# Convolutional Neural Network

CSE 849 Deep Learning

Spring 2025

Zijun Cui

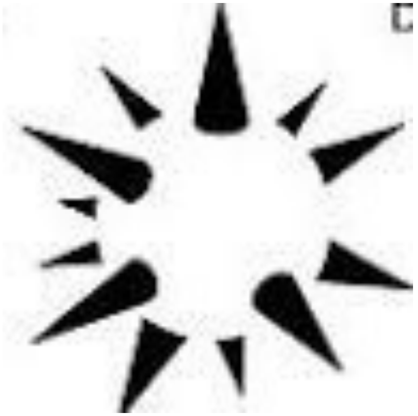
Partial slides are adapted from CMU 11-785 Introduction to Deep Learning by Bhiksha Raj

# Outline

- History of CNN
- General architecture of CNN
    - Convolutional layer
    - Pooling layer
    - Downsampling and upsampling

# How do animals see



- How do animals see?
  - What is the neural process from eye to recognition?
- Research:
  - how the brain processed images

# NeoCognitron

- Kunihiko Fukushima
- NeoCognitron (1980)
  - Visual system consists of a hierarchy of modules, each comprising a layer of "S-cells" followed by a layer of "C-cells"

Figures from Fukushima, '80

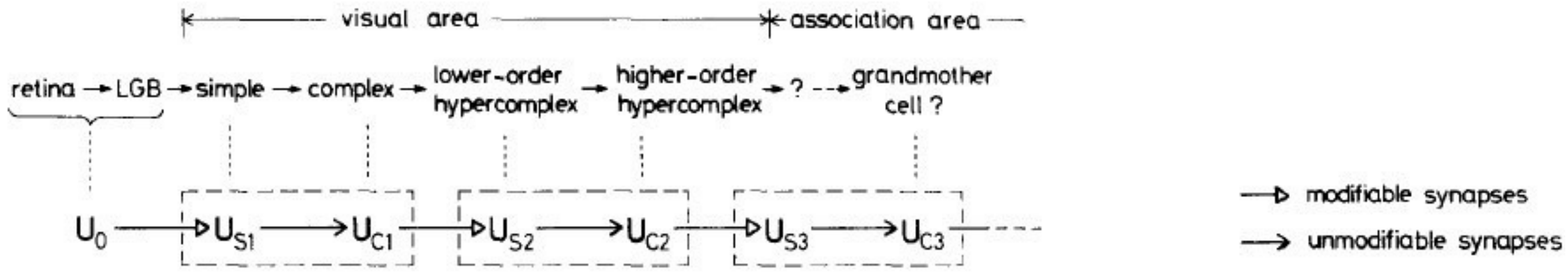
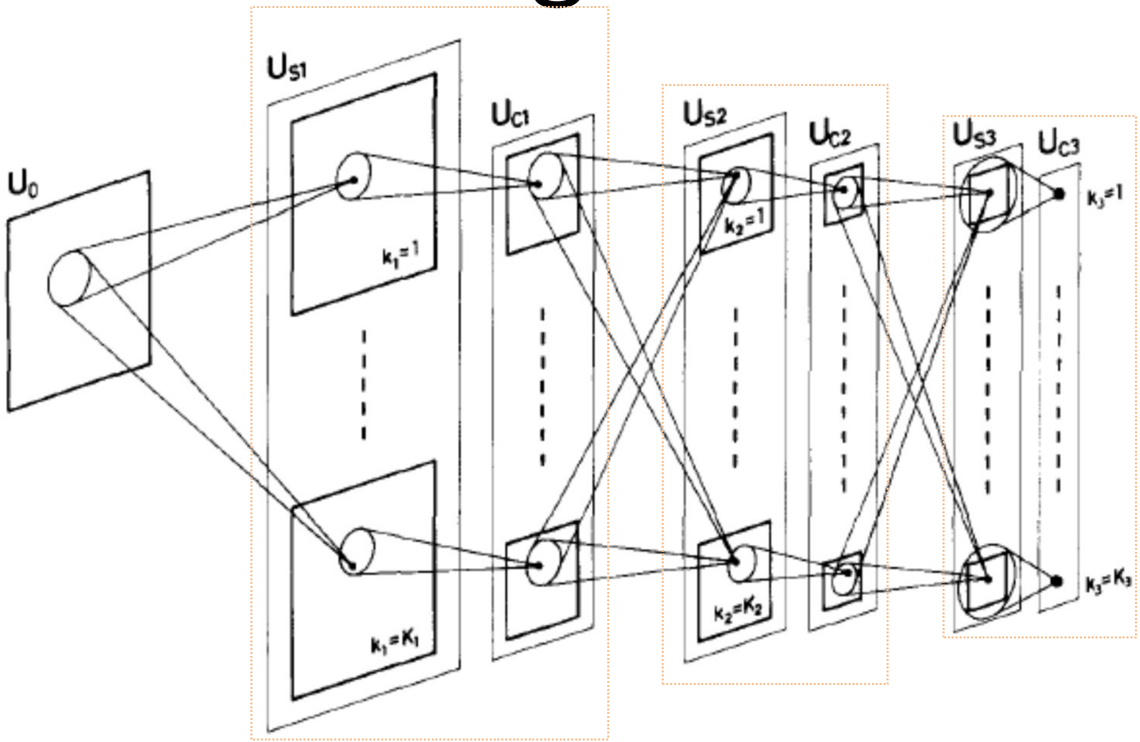



**Fig. 1.** Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

# NeoCognitron




- The complete network
- $U_0$ is the retina

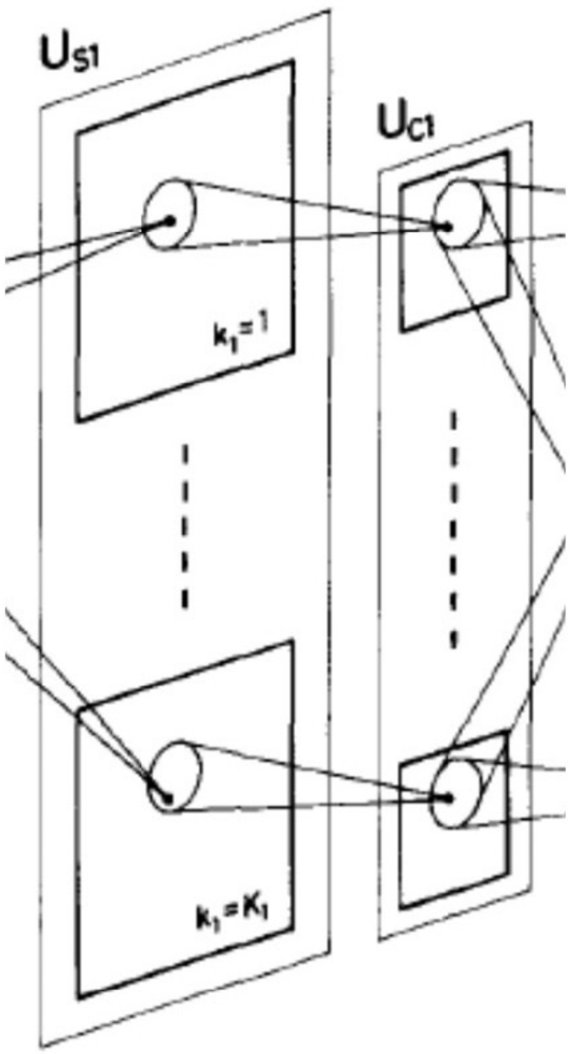
- In each subsequent module, the planes of the S layers detect plane-specific patterns in the previous layer (C layer or retina)

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

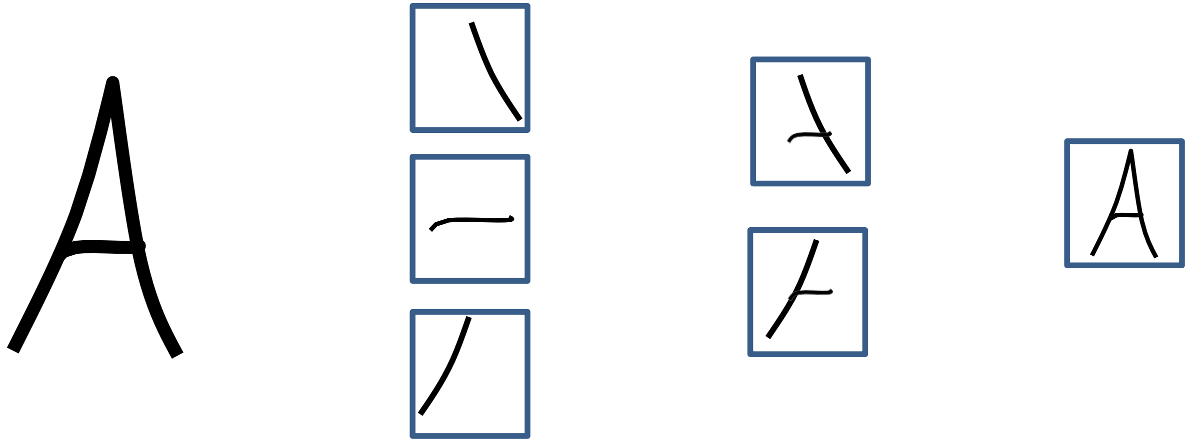
# S cell and C cell

- S-cells respond to the signal in the previous layer; C-cells confirm the S-cells' response
- Only S-cells are "plastic" (i.e. learnable), C-cells are fixed in their response

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

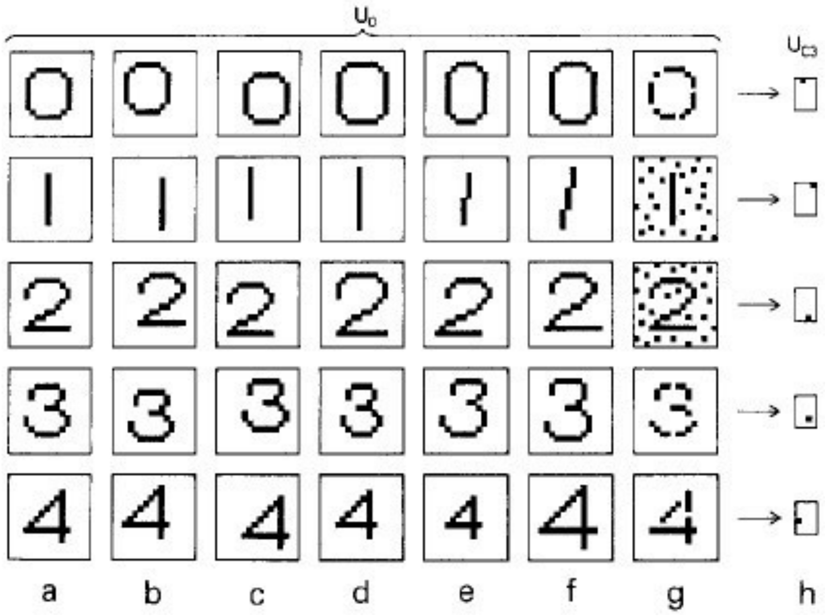


# Learning in the neocognitron



- **Unsupervised learning**
- Ensures different planes learn different features
  - 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

# Neocognitron – finale



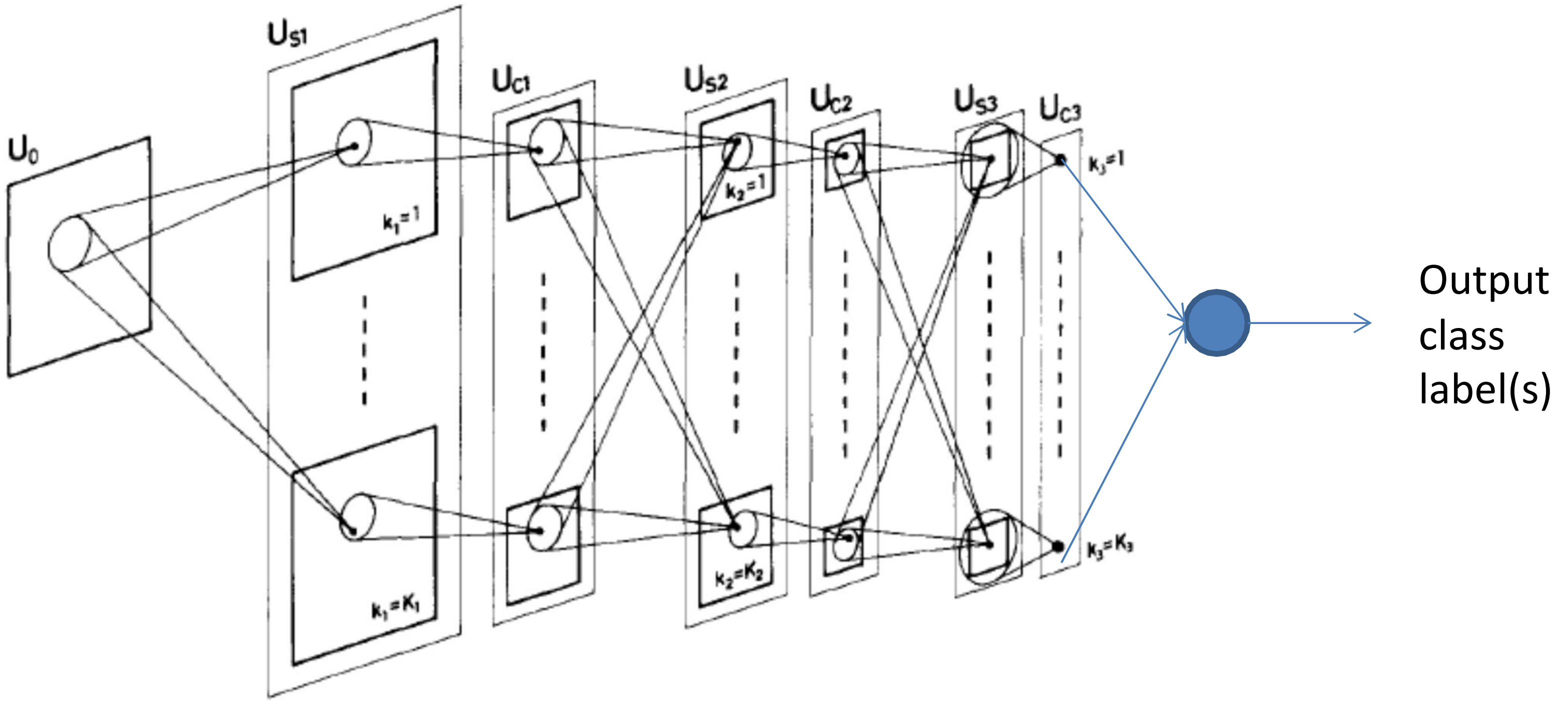

For an input digit, the model activates a specific neuron that represents the category of that digit.

- Fukushima showed it successfully learns to cluster semantic visual concepts
  - E.g. number or characters, even in noise

# Adding Supervision

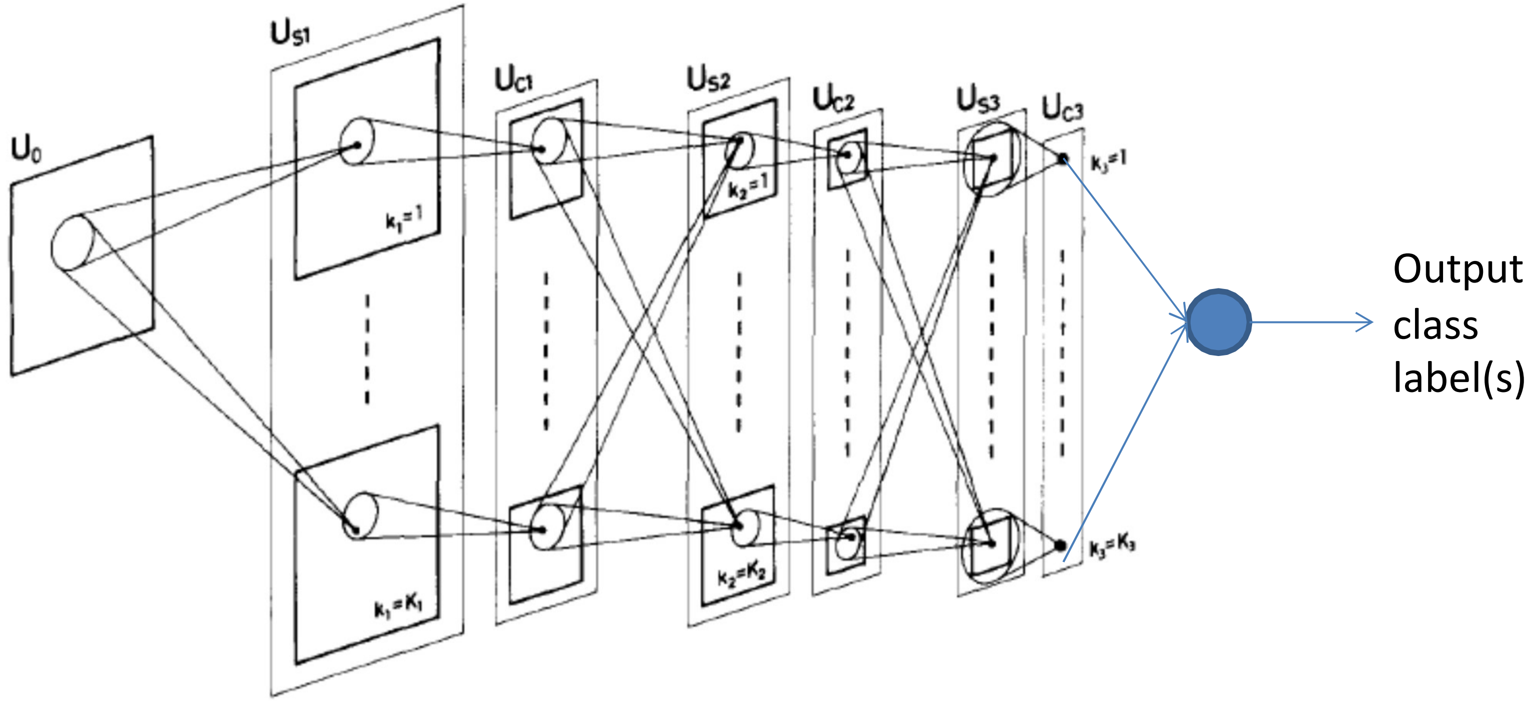
- The neocognitron is fully unsupervised
  - Semantic labels are automatically learned
- Can we add external supervision?
- Various proposals:
  - Temporal correlation: Homma, Atlas, Marks, ‘88
  - TDNN: Lang, Waibel et. al., 1989, ‘90
- Convolutional neural networks

# 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

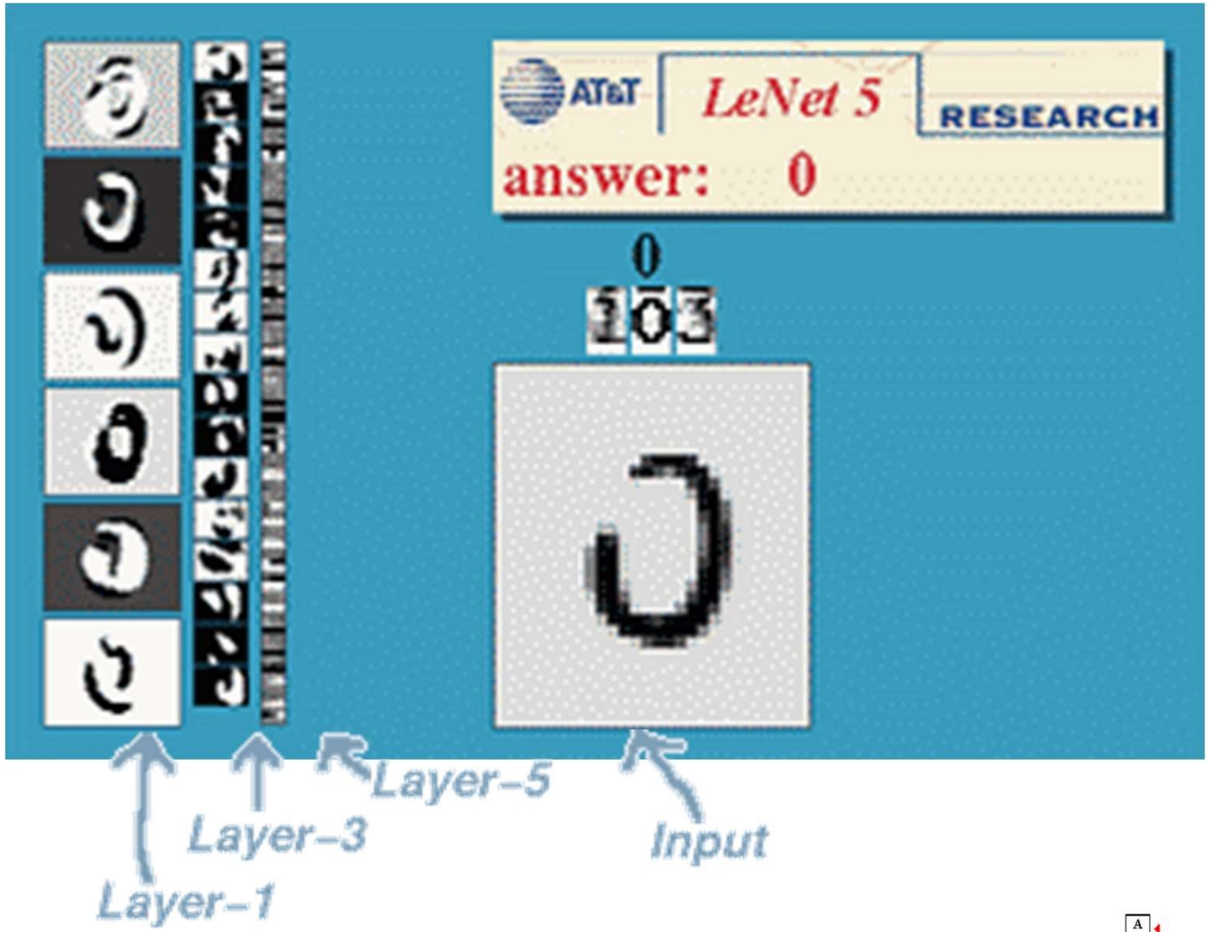
# Supervising the neocognitron

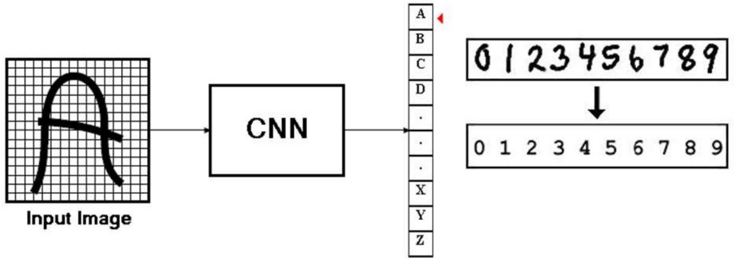



- S planes of cells are modelled by a scan (convolution) over image planes by a single neuron
  - Convolutional layer
- C planes are emulated by cells that perform a max over groups of S cells
  - Pooling layer
- Giving us a "Convolutional Neural Network"

