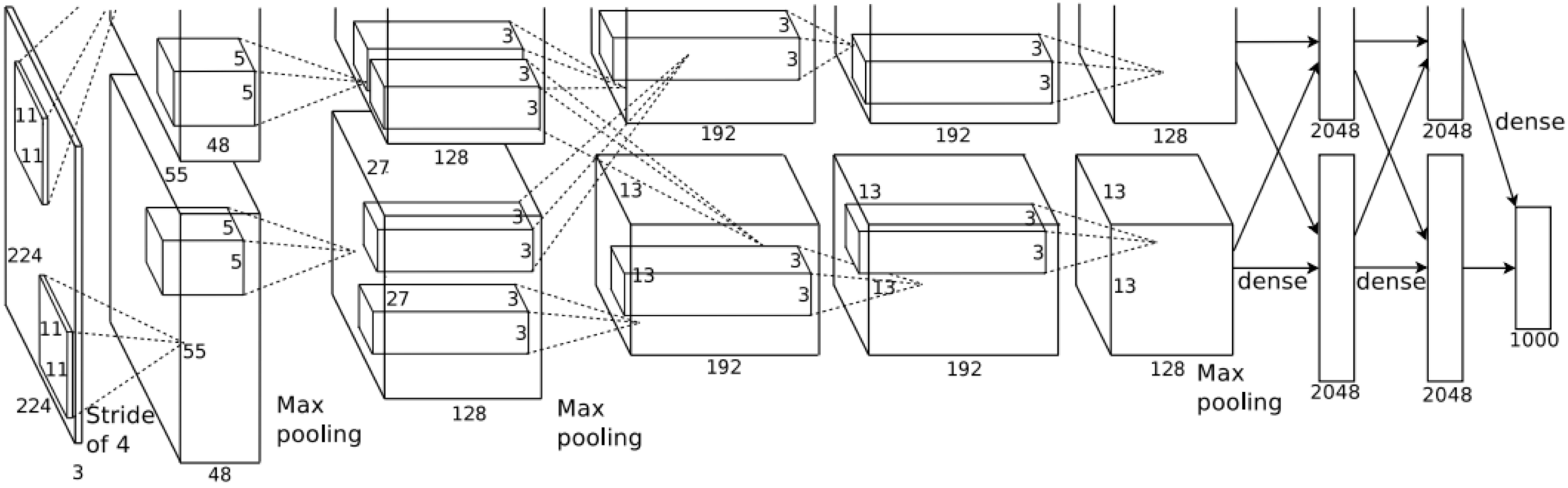
Deep Learning in Comp. Photo

With slides from Phil Isola, James Hays, and Andrea Vedaldi



CS194: Image Manipulation & Computational Photography

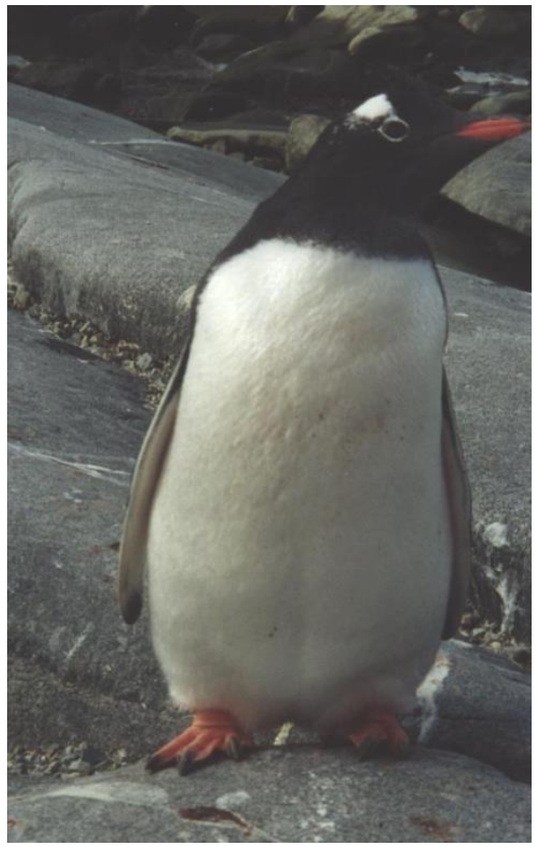
Alexei Efros, UC Berkeley, Fall 2018

Visual similarity via labels

“Penguin”

?

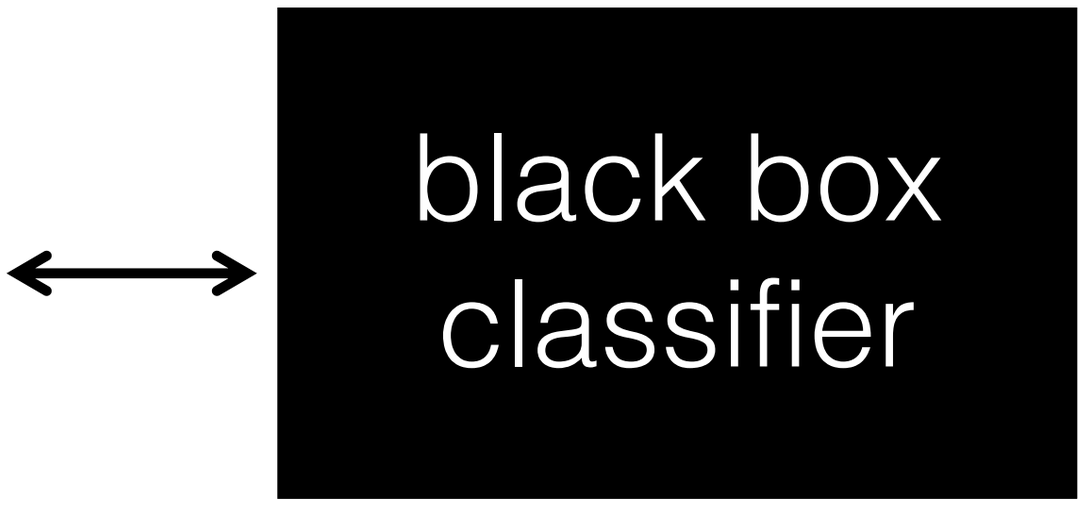
==



“Penguin”

Machine Learning as data association

image X



label Y

At test time…

image X

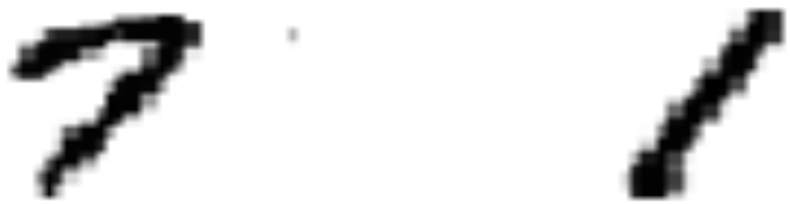
?

Quick Background on Supervised Learning

Jitendra Malik

Fig. 4. Size-normalized examples from the MNIST database.

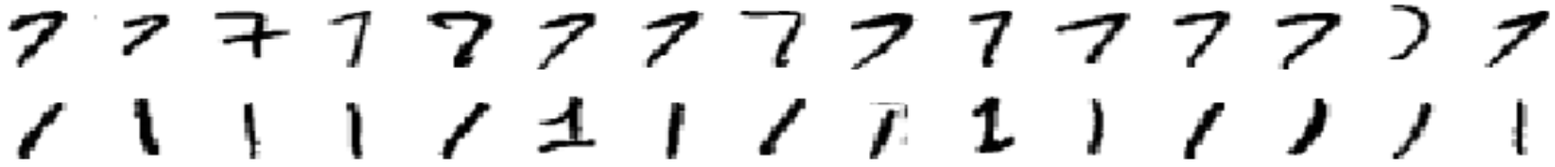
Warm-up Example: Binary Digit Classification



vs.

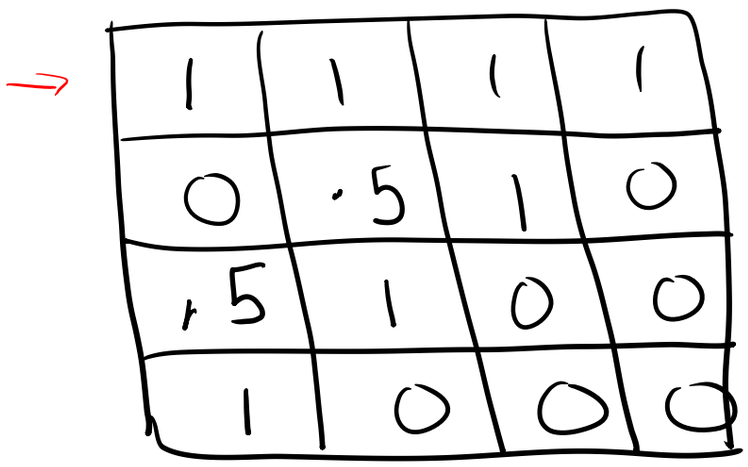
Learning Approach to Object Recognition

- Collect Training Images
  - Positive:
  - Negative:



- Training Time
  - Compute feature vectors for positive and negative example images
  - Train a classifier
- Test Time
  - Compute feature vector on new test image:
  - Evaluate classifier

Let us take an example…



Let us take an example…

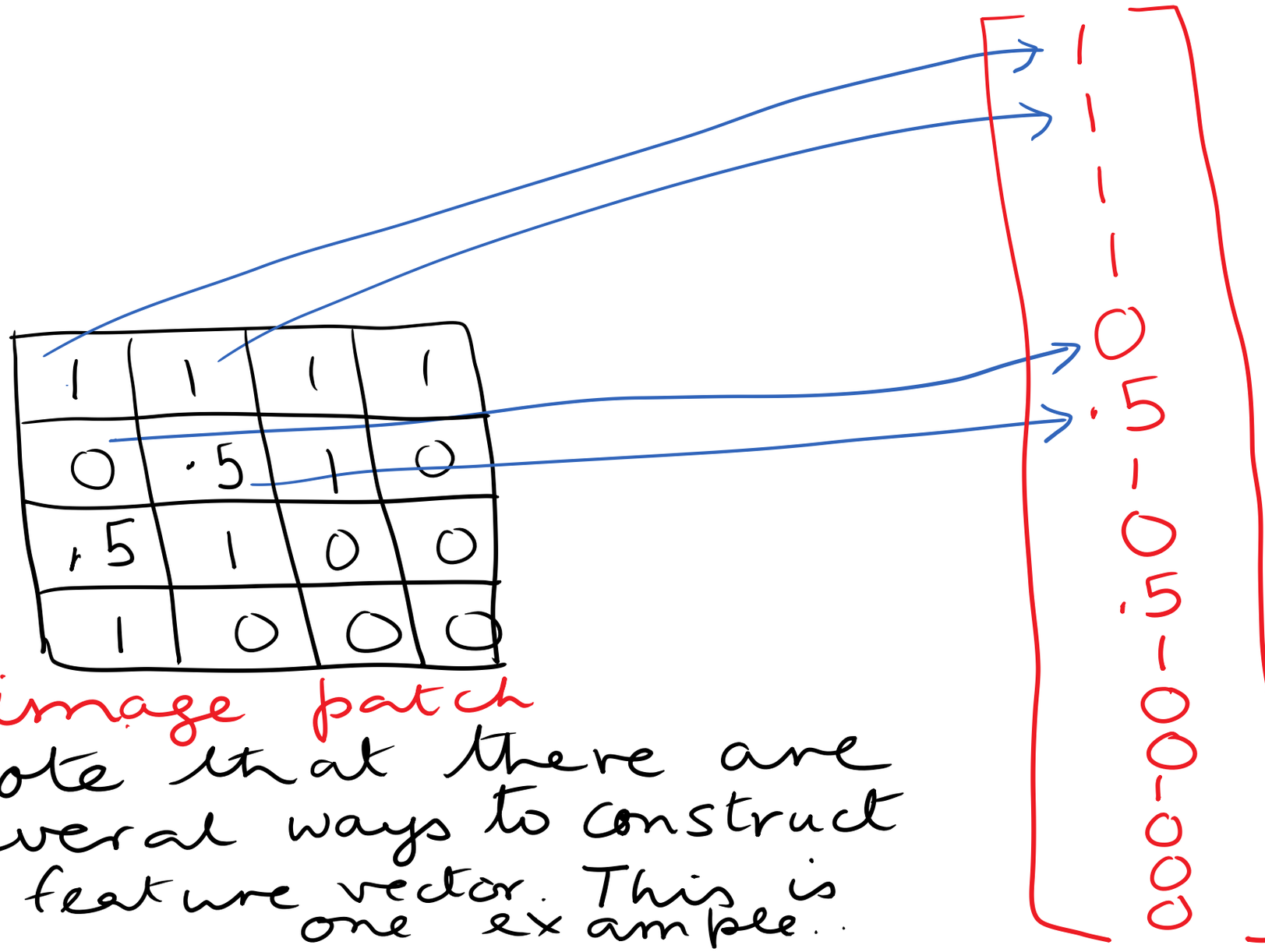
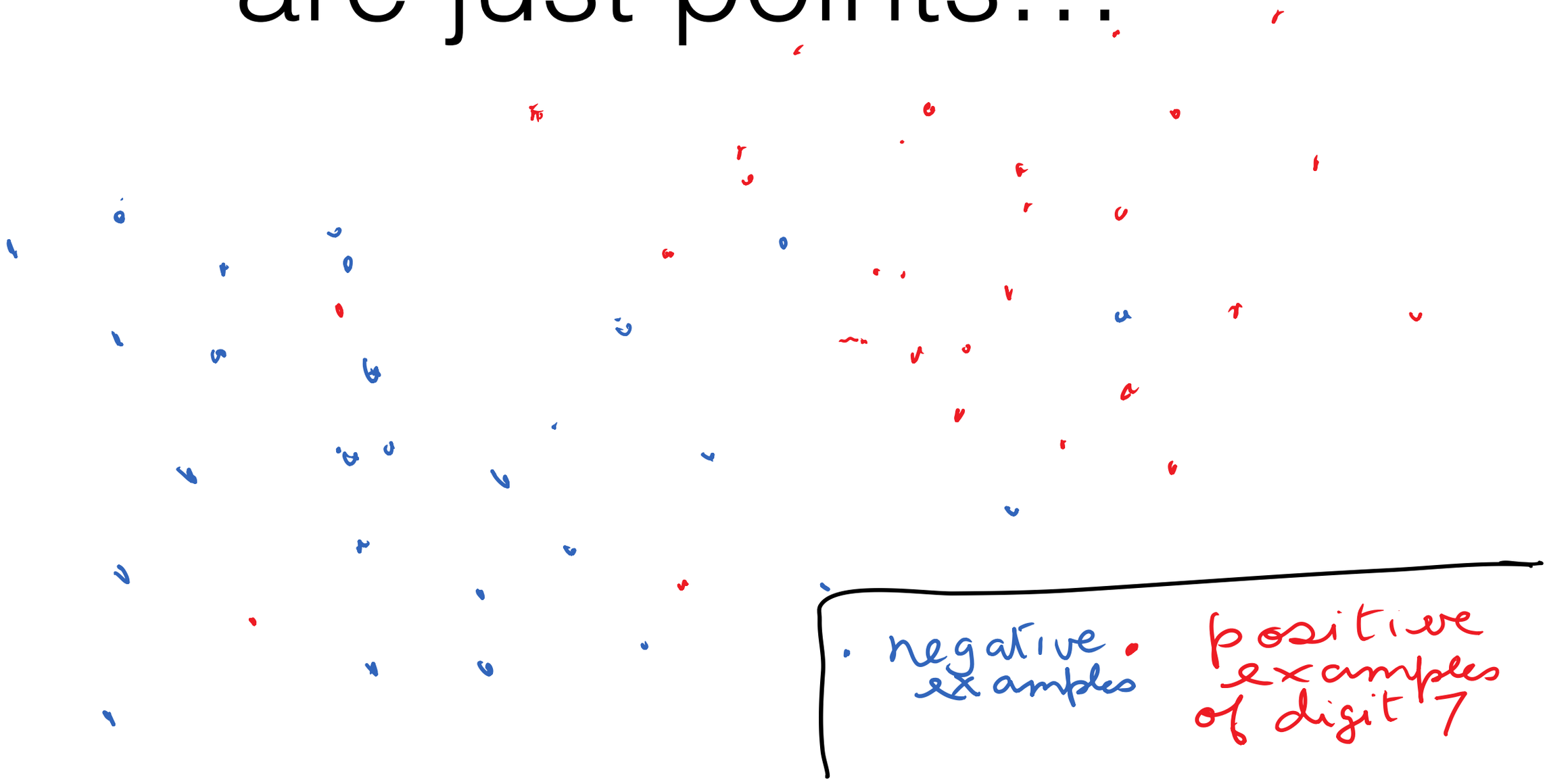


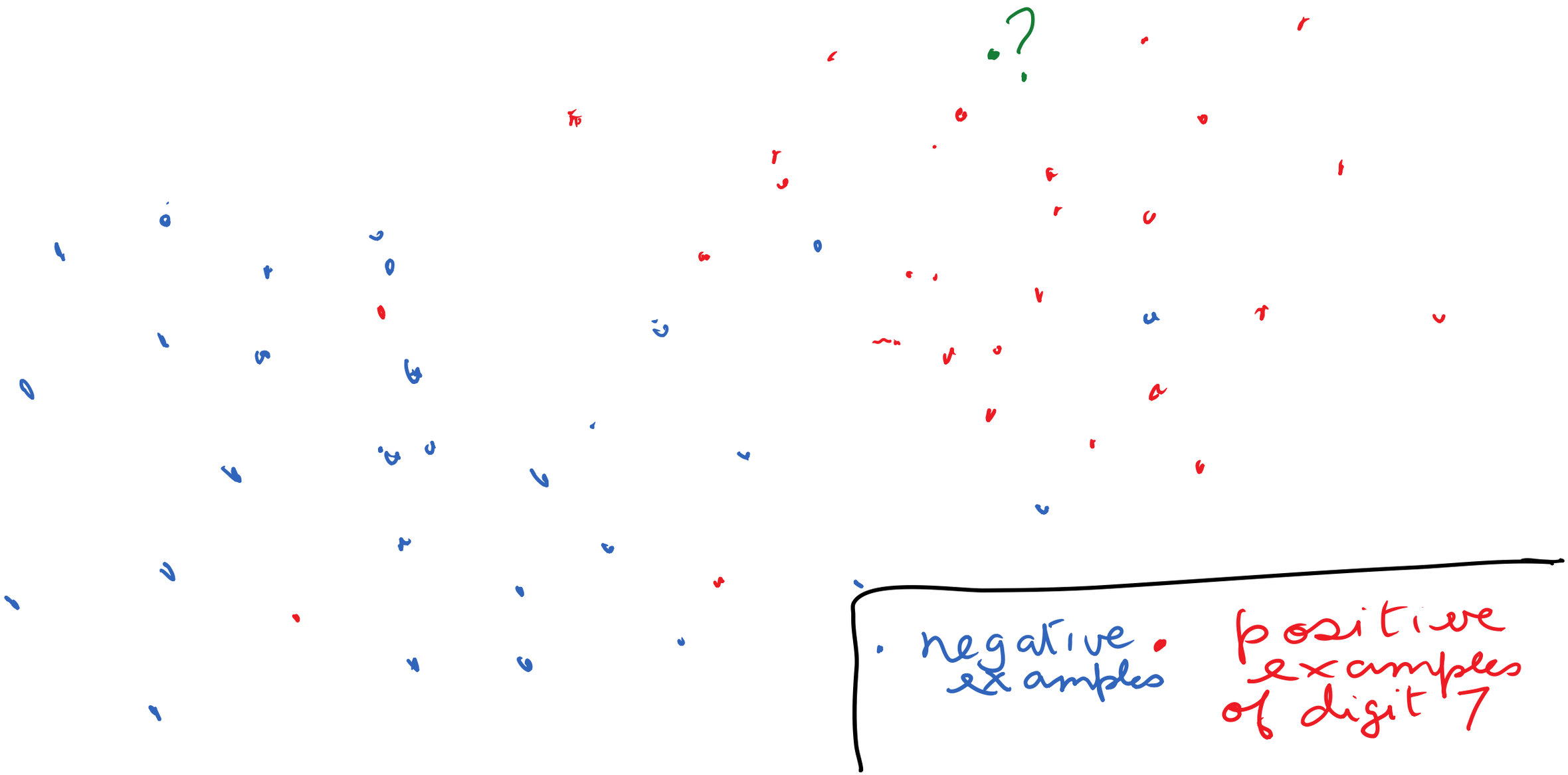
image patch

Note that there are several ways to construct a feature vector. This is one example.

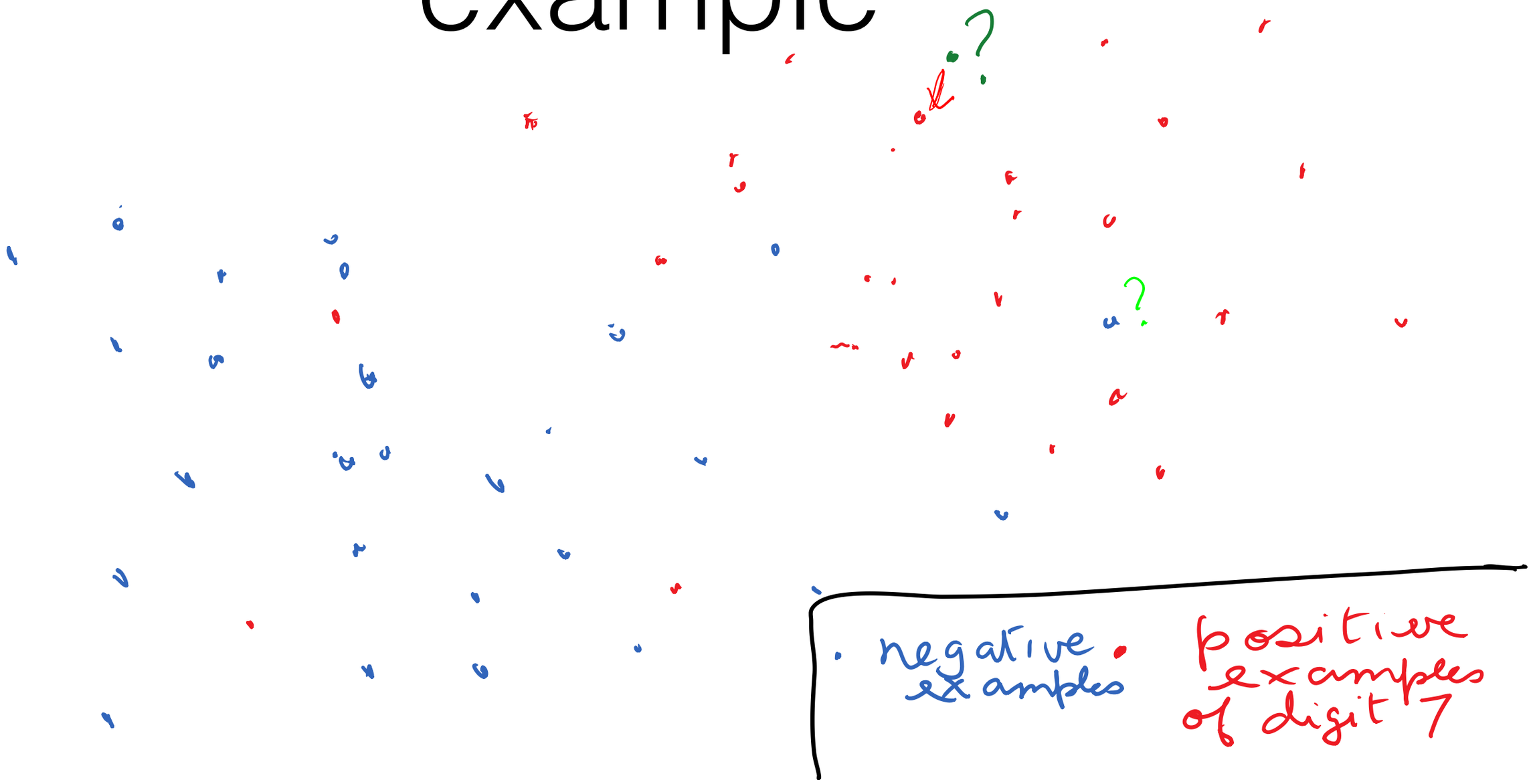
In feature space, positive and negative examples are just points…



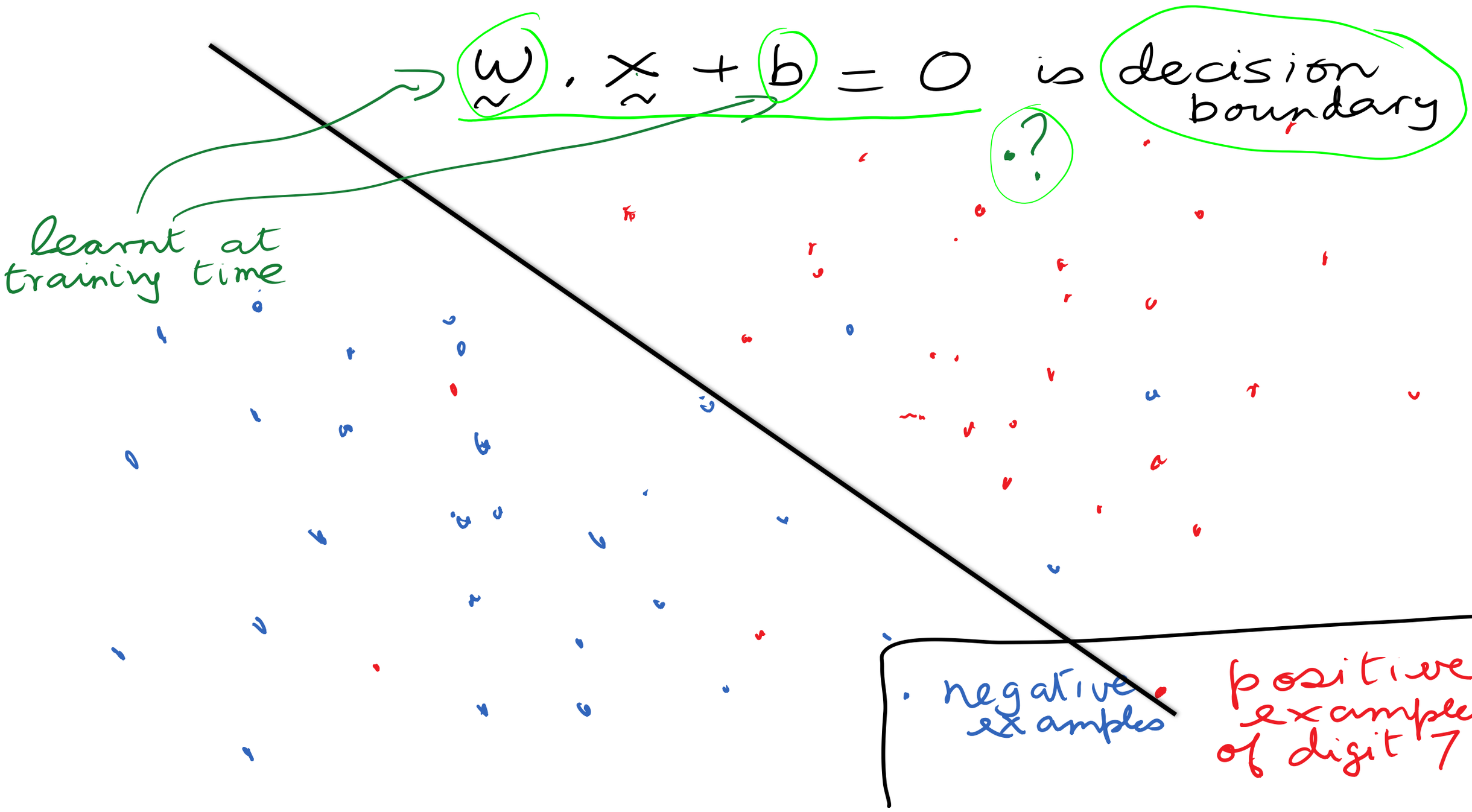
How do we classify a new point?

