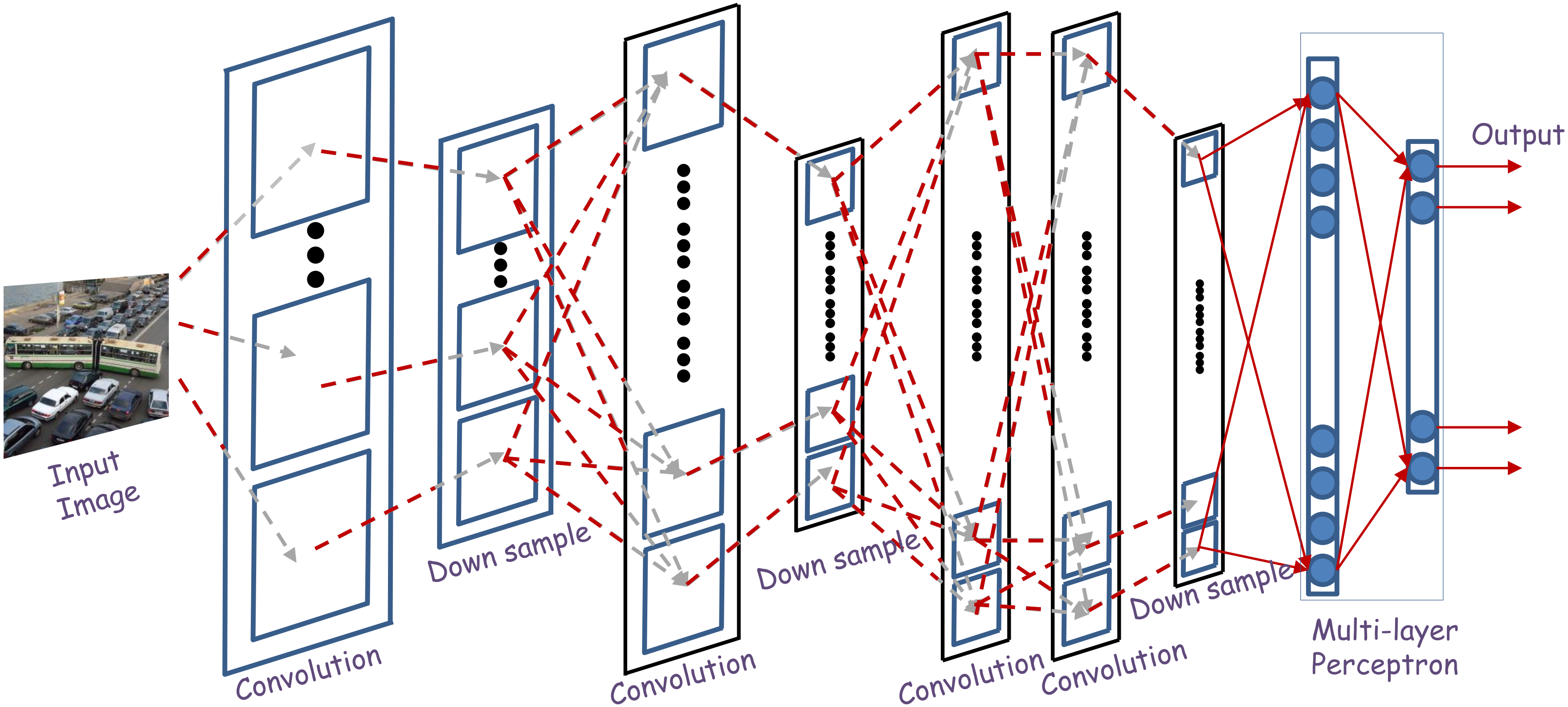
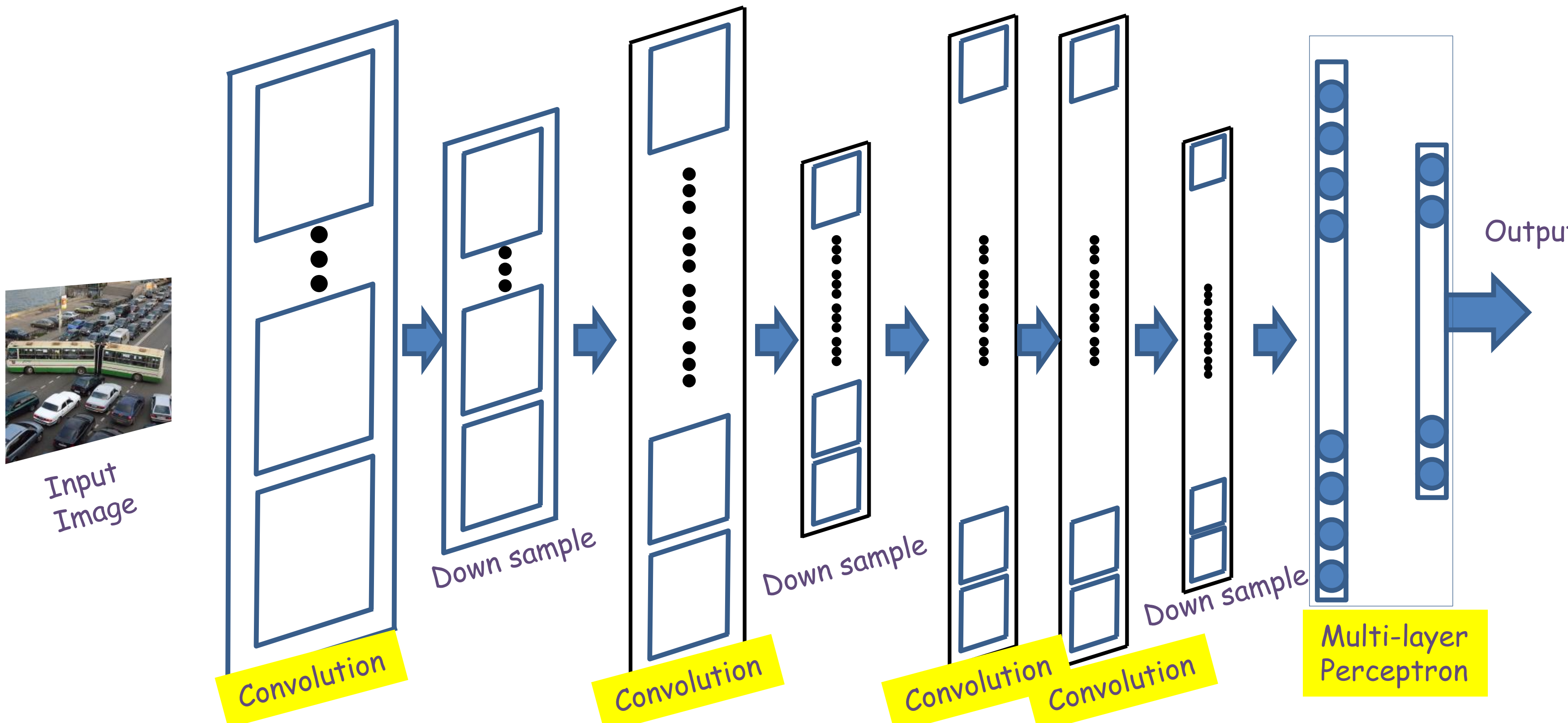
The general architecture of a convolutional neural network



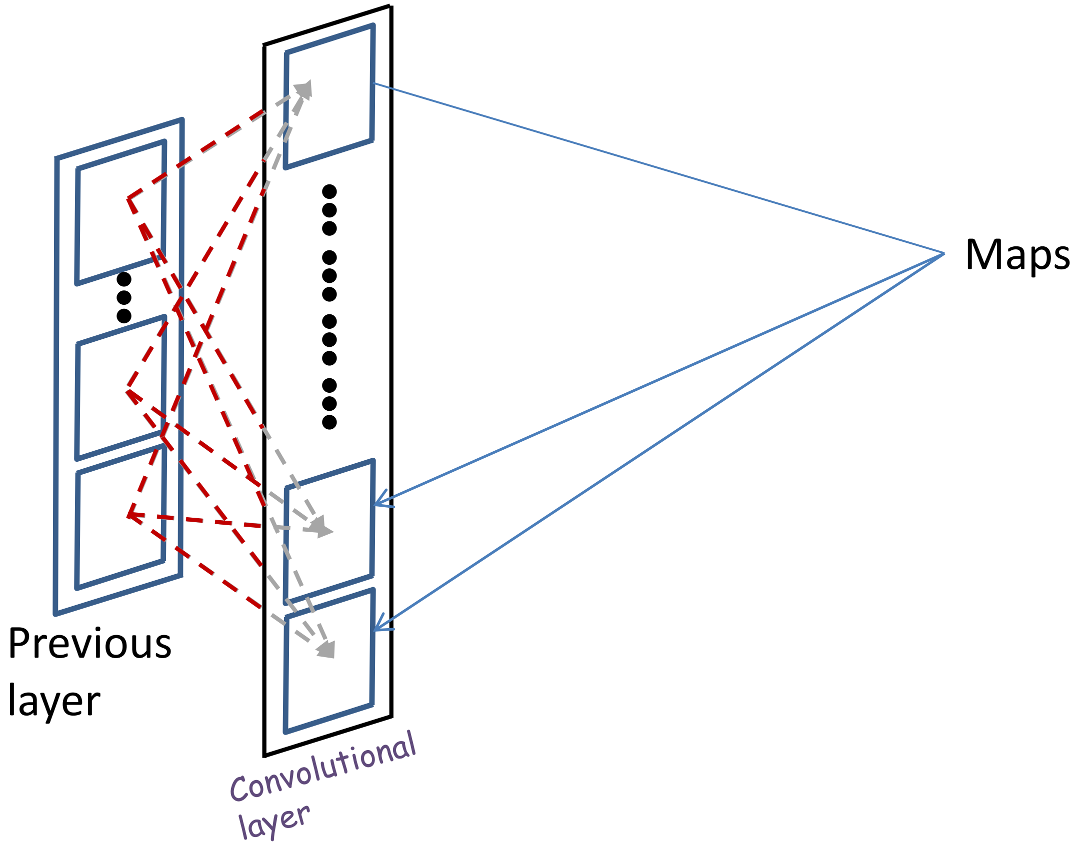
- A convolutional neural network comprises of “convolutional” and “downsampling” layers
  - The two may occur in any sequence, but typically they alternate
- Followed by an MLP with one or more layers

The general architecture of a convolutional neural network



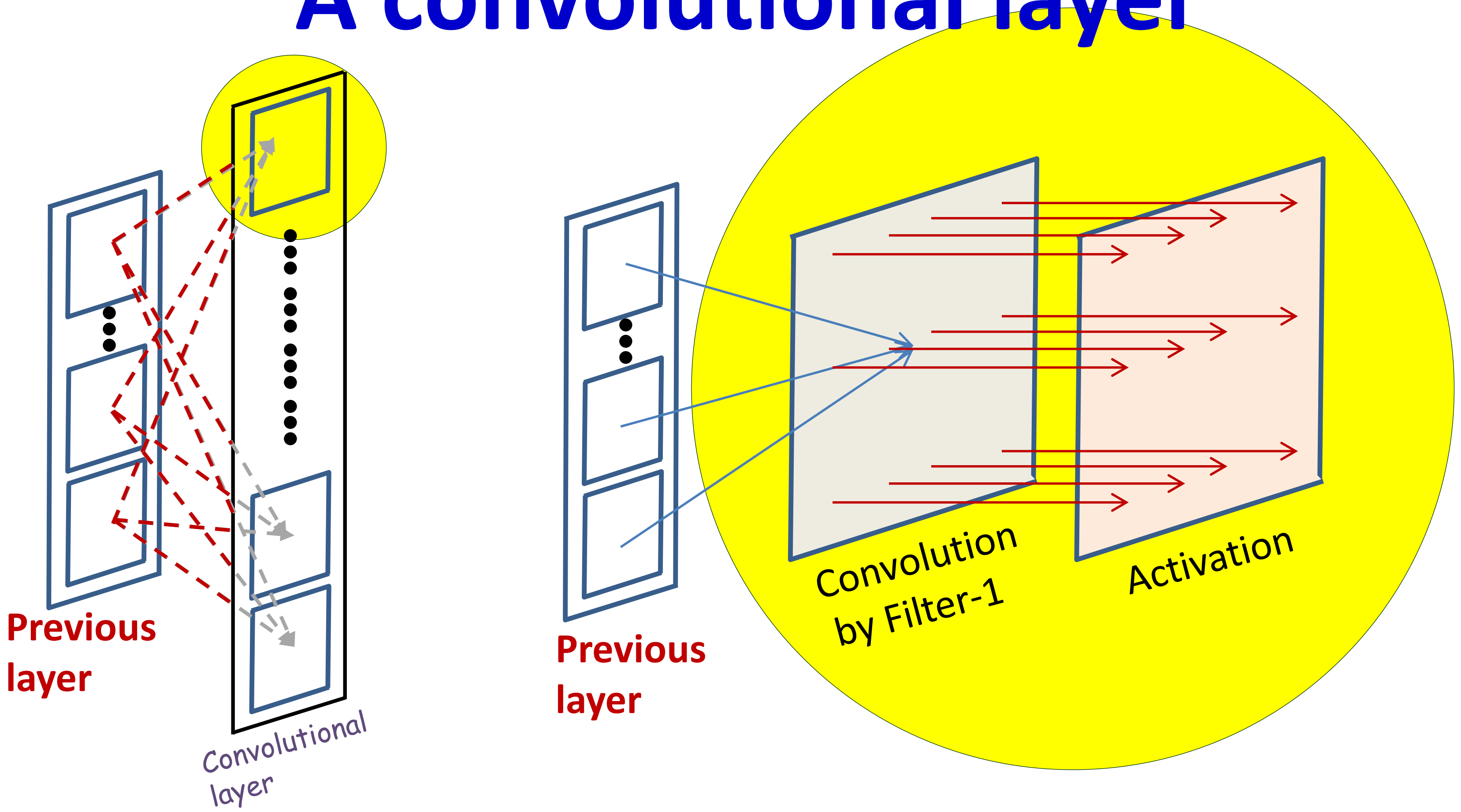
- **Convolutional layers and the MLP are *learnable***
  - Their parameters must be learned from training data for the target classification task
- Down-sampling layers are fixed and generally not learnable

A convolutional layer



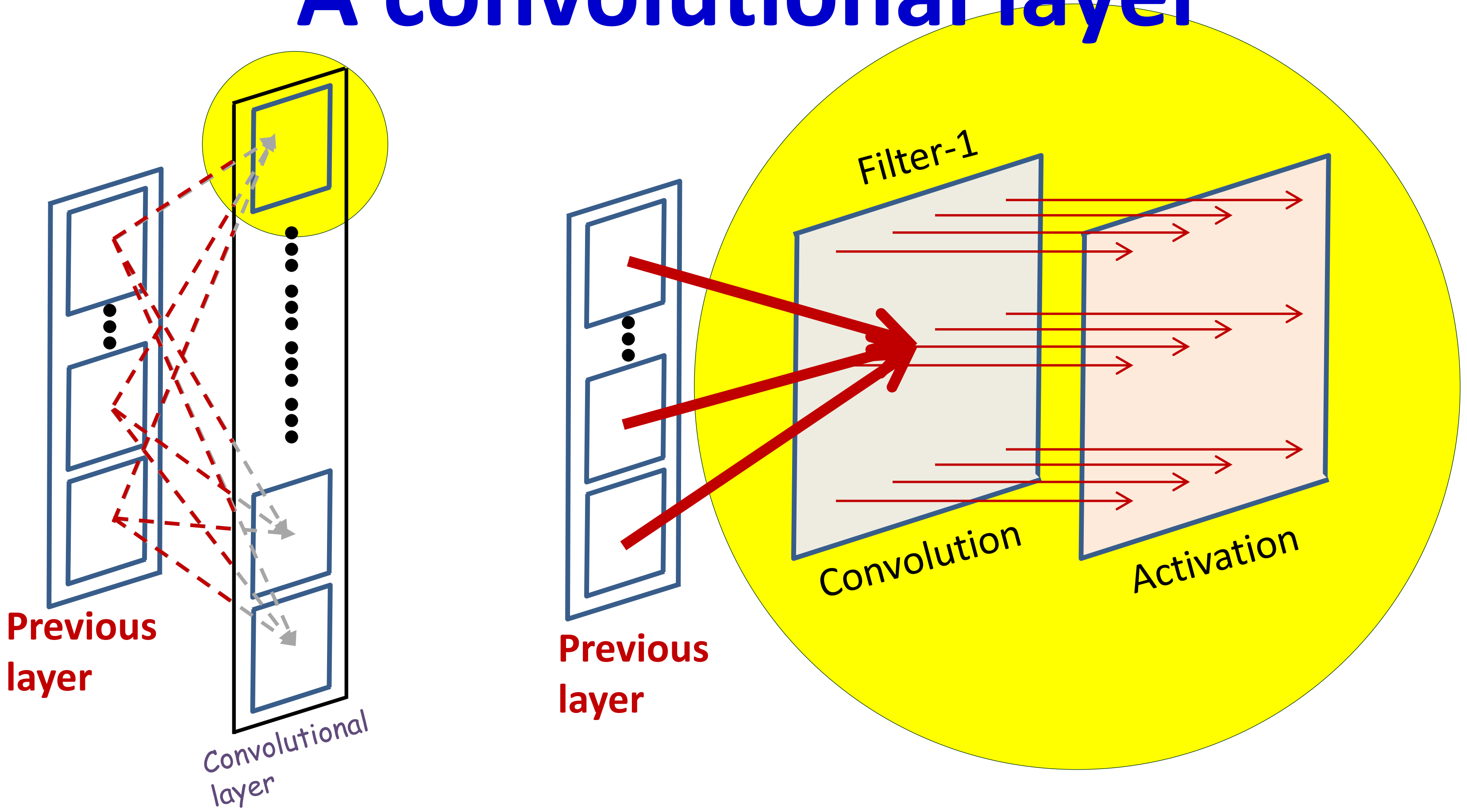
- A convolutional layer comprises of a series of “maps”
  - Corresponding the “S-planes” in the Neocognitron
  - Variously called feature maps or activation maps

A convolutional layer



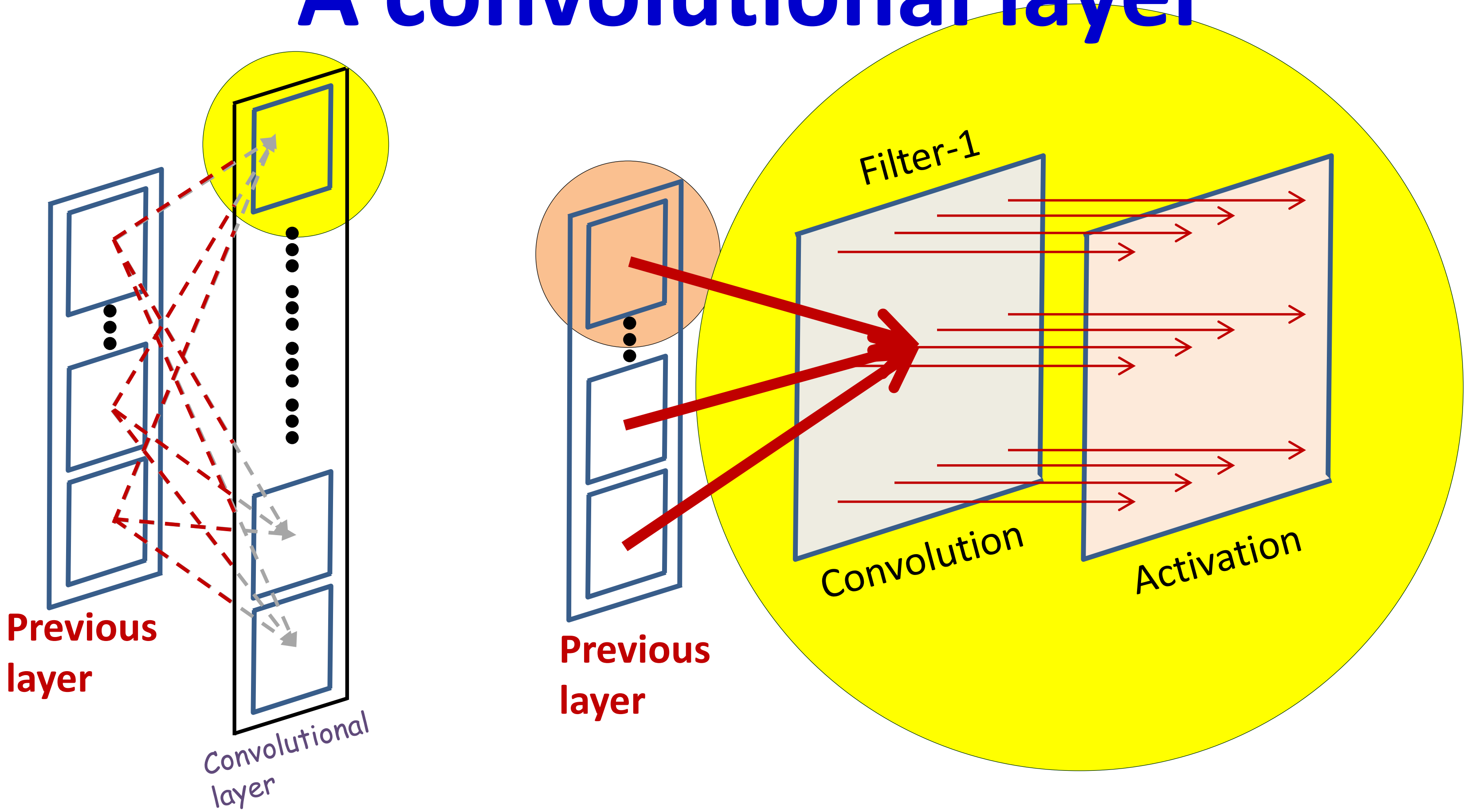
- Each activation map has two components
  - An *affine* map, obtained by *convolution* over maps in the previous layer
    - Each affine map has, associated with it, a ***learnable filter***
  - An *activation* that operates on the output of the convolution

