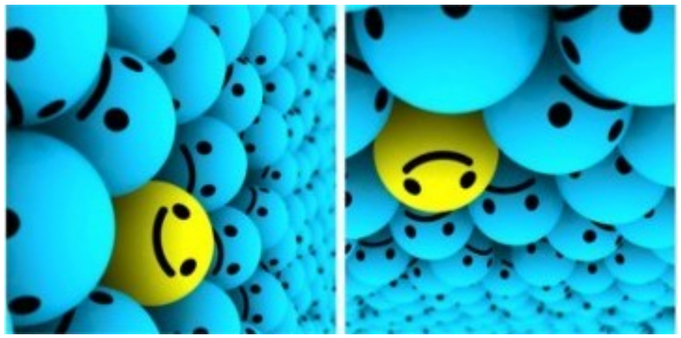
EMBEDDED MACHINE LEARNING LECTURE 06 - NEURAL ARCHITECTURE DESIGN

Holger Fröning
holger.froening@ziti.uni-heidelberg.de
HAWAII Group (formerly Computing Systems), Institute of Computer Engineering
Heidelberg University

EXAM

Save date & place

Feb 6, 2025, 16:15-18:15 - large lecture hall at COS (INF230)

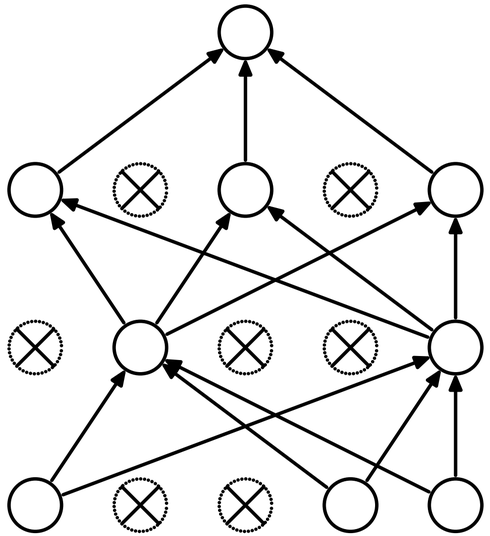


Data Augmentation

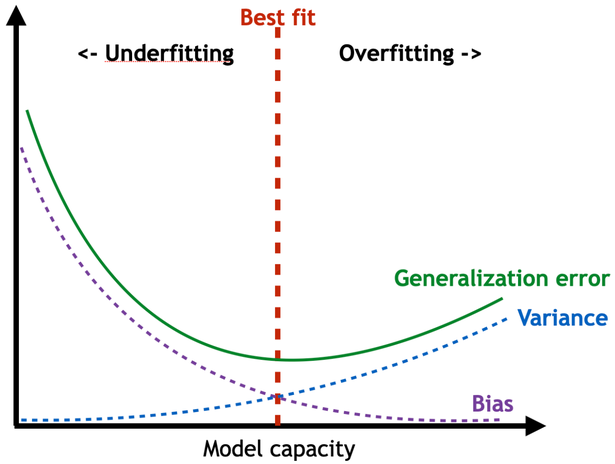
$$\mathcal{J}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(y(\mathbf{w},\mathbf{x}_n),t_n) + \lambda\mathcal{R}(\mathbf{w})$$

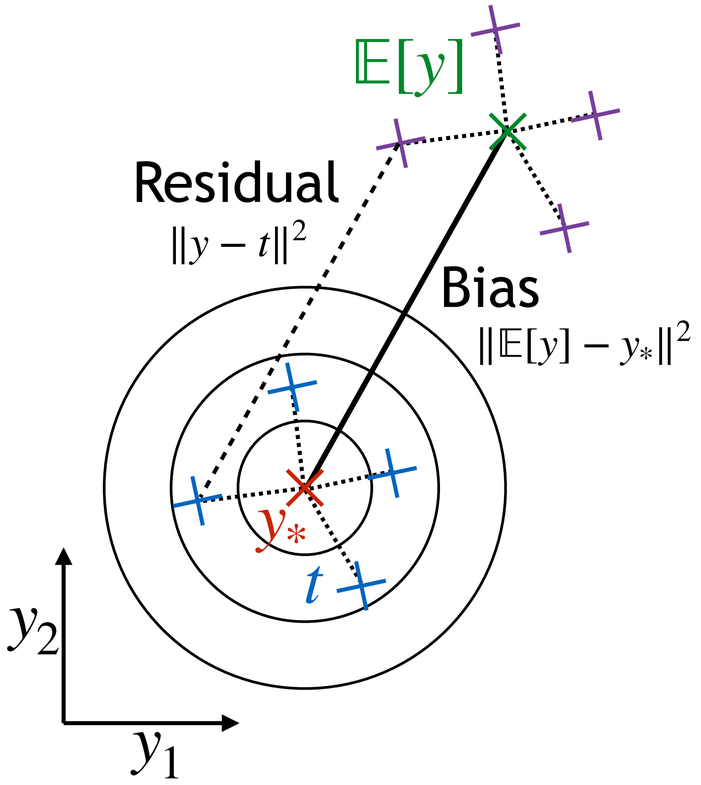
Weight Decay L1/L2

NORMALIZATION

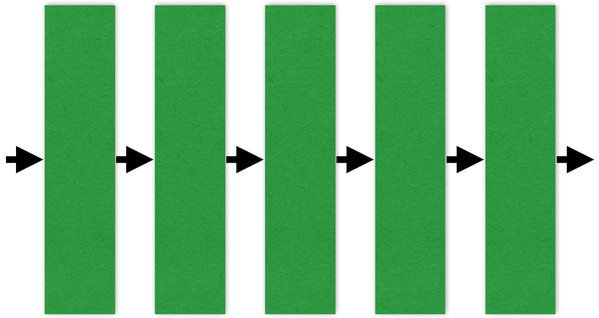


Dropout





DEEP NEURAL ARCHITECTURES …



… consist of sequentially connected layers

- I.e., the output of one layer is the input of another

Result: layer interdependence

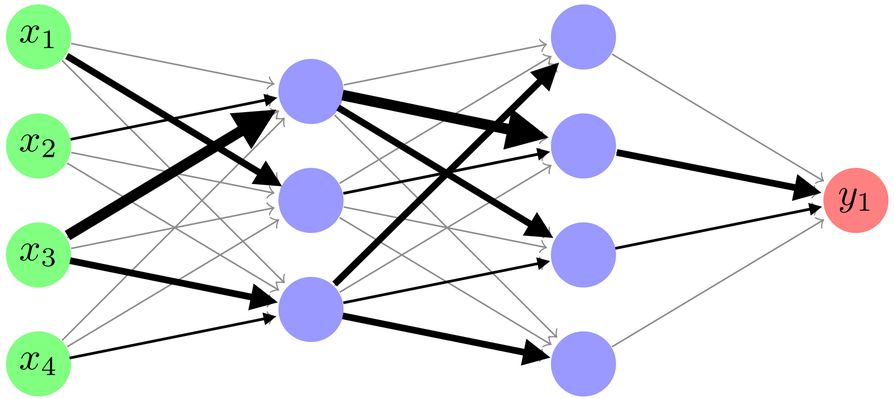
- Updated parameters of one layer change the output distribution
- Changes how downstream layers behave and learn
- Optimizer must account for these cascading effects, which can destabilize training

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

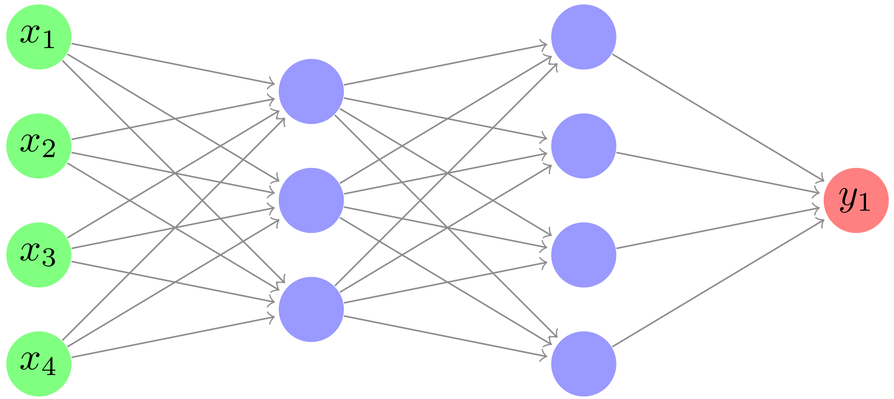
BATCH NORMALIZATION

Batch normalization is a method of adaptive reparametrization, motivated by this difficulty

