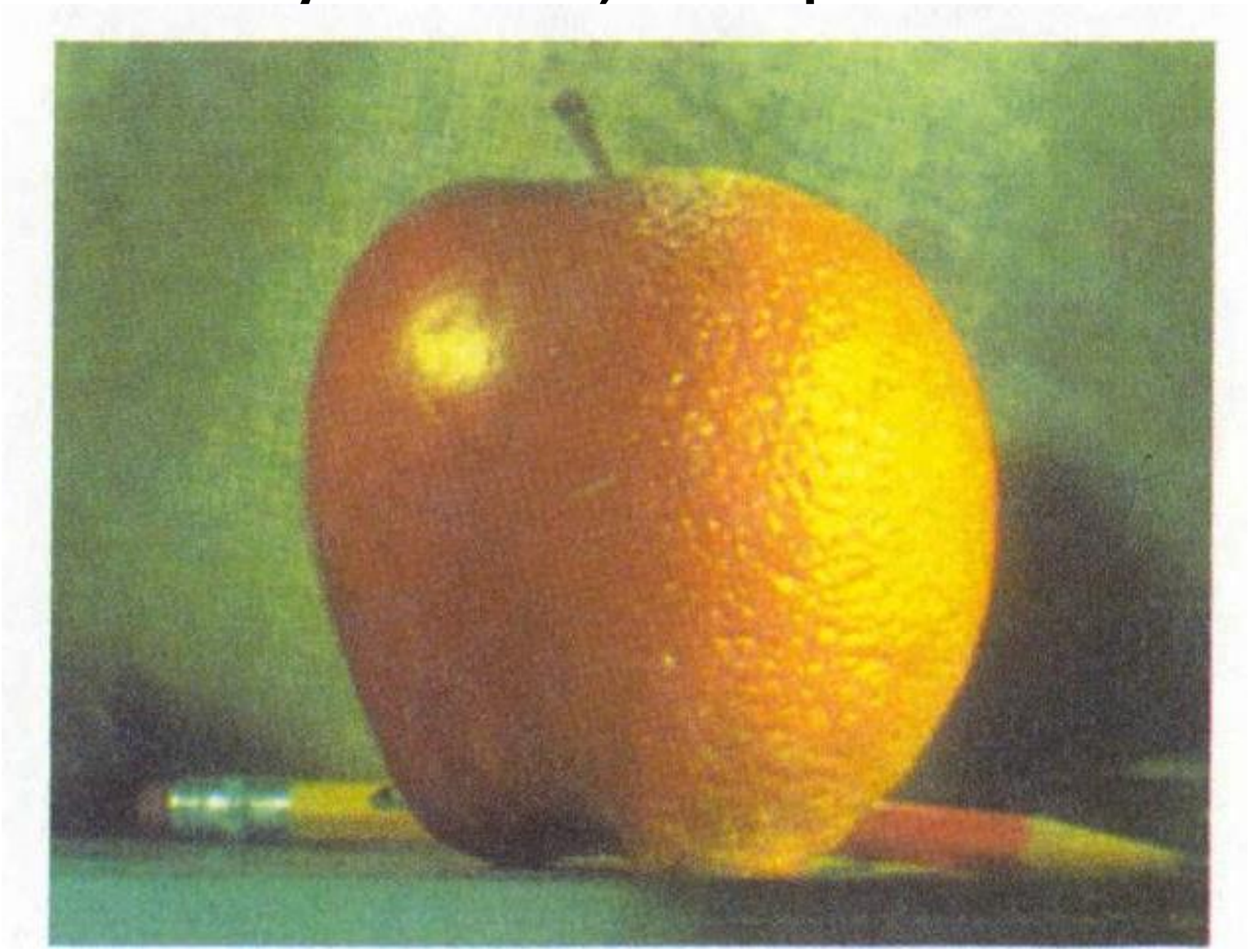
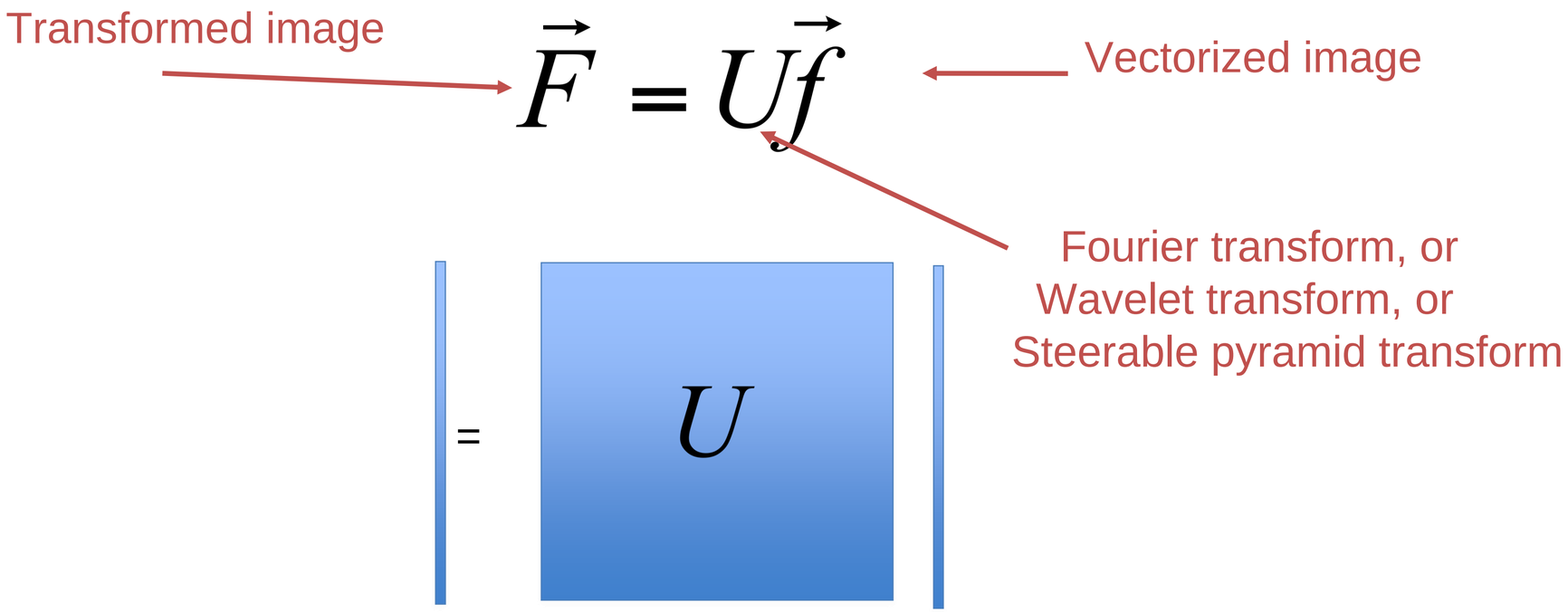
Pyramids, Deep Nets

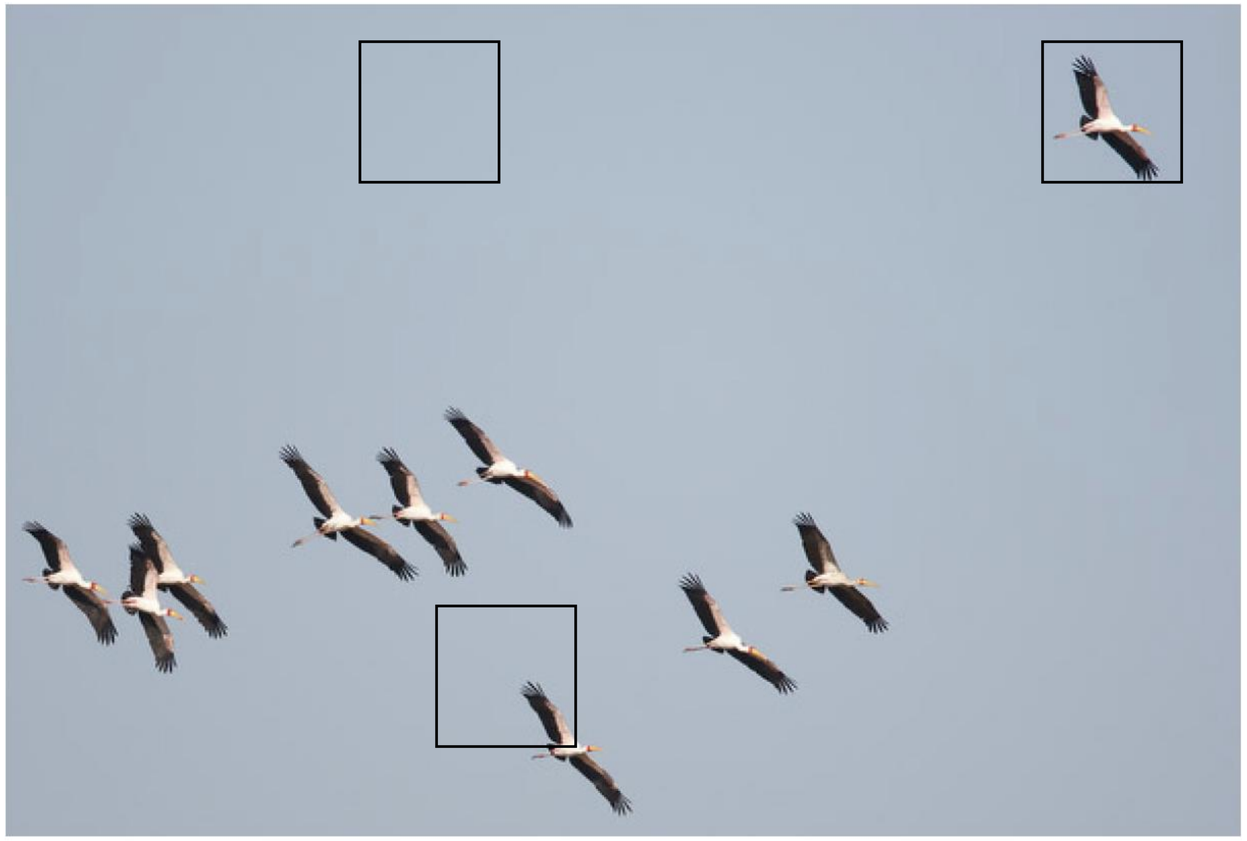


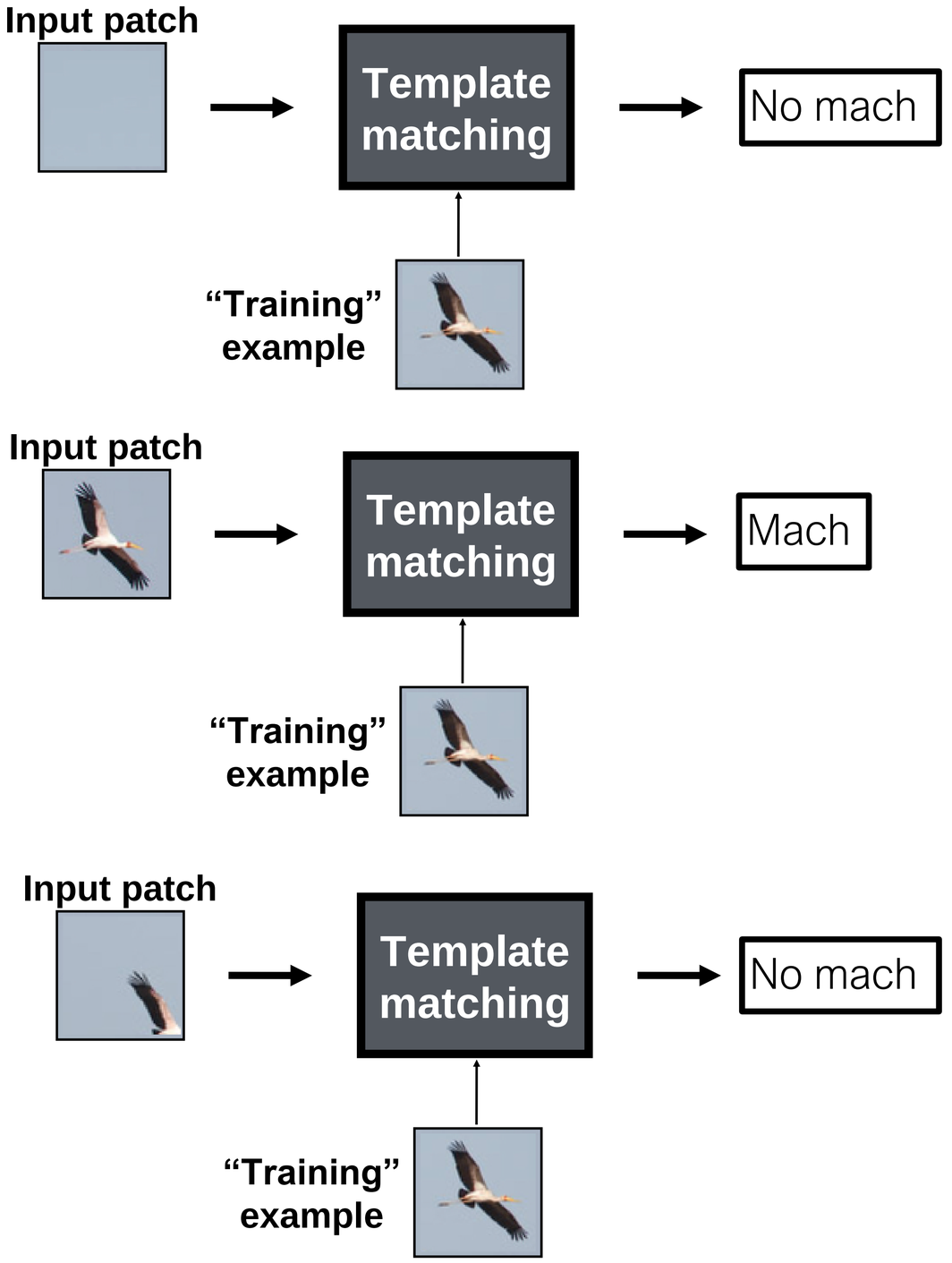
Linear image transformations

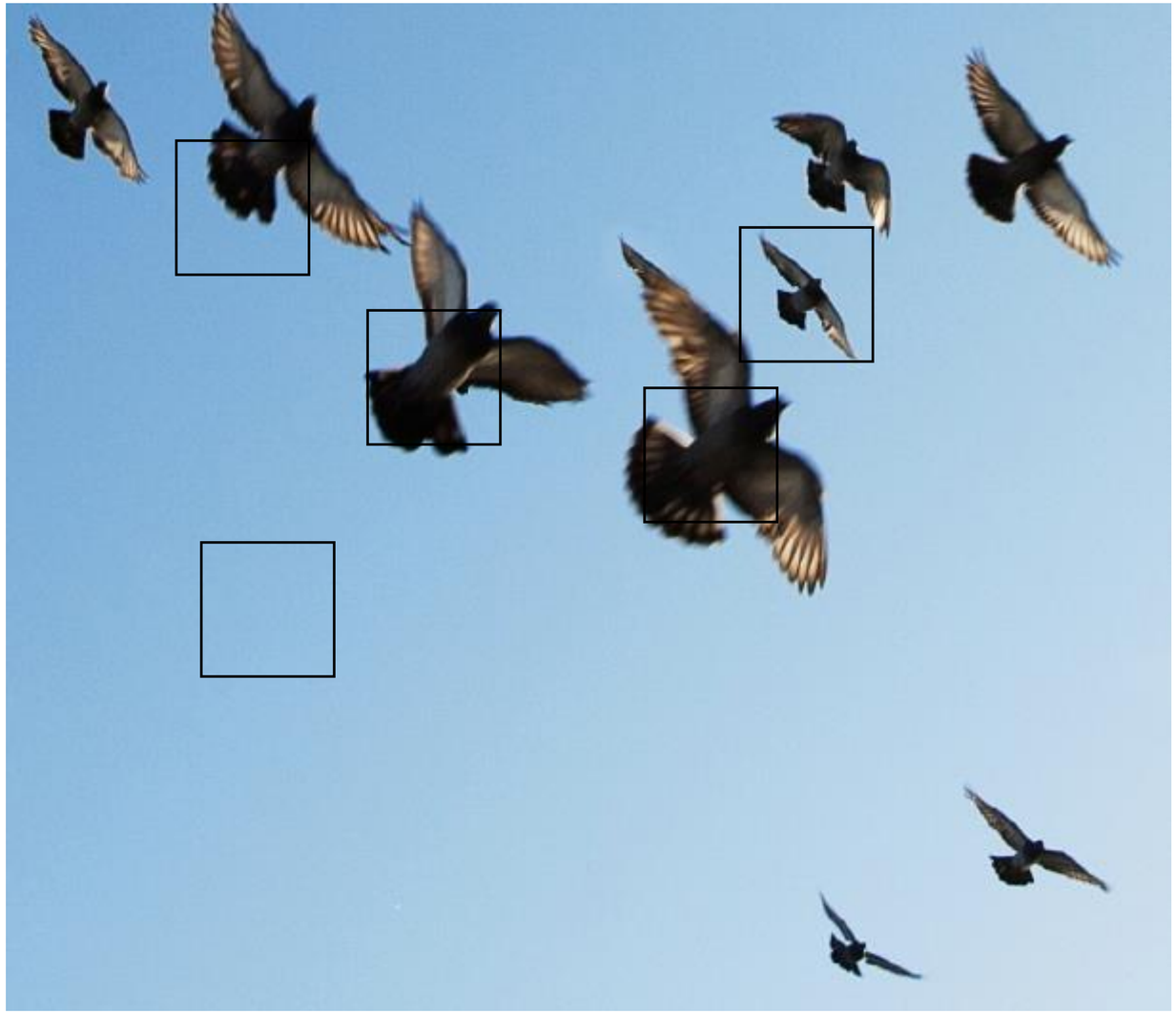
In analyzing images, it's often useful to make a change of basis.



We need translation invariance



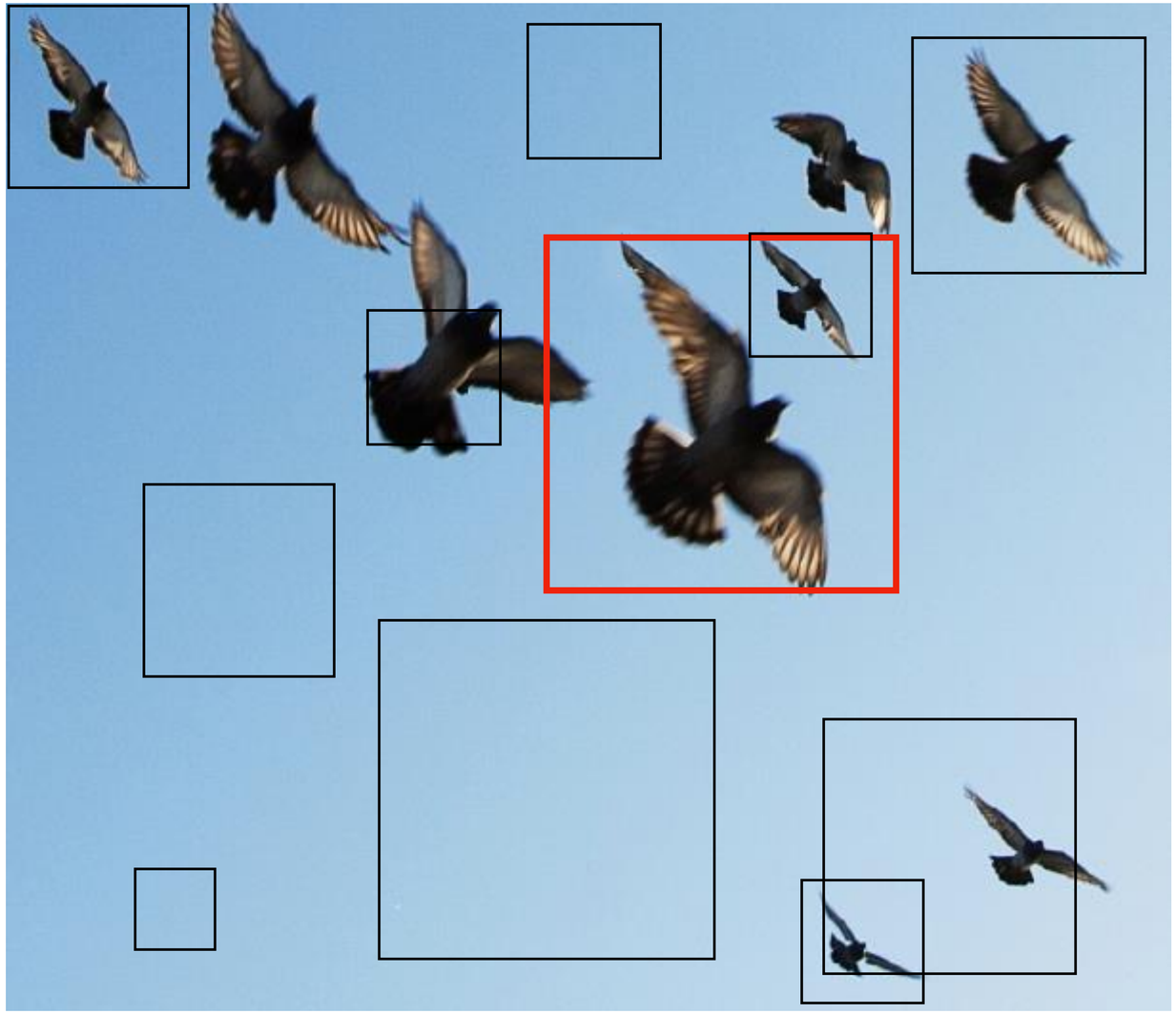

Input patch
Template matching
No mach
“Training” example
Input patch
Template matching
Mach
“Training” example
Input patch
Template matching
No mach
“Training” example

