Image classification

Image classification

- Given an image, produce a label
- Label can be:
  - 0/1 or yes/no: *Binary classification*
  - one-of-k: *Multiclass classification*
  - 0/1 for each of k concepts: *Multilabel classification*

MNIST

- 2D
- 10 classes
- 6000 examples per class

1990’s

Caltech 101

- 101 classes
- 10 classes
- 30 examples per class
- Strong category-specific biases
- Clean images

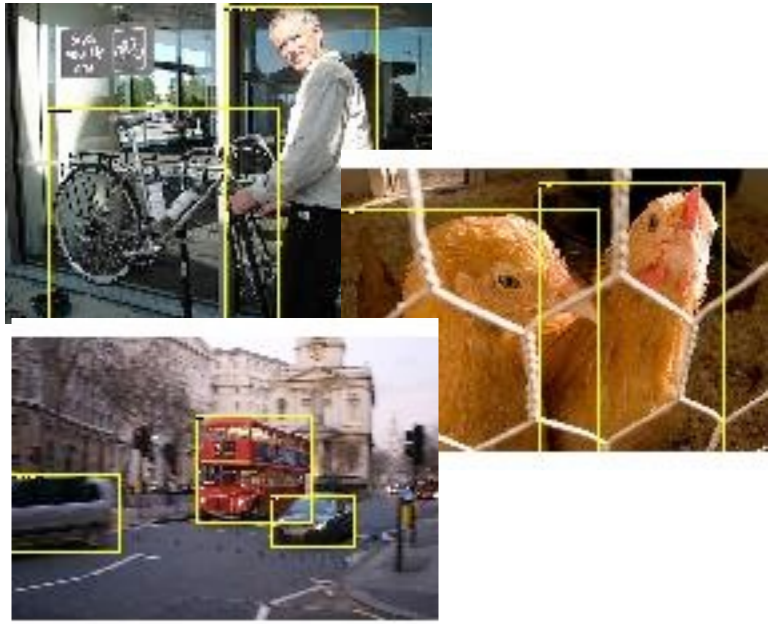
MNIST

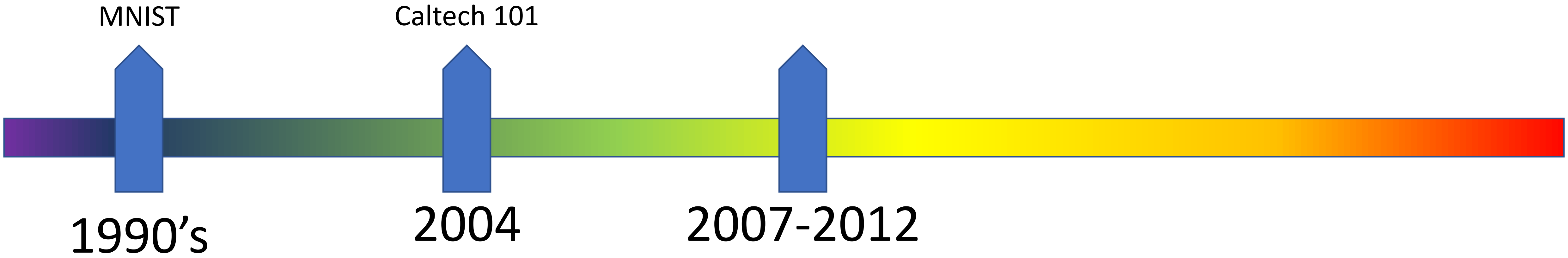
1990’s

2004

PASCAL VOC

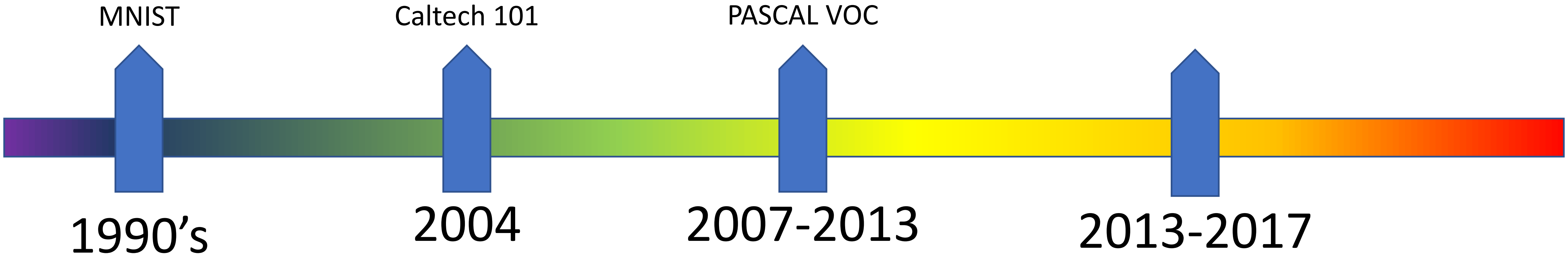
- 20 classes
- ~500 examples per class
- Clutter, occlusion, natural scenes



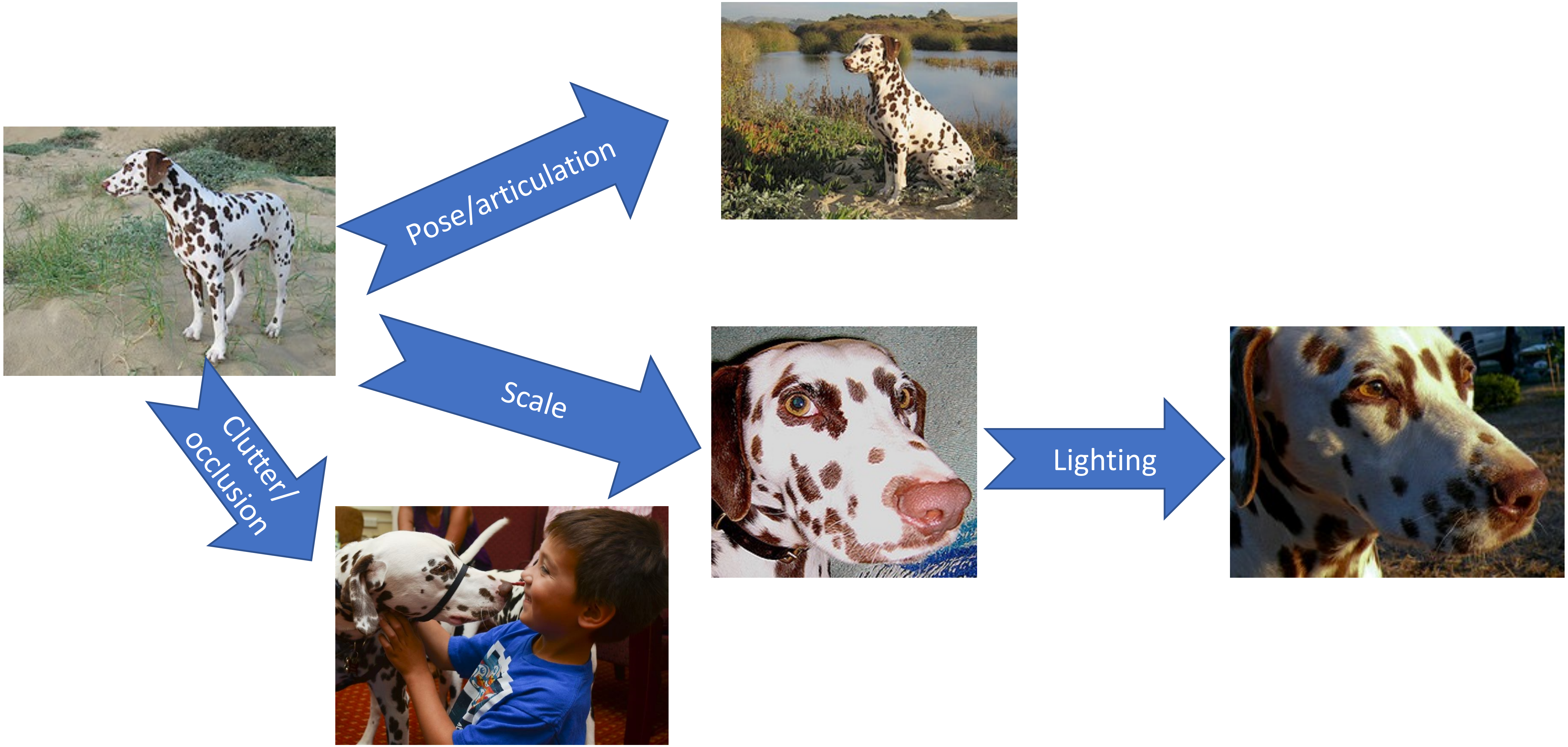


ImageNet

- 1000 classes
- ~1000 examples per class
- Mix of cluttered and clean images



Why is recognition hard?



Learning

- Key idea: teach computer visual concepts by *providing examples*

$\mathcal{X}$:Images

$\mathcal{Y}$:Labels

$\mathcal{D}$:Distribution over $\mathcal{X} \times \mathcal{Y}$

Training Set

$$S = \{(x_i, y_i) \sim \mathcal{D}, i = 1, \ldots, n\}$$

Example

- Binary classifier “Dog” or ”not Dog”
- Labels: {0, 1}
- Training set

{( , 1), ( , 1), ( , 0) , … }

Learning

- Key idea: teach computer visual concepts by *providing examples*

$$S = \{(x_i, y_i) \sim \mathcal{D}, i = 1, \dots, n\}$$

- Want to be able to estimate label y for *new images* x
  - Want to give score $s(y, x)$ for each possible label y, then pick highest scoring
  - Want to estimate $y(x)$
  - Want to estimate $P(y|x)$, then pick most likely

Choosing a model class

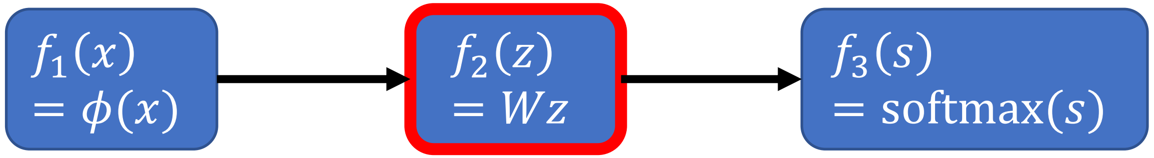
- Will estimate a probability P(y | x)
- Any function that takes x as input and outputs probability distribution
  - $h : \mathcal{X} \to C^{|\mathcal{Y}|}$ where C^d is a probability distribution over d classes
  - Very large set of possibilities for h
- Constrain choice: Choose a family of possible functions H
  - Hypothesis class

Hypothesis class I: Classical models

- Choose h to be a linear classifier over some feature space
- First extract features: $\boldsymbol{z} = \phi(x)$
  - ϕ is a fixed, hand-crafted function that converts images into features useful for recognition: $\phi: \mathcal{X} \to \mathbb{R}^d$
- Next multiply by a weight matrix to produce class scores: $\boldsymbol{s} = W\boldsymbol{z}$
  - W is unknown a priori
- Next normalize scores to a probability
  - $P(y = k|x) \propto e^{s_k}$
  - “Softmax”

Hypothesis class I: Classical models

- $h(x; W) = \text{softmax}(W\phi(x))$
- For different settings of W, get different hypotheses
- Hypothesis class $H = \{h(\cdot; W); W \in \mathbb{R}^{|\mathcal{Y}| \times d}\}$
- W are *parameters:* index hypotheses in hypothesis class

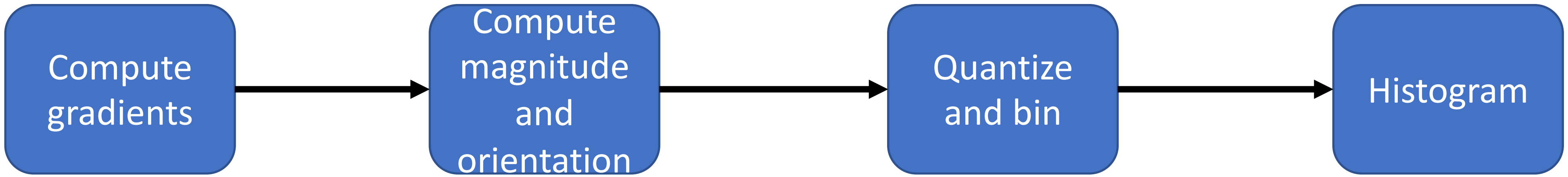


Choice of feature extractor?

- SIFT, HOG, GIST, BOW….
- The rest of the pipeline is very simple: linear function + softmax
- So heavy lifting must be done by feature extractor
- But how do we design feature extractor?

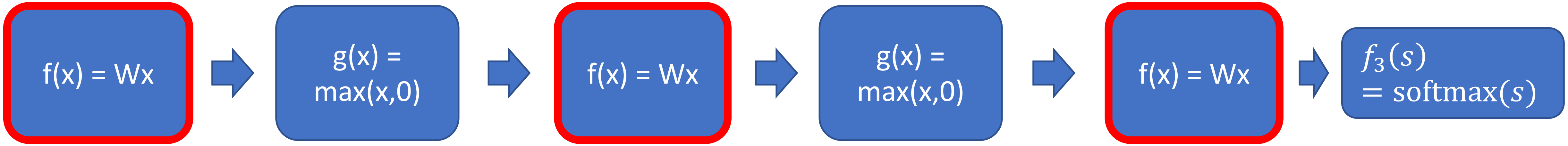
SIFT

- SIFT itself a series of simple, fixed steps
- Make some of them parametric?