- 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

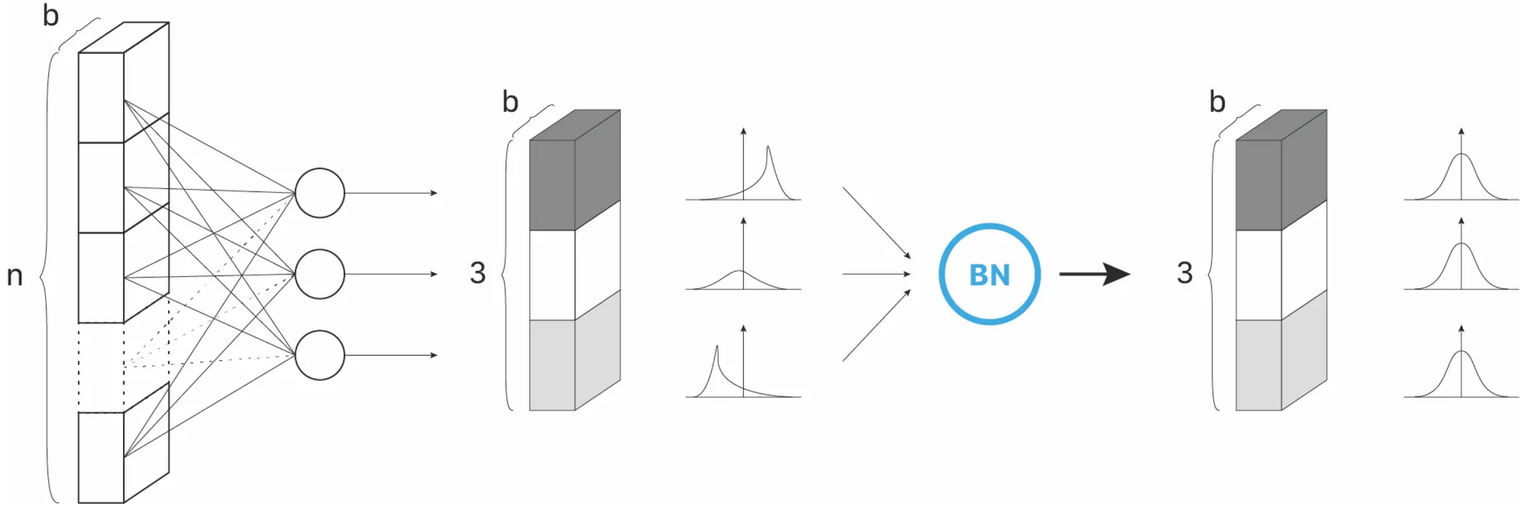
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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



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 β => 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

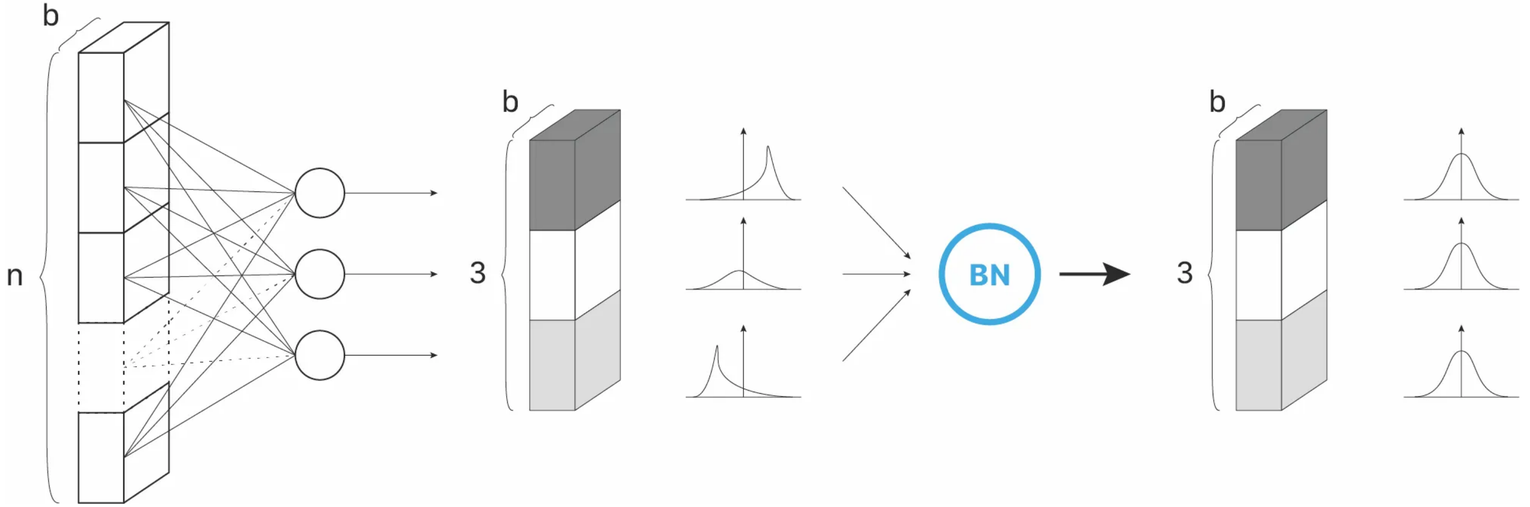
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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



Example of 3 hidden layers with a batch size b

1. Replace $\mathbf{Wx + b}$ 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: μ => μ', σ => σ'

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

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] by stabilizing the distribution of activations

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]

Improved gradient flow by keeping the scale of layer activations consistent throughout training

Also: less sensitive to hyperparameters such as learning rate

Theory 3: BN introduces some noise into the network because it normalizes over a mini-batch rather than the full dataset

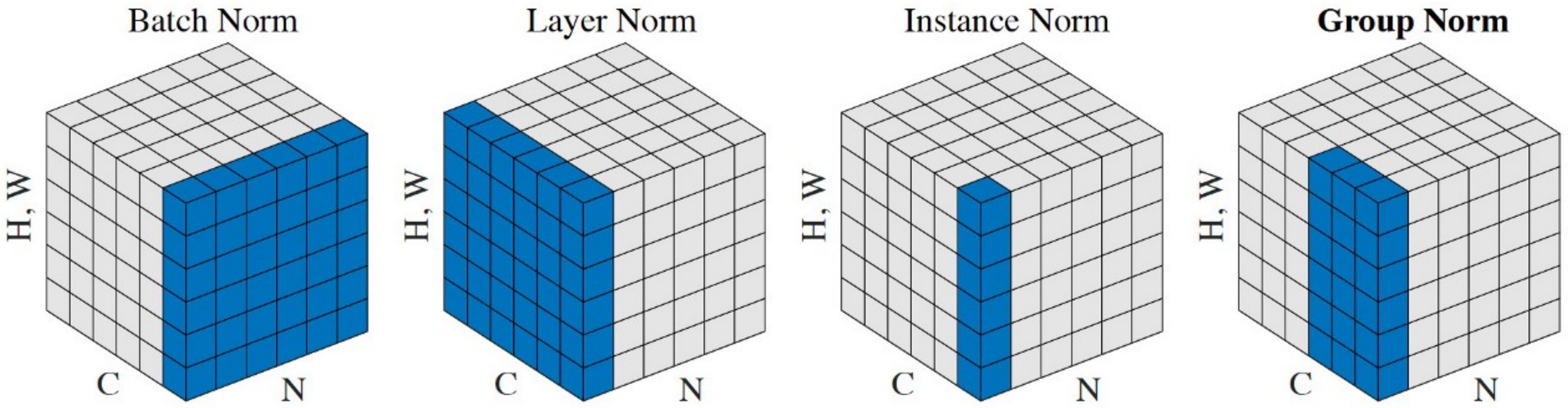
Observation: BN smoothes the optimization landscape [3]

[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

Y. Wu and K. He, "Group Normalization", ECCV 2018, arxiv.org/abs/1803.08494

CONVOLUTIONAL ARCHITECTURES

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 $\mathbf{B}$

ofmap = output filter map (output activations)

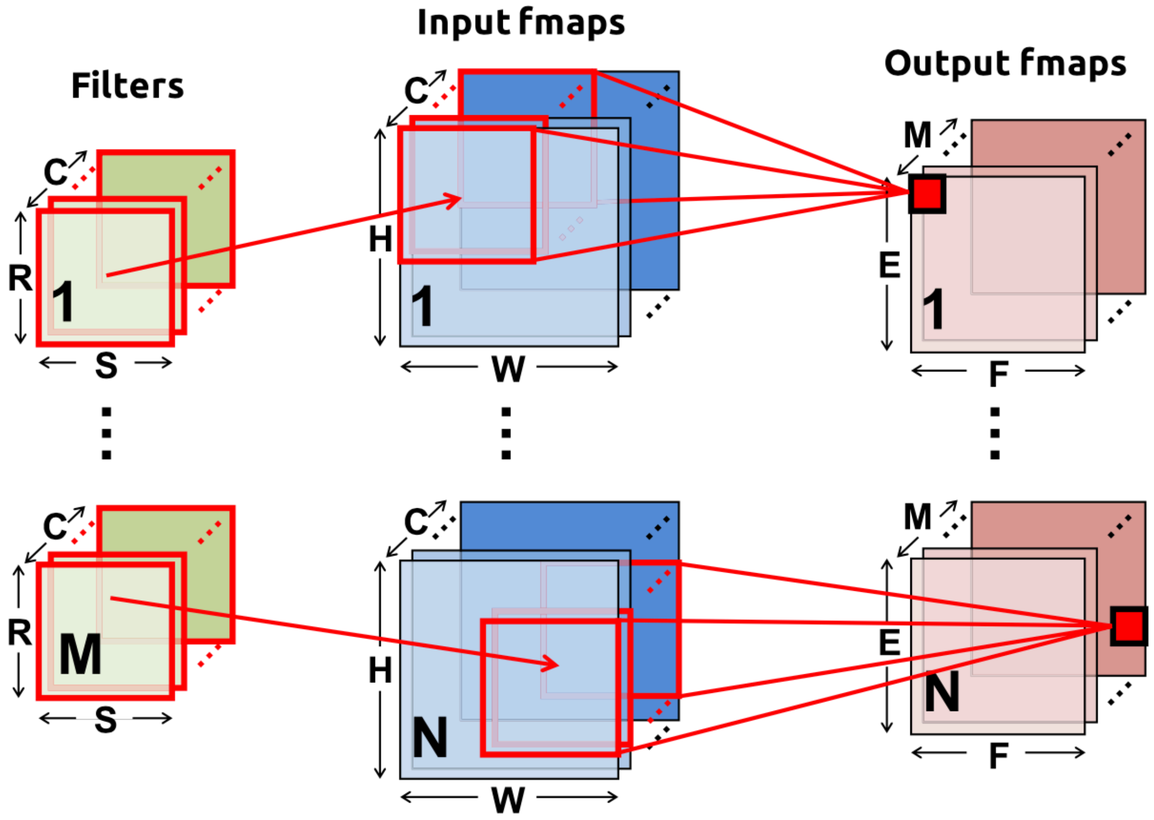
ifmap = input filter map (input activations)

$$E = (H - R + U)/U$$
$$F = (W - S + U)/U$$

N	Batch size (3D fmaps)	0 <= z <= N
M	number of 3D filters / number of ofmaps	0 <= u <= 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 <= x <= F, 0 <= y <= E
U	stride	

