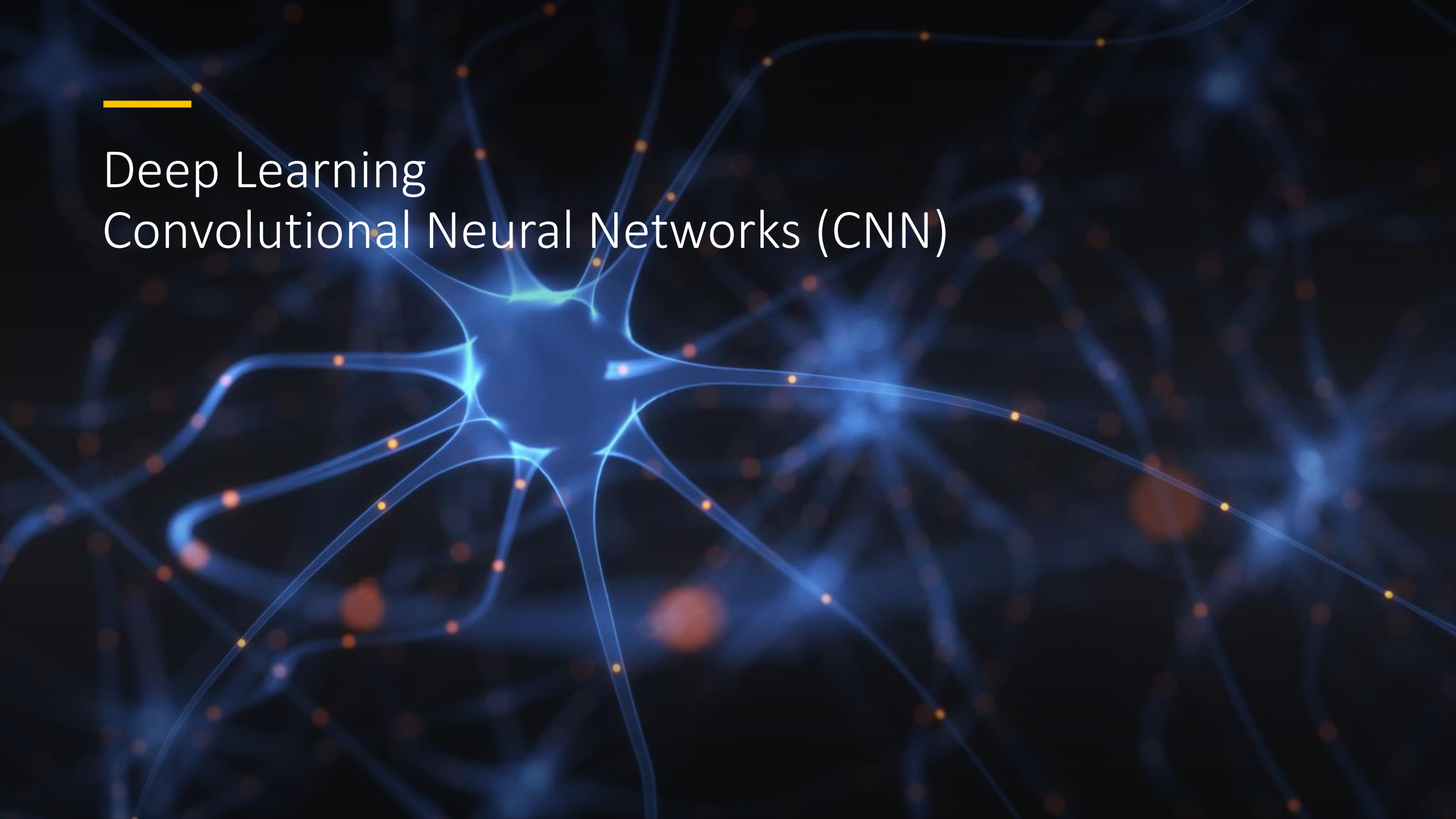
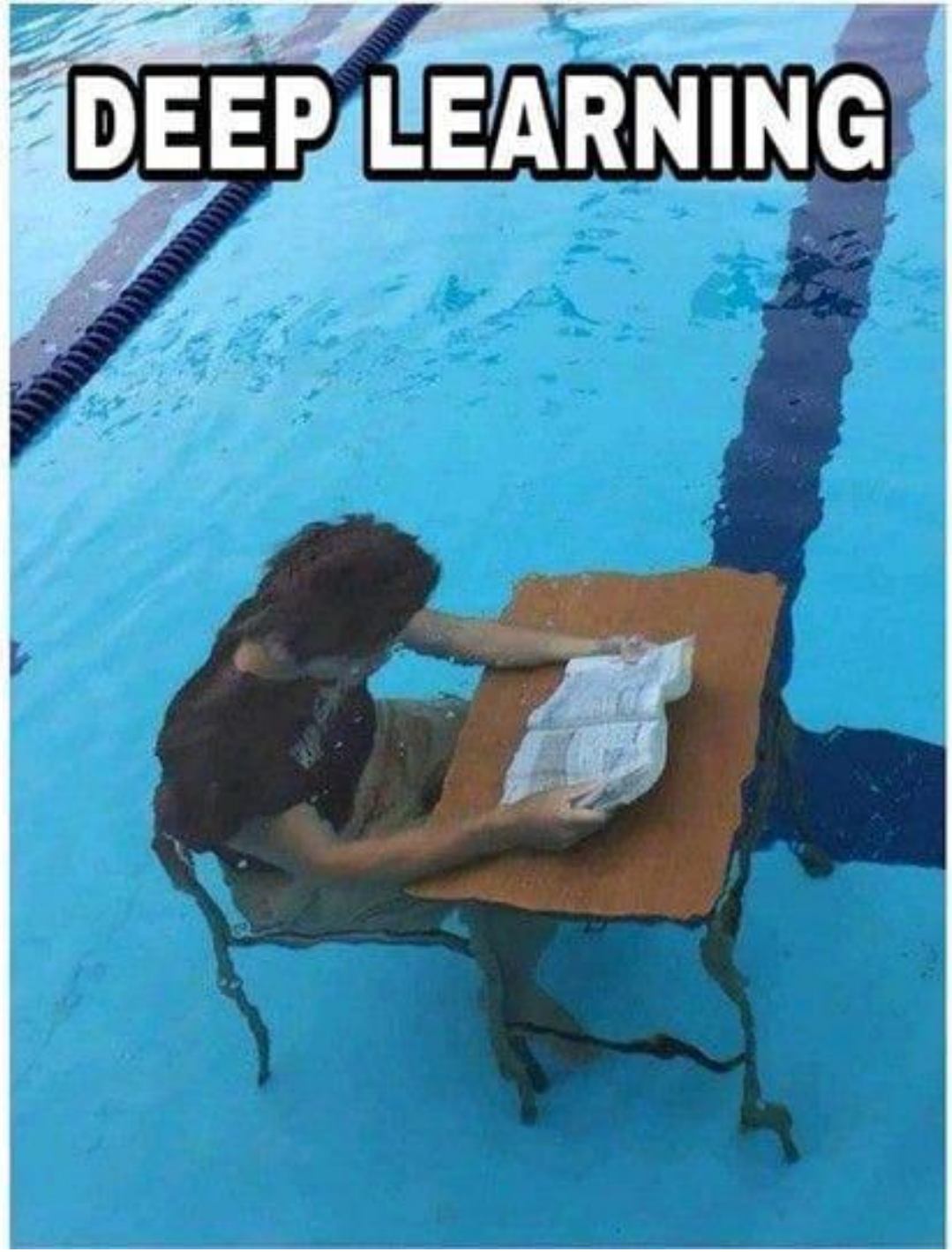


Deep Learning
Convolutional Neural Networks (CNN)


Sales team presenting the solution in Powerpoint
Excited Customer
Egineering team knowing the solution is not technically possible


data scientist
unknown distribution
Gaussian


Https://expert-programming-tutor.com
Machine Learning
Statistics
Calculus
Discrete Structure
Problem solving
Data Structure and Algorithm
python


DEEP LEARNING

ACM Transactions on Graphics

Search within TOG

ACM Transactions on Graphics

Association for Computing Machinery

ACM Transactions on Graphics (TOG) is the foremost peer-reviewed journal in the graphics field. In the colorful pages of *TOG*, leading researchers discuss breakthroughs in computer-aided design, synthetic image generation, rendering, solid modeling, and other areas. "Research," the largest regular section, ... (More)

SIG

SIGGRAPH

Special Interest Group on Computer Graphics

ACM SIGGRAPH Computer Graphics

Association for Computing Machinery

SIGGRAPH's mission is to promote the generation and dissemination of information on computer graphics and interactive techniques. Members include researchers, developers and users from the technical, academic, business, and artistic communities. Membership in SIGGRAPH includes a subscription to the ... (More)

RESEARCH-ARTICLE

Prince of Persia: the Forgotten Sands

Author: Szilvia Aszmann Authors Info & Claims

SIGGRAPH '10: ACM SIGGRAPH 2010 Computer Animation Festival • July 2010 • Pages 87 • https://doi.org/10.1145/1836623.1836682

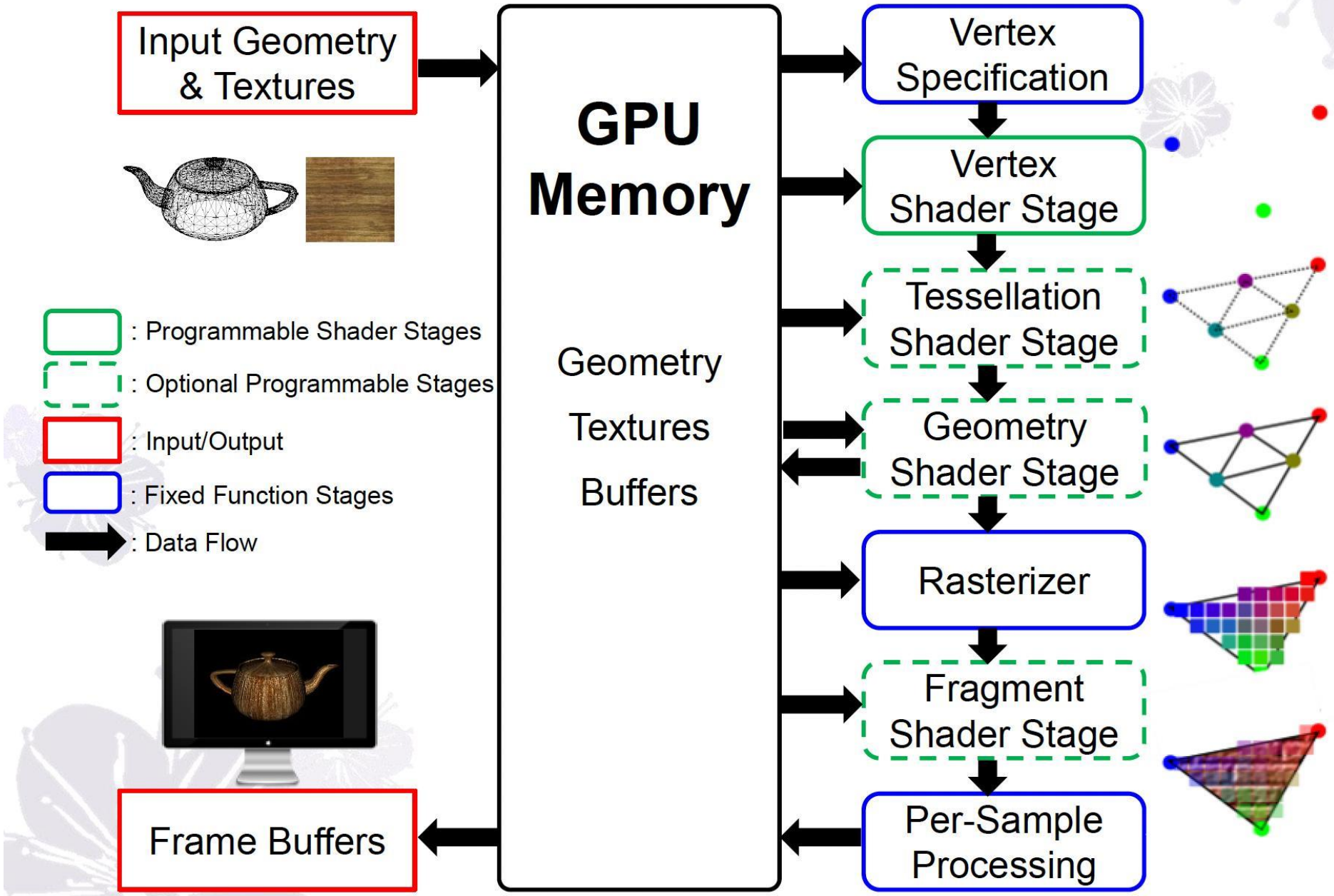
Published: 26 July 2010 Publication History

ABSTRACT

The young Prince of Persia, eager to defend his brother's kingdom, learns that he holds the key to defeating the relentless enemies of legend. Outnumbered and desperate, it soon becomes clear


PRINCE OF PERSIA®

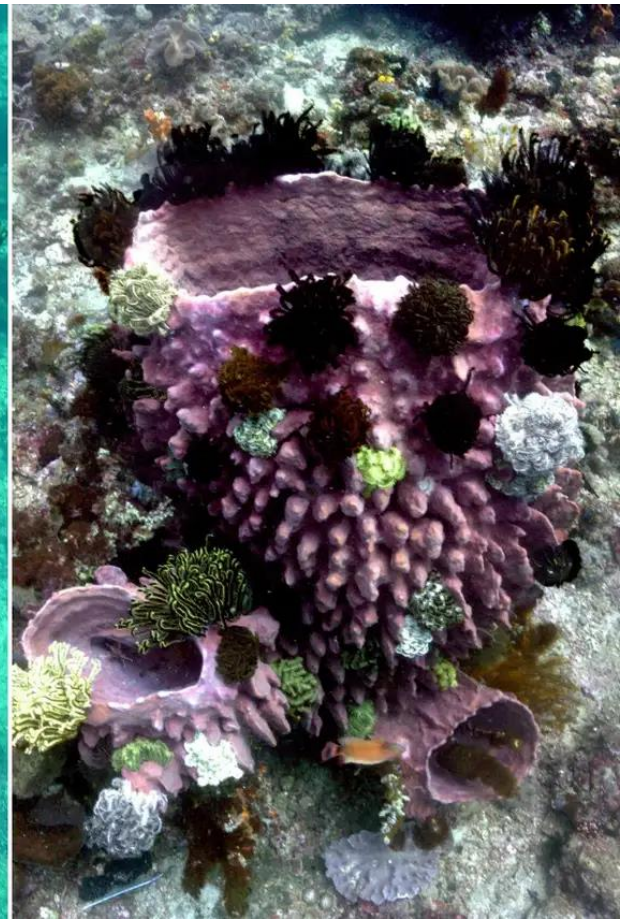
The OpenGL Rasterization Rendering Pipeline



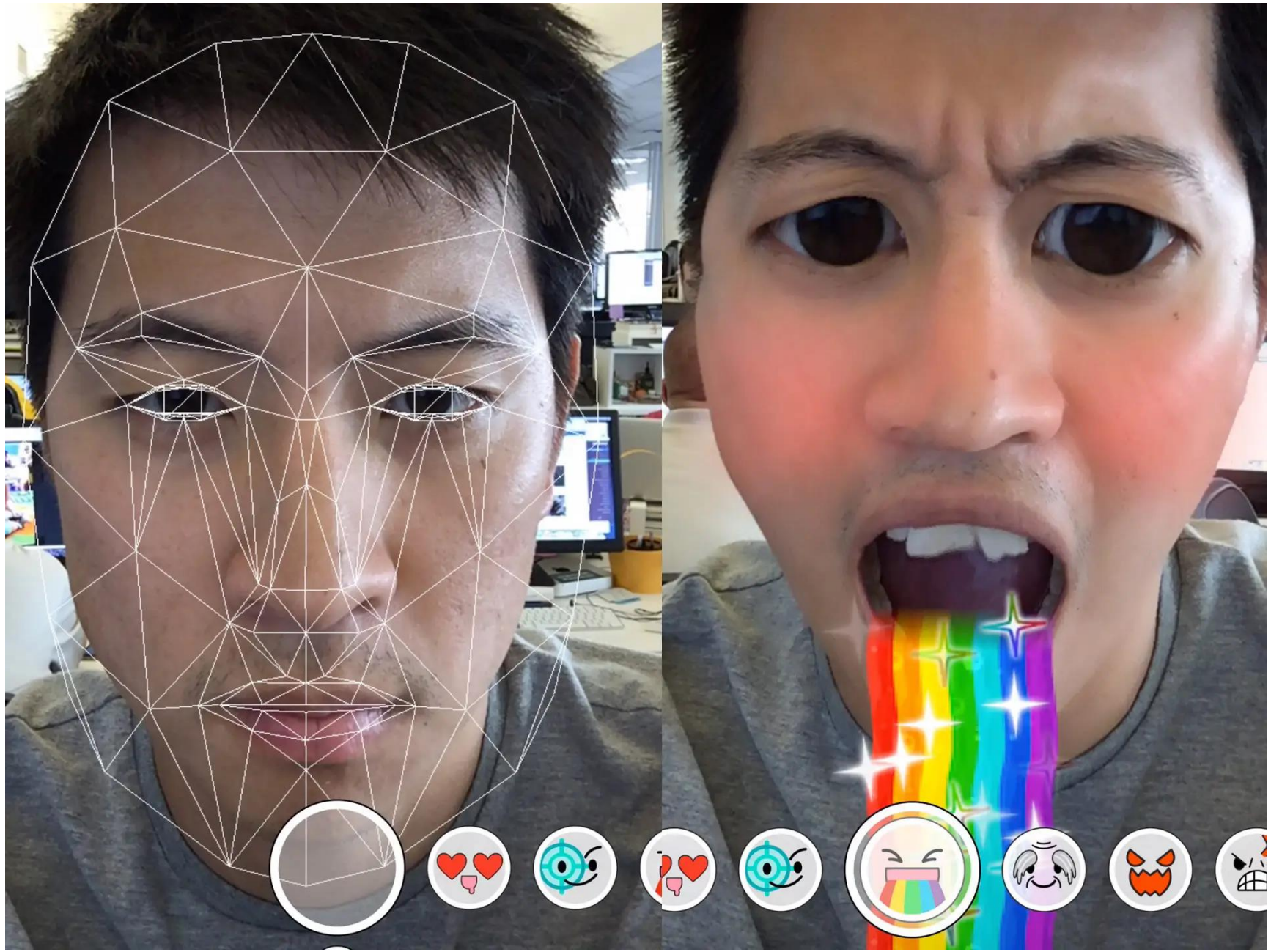
A Scanner Darkly (2006)

[That 'AI-Generated' Anime Is Pissing Off Professional Animators]

Sea-Thru



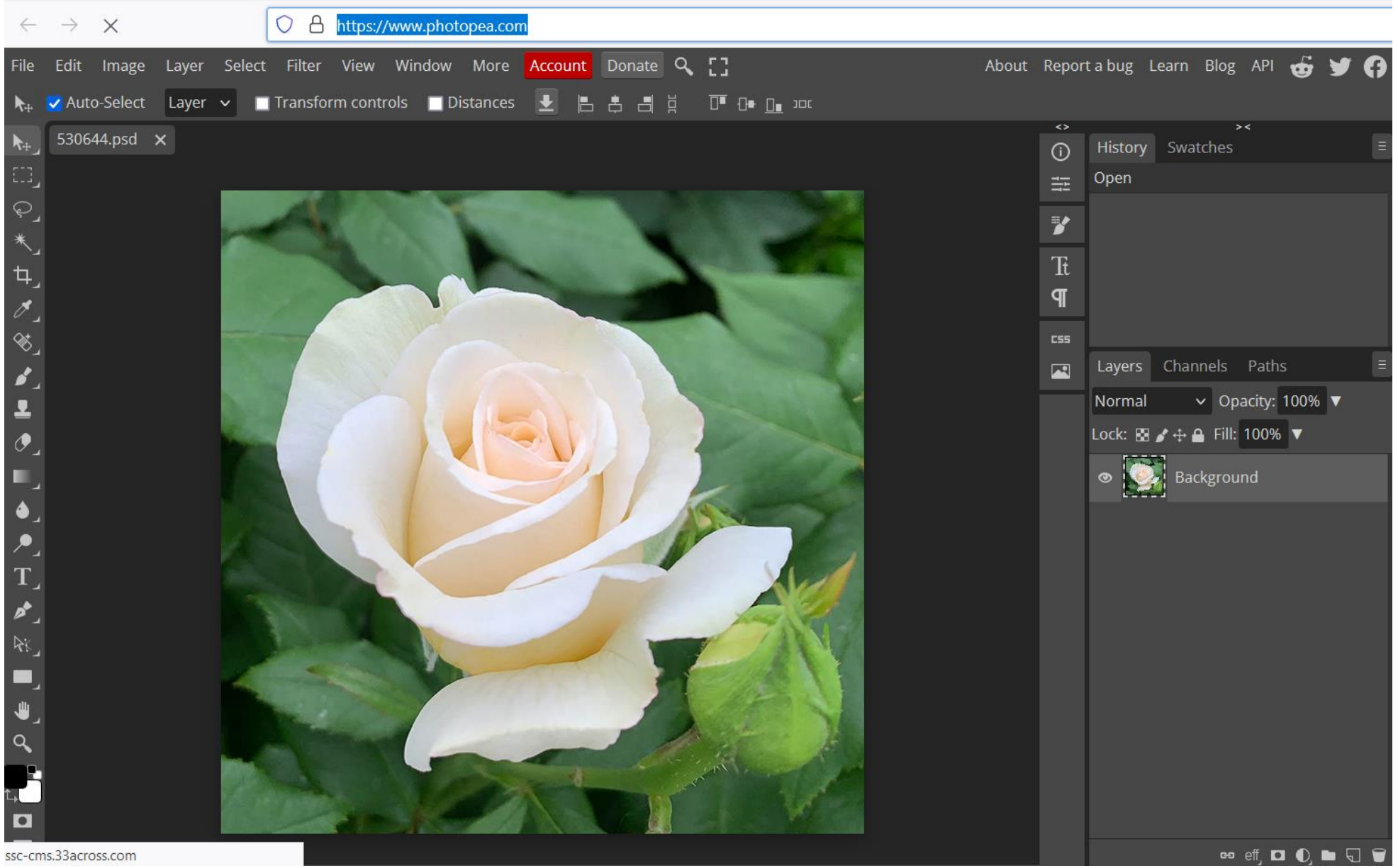

SEEING THROUGH THE SEA
SCIENTIFIC
AMERICAN.



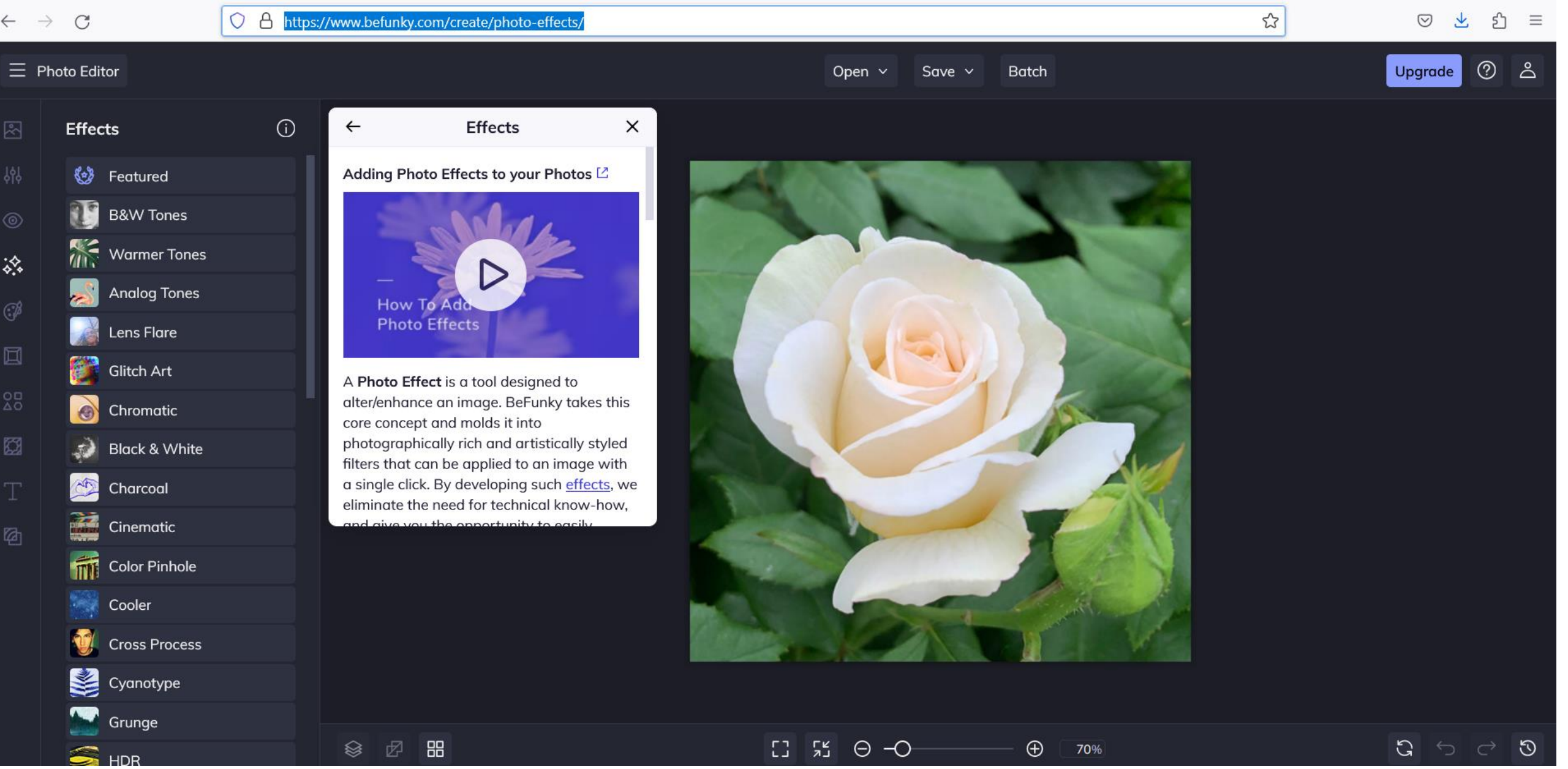

SYSTEM 02H789
BRIDGE CONTROL

https://www.youtube.com/watch?v=cW3rx7jiX8E

Light L16 Camera

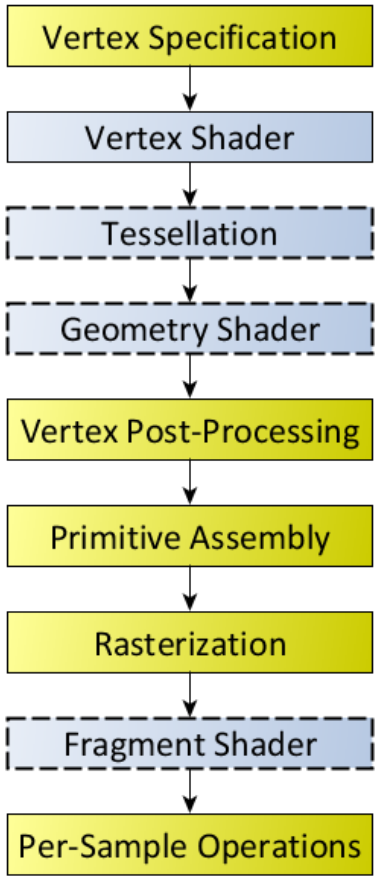


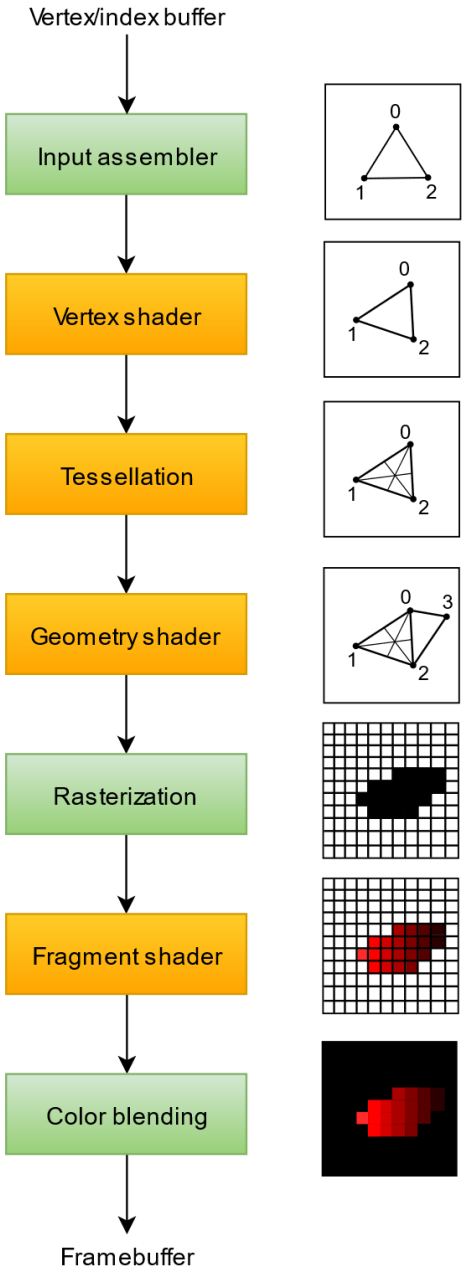
https://www.photopea.com/



https://www.befunky.com/create/photo-effects/

GPU


Vertex Specification
Vertex Shader
Tessellation
Geometry Shader
Vertex Post-Processing
Primitive Assembly
Rasterization
Fragment Shader
Per-Sample Operations


Vertex/index buffer
Input assembler
Vertex shader
Tessellation
Geometry shader
Rasterization
Fragment shader
Color blending
Framebuffer
0
1
2
0
1
2
0
1
2
0
3
1
2

A little history

- How do animals see?
  - What is the neural process from eye to recognition?

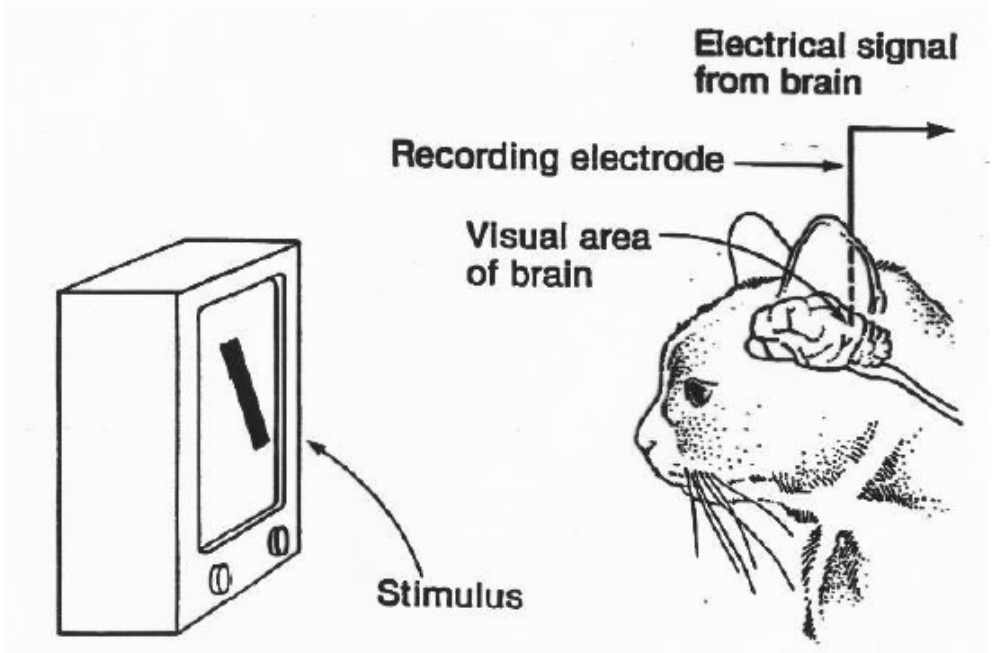
- Early research:
  - largely based on behavioral studies
    - Study behavioral judgment in response to visual stimulation
    - Visual illusions
  - and gestalt
    - Brain has innate tendency to organize disconnected bits into whole objects
  - But no real understanding of how the brain processed images

Hubel and Wiesel 1959

- First study on neural correlates of vision.
  - "Receptive Fields in Cat Striate Cortex"
    - "Striate Cortex": Approximately equal to the V1 visual cortex
      - "Striate" – defined by structure, "V1" – functional definition

- 24 cats, anaesthetized, immobilized, on artificial respirators
  - Anaesthetized with truth serum
  - Electrodes into brain
    - Do not report if cats survived experiment, but claim brain tissue was studied

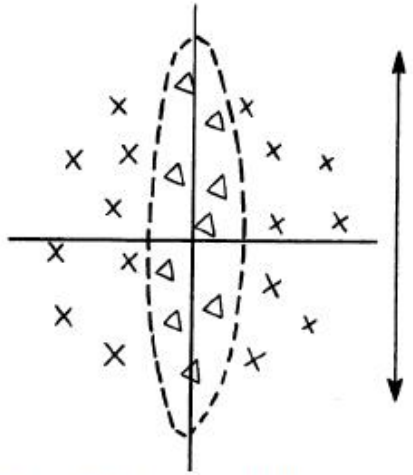
Hubel and Wiesel 1959



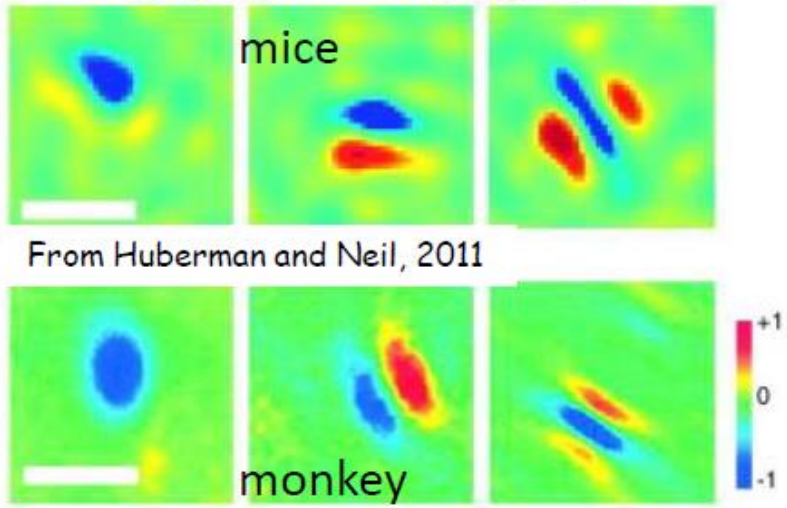
- Light of different wavelengths incident on the retina through fully open (slitted) Iris
  - Defines *immediate* (20ms) response of retinal cells

- Beamed light of different patterns into the eyes and measured neural responses in striate cortex

Hubel and Wiesel 1959



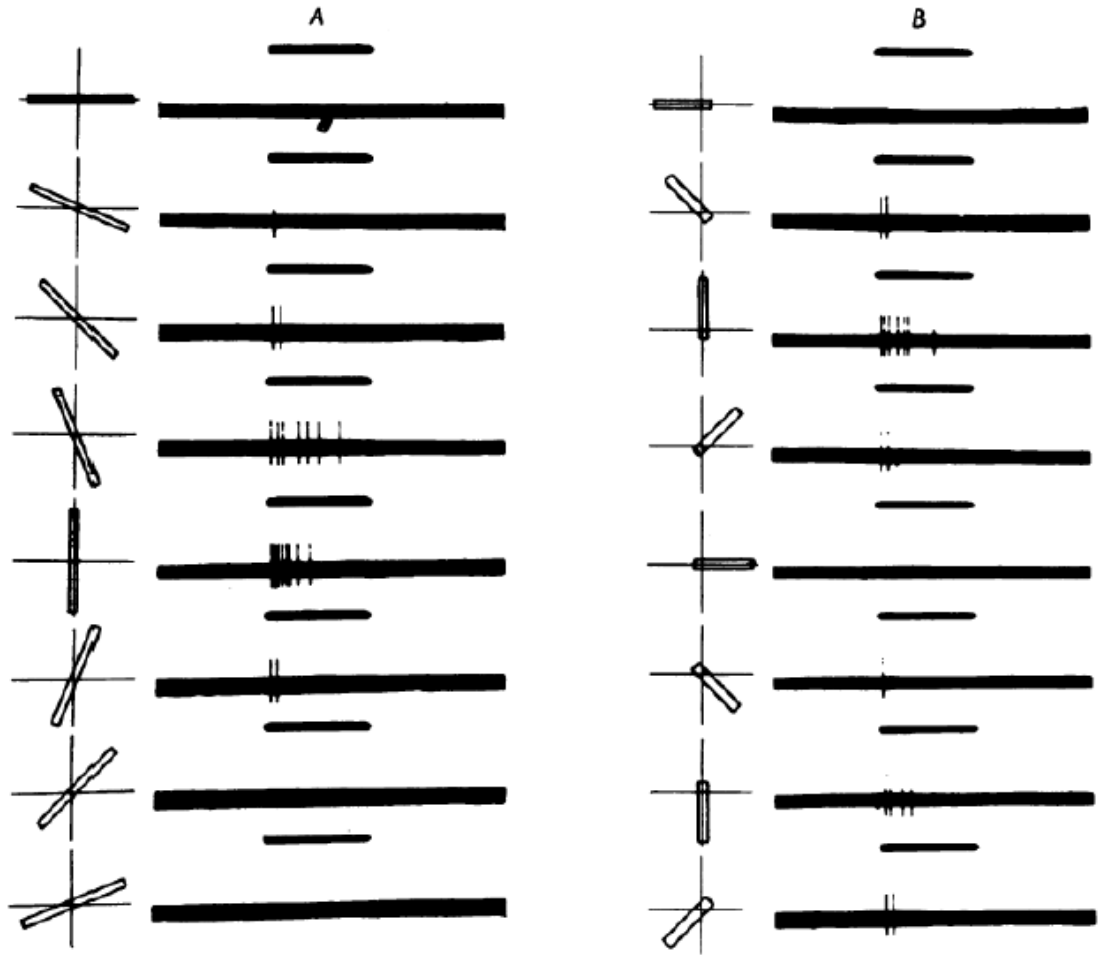
From Hubel and Wiesel



From Huberman and Neil, 2011

- Restricted retinal areas which on illumination influenced the firing of single cortical units were called ***receptive fields***.
  - These fields were usually subdivided into excitatory and inhibitory regions.
- Findings:
  - A light stimulus covering the whole receptive field, or diffuse illumination of the whole retina, was ineffective in driving most units, as excitatory regions cancelled inhibitory regions
    - Light must fall on excitatory regions and NOT fall on inhibitory regions, resulting in clear patterns
  - Receptive fields could be oriented in a vertical, horizontal or oblique manner.
    - Based on the arrangement of excitatory and inhibitory regions within receptive fields.
  - A spot of light gave greater response for some directions of movement than others.

Hubel and Wiesel 59



- Response as orientation of input light rotates
  - Note spikes – this neuron is sensitive to vertical bands

Hubel and Wiesel

- Oriented slits of light were the most effective stimuli for activating striate cortex neurons

- The orientation selectivity resulted from *the previous level of input* because lower level neurons responding to a slit also responded to patterns of spots if they were aligned with the same orientation as the slit.

- In a later paper (Hubel & Wiesel, 1962), they showed that within the striate cortex, two levels of processing could be identified
  - Between neurons referred to as *simple* S-cells and *complex* C-cells.
  - Both types responded to oriented slits of light, but complex cells were not "confused" by spots of light while simple cells could be confused

Hubel and Wiesel

- Complex C-cells build from similarly oriented simple cells
  - They “fine-tune” the response of the simple cell

- Show complex buildup – building *more complex patterns* by composing early neural responses
  - Successive transformation through Simple-Complex combination layers

- Demonstrated more and more complex responses in later papers
  - Later experiments were on waking macaque monkeys
    - Too horrible to recall

Adding insult to injury..

- “However, this model cannot accommodate the color, spatial frequency and many other features to which neurons are tuned. The exact organization of all these cortical columns within V1 remains a hot topic of current research.”

Forward to 1980

- Kunihiko Fukushima
- Recognized deficiencies in the Hubel-Wiesel model

Kunihiko Fukushima

- One of the chief problems: Position invariance of input
  - Your grandmother cell fires even if your grandmother moves to a different location in your field of vision

NeoCognitron

Figures from Fukushima, '80

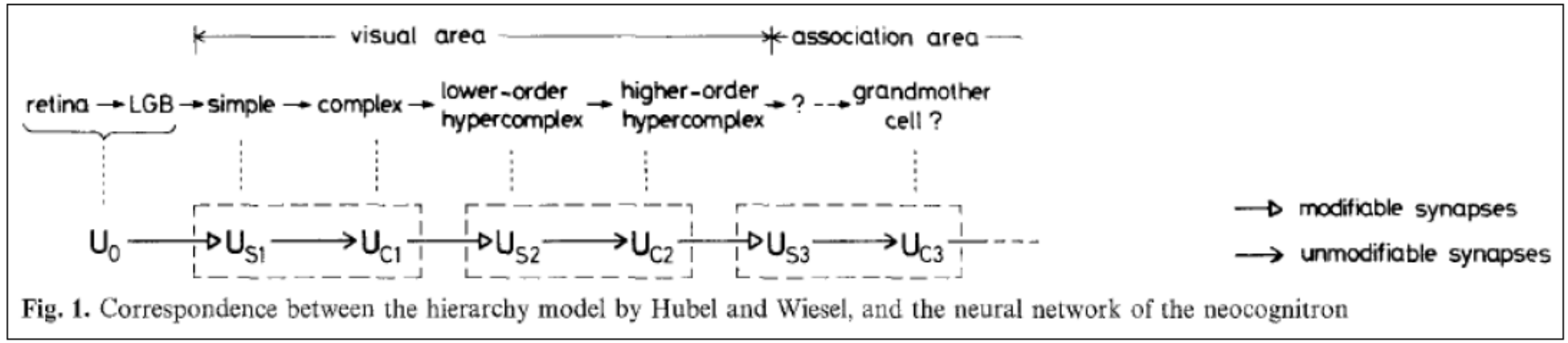


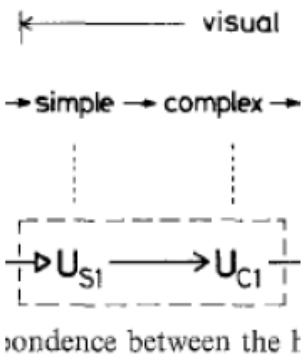
Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

- Visual system consists of a hierarchy of modules, each comprising a layer of "S-cells" followed by a layer of "C-cells"
  - U_{Sl} is the l[th] layer of S cells, U_{Cl} is the l[th] layer of C cells

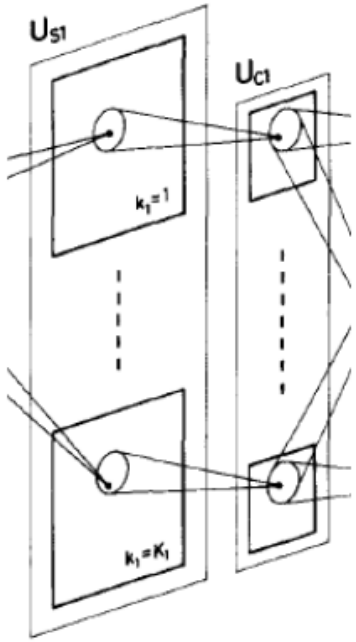
- Only S-cells are "plastic" (i.e. learnable), C-cells are fixed in their response

- S-cells ***respond*** to the signal in the previous layer
- C-cells ***confirm*** the S-cells' response

NeoCognitron



ondence between the l



Each cell in a plane "looks" at a slightly shifted region of the input to the plane than the adjacent cells in the plane.

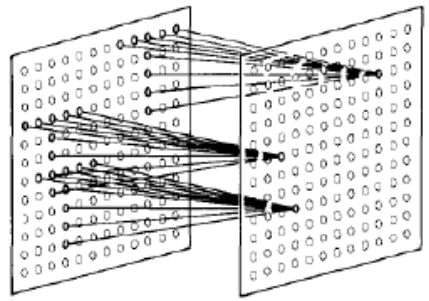
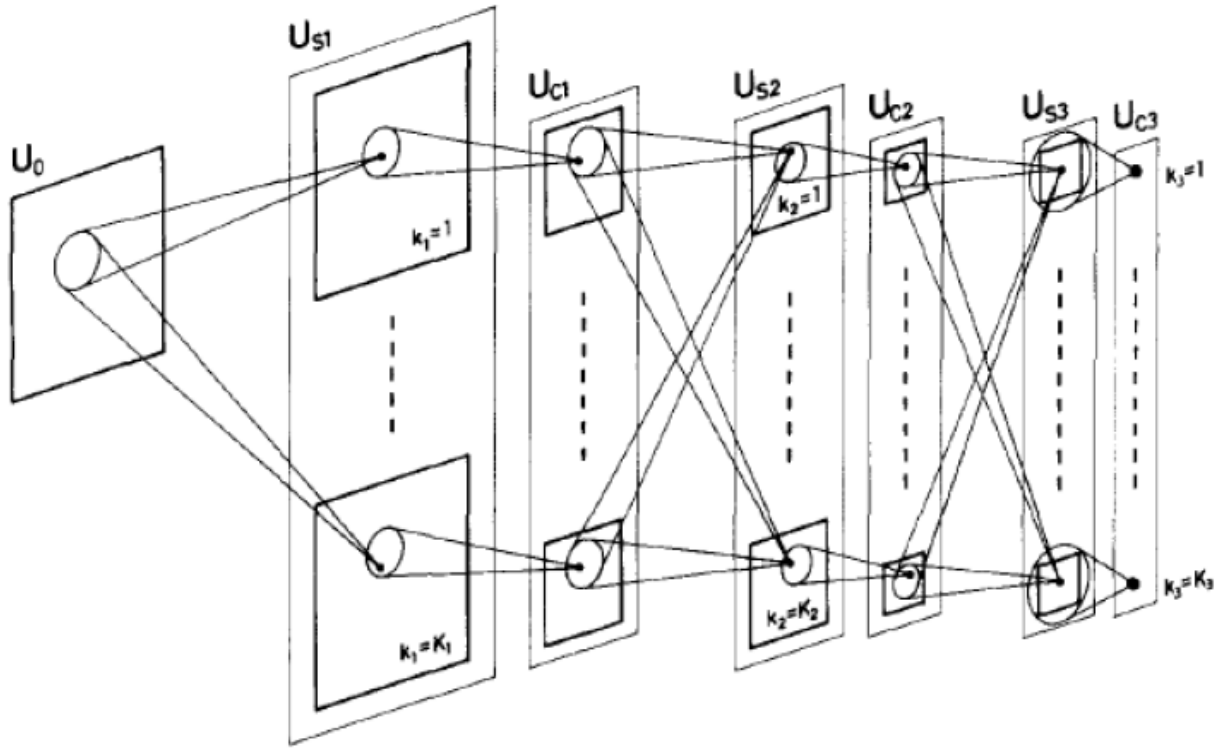


Fig. 3. Illustration showing the input interconnections to the cells within a single cell-plane

- Each simple-complex module includes a layer of S-cells and a layer of C-cells
- S-cells are organized in rectangular groups called S-planes.
  - All the cells within an S-plane have identical learned responses
- C-cells too are organized into rectangular groups called C-planes
  - One C-plane per S-plane
  - All C-cells have identical fixed response
- In Fukushima's original work, each C and S cell "looks" at an elliptical region in the previous plane

NeoCognitron



- The complete network
- U_0 is the retina

- In each subsequent module, the planes of the S layers detect plane-specific patterns in the previous layer (C layer or retina)

- The planes of the C layers “refine” the response of the corresponding planes of the S layers

Neocognitron

- S cells: RELU like activation

$$u_{Sl}(k_l, \mathbf{n}) = r_l \cdot \varphi\left[\frac{1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{\mathbf{v} \in S_l} a_l(k_{l-1}, \mathbf{v}, k_l) \cdot u_{Cl-1}(k_{l-1}, \mathbf{n}+\mathbf{v})}{1 + \frac{2r_l}{1+r_l} \cdot b_l(k_l) \cdot v_{Cl-1}(\mathbf{n})} - 1\right]$$

  - φ is a RELU

- C cells: Also RELU like, but with an inhibitory bias
  - Fires if weighted combination of S cells fires strongly enough

$$u_{Cl}(k_l, \mathbf{n}) = \psi\left[\frac{1 + \sum_{\mathbf{v} \in D_l} d_l(\mathbf{v}) \cdot u_{Sl}(k_l, \mathbf{n}+\mathbf{v})}{1 + v_{Sl}(\mathbf{n})} - 1\right]$$

  - $$\psi[x] = \varphi[x/(\alpha + x)]$$

Neocognitron

- S cells: RELU like activation

$$u_{Sl}(k_l,\mathbf{n}) = r_l \cdot \varphi\left[\frac{1+\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{\mathbf{v}\in S_l} a_l(k_{l-1},\mathbf{v},k_l)\cdot u_{Cl-1}(k_{l-1},\mathbf{n}+\mathbf{v})}{1+\frac{2r_l}{1+r_l}\cdot b_l(k_l)\cdot v_{Cl-1}(\mathbf{n})} - 1\right]$$

  - φ is a RELU

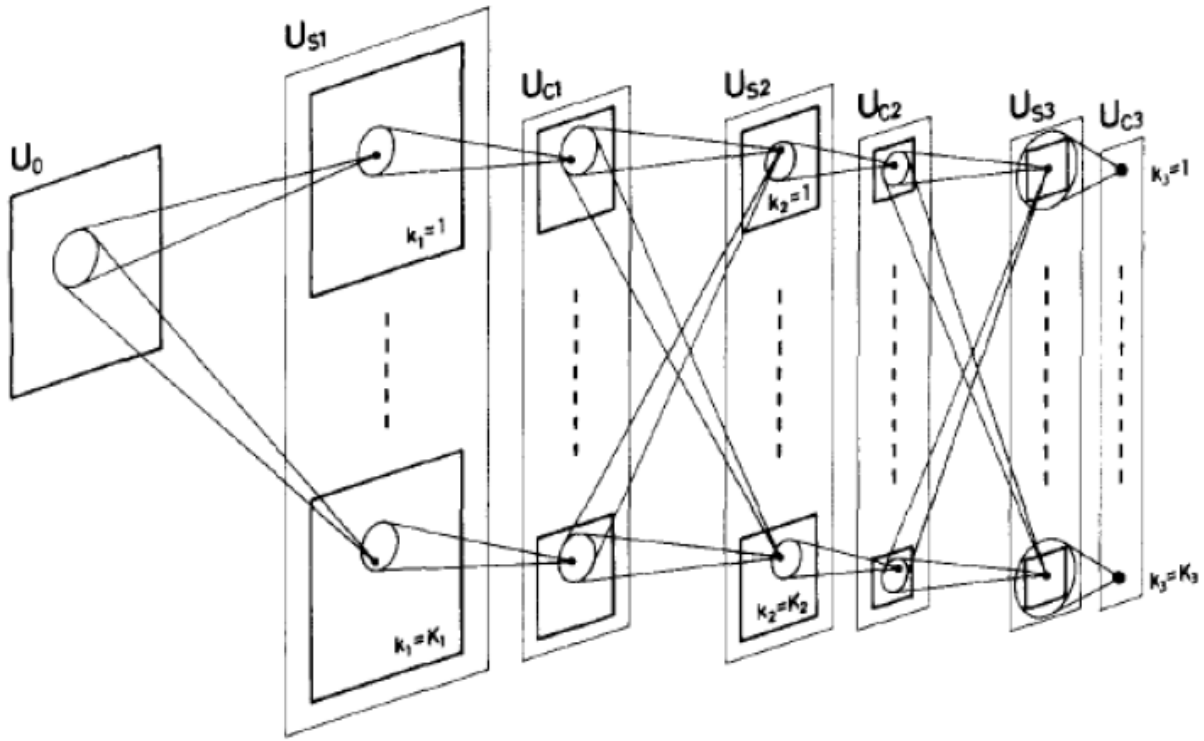
- C cells: Also RELU like, but with an inhibitory bias
  - Fires if weighted combination of S cells fires strongly enough

$$u_{Cl}(k_l,\mathbf{n}) = \psi\left[\frac{1+\sum_{\mathbf{v}\in D_l} d_l(\mathbf{v})\cdot u_{Sl}(k_l,\mathbf{n}+\mathbf{v})}{1+v_{Sl}(\mathbf{n})} - 1\right]$$

$$\psi[x] = \varphi[x/(\alpha+x)]$$

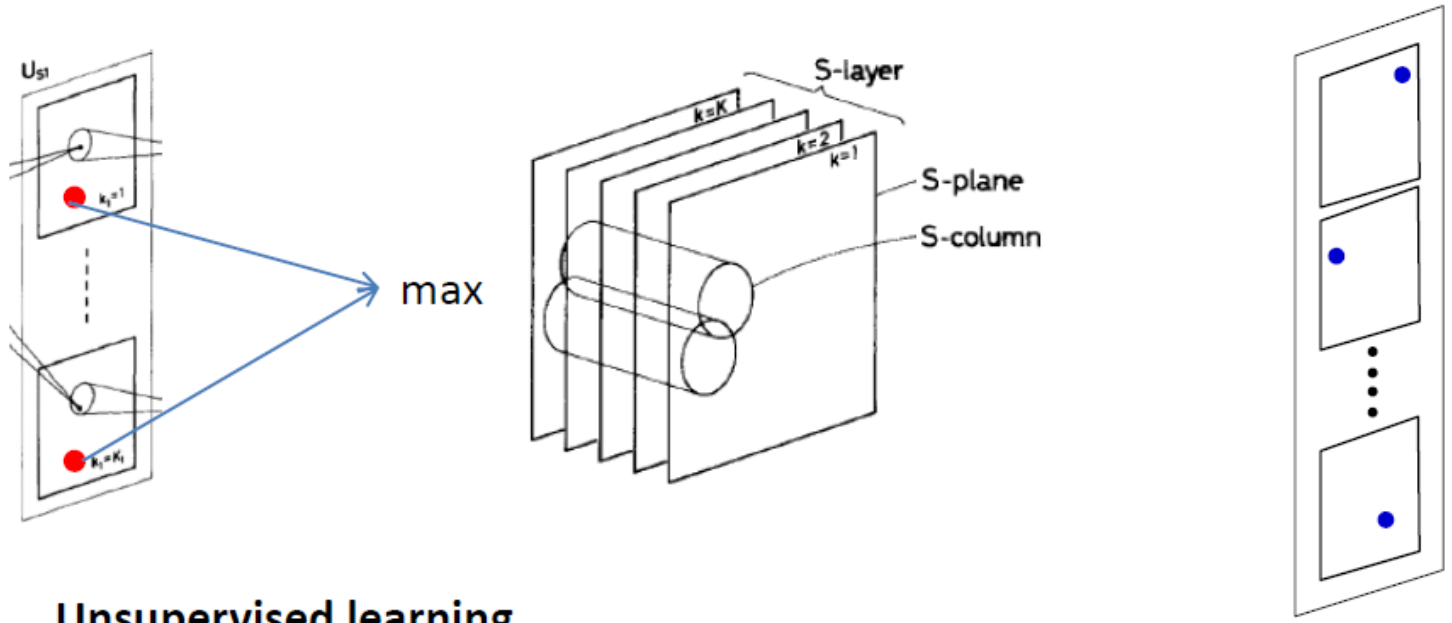
Could simply replace these strange functions with a RELU and a max

NeoCognitron



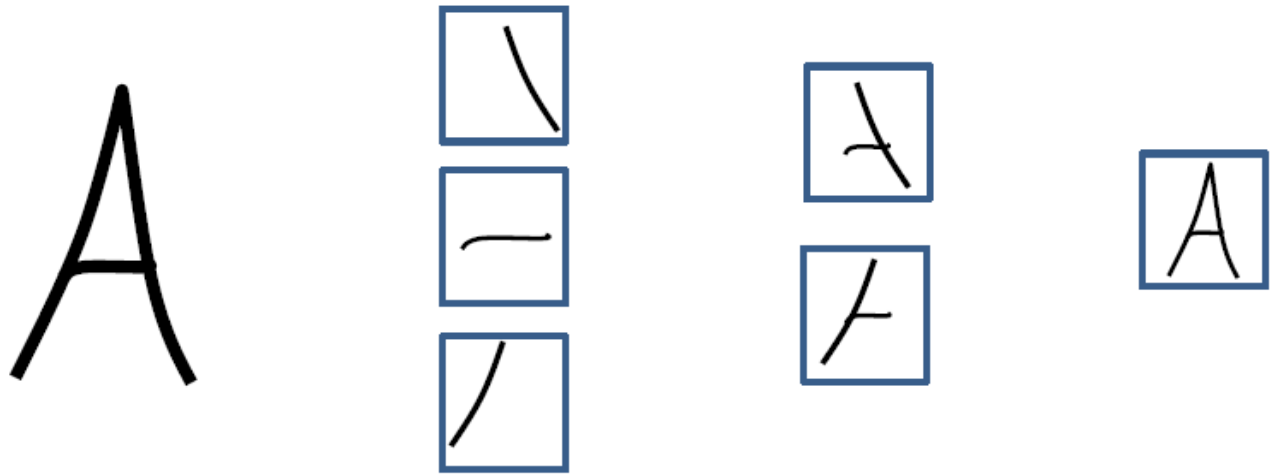
- The deeper the layer, the larger the receptive field of each neuron
  - Cell planes get smaller with layer number
  - Number of planes increases
    - i.e the number of complex pattern detectors increases with layer

Learning in the neo-cognitron



- **Unsupervised learning**
- Randomly initialize S cells, perform Hebbian learning updates in response to input
  - update = product of input and output : $\Delta w_{ij} = x_i y_j$
- Within any layer, at any position, only the maximum S from all the layers is selected for update
  - Also viewed as max-valued cell from each *S column*
  - Ensures only one of the planes picks up any feature
  - But across all positions, multiple planes will be selected
- If multiple max selections are on the same plane, only the largest is chosen
- Updates are distributed across all cells within the plane

Learning in the neo-cognitron

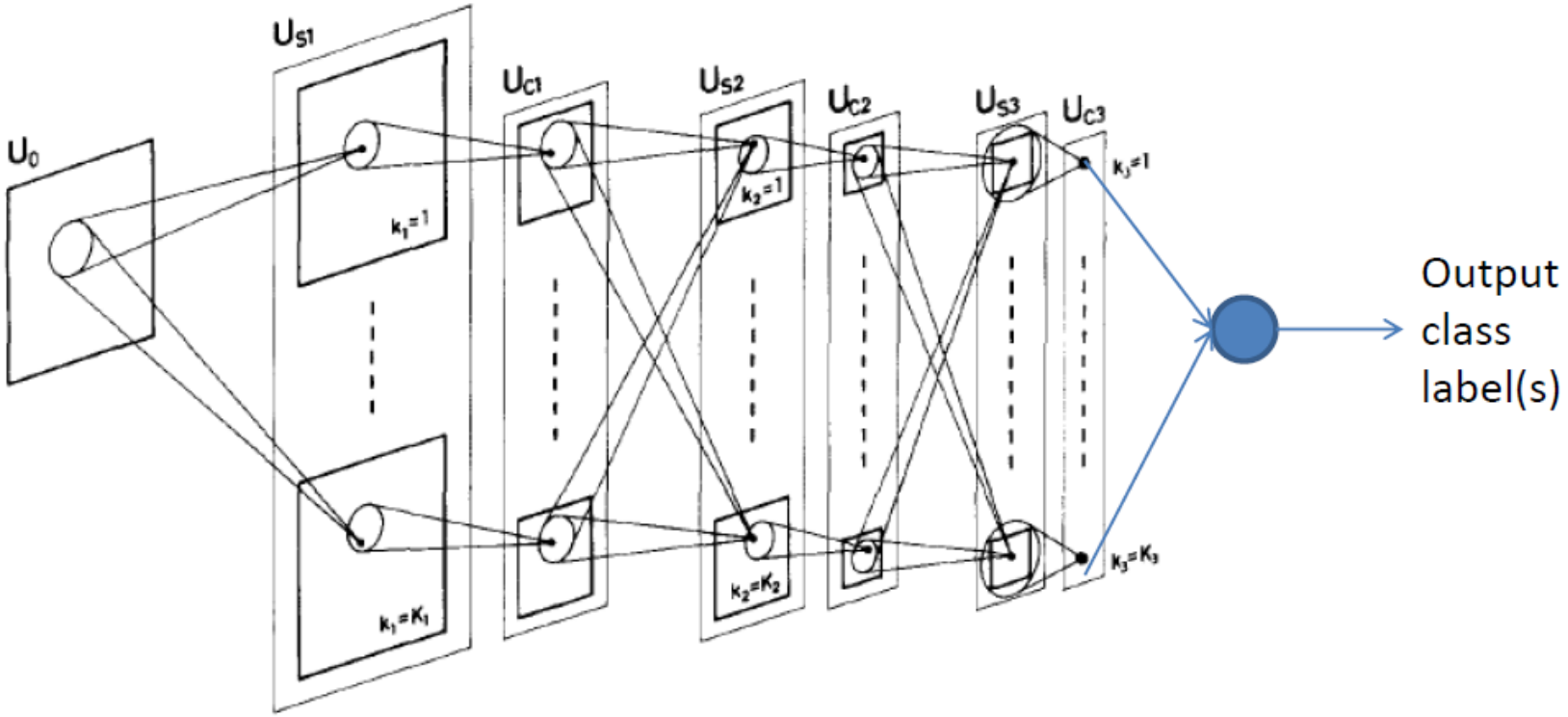


- Ensures different planes learn different features
- Any plane learns only one feature
  - E.g. Given many examples of the character “A” the different cell planes in the S-C layers may learn the patterns shown
    - Given other characters, other planes will learn their components
  - Going up the layers goes from local to global receptor fields
- Winner-take-all strategy makes it robust to distortion
- Unsupervised: Effectively clustering

