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

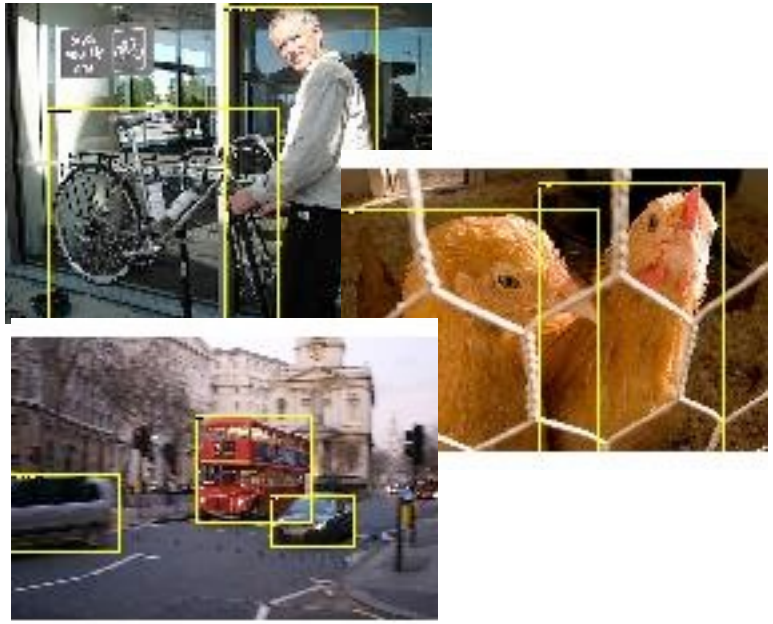
1990’s

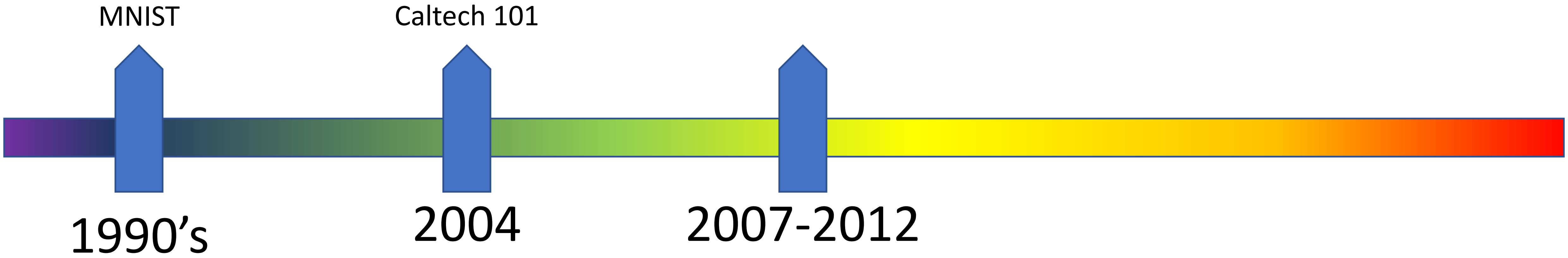
Caltech 101

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

PASCAL VOC

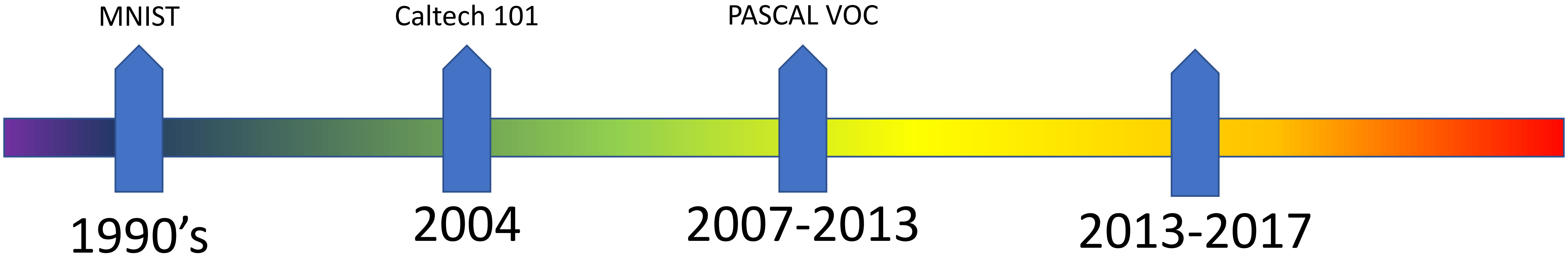
- 20 classes
- ~500 examples per class
- Clutter, occlusion, natural scenes