Hypothesis class 2: Multilayer perceptrons

- Key idea: build complex functions by composing *many* simple functions



General recipe

- Fix hypothesis class
  - $h_w(x) = \text{softmax}\left(f_3(\,f_2\,(g(\,f_1(\mathrm{x}, \mathrm{w}_1)), \mathrm{w}_2), \mathrm{w}_3)\right)$
  - $h_w(x) = \text{softmax}\left(W\phi(x)\right)$
- Define loss function
  - $L(h_w(x_i), y_i) = -\log p_{y_i}(x_i)$
- Minimize average (or total) loss on the training set

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} L(h_{\mathbf{w}}(x_i), y_i)$$

- *How do we minimize?*
- *Why should this work?*

Training: Choosing the best hypothesis

- Need to minimize an objective function.
- In general, optimization problem.
- If L is differentiable and h is differentiable: can do gradient descent

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} L(h_{\mathbf{w}}(x_i), y_i)$$

Training = Optimization

- Simple solution: *gradient descent*

$$\min_{\mathbf{w}} f(\mathbf{w})$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^{(t)})$$

Stochastic gradient descent

$$f(\mathbf{w}) = \frac{1}{n}\sum_i L(h_{\mathbf{w}}(x_i), y_i)$$

Objective function

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{n}\sum_i \nabla_{\mathbf{w}} L(h_{\mathbf{w}}(x_i), y_i)$$

Gradient

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = < \nabla_{\mathbf{w}} L(h_{\mathbf{w}}(x_i), y_i) >$$

Gradient = average of per example gradients

$$\nabla_{\mathbf{w}} f(\mathbf{w}) \approx \nabla_{\mathbf{w}} L(h_{\mathbf{w}}(x_i), y_i)$$

Stochastic gradient descent using single examples

$$\nabla_{\mathbf{w}} f(\mathbf{w}) \approx \frac{1}{|B|}\sum_{k=1}^{|B|} \nabla_{\mathbf{w}} L(h_{\mathbf{w}}(x_{i_k}), y_{i_k})$$

Stochastic gradient descent using minibatch

Stochastic gradient descent

- Randomly sample small subset of examples
- Compute gradient on small subset
  - *Unbiased estimate of true gradient*
- Take step along estimated gradient

Computing derivatives

$$\nabla_{\mathbf{w}} f(\mathbf{w}) \approx \nabla_{\mathbf{w}} L(h_{\mathbf{w}}(x_i), y_i)$$

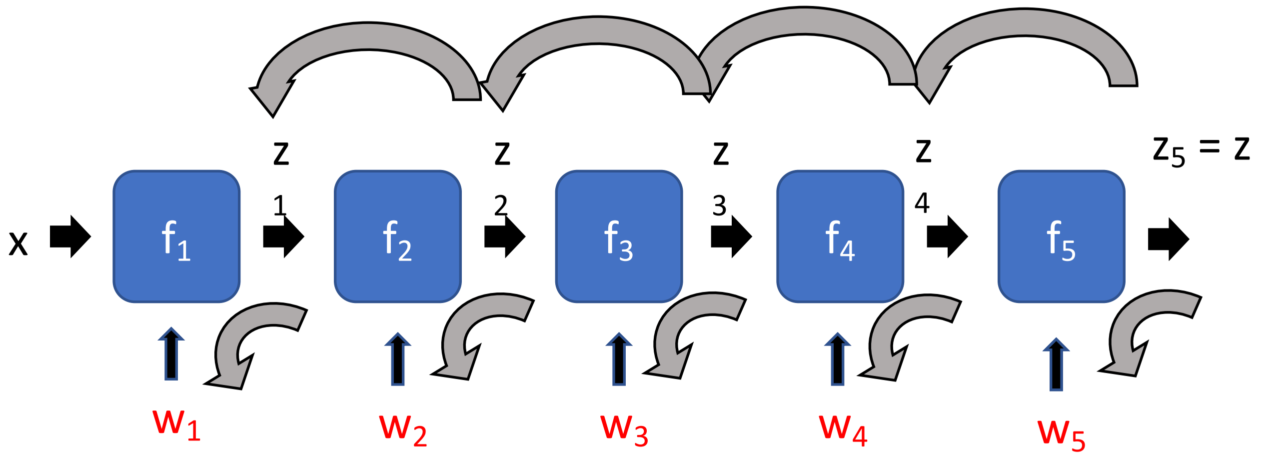
- How do we compute gradient?
- Composition of functions: use chain rule

$$z_1 = f_1(x, \mathbf{w}_1)$$
$$z_2 = f_2(z_1, \mathbf{w}_2)$$
$$z_3 = f_3(z_2, \mathbf{w}_3)$$
$$l = L(z_3, y)$$

$$g_1 = \frac{\partial l}{\partial z_1} = g_2 \frac{\partial z_2}{\partial z_1}$$
$$g_2 = \frac{\partial l}{\partial z_2} = g_3 \frac{\partial z_3}{\partial z_2}$$
$$g_3 = \frac{\partial l}{\partial z_3}$$

$$\frac{\partial l}{\partial \mathbf{w}_1} = g_1 \frac{\partial z_1}{\partial \mathbf{w}_1}$$
$$\frac{\partial l}{\partial \mathbf{w}_2} = g_2 \frac{\partial z_2}{\partial \mathbf{w}_2}$$
$$\frac{\partial l}{\partial \mathbf{w}_3} = g_3 \frac{\partial z_3}{\partial \mathbf{w}_3}$$

The gradient of convnets



Backpropagation

Risk

- Given:
  - Distribution $\mathcal{D}$
  - A hypothesis $h \in H$
  - Loss function L
- We are interested in Expected Risk:

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y)$$

- Given training set S, and a particular hypothesis h, Empirical Risk:

$$\hat{R}(S, h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

Risk

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

- By central limit theorem,

$$\mathbb{E}_{S\sim\mathcal{D}^n} \hat{R}(S,h) = R(h)$$

- Variance proportional to 1/n

- For randomly chosen h, empirical risk is an *unbiased estimator* of expected risk

Risk

- Empirical risk unbiased estimate of expected risk
- Want to minimize expected risk
- Idea: Minimize *empirical risk* instead
- This is the **Empirical Risk Minimization Principle**

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

$$\boxed{h^* = \arg\min_{h \in H} \hat{R}(S,h)}$$

Generalization

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

$$R(h) = \underbrace{\hat{R}(S,h)}_{\text{Training error}} + \underbrace{\left(R(h) - \hat{R}(S,h)\right)}_{\text{Generalization error}}$$

Overfitting

- We are minimizing training error
- Empirical risk of chosen hypothesis *no longer* unbiased estimate:
  - We chose hypothesis based on S
  - Might have chosen h for which S is a special case
- Overfitting:
  - Minimize training error, but generalization error *increases*

Controlling generalization error

- Variance of empirical risk inversely proportional to size of S
  - Choose very large S!

- *Larger* the hypothesis class H, *Higher* the chance of hitting bad hypotheses with low training error and high generalization error
  - Choose small H!

- For many models, can *bound* generalization error using some property of parameters
  - Regularize during optimization!
  - Eg. L2 regularization

Controlling generalization error

- How do we know we are overfitting?
  - Use a *held-out* “validation set”
  - To be an unbiased sample, must be completely *unseen*

Putting it all together

- Want model with least expected risk = expected loss
- But expected risk hard to evaluate
- Empirical Risk Minimization: minimize empirical risk in training set
- Might end up picking special case: overfitting
- Avoid overfitting by:
  - Constructing large training sets
  - Reducing size of model class
  - Regularization

Putting it all together

- Collect training set and validation set
- Pick hypothesis class
- Pick loss function
- Minimize empirical risk (+ regularization)
- Measure performance on held-out validation set
- Profit!

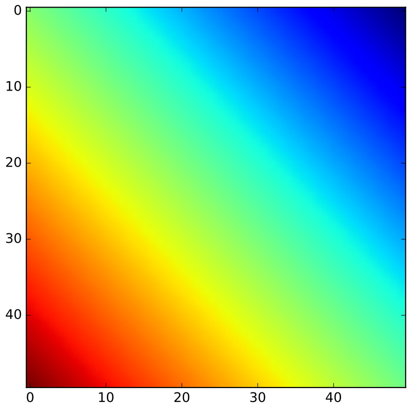
Loss functions and hypothesis classes

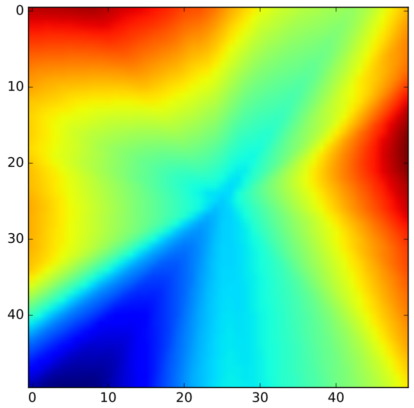
Loss function	Problem	Range of h	$\mathcal{Y}$	Formula
Log loss	Binary Classification	$\mathbb{R}$	$\{0, 1\}$	$\log(1 + e^{-yh(x)})$
Negative log likelihood	Multiclass classification	$[0, 1]^k$	$\{1, \ldots, k\}$	$-\log h_y(x)$
Hinge loss	Binary Classification	$\mathbb{R}$	$\{0, 1\}$	$\max(0, 1 - yh(x))$
MSE	Regression	$\mathbb{R}$	$\mathbb{R}$	$(y - h(x))^2$