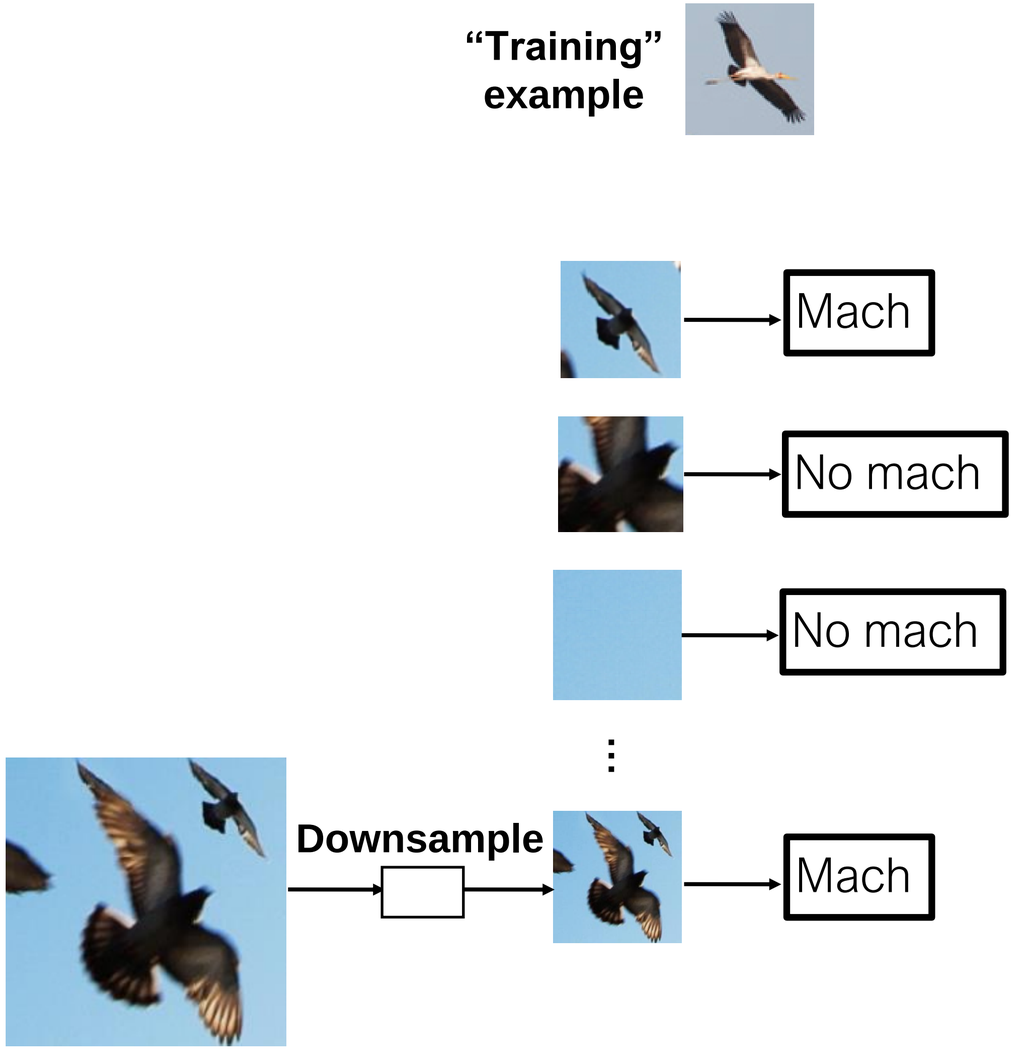


“Training” example

⋮

We need translation and scale invariance



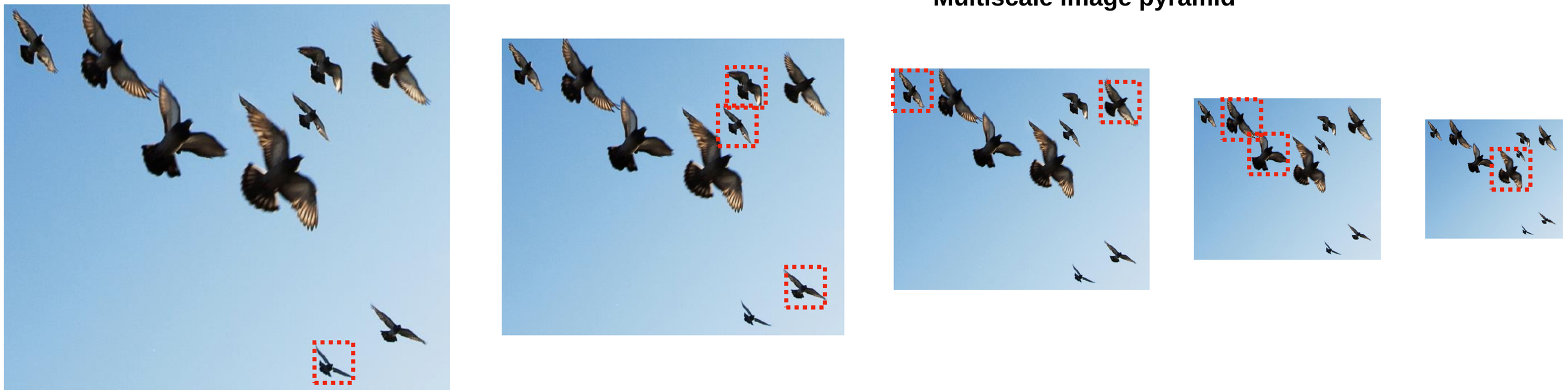


Consider all patch location and sizes

Multi-Scale Pyramids

Template

Multi-Scale Pyramids

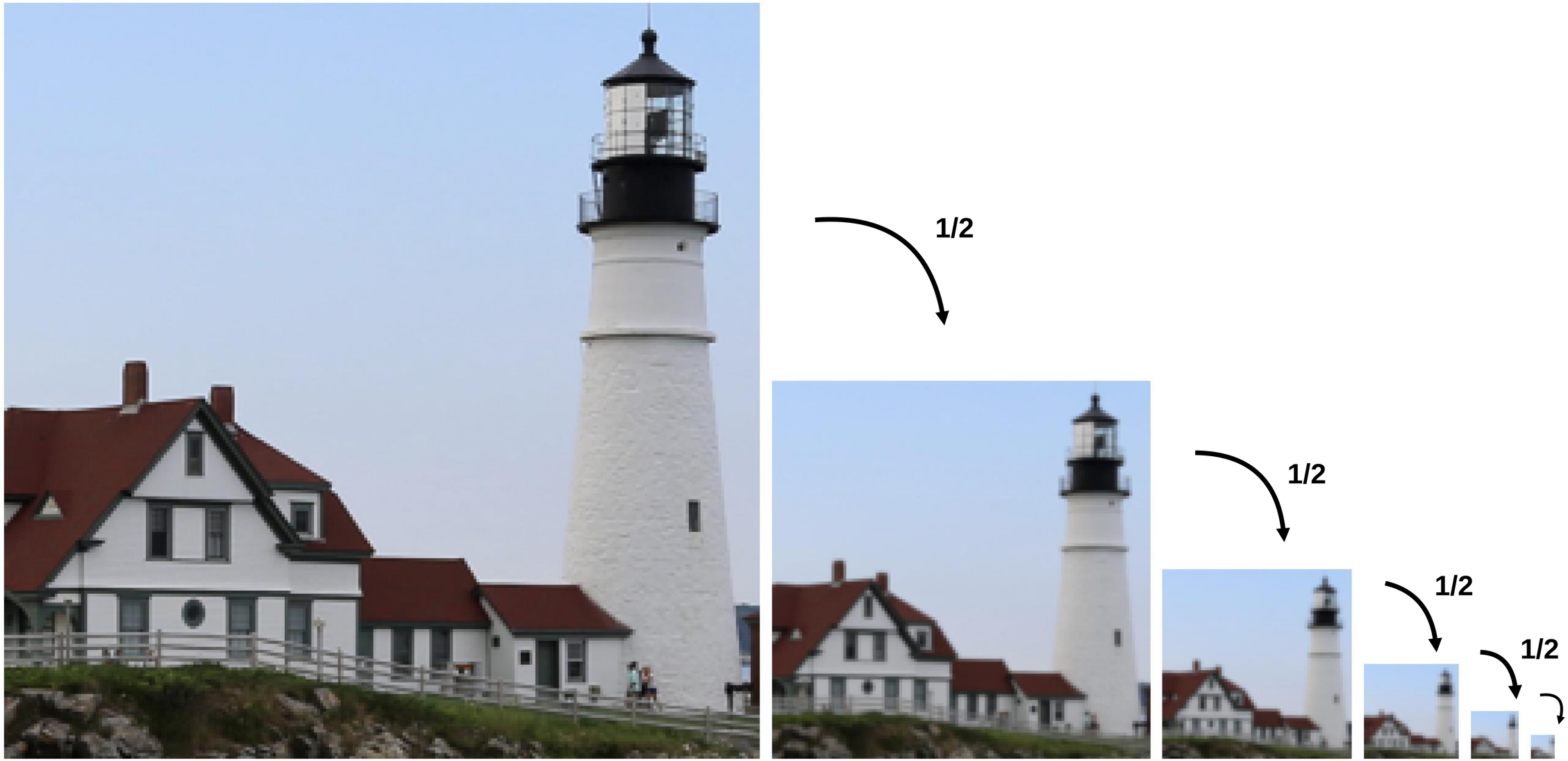


Multiscale image pyramid

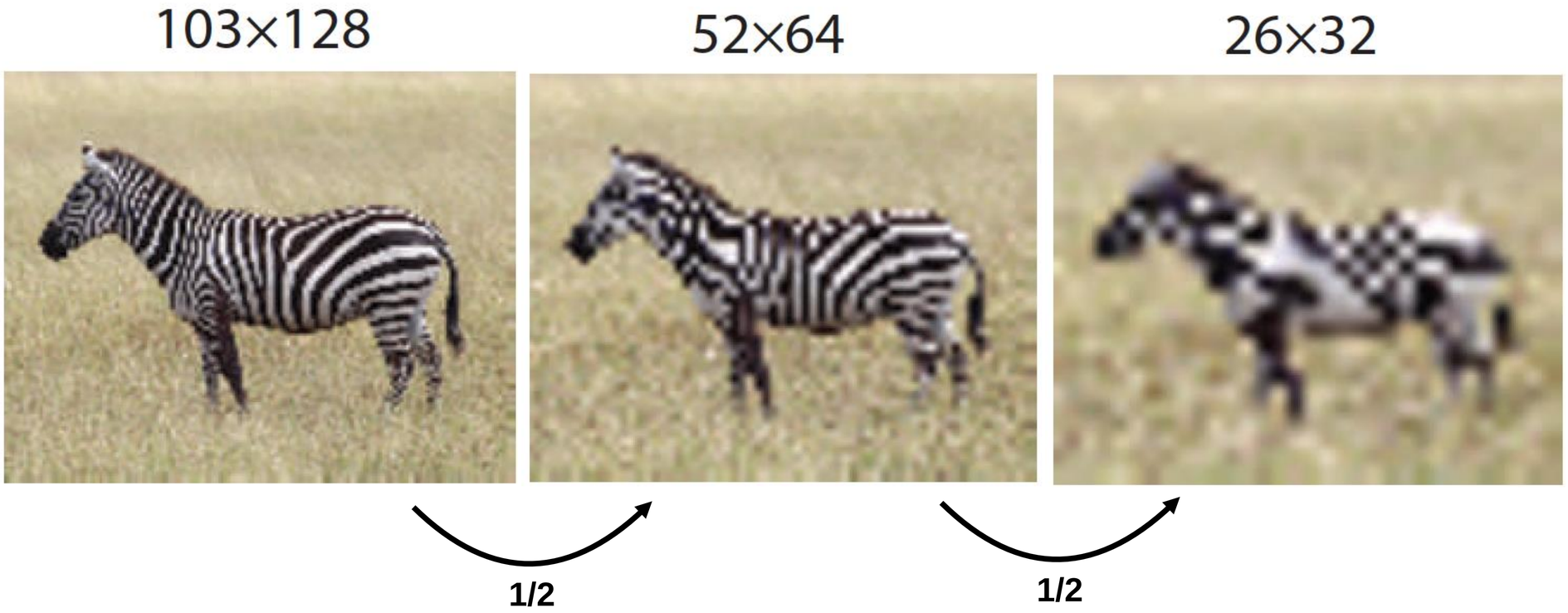
Template

A multiscale image pyramid provides an alternative image representation to achieve translation and scale invariance

Gaussian Pyramid



Subsampling and aliasing



The Gaussian pyramid

For each level

1. Blur input image with a Gaussian filter

[1, 4, 6, 4, 1]

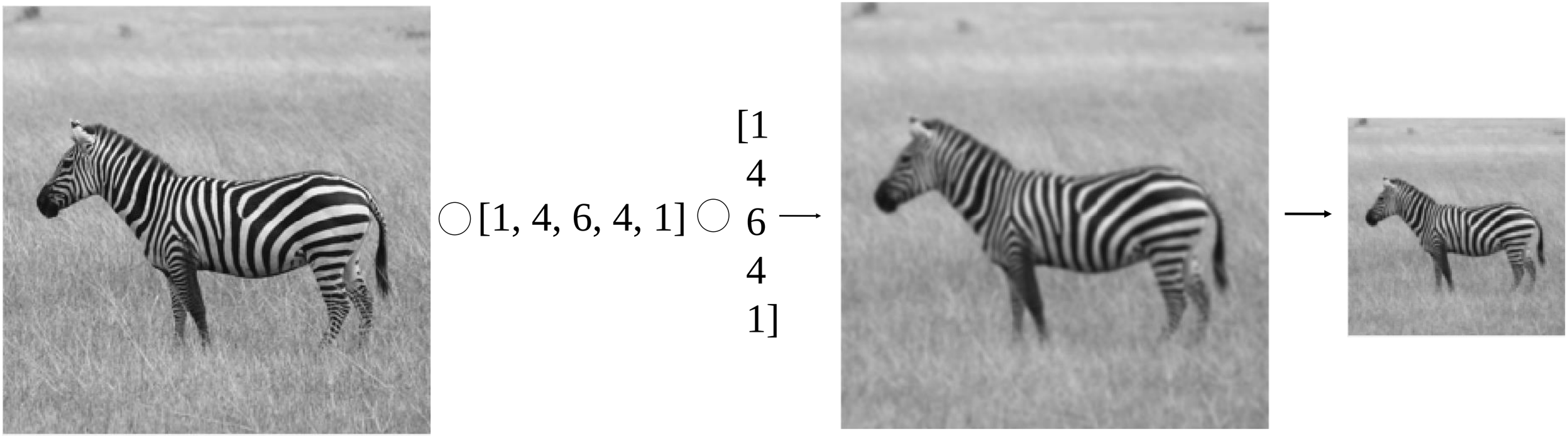
$\bigcirc$ [1, 4, 6, 4, 1] $\bigcirc$ $\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix}$ $\longrightarrow$

(The Gaussian filter is approximated with a binomial filter)

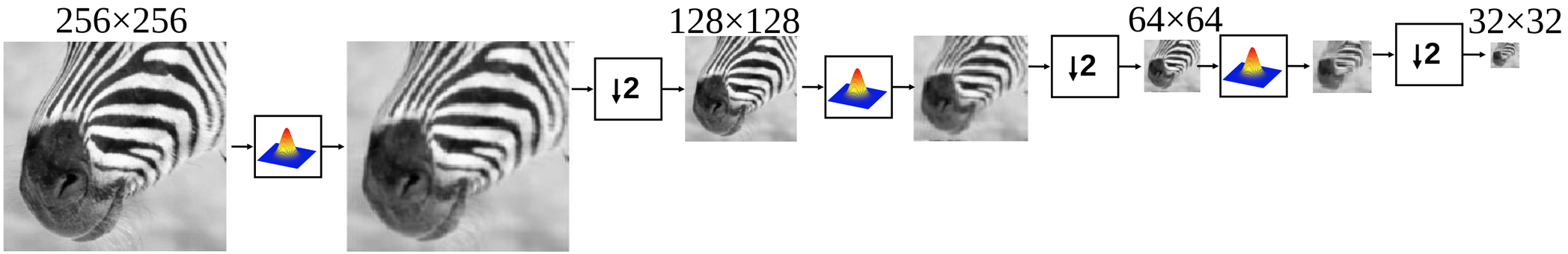
The Gaussian pyramid

For each level

1. Blur input image with a Gaussian filter
2. Downsample image



The Gaussian pyramid



The Gaussian pyramid

512×512

(original image)

256×256

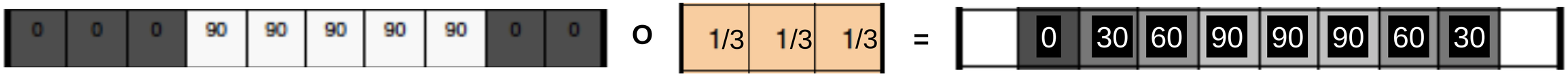
128×128

64×64

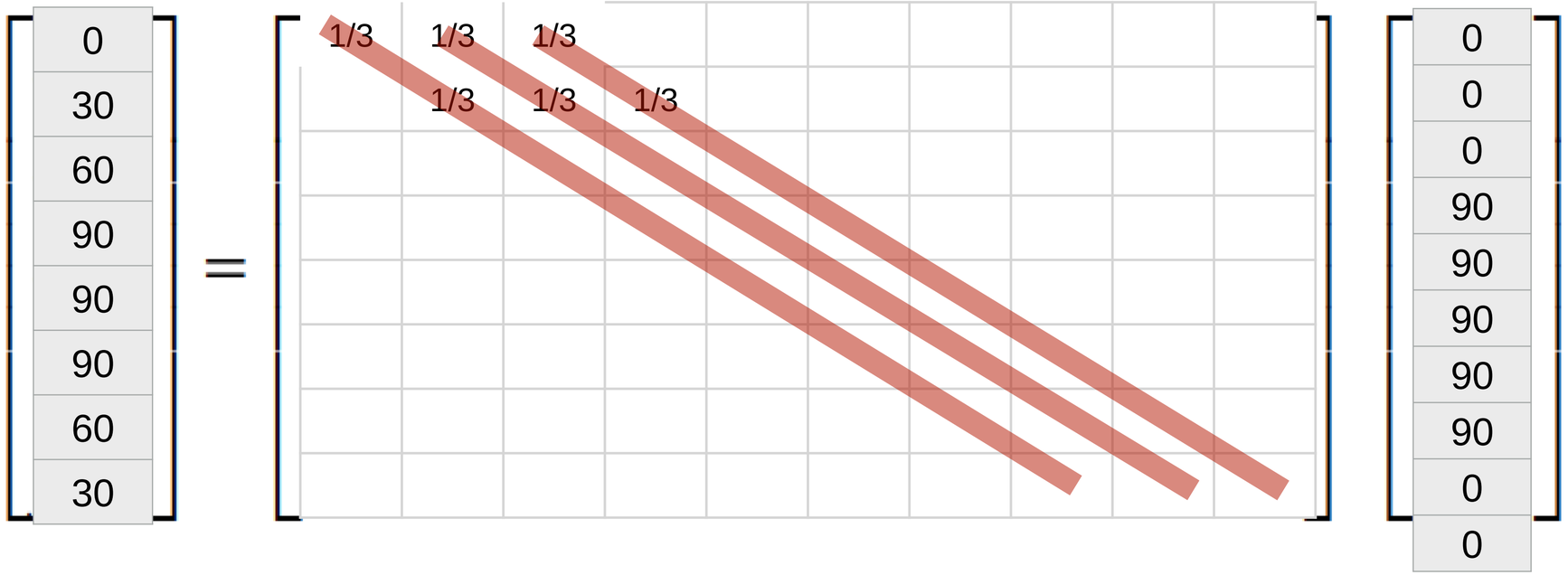
32×32

Convolution as matrix multiplication

In the 1D case, it helps to make explicit the structure of the matrix:



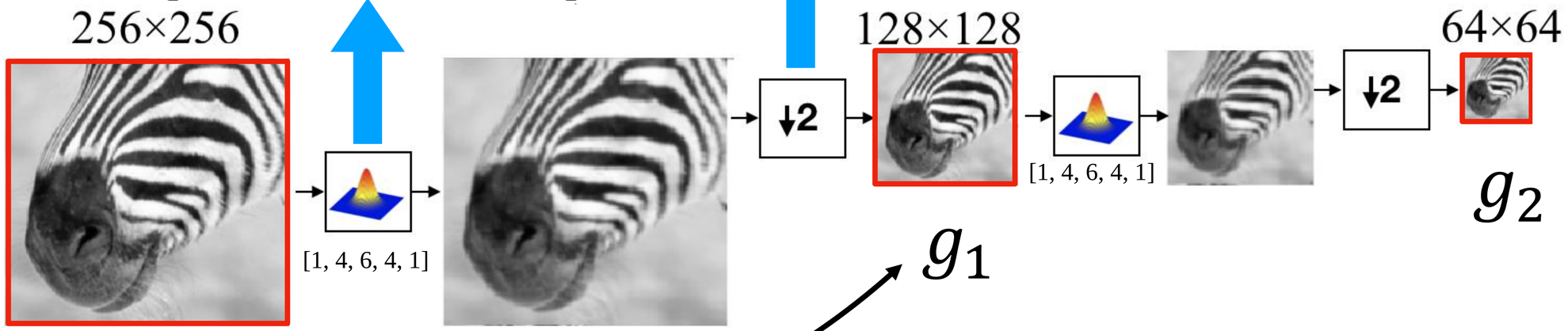
In the 1D case, it helps to make explicit the structure of the matrix:



Reminder from Lecture 3

The Gaussian pyramid

$$\frac{1}{16}\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



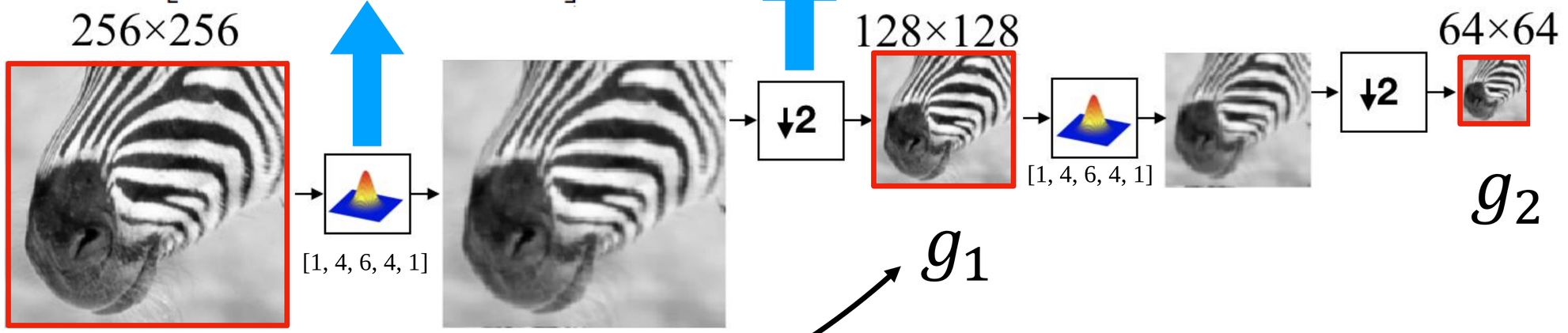
g_0 g_1 g_2

$$g_1 = G_0 g_0 \qquad G_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{16} \begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}$$

(The arrays shown here are for 1D signals)

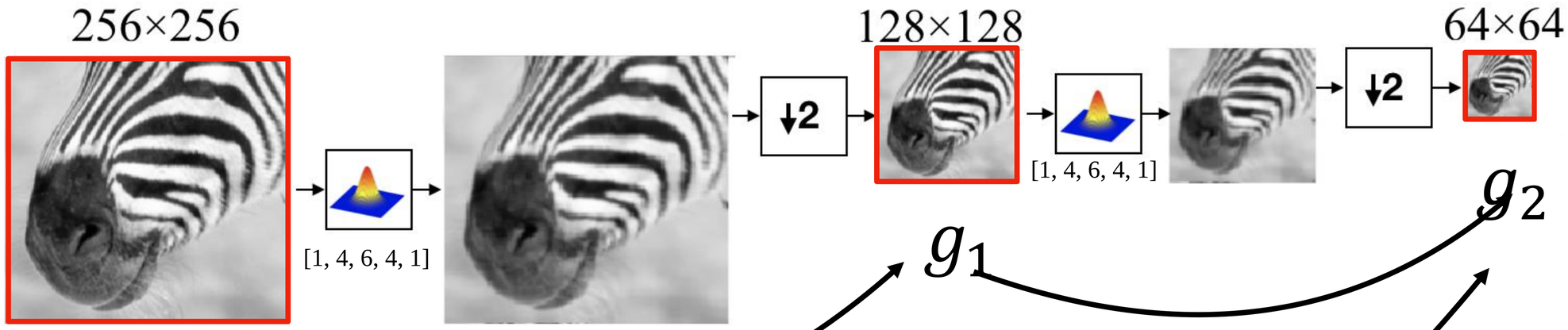
The Gaussian pyramid

$$\frac{1}{16}\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$$g_1 = G_0 g_0 \qquad G_0 = \frac{1}{16}\begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \end{bmatrix}$$

The Gaussian pyramid



g_0

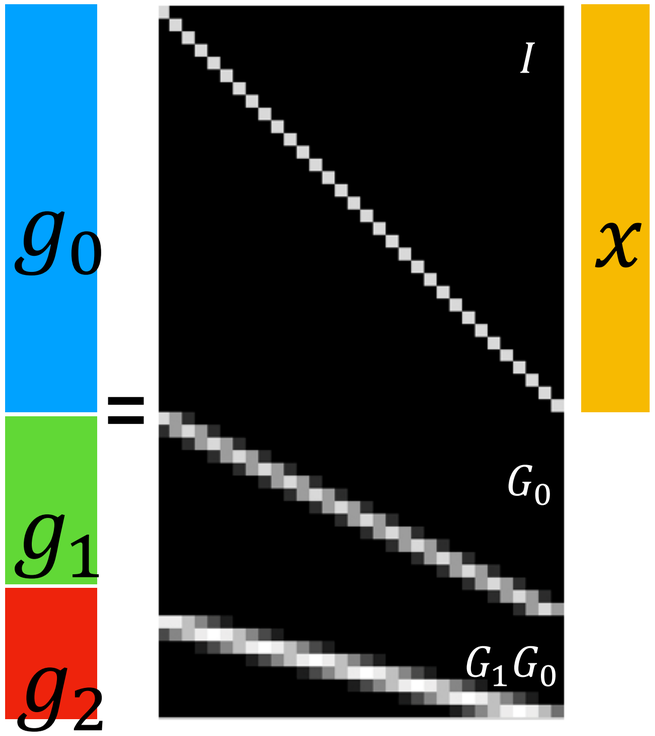
g_1

g_2

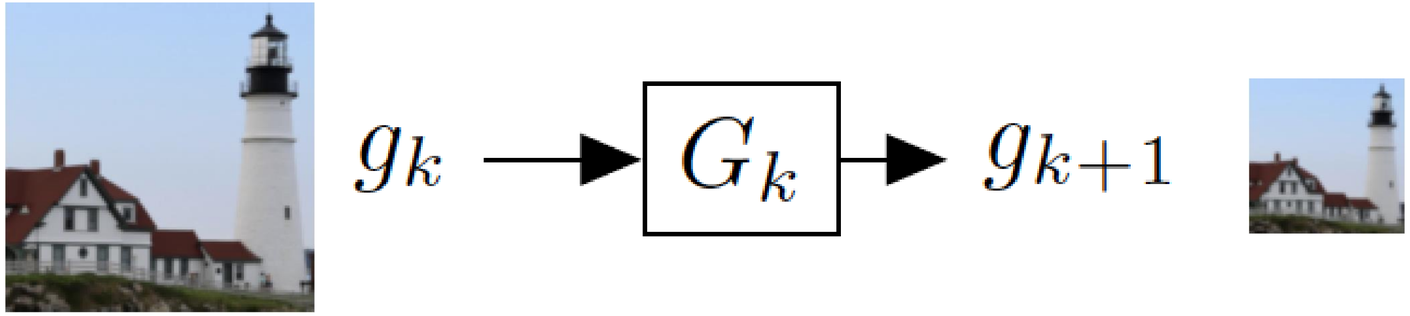
$g_1 = G_0 g_0$

$g_2 = G_1 g_1$

$g_2 = G_1 G_0 g_0$



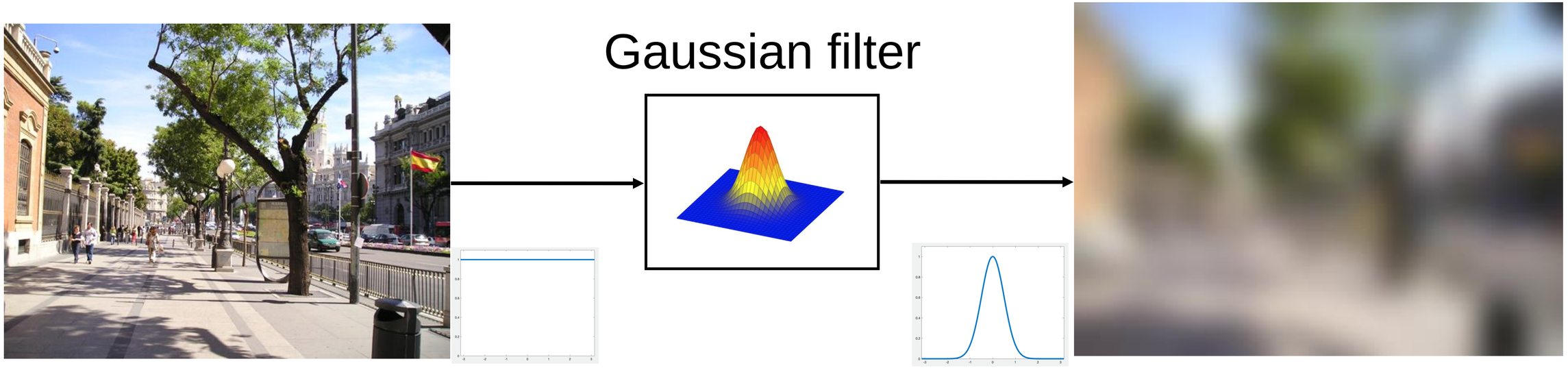
The Gaussian pyramid

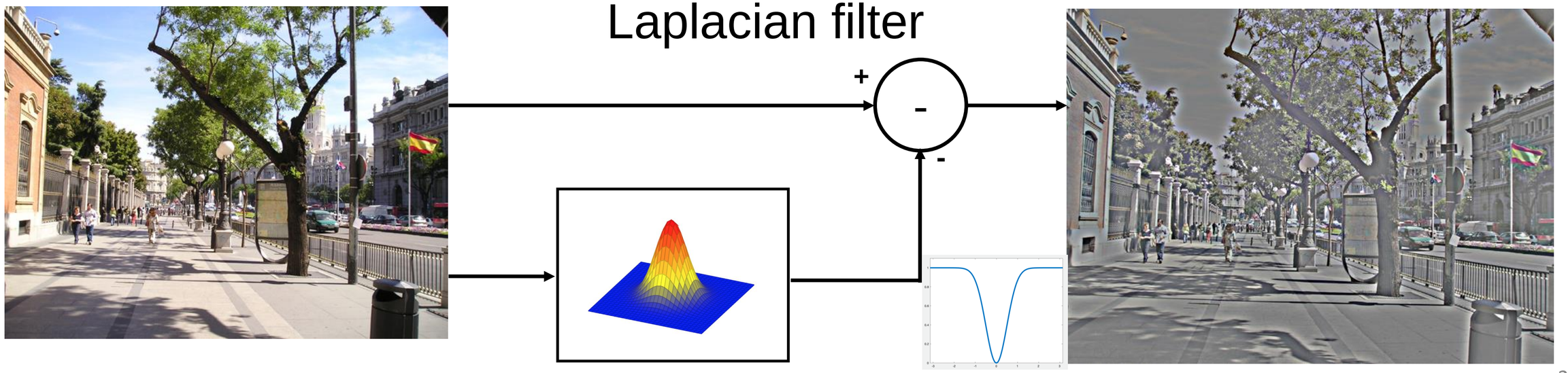


For each level

1. Blur input image with a Gaussian filter
2. Downsample image

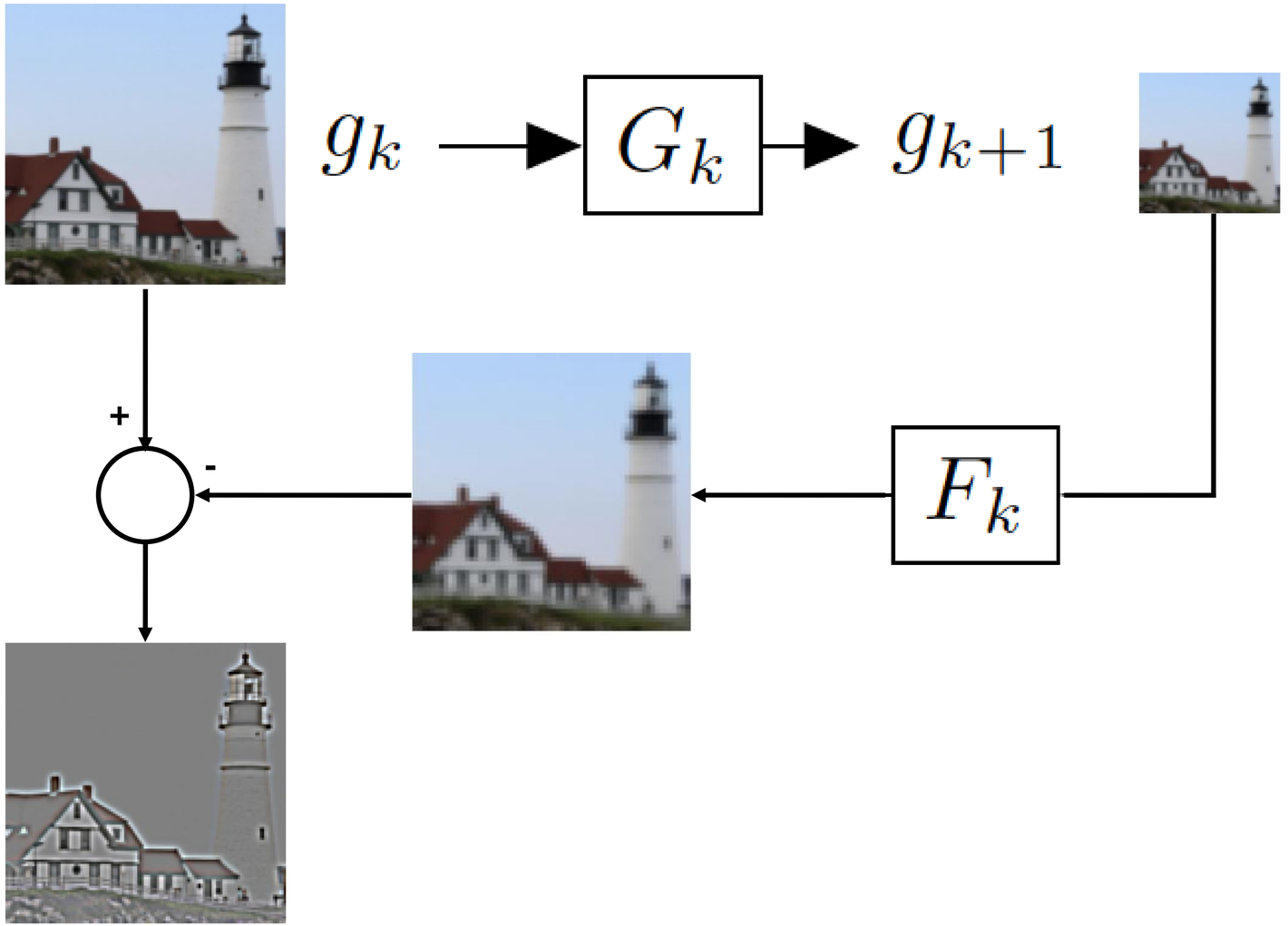
What about the opposite of blurring?