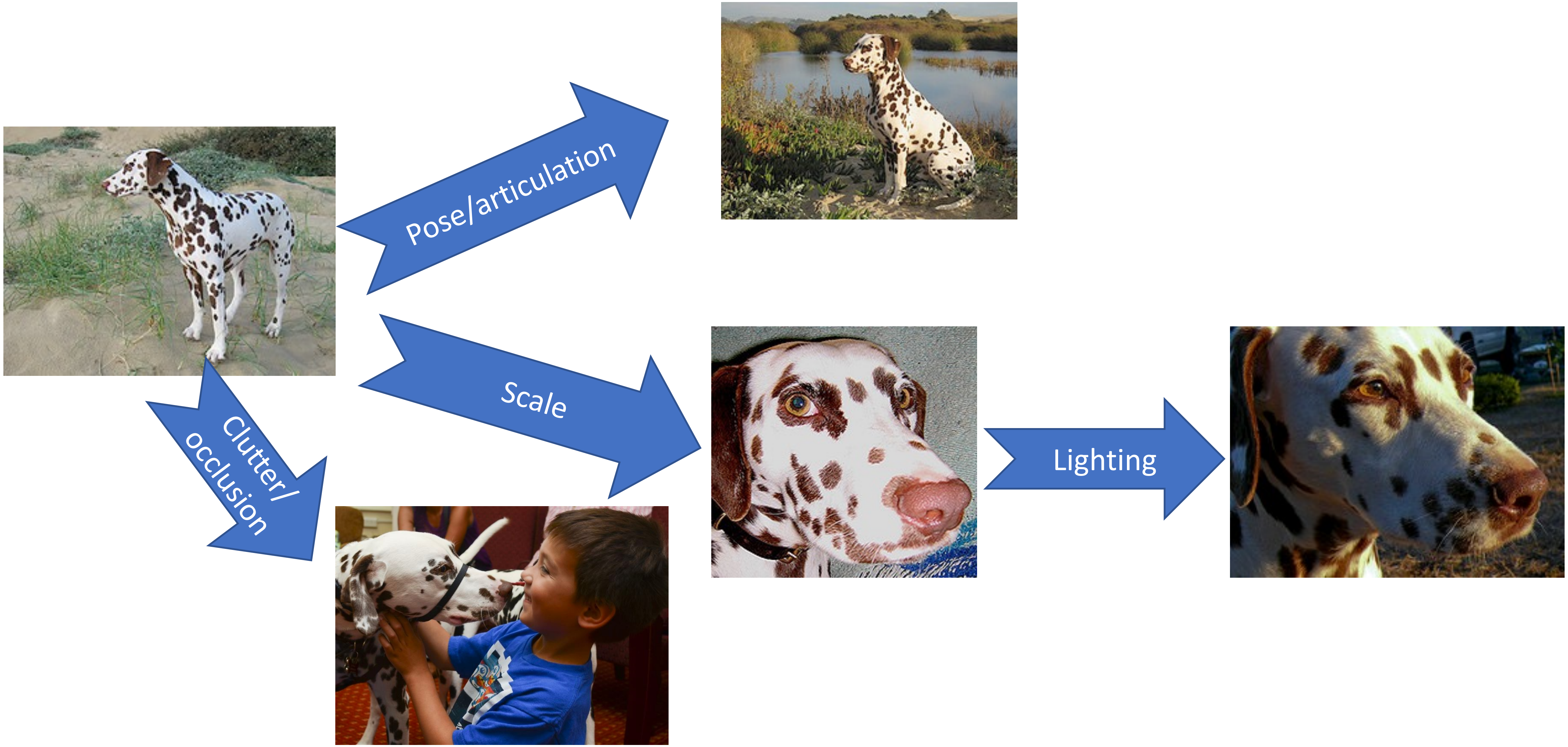


ImageNet

- 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

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

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*

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

- 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

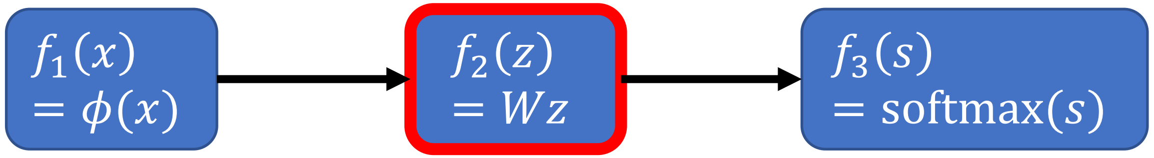
- Will estimate a probability P(y | x)
- Any function that takes x as input and outputs probability distribution
  - $h : \mathcal{X} \rightarrow C^{|\mathcal{Y}|}$ where C^d is a probability distribution over d classes
  - Very large set of possibilities for h
- Constrain choice: Choose a family of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

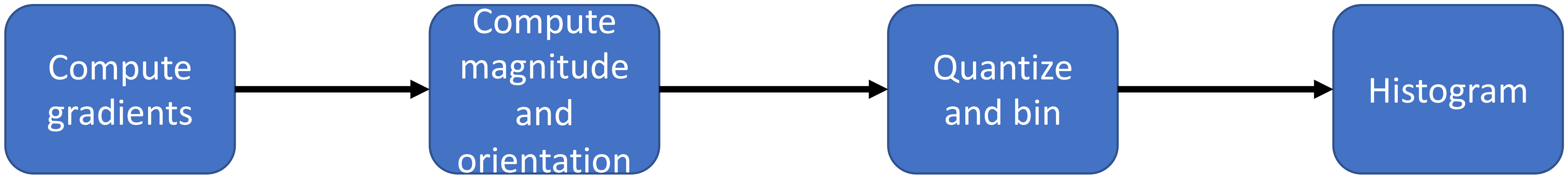


Choice of feature extractor?

- 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?

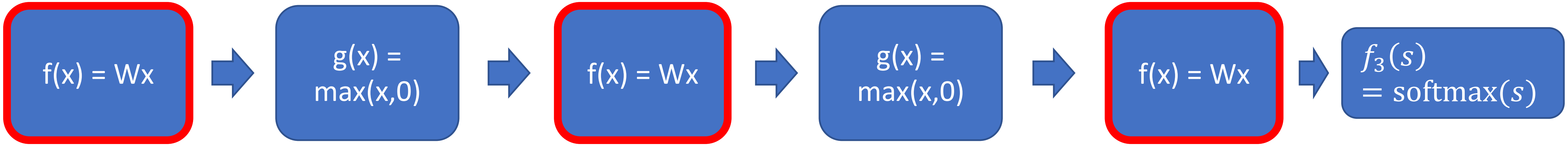
SIFT

- 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

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

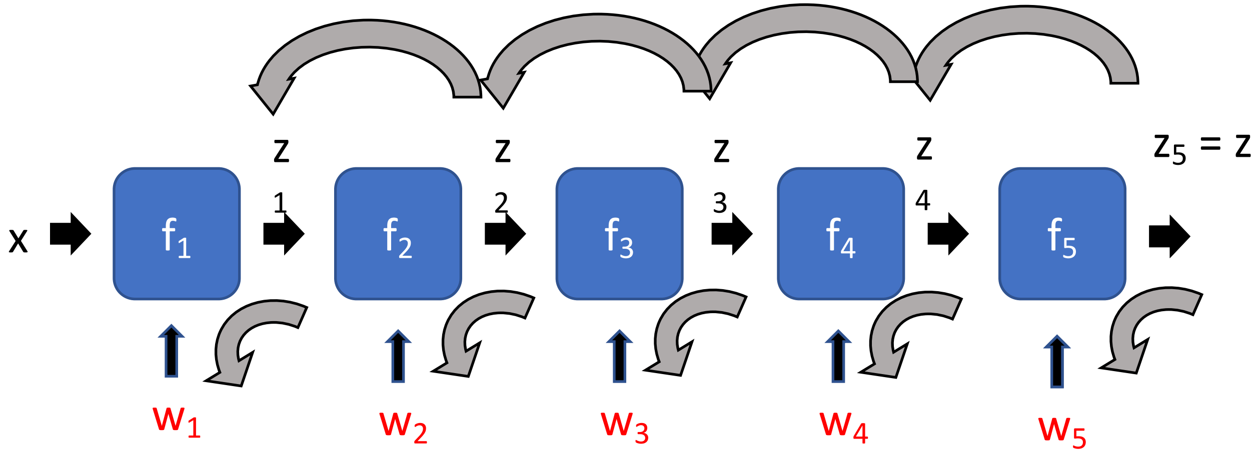
- How do we compute gradient?
- Composition of functions: use chain rule