Multilayer perceptrons

- Key idea: build complex functions by composing simple functions

f(x) = Wx → g(x) = max(x,0) → f(x) = Wx → g(x) = max(x,0) → f(x) = Wx

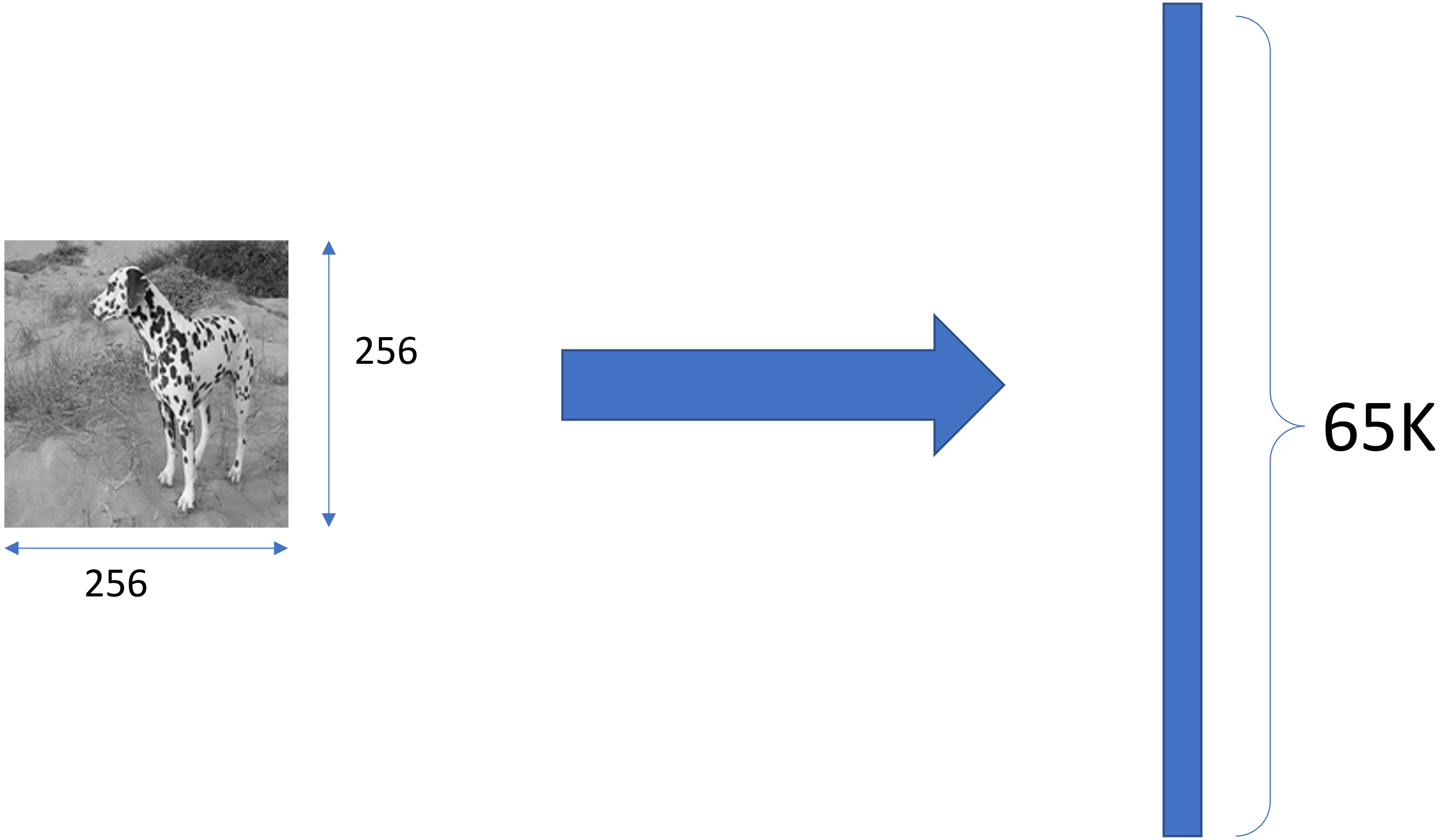




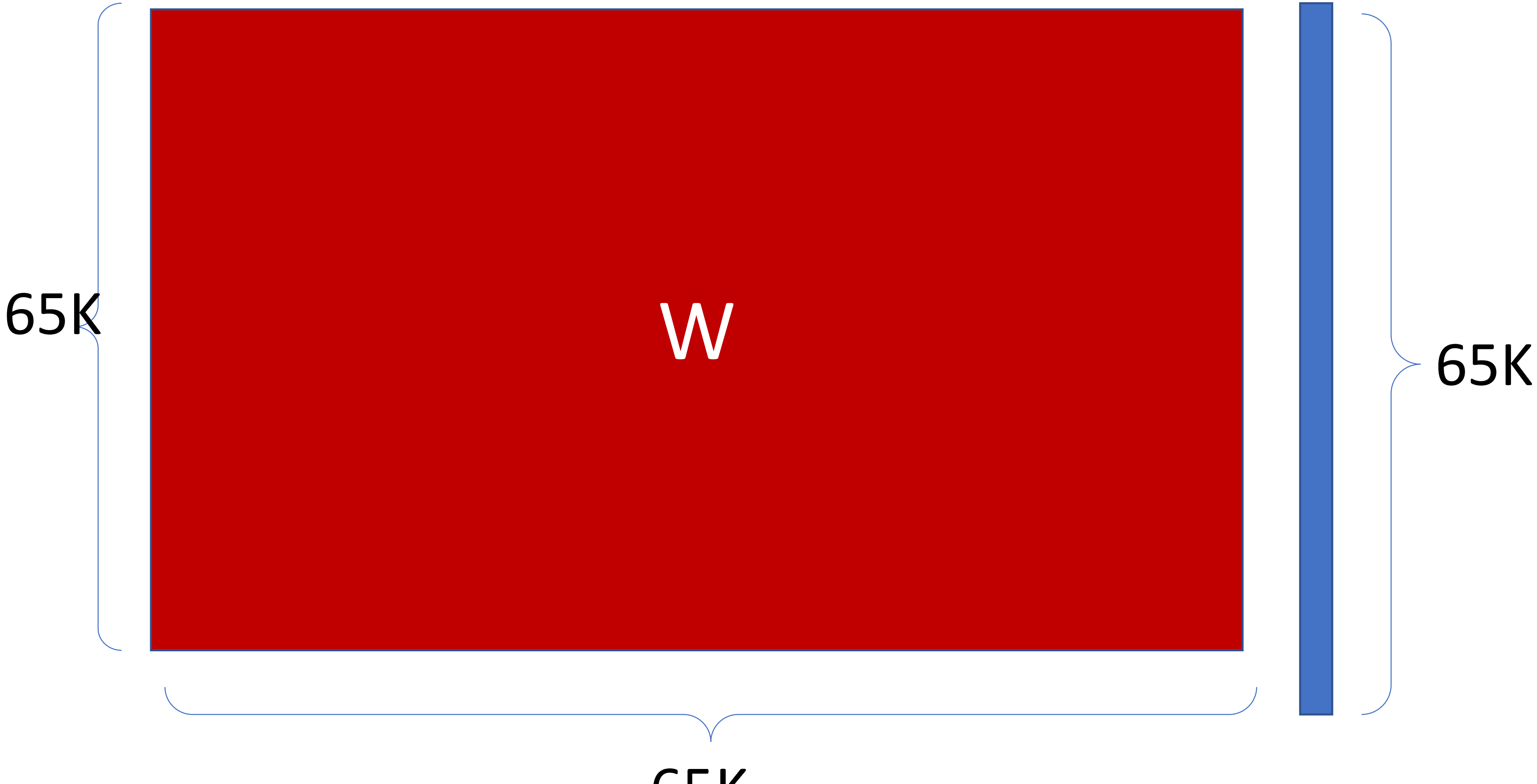
Multilayer perceptrons

- Key idea: build complex functions by composing simple functions
- Caveat: simple functions must include non-linearities
- $W(U(Vx)) = (WUV)x$

Reducing capacity

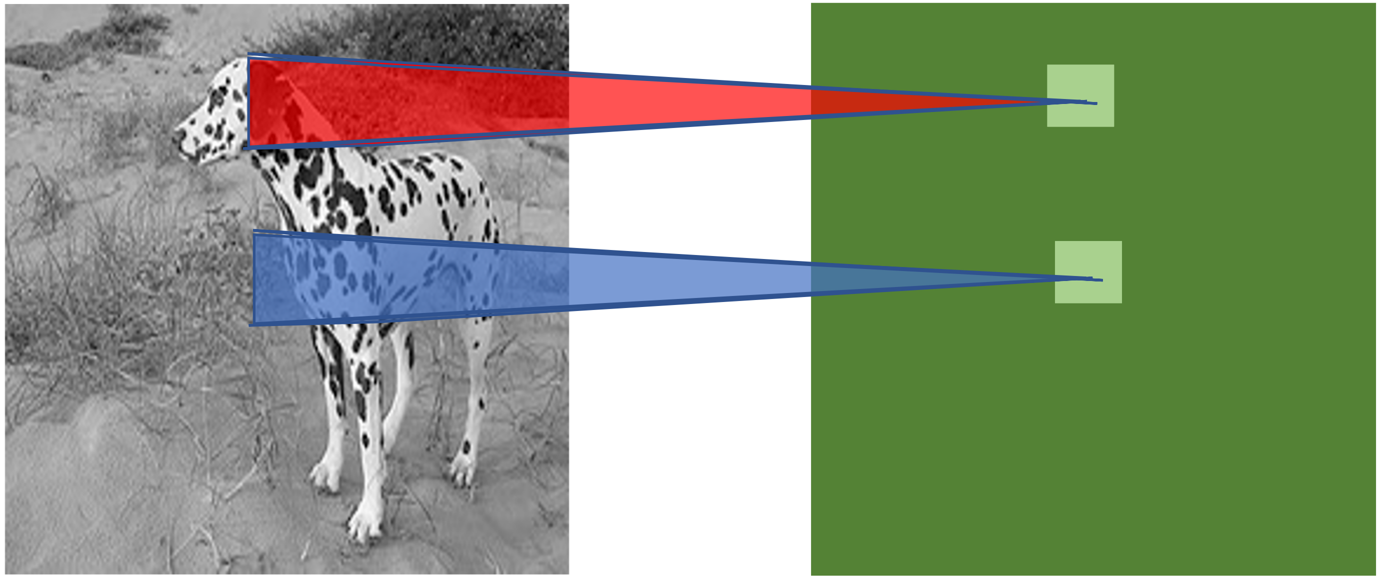


Reducing capacity



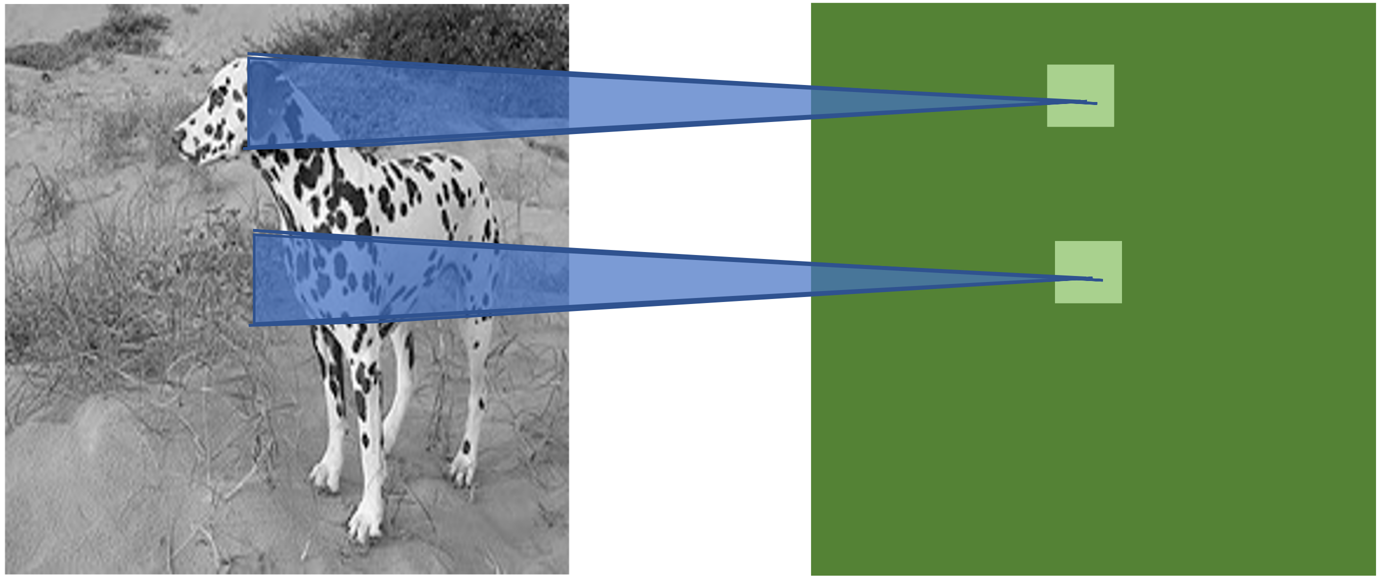
Idea 1: local connectivity

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



Idea 2: Translation invariance

- Pixels only related to nearby pixels

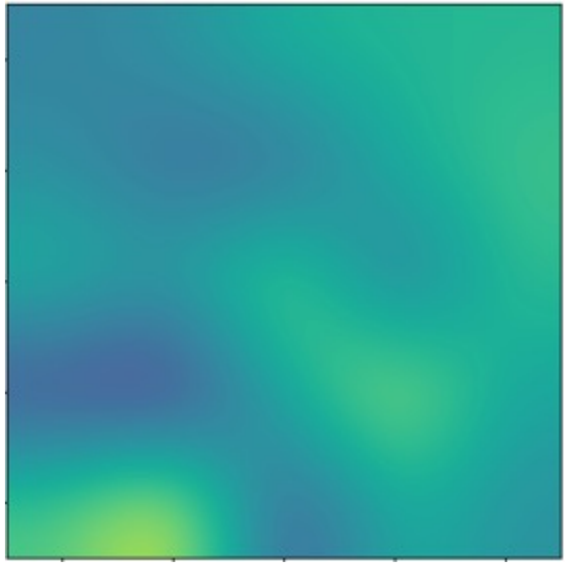


Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

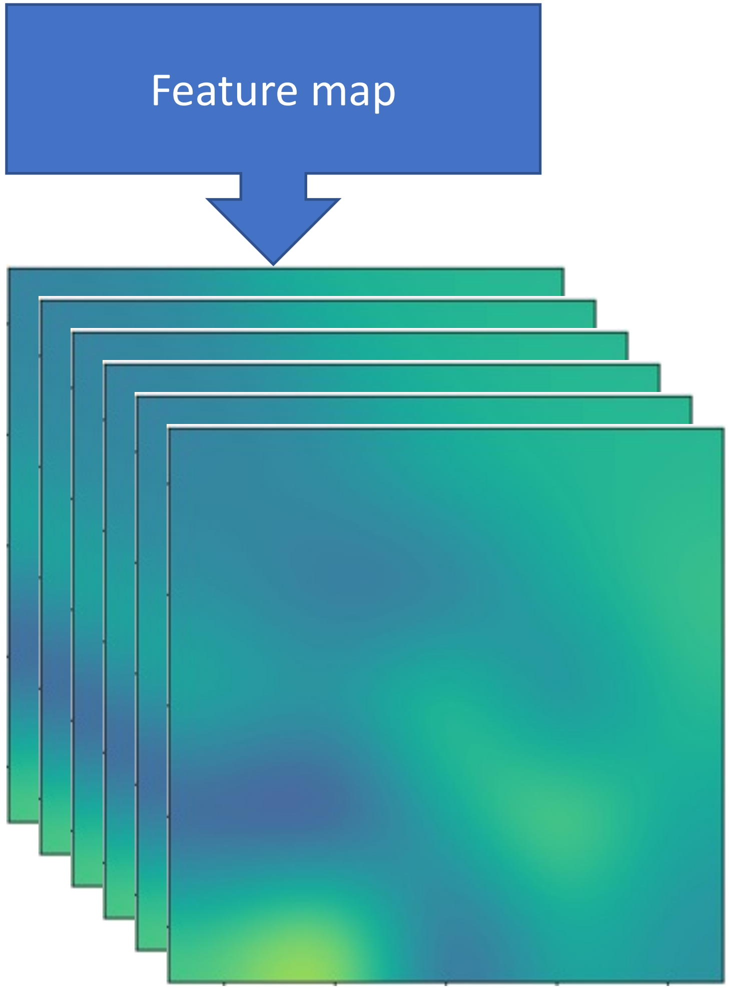
Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