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

CONVOLUTION OPERATION



Output filter height

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

For padding P

Output filter width

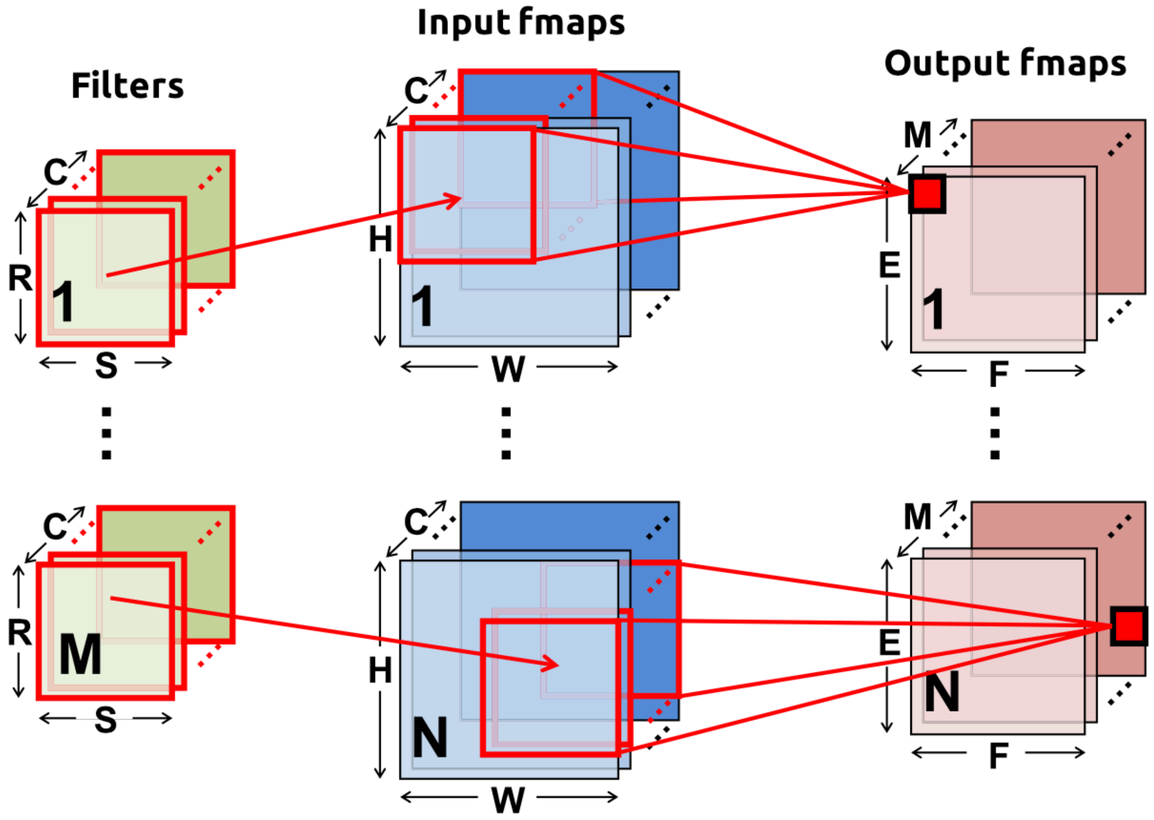
$$F = \frac{W - S + U + 2P}{U}$$

For padding P

Output filter depth (channels) equals numbers of filters M

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

CONVOLUTION: NUMBER OF PARAMETERS



Convolution

Weights $W_c = RSCM$

Biases $B_c = M$

Total $P_c = W_c + B_c$

Compare to FC

Number of neurons O

Weights $W_f = WHC \cdot O$

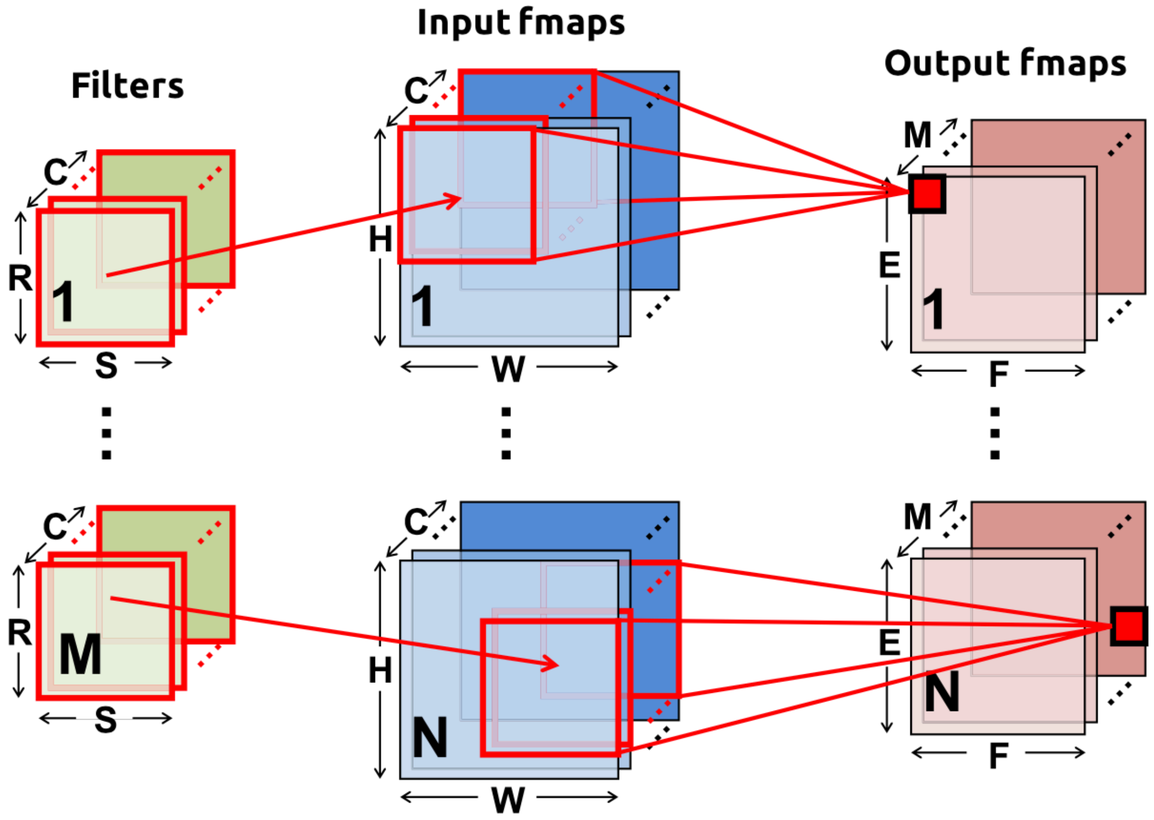
Biases $B_f = O$

Total $P_f = W_f + B_f$

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

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

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

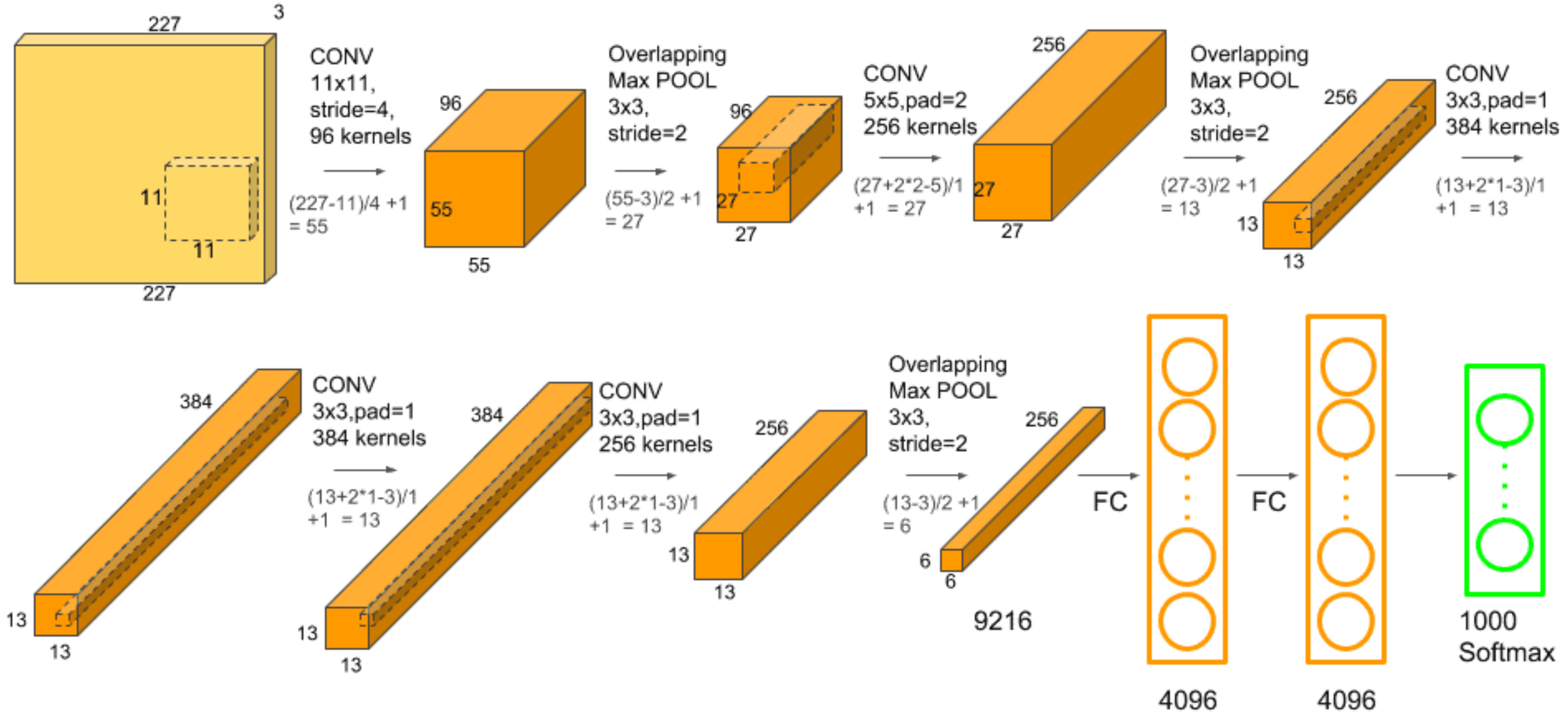
$$\text{MAC}_f = W_f = WHC \cdot O$$

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

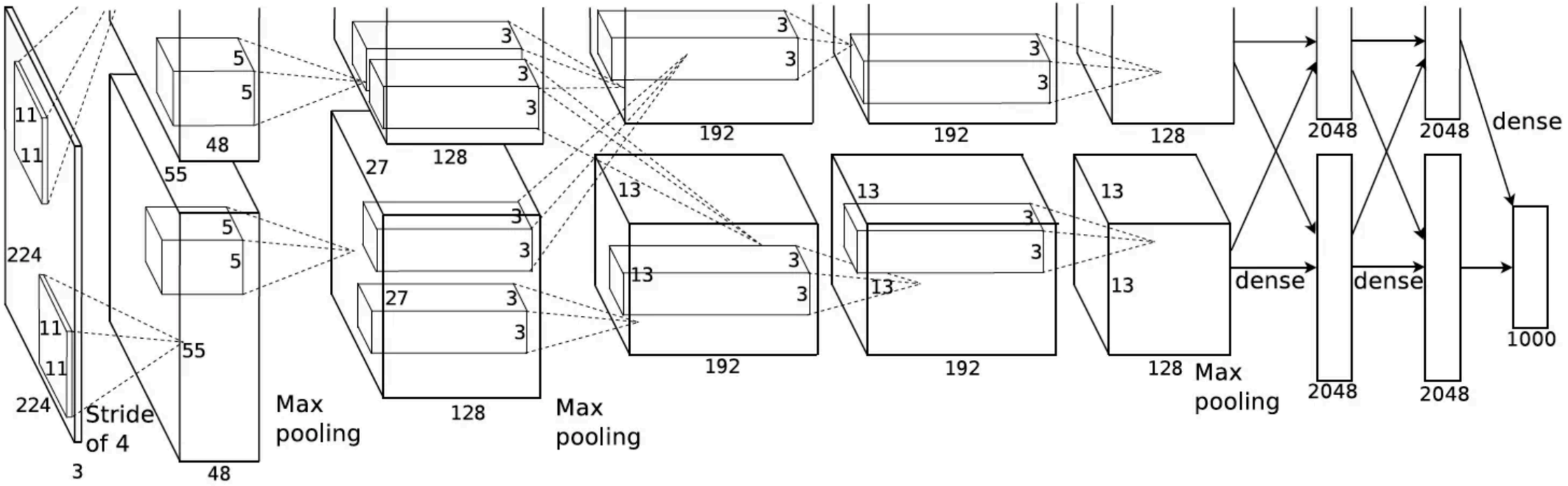
ALEXNET

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

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



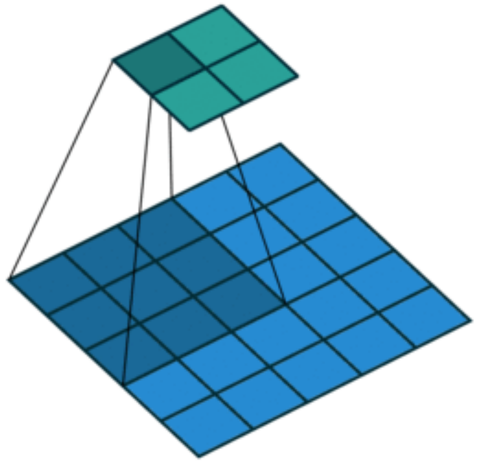
ALEXNET


11
11
224
224
3
Stride
of 4
5
5
48
55
48
5
5
55
3
3
3
128
27
128
3
3
27
3
3
Max
pooling
Max