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

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

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

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

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

$$R(h) = \hat{R}(S,h) + (R(h) - \hat{R}(S,h))$$

Training error

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

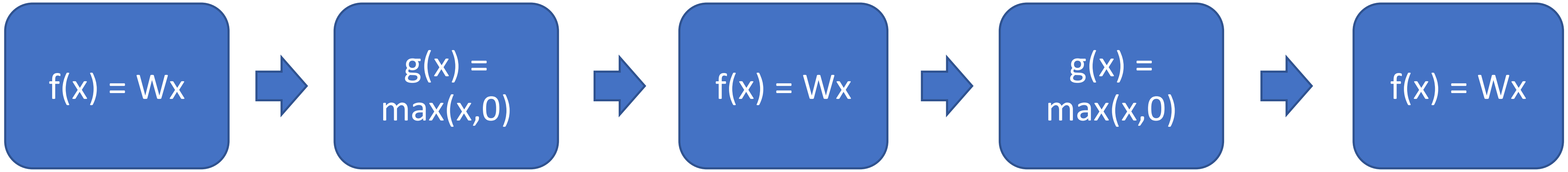
- Collect training set and validation set
- Pick hypothesis class
- Pick loss function
- Minimize empirical risk (+ regularization)
- Measure performance on held-out validation set
- Profit!

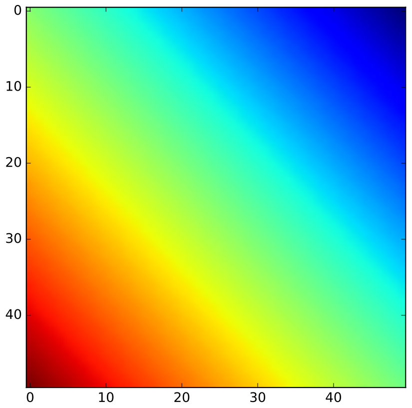
Loss functions and hypothesis classes

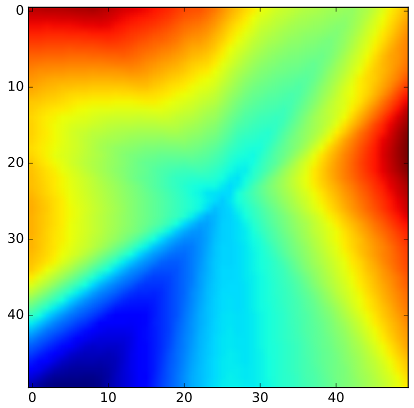
Loss function	Problem	Range of h	$\mathcal{Y}$	Formula
Log loss	Binary Classification	$\mathbb{R}$	$\{0,1\}$	$\log(1+e^{-yh(x)})$
Negative log likelihood	Multiclass classification	$[0,1]^k$	$\{1,\ldots,k\}$	$-\log h_y(x)$
Hinge loss	Binary Classification	$\mathbb{R}$	$\{0,1\}$	$\max(0,1-yh(x))$
MSE	Regression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