Adding Supervision

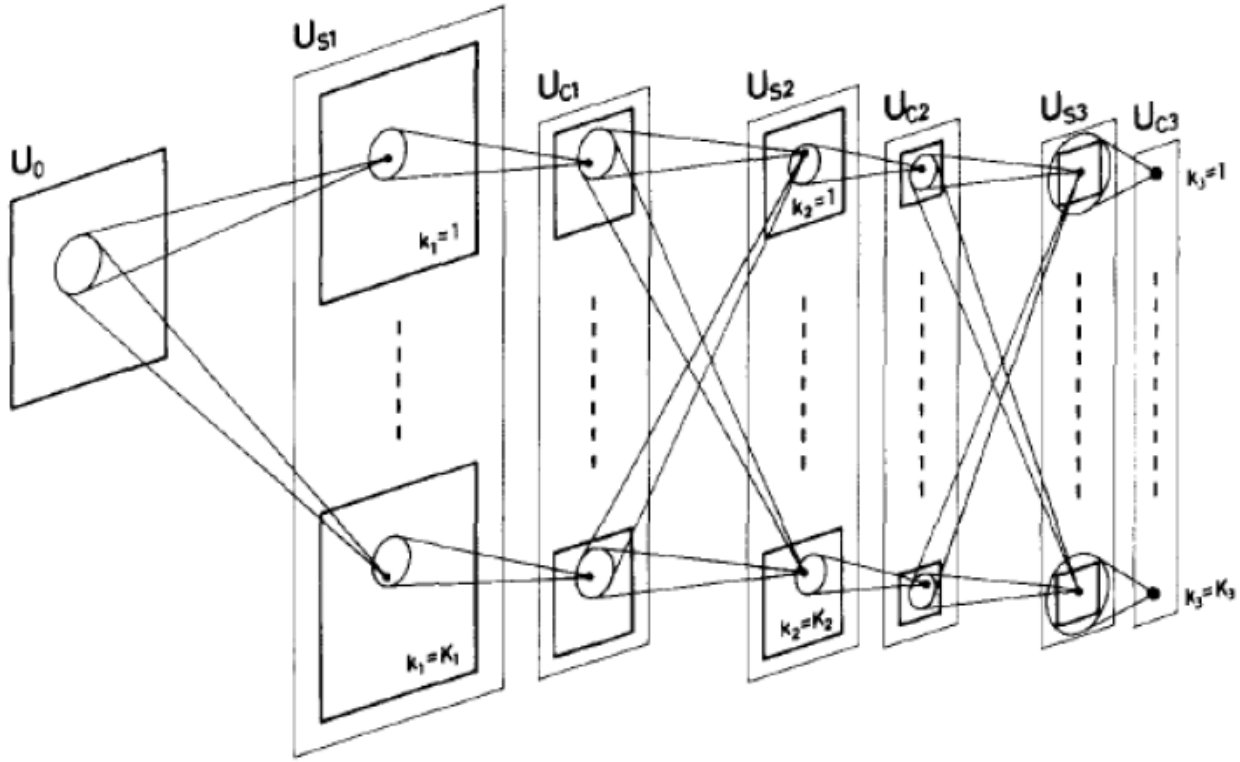
- The neocognitron is fully unsupervised
  - Semantic labels are automatically learned
- Can we add external supervision?
- Various proposals:
  - Temporal correlation: Homma, Atlas, Marks, '88
  - TDNN: Lang, Waibel et. al., 1989, '90
- Convolutional neural networks: LeCun

Supervising the neocognitron



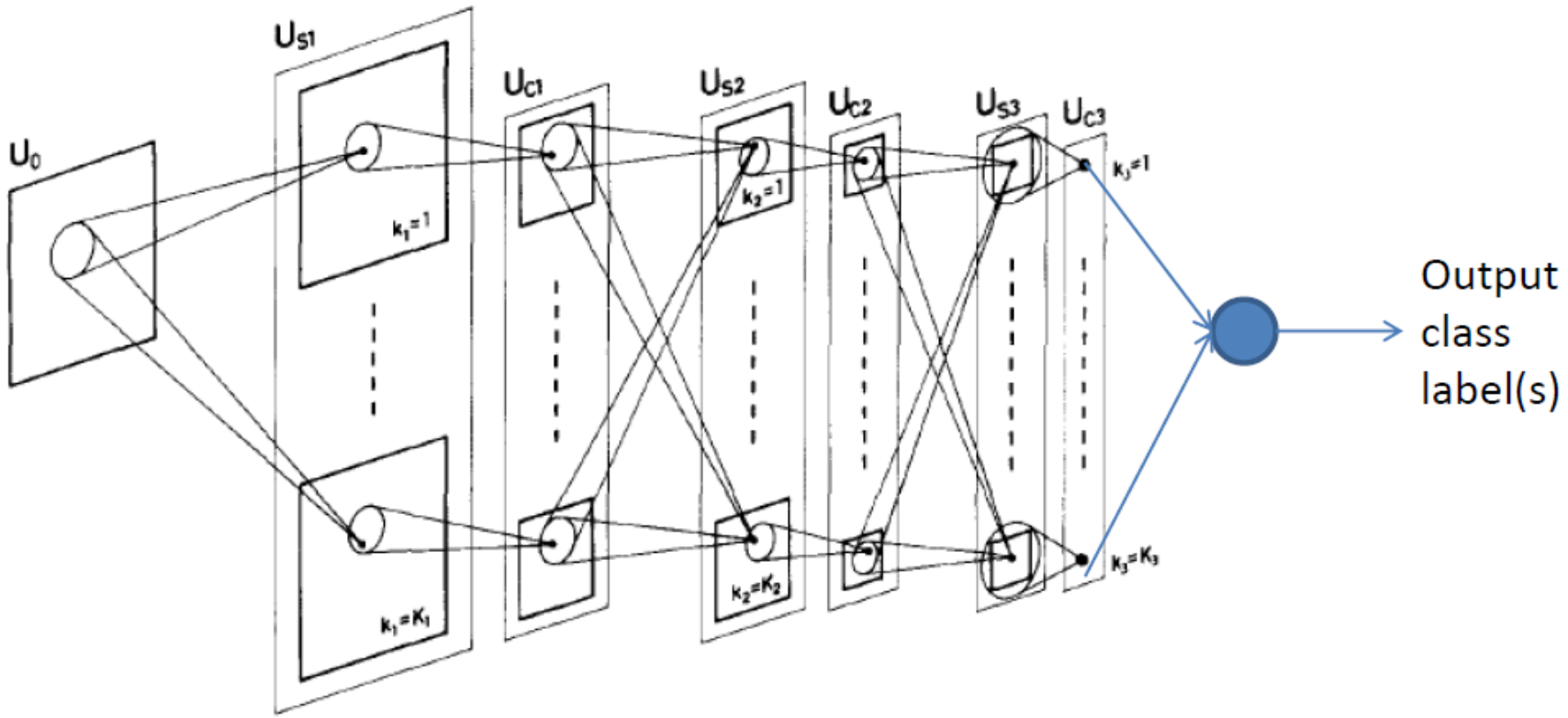
- Add an extra decision layer after the final C layer
  - Produces a class-label output
- We now have a fully feed forward MLP with shared parameters
  - All the S-cells within an S-plane have the same weights
- Simple backpropagation can now train the S-cell weights in every plane of every layer
  - C-cells are not updated

Scanning vs. multiple filters



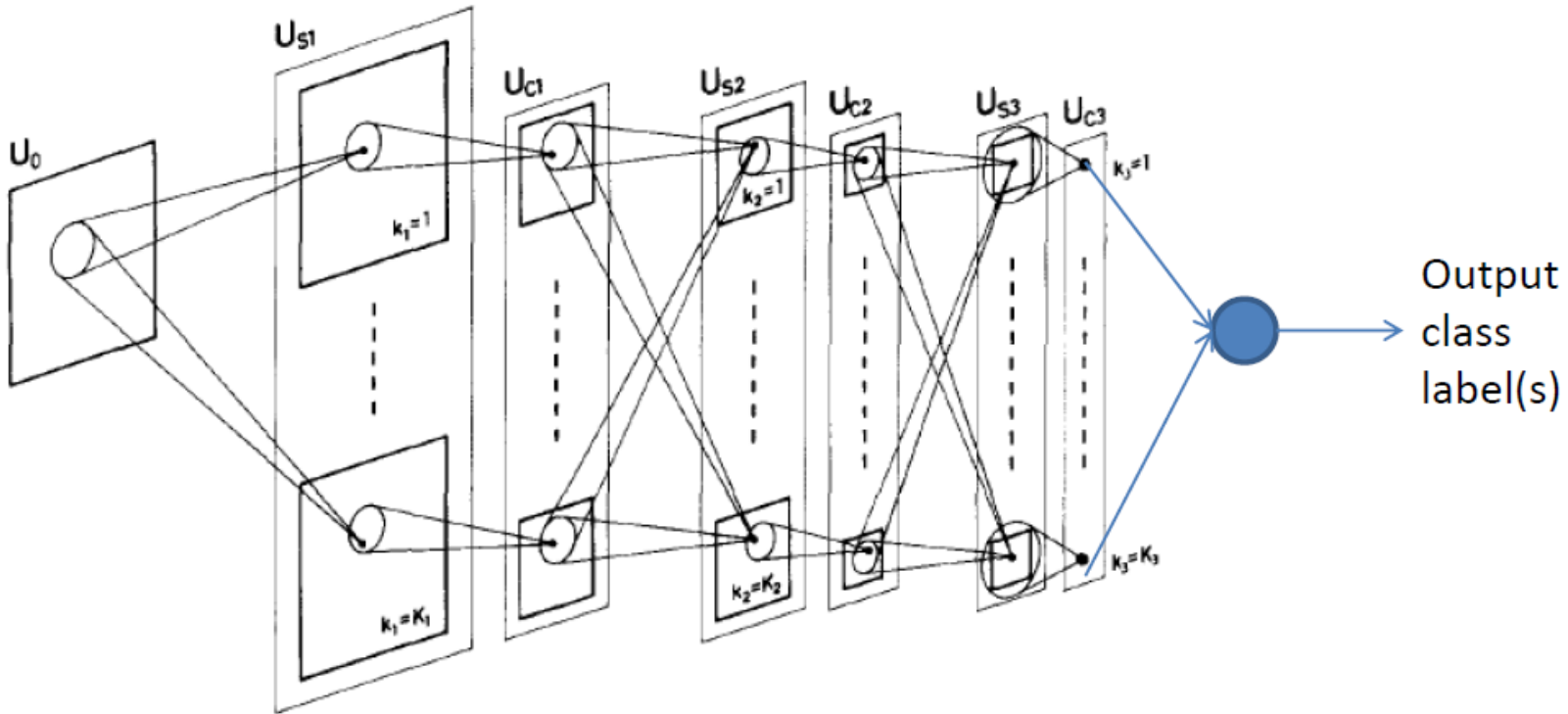
- **Note**: The original Neocognitron actually uses many identical copies of a neuron in each S and C plane

Supervising the neocognitron



- The Math
  - Assuming *square* receptive fields, rather than elliptical ones
  - Receptive field of S cells in lth layer is $K_l \times K_l$
  - Receptive field of C cells in lth layer is $L_l \times L_l$

Supervising the neocognitron

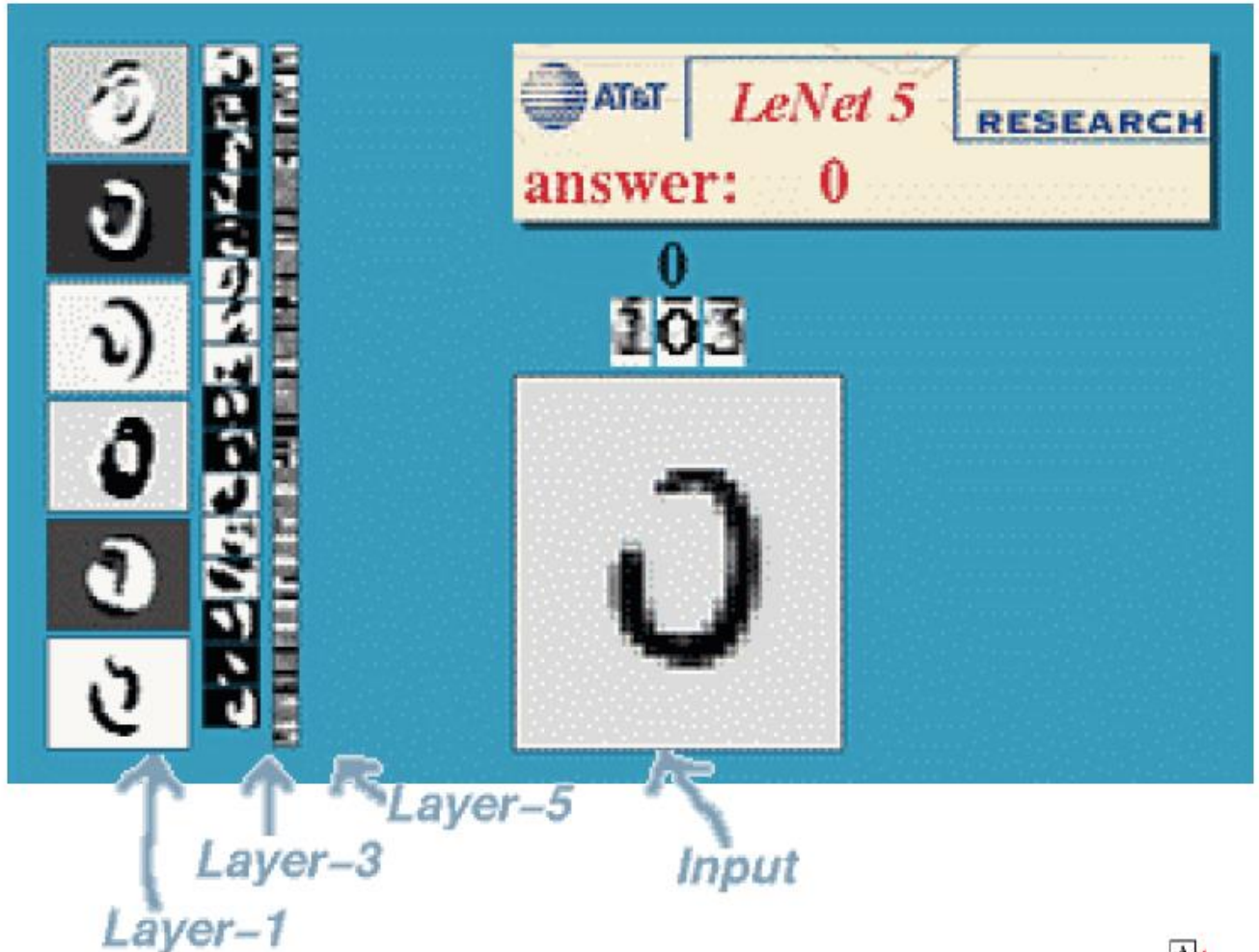


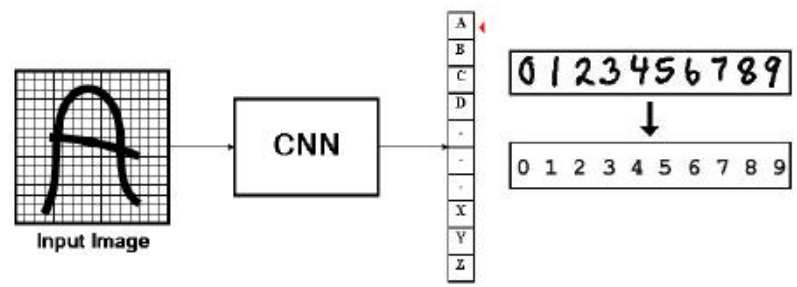
$$U_{S,l,n}(i,j) = \sigma\left(\sum_{p}\sum_{k=1}^{K_l}\sum_{l=1}^{K_l} w_{S,l,n}(p,k,l)\,U_{C,l-1,p}(i+l-1, j+k-1)\right)$$

$$U_{C,l,n}(i,j) = \max_{k\in(i,i+L_l),\,j\in(l,l+L_l)}\left(U_{S,l,n}(i,j)\right)$$

- This is, however, identical to "scanning" (convolving) with a single neuron/filter (what LeNet actually did)

Convolutional Neural Networks



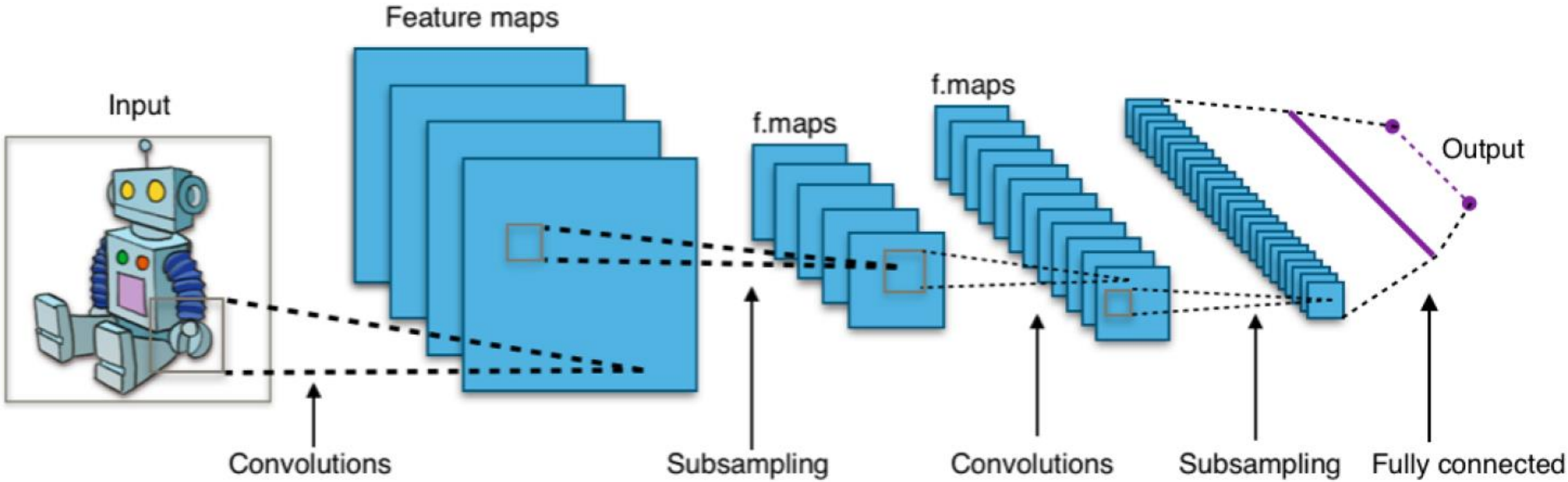


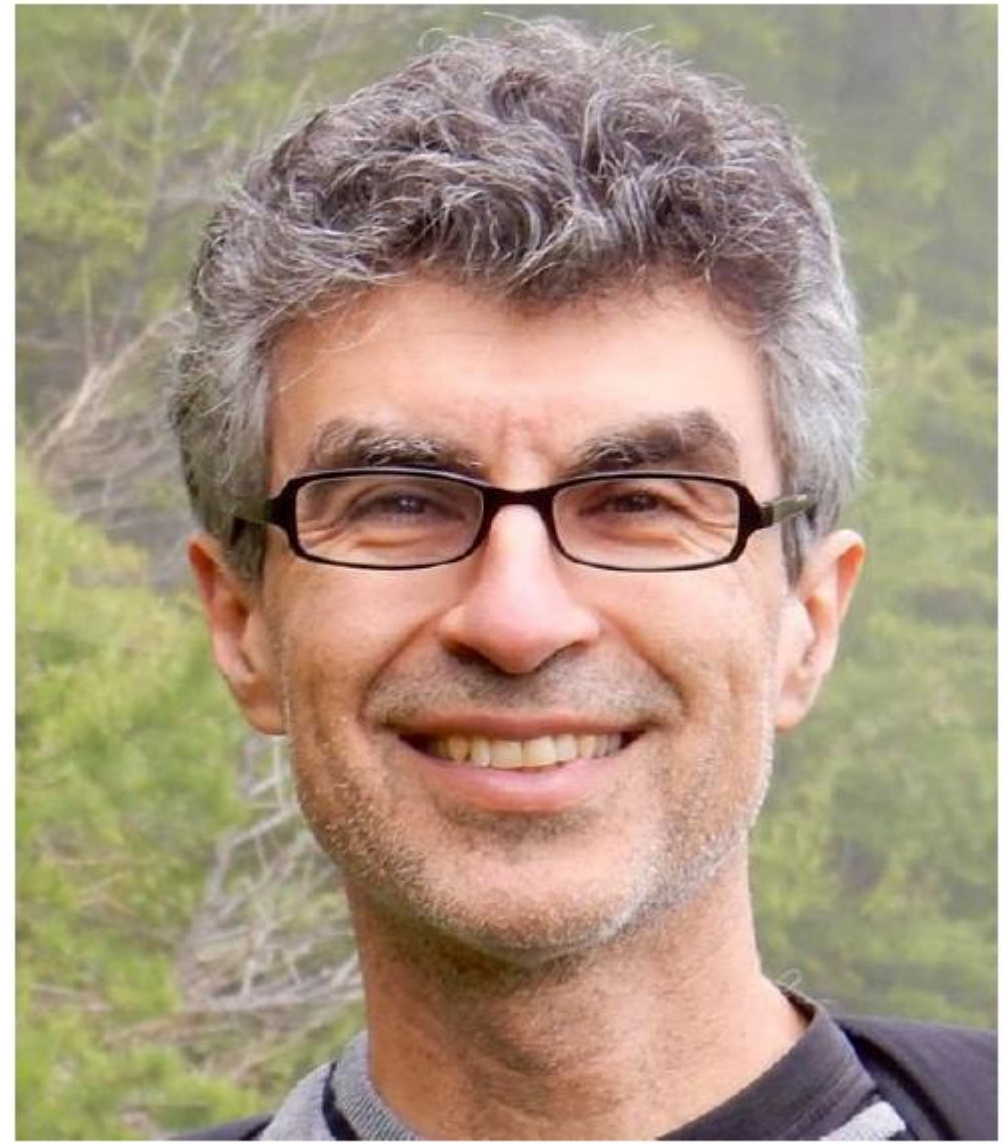
Story so far

- The mammalian visual cortex contains of S cells, which capture oriented visual patterns and C cells which perform a "majority" vote over groups of S cells for robustness to noise and positional jitter

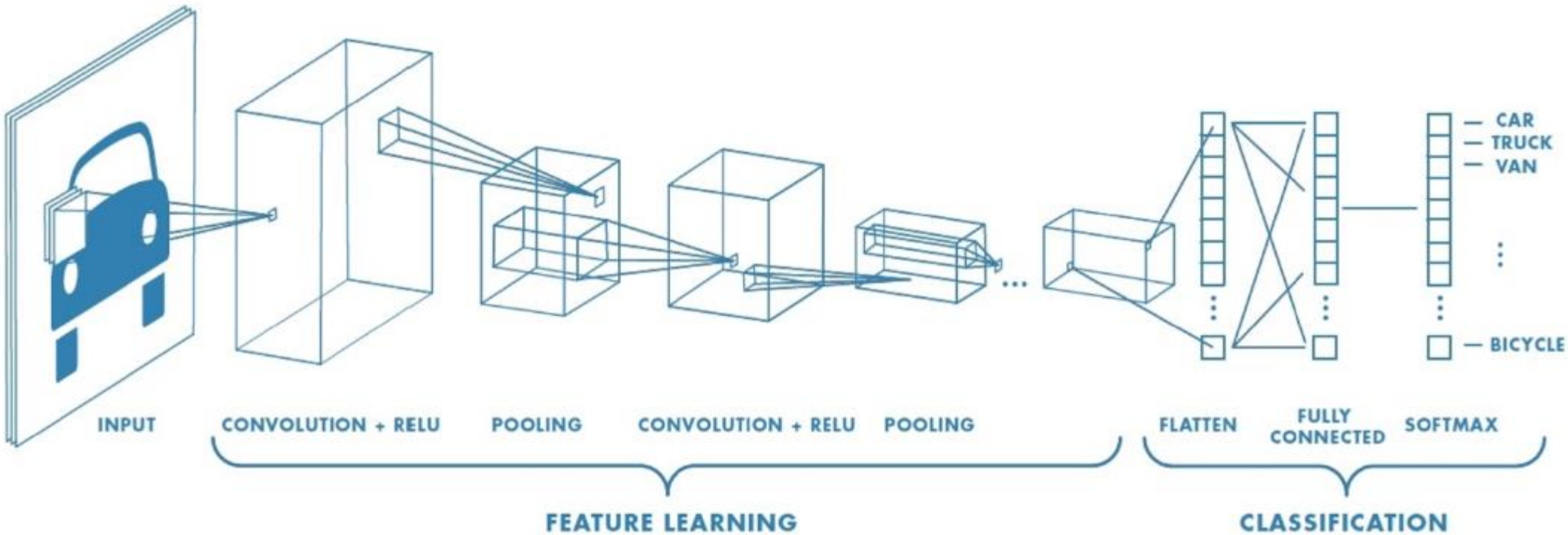
- The neocognitron emulates this behavior with planar banks of S and C cells with identical response, to enable shift invariance
  - Only S cells are learned
  - C cells perform the equivalent of a max over groups of S cells for robustness
  - Unsupervised learning results in learning useful patterns

- LeCun's LeNet added external supervision to the neocognitron
  - S planes of cells with identical response are modelled by a scan (convolution) over image planes by a single neuron
  - C planes are emulated by cells that perform a max over groups of S cells
    - Reducing the size of the S planes
  - Giving us a "Convolutional Neural Network"


Input
Feature maps
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected



Yann LeCun and Yoshua Bengio

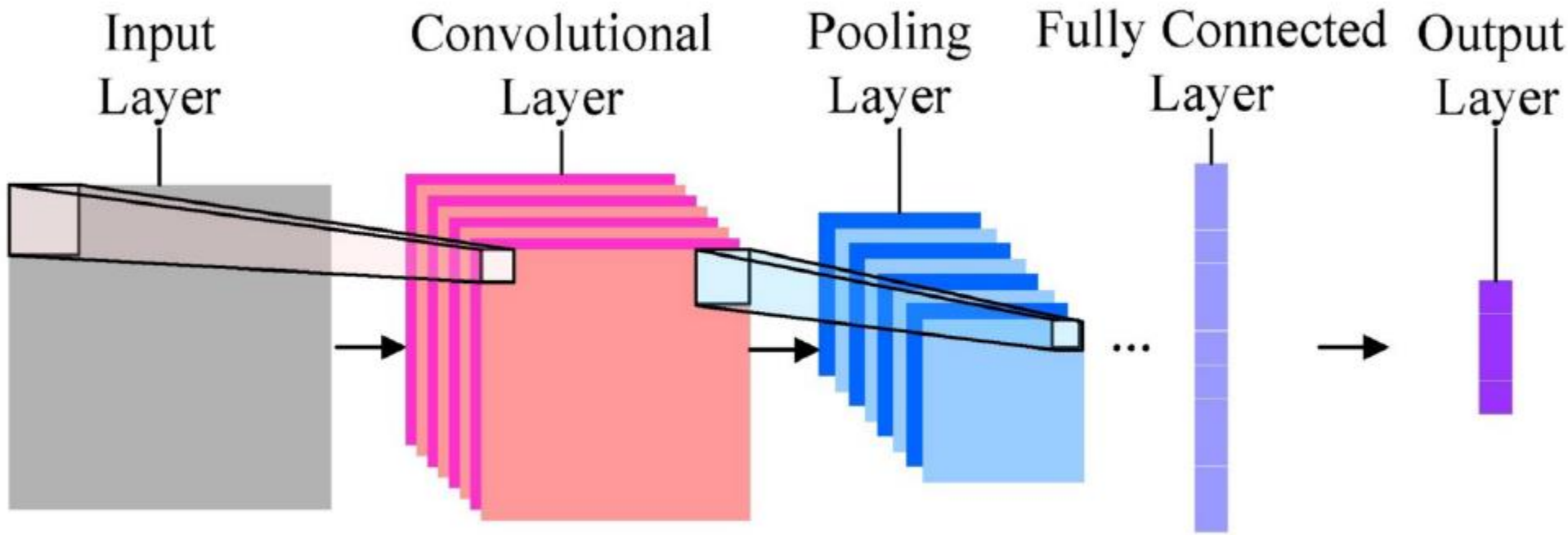


The basic layers of CNN.

The CNN is a combination of two basic building blocks:

1. **The Convolution Block** — Consists of the Convolution Layer and the Pooling Layer. This layer forms the essential component of *Feature-Extraction*

2. **The Fully Connected Block** — Consists of a fully connected simple neural network architecture. This layer performs the task of *Classification* based on the input from the convolutional block.



Stacking the concepts under one roof

Steps

We give input an RGB image. It is generally a 2-D matrix defined for the 3 color channels. Let each channel be of size n X n. Thus, the input is a n X n X 3 dimension matrix.

We have a 3-D matrix, consisting of (say) ‘k’ no. of filters of a size (say f X f).

We perform PADDING on the image of say ‘p’ rows & columns. Thus, the input-matrix becomes (n+2*p) X (n+2*p) X 3 dimensions.

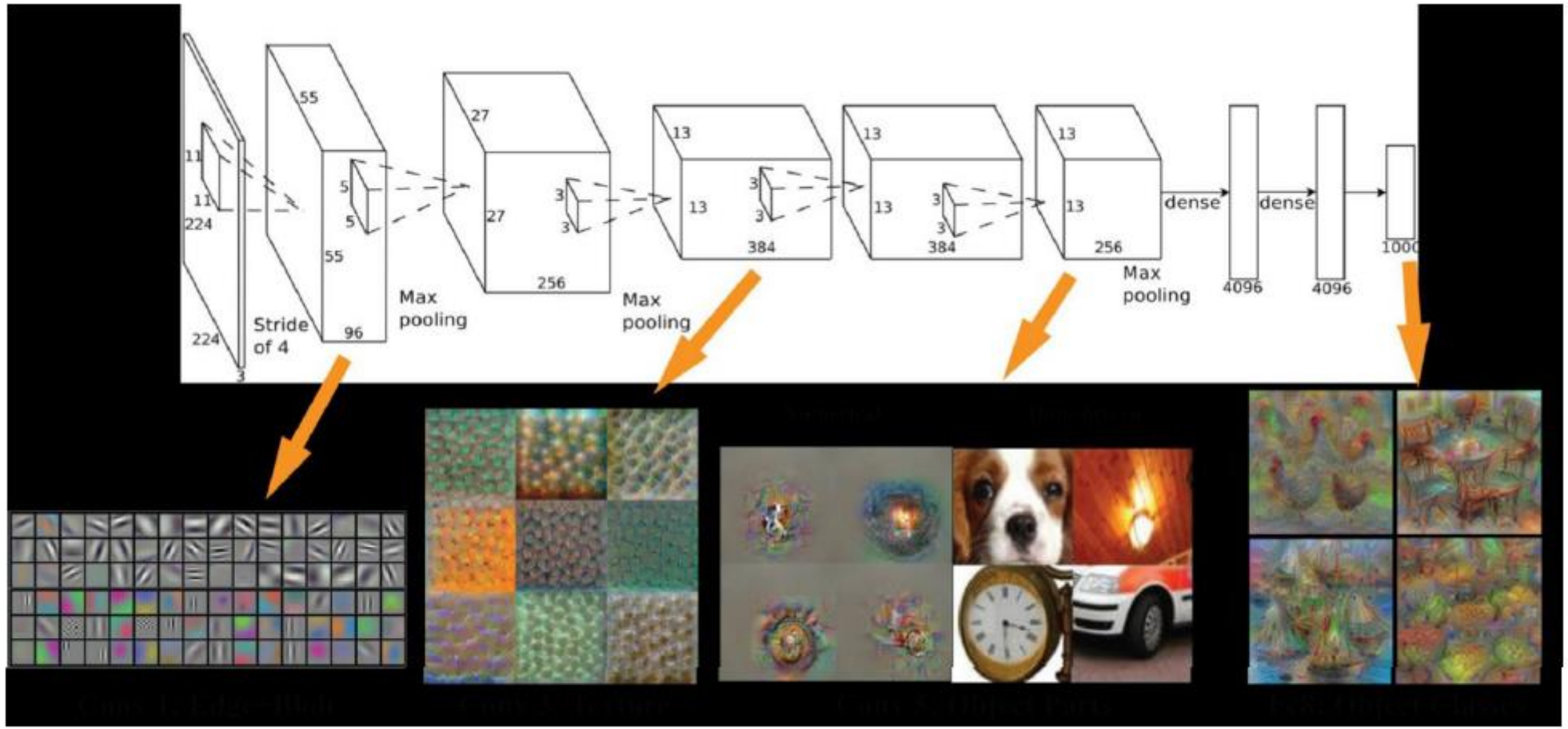
Next, we perform the strided convolution operation of the filter-matrices on the inputimage-matrices, as described before, using a stride of say ‘s’. Thus, the output matrix becomes.

We perform POOLING over the output-matrix of each layer. The dimension of the output-matrices depends of the size of the pooling-filter and the stride length we have defined.

We perform the same operation from step 3–5, nearly three time.

On receiving the output-matrices of some dimension say a X b X l , we flatten theoutput into a 1-D array, i.e., we arrange all the values from the matrices sequentially inan array which forms the input matrix for the Fully — Connected Neural Network.

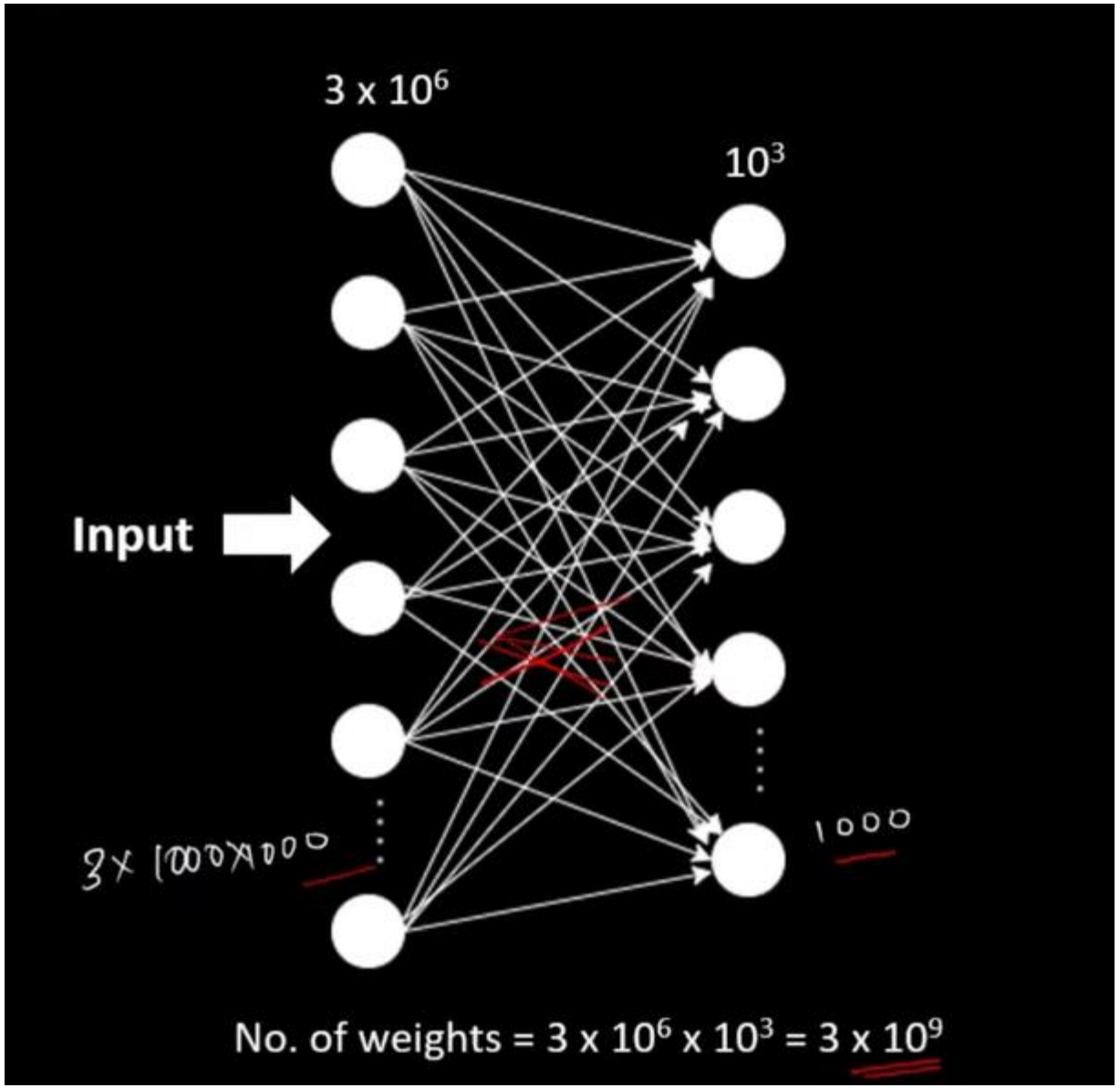
This neural network performs the desired calculation and gives the result.



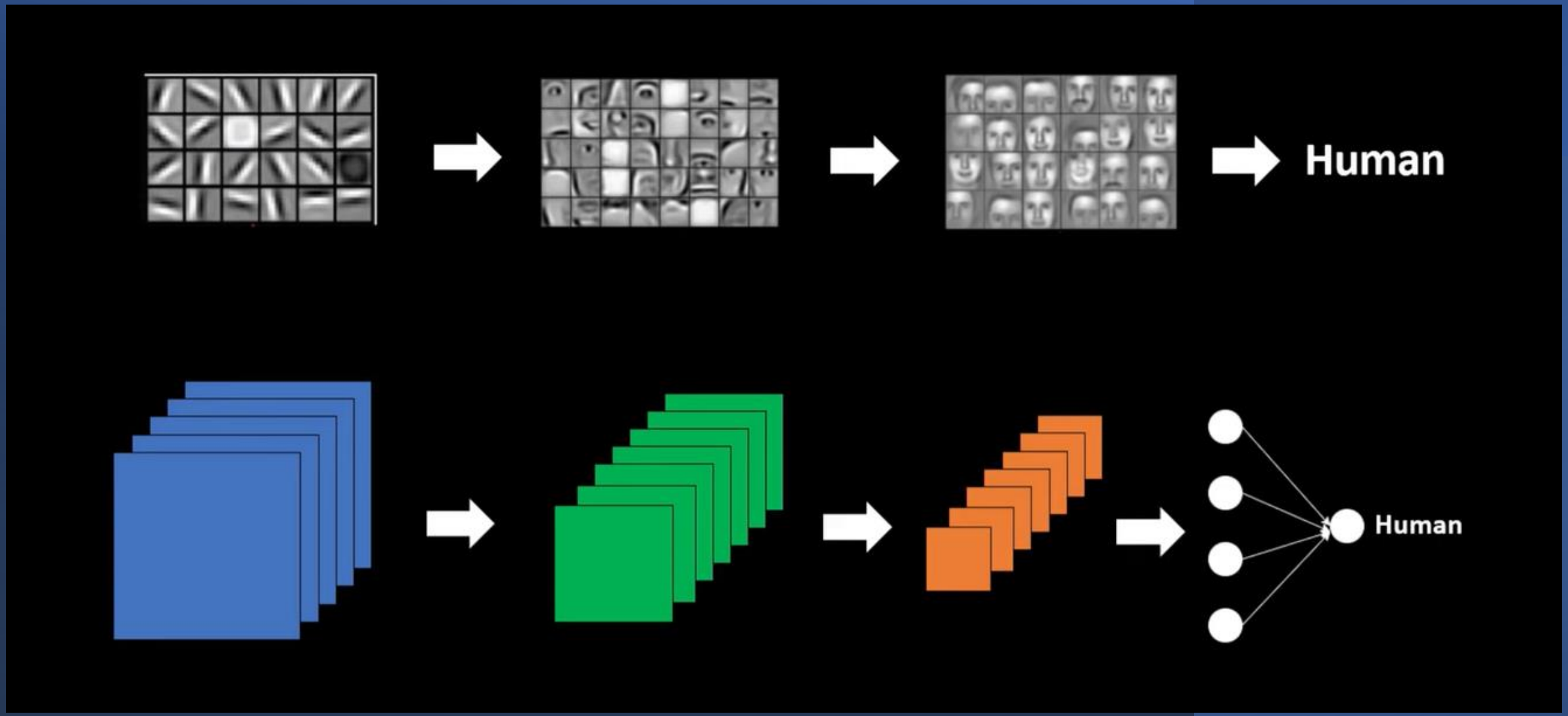
The visualization obtained at each layers in all the levels.

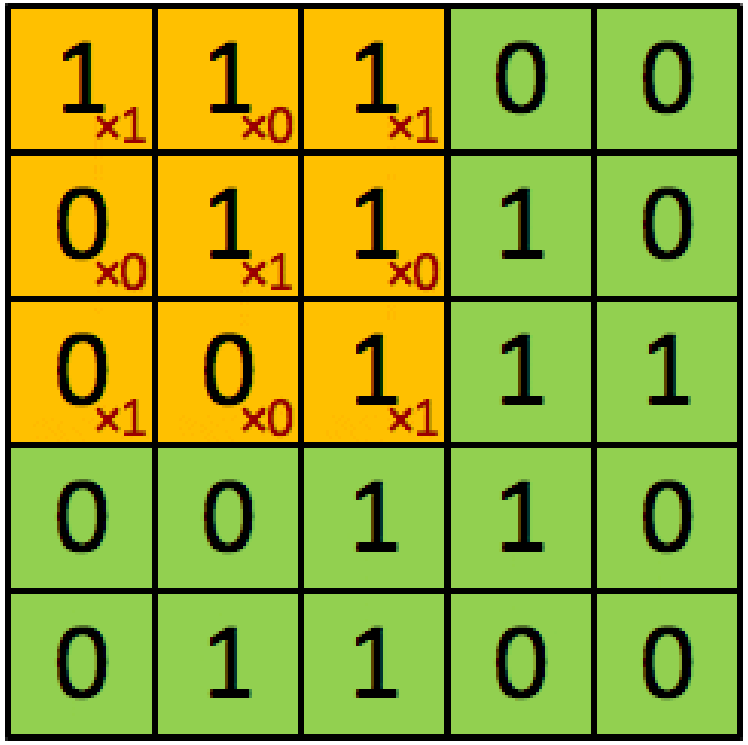
WHY USE IT AT ALL?

- a color image of size 256 X 256 X 3(for each color channel), the size of the input layer of a simple neural network would be 196,608 neurons, each connected to a large no. of neurons in the hidden layers. This increases the size of the stored data and the overall computation cost.
- Here you can see an image of 1000x1000

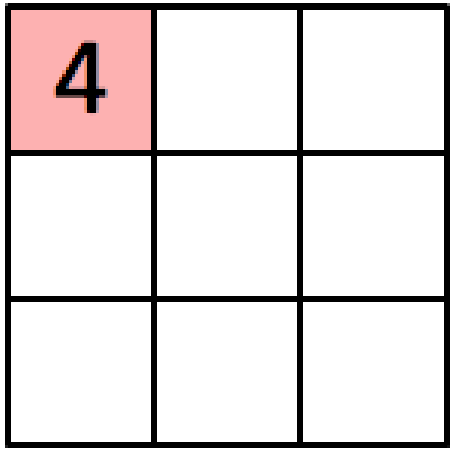


Deep Neural Networks
Scanning for patterns
(aka convolutional networks)


Human
Human



Image



Convolved Feature

1*1	6 0	9 -1	10	2	8
2*1	5 0	1 -1	8	4	2
3*1	7 0	4 -1	9	10	3
9	8	3	6	7	9
8	0	9	4	7	2
9	10	12	6	9	8

=

-8			

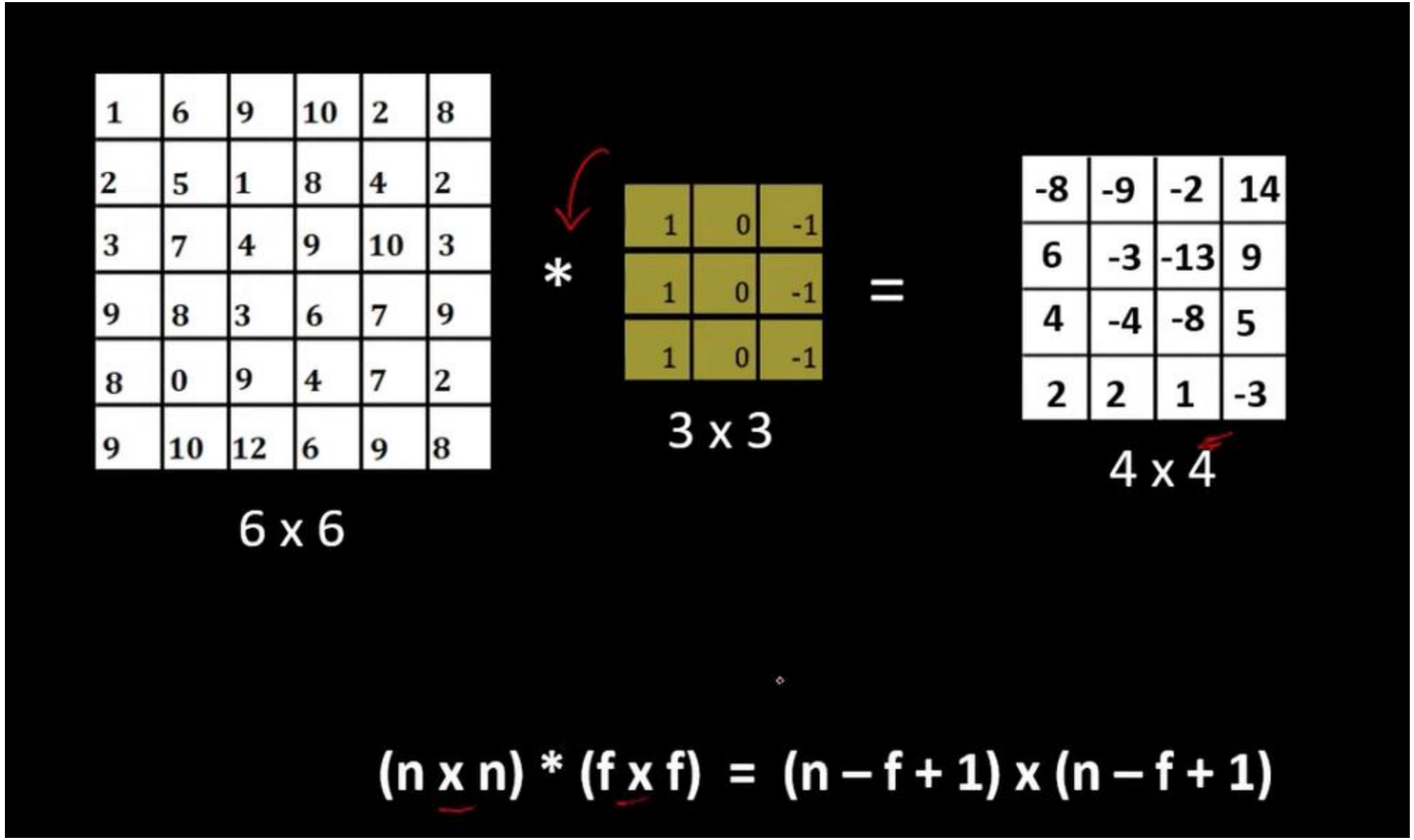
1*1 + 2*1 + 3*1 + 6*0 + 5*0 + 7*0 + 9*(-1) + 1*(-1) + 4 *(-1) = -8

1	6	9 1	10 0	2 -1	8
2	5	1 1	8 0	4 -1	2
3	7	4 1	9 0	10 -1	3
9	8	3	6	7	9
8	0	9	4	7	2
9	10	12	6	9	8

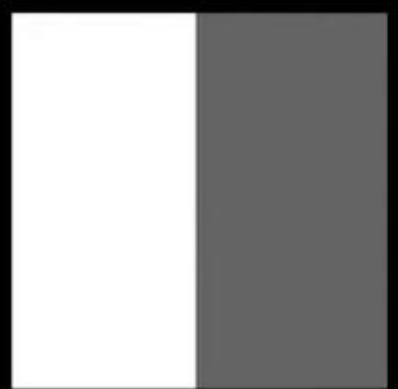
=

-8	-9	-2	

9*1 + 1*1 + 4*1 + 10*0 + 8*0 + 9*0 + 2*(-1) + 4*(-1) + 10*(-1) = -2


1 6 9 10 2 8
2 5 1 8 4 2
3 7 4 9 10 3
9 8 3 6 7 9
8 0 9 4 7 2
9 10 12 6 9 8
6 x 6
*
1 0 -1
1 0 -1
1 0 -1
3 x 3
=
-8 -9 -2 14
6 -3 -13 9
4 -4 -8 5
2 2 1 -3
4 x 4
(n x n) * (f x f) = (n – f + 1) x (n – f + 1)

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



1*1 + 1*1 + 1*1
+ 1*0 + 1*0 + 1*0
+ 1*(-1) + 1*(-1) + 1*(-1) = 0

1*1 + 1*1 + 1*1
+ 1*0 + 1*0 + 1*0
+ 0*(-1) + 0*(-1) + 0*(-1) = 3

1*1 + 1*1 + 1*1
+ 0*0 + 0*0 + 0*0
+ 0*(-1) + 0*(-1) + 0*(-1) = 3

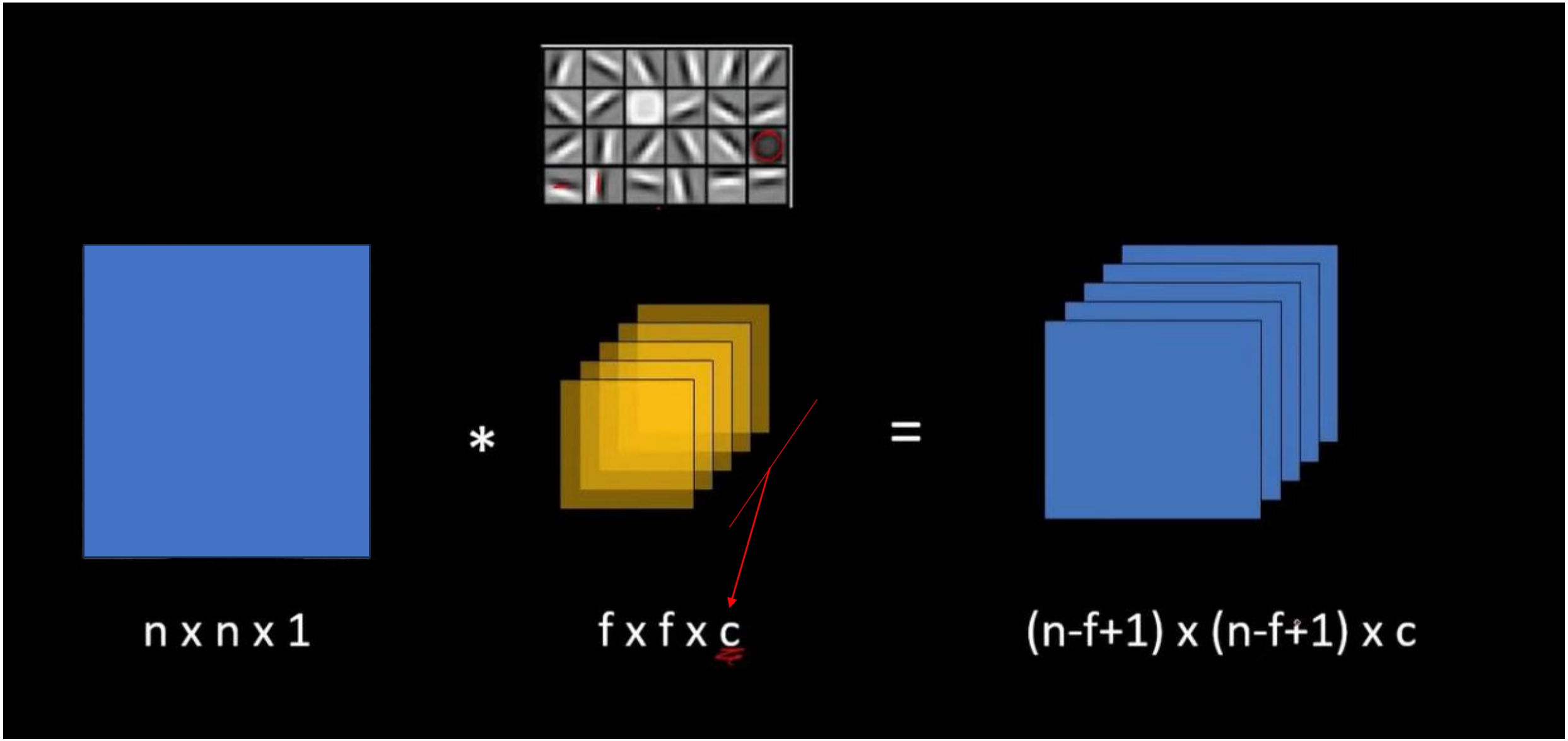
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0

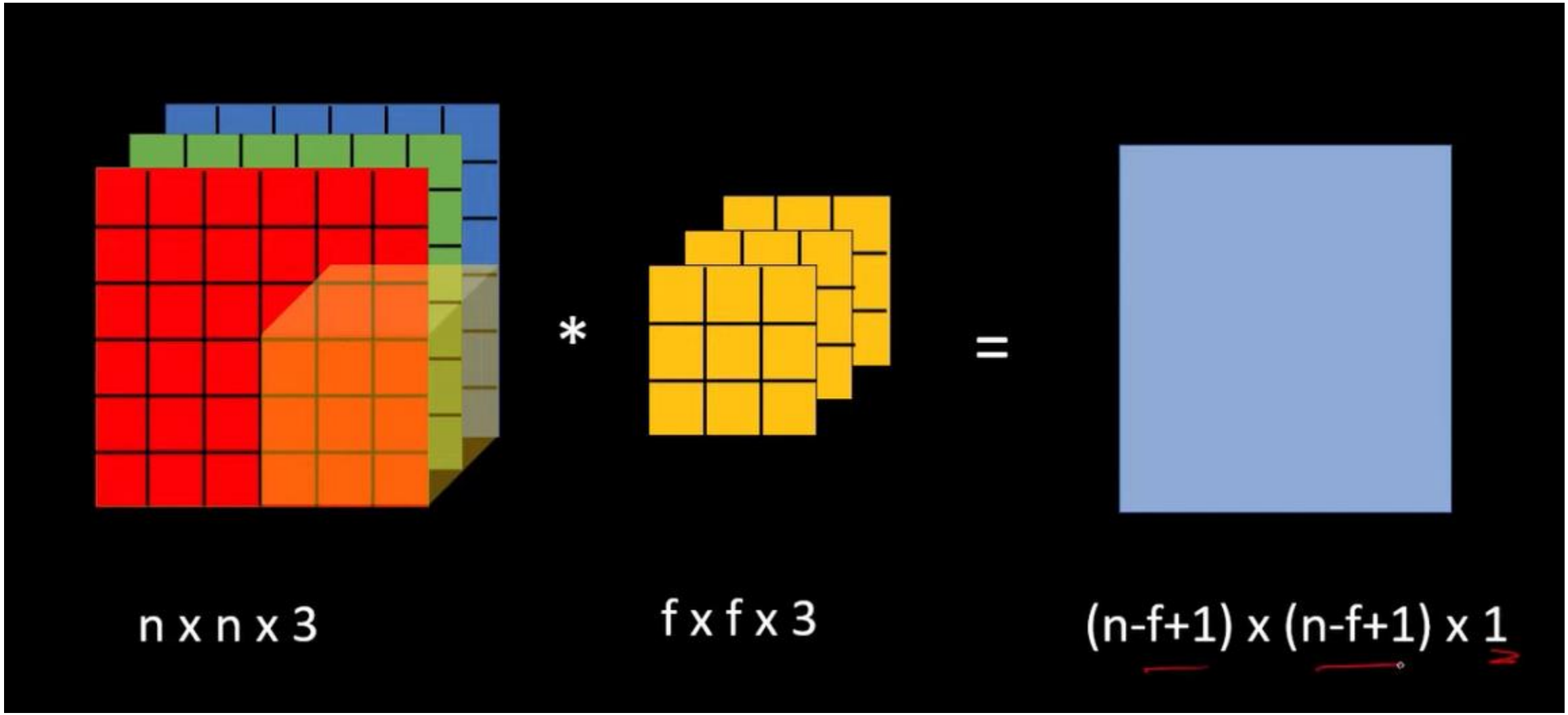
*

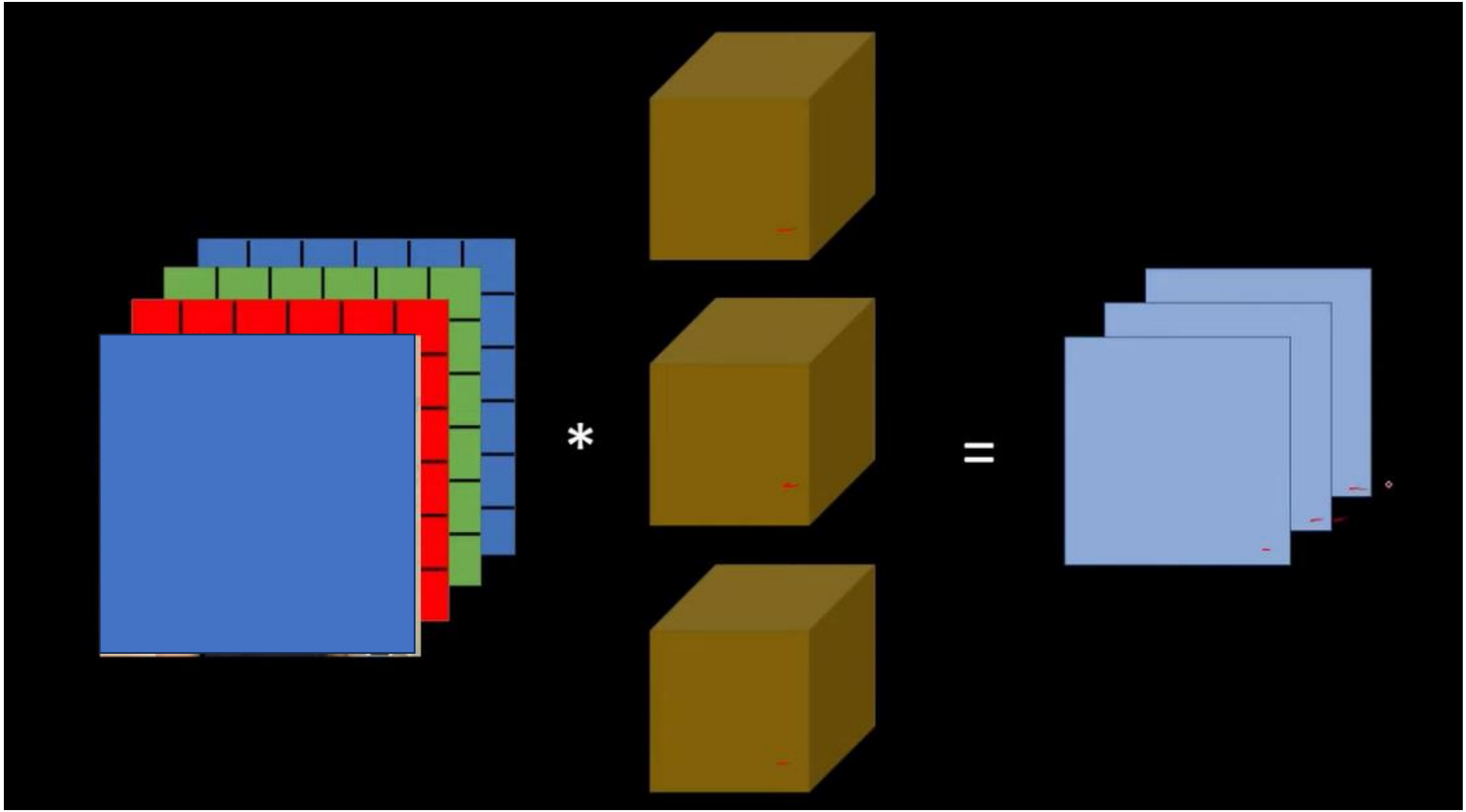
1	0	-1
1	0	-1
1	0	-1

=

0	3	3	0
0	3	3	0
0	3	3	0
0	3	3	0


n x n x 1
*
f x f x c
=
(n-f+1) x (n-f+1) x c


*
=
n x n x 3
f x f x 3
(n-f+1) x (n-f+1) x 1



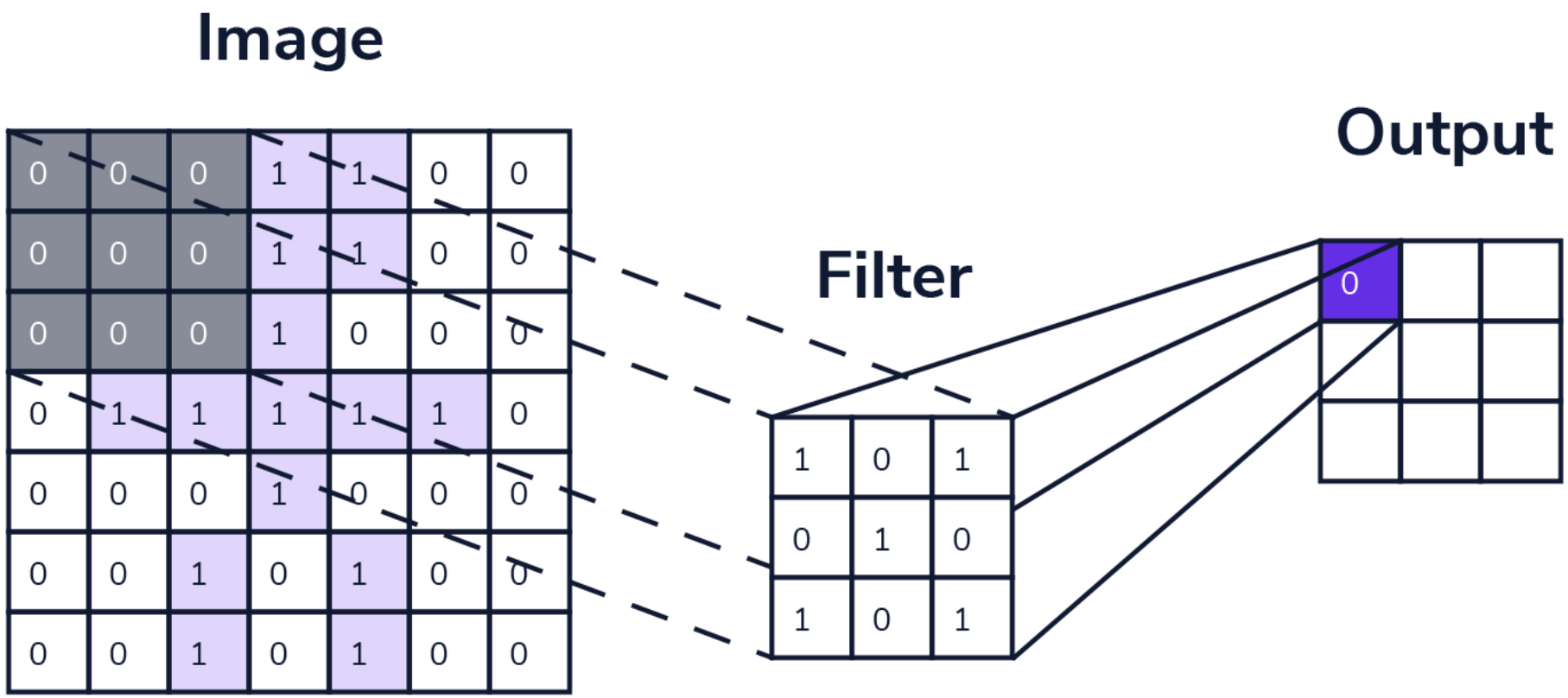
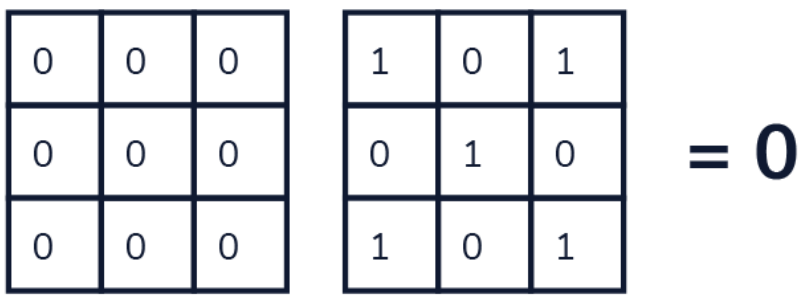
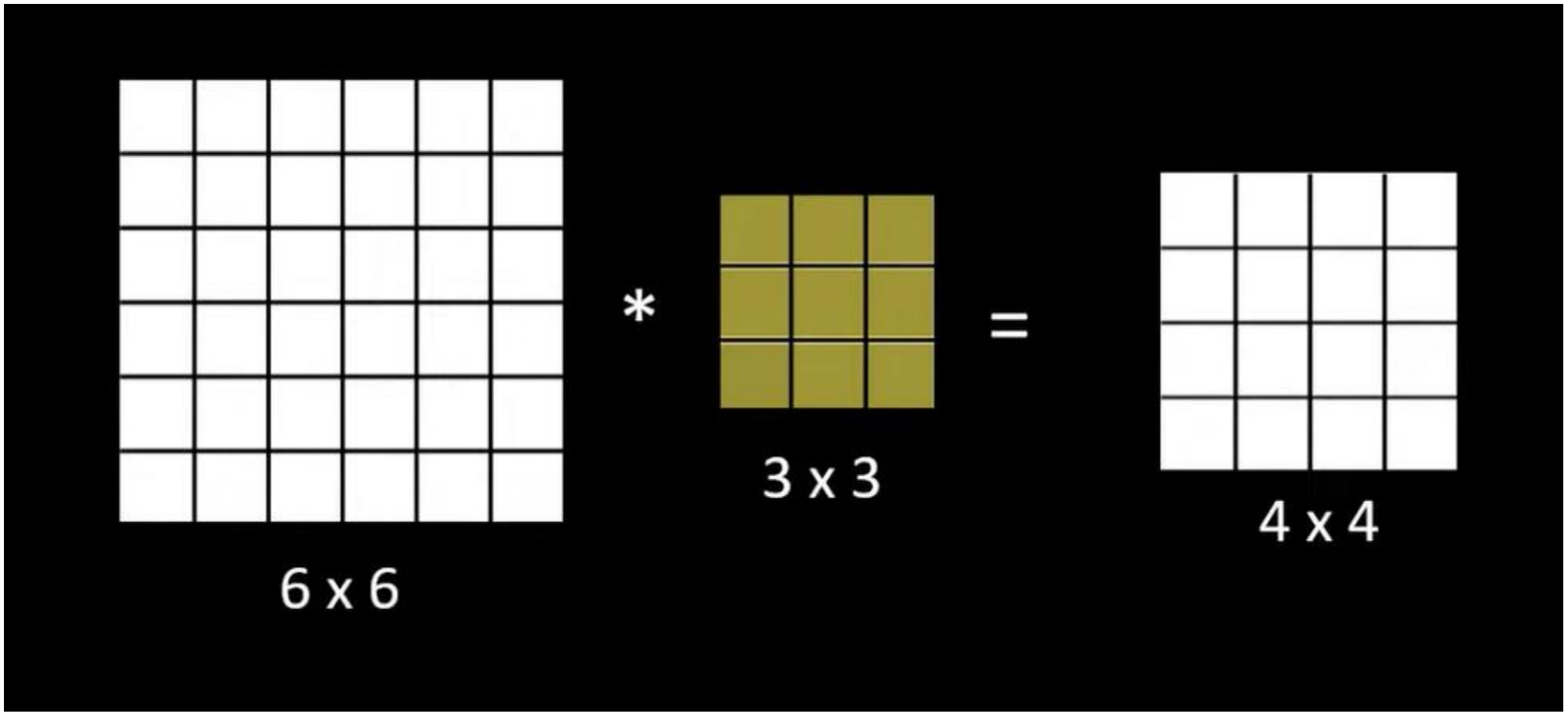


