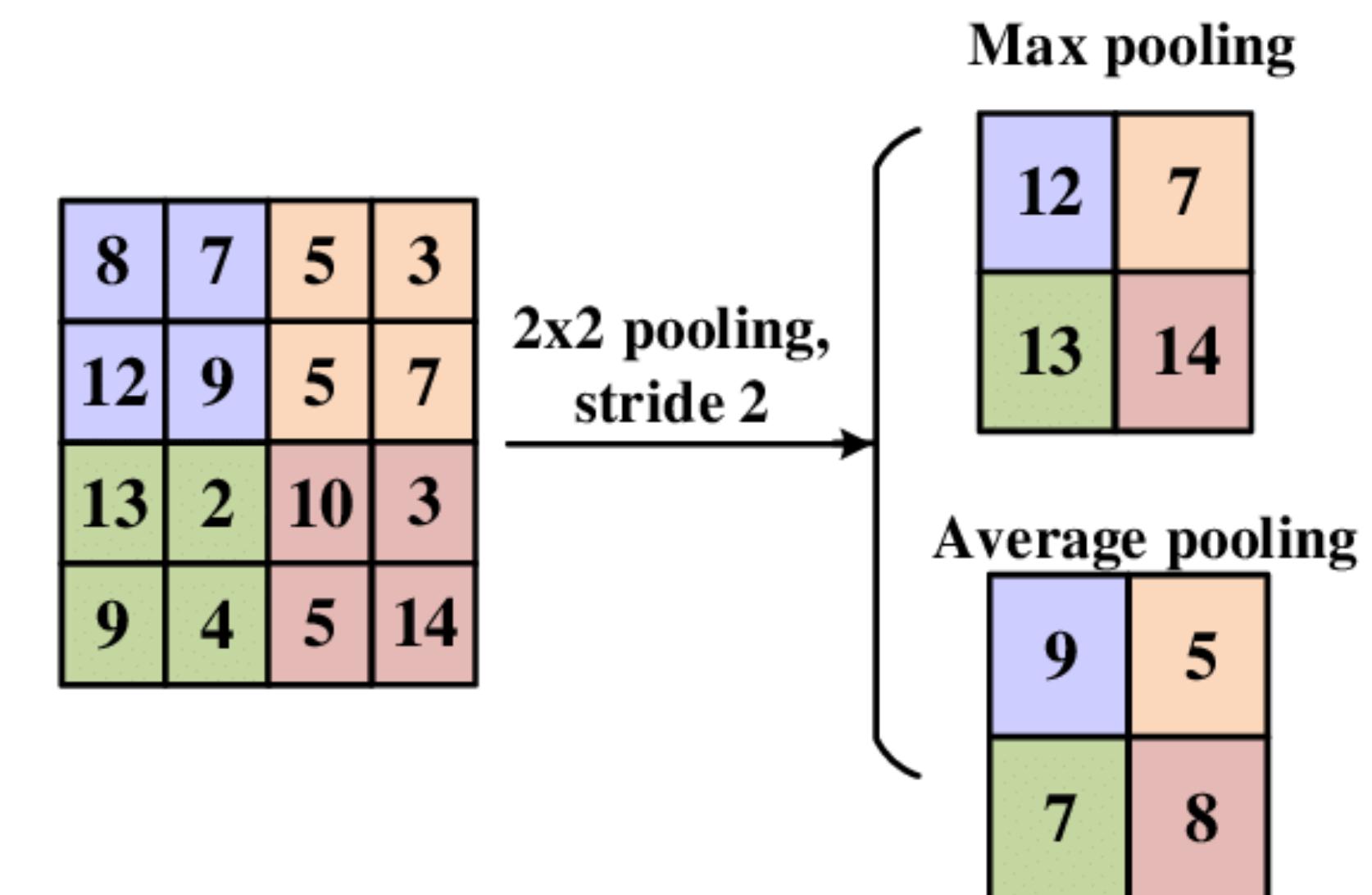
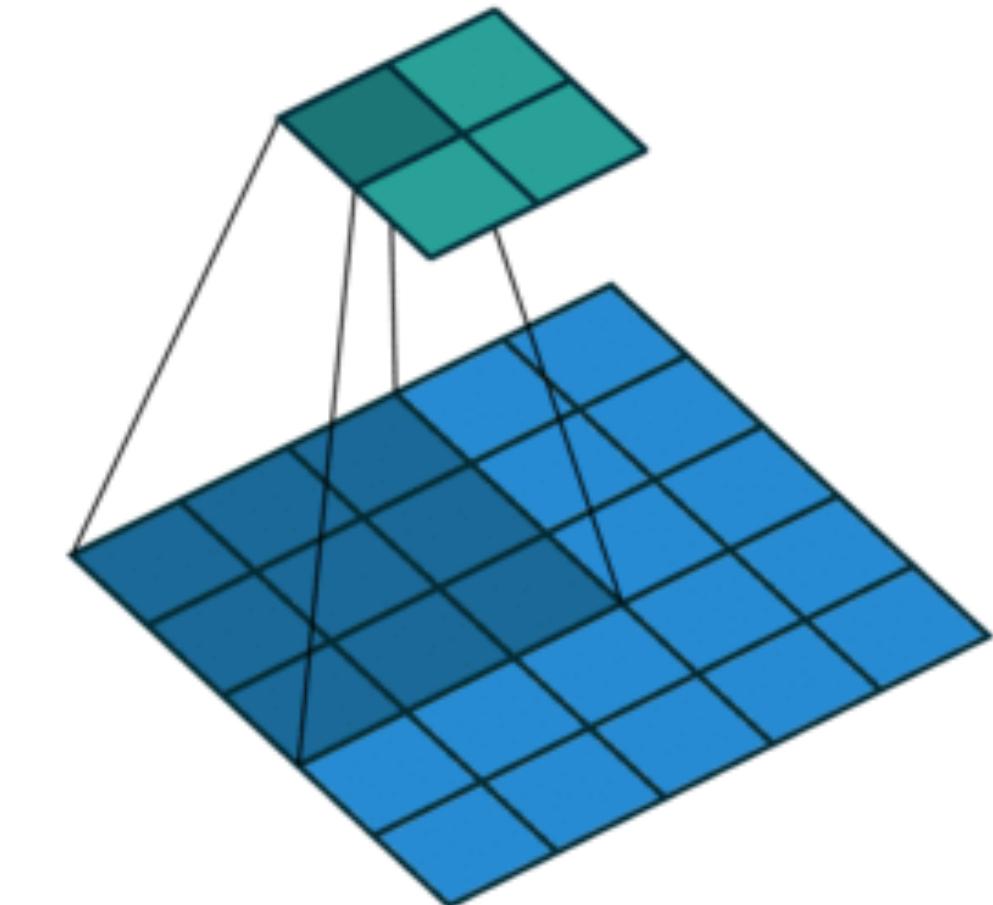
Operates over each channel independently

No learnable parameters

For max pooling: $y_i = \max_j(z_j)$

There are other variants, such as average pooling for smoothening

Remember backprop: a max gate is a gradient router (downstream gradients depend on comparison of inputs)



POOLING

1. Pooling reduces the size of the output, thus the size of the input map for the next layer

Reduces computational complexity by reducing W, H

$$\text{MAC}_c = (EF \cdot RSC) \cdot M, \text{ but } E = \frac{H - R + U + 2P}{U} \text{ and } F = \frac{W - S + U + 2P}{U}$$

depend on H and W , respectively

Space complexity (number of parameters) unchanged

2. Invariance sought as image classification is usually translation-invariant

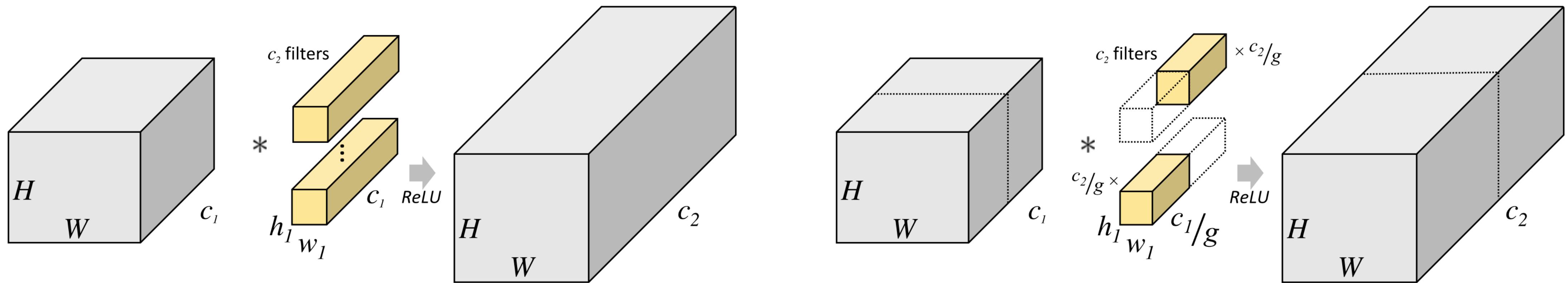
Convolutional layers: equivariant (input translations are preserved)

Pooling layers provide invariance to small translations

Too much invariance can be harmful

Pooling reduces width and height of the output, but not the number of channels

GROUPED CONVOLUTIONS



A grouped convolution is convolved separately with $M' = M/g$ for g groups

The output is the concatenation of all groups output along the channel axis

Input is also split along the channel axis

Easier to parallelize (AlexNet) and more efficient to compute

$$\text{MAC}_c = (EF \cdot RSC) \cdot M \Rightarrow \text{MAC}_{cg} = g \cdot (EF \cdot RS \frac{C}{g}) \cdot \frac{M}{g} = \frac{\text{MAC}_c}{g}$$

$$\text{Weights } W_c = RSCM \Rightarrow W_{cg} = g \cdot RS \frac{C}{g} \frac{M}{g} = \frac{W_c}{g}, \text{ Biases } B_c = M \Rightarrow B_{cg} = g \cdot \frac{M}{g} = B_c$$