Multilayer perceptrons

- Key idea: build complex functions by composing simple functions



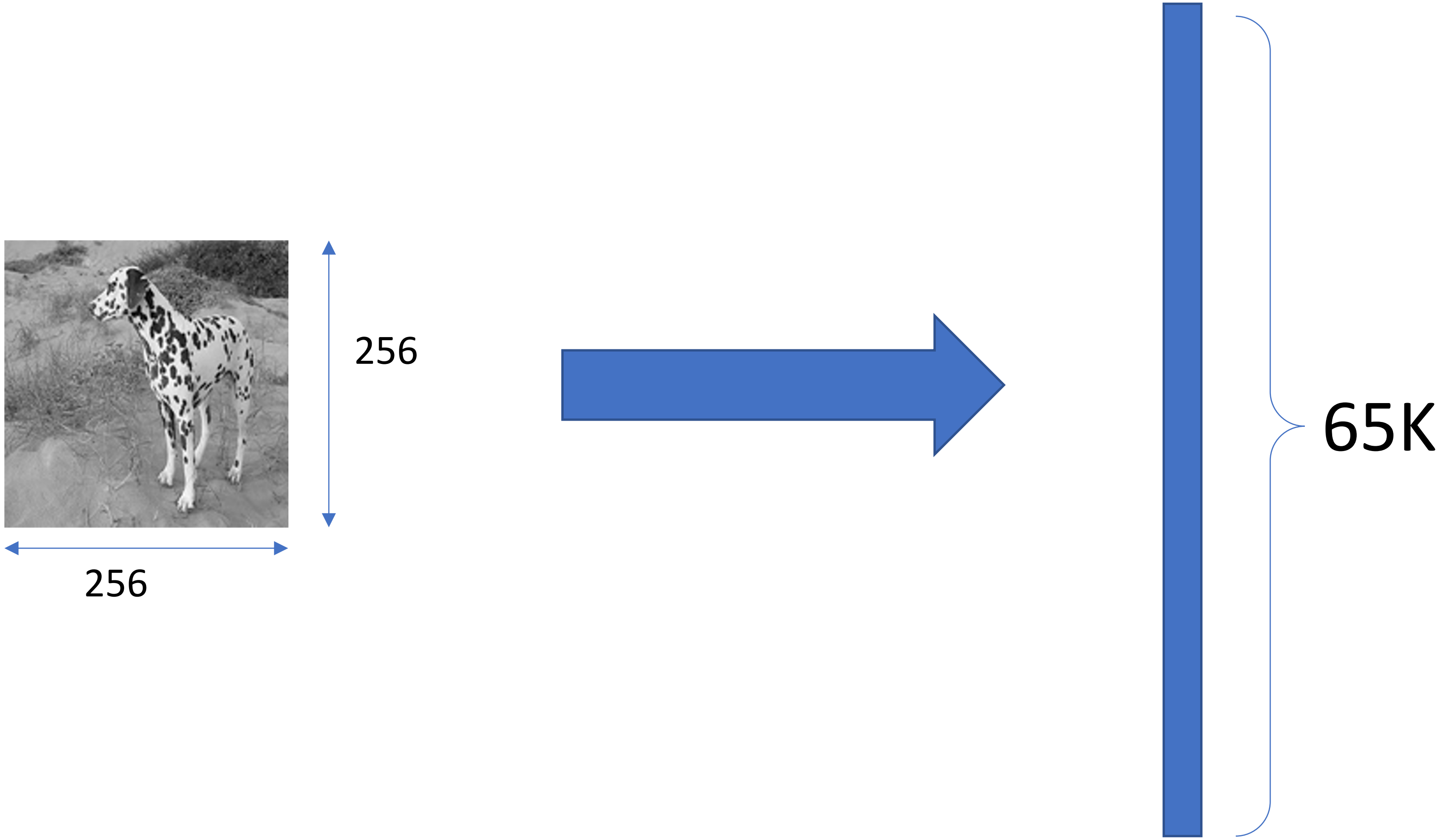




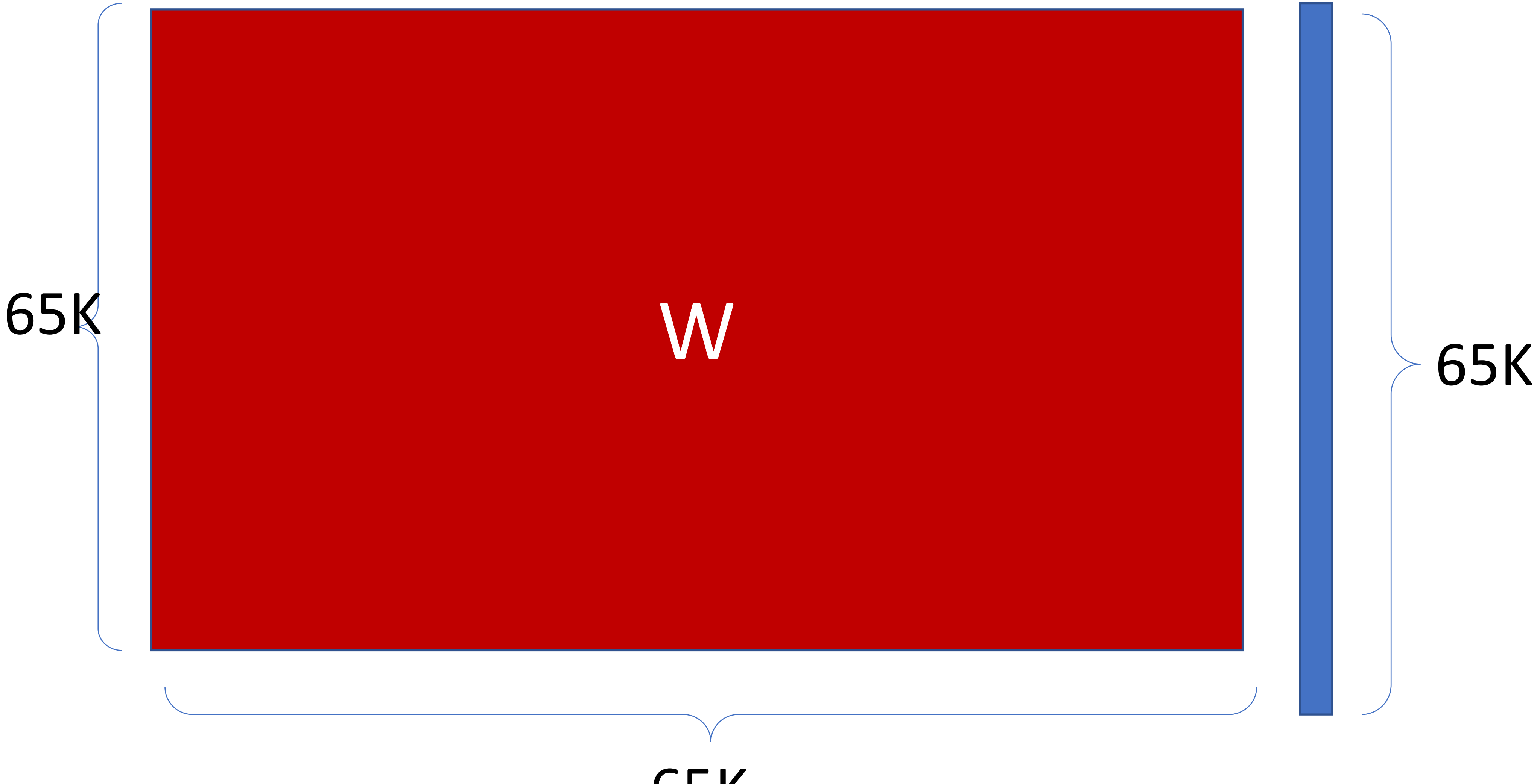
Multilayer perceptrons

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

Reducing capacity

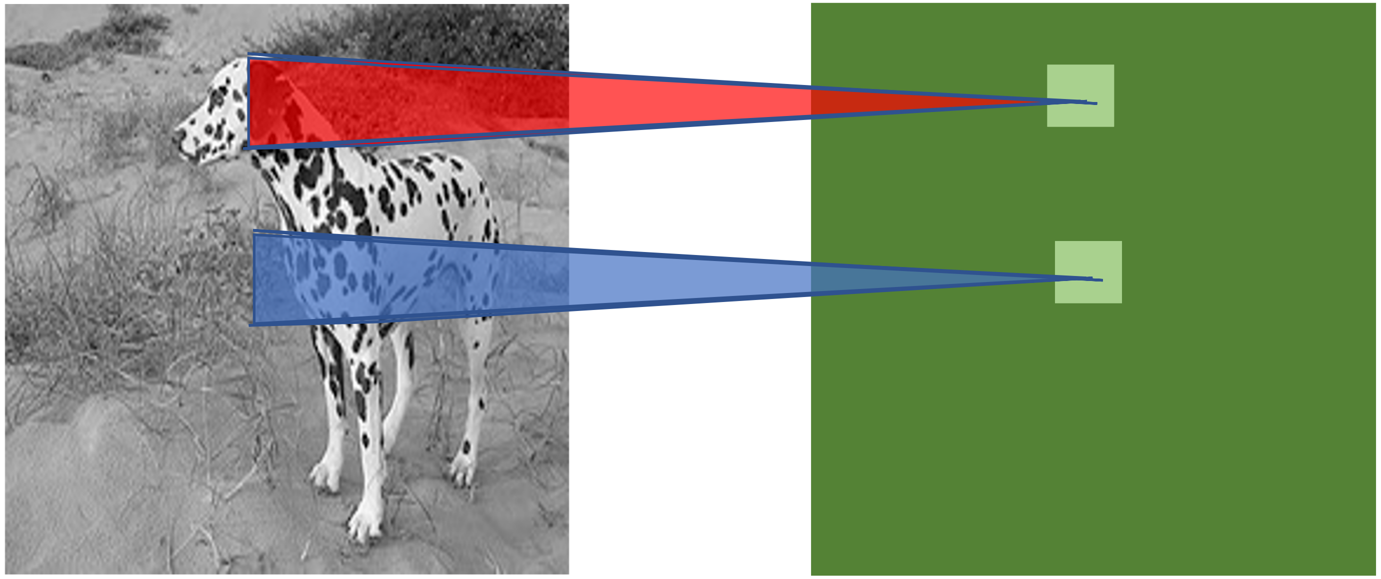


Reducing capacity