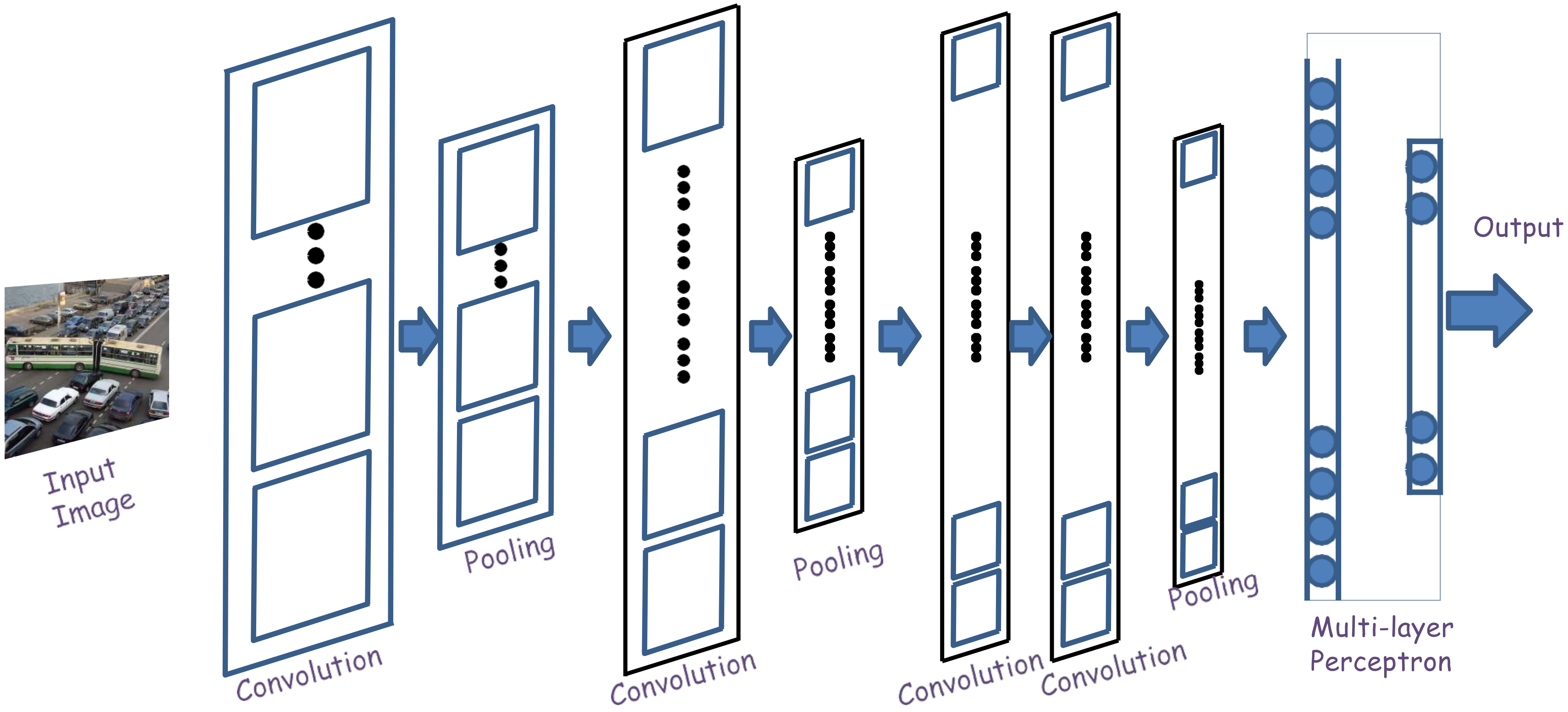
LeNet (Lecun, Y, 1998)

# Convolutional Neural Networks



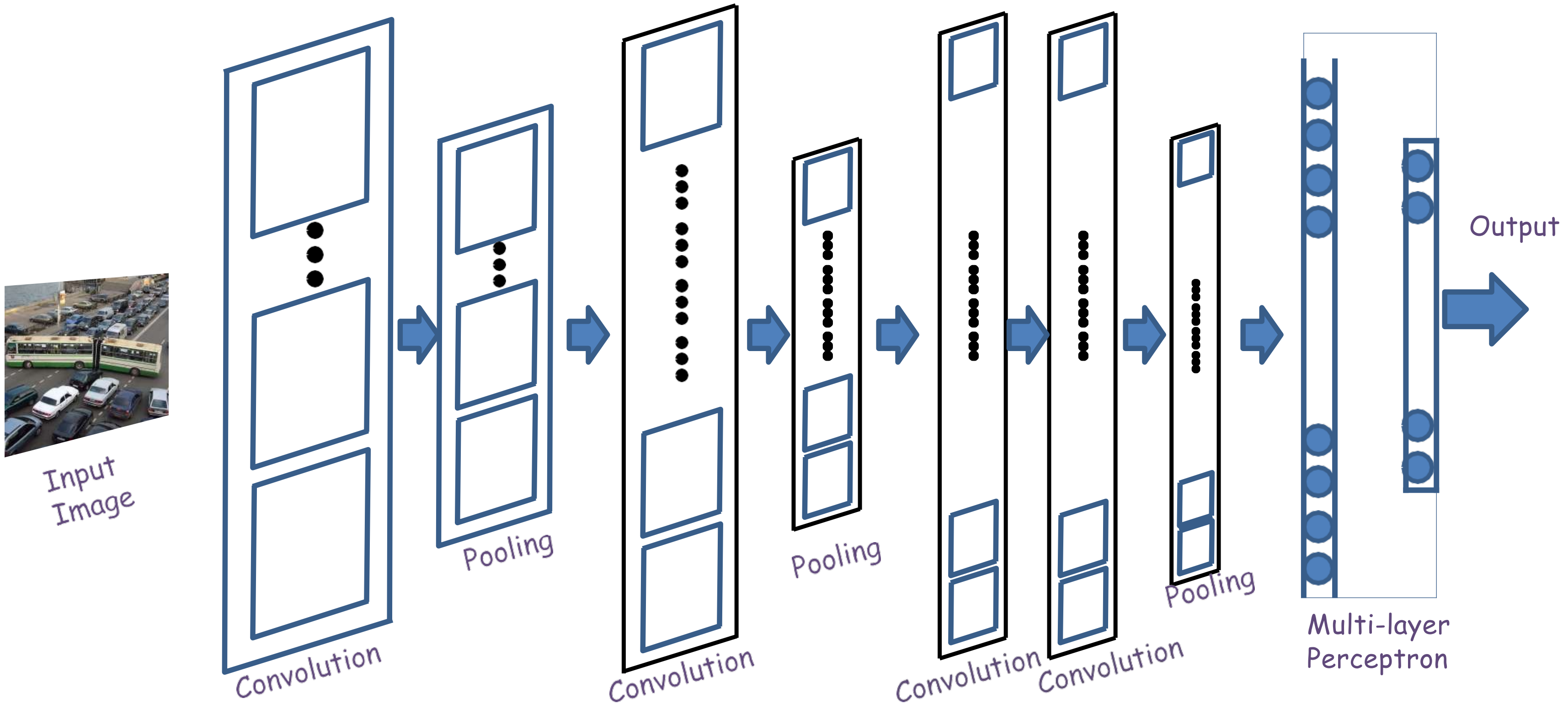



# The general architecture of a CNN



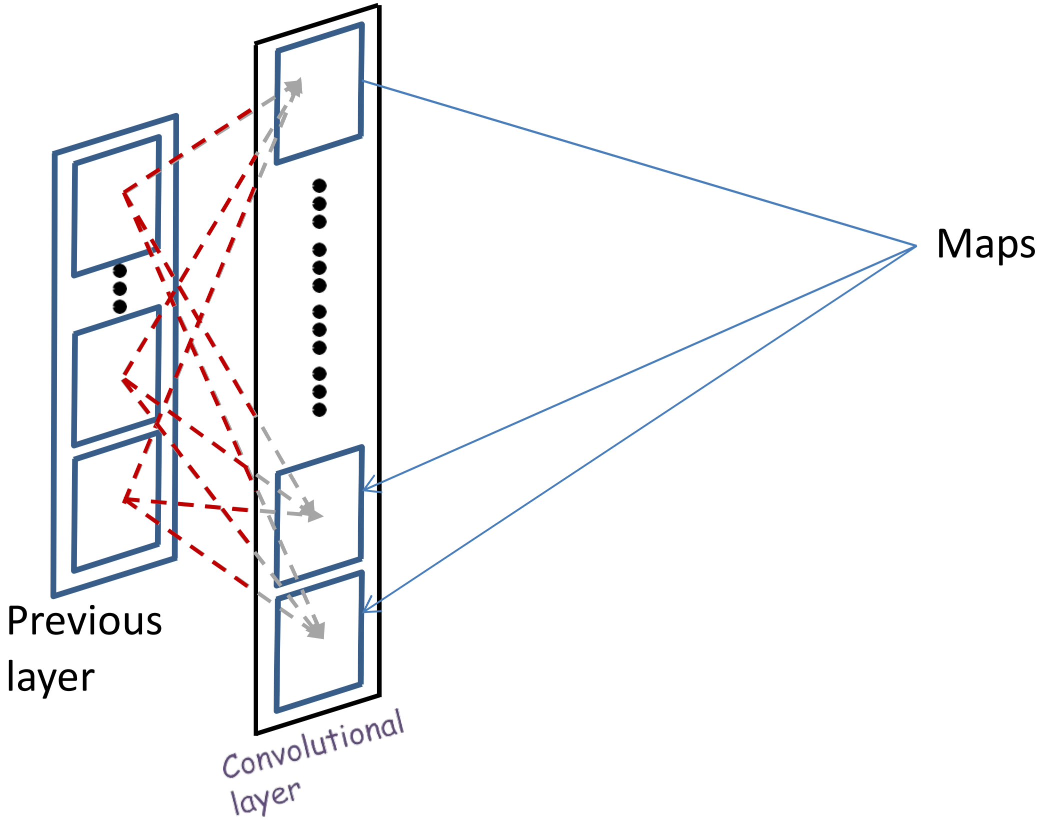

- A convolutional neural network comprises "convolutional" and "pooling" layers
  - Convolutional layers comprise neurons that scan their input for patterns
    - Correspond to S planes
  - Pooling layers perform max operations on groups of outputs from the convolutional layers
    - Correspond to C planes
  - **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 CNN



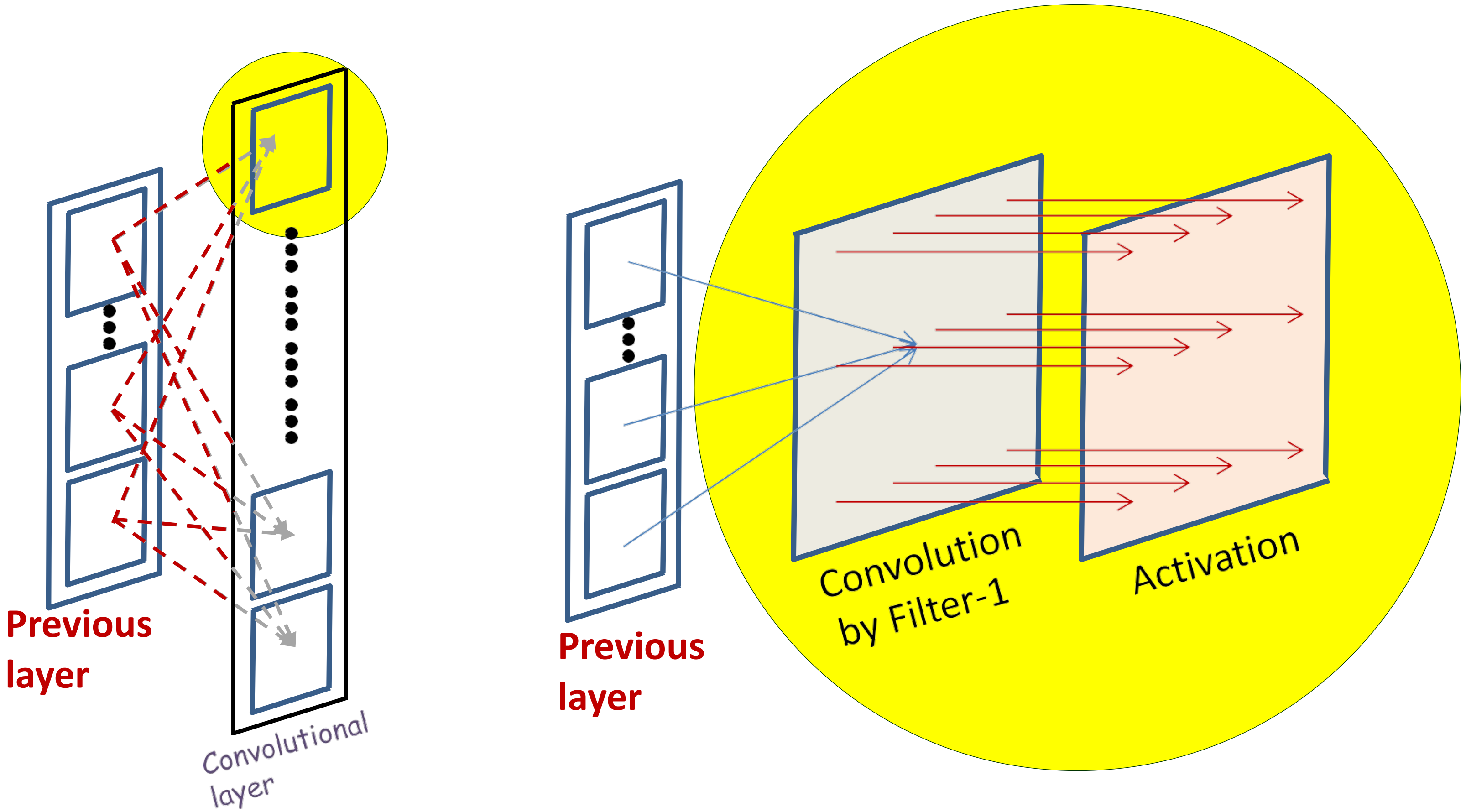

- **Convolutional layers and the MLP are *learnable***
  - Their parameters must be learned from training data for the target classification task
- Pooling layers are fixed and generally not learnable

# A convolutional layer



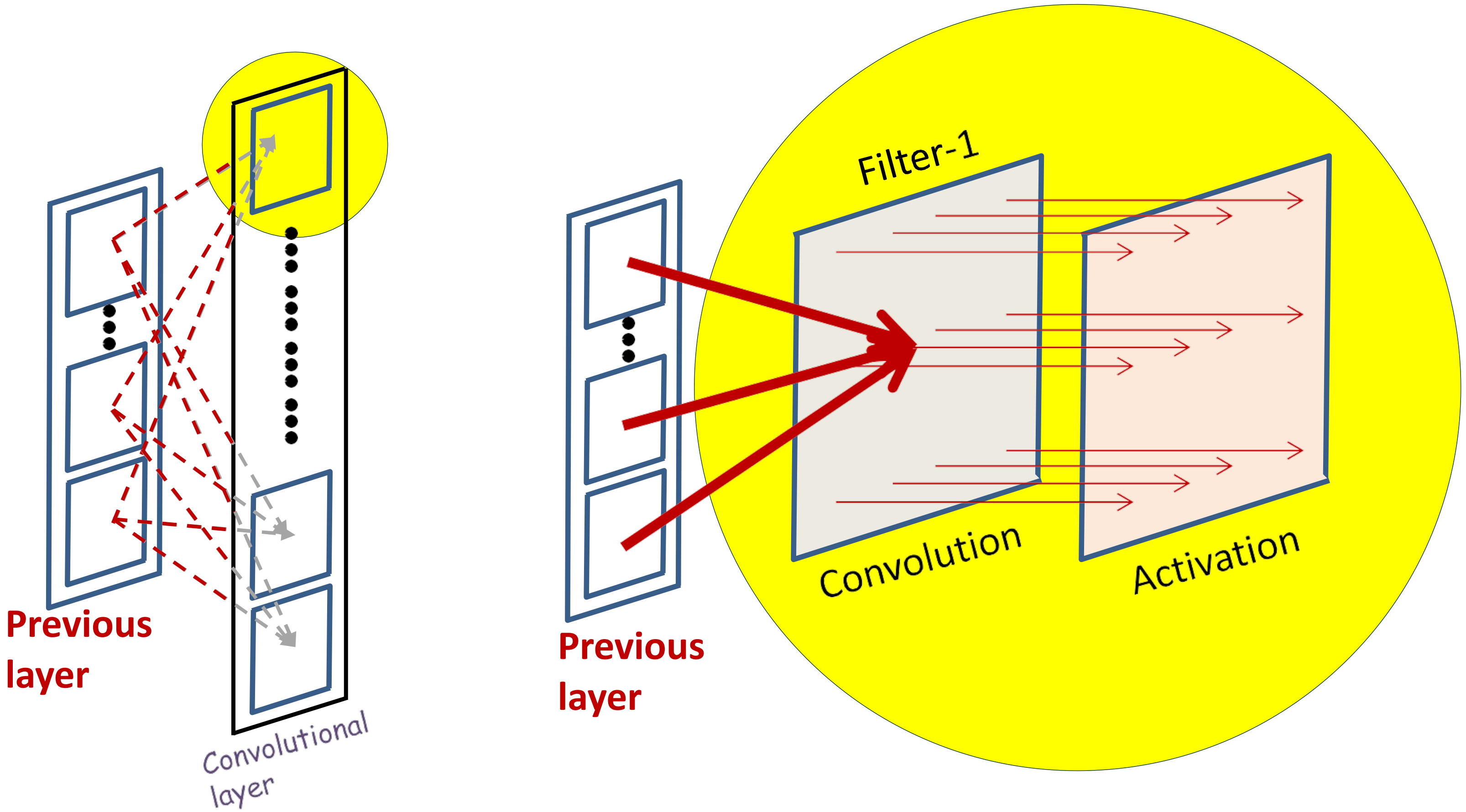

- A convolutional layer comprises of a series of “maps”
  - Corresponding the “S-planes” in the Neocognitron
  - Variously called feature maps or activation maps

# A convolutional layer



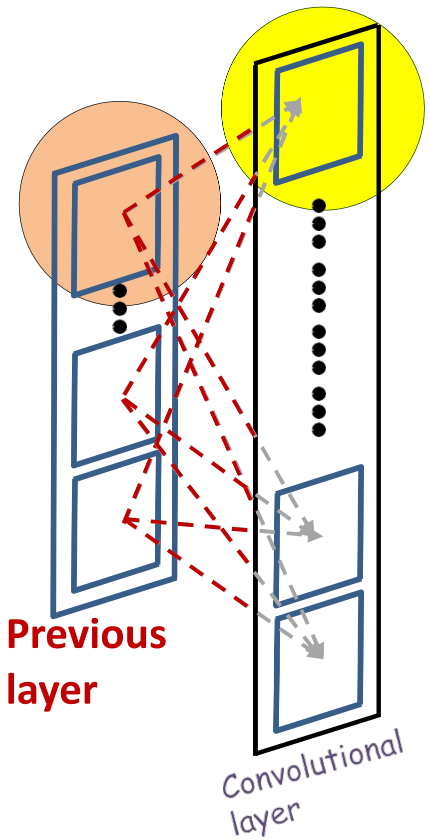

- Each activation map has two components
  - An *affine* map, obtained by *convolution* over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates on the output of the convolution

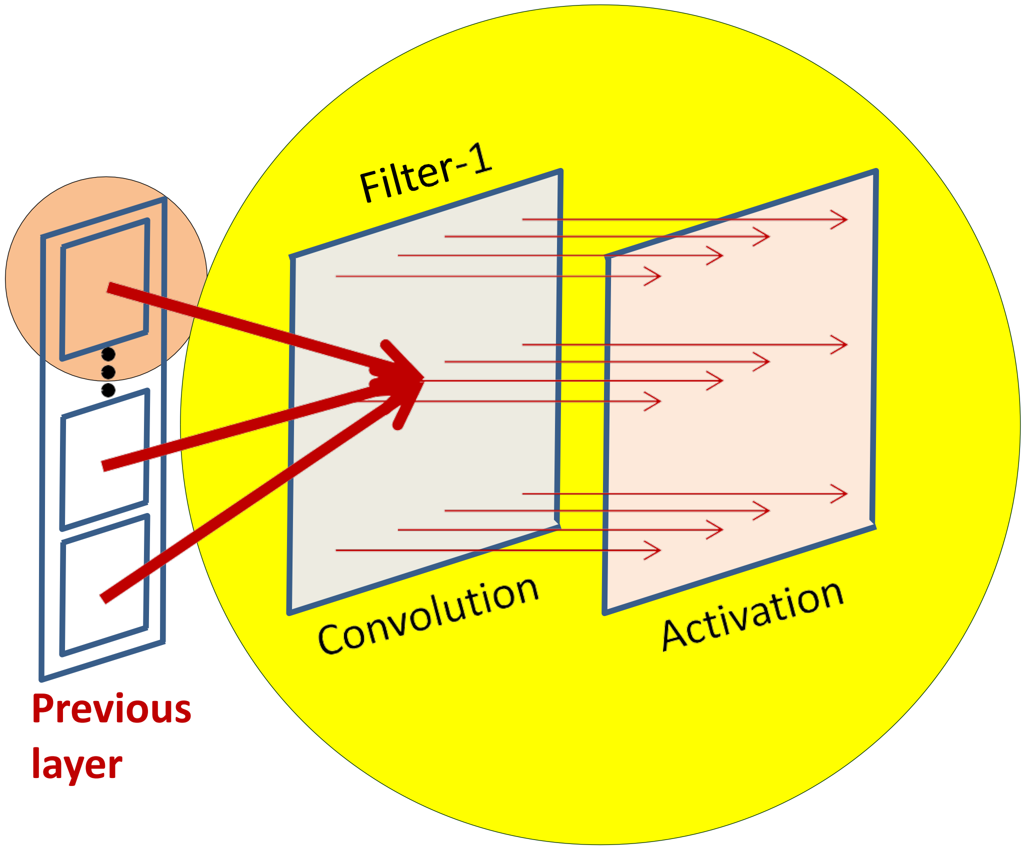
# A convolutional layer: affine map




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

# A convolutional layer: affine map




- All the maps in the previous layer contribute to each convolution
  - In the following, we 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 |

**Input Map**

Example 3x3 filter

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

bias

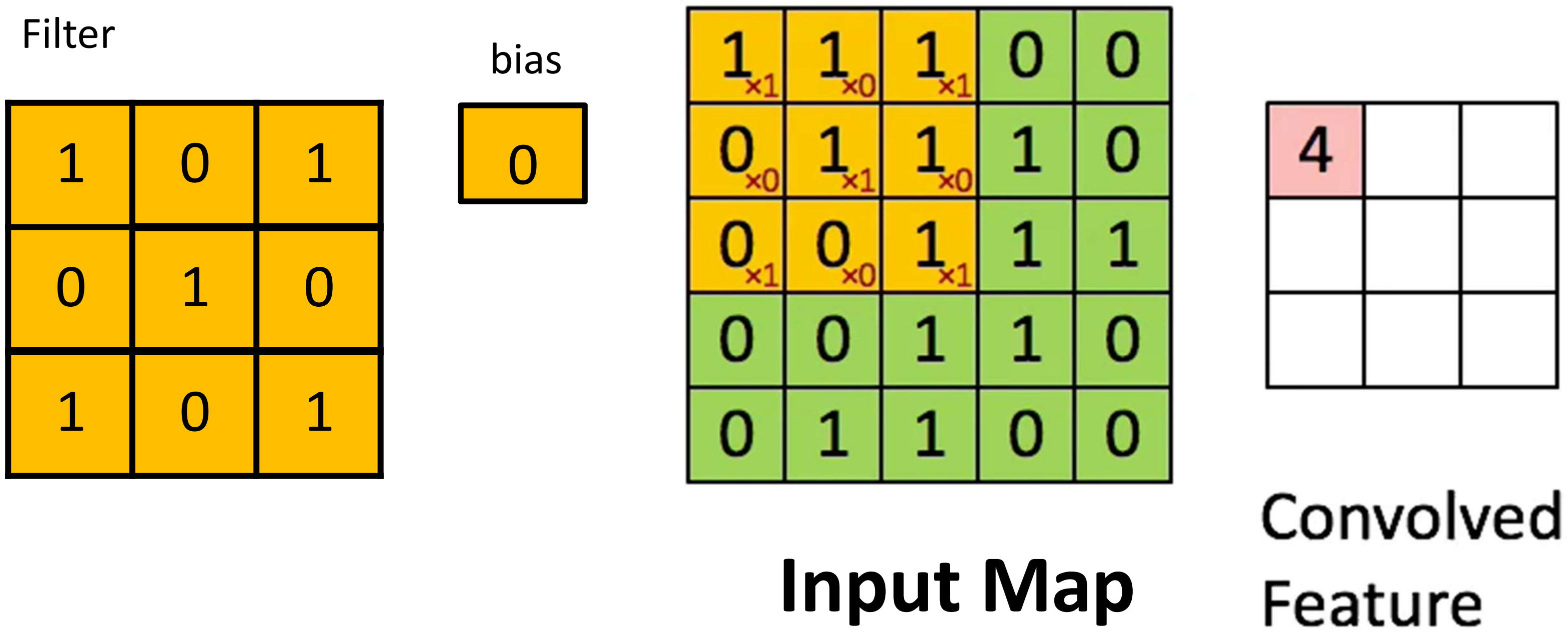
| |
|---|
| 0 |

Elements are all learnable
We specify values here for illustration purpose

- Scanning an image (generally speaking, a feature map) with a “filter”
  – Note: a filter is really just a perceptron, with weights and a bias