PUTTING IT ALL TOGETHER (FOR ALEXNET)

	Input/ output	Input/ output	Input/ output	Filter	Filter / pooling	Filter / pooling	Filter / pooling	Filter / pooling	FC	MACs	Params	Output act.
	height	width	channels	count	height	width	stride	padding	groups	neurons	w+b	
	H / E	W / F	C	M	R	S	U	P	G	O		
Input	227	227	3									
CONV-1	55	55	96	96	11	11	4	0	1	105,415,200	34,944	290,400
POOL-1	27	27	96		3	3	2					
CONV-2	27	27	96	256	5	5	1	2	2	223,948,800	307,456	69,984
POOL-2	13	13	256		3	3	2					
CONV-3	13	13	256	384	3	3	1	1	2	74,760,192	442,752	43,264
CONV-4	13	13	384	384	3	3	1	1	2	74,760,192	442,752	64,896
CONV-5	13	13	384	256	3	3	1	1	2	74,760,192	442,624	64,896
POOL-3	6	6	256		3	3	2					
FC-1	9216	1	1						4096	37,748,736	37,752,832	9,216
FC-2	4096	1	1						4096	16,777,216	16,781,312	4,096
FC-3	4096	1	1						1000	4,096,000	4,097,000	4,096
									Sum CONVs	553,644,576	1,670,528	533,440
									Sum FCs	58,621,952	58,631,144	17,408
									Total	612,266,528	60,301,672	550,848

INCEPTION ARCHITECTURE

RECAP: ANN TRENDS

	LeNet 5	AlexNet	Overfeat fast	VGG-16	GoogLeNet v1	ResNet 50	ResNet 152
Top-5 error [%]	n/a	16.4	14.2	7.4	6.7	5.3	4.5
# CONV layers	2	5	5	13	57	53	155
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M	58M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G	11.3G
# FC layers	2	3	3	3	1	1	1
Weights	58k	58.6M	130M	124M	1M	2M	2M
MACs	58k	58.6M	130M	124M	1M	2M	2M
Total weights	60k → 61M → 146M → 138M			7M	25.5M	60M	
Total MACs	341k → 724M → 2.8G → 15.5G			1.43G	3.9G	11.3G	

FORWARD PATH ONLY. ADDITIONAL LAYERS (POOLING, BATCH NORMALIZATION, ...) AND ACTIVATION FUNCTION NOT INCLUDED.

INCEPTION

Main motivation: reduce number of parameters to enable deployment on mobile devices

Google's datacenter allow to train virtually any model independent of size

Similarly, lots of units (space complexity for backprop) is ok

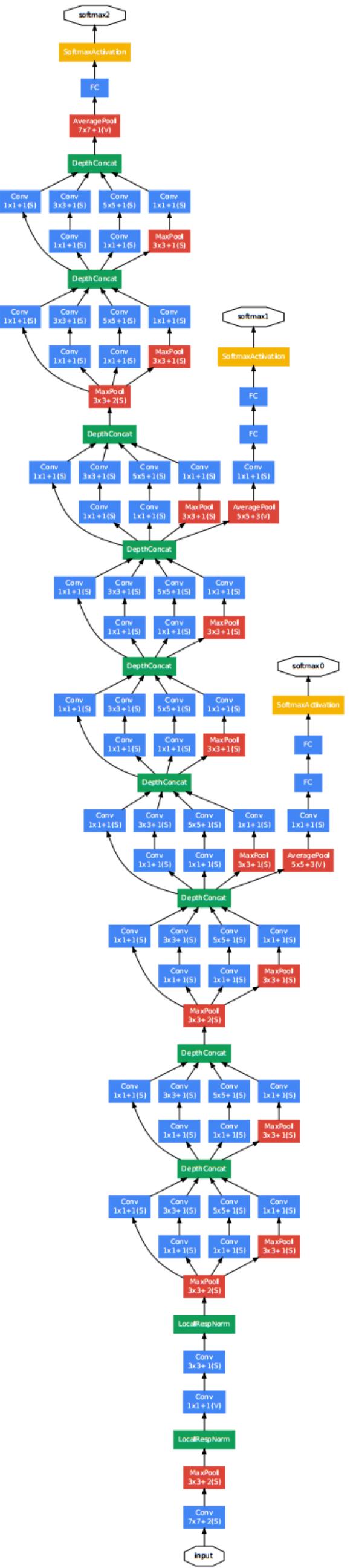
1. Convolutions only, no FC layers any more

Going deeper, 22 layers total

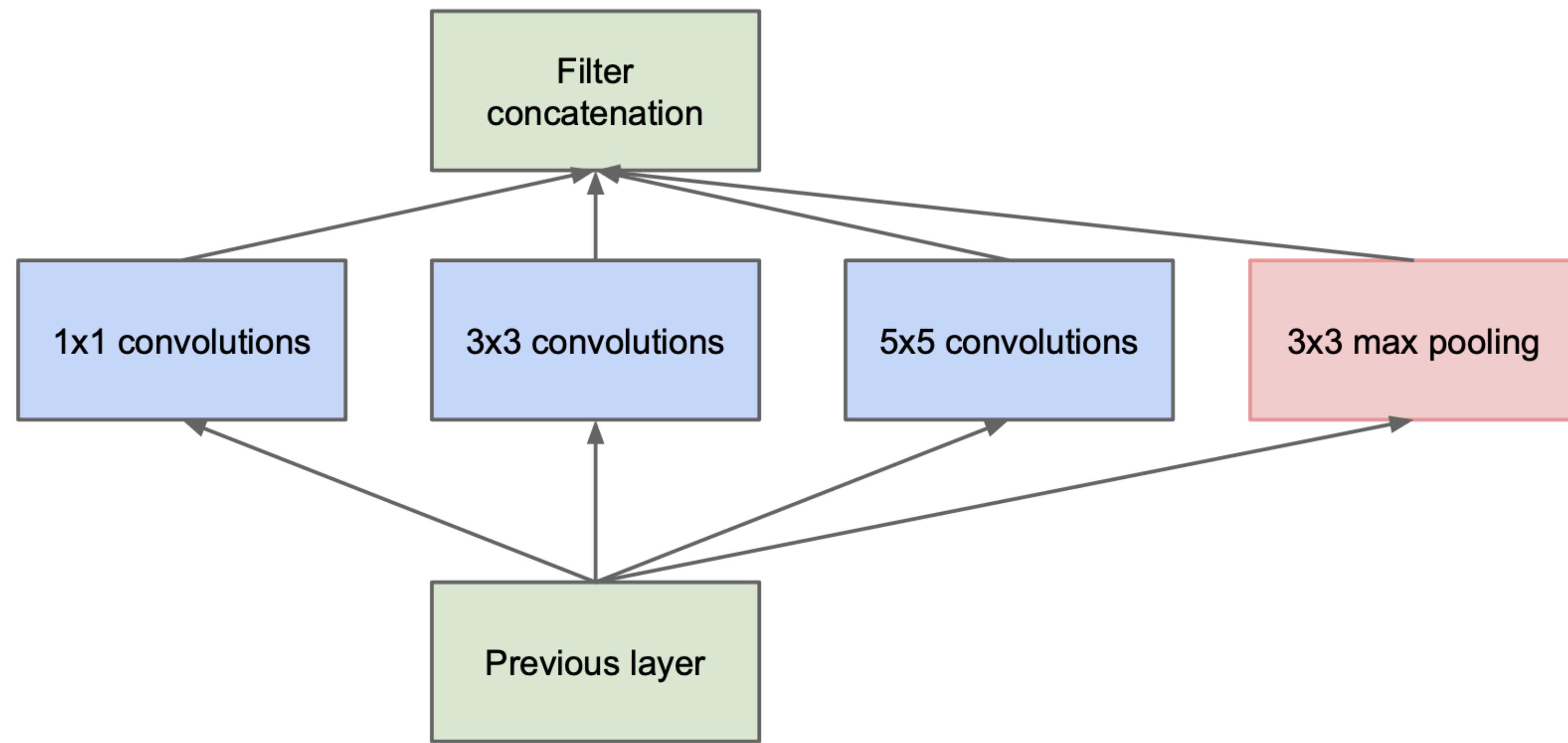
2. Replace large convolutions by multiple small convolutions

Two 3x3 filters have less parameters than one 5x5 filter, but cover a similar field

=> Layer-within-layer architecture (Inception movie: dreams-within-dreams)