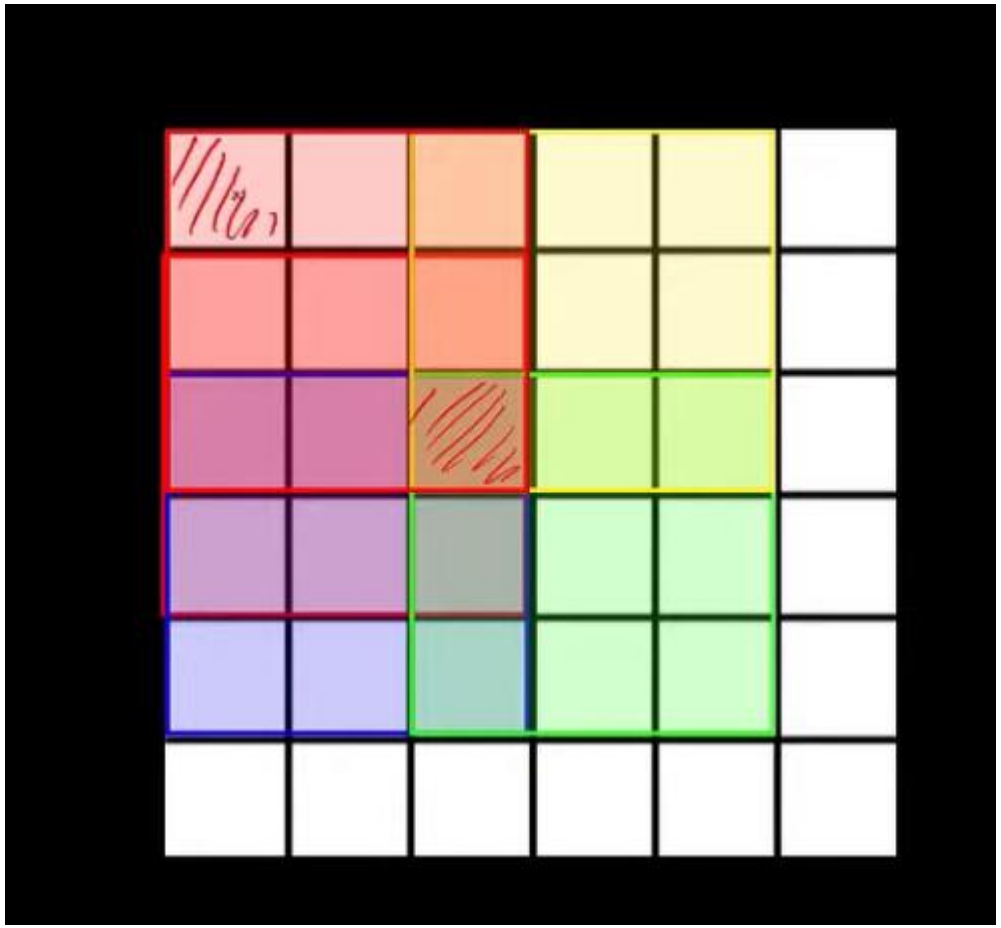
Image
Output
Filter
0 0 0 1 1 0 0
0 0 0 1 1 0 0
0 0 0 1 0 0 0
0 1 1 1 1 1 0
0 0 0 1 0 0 0
0 0 1 0 1 0 0
0 0 1 0 1 0 0
1 0 1
0 1 0
1 0 1
0

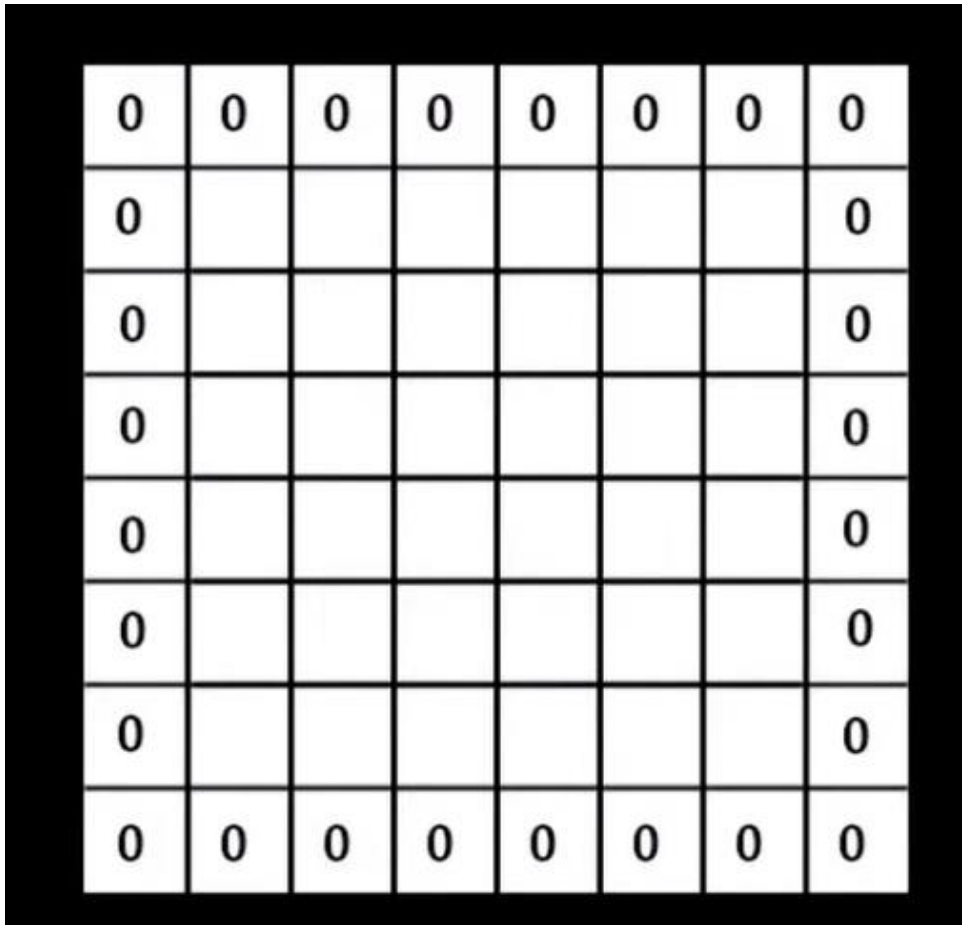
Stride = 2

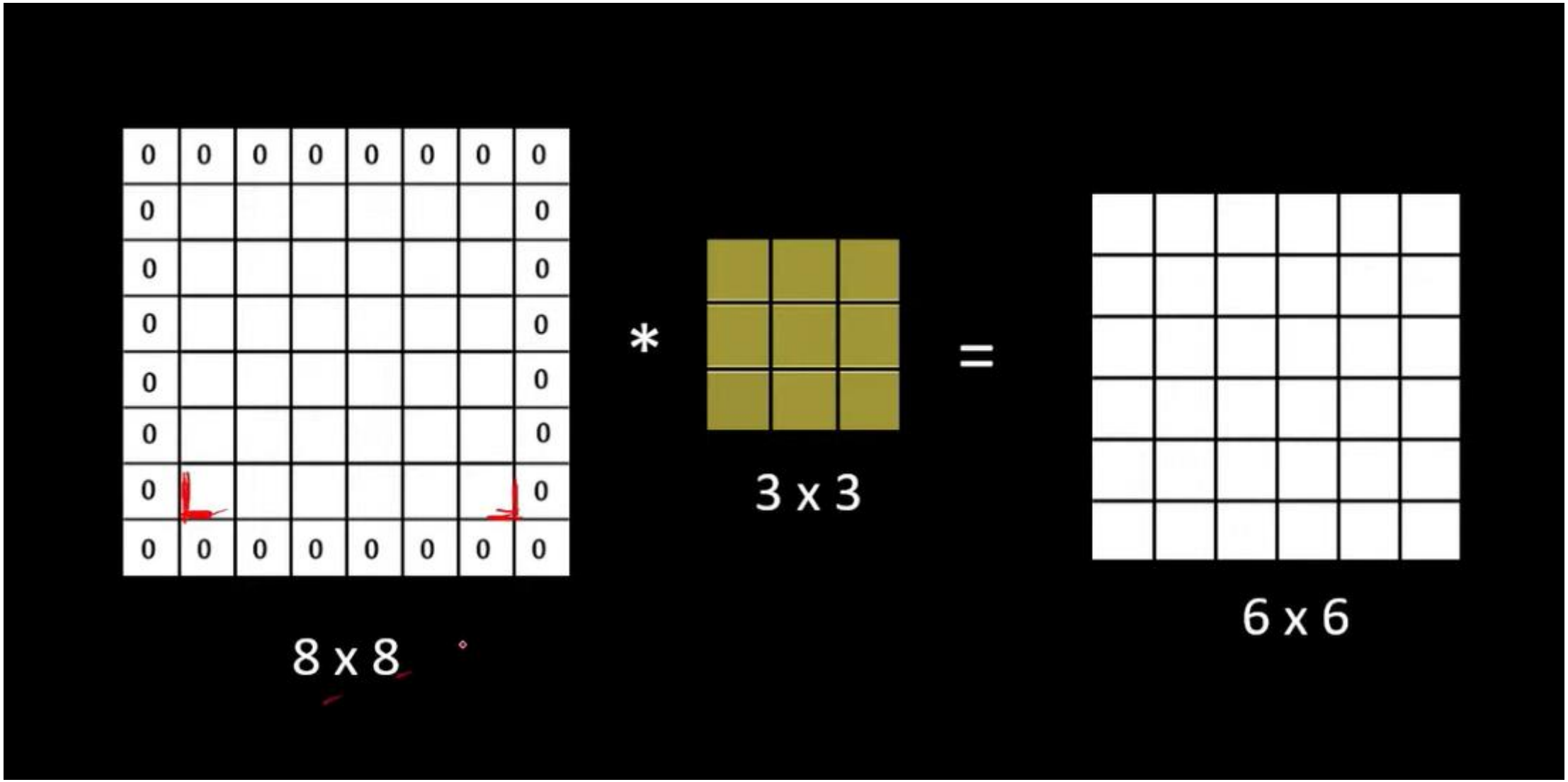

0 0 0
0 0 0
0 0 0
1 0 1
0 1 0
1 0 1
= 0

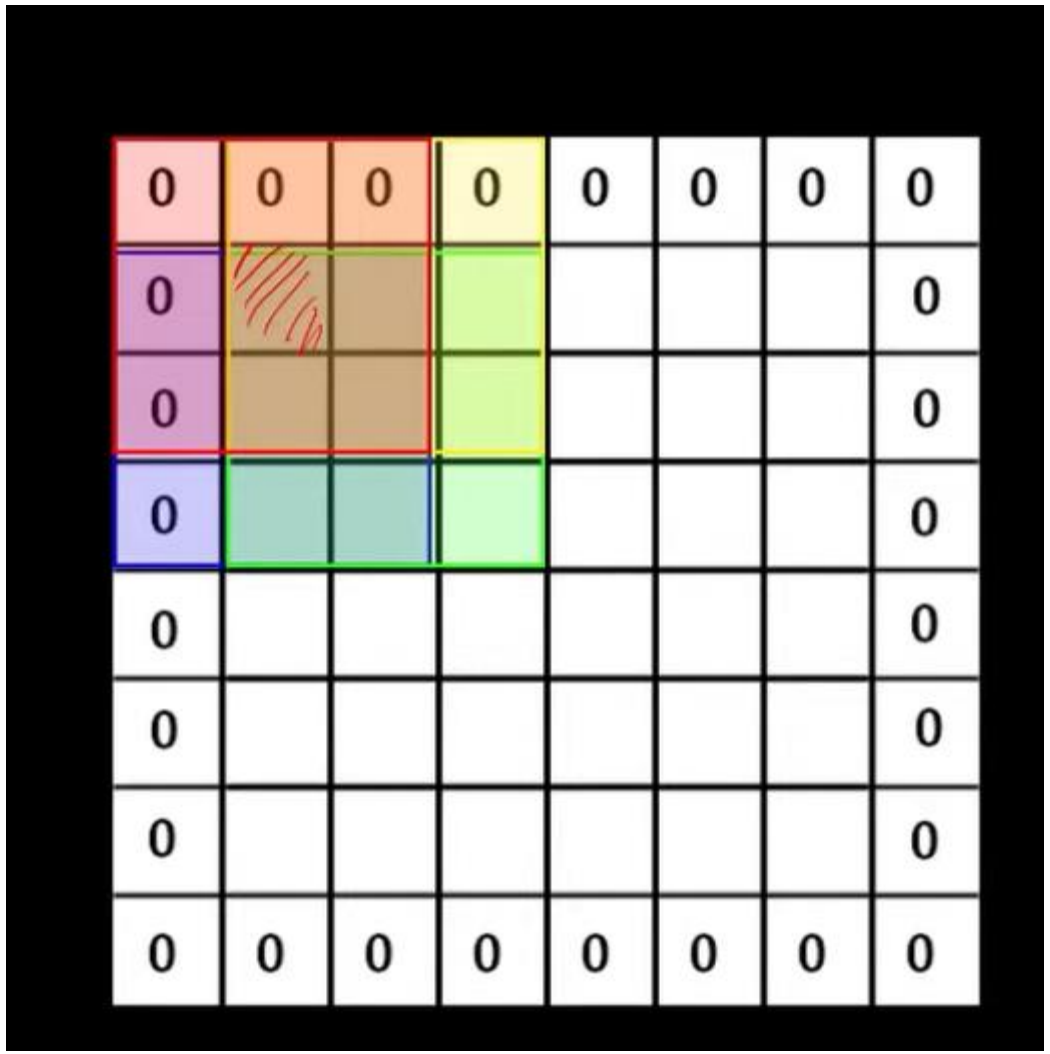

6 x 6
*
3 x 3
=
4 x 4

Padding






0 0 0 0 0 0 0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0 0 0 0 0 0 0
8 x 8
*
3 x 3
=
6 x 6


0 0 0 0 0 0 0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0
0 0 0 0 0 0 0 0

“VALID” and “SAME” convolution

1. Valid Convolution

   No padding

2. Same Convolution

“VALID” and “SAME” convolution

1. Valid Convolution

No padding

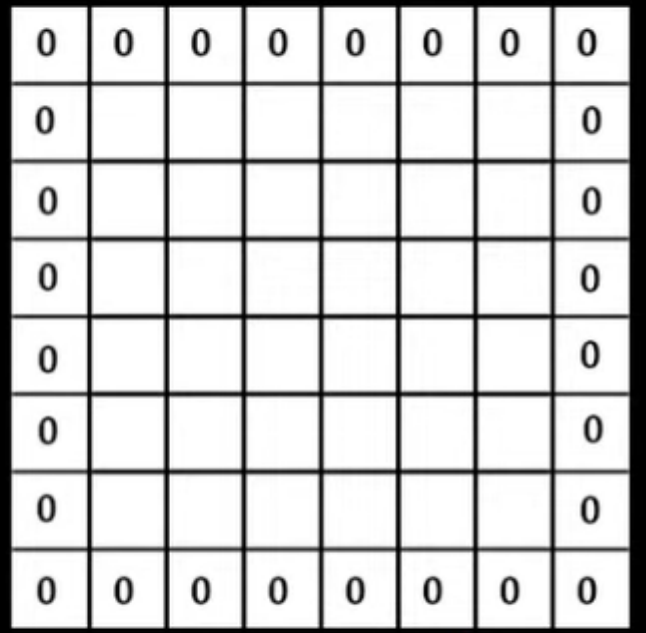
2. Same Convolution

$n' = n + 2p$

$(n' - f + 1) = n$

$(n + 2p - f + 1) = n$

$p = \left(\frac{f-1}{2}\right)$




```
tf.nn.conv2d(X, W1, strides = [1,1,1,1], padding = 'VALID')
```

Stride = 2

1	6	9	10	2	8	5
2	5	1	8	4	2	4
3	7	4	9	10	3	7
9	8	3	6	7	9	3
8	0	9	4	7	2	1
9	10	12	6	9	8	0

6×7

$n_1 \times n_2$

$f \times f$

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor \quad \text{stride} \quad \text{floor}$$

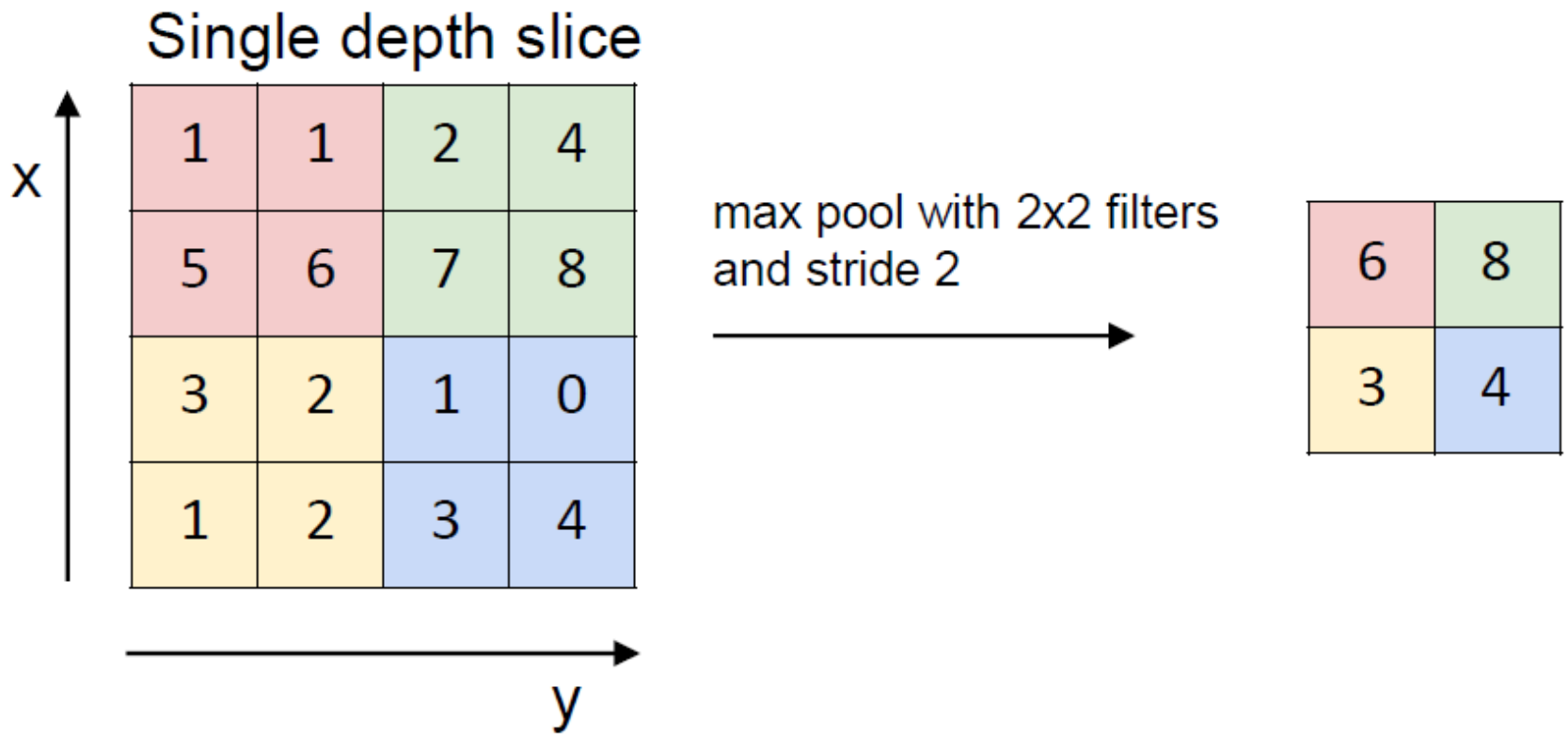
$\lfloor 2.7 \rfloor = 2$

$$\left\lfloor \frac{n_1 - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_2 - f}{s} + 1 \right\rfloor$$

$$\left\lfloor \frac{6-3}{2} + 1 \right\rfloor \times \left\lfloor \frac{7-3}{2} + 1 \right\rfloor = \lfloor 2.5 \rfloor \times \lfloor 3 \rfloor$$

$$= 2 \times 3$$

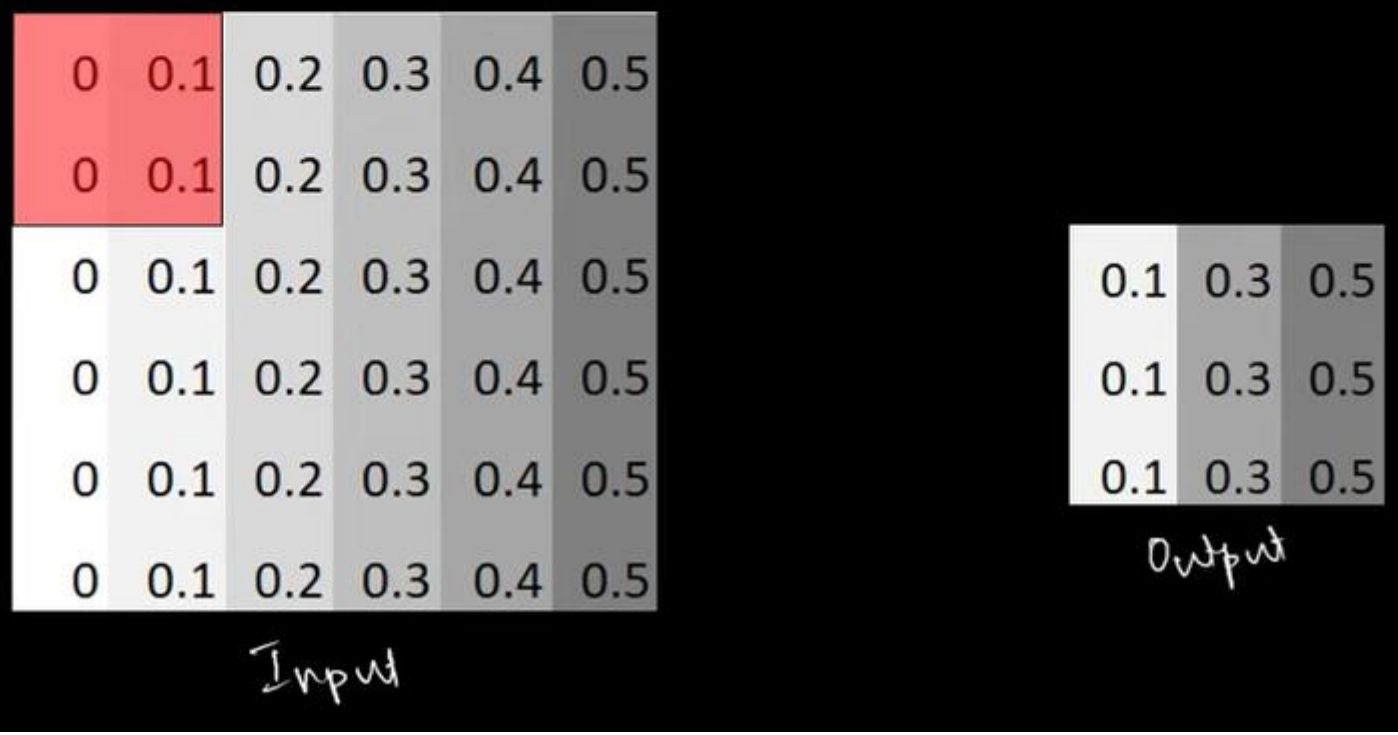
Max Pooling

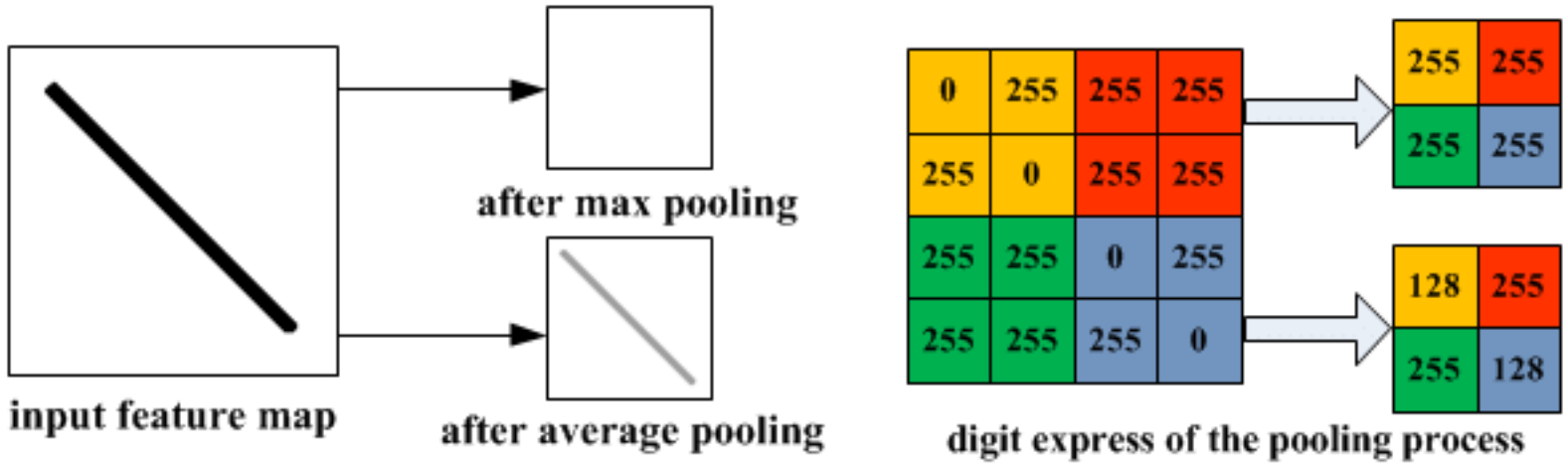

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Why do we need Max Pooling?

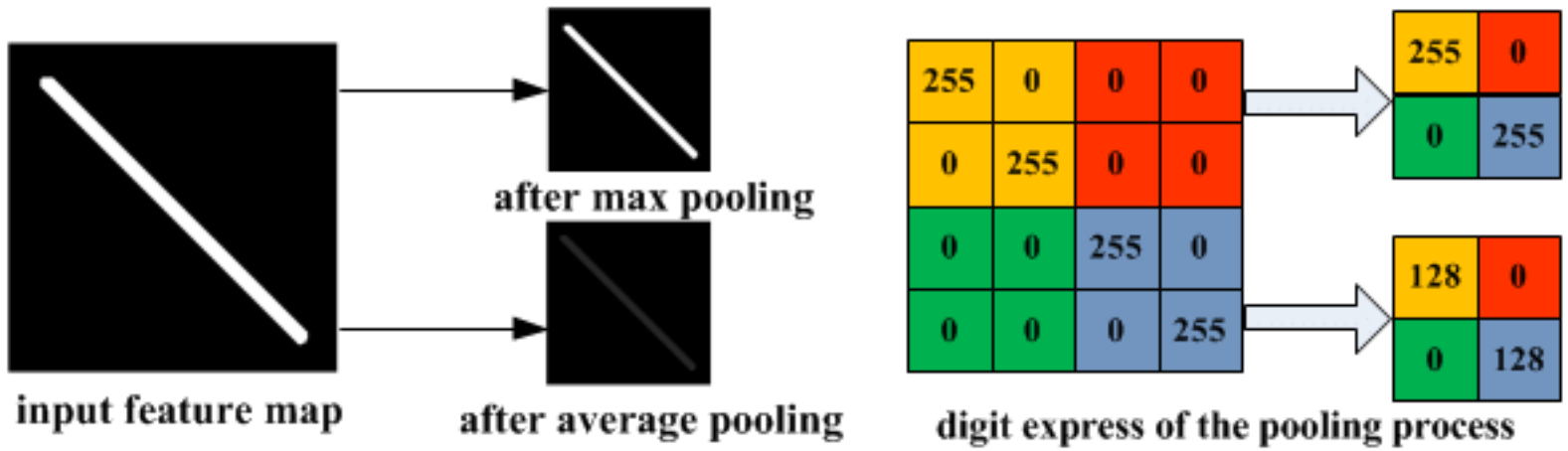
1. Reduce image size, thus reduce computational cost

2. Enhances Features



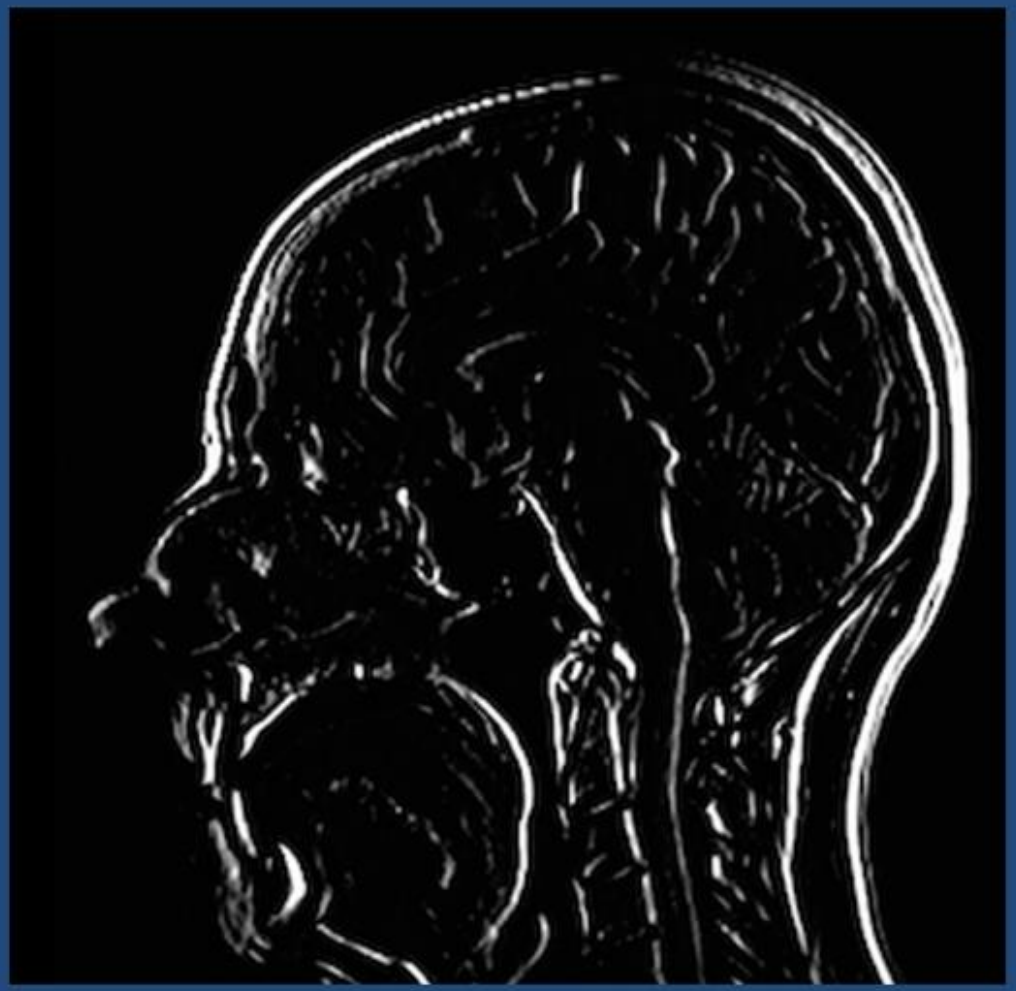


(a) Illustration of max pooling drawback



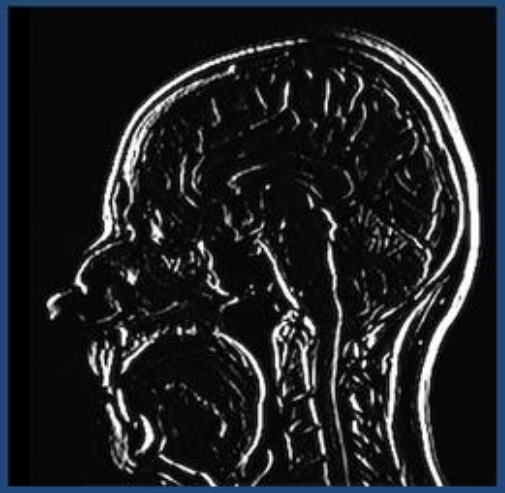
(b) Illustration of average pooling drawback

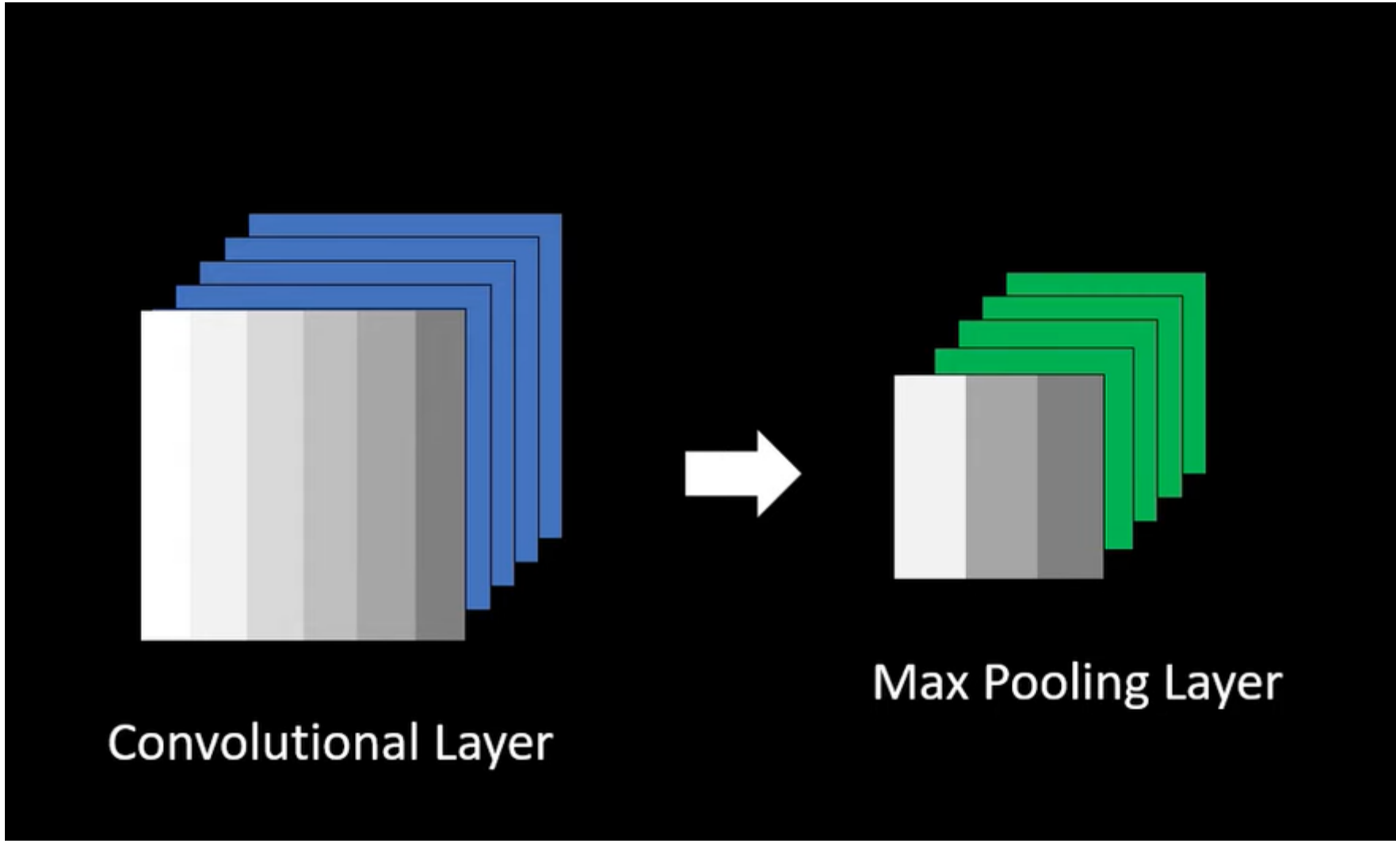
Rectified Feature Map



2x2 Max Pooling

2x2 Avg Pooling




Convolutional Layer
Max Pooling Layer

Summary

Why do we need Max Pooling?

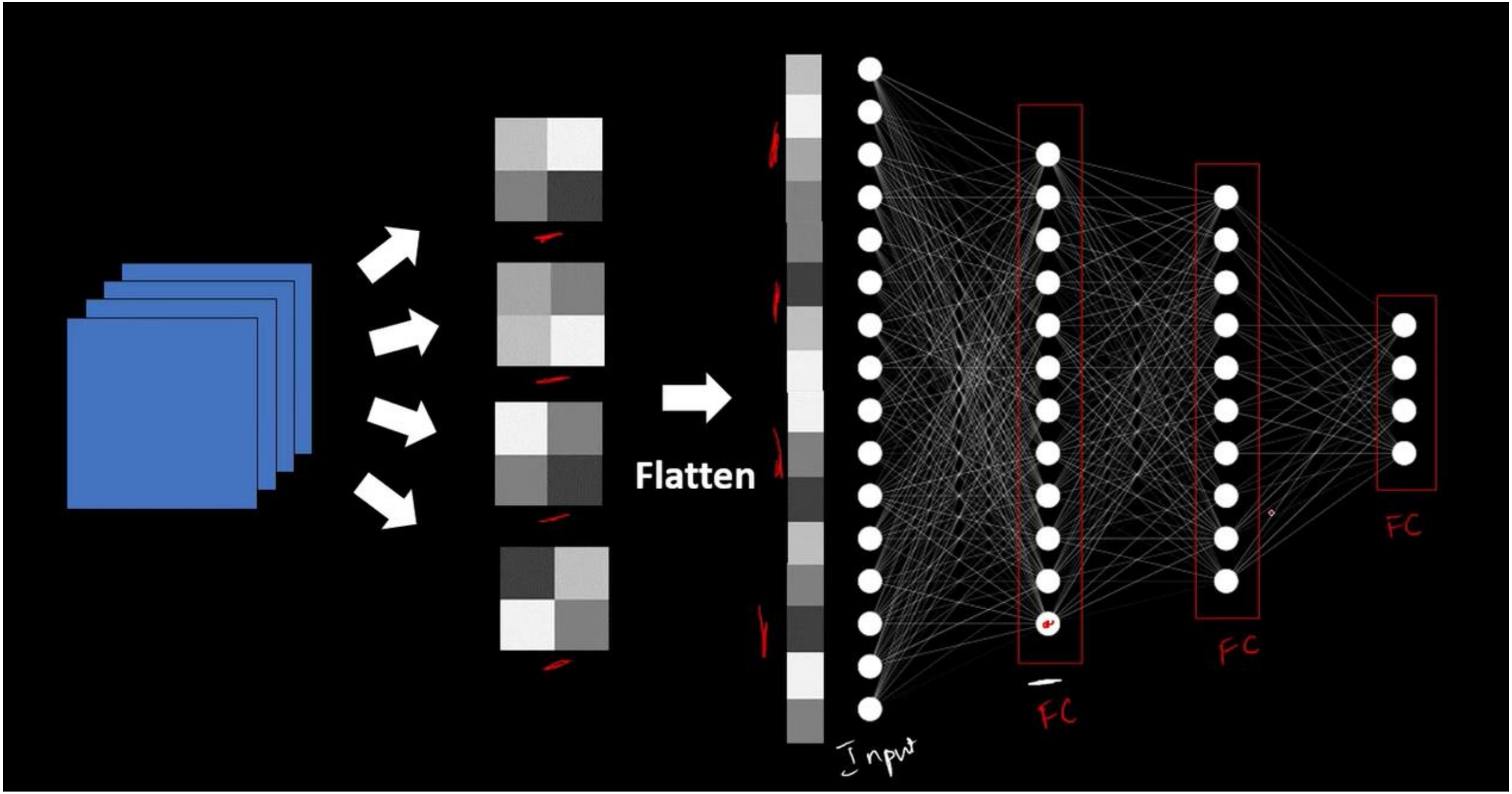
1. Reduce image size, thus reduce computational cost
2. Enhances the Features of the image

Where?

After Convolutional layer

Other points

1. No parameters involved, thus no training
2. Same number of channels in output as input


Flatten
Input
FC
FC
FC

Convolutional Layer

fxfx3

*

fxfx3

ReLU(+ b)

Convolutional Layer

fxfx3 * → ReLU(+ b1) →

fxfx3 * → ReLU(+ b2) →

fxfx3 * → ReLU(+ b3) →

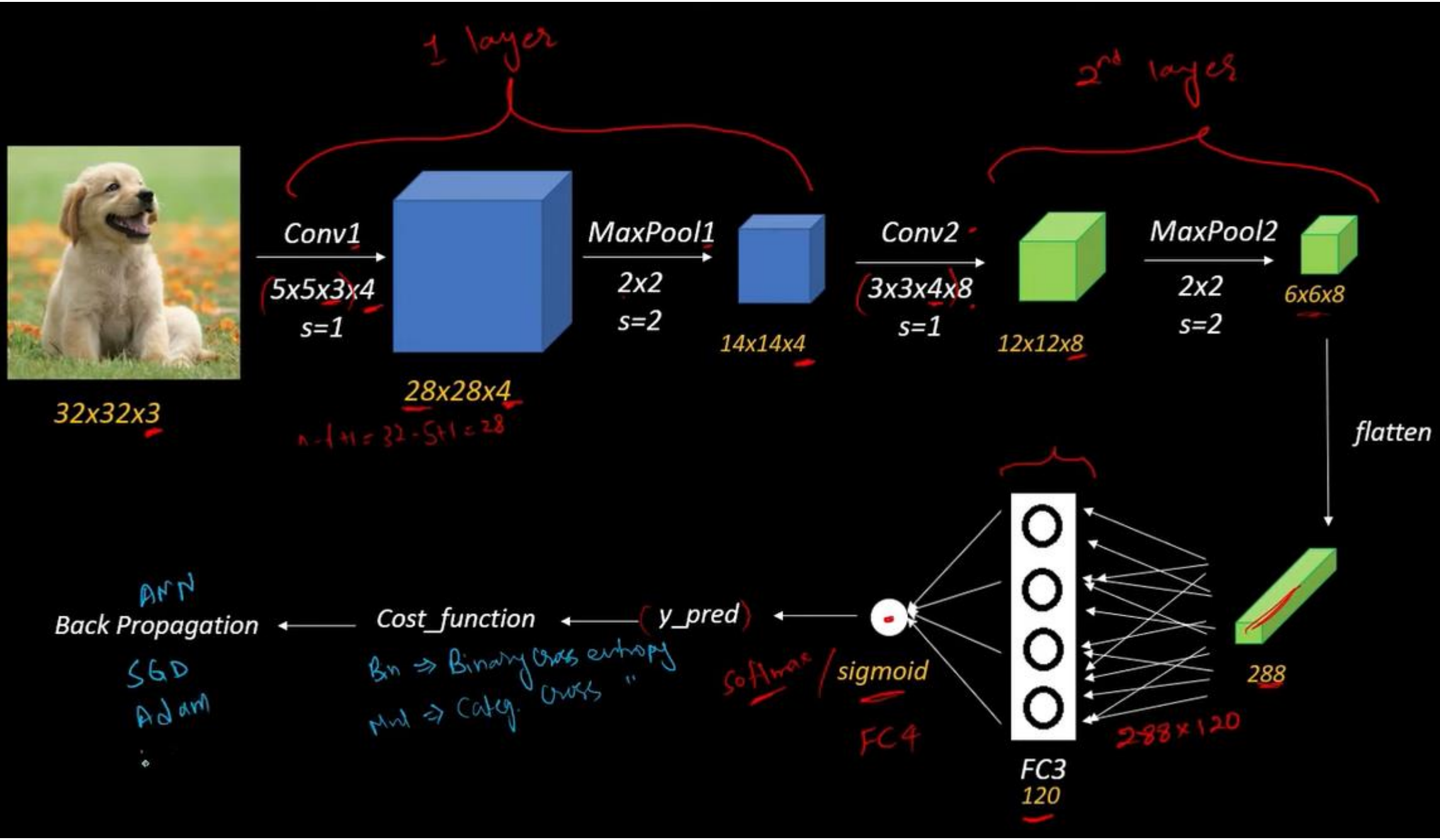
fxfx3 * → ReLU(+ b4) →

Convolutional Layer

Conv

fxfx3x4

(n-f+1)x(n-f+1)x4


1 layer
2nd layer
Conv1
5x5x3x4
s=1
MaxPool1
2x2
s=2
Conv2
3x3x4x8
s=1
MaxPool2
2x2
s=2
6x6x8
32x32x3
28x28x4
14x14x4
12x12x8
flatten
288
288x120
FC3
120
FC4
Softmax / sigmoid
y_pred
Cost_function
Back Propagation
ANN
SGD
Adam
Bn ⇒ Binary Cross entropy
Mul ⇒ Categ. Cross "

CNN

input

Conv1

MaxPool1

Conv2

MaxPool2

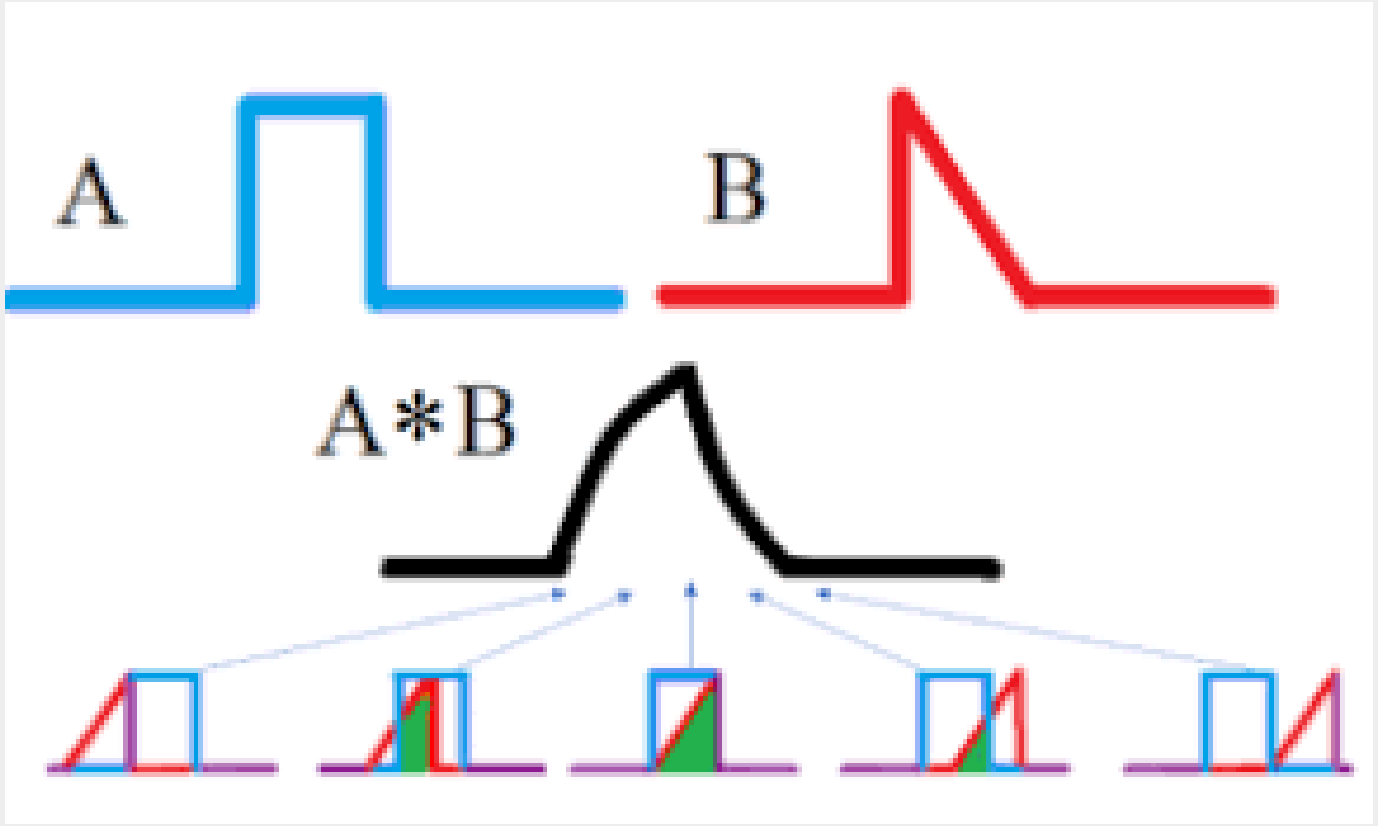
y_pred

ANN

input

y_pred

Convolution


A
B
A*B

The Convolution Integral

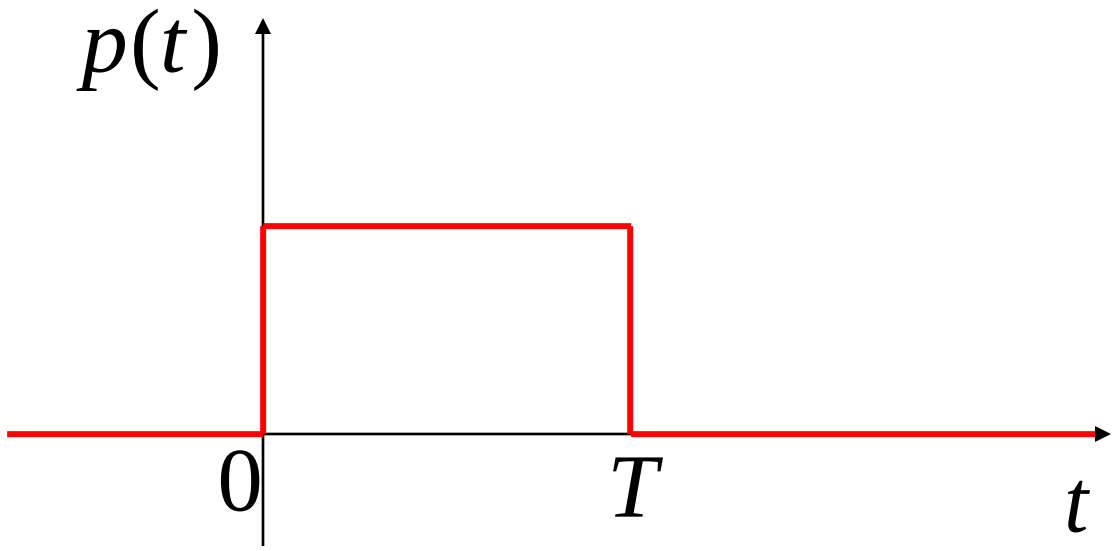
- This particular integration is called the convolution integral

$$y(t) = \underbrace{\int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau}_{x(t)*h(t)}, \quad t \geq 0$$

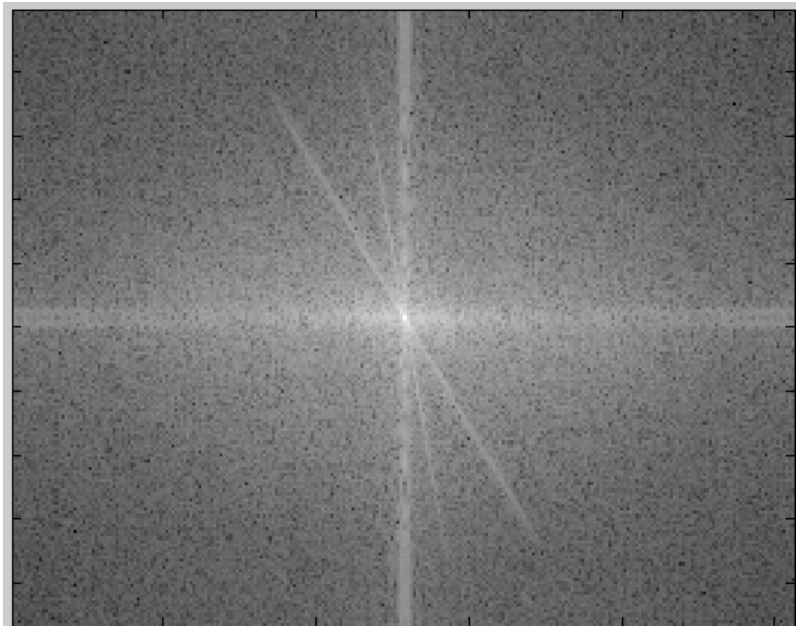
- Equation $y(t) = x(t) * h(t)$ is called the *convolution representation of the system*

Example: Analytical Computation of the Convolution Integral

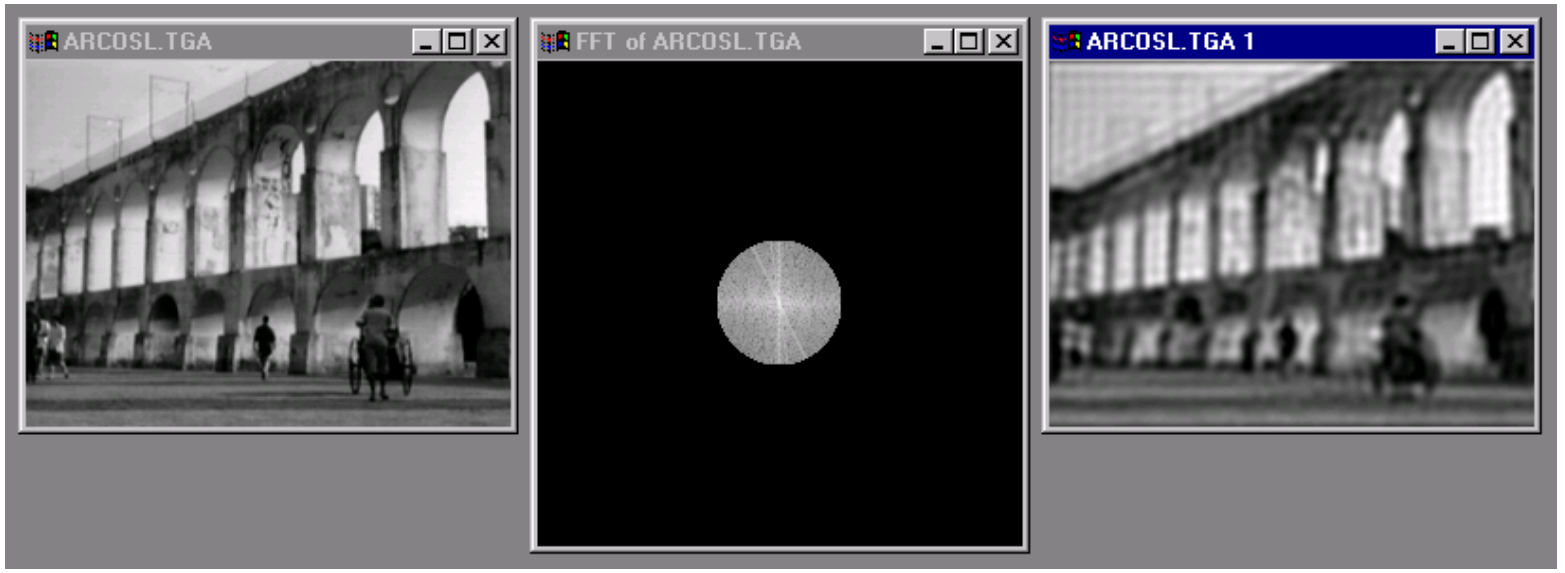
- Suppose that $x(t) = h(t) = p(t)$, where $p(t)$ is the rectangular pulse depicted in figure

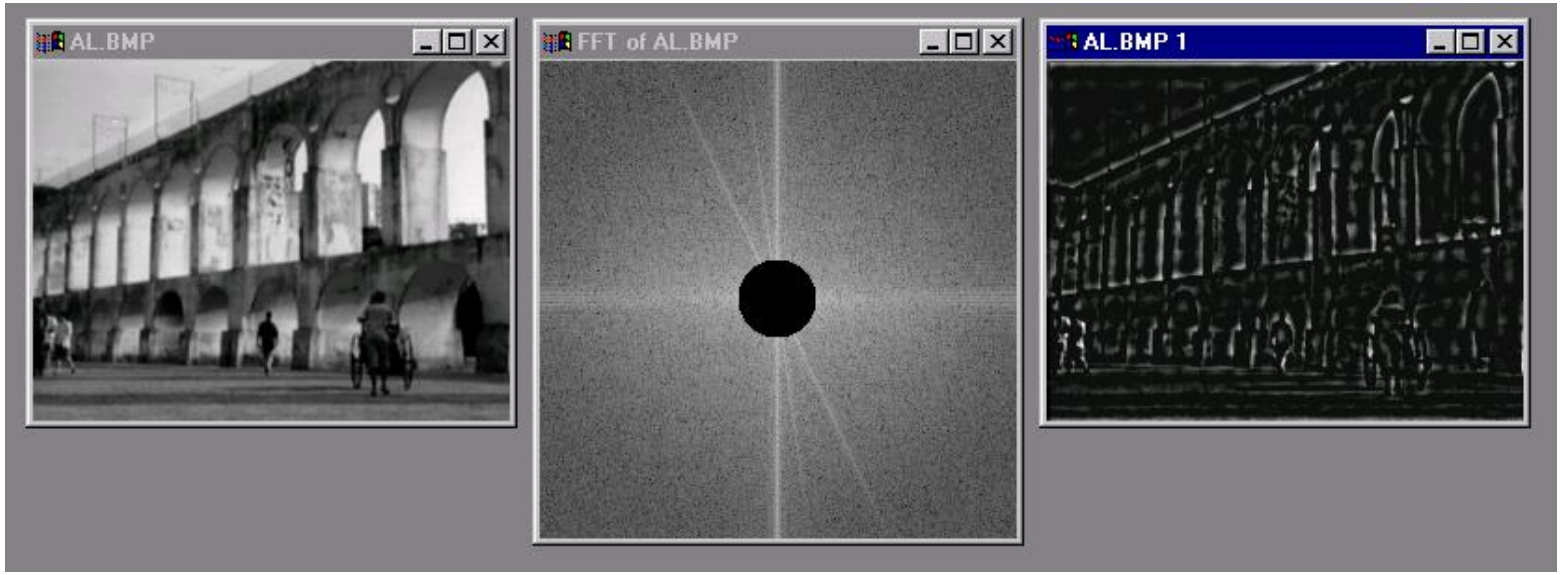


Fourier spectrum



Fun and games with spectra





Gaussian filtering

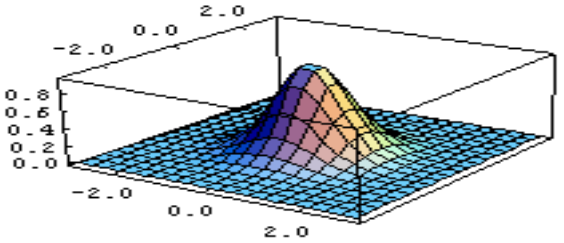
- A Gaussian kernel gives less weight to pixels further from the center of the window

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

$H[u, v]$



- This kernel is an approximation of a Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

$$G = H \otimes F$$

- Remember **cross-correlation**:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

- A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

- It is written:

$$G = H \star F$$

- Suppose H is a Gaussian or mean kernel. How does convolution differ from cross-correlation?

