Neural networks in computer vision

Key idea

- Build complex functions by composing simple ones.
- Usually have a library of possible functions
  - Some functions have learnable parameters
- Fix hypothesis class by fixing *network architecture*
  - Fixed set of functions composed in fixed ways
  - Different settings of the learnable parameters yield different hypotheses

Recall: Empirical Risk Minimization

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(h(x_i; \boldsymbol{\theta}), y_i)$$

Neural network

$$\underbrace{\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^{N} \nabla L(h(x_i; \boldsymbol{\theta}), y_i)}_{\text{Gradient descent update}}$$

Computing the gradient of the loss

$$\nabla L(h(x;\boldsymbol{\theta}), y)$$

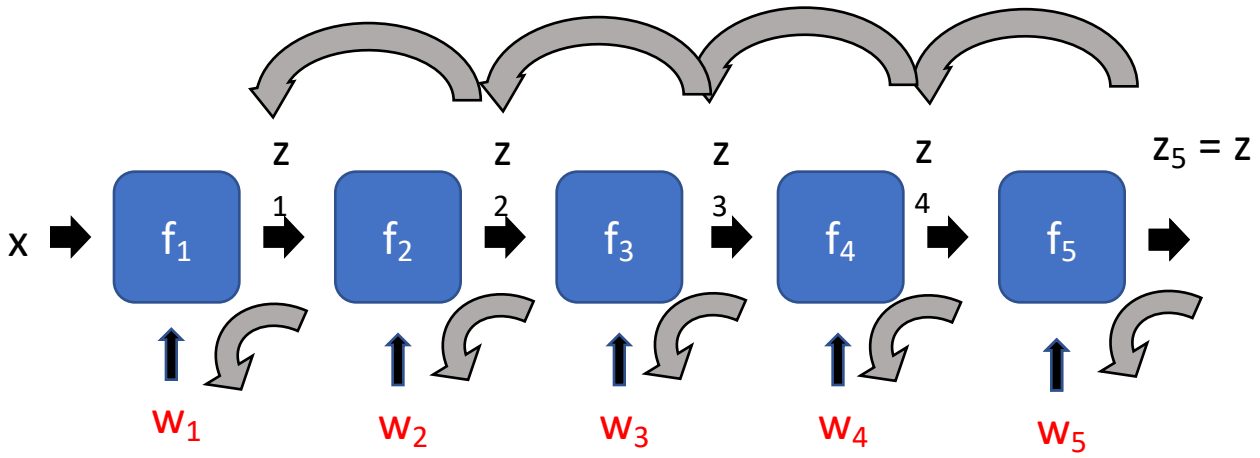
$$z = h(x;\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} L(z, y) = \frac{\partial L(z, y)}{\partial z} \frac{\partial z}{\partial \boldsymbol{\theta}}$$

Learning with function compositions

- $F = f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1$
- Suppose f_i has learnable parameters w_i , takes input z_{i-1} and produces output z_i
- Need to compute $\frac{\partial F}{\partial w_i}$. How?
- Key idea: recurrence
  - If we know $\frac{\partial F}{\partial z_i}$, then chain rule gives: $\frac{\partial F}{\partial z_i} \frac{\partial z_i}{\partial w_i}$, second term only requires each function be differentiable
  - Also $\frac{\partial F}{\partial z_i} = \frac{\partial F}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i}$

Learning with function compositions



Backpropagation

Backpropagation for a sequence of functions

$$z_i = f_i(z_{i-1}, w_i)$$
$$z_0 = x$$
$$z = z_n$$

Previous term

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i}$$

Function derivative

$$\frac{\partial z}{\partial w_i} = \frac{\partial z}{\partial z_i} \frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions

$$z_i = f_i(z_{i-1}, w_i) \qquad z_0 = x \qquad z = z_n$$

- Assume we can compute partial derivatives of each function

$$\frac{\partial z_i}{\partial z_{i-1}} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial z_{i-1}} \qquad \frac{\partial z_i}{\partial w_i} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial w_i}$$

- Use $g(z_i)$ to store gradient of z w.r.t z_i, $g(w_i)$ for w_i
- Calculate g_i by iterating backwards

$$g(z_n) = \frac{\partial z}{\partial z_n} = 1 \qquad g(z_{i-1}) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial z_{i-1}} = g(z_i)\frac{\partial z_i}{\partial z_{i-1}}$$

- Use gi to compute gradient of parameters

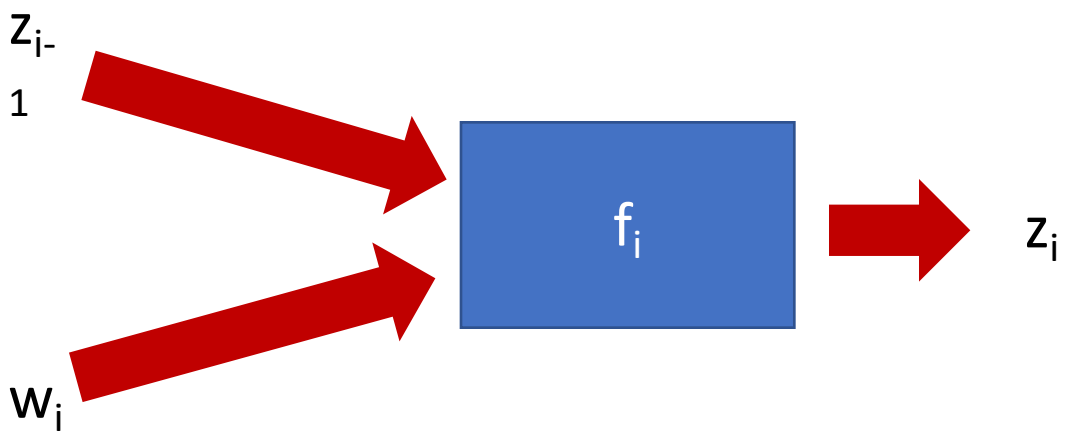
$$g(w_i) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial w_i} = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions

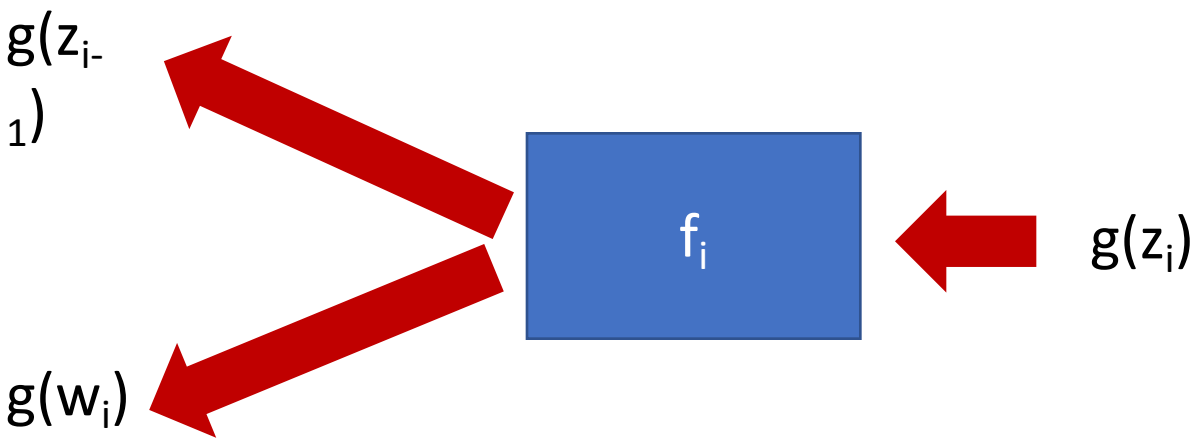
- Each “function” has a “forward” and “backward” module
- Forward module for f_i
  - takes z_{i-1} and weight w_i as input
  - produces z_i as output
- Backward module for f_i
  - takes $g(z_i)$ as input
  - produces $g(z_{i-1})$ and $g(w_i)$ as output

$$g(z_{i-1}) = g(z_i)\frac{\partial z_i}{\partial z_{i-1}} \qquad g(w_i) = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions



Backpropagation for a sequence of functions



Chain rule for vectors

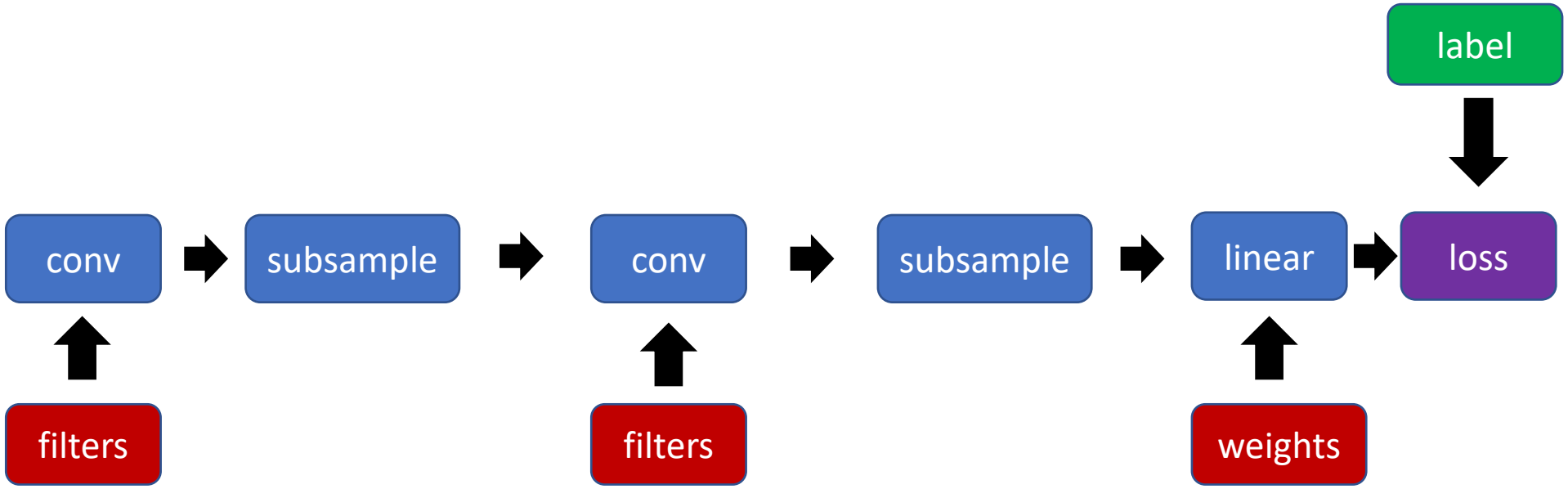
$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c}\frac{\partial c}{\partial b} \qquad \frac{\partial a_i}{\partial b_j} = \sum_k \frac{\partial a_i}{\partial c_k}\frac{\partial c_k}{\partial b_j}$$