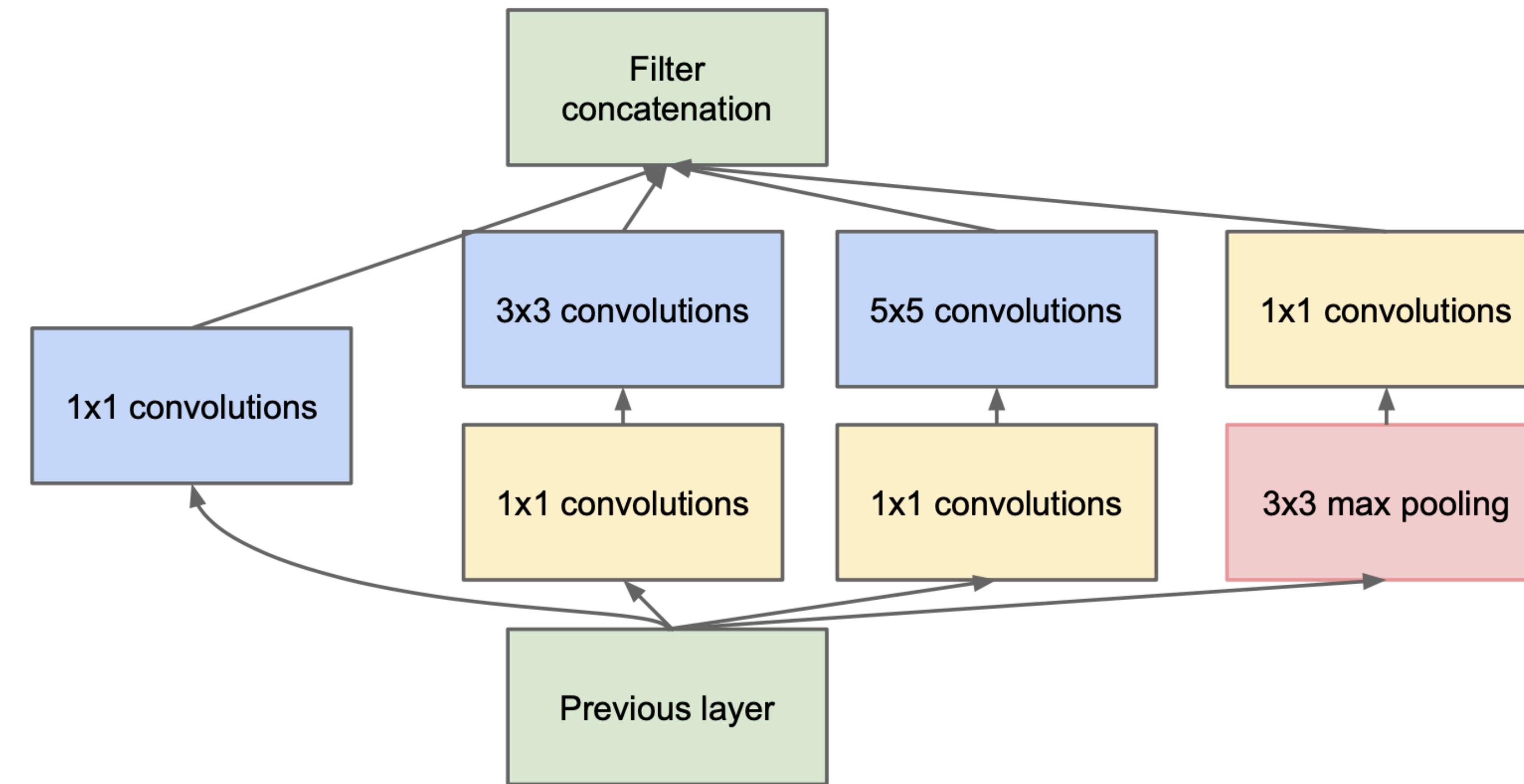
INCEPTION MODULE, NÄIVE VERSION



Key idea: use different convolutions in parallel, forward all results to the next layer which decides on importance

Max pooling included as “historically” good for complexity reduction

INCEPTION MODULE WITH DIMENSIONALITY REDUCTION



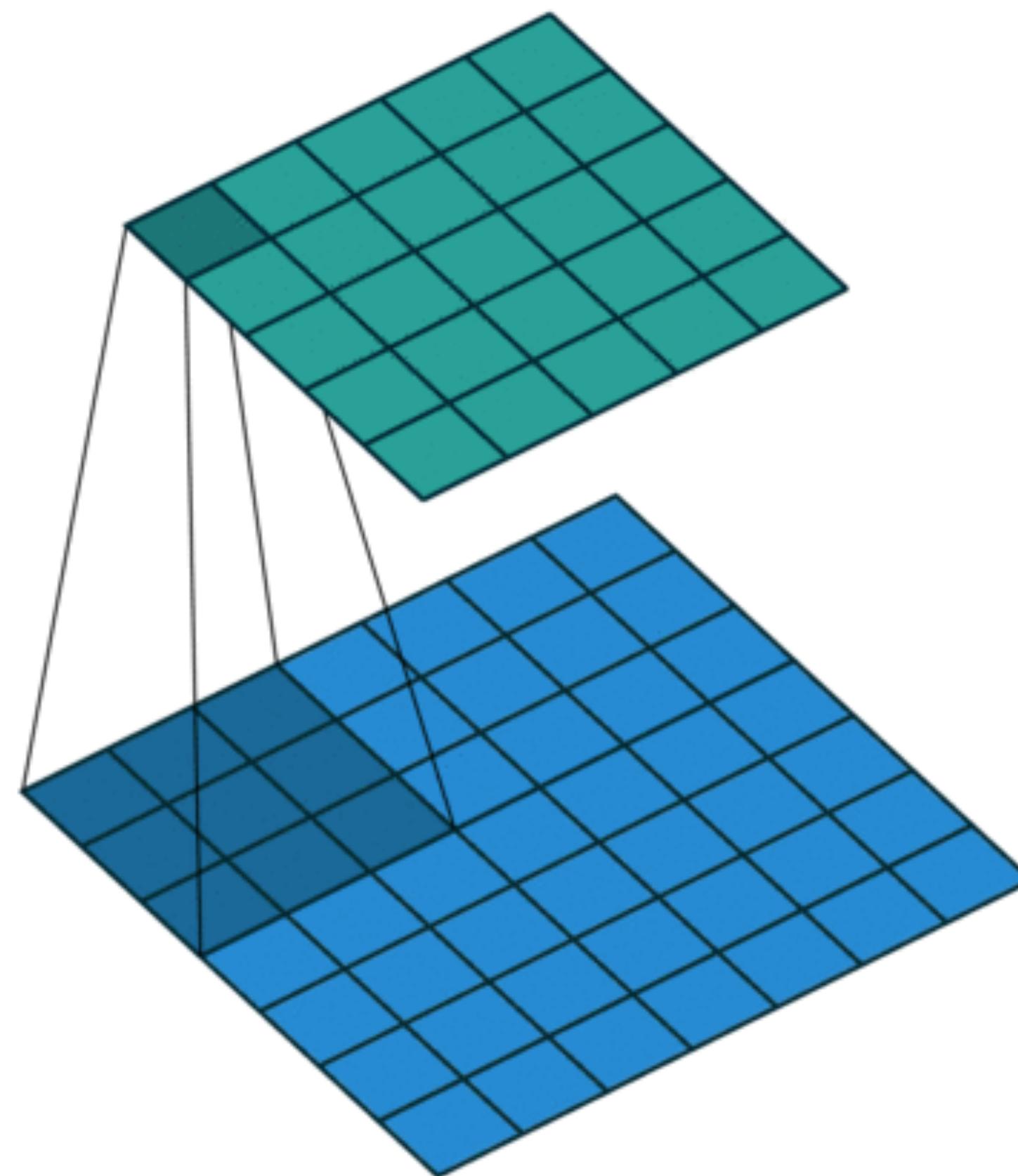
1x1 convolutions inspired by “Network-in-Network” architecture

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. <https://arxiv.org/abs/1312.4400>

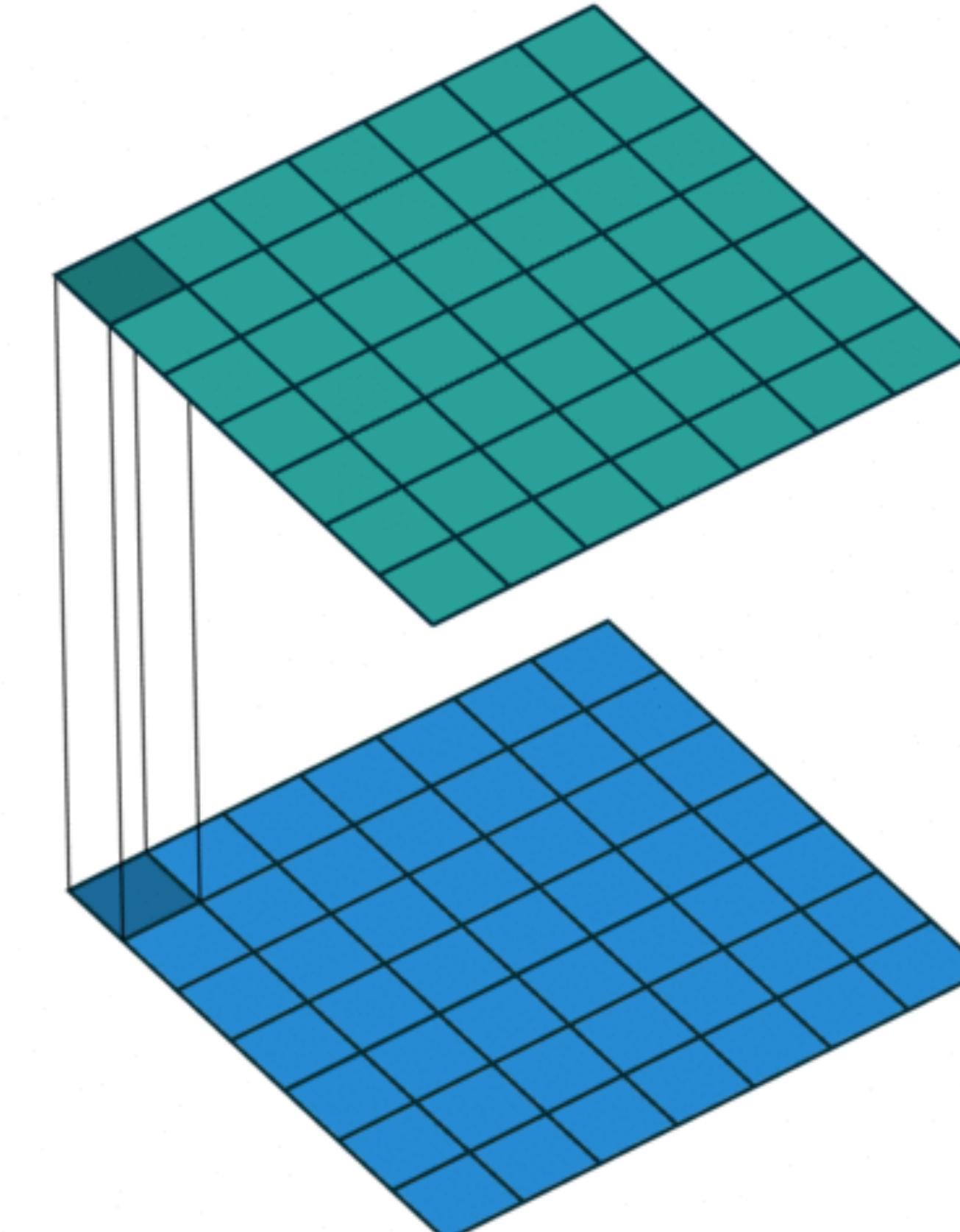
Why though?