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

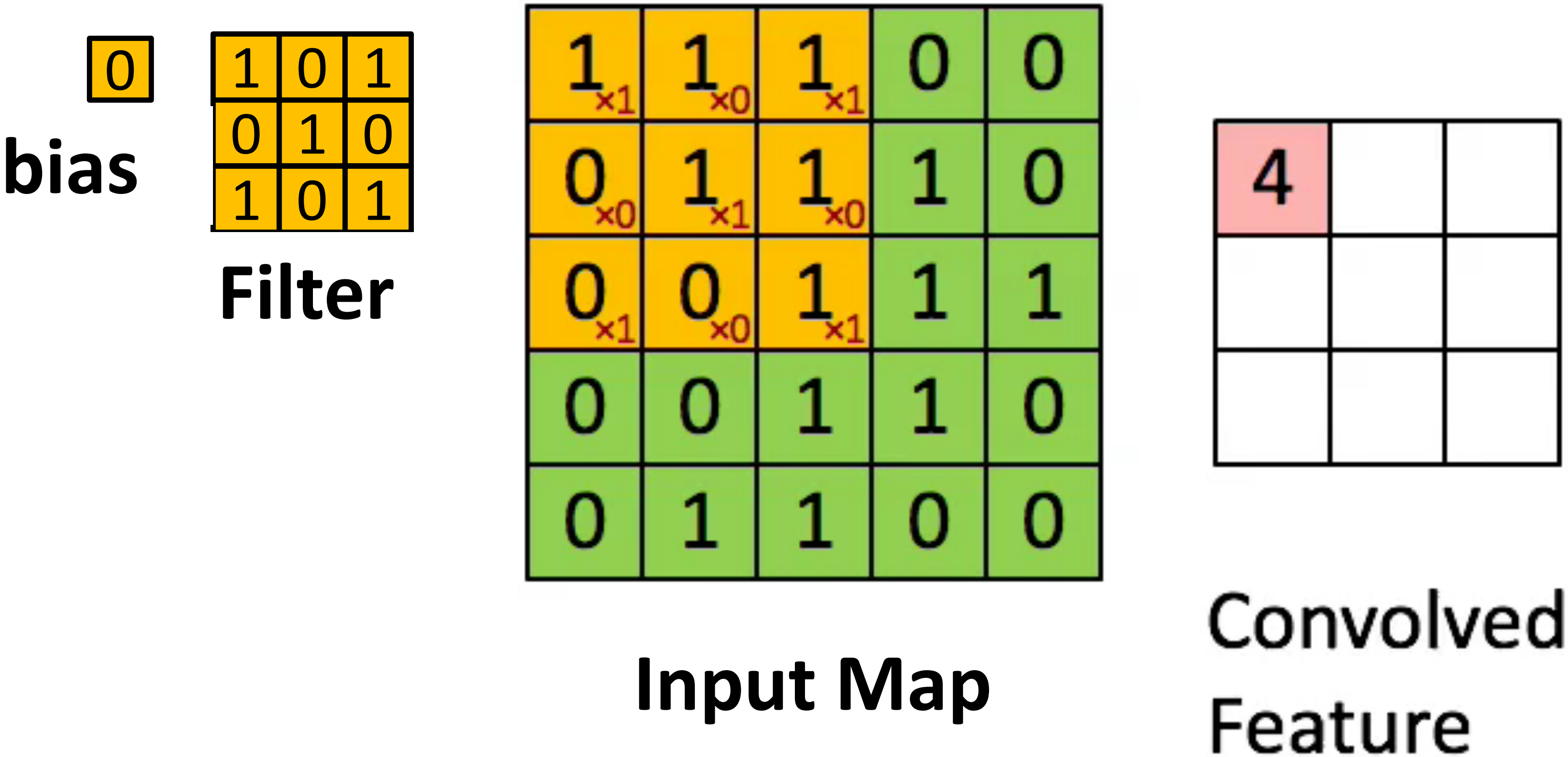
1	0	1
0	1	0
1	0	1

bias

0

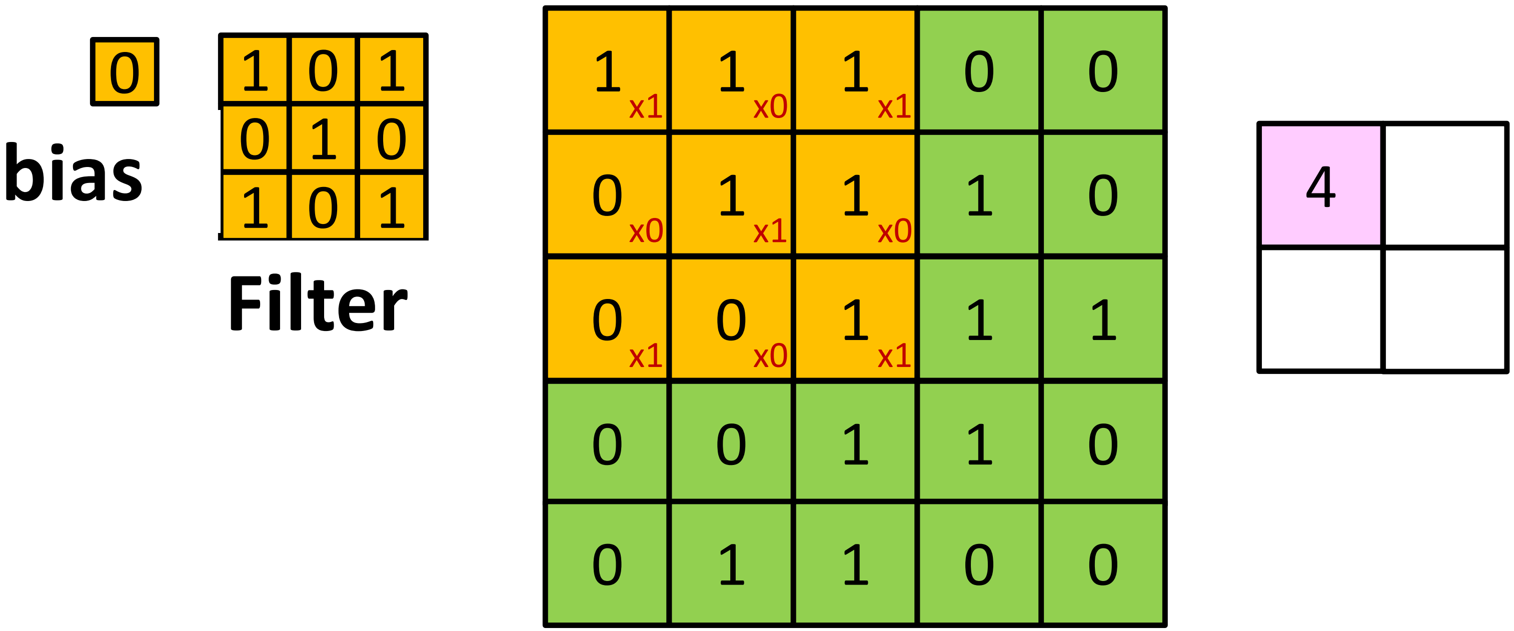
- Scanning an image with a “filter”
  - Note: a filter is really just a perceptron, with weights and a bias

What is a convolution



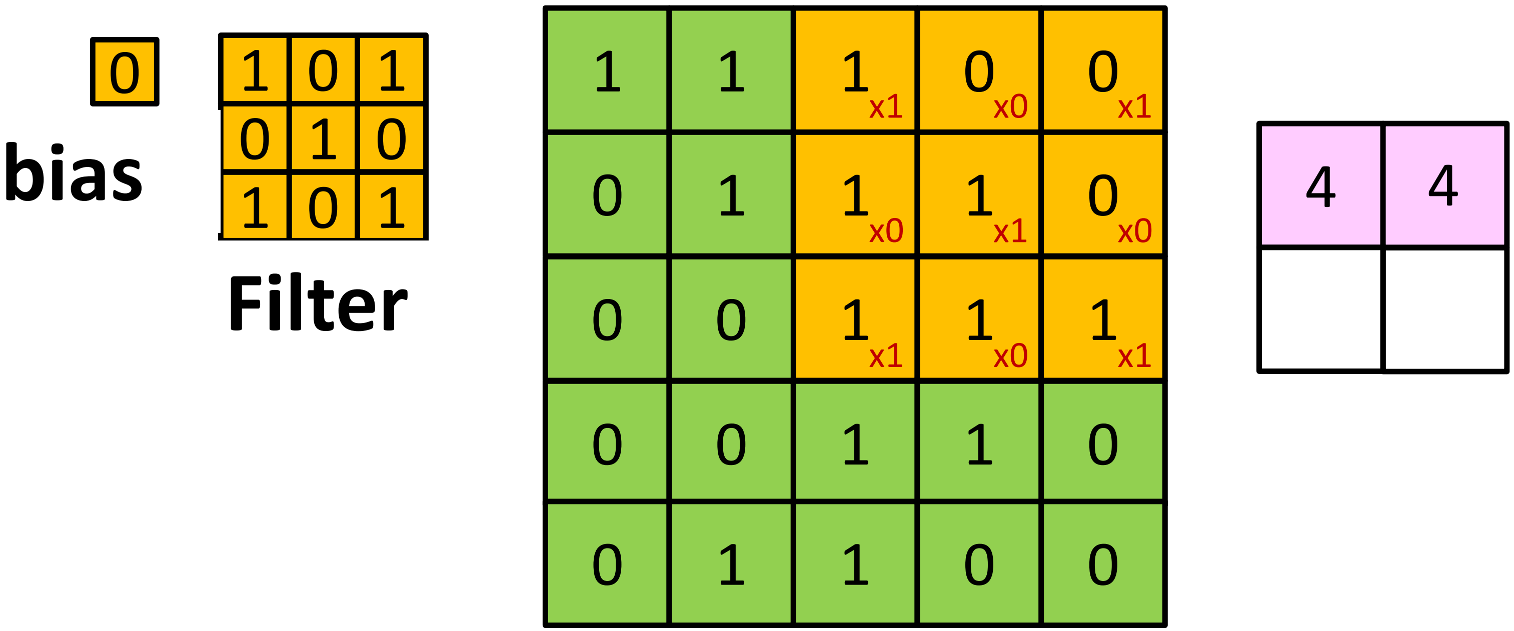
- Scanning an image with a “filter”
  - At each location, the “filter and the underlying map values are multiplied component wise, and the products are added along with the bias

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



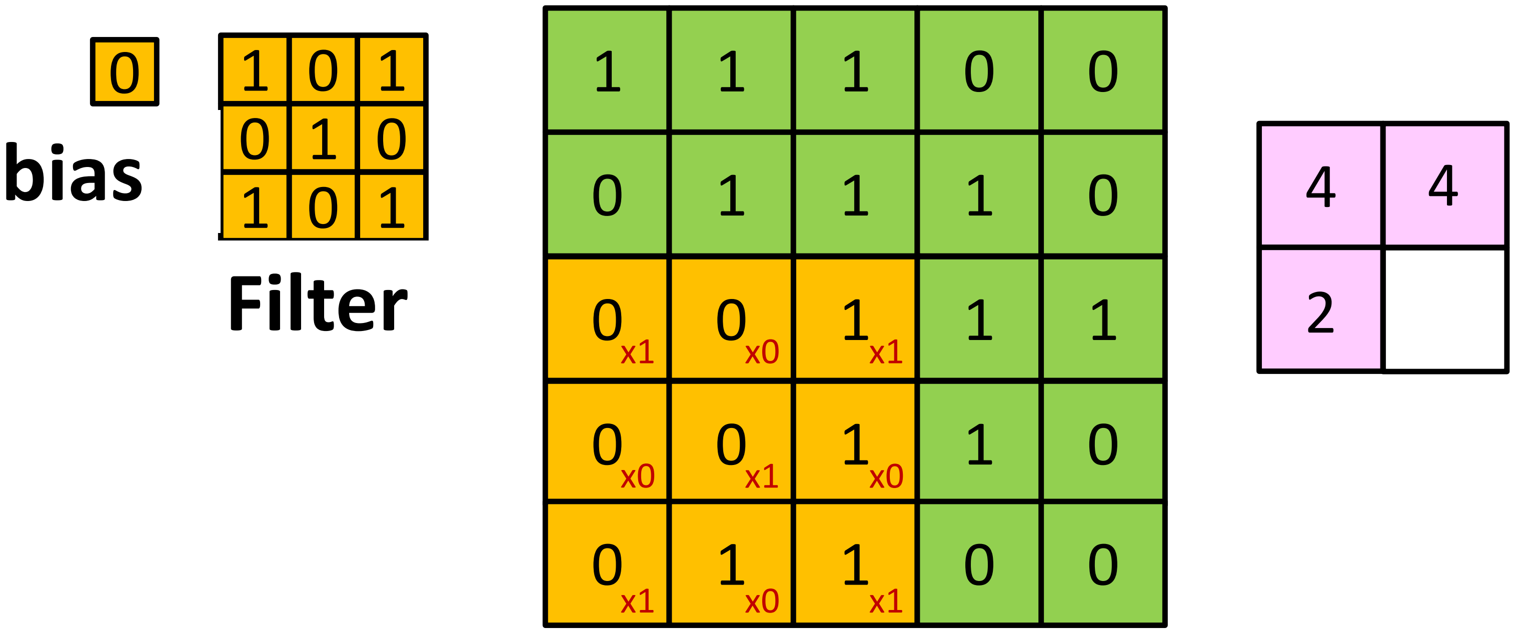
- Scanning an image with a “filter”
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a “stride” of *two* pixels per shift

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



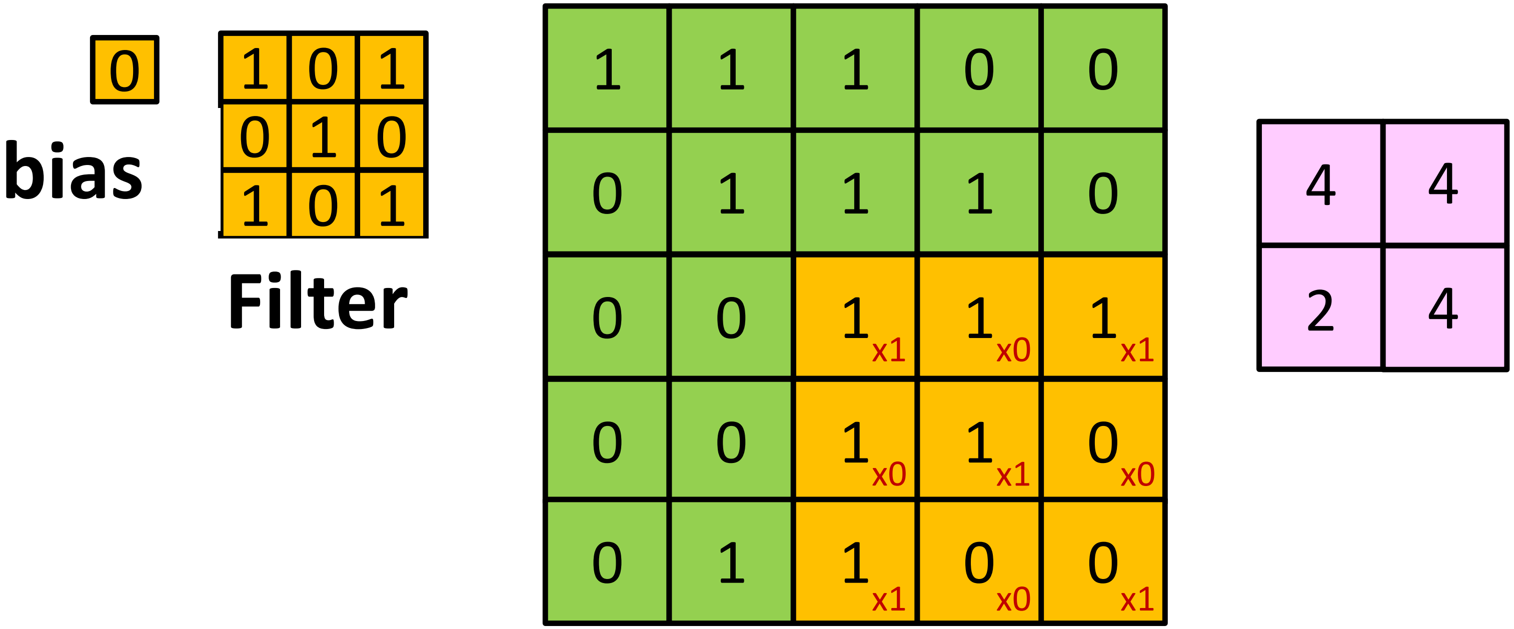
- Scanning an image with a “filter”
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a “hop” of *two* pixels per shift

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



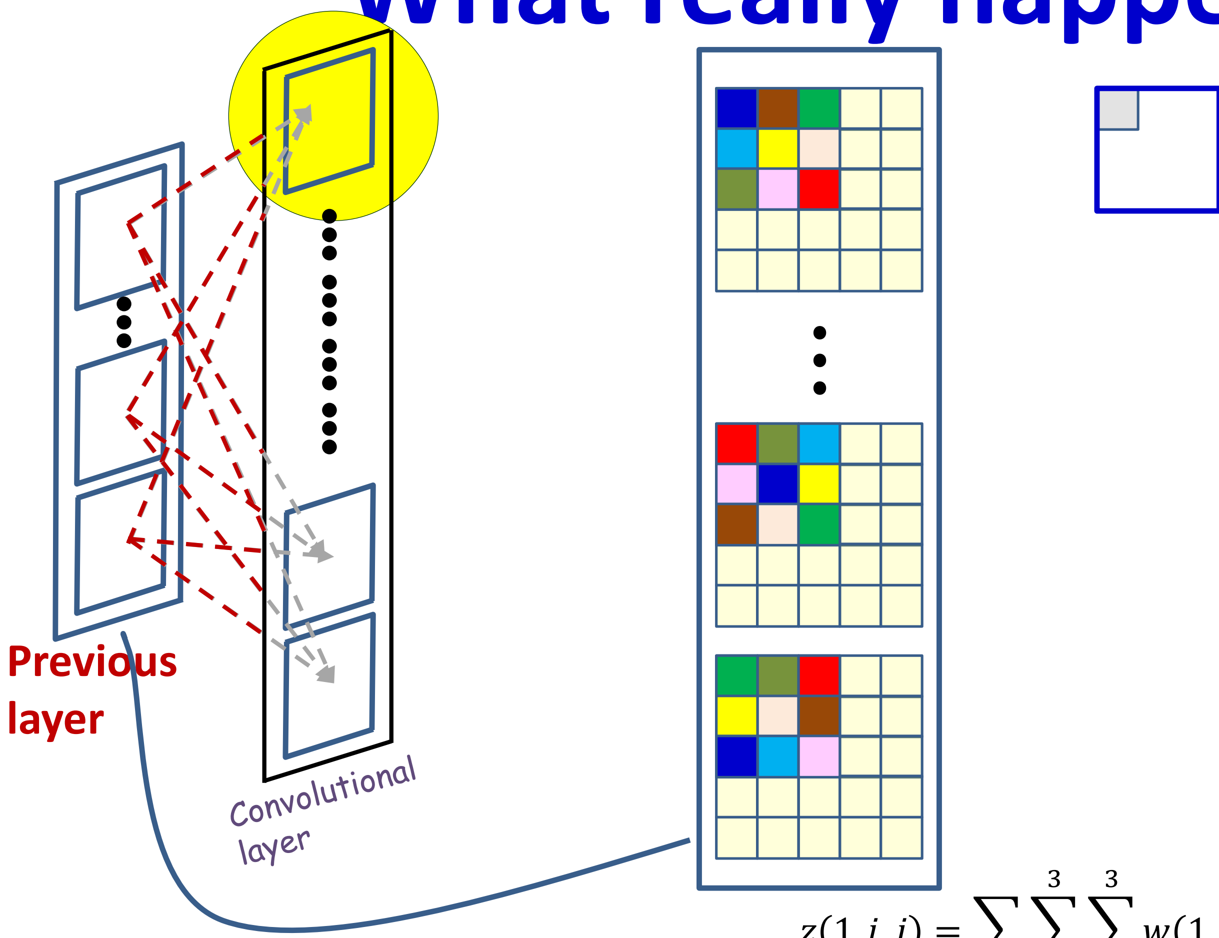
- Scanning an image with a “filter”
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a “hop” of *two* pixels per shift

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

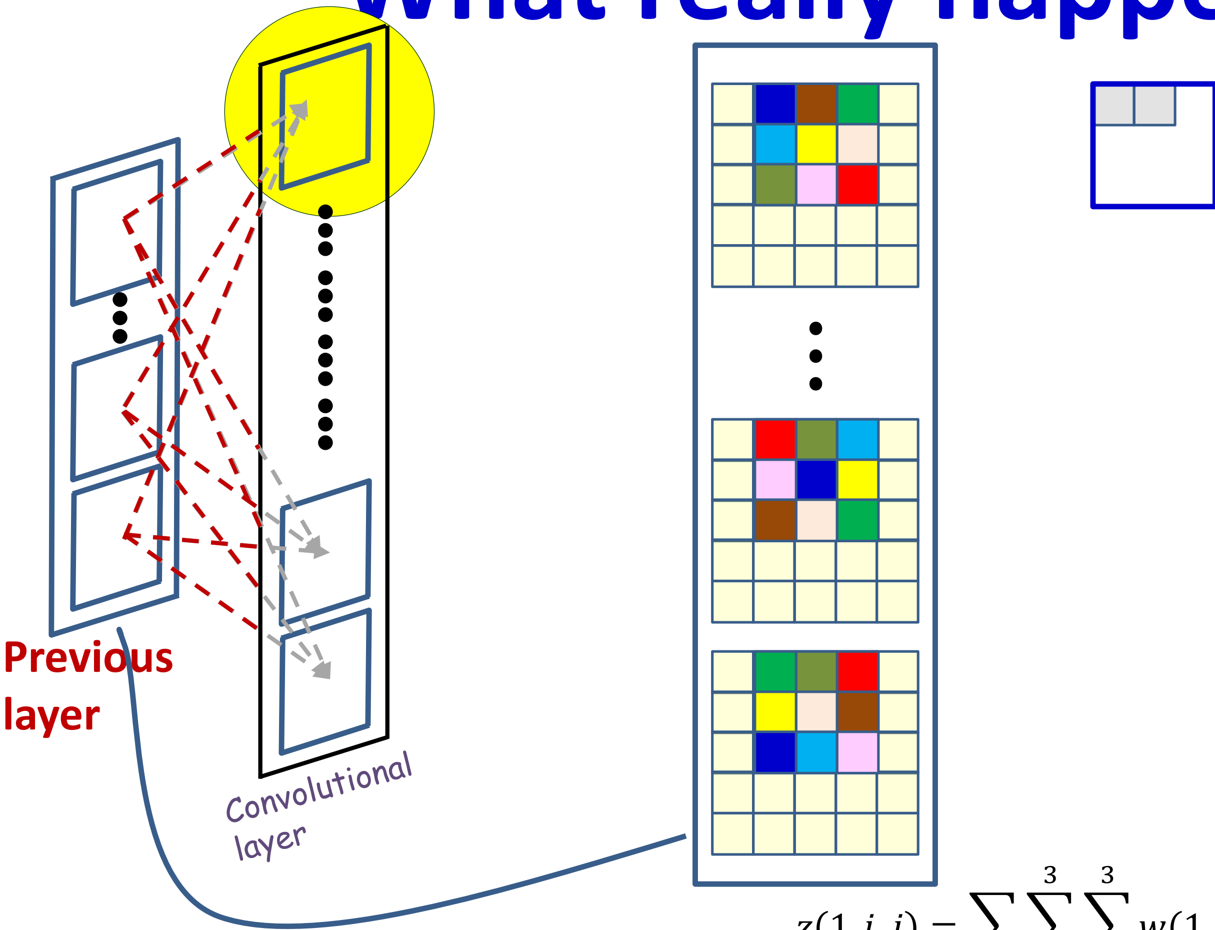


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- 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

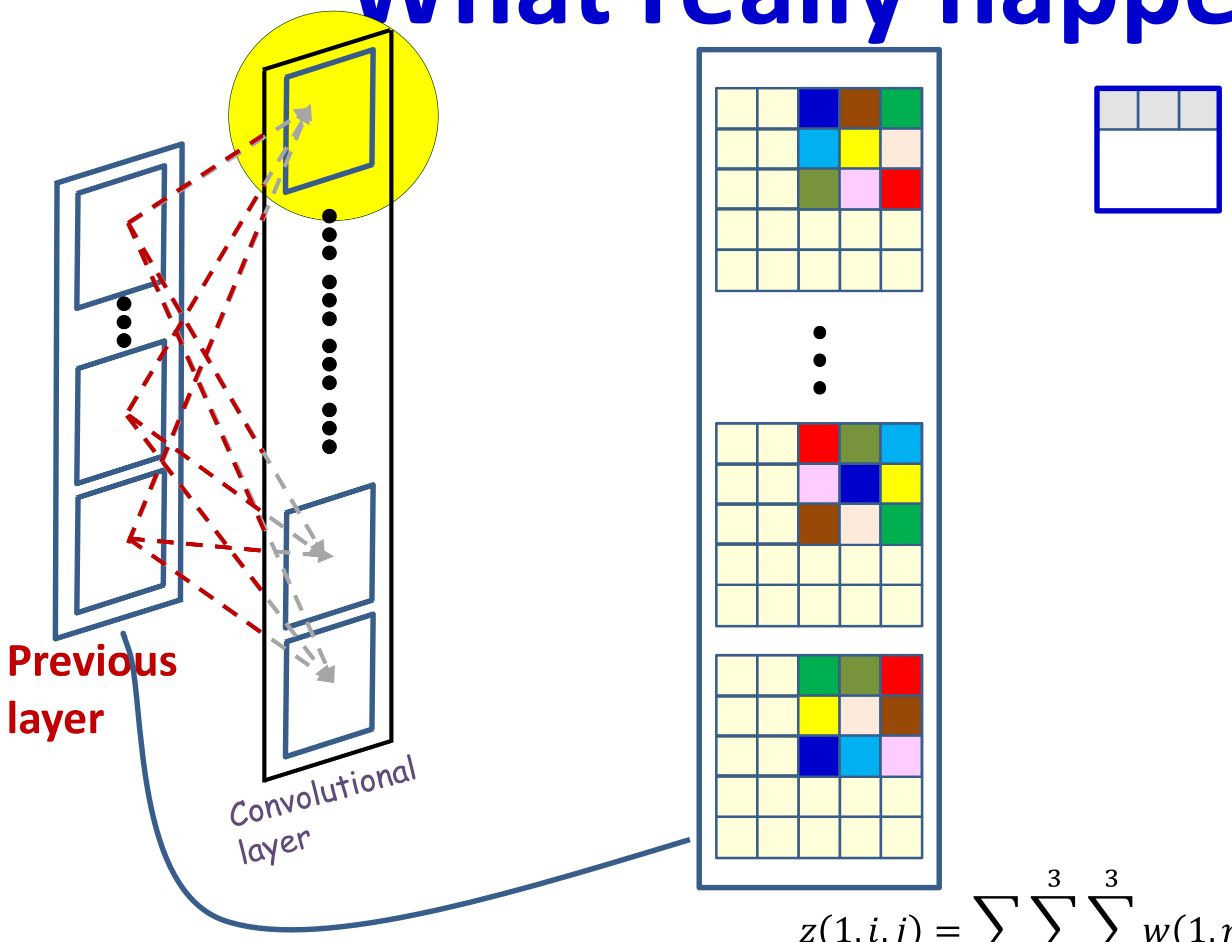


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m,i+l-1,j+k-1) + b$$

- 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

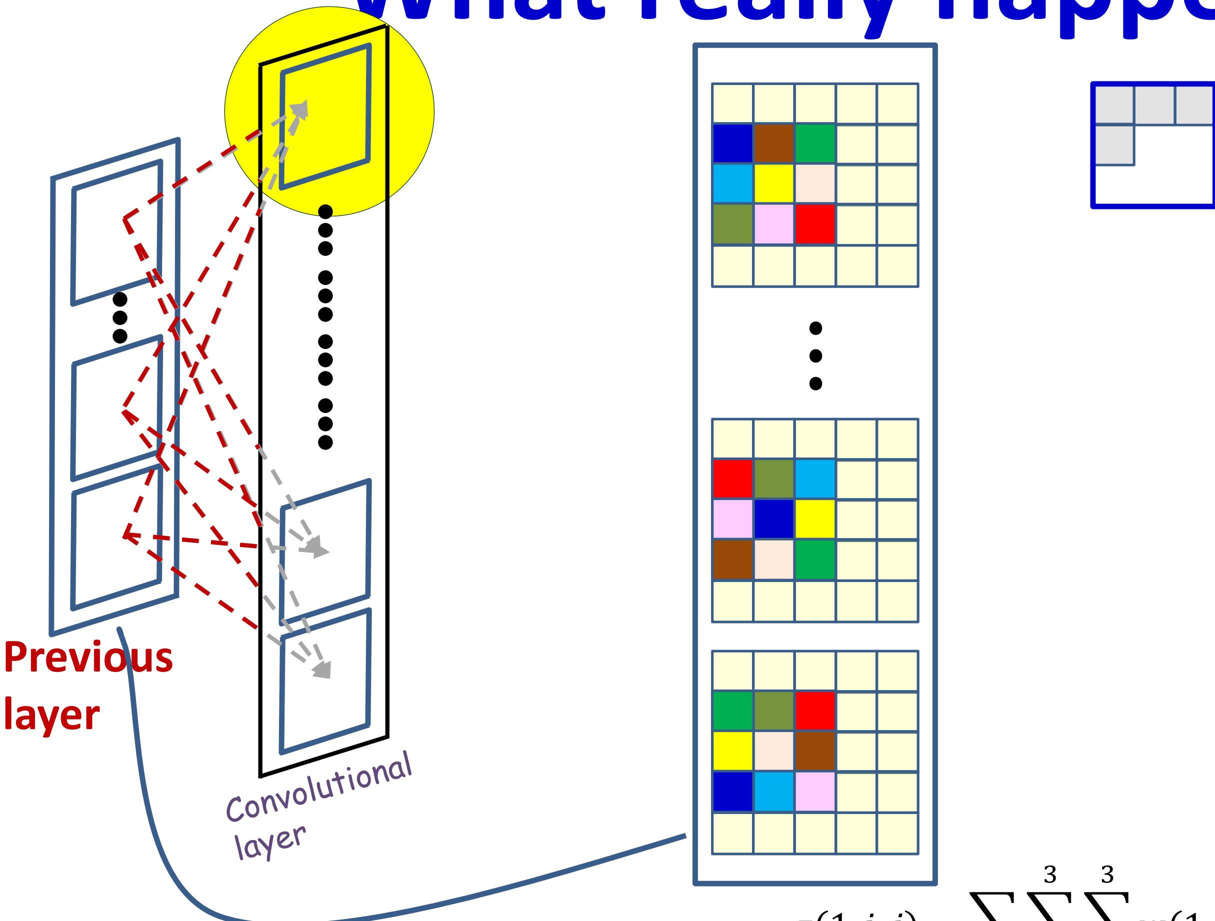


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- 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