$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}}(i,j) = \frac{\partial a_i}{\partial b_j}$$

Jacobian

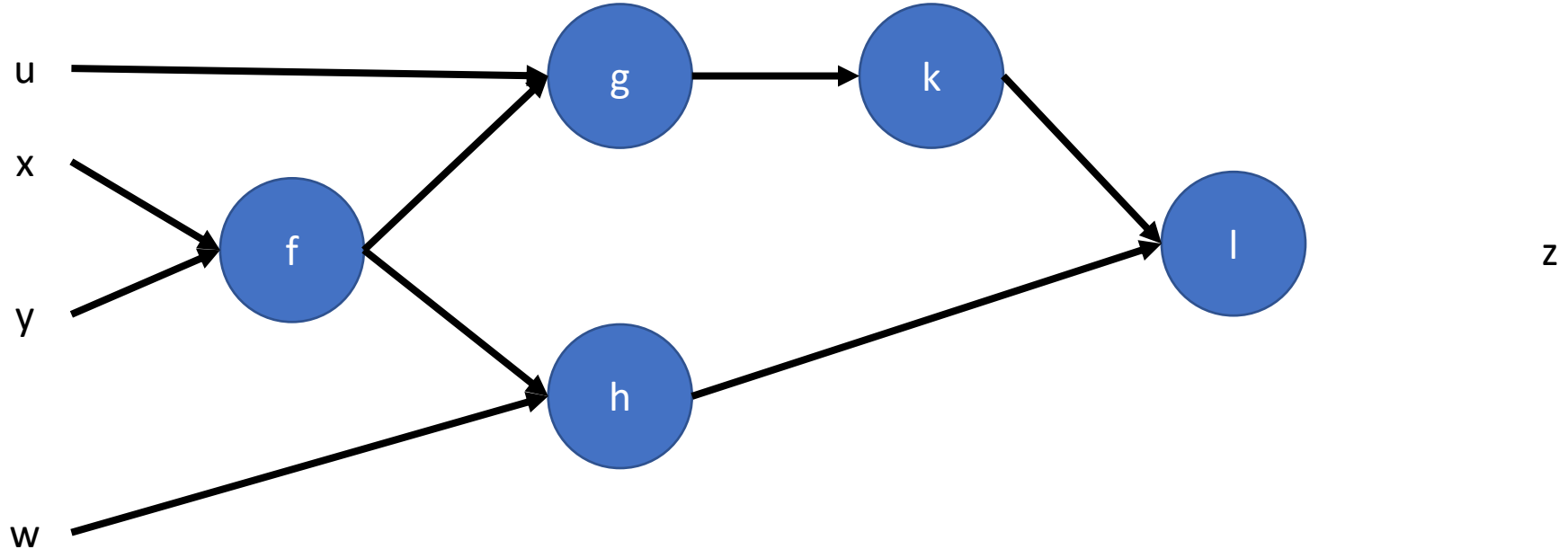
$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}} = \frac{\partial \mathbf{a}}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{b}}$$

Loss as a function



Beyond sequences: computation graphs

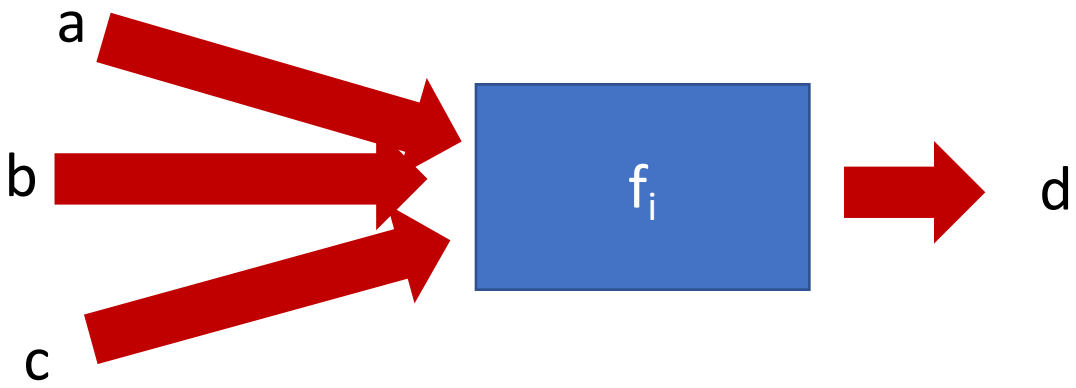
- Arbitrary *graphs* of functions
- No distinction between intermediate outputs and parameters



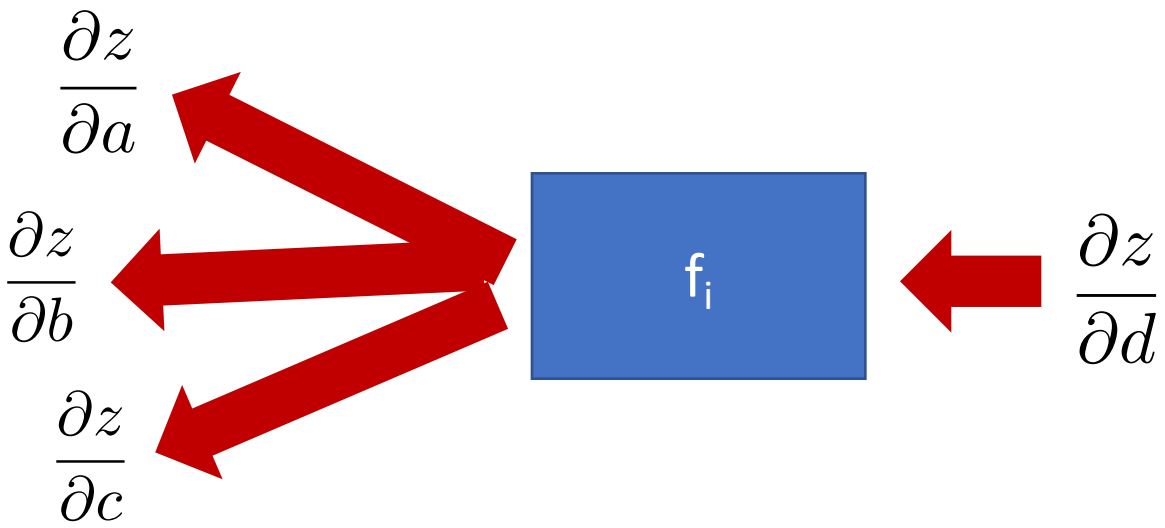
Computation graph - Functions

- Each node implements two functions
  - A “forward”
    - Computes output given input
  - A “backward”
    - Computes derivative of z w.r.t input, given derivative of z w.r.t output