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich; Going Deeper With Convolutions, CVPR 2015

CONVOLUTIONS: 3X3 VS 1X1 - USELESS?



3x3



1x1

1X1 CONVOLUTIONS

The number of filters (input depth) used in convolutional layers often increases with the depth of the network

=> increase in number of feature maps

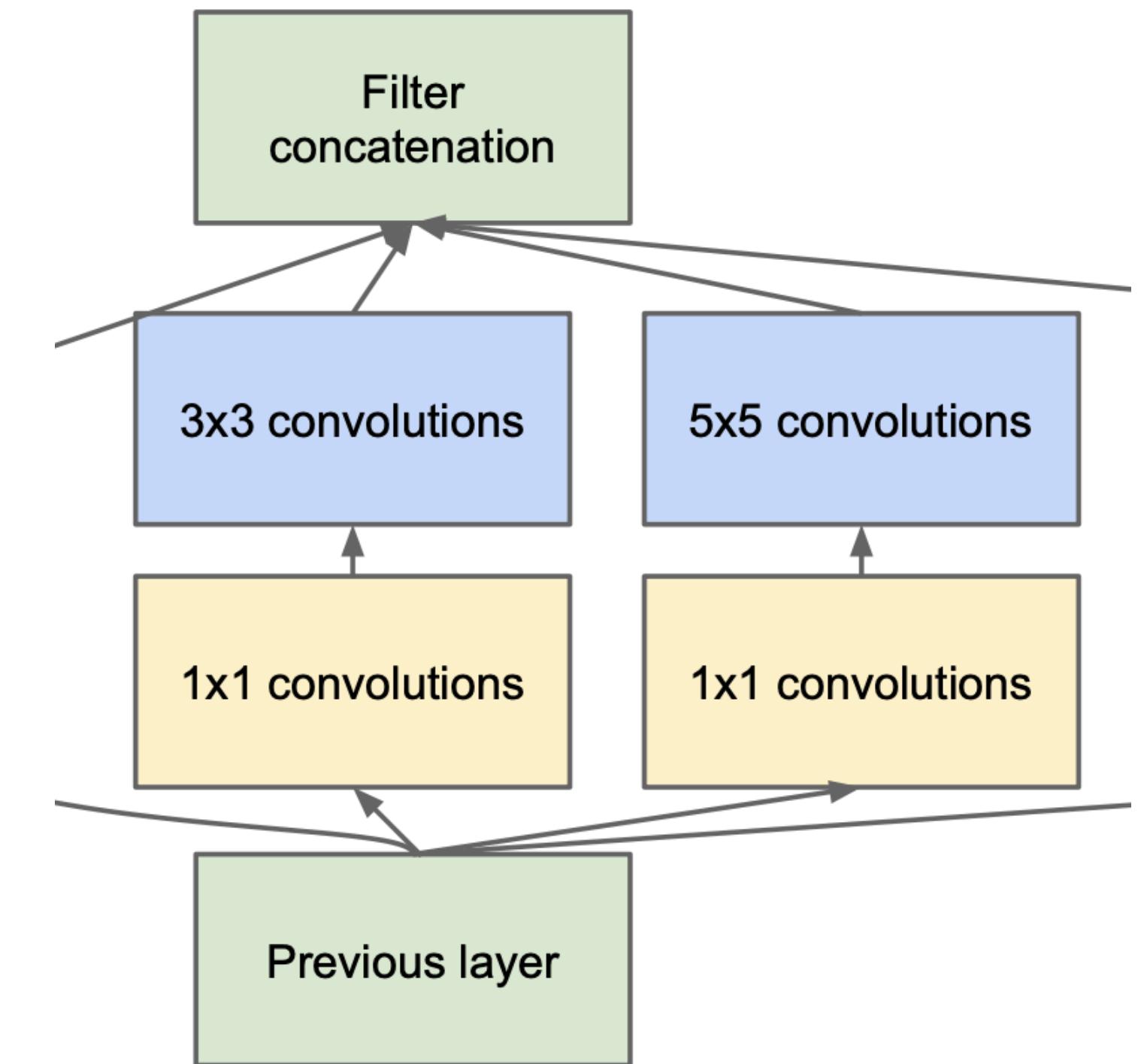
Concatenating the output of feature maps of multiple convolutions also increases the input depth

=> Larger time and space complexity

Pooling layers only reduce the width and height, but not the depth of an input

1x1: alternative to downsample feature maps

See a 1×1 convolution rather as $1 \times 1 \times C$, with the number of filters M being the new C for the next layer



1x1 convolutions reduce MACs and parameters, but increase activation space

1X1 CONVOLUTIONS FOR DIMENSIONALITY REDUCTION

Weights $W_c = RSCM$ and MACs $MAC_c = (EF \cdot RSC) \cdot M$

$$E = \frac{H - R + U + 2P}{U} \text{ and } F = \frac{W - S + U + 2P}{U}$$

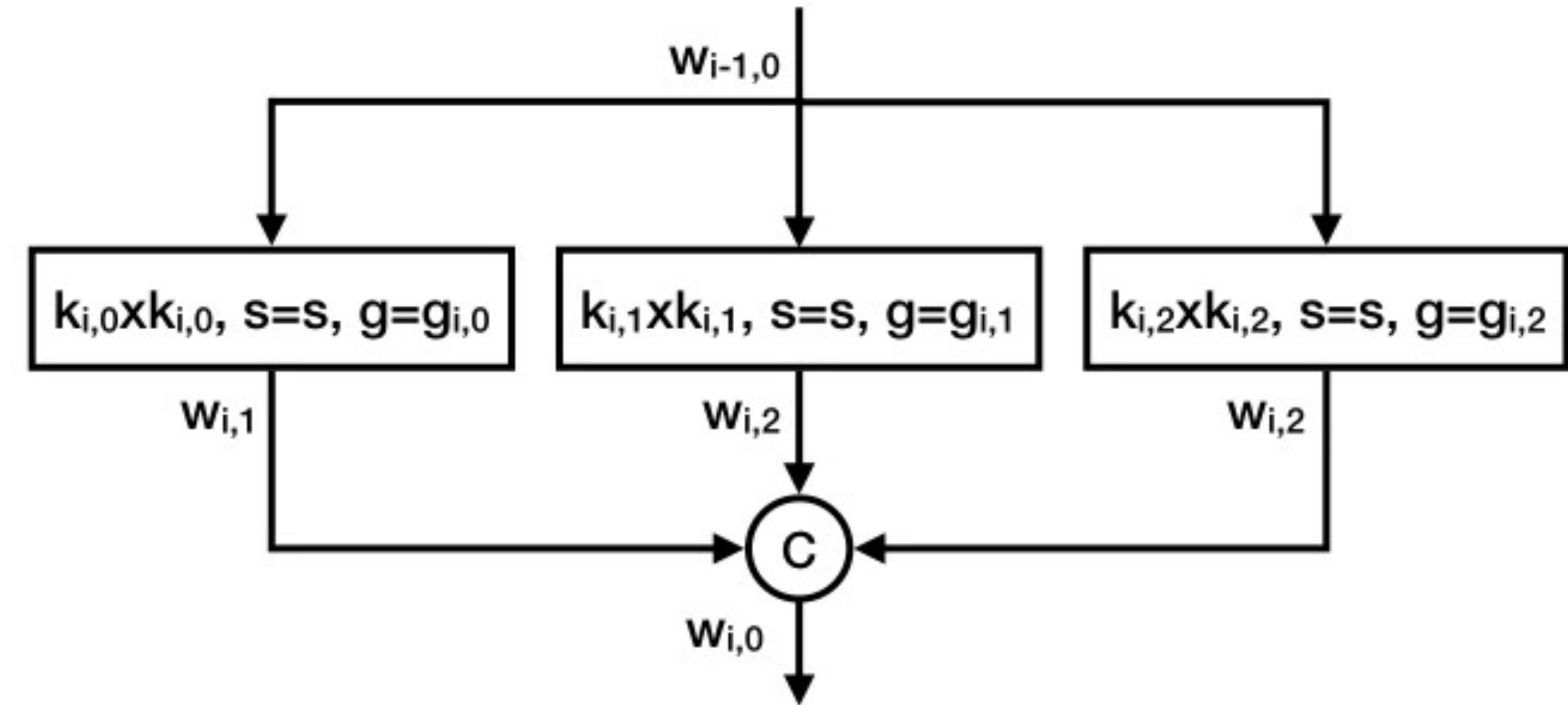
Consider an input of $WxHxC = 28x28x192$