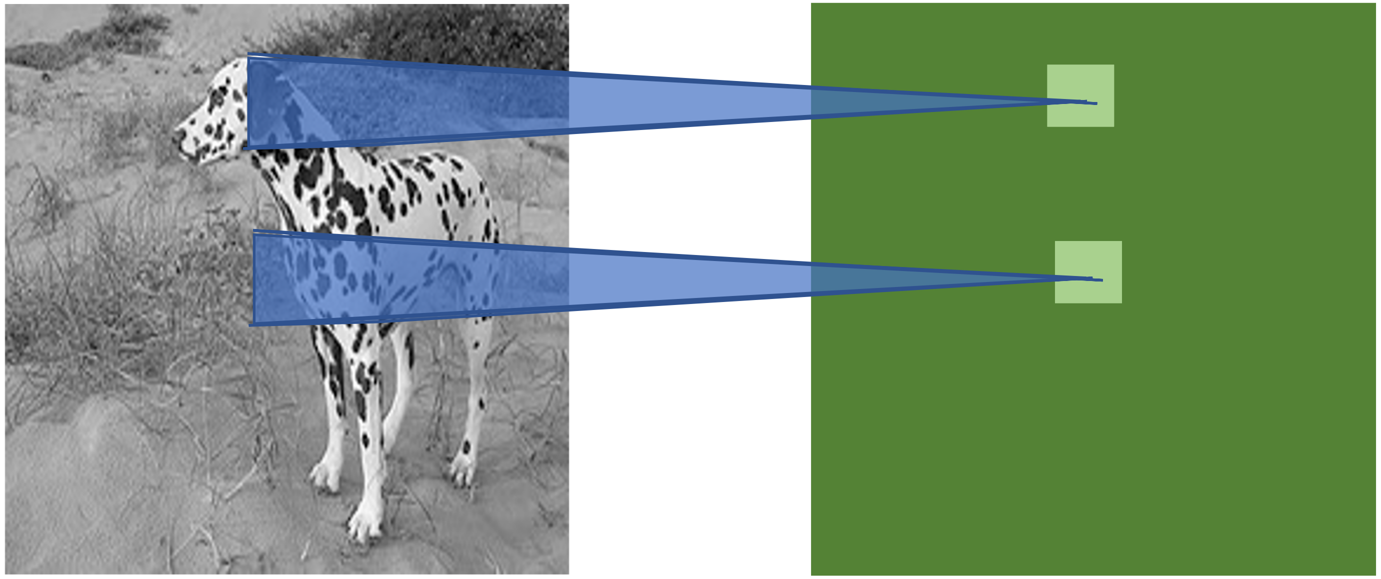
Idea 1: local connectivity

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



Idea 2: Translation invariance

- Pixels only related to nearby pixels

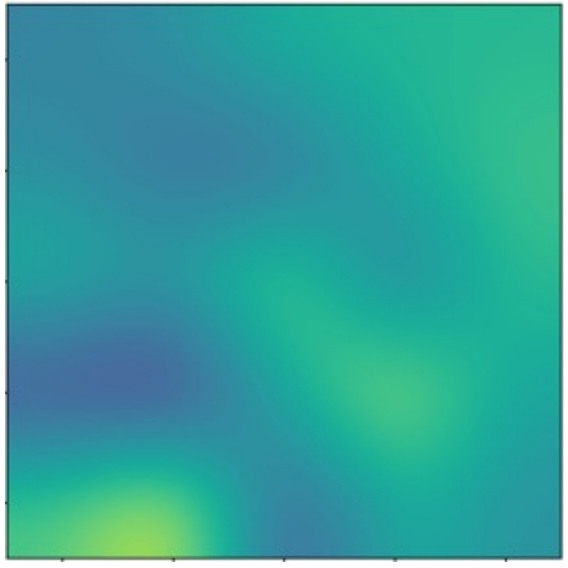


Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

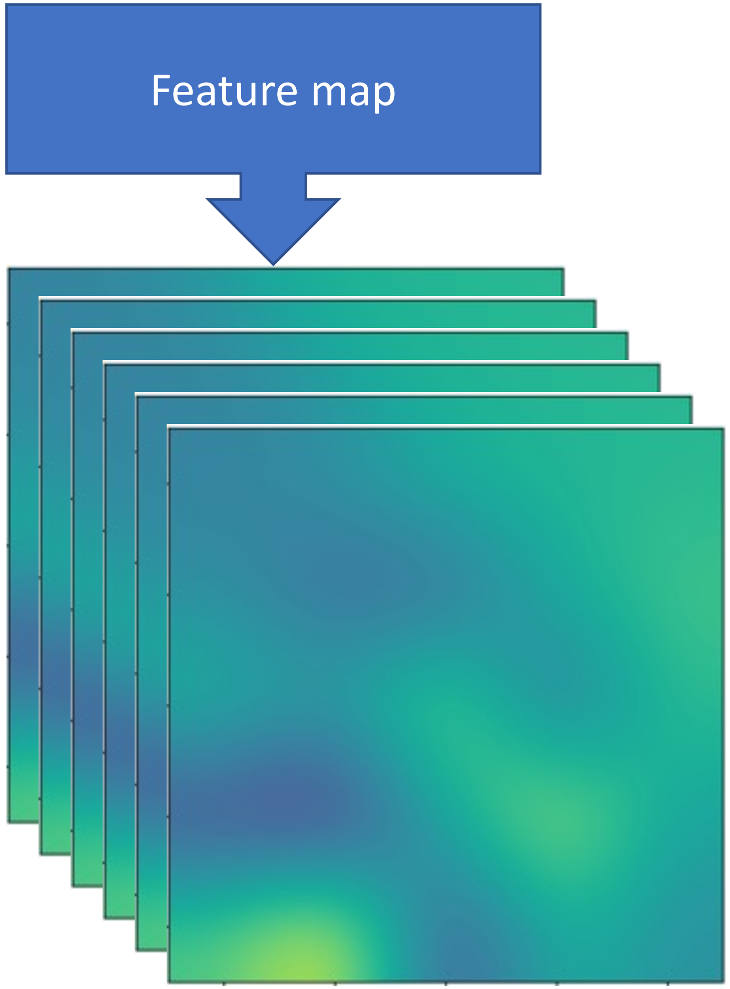