Jargon: filters are often called “**Kernels**”

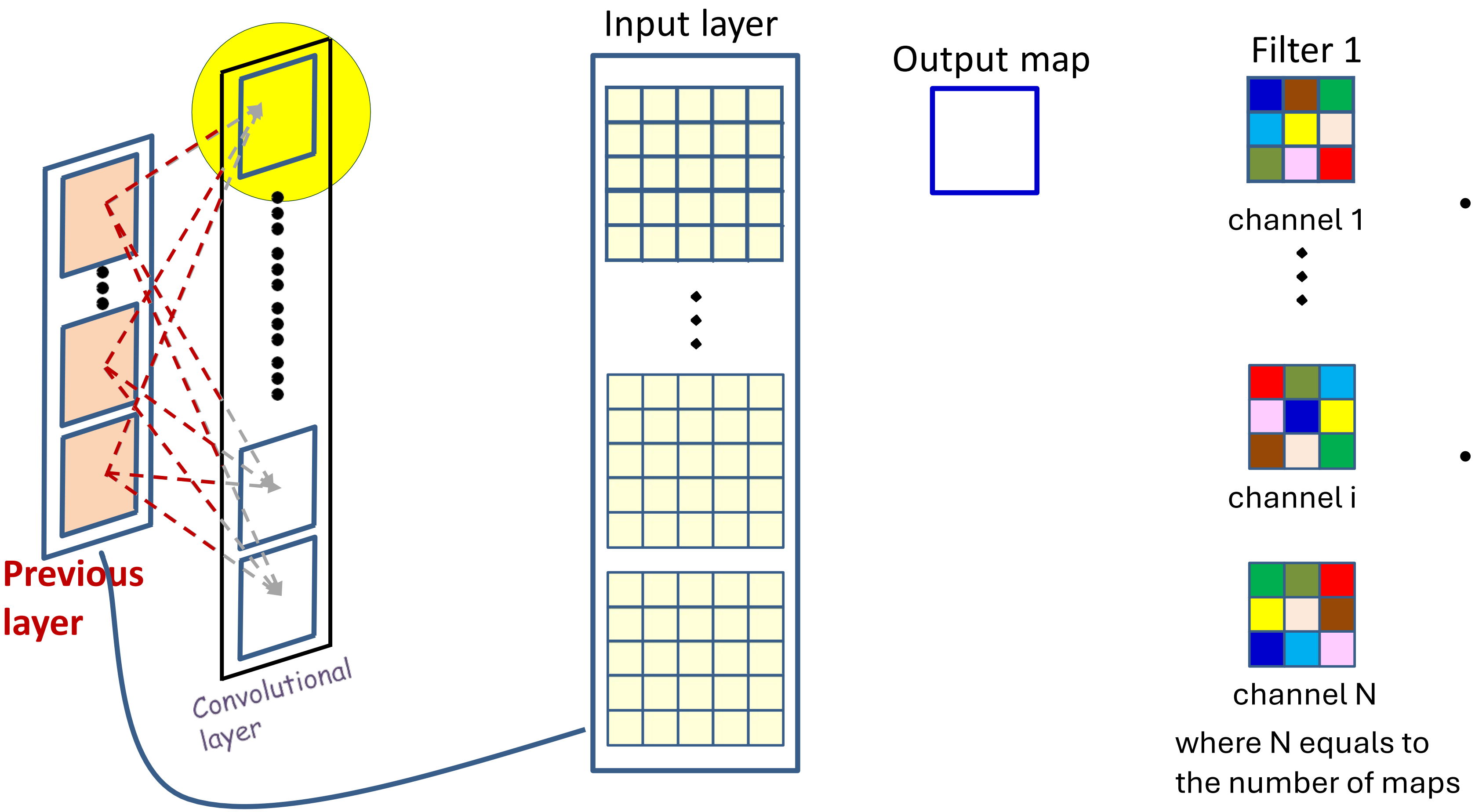
# What is a convolution




- 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

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

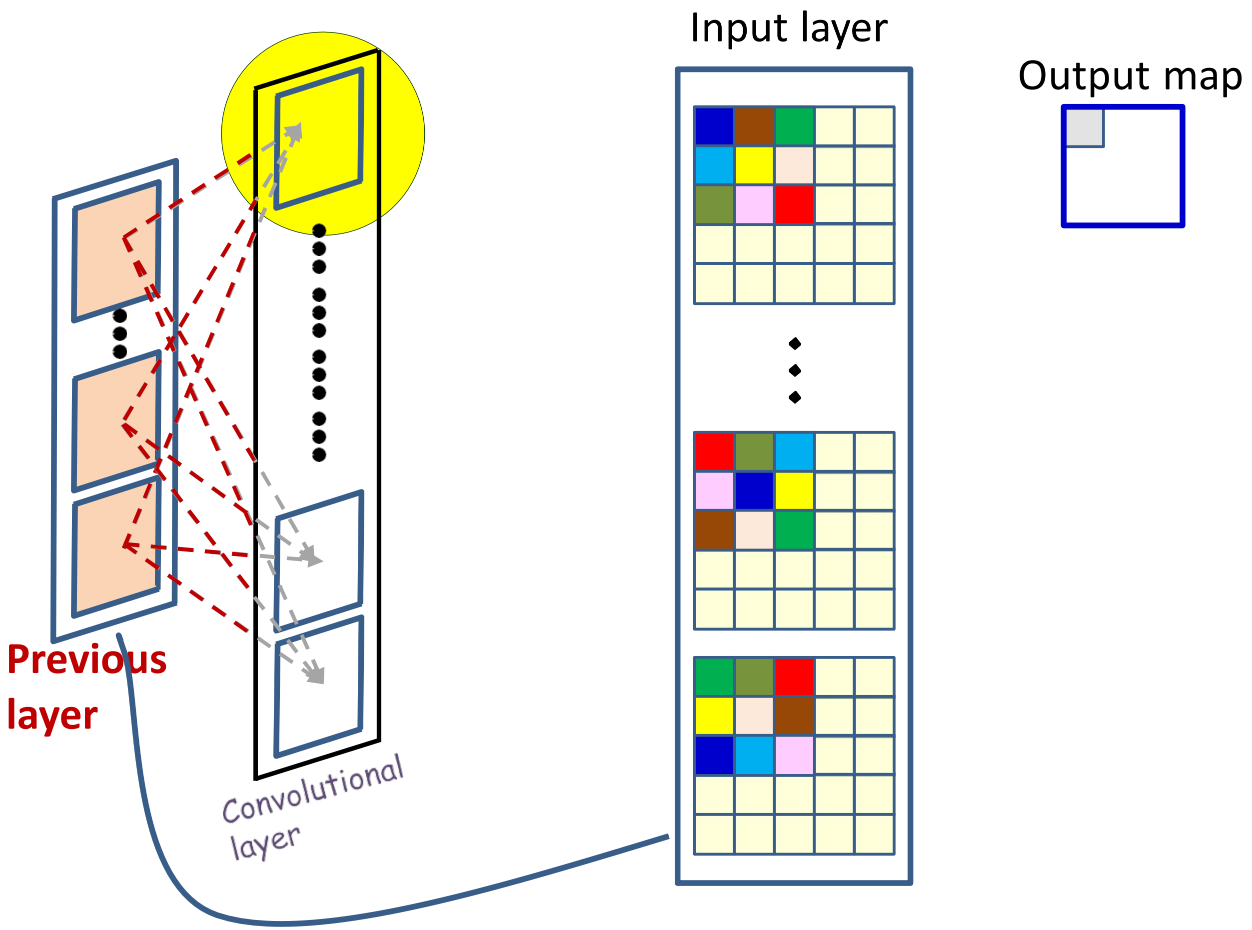
# With multiple maps



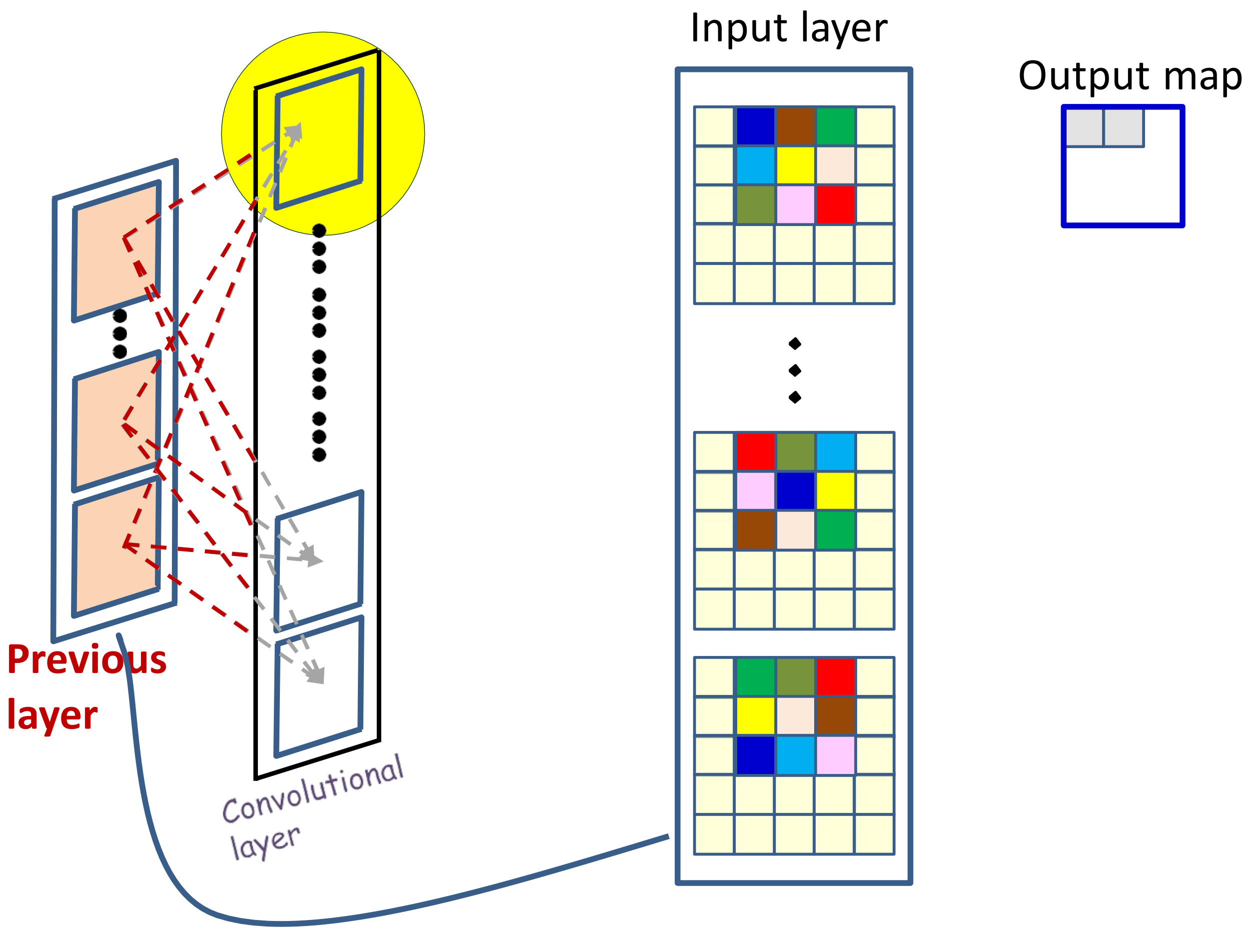

- Each filter has **multiple channels** (corresponding to the number of input maps).
- Each channel **processes one input map**, and the results are **summed up to produce one output map**

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

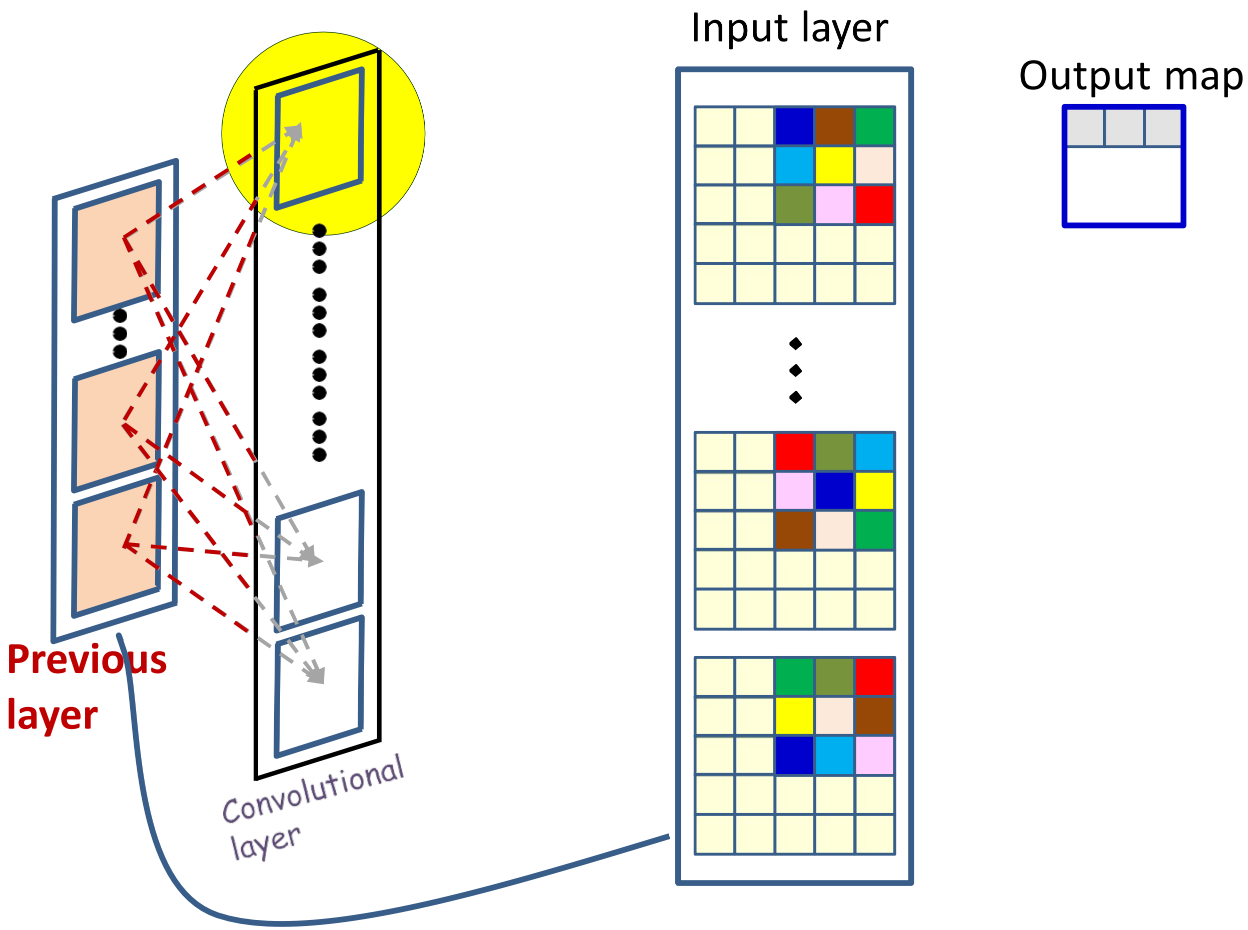
  *size of the filter* x *no. of maps in previous layer*