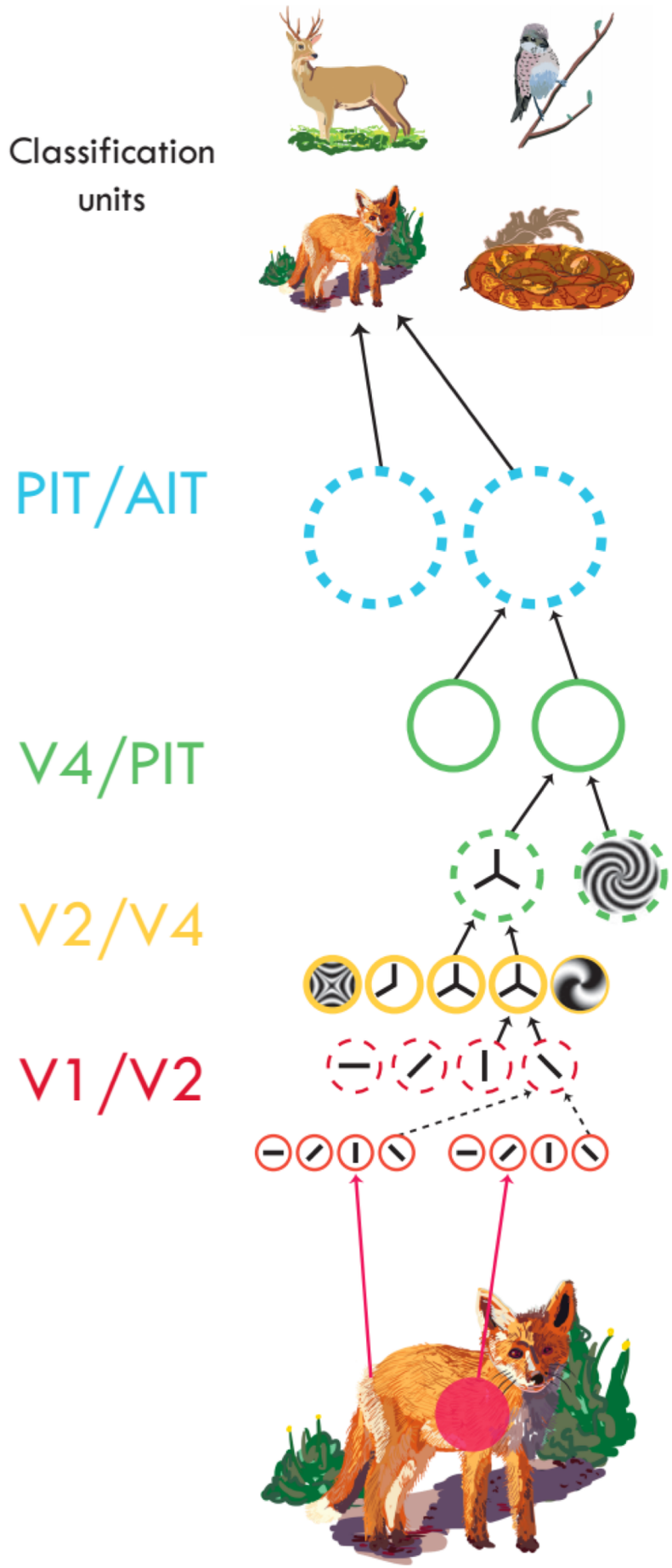


Nearest neighbor rule "transfer label of nearest example"



Linear classifier rule





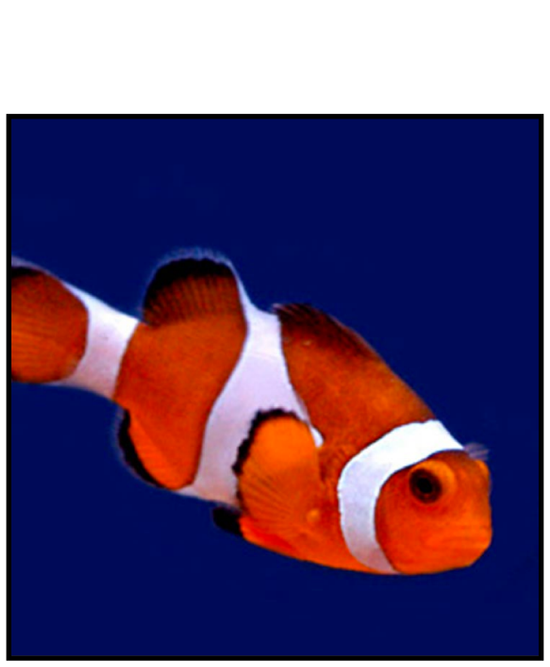
Serre, 2014

Basic idea

Brain/Machine

“clown fish”

Object recognition



Edges

Texture

Colors

Segments

Parts

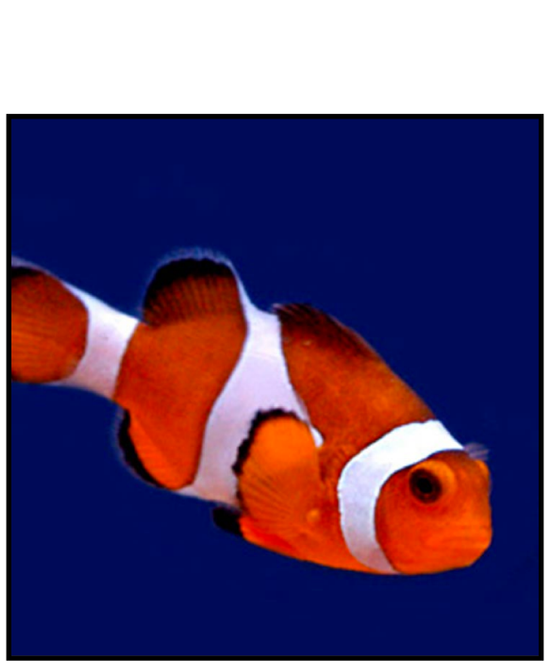
“clown fish”

Feature extractors

Classifier

Object recognition

Learned



Edges

Texture

Colors

Segments

Parts

“clown fish”

Feature extractors

Classifier

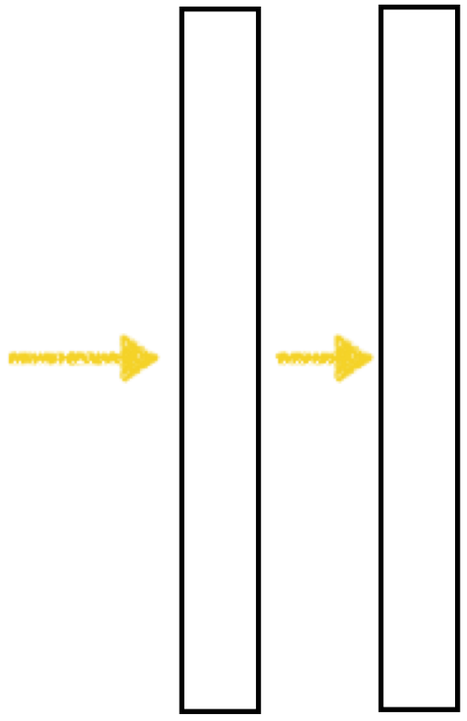
Neural network

Learned

“clown fish”

Neural network

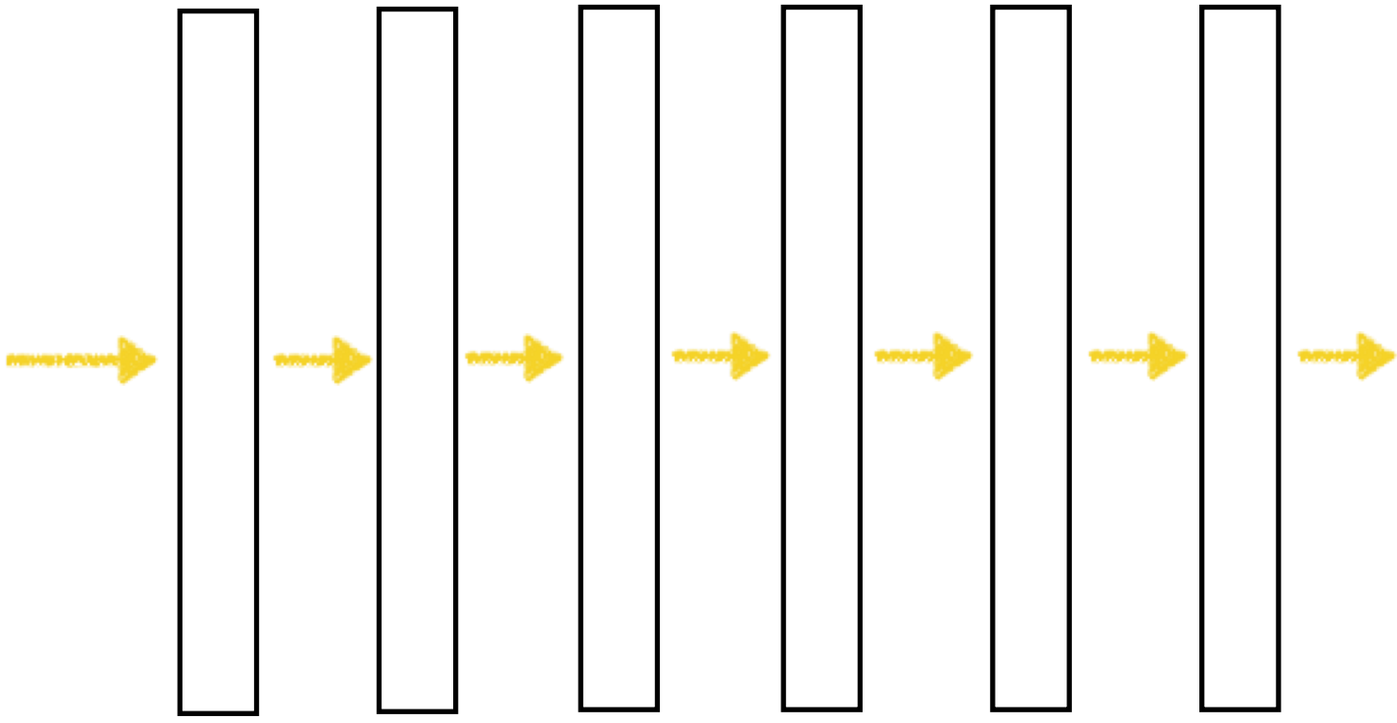
Learned



“clown fish”

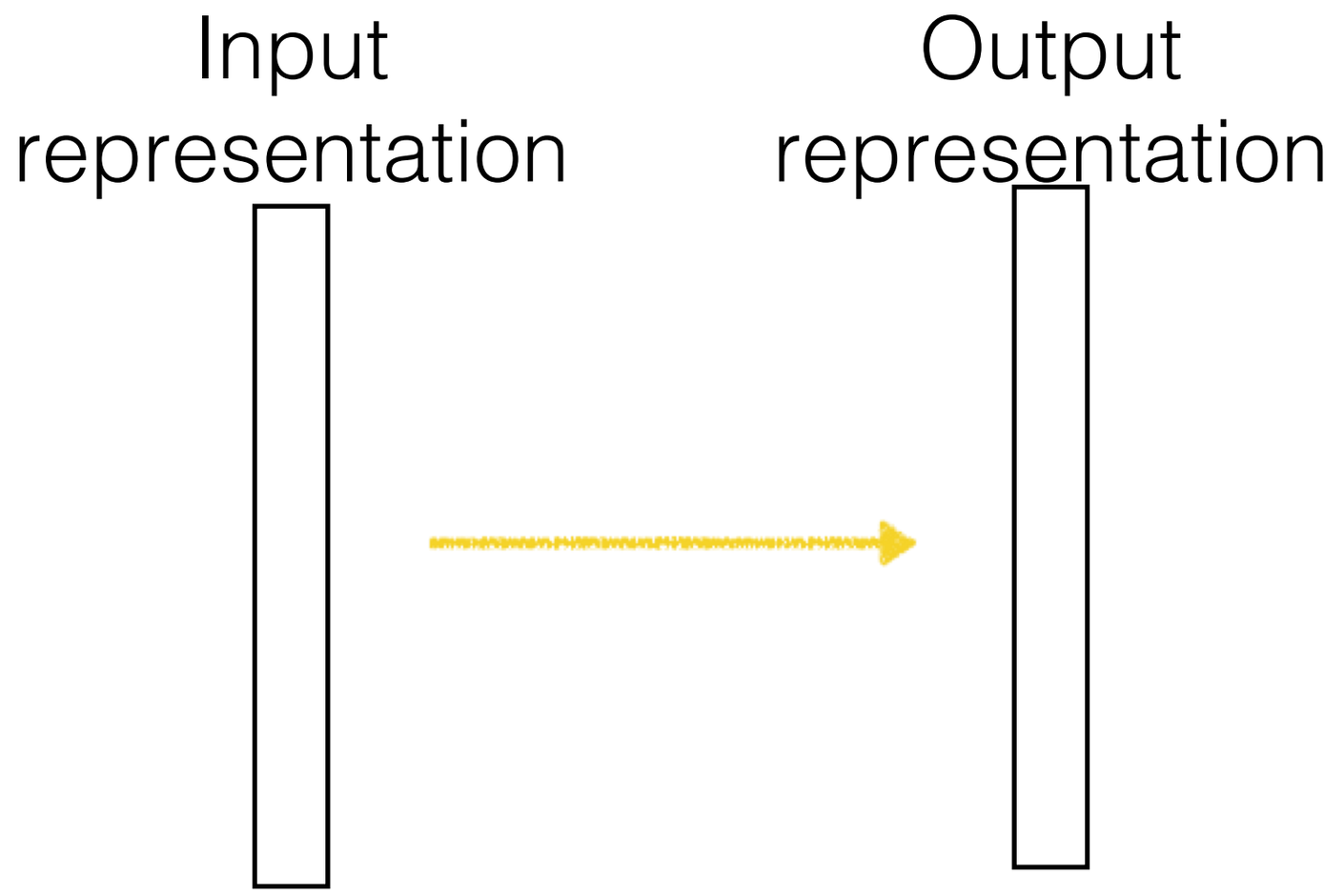
Deep neural network

Learned

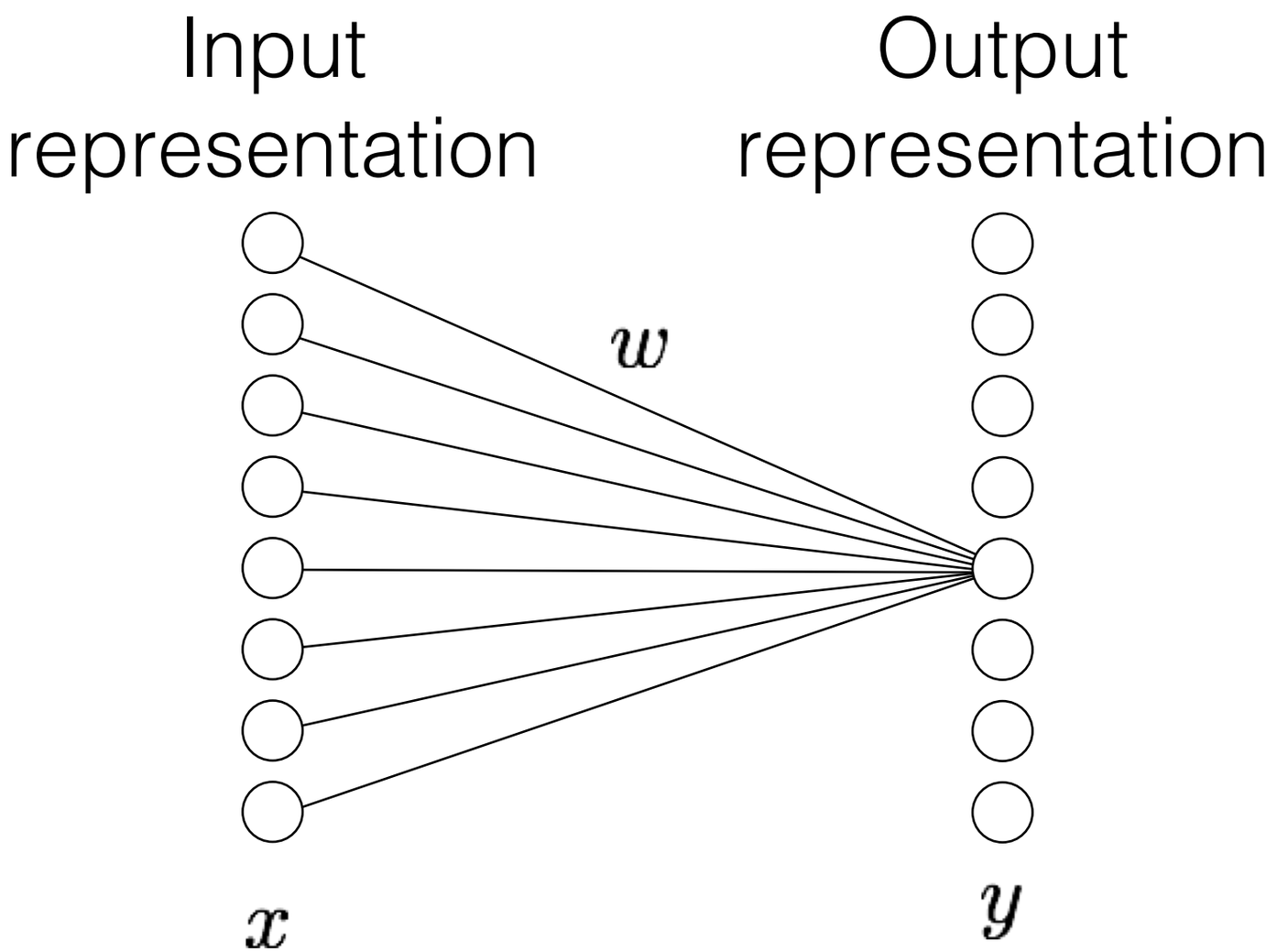


“clown fish”

Computation in a neural net

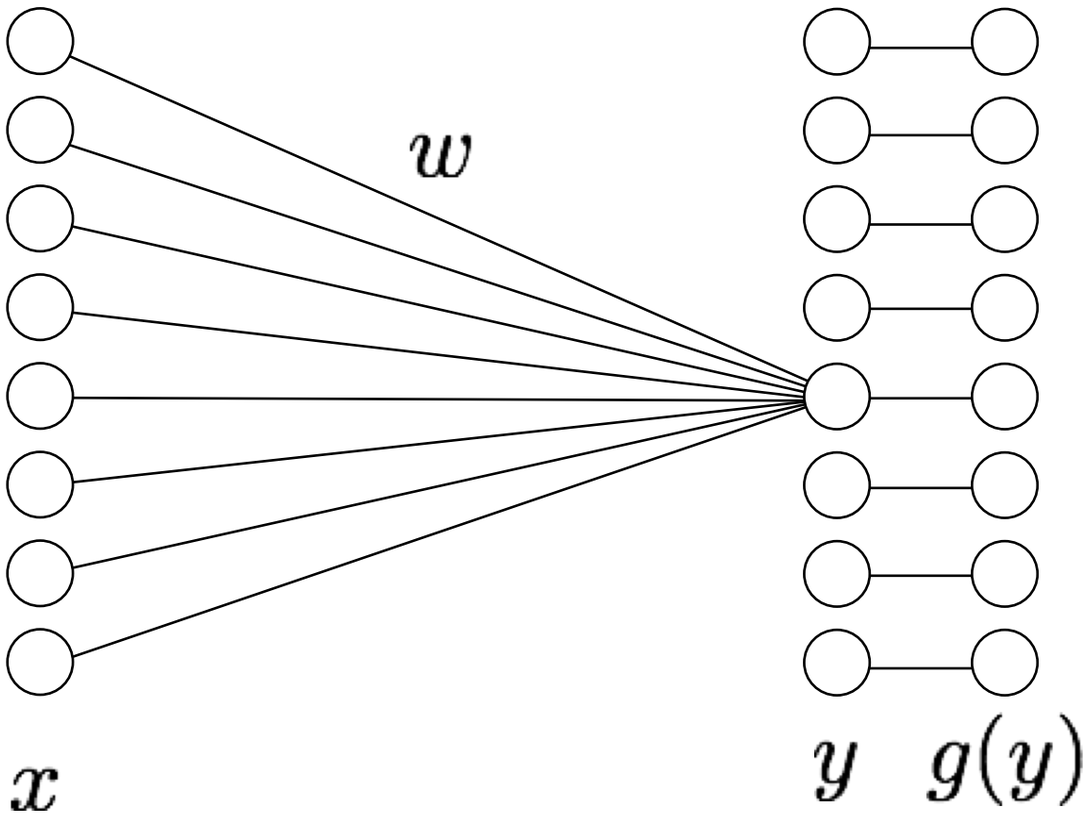


Computation in a neural net

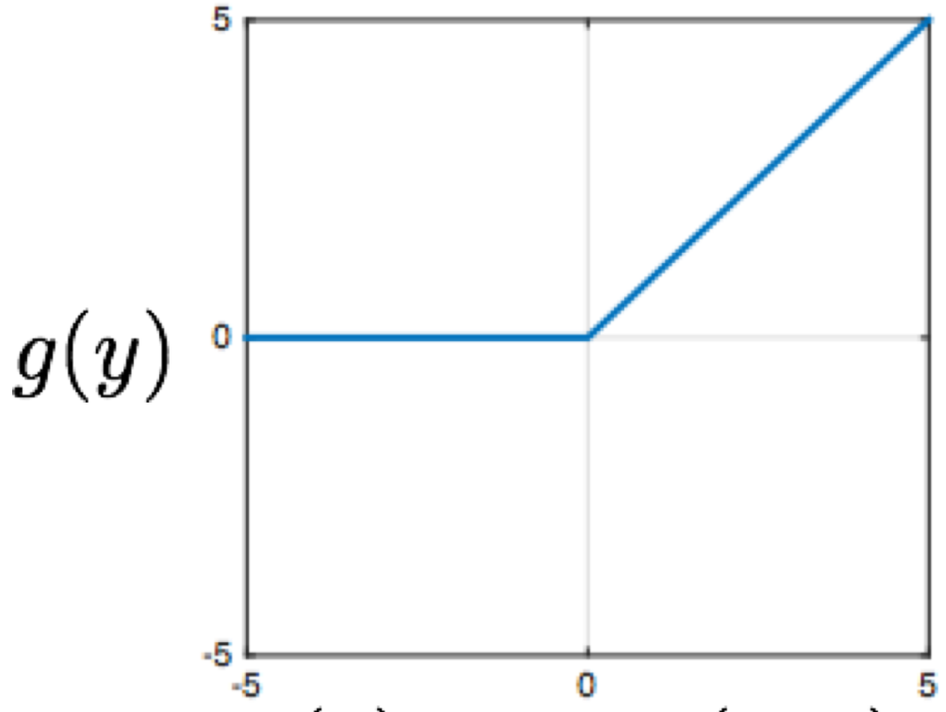


$$y_j = \sum_i w_{ij} x_i$$

Computation in a neural net



Rectified linear unit (ReLU)



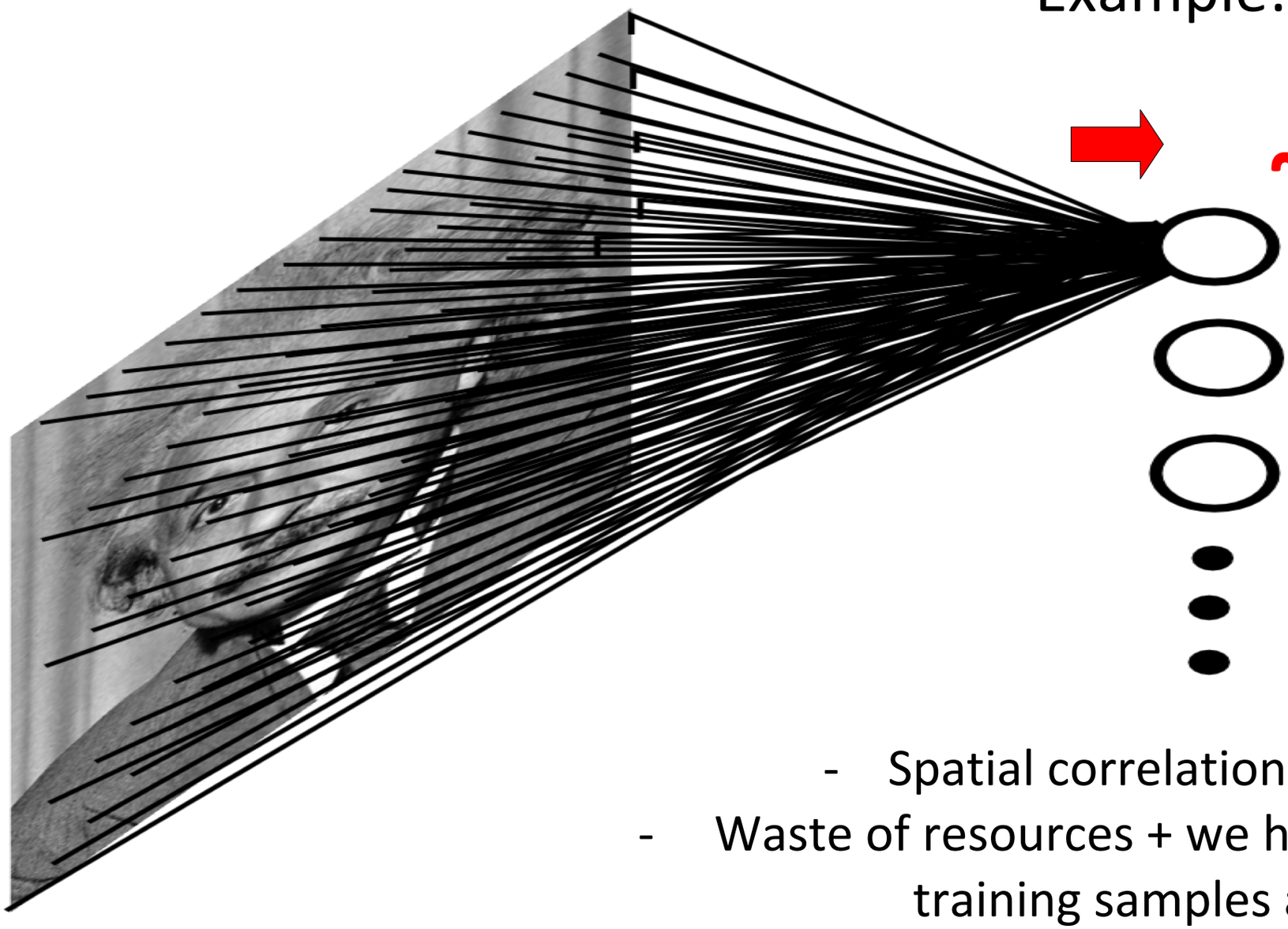
$$g(y) = \max(0, y)$$

Fully Connected Layer

Example: 200x200 image

40K hidden units

~2B parameters!!!