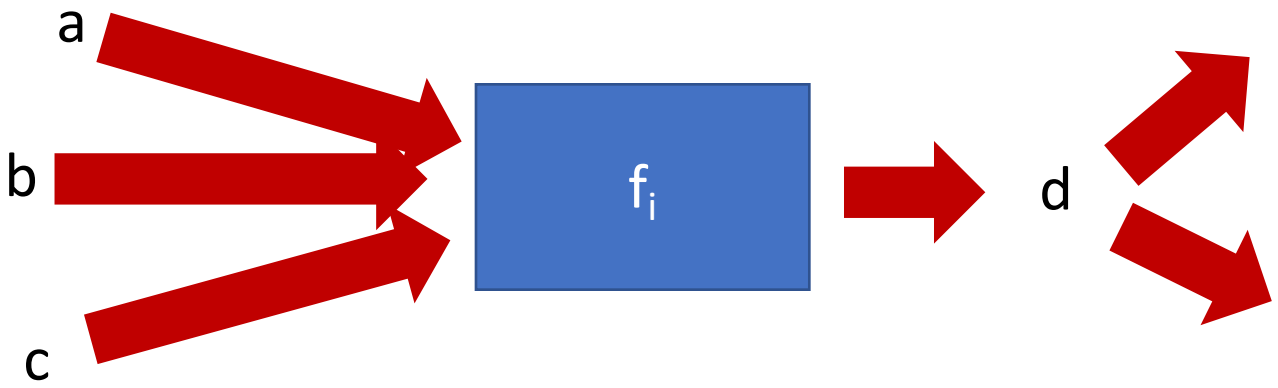
Computation graphs



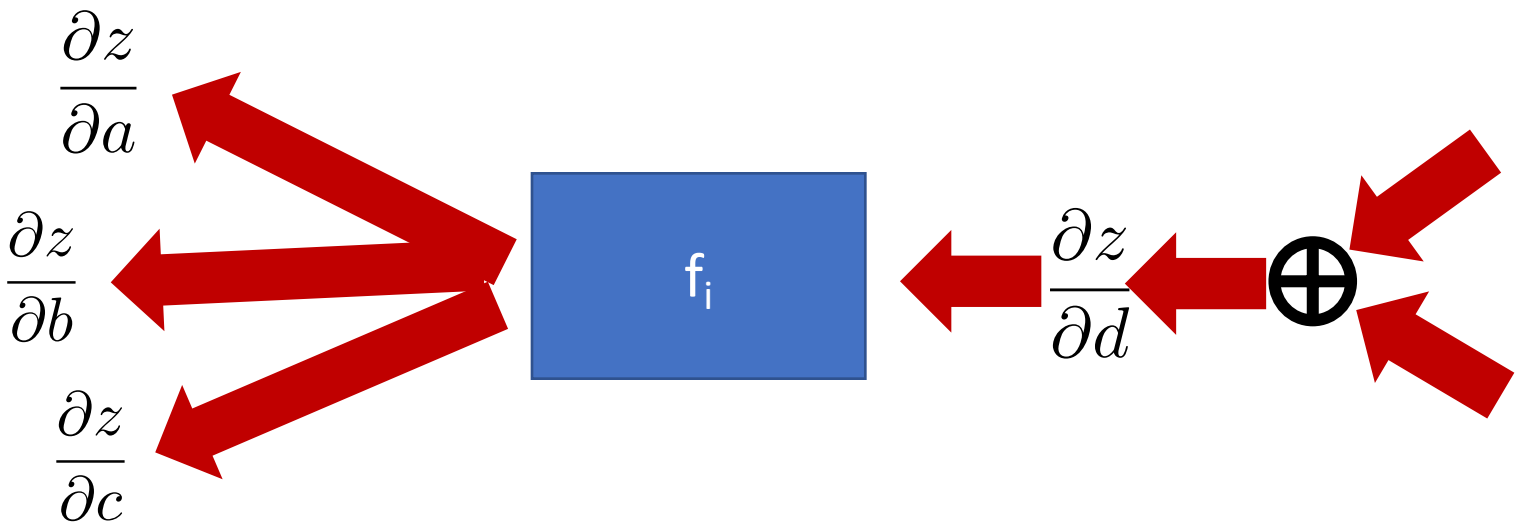
Computation graphs



Computation graphs



Computation graphs

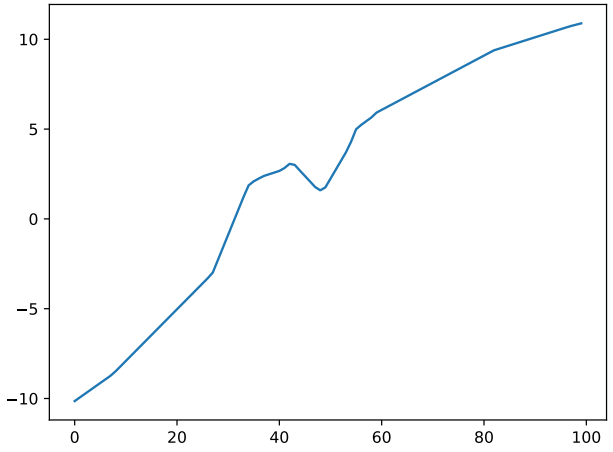


Module Library 1

- Linear
  - $y = Wx + b$
  - Learnable parameters: W, b
  - Hyperparameter: Output dimensionality
- Rectified Linear Unit (ReLU)
  - $y = \max(x, 0)$
  - Learnable parameters: None
  - Inspired from signal processing("half-wave rectification"); easy to implement in hardware.
- Softmax
  - $y_k = \frac{e^{x_k}}{\sum_i e^{x_i}}$
  - Learnable parameters: None
  - Converts a set of scores into probabilities (bounded in [0,1], sum to 1)

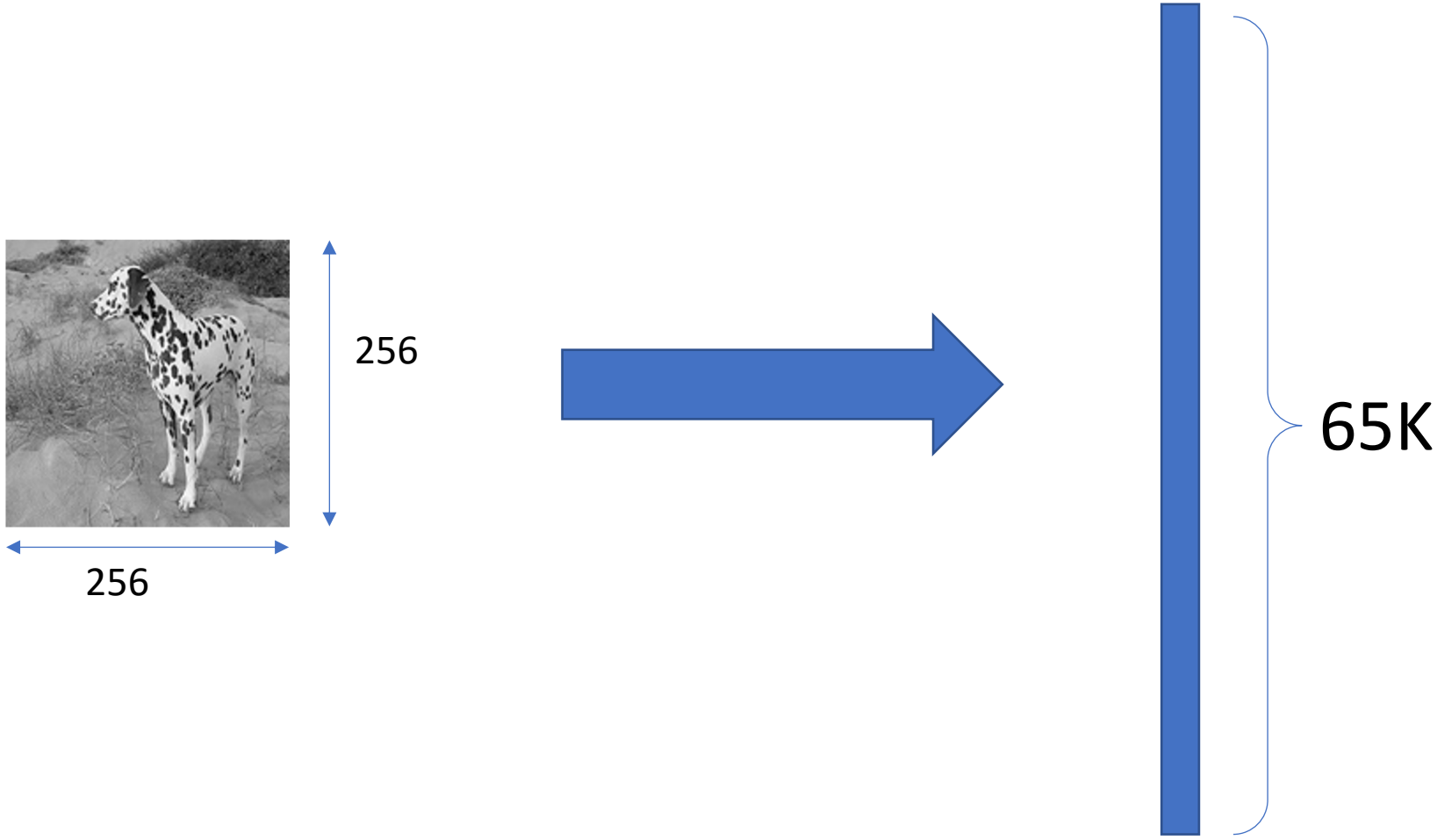
Multilayer perceptrons

Linear	ReLU	Linear	ReLU	Linear
$f_1(x) = W_1x$	$g(x) = \max(x,0)$	$f_2(x) = W_2x$	$g(x) = \max(x,0)$	$f_3(x) = W_3x$