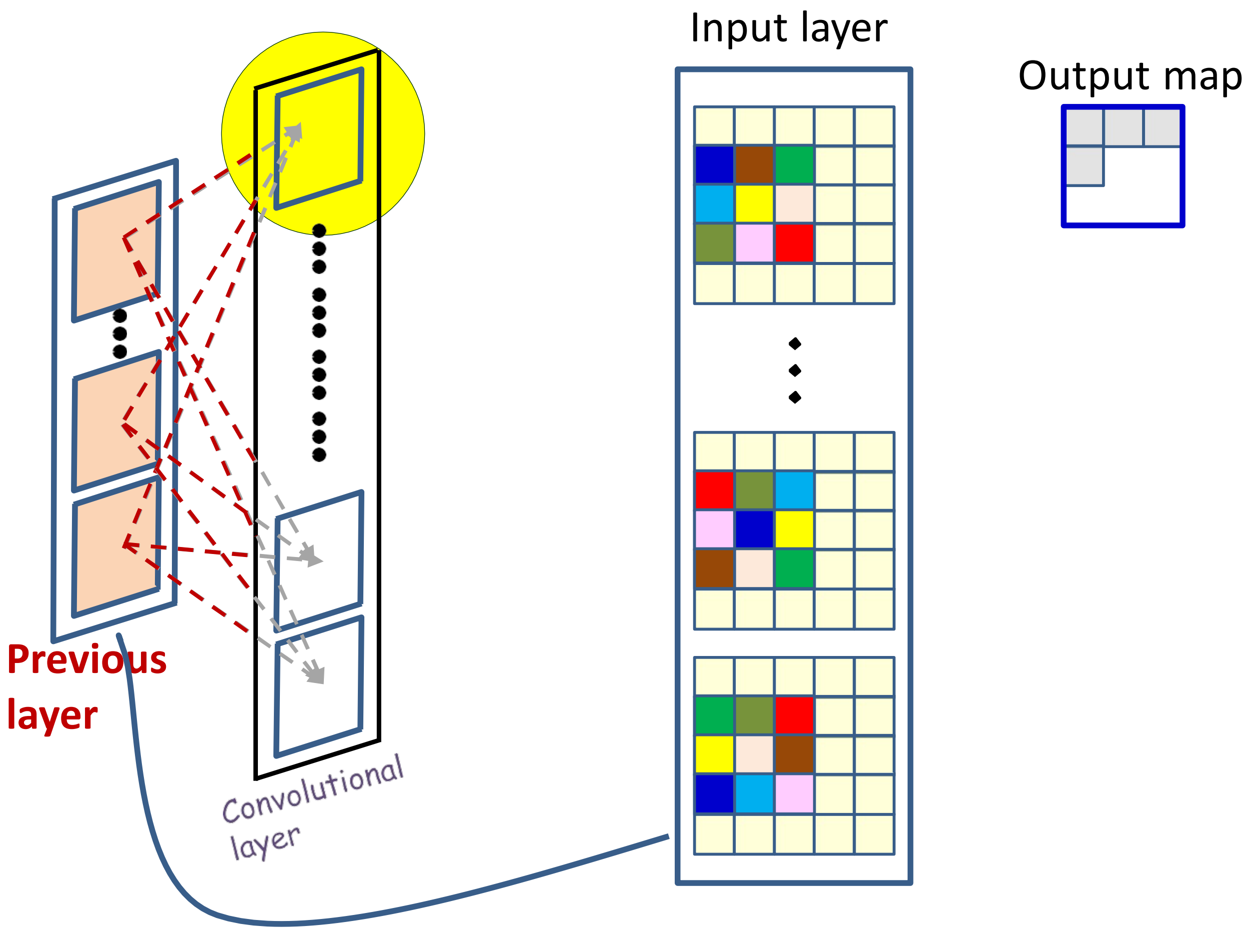

$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$



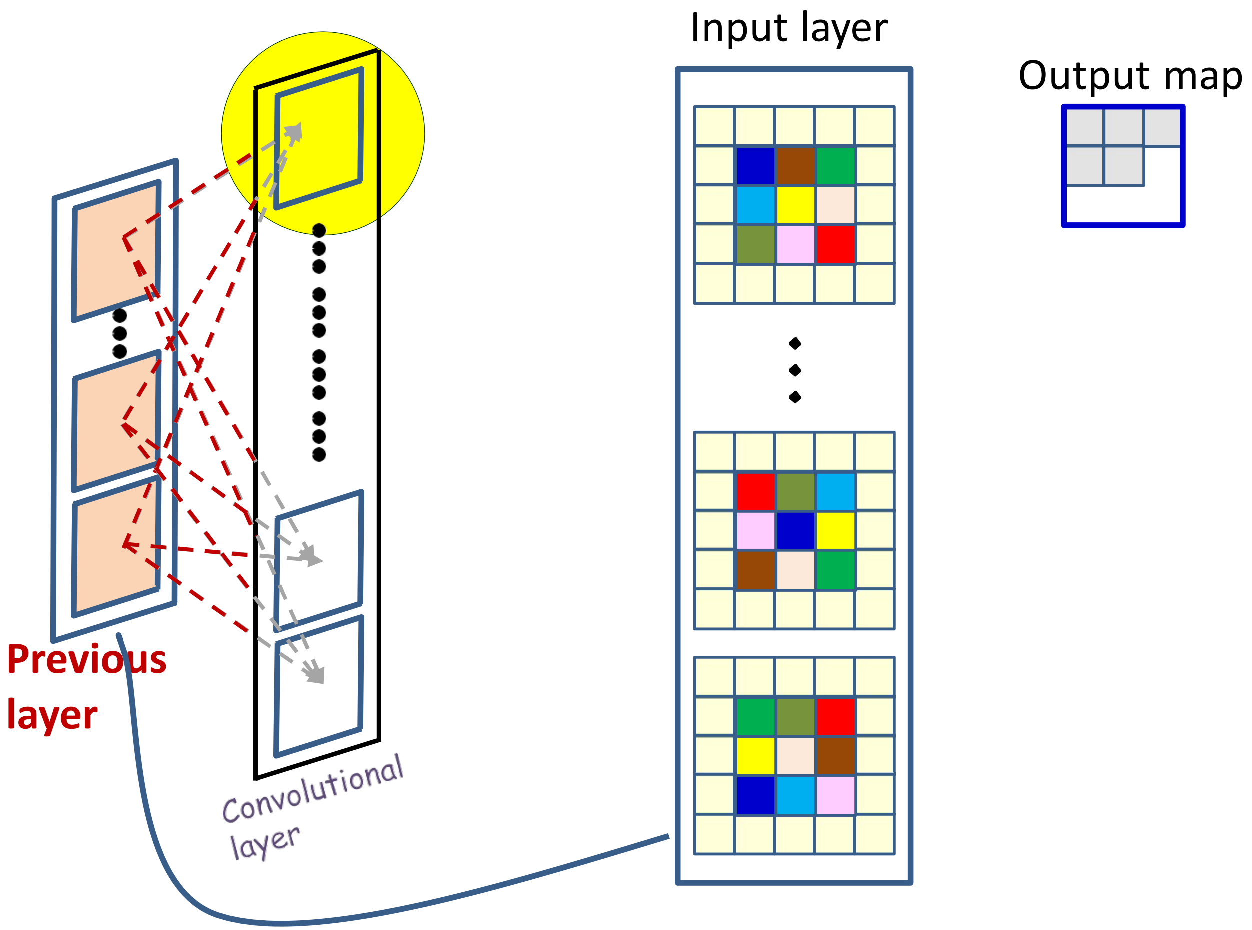

$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$



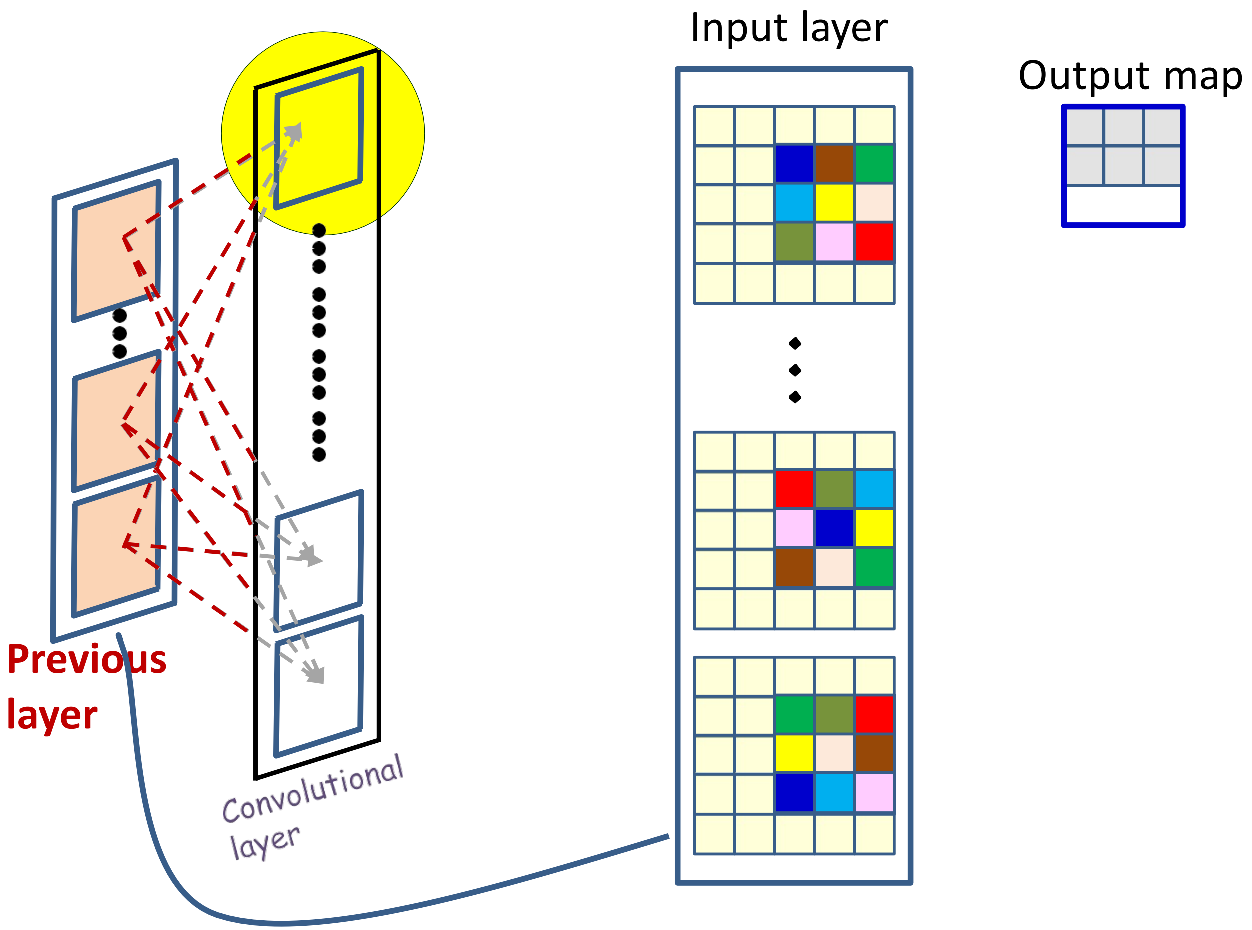

$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$



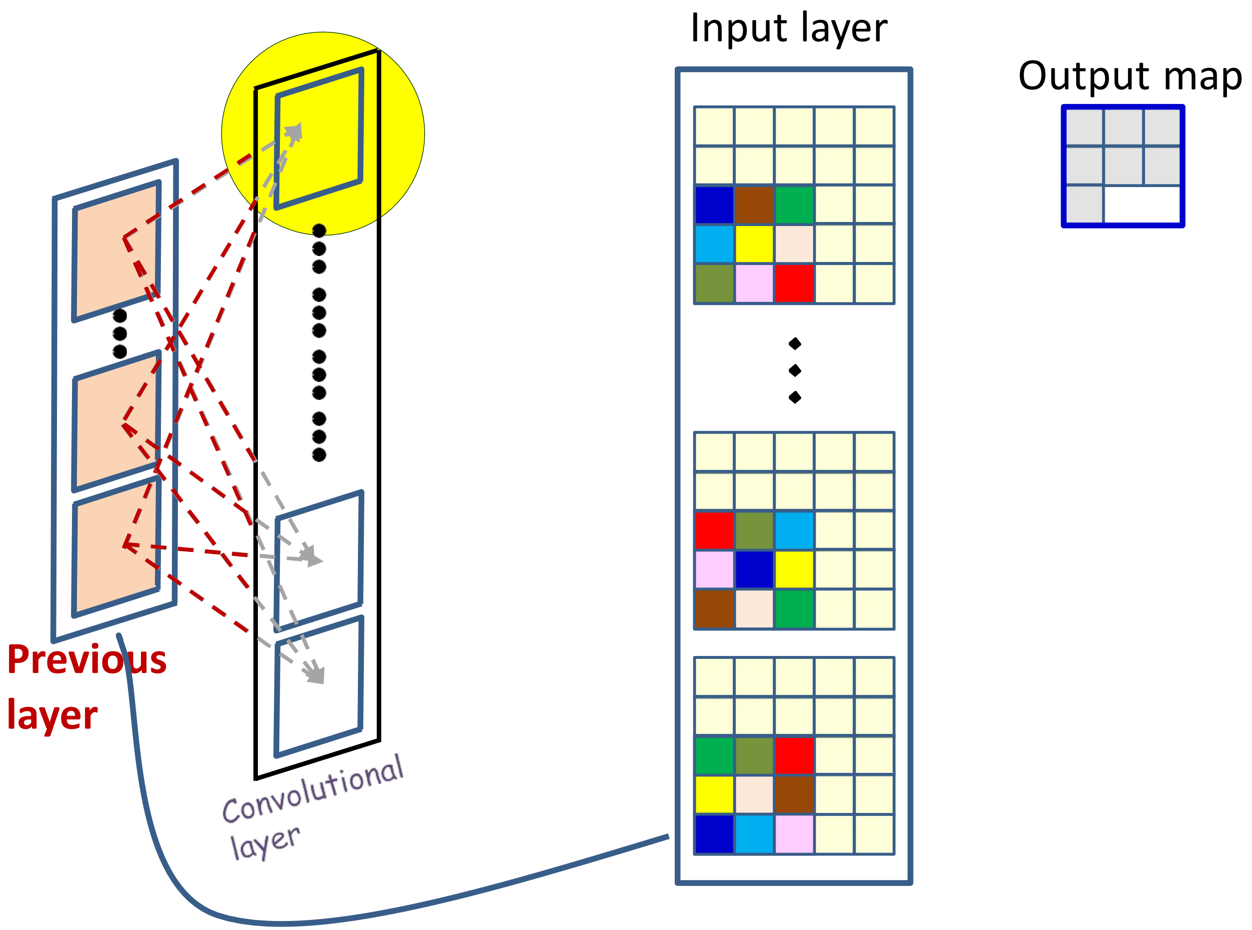

$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$



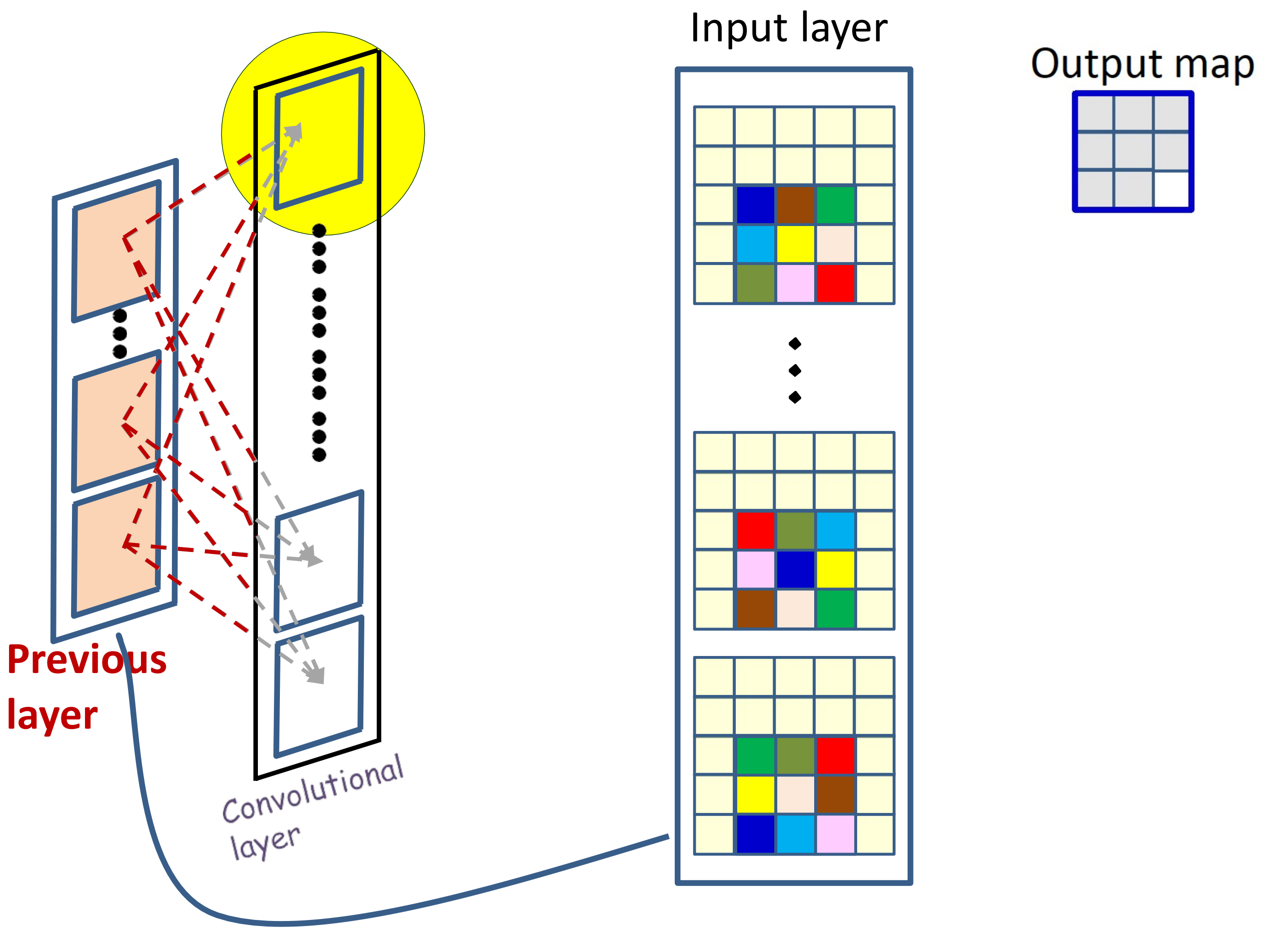

$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$



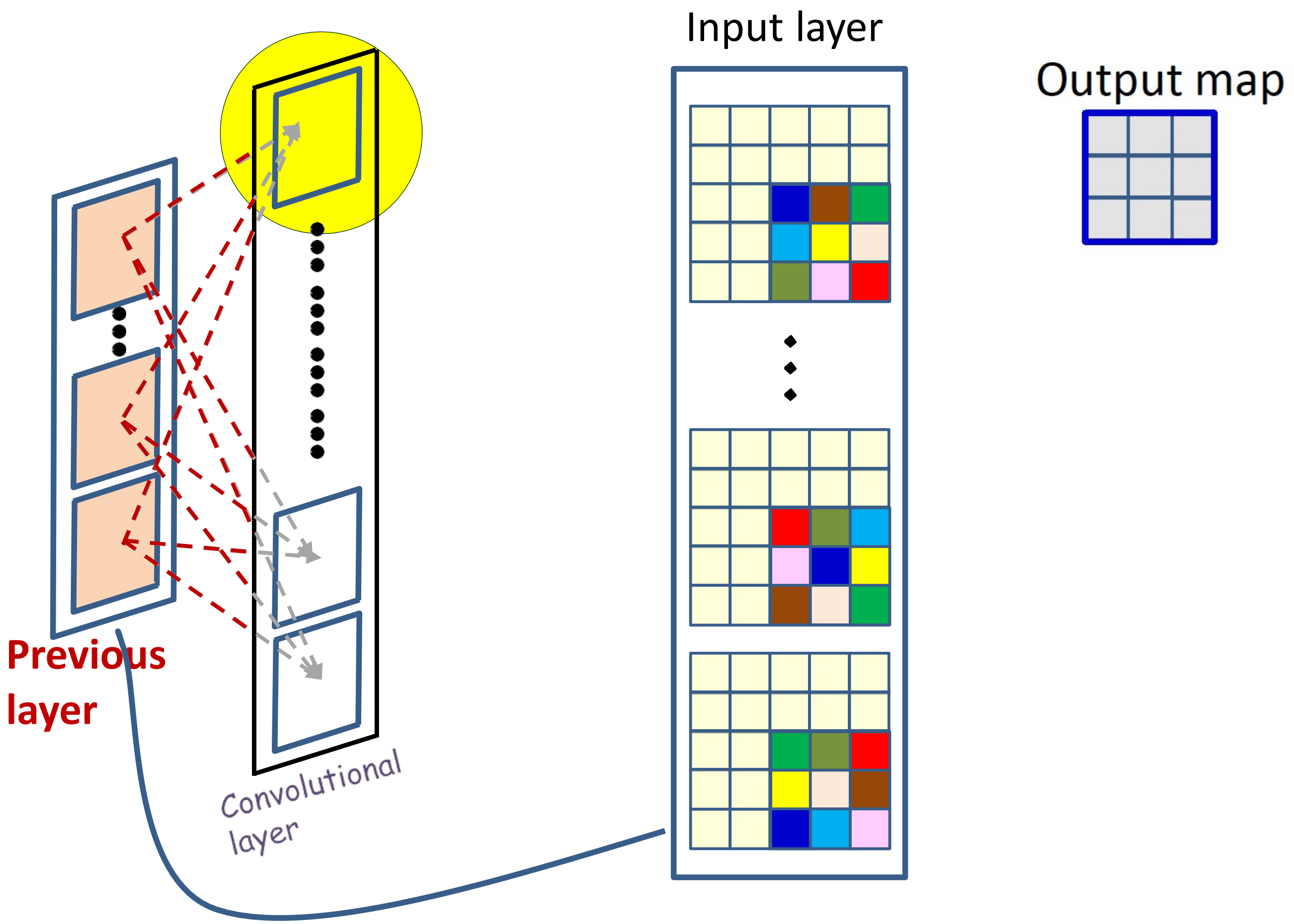

$$z(1,i,j)=\sum_{m}\sum_{k=1}^{3}\sum_{l=1}^{3} w(1,m,k,l)I(m,i+l-1,j+k-1)+b$$




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

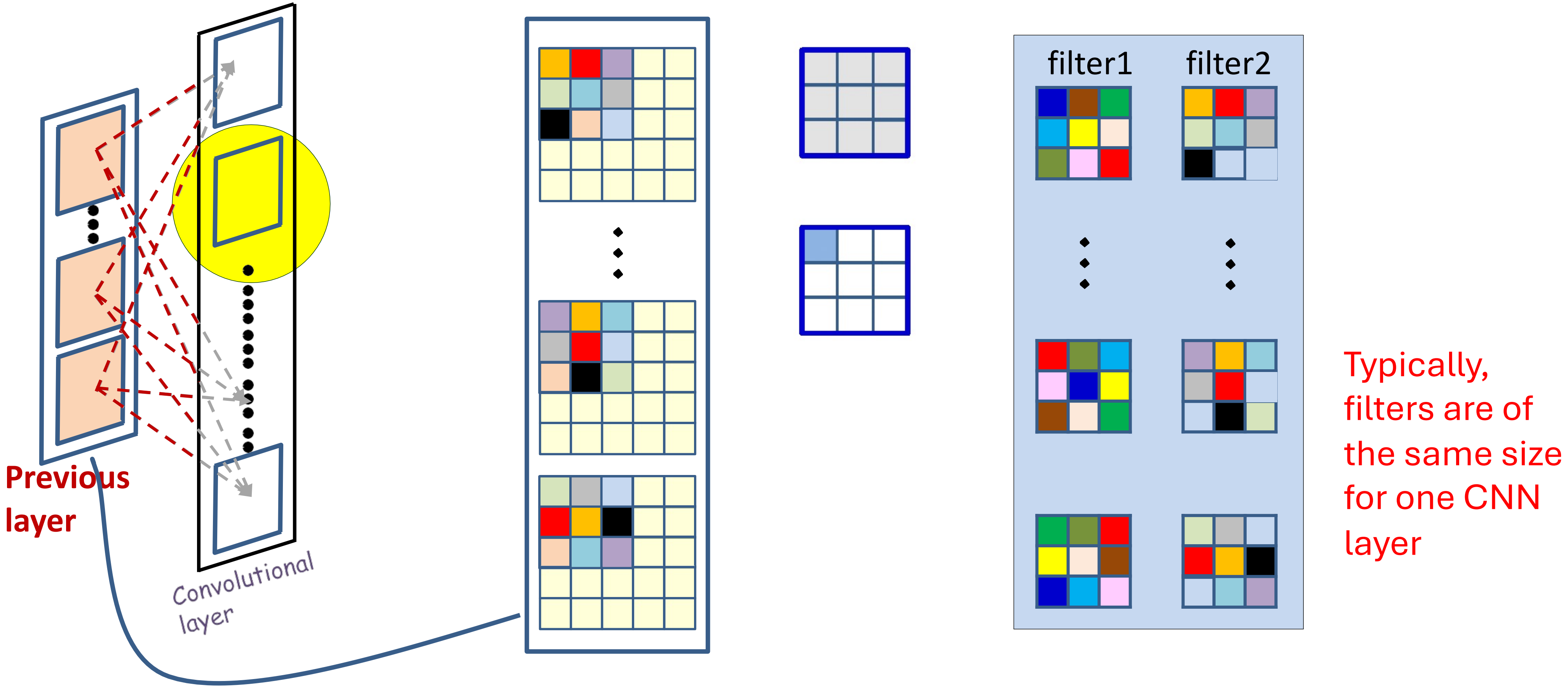



$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$




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

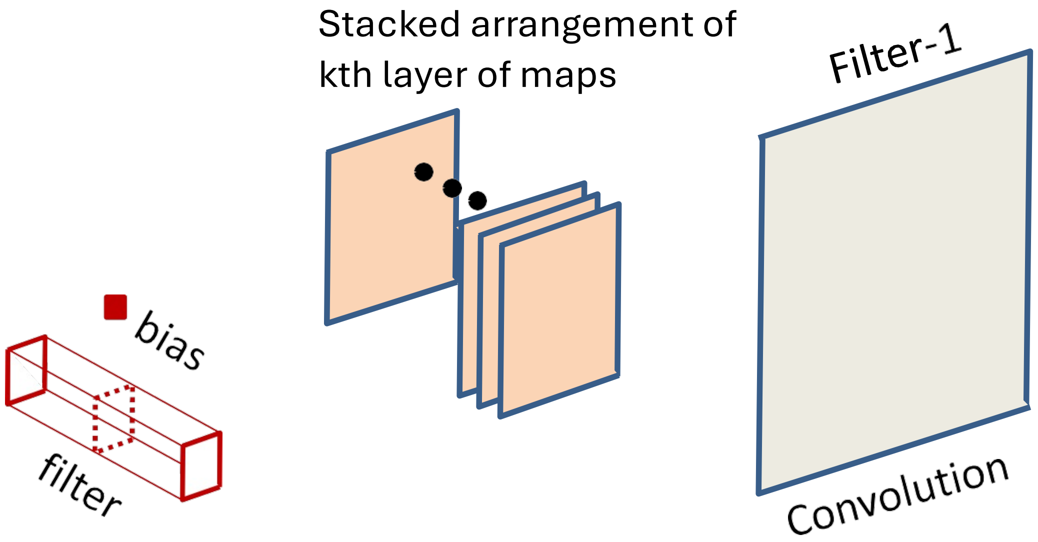
$$z(2,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(2,m,k,l) I(m, i+l-1, j+k-1) + b(2)$$




- 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*

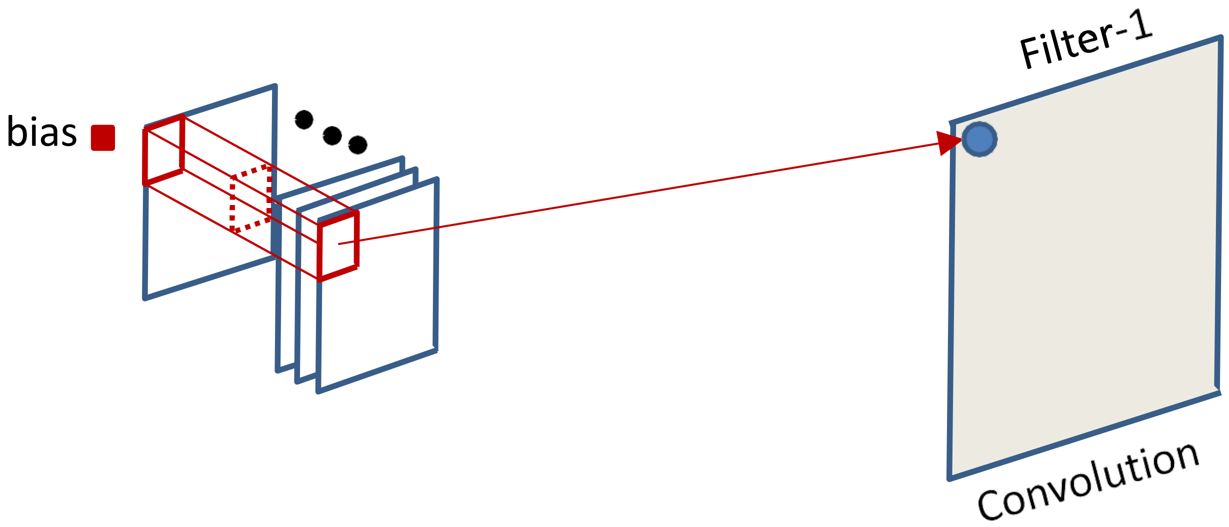
# A different view

Filter applied to kth layer of maps (convolutive component plus bias)