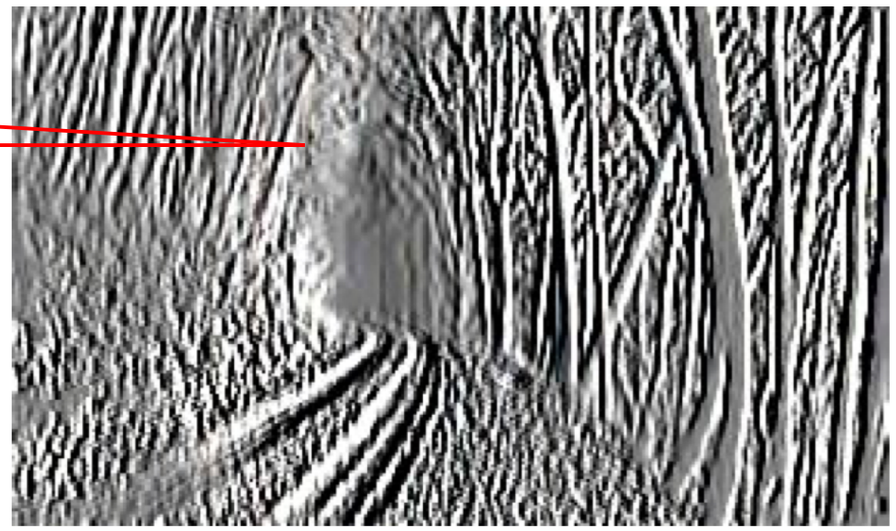


- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Ranzato

Convolutional of Two Signals

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

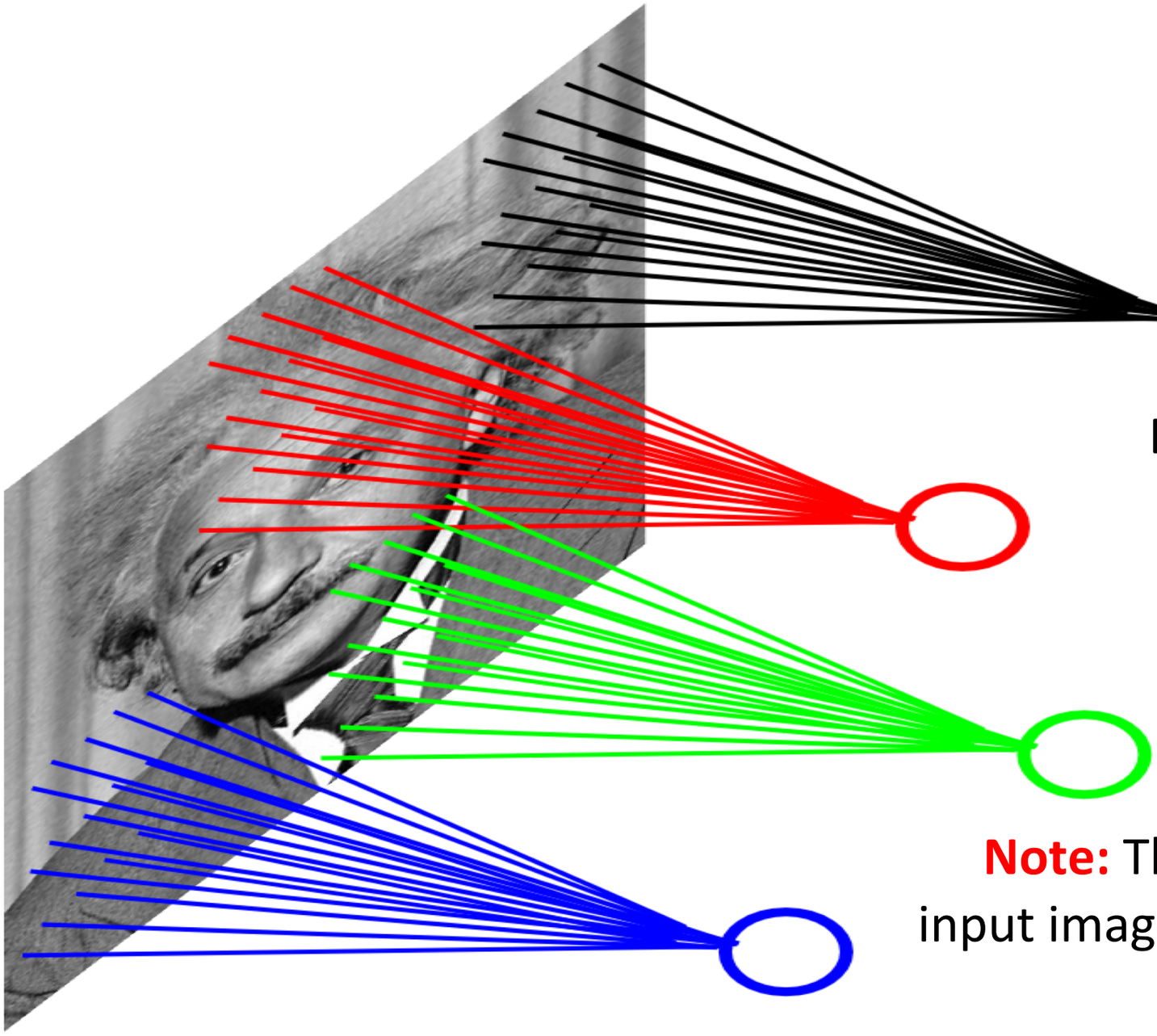


$$g(x,y) = (h * f)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(i,j) f(x-i, y-j)$$

- elementwise multiplication and sum of a filter and the signal (image)

Ranzato

Locally Connected Layer

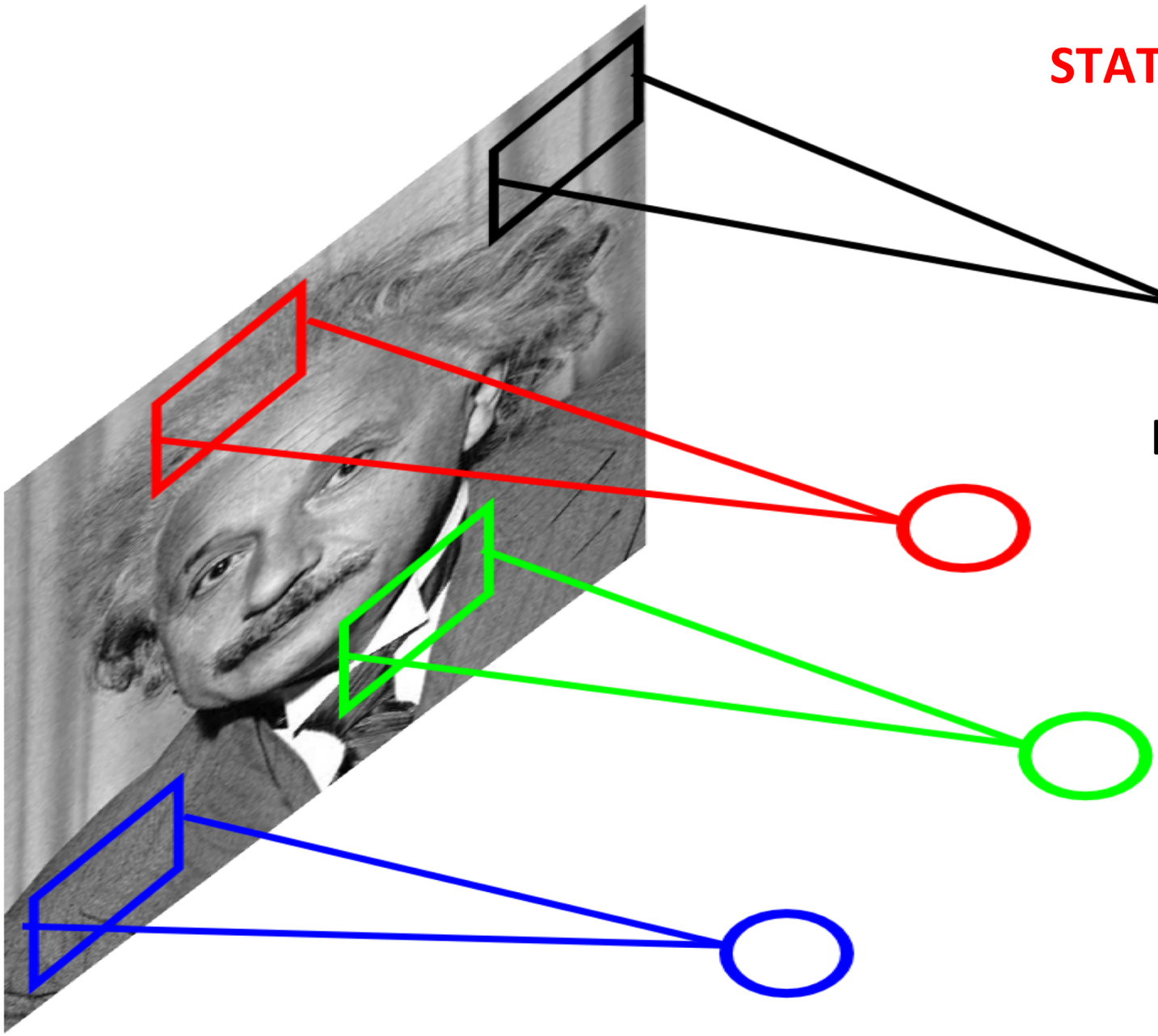


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Ranzato

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Example: 200x200 image

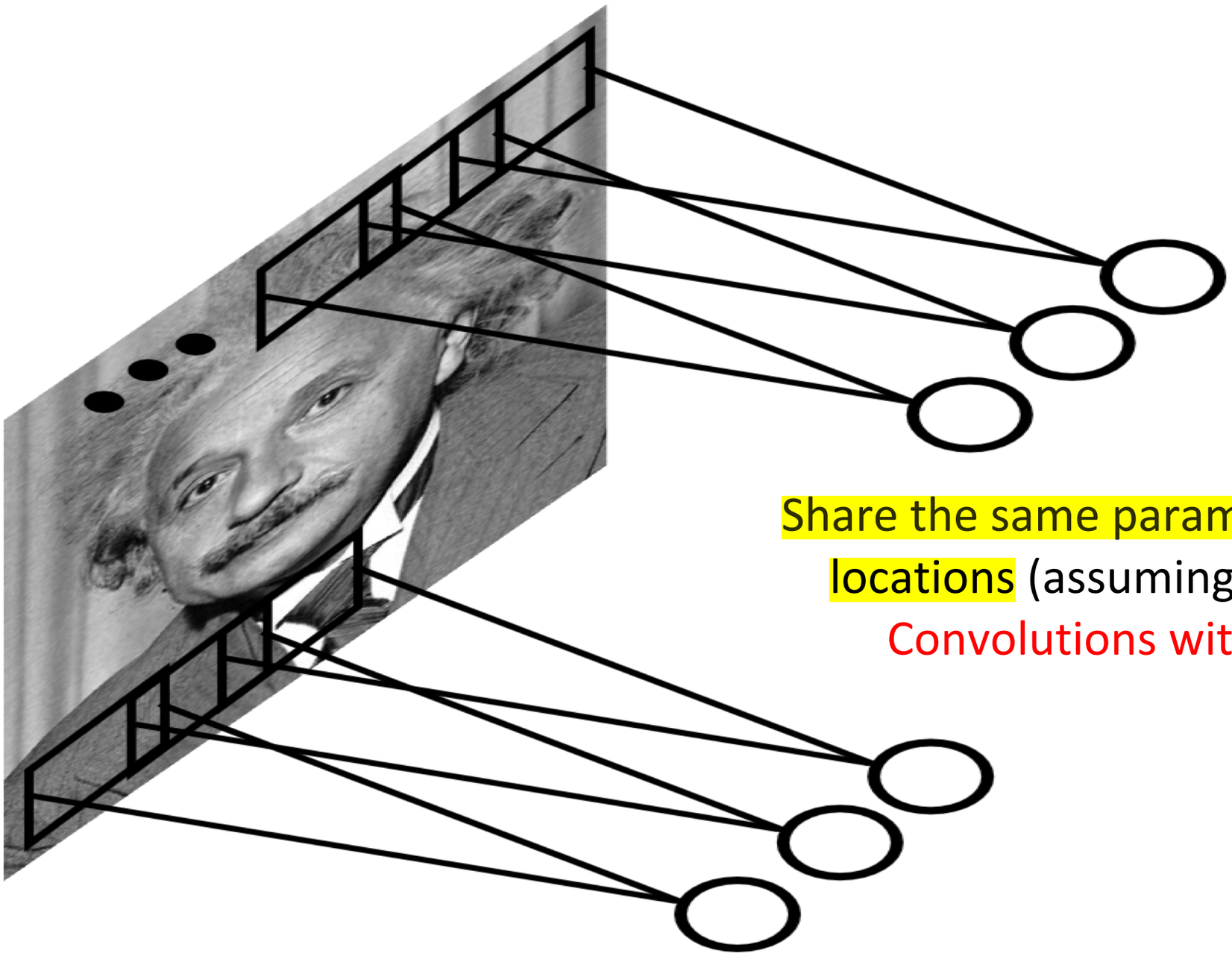
40K hidden units

Filter size: 10x10

4M parameters

Ranzato

Convolutional Layer

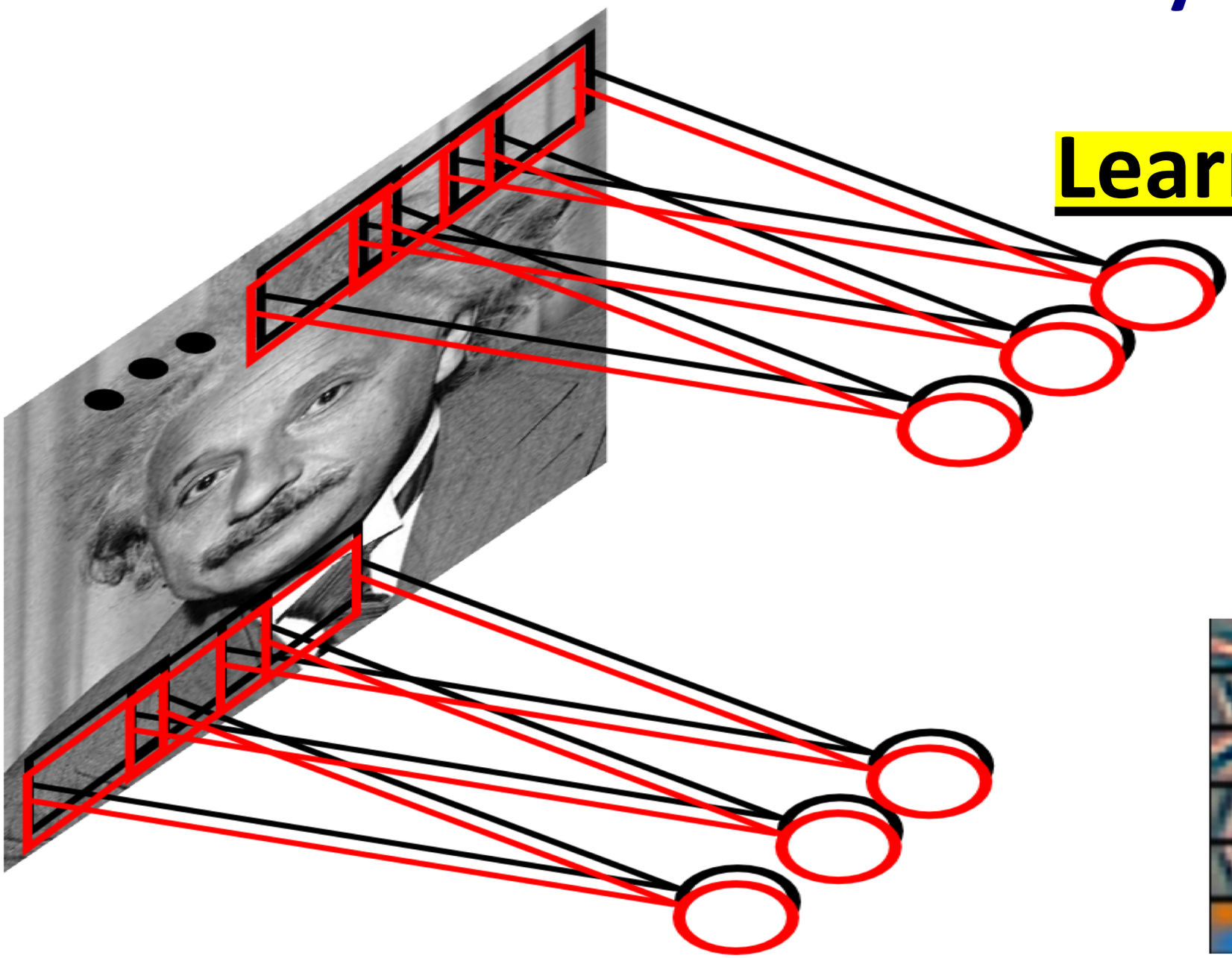


Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

Ranzato

Convolutional Layer

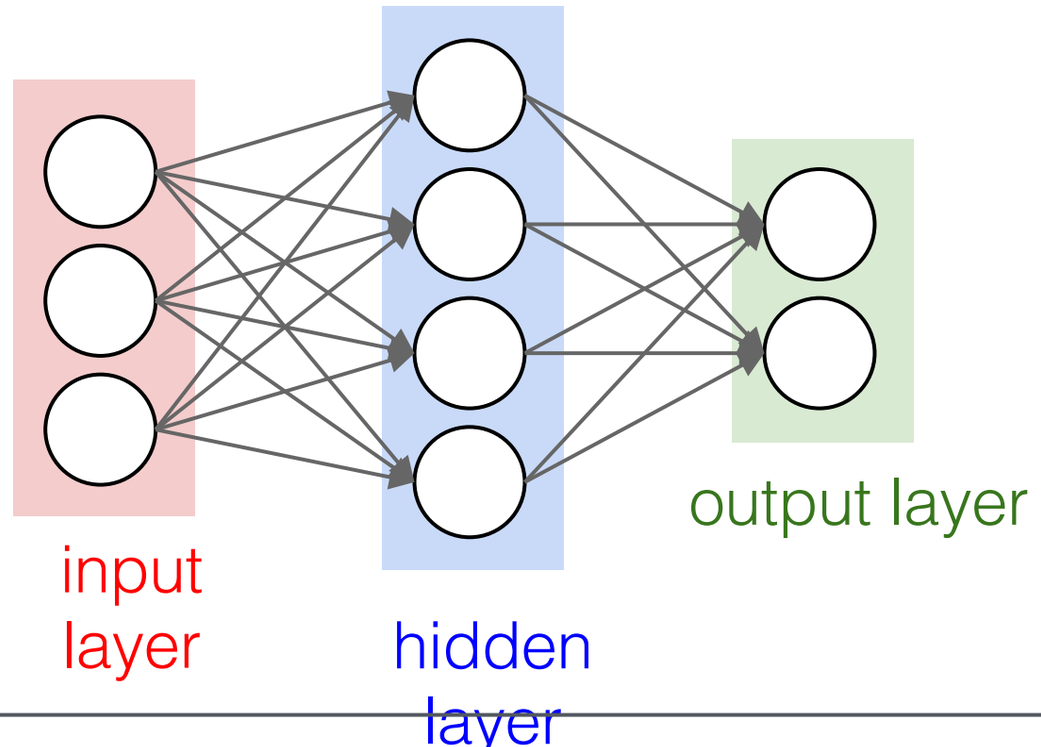


Learn multiple filters.

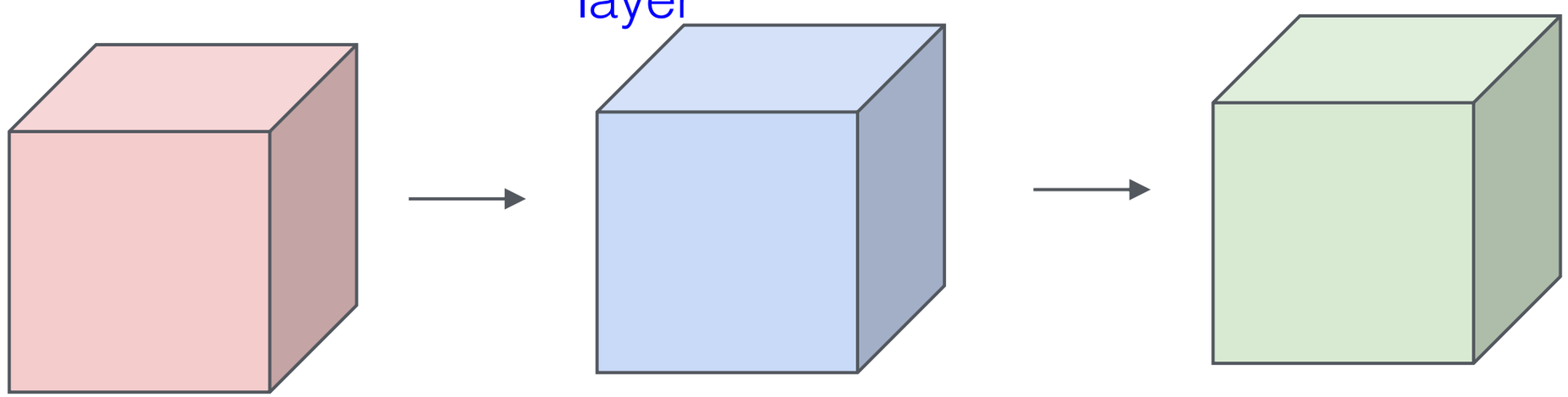
E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

before:

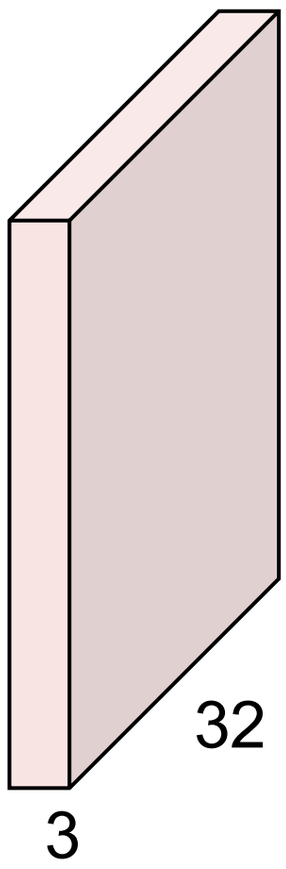


now:



Convolution Layer

32x32x3 image



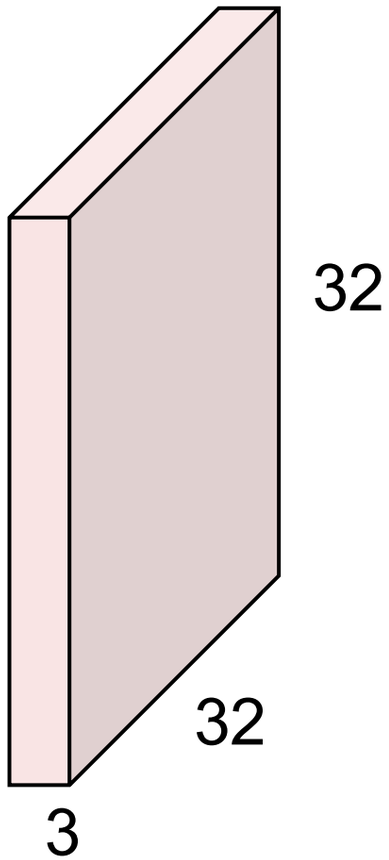
height

width

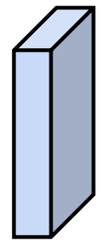
Depth (color)

Convolution Layer

32x32x3 image



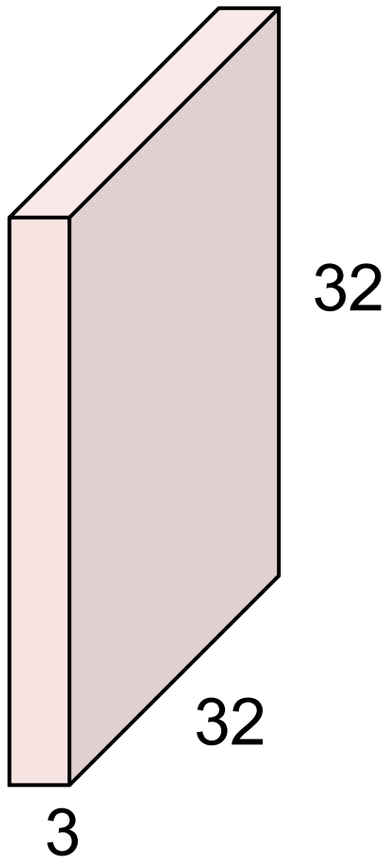
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

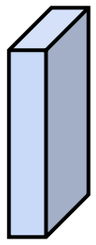
Convolution Layer

32x32x3 image



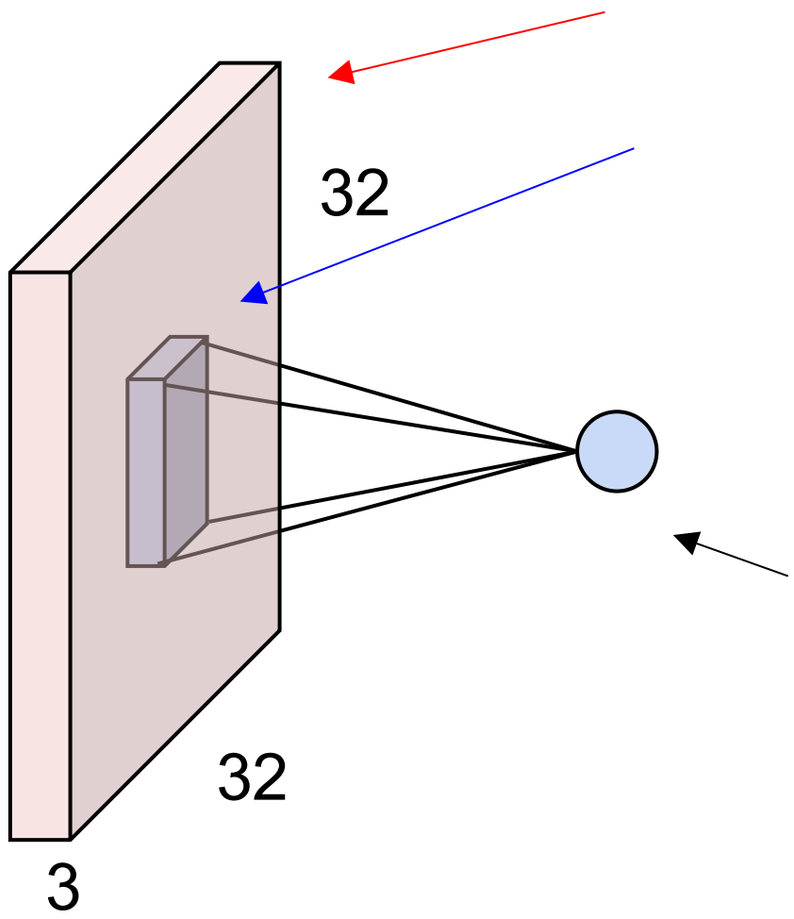
Filters always extend to the full depth of the input volume