The Convolution Theorem

- The greatest thing since sliced (banana) bread!

- The Fourier transform of the convolution of two functions is the product of their Fourier transforms

$$\mathrm{F}[g * h] = \mathrm{F}[g]\mathrm{F}[h]$$

- The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

$$\mathrm{F}^{-1}[gh] = \mathrm{F}^{-1}[g] * \mathrm{F}^{-1}[h]$$

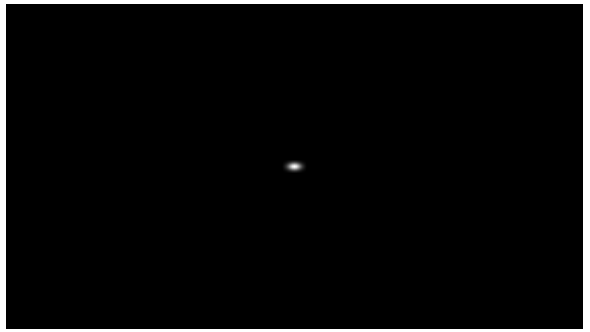
- **Convolution** in spatial domain is equivalent to **multiplication** in frequency domain!

2D convolution theorem example

$f(x,y)$

$*$

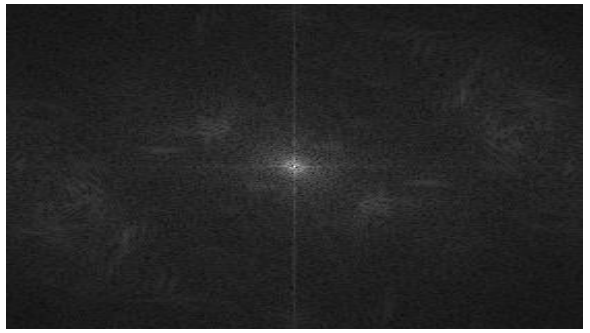
$h(x,y)$



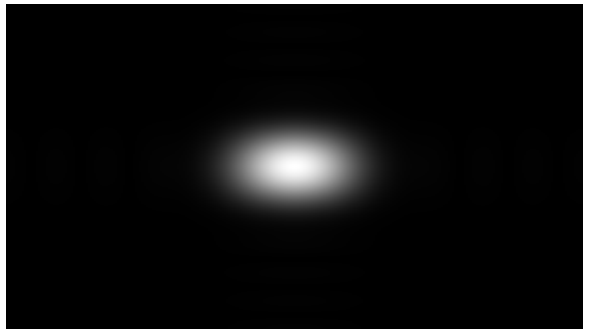
$\Downarrow$

$g(x,y)$

$\asymp$

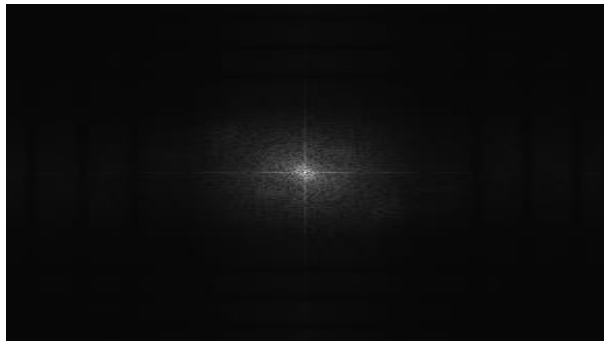


$|F(s_x,s_y)|$



$\Downarrow$

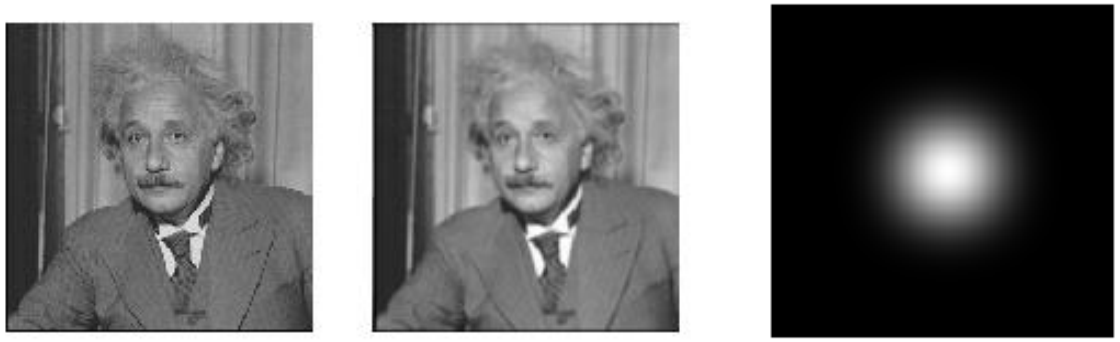
$|H(s_x,s_y)|$



$|G(s_x,s_y)|$

Low-pass, Band-pass, High-pass filters

low-pass:



band-pass:

what's high-pass?

Edges in images

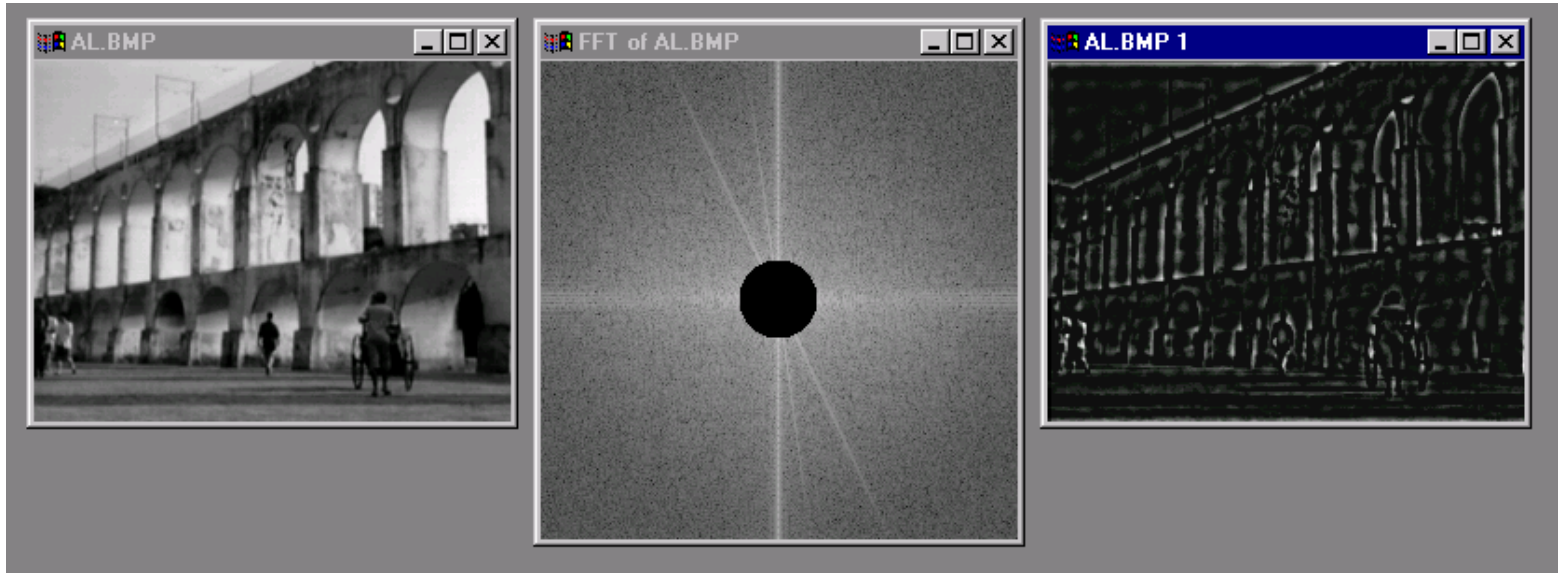
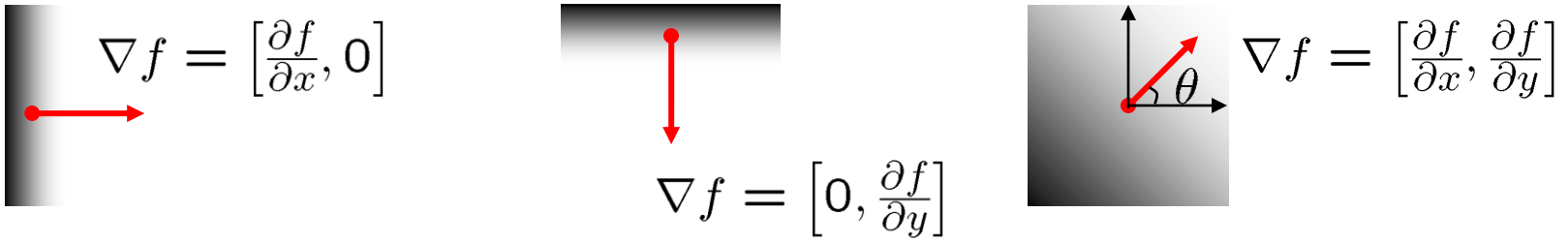


Image gradient

- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- The gradient points in the direction of most rapid change in intensity



The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

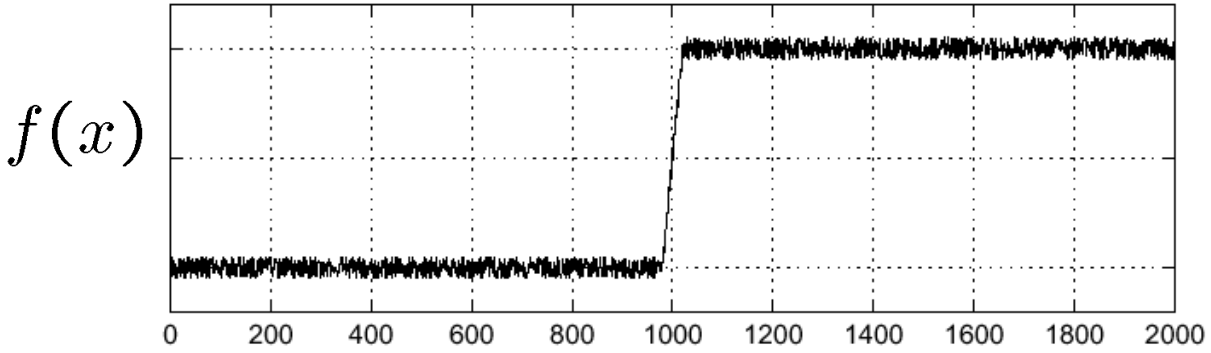
- how does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

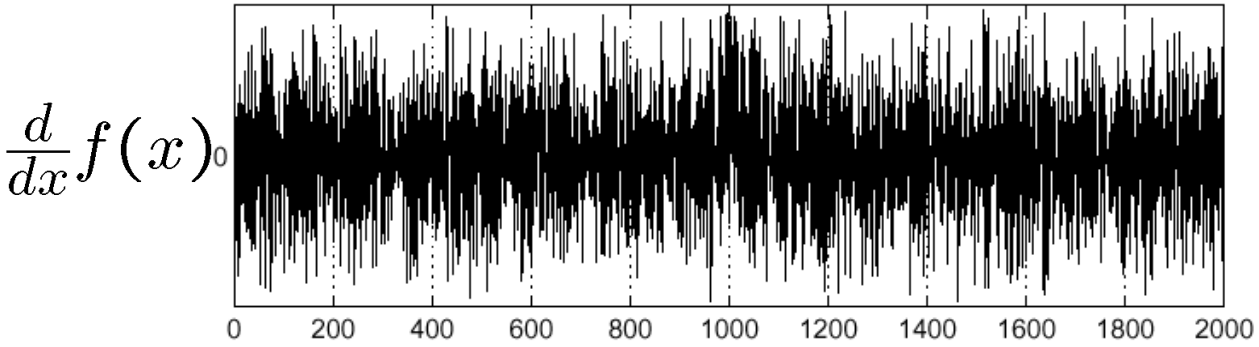
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



How to compute a derivative?



Where is the edge?

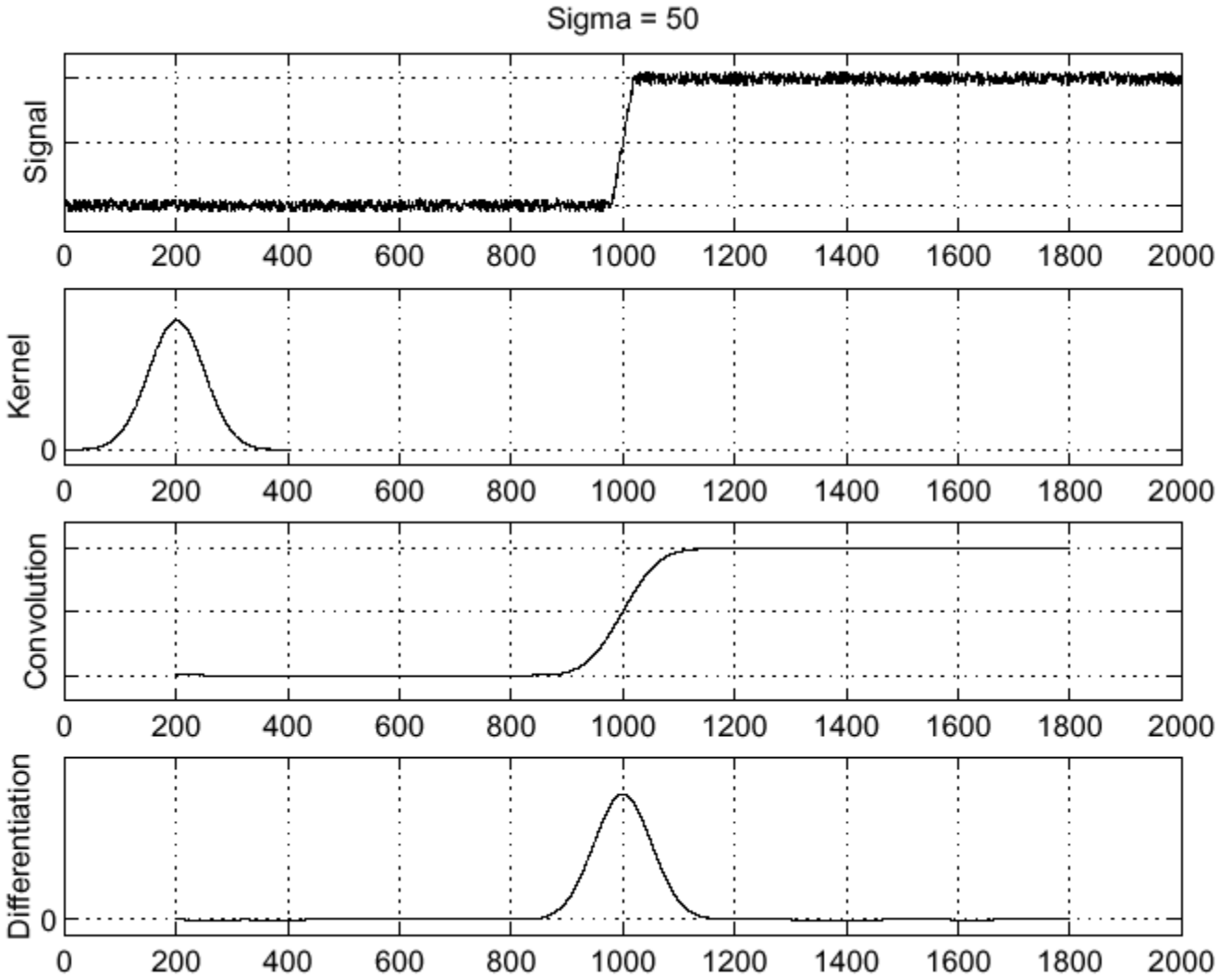
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

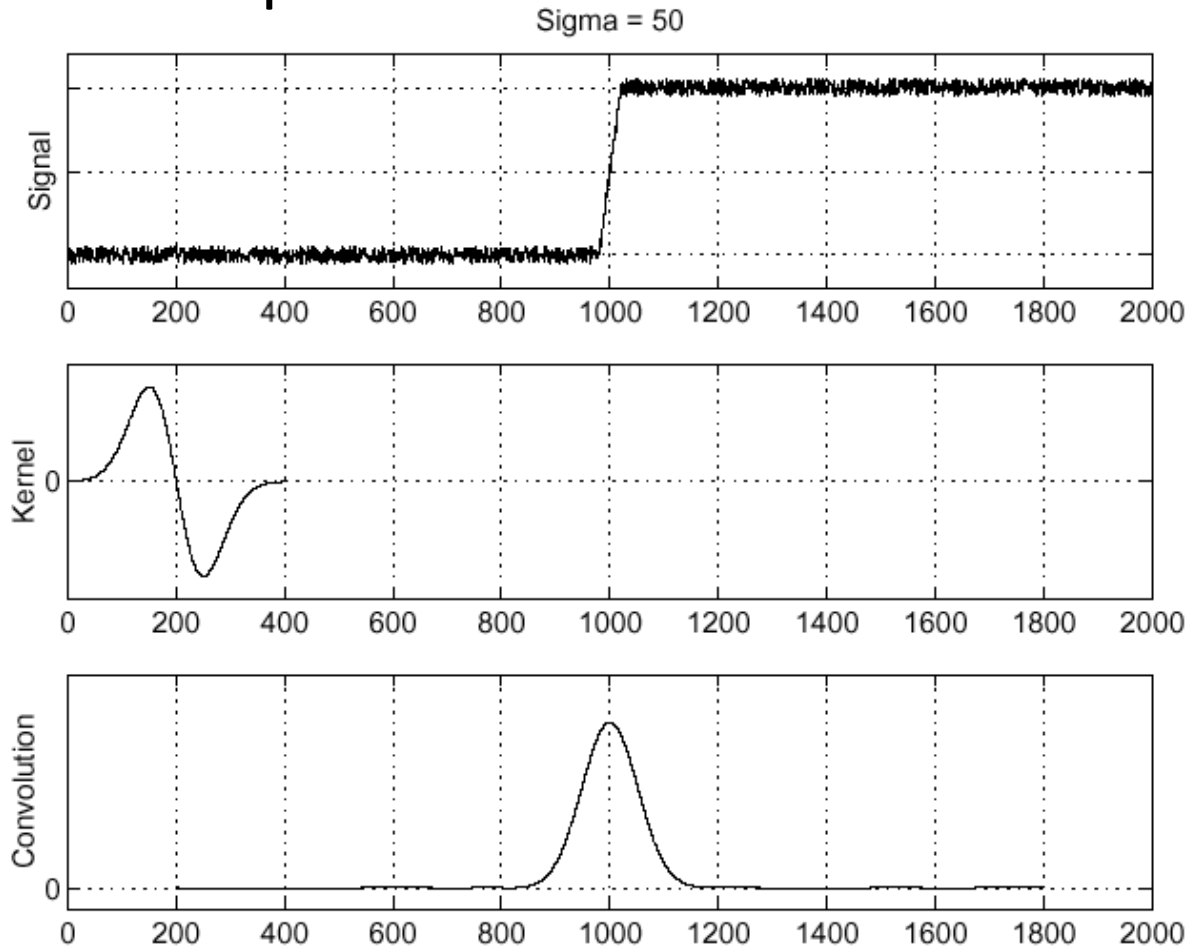
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

f

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



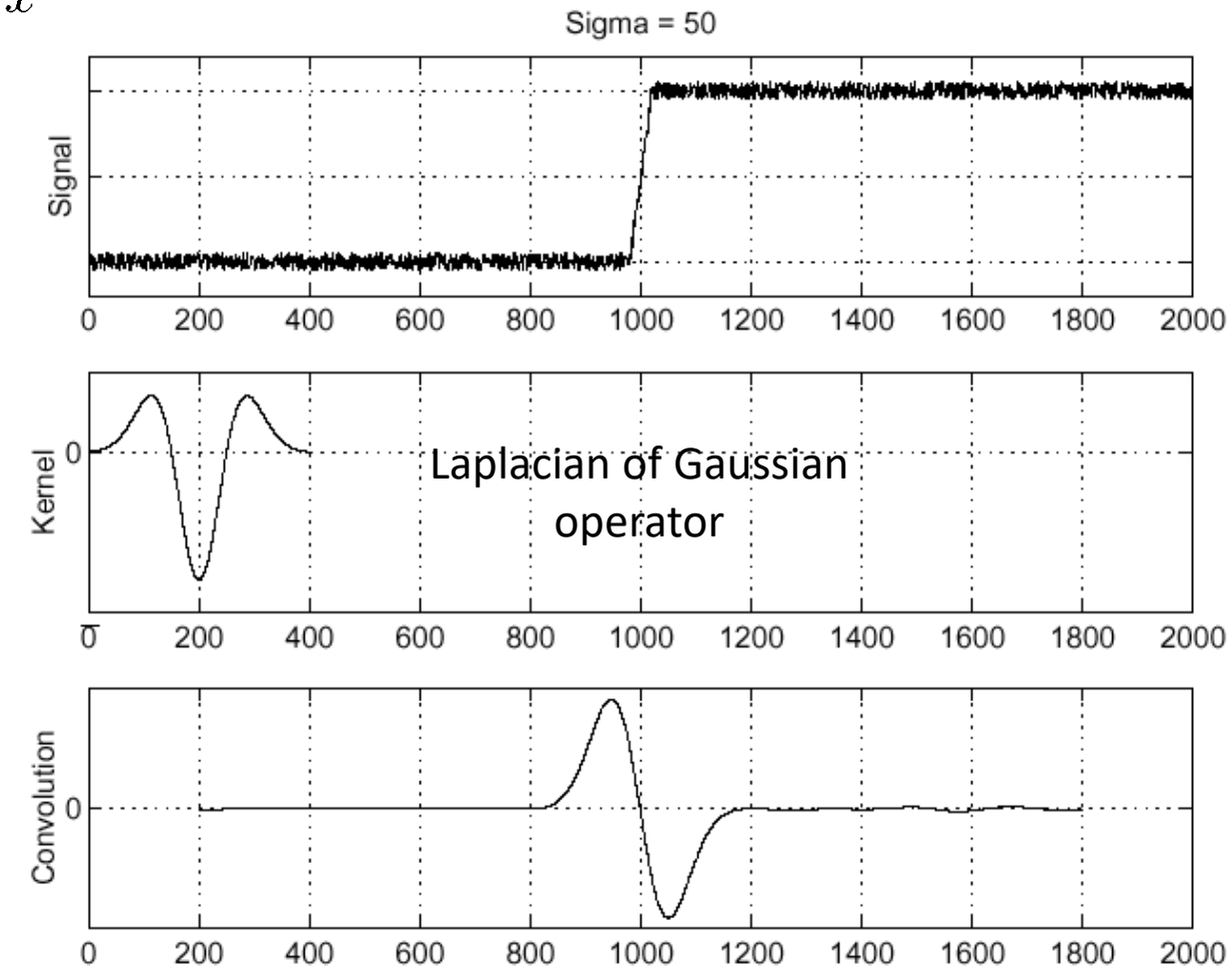
Laplacian of Gaussian

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

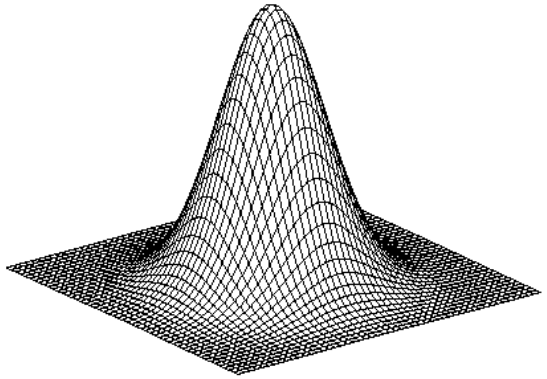
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



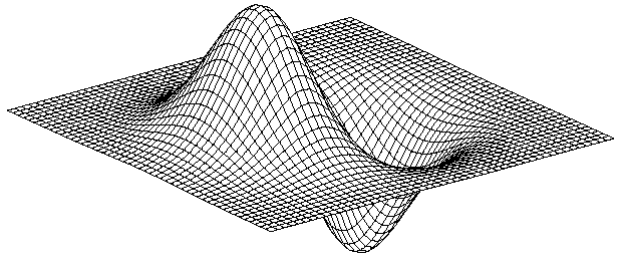
Where is the edge? Zero-crossings of bottom graph

2D edge detection filters



Gaussian

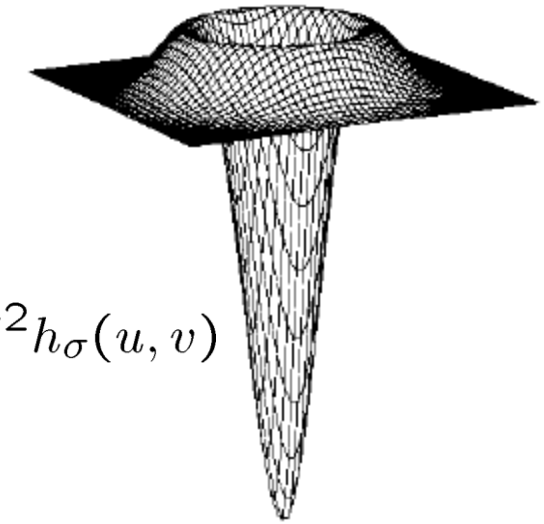
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

∇^2 is the **Laplacian** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Linear Image Filters

Linear operations calculate the resulting value in the output image pixel f(i,j) as a linear combination of brightness in a local neighborhood of the pixel h(i,j) in the input image.
This equation is called to **discrete convolution**:

$$f(i,\ j) = w * h = \sum_{m=-a}^{a} \sum_{n=-b}^{b} w(m,n)h(i+m,\ j+n)$$

Function w is called a **convolution kernel** or a **filter mask**. In our case it is a rectangle of size (2a+1)x(2b+1).

Edge detectors

- locate sharp changes in the intensity function
- edges are pixels where brightness changes abruptly.

ϕ

- Calculus describes changes of continuous functions using derivatives; an image function depends on two variables - partial derivatives.
- A change of the image function can be described by a gradient that points in the direction of the largest growth of the image function.
- An edge is a property attached to an individual pixel and is calculated from the image function behavior in a neighborhood of the pixel.
- It is a **vector variable**:

magnitude of the gradient and **direction**

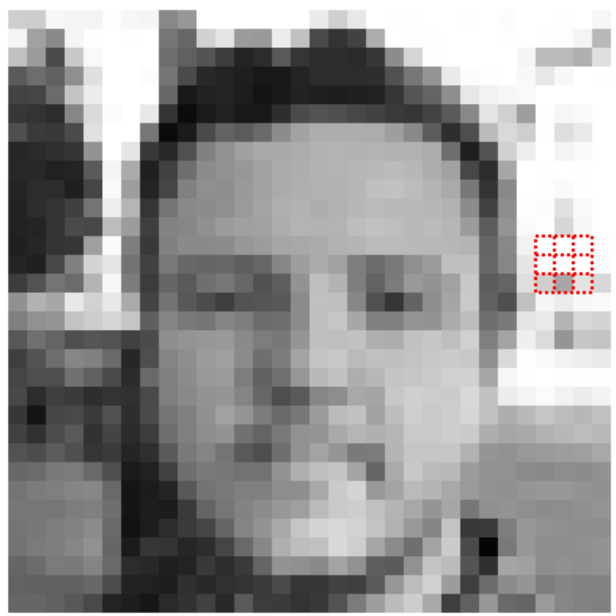
https://setosa.io/ev/image-kernels/

Let's walk through applying the following 3x3 **custom** kernel to the image of a face from above.

left sobel

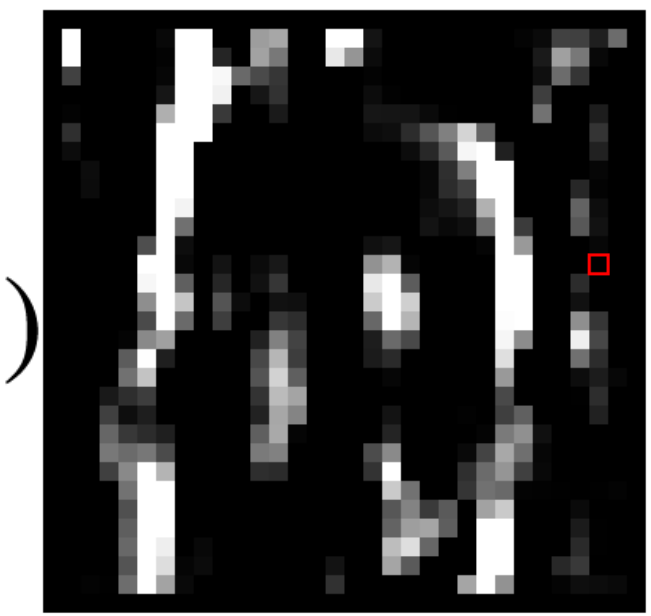
$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



$$\begin{pmatrix} 255 \times 1 + 255 \times 0 + 254 \times -1 \\ + 253 \times 2 + 253 \times 0 + 255 \times -2 \\ + 207 \times 1 + 167 \times 0 + 227 \times -1 \end{pmatrix} = -23$$

kernel: left sobel



input image

output image

Simple box blur

Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:

A simple blur done with convolutions

Gaussian blur

Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Here's a result that I got:

Line detection with image convolutions

With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

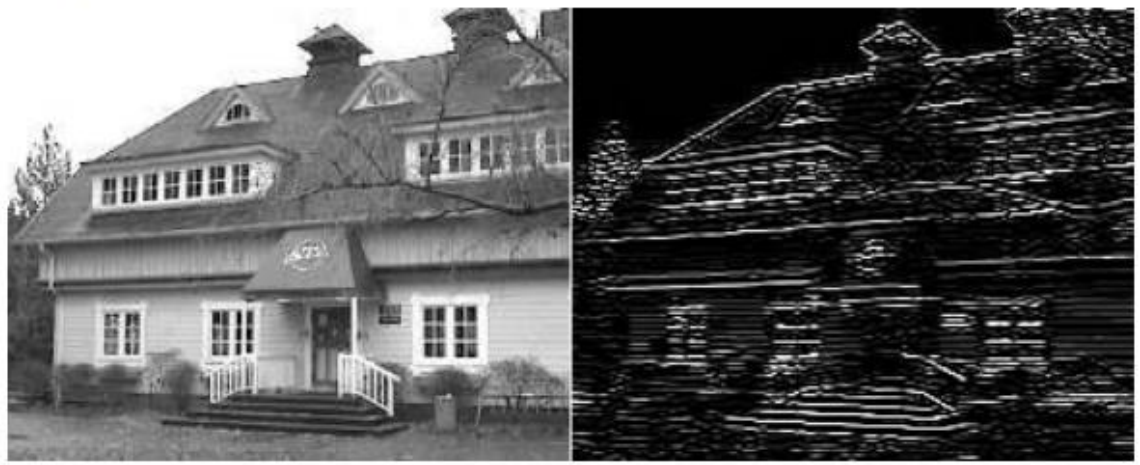
-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

I looked for horizontal lines on the house image. The result I got for this image convolution was:



Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to "combine" the results is to merge the convolution kernels. The new image convolution kernel looks like this:

-1	-1	-1
-1	8	-1
-1	-1	-1

Below result I got with edge detection:

The Sobel Edge Detector

The Sobel edge detector is a gradient based method. It works with first order derivatives. It calculates the first derivatives of the image separately for the X and Y axes. The derivatives are only approximations (because the images are not continuous). To approximate them, the following kernels are used for convolution:

-1	0	1
-2	0	2
-1	0	1

Horizontal

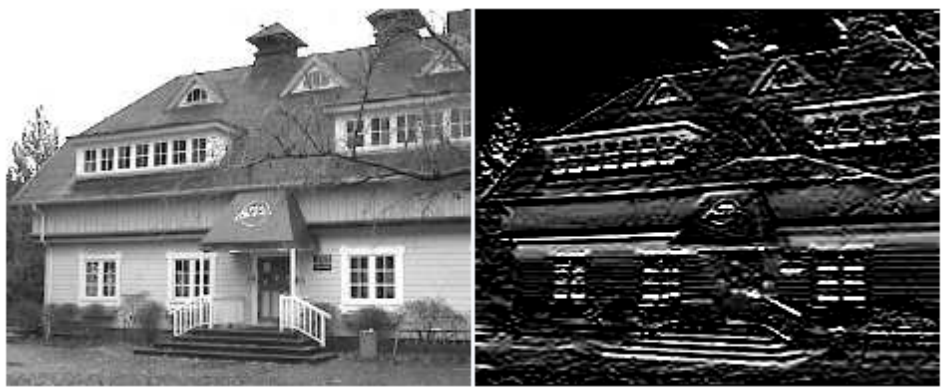
1	2	1
0	0	0
-1	-2	-1

Vertical

Kernels used in the Sobel edge detection

The kernel on the left approximates the derivative along the X axis. The one on the right is for the Y axis. Using this information, you can calculate the following:

- Magnitude or "strength" of the edge: $\sqrt{G_x^2 + G_y^2}$
- Approximate strength: $|G_x| + |G_y|$
- The orientation of the edge: $\arctan(\frac{G_y}{G_x})$



Result of the horizontal sobel operator

The Laplacian Edge Detector

Unlike the Sobel edge detector, the Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass. Here's the kernel used for it:

0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

The kernel for the laplacian operator

You can use either one of these. Or if you want a better approximation, you can create a 5x5 kernel (it has a 24 at the center and everything else is -1). Simple stuff. One serious drawback though - because we're working with second order derivatives, the laplacian edge detector is extremely sensitive to noise. Usually, you'll want to reduce noise - maybe using the Gaussian blur. Here's a result I got:

The result of convolution with the laplacian operator

Laplacians are computationally faster to calculate (only one kernel vs two kernels) and sometimes produce exceptional results!

Morphological Filter

$$dilation(W(x,y)) = \max(W(x,y))$$

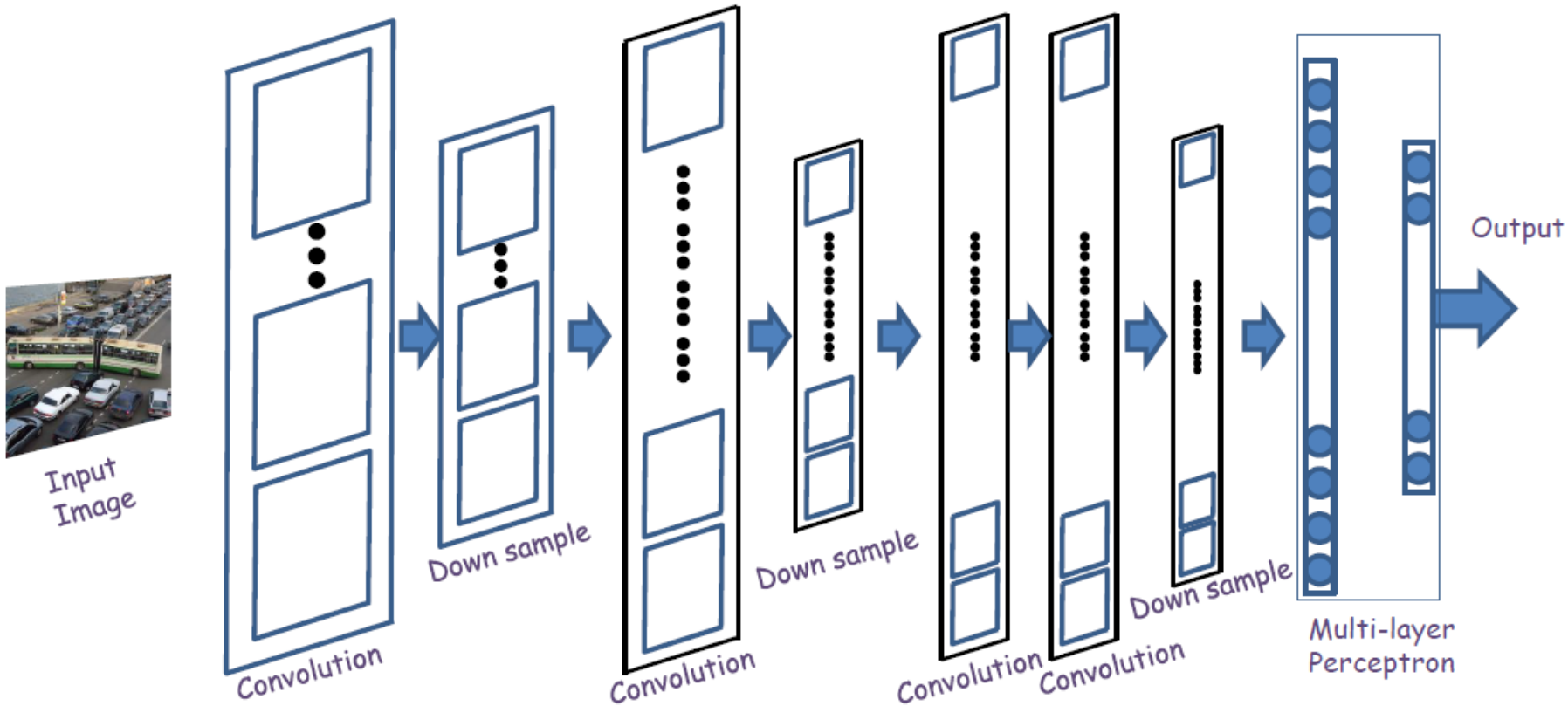
$$erosion(W(x,y)) = \min(W(x,y))$$

$$opening = dilation \circ erosion$$

$$closing = erosion \circ dilation$$

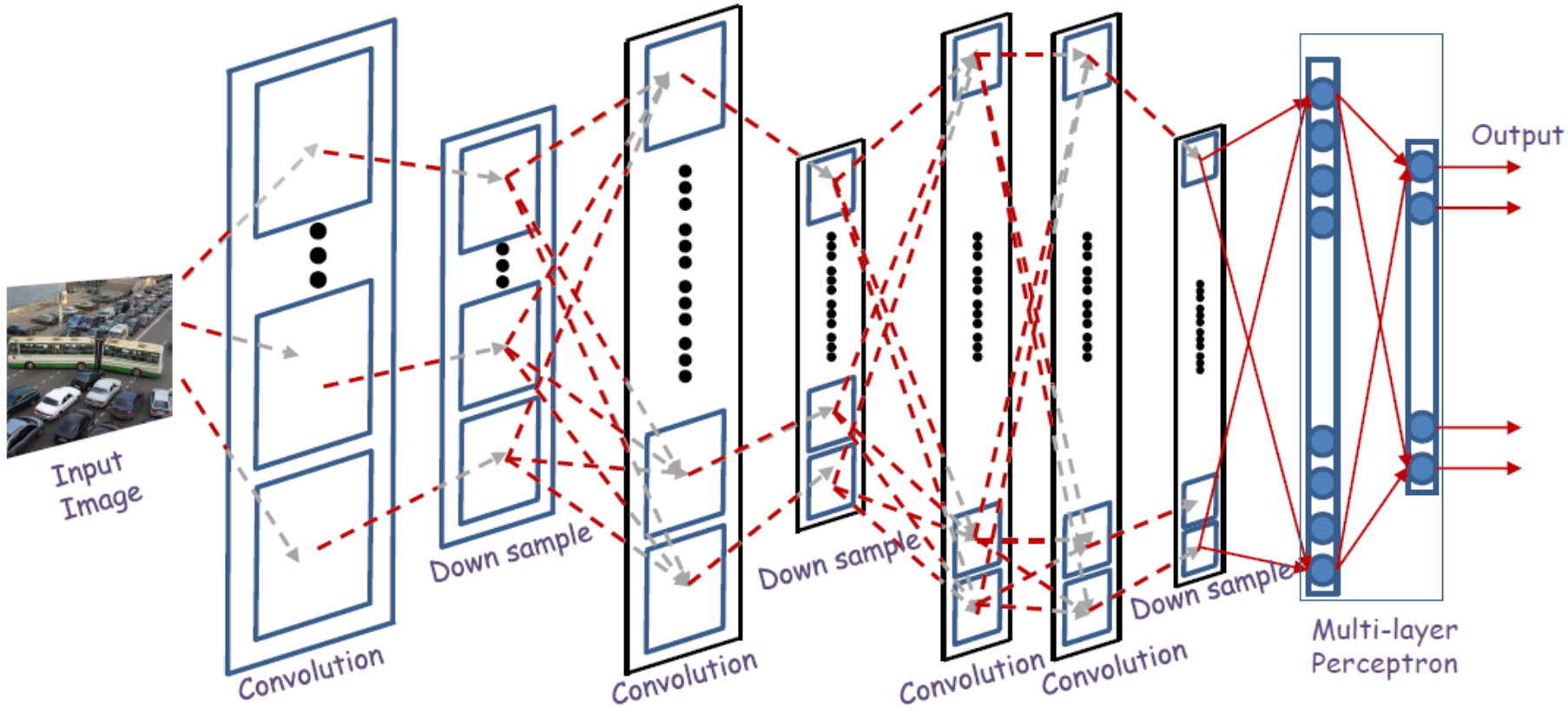
$$W(x,y) = \begin{bmatrix} 110 & 110 & 104 \\ 100 & 114 & 104 \\ 95 & 88 & 85 \end{bmatrix} \Rightarrow dilation(W(x,y)) = 114$$

The general architecture of a convolutional neural network



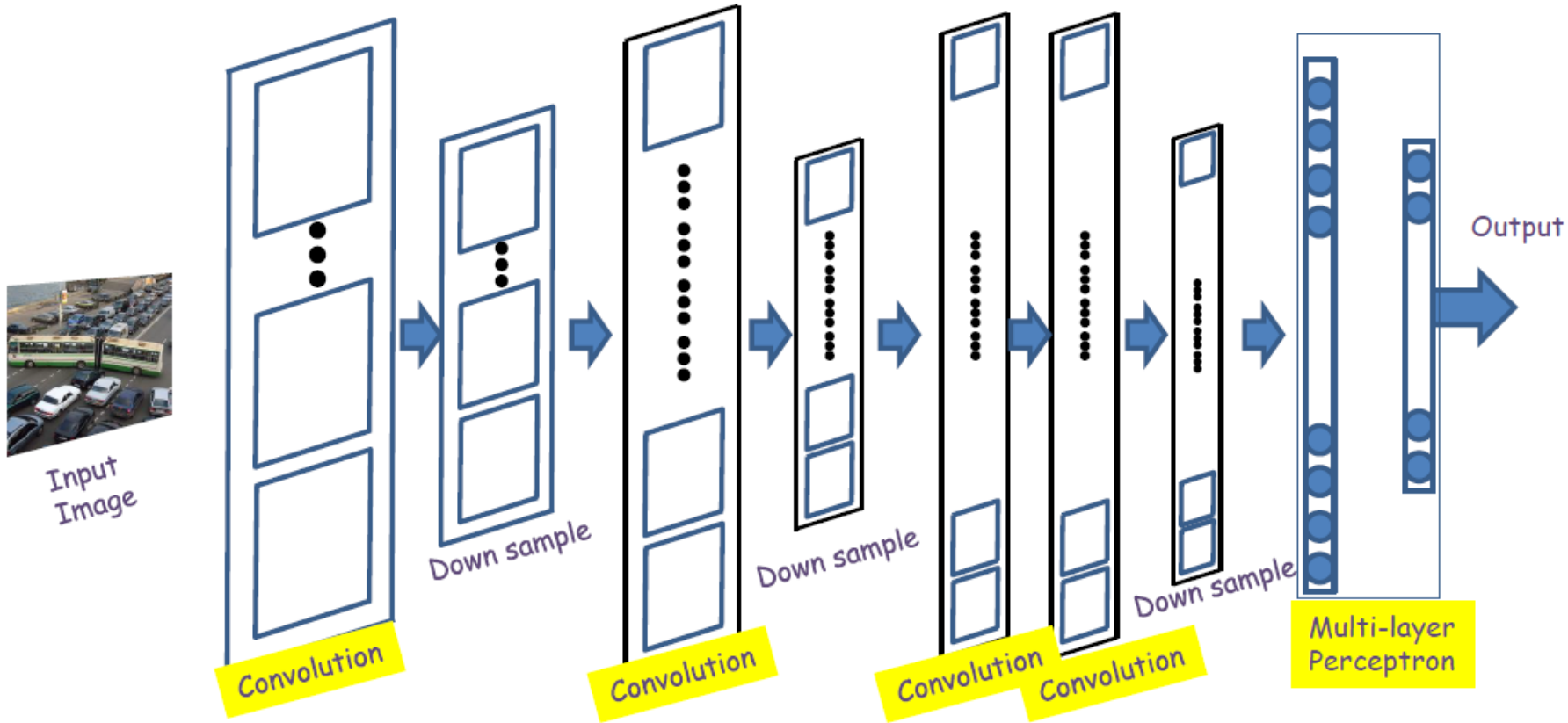
- A convolutional neural network comprises "convolutional" and "downsampling" layers
  - Convolutional layers comprise neurons that scan their input for patterns
  - Downsampling layers perform max operations on groups of outputs from the convolutional layers
  - The two may occur in any sequence, but typically they alternate
- Followed by an MLP with one or more layers

The general architecture of a convolutional neural network



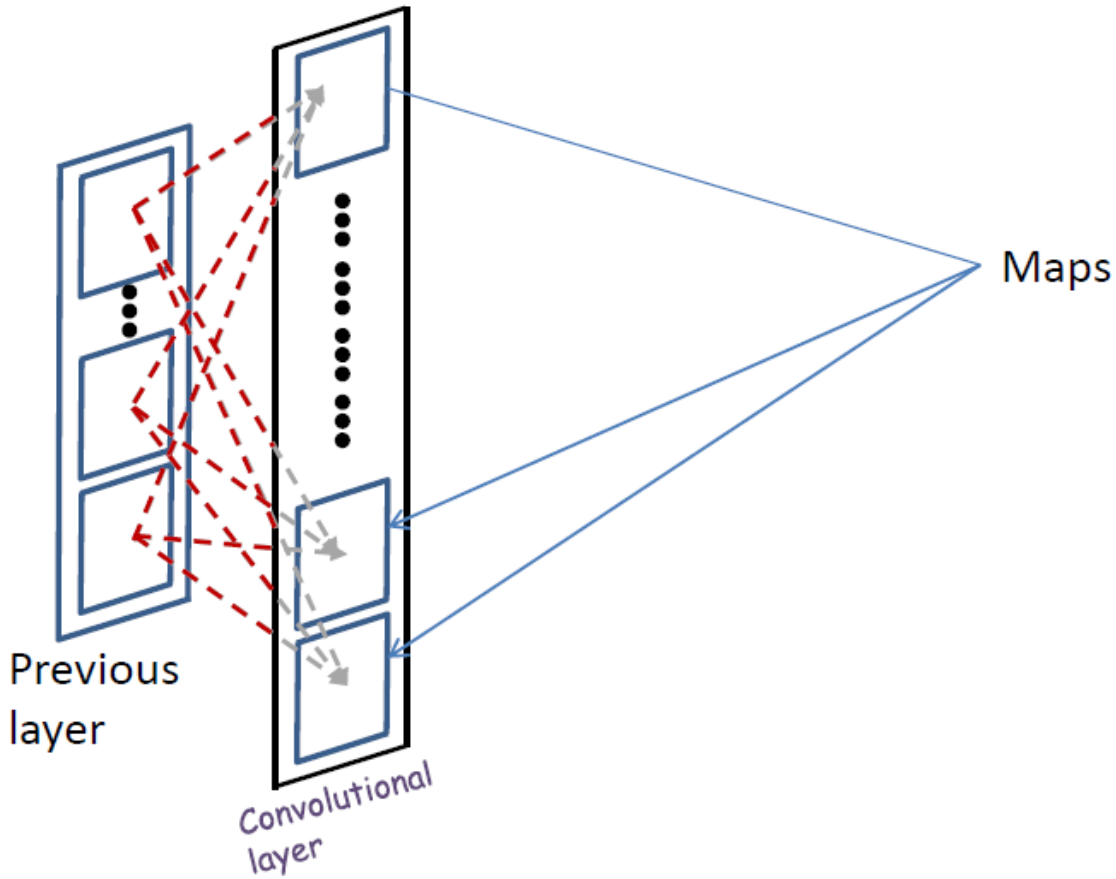
- A convolutional neural network comprises of "convolutional" and "downsampling" layers
  - The two may occur in any sequence, but typically they alternate
- Followed by an MLP with one or more layers

The general architecture of a convolutional neural network



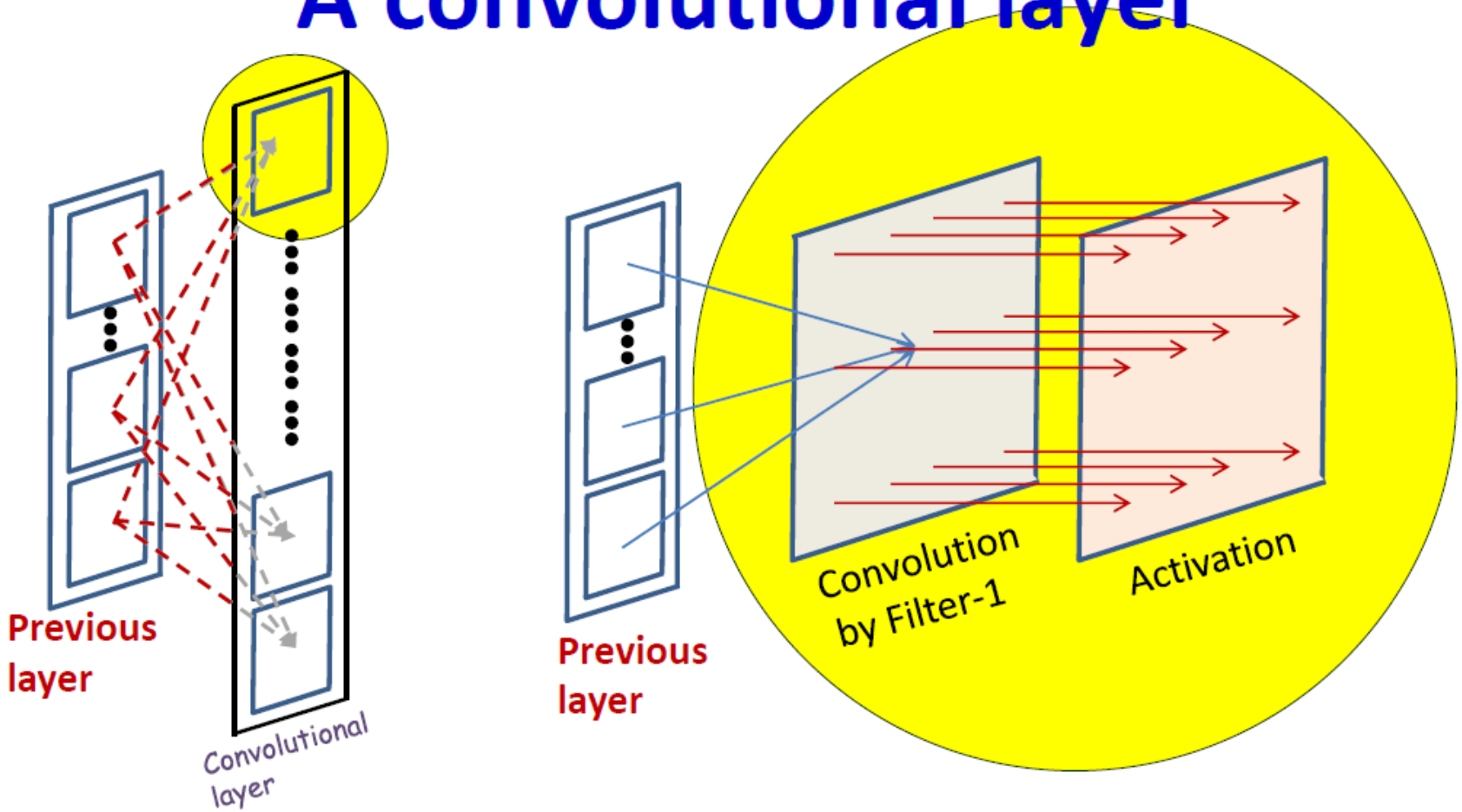
- **Convolutional layers and the MLP are *learnable***
  - Their parameters must be learned from training data for the target classification task
- Down-sampling layers are fixed and generally not learnable

A convolutional layer



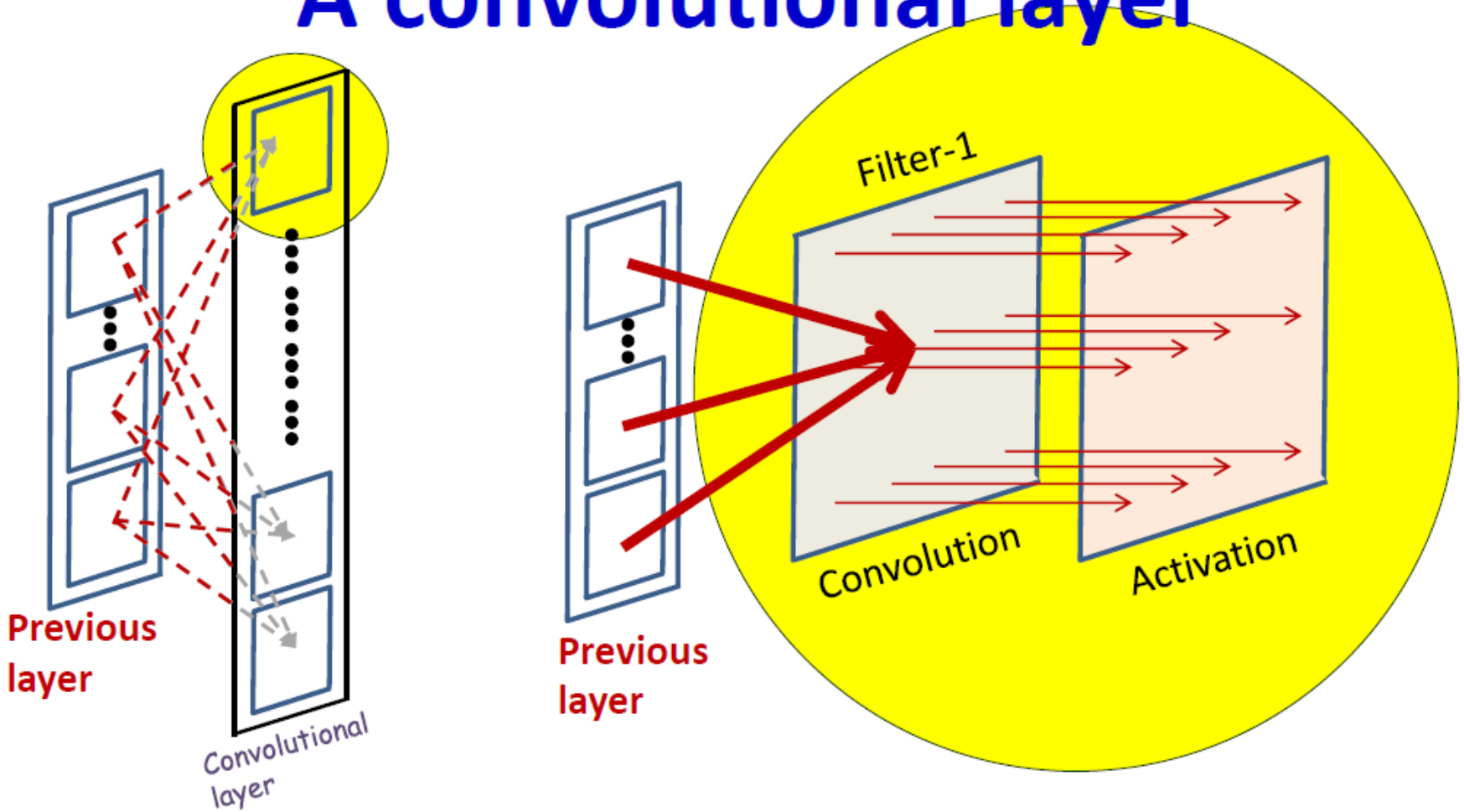
- A convolutional layer comprises of a series of "maps"
  - Corresponding the "S-planes" in the Neocognitron
  - Variously called feature maps or activation maps

A convolutional layer



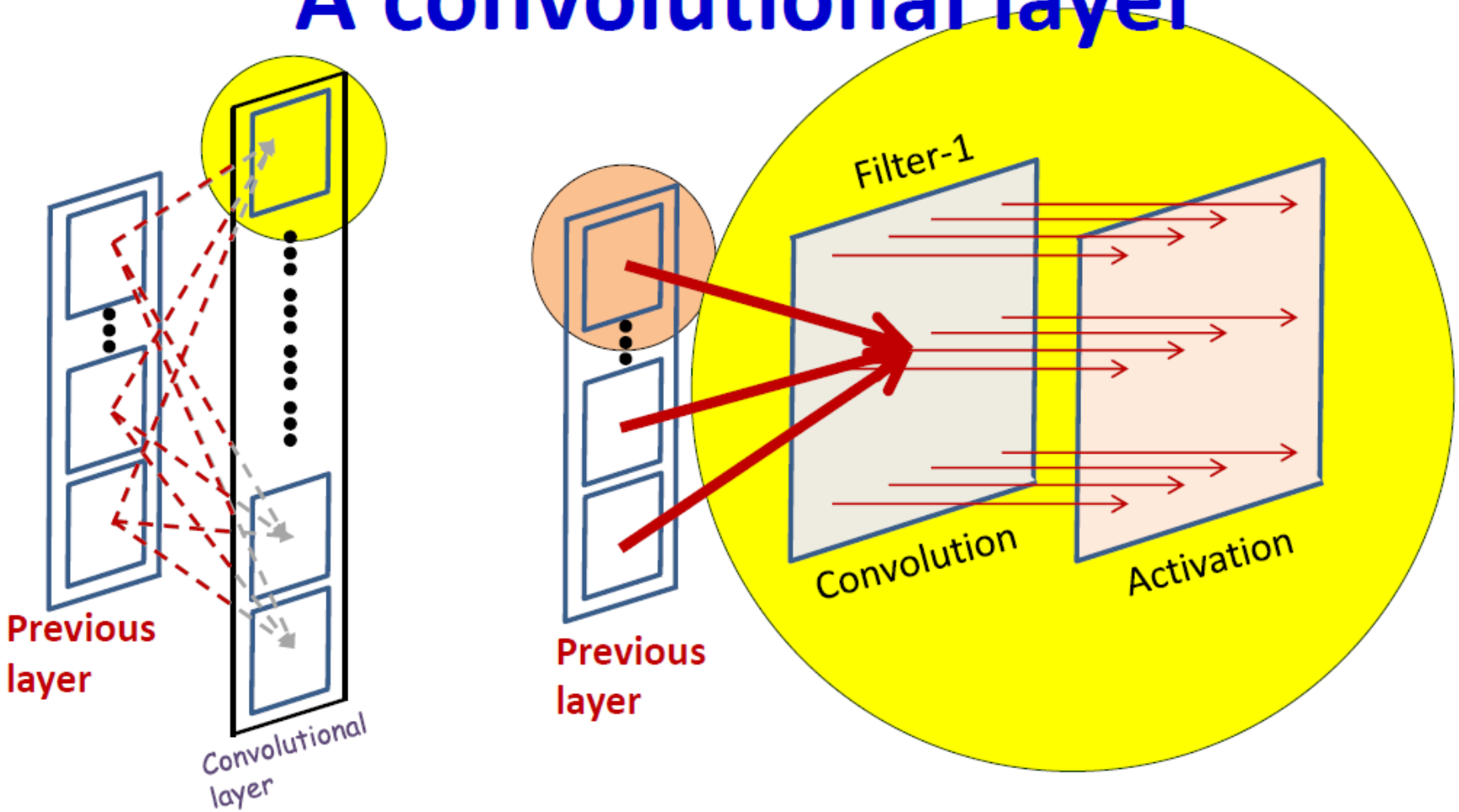
- Each activation map has two components
  - An *affine* map, obtained by *convolution* over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates on the output of the convolution

A convolutional layer



- All the maps in the previous layer contribute to each convolution

A convolutional layer



- All the maps in the previous layer contribute to each convolution
  - Consider the contribution of a *single* map