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

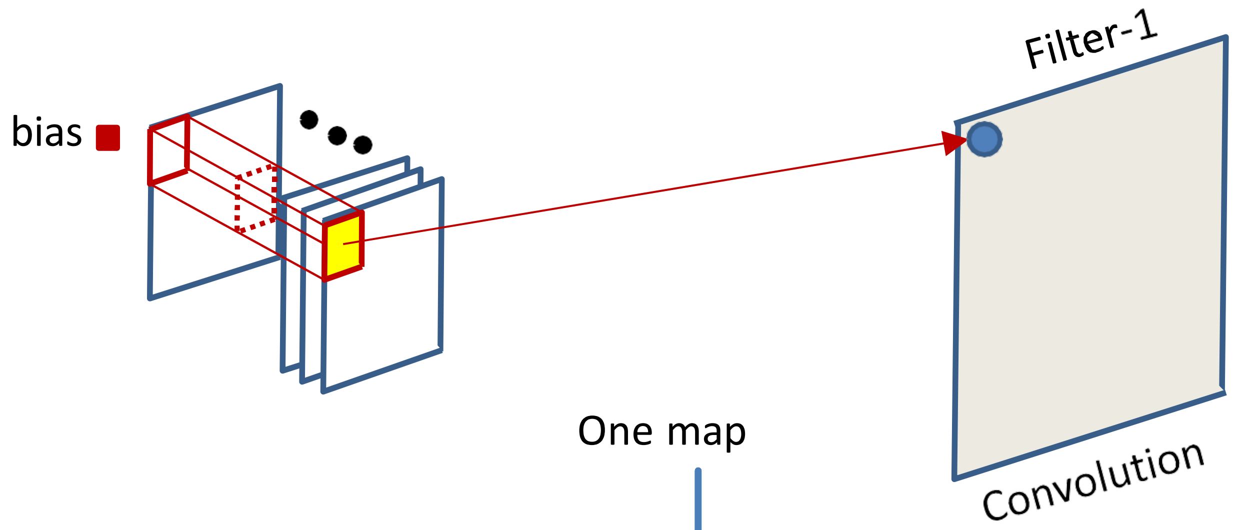
# The "cube" view of input maps




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

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

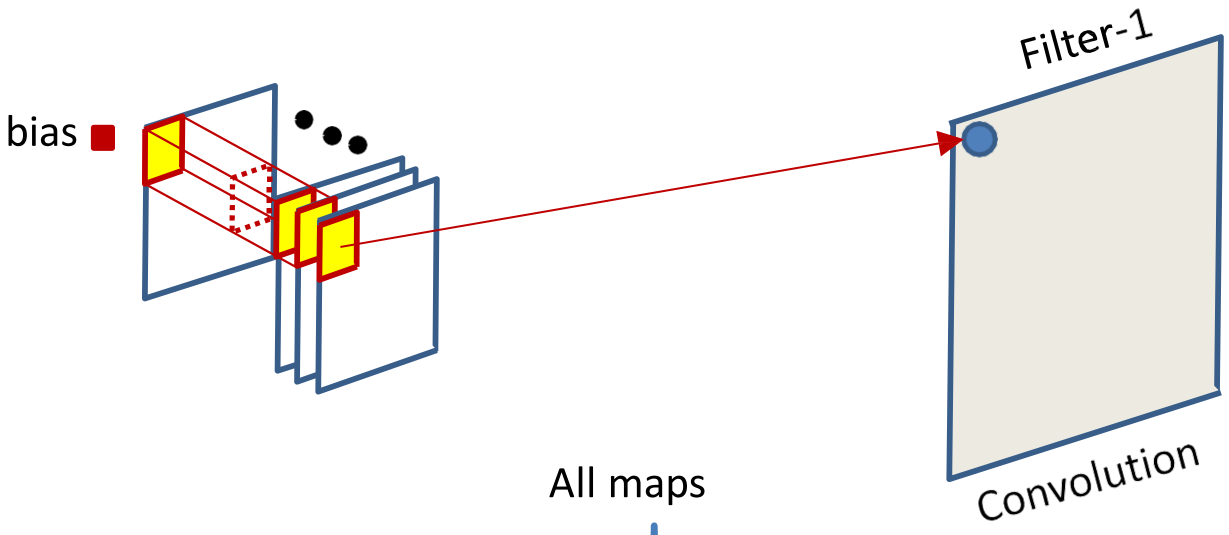
# The "cube" view of input maps




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

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

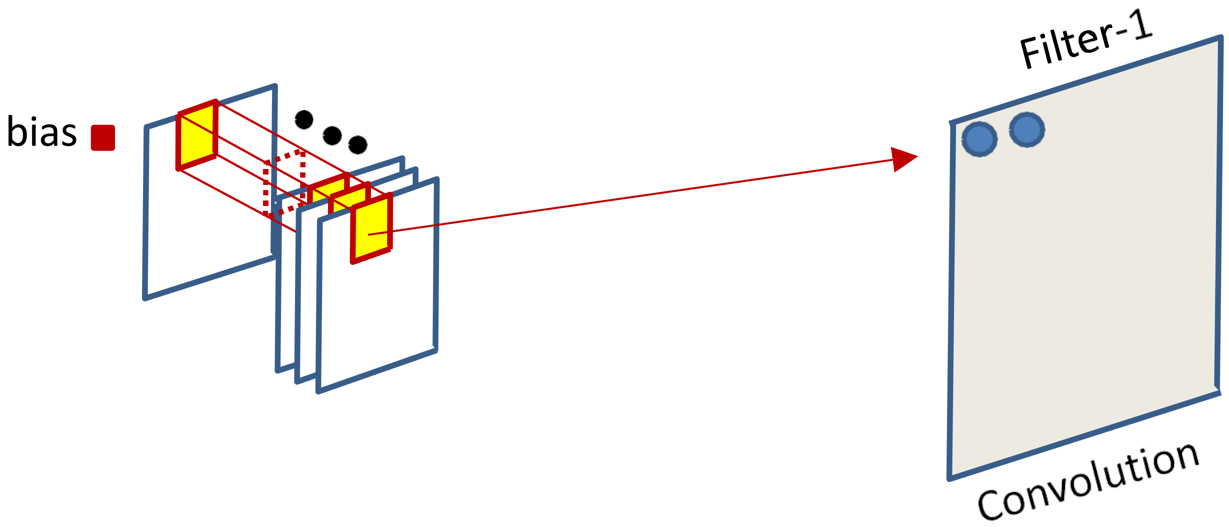
# The “cube” view of input maps




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

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

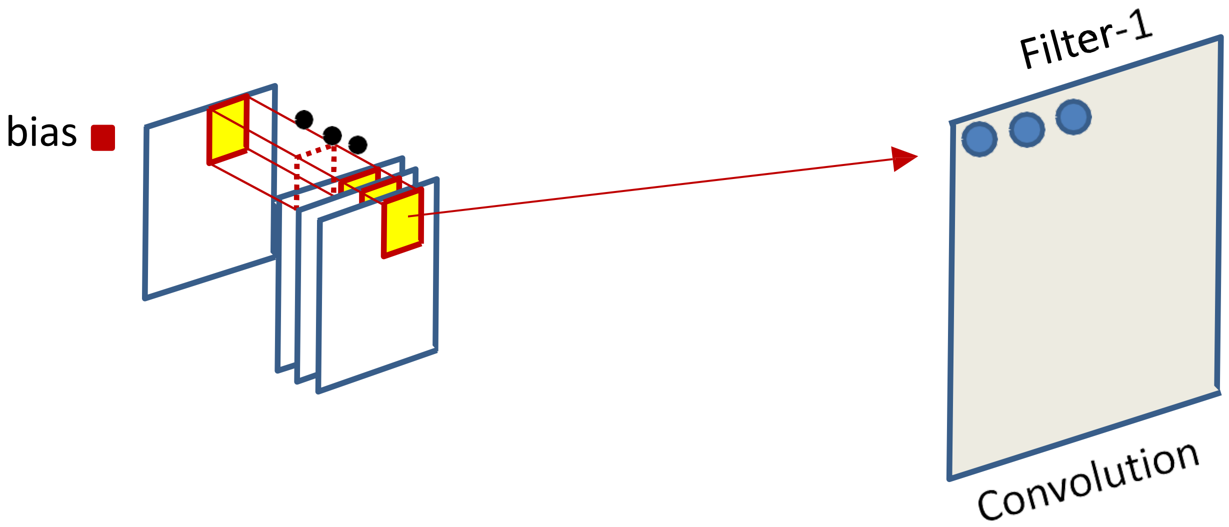
# The “cube” view of input maps




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

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

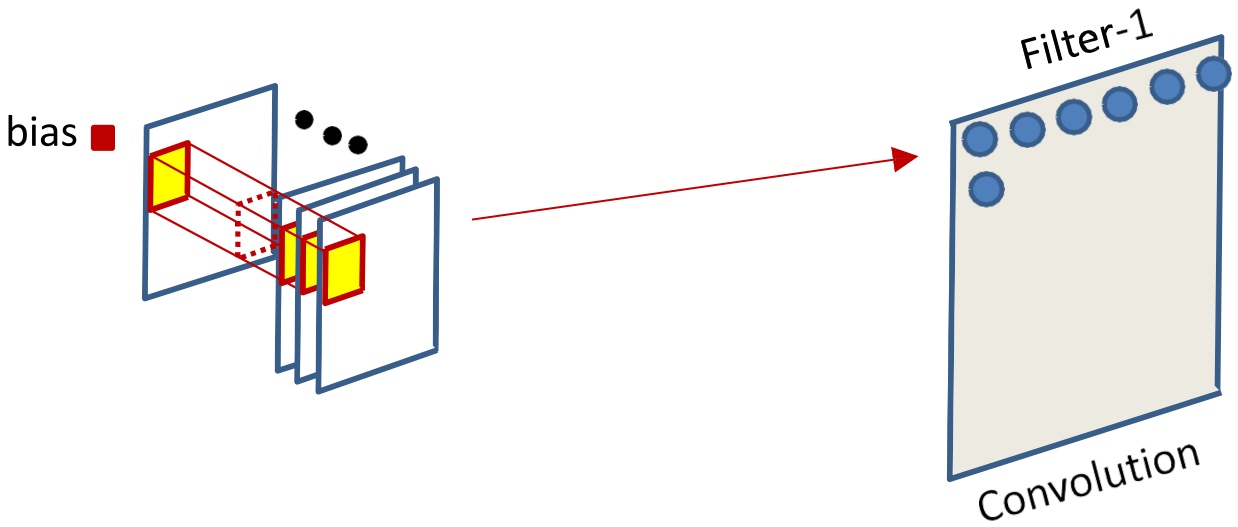
# The “cube” view of input maps




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

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

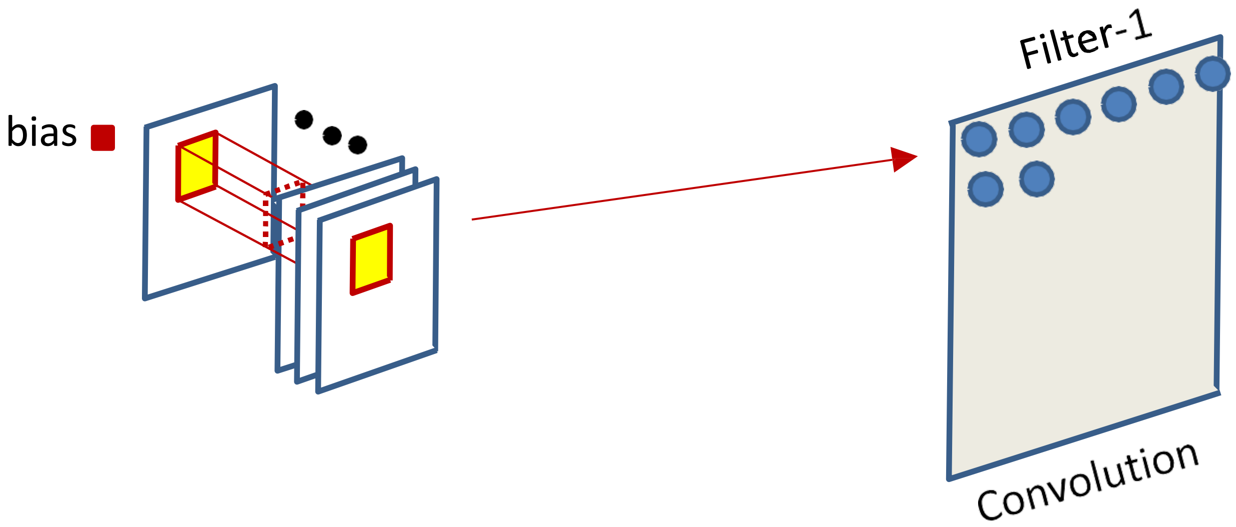
# The "cube" view of input maps




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

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

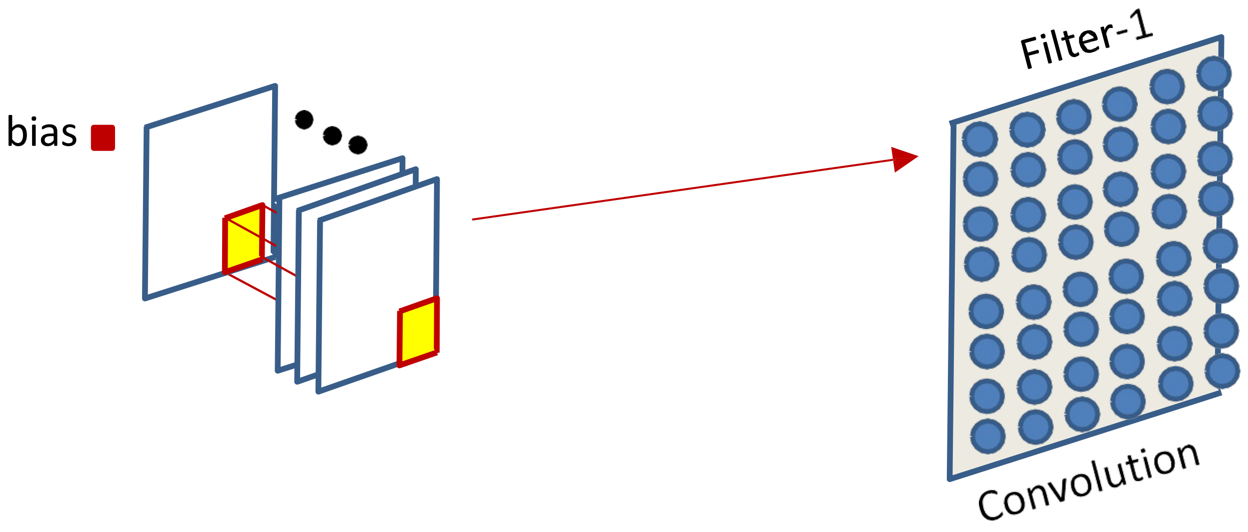
# The "cube" view of input maps




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

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

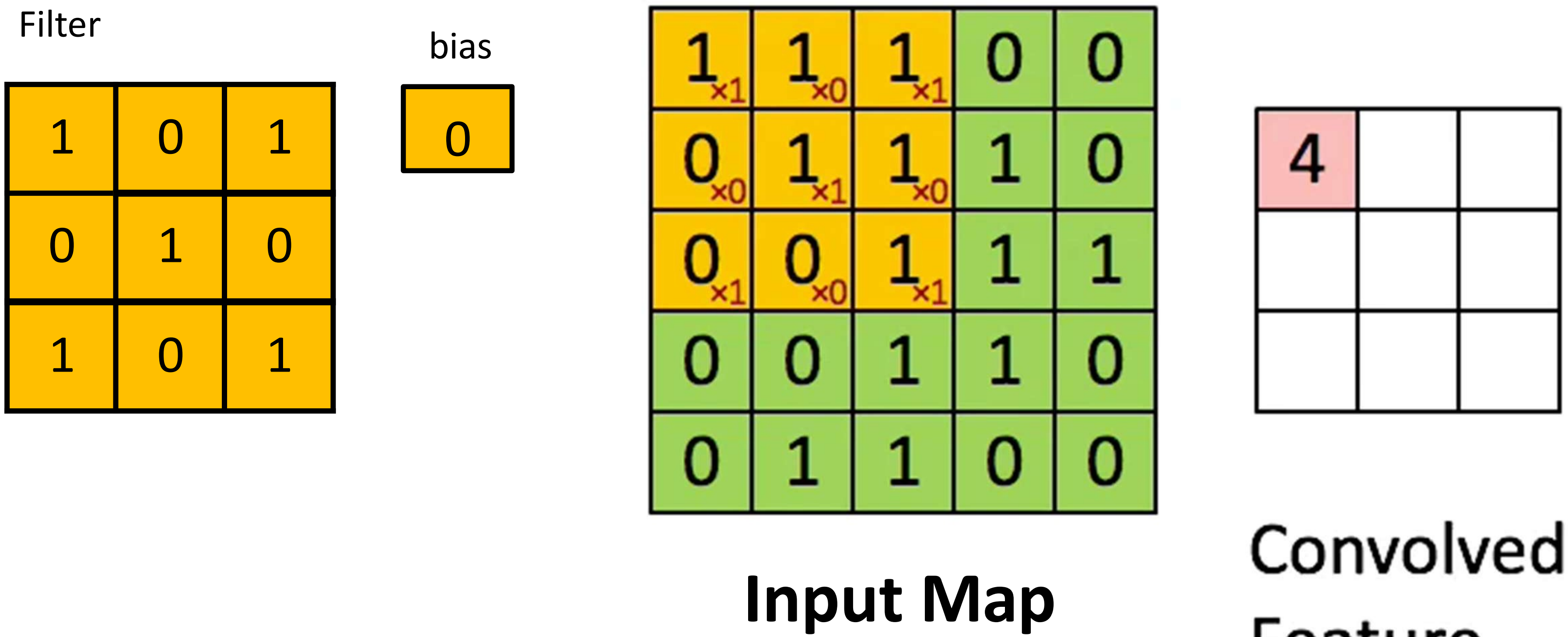
# The “cube” view of input maps




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

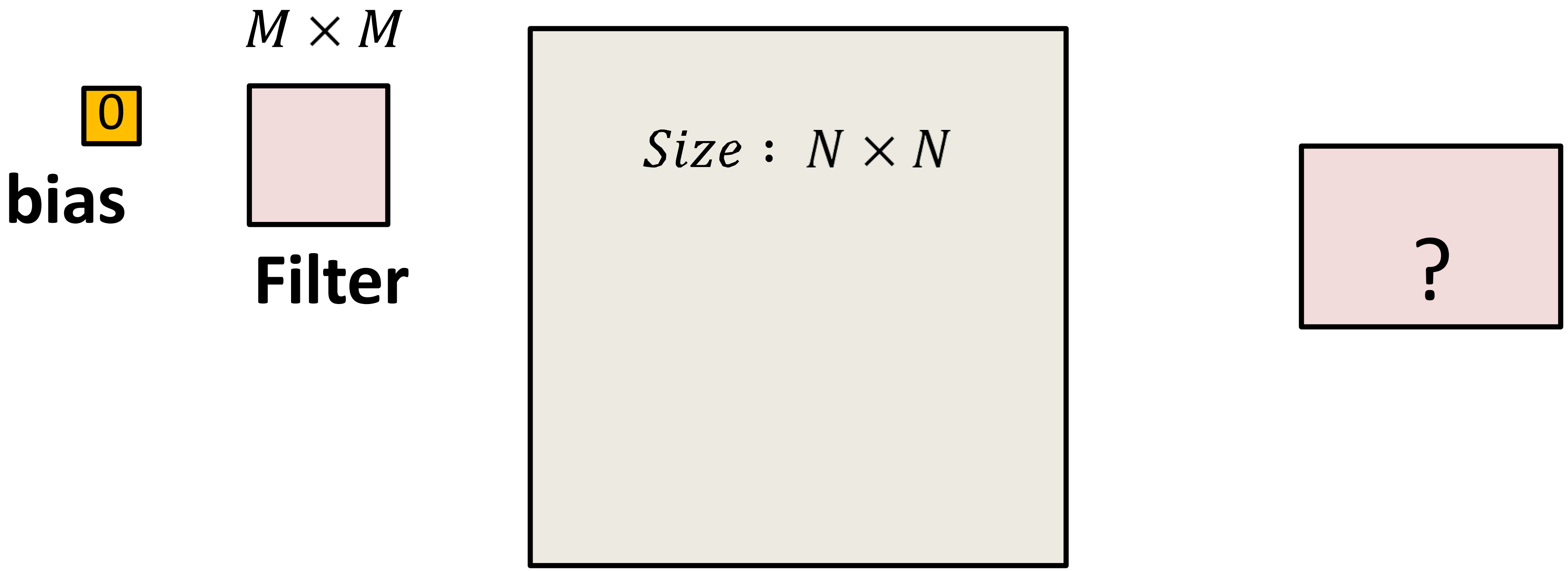
- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

# The size of the convolution




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

# The size of the convolution




- Image size: $N \times N$
- Filter: $M \times M$
- Output size = (N-M)+1 on each side

# Convolution Size

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

- Results in a reduction in the output size
  - Sometimes not considered acceptable

# Solution

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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

# Zero padding

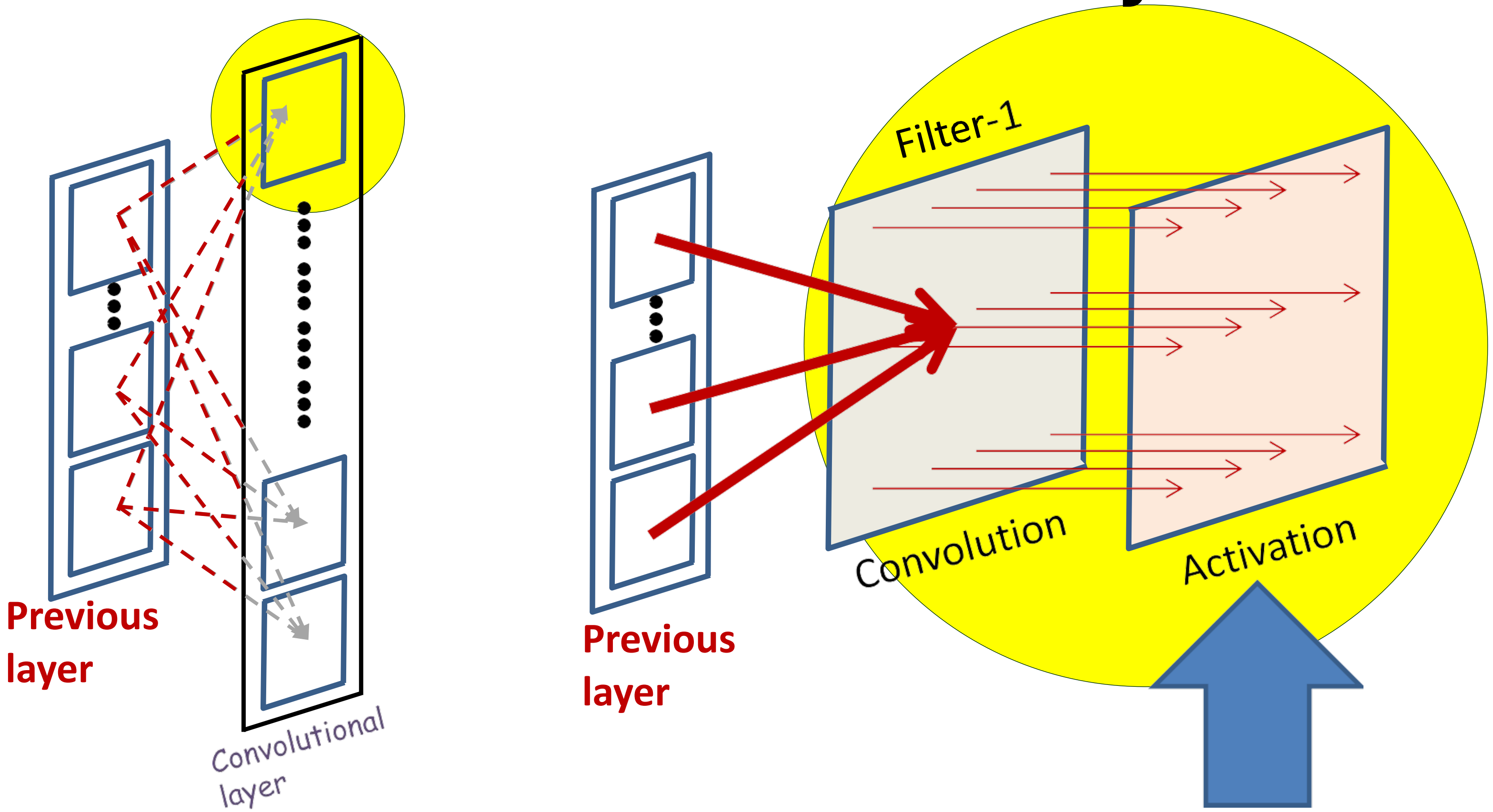
- For an $L$ width filter:
  - Odd $L$ : Pad on both left and right with $(L-1)/2$ columns of zeros
  - Even $L$ :  Pad one side with $L/2$ columns of zeros, and the other with $\frac{L}{2}-1$ columns of zeros
  - The resulting image is width $N+L-1$
  - The result of the convolution is width $N$