5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer



32x32x3 image

5x5x3 filter

w

1 number:

the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

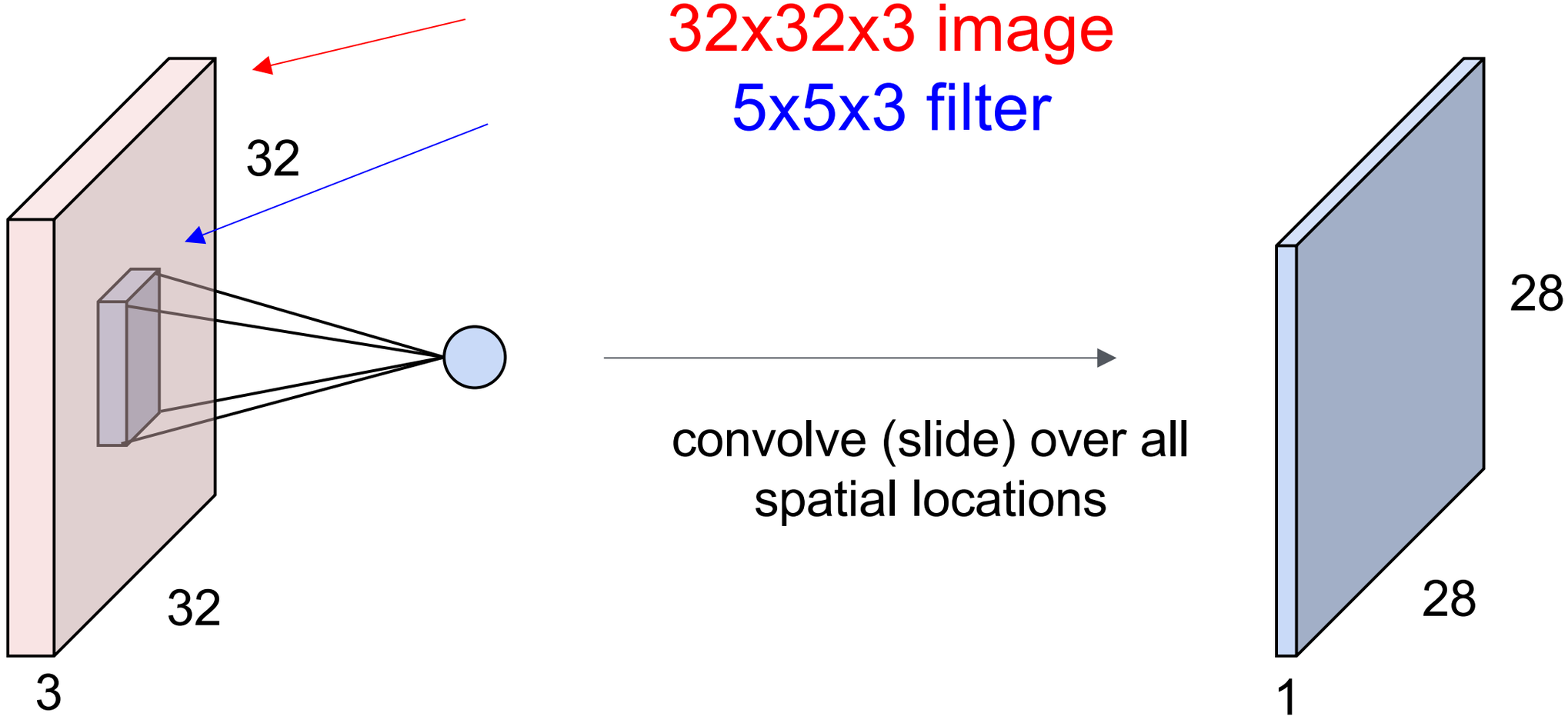
$$w^T x + b$$

1 channel x 1 kernel = 1 matrix
add 3 matrices = activation map

Convolution Layer

activation map

32x32x3 image
5x5x3 filter

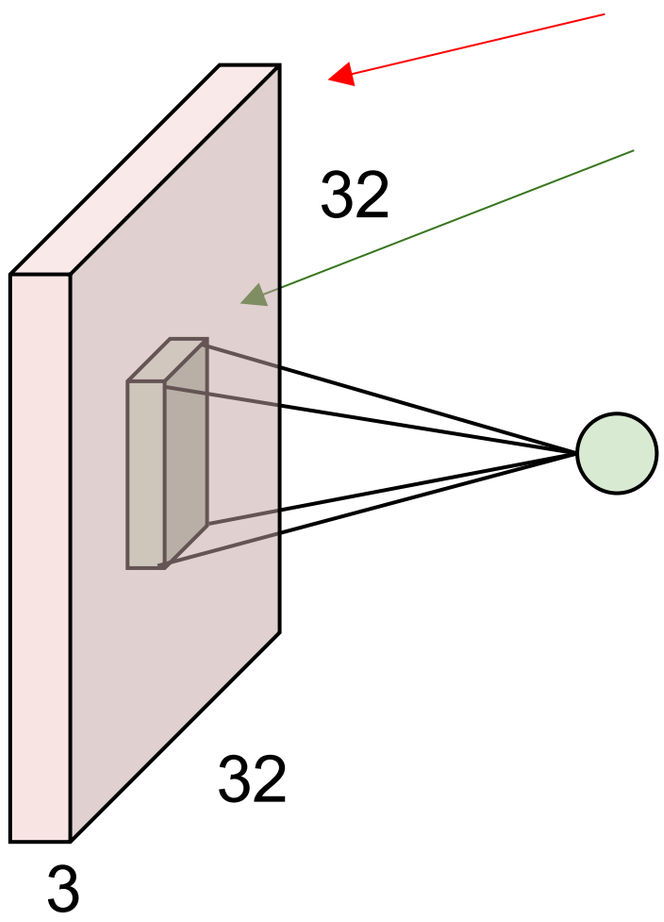


Convolution Layer

consider a second, green filter

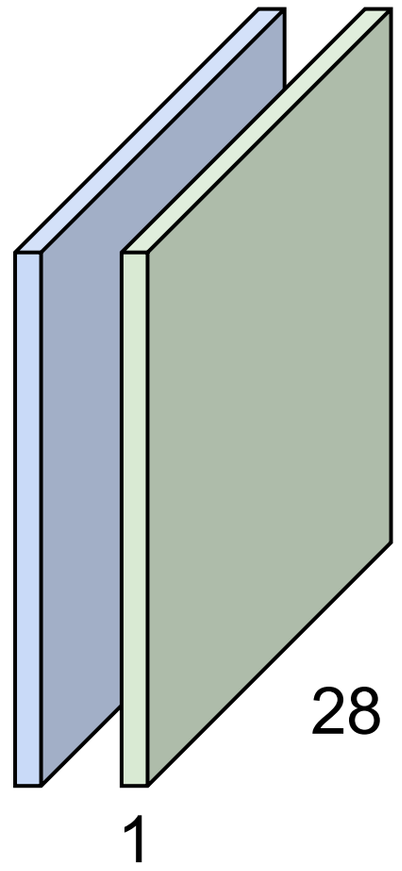
32x32x3 image

5x5x3 filter

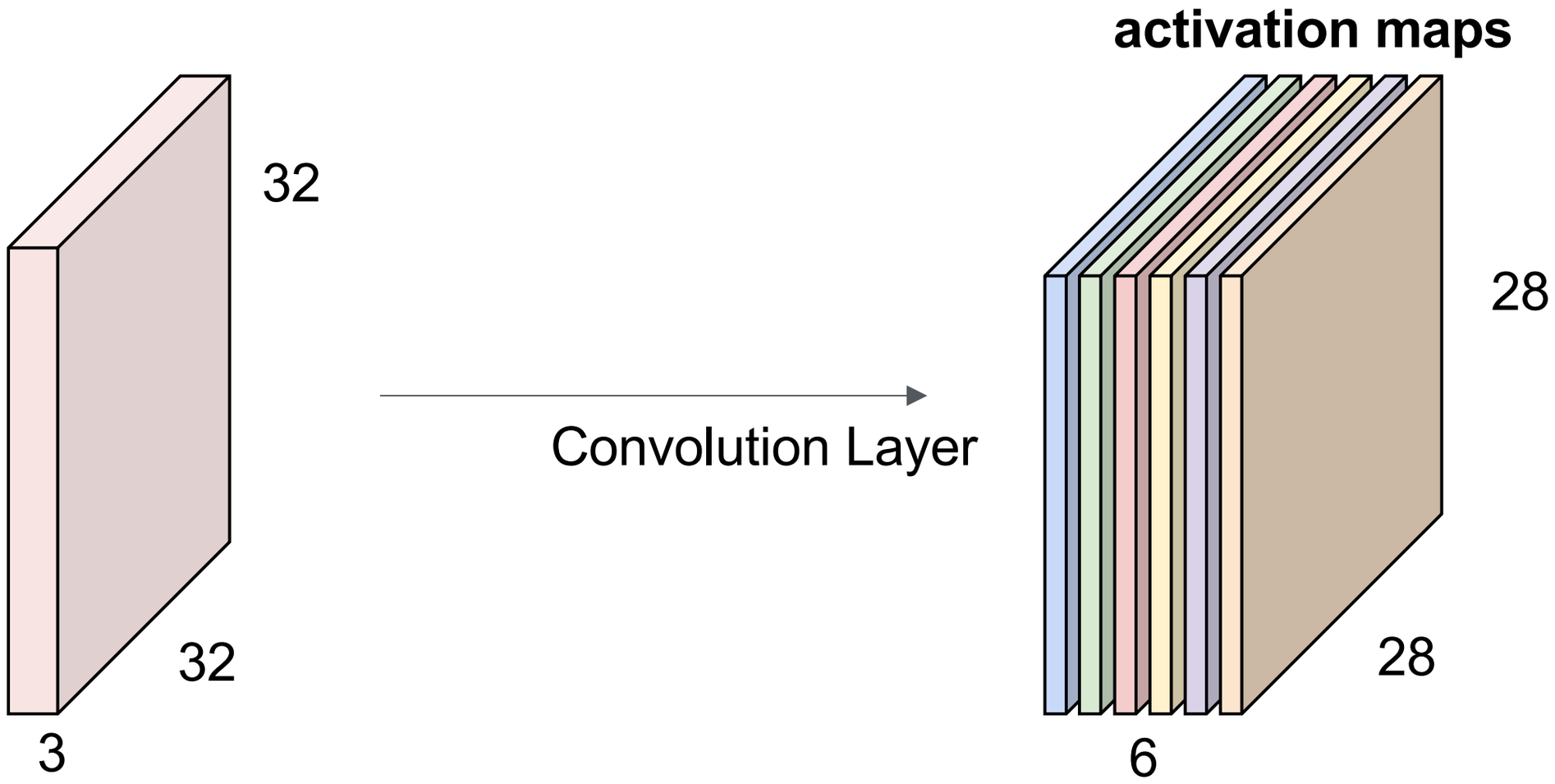


convolve (slide) over all spatial locations

activation maps

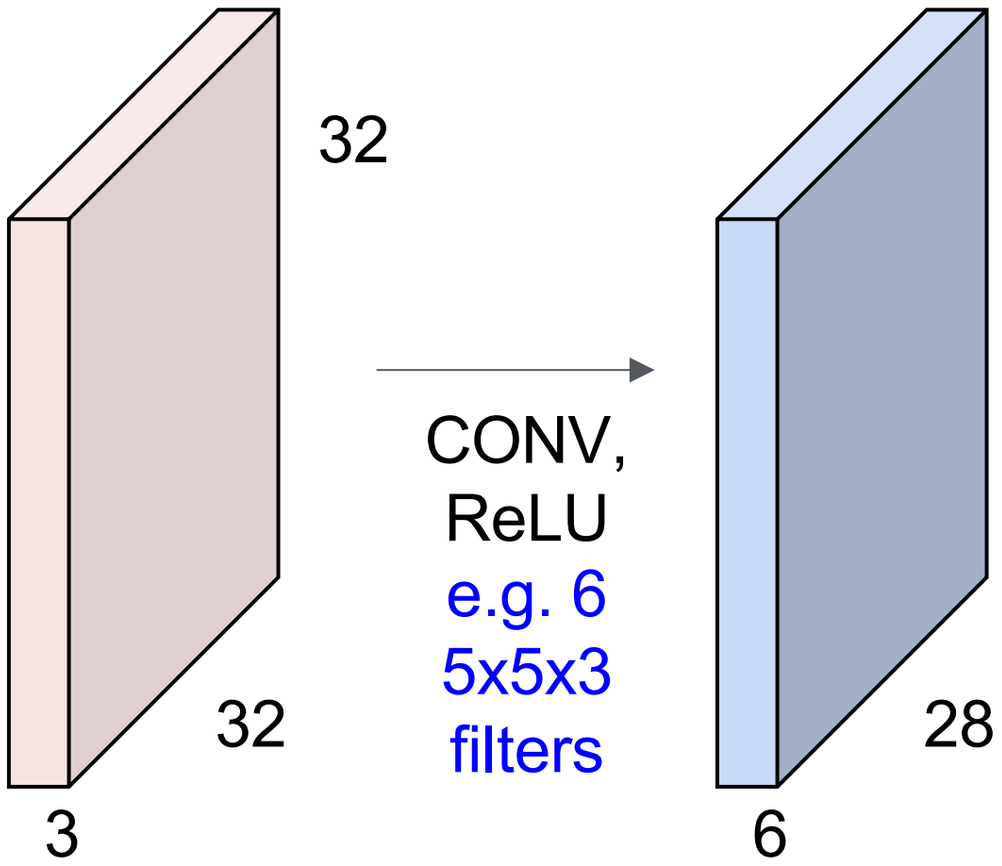


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

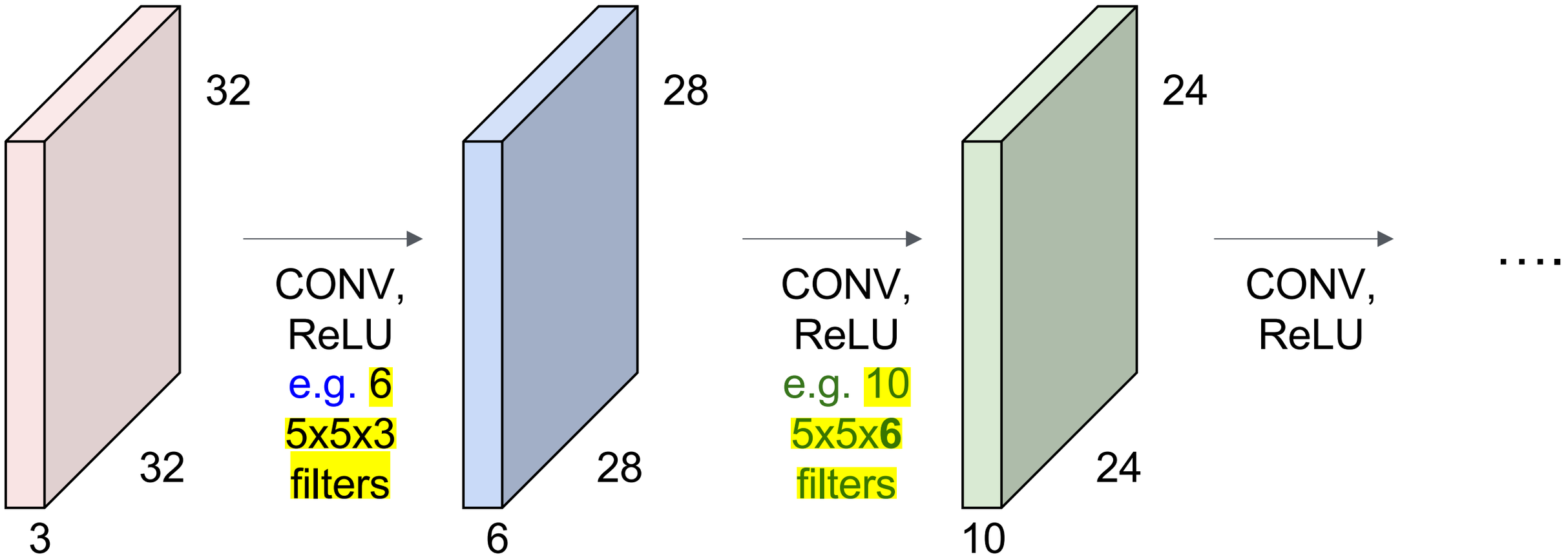


We stack these up to get a "new image" of size 28x28x6!

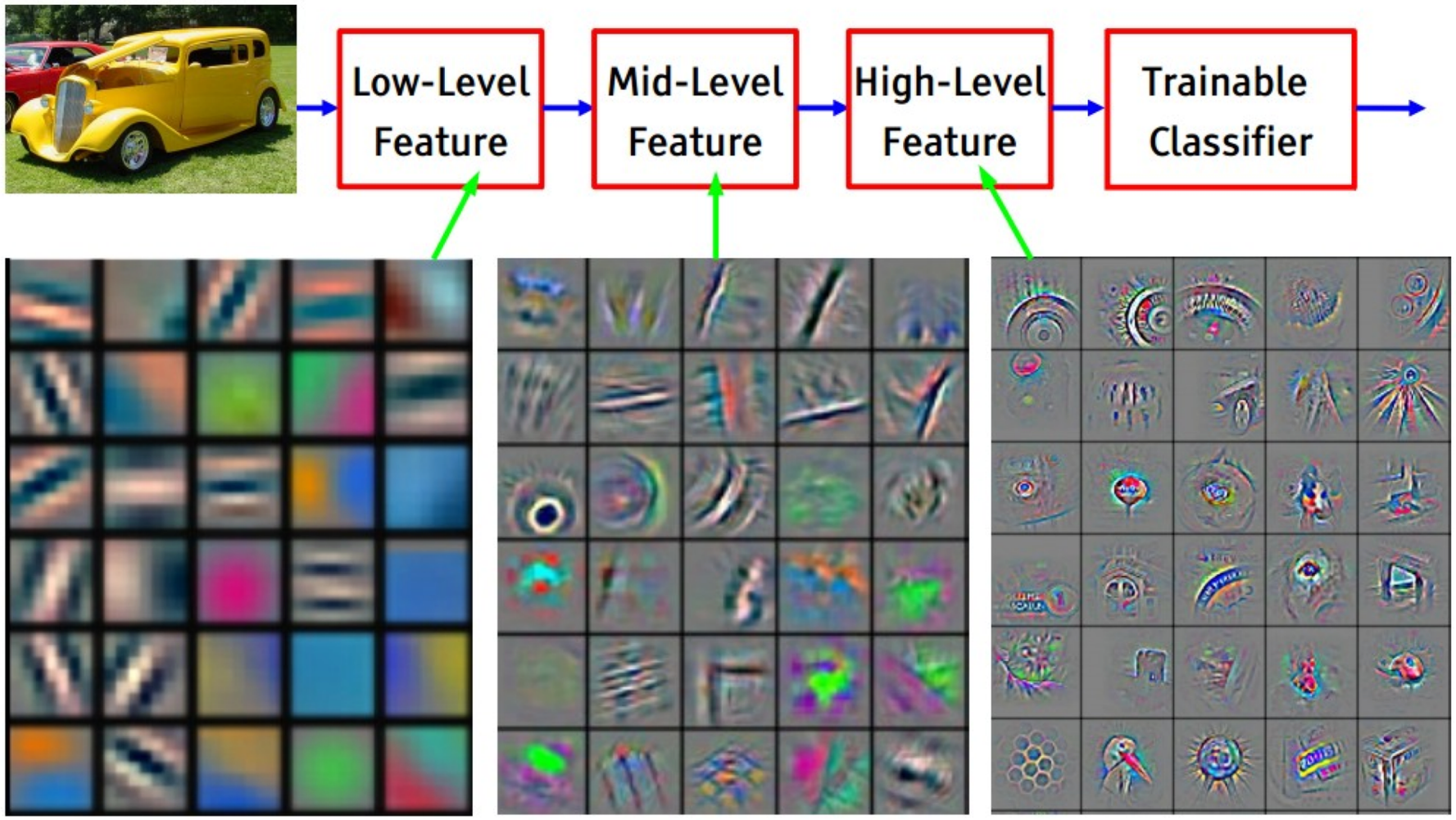
Preview: ConvNet is a sequence of Convolution Layers, interspersed with non-linear activation functions



Preview: ConvNet is a sequence of Convolution Layers, interspersed with non-linear activation functions



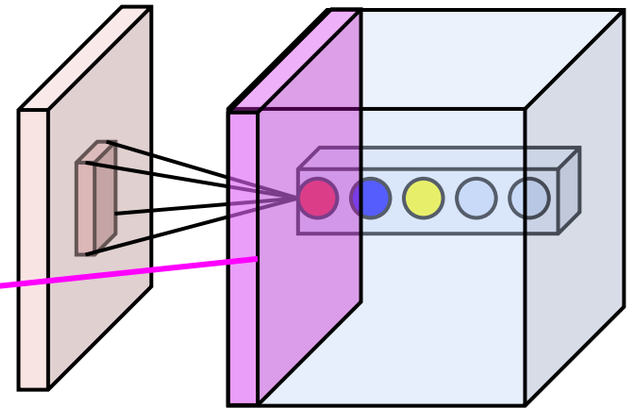
Recall



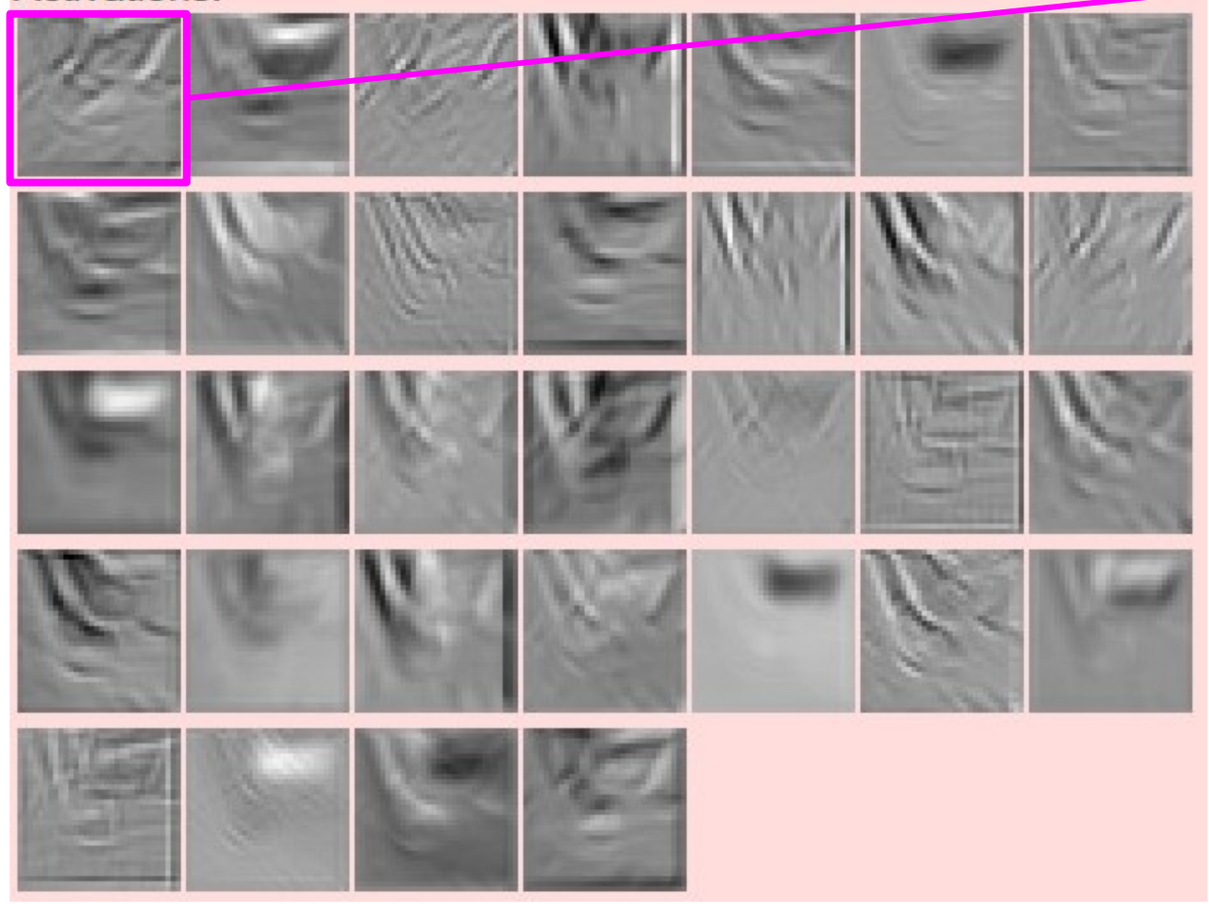
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

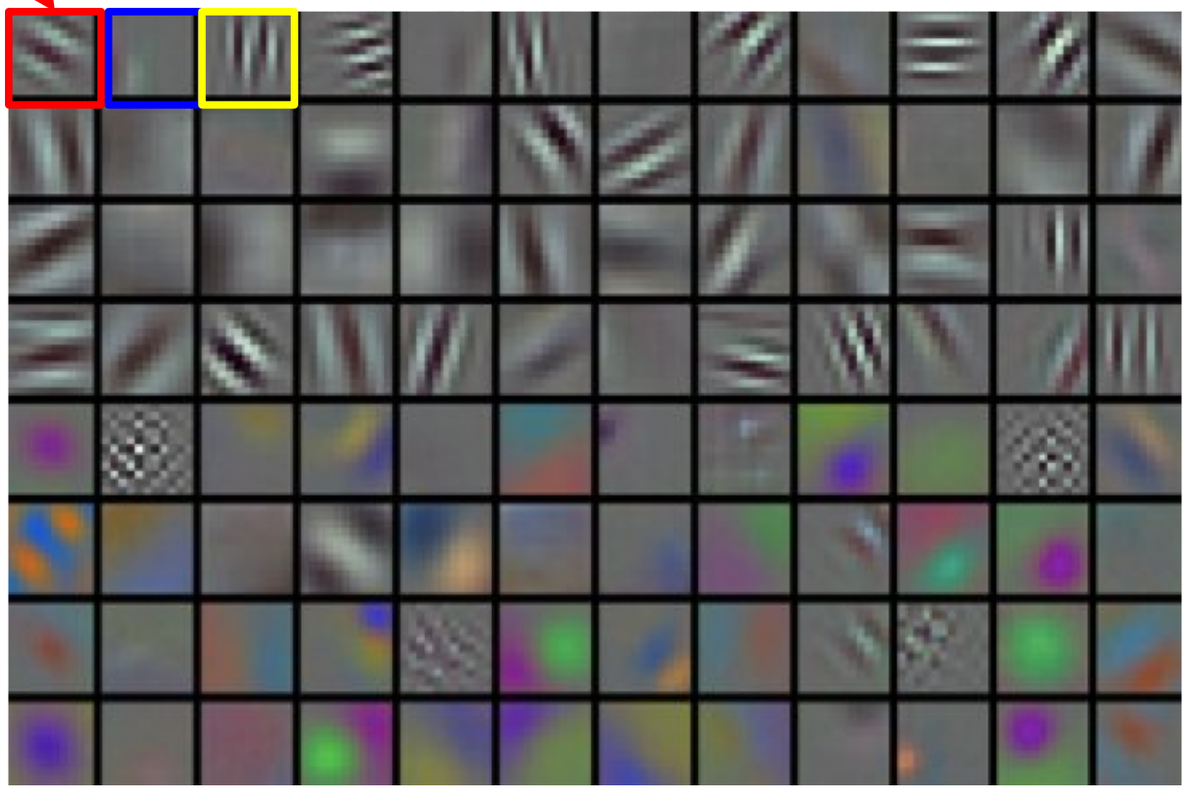
Activations:

convolving the first filter in the input gives
the first slice of depth in output volume

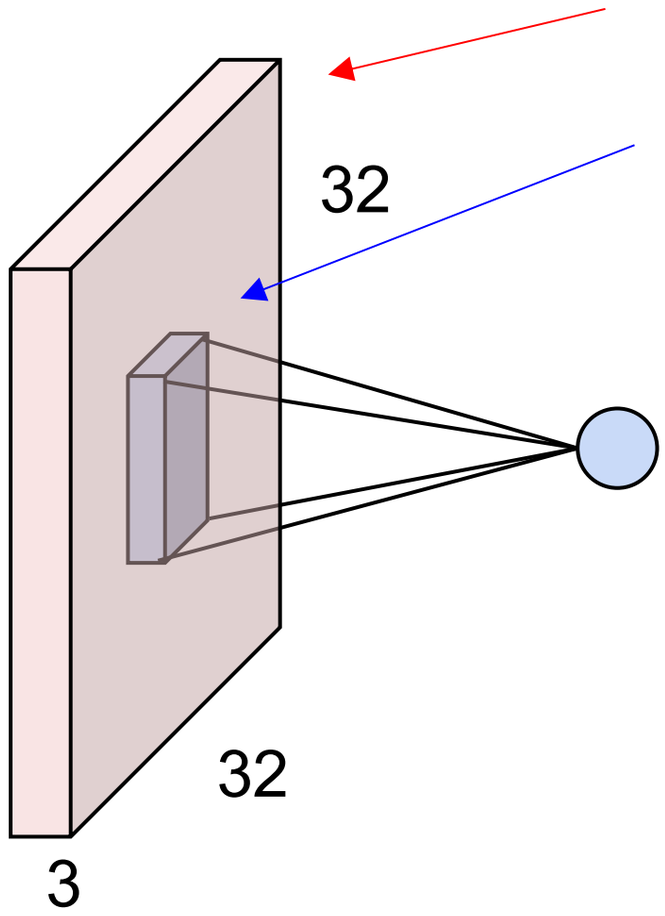


Activations:





A closer look at spatial dimensions:

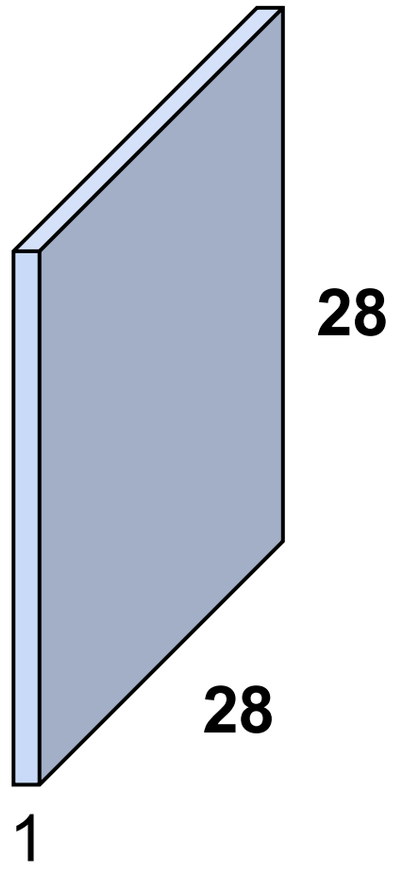


32x32x3 image

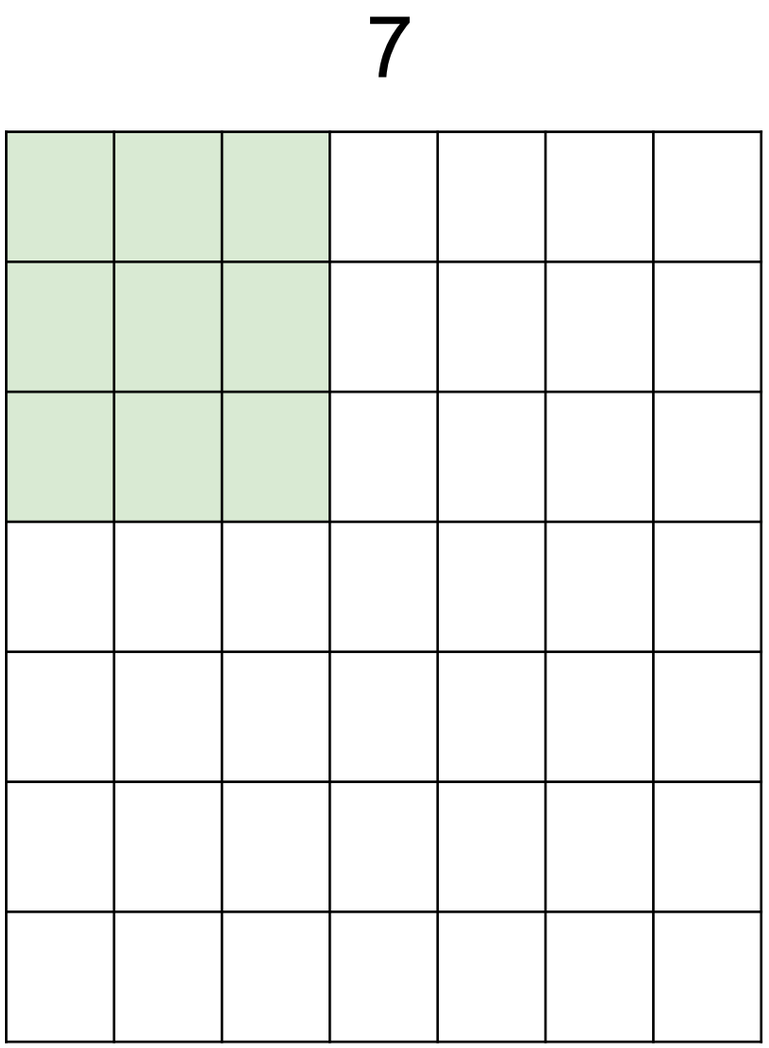
5x5x3 filter

convolve (slide) over all spatial locations

activation map

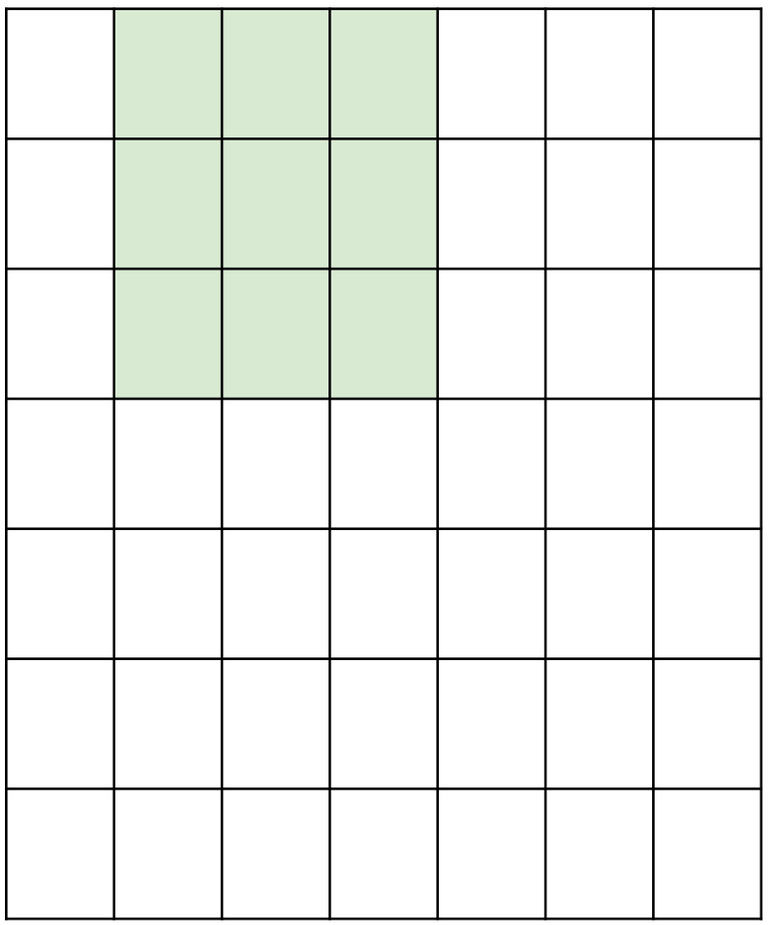


A closer look at spatial dimensions:



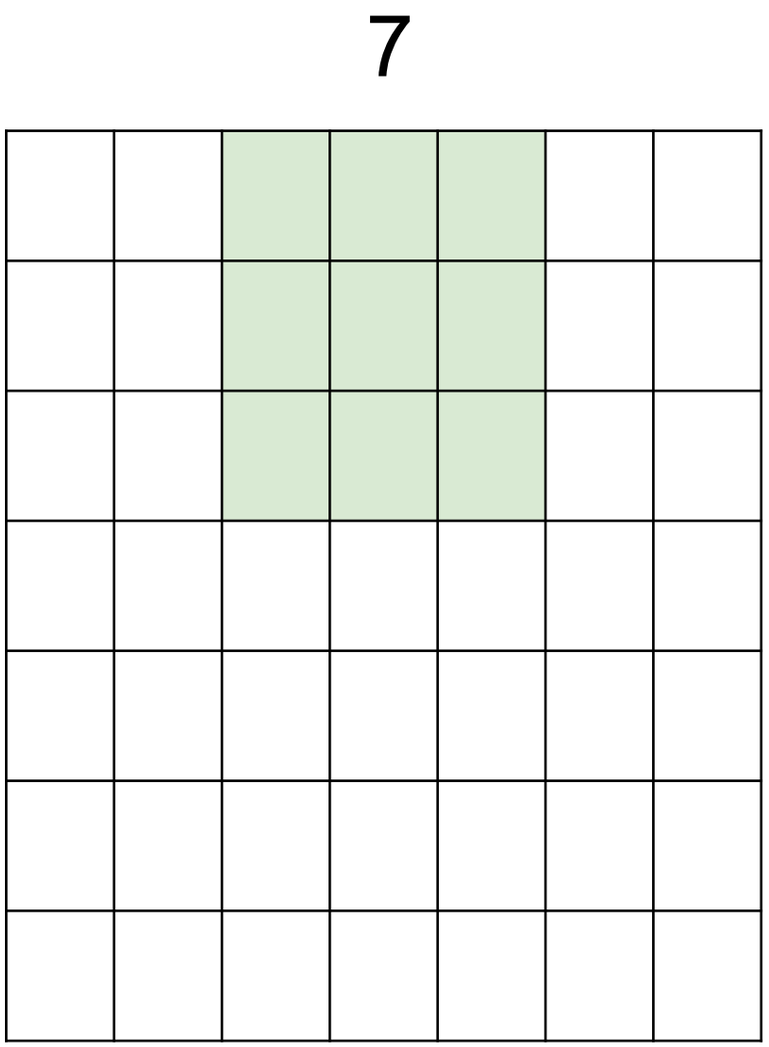
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



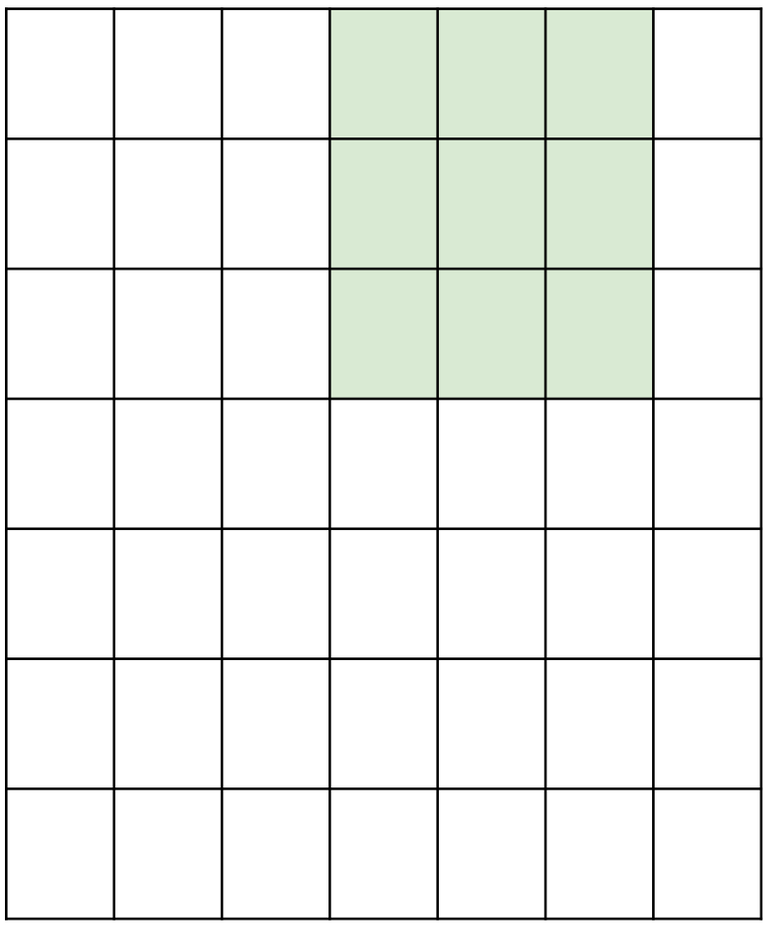
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



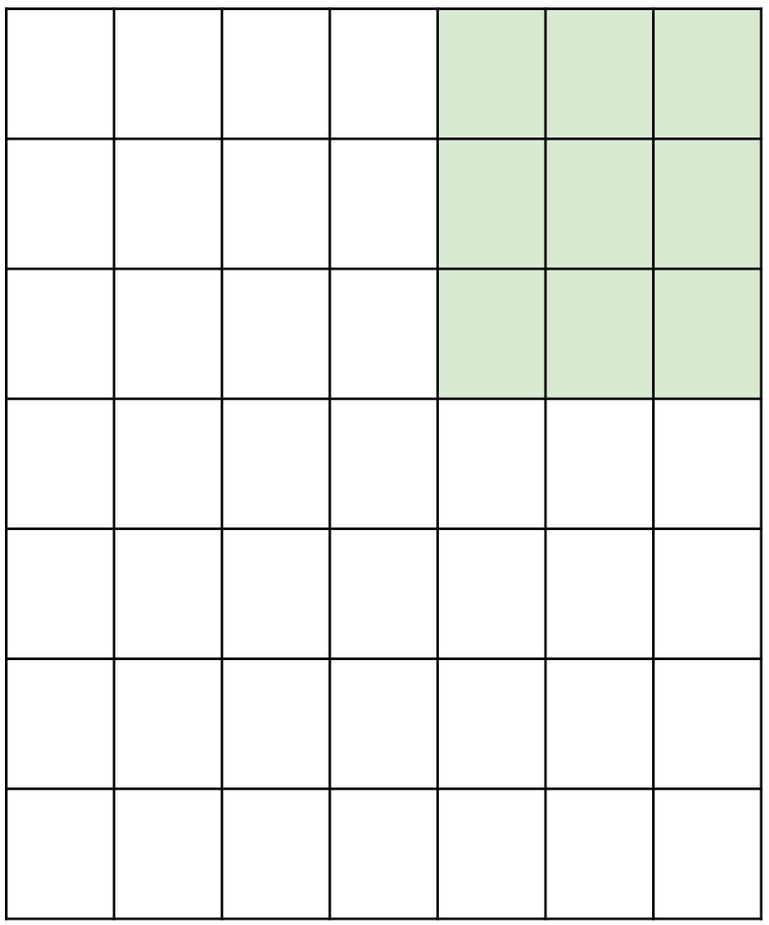
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

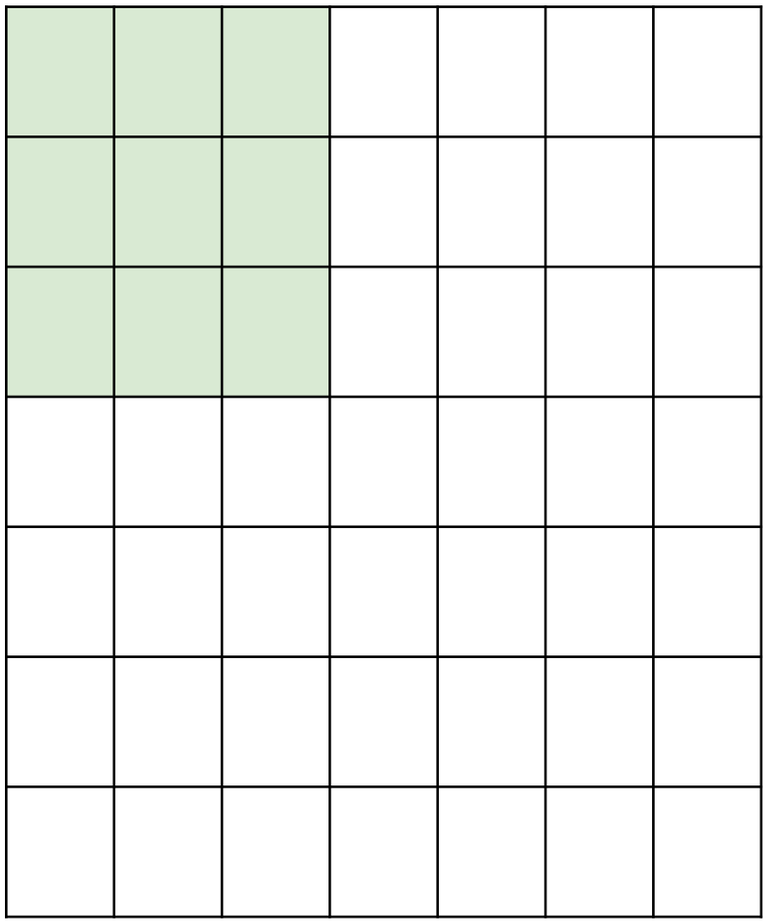


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:

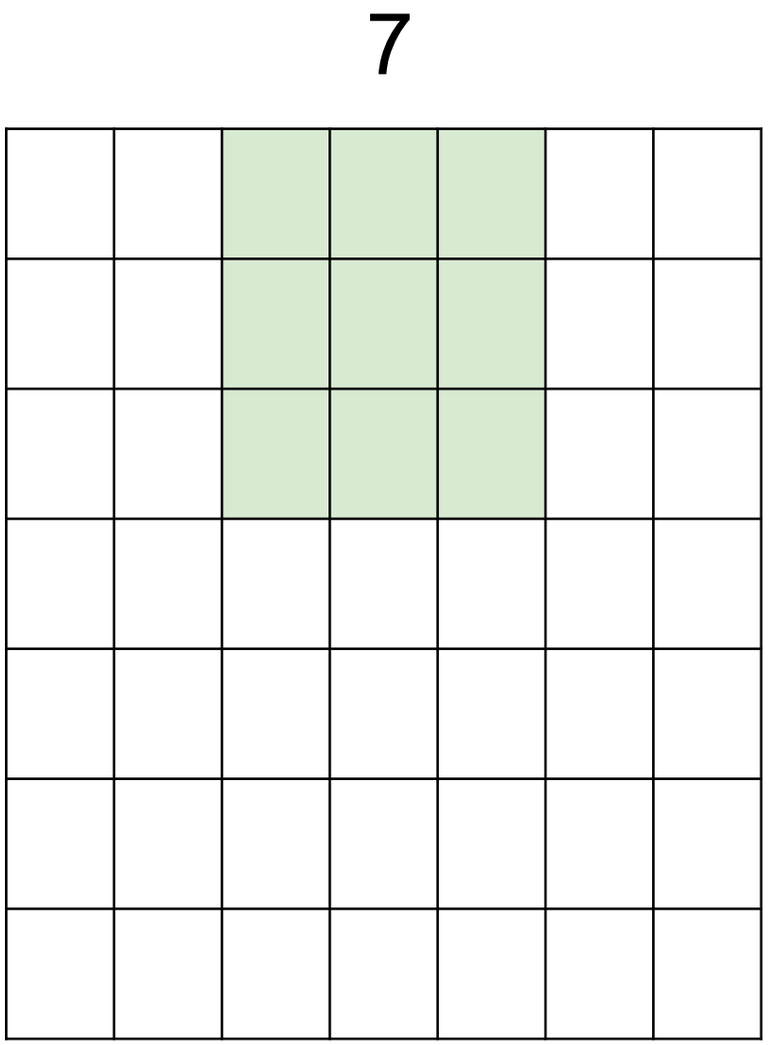
7



7

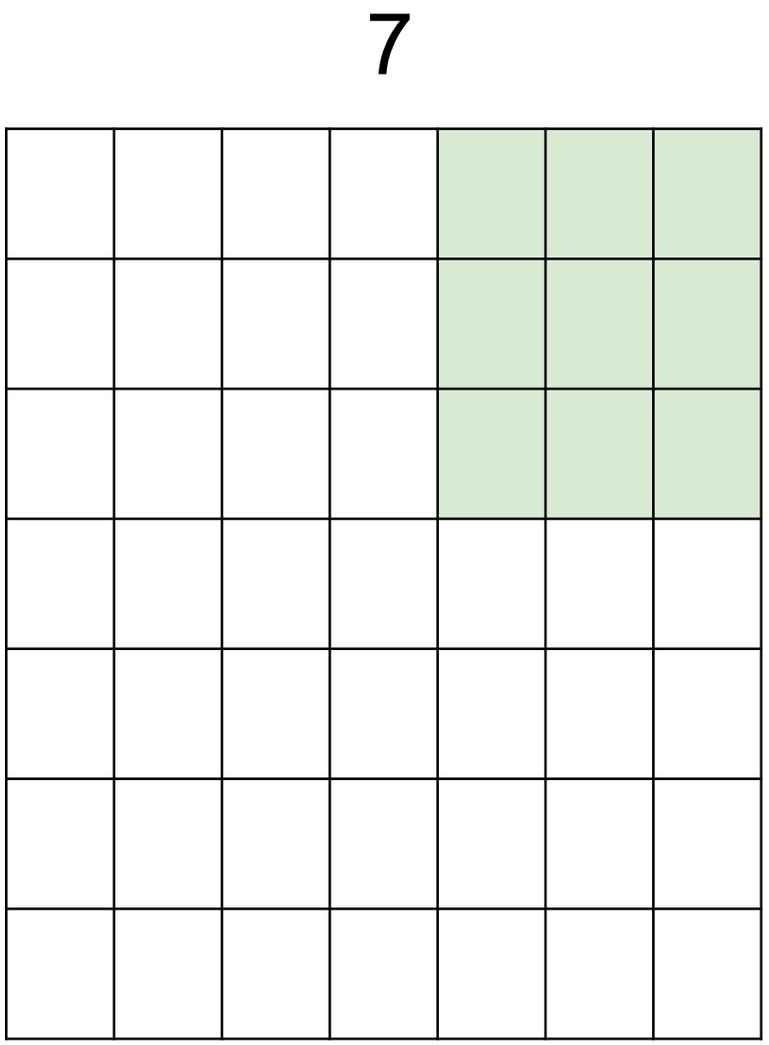
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



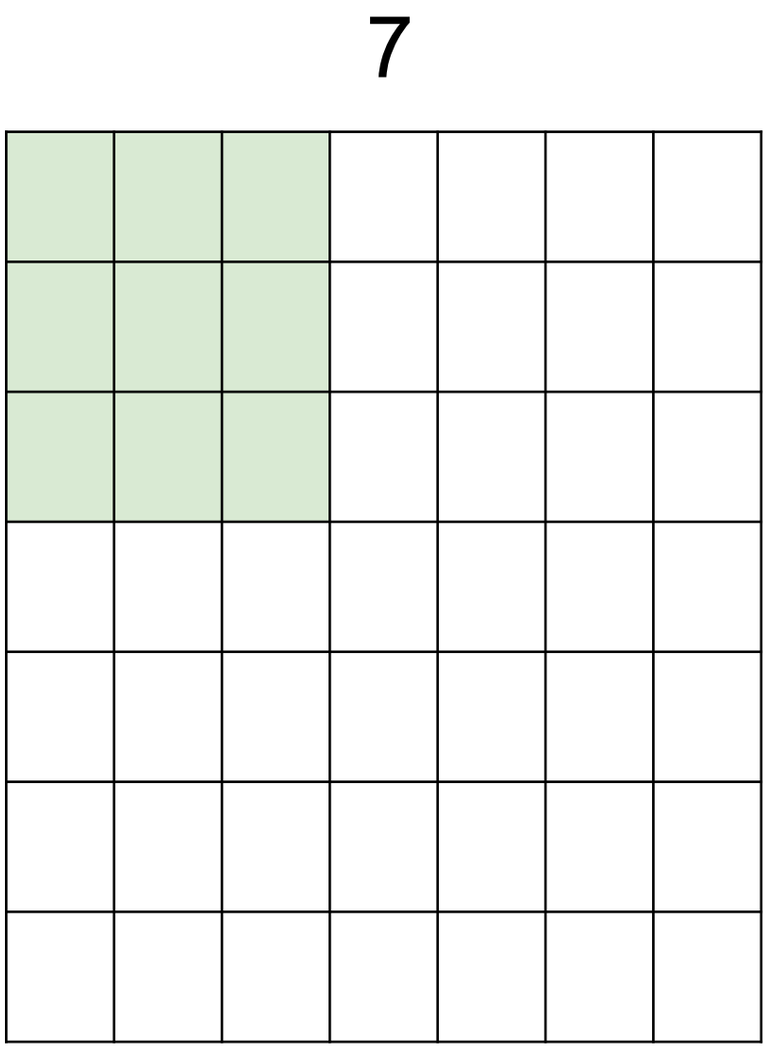
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



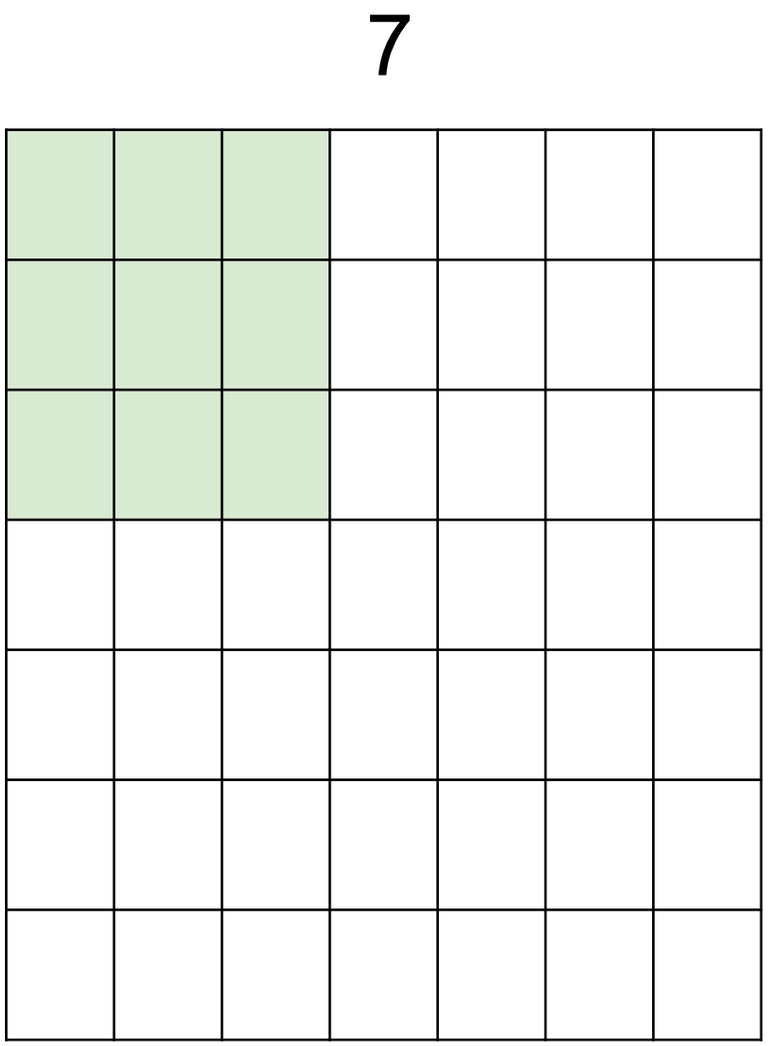
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



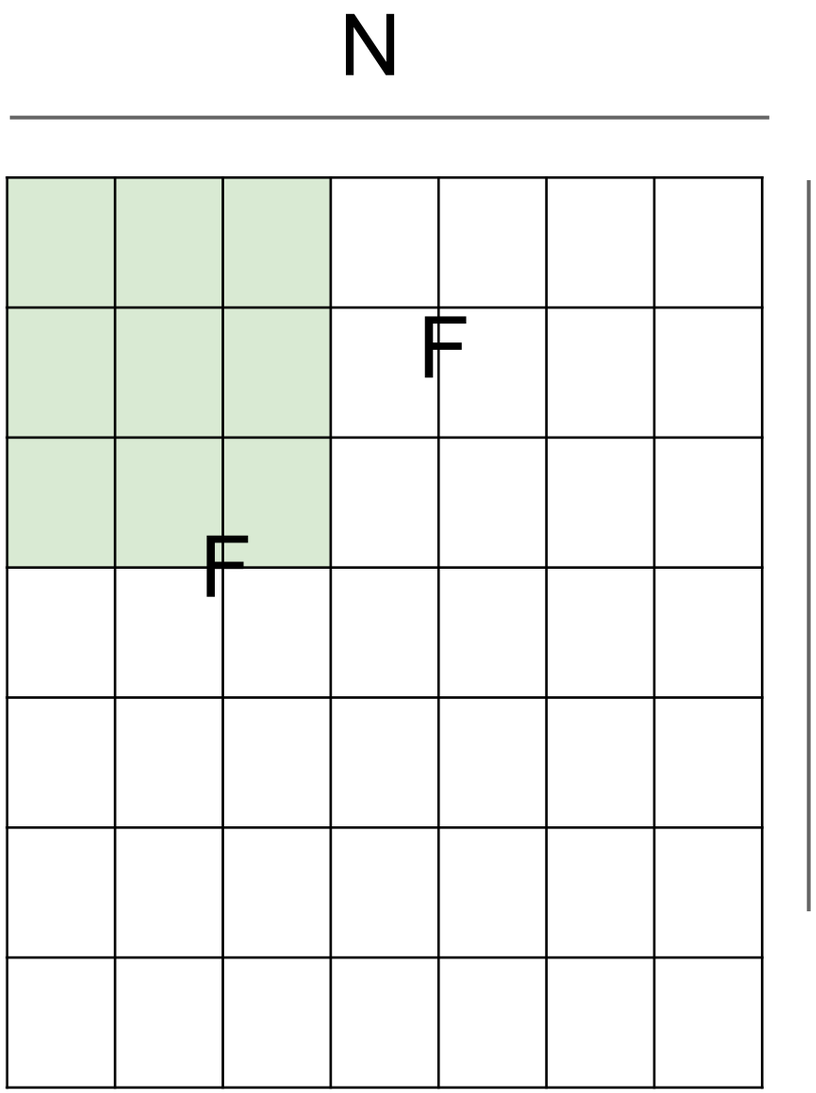
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
Wastes some input pixels.