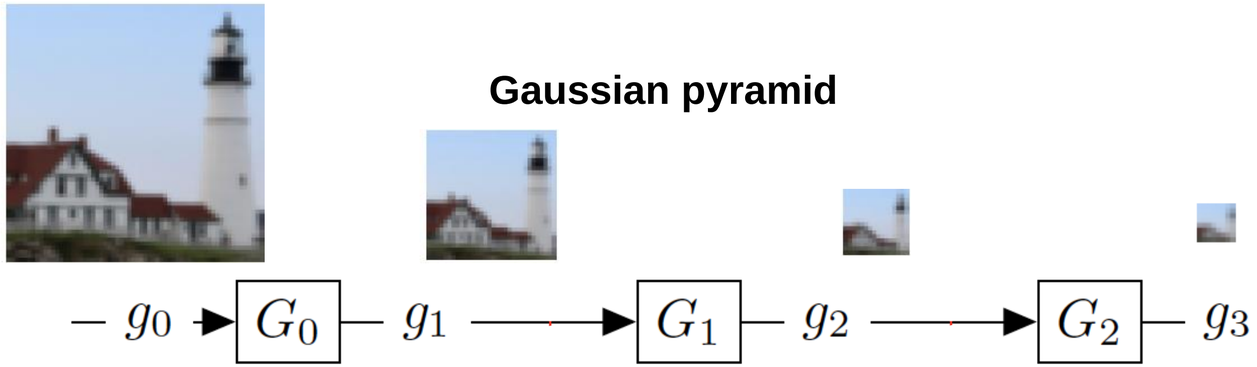


The Laplacian Pyramid

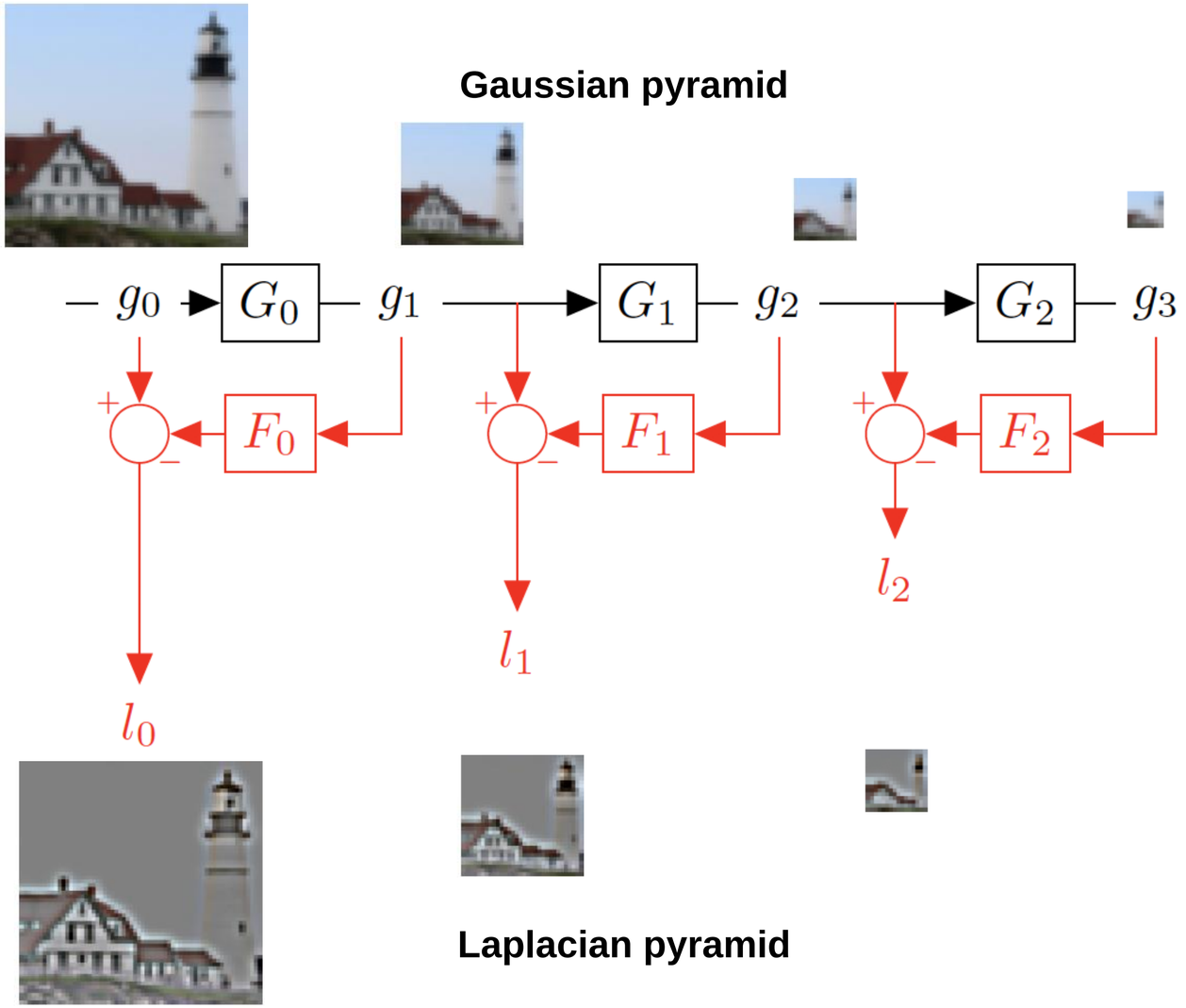
Compute the difference between upsampled Gaussian pyramid level k+1 and Gaussian pyramid level k.



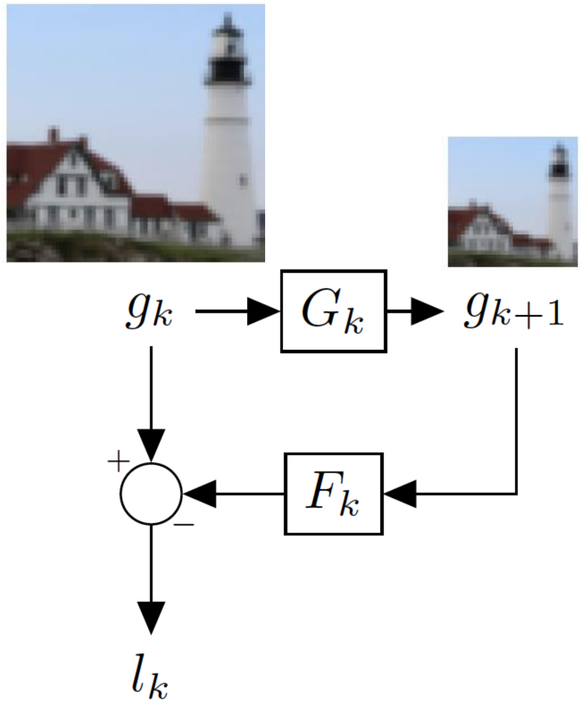
The Laplacian Pyramid



The Laplacian Pyramid



The Laplacian Pyramid



Blurring and downsampling:

$$G_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{16} \begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}$$

(Downsampling by 2) (blur)

Upsampling and blurring:

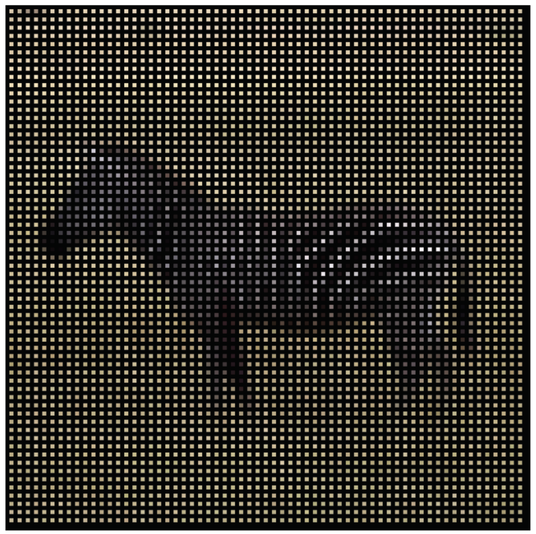
$$F_0 =$$

Upsampling

64x64

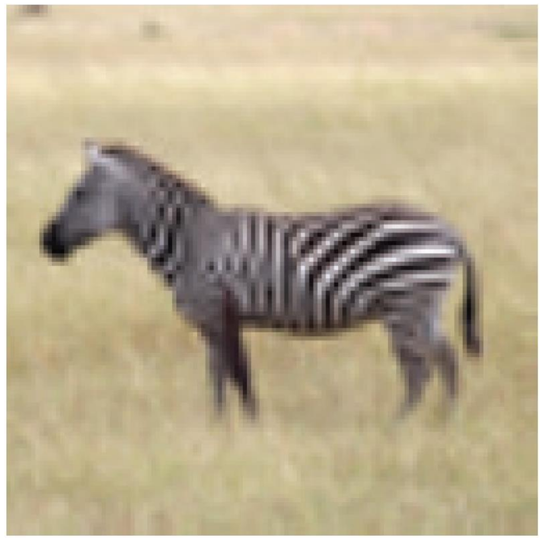
Insert zeros

128x128



$$\bigcirc \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix} =$$

128x128



64x64

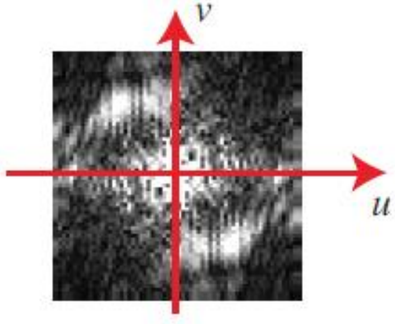
128x128

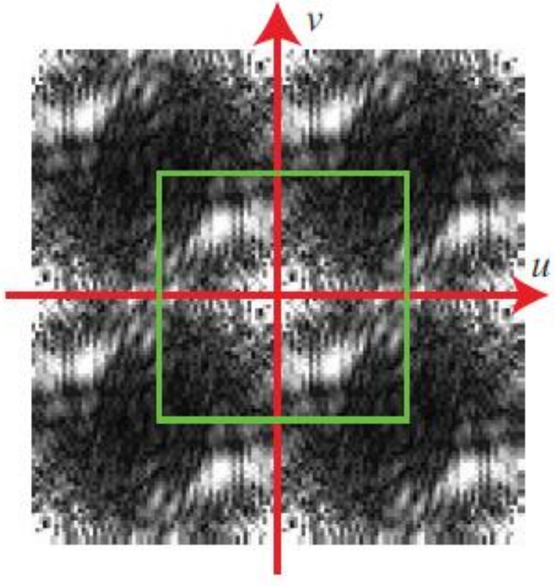
$\bigcirc$

$$\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$$

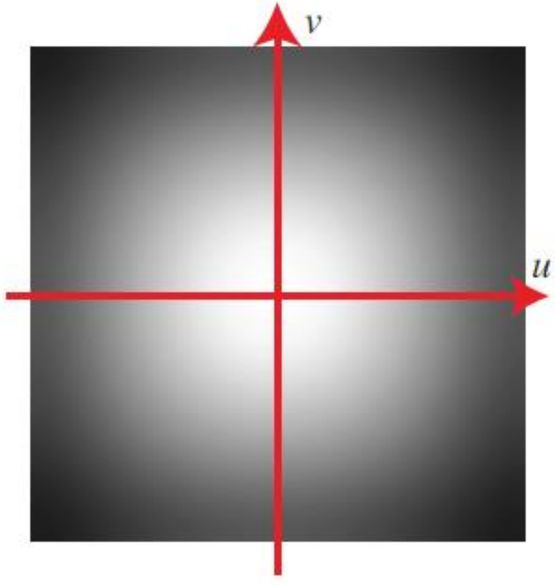
$=$

128x128

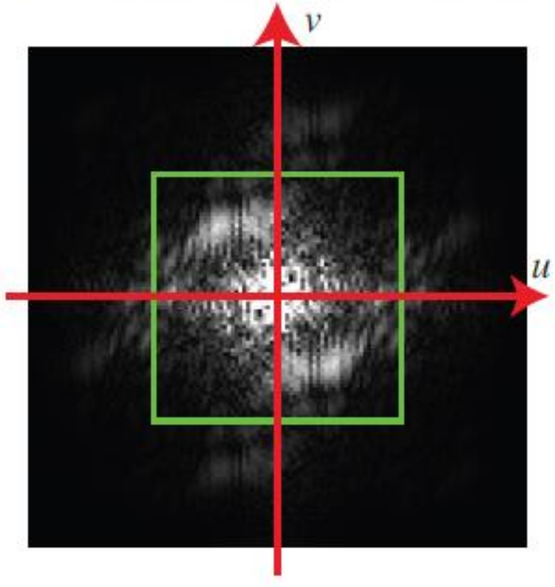




$\times$



$=$



Upsampling

1	0	1	0	1
0	0	0	0	0
1	0	1	0	1
0	0	0	0	0
1	0	1	0	1

$\bigcirc \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$ **= ?**

Upsampling

1	0	1	0	1
0	0	0	0	0
1	0	1	0	1
0	0	0	0	0
1	0	1	0	1

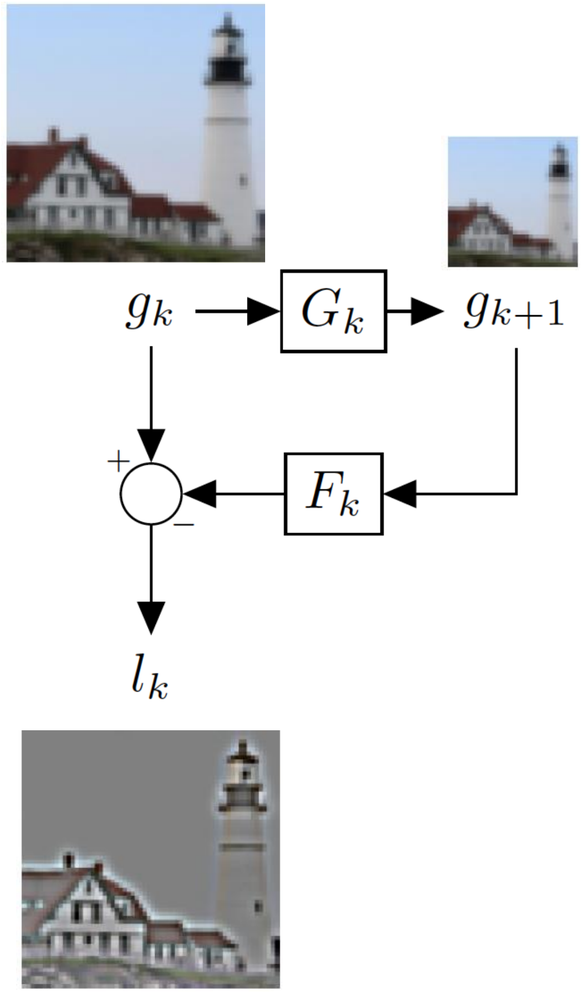
$\bigcirc$

$$\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$$

=

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

The Laplacian Pyramid



Blurring and downsampling:

$$G_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{16} \begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix}$$

(Downsampling by 2) (blur)

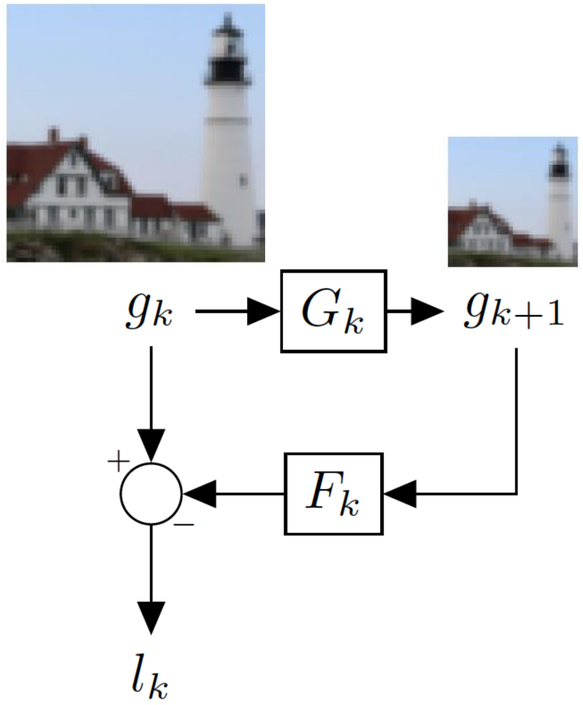
Upsampling and blurring:

$$F_0 = \frac{1}{8} \begin{bmatrix} 6 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 6 & 4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 & 6 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 4 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(blur) (Upsampling by 2)

$$l_0 = (I_0 - F_0 G_0) g_0$$

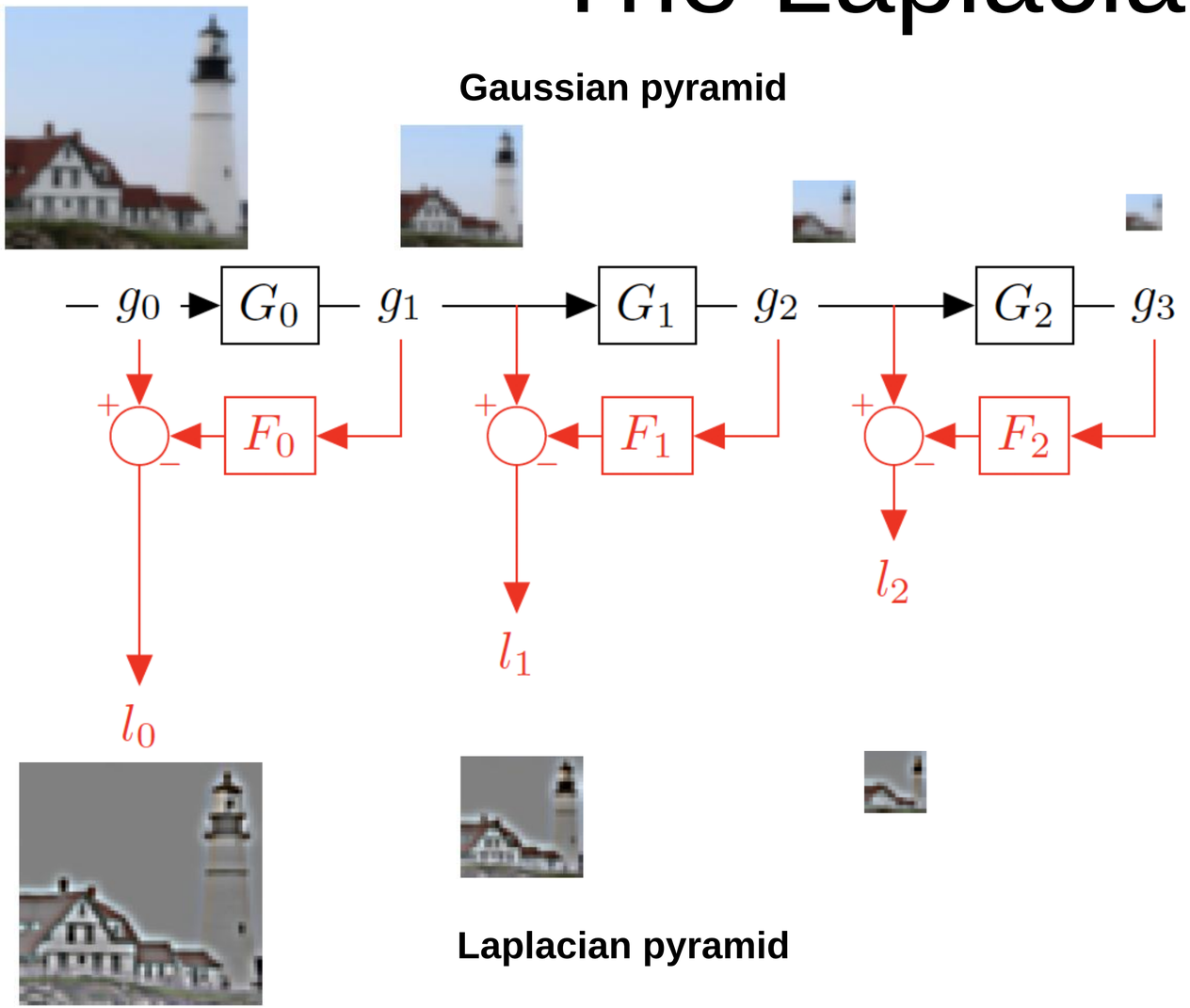
The Laplacian Pyramid

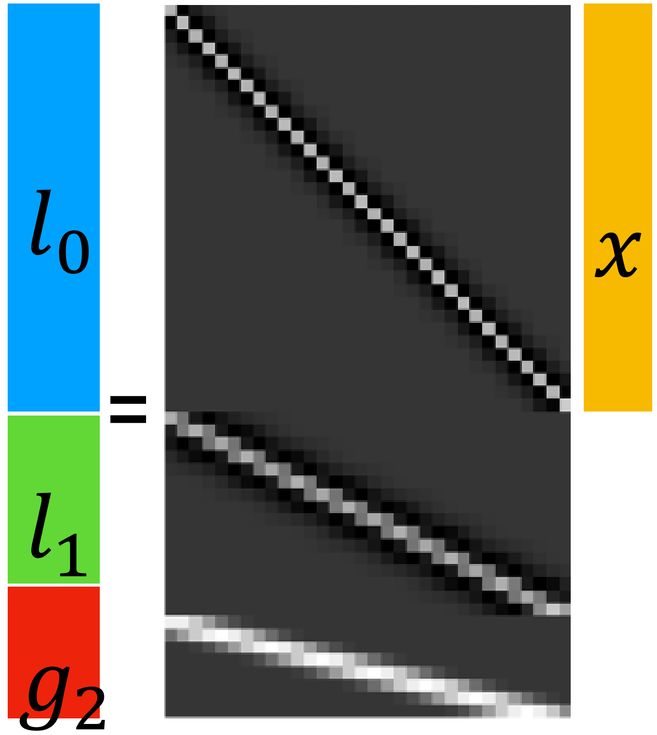


$$l_0 = (I_0 - F_0 G_0) g_0$$

$$= \frac{1}{256} \begin{bmatrix} 182 & -56 & -24 & -8 & -2 & 0 & 0 & 0 \\ -56 & 192 & -56 & -32 & -8 & 0 & 0 & 0 \\ -24 & -56 & 180 & -56 & -24 & -8 & -2 & 0 \\ -8 & -32 & -56 & 192 & -56 & -32 & -8 & 0 \\ -2 & -8 & -24 & -56 & 180 & -56 & -24 & -8 \\ 0 & 0 & -8 & -32 & -56 & 192 & -56 & -32 \\ 0 & 0 & -2 & -8 & -24 & -56 & 182 & -48 \\ 0 & 0 & 0 & 0 & -8 & -32 & -48 & 224 \end{bmatrix} x$$