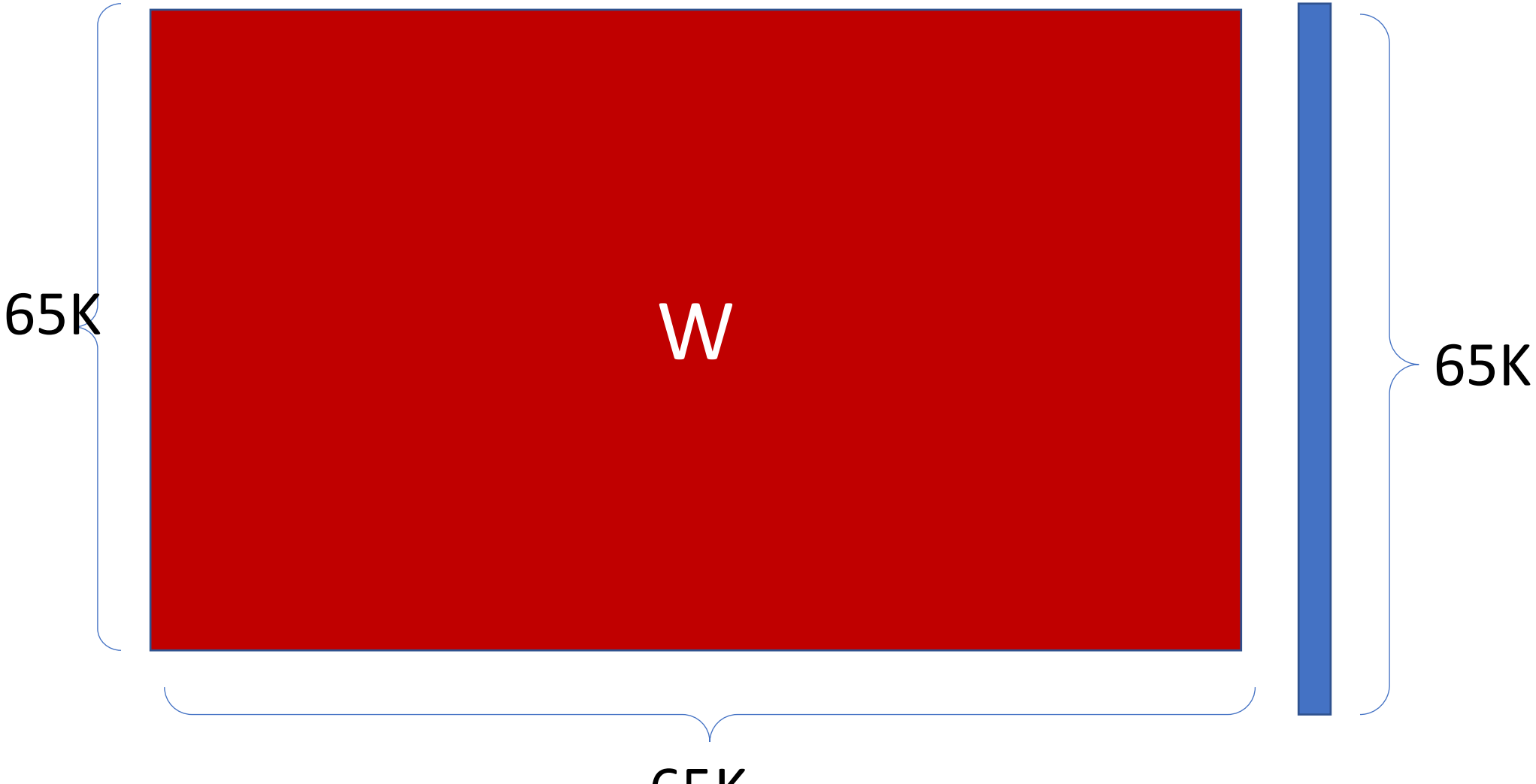


Example function

Reducing capacity

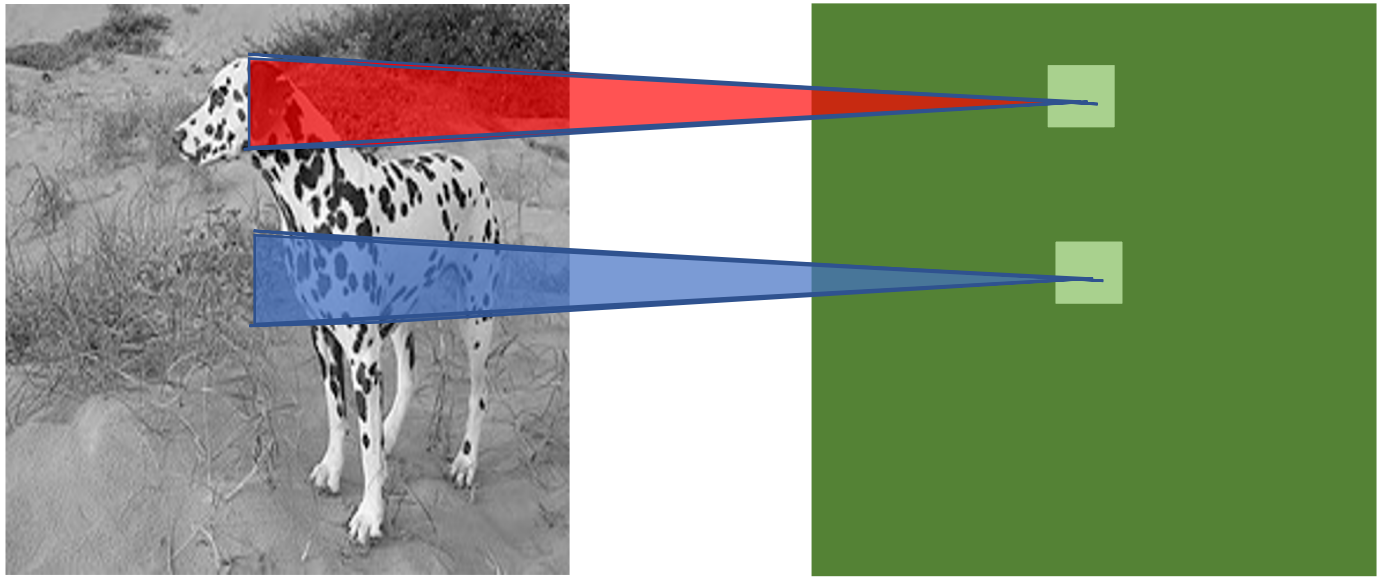


Reducing capacity



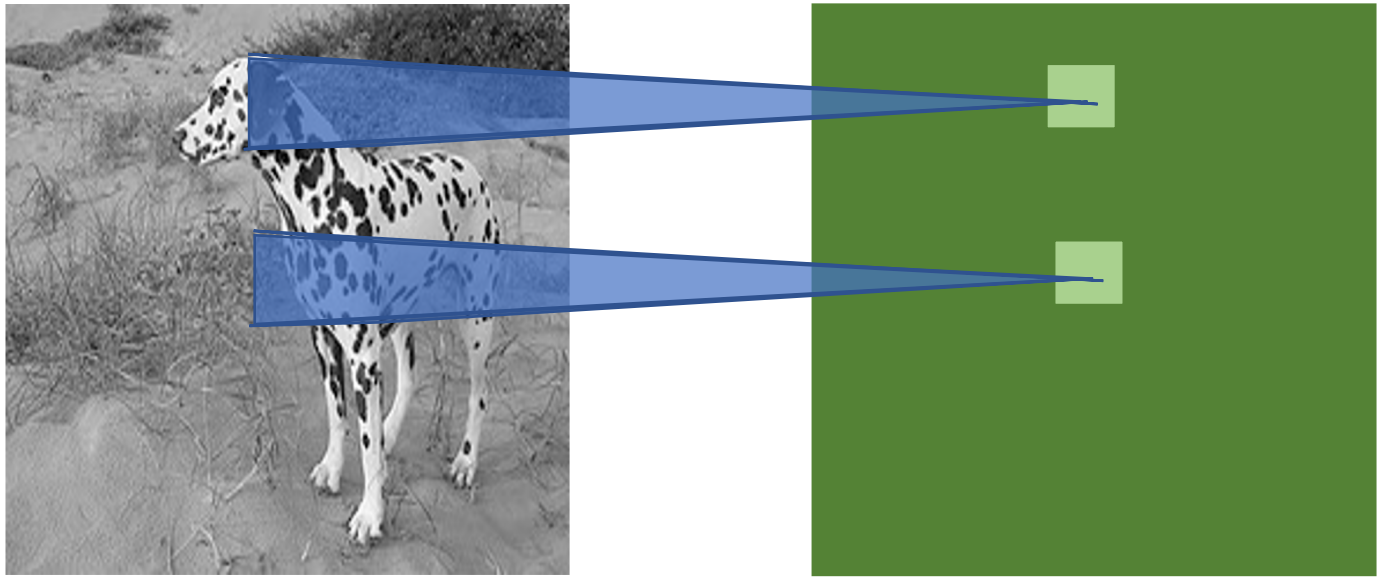
Idea 1: local connectivity

- Inputs and outputs are *feature maps*
- Pixels only related to nearby pixels



Idea 2: Translation invariance

- Pixels only related to nearby pixels

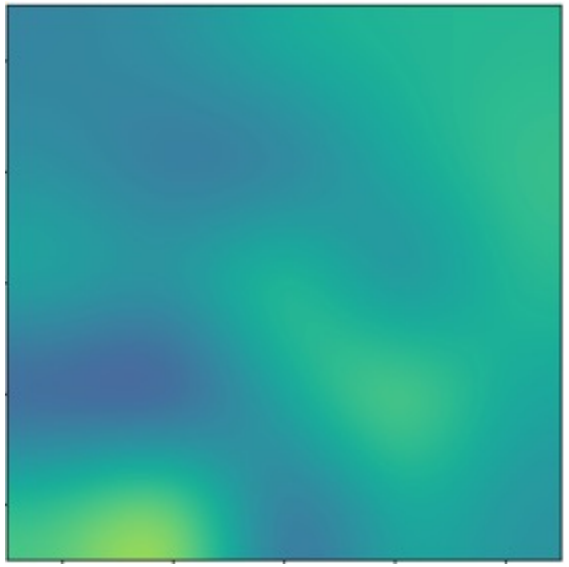


Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

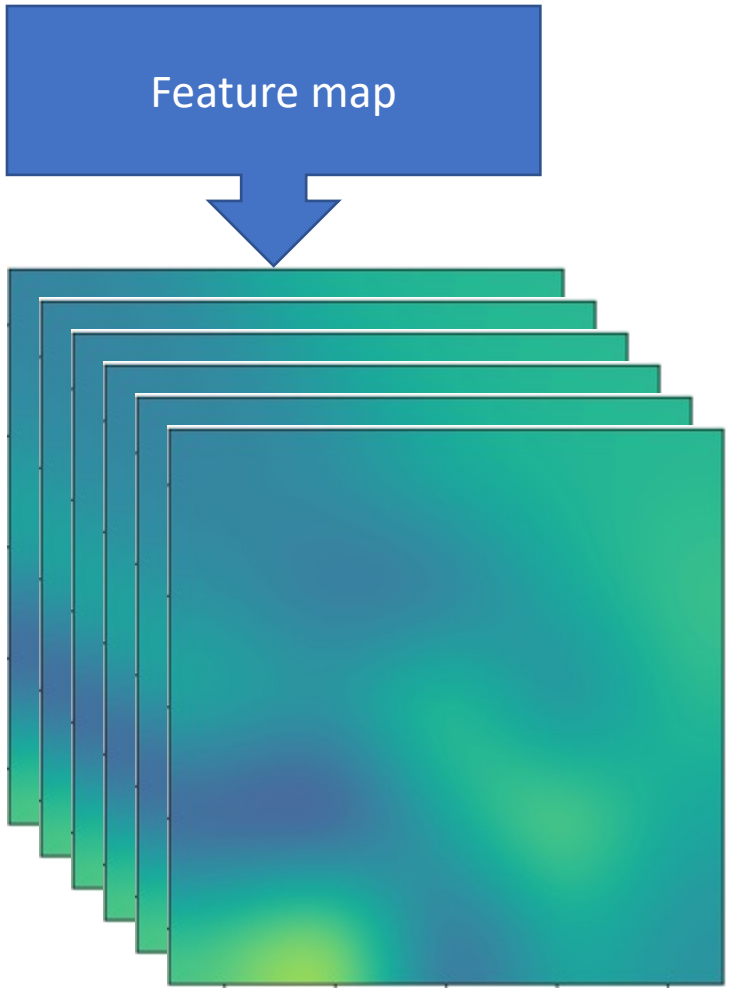
Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

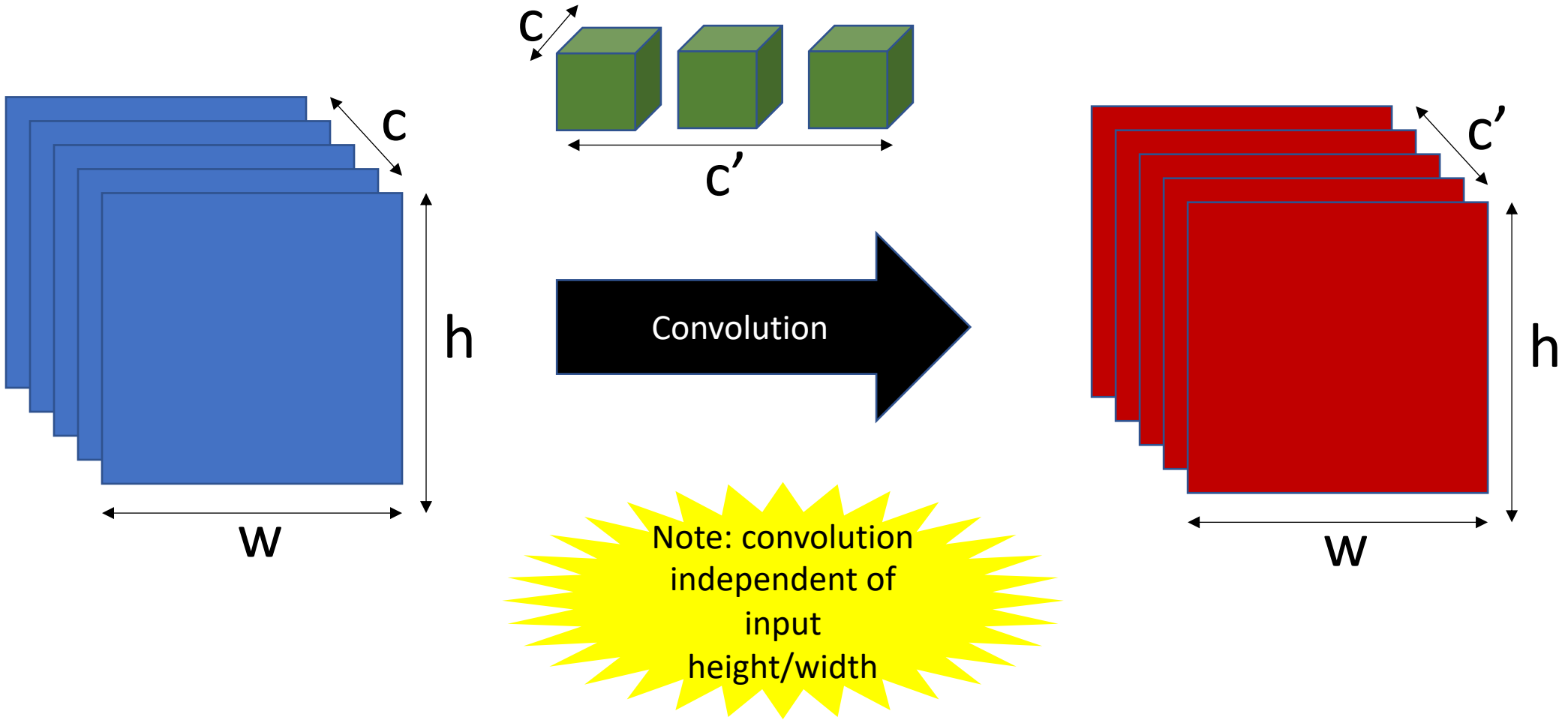


Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2



Convolution as a primitive

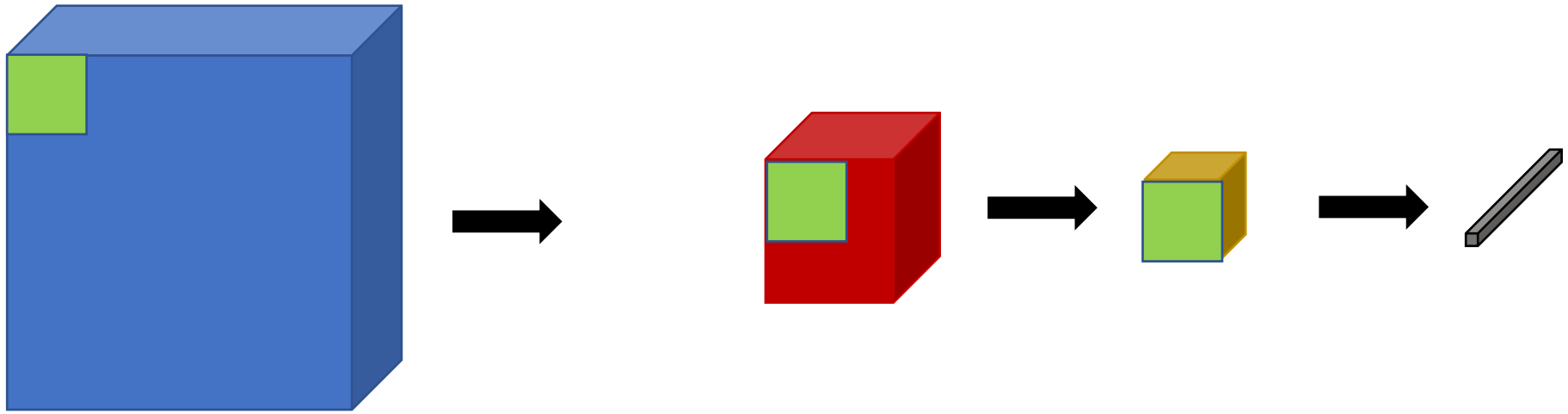


Convolution with subsampling

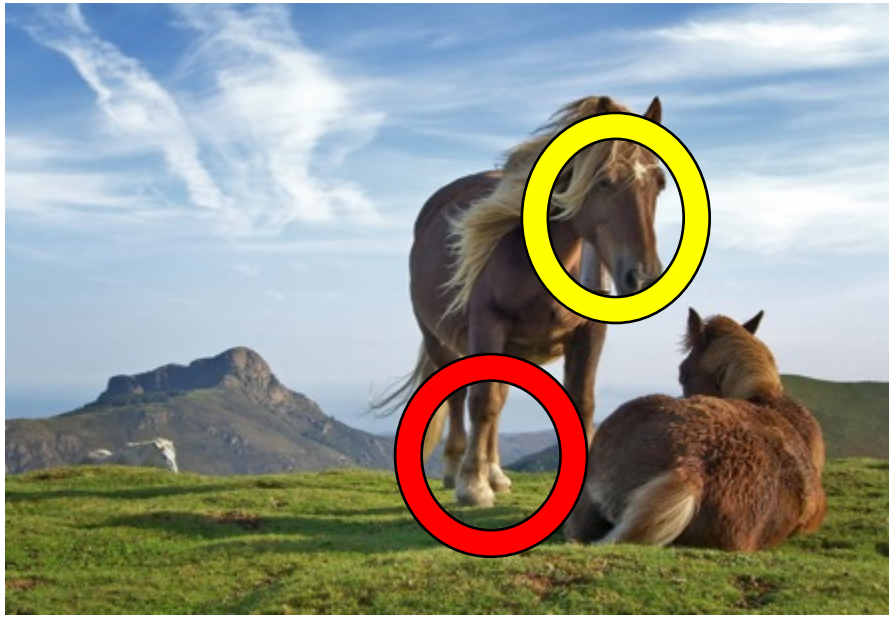
- *Subsampling* = reducing resolution by dropping rows and columns
- Can be done with *strided* convolution
  - Stride of k means output pixel every k input pixels
- Typically done *without anti-aliasing,* though *anti-aliasing helps*[1]