pooling
3
3
192
13
13
3
3
192
3
3
192
13
13
3
3
192
128
13
13
128
Max
pooling
dense
dense
2048
2048
2048
2048
dense
1000

DOWNSAMPLING BY POOLING

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

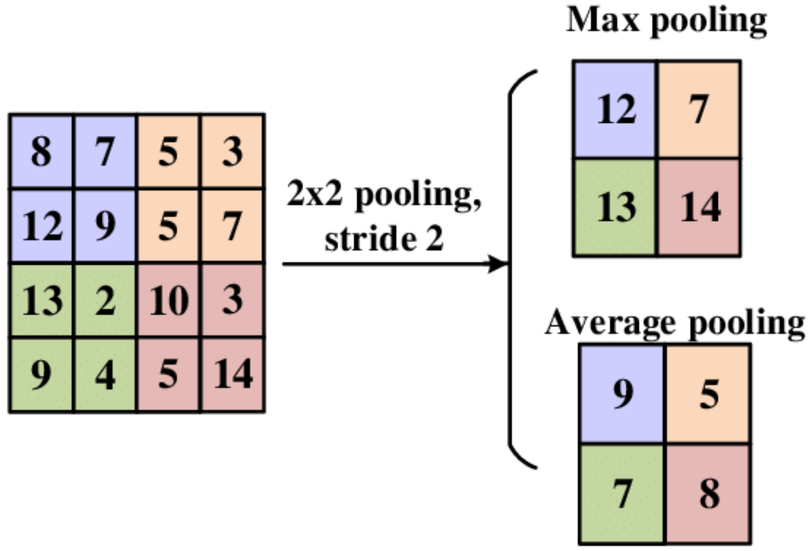
- Pool size = kernel size k, and stride s (usually $s = k$)
- Operates over each channel independently
- No learnable parameters



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

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

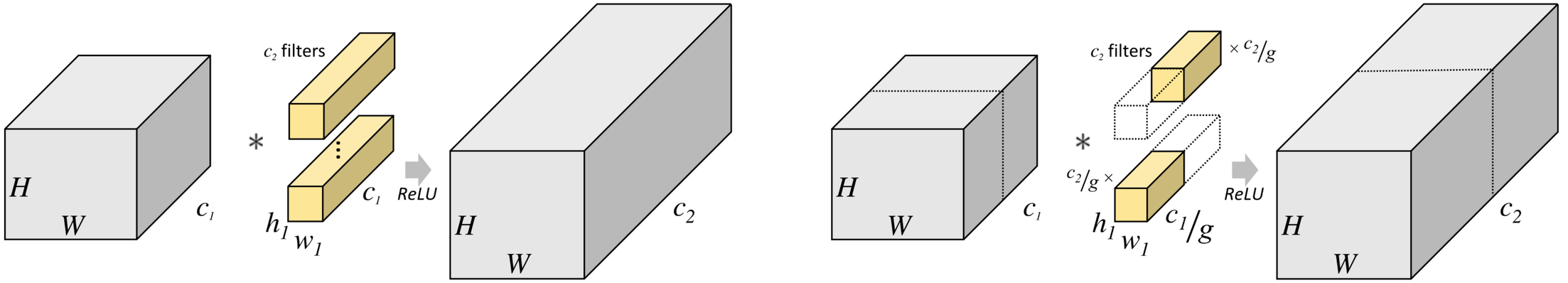
   Convolutional layers: equivariant (input translations are preserved)

   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

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

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

PUTTING IT ALL TOGETHER (FOR ALEXNET)

	Input/ output height H / E	Input/ output width W / F	Input/ output channels C	Filter count M	Filter / pooling height R	Filter / pooling width S	Filter / pooling stride U	Filter / pooling padding P	Filter / pooling groups G	FC neurons O	MACs	Params w+b	Output act.
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

INCEPTION

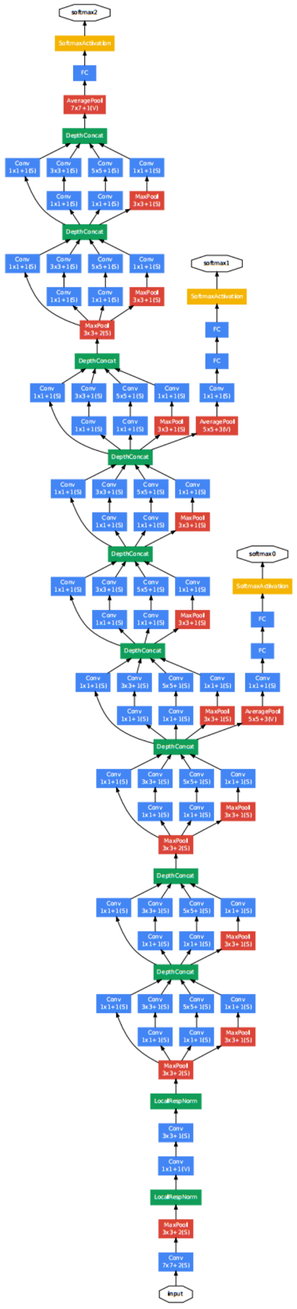
Main motivation: **reduce number of parameters** to enable deployment on mobile devices