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

# Convolution Summary

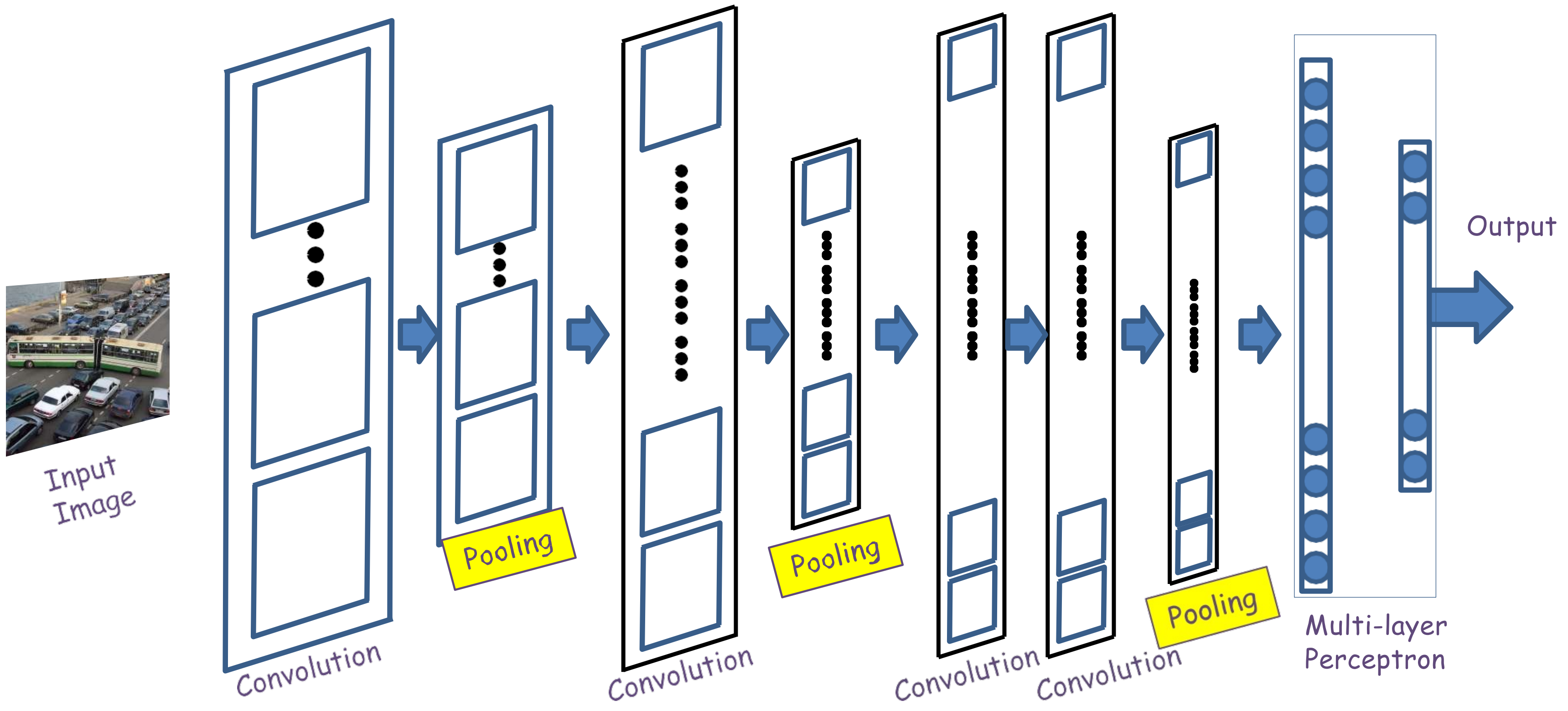
- Convolutional layers "scan" the input using a bank of "filters"
  – A "filter" is just a neuron in a scanning layer
- Each filter jointly scans the maps in the previous layer to produce an output "map"
  – As many output maps as ***filters*** (one output map per filter)
    - Regardless of the number of input maps
- Number of channels of a filter equals to the number of input maps
- We may have to pad the edges of the input maps to ensure that the output maps are the same size as input maps
  – If not, convolution loses rows/columns at the edges of the scan

# A convolutional layer



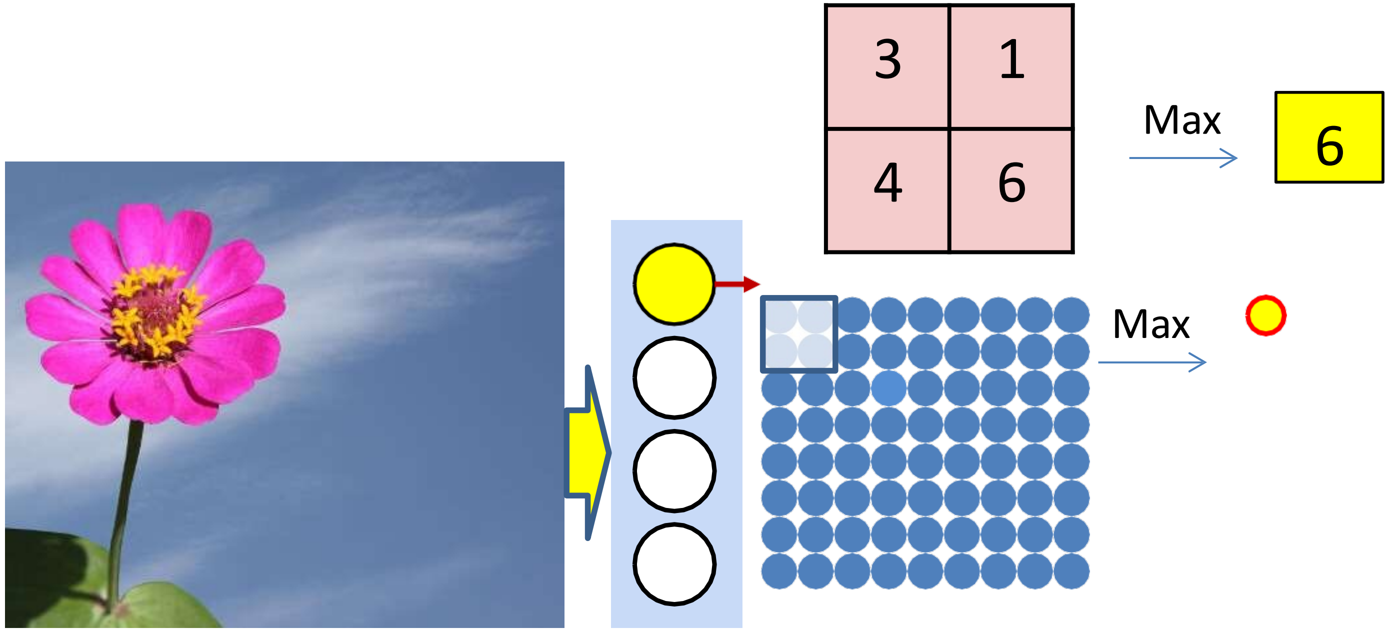

- The convolution operation results in an affine map
- An *Activation* is finally applied to every entry in the map

# The other component: Pooling



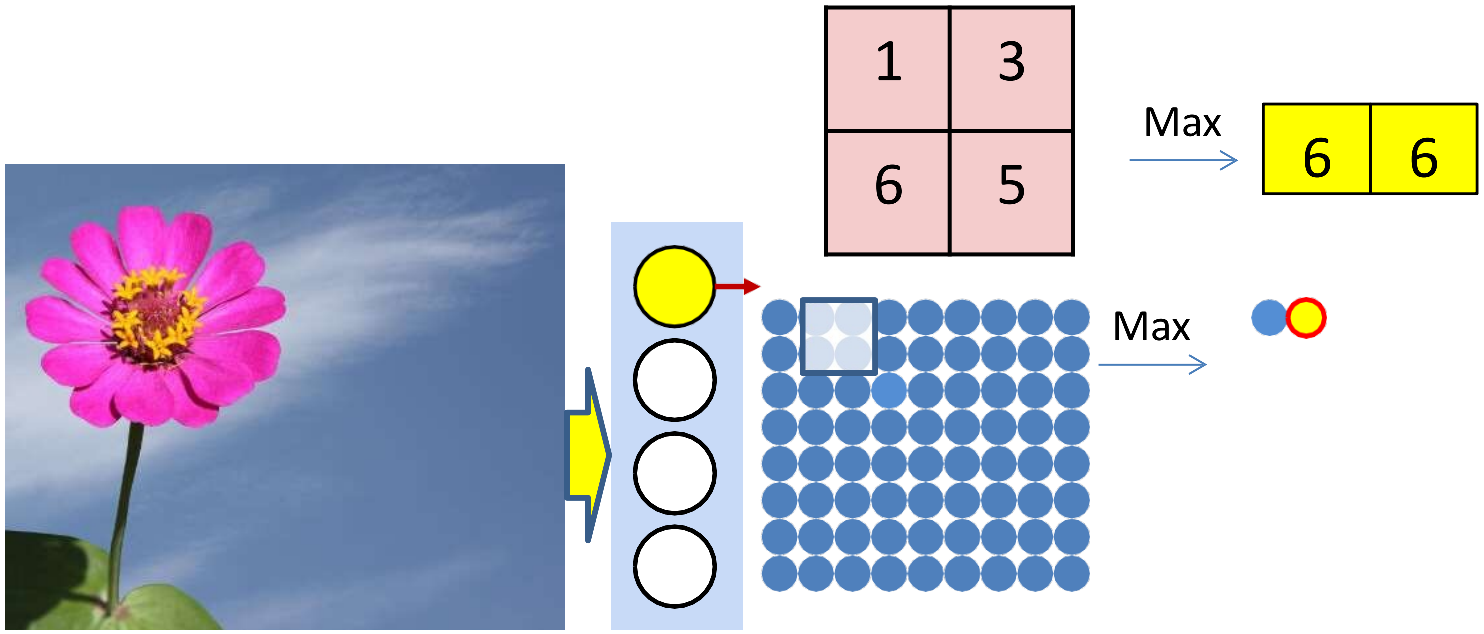

- Convolution (and activation) layers are followed intermittently by “pooling” layers
  - Typically (but not always) “max” pooling
  - Often, they alternate with convolution, though this is not necessary

# Max pooling

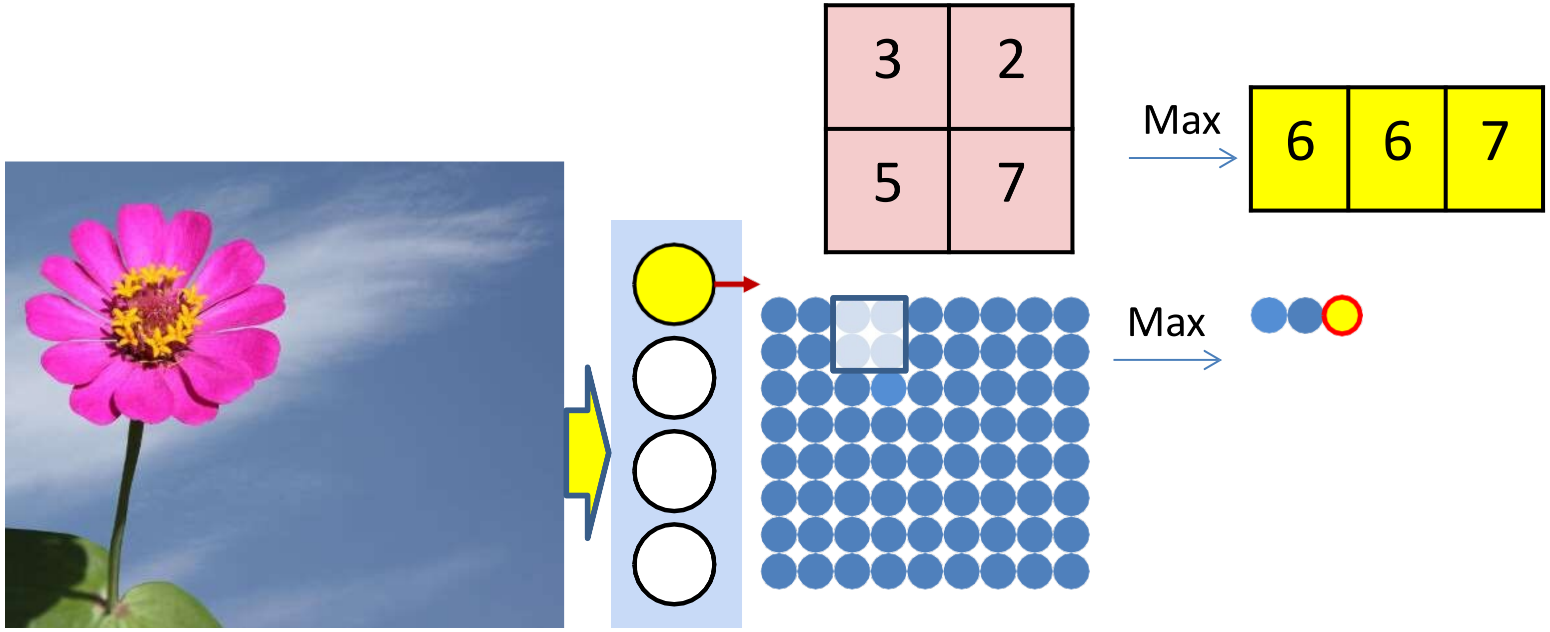



- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

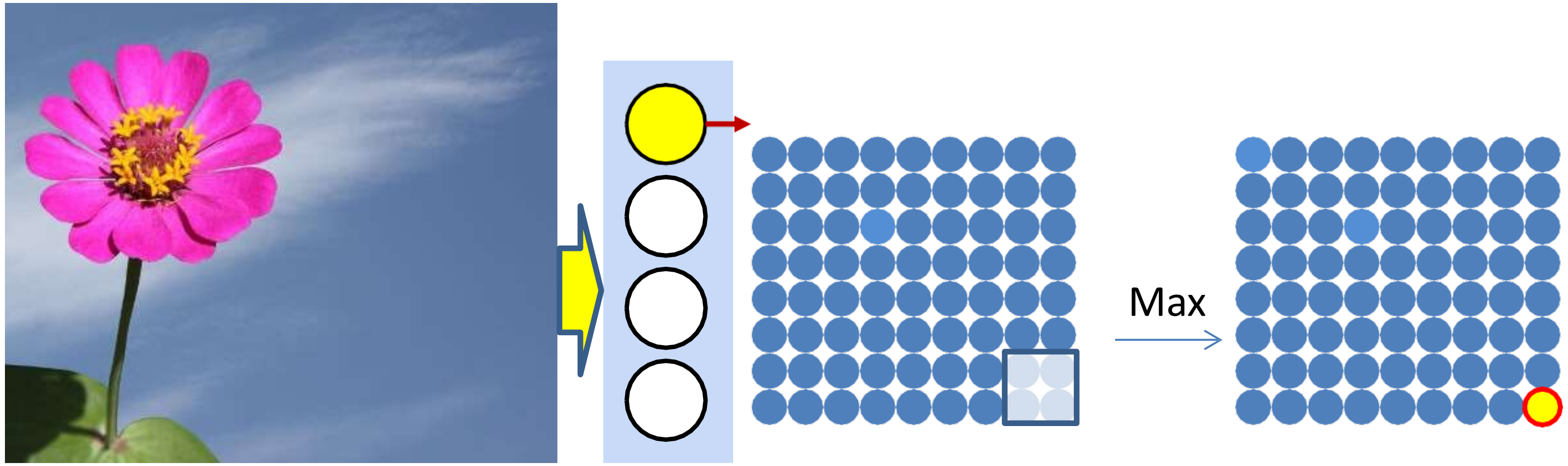
# Max pooling



# Max pooling

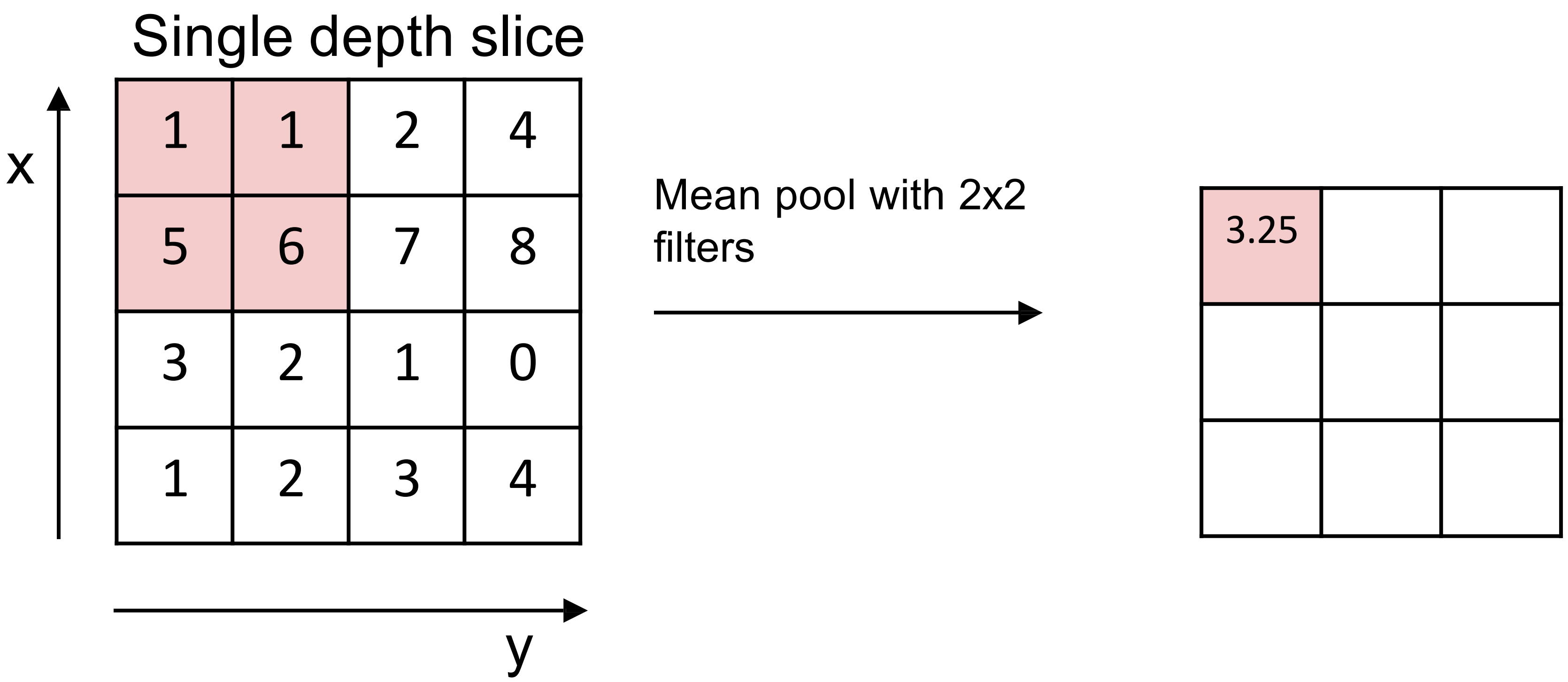


# Max pooling



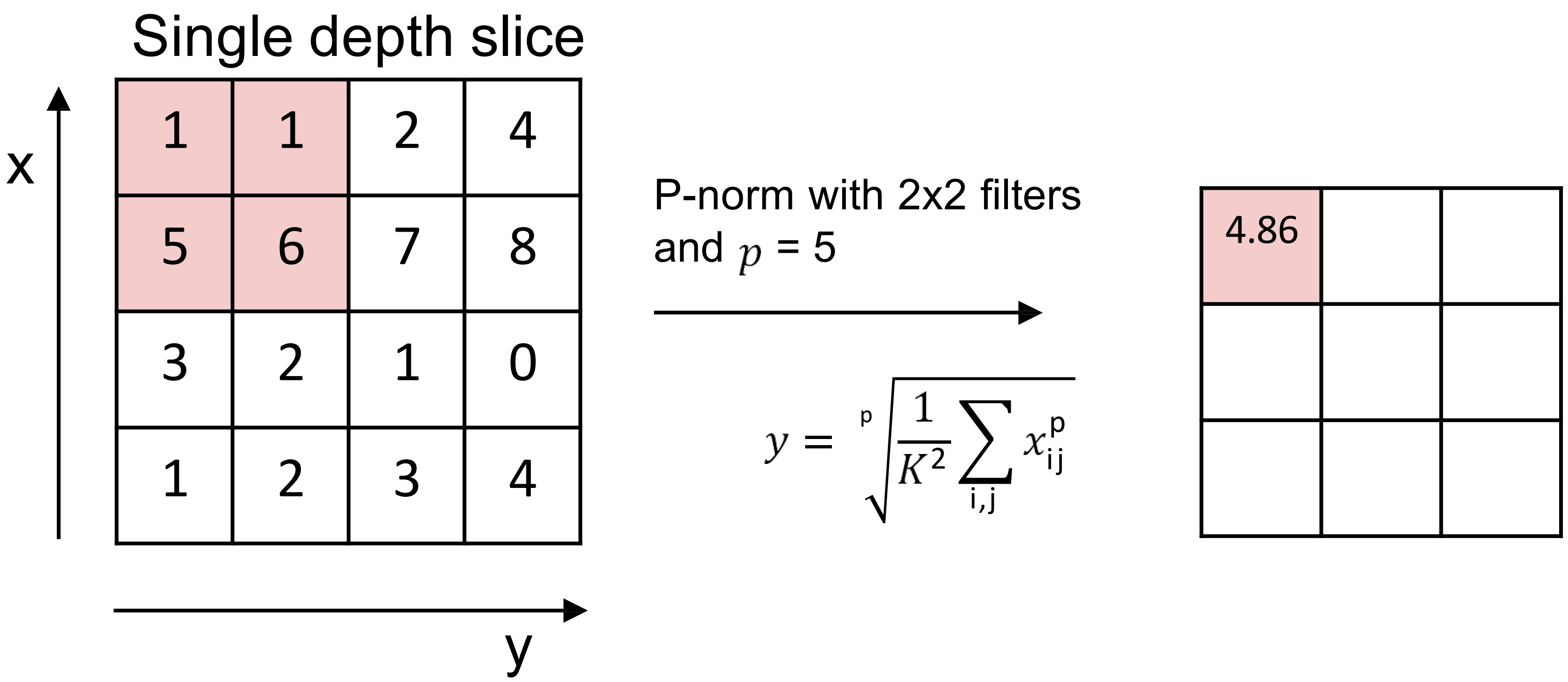

- Max pooling scans with a stride of 1 confer jitter- robustness
  - Typically performed with a stride > 1, whereupon it also results in “downsampling”

# Alternative to Max pooling: Mean Pooling




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

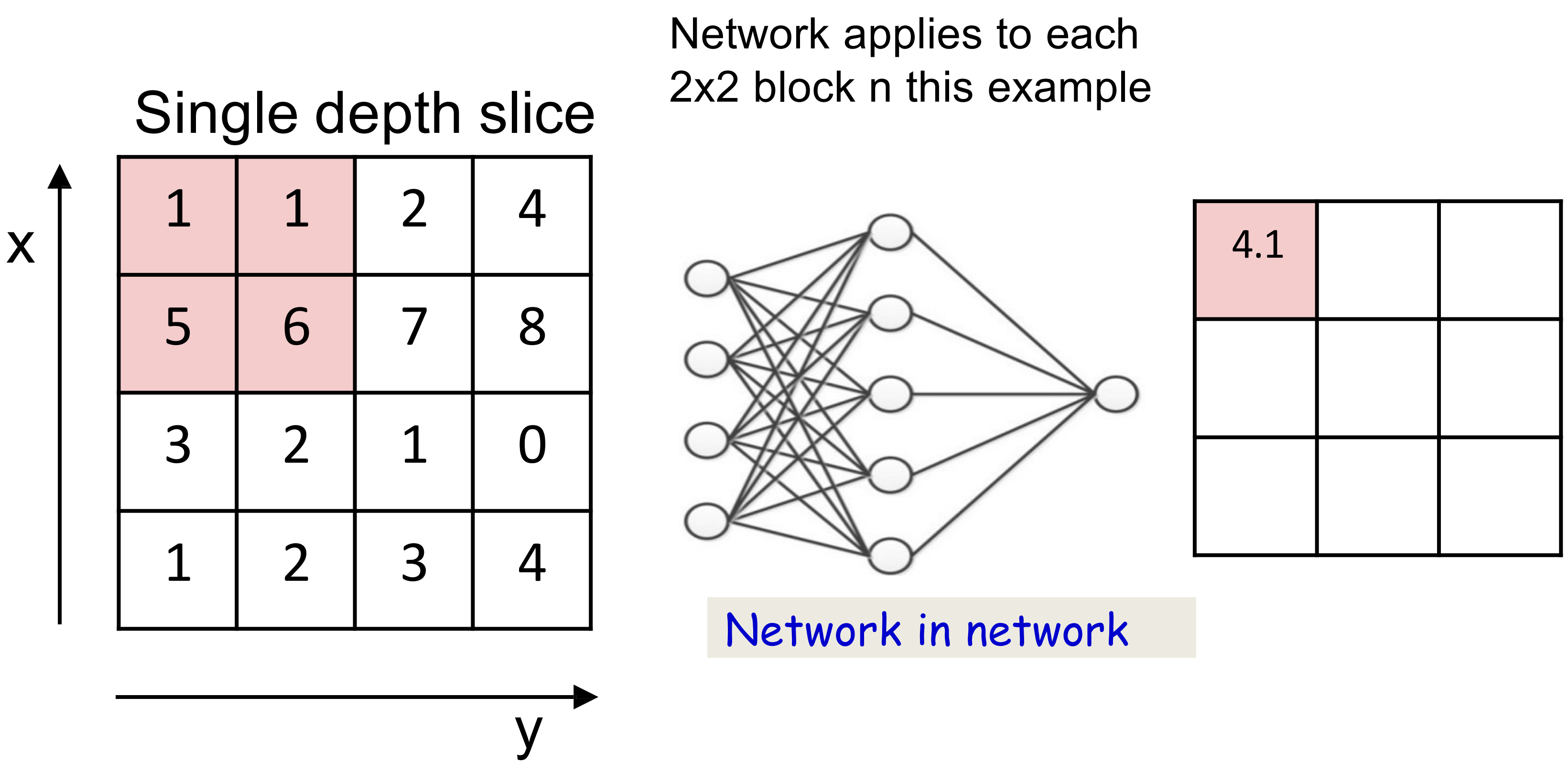
# Alternative to Max pooling: $\boldsymbol{p}$-norm