The Laplacian Pyramid





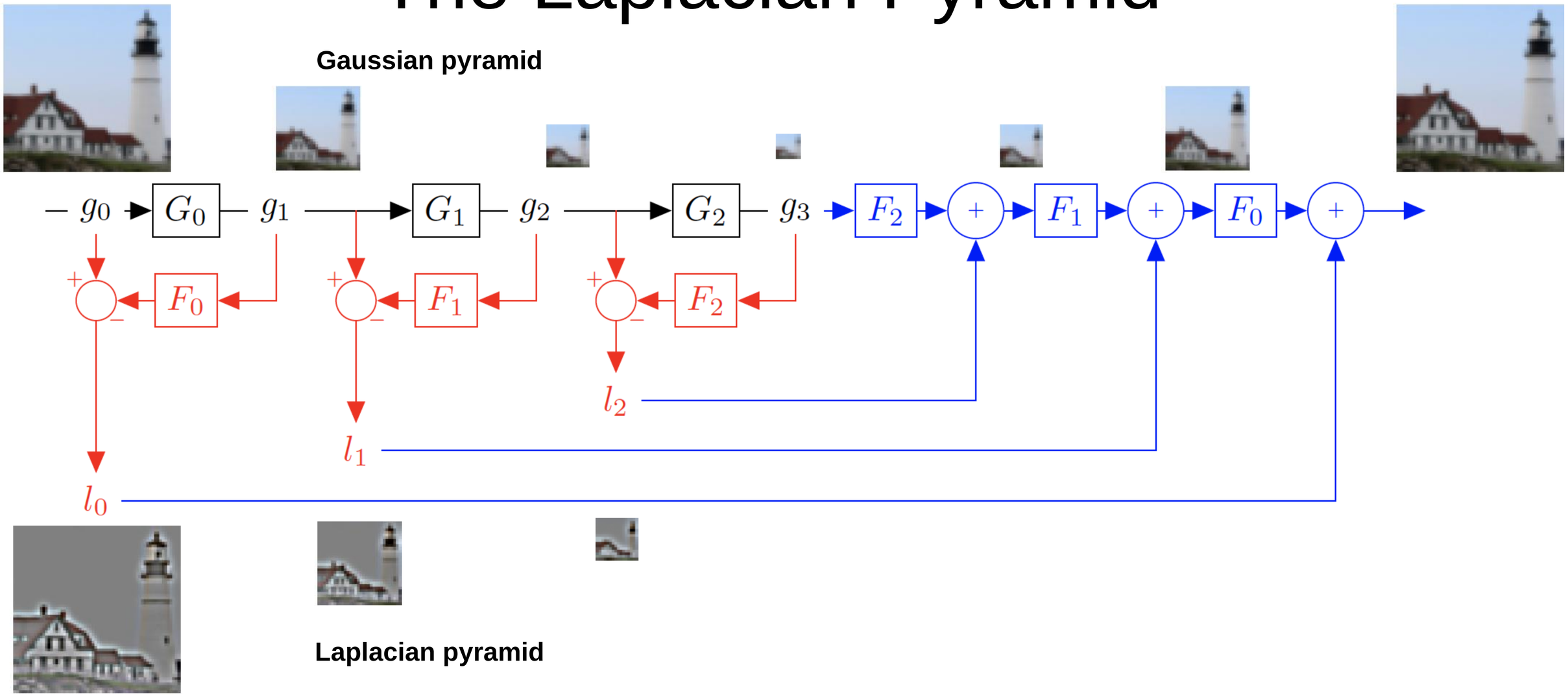
The Laplacian Pyramid

Gaussian residual

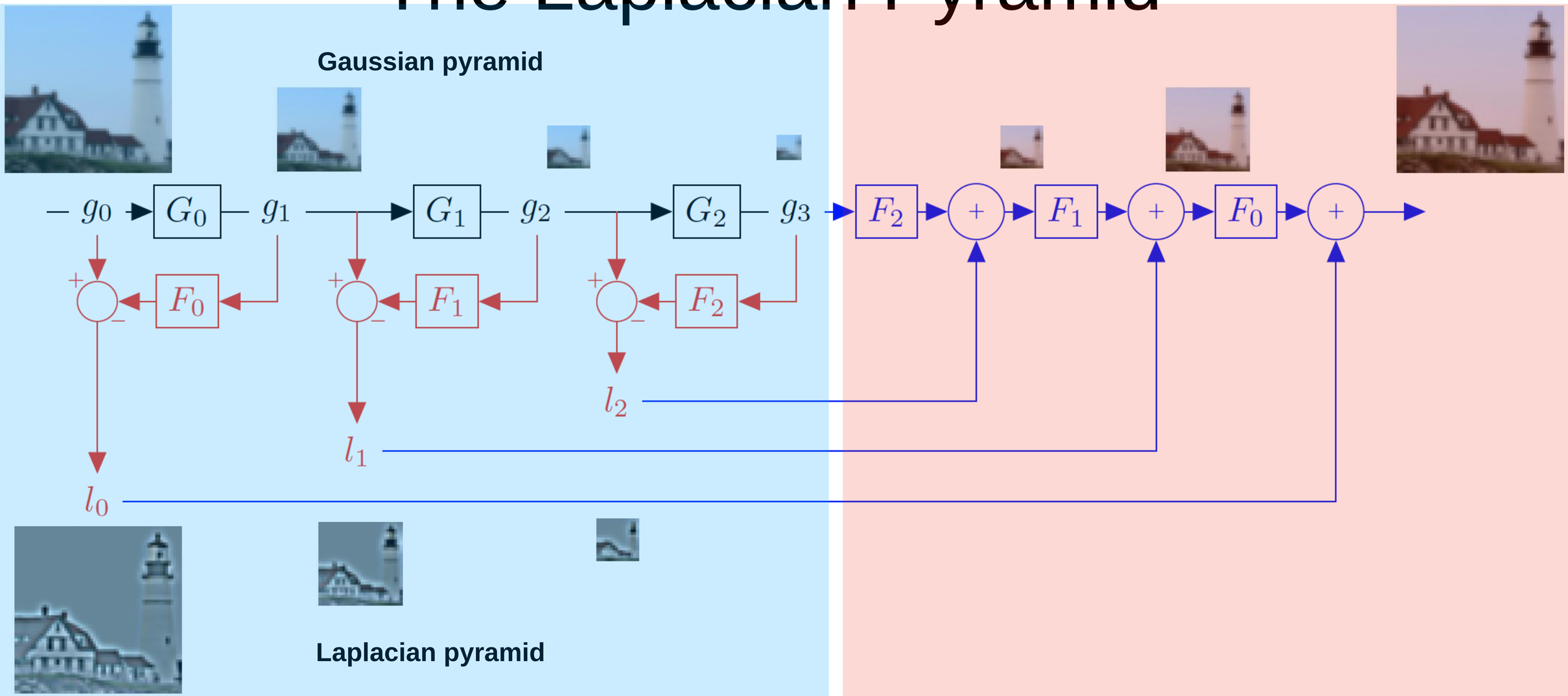
Laplacian pyramid

Can we invert the Laplacian Pyramid?

The Laplacian Pyramid



The Laplacian Pyramid



The Laplacian Pyramid

Laplacian pyramid

Gaussian residual

Laplacian pyramid

Gaussian residual

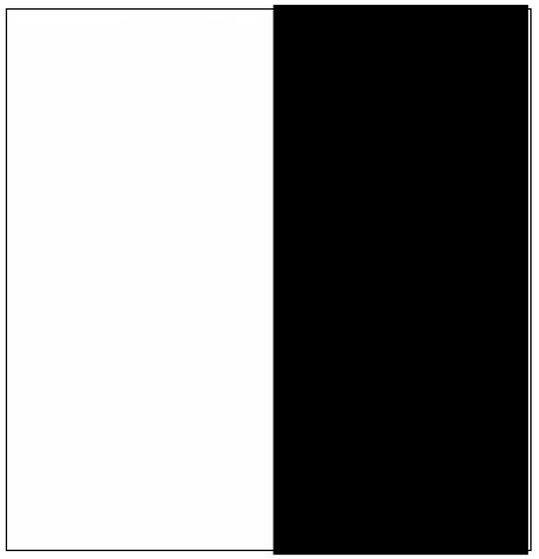
Image Blending

Image Blending

Image Blending

I^A

I^B

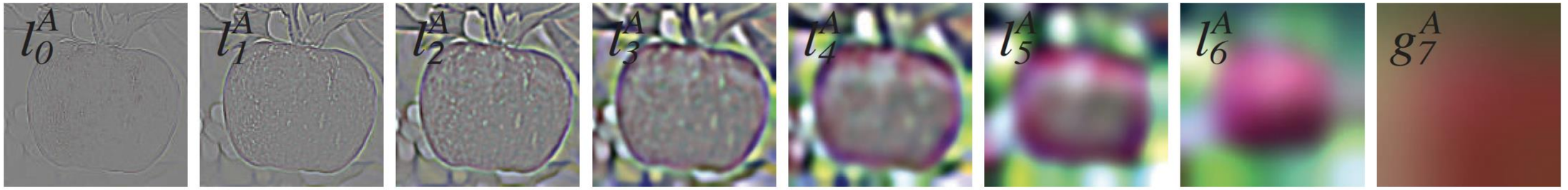


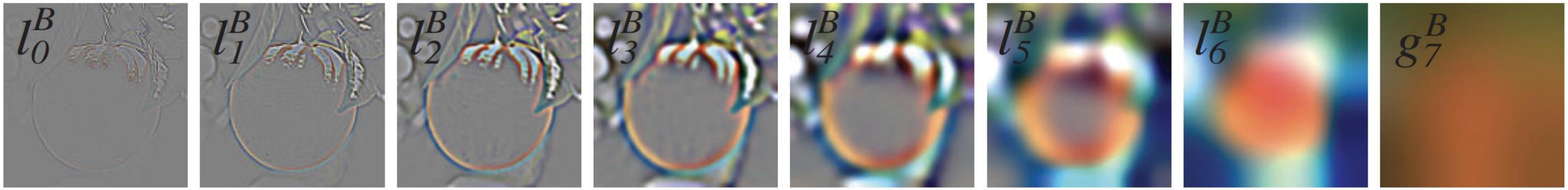
m

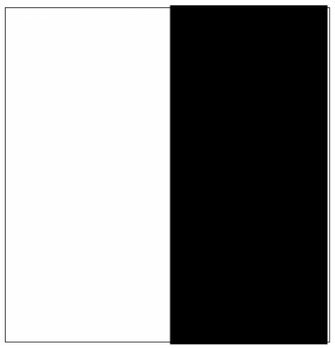
I

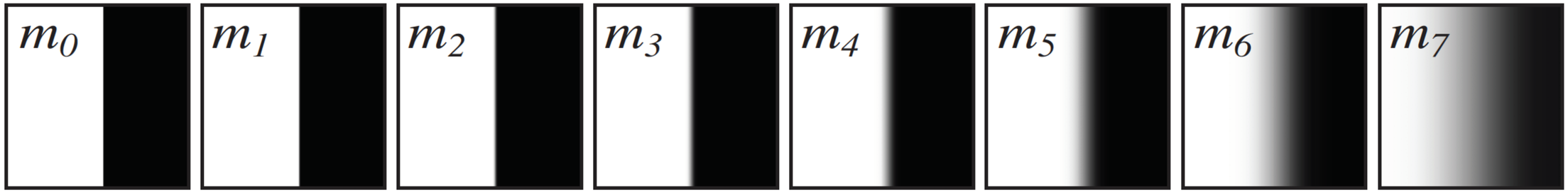
$$I = m * I^A + (1 - m) * I^B$$

Image Blending with the Laplacian Pyramid







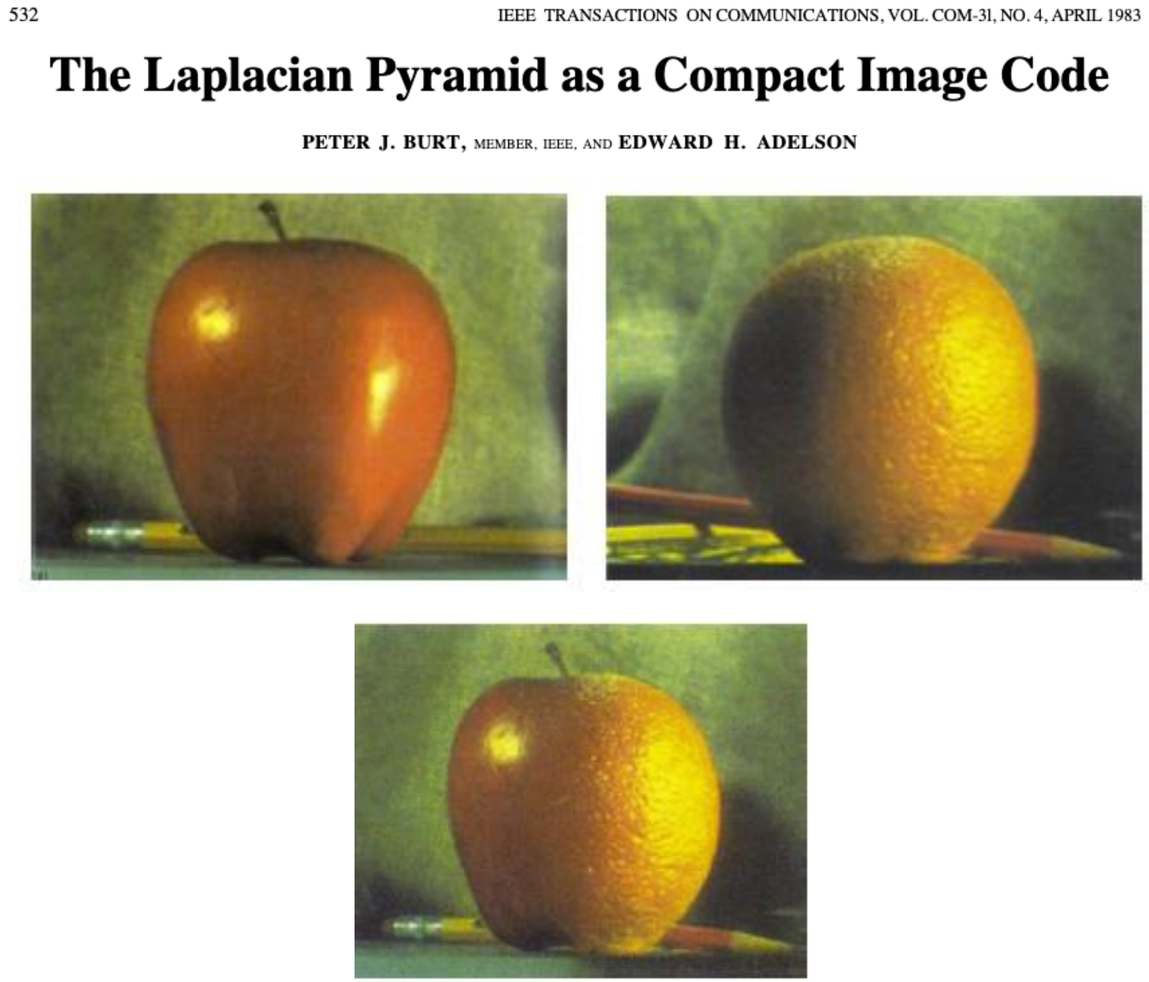


$$l_k = l_k^A * m_k + l_i^B * (1 - m_k)$$

Image Blending with the Laplacian Pyramid

Image Blending with the Laplacian Pyramid

- Build Laplacian pyramid for both images: LA, LB
- Build Gaussian pyramid for mask: G
- Build a combined Laplacian pyramid:
- Collapse L to obtain the blended image



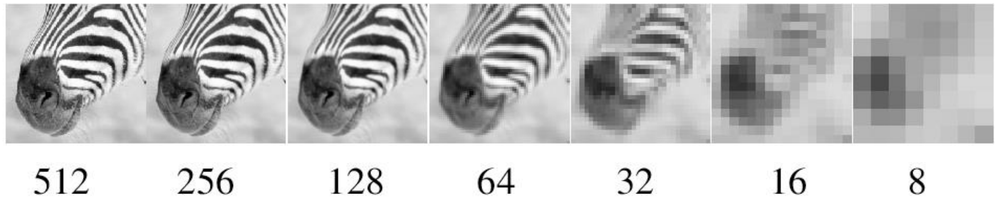
532 IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-31, NO. 4, APRIL 1983

The Laplacian Pyramid as a Compact Image Code

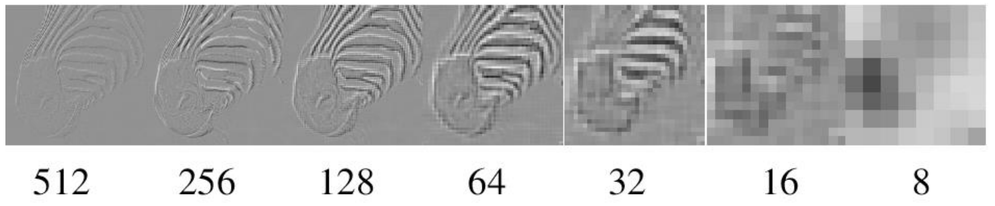
PETER J. BURT, MEMBER, IEEE, AND EDWARD H. ADELSON

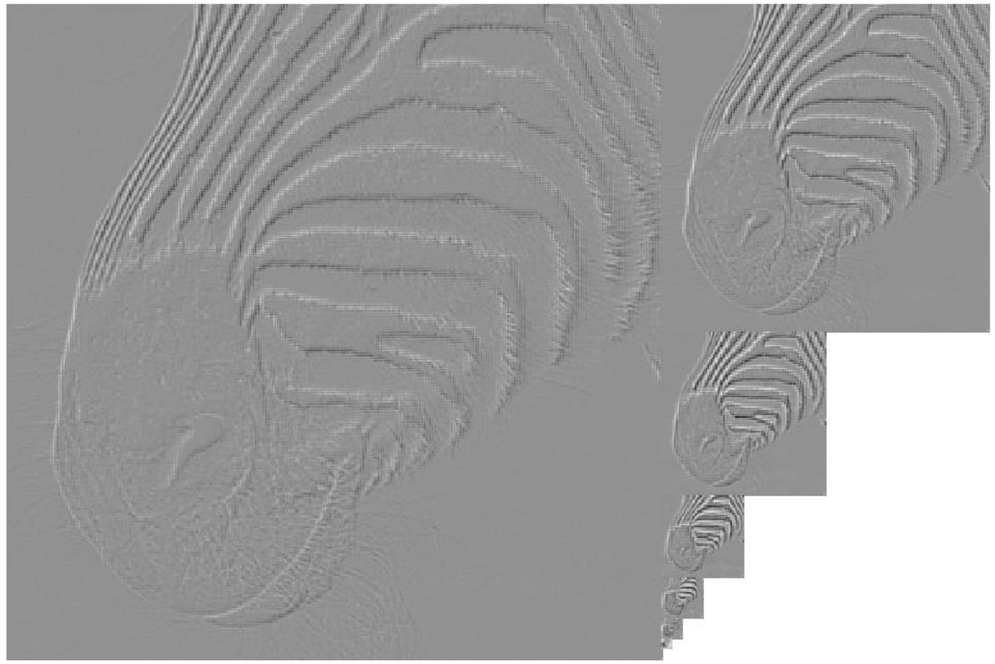
http://persci.mit.edu/pub_pdfs/pyramid83.pdf

Image pyramids



Gaussian Pyr

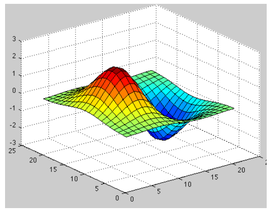


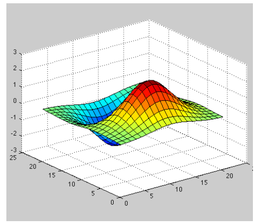


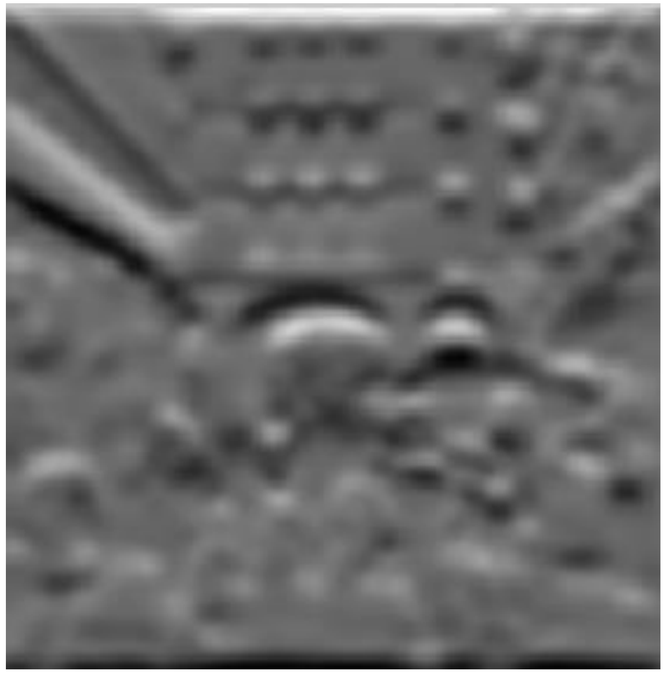
Laplacian Pyr

And many more: QMF, steerable, …Convnets!