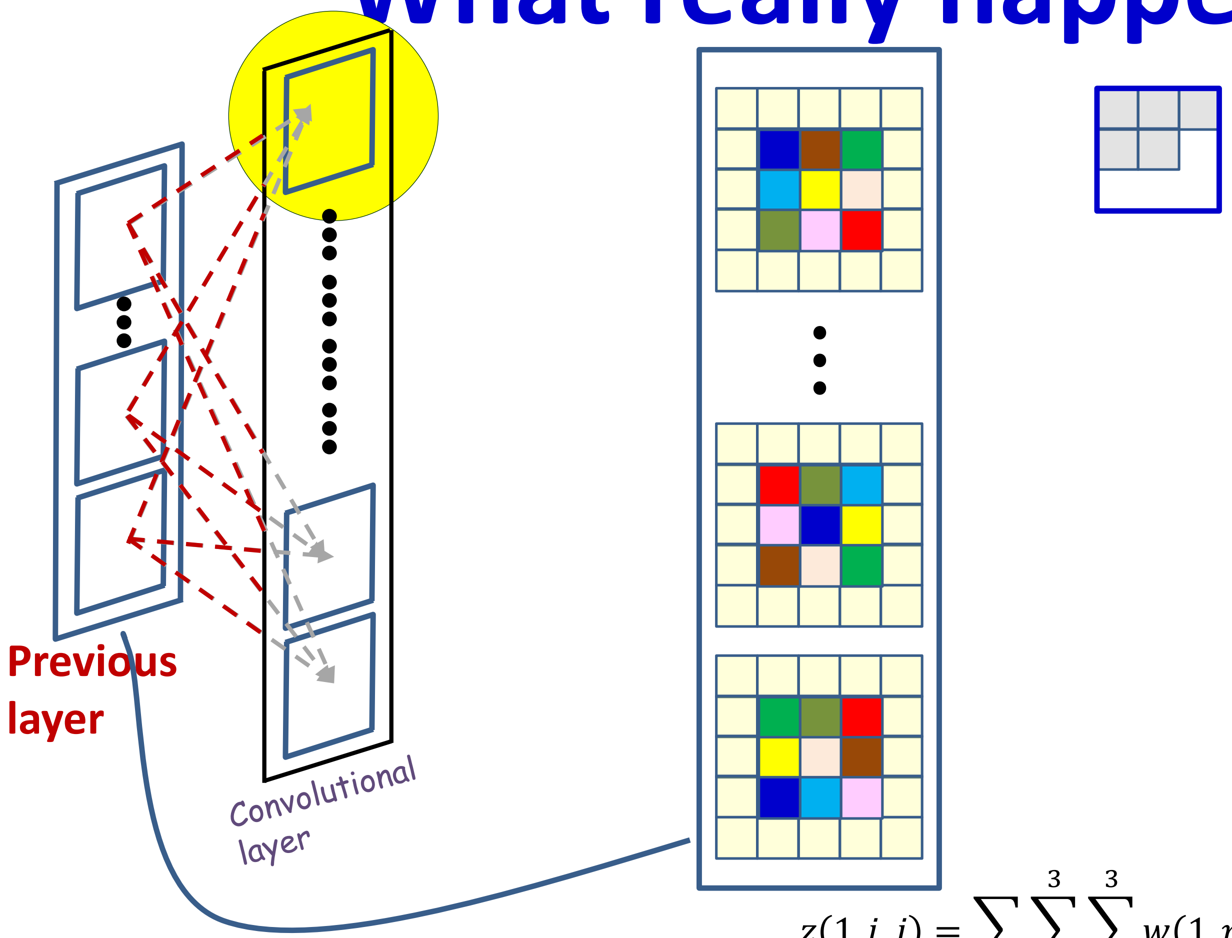


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- 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

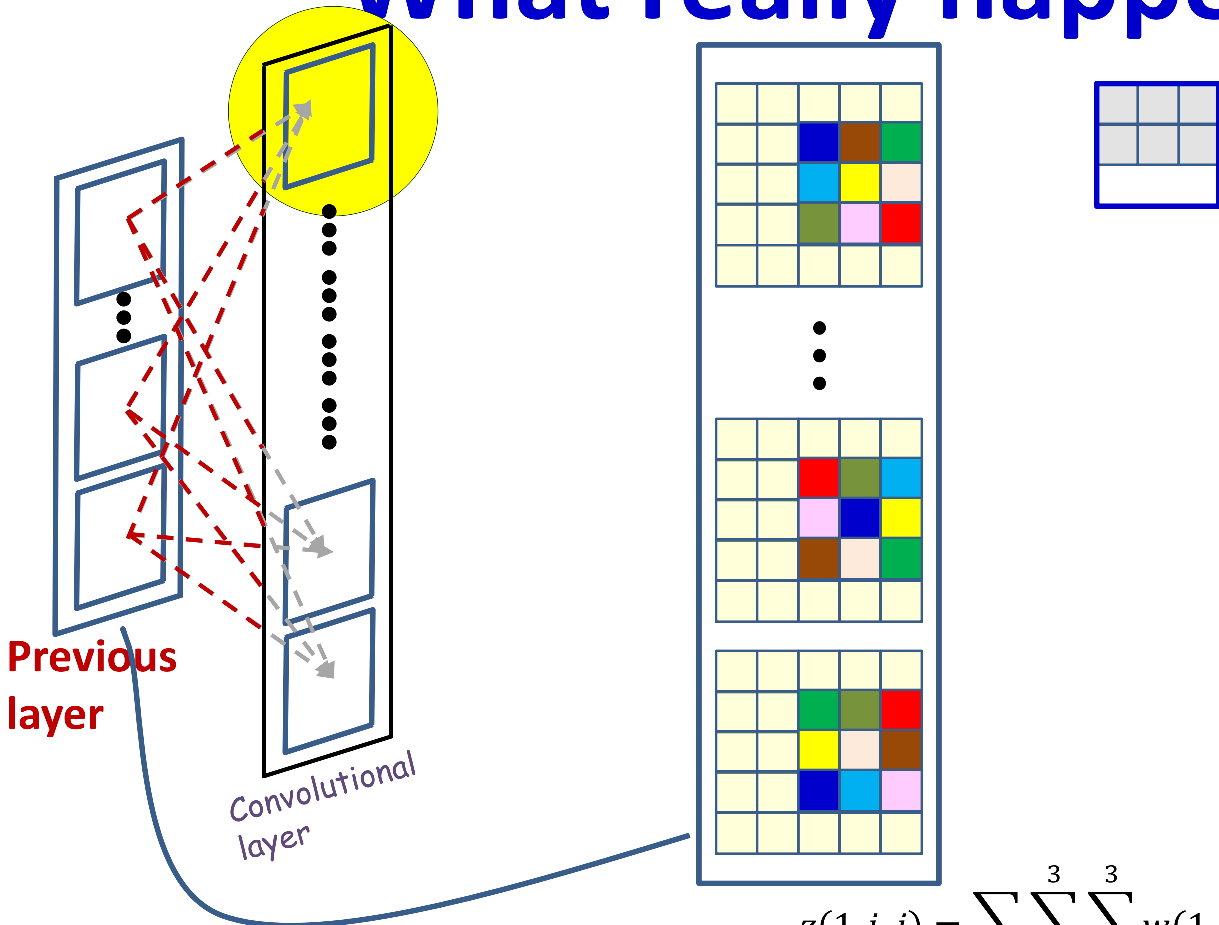


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- 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

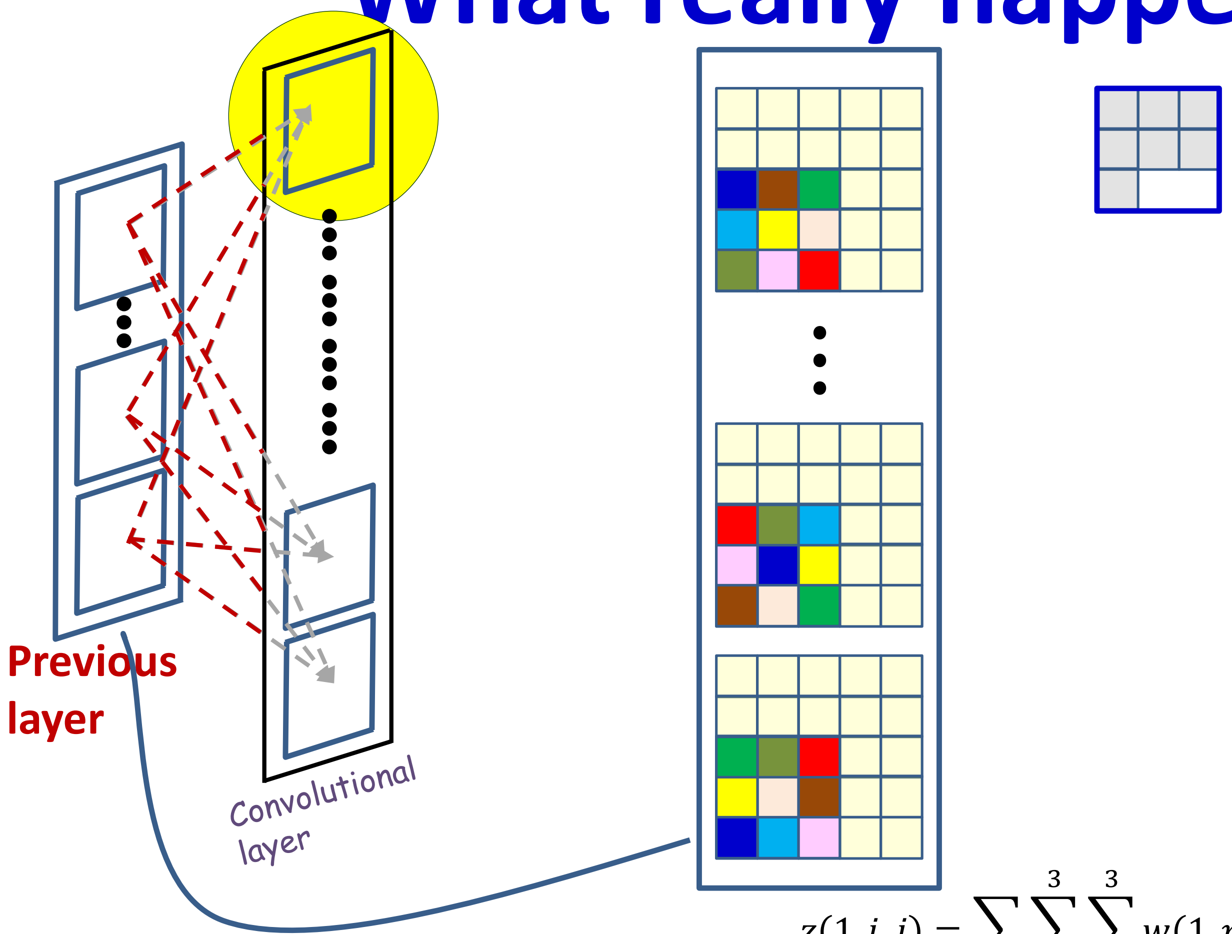


$$z(1,i,j) = \sum_{m}\sum_{k=1}^{3}\sum_{l=1}^{3} w(1,m,k,l)I(m,i+l-1,j+k-1) + b$$

- 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*

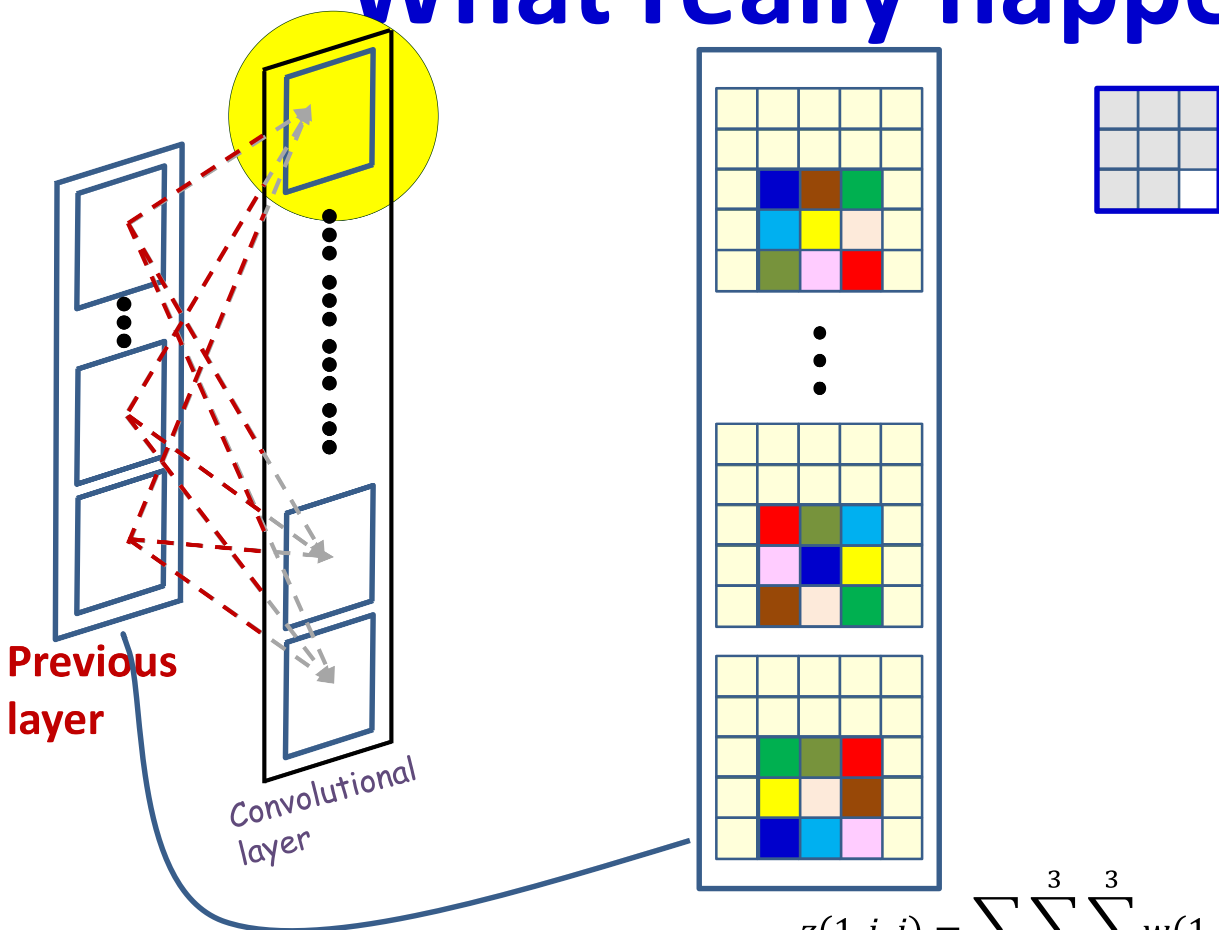
What really happens



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

- 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

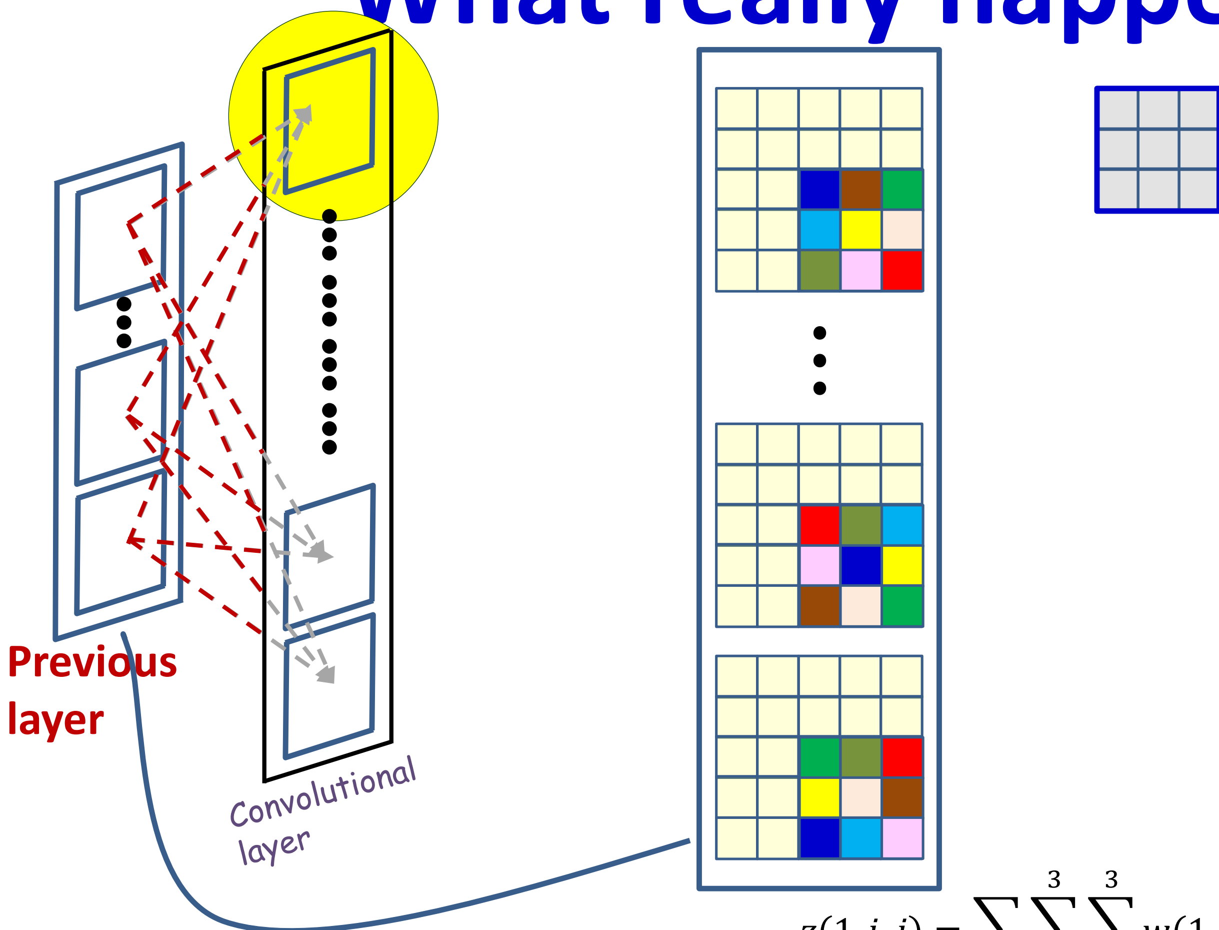


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

- 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*

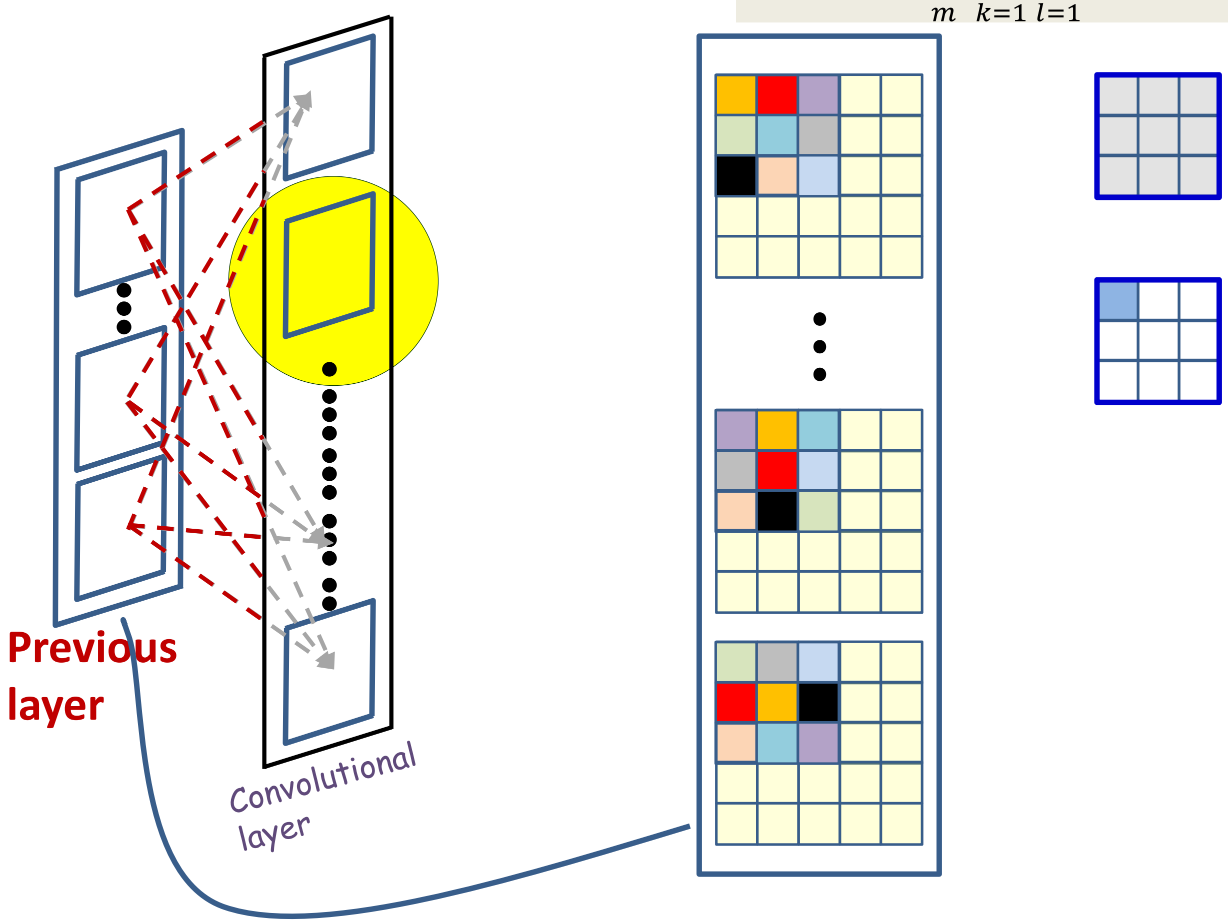
What really happens



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

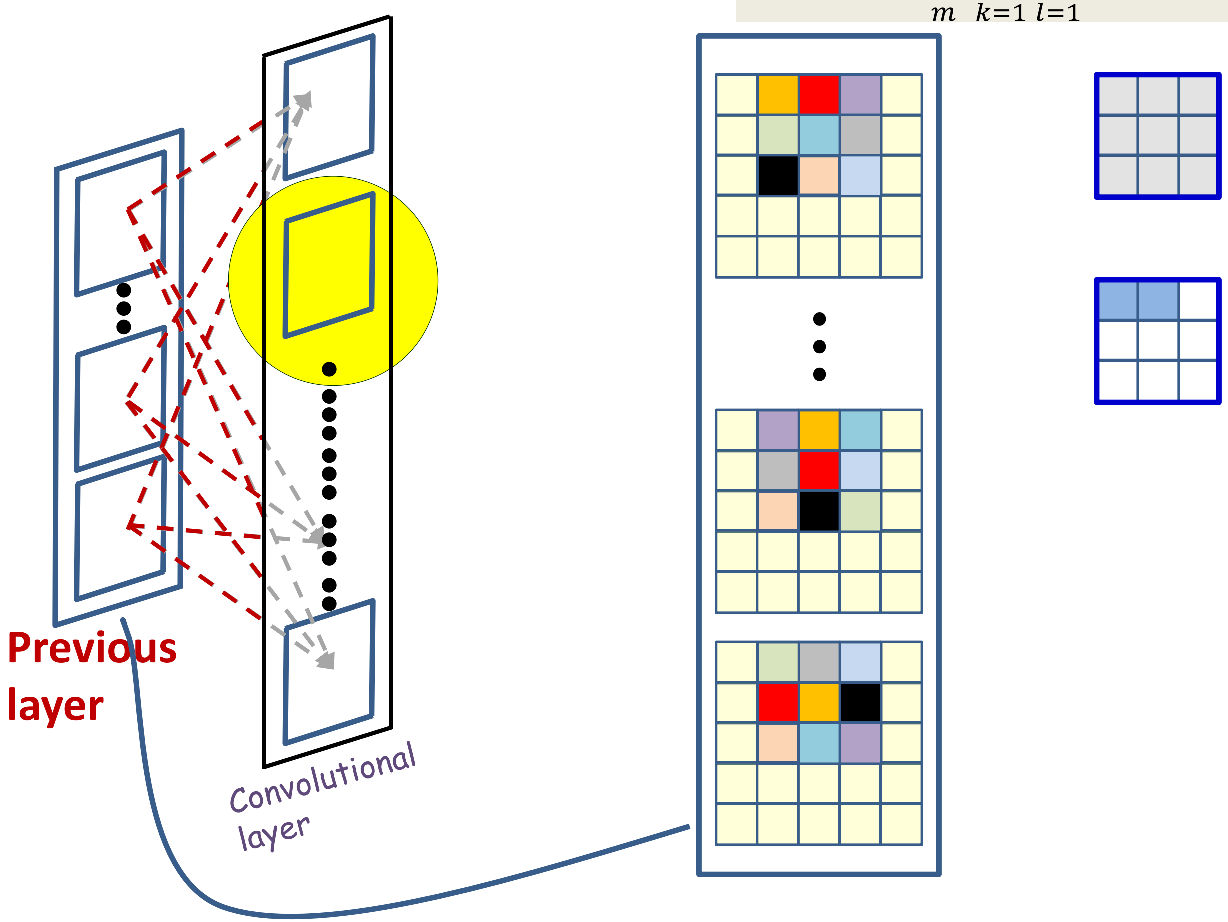
- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