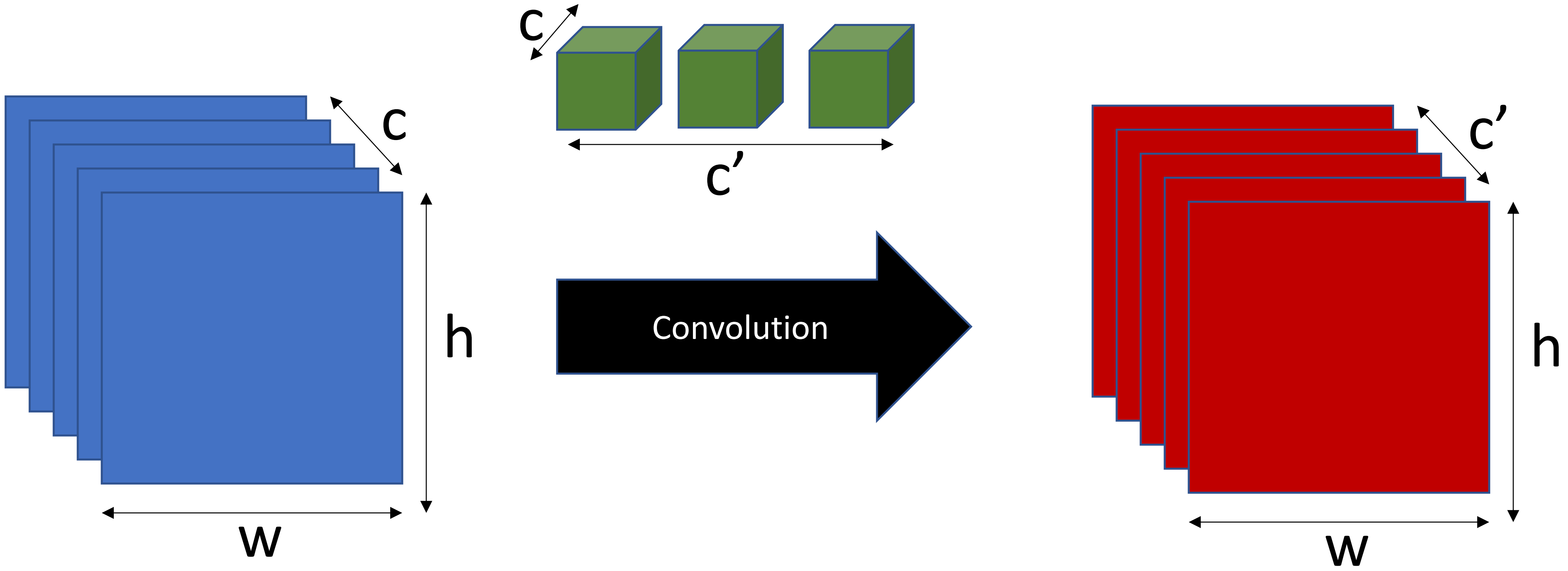


Local connectivity + translation invariance = *convolution*

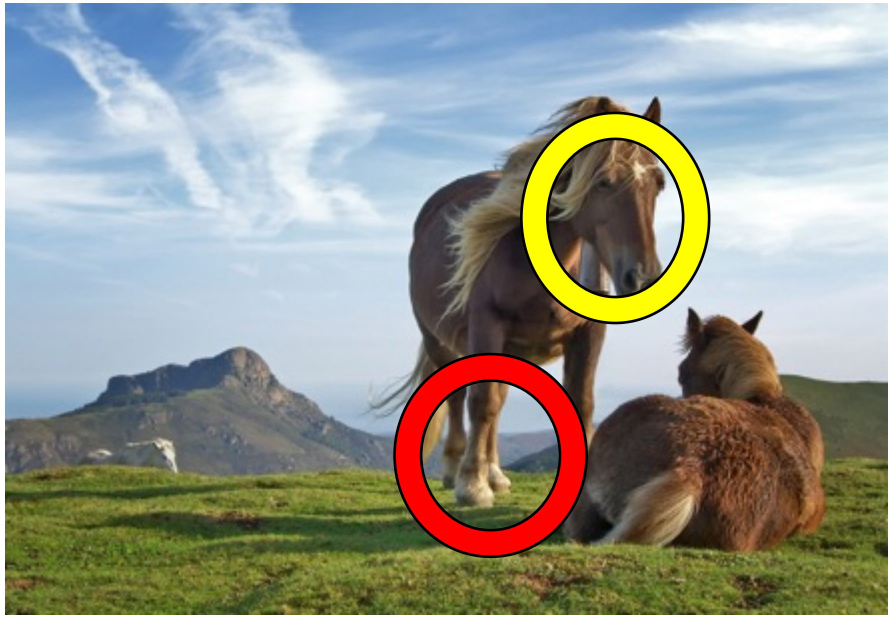
5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2