Orientations

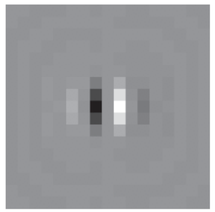


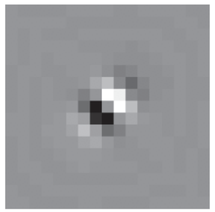


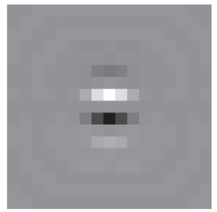


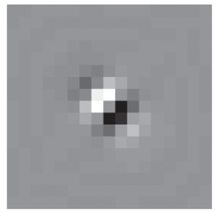
Orientations

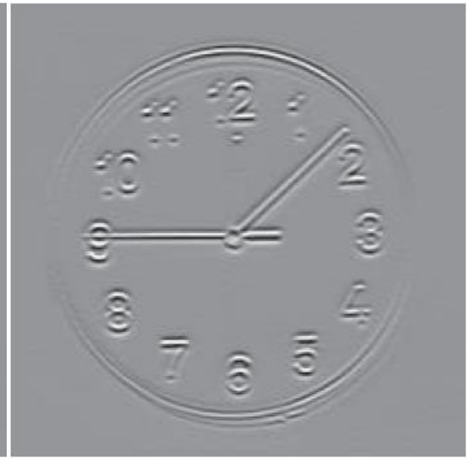
Steerable Pyramid





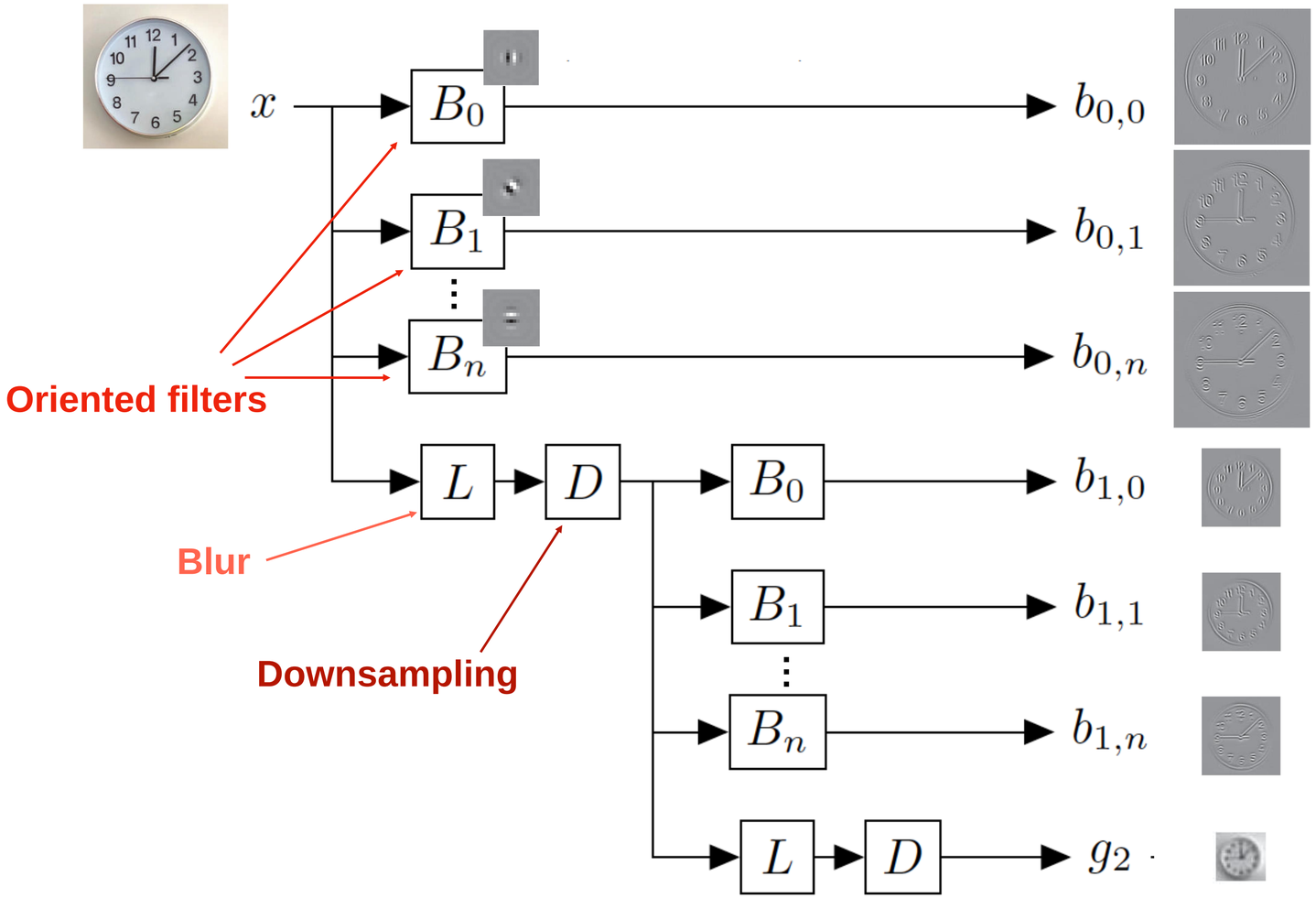


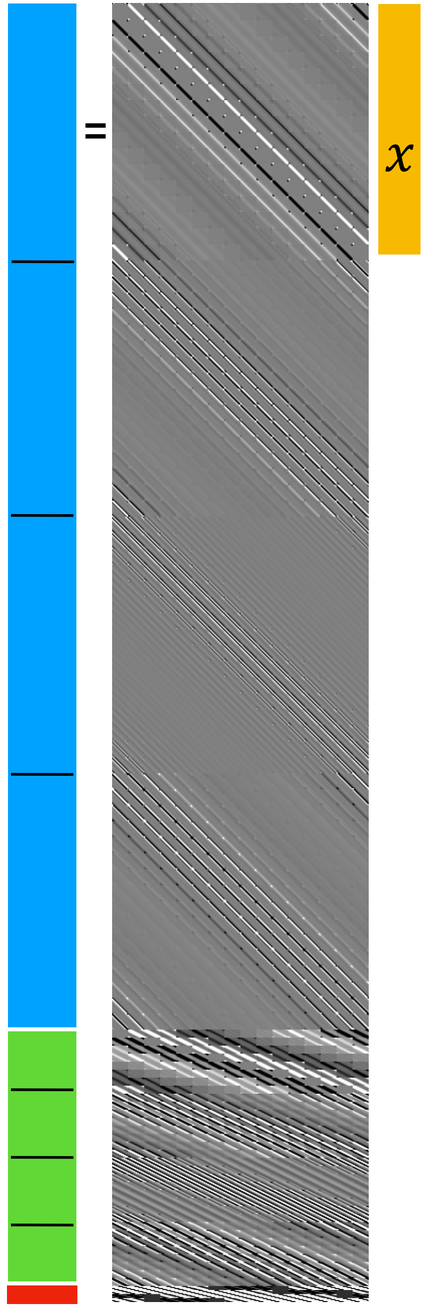




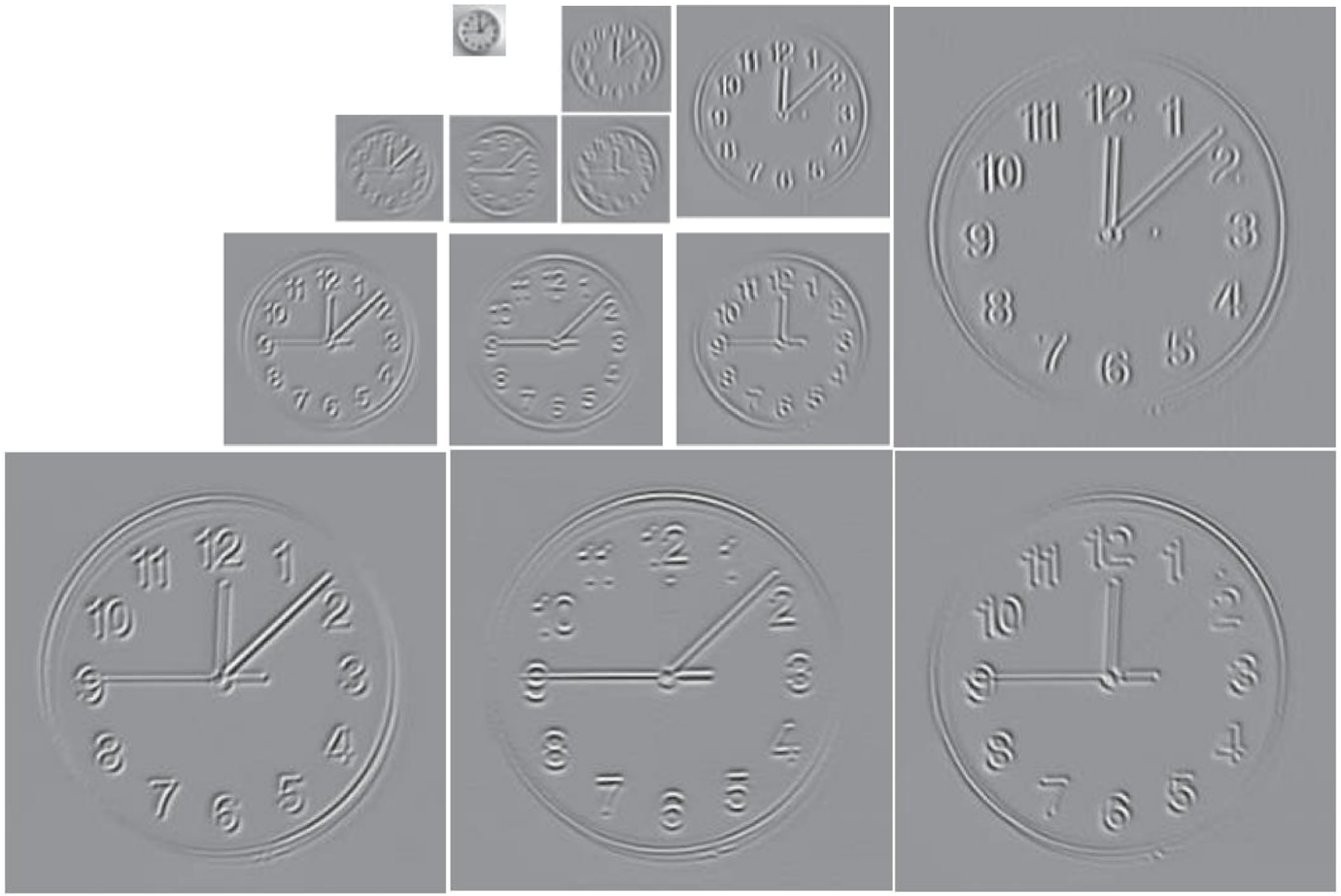
Low pass residual

Steerable Pyramid



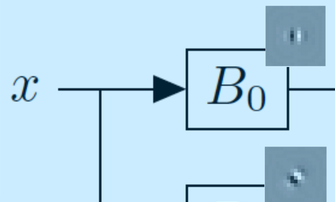


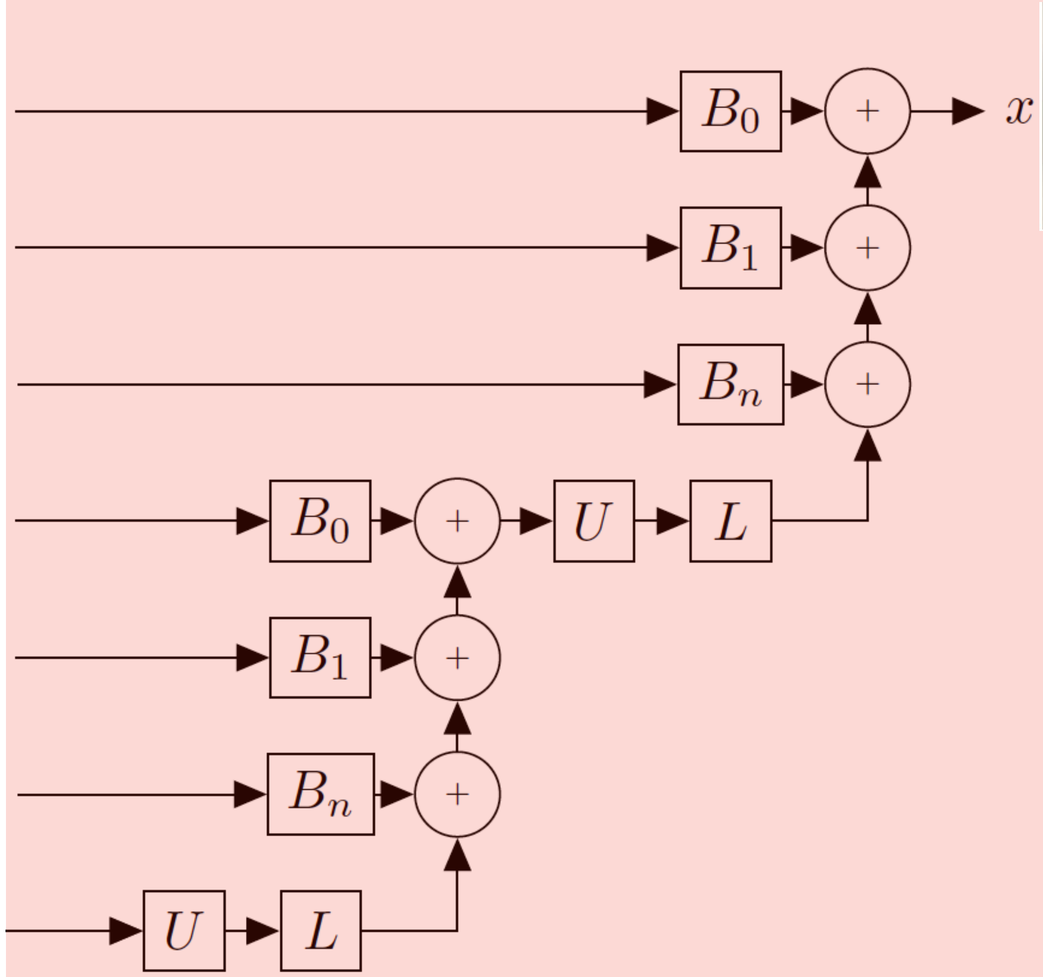
Steerable Pyramid



3 scales, 4 orientations

Steerable Pyramid





Analysis/Encoder

Synthesis/Decoder

What about **learning** the filters?

Wavelet-like receptive fields emerge from a network that learns sparse codes for natural images.

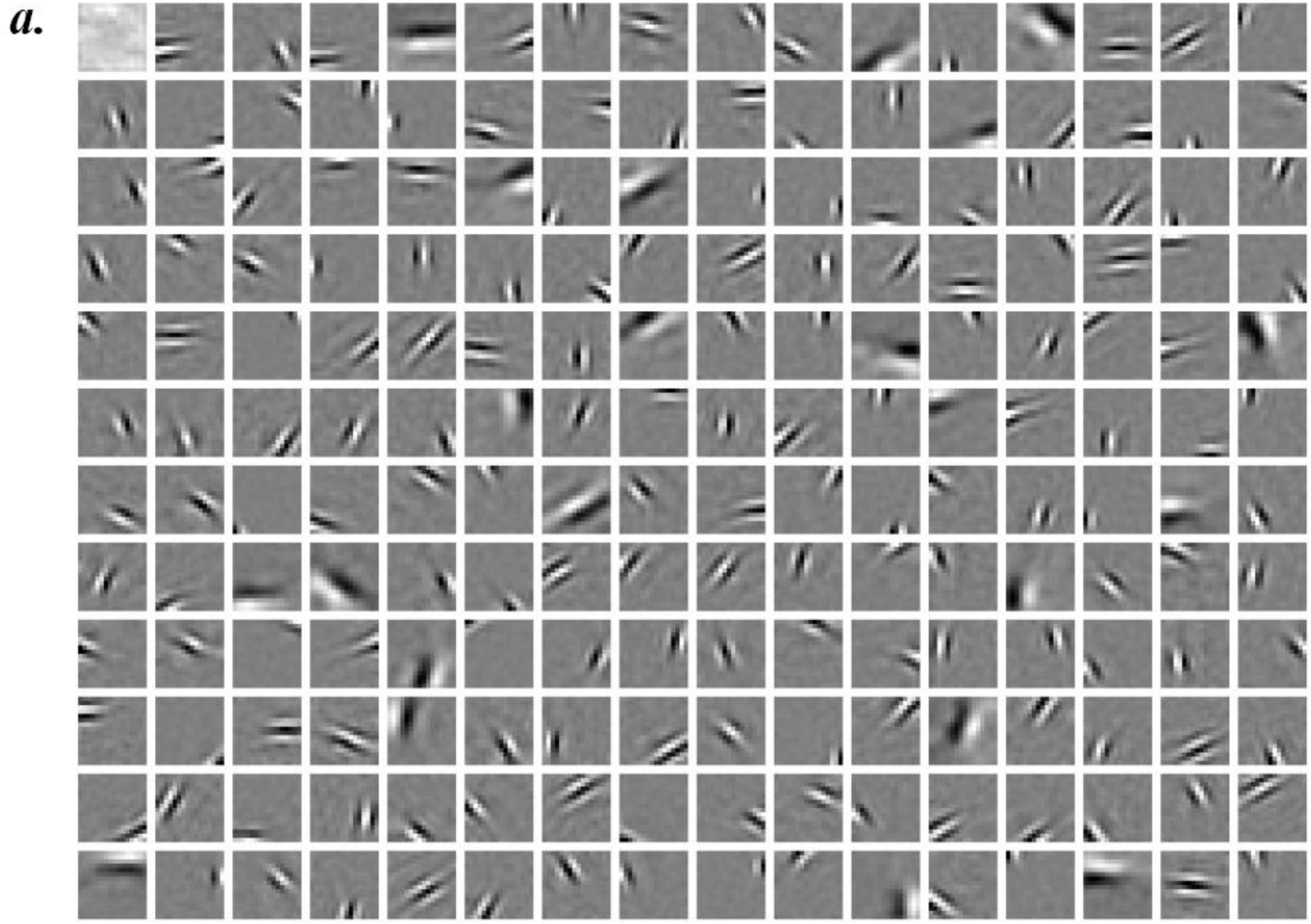
Bruno A. Olshausen[1] and David J. Field

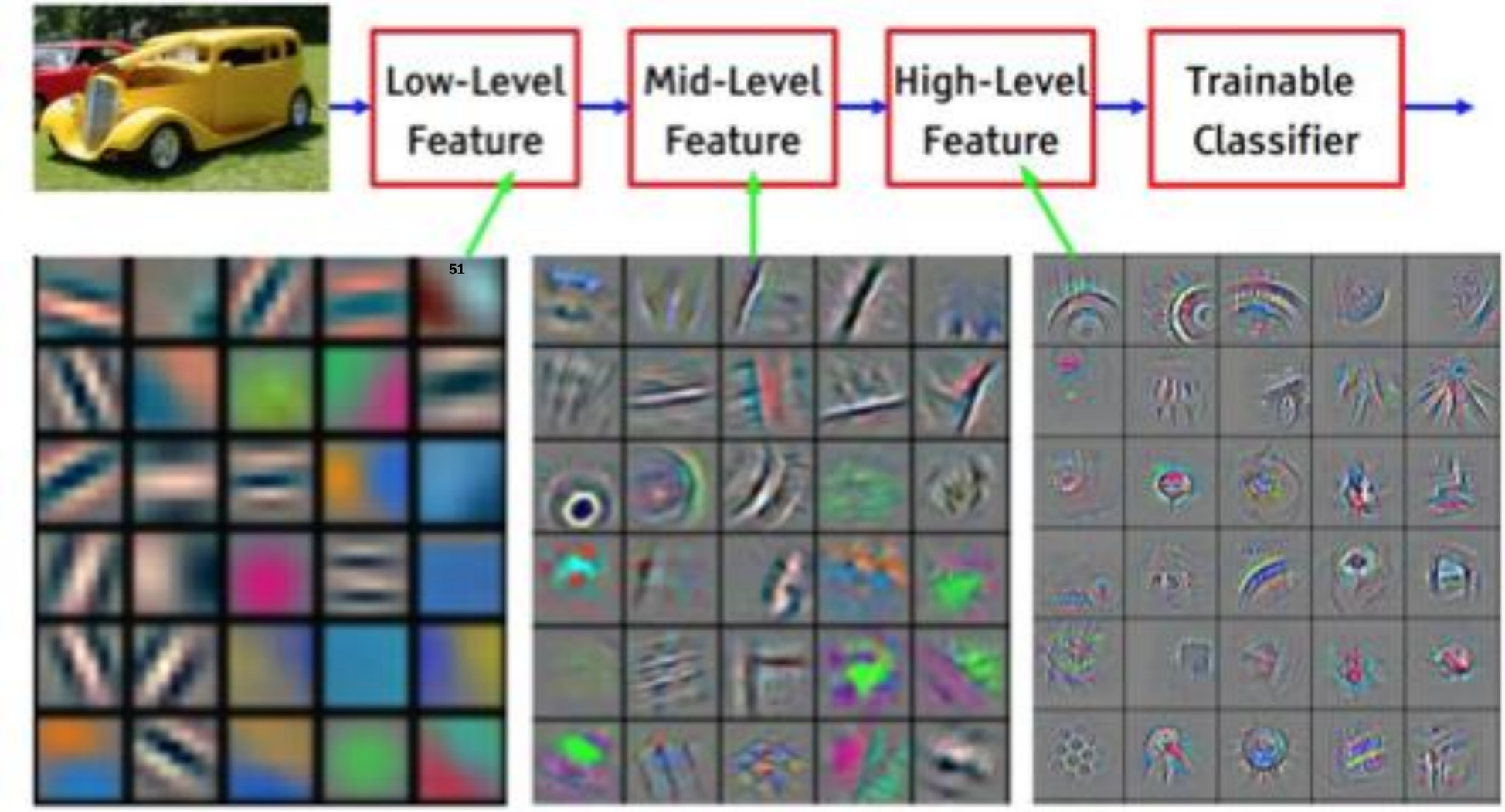
$$E = -[\text{preserve information}] - \lambda[\text{sparseness of } a_i]\,, \tag{2}$$

where λ is a positive constant that determines the importance of the second term relative to the first. The first term measures how well the code describes the image, and we choose this to be the mean square of the error between the actual image and the reconstructed image:

$$[\text{preserve information}] = -\sum_{x,y}\left[I(x,y) - \sum_i a_i\,\phi_i(x,y)\right]^2 . \tag{3}$$

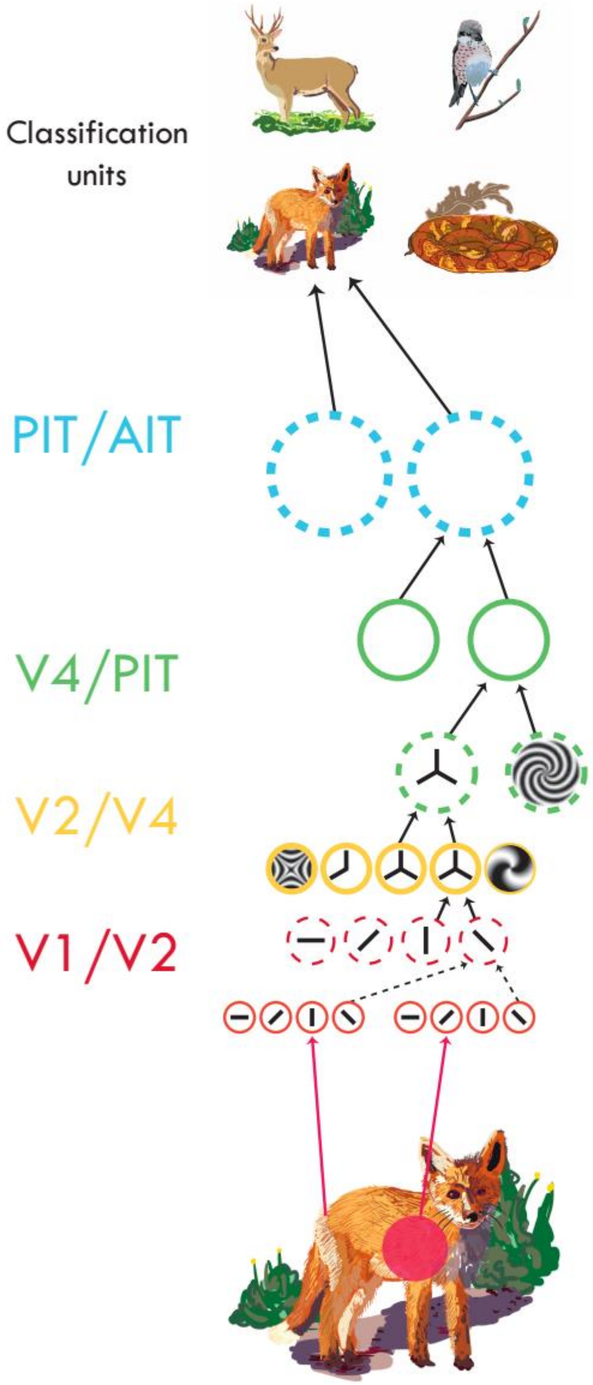
Learned filters





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

[slide courtesy Yann LeCun]



Serre, 2014

Image classification

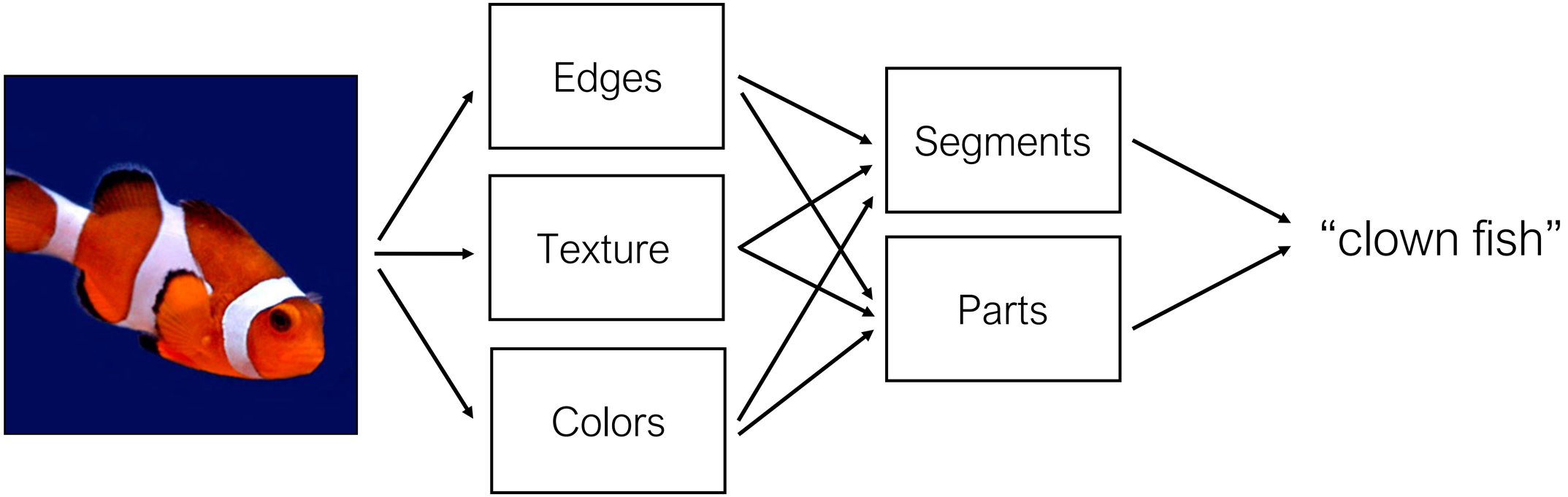
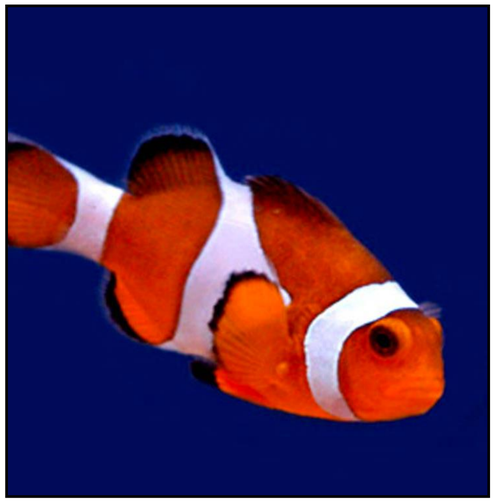


Image classification



Edges

Texture

Colors

Segments

Parts

Learned

“clown fish”

Classifier

Image classification

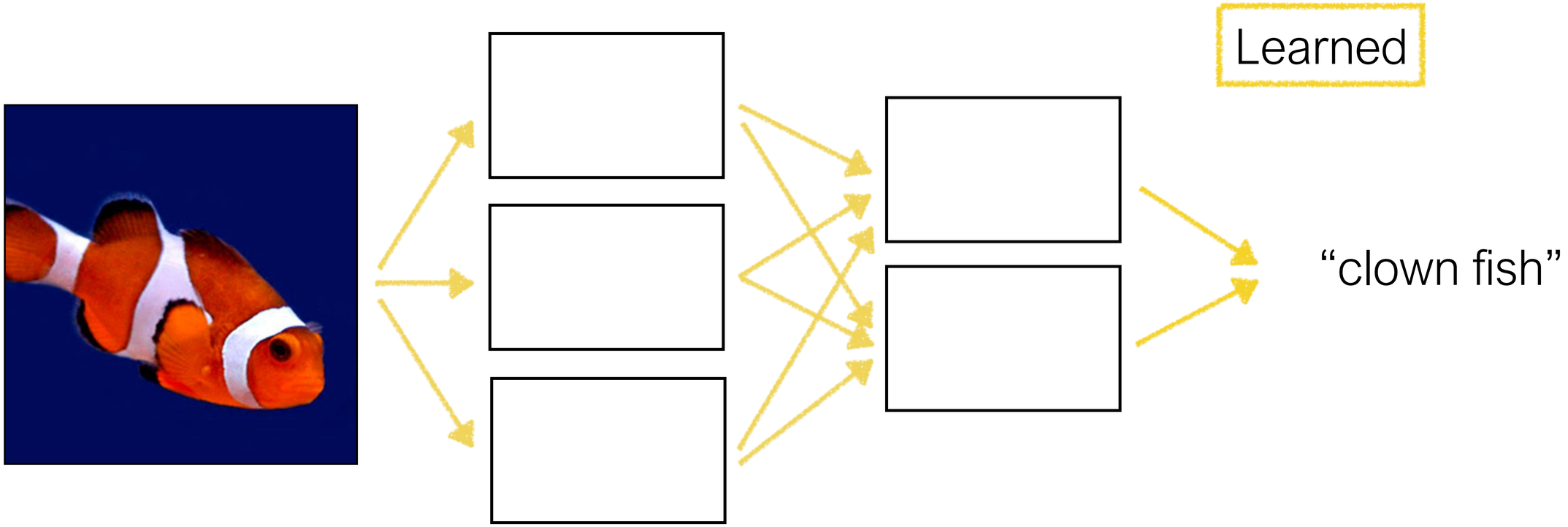
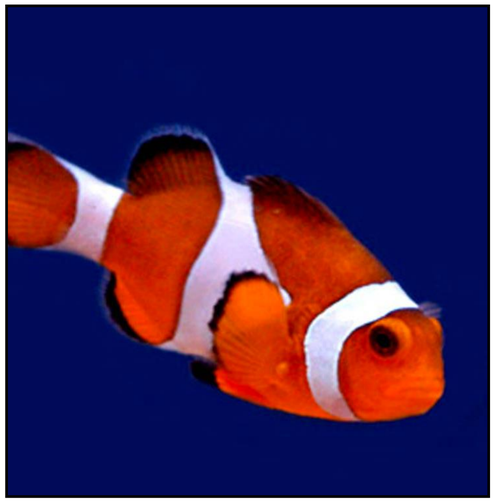
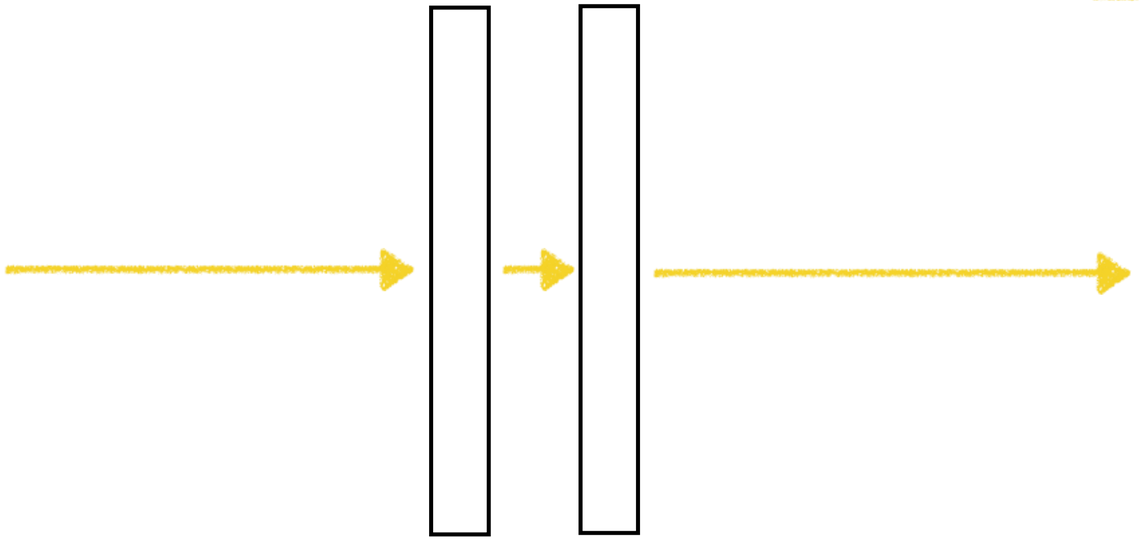