What is a convolution

Example 5x5 image with binary pixels

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

Example 3x3 filter

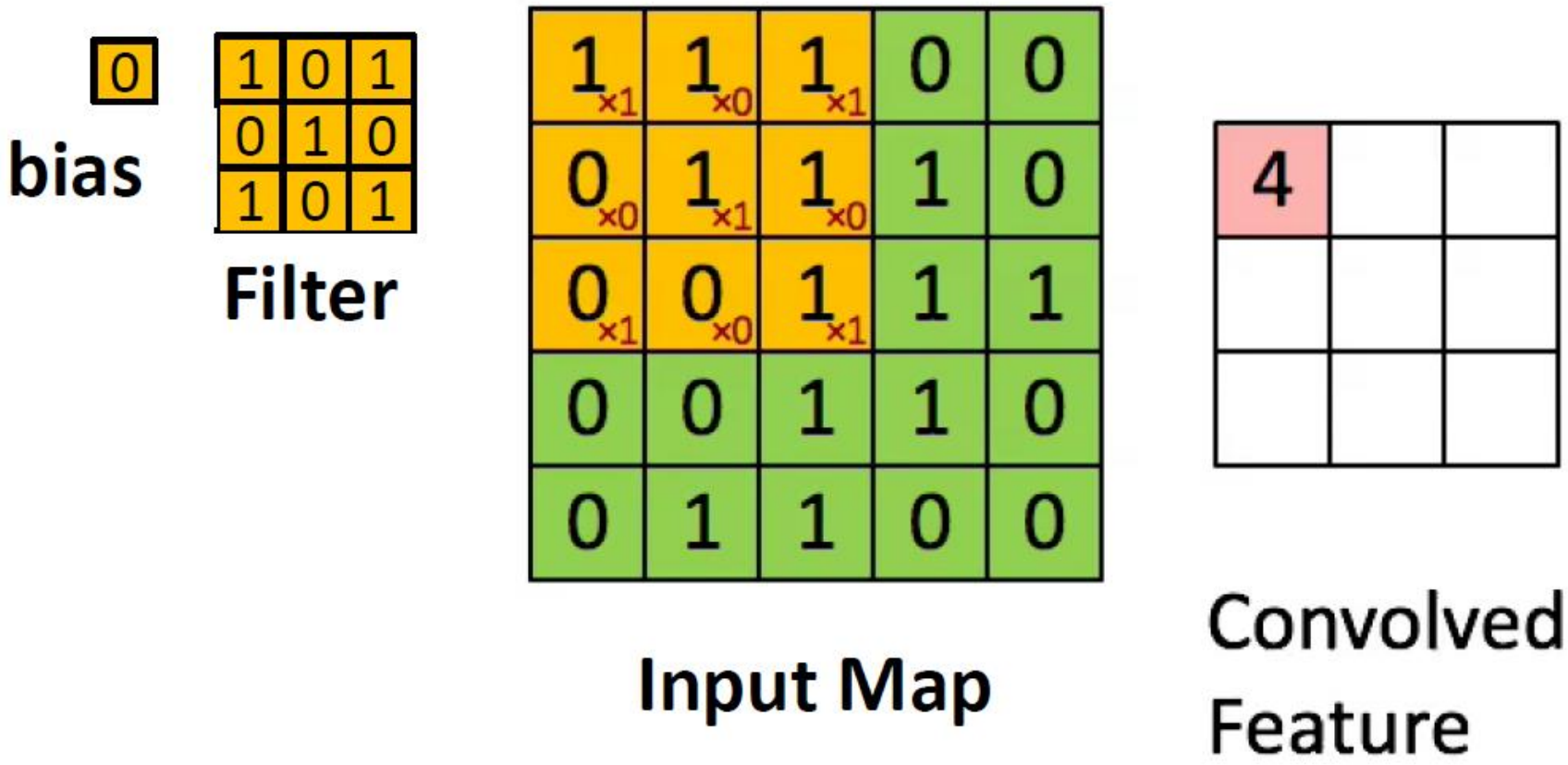
1	0	1
0	1	0
1	0	1

bias

0

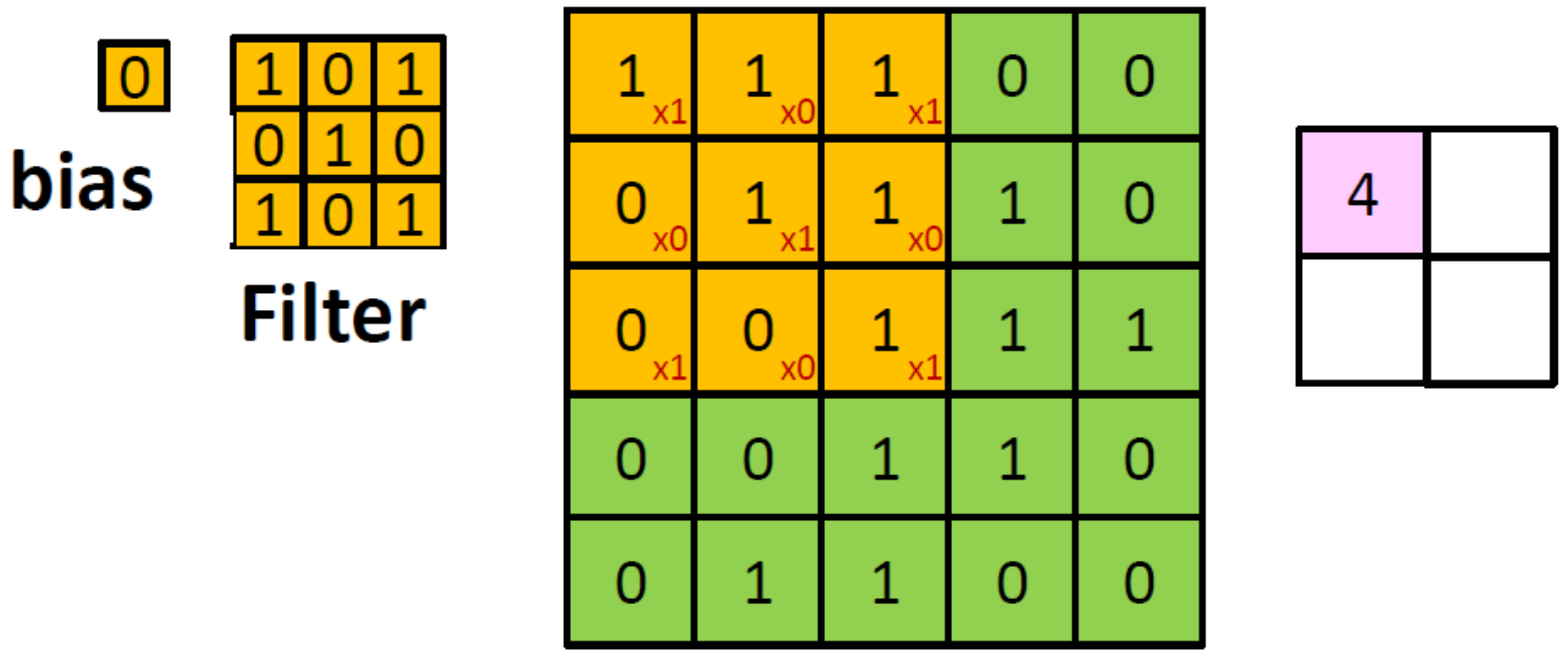
- Scanning an image with a “filter”
  - Note: a filter is really just a perceptron, with weights and a bias

What is a convolution



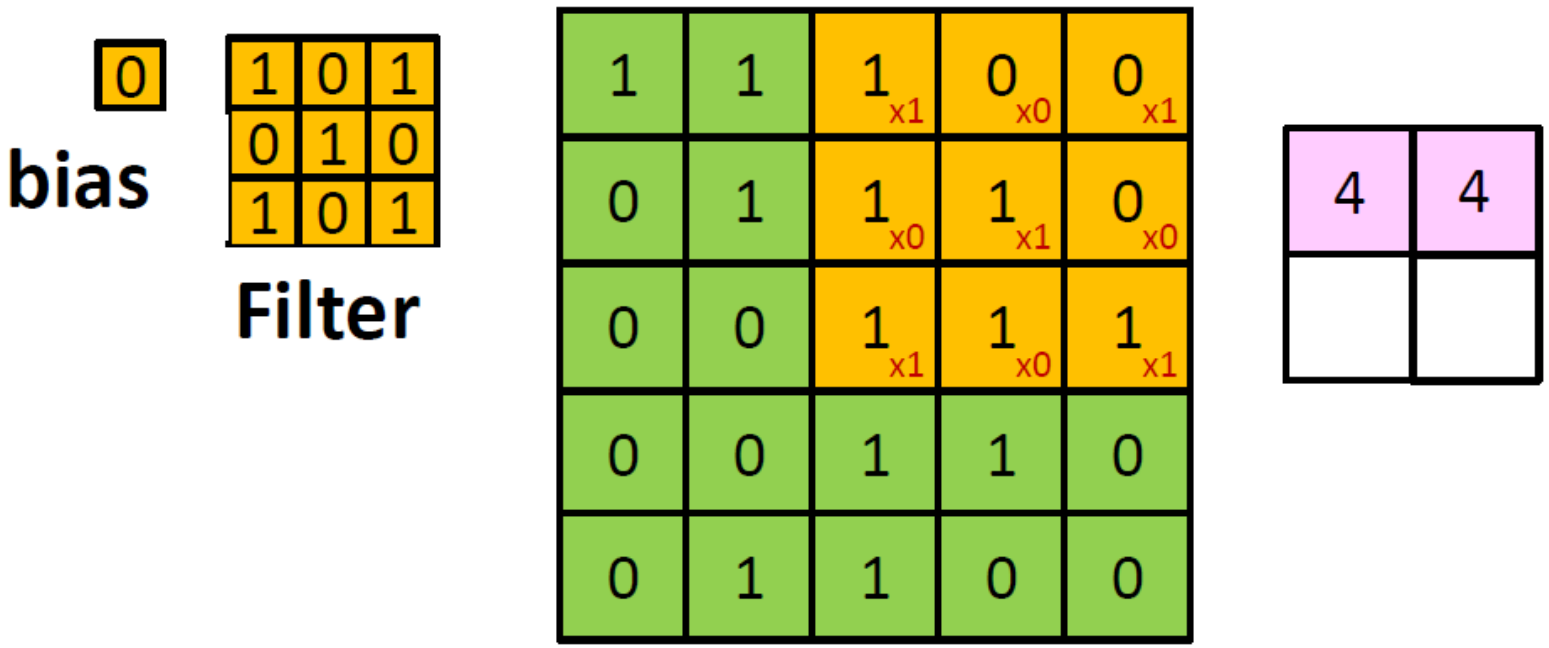
- Scanning an image with a “filter”
  - At each location, the “filter and the underlying map values are multiplied component wise, and the products are added along with the bias

The “Stride” between adjacent scanned locations need not be 1



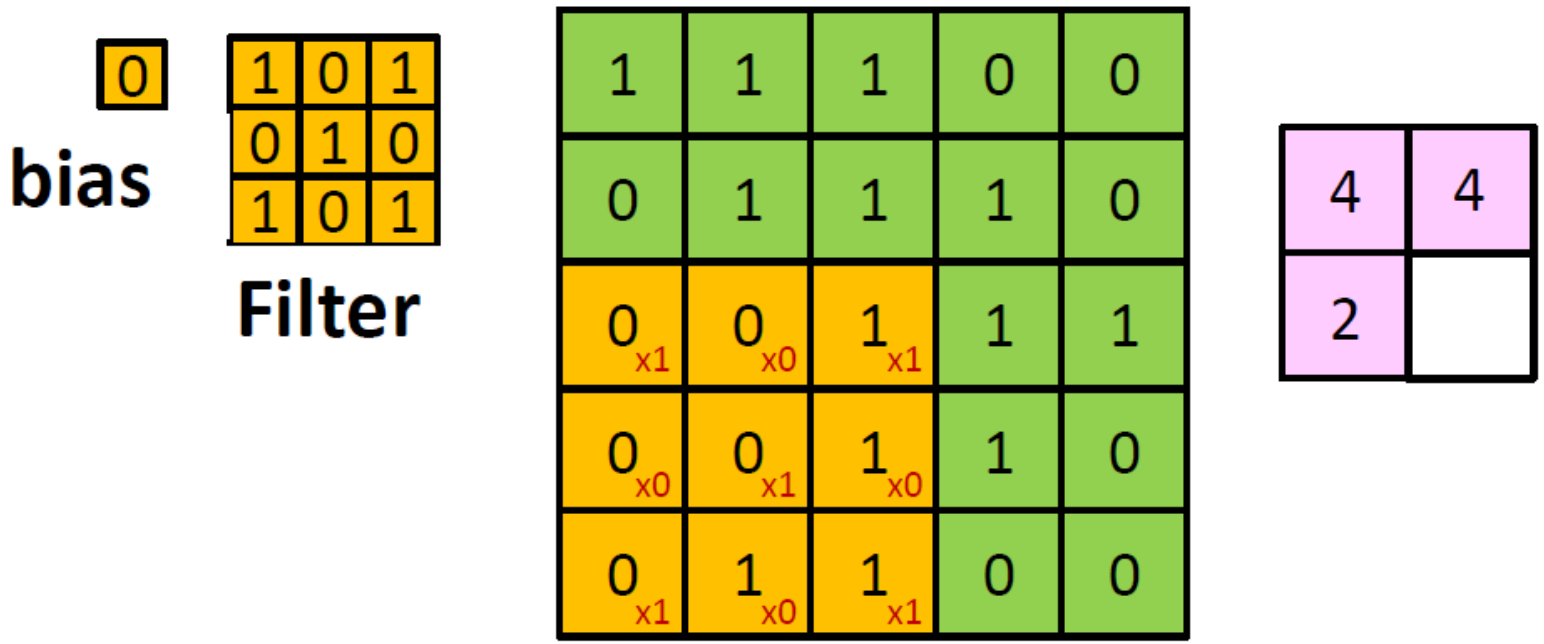
- Scanning an image with a “filter”
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a “stride” of *two* pixels per shift

The "Stride" between adjacent scanned locations need not be 1



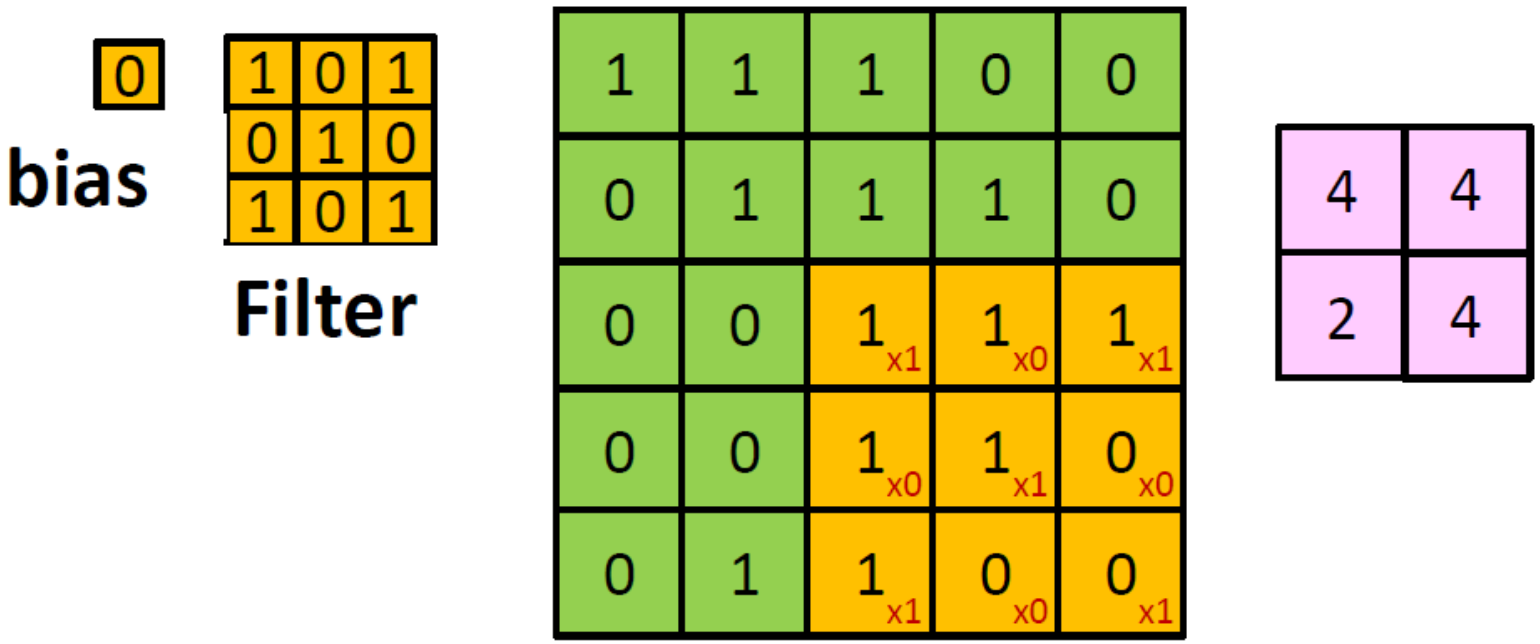
- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "hop" of *two* pixels per shift

The “Stride” between adjacent scanned locations need not be 1



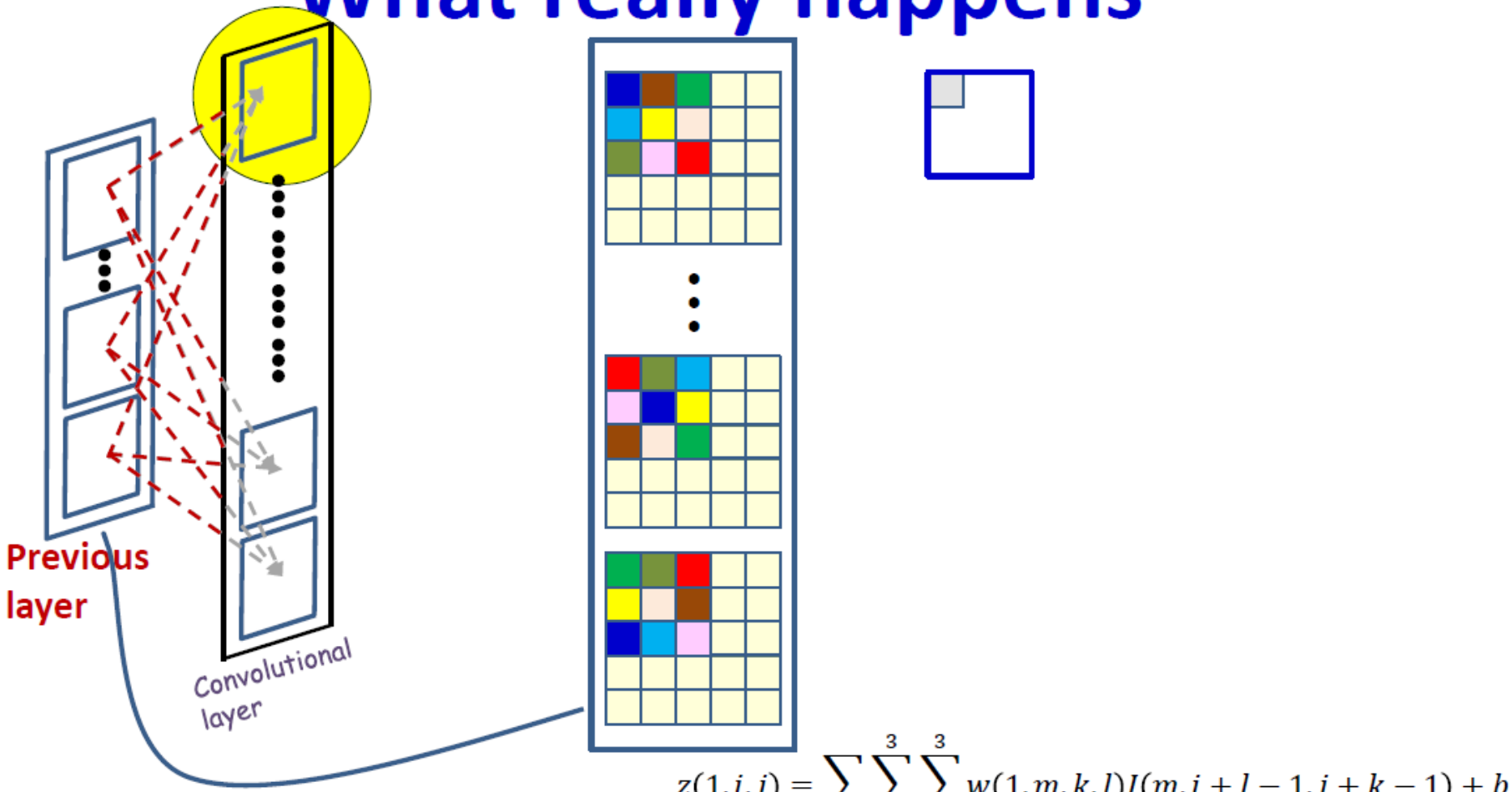
- Scanning an image with a “filter”
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a “hop” of *two* pixels per shift

The "Stride" between adjacent scanned locations need not be 1



- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "hop" of *two* pixels per shift

What really happens

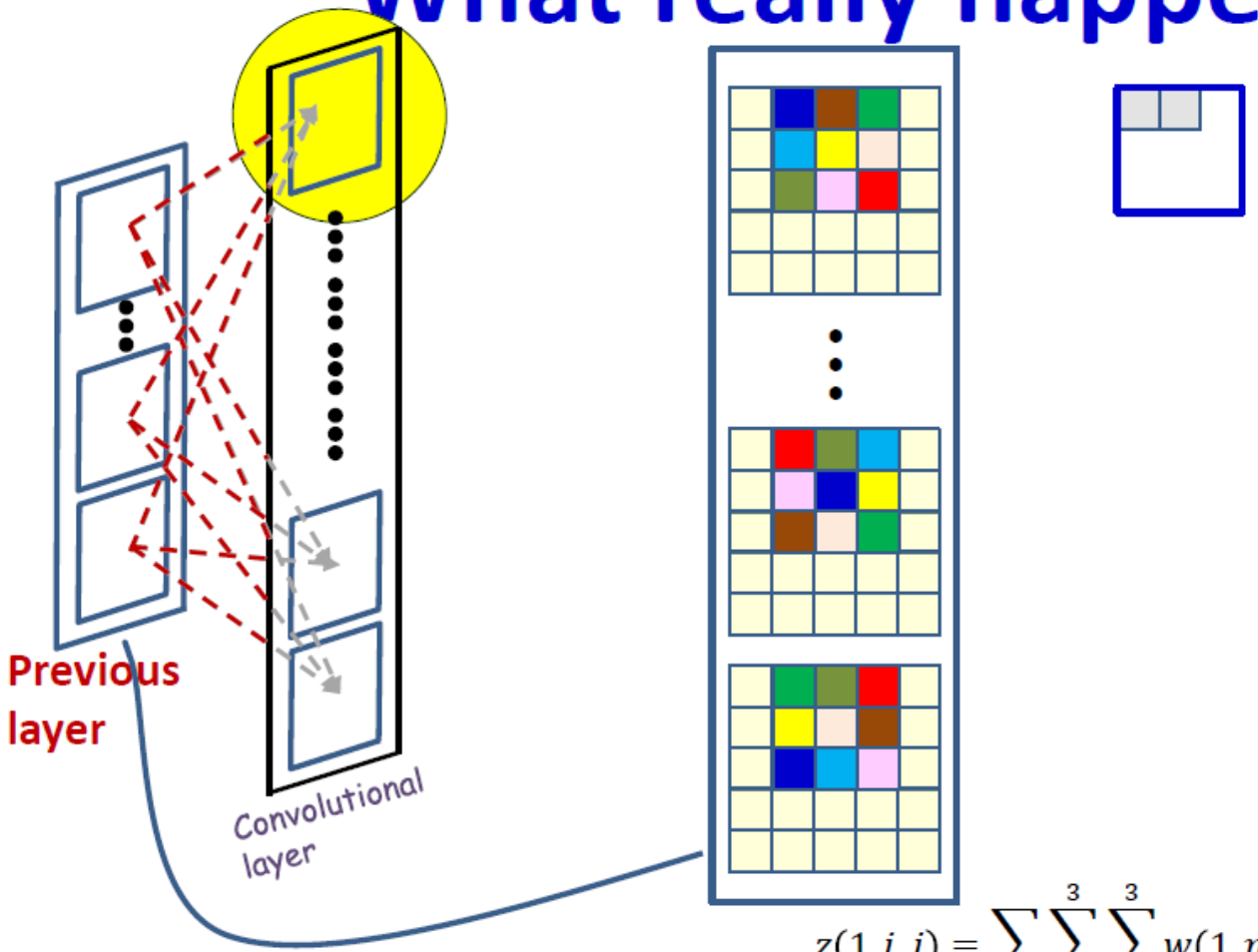


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

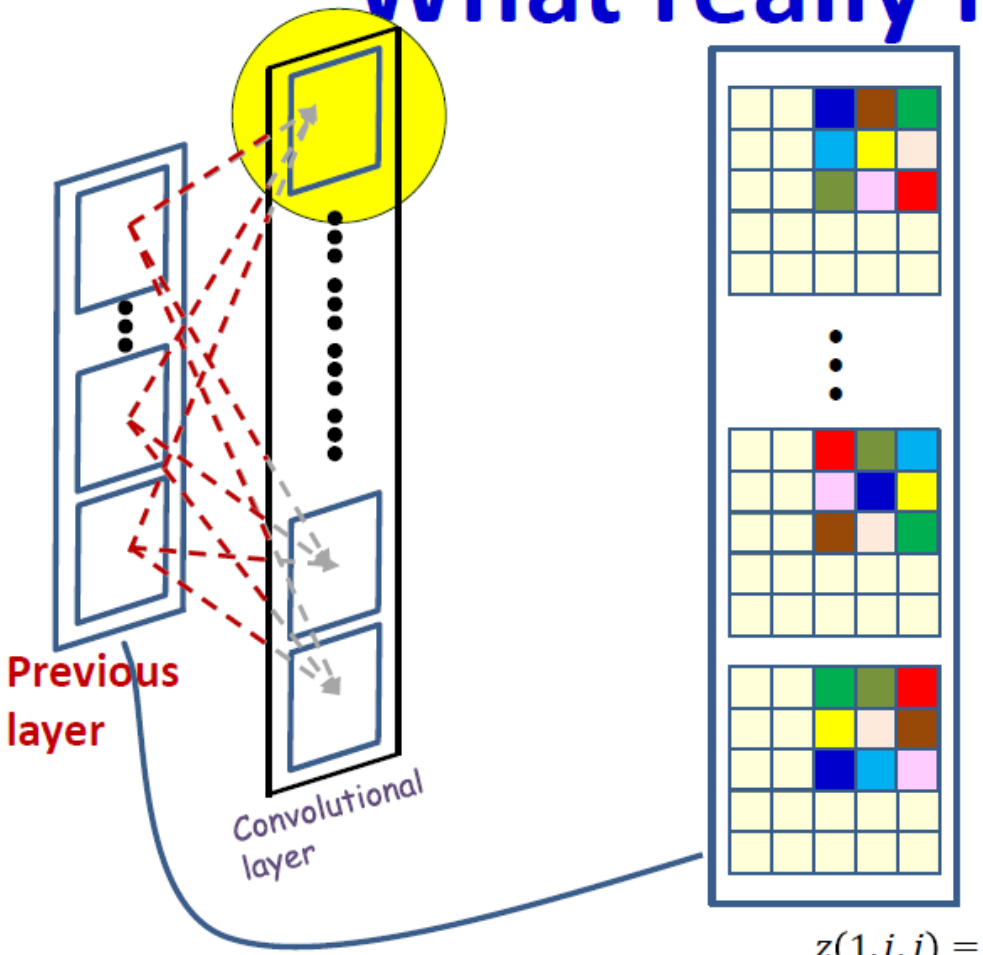
What really happens



$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

What really happens

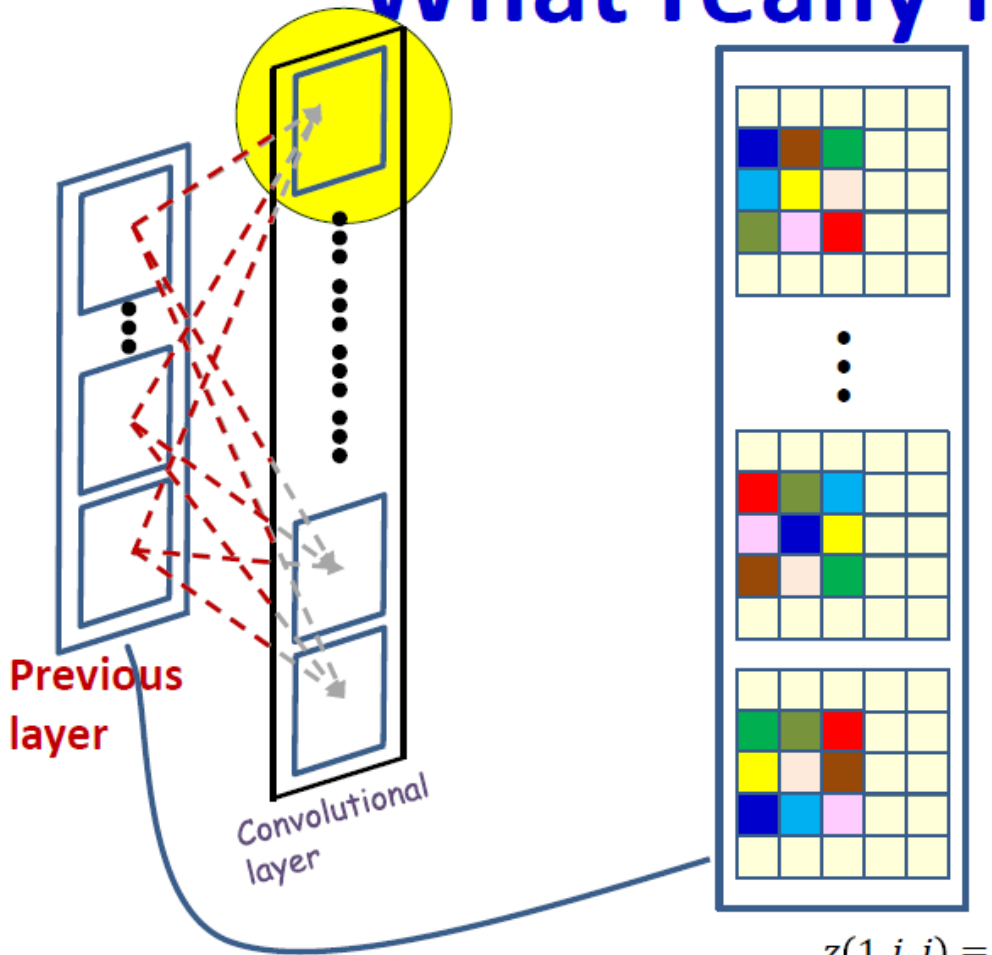


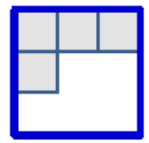
$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

What really happens

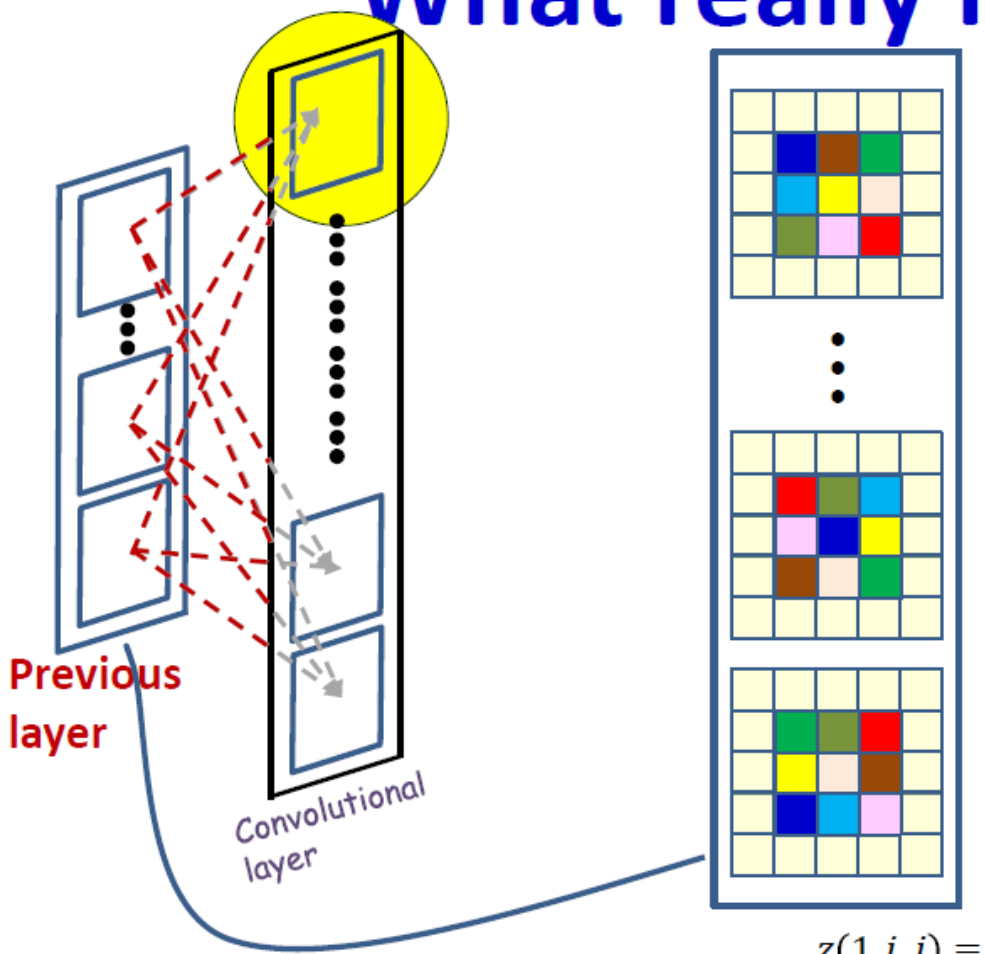


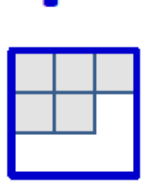


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

What really happens



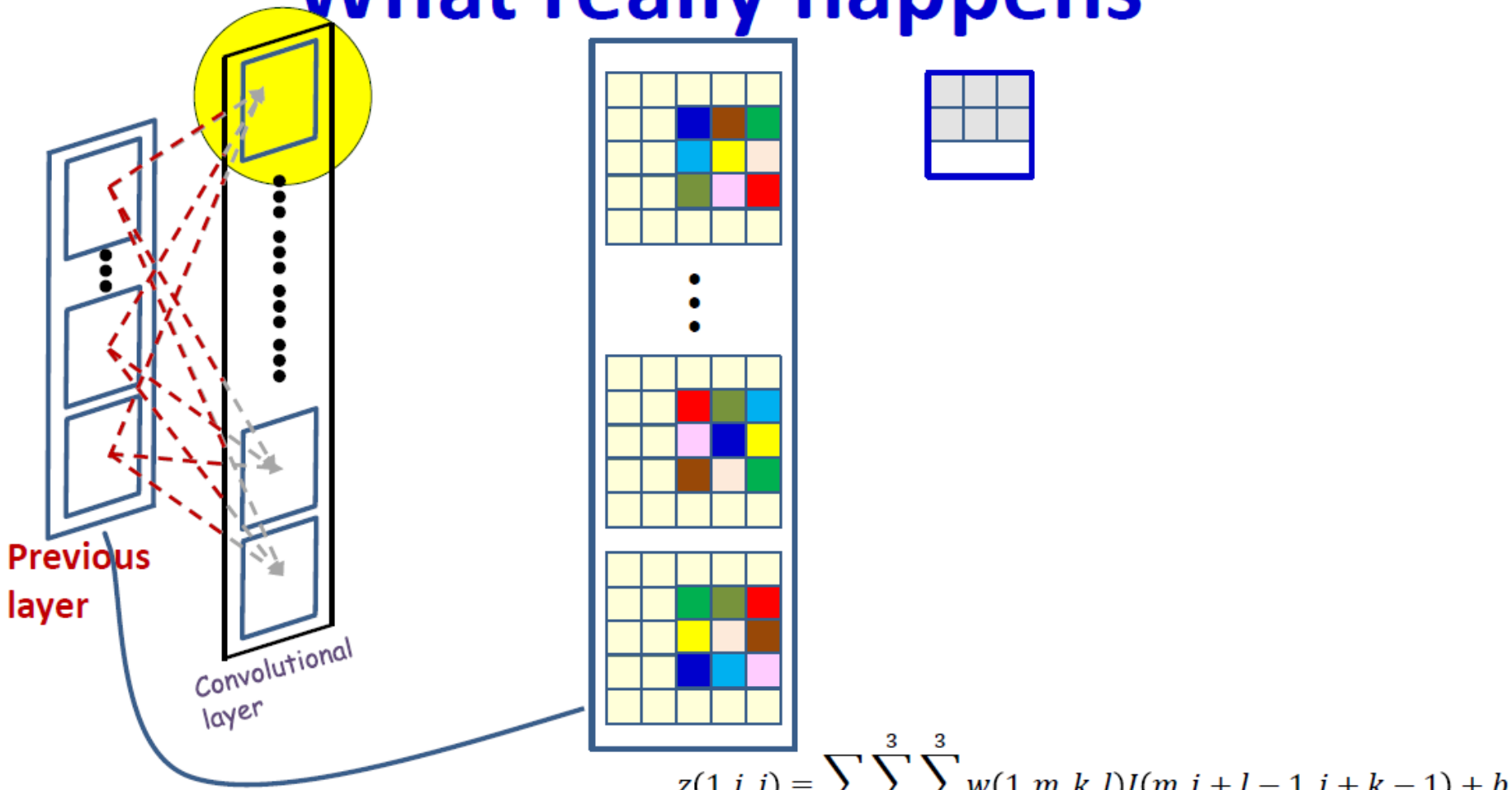


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

What really happens

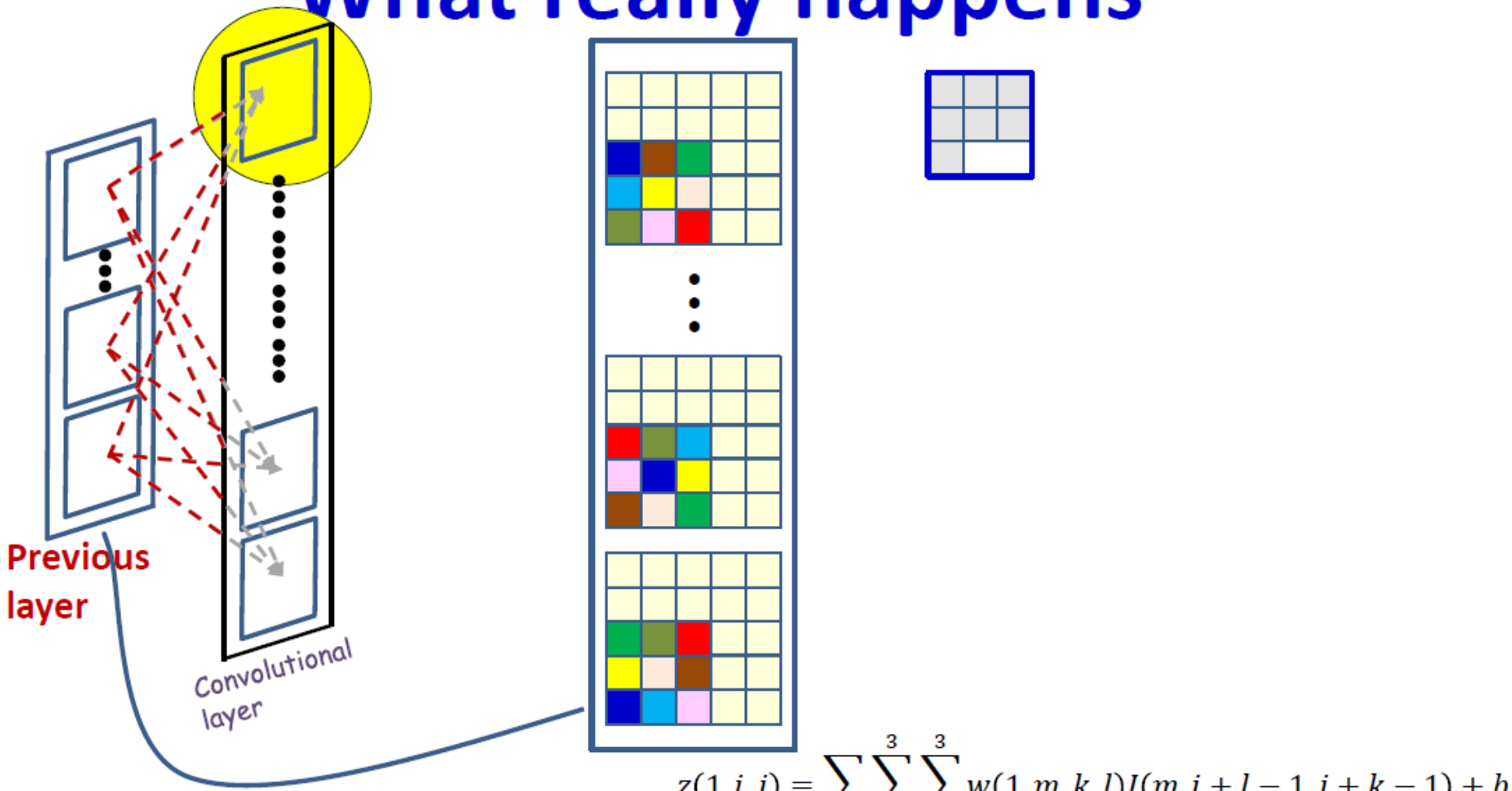


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

What really happens

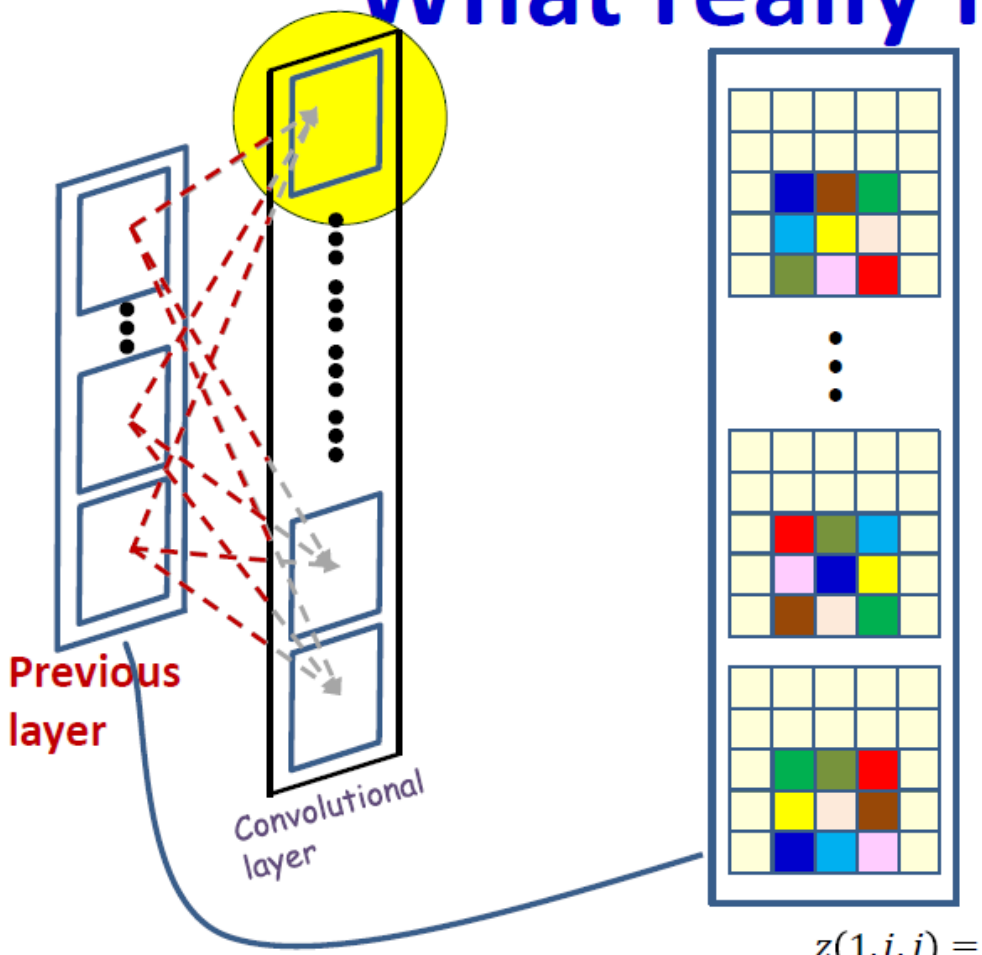


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

What really happens

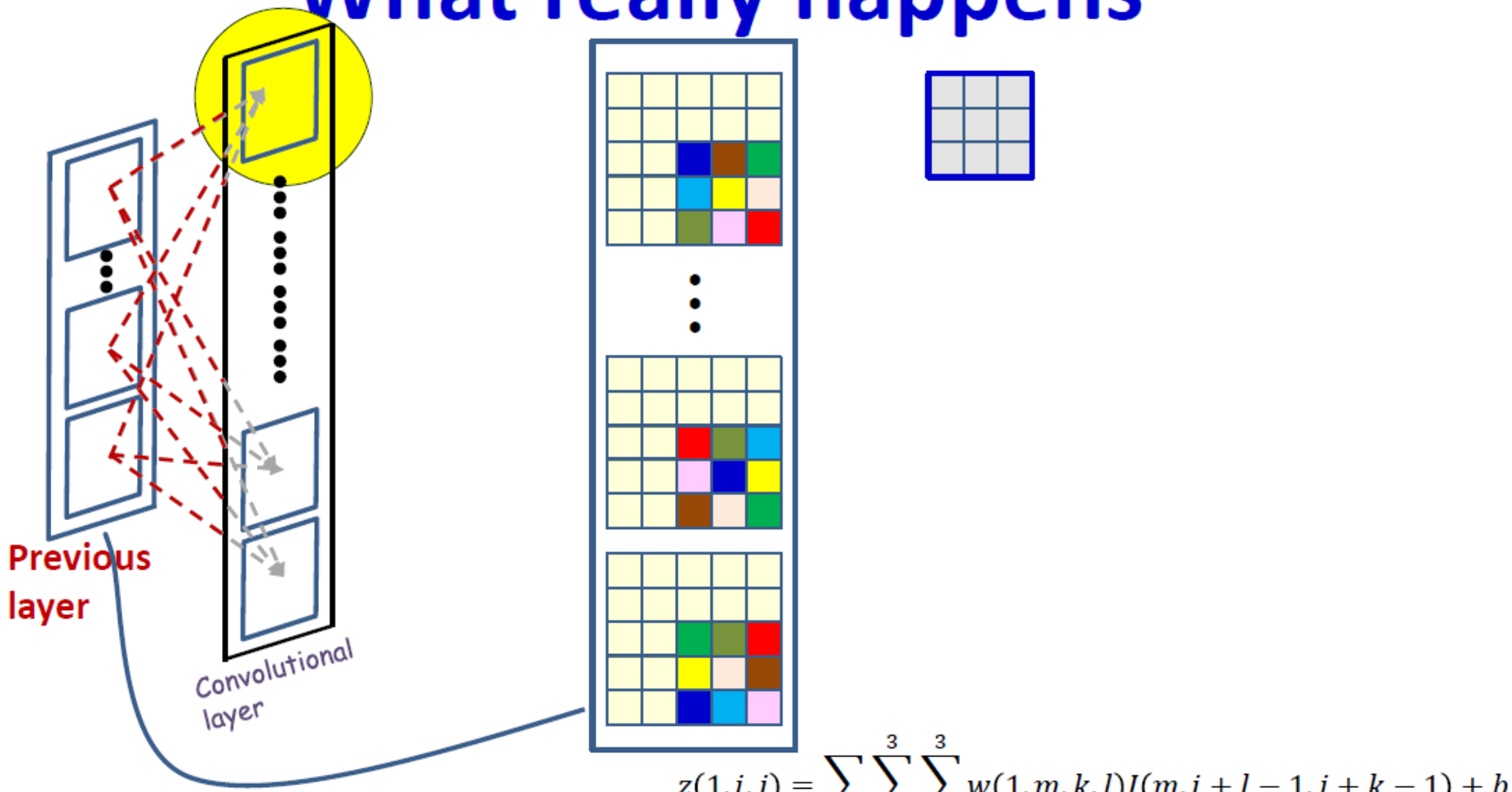


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

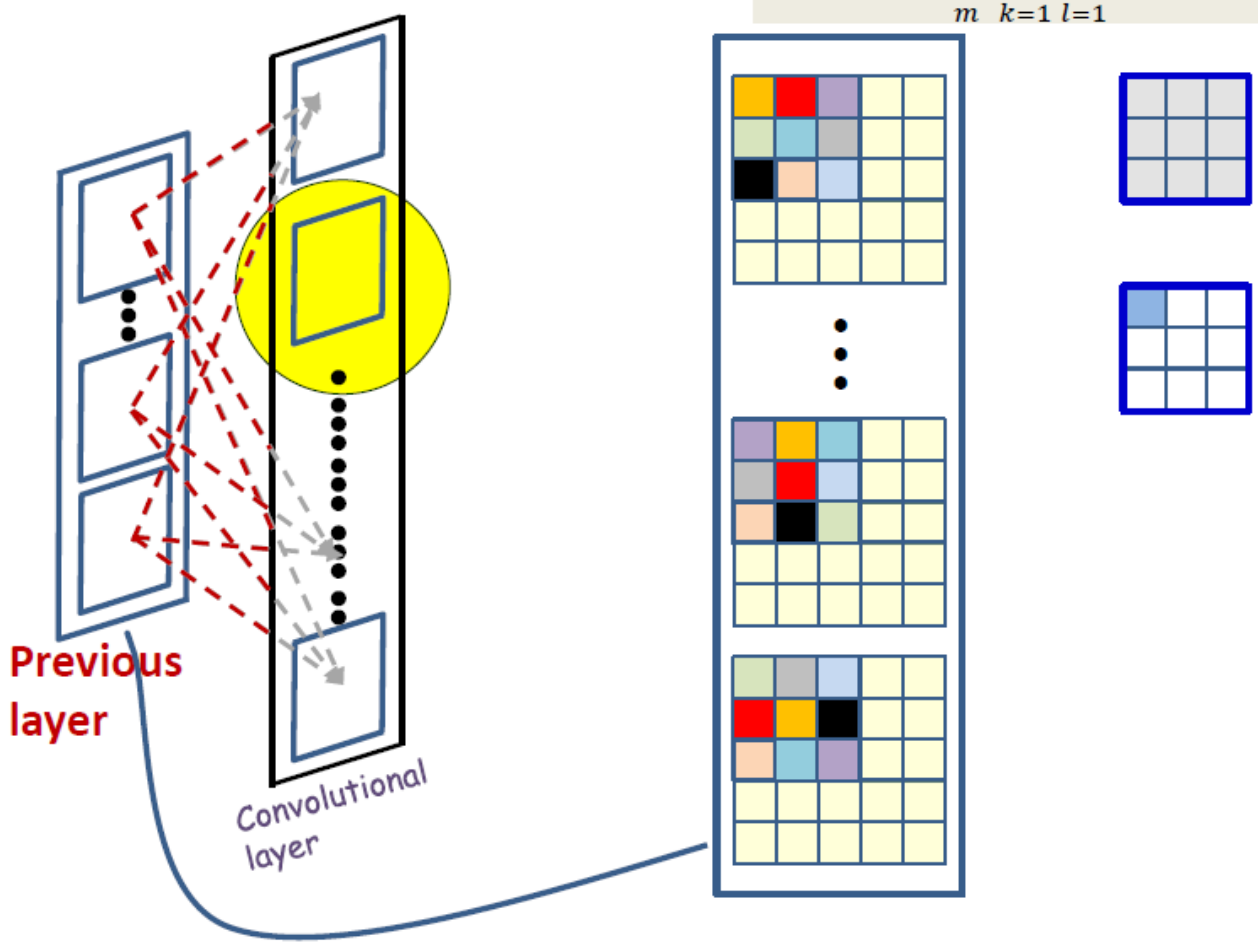
What really happens



$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

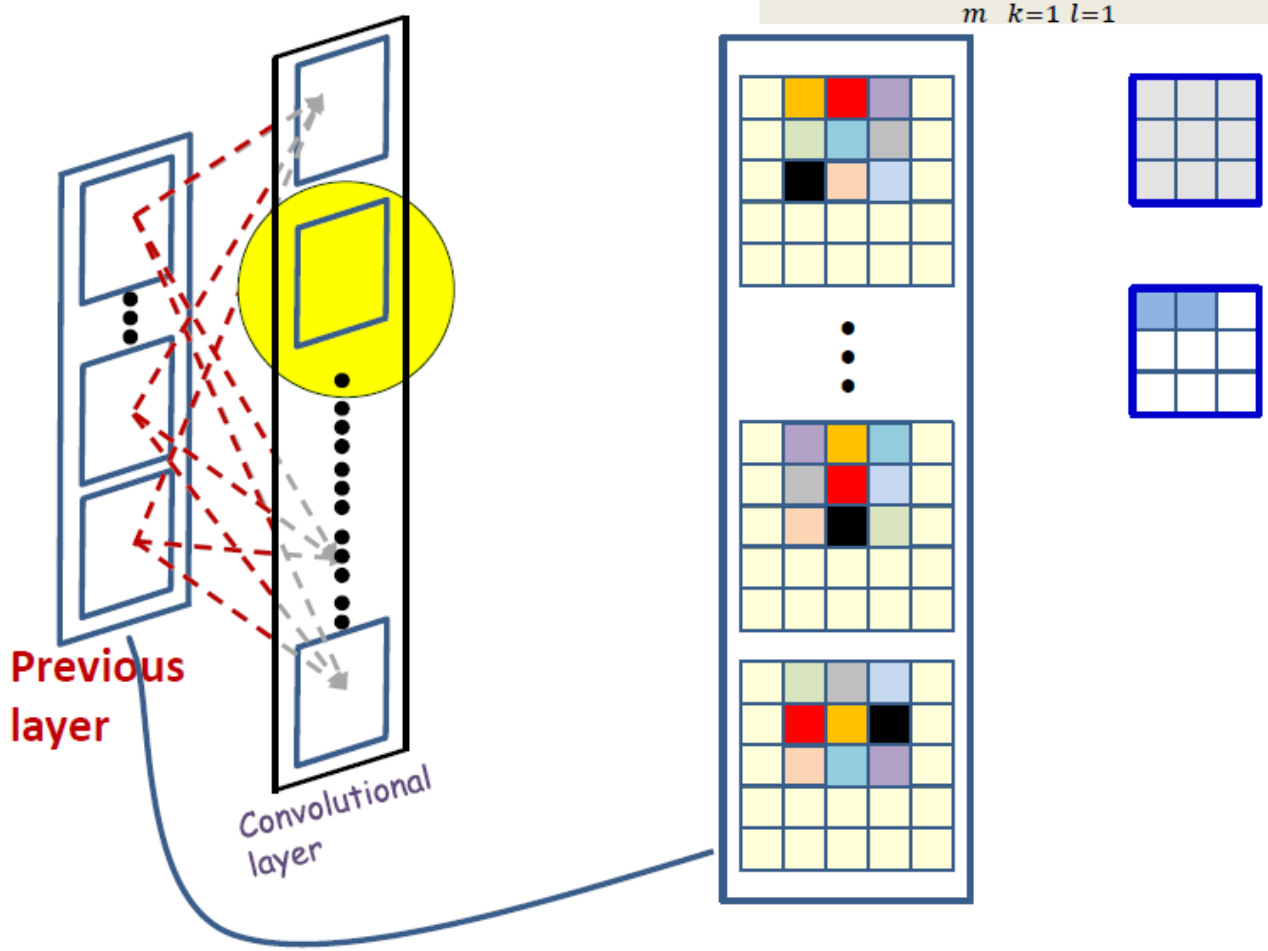
$$z(2,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(2,m,k,l) I(m, i+l-1, j+k-1) + b(2)$$



- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

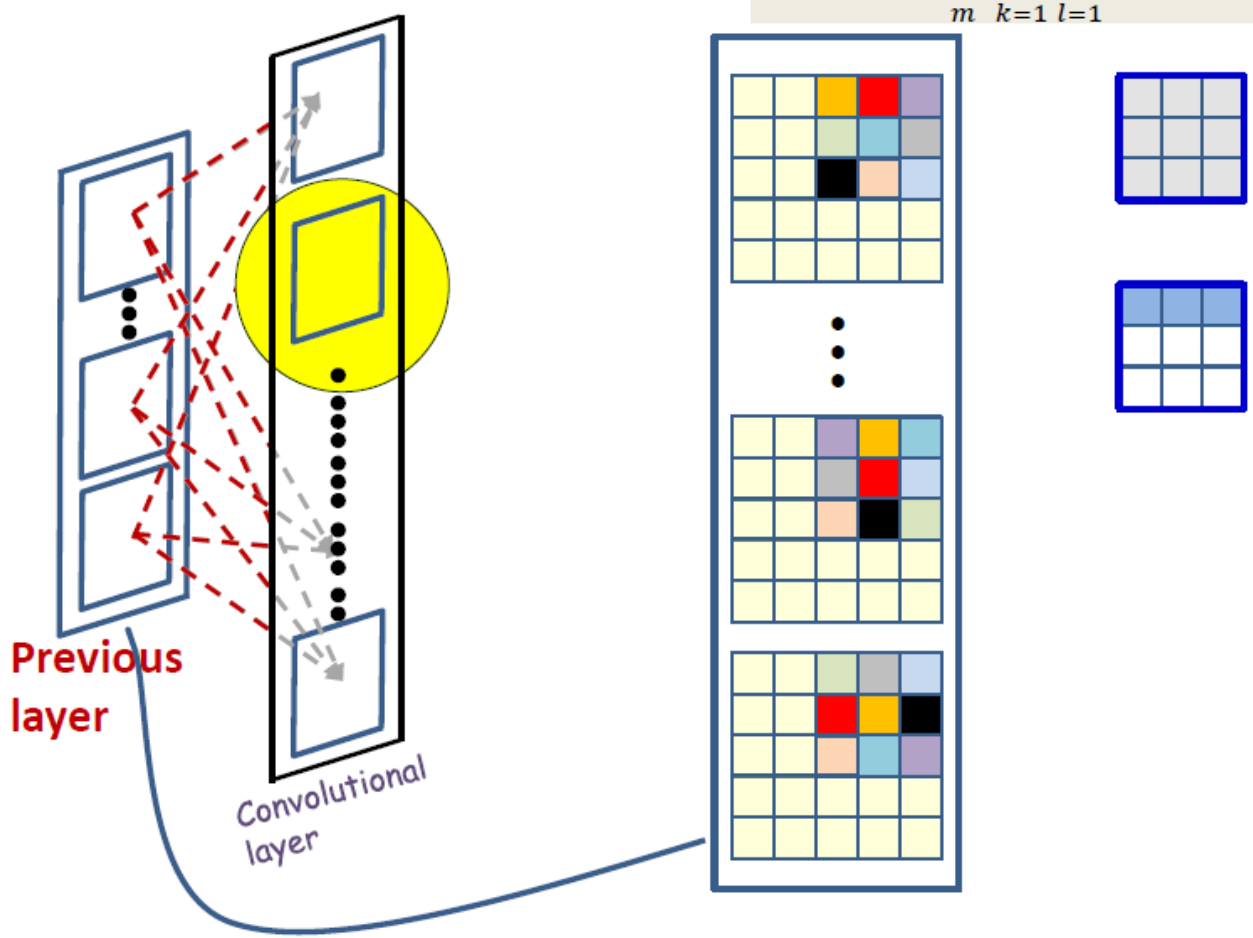
$$z(2,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(2,m,k,l) I(m, i+l-1, j+k-1) + b(2)$$



- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

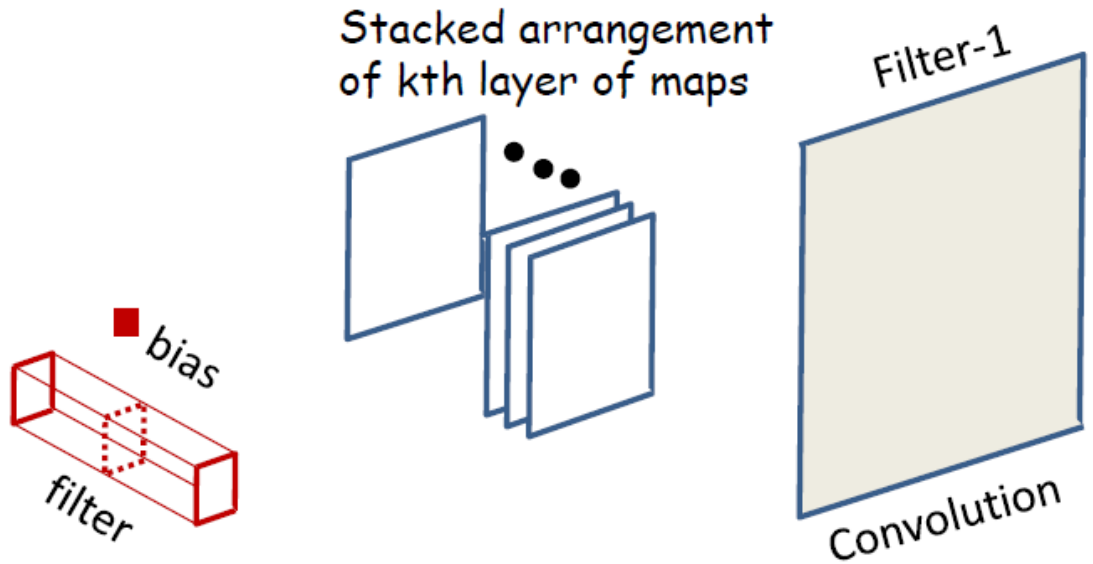
$$z(2,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(2,m,k,l) I(m, i+l-1, j+k-1) + b(2)$$



- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

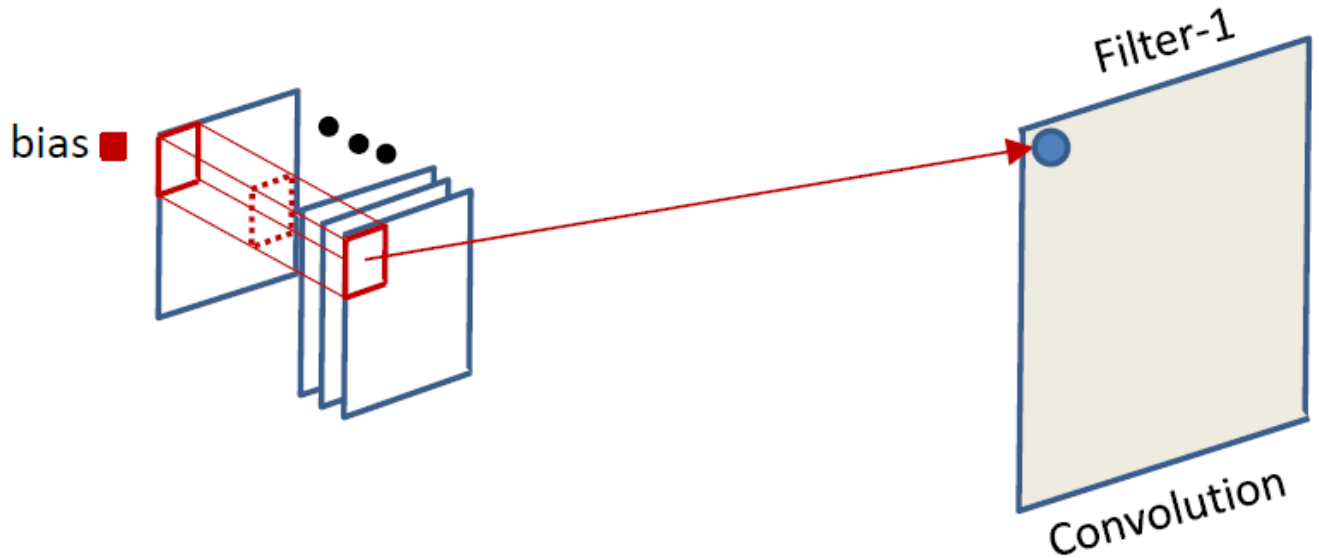
  size of the filter x *no. of maps in previous layer*