Local connectivity + translation invariance = *convolution*

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

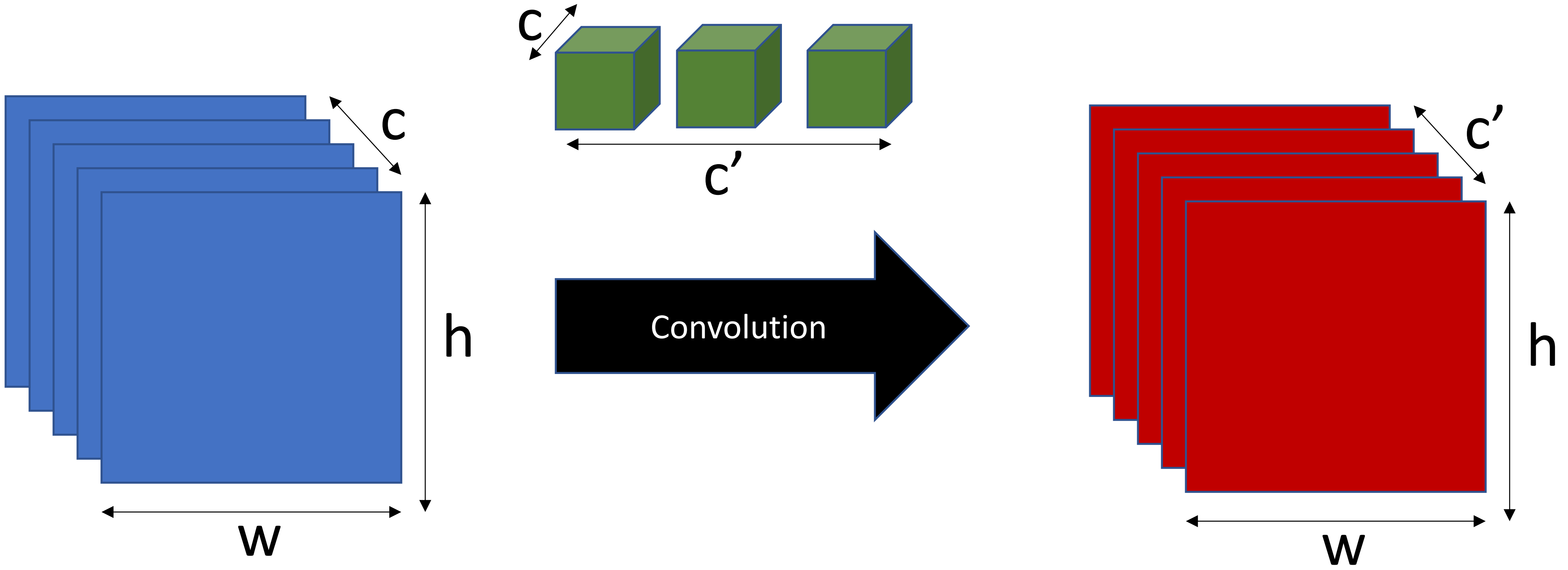


Local connectivity + translation invariance = *convolution*

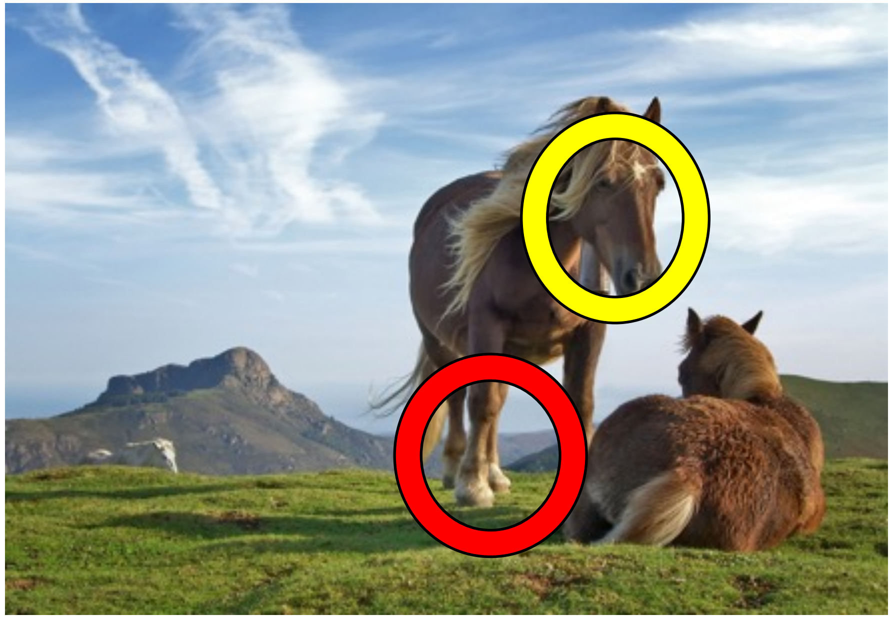
5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2