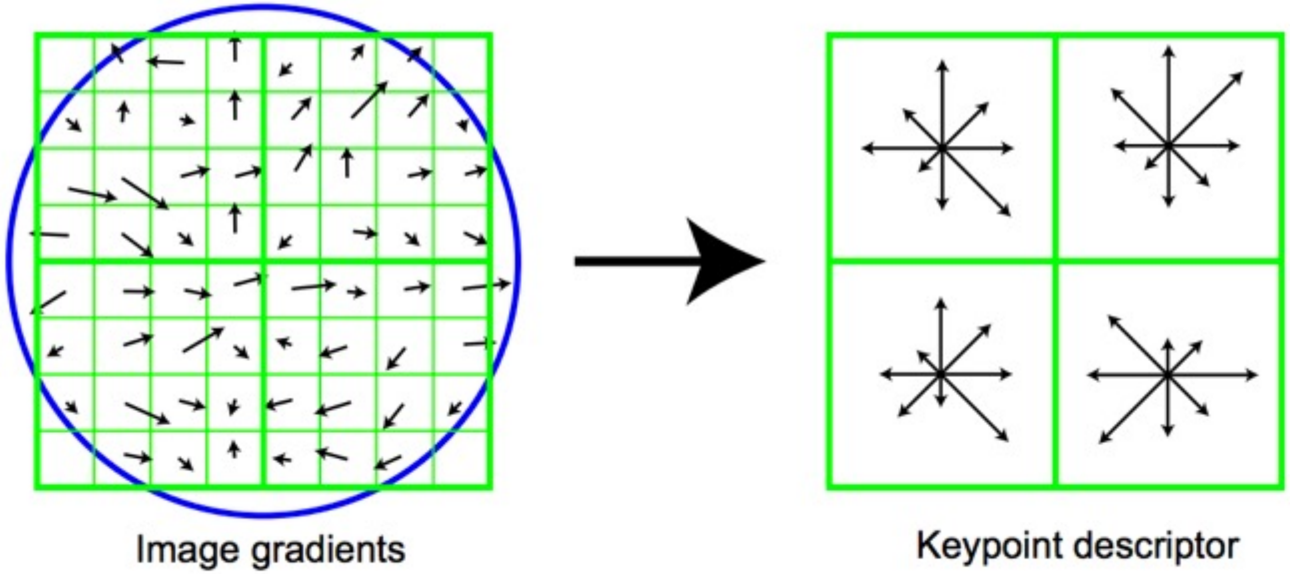
Convolution as a primitive



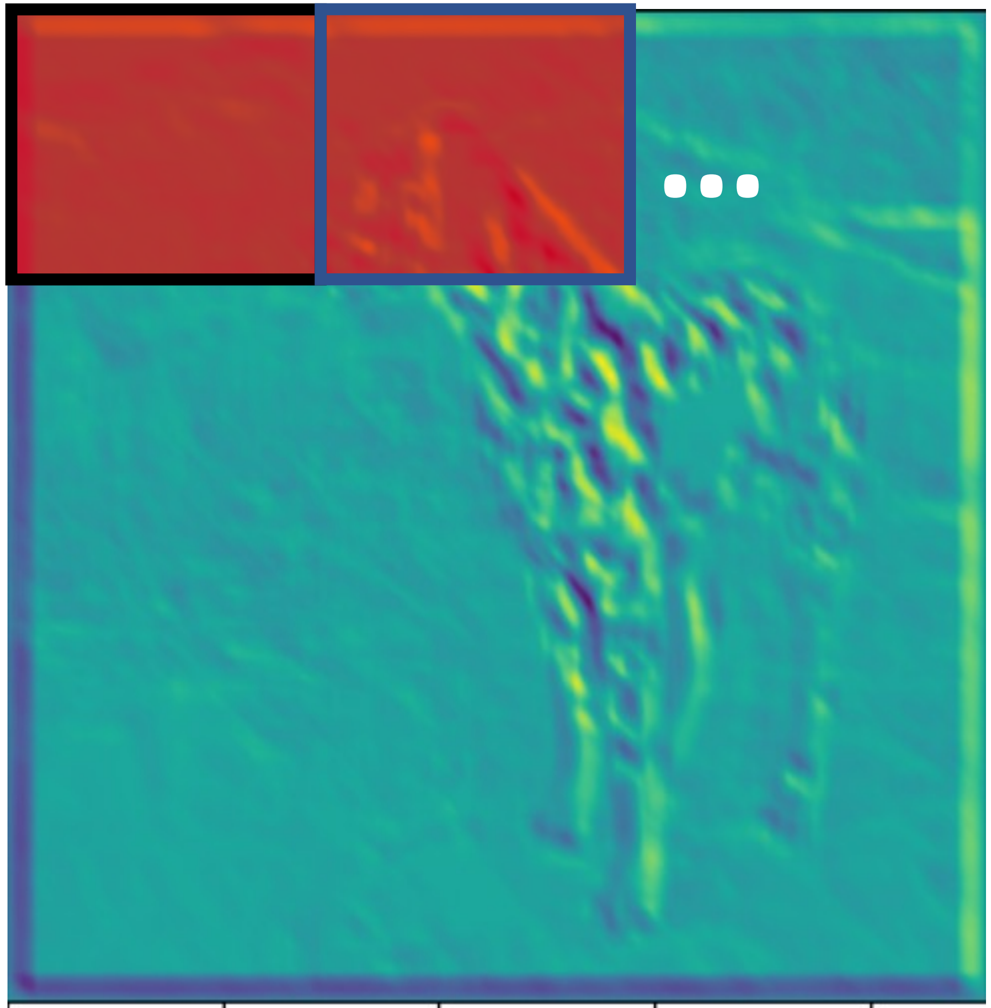
Invariance to distortions



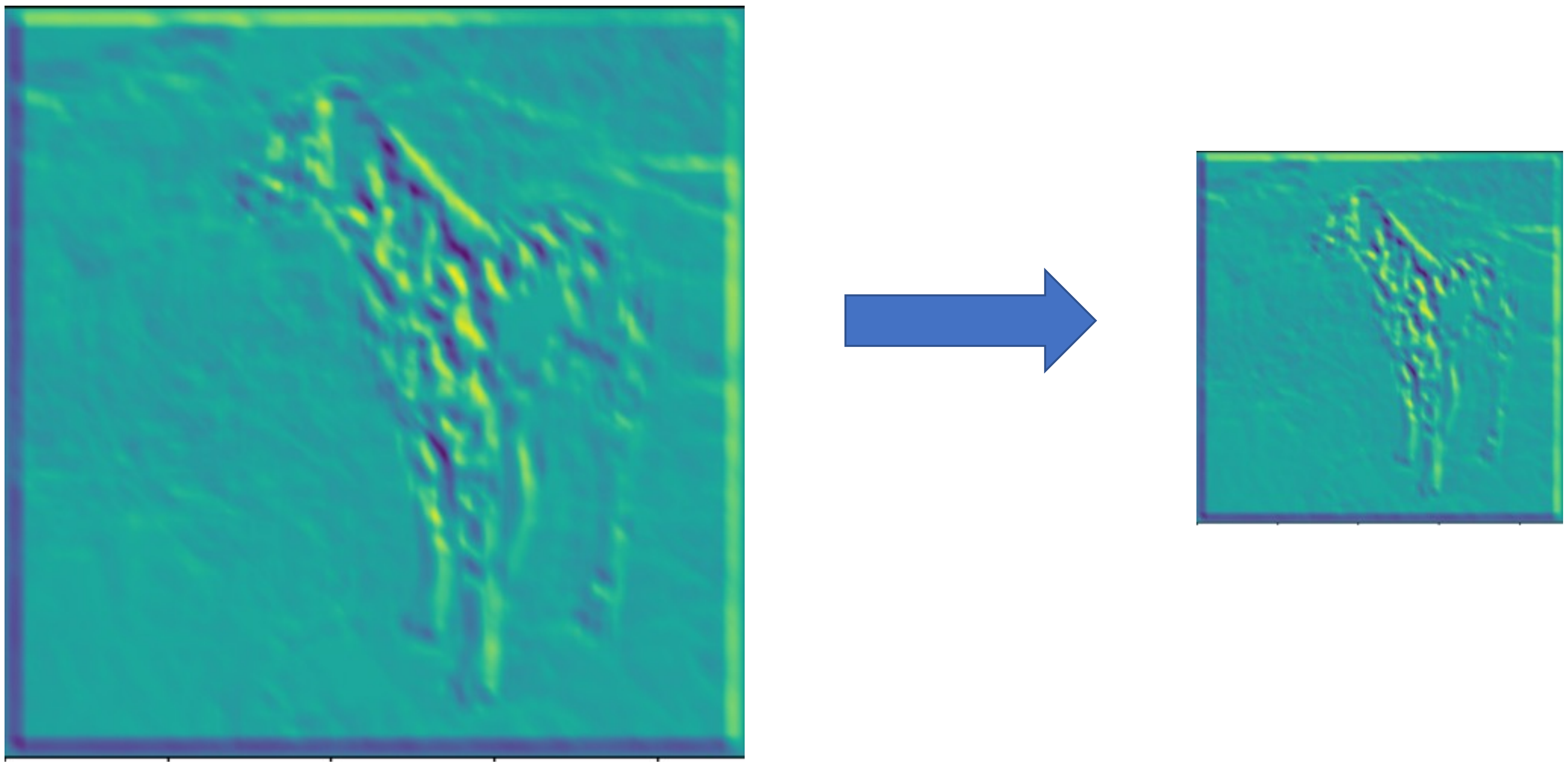
Invariance to distortions



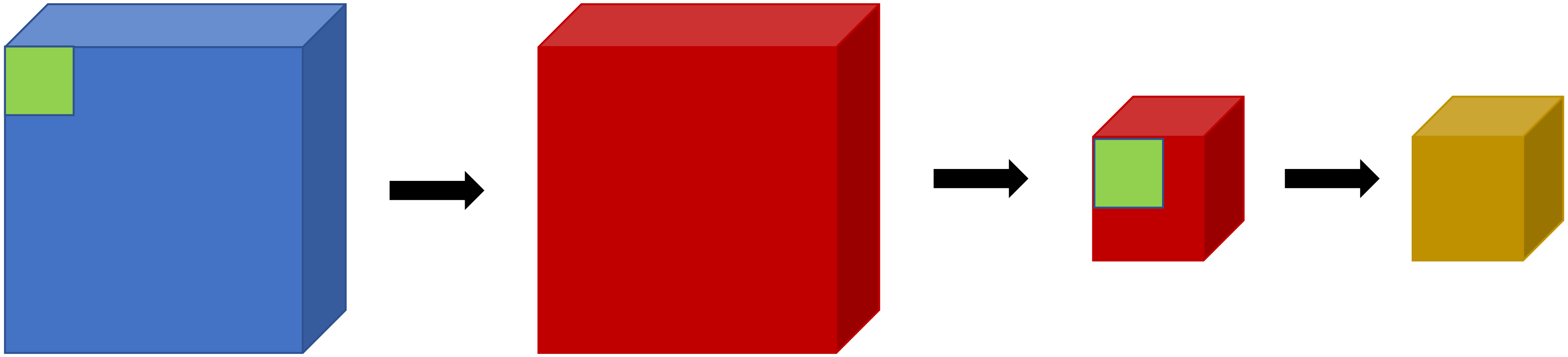
Invariance to distortions: Pooling



Invariance to distortions: Subsampling



Convolution subsampling convolution



Convolution subsampling convolution

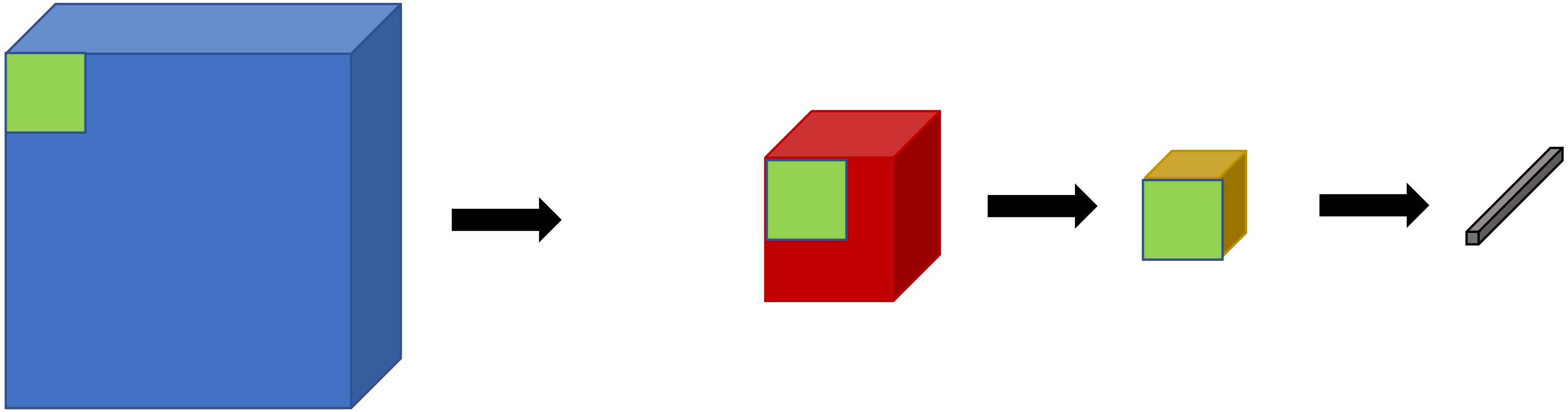
- Convolution in earlier steps detects *more local* patterns *less resilient* to distortion
- Convolution in later steps detects *more global* patterns *more resilient* to distortion
- Subsampling allows capture of *larger, more invariant* patterns

Convolution with subsampling

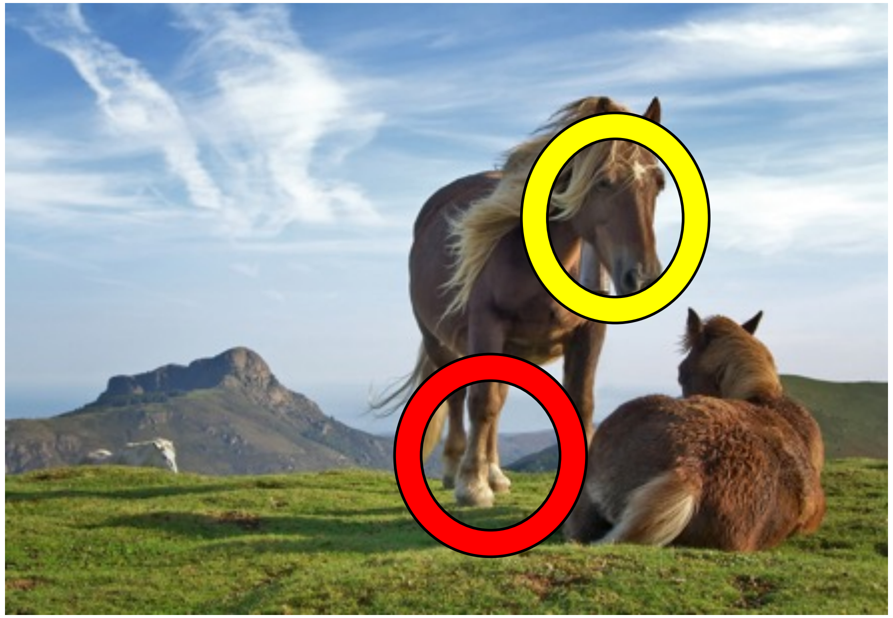
- *Subsampling =* reducing resolution by dropping rows and columns
- Can be done with *strided* convolution
  - Stride of k means output pixel every k input pixels
- Typically done *without anti-aliasing,* though *anti-aliasing helps*[1]