A different view



- ..A *stacked* arrangement of planes
- We can view the joint processing of the various maps as processing the stack using a three-dimensional filter

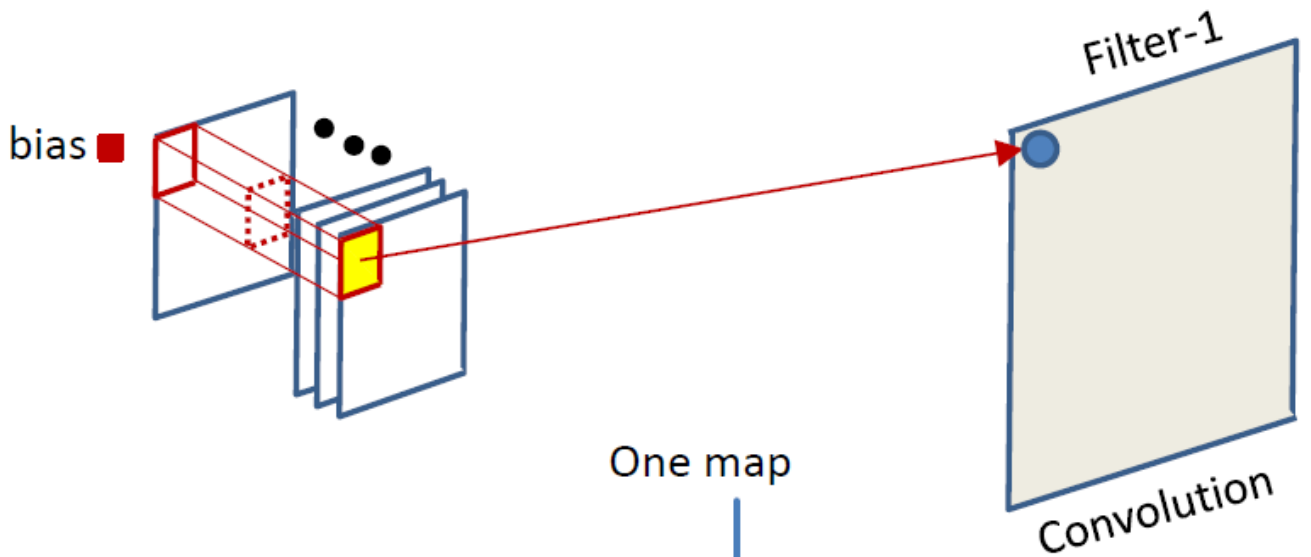
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

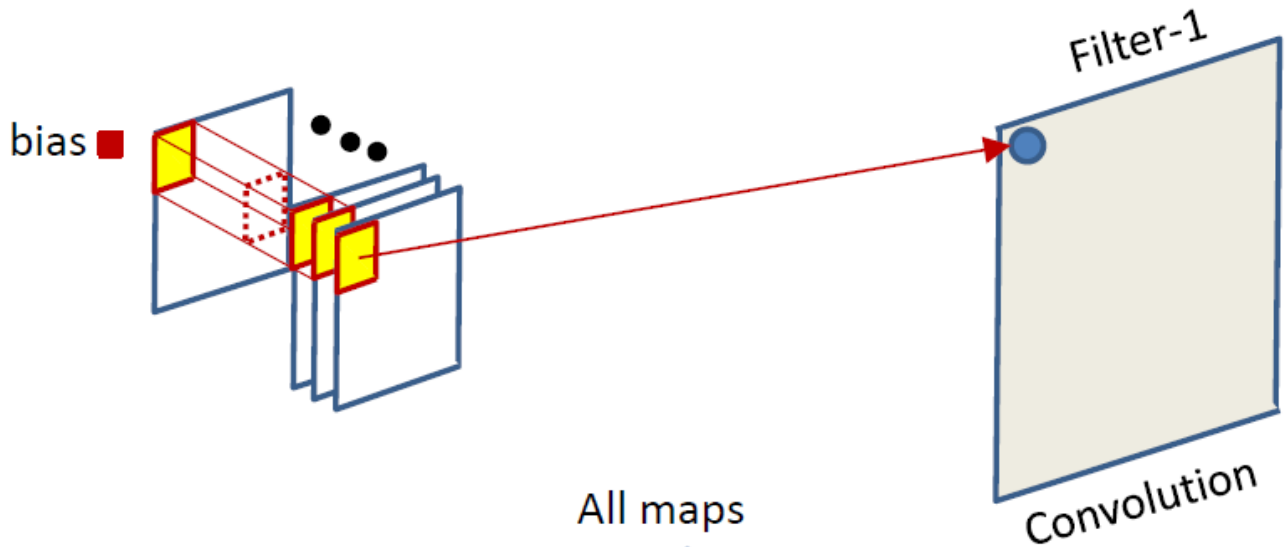
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

Extending to multiple input maps

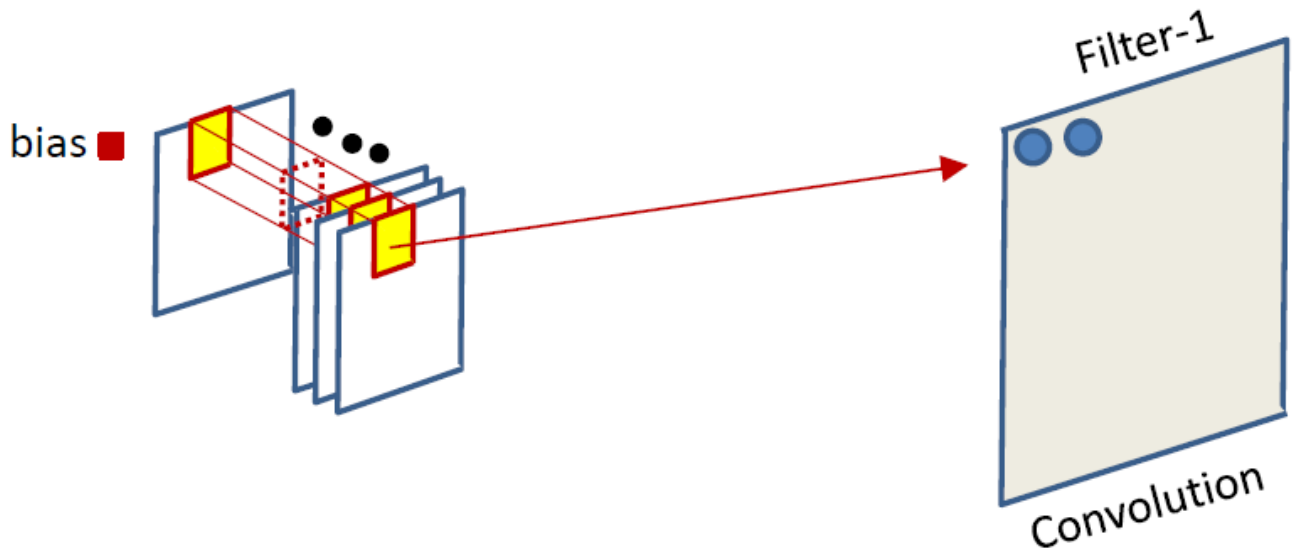


All maps

$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

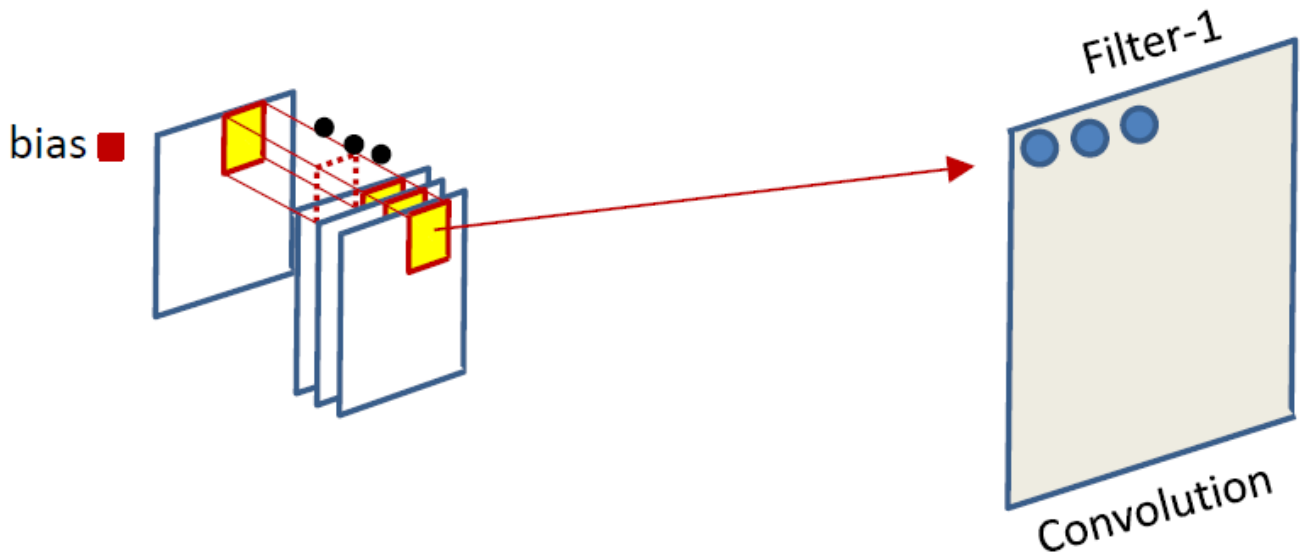
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

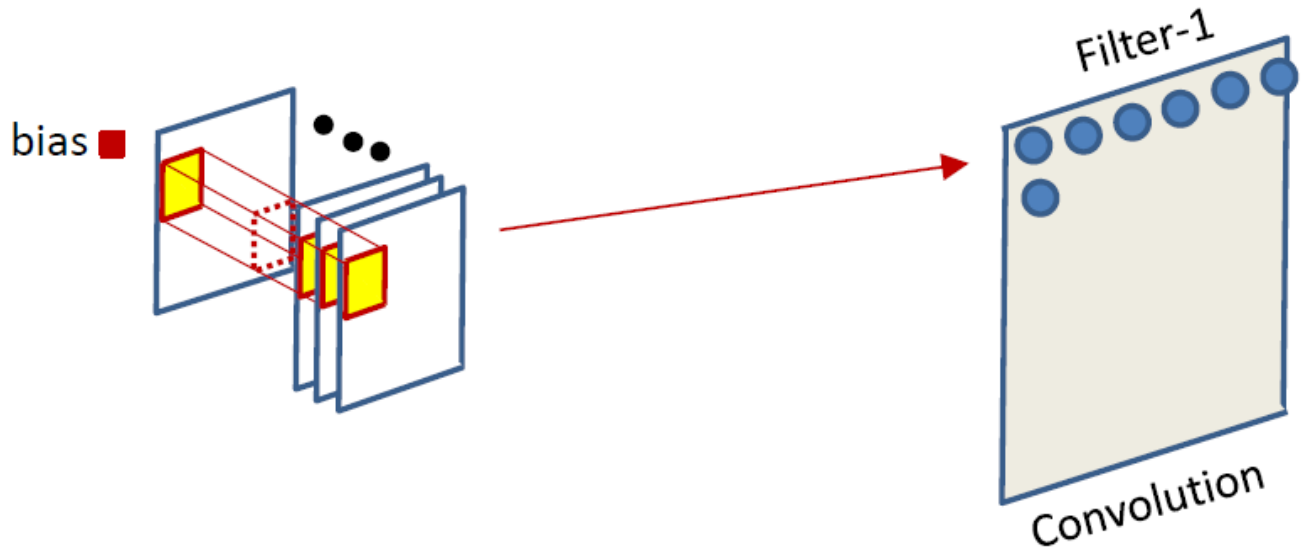
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

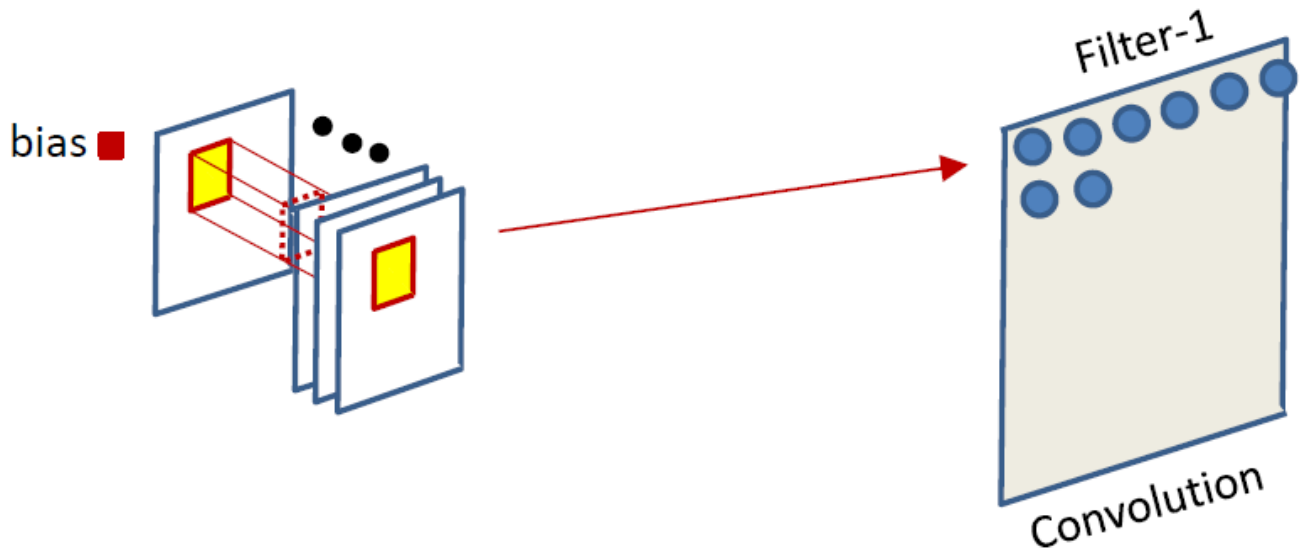
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

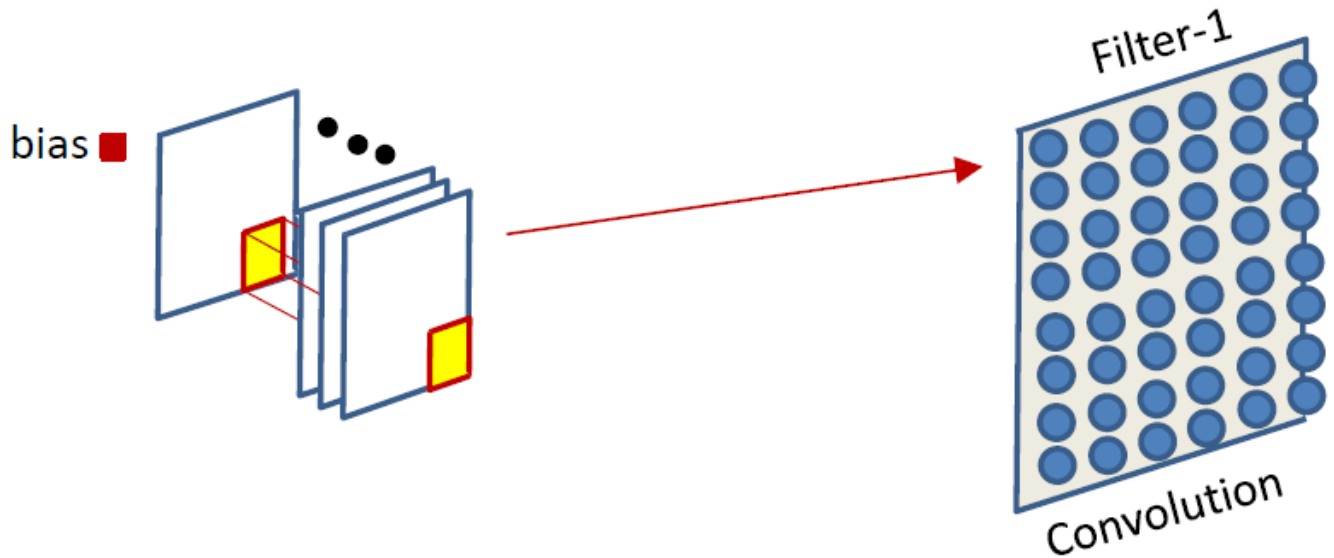
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

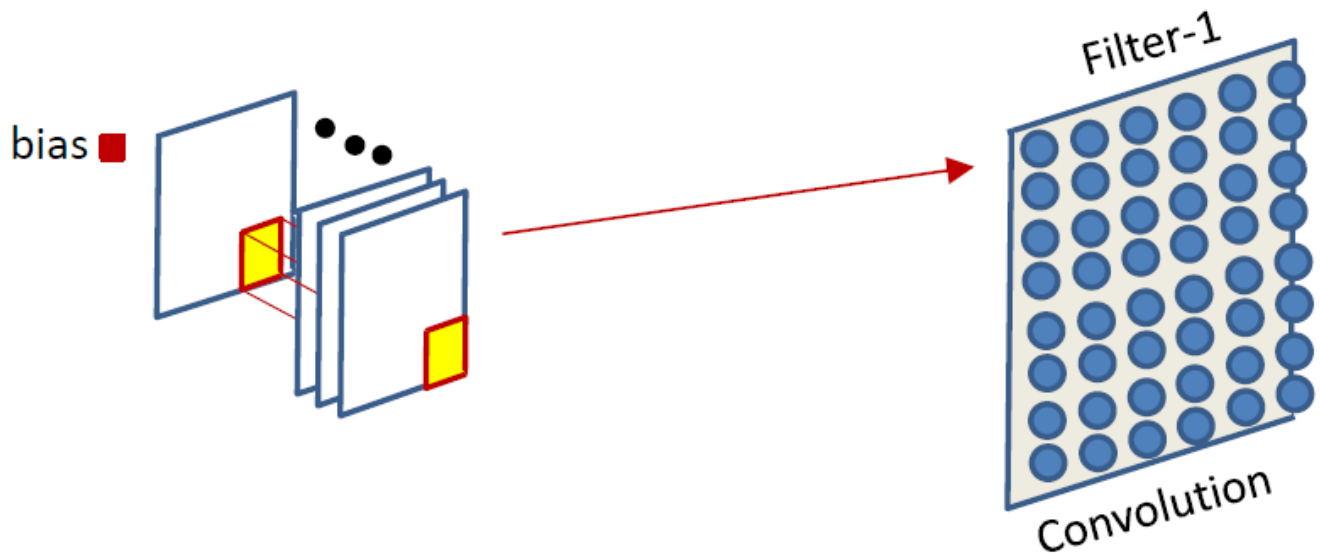
Extending to multiple input maps



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

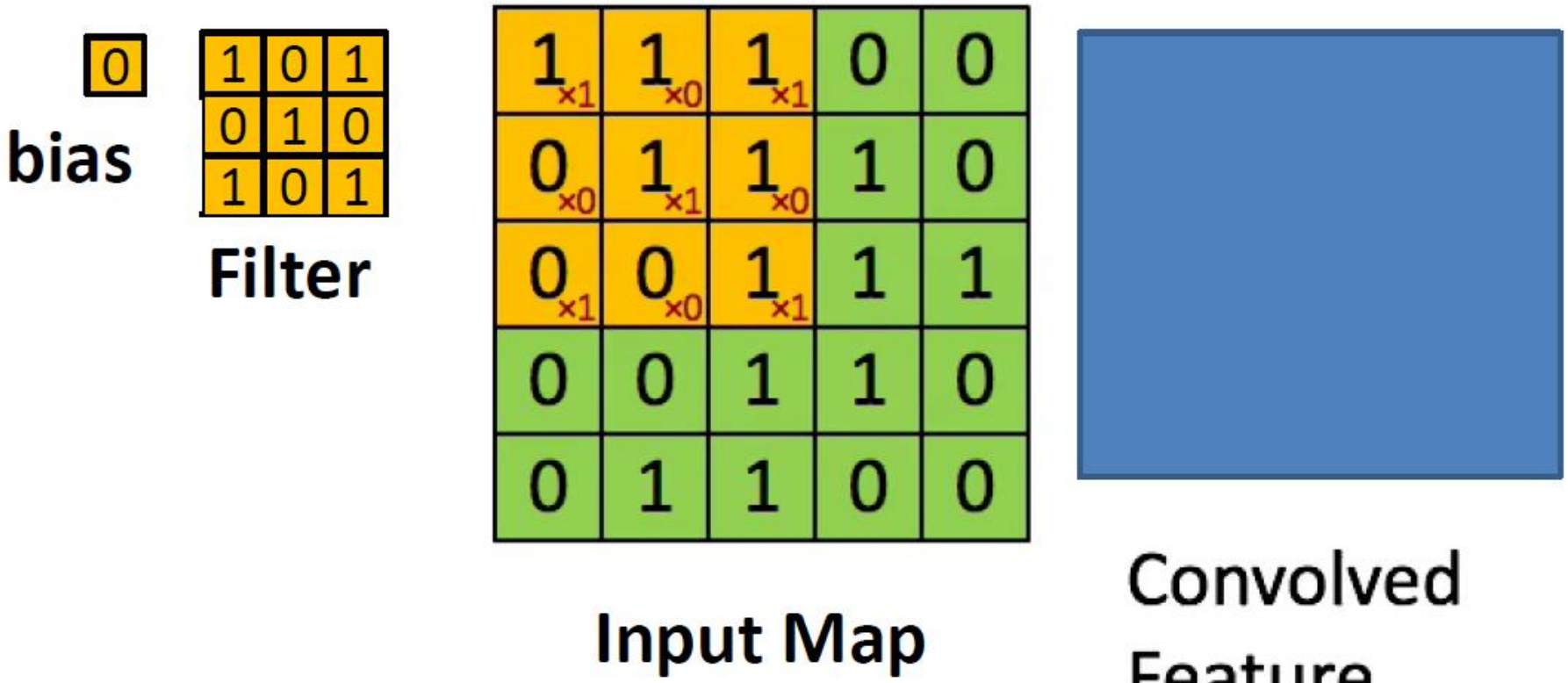
- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

Engineering consideration: The size of the result of the convolution



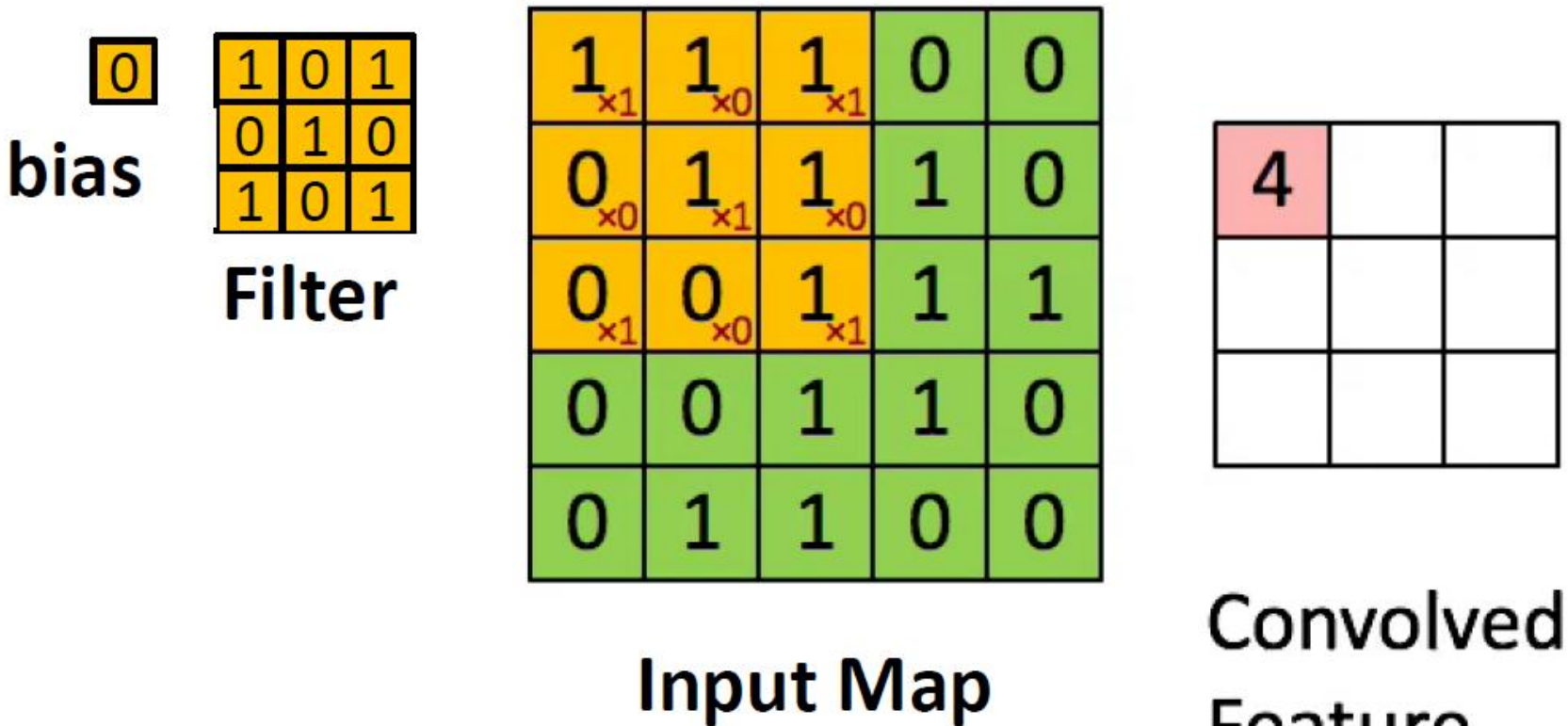
- **Recall**: the "stride" of the convolution may not be one pixel
  - I.e. the scanning neuron may "stride" more than one pixel at a time
- The size of the output of the convolution operation depends on implementation factors
  - And may not be identical to the size of the input
  - Lets take a brief look at this for completeness sake

The size of the convolution



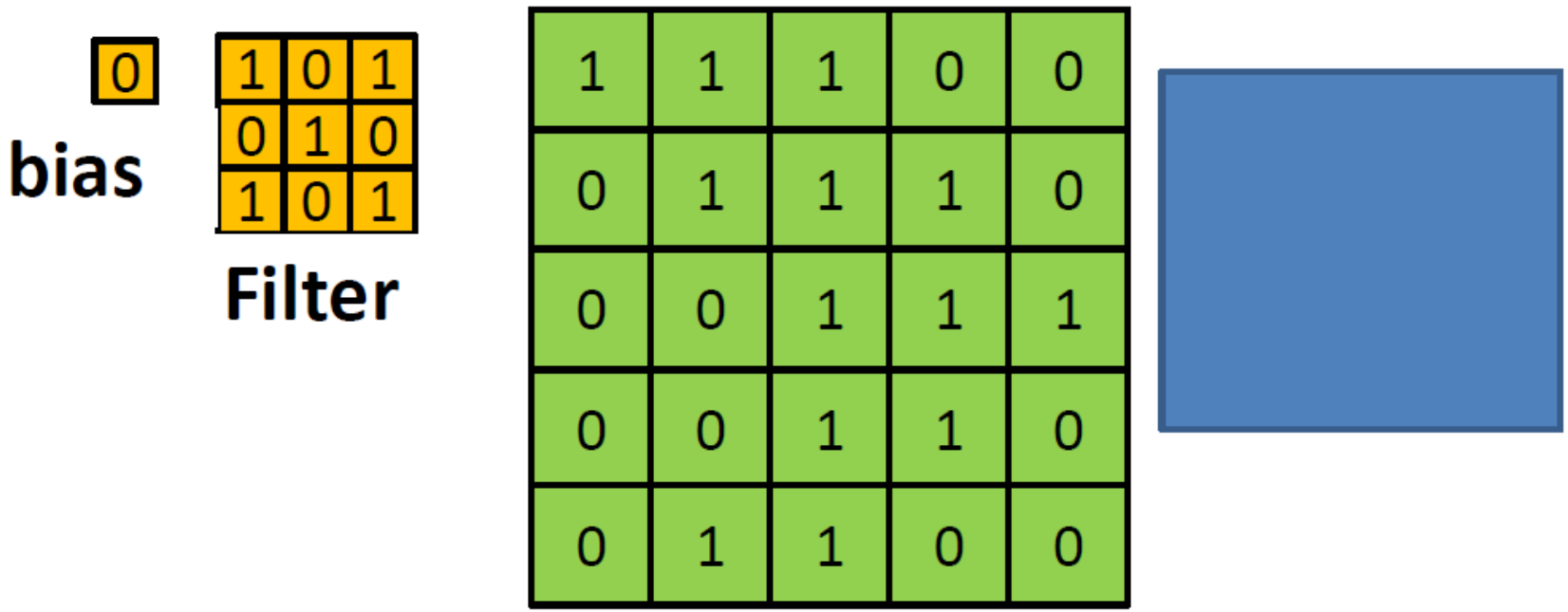
- Image size: 5x5
- Filter: 3x3
- “Stride”: 1
- Output size = ?

The size of the convolution



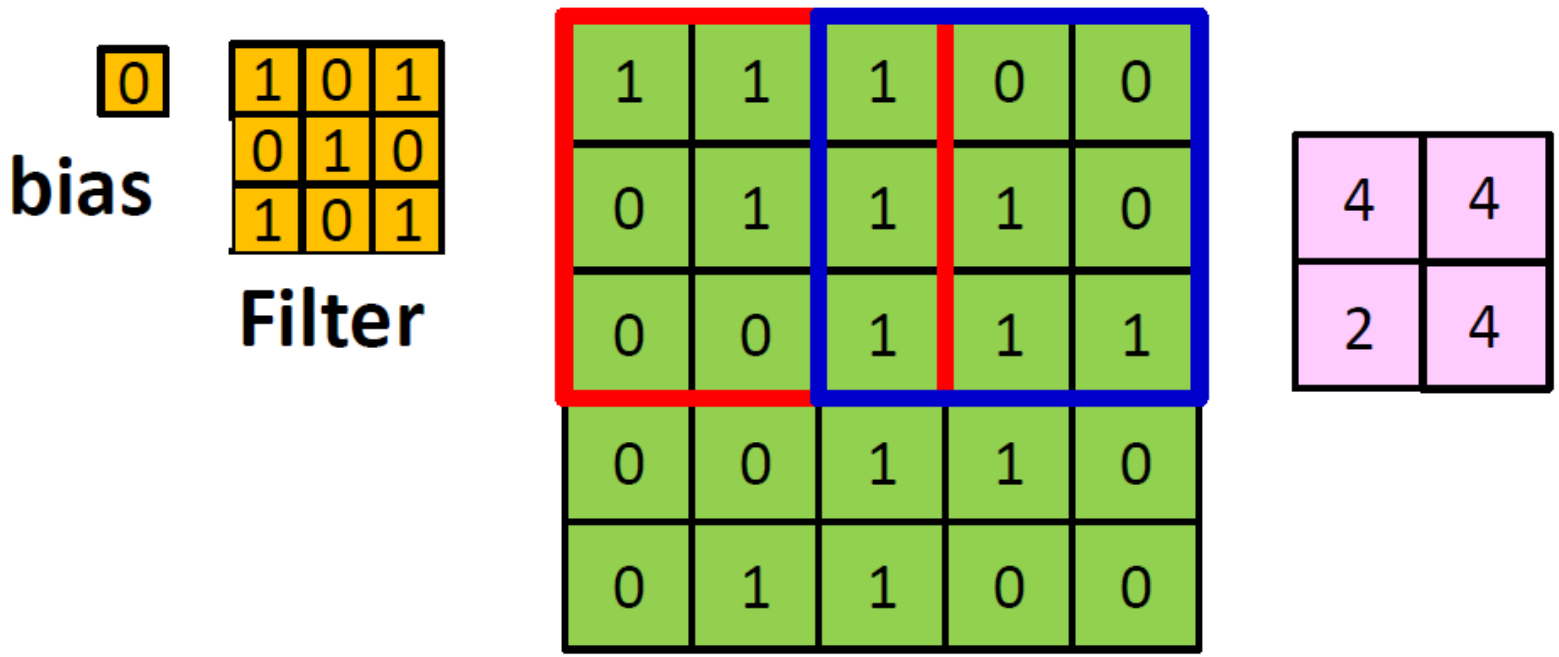
- Image size: 5x5
- Filter: 3x3
- Stride: 1
- Output size = ?

The size of the convolution



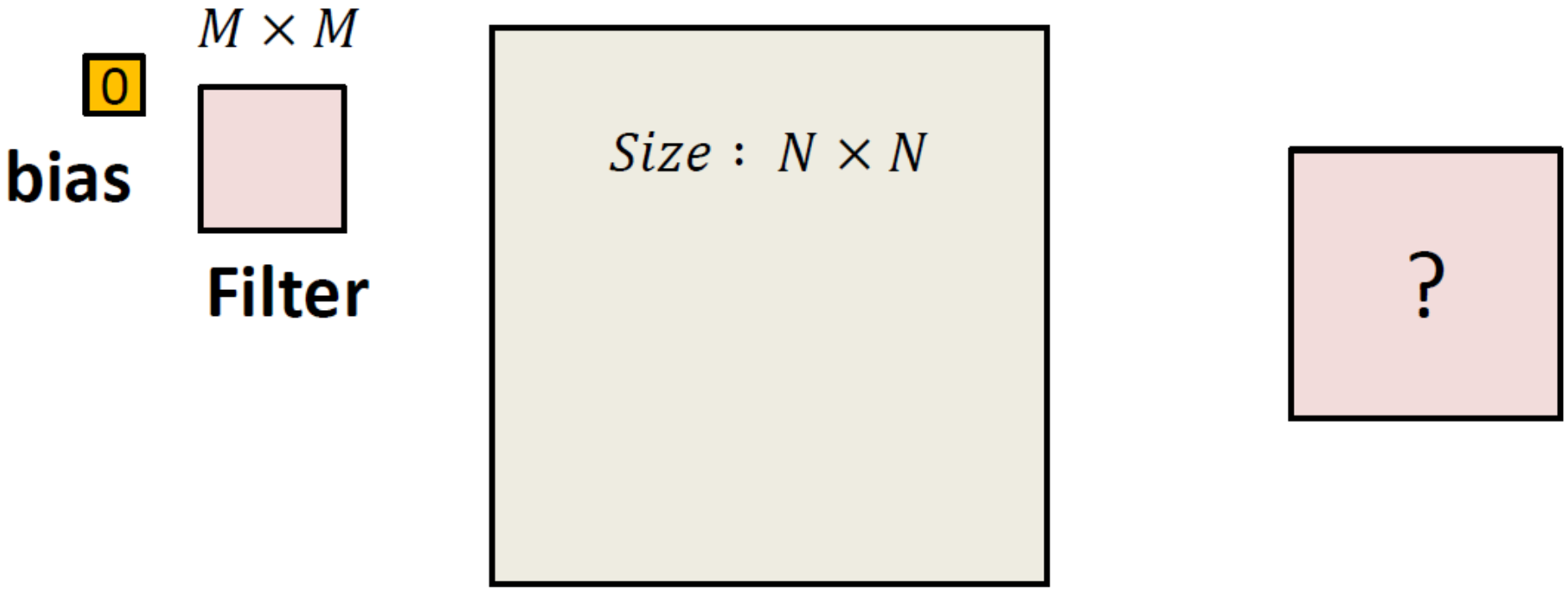
- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

The size of the convolution



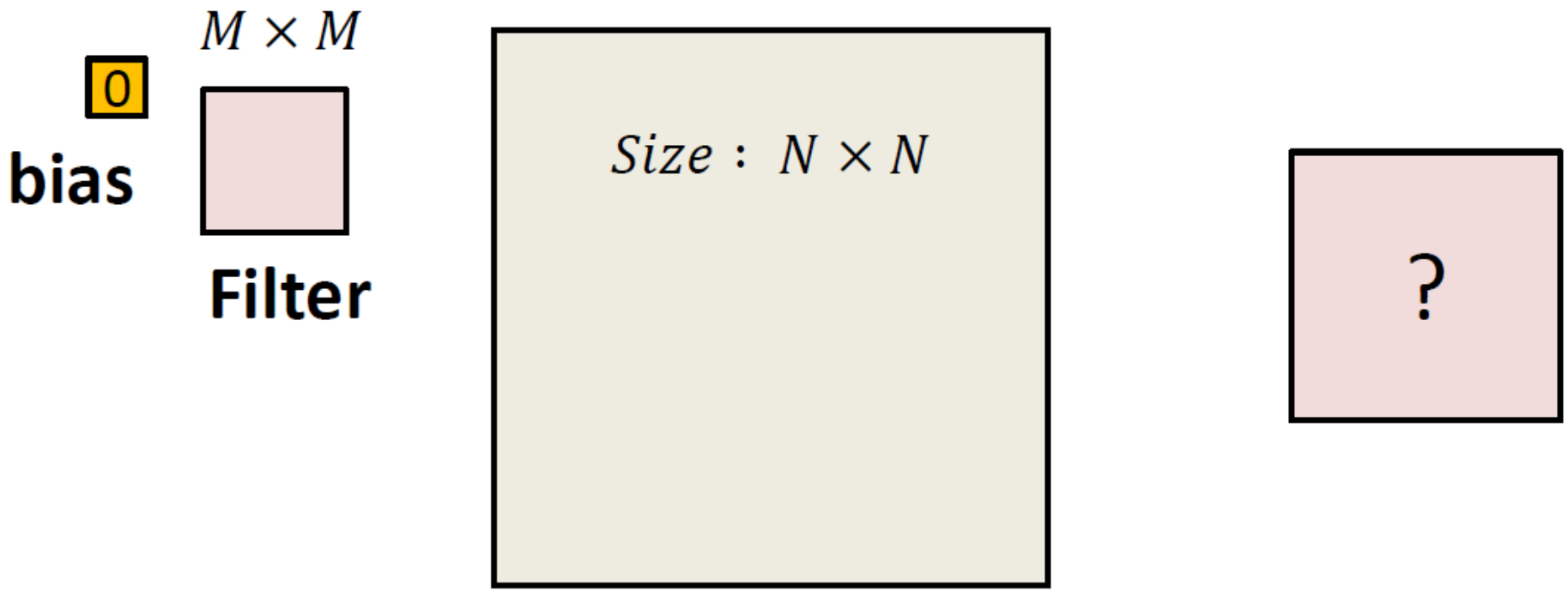
- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

The size of the convolution



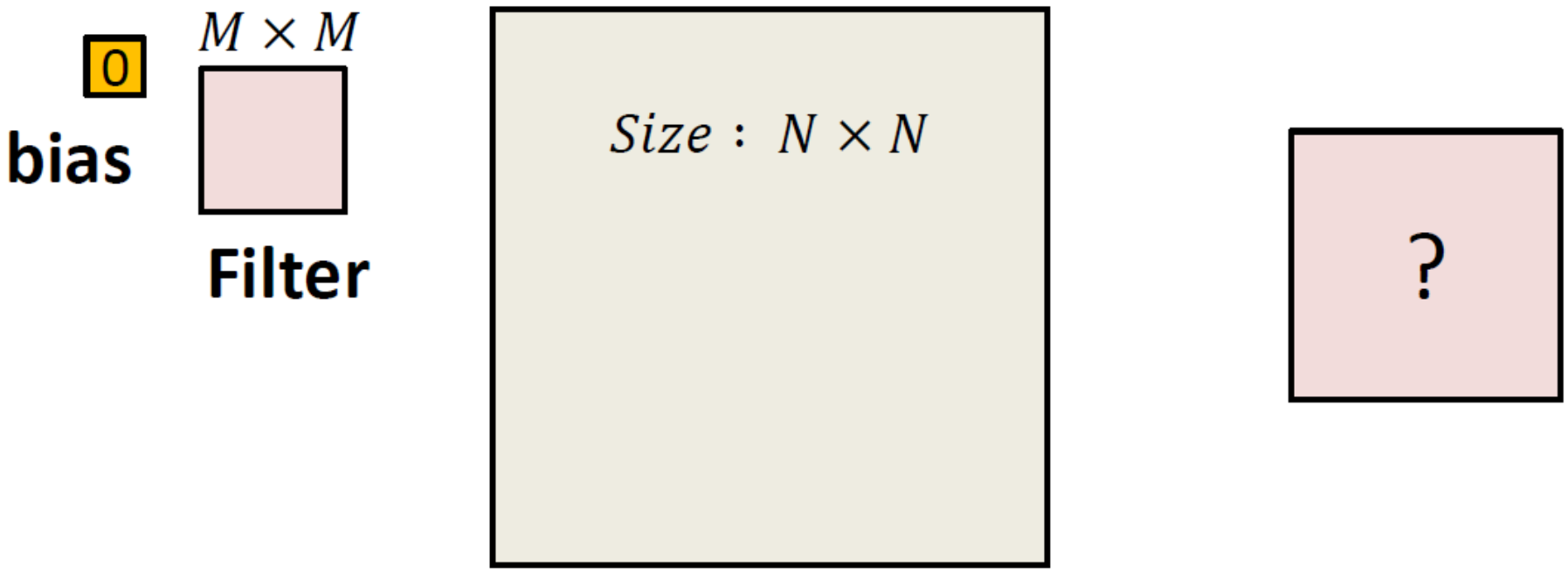
- Image size: $N \times N$
- Filter: $M \times M$
- Stride: 1
- Output size = ?

The size of the convolution



- Image size: $N \times N$
- Filter: $M \times M$
- Stride: S
- Output size = ?

The size of the convolution



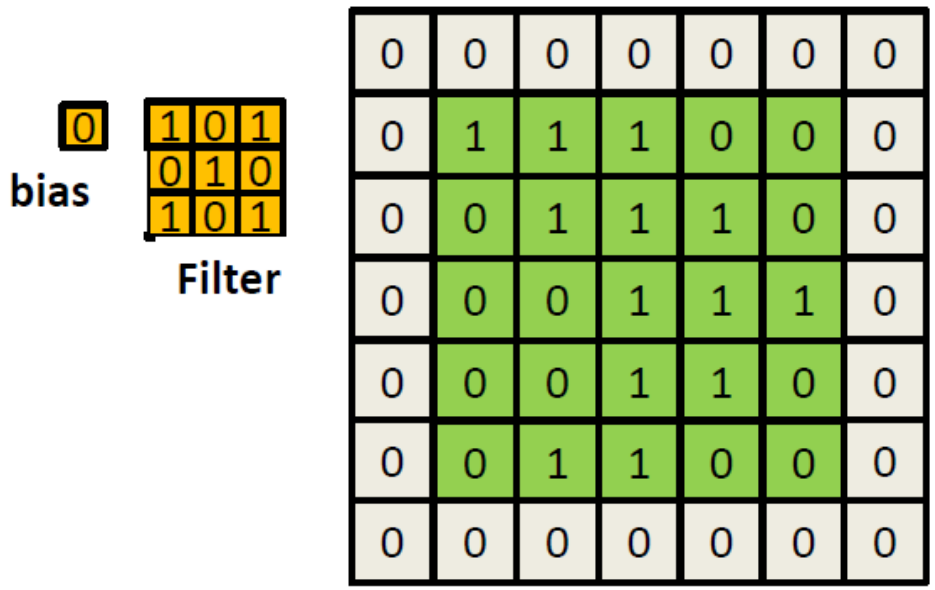
- Image size: $N \times N$
- Filter: $M \times M$
- Stride: S
- Output size (each side) = $\lfloor (N - M)/S \rfloor + 1$
  - Assuming you're not allowed to go beyond the edge of the input

Convolution Size

- Simple convolution size pattern:
  - Image size: $N \times N$
  - Filter: $M \times M$
  - Stride: S
  - **Output size (each side)** = $\lfloor (N - M)/S \rfloor + 1$
    - Assuming you're not allowed to go beyond the edge of the input

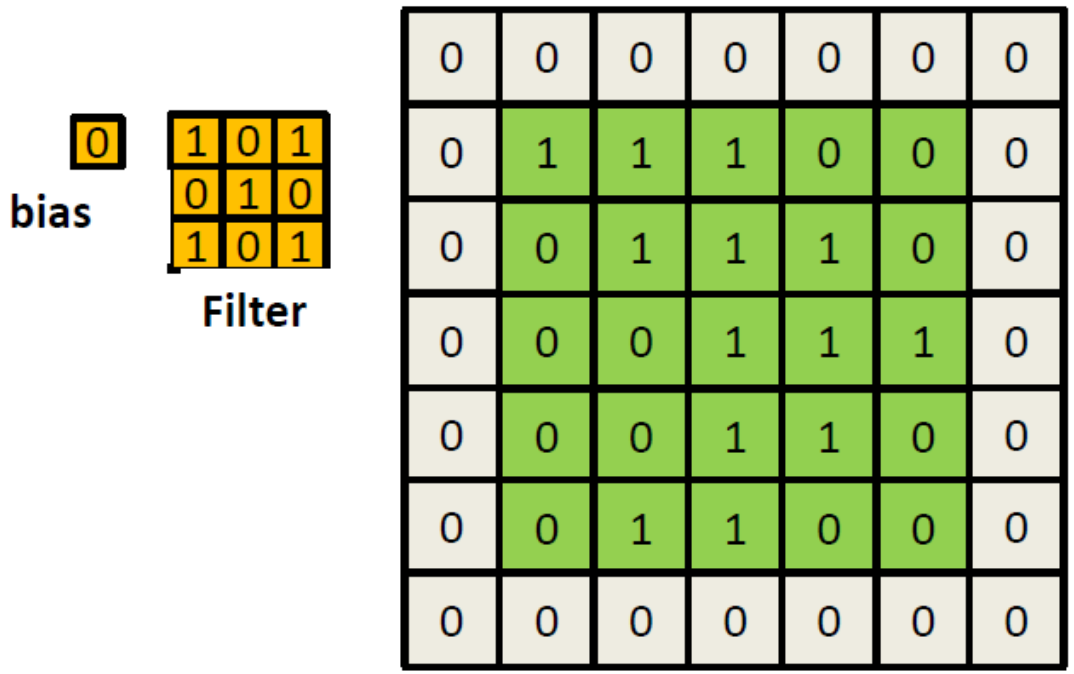
- Results in a reduction in the output size
  - Even if $S = 1$
  - Sometimes not considered acceptable
    - If there's no active downsampling, through max pooling and/or $S > 1$, then the output map should ideally be the same size as the input

Solution



- Zero-pad the input
  - Pad the input image/map all around
    - Add P_L rows of zeros on the left and P_R rows of zeros on the right
    - Add P_L rows of zeros on the top and P_L rows of zeros at the bottom
  - P_L and P_R chosen such that:
    - $P_L = P_R$ OR $|P_L - P_R| = 1$
    - $P_L + P_R = M-1$
      - **For stride 1, the result of the convolution is the same size as the original image**

Solution



- Zero-pad the input
  - Pad the input image/map all around
  - Pad as symmetrically as possible, such that..
  - **For stride 1, the result of the convolution is the same size as the original image**

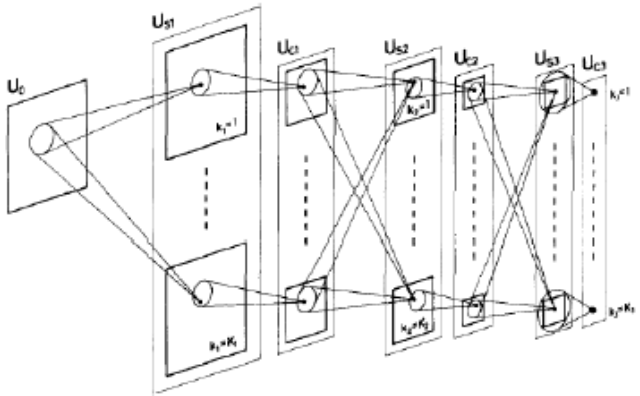
Zero padding

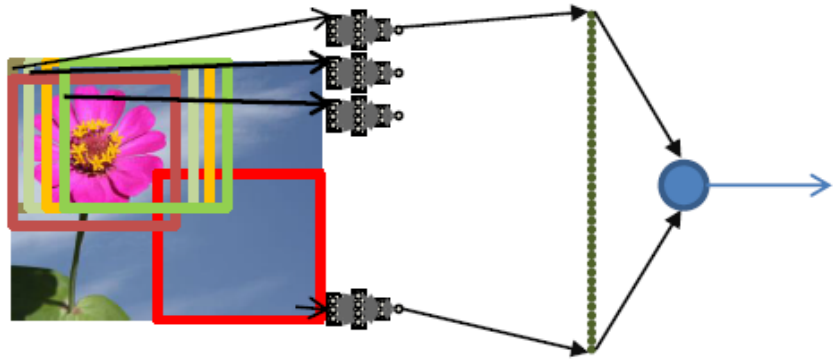
- For an L width filter:
  - Odd L : Pad on both left and right with $(L-1)/2$ columns of zeros
  - Even L : Pad one side with $L/2$ columns of zeros, and the other with $\frac{L}{2}-1$ columns of zeros
  - The resulting image is width $N+L-1$
  - The result of the convolution is width N

- The top/bottom zero padding follows the same rules to maintain map height after convolution

- For hop size $S>1$, zero padding is adjusted to ensure that the size of the convolved output is $\lceil N/S \rceil$
  - Achieved by *first* zero padding the image with $S\lceil N/S \rceil - N$ columns/rows of zeros and then applying above rules

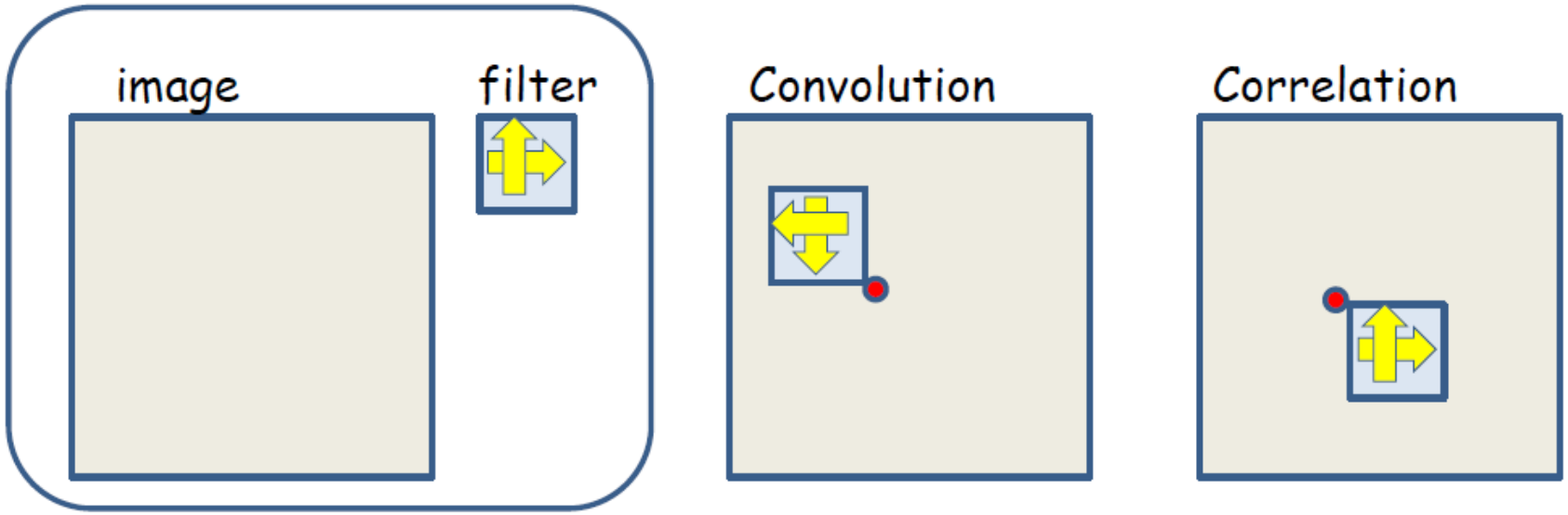
Why convolution?





- Convolutional neural networks are, in fact, equivalent to *scanning* with an MLP
  - Just run the entire MLP on each block separately, and combine results
    - As opposed to scanning (convolving) the picture with individual neurons/filters
  - Even computationally, the number of operations in both computations is identical
    - The neocognitron in fact views it equivalently to a scan
- So why convolutions?

Correlation, not Convolution



- The operation performed is technically a correlation, not a convolution
- **Correlation**:

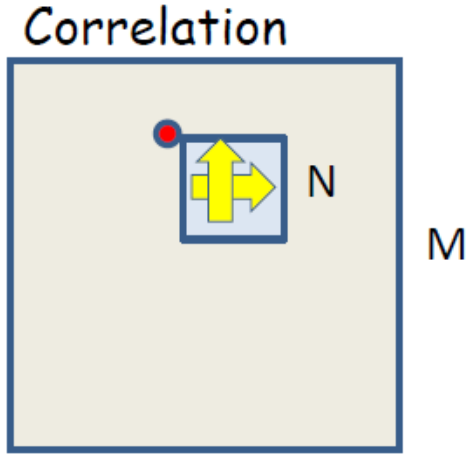
$$y(i,j) = \sum_l \sum_m x(i+l, j+m) w(l,m)$$

  - Shift the "filter" w to "look" at the input x block *beginning* at (i,j)
- **Convolution**:

$$y(i,j) = \sum_l \sum_m x(i-l, j-m) w(l,m)$$

- Effectively "flip" the filter, right to left, top to bottom

Cost of Correlation

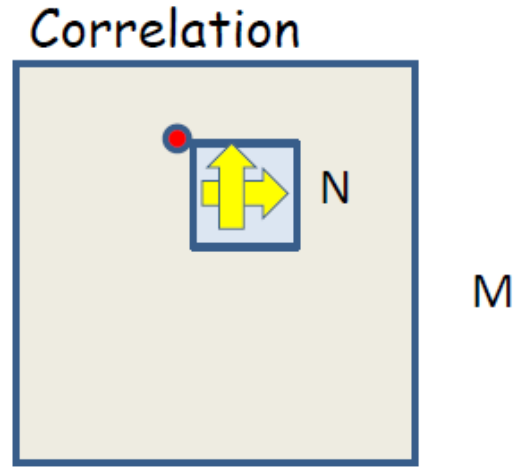


- **Correlation**:

$$y(i,j) = \sum_{l}\sum_{m} x(i+l, j+m)w(l,m)$$

- Cost of scanning an $M \times M$ image with an $N \times N$ filter: O(M^2N^2)
  - N^2 multiplications at each of M^2 positions
    - Not counting boundary effects
  - Expensive, for large filters

Correlation in Transform Domain



- **Correlation usind DFTs**:

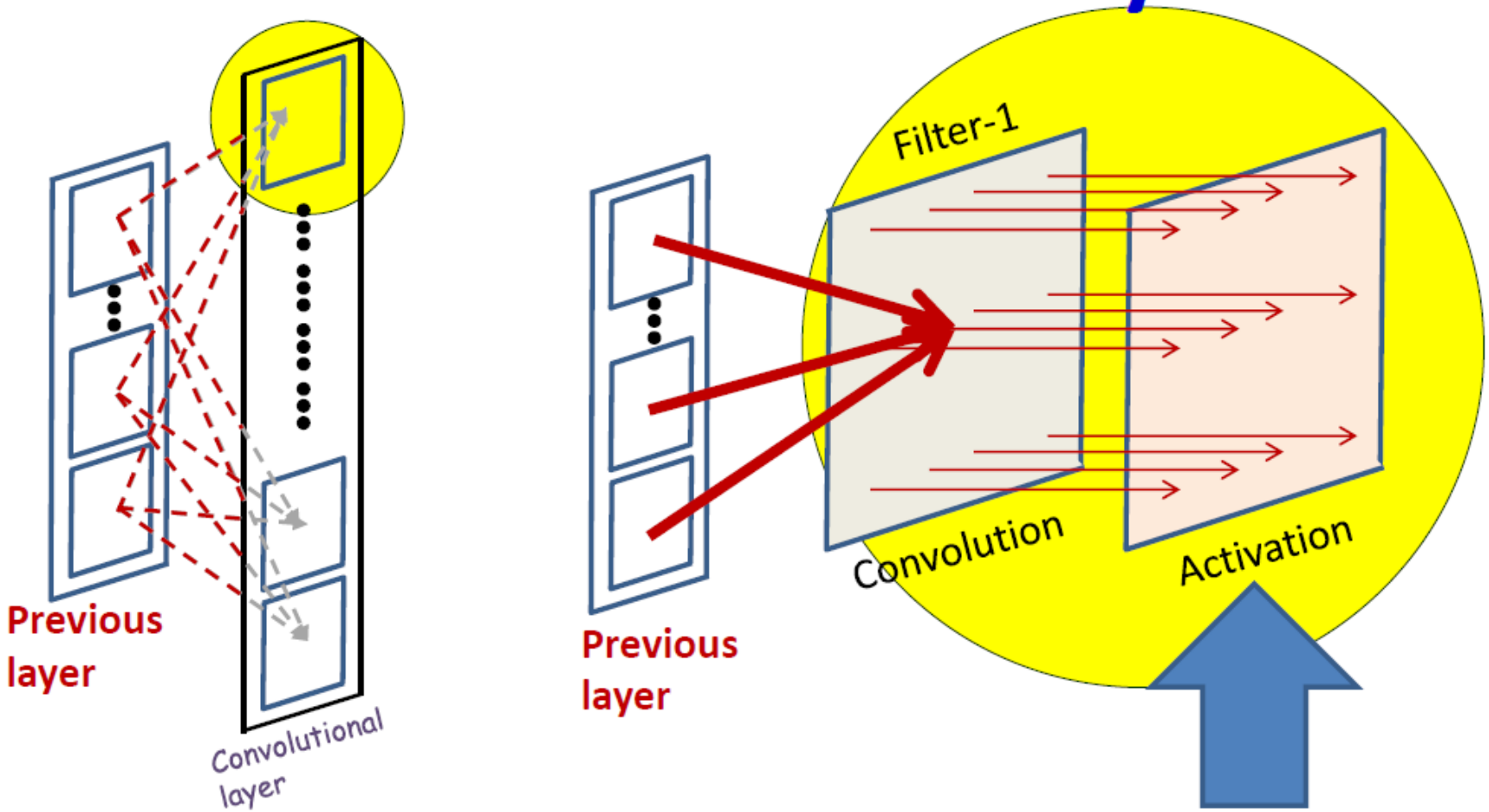
$$\mathrm{Y} = IDFT2\big(DFT2(X) \circ conj(DFT2(W))\big)$$

- Cost of doing this using the Fast Fourier Transform to compute the DFTs: $\mathrm{O}(M^2 log N)$
  - Significant saving for large filters
  - Or if there are many filters

Returning to our problem

- … From the world of size engineering …

A convolutional layer



- The convolution operation results in a convolution map
- An *Activation* is finally applied to every entry in the map

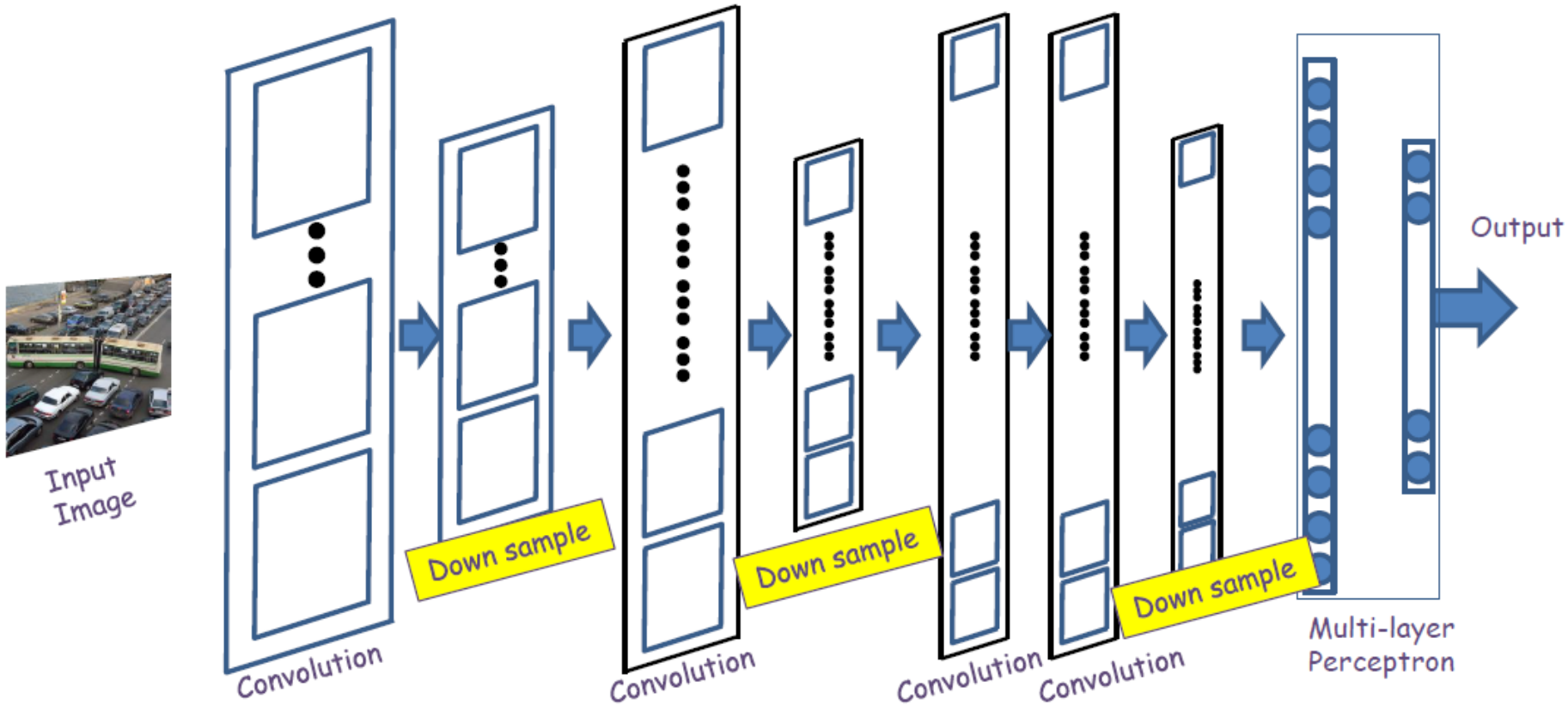
Convolutional neural net:

The weight W(l,j) is now a 3D $D_{l-1}xK_lxK_l$ tensor (assuming square receptive fields)

The product in blue is a tensor inner product with a scalar output

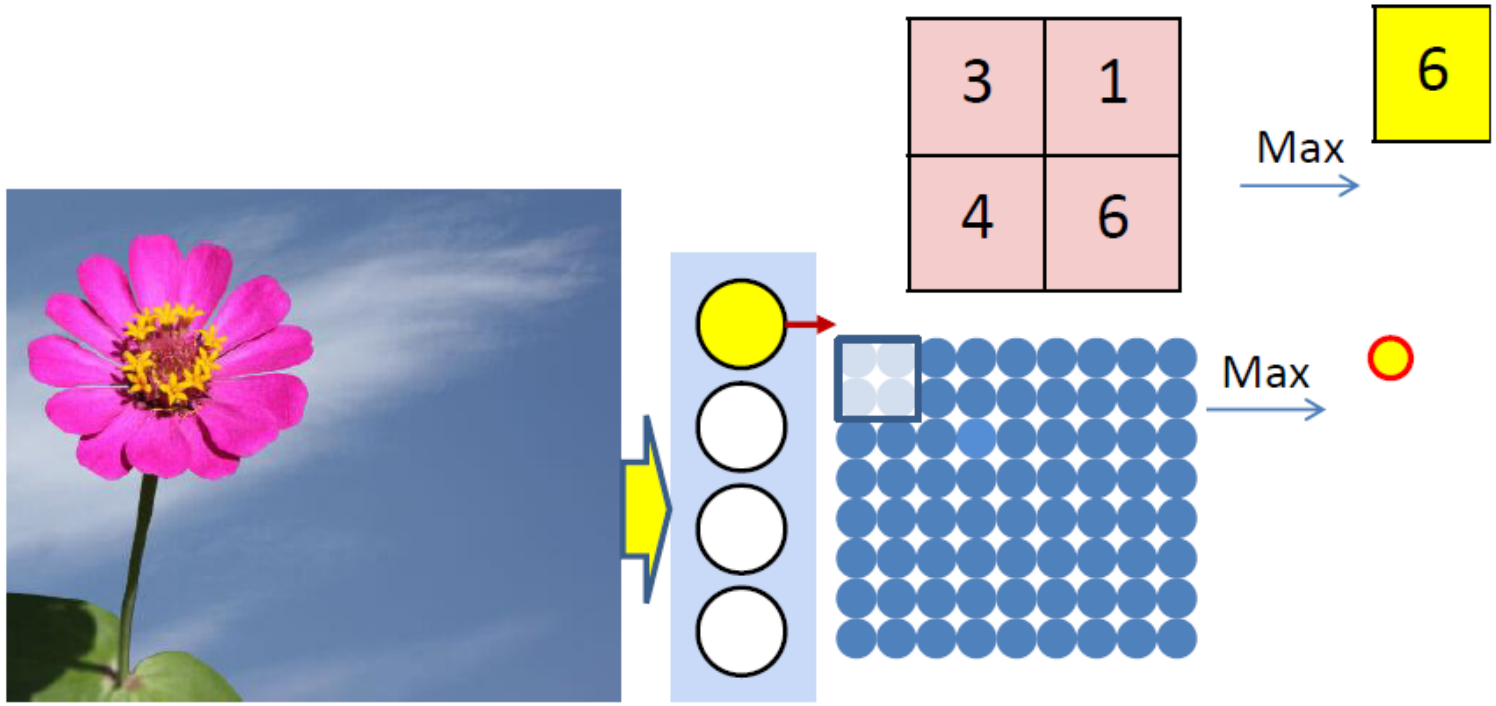
```
Y(0) = Image
for l = 1:L  # layers operate on vector at (x,y)
  for j = 1:D_l
     for x = 1:W_{l-1}-K_l+1
        for y = 1:H_{l-1}-K_l+1
          segment = Y(l-1,:,x:x+K_l-1,y:y+K_l-1) #3D tensor
          z(l,j,x,y) = W(l,j).segment #tensor inner prod.
          Y(l,j,x,y) = activation(z(l,j,x,y))

Y = softmax( {Y(L,:,:,:)} )
```

The other component
Downsampling/Pooling



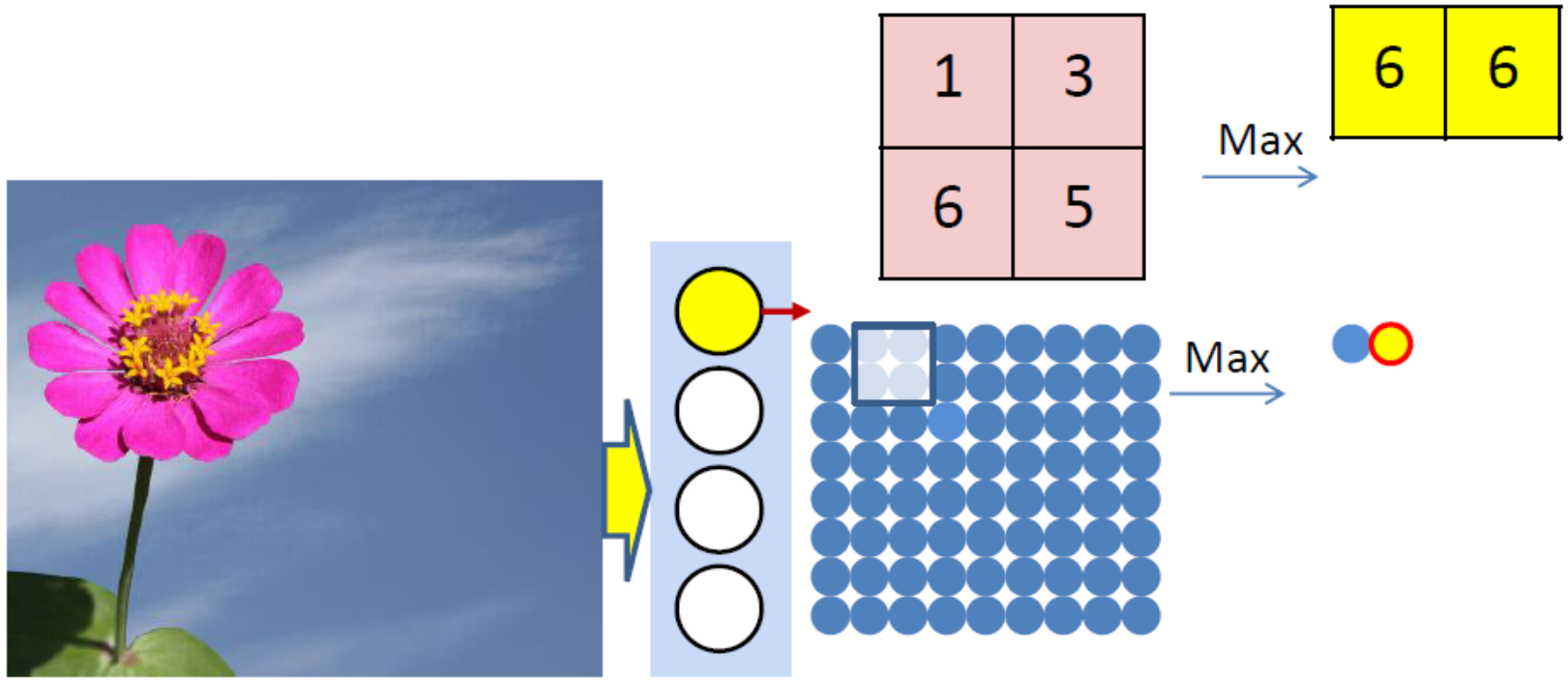
- Convolution (and activation) layers are followed intermittently by "downsampling" (or "pooling") layers
  - Often, they alternate with convolution, though this is not necessary

Recall: Max pooling



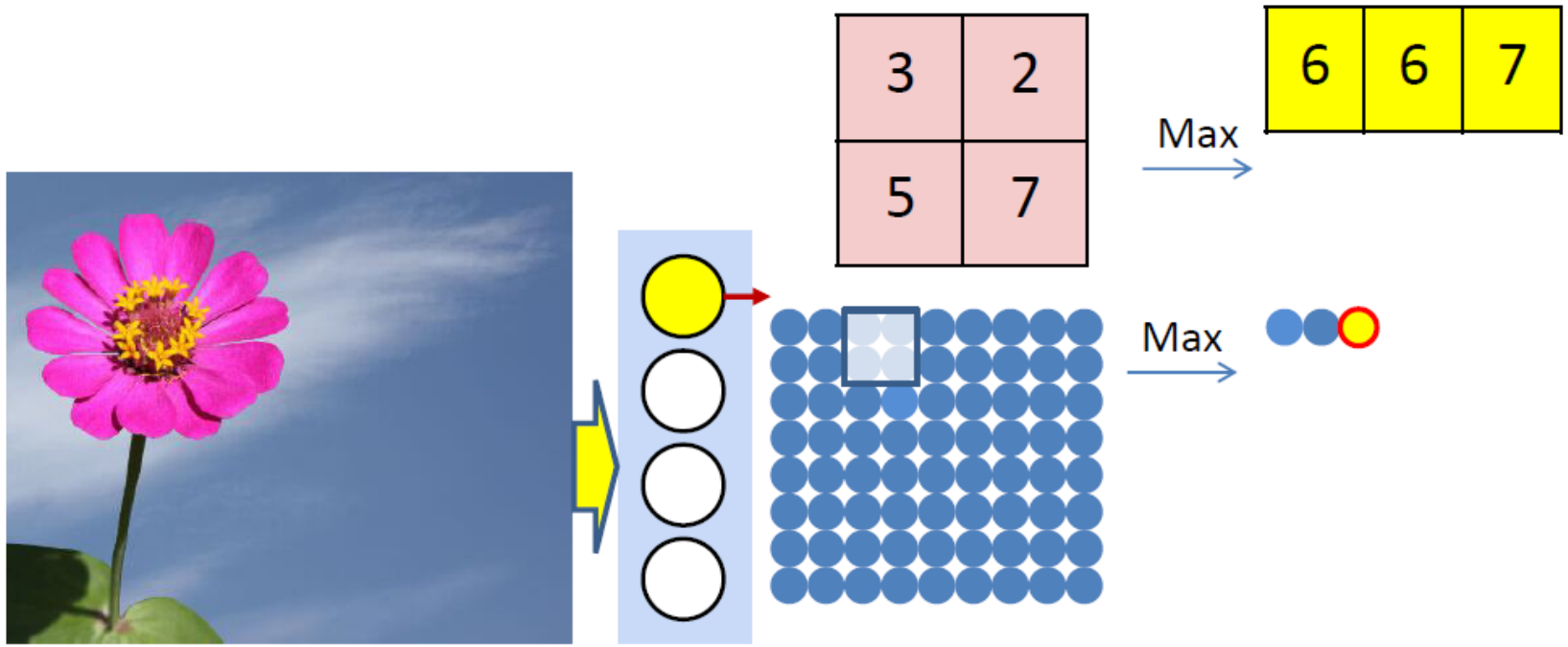
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Recall: Max pooling



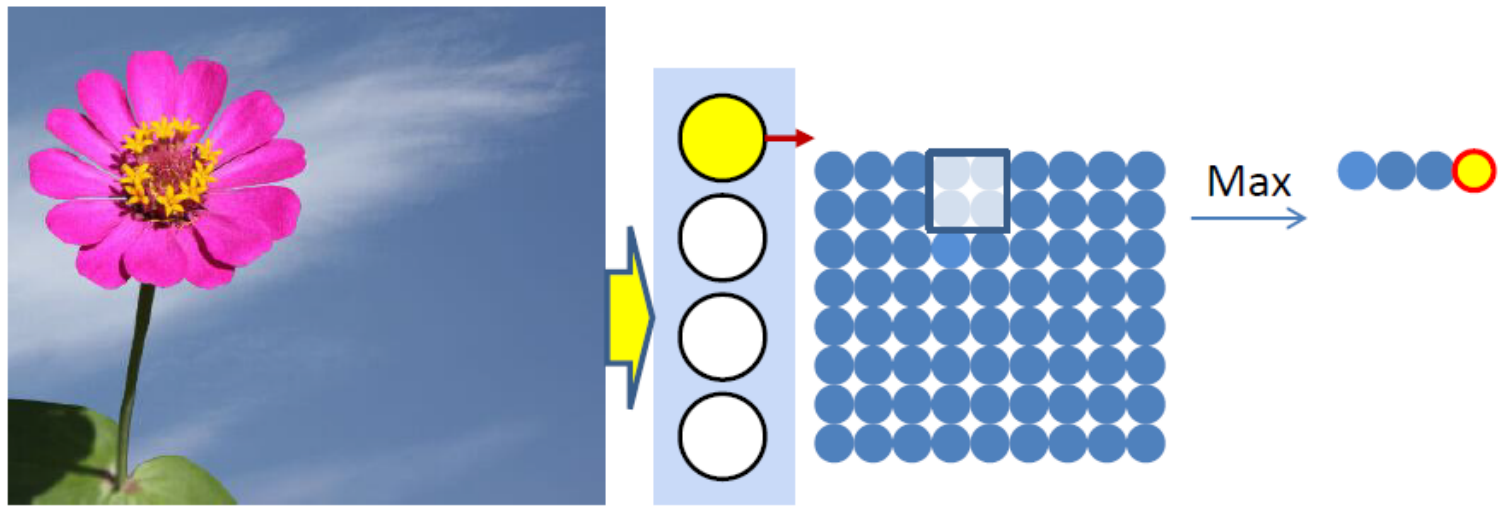
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Recall: Max pooling



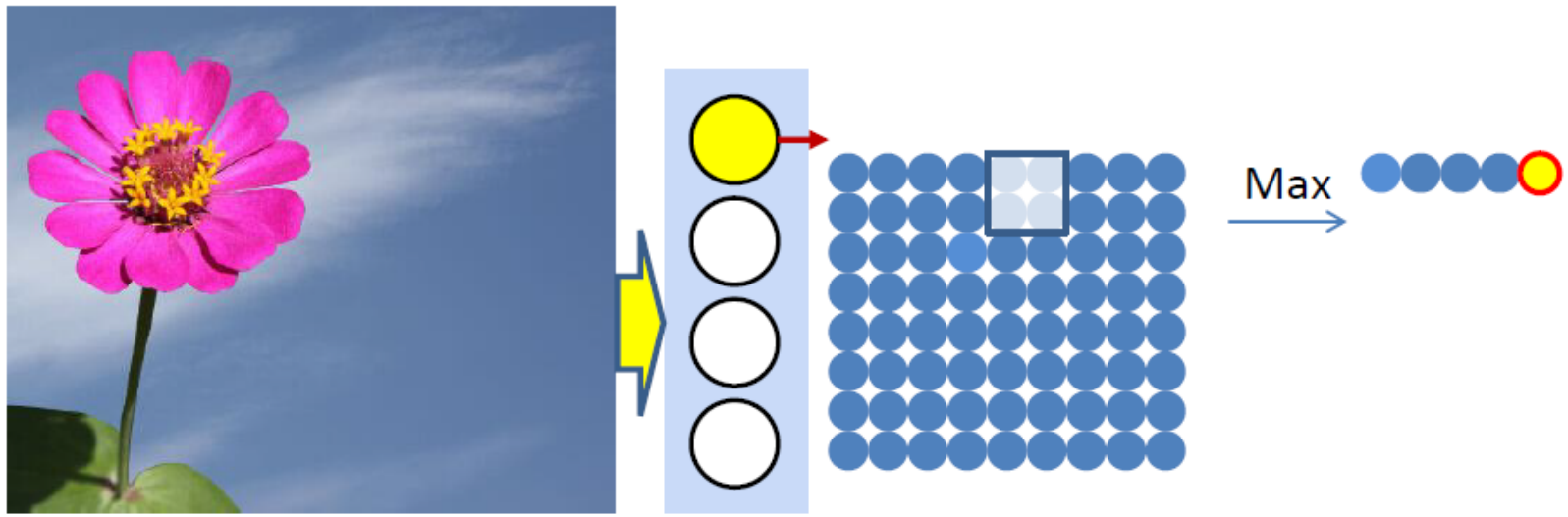
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Recall: Max pooling



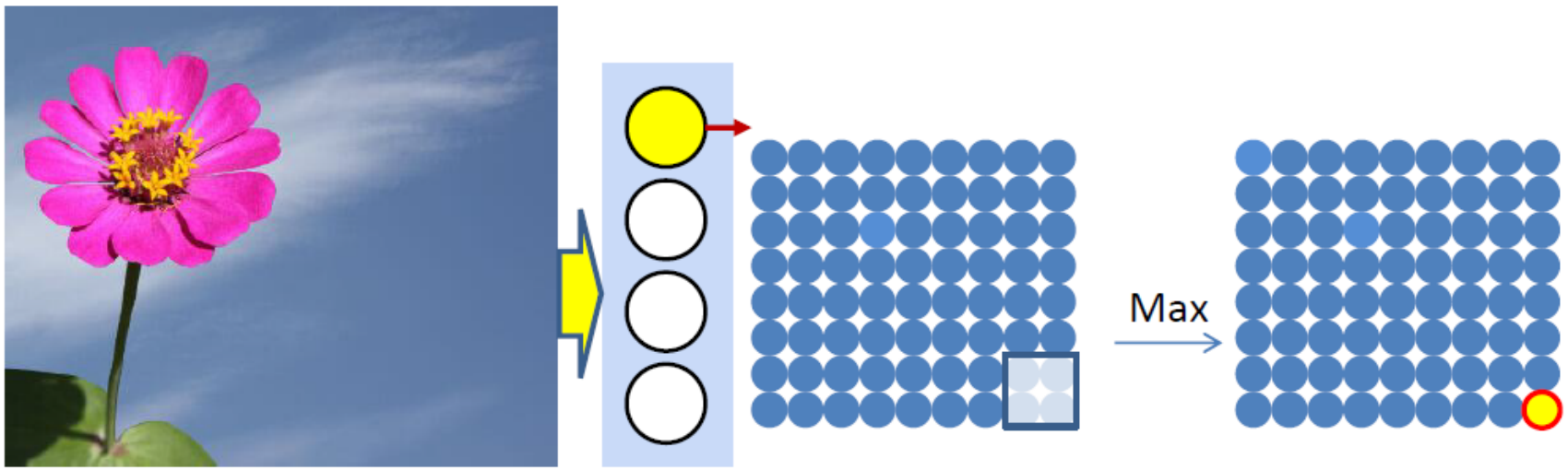
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Recall: Max pooling



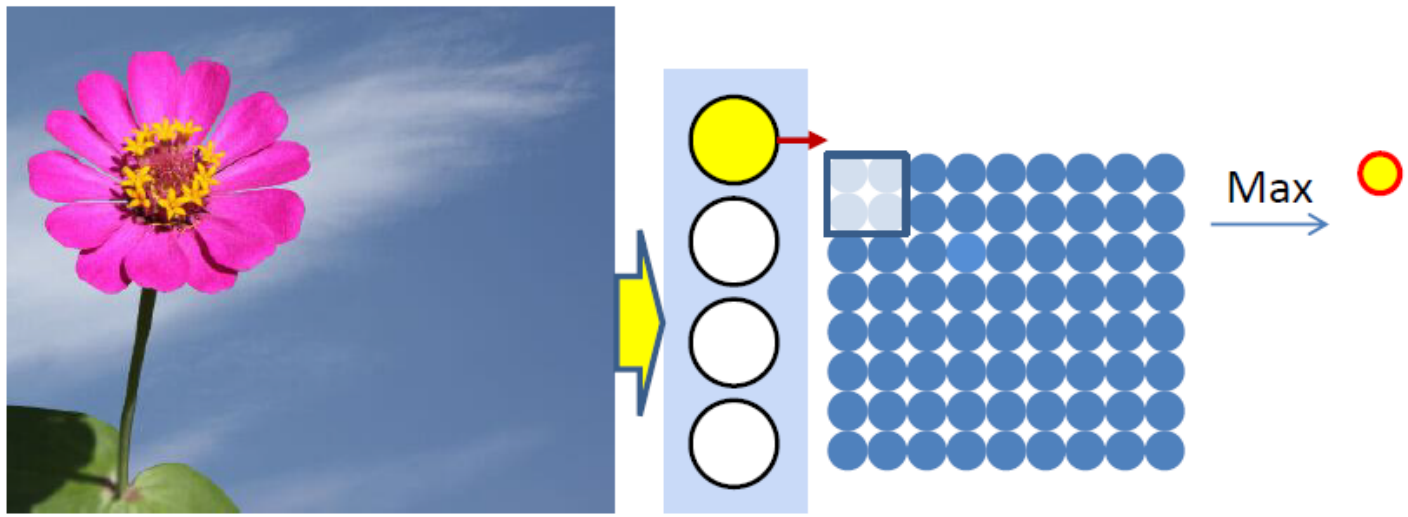
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Recall: Max pooling



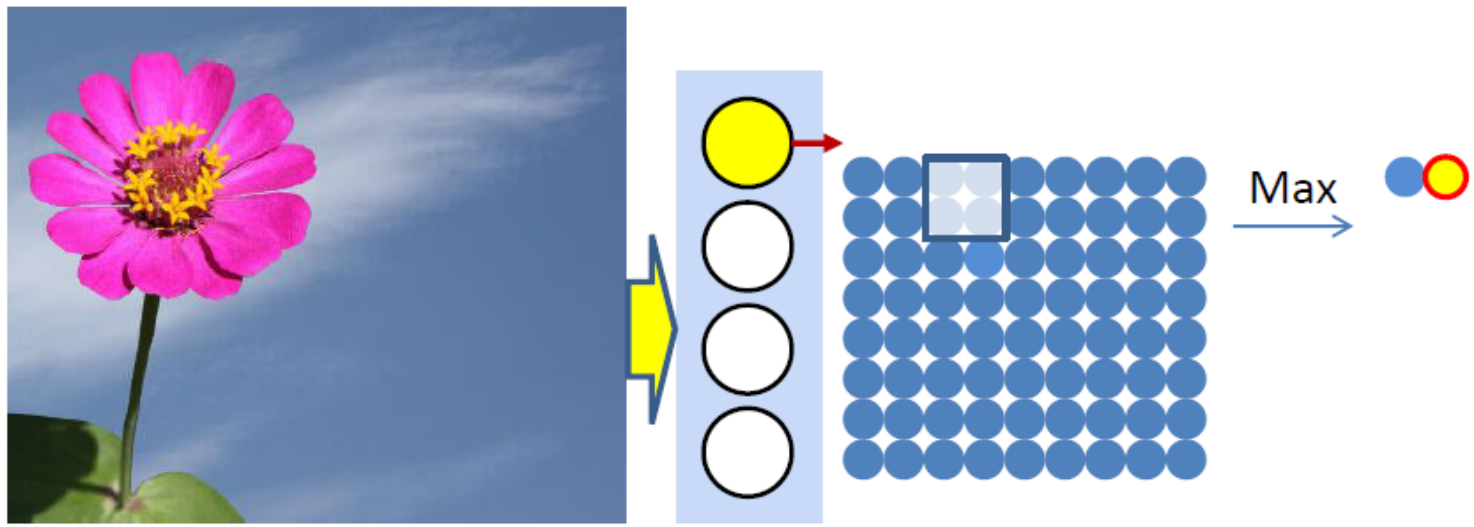
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

“Strides”



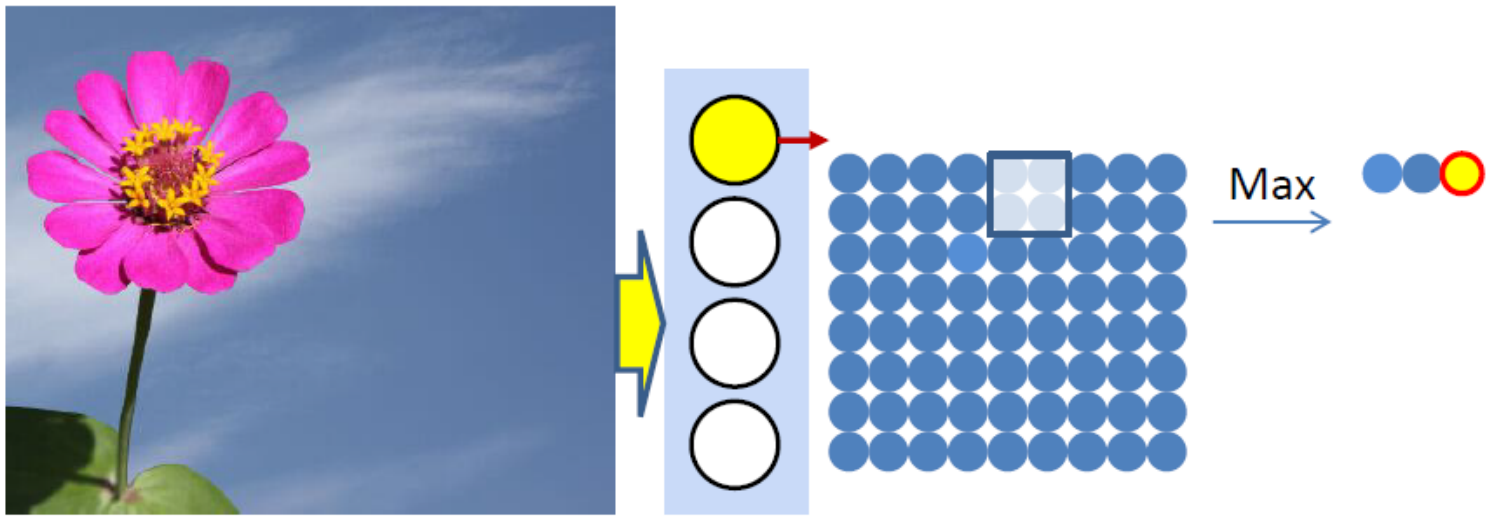
- The “max” operations may “stride” by more than one pixel

“Strides”



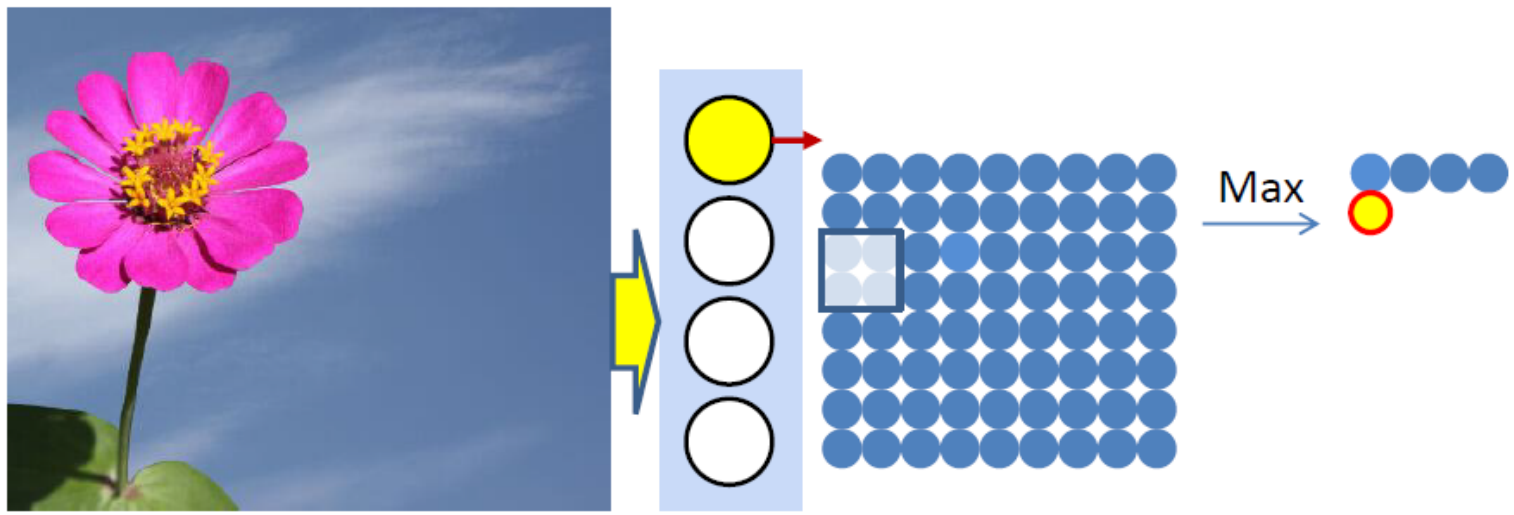
- The “max” operations may “stride” by more than one pixel

“Strides”



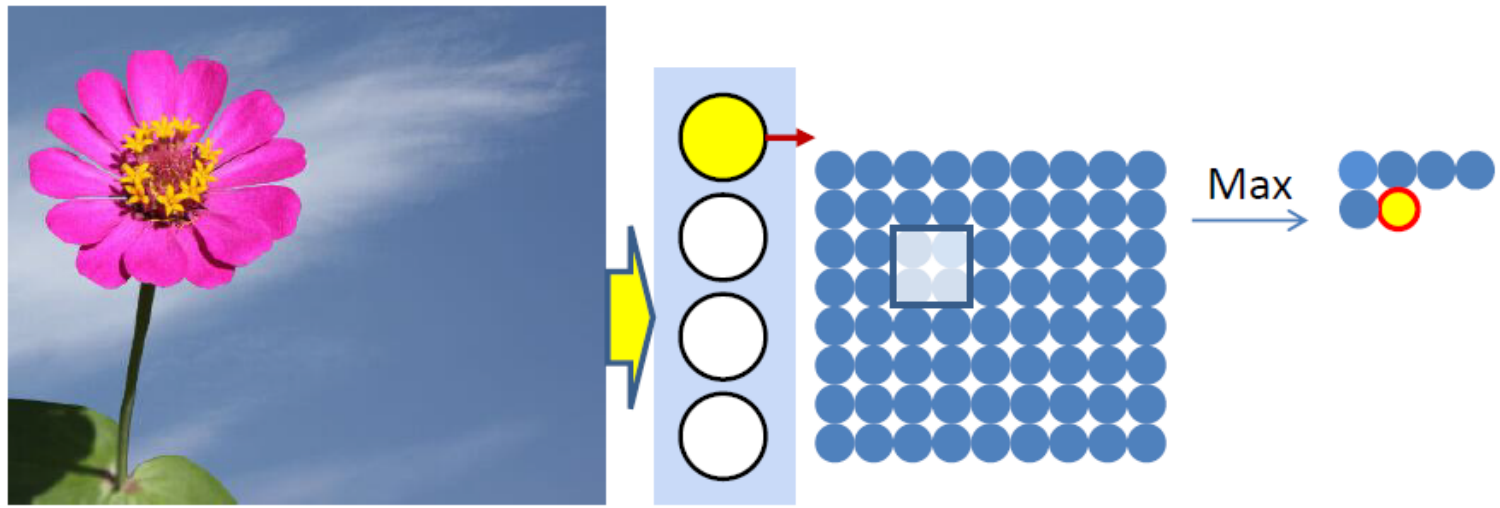
- The “max” operations may “stride” by more than one pixel

“Strides”



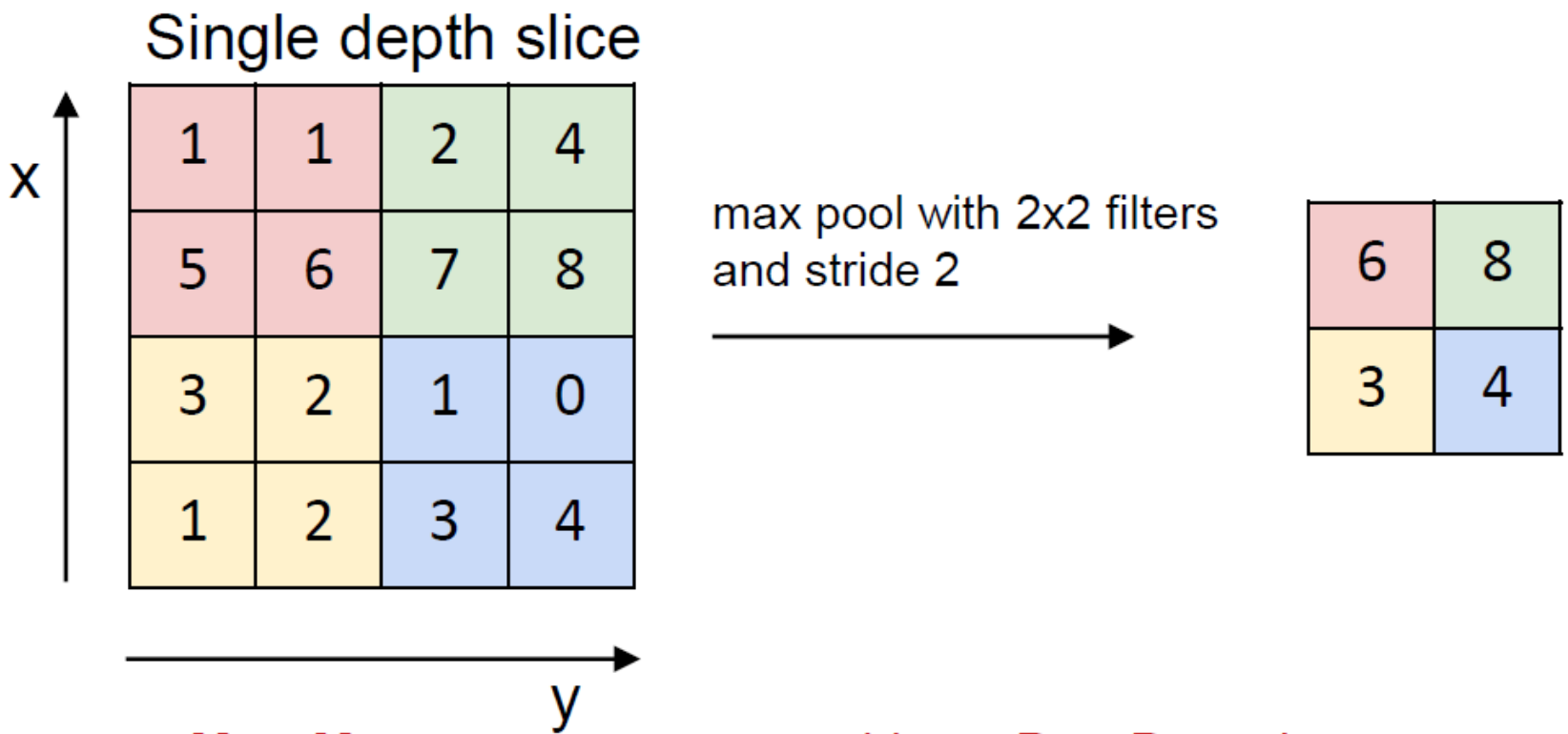
- The “max” operations may “stride” by more than one pixel

“Strides”



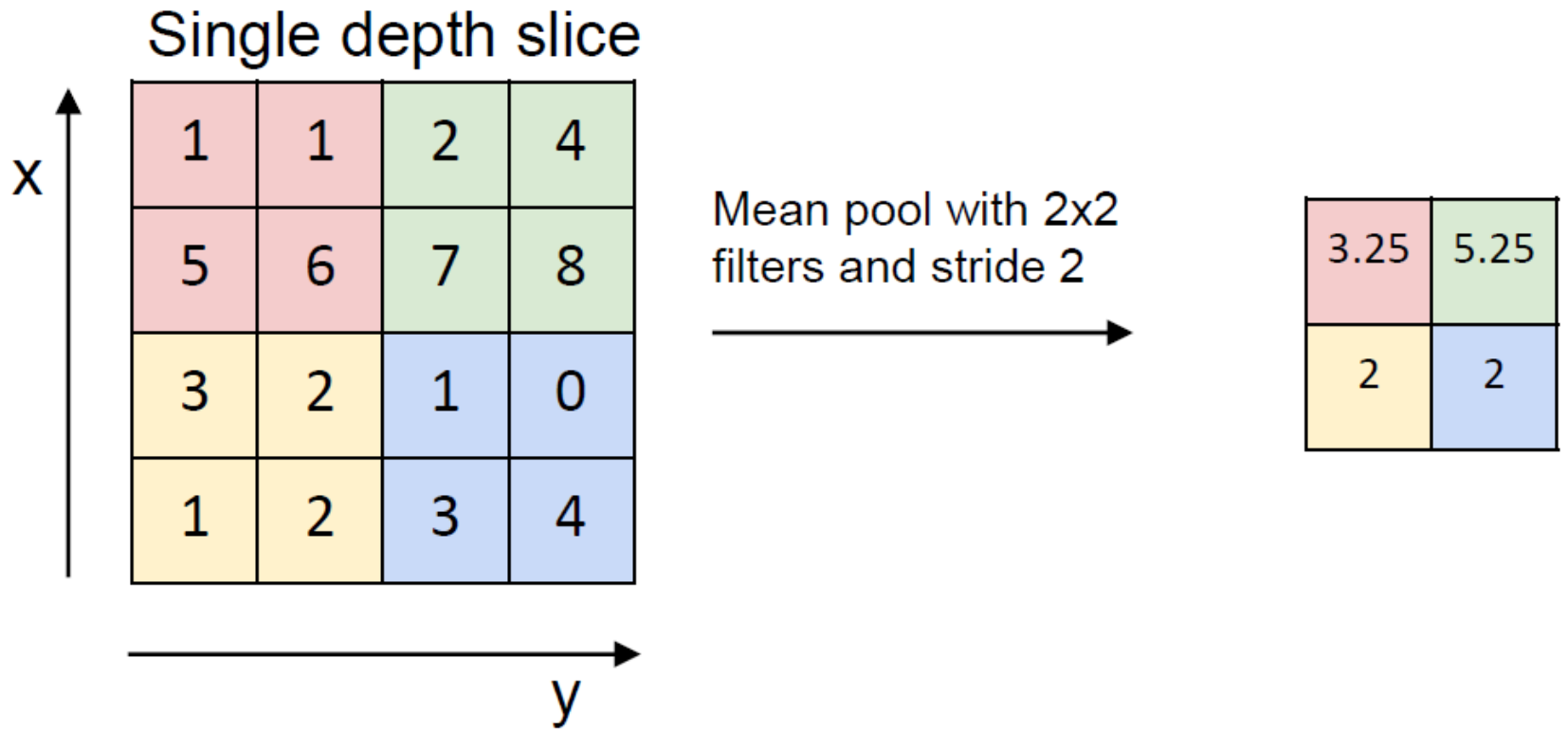
- The “max” operations may “stride” by more than one pixel

Pooling: Size of output



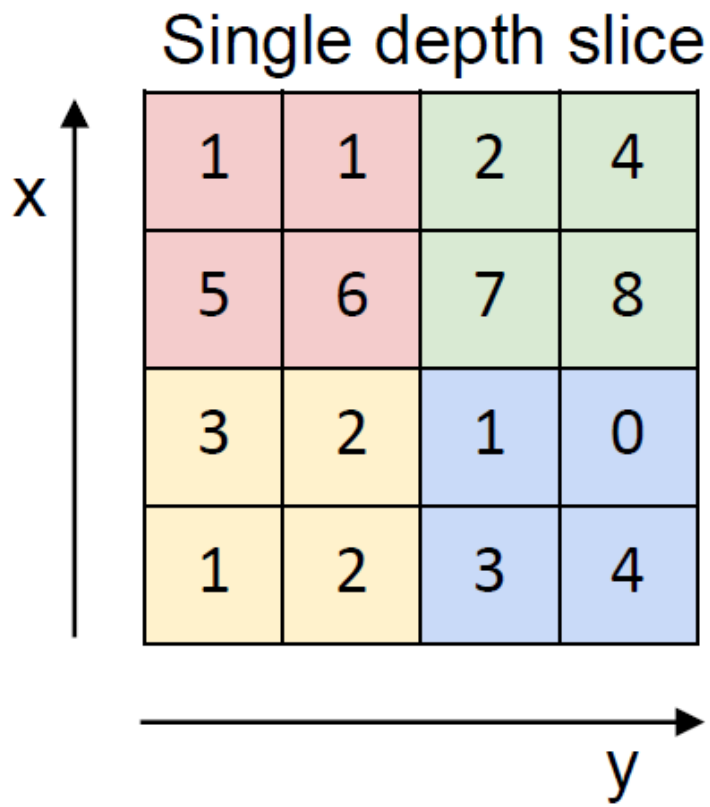
- An $N \times N$ picture compressed by a $P \times P$ pooling filter with stride D results in an output map of side $\lceil (N -$

Alternative to Max pooling: Mean Pooling



- Compute the mean of the pool, instead of the max

Alternative to Max pooling: P-norm



P-norm with 2x2 filters and stride 2, p = 5

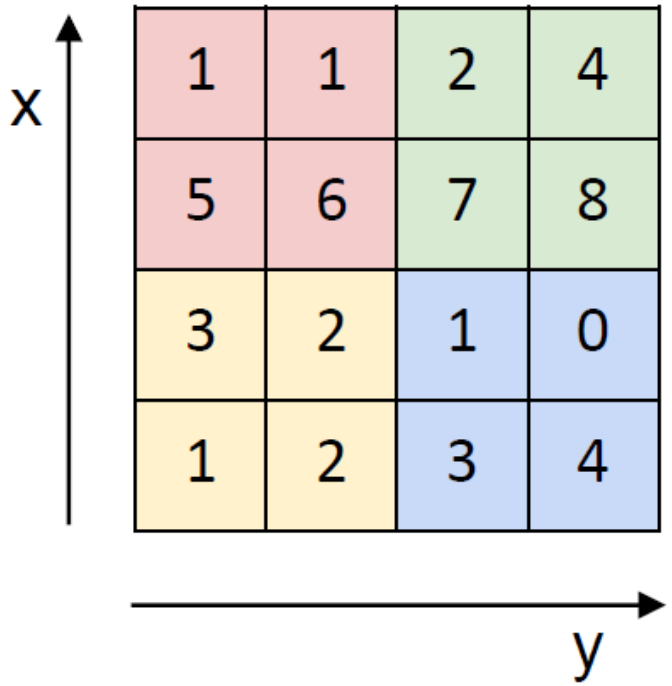
$$y = \sqrt[p]{\frac{1}{P^2}\sum_{i,j} x_{ij}^p}$$

4.86	8
2.38	3.16

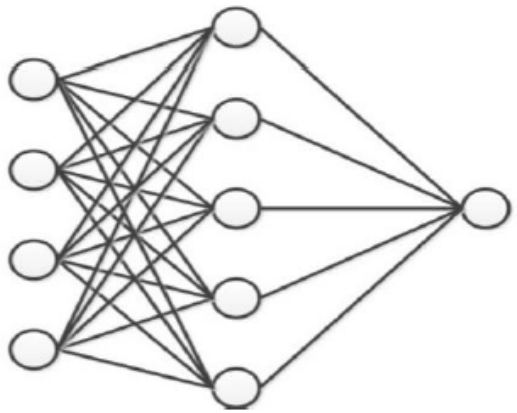
- Compute a p-norm of the pool

Other options

Single depth slice



Network applies to each 2x2 block and strides by 2 in this example

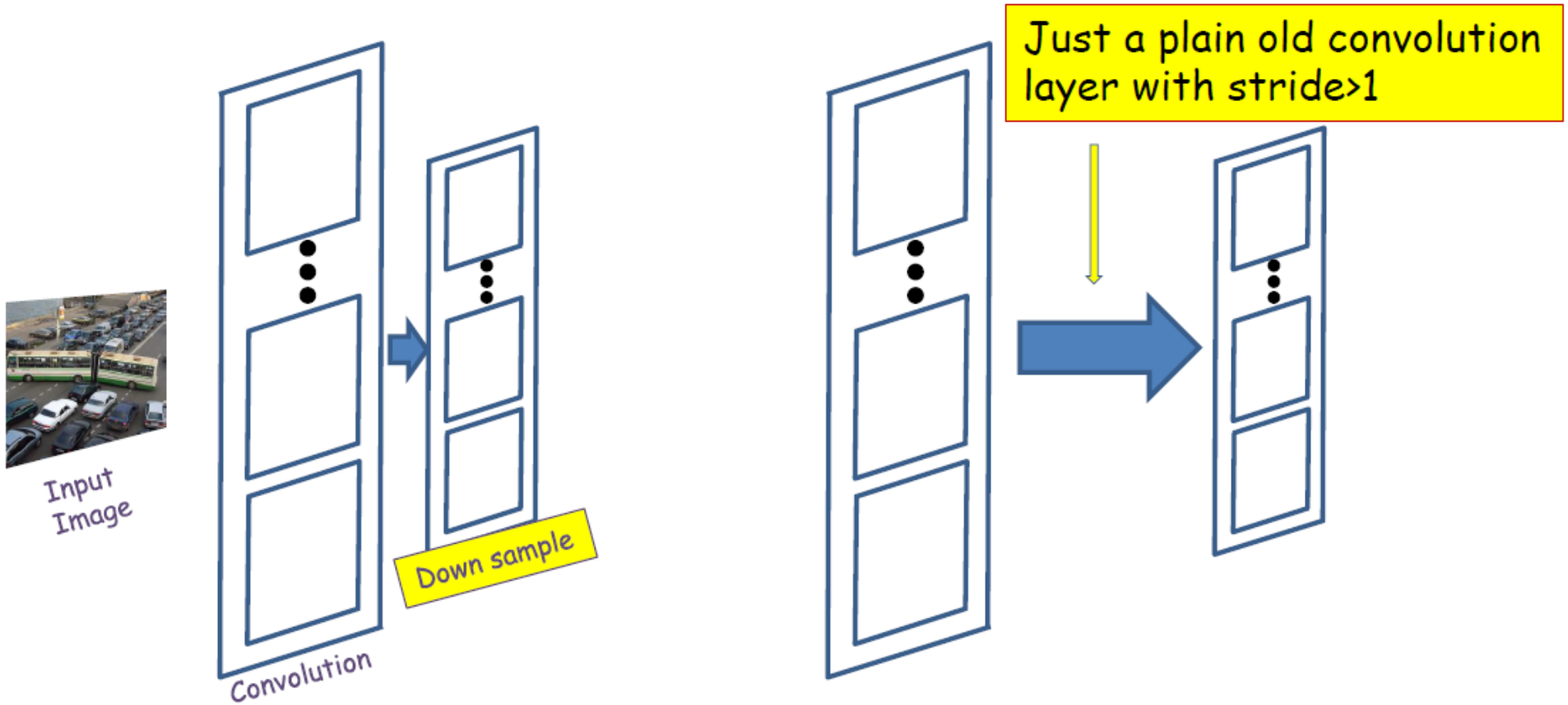


6	8
3	4

Network in network

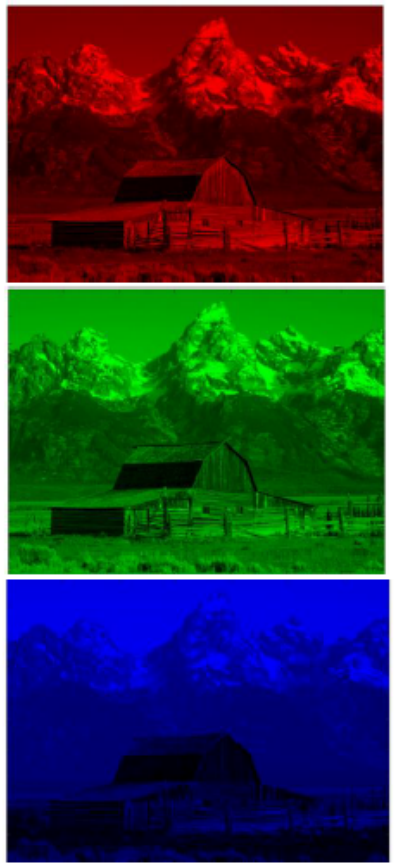
- The pooling may even be a *learned* filter
  - The *same* network is applied on each block
    - (Again, a shared parameter network)

Or even an "all convolutional" net



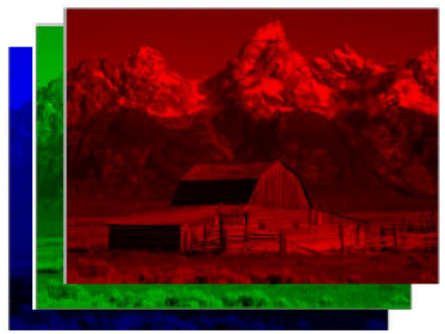
- Downsampling may even be done by a simple convolution layer with stride larger than 1
  - Replacing the maxpooling layer with a conv layer

Convolutional Neural Networks



- Input: 3 pictures

Convolutional Neural Networks



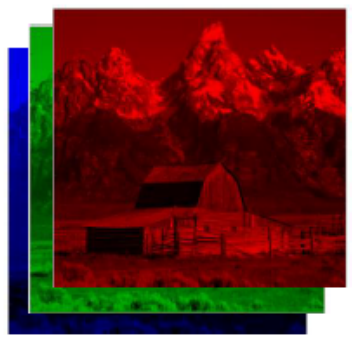
- Input: 3 pictures

Preprocessing

- Typically works with *square* images
  - Filters are also typically square

- Large networks are a problem
  - Too much detail
  - Will need big networks
- Typically scaled to small sizes, e.g. 32x32 or 128x128
  - Based on how much will fit on your GPU

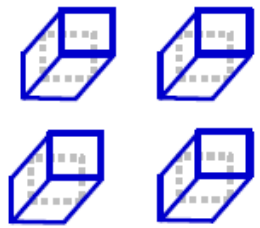
Convolutional Neural Networks



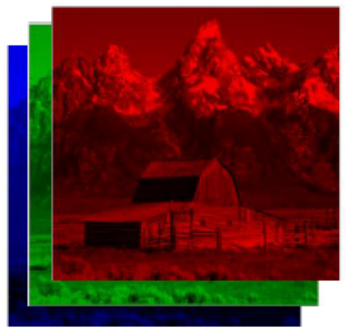
$I \times I$ *image*

- Input: 3 pictures

Convolutional Neural Networks



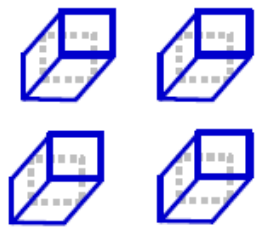
K_1 total filters
Filter size: $L \times L \times 3$



$I \times I$ *image*

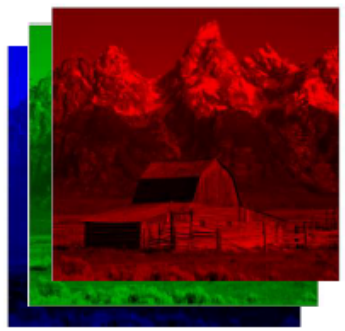
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

Convolutional Neural Networks



K_1 total filters
Filter size: $L \times L \times 3$

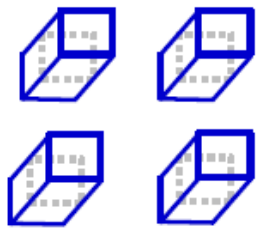
Small enough to capture fine features
(particularly important for scaled-down images)



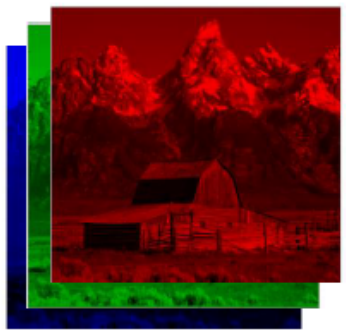
$I \times I$ *image*

- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

Convolutional Neural Networks



K_1 total filters
Filter size: $L \times L \times 3$



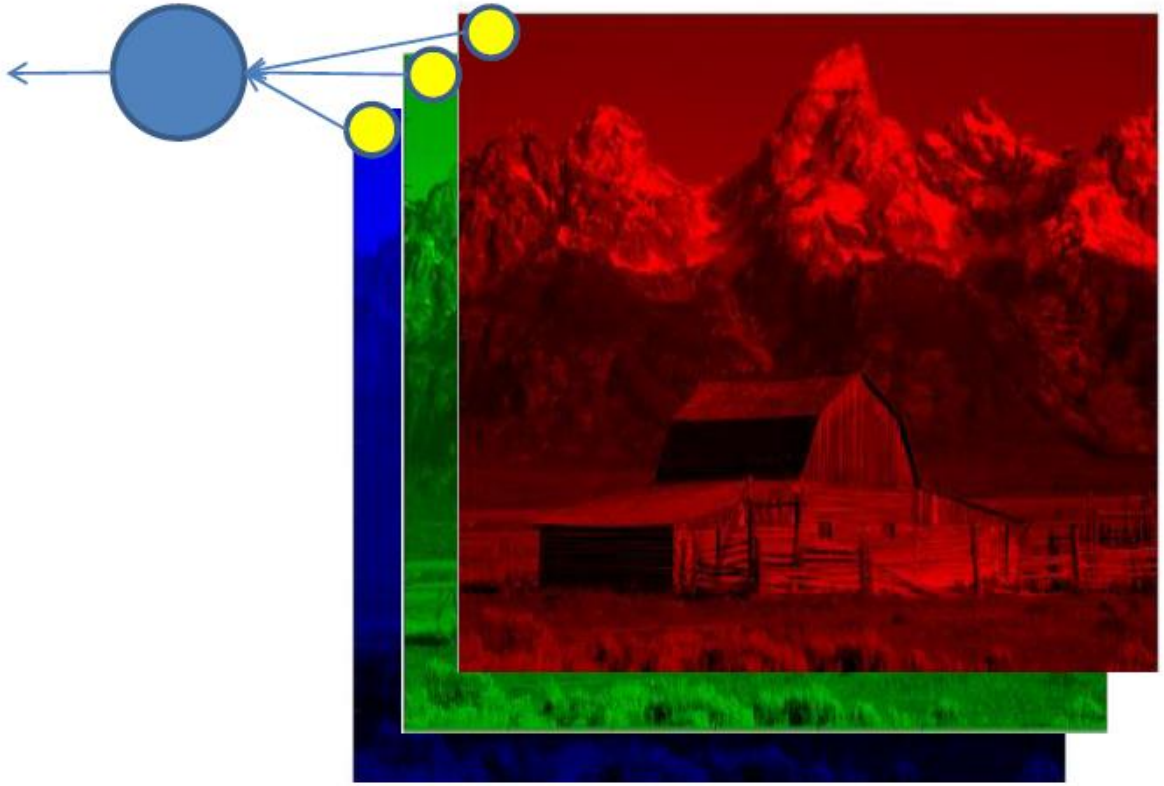
$I \times I$ *image*

Small enough to capture fine features
(particularly important for scaled-down images)

What on earth is this?