Max pooling before inception block 3a, padding set to preserve dimensions

	H	W	M	R	S	C	E	F	W_c	MAC_c
Plain 5x5x32	28	28	32	5	5	192	28	28	153,600	120,422,400
Initial 1x1x16	28	28	16	1	1	192	28	28	3,072	2,408,448
New 5x5x32	28	28	32	5	5	16	28	28	12,800	10,035,200

$M = 16$ is a trade-off made in the paper

INCEPTION BLOCK

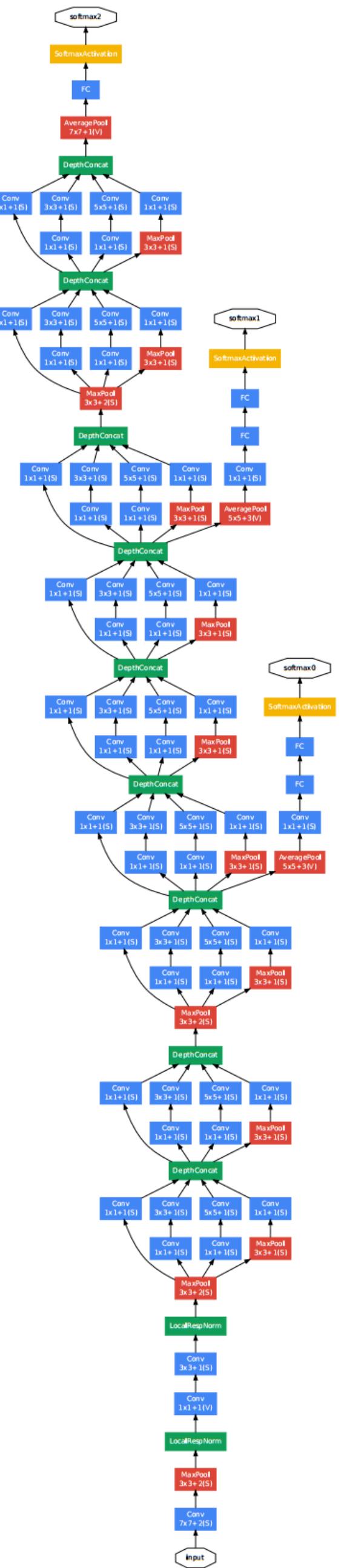


Key idea: use different convolutions in parallel, forward all results to the next layer which decides on importance

Add more non-linearity by having ReLU immediately after every 1x1 convolution

1x1 convs usually not used in the first layer to preserve receptive field

Relatively shallow to avoid problems with vanishing gradients



RESIDUAL ARCHITECTURE

ARCHITECTURE DESIGN

Architecture	Layers	Parameters	MACs	ImageNet Error
AlexNet	7	61M	724M	16.4
VGG	16	138M	2.8G	7.4
Inception	69	24M	5.7G	5.6
ResNet	152	60M	11G	4.5

Deeper is more efficient (parameter or MACs required to achieve a certain target accuracy) than wide (empirically determined)

RECAP: ISSUES WITH GRADIENT DESCENT IN NEURAL NETWORKS

$$\mathbf{y}(\mathbf{W}, \mathbf{x}_0) = \mathbf{x}_L = f(\mathbf{W}_L \oplus f(\mathbf{W}_{L-1} \oplus (\dots \oplus f(\mathbf{W}_1 \oplus \mathbf{x}_0) \dots)))$$

$$\frac{\partial l}{\partial \mathbf{x}_1} = \underbrace{\frac{\partial l}{\partial \mathbf{x}_L} \cdot \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \cdot \dots \cdot \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \cdot \dots \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \cdot \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}}_{\text{Downstream gradients}}$$

Observation 1: Downstream gradients greatly depend on value of activations \mathbf{x}_i and activation function f

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \frac{\partial f(\mathbf{W}_i \oplus \mathbf{x}_{i-1})}{\partial \mathbf{x}_{i-1}}$$

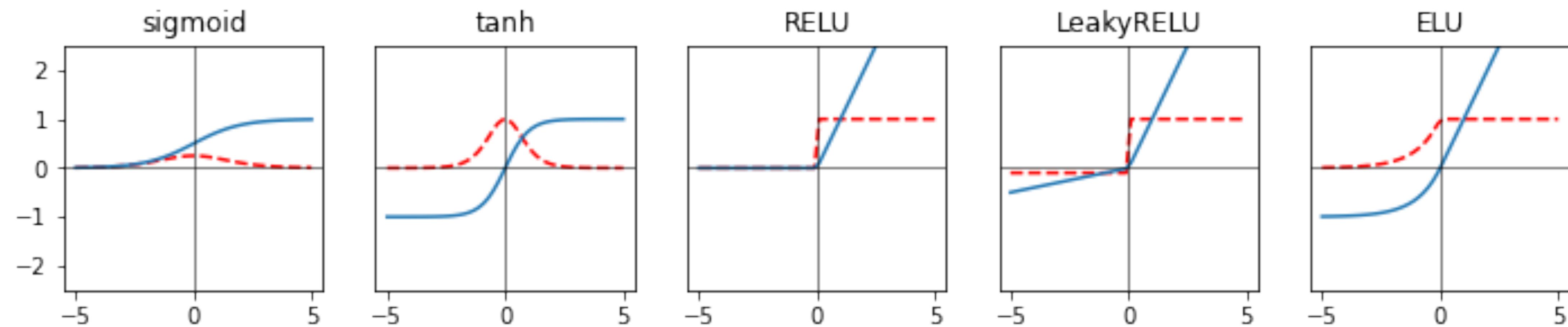
Multiplication gate can diminish gradients ($\frac{\partial \text{mul}(w_i, x_{i-1})}{\partial w_i} = x_{i-1}$)

Given comparable effort (epochs), deeper networks have larger training and test error