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

Convolution with subsampling



Invariance to deformations

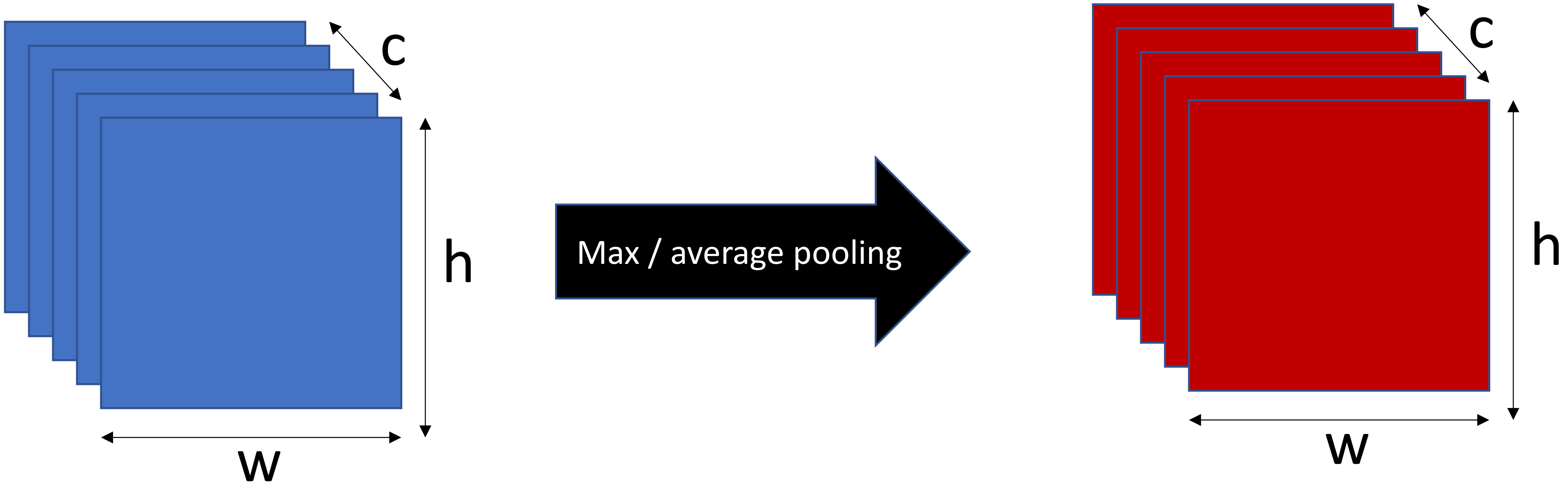


Effect of subsampling

- 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

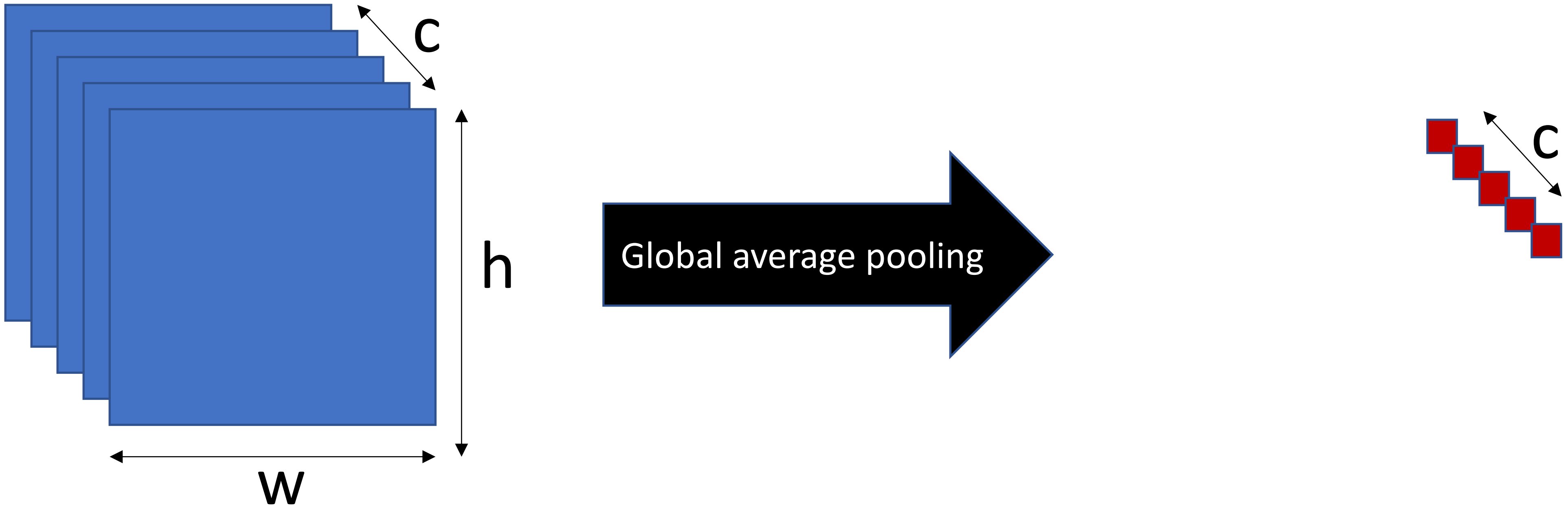
Pooling

- 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



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

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

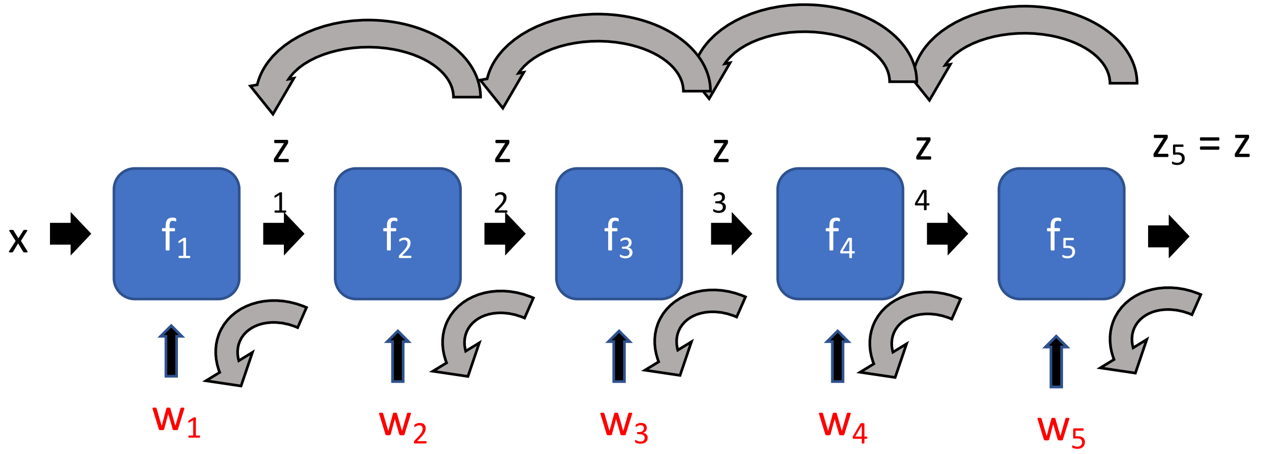
$$z = h(x;\boldsymbol{\theta})$$

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

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

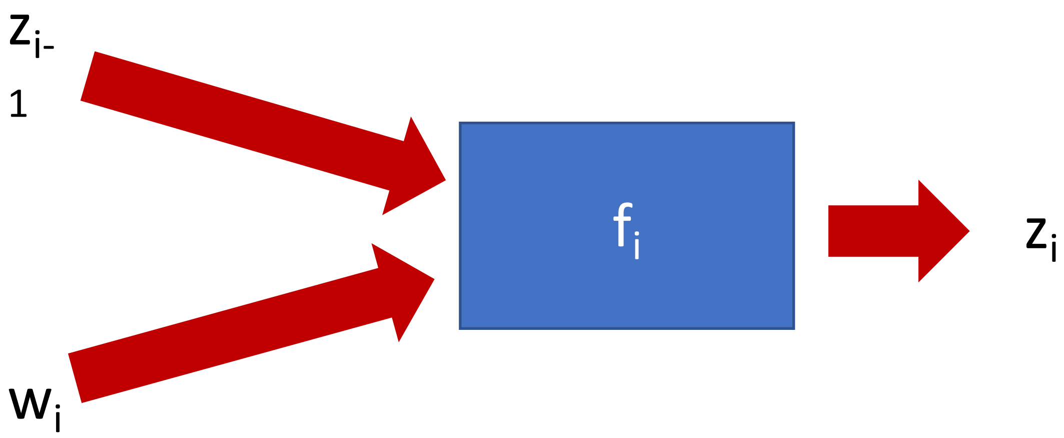
$$g(w_i) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial w_i} = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions

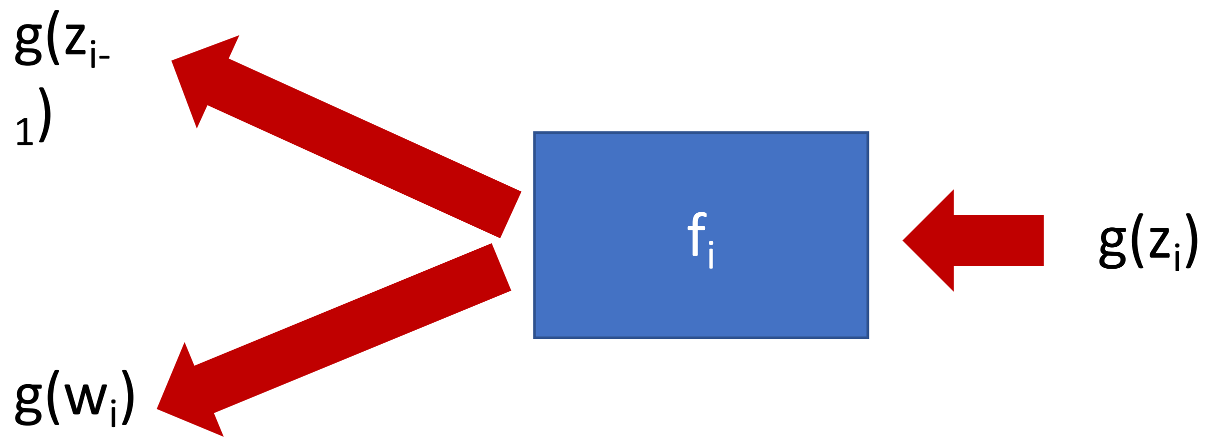
- Each “function” has a “forward” and “backward” module
- Forward module for f_i
  - takes z_{i-1} and weight w_i as input
  - produces z_i as output
- Backward module for f_i
  - takes $g(z_i)$ as input
  - produces $g(z_{i-1})$ and $g(w_i)$ as output

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

Backpropagation for a sequence of functions



Backpropagation for a sequence of functions



Chain rule for vectors

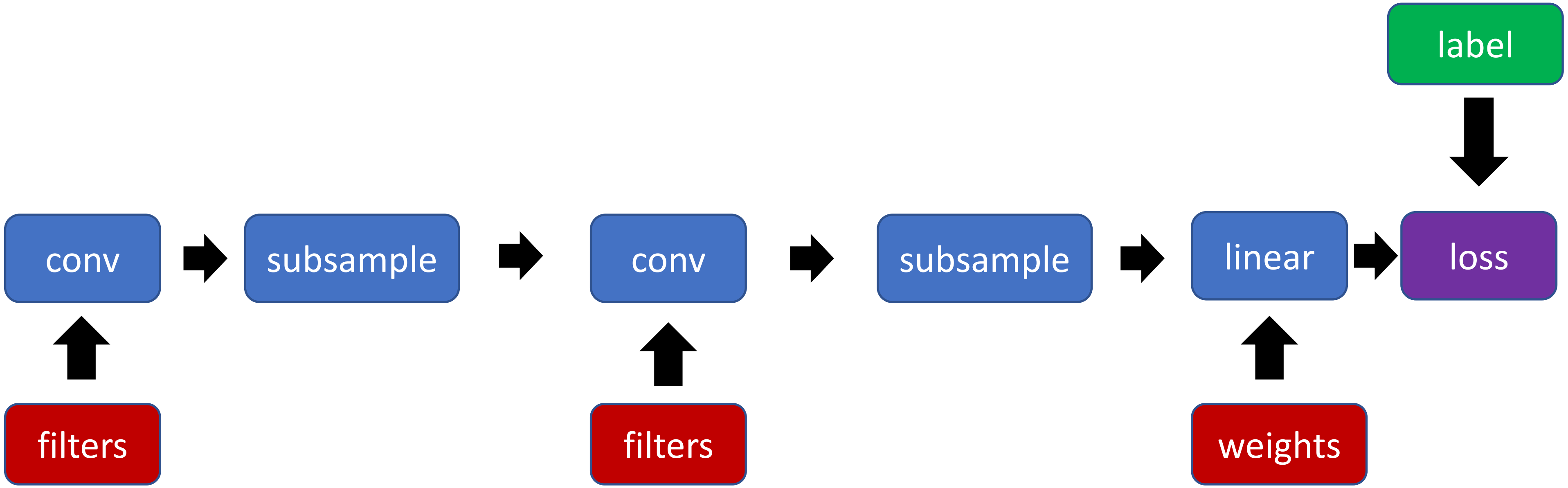
$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c}\frac{\partial c}{\partial b} \qquad \frac{\partial a_i}{\partial b_j} = \sum_k \frac{\partial a_i}{\partial c_k}\frac{\partial c_k}{\partial b_j}$$

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

Jacobian

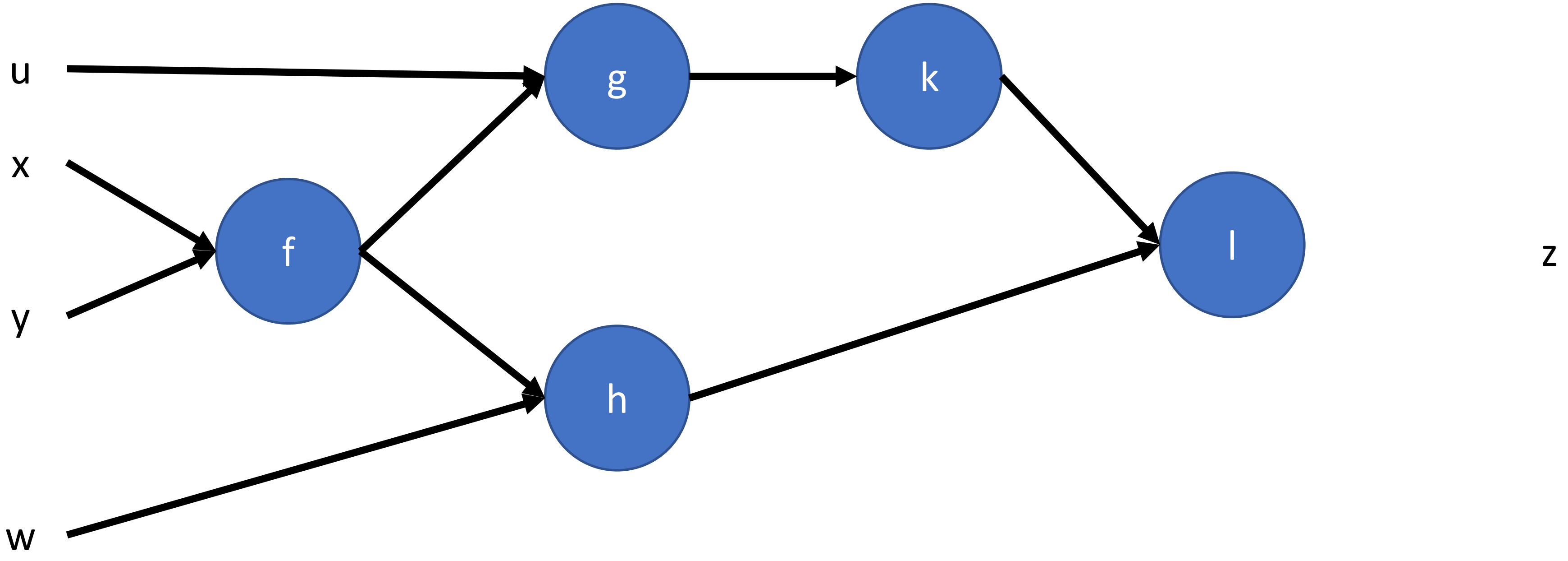
$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}} = \frac{\partial \mathbf{a}}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{b}}$$

Loss as a function



Beyond sequences: computation graphs

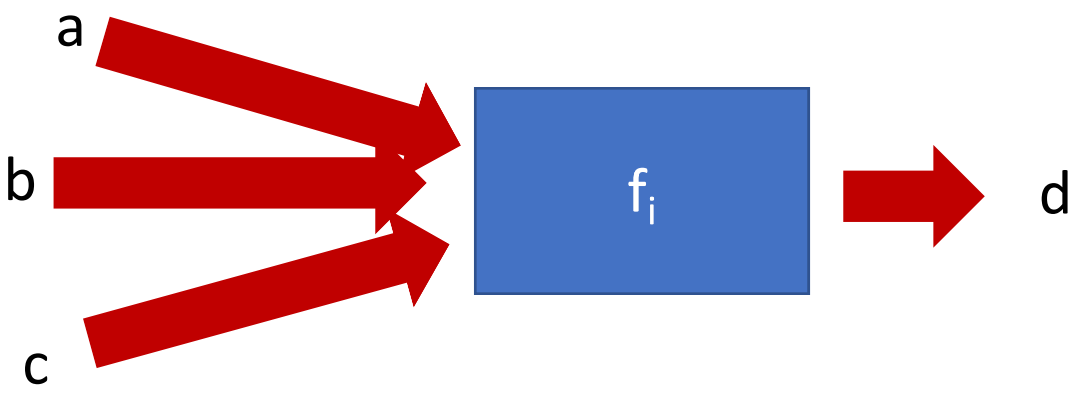
- Arbitrary *graphs* of functions
- No distinction between intermediate outputs and parameters