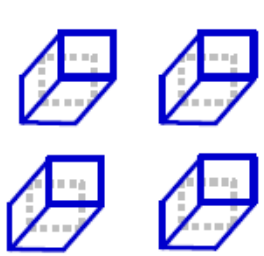
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

The 1x1 filter

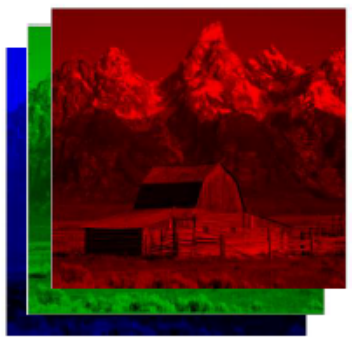


- A 1x1 filter is simply a perceptron that operates over the *depth* of the map, but has no spatial extent
  - Takes one pixel from each of the maps (at a given location) as input

Convolutional Neural Networks



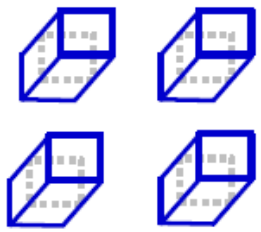
K_1 total filters
Filter size: $L \times L \times 3$



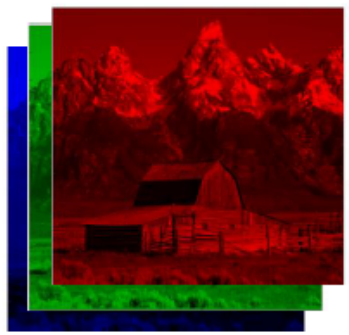
$I \times I$ *image*

- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - **Better notation:** Filters are typically 5x5(x3), 3x3(x3), or even 1x1(x3)

Convolutional Neural Networks



K_1 total filters
Filter size: $L \times L \times 3$



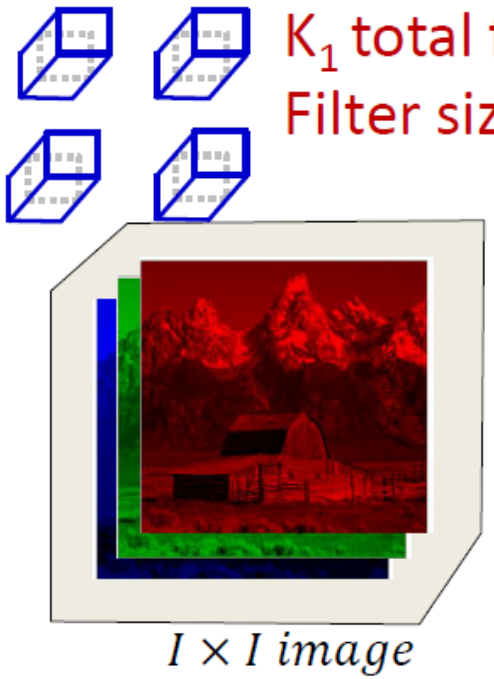
$I \times I$ *image*

Parameters to choose: K_1, L and S
1. Number of filters K_1
2. Size of filters $L \times L \times 3 + bias$
3. *Stride* of convolution S

Total number of parameters: $K_1(3L^2 + 1)$

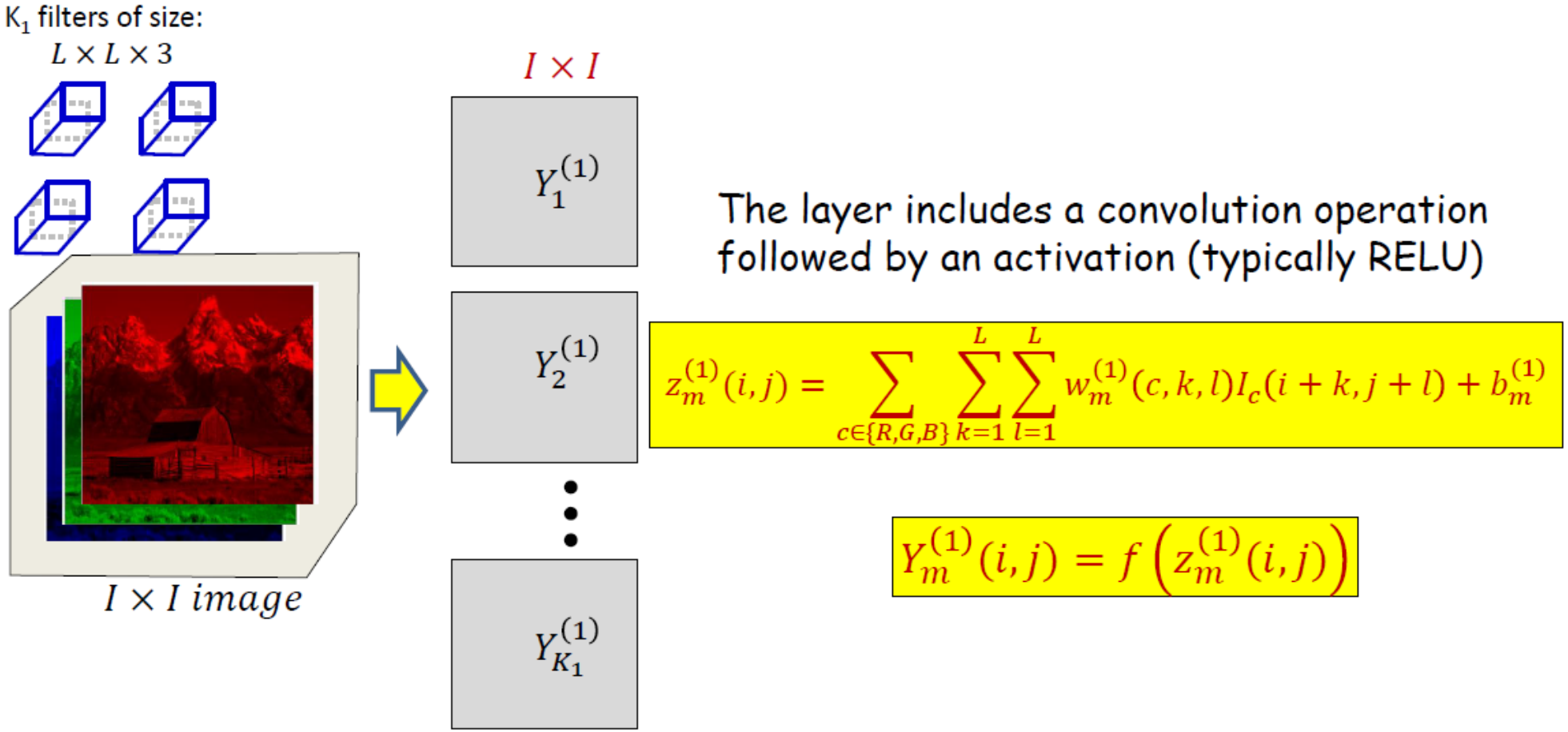
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - **Better notation:** Filters are typically 5x5(x3), 3x3(x3), or even 1x1(x3)
  - ***Typical stride:*** 1 or 2

Convolutional Neural Networks



- The input may be zero-padded according to the size of the chosen filters

Convolutional Neural Networks



The layer includes a convolution operation followed by an activation (typically RELU)

$$z_m^{(1)}(i,j) = \sum_{c\in\{R,G,B\}} \sum_{k=1}^{L} \sum_{l=1}^{L} w_m^{(1)}(c,k,l) I_c(i+k, j+l) + b_m^{(1)}$$

$$Y_m^{(1)}(i,j) = f\left(z_m^{(1)}(i,j)\right)$$

- **First convolutional layer:** Several convolutional filters
  - Filters are "3-D" (third dimension is color)
  - Convolution followed typically by a RELU activation
- Each filter creates a single 2-D output map

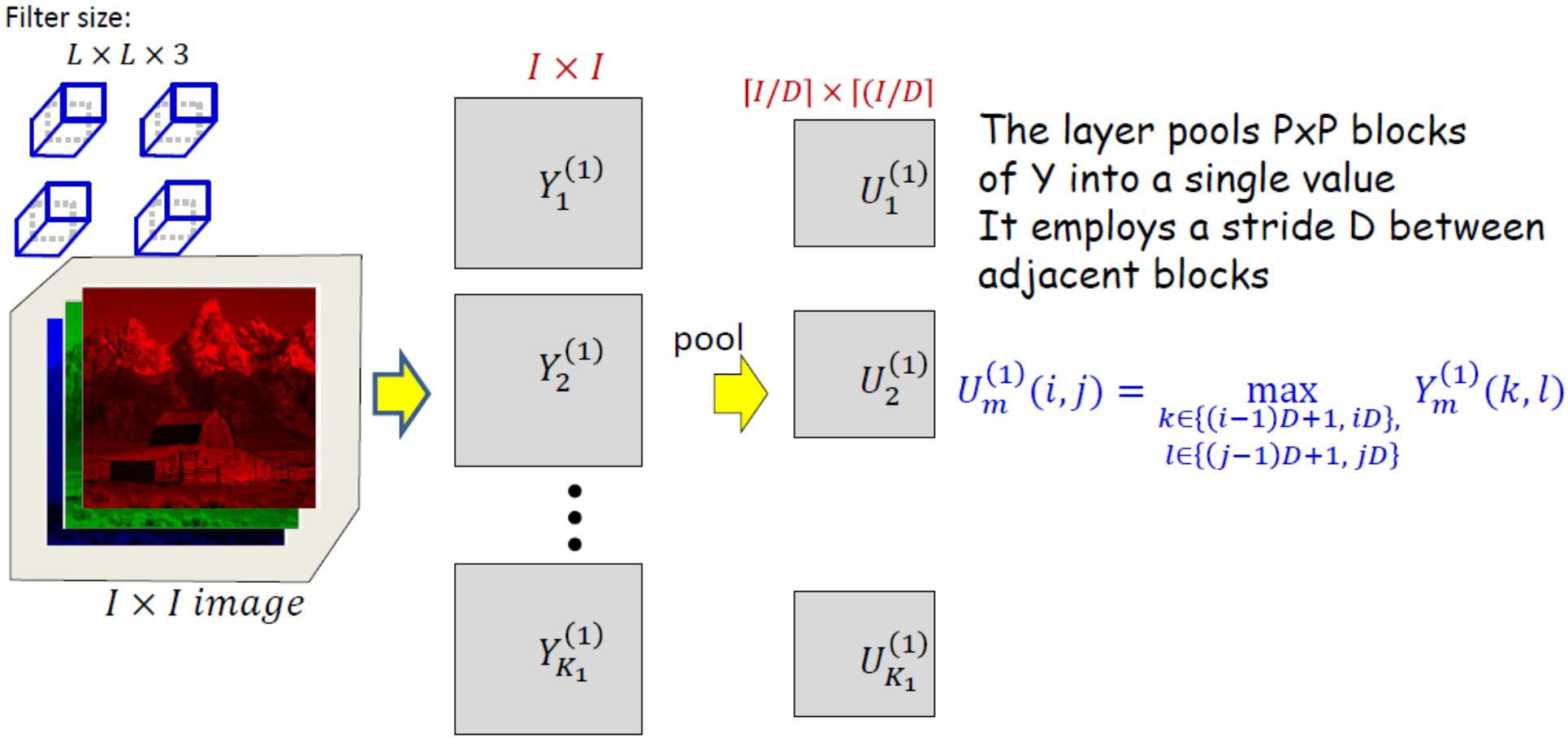
Learnable parameters in the first convolutional layer

- The first convolutional layer comprises K_1 filters, each of size $L \times L \times 3$
  - Spatial span: $L \times L$
  - Depth : 3 (3 colors)

- This represents a total of $K_1(3L^2 + 1)$ parameters
  - "+ 1" because each filter also has a bias

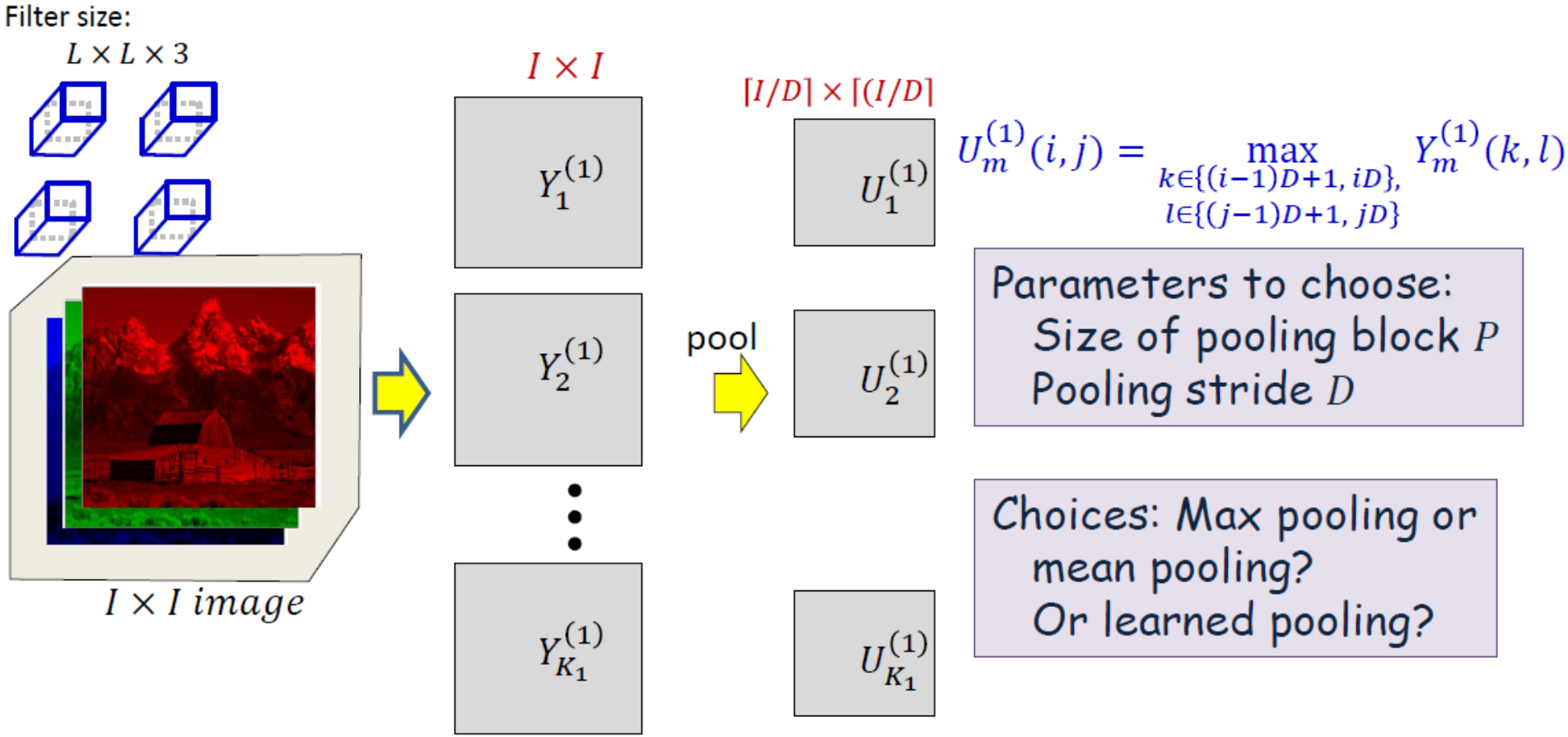
- All of these parameters must be learned

Convolutional Neural Networks



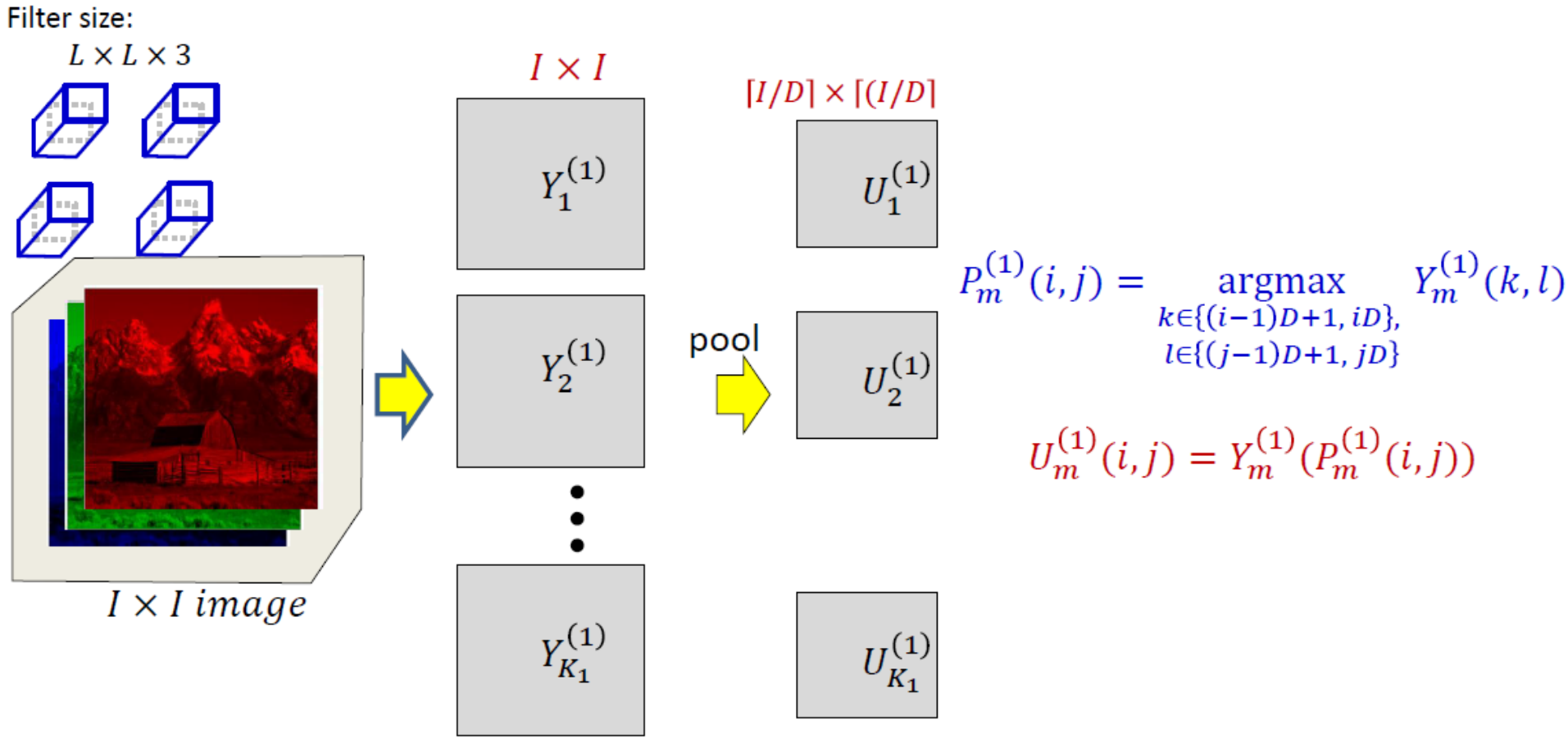
- **First downsampling layer:** From each $P \times P$ block of each map, *pool* down to a single value
  - For max pooling, during training keep track of which position had the highest value

Convolutional Neural Networks



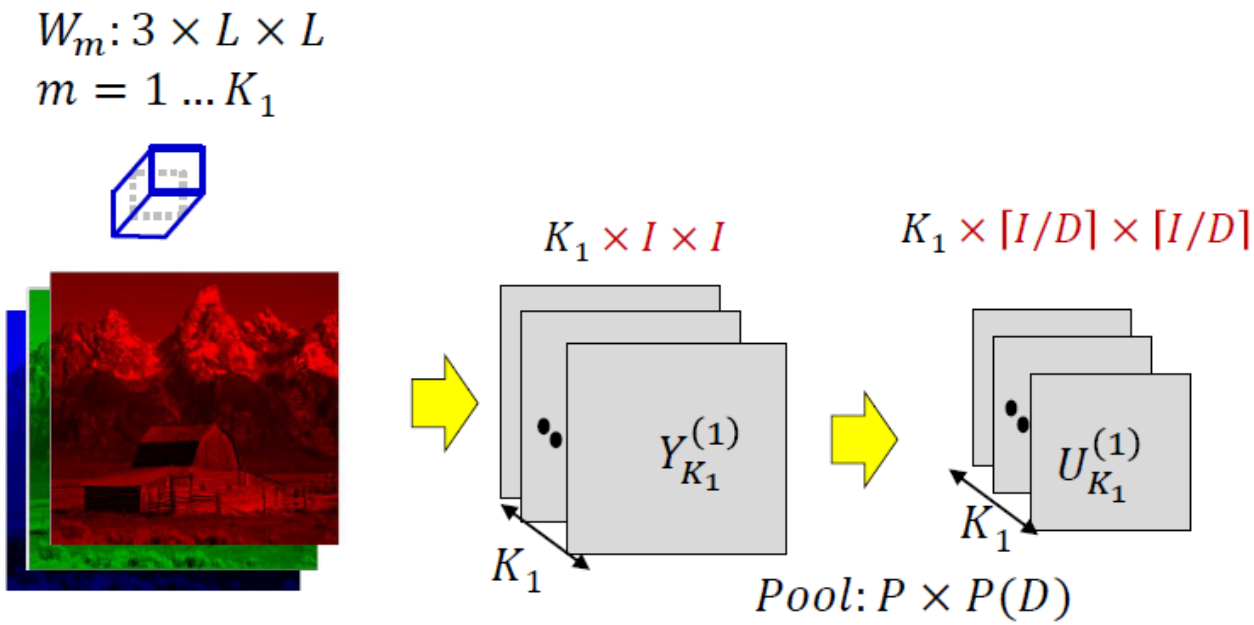
- **First downsampling layer:** From each $P \times P$ block of each map, *pool* down to a single value
  - For max pooling, during training keep track of which position had the highest value

Convolutional Neural Networks



- **First downsampling layer:** From each $P \times P$ block of each map, *pool* down to a single value
  - **For max pooling, during training keep track of which position had the highest value**

Convolutional Neural Networks

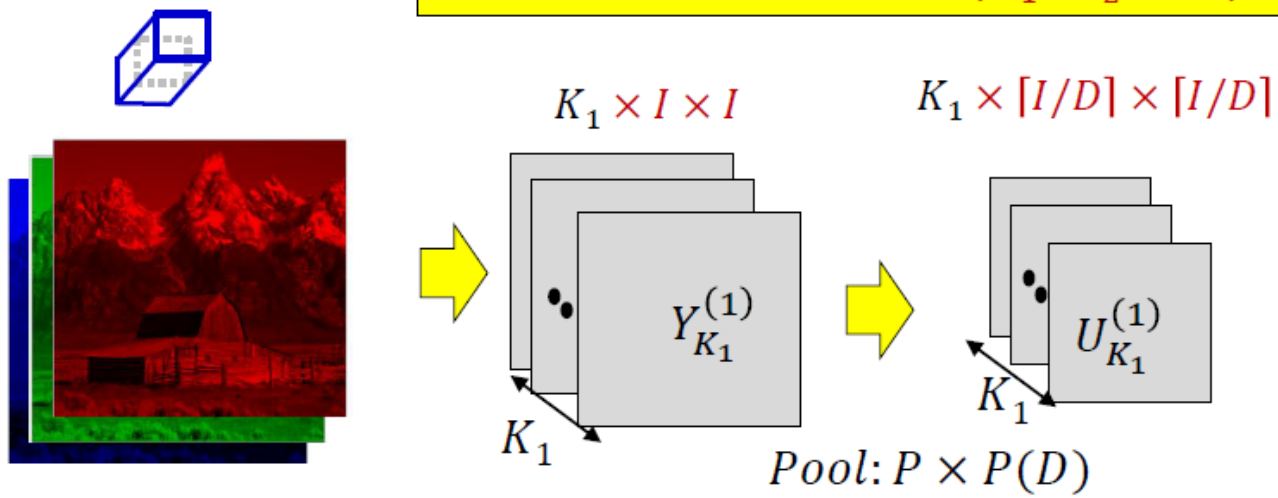


- **First pooling layer:** Drawing it differently for convenience

Convolutional Neural Networks

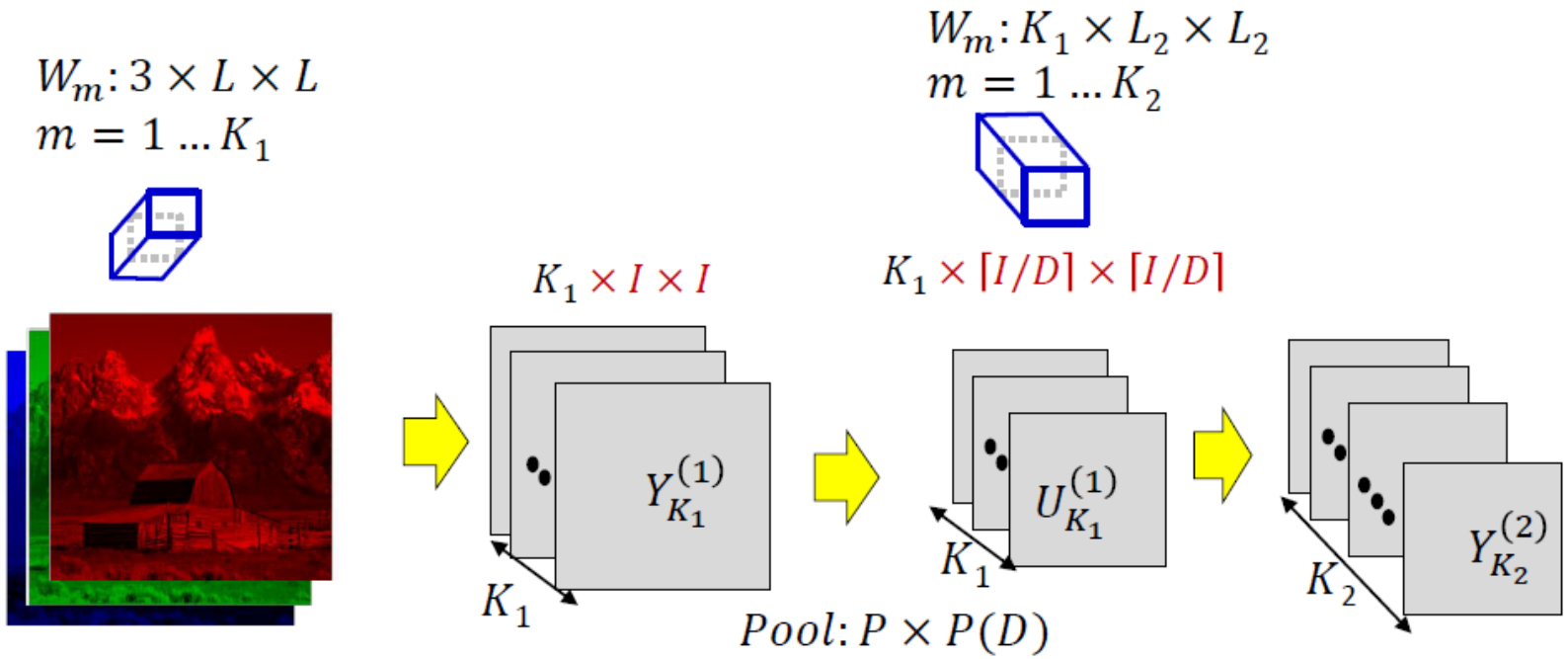
$W_m: 3 \times L \times L$
$m = 1 \ldots K_1$

Jargon: Filters are often called "Kernels"
The outputs of individual filters are called "channels"
The number of filters (K_1, K_2, etc) is the number of channels



- **First pooling layer:** Drawing it differently for convenience

Convolutional Neural Networks

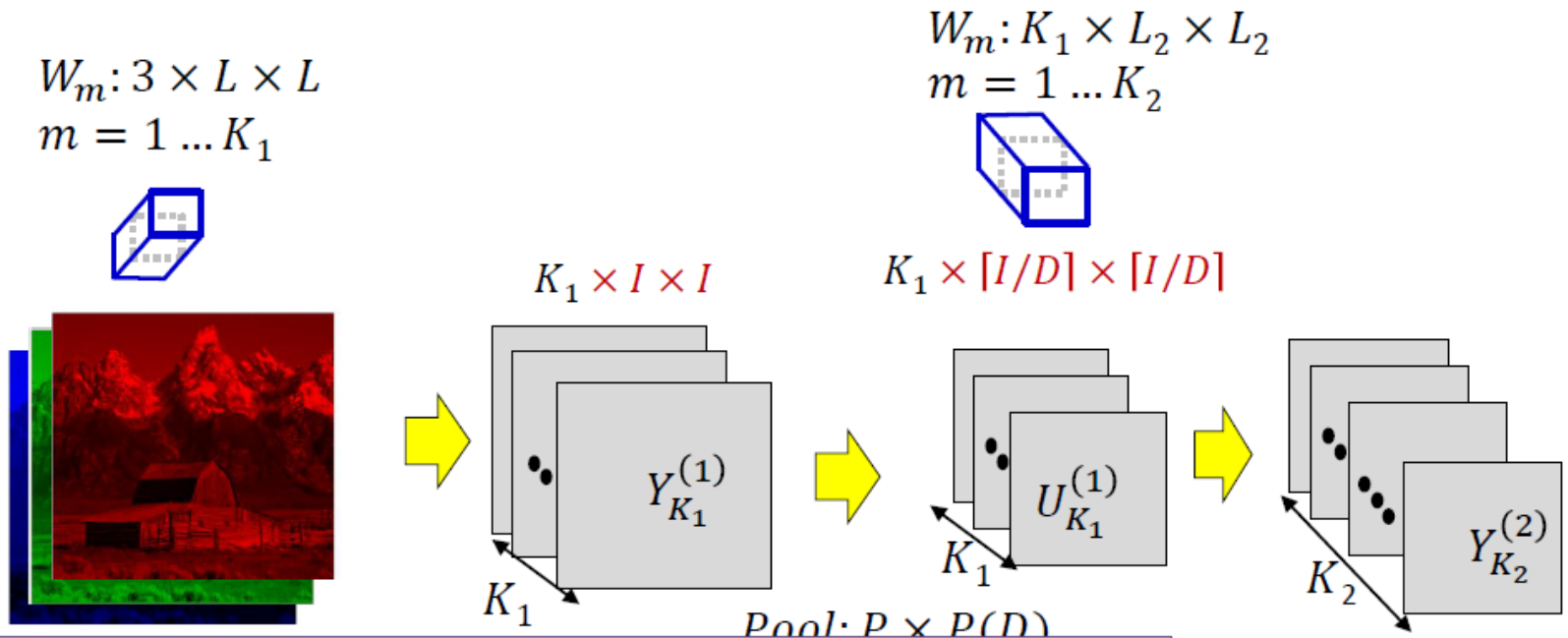


$$z_m^{(n)}(i,j) = \sum_{r=1}^{K_{n-1}} \sum_{k=1}^{L_n} \sum_{l=1}^{L_n} w_m^{(n)}(r,k,l) U_r^{(n-1)}(i+k, j+l) + b_m^{(n)}$$

$$Y_m^{(n)}(i,j) = f\left(z_m^{(n)}(i,j)\right)$$

- **Second convolutional layer:** K_2 3-D filters resulting in K_2 2-D maps
  - Alternately, a kernel with K_2 output channels

Convolutional Neural Networks



Parameters to choose: K_2, L_2 and S_2

1. Number of filters K_2
2. Size of filters $L_2 \times L_2 \times K_1 + bias$
3. *Stride* of convolution S_2

Total number of parameters: $K_2(K_1 L_2^2 + 1)$

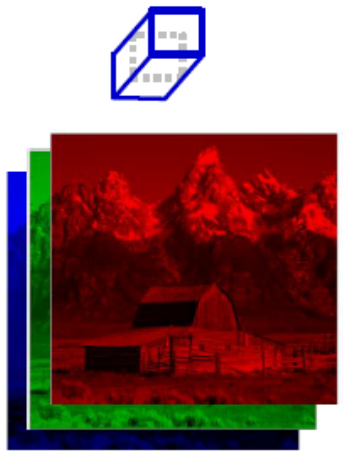
All these parameters must be learned

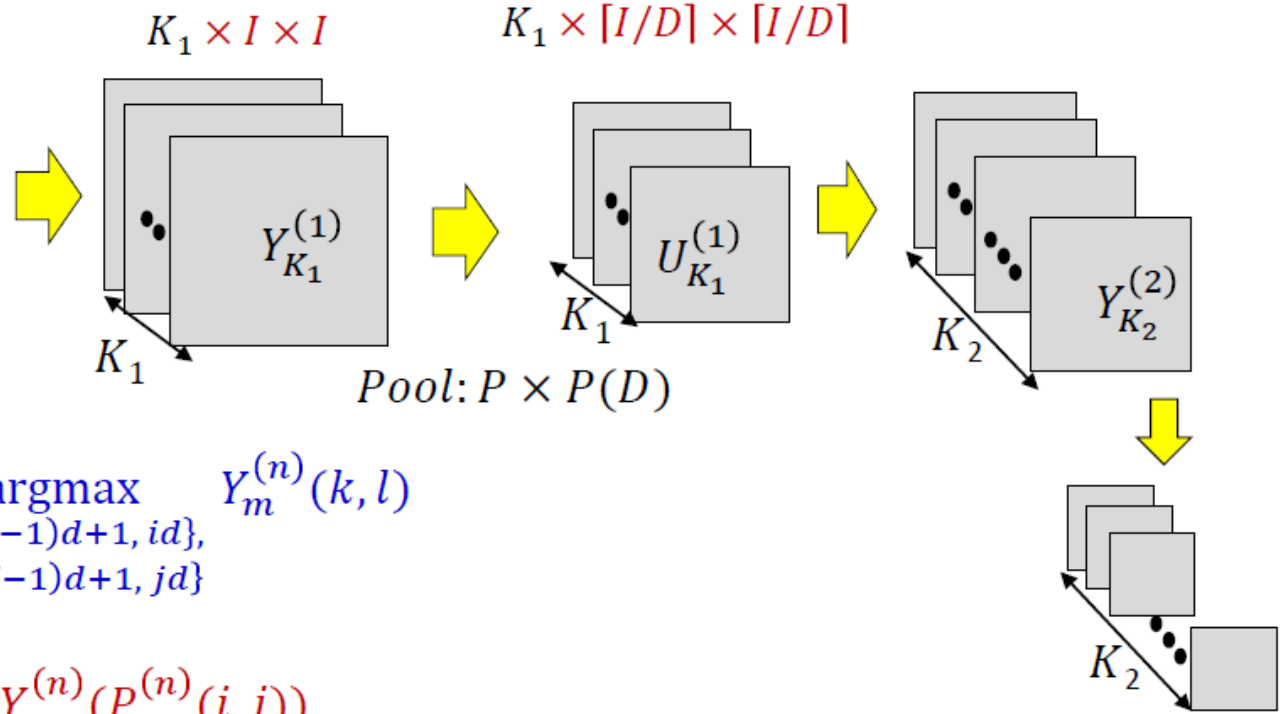
ng in K_2 2-D maps

Convolutional Neural Networks

$W_m: 3 \times L \times L$
$m = 1 \ldots K_1$

$W_m: K_1 \times L_2 \times L_2$
$m = 1 \ldots K_2$





$$P_m^{(n)}(i,j) = \underset{\substack{k \in \{(i-1)d+1,\, id\},\\ l \in \{(j-1)d+1,\, jd\}}}{\operatorname{argmax}} Y_m^{(n)}(k,l)$$

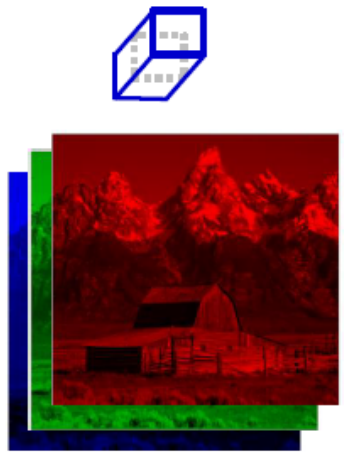
$$U_m^{(n)}(i,j) = Y_m^{(n)}(P_m^{(n)}(i,j))$$

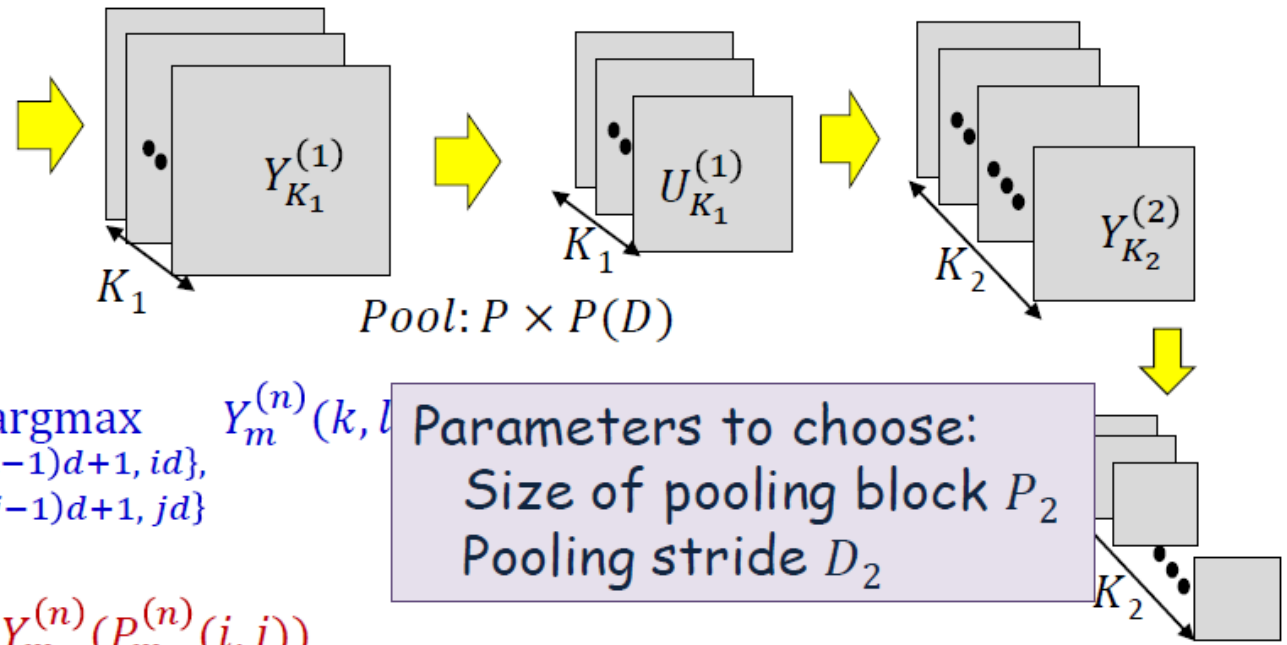
- **Second convolutional layer:** K_2 3-D filters resulting in K_2 2-D maps
- **Second pooling layer**: K_2 Pooling operations: outcome K_2 reduced 2D maps

Convolutional Neural Networks

$W_m: 3 \times L \times L$
$m = 1 \ldots K_1$

$W_m: K_1 \times L_2 \times L_2$
$m = 1 \ldots K_2$





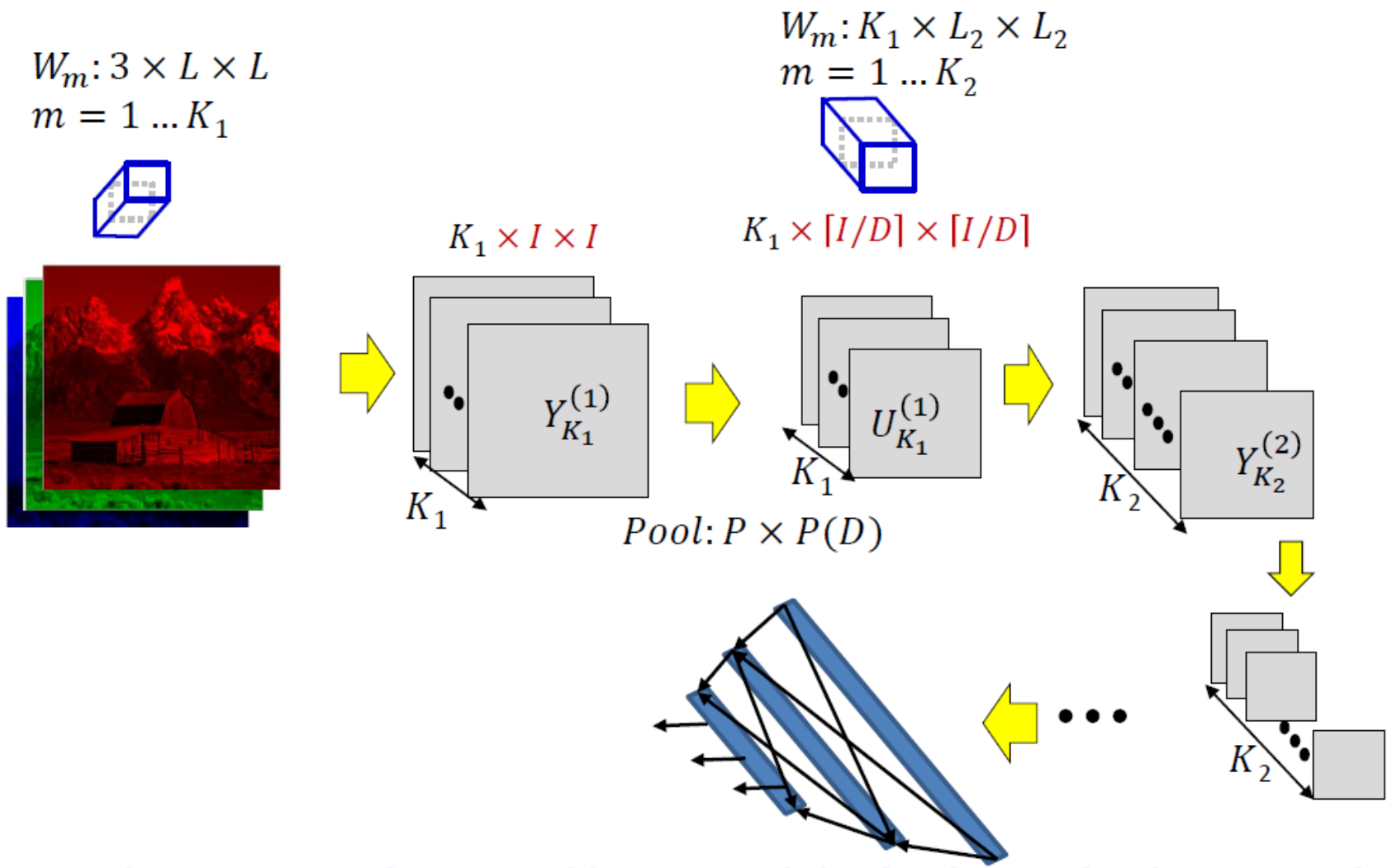
$$P_m^{(n)}(i,j) = \underset{\substack{k \in \{(i-1)d+1,\, id\},\\ l \in \{(j-1)d+1,\, jd\}}}{\operatorname{argmax}} Y_m^{(n)}(k, l$$

Parameters to choose:
Size of pooling block P_2
Pooling stride D_2

$$U_m^{(n)}(i,j) = Y_m^{(n)}(P_m^{(n)}(i,j))$$

- **Second convolutional layer:** K_2 3-D filters resulting in K_2 2-D maps
- **Second pooling layer**: K_2 Pooling operations: outcome K_2 reduced 2D maps

Convolutional Neural Networks



- This continues for several layers until the final convolved output is fed to a softmax
  - Or a full MLP i

The Size of the Layers

- Each convolution layer maintains the size of the image
  - With appropriate zero padding
  - If performed *without* zero padding it will decrease the size of the input
- Each convolution layer may *increase* the *number* of maps from the previous layer
- Each pooling layer with hop D *decreases* the *size* of the maps by a factor of D
- Filters within a layer must all be the same size, but sizes may vary with layer
  - Similarly for pooling, D may vary with layer
- In general the number of convolutional filters increases with layers

Parameters to choose (design choices)

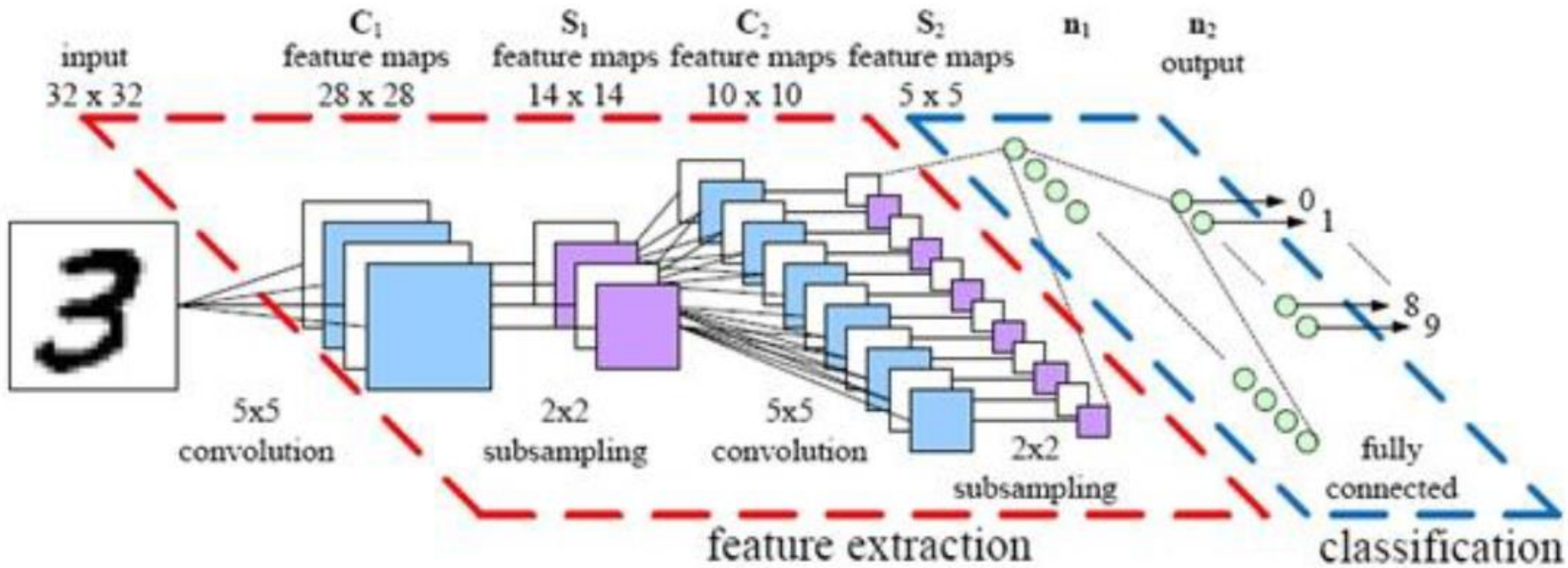
- Number of convolutional and downsampling layers
  - And arrangement (order in which they follow one another)

- For each convolution layer:
  - Number of filters K_i
  - Spatial extent of filter $L_i \times L_i$
    - The "depth" of the filter is fixed by the number of filters in the previous layer K_{i-1}
  - The stride S_i

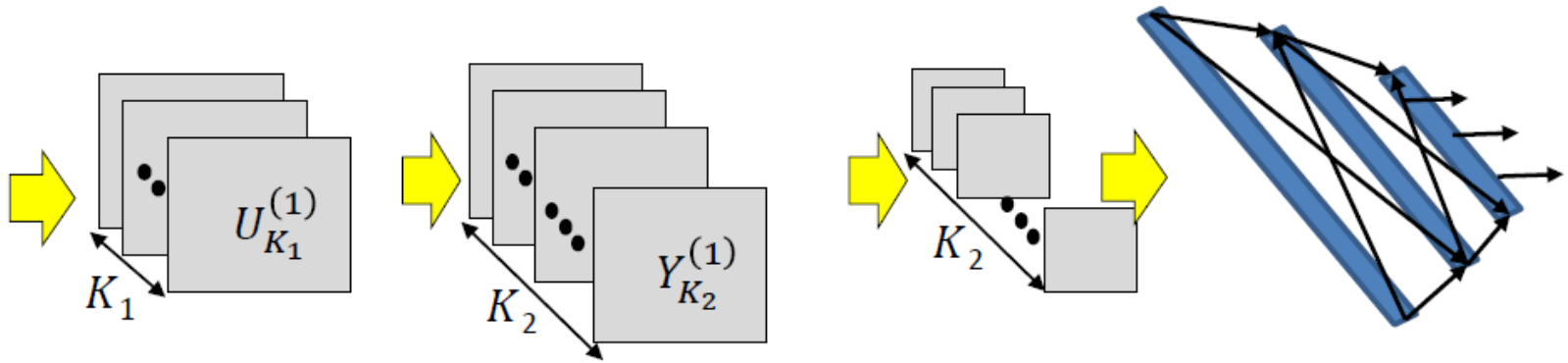
- For each downsampling/pooling layer:
  - Spatial extent of filter $P_i \times P_i$
  - The stride D_i

- For the final MLP:
  - Number of layers, and number of neurons in each layer

Digit classification

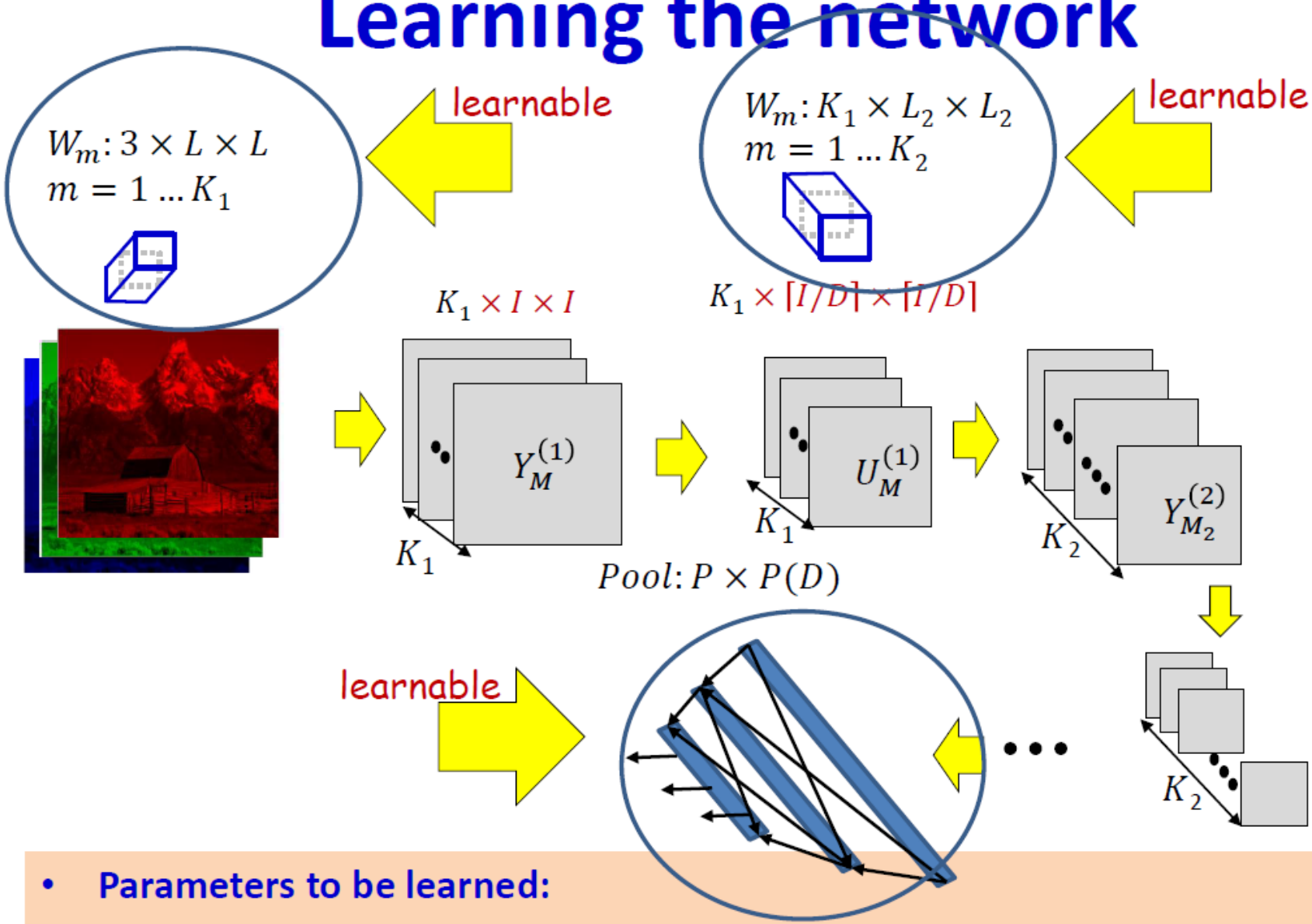


Training



- Training is as in the case of the regular MLP
  - The *only* difference is in the *structure* of the network
- **Training examples of (Image, class) are provided**
- Define a divergence between the desired output and true output of the network in response to any input
- **Network parameters are trained through variants of gradient descent**
- **Gradients are computed through backpropagation**

Learning the network



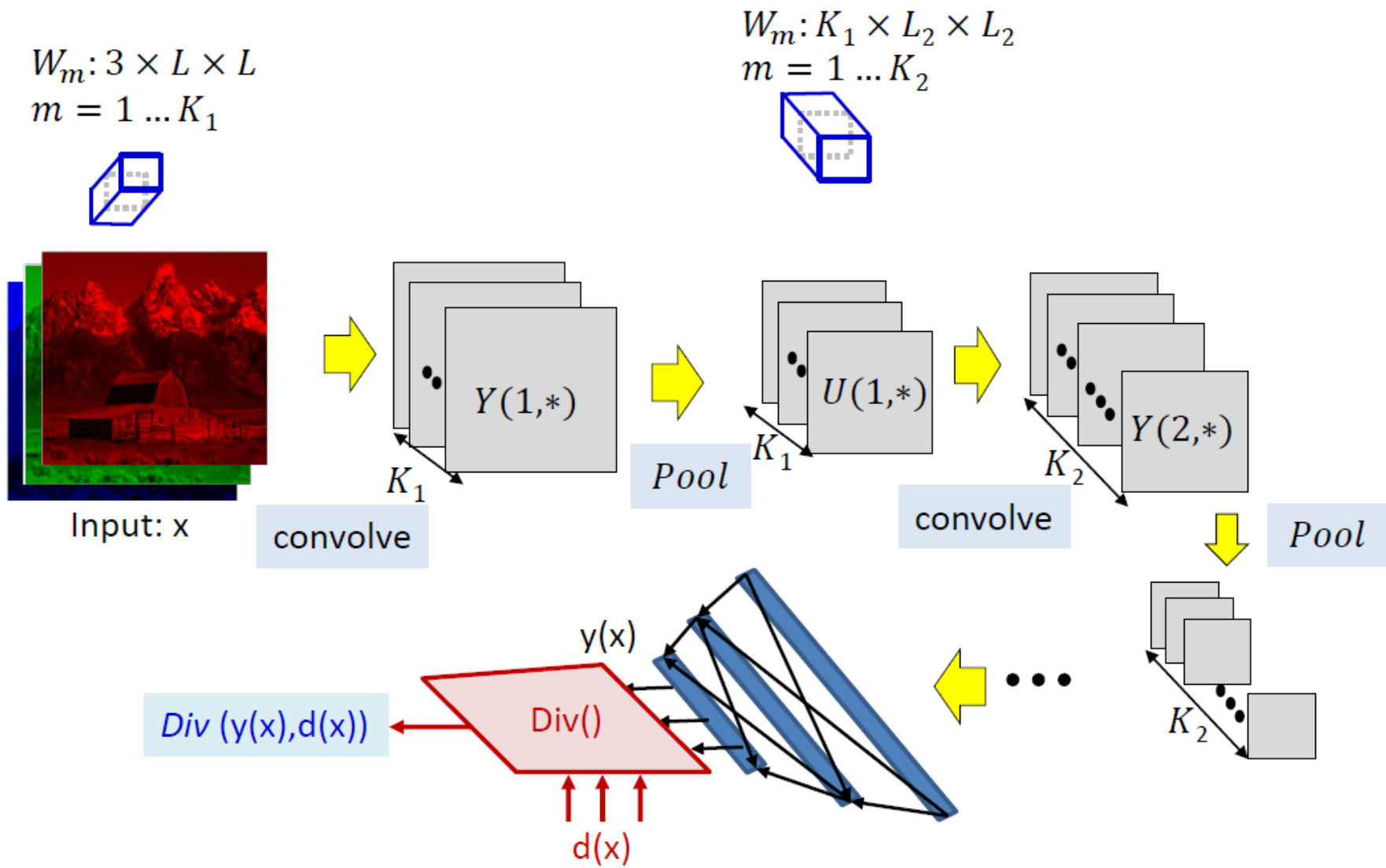
- **Parameters to be learned:**
  - **The weights of the neurons in the final MLP**
  - **The (weights and biases of the) filters for every *convolutional* layer**

Learning the CNN

- In the final "flat" multi-layer perceptron, all the weights and biases of each of the perceptrons must be learned

- In the *convolutional layers* the filters must be learned
- Let each layer J have K_J maps
  - K_0 is the number of maps (colours) in the input
- Let the filters in the J^{th} layer be size $L_J \times L_J$
- For the J^{th} layer we will require $K_J\left(K_{J-1}L_J^2 + 1\right)$ filter parameters
- Total parameters required for the *convolutional* layers:
  $\sum_{J \in convolutional\ layers} K_J\left(K_{J-1}L_J^2 + 1\right)$

Defining the loss



- The loss for a single instance

Problem Setup

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$

- The loss on the ith instance is $div(Y_i, d_i)$
- The total loss

$$Loss = \frac{1}{T}\sum_{i=1}^{T} div(Y_i, d_i)$$

- Minimize $Loss$ w.r.t $\{W_m, b_m\}$

Training CNNs through Gradient Descent

Total training loss:

$$Loss = \frac{1}{T}\sum_{i=1}^{T} div(Y_i, d_i)$$

Assuming the bias is also represented as a weight

- Gradient descent algorithm:
- Initialize all weights and biases $\{w(:,:,:,:,:)\}$
- Do:
  - For every layer l for all filter indices m, update:
    - $w(l,m,j,x,y) = w(l,m,j,x,y) - \eta \frac{dLoss}{dw(l,m,j,x,y)}$
- Until $Loss$ has converged

Training CNNs through Gradient Descent

Total training loss:

$$Loss = \frac{1}{T}\sum_{i=1}^{T} div(Y_i, d_i)$$

Assuming the bias is also represented as a weight

- Gradient descent algorithm:
- Initialize all weights and biases $\{w(:,:,:,:,:)\}$
- Do:
  - For every layer l for all filter indices m, update:
    - $w(l,m,j,x,y) = w(l,m,j,x,y) - \eta \frac{dLoss}{dw(l,m,j,x,y)}$
- Until $Loss$ has converged

The derivative

Total training loss:

$$Loss = \frac{1}{T}\sum_{i} Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

$$\frac{dLoss}{dw(l,m,j,x,y)} = \frac{1}{T}\sum_{i} \frac{dDiv(Y_i, d_i)}{dw(l,m,j,x,y)}$$

The derivative

Total training loss:

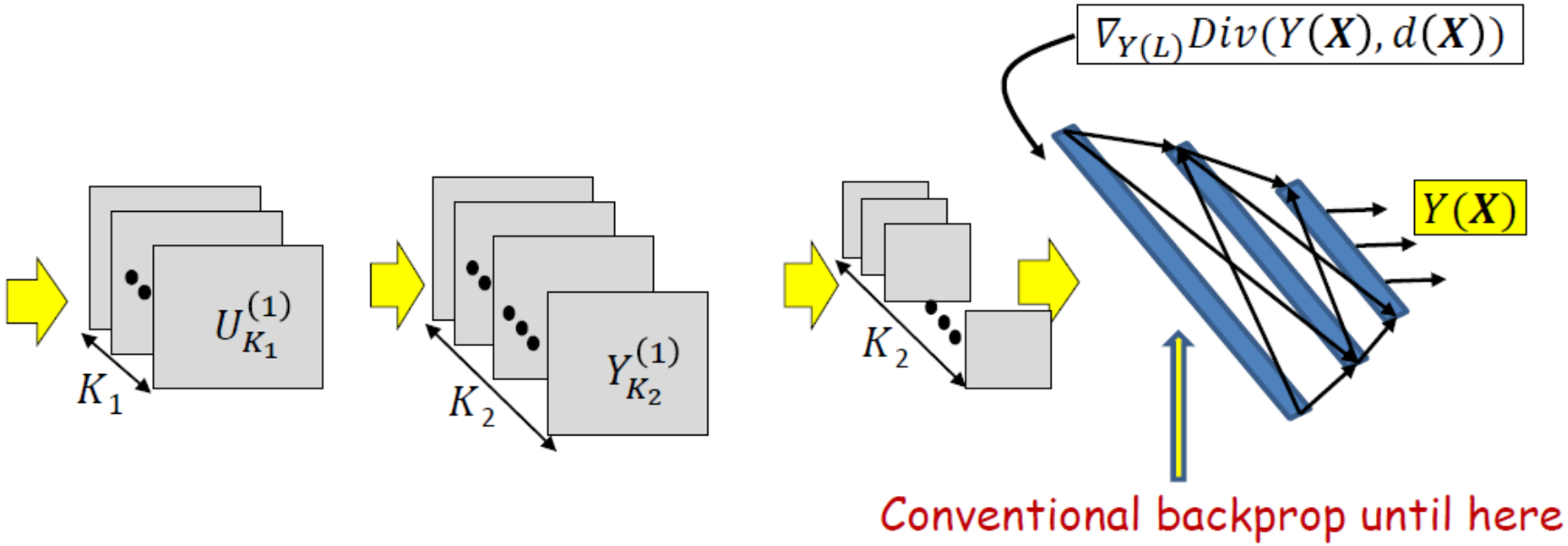
$$Loss = \frac{1}{T}\sum_{i} Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

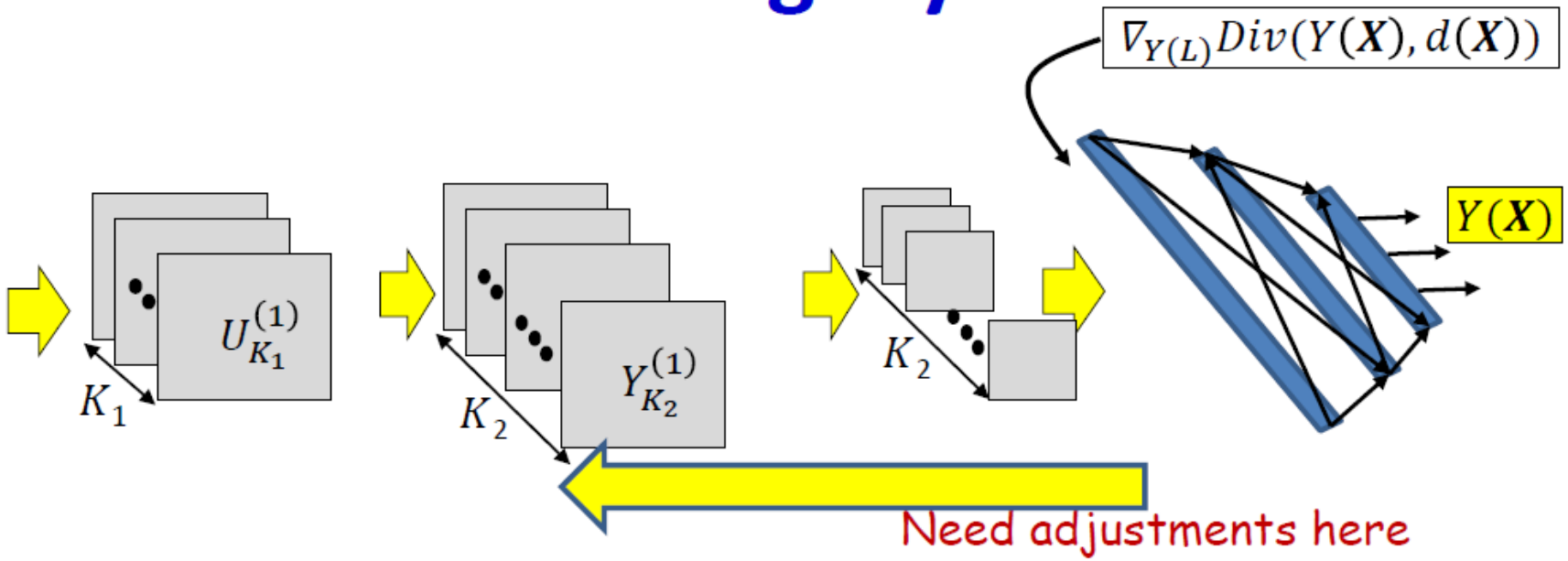
$$\frac{dLoss}{dw(l,m,j,x,y)} = \frac{1}{T}\sum_{i} \frac{dDiv(Y_i, d_i)}{dw(l,m,j,x,y)}$$

Backpropagation: Final flat layers



- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first "flat" layer
  - Important to recall: the first flat layer is only the "unrolling" of the maps from the final convolutional layer

Backpropagation: Convolutional and Pooling layers



- Backpropagation from the flat MLP requires special consideration of
  - The shared computation in the convolution layers
  - The pooling layers (particularly maxout)

CNN Architectures

Table 2 Brief overview of CNN architectures

Model	Main finding	Depth	Dataset	Error rate	Input size	Year
AlexNet	Utilizes Dropout and ReLU	8	ImageNet	16.4	227 × 227 × 3	2012
NIN	New layer, called 'mlpconv', utilizes GAP	3	CIFAR-10, CIFAR-100, MNIST	10.41, 35.68, 0.45	32 × 32 × 3	2013
ZfNet	Visualization idea of middle layers	8	ImageNet	11.7	224 × 224 × 3	2014
VGG	Increased depth, small filter size	16, 19	ImageNet	7.3	224 × 224 × 3	2014
GoogLeNet	Increased depth,block concept, different filter size, concatenation concept	22	ImageNet	6.7	224 × 224 × 3	2015
Inception-V3	Utilizes small filtersize, better feature representation	48	ImageNet	3.5	229 × 229 × 3	2015
Highway	Presented the multipath concept	19, 32	CIFAR-10	7.76	32 × 32 × 3	2015
Inception-V4	Divided transform and integration concepts	70	ImageNet	3.08	229 × 229 × 3	2016
ResNet	Robust against overfitting due to symmetry mapping-based skip links	152	ImageNet	3.57	224 × 224 × 3	2016
Inception-ResNet-v2	Introduced the concept of residual links	164	ImageNet	3.52	229 × 229 × 3	2016
FractalNet	Introduced the concept of Drop-Path as regularization	40,80	CIFAR-10 CIFAR-100	4.60 18.85	32 × 32 × 3	2016
WideResNet	Decreased the depth and increased the width	28	CIFAR-10 CIFAR-100	3.89 18.85	32 × 32 × 3	2016
Xception	A depthwise convolutionfollowed by a pointwise convolution	71	ImageNet	0.055	229 × 229 × 3	2017
Residual attention neural network	Presented the attention technique	452	CIFAR-10, CIFAR-100	3.90, 20.4	40 × 40 × 3	2017
Squeeze-and-excitation networks	Modeled interdependencies between channels	152	ImageNet	2.25	229 × 229 × 3 224 × 224 × 3 320 × 320 × 3	2017
DenseNet	Blocks of layers; layers connected to each other	201	CIFAR-10, CIFAR-100,ImageNet	3.46, 17.18, 5.54	224 × 224 × 3	2017
Competitive squeeze and excitation network	Both residual and identity mappings utilized to rescale the channel	152	CIFAR-10 CIFAR-100	3.58 18.47	32 × 32 × 3	2018
MobileNet-v2	Inverted residual structure	53	ImageNet	–	224 × 224 × 3	2018
CapsuleNet	Pays attention to special relationships between features	3	MNIST	0.00855	28 × 28 × 1	2018
HRNetV2	High-resolution representations	–	ImageNet	5.4	224 × 224 × 3	2020

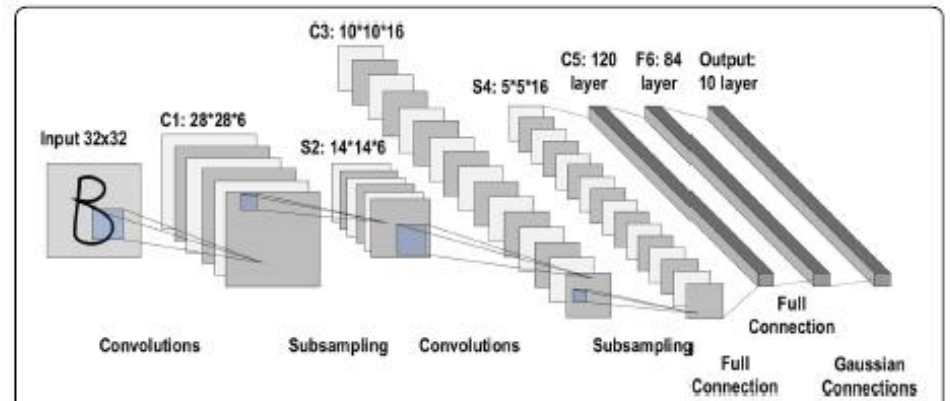


Fig. 14 The architecture of LeNet

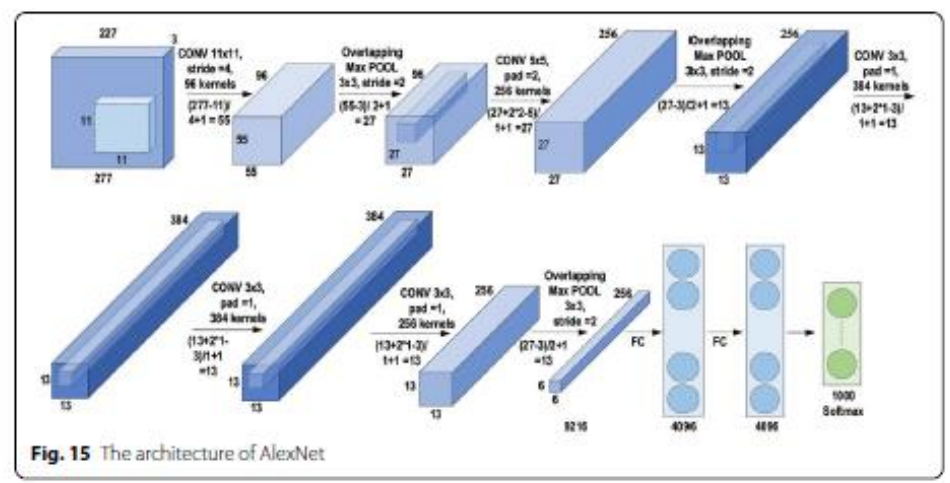


Fig. 15 The architecture of AlexNet