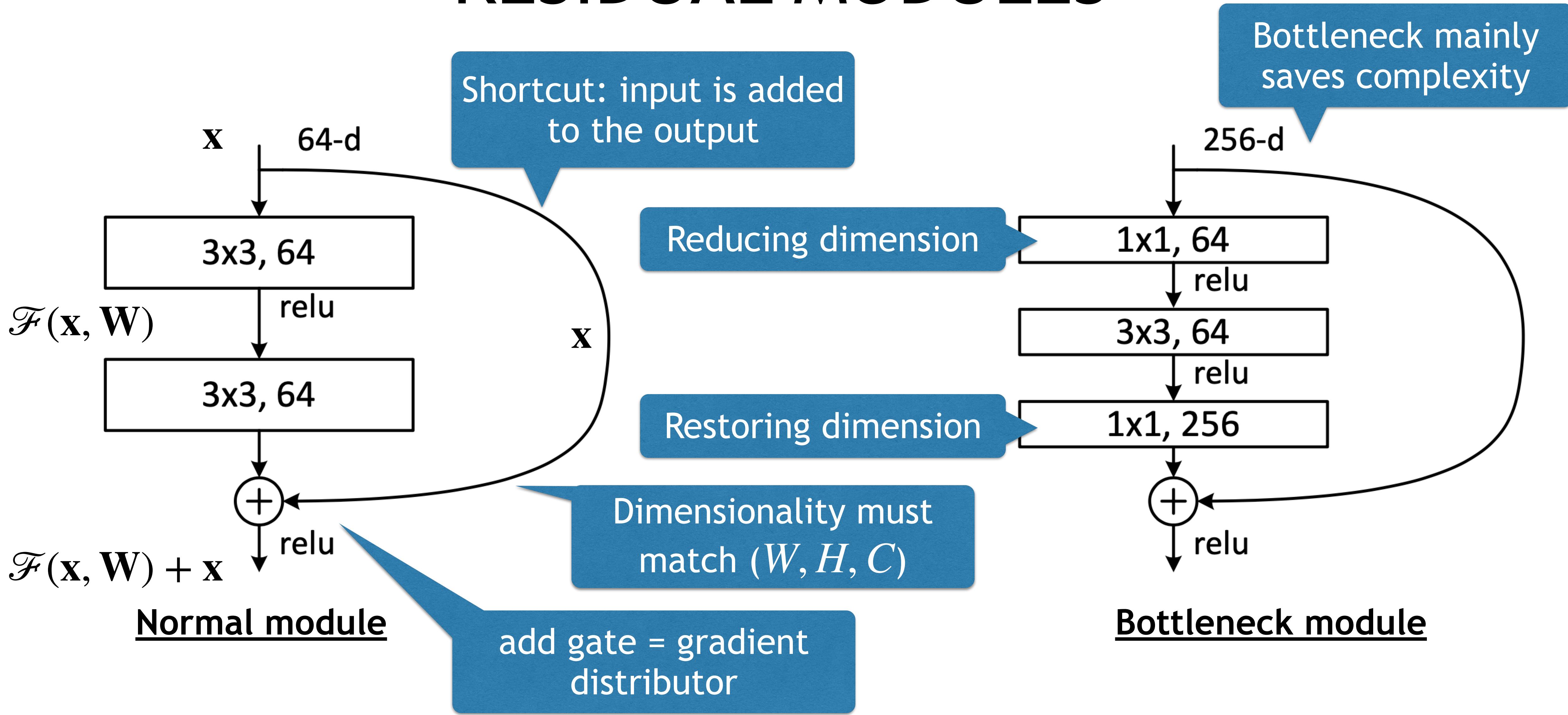
RECAP: VANISHING/EXPLODING GRADIENTS

Activation functions have a strong impact on the value of gradients

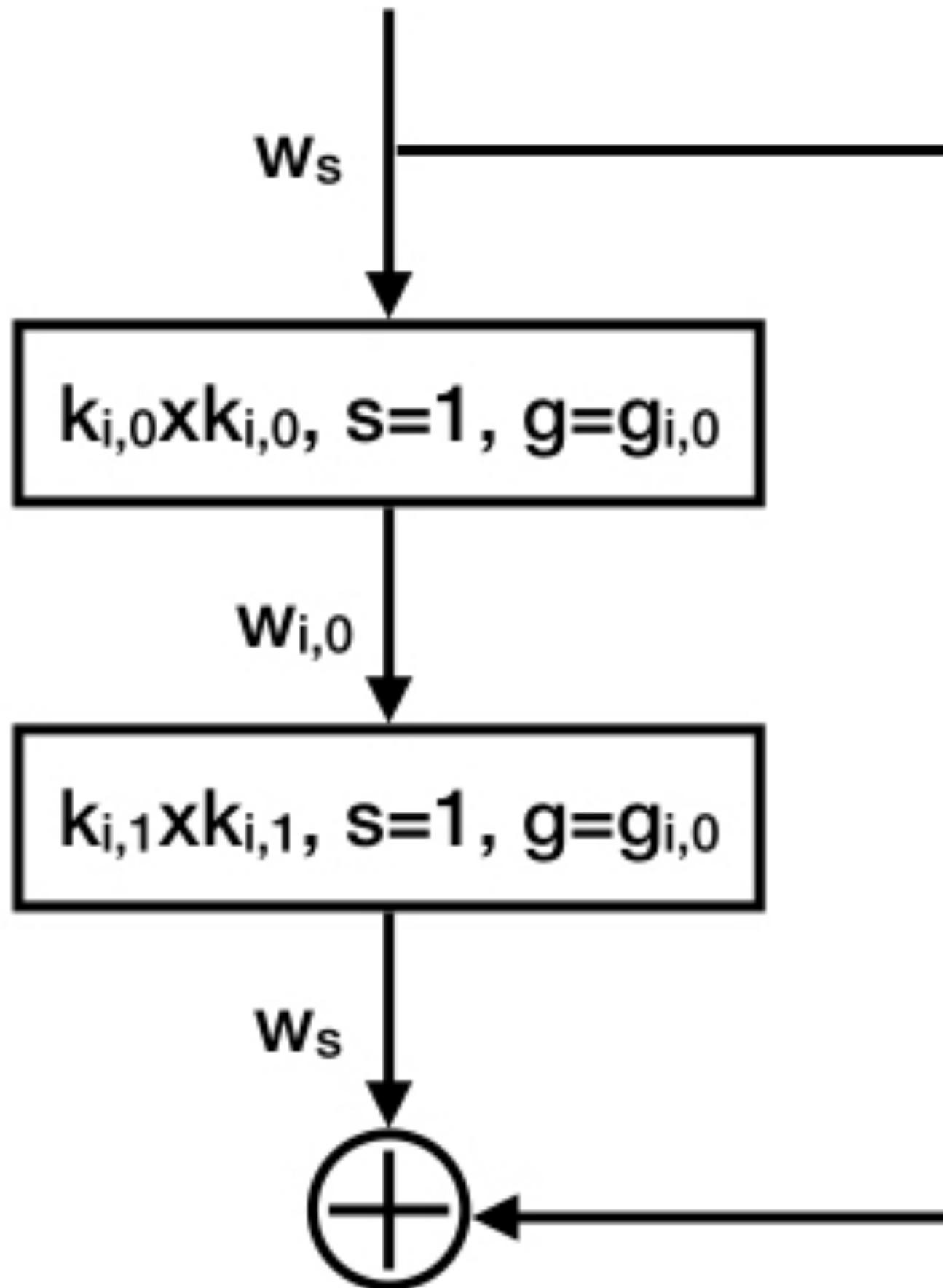
Non-squashing activation function with large derivative => RELU and variants



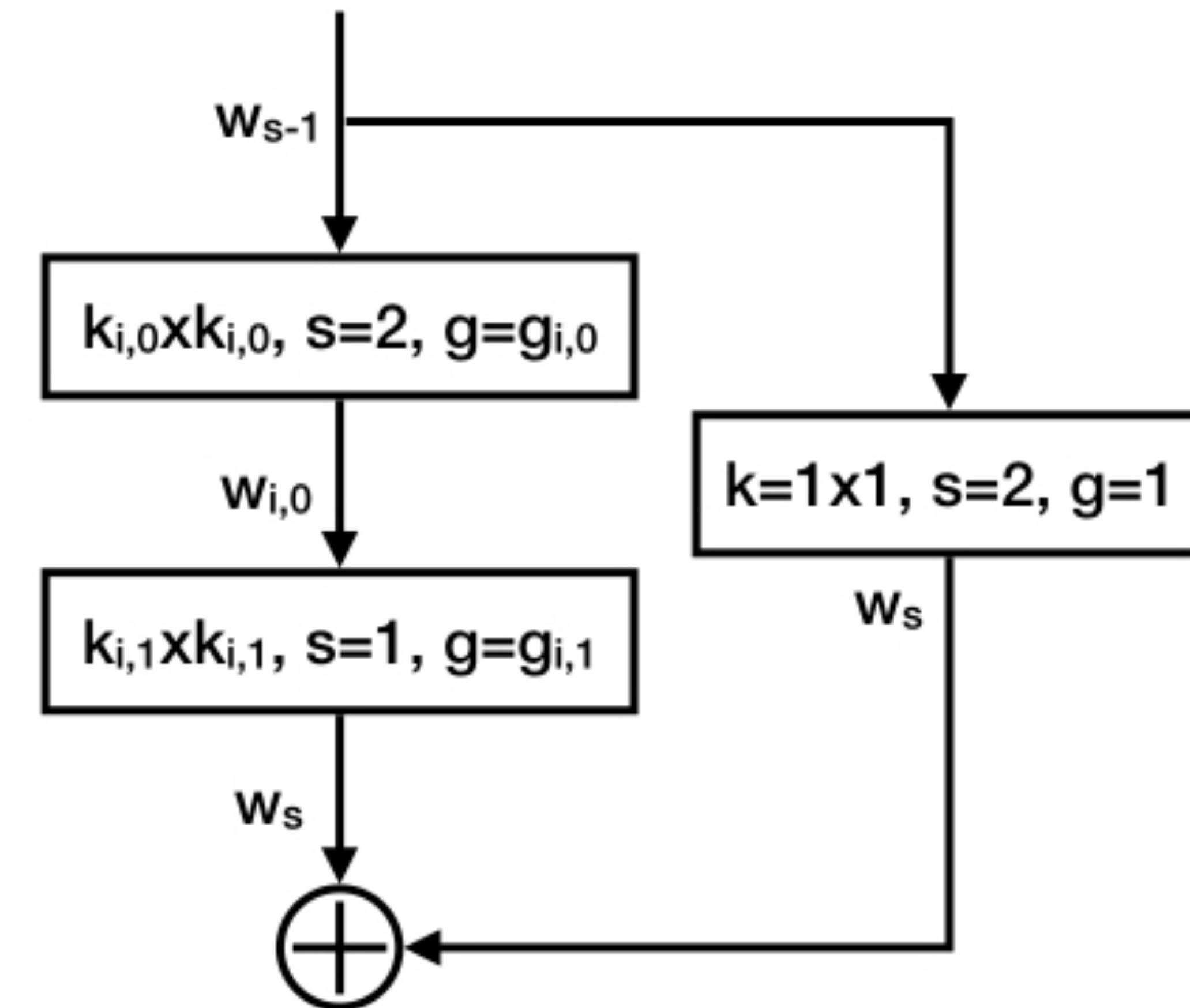
RESIDUAL MODULES



RESIDUAL AND DOWNSAMPLING



Without downsampling



With downsampling

Enables $w_s ! = w_{s-1}$

RESIDUAL NETWORKS

Residual Unit

$$x_{l+1} = \mathcal{F}(\mathbf{x}_l, \mathbf{W}_l) + \mathbf{x}_l$$

Residual Network

$$x_L = \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathbf{W}_i) + \mathbf{x}_l$$