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



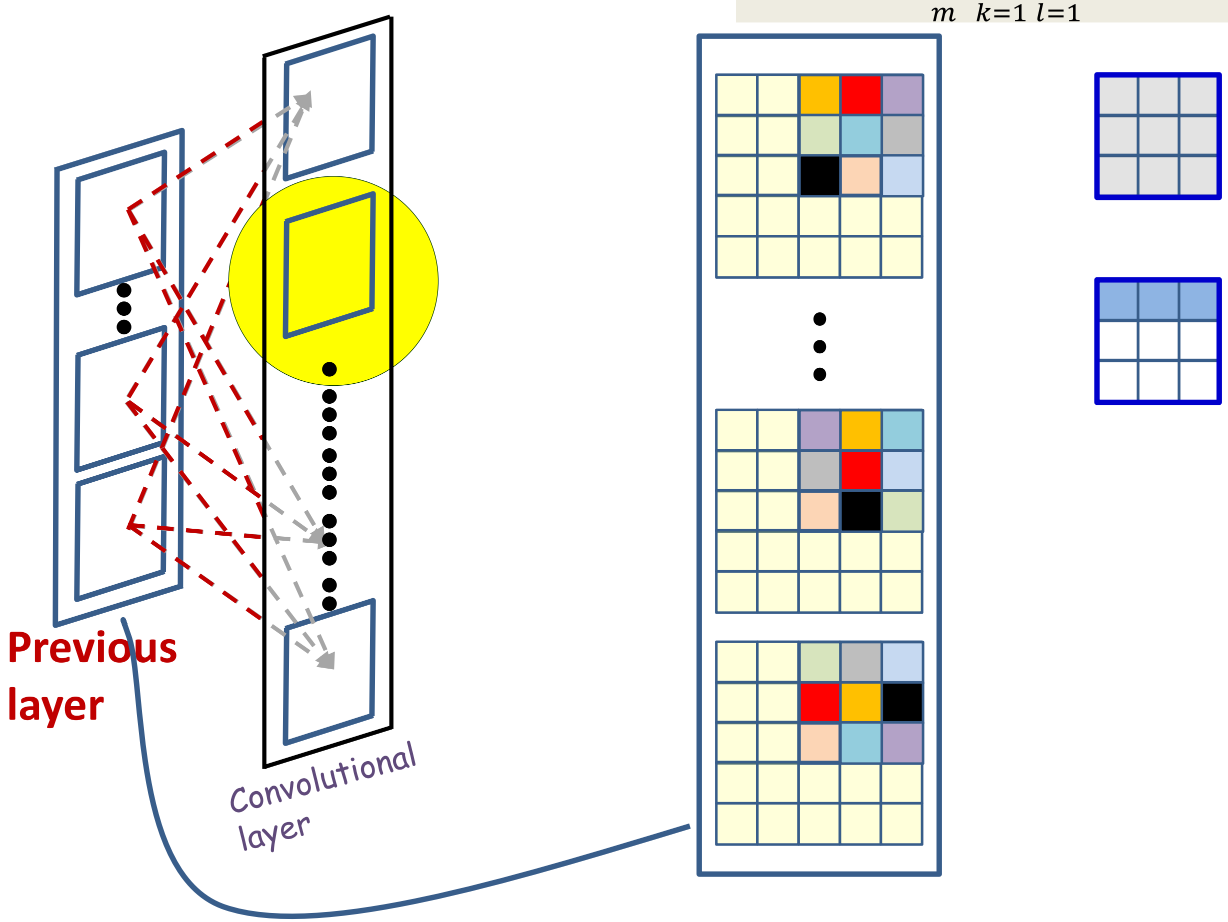
- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

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



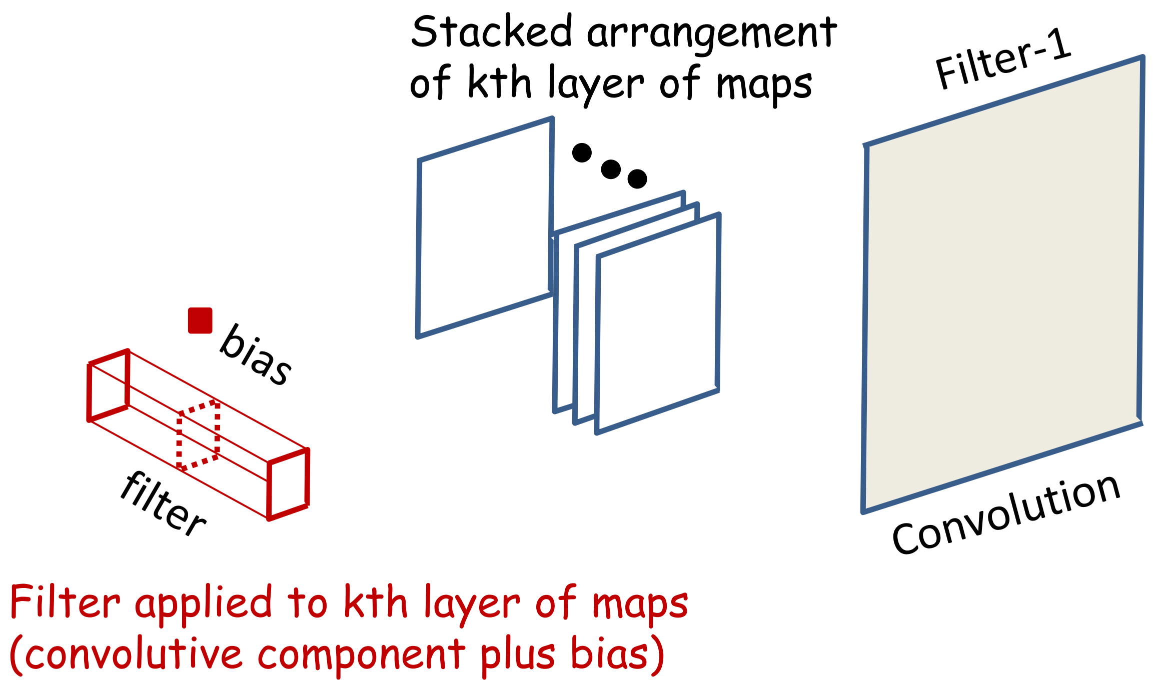
- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

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



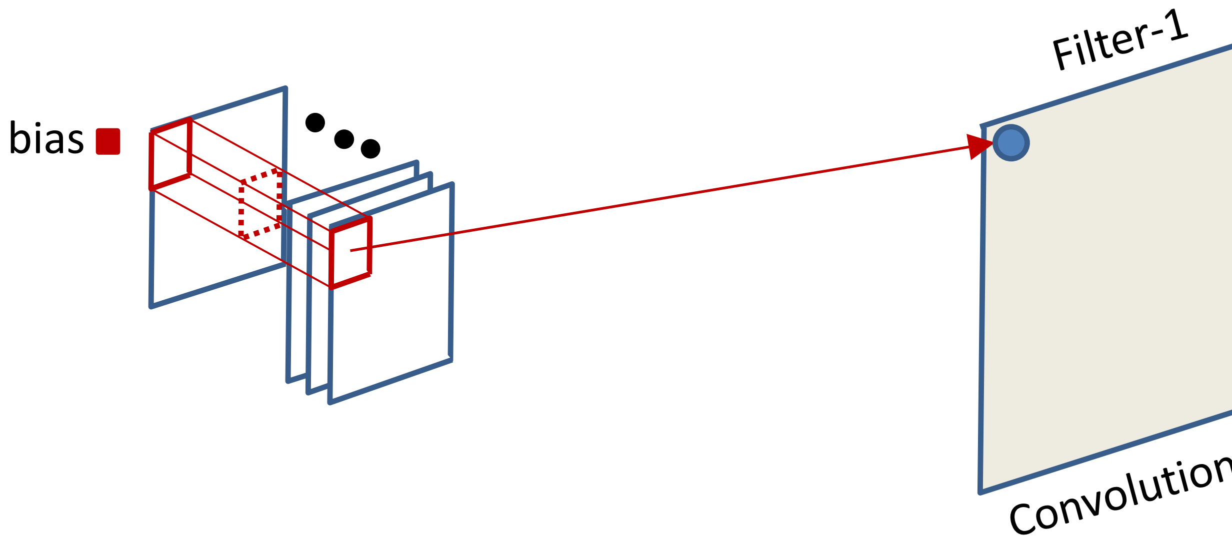
- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as
  size of the filter x *no. of maps in previous layer*