- Google's datacenter allow to train virtually any model independent of size
- Similarly, lots of units (space complexity for backprop) is ok

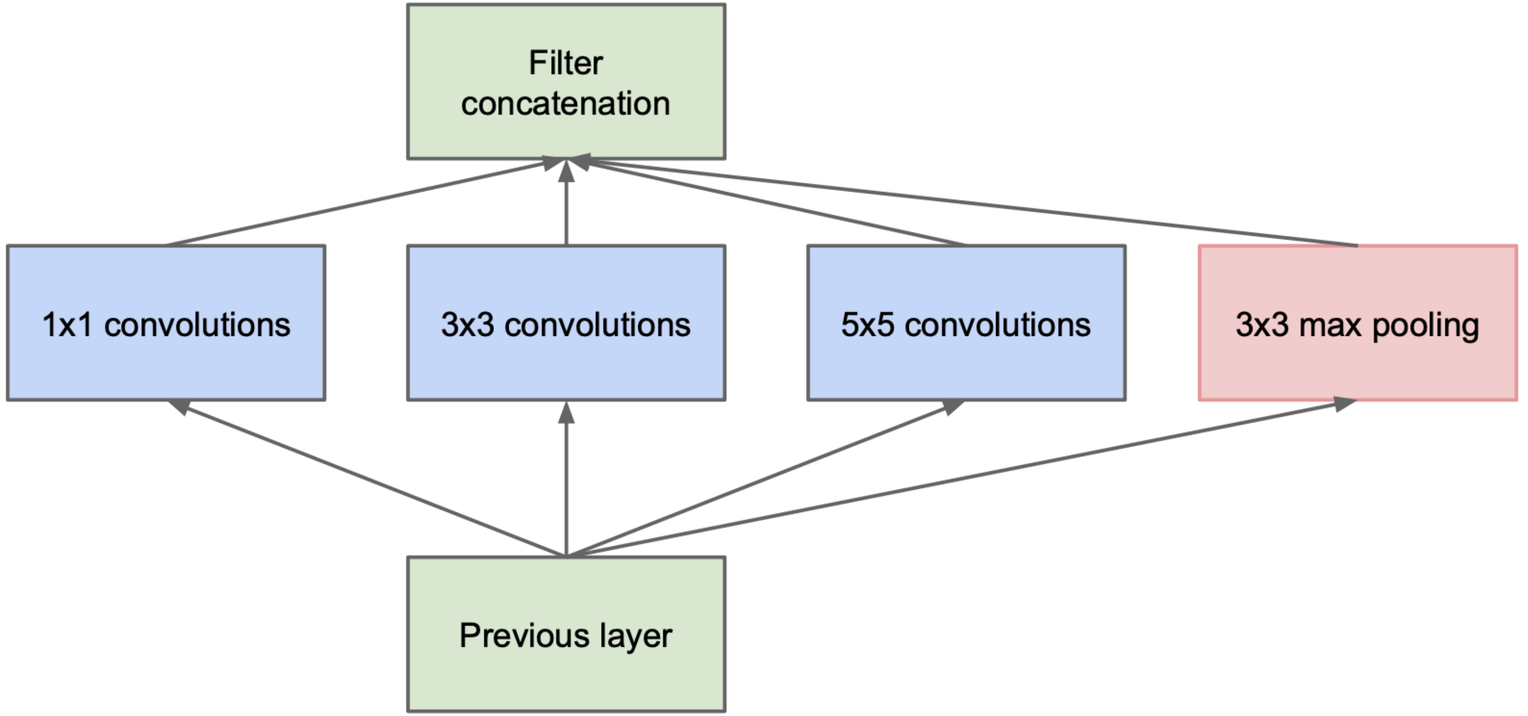
1. Convolutions only, no FC layers any more
   - Going deeper, 22 layers total
2. Replace large convolutions by multiple small convolutions
   - Two 3x3 filters have less parameters than one 5x5 filter, but cover a similar field

$$W_{3x3} = 9CM,\ W_{5x5} = 25CM$$

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



INCEPTION MODULE, NÄIVE VERSION

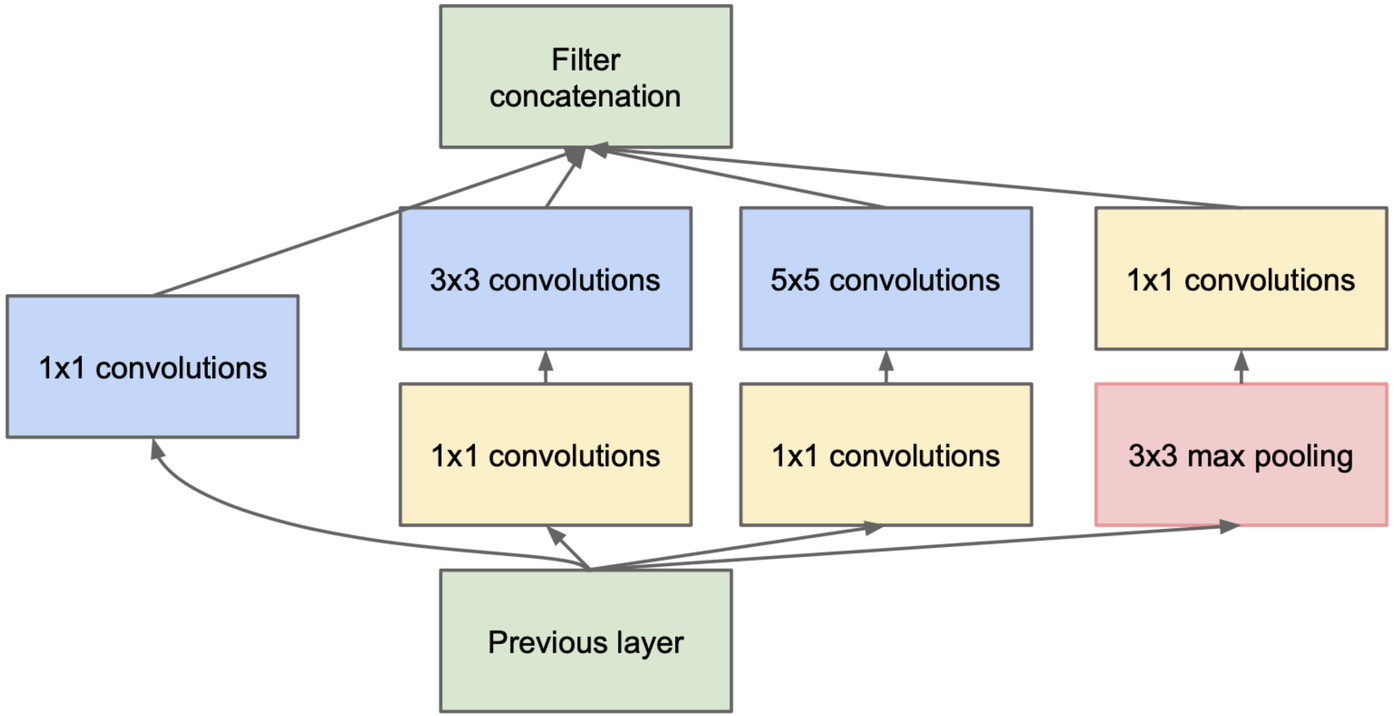


Key idea: use different convolutions in parallel, forward all results to the next layer which decides on importance

Max pooling included as “historically” good for complexity reduction

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

INCEPTION MODULE WITH DIMENSIONALITY REDUCTION



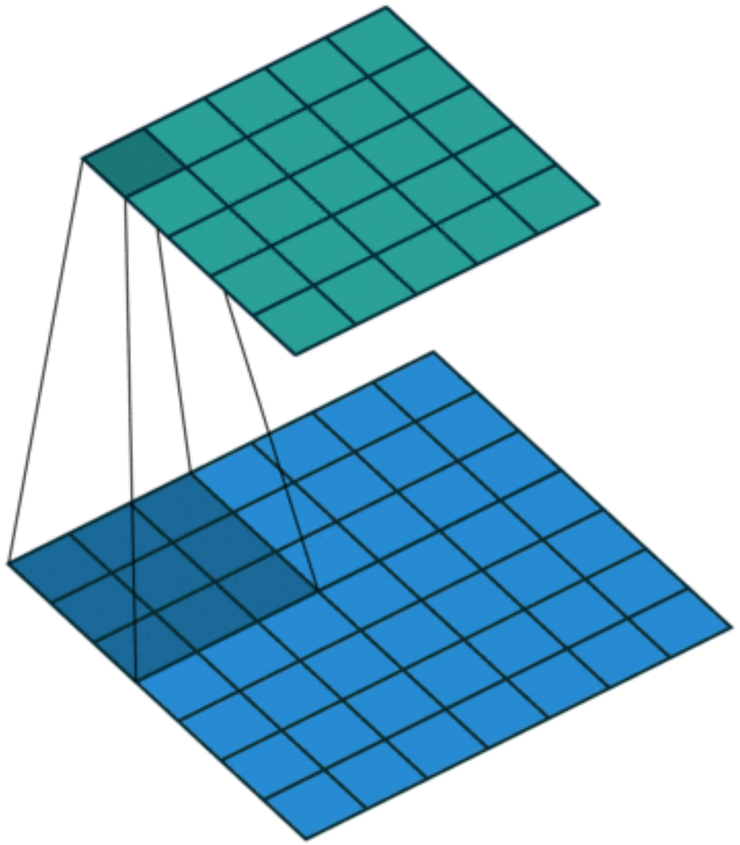
1x1 convolutions inspired by “Network-in-Network” architecture