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

- Compute a p-norm of the pool

# Other options

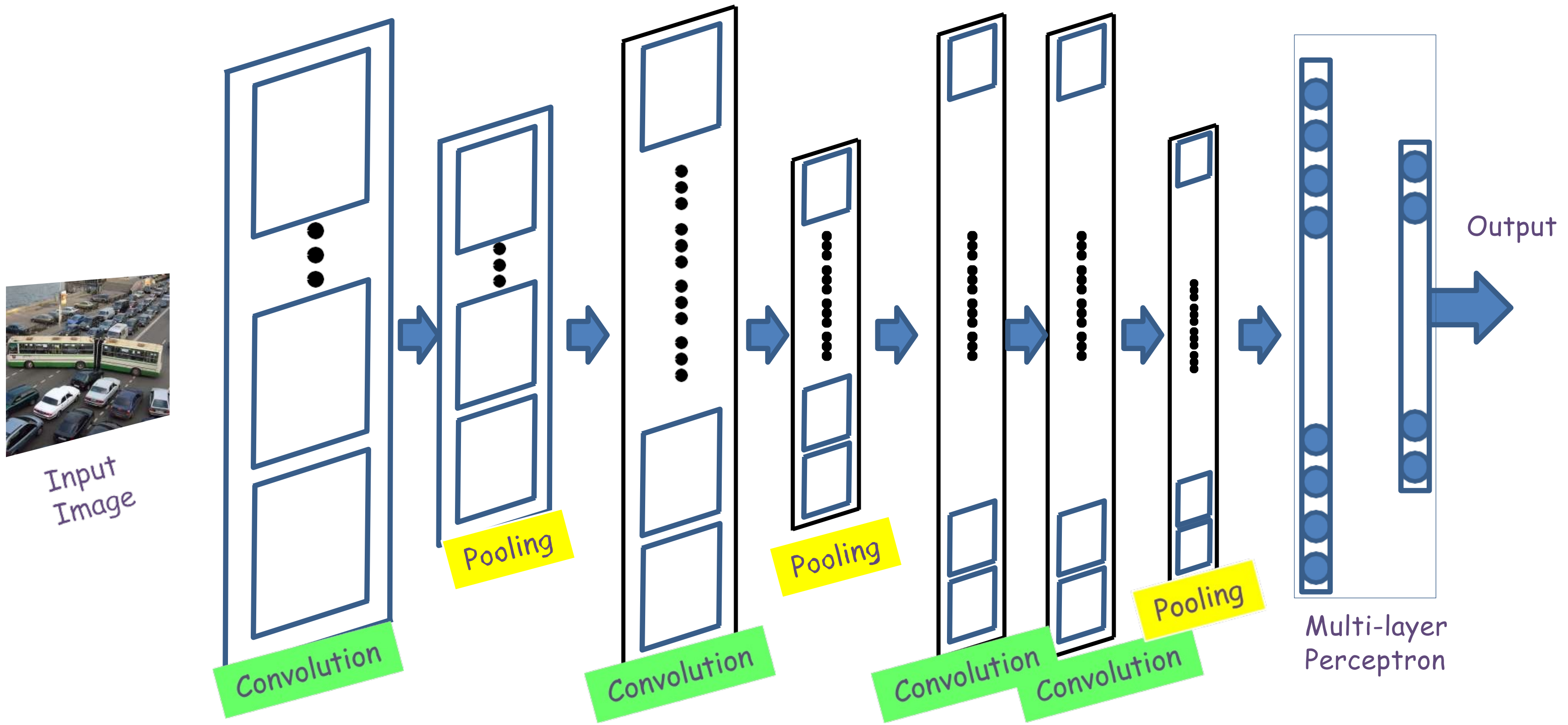



- The pooling may even be a *learned* filter
    - The *same* network is applied on each block
        - (Again, a shared parameter network)

# Pooling Summary

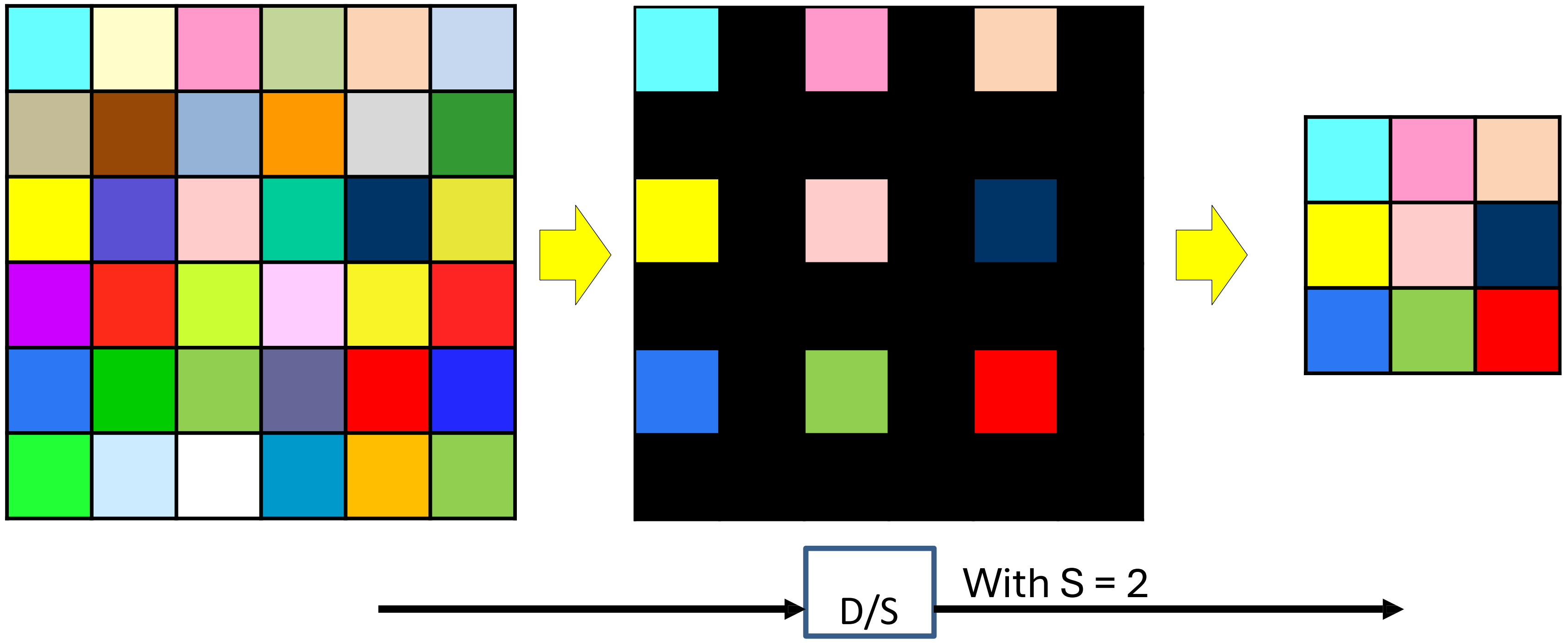
- Pooling layers “scan” the input using a “pooling” operation
  - E.g. selecting the max from a K x K block of input
- Each “pooling filter” scans an *individual* maps in the previous layer to produce an output “pooled map”
  - As many output maps as input maps
- For pooling we do not generally pad the edges
  - The zeros may result in incorrect pooled values, e.g. when all inputs are negative, and we apply max pooling

# The types of layers considered so far



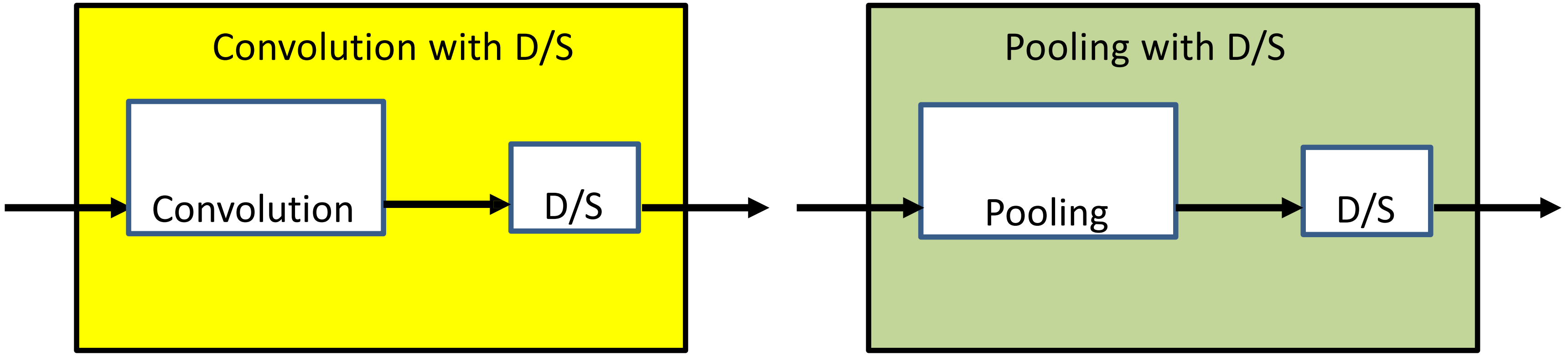

- So far we have only considered layers where the output size is approximately equal to input size
- There are two other operations that *change* the size of the output

# The Downsampling Layer



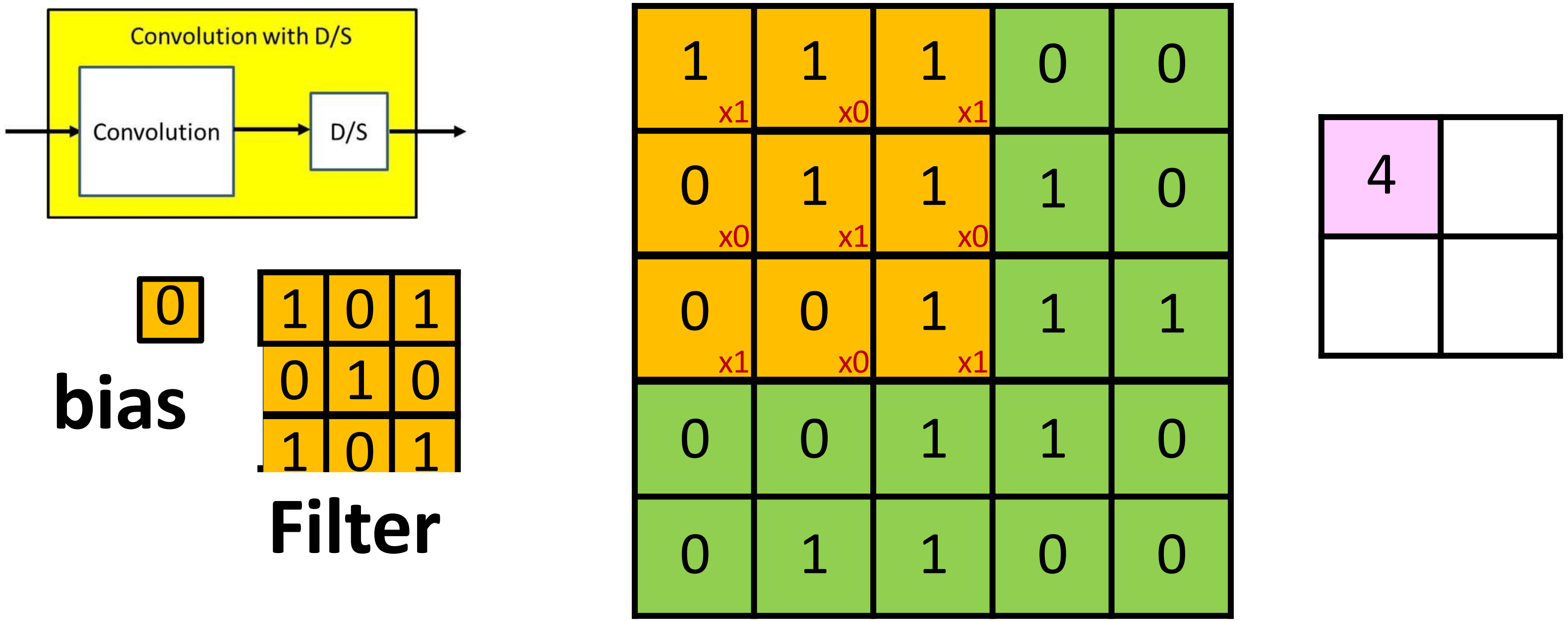

- A *downsampling* layer simply "drops" $S-1$ of $S$ rows and columns for every map in the layer
  - Effectively reducing the size of the map by factor S in every direction

# Downsampling in practice



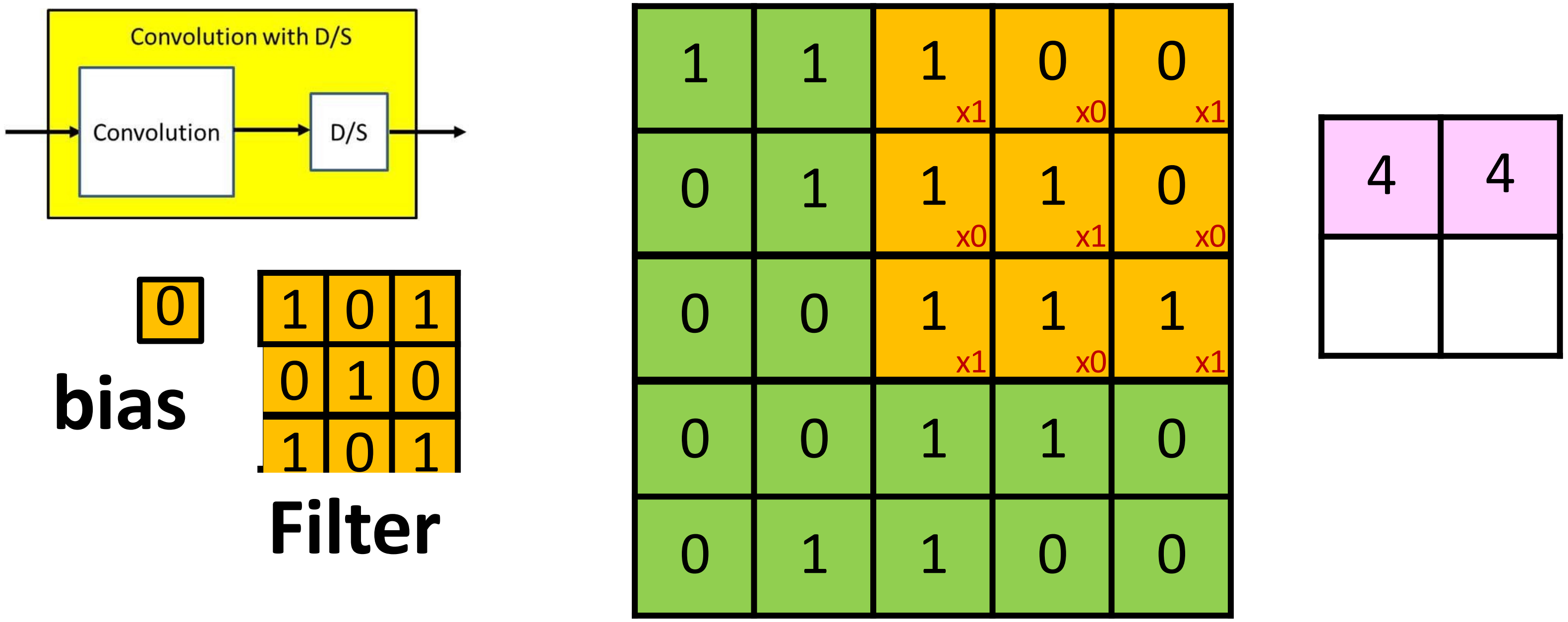

- In practice, the downsampling is combined with the layers just before it
  - Which could be convolutional or pooling layers

# Downsampling after Convolution



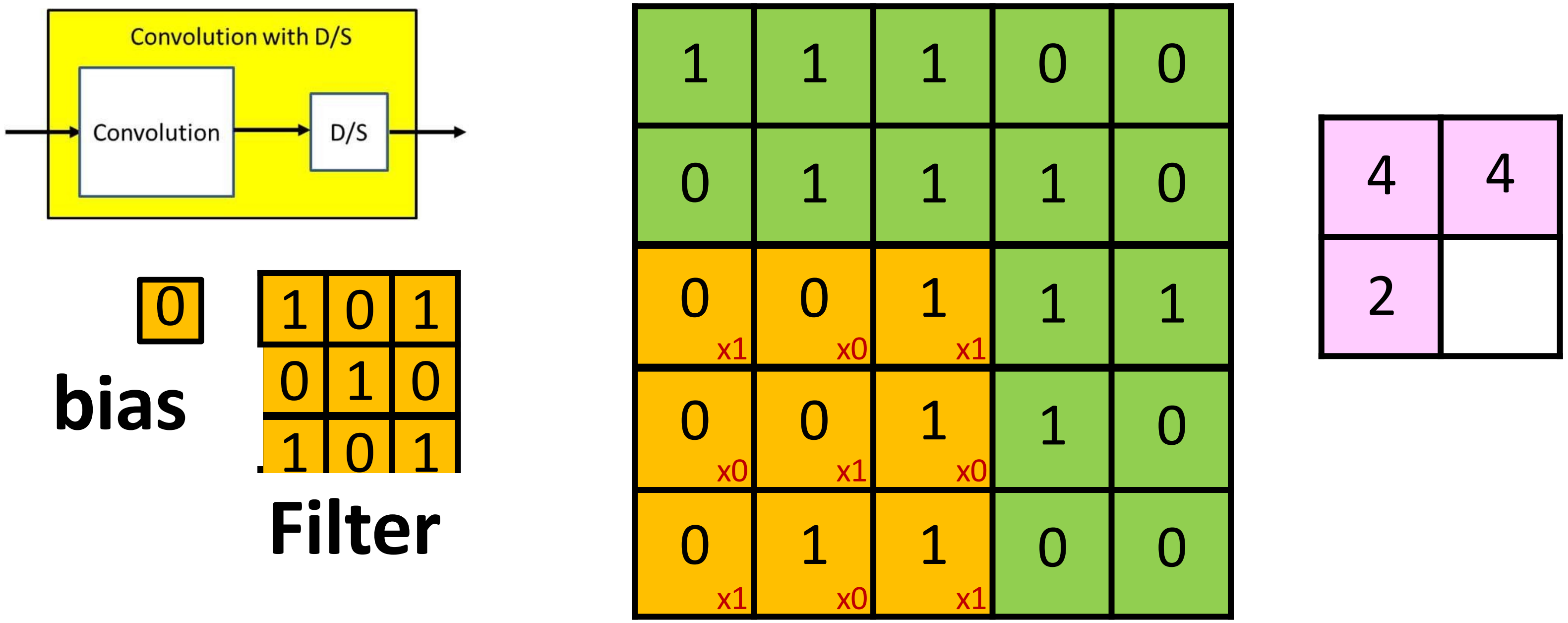

A downsampling layer can be combined with a convolutional layer into a single convolutional layer with ***convolution stride S***

# Downsampling after Convolution



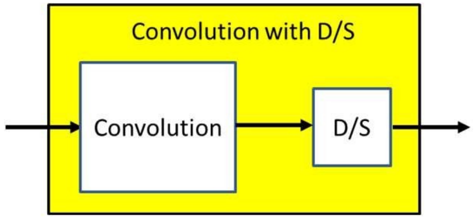

A downsampling layer can be combined with a convolutional layer into a single convolutional layer with ***convolution stride S***

# Downsampling after Convolution




A downsampling layer can be combined with a convolutional layer into a single convolutional layer with ***convolution stride S***

# Downsampling after Convolution




**bias**

| |
|---|
| 0 |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

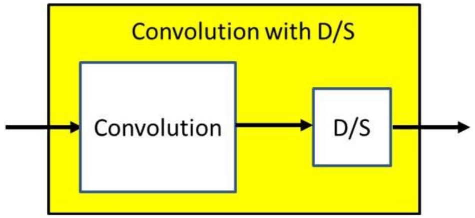
**Filter**

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 x1 | 1 x0 | 1 x1 |
| 0 | 0 | 1 x0 | 1 x1 | 0 x0 |
| 0 | 1 | 1 x1 | 0 x0 | 0 x1 |

| | |
|---|---|
| 4 | 4 |
| 2 | 4 |

A downsampling layer can be combined with a convolutional layer into a single convolutional layer with ***convolution stride S***

# Downsampling after Convolution




| |
|---|
| 0 |

**bias**

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

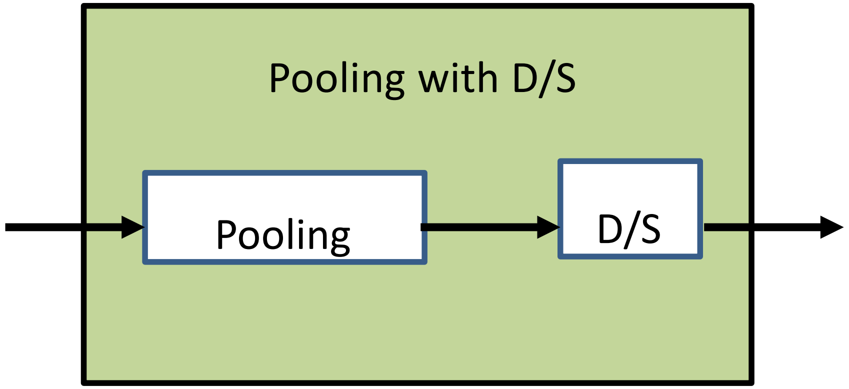
**Filter**

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 x1 | 1 x0 | 1 x1 |
| 0 | 0 | 1 x0 | 1 x1 | 0 x0 |
| 0 | 1 | 1 x1 | 0 x0 | 0 x1 |

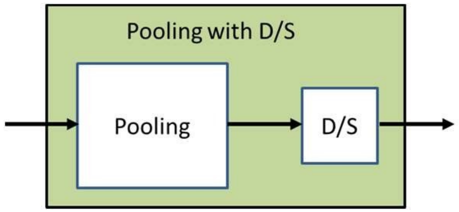
| | |
|---|---|
| 4 | 4 |
| 2 | 4 |

For an input of size $N \times N$ and filters of size $M \times M$ and stride $S$, the output size will be $\left\lfloor \frac{N-M}{S} \right\rfloor + 1$ on every side