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

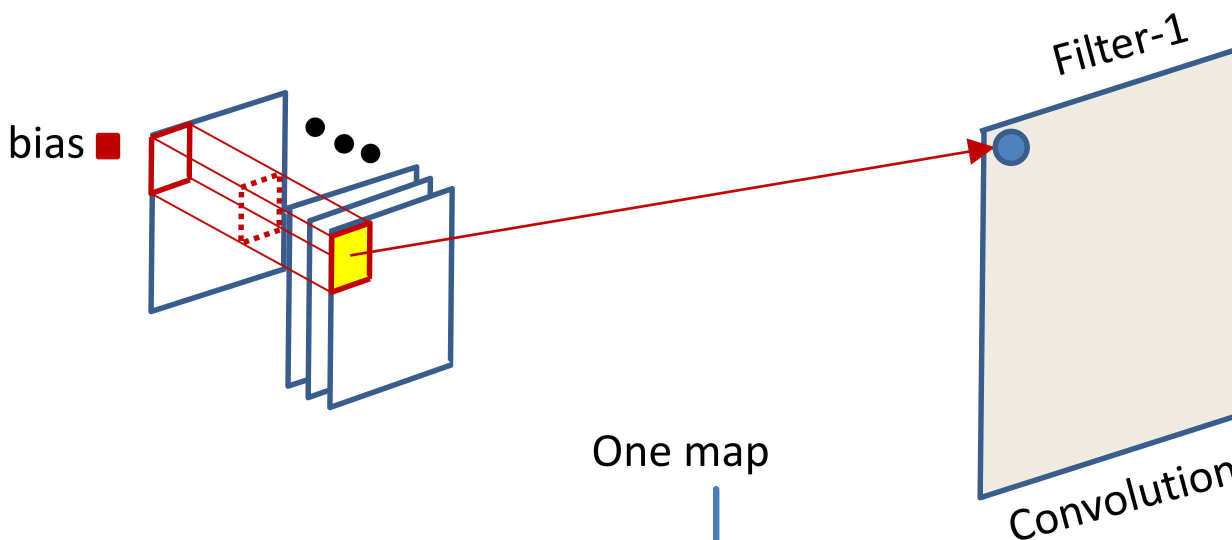
Extending to multiple input maps



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

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

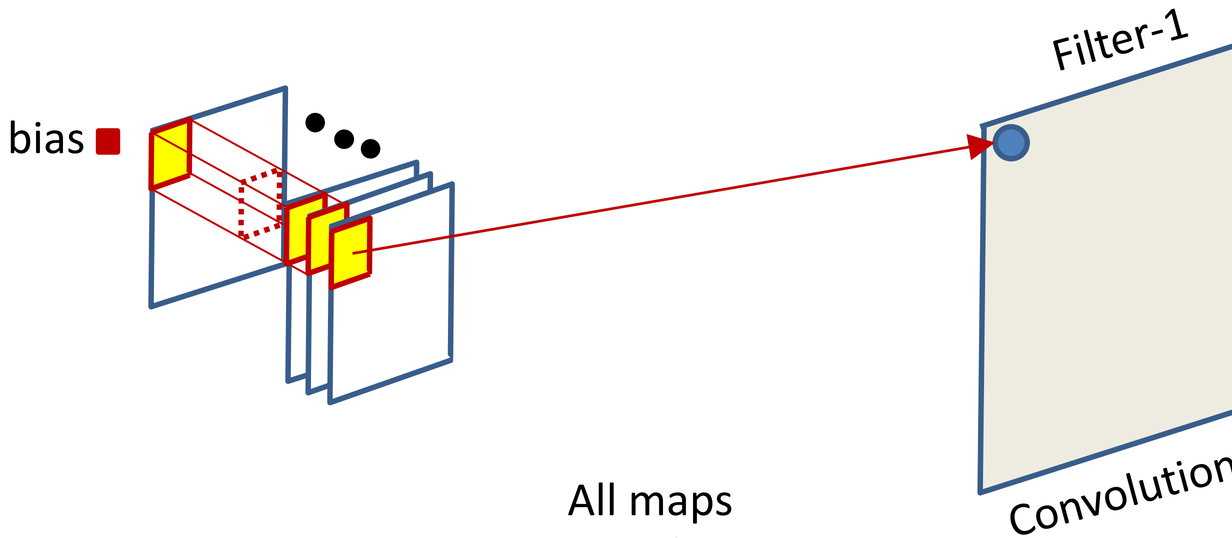
Extending to multiple input maps



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

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

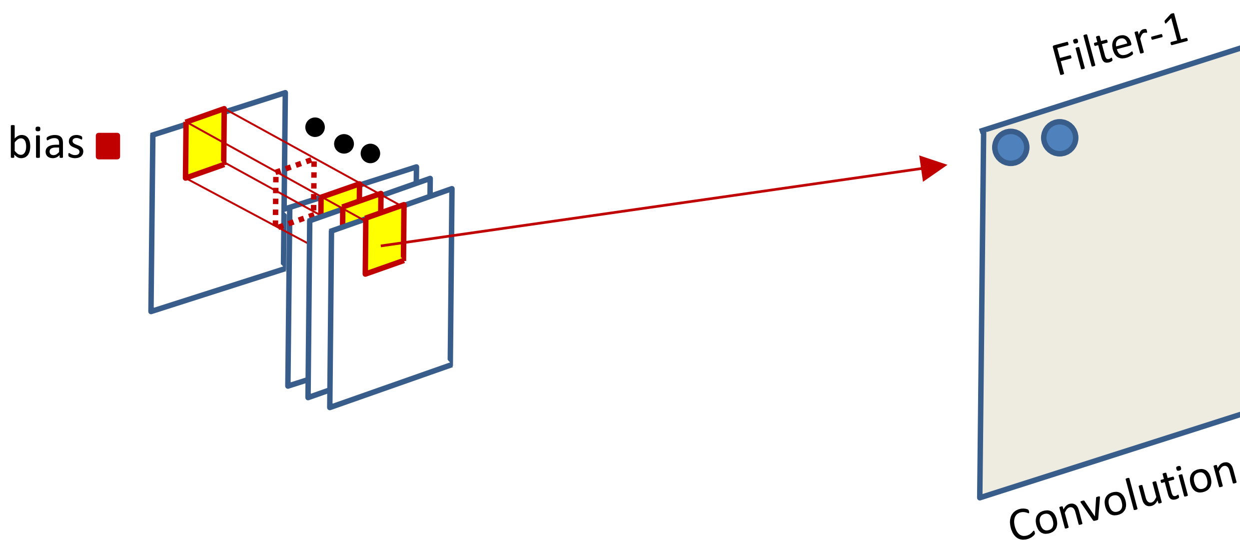
Extending to multiple input maps



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

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

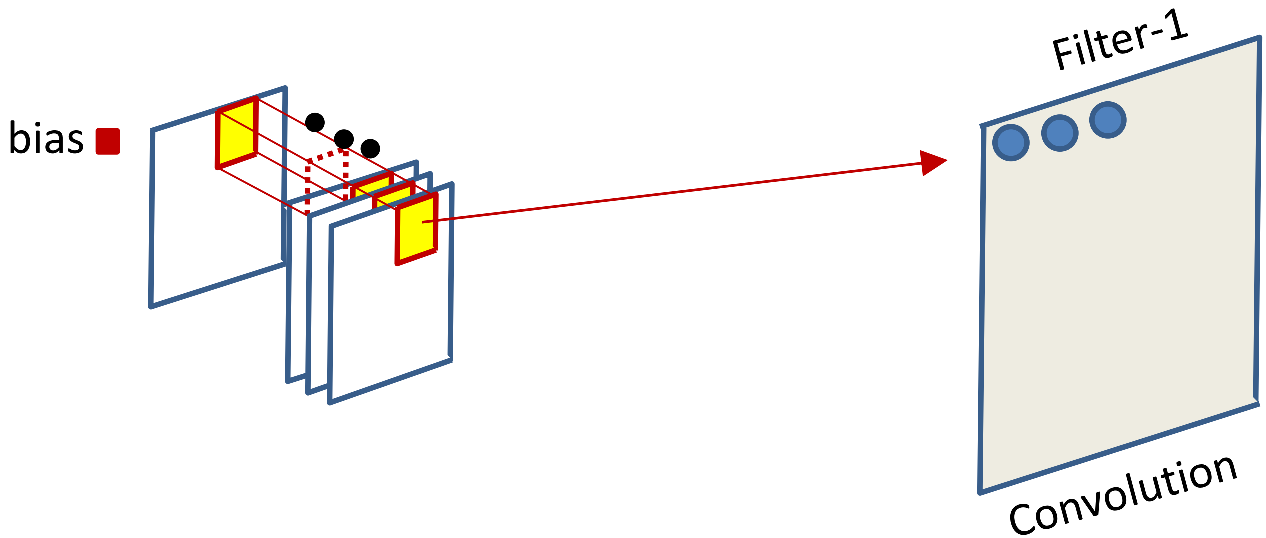
Extending to multiple input maps



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

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

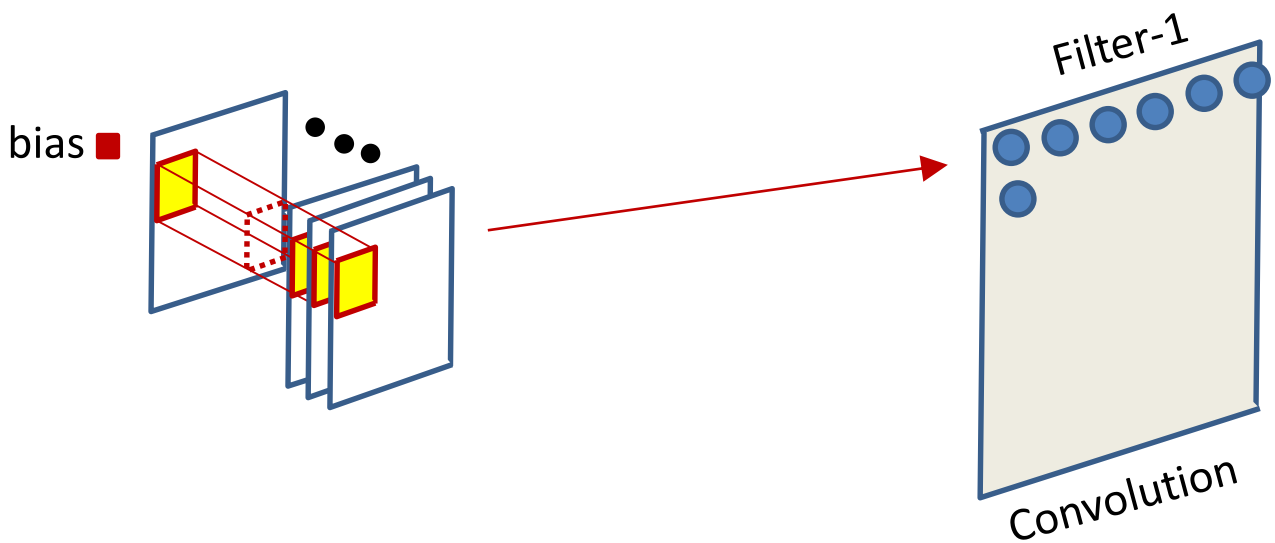
Extending to multiple input maps



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

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

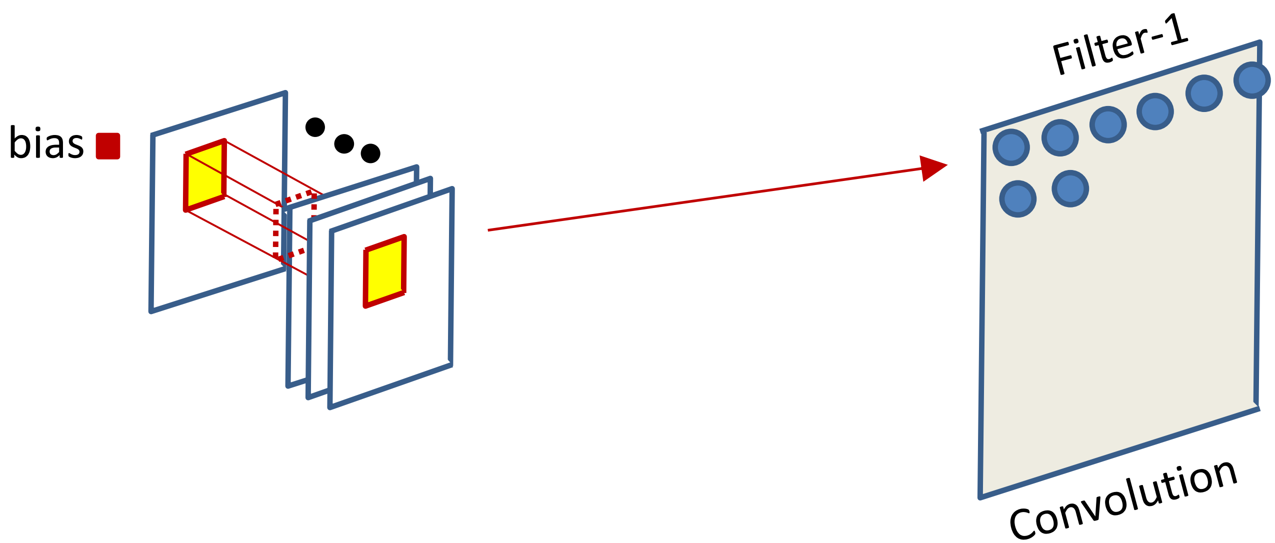
Extending to multiple input maps



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

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

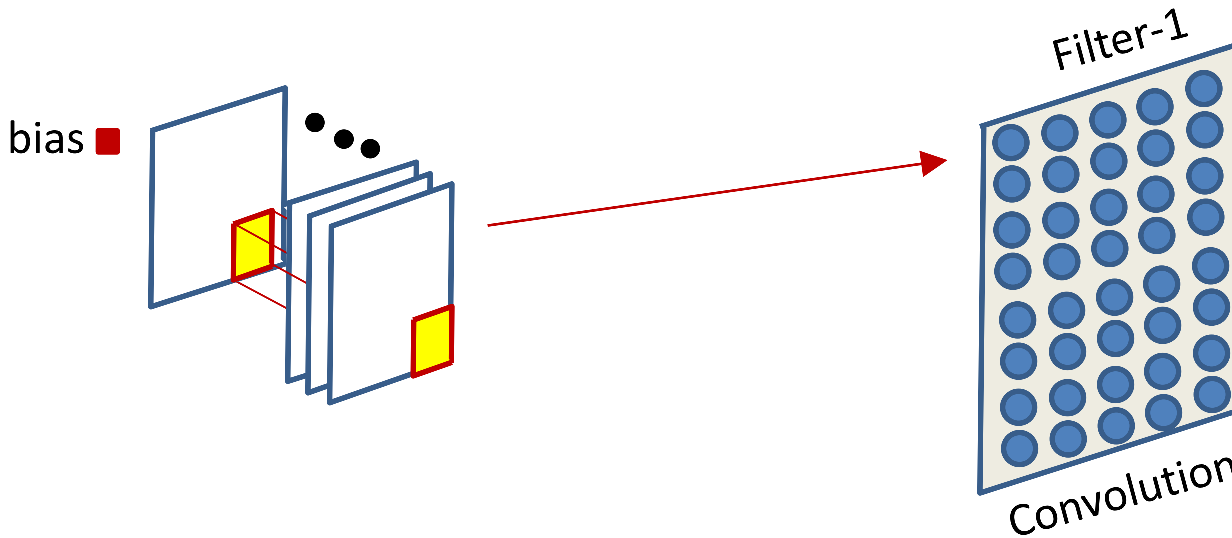
Extending to multiple input maps



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

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

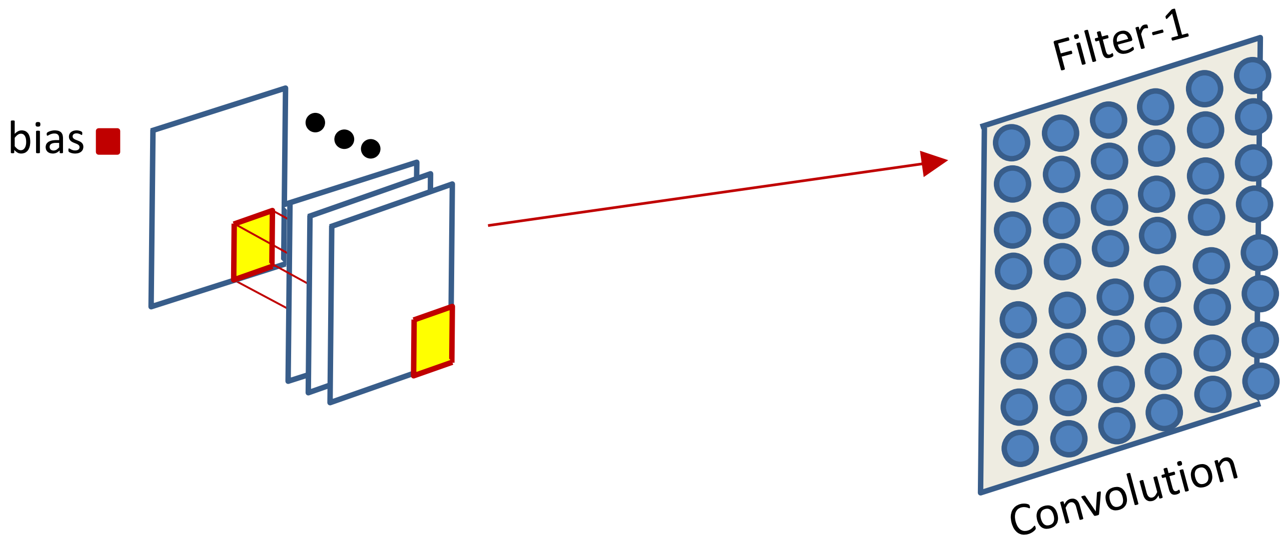
Extending to multiple input maps



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

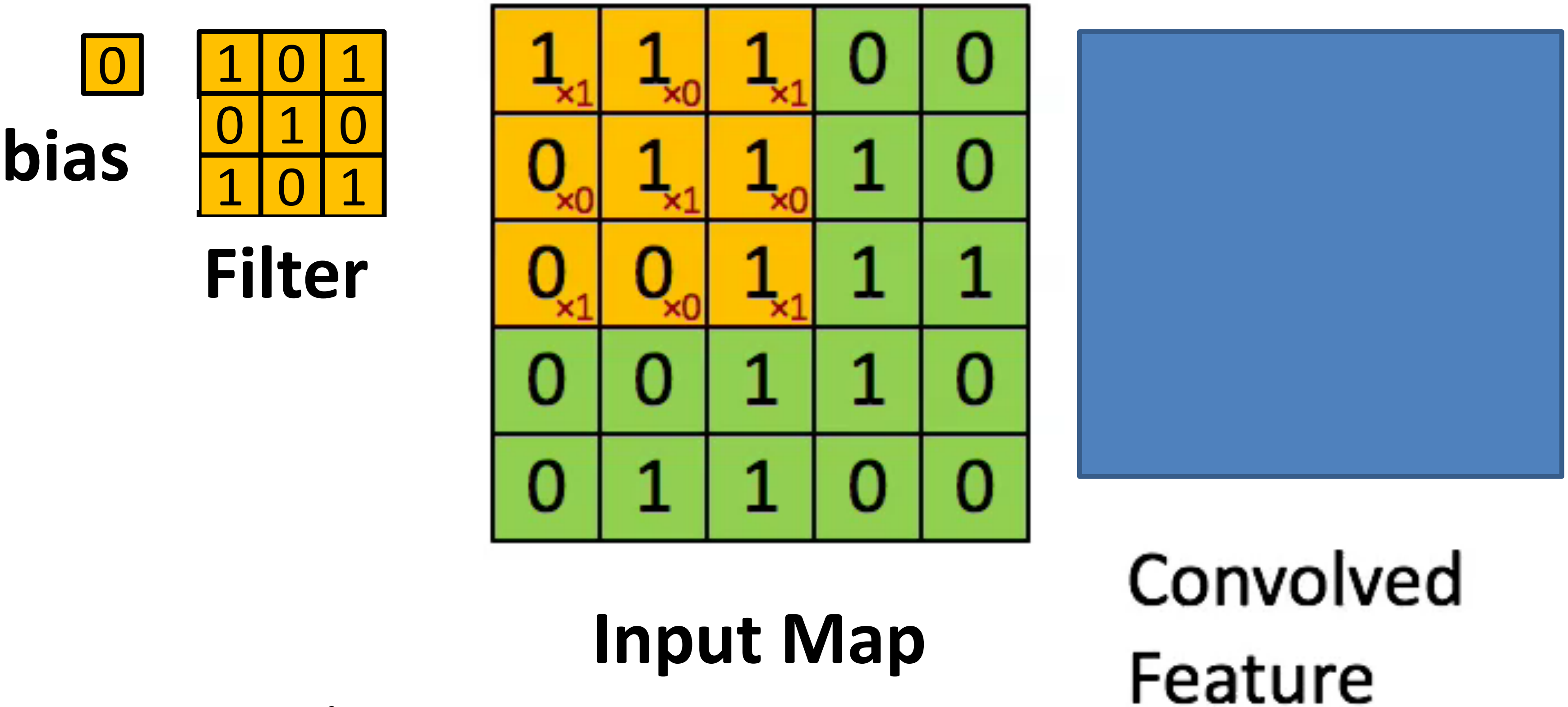
- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

Engineering consideration: The size of the result of the convolution



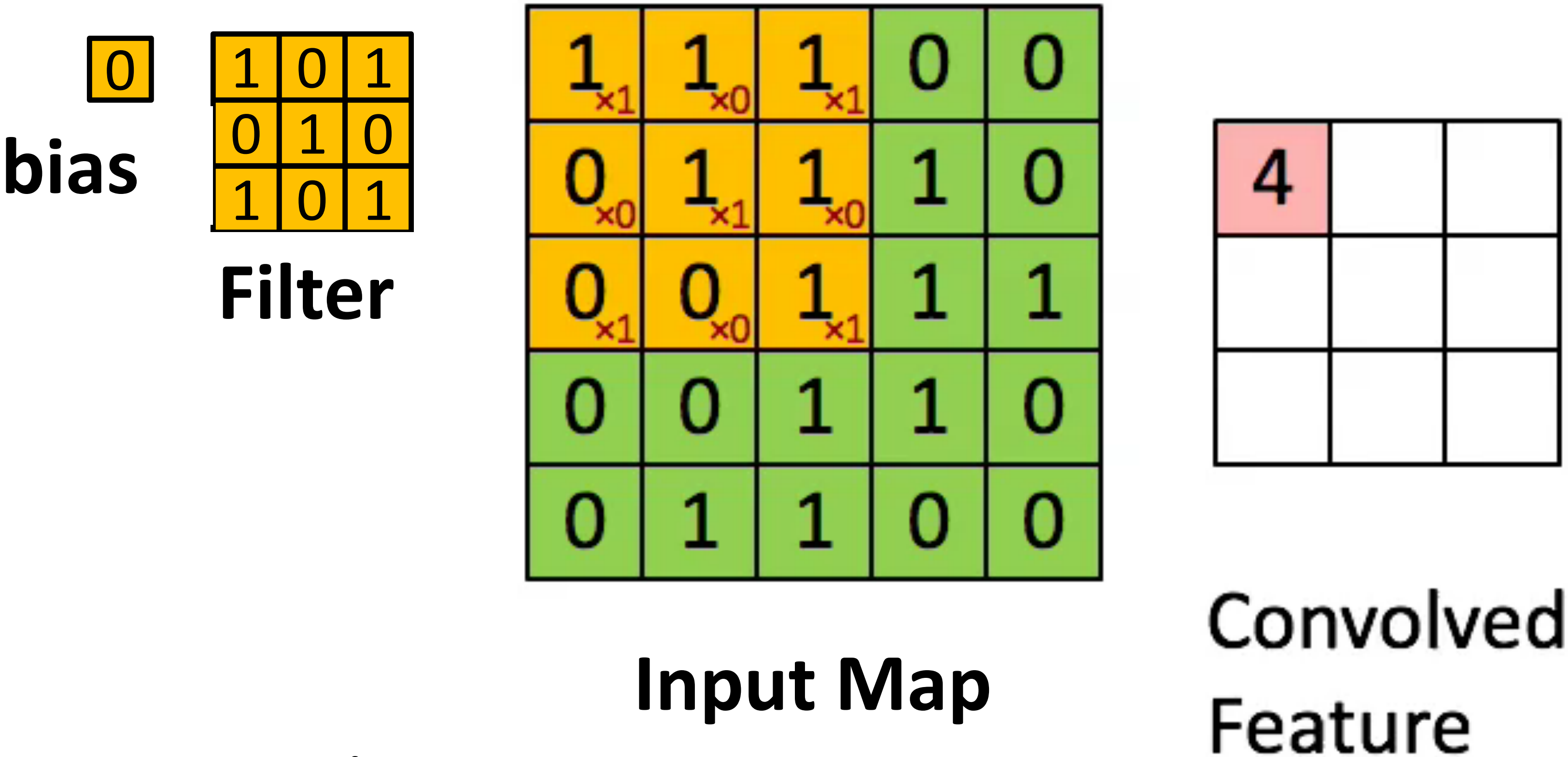
- **Recall**: the "stride" of the convolution may not be one pixel
  - I.e. the scanning neuron may "stride" more than one pixel at a time
- The size of the output of the convolution operation depends on implementation factors
  - And may not be identical to the size of the input
  - Lets take a brief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

The size of the convolution



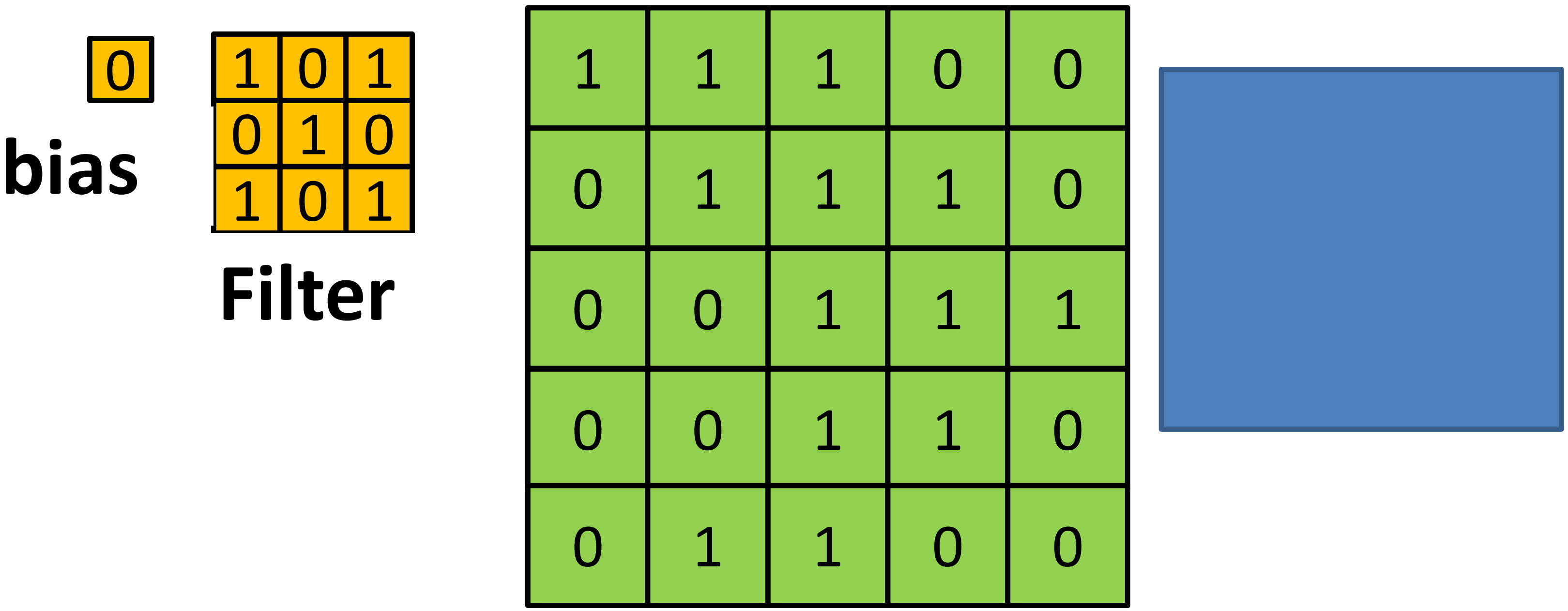
- Image size: 5x5
- Filter: 3x3
- “Stride”: 1
- Output size = ?

The size of the convolution



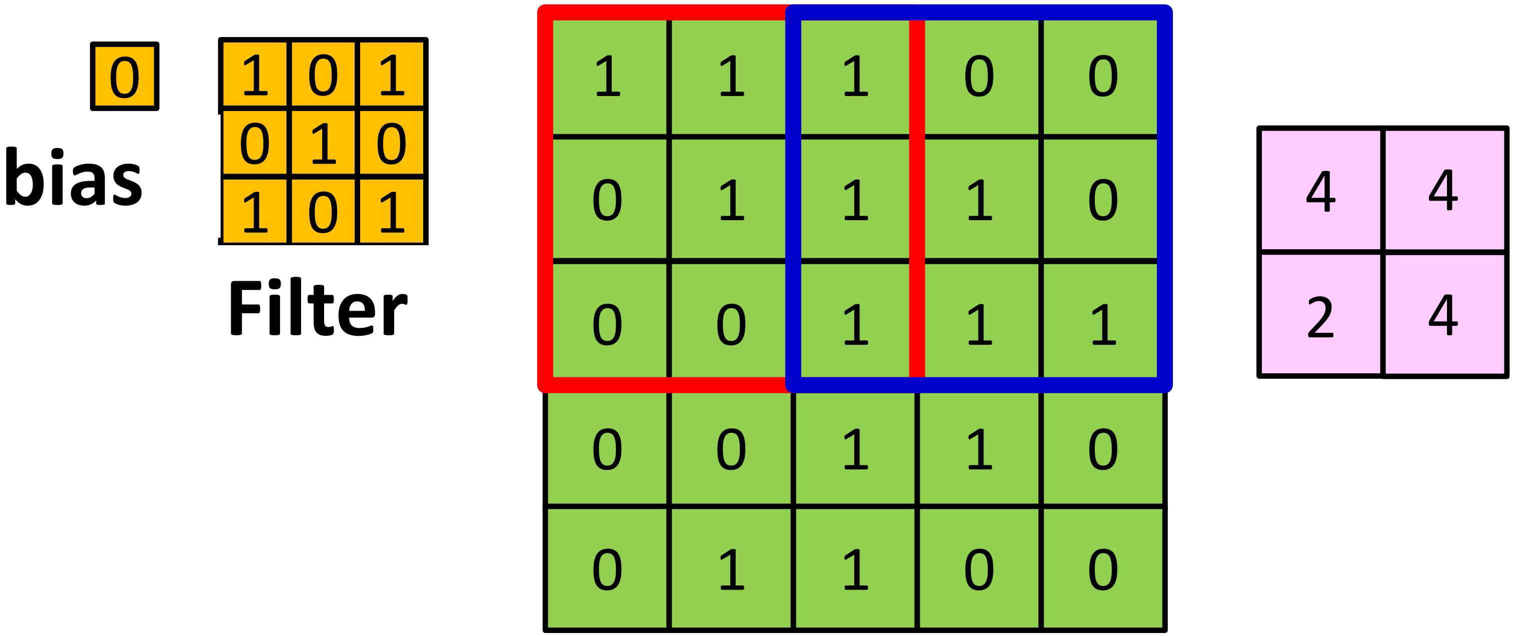
- Image size: 5x5
- Filter: 3x3
- Stride: 1
- Output size = ?

The size of the convolution



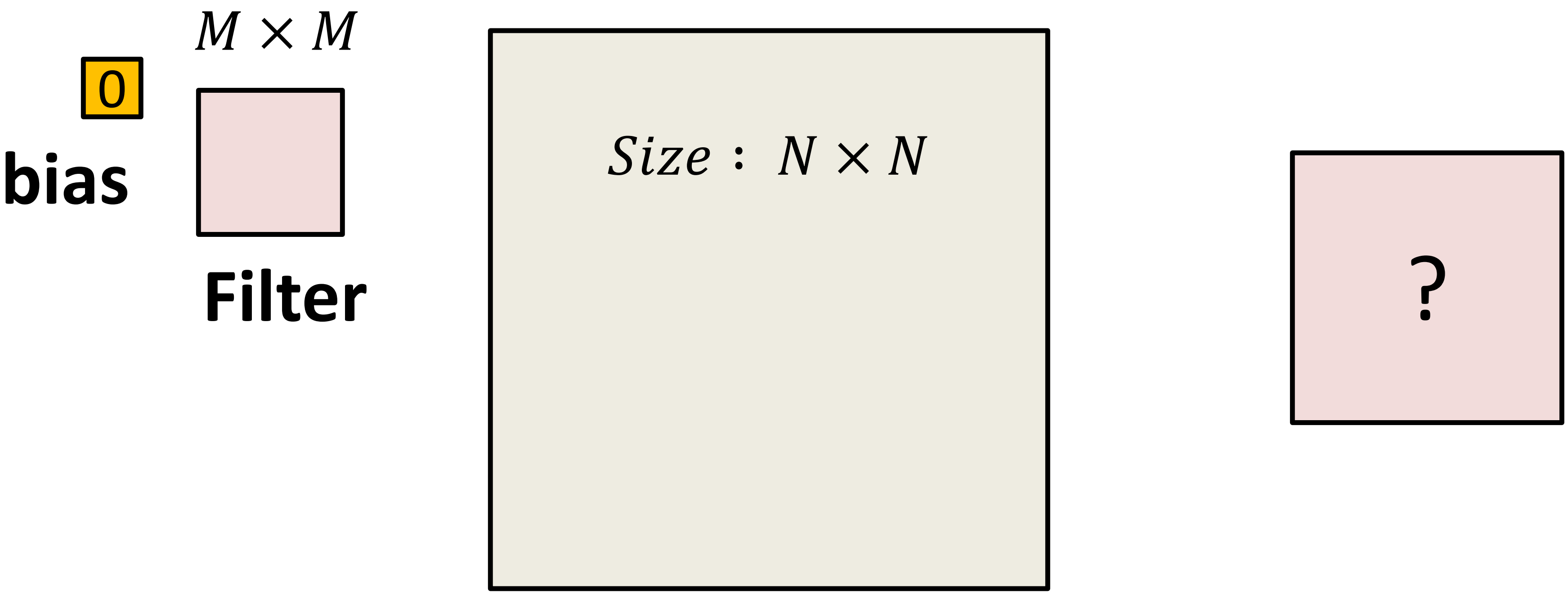
- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

The size of the convolution



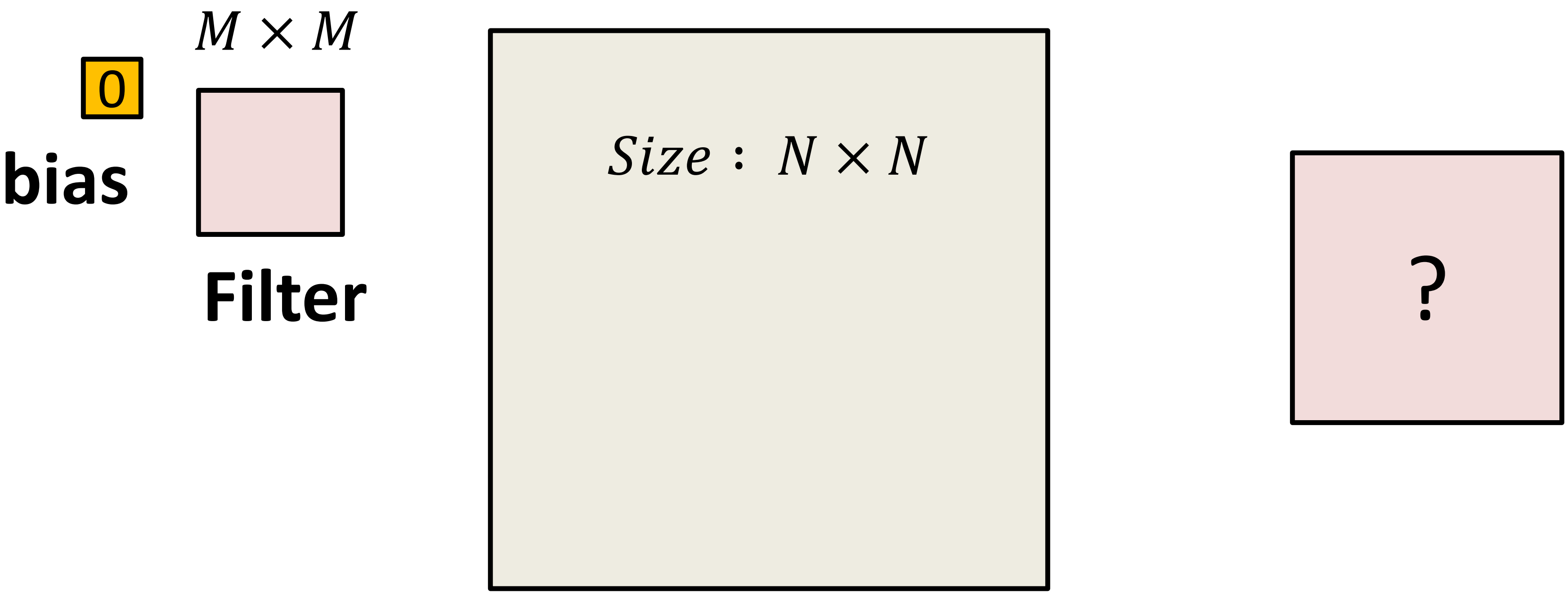
- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

The size of the convolution



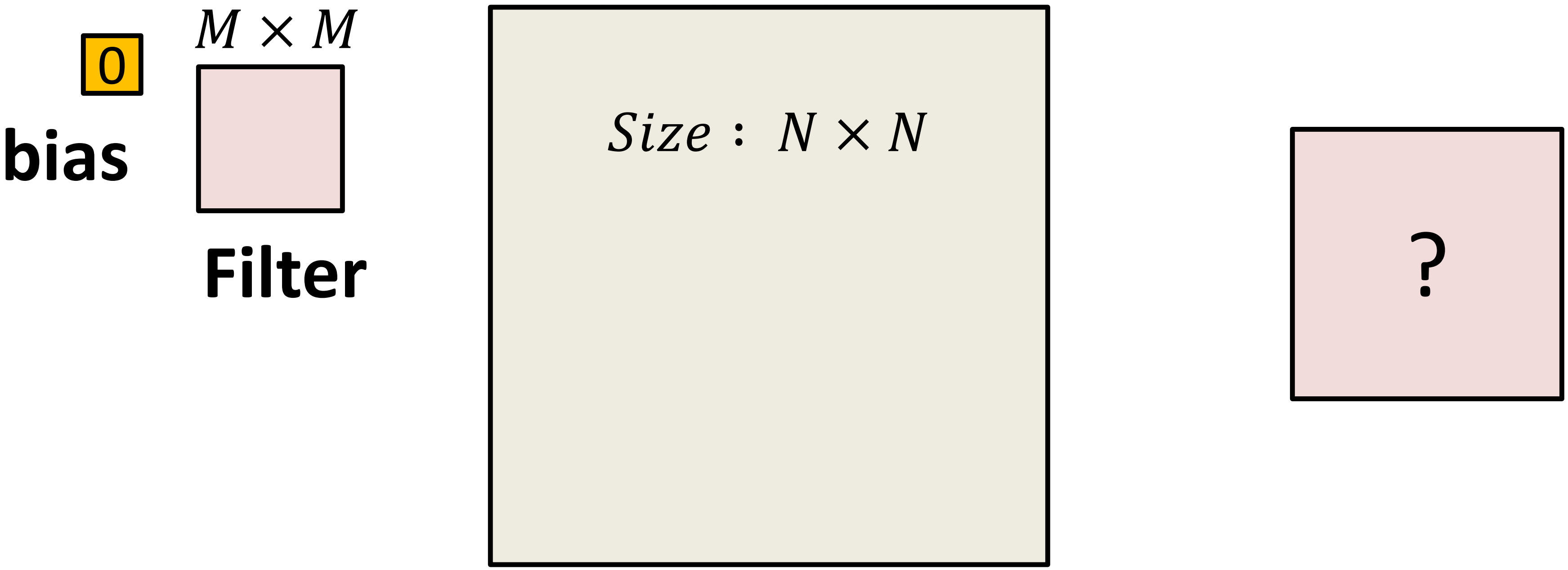
- Image size: $N \times N$
- Filter: $M \times M$
- Stride: 1
- Output size = ?