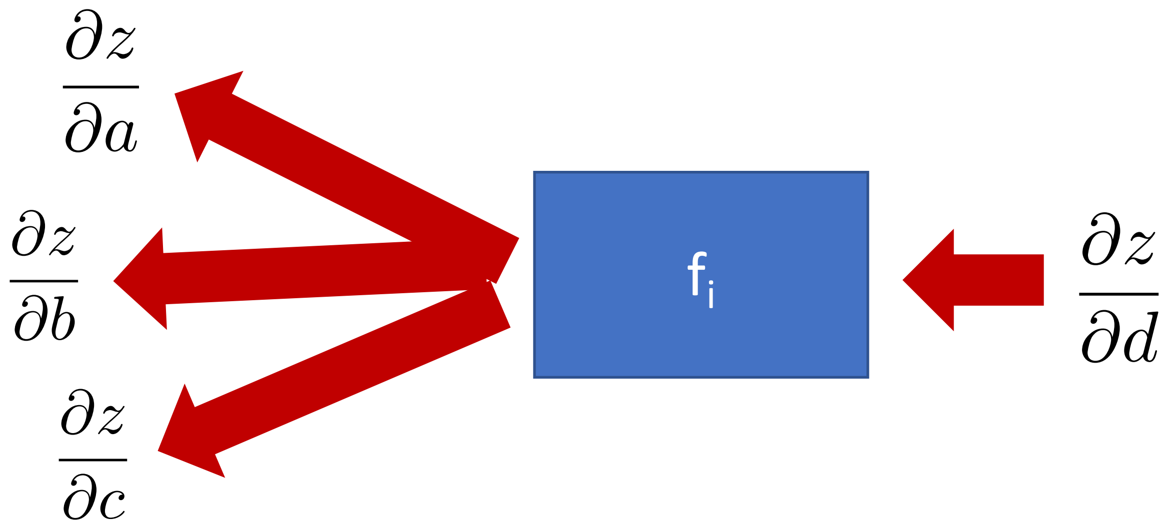
Computation graph - Functions

- 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

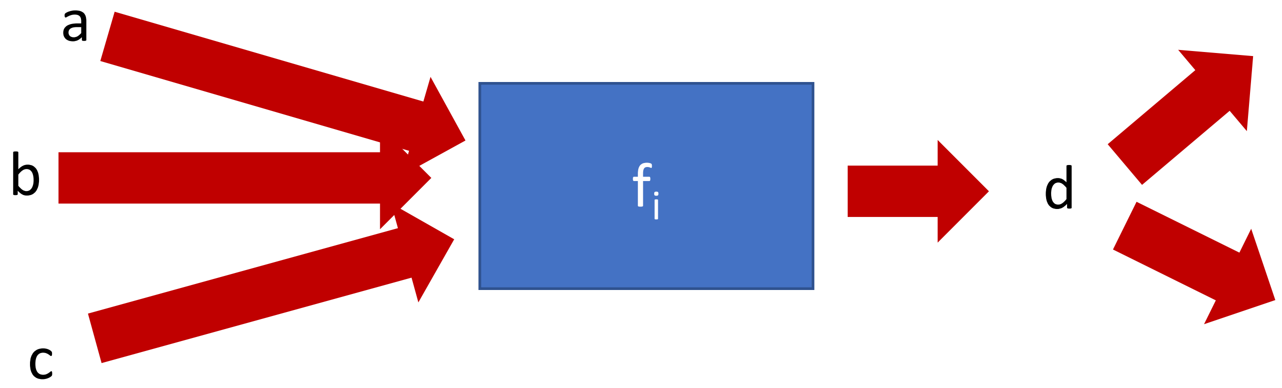
Computation graphs



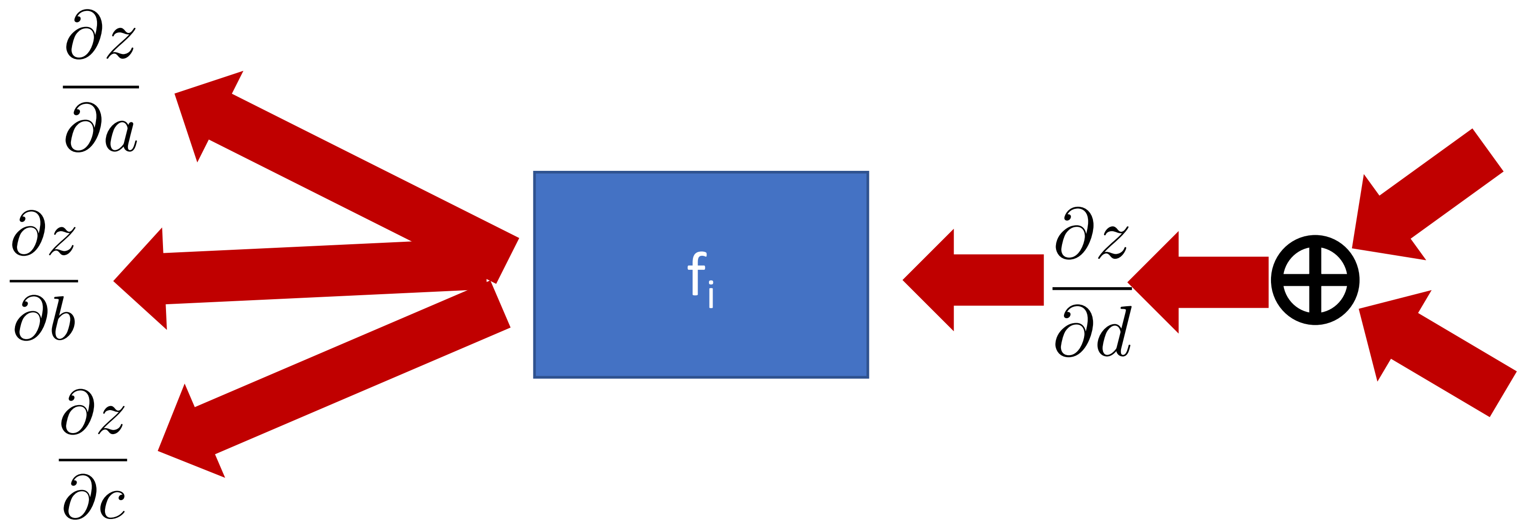
Computation graphs



Computation graphs

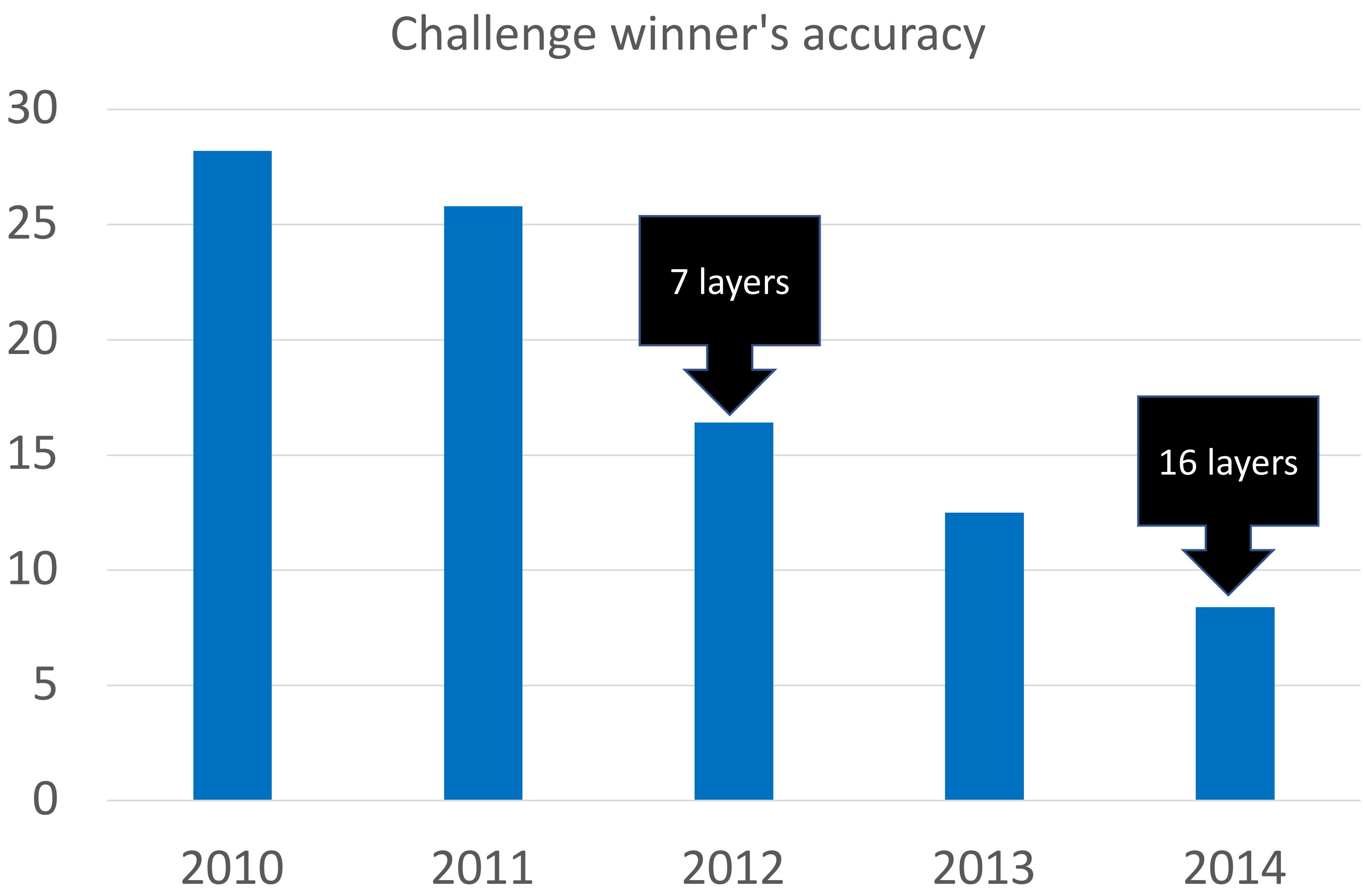


Computation graphs

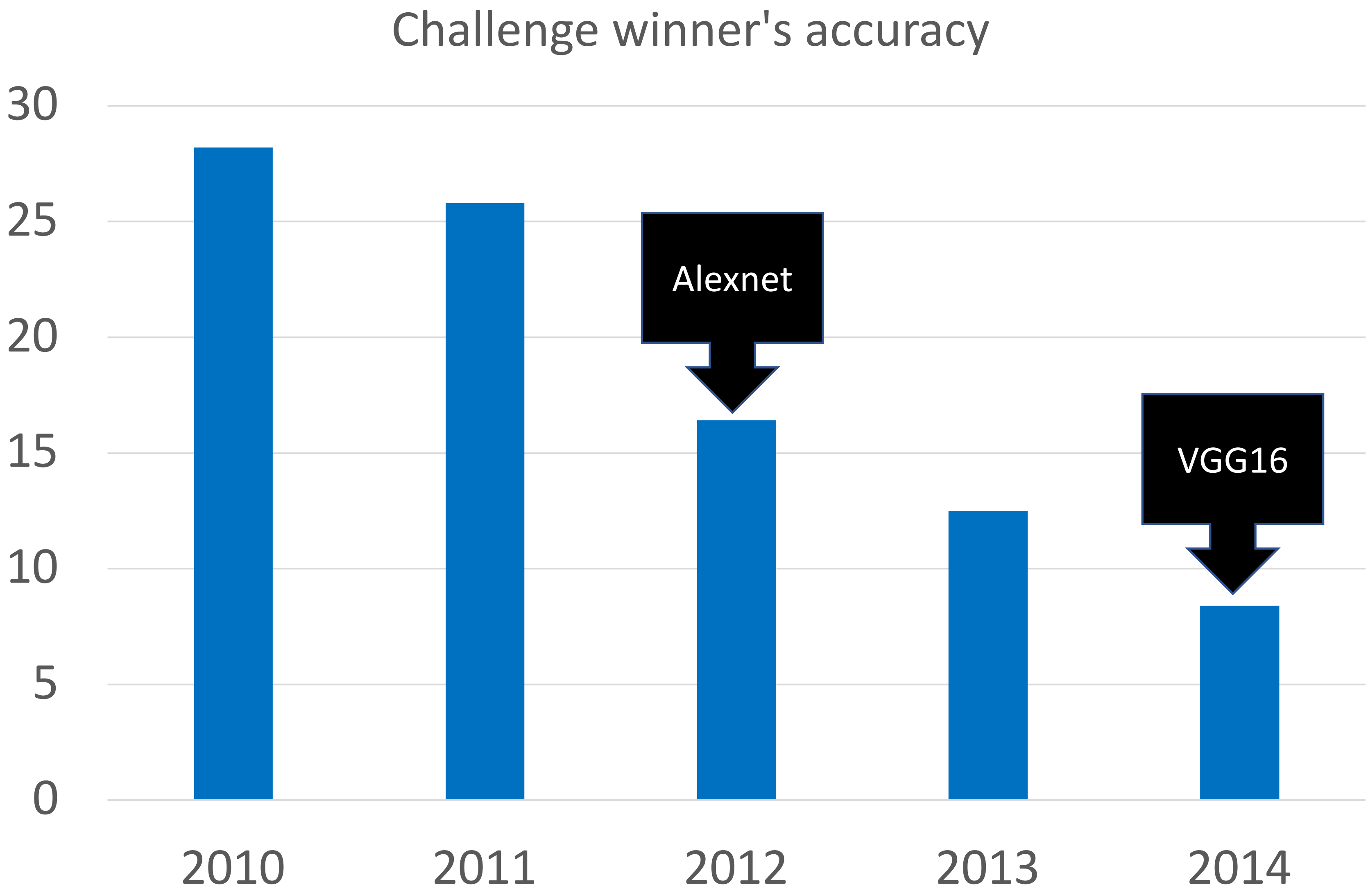


Exploring convnet architectures

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

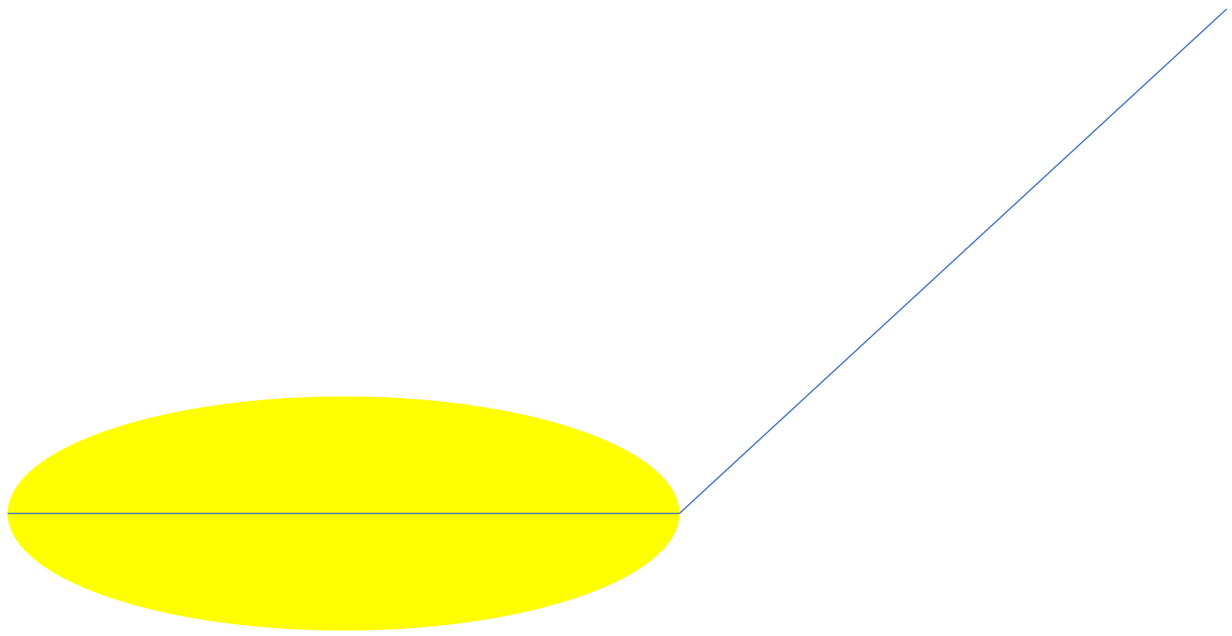
Challenges in training: exploding / vanishing gradients

- 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