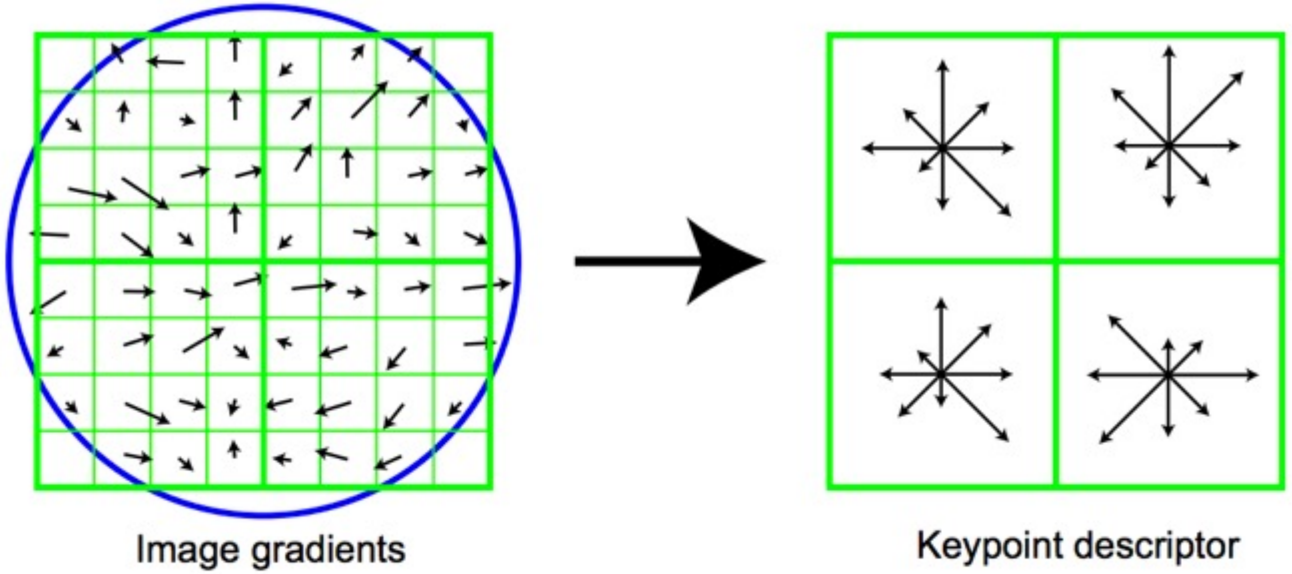
Convolution as a primitive



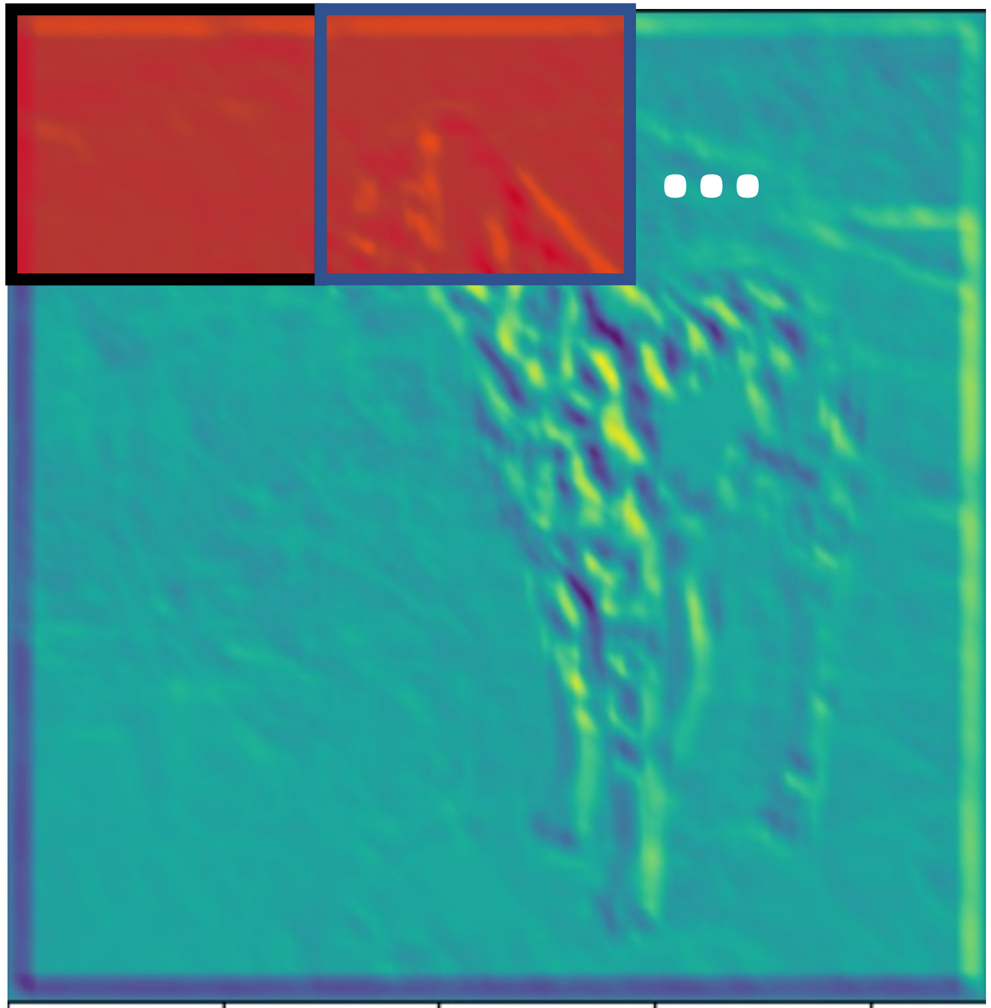
Invariance to distortions



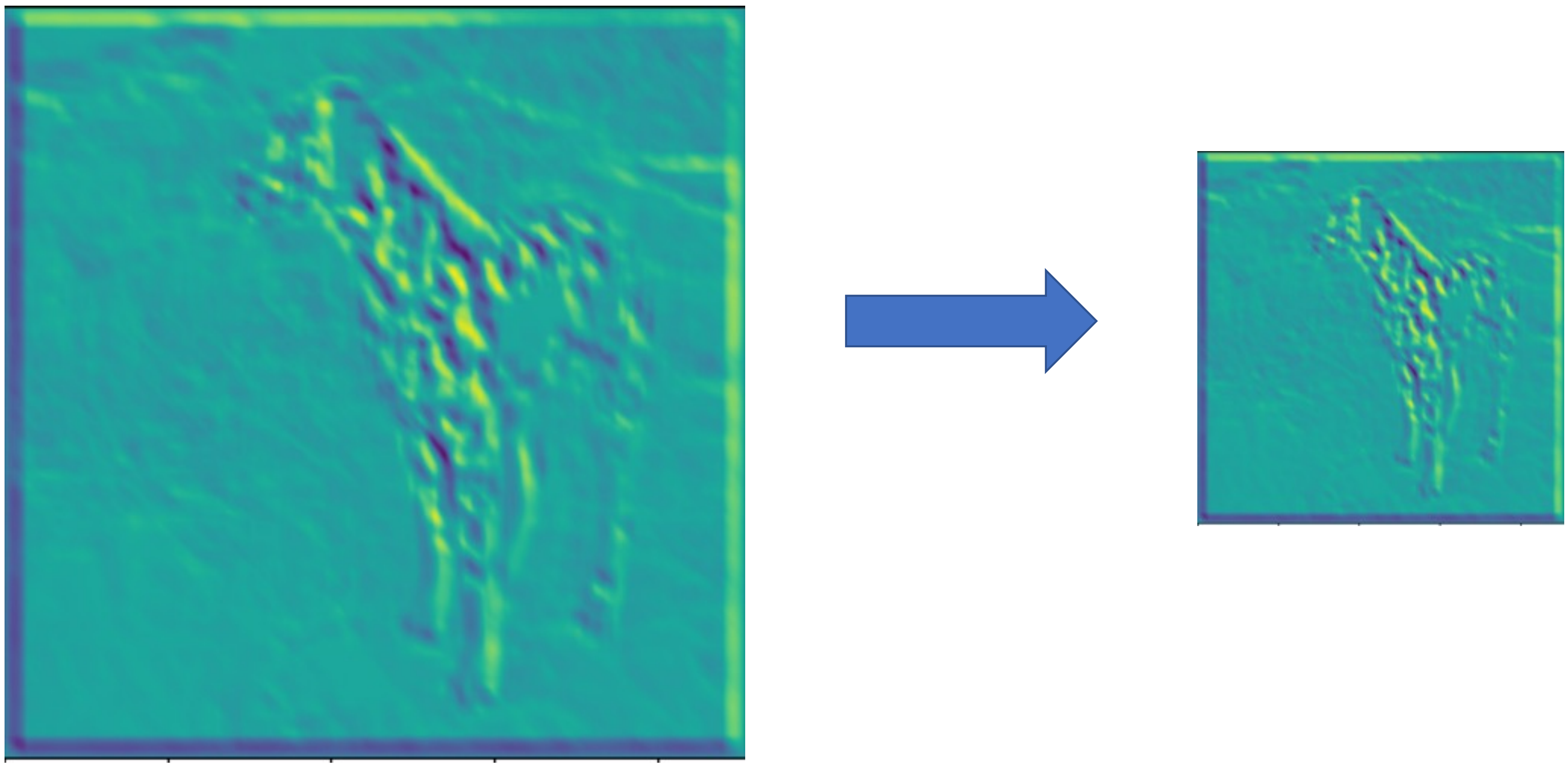