[1]https://richzhang.github.io/antialiased-cnns/

Convolution with subsampling



Invariance to deformations

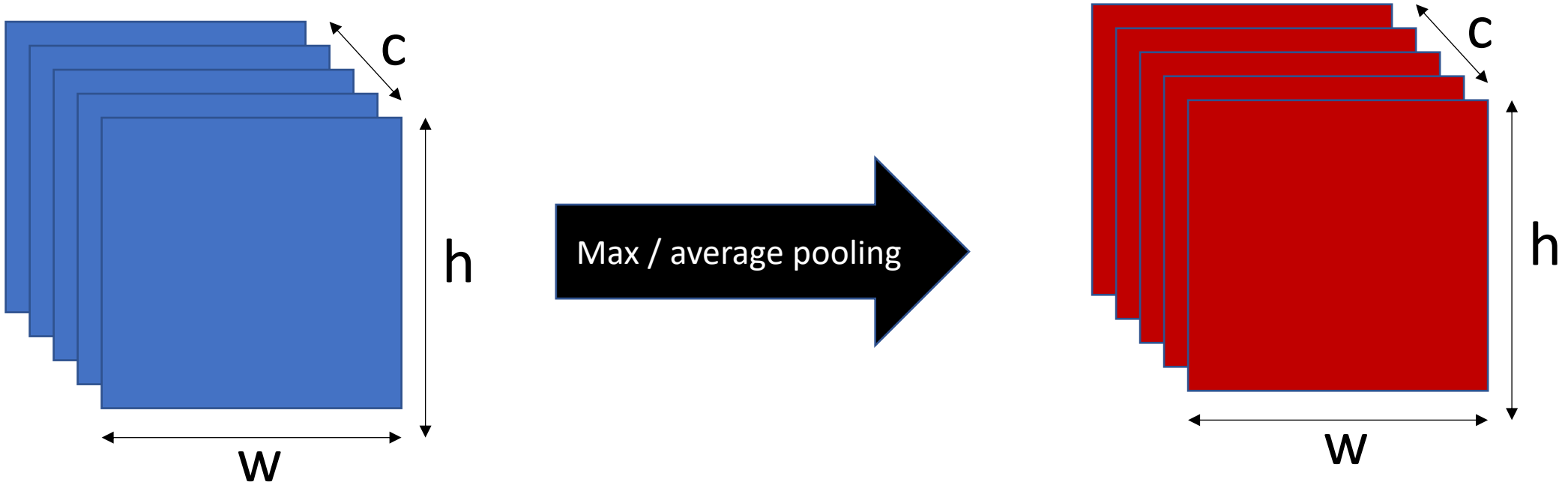


Effect of subsampling

- Same sized filters captures larger neighborhoods on lower resolution features
- Magnitude of translations / deformations reduce with lower resolution
- Convolution in earlier steps detects *more local* patterns *less resilient* to deformations / translations
- Convolution in later steps detects *more global* patterns *more resilient* to deformations / translations
- Subsampling allows capture of *larger, more invariant* patterns

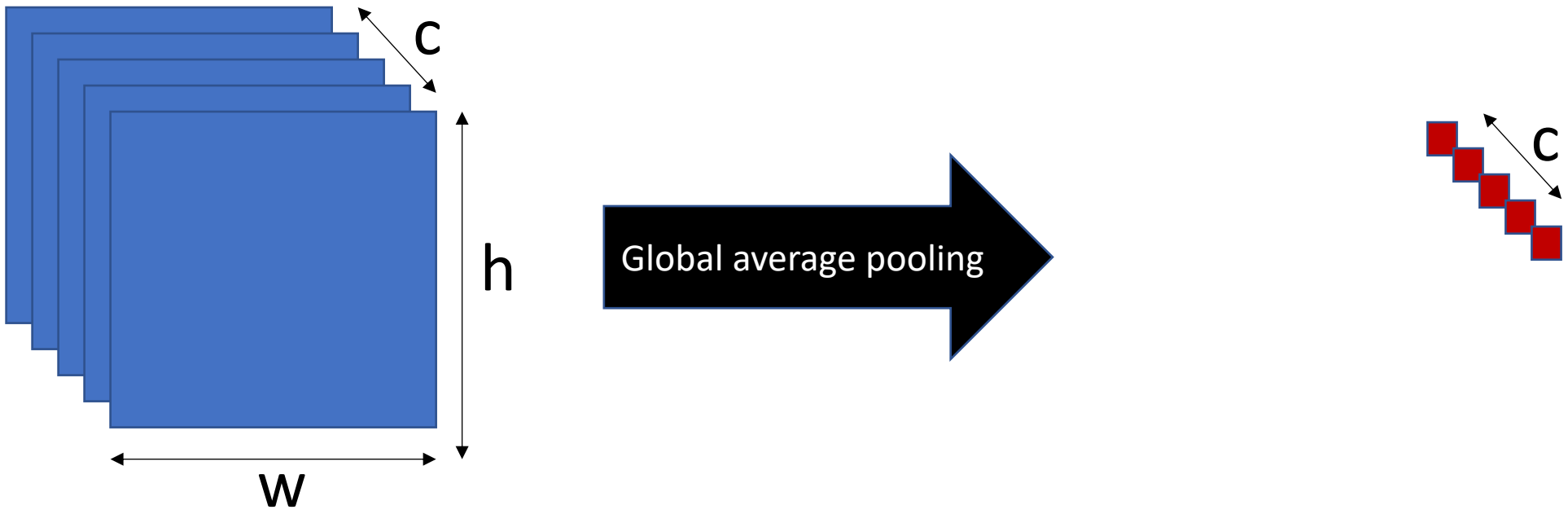
Pooling

- Similar to convolution, but take *max* or *average* across window for every channel
- No learnable parameters



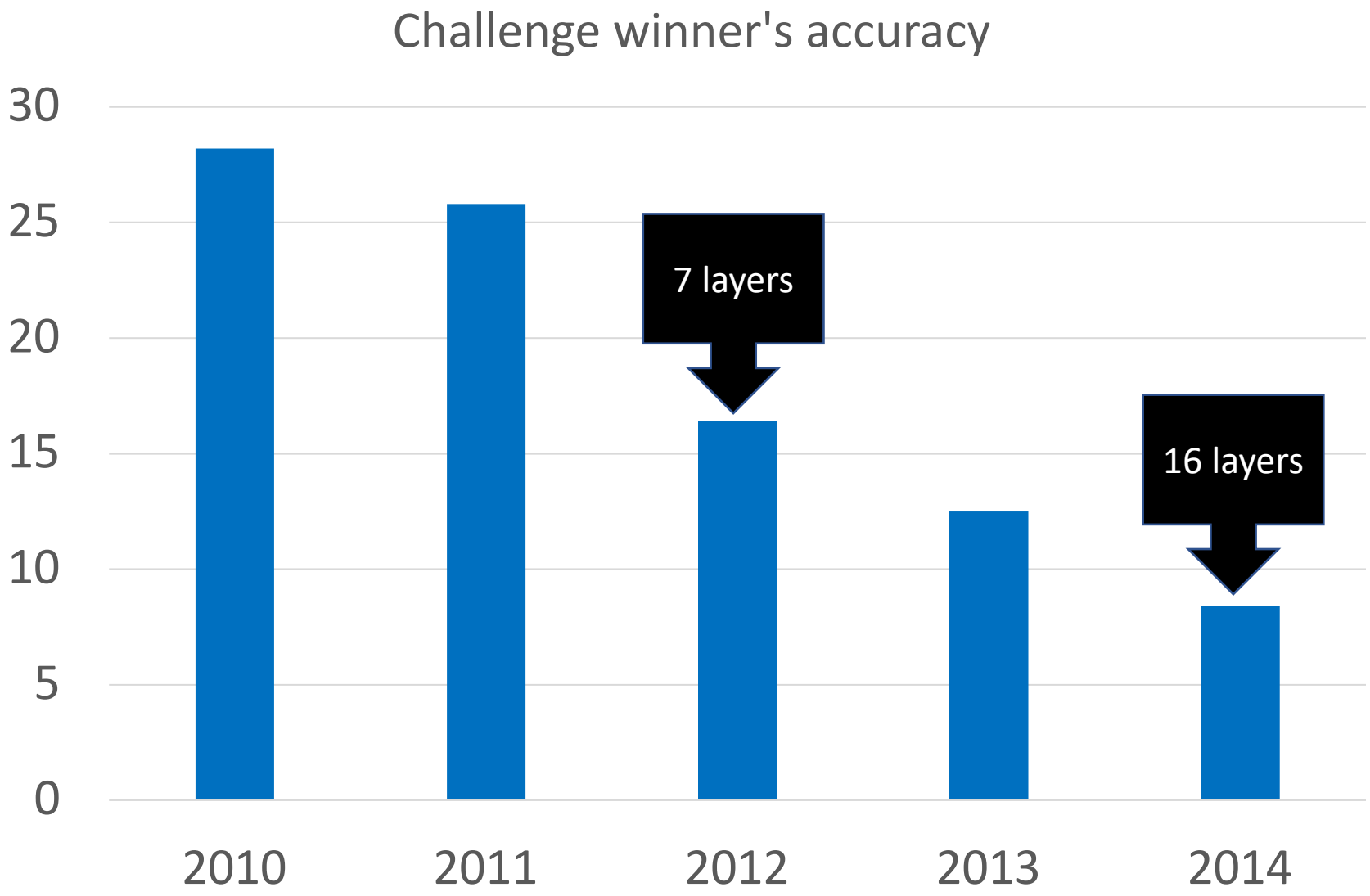
Global Average Pooling

- Special case: take average across entire input space for every channel
- Useful for converting feature maps to vector of image features