Image classification



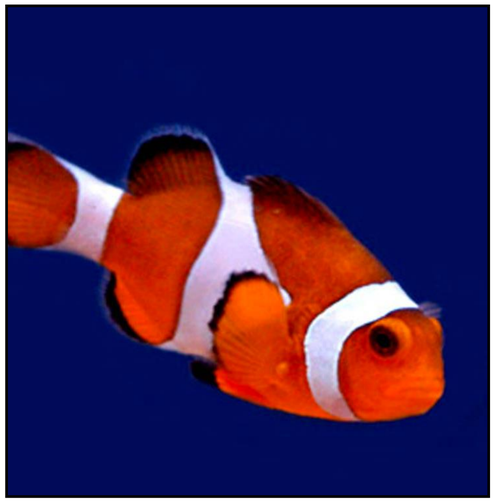
Learned



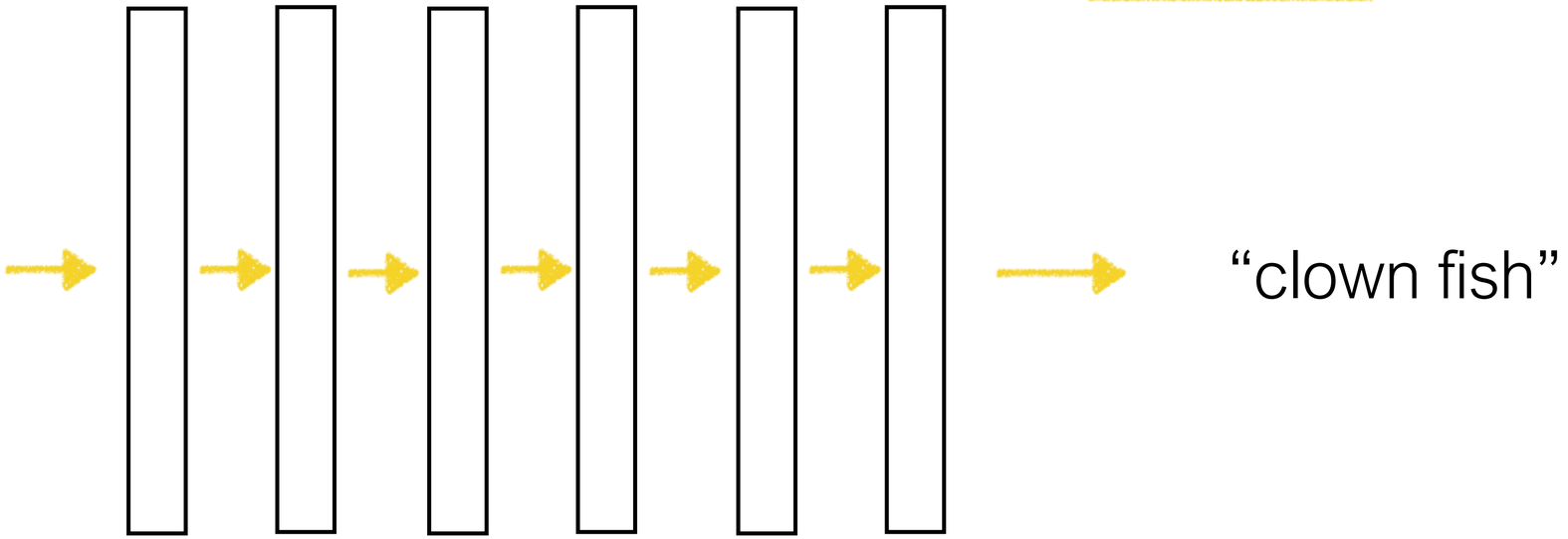
“clown fish”

Neural net

Image classification



Learned



“clown fish”

Deep neural net

Deep learning

$\mathbf{y}^{(i)}$

"clown fish"

Learned

Training data

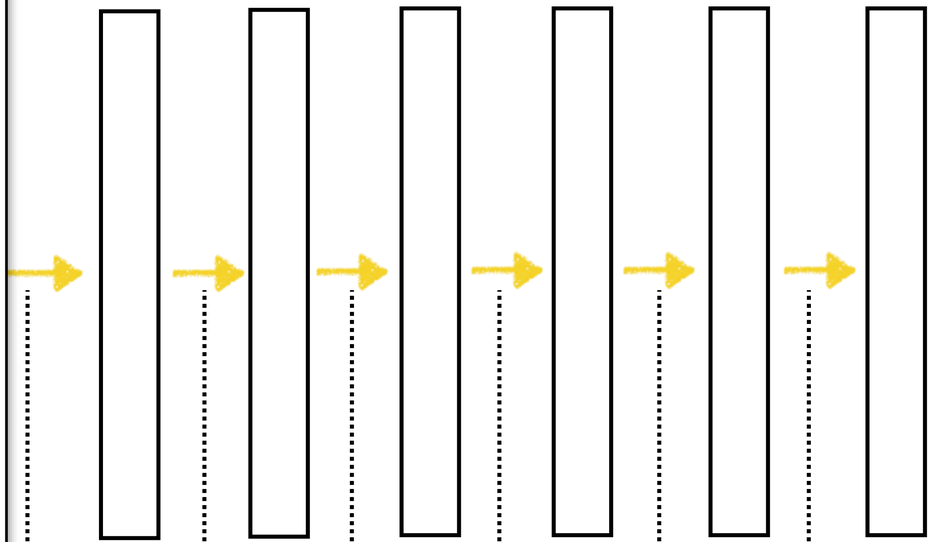
$\mathbf{x}$ y

"Fish"

"Grizzly"

"Chameleon"

⋮



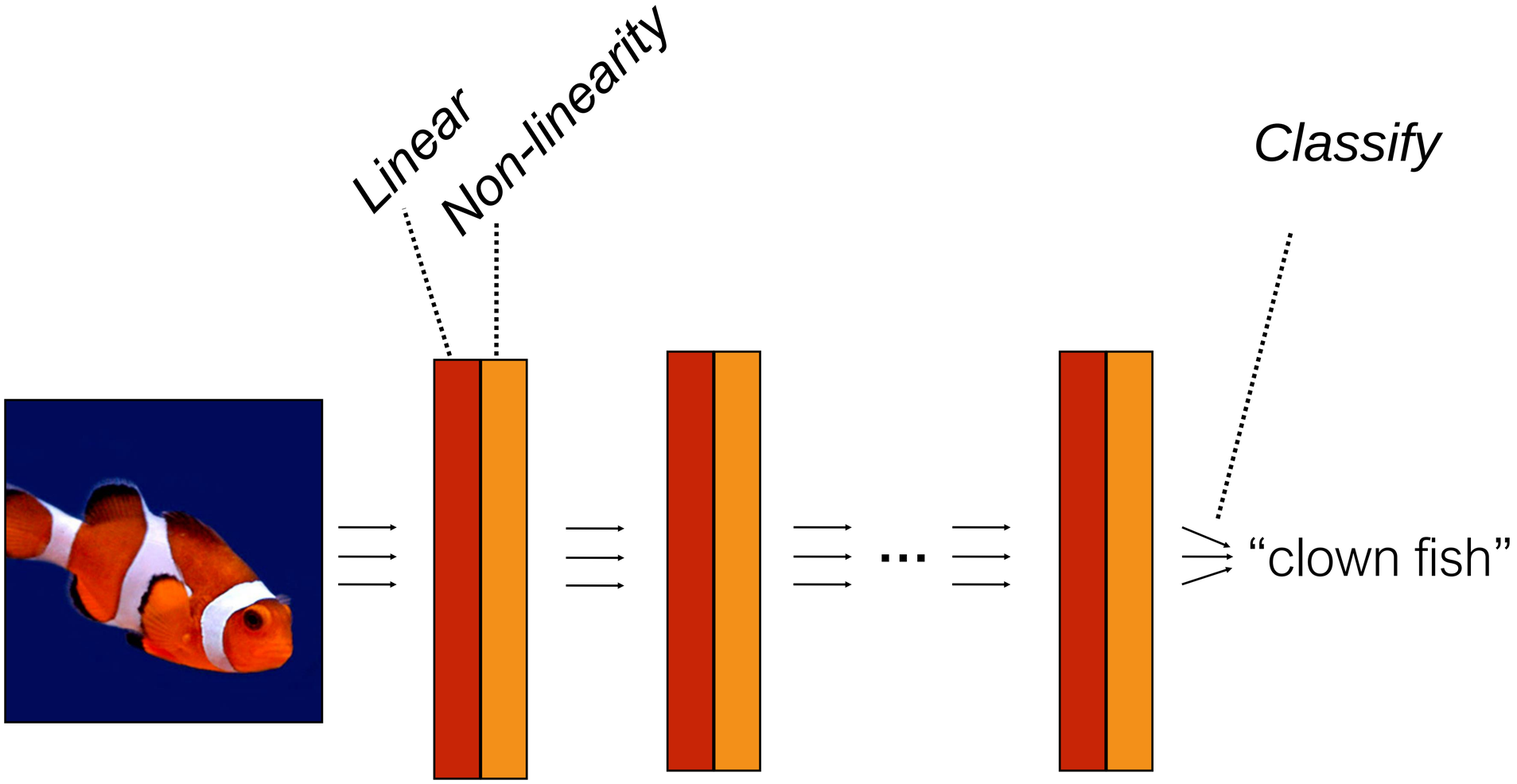
θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

Loss

$\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$

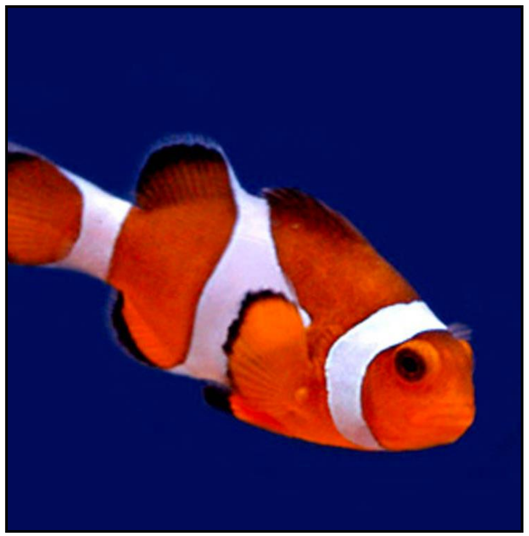
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Deep nets



$$f(\mathbf{x}) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}))))$$

Image classification is boring!



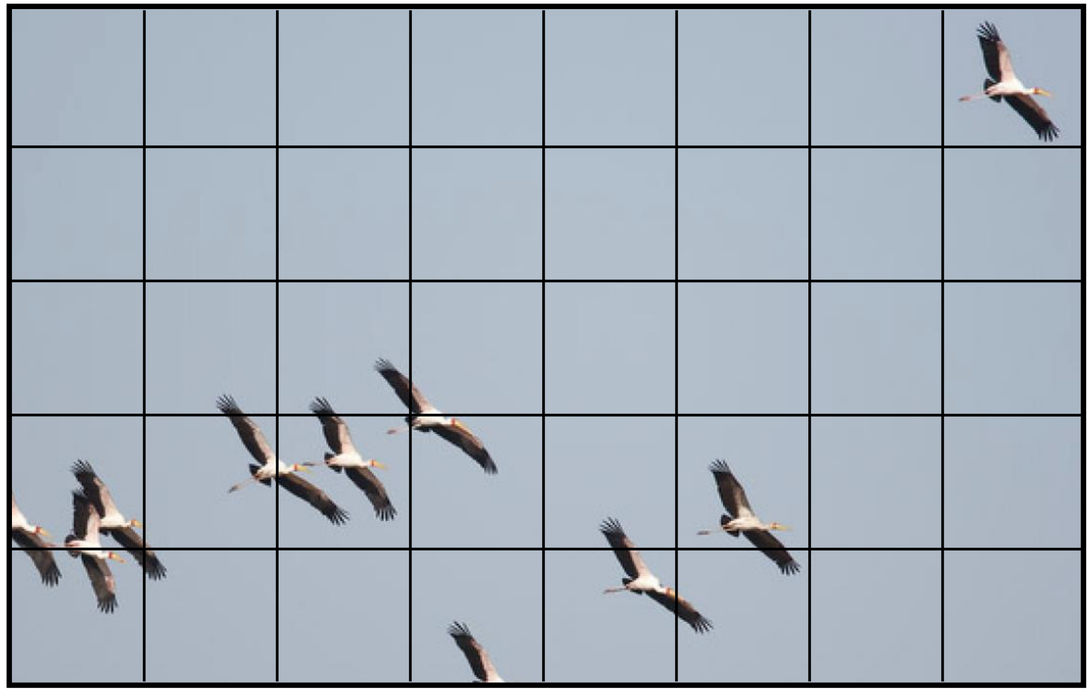
Classifier

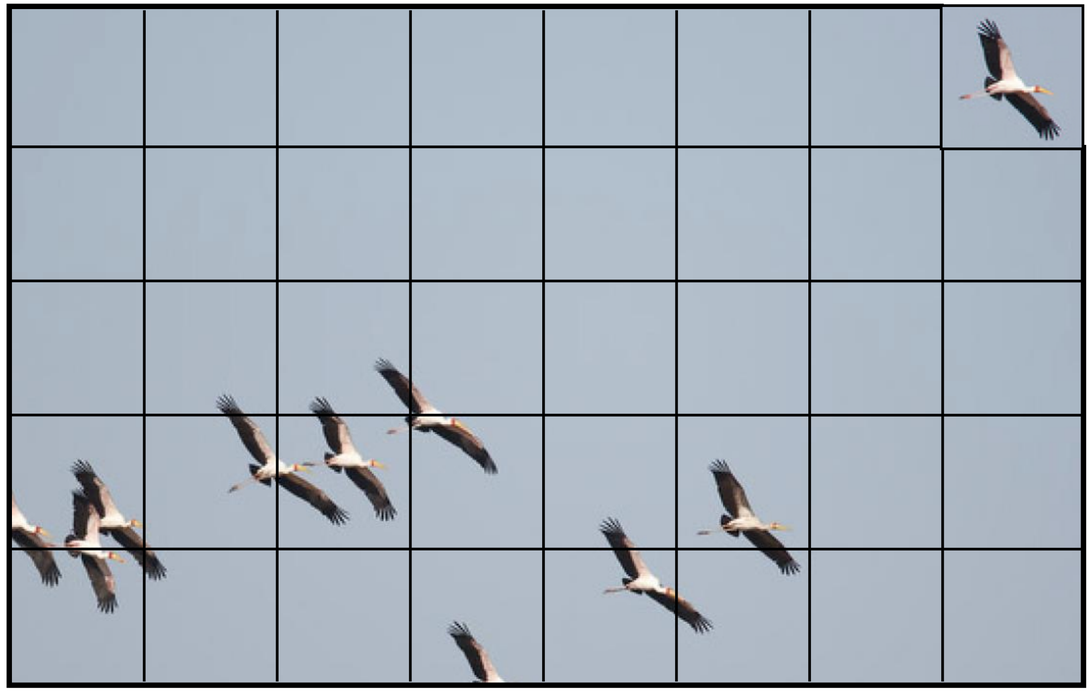
“Fish”

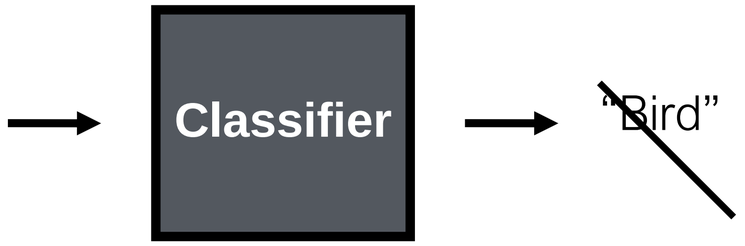
image **x**

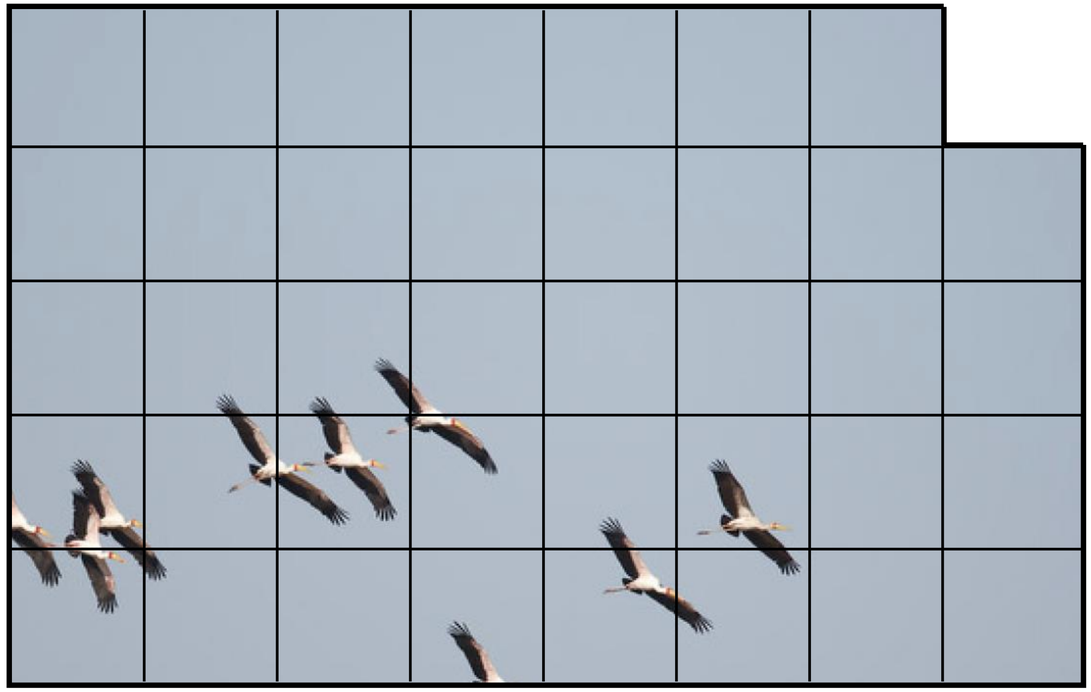
label y

Photo credit: Fredo Durand



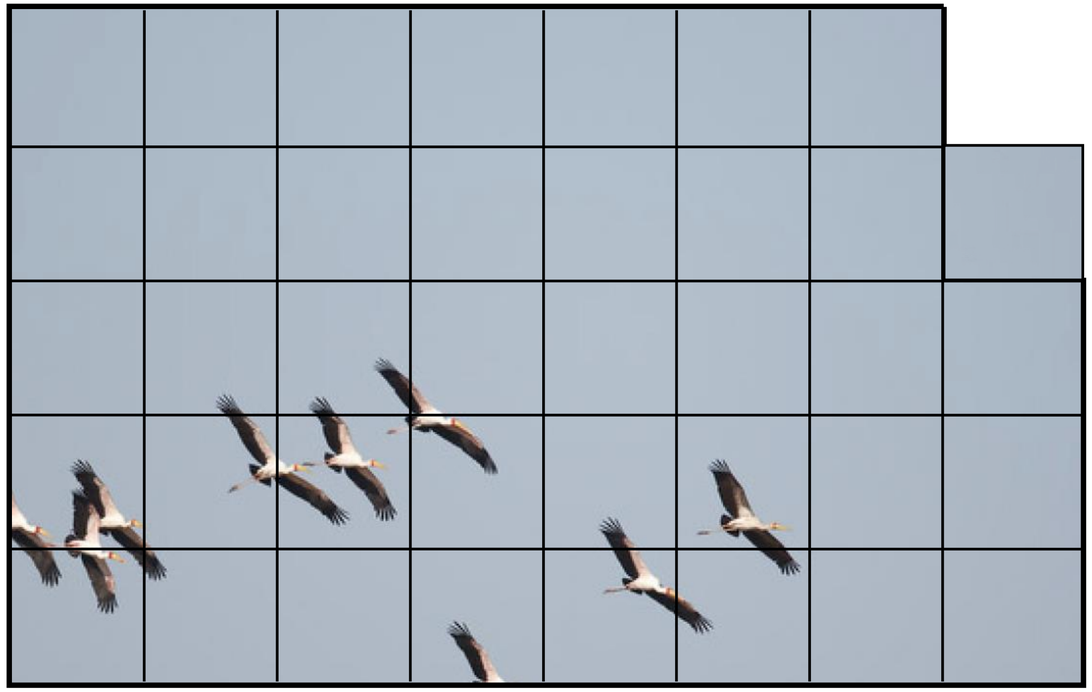



Classifier
“Bird”

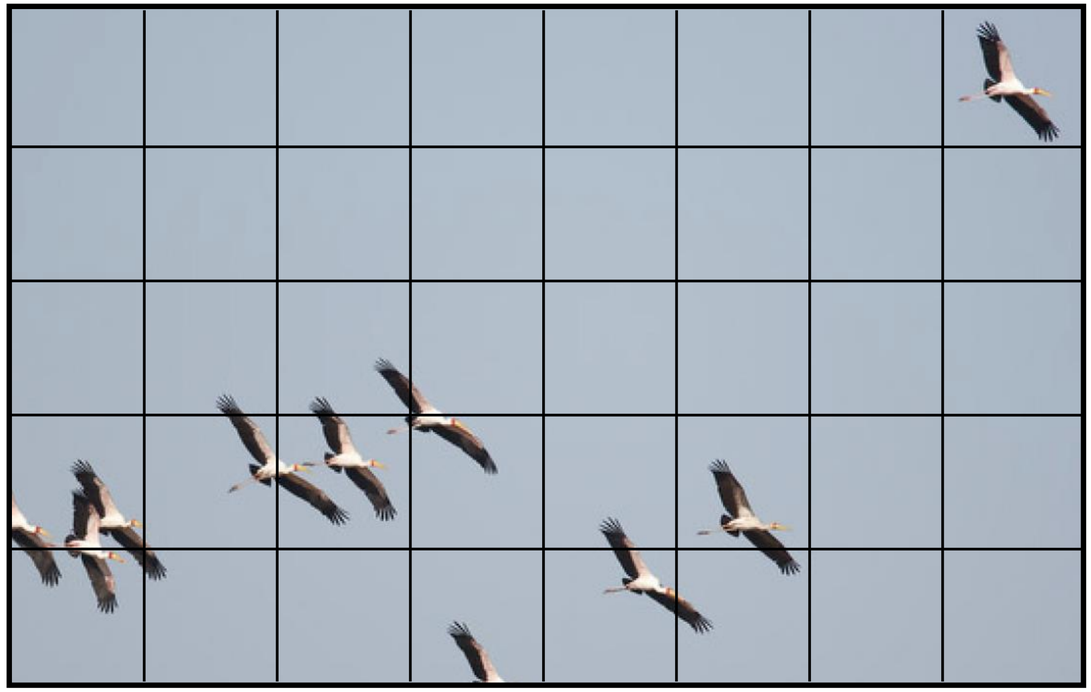


Classifier

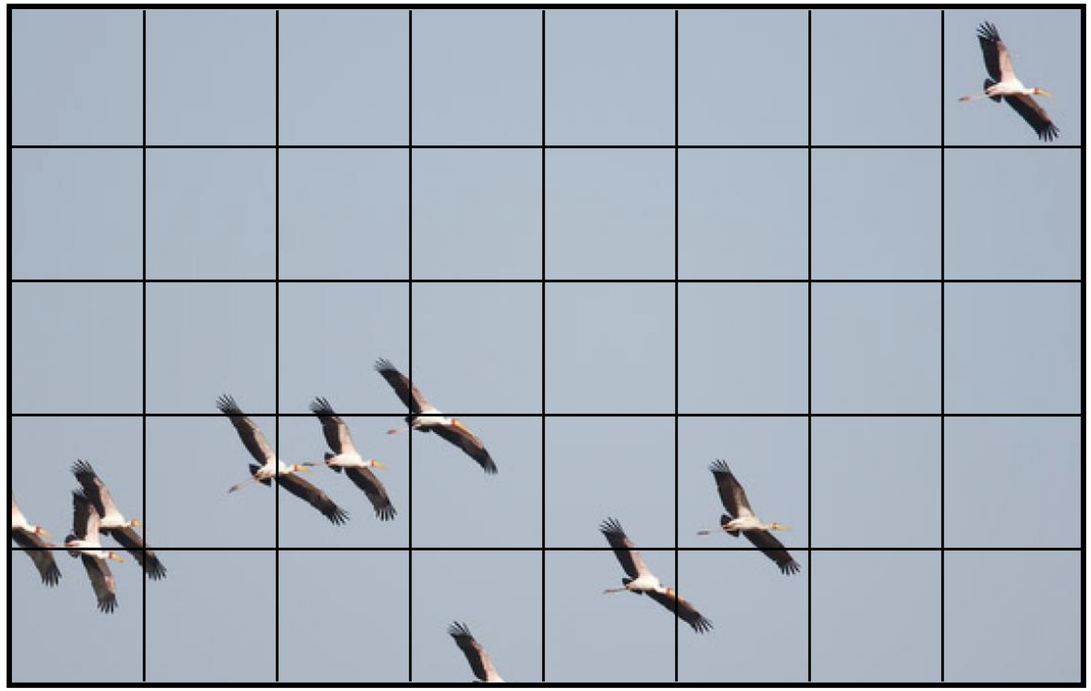
“Bird”



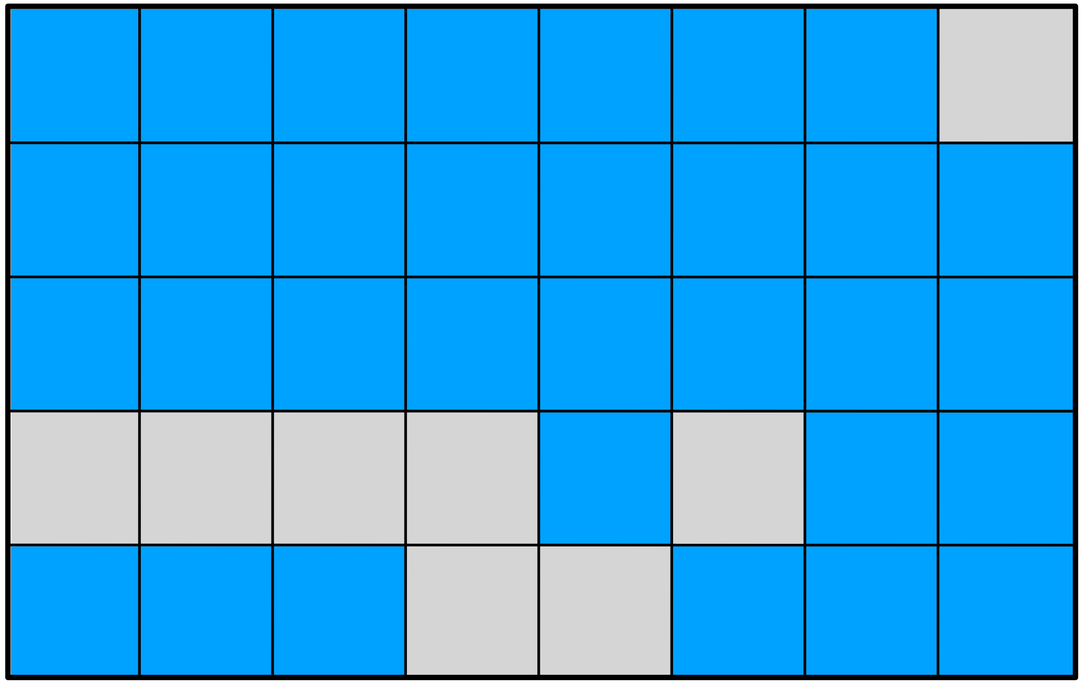
							Bird



Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



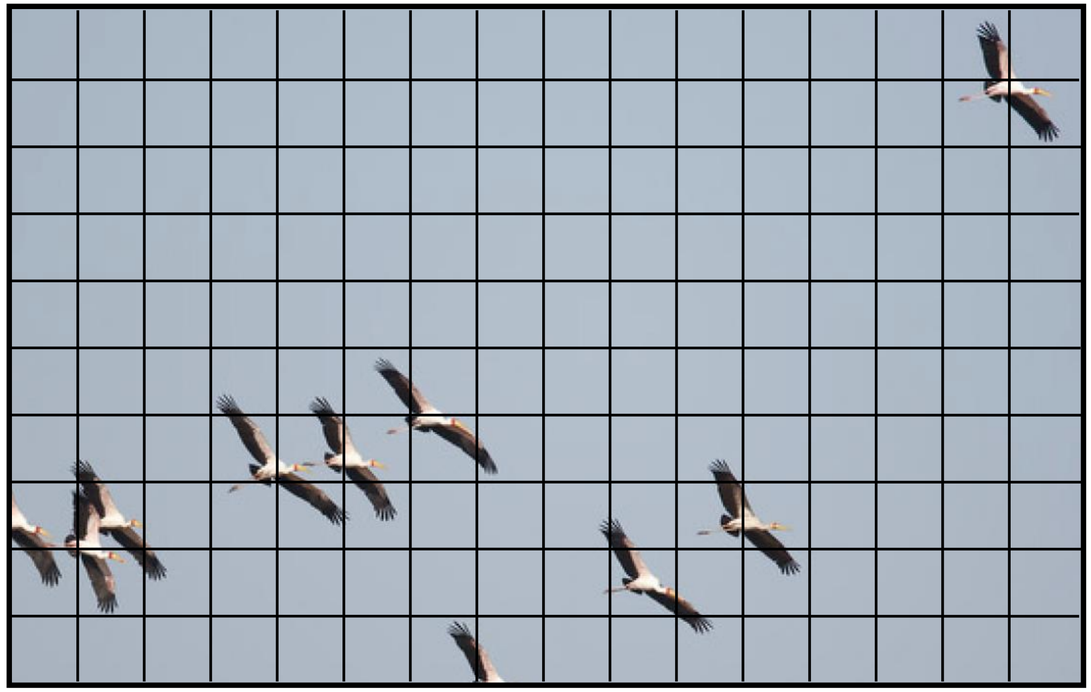
f



Problem:

What happens to objects that are bigger?