Invariance to distortions



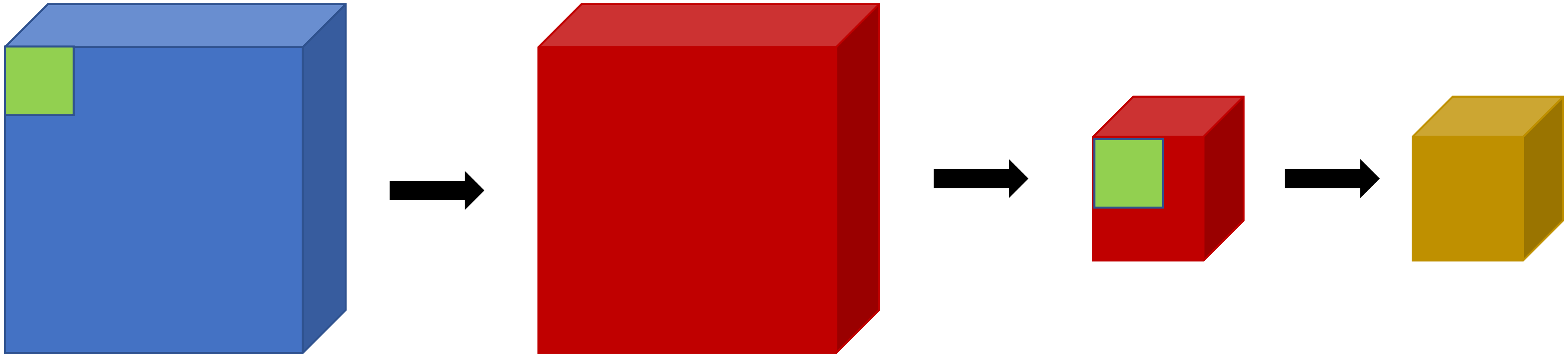
Invariance to distortions: Pooling



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