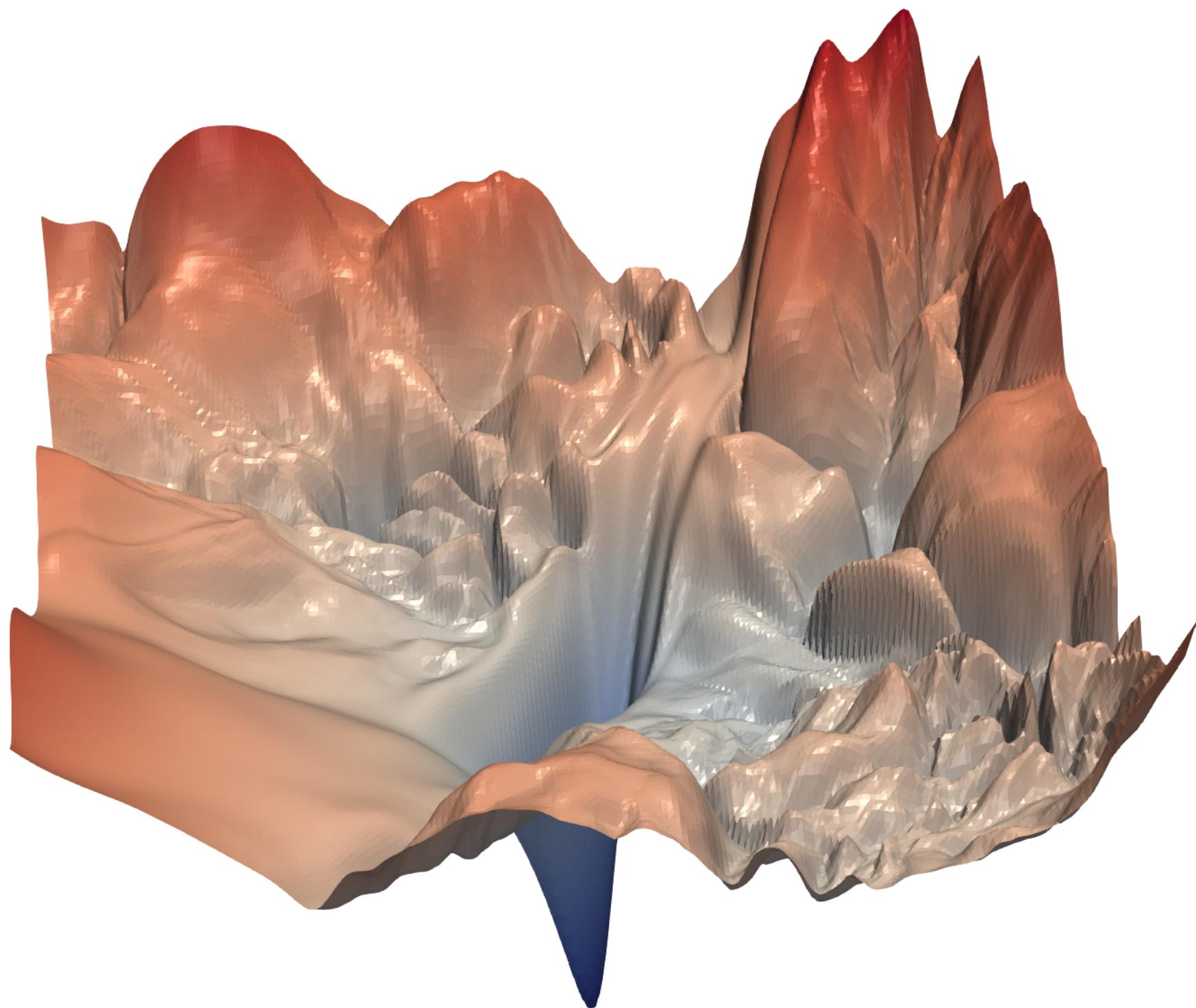
For any deeper unit L and
shallower unit l

Backward Path

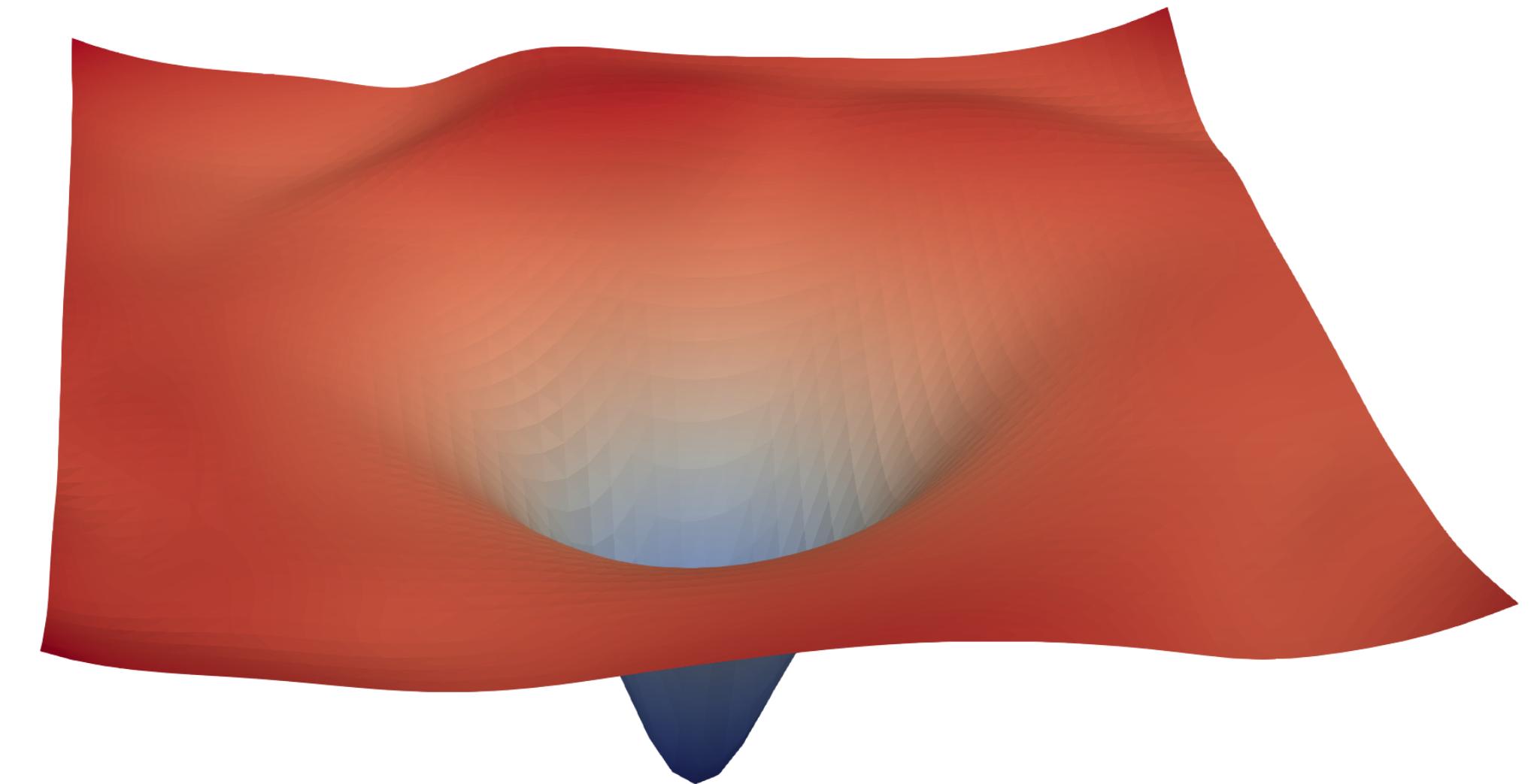
$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L} \frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L} \left(1 + \frac{\partial x_L}{\partial x_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathbf{W}_i) \right)$$

RESIDUAL NETWORKS

Without connections



With connections



DENSENET

Normal layer: $x_l = \mathcal{F}_l(x_{l-1})$

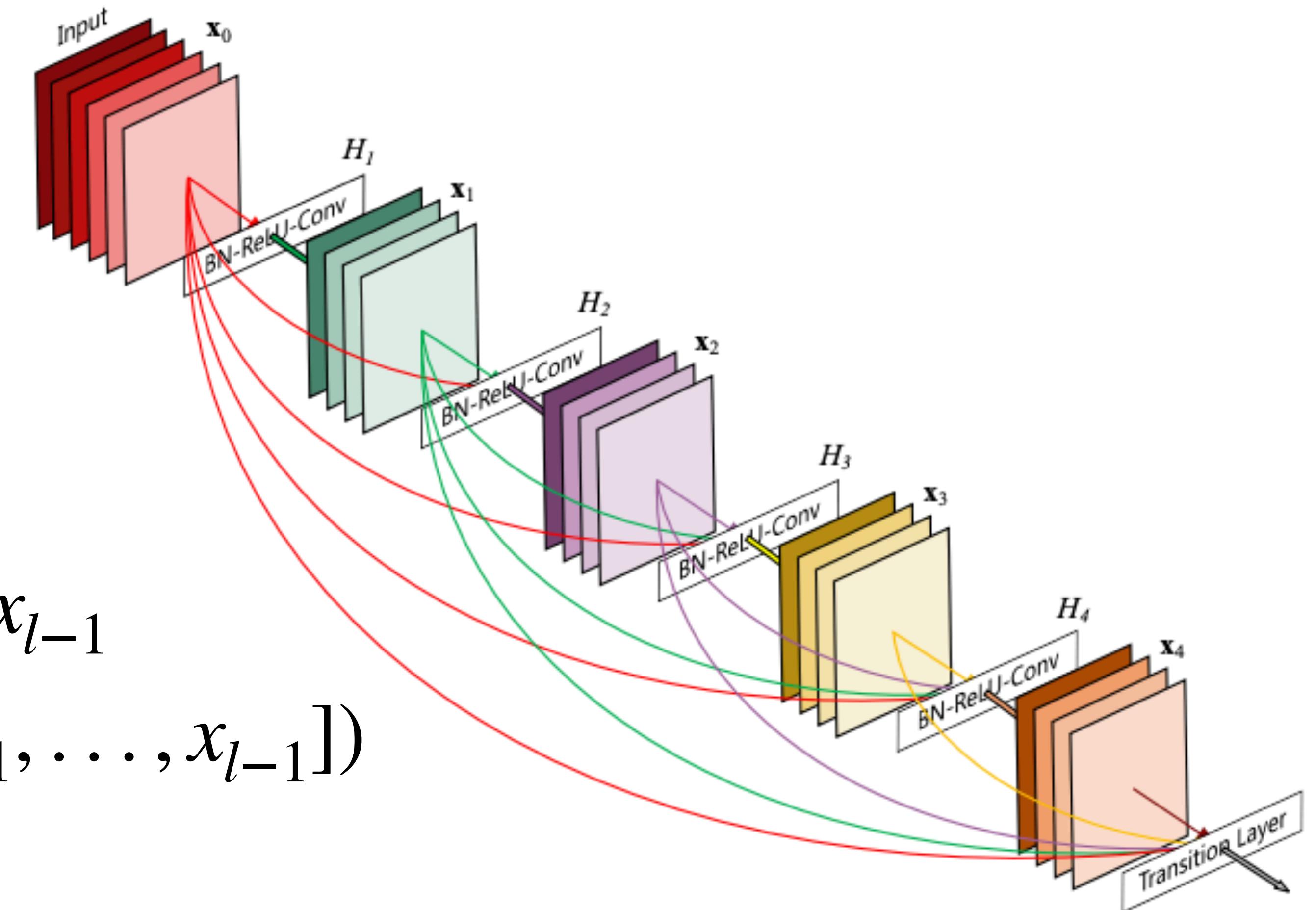
ResNet layer: $x_l = \mathcal{F}_l(x_{l-1}) + x_{l-1}$