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

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

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

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

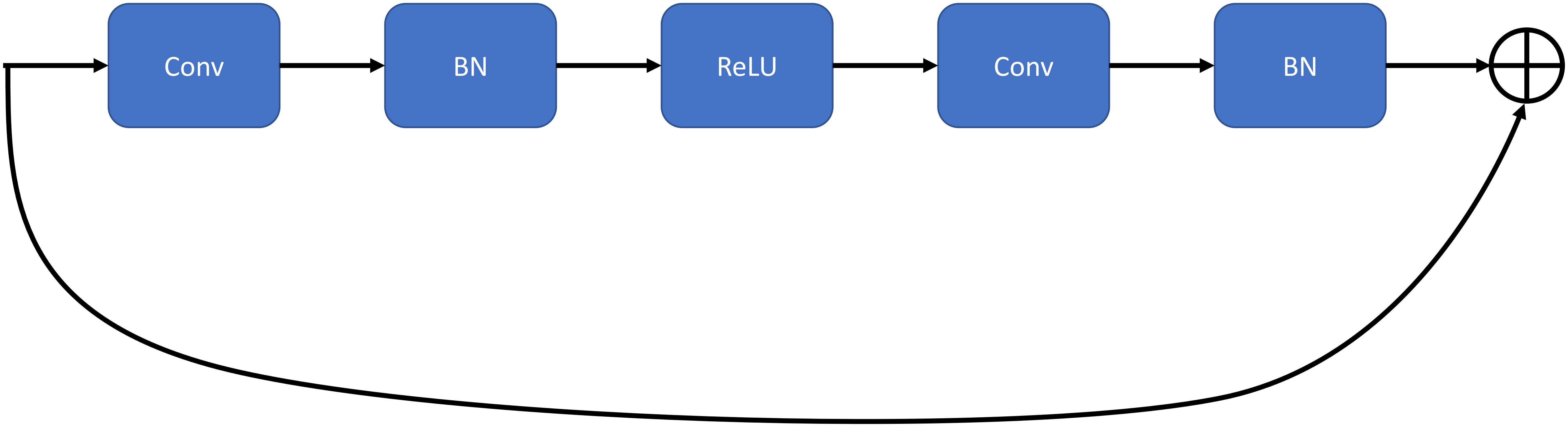
Residual connections

- 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



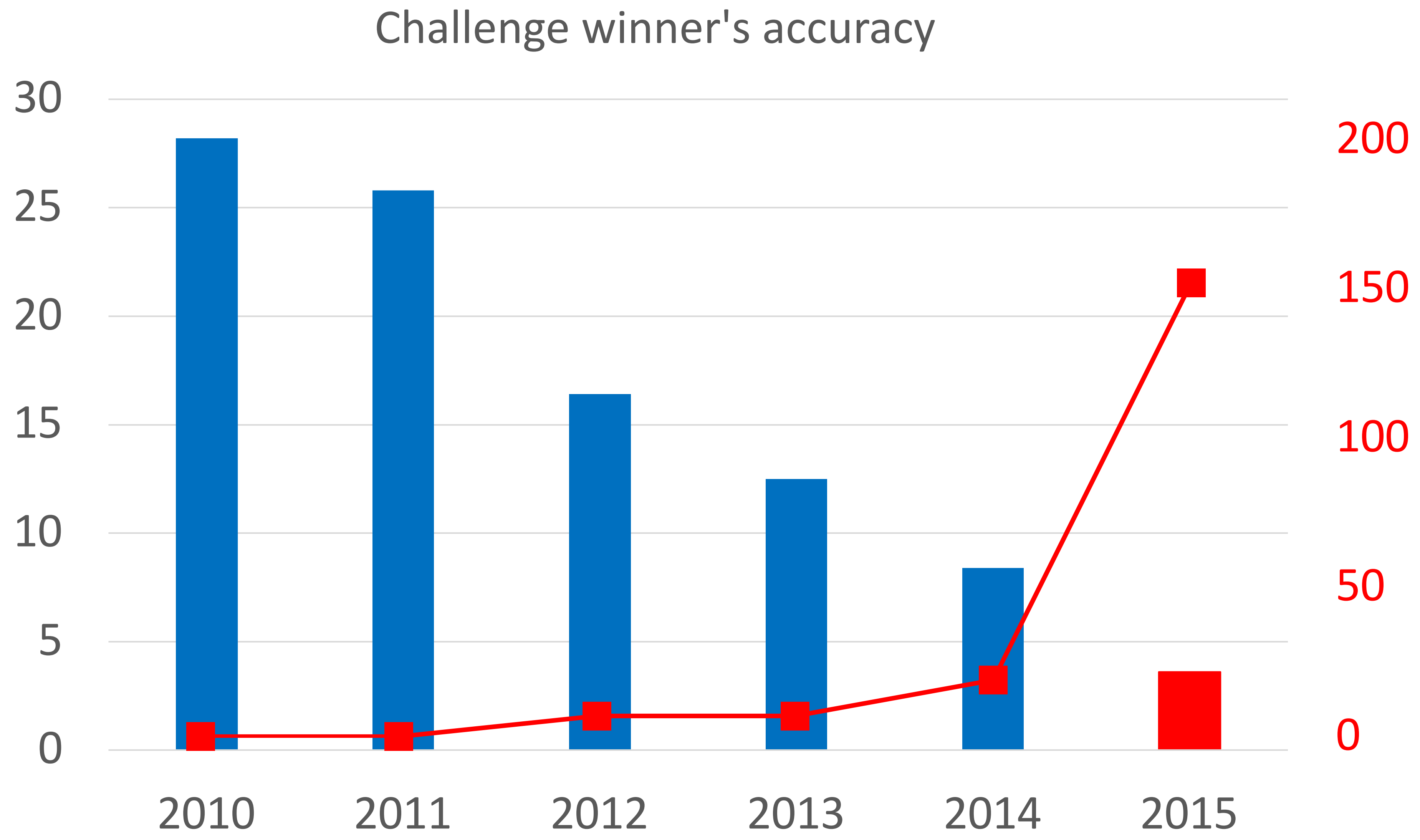
Bottleneck blocks

- 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
  - $c \times d \, + \, 3 \times 3 \times d^2 \, + \, d \times c$

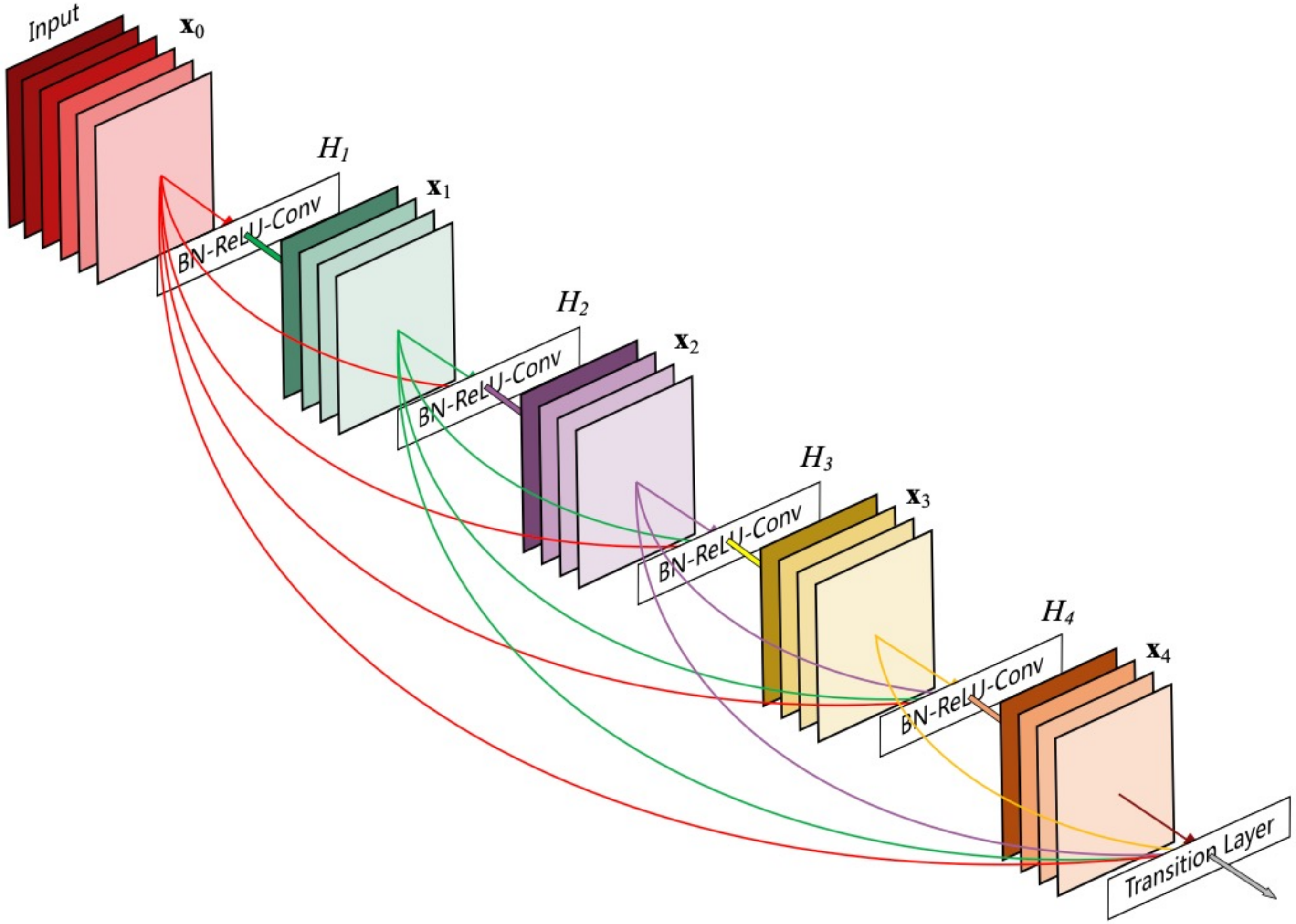
The ResNet pattern

- 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