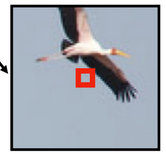
What if an object crosses multiple cells?

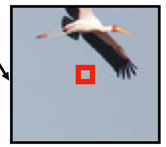


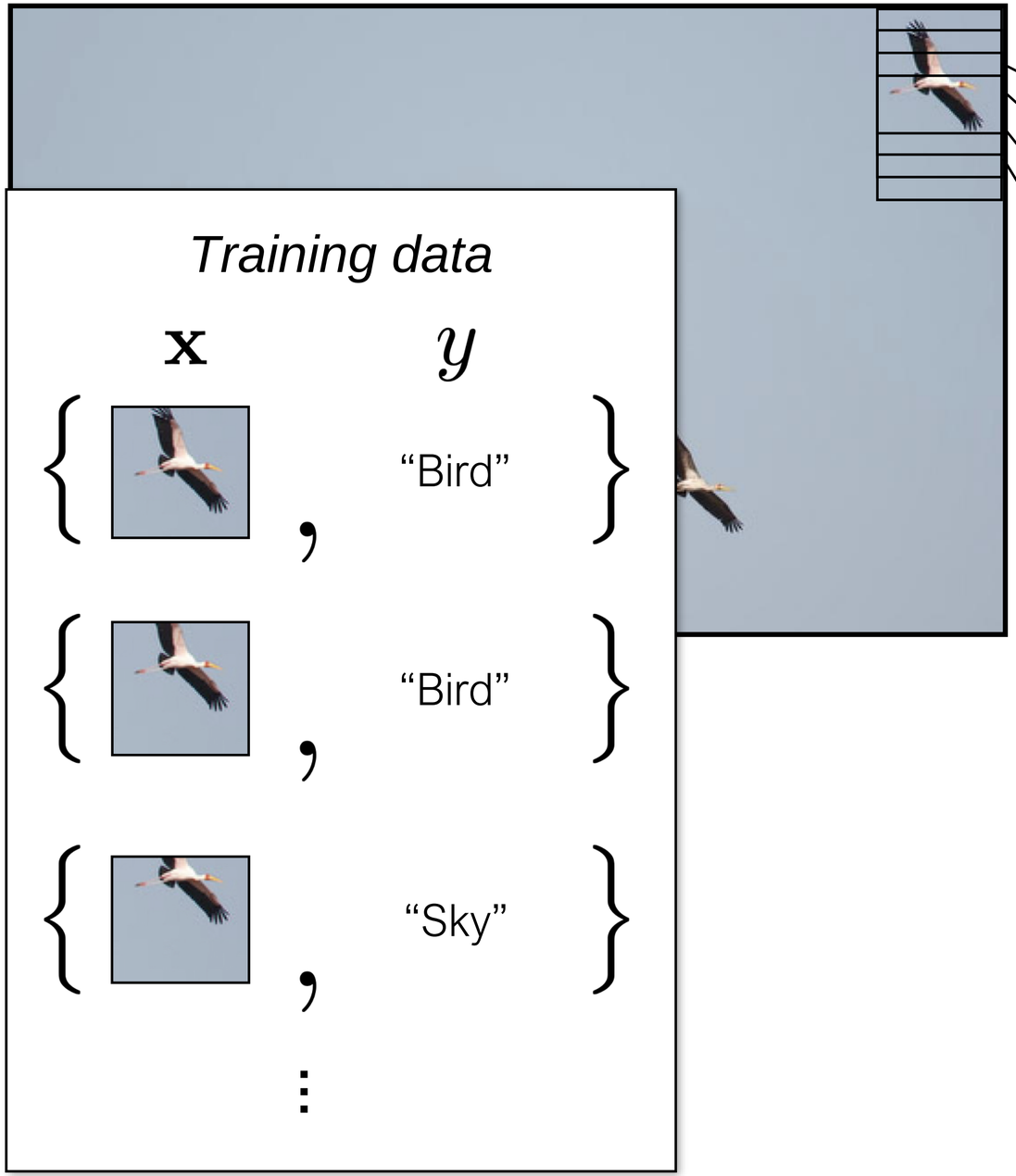
“Cell”-based approach is limited.

What can we do instead?

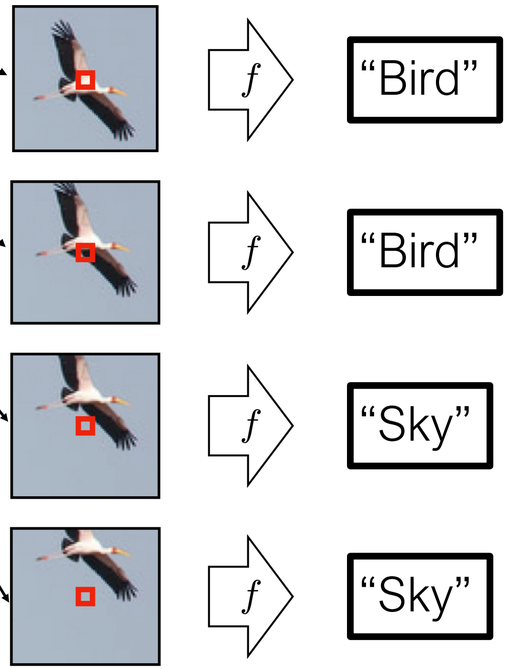
What's the object class of the center pixel?



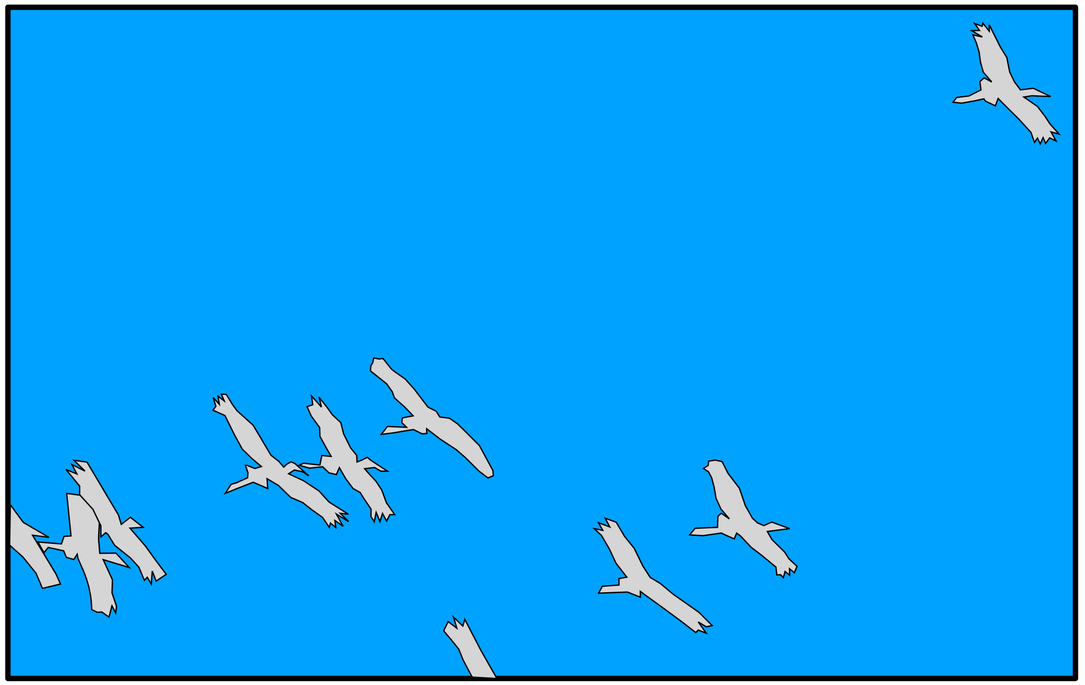




What’s the object class of the center pixel?



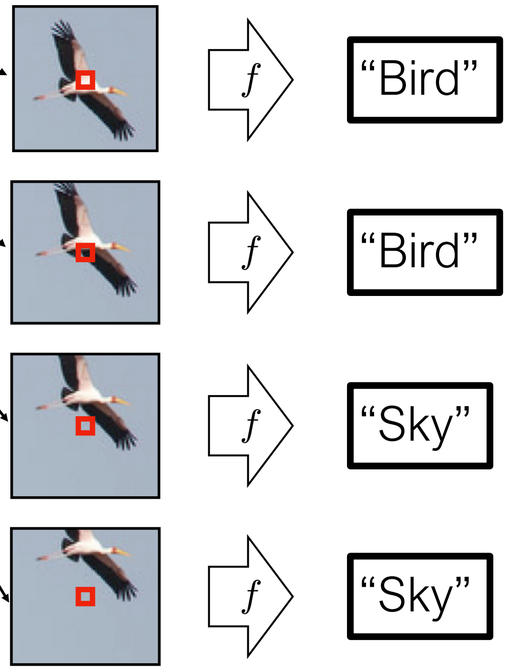
f

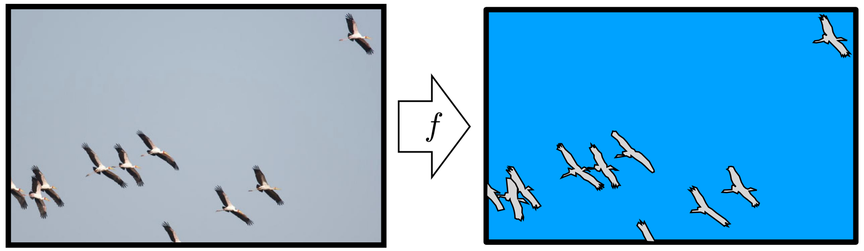


(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



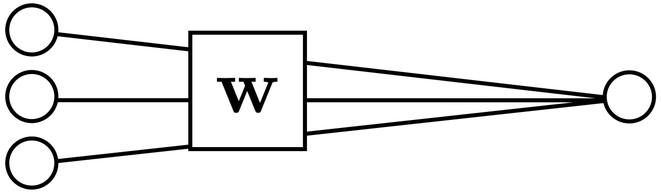


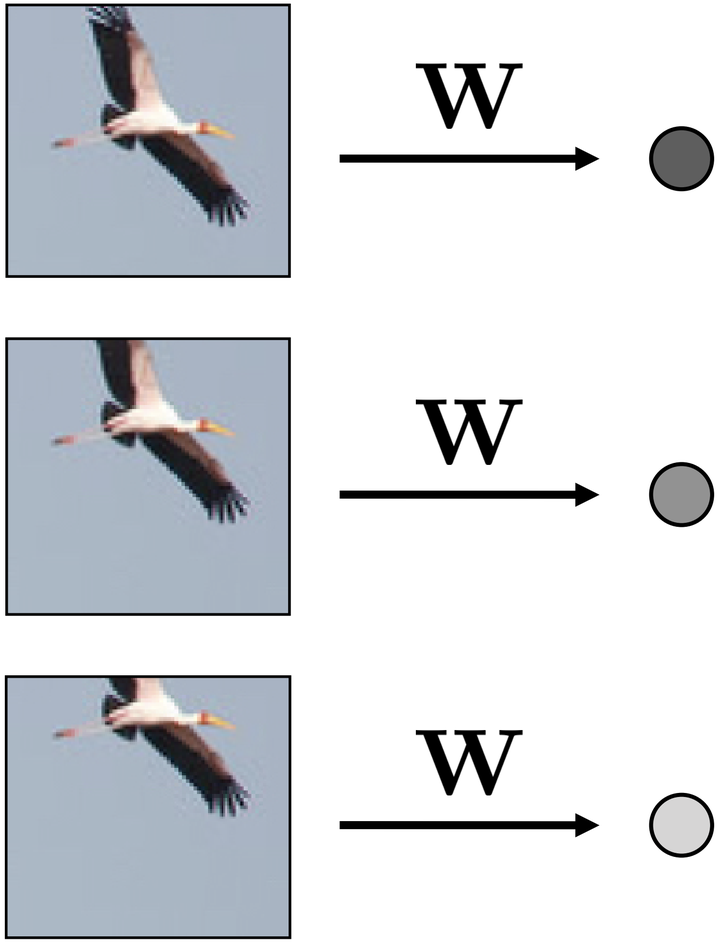
Translation invariance: process each patch in the same way.

An *equivariant* mapping:

$$f(\texttt{translate}(x)) = \texttt{translate}(f(x))$$

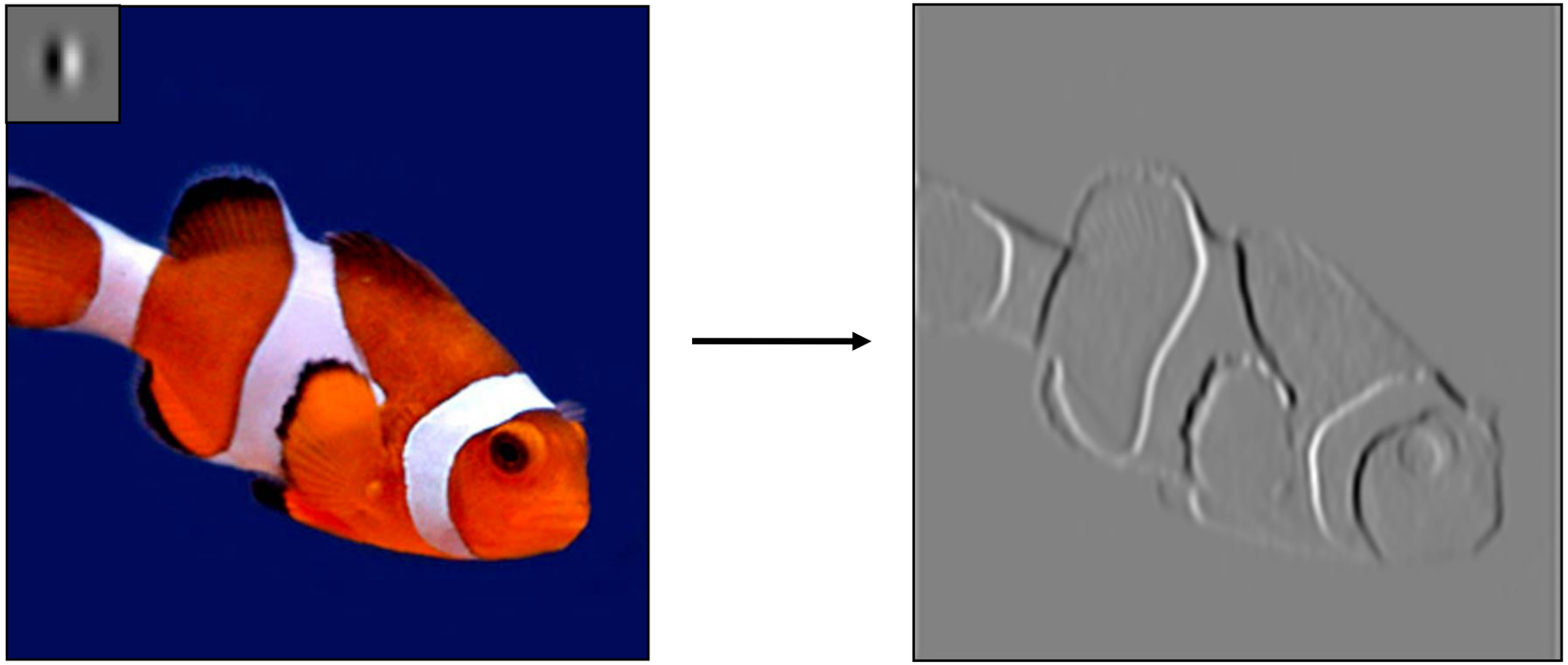
W computes a weighted sum of all pixels in the patch



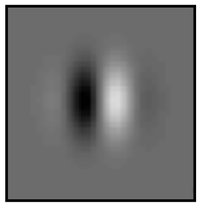


W is a convolutional kernel applied to the full image!

Convolution

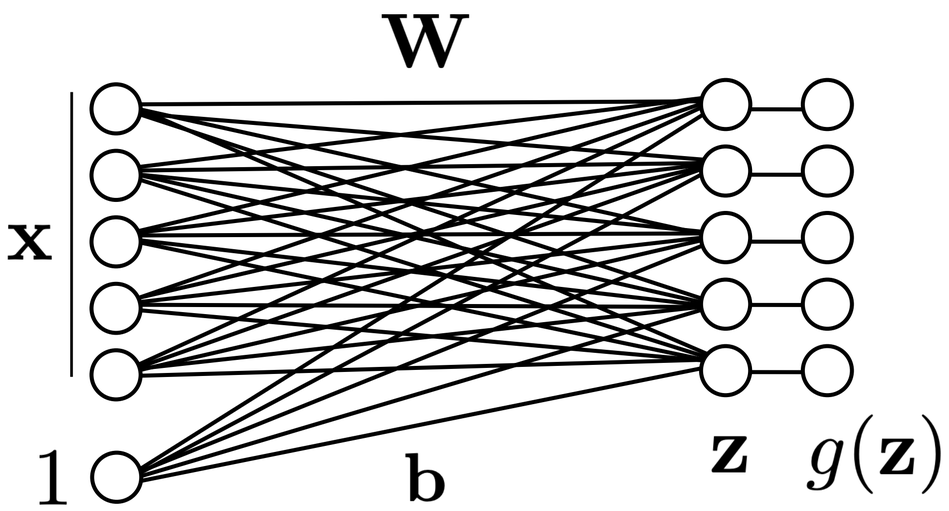


filter

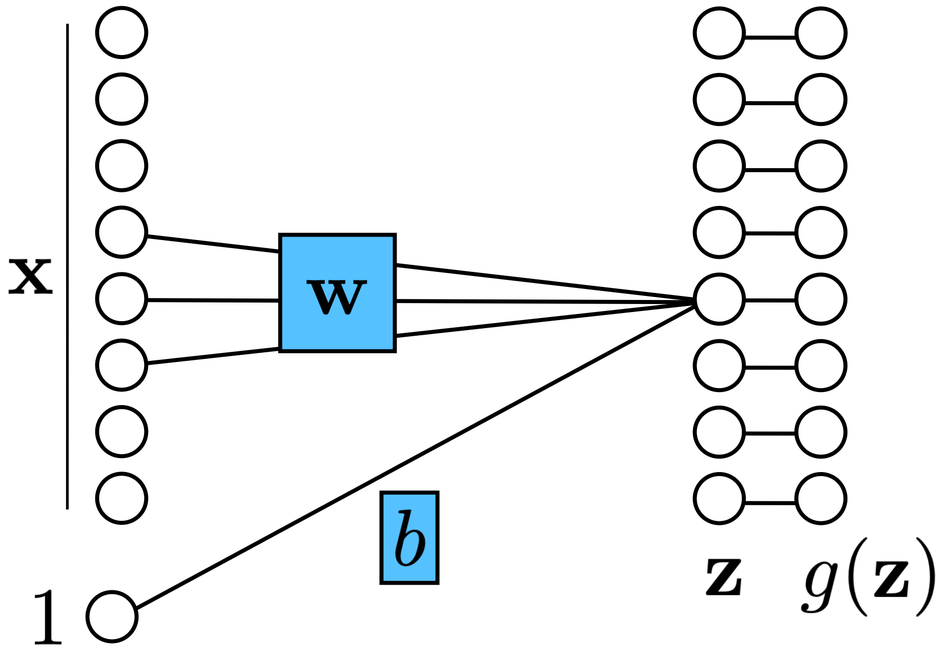


Fully-connected network

Fully-connected (fc) layer



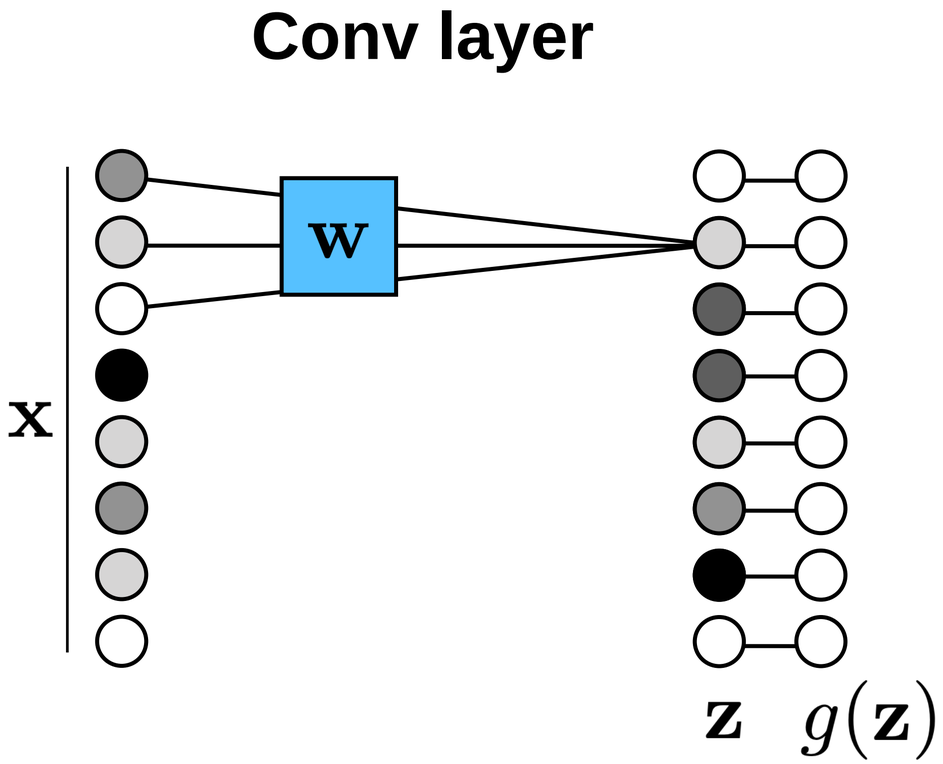
Locally connected network



Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Convolutional neural network



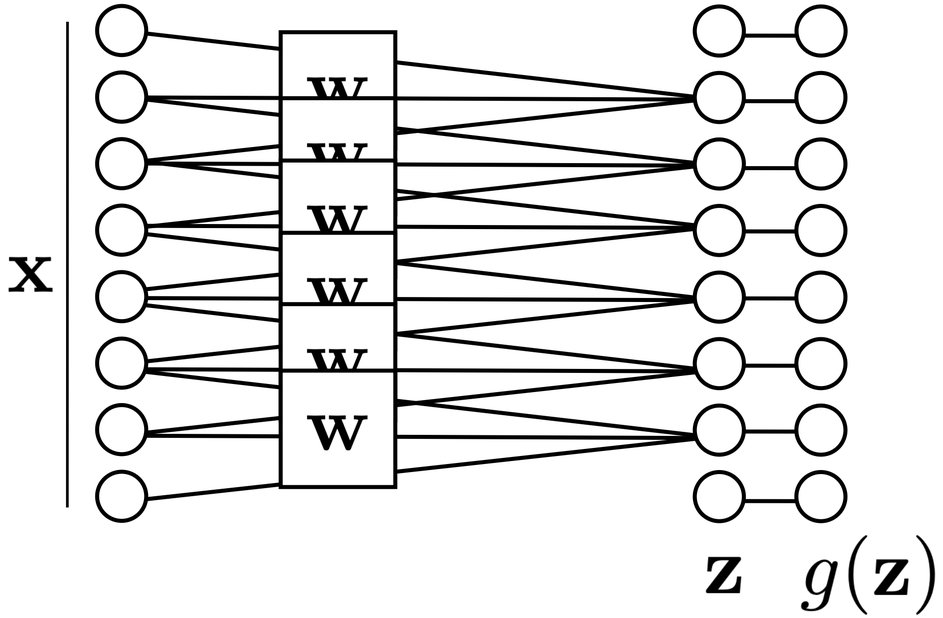
$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Weight sharing

Conv layer



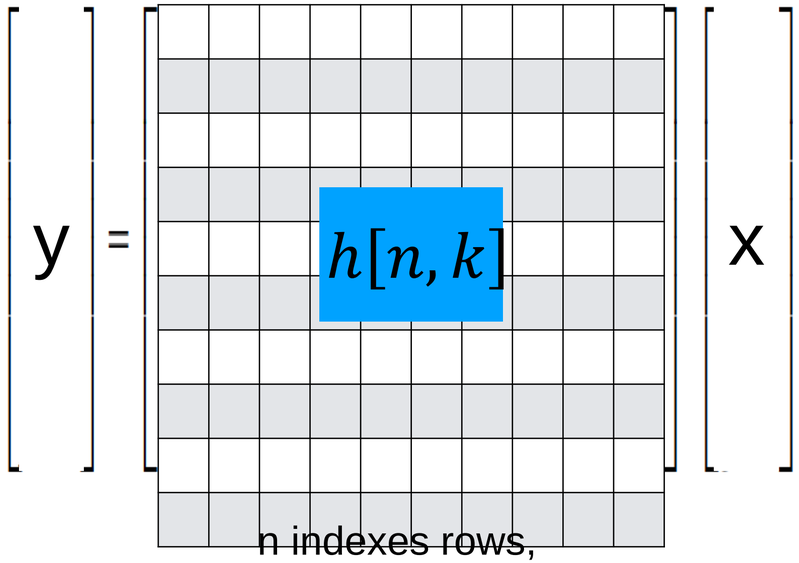
$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