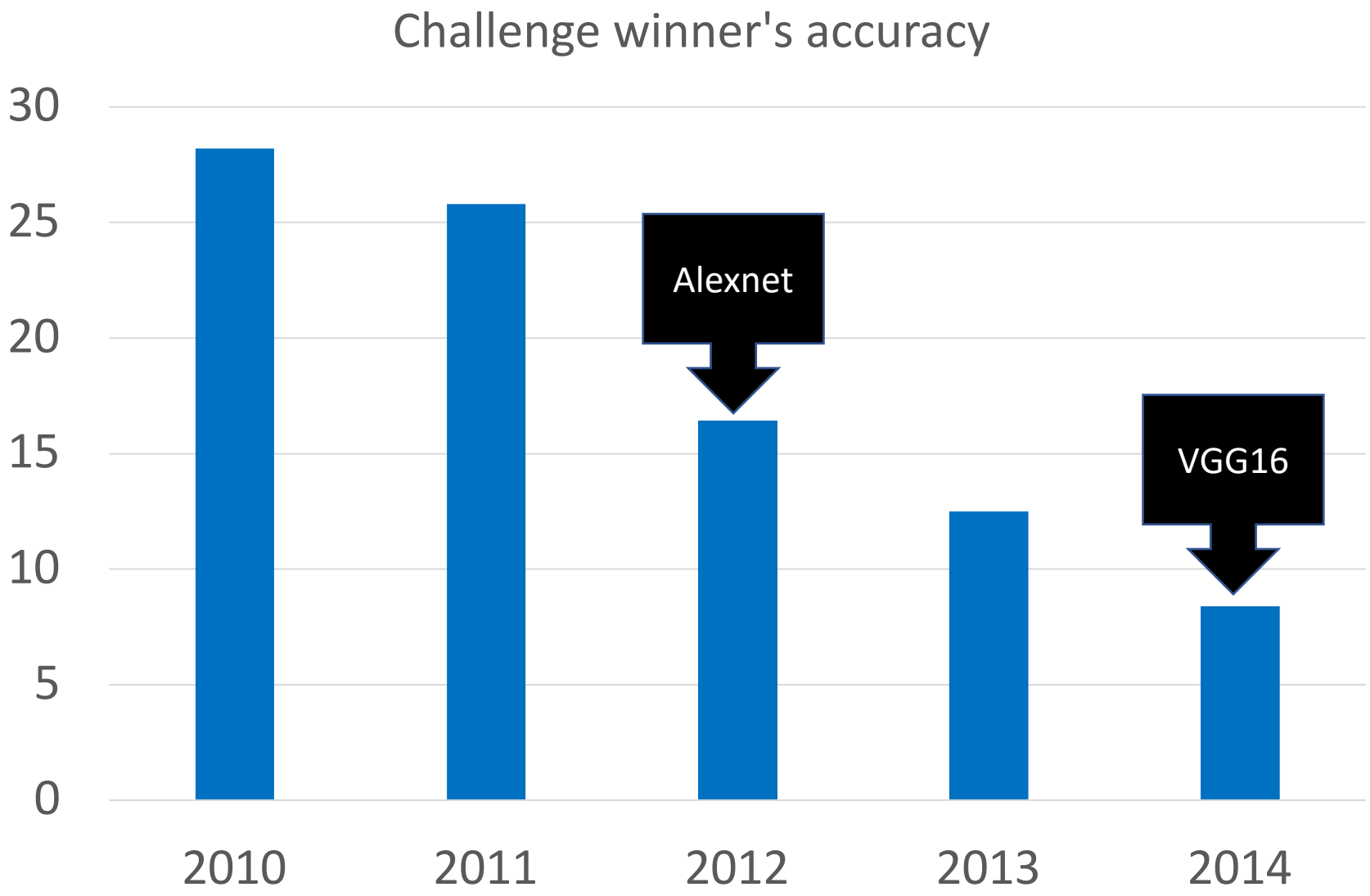


Exploring convnet architectures

Deeper is better



Deeper is better



The VGG pattern

- Every convolution is 3x3, padded by 1
- Every convolution followed by ReLU
- ConvNet is divided into “stages”
  - Layers within a stage: no subsampling
  - Subsampling by 2 at the end of each stage
- Layers within stage have same number of channels
- Every subsampling → double the number of channels

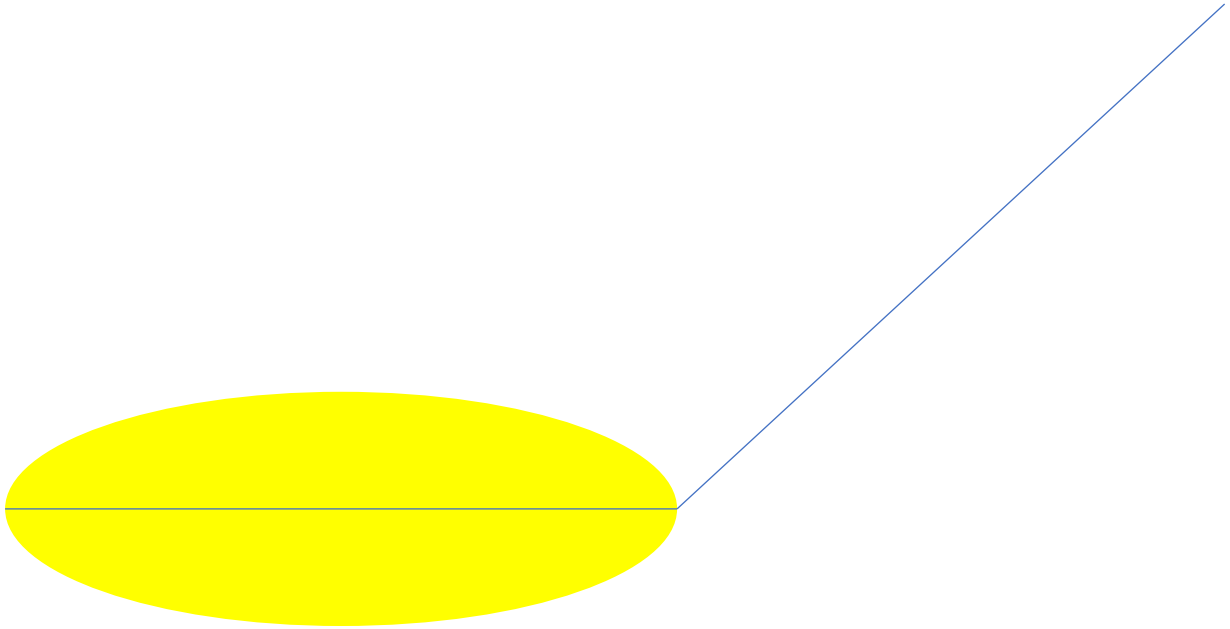
Challenges in training: exploding / vanishing gradients

- Vanishing / exploding gradients

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}$$

  - If each term is (much) greater than 1 → *explosion of gradients*
  - If each term is (much) less than 1 → *vanishing gradients*

Challenges in training: dependence on init



Solutions

- Careful init
- Batch normalization
- Residual connections

Careful initialization

- Key idea: want variance to remain approx. constant
  - Variance increases in backward pass => exploding gradient
  - Variance decreases in backward pass => vanishing gradient
- “MSRA initialization”
  - weights = Gaussian with 0 mean and variance = 2/(k*k*d)

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. K. He, X. Zhang, S. Ren, J. Sun

Batch normalization

- Key idea: normalize so that each layer output has zero mean and unit variance
  - Compute mean and variance for each channel
  - Aggregate over batch
  - Subtract mean, divide by std
- Need to reconcile train and test
  - No ”batches” during test
  - After training, compute means and variances on train set and store

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

Residual connections

- In general, gradients tend to vanish
- Key idea: allow gradients to flow unimpeded

$$z_{i+1} = f_{i+1}(z_i, w_{i+1})$$

$$\frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial f_{i+1}(z_i, w_{i+1})}{\partial z_i}$$

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}$$

Residual connections

- In general, gradients tend to vanish
- Key idea: allow gradients to flow unimpeded

$$z_{i+1} = g_{i+1}(z_i, w_{i+1}) + z_i \qquad \frac{\partial z_{i+1}}{\partial z_i} = \frac{\partial g_{i+1}(z_i, w_{i+1})}{\partial z_i} + I$$

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}$$

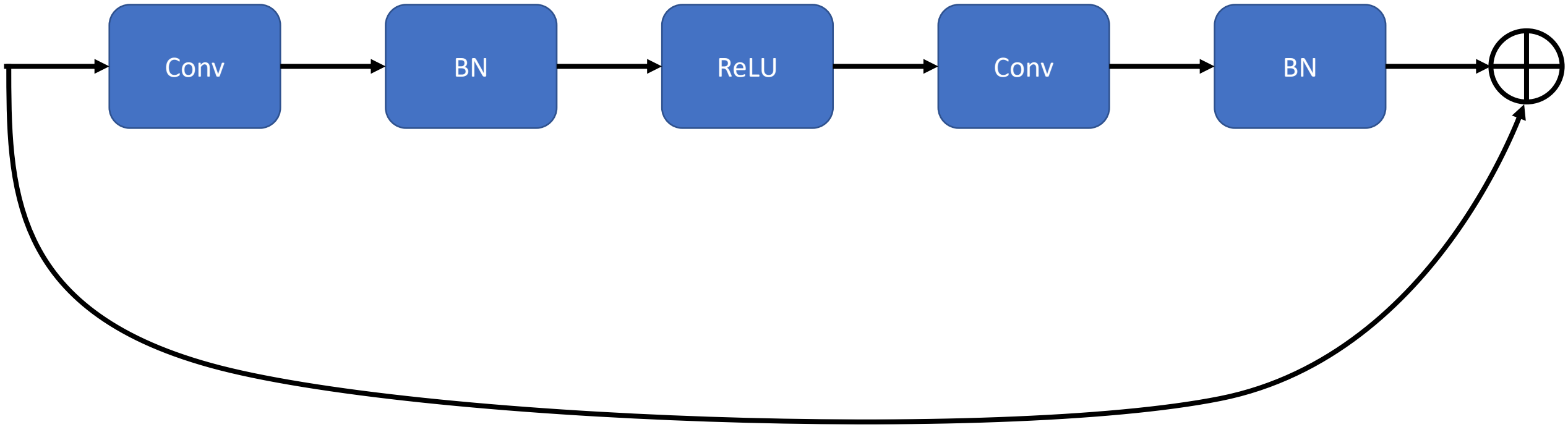
Residual connections

- Assumes all z_i have the same size
- True within a stage
- Across stages?
  - Doubling of feature channels
  - Subsampling
- Increase channels by 1x1 convolution
- Decrease spatial resolution by subsampling