DenseNet layer: $x_l = \mathcal{F}_l([x_0, x_1, \dots, x_{l-1}])$

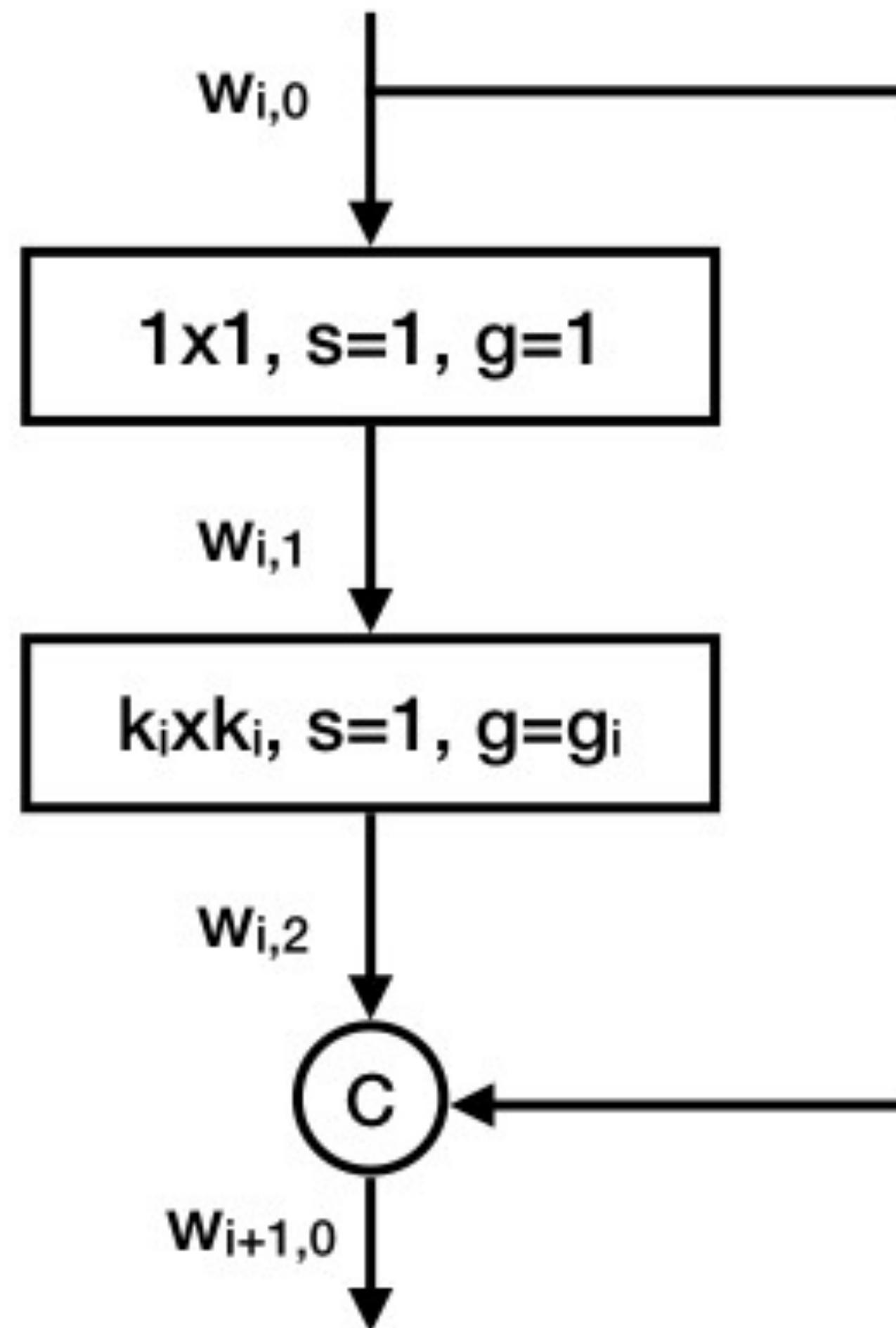
Concatenation, not addition

=> Grouping requires same dimensionality

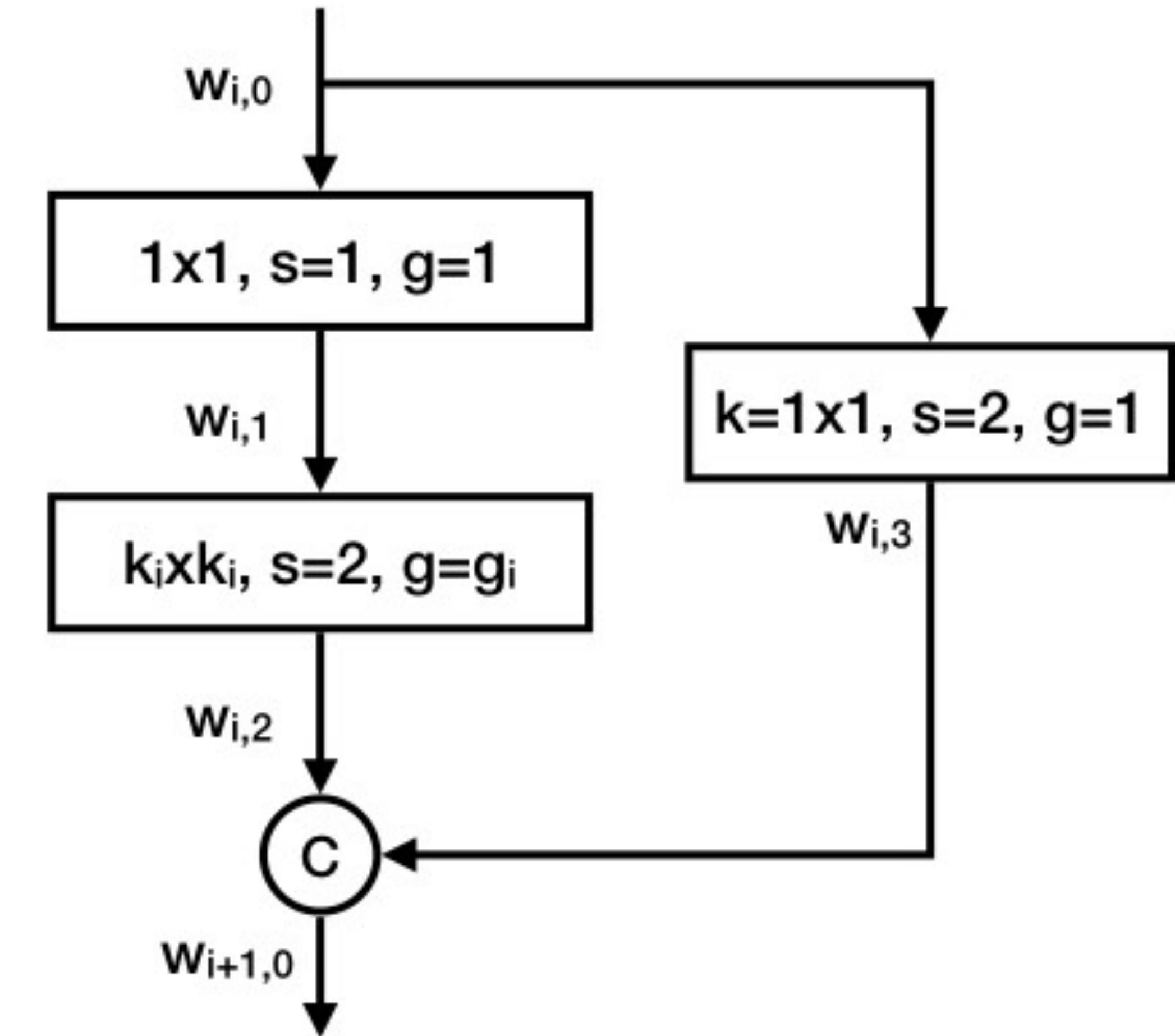
=> Transition Layers based on BatchNorm, 1x1 convs, and max pooling



DENSENET BOTTLENECK BLOCK



Without downsampling



With downsampling

WRAPPING UP

SUMMARY

Convolutions as the foundation of modern architectures for image classification and beyond

- Very computational intensive => inline with modern processor architectures

- Equivariance, resp. invariance in combination with pooling

Residuals for deep architectures

- Skip connections act as an add gate which acts as a gradient distributor

Dense connectivity for even deeper architectures

- High resource demands (usually MACs, parameters, units), but the architecture has also been reported as being resource-efficient

Metrics for computational efficiency

- Hard metrics: number of MACs, parameters, and units

- Soft-ish metric: parameter or MACs required to achieve a certain target accuracy

EFFICIENCY METRICS

	MACs	Parameters (weight state)	Units (activation state)
FC	$MAC_f = WHCO$	$W_f = WHCO$	$U_c = O$
Convolution	$MAC_c = (EF \cdot RSC) \cdot M$	$W_c = RSCM$	$U_c = EFC$
Grouped convolution	$MAC_{cg} = \frac{MAC_c}{g}$	$W_{cg} = \frac{W_c}{g}$	$U_{cg} = EFC$