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

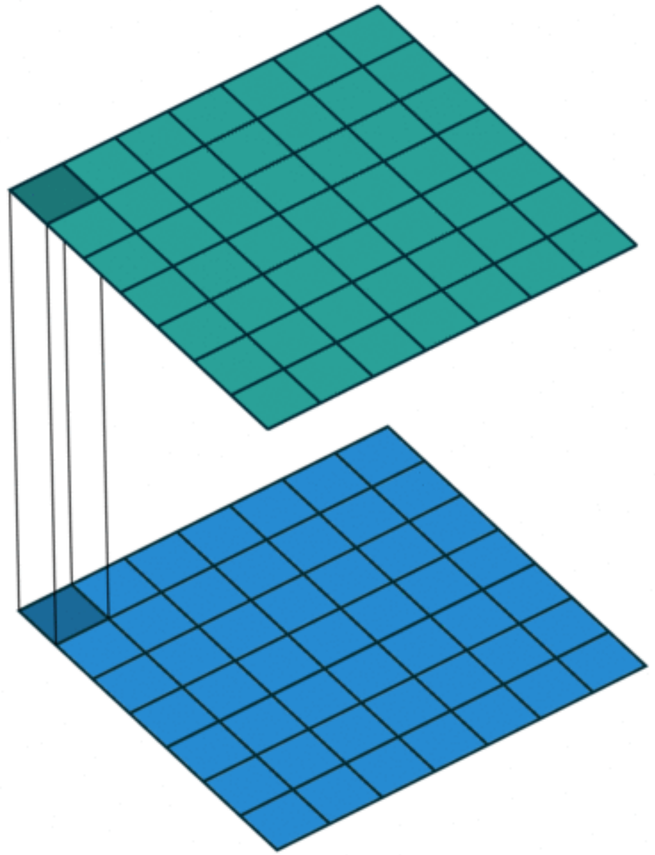
Why though?

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

https://github.com/vdumoulin/conv_arithmetic

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

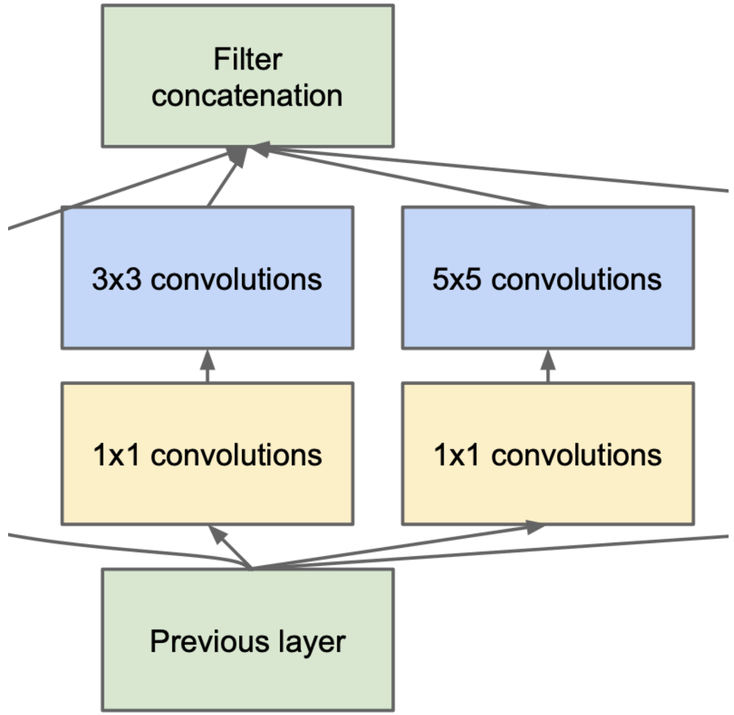
Concatenating the output of feature maps of multiple convolutions also increases the input depth

=> 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 $1x1$ convolution rather as $1x1xC$, 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 $\text{MAC}_c = (EF \cdot RSC) \cdot M$

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

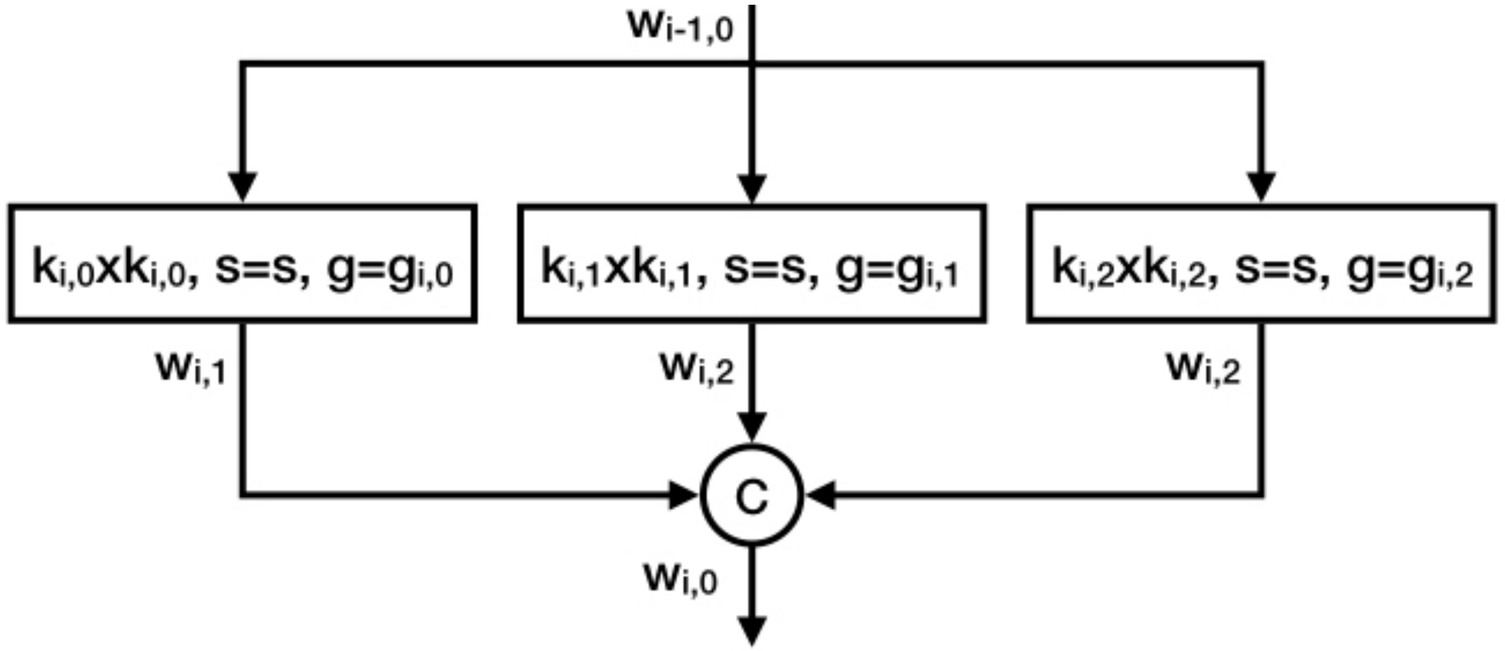
Consider an input of $WxHxC = 28x28x192$

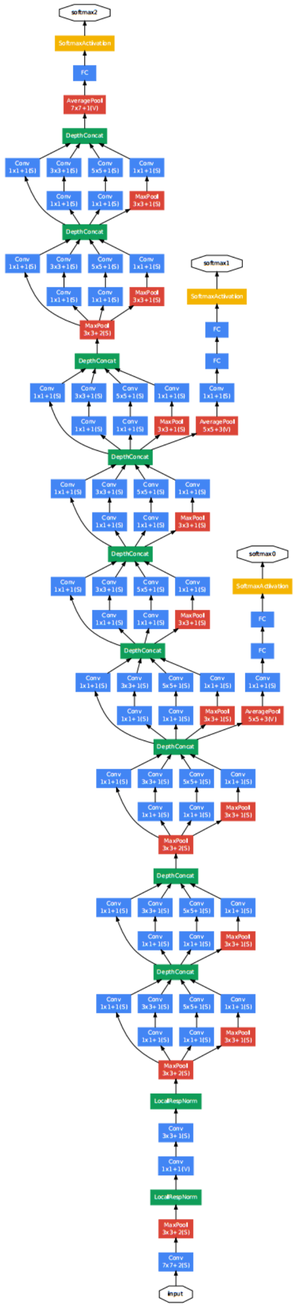
Max pooling before inception block 3a, padding set to preserve dimensions

	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

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

$$\frac{\partial l}{\partial \mathbf{x}_1} = \frac{\partial l}{\partial \mathbf{x}_L} \cdot \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \cdot \ldots \cdot \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \cdot \ldots \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \cdot \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}$$