$$z_{i+1} = g_{i+1}(z_i, w_{i+1}) + \text{subsample}(W z_i)$$

A residual block

- Instead of single layers, have residual connections over block



Bottleneck blocks

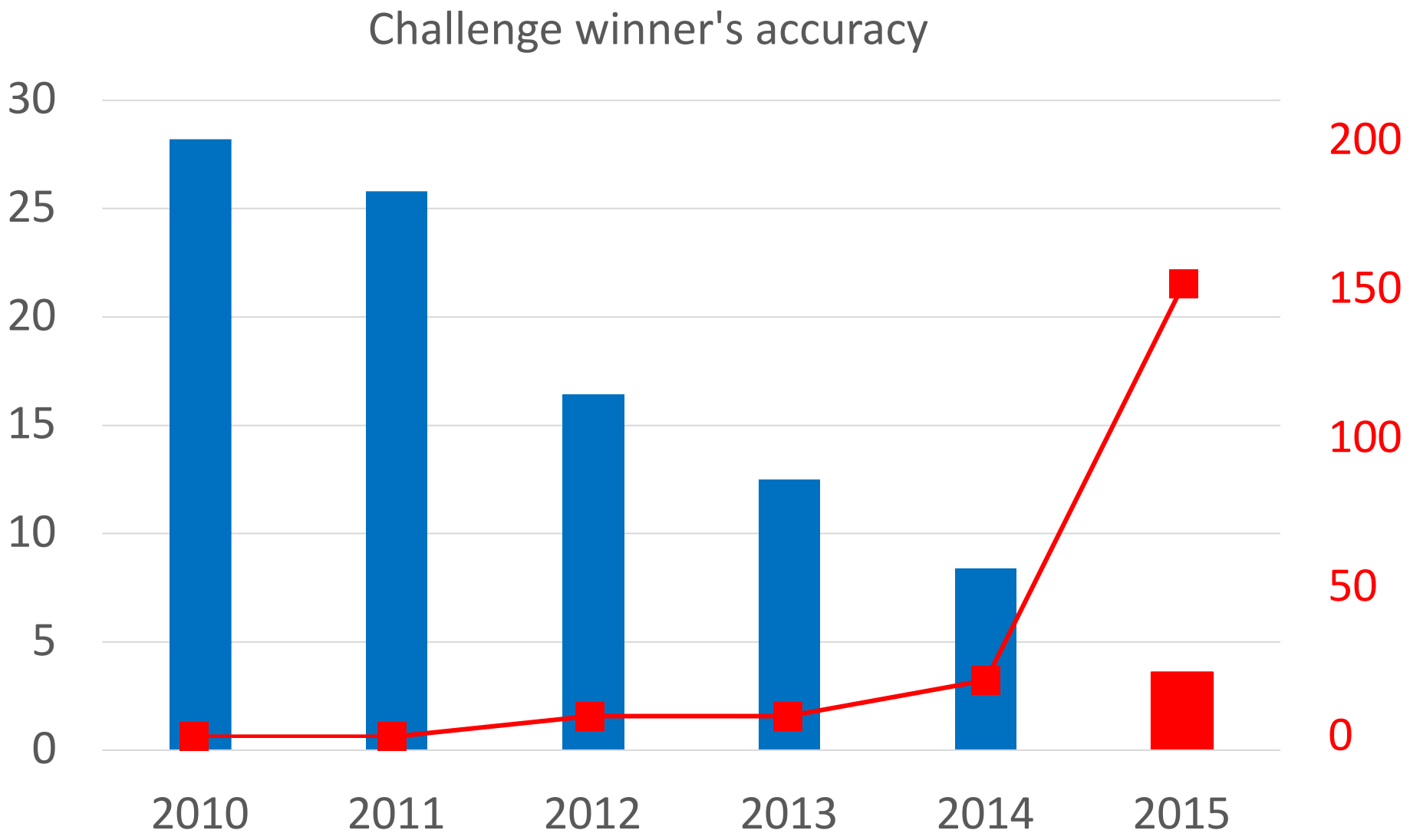
- Problem: When channels increases, 3x3 convolutions introduce many parameters
  - $3 \times 3 \times c^2$

- Key idea: use 1x1 to project to lower dimensionality, do convolution, then come back
  - $c \times d + 3 \times 3 \times d^2 + d \times c$

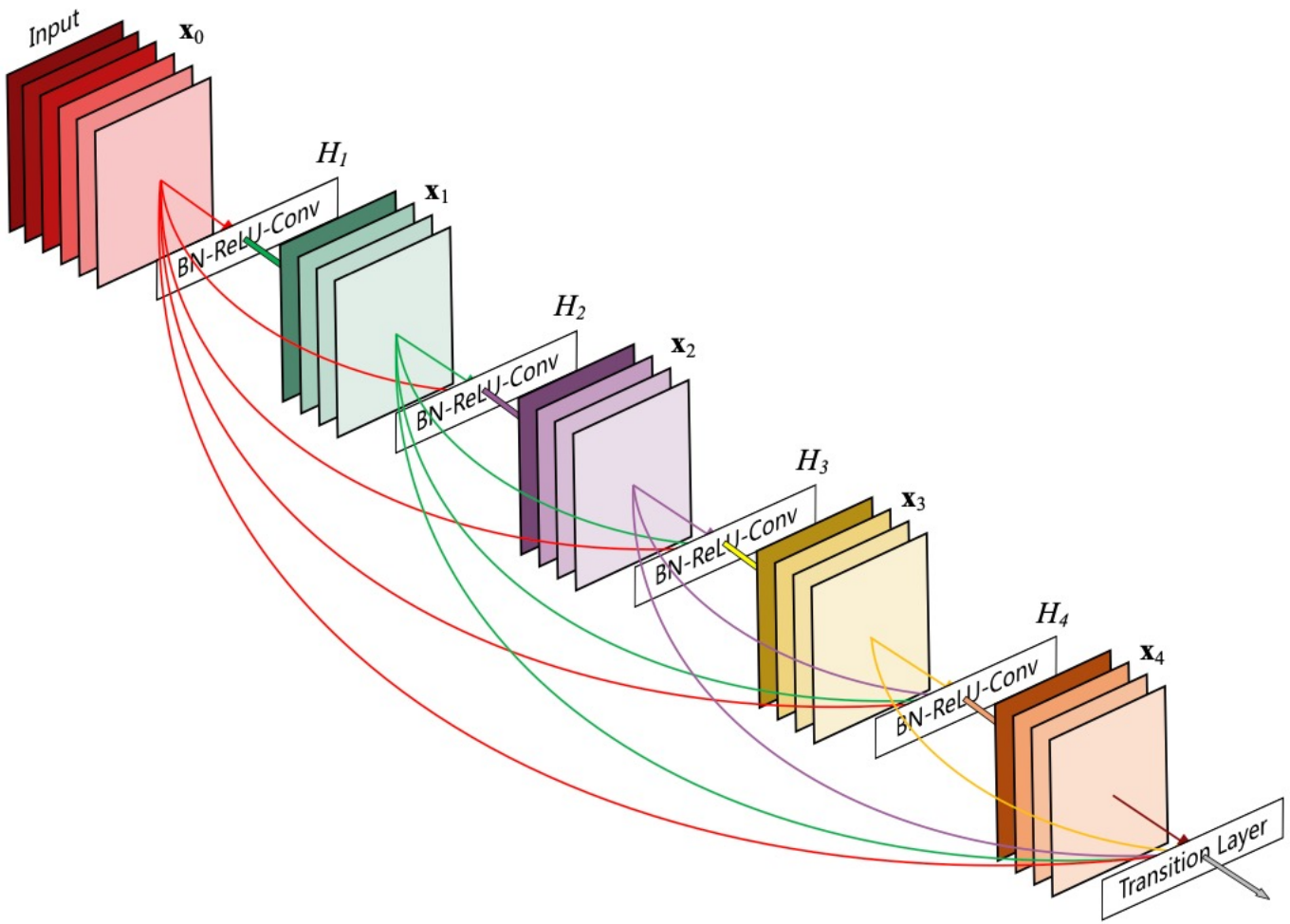
The ResNet pattern

- Decrease resolution substantially in first layer
  - Reduces memory consumption due to intermediate outputs
- Divide into stages
  - maintain resolution, channels in each stage
  - halve resolution, double channels between stages
- Divide each stage into residual blocks
- At the end, compute average value of each channel to feed linear classifier

Putting it all together - Residual networks



DenseNets


Input
$\mathbf{x}_0$
H_1
BN-ReLU-Conv
$\mathbf{x}_1$
H_2
BN-ReLU-Conv
$\mathbf{x}_2$
H_3
BN-ReLU-Conv
$\mathbf{x}_3$
H_4
BN-ReLU-Conv
$\mathbf{x}_4$
Transition Layer