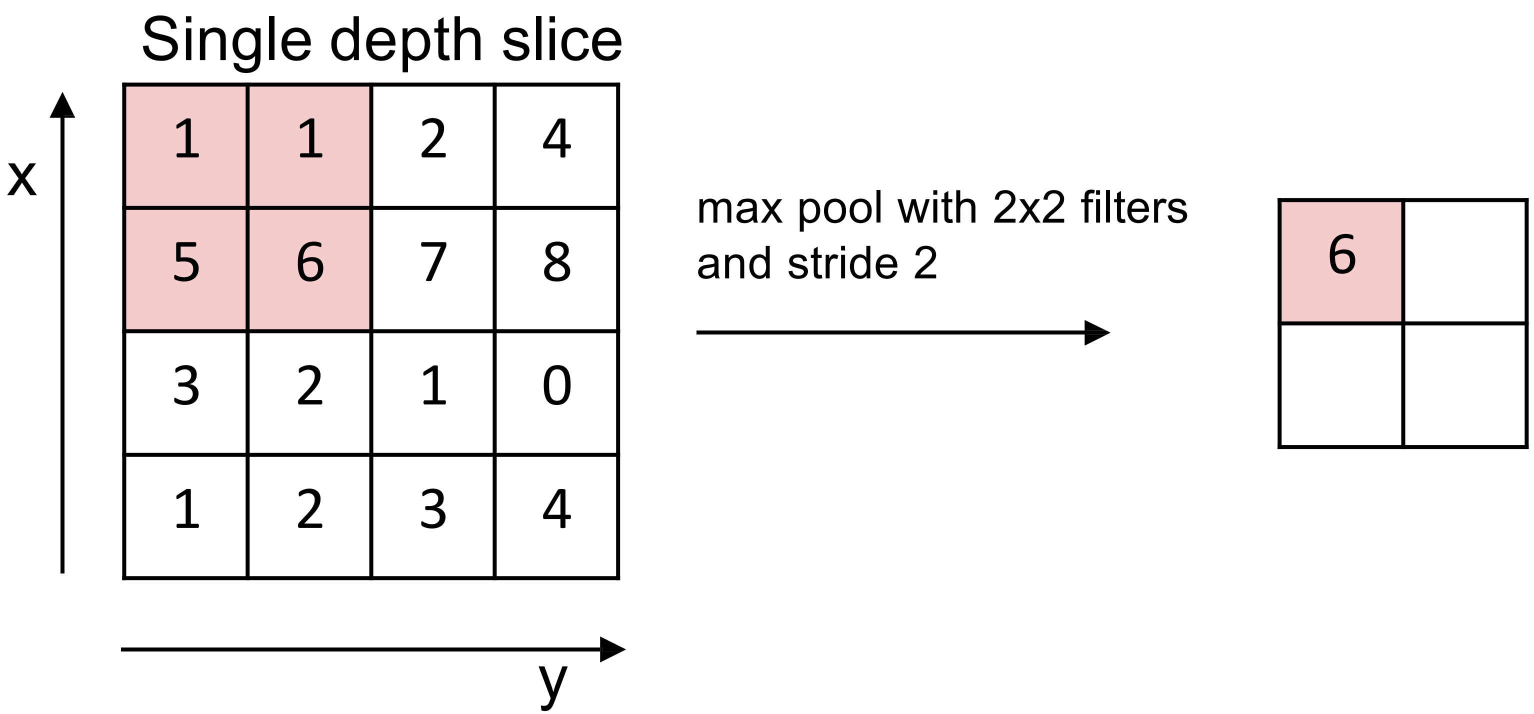
# Downsampling and Pooling



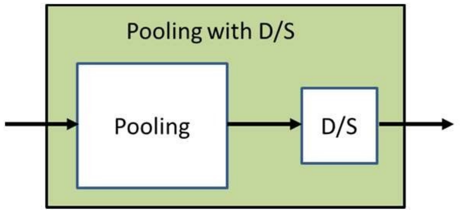

- Downsampling after a pooling layer can be merged with it to obtain pooling with stride S



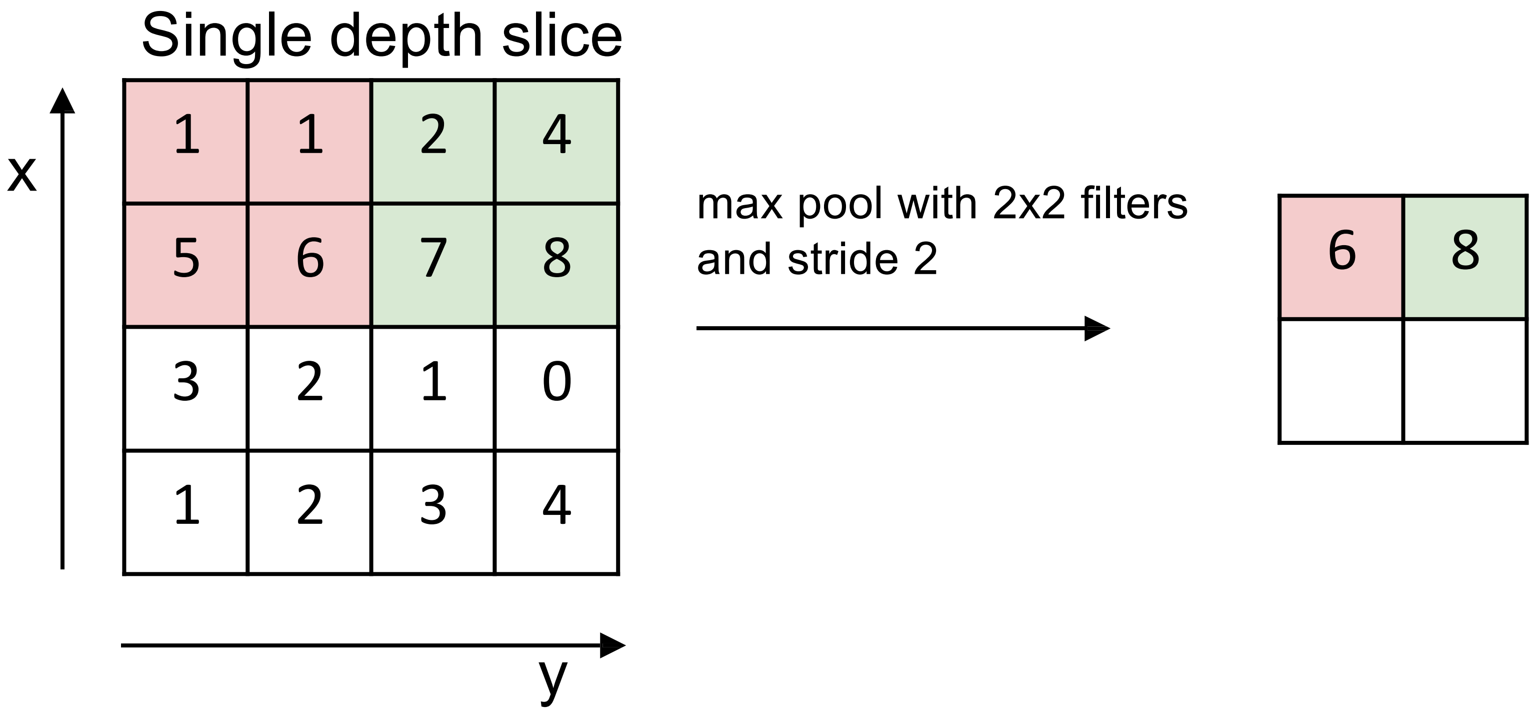

# Max Pooling



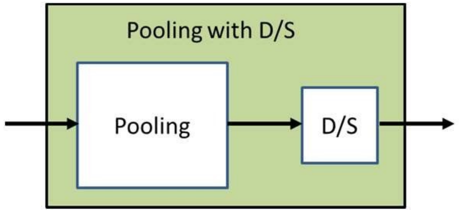

- Find the max in each block and stride by 2




# Max Pooling




- Find the max in each block and stride by 2




# Max Pooling

Single depth slice

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

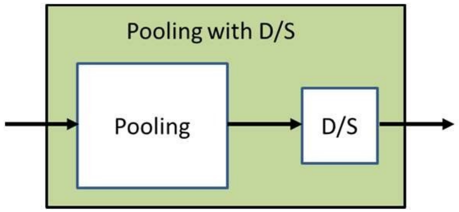
x

y

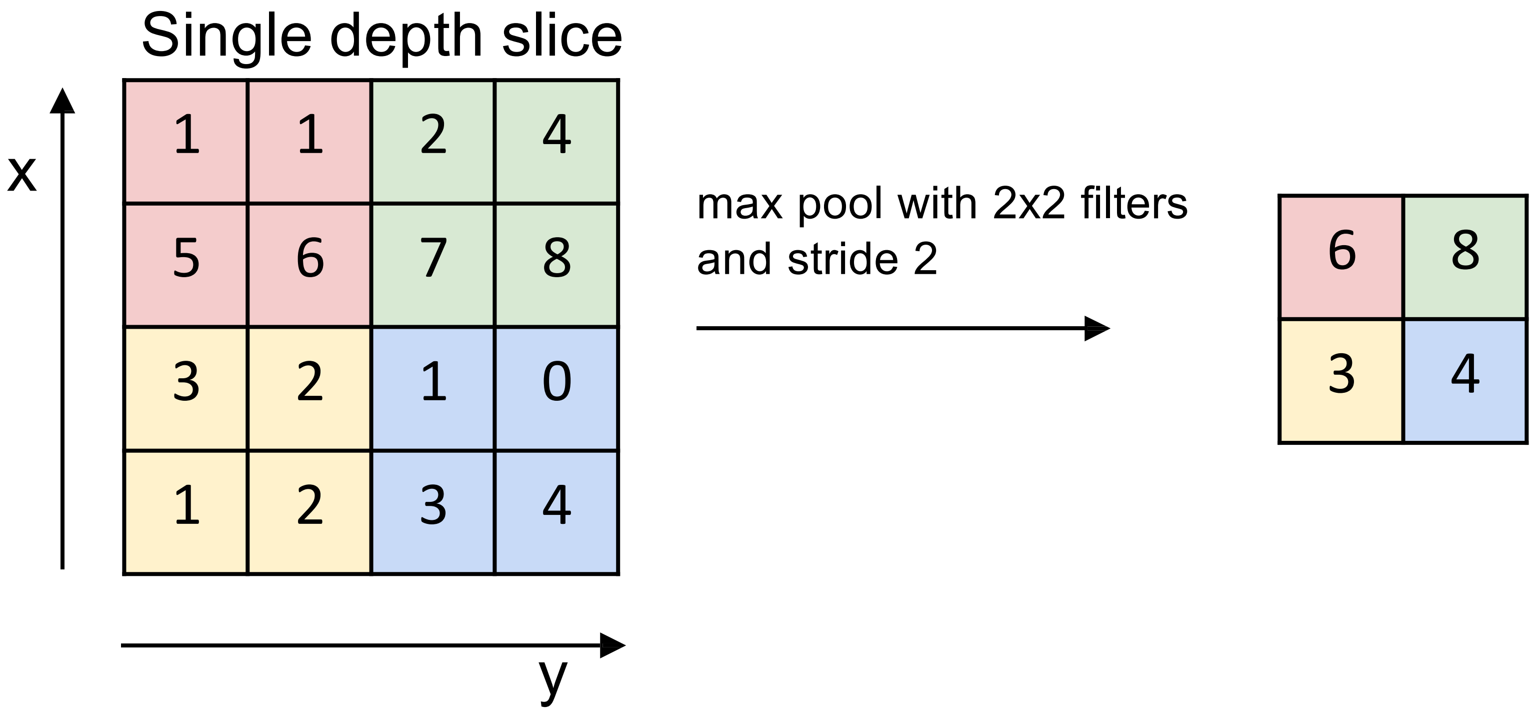
max pool with 2x2 filters and stride 2

| | |
|---|---|
| 6 | 8 |
| 3 | |

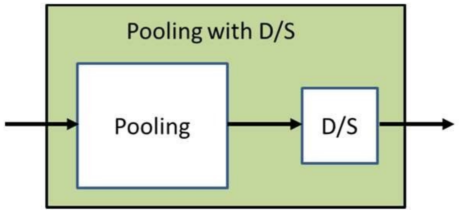
- Find the max in each block and stride by 2



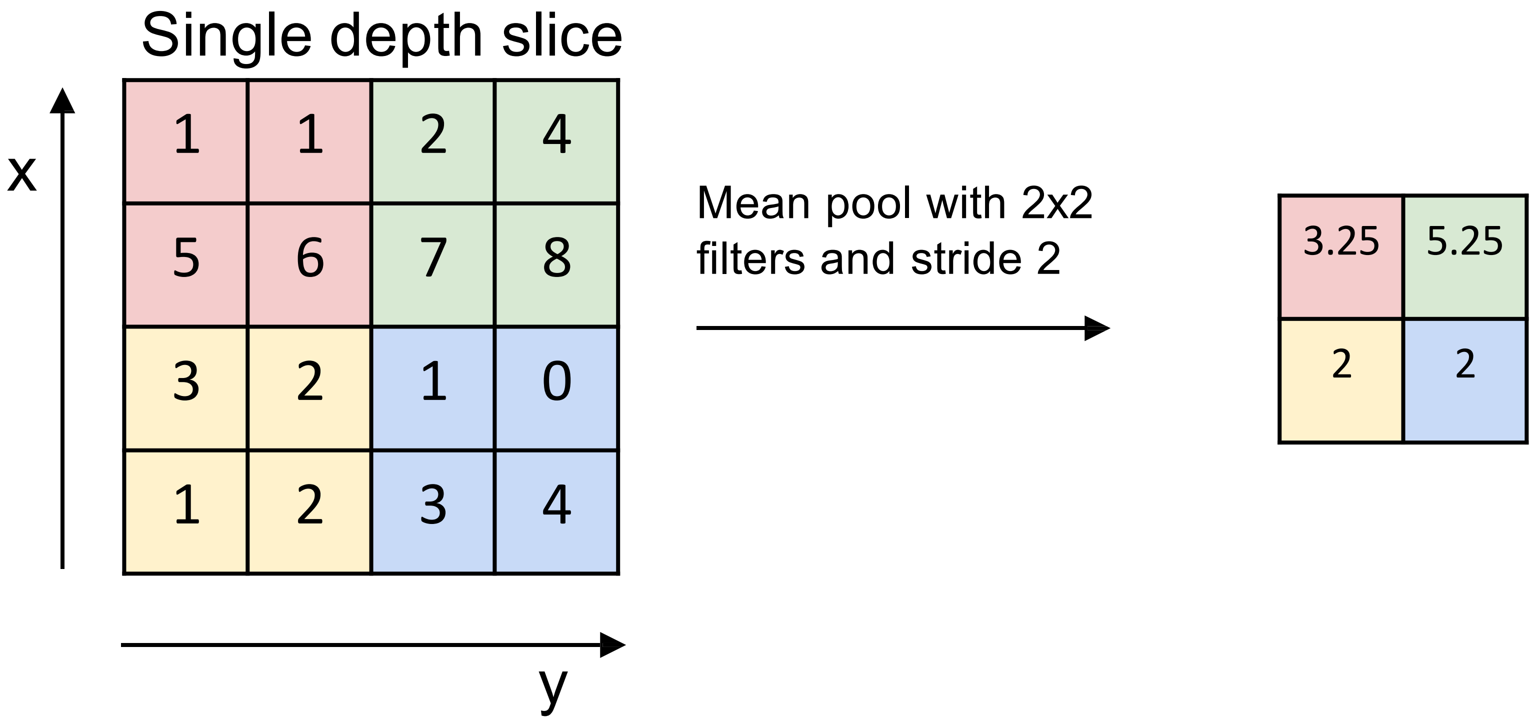

# Max Pooling




- Find the max in each block and stride by 2

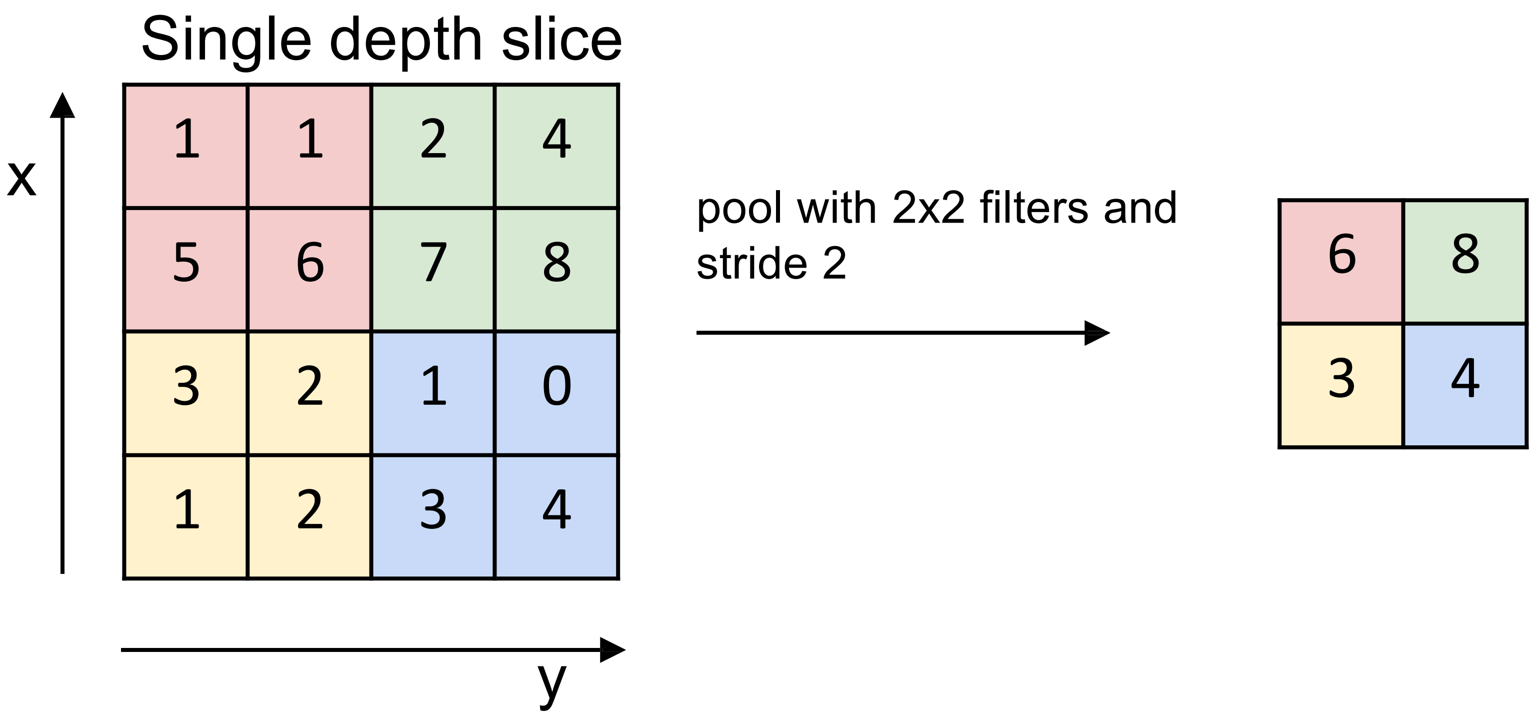



# Mean Pooling



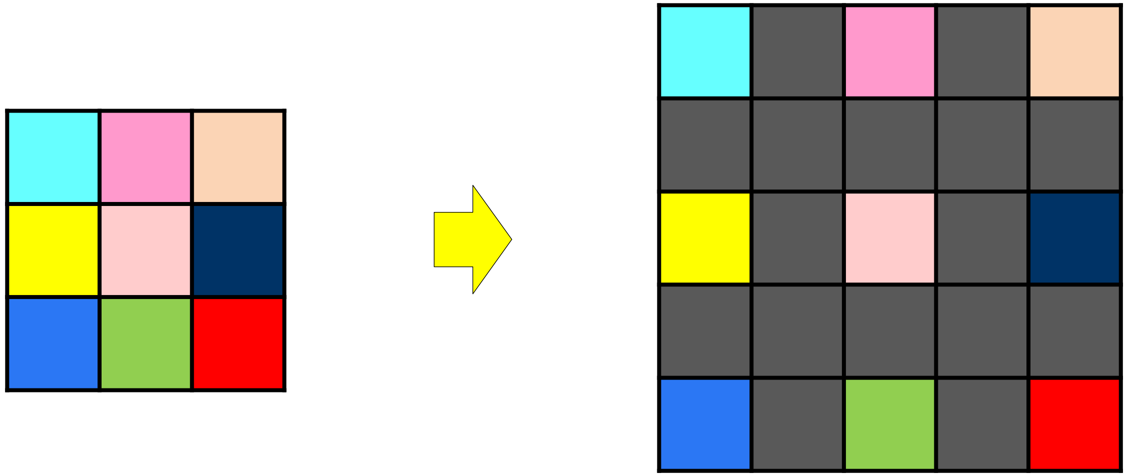

- Compute the mean of the pool, instead of the max

# Downsampling: Size of output



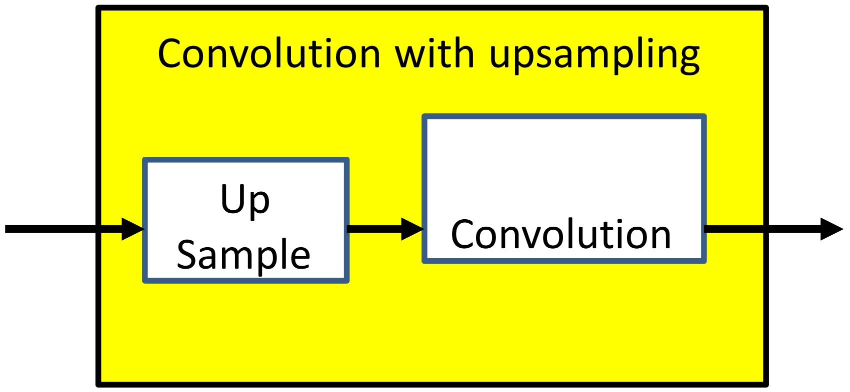

- An $N \times N$ picture compressed by a $P \times P$ pooling filter with stride $D$ results in an output map of side $\lceil (N - P)/D \rceil + 1$
  - Typically do not zero pad

# The Upsampling Layer



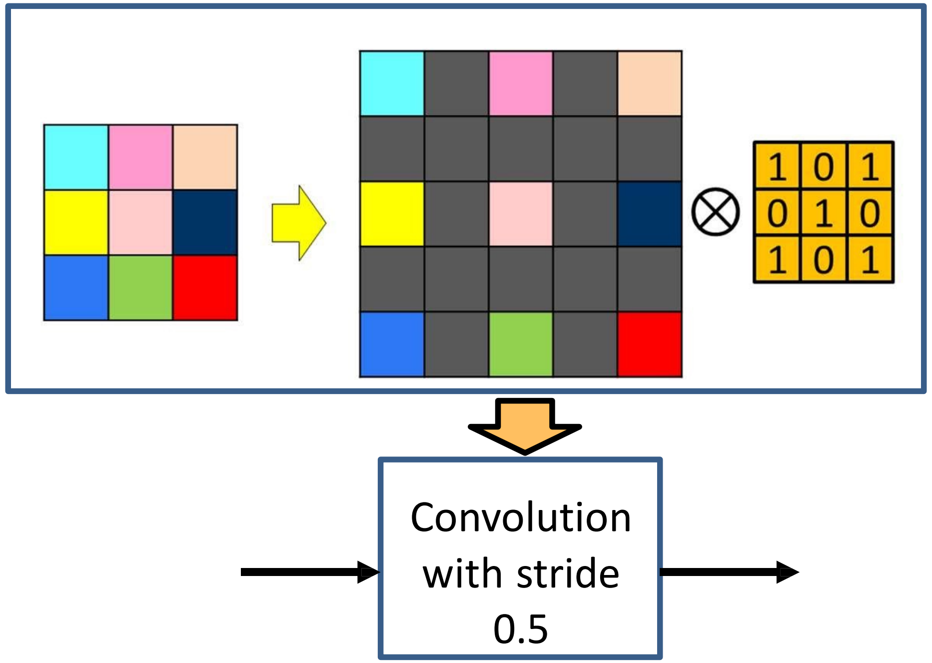
- A *upsampling* (or dilation) layer simply introduces $S-1$ rows and columns for every map in the layer
  - Effectively *increasing* the size of the map by factor $S$ in every direction
- Used explicitly to increase the map size by a uniform factor

# The Upsampling Layer




- A *upsampling* layer is generally followed by a CNN layer
  - It is **not** useful to follow it by a pooling layer
  - It is also **not** useful as the *final* layer of a CNN

# The Upsampling Layer




- Upsampling layers followed by a convolutional layer are also often viewed as convolving with a fractional stride
  - Upsampling by factor $S$ is the same as striding by factor $1/S$

# Resampling Summary

- Map sizes can be changed by downsampling or upsampling
  - Downsampling: Drop S-1 of S rows and columns
  - Upsampling: Insert S-1 zeros between every two rows / columns
- Downsampling typically *follows* convolution or pooling
  - Reduces the size of the maps
- Upsampling occurs *before* convolution
  - Increases the size of the map
  - Does not generally occur before pooling layers