Output size:

(N - F) / stride + 1

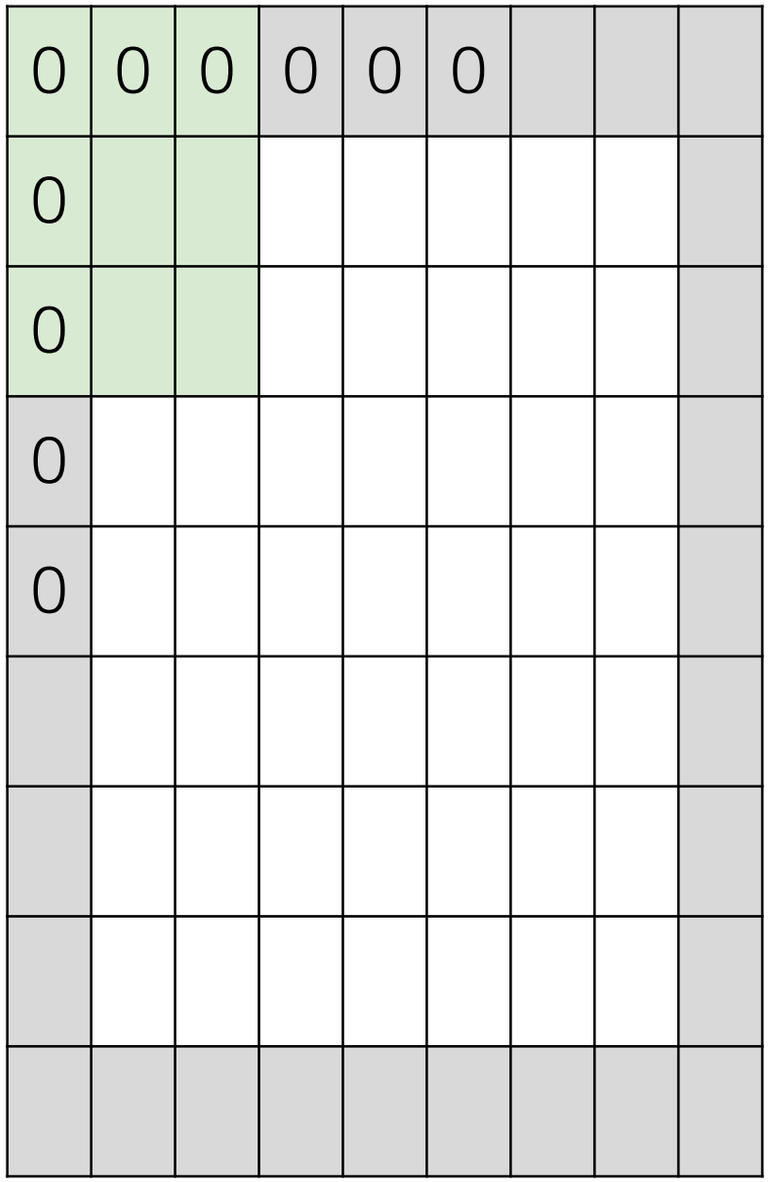
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border

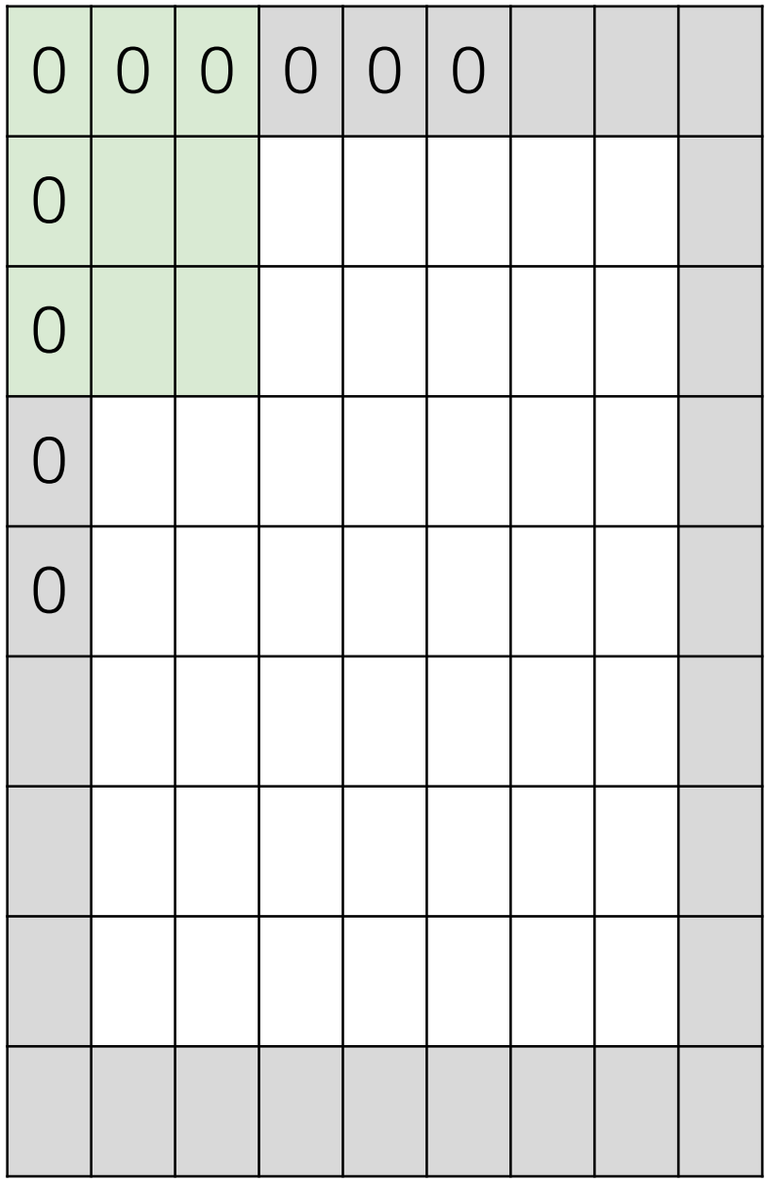


Aside: remove image mean first!

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

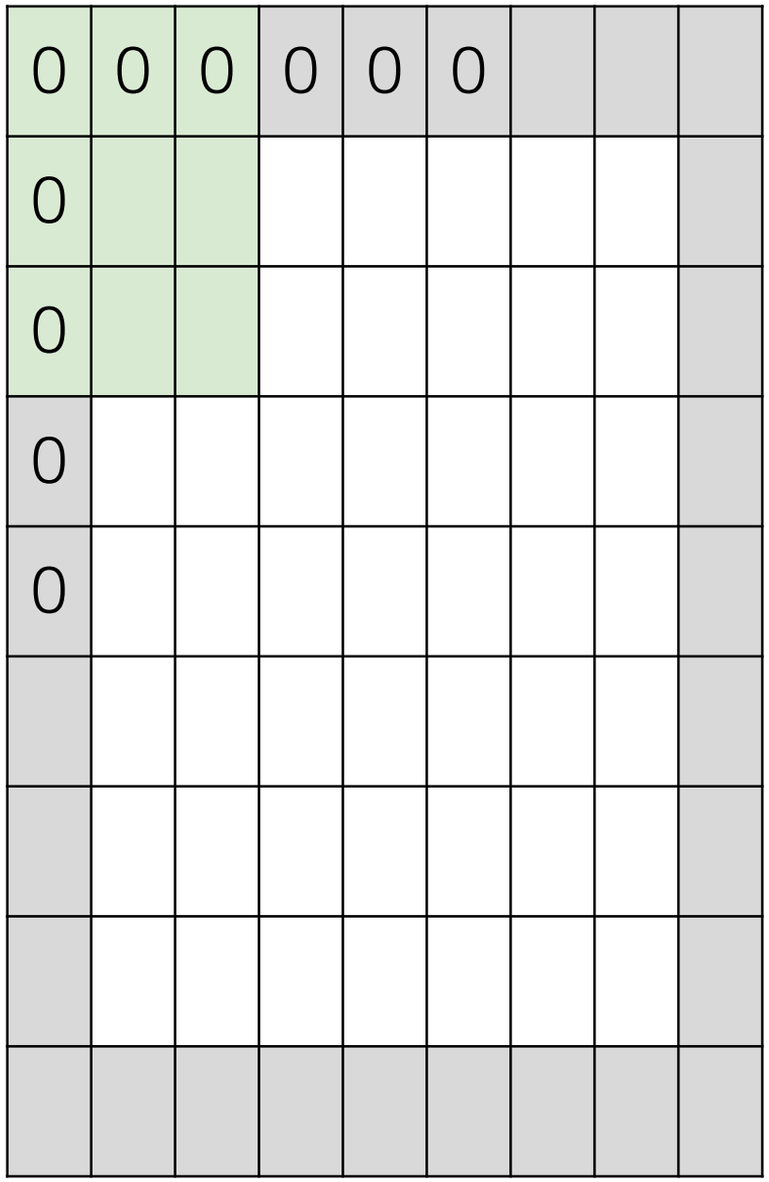
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

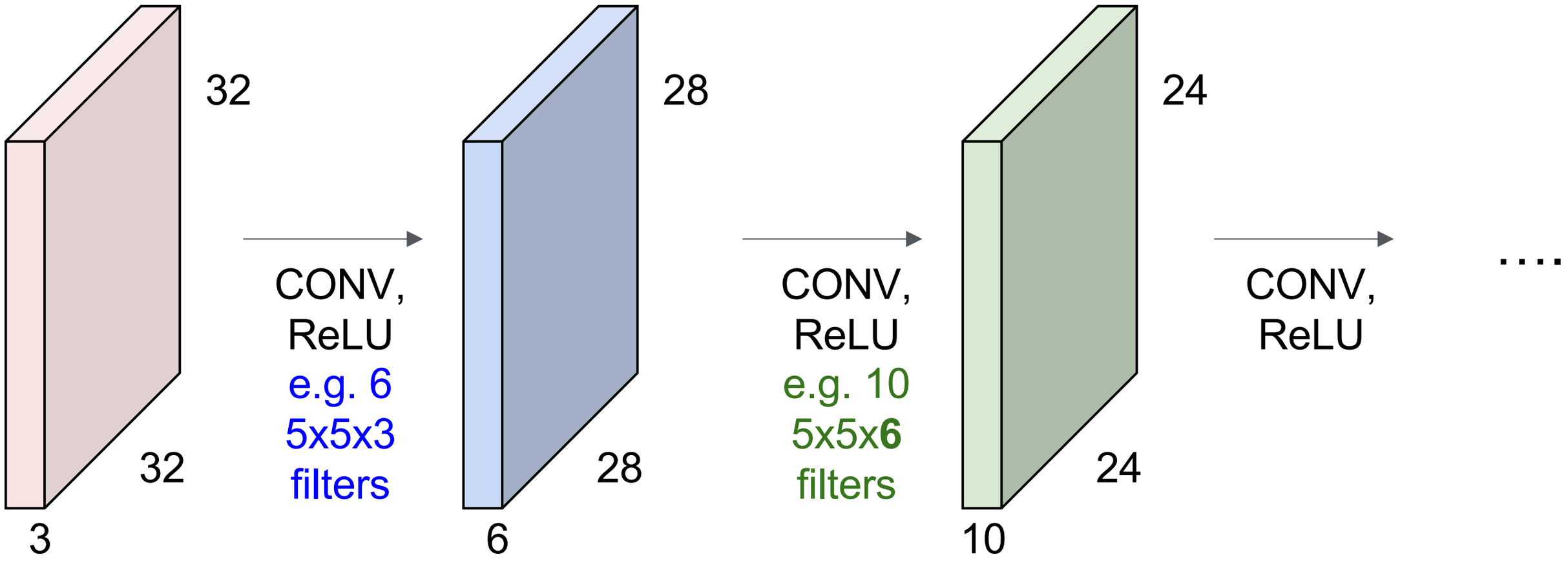
7x7 output!

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

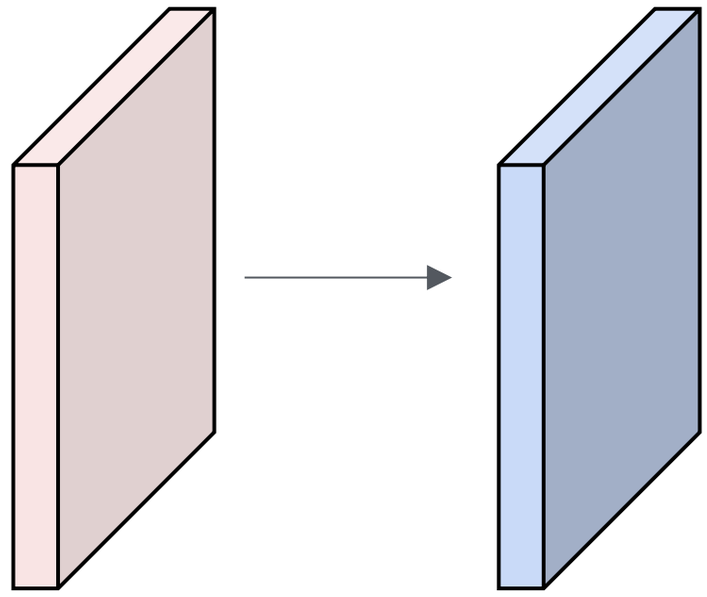
Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



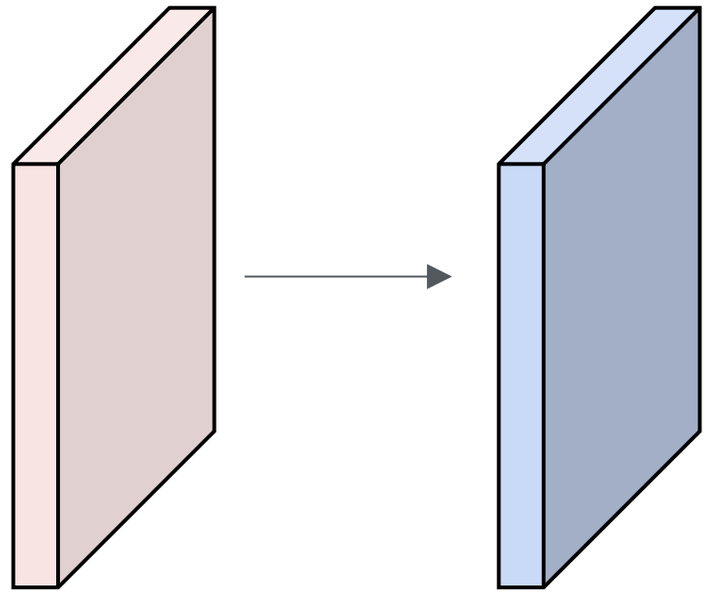
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size: ?

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

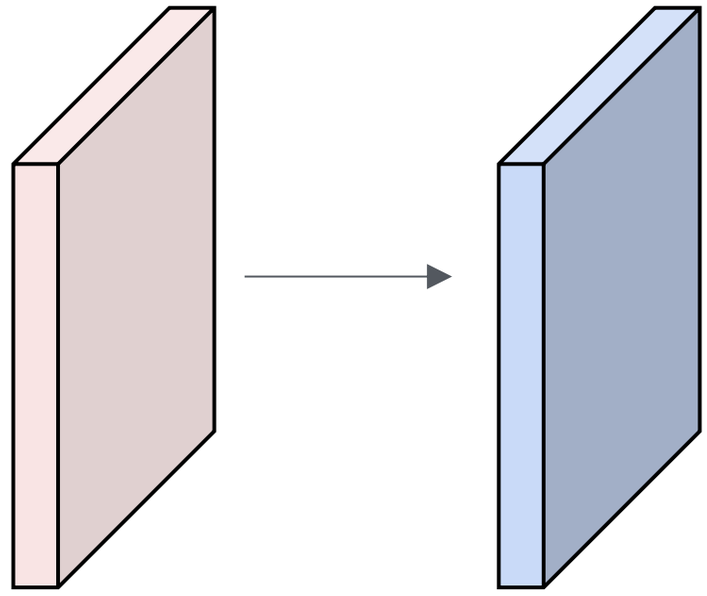
Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

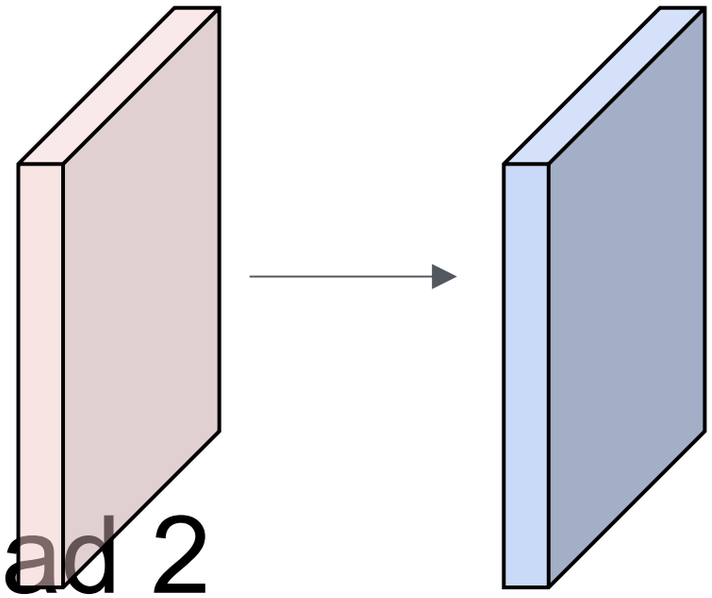
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary. To summarize, the Conv Layer:

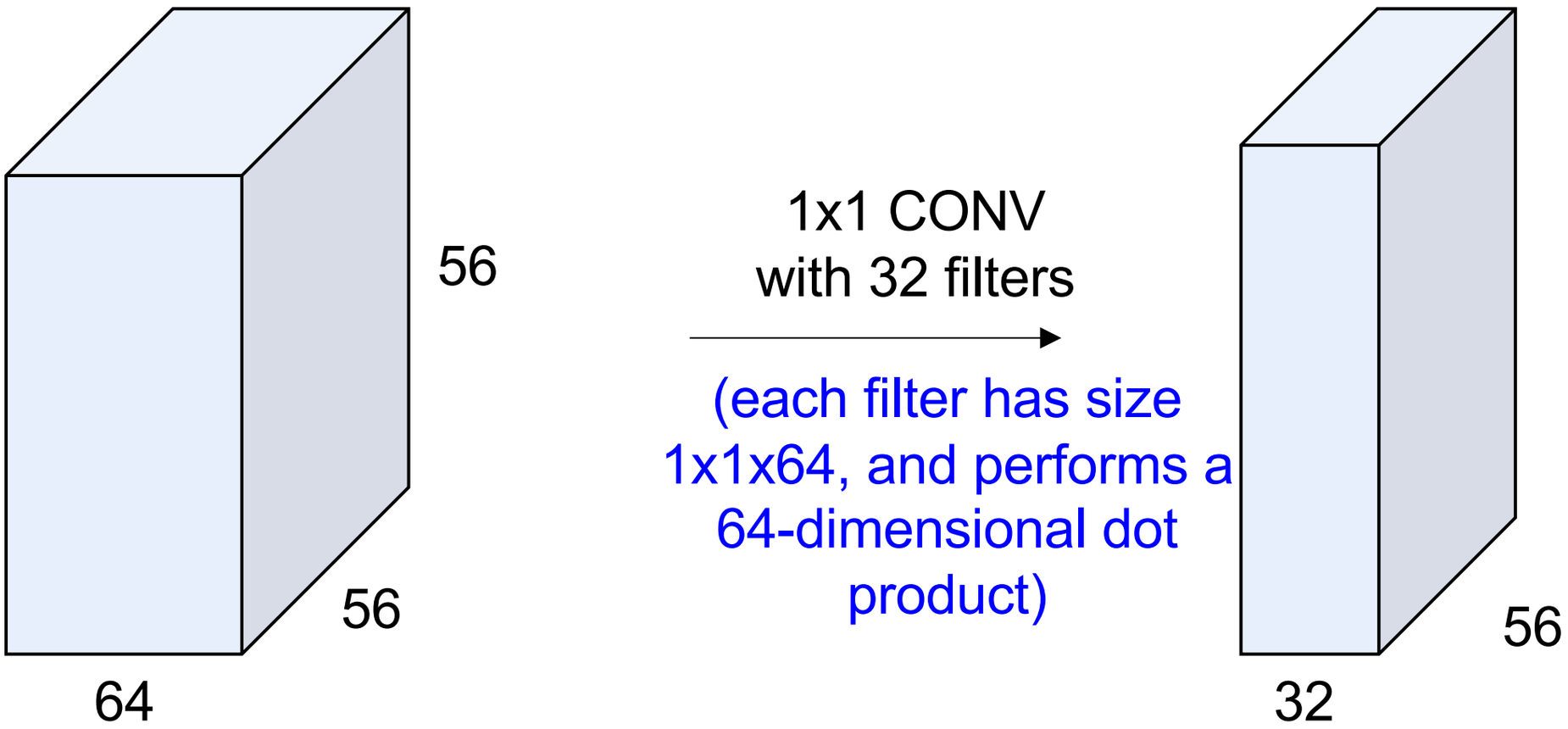
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

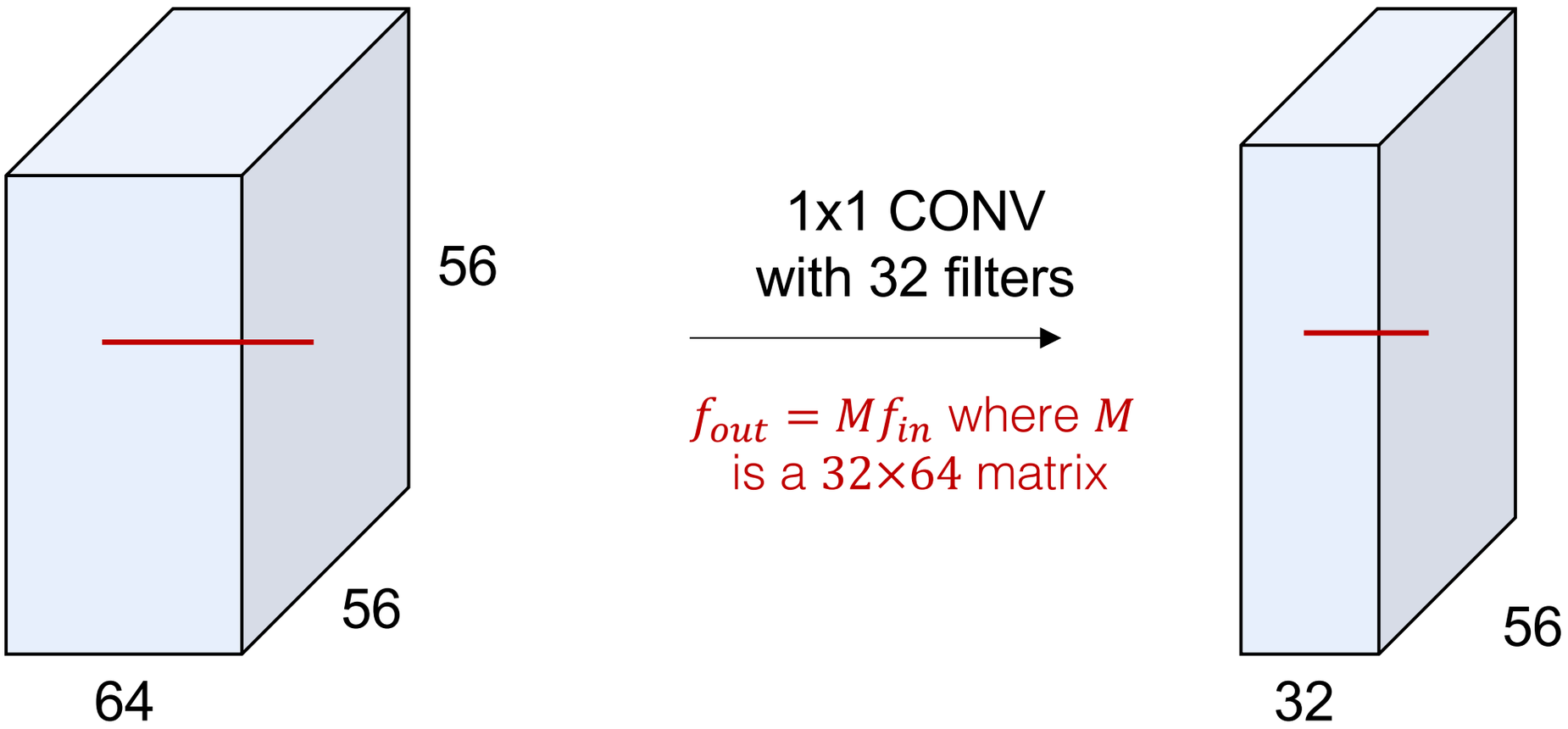
K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

(btw, 1x1 convolution layers make perfect sense)