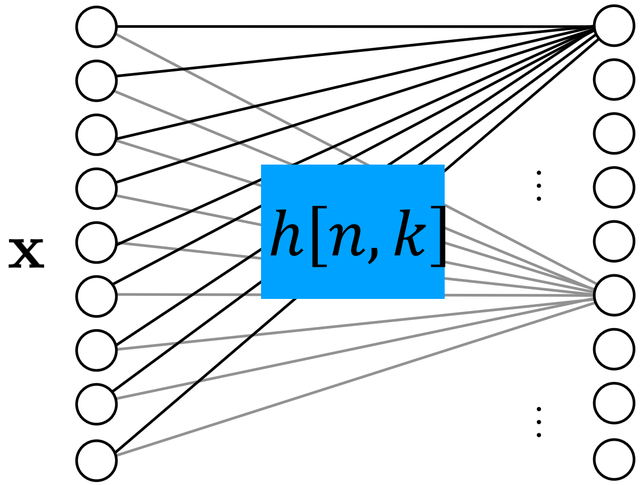
Linear system: $\mathbf{y} = f(\mathbf{x})$

A linear function f can be written as a matrix multiplication:



n indexes rows,
k indexes columns

It can also be represented as a fully connected linear neural network

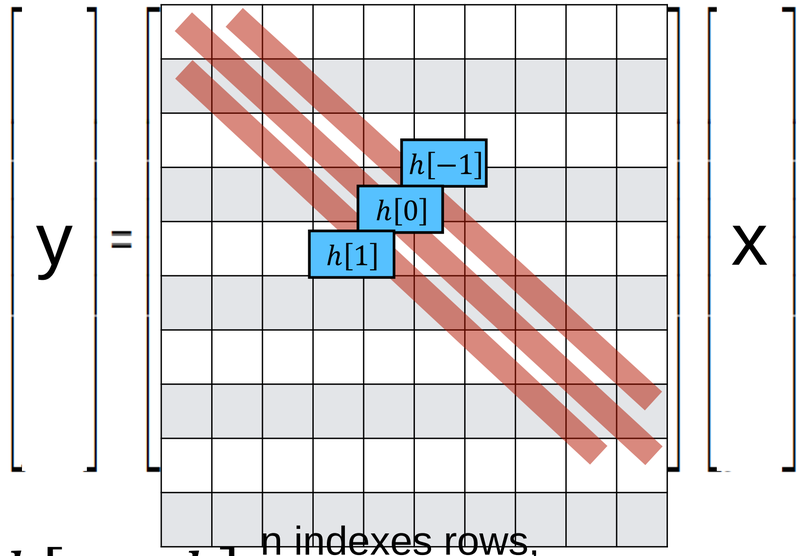


$$y[n] = \sum_{k=0}^{N-1} h[n,k]x[k]$$

$h[n,k]$ Is the strength of the connection between x[k] and y[n]

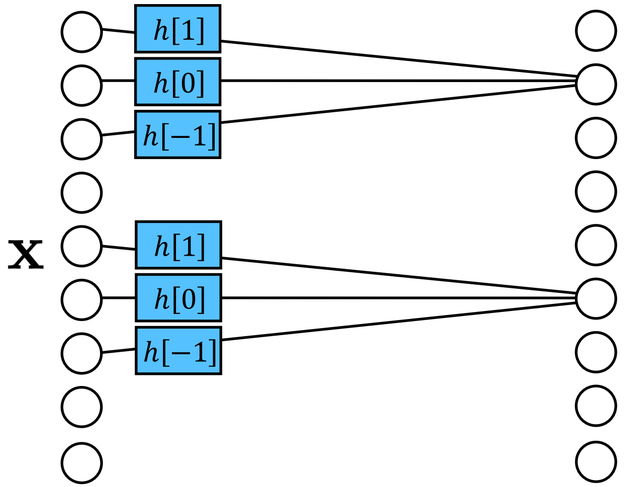
Convolution

A linear shift invariant (LSI) function f can be written as a matrix multiplication:



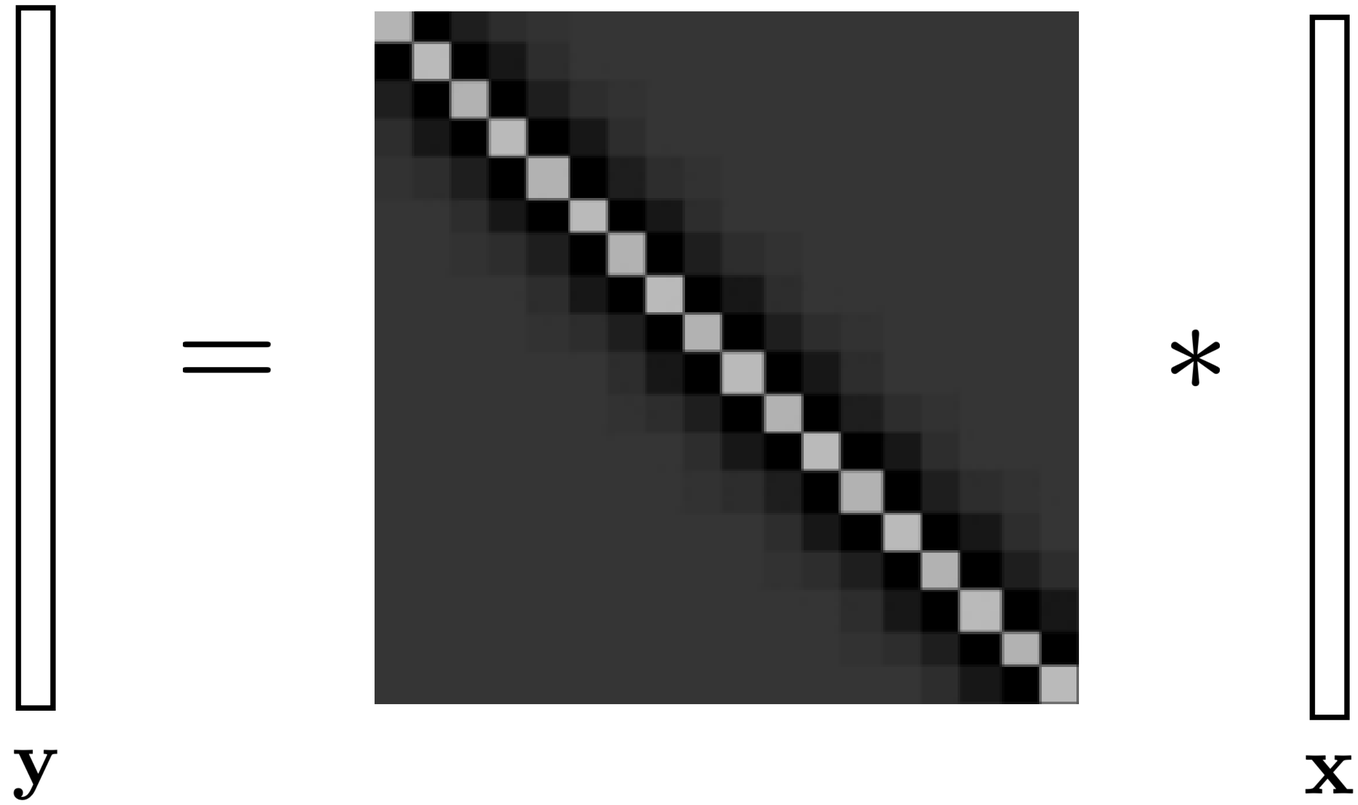
$h[n-k]$ n indexes rows, k indexes columns

It can also be represented as a convolutional layer of neural net:



$$y[n] = \sum_{k=-1}^{1} h[k]x[n-k]$$

$h[n-k]$ Is the strength of the connection between x[k] and y[n]



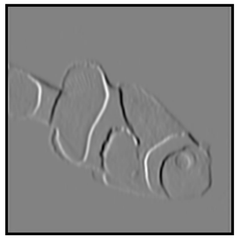
e.g., pixel image

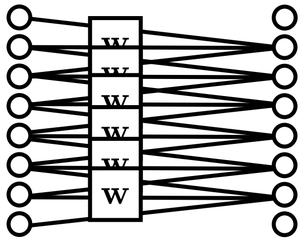
- Constrained linear layer
- Fewer parameters —> easier to learn, less overfitting

Five views on convolutional layers

1. Equivariant with translation $f(\texttt{translate}(x)) = \texttt{translate}(f(x))$

2. Patch processing

3. Image filter 

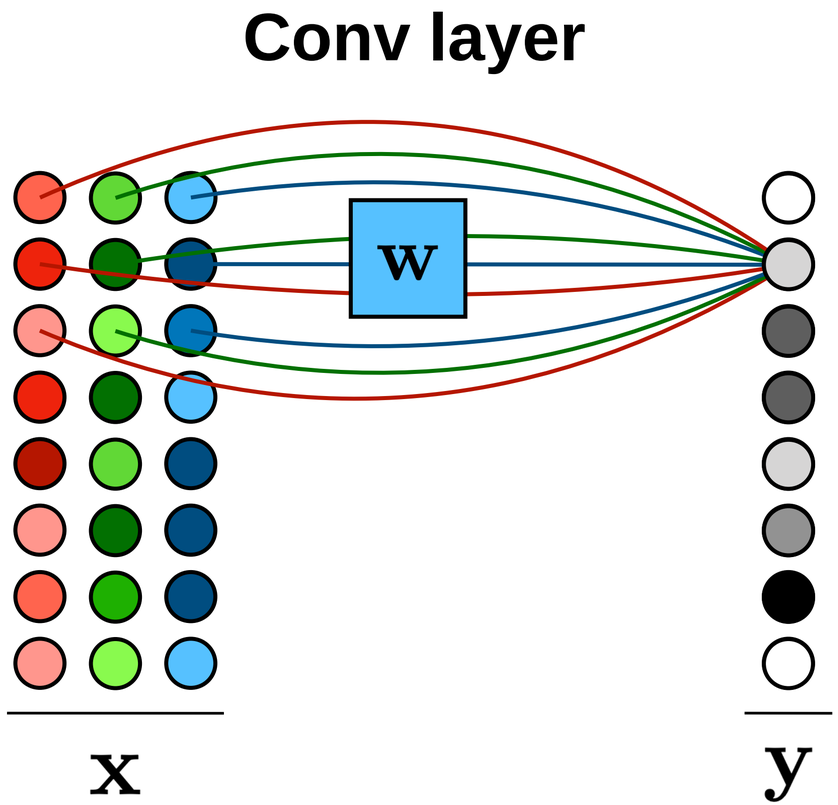
4. Parameter sharing 

5. A way to process variable-sized tensors

What if we have color?

(aka multiple input channels?)

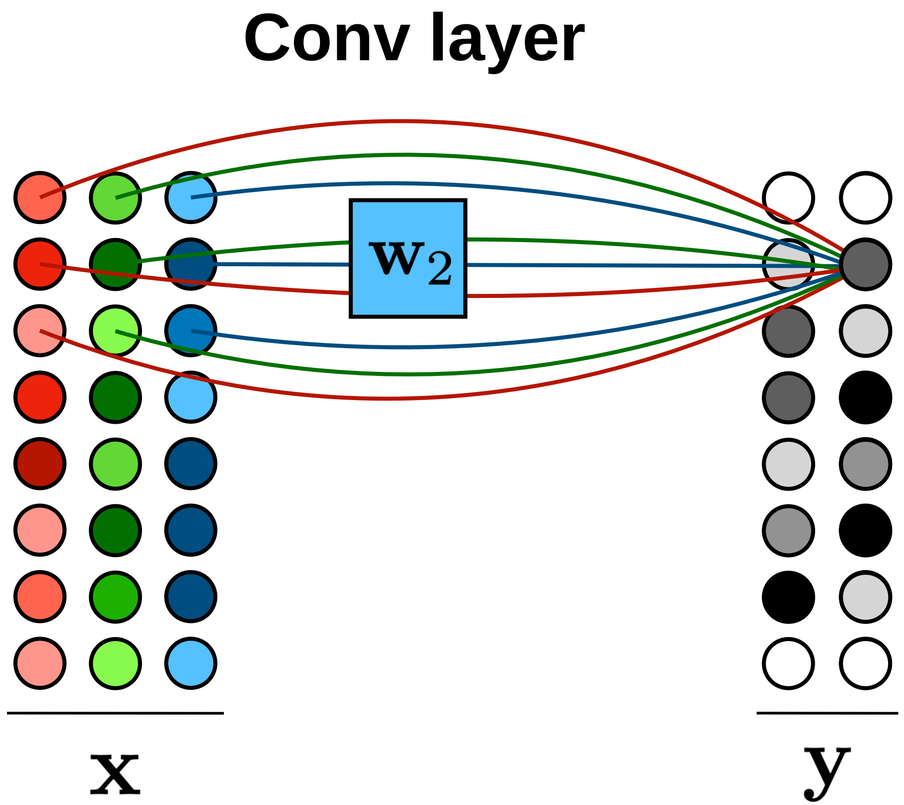
Multiple channel inputs



$$\mathbf{y} = \sum_c \mathbf{w}_c \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times 1}$$

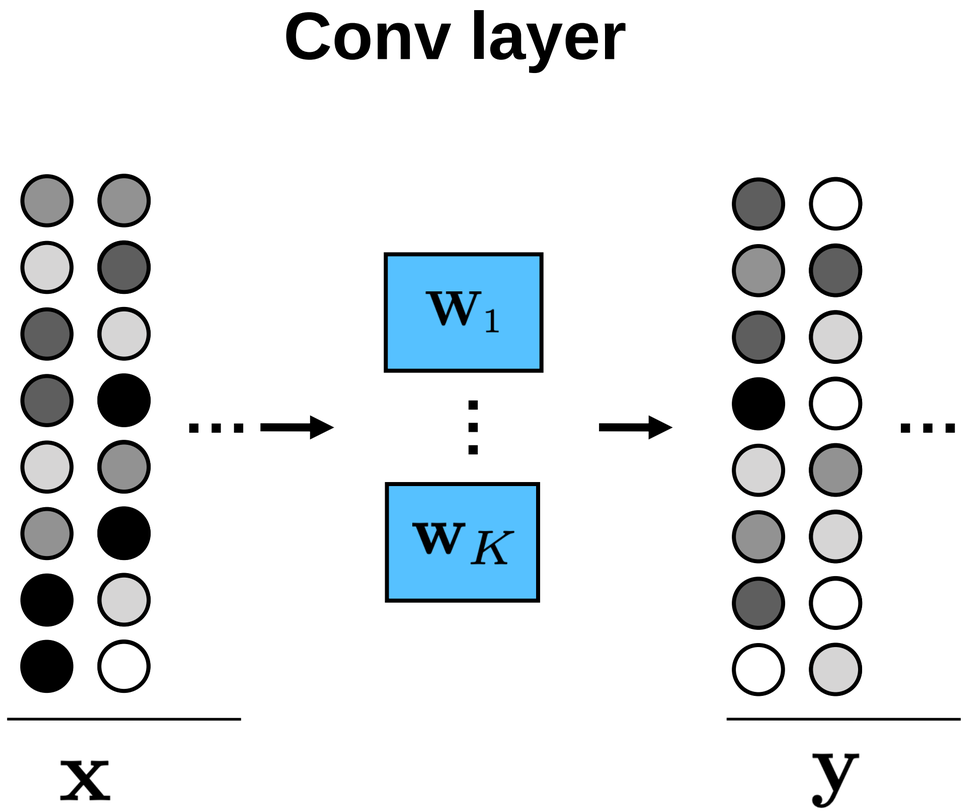
Multiple channel *outputs*



$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times K}$$

Multiple channels



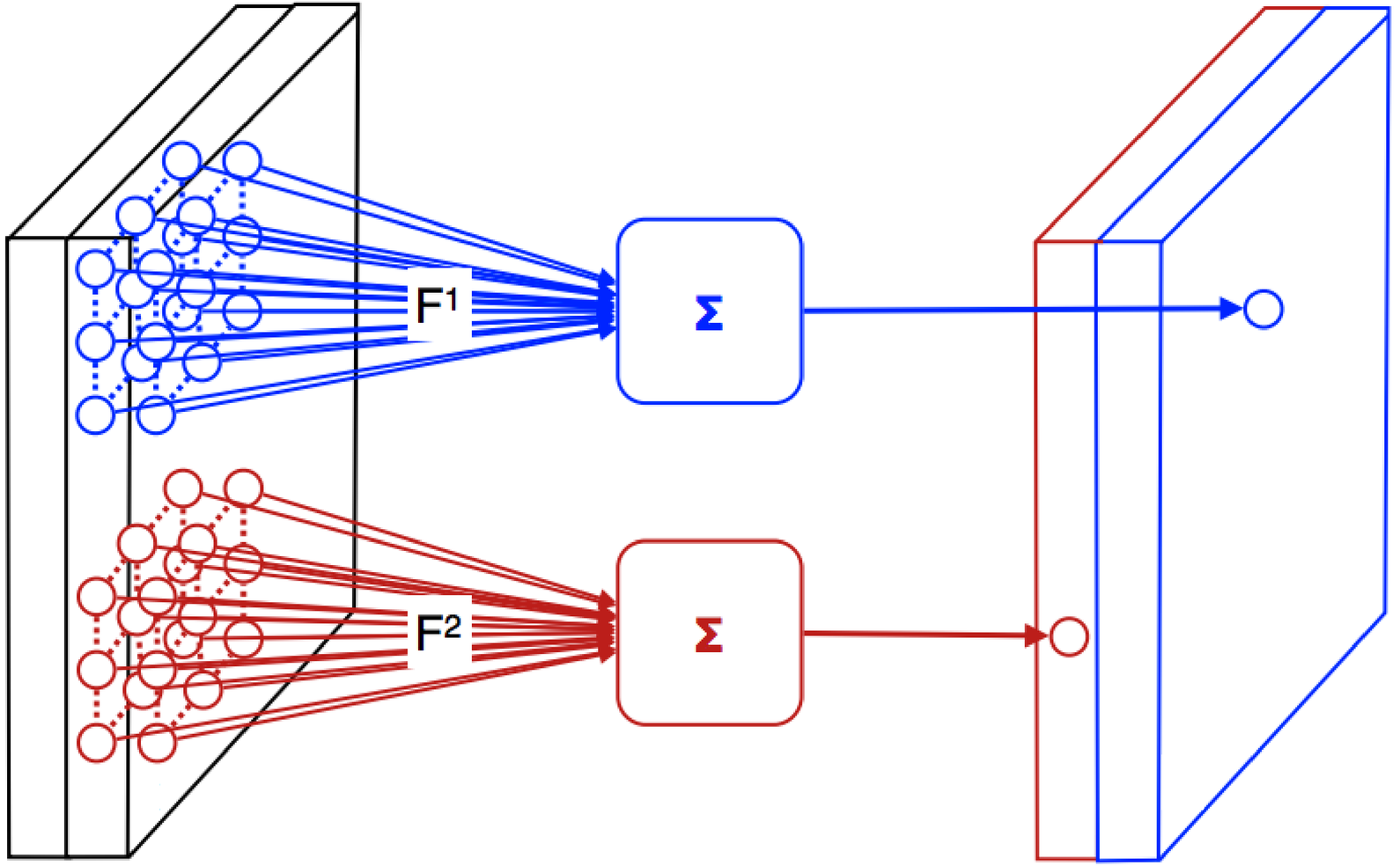
$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times K}$$

Input features

A bank of 2 filters

2-dimensional output
feature maps



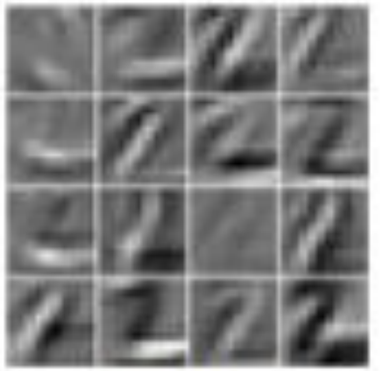
$$\mathbf{x}_l \in \mathbb{R}^{H \times W \times C_l} \quad \longrightarrow \quad \mathbf{x}_{(l+1)} \in \mathbb{R}^{H \times W \times C_{(l+1)}}$$

[Figure modified from Andrea Vedaldi]

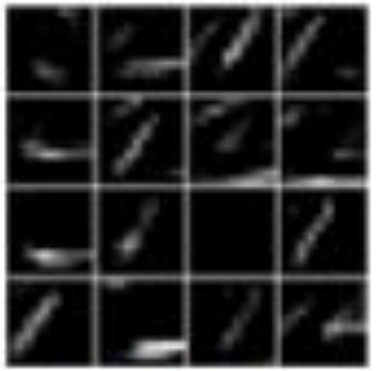
Feature maps

conv1

relu1



conv2

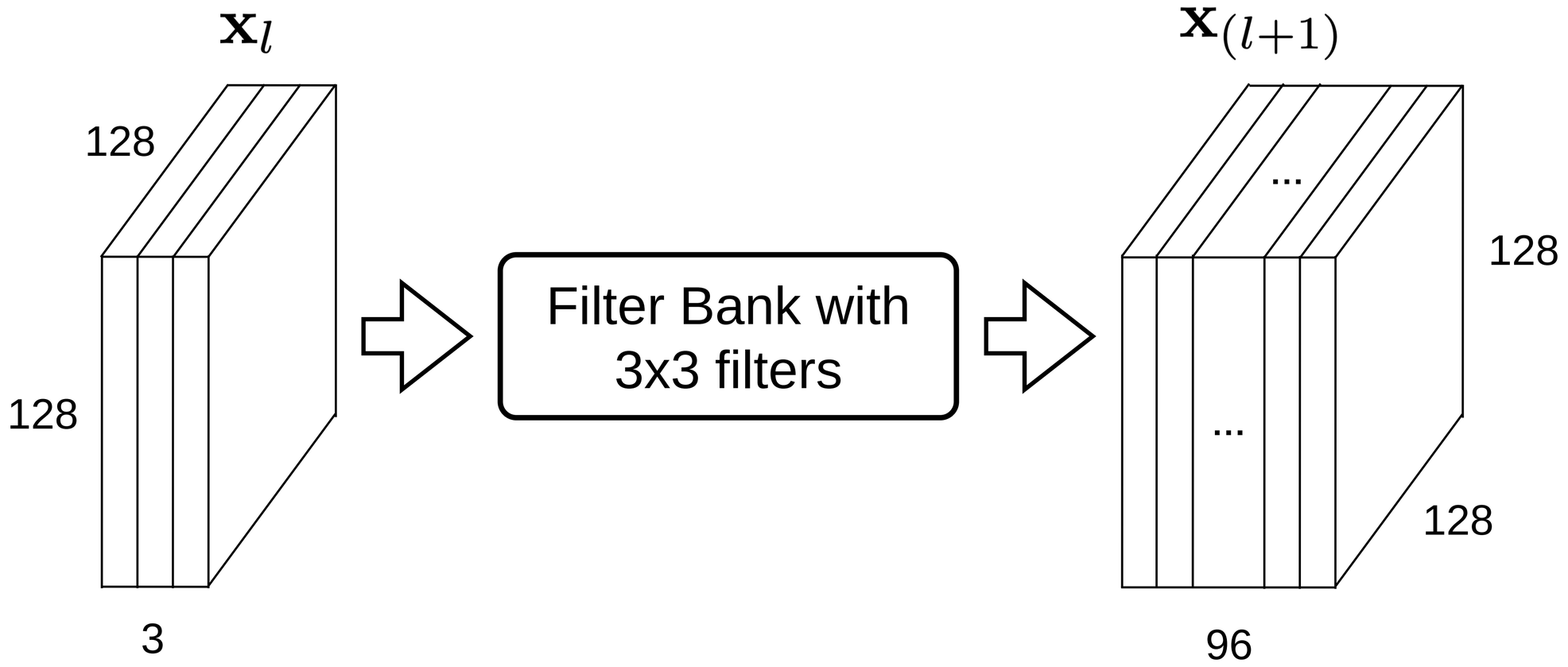


relu2

- Each layer can be thought of as a set of C **feature maps** aka **channels**
- Each feature map is an NxM image

Image source: https://stackoverflow.com/questions/45678473/convolution-neural-networks-all-feature-maps-are-blackpixel-value-is-0

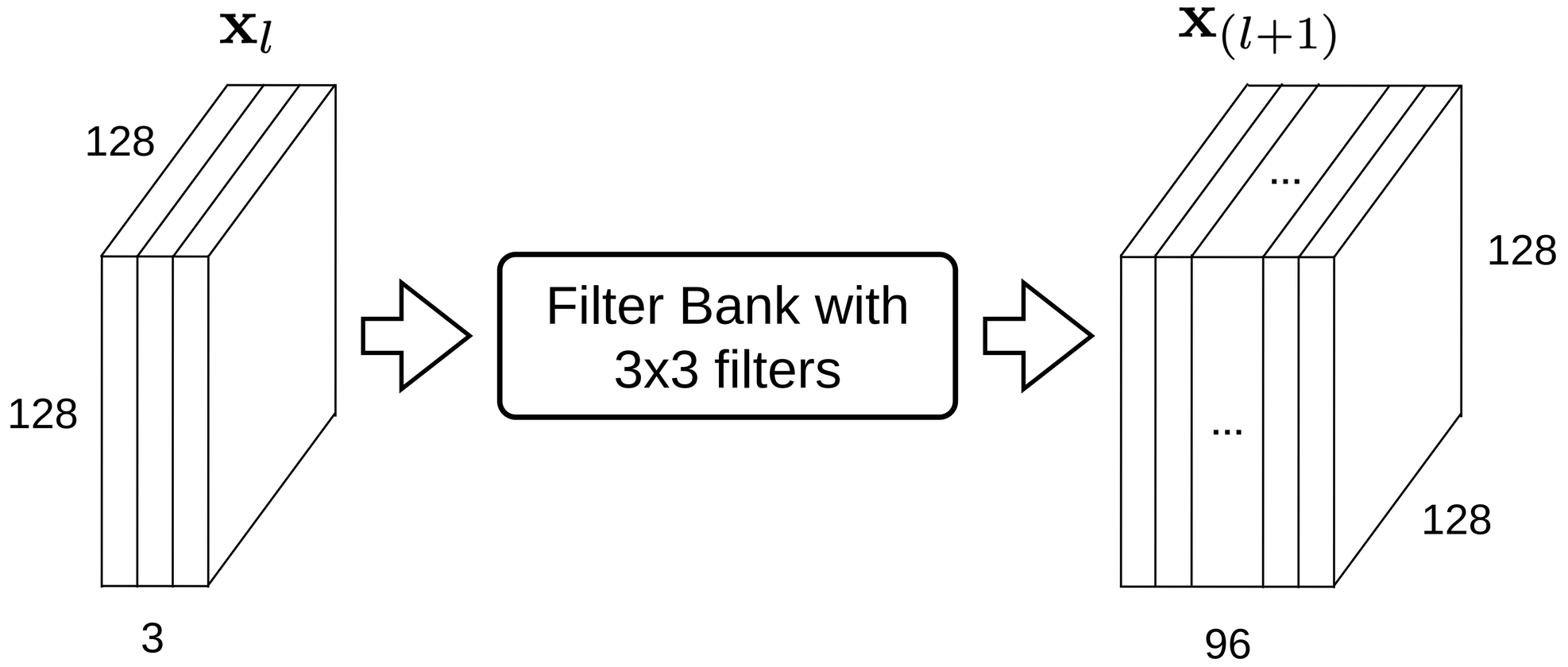
Multiple channels: Example



How many parameters does each ***filter*** have?

(a) 9 (b) 27 (c) 96 (d) 864

Multiple channels: Example



How many filters are in the bank?

(a) 3 (b) 27 (c) 96 (d) can’t say