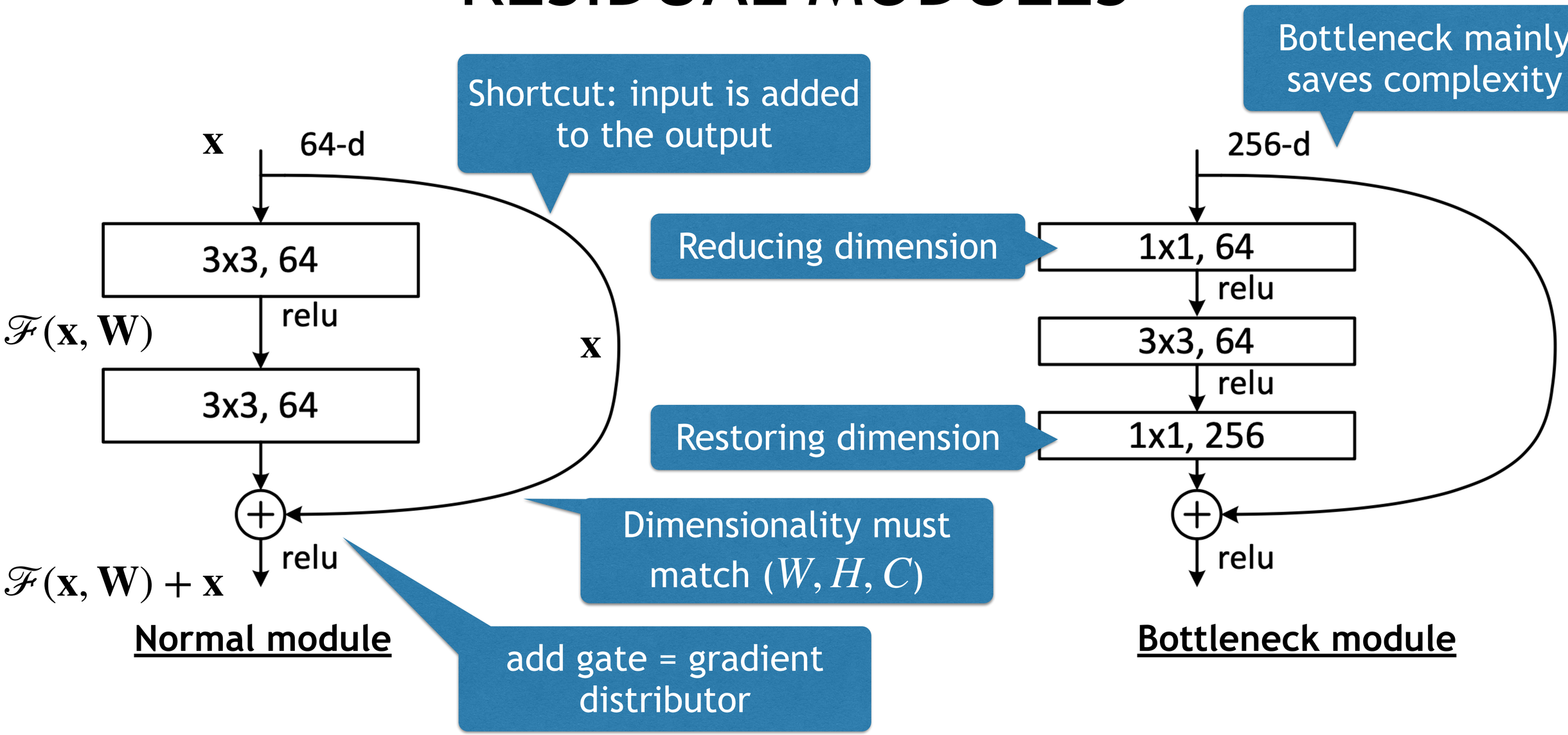
Observation 1: Downstream gradients greatly depend on value of activations $\mathbf{x}_i$ and activation function f

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

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

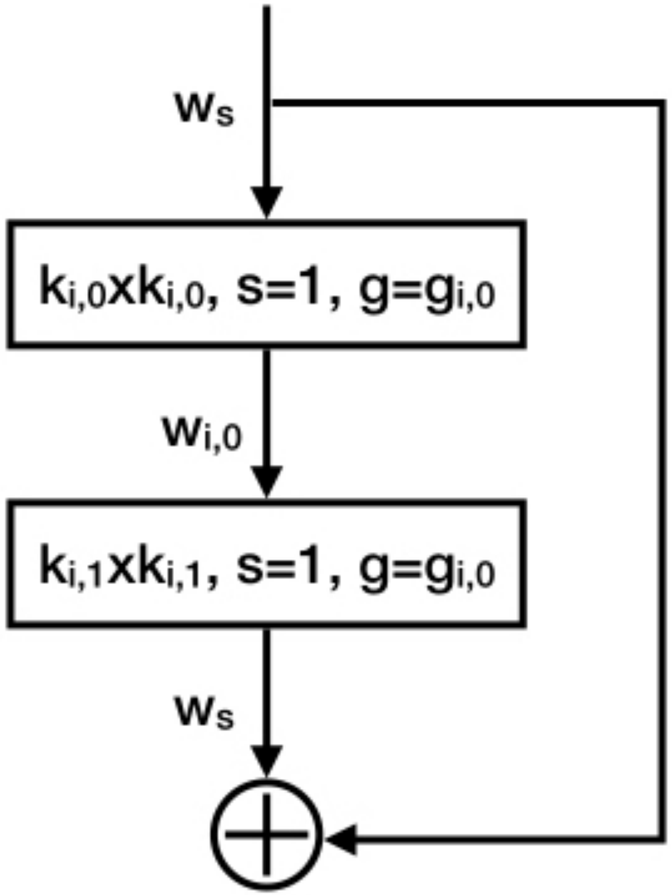
Given comparable effort (epochs), deeper networks have larger training and test error

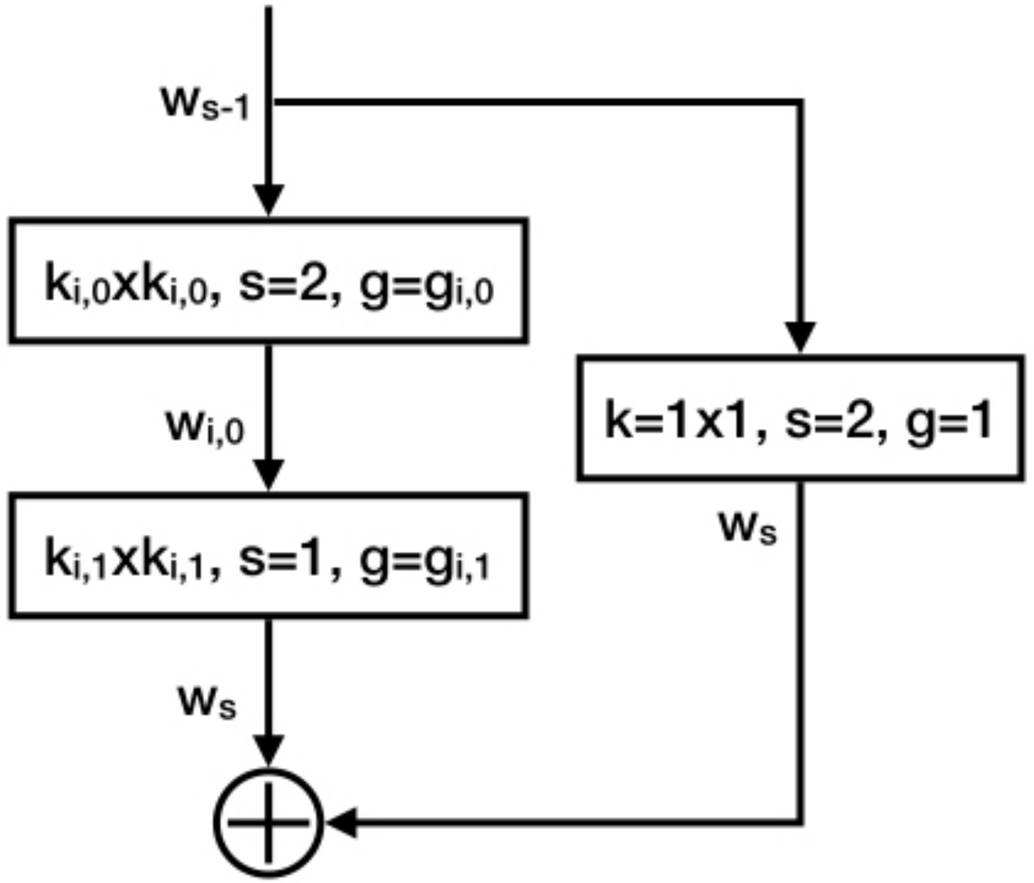
RESIDUAL MODULES



Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, arxiv.org/abs/1512.03385, 2015

RESIDUAL AND DOWNSAMPLING





Without downsampling

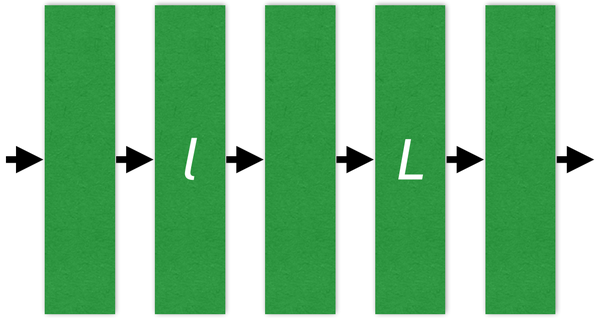
With downsampling

Enables $w_s \neq w_{s-1}$

RESIDUAL NETWORKS

Residual Unit

$$x_{l+1} = \mathcal{F}(\mathbf{x_l}, \mathbf{W_l}) + \mathbf{x_l}$$



Residual Network

$$x_L = \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x_i}, \mathbf{W_i}) + \mathbf{x_l}$$