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

bias: 0

Filter:

1	0	1
0	1	0
1	0	1

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
    - 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

bias

0

Filter

1	0	1
0	1	0
1	0	1

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..
  - **For stride 1, the result of the convolution is the same size as the original image**

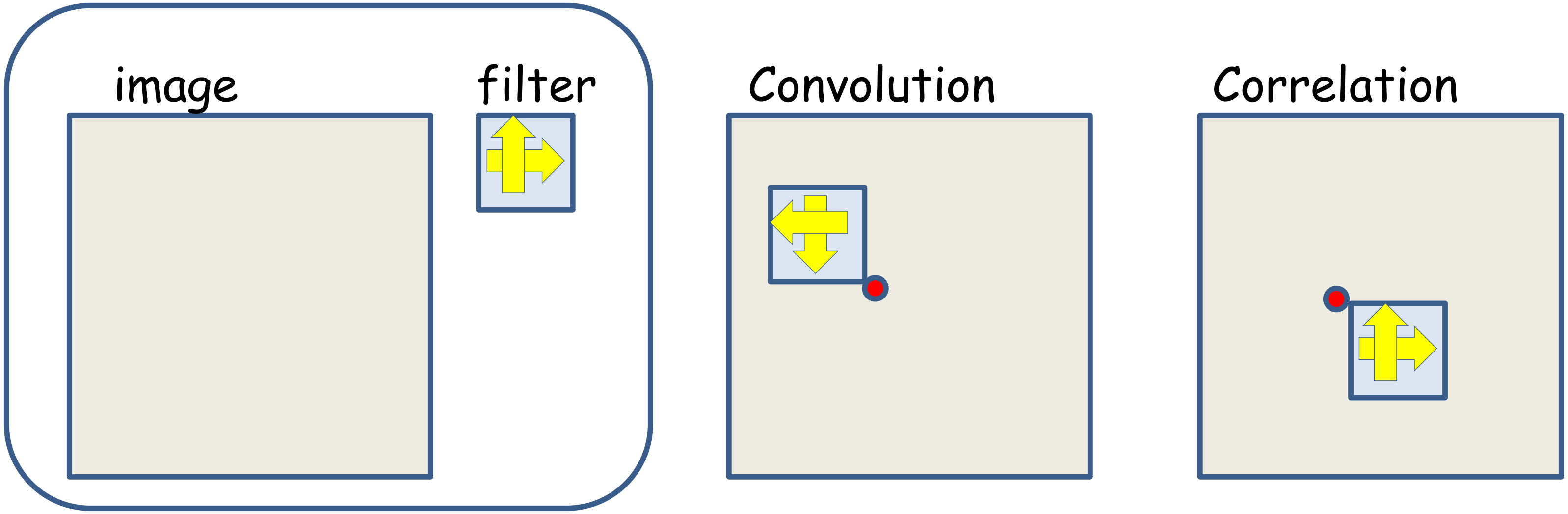
Zero padding

- 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

Correlation, not Convolution



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

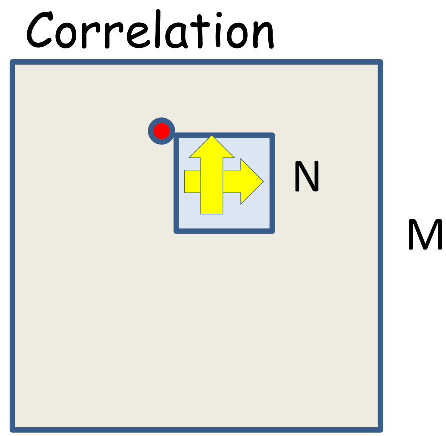
$$y(i,j) = \sum_{l} \sum_{m} x(i+l, j+m) w(l,m)$$

  - 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

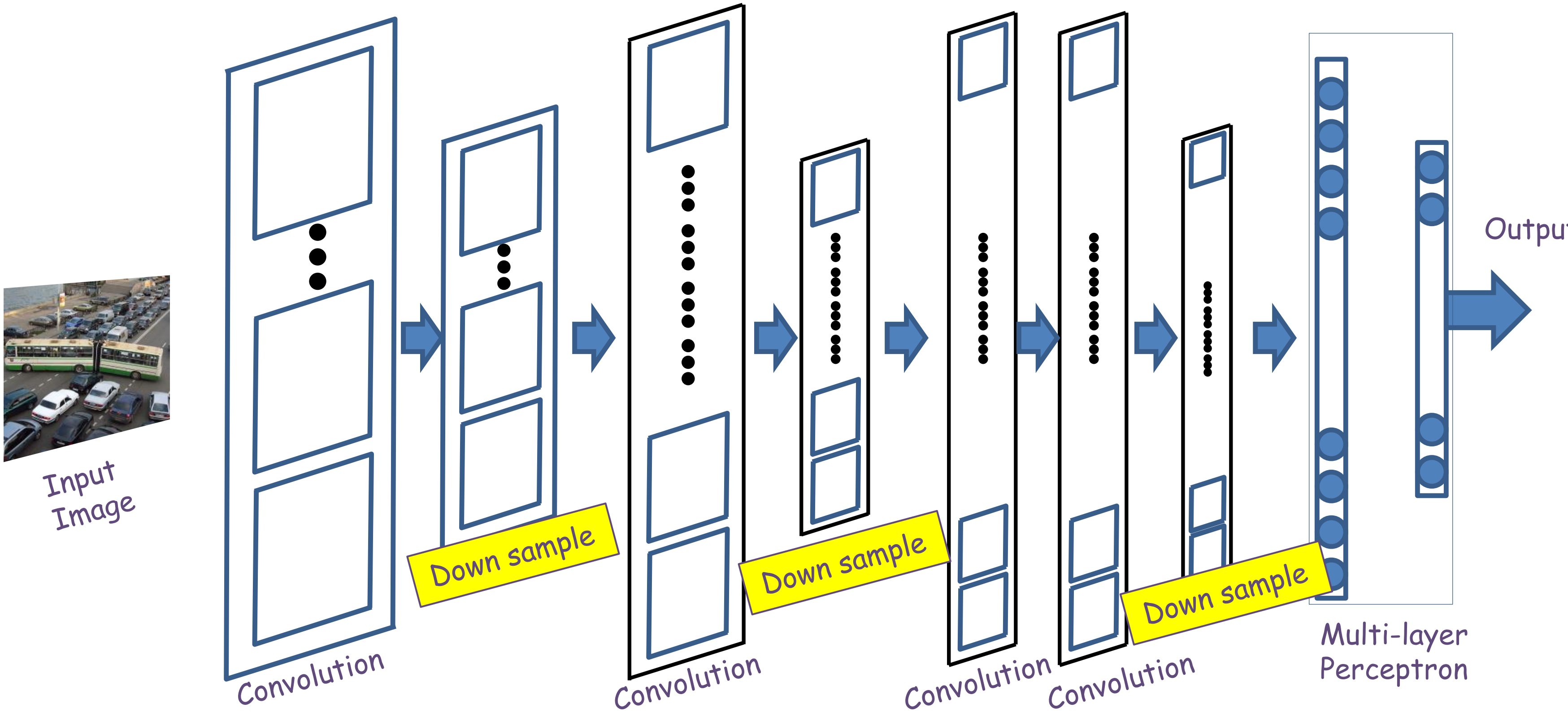


- **Correlation**:

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

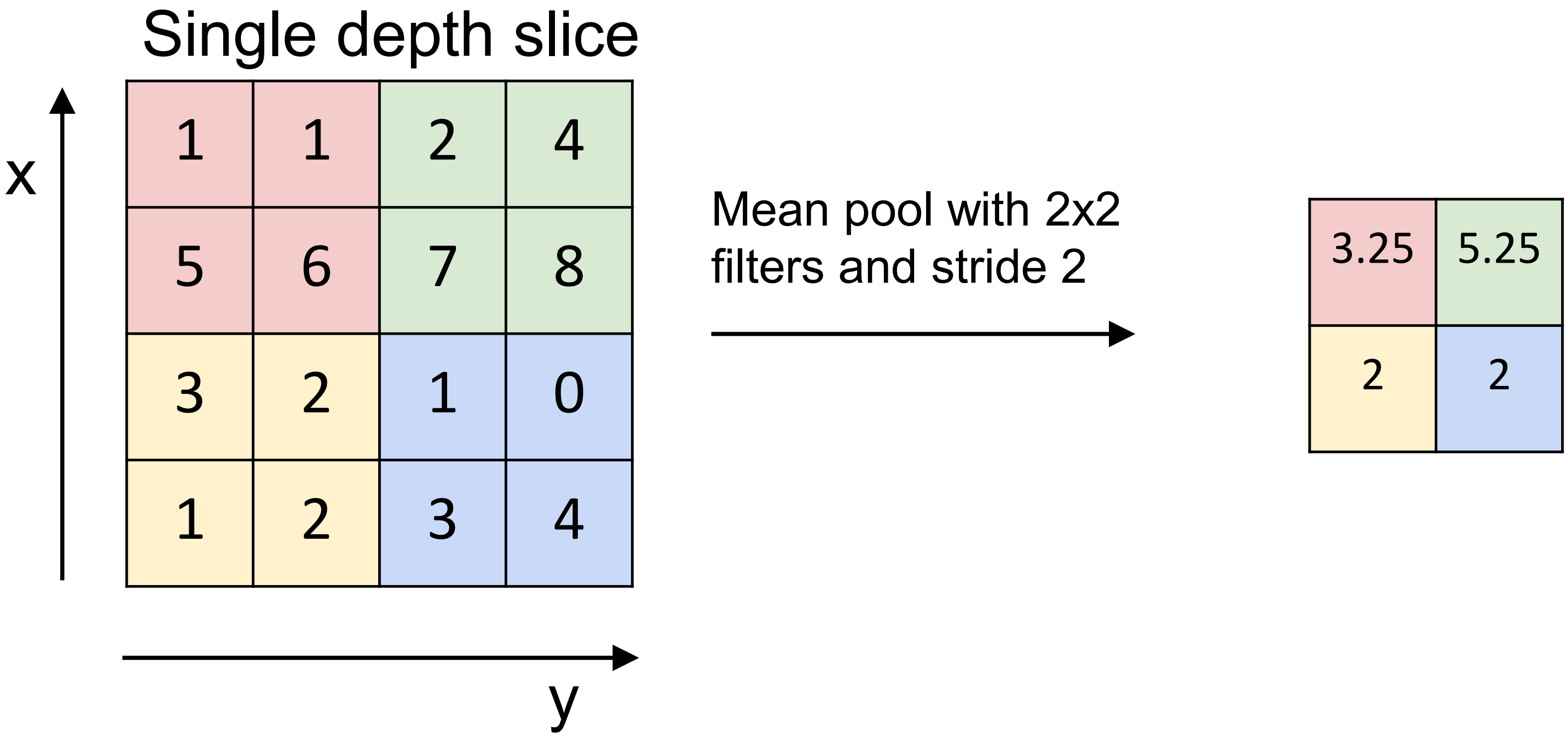
- Cost of scanning an $M \times M$ image with an $N \times N$ filter: $O(M^2N^2)$
  - N^2 multiplications at each of M^2 positions
    - Not counting boundary effects
  - Expensive, for large filters

The other component
Downsampling/Pooling



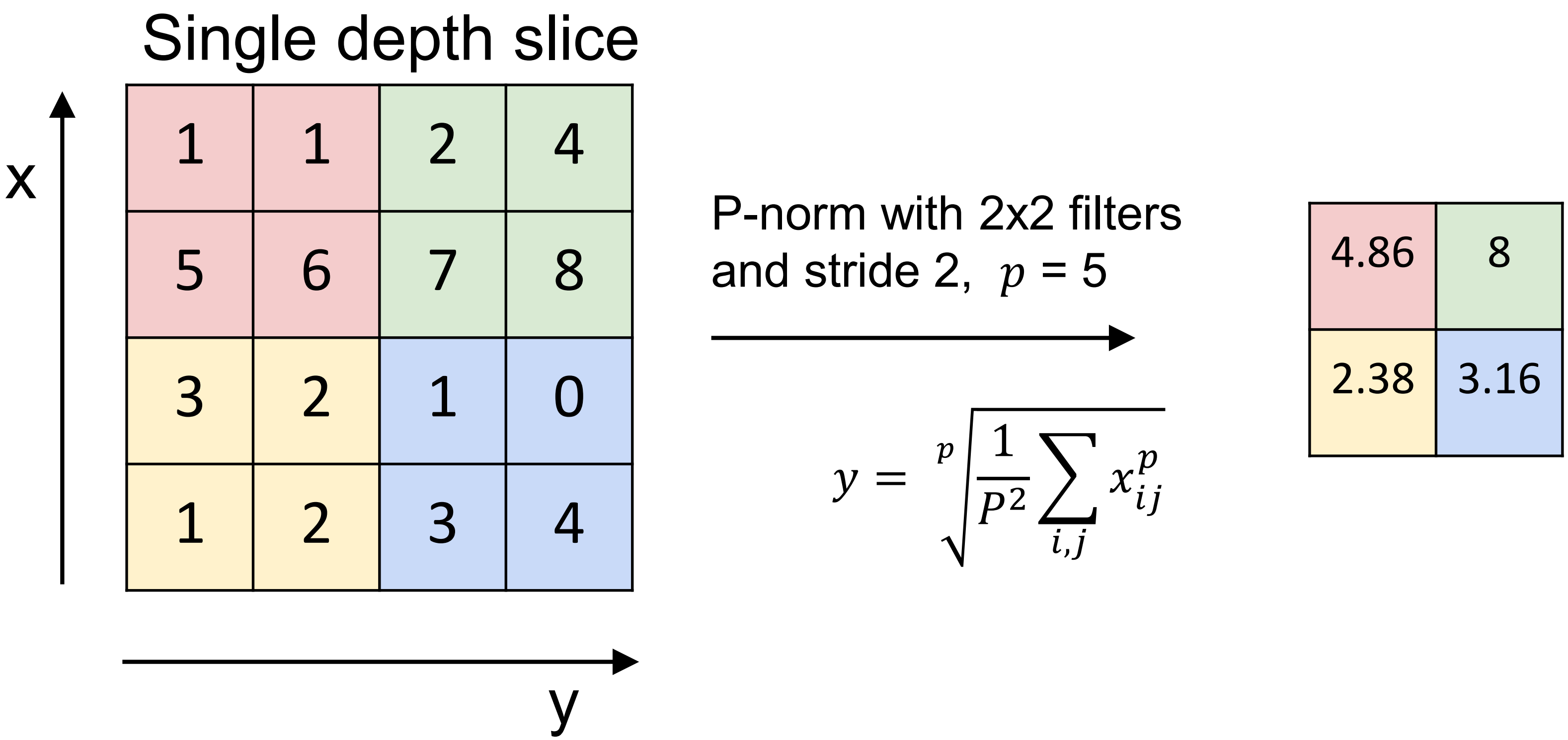
- Convolution (and activation) layers are followed intermittently by "downsampling" (or "pooling") layers
  - Often, they alternate with convolution, though this is not necessary

Alternative to Max pooling: Mean Pooling



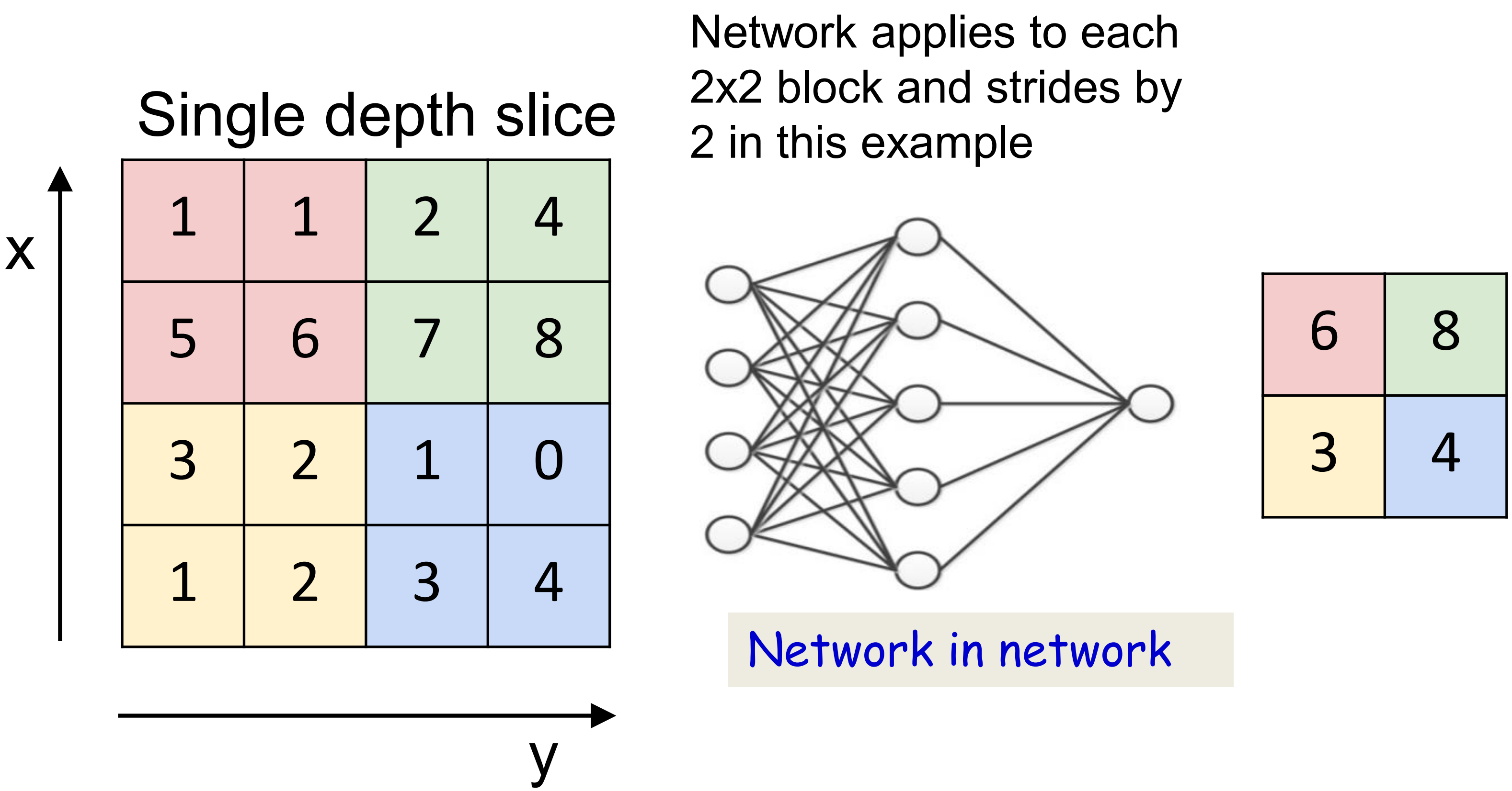
- Compute the mean of the pool, instead of the max

Alternative to Max pooling: P-norm



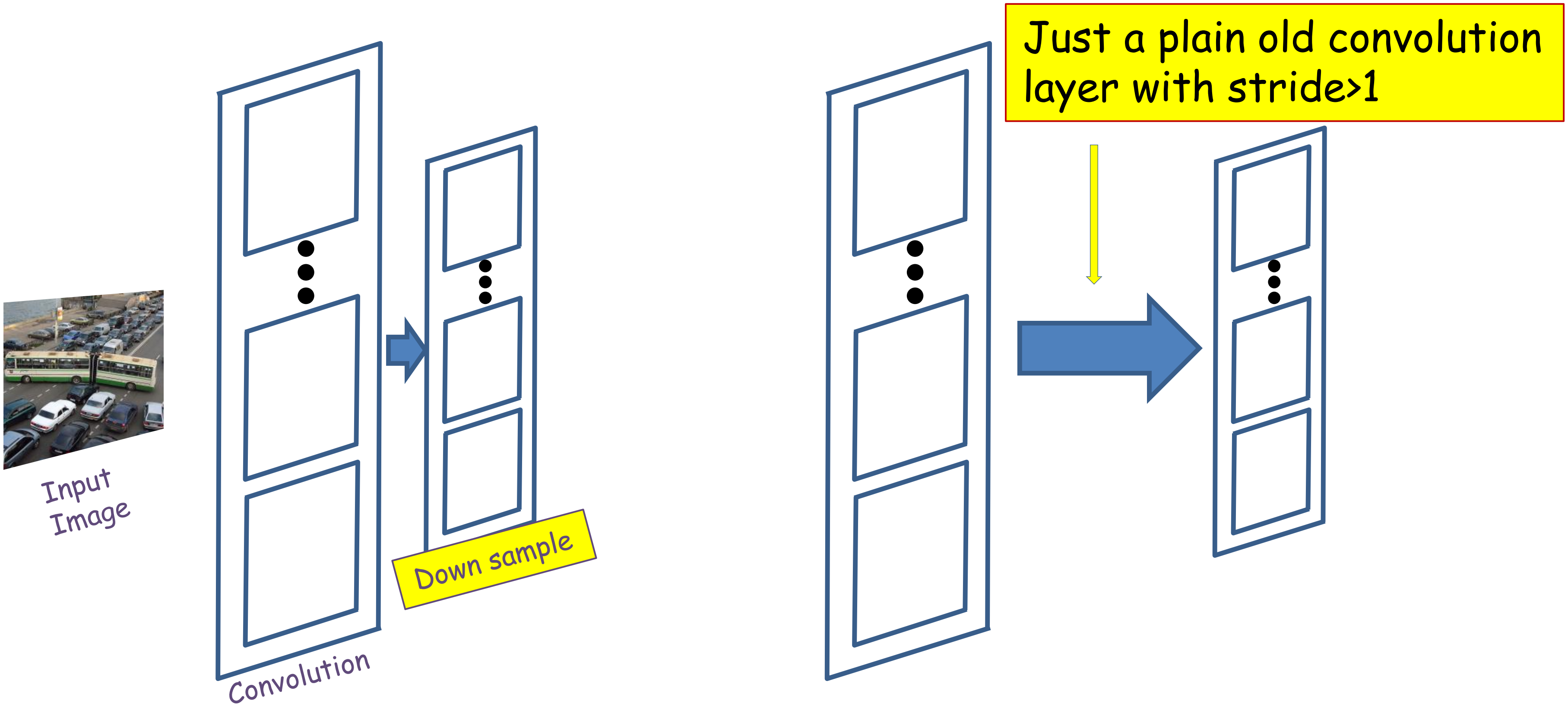
- Compute a p-norm of the pool

Other options



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

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

Setting everything together

- Typical image classification task
  - Assuming maxpooling..