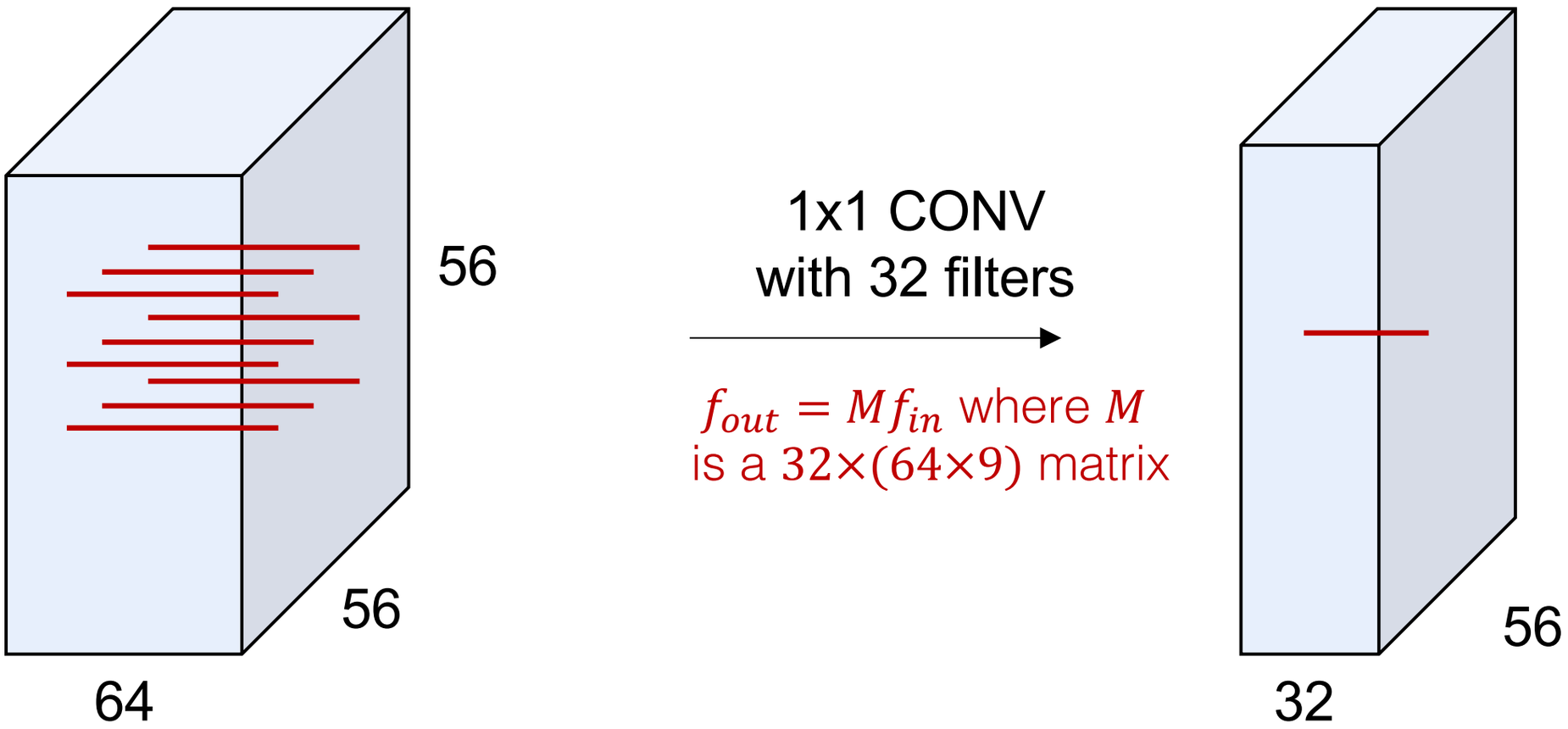
(btw, 1x1 convolution layers make perfect sense)



Input “fiber” f_{in}

Output “fiber” f_{out}

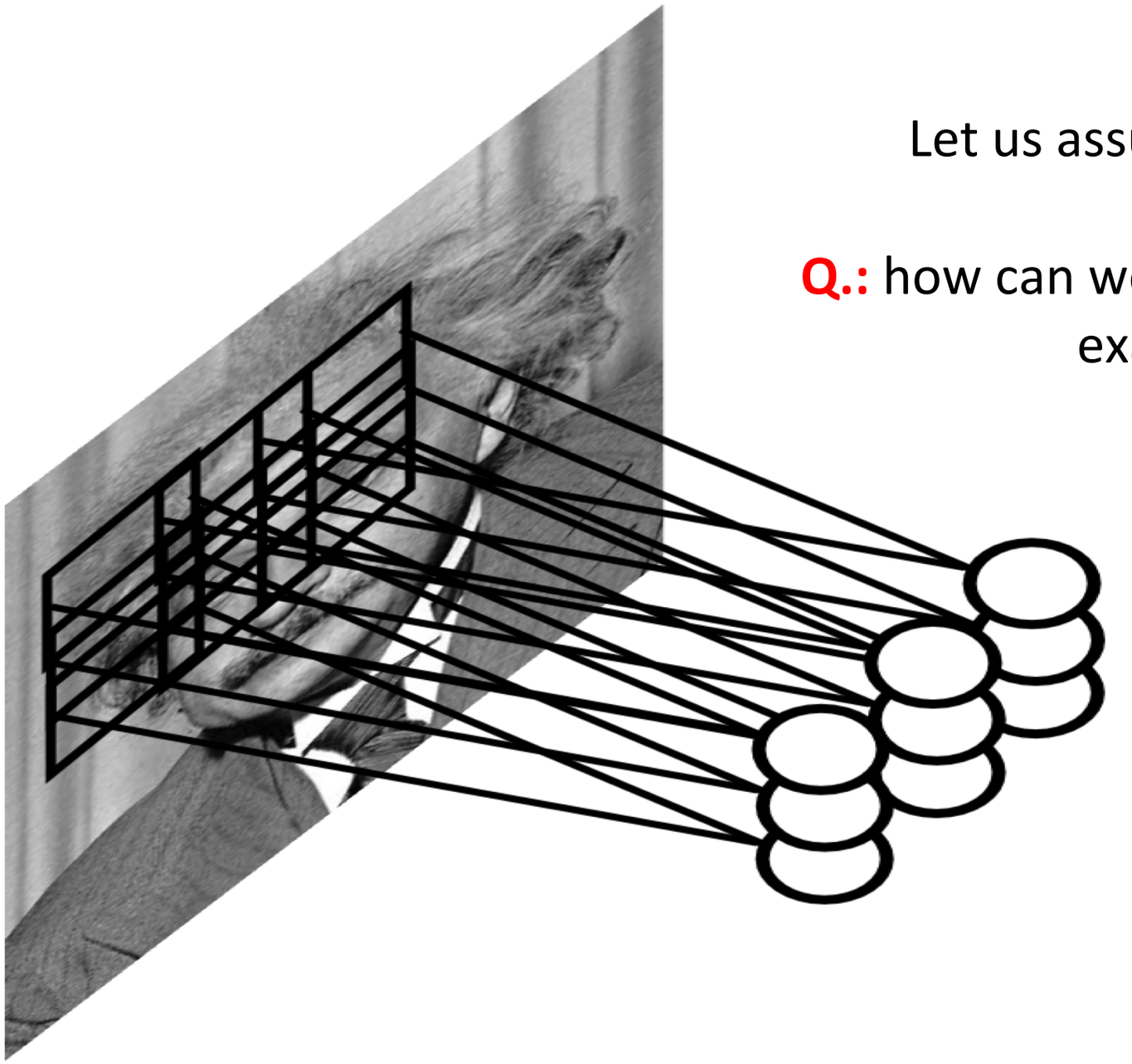
Aside: convolution via matrix multiply: im2col



Input f_{in} = concatenation of 3x3 array of fibers

Output "fiber" f_{out}

Pooling Layer



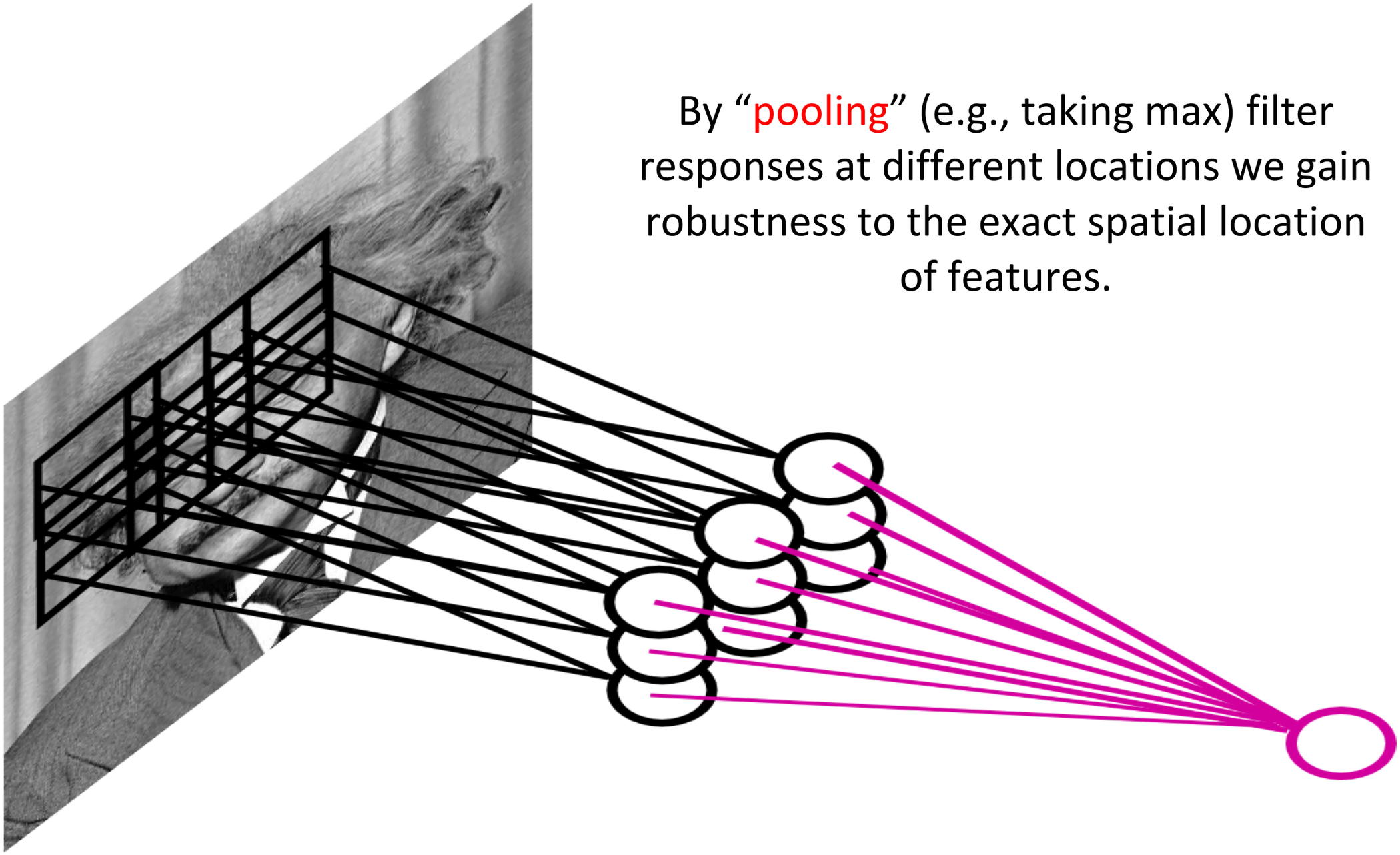
Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Ranzato

Pooling Layer

By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.



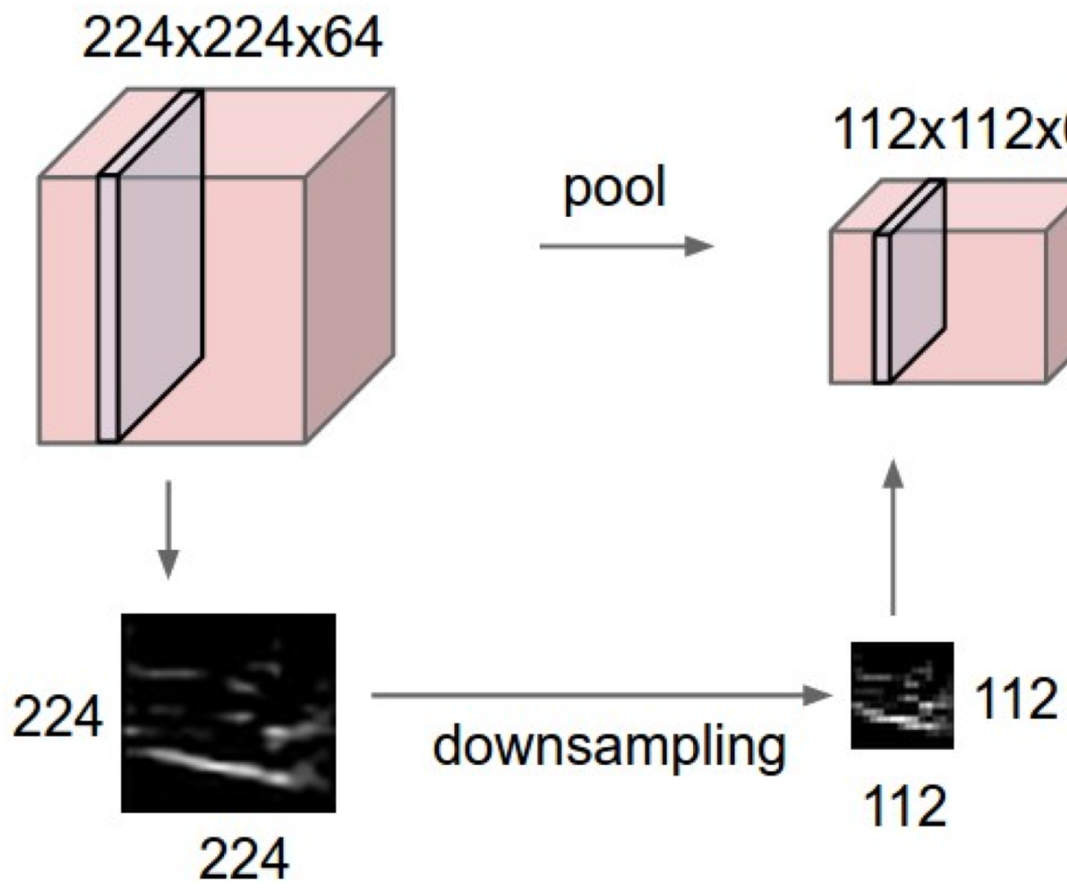
Ranzato

Pooling layer reduce width and height by half

For CONV layers: ((m * n * d)+1)* k) paras (1 for bias)
For pooling layer: 0 para
For FC: c * d + c * 1 (1 for bias) (c: current layer , p: previous layer)

Pooling layer

- makes the representations smaller and more manageable
  - operates over each activation map independently:



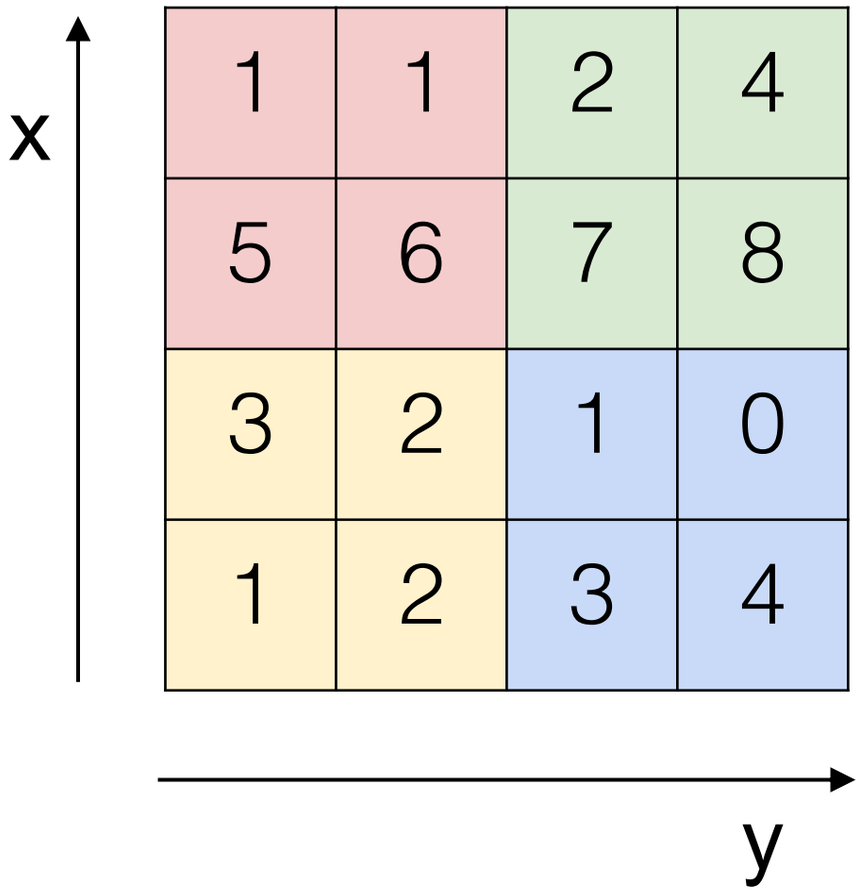
Max pooling: max value

Average pooling: average value

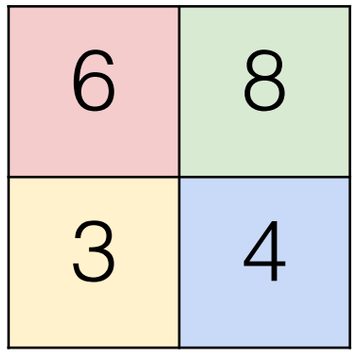
Global pulling: produce 1 x 1 x d feature map

MAX POOLING

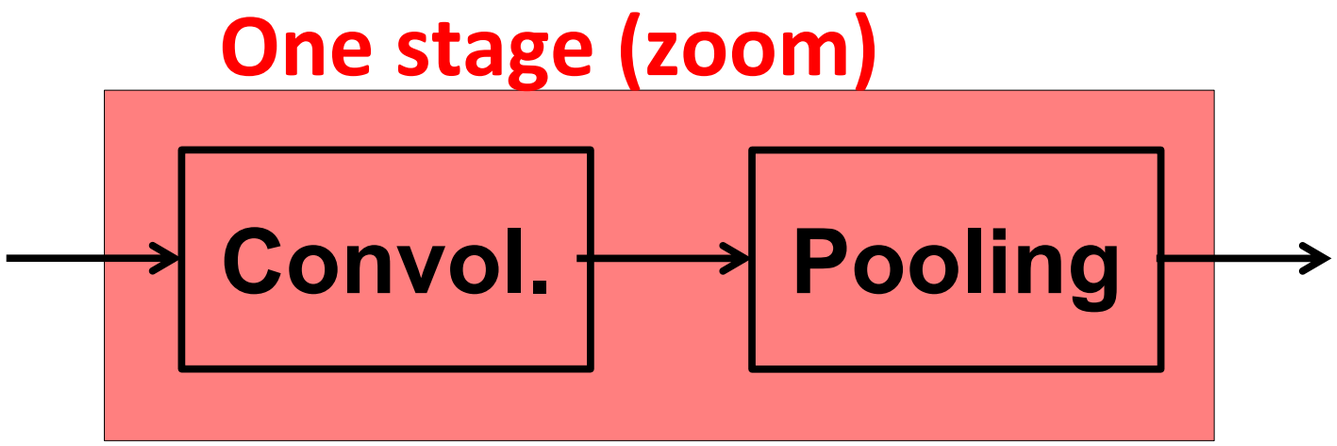
Single depth slice

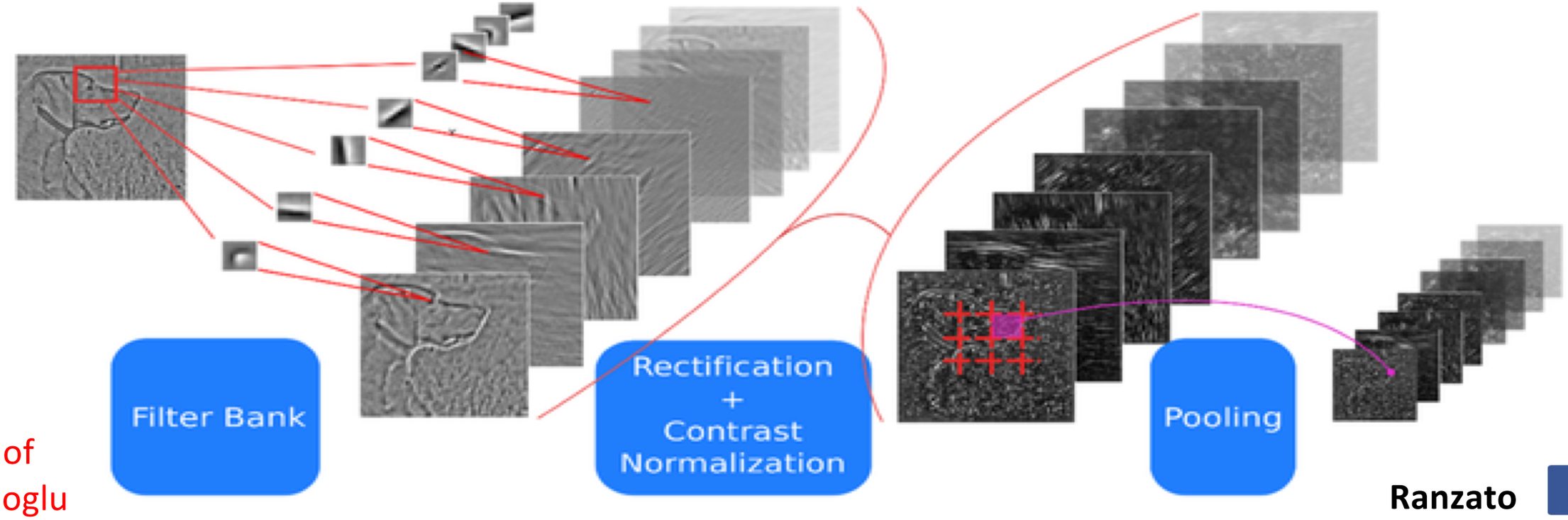


max pool with 2x2 filters and stride 2



ConvNets: Typical Stage

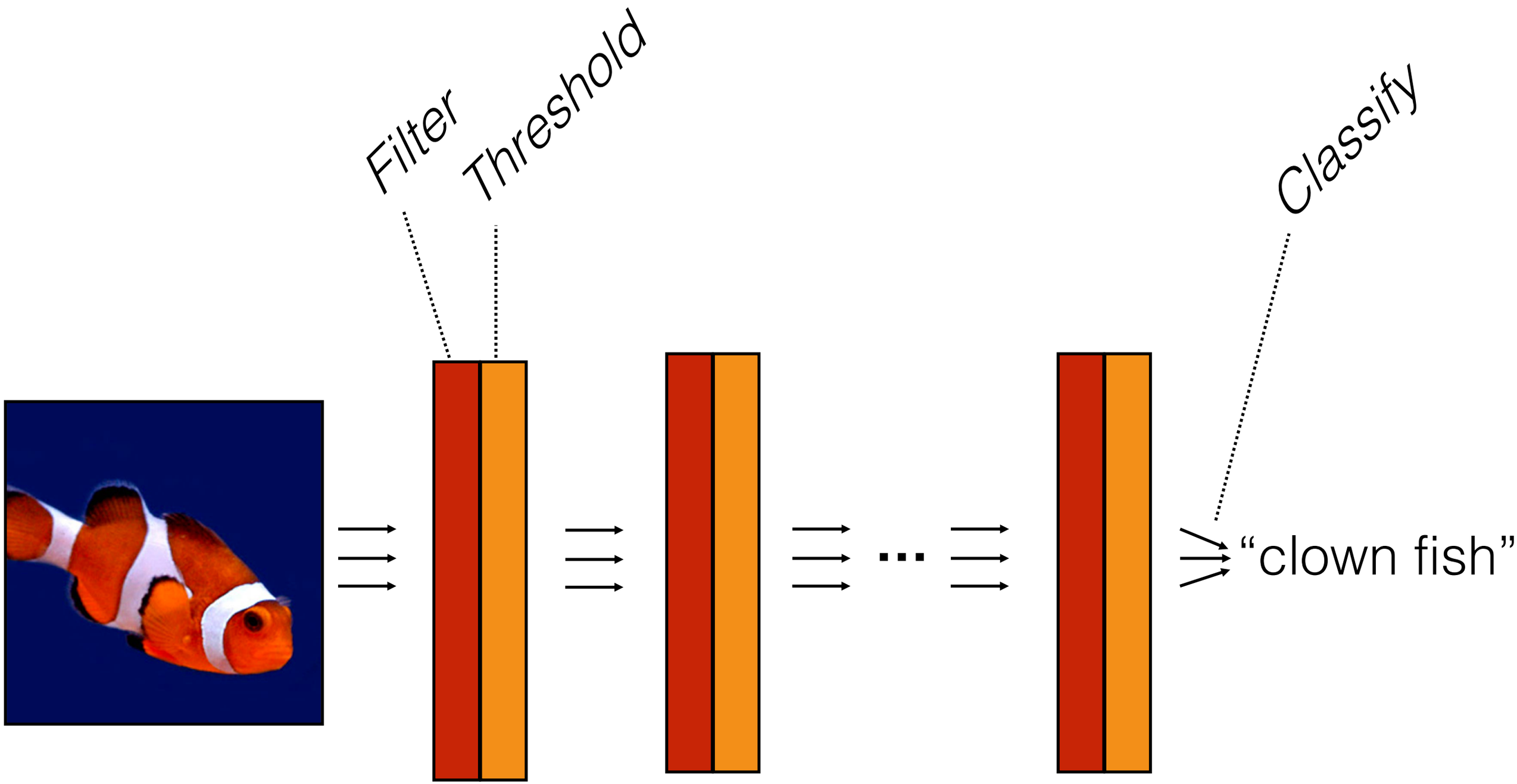




courtesy of
K. Kavukcuoglu

Ranzato

Computation in a neural net

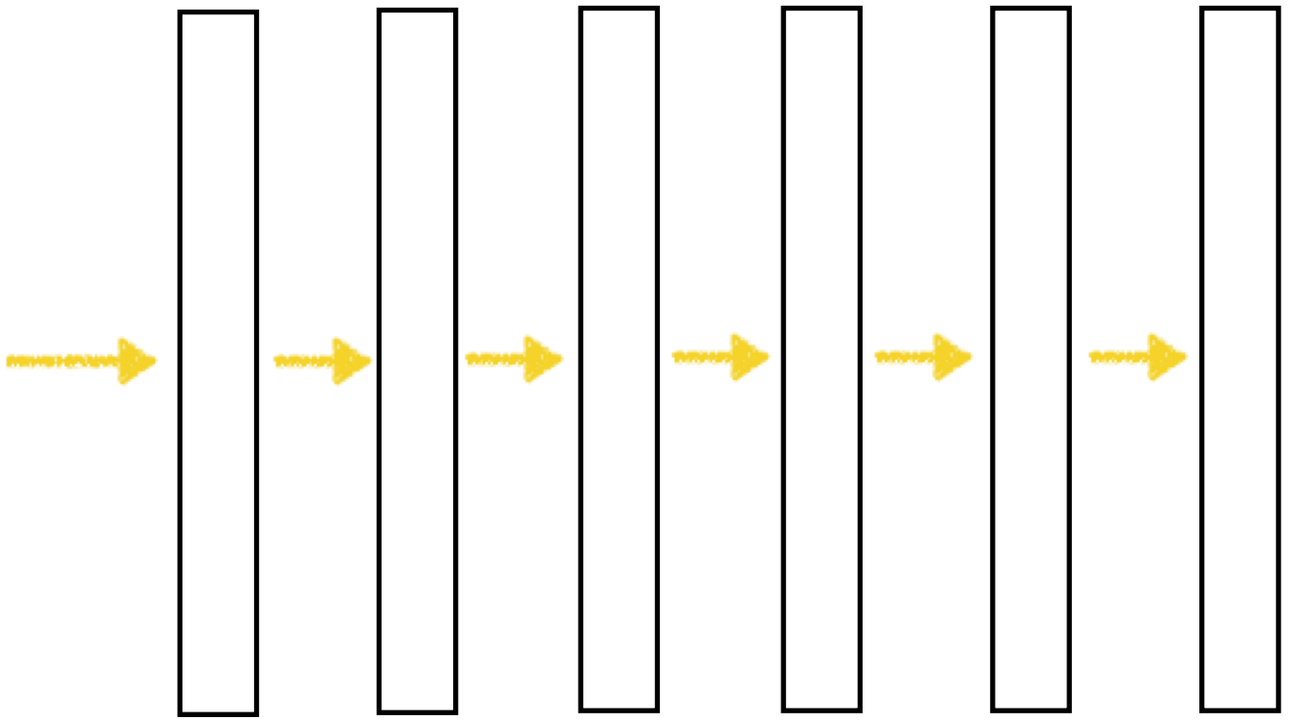


$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Computation in a neural net

Learning with deep nets

Learned



→ “clown fish”

Learning with deep nets

→ “clown fish”

→ “grizzly bear”