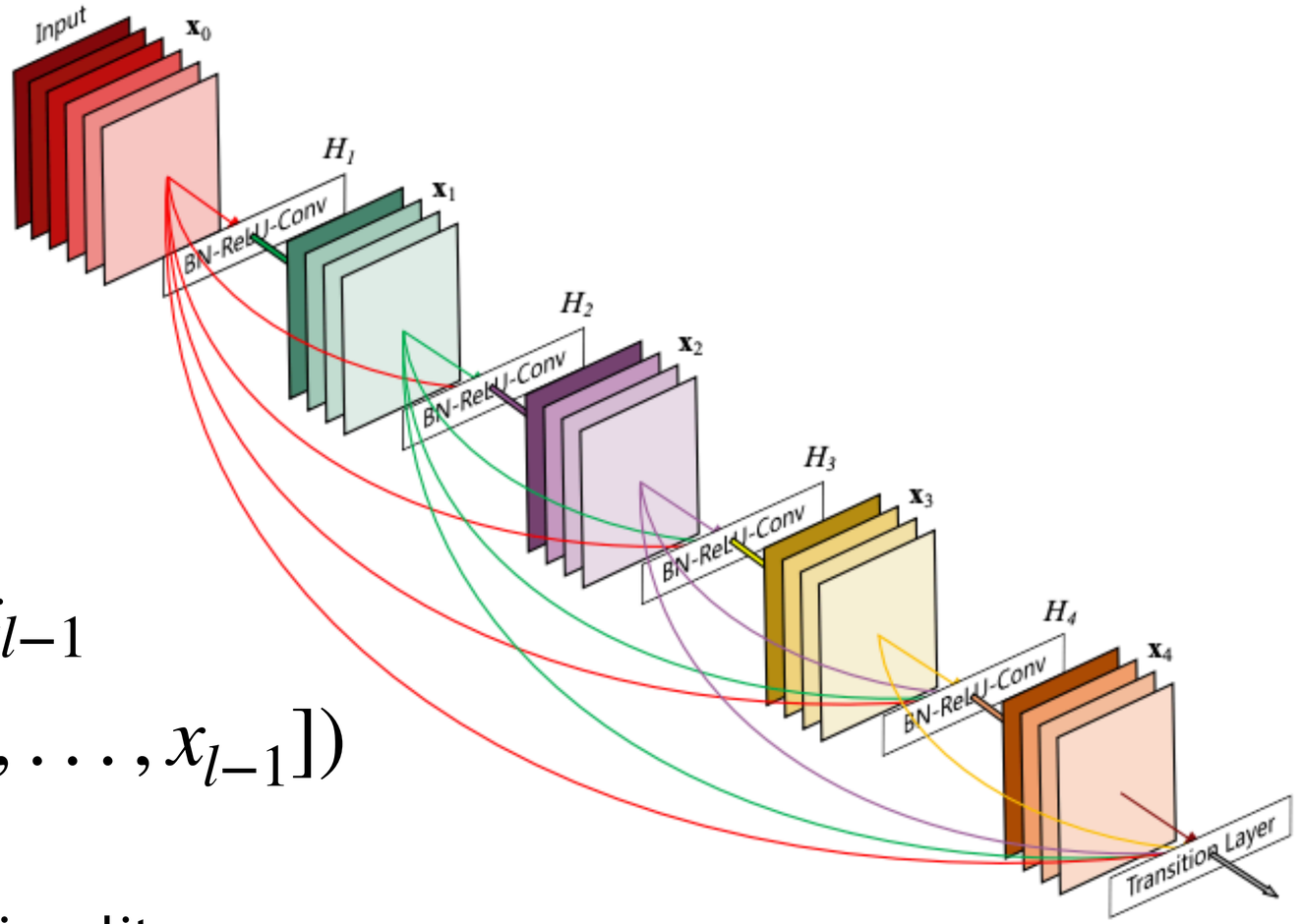
For any deeper unit L and shallower unit l

Backward Path

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

He et al., "Identity Mappings in Deep Residual Networks", https://arxiv.org/abs/1603.05027, ECCV 2016

DENSENET



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

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

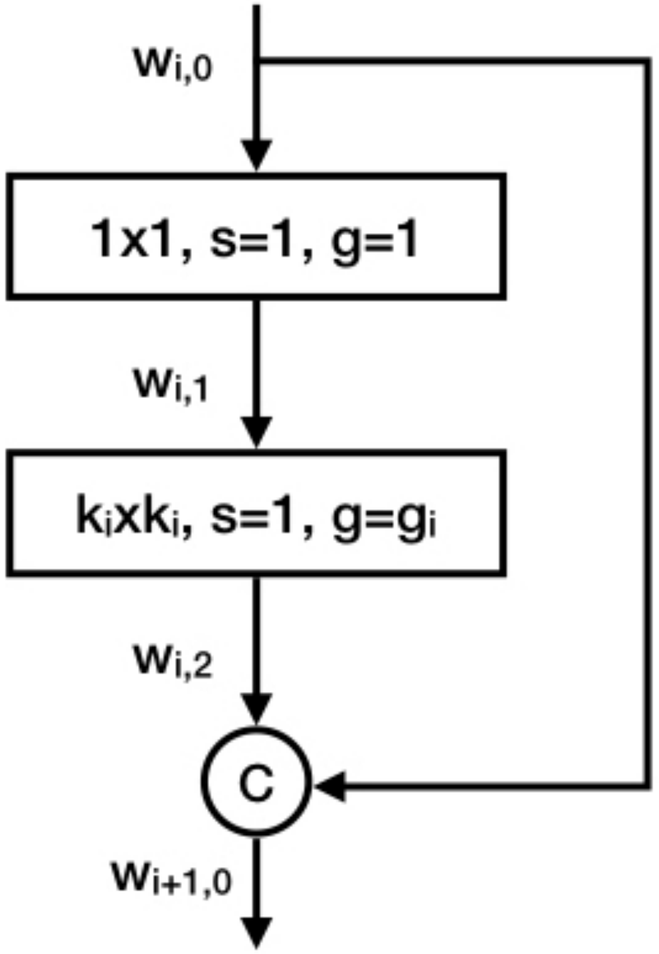
DenseNet layer: $x_l = \mathscr{F}_l([x_0, x_1, \ldots, x_{l-1}])$

Concatenation, not addition

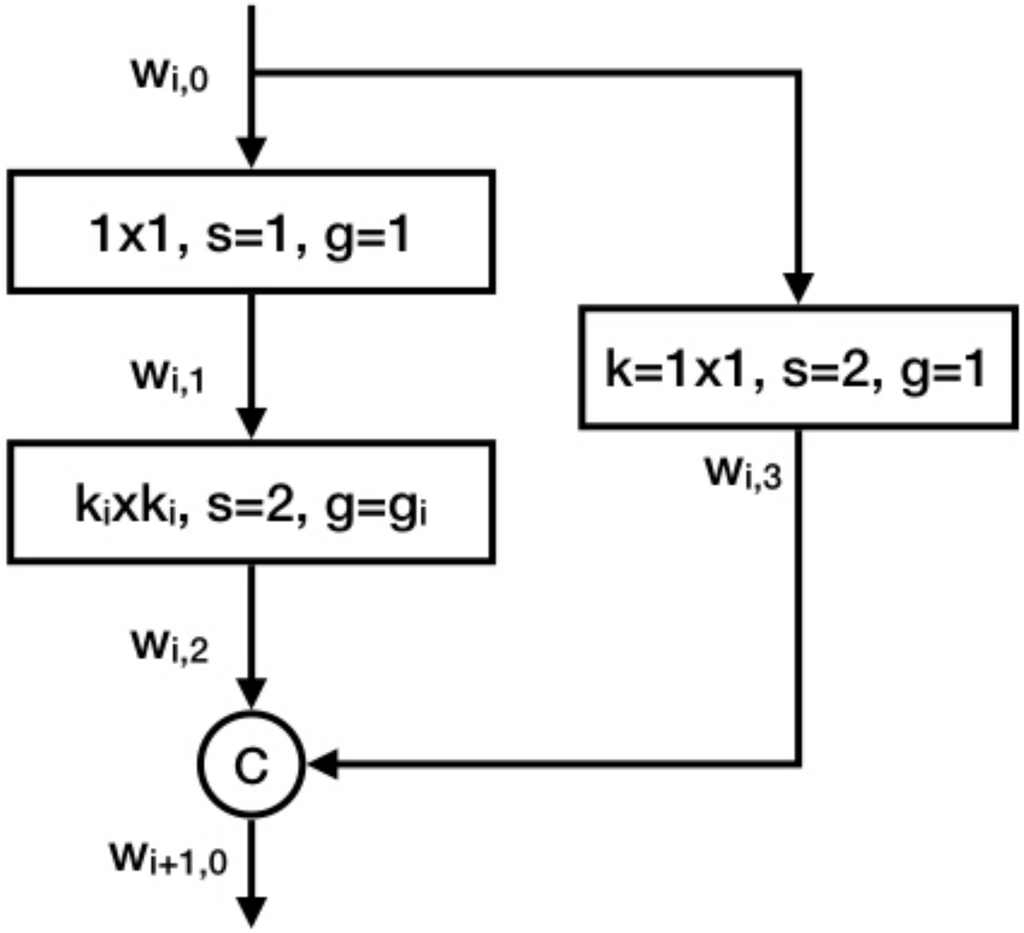
=> Grouping requires same dimensionality

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

DENSENET BOTTLENECK BLOCK



Without downsampling



With downsampling

WRAPPING UP

EFFICIENCY METRICS

	MACs	Parameters (weight state)	Units (activation state)
FC	$\text{MAC}_f = WHCO$	$W_f = WHCO$	$U_c = O$
Convolution	$\text{MAC}_c = (EF \cdot RSC) \cdot M$	$W_c = RSCM$	$U_c = EFM$
Grouped convolution	$\text{MAC}_{cg} = \frac{\text{MAC}_c}{g}$	$W_{cg} = \frac{W_c}{g}$	$U_{cg} = EFM$

SUMMARY

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

- Very computational intensive => inline with modern processor architectures
- Equivariance, resp. invariance in combination with pooling

Batch Normalization mitigates the interdependency between layers during training

- Among various other theories

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

Residuals for deep architectures

- Skip connections are based on an add gate which acts as a gradient distributor
- Dense connectivity for even deeper architectures

$$\mathcal{F}(\mathbf{x}, \mathbf{W}) + \mathbf{x}$$

Bottleneck layers balance efficiency and expressiveness

- Learning compact representations; compression also acting as a regularizer
- Making them crucial for building deep and computationally tractable networks

$$1x1xC$$