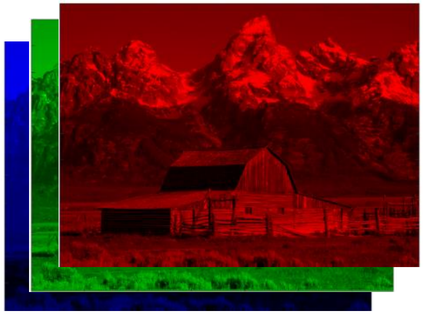
Convolutional Neural Networks

- Input: 1 or 3 images
  - Black and white or color
  - Will assume color to be generic

Convolutional Neural Networks

- Input: 3 pictures

Convolutional Neural Networks

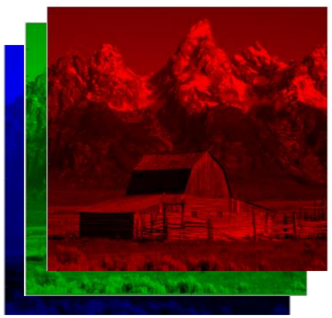


- Input: 3 pictures

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

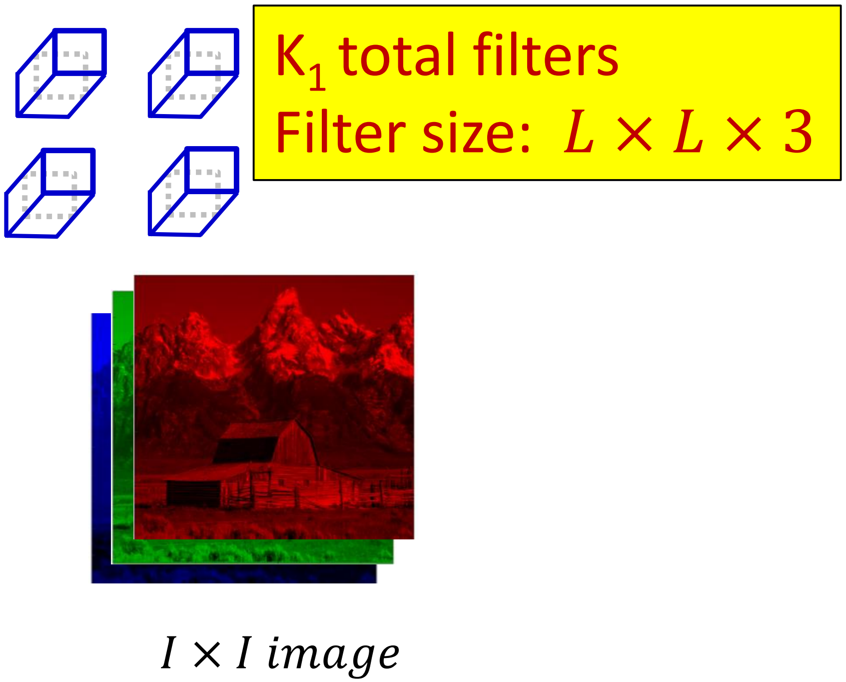
Convolutional Neural Networks



$I \times I$ *image*

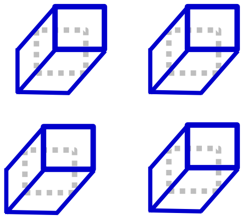
- Input: 3 pictures

Convolutional Neural Networks



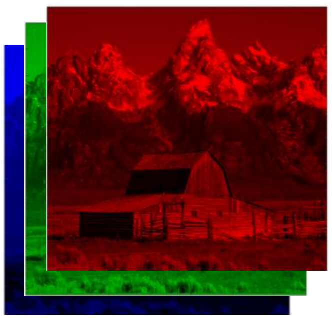
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

Convolutional Neural Networks



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

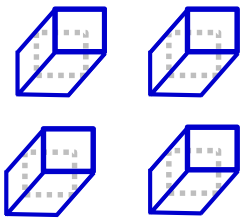
Small enough to capture fine features
(particularly important for scaled-down images)



$I \times I$ *image*

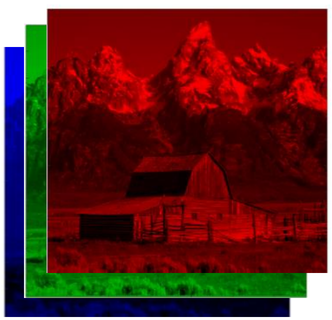
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

Convolutional Neural Networks



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

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

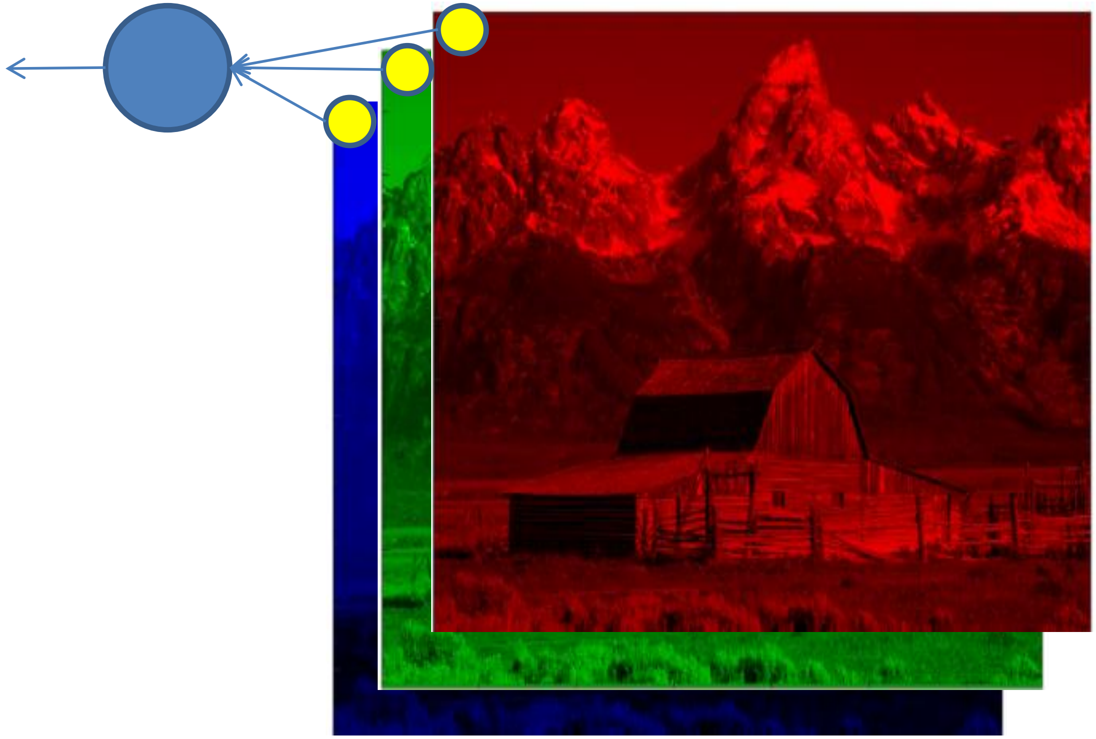


$I \times I$ *image*

What on earth is this?

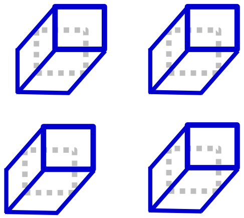
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - Filters are typically 5x5, 3x3, or even 1x1

The 1x1 filter

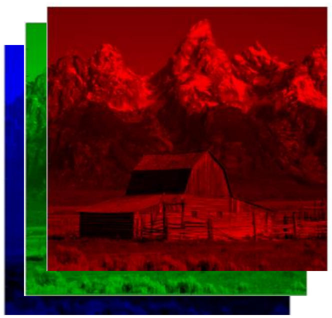


- A 1x1 filter is simply a perceptron that operates over the *depth* of the map, but has no spatial extent
  - Takes one pixel from each of the maps (at a given location) as input

Convolutional Neural Networks



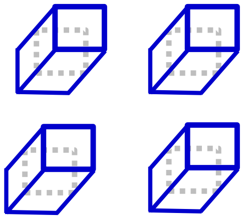
K_1 total filters
Filter size: $L \times L \times 3$



$I \times I$ *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)

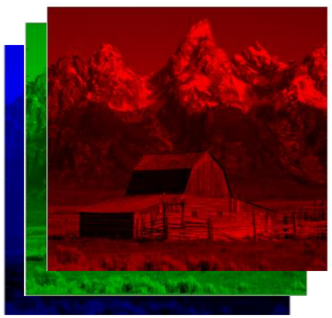
Convolutional Neural Networks



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

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

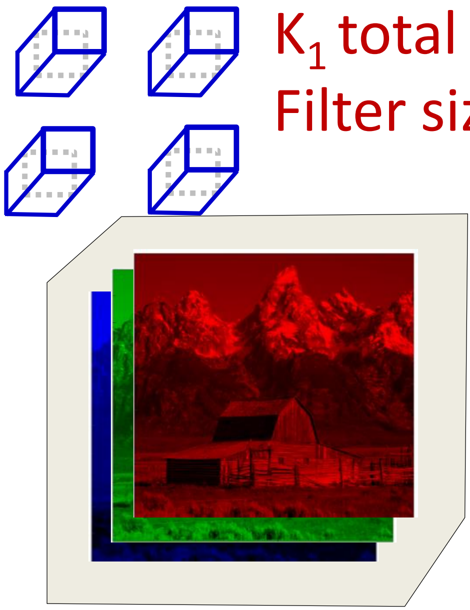


$I \times I$ *image*

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

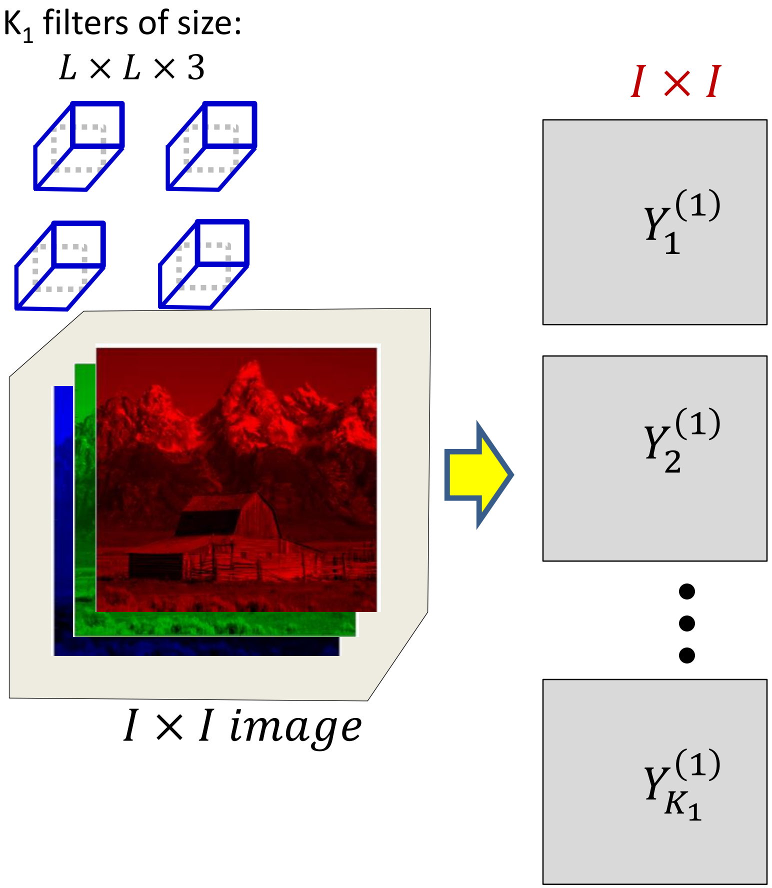
- Input is convolved with a set of K_1 filters
  - Typically K_1 is a power of 2, e.g. 2, 4, 8, 16, 32,..
  - **Better notation:** Filters are typically 5x5(x3), 3x3(x3), or even 1x1(x3)
  - ***Typical stride****:* 1 or 2

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

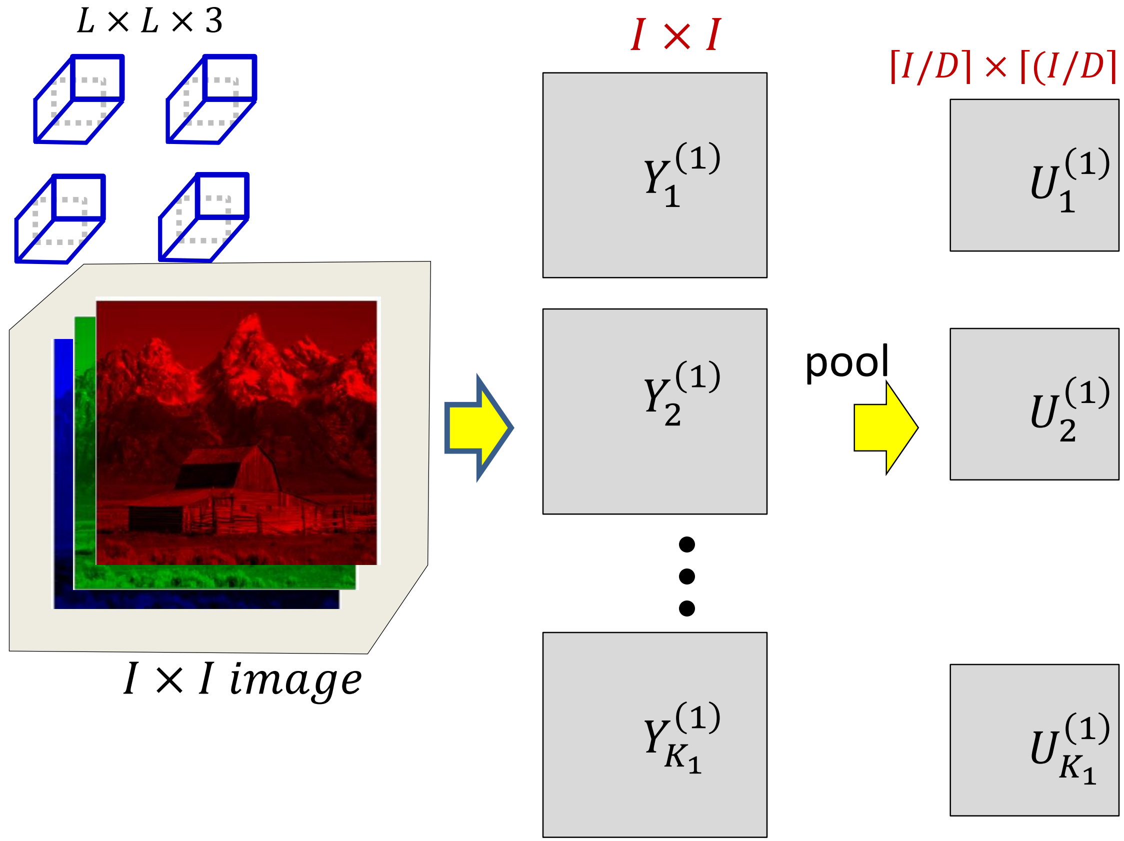
Learnable parameters in the first convolutional layer

- 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

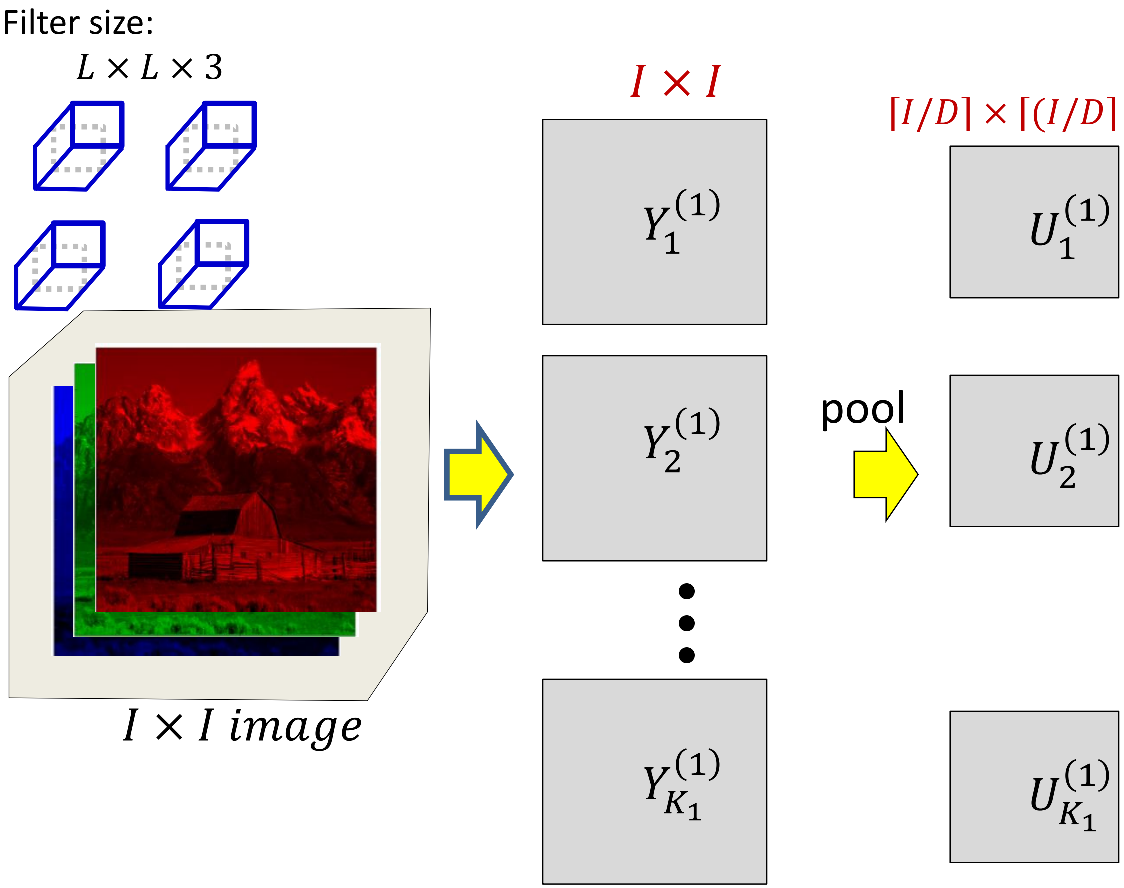


The layer pools PxP blocks of Y into a single value
It employs a stride D between adjacent blocks

$$U_m^{(1)}(i,j) = \max_{\substack{k\in\{(i-1)D+1,\, iD\},\\ l\in\{(j-1)D+1,\, jD\}}} Y_m^{(1)}(k,l)$$

- **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



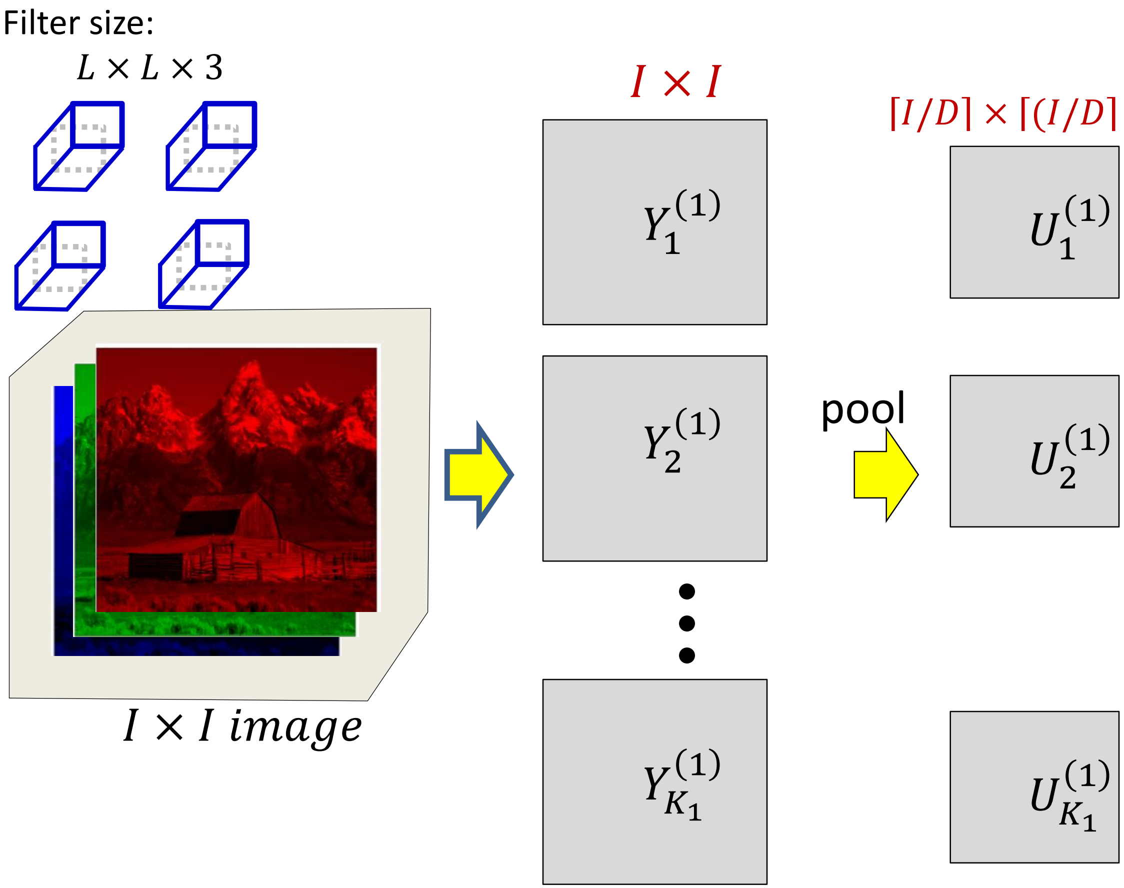
$$U_m^{(1)}(i,j) = \max_{\substack{k \in \{(i-1)D+1,\, iD\},\\ l \in \{(j-1)D+1,\, jD\}}} Y_m^{(1)}(k,l)$$

Parameters to choose:
- Size of pooling block P
- Pooling stride D

Choices: Max pooling or mean pooling?
Or learned pooling?

- **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

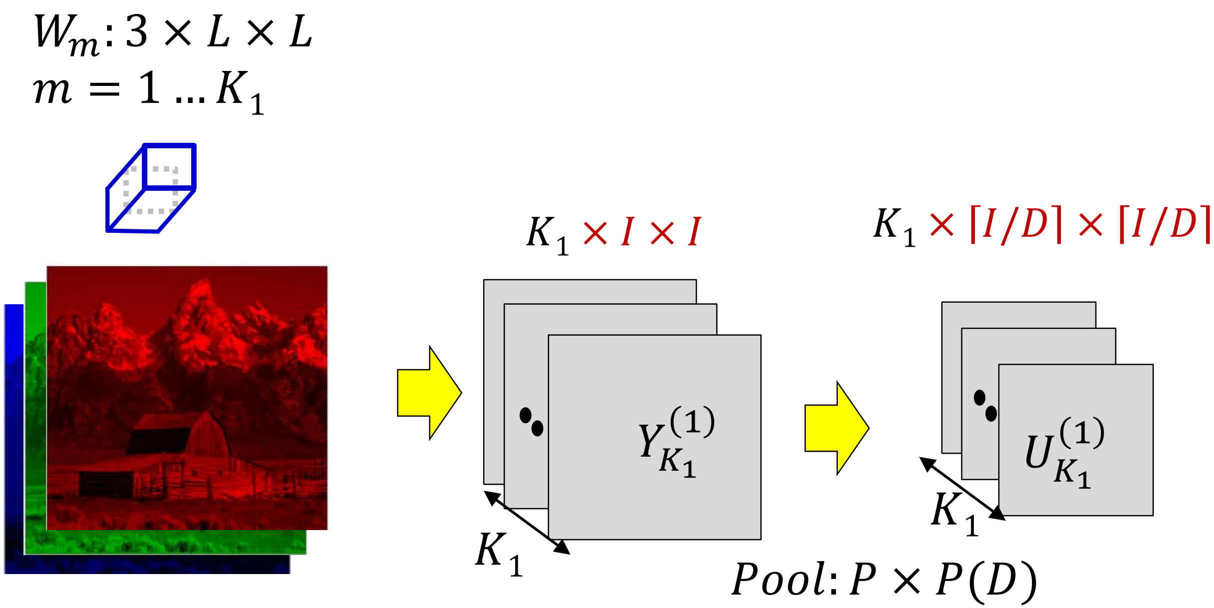


$$P_m^{(1)}(i,j) = \underset{\substack{k\in\{(i-1)D+1,\ iD\},\\ l\in\{(j-1)D+1,\ jD\}}}{\operatorname{argmax}} Y_m^{(1)}(k,l)$$

$$U_m^{(1)}(i,j) = Y_m^{(1)}(P_m^{(1)}(i,j))$$

- **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

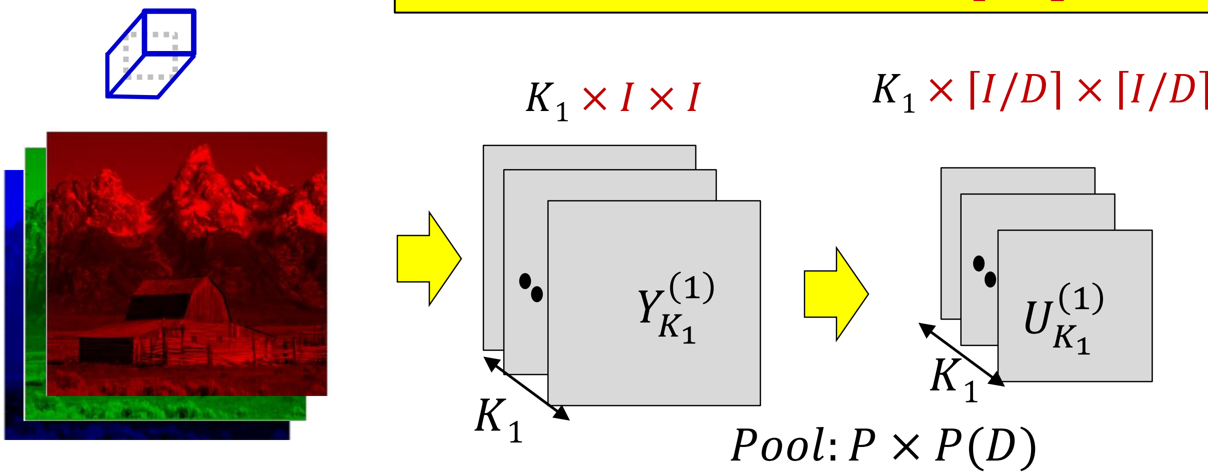


- **First pooling layer:** Drawing it differently for convenience

Convolutional Neural Networks

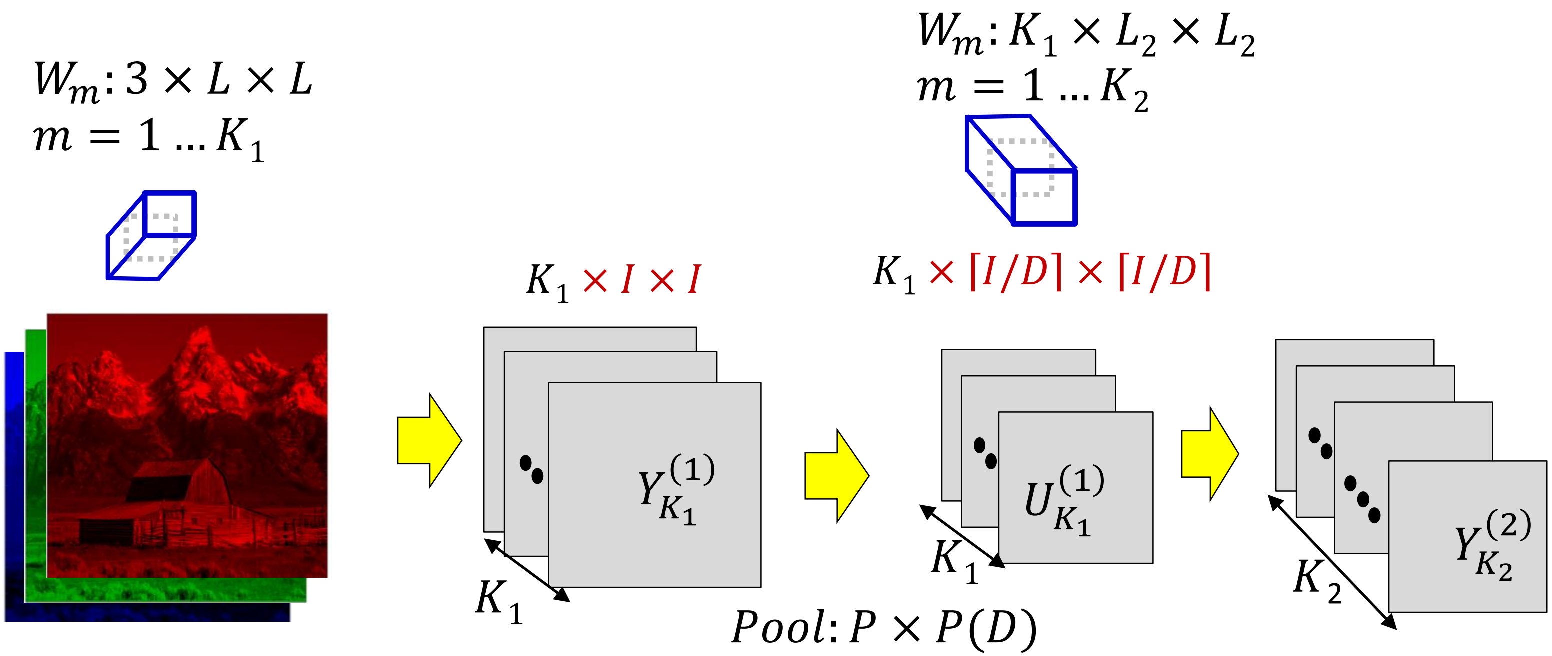
$W_m: 3 \times L \times L$
$m = 1 \ldots K_1$