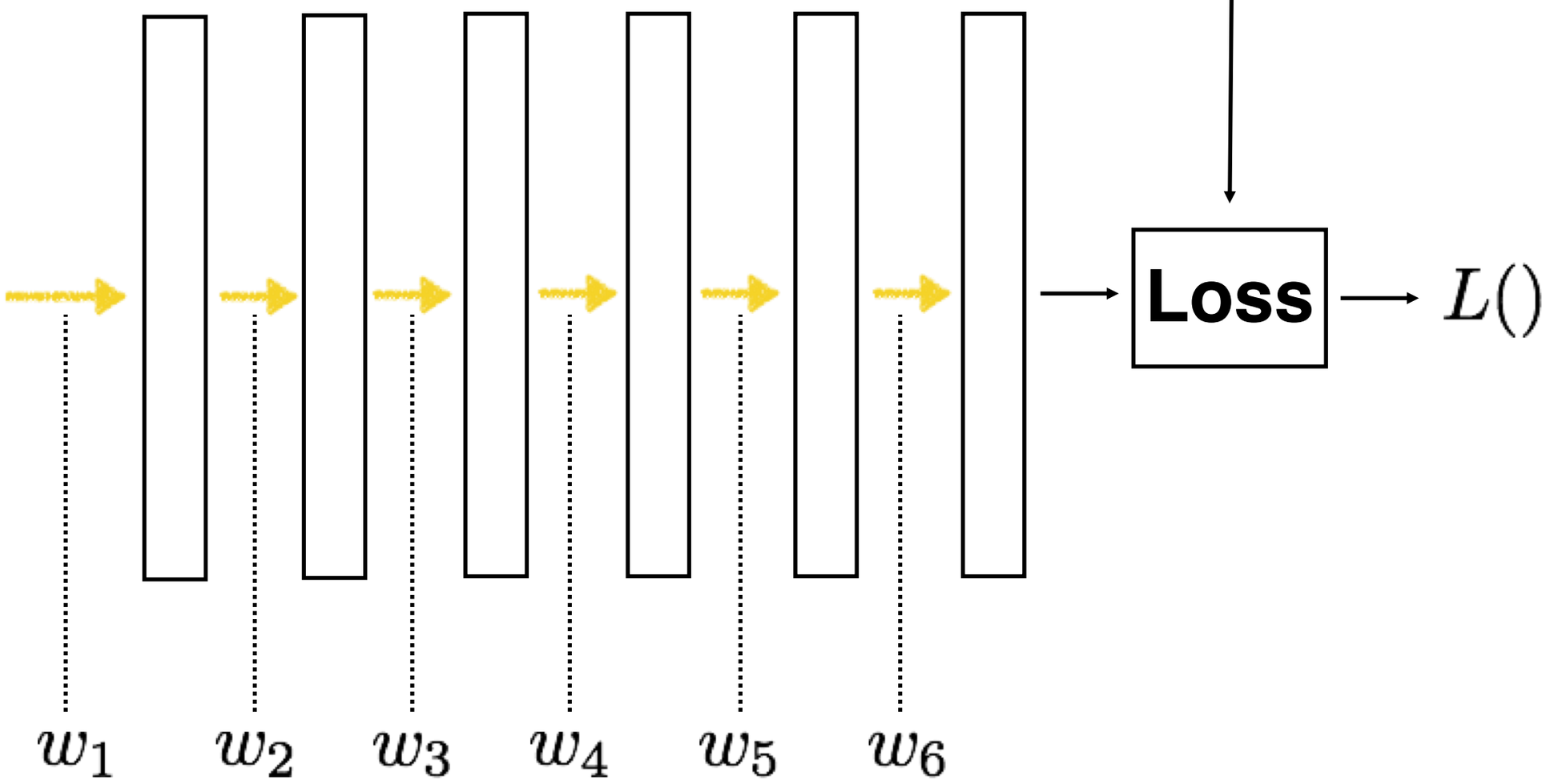
→ “chameleon”

Train network to associate the right label with each image

Learning with deep nets

Learned

"clown fish"

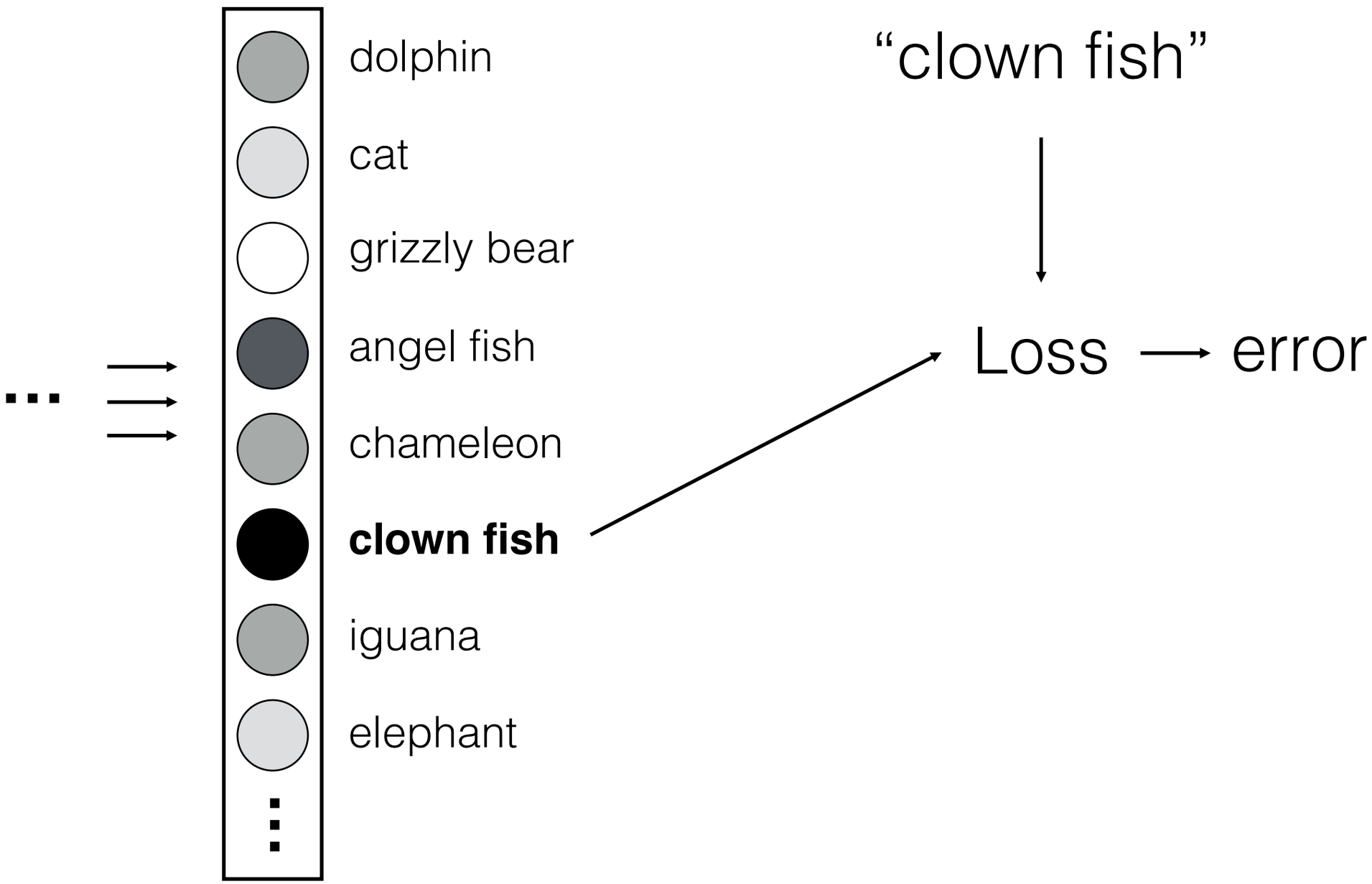


$$\underset{\mathbf{w}}{\text{argmin}} \quad L(w_1, \ldots, w_6)$$

Loss function

Network output

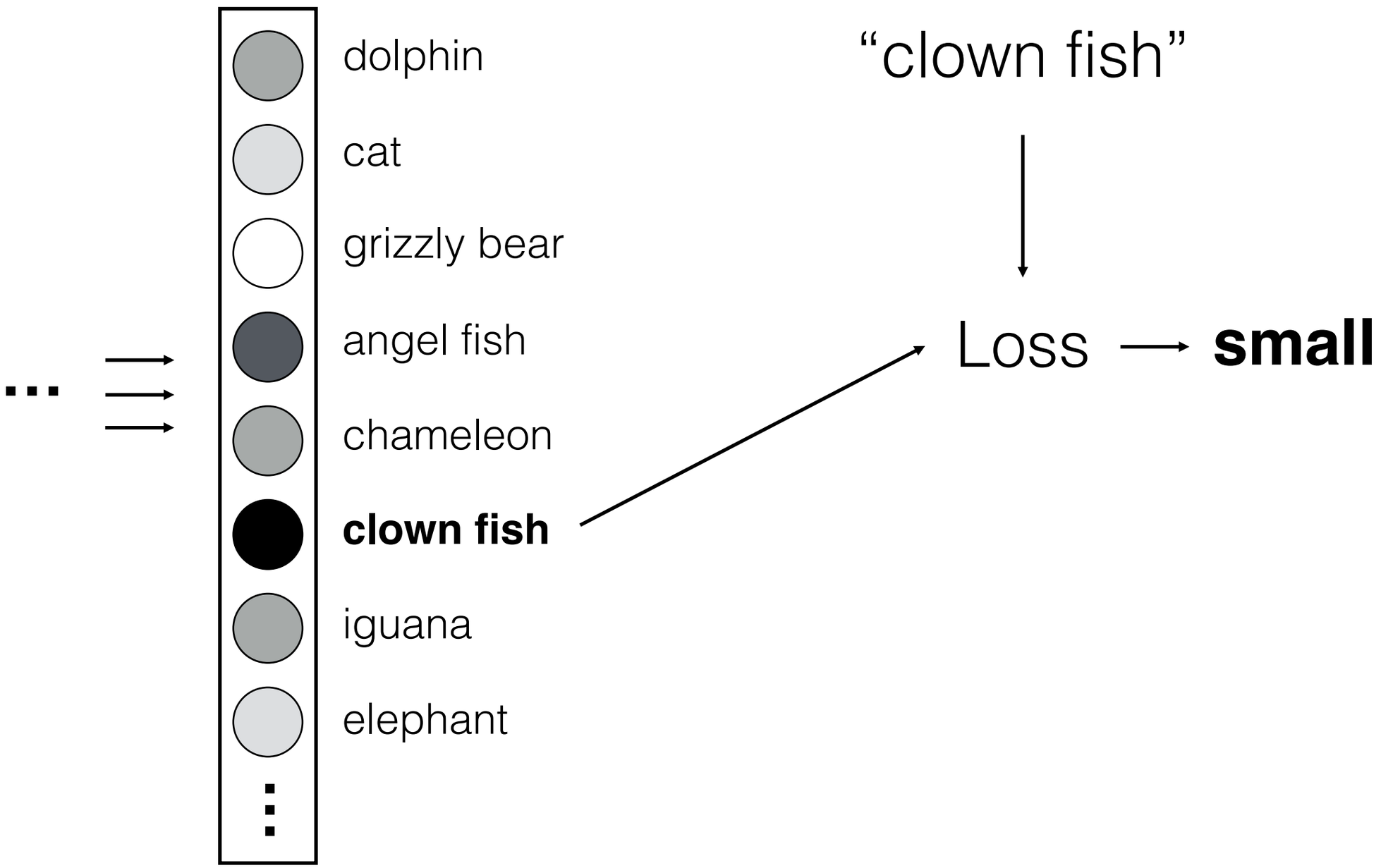
Ground truth label



Loss function

Network output

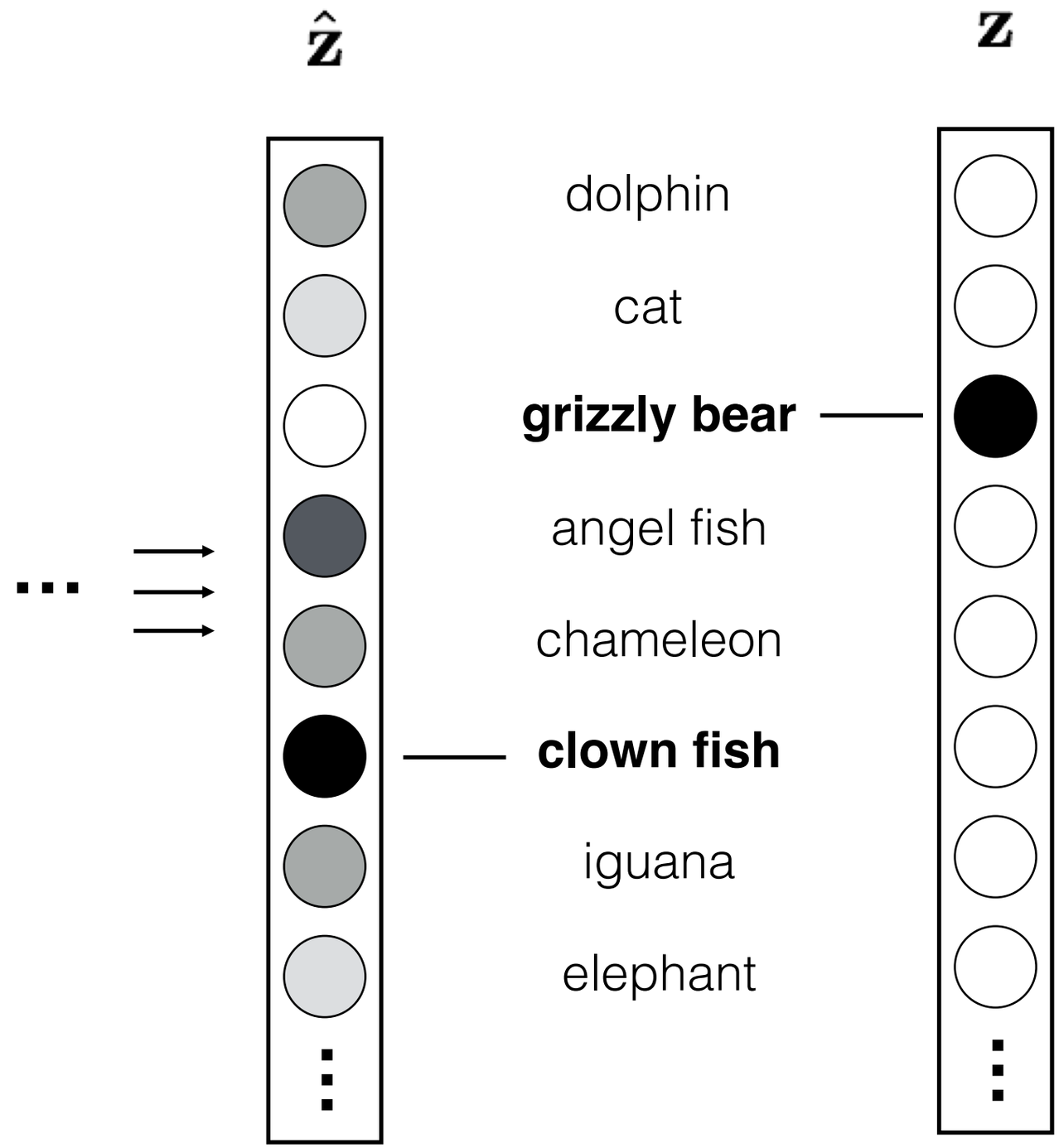
Ground truth label



Loss function

Loss function for classification

Network output Ground truth label



Probability of the observed data under the model

$$H(\hat{\mathbf{z}}, \mathbf{z}) = -\sum_c \hat{\mathbf{z}}_c \log \mathbf{z}_c$$

Results in learning a probability model $p(c|\mathbf{x})$*!*

Learning with deep nets

Learned

$\mathbf{z}_1$
"clown fish"

$\mathbf{x}_1$

Loss

$\ell(\mathbf{z}_1, f(\mathbf{x}_1; \mathbf{w}))$

w_1 w_2 w_3 w_4 w_5 w_6

Learning with deep nets

Learned

$\mathbf{z}_2$

"grizzly bear"

$\mathbf{x}_2$

Loss

$\ell(\mathbf{z}_2, f(\mathbf{x}_2; \mathbf{w}))$

w_1 w_2 w_3 w_4 w_5 w_6

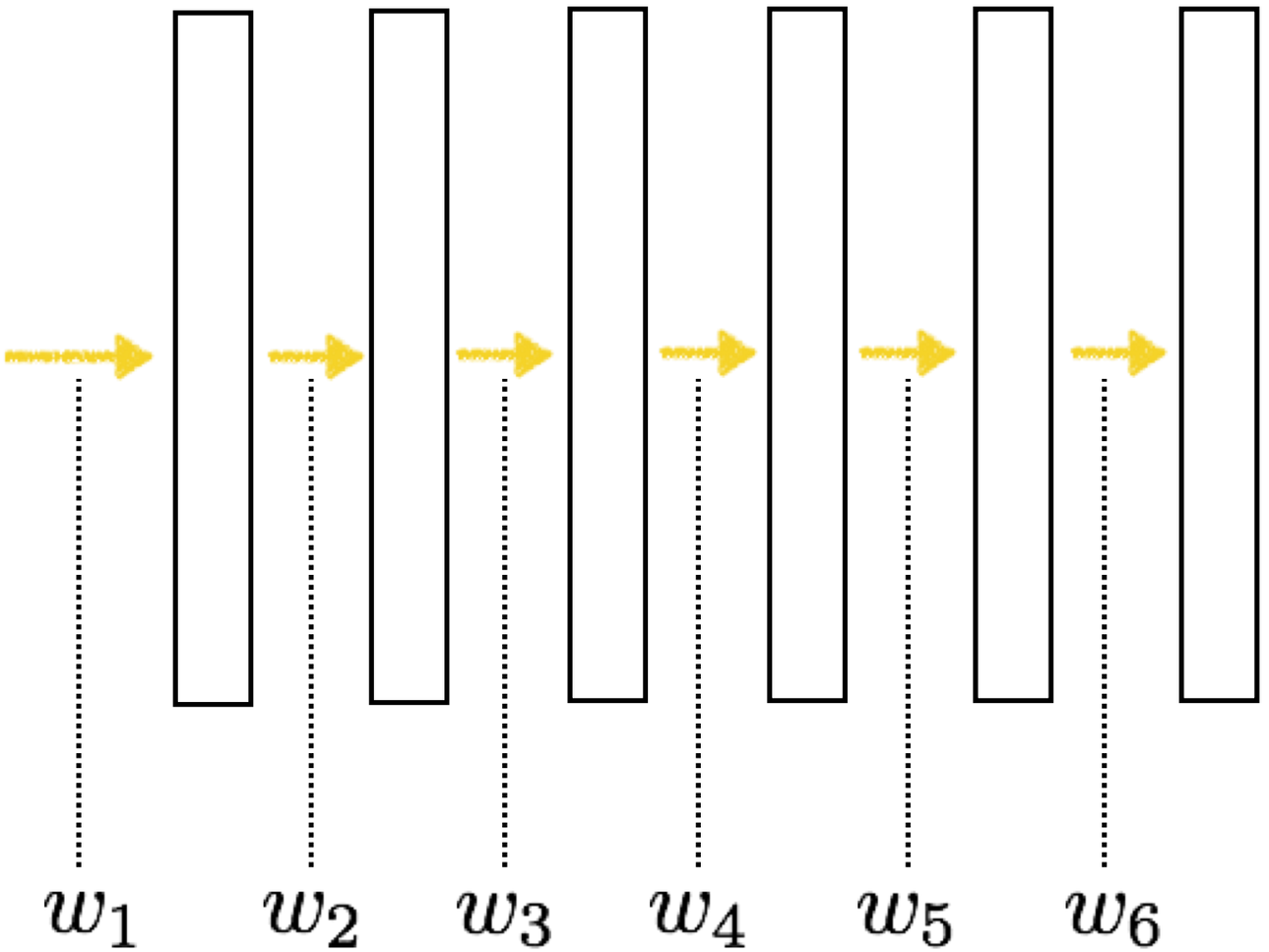
Learning with deep nets

Learned

$\mathbf{z}_i$

"chameleon"

$\mathbf{x}_i$



$\ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w}))$

$$\underset{\mathbf{w}}{\operatorname{argmin}} \sum_i \ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w}))$$

Gradient descent

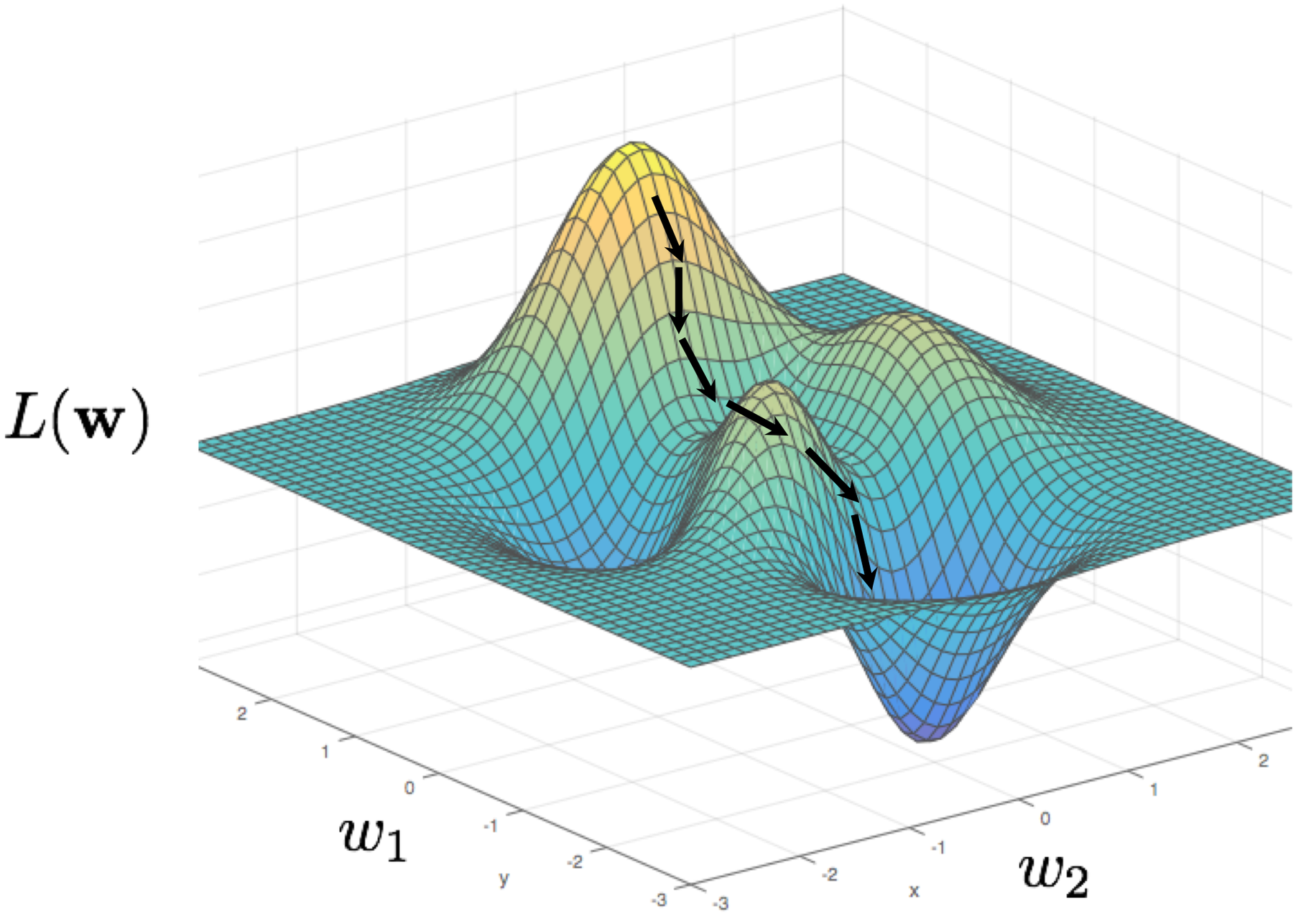
$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \sum_i \ell(\mathbf{z}_i, f(\mathbf{x}_i; \mathbf{w})) = L(\mathbf{w})$$

One iteration of gradient descent:

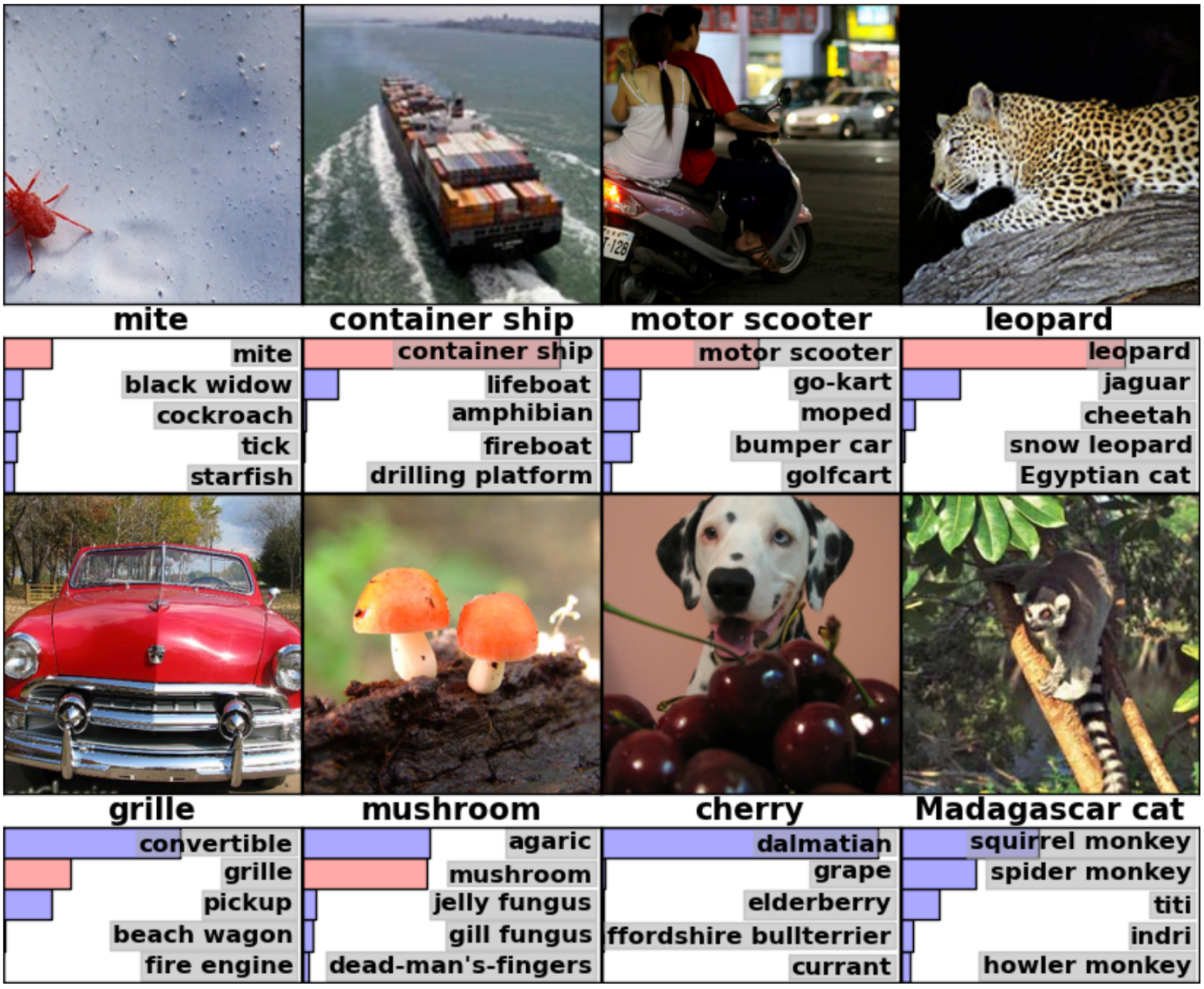
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \frac{\partial L(\mathbf{w}^\mathbf{t})}{\partial \mathbf{w}}$$

learning rate

Gradient descent



$$p(c|\mathbf{x})$$



[Krizhevsky et al. NIPS 2012]