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

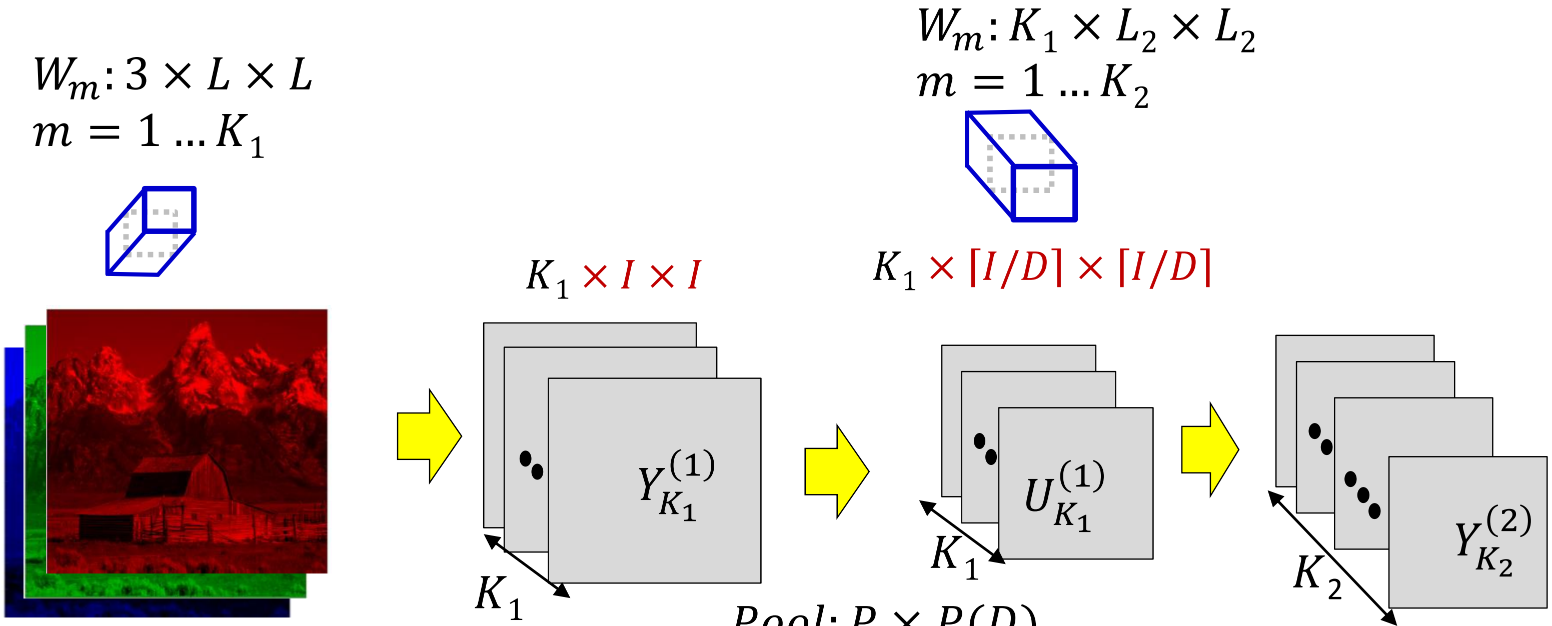


$$z_m^{(n)}(i,j) = \sum_{r=1}^{K_{n-1}} \sum_{k=1}^{L_n} \sum_{l=1}^{L_n} w_m^{(n)}(r,k,l) U_r^{(n-1)}(i+k, j+l) + b_m^{(n)}$$

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

- **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

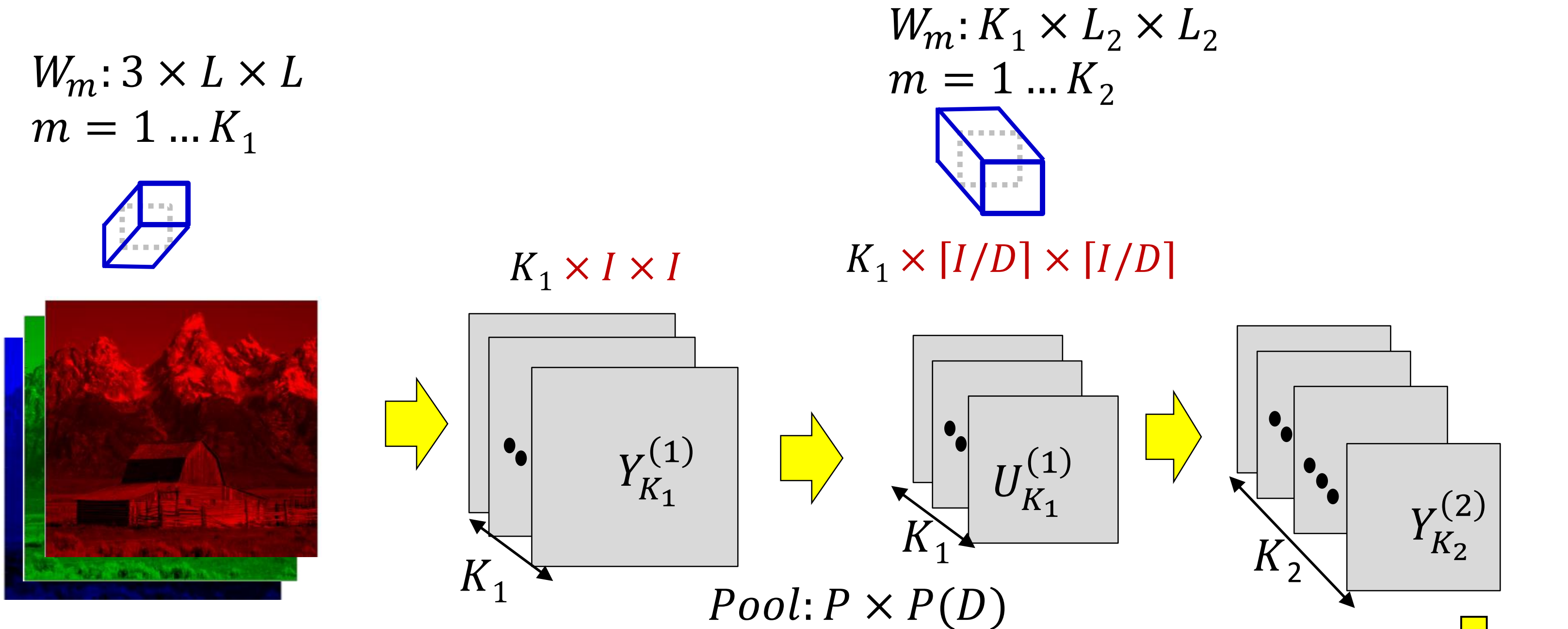
1. Number of filters K_2
2. Size of filters $L_2 \times L_2 \times K_1 + bias$
3. *Stride* of convolution S_2

$_m^{(n)}$

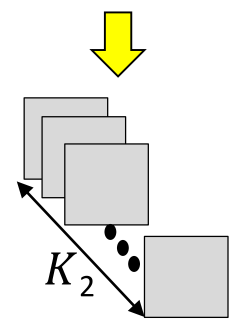
Total number of parameters: $K_2(K_1 L_2^2 + 1)$
All these parameters must be learned

ng in K_2 2-D maps

Convolutional Neural Networks



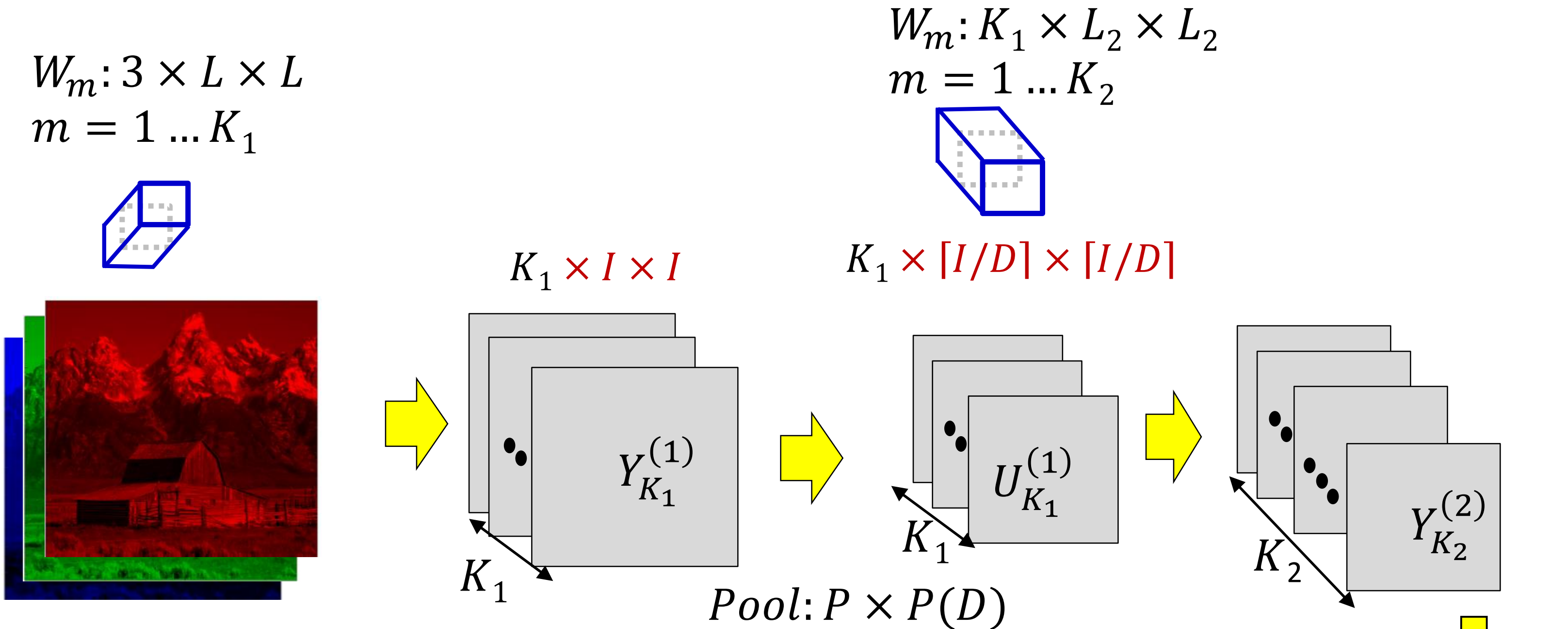
$$P_m^{(n)}(i,j) = \underset{\substack{k \in \{(i-1)d+1,\, id\},\\ l \in \{(j-1)d+1,\, jd\}}}{\operatorname{argmax}} Y_m^{(n)}(k,l)$$



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

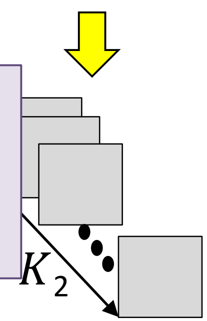
- **Second convolutional layer:** K_2 3-D filters resulting in K_2 2-D maps
- **Second pooling layer**: K_2 Pooling operations: outcome K_2 reduced 2D maps

Convolutional Neural Networks



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

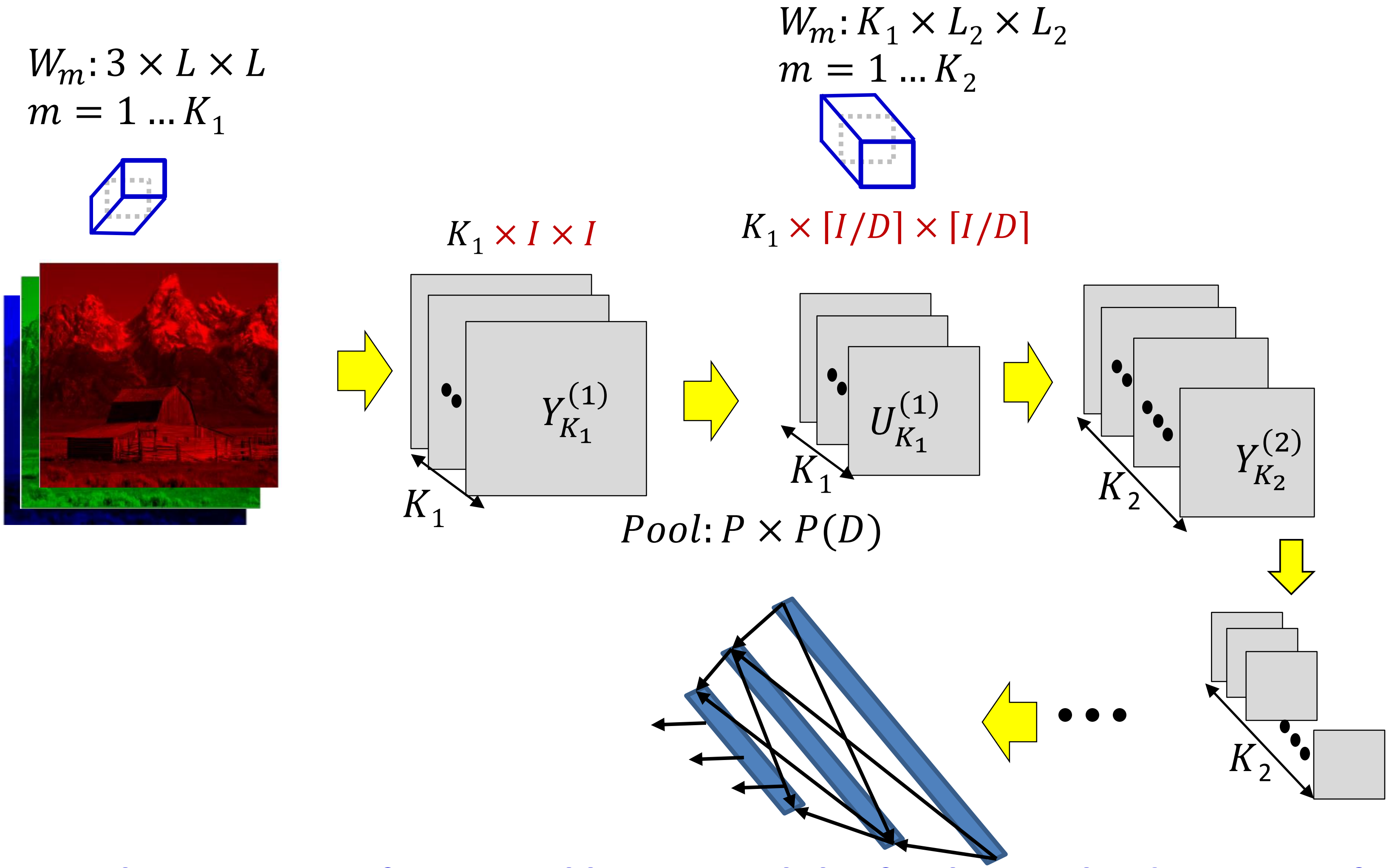
Parameters to choose:
  Size of pooling block P_2
  Pooling stride D_2



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

- **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)

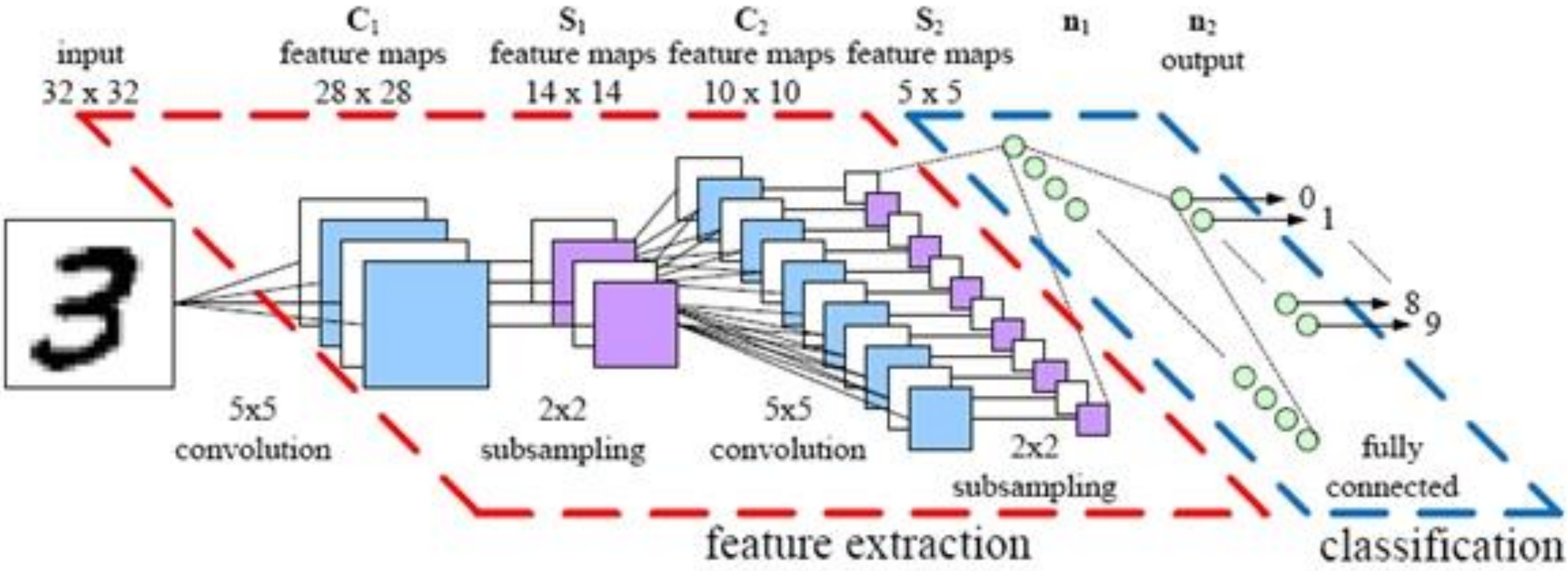
- Number of convolutional and downsampling layers
  - And arrangement (order in which they follow one another)

- 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

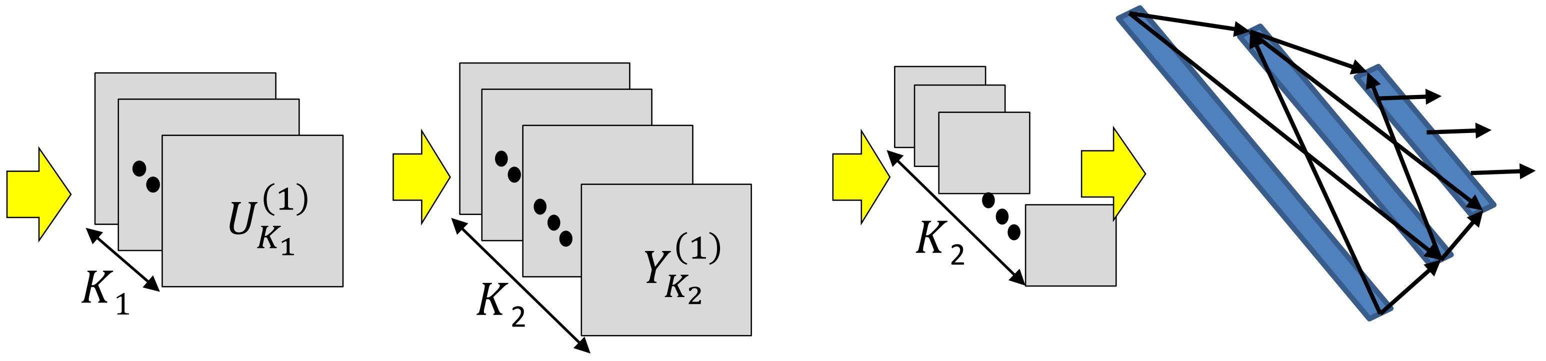
- For each downsampling/pooling layer:
  - Spatial extent of filter $P_i \times P_i$
  - The stride D_i

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

Digit classification

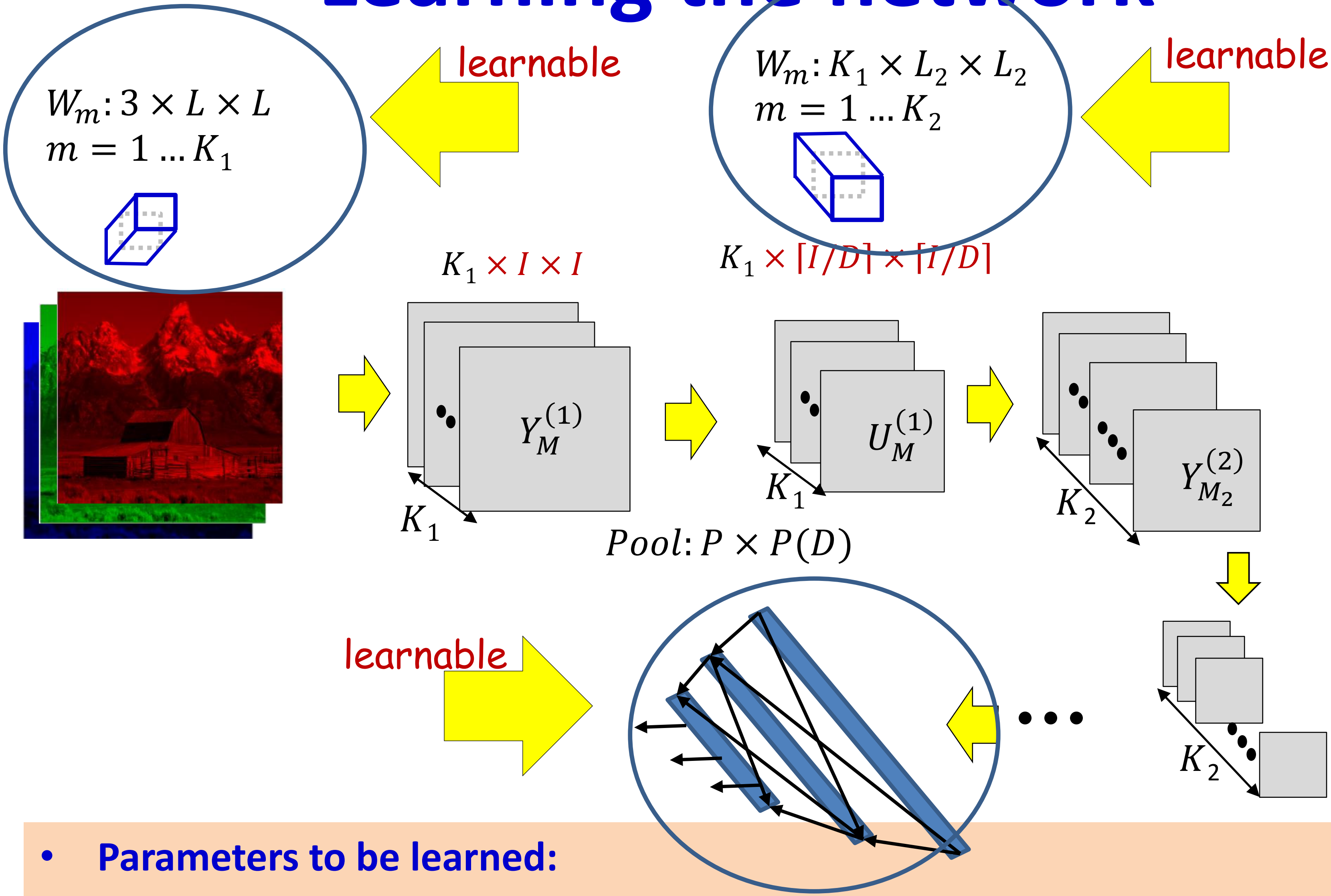


Training



- 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: $\sum_{J \in convolutional\ layers} K_J\left(K_{J-1}L_J^2 + 1\right)$